This is to certify that the

thesis entitled

EVALUATION OF A REAL-TIME OPERATING SYSTEM IN SELF-CONTAINED DATA RECORDERS

presented by

Savinay Berry

has been accepted towards fulfillment of the requirements for

Master's degree in Electrical Eng

Major professor

Date Feb. 7, 2000

EVALUATION OF A REAL–TIME OPERATING SYSTEM IN SELF–CONTAINED DATA RECORDERS

By

Savinay Berry

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

1999

ABSTRACT

EVALUATION OF A REAL-TIME OPERATING SYSTEM IN SELF-CONTAINED DATA RECORDERS

By

Savinay Berry

Real-time operating systems, RTOSs, have traditionally centered around optimizing and simplifying multitasking operations in large embedded systems and personal computers. This has created an ever-faster trend in object-oriented programming enabling the proliferation of task management and error management structures. Unfortunately, there has not been a similar growth in the application of RTOS in small to medium size embedded systems.

The objective of this thesis is to partially fill this gap through the verification of a RTOS for a small embedded system. Various aspects of the RTOS were verified with the aim of developing a multitasking and conflict free platform for a self-contained data recorder. Results from the application of this methodology show that the multitasking feature of RTOS provides an object-oriented environment for applications. This environment consists of objects such as tasks, semaphores and other services provided by the kernel. Several contributions emerged from this research in addition to a multitasking paradigm that encourages the user to view an application as a collection of interacting objects rather than as a set of sequential operations or as a state machine.

To my parents

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Michael Shanblatt for his support, motivation and wise guidance during my years as a Masters student, especially during the critical times of the research. I would also like to thank my mentor and supervisor Rod Lambert for his constructive comments and valuable suggestions to overcome various problems during the course of this research.

I would also like to thank Greg Hoshal for giving me the opportunity to work in his company and be a part of the elite team of engineers. I am also grateful to my parents for their support and for helping me achieve all the objectives I have set in my life.

Thanks to my friend and colleague Mahesh for his help whenever I needed it.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Real-time computing refers to the constraint of timeliness on the relevant aspects of the execution environment, *e.g.*, event occurrences, resource needs and contention. The real-time environment requires these factors to be deterministic as well [1]. Systems operate in real-time to the degree that they achieve these objectives. This kind of a requirement is very rigid and thus such real-time systems are restricted to very simple yet important applications.

A real-time system is one in which the correctness of the computation depends not only on the correctness of the result but also on the time it takes to reach the result. If the timing constraints are not met, the system is said to have failed. The failure can be of two types: the output of the system is incorrect or the throughput of the system is decreased.

The first case corresponds to the constraints defined by the hard real-time system and the latter corresponds to the soft real-time system . A 'soft' real-time system is most popular since it does not produce a totally incorrect result. Soft real-time computing is conventionally not defined, except by default as not a hard real-time system. It violates either, or both, of hard real-time system's axioms. First, it is acceptable to miss some deadlines by some amount. These situations do not necessarily lead to the failure of a system. Second, some tasks may have action time constraints, which are multi-valued. These constraints specify the action's utility to the system in terms of best, better, worse, and worst completion times--instead of binary-valued deadlines. Clearly in this sense soft

real-time is the general and most realistic case, of which hard real-time is an extreme special case.

An object-oriented model turns out to be a better model for most embedded systems [5]. The other models like the state machine model do not deal well with the complexities of multiple, simultaneous events which are typical in embedded systems. Flow charts are good for describing sequential processes. State machines are good if there can be only one state at a time. Neither is good for describing systems of inter-dependant parts. Multitasking on the other hand is ideal for such systems. A real-time operating system has the capability to structure the code into various tasks. Changes are accomplished merely by adding, deleting, or changing tasks. Since the code is compartmentalized into tasks, propagation of changes through the code is minimal.

This chapter begins with a brief overview of the real-time operating system. The problem to be solved is then defined, followed by the research tasks. Finally the organization of the thesis is outlined.

1.1 Overview

An operating system is the low-level software that handles the interface to peripheral hardware, schedules tasks, allocates storage, and presents a default interface to the user when no application program is running. The operating system (OS) may be split into a kernel which is always present and various system programs which use facilities provided by the kernel to perform higher-level house-keeping tasks, often acting as servers in a client-server relationship. The operating system loader, BIOS, or other firmware required

at boot time or when installing the operating system, would generally not be considered part of the operating system.

A real-time operating system, RTOS, is a system where interrupts are handled within a certain specified maximum time, thereby making it suitable for control of hardware in embedded systems and other time-critical applications. RTOS is not a specific product but a class of operating systems. The facilities an operating system provides and its general design philosophy exert an extremely strong influence on programming style and on the technical cultures that grow up around the machines on which it runs.

A simple definition of a recorder would be a device that records data. It is also referred to as a data acquisition system (DAS). For our purposes it refers to a self-contained subsystem which includes the hardware and software involved in the sampling and analog-to-digital (A/D) conversion of analog signals, acquisition of data directly in digital form, digital signal processing prior to data recording, real-time monitoring, organization of data into structured blocks, and the recording of these blocks on disk and/or tape.

There is usually a central unit in the system which handles most of the data acquisition functions. This accepts analog and digital signals from the exterior and has interfaces to communicate with monitoring and storage devices, which are also an integral part of the DAS.

The data acquisition system is often the central component of complex engineering and scientific systems for data collection and analysis. Its basic tasks are the collection, distribution, monitoring, pre-processing, and recording of data. The system

must be capable of handling data, which is received directly in digital form, as well as analog signals, which must first be converted into digital form. Depending on the type of application, multiple sampling rates may have to be used, so non-critical signals may be sampled at lower rates to avoid unnecessary overhead. Also of great importance is the ease with which the configuration of the system may be modified to handle different applications.

The basic characteristics described above for a digital DAS must be achieved in a real-time environment, *i.e.*, the correctness of the computations involved depends not only on the logical results, but on the system meeting certain timing constraints to produce these results [1].

Recorder A operates on the principles defined above, but is built on a state machine structure. In this design, different recording tasks are performed independently and at a preset time. There is no inter-task communication and no task priorities.

1.2 Problem Statement

Recorders designed on a state-machine structure are very rigid and addition of new features in these recorders is not always an easy task. This leads to a very inflexible operation of the recorder. The general problem that this research addresses is the flexibility problem in recorders with a state-machine design structure [5]. Historically recorder designs have more closely resembled state machines than RTOSs. The literature reviewed in the later chapters discusses the importance of a RTOS design of the recorder.

The motivation behind this research is to develop a RTOS design estimation, which targets the general recorder structure. A recorder based on a RTOS will be capable of more advanced features due to the RTOS's ability to handle concurrent events that advanced features are likely to create. In the thesis, two separate recorders will be discussed, one with the state machine design structure (Recorder A) and the other with the RTOS design structure (Recorder B). This will enable a comparative study of the functioning of various aspects of the recorder.

1.3 Research Tasks

The research tasks identified for this work are as follows:

1. **Task 1-** Verify that the existing RTOS has the capability of performing the task management required by the data acquisition system.

Objective: To verify and check the proper functionality of the RTOS of Recorder B.

Significance: At present, Recorder B has a limited RTOS and the majority of its responsibilities deal with acquiring event acceleration data. A careful study to verify that data is recorded correctly in all situations is required. This includes the effects of sample jitter, resource conflicts, and deadlock prevention.

Approach: The verification will be performed with the help of an In-circuit Emulator (ICE) [4]. The communication and active stages of the recorder will be monitored closely and tracked down for any abnormalities with the help of assembly level code for the PIC 16C77 microcontroller used in the data acquisition system.

2. **Task 2-** Add new recording features to the existing data acquisition system.

Objective: To demonstrate the flexibility of the RTOS based recorder over the rigid state machine version.

Significance: New recording features will be added to the existing event based acceleration recording. These features will be chosen based on the difficulty associated with implementing them in the state machine recorder. Some comparisons will be made between the tasks involved in implementing the new recording features in each recorder.

Approach: The implementation of the new mode will be performed by processing the recorder data and writing code to add new features to the recorder.

3. **Task 3-** This is similar to Task 1 except this task will also be analyzing the interaction between the event based recording features added to Recorder B in Task 2.

Objective: To analyze the performance of Recorder B (RTOS based recorder) with respect to Recorder A (state machine based recorder) and to determine the effectiveness of the approach adopted in Recorder B.

Significance: This task will demonstrate the flexibility of a RTOS based structure for a self-contained data recorder.

Approach: Comparison between the results for Recorder B and previous results for Recorder A will be explained. An exhaustive comparison of the two recorders based on various points like flexibility, ease of troubleshooting, and ease of use will be done.

1.4 Organization of the Thesis

This thesis has been organized into eight chapters. Chapter 2 discusses the relationships among various software and hardware techniques. Also, in chapter 2, tools used for the completion of the research are presented and explained. It begins with a brief introduction to a self–contained data recorder. The concept of a RTOS is introduced. The architecture of the system is described with the emphasis on the microprocessor, the serial communication bus and software used to analyze data. A brief description of the emulator system used to analyze the device-under-test (DUT) is also given. The chapter concludes with a discussion of few scheduler tasks in the RTOS.

Chapter 3 explains the main experiments of this research. A thorough review of all the issues with the verification process of the recorder operating system are described. The material covered in this chapter summarizes the test points in a verification process. It also introduces the concept of the memory shutdown feature that is an important addition to the recorder system.

Chapter 4 discusses the problems encountered while changing the recorder from a prototype design to a PCB. The content of this chapter is essential for the research since this involves tracking the data flow from the analog section to the digital section and right down to the communication task of the recorder. Without the completion of this task, it would not be possible to test for the various tasks working on a RTOS. The analysis of the PCB is described. Various malfunctioning sections of the recorder like the reference voltage section are identified and their problems are solved.

New modes of recording that are useful in analyzing the flexibility claim that was made in the earlier chapters is presented in Chapters 5 and 6. Slow track recording and delay start/stop recording features are added to the existing recorder, which highlight the significance of the RTOS over the state machine structure. The results are tabulated in graphical form for ease of comparison.

An explicit comparison of the two recorder structures namely RTOS and state machine are detailed in chapter 7. An exhaustive comparison is made and the observations are used to provide the conclusion of the research.

Finally, chapter 8 lists major contributions of the research and identifies future directions of this work.

CHAPTER 2

PROBLEM BACKGROUND

2.1 Introduction

An operating system is the low-level software that handles the interface to peripheral hardware, schedules tasks, allocates storage, and presents a default interface to the user when no application program is running. The operating system (OS) may be split into a kernel which is always present and various system programs which use facilities provided by the kernel to perform higher-level house-keeping tasks, often acting as servers in a client-server relationship. The operating system loader, BIOS, or other firmware required at boot time or when installing the operating system would generally not be considered part of the operating system [7].

This chapter discusses the real-time operating systems (RTOS). A review of the elements of a RTOS is described. Next, various components in the experiment setup are provided. Thereafter, the scheduler tasks are defined and their importance to the recording process is highlighted.

2.2 Elements of a Real - Time Operating System

A real-time operating system is a system where interrupts are guaranteed to be handled within a certain specified maximum time, thereby making it suitable for control of hardware in embedded systems and other time-critical applications [1]. RTOS is not a specific product, but a class of operating systems. The facilities an operating system provides and its general design philosophy exert an extremely strong influence on programming style and on the technical cultures that grow up around the machines on which it runs. Real-time operating systems started to become popular a few years ago. Apart from the significant advantages of a RTOS, there are certain drawbacks of this system as well. Since a RTOS requires memory banks to facilitate concurrent processing, the memory requirement in this system is high. Secondly, the number of instruction cycles increases, thus, increasing the demand for power. Thus, RTOS is best considered in cases where power, memory and cost do not play a significant role in defining the structure of the system. Below, a description of the basic components and aspects of a real-time OS is presented.

1. **Kernel** – The interface between the operating system and user is known as the kernel. A kernel can be either a pre-emptive kernel or a non-preemptive kernel. The non-preemptive kernel is also called as a cooperative kernel because the tasks only give-up control when they want/need in coordination with other tasks, and events. In a preemptive kernel, a running task can be swapped out for a higher priority task when it becomes ready. The preemptive kernel relies more on interrupts as its driving force.

