A COMPARATIVE STUDY OF LINK PROTOCOLS FOR A TYPICAL FAULT-TOLERANT MULTIPROCESSING UNIT FOR ON-BOARD SATELLITE APPLICATIONS

J. C. MALDONADO

Nº 17

São Carlos
1985

A COMPARATIVE STUDY OF LINK PROTOCOLS FOR A TYPICAL FAULT-TOLERANT MULTIPROCESSING UNIT FOR ON-BOARD SATELLITE APPLICATIONS

J.C. Maldonado*

RESUMO

Este trabalho apresenta um estudo comparativo de protocolos de enlace ("link protocols") com o intuito de implementar tal protocolo de forma a tornar viável a comunicação entre módulos que compõem uma unidade típica de multiprocessamento tolerante a falha. Basicamente dois enfoques são discutidos e os resultados da análise são apresentados.

ABSTRACT

This work presents a comparative study of link protocols looking at the implementation of such protocols in order to make viable the communication between modules which compose a typical fault-tolerant multiprocessing unit. Basically two approaches are discussed and some results of the analysis are presented.

1. INTRODUCTION

In an envoronment where repairs are difficult or not possible, hardware and software reliability is essential to the survival of long missions. Martins and De Paula (1983, 1984) presented a standard for fault-tolerant 16-bit multiprocessing systems to be utilized on board of satellites for their supervision and control. The systems, defined by the proposed standard, are organized in logical hierarchical layers characterized by their modularity, easy adaptation to different satellites and tolerance to single point failures.

These systems, accordingly with this standard, are composed by multiprocessing units. Each one of these multiprocessing units is

---

*Departamento de Ciências de Computação e Estatística - Instituto de Ciências Matemáticas de São Carlos - USP; Caixa Postal 668; 13560 - S.Carlos-SP

partitioned into modules interconnected by redundant buses, each module having an unique address on the bus. Figure 1 shows a block diagram of a typical multiprocessing unit.

The main objective of this work is to present a comparative study of link protocols looking at the implementation of such protocols in a typical multiprocessing unit in order to make viable the communication between modules. In the next sections, a brief description of the Bus Protocol (Physical Layer Protocol), proposed by Alle (1982) and of the link protocol study, conducted at SPAR AEROSPACE LIMITED are provided.

2. BRIEF DESCRIPTION OF THE PROTOCOL (PHYSICAL LAYER PROTOCOL)

The serial unit Bus is used to interconnect modules within the multiprocessing unit. The bus consist of six signal wires. Following the relevant lines are described:

a) Sync - this line carries a pulse used to synchronize transfers on the bus. It has two operating modes: search and acquired.

b) Command data - this line is used to transmit messages between modules. A message is composed of one or more command words.

c) Reply data - this line carries an automatic replay to the previous command word. The reply is supplied by the module that received the command word.

d) Reset - this line is used to force all modules on the bus into a defined "safe" state.

There are two types of interfaces to the Serial Unit Bus mamely Serial Unit Bus Master and Serial Unit Bus Slave Interface. Only Master Interfaces can originate transmissions on the bus. The Slave Interfaces provide an automatic reply word to the previous command word received. A command word is composed of

a) 4-bit binary master address;

b) 4-bit binary slave address;

c) 8-bit function address;

d) 16-bit data field

A reply word from a Slave Interface is composed of:

a) 4-bit status/master address;

b) 4-bit slave address;

c) 4-bit function address;

d) 16-bit reply data field/16-bit data field.

The modules that constitute one unit were classified in two principal classes: CPU module and non-CPU module. A CPU module contains a Master Interface and a Slave Interface, whereas a non-CPU module contains only a Slave Interface. Therefore only a CPU Module can originate transmissions on the Serial Unit Bus.

