AN OPERATING SYSTEM FOR DATA ACQUISITION AND CONTROL APPLICATIONS

Tolga AYAV

August, 1999

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

T000049

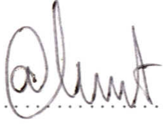
We approve the thesis of **Tolga AYAV**

Date of Signature

Prof. Dr. Sıtkı AYTAÇ

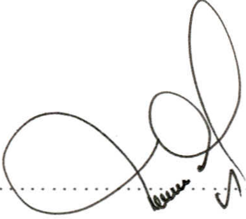
Supervisor

Department of Computer Engineering — 25 / 08 / 1999



Asst. Prof. Dr. Ahmet KOLTUKSUZ

Department of Computer Engineering — 25 / 08 / 1999



Prof. Dr. Şaban EREN

Department of Computer Engineering — 25 / 08 / 1999

Ege University

Prof. Dr. Sıtkı AYTAÇ

Head of Department — 25 / 08 / 1999

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor, Prof. Dr. Sıtkı Aytaç, for his support to my project in many aspects. I also thank Asst. Prof. Ahmet Koltuksuz for encouraging me in all stages of the project.

I would also like to thank Prof. Dr. Halis Püskülcü, since he willingly enabled me to prepare this thesis in spite of my primary jobs.

Finally, I would like to thank my family and my friends who have given me the real support so far.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

ABSTRACT

The common controllers used in industrial environments today cannot fulfill the requirements of many data acquisition and control applications. As a result of this, personal computers has become to be popular in industry, as they have been so in many areas due to the fact that today's PCs have many advantages compared with their relatively low price. In this project, a PC based embedded controller was designed for data acquisition and control purposes, and a real-time executive running on DOS operating system was developed.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

ÖZ

Günümüzde endüstriyel ortamlarda yaygın olarak kullanılan kontrol cihazları, karşılaşılan bir çok veri toplama ve kontrol uygulamasının ihtiyaçlarını karşılayamamaktadır. Bunun sonucu olarak kişisel bilgisayarlar, düşük fiyatlarının yanında bir çok avantajlara sahip olmalarından dolayı, diğer alanlarda da olduğu gibi endüstride de popülerlik kazanmıştır. Bu projede veri toplama ve kontrol amaçları için kişisel bilgisayar tabanlı bir gömülü kontrol sistemi tasarlanmış ve bu sistemi gerçek zamanlı olarak yürütebilen ve DOS işletim sistemi üzerinde çalışan bir program geliştirilmiştir.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

TABLE OF CONTENTS

LIST OF FIGURES viii

LIST OF TABLES ix

CHAPTER 1. INTRODUCTION 1

CHAPTER 2. THE DEFINITIONS AND DESIGN CONSIDERATIONS 3

2.1. The Definition of the Term "Embedded Controller" 3

2.2. The Definition of the PC104 Standard 4

2.3. The Definition of the Term "Real-Time" 4

2.4. Design Considerations for IBM PC Extension Boards 5

2.5. Design Considerations for Data Acquisition & Control Applications 6

2.6. Design Considerations for Real-Time Systems 7

CHAPTER 3. THE PRINCIPLES OF DATA ACQUISITION & CONTROL 9

3.1. Data Acquisition Speed 9

3.2. Data Acquisition Accuracy 10

3.3. Aliasing 12

3.4. Signal Conditioning 13

3.5. Environmental Limitations of Data Acquisition Equipment 13

3.6. Software 14

CHAPTER 4. THE PARTS OF THE DEVELOPED EMBEDDED CONTROLLER 16

4.1. The PC Mainboard 16

4.2. Digital Input / Output Board 16

4.2.1. Address Decoder Circuit 17

4.2.2. Opto-Isolated Input 20

4.2.3. Relay Output 21

4.2.4. Connectors and Indicators 21

4.3. Analog Input & Power Supply Board 21

4.3.1. Address Decoder & Latch Circuit 22

4.3.2. Analog Multiplexer and Signal Conditioning Circuit 23

4.3.3. DC/DC Converter 24

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

4.3.4. Connectors 24
4.4. Enclosure 24
CHAPTER 5. THE SOFTWARE OF THE EMBEDDED CONTROLLER 25
5.1. Accessing the Hardware 25
5.1.1. Using the Digital I/O Board 25
5.1.2. Using the Analog Input Board 28
5.2. Explanation of he Developed Real-Time Executive 30
5.2.1. Scheduler 32
5.2.2. Interprocess Communication 32
5.2.3. Preparing the System Configuration File, "load.dac" 33
5.2.4. System Calls 34
5.2.5. Making Processes 39
5.2.6. Making Timer Functions 43
CHAPTER 6. CONCLUSION, FUTURE WORK 44
SUMMARY 46
ÖZET 47
REFERENCES 48

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

LIST OF FIGURES

Figure 2.1 The Block Diagram of a Prototype Board

Figure 3.1 Block Diagram of a Data Acquisition and Control Application

Figure 3.2 Single-Ended and Differential Measurements

Figure 3.3 Measurement with Resistor Bridge, Measurement without Ground

Figure 3.4 The Signal Diagram of Aliasing

Figure 4.1 The Block Diagram of the Digital I/O Board

Figure 4.2 Address Decoder Circuit of the Digital I/O Board

Figure 4.3 The Input Layer of Digital Inputs

Figure 4.4 The Output Layer of Digital Outputs

Figure 4.5 The Block Diagram of the Analog Input & Power Supply Board

Figure 4.6 The Address Decoder of the Analog Input & Power Supply Board

Figure 4.7 Analog Multiplexing and Signal Conditioning Circuit

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

LIST OF TABLES

Table 4.1 Input and Output States of the Address Decoder Circuit
Table 4.2 Addressing of the Digital I/O Board
Table 4.3 Explanation of the Ports Used by the Digital I/O Board
Table 4.4 Voltage Levels of the Digital Inputs
Table 4.5 Explanation of the Ports Used by the Analog Input & Power Supply Board
Table 5.1 Explanation of the Ports Used by the Digital I/O Board
Table 5.2 Explanation of the Ports Used by the Analog Input & Power Supply Board
Table 5.3 Explanation of the Reading Operations from Digital to Analog Converter

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

CHAPTER 1

INTRODUCTION

Developing high performance data acquisition and control systems takes better tools than the industry normally provides. Today the most common controllers in industrial environments are Programmable Logic Controllers (PLC). They are the devices which were developed especially for industry, hence they are quite suitable for control applications. However, despite today's technology, PLCs can be still weak in many applications. For instance they are not suitable for data acquisition due to their limited capability of computation, speed, memory and programming constraints. On the other hand, the fact that today's PCs have many advantages compared with their relatively low price made them very popular everywhere, in every environment. As a result of this intendation, PCs were used firstly for data acquisition purposes in laboratories, then for control purposes in industry. The huge capacity of computation and memory, user-friendly environment, ease of programming and impressive graphical screens made people to prefer PCs instead of PLCs or other systems. On the other hand, it was the fact that desktop PCs were not suitable for industrial regions, since industry has different working conditions. In order to overcome this problem, different methods were tried so as to use conventional desktop PCs, or new products which are suitable for industry were developed in time. The number of PCs used in industry is increasing day by day, and different options, products are being developed as a result of this.

In respond to this need, a new embedded controller was designed in this project. This controller is actually a PC controller and consists of basically a PC mainboard, digital and analog input/output boards, a power supply unit, an enclosure and a suitable software for a harmonious working of all these parts. This project deals with designing the digital and analog I/O boards, the enclosure and developing the software, not the mainboard and power supply. Therefore the mainboard and power supply unit were chosen from on-the-shelf products which are extremely widespread and appropriate in many aspects such as technical specifications and prices.

Therefore this project basically contains a hardware design, and a software development. In the first part, two electronic boards, analog input and power supply board and digital I/O board were designed in accordance with the PC104 standard which is the format of the PC mainboard chosen. Then an executive running on DOS and providing a real time multitasking environment was developed.

This controller has 24 opto-isolated digital inputs and 24 relay digital outputs which are capable of doing on/off operations and completely isolated from the PC against any possible hazardous electrical shocks. In addition, it has 16 channels of analog input for acquisition of analog signals such as temperature, pressure, displacement, etc. Analog inputs accept the voltage level within the range from –10 to +10 volts and also have fault-protection up to 40 volts. Maximum sampling rate is 16,000 samples/s and the processor

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

may handle many complicated tasks at even the highest sampling rate, without losing any data. The performance-cost ratio of this system is expected to be higher than similar ones.

The controller has the following features:

1. AMD 5x86 133 MHz CPU with arithmetic coprocessor.
2. 2MB Flash Ram for operating system and programs.
3. 24 volt dc power supply.
4. One RS-232C serial port.
5. VGA display connector.
6. Keyboard connector.
7. 24 channels of opto-isolated input (provided with three connector blocks on the enclosure).
8. 24 channels of relay output (provided with three connector blocks on the enclosure).
9. 16 channels of analog input (provided with four connector blocks on the enclosure).
10. 48 LEDs as indicator of digital input and outputs and two LEDs for power OK.
11. 250x170x75 mm aluminium enclosure.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

CHAPTER 2

DEFINITIONS AND DESIGN CONSIDERATIONS

2.1. THE DEFINITION OF THE TERM "EMBEDDED CONTROLLER"

An embedded system is a digital system which is acting as part of a larger system. The term *embedded* means being part of a larger system and providing a dedicated service to that unit[1]. A PC which provides a dedicated software and some graphics and communication interfaces can be an embedded control system in a production line of a factory. According to this definition a microprocessor can be regarded as an embedded controller, if it gives a dedicated service such as handling graphics operations.

The earliest embedded systems were banking processing systems running on mainframe computers. These systems were very expensive and applications were small. However, as the integrated circuits manufacturing technology developed, new embedded systems found new application areas. Today's embedded systems can be itemized as:

- Industrial controllers, where there is need for maintainability, reliability and programmability.
- Safety critical controller, such as ABS controller in a car.
- Laser printers, where there is need for computationally intensive.
- Home appliances, where there is a need for user interface and other advanced features which are provided by microcontrollers.

By the term "Embedded Controller", an embedded system which is designed for industrial environments is meant in this project.

The embedded PC can be very different from a desktop PC. It can be hidden from user, it might not have user interface, a display or keyboard, or it might have a different user dialog unit that we are not used to. They frequently include a LCD display and a keypad for user interface. At this point they can seem to have no difference from microcontroller-based designs but they have the distinct advantage of using a PC platform. This provides user with PC development tools and desktop PC's all advantages. The advantages of today's PCs can be listed as follows:

- PCs are ubiquitous.
- PCs make it easy to create prototypes.
- It can be easier to develop a project on a PC
- PC expertise is available.
- PCs offer low cost hardware.
- Low cost, high quality development tools are available for PC.
- A wide variety of PC-compatible products are available.
- Very high level automated tools are available.
- The PC architecture continues to offer increasing performance.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

- PCs offer a huge range of display and input options.
- Many low-level drivers are available.
- The shrinking PC for notebook computers results in technology that is ideal for embedded systems.

2.2. THE DEFINITION OF THE PC104 STANDART

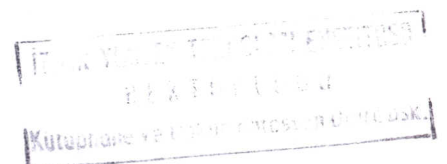
Due to the preceding advantage list, companies which design PCs as controllers began to seek a new product that reaps the benefits of using the PC architecture. However, the standard PC bus form-factor (12.4" x 4.8") and its associated card cages and backplanes are too bulky (and expensive) for most embedded control applications. A need therefore arose for a more compact implementation of the PC bus, satisfying the reduced space and power constraints of embedded control applications. Yet these goals had to be realized without sacrificing full hardware and software compatibility with the popular PC bus standard. This would allow the PC's hardware, software, development tools, and system design knowledge to be fully leveraged.
PC/104 was developed in response to this need. It offers full architecture, hardware and software compatibility with the PC bus, but in ultra-compact (3.6" x 3.8") stackable modules. PC/104 is therefore ideally suited to the unique requirements of embedded control applications.

