



BINDING SERVICES
Tel +44 (0)29 2087 4949
Fax +44 (0)29 20371921
e-mail bindery@cardiff.ac.uk

**Intelligent Distributed Process Monitoring and
Management System**

By

Marcos R. Frankowiak

February, 2004

**Intelligent Process Monitoring and
Management (IPMM) Centre
Cardiff School of Engineering
Cardiff University**

UMI Number: U584639

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI U584639

Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ACKNOWLEDGEMENTS

This work was carried out at the Intelligent Process Monitoring and Management (IPMM) Centre, Cardiff School of Engineering, Cardiff University. Special thanks are owed to Dr. R.I. Grosvenor and Mr. P.W. Prickett who originated and enthusiastically supervised this research project, contributing with their knowledge and expertise to properly advise and guide the activities that resulted in the realisation of this work.

Grateful thanks are offered to Dr. A.D. Jennings and Mr. J.R. Turner for their support and contribution to this investigation, while staff members of IPMM Centre, always helpful in facilitating the research activities.

Equally important was the support provided by my sponsor, CAPES of Brazil, giving financial support and guidance that enabled me to fully concentrate on the important aspects of the research activities.

Microchip Technology Inc. many times offered technical support and contributed with samples required and employed in the research, thus easing many aspects that resulted in the final results.

Finally, my greatest thanks go to my family specially my wife Jeanete and my son Gustavo, that have endured patiently with me during this work.

SUMMARY

Monitoring systems represent an important tool to support efforts aimed at improving productivity and quality, reducing waste and enhancing safety in manufacturing. Modern technologies including electronic devices, communication technology, the Internet, database systems and modern computer technology represent resources that can provide flexible and cost accessible attractive and efficient solutions for the implementation of distributed and intelligent monitoring systems.

A new generation of microcontrollers offer a high level of integrated devices and operate at low power, making them the ideal choice for many embedded industrial applications. However, the development of application software for microcontroller-based implementations has normally been a restrictive factor. Before this work this has resulted in most process and condition monitoring systems being PC based.

This research presents an intelligent and distributed monitoring system based on microcontroller technology, specifically the PIC18C452. The system uses a flexible architecture that can be adapted to the necessities of different monitoring applications. "Monitoring Modules" that can be deployed according to the application requirements were developed. Industrial networks and Internet technologies are employed to enhance communication, therefore allowing monitoring records to be made available in a remote database. The Petri-net concept is used to represent the monitoring task in such a way as to provide independence from the system's hardware and software. Extensions to the original Petri-net theory and new modelling elements, including the acquisition of analogue signals, required to support the use of this method in a microcontroller-based environment, are presented. These enhancements represent a major contribution of this research.

Finally, the benefits of the system are considered by means of three application examples; a simple Press Rig to illustrate the general features and use of the system, a more complicated Assembly Process Rig to show the flexibility of the modelling approach, and finally a CNC Milling Machine tool changer is used to demonstrate the system in a real manufacturing application.

CONTENTS

ACKNOWLEDGEMENTS	i
SUMMARY	ii
NOMENCLATURE	viii
CHAPTER 1 – INTRODUCTION	1
CHAPTER 2 – RESEARCH MOTIVATION	5
References for Chapter 2	10
CHAPTER 3 – LITERATURE REVIEW	
3.1 – Introduction	12
3.2 – Fault Diagnostics	14
3.2.1 – Fault Tree Analysis Approaches	14
3.2.2 – Model Based Approaches	17
3.2.3 – Intelligent System Based Approaches	18
3.3 – Condition Monitoring	20
3.3.1 – IPPM Centre Research	21
3.3.2 – Machine Tool Condition Monitoring	22
3.3.2.1 – Sensor-based Systems	23
3.3.2.2 – Non-sensor Based Monitoring	26
3.4 – Integrated Monitoring Systems	27
3.5 – Distributed Monitoring Systems	34
3.6 – Embedded and Microcontroller-based Monitoring Systems	43
3.7 – Future Directions	48
3.8 – Summary	54
References for Chapter 3	55
CHAPTER 4 – TECHNOLOGY FUNDAMENTALS	
4.1 – Introduction	61
4.2 – Processing Technology	61

4.2.1 – The PIC18C452 Microcontroller	65
4.3 – Data Acquisition	73
4.4 – Industrial Networks	77
4.4.1 – Controller Area Network (CAN)	80
4.4.2 – MCP2510 CAN Controller	88
4.5 – The Internet Protocols	90
4.5.1 – The IP Protocol	91
4.5.2 – The TCP Protocol	94
4.5.3 – The UDP Protocol	96
4.5.4 – Management Protocols	97
4.6 – Internet Embedded Technology	100
4.7 – Database Systems	101
4.8 – Summary	103
References for Chapter 4	104

CHAPTER 5 – PETRI-NET CONCEPT

5.1 – Introduction	108
5.2 – Petri-net Representation	109
5.3 – Petri-net Definitions	111
5.4 – Petri-net Properties	116
5.5 – Petri-net Extensions	119
5.6 – Summary	122
References for Chapter 5	123

CHAPTER 6 – PETRI-NET MONITORING MODEL

6.1 – Introduction	124
6.2 – The Modelling Approach	125
6.3 – The Model Structures	128
6.3.1 – Places	128
6.3.2 – Sub-nets	129
6.3.3 – Transitions	129
6.3.3.1 – Ordinary Transition	129
6.3.3.2 – Analogue Transition	133
6.3.3.3 – Delay Transition	134

6.3.3.4 – Output Transition	135
6.4 – Implementations Aspects and Representation	136
6.5 – Monitoring Records	138
6.6 – Fault Diagnostics Approach	142
6.7 – Summary	144
References for Chapter 6	146

CHAPTER 7 – SYSTEM DESCRIPTION

7.1 – Introduction	148
7.2 – Monitoring Module (MM)	149
7.2.1 – Hardware Description	149
7.2.2 – Software Description	152
7.2.2.1 – Data Communication Aspects	158
7.3 – Connectivity Module (CM)	161
7.3.1 – Hardware Description	161
7.3.2 – Software Description	162
7.3.2.1 – CAN Node Implementation	163
7.3.2.2 – Internet Connectivity Implementation	166
7.3.2.2.1 – Protocols Implementation	166
7.3.2.2.2 – Operation Description	169
7.4 – Management Application (MA)	170
7.4.1 – System Data Tables	171
7.5 – System Tests and Measurements	172
7.5.1 – Linearity	172
7.5.2 – Repeatability	173
7.5.3 – Analogue Input Mean Value Accuracy	173
7.5.4 – System Communication Testing	174
7.6 – Summary	175
References for Chapter 7	176

CHAPTER 8 – PRESS RIG MONITORING TASK

8.1 – Introduction	178
8.2 – Process Overview	178
8.2.1 – Press Rig Component Description	179

8.2.2 – Monitoring Task Analysis	180
8.3 – Press Rig Petri-net	182
8.4 – Monitoring Results	187
8.4.1 – Data presentation Approach	188
8.4.2 – Monitoring Presentation Examples	188
8.5 – Summary	193
References for Chapter 8	194

CHAPTER 9 – CONVEYOR RIG MONITORING TASK

9.1 – Introduction	195
9.2 – Process Description	195
9.2.1 – Conveyor Rig Component Description	197
9.2.2 – Monitoring Task Analysis	197
9.3 – Conveyor Rig Petri-net	198
9.4 – Monitoring Results	206
9.5 – Summary	212
References for Chapter 9	213

CHAPTER 10 – TOOL CHANGER MONITORING TASK

10.1 – Introduction	214
10.2 – Operation Description	214
10.2.1 – Component Description	216
10.2.2 – Monitoring Task Analysis	217
10.3 – Tool Changing Petri-net	218
10.3.1 – Tool Selection and Cutting Process Monitoring	219
10.3.2 – Z Axis Positioning and Spindle Orientation	222
10.3.3 – Tool Changer Horizontal Movement	223
10.3.4 – Tool Changing Operation	224
10.4 – Monitoring Results	227
10.5 – Summary	230
References for chapter 10	231

CHAPTER 11 - DISCUSSION

11.1 – Introduction	232
---------------------	-----

11.2 – Implementation Aspects	233
11.2.1 – Hardware Considerations	233
11.2.2 – Local Communication	235
11.2.3 – Remote Communication	238
11.2.4 – Data Analysis and Integration	240
11.3 – Modelling Method Aspects	241
11.3.1 – Extended Features Supported by the Petri-net Approach	247
11.3.2 – Fault Diagnostics	249
11.4 – The Research in the Monitoring Context	251
References for Chapter 11	253

CHAPTER 12 – CONCLUSION AND FUTURE WORK

12.1 – Main Contributions of the Research	255
12.2 – Conclusions	256
12.3 – Future work	258

Appendix A: Development Related Details

Appendix B: Monitoring Applications Related Details

Appendix C: Publications and Presentations

Note: Additional documentation produced as a result of this research (i.e. program listings, test set ups and results, previous publications, etc.) are provided in electronic format and included in the attached CD-ROM.

NOMENCLATURE

AC – Alternate Current
ADC – Analogue to Digital Converter
AE – Acoustic Emission
ARP – Address Resolution Protocol
CAD – Computer Aided Design
CAN – Controller Area Network
CM – Connectivity Module
CNC – Computer Numerical Control
CRC – Cyclic Redundancy Check
DAS – Data Acquisition System
DBMS – Database Management System
DC – Direct Current
DNC – Direct Numerical Control
DSP – Digital Signal Processor
EPROM – Erasable Programmable Read-Only Memory
ES – Expert System
FDI – Fault Detection and Isolation
FMS – Flexible Manufacturing System
FTA – Fault Tree Analysis
Hz – Hertz
ICMP – Internet Control Message Protocol
IEEE – Institute of Electrical and Electronics Engineers
IP – Internet Protocol
IPMM – Intelligent Process Monitoring and Management Centre
ISO – International Standards Organisation
KBS – Knowledge-Based System
LAN – Local Area Network
LED – Light-Emitting Diode
LSB – Least Significant Byte
MA – Management Application
MM – Monitoring Module

MSB – Most Significant Byte
NCAP – Network Capable Application
NRZ – Non-Return to Zero
ODBC – Open Database Connectivity
OEE – Overall Equipment Efficiency
OSI – Open System Interconnection
PC – Personal Computer
PIC – Peripheral Integrated Controller
PLC – Programmable Logic Controller
RAM – Random Access Memory
RISC – Reduced Instruction Set Computer
RMS – Root Mean Square
rpm - rotation per minute
SPI – Serial Peripheral Interface
SQL – Structured Query Language
STIM – Smart Transducer Interface Module
TCP – Transmission Control Protocol
UDP – User Datagram Protocol

CHAPTER 1

INTRODUCTION

Present day competition in manufacturing requires the deployment of new and efficient methods and techniques to obtain the best results. New machines and equipment are capable of high speed production to high quality standards. However, to achieve the best results when exploiting such features, adequate management of resources is required.

Monitoring systems have become an important element in this context. They can be used to provide information in order to allow engineers to manage assets and control industrial processes. They can follow process events and monitor machine conditions, thus providing the means to prevent losses, help in predicting critical situations and to enable efficient planning. As a consequence machine manufacturers should be able to monitor their deployed equipment all over the world and provide their customers with fast and efficient maintenance and updating services.

This research investigates the development of low-cost monitoring systems, that are capable of being deployed and embedded in manufacturing process and machines. These are provided with the required level of processing capabilities and intelligence, based on modern technology. It was considered that such systems require the capability of providing precise records that can be further integrated within a wider range of applications, from remote monitoring to top-level management.

Each of these areas is dealt with in this thesis, which is organised in 12 chapters that cover the technology, techniques and methods employed and presents a range of examples that demonstrate the results obtained from the use of the proposed system.

The aims of this research are presented in Chapter 2 based upon the demands and requirements of modern manufacturing systems. It considers why it is that, although

the technology is available and the benefits well known, the use of monitoring techniques is still restricted.

Chapter 3 provides a review of several aspects related to monitoring techniques. Condition Monitoring, an important sector within the monitoring area, has generated a large number of investigations and still offers a number of opportunities due to the area's complexity. Fault diagnostics is another important area that has found added impetus with the development of artificial intelligence tools. The integration of such techniques, under the designation of monitoring systems, is also experiencing increased interest. Initially Condition Monitoring was limited to stand alone computers, however new networking technologies and protocols have increased flexibility and reduced restrictions. At the same time, compact and powerful processors enable the development of a distributed concept in the control and monitoring field.

The technology on which this research was based is reviewed in Chapter 4. It introduces important aspects of the PIC18C452 microcontroller, that forms the core of the monitoring system implemented as result of this investigation. CAN bus and Internet protocols, which represent important elements used in order to enable the development of a distributed monitoring concept, are also introduced in this chapter. A brief description of some aspects related to database systems is also presented, as they become an important component in the System's structure for the analysis, presentation and integration of the information produced.

An important element of the System is the modelling technique that enables the design of the monitoring task. The Petri-net concept was selected in this research for this purpose. Two chapters, 5 and 6, were dedicated to the topic. The first one considers the fundamental aspects of the theory and the extensions introduced in order to improve its use in practical applications. Chapter 6 describes particular extensions that were required as a result of this research, in order to enable the use of the method as a modelling technique for monitoring purposes.

The structure and development aspects of the System are presented in Chapter 7. Different components were required to allow monitoring records to reach a remote

database over the Internet. Hardware and software aspects of the Monitoring Module, Connectivity Module and Management Application are provided in detail. These represent the structure of the distributed concept implemented in this research.

The use of the produced System is described in the following three chapters. In Chapter 8 a laboratory-based functional model of an Asea Press, referred to herein as the Press Rig, was monitored by the System to show how maintenance and managerial information can be produced and presented using a web page based approach. Particularities of the application and modelling technique are also described.

In Chapter 9, another experimental application task is presented, using further laboratory-based equipment, namely the Conveyor Rig. This represents a very important example, as it provided a good level of complexity to test and subsequently demonstrate the System's capabilities in terms of modelling flexibility and generality. From the records produced by the Monitoring Module, a wide range of information is presented to meet different managerial requirements.

In the last example, in Chapter 10, a tool changer on an industrial milling machine is used as a monitoring application. A CNC machine tool changer is an important component and its operation and failure can significantly affect the efficiency of the machining process. The monitoring system was deployed to monitor the operation of this tool changer. Using this system's special resources, the actual cutting tool usage was monitored in order to provide additional management information to help in their assessment.

Chapter 11 is dedicated to the discussion of different aspects of this research. Questions related to the capabilities and limitations of microcontrollers for such applications are considered. The importance of the System's architecture in achieving the final results is also analysed. Similarly, the Petri-net approach as a modelling method for monitoring purposes and its flexibility in dealing with real situations represented by the application examples is considered. The potential and the range of applications of the System emerge as a result.

Finally in Chapter 12, the conclusions and future work suggestions are presented. The conclusions are based on the different aspects addressed by the research: the microcontroller, architecture, modelling method and System's capabilities. The continuity and enhancements of the System are presented as future work topics and give an indication of the opportunities resulting from this research in the investigation and development of flexible and feasible monitoring technology.

CHAPTER 2

RESEARCH MOTIVATION

The demand for systems and processes that can operate without failures and errors continuously provides motivation for new research in all areas of manufacturing. The main motivation for such research is based on economic factors. However, any implementation of new technologies and techniques to support the manufacturing process will also result in a more efficient use of resources and thus is environmentally friendly.

Business competitiveness is directly affected by manufacturing factors such as cost, quality, flexibility and timing, as indicated by the results of a manufacturing strategy survey [2.1]. As a result of this, machine manufacturers are continuously encouraged by their customers to develop machines that allow higher levels of productivity whilst at the same time providing better quality outputs. Such machines require repeatability, flexibility and must operate with minimum downtime in order to lower production costs. These requirements imply that machines must run faster and be capable of operating in a more precise way. Therefore, modern machines and their controllers are becoming more complicated. In addition it is becoming increasingly vital that the operating conditions of these complicated machines should be kept close to their optimum specifications [2.2]. Hence the need for more accurate and cost effective maintenance and monitoring tools.

Swanson [2.3] carried out research to identify the impact of new technologies in manufacturing processes with respect to maintenance activities. This indicated that modern processes may integrate several operations into one single step. This creates a bigger dependency on equipment, with high cost implications in the case of failure. Processes with higher levels of automation rely much more on efficient maintenance practises. Considering all the implications, Swanson stated that “maintenance resources must be quickly and properly directed to solve problems”. The survey enabled Swanson to conclude that “preventive and predictive maintenance allow the

maintenance function to better support equipment availability and performance”. There was also an indication that the integration of the information process between maintenance and production management is equally important to enable rapid, right-first-time responses to improve overall equipment performance.

An investigation attempting to quantify the financial benefits of condition monitoring based maintenance indicated that direct repair cost of machinery can be 80% higher without the use of such techniques [2.4]. In their research the authors only considered measurement methods based upon the use of hand held devices that require human intervention to collect the data. Clearly these savings can be further increased by the deployment of fully automated techniques for such a purpose. These methods can extend the intervals between maintenance, manage plant degradation in a controlled way, avoid wasteful maintenance routines and reduce maintenance induced failures and thus decrease lost production time [2.5].

In considering all the factors concerning modern manufacturing and maintenance requirements, it is becoming essential that computerised tools should be deployed in order to achieve high production standards [2.2, 2.3, 2.6, 2.7, 2.8]. Such tools can help to diagnose existing faults, or even better, monitor the system and manage any measured degradation in a controlled fashion.

Looking from another perspective, information can be seen to be a crucial asset in modern manufacturing enterprises. Gunasekaran and McGaughey [2.6] indicated the importance of information in a new manufacturing configuration, called virtual manufacturing, where partners, normally located at different sites, have to react together in response to market demands. In many cases, the concept of manufacturing for stock, based on economic batch quantities, has now been replaced by production models that prize flexibility [2.9]. To support these developments, real-time process monitoring at different levels (machine health, process efficiency, etc) is increasingly being identified as a key issue in order to provide accurate information regarding the process and in supporting decisions that need to be made concerning production dynamics [2.10].

From the point of view of machine manufacturers competitiveness is also associated with their capability to provide customers with an efficient after sales support [2.2]. In an article de Vicq indicated that as businesses spread all over the world, Internet based technologies should be considered as an mechanism to support remote maintenance activities and to facilitate actions aimed at reducing the downtime due to breakdowns. In doing this the machine manufacturers will also generate databases capturing data gathered from different machines at diverse customers sites. From this they can build the required knowledge to answer customers' requests quickly.

Similar opinion was issued in another article [2.8]. Here it was suggested that competitiveness could be improved by establishing a positive relationship between supplier and customers. The article considers that new generation of machines will be supplied with embedded monitoring capabilities using Internet technology. In such a scenario, the machine manufacturers would be able to remotely monitor and compare machines at different locations. A consistent database built on data provided by different machines and locations would allow the prediction of machine faults or identify corrections, therefore helping customers to achieve higher levels of performance with less downtime. The same article indicated that remote diagnostics might also reduce maintenance intervention costs due to expertise allocation at faraway sites. Tulpule [2.11] described the use of such an implementation in the Heating, Ventilation and Air-Conditioning (HVAC) manufacturing sector, calling it "e-service". He considered that the deployment of this sort of technology is vital for manufacturer's competitiveness.

The idea of employing Internet technology to improve maintenance capabilities was also supported by Lofall [2.12]. He argues that information can be shared over the web using a widely employed interface such as a web browser and that other technologies, for instance e-mails, are already available to supply timely information to specific users. In this way powerful web based condition analysis tools could be deployed, reducing costs (by sharing expensive software) and training requirements. Davies and Greenough [2.7] conducted a survey which showed that among Computerised Maintenance Management Systems (CMMS) users, 50% indicated that such systems should include better integration with web and office based technologies, in order to improve their application. Another argument in favour of

such technology is the indication that web based applications represent the fastest growing information systems in the manufacturing industry [2.13], showing that it is becoming the preferred platform for information sharing.

Despite all the new developments that may enhance manufacturing technology, it was suggested by Dunn [2.14] that in terms of condition monitoring, the technology still has not addressed business needs and that it has “a long way to improve”. He considered that managers are not fully convinced of the benefits of the technology and that in many cases return-on-investment is difficult to assess. In his analysis, Dunn considered some points that should be addressed to improve existing condition monitoring technology. Among these the necessity of considering the real-time equipment operating conditions when assessing specific information and also the requirement for a more accessible technology in terms of cost were suggested. He also considered that the equipment efficiency is a parameter that should be monitored in order to support condition-based maintenance. The most promising and important trends in the area, as indicated by Dunn, are the development of low-cost and online monitoring devices that will permit cost-effective monitoring of key equipment parts. At a higher level in the monitoring activity it is sensible that high-level software tools, that are often expensive, should be based in central organisations where expertise can be deployed to analyse the monitored data.

In considering all these aspects and also taking into account the support provided by existing technology and new breakthroughs, this research proposes to investigate the engineering of a new type of monitoring system to overcome existing limitations and meet these requirements. The following outlines how the research aims to approach the main considerations:

a) Cost:

It is proposed to utilise microcontroller technologies in order to deal with the cost factor restricting the take-up of current monitoring systems. This was seen to be timely since continuous development within the electronics industry is providing such devices with ever increasing processing capabilities. This will enable their use in many applications until recently restricted to computers.

b) Reusability:

The use of microcontrollers in a practical way for such applications requires the development of an approach that enables the modelling of monitoring tasks to be undertaken in a straightforward manner. It is essential for this approach to support the reusability of basic hardware and software in different implementations. In this research it is proposed that the use of a developed Petri-net concept as a modelling tool will provide reusability and easy implementation.

c) Flexibility and Integration:

The capability of adapting the basic hardware and software to form a complete monitoring system to meet application specific requirements is to be achieved by providing data communication facilities. Industrial networks, in the specific form of Control Area Network (CAN) bus, are to be used as the basis for communication to enable several monitoring modules to be joined and dedicated to perform a major task. In this way, the individual modules can be deployed where they are required within the system in a flexible manner, each one monitoring a specific part of the process, whilst communicating with each other in order to remain synchronised.

d) Data Acquisition and Presentation:

In order to make the monitored data available for further use, such as the deployment of analysis tools or to support other management application tasks, remote database storage and access is proposed. This will be based on Internet connectivity, current database technology and the use of tools such as Structured Query Language (SQL). Such standards, which are becoming more widely used, enable different applications to access the data and information obtained from process or machine monitoring. The same structure will provide a way to present information on a general and widely used environment represented by Internet web browsers.

In proposing this research it is recognised that, considering the complexity concerning such a wide research area, it is not possible to solve all the current problems. What is intended is that this work should support the development of process and condition monitoring system. This can be used to provide examples of the use of these systems and thus help in the establishment of this type of approach, which is clearly much needed.

REFERENCES

- 2.1 **Demeter, K.** Manufacturing Strategy and Competitiveness. *International Journal of Production Economics*, 2003, 81-82, 205-213.
- 2.2 **de Vicq, A.** 21st Century Machinery. *IEE Manufacturing Engineer*, 2001, 80(3), 104-109.
- 2.3 **Swanson, L.** An Information-processing Model of Maintenance Management. *International Journal of Production Economics*, 2003, 83, 45-64.
- 2.4 **Rajan, B.S. and Roylance, B.J.** Condition-based Maintenance – A Method of Assessing the Financial Benefits. *International Journal of COMADEM*, 2001, 4(2), 13-18.
- 2.5 **Tinham, B.** Save Money with Condition Monitoring. *Control and Instrumentation*, 1995, January, 31 and 32.
- 2.6 **Gunasekaran, A. and McGaughey, R.** Information Technology/Information Systems in 21st Century Manufacturing. *International Journal of Production Economics*, 2002, 75, 1-6.
- 2.7 **Davies, C. and Greenough, R.** The Use of Information Systems in Fault Diagnosis. Plant Maintenance Resource Centre. Available from: http://www.plant-maintenace.com/articles/Information_Systems_in_Fault_Diagnosis.pdf [Accessed 14 February 2003].
- 2.8 Maintenance is not as Mundane as it Sounds. ManufacturingNews.com Web Site, Available from: <http://www.manufacturingnews.com/news/01/1130/art1.html> [Accessed 14 February 2003].
- 2.9 Manufacturing Control in Real Time. Online.com Web Site, Available from: <http://www.es2002online.com/frontpage/es2002/SpecialR.asp?ID=52> [Accessed 12 July 2002].
- 2.10 **Jennings, A.D., Prickett, P.W., Grosvenor, R.I. and Frankowiak, M.R.** Process and Condition Monitoring using the Internet (E-Monitoring). *In Proceedings of COMADEM 2002*, Birmingham, UK:Comadem International, 2002, 45-52.

- 2.11 **Tulpule, S.** E-Service – Customer Centric Maintenance. *In Proceedings of COMADEM 2000*, Houston, USA:The Society for Machinery Failure Prevention Technology, 2000, 895-904.
- 2.12 **Lofall, D.** Information Everywhere – Connectivity is the Future of Predictive Maintenance. DLI Engineering Corporation, DLI Machine Condition Monitoring Web Site. Available from: <http://www.dliengineering.com/downloads/info-everywhere-dl.pdf>, [Accessed 7 March 2003].
- 2.13 **Coronado, A.E., Sarhadi, M. and Millar, C.** Defining a Framework for Information Systems Requirements for Agile Manufacturing. *International Journal of Production Economics*, 2002, 75, 57-68.
- 2.14 **Dunn, S.** Condition Monitoring in the 21st Century. Plant Maintenance Resource Center, Plant Maintenance Web Site, Available from: <http://www.plant-maintenance.com/articles/ConMon21stCentury.shtml>, [Accessed 10 August 2002].

CHAPTER 3

LITERATURE REVIEW

3.1 – Introduction

Monitoring systems have numerous applications in many different areas. They are seen as a growth area in building technology [3.1], the environment [3.2] and human health [3.3] for example. This research and hence this literature review concentrates specifically on industrial applications, although the resulting benefits may be of use in the other application fields.

Monitoring in industrial environments is becoming an increasingly important issue, since companies have to pursue many different ways of cost reduction and efficiency improvement, in order to remain competitive. Process and condition monitoring are useful tools in achieving these targets, because of their capability of providing key information that is necessary to plan production in a strategic and efficient way. Some functions such as maintenance that are not value adding production activities are being continuously pushed to reduce their cost, while keeping machinery and equipment running as long as possible without interruption. Condition monitoring is one tool that enables such actions by reducing part replacement costs, machinery downtime and hopefully, avoiding catastrophic damage to a company's assets.

This chapter is structured to present a review of some important topics that should be considered when implementing or deploying a monitoring system. The review concentrates on the ability of such systems to supply process and machine information required for their management and assessment. These systems should provide not only cost benefits, but also more environment friendly industrial processes.

The chapter is divided into sections that consider the evolution of modern process monitoring and management systems.

Chapter 3 – Literature Review

Section 3.2 considers the tools that can be used for Fault Diagnostics. It presents an overview of the different approaches that have been taken to identifying and diagnosing possible fault conditions.

Section 3.3 presents a review of both sensor and non-sensor based Condition Monitoring. This includes an overview of the initial contributions made to this field by the Intelligent Process Monitoring and Management Centre, within which this research was conducted.

Section 3.4 considers how condition monitoring technology has been integrated to form more complete Integrated Monitoring Systems that have process monitoring and management functions. These systems normally run using PC based technology, and this section considers the benefits and disadvantages this brings.

Section 3.5 outlines the evolution of Distributed Monitoring Systems, which make best use of the ever more effective and accessible tools that are available to connect and integrate computing power. The resulting systems operate with monitoring functions being distributed around the process, so that timely and accurate monitoring actions may be enabled.

Section 3.6 considers how the availability of powerful microcontroller technology has been used to develop low-cost distributed monitoring systems. Microcontroller based Monitoring Systems are being developed using this technology that distributes microcontroller-based measurement and processing functions within the process.

Finally, Section 3.7 on Future Directions considers the steps that are being taken to support the development of the next generation of such systems, including the work that forms the basis of this research.

3.2 – Fault Diagnostics

The complexity of modern manufacturing systems requires the deployment of techniques such as Fault Detection and Isolation (FDI) to support the achievement of high quality standards, to facilitate fast recovery from fault states and to enable the timely detection of the development of part and equipment fault conditions. FDI development has therefore been the objective of several reported research programs aiming to investigate different methods to better achieve such goals.

In outlining approaches to be deployed within the development of FDI systems Stephanopoulos and Han [3.4] considered that the non-linear behaviour normally associated with real systems was a source of complexity that must be recognised when trying to implement fault diagnostic systems. Different methods were applied, depending upon the specific application requirements. They considered pattern recognition to represent the best approach for fault diagnosis. This category included look-up tables relating symptoms and faults; neural networks to compute a pattern representing the system state; and decision trees, which were stated to be direct and open to human interpretation, since knowledge is organised in a hierarchical way, where a sequence of “questions” and “answers” within the knowledge base lead to the decision process [3.5].

Meziane et al [3.6] provided a review that considered the employment of intelligent systems in the manufacturing area. It indicated that for the specific application of maintenance and fault diagnostics researchers have mainly concentrated on Knowledge Based Systems (KBS) and Neural Networks (NN). KBS were largely used for fault classification and diagnostics due to their capability to incorporate human knowledge. NN were presented as a better alternative for the cases where domain expertise is not available, due to their learning capabilities.

3.2.1 – Fault Tree Analysis Approaches

Fault Tree Analysis (FTA) represents one of the most natural ways to analyse and classify faults. Fault trees have the ability to model a physical system failure as a combination of components failure with associated failure rates, but their analysis can

Chapter 3 – Literature Review

become computationally intensive [3.7]. Raaphorst et al [3.8] described the implementation of an automated fault tree generation algorithm for fault diagnostic purposes. The proposed model was based on an indexed network method where the input nodes were associated to symptoms. Questions were then presented by the system in reaction to existing symptoms. The provided answer could lead to further questions or to the matched fault. New or updated facts would result in an automatic regeneration of the network indexes in order to ensure system efficiency.

A CAD system for fault diagnostics in chemical processes was reported by Kavčič and Juričić [3.9]. Two approaches based on Fault Tree Analysis were introduced. In the first one a fault tree representing the process components considered the deviation of the process variables from their modelled output as an indication of abnormality. For this approach the variable could not be part of a control loop. In such a case, a special template tree was necessary to model the specific variable. Kavčič and Juričić argued that this approach would result in much time-consuming processing, especially when a set of interrelated faults was detected (affecting many different process variables). Therefore, a second approach, identified as an event tree, was presented. This approach consisted of representing the fault symptoms at the root of the tree. Branches were associated with the process variables, and were grouped by component. Dubious process measured variables provided a set of fault candidates, each one associated with a “believe” index. Rules were created to match the fault most likely to be related with the root symptom.

Andrews and Dunnett [3.10] compared the traditional FTA method with a new approach, termed a Binary Decision Diagram (BDD). They argued that traditional methods require considerable computer processing capabilities, particularly in cases of large fault trees. The new method, in which the events were ordered in a way that allows only two paths (true or false) to be followed after each event, was suggested to overcome such limitations.

Hu et al [3.11] employed FTA for fault diagnostic purposes in Flexible Manufacturing Systems (FMS). The developed diagnostic model enabled the description of the fault propagation process in terms of a tree. The system was divided into subsystems, which depending on complexity and modularity factors, could be further divided into

sub-levels. They also proposed the modularisation of the knowledge, in order to improve the reasoning process. To do this a specific knowledge base would only be considered at a specific sub-level. A functional fault symptom was considered as representing the root of the tree. Following the tree structure, functional modules were assessed in terms of the system controllers' signals that were expected to be present at this specific state. Finally, expert knowledge represented in terms of rules, was deployed to suggest the best matching cause and the required actions.

In related work considering a FMS application, Hu et al [3.12] proposed another method. They considered that in such an application, the correct process operation could be characterised by a sequence of states and events. In this approach the process controllers' signals and switches are an important source of data in order to diagnose faults. The resulting approach to diagnostics was based on the acquisition of digital data and analogue parameters and a "diagnostic expert system". The proposed system was based on numerical computation, symbolic reasoning and a management module. Analogue sources were classified in two categories of behaviour: slow and fast changing. Amplitude, variance and mean value were features of interest in the first "slow" category. The fast category considered features of interest to be the changing rate (gradient) and trends of the analysed signals. Thresholds were established for each feature in order to detect abnormal conditions. The system knowledge was structured based on fault trees, control information, condition monitoring and the collection of domain expertise. Fault classification and cause detection was reported to be achieved by searching functional trees. Fault diagnostics was based on linked rules in the knowledge base.

Hu et al [3.13] further reviewed their previous approaches [3.11, 3.12] to concentrate on operational fault diagnostics. Here a sequential model, representing the sequential changes in the machine operating states, was employed to investigate faults based on knowledge of the actual states. A logical diagnostic model was used to provide the fault source indication, by matching the controller's signals against the expected (modelled) states. This approach only considered digital sources. Although an example was described, it was not made very clear whether the signals were retrieved from the controller or monitored using a specific hardware structure, in order to enable the implementation of the sequential diagnostic approach.

3.2.2 – Model Based Approaches

The further development of systems that monitored processes via an indication of states may be considered in the guise of approaches based upon the deployment of Petri-nets. A Petri-net model which integrated parameter trend and fault trees, was proposed by Yang and Liu [3.14] to support early fault detection and isolation. Here fault trees were converted into Petri-nets, making use of their concept to describe the evolution and the state of system degradation. Petri-net places were associated with process parameters and warning levels were used to provide the marking state of these places. The existence of these conditions would lead to events (fired transitions), resulting in a new state in the system description. Threshold conditions associated would therefore provide the level of information required to plan and undertake maintenance activities. Following this approach early detection was provided by the system, which had the capability to issue alarms. Fault isolation would be achieved by the system state description (marking vector). Shutdown capabilities were also supported, in order to prevent further equipment damage, if maintenance had failed to intervene.

Another approach that investigated the use of the Petri-net concept for fault detection and isolation was described by Prickett [3.15] and Davey et al [3.16]. Here the processes were modelled as Petri-nets, with process events characterised as transitions. Faults were detected when the operating time associated with an event was exceeded. Fault isolation could then be performed based on the indication of the process signal that had prevented the event from proceeding within the established time. Such an approach is restricted to discrete signals, although it could also be used to indicate the development of faulty conditions by recording changes in the timed process intervals.

Ajtonyi and Terstánszky [3.17] described fault diagnosis methods based on system models. Their approach considered that, once a process was modelled, the variation of the process parameters, when compared with the models, could be an indication of faults. Another method presented considered the process signals as inputs of a stochastic model, which could thus evaluate faults based on the residual between the model output and the actual physical parameter. Ajtonyi and Terstánszky considered

that such approaches would enable the prediction of faults. Concerns were raised on the requirements of the computational system to provide a real-time response for such implementations. Therefore, they suggested a parallel processing method, which in their view had to take into account each individual application in order to identify the sources of parallelism to define the tasks. Also, arguing that real process models were difficult to obtain, they suggested the modelling of individual process parameters and deployed methods aimed at establishing relations between them.

3.2.3 – Intelligent System Based Approaches

The formulation of expert system rules for a generic fault diagnostic application is considered to be a complex task, especially in cases where process parameters vary in a dynamic way according to the different process settings. It has been suggested that one approach to ease the difficulties experienced with such an application development is to dynamically generate the set of rules for each process task [3.18]. Here the possible faults were defined in terms of fault-trees and symptoms are then correlated to them. Off-line processing was employed to build the rules required for the fault diagnostic process. The required knowledge base would be selected in accordance with the process settings.

Wang et al [3.19] aimed to generate rules from a fault event database, using probabilistic networks. Although presenting a case employing an automatic method, they considered that a semi-automatic approach was more appropriate when large databases were involved. In such a case, domain knowledge was said to be required to define the highest and lowest layers of a decision tree, therefore providing means to orientate the rule construction.

The use of neural networks in this context can be illustrated by Lennox et al [3.20]. They employed neural networks to detect and predict the failure of a melt vessel. They considered this to be the best approach. It was based on the dynamics of the process, which indicated that the vessel's thermal properties were affected by its age. However, they added that the implementation only became possible due to the acquisition of existing previous data records that enabled the neural network training process. Failure prediction was based on the error relation between the measured and

Chapter 3 – Literature Review

the predicted temperature, obtained from the neural network model. The error was then compared against defined thresholds to issue alarms.

Benefits from the use of neural networks for fault diagnostics come from their parallel processing capabilities (providing a quick response to the process measurements), non-linear mapping capabilities, the ability of learning from examples and robustness. Mageed et al [3.21] reported a method employing artificial neural networks in fault detection, isolation and identification. They contended that industrial systems represent a complex environment due to the large number of measurements and possibilities of faults. This could result in oversized networks, making the learning process extremely difficult. Their approach proposed consisted of a two level neural network. The first level was required to provide fault detection and isolation. The second one, using the first level outputs, provided the indication of the different levels of fault in terms of the probability of occurrence of each specific fault mapped to happen.

Fault diagnosis was investigated in order to improve the quality standards in a flash smelting process [3.22]. The presented method was based on neural networks, in the form of Self-Organising Maps (SOM), and heuristic rules. The method described considered specific neurons that were labelled accordingly the process state they most likely represent. Based on an input vector representing the process parameters, the labelled neurons would indicate the existence of the specific state based on probability indices. A rule-based implementation was said to provide process diagnostics based on the state changes detected by the neural network. Messages were issued whenever rules were matched. It was argued that fault detection could be improved by increasing the number of labelled neurons. However, it was recognised that this could also result in an increase in false alarms. Therefore, more precise process measurement and a good knowledge base to provide robust rules would be required.

The use of fuzzy based models was the object of research of Ballé and Isermann [3.23]. Their approach was proposed as an alternative for use with non-linear systems. It consisted of representing a non-linear function as the sum of discrete linear segments. Symptoms were generated on the basis of residuals and ratios between

Chapter 3 – Literature Review

process measurements and the respective model outputs. A knowledge base system, with rules relating symptoms to faults, was employed for fault decision purposes.

Chafi et al [3.24] described a similar approach in which residuals were generated by comparing actual process measurements with the output of the respective parameter model. To overcome the non-linearity characteristics of the process parameters, the models were based on fuzzy clusters. Arguing that residuals might be corrupted by noise, a fuzzy decision method was applied in order to extract the process symptoms. Finally, symptoms were applied as inputs to a neural network to establish a fault confidence index, therefore providing the most probable fault cause.

To summarise, although several different methods for fault diagnostics have been investigated, there is not an agreement about the best one. Geropp [3.25] considered that despite recent research having concentrated on novel methods, such as neural networks and fuzzy logic (or the conjunction of both), it is still required that application particularities be taken into account when selecting the approach to be used. He also argues that to apply such methods, a significant amount of representative data is required for training purposes. The use of knowledge-based systems only becomes possible if enough rules exist to be activated, in order to classify and diagnose faults. Therefore, a good knowledge of the process and the application requirements are necessary to obtain the best results.

3.3 – Condition Monitoring

Condition monitoring represents one of the areas where use of monitoring systems has been widely investigated. Normally, in such applications, the systems are supposed to predict and detect any particular “abnormal” conditions. This information can then be incorporated into attempts aimed at preventing losses that might result from machine damage, the manufacture of poor quality products and from the large downtimes currently needed to recover from faulty states.

3.3.1 – IPPM Centre Research

IPPM researchers have previously considered a wide range of applications of condition monitoring to machine tools and process plants. Sharif and Grosvenor [3.26] produced a review focused on the vital components and measurement techniques to support such application. Temperature, level and flow rate sensors were indicated as the most commonly used in industrial applications. The work proposed that different sensing and transmitting methods should be selected according to cost and accuracy requirements.

Sharif and Grosvenor also included monitoring techniques for vital plant parts. In particular, infrared thermography, Acoustic Emission (AE) and vibration were reviewed as monitoring methods for valves and actuators. A method developed by the same authors that considers historical operating data was also presented. For pumps and motors, measurement methods such as vibration, supply current and temperature were considered. Current and temperature based monitoring was indicated to perform better in enabling the early detection of faults, when compared to vibration. It was suggested that vibration should be used together with one of the other methods, in order to provide accurate fault detection. For some specific applications, the review presented research that obtained good results using AE. In such cases, the signal spectrum tended to change with the development of faults.

In terms of fault diagnosis, the review presented some techniques still under investigation. Knowledge-Based Systems (KBS) and Expert Systems (ES) were included in the category of those that rely on previously assimilated information or existing plant experts' acquired knowledge, to support a rule based diagnosis strategy. The dependency on plant experts to enhance system performance was indicated as one of the methods main drawbacks. Statistical methods, based on properties such as mean value, variance and standard deviation of the measured signal were also reviewed for diagnosis purposes. It was stated that fault development could be detected based on changes on the signals' statistical properties, since process parameters were held unchanged. Model-based techniques, using mathematical models that describe the plant's behaviour, were considered to be of a high level of complexity and therefore it was contended that this approach could result in some

Chapter 3 – Literature Review

errors in systems that do not response linearly. It was suggested the use of the statistical properties of the monitored parameters in order to improve the model results. With the same purpose, investigations using fuzzy logic were also reviewed, as a way to deal with the existing non-linearity. Neural networks were considered as a good method for fault diagnosis, capable of dealing with non-linear systems responses, since sufficient training could be provided. Finally, state transition diagrams (Petri-nets), were indicated for those cases where the process events could be sequentially represented.

3.3.2 – Machine Tool Condition Monitoring

Much of the research undertaken in this area relates directly to machine tools. A review of the approaches for end milling tool monitoring was presented by Prickett and Johns [3.27]. It described the investigation of different sensing techniques, feature extraction methods and decision-making approaches. Indirect measurement methods were considered the most appropriate for tool monitoring, since they allow dynamic assessment, without requiring the stopping of the cutting process. Dynamometer and spindle motor current signals were indicated as the most appropriate in order to assess cutting forces, which were considered as a good parameter to monitor tool wear. For tool breakage detection, vibration measurement was said to give a better indication of such sort of fault. Acoustic emission was also mentioned, but its use was suggested in conjunction with other signals in a multi-sensor approach.

Several references that investigate time series modelling for feature extraction were presented. The method that consists in the analysis of the measured signals over a period of time indicated good results in detecting broken teeth. However, it was indicated as being of limited practical use due to the computational processing time required. A good number of references investigated by Prickett and Johns employed the method of real-time signal monitoring. The method described was based on using threshold conditions of the measured signals to detect the faulty conditions, indicating a better performance in tool wear monitoring. The last method for feature extraction considered was the one based on frequency domain analysis. Although representing a method with increasing interest in the detection of tool breakage, its required processing time was considered to be a constraint, since that sort of fault detection

Chapter 3 – Literature Review

demands immediate reaction to avoid further damage. It was further stated that the method requires a good understanding of the measured signal to identify the process's natural frequencies in order to prevent false alarms.

The same authors also reviewed the use of artificial intelligence as decision-making methods. In some cases, the combination of different methods, such as neuro-fuzzy or neural networks and mathematical modelling were reported to give the best results. Nevertheless, in all cases processing and training time were considerably high. It is finally suggested that better results could be obtained by employing sensor-fusion techniques, providing different views of the same phenomenon in the cutting process.

3.3.2.1 – Sensor-based Systems

The choice of the adequate sensors to provide the best signal is clearly important to improve confidence in sensor-based monitoring systems. Dimla [3.28] provided a review of sensor signals for tool condition monitoring in cutting processes. The subject was considered to be of high complexity due to process dynamics and variations, such as cutting conditions, work piece and tool characteristics. Acoustic Emission (AE) was found to be of use to investigate both tool wear and tool breakage, within different cutting processes. Root Mean Square (RMS) was identified as the most popular method for AE feature extraction purposes. Higher levels of AE were found to be released with tool breakage or fracture, suggesting that AE based monitoring may be used for such purposes. However, AE techniques were also said to be difficult to deploy, mainly due to the consideration of the path followed by the signal. As such its use is suggested as complementary to other techniques.

Methods based on temperature were also reviewed in the same paper. Direct temperature measurements were considered difficult under most circumstances when moving parts etc. make the use of a thermocouple impractical. Therefore, most practical temperature sensing methods were based on non-contact techniques, such as infrared imaging. The review shows that the method could perhaps be applied to detect tool wear. However, temperature distribution was known to be affected by the process dynamics and thus, complicated mathematical models are required. Another method employed for tool wear detection is to monitor cutting forces using a

Chapter 3 – Literature Review

dynamometer. Cutting force signal feature extraction methods have evolved, normally based on time and frequency analysis. Statistic analysis was used in many cases to provide relationships to enable tool wear detection. Although the method seems to provide a good indication of tool wear, difficulties associated with the process dynamics were found. Dimla considers that a good knowledge is required for a good estimation of the static and dynamic forces.

Vibration signatures were also considered in the same review [3.28]. These are said to be difficult to analyse since vibration frequencies and levels can be expected to change due to the many variations in parameters experienced during a normal cutting process. The signal power in the time domain and the power spectrum in the frequency domain were the feature extraction methods mostly employed. The review also indicated that the use of this technique was largely associated with tool wear. Electric motor current and power measurements were reviewed. Some of the features analysed were the signal waveform, peak value, and mean and accumulative sum power. Also, spectral energy fluctuation was said to be an indication of tool wear.

The conclusion of Dimla's review points out that forces and vibration are the most widely used measurement parameters. Nevertheless, the process dynamics imposes severe difficulties in detecting tool faults. Therefore, sensor-fusion is indicated as an alternative to improve condition monitoring systems reliability.

In another example of a process that is considered important, the methods applied to condition monitoring in drilling processes were reviewed by Jantunen [3.29]. Justifying that direct measurements are not very efficient in economical and technical terms, he targeted those methods that indirectly measure sudden failure and tool wear.

Based on the reviewed research, Jantunen placed the measurement methods into three main groups. The first one considered cutting parameters such as the torque, feed force and drift force. This indicated that such signals tend to change with the increasing amounts of tool wear. However factors such as work piece hardness cause similar variations, which can affect the confidence in this measurement method. The second group consisted of vibration and sound measurements. Such methods were considered adequate for rotating machines and were said to be easy to implement in

Chapter 3 – Literature Review

terms of sensor deployment. Vibration measuring was also considered very reliable. Changes in the signal could normally be associated with faults, since process parameters were not changed. The group also included AE and ultrasonic vibration. It was stated that AE sensors should be deployed very close to the work-piece in order to avoid signal attenuation. The use of such methods, especially AE, was thus found to be limited to those cases where signals with very high frequencies had to be measured. The last group focused on spindle motor and axis feed drive currents. Current measurement was found to be easier to implement and gave better results. For the spindle motor these were comparable to the results obtained by measuring the torque.

Jantunen's review then considered the subject of signal analysis. Time domain based methods usually employed by the researchers were the raw signal (real-time) analysis, Root Mean Square (RMS), mean value and peak value. Statistical analysis was in many cases applied to detect tool failure indication. Frequency domain based methods were said to be more sophisticated and found to provide a better picture of the tool condition, for both wear and breakage. The counter point indicated was the required processing power for frequency-based methods, which in many cases made it difficult to produce fast responses for existing or developing failures. Finally, some diagnostic methods applied to the signals and features were considered by the review. The simplest methods were based on the comparison to predefined parameters. An evolution of those methods was the use of thresholds with defined trend limits. Neural networks, although capable of dealing with the process non-linearity, were seen to require huge amounts of data for training purposes, if all process dynamics were to be considered. However, it seems that neural network was the method that provided the best results. Jantunen concluded that satisfactory results were normally achieved when the process parameters (cutting conditions, work-piece) were kept unchanged. Since this cannot normally be assured during real-life processes sensor-fusion was suggested to reduce such dependency.

Turning is another very important manufacturing process that has attracted researchers. Sick [3.30] carried out a review that surveyed 138 publications related to the subject. Only online and indirect methods were considered for both, tool wear and fracture. As a result, forces and vibration were indicated as the process parameters

Chapter 3 – Literature Review

mostly employed, followed by AE. In the particular case of forces, the spindle current measurement was suggested as the preferred method in terms of implementation cost, when compared with specific sensors. AE was indicated as problematic, in cases of parallel processes with several tools.

In his investigation, Sick observed that those researches employing multi-sensor approaches performed the information processing at the tool wear model level. Many different methods were employed for feature extraction. The extracted features were employed as inputs to different types of neural network configurations, therefore providing the model implementation to identify the tool condition.

Sick in his remarks indicated that there was not much agreement about the best features and models to be selected. Just a few researches were said to consider cutting condition in their models. The use of such process dynamics would increase the complexity of the tool condition model, resulting in higher training and processing power requirements. He also considered that multi-sensor approaches could provide better results, by comparing process parameters obtained from different sensor's sources. However, in his view such approach should be applied at earlier stages of the monitoring system in order to provide reliable input information.

3.3.2.2 – Non-sensor Based Monitoring

An approach for condition monitoring based on a machine's existing signals was presented by Prickett and Grosvenor [3.31]. They considered processes that can be described as a set of machine states and events, and can thus be represented in terms of a Petri-net [3.32]. In such a case, the digital levels of the machine controller signals and embedded process sensors are monitored in order to detect events and update states. A method was suggested to monitor machine operation, extract performance information and contribute to the identification of eventual faults. The basis of this approach is that the actual process state and former events could give an indication of the fault cause.

Prickett and Grosvenor also implemented the non-added-sensor method in cutting process monitoring. The existing signals of the cutting machine axis feed driver were

Chapter 3 – Literature Review

used to monitor the tool condition. They argued that the machine controller reacts in order to keep operation close to the process set parameters. Therefore, by for example, monitoring the tachometer of the X axis of a CNC machine, the signal signature would provide evidence to distinguish a healthy tooth from a broken one. Some different cutting conditions were tested, with the method giving an indication of efficiency.

The use of those existing signals, rather than the deployment of special sensors, was presented as an alternative to traditional condition monitoring methods. Arguments in favour were the fact that it does not require process disruption for the implementation, since all signals are already present. Further economic benefits were said obtained due to the high cost of traditional methods.

Although representing an area within which a number of researchers are active, condition monitoring still has to progress. The reviewed investigations indicated that despite the range of different methods employed, process dynamics play an important role when analysis methods are deployed to distinguish a normal from an abnormal condition. The requirement, in some cases, of a rapid reaction in order to immediately detect a faulty state, makes it even more difficult, since cross linked information may not be possible due to time constraints. Therefore, methods and techniques may vary accordingly the application requirements, benefits and cost considerations.

3.4 – Integrated Monitoring Systems

Many factors, ranging from technological developments to consumer demands [3.33, 3.34], have resulted in increase levels of complexity for modern manufacturing processes. The management of such industrial processes requires the use of tools that enable immediate responses to critical events or the capability to intervene in a planned way to reduce as quickly as possible the negative effects of fault or drift conditions.

In the early nineties, Tönshoff et al [3.35] presented what they called a new approach to machine monitoring and diagnosis. The work, which concerned process dynamics

Chapter 3 – Literature Review

stated that monitoring systems should no longer be based on the analysis of individual signals, but should consider the effect of several process variables that together could provide a better picture of the process health. The approach was also concerned with flexibility, proposing the deployment of separate modules for monitoring and diagnostic tasks. A common database was also included, providing the means by which to acquire process knowledge. Although technology has since improved the main concepts pointed out still have relevance to current investigations.

During recent years, the development of online monitoring systems in response to complex process requirements has increased. This has been facilitated by the availability of new hardware and software tools based on computers (PC) and operating systems with graphic support. The implementation time, even in case of complex systems, can now be reduced by putting together parts and components based on or provided with standard libraries. The online system described by Ramakrishna [3.36], for example, used a PC with standard data acquisition cards. This system was used to monitor vibration of hydro turbines in a power plant. The monitored signals provided records that were processed at configurable time intervals, supplying a set of information that allowed the assessment of turbine performance. The application software also provided a schematic display of the turbine and specific parameters. The increased capabilities of modern PCs and associated hardware and software enabled an integrated monitoring system to be deployed. Furthermore, the PC used in the application was installed 100 metres away from the sensing points. However, the use of such a system is often a relatively high cost solution and therefore can find its use restricted to applications where the plant itself is very expensive or where critical events may result in catastrophic situations. Ramakrishna described the system as an online monitoring system, making it clear that all the processing was based on batches of acquired data. It enabled the analysis of the condition of the turbine and the establishment of predictive maintenance schemes. The development of critical or catastrophic events could also be monitored at early stages of their development. Only a true real-time system, based on process events, would be able to detect an unpredicted event immediately when it happens. Depending on the complexity of the application, such system would probably require more resources than those provided by a single computer.

Chapter 3 – Literature Review

In another reported example of an online system, Jeng and Wei [3.37] described the implementation of a condition monitoring and diagnosis system for feed rolls in plate mills. They stressed the importance of such a monitoring system, since, depending on the completed processing and future process requirements, the scheduled replacement of some parts might be inappropriate. By considering the monitored data and production settings, such as rolling speed and load, the system provided means to assess the condition of parts and thus support decisions related to part replacement tasks. This had the effect of reducing maintenance costs, enabled programmed interruptions of the production line and helped to reduce the waste of materials. They further described the system's capability to utilise specific settings such as filters of a specific range of frequencies, thus simplifying data analysis and fault diagnosis methods. The derived parameters, in certain operating conditions, might then indicate the development of a faulty condition. However, they made it clear that a good knowledge of the monitored system or process was fundamental. The system was based on specific sensors applied to the process. A front end employing industrial PC hardware was used for process signal conditioning and data acquisition. The front end was provided with a network card enabling monitored data to be transferred to an office computer, where data analysis and associated fault diagnosis could be continuously displayed. Figure 3.1 shows the system diagram.

In this system tasks were split between the two PC's. At the shop floor end a robust device was deployed. A PC with more graphical, computational and software tools, that could also be interfaced with production planning and maintenance management systems, was used at the office-based location. Such a configuration increased the overall capability and reliability and made the updating of any individual module much easier. Despite all the reported benefits, no cost details were outlined. It is fairly obvious that the cost of such an implementation can become high, especially if it is considered that only a specific part of the whole process is monitored. This must however be set against the previously raised considerations of plant cost and the possible impact of breakdowns.

The deployment of a system that covers a single part of a process may also be feasible in cases where this part may represent a production bottleneck. Therefore, a comparison between investments and the resulting benefits should be considered on

the basis of the cost to the organisation as a whole of any process related failures. This must also consider the way in which process reliability and product quality can be enhanced by timely actions, such as can be made by real-time integrated monitoring systems.

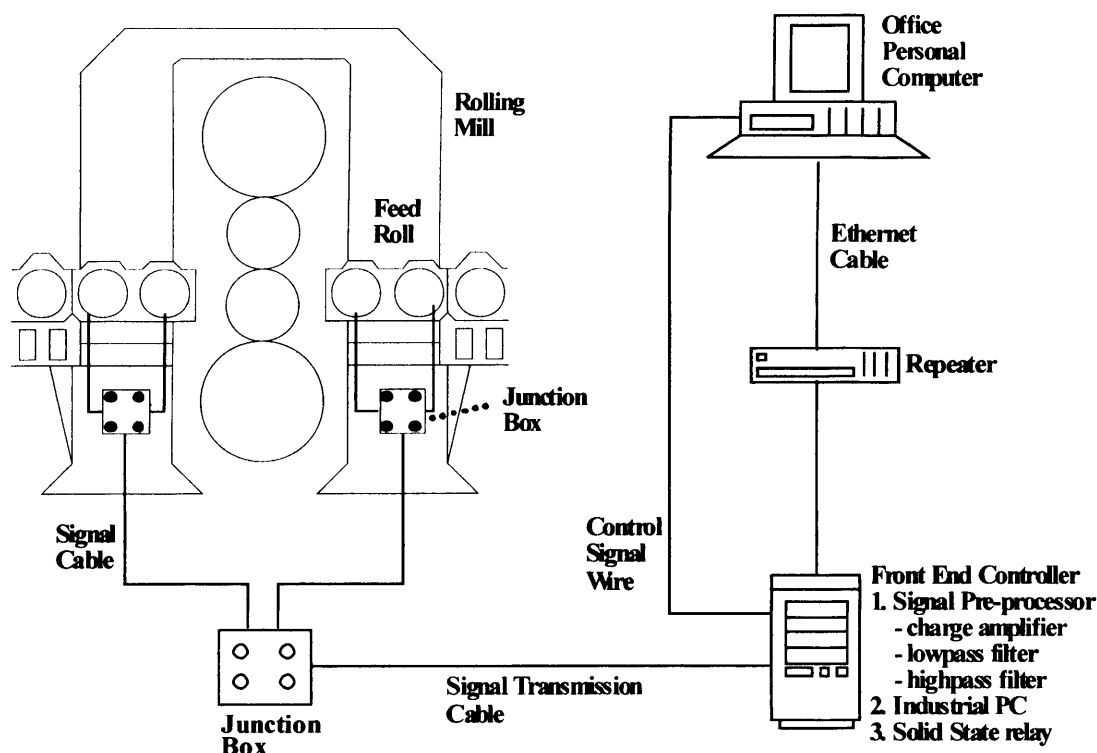


Figure 3.1: Monitoring and diagnosis system for feed rolls [3.37].

Alexandru [3.38] reports the implementation of a real-time system. She discussed the requirements of supervisory functions in modern systems, as they relate to increasing complexity and growing production demands. Fault detection and isolation were considered essential to maintain performance levels of such industrial applications. Alexandru thus suggested an online and real time system focused on fault tolerance and diagnosis, based on early detection and “fault time behaviour”. To achieve the required results, she proposed the implementation of three methods. The first considered the measured signals (which although providing ways of detecting faults, were considered inappropriate for early detection of faults). The second method used mathematical models that represented the physical parameters of the process. It allowed comparison of the current system behaviour with established fault dictionaries. The modelling of the entire system’s “ideal” behaviour and fault states

was described as being very hard to achieve, particularly when considering large and complex systems. The third method proposed as a complement to the former methods, was based on a description of rules and facts obtained from human observation. This method was therefore potentially capable of describing the system's behaviour under initially unknown conditions. It was suggested that such method enabled the use of system symptoms, rather than just output signals, to identify faulty conditions. Intelligent decision making techniques were then required to perform the system evaluation.

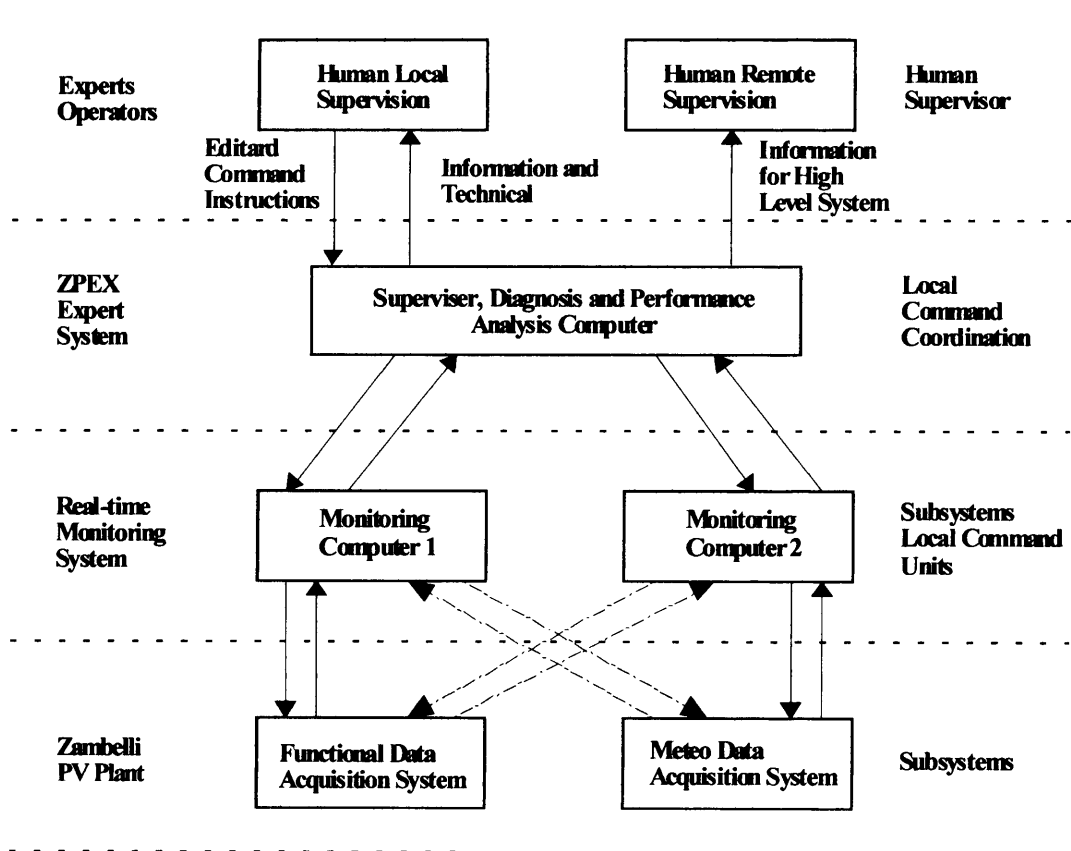


Figure 3.2: Architecture of the fault detection and diagnosis system [3.38].

A schematic diagram of the system implemented in [3.38] is shown in Figure 3.2. Considering the different methods employed in the system implementation, it is seen that considerable computational resources were required, especially when the real-time nature of the system is taken into account. User interface, data acquisition, databases and expert system functions would better fulfil their purpose if based on a

Chapter 3 – Literature Review

distributed structure that could provide each part with the required resources. Such an implementation, if totally based on PC hardware, could become very costly and thus such applications are usually restricted to very specific processes.

As a rule based system, inclusions of new rules that describe unknown system states were required. A system domain expert may be required to input a reasonably large initial number of rules or to later acquire the knowledge from the system behaviour based on its real operation. This can result in a long term learning process, if little or no initial knowledge is available. The system could benefit from a shared knowledge base, constructed from the knowledge of other existing systems with similar behaviour. This would require a distributed environment where databases could be shared, speeding up the process of acquiring system behaviour knowledge. This secondary effect, with the sharing resources, could result in an overall system cost reduction. Depending on the application, cost may not be the most important issue, but the rapid use of the system, exploring its full capabilities, is certainly very important, especially in cases of complex and strategic applications.

Many other systems have been developed to provide application specific support for fault detection and isolation. For example, Angeli [3.39] presented an online expert system for fault diagnosis in hydraulic systems. The implemented system provided data acquisition, data processing and expert system capabilities. Hardware and software specific developments were made by considering the application requirements. A diagram of the system components and their relations is shown in Figure 3.3. Digital and analogue sensors captured the process signals that were then used in calculations based on the mathematical model of the system, thus providing information that enabled the recognition of system faults or unacceptable process deviations. Angeli argued that it could be considered almost impossible to create a mathematical model that completely described the system's behaviour. Therefore, the method used acquired experimental knowledge to compensate for the mathematical model constraints.

The scientific model produced was employed for detection, prediction and fault compensation, while experimental knowledge was used to isolate and diagnose faults. The comparison between the system measurements and the results of the

mathematical model was indicated as a way to detect faulty conditions. Fault prediction was achieved by comparing the results of the measured signals applied to the mathematical model against existing information in the knowledge base. Fault compensation was presented as a way used by the expert system to provide parameter compensation to certain worn parts of the process, overcoming efficiency losses. To benefit from this approach, a mathematical model of the specific part was required to calculate the appropriate compensation factor. Fault diagnosis was based on a knowledge acquisition system, using a qualitative model to provide diagnosis based on reasoning procedures and thus making it possible to associate faults with symbolic language.

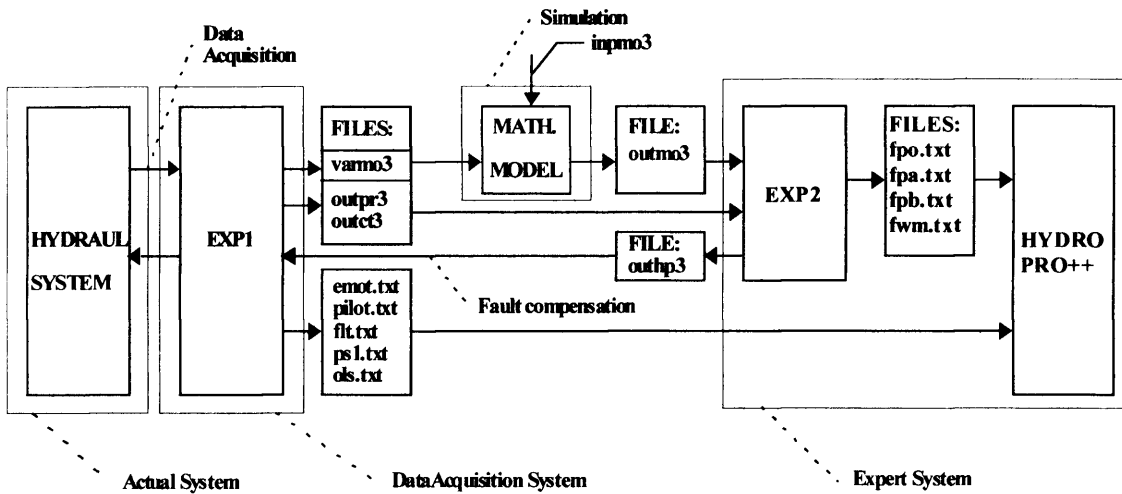


Figure 3.3: The interaction scheme between the modules [3.39]

Angeli also made it clear that “scientific knowledge of model-based systems cannot cover the whole range of diagnostic tasks since diagnostic activity is mainly based on experience”. Thus, the integration of both methods, scientific and experimental, should be used to maximise results. Reasoning was provided by the use of rules that are generated and grouped in topics that might be correlated in terms of sub-problems. It was recognised by the Angeli that fast responses were critical due to several tasks being executed, besides the processing requirements of a reasoning system based on knowledge bases.

Chapter 3 – Literature Review

It is true to say that the implementation of this system if based on a distributed structure, such as those outline in the next section, could offer an alternative and would improve efficiency. Nevertheless, it seemed that several functions and parameters were intrinsic to the application. For instance, there were no references of means that enable the adjustment of parameters in the mathematical model, despite of using a second method to complement this one. System replaced parts might behave in a slightly different manner and adjustments might be required. Some dynamic factors, such as production setting, could in some cases result in a different behaviour, conflicting with faulty conditions previously established. The design of such a system should consider the flexibility of dynamic factors, thus increasing its potential benefits.

3.5 – Distributed Monitoring Systems

Splitting the modules outlined above and executing them on separate sites provided with features that would be specific to the required task would allow greater flexibility and support more effective monitoring functions. Further, such a deployment could enhance the acquisition of the knowledge base, since several systems could provide information resulted from individual experiences. This is the basis of distributed monitoring systems.

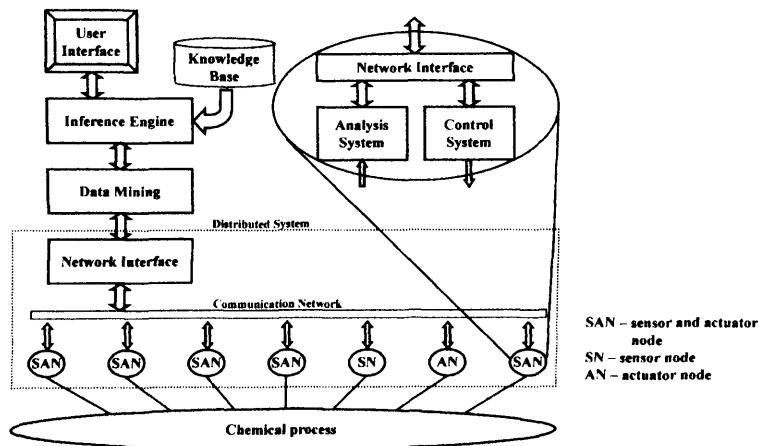
Distributed expert systems were presented by Bonastre and Peris [3.40] as an important trend in the monitoring and control of chemical analysis process. They considered that computers are advancing rapidly both in terms of technological developments and in the reduction of cost. As a further benefit of these developments, it was suggested that chemical instrumentation would soon incorporate data processing capabilities that, together with communication networks support, would allow the implementation of local algorithms for data acquisition and analysis. This is seen as enabling in the future the deployment of distributed structures that could enhance the systems capabilities. It was proposed that centralised programming would be the next step in the integration of intelligent devices. This is to be undertaken in a user-transparent way, making it an important part of the basis of distributed systems.

Chapter 3 – Literature Review

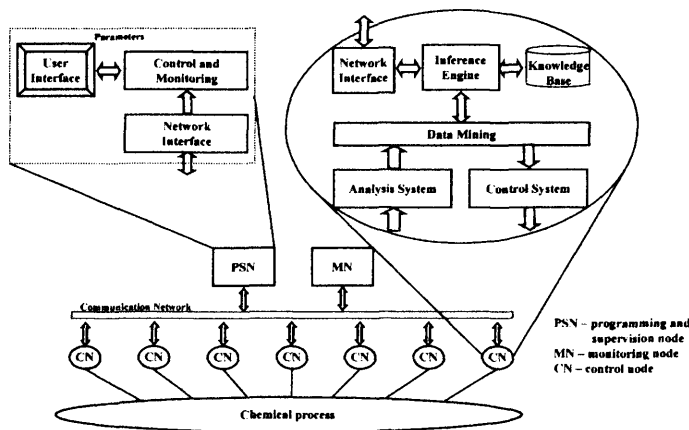
Distributed Expert Systems (DES) were defined by Bonastre and Peris as being a technique that combines intelligent instrumentation with monitoring and analysis systems, using local area networks to provide geographic location independence, all in a user transparent environment. The simplest distributed topology was described as that one where data acquisition nodes are autonomous and provide the expert system with raw data (Figure 3.4a). The reasoning task is carried out by a centralised expert node. In such a case, the data acquisition nodes would be provided with the measurement algorithms and network protocols to carry out communication. Bonastre and Peris argued that although the network could represent a bottleneck, existing technology would make it feasible, especially for simple systems applications.

The following suggested approach considers that the measurement nodes might have some sort of previous processing, participating in running the expert system (Figure 3.4b). In such case, a considerable level of complexity would be transferred to these so called Control Nodes (CN). As result, flexibility and modularity would be aggregated to the system, representing considerable fewer constraints when enlarging it. In the specific case of this model, only relevant data would be sent by the measurement nodes, reaching either the central expert node or any of the distributed control nodes that might make further use of supplied data.

Bonastre and Peris [3.40] considered at this point that a certain level of complexity would be brought forward by the use of distributed systems based on nodes provided with local processing capabilities (Figure 3.4b). Programming an expert system based on multiple nodes, some perhaps different from each other, could become a difficult task. To overcome this challenge, centralised programming was proposed, with definitions and tests performed considering an application, in which the distributed nodes would appear as virtual devices. Afterwards, a separate task had to be executed to distribute the rules among the nodes. It might be considered that, although described by the Bonastre and Peris as an implementation for chemical processes, such a system could be applied to many other areas where flexibility is a requirement. Further benefits might be achieved if the system proposed would be able to interface to other manufacturing functions, therefore providing more flexibility by considering the process dynamics.



(a)



(b)

Figure 3.4: Expert system with (a) distributed data acquisition and (b) fully distributed expert system [3.40].

In another reference to distributed systems, Manders et al [3.41] considered the importance of Fault Detection and Isolation (FDI) strategies. As part of supervisory and control functions of complex engineering systems these can improve safety and functionality. They suggested that the conventional techniques, based on hardware redundancy and localised hardware safety mechanisms would not provide a good result with complex systems. In such cases the process dynamics should be considered in the fault detection and isolation procedures. Distributed Measurement and Control (DMC) was advocated as a way to enrich complex systems supervisory

Chapter 3 – Literature Review

and control tasks, especially when considering new technologies such as networked smart transducers that can benefit equally fault detection and isolation. Having in mind these aspects, they described a system for online fault detection and isolation of a multitank fluid system. Smart transducer technology was employed to provide a distributed measurement and control structure. It was made clear that the smart sensors used in the experiment were based on the IEEE 1451 standard. This standard is reviewed elsewhere [3.42, 3.43].

Their design was said to be one in which model-based qualitative fault isolation was applied, requiring a signal-to-symbol transformation. This system was passed upon a process by which continuously sampled measurement data was computed in a symbolic form. Fault isolation based on the symbolic description was separated from the signal-to-symbol transformation. They stated that one of the goals in building the distributed application was realising the symbol generation on the transducer node itself. This was seen as potentially a way to considerably reduce the network load of monitoring and supervision tasks.

The transducer implementation in [3.41] was based on the available parts of the IEEE 1451 standard. It implemented the Smart Transducer Interface Module (STIM) and Network Capable Application (NCAP). STIM was based on a standard microcontroller and an embedded Ethernet controller was used to implement the NCAP. A publish-subscribe mechanism was implemented, based on IP/multicast [3.44]. The distributed measurement nodes published their data, thus reporting the sensor's signals. The system architecture diagram is shown in Figure 3.5. The experiment reported the implementation of nodes with control functions embedded. This task would automatically be executed by each distributed node, reacting in response to the node measurements. The main node, responsible for the supervisory task, subscribed the published measurement data of each distributed node. The temporal causal graph concept [3.45] was used to model the process dynamic characteristics in the fault detection and isolation algorithm. Fault detection was based upon the difference between observed and predicted system behaviour. The residual was used to generate hypothesis using temporal causal graphs. A set of possible faults was generated, each one with an associated predicted behaviour. The method was said

to be applicable to those components that could be modelled as parameters and qualitatively estimated.

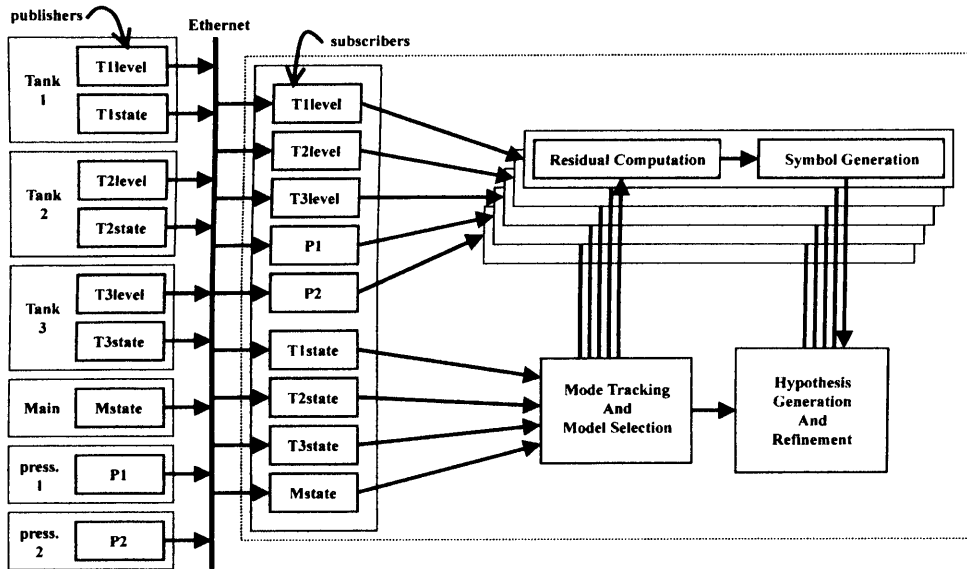


Figure 3.5: FDI application architecture [3.41].

Manders et al [3.41] also indicated the existence of system limitations, most of them related to the network interface, imposing restrictions to the system overall capabilities. Although Ethernet is one of the most popular network standards in use, there have been many considerations about its use for industrial applications, especially when control tasks are focused [3.46]. The use of smart devices deployed in a distributed environment seems to be the next step in the development of monitoring and control technologies. This will provide flexibility and efficiency based on shorter development time and integration facilities, resulting in lower cost of implementations and consequently stimulating the use of these techniques in a larger scale.

An intelligent online monitoring system for end milling has been developed by Tseng and Chou [3.47]. It was presented as an effort to provide lower cost monitoring tools, with the benefits obtained through the latest research in the subject area and thus stimulate the use of intelligent monitoring techniques. This work aimed to implement a tool to help to reduce the waste resulting from low quality cutting processes due to tool breakages and wear problems. The proposed system required a dedicated PC to

interface the milling machine and host the intelligent monitoring application. Rule based techniques were employed for reasoning purposes, providing a high-level explanation mechanism to describe the development of faulty conditions. Although requiring specific hardware development to interface to the CNC machine electrical levels by the computer, it was said to explore the existing process information available in such machines by communicating throughout standard interfaces in the deployed Direct Numerical Control (DNC) modules thus avoiding the use of further sensors. Figure 3.6 shows details of the implementation.