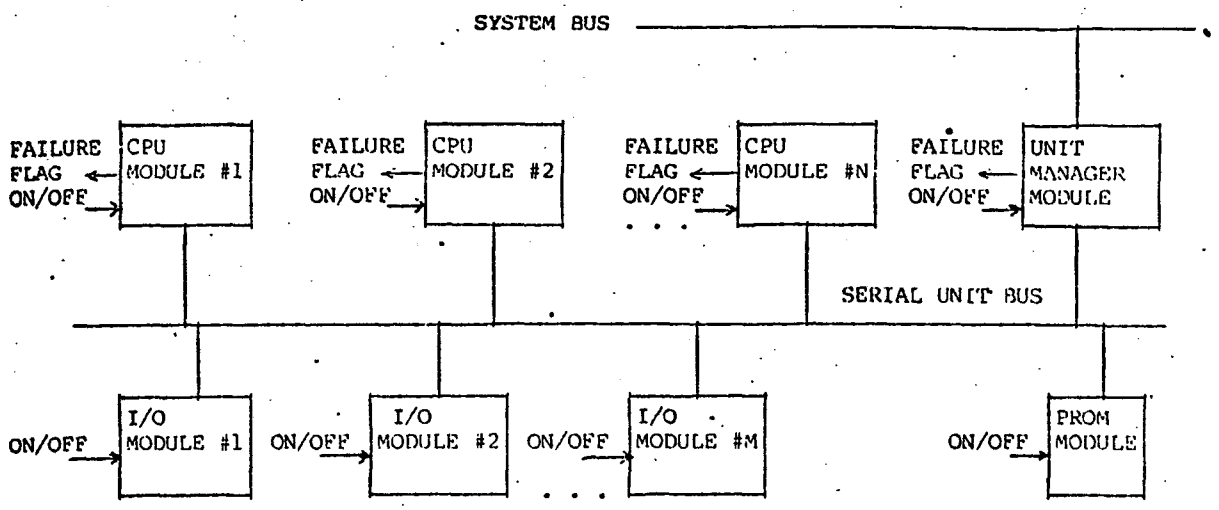


Figure 1 - BLOCK DIAGRAM OF A TYPICAL MULTIPROCESSING UNIT

2.1. CPU Module to non-CPU Module Communications Aspects

This type of communication requires two transfers for a read from the slave. The first transfers defines the data required, the second asks for the data to be returned. The required data is returned on the reply line in parallel with the second command word. A slave, if it is not ready, can ignore the command word. A CPU Module recognizes an ignored command through one flag in the reply word. The slave interface stores the master address of the requesting CPU and ignores commands from any other master until one "dummy command" has been transmitted by the originating CPU module.

2.2. CPU Module to CPU Module Communication Aspects

Communications between two CPU modules was dealt with in a different manner since both have master capability and both are intelligent. The traffic is always one way - a write operation to the CPU module receiving the command. The previous command word is sent automaticaly by the receiving CPU module to the sending CPU module. The slave interface ignores the bus until the receiving CPU reads the incoming command. When the CPU reads the incoming command the slave interface can accept another command from any CPU module in the bus.

2.3. Unit Bus Acess

In a busy multiprocessor system, there are likely to be several master interfaces attempting to use the bus simultaneously. The conflict resolution adopted is described in Alee (1982). This is done by the master interfaces themselves using the master address. The method works by cheching the address on the bus against the master's own address. In this way, any master interface can immediately detect the presence of a higher priority master interface. In this situation it ceases transmission and waits for the next sync pulse to retry.

In the next section the options to implement the protocol are presented jointly with the advantages and disadvantages of each one.

3. LINK PROTOCOL STUDY

The main objective of this section is to present the alternatives to specify the link protocol, it means, allowable messages and operations, the sequence of information and the coding rules for communication between modules of a typical multiprocessing unit. Basically, one has two approaches. The first (OPTION I) uses the automatic reply to confirm the correct reception of the previous command word transmitted to a specific module. The second (OPTION II), does not use the automatic reply to confirm the correct reception of the previous command word transmitted.

In both cases, the following information must be sent between the sending and receiving modules:

a) Slave interface address (fixed by hardware);

b) Master interface address (fixed by hardware);

c) Destination process address;

d) Source process address;

e) Message length;

f) Function to be performed;

g) Data

It is also important to provide some mechanisms to detect some class of errors at the link protocol level. If not, these classes of errors must be detected in a higher layer. When an undected error propagates to higher layers, an increasing amount of states an data structures may be affected, and consequently, the recovery procedures become more complex and a longer time is required to recover the affected data structure. In addition, the following requirements, have to be satisfied:

a) The link protocol must be independent of the mission;

b) The hardware characteristic must be transparent to the application process of the Operating System;

c) The non-CPU modules must be seen as processes, whenever possible;

d) The software that will implement the link protocol has to be as simple as possible;

e) The detection of corruption in the command word transmission has to be detected as fast as possible;

In the following, the advantages and disadvantages of each approach cited above are presented.