2.3. THE DEFINITION OF THE TERM "REAL TIME"

A real-time system is defined as a system where the correctness of the system depends not only on the result of computations but also on the time at which it is produced [2]. According to this definition, the time is the most important item that should be managed in real-time systems. The tasks must be assigned and scheduled in such way in order to be completed by their deadlines. Messages and signals to be sent and received between tasks must be arranged to facilitate this timing as well. The other crucial issue in a real-time system is reliability. In many real-time systems such as flight control systems, nuclear plant control systems and various industrial systems, a failure may not only cause economical losses, but also may cost human lives.

A real-time application consists of some coordinated tasks. In some applications these tasks should be activated periodically and has to be completed before a deadline. For instance, in an antilock braking system (ABS), some tasks may be sensing the speed of the wheel and control of the pressure of the breaking pedal. Both the tasks have to be repeated periodically in order to keep the ABS system active. Such tasks are called periodic. Periodic tasks are generally time-critical. Critical tasks should always be completed before their deadlines under any conditions. On the other hand, some tasks can be run aperiodically. For example, if any failure is detected in the system, some action needs to be taken that means that the task is invoked when an event occurred. Aperiodic tasks can also be time-critical, that is they have to be completed before their deadlines. However, in case that they are not time-critical, they again have to be completed as soon as possible in order not to prevent the deadlines of other time-critical tasks.

In addition to timing constraints, there are also other constraints:



- Resource constraints: Other than the processor, tasks may want to access to some hardware or software resources in the system, such as I/O devices, application-specific hardware components, networks, databases etc.
- Precedence constraints: Some tasks may require the results of some other tasks. Therefore they cannot be completed unless the others have been completed.
- Dependability/performance constraints: A task may have to meet some performance requirements.

2.6. Design Considerations for IBM PC ISA Bus Extension Boards

On a PC, there are several types of buses, which provides simply communication between different devices such as the memory bus, the I/O bus and the address bus. The bus concerning with this project is the I/O bus which allows the mainboard or CPU to communicate with any peripheral devices [3]. This bus is used to attach the digital and analog I/O boards to the mainboard in this project.
The type of the I/O bus is important, since it greatly affect the speed of the computer. The oldest bus IBM declared with the first PC is 8-bit ISA (Industry standard architecture) bus. This bus uses 8 bits data bus therefore it allows only 8-bits transfer between the CPU and any plugged device at a time. This is the slowest of all buses, but the basic standard that is still frequently used today. Following the 8 bit ISA bus was the 16 bit ISA bus. This 16 bit bus is double the size of the 8 bit bus and thus can transfer data at twice the speed of the 8 bit bus. The PC104 computer used in this project has a 16 bit ISA compatible bus, however the integrated circuits used on the digital and analog I/O boards have 8 bit data bus. Therefore the designed 8 bit cards could not take advantage of the 16 bit bus.
The PC104 mainboard has a standard 114-contact extension slot. However, 22 signals from this bus are most needed and may be enough for an expansion card. The descriptions of these signals are as follows [4]:

SD0 - SD7
Data is transferred on these lines between CPU and I/O.

SA0 - SA9
These lines are used on the ISA bus to address I/O devices. For I/O board accesses, the first 10 bits of the address bus can be used.

IOR#, IOW#
Signal IOR# indicates a read, signal IOW# a write cycle on an I/O device if signal AEN is active at the same time.

AEN
This low active signal specifies the I/O address space. It must always be used for I/O address coding for expansion cards.

IRQ3-IRQ7, IRQ9-IRQ12, IRQ14, IRQ15
The interrupt signals are used to interrupt program currently executed by the processor and indicates that an I/O device needs to be attendant by the CPU.

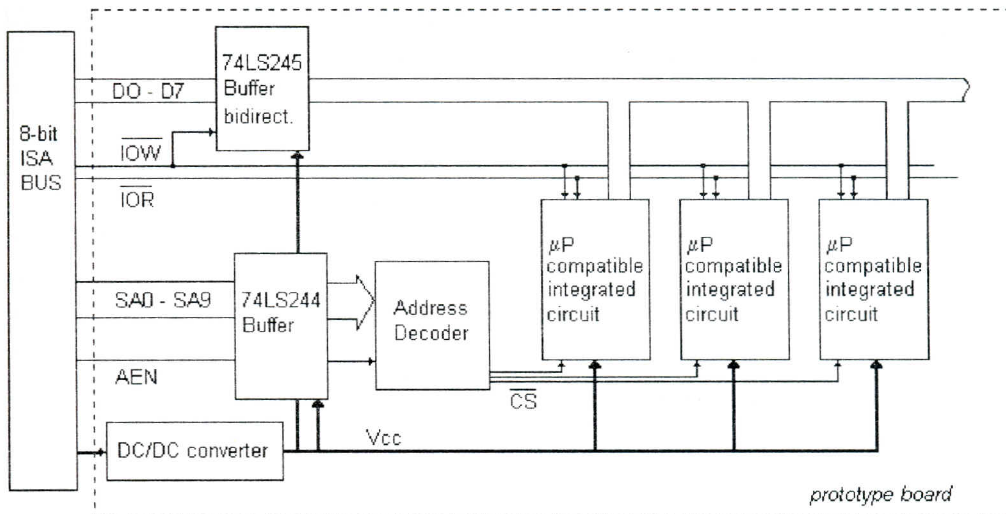


Figure 2.1: The Block Diagram of a Prototype Board

The addresses 0x300 - 0x31F are reserved for prototype cards. In order to make the board to communicate with the CPU properly, a space I/O region must be selected. This selection is made by the address decoder circuit. When the address bus is driven with a predefined address, the decoder generates necessary chip select signals to enable the integrated circuits on the board. When an IC is enabled, it communicates with the CPU via the data bus, using IOR and IOW signals.

On the ISA bus, there are also +5v and GND signals for supplying the circuits. However instead of using this supply directly, it is recommended to use a dc/dc converter against any faulty case.

The data lines are generally buffered by a bi-directional 3-state buffer (74LS245) in order not to load the data bus too much. In same way, the address lines can be buffered with 74LS244 IC. However, since there is only one IC connected to the data bus, which is ADS7803 A/D converter, on the analog input board designed in this project, both 74LS244 and 74LS245 were not used. They were not used on the digital I/O board either due to the same reason.

2.5. Design Considerations for Data Acquisition & Control Applications

There are some requirements that must be taken into account when designing a DAC system. These requirements can be classified as hardware and software requirements. The first step of the process of hardware design is defining the numbers of the inputs and outputs. In order to be able to do this, most common applications can be examined. Larger numbers does not mean better, however the optimum number should be found out. Most DAC systems has 16 single-ended analog inputs and 2 analog outputs. These numbers are suitable for many applications, considering also the cost and size of the system. For digital

I/O, systems generally have 16 inputs and 16 outputs. In this system, the number of analog inputs was chosen as 16, but analog outputs were not used.

On the other hand, the number of digital I/Os was chosen higher, since digital I/Os were extremely needed in control applications, and it is observed that the expansion modules are frequently used to increase the digital I/Os. The other observation is that many DAC boards have TTL I/Os which need additional interfaces for connection between the controller and the real world. In respond to these needs, the number of digital I/Os was chosen as 48 for both inputs and outputs and the necessary interfaces such as opto-coupled isolators and relays were embedded into the digital I/O board. That the digital I/O board contains all the ICs for communication with the CPU, opto-isolator circuits and relays altogether provides a compact and reliable structure.

The other need in data acquisition applications is a programmable gain amplifier for measuring the sensors which have low-voltage output. Programmable gain amplifiers are the integrated circuits which amplifies the input signal and the gain of the amplifier is defined by the user. Gain can generally take three values; 10, 100, 500. Gain amplifier was not used in this project however.

2.6. DESIGN CONSIDERATIONS FOR REAL TIME SYSTEMS

As mentioned in section 2.3, time is the most important factor in real-time systems [5]. How the system handles the time determines the characteristics of the system, and consequently its correctness. It is scheduling that allocates time and resources to task in order to fulfill the timing constraints. Therefore choosing the best scheduling algorithm is vital when setting up a real-time system [6].

There are a number of performance criteria which have been proposed for different types of systems. However, different scheduling algorithms can be grouped as follows:

1. Static table-driven approaches
2. Static priority driven preemptive approaches
3. Dynamic planning-based approaches
4. Dynamic best-effort approaches

Static table-drive scheduling is based on the idea that in order to assure a priori that the deadlines of critical tasks, the resources must be preallocated. The scheduling of such safety-critical tasks are done statically considering the worst possible run-time conditions.

Priority-driven preemptive scheduling is frequently used in multitasking systems. In non-real-time systems, the priority of a task depends whether it is CPU-bound or I/O bound. In real-time systems, the priority of tasks must be related to their timing constraints. Two typical examples of priority-driven algorithms are the rate-monotonic algorithm and earliest-deadline-first algorithm.

Dynamic planning-based schedulers use dynamic feasibility checks to guarantee newly arriving tasks, which means that a task is guaranteed by finding a plan for execution, whereby all tasks meet their deadlines.

Best effort scheduling is a very popular approach in real-time systems today. In this approach, a priority value is computed for each task, according to its characteristics and the tasks are ordered regarding this priority value. Such systems can hardly be predictable and confidence in the scheduling method has to be gained via extensive simulation. Two typical examples of this type of algorithm are earliest-deadline and minimal-laxity approaches.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ

CHAPTER 3

THE PRINCIPLES OF PC BASED DATA ACQUISITION AND CONTROL

A PC based data acquisition & control system consists of the following parts [7]:

1. Physical systems (real-world phenomena)
2. Transducers and Actuators
3. Signal Conditioning equipment
4. Data Acquisition & Control Hardware
5. A PC Software

The following figure shows the interrelations between these parts.

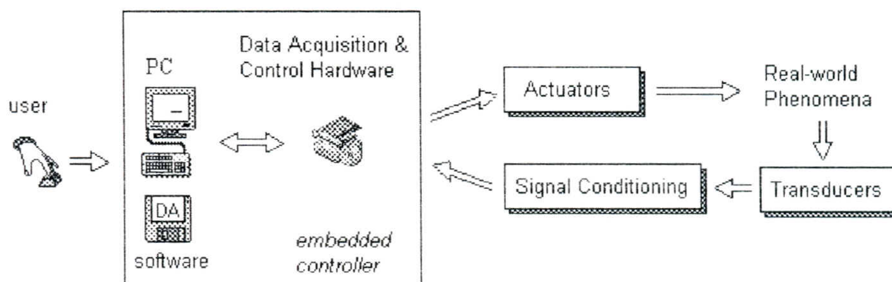


Figure 3.1: Block Diagram of a Data Acquisition and Control Application

There are different types of data acquisition products such as digital meters, chart recorders, data loggers, external boxes and PC plug-in boards. In PC based applications plug-in boards are used and they plug into inside a computer, directly to a bus, with an external "Terminal Panel" or connectors on itself in order to make sensor and other connections. The bus may be ISA, EISA, or PCI, or can be a PCMCIA card.

3.1. DATA ACQUISITION SPEED

Defining the sampling rate of a data acquisition system is vital in many aspects. For slowly changing signals, like temperature, the only consideration is to provide a new reading often enough that the data is reasonably up-to-date. In some applications it is necessary to accurately represent an input waveform. For example, an electrocardiogram must be reproduced with considerable precision since some of the important things a doctor looks for may be very small changes.