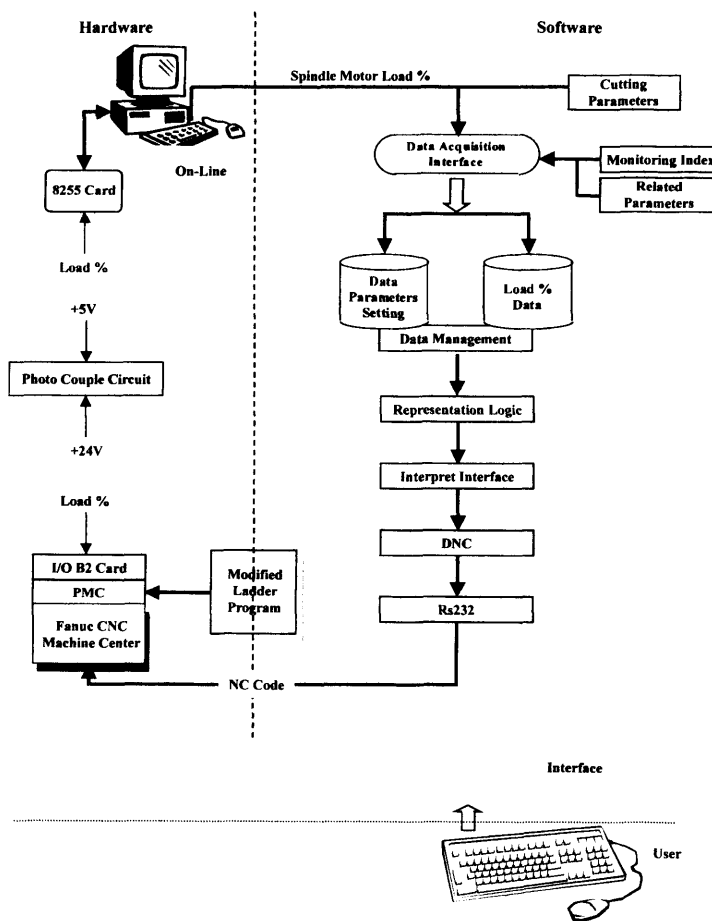


Figure 3.6: System architecture and functions [3.47].

Among the other benefits of the implementation was the capability of setting up a remote control room. In their paper Tseng and Chou suggested that it could be easily implemented by providing the system with Internet connectivity. With this in mind the software development was based upon some de facto standards such as a PC computer, and the Microsoft Windows operating system, programming tools and

Chapter 3 – Literature Review

libraries. Since cost is one of the identified issues that have previously limited the take-up of this type of system, it is worth considering how this approach can be improved with the development of dedicated hardware to interface to the process machine, thus avoiding the use of a PC. Such implementation should enable the processed data to be delivered to a central system. This central system could then provide the same functionality to several machines, implementing the distributed system concept. A further enhancement to this approach would be to implement truly distributed systems using the Internet as the basis for data transfer and communication.

The use of web-based technology to help manufacturers with organisational challenges, such as geographically spread out manufacturing plants, was presented by Ong et al [3.48]. Information was identified in this work as a strategic resource that becomes essential in such a situation. The employment of monitoring tools based on Internet technologies is a way by which manufacturing activities in many regions and even in different countries can be integrated and monitored. Among many benefits cited, perhaps the most relevant here is that the performance of a machine or process can be monitored and accessed from anywhere in the organisation. It was also proposed that information of productivity, diagnosis and staff training on the effective operation of manufacturing systems could be shared among partners at different locations. Internet technology was praised by Ong et al due to its rapid development, and its capacity of providing access to the most remote locations all over the world.

They [3.48] did not however support the idea of client/server architectures where a web-page server is integrated at the machine side and remote diagnosis is carried out using normal web browsers. This kind of implementation was criticised from the point of view of the timing involved, when considering remote diagnosis. They suggest the use of an agent technology, based on a peer-to-peer protocol. In this way each agent can be focused on a specific engineering project application. Agents can also initiate a request to other agents and carry out transactions with each other. Such capability was indicated as being fundamental to enable the implementation of a central knowledge base. It was considered that knowledge bases used in diagnostic systems were normally rule based. To provide such knowledge base with a sufficient number of rules to enable an efficient diagnostic analysis, a considerable amount of

time might be required. A central knowledge base, shared by different users at different locations, could be fed with the required knowledge and consequently provide an efficient diagnostic analysis, in much less time.

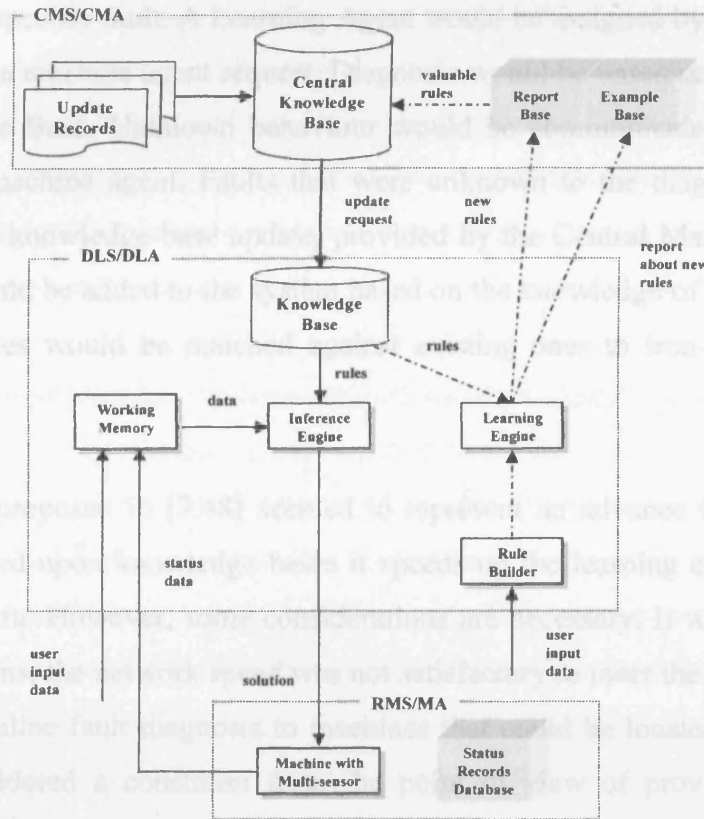


Figure 3.7: System architecture and agent components [3.48].

The system architecture proposed in [3.48] was therefore based upon a multi-agent structure (Figure 3.7). A Diagnostic and Learning Agent (DLA) is centrally positioned to facilitate the use of the knowledge base by remote users as they carry out diagnosis procedures. In this way the established system “learns” from faults that are detected and diagnosed at all sites. The Central Management Agent (CMA) updates the knowledge information in the Central Knowledge Base (CKB). The Machine Agent (MA) monitored the machine operation. All these agents cooperate to enable remote monitoring and fault diagnosis, and on-line knowledge acquisition. Fault diagnosis in the context of this system was based on the Central Management System (CMS), the Diagnosis and Learning System (DLS) and the Remote Machine Site (RMS). The tasks of monitoring and diagnosis were split between the Machine

Chapter 3 – Literature Review

Agent and the Diagnostic and Learning Agent. Once they have been “learned”, new events were sent to the Central Management Agent and then become available to all the other users sharing the same central knowledge base. Confidence factors were implemented as a way of assigning weights associated to each one of the actions taken to diagnose a specific fault. A Learning Agent would be assigned by the central agent in response to a machine agent request. Diagnosis would be based on existing rules in the Knowledge Base. Unknown behaviour would be communicated in terms of an alarm to the machine agent. Faults that were unknown to the diagnosis task might require a local knowledge base update, provided by the Central Management Agent. New rules should be added to the system based on the knowledge of a domain expert. These new rules would be matched against existing ones to iron-out any kind of conflict.

The structure proposed in [3.48] seemed to represent an advance in terms of fault diagnosis. Based upon knowledge bases it speeds up the learning curve of the fault diagnosis system. However, some considerations are necessary. It was indicated that in practical terms, the network speed was not satisfactory to meet the system demands in providing online fault diagnosis to machines that could be located far away. This could be considered a constraint from the point of view of providing immediate reaction to the development of critical faults. In the author’s opinion an alternative to minimize such problem could be providing the local agent with more capabilities in terms of abnormal behaviour detection, which in the case of the proposed work relied completely on the Diagnostic and Learning Agent. Nevertheless, the approach might have further application in the detection of long term fault development and process parameter degradation. In this way it might represent a useful tool to machine manufacturers to support their customers by deploying predictive maintenance actions that should be taken. Potential also exists for using it as an information base to improve future machine designs.

To achieve such goal, further investigation should be made to reduce cost requirements for the necessary agents, possibly enabling the machine agent to be integrated within the machine at the remote site. Another way forward would be to reduce the cost of the various agents, particularly the machine agents, by perhaps deploying microcontroller technology.

3.6 – Embedded and Microcontroller-based Monitoring Systems

Compact and powerful devices are becoming available as a result of the continuous development in the electronics industry, providing further options for the implementation of monitoring systems. However, such implementations should take into account the requirement of modern systems, which should not exist in isolation. Modern and efficient fault diagnosis techniques still require powerful hardware and software tools. A distributed system can provide a good balance, with efficient and low-cost devices at the measurement points and powerful and friendly to use software applications shared at higher levels.

The use of Digital Signal Processors (DSP) for fault detection was presented by Baccigalupi et al [3.49]. Online monitoring and diagnosis of electrical and electronic components was implemented using two DSPs, one for measurement purposes and the second one for fault diagnosis. The fault detection approach was based on the comparison between measured and nominal quantities. It was considered that the implementation of the system should take into account two main problems. The first was the computational requirements of an online monitoring and diagnosis system that demands a high rate of measurement updates. The second concerned external factors, such as temperature, that could induce a false diagnostic. Dedicated processors for each proposed task dealt with the first problem, while the second one was claimed to be solved by implementing repetitive signal sampling when an abnormal quantity is detected. Whenever a fault was detected, a flag was sent to a PC connected to the apparatus. The implementation of a fault detection algorithm that uses mathematical manipulations was facilitated by the use of a DSP, a device provided with a specific set of instructions for such a purpose. In the author's opinion, a limitation could be the incapability of varying the model parameters dynamically.

In their case study Roberts et al [3.50] raised questions concerned with the cost of dedicated monitoring systems for fault diagnosis purposes, in cases where multiple electro-mechanical assets in the manufacturing system require such a resource. They proposed the implementation of distributed computational intelligence. It suggested that quantitative fault detection should be carried out at the asset (or machine) level. The data should be passed across a network, enabling remote diagnosis based on

Chapter 3 – Literature Review

sophisticated analysis tools, in an adequate processing environment. The system architecture would have Fieldbus nodes at the assets level, equipped with a transducer. A local embedded processor would receive the Fieldbus nodes data and apply fault detection algorithms. The assessment of residuals of the comparison between measured values and modelled behaviour of different parameters would be used to detect the existence of a fault condition. Such a condition would be transmitted to a higher level where a PC implementing a qualitative linguistic rule based diagnostic system would provide fault isolation based on the parameter residuals. Through a communication link, reports could then be made available at a management layer. The use of a distributed structure employing dedicated devices such as microcontrollers and embedded processors helped to achieve the main objective, which was to conceive a low-cost system. Nevertheless, it was said that all the monitored applications (Fieldbus nodes) were required to be of the same type, since the local processing node (embedded processor) was provided with a single fault detection algorithm. A suggestion to improve this structure could be providing each Fieldbus node with its own fault detection capability. This would make it possible to have assets with different behaviours monitored by the same system.

The use of dedicated processors applied to process and condition monitoring was further investigated by Baek et al [3.51]. They introduced a monitoring system based on a DSP for real time monitoring of tool failure in a milling process. The system diagram is shown in Figure 3.8. Real time was stated as especially important for such application, since tool breakage must, if possible, be monitored in real-time. The system implemented two neural networks embedded in the DSP, to monitor tool breakage and tool chipping states. The DSP was required due to its capability in processing mathematical functions based on specific instructions, instead of requiring software routines for such a purpose. The DSP parallel processing capability was equally important, resulting in a more time efficient implementation.

The developed monitoring system employed the pattern classifier concept to decide on the tool condition. Neural networks with back propagation were implemented to function as the pattern classifier. The neural network inputs were obtained from features extracted from the process cutting force signals. An Auto-Regressive (AR) model was used as the signal-processing algorithm. This specific model was chosen

due to its lower time requirements when compared with other signal processing algorithms. Individual neural networks were implemented for each monitored condition, tool breakage and chipping. The tool conditions would be classified as “normal” or “abnormal” and these states indicated by a LED and on a connected PC computer display. This implementation showed the growing capabilities provided by new compact devices, such as a DSP and embedded microcontrollers. The possibility of developing neural networks embedded in the processor seemed to be an indication of such devices capabilities. Nevertheless, they agreed that processing time was critical, causing data buffer overflow in some cases, depending on the operation speed. Therefore, it seems that although all the increasing processing power provided by these new generation of devices, there are still tasks that should be shifted to appropriate environments such as a computer. Such a system would then benefit from greater processing power, adaptability to the demands and software tools that provide high level of flexibility.

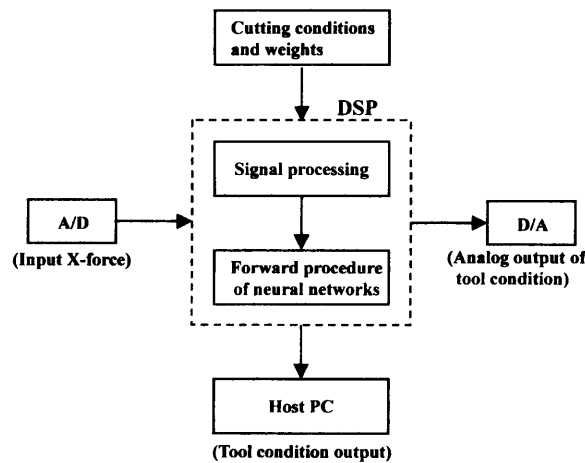


Figure 3.8: DSP based real time monitoring system [3.51].

Microcontrollers were also employed for measurement and control purposes in a distributed system configuration investigated by Bolic et al [3.52]. They suggested that PC computers represent the best choice for the central node in distributed systems, due to the wide range of hardware and software resources available for measurement and control implementation. Nevertheless, the use of microcontrollers was defended in cases where instead of the processor power such as a PC computer, the requirements were for small, low-cost and robust devices. Bearing in mind such

points, they described the implementation of a distributed system where the individual hardware modules were based on 8 bit microcontrollers that were provided with communication interfaces to enable them to connect together. The system was implemented with an application program generation and distributed units configured from a central unit by entering the required parameters. The central unit, also based on an 8 bits microcontroller, provided a user interface and arbitrated the tasks among the distributed units. Different implementations of the distributed units were provided, accordingly the devices they should support. Figure 3.9 shows a diagram of their distributed system implementation.

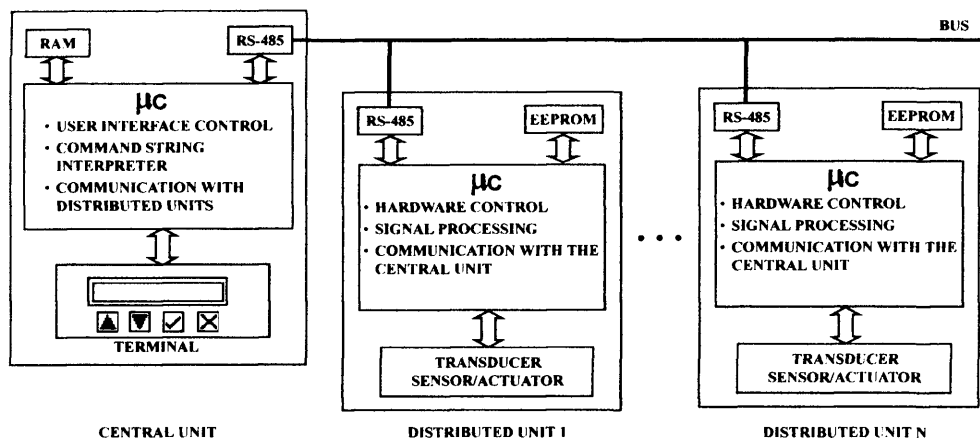


Figure 3.9: Distributed system block diagram [3.52].

The system described in [3.52] used a 3 layer (physical, transport and application) data communication protocol, identified as a “reduced OSI model”. The central unit would control the system communication by sending broadcast and addressed commands. These commands would provide to or request parameters from the distributed unit. Although presented as a system with automatic program generation, such a feature seemed to represent an automatic selection from the configuration menu in the central unit, based on the distributed units’ characteristics. The system description suggested its capability of performing distributed control tasks. However, it was not made clear if the system was capable of dealing with dynamic conditions, where the distributed units might need to have their parameters updated. Despite recognising earlier that a PC computer could be a better choice for a central unit in a distributed environment, an 8 bit microcontroller implementation was used to perform

Chapter 3 – Literature Review

this task. Such an implementation, especially when considering the master/slave hierarchy, could limit the system capabilities. However, the described system showed that for many applications where size, power consumption and cost are issues, the new generation of microcontrollers could provide an adequate answer. A better performance might be obtained if, rather than developing a communication protocol, one of the existing Fieldbus standards were used. Provided that there are such device controllers already available this will have the effect of, easing software implementation and improving processing performance.

The importance of data communication for the development of distributed systems was recognised by de Frutos and Giron-Sierra [3.53]. They described the implementation of a distributed control system that exploits the power of an embedded PC, enhanced by the use of existing and well-developed software tools that ease such developments. The system was provided with an interactive graphic tool to build control and measurement functions, that can be implemented in distributed nodes. A full range of control and measurement functions for analogue and digital I/Os were made available. The system architecture was provided with a supervisory node, based on a PC, centralising the distributed nodes knowledge base. Data communication was implemented using modems and standard telephone lines, since it was considered that the distributed nodes might be located at far away locations. Each distributed node supported 3 basic tasks: communication, data acquisition and control. Learning capabilities were provided, enabling the distributed nodes to retrieve information from the supervisory module. Data update was periodically requested to the distributed nodes by the supervisory node. Figure 3.10 illustrates the system architecture and software components.

An argument made in [3.53] was that embedded PCs represent a low-cost, whilst powerful solution, in cases where distributed nodes required greater processing capabilities. The high scale of integration of the current generation of electronic components enabled such an implementation. Existing software tools and standards helped to ease the development of such systems provided with graphical capabilities. However, it is the author's opinion that the system described by de Frutos and Giron-Sierra could equally have been developed, probably in a more flexible way, using commercially available tools [3.54].

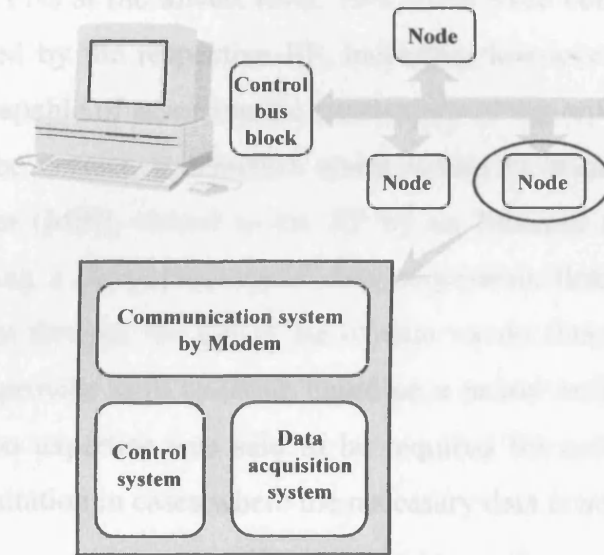


Figure 3.10: System architecture [3.53].

In attempting to develop the next generation of monitoring systems it is important to be aware of current and future research and development in the area of system architecture and possible related developments.

3.7 – Future Directions

An architecture for Distributed Fault Detection and Isolation (FDI) was proposed by Dassanayake et al [3.55]. The system uses industrial network standards (Fieldbus) in its implementation to provide distributed monitoring and control capabilities for assets in buildings and manufacturing processes. Industrial networks, rather than conventional ones, were chosen since such standards consider the requirements of distributed control applications. The proposed architecture was based on 3 layers or levels, as seen in Figure 3.11.

The lowest layer was identified as the Fieldbus Node (FN). This node used a microcontroller to implement the Fieldbus communication protocol. Measurement and control capabilities were provided by a DSP. At the middle layer, an Embedded Processor (EP) provided fault detection to the connected FNs that had common operating characteristics. Thus EPs had built-in Fieldbus interfaces to enable them to

communicate with FNs at the lowest level. FNs could issue control alarms, based on knowledge provided by the respective EP, indicating low-level fault detection. The EP was therefore capable of assessing the consistency of the reported fault. Whenever it was proven to be correct, a detection alarm would be passed to a Management Information System (MIS), linked to the EP by an Ethernet network. The MIS is capable of activating a direct high-speed data acquisition link, routing the FP that generated the alarm through the EP. A set of data would thus be transferred to the MIS, which could provide fault isolation based on a neural network implementation. Previous application expertise was said to be required for network training, which may represent a limitation in cases where the necessary data is not available.

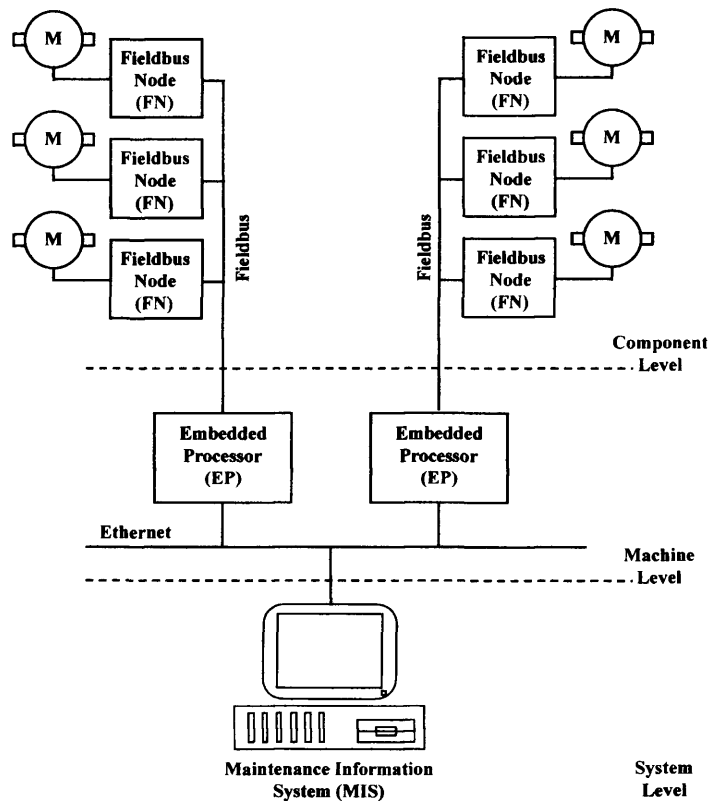


Figure 3.11: Proposed architecture for system-wide FDI [3.55].

In this proposed approach [3.55], fault detection was implemented based on the comparison between the nominal behaviour and system measurements, taking into consideration a predefined tolerance band. A detected fault would result in a FN data request. The data is compared to models describing the individual parameter's behaviour. The resulting vector, with each specific parameter component, would be

Chapter 3 – Literature Review

used as an input to the system's neural network, thus providing fault isolation. The proposed work showed the flexibility of a distributed system. Devices with less processing capabilities were deployed closer to the measurement or control points, reducing the implementation cost and improving system reliability. At the same time, processing capabilities at lower levels could reduce the load of the upper layers. It would enable the possibility of providing specialised services or functions to a larger number of devices deployed at a lower level, keeping the low-cost strategy with an increasing processing capability. At the highest level, powerful hardware and software deployment could provide the processing power and flexibility required for an efficient use of artificial intelligence tools. The use of adequate communication networks in accordance with the environment and application requirements were seen to be equally important in improving system robustness and easing implementation. However, this implementation required all devices at the Fieldbus level to be measurement or control devices with the same characteristics. Such requirement seems to represent a constraint, limiting the system use. In cases where assets with different characteristics might be present, several separate systems would be required, reducing flexibility.

With the purpose of providing a base for the establishment of standards, some research has been conducted to investigate the requirements of data acquisition systems based upon distributed architecture. Work conducted by Ehrlich et al [3.56] investigated and proposed a generic model for the deployment of smart sensors. In this context, such devices were considered for carrying out conditioning, digitisation and processing of signals delivered by transducers. Such tasks could therefore be executed near to the measurement points, increasing processing power compared with a centralised Data Acquisition System (DAS). A wider coverage area could equally be achieved by employing industrial networks (Fieldbus) to connect together all system components.

This investigation [3.56] suggested that the key-point in using smart sensors consisted in their ability in responding to other system components requests. A system employing such technology would be enabled to embody a higher level of versatility, since the system components could be dynamically reconfigured in response to new requirements. The work proposed a graphical software tool to support the

implementation of the distributed data acquisition system, identified as the “instrumentation plan”, in a generic way. The system representation was said achieved using Data Dependence Graphs (DDG), illustrated in Figure 3.12, which were supposed to represent specific levels of system information. In the described approach, the smart sensors would provide signal conditioning, data processing that takes into account local and external data (the latest provided by other system components) and communication capabilities. A system controller was added to provide a user interface, data storage and to allow smart sensors management based on the system diagram. Data communication was based on Fieldbus technology, enabling system components to be connected together and to share data resources.

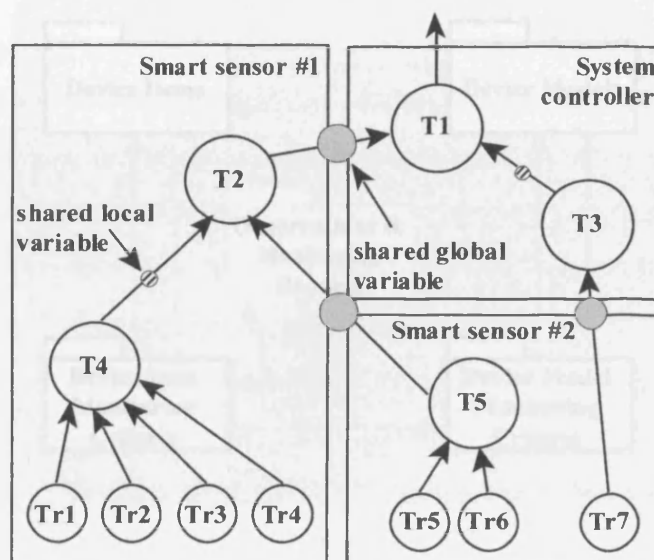


Figure 3.12: Data dependence graphs (DDG) for instrumentation plan [3.56].

The model proposed in [3.56] was supposed to increase the systems data processing capabilities by providing parallel processing and allowing system variables to be shared at a local level. The work did not reference to existing standards for the covered area. To ensure the continuity and compatibility of the proposed model, the smart sensor implementation should take into account standards such as IEEE 1451 [3.42, 3.43]. Another important aspect of such an implementation should concern with the Fieldbus standard employed at the communication level of the system. The model suggested the broadcast of global variables in an asynchronous way, based on events. This would require a network specification that could support such an

implementation. The existing standards in the field should be analysed to select the most appropriate [3.57].

Nieva and Wegmann [3.58] proposed a model of a conceptual DAS. It was proposed in order to support system designers with the necessary levels of abstraction for the development of DASs, either when based on existing standards or design specific. It was identified as a conceptual model of a generic system and was said to represent a formal description of a system. The approach was also described as an easier way to understand a system, since it focused on the main aspects, while hiding the low-level details that might be difficult to comprehend.

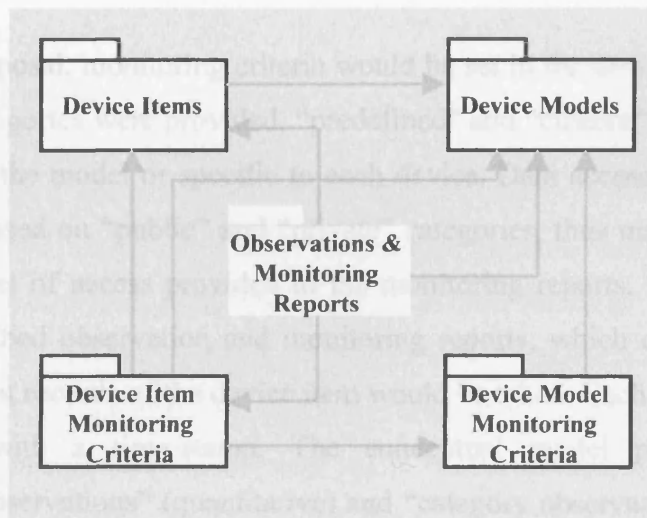


Figure 3.13: Conceptual model main components [3.58].

Figure 3.13 illustrates the main components of the generic DAS conceptual model, as described in [3.58]. In the proposed model, device models were presented as describing the model representing a set of real world devices, defining any measurement points with their respective measurement type and the phenomenon type (quantitative or qualitative) they represent. Device items were defined as a real world device created in the manufacturing process. It inherits characteristics defined in the “device model”, allowing the association of addresses to the measurement points and enabling a phenomenon to be associated to a specific location. Device model monitoring criteria was described as the ability of selecting data record features, providing the DAS with the ability to define reports, with a consistent status. To

enable this the proposed conceptual model provided trigger conditions, enabling automatic recording based on time or system states. Three monitoring criteria were specified, as shown in Table 3.1.

Table 3.1: Monitoring criteria and respective trigger condition.

Monitoring Criteria	Trigger Condition
Device model composition	Records of changes in system composition
Device model status	Records at specific scheduled time
Device model event	Records of certain system states

In the model proposal, monitoring criteria would be set in the device item monitoring criteria. Two categories were provided, “predefined” and “custom”, meaning that they could be part of the model or specific to each device. Data access mechanisms were also provided, based on “public” and “private” categories, thus making it possible to establish the level of access provided to the monitoring reports. Finally, Nieva and Wegmann described observation and monitoring reports, which defined the way in which observation records of the device item would be taken. Each observation would be associated with a time-stamp. The conceptual model proposal presented “measurement observations” (quantitative) and “category observations” (qualitative). The latest includes a Boolean category identified as “present”, aiming to represent an indication of presence or absence of a certain condition or phenomenon.

The conceptual model presented in [3.58] must be considered in association with concerns with the existence of standards related with the subject area. Many of them are under development [3.59, 3.60, 3.61, 3.62]. Additionally, other technologies such as Universal Plug and Play (UPnP) were also referred to [3.63].

The development and implementation of a model based on such a proposition would make system developments easier by providing a so-called level of abstraction from the field devices. Nevertheless, to provide such capabilities, field devices must be fully compatible with the proposed technology. The paper suggested a codification scheme, where the device model, manufacturers and serial numbers would be used to

Chapter 3 – Literature Review

compose an identification method. Although such scheme would be only applicable for new forthcoming devices, there was not a suggestion to support those that do not fully comply with the proposed model. However, the implementation of such a model could represent a step forward in providing ways to create data acquisition systems based on layers, where monitoring applications would not have to directly access the field devices. Data could be made available by means of a common interface, provided that field devices were aware about the criteria that should be followed to collect and deliver the data. The standards concerning the data acquisition systems and devices, and the way in which they communicate would at this stage be very important to enable the technology to become fully available.

3.8 – Summary

With the purpose of improving manufacturing processes productivity and quality rates, condition monitoring and fault diagnostics methods have been intensively investigated. Developments in PC hardware and software have provided means to enhance such investigations, enabling the development of monitoring systems capable of analysing and integrating the acquired data from processes and machines. However, the dynamics associated with most of the processes still represent an obstacle to be overcome, requiring the consideration of a large number of parameters and variables. The development of distributed systems introduces an alternative by spreading the knowledge and expertise to different levels within the system. Low-cost processing devices, such as microcontrollers, emerge as an alternative, providing remote processing capabilities and thus supporting distributed and intelligent applications. To integrate a wide range of intelligent devices, communication protocols and modelling models/methods that consider modern technologies such Internet, database and artificial intelligence are required.

The next chapter of this thesis will concentrate on the technological aspects that relate to data acquisition and monitoring systems. The development of powerful electronic devices and communications protocols represent a step forward in the implementation of distributed and intelligent structures required to achieve the goals aimed by most of the research in the field.

REFERENCES

- 3.1 **Piete, M.A., Kinney, S.K. and Haves, P.** Analysis of an Information Monitoring and Diagnostic System to Improve Building Operations. *Energy and Buildings*, 2001, 33, 783-791.
- 3.2 **Kosmatopoulos, C. and Tsagourias, N.** Development of a Stand Alone Monitoring System (S.A.M.O.S.). *In Proceedings: International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Application*, Foros – Ukraine, IEEE, 1- 4 July, 2001, 36-39.
- 3.3 **Pollard, J.K., Rohman, S. and Fry, M.E.** A Web-Based Mobile Medical Monitoring System. *In proceedings: International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Application*, Foros – Ukraine, IEEE, 1- 4 July, 2001, 32-35.
- 3.4 **Stephanopoulos, G. and Han, C.** Intelligent System in Process Engineering: a Review. *Computers Chemistry Engineering*, 1996, 20(6/7), 743-791.
- 3.5 **Giarratano, J. and Riley, G.** Expert Systems – Principle and Programming. Boston, USA: PWS Publishing Company, 1998.
- 3.6 **Meziane, F., Vadera, S., Kobbacy, K. and Proudlove, N.** Intelligent Systems in Manufacturing: Current Developments and Future Prospects. *Integrated Manufacturing Systems*, 2000, 11(4), 218-238.
- 3.7 **Kuzawinski, K.M. and Smurthwaite, R.** Automated Fault Tree Analysis Via AI/ES. *In Proceedings: Annual Reliability and Maintainability Symposium*, IEEE, 1988, 331-335.
- 3.8 **Raaphorst, A.G.T, Netten, B.D. and Vingerhoeds, R.A.** Automated Fault-Tree Generation for Operational Fault Diagnosis. *In proceeding: Electric Railways in a United Europe*, IEE, 27-30 March, 1995, 173-177.
- 3.9 **Kavčič, M. and Juričić, D.** CAD for FaultTree-Based Diagnosis of Industrial Processes. *Engineering Applications of Artificial Intelligence*, 2000, 14, 203-216.
- 3.10 **Andrews, J.D. and Dunnett, S.J.** Event-Tree Analysis Using Binary Decision Diagrams. *IEEE Transactions on Reliability*, 2000, 49(2), 230-238.

- 3.11 **Hu, W., Starr, A.G. and Leung, A.Y.T.** Integrated Hierarchical Diagnostic Reasoning for FMS's. In *Proceedings: First International Conference on the Integration of Dynamics, Monitoring and Control (DYMAC '99)*, 1-3 September, Manchester – UK, 1999, 125-130.
- 3.12 **Hu, W., Starr, A.G., Zhou, Z. and Leung, A.Y.T.** A Systematic Approach to Integrated Diagnosis of Flexible Manufacturing Systems. *International Journal of Machine Tools and Manufacture*, 2000, 40, 1587-1602.
- 3.13 **Hu, W., Starr, A.G. and Leung, A.Y.T.** Operational Fault Diagnosis of Manufacturing Systems. *Journal of Material Processing Technology*, 2003, 133, 108-117.
- 3.14 **Yang, S.K. and Liu, T.S.** A Petri-net Approach to Early Failure Detection and Isolation for Preventive Maintenance. *Quality and Reliability Engineering International*, 1998, 14, 319-330.
- 3.15 **Prickett, P.** A Petri-net Based Machine Tool Maintenance Management System. *Industrial Management and Data Systems*, 1997, 97(4), 143-149.
- 3.16 **Davey, A., Grosvenor, R., Morgan, P. and Prickett, P.** Petri-net Based Machine Tool Failure and Diagnosis. In *Proceedings: COMADEM '96*, 16-18 July, Sheffield – UK, 1996, 723-731.
- 3.17 **Ajtonyi, I. and Terstánszky, G.** Fault Diagnosis in Industrial Processes via Parallel Processing. In *Proceedings: CONTROL '94*, IEE, 21-24 March, 1994, 1117-1121.
- 3.18 **Jantunen, E. and Jokinen, H.** Reduction of Data Needed in an Expert System for Condition Monitoring of FMS Using Regression Analysis Techniques. In *Proceedings: COMADEM '96*, 16-18 July, Sheffield – UK, 1996, 675-684.
- 3.19 **Wang, X.Z., Chen, B.H. and McGreavy, C.** Data Mining for Failure Diagnosis of process Units by learning Probabilistic Networks. *Transaction of the Institution of Chemical Engineers – IChemE*, 1997, 75(B), 210-216.
- 3.20 **Lennox, B., Montague, G.A., Frith, A.M., Gent, C. and Bevan, V.** Industrial Application of Neural Networks – an Investigation. *Journal of Process Control*, 2001, 11, 497-507.
- 3.21 **Mageed, M.F.A., Sakr, A.F and Bahgat.** Fault Detection and Identification Using Hierarchical Neural Network – Based System. In *Proceedings: IECON'93*, IEEE, 15-19 November, 1993, 338-343.

- 3.22 **Jämsä-Jounela, S.L., Vermasvuori, M., Endén, P. And Haavisto, S.** A process Monitoring System Based on the Kohonen Self-Organizing Maps. *Control Engineering Practices*, 2003, 11, 83-92.
- 3.23 **Ballé, P. and Isermann, R.** Fault Detection and Isolation for Nonlinear Processes Based on Local Linear Fuzzy Models and Parameter Estimation. *In Proceedings: American Control Conference, AACC*, 1998, 1605-1609.
- 3.24 **Chafi, M.S., Akbarzadeh, M. and Moavenian, M.** Fault Detection and Isolation in Nonlinear dynamic Systems: a Fuzzy-Neural Approach. *In Proceedings: IEEE International Fuzzy Systems Conference, IEEE*, 2001, 1072-1075.
- 3.25 **Geropp, B.** Artificial Neural Networks and Fuzzy-Logic Used for Reliable Machine Diagnosis. *In Proceedings: COMADEM '97, 9 and 10 June, Espoo – Finland*, 1997, 565-573.
- 3.26 **Sharif, M.A. and Grosvenor, R.I.** Process Plant Condition Monitoring and Fault Diagnosis. *Proceedings of the I MECH E Part E Journal of Process Mechanical Engineering*, 1998, 212(1), 13-30.
- 3.27 **Prickett, P.W. and Johns, C.** An Overview of Approaches to End Milling Monitoring. *International Journal of Machine Tools & Manufacture*, 1999, 39, 105-122.
- 3.28 **Dimla Snr., D.E.** Sensor Signals for Tool-wear Monitoring in Metal Cutting Operations - a Review of Methods. *International Journal of Machine Tools & Manufacture*, 2000, 40, 1073-1098.
- 3.29 **Jantunen E.** A Summary of Methods Applied to Condition Monitoring in Drilling. *International Journal of Machine Tools & Manufacture*, 2002, 42, 997-1010.
- 3.30 **Sick B.** On-line and Indirect Tool Wear Monitoring in Turning with Artificial Neural Networks: a Review of More than a Decade of Research. *Mechanical Systems and Signal Processing*, 2002, 16(4), 487-546.
- 3.31 **Prickett, P.W. and Grosvenor, R.I.** Non-sensor Based Machine Tool and Cutting Process Condition Monitoring. *International Journal of COMADEM*, 1999, 2(1), 31-37.
- 3.32 **Peterson, J.L.** Petri Net Theory and the Modelling of Systems, Englewood Cliff, Prentice-Hall, 1981.

- 3.33 **Dunn, S.** Condition Monitoring in the 21st Century. Plant Maintenance Resource Centre, Plant Maintenance Resource Centre Web Site, Available from: <http://www.plant-maintenace.com/articles/ConMon21stCentury.shtml>, [Accessed 20 August 2002].
- 3.34 **de Vicq, A.** 21st Century Machinery. *IEE Manufacturing Engineer*, 2001, 80(3), 104-108.
- 3.35 **Tönshoff, H.K., Janocha, H., Seidel, D. and Roethel, J.** A New Approach to Machine Monitoring and Diagnosis. *The Journal of Condition Monitoring*, 1990, 3(3), 161-168.
- 3.36 **Ramakrishna, K.** Development of a Computerized On-Line Vibration Monitoring, Analysis and Assessment System for Power Plant Machinery. *International Journal of COMADEM*, 2001, 4(2), 5-12.
- 3.37 **Jeng, J.J. and Wei, C.Y.** An On-line Condition Monitoring and Diagnosis System for Feed Rolls in the Plate Mill. *Journal of Manufacturing and Science Engineering*, February 2002, Vol. 154, 52-57.
- 3.38 **Alexandru, A.** A Real Time Knowledge-based Approach for Fault Diagnosis and its Applications. *International Journal of COMADEM*, 2000, 3(2), 32-38.
- 3.39 **Angeli, A.** An Online Expert System for Fault Diagnosis in Hydraulic Systems. *Expert Systems*, 1999, 16(2), 115-120.
- 3.40 **Bonastre, A.R Ors and Peris, M.** Distributed Expert Systems as a New Tool in Analytical Chemistry. *Trends in Analytical Chemistry*, 2001, 20(5), 263-271.
- 3.41 **Manders, J., Barford, L.A. and Biswas, G.** An Approach for Fault Detection and Isolation in Dynamic Systems from Distributed Measurements. *IEEE Transaction on Instrumentation and Measurement*, 2002, 51(2), 235-240.
- 3.42 **Ranky, P.G.** Smart Sensors. *Sensor Review*, 2002, 22(4), 312-318.
- 3.43 **Lee, K.** IEEE 1451: A Standard in Support of Smart Transducer Networking. *In Proceedings: IEEE Instrumentation and Measurement Technology Conference*, 1 to 4 May, Baltimore – USA, 2000, 525-528.
- 3.44 Network and Telecommunication Research Group, Undergrad Coursework and Projects, Network and Telecommunication Research Group Web Site, Available from: <http://ganges.cs.tcd.ie/undergrad/4ba2/multicast/index.html>, [Accessed 13 September 2002].

- 3.45 **Mosterman, P.J. and Biswas, G.** Model Based Diagnosis of Dynamic Systems. Vanderbilt University, Centre for Intelligent Systems, Centre for Intelligent Systems Web Site, Available from: <http://www.isis.vanderbilt.edu/activities/mic97/papers/mbd/p.html>, [Accessed 13 September 2002].
- 3.46 **Thomas, G.** Ethernet, Arcnet and CAN – Proposed Network Hierarchy for Open Control. Contemporary Controls, Customer Support, Contemporary Controls Web Site, Available from: <http://www.ccontrols.com/whitepaper.htm>, [Accessed 13 September 2002].
- 3.47 **Tseng, P.C. and Chou, A.** The Intelligent On-line Monitoring of End Milling. *International Journal of Machine Tools & Manufacture*, 2002, 42, 89-97.
- 3.48 **Ong, S.K., An, N. and Nee, A.Y.C.** Web-Based Fault Diagnosis and Learning System. *The International Journal of Advanced Manufacturing Technology*, 2001, 18, 502-511.
- 3.49 **Baccigalupi, A., Bernieri, A. and Pietrosanto, A.** A Digital-Signal processor-Based Measurement System for On-Line Fault Detection. *IEEE Transaction on Instrumentation and Measurement*, 1997, 46(3), 731-736.
- 3.50 **Roberts, C., Dassanayake, N., Lehrasab, N. and Googman, C.J.** Distributed Quantitative and Qualitative Fault Diagnosis: Railway Junction Case Study. *Control Engineering Practice*, 2002, 10, 419-429.
- 3.51 **Baek, D.K., Ko, T.J. and Kim, H.S.** Real Time Monitoring of Tool Breakage in a Milling Operation Using Digital Signal Processor. *Journal of Materials Processing Technology*, 2000, 100, 266-272.
- 3.52 **Bolic, M., Drndarevic, V. and Samardzic, B.** Distributed Measurement and Control System Based on Microcontrollers with Automatic Program Generation. *Sensors and Actuators A*, 2001, 90, 215-221.
- 3.53 **De Frutos, J.A. and Giron-Sierra, J.M.** Design of a Distributed System Architecture Including an Automatic Code Generator. *Microprocessors and Microsystems*, 2002, 26, 207-213.
- 3.54 National Instruments. Building Intelligent Ethernet-Based Distributed I/O Systems with National Instruments LabVIEW. User Development Zone Web Site, Available from: <http://zone.ni.com/devzone/conceptd.nsf/webmain>, [Accessed 01 August 2002].

Chapter 3 – Literature Review

- 3.55 **Dassanayake, H.P.B., Roberts, C. and Goodman, C.J.** An Architecture for System-wide Fault Detection and Isolation. *Proceedings of the Institute of Mechanical Engineers*, 2001, 215(I), 37-46.
- 3.56 **Ehrlich, J., Zerrouki, A. and Demssieux, N.** Distributed Architecture for Data Acquisition: a Generic Model. *In Proceedings: IEEE Instrumentation and Measurement Technology Conference*. Ottawa – Canada, 19 - 21 May 1997, 1180-1185.
- 3.57 Synergetic Micro Systems, Factory Communications Comparison, Synergetic Micro Systems Web Site. Available from: <http://www.synergetic.com/compare.htm>, [Accessed 19 November 2002].
- 3.58 **Nieva, T. and Wegmann, A.** A Conceptual Model for Remote Data Acquisition Systems. *Computers in Industry*, 2002, 47, 215-237.
- 3.59 OPC Foundation, OLE for Process and Control Standard, OPC Foundation Web Site. Available from: <http://www.opcfoundation.org>, [Accessed 25 November 2002].
- 3.60 VI Foundation, Interchangeable Instrumentation Standard, VI Foundation Web Site. Available from: <http://www.ivifoundation.org>, [Accessed 25 November 2002].
- 3.61 ODAA, Open Data Acquisition Standard, Open Data Acquisition Association Web Site. Available from: <http://www.opendaq.org>, [Accessed 25 November 2002].
- 3.62 OMG, DAIS – Data Acquisition from Industrial Systems RFP (dte/99-01-02), Object Management Group Web Site. Available from: <http://www.omg.org>, [Accessed 25 November 2002].
- 3.63 UPnP Forum, Universal Plug and Play Technology, UPnP Web Site. Available from: <http://www.upnp.org>, [Accessed 25 November 2002].

CHAPTER 4

TECHNOLOGY FUNDAMENTALS

4.1 – Introduction

The distributed monitoring system produced as a result of this research is based upon a range of currently available technologies. The development of these new technologies has enabled the design of systems that explore the best features of each individual component, sharing resources in order to achieve a better final result. In particular, new network technologies, either in terms of hardware or software protocols, open the opportunity for the development of distributed systems. Tanenbaum and van Steen [4.1] considered distributed systems as a large number of computers connected by a high-speed network. They also added that such system should be easy to expand, considering each member as independent. But they presented as the main goal for such a distributed system “is to make it easy for users to access remote resources”.

This present research explored the general concept of distributed systems in order to provide the resulting monitoring system with flexibility and improved capability. Thus, the technology required to support such an implementation becomes a very important aspect and will be reviewed in the following sections, with special emphasis to those components that are the most relevant in the investigation.

4.2 – Processing Technology

Processing technology represents a key element in distributed monitoring research, since it establishes a reference in terms of a system’s capabilities and limitations. The evolution of integrated circuit technology in the 1970s provided the means required for the development of a generation of devices with sufficient processing capabilities. The early generation of 4 bits microprocessors, initially designed for particular

purposes, has developed to a range of general-purpose devices based on 8, 16, 32 and more recently 64 bits [4.2]. Although initially conceived and concerned with applications such as the development of microcomputers, microprocessors were soon seen as a flexible solution for industrial applications, due to their programming capabilities [4.3]. In general terms, a microprocessor represents the Central Processing Unit (CPU) of a computer system and can be mainly divided into a Control Unit (CU), an Arithmetic and Logic Unit (ALU) and a set of registers that includes the Program Counter (PC) [4.4], as represented in Figure 4.1. Table 4.1 describes the main functions of each element. In order to operate, a CPU requires a main memory system (to hold execution code and data) and an input/output (I/O) system (to enable communication with the external world).

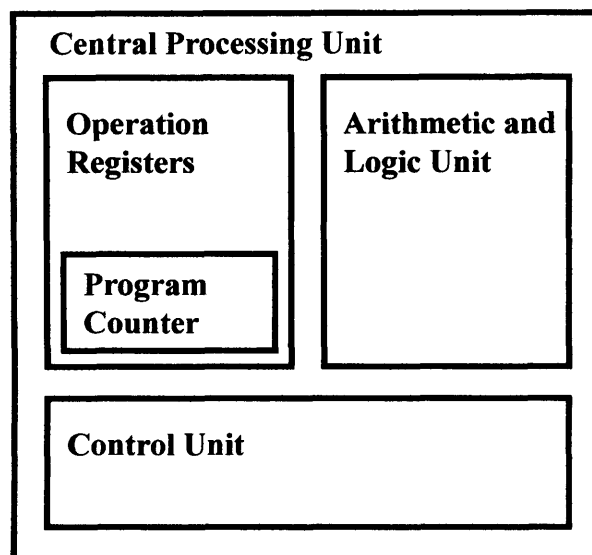


Figure 4.1: Central processing unit general diagram [4.4].

Table 4.1: CPU main parts description [4.4].

CPU Element	Function Description
Control unit	<ul style="list-style-type: none"> • Controls the computing operation.
Arithmetic and logic unit	<ul style="list-style-type: none"> • Performs arithmetic, logic and shift operations.
Register set	<ul style="list-style-type: none"> • Holds values during computing operation.
Program counter	<ul style="list-style-type: none"> • Holds the instruction memory address.

Microprocessors can be classified in terms of the way they interact with the main memory system. Models with von Neumann architecture use the same addressing and data structure for data and instructions. Alternatively, Harvard architecture devices provide independent pathways for data and instructions [4.4]. The concepts are depicted in Figure 4.2. Other types of computer architecture exist but are outside the purpose of this technology review.

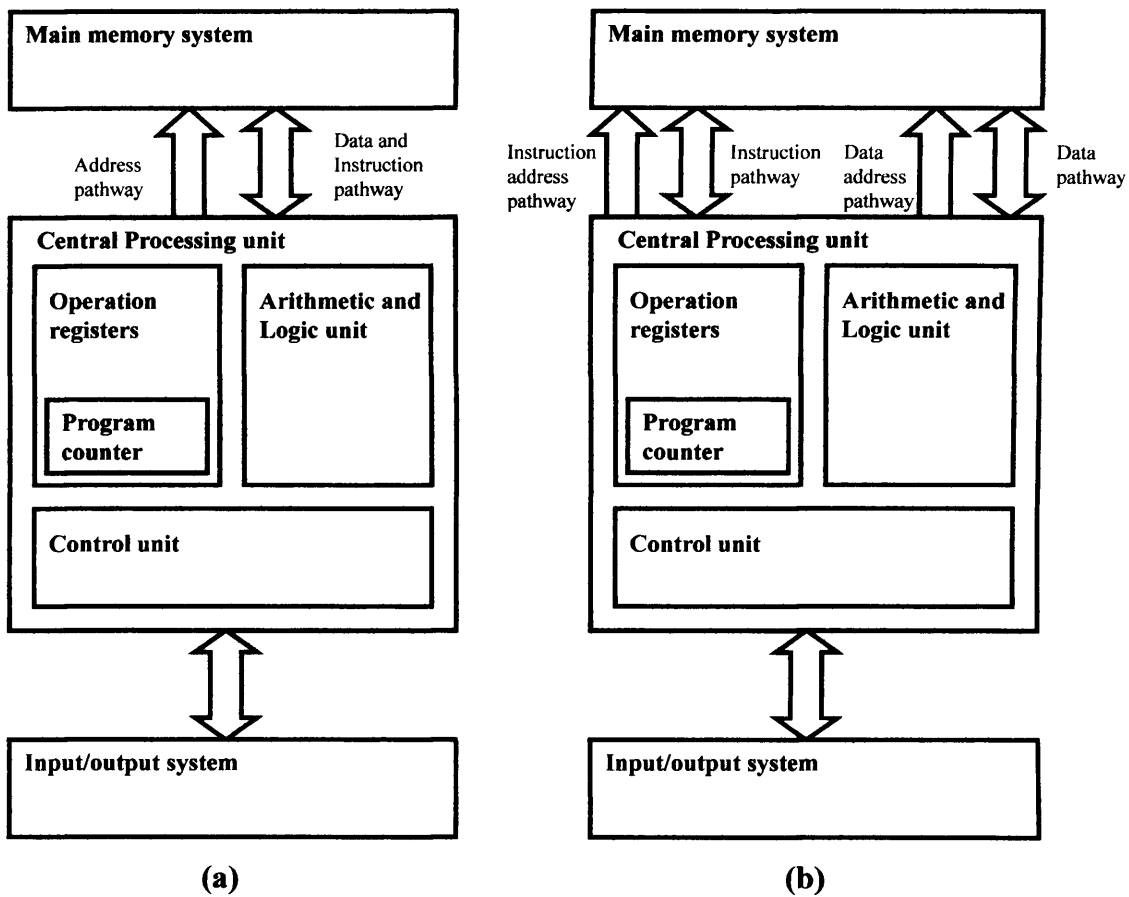


Figure 4.2: Computer architecture main components (a) conventional von Neumann and (b) Harvard [4.4].

Within microprocessors, basic arithmetic, logic and shifting operations are performed by the Arithmetic and Logic Unit (ALU). The ALU operations in many cases affect the CPU status registers (flags indicating the operation results, such as zero, negative, borrow, overflow, etc.).

The control unit executes a sequence of “instruction fetch” and “instruction execution”. There are basically two different types: microprogrammed and conventional (hardwired). The first type executes each instruction of the CPU instruction set as a microprogram, which normally represents a sequence of several microinstructions (depending on the instruction, it may require only one microinstruction). This is usually the basis for the implementation of Complex Instruction Set Computer (CISC) architecture. Although providing a greater level of flexibility in terms of the instruction set, this may have a performance cost. Conventional or hardwired control units have better performance, since instead of using microprograms, the instruction directly interacts with the control unit logical circuits. These units are now often the basis for the implementation of so called Reduced Instruction Set Computer (RISC). The main features of RISC devices are [4.4, 4.5]:

- A smaller instruction set;
- Instructions have a uniform length;
- Normally, a single instruction format;
- Register-to-register instructions;
- Pipelined organisation;
- Usually, one cycle per instruction;
- Usually, based on Harvard architecture.

Arguments in favour of RISC include the fact that by simplifying the Instruction Set Architecture (ISA), CPUs based on the principle would result in a simpler hardware, therefore providing means to ease design, improve processing performance and enabling cost reduction. Baron and Higbie [4.4] considered that the improvement in performance is also the result of changes in the electronic industry, mainly those related with the improvements of memory devices.

The emerging RISC architectures enabled designers to concentrate on optimising power consumption and to integrate more capabilities in a single silicon piece, making it the industry standard for microcontroller design [4.5]. In simple terms, a microcontroller became defined as a “computer on a chip” [4.5]. Such devices,

besides having CPU core, provide program memory, data memory, I/O controllers and other specific features such as timers and communication, all integrated into a single device. The continuous evolution of the electronic industry is providing new versions of microcontrollers with increasing processing capabilities, making them a flexible choice for a larger range of applications. These include monitoring tasks that might require communication and local processing abilities.

4.2.1 – The PIC18C452 Microcontroller

Of the range of microcontrollers available, those provided by Microchip Technology Inc. were selected as being appropriate for the current research. The particular PIC™ single chip microcontroller and its capabilities are thus reviewed in this section. This family of microcontrollers represents an example of the recent developments of the electronic industry. Flexibility is provided by the existence of a range of 8 bit processing devices to select from, according to the application requirements. Different packages, memory sizes and technologies, device special features and power/operating options are available, allowing system designers to balance cost and performance requirements. Table 4.2 lists the main groupings within the PIC family of microcontrollers and their characteristics. Detailed information on all devices, their features and configurations, is available from the company's web site [4.6].

As stated, for the context of this research, the use of this family of microcontrollers was seen as a good alternative for the implementation of a monitoring structure, since they are capable of providing low-cost, simple designs and deployment flexibility. The PIC18C452 in particular became available during this research and has features that allow its use in a wider and general range of applications. The device is provided with 32 Kbytes program memory (16 K instructions) and 1.5 Kbytes of data memory. The technology employed enables the device to operate at frequencies up to 40 MHz. Each instruction requires 4 periods of clock to execute, represented by the CPU instruction cycle described in Table 4.3. This allows the microcontroller to achieve a performance of 10 million instructions per second (MIPS), at the maximum rated frequency [4.7].

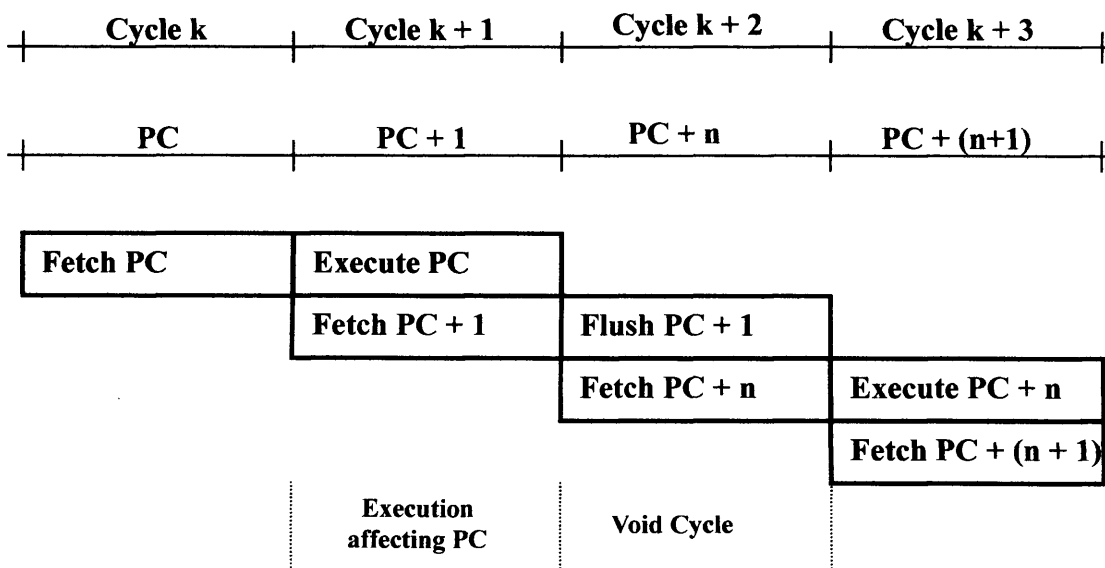
Table 4.2: PIC family of microcontrollers main groups and characteristics.

Groups	Package	Max. Freq.	Program Memory	Data Memory	I/O	Analogue	Timers	Serial I/O
PIC12Cxxx	8 pins	4 - 10 MHz	768 - 3584 bytes 512 - 2048 instr.	25 - 128 bytes	6 pins	8 bits	8 bits	None
PIC16Cxxx	18 - 68 pins	8 - 24 MHz	768 - 14336 bytes 512 - 8192 instr.	24 - 368 bytes	6 - 52 pins	8, 10 & 12 bits	8 & 16 bits	USART / I ² C / SPI
PIC17Cxxx	40 - 84 pins	33 MHz	4096 - 32768 bytes 2048 - 16384 instr.	232 - 902 bytes	12 - 52 pins	10 bits	8 & 16 bits	USART / I ² C / SPI
PIC18Cxxx	28 - 84 pins	40 MHz	16384 - 32768 bytes 8192 - 16384 instr.	512 - 1536 bytes	23 - 68 pins	10 bits	8 & 16 bits	USART / I ² C / SPI / CAN

Table 4.3: PIC18C452 instruction cycle clock period operation.

Clock Period	CPU Action Description
1	Instruction decode cycle or no operation
2	Instruction data read cycle or no operation
3	Data processing
4	Instruction data write or no operation

No operation cycles may be required accordingly the actual instruction action.



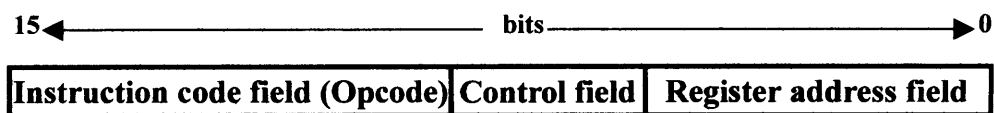
PC – Program Counter

Figure 4.3: PIC18C452 instruction flow.

The device architecture design employs concepts such as two-stage pipeline, overlapping the instruction fetch with the execution of the previous instruction. By

utilising the Harvard architecture, instructions can be fetched using a separate bus from the one used to access the data memory. These approaches provide the required means to enable instruction fetch and execution to be performed in one single cycle. Operations affecting/changing the processing unit program counter register (typically branch type instructions) result in an “exception to the rule”. For such operations, two cycles are required. Figure 4.3 provides a diagram of an instruction flow [4.7].

The PIC18C452 instruction set is based on a 16 bit instruction width (instruction word). This means that the instruction pathway is also 16 bits, in contrast with the 8 bits of the data pathway. The device instruction set is composed of 77 basic instructions. Of these, 4 require a double word instruction (32 bits) for their codification, therefore demanding two cycles to fetch the entire instruction.



The fields may assume different length accordingly specific instructions

Figure 4.4: General instruction format, considering registers addressing mode.

The instructions are formatted in such way that all the microcontroller 8 bits registers (which include the 1532 data registers and the 128 special function registers) are addressed within the instruction. Figure 4.4 shows details of such a formatting structure. A complete description of the PIC18C452 instruction set and the respective format can be found within the device documentation [4.8].

The PIC18C452 data memory is organised into 16 banks of 256 bytes each (file registers). Although representing an addressing range of 4 Kbytes, only 1.5 Kbytes are effectively implemented as application enabled registers. These are located in banks 0 to 5. Another 128 bytes are physically implemented for device control purposes and are called Special Function Registers (SFR), occupying the second half of the last bank (bank 15). The map representation of the registers distribution within the microcontroller data memory area is shown in Figure 4.5. Access to a specific register

within each bank can be via two addressing modes. Direct addressing mode requires the previous selection of the bank wherein the register is located. Indirect addressing mode is provided by means of three 12 bits pointers, which provide continuous access to the entire 4 Kbytes addressable registers.

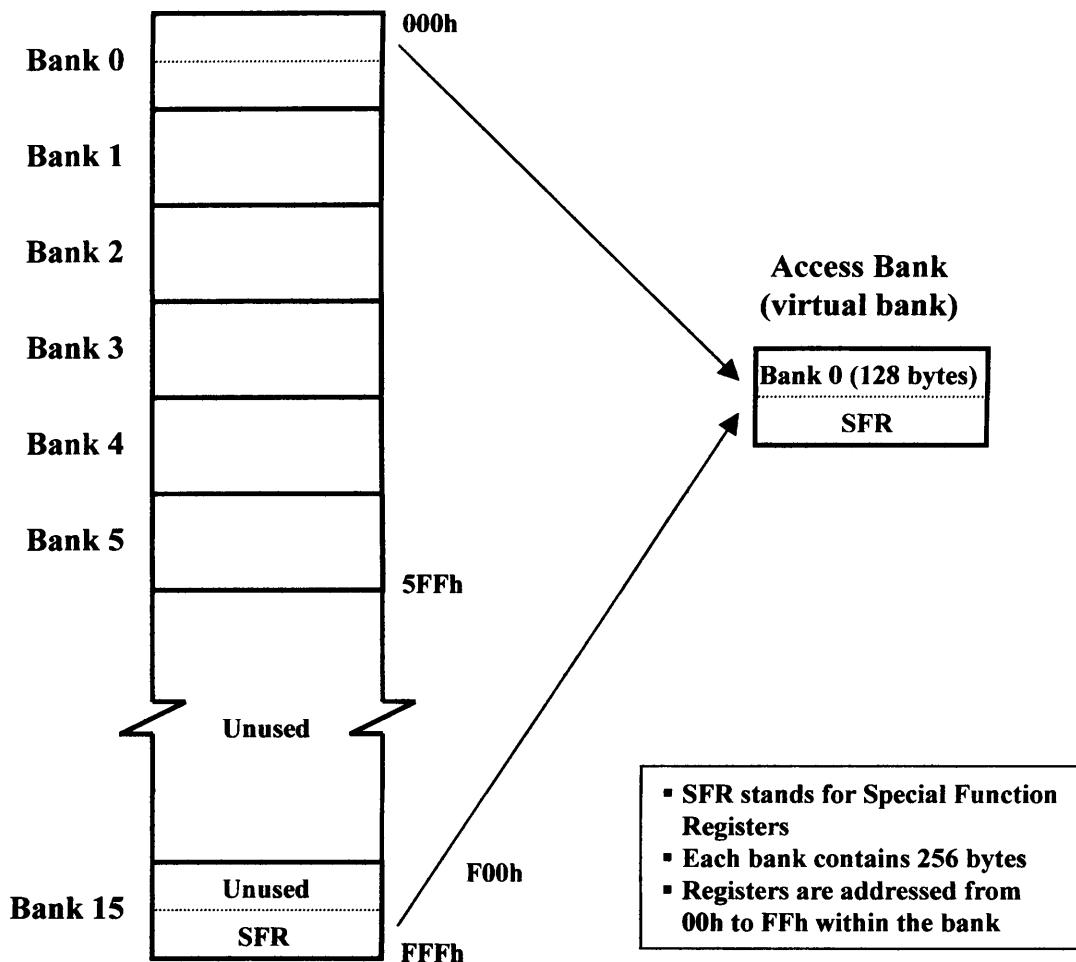


Figure 4.5: PIC18C452 registers map organisation and access bank mode [4.8].

To ease software implementation (and support compilers), a special addressing feature is provided with the PIC18Cxxx series. It is called an “access bank” and can be selected by setting (or unselected by clearing) a bank access control bit, which exists within the control field of most instructions. Setting the bit provides access to a virtual bank consisting of the first half of bank 0 and the second half of bank 15 (device SFR). Compilers and applications can thus have access to the SFR and also to 128 bytes (variables), without caring about the actual selected bank. The register

segmentation is a requirement for RISC processors, since a register-to-register architecture would require a very large instruction word to accommodate the absolute register address [4.2].

The available 32 Kbytes program memory in the PIC18C452 enables a maximum of 16,384 instructions, since each instruction has in general a length of 2 bytes. These instructions are addressed by a program counter that is 21 bits wide, organised in 3 registers. The microcontroller is also provided with interrupt capabilities, enabling embedded peripherals and specific external hardware lines to generate such request. Two locations in the program memory are vectored by the interrupts accordingly their priority, low (0008h) or high (0018h). For improved performance, a 31 level program counter wide (21 bits) embedded stack is implemented in the PIC18C452, to support interrupts and subroutine calls by automatically storing the returning address. An equally interesting feature is the capability of asynchronously accessing byte locations in the program memory, organised in the form of data tables. A 21 bits wide addressing mode to access such tables is obtained by joining 3 “table address” registers, which can be manipulated by the application. Automatic address increment and decrement are provided by some instructions, although 2 instruction cycles may be required for execution. Read and write modes are supported. The device specific documentation is the source for further details [4.8].

The microcontroller’s CPU is provided with an 8 bit ALU, responsible for the arithmetic, logic and shifting operations. An operation with two operands requires one of them being placed in the Working Register (WREG). The ALU of the 18Cxxx series is also capable of unsigned 8 x 8 bits multiplications, with results up to 16 bits wide. All such instructions are single cycle execution based. Nevertheless, further cycles may be required, in some cases, to initialise registers or to retrieve results. An 8 bits Status Register (STATUS) holds the device flags, which indicate the resulting effects of the operation performed by the ALU. Some examples are zero-results, negative-results, carry states and overflow indications.

One of the benefits provided by microcontrollers is their embedded peripherals and I/O control, allowing development of an application with minimal external hardware. The PIC18C452 in particular is provided with 34 I/O pins, each typically with several

multiplexed functions. The simplest use of these pins is as digital I/Os, organised in three 8 bit, one 7 bit and one 3 bit ports. Each port can have their individual I/O lines configured either as inputs or as outputs. These configurations can be dynamically changed during program execution, increasing application flexibility. The PIC18C452 also provides support for analogue input signals. The multiplexing feature enables 8 of the I/O pins to be selected as analogue inputs. A single 10 bit analogue-to-digital (A/D) converter is provided, thus requiring analogue acquisition and conversion to be multiplexed. These characteristics were explored to implement the monitoring system that resulted from this research.

Four timers, with 8 or 16 bit timing registers, are available with the PIC18C452. Most of them (3), allow either an internal (system clock) or an external (input pin) clocking. Prescaling of the clocking source is selectable, at individual rates for each timing device. Three of the timing devices are also capable of acting as counters, triggered by an external source. All the 4 timing peripherals have interrupt capabilities and their registers can be read / written at any time. The capture function acquires specific timer registers when triggered by an external signal. The compare function watches specific timer registers and when the comparison matches, it drives high/low (configurable) a specific I/O pin. The combined capture/compare functions can also generate interrupts, if required. A third related function may provide a 10 bit resolution Pulse Width Modulation (PWM) output.

Communication peripherals are also embedded within the PIC18C452. The first such device to be considered is the Universal Synchronous/Asynchronous Receiver/Transmitter (USART). The asynchronous mode is useful in implementing communication capabilities in order to exchange data with other devices/equipment, by means of a standard RS232C serial interface. In such a mode, the data transmit and receive clock are derived from the microcontroller's operating clock and data transfer is full duplex. Separate 1 byte receive/transmit buffers are provided. Interrupts are also separate for the transmitter and receiver modules and may be generated if enabled. The transmitter interrupt is associated with the respective buffer becoming empty event. On the other hand, the receiver interrupt is associated with the receiver buffer full event. Start bit, stop bit and ninth bit (parity checking or multi-processor addressing mode) are selectable. Error indications such as received frame error (stop

bit) and buffer overrun are automatically generated. Supported baud rates range from 300 bps to 115.2 Kbps. The USART also operates in synchronous mode, acting either in master or slave mode.

The second available serial communication method is provided by means of an embedded Synchronous Serial Port (SSP). Two operating modes are supported: Serial Peripheral Interface (SPI®) and Inter-Integrated Circuit (I²C®). The SPI (developed by Motorola) can operate as master (providing clock) or as slave (clock provided by a master). Although using separate hardware lines for data transmission and reception, one single buffer is provided for both events (bits are continuously shifted). A single interrupt is provided and is synchronised with the buffer full condition (8 bits shifted in and uploaded). The data transfer clock is unique and controlled by the master device. To receive data from a slave device, besides enabling the clock, the master must transmit a stream of “dummy data” equivalent to the amount to be received. Supported data transfer rates are up to 10 Mbps.

The PIC18C452 is fully compliant with the I²C specification (developed by Phillips) [4.7]. In this operating mode the device uses one hardware line for data transfers and a second one for the clocking signal. Master and slave modes are both supported. Interrupts can be generated by the reception of a start bit (synchronising the beginning of a data transfer) and by the identification of a stop bit (end of data transfer). The I²C uses an addressing mode (7 or 10 bits long), which is automatically searched and detected by the microcontroller’s data communication interface. Supported data transfer rates may be as much as 1 Mbps. Further details concerning the microcontroller’s communication peripherals may be found in [4.7].

Within this research context, the USART and SPI communication methods were employed. As it will be seen in Chapter 7, USART became the natural choice to communicate between third party development kits, employed in the implementation of a “connectivity module”. In the development of a “monitoring module” (also described in Chapter 7), SPI was selected due to design simplicity and compatibility with other peripheral devices, such as the CAN bus controller, considered later in this chapter (Section 4.4.2).

The main elements concerning the PIC18C452, especially those that were subsequently used in this research have been described in this section. The device was the heart of the developed distributed monitoring system and provided data acquisition, inter-chip communication and information exchange implementation. The hardware and software developments are detailed later, in Chapter 7.

4.3 – Data Acquisition

Having identified the PIC microcontroller as the device to provide, among other things, the data acquisition function for the distributed monitoring system, other more general data acquisition topics were considered and are briefly reviewed.

The development of faster computers and hardware standards for computer bus architectures has made Personal Computers (PCs) one of the preferred platforms for data acquisition implementations. PCs become an attractive choice due to their processing power and data storage capabilities, together with the support provided by multi-task operating systems [4.9]. However, hardware specific implementations still represent an important element in data acquisition design. In many cases there are real-time requirements and the hardware specific implementations often provide local processing and reduce the demands on the main PC processor [4.9]. Bolic et al [4.10] suggested that there are also several applications that do not require the power of a PC or furthermore, may require low-power consumption, low-cost or small physical dimensions.

Figure 4.6 represents the general aspects of a data acquisition implementation. Apart from the chosen processing platform, many considerations in terms of the data acquisition hardware are common. Transducers are required to convert the physical phenomena of a process into an electrical signal. Signal conditioning may be required in many cases, in order to adapt the transducer signal to those required by the data acquisition hardware. Table 4.4 summarises the main signal conditioning techniques. Different sorts of transducers are available, suitable to the different physical phenomena to be measured. The methods employed by such devices, in order to convert measurement into a useful signal may vary according to operating range,

accuracy and also cost. Some transducers of common use in industrial applications, together with their sensing method, which are appropriate for automatic measurements, are summarised in Table 4.5.

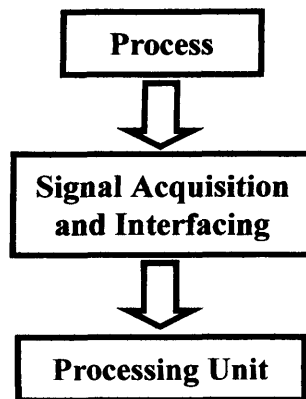


Figure 4.6: General representation of a data acquisition structure.

Table 4.4: Signal conditioning aspects.

Signal conditioning	Purpose description
Amplification	<ul style="list-style-type: none"> • Increase low-level electrical signals to reduce noise ratio (full voltage range amplification for better accuracy).
Isolation	<ul style="list-style-type: none"> • Electrically isolate the transducer from the processing unit, protecting equipment and avoiding measurement noise due to ground loop.
Multiplexing	<ul style="list-style-type: none"> • Enable several signals to be measured by a single measurement equipment by means of multiplexing technique (signal changing rate must be considered).
Filtering	<ul style="list-style-type: none"> • For DC-class signals, aims to reduce level of noise by using low-pass filtering techniques; • For AC-class signals, filtering methods may require very steep cut-off rates.
Excitation	<ul style="list-style-type: none"> • Transducers power supply requirements, such as currents or specific voltages.
Linearisation	<ul style="list-style-type: none"> • Compensate measurements due to non-linear response of the sensors / transducers (some times provided in software).

Table 4.5: Transducers and sensing techniques (compiled from [4.3]).

Physical phenomena	Measurement method	Sensing technique
Pressure	<ul style="list-style-type: none"> • Strain gauges • Variable capacitance • Linear var. diff. transformer • Piezo-electric effect 	<ul style="list-style-type: none"> • Pressure applied to the strain gauge, resulting in a resistance change; • Capacitance changes due to movement of a dielectric caused by pressure; • Transformer core moves due to pressure; • Output voltage resulting from applied pressure.
Temperature	<ul style="list-style-type: none"> • Thermocouple • Thermistor • Pyrometer 	<ul style="list-style-type: none"> • Electromotive force due to dissimilar metallic junctions; • Resistance variation due to temperature changes; • Heat wavelength radiation.
Flow	<ul style="list-style-type: none"> • Orifice plate • Venturi tube • Pilot tube • Turbine • Magnetic • Ultrasonic 	<ul style="list-style-type: none"> • Differential pressure due to restriction in the flow area; • Diff. pressure due to smooth and gradual reduction in tube diameter; • Differential pressure between static pressure and fluid flow; • Turbine rotor generates a electrical signal proportional to the flow rate; • Changes in the inductive voltage in a coil due to flow rate variations; • Measurement of acoustic wavelength changes due to flow rate variations.
Level	<ul style="list-style-type: none"> • ON/OFF switches • Continuous level 	<ul style="list-style-type: none"> • Beam breaking, capacitance, conductivity and float type level switches; • Capacitance (dielectric variation), differential pressure (level column), ultrasonic (wavelength reflection) and radioactive (absorbed radiation).
Displacement	<ul style="list-style-type: none"> • Angular and linear 	<ul style="list-style-type: none"> • Potentiometers, capacitance (parallel metal plates), inductive coil (permeable core), pulse counting, encoders and ON/OFF switches.
Velocity	<ul style="list-style-type: none"> • Linear • Angular 	<ul style="list-style-type: none"> • Time measurement based on pulse sensing; • Pulse sensing, electro-mechanical and digital tacho-generators.
Vibration	<ul style="list-style-type: none"> • Magnetic 	<ul style="list-style-type: none"> • Permanent magnet within a coil field, generating electrical signal.
Acceleration	<ul style="list-style-type: none"> • Strain gauges • Piezo-electric crystal 	<ul style="list-style-type: none"> • Changes in resistance due to applied forces; • Voltage variations due to strain in the crystal.
Force	<ul style="list-style-type: none"> • Weight • Force/torque 	<ul style="list-style-type: none"> • Load cells based on strain gauges principle; • Strain gauge and magnetic permeability changes due to tension variation.