3.1. OPTION I - Using the Automatic Reply

In one uses the reply line to confirm the correct reception of the previous command word, one most "lock-out" the CPU slave interface. "Lock-out" means one has to store the sending CPU module address and the CPU slave interface won't accept command words from other CPU modules until it receives a "dummy command word" from the sending CPU module.

The biggest advantage of using the automatic reply is the possibility of detecting corruption in the command word transmission, for example, in the function field or data field. In this way, another process cannot take incorrect decisions based on erroneous information. The recovery procedure becomes less complex and one can avoid a system reinitialization/ reconfiguration.

Another advantage of this approach is that a message receiving process only has to handle one message at a time as a specific module can only receive a message from one CPU module at a time. In This sense, the receiving process is simple and easy to implement. This feature is very important, principally in distributed computer configurations where there exists the potentical for chaotic complexity unless rigid standards are imposed on programming as well as in the intercommunications between computers.

The last important advantage is that the CPU slave interface becomes similar to a non-CPU slave interface, and can minimize costs in the future.

In addition, one can easily satisty the hardware transparency requirements of the Operating System specified in Martins and De Paula (1983). Application process can be programmed independent of the specific CPU module in wich they will run. The basic software remains unchanged, even when functions are reassigned to different computers. One can also see non-CPU modules as processes as recommended by Martins and De Paula (1983). Figure 2 shows the message format for this case.

The disadvantage of this option is that hardware changes is necessary since originally a CPU slave interface doesn't provide a way to "lock-out" itself.

3.2. OPTION II - Without Using the Automatic Reply

If one doesn't "lock-out", the CPU slave interface no hardware change is required. However the automatic reply cannot be used to check the correct transmission of a command word (except for a non-CPU module). In addition, the receiving process, at the same time, has to be able to receive messages from (n-1) other CPU modules where n is the number of CPU modules in the Serial Unit Bus. It is clear that in this case the data structure of the receiving process is more complex than in the OPTION I. Three different cases have been discussed:

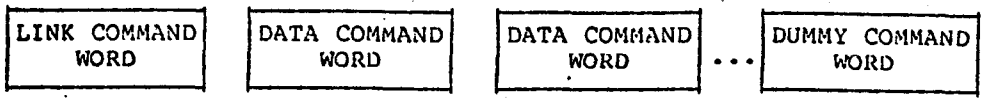
i) A Message composed of only one command word. (OPTION II, case a).

The largest disadvantage of this solution is that it is very difficult to accommodate all information required into a single command word. It is also difficult to provide a way to check the correct transmission of the command word to a CPU slave interface without changes. It can, however, be done by the completion cheker system process, Martins and De Paula (1984). It must be pointed out that this possiblity does exist for the non-CPU Slave Interface since one has to trasmit at least two command words to a non-CPU slave interface to get a data, but the

function will have already been executed.

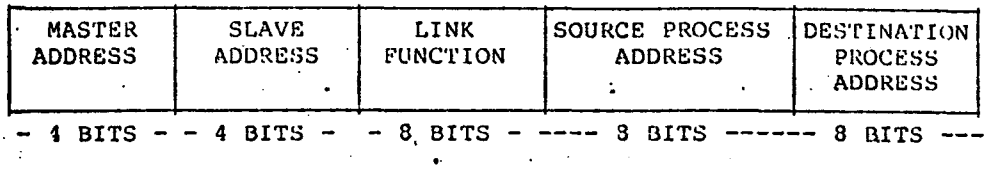
In this case, the sending process does not need to control the context switching between a transmission to a CPU module and to a non-CPU module. However, the data structure of the sending and receiving becomes strongly dependent on the application process location. The communication between two processes located in the same CPU module also becomes more limited if we try to satisfy the hardware transparency requirements.

