EMBEDDED SIGNAL PROCESSING ON MICROCONTROLLERS

APPROVED BY

SUPERVISING COMMITTEE:

Brian L. Evans, Supervisor

Lizy K. John

I dedicate this report to my Mom and Dad, without whose inspiration and support this report would have been a dream.

EMBEDDED SIGNAL PROCESSING ON MICROCONTROLLERS

by

AMEY ARUN DEOSTHALI, B.E.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May, 1998

Acknowledgments

I would like to thank my parents for providing me support and boosting my confidence when I needed most. They have played an instrumental role in making me what I am now.

I would like to thank my advisor Dr. Brian L. Evans for guiding me in my research area and providing me financial assistance. He has been extremely motivating and helped me realize the meaning of *perfection* and *professionalism*. His idea of holding seminars for the Embedded Signal Processing Laboratory (ESPL) group meeting helped me develop my presentation skills. I am also grateful to him for assigning me the responsibility of organizing departmental DSP seminars.

I would like to thank Mr. Shawn McCaslin, Cicada Semiconductor Corp., for letting me work on his ideas. I feel myself extremely lucky for getting the opportunity to work with someone having 15 years of industrial experience. His practical suggestions and hints on my projects were extremely valuable.

I would like to express my gratitude to my committee members- Dr. Lizy John and Dr. Brian L. Evans, for thoroughly reading my report. I would like to thank Dr. Alan C. Bovik, who is the director of Laboratory for Vision Systems, for teaching excellent courses on Digital Signal Processing and Digital Image Processing. I would like to express my thanks to Dr. Jack Lee for offering me a teaching assistantship in Spring 1997 semester, and relieving me of any financial burden during my first semester at The University of Texas at Austin.

I would like to thank Srikanth Gummadi for working with me and helping me in one of my projects for the report. I also enjoyed the food he used to cook (especially chicken dishes). I would like to thank Güner Arslan for helping me with the microcontroller implementation issues related to my projects. I would also like to thank Biao Lu and Wade Schwartzkopf for proof-reading my chapters. I am also grateful to Marios Pattichis, Thomas Kite, Dong Wei, Kartick Suriamoorthy, Jong-il Kim and Niranjan Damera-Venkata for their helpful suggestions. I would like to show my appreciation towards my friends Arindam Nag, Amit Apte, Suhrud Atre, Razak Mohammedali, Rajeev Mahajan, Ritu Chaba, and Shylaja Katakam. I enjoyed the debates we had on some strange topics.

I would also like to express my thanks to my brother, Ajit, and my sister-in-law, Rina, for their words of wisdom. I would like to thank my cousin, Swati, and her husband, Vasudeo, who live in Austin, for making my stay in Austin memorable.

AMEY ARUN DEOSTHALI

The University of Texas at Austin
May, 1998

EMBEDDED SIGNAL PROCESSING ON MICROCONTROLLERS

AMEY ARUN DEOSTHALI, M.S.E.
The University of Texas at Austin, 1998

Supervisor: Brian L. Evans

The use of microcontrollers for embedded applications has become commonplace. Microcontrollers provide single-chip, low-cost solutions for low-bandwidth, low-power applications. In this report, I present two communications applications and two new approaches to realize these applications on a PIC microcontroller from Microchip Technology Inc.

I present a new approach to decode the WWVB broadcast time information transmitted by National Institute of Science and Technology (NIST) and generate an accurate frequency reference calibrated to the NIST primary standard. The key innovations are new, zero-buffering algorithms and microcontroller implementations to decode WWVB time code information and the use of a pulse-width modulator to generate an accurate frequency reference.

In this report, I also present a new, low-complexity ITU-compliant algorithm for Dual Tone Multiple Frequency (DTMF) touchtone signal detection. The key innovations include the use of adaptive notch filtering, accurate frequency estimation techniques, and new sophisticated validation logic to check whether a detected DTMF digit satisfies the ITU requirements. The proposed method is the first zero-buffering, ITU-compliant method that can be implemented on an 8-bit microcontroller.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The past two decades have witnessed tremendous growth in the digital computer industry and integrated circuit fabrication. Due to the availability of inexpensive, relatively fast digital circuitry, a trend has been to replace as much analog circuitry with digital circuitry as possible. Of course, not all analog circuitry can be replaced by digital circuits. Some wide bandwidth and ultra-low power applications are better served by analog circuitry. However, where digital circuits are available and have sufficient speed for the desired tasks, they are usually preferable for several reasons.

Digital systems provide better accuracy control than analog systems. Accuracy control is very difficult in analog systems because it depends on the tolerance of the components that are used in the system. In a digital system, increasing accuracy is as simple as adding bits to data words. When a signal is sampled and quantized to N bits, its output signal-to-noise ratio (SNR), which is expressed as $10\log_{10}\left(\frac{\text{signal power}}{\text{noise power}}\right)$, is proportional to $6.02N$. SNR can be increased by increasing the number of bits used for quantization. In analog systems, increasing the signal-to-noise ratio requires boosting the signal power and/or replacing analog circuits with those of lower tolerance.

In communication systems, the analog and digital system performance is related to the available bandwidth. Analog system performance is propor-

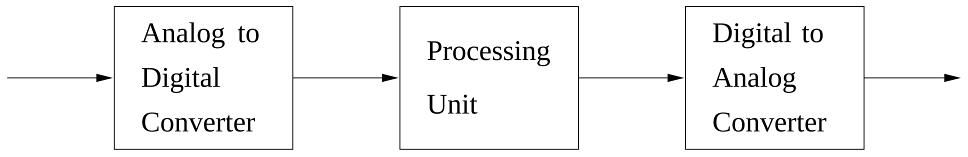


Figure 1.1: Block diagram of a digital signal processing system

tional to the square of the available bandwidth, whereas digital system performance is exponentially related to the available bandwidth because bandwidth is directly proportional to the number of bits. Thus, digital communications systems generally outperform analog communications systems.

Digital implementation provides increased flexibility for modification, as compared to analog implementation. Analog data can be stored on analog media (e.g. cassette tape). However, digital data can be stored using a error resilient format, e.g. Reed-Solomon codes for audio compact disks. In addition, digital implementation is generally cheaper than analog implementation.

A simple model of a digital signal processing system is shown in Figure 1.1 [1]. Analog signals are converted into digital form by an Analog-to-Digital (A/D) converter. A Digital-to-Analog (D/A) converter performs the dual task of converting a digital signal to analog signal. The processing unit implements a set of operations on a signal using a mixture of hardware and software. The function of the processing unit is to transform the digital signal in the desired manner (e.g. filter the signal to remove noise). Since the processing unit operates on a digital signal, the entire system is called a digital signal processing system. The operations that are performed by the processing unit (by hardware and/or software) are called digital signal processing algorithms.