Recorder B can be said to have a semi-preemptive kernel [2] structure without any context switching. This structure combines the features of a non-preemptive and fully preemptive kernel structures. In this type of a structure, a running task can be swapped out for a higher priority task when it becomes ready. The priority of the tasks are decided by the scheduler option in the operating system. This type of a kernel relies on interrupts as its driving force hence the interrupt service routine (ISR) forms an important aspect of this structure. Recorder A, on the other hand, has a non-preemptive kernel structure.

2. **Semaphores** - A semaphore is a flag that is used to monitor and manage various tasks that are trying to gain control of a specific routine. A binary semaphore used in Recorder B allows only one owner to a particular resource and all the other tasks waiting to access it are made to wait [2]. The semaphore used in this case is in the form of a flag to keep track of the resource utilization. The semaphore is also used to avoid situations called 'Deadlocks' and to synchronize tasks.

3. **Interrupts** – Interrupts are very helpful in resolving real-time problems. An active task can be stopped, stored and another task can be run with the help of interrupts. The only thing to keep in mind is 'training' an interrupt request. This means that the interrupt should be serviced in such a way that it affects only the portion of the code targeted and all the important registers are saved so that they can be restored after the interrupt has been serviced and the kernel can start where it left off. The ISR and the initialization of the interrupts take place in the kernel.

4. **Mailboxes (buffers)** - Buffers are a nice feature in real–time computing if there is enough RAM space. They are used to pass messages between tasks, allow messages to be looked at when the task is ready, and to reply telling the sender that the message was

resolved. The buffer in the Recorder B helps to relay the tasks from the main memory in the chip to the EEPROMs. This procedure helps to clear up the main memory very fast and make way for new data to come in. It also helps to eliminate the EEPROM write delay of 40ms caused in the absence of a buffer in Recorder A. Basically, it is helpful in streamlining the flow of data from the main memory to the EEPROMs.

5. **Context switching** - This part of the real–time operating system helps to save all important registers of an outgoing task and introduce new registers for the incoming task. In a preemptive kernel when a higher priority task needs to be executed, the lower priority task is interrupted and the address is saved. There is no context switching in non-preemptive kernels [2]. The kernel structure in Recorder B does not need context switching and the tasks in this recorder are designed to avoid this.

6. **Scheduler** – At the core of the real-time multitasking kernel is the scheduler. It is the most important part of the kernel structure in Recorder B. It is that part of the kernel that decides which task will run next. The various tasks to be performed are arranged according to their priorities in the scheduler and are carried out in the same way. There is a certain time set for the event to be performed. If the procedure carries through all the events before the slotted time, the scheduler puts the system into 'sleep' mode. Since the scheduler is practically the heart of the kernel, it is very difficult to differentiate between the scheduler and the kernel.

7. **Deadlock** - Deadlock is a situation in which a semaphore flag is set for a particular task using a resource and a higher priority task wants to access that same resource. At this time a deadlock situation occurs since the higher priority task has to be performed first and it cannot be performed without the resource. This resource is being utilized by the

lower priority task and the lower priority task cannot let go of the resource unless it finishes the task.

8. **Mutual Exclusion or mutexes** - This is a kind of semaphore which serves the purpose of excluding other tasks from gaining access to critical sections of code. Recorder B does not use this technique for managing resources.

9. **Events** - Various tasks can be carried out in Recorder B. These tasks can be processed concurrently. Tasks such as recording, communicating, processing *etc.* can be categorized as events.

Other aspects that form an important part of the RTOS structure, but not directly related to the RTOS structure used here are the critical section of the code, time management, routing techniques, interrupts management, instrumentation, task management, and queues.

It is important to realize the potential of a RTOS as a base for data acquisition systems. Various aspects of the above mentioned properties of a RTOS will be used to explain the structure of a self-contained data recorder based on a RTOS design in the following chapters.

2.3 Recorder Overview

The data acquisition system (DAS) is often the central component of complex engineering and scientific systems for data collection and analysis. Its basic tasks are the collection, distribution, monitoring, pre-processing, and recording of data. The system

must be capable of handling data which is received directly in digital form, as well as analog signals which must first be converted into digital form. Figure 2.1 shows a general block diagram for a data acquisition system.

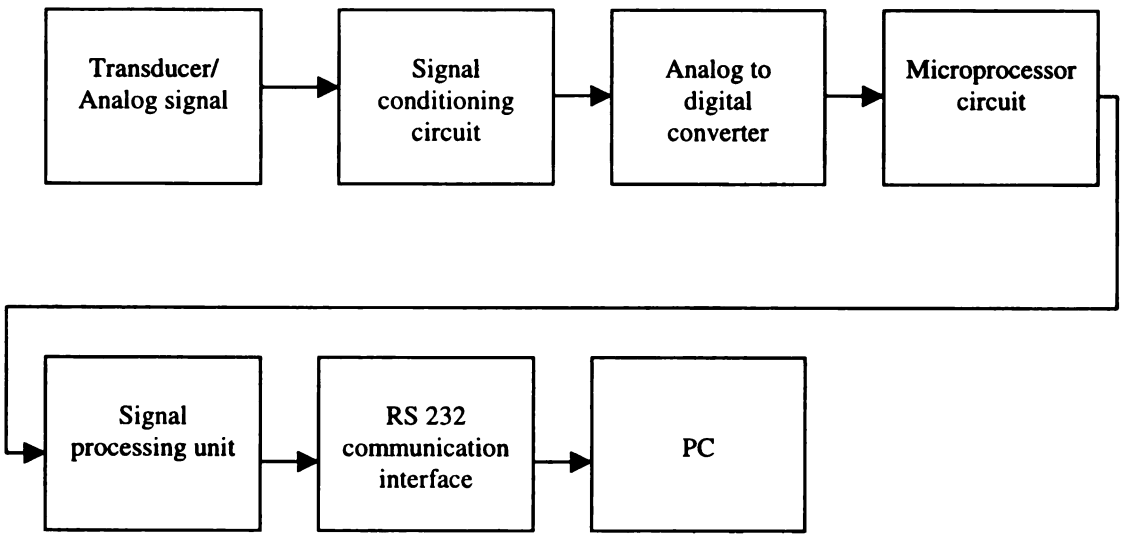

Figure 2.1. Block diagram representation of a data acquisition system.

Recorders used in this study represent a specific data acquisition system, which acquire and process shock and vibration data. The recorders employ two modes of recording: event based recording and slow track recording. The sensor used to record events is an accelerometer. **Recorder A** is an *event-based recorder*, which records data instantaneously when it goes above a user defined threshold level . This data is then grouped together to form an intelligible pattern for analyzing and studying in the form of graphs or other patterns. This recorder is designed for documenting dynamic

environments such as moving vehicles, trains, planes *etc.* It has a sensing element for sensing various events, a microprocessor controlled data acquisition system, user-selectable filtering, signal conditioning and solid state data memory (EEPROM).

This recorder can operate in two modes: *active mode* and *communication (COM) mode*. The *COM* mode is responsible for getting the data from the recorder to the PC and the *active mode* measures and reads data that is logged by the recorder. It uses serial data communication for communicating with the PC.

Recorder B has the capacity to perform all the above mentioned modes, but is designed on a RTOS structure. This structure enables Recorder B to define tasks and assign priorities to them. Distinct sections of code represent each task. Thus, troubleshooting for errors and adding new tasks to the recorder can be accomplished with greater ease than in Recorder A. This statement will be justified in the following chapters.

2.4 Limitation of Recorder A

Conventional recorder designs have more closely resembled state machines rather than RTOSs. Recorder A is based on this kind of state machine design. Adding a new feature to a recorder based on this structure is a very complex process and often leads to undesirable results. Current thinking is that recorder design could benefit from a limited form of RTOS. It is expected that the RTOS can reduce future development time by allowing tasks to be re-used in future recorders. It can also simplify the task of implementing features in later revisions. Recorder B is based on the RTOS structure and

is thus, capable of performing all the above mentioned functions. It is capable of more advanced features due to the RTOS's ability to handle the concurrent tasks that the advanced features are likely to create.

Events are defined as exceeding of change in velocity (ΔV), peak, duration and acceleration threshold above a user-defined threshold. By configuring the recording control parameters (RCPs) the user can set the recorder up so that available memory is utilized efficiently. Figure 2.2 shows a typical acceleration event. It shows an event, which is caused due to the exceeding of acceleration above the acceleration threshold. The event peak, duration and change in velocity (as the area under the curve) are also illustrated.

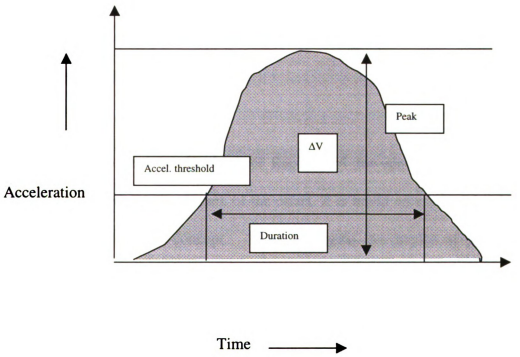

Figure 2.2. Acceleration *vs.* time: Illustrating acceleration threshold, peak acceleration, event duration, and change in velocity.

Recorder A is configured to operate in an event-triggered manner. That is, the recorder stores acceleration-event data only when one or more of the three accelerometers exceeds the trigger level and trigger duration threshold. To be recorded, an acceleration event must exceed the trigger level for at least the trigger duration threshold.

Another important aspect of event based recording is the correction of the offset. Offset is the deviation of the reference level for recording events from the desired reference level after the recorder goes through an event. Before the next sample of data is taken, this offset has to be removed to bring the reference level back to the desired value. If this is not done, the data logged by the recorder would be incorrect as it would be riding on an offset level. The amplitude of the readings would be incorrect.

2.5 Modifications

In order to overcome the drawback of Recorder A for concurrent recording, the new recorder (Recorder B) keeps track of the offset. It is based on a scheduler design that is derived from the RTOS concept. There are again two modes of operation for this recorder: *active mode* and *non-active* mode. During active recording, acceleration signals are digitized to the desired bit resolution and stored in onboard digital memory. The addition of the RTOS as a base for the recorder provides for flexibility in design and easy troubleshooting for problems since the tasks are now written in separate and distinct sections of code. The block level design of the two recorders is the same. Both contain a microcontroller-based data acquisition system. The microprocessor contains its own

operating firmware, which is programmed at the factory prior to delivery. Once programmed, the microprocessors operating firmware is permanent and cannot be changed. The microprocessor is interfaced with a 12-bit analog-to-digital converter (ADC). During recording, the microcontroller directs the ADC to sequentially sample each of the three selected accelerometer channels. The ADC converts analog acceleration and temperature channel voltages into discrete numbers for storage in data memory.

There are certain corrective measures that are adopted in the construction of Recorder B. One of the most important is the correction of the oscillator frequency that may tend to drift with the variation in temperature since the oscillator consists of a R-C network. Digital potentiometers keep track of this variation and negate any change in resistance that may take place due to thermal drift. A digital potentiometer is a collection of transistors with a wiper attached to it. The variation in resistance is controlled by software and increasing resistance means increasing the number of transistors in series and that is done by the wiper adjustment (controlled through software). The interface between the software and the digital potentiometers is an Inter-Integrated Circuit (I^2C) bus. The I^2C bus is a bus protocol developed by Phillips to facilitate communications between microcontrollers and peripheral devices.

2.6 Description of the PIC Microcontroller

The code for the various tasks to be performed on the measurements is written in assembly language for the PIC 16C7X microcontroller. The PIC 16C7X is a low cost, high performance, CMOS, 8-bit microcontroller with inbuilt A/D converters [3]. It

employs a RISC architecture and also follows the Harvard type architecture of the separate instruction and data buses. This allows a 14-bit wide instruction word and a 8-bit wide data word. The instruction set contains 35 instructions. The PIC micro used here has about 400 bytes of RAM. It also contains a USART for serial communication, which is carried out by a two wire Inter-Integrated Circuit (I^2C) bus. PIC contains an 8-bit ALU and working register. In two-operand instructions in the PIC, generally one is a file register (F) and the other is the working register (W). In single operand instructions, the operand is either the W register or a file register. The PIC 16C7X has a 13-bit program counter capable of addressing an 8Kx14 program memory space. The data memory is partitioned into two banks which contain general purpose registers and special function registers. The special function registers are used by the CPU and peripheral modules for controlling the desired operation of the device.

