ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE
C.N.R. – PISA

A MODEL FOR PARALLEL SYSTEMS COMPOSED BY MUTUALLY ASYNCHRONOUS MODULES

G.F. CASAGLIA*, G. CONTI*, M. VANNESCHI*

Internal Report E73-I – Special Series
Convention CNR-ENI
January 1973

*Pignone Sud, at Centro ENI, Pisa

ABSTRACT

A model for digital parallel systems composed by mutually asynchronous modules, each of them with processing speed in general variable, is described, by means of which it is possible to realize parallel and/or pipeline computers in a systematic and correct way. Each module of a system corresponding to the proposed model has its own control part and its own operation part, and the behaviour of the entire system, called PMC system, depends exclusively on the exchange of messages among the control parts of the modules belonging to the system. Moreover the modules exchange each other data by means of queues: data held in queues and messages are the only resources shared by modules.

Three rules of message exchange among modules are defined, obtaining three types of systems called PMC systems with weakly linked modules, with strongly linked modules and with strongly linked modules looking-ahead.

Some ways of transmitting messages are introduced which give rise to several schemes of the control structure of the entire system.

The activity of a PMC system is described by a graph; by some constraints imposed on this graph it is possible to define a class of systems, called PMC linear systems, into which any PMC system can be transformed.

For PMC linear systems, with regard to the rules of message exchange, general conditions are given under which such systems are determinate and deadlock-free.

CONTENTS

1. INTRODUCTION

The development of new hardware technologies and the need of performance enhancement in computer systems, with the purpose of resolving more complex and inherently parallel problems, justify the growing of interest in highly parallel systems organization.

Various machines, [THO 1], [BON 1], [FLY 1], [FLY 2], [MUR 1], [GRA 1], [IBB 1], with very different architectures but all composed by independent, cooperating blocks, have been realized or proposed. The co-operation control, generally simplified by utilizing mutually synchronized blocks or blocks with related processing speeds, seems to be realized by intuitive considerations and no formal methods are given for its design. Besides, the full utilization of the characteristics of a system composed by co-operating blocks is obtained only if the processing speeds of the blocks are not related.

Many theoretical works, formalizing control synthesis of parallel systems composed by mutually asynchronous independent blocks, can be found in the literature.

Some of these, due to Patil [PAT 1] and to Bruno, Altmann and Denning [BRU 1], [AIT 1], introduce a highly structured realization of the control of a digital system; moreover Patil defines a method for obtaining this realization from a formal description of system operations.

No one of these approaches consider the connections between the control part and the operation part of the system.

It follows that:

1) the way of exchanging information among control blocks is imposed by the block definition, therefore, given a specific operation part, the performance of the whole system might be limited;

2) nothing can be said about determinacy and deadlock in the whole system.

In this report we intend to give a model for digital parallel systems composed by mutually asynchronous modules with processing speed in general variable; the co-ordination

of system operations is obtained by means of exchange of messages among modules.

In particular we intend:

a) to define the modules in such a way as their realization comes out enough regular and structured, concerning both their control part and their operation part;

b) to define the rules of message exchange among modules for obtaining a system whose behaviour is determinate and deadlock-free;

c) to define some feasible structures realizing the rules of message exchange with regard to the best performance of the whole system.

At last, this model must be easily applicable to the design of parallel and/or pipeline computers.

In Sect. 2 the model is defined starting from the description of the structure of the generic module; according to three rules of message exchange among modules, three types of systems are introduced, called PMC systems with weakly linked modules, PMC systems with strongly linked modules and PMC systems with strongly linked modules looking-ahead. Finally several control schemes are described specifying for each of them the way of transmitting messages and the management of the control functions.

In Sect. 3, in order to obtain a formal description of the activity of PMC systems, some graphs are defined: the cycle graph describing the activity of any module, the system graph describing the activity of the entire system and the precedence graph defining the precedence relations among the tasks of the system. With this graphical description it is possible to define a class of systems, called PMC linear system, into which any PMC system can be transformed.

In Sect. 4 sufficient conditions are given under which a PMC linear system is determinate.

In Sect. 5 the deadlock problem is discussed referring to consumable resources, like messages and data, which are the only resources of interest in PMC systems. Conditions under which a PMC linear system is deadlock-free are derived.

In Sect. 6 conclusions are given together with suggestions for future works.

2. DESCRIPTION OF THE MODEL

A digital parallel system can be thought of as composed by a number of mutually asynchronous, co-operating modules $\mathcal{M}_i$ (i = 1, ..., n). *Data lines* and *message lines* (both in general multiwired) provide information exchange among modules. Information by which processing of the modules is conditioned are exchanged by means of message lines; the results of such processing are exchanged by means of data lines. Each message line carries one and only one *message*, each data line carries one and only one datum.

Data lines are in general constituted by queues; this allows the speed matching among modules and therefore a more continuous and uniform flow of data (fig. 1). Denote by q_{ij} the queue connecting $\mathcal{M}_i$ with $\mathcal{M}_j$ such that $\mathcal{M}_i$ sends data into q_{ij} and $\mathcal{M}_j$ takes data out from q_{ji}.

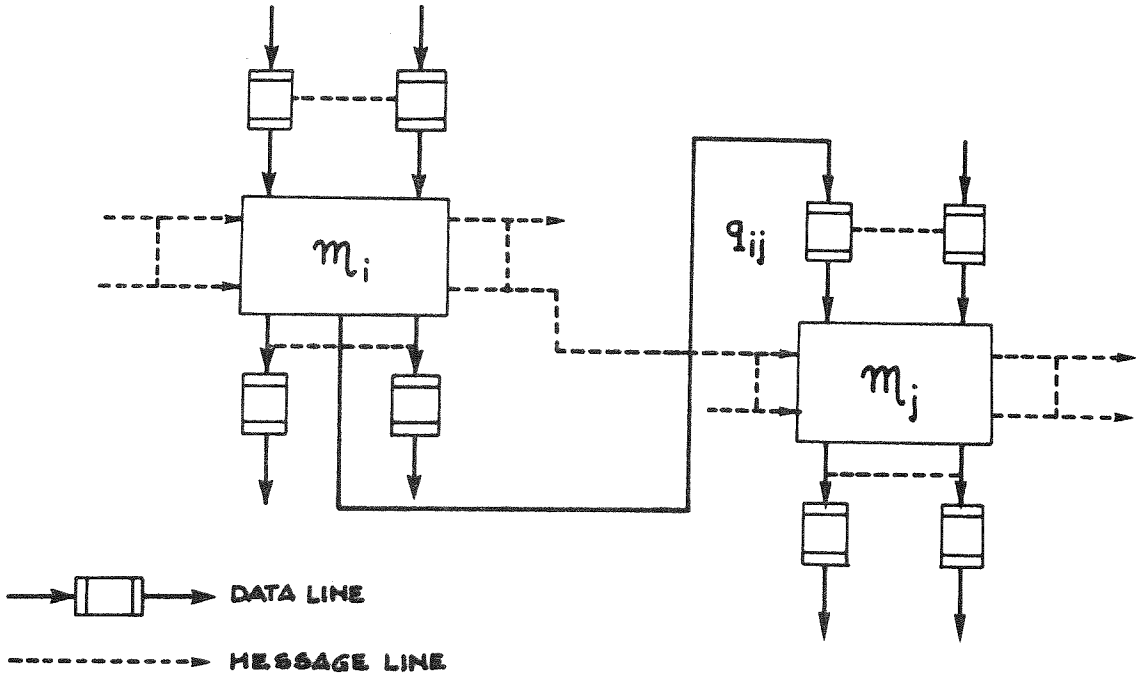


Fig. 1 - Model of digital parallel system

Each module $\mathcal{M}_i$ is composed by two interconnected parts, an *operation part* $O\mathcal{M}_i$ and a *control part* $C\mathcal{M}_i$. $C\mathcal{M}_i$ controls the exchange of information between $\mathcal{M}_i$ and other modules, and the execution and sequence of operations in $O\mathcal{M}_i$. The control parts of the system are connected by message lines and the operation parts are connected by data lines.

The digital system is therefore composed by two separate, interacting structures, *data flow structure* and *control structure* (fig. 2): all the $O\mathcal{M}_i$'s (i = 1, ..., n) belong to the former and all the $C\mathcal{M}_i$'s (i = 1, ..., n) belong to the latter.

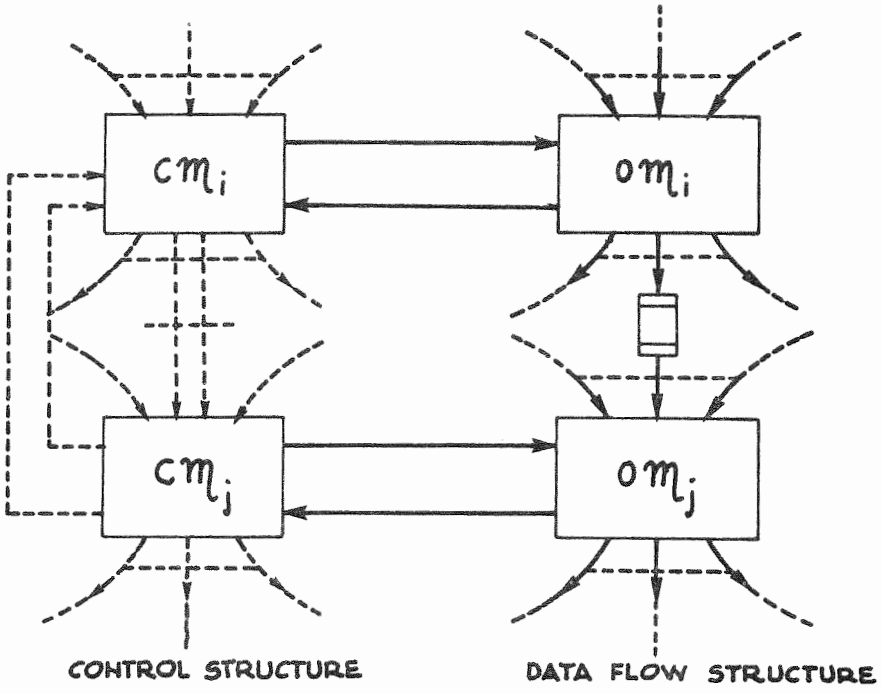


Fig. 2 - Digital parallel system composed by two separate, interacting structures: control structure and data flow structure.

The definition of the model is complete if we define also:

a) the structure of the generic module $\mathcal{M}_i$, including the access mechanism and the queue discipline of the queues constituting the data lines;

b) the rules of message exchange among modules.

A system corresponding to the proposed model will be called Parallel Message Exchange Controlled System, or shortly PMC system.