The important rule that should be taken into account when defining the sampling rate is the Nyquist theorem. It states that the sample rate must be more than twice the highest frequency in order to measure all the frequencies present.

When the sampling rate is too slow than aliasing may occur. However if it is too high, more than 1 kHz for instance, the internal noise will be more, which reduces the accuracy of the measurement. On the other hand, handling the data in software will be more difficult at high sampling rates, which will be discussed in chapter 4.
In many data acquisition systems, there is only one A/D converter which is multiplexed to 16 or a different number of analog inputs by multiplexer ICs. In case of using more than one analog channel, the maximum sampling rate of the system is divided by the number of the channels used.
For example, if the system has 100 kHz max sampling rate, in case that four analog channels are measured, the maximum sampling rate for each channel will be 25 kHz.

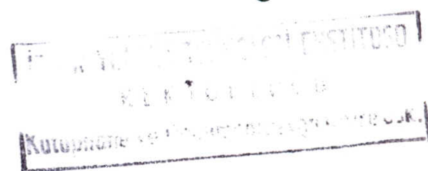
3.2. DATA ACQUISITION ACCURACY

Accuracy is the combined effect of number of factors such as resolution, gain, offset, calibration, common mode rejection, linearity, drift, noise etc. The most important of these factors are briefly described as follows.

Resolution.
Resolution is one of the most important features that differentiate A/D converters. The resolution defines the smallest measurable change in the input signal. The A/D converter converts analog voltage or current to binary words. Resolution is the number of bits for which the A/D converter is rated. A 1-bit converter can detect only two states of a signal, high or low. A 2-bit number can be arranged in four combinations, "00", "01", "10", or "11". Each additional bit doubles the number of possible combinations. For example, a 12-bit converter has 4096 combinations. The typical resolutions used in industrial environments are 12 and 16-bits. Since there are number of factors that affects the accuracy, higher resolutions does not mean better. All factors should be considered when selecting resolution.

Gain and Offset Errors
Gain error results when a change in the input signal yields a different change in the measured value. This is caused by the differences in the components from one device to another. Offset error is the error in the reading at zero input. Both errors can be minimized by adjusting potentiometers. However it is possible to calibrate analog inputs from software. Software calibration does not correct the analog readings physically, but corrects the wrong measurements with some arithmetic operations using gain and offset errors predefined in the software.

Common Mode Rejection, Differential Measurement.
A common mode signal is one that appears equally on both the positive and negative input terminals. Since this type of input appears equally on both inputs, it is ideally completely rejected. However single-ended inputs do not have independent positive and negative inputs therefore do not reject common mode signals.
When the negative input is not exactly at ground, the differential input should be used. A differential input measures the signal that is the difference between the positive and negative terminals. The degree to which differential input is rejected is called common mode rejection. This would measure how much a voltage difference between ground and



sensor's negative output terminal would affect the measured sensor signal. Common mode rejection is expressed in percent or decibels. It is ratio of the measured response divided by the common mode signal.
Differential measurement may be a necessity due to the nature of some sensors such as strain gauges, termistors or may be preferred in order to remove the measurement errors caused by the voltage difference between two ground points. The following figures show how the differential measurement prevents the ground loop errors.

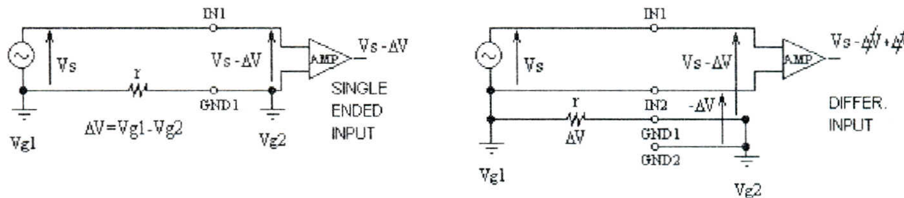


Figure 3.2: Single-Ended and Differential Measurements

As shown in figure 3.2, when there is a potential difference between the two grounds especially with long sensor cables, single-ended input measures the signal with ΔV error. In the left figure, although the self-resistance of the cable actually exist on both lines, ΔV occurs on the bottom line only since no considerable current flows into IN1 point due to the high impedance of the input.
In differential measurements, as one single-ended input measures the faulty signal Vs-ΔV, one more single-ended input is used to measure the error -ΔV, and the input amplifier responds to the difference between these two inputs.
The other reason of using differential input is that some circuits like resistor bridge used to measure strain-gauges and termistors cannot be connected to a single-ended input due to the fact that this will corrupt the balance of the bridge.
It should be noted that when measuring with differential input, the ground must be brought and connected to the ground terminal of one of the single-ended inputs in order to provide a reference. A connection to the ground terminal is also necessary to provide the bias current for proper working of input layers.

If a system ground is not available, a resistor may be placed between each single-ended input and ground, as shown in figure 3.3. The 100KΩ resistors provide current paths to the differential signal to maintain a reference with the ground. If the output impedance of the differential signal is low, the voltage level of the signal is effectively unchanged. The impedance of the differential signal source, which can be considered as being in a voltage divider network with the resistors, causes only a negligible drop in voltage.

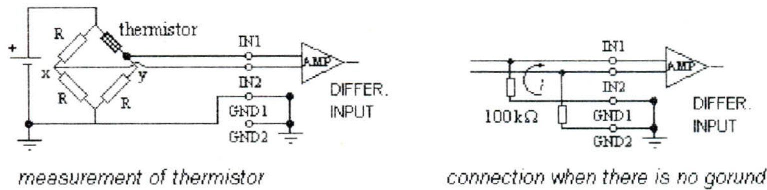


Figure 3.3: Measurement with Resistor Bridge, Measurement without Ground

Although the above circuit will keep unreferenced signals from floating, it should not be used to measure signals that have a large common-mode or DC voltage. If the voltage of either of the differential inputs exceeds the input voltage range, it will be clipped, and erroneous readings may occur.

Linearity.

Linearity error occurs when the gain error varies with different input levels. For example, a change in the input signal from 0 to 1 Volt may show a different change in the measurement than an input signal change from 1 to 2 Volts. This is due to linearity error. Generally the linearity error cannot be removed by calibration. However the linearity errors which are caused by the sensors can be eliminated by some software linearization methods.

Noise.

Noise is probably the worst factor when reading analog inputs. It is seen as random variations on readings. Noise signals are picked up from electrical and electronic devices such as fluorescent lights, motors, radio transmitters, power wires and computers. A certain amount of noise is created inside the data acquisition circuit or device itself. Faster devices have more noise.

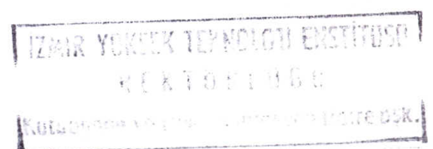
The important point is that noise is much larger than the resolution on most data acquisition systems. Such a noise reduces the resolution seriously. Noise is generally at high frequencies and can be eliminated by hardware or software low-pass filters.

3.3. ALIASING

Aliasing is caused by digitization of data. In PC based systems, data must be digitized since PC can only handle digital bits. Signal is measured at a constant rate and amplitude is represented by digital bits in the PC. The changes between samples are not detected.

Aliasing causes signals to appear in the data with frequencies that never actually occurred. A common example is the effect that causes a wheel to appear to move backwards when seen on a television.

The Nyquist Theorem states that the sample rate must be more than twice the highest frequency in order to measure the signal accurately. If the sample rate is too slow, lower frequencies are created in the data that does not exist in the signal. This is shown in figure 3.4.



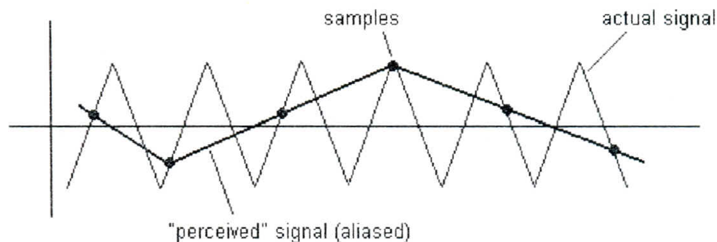


Figure 3.4: The Signal Diagram of Aliasing

The only way to ensure that aliasing does not occur is to remove high frequencies with an anti-aliasing filter which is a low-pass filter. This filtering must be done before the data is digitized since once it is captured in the PC there is no way to correct the data, or even to determine if aliasing happened.

3.4. SIGNAL CONDITIONING

Signal conditioning provides any required gain, isolation, noise rejection, offsetting, or linearization of the output of a transducer.

Gain is required in those cases in which the output id too small to be directly useful in a measurement or control system. The required gain is supplied by operational amplifiers or instrumentation amplifiers.

Offsetting is required when the level of a signal must be shifted by some predictable amount. Offsetting includes the conversion of one measurement scale to another, cold junction compensation for thermocouples and conversion of a voltage signal to a current signal for transmission purposes.

Linearization may be accomplished by digital or analog methods. In digital approach, the computer may linearize the readings by performing mathematical operations on them. Another computer approach is to convert each digitized value to a corresponding corrected value by using readings stored in computer memory. The linearized values are stored in look-up table. Analog linearization uses amplifiers and other circuits that have a nonlinear response that is complementary to the characteristic curve of the transducer. For example, an amplifier with a logarithmic response can be used to linearize a sensor with an exponential output.

3.5. ENVIRONMENTAL LIMITATIONS OF DATA ACQUISITION EQUIPMENT

A data acquisition system may not perform as expected unless it is suited to the environment in which it will be used. Since they measure sensors in all kinds of places, they can be subjected to different environments. Most systems are designed for benign environment like a PC requires. Some can handle extreme environments, but this makes them more expensive. The environmental extremes are temperature, humidity and water, dust and oil, shock and vibration, electrical noise and high voltage.

Temperature extremes may cause that the acquisition system does not work properly or stops functioning. They can be reduced by putting the device in a heated or cooled enclosures or building.

High humidity may cause corrosion and electrical conduction, and low humidity can increase the chance of electrostatic shock, which may cause momentary measurement errors or permanent damage. The way to avoid this extreme is to put the device in a heated enclosure or a sealed enclosure.
Dust and oil may cause electrical conduction and interfere with moving parts, such as fans. This protection requires placing the device in a protective enclosure or placing it remotely.
Shock and vibration may damage sensitive electronic parts or damage the enclosure. The protection can be provided by locating the device away from violent motion.
Electrical noise does not often cause a damage to the device, but reduce the measurement quality. Power line noise can come from motors and other electrical devices. High frequency noise can come from radio stations and other transmitters or the devices which are switch-mode. Electrical noise can be reduced by shielding sensor wires and by properly connecting the ground terminals. Using an enclosure does not provide much protection from noise because it is usually picked up by the sensor wires. Electrical noise can also be reduced by filtering. Proper design of inputs reduces suspectibility to high frequency radio sources.

3.6. SOFTWARE

One of the important things that affects the system performance directly is the software running on the data acquisition & control system. In these systems software is expected to provide with the following features:

Acquisition. Software is responsible for capturing analog inputs from sensors, or digital inputs from switches and other on-off devices. At especially high sampling rates, it may capture data in a burst. When capturing data in a burst, data is placed in a buffer in the data acquisition device and transferred to the PC after burst is complete. There is no display or data manipulation until the burst of data fills the buffer. Therefore burst acquisition is not suitable for control.

Data Manipulation. Software should provide for data manipulation and analysis functions such as scaling, trigonometry, arithmetic, logic and much more complex ones such as linearization, FFT, filtering etc.

Logging to disk. Captured data or at least non-volatile data may be written to disk.

Display. Data may be showed on the display in different forms such as x-y charts, different kinds of graphs or digital meters.

Control. Software should be able to operate the devices in the control system by turning them on and off through digital outputs.