There are three timer modules in the PIC. They can be classified as Timer0, Timer1, Timer2. Timer0 is an 8-bit timer/counter, Timer1 is a 16-bit timer/counter and Timer3 is an 8-bit timer with pre and post scaling facilities. The various uses of the timers in context with the Recorder functioning can be listed as follows:

Timer0: a) Used to time a particular resource (introduces a dead time).

b) Separates out the packets of data.

c) Used to adjust main processor clock (setting the baud rate).

Timer1: a) Fires when it is time to take samples from the A/D, runs off a 32KHz crystal.

b) Keeps track of the Real-time Clock (RTC) and is an accurate timekeeper.

2.7 Serial Bus Interface

The Inter-Integrated Circuit (I^2C) bus employs a comprehensive protocol to ensure reliable transmission and reception of data [3]. While transmitting data, one device is the 'Master' (generates the clock), while the other device(s) acts as the 'Slave'. The output stages of the clock (SCL) and data (SDA) are responsible for establishing a communication link between the recorder and the PC. The START and STOP conditions determine start and stop of data transmission. The START condition is defined as a high to low transition of the SDA when the SCL is high. The STOP condition is defined as a low to high transition of the SDA when the SCL is high. Due to the definition of the START and STOP conditions, when data is being transmitted, the SDA line can only change state when the SCL line is low.

2.8 Scheduler Tasks

Three main tasks of the recording process are biasing, triggering and communicating. Assume that these tasks can be categorized as "super-tasks". It is interesting to note that there are some tasks performed by Recorder B that are within other 'super-tasks'. These tasks as may be called 'sub-tasks'. The scheduler contains a prioritized list of the super-tasks . One of the objectives of this study is to see if some or all of these tasks can be performed concurrently. The list of 'super-tasks' in the scheduler is as follows:

1. Accumulation of data - This accumulates the event data in all the three directions (x,y,z).
2. Triggering - This is the main trigger task for event recording and it keeps a check on the events that cross the threshold value set by the user.
3. Event over task - Signals the end of an event accumulation period. This will happen when the user-defined threshold of event duration is exceeded.
4. Start event task - Responsible for shifting in the next set of acceleration samples and clearing the previous task flag.
5. Outgoing character task - Sends a character out of the COM port of the microprocessor into the buffer.
6. Time-out task - Determines the time-out of the COM port, so that each task clears its flag and is not rescheduled.
7. Packet dead time task - Sets the timer0 to proper value so that when it rolls over and sets the interrupt flag, enough dead time has passed.
8. Writing to the EEPROM task - In this task, the event data is written onto the EEPROMs via the buffer from main memory. The writing of data is carried out with the serial interface bus (I^2C) using the SDA and the SCL lines for transmission and reception of data packets. Since there is more than one EEPROM on the circuit board, first a single chip is addressed. Data is read/written onto that chip and then the next chip is addressed. The I^2C bus standard used here actually introduces a dead time between events so that the next packet of data does not get interpreted as the next address. The addressing of the EEPROM is done by a control byte and subsequently, the data byte is sent from the buffer.

9. Move event task - Moving the accumulated event data from the main memory to the buffer. If the buffer is in use, the event can't be moved.
10. Initialize biasing task - Here the initialization procedure for the offset correction is carried out. Biasing task is a very important task, since it gets rid of the offset and eliminates any errors that might occur due to an incorrect reference level. There are two types of biasing: *fast biasing* and *slow biasing*. In Recorder B, the offset correction is carried out by keeping a history of offsets, *i.e.,* offset is accumulated, waveform tracked down and then the offset is implemented (corrected) at the end of the event. The fast bias is eight times faster than the slow bias. Due to this, as the fast bias is tracking the offset, the slow bias is recording the values of all these offsets. This is helpful in situations when the recorder comes out of an event abruptly. For example, if the event duration is exceeded and the recorder comes out of an event, the slow bias will contain the most recent offset value. This will be helpful to start the biasing process again.
11. Clear event task - Clears all the event data from the main memory so that a set of event data can be recorded.
12. Frequency synchronization task - Is responsible for comparing the timer1 to the instruction clock through timer0. This task is given a low priority since there is already a clock for synchronization present in the system, made from a resistive network and is very minimally affected by thermal drift.
13. EEPROM reading task - Is responsible for reading event data from the EEPROM. This transfer of data also takes place in the same way as the writing process.

14. Switch task - This is to test the switch state for a particular task. After completion of the task, the switch should reflect the state. If it does not, then the task toggles the mode of the switch.
15. Overrun check task - This task checks if there was any overflow in the FIFO queue of events through the COM port. If yes, this task clears the FIFO and enables the reception of event data.

After completion of each task, the system goes back again to the scheduler so that the task with the next priority can be executed.

2.9 Experiment Setup

Most of the verification and analysis of Recorder B was done with the help of a prototype of the recorder, an emulator setup, a PC fitted with a common interface card and a PC bus. The emulator provided for easy and flexible troubleshooting for various sections of the assembly code of the recorder.

2.9.1 The Emulator System

An emulator system is one which performs all the functions of a device-under-test (DUT) without the need for any specific hardware. The four major components of the emulator system are an emulator pod, PC interface card, probe and an integrated development environment (MPLAB). The emulator contains all emulation and control logic common to a family of microcontroller devices. The emulator contains emulation

memory, trace memory, event cycle timers, and trace/breakpoint logic [4]. The emulator controls and interfaces to an interchangeable probe via a 14 inch ribbon cable. The PC interface card connects the emulator system to a compatible PC host. A 37-conductor cable connects the PC interface card to the emulator pod. The emulator requires an interface connection to the target microcontroller device that is being emulated. A probe specific to the microcontroller family that is being emulated provides this connection. The installed probe configures the emulator system to a target microcontroller, and connects to the ribbon cable coming from the emulator. Figure 2.3 shows the block diagram representation of the experiment setup.

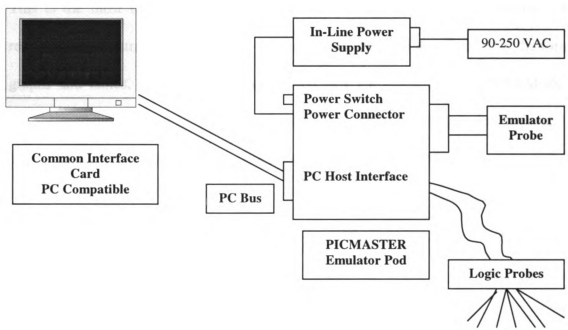

Figure 2.3. Block diagram of experiment setup.

2.9.2. MPLAB

The MPLAB software runs in a Windows environment and provides full display, modification and control of the target application under emulation. MPLAB software supports the PIC 16C77 microcontroller that is used in this research. The MPLAB software automatically identifies the probe type and configures the emulator system to the target processor.

2.9.3. DYNAMAX

This is the most important and relevant software from a users point of view. It is responsible for extracting information from the event data and displaying it in the form of graphs and tables. Various views and tables that are supported by DYNAMAX are histogram view, event table, time of event, amplitude of event and direction of event.

Figure 2.4 shows a typical event displayed in DYANAMAX.

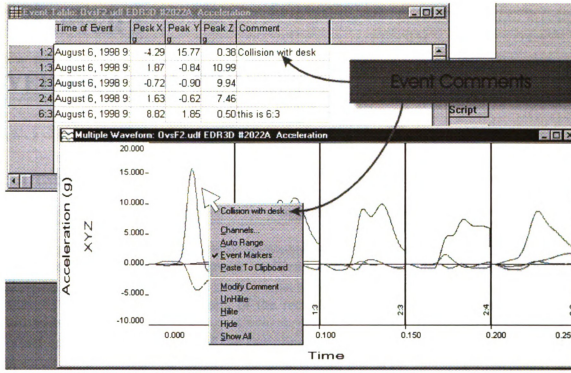

Figure 2.4. A typical event displayed in DYANAMAX .

2.9.4. MFRCOM

This software performs all communications between the PC and the recorder. This includes programming the recorder prior to a test and retrieving the data from the recorder after the test. All the options for trigger levels, event length, recording control parameters (RCPs), filter levels, auto start/stop enable and others, are set and modified in this interface software. From an analysis point of view, this is a very important component of the verification process since it provides the link between the firmware, software and the hardware. Most of the changes, like that in the recording control parameters, would not be possible without the help of this software.

CHAPTER 3

VERIFICATION PROCESS

3.1 Introduction

This chapter deals with the stepwise verification process by putting forth an exhaustive analysis of the all the key aspects of the recorder mechanism. Serial communications, offset correction, verification of the sampling frequency, and calibration of the recorder are some of the topics covered in this chapter. Apart from that a careful analysis has been done to distinctly identify the importance of each task and its relevance in the research. In the following sections, the addition of a new feature has been described and its significance in the recording and communicating processes has been highlighted. The verification process presented in this chapter is done for Recorder B.

3.2 Test Points and Verification

Verification of the proper functioning of the various tasks in the recorder was done with the help of an In-Circuit-Emulator (ICE) [4]. During the initial testing of system components, power-up testing was carried out. Then the system was verified to determine if all the signal behavior is within specification. A host of precise measurements were made in real time, to determine if the tasks were being synchronized

properly and no conflicts were occuring. Verification typically included measuring rise/fall times, edge-to-edge timing, jitter and jitter tolerances, signal path skew, and data path variations.

The emulator system helped to perform the above mentioned tasks. One system is said to emulate another when it performs in exactly the same way, though perhaps not at the same speed. The PIC ICE was a very flexible testing apparatus which worked as if it was actually a part of the circuit under test. All tests could be carried out using the ICE including detecting a high/low on any of the pins of the PIC chip. A typical example would be emulation of one computer by (a program running on) another. Emulation is used as a replacement for a system whereas simulation is used if the system just has to analyzed and the behavior of the system has to be predicted. Simulation is used to predict the behavior of a system. A simulator cannot account for errors which might take place when the original system is set up. An in-circuit emulator acts as a part of the system and operates as if it were the original system, thereby taking into account every error in the system.

MPLAB-ICE is Microchip's new Universal In-Circuit Emulator (ICE) for the PICmicro 8-bit microcontrollers (MCUs) [4]. Complex triggering of the emulator provides sophisticated trace analysis and precision breakpoints. The trace analyzer captures real-time execution addresses, opcodes, and read/writes of external data. It also traces all file register RAM usage showing internal addresses and data values, as well as all accesses to special function registers, including I/O, timers, and peripherals. Triggers and breakpoints can be set on single events, multiple events, and sequences of events.

The basic operation of the recorder can be divided into three broad categories, considering the scheduler structure that is being evaluated.

1.Biasing - The process of offset correction is used to reduce the offset to zero in a train of event samples and enable correct logging of acceleration samples with the correct reference points.

2.Triggering - The process of switching over from the communication mode to actively recording acceleration samples is called triggering.

3.Communicating - "Talking" to the recorder from the PC, loading recording control parameters in the recorder and uploading the acquired data from within the recorder to the PC .

Processing of the above mentioned tasks require several other sub-tasks to be performed in synchronism with each other. The flow of tasks starts with the initialization memory task, which clears all the registers in the memory and empties space to accommodate the acceleration samples. It is also responsible for setting up the linear addresses and initializing the memory pointer. The size of the recording control parameters blocks are also initialized and loaded into the memory. This is followed by loading the recording control parameters into the memory before the recorder goes active. The offsets for all the axes, gains and filter settings are loaded into the memory in this task. The trigger levels for all the axes, which are an important aspect of the biasing process are loaded and can be changed as and when the user desires. After the parameters have been loaded, the biasing process is started by the *InitBias* task. This task is only run when there is no event data in the event accumulation space, *i.e.*, the recorder cannot be above trigger or be

waiting to move an event. The *CInitBias* task continues the *InitBias* process and starts the fast bias process. The triggering is disabled for the period when the recorder is biasing. The two types of biasing: slow and fast bias work together to completely bias out the offset. Fast bias works eight times faster than the slow bias and is responsible for tracking the movement of the event when the recorder is above trigger level. Simultaneously the slow bias is keeping the history of offsets as the fast bias is proceeding. After the recorder comes out of an event, the control is transferred from the fast bias to the slow bias. Since the slow bias contains the exact location of the time the recorder came out of the event, the next acceleration sample would require less biasing time to zero out the offset. This methodology is useful to decrease biasing time and increase processing speed.

Based on the above mentioned process and after rigorous testing and tracing analysis on various sections of the code for the recorder, the following points were found to be the most important to detect bugs in distinct sections of the recorder mechanism.

1. *CINITBIAS* – Triggering check.