2.1. Structure of the module $\mathcal{M}_i$.

The operation part $O\mathcal{M}_i$ (fig. 3) is composed by:

a) *input switch* K_i: by this switch one or more input data lines are chosen from which data to be processed are taken out;

b) *functional unit* $\mathcal{F}_i$: by this unit, including logic networks and memory cells, data taken out from input data lines are processed;

c) *output switch* S_i: by this switch output data lines are chosen to which some results of processing are sent.

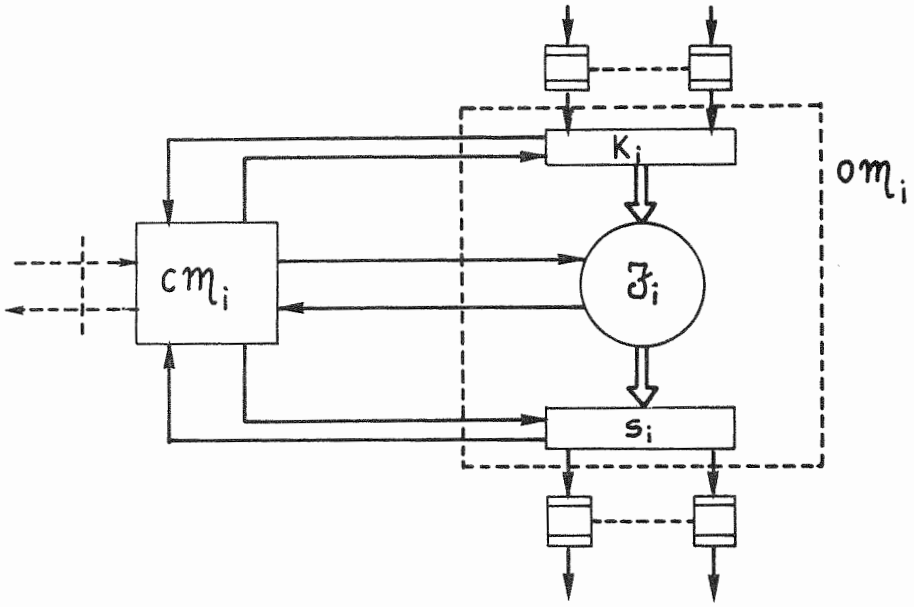


Fig. 3 – Operation part $O\mathcal{M}_i$ of the module $\mathcal{M}_i$.

Queues in data lines are *single-channel, finite length* queues and *First-In-First-Out* is their queue discipline, if not otherwise specified. Each queue allows two simultaneous accesses, one for reading and one for writing. The "*Producer-Consumer*" algorithm [DIJ 1] allows a correct utilization of the data stored in queues. Two semaphores are associated to

each queue q_{ij} : the first one CV_{ij} , a private semaphore of the producer module $\mathcal{M}_i$, counts the queue empty positions; the second one CP_{ij} , a private semaphore of the consumer module $\mathcal{M}_j$, counts the queue full positions. Since semaphores CV_{ij} and CP_{ij} are common resources between modules $\mathcal{M}_i$ and $\mathcal{M}_j$, it is necessary to protect them by means of a mutual exclusion network [BRE 1].

Two distinct purposes are assigned to the control part $C\mathcal{M}_i$:

i) message exchange with other modules;

ii) control of processing in the operation part, depending on messages from other modules, and generation of messages to be sent to other modules.

At least logically these purposes can be thought of as carried out by two distinct parts of $C\mathcal{M}_i$; in the following we will refer to these parts as *message exchange control* and *processing control* respectively.

In par. 2.3 we will consider some cases in which the message exchange control is included in the processing control and other cases in which the first one is physically distinct and independent from the second one.

Processing control is divided into five distinct *sub-controls* (fig. 4), each of which is a sequential machine.

Acting on K_i and S_i , CK_i and CS_i determine respectively the choice of input queues and output queues. These choices are made on the basis of messages, coming from other modules, and/or conditions belonging to the processing control. $C\mathcal{F}_i$ controls the execution and the sequence of operations in $\mathcal{F}_i$. CQ_{IN_i} and CQ_{OUT_i} manage respectively all the input queues and all the output queues of the module $\mathcal{M}_i$.

The following sequence of steps is defined *cycle* Γ_i of the processing control of $C\mathcal{M}_i$:

1) CK_i (initially active) selects input queues $q_{t_m i}$ ($t_m \neq i$, $m = 1, ..., z$) and then activates CQ_{IN_i} ;

2) CQ_{IN_i} executes Dijkstra's $\mathcal{P}$-primitive on $CP_{t_m i}$ and then activates CK_i ;

3) CK_i causes data transfers from selected input queues to $\mathcal{F}_i$, executes Dijkstra's $\mathcal{V}$-primitive on $CV_{t_m i}$ and then activates $C\mathcal{F}_i$;

4) $C\mathcal{F}_i$ controls processing of data taken out from input queues and then activates CS_i ;

5) CS_i selects output queues q_{if_n} ($f_n \neq i$, $n = 1, ..., v$) and then activates CQ_{OUT_i} ;

6) CQ_{OUT_i} executes $\mathscr{P}$-primitive on CV_{if_n} and then activates CS_i;

7) CS_i causes data transfers from $\mathscr{F}_i$ into selected output queues, executes $\mathscr{V}$-primitive on CP_{if_n} and then activates CK_i. Go to 1.

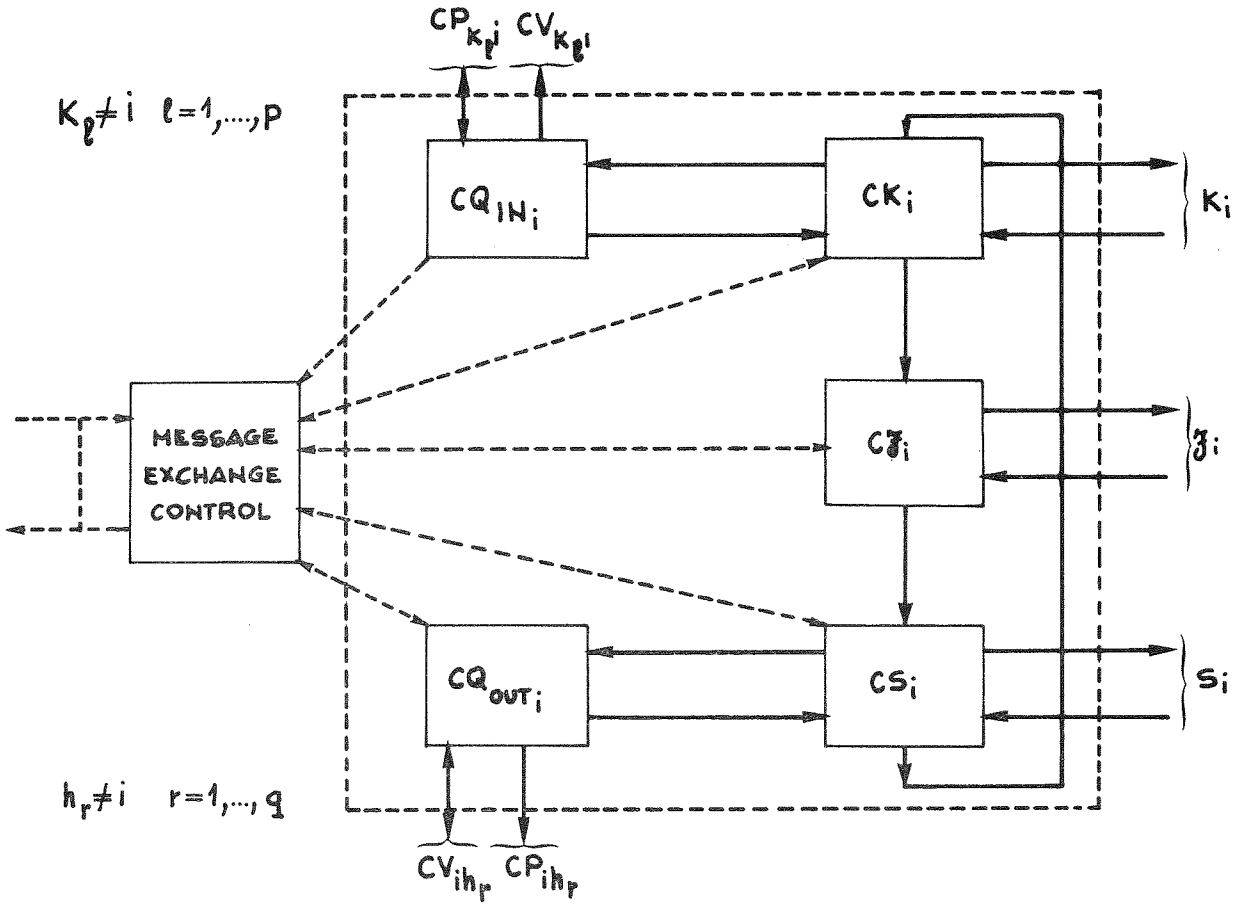


Fig. 4 – Control part $C\mathcal{M}_i$ of the module $\mathcal{M}_i$.

The time taken for the execution of each cycle Γ_i is in general variable.

As it results from the definition of cycle, one and only one sub-control is active at a time. Each sub-control, when it is active, can send to and/or receive messages from other modules via the message exchange control.

Some examples can now be useful to verify the possibility of describing, by the proposed model, a large class of digital systems. The CPU of a digital system can be described

by a module $\mathfrak{M}_i$ when we want to study the connections between CPU and external world; a digital system can be described by three modules, appropriately interconnected, if we consider memory, instruction preparation unit and execution unit as three independent co-operating blocks. At last, each stage of a pipeline system can be described by a module, as well as each Element of a multiprocessor system.

2.2. Rules of message exchange among modules.

Let $\mathfrak{M}_i$ be a module receiving messages from a set of other modules $\{\mathfrak{M}_{j_l}\}$, $j_l \neq i$, $l = 1, ..., p$.

Suppose that $\mathfrak{M}_{j_l}$ sends a message to $\mathfrak{M}_i$ on the relative message line; after utilizing the message, $\mathfrak{M}_i$ deletes it from the message line.

Let C_i be a sub-control of $\mathfrak{M}_i$. C_i can receive, in a certain state γ_k, a known set of messages $\{M_i\}_{\gamma_k}$ from a subset of modules belonging to $\{\mathfrak{M}_{j_l}\}$.

Three rules of message exchange are considered:

1) C_i examines, in the state γ_k, the message lines relative to the set $\{M_i\}_{\gamma_k}$ and starts immediately the processing, behaving, if some messages of $\{M_i\}_{\gamma_k}$ are not present, as if the sending modules have no messages to send it. C_i stops, waiting for a message, only when this is indispensable in a certain state. In the following we will refer to a *PMC system with weakly linked modules* to denote a PMC system with modules exchanging messages according to this rule.