There are quite different software options for data acquisition and control systems. The DAC systems my be classified from the programming aspects as the follows:

1. A conventional PC & a plug-in DAC device with no microprocessor.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

In this system, there is one program running on the PC and it is responsible for everything. This program has to run on one of the common platforms such as DOS, Windows or Unix and may probably have some difficulties in meeting the requirements, since these operating systems are not suitable for control applications. It should be noted that there are also some real-time operating systems developed for these purposes, however common data acquisition applications were not developed for these platforms.

2. A Desktop PC & a plug-in DAC device with its own microprocessor.
The difference is that there are two programs here that share the load. There must be also one more operating system running one the data acquisition device which may meet all the requirements in real-time. The other program provides only user communication, monitoring and storage of the data therefore the operating system on which this program runs does not have to be a specific one.

3. A special PC including the DAC functions.
This is the most preferable type of system from many aspects such as speed and size. The microprocessor and A/D converters are on the same board, which means faster data transfer rates and very compact structure. There is an operating system running on this system which is probably dedicated to data acquisition and control. However they are not common in industrial applications due to their high prices and less flexibility compared to the other types of data acquisition systems.

4. An embedded PC & a plug-in DAC device with no microprocessor.
The only difference from the type 1 is the difference between desktop and embedded computers described in chapter 2. On the other hand, the operating system is not a common one. This operating system is dedicated to data acquisition and control and user mostly has to develop an application program running on this platform.

User has the following advantages and disadvantages when he develops his own program using high or low level languages:

- It will be specific to the application and may fulfil all the requirements of that application in case that package programs are insufficient.
- The program includes the features which user needs only, which makes it smaller and faster than ready-made programs.
- The user knows very well his program, therefore he makes the possible changes in the program easily in future.
- It is flexible to develop a program with especially low-level languages which means that the program is limited by only user's imagination.
- Ready-made programs are usually expensive compared with those user develops.
- The only disadvantage may be that programming takes a lot of time and effort. Package programs often have a more user-friendly environment which makes the development period shorter, and they are also more reliable.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

CHAPTER 4

THE PARTS OF THE DEVELOPED EMBEDDED CONTROLLER

The developed embedded controller consists of four main parts:
1. The PC mainboard
2. The Digital I/O board
3. The Analog I/O board & Power Supply module
4. The Enclosure

4.1. THE PC MAINBOARD

The PC mainboard is Microdesign™ Powerdwarf 486/R which serves as a processor board in a PC/104 environment and provides for a fully ISA-compatible computer system [8]. It has different options for users, however the board chosen for this project has a 5x86 P75 CPU, 4MB DRAM, 2 MB FlashDisk, one parallel, two serial, keyboard, SVGA and LCD display interfaces. Powerdwarf single board computer also has an expansion slot according to the PC/104 norm. The digital I/O and analog input boards are plugged into this slot.

4.2. DIGITAL INPUT/OUTPUT BOARD

The digital I/O board was designed in order to provide basic digital inputs and outputs which are highly needed in control applications. The board has an address decoder circuit, two 82C55 programmable I/O ICs, 24 transistor&relay pairs, 6 TLP521-4 ICs, connectors and LEDs as indicators [9]. Figure 4.1 is the block diagram of the board.

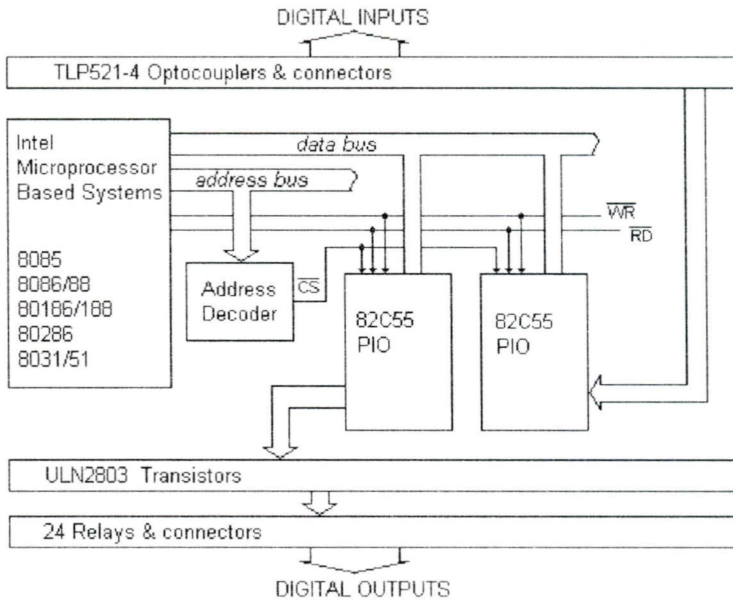


Figure 4.1: The Block Diagram of the Digital I/O Board

4.2.1. ADDRESS DECODER CIRCUIT

This part of the board is used to address the 8255 chips. The address decoder allocates an I/O region for communication between the CPU and 8255s. For this purpose, two 74'138 decoder/demultiplexer ICs were used. In order to do a data transfer between CPU and 82'55, first its CS (Chip Select) pin must be given LOW. For timing diagrams of these data transfer operations, please refer to the datasheet [9]. The following diagram shows the address decoder circuit.

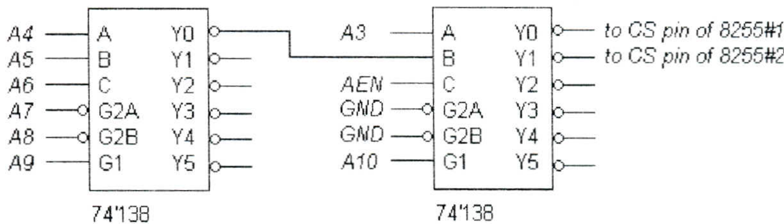


Figure 4.2: Address Decoder Circuit of the Digital I/O Board

As shown in figure 4.2, 8 address lines of the address bus and AEN signal are used to resolve the address. A1 and A0 signals are connected to the A1 and A0 pins of the 82'55 that provides a selection among the three ports of the chip internally. However A2 signal is not used due to some PCB problems, which costs wasting of 8 bytes of the address space. The following table best explains the working of this circuit.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

Inputs												Outputs	
A10	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**	**AE N**	**Y0**	**Y1**
H	H	L	L	L	L	L	L	x	x	x	L	L	H
H	H	L	L	L	L	L	H	x	x	x	L	H	L
Any other combination of these bits												H	H

Table 4.1: Input and Output States of the Address Decoder Circuit

As seen from the above table, an address word between 0x600 and 0x60F will activate the Y0 and Y1 outputs of the address decoder. At this point A1 and A0 signals decide which port of 82'55 is to be used. The following is a more comprehensive table which considers A1 and A0 signals and also explains the operations with these ports.

Address	*Description*
0x600	The states of the first group of digital inputs are obtained by reading from this port
0x601	The states of the second group of digital inputs are obtained by reading from this port
0x602	The states of the third group of digital inputs are obtained by reading from this port
0x603	The programming byte for the first 8255. No need to use it.
0x608	The first group of digital outputs are updated by writing to this port
0x609	The second group of digital outputs are updated by writing to this port
0x60A	The third group of digital outputs are updated by writing to this port
0x60B	The programming byte for the second 8255. 0x80 should be written

Table 4.2: Addressing of the Digital I/O Board

82C55 is a programmable I/O device therefore one of the chips on board should be programmed as all of its pins are input, and the other should be output in the same way. After reset, all ports of 82C55 are set to input mode therefore the first 82C55 does not need to be programmed. In order to program the second one, 0x80 is written into the port 0x60B. Table 4.3 is the description of the ports that the board uses.

Port 0x600	Digital I/O board . Output port A of 8255#1							
InA	I0.3	I0.4	I0.2	I0.5	I0.7	I0.0	I0.6	I0.1
	w	w	w	W	w	w	w	w
	D7	D6	D5	D4	D3	D2	D1	D0

Port 0x601	Digital I/O board . Input port B of 8255#1							
InB	I2.6	I2.7	I2.5	I2.4	I0.2	I1.7	I2.1	I2.0
	w	w	w	w	w	w	w	w
	D7	D6	D5	D4	D3	D2	D1	D0

Port 0x602	Digital I/O board . Input port C of 8255#1							
InC	I1.6	I2.3	I1.5	I1.0	I1.2	I1.3	I1.0	I1.4
	w	w	w	w	w	w	w	w
	D7	D6	D5	D4	D3	D2	D1	D0

Port 0x603	Digital I/O board . 8255#1 control port							
Control	1	0	0	1	1	0	1	1
	w	w	w	w	w	w	w	w
	D7	D6	D5	D4	D3	D2	D1	D0

Port 0x608	Digital I/O board . Output port A of 8255#2							
OutA	O0.3	O0.4	O0.2	O0.5	O0.7	O0.0	O0.6	O0.1
	r	r	r	r	r	r	r	r
	D7	D6	D5	D4	D3	D2	D1	D0

Port 0x609	Digital I/O board . Output port B of 8255#2							
OutB	O2.6	O2.7	O2.5	O2.4	02.2	O1.7	O2.1	O2.0
	r	r	r	r	r	r	r	r
	D7	D6	D5	D4	D3	D2	D1	D0

Port 0x60A	Digital I/O board . Output port C of 8255#2							
OutC	O1.6	O2.3	O1.5	O1.0	01.2	O1.3	O1.0	O1.4
	r	r	r	r	r	r	r	r
	D7	D6	D5	D4	D3	D2	D1	D0

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ

Port 0x60B	Digital I/O board . 8255#2 control port							
Control	1	0	0	0	0	0	0	0
	w	w	w	w	w	w	w	w
	D7	D6	D5	D4	D3	D2	D1	D0

Table 4.3: Explanation of the Ports Used by the Digital I/O Board

4.2.2. OPTO-ISOLATED INPUT

One of the popular protection methods for digital inputs is using opto-couplers. In order to provide an isolation between the PC and real world, TLP521-4 opto-coupler ICs were used on the digital I/O board. Figure 4.3 is the circuit diagram of the input layer.

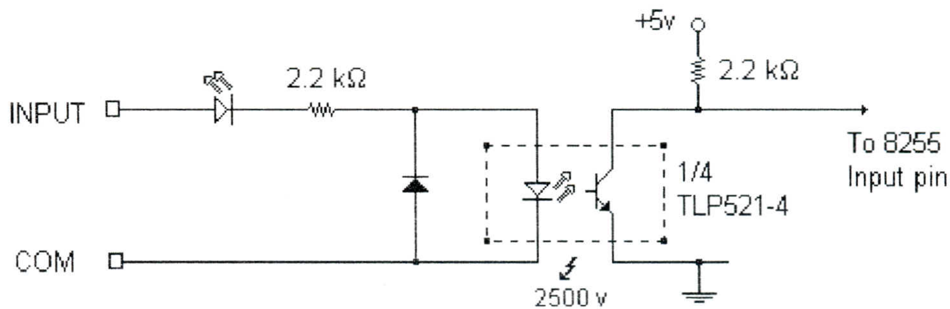


Figure 4.3: The Input Layer of Digital Inputs

This circuit provides an isolation up to 2500 volts and a voltage conversion which increases the noise margin. By means of this isolation, in abnormal cases although the input layer of the opto-couplers may get damaged, the computer will not be affected. According to table 4.4, in order for the computer to perceive "0" state, the input must be nominally within the range from 18 to 48 volts, and for "1" state, it must be from 0 to 8 volts. Applying a voltage continuously over 48 volts may damage the input, however it is able to resist up to peaks at very high levels without damaging.

Input Voltage (INPUT-COM)	**The state of the output**
0-8 volt	"1" state (~ +5 volt)
18-48 volt	"0" state (~ 0 volt)

Table 4.4: Voltage Levels of the Digital Inputs

4.2.3. RELAY OUTPUT

By means of the relays, the board gives user actually 24 pieces of computer-controlled switches.