This point in the code was essential to check if the recorder was switching over from biasing to triggering or not. Figure 3.1 shows the code used to check the triggering status of the recorder. If the system hung up after biasing (correction of offset), then it was concluded that the there was a bug in the biasing process and thus, the portion of the code for the biasing process was singled out. Again the emulator was used in that section of

the code to determine the exact location of the error using trace analysis and conditional break points.

```
Bcf     INTCON,GIE;suspend all interrupts while changing
Movf    Tmr1Lbak,W; load the sample period
Clrf    EDX_lo ;clear the counter and timer
Clrf    EDX_hi
Bcf     PORTD,TrigLED; turn off trigger LED
Bsf     Task2,TrigEn; enable triggering since we are now biased
Goto    SCHED
```

Figure 3.1. Section of code for triggering check.

2. *TESTSWITCHTASK* – Check for active and COM mode toggling.

Figure 3.2 shows the section of code that was useful in identifying if the switch was actually toggling from the active to record mode in the recorder. A break point was placed at this section and the switch was physically flipped over. If the execution hit the break point, then the changeover was proper and then the trigger enable was checked. If the code did not break, the problem was traced back to the response of the switch with the changes in the firmware.

```
.
.
Lcall       LoadParam       ;call the load parameters procedure
Movlw       b'00001111'     ;mask the upper nibble of the StateInfo
register
Andwf       StateInfo
Bsf         StateInfo, 7    ;mark as an On-time
Lcall       WriteTime       ;run the write time task
;clear bias numbers
Clrf        Xoff
Clrf        Yoff
Clrf        Zoff
Lcall       RecordNow; enable recording if all parameters are set
.
.
```

Figure 3.2. Section of code for mode check.

3. *INCRLINTSK* - Memory overflow check.

Figure 3.3 show the point at which the control is transferred from the point when the difference between the top memory and the bottom memory pointer reduces to less than a specified number of bytes. The code given below checks if the memory overflow flag has been set. In response to that, it initiates the *StopAct* procedure, which switches off the analog power to the recorder and stops recording to avoid any further logging of data into the recorder. This maintains data integrity and makes the recording process very data safe.

```
Btfsc     Res0,MemFull   ; if the memory is already full, do not call StopAct
Return
Bsf       Res0,MemFull   ;memory overflow flag
Movlw     b'01000001'    ; mask all bits
Movwf     StateInfo      ; write the marker to reflect the state of the recorder
Lcall     StopAct        ; stop recording
Return
```

Figure 3.3. Section of code for memory overflow check.

4. *Z TRIGGER TASK* – Check for proper calculation of offsets in the z direction.

Figure 3.4 contains code, which checks for the occurrence of any instances when the recorder crosses the user-defined threshold level in the z direction. If the event crosses the set trigger level, the event is categorized as an acceleration sample and is stored in the memory. The offset is stored and accumulated along with the other stored offsets. Figure 3.5 shows the flow of execution for the above mentioned processes until the biasing stage. The process execution after the biasing stage is shown in Figure 3.6. It shows the triggering process of the recorder.

```
.
.
Movlw     b'00000111'    ;mask all the nigh bits
Movf      ZUL,W          ;move low byte of trigger into W
Subwf     ZCL,W          ;subtract trigger from sample
Movf      ZUL,W          ;move high byte of trigger into W
Btfss     STATUS,C       ;did the subtract leave a carry?
Incf      ZUH,W          ;add one to the trigger
Subwf     ZCH,W          ;subtract the trigger from the
sample
Btfss     STATUS,C       ;is there still a carry?
Goto      NoZtrig        ;no trigger crossing on Z-axis
Call      FrezOff        ;trigger crossed..store Z offset
.
.
```

Figure 3.4. Section of code for offset check.

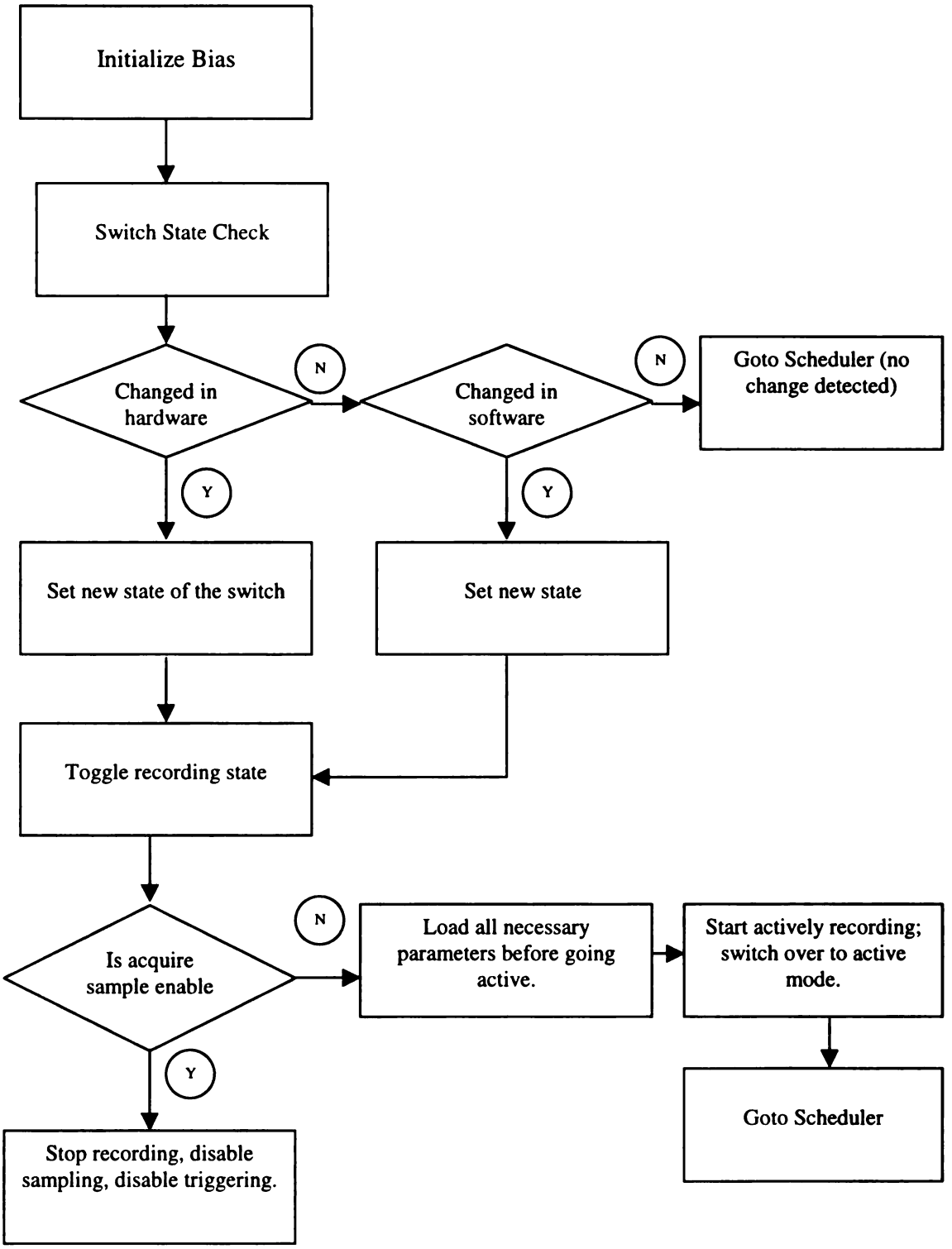

Figure 3.5. Biasing process.

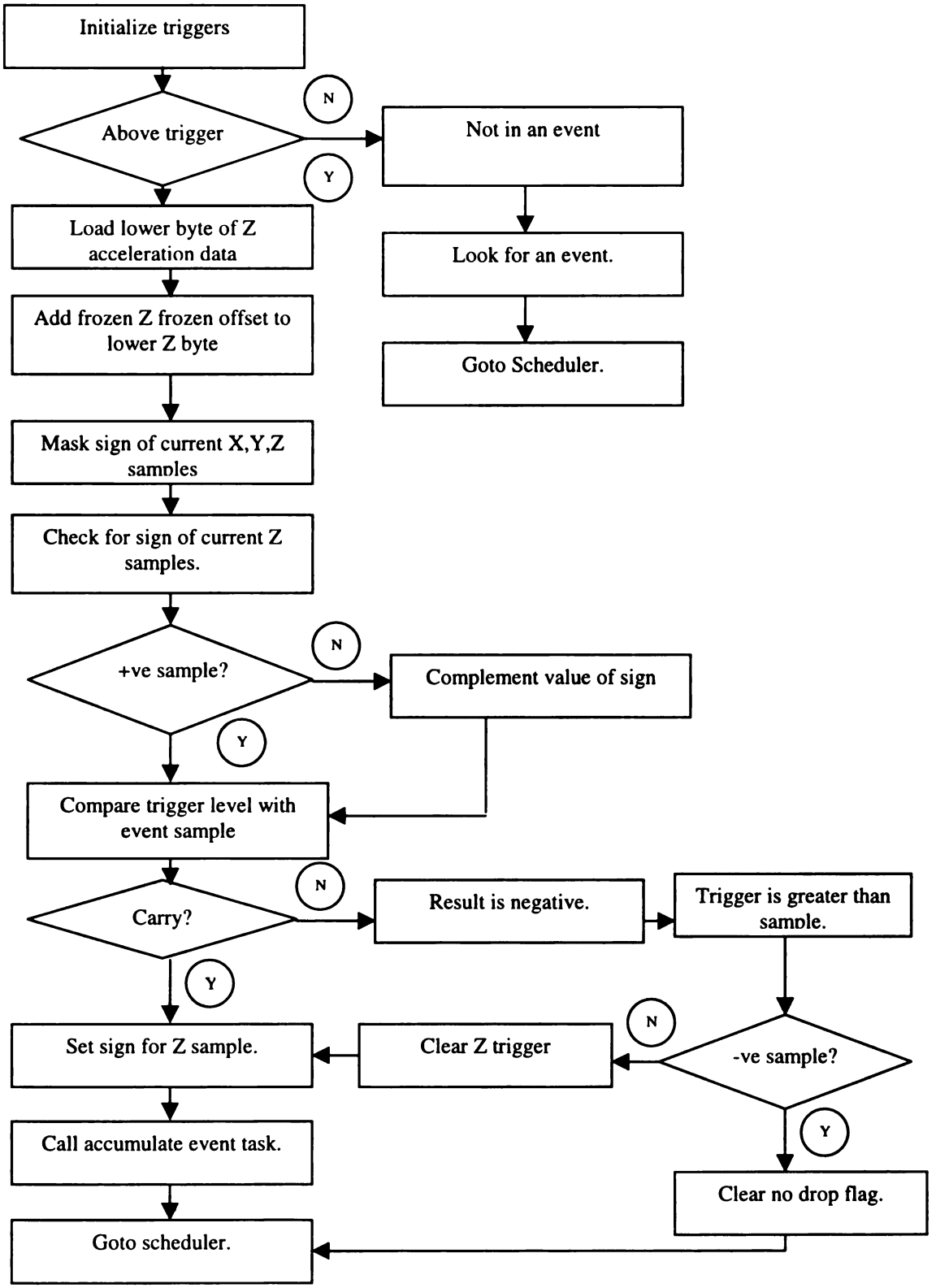

Figure 3.6. Triggering process.

3.3 RS-232 Communications

As the popularity of asynchronous serial communication became widely accepted by the industry, the RS-232 standard gained very wide acceptance. The use of this standard is visible in almost all industrial, portable, desktop, data acquisition and test measurement applications that use a serial port for communication. Even though the standard specifies a maximum data rate for RS-232 of 20 Kbps, some applications need for higher speed is overwhelming [2]. The recorder uses serial communication to interact with the PC. Data is sent in a packet protocol, which defined the number of bytes to be transferred in each packet and the associated command with that packet of data. The physical connection consisted of an RS-232 male-female cable connected from the output of the MAX3220 transceiver to the serial port of the PC. The transceiver had to be setup in the auto-shutdown mode. This required configuring the MAX3220 in a way different from what it was setup earlier for the prototype board. During the switch from the prototype to the actual PCB, this change in transceiver setup was overlooked, so it caused a lot of problems in talking to the recorder. The FORCEOFF pin in the transceiver IC needed to be connected to the positive supply to enable the auto-shutdown mode.

Testing was carried out for active and COM modes after the communication was restored with the recorder. It was found that initially the recorder was only loading the parameters in the recorder and not retrieving the logged data from the recorder. This was caused due the reason mentioned below.