For example, in a pipeline computer with a look-ahead instruction fetching unit (LFU), the instruction sequence may be interrupted by a jump decoded by the instruction decoding unit (IDU). The IDU sends a message to the LFU on message line J only if it decodes a jump instruction; before fetching a new instruction, LFU examines message line J and, if the relative message is not present, it fetches the next instruction in sequence.

Since, in general, there is no relation among the processing speeds of the modules, a message may arrive to a module after processing is started by sub-control C_i, and this message may specify that a wrong processing was started by C_i. Referring to the example above, this drawback can be avoided when it is possible to delete the effects of the wrong sequence of instructions and to restart processing from the last right operation.

2) Being in state γ_k , C_i starts the processing only when it receives all the messages belonging to $\{M_i\}_{\gamma_k}$. This implies that the time interval between sending two successive messages must be finite.

Further information, specifying *the presence or the absence* of messages, must be associated to each message line, the presence being tagged by sending modules and the absence by receiving modules.

If the number of cycles Γ_i between sending two successive messages is fixed, the sending module must be able to send also a particular type of message whose meaning is *nothing to communicate*, shortly called n.t.c. message.

In the following we will refer to a *PMC system with strongly linked modules* to denote a PMC system with modules exchanging messages according to this rule.

In the example above, relating to a pipeline computer, if the IDU is requested to send a message to the LFU for each decoded instruction, this message would be (e.g.) zero if the instruction just decoded is not a jump and one if yes.

3) Being in state γ_k, C_i starts immediately the processing assuming the absent messages belonging to $\{M_i\}_{\gamma_k}$ as n.t.c. messages; during processing all the possible partial results are stored in auxiliary memory cells inside the module $\mathfrak{M}_i$; processing goes on, eventually by the activation of the sub-controls associated with the successive steps of Γ_i , until C_i reaches a state in which it is necessary to send results into output queues and messages to other modules. All the messages not arrived in state γ_k are waited for in this state, referred to as *check state*: if these messages are messages of the type n.t.c., processing is continued by (i) transferring the partial results from auxiliary memory cells to the actual memory cells or sending them into output queues, (ii) sending messages to other modules, (iii) starting next step of Γ_i ; otherwise processing is restarted from state γ_k .

In the following we will refer to a *PMC system with strongly linked modules looking-ahead* to denote a PMC system with modules exchanging messages according to this rule.

An improvement in processing speed can be obtained, in respect of a PMC system with strongly linked modules, if time interval between sending two successive messages is fixed and the n.t.c. messages occur with high probability.

2.3. Message exchange control.

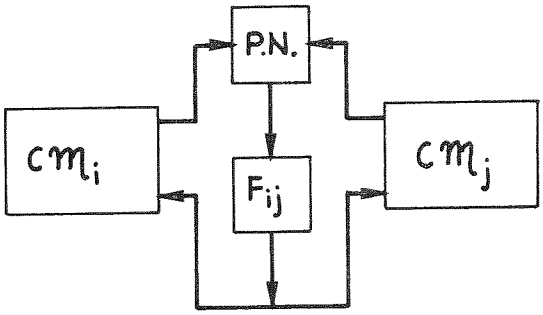
In par. 2.1. the structure of processing control of generic module $\mathcal{M}_i$ was examined and it was premised that message exchange control may be either distributed in each of sub-controls C_i or physically distinct from them and independent. It follows that two *types of management* of the $C\mathcal{M}_i$ functions are possible: the first, with single control, is called *Sequential Management*; the second, with two or more controls, is called *Parallel Management.*

2.3.1. Ways of transmitting messages.

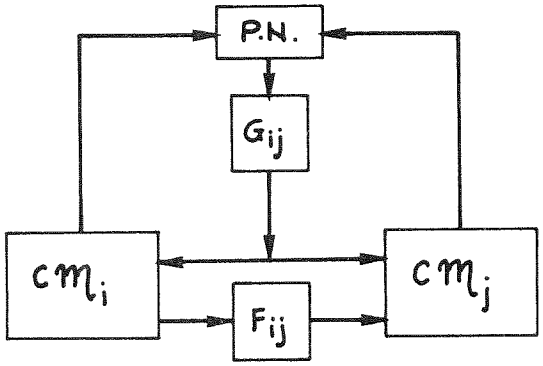
Let us consider two generic controls $C\mathcal{M}_i$ and $C\mathcal{M}_j$, which exchange messages with each other; let $C\mathcal{M}_i$ be the sending control and $C\mathcal{M}_j$ be the receiving control.

Three different ways of transmitting messages, called Interlocking, Queueing of messages and Interrupting are possible.

1) *Interlocking*: $C\mathcal{M}_i$ sends the message on condition that $C\mathcal{M}_j$ has utilized the message previously sent by $C\mathcal{M}_i$, otherwise $C\mathcal{M}_i$ waits until this occurs. In fig. 5-a the case of PMC systems with weakly linked modules is shown; the message line is composed by a register F_{ij} whose bit-length is sufficient to store a message.



a) The case of PMC systems with weakly linked modules.



b) The case of PMC systems with strongly linked modules, eventually looking-ahead.

Fig. 5 – Interlocking

When $C\mathcal{M}_i$ is in a state in which it may have messages to send to $C\mathcal{M}_j$, its behaviour is the following ($F_{ij} = 0$ defines the situation: "no message on the line"):

$\vdots$

test$_i$: **if** $C\mathcal{M}_i$ has no message to send **then goto** rem;
if $F_{ij} \neq 0$ **then** $C\mathcal{M}_i$ waits **else** $C\mathcal{M}_i$ sends the message into F_{ij} ;

rem : remainder of Γ_i

$\vdots$

In its turn $C\mathcal{M}_j$, when in a state in which it may receive messages from $C\mathcal{M}_i$, behaves as follows:

$\vdots$

read : $C\mathcal{M}_j$ reads F_{ij} ;
$F_{ij} := 0$;
remainder of Γ_j ;

$\vdots$

In fig. 5-a P.N. is a priority network which, in the case of simultaneous accesses to F_{ij}, grants the execution of the operation $F_{ij} := 0$ before $C\mathcal{M}_i$ sends a message into F_{ij}.

In fig. 5-b the case of PMC systems with strongly linked modules, eventually looking-ahead, is shown. G_{ij} is a tag indicating the presence ($G_{ij} = 1$) or the absence ($G_{ij} = 0$) of a message on the line.

When $C\mathcal{M}_i$ must send a message to $C\mathcal{M}_j$, its behaviour is the following:

$\vdots$

test$_i$: **if** $G_{ij} = 1$ **then** $C\mathcal{M}_i$ waits **else** $C\mathcal{M}_i$ sends the message into F_{ij} ;
$G_{ij} := 1$;
remainder of Γ_i ;

$\vdots$

The behaviour of $C\mathcal{M}_j$, when it is in a state in which it must receive a message from $C\mathcal{M}_i$, is the following:

⋮

test$_j$: if $G_{ij} = 0$ **then** $C\mathcal{M}_j$ waits **else** $C\mathcal{M}_j$ takes the message out from F_{ij} ;
$G_{ij} := 0$;
remainder of Γ_j ;

⋮

The network P.N. grants operation $G_{ij} := 0$ always preceeds operation $G_{ij} := 1$.

Using Interlocking as way of transmitting messages, the cases in which $C\mathcal{M}_i$ waits can be reduced by *buffering*, inside $C\mathcal{M}_i$ itself, the messages to be sent. In this case $C\mathcal{M}_i$ puts always the message into a First-In-First-Out buffer and, if possible, sends to $C\mathcal{M}_j$ the message at the head of the buffer.

2) *Queueing of messages*: $C\mathcal{M}_i$ sends the messages into a queue realizing the message line; $C\mathcal{M}_j$ takes messages out from this queue with First-In-First-Out discipline. Synchronization of queue is accomplished by the "Producer-Consumer" technique described in par. 2.1: when $C\mathcal{M}_i$ wants to send a message to $C\mathcal{M}_j$ it is blocked only if the queue is full. Shown in fig. 6 is the way of transmitting messages by Queueing in PMC systems whatever modules are linked.

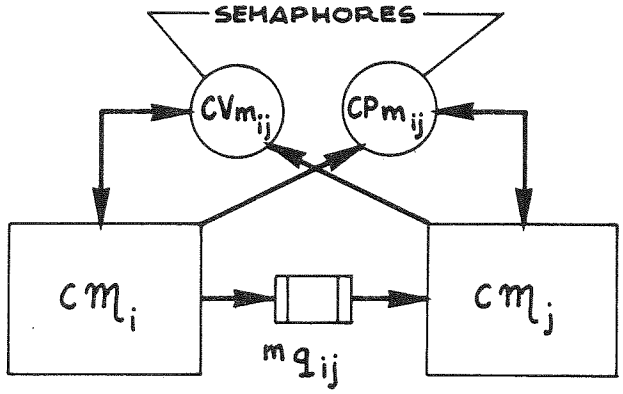


Fig. 6 – Queueing of messages.

Consider a PMC system with weakly linked modules. When $C\mathcal{M}_i$ is in a state in which it may have a message to send to $C\mathcal{M}_j$, its behaviour is the following:

$\vdots$

$test_i$: **if** $C\mathcal{M}_i$ has no message to send **then goto** rem ;
$\mathcal{P}(CVm_{ij})$;
$C\mathcal{M}_i$ sends the message into mq_{ij} ;
$\mathcal{V}(CPm_{ij})$;

rem : remainder of Γ_i ;

$\vdots$

In its turn $C\mathcal{M}_j$, when it is in a state in which it may receive a message from $C\mathcal{M}_i$, behaves as follows

$\vdots$

$test_j$: **if** $CPm_{ij} = 0$ **then goto** rem;
$CPm_{ij} := CPm_{ij} - 1$;
$C\mathcal{M}_j$ takes the message out from mq_{ij} ;
$\mathcal{V}(CVm_{ij})$;

rem : remainder of Γ_j ;

$\vdots$

Consider now a PMC system with strongly linked modules, eventually looking-ahead; when $C\mathcal{M}_i$ must send a message to $C\mathcal{M}_j$, its behaviour is the following:

$\vdots$

$test_i$: $\mathcal{P}(CVm_{ij})$;
$C\mathcal{M}_i$ sends the message into mq_{ij} ;
$\mathcal{V}(CPm_{ij})$;
remainder of Γ_i ;

$\vdots$

Finally when $C\mathcal{M}_j$ is in a state in which it must receive a message from $C\mathcal{M}_i$, it behaves as follows:

$$
\begin{array}{ll}
 & \vdots \\
\text{test}_j : & \mathcal{P}(CPm_{ij}) ; \\
 & C\mathcal{M}_j \text{ takes the message out from } mq_{ij} ; \\
 & \mathcal{V}(CVm_{ij}) ; \\
 & \text{remainder of } \Gamma_j ; \\
 & \vdots
\end{array}
$$

In this way of transmitting messages the queue realizing the message line allows to adapt the speed, with which $C\mathcal{M}_i$ produces messages, to that with which $C\mathcal{M}_j$ consumes them; notice that, in the case of Interlocking with buffering, the buffer does not realize a similar function, because $C\mathcal{M}_j$ still receives messages at time intervals depending on length of cycle Γ_i of $C\mathcal{M}_i$.