Relay is an electrically controlled device that opens and closes electric contacts. The relays used on this board are Phoneix Contact micro relays. The contact rates of these relays are 6 Amps, 250 Volts and these rates are quite sufficient for driving many devices without a need for any other interface. The other reason of having chosen this relay is that it is very small in size that is important when designing compact products.

Relays also provides an isolation up to 1500 volts between the computer and the real world that gives a protection against possible hazardous electrical shocks.

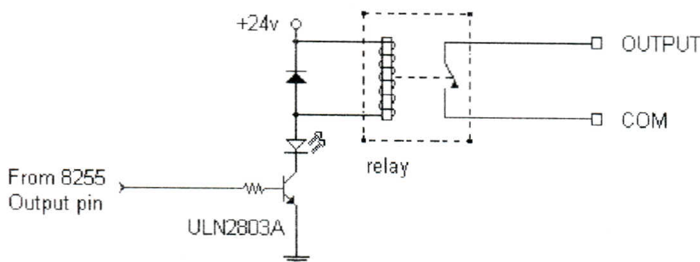


Figure 4.4: The Output Layer of Digital Outputs

4.2.4. CONNECTORS AND INDICATORS

24 relays and 24 opto inputs require 48+48 connection points on the enclosure and this means that there is a need for a very large enclosure to locate 96-pin connector. In order to reduce the size, and also considering the most common applications, inputs and outputs are grouped. 24 outputs are separated into three groups in a way that one pin of every relay are connected together internally and named as common. Therefore each of the group of 8 relays has a 9-pin connector. It is same for the inputs.

Therefore in an application, a group of relays has to be used for the devices which are similar with their input voltage levels. For example the common of the first group can be connected to the mains, in this case this group's relays can only drive the devices which are allowed to be supplied by mains.

There are also 48 LEDs (Light Emitting Diode) on the board to show the status of the I/Os.

4.3. ANALOG INPUT AND POWER SUPPLY BOARD

This board converts analog signals to digital signals so that we can measure many quantities such as temperature, pressure, flow, displacement, etc. The analog input board provides 16 analog inputs each of which has the input range from -10 to +10 volts. These inputs pass through two analog multiplexers and then to an op-amp for signal conditioning. Signal conditioning is nothing but scaling and filtering the signal. The board, like the digital I/O board, has an address decoder and latch, one A/D converter IC, two analog multiplexer

REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

ICs, opamps for signal conditioning, a DC/DC converter as the power supply and connectors. The block diagram of the analog board is shown below.

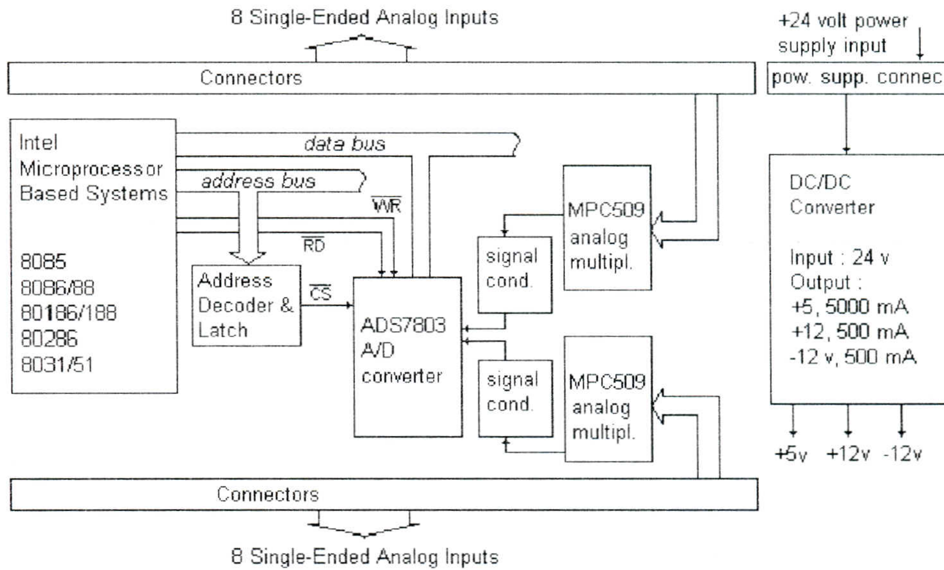


Figure 4.5: The Block Diagram of the Analog Input & Power Supply Board

4.3.1. ADDRESS DECODER & LATCH CIRCUIT

This part of the board is used to address the ADS7803 IC and to select the analog input channel. The address decoder allocates an I/O region for communication between the CPU and ADS7803. For this purpose, two 74'138 decoder/demultiplexer ICs and a 74'373 Latch IC were used.

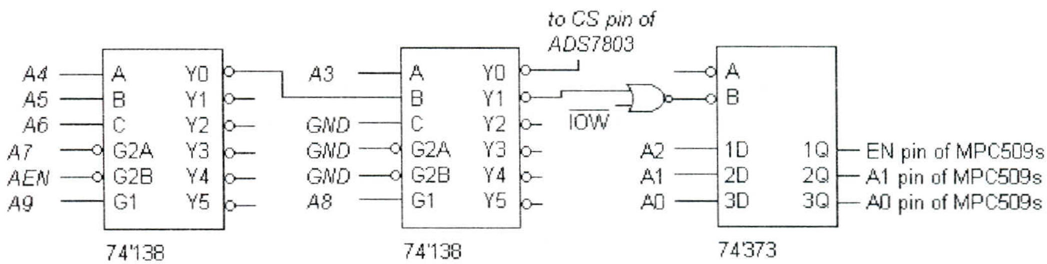


Figure 4.6: The Address Decoder of the Analog Input & Power Supply Board

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

Address	*Description*
0x300	Writing any value to this port selects channel 0 - 1 pair. Reading from this port is undefined
0x301	Writing any value to this port selects channel 2 - 3 pair. Reading from this port is undefined
0x302	Writing any value to this port selects channel 4 - 5 pair. Reading from this port is undefined
0x303	Writing any value to this port selects channel 6 - 7 pair. Reading from this port is undefined
0x304	Writing any value to this port selects channel 8 - 9 pair. Reading from this port is undefined
0x305	Writing any value to this port selects channel 10 - 11 pair. Reading from this port is undefined
0x306	Writing any value to this port selects channel 12 - 13 pair. Reading from this port is undefined
0x307	Writing any value to this port selects channel 14 - 15 pair. Reading from this port is undefined
0x308	Writing the value of 0x0 to this port starts conversion from the one channel of the pair selected before. Writing 0x1 starts conversion from the other channel.
0x308	After conversion the low byte of the result can be read from this port.
0x309	After conversion the high byte of the result can be read from this port.

Table 4.5 Explanation of the Ports Used by the Analog Input & Power Supply Board

4.3.2. Analog Multiplexer and Signal Conditioning Circuit

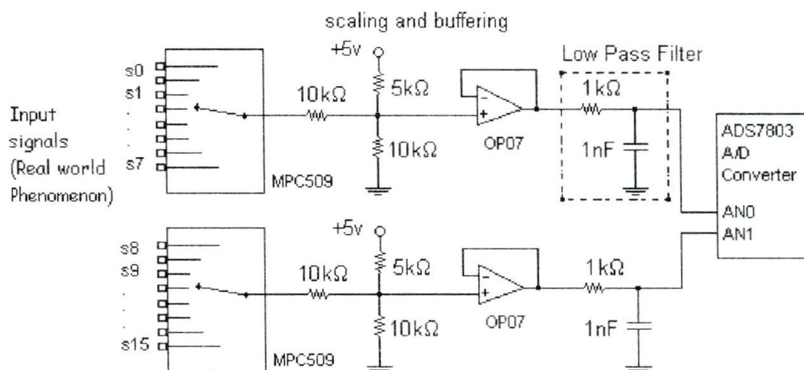


Figure 4.7: Analog Multiplexing and Signal Conditioning Circuit

The MPC509 analog multiplexer ICs multiplex 16 channels into an A/D converter. However the input signal needs to be scaled after the multiplexer in order to be able to

make the input range different from the input range of the A/D. The input range of ADS7803 is between 0 and 5 volts [10]. Most common range in the industrial environments when we consider transducers and other electronic equipment is (-10, +10 volt). The resistor network shown in the above diagram converts (0, +5v) analog voltage to (-10, +10v). After the resistor network, the input signal has to be buffered before going to the filter. OP07 buffers this signal, then the signal passes through a low-pass filter which does not reduce the bandwidth of the A/D. The 1k resistor in the low-pass filter also provides a protection for the analog input layer of the A/D converter.

4.3.3. DC/DC CONVERTER

To power the controller, Melcher 24IMR40-051212-2 DC/DC converter is used and it is located on the analog input board in order to reduce the size of the controller. The reason of using this converter is that it has been developed for powering commercial type of electronic circuits such as telephone systems components, industrial controllers and small appliances. The input voltage is 24 volts and it provides +5, +12, -12 volts at 6000mA, 500mA, 500mA current levels respectively. +5 volt output is the main supply voltage for the Powerdwarf computer. The computer does not need any other supply input. +12 and -12 volt outputs are necessary for the multiplexers and opamps. For further information about the converter please refer to the data-sheet [11].

4.3.4. CONNECTORS

16 analog inputs require 32 connection points on the enclosure. Each analog input has a signal and a ground line. All of the ground points are connected to the system's ground internally, however the reason of using separate ground points for all inputs is to prevent ground loop errors. There is also a connector on this board for power input.

4.4. ENCLOSURE

Enclosure covers all the hardware equipment in order to provide a protection and ease of montage. The enclosure of the embedded controller developed in this project is made by aluminum and has 250x170x70 mm dimensions.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

CHAPTER 5

THE SOFTWARE OF THE DEVELOPED EMBEDDED CONTROLLER

5.1. ACCESSING THE HARDWARE

In this section, the ways of accessing the designed digital and analog boards will be explained. Example codes are all written in C, since it provides a flexible programming.

5.1.1. USING THE DIGITAL I/O BOARD

The below table describes the related ports needed to control the board.

Address	*Description*
0x600	The states of the first group of digital inputs are obtained by reading from this port
0x601	The states of the second group of digital inputs are obtained by reading from this port
0x602	The states of the third group of digital inputs are obtained by reading from this port
0x603	The programming byte for the first 8255. No need to use it.
0x608	The first group of digital outputs are updated by writing to this port
0x609	The second group of digital outputs are updated by writing to this port
0x60A	The third group of digital outputs are updated by writing to this port
0x60B	The programming byte for the second 8255. The value of 0x80 should be written

Table 5.1 Explanations of the Ports Used by the Digital I/O Board

When the system is booted, all pins of the 8255s are defined as input. Therefore the second 8255 should be programmed with the following command to provide that all of its pins are output for driving relays.

```
outportb( 0x60B, 0x80 );
```

The first 8255 does not need to be programmed, so the above command is all the initialization of the board. After the initialization, inputs and outputs are controlled by accesing the related ports. For example, writing 0x1 to port 0x608 makes one of the relays to close its contact.

Both inputs and outputs are grouped as described in section 4.2.4. The Outputs have three groups of relays, namely O0, O1 and O2 and outputs of the first group are named as O0.0, O0.1, O0.2, ..., outputs of the second group are named as O1.0, O1.1, O1.2, ..., and so on. This is same for the inputs, for example I0.0, I1.0 and I2.0 are the first bits of the three groups of digital inputs.