Conflict between the tasks that loaded the RCPs into the recorder and the *testswitch* task that initiated the active mode can lead to the incorrect operation of the COM mode. Due to the conflict, the values of the RCPs may be corrupted and thus, disable the entire recording process. As a consequence, the recorder can log no data. Hence, the COM operation will not fetch data from the recorder, since there is no data in the recorder.

3.3.1 Packet Protocol

Data was transferred from the recorder to the PC in the form of packets. These packets were a good way to check for data integrity and detect corrupt data, if any. The protocol used for the transfer of these packets was called as a packet protocol. It used serial RS-232 communication for data transfer. The packets consisted of serial numbers to identify the packet being sent, command identifiers, data and a checksum to confirm correct transfer of data. Markers were assigned to each event such that whenever an event was loaded into the event accumulation space, its presence was indicated by a corresponding marker. The first marker was the on marker and the last was an off-marker. Thus, if no events were recorded, the event accumulation space would contain only two markers.

3.4 Offset Correction

Offset is the deviation of the reference level for recording events from the desired reference level after the recorder goes through an event. It is caused due to electrical noise and change in the orientation of the recorder. Offset correction is one of the most

important processes in the recording process. A new approach was used to zero out the offset created due to successive acceleration samples and noise components in the readings. The offset correction was divided into software and hardware offset correction. The hardware offset correction was enabled by adjusting the gain of the instrumentation amplifier which controlled the analog signal to the recorder. This was done with the help of a digital potentiometer, which served as the gain resistor of the instrumentation amplifier. Software offset correction is a more conventional method of offset correction. It tracks and identifies the position of the event after one acceleration sample has been taken and compared with the reference position. This process then adjusts the reference according to the position of the acceleration sample. The trends for a presence of offsets in a train of samples were observed. A positive trend indicated that the offset had been consistently above the reference level and negative trend indicated just the opposite. The hardware offset correction was synchronized with the reference voltage of the instrumentation amplifier. Half the full-scale voltage was applied to the instrumentation amplifier, *i.e.,* half of 3.3 V was set to be equal to 800 counts of the A/D. After an event had occurred, the frozen offset mechanism of the recorder could track the event waveform and determine the magnitude and the sign of the offset. This helped to bring back the reference for the event back to the 1.65 V level. This adjustment was done by the hardware offset for all the three axes. The hardware offset was calculated at the output of the instrumentation amplifier. At this point, it was set to 1.65 V, *i.e.,* half of the total range of 3.3 V input voltage to the analog section. This could be controlled and changed with the digital potentiometer. The nominal value in the digital potentiometer was 128 counts.

The range of offset was limited by 128 counts either way of the reference. Thus, if a value greater than 128 counts was noted, the software offset would take over. If the hardware offset value was greater than the reference voltage, then the count in the digital potentiometer would increase and vice-versa. At this point all the other operations had to be suspended since without the offset, neither the biasing process nor any other process can start. After this point, the software offset took over and stabilized the waveform to reflect the nominal value of 800_{16}.

Event length was a useful parameter. It was used to cut short an event in which the orientation of the recorder had changed permanently and displayed a constant offset from the reference point. Under such conditions, the recorder would come out of the event after the event length was exceeded and immediately start with the offset correction process. At this point, the slow bias would take over since it had been continuously storing the offset when the recorder was in the event.

Certain problems were encountered while going through the offset correction process of the recorder. The offset correction process was divided into the offset correction for the X, Y and Z axes. The offset correction process for the Y and Z axis were functioning properly but the X axis offset correction was behaving inconsistently. The X channel offset was not stabilizing at the mid-scale level of 800_{16} as it should have been to determine the reference level for the recorder and record events thereon.

The problem was traced to inconsistencies in the analog section of the recorder. The analog section consisted of a sense/program resistor which was responsible for measuring the acceleration change in the circuit. Since all the instrumentation amplifiers had very high input impedance, the program resistor (which was connected to all the

accelerometers) was adjusted to reflect the change in acceleration. It turned out that the common mode rejection, CMR, of the X channel instrumentation amplifier was higher than the maximum limit given in the datasheet for a 3.3 V supply. CMR is the unwanted part of the voltage between each input connection point and ground that is added to the voltage of each original signal. The CMR was observed to be 2.58 V, whereas the upper limit for the CMR was 2.1 V. As soon as the CMR would exceed the 2.1 V limit, the instrumentation amplifier would get saturated and try to drive the voltage down and then kick off the circuit. This resulted in the fluctuation of voltage levels as observed in the previous case. Thus, the value of the program resistor had to be changed to reduce the CMR. Before the modification, the voltage across the sense resistor was 0.45 V and after modification it was reduced to 0.2 V.

3.5 Verification of Sampling Frequency

The sampling frequency of the recorder determines the rate at which events are sampled in real time. More samples are recorded each second at faster frequencies. This helps in more accurately characterizing the nature of the event. The sampling frequency for Recorder B was adjusted to 1024Hz. Thus, the A/D clock would run and sample events at a frequency of 1024Hz. It was to be verified that the sampling frequency at the A/D clock input was 1024Hz and not any other value. This was done by observing the transition levels at the A/D clock input on a logic analyzer. By measuring the time period between two consecutive samples, the frequency was calculated over a wide range of

readings. The variation in the frequency, if any, would cause jitter in the recorder readings.

Another method for verification of the sample rate involved directly reading the hex values stored in the registers. The sample rate of the recorder was constant during the biasing process. This was done since variable sample rates during biasing induced inconsistencies in the time for biasing. This was undesirable, since the biasing process had to be uniform for all conditions. The user could vary the sample rate when the recorder was in the active mode, but could not adjust the sample rate during biasing. The sample rate in the active mode could be set in the software interface for the recorder (MFRCOM). The hex values corresponding to a certain sample rate were observed. For example, assume that a hex value of FF80 was observed. Thus, the sample rate of the recorder would be

$$FFFF/(FFFF-FF80) = 204_{16}Hz = 516_{10}Hz.$$

This was done since at FFFF, the recorder is sampling at 64*1024Hz and in the real time clock, RTC, every rollover of the counter indicates a 1 second interval. Thus, once an interrupt is generated at FFFF rollover, the control goes back to FF80, takes its complement and adds it to the value in the RTC to indicate a one second interval.

3.6 Verification of Filter Values

Low-pass filtering is a process that eliminates unwanted frequencies in a given frequency spectrum. The low pass filter in the recorder is designed to filter out accelerations above a given cut-off frequency. An example of unwanted frequencies might be constant vibration from a motor or other piece of machinery. The filter values were adjusted in the MFRCOM software interface for the recorder. This task involved checking to see if the filter value entered by the user was commensurate with the filter value in the code. Also it was checked and verified that a change in the filter value was recognized by the operating system of the recorder.

3.7 Calibration of the Recorder

Calibration is the process of verifying an instrument's performance against a known standard to determine whether it is still within specifications and hence still suitable for the task. What happens when an instrument is found to be out of specifications is most important - especially to companies operating within a Quality System (such as ISO9000) [6]. Instruments are chosen for an application based on their ability to make measurements with an appropriate level of accuracy. The accuracy of an instrument is defined in its specifications.

The calibration of Recorder B was done using a 1G gravity rollover test. In this test the recorder was turned over its side by an angle of 90 degrees. The operational amplifier gain values were adjusted to produce an acceleration of 1G corresponding to

this rollover. The maximum full-scale acceleration reading of the system is 50Gs. The maximum swing of the A/D in the recorder is 2048 counts either side of zero. This relation can be described as follows. 50Gs correspond to 2048 counts of the A/D, thus, 1G corresponds to 2048 divided by 50 which is equal to 41 counts of the A/D.

The gain of the operational amplifiers for all the three axes had to be adjusted such that a 1G change took place. This 1G change was equal to a change of 41 counts in the A/D readings. These readings were found after a few tests. Earlier the gains were adjusted using the full-scale range of the A/D as 0 to 4096 counts. But it was found that the correct range was from negative 2048 to 0 and 0 to positive 2048 counts of the A/D. The gains were modified again to reflect this change. The readings obtained for the three axes are summarized in Table 3.1.

Table 3.1. Gain adjustments for the three axes.

Axis	Gain (in counts)
X	241
Y	236
Z	244

The maximum gain setting for the operational amplifier was 255 counts. This corresponded to the change in resistance in the digital potentiometer, which was actually the *Rgain* of the operational amplifier. It was found that the variation of the gain of the instrumentation amplifier with respect to change in the resistance of the digital

potentiometer was not linear. This was due to the fact that the relation between these two parameters was non-linear and was given as

$$G = 5 + 200K / 400 + (255 - Gain) * 39.06 \tag{3.1}$$

The 39.06 factor is determined as follows: In the digital potentiometer, 256 counts correspond to 10KOhms (maximum range of the digital pot). Thus, 1 count in the digital potentiometer corresponds to 39.06 ohms of resistance.

As given by equation 3.1, the relationship between G (gain of the instrumentation amplifier) and Gain (equivalent counts for the change of resistance in the digital pot) is non linear and is further illustrated by Figure 3.7.

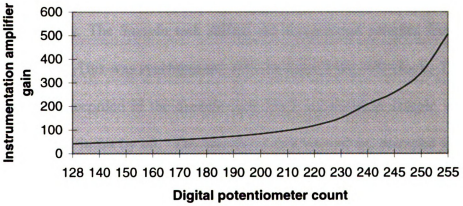

Figure 3.7. Non-linear variation of instrumentation amplifier gain.

3.8 Evaluation of Jitter

Jitter is the variation in sampling rate of the recorder. In a multitasking environment, conflicts between various tasks is always possible and care must be taken in order to provide a smooth transition from one task to another. These conflicts sometimes lead to variation in sampling frequencies or jitter. Jitter occurs when the control of the recorder is being transferred back to the scheduler from one of the running tasks. It may also occur between transitions of the tasks themselves. One type of jitter causes the loss of samples when the time period of sampling is smaller than the time taken for the events to be written into the memory. This task is aimed at determining the extent to which this kind of jitter would effect the logged data.

The two functions of the recorder which were responsible for sampling the data and moving the events from the event space to the buffer were the *SampIn* and *MovEvent* routines, respectively. The *SampIn* task shifted the acceleration samples from the A/D output to the buffer. This was synchronized with the help of the A/D clock. The data for the three axes was recorded in the *SampIn* task. Each acceleration sample was 12-bits long and was logged in bit by bit. The *MovEvent* task moved the recorded acceleration samples from the event space to the buffer. If the buffer was found to be in use, the control was transferred back to the scheduler.

Typically when a *SampIn* task request comes in between the *MovEvent* task, the recorder will not be interrupted until the *MovEvent* task has been completed. This will result in the loss of a few samples of data since the recorder did not accept the sampling request. The frequency of such an occurrence was determined. If the frequency of the

SampIn task being enabled between a *MovEvent* task was high, it can lead to loss of data and can be problematic. A stop-watch was used to set a time period or watch window between which the occurences of the *SampIn* task were accumulated. A counter was set to calculate the number of times the *SampIn* task was enabled within that time period. Sampling rate observations are given in Table 3.2.

Table 3.2. Results for sampling rate variation.

Sampling Rate	Occurrence
1000 Hz	Every 0.02 seconds
512 Hz	Every 0.1 seconds
400 Hz	Every 0.18 seconds

For frequencies less than 400Hz, the *SampIn* task is not enabled at all. Since it is not possible to operate the recorder at such low frequency, higher frequencies are chosen with certain constraints. Thus, only at frequencies of 1000Hz or greater can this phenomenon be problematic.

Table 3.3 contains a comparison between the occurrences for more realistic sampling frequencies of 1024Hz and 2048Hz.

Table 3.3. Occurrence of jitter at 1024Hz and 2048 Hz.

Occurrence for 1024Hz (ms)	Occurrence for 2048 Hz (ms)
10	1.01
20	0.978
10	1.02
10	1.03
10	0.975
10	1.02
10	1.04
10	1.04
20	1.03

Table 3.3 presents a very interesting observation. The occurrence of jitter for the 1024Hz sampling rate is once every 20 samples and for the case of a sampling frequency of 2048Hz, the occurrence is once every 2 samples. Thus, operating the recorder at 2048Hz will likely lead to the loss of data samples due to the frequent occurrence of jitter.