Analogue signals represent an important source of monitored process parameters. The processing of such signals requires them to be converted into a digital format, which is normally carried out by an Analogue to Digital Converter (ADC). Currently ADC functionality is usually provided by specific electronic integrated devices or as an embedded function in many microcontrollers (e.g. PICs). The successive approximation implementation represents the fastest ADC method and is used in the PIC18C452 device.

Other parameters must be considered when using ADCs, of which resolution, sampling rate, input signal range, linearity, repeatability and code width are probably those of major importance. In simple terms, resolution expresses the number of bits used to represent an analogue signal sample. The higher the resolution, the lower the gap between successive levels of representation in the digital format. As an example, an 8 bits ADC provides 256 levels of representation of an analogue signal and therefore has a full-scale resolution of 0.39%. By using a 16 bit ADC the number of levels increases to 65,536, with a consequent full-scale resolution of 0.0015% [4.3]. Nwagboso et al [4.11] considered that a higher resolution would result in an increase in the data acquisition accuracy. However, other factors should be considered [4.9].

The sampling rate of an ADC indicates how often a conversion can take place and therefore directly affects the system accuracy. A fast sampling rate can provide a larger number of acquisitions of the analogue signal and must be considered as an important factor for fast changing signals [4.9]. There are also devices that, although providing several analogue channels, may share a single ADC unit among them by employing multiplexing techniques. In such cases, the sampling rate must consider the acquisition and conversion time and the number of analogue channels sharing the same resource. The PIC18C452 represents an example of a device with this type of implementation, providing one ADC for 8 analogue input channels [4.8].

Input signal range is characterised by the range of input voltages that are supported by the device or analogue channel. Linearity expresses the relationship between the analogue signal and the respective digital representation over the full input range and requires a good analogue circuitry design to achieve best results [4.9]. Similarly, repeatability becomes important in many applications where precise measurements

are required [4.3]. The code width represents the smallest detectable change of the voltage in the natural signal and is thus related to resolution, input range and gain. It can be calculated by means of Equation 4.1.

$$\text{CodeWidth} = \frac{\text{VoltageRange (Volts)}}{\text{Gain} \times 2^{\text{BitResolution}}} \quad (\text{eq. 4.1})$$

In recent years, a new generation of transducers have been investigated and developed. These devices were named as “smart transducers”. Ranky [4.12] described the term smartness as “on-board data storage/processing capability, interfaced/integrated with the analogue and/or digital sensor”. Such devices should also be connected to a single bus, representing a shared medium where appropriate communication protocols would be used to deliver data in a digital format. The IEEE 1451 family of standards [4.13] provides guidance for the development and use of smart transducers, including a network interface specification and device characteristics, such as calibration data, sensitivity, measurement range and manufacturer’s identification [4.12]. Ultimately, it is expected this sort of device be capable of providing self-calibration, self-compensation, self-validation and of communicating by means of digital networks, to deliver the measurement data [4.14].

The selected PIC microcontroller was recognised as being a solution capable of providing most of the general data acquisition requirements. Although presenting restrictions, mainly due to the general context of the application, the benefits represented by the versatility of the configurable range of embedded functions and network based communication support, suggested an attractive low-cost alternative that is capable to overcome the restrictions identified.

4.4 – Industrial Networks

As stated before, one of the major benefits of using the PIC device was its networkability. The technology review provided by this chapter therefore continues with a brief and general review of industrial networks, followed by a more detailed description of the protocol used in the monitoring system implementation.

The development of electronic devices with networking capabilities has been manifested, for example, in distributed control systems [4.15]. In simple terms, in such an environment, transducers with processing capabilities are deployed distributed across the system. As a result, the way in which the transducers communicate with the main control elements, such as Programmable Logic Controllers (PLCs), has been enhanced by the use of industrial networks. Jacob et al [4.16] classified these networks in two ways: sensor buses and Fieldbus. They considered that both are suitable for sending control and measurement signals, however Fieldbus was evaluated as being more structured and capable of making distinction between different classes of messages. Although many different specifications have been proposed and implemented, all of them aim to provide a digital network that enables communication between control and measurement devices sharing a single bus [4.17]. Figure 4.7 illustrates such an environment. The concept was utilised in the current research in order to provide flexibility in the monitoring structure.

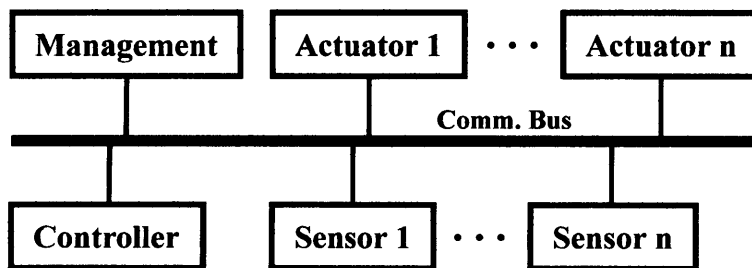


Figure 4.7: A distributed control system based on industrial networks.

A benefit, reported by many authors, resulting from the use of industrial networks was the reduction of cabling costs in the automation field [4.12, 4.17, 4.18, 4.19]. Other benefits accrued when upgrading, updating and replacing devices, since less effort would be required with the devices being easily connected and configured [4.12, 4.17, 4.19]. However, concerns have been manifested by the same authors, regarding the considerable range of specifications proposed and implemented, making it difficult to establish a common standard governing the subject. A summary of the main existing implementations can be found in [4.20].

The Open Systems Interconnection (OSI) model of the International Standards Organisation (ISO) provides a reference for the implementation of modern communication systems, based on a 7 layer protocol model, as shown in Figure 4.8(a). Tanenbaum [4.21] considered that this model provided a useful framework for the discussion of computer networks. Nevertheless he added that the 7 layer model is very complex, difficult to implement and has many functions repeated at different layers. Therefore, depending on the implementation, low efficiency might be expected. In considering all these factors and also assuming that generally there is no need to implement complex network functions in the control field, Fieldbus specifications are generally based on a lower number of layers [4.19, 4.22], as seen in Figure 4.8(b). The reduced layer model is normally referred to as an Enhanced Performance Architecture (EPA) and considers the real-time requirements of control applications. Table 4.6 summarises the functions associated to each layer. Unfortunately no agreement has yet been reached with respect to a single Fieldbus standard and existing implementations have included more or less layers and functionalities [4.18, 4.22]. Within this research the subject was limited to the use of an existing standard (CAN bus) based on the reduced model, eased by the availability of electronic devices that provide the means to implement the protocol required functions. This will be presented later, in Chapter 7. CAN bus technology is introduced in the next section.

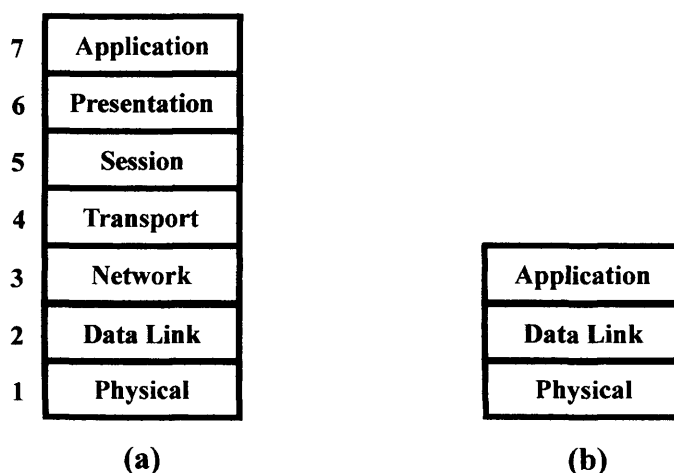


Figure 4.8: Communication networks (a) OSI and (b) reduced model.

Comparing the main existing specifications [4.20], many differences can be observed at different levels of implementations. In terms of the physical layer, although

twisted-pair is a supported medium by almost all specifications, different network topologies, signalling methods (digital coding), data transfer rates and electrical levels can be observed. Also, the maximum number of connected nodes and network length may vary. At the data link layer there are different media access arbitration methods, which are in many cases related with the network topology. The Cyclic Redundancy Check (CRC) is usually employed to verify the data stream integrity, although its length may vary from one specification to another. Also, the maximum size of the supported data transfer field varies accordingly the specification.

Table 4.6: Reduced model layer description (compiled from [4.19, 4.21, 4.22]).

Layer	Main functionality description
Physical	<ul style="list-style-type: none"> • Define electrical and mechanical characteristics of the physic interface and transmission medium.
Data link	<ul style="list-style-type: none"> • Provide medium access control; • Implement an error-free delivery mechanism; • Data stream synchronisation.
Application	<ul style="list-style-type: none"> • Provide communication mechanisms to exchange messages between network pairs (response to requests, interrupt driven, time cyclic, etc).

The communication method between network pair nodes also changes between specifications (client/server, peer-to-peer, master/slave, producer/consumer, multicast, etc), with some implementations making use of more than one method, in order to cope with priority and hierarchy requirements. In some cases the communication method does not obey the boundaries represented by the layer model, with some of the functionality associated to the data link layer and others to the application layer [4.18], depending on the provided services (cyclic messages, event based messages, request response, etc). Fieldbus characteristics were further summarised in [4.20]. A description of the characteristics of some specifications can be found in [4.18].

4.4.1 – Controller Area Network (CAN)

The CAN protocol was a result of application area requirements for a robust and fast serial communication technique. It was initially developed by Robert Bosch GmbH

and released in 1986, to be used in automotive automation systems [4.23, 4.24]. The CAN protocol first received the standard reference ISO 11898 in the early 1990s. It specifies the implementation of the two lowest layers of the OSI model (physical and data link), for a serial bus application. It became the basis for some fieldbus implementations, such as CANopen, DeviceNet and CANkingdom, which complemented the application layer [4.23]. Despite its initial purpose, many factors including simplicity, openness and also the number of CAN controller suppliers, encouraged its use in a wider range of applications, including many in the industrial area [4.23, 4.24, 4.25]. The family of PIC microcontrollers fully support this protocol, therefore making it the adequate choice for the implementation of the monitoring system resulting from this research.

Despite having only two defined layers, the protocol specification defines several functions related to each one, especially within the data link layer. Table 4.7 summarises this functionality. The physical layer for practical reasons is divided into 3 sub levels, each one concerned with specific characteristics and requirements of the entire layer. The Medium Dependent Interface (MDI) and Physical Medium Attachment (PMA) have separate specifications from the Physical Signalling (PLS). There is also more than one specification for the MDI/PMA, depending on the application requirements and the body responsible for issuing the specification. Table 4.8 summarises the position with respect the existing specifications. One of the most widely accepted is the ISO 11898-2 (high-speed). It recommends a bus topology based on a pair of wires (CAN_H and CAN_L), with a signalling method based on the pair differential voltage (0 V for recessive polarity and 2 V for dominant), as illustrated in Figure 4.9. It also specifies the use of bus terminator elements, in order to reduce reflection effects. Bus transceivers are equally specified, to provide compliance guidance. Each transmission node reads the output back from the bus. Due to the overall philosophy, timing became an important issue in protocol implementation. Therefore, the specification includes a relationship between data rates and the bus length. The maximum baud rate is 1 Mbps, for a length of no more than 40 meters. A length increment can be achieved by reducing the data transmission rate, with maximum limit fixed at 1 Km. At the bus level, the maximum number of nodes is dependent on the employed transceiver characteristics. However, specifications for repeaters are included in the ISO 11898-2, in order to permit

deployment flexibility and to provide galvanic isolation. Other mediums and application restrictions were provided by specific standards [4.26].

Table 4.7: CAN protocol layers and functionality [4.26, 4.27].

Layer	Sub-division	Functionality
Data Link Layer (DLL)	• Logic Link Control (LLC)	<ul style="list-style-type: none"> • Acceptance filtering; • Overload notification; • Recovery management.
	• Medium Access Control (MAC)	<ul style="list-style-type: none"> • Data en-/de-capsulation; • Frame coding (stuffing/de-stuffing); • Medium access management; • Error detection and signalling; • Acknowledgement; • Serialisation/de-serialisation.
Physical Layer (PL)	• Physical Signalling (PLS)	<ul style="list-style-type: none"> • Bit encoding/decoding; • Bit timing; • Synchronisation.
	• Physical Medium Attachment (PMA)	<ul style="list-style-type: none"> • Transceiver characteristics;
	• Medium Dependent Interface (MDI)	<ul style="list-style-type: none"> • Cable/connector.

Table 4.8: Medium attachment specifications [4.26].

Application Specific	Standard / Specification
CAN high speed	ISO 11898-2
CAN low speed	ISO 11519-1
Fault tolerant transceiver	ISO 11898-3
Truck/trailer transceiver	ISO 11992
Single wire	SAE 2411
Fibre optical transmission	Proprietary solutions
Wire-less transmission	Proprietary solutions
Power-supply transmission	Not commercially available

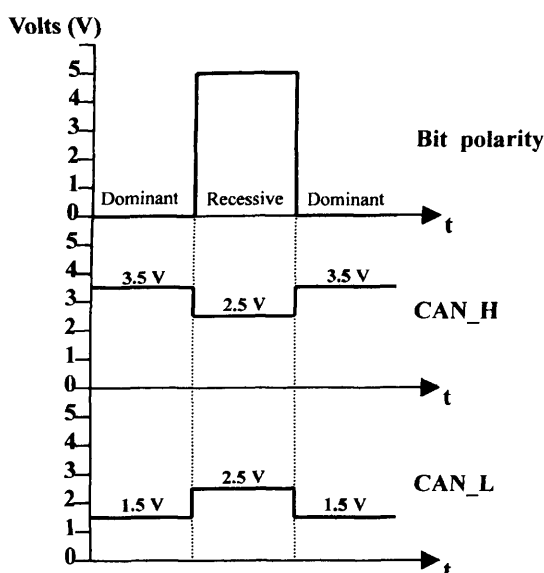


Figure 4.9: CAN bus electrical levels.

The PLS subdivision is specified in ISO 11898-2. In particular, it defines Non-Return-to-Zero (NRZ) as the signal coding method (Figure 4.9). Benefits represented by the method are a larger signal bandwidth and a better immunity to noise [4.18, 4.26]. However, it may result in a lack of synchronisation at the receiving end, when a large sequence of bits with same polarity is transmitted. Therefore, the bit stuffing technique is employed, resulting in the insertion of a bit with inverse polarity after a sequence of five consecutive bits of equal polarity (bit stuffing is implemented within the MAC subdivision). The protocol proposes that each bit should be considered as 4 non overlapping segments, each one being defined as a number of discrete units of time called time quantum (tq), representing the smallest timing resolution used by a CAN node. Such a scheme is supposed to enable the CAN controller to provide synchronisation and compensate signal delays and phase errors. The protocol is considered to be time critical and therefore it requires methods of synchronisation and compensation, which keep the relation with transmission rates and bus length. Each node has to provide resynchronisation [4.26]. Besides the summarised existing standards, detailed information concerning the implementation of the concepts referred to, can be found within the CAN controller's data sheets [4.28], provided for the Microchip devices.

The standard reference ISO 11898 provides the specification for the CAN protocol Data Link Layer (DLL). As was shown in Table 4.7, this layer has its services organised into two subdivisions: the Logic Link Control (LLC) and the Medium Access Control (MAC). The protocol is message-based, which means that messages flowing on the bus do not have an address field identifying the destination node. Rather, messages are provided with an identifier field, which each node has to check in order to decide whether or not to accept the received message. Messages are broadcast and all nodes can listen to them.

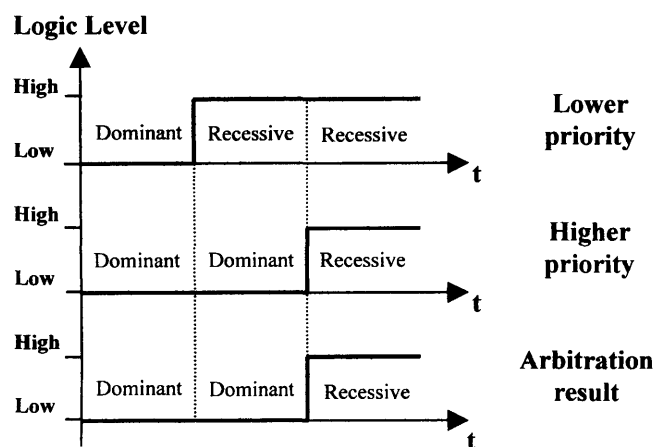


Figure 4.10: Bit-wise arbitration method illustration.

The MAC implements a bus access method based on Carrier Sense Multiple Access, with Collision Detection (CSMA/CD). It requires the node wishing to transmit to continuously listen to the bus to detect whether or not it is free. Different to other protocol implementations using the CSMA/CD technique, such as Ethernet (IEEE 802.3), a collision will not result in the destruction of the messages on the bus and thus the loss of the transmission time slot. With CAN, the message with the highest priority will remain intact, due to a bit-wise arbitration technique employed within the MAC. In order to implement the technique, bits are classified as recessive (1) or dominant (0), as illustrated in Figure 4.9. The higher the priority of a message, the lower its identifier field value will be. A collision will occur if two or more nodes start to transmit at the same time. Each node involved will transmit, while listening to the bus at the same time. If a dominant and a recessive bit transmitted by different nodes reach the bus at the same time, the result will be that the dominant bit prevails. The

node transmitting the recessive bit will detect the collision and immediately abort its transmission, preserving the integrity of the other, higher priority message [4.24, 4.27]. Figure 4.10 summarises the bit-wise operation implemented by the protocol.

Two communication services are defined by the CAN protocol: write object (transmission) and request object (transmission request). These services are structured, respectively, as “data frame” and “remote frame”. The first one is employed by a node, the producer, to broadcast data on the bus, which may be used by the listening nodes (the consumers). The second is employed by a consumer to request a specific message, resulting in producers owning the required data to transmit it utilising a data frame. The serialisation and de-serialisation of the frames, including bit stuffing, is performed within the MAC. Figure 4.11 provides a general view of the overall frame structure. The Start Of Frame (SOF) is a dominant bit that serves to synchronise all nodes listening to the bus. The following field, the “arbitration field”, basically contains the message identifier (priority scheme), a frame format identifier (standard or extended), and a frame type indicator (RTR - remote transmit request). The field is defined with two different lengths, which are dependent on the protocol version. Version 2.0A handles frames with 11 bits message identifiers (standard). Version 2.0B handles both, the 11 bits identifier and 29 bits long identifiers (extended). Extended frames enable a much larger number of different messages identifiers, when compared with the standard frame. Version 2.0B is sub-divided into active and passive. Active is fully capable of receiving and transmitting standard and extended frames. Passive receives both, but transmits only standard frames [4.27, 4.29]. In considering the requirements of the protocol implementation in the current research, the standard frame format (Version 2.0A) was employed.

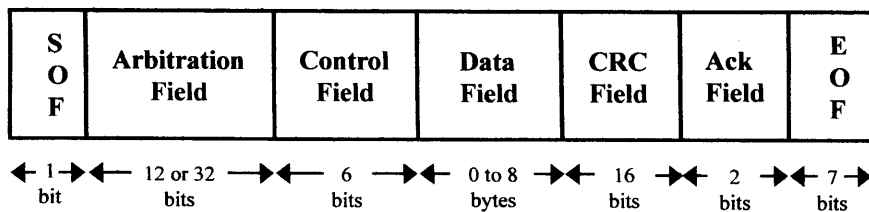


Figure 4.11: CAN frame structure.

Next in the “data frame”, the “control field” provides the Data Length Code (DLC) via a 4 bits coding that identifies the number of bytes in the data field. Slight differences are observed between the “standard” and “extended” identifier frame formats within this field. The control field of a standard format frame includes a reserved bit and an Identifier Extension (IDE) bit. In the extended format, the IDE becomes part of the arbitration field (which changes in size from 12 to 32 bits) and the bit slot left is defined as a second reserved bit [4.24, 4.27].

The “data field” then contains the data message transmitted by the producer. The number of bytes in the field may vary from 0 to 8 and is defined by the DLC (in control field). It is suggested that a frame with no data bytes (DLC = 0) might be desirable in some cases where the message identifier is enough [4.27].

The Cyclic Redundancy Check (CRC) field is employed to validate the transmitted message at the recipient end. The field reserves the first 15 bits for the CRC and the last one as a CRC delimiter (recessive). The CAN protocol employs a polynom of order 15, which is considered well suited for sequences with less than 127 bits [4.27]. Other data transfer protocols, such as Ethernet, employ a similar method, however with polynom generators of different order. The method is considered suitable for implementation in hardware [4.30], and is therefore normally embedded within controllers [4.28]. CRC calculation includes SOF, arbitration, control and data fields. All nodes on the bus calculate the CRC of the received sequence and compare it against the one sent by the transmitter. Whenever detecting a CRC failure, the node sends a “frame error” to make it public and thus request a retransmission. Both data and remote frames are provided with the field.

The “acknowledgement field” (Ack) follows the CRC in the frame structure. It is two bits long, with the first transmitted as recessive (logic high). This provides a bit slot used by the consumer nodes to acknowledge the message reception, by transmitting a dominant bit. Therefore, messages confirmation does not require further delay. A message confirmed ensures that at least one node received it correctly. The second bit represents an Ack delimiter also transmitted as recessive [4.27].

An End Of Frame (EOF) field closes the CAN frame. It is represented by a sequence of 7 recessive bits. This field is to provide a slot for error notification within the message length. After a frame, a 3 bits long intermission field is required to separate contiguous frames. These 3 bits are specified as recessive.

Error detection and reaction mechanisms are also specified within the MAC. Any node detecting an error condition (CRC, bit stuffing, framing error) will react by sending an error frame immediately, overwriting the message. This frame consists of an “error flag field” (6 dominant bits) and an “error delimiter” (8 recessive bits). The bit stuffing rule does not apply in this case and therefore all nodes listening to the bus, including the message producer, will detect it. As a result, the producer will restart the transmission (after an intermission) and all other nodes will discard the sequence that was received [4.27].

The Logic Link Control (LLC), the second sub-division of the DLL, provides services concerning message filtering, overload notification and recovery management. Message filtering corresponds to the ability of each node to decide whether or not to accept a received message, based on the identifier in the arbitration field. Overload notification, which is based on “overload frames”, is specified as a mechanism that enables a node to notify its pairs to delay the next transmission, in order to enable the node to sort out the internal conditions that caused such a state. Overload frames are similar to error frames in terms of structure. However, their transmission does not overwrite a message on the bus and therefore they do not result in retransmissions [4.27].

The LLC also considers error control features, thus enabling a management mechanism. Receive Error Counter (REC) and Transmit Error Counter (TEC) should be incremented by the detection of the respective errors and decreased in case of successful events. Based on the counters values, different states were defined. The initial and normal operation state is the “error active”, which allows a node to participate in the bus communication, including the right to transmit error flags. Such a state requires REC and TEC below 128. A counting error greater than 127 (either REC or TEC) forces the node into an “error passive” state [4.24]. In this state, instead of a normal error flag (also active error flag), the node is allowed to transmit passive

error flags (sequence of 6 recessive bits). Since based on recessive bits, error passive receivers cannot stop other transmitters, although it can still interrupt its own transmission [4.27]. A “bus-off” state is the most critical in the system and results in the suspension of the communication (reception and transmission) by the faulty node. Such a state is reached whenever the TEC is greater than 255. The protocol specification defines that bus-off recovery to an error active state should be performed by a software-reset procedure, ensuring 128 x 11 recessive bits on the bus before resuming transmission. Any hardware reset should be avoided, since it would not provide such a mechanism [4.27].

The CAN protocol specification, with a detailed description, also including recommendations and guidance for controller and driver implementations, are found in [4.26, 4.27, 4.29].

4.4.2 – MCP2510 CAN Controller

The CAN protocol provides the basic network infrastructure required to support the implementation of a distributed system. An important factor considered when selecting this industrial network protocol was its acceptance for industrial application [4.25, 4.31], thus indicating robustness; and the continuity of the technology, ensured by the existence of a diversity of suppliers [4.20]. Among these suppliers, Microchip provides the MCP2510, a CAN bus controller fully compatible with the PIC18C452, easing hardware interfacing and software implementation aspects. This section will introduce the main characteristics of this CAN bus controller, thus providing the basis for the implementation aspects considered later in Chapter 7.

The Microchip MCP2510 CAN controller is fully compatible with the protocol versions 2.0A and 2.0B and is capable of transmitting at 1Mbps. The device provides all the functionality described in the protocol LLC, MAC and PLS. A Serial Peripheral Interface (SPI) is embedded to enable the system processor to interact with the controller (data transfers and commands).

The device is provided with 2 receive and 3 transmit buffers, each one with the individual set of registers required to build/recover the protocol frame’s fields. The

transmission of messages already mounted in the respective buffers can be triggered either by hardware or software actions. Registers provide dynamic program acceptance filters and masks, used by the controller to selectively load messages from the bus. In the particular case of the masks, these indicate (by configuration) which bits of a received message identification field should effectively be considered when matching it against the programmed filters.

Different interrupt capabilities were also implemented in the controller. Each reception buffer was provided with an individual hardware line to generate a “buffer full” interrupt request. Such functionality is programmable. A general interrupt signal was also provided. This sort of interrupt request further requires a status register to be read, in order to identify the interrupt source. The controller interrupt sources are listed in Table 4.9. Interrupt conditions have to be removed by servicing the source (unloading a reception buffer) or by acknowledging it (others). Interrupt sources can be individually enabled. A full description of the MCP2510 CAN controller, including hardware and software programming features, can be found in the device documentation [4.28, 4.32].

Table 4.9: MCP2510 CAN controller interrupt sources description [4.28].

Interrupt Source	Description
Message error interrupt	<ul style="list-style-type: none"> Indicates an error during the transmission or reception of a message.
Wakeup interrupt	<ul style="list-style-type: none"> Indicates that the controller detected bus activity during a sleep state.
Multiple sources error	<ul style="list-style-type: none"> Indicates one of several sources, such as overflow, error passive, bus-off and warnings (error counters values).
Transmit interrupt	<ul style="list-style-type: none"> Indicates that a transmission buffer became empty, after a well succeeded transmission. Each buffer is provided with its individual flag.
Receive interrupt	<ul style="list-style-type: none"> Indicates that a message was successfully received. Each buffer is provided with its individual flag

4.5 – The Internet Protocols

Another technology area that was considered in the context of this research was Internet connectivity. The following sections provide a review of the general protocol. This is complemented with an overview of the use of embedded Internet features presented in Section 4.6, including the use of PIC microcontrollers for this purpose.

The necessity of sharing computer resources in a co-operative way, led to the development of a software-based protocol suite during the 1970s. Although popularly known as TCP/IP, the set of Internet protocols actually consists of more than just these two components. They do however represent the most often employed protocols [4.33, 4.34].

Table 4.10: Internet main protocols (compiled from [4.33, 4.34]).

Layer	Protocol	Description
Network	<ul style="list-style-type: none"> Internet protocol (IP) 	<ul style="list-style-type: none"> Provides the required mechanisms to send an IP frame (datagram) from the originator to a recipient.
Transport	<ul style="list-style-type: none"> Transmission control protocol (TCP) User datagram protocol (UDP) 	<ul style="list-style-type: none"> Connection-oriented transport protocol; Flow control services; Error-correction mechanisms. Connection-less transport protocol; Does not provide error-correction and flow control mechanism; High performance, with low overhead.
Application	<ul style="list-style-type: none"> File transfer protocol (FTP) Network terminal protocol (TELNET) Simple message transfer protocol (SMTP) Hyper-text transfer protocol (HTTP) 	<ul style="list-style-type: none"> Provides means to exchange files with another computer. Enables to establish a session in a remote computer (remote login). Electronic mail system, to move mail around the Internet. Access data stored in web servers, as web pages.

Basically, the set of protocols that form the Internet standard are classified in 3 groups, organised in a layered stack. The lowest layer in this stack is the internetworking layer. In the middle, there is the transport layer and the top layer

corresponds to the one defining application protocols, also known as TCP/IP services. Table 4.10 summarises the main existing members of the set of protocols. Special protocols, for support and management, are listed in Table 4.11. Internet protocol standards are normally referred to as RFC (request for comment) numbers. A collection of RFCs is found in [4.34] and updated lists are available on the Internet [4.35].

Table 4.11: Internet management protocols (compiled from [4.33]).

Protocol	Description
Address Resolution Protocol (ARP)	<ul style="list-style-type: none"> Provides means to interrogate the physical layer in order to locate the hardware address of an IP node on the network.
Internet Group Management Protocol (IGMP)	<ul style="list-style-type: none"> Provides support for multicast events. Although transmitted within an IP datagram, it is not considered as a transport layer protocol.
Internet Control Message Protocol (ICMP)	<ul style="list-style-type: none"> Provides a mechanism to notify the incapacity of the network to deliver a datagram to the destination.

Taking as reference the OSI model for network communication systems (Figure 4.8), the IP protocol can be considered to be equivalent to the OSI's network layer. TCP and UDP implement functions specified in the OSI's transport and session layers. In the same way, functionality specified in the OSI's application and presentation layers were implement within the Internet application protocols. The lower layers of the OSI model were not specified as Internet protocols. Rather, they rely on the communication hardware within systems to provide such services. An example is Ethernet (IEE802.3), which provides the physical and data link layer implementations of the OSI reference model.

4.5.1 – The IP Protocol (Network Layer)

The term “datagram” is used to designate the data and the header encapsulating it, at a specific layer. Communication is implemented by exchanging datagrams between pairs at the same level, throughout the network. IP provides such functionality at the

network layer. The protocols located at the layer immediately above use the IP services to exchange their datagrams. IP adds and removes (depending whether transmitting or receiving) the protocol header. Figure 4.12 shows the header structure.

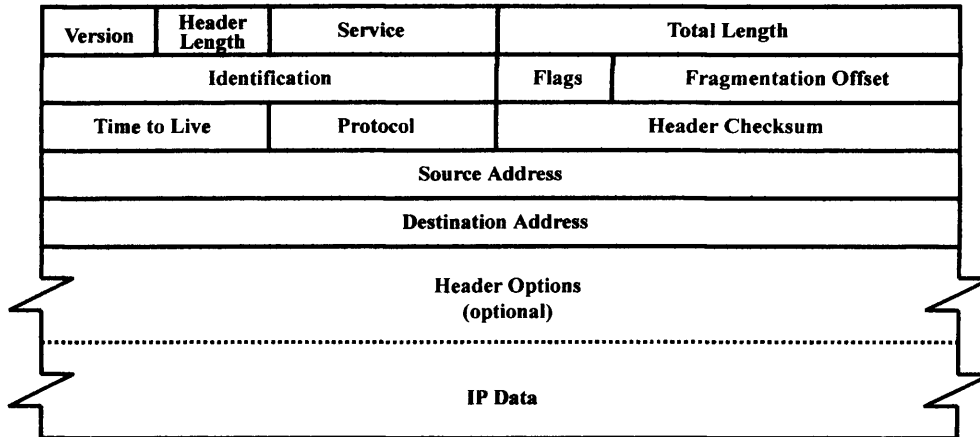


Figure 4.12: IP protocol datagram structure [4.30, 4.34].

The first field in the header provides the IP version, required by other network members to ensure compatibility. The header length provides the datagram header size, including the header options. Such options may be used in some cases to add special routing conditions or security requests. The service field provides priority information for routers and hosts. The total length field informs the datagram size (header and data). It may vary from 21 to 65,535 bytes. The identification field provides further information that enables the original datagram to be reassembled, in case it was fragmented due to network constraints. The flag field contains details regarding any fragmentation or to request that such a sort of action should be avoided. The fragmentation offset indicates the range of bytes in the specific fragment. The time to live parameter indicates to the network how many hops the datagram should take before considered undeliverable. The field is updated each time the datagram is handled. The protocol field is the identification of the higher layer protocol requesting (or provided with) the IP service. The checksum field incorporates the header validation information used by the other nodes along the network dealing with the message to detect data corruption. Source and destination addresses represent respectively, the originator and the final destination of the datagram. The IP actual version provides address fields 32 bits long. Due to increasing use and new addressing

requirements, IPv6 is under development, in order to provide a much wider address field (128 bits) [4.21]. Finally, the data field contains the higher layer protocol datagram. Nevertheless, when fragmentation has occurred, it may contain just a part of the original message. The IP header is required to have its length multiple of 32 bits. Whenever it is not, due to the use of options, bit padding is required to complete the missing bits [4.30, 4.33].

Each Internet connection is provided with an IP address. IP provides the higher layer (transport) with a virtual network, hiding all the complexity associated with internetworking. Whenever a message transmission is required, the “transport protocol” hands to the IP protocol a “transport layer datagram” (presented in following sections) and the destination address. A header is added to the data, thus resulting in the IP datagram, which is forwarded to the following layer in the network hierarchy, usually provided by the local network system (e.g. Ethernet). A table relating IP addresses and the network hardware addresses is required, in order to enable the local network system to deliver the message properly. Such a table is dynamically implemented, based on previously received messages or employing ARP requests. If the destination address is not within the local network, the IP datagram has to be “routed” across the network, permitting it to reach its final destination. Therefore, the IP address becomes important and must to be unique within the environment. IP addressing fundamentals are discussed in detail in [4.33]. At the destination end, the IP header is removed and the data field delivered to the transport protocol. The protocol field in the IP header provides this information [4.30, 4.33, 4.34].

Although using IP all over the network, the Internet is a set of sub-nets that may have different characteristics. Such characteristics may be related with the medium or local network protocols (data link and physical layers). Therefore, the original IP datagram may have to be fragmented into smaller packets, in order to get across the entire network. To enable the final destination to reassemble the original datagram, the IP header provides the fragmentation flags and offset fields. The implementation of the protocol requires a much deeper analysis of the aspects here considered. Detailed information regarding the IP protocol can be found in [4.33].

4.5.2 – The TCP Protocol (Transport Layer)

One of the most complex components in the Internet suite of protocols is TCP. With IP providing delivery service, this protocol, located at the transport layer, provides the application with a service that represents reliability and control. As observed with IP, TCP applies a header to the data received from the higher layer requesting the service, thus producing a TCP datagram. This is shown in Figure 4.13.

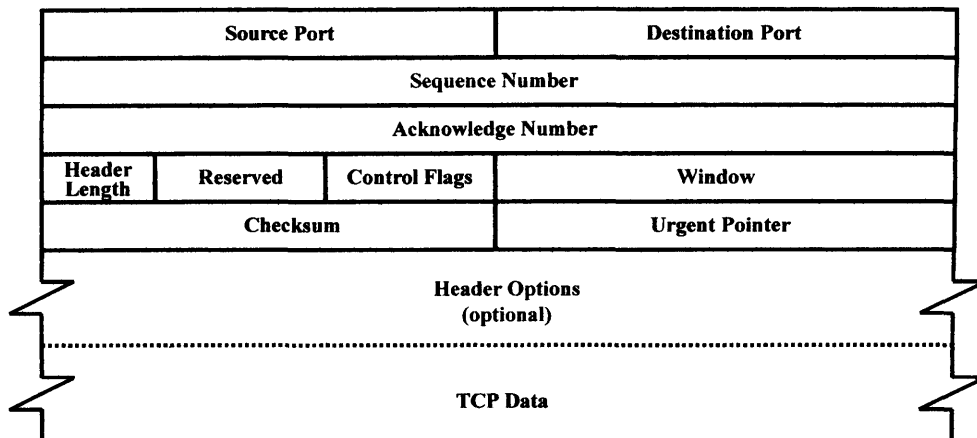


Figure 4.13: TCP protocol datagram structure [4.30, 4.34].

Since TCP can provide the same service to several applications, each application must be defined by a port number. Therefore, the source and destination port fields represent, respectively, the application requesting the service and the one receiving the message. The sequence number field identifies the first byte in the data segment contained by the datagram, therefore providing a means to reassemble the entire message at the final destination, in the right order. The complement of the mechanism is the acknowledgement number, used to confirm the accurate reception of a sequence of data. Different to the sequence number, instead of indicating the first byte in the data segment, the acknowledgement number confirms the reception of the data by indicating the next expected number (sequence number of the next datagram). The header length field specifies the size of the TCP header, including the optional field. Such options provide extended information to end nodes. An updated list of options can be found in [4.36]. The reserved field has no use at present and should always remain assigned to “0”. The control flag field provides data and virtual circuit

message control parameters. Some of these flags are “urgent”, for high priority data; “reset”, to abort and “finish”, which ends the virtual link. The window field dynamically informs the destination end-point about the amount of non-acknowledged data the sender is able to receive (available reception buffer). Checksum supplies the destination with the means to assess and validate the received data. In a different way to IP, the TCP checksum includes header and data field. The urgent pointer field acts as an offset within the data segment, indicating the end of urgent data. It is used in conjunction with the urgent flag in the control flag field to indicate an urgent datagram. The urgent pointer is represented as a relative value and must be added to the sequence number to obtain the absolute pointer. Finally, the data field holds the application data. It is possible TCP datagrams being used to exchange only control information, therefore without any data. Padding may be required, in order to make the header size multiple of 32 bits.

TCP is a connection-based protocol. It means that when an application requests a data exchange service, TCP starts a negotiation procedure between end-points in order to establish a virtual circuit. Such virtual links must be unique. In this initial phase, the TCP protocol at both ends exchange empty datagrams (no data in the data field), with appropriate settings in the control flag field, to synchronise the virtual circuit and therefore initiate the “session” between end applications. In the next phase, where applications are able to exchange data, TCP performs several tasks that aim to provide an efficient and reliable service. The protocol must be able to manage errors, duplication, lost and out of order data [4.33].

The application data handed to TCP is, if required, broken down in smaller bits, in order to benefit from network overall throughput (buffer size, physical medium capability, data block size). Special techniques may be provided by the protocol implementation to detect such a figure. These smaller segments are sequenced and handed to IP. The TCP header provides the means required to control the connection and the data transaction in a manner that facilitates the rebuilding of the original data stream, even if individual segments follow different routes in the entire network. The sender implements time control mechanisms, as a mean to retransmit data that apparently was not delivered. Received data is validated by the checksum mechanism. In order to enhance efficiency, the protocol enables the confirmation of



received data within a datagram transmitting data, using the acknowledgement field for this purpose. Also, more than one data segment can be confirmed at once (the acknowledge number confirms all previously received data). Flow control is provided by informing the remote end about the local reception capability, based on the window parameter [4.33].

In considering the dynamics of an environment such as the Internet, it is easy to perceive the complexity involved in the management of a reliable transport protocol, which requires tight control of the mechanisms. In order to achieve efficiency, hardware and software resources become very important. A detailed description of the TCP protocol and its management requirements can be found in [4.33]. Implementation issues are considered in [4.30].

4.5.3 – The UDP Protocol (Transport Layer)

Besides TCP, the Internet suite of protocols specifies a second transport protocol, UDP. Apart from the fact that both are located on the same layer in the model hierarchy, the way they provide their service is based on different concepts. One of the major differences is that UDP is connectionless and does not ensure delivery. Such characteristics can easily be identified when looking the protocol header, shown in Figure 4.14. The UDP header, as TCP, requires the source and destination applications identification, represented in terms of port numbers. The length field specifies the number of bytes that compose the message, including header and data. It cannot exceed 65,535 bytes. The checksum field is equivalent to the one provided in the TCP header and enables the remote end to validate the received data, although it is not compulsory and can be disabled by sending the field as “0”. The data field will contain the source application data.

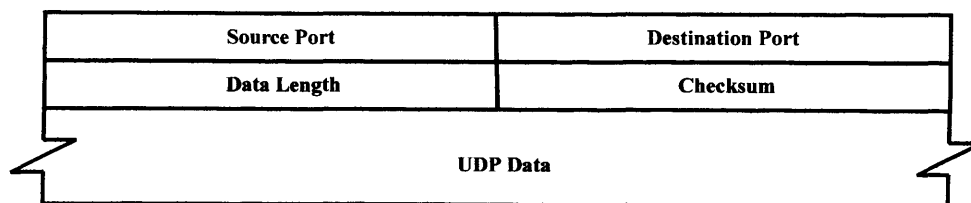


Figure 4.14: UDP protocol datagram structure [4.30, 4.33].

In technical terms, UDP provides a simple interface between the application and the IP protocol. The application requests a data transfer service, supplying the port numbers. The destination IP address is also required. UDP adds the header, calculates the checksum (if required) and requests IP's networking service. No previous transactions between end-points are carried out. The transmitted message is discharged from the transmitter buffer, since there is no commitment with delivery guarantee. The remote end simply forwards the data to the application declared in the destination port field. Messages that are not validated by the checksum matching are not considered.

Benefits in using the UDP protocol are realised due to its simplicity. A very low overhead is introduced by the protocol, when compared with TCP. Messages are transmitted on demand, without establishing a virtual circuit. Asynchronous event systems may specially benefit from such simplicity. This was taken into consideration in the implementation of the system that resulted from this research, since the monitoring events assume an asynchronous characteristic and the resulting records are short and of a fixed size. However, UDP does not guarantee that the data is received. This must be externally addressed and will be considered later in Chapter 7. Further details related with the protocol and its implementation can be found in [4.30, 4.33].

4.5.4 – Management Protocols

To support the main task of the Internet protocols, the data exchange in a controlled environment, a few other protocols are required. One such protocol is Address Resolution Protocol (ARP, Table 4.11), which is mainly employed as a tool to relate the Internet addressing model with those addresses used on the local network. The protocol packet structure is shown in Figure 4.15. Within the ARP packet, each medium type has a unique identification, declared in the hardware type field. The protocol field identifies IP, when Internet protocols are used. The hardware address field provides the number of bytes used to represent the local network hardware-address. It is required due to the fact that IP (network layer) can be used with different data link and physical layers, in different systems. The protocol address length is 32 bits, when using IP. The source hardware address is a variable field that is supposed to represent the local network hardware-address of the source node. The source protocol

address is equally variable, although when using IP it will be 32 bits long. Similarly defined are the destination addresses (hardware and protocol).

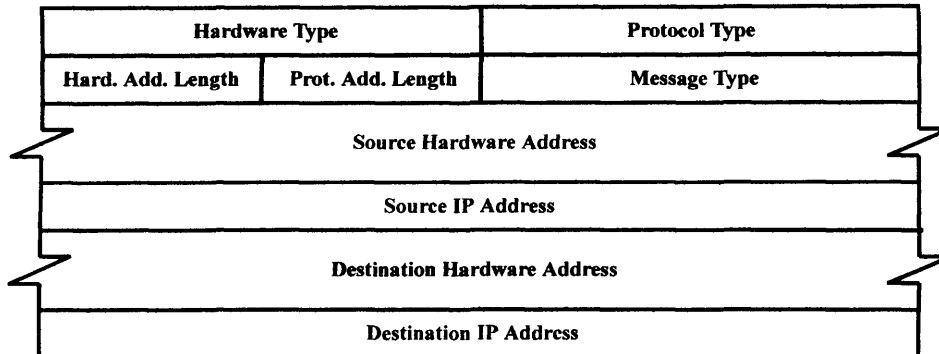


Figure 4.15: ARP packet structure [4.30].

When an application requests a data exchange service that involves a remote location whose hardware address is unknown, the ARP protocol is required to build a table relating the IP and hardware addresses. ARP requests that a message is broadcast on the local network. All the fields in this message are set accordingly with their meaning, except the unknown hardware address (destination), which is sent as “0”. Such a packet is called as ARP request and is identified by the request code in the message type field. The node on the local network that possesses the broadcast IP address will reply with an ARP response packet providing the hardware address of interest. Further considerations are required when considering the dynamic allocation of IP address or the identification of hardware addresses of IP pairs located on remote networks [4.33].

There are three ways in which IP communicates on the network with other pairs: unicast, which involves two end nodes; broadcast, which involves all nodes; and multicast, aimed to send messages to a selected group of nodes (hosts). Multicast, differently from broadcast, can have several groups, with distinct interests. This requires the management of such groups. The Internet Group Management Protocol (IGMP) was provided for this task. By using IGMP, network hosts inform other network pairs (servers or routers) of their interest in participating in specific groups. The opposite is also based on IGMP, when routers search for specific group

memberships. Membership reports are specified within the protocol. A detailed description of the protocol is provided in [4.33].

In considering that IP does not provide any support in order to detect delivery (or network) errors, there was a requirement for a protocol capable of such notification. The Internet Control Message Protocol (ICMP) was thus provided and, in conjunction with IP, provides the network nodes with the ability to diagnose and reply error messages to the IP datagram originator. The ICMP header format is shown in Figure 4.16. The header type field indicates a specific ICMP error. The code field further describes the error within a subclass. The checksum provides a mechanism for remote message validation. Data is a field used within the header for specific message needs. As an example, it could return the IP address of a preferable router. It is said to be not in very common use [4.33]. The “original header” field will reply with the header of the original IP datagram that could not be delivered. The “original data” field will contain the source and destination port addresses, part of the transport protocol header of the originator message.

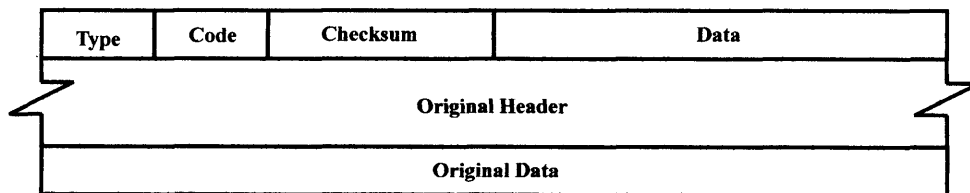


Figure 4.16: ICMP packet structure [4.30, 4.33].

ICMP messages are sent within IP datagrams, with the IP protocol field set to 1. Although in many cases delivery problems may be related with the IP addresses and routing, there may be cases in which this is related with the remote application that for some reason becomes unreachable. Therefore, the ICMP header provides a complete set of information, to help the originator to identify the reason of the verified error, avoiding further attempts or deciding for other means.

An implementation that uses ICMP is echo request, usually known as “ping”. Such an implementation aims to test the network by sending an echo-request to a specific IP address, using an ICMP message. The destination, if reachable, responds with an

ICMP echo-reply. Among other explicit uses of ICMP, it is employed to fix the best system setting, such as largest datagram supported all over the connection. Hall [4.33] provides a detailed description of the protocol and its application. Implementation details were described in [4.30].

4.6 – Internet Embedded Technology

Recent developments in Internet technology have stimulated the investigation of new application areas, enhancing its use, mainly concentrated on personal computers. Agranat [4.37] suggested that web browsers became the de facto standard user interface for a variety of applications, replacing “traditional proprietary command-line and graphical user interfaces”. Nath [4.38] considered the development of applications, such as home appliances, remote monitoring and industrial control systems, that could in some way benefit from the deployment of Internet technologies. Requirements for embedded applications would be low-cost, low-power consumption and size constrained implementations [4.38].

Agranat [4.37] when analysing the implementation of Internet capabilities within embedded applications raised the point that usually resources are not available in large scale, as they are in modern desktop computers, thus imposing difficulties that should carefully be considered. A similar view was expressed in [4.39]. Singer [4.40] discussed the requirements for the implementation of the full set of Internet protocols within embedded devices, considering the integration of existing implementations (hardware devices or software libraries). Usual implementations include the Internet transport and network layers, normally referred to as TCP/IP stack [4.38]. The data link layer of the reference model in many cases was based on the Point-to-Point Protocol (PPP), suitable for use with serial links such as modems. This sort of support makes such implementation a good option for isolated applications. Some implementation examples based on devices with limited resources provide a good idea of the level of complexity involved [4.39, 4.41, 4.42].

Microchip [4.6] developed an Internet Development Kit based on the PIC family of microcontrollers, which provides Ethernet capabilities on board. More recently,

Dallas [4.43] announced a microcontroller embedding TCP/IP stack, Ethernet and CAN bus interfaces, indicating that low-cost and Internet capable industrial applications are becoming a reality. A detailed analysis of some up to date existing solutions was provided by Eisenreich and DeMuth [4.44].

4.7 – Database Systems

The remaining distributed monitoring system features to be briefly reviewed in this chapter relate to the exchange and integration of information based on database systems. Whittington [4.45] considered that information systems “provide a resource that enables organizations to operate more effectively”. This larger use of information systems requires database systems capable of providing fast responses and multi-user access. This will be the case for online monitoring systems.

Blahe [4.46] described a database as “a permanent, self-descriptive store of data that is contained in one or more files”. Therefore, the database contains the data structure (description of data) and the data, where the latest one represents a dynamic element. Modern applications are usually based on relational databases [4.46]. In such databases, the data is organised in tables reflecting the relation defined among the data [4.45]. Relations may also be established between different tables within the database. To enable the manipulation of data within relational databases, the Structured Query Language (SQL) was developed and represents a common language supported by most of the commercially available Database Management Systems (DBMS). SQL commands operate on database tables, using them as input sources and output targets. However, SQL commands specify properties and characteristics of the desired data, rather than determine how to access the data, which is an attribution of the DBMSs. SQL commands can be used interactively or embedded within the application programming language [4.45].

DBMSs provide access control to databases. As illustrated in Figure 4.17, a DBMS represents an interface through which all data definition and manipulation is carried out, releasing application software from such task. DBMSs are available as commercial applications and were considered “mature and reliable” [4.46]. Among

the main functionality provided by a DBMS are data validation, sharing, manipulation, analysis and security. The last one could be further classified in physical data protection (backups, log of activities and transaction completion) and logical (data access control) [4.45, 4.46].

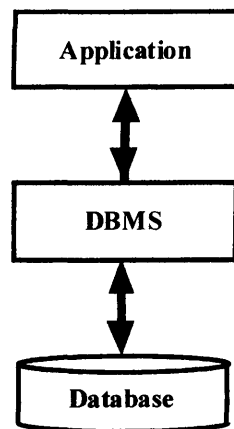


Figure 4.17: Database approach relations.

Although data manipulation is largely eased by the use of SQL, the way in which the data is accessed varies among different DBMSs. Differences may be found in the provided data types, error codes, security approaches and occasionally different SQL syntaxes [4.47]. In considering the requirements of modern applications, such aspects would represent a limitation, requiring many different DBMSs interfaces implemented within an application. Open Database Connectivity (ODBC) was proposed to solve such problem, by defining a standard that enables an application to use a unique interface and SQL syntax. Data types and error codes should also be unique [4.47]. Therefore, it is up to the DBMS manufacturer to supply a software driver that complies with the ODBC standard.

Due to their special abilities in manipulating high quantities of data with efficiency, giving simultaneous access to a large number of users and providing data security, database systems have had their use increased with the growth of the Internet. Therefore SQL based database systems were considered the natural choice in this research for the storage of the monitoring system records, easing data analysis and presentation approaches and enabling the integration of monitoring information in

other applications. This is considered in Chapter 7, where the system’s “management application” is introduced.

4.8 – Summary

A number of factors have been responsible for enabling the development of new and flexible applications. The electronic industry is constantly yielding new devices with higher levels of integration and increasing processing power. The development of distributed systems technology greatly benefits from the existence of such empowered devices. Efficient communication protocols became a requirement to enable the integration of intelligent devices in such a distributed environment. At the same time, information is a vital resource in modern management practices and the possibility of integrating the information provided by different systems is further supported by the capability of such systems to connect to the Internet and access databases, thus establishing a common interface.

In order to benefit from such technological enhancements in the context of this research, a modelling technique is required. This will be presented in the following chapters, beginning with the introduction of the Petri-net general concept.

REFERENCES

- 4.1 **Tanenbaum, A.S. and van Steen, M.** Distributed Systems – Principals and Paradigms. New Jersey, USA: Prentice-Hall Inc., 2002.
- 4.2 Brief History of Microprocessors, University of Teesside Web Site, Available from: <http://wheelie.tees.ac.uk/users/a.clements/History/History.htm> [Accessed 7 April 2003].
- 4.3 **Kochhar, A.K. and Burns, N.D.** Microprocessors and their Manufacturing Applications. London, UK: Edward Arnold Ltd., 1983.
- 4.4 **Baron, R.J and Higbie, L.** Computer Architecture. New York, USA: Addison-Wesley Publishing Company, 1992.
- 4.5 Microprocessors and Microcontrollers, ePanorama.net Web Site, Available from: <http://www.epanorama.net/links/microprocessor.html> [Accessed 7 April 2003].
- 4.6 Microchip Technology Inc. Microchip Products Web Site, Available from: <http://www.microchip.com/1010/search/prodsel/index.htm> [Accessed 14 April 2003].
- 4.7 Microchip Technology Inc. PICMicro® 18C MCU Family Reference Manual. USA: Microchip Tech. Inc., 2000.
- 4.8 Microchip Technology Inc. PIC18CXX2 Data Sheet. USA: Microchip Tech. Inc., 1999.
- 4.9 National Instruments. Application Note 007 – Data Acquisition Fundamentals. National Instruments Corporation Web Site, Available from: <http://www.ni.com> [Accessed 23 April 2002].
- 4.10 **Bolic, M., Drndarevic, V. and Samardzic, B.** Distributed Measurement and Control System Based on Microcontrollers with Automatic Program Generation. *Sensors and Actuators A*, 2001, 90,215-221.
- 4.11 **Nwagboso, C.O., Whomes, T.L. and Davies, P.B.** Considerations on the Development of Computer Aided Data Acquisition, Control and Analysis Systems (CADACAS) for Condition Monitoring Tasks. *Journal of Condition Monitoring*, 1989, 2(4), 243-268.
- 4.12 **Ranky, P.G.** Smart Sensors. *Sensor Review*, 2002, 22(4), 312-318.

- 4.13 **Lee, K.** IEEE 1451: A Standard in Support of Smart Transducer Networking. *In Proceeding: IEEE Instrumentation and Measurement Technology Conference*, 1-4 May, Baltimore – USA, 2000, 525-528.
- 4.14 **Tian, G.Y., Zhao, Z.X. and Baines, R.W.** A Fieldbus-based Intelligent Sensor. *Mechatronics*, 2000, 10, 835-849.
- 4.15 **Madan P.** LonWorks® Technology for Intelligent Distributed Interoperable Control Networks. Echelon Corporation Web Site, Available from: <http://www.echelon.com/solutions/opensystems/intelnet.pdf> [Accessed 21 April 2002].
- 4.16 **Jacob, P., Ingram, S. and Ball, A.** Fieldbus: The Basis for an Open Architecture Condition Monitoring Revolution. *Maintenance*, 1996, 11(5), 3-9.
- 4.17 **Hanzálek, Z. and Pácha, T.** Use of the Fieldbus Systems in Academic Setting. *In Proceeding: 3rd IEEE Real-Time Systems Education Workshop*, 21 November, Poznan – Poland, 1998, 93-97.
- 4.18 **Jordan, J.R.** Serial Networked Field Instrumentation. Chichester, UK: John Wiley & Sons Ltd, 1995.
- 4.19 **Atkinson, J.K.** Communication Protocols in Instrumentation. *Journal of Physics E: Scientific Instrumentation*, 1987, 20, 484-491.
- 4.20 Synergetic Microsystems Inc. Fieldbus Comparison Chart. Lantronix Web Site, Available from <http://www.synergetic.com/compare.htm> [Accessed 24 April 2003].
- 4.21 **Tanenbaum, A.S.** Computer Networks, New Jersey, USA: Prentice Hall PTR, 2003.
- 4.22 **Schumny, H.** Fieldbus in Measurement and Control. *Computer Standards and Interfaces*, 1998, 19, 295-304.
- 4.23 CAN in Automation (CiA). CAN History. CiA Web Site, Available from <http://www.can-cia.de/can/protocol/history/history.html> [Accessed 28 April 2003].
- 4.24 **Pazul, K.** Controller Area (CAN) Basics – AN713. Microchip Technology inc. Web Site, Available from <http://www.microchip.com/download/appnote/analog/can/00713a.pdf> [Accessed 9 August 2000].

- 4.25 **Thomas, G.** Proposed Network Hierarchy for Open Control. Contemporary Control Web Site, Available from <http://www.ccontrols.com/whitepaper.htm> [Accessed 13 September 2002].
- 4.26 CAN in Automation (CiA). CAN Physical Layer. CiA Web Site, Available from <http://www.can-cia.ru/CANphy.pdf> [Accessed 28 April 2003].
- 4.27 CAN in Automation (CiA). CAN Data Link Layer. CiA Web Site, Available from <http://www.can-cia.ru/CANdll.pdf> [Accessed 28 April 2003].
- 4.28 Microchip Technology Inc. MCP2510 Data Sheet. USA: Microchip Tech. Inc., 1999.
- 4.29 CAN in Automation (CiA). CAN Implementation. CiA Web Site, Available from <http://www.can-cia.ru/CANimpl.pdf> [Accessed 28 April 2003].
- 4.30 **Bentham, J.** TCP/IP Lean – Web Servers for Embedded Systems. Lawrence, USA: CMP Books, 2000.
- 4.31 **Scott A.V. and Buchanan W.J.** Truly Distributed Control Systems using Fieldbus Technology. *In Proceedings: 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, Edinburgh – Scotland, IEEE, 2000, 165-173.
- 4.32 Microchip Technology Inc. MCP2510 – Stand-Alone CAN Controller with SPI™ Interface. Microchip Web Site, Available from: <http://www.microchip.com/download/lit/pline/analog/interfcee/can/21291e.pdf> [Accessed 28 April 2003].
- 4.33 **Hall, E.A.** Internet Core Protocols – The Definitive Guide. Sebastopol, USA: O'Reilly and Associates, 2000.
- 4.34 **Hedrick, C.L.** Introduction to the Internet Protocols, Rutgers University, Available from <http://oac3.hsc.uth.tmc.edu/staff/snewton/tcp-tutorial/index.html> [Accessed 8 May 2003].
- 4.35 The Internet Engineering Task Force, Internet Society, IETF Web Site, Available from <http://www.ietf.org/rfc> [Accessed 9 May 2003].
- 4.36 Internet Assigned Numbers Authority, TCP Option Numbers, IANA Web Site, Available from <http://www.iana.org/assignments/tcp-parameters> [Accessed 12 May 2003].

- 4.37 **Agranat, I. D.** Engineering Web Technologies for Embedded Applications. IEEE Internet Computing Online Web Site, Available from <http://www.computer.org/internet/v2n3/w3agranat.htm> [Accessed 14 May 2002].
- 4.38 **Nath, M.** Low-cost Techniques Bring Internet to Embedded Devices. *EDN Magazine*, 1999, November 11, 159-166.
- 4.39 National Semiconductor, An Embedded Web Server for National semiconductor's CR16MCS9 CannonBall, National Semiconductor Corp. Web Site, Available from http://www.nationa.com/appinfo/cr16/files/embedded_web_server_CR16MCS9.pdf [Accessed 15 May 2003].
- 4.40 **Singer, A.** Internet Connectivity. *Circuit Cellar*, 2000, 123, 1-4.
- 4.41 **Eady, F.** An S-7600A/PIC16F877 Journey. *Circuit Cellar Online*, 2001, January, 1-4.
- 4.42 **Loewen, M.** Using PICmicro MCUs to Connect to Internet via PPP –AN724. Microchip Technology inc. Web Site, Available from <http://www.microchip.com/download/appnote/internet/00724c.pdf> [Accessed 20 January 2001].
- 4.43 **Tubb, L.** Dallas Semiconductors Offers 75MHz 8051 Microcontroller with 10/100 Ethernet. Microcontroller.com Web Site, Available from http://microcontroller.com/news/printer_dallas_8051_ethernet.asp [Accessed 16 February 2003].
- 4.44 **Eisenreich, D., DeMuth, B.** Designing Internet Embedded Devices. New York, USA: Newnes Elsevier Science, 2003.
- 4.45 **Whittington, R.P.** Database Systems Engineering. Oxford, UK: Clarendon Press, 1988.
- 4.46 **Blaha, M.R.** A Manager's Guide to Database Technology. New Jersey, USA: Prentice Hall Inc. 2001.
- 4.47 IBM Corporation, IBM Book Manager, IBM Book Server Web Site, Available from <http://publib.boulder.ibm.com/cgi-bin/bookmgr/FRAMESET/QBKACO03/APPENDIX1> [Accessed 8 March 2001].

CHAPTER 5

PETRI-NET CONCEPT

5.1 – Introduction

Petri-nets were proposed in the early 1960s by Carl Adam Petri, as a result of his investigation into a method to model and analyse the relationship between components of a system [5.1]. Since then, several investigations have been conducted aimed at making further use of the concept. Some of these have suggested changes and improvements to the original idea, in order to better represent real-life situations [5.1, 5.2].

Although representing a mathematical formalism, the main feature of Petri-nets is their capability of describing a system's behaviour in a graphical manner [5.2], making the approach suitable for many engineering applications. Figure 5.1 provides an example of such graph.

Characteristics such as concurrency, sequencing and synchronisation make Petri-nets a powerful tool for the representation and modelling of a variety of different and real discrete event systems. Analysis methods were also developed in order to validate such systems' models [5.1, 5.2, 5.3]. Most recently, reports have been made of the use of the technique for the development and implementation of Programmable Logic Controller (PLC) programming languages [5.4] and as a graphical programming method [5.5]. Earlier uses of the method were reported in relation to process and condition monitoring [5.6] and for the development of fault detection and isolation methods [5.7].

5.2 – Petri-net Representation

Referring to the graphical representation in Figure 5.1, Petri-nets elements are classified in places (circles p1, p2, p3, p4, p5, p6, p7 and p8), transitions (bars t1, t2, t3 and t4) and arcs (arrows a11, a13, a14, a22, a32, a43, a53, a26, a37, a64, a74 and a48). Places are elements representing the system states, transitions represent the system events and arcs define the relationship between places and transitions. Girault and Valk [5.3] considered places as passive elements, such as real-life conditions and resources. Following similar consideration, transitions were assumed as active elements of a system (events, actions and executions). The existence of conditions or resources within the system is represented by tokens (dots in p3, p5 and p8). Often the diagrammatic elements will be further identified and labelled with appropriate text, in addition to the coded numbering.

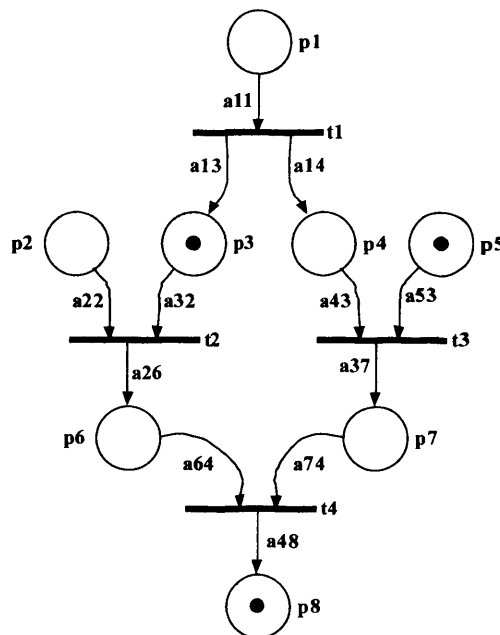


Figure 5.1: Petri-net graph representation.

The execution of a modelled system is controlled by the number of tokens and their distribution in the Petri-net. Petri-nets execute by firing transitions. Transitions fire by removing tokens from input places and adding tokens to output places [5.1]. In the graph representation of Figure 5.1, the event defined by t3 would be enabled if conditions p4 and p5 became true. Therefore, such event would require t1 to fire first,

in order to provide p4 with the required token. The occurrence of an event determines a new state within the system (state mapping). A Petri-net execution carries on as long as at least one transition is enabled to fire. Otherwise, it will be halted.

The above features provide Petri-nets with the capability to represent systems as a sequence of discrete events. Synchronisation is an intrinsic characteristic of Petri-nets, since an event requires the existence of conditions that enable it. Parallelism or concurrency is also a natural feature of the method, enabling the representation of such an important characteristic, normally found in real-life applications [5.2].

Another important concept is Petri-net hierarchy, via the use of sub-nets. In considering important features of the technique, such as synchronisation and concurrency, a system can be modelled employing sub-nets that represent specific parts of the system, easing modelling and analysis practices [5.1]. By employing such notation, a set of elements of the Petri-net might be abstracted to a single element to simplify the main net representation. A complex system based on several hierarchical levels can be represented by nested sub-nets representing each level. Figure 5.2 illustrates the sub-net concept, showing p4, p5, t3 and p7 in Figure 5.1 being replaced by a sub-net block.

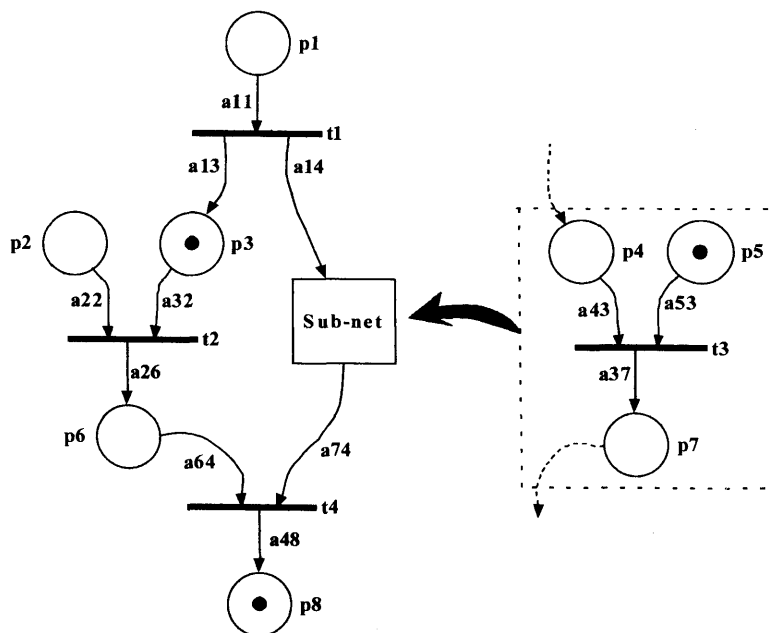


Figure 5.2: Sub-net concept illustration.

5.3 - Petri-net Definitions

Petri-net theory formally defines places and transitions as two disjoint sets of elements of a system. Assuming P as the set of places and T the respective set of transitions, the following relation is established:

$$P \cap T = \emptyset \quad (\text{eq. 5.1})$$

where,

$P(p_1, p_2, \dots, p_n)$ is a finite set of places, $n \geq 0$

and

$T(t_1, t_2, \dots, t_k)$ is a finite set of transitions, $k \geq 0$

The relationship between places and transitions is represented by arcs, defined in terms of input (I) and output (O) functions. Peterson [5.1] described $I(t_j)$ as a mapping of the input places of a transition t_j , while $O(t_j)$ maps the output places of a transition t_j . The relationship between places and transitions, in terms of input and output functions, can be described as follows:

p_i is an input place of t_j , if $p_i \in I(t_j)$;
 p_i is an output place of t_j , if $p_i \in O(t_j)$.

As an extension of such relations, the following is assumed:

t_j is an input transition of p_i , if $t_j \in I(p_i)$;
 t_j is an output transition of p_i , if $t_j \in O(p_i)$.

The original Petri-net formulation suggested some modelling restrictions. It was assumed that a place could not be related more than once to a specific transition related function. Such restriction in “multiplicity” characterises ordinary transitions. This property is defined as follows:

$$\#(p_i, I(t_j)) \leq 1 \quad (\text{eq. 5.2})$$

$$\#(p_i, O(t_j)) \leq 1 \quad (\text{eq. 5.3})$$

where,

$$p_i \in P \text{ and } t_j \in T$$

Another restriction imposed in the original proposition assumes that input and output places of a transition had to be disjointed sets. Such property was named self-loop-free and meant that a place could not be an input and output of the same transition.

$$I(t_j) \cap O(t_j) = 0 \quad (\text{eq. 5.4})$$

By considering Equations 5.2, 5.3 and 5.4, it results that:

$$\#(p_i, I(t_j)) \cdot \#(p_i, O(t_j)) = 0 \quad (\text{eq. 5.5})$$

In the early definition of Petri-net theory, such restrictions were seen as a simplification, without use in the modelling practice. Later, the application of Petri-nets for modelling real-life and complex systems required extended classes of Petri-nets, which included the Petri-net restrictions as sub-classes [5.1].

Some modern applications require multiplicity and therefore do not comply with ordinary transitions. Places with multiple occurrences would violate the set theory (a single occurrence of each element within the set). Peterson [5.1] introduced the concept of bags, in analogy to sets, to overcome such limitation. Bags would allow the multi-occurrence of an element. Following this definition, a set would always be considered a bag, but a bag would not necessarily become a set.

The use of multiplicity resulted in places allowed having multiple occurrences as inputs or outputs of a specific transition. Restrictions imposed by Equations 5.2 and 5.3 were not considered within such extension and new relations must be proposed:

$\#(p_i, I(t_j))$ is the number of occurrences of p_i in the input bag of t_j .

$\#(p_i, O(t_j))$ is the number of occurrences of p_i in the output bag of t_j .

In addition, considering the dual relationship between places and transitions, the following is true:

$$\#(t_j, I(p_i)) = \#(p_i, O(t_j)) \quad (\text{eq. 5.6})$$

$$\#(t_j, O(p_i)) = \#(p_i, I(t_j)) \quad (\text{eq. 5.7})$$

Having established such relations, a Petri-net structure can be defined [5.1, 5.2, 5.3]:

$$C(P, T, I, O)$$

The dynamic of the net structure is defined by the distribution of the tokens in the places, when considering the Petri-net execution. Since the Petri-net represents a system in terms of a sequence of events and states, the firing of transitions modifies such states by adding (generating) and removing (destroying) tokens. The collection of tokens that characterises the status of the system states is called as the Petri-net mapping and is defined by the marking vector μ . The marking vector and the set of places are closely related. The μ of a Petri-net $C(P, T, I, O)$ is defined as a function of P and is normally expressed as a vector:

$$\mu = (\mu_1, \mu_2, \dots, \mu_n) \quad (\text{eq.5.8})$$

where

$$n = |P| \quad (\text{eq.5.9})$$

is the number of places of the Petri-net and each vector's component is defined as

$$\mu_i = N(p_i) \quad (i = 1, 2, \dots, n) \quad (\text{eq.5.10})$$

where N represents the number of tokens of a place p_i . The marking M of a Petri-net C can therefore be described as $M(C, \mu)$, or:

$$M(P, T, I, O, \mu)$$

The initial marking vector is described as μ^0 and represents the very first marking map within the system. Petri-net languages defined some criteria to establish μ^0 [5.1]:

- 1) an initial start place marked with 1 token and 0 tokens elsewhere;
- 2) an arbitrary marking μ specified as the initial mapping;
- 3) a set of initial markings.

Despite such reference, there is not a rule that makes it obligatory such an initial map and transitions are fired as long as there is a marking vector enabling them. If δ is defined as the next state (or marking) function, the following equation describes a new marking in terms of a transition t_j and the marking that enables it fire:

$$\mu^k = \delta(\mu^{k-1}, t_j) \quad (\text{eq. 5.11})$$

As a result of such a relationship, if there is an initial marking vector μ^0 that enables a transition t_j , a new marking μ^1 will be produced. Therefore, the execution of a Petri-net can be defined in terms of a sequence of firing transitions ($t_{j0}, t_{j1}, t_{j2}, \dots$) and a sequence of marking vectors ($\mu^0, \mu^1, \mu^2, \dots$). Given an initial μ^0 and the sequence of transitions that represent the Petri-net execution, it is possible to determine the sequence of marking vectors. Similarly, in having the marking sequence, it is possible to establish the sequence of transitions of the Petri-net execution. The capability of reaching a marking μ^k from a previous marking μ^f ($f < k$), by means of a sequence of firing transitions, is defined as “reachability” and represents an useful property for the analysis of Petri-nets models [5.1, 5.2, 5.3].

The marking vector can be further described in terms of input and output functions. Assuming $I(t_j)$ as the input function of t_j and $O(t_j)$ the respective output function of t_j , the following relation can be drawn:

$$\mu^k = \mu^{k-1} + O(t_j) - I(t_j) \quad (\text{eq. 5.12})$$

Considering that $O(t_j)$ and $I(t_j)$ represent the relationship between places and transitions, these functions can be described as vectors of such a relationship. The elements of such vectors are the weights (multiplicity) of the arcs linking places and transitions. Therefore, the entire space vector of these functions can be represented in terms of matrices, as follows:

$$D^-[j, i] = \#(p_i, I(t_j)) \quad (\text{eq. 5.13})$$

$$D^+[j, i] = \#(p_i, O(t_j)) \quad (\text{eq. 5.14})$$

In assuming $e[t_1, t_2, \dots, t_j, \dots, t_k]$ as the vector representing the Petri-net transitions, so that $e[j]$ is a unit vector where the j^{th} element is 1 and 0 everywhere else, then:

$$I(t_j) = e[j] \cdot D^- \quad (\text{eq. 5.15})$$

$$O(t_j) = e[j] \cdot D^+ \quad (\text{eq. 5.16})$$

Equation 5.12 can thus assume the following form:

$$\mu^k = \mu^{k-1} + e[j] \cdot D^+ - e[j] \cdot D^- \quad (\text{eq. 5.17})$$

Peterson [5.1] defined the relation $D^+ - D^-$ as the changing matrix D of the Petri-net. Therefore,

$$\mu^k = \mu^{k-1} + e[j] \cdot D \quad (\text{eq. 5.18})$$

Assuming that there is a μ that enables a transition t_j , the resulting sequence of transitions firing can be defined as:

$$\sigma = (t_{j_1}, t_{j_2}, t_{j_3}, \dots, t_{j_k}) \quad (\text{eq. 5.19})$$

The marking vector μ^σ resulting of such sequence can be expressed as:

$$\mu^\sigma = \mu + (e[j_1] + e[j_2] + \dots + e[j_k]) \cdot D \quad (\text{eq. 5.20})$$

Defining $f(\sigma)$ as the firing vector of the transitions firing sequence, so that:

$$f(\sigma) = e[j_1] + e[j_2] + \dots + e[j_k] \quad (\text{eq. 5.21})$$

then

$$\mu^\sigma = \mu + f(\sigma) \cdot D \quad (\text{eq. 5.22})$$

And finally, in considering the general expression introduced by Equation 5.11, a general relation is obtained:

$$\delta(\mu, t_j) = \mu + f(\sigma) \cdot D \quad (\text{eq. 5.23})$$

This relationship can be used to determine the firing sequence vector, since the marking vector is known. Although the main strength of Petri-nets is its graphical representation, this sort of mathematical formalism becomes important to support analysis methods to validate a model. Nevertheless, Peterson [5.1] considered that such methods may not be enough to ensure the solution proposed by a model. As examples of these limitations, the matrix D is not capable of differentiating a self-loop (a place acts as input and output of the same transition) from a void link (place is not linked to the transition neither as input, nor as output). This is due to the fact that the same positions in D^+ and D^- will be cancelled in D ($D = D^+ - D^-$). Also, when Equation 5.22 is used to determine the firing sequence vector required to change a Petri-net from an initial marking to another reachable one, it may identify how many times a transition has been fired, without necessarily indicating the sequence in which it happened (i.e. a specific transition may be fired more than once in the same sequence). A level of difficulty is introduced when the number of elements and sequences, which might result from real applications modelling, are considered. In this context however, benefits may result from the modularity of a Petri-net, which enables system designers to split the problem into smaller units, thus applying the analysis methods individually to each one [5.1, 5.2].

5.4 – Petri-net Properties

There is a set of Petri-net properties that are important when modelling and analysing systems. Among these, an important feature of Petri-nets is their capability to easily implement or model concurrency. This is a natural characteristic of many real applications. Peterson [5.1] named the graph elements supportive of concurrency as “fork” and “join”, which are represented in Figure 5.3. In using a fork (Figure 5.3(a)), parallelism is achieved by providing each output place with a token.

The join represented in Figure 5.3(b) provides the means required to the synchronisation of parallel executions, thus representing another feature of Petri-nets. The operation of join is similar to a logic AND, requiring both conditions (p_i and p_j) to be “true”, in order to enable the transition to be fired.