When one of the relays is needed to be activated, the related bit must be set to "1" and one byte must be written to the related port. Since there is not a peer to peer logical

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

connection between the bits and the outputs, that means writing 0x1 to the port 0x608 in the above example activates the second relay not the first one, a kind of mapping of the bits is necessary. This non-matching was the result of some physical difficulties with which were confronted in preparing the printed circuit board. However, this difficulty may easily be overcome by software. C gives the ability to handle each bit of a byte separately. The following structures provides an output mapping:

```
struct BITS{
       int o0_3 : 1 ;
       int o0_4 : 1 ;
       int o0_2 : 1 ;
       int o0_5 : 1 ;
       int o0_7 : 1 ;
       int o0_0 : 1 ;
       int o0_6 : 1 ;
       int o0_1 : 1 ;

       int o2_6 : 1 ;
       int o2_7 : 1 ;
       int o2_5 : 1 ;
       int o2_4 : 1 ;
       int o2_2 : 1 ;
       int o1_7 : 1 ;
       int o2_1 : 1 ;
       int o2_0 : 1 ;

       int o1_6 : 1 ;
       int o2_3 : 1 ;
       int o1_5 : 1 ;
       int o1_0 : 1 ;
       int o1_2 : 1 ;
       int o1_3 : 1 ;
       int o1_1 : 1 ;
       int o1_4 : 1 ;
};

struct BYTES {
       char portA;
       char portB;
       char portC;
};

union {
       struct BITS bit;
       struct BYTES byte;
} out;
```

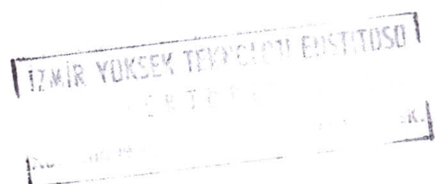
İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve ... Daire Bşk.

The following code explains how the outputs O0.3 and O2.1 are set to "1":

```
out.byte.portA = out.byte.portB = out.byte.portC =0;

out.bit.o2_1 =1; /* set the 2th output of the second port to 1 */
out.bit.o0_3 =1; /* set the 4th output of the first port to 1 */
/* Although the related bits are changed, the outputs does not
change until the ports are refreshed */

/* refresh the ports */
outportb(0x608, out.byte.portA);
outportb(0x609, out.byte.portB);
outportb(0x60A, out.byte.portC);
```

This is the same for the digital inputs. The following structures provides an input mapping.

```
struct BITS{
      int i0_3 : 1 ;
      int i0_4 : 1 ;
      int i0_2 : 1 ;
      int i0_5 : 1 ;
      int i0_7 : 1 ;
      int i0_0 : 1 ;
      int i0_6 : 1 ;
      int i0_1 : 1 ;

      int i2_6 : 1 ;
      int i2_7 : 1 ;
      int i2_5 : 1 ;
      int i2_4 : 1 ;
      int i2_2 : 1 ;
      int i1_7 : 1 ;
      int i2_1 : 1 ;
      int i2_0 : 1 ;

      int i1_6 : 1 ;
      int i2_3 : 1 ;
      int i1_5 : 1 ;
      int i1_0 : 1 ;
      int i1_2 : 1 ;
      int i1_3 : 1 ;
      int i1_1 : 1 ;
      int i1_4 : 1 ;
};

struct BYTES {
      char portA;
      char portB;
      char portC;
};
```



```
union {
      struct BITS bit;
      struct BYTES byte;
} in;
```

The following code is an example that reads the input I1.2 and performs a function if it is "1".

```
/* read the ports */
outportb(0x600, in.byte.portA);
outportb(0x601, in.byte.portB);
outportb(0x602, in.byte.portC);

if( in.bit.i1_2 ) function1();
```

On the other hand, the real-time executive provides some useful system calls for handling digital inputs and outputs, which will be described later.

5.1.2. USING THE ANALOG INPUT BOARD

The below table describes the related ports needed to control the analog input board.

Address	*Description*
0x300	Writing any value to this port selects channel 0 - 1 pair. Reading from this port is undefined
0x301	Writing any value to this port selects channel 2 - 3 pair. Reading from this port is undefined
0x302	Writing any value to this port selects channel 4 - 5 pair. Reading from this port is undefined
0x303	Writing any value to this port selects channel 6 - 7 pair. Reading from this port is undefined
0x304	Writing any value to this port selects channel 8 - 9 pair. Reading from this port is undefined
0x305	Writing any value to this port selects channel 10 - 11 pair. Reading from this port is undefined
0x306	Writing any value to this port selects channel 12 - 13 pair. Reading from this port is undefined
0x307	Writing any value to this port selects channel 14 - 15 pair. Reading from this port is undefined
0x308	Writing the value of 0x0 to this port starts conversion from the one channel of the pair selected before. Writing 0x1 starts conversion from the other channel.
0x308	After conversion the low byte of the result can be read from this port.
0x309	After conversion the high byte of the result can be read from this port.

Table 5.2 Explanation of the Ports Used by the Analog Input & Power Supply Board

As seen from the above table, the I/O region between 0x300 and 0x309 are reserved for the analog input board. The board has sixteen analog channels multiplexed by analog multiplexers. Therefore, in order to make a read operation, first the analog channel pair must be selected by writing the ports between 0x300 and 0x307. The channel which will be read from this selected pair is defined when the conversion is started. Writing 0x0 to the port 0x308 selects the first analog channel of the pre-selected channel pair and also starts the conversion. Writing 0x1 to the port 0x308 selects the second analog channel of

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

the pre-selected channel pair and also starts the conversion. After conversion is started, the A/D converter completes the conversion process about 20μs later. For further information about how the A/D operates please refer to ADS7803 data sheet [10].

After conversion is completed, the result value can be read from the ports 0x308 and 0x309. Since the A/D has twelve bits resolution, but 8-bit data transfer is used with this system, two consecutive read operations are needed to get the 12-bit data. The low byte of the value is read from the port 0x308, the high 4-bit value is read from the port 0x309. The following tables describe these ports.

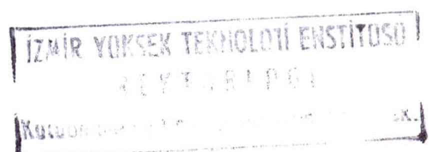
Port 0x308	Analog Input Board							
Bit	7	6	5	4	3	2	1	0
	w/r	w/r	w/r	w/r	w/r	w/r	w/r	w/r
	D7	D6	D5	D4	D3	D2	D1	D0

Port 0x309	Analog Input Board							
Bit	7	6	5	4	3	2	1	0
	R	r	r	R	r	r	r	r
	0	0	0	0	D11	D10	D9	D8

Table 5.3 Explanation of the Reading Operations from Analog to Digital Converter

Therefore, in order to obtain the combined value from these ports the following structure can be used.

```
float analog_value[16];          /* array for 16 analog channels */

struct NIBLES {

        char high_byte;
        char low_byte;
};

union {

        struct NIBLES nible;
        short combined;

} value;
```



İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

```
outportb(0x300,  0x0);      /* select the first pair of analog
channels*/

outportb(0x308,  0x1);    /* select the second analog channel and
start the conversion at the same time */

u_delay(25);    /*   wait 25µs in order for conversion to be
completed */

value.nible.low_byte = inportb(0x308);    /* read the low byte */
value.nible.high_byte = inportb(0x309);   /* read the high byte */

printf("Analog value = %u \n", value.combined);       /* now the
result in the variable
                                             value.combined  */
```

Variable value.combined which digitally represents the analog signal can only be between 0 and 4095, which means it can take 4096 different values. The analog input range is from –10 to +10 volts, therefore –10 volts is converted to 0, and +10 volts is converted to 4095 theoretically. The following simple formula can be used so as to obtain the input voltage value from the digital value.

```
analog_value[1] = -10.0 + (float)value.combined * 20.0 / 4096.0 ;
```

On the other hand, the above equation is not always valid due to calibration problems. There are many factors that affect the calibration such as temperature and other environmental conditions, which makes it difficult to get the true analog value.

Assuming that the linearity is good as it should be, two values, the offset and the gain should be known so as to correct the analog value.

The real-time executive uses *readAnalog* internal function to acquire the analog data. This function is called from timer interrupt service routine at every 62.5 µs and it performs analog to digital conversion algorithm expressed above, from channel 0 to 15 respectively every time it is called. This means that all analog channels are sampled once at every one millisecond. Therefore the maximum sampling rate for a channel is 1kHz. If a sampling procedure is defined in "load.dac" file for an analog channel, *readAnalog* function also puts the data into the defined pipe.

5.2. EXPLANATION OF THE DEVELOPED REAL TIME EXECUTIVE

As the last step, a real-time executive was developed for the embedded controller. This executive runs on DOS and provides user with a real-time multitasking environment. An executive was preferred to a new operating system so that software development could be easier and faster. This executive uses all useful features of DOS such as I/O handling, and

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
Kütüphane ve Dokümantasyon Daire Bşk.

after it starts to run, takes over the CPU and all the system management. The only disadvantage of it is that a DOS has to be used with every embedded controller.

This executive schedules and performs many tasks concurrently and runs in real-time without losing data. This executive controls the analog sampling hardware, and the digital board. It is responsible for reading the digital inputs, updating the digital outputs and controlling the analog hardware including acquiring data at predefined sampling rate and placing them into predefined buffers and moreover scheduling the tasks meeting the real-time requirements [12].

The executive was compiled and linked with Microsoft Quick C 2.01 and Microsoft Quick C Linker 4.07 and it consists of the following files:

custom.h : The definitions of the max numbers for tasks, pipes, variables etc.
dacos.h : All prototypes for functions and data structures used by the real-time system.
calls.h : The definitions for the system calls.
defs.asm : Some definitions for the assembly functions.
dacos.c : Almost all of the function implementations of the real-time executive.
kernel.asm : The assembly language routines providing task switching which cannot be done well within C.
main.c : Includes main function which does the initialization of the executive and shell functions for interacting with user.

To make the executive the following lines are used:

```
qcl -c -AL -Zi -Ox -Aw -Gs dacos.c
qcl -c -AL -Zi -Ox -Aw -Gs main.c
qcl -c -AL -Zi -Ox -Aw -Gs kernel.asm

link /co /stack:16384 dacos.obj kernel.obj main.obj, dac.exe;
```

Tasks are converted from executable files that are generated for DOS. The converter tool is quite simple and it is available. One can write as many processes as he wants using C compilers what he can do in a process is limited only by his imagination. How a process can be prepared will be explained comprehensively in section 5.2.6.

The executive supports code sharing in two ways. The first one is that all processes may share the system calls existing in the memory. How a process can use the system calls will be explained later. By the way, the object code of a process does not have to include the common functions used by many processes and therefore code duplications are prevented, which results in lower memory and disk usage. The second method of preventing the duplications is used for processes, considering a need in data acquisition applications that is using the same algorithm for many channels. For instance, a low-pass filter algorithm or process may be used for all of the analog channels. In this case, the

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

low pass program (i.e. "lowpass.bin") is loaded into the memory once and all processes use the same object code, however each process has its own stack which stores the register values, variables and parameters of that process.

5.2.1. SCHEDULER

Many different performance criteria have been proposed for different types of real-time systems. However this diversity of criteria sometimes make it difficult to compare the scheduling algorithms. Another difficulty is in the variety of the characteristics of the tasks one can face. Tasks are characterized by their deadlines, execution times, resource requirements, criticalness or importance levels. For periodic tasks, the period is important property, whereas for aperiodic tasks, the deadline becomes important. Different scheduling methods are discussed in chapter 2. In this project, round robin scheduling was used [13]. Round-robin scheduling algorithm is easy to implement and widely used in many systems. In this scheduling, each executable task is assigned a fixed time quantum called a time slice. Time slice is statically defined as 500 μs for each process, which means that currently running task is preempted at every time the scheduler is called by the clock interrupt, giving the next task in the queue a chance to run.

Scheduler is called by the timer interrupt service routine and also by some system calls such as *get_pipe, get_pipe_buf* etc. These functions call *waitEvent* function and cause that the running task to be suspended until the related event has been occurred.