The term digital signal processing (DSP) has become a common term.

The increasing supply of faster and better chips to support DSP algorithms has fueled an explosive growth of signal processing applications in various fields such as communications, networking, and consumer electronics. In the late 1970s, programmable DSP processors were developed to match the data-intensive vector-oriented computations in DSP algorithms. Over the past fifteen years, DSP processor performance has increased at the rate of approximately forty percent per year [2]. It is expected that the current $3 billion market for DSP processors will reach $50 billion over the next ten years [3]. However, it is important to note that not all DSP applications need DSP processors. Microcontrollers and general purpose processors are sufficient for many signal processing applications [4].

1.1 Embedded Systems

An embedded system is a part of a product with which an end user does not directly interact or control [5]. Another feature that characterizes embedded systems is that they are designed to perform a specific task. In embedded applications, hardware is unique to a given system, and software has to be specifically written so as to make optimum use of available resources. Codesigning the hardware and software for the system instead of committing too early to a particular hardware design and writing the software later is an important aspect of embedded systems [6].

The embedded systems market is large and diverse. Table 1.1 surveys products that have embedded systems in them. Flexibility and modularity are important in embedded system implementation. Most embedded systems

Alarm clocks	Pagers	Video disk players	AM/FM radios
Car stereos	CD players	Cellular phones	Music synthesizers
Calculators	Telephones	Video telephones	Computer scanners
Video games	Modems	Microwaves	Credit card readers
Disk drives	Smart cards	Digital cameras	Answering machines

Table 1.1: Embedded systems products

are heterogeneous. That is, they use different implementation technologies for various subsystems of an embedded system. Next, we profile some of the commonly used embedded processor technologies.

1.1.1 Digital Signal Processors

Digital signal processors are fast computer chips capable of doing complex mathematical operations. Most DSP processors have the ability to perform a single cycle multiply-accumulate operation, which is a commonly found operation in signal processing applications. The multiply-accumulate operation may be used to compute vector dot products. Accumulation is typically maintained at double the data word width to reduce truncation effects in a running summation.

Almost all DSP processors have distinct data paths for parallel access of instructions and operands. This allows the processor to perform multiple operations in a single instruction cycle (e.g. up to 11 RISC instructions can be performed in one DSP instruction cycle). DSP processors also provide special addressing modes, such as modulo addressing (for circular buffers in digital filters) and bit-reversed addressing (for the fast Fourier transform). Most DSP processors have specialized hardware features, such as zero-overhead looping and low-latency deterministic interrupt handling, which improve processor performance. Additionally, some of the processors also incorporate peripheral and I/O handling mechanism such as Direct Memory Access (DMA). For a more detailed study of the features of digital signal processors, we refer the reader to [7].

Programmable DSP processors provide real-time performance which is critical in certain products such as modems, audio CD players, and cellular phones. DSP processors are available in compact sizes and have relatively low power dissipation for the amount of available computational power. DSP processors can perform many of the tasks which are currently being performed by general-purpose processors. DSP processors can manipulate digital signals in a variety of ways. As a result of these advantages, the DSP processor market has grown tremendously. Research shows that the DSP processors will continue to penetrate not only the embedded systems market but also the one currently being occupied by general purpose processors [8]. The leading manufacturers of DSP processors by revenue are Texas Instruments, Lucent Technologies, Analog Devices, and Motorola [7]. DSP processors range in cost from $6 in volume (of 1,000 units) to $200 in volume, excluding the Texas Instruments TMS320C80 which costs $400 in volume.

1.1.2 Microcontrollers

A controller is a device that is used to control some process or aspect of an environment. A microcontroller is a highly integrated programmable chip which is ideally suited for control applications. By only including features specific to the task (control), cost is generally low. Usually, microcontrollers are not used as standalone processors. They are used in embedded systems with many different peripherals. Hence, all microcontrollers provide general purpose I/O ports for peripheral interfacing. Some microcontrollers, however, also provide enhanced peripheral features such as pulse-width modulators, RS232 interfaces, and LCD drivers. Microcontrollers are a *one-chip solution* which can enable a reduction in part counts and design costs [9].

The next question that immediately comes to mind is: *"What is really an embedded microcontroller"*? Simply stated, an embedded microcontroller is a controller which is embedded into a larger system for purposes other than general purpose computing. To make a more rigid definition is difficult, but one way to define it is that embedded microcontrollers are highly integrated chips customized according to the requirements of the application in which they are used. They usually have limited on-chip memory, e.g. 1000 words of program memory (1000×14 bits) and 36 bytes of data memory on a PIC16C71. Increasingly, microcontrollers are now being used in a broader range of applications. They are typically used where high throughput is not required. Some of the leading manufacturers of microcontrollers by revenue are Motorola, Philips, Intel, and IBM. Microcontrollers range in cost from $0.50 to $15 in volume.

1.1.3 Hybrid DSP/Microcontrollers and ASICs

The first phase of a design cycle for any new product starts with the formulation of specifications and requirements for realizing a product. It is then followed by algorithm design and simulation. The final phase of the design cycle concerns the implementation of individual subsystems using specific programmable, configurable, and dedicated technologies.

In engineering projects, cost is very important. The consumer demand is for better, smaller and cheaper products. The first generation of consumer electronics products are often built using available commercial off-the-shelf components. In products like cellular phones, however, the available power, area, and volume are limited. As a result, industries are trying to put more hardware and software features specific to the application onto a single chip. One trend is towards hybrid processors that have microcontroller and DSP processor features, e.g. the Motorola MC56800 processor. An even higher level of system integration is possible by using *processor cores* as the basic building blocks for Application Specific Integrated Circuits (ASICs). Designers can tailor these ASICs to the exact requirements and requisite functionality of the products. Core-based ASICs provide *system-on-a-chip* solutions, thus enhancing the capabilities of the product [10]. The *system-on-a-chip* integrates the functions of multiple individual integrated circuits (e.g. microprocessor, memory, DSP, and I/O) onto a single piece of silicon.

1.2 Algorithm Development for Microcontrollers

1.2.1 Microcontroller Features

Today's microcontrollers come in various flavors. Even in a particular microcontroller family, many variations may exist. For example, the Motorola MC68HC11 microcontroller has more than 50 members. Microcontrollers have all or some of the following features.

1. **Architecture:** Two types of architecture are Princeton and Harvard.

   (a) **Princeton architecture** uses a single data bus for accessing both instructions and operands. Thus, it is structured for sequential execution and stores both instructions and operands in the same memory. This potentially slows down program execution. Examples of processors using this type of architecture are the Motorola MC68HC11 and Intel 8051.