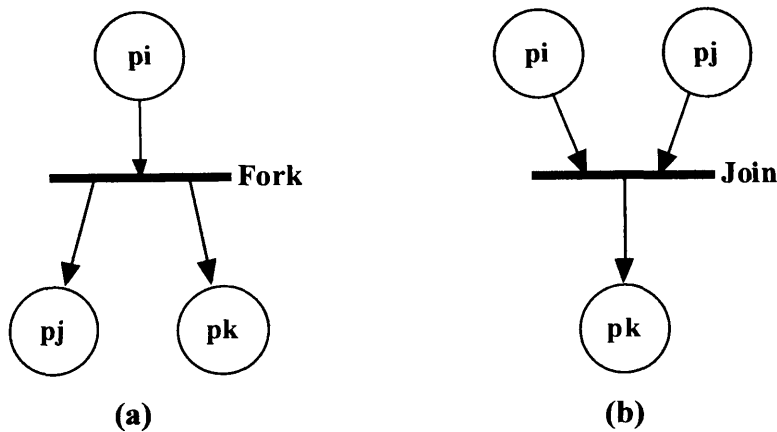


Figure 5.3: Graph elements to support parallelism, fork (a) and join (b) [5.1].

From another perspective, Petri-nets are asynchronous event based, since events are controlled by conditions that are time independent [5.1, 5.2]. Rather, Petri-net execution dependence is represented by the sequence of the modelled events. Theoretically, the firing of a transition is an instantaneous event. It is also assumed that there are no simultaneous events. This resulted in the definition of “conflict”, as the existence of two or more transitions enabled at the same time, having a shared resource (token). Peterson [5.1] considered that conflict resolution in system simulation is “a matter of philosophy”. He suggested a non-deterministic (random) event or an external agent that provides some sort of weighting to support the decision making method.

Mutual exclusion is a feature that enables the execution of Petri-net components or modules in a controlled manner. DiCesare et al [5.2] referred to it as a “simple, appealing and powerful synchronisation mechanism”, suggesting its use as a semaphore to control the sequence in which Petri-net modules/components would execute. There might be cases where Petri-nets modules, although independent of each other in terms of graphical representation (parallelism), should not execute simultaneously. Peterson [5.1] provided an example, illustrated in Figure 5.4, where two processes are required to access a process critical section, but not simultaneously. In the example, “m” represents a permission that avoids both processes “a” and “b” entering the section at the same time. In firing “t1”, “t2” will be disabled until the critical section is finished, then returning a token to place “m”. Equally, if “t2” is the first to be fired, then “t1” will be disabled.

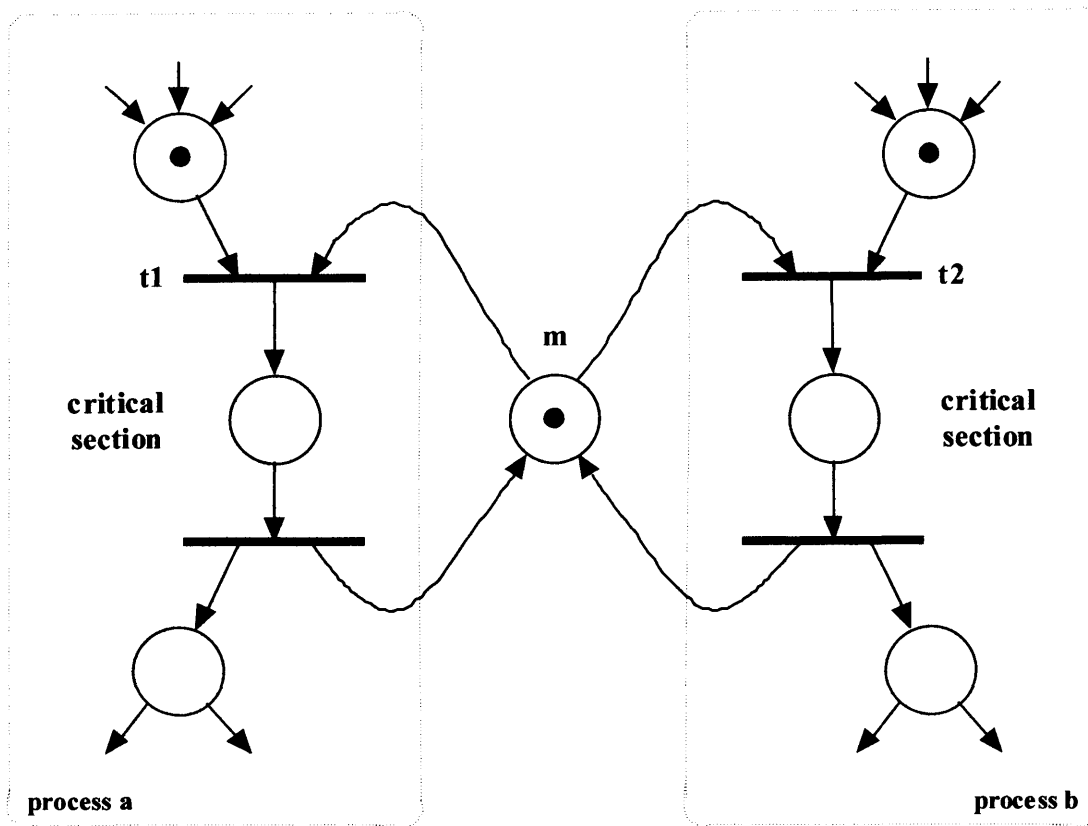


Figure 5.4: Mutual exclusion example [5.1].

Reachability is a property that plays an important role in Petri-net analysis. It represents the capability of a Petri-net to produce a specific marking μ from an initial marking μ^0 and an existing firing sequence. If such marking μ is possible, then it is said to be reachable [5.1, 5.2]. In many Petri-net applications, reachability is a requirement, since a model usually aims to achieve a specific state under certain conditions.

Reversibility is described as a “good” property of a Petri-net model and represents the capability of the system to return to its initial marking from any reachable state [5.2, 5.3]. Other “good” properties are liveness and boundness. The former indicates the absence of deadlock conditions in the Petri-net model, which means that for any marking, there is at least one transition that can be fired. The latter was said to mean the fitness of the state space [5.2, 5.3]. In other words, a place is bound if the number of tokens it can receive is finite. These qualitative properties thus represent good model behaviour characteristics [5.2].

In specific applications, safeness may be an important characteristic. Peterson [5.1] considered that a place is safe if the number of tokens never exceeds one. He cited hardware systems, since the only permitted states of a safe place are 0 or 1. In the original definition of Petri-nets, all places were safe [5.1].

Similarly, conservation may also be a requirement for some applications. If a Petri-net models system resources, such as computer systems I/O devices, the tokens may represent these resources. Therefore, tokens cannot be created or destroyed, in the same way as real resources (that can only be allocated or released) [5.1]. It requires no transition to change the number of tokens in the Petri-net ($|I(t_j)| = |O(t_j)|$).

Many systems representations may result in very complicated models, leading to analysis difficulties. Since the model properties are preserved, reduction techniques can be employed in order to simplify the model. DiCesare [5.2] suggested a “basic reduction kit”, composed of some graphic structures examples that might simplify some blocks within the Petri-net model.

5.5 – Petri-net Extensions

Since the original proposition of Petri-nets, several extensions were made in order to satisfy specific requirements to the modelling of real applications [5.1]. Initial Petri-nets were considered a sequence of non-deterministic events, instantaneous and non-simultaneous. Events with such characteristics were identified as “primitive events”. Also, original Petri-nets allowed only one token in each place [5.1].

Peterson [5.1] considered that most real-life events take time, thus defining non-primitive events. The representation of non-primitive events does not follow a specific rule. He suggested as a representation example a place between two primitive events (start and end transitions), as seen in Figure 5.5(a). He also presented Petri’s original suggestion, that a non-primitive event should be represented as a box (Figure 5.5(b)). However, he considered that the box symbolism should be employed to represent sub-nets.

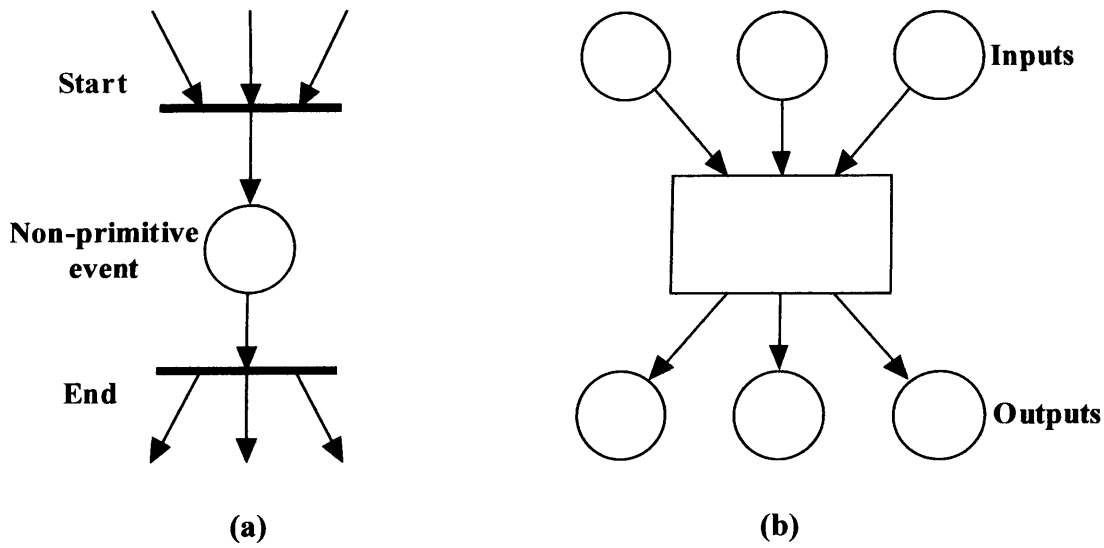


Figure 5.5: Examples of the representation of non-primitive events as two primitive events (a) and a box (b) [5.1].

It was also recognised that Petri-nets were limited in testing “zero” conditions. One of the most accepted extensions is the use of inhibitor arcs, rather than arrowed ones, to represent a zero test (Figure 5.6). Such representation was the most straightforward for modelling purposes [5.1]. It made modelling easier by enabling the representation of a wider range of logic operations (e.g. NOT, eXclusiveOR) and system switches states.

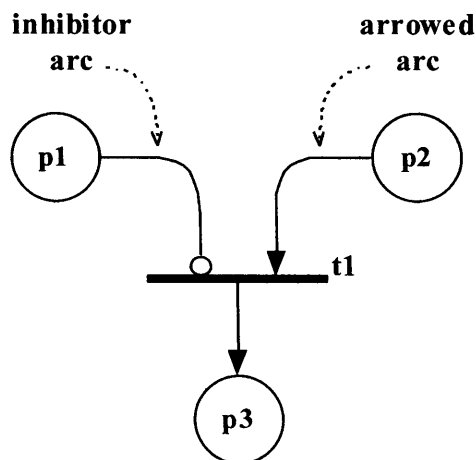


Figure 5.6: Example of an inhibitor arc for zero testing.

The use of a priority scheme was suggested in order to help in the decision of which transition should be fired first, in a conflict situation, providing transitions with priority indices [5.1]. The concept of time Petri-net was also proposed for the same purpose, with each transition t_j having 2 time constants, τ_1 and τ_2 . Therefore, t_j would only be fired after enabled for at least τ_1 and before τ_2 , providing a scheme similar to priority [5.1].

DiCesare et al [5.2] described timed firing, considering another timing approach. In such case, a firing time is associated with a transition. They described the firing of a timed transition as a 3 phase event, as follows:

- 1) after a transition is enabled, firing is initiated by removing tokens from input places;
- 2) the firing process remains for the firing time;
- 3) after the firing time has elapsed, output places are updated.

Place timed Petri-nets were also approached by DiCesare et al [5.2]. In such a case, a token received by a place could only be assumed valid after an elapsed time associated with the place, thus representing another option for conflict resolution.

The use of Petri-nets for the modelling of manufacturing systems provided another extension, named coloured Petri-nets [5.8]. In such systems, besides representing the existence of resource conditions, tokens may also be required to represent a product or a work piece that must be identified. Thus, coloured Petri-nets utilise two different places: elementary and object. The first deals with normal tokens (basic Petri-nets), used to control resources and provide enabling conditions, while the second one holds coloured tokens, which are identified and represent objects within the Petri-net.

Certainly, there are more extensions to the original proposition of Petri-nets, such as the one considered in [5.6] for process and condition monitoring, developed in order to implement specific modelling requirements. Such continuous updating of the method is an indication of the interest that many areas have in the use of Petri-nets as a modelling tool.

5.6 – Summary

Petri-nets represent a modelling method that has mathematical formalism, however its main strength is in providing graphical representations. One of the characteristics of the method is the capability of modelling systems as sequences of discrete events and states. To enable the method to better represent real-life systems, extensions to the original concept were proposed.

The following chapter will outline the use of the Petri-net concept and some of its extensions, for the implementation of a monitoring modelling tool, showing how it was applied in this research work.

REFERENCES

- 5.1 **Peterson, J.L.** Petri Net Theory and the Modeling of Systems. Englewood Clift, USA: Prentice-Hall Inc., 1981.
- 5.2 **DiCesare, F., Harhalakis, G., Proth, J.M., Silva, M. and Vernadat, F.B.** Practice of Petri Nets in Manufacturing. London, UK: Chapman & Hall, 1993.
- 5.3 **Girault, C. and Valk, R.** Petri Nets for Systems Engineering. Berlin, Germany: Springer-Verlag, 2003.
- 5.4 **Peng, S. and Zhou, M.** Sensor-based Stage Petri Net Modelling of PLC Logic Programs for Discrete-event Control Design. *International Journal of Production Research*, 2003, 41(3), 629-644.
- 5.5 **Frey, G. and Minas, M.** Internet-based Development of Logic Controllers Using signal Interpreted Petri Nets and IEC 61131. Friedrich-Alexander-Universität Institut für Informatik Web Site. Available from: <http://www2.informatik.uni-erlangen.de/download/Papers/Sci2001.pdf> [Accessed 02 December 2002].
- 5.6 **Prickett, P.** A Petri-net Based Machine Tool Maintenance Management System. *Industrial Management and Data Systems*, 1997, 97(4), 143-149.
- 5.7 **Yang, S.K. and Liu, T.S.** A Petri-net Approach to Early Failure Detection and Isolation for Preventive Maintenance. *Quality and Reliability Engineering International*, 1998, 14, 319-330.
- 5.8 **Zimmermann, A. and Hommel, G.** Modelling and Evaluation of Manufacturing Systems Using Dedicated Petri Nets. *The International Journal of Advanced Manufacturing Technology*, 1999, 15, 132-138.

CHAPTER 6

PETRI-NET MONITORING MODEL

6.1 – Introduction

As introduced in Chapter 5, Petri-nets have found their main application as a system modelling and simulation tool. Much of the Petri-nets' power is related with the graphical representation the method enables [6.1], stimulating its use for many engineering application [6.2, 6.3, 6.4]. The use of the Petri-net concept for process and condition monitoring was investigated by a research group at Cardiff University [6.5]. Due to its potential as a monitoring method, further investigations were carried out [6.6, 6.7], resulting in a monitoring system using a graphical software tool, running in a PC/Windows® environment [6.8].

Peng and Zhou [6.9] described a modern manufacturing process as a sequence of discrete events, where events trigger each other and thus cause system operation to take place. Petri-nets are suitable for the representation of discrete systems [6.1, 6.10], where process events can be characterised as transitions and the conditions (process states) that enable such events are represented by the Petri-net places [6.5]. Such an approach represents an alternative way to describe the operation of a process characterised by sequential events. It has resulted in the formulation of a method for non-sensor based monitoring systems [6.11], where “non-sensor” was intended to indicate the use of the already existing process signals, i.e. without deploying additional specific sensors or transducers.

Although a monitoring system based on a microcontroller implementation may represent a lower-cost alternative than previous, PC-based, methods in terms of hardware investment, such benefits would be considerably reduced if new software and hardware designs were required for each application. Thus, in this research, the Petri-net approach was considered as a tool to be used to model and describe the monitoring task. As a result, both hardware and software could be independent from

the monitoring task, enabling system reusability and consequently obtaining the low-cost benefit. In general terms, the approach employed follows the one described by Prickett and Grosvenor [6.5], however considering the use of microcontrollers rather than PCs.

6.2 – The Modelling Approach

The Petri-net theory, outlined in Chapter 5 defines places as conditions that enable transitions [6.1, 6.10]. The use of this method for monitoring purposes requires that places should be understood as being representative of the process states. In order to identify a new process event (corresponding to a Petri-net transition), the process signals have to be considered in conjunction with the actual process states [6.5]. Such an approach is illustrated in Figure 6.1.

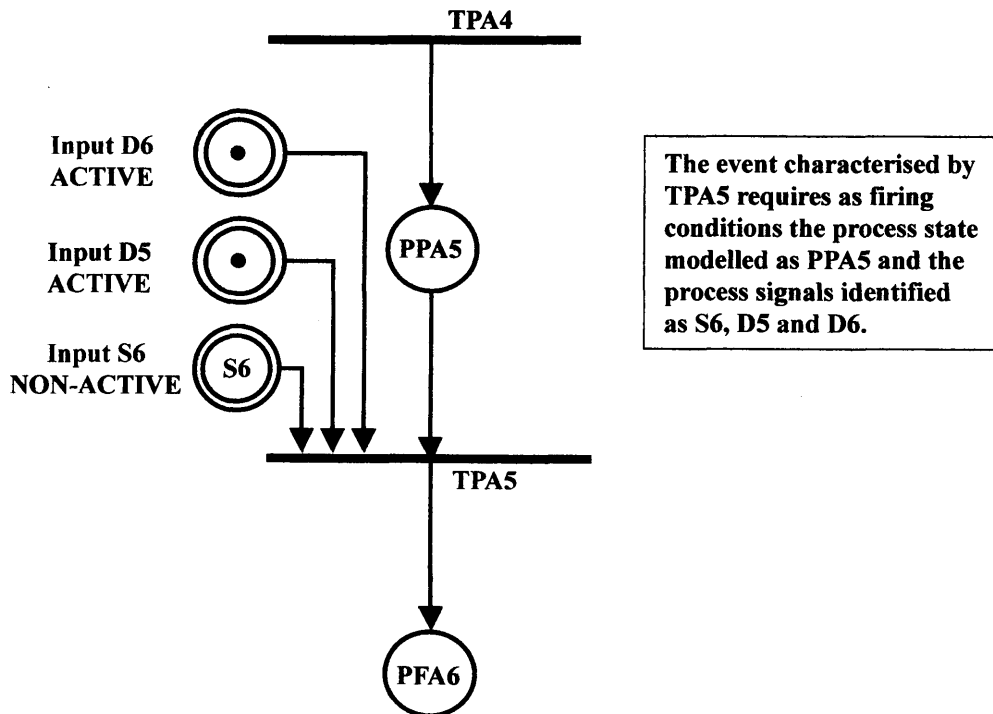


Figure 6.1: Petri-net approach for monitoring purposes (after [6.11]).

In considering this approach, it can be assumed that a Petri-net for monitoring purposes requires two sets of conditions that have to be validated to enable a

transition that characterises a process event. A similar approach was presented by Peng and Zhou [6.9], when describing the use of Petri-nets as a Programmable Logic Controller (PLC) programming method. They defined an event trigger as a combination of places and input signal status. In following such an approach in this research, it is necessary to define two separate sets of input conditions required to enable a transition. If it is assumed that z is a finite number of input signals of a system, then there is a set of input signals conditions required to enable a transition t_j represented by $X(x_{j1}, x_{j2}, x_{j3}, \dots, x_{jh})$, $0 \leq h \leq z$. If the set of places of a Petri-net is defined by $P(p_1, p_2, p_3, \dots, p_m)$, then the firing event of a transition t_j is controlled by two functions:

$$S(t_j) = f(x_{j1}, x_{j2}, \dots, x_{jh}), \quad 0 \leq h \leq z \quad (\text{eq. 6.1})$$

$$Q(t_j) = f\{I(t_j), \mu\} \quad (\text{eq. 6.2})$$

where

$$I(t_j) = f(p_1, p_2, \dots, p_n), \quad 0 \leq n \leq m \quad (\text{eq. 6.3})$$

was defined in Chapter 5 as the input function of the transition t_j and μ represents the Petri-net actual marking vector.

The function $S(t_j)$ represents the logical relation of all the input signal conditions associated to a specific transition and thus can only assume 0 (false) or 1(true).

$$S(t_j) = x_{j1} \cdot x_{j2} \cdot x_{j3} \cdot \dots \cdot x_{jh} \quad (\text{eq. 6.4})$$

Each x_{ji} represents a condition of a specific input signal (i) in the domain of a specific transition (j). Therefore, x_{ji} can also be logically represented as 0 (false) and 1 (true). Considering a transition t_j , which requires an input signal condition x_{ji} to be satisfied, then x_{ji} can be described as:

$$x_{ji} = f(y_i, c_{ji}) \quad (\text{eq. 6.5})$$

where y_i is a specific input signal of the entire set $Y(y_1, y_2, \dots, y_i, \dots, y_z)$ of the process input signals and c_{ji} is the desired status of this signal in the domain of transition t_j . Thus,

$$x_{ji} = 0 \text{ if } y_i \neq c_{ji} \quad (\text{eq. 6.6})$$

$$x_{ji} = 1 \text{ if } y_i = c_{ji} \quad (\text{eq. 6.7})$$

and

$$S(t_j) = 1 \text{ if all } x_{ji} = 1 \quad (\text{eq. 6.8})$$

$$S(t_j) = 0 \text{ if any } x_{ji} = 0 \quad (\text{eq. 6.9})$$

The second condition required to enable a transition t_j is represented by $Q(t_j)$. There are only two options: enabled or not enabled. Hence, the function can also be assumed as representing two logical states: 0 for false and 1 for true. The following relations can be obtained, considering an existing marking μ :

$$Q(t_j) = 1 \quad \text{if } \mu \text{ satisfies } I(t_j) \text{ so that } t_j \text{ might be enabled} \quad (\text{eq. 6.10})$$

$$Q(t_j) = 0 \quad \text{if } \mu \text{ does not satisfy } I(t_j) \text{ in order to enable } t_j \quad (\text{eq. 6.11})$$

Either $S(t_j)$ and $Q(t_j)$ can prevent t_j from being fired, however both are required to be true to enable t_j .

$$S(t_j) \cdot Q(t_j) = 0, \text{ then } t_j \text{ is not enabled} \quad (\text{eq. 6.12})$$

$$S(t_j) \cdot Q(t_j) = 1, \text{ then } t_j \text{ is enabled} \quad (\text{eq. 6.13})$$

From another perspective, considering Equation 5.12 in Chapter 5, it might be assumed that the marking μ^k is possible if t_j is enabled, so that

$$\mu^k = \mu^{k-1} + O(t_j) - I(t_j), \text{ if } S(t_j) = 1 \quad (\text{eq. 6.14})$$

$$\mu^k = \mu^{k-1}, \text{ if } S(t_j) = 0 \quad (\text{eq. 6.15})$$

It becomes clear that the marking of the Petri-net will depend only of its representation in terms of places and transitions, the relationship between them and the initial marking μ^0 . It is also a requirement, since the model must represent the process uniquely, without dependency on any signal or specific state entered by the

process. However, transitions besides being enabled by the process actual states (Petri-net marking), require the process signals to fulfil conditions within the transition domain, in order to be enabled to fire. It is not enough for only the condition provided by the process signals to be true. Neither is it enough for the process to be in the required states. A new marking within the Petri-net is only possible if all these conditions are present together. Therefore, process monitoring in terms of states and events is made possible by keeping records of the process marking and taking into account the process signals and their required status within the domain of each specific transition of the Petri-net that models the process.

6.3 – The Model Structures

In terms of behaviour, the structures required to model a process using the approach proposed by this research could be classified as dynamic and static. Dynamic are those that may change their status during the execution of the Petri-net. Such category includes places and sub-nets. Petri-nets transitions were considered as static structures. This is because within this approach, the transitions define the process sequence of events and therefore do not change under any circumstance. Such classification becomes important for the implementation of the Petri-net approach based on a microcontroller. Static elements can use the program memory, avoiding the use of the data memory, a valuable and limited resource in a microcontroller.

6.3.1 – Places

The Petri-net places require an identification to distinguish them from each other. Here a continuous numbering method was employed (1, 2, 3, ..., n). Each place must provide a container (counter) that holds the number of tokens belonging to the place. The number of tokens will then vary (increase or decrease) during the Petri-net execution. The maximum number is bounded by the container data type size. In considering the use of an 8 bits microcontroller, the “byte” was selected as the place container data type, since instructions are optimised for the processor’s natural data format. Thus, the number of tokens of a place was bounded to 255.

6.3.2 – Sub-nets

In the Petri-net theory presented in Chapter 5, sub-nets were introduced as a concept to simplify the analysis of a complex network by replacing a set of elements (transitions, places, arcs), which could be isolated and then analysed. In this research, the term sub-net assumed a slightly different meaning, although it was kept in the same context. It is considered that a sub-net is any Petri-net affecting or being affected by the one executing. This opens-up the possibility of communicating (synchronise) individual Petri-nets, running on different microcontroller's environments. By using this feature a larger process might be monitored by deploying several structured Petri-net systems that are synchronised with each other. Sub-nets must be assumed as dynamic elements within the system, since their status may change during the Petri-net execution. For simplicity, sub-nets might be considered as a place that is bounded to 1. Their identification follows a principle similar to places, being sequentially numbered (1, 2, 3, ..., n).

6.3.3 – Transitions

The characterisation of the process events as transitions enables the establishment of a Petri-net skeleton. A collection of static structures can thus be used to describe the process Petri-net in terms of such events. These structures can be made to be auto-descriptive to allow a totally independent execution. In considering the representation of these structures by the Petri-net transitions, there is a need to provide in their description the identification, pre-conditions (inputs) and post-conditions (outputs). Within this approach, in order to capacitate the modelling method, some different transition structures were defined and will be described in the following sections. Particular implementation details are discussed later, in Section 6.4.

6.3.3.1 – Ordinary Transition

The first such structure was named an “ordinary transition” due to the fact that it represents the basic structure required in the modelling process. Figure 6.2 presents this structure diagram. The first field in the structure is the transition identification, which is numerically represented (1, 2, 3, ..., n). The second field provides specific

information about the transition. Flags within this field tell the system what sort of transition it is. There is also a flag to indicate whether the firing of the transition should become a public event (i.e. is associated with the transmission of a message), or if it is only of interest within this Petri-net domain, to update the process marking. Figure 6.3 shows the details of the status field flags. These two initial fields (Transition ID and Status) are common to all transition structures, as it will be seen in the following sections.

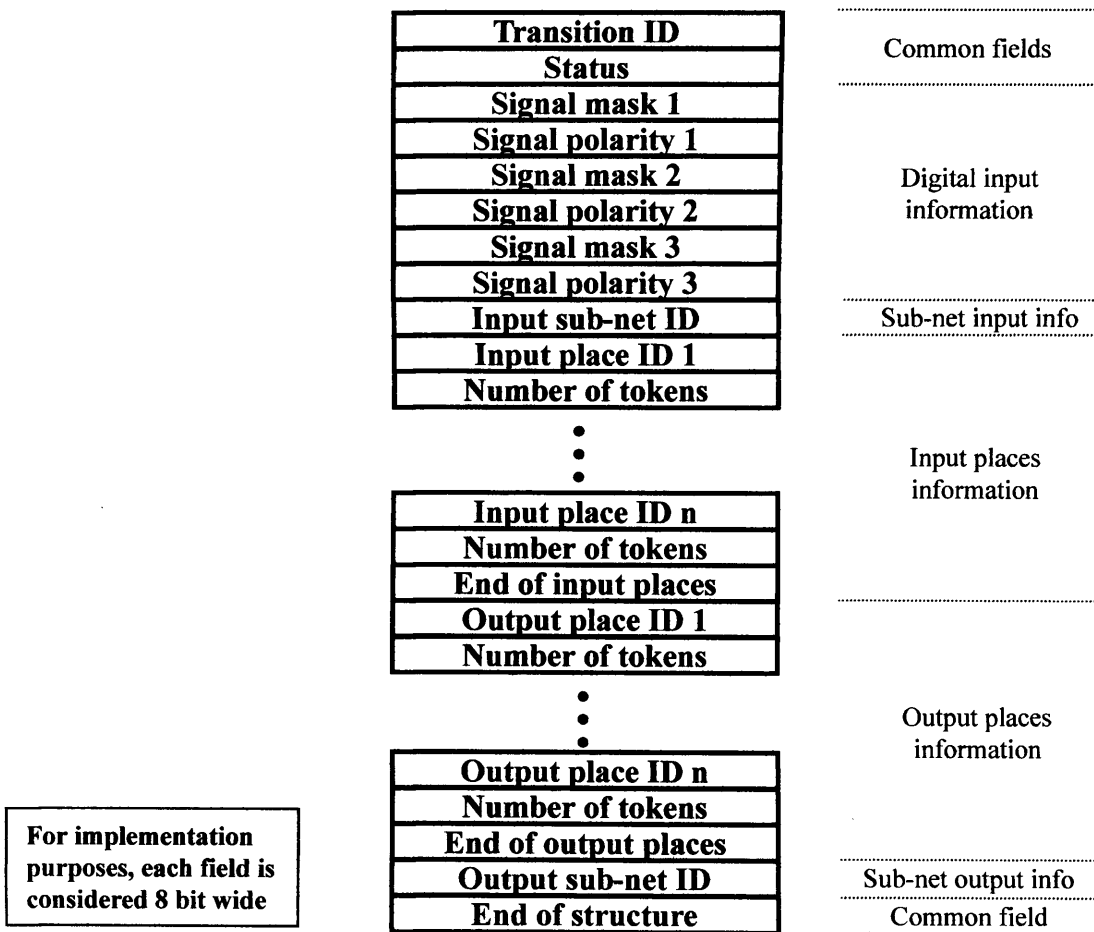


Figure 6.2: Ordinary transition data structure.

Ordinary transitions can only handle digital input signals. The following 6 (3 x 2) alternate fields in the structure provide information regarding these signals and their required status to enable the specific transition. The signal mask reserves 1 bit position for each signal (0 to 7). A bit level 1 at a specific position indicates that this signal must be taken into consideration at the given transition. The signal polarity

field indicates the required signal status (0 or 1), i.e. is the input to the transition required to be logically true or false, and is aligned with the respective bit position in the signal mask field. For simplified implementation, the polarity assigned in the field must be inverted, meaning that it is expressed as 0 when the signal is required to be logically true. Figure 6.4 presents the relationship between the pair of digital input fields. Figure 6.5 confirms the logic required to identify a true condition. The existence of 3 groups of such field pairs indicates the ability of a transition to handle up to 24 digital inputs. For implementation purposes, those signals that are of no relevance to the particular transition the corresponding bit position polarity was assigned as 1. Thus provided a simpler way to test an entire set of 8 inputs simultaneously, rather than individually.

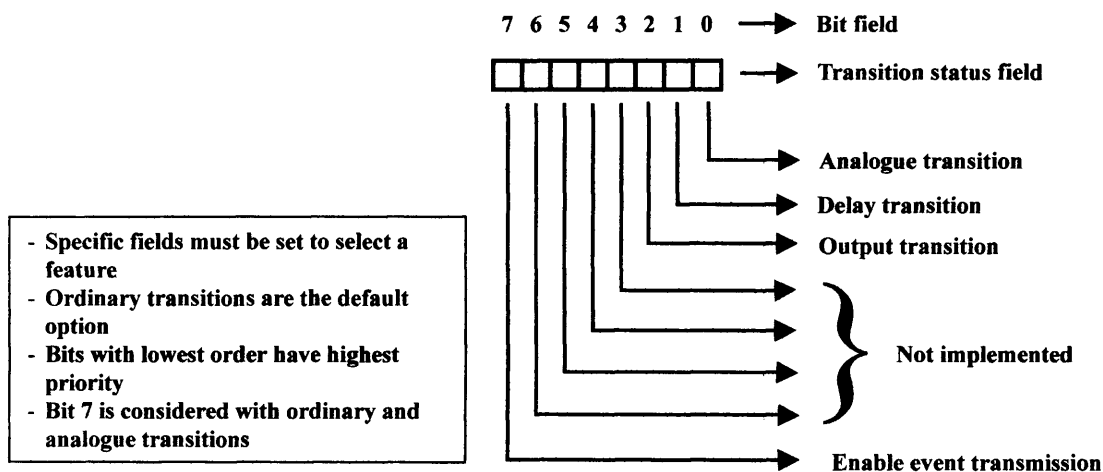


Figure 6.3: Transition status field details.

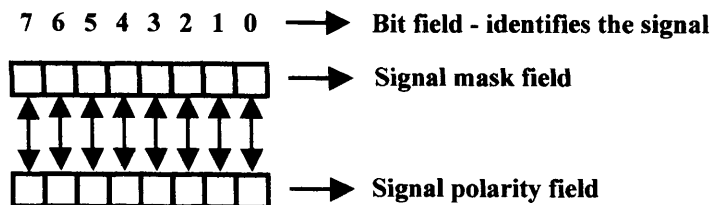
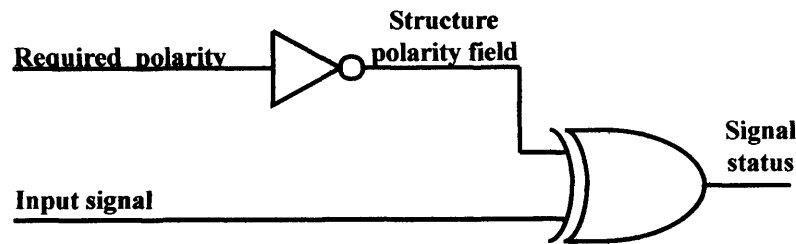


Figure 6.4: Ordinary transition signal mask and polarity fields' relationship.



Required polarity	Structure polarity field	Input signal	Signal status	Meaning
1	0	0	0	FALSE
1	0	1	1	TRUE
0	1	0	1	TRUE
0	1	1	0	FALSE

Figure 6.5: Digital input signal validation method.

Continuing with the structure definition of Figure 6.2, the next field is the input sub-net (Input sub-net ID). By definition, if this field is assigned as 0 (zero), no sub-net input is linked to the transition; otherwise the appropriate sub-net ID code is identified.

The collection of input places is the next information in the ordinary transition structure. The input place ID identifies the Petri-net place that is assigned as an input condition of the transition. The number of tokens indicates the “multiplicity” of the arc linking the input place to the transition (a condition becomes true if the assigned number of tokens is found in the place). An ordinary transition may have several input places, each one with its individual multiplicity mark (limited to 255). An “end of input places” field indicates that there are no more input places to be considered in the domain of the specific transition.

Similarly, the next collection of fields within the structure is associated with output places. The output place ID identifies the Petri-net place that should be updated due to the transition firing. The number of tokens represents the multiplicity of the arc linking the transition to the output place, thus indicating the number of tokens the place should receive. The “end of output places” is the mark indicating that there are no more output places linked.

The output sub-net field identifies, if appropriate, the sub-net element requiring notification of the specific transition firing. A value 0 in the output sub-net ID field indicates that there is no such a requirement. The way in which the sub-net ID is made public is a matter of system implementation. However, considering the nature of such feature and its purpose, a message broadcast method was adopted, allowing consumers (other Petri-nets) to decide whether or not to make use of the information. Such a method is becoming common in the implementation of some distributed systems, provided with local processing capabilities [6.12].

The final field of an ordinary transition structure is the end mark “end of structure”, meaning that there are no more fields in this transition. As it will be seen, the same field is mandatory in all the transition structures used in this Petri-net approach.

6.3.3.2 – Analogue Transition

A second modelling structure defined for the Petri-net monitoring approach was the “analogue transition”. This handles non-logical signals, instead of discrete ones (as ordinary transition do). Since a transition represents an event, the analogue transition is required to provide means that enable the identification of such an event, based on analytical information. A method that considers two parameters was proposed, in order to characterise such an event. The first of such parameters would represent a threshold and the second an analysis condition (<, >, =).

Figure 6.6 illustrates the data structure of an analogue transition. The transition ID and status fields follow exactly the same description provided for the similarly named fields in the ordinary transition structure (Section 6.3.3.1). The condition field is the identification of the comparison method requested (<, =, >). Further implementation details are summarised later, in Table 6.1 (Section 6.4).

The source ID (low half of the third field) identifies which specific process analogue source should be measured, in order to determine the firing condition. The analogue transition allows only one analogue source as input condition. The next two fields (Threshold MSB and LSB) of the data structure need to be combined to produce the threshold value required in the comparison test.

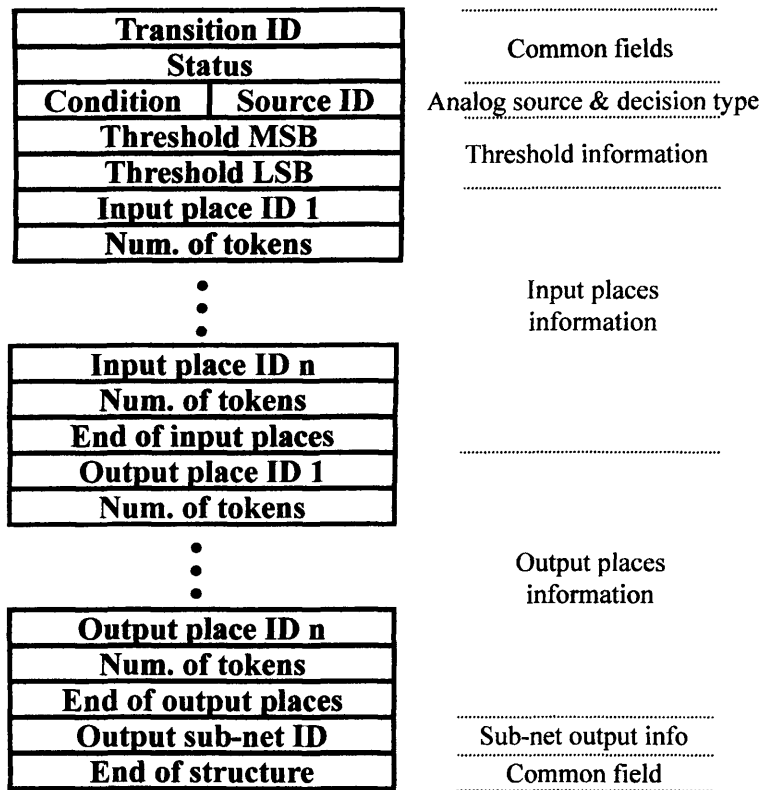


Figure 6.6: Analogue transition data structure.

All the following remaining fields, from “input place ID” to “end of structure”, have exactly the same meaning and representation described for the ordinary transition structure (Section 6.3.3.1, Figure 6.2). Although linkage to an output sub-net field was permitted by the structure, the equivalent input field (Input sub-net ID) was not considered to be necessary.

6.3.3.3 – Delay Transition

Although the original Petri-net theory considered that transitions were instantaneous events [6.10], the use of the method to model real applications showed the necessity to represent events that take time to execute [6.1]. To monitor a process through its signals, there might be occasions where some sort of flexibility would be required. For example, the switching action of an electrical signal can introduce noise that may induce the misinterpretations of the signal’s levels. Therefore, this modelling approach has provided a structure that enables the insertion of a time delay, named a

“delay transition”. The method is based on the Petri-net extension described by DiCesare et al [6.1], previously introduced in section 5.5.

Figure 6.7 therefore defines the delay transition data structure used. The proposed element considers only one input and one output place. The firing is enabled by the existence of a single token in the input place, i.e. it has arcs that do not support multiplicity. The two first fields in the structure are as previously defined (for ordinary and analogue transitions). The same applies to the last field (End of structure). The input place represents the unique condition required to enable the transition to fire. The output place field indicates which of the Petri-net places should receive a single token when the transition is fired. The delay value is specified via a two fields parameter (Parameter MSB and LSB). These together define the time delay in milliseconds (ms). Immediately after being enabled, the token of the input place is removed. After the delay expressed in the parameter field is elapsed the transition is fired and the output place updated.

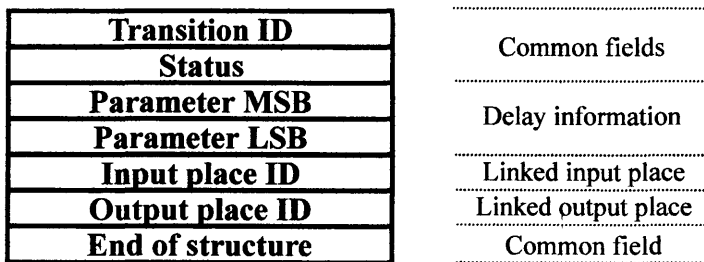


Figure 6.7: Delay transition data structure.

6.3.3.4 – Output Transition

The final structure defined was required to enable the monitoring hardware to issue local (hardware) alarms, following a modelled event. Such an element was named as an “output transition” and its data structure is shown in Figure 6.8. In this simple structure definition the common fields apply as before. In summary, an output transition is enabled by only one input place (Input place ID), which requires a single token. Although resulting in a token being removed from the input place, there is no output place to be updated. In line with formal Petri-net theory, it could be assumed

that an output transition, when fired, sends a token to a sub-net represented by the monitoring hardware/software implementation. The resulting action of such an event is then a matter of system implementation (hardware and software).

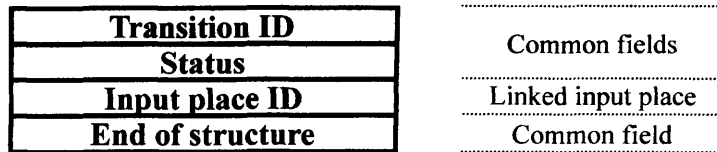


Figure 6.8: Output transition data structure.

6.4 – Implementations Aspects and Representation

In considering the way the modelling elements and the method proposed were presented, the Petri-net monitoring approach is not restricted to a specific implementation. It can be employed in many different developments, and could potentially be based on different processors since following the Petri-net concept. The way in which each Petri-net is actually executed becomes only a question of software development. However, since one of the main objectives of this investigation is the proposition of a low-cost monitoring system, work was focused on the use of microcontrollers.

In order to enable the implementation of such an approach, the elements defined in section 6.3 require a tighter description. Considering the byte as the natural data type for many microcontrollers (including the PIC18C452), the structures earlier presented were assumed 8 bit wide, resulting in the parameterisation shown in Table 6.1.

Place “0” has a special meaning for the system implementation. It represents the initial state in the Petri-net, i.e. the place that should receive the first token after initialisation. It follows Peterson’s [6.10] suggestion of a “start place” with a token and no tokens elsewhere. A second meaning of place “0” is for a Petri-net reset request. A reset condition is identified by an output place “0”, which should result in the system restarting the Petri-net execution (initial start state).

Table 6.1: Transition structures implementation definitions.

Field	Size	Representation	Structure	Description
Transition ID	8 bits	1 to 254	All	<ul style="list-style-type: none"> Petri-net transition identification.
Place ID	8 bits	0 to 254	All	<ul style="list-style-type: none"> Petri-net place identification (input & output).
Sub-net ID	8 bits	0 to 255	Ordinary & analogue	<ul style="list-style-type: none"> Petri-net sub-net identification (input & output).
Number of tokens	8 bits	0 to 255	Ordinary & analogue	<ul style="list-style-type: none"> Arc multiplicity – number of tokens required from an input place or added to an output place.
Status	8 bits	-	All	<ul style="list-style-type: none"> Defines transition structure and actions. Detailed in Figure 6.3.
Signal mask	8 bits	-	Ordinary	<ul style="list-style-type: none"> Selection of the digital signals considered in the transition domain.
Signal polarity	8 bits	-	Ordinary	<ul style="list-style-type: none"> Digital signals level, with reversed polarity - default binary 1.
Condition	4 bits	= : 0000 binary > : 1000 binary < : 0001 binary	Analogue	<ul style="list-style-type: none"> Comparing condition of a non-digital parameter in an analogue data structure.
Source ID	4 bits	1 to 15	Analogue	<ul style="list-style-type: none"> Non-digital input parameter identification, representing the signal input in an analogue transition
Threshold MSB + LSB	16 bits	0 to 65535	Analogue	<ul style="list-style-type: none"> Value to be considered in the comparison process of an analogue transition.
Parameter MSB + LSB	16 bits	0 to 65535	Delay	<ul style="list-style-type: none"> Delay, in milliseconds, to be performed by a delay transition.
End of input places	8 bits	255	Ordinary & analogue	<ul style="list-style-type: none"> Input places delimiter.
End of output places	8 bits	255	Ordinary & analogue	<ul style="list-style-type: none"> Output places delimiter.
End of structure	8 bits	255	All	<ul style="list-style-type: none"> Defines the end of the structure.

The description of a monitoring task within the system is represented by the transition structure's characterisation of the process events. In terms of an implementation, such a set of data structures could be defined as a data table. An element named "end of table" and represented by the numerical "0" identifies the condition that indicates the end of such table. Figure 6.9 provides a block diagram where the use of this element is made clear.

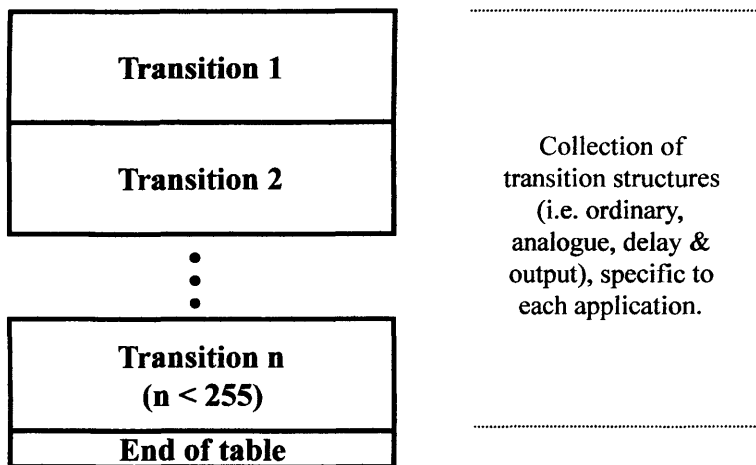


Figure 6.9: Example of use of the “end of table” mark.


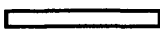







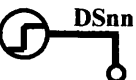



The modelling of the monitoring task is made easier by representing the process' Petri-net as a graph. Although keeping the main graphical characteristics of the original Petri-net concept, a few new elements were required in the developed monitoring approach, in order to represent all of the structures described. Table 6.2 summarises the entire set of graphical elements, their identification and description.

6.5 – Monitoring Records

Another aspect of system implementation is the capability of the monitoring system to report events. The possibility of issuing local alarms, using an output transition has been considered (Section 6.3.3.4). Nevertheless, as considered in Chapter 2, modern monitoring systems are expected to integrate the data they provide, helping in the generation of information to support increases in process management efficiency.

Therefore, such systems should be capable of issuing data records, using data communication standards.

Table 6.2: Petri-net monitoring approach graphic modelling elements.

Representation	Element	Description
 T _{nnn}	Transition	Representation of ordinary and analogue transitions.
 T _{nnn}	Transition	Representation of a delay transition.
 T _{nnn}	Transition	Representation of an output transition.
 P _{nnn}	Place	Representation of a conventional place.
	Place	Output place indicating a restart request.
	Arc	Arc with multiplicity n (number of tokens). No indication means a single token.
	Sub-net	Input sub-net (from another Petri-net).
	Sub-net	Output sub-net (to another Petri-net).
	Digital signal	Digital input with required high (1) level.
	Digital signal	Digital input with required low (0) level.
	Analogue input	Analogue input , with comparison method " lower than ".
	Analogue input	Analogue input , with comparison method " higher than ".
	Analogue input	Analogue input , with comparison method " equal to ".

In considering the Petri-net concept, the firing of a transition would be a natural monitoring record, since it represents a “process event”. Taking into account such a requirement, the “enable event transmission” flag (bit 7 of Figure 6.3) was incorporated in the transition’s structure status field. Nevertheless, there are other events that can be of interest that should be considered and reported [6.13].

Prickett and Grosvenor [6.11] referred to the use of a timeout feature, characterised by a transition failing to fire within a defined period of time. Considering the implementation aspects in a microcontroller environment, it was decided to associate such a feature with places, rather than transitions. Thus, in the approach proposed by this research, “timeout” records will be produced in response to a process state (selected Petri-net place) lasting longer than previously recorded (or defined).

Additionally, two complementary records were defined to monitor the “beginning” and “ending” of process specific states. Such records could provide for example, means to enable a remote visualisation of the process operation (active or inactive) or else, help to measure the process operating time. These are in many cases managerial requirements [6.14].

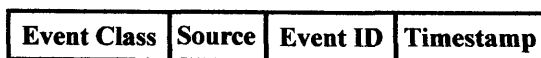
It was also considered that Petri-net places representing process states could be used to control the acquisition of process specific parameters, such as analogue signals. This assumes that specific process information is of interest under certain conditions. For example, the behaviour of a motor’s current while switched on. Therefore, the acquisition of the specific analogue signal would be triggered by the associated process state becoming active (Petri-net place receives first token). At the end of this state (Petri-net place last token removed), a monitoring “special” record would be issued providing the required signal information. Table 6.3 lists the defined monitoring records proposed in this monitoring approach and that were presented in this section.

Figure 6.10 shows the general format of the monitoring messages. The “event class” field represents the identification of the messages described in Table 6.3. The “source” field would indicate the Petri-net that generated the message. The “event ID” field would identify the transition or place that motivated the message. The timestamp

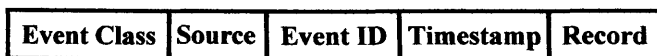
provides a time record linked to the event reported by the message. Figure 6.10(b) shows an alternative extended record field, for use by messages as “timeout” and “special record”, described in Table 6.3.

Table 6.3: Monitoring messages definition.

Monitoring Record	Purpose Description
Process event	<ul style="list-style-type: none"> Message issued in response to a process event (fired transition). The event enable transmission flag of the status field in the transition data structure (Figure 6.3, bit 7) must be set to enable such record. Only ordinary and analogue transitions can issue this sort of message.
Beginning of a process state	<ul style="list-style-type: none"> Enabled places issue such message when receiving the first token, indicating the “beginning” of the associated process state (state became active).
Ending of a process state	<ul style="list-style-type: none"> Enabled places issue such message when becoming empty (last token removed), indicating the “ending” of the associated process state (became inactive).
Process state timeout	<ul style="list-style-type: none"> Messages issued by selected places to indicate that a process state has lasted longer than expected.
Special record	<ul style="list-style-type: none"> Message issued at the end of a selected process state containing a record with a feature extracted from a process analogue signal. The signal is acquired as long as the process state remains active (e.g. a DC motor current mean value, to indicate the motor’s operating condition).



(a)



(b)

Figure 6.10: Monitoring messages general format.

Within this proposed monitoring approach, Petri-net places assume greater relevance in the implementation of special functions for the assessment of processes. Special places require to be defined in the system implementation as a set, associated to the function they will perform (i.e. monitor the status of process specific state by means of “beginning” and “ending” records or triggering the acquisition of analogue signals). This is different to transitions, which are defined individually and thus can provide additional information within the data structure. In considering the application based on a microcontroller, such places could be listed in a data table, using the device program memory. Also considering implementation aspects, places should be grouped into those that allow timeout and those that do not. This could be easily implemented by means of a separating “mark”, dynamically defined accordingly to the application requirements. In the same way that many transitions in the Petri-net might represent a minor event that do not require a message, a timeout feature associated to some places in the process monitoring model might generate a considerable number of records with little significance. More details must be considered at the implementation level.

6.6 – Fault Diagnostics Approach

The use of a method that follows the process events and keeps track of process states can represent a tool in the identification of operational faults. Such an approach was considered by Prickett and Grosvenor [6.11], suggesting that a timeout condition in the system might represent the existence or development of a faulty state.

Hu et al [6.15] proposed operational fault diagnostics based on tree analysis, assuming that by knowing the process states and the actual controller’s (PLC) signals, a fault could be isolated.

The Petri-net monitoring approach provides the required characteristics to enable such an implementation. The existence of process states timeout, since understood as a symptom, could identify the element that requires investigation and thus isolate a fault. Thus the Petri-net mapping (process states) together with the process model structures (set of transitions) may provide the necessary means for an operational fault

search. The assumption would be that faults are associated with the process signals, though considering the actual process state.

For analysis purposes, Giarratano and Riley [6.16] described a tree as “a hierarchical data structure consisting of nodes, which store information or knowledge, and branches, which connect the nodes”. Peterson [6.10] showed that trees can be used as an analysis method for Petri-nets. In considering the Petri-net monitoring approach, the transitions’ data structures provide the required information to enable the diagnostic process based on tree analysis, with the symptom characterised by a timeout event indicating the tree root.

Faults could be characterised by the absence of an expected signal, within a specific process state. Assuming that the timeout condition was originated by an input place of a transition t_j , if in the relationship represented by Equation 6.12 $Q(t_j)$ is proved true, then t_j could only be prevented from firing by $S(t_j)$. A fault would then be characterised by solving Equation 6.4. In such case, if $S(t_j)$ is false, then there should be at least one false condition

$$x_{ji} = 0 \quad (\text{eq. 6.16})$$

where i represents the missing signal that prevented t_j from firing.

However, if $Q(t_j)$ is false, then the fault would be related with another event (transition). In this case, the analysis of previous missing events is required, characterising a backward search method, considered the most appropriate for diagnostic approaches [6.16]. In considering that to enable t_j it is required a marking μ that satisfies the input function $I(t_j)$, then if

$$|p_k| < \#(p_k, I(t_j)) \quad (\text{eq. 6.17})$$

p_k is preventing t_j from firing and therefore will represent a new branch in the analysis tree, leading to a transition t_m , where $p_k \in O(t_m)$.

Based on Petri-net graph properties, which enable reversibility (Chapter 5), and also on the limitations in terms of microcontroller’s available resources, the “breadth-first” search method [6.16], illustrated in Figure 6.11, was considered the most appropriate. A search should follow on until all dubious states are identified or the Petri-net initial state reached. Since Petri-nets allow loops, the software implementation should provide a method to flag those places that have already been set as branches in the tree, in order to control deadlock situations.

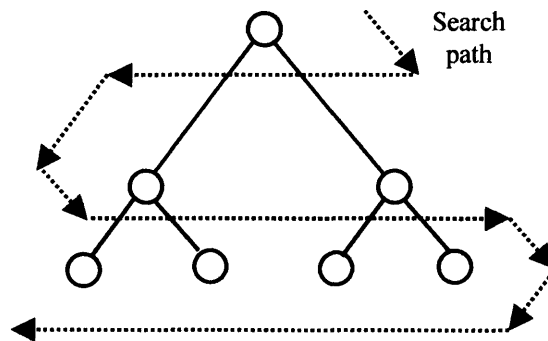


Figure 6.11: Breadth-first search-method representation (after [6.16]).

6.7 – Summary

A sequential process can be modelled in terms of a Petri-net, based on its states and the events that characterise the transitions between them. An extension to the conventional Petri-net properties was required and specified in order to interface and handle process signals. In defining a Petri-net model that describes each event as a self-contained data structure, it was proposed a method that has no hardware dependency. Furthermore, the Petri-net approach functionality was extended by enabling it to trigger the acquisition of processes’ specific parameters (a matter of implementation) and to monitor the “beginning” and “ending” of processes active states. This extended functionality must be supported by a set of messages that enable database records to be produced. Also, in considering the capability of the Petri-net approach to memorise the actual state, whilst having the knowledge about the relationship between the modelled processes states, this can be used to help in the investigation of the source of an eventual operational fault.

The definition of the proposed method was an important stage in the current research. It sets the framework for the subsequent development of the required tools for the implementation of a monitoring system. The system implementation considerations will be presented in the following chapter, with the adoption of an 8 bit microcontroller for the purpose. Additionally a layered architecture that, supported by the Petri-net approach, enables the development of a distributed structure that permits the use of the system in a wider range of applications will be presented.

REFERENCES

- 6.1 **DiCesare, F., Harhalakis, G., Proth, J.M., Silva, M. and Vernadat, F.B.** Practice of Petri Nets in Manufacturing. London, UK: Chapman & Hall, 1993.
- 6.2 **Girault, C. and Valk, R.** Petri Nets for Systems Engineering. Berlin, Germany: Springer-Verlag, 2003.
- 6.3 **Yang, S.K. and Liu, T.S.** A Petri-net Approach to Early Failure Detection and Isolation for Preventive Maintenance. *Quality and Reliability Engineering International*, 1998, 14, 319-330.
- 6.4 **Zimmermann, A. and Hommel, G.** Modelling and Evaluation of Manufacturing Systems Using Dedicated Petri Nets. *The International Journal of Advanced Manufacturing Technology*, 1999, 15, 132-138.
- 6.5 **Prickett, P. and Grosvenor, R.** A Petri-net-based Machine Tool Failure Diagnosis System. *Journal of Quality in Maintenance Engineering*, 1995, 1(3), 47-57.
- 6.6 **Davey, A., Grosvenor, R., Morgan, P. and Prickett, P.** Petri-net Based Machine Tool Failure and Diagnosis. *In Proceedings: COMADEM '96*, 16-18 July, Sheffield – UK, 1996, 723-731.
- 6.7 **Prickett, P.** A Petri-net Based Machine Tool Maintenance Management System. *Industrial Management and Data Systems*, 1997, 97(4), 143-149.
- 6.8 **Jennings, A.D, Nowatschek, D., Prickett, P.W., Kennedy, V.R., Turner, J.R. and Grosvenor, R.I.** Petri Net Based Process Monitoring. *In Proceedings: COMADEM 2000*, 3-8 December, Houston – USA, 2000, 643-650.
- 6.9 **Peng, S. and Zhou, M.** Sensor-based Stage Petri Net Modelling of PLC Logic Programs for Discrete-event Control Design. *International Journal of Production Research*, 2003, 41(3), 629-644.
- 6.10 **Peterson, J.L.** Petri Net Theory and the Modeling of Systems. Englewood Clift, USA: Prentice-Hall Inc., 1981.
- 6.11 **Prickett, P.W. and Grosvenor, R.I.** Non-sensor Based Machine Tool and Cutting Process Condition Monitoring. *International Journal of COMADEM*, 1999, 2(1), 31-37.

- 6.12 **Manders, J., Barford, L.A. and Biswas, G.** An Approach for Fault Detection and Isolation in Dynamic Systems from Distributed Measurements. *IEEE Transaction on Instrumentation and Measurement*, 2002, 51(2), 235-240.
- 6.13 **Nieva, T. and Wegmann, A.** A Conceptual Model for Remote Data Acquisition Systems. *Computers in Industry*, 2002, 47, 215-237.
- 6.14 **Jennings, A.D., Prickett, P.W., Grosvenor, R.I. and Frankowiak, M.R.** Process and Condition Monitoring using the Internet (E-Monitoring). *In Proceedings of COMADEM 2002*, Birmingham, UK:Comadem International, 2002, 45-52.
- 6.15 **Hu, W., Starr, A.G. and Leung, A.Y.T.** Operational Fault Diagnosis of Manufacturing Systems. *Journal of Material Processing Technology*, 2003, 133, 108-117.
- 6.16 **Giarratano, J. and Riley, G.** Expert Systems – Principle and Programming. Boston, USA: PWS Publishing Company, 1998.

CHAPTER 7

SYSTEM DESCRIPTION

7.1 – Introduction

Chapter 6 presented a modelling method for the implementation of monitoring systems based on the use of microcontrollers. Despite the benefits of using such devices (cost, power consumption, embedding capabilities), it must be considered that microcontrollers, when compared with computers, are limited in resources, processing power and development tools. Such issues must be addressed by a microcontroller-based implementation that aims to provide solutions for a large range of applications.

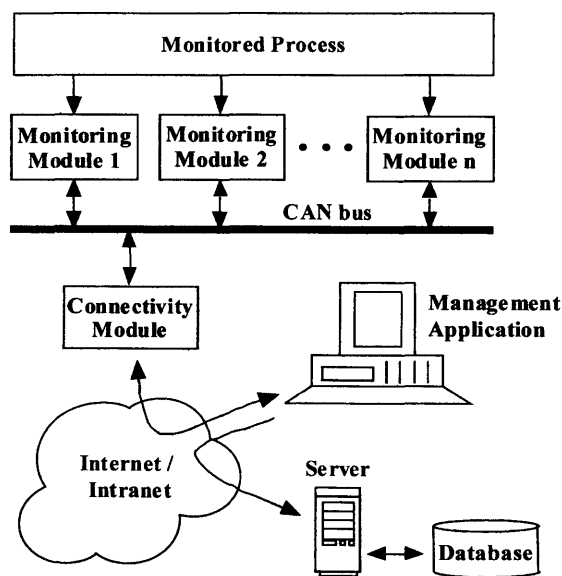


Figure 7.1: Monitoring system architecture.

In considering these factors, this research proposes the implementation of a monitoring system based on an architecture (Figure 7.1) that enables flexibility, data integration and provides resource sharing capability. In this architecture, the Monitoring Module (MM) is a data acquisition and processing module based on the PIC18C452 microcontroller. The Connectivity Module (CM) is a hardware specific

development based on PIC microcontrollers and provides Internet connectivity, therefore enabling the monitored events to be recorded in a remote database. The Management Application (MA), a software implementation based on PCs, provides a common interface to databases. A common communication bus (CAN) connects together the MMs and CM, thus providing a way to address some of the critical aspects of microcontroller-based implementations.

In providing the Monitoring System with such an architecture, cost can be kept in proportion with the application requirement by varying the number of MMs, to those needed to provide the application with its demanded resources. MMs can be deployed close to the data acquisition points, reducing special installation needs. The systems reusability, another important issue in terms of practical applications, is provided by the implementation of the method previously described in Chapter 6. The implementation of the system, with the details of each of the stated components, is described in the following sections.

7.2 – Monitoring Module (MM)

The Monitoring Module is provided with data acquisition, communication and processing capabilities. Individual descriptions will follow, considering hardware, software and modelling implementation issues.

7.2.1 – Hardware Description

A block diagram illustrating the main MM components is shown in Figure 7.2. A further detailed diagram can be found in Appendix A (A.1 and A.2). The core of the MM is a PIC18C452 microcontroller (described in Chapter 4). At the time this research was conducted it represented the best commercially available choice, in terms of the relation between facilities and cost.

In order to provide the necessary flexibility and data integration capabilities, communication assumed a great importance at different levels within the system. Each MM implements a CAN bus node. The Microchip MCP2510 was the selected

CAN controller, because of its full compatibility with those PIC microcontrollers provided with a SPI interface. Such a serial link reduces considerably the hardware design (and consequently cost), although increasing software engineering complexity. Both, the microcontroller and CAN controller, share a single 20 MHz oscillator, whose limit was imposed by the MCP2510. A CAN transceiver, UC5350 [7.1], compatible with ISO 11898 physical layer specification (Chapter 4), was employed to physically interface the CAN bus.

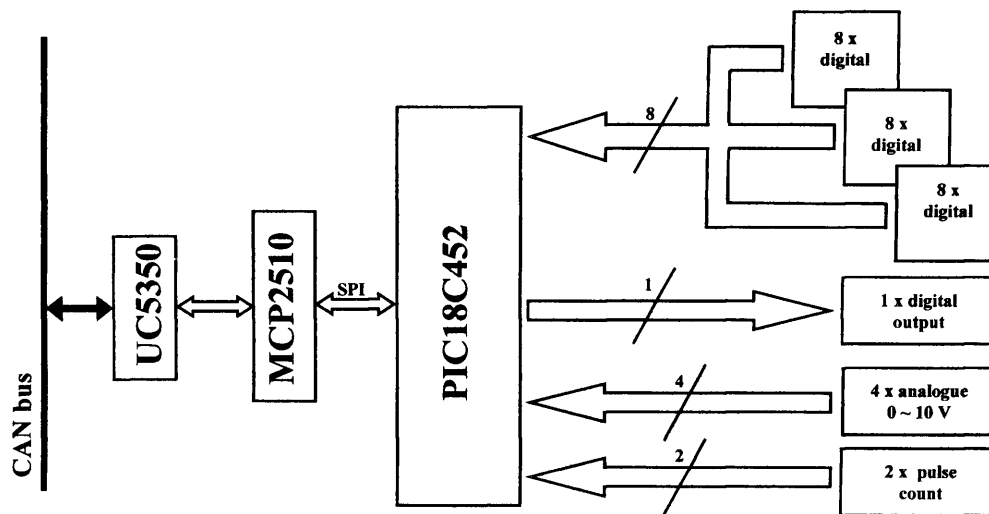


Figure 7.2: Monitoring Module hardware block diagram.

Monitoring Modules were provided with 3 different sources of signal input: digital, analogue and pulse. In order to combine simple hardware design and improved system capabilities, each MM allows up to 3 digital cards (with 8 inputs each) to be attached, all sharing one of the PIC18C452 microcontroller's input port (D). Therefore, each MM is able to interface with up to 24 digital inputs. A 3 bit port (E) was used to implement the card selection logic. Such logic was implemented in software.

Each digital input was provided with an optocoupler, in order to interface to the process signals electrical levels and ensure equipment protection. The main consideration in the selection of the optocoupler was the device response time. The HCPL-2200 [7.2] is a device that, besides fast response time, is also compatible with TTL electrical levels. An interesting feature for the proposed application is the capability of leaving the optocoupler output in a three-state mode, as long as its enable

pin is not selected. This characteristic reduced the digital card hardware design complexity, since further circuit requirements were avoided. Full electrical and timing diagrams can be found in Appendix A (A.1 and A.3).

Four analogue inputs were implemented to support special monitoring purposes. Attention was given to the fact that the microcontroller required input voltages to be no higher than 5 Volts. Manufacturer's recommendation to avoid significant offset voltages, due to analogue input pins leakage current, were taken into account [7.3]. The resulting hardware design considered the use of transducers with an output range of 0 to 10 V. No further conditioning or signal filtering method was employed. Circuit design details can be found in Appendix A (A.2).

Two pulse inputs were implemented. They were intended for special monitoring purposes, as described in Chapter 6. From the hardware perspective, these pulse inputs were connected to two of the microcontroller's port B pins, configured as external interrupts, therefore providing means to ease the software design. Each pulse input was interfaced by using an optocoupler (HCPL-2200), thus enabling a wider range of input voltages, while protecting the circuit electronics. In terms of hardware design, maximum input frequencies are determined by the optocoupler dynamic characteristics, superior to 10 MHz [7.2].

In order to support the "transition output" implementation (Chapter 6), one of the microcontroller's pins (port A, bit 6) was configured as an output. Further interfacing may be required to adapt the electrical levels or latching mechanisms to an external alarm-signalling device, depending on the application specifics.

In considering the requirement of a low-cost system, the number of externally added components was kept to the minimum required. Applications that require more resources should deploy more than one MM, exploring the flexibility offered by the system architecture. Table 7.1 gives a general idea in terms of cost of the main components used in a MM. The microcontroller employed was an EPROM based device (UV window), usually required for development purposes, therefore representing a higher cost option. A ROM or flash programming memory based device, normally employed for production purposes, would represent a reduction of

65% in this item cost. Also, the PIC18F458 has recently become commercially available. It provides similar features those found in the PIC18C452 but also has an embedded CAN controller, with cost standing at 50% of the one stated in Table 7.1 for the microcontroller.

Table 7.1: MM main components cost (prices based on [7.4]).

M	Item			Cost (£)		
	Code	Number	Description	Unit	/Item	/Module
Processor Module	Cristal 20 MHz	1	System oscilator	3.00	3.00	38.20
	PIC18C452	1	Microcontroller	20.00	20.00	
	MCP2510	1	CAN bus controller	3.50	3.50	
	UC5350	1	CAN bus transeiver	2.00	2.00	
	HCLP-2200	2	Optocoupler (pulse input)	1.60	3.20	
	74HCT04	1	Oscilator booster	0.50	0.50	
	Others		Connectors, resistors, etc	6.00	6.00	
Digital Card	HCLP-2200	8	Optocoupler (digital input)	1.60	12.80	20.80
	Others		Connectors, resistors, etc	8.00	8.00	

Prices listed represent a rough reference.

7.2.2 – Software Description

The MM software development to support the implementation of the Petri-net monitoring approach described in Chapter 6 was based on the Microchip MPLAB® development environment. PIC 18xxx family assembler was employed as the programming language and the ICE2000 emulator was used to debug the software implementations. The Microchip CAN development kit was used for the CAN node implementation and testing. A detailed description of any of these development tools can be found on the Microchip website [7.5].

Although mainly concerned in executing the process Petri-net, a number of other tasks, such as data acquisition, communication and timing were required. The flow diagram shown in Figure 7.3 illustrates the main software tasks. The microcontroller's interrupt capability was explored in order to reduce software complexity and increase efficiency. The interrupt flow diagram is shown in Figure 7.4. The software description will consider these illustrations.

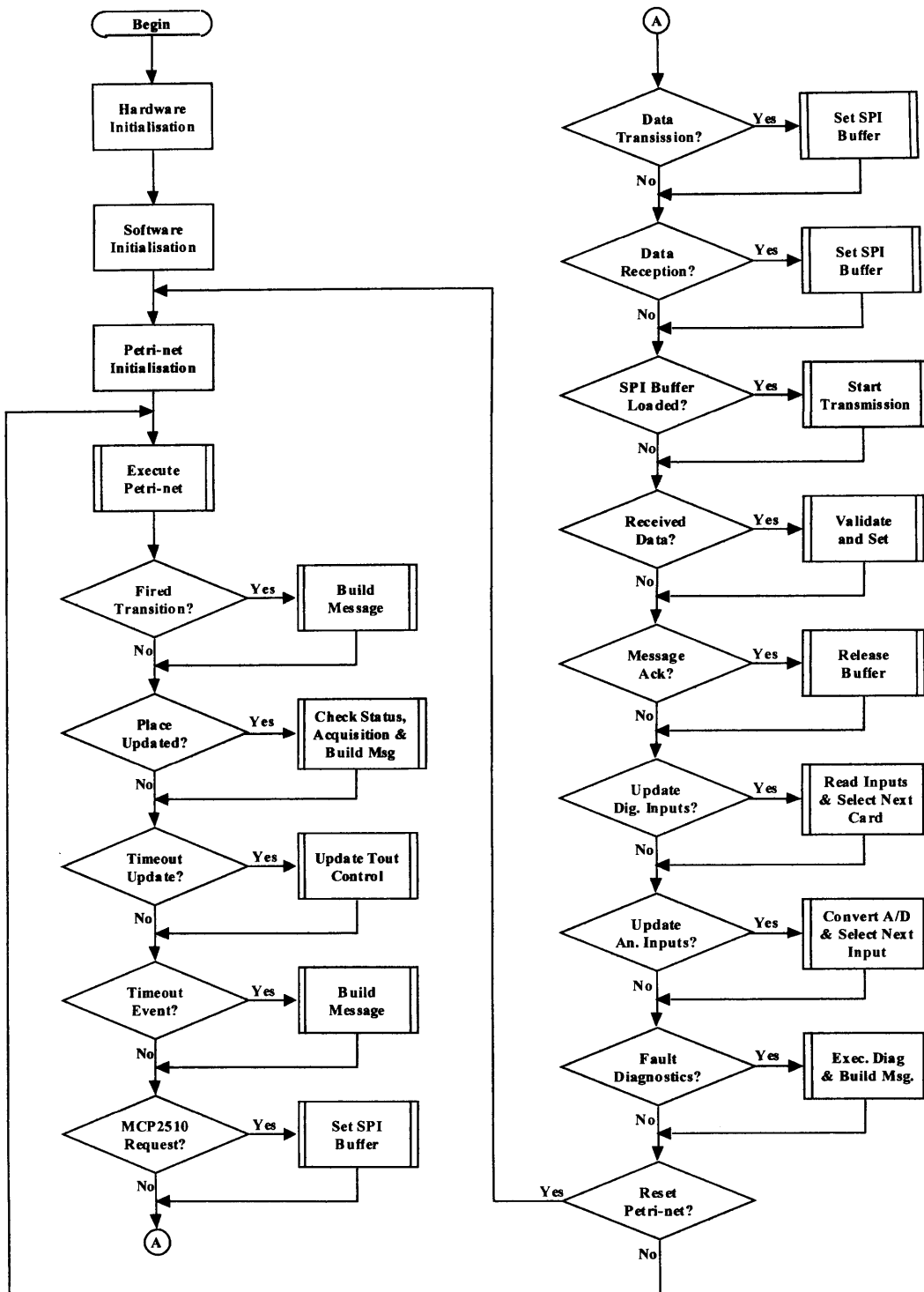


Figure 7.3: Monitoring Module application flow diagram.

The initialisation process sets variables, buffers and configures hardware devices. The microcontroller’s memory was divided into system’s variables and stack pointer, communication buffers and the Petri-net implementation area. Table 7.2 details this

distribution. Interrupts were deployed to synchronise the SPI interface data transmission/reception. A similar technique was employed in order to enable the MCP2510 to notify CAN related events, such as transmission / reception and error indications.

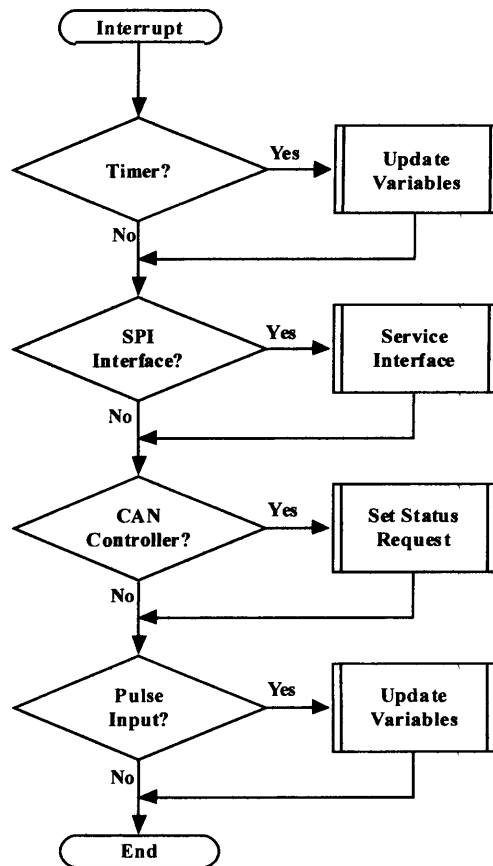


Figure 7.4: Monitoring Module interrupt service flow diagram.

Each Monitoring Module supports 254 transitions, 254 places and 255 sub-net IDs. The implementation considered the application's Petri-net description to be based on three attached text formatted files. The first of such files represents the Petri-net main structure (event descriptions), MM identification and the identification of the places required to provide timeout events. The second file identifies the Petri-net's places representing the process states required to have their active status watched (and reported). This was defined as a 32-byte structure, in which 1 bit is used to represent each possible place. The last attached file correlates places with analogue or pulse inputs. It contains a table with 254 inputs, sequentially representing the Petri-net's

places and indicating an input source, analogue or pulse (“0” meaning no source associated), used to trigger the data acquisition of process specific parameters. Examples of such files can be found in Appendix B (B.1 to B.14).

Table 7.2: Monitoring Module data memory distribution.

Buffer Description	Buffer Size (bytes)
System variable	256
Software stack	32
SPI transmit buffer	96
SPI receive buffer	32
Transmit message buffer (built)	16
Receive message buffer (rebuilt)	16
Event record buffer	64
Petri-net places buffer	256
Petri-net places timeout control buffer	512
Petri-net sub-net buffer	32
Timeout devices buffer	150
Active places mapping buffer	32

Places were defined in the microcontroller’s data memory as a continuous set of 256 bytes (only 254 effectively used). The place identification indexes the place location within this data structure. Each place location will hold its respective number of tokens, being updated by the Petri-net execution. Places have also individual timeout control, implemented as a 16 bits variable. The maximum timeout record was defined as 65,535s. The variables will be initialised with this maximum value and updated each time the respective place is processed. The smallest time unit is 1s. A bit map structure was also provided, with 1 bit to represent each place, in order to control the status of individual places and to ease status change identification.

The Monitoring Module makes public a sub-net event by broadcasting its identification (sub-net ID). Internally, sub-nets will be assigned in a bit mapped data structure (32 bits), with one bit representing each sub-net ID. The Petri-net execution, when required, searches for sub-net events and updates this bit-mapped structure.