3.8.1 Conclusion

The above sets of observations indicate that operating the recorder at a sample rate of 2048Hz would lead to erroneous sampling data. Thus, as of now with a processor speed of 4MHz, the recorder would operate properly at a maximum sampling rate of only

1KHz. When the processor speed is increased to 16MHz and the code is optimized to reduce the number of iterations that a routine takes for execution, the recorder maybe able to operate at 4KHz without logging losses.

The above analysis highlighted the importance of time-stamping in a RTOS. By observing the times at which jitter occurs, the optimum sampling rate for the recorder was determined such that there was no loss of data. Time-stamping and synchronization of events is an important aspect of a RTOS. These characteristics make this system different and distinct from other simpler forms of operation like that of a non-preemptive kernel.

3.9 Memory Shutdown Feature

In situations where the number of events exceeded the maximum of number of allowed events, the recorder was designed to shutdown after saving all the event data in the EEPROM. This feature was called as a memory shutdown feature. Before the memory shutdown feature was introduced, the recorder behaved very inconsistently and unpredictably when it ran out of memory. Common among the observations noted during that stage of the verification process was the jamming of the trigger LED, corruption of recording control parameters (RCPs) values and the calibration parameters. This was due to the overlapping of memory to the staring memory location, once it ran out of memory space.

In order to facilitate the introduction of the memory shutdown feature of the recorder, a flow of steps was established and implemented. The addition of this feature demonstrated the flexibility of the RTOS. By making certain modifications in the

scheduler and adding additional tasks, the memory shutdown feature was implemented successfully.

A memory pointer was adjusted to point to the starting memory location. The initialization process and the biasing process were then executed and the recorder went through its normal routine of triggering and logging acceleration samples in the active mode. As soon as it was detected that there were less than 64 bytes remaining in the memory, the control was transferred to a *StopAct* procedure as given in Figure 3.8. This procedure stopped the event accumulation task, wrote all the event data into the EEPROM, masked all recording flags, and finally turned off the power to the analog section of the recorder to inhibit any further recording. At the same time, a memory full flag was set to indicate the memory full state.

The recorder had 32Kb of memory. Since each event occupied 32 bytes of memory, the maximum number of events that can be stored in the recorder was 1000. In situations where the number of events exceeded the maximum of number of allowed events, the recorder was designed to shutdown after saving all the event data in the EEPROM. This feature was also utilized in other routines like the *DelayStart* and the *DelayStop* routines, when they were added as additional features in the recorder.

```
Movlw     b'00000001'     ;mask all the tasks that need to be
stopped
Andwf     task2,F
Bcf       Apwr            ;turn off the analog power
Bsf       StateInfo,6     ;mark as an off time
Bsf       Task0,WriteOff  ;schedule the off marker writing
task
return
```

Figure 3.8. *StopAct* procedure.

Let E_o be the accumulated events and let E_{max} be the maximum number of events allowed which is equal to 1000. If E_o is greater than E_{max}, then recorder shuts down and saves all the event data into the EEPROM. This system was adjusted to operate on a 1024Hz sampling frequency.

This task was added to the existing code for the recorder. Care had to be taken to introduce this task at the right places. Certain problems were encountered when trying to link the flow of the recording process with this new feature. Prominent among them was the fact that if the recorder was not RESET, it wrapped around the memory and treated the gap caused due to disable triggering and sampling, as a dead time between packets. Moreover it restarted. This resulted in overwriting events and caused incorrect readings. This problem was traced to an incorrect call of the *StopAct* procedure. The *StopAct* procedure was not being called at the point at which the difference between the top of stack and bottom of stack (delta) was actually being decremented during the recording process. It was being called in the initialization memory procedure, which just operated on delta only once when initializing the memory and not during the actual recording process.

3.10 Summary

Various tasks that are required to evaluate an operating system of a data acquisition system have been presented. The previous sections reveal an underlying advantage of the real–time operating system over the conventional state machine design. The addition of the memory shutdown feature demonstrated the flexibility of the RTOS since this feature

was added after the entire RTOS had been written. The memory shutdown feature enables the recorder to log in correct data and prevents any data from being lost or corrupted. Most of the experimental work done to establish the importance of the RTOS has been explained.

CHAPTER 4

ANALYSIS OF LAYOUT

4.1 Introduction

The layout for Recorder B is described in this chapter. After the new printed circuit board (PCB) was received, the task of verifying and checking for the correct layout was done. This process is sometimes referred to as a signal integrity check. It was done to establish the correct links and layout of components in accordance with the design specifications. Signal integrity is the process of evaluating the signal flow in a circuit. It considers the issues of ringing, cross-talk, ground bounce, and power supply noise. Without due consideration of the basic signal integrity issues, typical high-speed products will fail to operate on the bench [9]. A good simulation, or a good laboratory demonstration, can usually put to rest any question about the efficacy of a particular solution. Some issues involved in the layout verification process are described in this chapter.

A greater percentage of PCB traces in new designs will likely require terminators. Terminators help control ringing and overshoot on transmission lines. As speeds increase, more and more PCB traces begin to take on aspects of transmission line behavior and thus will require terminators. Propagation delay plays an important part in the verification process and thus is an integral component of the trace analysis.

4.2 Noise Level Check

Every electrical circuit has noise. Noise becomes undesirable when the signal-to-noise ratio becomes low enough to adversely effect the operation of the electrical circuit. Electrical or electromechanical devices that cause fast or large changes in voltage or current are common sources of noise. Typical noise sources include lightning, power-line switching, switching inductive loads, arcs, fluorescent lights, machinery, inadequate separation of conductors of different levels, static discharge, harmonics, and ground loops [9]. Noise can appear in both power and signal (control) lines. Noise is often described by how it is coupled into a circuit. Five basic types of couplings are capacitive, inductive, radio frequency, common impedance, and conducted. Noise couplings relevant to this circuit are described below.

Capacitance is coupled into a circuit via a capacitive effect and is voltage-based. A voltage difference between two conductors separated by air or other insulating materials create a capacitor through which noise can be coupled. Common impedance coupling occurs when different circuits share common wires. Shared ground wires and long common neutrals or return paths can cause common impedance coupling. Conducted noise is coupled into a circuit via transmission of noise by wires or other conducting materials [9]. Noise in the recorder was determined by the process described below.

A potentiometer was connected as the input to the circuit in place of the accelerometers. The output of the circuit now reflected the response of the input, which was a potentiometer. This was simply the noise in the circuit, since there were no inputs.

The resulting output was the noise level of the circuit. It was found that the noise did not distort the signal to an extent that it could change the readings for the various events in the recorder. The signal to noise ratio was high and very tolerable.

4.3 The LED Problem

The recorder had light emitting diode (LED) indicators to display its state, *i.e.*, to determine if it was in the communication mode, active mode or triggering mode. There were two LEDs, one indicating the biasing and triggering mode, and the other indicating the communication (COM) mode. The LED that indicated the biasing and triggering was called as the trigger LED and the LED that indicated a COM operation was called as the status LED. The status LED was activated when the RCPs (recording control parameters) were being written into and read from the recorder. The trigger LED was activated when the recorder was removing the offset and also when the recorder was recording events.

The transition from the prototype board to the actual PCB for the recorder brought about a change in the LED configuration. This change was not detected at the time the transition was made. Thus, it led to many problems manifested as incorrect and random behavior of LEDs. Various problems were noted in the layout of the LEDs in the new PCB and to the ports of the microprocessor to which they were connected. Test points to determine the functionality of the LEDs were determined. These were defined as: the TRISE register (the register containing the I/O configuration for port E on the PIC 16C77), PORTD register (specification of the PORTD on the PIC16C77), and define statements for the status and trigger LEDs as given in Figure 4.1. Another approach that

was adopted to track down the problem was to force the recorder to trigger continuously by setting a very low trigger level in the interface software. By this method, the LED was made to show the trigger status of the recorder. At the point that the LED stopped indicating this status, the system was stopped and the execution process was checked by trace analysis with the help of an in-circuit emulator (ICE).

```
#define     TrigLED     1     ;Trigger LED (value=1=OFF on startup)
#define     StatLED     0     ;Status LED (value=0=OFF on startup)
```

Figure 4.1. LED define statements

The fault in the status LED was tracked down to the port configuration on the new PCB and the define statements used to reflect the firmware-hardware interface. The bit value of the port to indicate an output port was forced into the PORTD register. For both the LEDs, the ports at which they were attached were changed from PORTE in the prototype board to PORTD in the new PCB. This was done in accordance with the new layout to develop an optimized design. The trigger LED problem was traced to faulty assignment of the output port pin in the PORTD register. More specifically, instead of PORTD, 2 (second pin of PORTD) as the output pin for the trigger LED, the third pin of PORTD was assigned as the output bit.

The trigger and status LEDs were assigned default values that they would take when the recorder is first turned on. The trigger LED was assigned a value of 1 on startup and the status LED was assigned a value of 0 on startup as shown previously in

Figure 4.1. These values indicated that by default the LEDs were turned off and activated only when a recording or COM operation changed their value.

4.3.1 Conclusion

This problem demonstrated the hazards attached to switching over from an older system to a newer system. This process was a part of the verification process to ensure the correct functioning of the recorder. It showed the importance of an indicator like an LED in the complete design process. Without an indictor, it would be very tedious to establish the state of the recorder and check for its correct operation. Some of the original operating code was changed to reflect the changes in the new board.

4.4 Reference Voltage Check

The reference voltage was generated by the analog circuitry. It was used to provide a reference level for the offset correction mechanism. The reference voltage was fixed at 1.65 V (half the supply voltage to the digital circuit). This voltage was checked from time to time to maintain the accuracy of offset correction. In these routine checks it was found that the reference voltage was varying by a small amount in some of the readings. This variation was of the order of a few millivolts. This variation was traced to the potential divider circuit used to generate the reference voltage. The resistors used in the potential divider circuit did not have sufficiently tight tolerance and were deviating from

their original value. It was checked to see how much difference this variation can have on the offset correction process. The tolerable range is within 5 counts of the A/D output. The average deviation was calculated to be 0.011 V. So considering that the A/D has a maximum count of 1000_{16} counts, we get the following relation. A supply of 1.65 V corresponds to 800_{16}. Also, 800_{16} is equal to 2048_{10}. Thus, 1 V corresponds to 2048_{10} divided by 1.65, which is equal to 1241_{10}. Hence, 0.011 V corresponds to 13.65_{10} counts of the A/D. From the above calculations, it was observed that the average deviation was greater than the tolerable range of 5 counts. It was noted that the difference or the variation exceeded the allowed tolerance range. The only problem that this discrepancy could cause was disproportionate peaks of the events about the zero level. In order to avoid this occurrence, the resistors were changed a higher tolerance range device.

4.5 Summary

The previous discussion highlighted the problems that could occur in real-time. PCB testing is an integral part of any design and this was made evident by the detection and solution of significant problems like the LED problem and the variation in the reference voltage. These were purely hardware issues with a slight hint at the firmware design. Solutions to these problems eased the task of adding new features to the recorder. It also simplified the task of analyzing the RTOS by eliminating the bugs in the circuit.

CHAPTER 5

SLOW TRACK RECORDING

5.1 Introduction

The slow track data logger allowed the user to collect information about the surrounding environment of the recorder. This was non-triggered data and was collected independently of any triggered data like shock or vibration events. The recorder was equipped with an internal temperature sensor that was monitored and provisions for humidity and pressure sensors were also given . This was again an optional feature and could be enabled by the user as and when required. Slow track data was collected in-between triggered acceleration events. If a recording interval of less than the maximum event duration was set, it could lead to a situation where a long event could cause a slow-track sample to be missed. This would lead to erroneous data.

The addition of this feature took advantage of the flexibility of the RTOS. The successful implementation of this feature lead to the conclusion that a RTOS was indeed a more flexible and, more importantly, a feasible option as a platform for self-contained data recorders.

5.2 Procedure and Implementation

Slow track recording necessitated the need to differentiate the event data from the slow track data. This was essential when both slow track and fast track recording were occurring concurrently requiring the introduction of three sub-tasks that maintained data integrity and did not allow data corruption to take place. These tasks were defined as: logging slow track data into the buffer, transferring data from the buffer to the stack and then, finally, storing the data from the stack to the EEPROM. A very important condition which had to be kept in mind when storing the data in the buffer was the memory full condition. The code segment in Figure 5.1 checked this condition.

```
.
.
delta = stack pointer – linear address pointer
size of slow track event = slowsize
if
        delta – slowsize > 0
        then    memory full is false
        else
if
        delta – slowsize = 0
        then    memory full is true.
.
.
```

Figure 5.1. Check for memory full condition.