   (b) **Harvard architecture** is oriented towards parallel architecture. It has separate data buses for instructions and operands. Thus, it has distinct program and data memories. This type of architecture speeds up execution but requires more silicon. PIC microcontrollers from Microchip Technology Inc. use this type of architecture.

2. **Instruction Set:** Instruction sets can be classified into Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC).

   (a) **CISC** is the traditional organization used for microcontrollers. The processors having this type of organization typically have a large number of instructions targeted towards specific control tasks, and many different addressing modes. An advantage of the CISC organization is that many of the instructions are macro-like, allowing one instruction to be used for many other simpler ones. A key advantage is the packing of bits in the opcodes to reduce program word length, which reduces program length in bytes and ultimately yields lower power consumption.

(b) **RISC** organization is slowly penetrating the microcontroller market. A processor with this structure provides very few instructions. Operations are, however, register-to-register with only LOAD and STORE accessing memory. The instruction size is uniform and hence RISC architecture is better suited for pipelining. The advantages of this type of organization is that it reduces chip size and pin count. RISC is better matched to compiler technology and better suited for higher clock rates. The RISC instruction set is more orthogonal, permitting each instruction to use any register or addressing mode.

3. **Memory:** Almost all microcontrollers have on-chip memory (either program or data or combined). The various types of memories found on-chip are listed below

   (a) **Electrically Erasable Programmable Read Only Memories (EEPROM)** can be erased and programmed without ever removing them from a circuit.

   (b) **Erasable Programmable Read Only Memories (EPROM)** can be programmed electrically and can be erased using ultraviolet light.

   (c) **FLASH PROM** provides a better solution than regular EEPROM whenever there is a requirement for large amounts of non-volatile program memory. It is faster and permits more erase/write cycles than EEPROM.

   (d) **One Time Programmable (OTP)** memories can be programmed only once. These types are good for limited production runs in industries having short product design cycles.

4. **Power management:** Microcontrollers are relatively low-power processors primarily because their clock rates are low and their word lengths are short. Some microcontrollers also have brownout protection circuitry which resets the device when the operating voltage (V_{cc}) is lower than the brownout voltage. Optionally, some of them may have a `SLEEP/WAKEUP` mode. This mode is very helpful in reducing the power consumption when the processor is idle.

5. **I/O interface:** All microcontrollers have one or more bi-directional ports for peripheral interfacing. Occasionally they may also have an on-chip Universal Asynchronous Receiver Transmitter (UART) or a Universal Synchronous Asynchronous Receiver Transmitter (USART).

6. **Analog Interface:** Some more expensive versions of microcontrollers also have Analog-to-Digital (A/D) and Digital-to-Analog (D/A) converters on chip. This greatly reduces the peripheral access time. Some microcontrollers also have a pulse-width modulator (PWM) unit for generating waveforms.

7. **Timer:** Almost all microcontrollers have at least one timer. Some of them have a watchdog timer. A watchdog timer is a programmable timer, which when activated, resets the processor after a specified interval. Such

a mechanism provides a means of graceful recovery from a system problem.

8. **Miscellaneous features:** Some of the other features included on the microcontrollers are interrupt handling capabilities, code protection bits and bit manipulation instructions such as bit set, bit clear and bit test instructions.

1.2.2 Algorithm Development

All microcontrollers come with a combination of some of the previously described features. It is important to understand the features available on the processor selected for algorithm implementation. Even though many microcontrollers can run on a high frequency clock (as high as 20 MHz), they do not necessarily execute an instruction in a single clock cycle. Some RISC microcontrollers employ pipelining, but still require few clock cycles to execute a single instruction cycle. This instruction cycle speed is necessary to find the MIPS (million instructions per second) power of the processor.

Timing plays an important role in peripheral interfacing. Most of the peripherals run at lower speeds than the processors. Some DSP processors, such as the Analog Devices SHARC processors, have automatic wait state insertion to interface with the slow running peripherals [7]. However, microcontrollers do not have this feature, so wait states need to be implemented by the developer in software. Other aspects of timing that need to be taken into consideration are the interrupt latency and subroutine call latency. These aspects will play a role in deciding how to implement algorithms on a microcontroller and whether to use macros or subroutine calls (provided that enough code space is available).

Multiplication is one of the most common operations found in signal processing applications. Most microcontrollers do not have a multiply instruction (except the ones which have DSP capabilities added to them). In this case, multiplication needs to be emulated in software (for an example see Appendix A.1). This severely restricts the scope of microcontrollers to applications requiring lower processing power (in terms of the number of multiply-accumulate operations). However, it is often possible to optimize multiplication operations for a particular application at hand, thus indirectly increasing the processing power. For example, a multiplication between a variable and a constant can often be implemented faster than a multiplication between two variables (refer to Appendix A.2 for an example). Also, a multiplication result may only be needed to a few bits of precision which reduces the number of instructions required.

Most microcontrollers also do not have a divide instruction, so a divide operation would also need to be emulated in software. This can prevent an implementation from satisfying the timing requirements of the target application (see Appendix B for an example). So, it is desirable to eliminate as many divides as possible. But if division is required, then it may be possible to optimize the divide routine in much the same way as a multiply routine.

Another feature characteristic of many signal processing applications is the repeated execution of a section of a code. Many DSP processors provide hardware looping, thus eliminating the overhead associated with updating a loop index and the pipeline flushes associated with branching. Microcontrollers, on the other hand, do not provide such features. They do, however, provide special control instructions (such as branching on a bit change) which can be used to reduce the looping overhead. These instructions play a crucial role during program flow control.

1.3 PIC Microcontroller

When deciding which microcontroller to use in a particular design, we need to consider many issues. Some of them are listed below:

1. Fixed-point processor or a floating-point processor [11]
2. The processing power required for a particular application
3. Acceptable interrupt and subroutine latency
4. Peripherals integrated on chip (e.g. an A/D converter)
5. Available programming languages and software tools
6. Availability of development support for a particular processor
7. Accessibility of documentation

All of the above criteria were critical in our selection process for the appropriate microcontroller. Our applications can be implemented on a fixed-point processor. We decode a broadcast time and frequency standard using on the order of 25,000 multiplications per second, so a low-end microcontroller can be used. The decoding of Dual Tone Multiple Frequency (DTMF) signals requires on the order of 250,000 multiplications per second, so a high-end microcontroller will be required. A detailed analysis of the required processing power for each application is performed at the end of each chapter. We selected the PIC microcontroller, which is a RISC microcontroller from Microchip Technology Inc. This processor has high processing power for a microcontroller (5 MIPS at 20 MHz). Because timing was critical for decoding of broadcast time and frequency standard, we selected the PIC16C7x series with an internal A/D converter for this application. Also there is a mailing list for PIC programmers where one can find useful hints and solutions to their problems (PICLIST@MITVMA.MIT.EDU). In addition, Microchip provides a software library for these processors containing useful routines such as binary coded decimal (BCD) to binary conversion, 16-bit divide/multiply, and notch filter, which dramatically reduces prototyping time and cost.