One of the microcontroller’s timers (TMR0) was configured to generate a 1 ms time base that is used to update the Monitoring Module date/time record. Such record

follows the “datetime” format defined and used by Microsoft’s SQL2000® DBMS [7.6]. The microcontroller’s interrupt functionality was employed in order to generate a precise and reliable timing method. The same time base will be used in other tasks that require time measurement.

Digital input updating was synchronised with the 1 ms time base generated by TMR0. Since there are a possible 3 digital cards, all using the same microcontroller’s port (D), switching time had to be considered. The employed procedure reads the 8 digital inputs of the selected card and then identifies and moves on to the following one, which will be read in the next acquisition cycle, thus ensuring enough time to make the bus stable when the next update is carried out. By using such an approach, digital inputs are updated every 3 ms (333.33 updates / second).

The pulse inputs were configured to automatically generate interrupts whenever such an event is matched. Counters (one for each input) will be incremented during the interrupt service. The counters will be read and reset by the system’s application every 1 s, providing a monitoring parameter in term of pulses / second (P_m). Such a parameter might be further used by an “analogue transition” and as basis for a “special record” (Chapter 6).

Analogue inputs were configured for 10 bits resolution. They are updated periodically, by polling the A/D converter in the system’s application execution main loop (Figure 7.3). Based on the microcontroller’s analogue channels specifications and software implementations, the sampling rate is approximately 2.5 K sample / s. Small variations may occur due to different tasks being performed in different execution loops. The sampled analogue input data will be integrated over a period equivalent to 256 samples, resulting in an average value, as represented by Equation 7.1. The result may be further used as an input parameter into “analogue transition” and to calculate the mean value of the analogue channel observation (special record).

$$A_m = \left(\sum_{i=1}^{256} S_{ANi} \right) / 256 \quad (\text{eq. 7.1})$$

Following the approach described in Chapter 6, Petri-net's places could be used to trigger the acquisition of process specific parameters, resulting in a "special record". In this implementation such a parameter was defined as the mean value of the observed analogue or pulse inputs, based on observation time. To obtain such a parameter, the system will perform a calculation based on Equation 7.2. Previous calculations (P_m for pulse inputs and A_m for analogue inputs) will be used as inputs in the equation (S_i). Such values will be continuously added, as long as the process state remains active. The resulting sum will be divided by the number of added samples (n).

$$M = \left(\sum_{i=1}^n S_i \right) / n \quad (\text{eq. 7.2})$$

The coefficient n is limited in practical terms by the size of the variable that contains it to 65,535 (16 bits). For an analogue channel, considering a sample rate of 2.5 K samples / s and also that a new value will be added once after 256 samples (A_m), the maximum observation time will be limited to 1 hour, 52 minutes and 36 seconds. Since the microcontroller does not provide an instruction to perform division, a 16 bit division routine based on an algorithm for 8 bits processors presented by Leventhal [7.7] was employed in the calculation to maximise efficiency.

The monitoring Petri-net will be run by executing the transitions defined in the Petri-net table (text format file). The data retrieved from this table will be verified in the transition structure context (Chapter 6), thus checking whether or not the transition is enabled. Transition firing actions will include updating input and output places, requests of sub-net broadcast and event messages. Transitions will execute sequentially in the order in which they were defined in the Petri-net table, though handling one transition each time, per cycle. For those fired transitions that require a message, a record will be stored in the "event record buffer".

Whenever places were updated (following a Petri-net execution), further verification will be carried out. Such verification will aim into identify the "beginning" or "end" of an active state of selected places or to trigger the "mean value calculation" of

analogue or pulse input. When required, a message transmission will be requested by inserting a record in the application “event record buffer”.

Places enabled to have a timeout control and whose active state lasted longer than has been previously recorded will produce a timeout record, stored in the “event record buffer”. The default approach compares the actual cycle with previous one. However, such comparison parameters can be fixed by supplying a command, received through the data communication interface (CAN).

Fault isolation was implemented following the approach described in Chapter 6. This procedure will be started when a “diagnostics request” identifying the place that provided the symptom (timeout) is received through the data communication interface. The Petri-net will be searched in order to find a transition having the provided place as an input. Verifications will be made to detect whether the fault relates to the transition input signals or to other places that failed to enable this transition. Transitions and places will be alternatively investigated using the “breadth-first” search approach [7.8]. Successive interactions will be carried out until a result is obtained or the entire set of transitions is investigated. Loops will be avoided by marking places that have already been verified. Records will be placed in the “event record buffer”, identifying the transition and the signal(s) that failed to enable the transition. Sub-net, digital and analogue / pulse inputs are considered as possible sources of faults.

7.2.2.1 – Data Communication Aspects

In order to improve system’s efficiency and ease software development, data buffers were set at different levels. Referring to Table 4.2, the “event record buffer” was implemented to hold a number of monitoring events recorded by the system. Each of these records will be converted into a system’s message, then assembled accordingly one of the formats shown in Figure 7.5 and stored in the “transmit message buffer”. This buffer is capable of handling only one message each time.

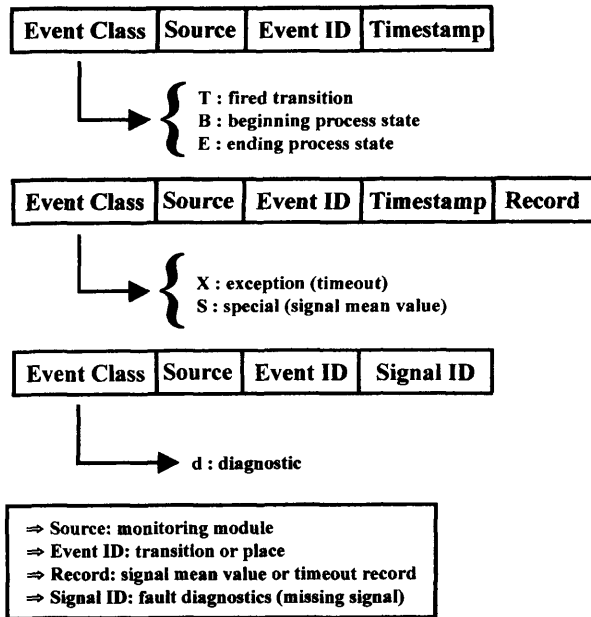


Figure 7.5: Monitoring Module transmitted messages formats.

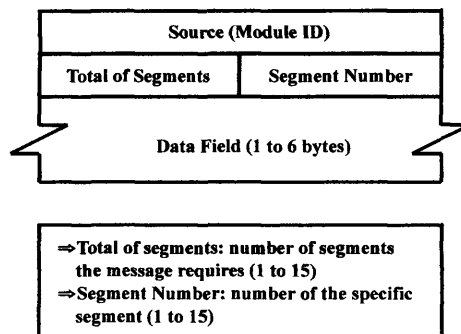


Figure 7.6: System’s application layer protocol with message segmentation.

At the next stage, the system’s messages are handled by a “CAN application layer”, before being stored in the “SPI transmit buffer”. At this application layer, messages will be sized accordingly to the CAN protocol, and split when required. A sequencing method was developed, to enabled messages to be reassembled at the destination end. Details relating to the “application layer protocol” are shown in Figure 7.6. A transmission timeout feature was implemented to control message delivery. Transmitted messages that were not acknowledged by the recipient in a predefined time will be retransmitted. The CAN controller’s commands are required to be appended to the application layer message, in order to properly set up the required

task. The “SPI transmit buffer”, will also be used to request the controller’s status and carry out configurations.

Messages received from the CAN bus, such as sub-net broadcast and system commands, will be transferred to the “SPI receive buffer” and then reassembled in the “receive message buffer”. Figure 7.7 shows the format of received messages supported by an MM. Figure 7.8 shows special purpose messages formats. Once validated, commands will be executed or will result in flags requesting further actions (i.e. fault diagnostics).

Command ID		Record
Description	Command ID	Record
Reset Petri-net	/R	Void (0 bytes)
Set time	/T	Actual time in ms (4 bytes)
Set date	/D	Actual date (4 bytes)
Sub-net	/E	Sub-net event (1 byte)

Figure 7.7: System commands - broadcast messages format.

Module ID	Command ID	Place ID	Record
Description	Command ID	Place ID	Record
Set timeout	/t	Target place	Timeout parameter (2 bytes)
Fault diagnostic	/d	Starting place	Void (0 bytes)

Figure 7.8: System command - specific messages format.

The CAN controller filtering feature, based on the protocol message priority, was exploited to ease the software implementation. Sub-net messages are “visible” to Monitoring Modules, but not to the Connectivity Module. Conversely, monitored

event messages (fired transition, begin/end of places, places timeout and parameters mean value records) will only be received by the Connectivity Module.

The Monitoring Module application required 8,426 bytes of program memory. Considering that the data tables representing the process states to be watched (beginning / end) and those that trigger analogue/pulse acquisition require a fixed amount of 288 bytes (32 + 256), the representation of a process Petri-net can dispose of 24,054 bytes. Assuming as a general example an ordinary transition with 2 input and 2 output places, a Petri-net with a maximum of 254 of such transitions would require 5,588 bytes of program memory, which is much less than the total available. Appendix A (A.4) presents the software development main body, coded in Assembler.

7.3 – Connectivity Module (CM)

The CM was implemented to provide a common Internet interface to all MMs sharing the system's CAN bus, releasing them from heavy communication tasks such as those related with the Internet protocols. At the same time, by concentrating the entire monitoring task in a single Internet socket, management complexity was reduced. This was not developed purely as a connectivity solution, but was meant to support the distributed monitoring approach implementation. Nevertheless, in considering the use of microcontrollers for such an application, investigations were required in order to optimise the protocol's implementation.

7.3.1 – Hardware Description

The CM implementation was based on the Microchip PICDEM.NET™ Internet development kit [7.9]. The selected microcontroller was the PIC18C452, mainly due to its data memory availability and organisation. The PICDEM.NET is capable of interfacing an Ethernet network by means of an onboard Realtek RTL8019AS™ Ethernet controller [7.10]. This is provided with a 16 Kbytes embedded RAM, shared as reception and transmission buffers. The network physical connection is based on a twisted-pair (10BaseT), with data transmission rates of 10 Mbps. The operating clocks are 20 MHz for the Ethernet controller and 19.66 MHz for the microcontroller.

A Microchip MCP2510 CAN Development Board [7.11] was employed to implement the CM’s CAN node. The PIC18C452 was selected as the CAN node processor. This board uses a single 16 MHz oscillator for both CAN controller and microcontroller. An asynchronous serial communication link, based on the RS-232C standard [7.12], was employed to connect both modules. Figure 7.9 shows the hardware diagram.

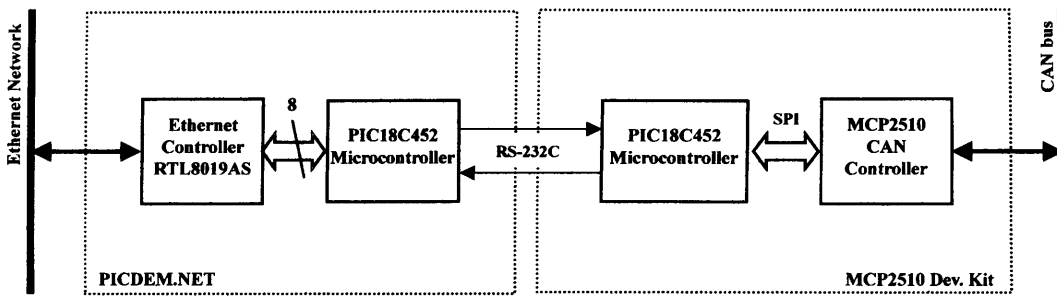


Figure 7.9: Connectivity module hardware block diagram.

7.3.2 – Software Description

In order to enable messages representing monitoring events to flow on the Internet and commands flowing in the opposite direction to reach the MMs, the CM has to implement the required gateway functionality. Figure 7.10 illustrates the different levels of protocols and implementations necessary to interface each network, which will be considered in the following section.

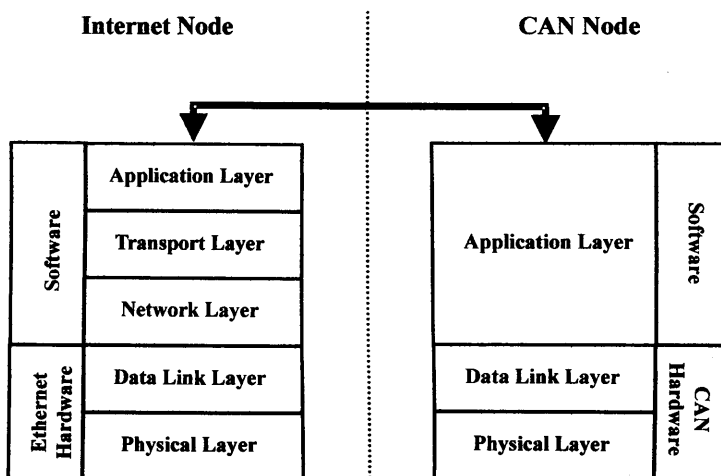


Figure 7.10: Connectivity module application tasks diagram.

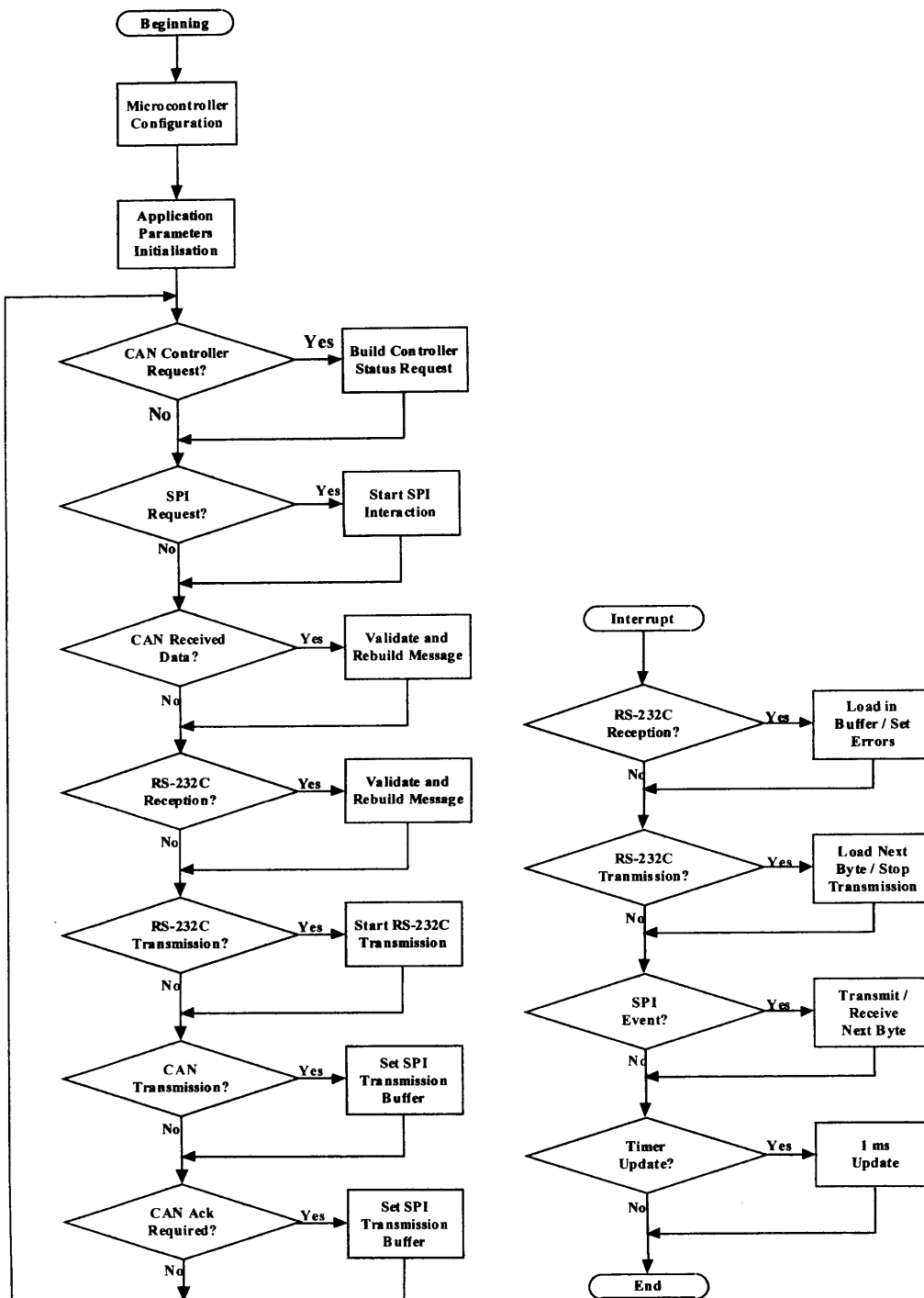


Figure 7.11: CAN node implementation flow diagrams.

7.3.2.1 – CAN Node Implementation

The software implementation of the CM’s CAN node was based on MPLAB-IDE development environment and used PIC’s Assembler as the programming language.

The flow diagram shown in Figure 7.11 illustrates the main tasks executed. In general terms, the application “watches” the CAN bus for MMs’ messages, reassembling, validating and replying them with acknowledgement messages. This message will then be forwarded to the Internet component. Similarly, messages arriving from the Internet, will be split, sequenced and transmitted on the CAN bus. Basically, the CAN protocol’s physical and data link layers (Figure 7.10) will be provided by the CAN hardware. The application layer functions, such as splitting, sequencing and reassembling messages, were implemented in the software application.

The microcontroller’s embedded communication devices were employed to support the application. Individual interrupts were selected to synchronise the serial asynchronous interface events. A single interrupt was used for the SPI interface. The CAN controller MCP2510 status will be polled once each time the application main loop is executed, in order to retrieve the device status. One of the microcontroller’s embedded timers (TMR1) was configured to provide a 1 ms time base, used by the CAN application layer to implement the messaging timeout feature.

Table 7.3: Data memory requirement to implement the CAN node.

Buffer Description	Buffer Size (bytes)
System variable	56
Software stack	32
SPI transmit buffer	96
SPI receive buffer	32
CAN to RS-232C intermediate buffer	64
RS-232C to CAN intermediate buffer	64
RS-232C transmit buffer	64
RS-232C receive buffer	64

The software implementation required 7,398 byte of program memory, meaning that another 25,370 bytes were left unused. In terms of data memory, it required 472 bytes, with 1,064 remaining unused. Table 7.3 summarises the memory usage. In considering that the PIC18C452 supports operating frequencies up to 40 MHz, it can be seen that the microcontroller was far from its total capability. Appendix A (A.7) presents the software development main body, coded in Assembler.

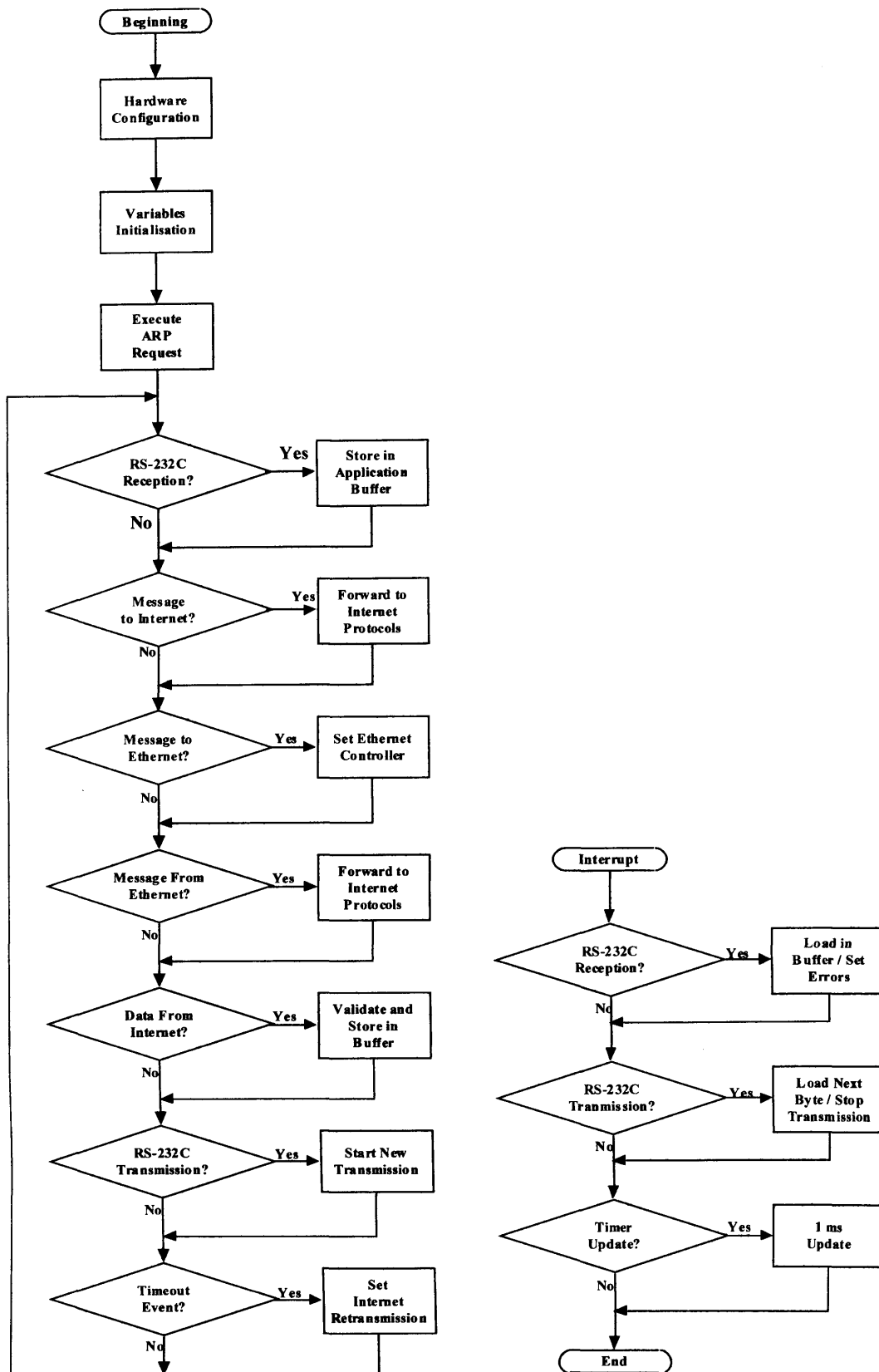


Figure 7.12: Internet implementation flow diagram.

7.3.2.2 – Internet Connectivity Implementation

The Internet connectivity application was developed using a C programming language environment for PIC microcontrollers, WIZ-C version 8.05 [7.13]. The choice of such an environment was made considering the intensive data manipulation and the use of more complex data types required by this sort of application [7.14], with increased software engineering complexity. Figure 7.12 shows the flow diagram, illustrating the application's main elements. Appendix A (A.9) presents the software development main body, coded in "C". Figure 7.13 shows the microcontroller's required resources, displayed in the compiler's output window.

```

PIC RAM
=====
Compiler Overhead : 13
Loc Opt : 4
Globals : 492
Free RAM: 1040
Start of C Program at 758 (0x02f6)
Total C & Library Program words 7868
Breakpoint Hit at main, 0394

```

Figure 7.13: Internet implementation required resources – compiler output.

7.3.2.2.1 – Protocols Implementation

Basically, the physical and data link layer protocols were provided by the Ethernet hardware (Figure 7.10). However, Ethernet frames are required to be assembled in the microcontroller's memory, before being handed to the Ethernet controller. Such a frame is shown in Figure 7.14. The checksum will be calculated and automatically appended after the data field, by the controller. Although supporting interrupts, such feature was not explored in the hardware design, therefore requiring the microcontroller to poll the Ethernet controller in order to retrieve the interface status. The Ethernet "level" will provide the data link layer services to the ARP and IP protocols.

The system needs to know the IP address of the remote location to which to send the monitoring records (Management Application). ARP request commands will be issued to search for the hardware address (Ethernet) of the remote IP. In considering the application purpose, only one single destination socket will be supported. During normal execution (after initialisation), the system will respond to incoming ARP requests by replying with the ARP response messages (Chapter 4).

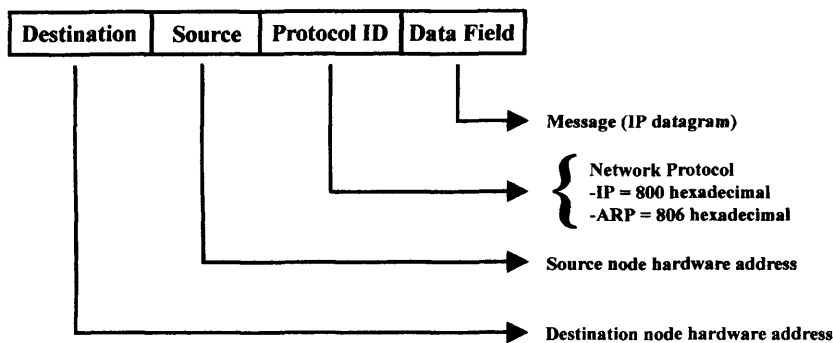


Figure 7.14: Ethernet frame fields mounted in the microcontroller’s memory.

At the network layer, the IP protocol was implemented, following the description provided in Chapter 4. Simplifications were made, considering the application’s requirements. The messages exchanged by the system will never exceed the minimum length defined in the protocol specification, therefore the implementation does not requiring datagram fragmentation or a reassembling mechanism (data padding must be provided). The IP implementation will provide the service to the Internet transport layer and the Control Message Protocols (ICMP).

The system implementation of the ICMP was restricted to receiving and answering such control messages. The ICMP will be used by the network to inform the CM about an “unreachable” remote application (MA).

The transport layer implementation was based on the User Datagram Protocol (UDP), rather than the Transmission Control Protocol (TCP), which is much more complex to implement and usually employed for secure data exchange. A similar approach was used by Al-Habaibeh et al [7.15], in an embedded monitoring application. The factors considered for such a choice here were the system’s requirements and implementation

simplification. The messages issued by the MMs will not require the segmentation feature provided by TCP. Furthermore, a CM represents a single application port, thus does not require heavy application management. Therefore UDP's connectionless approach was considered appropriate for the transmission of data records, since it avoids the overhead represented by TCP in such cases [7.16], especially when the asynchronous nature of monitoring events is considered. Nevertheless, it must be considered that UDP, apart from the datagram checksum, will not provide any control mechanism to ensure data delivery. Therefore, the application layer serviced by UDP must provide such control.

The Internet application layer implementation was concerned with the message exchange mechanism at the Internet level. At this layer, messages will have appended a header, providing the message purpose, the sequence number and length, to enable message delivery control. The application layer header is shown in Figure 7.15. Messages transmitted must be acknowledged by the destination. Such confirmation will be provided by the recipient replying with an acknowledgement message containing the received message sequence number. Messages that are not confirmed in an established time will be automatically retransmitted. One of the microcontroller's embedded timers (TMR0), configured to generate a 1 ms time base, was deployed to provide the timeout approach required to support this control.

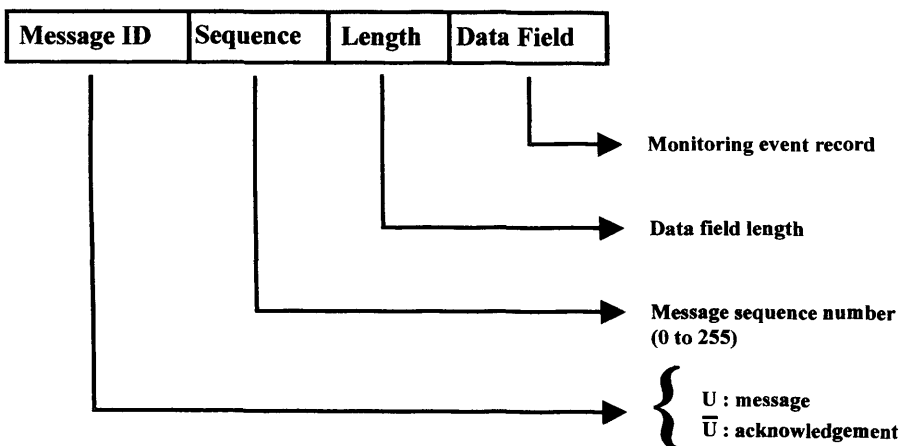


Figure 7.15: Application layer message format.

7.3.2.2.2 – Operation Description

After initialisation, an ARP request will be issued, in order to obtain the Ethernet address associated to the IP address of the destination node (MA). The application layer will handle the records received from the MMs, in order to perform the messaging control. These messages will be forwarded to the UDP and then to the IP protocol implementations, where checksums will be calculated and respective headers added. The Ethernet frames will be assembled and transferred to the controller, which assumes the transmission task at the network level, providing information about the status of the transmission. The message will be kept assembled at the application layer level until delivery confirmation was received.

Reception of messages from the Internet, for the MM, can be described similarly. Ethernet frames will be received by the controller, verified and forwarded to the Internet or ARP protocols. At the IP level, checksum, source and destination addresses will be verified. The data field will be handed to the upper layer (UDP or ICMP) for further verification or actions complying with the protocol specifications. At the application level, a validated message will be stored in the serial transmission buffer, while an acknowledgement message will be returned to the sender.

Although the implementation of the connectivity module was based on two separate microcontrollers for simplification reasons, the analysis of the resources required showed that a single of such device would have been enough. In terms of programming memory, the CAN node implementation required 7,398 bytes, while the Internet protocols implementation used 15,736 bytes, which together (23,134 bytes) still less than the 32,768 bytes available in each microcontroller. In a similar analysis considering the data memory, it can be seen that both applications together required 968 bytes, again less than the 1,536 data memory bytes available in a single microcontroller. Nevertheless, supposing that both applications were developed using C programming language, the compiler overhead should be taken into account. On the other hand, in the resulting implementation many resources were used to implement the serial link between the two hardware modules, including some duplicated buffers. In terms of processing power, the microcontroller supports a clock twice the one used by the applications, thus exhibiting a factor for further improvements.

From a cost perspective, the Connectivity Module based on the PICDEM.NET hardware, provided with a CAN node implementation, would represent, roughly estimating, nearly £ 210.00. This takes into consideration the module basic hardware (Microchip reference DM163004), the CAN controller (MCP2510) and transceiver (UC5350) [7.4]. Assuming that a single CM can provide Internet connectivity to several MMs (at a location), the system still represents a low-cost implementation. Additional cost benefits may arise from customised implementations, as a result from large-scale applications.

7.4 – Management Application (MA)

The Management Application was implemented with the aim of providing a common interface to databases where the monitoring events will be stored. This application was developed in a Microsoft Visual C++™ 6.0 environment, making use of Microsoft Foundation Classes (MFC®) library. It executes on any computer running Microsoft Windows® operating system, connected to the Internet. A single management session can be used to provide support to several remote monitoring “structures”, connected to the Internet at different locations. For practical reasons, such number was limited to 10, all required to use the same database.

The requirement for such an application was based on practical reasons. The task of directly supporting many different (commercially available) Database Management Systems (DBMS) is a hard task for computer-based implementations [7.17]. Such a task would become even more challenging for a microcontroller-based application, when considering the processing power and available resources. In considering such factors, it was concluded that a MA, benefiting from the Open Database Connectivity (ODBC), would provide a great level of flexibility to the system implementation.

The MA is able to select a system’s database, define the local socket (IP address and MA port number), configure the remote (CM) IP address and assign a name to the monitoring task, which will be used by the system to create a set of tables where monitoring records will be stored. These tables will be automatically created, being identified by the monitoring task name, followed by “_EV” (process events), “_ST”

(process states) and “_DG” (diagnostics table). These tables will be discussed later. The remote application port identification (CM) is not required, since such information is assumed static (the same for all CMs) and thus is hidden from the user.

After settings are made communication will be enabled to take place. The monitored events, recorded by the MM, will be sent to the MA over the Internet, making use of the CM. Records received by the MA will be displayed on the application window, while forwarded to the database. They will therefore be available for further analysis or presentation.

The MA can be used to issue commands toward the Monitoring Modules. Such commands include “set time”, “set date” and “reset Petri-net”. Also, the MMs fault diagnostics request and timeout setting will be managed by the MA. A fault diagnostic request requires the provision of the MM identification and also the place (symptom) to start the search. MMs’ timeout setting requires a template table containing the timeout parameters to be supplied. Such a table must be externally created, but must be part of the same database.

7.4.1 – System Data Tables

As introduced previously the Monitoring System employs 3 tables to store records associated with the tasks performed by the monitoring system. The first of these tables (*prefix_EV*), will store monitoring events, specifically fired transitions, timeout and analogue/pulse monitored parameters (special record). Table main fields (columns) include the Monitoring Module, the event (transition or place that originated it), the class that identifies the event, the event timestamp and a record field, eventually used to store the timeout information or an analogue/pulse mean value calculation.

A second table will store the status of the process states (*prefix_ST*). This table is aimed to provide time information of specific process states, indicating the “beginning” and “ending” of such states (Petri-net places). The MA will calculate the duration of such state after the “end record” was received. Another field was provided to indicate those process states that remain active and those that have already finished. Only records with class field “B” and “E” (Figure 7.5) will be inserted in this table.

The result of a diagnostics request will be stored in a table identified as “*prefix_DG*”. Each record will consist of the Monitoring Module, the transition (identification), the signal source (“d” – digital or “a” – analogue/pulse) and the identification of the signal to which the fault was related. Every time diagnostics is requested, the table will be cleared. Therefore, only the records related with the last diagnostic will be listed. Description fields were provided in order to allow further descriptive information to be added. Such information field will not be updated by the MA, requiring other database method to be employed (procedures, triggers, etc) [7.6].

The use of database systems to store monitoring records provides an easy way to integrate such records within different applications, with a minimum of software effort. Modern DBMS enable databases to be deployed in a distributed configuration [7.18]. This provides an ideal scenario for the use of the Internet in monitoring applications, therefore contributing to stimulate a wider use of the technology.

7.5 – System Tests and Measurements

Although it was not an aim of this research to investigate the design of data acquisition hardware, tests were carried out in order to assess the system’s functionality regarding the analogue and pulse inputs. The test procedure was based on the implementation of a test-Petri-net to trigger the acquisition of the 4 analogue channels and 2 pulse inputs, with the results stored in a database for further analysis. Details of this implementation, such as the Petri-net diagram, descriptive text files and graphic results can be found in Appendix A (A.14 to A.20).

7.5.1 – Linearity

Analogue input linearity was tested by applying an adjustable DC power supply and varying the voltage progressively from 0 to 10 V. A similar approach was employed in a regressive manner. A set of graphs showing the results of the test can be found in Appendix A (A.17). It can be seen that best results were verified between 2 V and just before 10 V. Low input levels were more affected by the analogue inputs offset voltage, because of the analogue pins drain current. At the upper range (10 V), the

saturation of the analogue input channels can be considered as the cause of such deviation trend. The graphs also showed that the inputs behaved similarly when the voltage was varied progressively and regressively.

Pulse input linearity was tested by using a signal generator (square wave) as an input, varying frequencies from 0 to 65 KHz and then oppositely from 65 KHz to 0. Graphs representing the results were included in Appendix A (A.18). The system operation was limited in software to approximately 65 KHz. In terms of error percentage, best results were obtained at higher frequencies. It can be justified by the fact that the system calculation is based on integers. At low frequencies a 1 Hz deviation could represent a considerable error percentage. In such cases performance might be improved by increasing the acquisition time (larger number of samples). The test was based on 15 s acquisition periods. The system behaved in a similar manner, either varying input frequency progressively or regressively.

7.5.2 – Repeatability

The repeatability test was performed by repeating the linearity test 3 times. The sets of data of each measurement were linearised and equations used to generate a standard set of data, comparing the results of the first against the second and third. Graphs showing the test results can be found in Appendix A (A.19). It can be seen that in terms of error rates, both analogue and pulse inputs have a better performance with higher input voltages and pulse rates, respectively. The justification for such behaviour is the same given for the linearity deviation, with analogue inputs compromised by the offset voltage and pulse measurement resulting in higher error percentages and therefore affecting repeatability at very low frequencies.

7.5.3 – Analogue Input Mean Value Accuracy

In order to test the system mean value calculation approach, a test consisting of applying a square wave form to the analogue inputs and varying the signal's frequency was carried out. The graph representing the result of such test can be found in Appendix A (A.20). In considering the system's limitation in terms of linearity, the mean value of the input signal was set at 2.5 V. It can be seen that the system provides

a good response at lower frequencies, with an increase in the error trend after nearly 800 Hz. The system analogue input sampling rate, around 2.5 K samples/second, represents a constraint for a better accuracy for input signals at higher frequencies. Similar results were obtained when increasing the input signal mean value, with errors below 1 % between 0 and 800 Hz. Input signals with mean value below to 1.5 V resulted in a general increase in the error level, specially at very low frequencies (tending to DC), as it would be expected due to considerations made when analysing the system's linearity response.

The test results showed that the system would require further improvements in terms of hardware design and analogue inputs sampling rates, in order to provide a better accuracy. As it was stated before, this was not a major concern of the research. However, such improvements could be achieved by increasing the microcontroller operating frequency (from 20 MHz to for 40 MHz), making use of interrupt techniques associated to the analogue acquisition and adding additional hardware components to enhance analogue signal conditioning.

7.5.4 – System Communication Testing

The system development required the implementation of communication capabilities. To develop and test such capabilities, tools to monitor the communication networks were required. Microchip MCP2510 Development Kit [7.11] was employed as the CAN bus monitoring tool. It allowed system's messages flowing on the CAN bus to be captured and analysed and when required, to transmit messages produced using the Kit's software. At the Internet side, free demonstration software, developed by LANSleuth [7.19], was used as the monitoring tool. This software monitoring capabilities are those related to Ethernet and Internet protocols. Ethernet frames flowing on the network were captured and displayed, showing the respective protocols' fields at the different layers. Appendix A (A.21 to A.24) shows examples of these tools.

7.6 – Summary

The use of microcontrollers for the implementation of the monitoring approach based on Petri-nets was presented in this chapter. In order to provide low-cost without compromising efficiency and capability, an architecture that explores the potentialities of individual and specialised levels was proposed. Monitoring modules that incorporate the task knowledge will deliver event records based on local decisions to a communication element that concentrates a set of complex protocols. Databases are interfaced by a PC based management application, benefiting from a set of well developed and reliable standards and protocols, therefore easing the implementation and dealing with critical aspects such as security, distributed data access, analysis and presentation. In using established standards in such a development, it is ensured that the proposed system can benefit from further technological enhancements and support a wider range of applications.

The following 3 chapters will present examples of the use of the system in monitoring applications. The first case considers the monitoring of the production cycle of a scale model that mimics a hydraulic press. The second example illustrates the monitoring of a laboratory rig representing a manufacturing system. The last example was based on the monitoring of a tool changer of a CNC machine centre. Each example was proposed in order to illustrate specific capabilities of the system, considered of significance in the development of the research.

REFERENCES

- 7.1 Texas Instruments Inc. UC5350 CAN Transceiver, Texas Instruments Products Web Site, Available from <http://www-s.ti.com/sc/ds/UC5350.pdf> [Accessed 4 July 2003].
- 7.2 Agilent Technologies, Product Information and Literature, Agilent Web Site, Available from <http://literature.agilent.com/litweb/pdf/5965-3596E.pdf> [Accessed 6 June 2003].
- 7.3 Microchip Technology Inc. PIC18CXX2 Data Sheet. USA: Microchip Tech. Inc., 1999.
- 7.4 Farnell Electronic, Farnell Web Site, Available from <http://www.farnell.co.uk> [Accessed 9 June 2003].
- 7.5 Microchip Technology Inc. Microchip Products Web Site, Available from: <http://www.microchip.com/products> [Accessed 14 April 2003].
- 7.6 **Vieira, R.** SQL Server 2000 Programming. Birmingham, UK: Wrox Press Ltd, 2000.
- 7.7 **Leventhal, L. A.** Z80 Assembly Language Programming. Berkeley, USA: Osborne/McGraw-Hill, 1979.
- 7.8 **Giarratano, J. and Riley, G.** Expert Systems – Principle and Programming. Boston, USA: PWS Publishing Company, 1998.
- 7.9 Microchip Technology Inc. PICDEM.net Internet/Ethernet Demonstration Board, Microchip Web Site, Available from <http://www.microchip.com/1010/pline/tools/picmicro/demo/pdemnet/index.htm> [Accessed 4 July 2003].
- 7.10 Realtek Semiconductor Corp, Realtek 8019AS Product Description, Realtek Web Site, Available from <http://www.realtek.com.tw/products/products1-2.aspx?modelid=1> [Accessed 01 July 2003].
- 7.11 Microchip Technology Inc. MCP2510 Development Kit User's Guide, Microchip Web Site, Available from <http://www.microchip.com/1010/pline/analog/anicateg/interface/can/devices/mcp2510/9640/index.htm> [Accessed 04 July 2003].

- 7.12 **Kochhar, A.K. and Burns, N.D.** Microprocessors and their Manufacturing Applications. London, UK: Edward Arnold Ltd., 1983.
- 7.13 Forest Electronic Developments, PIC C Compilers, Forest Web Site, Available from <http://www.fored.co.uk> [Accessed 20 November 2002].
- 7.14 **Bentham, J.** TCP/IP Lean – Web Servers for Embedded Systems. Lawrence, USA: CMP Books, 2000.
- 7.15 **Al-Haibaibeh, A., Whitby, D. R., Parkin, R. M., Jackson, M. R., Mansi, M. and Coy, J.** The Development of an Internet-based Mechatronic System for Remote Diagnostic of Machinery Using Embedded Sensors. *In Proceeding: ICOM 2003 - International Conference on Mechatronics*, 18-20 June, Loughborough – UK, 2003, 297-302.
- 7.16 **Hall, E.A.** Internet Core Protocols – The Definitive Guide. Sebastopol, USA: O'Reilly and Associates, 2000.
- 7.17 IBM Corporation, IBM BookManager® BookServer Library, IBM BookServer Web Site, Available from: <http://publib.boulder.ibm.com/cgi-bin/bookmgr/FRAMESET/QBKACO03 /APPENDIX1> [Accessed 08 March 2001].
- 7.18 **Blaha, M.R.** A Manager's Guide to Database Technology. New Jersey, USA: Prentice Hall Inc. 2001.
- 7.19 SSI Embedded Systems Programming, LANSleuth DEMO Ethernet Monitor, SSI Web Site, Available from <http://www.lansleuth.com> [Accessed 24 July 2003].

CHAPTER 8

PRESS RIG MONITORING TASK

8.1 – Introduction

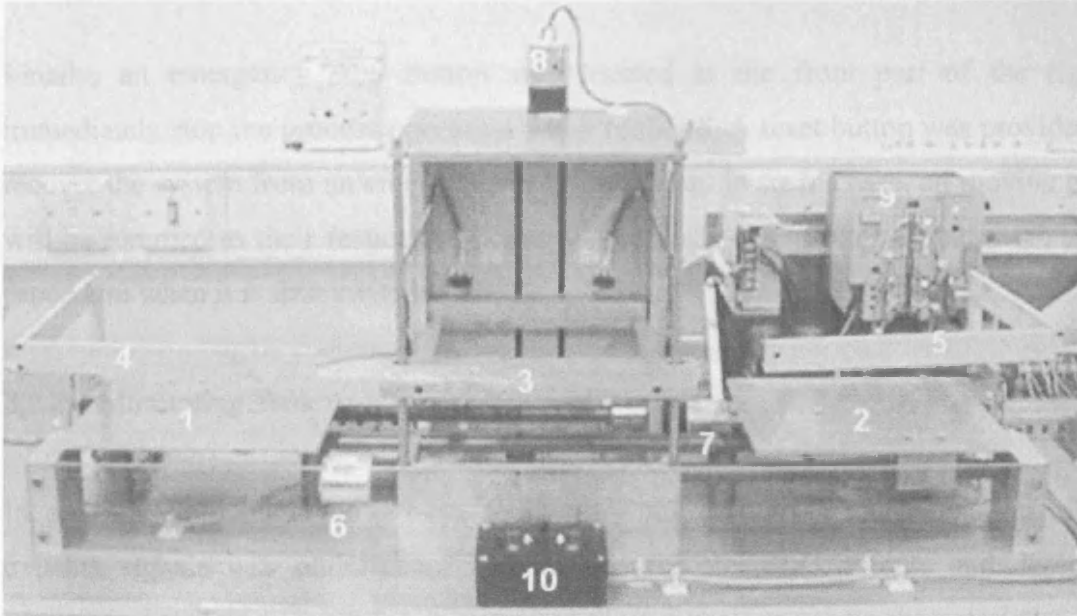
This chapter presents a monitoring application to demonstrate the effectiveness of the Petri-net approach described previously. A laboratory rig that mimics the basic operations of an industrial press was monitored in order to provide online process information and to demonstrate how the Petri-net monitoring system can be used to help identify the source of simulated faults. The demonstration further illustrates how results are captured and processed, and how the management information that they enable can be displayed using a web page based approach developed in conjunction with the existing methodologies that operate within the IPMM Centre.

8.2 – Process Overview

The Press Rig, shown in Figure 8.1, is a scale module based on an ASEA industrial hydro forming press. The Rig has two pallets, left and right, where, on the industrial machine, work pieces can be loaded. Load/unload operations take place at a designated “home position” at the outside end of the travel for each pallet. Each side is provided with a safety guard, in order to ensure that the work piece cannot be handled once the process was started.

A work piece loaded on either pallet is transported toward the central part of the machine, where the pressing operation will be performed. Once any of the horizontally moving pallets reaches the pressing position, the central vertical axis can be moved down. At this point, in the real process, a high-pressure operation would be performed, in order to mould the work piece. This operation was not mimicked directly in the rig, but was simulated by a short delay (20 s). At the end of the pressing operation the work piece will be transported back to the home position to be unloaded.

Whilst the first pallet is still moving, once it has left the central area of the rig, the opposite pallet, if loaded, can start to move toward the centre for the next operation sequence.



- | | |
|--------------------------------|--------------------------------|
| 1 – Left hand pallet | 6 – Right hand pallet DC motor |
| 2 – Right hand pallet | 7 – Left hand pallet DC motor |
| 3 – Central pressing structure | 8 – Central Structure DC motor |
| 4 – Left hand guard | 9 – Siemens S5-95U PLC |
| 5 – Right hand guard | 10 – Operation panel |

Figure 8.1: Press rig and main component parts.

8.2.1 – Press Rig Component Description

The Press Rig operation is controlled by a Siemens S5-95U PLC. Proximity sensors have been located to detect the pallets “home” and “central” positions. Limit switches are employed to detect the vertical axis “up” and “down” positions. Limit switches were also deployed at strategic positions for safety purposes, stopping out-of-range movements (potentially arising from any main sensor fault). Each of the safety guards controlling the access to the pallets was provided with a limit switch, to provide a means of sensing all open/closed conditions.

Three independent DC motors provide the mechanical movement of the pallets and the vertical axis, by means of lead screws. Movement reversal is achieved by inverting the motors polarity. Mechanically operated switches were inserted in the motors' supply lines, thus enabling the simulation of fault conditions.

Finally, an emergency stop button was located at the front part of the rig to immediately stop the process operation when required. A reset button was provided to recover the system from an emergency stop condition. In such a case, all moving parts will be returned to their respective home positions. The controller executes the same procedure when it is first switched on.

8.2.2 – Monitoring Task Analysis

In order to implement a Petri-net to monitor the process, an investigation of the existing signals was undertaken. This considered the PLC outputs and deployed sensors and switches. Table 8.1 summarises these signals and their characteristics.

Table 8.1: Press rig signals description.

	Description	Signal Source	State
1	Emergency Stop	Mechanical switch	On - 24 V
2	Reset	Push button	On - 24 V
3	Right hand cycle start	Push button	On - 24 V
4	Left hand cycle start	Push button	On - 24 V
5	Central positioning – right hand pallet	Proximity sensor	On - 24 V
6	Central positioning – left hand pallet	Proximity sensor	On - 24 V
7	Home position – right hand pallet	Proximity sensor	On - 24 V
8	Safety – right home position	Limit switch	On - 24 V
9	Home position – left hand pallet	Proximity sensor	On - 24 V
10	Safety – left home position	Limit switch	On - 24 V
11	Up position – central structure	Limit switch	On - 24 V
12	Safety – central up position	Limit switch	On - 24 V
13	Down position – central structure	Limit switch	On - 24 V
14	Safety – central down position	Limit switch	On - 24 V
15	Right hand pallet guard	Limit switch	On - 24 V
16	Left hand pallet guard	Limit switch	On - 24V
17	Motor command – central up	PLC output	On - 24 V
18	Motor command – central down	PLC output	On - 24 V
19	Motor command – right pallet to centre	PLC output	On - 24 V
20	Motor command – right pallet to home	PLC output	On - 24 V
21	Motor command – left pallet to centre	PLC output	On - 24 V
22	Motor command – left pallet to home	PLC output	On - 24 V

The task facing the monitoring system was to follow the sequential operation of the process in order to both provide enough data to yield production information and allow the identification of any process faults. The signals summarised in Table 8.1 provided the information source that was used to monitor process events and states. Accessing and processing this data then enables the system to calculate parameters such as “in process” and “stoppage” times, thus illustrating the benefits of the Petri-net implementation.

The DC motors were considered to be critical parts of the Rig that needed to be monitored during process operation. It was assumed that the condition of each motor could be determined from the motor current demands. Table 8.2 shows the motor specification. A simple circuit was produced to provide the motors parameters measurement to the Monitoring Module (Figure 8.2).

Table 8.2: DC motors specification [8.1].

Item Description	Specification
DC Motor RS 330-799	<ul style="list-style-type: none"> • 12 Vdc • 493 mA (maximum) • 40 rpm

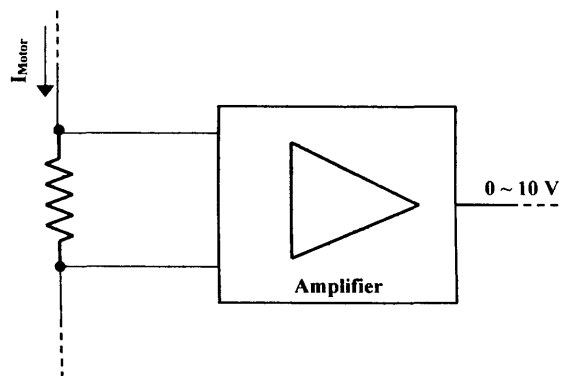


Figure 8.2: Block diagram of signal conditioning circuit.

8.3 – Press Rig Petri-net

In order to design the Petri-net to monitor the process, signals had to be selected and assigned to the Monitoring Module. At the same time, places representing special features (analogue acquisition and process states status) had also to be identified. Table 8.3 shows such details. It also shows that the timeout feature was limited to places up to number 200, reserving those above this number for modelling purposes that do not require this feature. This number, as described in Chapter 7, can be dynamically assigned by the Petri-net designer and may assume different values for different applications.

Table 8.3: Press Rig Petri-net definitions.

Representation	Description
DS 01	Emergency stop switch
DS 02	Vertical axis down motor command
DS 03	Right pallet to centre motor command
DS 04	Left pallet to centre motor command
DS 05	Vertical axis up motor command
DS 06	Right pallet to home motor command
DS 07	Left pallet to home motor command
DS 08	Reset command button
DS 09	Right guard switch
DS 10	Left guard switch
AN 1	Analogue 1 – vertical axis motor current
AN 2	Analogue 2 – right pallet motor current
AN 3	Analogue 3 – left pallet motor current
T004	Emergency stop event
T025	Vertical axis motor over current
T026	Vertical axis motor under current
T036	Right pallet motor over current
T038	Right pallet motor under current
T044	Left pallet motor over current
T046	Left pallet under current
P001	Left process cycle watch
P002	Right cycle process watch
P003	Left guard operation watch
P004	Right guard operation watch
P023	Analogue 2 acquisition trigger – right pallet to centre
P024	Analogue 3 acquisition trigger – left pallet to centre
P025	Analogue 1 acquisition trigger – down command
P026	Analogue 1 acquisition trigger – up command
P027	Analogue 2 acquisition trigger – right pallet to home
P028	Analogue 3 acquisition trigger – left pallet to home
P200	Last timeout enabled place

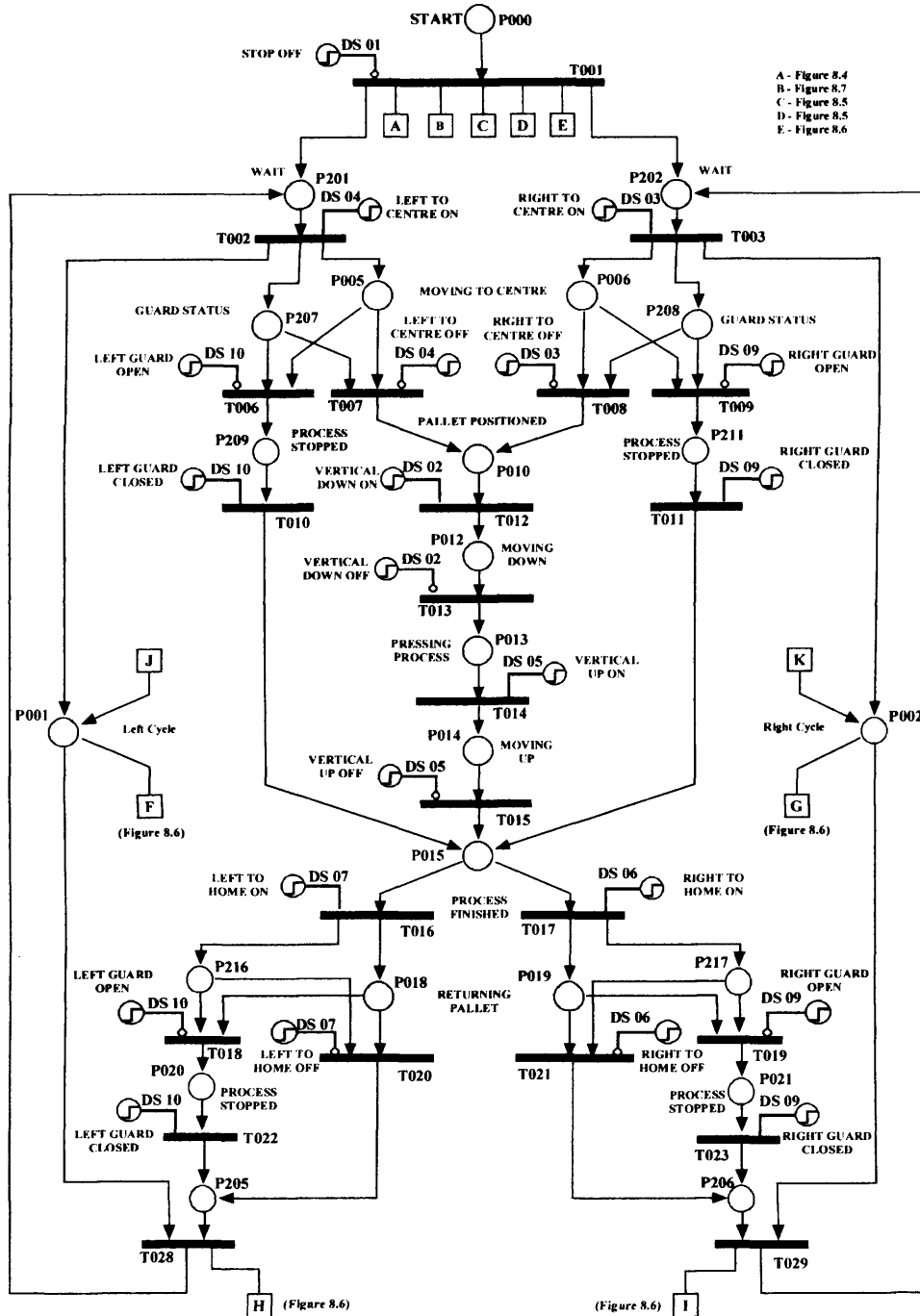


Figure 8.3: Press rig operation Petri-net.

The process Petri-net design was divided into blocks, each considering specific monitoring requirements. The main block, shown in Figure 8.3, follows the general operation of the process. Describing this part of the Petri-net, transition T001 initialises several components, since “emergency stop” has not been activated. Places P201 and P202 represent the “ready state” of each process pallet. These places provide input conditions that enable, respectively, the “left” and “right” parts of the

process. These events are defined by transitions T002 (left) and T003 (right) and will be triggered by the motors' commands (DS 04 – left and DS 03 – right). The firing of these transitions will result in a “moving to centre” state, identified by tokens in places P005 and P006. The PLC was programmed to restart process operation if the guard is opened while the respective pallet is moving. Therefore, the branch formed by P207 (P208), T006 (T009), P209 (P211) and T010 (T011) was required to enable initial conditions to be restored in such case. The end of the pallet movement will fire transition T007 (left) or T008 (right) and put a token in P010, indicating that the pallet was positioned.

At this point, a single branch represents the operation of the vertical axis and press forming operation, between transitions T012 and T015. This is finished when a token reaches P015. The respective pallet will return to its home position, which will be followed by T020 (left) and T021 (right). Again, the guards will be monitored by T018 / T022 (left) and T019 / T023 (right).

Places P001 and P002 were included in order to enable each pallet processing cycle time to be monitored. To do so, these places were defined in the “process states” table, thus providing records identifying the beginning and end of these states.

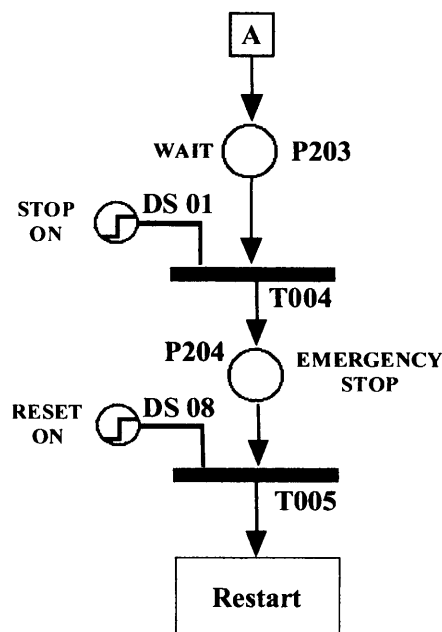


Figure 8.4: Emergency stop monitoring branch.

The action of pressing the “emergency stop” was implemented in a Petri-net branch shown in Figure 8.4. Transition T004 firing will result in a “fired transition record”. The “reset” command will fire T005 and “restart” the Petri-net execution.

Since considering that loading/unloading operation time might be used as an efficiency measurement in a real process, the operation of the pallets’ guards will be monitored by a specific Petri-net branch, shown in Figure 8.5. Places P003 and P004 will provide records notifying the beginning and ending of these operations.

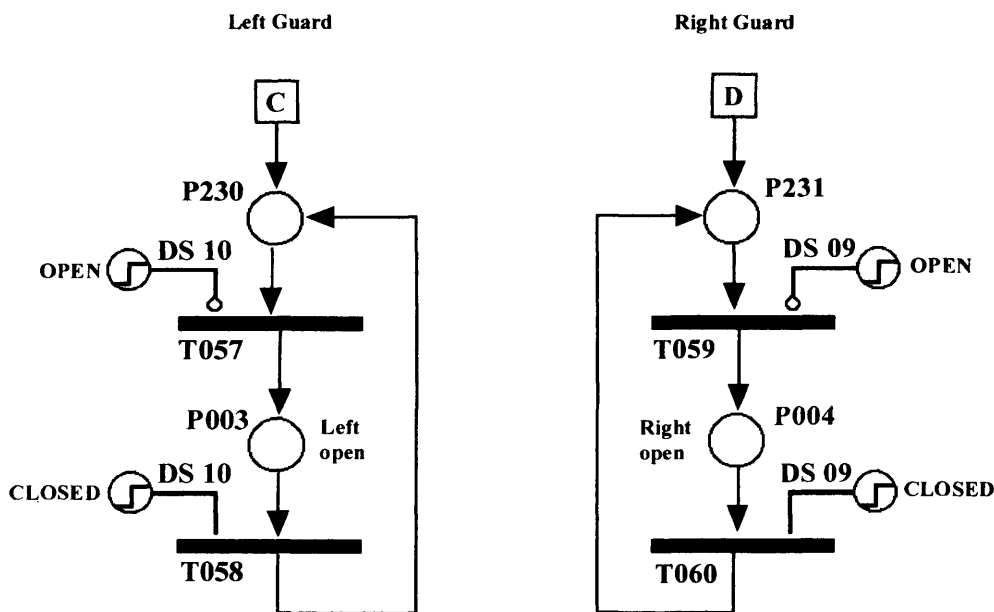


Figure 8.5: Guard operation monitoring branch.

Considering that a measure of process efficiency might be required, one of the Petri-net’s implemented branches was set up to monitor “processing time”, “available time” and “stopped time”. It is shown in Figure 8.6. A “process-stopped” condition was associated with the Rig stop switch, resulting in a token in place P008. The “processing” status will be provided by P001 (left cycle) and P002 (right cycle) and will result in a token in P009. The absence of any of the previous conditions will result in a token in P007, indicating “process availability”. Such parameters might be employed as a basis for the calculation of process assessment indices, such as Overall Equipment Efficiency (OEE) [8.2].

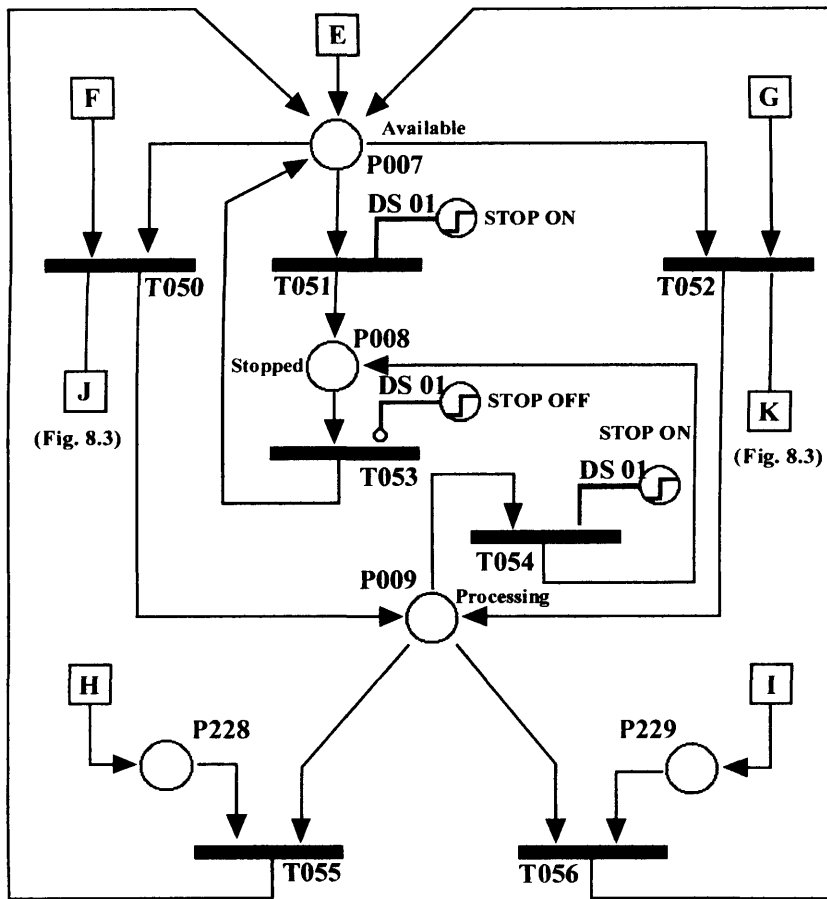


Figure 8.6: Equipment usage monitoring.

Individual Petri-net branches monitored the 3 DC motors. Figure 8.7 shows one of these branches, associated with the vertical axis motor. Two “analogue transitions” were introduced to monitor extreme operating conditions. By doing this T025 and T026 were able to transmit a “transition record” associated with these conditions. To avoid false alarms, due to initial transients during the motor switching-on, a delay was introduced by means of T024. The average motor current during each operation will be “watched” by P025 (down) and P026 (up), which will trigger the acquisition of analogue input 1. Similar structures were implemented for the right and left pallet motors. In these cases, based on Figure 8.7, T044 (left) / T036 (right) will play equivalence to T025 and T046 (left) / T038 (right) to T026. Places P024 / P028 will trigger analogue input 3 acquisition (left motor current) and P023 / P027 analogue input 2 (right motor current). Monitoring such parameters during process operation might help to identify critical or degradation conditions of these specific components.

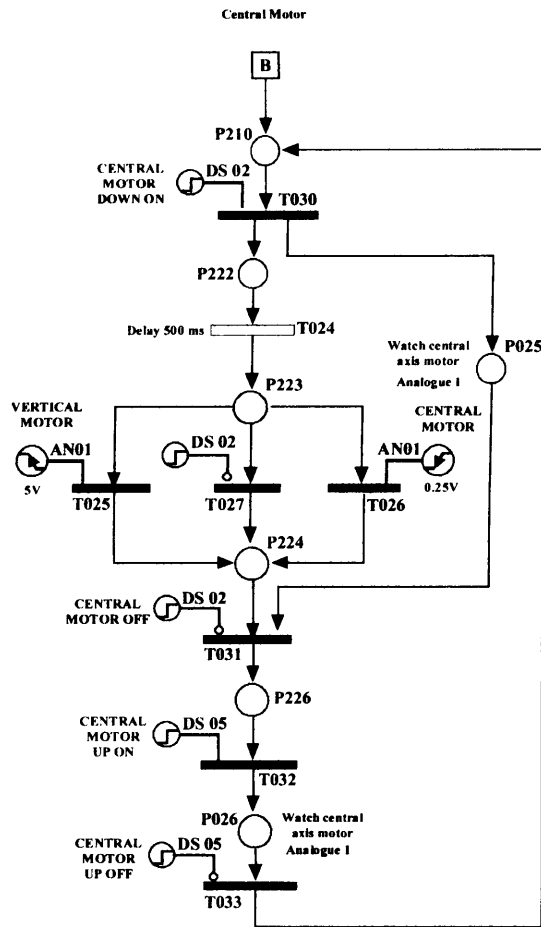


Figure 8.7: DC motor monitoring net.

The Petri-net implementation required 56 places and 61 transitions. Ten process signals were interfaced, requiring 2 digital input cards. Also, 3 analogue inputs were used to monitor the motors current. The text files that describe this Petri-net implementation tables can be found listed in Appendix B (B.1, B.11 and B.12). Some results of the implementation will be presented in the following section.

8.4 – Monitoring Results

A structure composed of one Monitoring Module (with 2 digital input cards), a Connectivity Module and a Management Application was deployed to monitor the Press Rig operation. The Management Application was executed on a desktop computer. The monitoring records were stored in a database based on a Microsoft

SQL 2000 Server ® DBMS, hosted by the IPMM Centre server. The same hardware also hosts the Centre Internet’s server, which is based on Microsoft’s Internet Information Server®.

8.4.1 – Data presentation Approach

To present the data and information retrieved from the monitoring records, the IPMM Centre web based presentation approach [8.3] was employed. In simple terms, the Centre web page provides a collection of hyperlinks, each one connecting to a specific presentation task. The task is defined in terms of the display format (graphs or data table) and an SQL statement to select and assemble the data in a way that can provide meaningful information. The web page is periodically refreshed. Examples of such web pages are referenced in Appendix B (B.18 [11]).

The embedded tools of the database system (triggers and procedures) [8.4] were employed to further process the monitoring data, whenever required. Such processing represents a simple and direct way to extract information or join the data in new tables, therefore enhancing the web based presentation method.

PressRigStates

DESCRIPTION
Processing right cycle
Left guard open

Figure 8.8: Process online status information example.

8.4.2 – Monitoring Presentation Examples

It was stated in Section 8.3 above that places P001, P002, P003 and P004 were defined in order to provide process status information, based on beginning / ending records. Figure 8.8 shows an example of such status information displayed on the web page. In the example shown it is the right cycle that is being processed, while at the

same time the left guard is kept open. An improved use of such a feature could be a graphic display of a process, with online updates of the machine / process operation.

In another example, shown in Figure 8.9, the latest process operation records were displayed. Each record is provided with a description, timestamp and when appropriate, an additional parameter (e.g. timeout record). This information is retrieved from the system's Event Table (*prefix_EV*), with the description provided in an externally defined database table. The events were sorted and associated with their description by a database procedure that is called each time the web page is refreshed. The example shows a transition (T004) that was fired when the rig's "emergency stop" was pressed; a fired transition (T025) due to an over current condition of the vertical axis motor (Figure 8.7); and 4 timeout events (P001, P006, P018 and P005). Timeout events provide further information in the "EventRecord" field, indicating the parameter (in seconds) associated to the event.

PressEvents

Description	TimeRecord	EventRecord
T004 - Emergency stop	2003-08-21T10:26:42	
P001 Timeout - left cycle	2003-08-21T10:26:09	122
T025 - Central motor over curren	2003-08-21T10:24:56	
P006 Timeout - Right pallet to ce	2003-08-14T21:49:35	43
P018 Timeout - Left pallet to ho	2003-08-14T21:27:59	45
P005 Timeout - Left pallet to cer	2003-04-17T15:46:01	45

Figure 8.9: Process list of latest events.

Places P001 and P002 records were again used to present process productivity information, shown in Figure 8.10. In this example a bar graph was employed to show the number of cycles processed using the left (P001) and right (P002) pallets. The system's States Database Table (*prefix_ST*) was queried to retrieve this information, directly manipulated by the web page defined SQL statement.

PIC-PressRig-Cycles

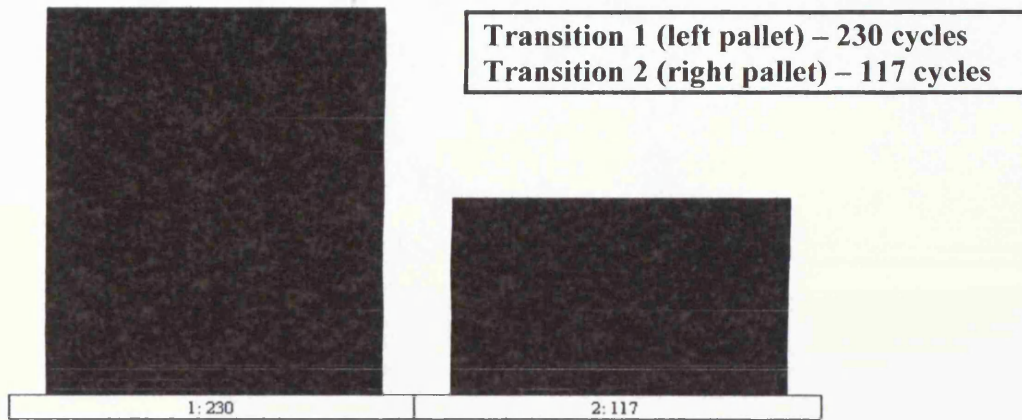


Figure 8.10: Number of cycle processed by the left (1) and right (2) hand pallets.

PIC-PressRig-Usage

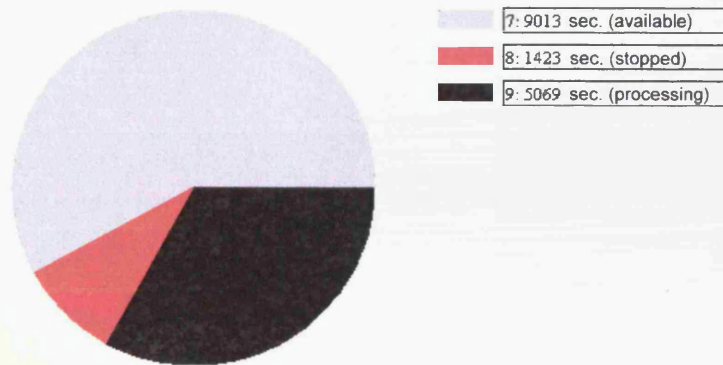


Figure 8.11: Press rig online operating time (s) information – available (7), stopped (8) and processing (9).

Using a similar approach, further productivity information was presented (Figure 8.11) based on monitored data provided by places P007 (process available), P008 (process stopped) and P009 (processing) active states recorded by the system’s Petri-net branch previously described (Figure 8.6). In Figure 8.11, besides the graphical

representation, absolute values in terms of the time (seconds) associated with each state are shown by the place number (7, 8 and 9). In a real manufacturing application such information could be integrated within other systems (i.e. production management systems) and thus further process assessment information might be provided (e.g. OEE).

A fault was introduced during the operation of the Press Rig, by interrupting the current supply of the central axis motor, while in the “pressing process” state (Figure 8.3, P013). Since executing a left hand cycle, such fault condition resulted in a timeout event associated to place P001 (Figure 8.9). In order to illustrate the fault diagnostics implementation, an analysis request having P001 as a symptom was sent to the Monitoring Module. The result of the diagnostics performed is shown in Figure 8.12. It indicates that three transitions might be related to the fault source: T054, T003 and T014.

PressRigFault

Event:Description	SourceDescription
54-Stopped	Emergency stop
3-Right process	Right to centre cmd
14-Moving up	Up command

Figure 8.12: Result of a fault analysis, based on a P001 timeout event.

Analysing this result, P001 is the input place of T028 (Figure 8.3) and T050 (Figure 8.6). Clearly, the timeout event occurred due to T028 failing to fire. However, T050 was taken into consideration, with the Petri-net analysis tracking P007, T053, P008 and finally reaching T054, which was enabled by P009, but missing DS01. With respect to T028, the analysis performed by the Monitoring System followed the way back from P205, finding two transitions, T003 and T014, enabled by their respective input places (P013 and P202), but missing the process signals linked to them (DS03 and DS05). Since the left cycle was executing, T003 / DS03 could not be the source of the fault. This result was due to P015, common to both cycles. Such information provided by the system could then be employed by operators and experts.

The result obtained shows the system capability in isolating an operational fault based on an initial symptom. It must be considered that at the same time the diagnostics analysis was performed, there were other places with tokens (P203, P230 and P231). Nevertheless, the transitions enabled by them were not listed in the fault diagnostics result.

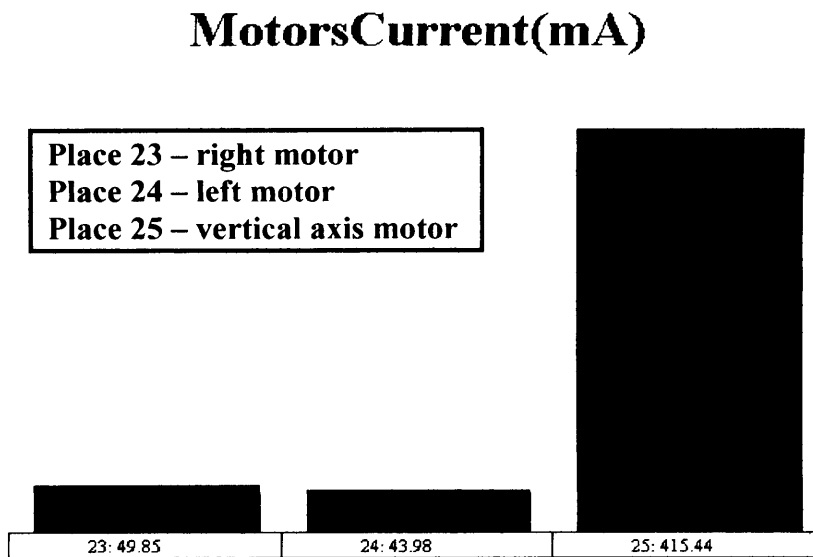


Figure 8.13: DC motors current measurement, using Petri-net places acquisition trigger approach – right (23), left (24) and vertical axis (25) motors.

In a last example, Figure 8.13 shows the behaviour of the 3 DC motors in terms of average current (mA). Such parameters were provided by using the Petri-net places triggering feature to undertake the motors' current acquisition under operating conditions (Section 8.3, Figure 8.7). Place P023 was associated with the right pallet, P024 with the left and P025 with the vertical axis movement. The results presented in Figure 8.13 show that the vertical axis motor operates at a higher current condition (415.44 mA), that is close to the specification limit (493 mA). This result is in tune with the event recorded by the analogue transition T025 (Figure 8.9), which previously recorded a peak over current condition. In this case the fact is that this motor is “working harder” than the other two, since it is lifting the bed. The bar graph was built considering the average of several measurements recorded in the database. For simplification reasons, only one movement direction was considered for each case.