MESSAGE FORMAT

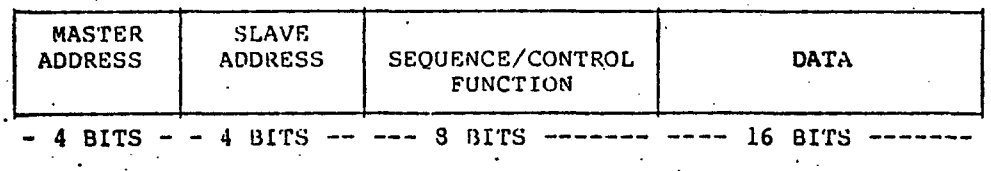


* The message length is fixed for each specific mission.

LINK COMMAND WORD



DATA COMMAND WORD



DUMMY COMMAND WORD

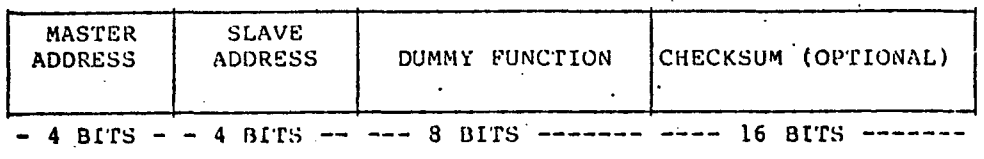


FIGURE 2 - MESSAGE FORMAT - OPTION I

ii) A message composed of a sequence of command words

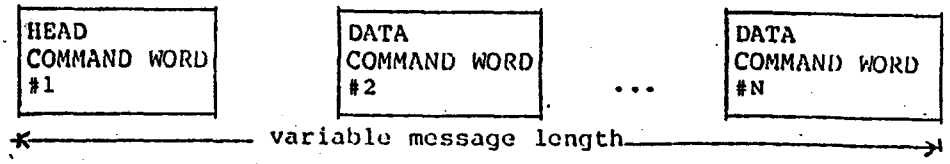
Here we have two different solutions. The first one (Option II, case b) uses the same message format shown in Figure 2. The biggest difference with the OPTION I (using the automatic reply) is that the receiving process has to receive messages from different CPU modules at the same time, therefore, the receiving process software becomes more complex. Another difference is that one can not check the corruption of the command word transmission immediately, but we can check in the correct module was addressed and we can also send a checksum to the receiving process. Therefore, it can check the consistency of the message received. In this manner, one can avoid erroneous decisions based on erroneous information. The second solution (Option II, case c) tries to minimize the overhead words and also supports variable message lengths. The message format for this case is shown in Figure 3. The biggest difference with the above solution is that using the message size indicated in Figure 3, we can acommodate the case a) a message composed of one command word-without overhead words. One must, however, accept that corruption of the command word transmission will be checked only by the completion checker system process (see Martins and De Paula, 1983). One also needs a data structure, a table, that provides some additional information, such as source process, destination process and function to be performed.

Also, in this situation, the receiving process has to receive messages from different CPU modules simultaneously and one can not check corruptions of the command word transmission immediately. It is therefore necessary to send a checksum or a CRC, to enable the receiving process to check the consistency of the received message.