3) *Interrupting*: $C\mathcal{M}_i$ sends to $C\mathcal{M}_j$ an interrupt request and transmits the message only after receiving by $C\mathcal{M}_j$ the interrupt acknowledgement.

Shown in fig. 7 is the way of transmitting messages by Interrupting in PMC systems, whatever modules are linked; two wires are associated with each message line: the one for the interrupt request and the other for the interrupt acknowledgement.

The way of transmitting messages by Interrupting is useful when time lost for interrupt management is negligible in comparison with the processing time; when interrupt

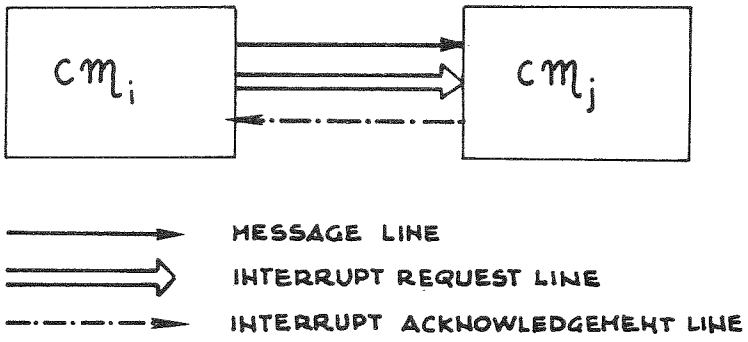


Fig. 7 – Interrupting

management time and processing time are of the same order of magnitude, Interrupting is not convenient for PMC systems with sequential management of control functions, while it may be still valid if used for PMC systems with parallel management of control functions.

In order to avoid blocking of $C\mathcal{M}_i$, it is necessary that the algorithm of interrupt management permits the acknowledgements to interrupt requests from other modules while $C\mathcal{M}_j$ is waiting for the acknowledgement to an interrupt request.

2.3.2. Control with parallel management.

If the control $C\mathcal{M}_i$ has a parallel management, the functions of message exchange control and processing control are executed by two distinct and parallel operating controls denoted by MEC_i and PC_i respectively.

MEC_i provides for:

a) sending messages to other modules, owing to PC_i requests;

b) receiving messages from other modules;

c) preparing messages received from other modules and sending these messages to PC_i either when PC_i requests them or according to a prefixed priority, independently of requests from PC_i.

In fig. 8 a generic control $C\mathcal{M}_i$ with parallel management is shown.

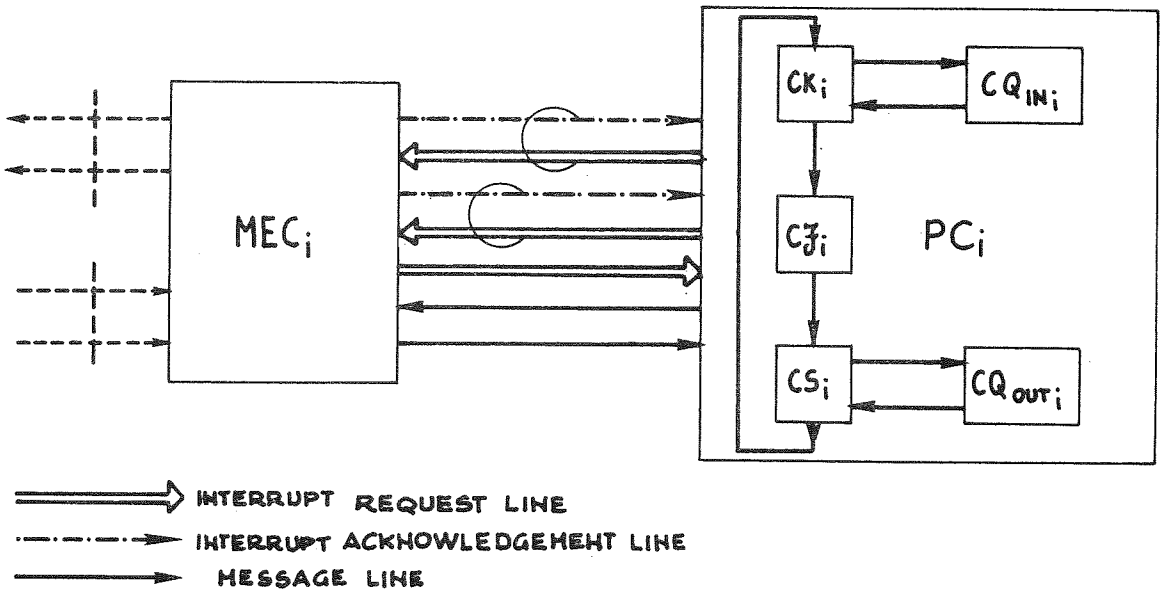


Fig. 8 – Control $C\mathcal{M}_i$ with parallel management.

Message exchange between MEC_i and PC_i is done by interrupt. Message exchanges with other modules may be done according to one of the ways of transmitting messages defined in par. 2.3.1.

Operations a), b), c) are executed sequentially and in fixed order by MEC_i; because of this, if message exchanges among modules are very frequent, it may be convenient to divide MEC_i in two distinct, parallel operating parts: the first, denoted by MEC_i^O, manages transmission of messages to other modules owing to PC_i requests; the second, denoted by MEC_i^I, receives messages from other modules and sends them, conveniently assembled, to PC_i (fig. 9).

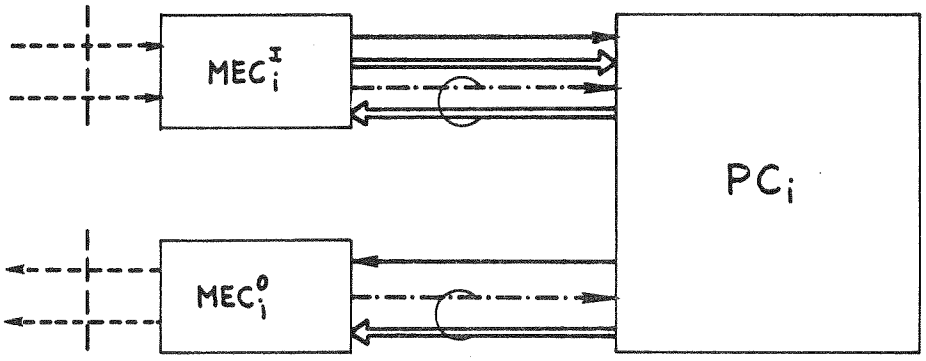


Fig. 9 – Control $C\mathcal{M}_i$ with parallel management and MEC_i divided in two parts.

2.4. Control schemes.

We define *control scheme* of a PMC system the network of controls of the system (i.e. the control structure), in which the type of management and the way of transmitting messages are specified.

As an example, in fig. 10 and in fig. 11 a control scheme with sequential management and transmission of messages by Queueing and a control scheme with parallel management and transmission of messages by Interrupting, respectively, are shown.

In the scheme shown in fig. 11 interrupts are acknowledged by MEC_i (not divided in two parts), with fixed priority. The algorithm of interrupt management is shown by a flow chart in fig. 12, where W is a queue initially empty containing the identifiers of modules which did not yet acknowledge the interrupt requests from MEC_i.

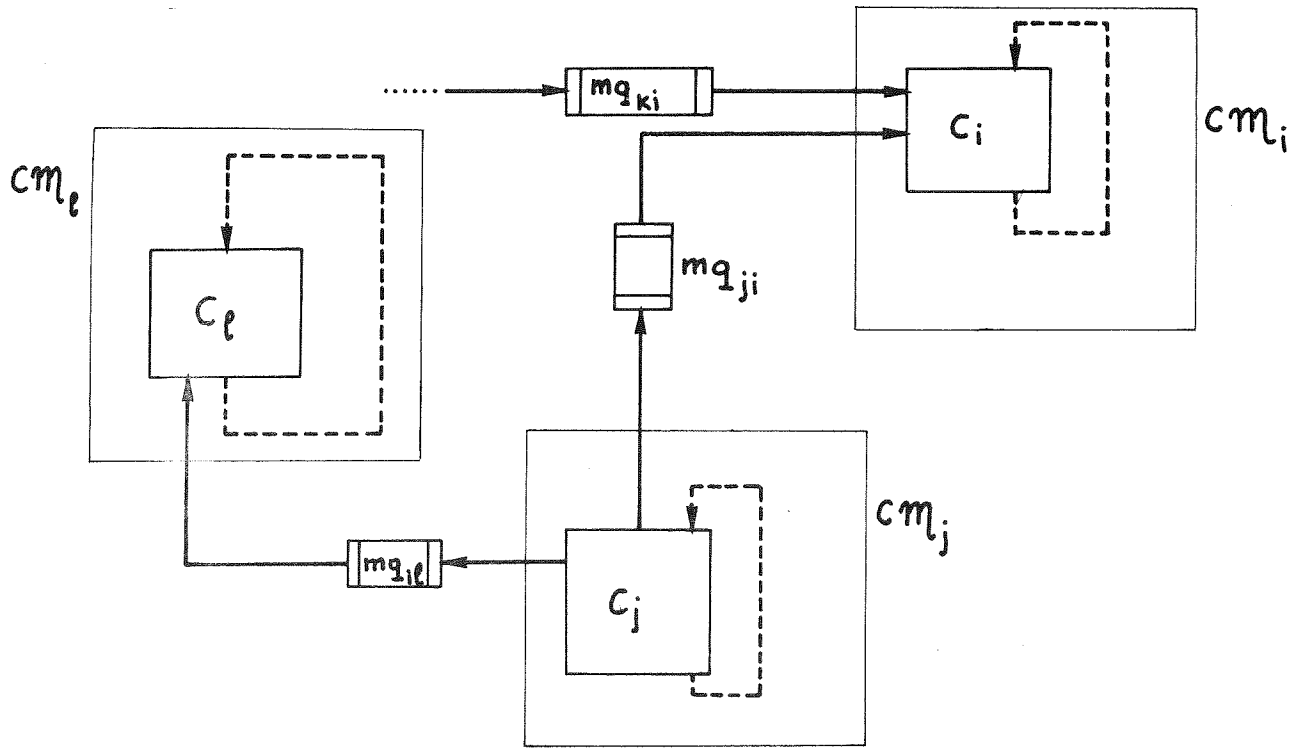


Fig. 10 – Control scheme with Sequential Management and with transmission of messages by Queueing.