It is possible to see how the system can be used to monitor changes in behaviour from “normal” to “abnormal”. There might be two approaches to consider here. In the first (and simplest) one, the event produced by the “analogue transition” (T025) could be used to produce an alarm, either locally by means of an “output transition” or by detecting such critical event when recorded in the database. From the database it could then be used by another application to request further actions (maintenance). As a simple instance, specific users might be notified by means of e-mails about such critical events. The second approach could be to follow the motors current, based on the records provided each time these devices are operated, thus making it possible to establish preventive actions based on the degradation of the motors condition.

8.5 – Summary

In this chapter it was explored an example that enabled to illustrate the use of the System to monitor the Press Rig. A dynamic illustration of this example is referenced in Appendix B (B.18 [12]). Besides the use of basic modelling structures, it was shown the employment of analogue and delay transitions. Specific places were selected to provide the usage of the Rig and to monitor the motors current. The example also allowed to demonstrate the fault diagnostics approach, used to isolate a fault whose symptom was provided by a timeout event.

It was shown the wide range of different information that can be obtained from the records provided by the Monitoring System. By making these records available on an open and easy to interface platform such as a database system, different applications might use such data in order to improve management capabilities. The easy way in which the information can be made available on the Internet opens opportunities for process and machine remote monitoring. In the next chapter another example is presented, further focusing on productivity data and exploring some other aspects of the system that were not used in this first case.

REFERENCES

- 8.1 RS Components Ltd, RS Electrical Products, RS Web Site, Available from: <http://rswww.com> [Accessed 14 August 2003].
- 8.2 **Willmott, P.** Total Productive Maintenance: The Western Way, Oxford, Butterworth-Heinemann, 1994.
- 8.3 **Kerrien, B.** To Develop User Configurable, Dynamic Web Pages for Presenting Real-time Process and machine Condition Information for manufacturing Processes. Academic Report, Cardiff University, August 2002.
- 8.4 **Vieira, R.** SQL Server 2000 Programming. Birmingham, UK: Wrox Press Ltd, 2000.

CHAPTER 9

CONVEYOR RIG MONITORING TASK

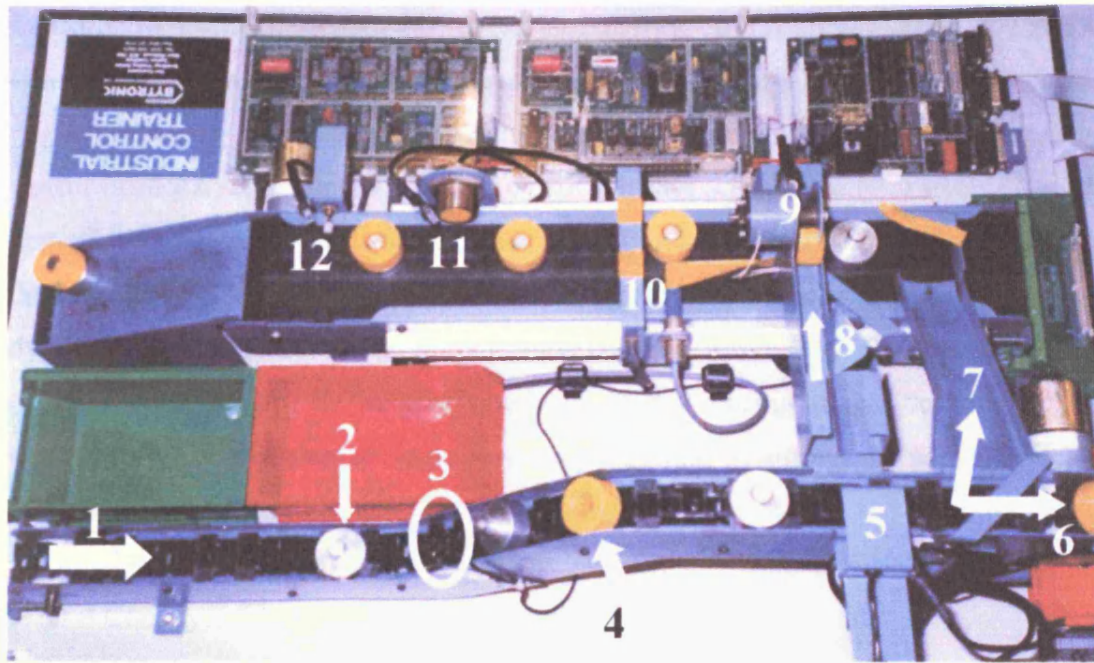
9.1 – Introduction

One of the purposes of a monitoring system is to support modern maintenance practices, by helping to establish machine performance parameters. Continuous flow manufacturing systems have to consider the overall efficiency of the process, where best results are achieved when all components operate at optimal conditions. This chapter presents the use of the Petri-net based implementation to monitor and manage such manufacturing systems using as an example process a laboratory rig, called the “Conveyor Rig”. The work is especially focused on providing information to analyse the performance of the process. Although not representing a real manufacturing application, this work demonstrates the capabilities of the developments previously outlined in this thesis.

9.2 – Process Description

The Conveyor Rig is an “Industrial Control Trainer” developed by Bytronic International Ltd. It emulates a continuous flow manufacturing system. The basic operation assembles components that are individually fed into the system and assesses the quality of the end product in order to detect and reject parts that do not comply with the established standard. In summary, the process operation, taking the notations used in Figure 9.1, can be described as: metal pegs (2) and plastic rings (4) are placed on the system’s input chain (1) and sequentially detected by a sensor (3). When arriving at the “sort area” (5), the components are classified: plastic rings will be forced into a “ring queue” (8), whereas metallic pegs continue on the chain, reaching the “slide” (7) that leads them towards the output conveyor. It is possible for the ring queue to become full (the maximum number of rings is 6). In this case, additional plastic rings arriving at the sort area will not be diverted into the queue by the

solenoid-sorting device but will remain on the chain and be rejected into the plastic rings “reprocess” area (6).



- | | |
|--|--------------------------------------|
| 1 – Input / loading | 7 – Sorted pegs path |
| 2 – Metallic peg | 8 – Plastic rings queuing |
| 3 – Input sensor | 9 – Assembly area |
| 4 – Plastic ring | 10 – Metal sensing (output conveyor) |
| 5 – Sorting area (metal/plastic sensing) | 11 – Plastic assembled sensing |
| 6 – Plastic rings reprocess | 12 – Reject area |

Figure 9.1: Conveyor Rig photo with main details.

Rings and pegs are assembled to form a single part in the “assembly area” (9), on the lower conveyor. An actuator, in the form of a “rotate gate”, allows one ring each time to roll into the assembly position. A peg on the output conveyor moving through the assembly area will enter into the ring and drag it out of its pre-assembly location, forming a single mounted unit.

The operation so far described should result in a good assembly, with a ring mounted on top of a peg. It is possible however (for several reasons) that the assembly will be a bad one, with rings and/or pegs moving unassembled on the lower conveyor. A faulty

assembly operation may be due to a missing ring in the assembly position (empty queue) or to the ring dropping off from the peg or keeling over in the assembly area.

In order to check the assembly operation, sensors are deployed along the output conveyor. At the “metal checking area” (10), passing parts are sensed to detect the peg component. At the next “assembly quality” checking point (11), a capacitive sensor is employed to detect whether or not the ring was correctly assembled on the peg. Finally, at a “reject station”, near the output (12) a sensor and actuator are deployed in order to provide the means to reject faulty parts. Due to the process dynamics, components on the output conveyor may move very close to each other (without enough gap) or be positioned out of the sensing range, “confusing” the system’s PLC programmed logic (as supplied) and resulting, in some cases, in wrongly rejected parts.

9.2.1 – Conveyor Rig Component Description

The Conveyor Rig operation is controlled by a Siemens S5-95U PLC. Reflective sensors are deployed in areas where a general part detecting method is required (Figure 9.1 – 3, 5, 6, 9, 11 and 12). Metal detection is based on proximity (inductive) sensors (Figure 9.1 – 5 and 10). Other sensors employed are a pass-through beam (10) for moving parts and a capacitive sensor (11) to sense mounted rings. Solenoid based actuators are used in areas where components are required to be moved or removed (5, 9 and 12). Two separate DC motors provide the input chain and output conveyor movement.

The equipment manufacturer uses specific electronics to interface sensors and actuators to the PLC’s inputs/outputs. The DC motor drivers were also provided by separate electronics. Mechanical switches are employed to start/stop these motors.

9.2.2 – Monitoring Task Analysis

The main purpose in monitoring the Conveyor Rig was to obtain information that would enable productivity and efficiency analysis. It was therefore required to design a Petri-net capable of providing the number of parts produced and those rejected or

reprocessed. At the same time, the actual use of the assembly system was considered a requirement in order to establish efficiency parameters.

The Petri-net implementation considered the existing process signals, which are summarised in Table 9.1. However, two signals considered of importance to measure the process usage had to be added. A “process stopped” (unexpected stop) condition was obtained from the DC motors supply line. To implement this a low-pass filter interface was necessary. The second condition, a “programmed stop”, was implemented as a sub-net input using the Microchip CAN MCP2510 Development Kit module to issue this command. In reality another digital input could have been used for this purpose. However, it was deemed to be a good opportunity to illustrate the use of a sub-net input in the Petri-net design. The MCP2510 development kit enables the assembly of the sub-net message and its transmission on the CAN bus, thus emulating a Monitoring Module function. In this case, it is used to enable the recording of an actual unexpected stop as a “programmed stop”.

Table 9.1: Conveyor Rig signals description.

	Description	Signal Source	State
1	Motor command	Mechanical switch	On - 12 V
2	Part on chain (input)	Reflective sensor	On - 24 V
3	Part in sort area	Reflective sensor	On - 24 V
4	Metal in sort area	Inductive sensor	On - 24 V
5	Ring queuing actuator – sort area	Solenoid actuator	On - 24 V
6	Ring reprocess detect	Reflective sensor	On - 24 V
7	Ring in assembly area	Reflective sensor	On - 24 V
8	Ring loading actuator – assembly area	Solenoid actuator	On - 24 V
9	Metal sensing on belt	Inductive sensor	On - 24 V
10	Moving part at metal sensing area	Pass through beam	On - 24 V
11	Assembled part sensing	Capacitive sensor	On - 24 V
12	Part at capacitive sensing area	Reflective sensor	On - 24 V
13	Part at reject area	Reflective sensor	On - 24 V
14	Part reject actuator (output)	Solenoid actuator	On - 24 V

9.3 – Conveyor Rig Petri-net

Table 9.2 shows the Rig signals assigned to the Monitoring Module and their associated Petri-net definitions. This information was required to achieve the desired

level of detail to meet the monitoring aims. Production outputs are classified by the manufacturing system (Conveyor Rig). This classification is associated to actions (accept or reject parts and reprocess rings). Such actions can be characterised as events (good detected, bad rejected) and therefore naturally represented as transitions in the Petri-net design. Transitions were also used to represent events resulting from the manufacturing system making wrong decisions (wrongly rejected or accepted).

Table 9.2: Conveyor Rig Petri-net definitions.

Representation	Description
DS 01	Object on chain sensor
DS 02	Part sensor - sort area
DS 03	Metal sensor - sort area
DS 04	Rotate gate actuator – ring to assembly position
DS 05	Plastic ring reprocess sensor
DS 06	Ring in assembly position sensor
DS 07	Inductive sensor - metal on belt
DS 08	Beam sensor – part moving on belt
DS 09	Part sensor - capacitive sensing area
DS 10	Capacitive sensor
DS 11	Part sensor – reject area
DS 12	Reject actuator
DS 13	Chain DC motor command
T009	Plastic ring reprocess detect
T028	Rejected part
T029	Failed to reject a out of standard part
T030	Good part wrongly rejected
T031	Good part assembled and accepted
T035	Excessive number of continuous good parts rejected
P028	Process stopped – motors command off
P029	System processing – assembly in operation
P030	Process available – operating but not assembling
P031	Programmed stop – external command
P200	Last timed out place

The process usage measurement was based on the time consumed by each of the operating states (available, processing/assembling, stopped and programmed stop). To provide such parameters, Petri-net specific places were selected and associated to these states and configured to provide status information (beginning / ending records). The timeout feature was disabled for those places with an identification number higher than 200. Similarly as stated in Chapter 8, this value was selected simply to provide a barrier to isolate places enabled to timeout from those that are not.

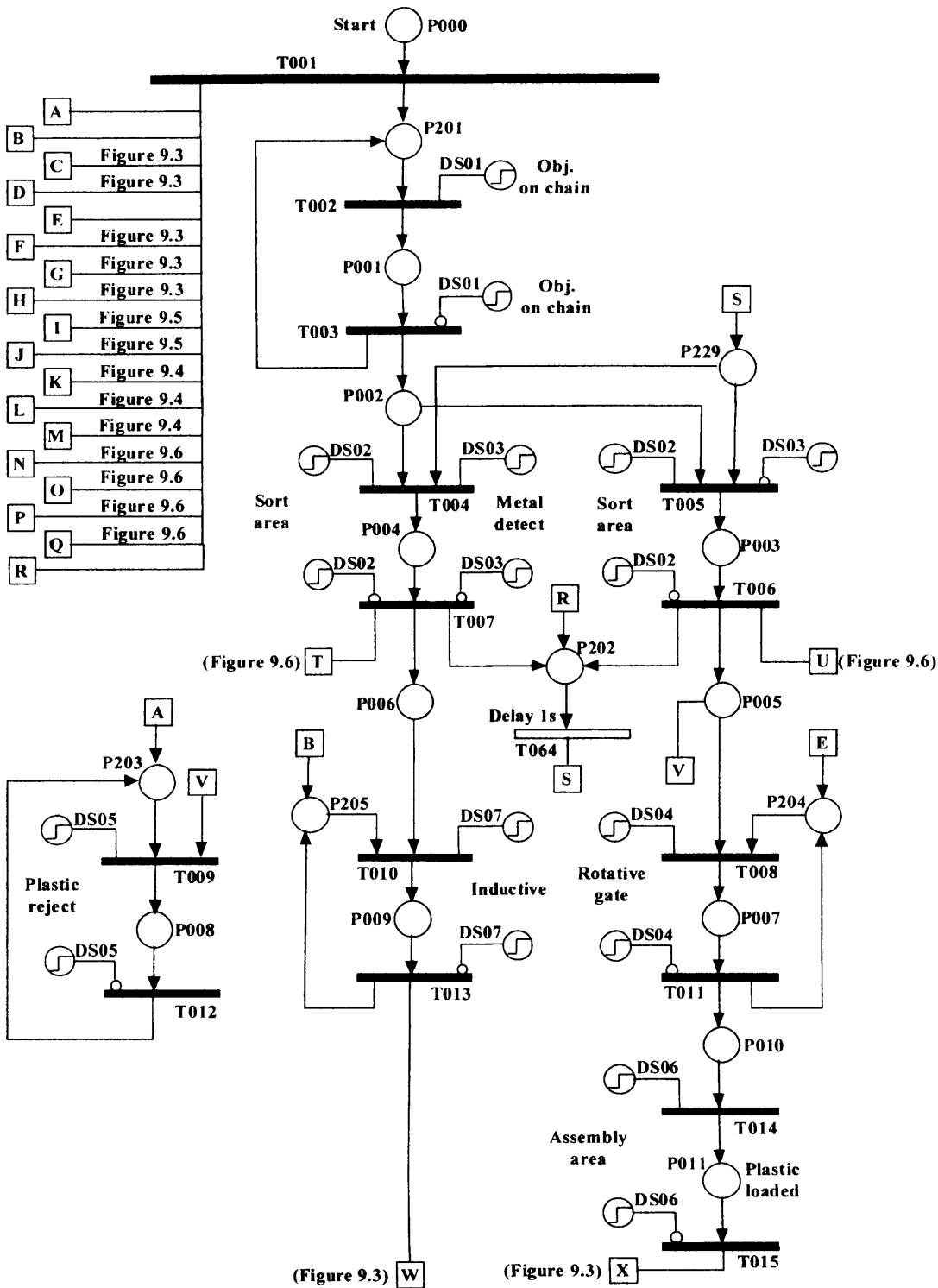


Figure 9.2: Rig operation Petri-net – load and sorting.

The monitoring task was divided into blocks accordingly the part/function of the process by them monitored. This sort of flexibility provided by the model eases considerably the development of monitoring applications. The Petri-net representing

the Rig operation was split into two in order to simplify its description, as shown in Figure 9.2 and Figure 9.3. The first of these figures shows the Petri-net designed for the monitoring of the feeding and sorting of pegs and rings. Figure 9.3 thus concentrates on monitoring the assembly and output checking actions.

In Figure 9.2, transition T001 is initially fired, enabling the initiation of several structures in the net. A token in place P201 enables the Petri-net to begin to monitor the components loaded on the input chain. The on/off states of DS01 result in tokens in P002, indicating a part moving toward the sorting area, while also re-enabling T002 for the next loaded component. When a component reaches the sorting area, T004 will detect it as a peg, whereas plastic rings will be detected by T005. The firing of either T004 or T005 will remove the token from P229, therefore avoiding multiple firing of these transitions, due to components “queued” in P002. Transitions T004 and T005 will only be enabled again after the component in the sorting area has left. A further delay of 1 s was introduced by means of a “delay transition” (T064), in order to avoid assembly components mismatch, thus insuring that the former part has already left the sorting area before checking the next one.

The information relating to component queues will be held by P005 (rings) and P006 (pegs). The existence of tokens in these places will be further used to identify the process operating states (T and U), which will be considered later. Reprocessed rings will be monitored by the net represented by elements P203, T009, P008 and T012. In terms of the Petri-net execution, a token will be removed from the ring queue (P005) each time T009 is enabled by DS05 (reprocess). Rings placed in the assembly position will fire T008, thus removing a token from the rings queue (P005). A ring in the assembly position will be indicated by a token in P011 (T014 fired). The use of this ring will be indicated by T015 being fired and P012 (Figure 9.3) receiving a token. Tokens in P006 (pegs) will be removed, one each time, when the inductive sensor (DS07) senses a peg moving on the output conveyor, thus firing T010. This results in another token in P012 (Figure 9.3).

Figure 9.3 shows the Petri-net elements that monitor the assessment of the parts flowing on the output conveyor. In order to ease the monitoring task, due to faulty events at the assembly area (Section 9.2), pegs and rings will result in tokens in a

common place (P012). A part arriving at the capacitive sensing area will fire T016, removing a token from P012. A second token may be removed by T017, if enabled by the capacitive sensor (DS10). A correctly assembled part will produce a token in P016 and another in P017.

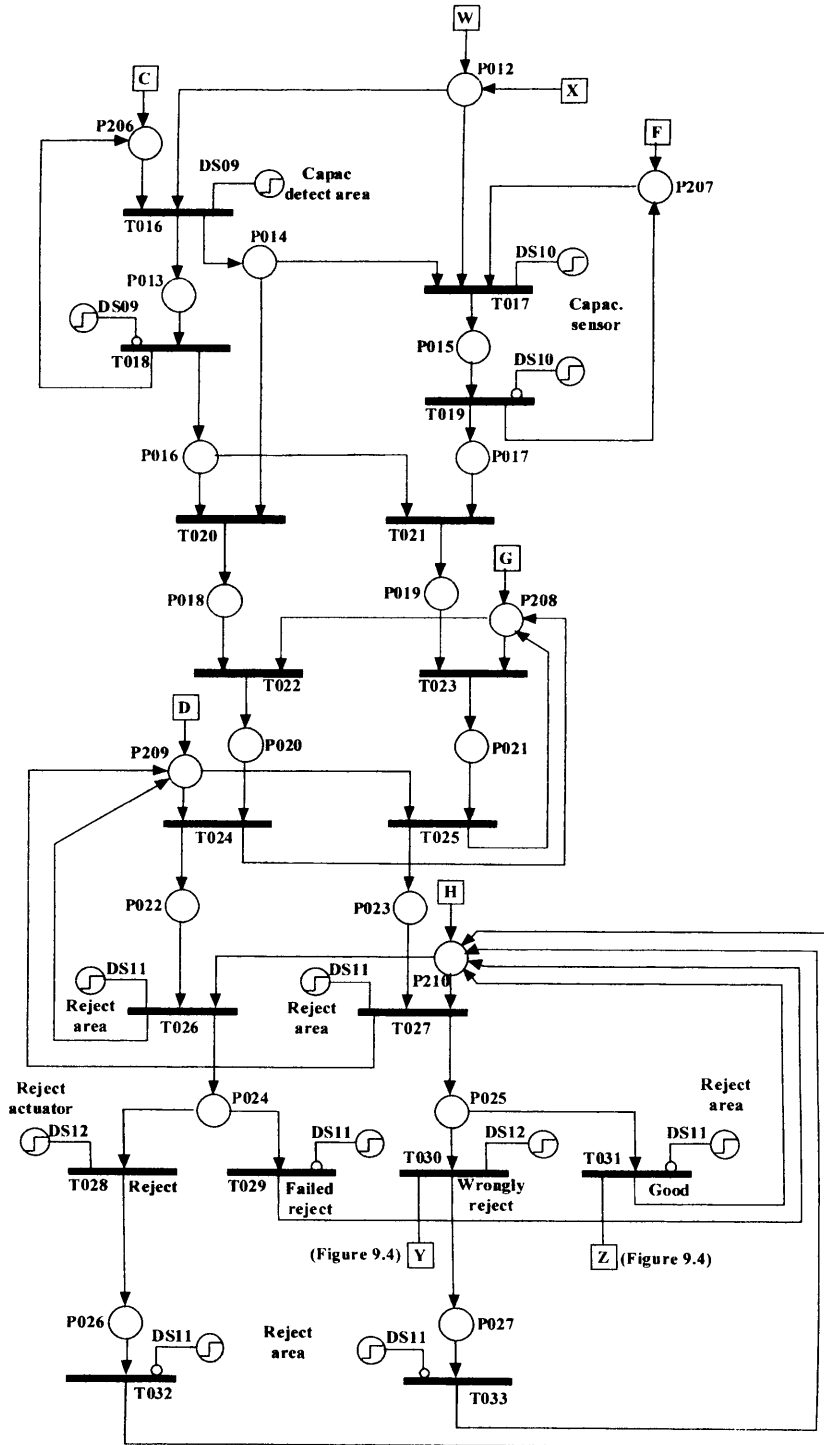


Figure 9.3: Rig operation Petri-net – assembly and checking.

Considering that a token in P014 will be removed by T017 firing, a “good” part results in T021 being fired, while a “faulty” part will fire T020. Transitions T022/T023 and T024/T025 were required to implement an “output queue”. The firing of T026 indicates a faulty part arriving at the reject area, thus providing two hypotheses: refusing the part (T028) or failing to refuse (T029). A well-assembled part will imply in T027 being fired and again leading to two hypotheses: correctly accepting a good part (T031) or wrongly rejecting it (T030).

This relatively simple assembly process is revealed by the above analysis to in fact be rather complicated. Conventional control systems, such as the PLC used to manage this activity, struggle to cope with the variations possible. One of the significant benefits of the monitoring approach developed by this research is the ability of the deployed system to be able to accurately follow the process.

The four transitions (T028, T029, T030 and T031) were enabled to transmit a record when fired, thus allowing the monitoring of the number of good, faulty, rejected and wrongly rejected parts. This illustrates the potential that exists within the system to provide simple, accurate and user-friendly information. These “pseudo-transitions” can be created to mimic the operation of the system as required.

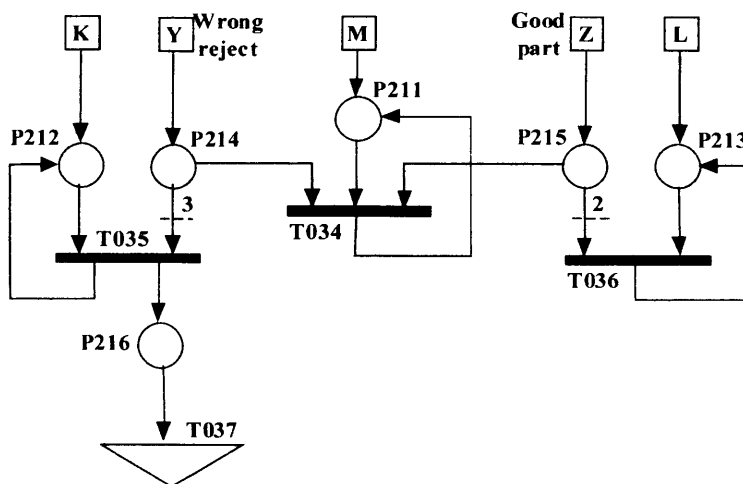


Figure 9.4: Output transition to alarm excessive wrong rejects.

The firing of transition T030 and T031 generates further tokens employed in a Petri-net branch implemented to control the excessive number of “wrongly rejected” parts. This is shown in Figure 9.4. The implementation considers that if a sequence of 3 (4 in the worse case) wrongly rejected parts are detected, transition T035 will fire, consequently resulting in the “output transition” T037 being fired and an output indicating that this fault sequence has arisen may be provided. This is important since a sequence of wrongly rejected parts may indicate that the Rig’s control system has lost synchronisation. A real-time local alarm, such as is provided here, would prevent excessive waste.

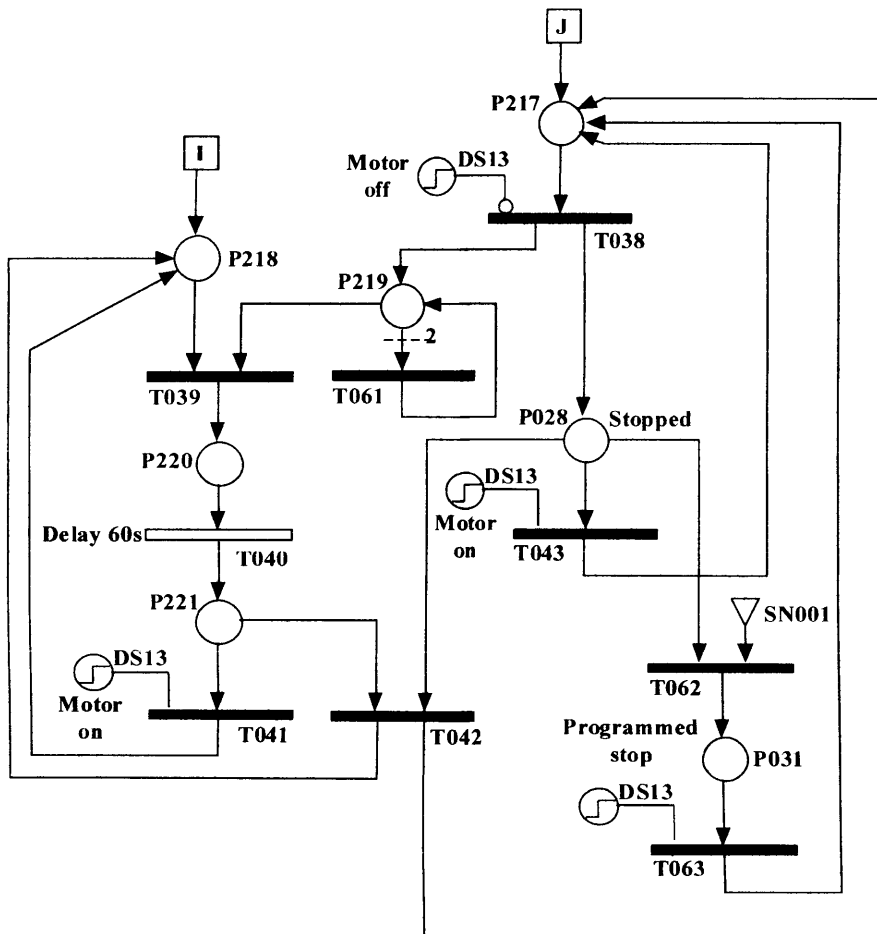


Figure 9.5: Stop and programmed stop implementation.

Four operating states were monitored by the Petri-net: unscheduled stop (P028), programmed stop (P031), available (P030) and processing (P029). Figure 9.5 shows the net branch implemented to monitor the states associated to P028 and P031. A

“stopped” condition is detected when the conveyor motor is switched off (DS13), thus firing T038 and placing a token in P028. Place P219 also receives a token, enabling T039 to fire. A “delay transition” T040 was included with the purpose of restarting P028 every 60 seconds and thus transmitting a record to the database. The process may be restarted and stopped while the delay is running. Transition T061 was therefore required to avoid multiple tokens in P219 and consequently unwanted cycles being processed. A stop state may be ended by the motor switching on (T043) or by a “programmed stop”, initiated by firing T062. A “programmed stop” is used to simulate activities such as planned maintenance or repair, where some pre-determined action is being undertaken. This is done by means of a sub-net signal (SN001), emulated from another CAN node (Development Kit). Transition T063 ends the programmed stop by removing a token from P031 when the motor switches on.

The other two possible operating conditions, “processing” (P029) and “available” (P030) were implemented in the Petri-net branch shown in Figure 9.6. A common characteristic of these states is that the motor (DS13) must be on. Thus, T050 and T058 are required to clear these places when the motor is switched off. For simplification reasons, it was assumed that the Rig is in the “processing” state when there are at least one peg and one ring in each queue (after sorting). This condition will be provided by T006 and T007 (Figure 9.2) and will result in T045 being fired. Transitions T046/T052 ensure that rings that were reprocessed will not be considered. Similarly, T044/T051 will remove from the queue pegs that do not have a matching ring. The components that were already assigned to their pairs will be held in P029. A process stop will move these tokens back to P226, representing a memory feature. Through T057, the condition can change from “available” to “processing”. The implementation hierarchy will ensure that as long as there are tokens in P029 (processing), T055 will not be enabled to fire. Such state change only happens after the last available pair (peg and ring) crosses the beam sensor (DS08).

The different states resulting from the operation of the Conveyor Rig represent an added level of complexity for the modelling task. However, the flexibility represented by the Petri-net approach enables such an implementation, although requiring some engineering effort. It required 77 places, 62 transitions and (by choice) one sub-net. Two digital input cards were deployed to interface the 13 process signals. The text

files required for this Petri-net implementation can be found listed in Appendix B (B.17 [4, 5, 6]). The next section will consider the use of these monitoring events to produce and manage process and performance information.

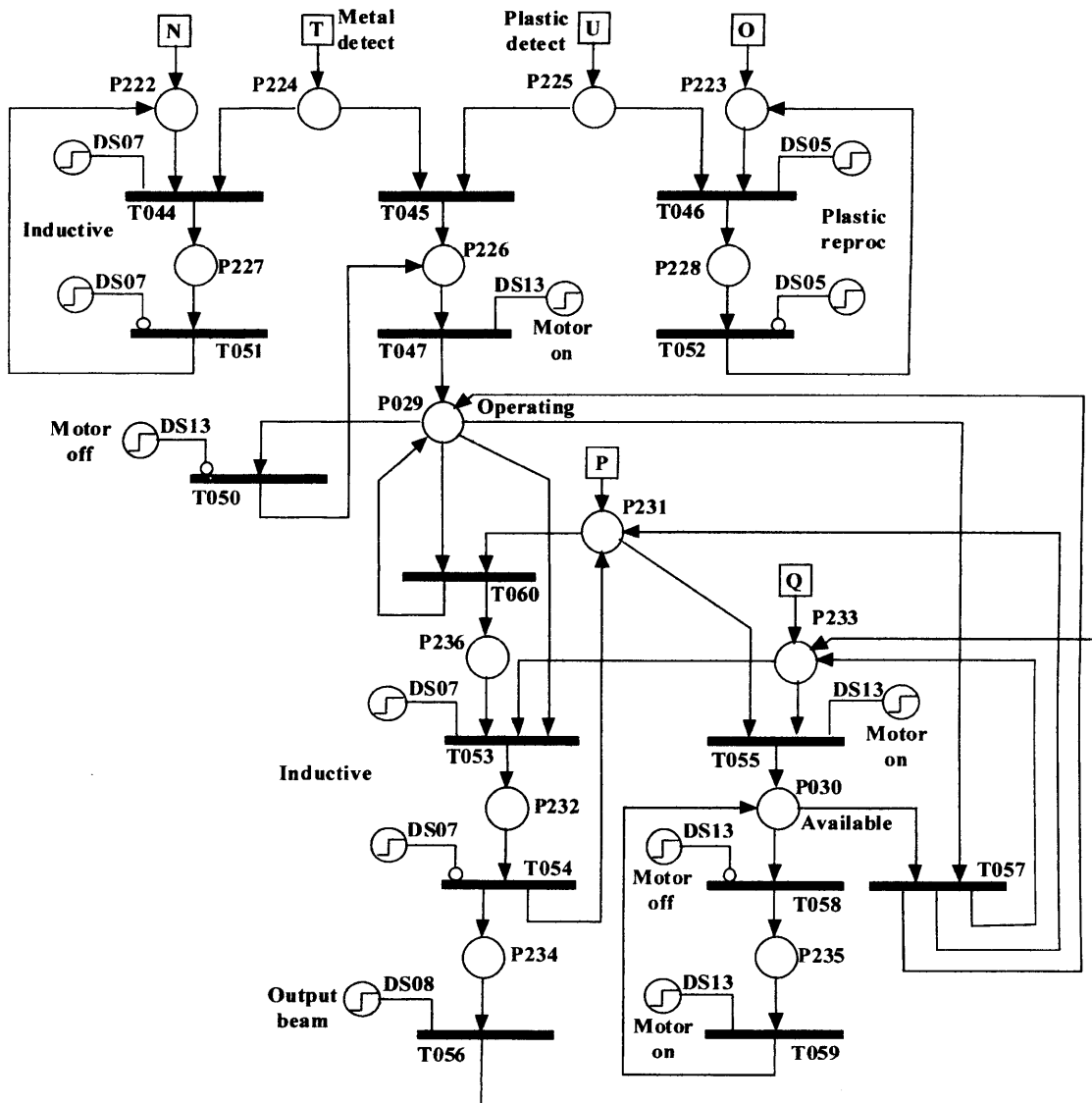


Figure 9.6: Available and processing operating states Petri-net.

9.4 – Monitoring Results

In order to present the results of the monitoring task, the web page approach introduced in Chapter 8 was again employed. This allowed the provision of online information from the process. A first example is shown in Figure 9.7, where a list of

the latest events recorded by the monitoring system are displayed. The first record (T030 – Wrongly rejected) shows that transition T030 in Figure 9.3 was fired, indicating that the manufacturing system made a wrong decision, rejecting a good part. The Monitoring System was capable of detecting such a mistake. The record could be further used to provide process statistics or to request intervention (based on other management applications). A timeout at P011 revealed that a ring was positioned in the assembly area longer than usual, suggesting that components might have been loaded in an irregular sequence (more rings than pegs).

Conveyor Events

Description	TimeRecord	EventRecord
T030 - Wrongly rejected	2003-08-21T13:46:57	
P011 - Excessive wait time in asse	2003-08-21T13:44:21	16
T035 - Excessive good parts rejec	2003-08-21T13:42:51	
T029 - Failed to reject	2003-08-21T13:41:26	
T009 - Plastic ring reprocess	2003-08-21T13:05:33	

Figure 9.7: Conveyor rig monitored events displayed on a web page.

As previously described, a control mechanism was modelled to detect an excessive number of wrongly rejected parts (Figure 9.4). The record “T035 – Excessive good parts rejected” in Figure 9.7 is an example of such event, meaning that at least 3 well assembled parts were continuously rejected by the Conveyor Rig control system and that such mistake was caught by the Monitoring System. A consequence of T035 firing would be the “output transition” T037 (Figure 9.4) producing a local alarm. This reinforces the modelling flexibility allowed by Petri-nets. In a similar way, T029 (Figure 9.3) was deployed to monitor unassembled parts (poor quality) missed by the Conveyor Rig control system. Such an event is show in Figure 9.7 by means of the record “T029 – Failed to Reject”. Finally, the last record shown in Figure 9.7 indicates that a plastic ring could not be queued (reprocessed). This was provided by transition T009 (Figure 9.2), which monitors the number of plastic rings that were not inserted in the queue due to lack of an empty slot. It can represent an important management information, since rings were loaded in a higher number (compared with pegs). This can affect process efficiency, since it may result in an irregular output.

PIC-ConveyorOutput

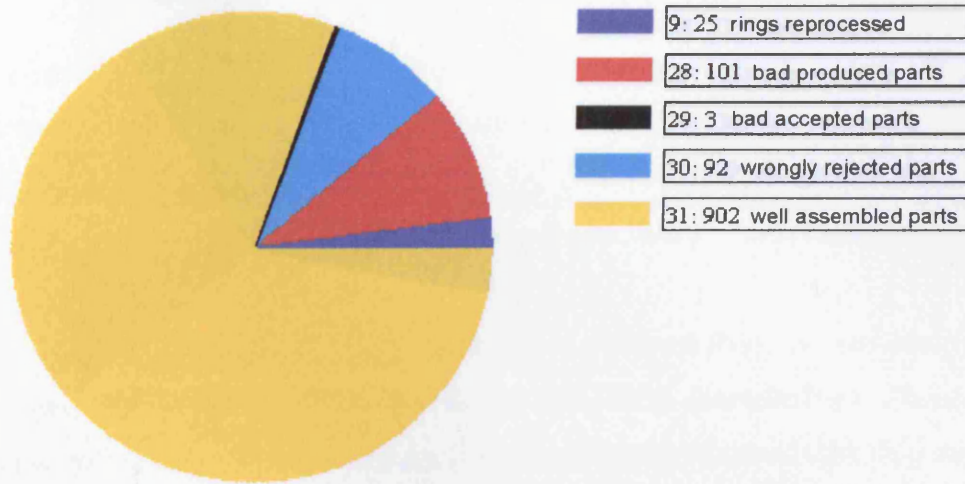


Figure 9.8: Conveyor Rig production statistics provided by the Monitoring System and shown on the web page, indicating the number of “reprocessed rings” (9), “bad parts output” (28), “bad accepted parts” (29), “wrongly rejected” (30) and “well assembled parts” (31).

The records provided by the Petri-net transitions that monitored the Rig “production” output (Figure 9.3) were employed to build a graph that can be displayed on the IPMM web page, as shown in Figure 9.8. It can be seen that during the observation time there were 25 reprocessed rings (T009), 101 parts rejected (T028), 3 bad parts accepted (T029), 92 assembled parts wrongly rejected (T030) and 902 well-assembled parts correctly accepted. This information was continuously updated on the web page, in response to new records from the monitoring system. One interesting point here is that a total of 95 parts (bad parts accepted and good parts rejected) were mis-managed by the existing Rig controller (PLC) and that this, although easy to be observed by a machine operator standing by, would remain undetected in the case of a fully automated operation. This is a clear illustration of the advantages offered by the deployment of the new system.

The bases of such assessments are availability, performance efficiency and quality rates (OEE = Availability x Performance x Quality Rate) [9.1]. Just as an example

PIC-ConveyorUsage

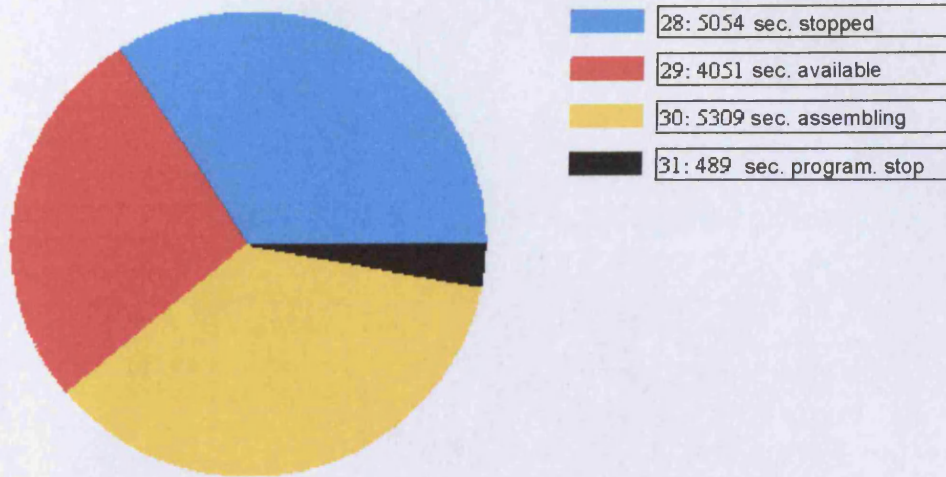


Figure 9.9: Conveyor Rig utilisation provided by the Monitoring System and shown on the web page, indicating how long (in seconds) the process remained “stopped” (28), “available” (29), “processing/assembling” (30) and in a “programmed stop” (31) state.

Information related to the Rig’s utilisation was provided by places P028, P029, P030 and P031 (Figure 9.5 and Figures 9.6). The graph shown in Figure 9.9 represents an example of the presentation of such information. It can be seen that for a total of 5054 seconds the process remained stopped (P028), 4051 seconds were expended running the process without assembling components (P029), 5309 seconds were effectively dedicated to assembling loaded components (P030) and 489 seconds were used by programmed stoppages (P031). The information displayed was obtained by means of a simple SQL statement that totalised each individual group and was called using the IPMM’s web page based approach. Such simplicity is a result of the way in which the Monitoring System provides the monitoring records.

The results shown might be further employed to assess the process in terms of OEE. The bases of such assessment are availability, performance efficiency and quality rates (OEE = Availability x Performance x Quality Rate) [9.1]. Just as an example

that assumed some simplifications due to the nature of the experiment, a database procedure was used to calculate these parameters, considering a period represented by the time elapsed during the actual date. The procedure that implemented these calculations can be found in Appendix B (B.15).

Availability is defined as the relation between the total available time and the process down time (breakdown and set up), as it follows:

$$\text{Availability} = (\text{TotalTime} - \text{DownTime}) / \text{TotalTime} \quad (\text{eq. 9.1})$$

In the example here proposed, this parameter was obtained from the records provided by places P028 (stopped), P029 (available) and P030 (assembling). Programmed stoppages, recorded by P031 were excluded, since it was assumed that they represent previously known conditions and thus should not be considered in the calculation of the process availability. Therefore, in this example availability was obtained as:

$$\text{Availability} = (T_{P029} + T_{P030}) / (T_{P028} + T_{P029} + T_{P030}) \quad (\text{eq. 9.2})$$

Performance efficiency considers in its definition the ideal cycle time to produce a part, the operating time (without down time) and the total output (good and defective parts). In simple terms, it is represented as follows:

$$\text{Performance} = (\text{IdealCycleTime} \times \text{NumberOfOutputs}) / \text{OperatingTime} \quad (\text{eq. 9.3})$$

Considering the purpose of this example, to show the variety of information produced by the Monitoring System and its use, some simplifications were made. It was assumed that the ideal cycle is the one associated with the beginning/ending records produced by place P030 (assembling), effectively the time consumed to produce the total output ($\text{IdealCycle} = T_{P030} / \text{TotalOutput}$). Thus, performance was here obtained as it follows:

$$\text{Performance} = T_{P030} / (T_{P029} + T_{P030}) \quad (\text{eq. 9.4})$$

Quality rate is defined as the relation between the total produced and the defective parts that resulted of the process operation and is represented as it follows:

$$\text{Quality Rate} = (\text{TotalOutput} - \text{NumberOfDefects}) / \text{TotalOutput} \quad (\text{eq 9.5})$$

The records produced by the Monitoring System enable to easily obtain such parameters. In this particular example, the number of good parts produced is automatically obtained from the records provided by transition T031. Similarly, the defective parts can be obtained from the records produced by T028 (bad parts), T029 (bad accepted) and T030 (wrongly rejected). As a result, the quality rate of the Conveyor Rig was calculated as it follows:

$$\text{Quality Rate} = T_{T031} / (T_{T028} + T_{T029} + T_{T030} + T_{T031}) \quad (\text{eq. 9.6})$$

Process OEE (%)

ProcessID	ActualOEE	BestOEE
ConveyorRig	32.37	60.68

Figure 9.10: Conveyor Rig OEE parameter shown on a web page.

An example of the OEE information displayed on the web page is shown in Figure 9.10. Figure 9.11 shows a comparison between OEE measurements obtained at different periods, considering all individual components of the calculation. Since aimed to illustrate the use of the system, the calculation was based on a period of 24 hours, starting at midnight. It must be considered that in a real manufacturing application further information will be probably required. However, it can be seen that the records provided by the Monitoring System can ease considerably such task and help to provide important management parameters.

OEE Comparison (%)

TheDate	Availability	Performance	QualRate	OEE
2003/01/24	49.12	37.53	88.04	16.23
2003/02/04	41.32	42.74	92.20	16.28
2003/02/16	83.60	29.60	92.02	22.77
2003/02/17	83.03	12.29	76.61	7.82
2003/02/19	48.84	51.69	90.90	22.95
2003/02/20	69.82	60.72	92.81	39.35
2003/02/21	91.48	63.51	51.64	30.01

Figure 9.11: Conveyor Rig OEE comparison, considering individual components of the calculation.

9.5 – Summary

In this chapter it was illustrated the use of the System to monitor an automated manufacturing process, based on a laboratory equipment. A dynamic illustration of this example is referenced in Appendix B (B.18 [12]). The example required a high level of flexibility from the modelling approach in order to correctly model all the complexity represented by the application and thus making it possible to detect misjudgements (actions) performed by the equipment's controller. Also, the use of a sub-net approach was introduced. In using the System's available resources, time measurements of process specific operating states were obtained and thus showed the capability of the System in helping to provide management information that would be praised in many manufacturing environments.

Although the examples so far presented were based on the use of laboratory equipment, they served to illustrate the modelling approach and the System's monitoring resources. The next chapter will present the use of the Monitoring System to monitor a tool changer of a CNC machine centre, thus illustrating a real manufacturing application.

REFERENCES

- 9.1 Willmott, P, Total Productive Maintenance: The Western Way, Oxford, Butterworth-Heinemann, 1994.

CHAPTER 10

TOOL CHANGER MONITORING TASK

10.1 – Introduction

Two monitoring tasks, using laboratory rigs, were presented in the previous chapters. These examples were presented to demonstrate the flexibility of the Petri-net based monitoring method in modelling an application and its capability in providing online information. This chapter will present the use of the Monitoring System in a real manufacturing application.

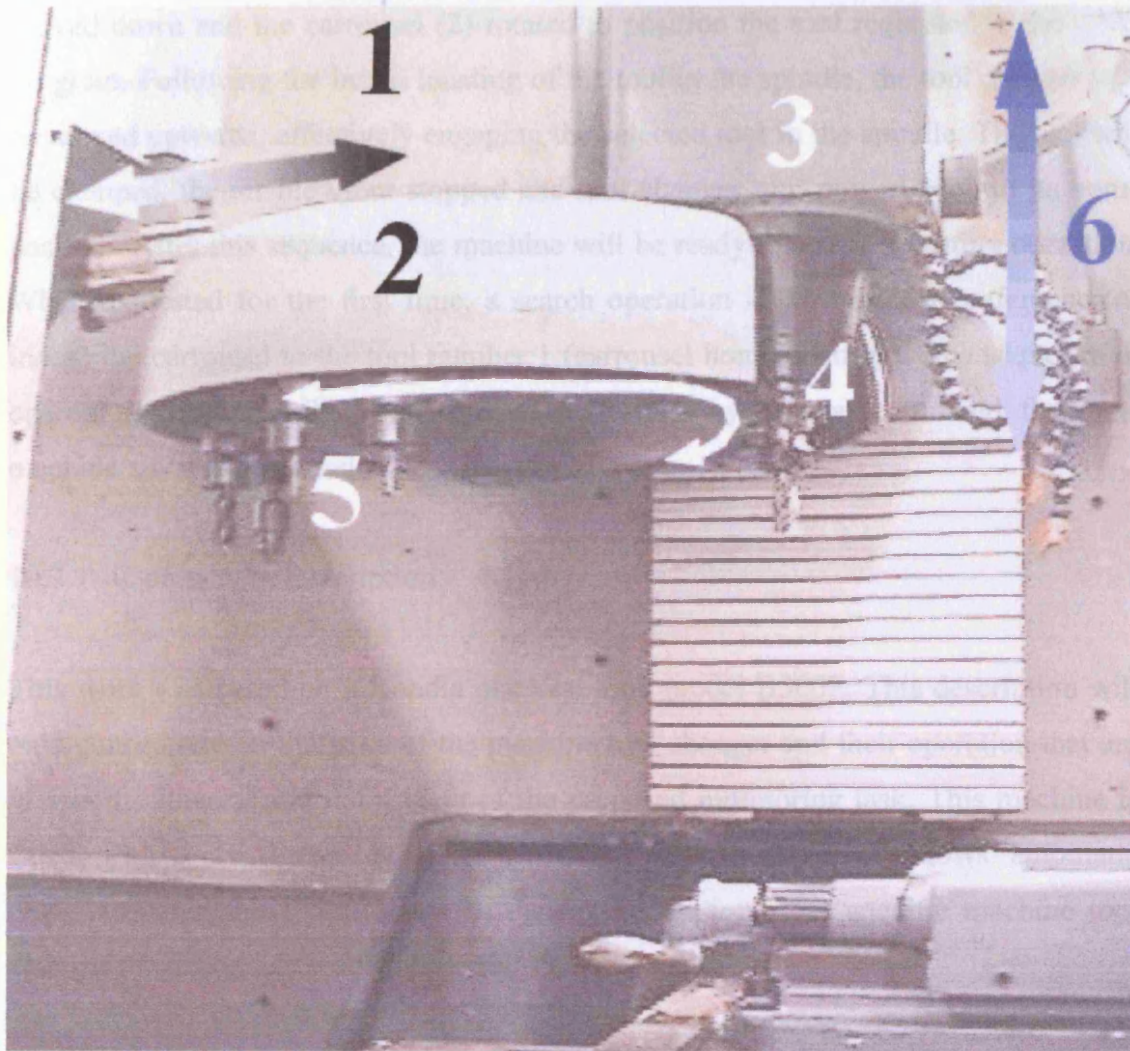
In a previous investigation [10.1], it was shown that CNC machine centres were affected by a number of disturbances that were originated by peripheral actions, with the highest indices related to the tool changer. It is apparent that by monitoring the sequence of events related with the operation of such devices, faults may be detected and thus the downtime due to this sort of disturbance reduced. It should also be possible to provide statistics based on the number of operations, helping to enhance process efficiency. It was also intended that an indication of individual tool utilisation and usage might be provided as an important input into a planned tool-life assessment system.

The example provided in this chapter will illustrate the use of the Petri-net modelling approach to monitor the operation of a tool changer of a milling machine, thus enabling a demonstration of how the Petri-net based monitoring system is able to help in achieving such benefits.

10.2 – Operation Description

The complex and engineered operations performed by a CNC machine require a set of tools with different shapes and sizes. To allow the automatic selection of these tools,

from within the CNC cutting program commands, an automated tool changer is required. Figure 10.1 shows the main elements of a tool changer that can be involved in such an operation.



- | | |
|------------------|----------------------|
| 1 – Tool changer | 4 – Clamped tool |
| 2 – Carrousel | 5 – Tool magazine |
| 3 – Spindle | 6 – Z axis direction |

Figure 10.1: Details related with tool changing operation.

Referring to the details indicated in Figure 10.1, to perform a tool changing operation, the spindle unit (3), which is located on the “Z axis” (6), is required to be at “zero speed” (stopped) and in the “up position” (home). The spindle must be “orientated” (i.e. at zero degrees), in order to align the clamped tool (4) with the holder in the tool magazine (5). Once these conditions are satisfied, the tool changer unit (1) will be

moved forward, from the home (far left) to the front (changing) position. Once engaged in the magazine holder, the tool will be ejected from the spindle. An air blow-out jet based function is then activated to remove dust and residuals that may tend to settle inside the clamping cavity. The whole tool changer unit will then be moved down and the carousel (2) rotated to position the tool requested in the CNC program. Following the initial locating of the tool in the spindle, the tool changer will be moved upwards, effectively engaging the selected tool in the spindle. The tool will be clamped, the air blow-out stopped and tool changer unit moved back to its home position. After this sequence, the machine will be ready to start the cutting operation. When requested for the first time, a search operation is performed to reference (or index) the carousel to the tool number 1 (carousel home position). The sequence of operations here described can also be performed by manual command from the machine's controller panel.

10.2.1 – Component Description

This work was based on a Kondia machine tool, model B500P. This description will only concentrate on the parts of the machine tool changer and their operation that are of specific interest and the subject of the proposed monitoring task. This machine is fitted with a GE-Fanuc Series O-M CNC controller, which allows automatic (program) or manual control of the machine operation. This specific machine tool changer magazine holds up to 18 tools.

In order to manage the sequence of events previously described, several sensors and specific signal commands are required. A pneumatic system provides the tool changer's horizontal movement. This is controlled by a "feed forward" command. Two of the deployed sensors will detect the "back" (home) and "front" (changing) position. A similar system manages the vertical movement, with sensors to detect the "up" and "down" position.

The carousel operation is controlled by a pair of signals: the rotate command and rotating direction. A sensor was provided to detect the initial (indexed) position. To position a new selected tool the entire set of tools is moved around until it reaches the tool-changing operation position. A single sensor is used to implement a counting

approach, generating a pulse each time a tool crosses the changing position. The controller identifies the tool by counting the number of pulses relative the index (tool 1), taking into account the position where it is located before starting to rotate. The shortest way to reach the new tool will be selected, an operation requiring incremental and decrementing counting. A third sensor exists to detect whether or not a tool is placed in the holder, when it is aligned with the carousel changing position.

The tool changing operation will take place once the spindle is correctly positioned and orientated. A sensor indicates that the “Z axis” is in the required changing position. Further sensors detect when the tool is unclamped and clamped. Controller commands provide signals for the “spindle orientation” (zero degree), “zero speed” and “blow-out”.

The spindle rotation is provided by an AC motor, controlled by a servo device. The maximum cutting speed delivered is of 6000 rpm. Two analogue outputs (0 ~ 10 V) provide speed information and motor’s load current.

10.2.2 – Monitoring Task Analysis

The monitoring application considered the use of the existing process signals. Table 10.1 summarises the signals that were accessed for this implementation.

The Petri-net model was to be designed to follow the events related to the tool changing operation, therefore providing real-time information to allow a fast recovery in case of operational faults. In addition, in order to provide statistics related to the tool usage, the implementation was also enhanced to provide a mean of indicating how long each tool was effectively used. Similar functionality is provided by some machine tool centres within the machine’s controller. However, by exploring the Monitoring System’s capabilities, the potential for the development of the system into one capable of better managing tool utilisation was further improved. This improvement was based upon using the spindle driver analogue signals, the motor load current and cutting speed parameters, which were used to provide more accurate information about the cutting process being performed by the current cutting tool. Considering such information can be made available via a database, further

manipulation and integration within production management systems might help to improve quality and perhaps productivity.

Table 10.1: Tool changing operation signals description.

	Description	Signal Source	State
1	Carrousel ON / OFF - command	Relay	On - 0 V
2	Carrousel rotate clockwise - command	Relay	On - 0 V
3	Carrousel rotate counter clockwise - command	Relay	On - 24 V
4	Carrousel feed forward - command	Relay	On - 0 V
5	Carrousel down - command	Relay	On - 0 V
6	Unlock tool – command	Relay	On - 0 V
7	Spindle orientation – command (pulse)	Controller output	On - 24 V
8	Spindle zero speed – command	Controller output	On - 24 V
9	Spindle blow-out - command	Relay	On - 0 V
10	Carrousel indexed	Proximity sensor	On - 24 V
11	Tool counter	Proximity sensor	On - 24 V
12	Carrousel is forward (changing position)	Proximity sensor	On - 24 V
13	Carrousel is back (home position)	Proximity sensor	On - 24 V
14	Carrousel is up	Proximity sensor	On - 24 V
15	Carrousel is down	Proximity sensor	On - 24V
16	Tool in carrousel – changing position	Proximity sensor	On - 24V
17	Tool is clamped	Switch	On - 24 V
18	Tool is unclamped	Switch	On - 24 V
19	Z axis at changing position	Proximity sensor	On - 24 V
20	Spindle motor load current	Analogue	0 ~ 10 V
21	Spindle rotating speed	Analogue	0 ~ 10 V

10.3 – Tool Changing Petri-net

The Petri-net was designed to monitor the tool changing operations and provide the tool usage parameters. Table 10.2 shows the definitions employed in the implementation. In considering the way the tool changing operation is carried out, the Petri-net was divided in four branches, related to each other:

- 1) Tool selection and cutting process monitoring (Figure 10.2);
- 2) Z axis positioning and spindle orientation (Figure 10.3);
- 3) Tool changer horizontal movement (Figure 10.4);
- 4) Tool changing operation (Figure 10.5).

Table 10.2: Tool changing Petri-net's definitions.

Representation	Description
DS 01	Carrousel ON /OFF
DS 02	Carrousel indexed
DS 03	Tool counter
DS 04	Carrousel rotating direction
DS 05	Carrousel is back
DS 06	Spindle zero speed
DS 07	Spindle orientation command
DS 08	Z axis changing position
DS 09	Carrousel feed forward
DS 10	Carrousel is forward
DS 11	Tool in carrousel sensing
DS 12	Carrousel is up
DS 13	Carrousel is down
DS14	Tool unclamped
DS15	Tool clamped
DS16	Spindle blow-out
AN01	Spindle motor current
AN02	Spindle rotating speed
P201 – P218	Cutting state – tools 1 to 18
P100	Last timed out place

10.3.1 – Tool Selection and Cutting Process Monitoring

This initial section considers the selection of the tool that is to be inserted into the machine and its subsequent usage. The beginning of the Petri-net execution is shown in Figure 10.2. Transition T001 will be fired if the tool changer is at the home position (DS05) and the carrousel stopped (DS01). Following this, the Petri-net needs to synchronise with the carrousel index position, in order to identify the tools correctly. This is achieved by means of transition T002. The signal generated by the carrousel home (index) sensor (DS02), together with an active indication provided by the tool count sensor (DS03), will mean that tool number 1 is synchronised, thus resulting in a token in place P101. The firing of T002 will also enable the other Petri-net branches (changer movement, Z axis positioning and changing operation), thus synchronising the entire monitoring task.

In order to position a selected tool for changing, the carrousel is turned either clockwise or counter-clockwise. This operation requires two signals: the carrousel command (DS01) and the rotating direction setting (DS04). The tool counting sensor

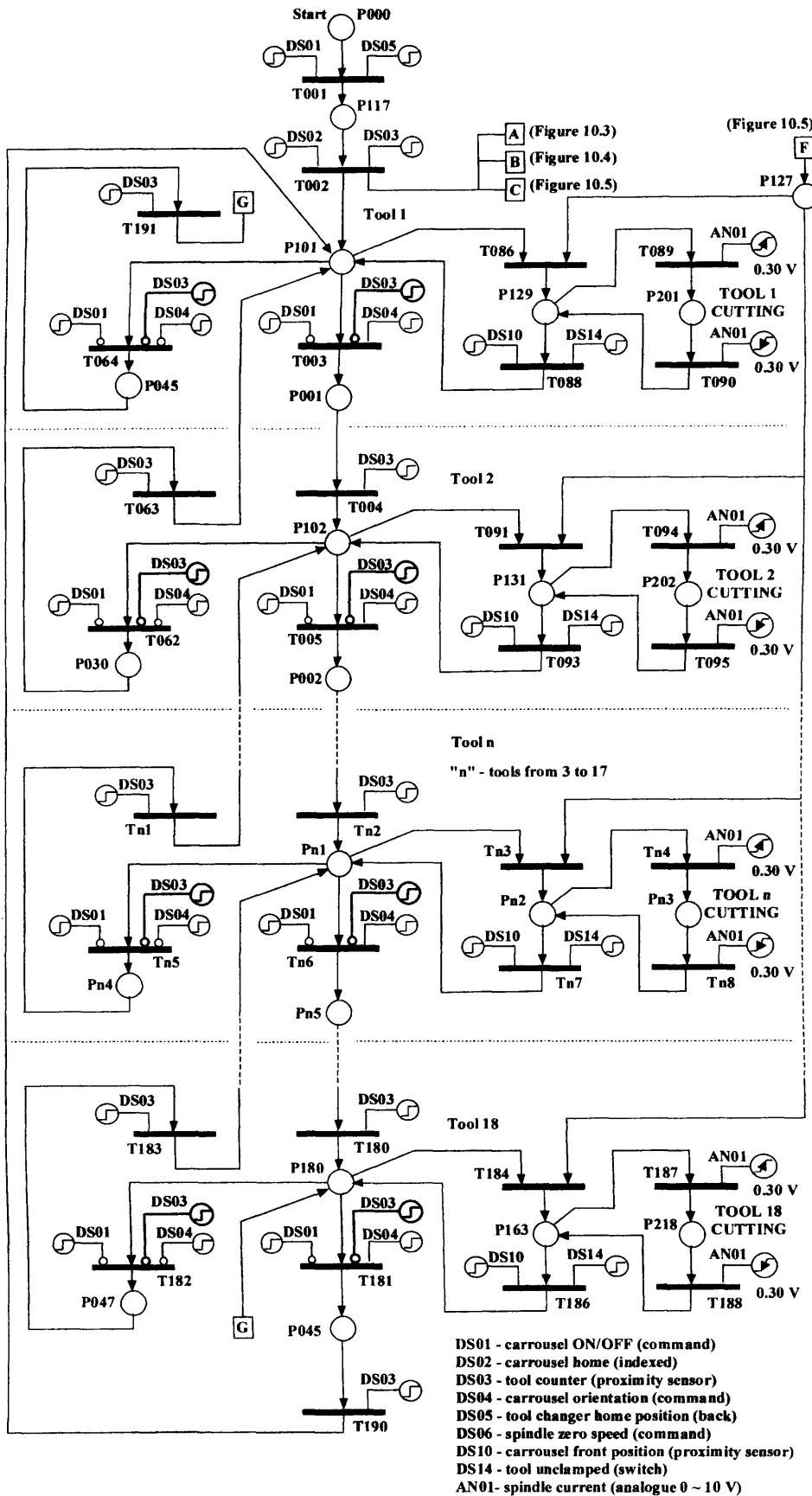


Figure 10.2: Tool selection and usage Petri-net diagram.

(DS03) becoming inactive (off) indicates that the carousel is moving between two adjacent tools. In Figure 10.2, assuming that “tool 1” is the actual position (represented by a token in P101), this movement is followed in the Petri-net by the firing of either T003 (clockwise) or T064 (counter-clockwise). A token in P001 indicates that the carousel is moving towards tool number 2, while a token in P045 indicates that the movement is in the direction of tool number 18. The next tool position (2 or 18) will be detected when the DS03 (tool count) sensor becomes active (on), which results in T004 or T191 being fired. The carousel may stop or carry on moving, depending upon which tool is being positioned. The representation of this operation, in terms of the Petri-net, is similar for all tools. Thus, for simplification reasons, Figure 10.2 provides specific representation for tool’s 1, 2 and 18 and shows a generic model “n” for all the remaining ones (3 to 17). Table 10.3 provides a reference for the Petri-net places and transitions represented by the generic element tool “n”.

Table 10.3: Reference to generic element “n” in Figure 10.2.

Generic Element	Associated Petri-net elements (Tool-3, ..., Tool-17)
Tn1	T061, T059, T057, T055, T053, T051, T049, T047, T045, T043, T041, T039, T037, T035, T173.
Tn2	T006, T008, T010, T012, T014, T016, T018, T020, T022, T024, T026, T028, T030, T032, T170.
Tn3	T096, T101, T106, T111, T116, T121, T126, T131, T136, T141, T146, T151, T156, T161, T174.
Tn4	T099, T104, T109, T114, T119, T124, T129, T134, T139, T144, T149, T154, T159, T164, T177.
Tn5	T060, T058, T056, T054, T052, T050, T048, T046, T044, T042, T040, T038, T036, T034, T172.
Tn6	T007, T009, T011, T013, T015, T017, T019, T021, T023, T025, T027, T029, T031, T033, T171.
Tn7	T098, T103, T108, T113, T118, T123, T128, T133, T138, T143, T148, T153, T158, T163, T176.
Tn8	T100, T105, T110, T115, T120, T125, T130, T135, T140, T145, T150, T155, T160, T165, T178.
Pn1	P103, P104, P105, P106, P107, P108, P109, P110, P011, P112, P113, P114, P115, P116, P170.
Pn2	P133, P135, P137, P139, P141, P143, P145, P147, P149, P151, P153, P155, P157, P159, P161.
Pn3	P203, P204, P205, P206, P207, P208, P209, P210, P211, P212, P213, P214, P215, P216, P217.
Pn4	P029, P028, P027, P026, P025, P024, P023, P022, P021, P020, P019, P018, P017, P016, P046.
Pn5	P003, P004, P005, P006, P007, P008, P009, P010, P011, P012, P013, P014, P015, P042, P048.

The sequence of places 201 to 218 is used to represent the tools, respectively from number 1 to 18, when they are in their respective cutting state (begin / end records). Once a tool is positioned and the tool changing operation has been executed, place P127 receives a token indicating that the tool is engaged in the spindle. Assuming that a generic tool “n” was positioned (token in Pn1) and the changing operation carried out (token in P127), transition Tn3 would be fired, resulting in a token in Pn2, thus indicating that the tool is ready for use.

The analogue input 1 is representative of the spindle motor current. It was determined by observation that, although variation obviously occurred accordingly the cutting speed, the analogue level representing this current was normally below 0.3 V during what can be called the idle state (i.e. when the cutter is rotating without cutting). This was therefore taken as the threshold to detect the onset of cutting and then to represent the fact that the cutter was actually cutting. Two “analogue transitions” for each tool were employed to detect the beginning (Tn4) and the end (Tn8) of the cutting state. Place Pn2 will retain the token if the tool is not cutting. This is an important development that will be discussed in detail later. At this point it is worth noting however that it would be possible, in theory, to set a series of thresholds that could be used to represent just how much cutting the cutter is actually doing. This information could then be used as the basis for a more accurate representation of tool usage than is currently available.

A new tool changing operation will result in the tool changer reaching the front position (DS10) and in the tool being unclamped (DS14), thus firing Tn7 and returning the token to Pn1. In doing so, the tool position is memorised and the following tool selection will be monitored considering the previously engaged tool. The places representing the cutting state (P201 to P218) were also used to trigger the acquisition of the analogue input channel 2, related to the cutting speed.

10.3.2 – Z Axis Positioning and Spindle Orientation

The first action in the tool changing operation is positioning the “Z axis” and performing the spindle orientation. The Petri-net branch in Figure 10.3 shows the monitoring sequence of this operation. Place 119 retains an initialisation token

provided by T002 (Figure 10.2). To allow a tool changing operation to take place, a sequence of events related to the machine’s spindle axis will be required. Initially, the spindle must be stopped, a condition indicated by the “zero speed” command (DS06), resulting in T065 being fired and P121 receiving a token. The next event would be the “Z axis” moving to the changing position, recorded by T066 being fired and P122 receiving the token. In the next step, the spindle will be orientated (to zero degrees), a condition detected by DS07. After this sequence of events, the spindle will be in the “changing state” (P031). The spindle rotating again will indicate the end of this state. Transitions T067 and T069 were provided, considering that for any unpredictable reason (e.g. a manual operation), the spindle starts to rotate before a entire tool changing sequence related to the “Z axis” was completed. In case of such an event, the token will be returned to P119 by means of these transitions, maintaining the Petri-net synchronisation.

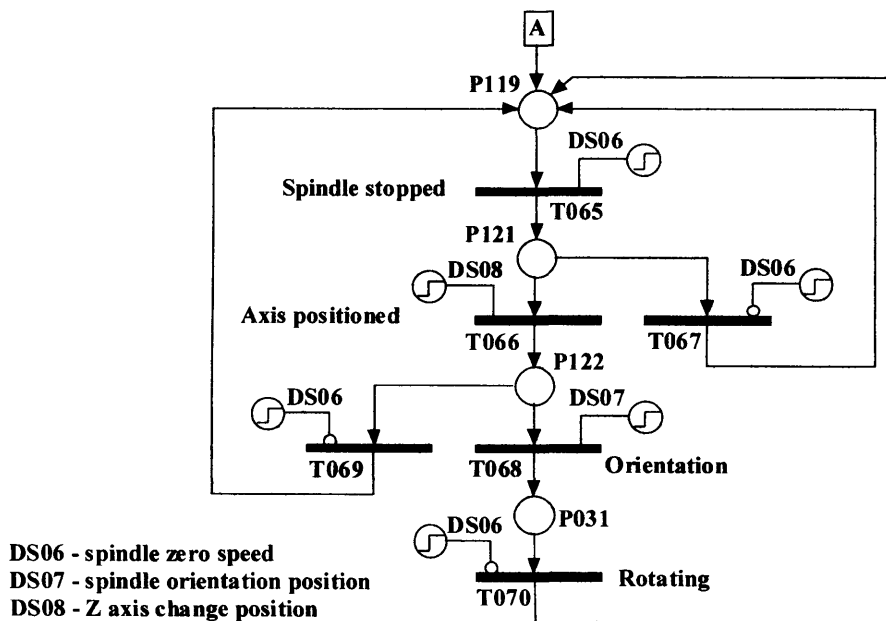


Figure 10.3: Z-axis and spindle positioning Petri-net.

10.3.3 – Tool Changer Horizontal Movement

Once the “Z axis” has been positioned the tool changer will be moved towards the changing position. Figure 10.4 shows the Petri-net branch that monitors this sequence of events. At initialisation (Figure 10.2, T002), P120 will receive a token. The

detection of the tool changer at its “home position” (DS05) will fire T071 and result in a token in P124. The “feed forward” command (DS09) will lead to a new state, “move forward” (P032), which should be immediately followed by T073 being fired to indicate that the tool changer has moved away from the home position (DS05). A timeout event related to P032 would indicate a fault condition (changer is not moving). The state represented by P033 indicates that the tool changer is on its way. This state will be ended by the changer reaching the “front position” (T074, DS10). Place P125 represents the tool changing operation, which will be ended with the tool changer leaving the “front position” (T075, DS10) and returning to the home position (T071, DS05).

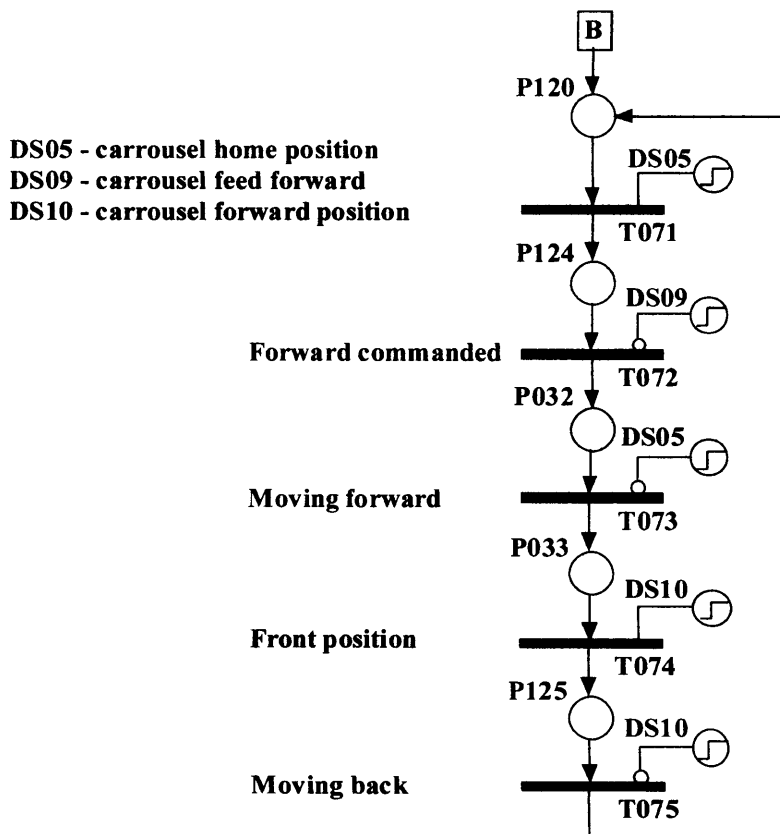


Figure 10.4: Tool changer horizontal movement monitoring.

10.3.4 – Tool Changing Operation

The last of the Petri-net’s branches relates to the tool changing operation that actually changes the tool in the spindle, as shown in Figure 10.5. The branch is enabled by a

token in P118. Three conditions are required in order to enter the tool changing sequence: Z axis at the “changing position” (DS08), tool changer at the “front position” (DS10) and the carousel must be “up” (DS12). The next event will be the unclamping of the engaged tool (T079, DS14). The carousel should start to move away from the “up position” (T080, DS12), removing the actual tool from the spindle. The “moving down” state is represented by P037, which will be finished when the “down position” (DS13) is reached, firing T081 and placing a token in P038. At this point, the tool selection operation represented in the Petri-net branch shown previously in Figure 10.2 will take place. Since the new tool was placed in the changing position, the carousel will leave the “down position” (DS13), firing T082.

There is a possibility that the new selected position in the carousel might be empty, resulting in a wrong tool usage parameter (Figure 10.2). This condition is detected by the “tool positioned” proximity sensor (DS11). A level of complexity arises due to the fact that this signal must be verified before the carousel reaches the “up position”, since the proximity to the spindle might induce a false “tool in place” indication. It was also observed that the tool changer starts to move up before the carousel has completely stopped rotating, meaning that an empty position might be sensed, even if a tool is in place. The solution was to use a “delay transition” structure (T192) to introduce a delay (500 ms) between the carousel leaving the low position (T082) and the tool sensing signal being checked (T078 and T077).

The existence of a tool (T077) will result in a token placed in P127 (Figure 10.2), enabling the tool usage to be monitored. This token will be generated after the tool was clamped (T167, DS15). If there is not a tool placed in the new selected position (T078), P127 will not receive the token. Place P039 was employed as a “joined” place to enable the next events to be monitored, whatever the “tool positioned” signal checking result was. Transition T083 will be fired by the tool changer reaching the “up position” (DS12). The next event would be the air blow-out (DS16) being stopped (T166) and the tool being clamped (T084, DS15), even if there is not a tool in place. Finally, the Petri-net branch in Figure 10.5 will monitor the tool changer returning to the home position (DS05), replacing the token in P118 and thus enabling the next cycle to be monitored.

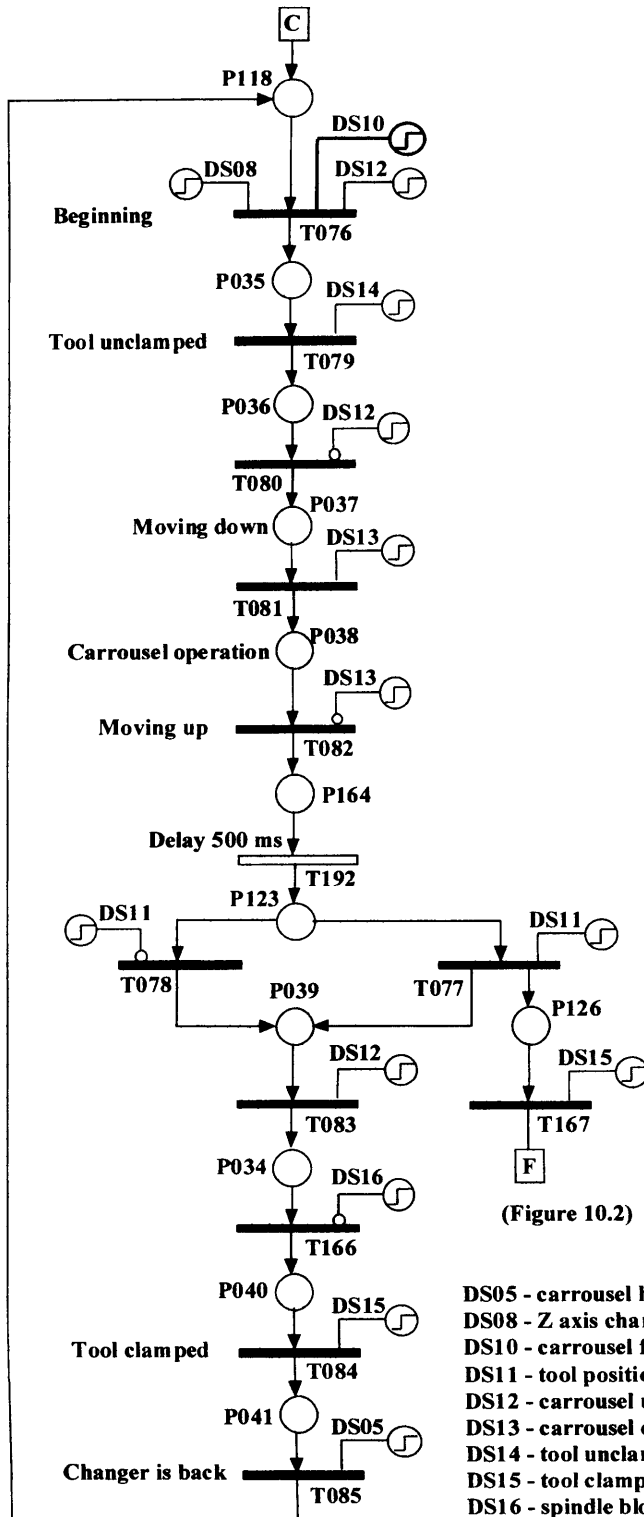


Figure 10.5: Tool changing operation Petri-net.