5.2.2. INTERPROCESS COMMUNICATION

For interprocess communication, pipes and variables can be used. [8] This can be explained with an example. The followings are two tasks one of which tells another to terminate itself. For this purpose, task A creates a pipe for communication with the other task, and puts the string "STOP" into this pipe. Task B continuously controls this pipe if any command is available for itself, and if this command is saying "STOP", then it calls *terminate* function and stops its execution.

```
/* TASK A*/
# include "calls.h"
# include "task.h"

void main()
{
  # include "init.c"

  TPIPE com_buffer_1;

  param_process(0);

  create_pipe(&com_buffer_1, 30, "COM1");
  if(com_buffer_1.p==NULL) terminate();    // failed to create a
pipe

  while(1){
```

```
        if(a_condition) put_pipe_buf(com_buffer_1, 4, "STOP");
        .
        .
        .
    }
}


/* TASK B*/
# include "calls.h"
# include "task.h"

void main()
{
  # include "init.c"

  TPIPE *com_buffer=NULL;
  short size;
  short buf[30];

  param_process(0);

  open_pipe(&com_buffer, "COM1");
  if(com_buffer==NULL) terminate(); // failed to open the pipe

  while(1){

        sizeof_pipe(combuffer1, &size);

        if(size>=4) {
            get_pipe_buf(com_buffer_1, 4, buf);
            if(buf[0]=='S'&&buf[1]=='T'&&buf[2]=='O'&&buf[3]=='P')
                terminate();
        }
        .
        .
        .
  }
}
```

5.2.3. PREPARING THE SYSTEM CONFIGURATION FILE

When the executive is started, it reads sampling procedures and definitions for processes, pipes and variables from the file “load.dac”. There are five commands that can be used in this file:

pipe.name[string].size[unsigned integer].

This command defines a pipe with the given name and size. A pipe is nothing but a FIFO buffer and the maximum number of the pipes which can be used in a program is defined

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

in "custom.h" header file. The length of *name* cannot be longer than 15 characters and *size* has the type unsigned integer. Pipes are used for placing the sampled analog data or other data produced by processes and also for interprocess communication.

var.name[string].initial_value[integer]

This command defines a variable with the given name and puts the initial_value in that variable. The length of *name* cannot be longer than 15 characters and *size* has the type integer. Variables can be used for interprocess communication like pipes.

sample.channel_number[integer0-15].sampling_period[unsigned integer].pipe_name[string].

Defines a sampling procedure for the given analog channel. The channel number must be between 0 and 15, since there are 16 analog channels in this system. The sampling period tells the executive to take one sample from the channel at every *sampling_period* milliseconds and to put the sample into the pipe with the name *pipe_name*. This pipe must be defined before it is used in this command, otherwise an error will occur. For example the command "*sample.0.10.temp2.*" means that the analog channel 0 will be sampled at every 10 ms and these samples will be placed in the pipe with the name "temp2".

pro.name[string].parameter1.parameter2.parameter3...

This command defines a process. The executive adds '.bin' extension to the end of *name* and looks for a file with this name in the directory \procs. If the file does not exist an error occurs and execution is stopped. The other parameters are the pipes and variables which will be passed to the process. The number of the parameters is not constant and was not limited. The type of the parameters written in this line and those the process express should match, which means if the first parameter is a pipe, the first parameter that the process sends with *param_process* function must be a pipe, otherwise unexpected errors may occur.

timer.period[unsigned int].name.parameter1.parameter2.parameter3...

Timer function definition is same as the process definition except for the additional *period* parameter. This period specifies the period with which the function is called. The period is in milliseconds and must be between 1 and 30,000.

5.2.4. System Calls

System calls are functions which are used by the executive and other processes in order for them to do the basic operations of the embedded system such as reading and writing operations to buffers, control of the hardware etc. All these functions are defined in the header file "dacos.h" and implemented in "dacos.c". Therefore, once the real-time executive is run, all these functions are loaded into the memory as well, which means one copy of each function is available in the memory after running the executive.

Therefore each process may share the same functions existing in the memory. When a binary code of a process is generated, it does not need to be linked with the codes of the common functions existing already in the memory such as get_pipe, put_pipe, set_out, reset_out etc, printk etc. It may contain only four-byte address pointers to these functions instead of the whole binary code of them. This way preventing the duplications, obviously results in a lower memory and disk usage [14].

The only initialization that the process has to do before starting is to get the addresses of the system call functions. Since the executive is compiled with the large-memory option, all function pointers are defined as 'far' that means each of them consists of four bytes. The following is the *_get_address* function of the executive. The parameters of this function are the system call number whose address is wanted to learn and a pointer to the pointer to that function.

```
#define SLEEPTASK     0
#define TERMINATE     1
#define PARAMPROCESS  2
#define PUTPIPE       3
 .
 .
 .

void far _get_address(int call_number, void (far pascal
**func)())
{

  switch(call_number){

     case  SLEEPTASK:   *func=(void far *)sleepTask;       break;
     case  TERMINATE:   *func=(void far *)terminate;       break;
     case  PARAMPROCESS: *func=(void far *)param_process; break;
     case  PUTPIPE:   *func=(void far *)put_pipe;    break;
     .
     .
     .
     default:   *func=NULL;
  }
}
```

In order for a process to call the function *_get_address*, first it has to know the address of this function. Then it gets the addresses of the other functions using *_get_address* function. The executive places the address of *_get_address* function into a location in the interrupt vector table which is actually reserved for interrupt 0x60. The interrupt 0x60 is a software interrupt reserved for users, therefore it is safe to use this known memory location for transferring the function address. Therefore the four memory bytes between 0x180 (0x60 * 4) and 0x183 keeps the address of the *_get_address* function. The process

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

must get this address to call the function. The following code shows how the executive places the function address into the interrupt vector table.

```
int far * pint= 0x60*4;
unsigned cs, ip;

union{
      struct {
                  int cs;
                  int ip;
      }h;

      long csip;
 }u;


u.csip=(long)_get_address;

*(pint+1)=u.h.ip;
*(pint)=u.h.cs;
```

How a process can use the system calls will be explained in the following section. Here, the system calls will be explained briefly.

*_get_address(int call_number, void (far pascal **func)()).*

This call is used to obtain the start address of the function with the number *call_number*.

sleep_task(int ticks).

Sleeps the currently executed task during the time of ticks. Each tick is 500 µs.

terminate().

Terminates the currently executed task, which means it is never given a chance to run. All of the code and stack of that task are emptied in the memory as well.

printk(char *string, int c).

This function prints the given string and integer on the screen. The string is not formatted like the standart printf function and only one integer is allowed to be written.

param_process(int num, ...).

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon D.B.

This function is used by processes to get their parameters from the executive.

*create_pipe(TPIPE *pipe, unsigned size).*

A pipe is nothing but a memory buffer. It is represented with a special structure variable TPIPE. This function creates a pipe with *size* of items each of which can store two bytes short integer.

*put_pipe(TPIPE *out, short a).*

Puts the variable *a* into the pipe *out*. The item is placed at the end of the buffer unless it is entirely full of data.

*get_pipe(TPIPE *in, short *a).*

Gets the first item in the pipe if available. If there is no item in the buffer, then *waitEvent* function is called to suspend the task until the pipe has an item. When *waitEvent* is executed, it calls *scheduler*, therefore the execution of *get_pipe* is interrupted until the related event has occurred. Before *waitEvent* is called, the variable *blocking_process* of TPIPE structure is set to one so that *put_pipe* function may understand that there is a process waiting for this event.

*sizeof_pipe(TPIPE *in, short *a).*

This function is called to learn the number of items in a given pipe. The return value is placed into the pointer variable.

*clear_pipe(TPIPE *in).*

Empties a given pipe.

*get_pipe_buf(TPIPE *in, unsigned short n, short *buffer).*

Processes often need the data more than one in order to perform an algorithm on them. For example an average process need a number of data to calculate the average. In this case, using *get_pipe* function to get the data is not efficient, since this way a process obtains the data one by one, which means time-consuming anyway. *get_pipe_buf* function provides obtaining a number of data at once. Like *get_pipe* function, get_pipe_buf calls *waitEvent* if there is no the given number of data in pipe and causes

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

that the scheduler suspends the task until the pipe is filled with at least that number of data.

*put_pipe_buf(TPIPE *out, unsigned short n, short *buffer).*

Puts multiple items of data into the given pipe. When there are items more than one, this function should be preferred to calling *put_pipe* many times.

*open_pipe(TPIPE **pipe, char s[]).*

This function is used to handle pipes by giving their names. Using this function, processes and timer functions may handle any pipe which is not sent as a parameter by the executive, so long as they know the name of that pipe. For example, let the following line be a pipe definition in "load.dac". The name of this pipe is temperature1.

pipe.temperature1.512

The following code puts a value into the pipe whose name is "temperature1", which may be a part of a process.

```
TPIPE *anypipe;

open_pipe(&anypipe, "temperature1");

put_pipe(anypipe, 3);
```

*open_variable(TVARIABLE **variable, char s[]).*

Provides another way for handling variables like *open_pipe* function.

set_out(char bit).

This function sets the given single output. The variable *bit* can be between 0 and 23, since there are 24 outputs in this system.

reset_out(char bit).

Resets the given output. The variable *bit* can be between 0 and 23, since there are 24 outputs in this system.

toggle_out(char bit).

Toggles the given output that means the new state of the output will be the complementary of the previous state.

clear_wdt().

If one of the processes stops executing due to any problem, the system must be informed about this unexpected case. To do this, a counter for each process is incremented at every timer interrupt tick and if one of these counters exceeds a certain value, the executive concludes that the related process has a problem. In order to prevent overflowing of their counters, processes must use *clear_wdt* function anywhere in the loop.

5.2.5. MAKING PROCESSES

The processes that can be executed by this executive should be compiled and linked in a different way. Borland C++ 4.5 compiler, linker and an additional utility program that converts executable files to binary files were used to make the processes in this project. User should follow the following steps to make a process;

An appropriate C source code is written using any editor program. How this source code is prepared will be explained later. Then the source code should be compiled with the following command:

```
C:\> bcc  -c  -ml   average.c
```

After a successful compilation, the output file proc.obj should be linked as follows without using any library:

```
C:\> tlink average.obj
```

Since no library is used by the linker, the linking process will generate the executable file but give a warning message "No stack" that means it cannot be run on DOS. Attempting to run this exe file results in crash. The compiler and linker add some extra code such as an exe signature, a relocation table into the object and executable files which are not necessary for the real-time executive, since it creates the necessary stacks itself for the processes. The only needed thing is the binary code of the functions implemented in the source code. Therefore the needed binary code should be selected from the exe file. For this purpose, a small program exeproc.exe can be used. This program selects the function codes from the file which is in DOS exe format and generates a binary file. To do this the following command is used:

```
C:\> exeproc  average.exe  /c
```

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

If all these steps were successful, the result file proc.bin is available to be used by the real-time executive.

A process probably needs to use the system calls of the real-time executive, for instance to access the hardware, as it may do everything by itself. However using the systems calls is faster and an efficient way. How a process can be created and how the source code is organized can be explained best with a sample C source code. The following is a process which reads n bytes from an input buffer and writes the average of them into an output buffer in infinite loop. This task may be used as a simple low-pass filter for analog input channels.