1.4 Scope of the Report

In this report, I develop two low-cost systems to decode standardized transmitted signals: broadcast time/frequency signals, and Dual Tone Multiple Frequency signals. I implement each decoder on a PIC microcontroller from Microchip Technology Inc. [12]. These applications show how microcontrollers can be used in low-bandwidth applications to reduce the system cost.

Chapter 2 describes the WWVB US time and frequency standard. We present our new, low-complexity decoding algorithm for obtaining global time information and describe our novel method for generating a frequency reference. We also present the microcontroller implementations for both decoders.

Chapter 3 focuses on decoding of Dual Tone Multiple Frequency (DTMF) signals. It summarizes the international standard for DTMF signals established by the International Telecommunication Union (ITU). We describe our new, efficient, ITU-compliant DTMF decoding algorithm. A detailed analysis of the memory requirements and the computational complexity for a microcontroller implementation and a DSP processor implementation is also presented in Section 3.4 of Chapter 3.

Chapter 2

Real-time Low-cost Decoding of Broadcast Time and Frequency Standard

2.1 Introduction

The notion of time and frequency is very important to us. Without even realizing it, we use time and frequency in our day-to-day lives. We need to be on time to catch a flight. Parking meters need to keep track of the time that has expired since the insertion of coins. Power companies supply power at 60 Hz. The telephone dial tone is a 400 Hz tone added to a 350 Hz tone.

Time and frequency are in fact very much interrelated. The rate at which events (pulses) occur is called *frequency*. One pulse (event) per second has a frequency of 1 Hz. An interval over which these events occur is called *time*. Wristwatches maintain the notion of a second by counting oscillations of a quartz crystal.

For most of us, high-precision accuracy of time and frequency information is not so important. Some businesses, however, need precise frequency and global time information, including power companies, radio and television stations, astronomical observatories, the aerospace industry, and telephone companies. These industries need to calibrate timing to a reliable, internationally recognized standard.

The Time and Frequency Division of the National Institute of Science and Technology (NIST), which is located in Boulder, Colorado, is responsible for maintaining and distributing the time and frequency standard for the United States. The NIST time standard is a highly accurate standard based on a Cesium atomic clock. NIST makes this standard available to the public by telephone and radio services. The time of day, frequency reference and time interval are broadcast from radio stations WWV located in Ft. Collins, Colorado, WWVH located in Kauai, Hawaii, and WWVB located in Ft. Collins, Colorado, as well as GOES (Geostationary Operational Environmental Satellite) satellites [13]. Note that WWV, WWVH and WWVB are names of radio stations and not acronyms. The transmitted time is Coordinated Universal Time (UTC) a.k.a. Greenwich Mean Time (GMT). UTC of course differs from the local time usually by a specific number of hours.

WWV and WWVH are high-frequency services while WWVB is a low-frequency service. WWV transmits time and frequency information at 2.5, 5, 10, 15 and 20 MHz. WWVH transmits time and frequency information at 2.5, 5, 10 and 15 MHz. WWVB uses a 60 kHz carrier for transmitting the time and frequency information. Existing WWVB time decoders are commercially available in the form of integrated circuits. The integrated chips offer a single chip solution with or without an RS232 interface. Clocks using such integrated receivers range in price from $30 to $100. Some wristwatches such as the *Atomic wristwatch from Arckon* use the WWVB signal to maintain accurate time. However, they are rather expensive and range in price from $150 to $200.

In our decoder, we first convert the 60 kHz analog signal into a digital signal and then track the power of the digital signal to decode the time information. We also obtain an accurate frequency reference synchronized to the NIST primary frequency standard. This frequency reference can be used as a secondary standard to calibrate other frequency sources since it is referenced to the NIST primary standard. The key innovations are new, zero-buffering algorithms and microcontroller implementations to (1) decode WWVB time code information and (2) generate an accurate frequency reference using a pulse-width modulator, from the WWVB signal. The key to the microcontroller implementation is to use bandpass sampling of the WWVB signal to reduce the sampling rate (and hence the data rate) by a factor of 20. A key for the frequency reference generation is the Friedman interpolator which is explained in Section 2.2.2.

Our technique has low complexity and is ideal for an implementation on a microcontroller which range in price from \$2 to \$15 in volume of 100. We prototype the time information decoder on a 10 MHz, 2.5 MIP PIC16C71 and the frequency reference generator on a 10 MHz, 2.5 MIP PIC16C72 microcontroller, which cost \$4 and \$6 respectively in volumes of 100 units. The total cost of the decoder including the antenna should be approximately \$15 to \$20. Our decoder provides a cost-effective alternative to the existing WWVB decoders.

2.2 Background

We use the 60 kHz pulse-width modulated WWVB signal to obtain global time and an accurate frequency reference. The power of WWVB signal carries the time code information. We explain WWVB time code in Section 2.2.1. Section 2.2.2 explains the Friedman interpolator. We use the Friedman interpolator to obtain the phase information of the incoming 60 kHz signal and generate a reference frequency.

2.2.1 WWVB Time Code

Radio station WWVB is located near Ft. Collins, Colorado. WWVB continuously broadcasts time and frequency signals at 60 kHz, primarily for the continental United States. The WWVB signal carries only the time code and no voice announcements. The transmitted accuracy of WWVB is normally better than 1 part in 100 billion (1×10^{-11}). Day-to-day deviations are less than 5 parts in 1000 billion (5×10^{-12}). The BCD time code can be received and used with an accuracy of approximately 0.1 ms. When proper receiving and averaging techniques are used, the received accuracy of WWVB should be nearly as good as the transmitted accuracy [14].

The WWVB time code is synchronized with the 60 kHz carrier and is broadcast continuously at a rate of 1 pulse per second using pulse-width modulation. Each pulse is generated by reducing the carrier power 10 dB at the start of the second, so that the leading edge of every negative-going pulse is on time. Full power is restored either 0.2, 0.5, or 0.8 seconds later to convey either a binary "0", binary "1", or a *position marker*, respectively [15].