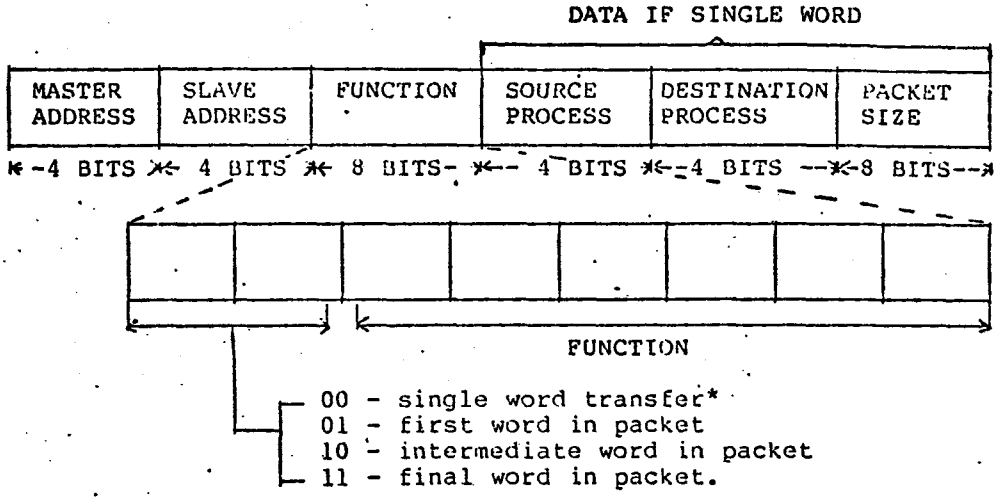
3.3. Error Control Aspects

At every level of protocol there are two types of information: control and data. A protocol provides some service with respect to the data. Control information is sent between, and interpreted by the cooperating protocol modules to provide protocol's service. Error control deals with achieving both reliable data and control transmission.

MESSAGE FORMAT



COMMAND WORD #1 (HEAD WORD)



(*) In this case, the 6 bits function must define the transfer totally. One solution is one 64 word look up table.

DATA COMMAND WORD #2 ... #n

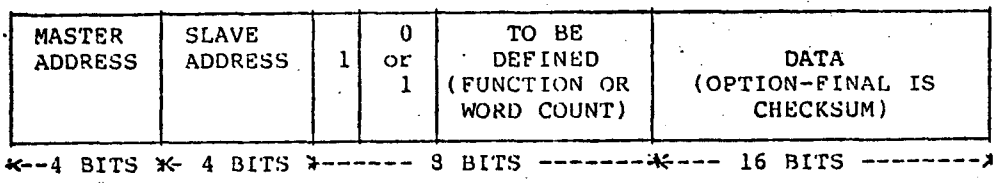


FIGURE 3 MESSAGE FORMAT - OPTION II, CASE C

The error control mechanisms required at any level of protocol depend on what assumptions can be made about the service provided by lower level protocols. We are assuming that the probability of information being damaged is low and delays are small so the error control used is a positive acknowledgement/retransmission procedures. We are assuming also that messages of information cannot be missequenced, duplicated or omitted.

Error detecting code is not used. Consequently, if an error occurs either in the function or slave address field during a transmission between a CPU module and a non-CPU module a wrong function can be executed.

Two different approaches are possible to solve this problem:

a) leave the error confinement strategies to be implemented at user levels

b) implement any mechanism at Serial Unit Bus Interface.

The second one can be implemented using the comand word "don't care" field. This field is composed of 9 bits as defined in Thoerner (1983) and could be used to transmit the comand word function field (8 bits) and the parity bit of the command word slave address field. The slave interface must compare command word function fields to accept a command word. Also the command word slave address field parity bit must be checked. This approach would require changes in the slave hardware design.

3.4. Flow Control Aspects

Resources are required at the receiver to process and store command word and messages. Flow control is the mechanism used to regulate the flow of information between two modules to a rate that the receiver can handle. The procedure utilized here is that the source not send information faster than the destination can absorb it independently of the particular resources management strategy chosen. In this manner a receiving module does not get flooded with messages being sent too rapidly by a sending CPU. The receiving module puts its interface in the busy state, if it can not receive a command word or a message. Any command word transmitted to this module will be discarded and then retransmitted later by the sending CPU-module.

3.5. Implementation Aspects

A full specification of OPTION I and II (case c) was performed according with the following procedure (D'ANGELO, 83).

a) Defining Processing Modules

A structured design breaks the software that implements the protocol into modules.

b) Developing of a State Transition Diagram

To describe the order of processing modules aplication, you can develop a state transition diagram.

c) Developing the State Table

From the state transition diagram, you can develop the state table wich in turn serves as the basis for program code.

Software written in this way is easy to design, code, test and maintain. The state table approach reduces programming to a simple table look-up program.

After the full specification of the options I and II a detailed analyse was performed. Table 1 summarizes the results of this analyse. Based on this table, the most appropriate solution can be chosse in accordance with the mission requirements and constraints.