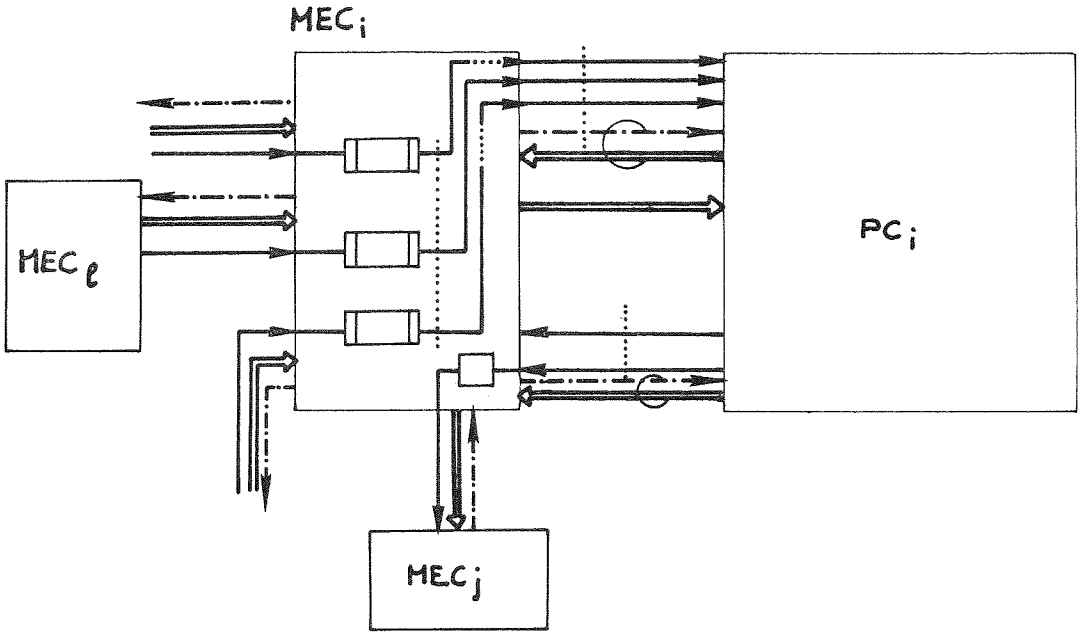
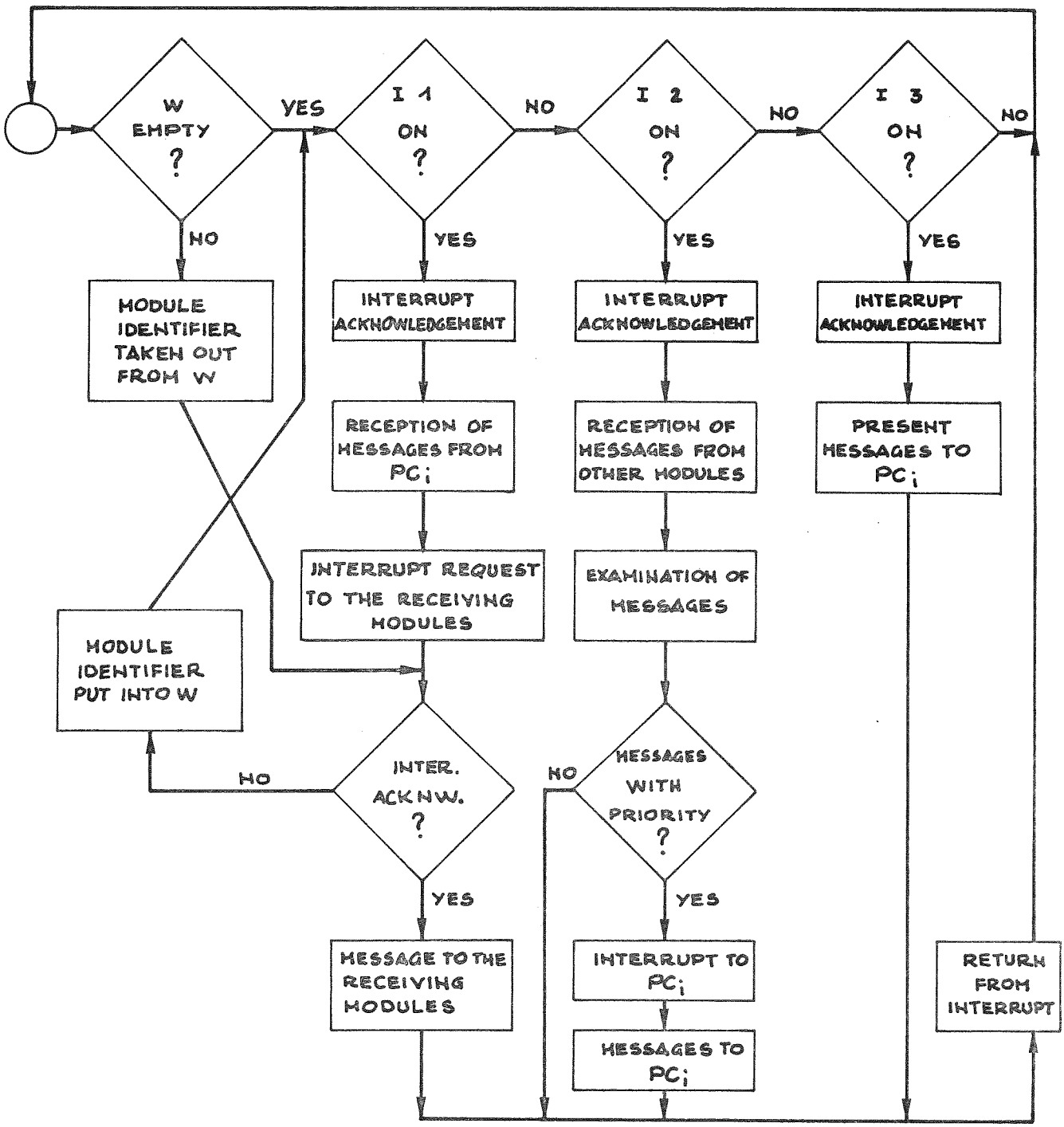


Fig. 11 – Control scheme with Parallel Management and transmission of messages by Interrupting.



I1 on : Interrupt request from PC_i for sending messages;
I2 on : Interrupt request from other modules;
I3 on : Interrupt request from PC_i for receiving messages.

Fig. 12 – Algorithm of interrupt management in MEC_i.

Notice that this algorithm avoids blocking of two (or more) modules waiting for receiving each other the acknowledgement to an interrupt request.

Blocking cannot ever occur in control schemes with parallel management and MEC_i divided in MEC_i^I and MEC_i^O.

3. GRAPHICAL DESCRIPTION OF THE ACTIVITY OF A PMC SYSTEM

3.1. Cycle graph and system graph.

In Sect. 2 we saw that the activity of control $C\mathcal{M}_i$ of generic module $\mathcal{M}_i$ is described by a cycle Γ_i and that, in some points of Γ_i, $C\mathcal{M}_i$ exchanges messages with other modules.

Inside Γ_i one may individuate a number of *tasks* defined as follows: a generic task $T_{ij} \in \Gamma_i$ is a three-tuple

$$T_{ij} = (SI_{ij}, t_{ij}, SF_{ij})$$

where t_{ij} is a *segment of cycle* and SI_{ij} and SF_{ij} are the states which delimit t_{ij}. SI_{ij}, called *initial state of* T_{ij}, is the only state of T_{ij} in which $C\mathcal{M}_i$ can receive a set $\{M^I_{ij}\}$ of messages sent from other modules; we shall say that $\{M^I_{ij}\}$ *belongs to* SI_{ij}. SF_{ij}, called *final state of* T_{ij}, is the only state of T_{ij} in which $C\mathcal{M}_i$ can send a set $\{M^O_{ij}\}$ of messages to other modules; we shall say that $\{M^O_{ij}\}$ *belongs to* SF_{ij}. $\{M^I_{ij}\}$ and $\{M^O_{ij}\}$ may eventually be empty sets.

A task T_{ij} is said to be *enabled* when $C\mathcal{M}_i$ is in state SI_{ij}. In such a state $C\mathcal{M}_i$, according to the messages, belonging to $\{M^I_{ij}\}$, arrived to it, gives rise to *initiation* of T_{ij}. In a PMC system with strongly linked modules, initiation of T_{ij} cannot occur before the arrival of all messages belonging to $\{M^I_{ij}\}$. In all cases the time interval between enabling and initiation of T_{ij} (if deadlock due to messages does not exist: see Sect. 5) is finite.

When $C\mathcal{M}_i$ is in state SF_{ij}, we say that T_{ij} *terminates*. The time interval between initiation and termination is finite (if deadlock due to data does not exist: see Sect. 5). In state SF_{ij} $C\mathcal{M}_i$ sends the set $\{M^O_{ij}\}$; at the same time the transition from final state SF_{ij} of T_{ij} to initial state SI_{ik_l} of task T_{ik_l} ($l = 1,..., s$) occurs, being T_{ik_l} one of the possible successor tasks of T_{ij}: in its turn T_{ik_l} becomes enabled.

We suppose that, at the beginning of its activity, $C\mathcal{M}_i$ is in a particular state, called

initial state of cycle Γ_i, $\mathscr{T}\Gamma_i \in SI_i$, where $SI_i = \{SI_{ij} \mid \forall j\}$, and that this state is always the same at the beginning of each activity of $C\mathfrak{M}_i$. The task, whose initial state coincides with $\mathscr{T}\Gamma_i$, will be said to be the *initial task* of Γ_i.

We suppose also that cycle Γ_i has one and only one *final task*, whose final state is called *final state of cycle* Γ_i and is denoted by $\mathscr{F}\Gamma_i \in SF_i$, where $SF_i = \{SF_{ij} \mid \forall j\}$. The successor state of $\mathscr{F}\Gamma_i$ is always $\mathscr{T}\Gamma_i$.