The WWVB time code is sent in a Binary Coded Decimal (BCD) format (see Figure 2.1). The binary-to-decimal weighting is 8-4-2-1. The most significant bit is sent first. The decimal number is obtained by multiplying each bit in the binary group by its weight and then adding the four products

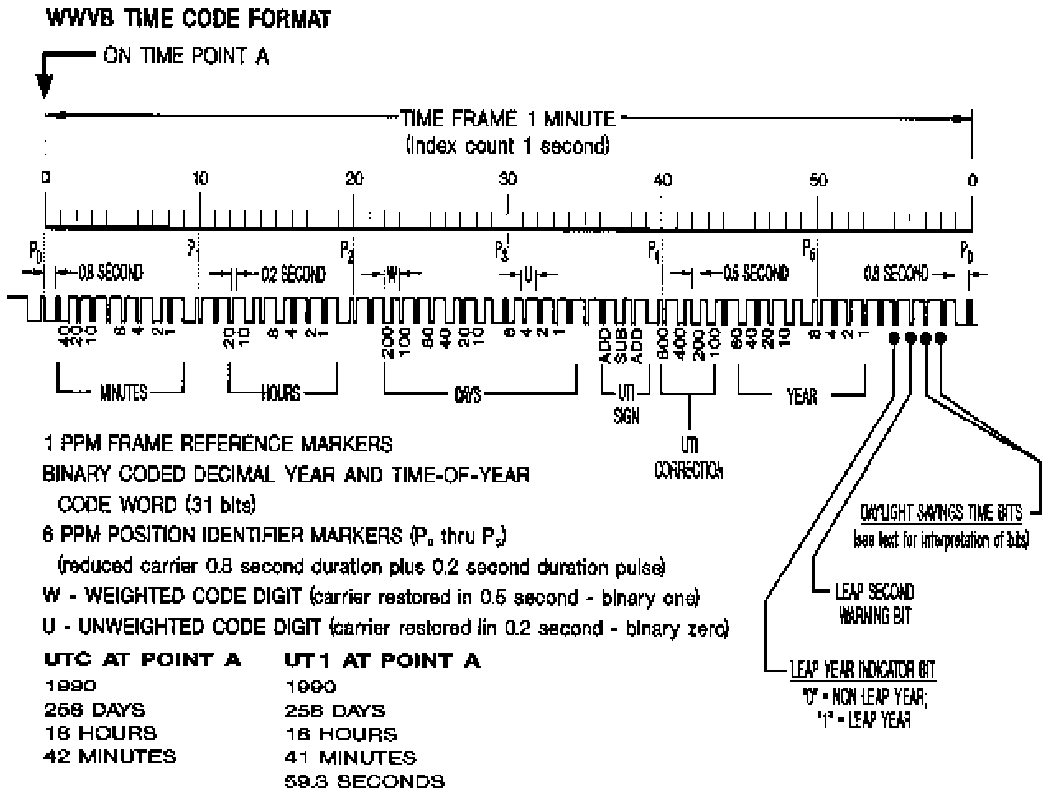

WWVB TIME CODE FORMAT
ON TIME POINT A
TIME FRAME 1 MINUTE (Index count 1 second)
0 10 20 30 40 50 0
0.8 SECOND
0.2 SECOND
W
U
0.5 SECOND
0.8 SECOND
MINUTES
HOURS
DAYS
UT1 SIGN
UT1 CORRECTION
YEAR
1 PPM FRAME REFERENCE MARKERS
BINARY CODED DECIMAL YEAR AND TIME-OF-YEAR CODE WORD (31 bits)
6 PPM POSITION IDENTIFIER MARKERS (P0 thru P5)
(reduced carrier 0.8 second duration plus 0.2 second duration pulse)
W - WEIGHTED CODE DIGIT (carrier restored in 0.5 second - binary one)
U - UNWEIGHTED CODE DIGIT (carrier restored in 0.2 second - binary zero)
UTC AT POINT A
1990
258 DAYS
18 HOURS
42 MINUTES
UT1 AT POINT A
1990
258 DAYS
18 HOURS
41 MINUTES
59.3 SECONDS
DAYLIGHT SAVINGS TIME BITS (see text for interpretation of bits)
LEAP SECOND WARNING BIT
LEAP YEAR INDICATOR BIT
"0" = NON LEAP YEAR;
"1" = LEAP YEAR

Figure 2.1: A sample time code format broadcast by radio station WWVB

together. For example, the binary group 0101 is equal to 5 which comes from $(0 \times 8) + (1 \times 4) + (0 \times 0) + (1 \times 1)$.

Every minute, the WWVB station transmits a time code that contains the current minute, hour, day of year, 2 digits of the current year, a UT1 correction, and Daylight Savings Time (DST) and leap year indicators. UT1 is a time scale derived by astronomers who monitor the speed of the Earth's rotation [14]. UT1 corrections are to the nearest 0.1 s and show if UT1 is positive or negative with respect to UTC. A BCD group of four bits is used to represent a decimal number from 0–9. Some bits in the BCD groups are unused, but may provide additional information in the future. Two BCD groups are needed to express the hour (00 to 23), minute (00 to 59), and year (00-99); and three groups are needed to express the day of year (001 to 366). To represent units, tens, or hundreds, the basic 8-4-2-1 weights are simply multiplied by 1, 10, or 100 as appropriate. The coded information refers to the time at the start of the one-minute frame. Seconds are determined by counting pulses within the frame. Each minute begins with a frame reference pulse lasting for 0.8 seconds. A position identifier pulse lasting for 0.8 seconds is transmitted every 10 seconds.

2.2.2 Friedman Interpolator

Our WWVB decoder for the frequency reference is based on the Friedman interpolator. The Friedman interpolator is an algorithm to estimate the frequency of a single sinusoid in white noise, based on the computation of the interval between zero crossings [16]. In the following analysis, only the negative to positive going zero crossings of the sinusoid are considered.

At the arrival of the first positive sample following the zero crossing, the estimate of the period of the sinusoid $T_e(n)$ is computed as

$$T_e(n) \; = \; [K(n) - \delta(n) + \delta(n-1)]T_{\mathrm{s}} \tag{2.1}$$

where $K(n)$ is the number of sampling intervals between the positive samples following the $(n-1)^{th}$ and n^{th} zero crossings, and $\delta(n)T_s$ and $\delta(n-1)T_s$ are the time intervals between these zero crossings and the next positive samples (Figure 2.2).