This Petri-net implementation needed, in order to monitor the entire set of 18 tools, 170 transitions and 104 places. Two digital input cards were required in order to enable the 16 process digital signals to be interfaced. An analogue input (AN01) was

used to detect the cutting condition. A second analogue input (AN02) was used to monitor the cutting speed. Places P201 to 218 were used to trigger the cutting speed acquisition. The files describing the Petri-net definition can be found in Appendix B (B.17 [7, 8 and 9]). The monitoring results will be presented in the following section.

10.4 – Monitoring Results

The records provided by the Monitoring System were analysed and presented using the IPMM web page based approach. One of such examples is shown in Figure 10.6. In this case, a database procedure was implemented to retrieve records of interest from the system's database tables, before organising and describing them in a way that they could help users / operators to follow the operations that were performed by the machine.

Cutting Events

Description	TimeRecord	EventRecord
Blowout delayed	2003-03-12T14:09:34	64
Z axis ready for tool change opera	2003-03-12T14:08:42	
Carrousel is at back position	2003-03-12T14:08:31	
Slow moving up	2003-03-12T14:08:23	8
Slow tool changing operation - Z	2003-03-12T13:57:16	128
Carrousel indexed at TOOL 1	2003-03-12T13:11:23	
Slow unclamp operation - check a	2003-03-06T10:47:19	256
Slow response to feed forward - ch	2003-03-06T10:42:51	64
Spindle stopped	2003-03-06T10:26:14	
Carrousel is forward at change pos	2003-03-06T10:25:10	

Figure 10.6: Machine operation events monitored by the system.

One of the main objectives of the Petri-net implementations was to monitor the usage of the cutting tools. Figure 10.7 shows the results of such an observation. Recalling Section 10.3.1, the states of the tools in use are represented by places P201 to P218 (one for each of the 18 tools). Thus, Figure 10.7 graphically indicates that during this test tool number 2 (202) was in use for a total of 98 seconds, number 4 (204) for 21 seconds and number 7 (207) for 248 seconds.

Cutting Tool Use (s)

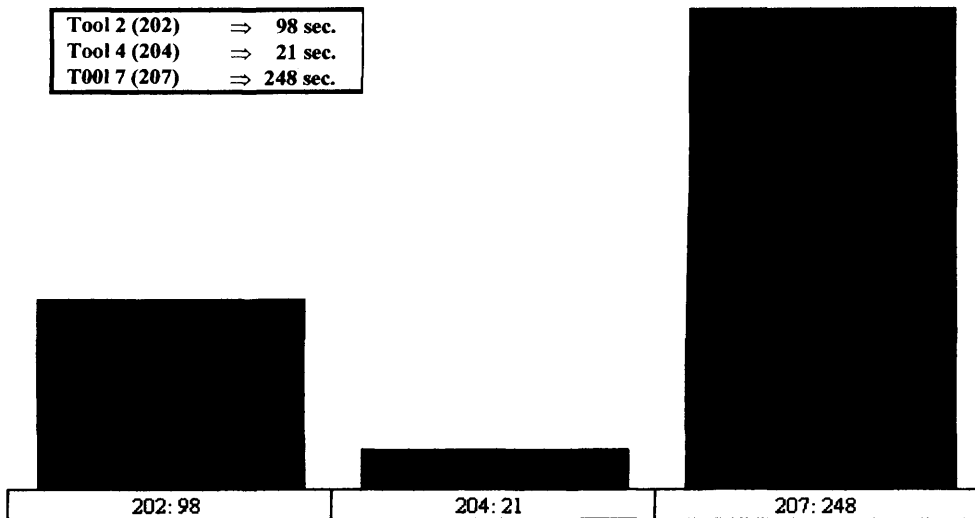


Figure 10.7: Usage parameter of tools 2, 4 and 7 (in seconds).

CuttingTool

ToolNum	Total min	Avg rpm
2	1.63	1341.94
4	0.35	1283.34
7	4.13	2320.56

Figure 10.8: Tool cutting information in terms of total usage and average speed.

The tool usage information was combined with the measurement of the cutting speed, to produce the online information shown in Figure 10.8. It shows the total use of each tool (in minutes), together with the average cutting speed (in rpm). The acquisition of the cutting speed was triggered by the places representing the tools cutting states (P201 to P208). After each cutting cycle, the measured speed was converted in an average parameter by the Monitoring Module and stored in the system's database tables. Further calculations were performed using SQL statements, in order to obtain the displayed information.

Condition monitoring of cutting tools has been exhaustively investigated and one of the main difficulties reported were the dynamics associated to the process (Chapter 3). Although not concentrating on specific cutting tool condition monitoring techniques, the features provided by this Monitoring System can help in providing additional information to improve the accuracy of cutting tool assessment.

A parameter of interest in such process, to maintenance engineers for example, might be the time required for each tool changing operation. By enabling place P125 (Figure 10.4) to transmit records to the database indicating the beginning and end of this operation, such parameter could be obtained, as shown in Figure 10.9. In this example, each tool operation recorded is shown associated with the respective timestamp and duration. The example shows the flexibility of the Monitoring System in producing different information.

ToolChangeOperation

Change	Seconds
2003-03-12T14:08:11	20
2003-03-12T14:06:06	4
2003-03-12T14:05:45	4
2003-03-12T14:04:05	4
2003-03-12T14:03:46	5
2003-03-12T14:02:36	4
2003-03-12T14:02:19	4
2003-03-12T14:00:48	4
2003-03-12T14:00:16	4
2003-03-12T13:53:47	4
2003-03-12T13:47:11	5
2003-03-12T13:15:59	5
2003-03-12T13:11:18	9
2003-03-06T10:52:29	6
2003-03-06T10:50:35	6
2003-03-06T10:48:46	20

Figure 10.9: Tool changing recorded time.

10.5 – Summary

In this chapter a final example illustrating the use of the monitoring system that resulted from this research was presented. An important component of the cutting process, the tool changer, was monitored in order to enable the associated operation events to be recorded. Additionally the cutting tools usage was also monitored to provide detailed information that can help in the assessment of the tools condition.

Although basic examples were employed to illustrate the use of the Monitoring System, they have helped to show the capability and flexibility provided by the Petri-net approach implemented here. The hardware architecture enabled an easy deployment and the use of the adequate resources at the right level. Although microcontrollers are becoming extremely powerful, there still are many parts in the processing hierarchy that are easier to implement and better performed at higher levels. Such discussion will be provided in the next chapter, considering the capabilities and limitations associated with the implementation of the model here proposed.

REFERENCES

- 10.1 **Prickett, P.W. and Grosvenor, R.I.** Non-sensor Based Machine Tool and Cutting Process Condition Monitoring. *International Journal of COMADEM*, 1999, 2(1), 31-37.

CHAPTER 11

DISCUSSION

11.1 – Introduction

The current generation of monitoring systems in general and especially of condition monitoring systems has benefited from recent developments in computer based technologies. Faster PCs with ever increasing resources, specific data acquisition hardware and a new generation of software tools for system integration and signal analysis have helped to reduce the complexity of developing monitoring infrastructures, allowing researchers to concentrate on the analysis aspects. However, the question that remains to be answered is how practical such implementations can be, particularly when they are deployed in large scale applications.

A number of researchers, including those reviewed in Chapter 3, have been investigating monitoring technology. In the particular area of condition monitoring these investigations have concentrated mainly on the physical phenomena and signal analysis, in order to provide the means to assess the state of the process and asset's operating conditions. Earlier, in Chapter 2, considerations were made with respect to the applicability of monitoring systems, with special emphasis on cost, deployment and data integration.

The use of these technologies in manufacturing plant monitoring applications has raised questions related to accessibility and data integration. This led to the proposition of models, some of them considered in Chapter 3, concerned with the implementation of monitoring systems, taking into consideration the development and management aspects of such systems.

The Internet has established itself and can now be seen as a mature and well-developed environment. It is thus becoming a vital part of an ever-increasing number of system implementations. Such a trend has also been observed in the monitoring

field, enabling the establishment of remote monitoring deployments which can benefit from the facilities offered by a central database, high-cost analysis tools and human expertise that this makes accessible.

The development of these network technologies has offered the prospect of the greater use of condition monitoring. Associated with such developments a new generation of smart and intelligent transducers have emerged, spreading the concept of distributed systems in the automation field. As a result standards have been proposed concerned either with industrial network aspects or guiding the development of this new generation of transducers. Despite all these advances in terms of technology and development, cost is still an issue to be tackled in order to stimulate the more widespread use of monitoring systems. These aspects were considered in the conduct of this research, as previously presented, and will be further discussed here.

11.2 – Implementation Aspects

The use of microcontrollers in this context, besides representing a low-cost alternative, provides further benefits by enabling low-power consumption and less installation requirements. In many cases, depending upon the plant complexity and on the accessibility of the machine or its relevant parts, these represent important factors for further consideration. However the use of microcontrollers in some cases may result in limitations, in terms of resources and in the flexibility of the applications developed.

11.2.1 – Hardware Considerations

The use of microcontrollers as part of a process monitoring system requires considerable effort in both hardware and software development. The benefits that should arise from such developments, especially in terms of cost, depend strongly on the capability of the resulting implementation being easily adaptable to meet any application requirements.

PIC microcontrollers are a generation of devices that offer attractive cost, together with a reasonable number of embedded programmable functions that are normally required in most monitoring applications. An initial investigation showed that the implementation of a system for general-purpose applications would need the highest specification PIC devices in terms of data and programming memory. The PIC18 family represented an advance in such terms (compared with other PIC families) and allowed the adoption of these devices as the basis for a general and low-cost monitoring system. In assessing the suitability of a microcontroller to undertake monitoring functions however there are a number of hardware related factors that must be considered. One of such factors was to base the System's implementation on the microcontroller's available resources, in particular the embedded data memory.

```

.
.
.
CLRF      TRISD          ;port D as output
MOVLW    MemAddrLsb    ;least significant part of the memory address
MOVWF    PORTD         ;LSB of address on the PIC's data bus
BCF      PORTB,LatchLsb ;latch LSB of address
MOVLW    MemAddrMsb    ;most significant part of the memory address
BSF      PORTB,LatchLsb ;unselect LSB latch
MOVWF    PORTD         ;MSB on PIC's data bus
BCF      PORTB,LatchMsb ;memory device MSB address
NOP      ;no operation to allow bus levels to be latched
BSF      PORTB,LatchMsb ;unselect MSB latch
SETF     TRISD         ;port D as input
BCF      PORTB,MemRead ;set memory read signal
NOP      ;delay to enable stable data on bus
MOVF     PORTD,W,A     ;read data byte
BSF      PORTB,MemRead ;release memory device
.
.
.

```

Figure 11.1: Example of a sequence of instructions required to retrieve a data byte from an external memory device.

It may be argued that memory should not be a constraint, since an externally added device can be employed. In fact, such consideration is valid, but the use of external devices must be properly assessed, in order to identify the effect that their use may have on performance. Any operation involving the microcontroller's file register (data memory) will generally speaking be performed in one single instruction cycle. Such

an operation could be retrieving, testing or altering the contents of a register. The same operation, using an external memory device, would require a sequence of instructions to be executed. The microcontroller's port must be configured, the memory address latched, the control line properly set and the data retrieved. If the result of the operation is supposed to be returned to the memory, a similar sequence must be carried out. Figure 11.1 shows a generic example, where 15 instruction cycles would be required to retrieve a data byte from an external memory.

In terms of hardware, additional components would be required in order to create the external buses. Such additional requirements, including the memory device, would probably double the cost of the monitoring module.

11.2.2 – Local Communication

This work has developed Monitoring Modules that will reside within a process. As such it needed the associated consideration of a communication method that was able to manage the Module-to-Module and the “external world” communication.

Communication is already of extreme importance from the viewpoint of data integration and assumes a special role in the system architecture proposed here. The use of multiple Monitoring Modules may be considered to meet different purposes. Cases may exist where a single Monitoring Module would not be powerful enough to implement the monitoring task requirements. In other cases the process signal sources might be dispersed within the process and may therefore be better handled by separate modules. In these and in many other cases, networking seemed the best alternative, considering the installation aspects and the technologies already available.

There is no widely accepted agreement as to the best networking technology currently available in the automation field, although industrial networks may be assumed to represent the best choice for such cases. There have been investigations that suggested the use of Local Area Networks (LAN), which are widely employed to interconnect PCs. These arguments are based on the fact that such networks have been very widely used and are therefore sufficiently tested and that there is an existing infrastructure in almost all application areas, therefore providing an alternative for the integration of

monitoring / automation and management systems. Ethernet (and Internet protocols) has been used to implement a distributed monitoring system [11.1]. However, it was considered that it could represent a heavy demand in terms of processing requirements for the distributed nodes. In an attempt to tackle these problems it was suggested in [11.2] that such requirements might be minimised by employing modern network switching technology, reducing the traffic handled by each specific node.

A sensible argument was presented by Thomas [11.3], who considered that communication requirements are different at specific levels. His argument was that at the sensor level very specific and focused messages (parameters measurements or process variables), rather than large quantities of data, will be exchanged, thus requiring “light” protocols (that do not require complex flow control and sequencing methods). At the upper layer, where information is normally provided in a very friendly format (including graphs), complex protocols might be required to provide a reliable service. Such aspects were considered in this research, resulting in the adoption of an industrial standard to interconnect the Monitoring Modules.

Recently, CAN bus has become popular in industrial applications. The main reasons for this are the openness of the standard, the number of available suppliers and also the reasonable simplicity of its implementation. In this particular research other justifications for the use of this networking method were the full compatibility with the PIC family of microcontrollers and the application requirements. The event based nature of the modelling method (and monitoring tasks) have found in the CAN protocol an adequate networking approach, reducing the system’s management requirements and simplifying the design. Since cost was a permanent concern during the investigation, such aspects assumed a great relevance.

The message priority based approach of the CAN protocol contributed to the simplifying of certain aspects of the implementation. For example the hardware-enabled message filtering methods of the CAN controller reduce the amount of communication related processing undertaken by each network node. In this respect perhaps a concern related to CAN might be the size of the protocol’s data field (8 bytes). Although the monitoring records issued by the system were relatively short, these normally exceed the maximum length and thus required more than one CAN

frame. A level of complexity was introduced in order to manage such messages, since it must be considered that more than one Monitoring Module might try to transmit simultaneously. This was addressed by the implementation of an application layer, together with a timeout mechanism.

It must be considered that a more appropriate approach might have been to use one of the existing CAN application protocols [11.4]. Although feasible, this would require a much greater development effort, which was not considered appropriate at this stage of the research. Nevertheless, it must be recognised that the possible use of a standard based application layer could result in some benefits. Existing CAN based installations could be shared, enabling some improvement of the monitoring method by employing data provided by “CAN enabled” transducers, already deployed. These benefits would also potentially be reflected in cost savings.

Some aspects of the network implementation represented a higher level of complexity, mainly due to the requirement for the SPI serial link to interface to the CAN controller. The dynamics and asynchronous nature of the data communication arising in a typical application may result in many events happening almost simultaneously and unpredictably. The management of a communication peripheral handling this over a serial link required a tight control of the associated events, to prevent deadlock situations in which microcontroller and bus controller lose synchronisation. Perhaps considering such complexity and also due to the growth in the use of CAN, the industry has recently begun to release microcontrollers with embedded CAN controllers. This is the case for the Microchip PIC18 family’s latest devices, which allow researchers to envisage further enhancements to the proposed Monitoring System.

The increased functionality is clearly yet another factor supporting the decision to adopt CAN since it will ensure the relevance and compatibility of the developed monitoring systems with current and future systems.

11.2.3 – Remote Communication

In deploying these process-based monitoring devices it must be recognised that the information they generate should be made as widely available as possible. Internet connectivity is becoming part of all new generation systems and therefore presented a natural way to integrate the monitoring results with other management applications. Research has been conducted considering the integration of industrial networks with the Internet. In [11.5] an approach was presented to enable remote management of networked transducers using CAN buses. Although interconnection was considered at different levels (even with the CAN nodes supporting the Internet TCP/IP set of protocols), a gateway device with more or less software capabilities was always required. In [11.6] a gateway, also acting as an Internet server to the industrial network, was the solution proposed. Again this solution considered remote management applications and utilised the Internet to retrieve control device information and to perform their configuration.

In considering the characteristics of the proposed system (low-cost) and consequently the resources available to each Monitoring Module, it was proposed that a single connectivity node should be used. This would be capable of providing Internet facilities (a requirement) to all Monitoring Modules deployed to the same monitoring task. Here a simple option, such as a PC based implementation, could be considered. However, in considering the low-cost aspect of the research, it seemed a good opportunity to assess the capabilities of a microcontroller for this purpose. So ready-to-use hardware (Microchip's Internet development kit) with full Ethernet capability was employed.

At this point some important considerations become necessary. The hardware design of the Internet development kit, although representing a useful tool for the understanding and testing of Internet based applications, lacked a very important feature for real-time applications such as network communication: a network controller interrupt capability. Interrupts become an important feature when trying to avoid unnecessary processing which may arise due to frequent status requests being directed towards the network processor. Rather than servicing the communication controller on demand (i.e. matched events), the microcontroller must poll the

communication device at regular intervals in order to detect any related event. It should also be remembered that network requests have to be handled as fast as possible, and that the microcontroller does not provide a huge memory buffer to queue such requests.

A consideration here is that microcontrollers (such as PICs) might not be the best alternative for such an implementation. They are not provided with an external address and appropriate data buses, besides other specific control signals that are common in microprocessors, to interface a general device such as a network controller or additional memory. However, a good hardware design can improve this situation considerably, as has been proven through some Internet implementations based on microcontrollers that are becoming available of which the Tini® module may be the most successful example [11.7].

The complexity of the challenge represented by the implementation of the Internet protocols was considered as an argument for employing a programming environment capable of naturally dealing with different data types and structures, apart from enabling a structured program organisation. The availability of “C” programming language compilers for PICs microcontrollers made it the natural choice for this purpose. It is important when adopting such an approach to consider potential hurdles. An investigation concerning the use of high-level programming tools with embedded processors [11.8] indicated that a “performance penalty” might be expected. Such an effect was observed to some extent in the implementation of the Connectivity Module. The overhead introduced by the compiler in order to deal with a number of different situations, such as microcontroller’s unsupported data types, affected the application considerably. As a simple example, suppose an operation needed to add two 16 bits (integer) variables and place the result in a third one. Although stated in a single “C” programming line ($\text{Var_A} = \text{Var_B} + \text{Var_C}$), this would require 14 instructions when compiled with Microchip C18® compiler. The same number, of microcontroller instructions was produced by the FED Wiz-C® compiler (employed in this work). Coding such an operation directly in PIC’s Assembler, it would be possible to implement it with 6 instructions. This is strong evidence that the final application requirements must be considered when making the choice of the programming tools that will be employed.

Consideration of such aspects also demanded an analysis of the real requirements of the application in order to avoid unnecessary complexity. The presence of short data records, asynchronous events and the single application characteristic of the Connectivity Module seemed to be sufficient arguments upon which to support the decision to base the entire record transmission process on the UDP protocol, rather than opting for the complexity of TCP. Such an approach has been shown to be an alternative in similar applications, where UDP was either employed as the only transport protocol provided (as in this research) [11.9] or else used as a combined method to reduce the use of TCP [11.2, 11.10].

Finally some consideration related to the implementation of the Connectivity Module may be appropriate. From a practical perspective the simplicity and support provided by a PC (computer) would probably represent the best alternative for the implementation of a connectivity module. However, if a dedicated hardware is a requirement (for reasons as the presented in this research), an appropriate design and adequate support for high-level programming languages should be provided to ensure that efficiency is not affected. On this basis, the PICDEM.net™ represented the right choice for the investigation of the Connectivity Module in this research, but would not be adequate for a final (commercial) application.

11.2.4 – Data Analysis and Integration

One of the fundamental issues related to monitoring systems is their capability of providing relevant information. Database systems, together with Internet technologies, can become an important part of any strategy to provide this information. The use of a Management Application within this Monitoring System was proposed to meet this function. Important aspects, such as DBMS access and data security, are normally supported by a wide range of PC based development tools and standards. Interfacing a DBMS directly from a microcontroller implementation was discussed in Chapter 7 (Section 7.4). The conclusion reached was that this would be a difficult and time-consuming software engineering task, almost certainly requiring further hardware resources. As a result, the low-cost features of this monitoring approach would be affected, and the added effort would be made with no guarantee of it supporting a long lasting solution (update difficulties, new DBMSs, etc).

In the examples provided in the previous chapters, the web based approach was employed to present the System monitoring results. This was not supposed to represent a complete solution, but was simply to show how easy it becomes to integrate the monitoring records, since they are made available at the right place and in an appropriate format. A web-based server approach, where the monitoring structure also integrates a front end could be employed. However in this case, although they might be easy to implement, applications would be required to query the remote monitoring system, in order to obtain the information of interest. A further complication would be introduced, since the server should also be capable of keeping a record of past events, increasing complexity and cost.

The architecture proposed in the research was intended to establish a balance between the levels of the Monitoring Systems in order to explore the best aspects at each specific level. Near to the process, intelligent Monitoring Modules are capable of detecting process events and can react in response to them with meaningful records. They can also be deployed accordingly to the application requirements, therefore keeping costs as low as possible. At the middle level, a connectivity element was provided with the facilities required to allow monitoring records to easily flow on the widely available network, thus helping to make remote monitoring feasible. Finally, at the upper layer, PC technology provides the best choice to analyse, process, integrate and present the monitoring results. In this way, the high-cost elements can be shared, thus resulting in a system that is accessible from many perspectives.

As a result of this work an innovative distributed monitoring system has been developed and demonstrated. This can form the basis of future research and support the evolution of the next generation of condition monitoring systems.

11.3 – Modelling Method Aspects

The cost benefits that might result from a monitoring system such as the one proposed here would be considerably compromised if an entire new software development were required for every application task. The use of a modelling technique that could

enable the easy representation of the monitoring task became a requirement to prevent such a drawback.

As introduced in previous chapters, Petri-nets had been mainly used for modelling and simulation of computational systems and were investigated as a monitoring method for sequential processes by the IPMM Centre [11.11], resulting in a PC based software tool developed for this purpose.

The Petri-net concept seemed to represent a good method that could possibly be employed in a microcontroller based implementation. Although supported by a mathematical formalism, Petri-nets are mainly considered to be useful in monitoring tasks due to being able to logically represent functions whilst allowing a direct graphical representation of the main elements of the function.

In choosing the Petri-net method as the modelling approach in this work, transitions became the natural option to define the Petri-net structure, since they can be associated with the process events and represented them as individual and isolated elements that relate input conditions to output actions. Such a characteristic eases the modelling task, since it enables an approach that describes the monitoring Petri-net as a set of self-contained elements that do not depend on each other. Such elements also define a static structure (process definition) and therefore can be placed in the microcontroller program memory, releasing the data memory and thus helping to tackle one of the microcontroller constraints. Efficiency is not affected, since the microcontroller's instruction set fully supports such an approach.

A drawback of this approach might be considered to be the fact that the Petri-net data structure must be stored within the program memory, along with the operational software, rather than being loaded dynamically. It was considered at this stage that technology improvements would soon provide an alternative. Such an assumption was proven to be true, since latest microcontroller releases (not employed in the research) are now provided with flash program memory, thus enabling the program memory to be updated at execution time. Also, new devices are becoming available with larger amounts of data memory, suggesting further alternatives in this direction. In all cases

the methods used in this research may be easily adapted to take advantage of these new innovations.

The definition of the system's overall capabilities, in terms of the largest Petri-net implementation, can be considered on the basis of the available resources and its efficiency. An initial hypothesis was to base such a definition on the microcontroller's internal data bus width. By proposing that the processing unit is usually optimised to its natural data type (8 bits for the PIC18C452), it was supposed that a more efficient execution would result. Therefore, transitions, places, tokens and sub-nets were constrained to a range from 0 to 255. The demand for control elements within the transition data structure imposed further limits on transitions (1 to 254), places (0 to 254) and sub-nets (1 – 255). Although these might seem somewhat limiting, the examples presented in previous chapters showed that relatively complicated Petri-net designs can be supported with such numbers. It is perhaps possible that a reduction in these numbers could allow more features to be added to the system, which might be a point of future consideration.

An important characteristic of this Petri-net approach is the fact that it enables the implementation of the distributed concept in the monitoring environment by providing each Monitoring Module with an amount of local knowledge. Rather than monitoring signal changes and forwarding them to some form of central processing units for analysis, the Monitoring Modules were provided with the intelligence required to detect process events and states, which can be made public when required. Bolic et al [11.12] presented a measurement system, based on microcontrollers, where the distributed nodes are configured from a central node by selecting the specific functions to be performed. The implementation of such an approach required the existence of specific libraries for each function. The Petri-net method however supports a generic approach in which the distributed elements (Monitoring Modules) become autonomous. This is seen as an important attribute of the developed system.

An important aspect in any distributed environment is the capability of the processing elements to produce information that can be used by consumers (other processing units) accordingly their requirements. Such a method in a client-server configuration is referred to as “publish-subscribe” [11.1]. From this perspective, the Petri-net

approach developed in this research implements a “producer-consumer” method, where one Monitoring Module makes specific events public (by means of sub-net broadcasts) and the other Modules utilise such information only if they require it in their particular domain. Using the publish-subscribe method might improve the system’s reliability, with sub-net events reported to a central node (server) that would replicate it to all those nodes that subscribed to the specific event.

Despite the fact that “ordinary transitions” and places allowed the representation of a sequence of events of a process, it soon became clear that these were not enough to model all real situations. The first function that needed greater support arose when trying to model events that depended on process signals having slow responses. A “delay transition” was developed to meet this need. A few examples of “delay transitions” were presented in the previous chapters (application examples). In one particular case shown in Chapter 10 (Figure 10.5), the “delay transition” was of extreme importance in overcoming the difficulty in identifying whether or not a tool was already placed in the carousel changing position. An extended use of the “delay transition” was illustrated in Chapter 9, shown in Figure 9.5. To enable up to date process information, a “delay transition” was inserted as a means of providing a record to the database after every 60 seconds. This element considerably increased the modelling flexibility by supporting time-dependant instructions and functions.

Although process actions are normally associated with digital signals and thus represent the main source of evidence of process events, analogue signals are often another important source for monitoring systems. With respect to the modelling approach developed here, “analogue transitions” were introduced to monitor such signals. This approach was developed on the basis of thresholds that identify specific and pre-established conditions. Thresholds are one of the simplest methods employed in condition monitoring, and are used in many system implementations to detect the indication of an abnormal condition [11.13, 11.14]. Although the identification of a fault is in many cases not so simple, it might be used to watch the behaviour of critical parts, as was shown with the example in Chapter 8. In that case, threshold conditions were defined to observe the motor’s operating limits. The records that the firing of such transition provides might be used to trigger maintenance actions. In another example, reported in Chapter 10, an “analogue transition” provided an efficient

method of detecting the tool cutting condition. This is a typical case where digital signals do not provide enough information. The design and use of analogue transitions in this research is an important innovation and allowed important added functionality to the monitoring approach.

Another structure introduced as part of this research was the “output transition”. Although this represents a very simple element, it becomes very useful in critical situations, such as described in the example in Chapter 9, where an unmanned process may suddenly start to produce a high number of low-quality parts. This was missed by the control system, but using an output transition the Monitoring Module is able to produce a local alarm. However, it is assumed in this work that the Monitoring Modules should not act directly on control system functions. This is because such actions could potentially produce damaging situations and, in extreme cases, great danger. In such cases, the records produced by the monitoring system should be used by supervisory systems or operators to intervene in a safe way. The fact that this may be managed locally using the distributed Monitoring Module is an important feature. So too is the fact that any recovery actions taken locally may be directed and monitored remotely, allowing centrally located “experts” to oversee operator centred activities.

Although significant developments have been made there are still some areas of potential further research. For example, it became apparent that an important modelling element is missing in this proposition. Such an element was not defined in the original Petri-net theory, although it has been proposed as an extension to the theory and called an “inhibitor arc” (Chapter 5, Section 5.5). To illustrate the use of such an element as a modelling resource, Figure 11.2 repeats the Petri-net branch presented earlier in Figure 9.6, showing how it could be modelled if a method to test an empty place, in order to identify a true condition, were available. The existence of such an element would simplify the modelling task in several situations, requiring fewer elements and therefore allowing a clearer representation. In this research, an approach to meet this need was introduced, by enabling the process signals’ “false” condition (digital “0”) to be tested in the transition firing process. This met the requirements of the systems modelled in this research, however, the same capability to test places is missing and certainly should be considered in a future development.

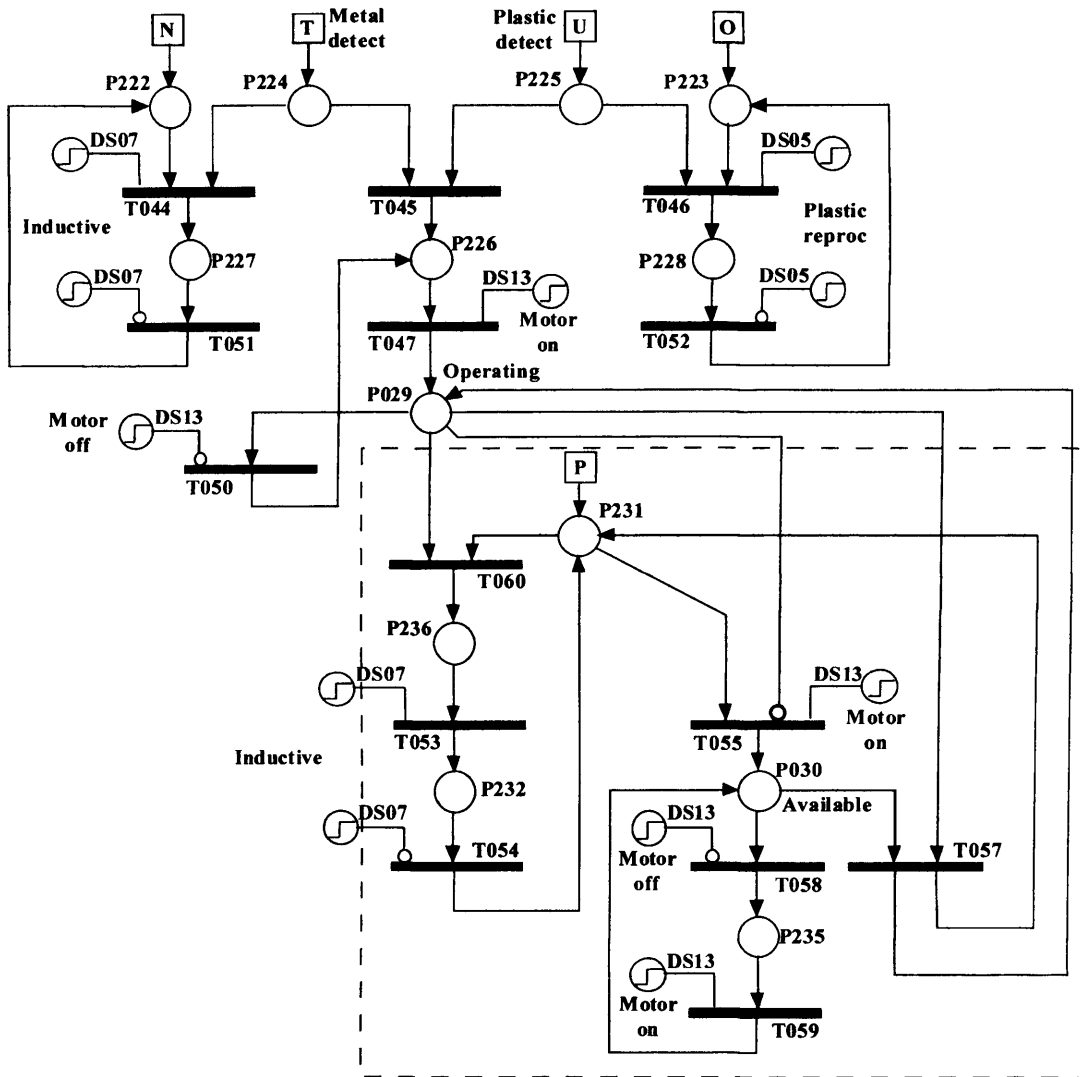


Figure 11.2: Simplification that would result from an empty place test element (arc between P029 and T055), when compared to Figure 9.6.

Despite the absence of an inhibitor arc, the Petri-net approach that was developed allows a great level of flexibility to model a wide range of applications. The examples used in previous chapters to demonstrate the approach were purposely selected to present different requirements. They illustrated the level of freedom that is allowed to the application designer in order to deliver the required solution. Furthermore, the Petri-net approach represents a method that allows a degree of independence from the microcontroller system used and thus, besides easing the modelling activity, does not limit its use to a single generation of such devices.

11.3.1 – Extended Features Supported by the Petri-net Approach

There is no doubt that Petri-nets can be used to model the sequence of events related to a process. They can thus provide relevant records that can be used to provide process related information of interest, as shown in the 3 example applications. An important contribution of Petri-nets is the fact that they provide a simple way to model the knowledge required to support more intelligent monitoring tasks. This facility was further explored in this research.

Hu et al [11.15] argued that many PLC controlled manufacturing systems operate in a predictable way, both in terms of the sequence of actions they manage and the time intervals between them. They concluded that since having such knowledge, operational faults could be detected. In the Petri-net approach developed by the IPMM Centre, a method that makes similar assumptions was considered [11.11], in which the Petri-net's selected transitions may have a timeout parameter configured. This allowed the detection of a transition that had failed to fire due to the absence of a process signal.

Such aspects were considered in this research. A timeout feature was integrated within the Monitoring Modules. This considers places rather than transitions, as was previously the case. This approach was based on two reasons. The first is because places can be associated to process states and these are the ones that consume time (between events) and hence a direct measure of process performance may be made. Secondly, a timeout related to a place requires predictable resources, since a state is a fixed entity in the system, while transitions may be enabled by tokens following different routes which can introduce different time constants. In terms of the PIC based implementation this would represent an additional level of complexity, which, considering the microcontroller's resources, may not be feasible.

The timeout represents an important feature that enables the Monitoring Modules to “understand and judge” process dynamics. The default method considers the active state of the previously monitored cycles. Here it must be recognised that improvements may be required, since it currently considers a single value of the time constant rather than a more realistic range. Such an improvement would require more

resources (which were not available at present) in the resulting implementation. Redefining the capabilities of the Monitoring Modules in terms of maximum number of places and transitions might be an alternative to “find” such resources. Considering the test carried out during the research, this might be reasonable. Another point to consider is the smallest timeout unit, presently defined as 1 second. This satisfied the main implementation tested. However, cases may exist where a smaller unit might be required. In consideration of all of the above the approach taken to this research was to build-in the potential for developing the methods as the new technology allowed these functions to be supported becomes available.

The flexibility allowed by the modelling method is directly associated to the fact that Petri-net elements (places and transitions) can be deployed over a wide range of process or systems without major restrictions, in order to model a specific task. In such environments there will be instances when the timeout feature could become undesirable. The examples reported here in this work illustrated several cases where places were not related to any process state, but rather were used as a modelling element to synchronise or control the Petri-net execution. In such cases an avalanche of meaningless “timeout” records could be generated. To prevent this a “barrier” was introduced into the parameter identification: places with an identification up to this parameter are timeout enabled, whereas those above it are not. Both types of places are important in modelling processes and in providing relevant information.

The research identified one further opportunity to enhance the capabilities of the Petri-net approach. The review presented in Chapter 3 showed that many condition monitoring applications needed to monitor analogue signals, in order to extract specific features that can help in detecting or predicting the development of critical conditions. Such signals are of particular interest under certain operating conditions, normally associated to a specific process state. The knowledge incorporated within the Petri-net model enables to detect such states and therefore provides a method that can be used to easily trigger (and stop) signal and data acquisition. In this way condition monitoring techniques may be focussed and managed to act only when required to do so, representing a considerable saving in the amount of data acquired and processed.

An example of this was presented in Chapter 8. The press rig motor current should sensibly only be monitored while the pallet is moving. Instead of sending every single sample to the database, the Monitoring Module provided a signal parameter based on the mean value calculation. There is no suggestion that this example is using the most appropriate condition monitoring technique to diagnose motor faults, which is understood as a very complex subject. The aim was to illustrate the potential of this approach to generic monitoring applications. While being potentially the basis of future intelligent monitoring systems it should be noted that at this time attempting the Petri-net execution with the calculation of complex signal features on the same microcontroller may not be feasible in all cases.

The anticipated further development of a new generation of processors with special signal analysis features (such as DSPs) may allow future researchers to use a mixture of techniques to monitor a wider range of applications. The architecture proposed in this research provides a framework to support such an approach and shows that it can work. Monitoring Modules with different capabilities or features could coexist within the system, with the Petri-net based models commanding the acquisition, processing and analysis of process parameters. Recent research conducted in the IPMM Centre has started to investigate more advanced signal analysis methods based upon the microcontroller implementations developed in this work. [11.16].

11.3.2 – Fault Diagnostics

Fault diagnostics is an important feature of any monitoring system. It is normally provided in order to help reduce the time required to recover manufacturing systems from faulty states. In research that considered such a method [11.17] it was suggested that operational fault diagnostics could be based on a map of the process signals and active states (in the event of a fault), using such information as inputs to an expert system.

The Petri-net model of a process represents the knowledge required for the investigation of operational faults. Once in operation, it contains the description of the sequence of events followed by the process and it retains this process related information by mapping the state of the process as a sequence of active Petri-net

places. Chapter 8 illustrated the fault diagnostic approach implemented as a result of this investigation. Although incapable of matching the exact cause of a fault in every single case, it is fully capable of rapidly presenting a set of hypothesis that could be used to further investigate such faults. Considering that such hypotheses will be available in a database, along with all previous process fault related conditions and associated actions, the method proposed in [11.17] could be further improved. By targeting the fault diagnosis towards the most probable faults the Monitoring System can help the expert system, which would be required to perform a smaller number of interactions in order to present the most probable fault source and as a result suggest repair actions.

The previously considered timeout method provides the diagnostic approach with an important feature: the generation of a fault symptom. To do this however, it is necessary to isolate the timeout events that may be related to faults and those resulting from process operating changes. This information must be communicated to a higher level (i.e. the database). Benefits from such an approach are the fact that information from other systems can be integrated (e.g. maintenance and planning systems). This information may then be used to plan subsequent action, and may also be interrogated to identify the timeout events that have resulted from process changes that have not been assimilated by the monitoring system yet. In this way the process based monitoring system can effectively be used to continuously update the knowledge concerning the process, and hence can continuously increase its effectiveness.

This is another example of the capability of this system that, although limited in resources, is capable of providing a wide range of interesting features. Many factors contribute to such a result, including the processing capability (and embedded resources) of the microcontroller and the application knowledge represented by the Petri-net. In deploying a structure that considers all such factors, the best of each System's elements could be explored.

11.4 – The Research in the Monitoring Context

In their proposal of a remote data acquisition system (reviewed in Chapter 3), Nieva and Wegmann [11.18] considered that a modern concept of such systems should support a wide range of management applications, rather than a single one. They indicated that the important requirements of such systems include their ability to: timestamp any record, provide the capability of reporting events, support the generation of quantitative (measurements) and qualitative (states) records, to detect any abnormal conditions and to perform measurements associated with specific events. All of these features are present at some level in the implementation that resulted from this research.

The examples presented in the previous chapters (8, 9 and 10) showed that the proposed Monitoring System has the particular characteristic of providing monitoring support to different applications without requiring the adjusting or adapting of application specific parts. This is an important aspect, since such generic characteristics demonstrate the flexibility of the approach to modelling applications that is fast and easy. In this way this work represents a method that can be widely employed. It also can be said to achieve the low-cost characteristic that was one of the initial aims of the research.

The generation of managerial information represents an important aspect of any system. The examples discussed earlier showed that the Monitoring System would be capable of providing direct information related to the manufacturing process, which could support management and production planning actions. At the same time, maintenance activities were also supported, and provided with the means to investigate events that might affect process efficiency and with the potential to limit and prevent excessive down time. Most importantly these different approaches can be enabled without any special requirements for many purposes.

The systems architecture played an important role in achieving the reported results. It supported the research that was undertaken to explore the best characteristics of each level, while making information available to the different managerial requirements. It

also fully supports the low-cost modular approach that allows individual systems to be built as required to suit specific applications.

Finally, an important consideration is the fact that the Petri-net approach provides each Monitoring Module with “independence”. This, together with the microcontroller’s local processing capability, and the embedded knowledge of the monitoring task, meets the requirement for an intelligent distributed system.

REFERENCES

- 11.1 **Manders, J., Barford, L.A. and Biswas, G.** An Approach for Fault Detection and Isolation in Dynamic Systems from Distributed Measurements. *IEEE Transaction on Instrumentation and Measurement*, 2002, 51(2), 235-240.
- 11.2 **Flammini, A., Ferrari, P., Sisinni, E., Marioli, D. and Taroni, A.** Sensor Integration in Industrial Environment: from Fieldbus to Web Sensors. *Computer Standards and Interfaces*, 2003, 25(2), 183-194.
- 11.3 **Thomas, G.** Ethernet, Arcnet and CAN – Proposed Network Hierarchy for Open Control. Contemporary Controls, Customer Support, Contemporary Controls Web Site, Available from: <http://www.ccontrols.com/whitepaper.htm>, [Accessed 13 September 2002].
- 11.4 CAN Protocols, CiA – CAN in Automation, CiA Web Site, Available from <http://www.can-cia.org> [Accessed 17 September 2003].
- 11.5 **Cena, G, Valenzano, A. and Vitturi, S.** Integrating Fieldbuses and Factory Intranets. *International Journal of Computer Manufacturing*, 2001, 14(1), 41-54.
- 11.6 **Payá, V.S, Oltra, J.M. and Ginés, E.U.** Remote Access to an Industrial Network MAP 3.0 Through Internet. Communication Department of Polytechnic University of Valencia, Available from: <http://casal.upc.es/~ieee/looking/sempere/Remote.html> [Accessed 19 December 2000].
- 11.7 **Eisenreich, D., DeMuth, B.** Designing Internet Embedded Devices. New York, USA: Newnes Elsevier Science, 2003.
- 11.8 **Chatzigeorgiou, A.** Performance and Power Evaluation of C++ Object-oriented Programming in Embedded Processors. *Information and Software Technology*, 2003, 45, 195-201.
- 11.9 **Al-Haibaibeh, A., Whitby, D. R., Parkin, R. M., Jackson, M. R., Mansi, M. and Coy, J.** The Development of an Internet-based Mechatronic System for Remote Diagnostic of Machinery Using Embedded Sensors. *In Proceeding: ICOM 2003 - International Conference on Mechatronics*, 18-20 June, Loughborough – UK, 2003, 297-302.

- 11.10 **Kurihara, S., Tsurusaki, T., Ohtsuka, S., Hashimoto, Y., Higashi, S. and Hikita, M.** Construction of Remote Monitoring System for Separative Measurement of Leakage Current of Outdoor Insulators. *In Proceedings: 7th IEEE International Conference on Properties and Applications of Dielectric Materials*. Nagoya – Japan, 1 – 5 June, 2003, 401-404.
- 11.11 **Prickett, P.** A Petri-net Based Machine Tool Maintenance Management System. *Industrial Management and Data Systems*, 1997, 97(4), 143-149.
- 11.12 **Bolic, M., Drndarevic, V. and Samardzic, B.** Distributed Measurement and Control System Based on Microcontrollers with Automatic Program Generation. *Sensors and Actuators A*, 2001, 90, 215-221.
- 11.13 **Roberts, C., Dassanayake, N., Lehrasab, N. and Googman, C.J.** Distributed Quantitative and Qualitative Fault Diagnosis: Railway Junction Case Study. *Control Engineering Practice*, 2002, 10, 419-429.
- 11.14 **Baccigalupi, A., Bernieri, A. and Pietrosanto, A.** A Digital-Signal_processor-Based Measurement System for On-Line Fault Detection. *IEEE Transaction on Instrumentation and Measurement*, 1997, 46(3), 731-736.
- 11.15 **Hu, W., Starr, A.G., Zhou, Z. and Leung, A.Y.T.** A Systematic Approach to Integrated Diagnosis of Flexible Manufacturing Systems. *International Journal of Machine Tools and Manufacture*, 2000, 40, 1587-1602.
- 11.16 **Amer, W., Ahsan, Q., Grosvenor, R.I., Jennings, A.D. and Prickett, P.W.** PIC Micro-controller Based Machine Tool Monitoring System. *In Proceedings: COMADEM 2003 – 16th Conference on Condition Monitoring and Diagnostic Engineering Management*, 27-29 August, Växjö – Sweden, 2003, 219-225.
- 11.17 **Hu, W., Starr, A.G. nad Leung, A.Y.T.** Operational Fault Diagnosis of Manufacturing Systems. *Journal of Material Processing Technology*, 2003, 133, 108-117.
- 11.18 **Nieva, T. and Wegmann, A.** A Conceptual Model for Remote Data Acquisition Systems. *Computers in Industry*, 2002, 47, 215-237.

CHAPTER 12

CONCLUSION AND FUTURE WORK

12.1 – Main Contributions of the Research

This research focused on the development of intelligent, distributed and low-cost monitoring systems. It investigated a range of different technologies, models and methods and lead to the following important contributions:

- The development of a flexible hardware / software architecture that is capable of supporting a wide range of generic applications;
- The development of a microcontroller-based Petri-net model that provides hardware and software independence;
- The novel incorporation of analogue signals within Petri-net models thus enhancing the use of the Petri-net methodology for monitoring purposes;
- The development of a method that enables the use of Petri-net places to retrieve process specific active state information from remote processes;
- The development of a method that enables the use of Petri-net places to control the acquisition of selected process parameters which can then be used for more detailed, off-line fault diagnosis.

In each case the efficacy of the approach and the flexibility of the systems produced by this research has been demonstrated with the deployment of the resulting systems within a range of different processes.

12.2 – Conclusions

The important conclusions that can be drawn as result of this investigation can be summarised as:

- The PIC18C452 microcontroller represents a reliable and flexible device for the implementation of intelligent, distributed and low-cost monitoring systems;
- Low-cost embedded Internet technology is available and can be used in monitoring applications. However, factors such as hardware design, programming languages and implementation simplifications must be considered in order to achieve better results;
- The distributed concept allows a designer to access the best features of each component of the system, increasing processing capabilities while keeping cost in proportion;
- The Petri-net modelling approach aggregates flexibility to the microcontroller implementation, easing the monitoring task representation and reducing development time;
- Petri-nets provide the elements required for the implementation of more complicated monitoring tasks, representing a simple way to model time measurements and to trigger the acquisition of analogue signals.

Considering each of these points in more detail, it can be said that the PIC18C452 microcontroller fulfils the requirements for the implementation of low-cost monitoring systems. The device is equipped with the input functions and embedded features required to support such an application. It must be clearly recognised that this is not a reference to a data acquisition system, but to a monitoring implementation that can be used to process data retrieved from a process and to convert it into information. The system produced is still low-cost because very few external components were required to enable the microcontroller to operate.

It can also be stated that, for this particular and similar applications, based on this type of devices, Assembler is still the best programming language, since it allows designers to directly and efficiently manipulate limited resources. In the same context, it was concluded that the best use of the microcontroller (already defined as a computer-in-a-chip) is based on using its embedded resources. It is on this basis that PICs perform the best. The use of this microcontroller in the Internet implementation demonstrated such an aspect, where a conjunction of factors such as external high-demanding devices (Ethernet), programming languages (C) and a restrictive hardware design (interrupts) combined to affect the microcontroller's performance.

The System architecture played an important role in achieving the final results. CAN bus protocols form a good partnership with PIC microcontrollers and are conceptually adequate for monitoring applications (enabling distributed nodes to transmit messages triggered by events). The distributed philosophy of this architecture represents an appropriate choice to achieve the initial proposition. By exploiting the best functions of each of its components it provides an excellent methodology to improve results and reduce cost.

The Petri-net approach has been developed and improved in this research, and has been shown to work well as a modelling method for monitoring purposes. It was initially proposed as an alternative to provide a low-cost microcontroller based implementation with a simple modelling method and resulted, in the end, in a powerful method that is capable of adding intelligence and flexibility to the System. It is simple and practical and is shown throughout this thesis to be capable of allowing engineers to represent different situations. Three examples with different characteristics, one of them a real manufacturing application, were modelled without major constraints, producing a wide range of useful management information.

As a consequence of this research Petri-nets may now assume a new profile. They have been shown to be much more than a way to describe and follow events. The Petri-net approach resulting from this research can be used to enable a simple microcontroller to be deployed to easily obtain process status information and to

trigger the acquisition of important parameters. They have also been developed to support a simple method to help in the detection of operational faults.

An important consideration (as a matter of evolution) is the fact that technology will still improve. For example, during the course of this research, new devices became available. Hence a method bound to a specific technology would struggle to find widespread acceptance. The Petri-net monitoring approach, as defined in this research, does not depend on a single processor or supplier. It can be used with almost any processor, and as device capabilities increase the method itself can be further investigated and enhanced. The PIC microcontroller became a good choice for this investigation and undoubtedly still represents a low-cost alternative, but certainly (and hopefully) it will not be the last.

Based on the results obtained, it is possible to conclude that the System proposed through this research is capable of providing a low-cost alternative for process and condition monitoring. It fulfils the main requirements by providing information for managerial and maintenance activities. This information can be made available on the Internet or can be integrated in other existing management applications. In considering the complexity of some condition monitoring requirements, the System can be used as a qualitative method. In this context it can be used in detecting suspicious conditions that may be further investigated using traditional condition monitoring techniques, therefore helping to reduce the high cost that the use of this technology normally represents. The proposed System is capable of providing the required intelligence to support this approach.

12.3 – Future Work

In considering the potential of the final version of the Monitoring System, it becomes evident that a further refinement would be the development of a more user-friendly computer based application to enhance the Petri-net designing task. In the same context the use of flash based microcontrollers should be investigated and developed, to allow the dynamic downloading of the monitoring task into the distributed Monitoring Modules.

Another possible step forward in the System implementation would be the provision of the capability to set up remote monitoring modules from a standard web page which is interacting with the System's Management Application. To facilitate this the Monitoring Modules should have the CAN bus application layer improved perhaps based on one of the existing standards. This would also enable Monitoring Modules to explore smart sensors deployed on the same CAN bus, rather than only those sensors and signals directly interfaced by each Module. This would also be an important step towards integrating specialised modules, which may be provided with analogue signal processing and analysis capabilities.

Another interesting development would be the use of artificial intelligence, probably an expert system, which is capable of making improved use of the records produced by the Monitoring System. This would provide an entire solution to extended and support condition monitoring and maintenance activities. Also, during the completion of this research, it was noticed that possible variations in process dynamics represent a major obstacle to the implementation of fully reliable and easy to use condition monitoring strategies. Therefore, it would be interesting if a system could be developed that is capable of automatically retrieving process information and based on this, generating a set up of specific parameters (that might be affected by process settings) within the deployed Monitoring Modules. This would make the System capable of detecting abnormal conditions more effectively and efficiently.

All of these future developments are possible to predict. They will continue to build upon this research which has for the first time shown that real progress can be made in this important sector with the bringing together of the separate Petri-net and PIC based technologies to form a flexible and low-cost solution to the problem of process monitoring. This represents a major step forward.

APPENDIX A

DEVELOPMENT RELATED DETAILS

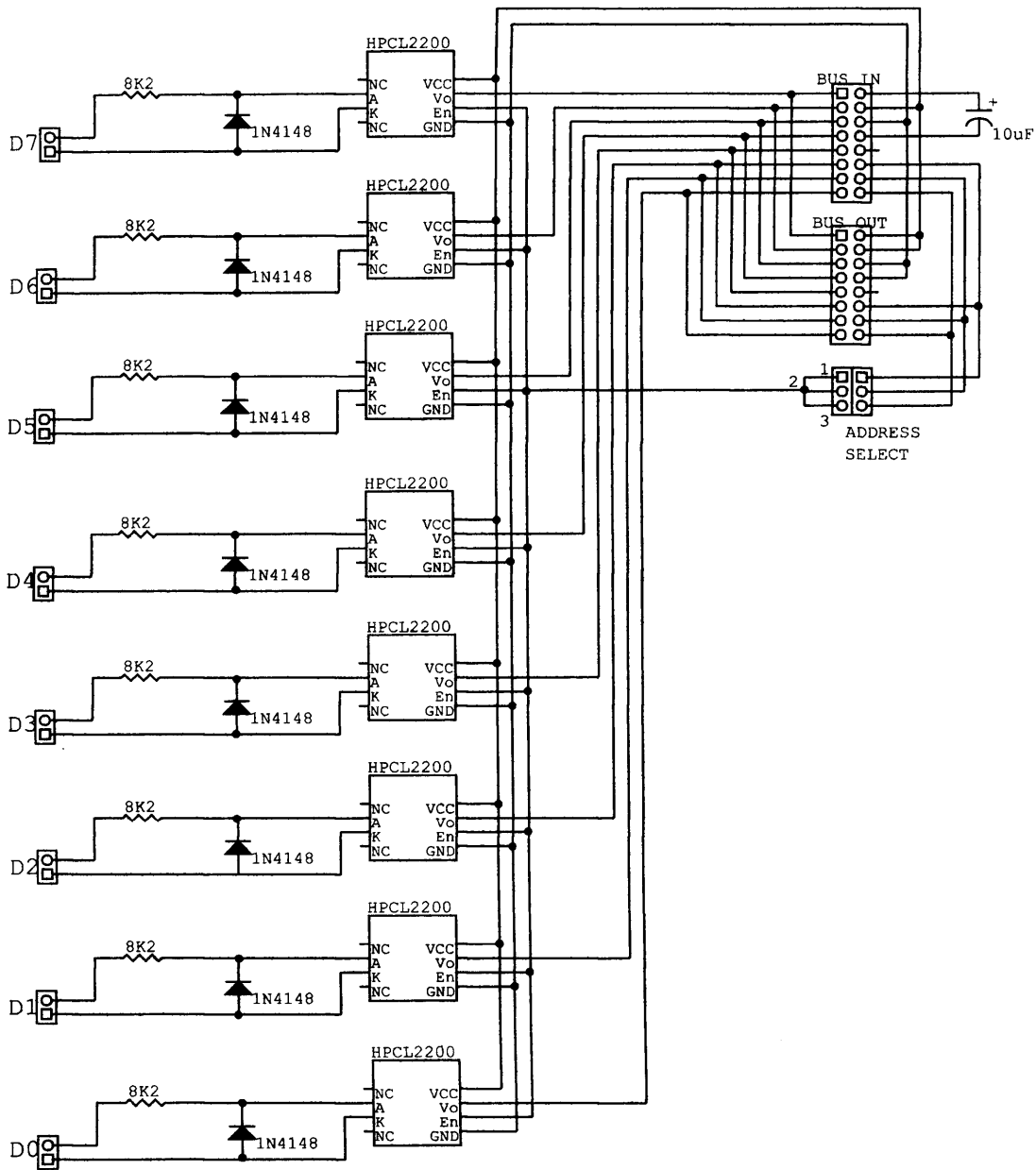


Figure A.1: Digital input card circuit diagram.

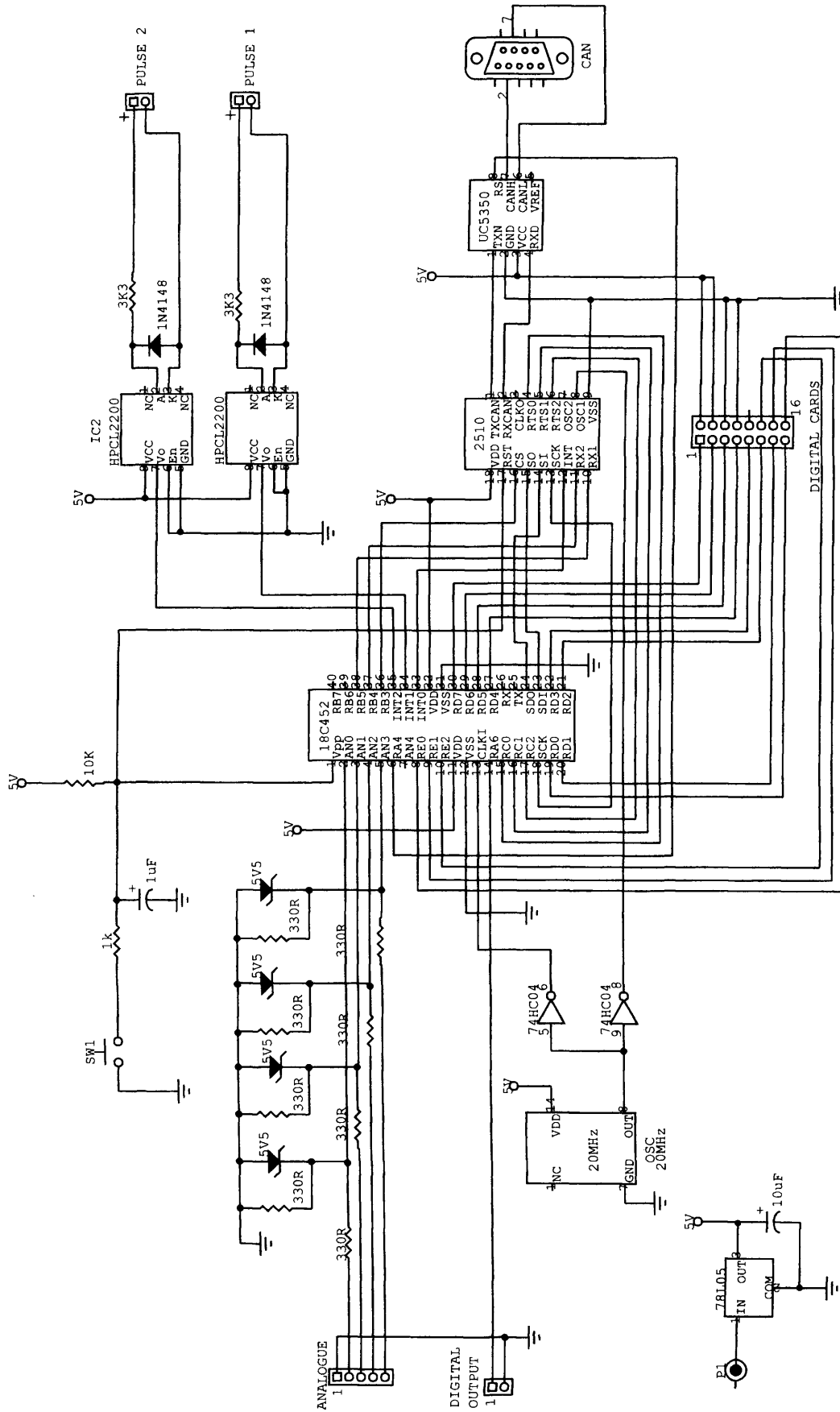


Figure A.2: Microcontroller module circuit diagram.

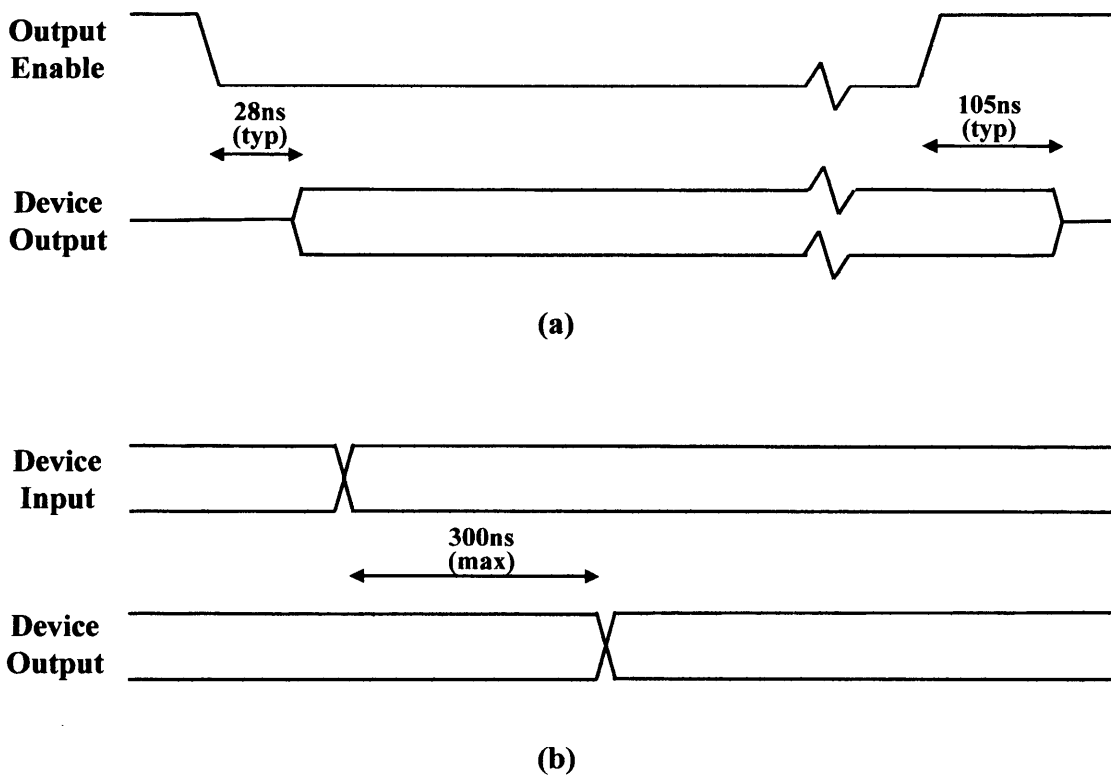


Figure A.3: Digital inputs card timing diagram, (a) the output with respect to the output enable signal and (b) the output with respect the input.

Monitoring Module Software Development

```

;*****
;* PhD Research
;* Intelligent Distributed Monitoring System
;* System Engineering Division
;* Student: Marcos R. Frankowiak
;* Supervisor: Paul W. Prickett / Roger I. Grosvenor
;*****
;* TITLE: PICNET.ASM
;* DESCRIPTION: PIC based petri-net running on a PIC 18C452 microcontroller
;* STARTING DATE: 24/09/2000
;* LAST UPDATE: 09/03/2003 (fault diagnostics)
;*****

```

```

list p=18C452 ;select microcontroller

```

```

;include files

```

```

include "p18C452.inc" ;microcontrollers header
include "mcp2510.inc" ;CAN microcontroller header
include "ascii.inc" ;ascii table
include "PicDef.inc" ;pic related definitions
include "IntMacro.inc" ;specific macros for the application
include "IntVar.inc" ;defined variables and constants

```

```

;
;cold start

```

```

ORG h'0000'

```

```

Start

```

```

NOP ;a string of "no operation instruction"
NOP ;to allow controller synchronisation
GOTO Main ;go to program execution

```

```

;
;interrupt vector addresses

```

```

;WREG, STATUS and BSR are automatically stacked up by the microcontroller

```

```

ORG h'0008' ;no priority scheme selected

```

```

SysInterrupt

```

```

BTFSC PIR1,TMR1IF,A
CALL InterrTimer ;real time update interrupt
BTFSC INTCON,TMR0IF,A
CALL InterrCANChSel ;CAN controller chip-select delay
BTFSC INTCON,RBIF,A
CALL InterrCANRx ;CAN controller RX interrupt
BTFSC INTCON3,INT1IF,A
CALL InterrPulse1 ;pulse input update
BTFSC INTCON3,INT2IF,A
CALL InterrPulse2 ;second pulse input update
BTFSC PIR1,SSPIF,A
CALL InterrSPI ;SPI interface interrupt
RETFIE FAST ;return from interrupt restoring PIC registers

```


Appendix A – Monitoring Module Code Main File

```
;  
;-----  
;main program  
  
Main  
    SystemInit                ;general system reset actions  
  
    Timer1Init                ;real time timer - updates an SQL datetime format  
                                ;variable (1 ms)  
    Timer0Init                ;configures timer used to generate a delay before  
                                ;removing the CAN controller chip select signal  
    SPIInit                   ;configures the SPI interface - data exchange with  
                                ;CAN controller  
    CANCtrlInit               ;MCP2510 - CAN controller operating mode  
  
    PetriNetInit              ;petri_net general initialisation  
  
    InterrConfig              ;configures and enables the interrupt operating mode  
  
;-----  
;application main loop  
  
ApplicLoop  
    ControllerReady           ;verifies whether or not the controller was initialised  
  
    WatchController           ;checks if CAN controller is requesting service  
  
    UpdateAnallInput          ;update the analogue inputs  
  
    UpdateDigInput            ;updates the digital inputs  
  
    OneSecondUpdate           ;time based variables update  
  
    CheckPetriNet             ;checks the Petri-net table for transition firing  
                                ;conditions  
    TransitionMsg             ;verifies if a transition was fired  
  
    BuildMessage              ;builds a message indicating that a transition was  
                                ;fired  
    LoadSpiBuffer            ;if a message was built, it must be loaded in the SPI  
                                ;transmit buffer  
    CheckCommand              ;verifies whether or not a command was received  
  
    BuildCanStatusReq         ;a message is loaded in the SPI transmit buffer to  
                                ;request the CAN controller status  
    CheckSpiBuffer            ;checks if there is any block of data downloaded from  
                                ;the CAN controller  
    BuildCanBufferReq         ;a message is loaded in the SPI transmit buffer to  
                                ;request the CAN controller receive buffer content  
    BuildCanAck               ;a message is loaded in the SPI transmit buffer to  
                                ;acknowledge a CAN controller status indication  
    SpiStartTx                ;begins the transmission of a new message in the SPI  
                                ;transmit buffer  
    ProcessUpdate             ;checks whether or not a place changed state  
  
    TimeOutUpdate             ;controls message exchange timeout  
  
    FaultDiagRequest          ;verifies if any fault diagnostic is requested  
  
    ProcFaultDiag             ;executes fault diagnostic
```

Appendix A – Monitoring Module Code Main File

```
RestartPNet                ;restarts the Petri-net if a token arrives the bin
;
;-----
;specific routine include files
        include    "intsub.inc"    ;system subroutines
        include    "interr.inc"    ;interrupt routines
;
;-----
;Petri-net required tables
;transition table
TRANSTABLE:
        include    "petrinet.inc"  ;Petri-net table
PROCESSTAB:
        include    "process.inc"   ;special states (watch status) table
WATCHTABLE:
        include    "watch.inc"     ;special parameters – analogue acquisition table
;
;-----
;program end
        END
```

The entire set of development files related to the Monitoring Module are provided in electronic format and can be found in the attached CD-ROM. Further details are available in the Section “Attached Documents and Files”, later in this appendix (A.25-1).

Connectivity Module Software Development – CAN Node

```

.*****
;* PhD Research *
;* Intelligent Distributed Monitoring System *
;* System Engineering Division *
;* Student: Marcos R. Frankowiak *
;* Supervisor: Paul W. Prickett / Roger I. Grosvenor *
.*****
;* TITTLE: CAN_MN.ASM *
;* DESCRIPTION: Connectivity Module – CAN node implementation (PIC18C452) *
;* STARTING DATE: 24/03/2001 *
;* LAST UPDATE: 27/10/2002 (timeout control update) *
.*****

```

```

include "p18c452.inc" ;microcontrollers header
include "mcp2510.inc" ;CAN microcontroller header
include "can_var.inc" ;application variables definition
include "can_cfg.inc" ;microcontroller configuration registers set up
include "can_mac.inc" ;defined macros in use by the application
include "ascii.inc" ;ASCII characters definitions

```

;application execution – beginning point

```

COLD_START ;microcontroller power-on reset

INTERRUPT_VECTOR ;microcontroller interrupt access vectors

```

;general set up

MAIN

```

INIT_PIC_REGISTERS ;sets microcontroller configuration registers

SET_STACK_POINTER HIGH_STACK,LOW_STACK – 1 ;defines software stack-pointer

SET_UP_BUFFERS ;sets up all registers used by the application

INIT_TIMER0 ;initialises timer to perform CAN chip-select delay

INIT_SPI_INTERFACE ;sets up SPI interface operating mode

INIT_USART ;sets up the operating mode of the RS232 interface

INIT_CAN_CONTROLLER ;sets up a sequence of data in the SPI TX buffer that will
;initialise the CAN controller

INIT_TIMER1 ;sets a device to deliver 1 ms timing

ENABLE_INTERRUPT ;enables peripheral interrupts

SOFTWARE_DELAY CTE1 ;performs a 2 seconds delay before reaching the main loop

```

Appendix A – Connectivity Module Code Main File: CAN Node

```
;main execution loop
MAIN_LOOP
    CHECK_CAN_INT           ;checks if the CAN controller generated an interrupt
    CHECK_CAN_STATUS       ;verifies whether or not to request CAN controller status
    START_SPI_TX           ;checks if there is any block of data which demands
                          ;transmission to be started
    CHECK_RS_REC           ;checks if there is a data block in RS receive buffer
    CHECK_SPI_MSG          ;checks if there is a message in the intermediate buffer to be
                          ;send through the SPI interface
    CHECK_RS_MSG           ;checks if there is a message in the intermediate buffer to be
                          ;send through the RS interface
    CHECK_CAN_RTS          ;checks if any of the CAN controllers transmit buffers is
                          ;awaiting a RTS command
    CHECK_SPI_REC          ;checks if there is a block of data in SPI RX buffer ready to
                          ;be transferred
    CHECK_CAN_RX           ;verifies if there is data available in the controller receive
                          ;buffers
    CHECK_CAN_ACK          ;checks if there is an acknowledgement to be send to the
                          ;CAN controller
    CHECK_ACK_REQ          ;monitoring module acknowledge is requested
    CHECK_TIMEOUTS         ;controls receive buffer usage - discharge timed out message
    BRA MAIN_LOOP          ;repeat loop

;sub-routines an interrupt service
    include    "can_sub.inc";sub-routines file
    include    "can_int.inc" ;interrupt routines file

;
;end of application
END
```

The entire set of files related to the Connectivity Module CAN bus development are provided in electronic format and can be found in the attached CD-ROM. Further details are available in the Section “Attached Documents and Files”, later in this appendix (A.25-2).

Connectivity Module Software Development – Internet Protocols

```

/*****
* Internet connectivity application
* Main module (PIC Wizard C compiler)
* Based on Microchip PICDEM NED demo board
* date: October 2001
* last update: November 2002
*****/

#include <P18C452.h> //microcontroller PIC 18C452 header file
#include "P18Cext.h" //microcontroller header file extension
#include "picnet.h" //application definitions and settings
#include "peripheral.h" //PIC18C452 hardware and peripherals
#include "support.h" //general functions
#include "application.h" //intermediate data layer
#include "lcd.h" //LCD display library
#include "ethernet.h" //ethernet interface library
#include "internet.h" //TCP/IP protocol related functions

//-----local functions prototype-----

void main(void);
void Interrupt(void);

//-----interrupt variables-----

//definitions to enable multi-use of C and Assembler

#asm
tx equ 0 ;int_flags bit 0
rx equ 1 ; bit 1
t0 equ 2 ; bit 2

RxIntSize equ d'100' ;initially defined as 30
Timer0Low equ d'236'

#asmend

struct
{
    byte tx:1; //USART TX interrupt flag
    byte rx:1; //USART RX interrupt flag
    byte t0:1; //TIMER0 interrupt flag
    byte :1;
    byte :1;
    byte :1;
    byte :1;
    byte :1;
}int_flags; //interrupt flags

#asm
tx_req equ 0 ;USART status variable bit 0
rx_err equ 4 ; bit 4
#asmend

struct
{
    byte tx_req:1; //USART TX request

```

Appendix A – Connectivity Module Code Main File: Internet Protocols

```

byte :1;
byte :1;
byte :1;
byte rx_err:1;           //USART RX error
byte :1;
byte :1;
byte :1;
}dev_status;           //activity status

byte tx_reg;           //temporary register - USART TX
byte in_indx;
byte out_indx;
byte rx_reg[RX_INT_SIZE]; // " " - USART RX

byte *ptr_rx_reg;

//byte system_status;

//interrupt backup procedure

byte bck_fsr0l;
byte bck_fsr0h;

//-----function declaration-----
//interrupt vector description - must be in the same file as main()

void Interrupt()
{
#asm

INT_INIT:
    MOVFF    FSR0L,bck_fsr0l;
    MOVFF    FSR0H,bck_fsr0h;

    BTFSS    PIE1,TXIE,0           ;is the USART TX interrupt enabled?
    BRA     INT_USART_RX
    BTFSS    PIR1,TXIF,0           ;is it an USART TX interrupt?
    BRA     INT_USART_RX
    MOVFF    dev_status,FSR0L
    BTFSS    FSR0L,tx_req,0
    BRA     DISABLE_TX
    MOVFF    tx_reg,TXREG
    BSF     TXSTA,TXEN,0           ;enable data transmission
    BCF     FSR0L,tx_req,0         ;system notification
    MOVFF    FSR0L,dev_status
    MOVFF    int_flags,FSR0L
    BSF     FSR0L,tx,0
    MOVFF    FSR0L,int_flags

    DISABLE_TX:
        BCF     PIE1,TXIE,0

INT_USART_RX:
    BTFSS    PIE1,RCIE,0           ;interrupt enabled?
    BRA     INT_TIMER0
    BTFSS    PIR1,RCIF,0           ;checks for RX interrupt
    BRA     INT_TIMER0
    BTFSC    RCSTA,OERR,0
        BRA     USART_RX_ERR
    BTFSS    RCSTA,FERR

```

Appendix A – Connectivity Module Code Main File: Internet Protocols

```
BRA STORE_RX_DATA

USART_RX_ERR:
    MOVF    RCREG,W,0        ;remove data
    MOVFF   dev_status,FSR0L
    BSF    FSR0L,rx_err
    MOVFF   FSR0L,dev_status
    BRA    INT_TIMER0

STORE_RX_DATA:
    MOVFF   in_indx,FSR0L
    MOVF    FSR0L,WREG,0     ;buffer offset index
    MOVFF   ptr_rx_reg, FSR0L ;buffer address
    MOVFF   ptr_rx_reg + 1, FSR0H
    MOVFF   RCREG,PLUSW0
    ADDLW   1                ;increment index
    MOVWF   FSR0L,0
    MOVLW   RxIntSize       ;buffer overrun position
    CPFSLT  FSR0L,0
    CLRF    FSR0L,0
    MOVFF   FSR0L,in_indx   ;store index
    MOVFF   int_flags,FSR0L
    BSF    FSR0L,rx,0
    MOVFF   FSR0L,int_flags

INT_TIMER0:
    BTFSS   INTCON,TMR0IE,0 ;interrupt enabled?
    BRA    INT_END
    BTFSS   INTCON,TMR0IF,0 ;interrupt matched?
    BRA    INT_END
    MOVLW   Timer0Low
    MOVWF   TMR0L,0         ;load time constant
    BCF    INTCON,TMR0IF,0 ;clear interrupt
    MOVFF   int_flags,FSR0L
    BSF    FSR0L,t0,0
    MOVFF   FSR0L,int_flags

INT_END:
    MOVFF   bck_fsr0l,FSR0L;
    MOVFF   bck_fsr0h,FSR0H;

#asmend
}

void main()
{
    byte system_status;

    //device set up

    in_indx = 0;           //rx interrupt buffer indices
    out_indx = 0;
    ptr_rx_reg = &rx_reg[0];

    pic_set_up();        //configures the microcontroller

    system_off;         //operating LED's switched off
    user1_off;
    user2_off;
```

Appendix A – Connectivity Module Code Main File: Internet Protocols

```
lcd_init(); //LCD display initialisation

//RS232 buffer initialisation

usart_buf_init(); //TX & RX buffer initialisation
usart_init(); //device operating mode set up

//other buffer initialisations

app_buffer_init(); //initialises intermediate buffers

//ethernet interface initialisation

system_status = eth_cold_start(); //performs the interface initialisation
if(system_status == ok)
{
    enable_pic_int(); //enable interrupt
    usart_rx_int(); //reception interrupt enabling
    timer0_init(); //1 millisecond interrupt
    init_internet(); //internet initialisation
    display_message("System Status:\n");
}

while(system_status == ok)
{
    #asm
    BTG PORTC,0,0;
    #asmend

    if(int_flags.t0)
    {
        timer_update();
        int_flags.t0 = false;
    }

    if(int_flags.tx) //transmission previously started
    {
        int_flags.tx = false;
        if(usart_tx(&tx_reg) == ok)
        {
            dev_status.tx_req = true;
            pie1.txie = true;
        }
    }
    else //start a new transmission
    {
        if(!dev_status.tx_req)
        {
            if(usart_tx(&tx_reg) == ok)
            {
                //status must be set before enabling the
                //interrupt
                dev_status.tx_req = true;
                pie1.txie = true;
                txsta.txen = true;
            }
        }
    }
}
```


Appendix A – Connectivity Module Code Main File: Internet Protocols

```
eth_virtual_int();

internet_status();           //performs internet protocols checking

eth_virtual_int();

ethernet_status();         //performs ethernet checking

eth_virtual_int();

app_keeping();             //checks the data exchange between buffer

eth_virtual_int();
}

display_message("Eth init error\n");
while(system_status != ok)
{
    delay_ms(10);
}
}

//test if there is received data in buffer

byte get_rx_buffer_data(byte *ptr_data)
{
    byte status;

    if(in_indx == out_indx)
        status = false;
    else
    {
        status = true;
        *ptr_data = rx_reg[out_indx];
        out_indx++;
        if(out_indx == RX_INT_SIZE)
            out_indx = 0;
    }

    return(status);
}
```

The entire set of files related to the Connectivity Module Internet protocols development are provided in electronic format and can be found in the attached CD-ROM. Further details are available in the Section “Attached Documents and Files”, later in this appendix (A.25-3).