Cycle Γ_i may be described by a graph, called *cycle graph*, whose nodes correspond to tasks; an arc, denoted by a^i_{jk}, connects node T_{ij} with node T_{ik} and it is directed from T_{ij} towards T_{ik} if and only if T_{ik} is one of the possible successor tasks of T_{ij}. An arc like a^i_{jk} will be generically called *a–arc*. The particular a-arc which connects the final task with the initial task of the cycle is called *return arc.*

Since the cycle graph contains only one initial node and only one final node, corresponding to initial task and final task respectively, it is always possible to transform any cycle graph in another one, called *linear cycle graph*, whose nodes, included between initial node and final node, are linearly ordered. The last description in general imposes a constraint to the activity of the module, since the reduction of the number of nodes restricts the number of points in which the module can send to or receive messages from other modules.

A PMC system, the activity of all modules of which is described by a linear cycle graph, is called *PMC LINEAR SYSTEM.*

In fig. 13 a linear cycle graph is shown: the initial task is marked with*.

The graph describing the activity of the entire system is called *system graph* and it is obtained by connecting, with a new set of arcs, the cycle graphs of all modules of the system; these arcs, called *message arcs*, are so defined: a message arc, labeled $m_{ij,rp}$, connects node T_{ij} with node T_{rp} and it is directed from T_{ij} towards T_{rp} if and only if $C\mathfrak{M}_i$, being in state SF_{ij}, sends to $C\mathfrak{M}_r$, $r \neq i$, a message $m'_{ij,rp} \in \{M^o_{ij}\}$ and $C\mathfrak{M}_r$, being in state SI_{rp}, expects to receive from $C\mathfrak{M}_i$ a message $m''_{ij,rp} \in \{M^I_{rp}\}$ and it is: $m'_{ij,rp} \equiv m''_{ij,rp} \equiv m_{ij,rp}$.

Message arcs are drawn as dashed edges. The system graph of a PMC linear system is shown in fig. 14.

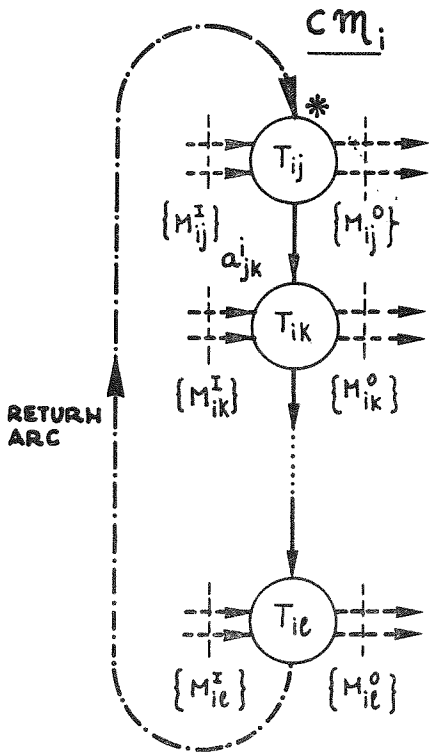


Fig. 13 – A linear cycle graph.

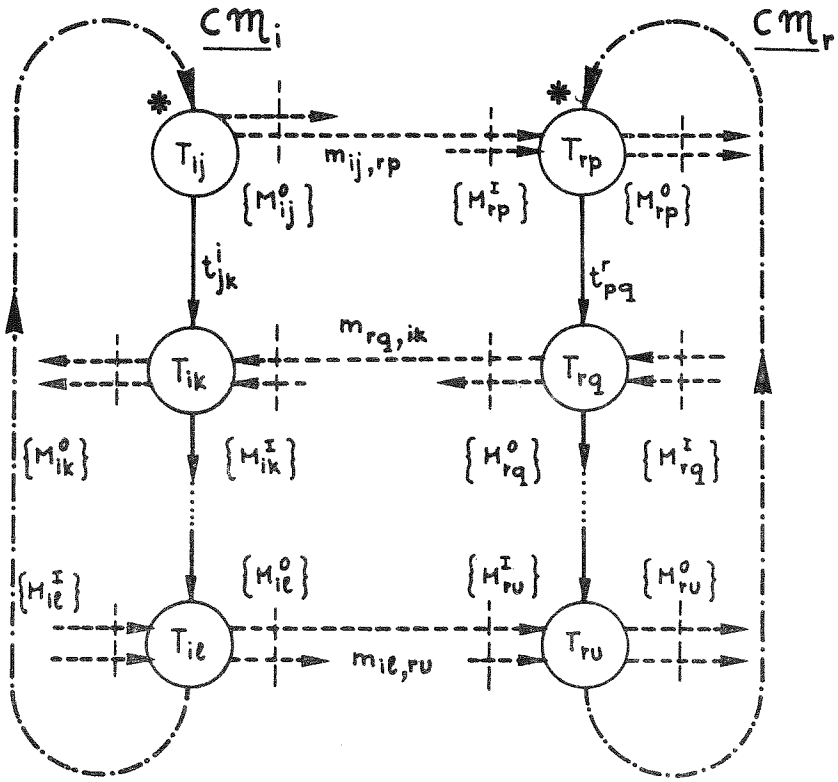


Fig. 14 – System graph.

3.2. Precedence graph.

System graph is the description of a collection of co-operating tasks, the co-operation being realized by message exchange.

A collection of co-operating tasks is what is called a *system of tasks* [DEN 1] when it contains a specification of precedence constraints among tasks constituting it; such precedence constraints may define a *partial ordering* of the tasks of the system. The partial ordering may be described by means of a graph, called *precedence graph,* in which:

i) nodes correspond to tasks;

ii) an arc of this graph, called *precedence arc* and denoted by π_{ab}, connects node T_a with node T_b and it is directed from T_a towards T_b if and only if task T_a precedes task T_b (in symbols: $T_a < T_b$) in the partial ordering of the system of tasks, and there

exists no task T_c such that $T_a < T_c < T_b$.

The precedence graph of a PMC linear system is derived from the system graph by applying the following Conversion Rule:

1) Nodes of precedence graph are all the nodes of system graph and only those.

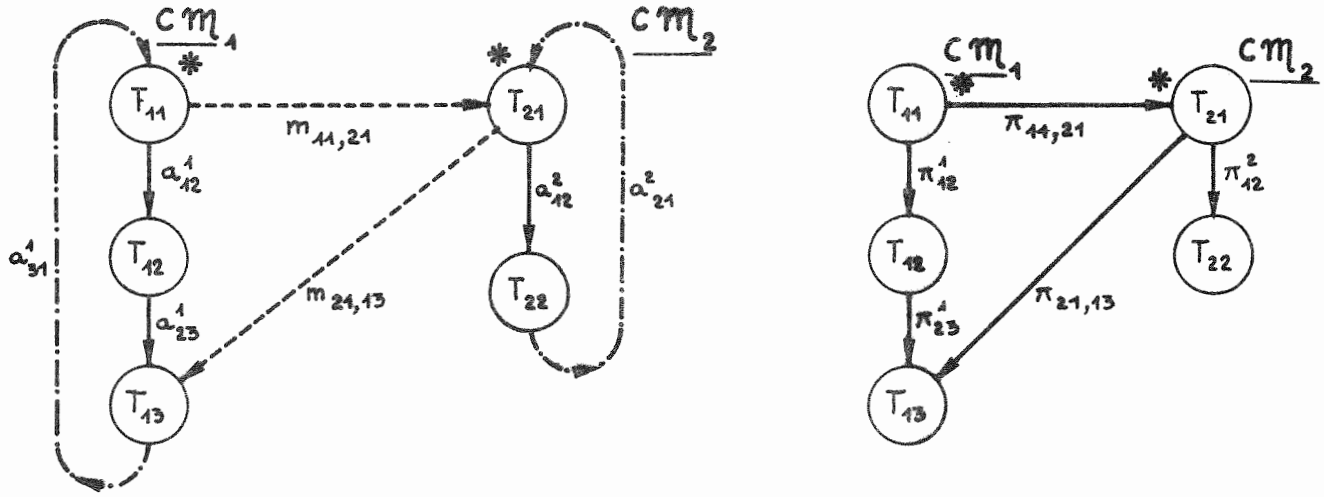
2) Precedence arcs are:

   a) all the message arcs of system graph;

   b) all the a-arcs of system graph, except return arcs.

3) Initial tasks of precedence graph (still marked with*) are all those of system graph and only those.

For example, consider the PMC linear system described by the system graph of fig. 15-a; the corresponding precedence graph is that of fig. 15-b.



a) system graph

b) precedence graph

Fig. 15 – Example of a linear system.

Let the symbols $\overline{T_a}$ and $\underline{T_a}$ denote the events initiation and termination of task T_a respectively.

Given a system of tasks $\{T_i \mid i = 1, ..., n\}$ defining a PMC linear system, we shall call *execution* & any string of initiations and terminations of tasks $\in \{T_i\}$ such that:

– it satisfies the partial ordering of the system of tasks;

– symbols $\overline{T_i}$ and $\underline{T_i}$ appear both once and only once, for all i.

At the end of each execution, all initial tasks, and only those, are again enabled.

For example, in the case of fig. 15, two possible executions are the following:

$$\&' = \overline{T_{11}}\ \underline{T_{11}}\ \overline{T_{12}}\ \overline{T_{21}}\ \underline{T_{21}}\ \overline{T_{22}}\ \underline{T_{12}}\ \overline{T_{13}}\ \underline{T_{22}}\ \underline{T_{13}}$$

$$\&'' = \overline{T_{11}}\ \underline{T_{11}}\ \overline{T_{21}}\ \overline{T_{12}}\ \underline{T_{12}}\ \underline{T_{21}}\ \overline{T_{13}}\ \overline{T_{22}}\ \underline{T_{13}}\ \underline{T_{22}}$$

Two tasks, T_a and T_b, are *independent* if they are not bound by precedence constraints: therefore they may be executed in parallel. Two independent tasks are recognized by the fact that [COF 1] no directed path connecting T_a with T_b exists in the precedence graph. In the example of fig. 15, independent tasks are T_{12} and T_{21}, T_{12} and T_{22}, T_{13} and T_{22}.

We shall call *computation* $\mathscr{C}$ any string composed by an arbitrary, but finite, number of executions:

$$\mathscr{C} = \&_1\ \&_2\ \ldots\ldots\ \&_n$$

The *initial state of a computation* $\mathscr{C}$, denoted by $\mathscr{TC}$, concides with the initial state of the first execution $\&_1$ of $\mathscr{C}$ and it is given by the vector of the initial states of all cycles of the system:

$$\mathscr{TC} = (\mathscr{T}\Gamma_1, \mathscr{T}\Gamma_2, ..., \mathscr{T}\Gamma_p)$$

being the system composed by modules $\mathfrak{M}_1$, $\mathfrak{M}_2$, ..., $\mathfrak{M}_p$.