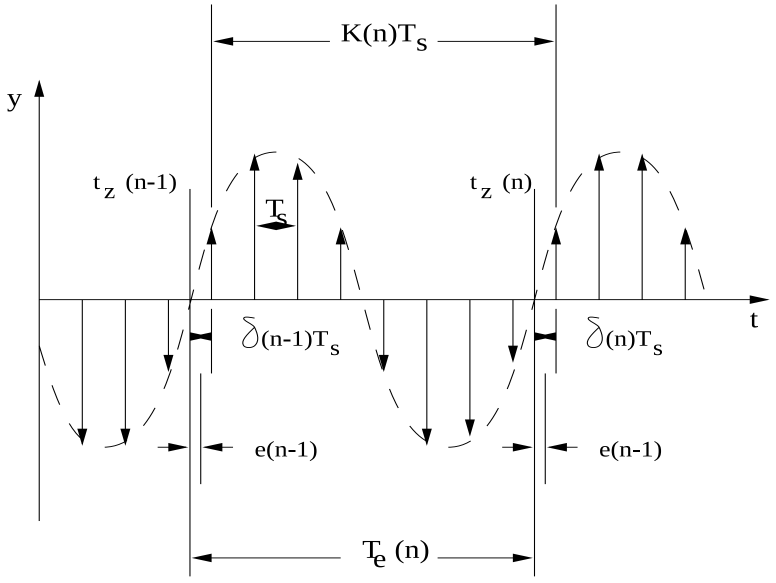


Figure 2.2: Frequency estimation by zero-crossing detection

If $e(n)$ is the error made in the computation of the n^{th} zero crossing, it can be shown [16] that

$$T_e(n) = T_{\sin} + e(n) - e(n-1) \tag{2.2}$$

where $T_{\sin}$ is the actual period of the sinewave. The spectrum of $T_e(n)$ contains a DC component which is equal to the period of the incoming signal, and the spectrum of the error signal which is concentrated in the high-frequency region. Thus, by using an appropriate lowpass filter, the period, and thus the frequency, can be computed to an arbitrary degree of accuracy (of the order of 10^{-4} to 10^{-6}).

2.2.3 Microchip PIC16C7X Microcontroller Family

We implement the Friedman interpolator and the rest of our WWVB decoders using 8-bit PIC microcontrollers. The PIC microcontrollers are available in four families– the 8-bit 8-pin microcontroller family (PIC12CXXX), the 8-bit base-line microcontroller family (18, and 28 pin PIC16C5X), 8-bit mid-range microcontroller family (18, 20, 28, 40, and 44 pin PIC16CXXX) and the 8-bit high-end microcontroller family (40, 44, 64, and 68 pin PIC17CXXX). The PIC12CXXX microcontrollers are well-suited for low-end control applications. The mid-range microcontrollers are better suited for WWVB decoders than the base-line family because of the advanced peripheral options available on the mid-range family. Since peripheral interface latency is an important issue, we choose PIC16C7XX series for implementation of the decoder. The PIC17CXXX microcontrollers offer a single instruction multiplication operation, but have far more computational power than we need for our WWVB decoders.

The PIC16C7XX microcontrollers contain a high-performance RISC CPU. They have only 35 single word instructions. All the instructions are single cycle, except for program branches which require two cycles. Operating speed is 200 ns per instruction cycle when the CPU is running on a 20 MHz clock (5 MIPS) or 400 ns per instruction cycle when the CPU is running on a 10 MHz clock (2.5 MIPS). On the PIC16C7XX microcontrollers, the length of 1 word is 14 bits. Other features included on the microcontrollers are a watchdog timer, an 8-bit analog-to-digital converter, a timer module, and a power saving `SLEEP` mode. Some microcontrollers of the PIC16C7XX series also have a pulse-width modulator (PWM) unit. The WWVB decoder needs a PWM to generate a frequency reference.

We used the PIC16C71 microcontroller for decoding the broadcast time standard. The PIC16C71 has 1000 words of internal program memory and 36 bytes of data memory. It does not have a PWM unit and has one timer module. The program memory is UV erasable. The microcontroller additionally has a 4-channel 8-bit A/D converter. The cost of PIC16C71 is about \$5 in volume (of 100 units).

Generation of a frequency reference requires a PWM unit, and hence we choose PIC16C72 for this application. PIC16C72 has 2000 words (2000×14 bits) of internal memory and 128 bytes of data memory. It has a PWM unit and a 5-channel 8-bit A/D converter. It also has 3 timer modules. The cost of PIC16C72 is about \$6 in volume (of 100 units).

2.3 Decoding of Broadcast Time

The power level of the 60 kHz transmitted signal carries the WWVB time code information. We need to track the 60 kHz signal power levels to extract the time code. Our decoder has two parts— a signal processing front-end and a decision logic back-end. The signal processing front-end is composed of a sampler, which samples the continuous-time analog signal to convert it into a discrete-time digital signal, and a power estimator, which estimates the power of the received signal. The decision logic back-end makes decisions on the bits that are supplied by the signal processing front-end. The back-end is responsible for calculating the correct local time from the transmitted UTC.

2.3.1 Signal Processing Front-end

Although we will be focusing on the signal processing algorithms performed by the microcontroller in this report, we consider the analog receiver to be a part of the signal processing front-end. The receiver consists of a loop antenna for receiving the low-frequency band, and a high-Q amplifier for selecting and amplifying the 60 kHz signal. We use this analog modulated signal as an input to the analog-to-digital converter. Below, we discuss the operations performed by the microcontroller on this analog signal.

Sampling and quantization

In our application, broadcast time decoding only needs estimation of the power level of the signal, and frequency reference generation requires estimation of the signal phase. For estimating the power of the 60 kHz signal, we convert the analog signal into a digital signal. We use a PIC microcontroller with an 8-bit A/D converter to do the conversion. An 8-bit converter has a dynamic range of approximately 48 dB. When decoding WWVB signals, the phase estimation algorithm for generating the reference frequency is robust and gives good performance at low SNR (such as 10 dB). Hence the dynamic range provided by 8 bits is sufficient and the use of an 8-bit converter is justified.

The Nyquist theorem [17] states that if $x_c(t)$ is a bandlimited signal with

$$X_c(j\Omega) = 0 \qquad \text{for} \quad |\,\Omega\,| \;>\; \Omega_N \tag{2.3}$$

then $x_c(t)$ is uniquely determined by its samples $x[n] = x_c(nT), n = 0, \pm1, \pm2, ..,$ if and only if