The implementation of this task demonstrated that when the fast track data (event data) is being accumulated, the slow track function is waiting for it to be enabled by the

user. As soon as the fast track event relinquished control of a routine that was to be used in slow track recording, the latter took over the control mechanism and started logging slow track data. Figure 5.2 displays the synchronization of the fast track and slow track tasks. Thus, more activities took place in the same interval of time. A binary semaphore, which in this case was a slow track enable bit, was responsible for ensuring that only one routine was accessed by only one mode of recording. Each task's stack pointer and other important registers associated with that task played the role of the context switch. They were responsible for saving all the important registers of an outgoing task and introducing new registers for incoming tasks. The slow track task was declared as a low priority task in the scheduler since it was a user enabled task and was not always active.

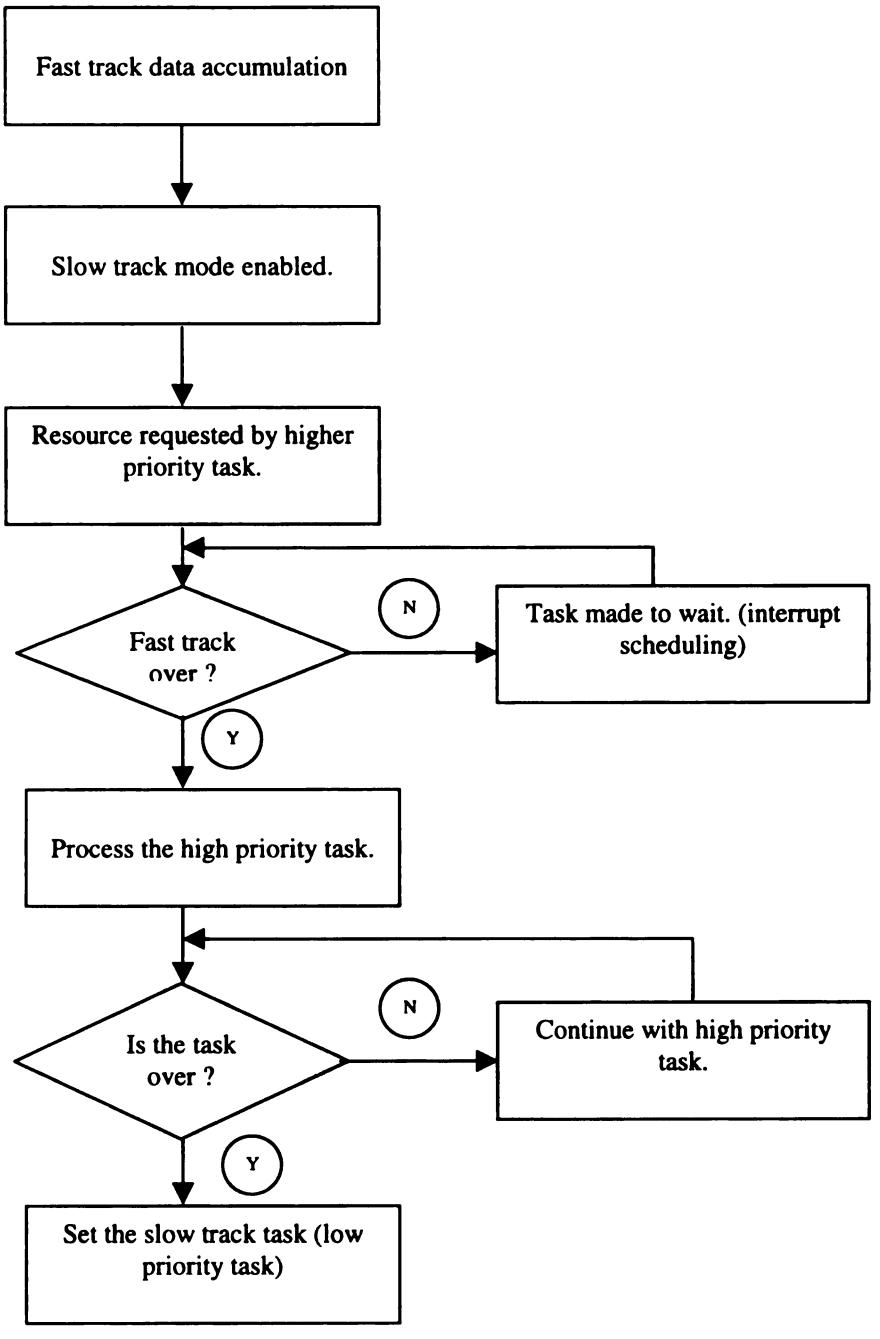

Figure 5.2. Synchronization of fast track and slow track recording modes.

5.3 Problems and Limitations

While implementing the slow track mode, it was noticed that the typical A/D conversion time was 40 microseconds. As soon as the slow track data was logged in, the processor was put to sleep for the time the A/D conversion took place. The end of conversion was determined when an interrupt was generated to wake up the processor from the sleep mode. The time for the sleep instruction was noted as 160 microseconds. Thus, as soon as the processor was put to sleep, a loss of 120 microseconds took place for every 8-bit conversion. This reduced the processing speed for the system and lead to a loss of data samples.

5.4 Summary

About 160–180 microseconds were used for the A/D conversion with the sleep command. Since this resulted in the loss of a few samples of data, a different approach was adopted to solve this problem. In the real circuit, *i.e.*, the circuit with the microprocessor (not with the emulator system) a polling mechanism was used to run the slow track task and the interrupt structure was not used at all. Although this eliminated the delay in the circuit, it gave rise to some circuit noise that was caused since the processor was not put to sleep when the A/D conversion was in progress.

CHAPTER 6

DELAY START AND DELAY STOP FEATURES

6.1 Introduction

Delay start and delay stop features were added to the recorder as part of the requirements of the user and also to demonstrate the flexibility of the RTOS on which the recorder was built. These features were added to the recorder after the entire verification had been completed. The addition of these tasks highlighted the capability of the RTOS to add and delete features as and when required.

6.2 Delay Start

The delay start feature allowed the user to 'program' the recorder to be turned on at a predetermined time or after a set time interval . The delay allowed the user to configure the start of the recording process. Delay stop feature allows the user to disable the recording process after a set interval of time. Since these tasks were included after the whole design of the recorder had been completed, there were few precautions which were kept in mind before starting on the design process. The task had to be enabled only when enabled by the user. Thus, the interrupt routine was modified to include the interrupt for the delay start recording. After the completion of the task, the instance of the delay start

task was immediately cleared. This was done since constant activation of this task will put the recorder in an infinite loop. An enable bit was defined to check for the status of this feature.

6.2.1 Algorithm (Delay Start)

A stepwise description of various tasks that performed to complete the delay start task is given below.

1. The real time clock is incremented to look for next instruction.
2. The system checks to see if the user has enabled the delay start feature. This is an optional feature and is not activated by default. It has to be specifically enabled by the user.
3. If delay start is not enabled, the system runs its interrupt routine and goes through the initialization process. Then it jumps into the scheduler where the next task is executed.
4. If delay start is enabled, then the present time in the RTC (a 32-bit value) is subtracted from the value set by the user.
5. Until the difference becomes zero, the delay start task is not initiated. As soon as the difference becomes zero, the delay start task is set in the interrupt routine and it is initiated in the scheduler.
6. The control drops down to the delay task routine from the scheduler and the first command in the routine is to clear the delay start task.

7. The sample enable is checked. Sample enable is a changeable user-defined parameter.
8. If the sampling is disabled then the recording is stopped and control is transferred to the scheduler.
9. If the sampling is enabled then the recording task is started and is terminated when the analog power to the recorder is turned off.

Figure 6.1 depicts a typical delay start process.

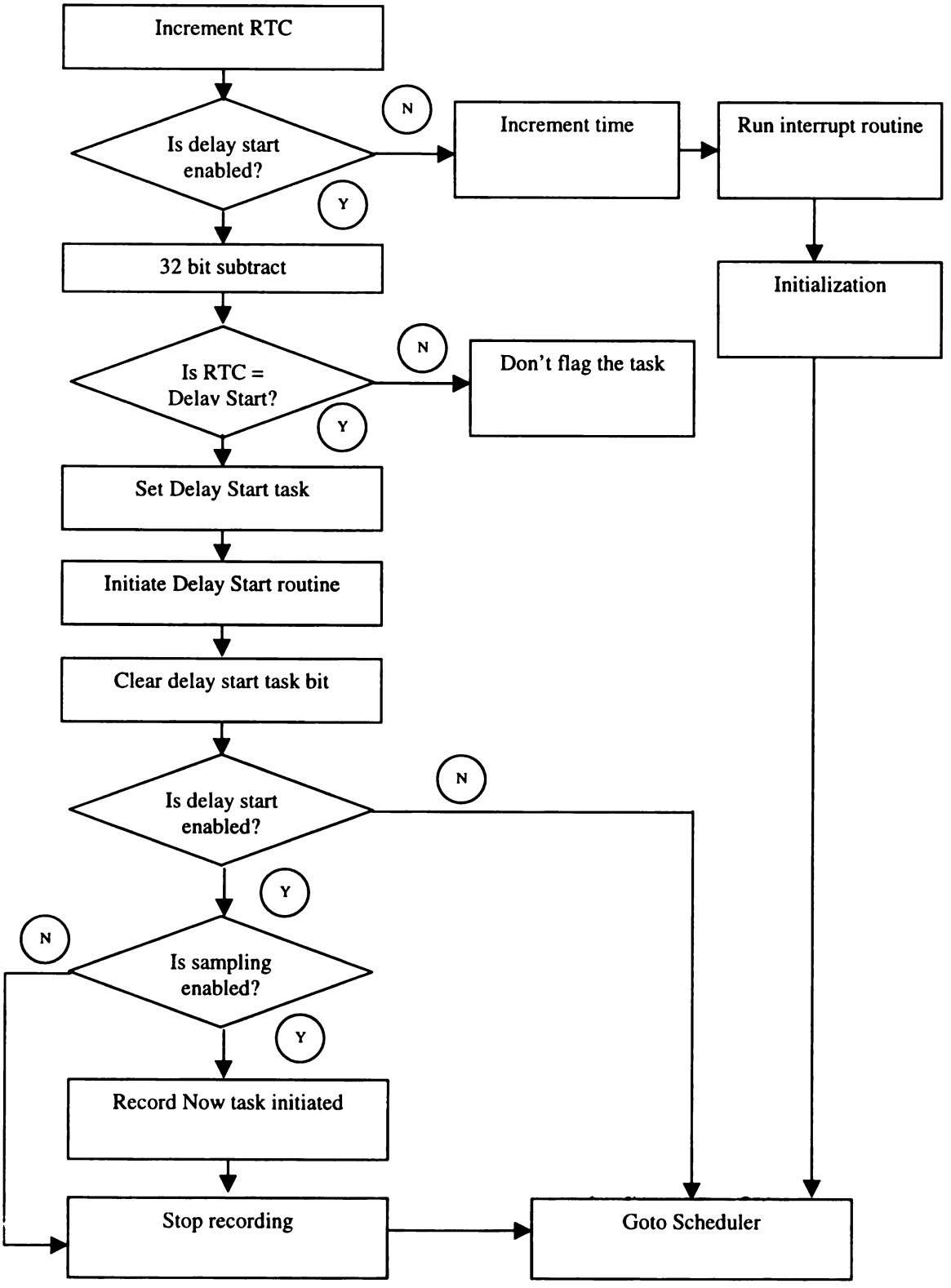

Figure 6.1. Delay start process.

6.3 Delay Stop Feature

The delay stop feature had a similar structure as the delay start feature. The only difference being that after the user-defined time had elapsed, the recorder shuts down instead of commencing the recording process. Essentially the time detection procedure remained the same as that of the delay start process.

6.3.1 Algorithm (Delay Stop)

A stepwise description of various tasks that were done to complete the delay start task is given below.

1. The real time clock is incremented to look for next instruction.
2. The system checks to see if the user has enabled the delay stop feature. This is an optional feature and is not activated by default. It has to be specifically enabled by the user.
3. If delay stop is not enabled, the system runs its interrupt routine, goes through the initialization process and transfers control to the scheduler that determines which task needs to be performed next.
4. If delay stop is enabled, then the present time in the RTC (a 32-bit value) is subtracted from the value set by the user.