4. DETERMINACY OF PMC LINEAR SYSTEMS

The determinacy problem always arises when we consider a system composed by modules which may operate in parallel and asynchronously each other: in generic terms in fact a system is determinate if the result of every processing is independent of the relative speeds of the modules constituting it. More particularly the problem arises when the modules share memory cells and/or when their behaviour, as in our case, depends on message exchange.

Consider a PMC system; let $S = \{s_i \mid i = 1, ..., h\}$ be the set of all memory cells and $Q = \{q_j \mid j = 1, ..., k\}$ be the set of all queues in the system $(S \cap Q = \emptyset)$; let $\Sigma(\mathscr{C})$ and $\Theta(\mathscr{C})$ denote the contents of S and Q respectively, as a result of computation $\mathscr{C}$.

Definition 1: let Σ_0 and Θ_0 be the initial contents of S and Q respectively and let $\mathscr{C}'$ and $\mathscr{C}''$ be two computations such that $\mathscr{T}\mathscr{C}' \equiv \mathscr{T}\mathscr{C}''$; a PMC system is *weak-sense determinate* if $\Sigma(\mathscr{C}') = \Sigma(\mathscr{C}'')$ and $\Theta(\mathscr{C}') = \Theta(\mathscr{C}'')$.

Let $[q_j(\mathscr{C})]$ and $[s_i(\mathscr{C})]$ now denote the sequences of values in queue q_j and in memory cell s_i respectively, as a result of computation $\mathscr{C}$.

Definition 2: let Σ_0 and Θ_0 be the initial contents of S and Q respectively and let $\mathscr{C}'$ and $\mathscr{C}''$ be two computations such that $\mathscr{T}\mathscr{C}' \equiv \mathscr{T}\mathscr{C}''$; a PMC system is *strong-sense determinate* if $[q_j(\mathscr{C}')] = [q_j(\mathscr{C}'')] \forall j$ and $[s_i(\mathscr{C}')] = [s_i(\mathscr{C}'')] \forall i$.

Definitions 1 and 2 are related to those available in the literature, for instance [DEN 2], [KAR 1], [KAR 2].

Theorem 1: A PMC linear system with strongly linked modules is strong-sense determinate.

Proof.

The following three points will be useful in the proof:

i) let the precedence graph be given; from conversion rule of the system graph in the precedence graph it follows that in a PMC linear system with strongly linked modules the ordering of tasks defined by the precedence graph is respected, whatever the computation be;

ii) independent tasks operate on distinct memory cells;

iii) independent tasks operate on distinct input and output queues.

Let $\mathscr{C}'$, $\mathscr{C}''$ be two computations such that $\mathscr{TC}' \equiv \mathscr{TC}'' \equiv \mathscr{TC}$ and let Σ_0 and Θ_0 be the initial contents of S and Q respectively. Without loss of generality, suppose that, among all the tasks enabled at the beginning of the computation, it is possible to find a set $\{T_i(0)\}$ not empty, such that sets $\{M_i^I(0)\}$, which belong to their respective initial states, are empty. Regardless of the particular computation, since Σ_0 and Θ_0 are the same, and according to points (ii) and (iii), the contents of S and Q take always the values Σ_1 and Θ_1; moreover tasks $\{T_i(0)\}$, when terminated, send always the sets $\{M_i^0(0)\}$ of messages. Let now $\{T_i(1)\}$ be the new set of enabled tasks for which is $\{M_i^I(1)\} \subseteq \{M_i^0(0)\}$: according to point (i), regardless of the particular computation, only these tasks can initiate. According to points (ii) and (iii) the contents of S and Q take always the values Σ_2 and Θ_2 and tasks $\{T_i(1)\}$, when terminated, send always the sets $\{M_i^0(1)\}$ of messages. The foregoing reasoning can now be applied repeatedly to sets $\{T_i(j)\}$ of tasks, whatever j is. It follows that Definition 2 is satisfied by the first executions, $\&_1'$ and $\&_1''$, of $\mathscr{C}'$ and $\mathscr{C}''$ respectively. Since the system is again in state $\mathscr{TC}$, according to points (i), (ii) and (iii) all other executions of $\mathscr{C}'$ and $\mathscr{C}''$ behave in a manner similar to $\&_1'$ and $\&_1''$ respectively. It follows that Definition 2, under the hypotheses of theorem, is satisfied for any pair of computations $\mathscr{C}'$ and $\mathscr{C}''$. ∎

An indirect confirmation of Theorem 1 is that it verifies the six axioms of Denning's theorem on determinacy [DEN 2].

Corollary 1: A PMC linear system with strongly linked modules looking-ahead is determinate.

The system will be strong-sense or weak-sense determinate, according to whether

auxiliary memory cells (see par. 2.2) are thought of as belonging to S or not.

Proof.

Proof follows directly from Theorem 1 and from definition of PMC system with strongly linked modules looking-ahead, for which such a system does not differ from a PMC system with strongly linked modules, with regard to: a) modification of contents of S and Q and b) production of messages by tasks. ∎

Referring to PMC linear systems with weakly linked modules, it is impossible to give a theorem like Theorem 1 because point (i) of the relative proof is not verified. It can be seen that a weak-sense determinacy can at most be got by forcing the system to go always from the same initial state to the same final state through any sequence of states (where state now is the configuration of contents of S and Q).

5. DEADLOCK PREVENTION IN PMC LINEAR SYSTEMS

Deadlock can be said generically to be the situation in which several tasks belonging to a system are permanently blocked because of requests for resources which cannot be satisfied any more.

System resources can be divided in *reusable*, when they can be reassigned after their release, and *consumable*, when they cease to exist after their use [DEN 1], [HOL 1].

In respect to deadlock problem only consumable resources have interest in a PMC system: specifically data in queues and messages. Initially we will distinguish, for this purpose, *deadlock due to messages* from *deadlock due to data.*

5.1. Deadlock due to messages.

From now we explicitly consider PMC systems with strongly linked modules.

For every task T_i, $\{M_i\} = \{M_i^I\} \cup \{M_i^0\}$ is the set of resources concerning T_i; $\{M_i^I\}$ are the resources T_i *asks for*, while $\{M_i^0\}$ are the resources T_i *holds* during the time from its initiation to its termination.

Definition 3: A PMC system with strongly linked modules is deadlocked due to messages if there is a set of enabled tasks $\{T_{i_l} | l = 1,..., p\}$ which cannot initiate because all messages belonging to $\{M_{i_l}^I\}$ cannot be sent to them.

An example of deadlock due to messages is shown in fig. 16.

Definitions of resources and of deadlock due to messages allow to relate our analysis to classical cases found in the literature and, in particular, allow to use Coffman, Elphick, Shoshani results [COF 2], for which there are three necessary conditions for the existence of deadlock: a) exclusive control; b) no-preemption; c) circular wait.

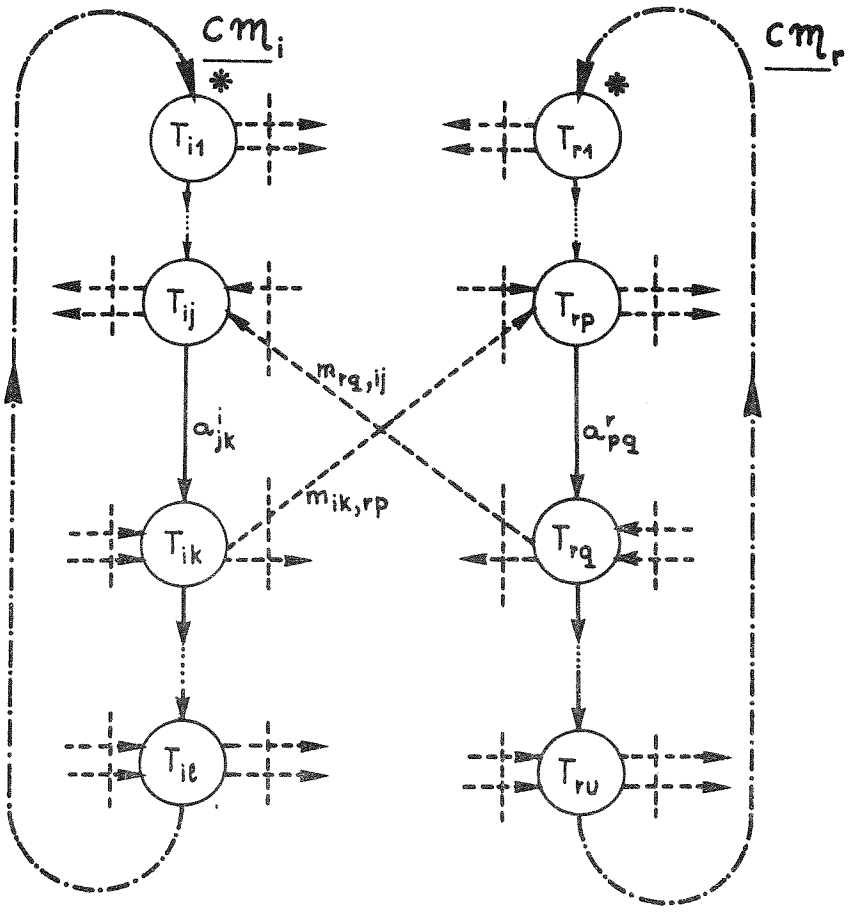


Fig. 16 – Example of deadlock due to messages in a PMC system with strongly linked modules.

In our case conditions a) and b) are always true; therefore, in order to design a deadlock-free system, condition c) has to be false.

Theorem 2: A PMC linear system with strongly linked modules is free from deadlock due to messages if and only if a partial ordering of all tasks of the system can be established.

Proof.

i) *Necessary condition*: if a PMC linear system with strongly linked modules is free from deadlock due to messages, it means, according to Definition 3, that a set of tasks

doesn't exist which, having not received messages from other tasks, cannot send messages to these tasks; it follows that there is not a cycle in the system graph which doesn't include return arcs; thus the corresponding precedence graph does not contain cycles and so it describes a partial ordering.

ii) *Sufficient condition*: if partial ordering exists, the precedence graph does not contain cycles, hence the corresponding system graph does not contain cycles not including return arcs. Then, since the system is composed by strongly linked modules, circular wait among tasks does not exist and so deadlock cannot arise. ∎

Considerations like those used to prove Corollary 1 allow to prove the following:

Corollary 2: A PMC linear system with strongly linked modules looking-ahead is free from deadlock due to messages if and only if a partial ordering of all tasks of the system can be established.

About PMC linear systems with weakly linked modules, it can be said that they are implicitly free from deadlock due to messages. Only if there are tasks which, once enabled, do not initiate for lack of particular messages, conditions of Theorem 2 are valid if partial ordering of tasks is imposed by such messages.