$$\Omega_s = \frac{2\pi}{T} > 2\Omega_N. \tag{2.4}$$

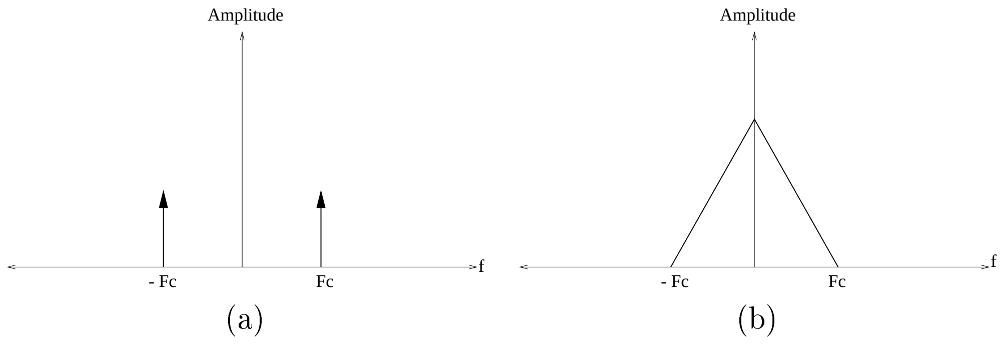


Figure 2.3: Spectrum of a (a) sinusoidal signal, and (b) complex signal

where $X_c(j\Omega)$ is the Fourier transform of $x_c(t)$, Ω_s is the sampling frequency, and Ω_N is the maximum frequency in the signal. The frequency Ω_N is commonly referred as the *Nyquist frequency*, and the frequency $2\Omega_N$ that must be exceeded by the sampling frequency is called the *Nyquist rate.* Sampling at any frequency at or below the *Nyquist rate* will cause aliasing, which is a phenomenon which causes high analog frequencies to appear as low digital frequencies.

The WWVB signal is a pulse-width modulated (PWM) message which is in turn modulated by a 60 kHz sinusoid. The PWM message consists of 1 pulse per second (1 Hz). Since the WWVB signal is extremely narrowband, we model it as a pure sinusoid. According to the Nyquist criterion, we need to sample this signal at more than 120 kHz. This frequency is too high for an application to be implemented on a microcontroller. However, in this application, we show that aliasing is actually beneficial and allows the application to be realized on a microcontroller.

Figure 2.3 shows the difference in the spectral content between a complex signal and a pure sinusoidal signal. Sampling a pure sinusoid at a

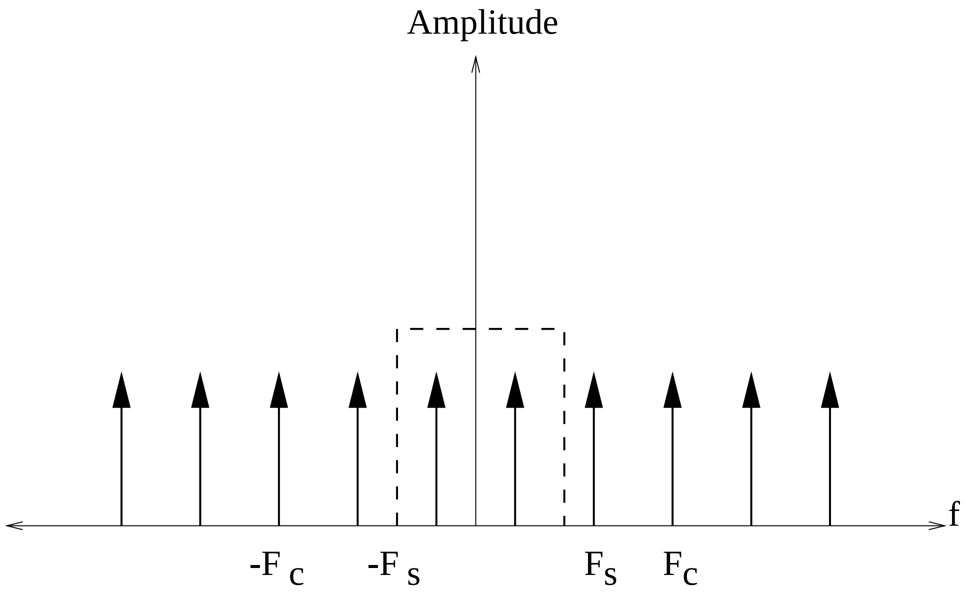


Figure 2.4: Sampling of a sinusoid signal of frequency F_c at a frequency F_s which is less than the Nyquist rate i.e. $F_s < 2F_c$. The dotted lines show the spectrum of a lowpass filter to extract the sinusoid of lowest frequency.

frequency which is lower than the Nyquist rate and following it by a lowpass filter would result in another sinusoid but of a different frequency [18], as shown in Figure 2.4. Since our aim is to track the changes in signal power, the frequency of the digitized signal is irrelevant. Hence it is possible to sample the 60 kHz signal at a frequency lower than the Nyquist rate.

The sampling frequency is restricted by the following five constraints:

1. The transmitted frequency must not be an integer multiple of the sampling frequency. Otherwise, the lowpass component of the aliased signal would be the DC component. Thus, all the information about the signal would be lost.

2. The sampling frequency should be low enough that we can perform all

the required calculations on the present sample before the next sample arrives.

3. The sampling frequency should high so that we receive as many samples as possible to achieve a more accurate estimate of the power.

4. The sampling frequency should map as an integer timer count on a microcontroller.

5. The sampling frequency should be greater than 2 Hz, which is twice the bandwidth of the message signal.

The first condition eliminates all frequencies whose integer multiple is 60 kHz, e.g. 1, 2, and 4 kHz, from being used as sampling frequencies. We could use 16 kHz for sampling, but it violates the second condition because it only gives 62.5 μs to perform the calculations between incoming samples. Taking all of these factors into consideration, we choose 6.25 kHz for the sampling frequency. Details about the choice of 6.25 kHz will be covered in Section 2.5.

Power estimation

Accurate estimation of the power of the received 60 kHz signal is a critical task in the decoding of the WWVB signal. Each pulse is generated by reducing the carrier power by 10 dB at the start of each second. Depending on the restoration of power 0.2, 0.5, or 0.8 seconds later, a binary “0”, binary “1”, or a *position marker*, respectively, is conveyed.

The signal power estimator has to be simple, efficient, accurate, and fast. Since microcontrollers do not have a multiply instruction, we want to reduce the number of multiplications in our power estimator. Hence, we cannot use a high-order filter. In our implementation, we have used a single-pole infinite impulse response (IIR) filter to estimate the signal power [19]. For a real pole located at α, the relation between the estimated power $P(n)$ and the received signal $x(n)$ is

$$P(n) \;=\; \alpha\, P(n-1) \;+\; (1-\alpha)\, x^2(n) \tag{2.5}$$

where $0 < \alpha < 1$. For stability, $\mid \alpha \mid < 1$. When α is close to 1, the current estimate of the power depends more on the previous estimate of the power than on the instantaneous value of signal power (like a weighted average). When α is close to 0, the power estimate depends heavily on the instantaneous value of the signal amplitude, causing large fluctuations in the power estimate. Thus, value of α is selected somewhere between 0 and 1. We use a value of 0.875 for α which has an exact representation as an 8-bit fractional number. This value of α gives a very fast and accurate estimate of the power.