It must be pointed out that as specified in Martins and De Paula (1983), this protocol is provided by the interrupt service routines and by the system processes. In Maldonado (1984) the characteristics of these routines and system processes are presented.

4. CONCLUSION

This work has described the link protocol study carried out

at SPAR AEROSPACE LIMITED[1], CANADA. Two approaches have been presented to implement the communication between modules. One of them uses the "lock-out" mechanisms in CPU modules and uses the automatic reply word to confirm the correct reception of the previous command word. The other approach does use the "lock-out" mechanism, and consequently, uses the automatic reply word only to confirm if the correct module was addressed. A brief trade-off analyse was presented providing the means to choose the most appropriate solution.

1-The author was sent by INPE - Instituto de Pesquisas Espaciais - in 1984 to SPAR AEROSPACE LIMITED, CANADA to participate in trainning program.

TABLE 1 TRADE-OFF STUDY SUMMARY

OPTIONS / RELEVANT ASPECTS	OPTION I	OPTION II		
		WITHOUT USING AUTOMATIC REPLY		
	USING AUTOMATIC REPLY	MESSAGE COMPOSED OF ONE COMMAND WORD (CASE a)	MESSAGE COMPOSED OF MORE THAN ONE COMMAND WORD	
			CASE b (FIGURE 2)	CASE c (FIGURE 3)
HARDWARE				
- Simplicity	80%	100%	100%	100%
- Equality between Non-CPU and CPU modules	100%	80%	80%	80%
- Versatility	100%	90%	90%	90%
- Compatibility with Existing design	70%	100%	100%	100%
SOFTWARE				
- Sending Process				
Simplicity	90%	100%	90%	90%
Versatility	90%	70%	90%	100%
Expandability	100%	70%	100%	100%
Validation and Maintenance	90%	100%	90%	90%
Data Structure	85%	100%	90%	90%
- Receiving Process				
Simplicity	100%	80%	90%	90%
Versatility	100%	80%	90%	90%
Expandability	100%	70%	90%	90%
Validation and Maintenance	100%	90%	80%	80%
Data Structure	100%	90%	90%	80%
- Communication Between Application Process				
Simplicity	100%	80%	100%	90%
Versatility	90%	70%	90%	100%
Expandability	100%	70%	100%	90%
Validation and Maintenance	100%	90%	90%	90%
- Executive Data Structure - Simplicity	100%	80%	100%	90%
REQUIREMENTS				
- Mission Independency	100%	70%	100%	90%
- Minimum Overhead	90%	100%	90%	100%
- Fault Detection and Error Confinement	100%	70%	90%	90%

REFERENCES

ALLEN, R. et al - "Advanced Attitude Control Eletronic for Satellites Using Microprocessor Technology: System Study Report", SPAR Technical Report RML-009-82-16, April 1982 - CANADA.

ALLEN, R. et al - "Advanced Attitude Control Electronic to Satellites Using Microprocessor Technology: Modular Microcomputer - Breadboard Hardware and Software Detail Description", SPAR Technical Report, RML-009-82-67, December, 1982. CANADA.

D'ANGELO, Phill - "State-Table Driver Code Simplies Software Development" EDN, September 1983, pp 189-196.

MARTINS, R.C.O. and DE PAULA, A.R. - "A Fault-Tolerant Multiprocessing Unit for On-Board Satellite Applications", SPAR Technical Report RML-009-84-57, April 1984 - CANADA.

SHATZ, S.M. - "Communication Mechanisms for Programming Distributed Systems", Computer, June 1984, pp 21-28.

THOERNER, R. and DRISCOLL, J. - "Completion of the A.A.C.E. Program" SPAR Technical Report RML-009-83-140, December 1983.

MARTINS, R.C.O. and PAULA, A.R. - "A Fault-Tolerant Multiprocessing Unit for On-Board Satellite Supervision and Control", 5º Congresso Bras. de Matemática/1º Congresso Lat. Americ. de Automática, 1984.

MALDONADO, J.C. - "A Comparative Study of Link Protocols for the SPAR ON-BOARD MODULAR MICROCOMPUTER", Technical Report RML-009-84-131, August, 1984.