5.2. Deadlock due to data.

Deadlock due to data may occur, in a set of modules $\{\mathfrak{M}_{i_l} \mid l = 0, ..., p\}$, when data lines connecting these modules form a cycle. Let $\{q_{i_l i_n} \mid l = 0, ..., p,\ n = \mid l + 1 \mid_{p+1}\}$ be the queues in the cycle. Under the above hypothesis, deadlock due to data occurs if one of the following conditions is true:

1) being empty all queues in the cycle, it happens that each module $\mathfrak{M}_{i_l}$ requests a datum from input queue $q_{i_k i_l}$ ($l = 0, ..., p$ and $k = \mid l + p - 1 \mid_{p+1}$), before sending a result into queue $q_{i_l i_n}$;

2) being full all queues in the cycle, it happens that each module $\mathfrak{M}_{i_l}$ wants to send a result into queue $q_{i_l i_n}$ before taking a datum out of queue $q_{i_k i_l}$.

We call *complete system graph* of a PMC system the graph derived from system graph by adding a new set of arcs, said data arcs (denoted by ⇒) and so defined: a data arc con-

nects node T_a with node T_b and it is directed from T_a towards T_b if and only if task $T_a \in \Gamma_a$ can cause sending a datum into output queue q'_{ab} and task $T_b \in \Gamma_b$ can cause taking a datum out of input queue q''_{ab}, and it is: $q'_{ab} = q''_{ab}$.

An example of complete system graph for a PMC linear system is shown in fig. 17.

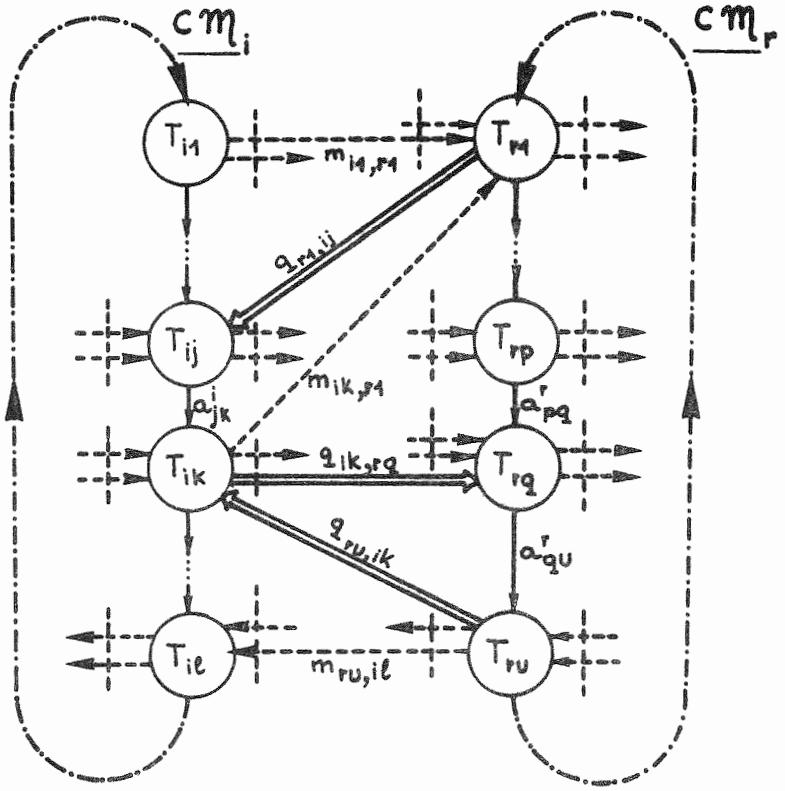


Fig. 17 – Example of complete system graph for a PMC linear system.

Theorem 3: A PMC linear system, whose complete system graph does not contain cycles composed exclusively by data arcs and by a-arcs, distinct from return arcs, is free from deadlock due to data.

Proof.

The condition of non-existence of cycles of the type mentioned in the theorem is sufficient to secure that either situation 1) or situation 2) cannot be ever verified with regard to any set of modules $\{\mathcal{M}_{i_l} \mid l = 0, ..., p\}$; therefore the system is free from deadlock due to data. ∎

The system of the example of fig. 17 is deadlocked due to data, owing to the cycle containing arcs $q_{ik,rq}$, a^{r}_{qu}, $q_{ru,ik}$.

5.3. Conclusions on deadlock in PMC linear systems.

From the definitions given in par. 5.1 and 5.2 the following definition results:

Definition 4: A PMC linear system with strongly linked modules is deadlocked if there exists a set of tasks $\{T_{i_l} \mid l = 1,..., p\}$ such that they cannot initiate, once enabled, because not all messages belonging to $\{M^{l}_{i_l}\}$ can be sent to them and/or they cannot terminate, once initiated,because data they need cannot be sent to them.

From Theorems 2 and 3 the following corollary is directly derived:

Corallary 3: A PMC linear system with strongly linked modules, whose complete system graph does not contain cycles not including return arcs, is deadlock-free.

For example, the PMC linear system of fig. 17 is deadlocked owing to the cycle containing arcs $q_{r1,ij}$, a^{i}_{jk} , $m_{ik,r1}$.

The condition appearing in Corollary 3 is sufficient to secure that also a PMC linear system with strongly linked modules looking-ahead is deadlock-free.

6. CONCLUSIONS

In this report the characteristics of a PMC system are described from the point of view of a) structure of the modules, b) ways of transmitting messages, and c) rules of message exchange.

a) While the structure of message exchange control (MEC) of each module, either it is contained in the processing control (PC) or not, depends essentially upon the particular system, the characteristics of operation part and of PC are such that their structure may be easily made regular and uniform for all systems. Moreover PC may be realized by means of usual techniques, as microprogramming used either only for system description or also for the implementation of PC itself [CAS 1] in a compact and flexible way.

b) Some ways of transmitting messages have been introduced for the purpose of providing alternative solutions for the realization of control schemes; among these solutions one can choose, whenever the distribution functions of processing time of the modules are known, the one which allows to balance, in the best way, the data exchange in the data flow structure with the message exchange in the control structure, and consequently to optimize the performance of the whole system. Definition of a method for the control scheme evaluation will be the subject of a successive work.

c) With regard to PMC linear systems with strongly linked modules, eventually looking-ahead, very general conditions have been derived in order that these systems are determinate and deadlock-free. With regard to PMC linear systems with weakly linked modules, conditions have been derived only for deadlock prevention; since one guesses that some classes of systems can be realized with weakly linked modules, we think it is necessary to search further conditions which secure determinacy of such systems.

The problems concerning point c) have been examined with regard to PMC linear systems. Although in these systems some contraints to the system activity are in general imposed as regards the points of the cycles in which modules can send and receive mes-

sages (and in this connection it seems interesting to investigate on PMC systems of any type, in which these constraints are removed), it is always possible to realize a PMC system as a PMC linear system.

From the points above it follows that the proposed model is directly applicable to the design of parallel and/or pipeline computers.

REFERENCES

[ALT 1] S.M. ALTMANN and P.J. DENNING, "Decomposition of Control Networks", *Proc. Project MAC Conf. on Concurrent Systems and Parallel Computation*, June 1970, pp. 81-92.

[BON 1] P. BONSEIGNEUR, "Description of the 7600 Computer System", *Computer Group News*, May 1969, pp. 11-15.

[BRE 1] T.H. BREDT and E.J. Mc. CLUSKEY, "A Model for Parallel Computer Systems", *Stanford University, Digital System Lab., Tech. Rep. N. 5*, April 1970.

[BRU 1] J. BRUNO and S.M. ALTMANN, "A Theory of Asynchronous Control Networks", *IEEE Trans. on Computer*, vol. C-20, N. 6, June 1971, pp. 629-638.

[CAS 1] G.F. CASAGLIA, G.B. GERACE and M. VANNESCHI, "Equivalent Models and Comparison of Microprogrammed Systems", *Convention CNR-ENI, Internal Report N. 3*, Pisa, August 1971.

[COF 1] E.G. COFFMAN and P.J. DENNING: private communications.

[COF 2] E.G. COFFMAN, M.J. ELPHICK and A. SHOSHANI, "System Deadlocks", *ACM Computing Sur.*, Vol. 3, N. 2, June 1971, pp. 49-65.

[DEN 1] P.J. DENNING, "Third Generation Computer Systems", *ACM Computing Sur.*, Vol. 3, N. 4, Dec. 1971, pp. 175-216.

[DEN 2] P.J. DENNING, "On the Determinacy of Schemata", *Proc. Project MAC Conf. on Concurrent Systems and Parallel Computation*, June 1970, pp. 143-147.

[DIJ 1] E.W. DIJKSTRA, "Co-operating Sequential Processes", in *Programming Languages*, F. Genuys (ed.), Academic Press, New York, 1968, pp. 43-112.

[FLY 1] M.J. FLYNN and P.R. LOW, "IBM/360 Model 91: Some Remarks on System Development", *IBM Journal of Research and Dev.*, Vol. 11, N. 1, January 1967, pp. 2-7.

[FLY 2] M.J. FLYNN, A. PODVIN, K. SHIMIZU, "A Multiple Instruction Stream Processor with Shared Resources", in *Parallel Processor Systems, Technologies, and Applications*, L.C. Hobbs et al. (ed.), Spartan Books, New York 1970, pp. 251-286.

[GRA 1] W.R. GRAHAM, "The Parallel and the Pipeline Computers", *Datamation*, April 1970, pp. 68-71.

[HOL 1] R.C. HOLT, "On Deadlock in Computer Systems", *Computer Systems Research Group, Univ. of Toronto, Tech. Rep.* CSRG-6, Jan. 1971.

[IBB 1] R.N. IBBET, "The MU5 Instruction Pipeline", *The Computer Journal*, Vol. 15, N. 1, Feb. 1972, pp. 42-50.

[KAR 1] R.M. KARP and R.E. MILLER, "Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing", *SIAM J. Appl. Math.*, Vol. 14, N. 6, Nov. 1966, pp. 1390–1411.

[KAR 2] R.M. KARP and R.E. MILLER, "Parallel Programs Schemata", *J. of Comp. & System Sciences*, 3, 1969, pp. 747-795.

[MUR 1] J.O. MURPHY and R.M. WADE, "The IBM 360/195", *Datamation*, Apr. 1970, pp. 72-79.

[PAT 1] S.S. PATIL, "Co-ordination of Asynchronous Events" *Ph. D. Thesis*, MIT, September 1967, (Project MAC TR-72).

[THO 1] J.E. THORNTON, "Parallel Operation in the Control Data 6600", AFIPS Conference Proceedings, Vol. 26, part II, FJCC 1964, pp. 33-41.