Analogue and Pulse Inputs Test Measurement

The test procedure was based on the implementation of a test-Petri-net to trigger the acquisition of the 4 analogue channels and 2 pulse inputs, as shown in Figure A.4. Describing this Petri-net, each acquisition cycle was controlled by a digital signal (DS01). Analogue channels and pulse inputs acquisition period was controlled by a sub-net command (SN002), issued by a CAN node acting as a second monitoring module, emulated by an MCP2510 CAN Controller Development Kit. The kit software’s support allows building and sending CAN messages, such as the sub-net broadcast shown in Figure A.5. The test results were stored in a test-database to enable further analysis. Table A.1 summarises the equipments employed in the test environment. The Petri-net descriptive files (text) are found in the attached CD_ROM (A.25-5).

Table A.1: Measurement test equipment set up.

Equipment Description	Use Description
Stabilised power supply FARNELL E30/1, 0 to 30 V / 0.5 A – 0 to 15 V / 1.0 A	<ul style="list-style-type: none"> • Analogue input linearity measurement test.
Global Specialities Corporation 2001 function generator – TTL, sine and triangular output and offset adjust – frequency range from 1 Hz to 100 KHz	<ul style="list-style-type: none"> • Pulse input linearity measurement test; • Analogue and pulse input mean value assessment.
Black Star true RMS multimeter – 200 m to 2000 Volts	<ul style="list-style-type: none"> • Analogue inputs linearity test – DC measurement.
Global Specialities Corporation 5001 universal counter / timer – 0.01 to 10 seconds update rate	<ul style="list-style-type: none"> • Pulse input monitoring – linearity measurement test.
Digital storage oscilloscope OS4100 Gold Advance, 2 mV to 5 V / division – 10 MHz	<ul style="list-style-type: none"> • Periodic wave form pick and period measurement

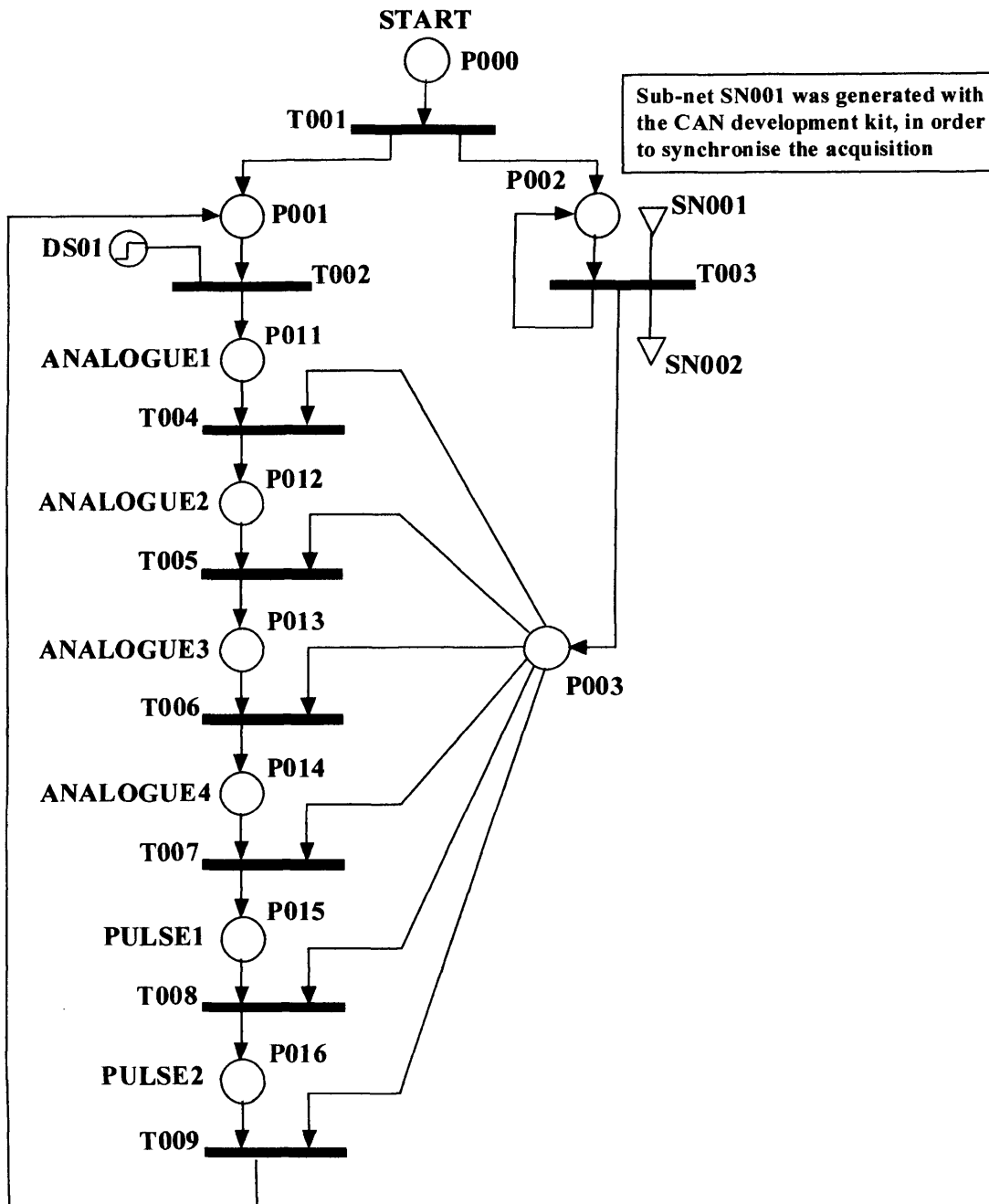


Figure A.4: Measurement test Petri-net.

The descriptive files related to the hardware test Petri-net can be found in the attached CD-ROM. Further details are available in the Section “Attached Documents and Files”, later in this appendix (A.25-5).

Appendix A – Measure

Universal page

CAN Envelope: \$7DF

Length: \$5

Send

Data 0 \$0

Data 1 \$1

Data 2 \$11

Data 3 \$0

Data 4 \$2F

Data 5 \$0

Data 6 \$45

Data 7 \$0

Number of bytes in the data field

Monitoring module ID (0 for broadcast)

Message ID –first part of a single part message

Separator “/”

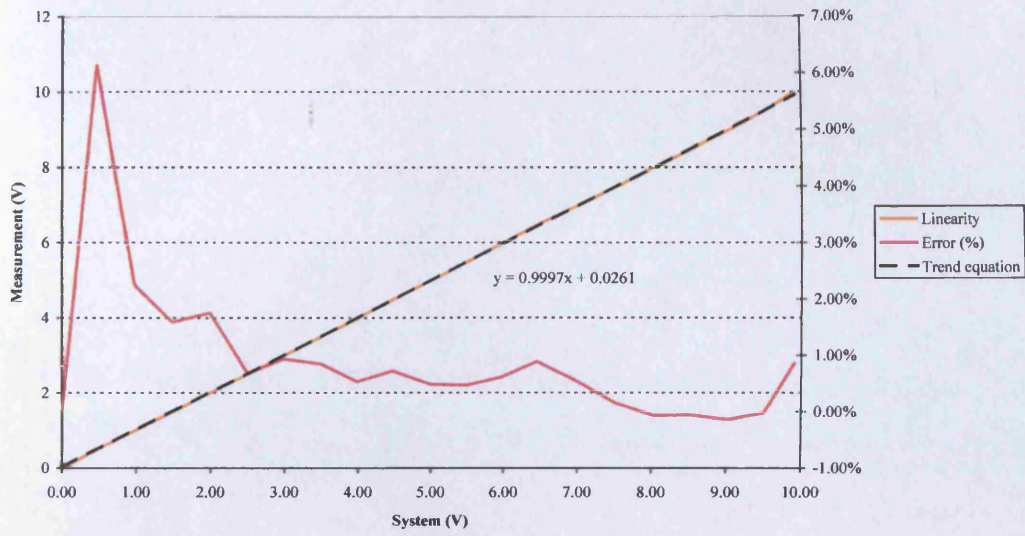
Sub-net sign “E” (together with envelope ID 7DF)

CAN frame identification (priority)

Sub-net identification

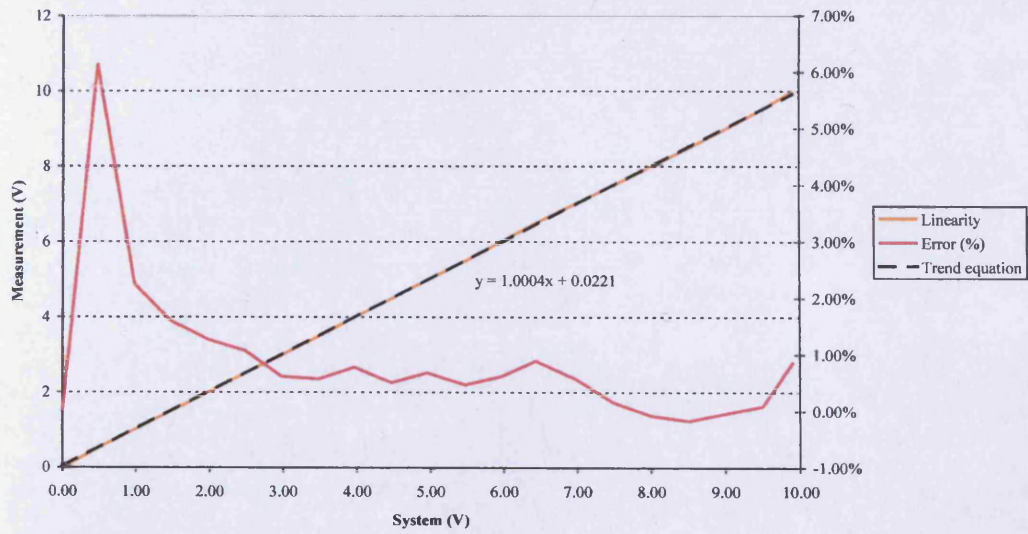
Figure A.5: Sub-net message format using CAN development kit.

Analogue 1 - Linearity (Ascending)



(a)

Analogue 1 - Linearity (Descending)



(b)

Figure A.6: Analogue-1 input linearity measurement, (a) increasing and (b) decreasing the voltage.

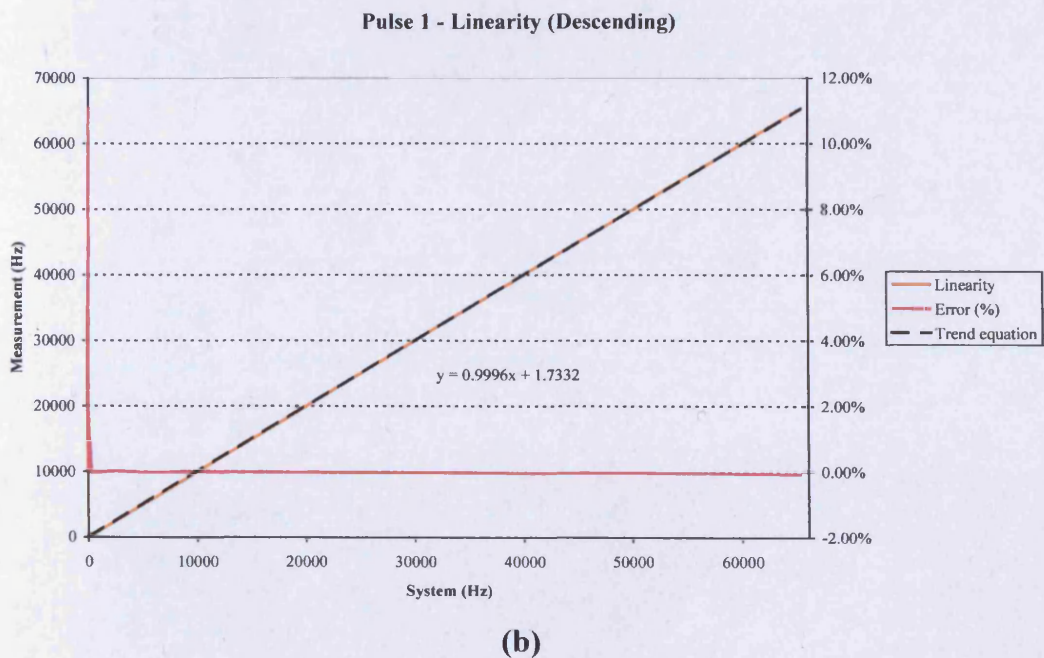
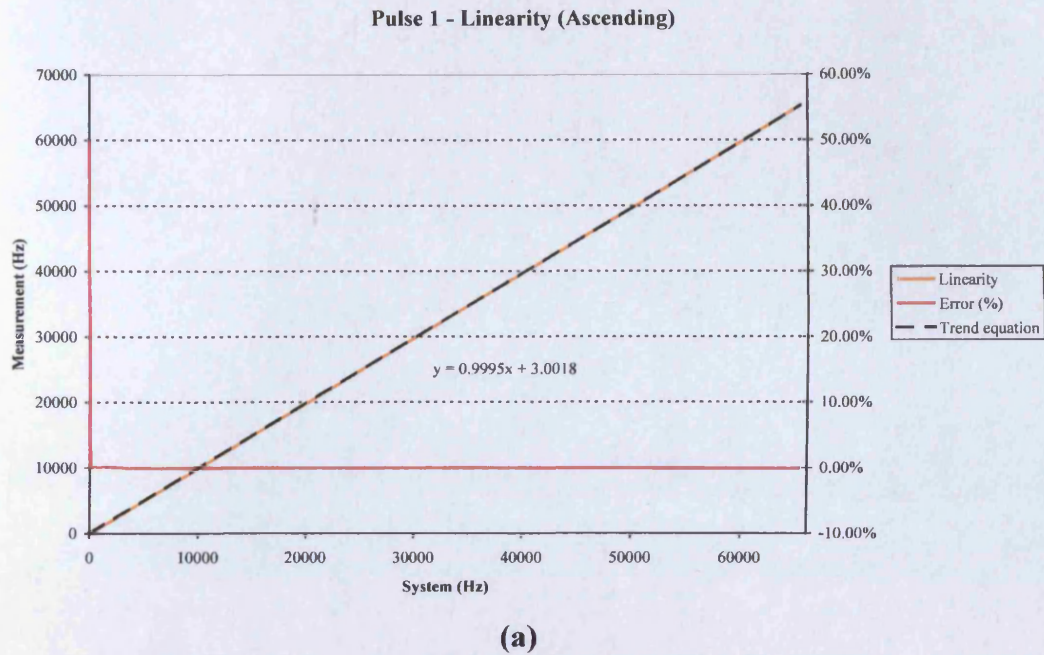


Figure A.7: Pulse-1 input linearity measurement, (a) increasing and (b) decreasing the frequency.

A complete set of graphs related to all (analogue and pulse) inputs linearity test can be found in the attached CD-ROM. Further details are available in the Section “Attached Documents and Files”, later in this appendix (A.25-6).

Appendix A – Measurement Test: Repeatability

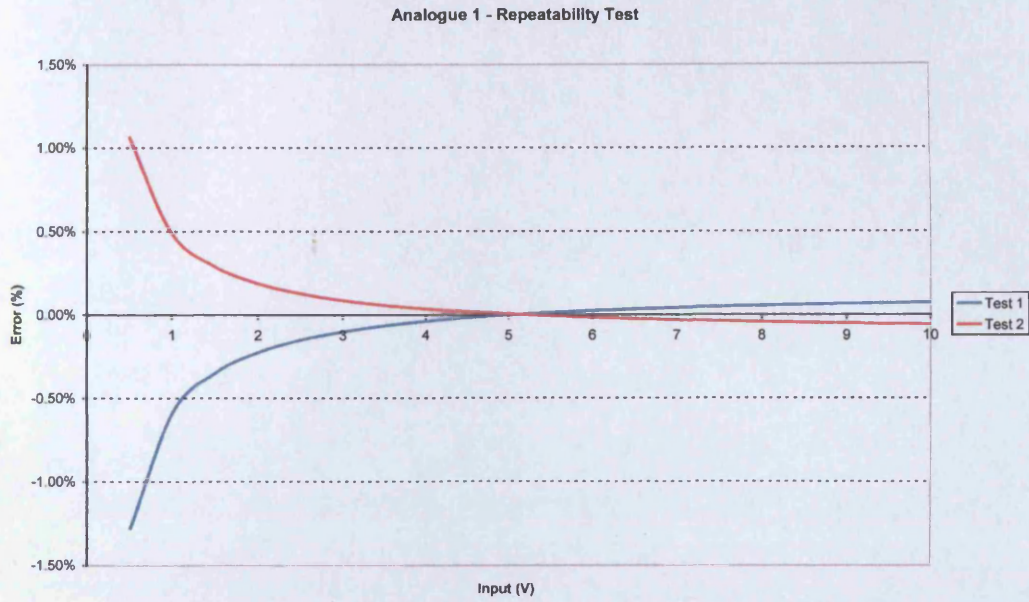


Figure A.8: Analogue-1 input repeatability test.

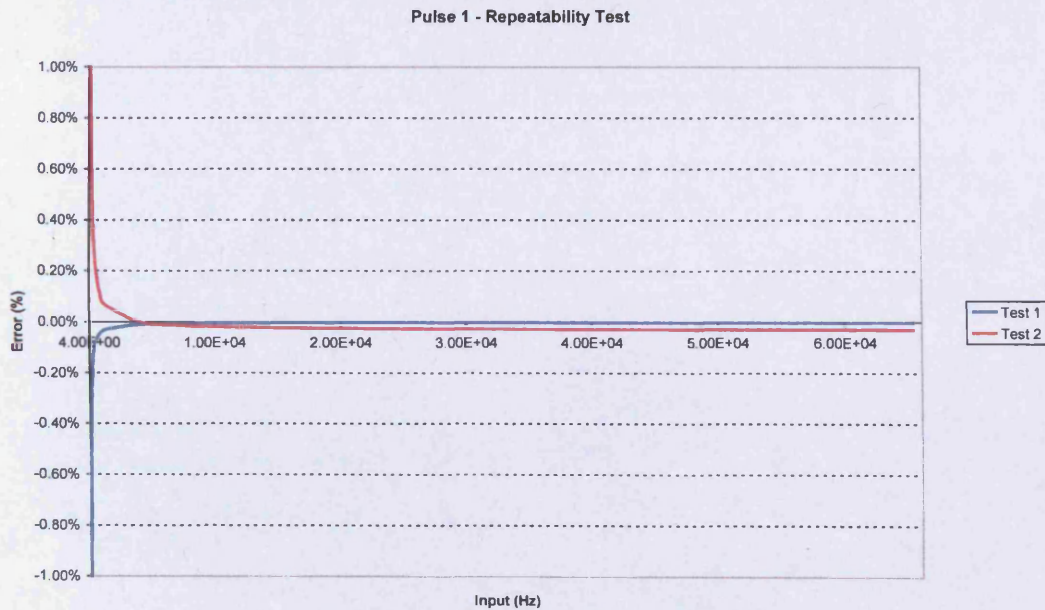
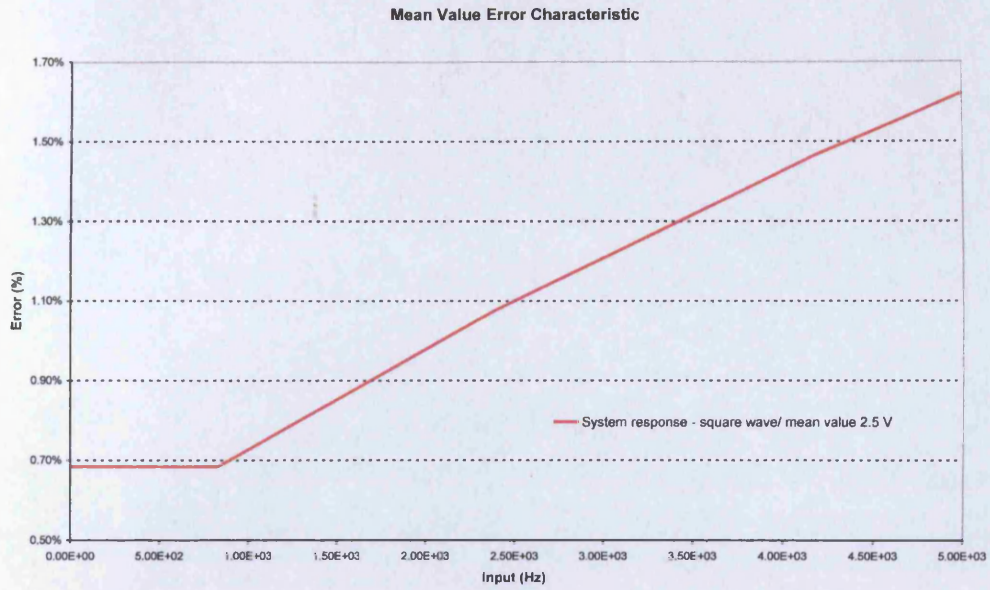


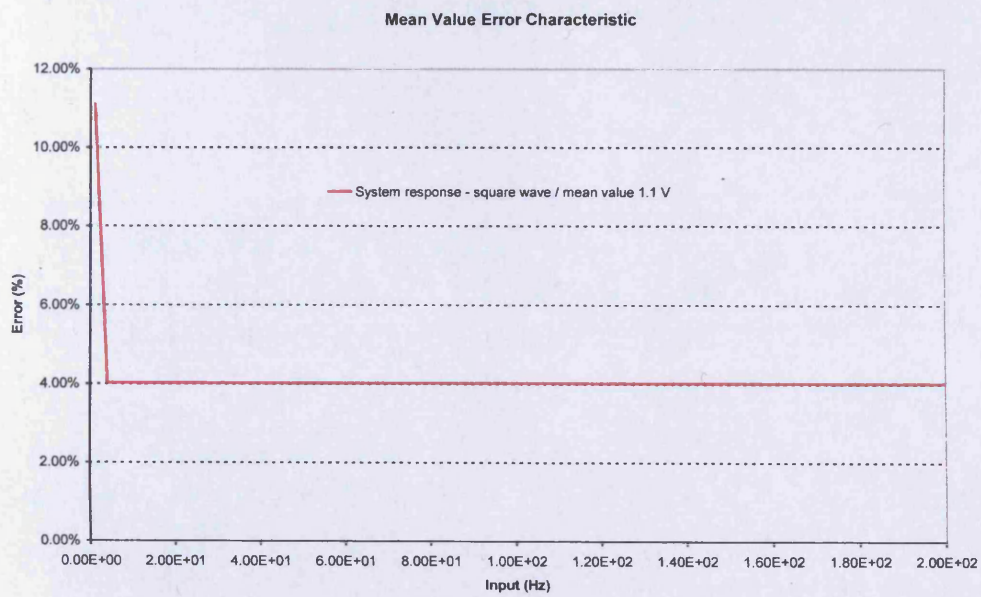
Figure A.9: Pulse-1 input repeatability test.

A complete set of graphs related to all (analogue and pulse) inputs repeatability test can be found in the attached CD-ROM. Further details are available in the Section “Attached Documents and Files”, later in this appendix (A.25-7).

Appendix A – Measurement Test: Mean Value Calculation



(a)



(b)

Figure A.10: Square wave mean value calculation error, having (a) 2.5 V and (b) 1.1 V as input signal mean value.

CAN bus Monitoring Examples

Figure A.11 shows a set of system’s commands flowing on the CAN bus captured with the CAN monitoring tool. The “output window” provides information regarding the message priority, data field length and the application data. The first line displayed represents a “set-time” command; the second a “set-date” and the last one a “reset Petri-net”. Such commands will be broadcast to all Monitoring Modules. Thus, the first field in the data block was assigned “0”.

Ident	Flg	Len	D0	D1	D2	D3	D4	D5	D6	D7	Time	Dir
07F7	8	8	00	11	2F	54	02	FD	EA	38	746.487	R
07F7	8	8	00	11	2F	44	40	E2	77	12	749.320	R
07F7	4	4	00	11	2F	52					755.172	R

Figure A.11: CAN bus monitoring window display system’s commands.

Figure A.12 shows a “fired transition” record transmitted by a monitoring module. The record was required to be split in 3 CAN frames. Frames with identification field “7EF” have as destination the Connectivity Module. The first element in the data field identifies the monitoring module that originated the message. The second element identifies the specific segment of the whole message (31 – first of 3; 32 – second of 3; 33 – last of 3). In the first segment (31), the third and fourth bytes of the data field inform the Connectivity Module the entire length (ASCII) of the record to be transmitted over the Internet. Following are a separator (“/”) and record class fields (54 – “T” for transition). Other fields are the event identification and the timestamp. After receiving all segments, the Connectivity Module will reply with a broadcast message (identification 7F7 in Figure A.12) acknowledging the reception (41 – “A”) to the monitoring module that originated it (01). Figure A.13 shows another “output window”, where a timeout record was captured. In this case, the entire message required further 2 bytes, in order to report the timeout value.

Ident	Flg Len	D0.....D7	Time	Dir
07KF	8	01 31 31 31 2F 54 01 02	859.707	R
07KF	8	01 32 80 A4 FF 02 12 77	859.707	R
07KF	3	01 33 83	859.707	R
07F7	5	00 11 2F 41 01	859.717	R

Figure A.12: CAN monitoring window displaying a fired transition record.

Ident	Flg Len	D0.....D7	Time	Dir
07KF	8	01 31 31 33 2F 58 01 03	1419.835	R
07KF	8	01 32 95 30 08 03 12 77	1419.835	R
07KF	5	01 33 95 08 00	1419.835	R
07F7	5	00 11 2F 41 01	1419.850	R

Figure A.13: CAN monitoring window displaying a timeout record.

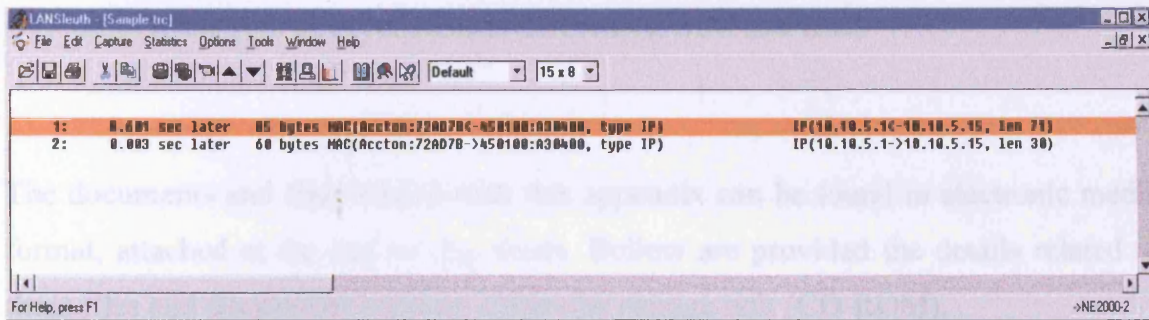
Figure A.14 shows two sub-net messages, identified by the last byte in the data field (01 and 02). Sub-nets' priority fields are identified as "7DF", making them "invisible" to the connectivity module. The first element in the data field (00) makes it acceptable for all monitoring modules (broadcast). The second element indicates total and specific segment number, while the following "E" (2F 45) identify the message at the application level (sub-net).

Ident	Flg Len	D0.....D7	Time	Dir
07DF	5	00 11 2F 45 01	912.481	T
07DF	5	00 11 2F 45 02	912.494	R

Figure A.14: CAN monitoring window displaying sub-net records.

Ethernet / Internet Protocols Monitoring Examples

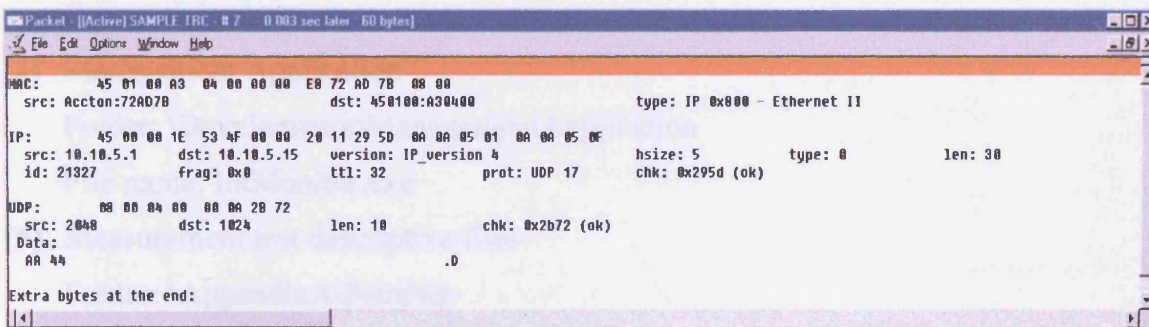
Figure A.15 illustrates messages exchanged between connectivity module and management application. Figure A.15(a) shows messages captured from the Ethernet network, where a record was sent from the Connectivity Module (1) and acknowledged by the Management Application (2). Details of such transmissions are shown in Figure A.15(b) and A.15(c). The first (b) shows a connectivity module transmission carrying a monitoring module timeout record. In such case, the UDP data field begins with a “U” character, which identifies it as a message. The next byte in the data field represents the message sequence number (44), followed by the record length (0D). The remaining valid bytes in the datagram data field represent the record originated by the Monitoring Module (class, timestamp and timeout value). Figure A.15(c) shows a similar window containing the acknowledgement issued by the management application, indicating the correctness of the received message. Such message will provide, in the UDP data field, two valid bytes: the first identifying the nature of the message (AA (hexa) – complement of the “U” character characterising an acknowledgement) and the second corresponding to the sequence number that it will confirm. The absence of such confirmation within a defined period of time will result in the message retransmission. Other messages exchanged by the system follow similar patterns.



(a)



(b)



(c)

Figure A.15: Ethernet/Internet protocols monitoring, (a) message exchange, (b) message details and (c) message acknowledgement details.

ATTACHED DOCUMENTS AND FILES

The documents and files related with this appendix can be found in electronic media format, attached at the end of this thesis. Bellow are provided the details related to these files and documents location within the storage unit (CD-ROM).

(1) Monitoring Module development files

Folder: \Development\MonitoringModule

File name: Pnet.htm (main application's code file)

Other files can be accessed by means of the hyperlink provided within Pnet.htm.

(2) Connectivity Module development files (CAN bus node)

Folder: \Development\ConnectivityModule\CANNode

File name: CANbus.htm (main application's code file)

Other files can be accessed by means of the hyperlink provided within CANbus.htm.

(3) Connectivity Module development files (Internet protocols)

Folder: \Development\ConnectivityModule\InternetProtocols

File name: Connectivity.htm (main application's C code file)

Other files can be accessed by means of the hyperlink provided within Connectivity.htm.

(4) Management Application

Folder: \Development\ManagementApplication

File name: IntMonitor.exe

(5) Measurement test descriptive files

Folder: \AppendixA\PetriNet

File name: PetriNet.htm (Petri-net), WatchStatus.htm (begin/end states) and WatchTrigger.htm (acquisition trigger)

(6) Measurement test linearity graphs

Folder: \AppendixA\HardwareTests

File name: Linearity.htm

(7) Measurement test repeatability graphs

Folder: \AppendixA\HardwareTests

File name: Repeatability.htm

APPENDIX B

MONITORING APPLICATIONS RELATED DETAILS

Press Rig Event Description File

```

MODULE_ID      EQU 1           ;monitoring module
SPECIAL_PL     EQU D'200'     ;timeout control limited to place 200

;transition 1
DB D'1', B'00000000'          ;T1                / ordinary, no transmission
DB B'00000001',B'11111111'   ;DS01 – stop       / active state OFF
DB B'00000000',B'11111111'   ;card 2 – none selected /does not matter
DB B'00000000',B'11111111'   ;card 3 – none selected /does not matter
DB D'0',D'0'                 ;no sub-net input  / input place 0
DB D'1',H'FF'                ;1 token           / end of input places
DB D'7',D'1'                 ;output place 7    / 1 token
DB D'201',D'1'               ;output place 201  / 1 token
DB D'202',D'1'               ;output place 202  / 1 token
DB D'203',D'1'               ;output place 203  / 1 token
DB D'210',D'1'               ;output place 210  / 1 token
DB D'212',D'1'               ;output place 212  / 1 token
DB D'214',D'1'               ;output place 214  / 1 token
DB D'230',D'1'               ;output place 230  / 1 token
DB D'231',D'1'               ;output place 231  / 1 token
DB H'FF',D'0'                ;end of output places / no sub-net output

;transition 2
DB H'FF', D'2'                ;end of structure T1 / T2
DB B'00000000',B'00001000'   ;ordinary, no transmission / DS04 – left motor
DB B'11110111',B'00000000'   ;active state ON    / card 2 – none selected
DB B'11111111',B'00000000'   ;does not matter    / card 3
DB B'11111111',D'0'          ;does not matter    / no sub-net input
DB D'201',D'1'               ;input place 201    / 1 token
DB H'FF',D'5'                ;end of input places / output place 5
DB D'1',D'207'               ;1 token            / output place 207
DB D'1',D'1'                 ;1 token            / output place 1
DB D'1',H'FF'                ;1 token            / end of output places
DB D'0',H'FF'                ;no sub-net output  / end of structure T2

;transition 3
DB D'3',B'00000000'          ;T3                / ordinary, no transmission
DB B'00000100',B'11111011'   ;DS03 - right motor / active state ON
DB B'00000000',B'11111111'   ;card 2 – none selected / does not matter
DB B'00000000',B'11111111'   ;card 3 – none selected / does not matter
DB D'0',D'202'               ;no sub-net input  / input place 202
DB D'1',H'FF'                ;1 token           / end of input places
DB D'6',D'1'                 ;output place 6     / 1 token
DB D'208',D'1'               ;output place 208   / 1 token
DB D'2',D'1'                 ;output place 2     / 1 token
DB H'FF',D'0'                ;end of output places / no sub-net output

;transition 4
DB H'FF',D'4'                ;end of structure T3 / T4
DB B'10000000',B'00000001'   ;ordinary, transmission / DS01 - stop

```

Appendix B – Press Rig Petri-net Descriptive File

```

DB B'11111110',B'000000000' ;active state ON / card 2 – none selected
DB B'11111111',B'000000000' ;does not matter / card 3 – none selected
DB B'11111111',D'0' ;does not matter / no sub-net input
DB D'203',D'1' ;input place 203 / 1 token
DB H'FF',D'204' ;end of input places / output place 204
DB D'1',H'FF' ;1 token / end of output places
DB D'0',H'FF' ;no sub-net output / end of structure T4

;transition 5
DB D'5',B'100000000' ;T5 / ordinary, transmission
DB B'10000000',B'111111111' ;DS08 - reset / active state OFF
DB B'00000000',B'111111111' ;card 2 – none selected / does not matter
DB B'00000000',B'111111111' ;card 3 – none selected / does not matter
DB D'0',D'204' ;no sub-net input / input place 204
DB D'1',H'FF' ;1 token / end of input places
DB D'0',D'1' ;output place 0 (restart) / 1 token
DB H'FF',D'0' ;end of output places / no sub-net output

;transition 6
DB H'FF',D'6' ;end of structure T5 / T6
DB B'10000000',B'000000000' ;ordinary, transmission / card 1 – none selected
DB B'11111111',B'00000010' ;does not matter / DS10 – left guard
DB B'11111111',B'000000000' ;active state OFF / card 3 – none selected
DB B'11111111',D'0' ;does not matter / no sub-net input
DB D'207',D'1' ;input place 207 / 1 token
DB D'5',D'1' ;input place 205 / 1 token
DB H'FF',D'209' ;end of input places / output place 209
DB D'1',H'FF' ;1 token / end of output places
DB D'0',H'FF' ;no sub-net output / end of structure T6

;transition 7
DB D'7',B'000000000' ;T7 / ordinary, no transmission
DB B'0001000',B'111111111' ;DS04 – left motor / active state OFF
DB B'00000000',B'111111111' ;card 2- none selected / does not matter
DB B'00000000',B'111111111' ;card 3 – none selected / does not matter
DB D'0',D'5' ;no sub-net input / input place 5
DB D'1',D'207' ;1 token / input place 207
DB D'1',H'FF' ;1 token / end of input places
DB D'10',D'1' ;output place 10 / 1 token
DB H'FF',D'0' ;end of output places / no sub-net output

;transition 8
DB H'FF',D'8' ;end of structure T7 / T8
DB B'00000000',B'00000100' ;ordinary, no transmission / DS03 – right motor
DB B'11111111',B'000000000' ;active state OFF / card 2 – none selected
DB B'11111111',B'000000000' ;does not matter / card 3 – none selected
DB B'11111111',D'0' ;does not matter / no sub-net input
DB D'6',D'1' ;input place 6 / 1 token
DB D'208',D'1' ;input place 208 / 1 token
DB H'FF',D'10' ;end of input places / output place 10
DB D'1',H'FF' ;1 token / end of output places
DB D'0',H'FF' ;no sub-net outputs / end of structure T8

;transition 9
DB D'9',B'100000000' ;T9 / ordinary, transmission
DB B'00000000',B'111111111' ;card 1 – none selected / does not matter
DB B'00000001',B'111111111' ;DS09 – right guard / active state OFF
DB B'00000000',B'111111111' ;card 3 – none selected / does not matter
DB D'0',D'208' ;no sub-net input / input place 208
DB D'1',D'6' ;1 token / input place 6
DB D'1',H'FF' ;1 token / end of input places
DB D'211',D'1' ;output place 211 / end of input places
DB H'FF',D'0' ;end of output places / 1 token

;transition 10
DB H'FF',D'10' ;end of structure T9 / T10

```

Appendix B – Press Rig Petri-net Descriptive File

```

DB B'00000000',B'00000000' / ordinary, no transmission / card 1 – none selected
DB B'11111111',B'00000010' / does not matter / DS10 – left guard
DB B'11111101',B'00000000' / active state ON / card 3 – none selected
DB B'11111111',D'0' / does not matter / no sub-net input
DB D'209',D'1' / input place 209 / 1 token
DB H'FF',D'15' / end of input places / output places 15
DB D'1',H'FF' / 1 token / end of output places
DB D'0',H'FF' / no sub-net output / end of structure T10
;transition 11
DB D'11',B'00000000' / T11 / ordinary, no transmission
DB B'00000000',B'11111111' / card 1 - none selected / does not matter
DB B'00000001',B'11111110' / DS09 – right guard / active state ON
DB B'00000000',B'11111111' / card 3 – none selected / does not matter
DB D'0',D'211' / no sub-net input / input place 211
DB D'1',H'FF' / 1 token / end of input places
DB D'15',D'1' / output place 15 / 1 token
DB H'FF',D'0' / end of output places / no sub-net output
;transition 12
DB H'FF',D'12' / end of structure T11 / T12
DB B'00000000',B'00000010' / ordinary, no transmission / DS02 – vertical axis motor
DB B'11111101',B'00000000' / active state ON / card 2 – none selected
DB B'11111111',B'00000000' / does not matter / does not matter
DB B'11111111',D'0' / does not matter / no subnet input
DB D'10',D'1' / input place 10 / 1 token
DB H'FF',D'12' / end of input places / output places 12
DB D'1',H'FF' / 1 token / end of output places
DB D'0',H'FF' / no sub-net output / end of structure T12
;transition 13
DB D'13',B'00000000' / T13 / ordinary, no transmission
DB B'00000010',B'11111111' / DS02 – vertical axis motor / active state OFF
DB B'00000000',B'11111111' / card 2 – none selected / does not matter
DB B'00000000',B'11111111' / card 3 – none selected / does not matter
DB D'0',D'12' / no sub-net input / input place 12
DB D'1',H'FF' / 1 token / end of input places
DB D'13',D'1' / output place 13 / 1 token
DB H'FF',D'0' / end of output places / no sub-net output
;transition 14
DB H'FF',D'14' / end of structure T13 / T14
DB B'00000000',B'00010000' / ordinary, no transmission / DS05 – vertical axis motor
DB B'11101111',B'00000000' / active state ON / card 2 – none selected
DB B'11111111',B'00000000' / does not matter / card 3 – none selected
DB B'11111111',D'0' / does not matter / no sub-net input
DB D'13',D'1' / input place 13 / 1 token
DB H'FF',D'14' / end of input places / output place 14
DB D'1',H'FF' / 1 token / end of output places
DB D'0',H'FF' / no sub-net output / end of structure T14
;transition 15
DB D'15',B'00000000' / T15 / ordinary, no transmission
DB B'00010000',B'11111111' / DS05 – vertical axis motor / active state OFF
DB B'00000000',B'11111111' / card 2 – none selected / does not matter
DB B'00000000',B'11111111' / card 3 – none selected / does not matter
DB D'0',D'14' / no sub-net input / input place 14
DB D'1',H'FF' / 1 token / end of input places
DB D'15',D'1' / output place 15 / 1 token
DB H'FF',D'0' / end of output places / no sub-net output
;transition 16
DB H'FF',D'16' / end of structure T15 / T16
DB B'00000000',B'01000000' / ordinary, no transmission / DS07 – left motor
DB B'10111111',B'00000000' / active state ON / card 2 – none selected
DB B'11111111',B'00000000' / does not matter / card 3 – none selected

```


Appendix B – Press Rig Petri-net Descriptive File

```

DB B'11111111',D'0'
DB D'15',D'1'
DB H'FF',D'216'
DB D'1',D'18'
DB D'1',H'FF'
DB D'0',H'FF'
;transition 17
DB D'17',B'00000000'
DB B'00100000',B'11011111'
DB B'00000000',B'11111111'
DB B'00000000',B'11111111'
DB D'0',D'15'
DB D'1',H'FF'
DB D'217',D'1'
DB D'19',D'1'
DB H'FF',D'0'
;transition 18
DB H'FF',D'18'
DB B'10000000',B'00000000'
DB B'11111111',B'00000010'
DB B'11111111',B'00000000'
DB B'11111111',D'0'
DB D'216',D'1'
DB D'18',D'1'
DB H'FF',D'20'
DB D'1',H'FF'
DB D'0',H'FF'
;transition 19
DB D'19',B'10000000'
DB B'00000000',B'11111111'
DB B'00000001',B'11111111'
DB B'00000000',B'11111111'
DB D'0',D'217'
DB D'1',D'19'
DB D'1',H'FF'
DB D'21',D'1'
DB H'FF',D'0'
;transition 20
DB H'FF',D'20'
DB B'00000000',B'01000000'
DB B'11111111',B'00000000'
DB B'11111111',B'00000000'
DB B'11111111',D'0'
DB D'216',D'1'
DB D'18',D'1'
DB H'FF',D'205'
DB D'1',H'FF'
DB D'0',H'FF'
;transition 21
DB D'21',B'00000000'
DB B'00100000',B'11111111'
DB B'00000000',B'11111111'
DB B'00000000',B'11111111'
DB D'0',D'217'
DB D'1',D'19'
DB D'1',H'FF'
DB D'206',D'1'
DB H'FF',D'0'
;transition 22
DB H'FF',D'22'
;does not matter
;input place 15
;end of input places
;1 token
;1 token
;no sub-net output
;T17
;DS06 – right motor
;card 2 – none selected
;card 3 – none selected
;no sub-net input
;1 token
;output place 217
;output place 19
;end of output places
;end of structure T17
;ordinary, transmission
;does not matter
;active state OFF
;does not matter
;input place 216
;input place 18
;end of input places
;1 token
;no sub-net output
;T19
;card 1 – none selected
;DS09 – right guard
;card 3 – none selected
;no sub-net input
;1 token
;1 token
;output place 21
;end of output places
;end of structure T19
;ordinary, no transmission
;active state OFF
;does not matter
;does not matter
;input place 216
;input place 18
;end of input places
;1 token
;no sub-net output
;T21
;DS06 – right motor
;card 2 – none selected
;card 3 – none selected
;no sub-net input
;1 token
;1 token
;output place 206
;end of output places
;end of structure T21
;no sub-net input
;1 token
;output place 216
;output place 18
;end of output places
;1 token
;end of structure T20
;ordinary, no transmission
;active state OFF
;does not matter
;does not matter
;input place 217
;input place 19
;end of input places
;1 token
;no sub-net output
;T20
;DS07 – left motor
;card 2 – none selected
;card 3 – none selected
;no sub-net input
;1 token
;1 token
;output place 205
;end of output places
;end of structure T20
;ordinary, no transmission
;active state OFF
;does not matter
;does not matter
;input place 217
;input place 19
;end of input places
;1 token
;no sub-net output
;T22

```

Appendix B – Press Rig Petri-net Descriptive File

```

DB B'00000000',B'00000000' ;ordinary, no transmission / card 1 – none selected
DB B'11111111',B'00000010' ;does not matter / DS10 – left guard
DB B'11111110',B'00000000' ;active state ON / card 3 – none selected
DB B'11111111',D'0' ;does not matter / no sub-net input
DB D'20',D'1' ;input place 20 / 1 token
DB H'FF',D'205' ;end of input places / output place 205
DB D'1',H'FF' ;1 token / end of output places
DB D'0',H'FF' ;no sub-net output / end of structure T22
;transition 23
DB D'23',B'00000000' ;T23 / ordinary, no transmission
DB B'00000000',B'11111111' ;card 1 – none selected / does not matter
DB B'00000001',B'11111110' ;DS09 – right guard / active state ON
DB B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB D'0',D'21' ;no sub-net input / input place 21
DB D'1',H'FF' ;1 token / end of input places
DB D'206',D'1' ;output place 206 / 1 token
DB H'FF',D'0' ;end of output places / no sub-net output
;transition 24
DB H'FF',D'24' ;end of structure T23 / T24
DB B'00000010',H'1' ;delay transition (500 ms) / MSB parameter
DB H'F4',D'222' ;LSB parameter / input place 222
DB D'223',H'FF' ;output place 223 / end of structure T24
;transition 25
DB D'25',B'10000001' ;T25 / analogue, transmission
DB B'10000001',H'02' ;high edge - Analogue 1 / MSB parameter
DB H'00',D'223' ;LSB parameter / input place 223
DB D'1',H'FF' ;1 token / end of input places
DB D'224',D'1' ;output place 224 / 1 token
DB H'FF',D'0' ;end of output places / no sub-net output
;transition 26
DB H'FF',D'26' ;end of structure T25 / T26
DB B'10000001',B'00010001' ;analogue, transmission / low edge - Analogue 1
DB H'00',H'1A' ;MSB parameter / LSB parameter
DB D'223',D'1' ;input place 223 / 1 token
DB H'FF',D'224' ;end of input places / output place 224
DB D'1',H'FF' ;1 token / end of output places
DB D'0',H'FF' ;no sub-net output / end of structure T26
;transition 27
DB D'27',B'00000000' ;T27 / ordinary, no transmission
DB B'00000010',B'11111111' ;DS02 / vertical axis motor
DB B'00000000',B'11111111' ;card 2 – none selected / does not matter
DB B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB D'0',D'223' ;no sub-net input / input place 223
DB D'1',H'FF' ;1 token / end of input places
DB D'224',D'1' ;output place 224 / 1 token
DB H'FF',D'0' ;end of output places / no sub-net output
;transition 28
DB H'FF',D'28' ;end of structure T27 / T28
DB B'00000000',B'00000000' ;ordinary, no transmission / card 1 – none selected
DB B'11111111',B'00000000' ;does not matter / card 2 – none selected
DB B'11111111',B'00000000' ;does not matter / card 3 – none selected
DB B'11111111',D'0' ;does not matter / no sub-net input
DB D'1',D'1' ;input place 1 / 1 token
DB D'205',D'1' ;input place 205 / 1 token
DB H'FF',D'201' ;end of input places / output place 201
DB D'1',D'228' ;1 token / output place 228
DB D'1',H'FF' ;1 token / end of output places
DB D'0',H'FF' ;no sub-net output / end of structure T28
;transition 29
DB D'29',B'00000000' ;T29 / ordinary, no transmission

```

Appendix B – Press Rig Petri-net Descriptive File

```

DB B'00000000',B'1111111111' ;card 1 – none selected / does not matter
DB B'00000000',B'1111111111' ;card 2 – none selected / does not matter
DB B'00000000',B'1111111111' ;card 3 – none selected / does not matter
DB D'0',D'2' /input place 2
DB D'1',D'206' /input place 206
DB D'1',H'FF' /end of input places
DB D'202',D'1' /1 token
DB D'229',D'1' /1 token
DB H'FF',D'0' /no sub-net output
;transition 30
DB H'FF',D'30' / T30
DB B'00000000',B'00000010' / DS02 – vertical axis motor
DB B'11111101',B'00000000' / card 2 – none selected
DB B'11111111',B'00000000' / card 3 – none selected
DB B'11111111',D'0' /no sub-net input
DB D'210',D'1' /1 token
DB H'FF',D'222' /output 222
DB D'1',D'25' /output place 25
DB D'1',H'FF' /end of output places
DB D'0',H'FF' /end of structure T30
;transition 31
DB D'31',B'00000000' / ordinary, no transmission
DB B'00000010',B'1111111111' / DS02 – vertical axis motor / active state OFF
DB B'00000000',B'1111111111' / does not matter
DB B'00000000',B'1111111111' / does not matter
DB D'0',D'25' /input place 25
DB D'1',D'224' /input place 224
DB D'1',H'FF' /end of input places
DB D'226',D'1' /1 token
DB H'FF',D'0' /no sub-net output
;transition 32
DB H'FF',D'32' / T32
DB B'00000000',B'00010000' / DS05 – vertical axis motor
DB B'11101111',B'00000000' / card 2 – none selected
DB B'11111111',B'00000000' / card 3 – none selected
DB B'11111111',D'0' /no sub-net input
DB D'226',D'1' /1 token
DB H'FF',D'26' /output place 26
DB D'1',H'FF' /end of output places
DB D'0',H'FF' /end of structure T32
;transition 33
DB D'33',B'00000000' / ordinary, no transmission
DB B'00010000',B'1111111111' / DS05 –vertical axis motor / active state OFF
DB B'00000000',B'1111111111' / does not matter
DB B'00000000',B'1111111111' / does not matter
DB D'0',D'26' /input place 26
DB D'1',H'FF' /end of input places
DB D'210',D'1' /1 token
DB H'FF',D'0' /no sub-net output
;transition 34
DB H'FF',D'34' / T34
DB B'00000000',B'000000100' / DS03 - right motor
DB B'11111011',B'00000000' / card 2 – none selected
DB B'11111111',B'00000000' / card 3 – none selected
DB B'11111111',D'0' /no sub-net input
DB D'212',D'1' /1 token
DB H'FF',D'213' /output place 213
DB D'1',D'23' /output place 23
DB D'1',H'FF' /end of output places
DB D'0',H'FF' /end of structure T34

```

Appendix B – Press Rig Petri-net Descriptive File

```

;transition 35
DB D'35',B'00000010' / delay transition (500 ms)
DB H'1',H'F4' / LSB parameter
DB D'213',D'218' / output place 218
;transition 36
DB H'FF',D'36' / T36
DB B'10000001',B'10000010' / high edge - Analogue 2
DB H'02',H'00' / LSB parameter
DB D'218',D'1' / 1 token
DB H'FF',D'219' / output place 219
DB D'1',H'FF' / end of output places
DB D'0',H'FF' / no sub-net output
;transition 37
DB D'37',B'00000000' / ordinary, no transmission
DB B'0000100',B'11111111' / active state OFF
DB B'00000000',B'11111111' / does not matter
DB B'00000000',B'11111111' / does not matter
DB D'0',D'218' / input place 218
DB D'1',H'FF' / end of input places
DB D'219',D'1' / 1 token
DB H'FF',D'0' / no sub-net output
;transition 38
DB H'FF',D'38' / T38
DB B'10000001',B'00010010' / low edge - Analogue 2
DB H'00',H'1A' / LSB parameter
DB D'218',D'1' / 1 token
DB H'FF',D'219' / output place 219
DB D'1',H'FF' / end of output places
DB D'0',H'FF' / no sub-net output
;transition 39
DB D'39',B'00000000' / ordinary, no transmission
DB B'0000100',B'11111111' / active state OFF
DB B'00000000',B'11111111' / does not matter
DB B'00000000',B'11111111' / does not matter
DB D'0',D'219' / input place 219
DB D'1',D'23' / input place 23
DB D'1',H'FF' / end of input places
DB D'225',D'1' / 1 token
DB H'FF',D'0' / no sub-net output
;transition 40
DB H'FF',D'40' / T40
DB B'00000000',B'00100000' / DS06 - right motor
DB B'11011111',B'00000000' / card 2 – none selected
DB B'11111111',B'00000000' / card 3 – none selected
DB B'11111111',D'0' / no sub-net input
DB D'225',D'1' / 1 token
DB H'FF',D'27' / output place 27
DB D'1',H'FF' / end of output places
DB D'0',H'FF' / no sub-net output
;transition 41
DB D'41',B'00000000' / ordinary, no transmission
DB B'00100000',B'11111111' / active state OFF
DB B'00000000',B'11111111' / does not matter
DB B'00000000',B'11111111' / does not matter
DB D'0',D'27' / input place 27
DB D'1',H'FF' / end of input places
DB D'212',D'1' / 1 token
DB H'FF',D'0' / no sub-net output
;transition 42
DB H'FF',D'42' / T42

```

Appendix B – Press Rig Petri-net Descriptive File

```

DB   B'00000000',B'00001000' ;ordinary, no transmission / DS04 – left motor
DB   B'11110111',B'00000000' ;active state ON / card 2 – none selected
DB   B'11111111',B'00000000' ;does not matter / card 3 – none selected
DB   B'11111111',D'0' ;does not matter / no sub-net input
DB   D'214',D'1' ;input place 214 / 1 token
DB   H'FF',D'215' ;end of input places / output place 215
DB   D'1',D'24' ;1 token / output place 24
DB   D'1',H'FF' ;1 token / end of output places
DB   D'0',H'FF' ;no sub-net output / end of structure T42
;transition 43
DB   D'43',B'00000010' ;T43 / delay transition (500 ms)
DB   H'1',H'F4' ;MSB parameter / LSB parameter
DB   D'215',D'220' ;input place 215 / output place 220
;transition 44
DB   H'FF',D'44' ;end of structure T43 / T44
DB   B'10000001',B'10000011' ;analogue, transmission / high edge - Analogue 3
DB   H'02',H'00' ;MSB parameter / LSB parameter
DB   D'220',D'1' ;input place 220 / 1 token
DB   H'FF',D'221' ;end of input places / output place 221
DB   D'1',H'FF' ;1 token / end of output places
DB   D'0',H'FF' ;no sub-net output / end of structure T44
;transition 45
DB   D'45',B'00000000' ;T45 / ordinary, no transmission
DB   B'00001000',B'11111111' ;DS04 – left motor / active state OFF
DB   B'00000000',B'11111111' ;card 2 – none selected / does not matter
DB   B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB   D'0',D'220' ;no sub-net input / input place 220
DB   D'1',H'FF' ;1 token / end of input places
DB   D'221',D'1' ;output place 221 / 1 token
DB   H'FF',D'0' ;end of output places / no sub-net output
;transition 46
DB   H'FF',D'46' ;end of structure T45 / T46
DB   B'10000001',B'00010011' ;analogue, transmission / low edge - Analogue 3
DB   H'00',H'1A' ;MSB parameter / LSB parameter
DB   D'220',D'1' ;input place 220 / 1 token
DB   H'FF',D'221' ;end of input places / output place 221
DB   D'1',H'FF' ;1 token / end of output places
DB   D'0',H'FF' ;no sub-net output / end of structure 46
;transition 47
DB   D'47',B'00000000' ;T47 / ordinary, no transmission
DB   B'00001000',B'11111111' ;DS04 – left motor / active state OFF
DB   B'00000000',B'11111111' ;card 2 – none selected / does not matter
DB   B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB   D'0',D'221' ;no sub-net input / input place 221
DB   D'1',D'24' ;1 token / input place 24
DB   D'1',H'FF' ;1 token / end of input places
DB   D'227',D'1' ;output place 227 / 1 token
DB   H'FF',D'0' ;end of output places / no sub-net output
;transition 48
DB   H'FF',D'48' ;end of structure T47 / T48
DB   B'00000000',B'01000000' ;ordinary, no transmission / DS07 - right motor
DB   B'10111111',B'00000000' ;active state ON / card 2 – none selected
DB   B'11111111',B'00000000' ;does not matter / card 3 – none selected
DB   B'11111111',D'0' ;does not matter / no sub-net input
DB   D'227',D'1' ;input place 227 / 1 token
DB   H'FF',D'28' ;end of input places / output place 28
DB   D'1',H'FF' ;1 token / end of output places
DB   D'0',H'FF' ;no sub-net output / end of structure T48
;transition 49
DB   D'49',B'00000000' ;T49 / ordinary, no transmission

```

Appendix B – Press Rig Petri-net Descriptive File

```

DB   B'01000000',B'11111111' ;DS07 – left motor      / active state OFF
DB   B'00000000',B'11111111' ;card 2 – none selected / does not matter
DB   B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB   D'0',D'28'                ;no sub-net input       / input place 28
DB   D'1',H'FF'                ;1 token                / end of input places
DB   D'214',D'1'              ;output place 214       / 1 token
DB   H'FF',D'0'               ;end of output places   / no sub-net output

;transition 50
DB   H'FF',D'50'              ;end of structure T49   / T50
DB   B'00000000',B'00000000' ;ordinary, no transmission / card 1 – none selected
DB   B'11111111',B'00000000' ;does not matter        / card 2 – none selected
DB   B'11111111',B'00000000' ;does not matter        / card 3 – none selected
DB   B'11111111',D'0'        ;does not matter        / no sub-net input
DB   D'1',D'1'                ;input place 1          / 1 token
DB   D'7',D'1'                ;input place 7          / 1 token
DB   H'FF',D'1'              ;end of input places    / output place 1
DB   D'1',D'9'                ;1 token                / output place 9
DB   D'1',H'FF'              ;1 token                / end of output places
DB   D'0',H'FF'              ;no sub-net output      / end of structure T50

;transition 51
DB   D'51',B'00000000'       ;T51                    / ordinary, no transmission
DB   B'00000001',B'11111110' ;DS01 - stop            / active state ON
DB   B'00000000',B'11111111' ;card 2 – none selected / does not matter
DB   B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB   D'0',D'7'                ;no sub-net input       / input place 7
DB   D'1',H'FF'              ;1 token                / end of input places
DB   D'8',D'1'                ;output place 8         / 1 token
DB   H'FF',D'0'              ;end of output places   / no sub-net output

;transition 52
DB   H'FF',D'52'              ;end of structure T51   / T52
DB   B'00000000',B'00000000' ;ordinary, no transmission / card 1 – none selected
DB   B'11111111',B'00000000' ;does not matter        / card 2 – none selected
DB   B'11111111',B'00000000' ;does not matter        / card 3 – none selected
DB   B'11111111',D'0'        ;does not matter        / no sub-net input
DB   D'2',D'1'                ;input place 2          / 1 token
DB   D'7',D'1'                ;input place 7          / 1 token
DB   H'FF',D'2'              ;end of input places    / output place 2
DB   D'1',D'9'                ;1 token                / output place 9
DB   D'1',H'FF'              ;1 token                / end of output places
DB   D'0',H'FF'              ;no sub-net output      / end of structure

;transition 53
DB   D'53',B'00000000'       ;T53                    / ordinary, no transmission
DB   B'00000001',B'11111111' ;DS01 - stop            / active state OFF
DB   B'00000000',B'11111111' ;card 2 – none selected / does not matter
DB   B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB   D'0',D'8'                ;no sub-net input       / input place 8
DB   D'1',H'FF'              ;1 token                / end of input places
DB   D'7',D'1'                ;output place 7         / 1 token
DB   H'FF',D'0'              ;end of output places   / no sub-net output

;transition 54
DB   H'FF',D'54'              ;end of structure T53   / T54
DB   B'00000000',B'00000001' ;ordinary, no transmission / DS01 - stop
DB   B'11111110',B'00000000' ;active state ON        / card 2 – none selected
DB   B'11111111',B'00000000' ;does not matter        / card 3 – none selected
DB   B'11111111',D'0'        ;does not matter        / no sub-net input
DB   D'9',D'1'                ;input place 9          / 1 token
DB   H'FF',D'8'              ;end of input place     / output place 8
DB   D'1',H'FF'              ;1 token                / end of output places
DB   D'0',H'FF'              ;no sub-net output      / end of structure T54

;transition 55

```

Appendix B – Press Rig Petri-net Descriptive File

```

DB   D'55',B'00000000'   ;T55           / ordinary, no transmission
DB   B'00000000',B'11111111' ;card 1 – none selected / does not matter
DB   B'00000000',B'11111111' ;card 2 – none selected / does not matter
DB   B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB   D'0',D'9'           ;no sub-net input      / input place 9
DB   D'1',D'228'        ;1 token               / input place 228
DB   D'1',H'FF'         ;1 token               / end of input places
DB   D'7',D'1'          ;output place 7        / 1 token
DB   H'FF',D'0'         ;end of output places  / no sub-net output

;transition 56
DB   H'FF',D'56'        ;end of structure T55  / T56
DB   B'00000000',B'00000000' ;ordinary, no transmission / card 1 – none selected
DB   B'11111111',B'00000000' ;does not matter       / card 2 – none selected
DB   B'11111111',B'00000000' ;does not matter       / card 3 – none selected
DB   B'11111111',D'0'    ;does not matter       / no sub-net input
DB   D'9',D'1'          ;input place 9         / 1 token
DB   D'229',D'1'        ;input place 229       / 1 token
DB   H'FF',D'7'         ;end of input places   / output place 7
DB   D'1',H'FF'         ;1 token               / end of output places
DB   D'0',H'FF'         ;no sub-net output     / end of structure T56

;transition T57
DB   D'57',B'00000000'   ;T57           / ordinary, no transmission
DB   B'00000000',B'11111111' ;card 1 – none selected / does not matter
DB   B'00000010',B'11111111' ;DS10 – left guard     / active state OFF
DB   B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB   D'0',D'230'        ;no sub-net input      / input place 230
DB   D'1',H'FF'         ;1 token               / end of input places
DB   D'3',D'1'          ;output place 3        / 1 token
DB   H'FF',D'0'         ;end of output places  / no sub-net output

;transition 58
DB   H'FF',D'58'        ;end of structure T57  / T58
DB   B'00000000',B'00000000' ;ordinary, no transmission / card 1 – none selected
DB   B'11111111',B'00000010' ;does not matter       / DS10 – left guard
DB   B'11111101',B'00000000' ;active state ON       / card 3 – none selected
DB   B'11111111',D'0'    ;does not matter       / no sub-net input
DB   D'3',D'1'          ;input place 3         / 1 token
DB   H'FF',D'230'       ;end of input places   / output place 230
DB   D'1',H'FF'         ;1 token               / end of output places
DB   D'0',H'FF'         ;no sub-net output     / end of structure T58

;transition 59
DB   D'59',B'00000000'   ;T59           / ordinary, no transmission
DB   B'00000000',B'11111111' ;card 1 – none selected / does not matter
DB   B'00000001',B'11111111' ;DS09 - right guard    / active state OFF
DB   B'00000000',B'11111111' ;card 3 – none selected / does not matter
DB   D'0',D'231'        ;no sub-net input      / input place 231
DB   D'1',H'FF'         ;1 token               / end of input places
DB   D'4',D'1'          ;output place 4        / 1 token
DB   H'FF',D'0'         ;end of output places  / no sub-net output

;transition 60
DB   H'FF',D'60'        ;end of structure T59  / T60
DB   B'00000000',B'00000000' ;ordinary, no transmission / card 1 – none selected
DB   B'11111111',B'00000001' ;does not matter       / DS09 – right guard
DB   B'11111110',B'00000000' ;active state ON       / card 3 – none selected
DB   B'11111111',D'0'    ;does not matter       / no sub-net input
DB   D'4',D'1'          ;input place 4         / 1 token
DB   H'FF',D'231'       ;end of input places   / output place 231
DB   D'1',H'FF'         ;1 token               / end of output places
DB   D'0',H'FF'         ;no sub-net output     / end of structure T60

;end of Petri-net
DB   EndOfTable         ;end of table

```

Press Rig Process States File (Status Watching)

DB B'11001111',B'00000001' ;places 1 to 16
DB B'00000000',B'00000000' ;places 17 to 32
DB B'00000000',B'00000000' ;places 33 to 48
DB B'00000000',B'00000000' ;places 49 to 64
DB B'00000000',B'00000000' ;places 65 to 80
DB B'00000000',B'00000000' ;places 81 to 96
DB B'00000000',B'00000000' ;places 97 to 112
DB B'00000000',B'00000000' ;places 113 to 128
DB B'00000000',B'00000000' ;places 129 to 144
DB B'00000000',B'00000000' ;places 145 to 160
DB B'00000000',B'00000000' ;places 161 to 176
DB B'00000000',B'00000000' ;places 177 to 192
DB B'00000000',B'00000000' ;places 193 to 208
DB B'00000000',B'00000000' ;places 209 to 224
DB B'00000000',B'00000000' ;places 225 to 240
DB B'00000000',B'00000000' ;places 241 to 256 (not processed)

- **Selected places:**

- **1 – left cycle**
- **2 – right cycle**
- **3 – left guard**
- **4 – right guard**
- **7 – process available**
- **8 – process stopped**
- **9 – process operation**

Press Rig Analogue/Pulse Acquisition Trigger

```
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',ANALOGUE2      ;places 22 & 23  
DB ANALOGUE3,ANALOGUE1 ;places 24 & 25  
DB ANALOGUE1,ANALOGUE2 ;places 26 & 27  
DB ANALOGUE3,D'0'      ;places 28 & 29  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'  
DB D'0',D'0'
```

Appendix B -- Press Rig Petri-net Descriptive File

DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'
DB D'0,D'0'

Appendix B – Press Rig Petri-net Descriptive File

DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'
DB D'0',D'0'

The descriptive files related to the other two applications presented as examples in this work (Chapters 9 and 10) are part of electronic documents placed in the attached CD-ROM. Further details are available in the Section “Attached Documents”, later in this appendix (B.17 and B.18).

Conveyor Rig OEE Calculation – Database Procedure

```

CREATE TRIGGER [StatusInsert] ON [PICuser].[ConveyorRig_ST]
FOR UPDATE
AS
    DECLARE @Availability    real
    DECLARE @Performance    real
    DECLARE @QualityRate    real
    DECLARE @OEE            real
    DECLARE @Idle           real
    DECLARE @Stopped        real
    DECLARE @Processing      real
    DECLARE @Good           real
    DECLARE @RejPeg         real
    DECLARE @RejRing        real
    DECLARE @WrongRej       real
    DECLARE @FailRej        real
    DECLARE @TheDay         varchar(12)

    SET @TheDay = CONVERT(varchar(12), GETDATE(), 111)
    SELECT @Idle = SUM(HOW_LONG) FROM [ConveyorRig_ST] WHERE((
        EVENT_ID = 30) AND (CONVERT(varchar(12), ENDED_AT, 111)
        = @TheDay))
    SELECT @Stopped = SUM(HOW_LONG) FROM [ConveyorRig_ST]
        WHERE(( EVENT_ID = 28) AND (CONVERT(varchar(12),
        ENDED_AT, 111) = @TheDay))
    SELECT @Processing = SUM(HOW_LONG) FROM [ConveyorRig_ST]
        WHERE(( EVENT_ID = 29) AND (CONVERT(varchar(12),
        ENDED_AT, 111) = @TheDay))
    SELECT @Good = COUNT(EVENT_ID) FROM [ConveyorRig_EV] WHERE((
        EVENT_ID = 31) AND (CLASS = 'T') AND
        (CONVERT(varchar(12), TIME_STAMP, 111) = @TheDay))
    SELECT @RejPeg = COUNT(EVENT_ID) FROM [ConveyorRig_EV]
        WHERE(( EVENT_ID = 28) AND (CLASS = 'T') AND
        (CONVERT(varchar(12), TIME_STAMP, 111) = @TheDay))
    SELECT @RejRing = COUNT(EVENT_ID) FROM [ConveyorRig_EV]
        WHERE(( EVENT_ID = 9) AND (CLASS = 'T') AND
        (CONVERT(varchar(12), TIME_STAMP, 111) = @TheDay))
    SELECT @WrongRej = COUNT(EVENT_ID) FROM [ConveyorRig_EV]
        WHERE(( EVENT_ID = 30) AND (CLASS = 'T') AND
        (CONVERT(varchar(12), TIME_STAMP, 111) = @TheDay))
    SELECT @FailRej = COUNT(EVENT_ID) FROM [ConveyorRig_EV]
        WHERE(( EVENT_ID = 29) AND (CLASS = 'T') AND
        (CONVERT(varchar(12), TIME_STAMP, 111) = @TheDay))
    IF(@Processing > 0 AND @Good > 0)
    BEGIN
        SET @Availability = ((@Idle + @Processing) / (@Idle +
            @Processing + @Stopped))
        SET @Performance = (@Processing / (@Processing + @Idle))
    
```

Appendix B – OEE Calculation Script

```
SET    @QualityRate = (@Good / (@Good + @RejPeg +
@WrongRej + @FailRej))
SET    @OEE = (@Availability * @Performance * @QualityRate *
100)
END
ELSE
BEGIN
    SET    @Availability = 0
    SET    @Performance = 0
    SET    @QualityRate = 0
    SET    @OEE = 0
END
UPDATE OEETable SET ActualAvail = @Availability WHERE ProcessID =
'ConveyorRig'
UPDATE OEETable SET ActualPerform = @Performance WHERE
ProcessID = 'ConveyorRig'
UPDATE OEETable SET ActualQualRate = @QualityRate WHERE
ProcessID = 'ConveyorRig'
UPDATE OEETable SET ActualOEE = @OEE WHERE ProcessID =
'ConveyorRig'
UPDATE OEETable SET BestAvail = @Availability WHERE((BestAvail <
@Availability) AND (ProcessID = 'ConveyorRig'))
UPDATE OEETable SET BestPerform = @Performance
WHERE((BestPerform < @Performance) AND (ProcessID =
'ConveyorRig'))
UPDATE OEETable SET BestQualRate = @QualityRate
WHERE((BestQualRate < @QualityRate) AND (ProcessID =
'ConveyorRig'))
SELECT  @Availability = BestAvail FROM OEETable WHERE ProcessID =
'ConveyorRig'
SELECT  @Performance = BestPerform FROM OEETable WHERE
ProcessID = 'ConveyorRig'
SELECT  @QualityRate = BestQualRate FROM OEETable WHERE
ProcessID = 'ConveyorRig'
SET    @OEE = (@Availability * @Performance * @QualityRate * 100)
UPDATE OEETable SET BestOEE = @OEE WHERE ProcessID = 'ConveyorRig'
```

ATTACHED DOCUMENTS AND FILES

The documents and files related with this appendix can be found in electronic media format, attached at the end of this thesis. Bellow are provided the details related to these files and documents location within the storage unit (CD-ROM).

- (1) Press rig - Petri-net diagram descriptive file
Folder: \AppendixB\PetriNet\PressRig
File name: Petrinet.htm
- (2) Press rig – state status (beginning/ending) descriptive file
Folder: \AppendixB\PetriNet\PressRig
File name: WatchStatus.htm
- (3) Press rig – analogue / pulse input descriptive file
Folder: \AppendixB\PetriNet\PressRig
File name: WatchTrigger.htm
- (4) Conveyor rig - Petri-net diagram descriptive file
Folder: \AppendixB\PetriNet\ConveyorRig
File name: PetriNet.htm
- (5) Conveyor rig - state status (beginning/ending) descriptive file
Folder: \AppendixB\PetriNet\ConveyorRig
File name: WatchStatus.htm
- (6) Conveyor Rig – analogue / pulse input descriptive file
Folder: \AppendixB\PetriNet\ConveyorRig
File name: WatchTrigger.htm
- (7) Tool Changer - Petri-net diagram descriptive file
Folder: \AppendixB\PetriNet\ToolChanger
File name: Petrinet.htm
- (8) Tool Changer – state status (beginning/ending) descriptive file
Folder: \AppendixB\PetriNet\ToolChanger
File name: WatchStatus.htm
- (9) Tool Changer – analogue / pulse input descriptive file
Folder: \AppendixB\PetriNet\ToolChanger
File name: WatchTrigger.htm

Appendix B – Attached Documents and Files

(10) OEE calculation database script

Folder: \AppendixB

File name: ConveyorRigOEE.htm

(11) Web page examples showing the IPMM approach

Folder: \AppendixB

File name: IPMMWebPage.htm

(12) System's application examples (best viewed with resolution set to 1280 x 1024)

Folder: \AppendixB

File name: ConveyorRig.avi and PressRig.avi

APPENDIX C

PUBLICATIONS AND PRESENTATIONS

- (1) Frankowiak, M.R., Grosvenor, R.I., Prickett, P.W., Jennings, A.D. and Turner, J.R. Design of a PIC Based Data Acquisition System for Process and Condition Monitoring. *In Proceedings: Comadem 2001*, Manchester, UK: Elsevier Science, 2001, 481-488.

Document (paper) in the attached CD-ROM: \AppendixC\Document1.htm

(Oral presentation in the 14th International Congress on Condition Monitoring and Diagnostic Engineering Management, Manchester, UK)

- (2) Frankowiak, M.R., Grosvenor, R.I., Prickett, P.W. and Jennings, A.D. A Microcontroller Based Distributed System Using a Petri-net Approach. *In Proceedings: Comadem 2002*, Birmingham, UK: Comadem International, 2002, 37-44.

Document (paper) in the attached CD-ROM: \AppendixC\Document2.htm

(Oral presentation in the 15th International Congress on Condition Monitoring and Diagnostic Engineering Management, Birmingham, UK)

- (3) Jennings, A.D, Prickett, P.W., Grosvenor, R.I. and Frankowiak, M.R. Process and Condition Monitoring Using the Internet (e-monitoring). *In Proceedings: Comadem 2002*, Birmingham, UK: Comadem International, 2002, 45-52.

Document (paper) in the attached CD-ROM: \AppendixC\Document3.htm

(Oral presentation in the 15th International Congress on Condition Monitoring and Diagnostic Engineering Management, Birmingham, UK)

- (4) Frankowiak, M.R., Grosvenor, R.I., Prickett, P.W. and Jennings, A.D. Distributed Monitoring System Using PIC Microcontroller Technologies. *In Proceedings: ICOM2003*, Loughborough, UK: Professional Engineering Publications, 2003, 415-420.

Document (paper) in the attached CD-ROM: \AppendixC\Document4.htm

(Oral presentation in the International Conference on Mechatronics, Loughborough, UK)

Appendix C – List of Publications

- (5) Frankowiak M.R. Manufacturing Process and Machine Monitoring Employing Microcontroller Technologies. Speaking of Science, Cardiff, 2003.

(Oral presentation in the BA Speaking of Science Conference, Cardiff, UK)

Document (abstract) in the attached CD-ROM: \AppendixC\Document5.htm

- (6) Frankowiak, M.R., Grosvenor, R.I., Prickett, P.W. and Jennings, A.D. A Petri-net Based Distributed Monitoring System Using PIC Microcontrollers.

Document (paper) in the attached CD-ROM: \AppendixC\Document6.htm

(Submitted to the Microprocessors and Microsystems Journal)