/ average.cpp: takes the average of a group of data in input pipe and puts the result into output pipe */*

```
1       # include "task.h"

2       # define PARAMPROCESS     5
3       # define PUTPIPE          7
4       # define GETPIPEBUF       15

5       void main()
6       {
7         typedef void (far *CALL)(...);

8         TPIPE *in, *out;
9         TVARIABLE *n;
10        short buffer[50];
11        int i;
12        unsigned int total;
13        int far * pint= (int far *)(0x60*4);
14        unsigned cs, ip;

15        CALL _get_address;
16        CALL param_process;
17        CALL put_pipe;
18        CALL get_pipe_buf;

19        ip= *(pint);
20        cs= *(pint+1);

21       _get_address= (CALL)( cs* 0x10000 + ip);
22       _get_address(PUTPIPE, &put_pipe);
23       _get_address(GETPIPEBUF, &get_pipe_buf);
24       _get_address(PARAMPROCESS, &param_process);

25       param_process(3, &in, &n, &out);

26       while(1){    // go to multitasking

27             get_pipe_buf(in, n->v, buffer);
```

```
        /* the task is blocked until the pipe 'in' is filled
        with                              n->v bytes of data */

28      for(i=0, total=0; i< n->v; i++) total+=buffer[i];

29      put_pipe(out, (short)(total/n->v));

30    }

31  }
```

The header "task.h" has the definitions for pipe and variable, therefore first it should be included in order to be able to use pipes and variables. The process must do two initializing operations; obtaining the addresses of system calls which are needed and obtaining the program parameters. The addresses should be obtained first so as to get the parameters, since the function that provides the parameters is also a system call. Line 7 is a type definition for using function pointers. Line 15, 16, 17 and 18 are the definitions of the function pointers to the system calls which will be used in this process. The types of the parameters of the process in this system are limited to two: pipe and variable. Therefore the definition of the parameters, which also must be pointers to the pipe and variable structures should be placed at the beginning of the main function. (Line 8, 9) Then transferring the addresses should start with obtaining the address of *_get_address* function, since all the other addresses will then be provided by this function. It is obvious that in order to be able to learn the addresses of the functions existing in the memory, at least one function's address should be known which is the function of providing the other addresses and this function is *_get_address* here. *_get_address* is implemented in the file "main.c" of the executive and its start address in the memory is placed into the interrupt vector table as mentioned before. Thus, line 13, 19, 20 and 21 show how the address of *_get_address* is obtained. The following lines 22, 23, 24 and 25 show obtaining of the other system calls' addresses. *_param_process* is also another important function which provides the parameters necessary for data transferring between the executive and itself. A parameter can only be a pipe or a variable and all these pipes, variables and the parameters sent to the processes are defined in the system configuration file "load.dac". This will be explained comprehensively in section 5.2.4.

The system call *_param_process* sends the parameters defined in "load.dac" to the process by which it is called. For instance, the following is a valid command that defines a process and the parameters which will be passed to it. It should be noted that all the parameters passed to the process must be defined before that process' definition.

pro.average.inpipe.var.outpipe.

The pipe *inpipe* is filled by the executive or another process. The process *average* reads from this pipe and calculates the average of the number of data specified by the variable *var*, and places the result into the pipe *outpipe*. There can be number of average processes all of which share the same code, however only the parameters passed by the executive

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
R E K T Ö R L Ü Ğ Ü
Kütüphane ve Dokümantasyon Daire Bşk.

differ. This is called code-sharing. As a result, the process reads its parameters by calling the *param_process*. The function *param_process* shown below accepts variable number of parameters provided that the first parameter should indicate the number of variables.

```
void far param_process(int num, ...)
{
  int i;
  Task far *task=currentTask;
  Timer far *timer=currentTimer;
  void far **argument;

  va_list ap;
  va_start(ap, num);

  for(i=0; i<num; i++) {

    argument=(void far **)va_arg(ap, void far **);

    if(currentTimer) *argument=timer->param.p[i];
     else *argument=task->param.p[i];
  }

}
```

When the function is called, it gets the parameters from the task or timer structure related to that process or timer function, and places them into the parameters which are actually pointers to the pointers to the pipe and variables existing in the memory. In other words, the function does not return any value, but uses the pointers for parameter transferring. Line 25 shows the usage of the function. It should be noted that there are three parameters, so the first parameter is 3 and the other parameters are the addresses of the pointer variables defined in line 8 and 9.

Another parameter transferring option is using *open_pipe* and *open_variable* functions. These functions provide handling pipes and variables by using their names. Please refer to section 5.2.4 for a detailed description of them.

After the initialization part, the algorithm may take place. The algorithm is generally in infinite loop, but remember that it is allowed to run no more than 500 μs due to multitasking. In this example given here, the algorithm is very simple and consists of three lines. In line 27, *get_pipe_buf* function is called to obtain n->v bytes of data from *inpipe*. When this function is executed it first looks if the given number of data is available in the input pipe, if so it gets the data and places them into the buffer defined in line 10, otherwise it suspends the process, calls the scheduler and consequently CPU is given to the other processes until the pipe has the data. In line 28 the sum of the data in the buffer is calculated so as to obtain the average of them. The last step is calling *put_pipe* function in order to place the result into *outpipe*.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

5.2.6. MAKING TIMER FUNCTIONS

Preparing of timer functions is almost same as the processes, but the only thing that must be considered is that these functions cannot be interrupted by the scheduler, therefore it must not include a loop or it must not be too long in order not to damage the integrity of the system. Like a process, a timer function is compiled and linked with the same methods. To explain how a timer function is prepared, the following sample will be used. The only job of this function is to toggle one of the digital outputs, which may be used for flashing of a lamp for alarming.

```
/* alarm.cpp : toggles the specified output bit */

1       # include "calls.h"
2       # include "task.h"

3       void main()
4       {
5               # include "init.c"

6               TVARIABLE *bit;

7               param_process(1, &bit);

8               toggle_out(bit->v);

9       }
```

The header file "init.c" contains all the initialization definitions and commands seen in lines from 13 to 24 of "average.cpp". The only parameter of the function specifies which relay is going to be toggled. Parameter transferring is done with *param_process* function again, and unlike processes it is done everytime the function is called, with all the other initializations, which spends an useless time. A timer function is defined in "load.dac" as follows:

timer.500.alarm.bitnumber.

bitnumber is a variable, and as seen in the above definition, other than the function parameters, there is one more parameter which specifies the period in which the function is called. The period is given in milliseconds. It should be noted that period must be between 1ms and 30,000 ms.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütü... Daire Bşk.

CHAPTER 6

CONCLUSION, FUTURE WORK

In this project, a new product "embedded controller" was developed for data acquisition and control applications. Considering the new developments in computer technology and the rising demand for computers in industrial environments, the controller was designed with a PC which is in PC/104 format and has rather high performance for time-critical applications. In Chapter 2, the definition of the important terms such as "embedded controller", "real-time" and the factors taken into account when designing the controller have been described. Chapter 3 gives the fundamentals of data acquisition and control, which will be the basis of the design of the controller.

In order to provide data acquisition and control operations the analog signal sampling board and the digital input/output board were prepared and connected to the PC. All analog inputs, digital inputs and outputs are accessible by the connectors situated on the enclosure. The designed boards are described comprehensively in chapter 4. This controller may measure many quantities such as displacement, pressure temperature at quite fast sampling rates, may acquire digital signals and perform basic on/off operations with relay modules. Along with this hardware, user is given a suitable software which runs in real-time and provides a multitasking environment. This software was developed taking into account many requirements of data acquisition applications so that it helps user with controlling the hardware and developing an application. User is completely free when developing an application, which means that he may use all advantages of a desktop PC such as computational power, graphical interface, flexibility of programming etc. The developed software is explained in chapter 5.

Such a system is feasible for many applications in different areas when we consider its performance-cost ratio. However it is expected to be used especially in industrial environments.

The advantages of this embedded controller can be listed as follows:

1. Performance-cost ratio is relatively higher than the similar products.
2. The total number of digital inputs and outputs is quite high compared to many other products.
3. It includes the necessary interfaces such as relays and opto-couplers for digital operations inside the enclosure.
4. Pentium based processor makes it possible to handle many complicated tasks in real-time.
5. Despite the high number of total inputs and outputs, it has a very compact structure, which provides ease of montage in electrical cabins.
6. It has also all features of a desktop PC.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

Where the number of digital or analog inputs and outputs of a controller is insufficient in an application, there may be two options to overcome this problem; connecting expansion modules to the controller to enhance the inputs and outputs, or using another controller communicating with the other controllers. The second option is most preferable in many aspects. Therefore, the next step of this project should be plugging an ethernet card into the controller so that all the controllers in an applications or in a factory may communicate with eachother and any desired computer. This feature provides increasing the inputs and outputs, a remote-control and also transferring the necessary information on the network such as some environmental measurements during a batch production, in order for user to determine the disturbing inputs so as to increase the total quality in that production.

The other requirement is a programmable gain amplifier for amplifying the low voltage signals. For example, a thermocouple needs to be amplified before the measurement. Therefore, in case of using a programmable gain amplifier, there is no need to use external amplifier circuits, which means more compact structure.

The last thing which has to be done in control systems is providing analog outputs for controlling some devices such as proportional valves. These works can be regarded as some possible future works for this project.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

SUMMARY

This thesis dealt with the definition of the fundamentals and the requirements of data acquisition and control applications in industrial environments and designing a new PC based embedded controller in respond to these requirements. The developed embedded controller will be able to be suitable for many applications, meeting the requirements discussed in this thesis such as real-time requirements. The developed controller is expected to be a good alternative to the existing controllers in many aspects.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve Dokümantasyon Daire Bşk.

ÖZET

Bu tezde endüstride veri toplama ve kontrol uygulamalarının temelleri anlatılmış, bu uygulamaların gereksinimleri tanımlanmış, ve bu gereksinimlere cevap olarak ta yeni bir kişisel tabanlı kontrol cihazı tasarlanmıştır. Geliştirilen bu kontrol cihazı endüstrideki bir çok uygulama için uygun olup, gerçek zamanda çalışabilmek gibi gereksinimleri karşılayabilecek yapıdadır. Bir çok açıdan, geliştirilen bu cihazın endüstride kullanılmakta olan benzer cihazlara iyi bir alternatif olacağı düşünülmektedir.

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
REKTÖRLÜĞÜ
Kütüphane ve ... DşK.

REFERENCES

[1] Auzas, Eric. Design Considerations For The Embedded PC. Embedded Systems Conference California, 1995.

[2] Laplante, Phillip A. Real-Time Systems Design and Analysis An Engineer's Handbook. New York: IEEE Press, 1993.

[3] IBM Corporation. Personal Computer Hardware Reference Library, Technical Reference, First Edition. March 1984.

[4] Intel Microprocessors: Volume II.

[5] Son, Sang H. Advances in Real-Time Systems. New Jersey: Prentice Hall, Inc., 1995.

[6] Tindell, Kenneth W. Fixed Priority Scheduling of Hard Real-Time Systems. Dept. of Computer Science, University of York, UK.

[7] Schuler, Charles A. & McNamee, William L. Modern Industrial Electronics. Macmillan/McGraw-Hill, 1993.

[8] Powerdwarf 486/R User's Manual. Microdesign Inc. October, 1997.

[9] Intel Peripheral Devices Handbook. Intel Corporation.

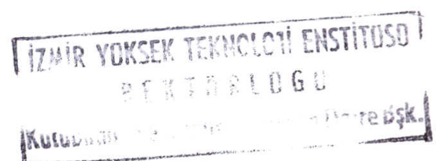
[10] ADS7803 12 bit Analog to Digital Converter Data Sheet. Burr-Brown Corporation, 1993.

[11] 40 W DC/DC Converter IMR-40 Family Data Sheet. Edition 3/11.97/IN 1.98 - © Melcher AG.

[12] Tanenbaum, Andrew S. Operating Systems Design and Implementation. Prentice Hall, 1987.

[13] Silberschatz, Abraham & Galvin, Peter B. Operating System Concepts. Addison-Wesley Publishing Comp., 1994

[14] See, Deborah & Thurlo, Clark. Managing Data in an Embedded System. Flash Software Development Group, Intel Corporation, 1995.


İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

İZMİR YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
Kütüph... Dşk.