5. Up until the difference becomes zero, the delay stop task is not initiated. As soon as the difference becomes zero, the delay stop task is set in the interrupt routine and it is initiated in the scheduler.
6. The control drops down to the delay stop task routine from the scheduler and the first command in the routine is to clear the delay stop task.
7. The sampling is disabled, analog power is turned off, recording is stopped and control is transferred to the scheduler.

6.4 Summary

Delay start and delay stop features were essential to the recorder since there are certain situations in which the recorder is not easily accessible. In such cases, the recorder can be programmed by using the delay start/stop features. A typical example will be in case the recorder has to be placed in an inaccessible location. Under such circumstances it is useful to program the recorder to start and stop at the desired interval of time and continue its recording process without any problems. The scheduler in the operating code was an integral part of the flexible mechanism of the recorder. These new features were just added as tasks in the scheduler and were given a lower priority of execution since they were not always enabled.

CHAPTER 7

COMPARISON OF THE TWO RECORDERS

7.1 Introduction

This section gives an analytical breakdown of some of the distinctive features of the two recorders. The most striking differences are highlighted and their implications on the performance of the recorder are pointed out. Various task management structures that make the scheduling algorithm for the Recorder B are:

1. Task management (Create a task, start a task, suspend a task, terminate a task, *etc.*);
2. Event management (Wait on an event, set an event, clear an event, *etc.*);
3. Communications management (Packet protocol management);
4. Buffer/stack management (Create a buffer, reset stack, add to stack, *etc.*);
5. Timer management;
6. Memory management.

The design based on the above-mentioned structures allows for a very flexible operation of the recorder. This structure incorporates the feature of concurrent processing.

Recorder A is built on a state machine design, *i.e.*, the outputs and the next state of the recorder are functions of their inputs and present state. For example, either toggling the switch or initiating the delay routine can initiate the recording process. In Recorder A, modification to include the delay start routine to initiate the recording process is problematic because the state of the execution process and the inputs to the process will have to be kept in mind. RTOS design incorporates interrupts, which are helpful in solving real time problems. Interrupt management is a very important aspect of a good RTOS design. The interrupts should be arranged in such a way that they affect only the portion of the code targeted and all the other important registers are saved. This enables the restoration of all registers after the interrupt has been serviced and the kernel can start where it left off. Such an arrangement would facilitate an error-free run of the recorder [7]. Adding another task in the scheduler will require adding a corresponding marker as a bit defining the task in the interrupt structure. The task can also be defined within another task.

7.2 Comparison of the Two Recorder Structures

Eight items of comparison, presenting the advantages and disadvantages of the two operating systems, are offered in this section.

1. The operation of Recorder A, which is driven by interrupts, has certain drawbacks. If a section of code is being executed and an interrupt request is serviced, the program is terminated irrespective of the extent of execution and the control is transferred to the ISR. This may result in mishandling of data. Recorder B has a scheduling of interrupts, *i.e.*,

interrupts run the scheduled tasks. Therefore, even if an interrupt occurs in between a scheduled task, it will not be serviced until the scheduled task has been completed. The interrupt request will be serviced according to the priority of the scheduled task. This may result in the loss of few samples of data, but does not corrupt the data to the extent of the other option.

2. Communication with the recorder while taking samples is possible in the scheduler structure. This multitasking feature allows the user to process data concurrently while it is actively recording. Considering that both these tasks have a priority assigned to them, both of the tasks can run concurrently as long as they maintain inter-task communication between them. Since the target registers in the two tasks mentioned above are different, they do not create any resource conflicts/hazards. Such an arrangement is not possible with Recorder A due to the absence of any distinct tasks. All operations are inter-linked making it very tedious to do concurrent processing.

3. The concept of auto-zero correction in Recorder B (keeping the history of offsets) enables the correction of offset even if the recorder comes out of an event. If this task is scheduled as a low-priority task, keeping in mind that there are other more important tasks like event accumulation taking place, such an arrangement will make the auto-zero correction process very data safe. Not only will the task be executed as and when required, but it will also result in utilizing the processor to the maximum. Since Recorder A lacks any base for a task structure, if auto zero correction is introduced, it will take up

valuable processing time. There is no way that this can be run as a low priority task in Recorder A.

4. In Recorder A, both the slow track and fast track tasks have to be sampled in one sample period to insure data integrity. Since the slow track task is low priority in the scheduler in Recorder B, it is sampled when the higher priority tasks of fast track event accumulation are completed. But the RTOS structure enables the slow track task to wait for it to be enabled while the fast track data is being sampled. Thus, the prioritized scheduler can actually determine when to sample a particular task and when to avoid it. Prioritizing the tasks will help to run a higher priority task, which must get done even if a lower priority task is being executed, or the data will be lost. Recorder A does not posses this flexible capability of prioritizing events and, thus may lead to data corruption.

5. In Recorder B, the tasks communicate between each other and also communicate with the subroutines. This inter-task communication helps to synchronize events and reduce troubleshooting time. Since Recorder A does not have any distinct blocks of code representing different tasks, time taken to troubleshoot is much higher.

6. Recorder B has a nice feature of buffer space. Messages are passed between tasks through the buffer. Buffers allow the messages to be looked at once the task is ready. One message can be sent to many tasks at the same time. This feature allows the user to operate on the highest prioritized messages before handling the rest. Since there is no

buffer space in Recorder A, the messages/commands/operands have to be sent exactly at the time an event requires them, which can create synchronization problems [2].

7. Recorder A is built on a state machine design, *i.e.,* the outputs and next state of the recorder are functions of their inputs and present state. For example, either toggling the switch or initiating the delay start routine can initiate the recording process. In Recorder A, modification to include the delay start routine to initiate the recording process is problematic because the state of the execution process and the inputs to the process will have to be kept in mind. In the RTOS design, tasks can be added as long as binary semaphores control them and do not cause any conflicts. In the delay start task, the enable bit presents the flag to initiate the task and its priority in the scheduler determines when it is executed.

8. An example of task-subroutine communication in Recorder B is the hardware offset correction. Whenever the offset goes out of range, the biasing task initiates the hardware offset. Synchronization of this process is very crucial and can be achieved only in the RTOS structure. Recorder A cannot incorporate this feature because of poor synchronization between different events and random interrupt requests.

CHAPTER 8

CONCLUSION

The design, implementation, and verification of a real time operating system in embedded systems, more specifically in self-contained data recorders has been presented in this thesis. The results of its application have been presented and analyzed indicating that real time operating systems are a viable, flexible and more advantageous ways to implement a data recorder design.

This chapter summarizes the work developed in this thesis, highlighting the contributions that have emerged through its completion, and identifying future research directions.

8.1 Research Summary

The review presented in Chapter 2 exposed the relationships among various software and hardware techniques. Also, in Chapter 2 tools used for the completion of this research were presented and explained. The structure of a self–contained data recorder was explained and the modifications needed to complete the objectives of this research were pointed out. The architecture of the system was described with the emphasis on the microprocessor, the serial communication bus and software used to analyze data. Further, the problem statement described the flexibility problem in the recorders with a state machine design structure. This statement, in addition to formally presenting the problem,

made evident the complexity involved in the search for a solution. An important point stressed in this chapter was the difference between the state machine and real-time operating system structure.

The main experiments of this research were presented in Chapter 3. A thorough review of all issues with the verification process of the recorder operating system was described. The observations were translated into inferences that were later used to provide the conclusion to the research. The material covered in this chapter summarized the test points in a verification process. Testing and analysis of the test points was then explained. It also included the memory shutdown feature that was an important addition to the recorder system. It led to the solution of unexplained problems and made the recorder stable and error free.

In Chapter 4, the problems encountered while changing the recorder from a prototype design to a PCB were elaborated. The analysis of the PCB was described. Various malfunctioning sections of the recorder like the reference voltage section were identified and their problems were solved.

Chapters 5 and 6 presented new modes of recording that were useful in analyzing the flexibility claim that was made in the earlier chapters. Slow track recording and delay start/stop recording features were added to the existing recorder, which highlighted the significance of the RTOS over the state machine structure. The addition of these features without any problems led to the conclusion that the RTOS was a better and more flexible way to design self-contained data recorders.

An explicit comparison of the two recorder structures, namely RTOS and state machine, were detailed in Chapter 7. Of special significance was the point made about

the task-subroutine and inter-task communication in the recorder, which supplemented the conclusion about the flexible operation of the RTOS based recorder.

8.2 Contributions and Conclusions

The objectives of this thesis required an approach that combined diverse areas of knowledge. At the same time, the development of this work produced several contributions, which enhance several of these areas. Three major contributions can be attributed to this research. Below, a summary is presented of these new concepts.

A Better Understanding of the Real-Time Operating System in Self-Contained Data Recorders - The results obtained supporting the newly developed methodology have provided a better insight into the performance of a data acquisition system working in a multitasking environment. Various problems associated with concurrent execution of tasks were discussed and their remedies were suggested. The relevance of a scheduler to maintain inter-task communication was highlighted.

Addition of New Tasks to the Recorder - Three new tasks were added to the recorder structure to verify the flexible operation of the RTOS. These tasks were also used to check for any inter-task conflicts and data corruption. Successful implementation of these tasks after the recorder structure had been formed, demonstrated the efficacy of the flexible design.

An Explicit Comparison of the Two Recorder Structures - A comprehensive comparison of the two recorder structures, namely the real-time operating system (RTOS) and the state machine structure, was explained. Of special significance was the point made about the task-subroutine and inter-task communication in the recorder, which supplemented the conclusion about the flexible operation of the RTOS based recorder. With this contribution, the most important reason for the development of a different structure for a data acquisition system has been established.

Based on the comparison of the two recorder structures a comprehensive list of verification points was established. These points helped to analyze the RTOS and state machine designs. Offset correction, verification of sampling frequency, calibration of the recorder, evaluation of jitter, memory shutdown feature and inter-task communications were the specific verification points used in the comparative analysis of RTOS and state machine structures.

8.3 Future Work

The validation of a RTOS as an operating system for self–contained data recorders, as was indicated earlier, only partially covers the need for a flexible solution of data acquisition systems. Similar approaches are needed in the hardware design to provide for a comprehensive solution to rigid data acquisition systems. Flexibility in hardware design has been initiated by the introduction of digital potentiometers as a source of

digitally controlled gain for the analog input signal. More work has to be done to improve and optimize the hardware design of recorder.

A multiprocessor architecture, which efficiently addresses some of the critical software and hardware issues involved in the design of such systems, can also be used for data acquisition systems. The objective of the multiprocessor architecture can be to efficiently distribute the computational tasks of the data acquisition system amongst different processors. The criterion used to distribute various tasks can be divided on the basis of functionality. For example, all tasks associated with sampling, A/D conversion, and signal processing of analog signals may be assigned to a processor. This approach clearly introduces a form of parallelism that will increase overall system throughput.

8.4 Closing Remarks

The idea behind this thesis, more than simply providing an insight to a RTOS based recorder structure, is the creation of a unified design methodology that could transform the way in which conventional data recorders are manufactured.

This is not an easy goal. After reaching this point and reflecting on all the work it took to complete this thesis, one realizes that this is just a small step in reaching that ultimate objective. The foundation has been laid and the objectives are clear.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Jensen, Douglas. Real-time operating systems. [Online] Available http://www.realtime-os.com/realtime.html, January 28, 1999.

[2] "Application Notes 585," *Microchip Technology Incorporated*, 1997, pages 1-15, August 1998

[3] "Microchip PIC 16C7X Data Sheet 8 bit CMOS microcontroller with A/D Converter." *Microchip Technology Incorporated*, pages 83-162, August 1998.

[4] "PICMASTER In-circuit emulator users guide." *Microchip Technology Incorporated, pages*, pages 1-150, August 1998.

[5] Moore, Ralph. How to use real-time multitasking kernels in embedded systems. [Online] Available http://www.smxinfo.com/howto/howto.htm, September 19, 1995.

[6] Frantz, William. Object Oriented Transaction Processing in the KeyKOS® Microkernel. [Online] Available http://www.cis.upenn.edu/~KeyKOS/KeyTXF/KeyTXF.html, September 1993.

[7] Tsoukarellas, Manthos. Systematically Testing a Real-time Operating System. [Online] Available http://www.computer.org/micro/mi1995/m5050abs.htm, October 1995.

[8] "Operating Systems, Internals and Design Principles," 3rd Edition, William Stallings, pages 10-150, July 1998.

[9] Microchip. Development Tools. [Online] Available http://www.microchip.com/10/Tools/picmicro/emulator/mpice/index.htm, November 20, 1998.