Initially, the power estimator tracks the signal to find two signal power levels– one corresponding to high power and another corresponding to low power. Once the two power levels have been established, we set a threshold midway between these values. As soon as the estimator finds the power below the threshold, it starts a timer to count the duration of the low power level. Based on the timer count, the estimator then decodes either bit *0*, *1*, or a *position marker.*

2.3.2 Decision Logic Back-end

The output of the signal processing front-end is a binary “0” or binary “1”. After obtaining the binary bits, we have to decode them so as to obtain the correct time. This would include adjusting the day and time depending on the following bits:

1. daylight savings time
2. UT1 correction
3. leap year

Since the transmitted time is UTC, the product which implements this WWVB decoder also needs to adjust the displayed time depending on which time zone (Eastern, Central, Mountain or Pacific) was selected.

The control logic needed to implement the decoder is large, but it is manageable due to the type of instructions on the PIC microcontroller. For example, the PIC microcontroller has *branch on bit test* instructions. These instructions can test any bit in any register and can branch accordingly to a given destination.

2.4 Generating a Frequency Reference

Many techniques exist for frequency calibration [20]. One such technique uses a time interval counter to measure the frequency difference with a high degree of precision (on the order of 10^{-12} s). Another technique for comparing two signals involves measuring the change in phase between the two signals. Both signals are applied to a linear phase comparator and the difference is used to calibrate the frequency source. In our novel technique, we use a pulse-width modulator (PWM) which is available on some of the PIC microcontrollers to generate an accurate frequency reference. We maintain the accuracy of the PWM frequency generator by calibrating it with the WWVB signal which is referenced to the primary frequency standard at NIST.

The Time and Frequency Division of NIST also maintains a primary frequency standard. The 60 kHz signal which carries the WWVB code is calibrated to the NIST primary frequency standard. Thus, it is extremely accurate. It is possible to obtain a secondary frequency reference calibrated to the NIST primary frequency standard with the help of a Friedman interpolator [16]. We use a new technique for generating a frequency reference. The Friedman algorithm is modified for estimating the phase information of the received signal. By knowing the phase of the sinusoid, we can know the exact time that has elapsed from the previous zero crossing. We have generated a frequency reference of 1250 Hz with a relative frequency of at least 10^{-4} to 10^{-6} using this phase information. Details about the implementation are discussed in the next section.

2.5 Decoder Algorithms and Implementations

We implement the time information decoder and the reference frequency generator, based on WWVB signal, using PIC microcontrollers from Microchip Technology Inc. The software for broadcast time decoder and frequency reference generator was initially tested on a simulator from Microchip Technology Incorporation. The broadcast time decoder was also prototyped on a hardware demonstration board provided by Sirius Microsystems which is connected to a loop antenna tuned to 60 kHz.

As discussed in Section 2.3, a sampling frequency of 6.25 kHz was used to sample the 60 kHz signal. This is equivalent to a sampling interval of 160 μs so that we can execute up to 400 instructions between incoming samples (when run on a 10 MHz clock which delivers 2.5 MIPS). The timer needs to be loaded with a specific value, so that it generates an interrupt every 160 μs. The following calculations show how to compute the timer value. The PIC16C71 and PIC16C72 were run at 10 MHz for this application.

$$1\ instruction\ cycle = 400\ ns\ (at\ 10\ \text{MHz})$$

$$sampling\ interval = 160\ \mu\text{s}$$

$$timer\ value = \frac{160 \times 10^{-6}\ \text{s}}{400 \times 10^{-9}\ \text{s}} = 400$$

Both PIC16C71 and PIC16C72 have 8-bit timers which can hold values up to 256. The timer, however, is provided with a prescaler. Adjusting the prescaler value to n decrements the timer every n cycles. Here, we set the prescaler value to 2, so that the timer decrements every 2 cycles. Thus, the original timer value of 400 is now equivalent to 200, which fits into 8 bits.

2.5.1 Decoding of Time Standard

The decoder works in three distinct phases. Phase 1 is the power tracking phase. The decoder enters this phase on power-up. In this phase, the decoder tracks the 60 kHz signal to find two distinct power levels. The power estimator is a first-order IIR filter as described by (2.5). Figure 2.5 shows output of the estimator for an input test data sequence. When the power level changes, the power estimator tracks the power change as shown in Figure 2.5.

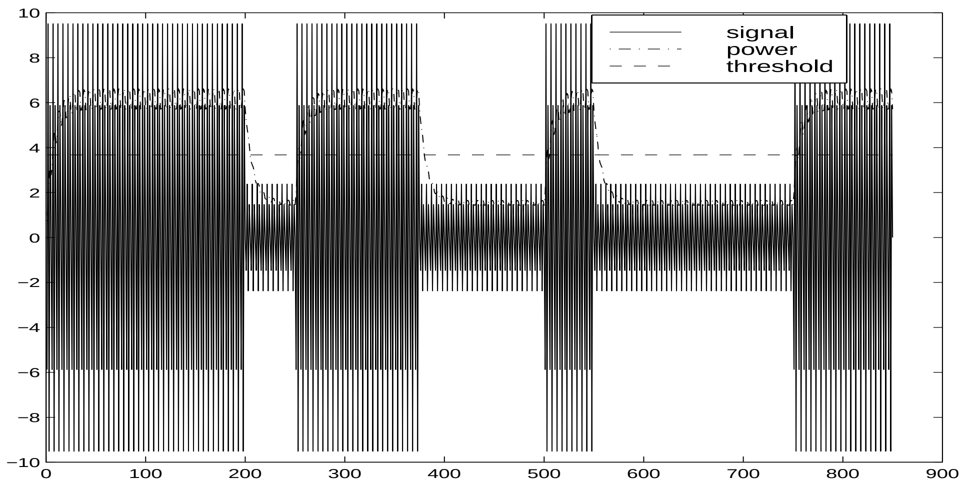


Figure 2.5: The WWVB power estimator output for synthetic data sequence.

Figure 2.6 shows the program flow in phase 1. In state A, the program waits for the timer interrupt. Upon timer interrupt, the program enters the interrupt service routine (ISR). ISR reloads the timer with the value 200 and sets the interrupt status bit. After the program returns from the ISR, the main program enters the data processing module. The power estimator estimates the current power (*CP*) according to (2.5). The current power is then checked with the previous power to determine whether the difference is within a fixed tolerance level. If the difference is less than the tolerance level, the power has stabilized. The program then checks whether the first power level has been established. If it has not been established, the program waits until the power has stabilized for certain period, and then equates the first power level *P1* to this level. It also sets *P1_detect* indicating that the first power level has been established. The program then waits for the power level to change to a second level. When the power level changes, the power estimator tracks this change. Once the power has stabilized to the new level, the second power level *P2* is

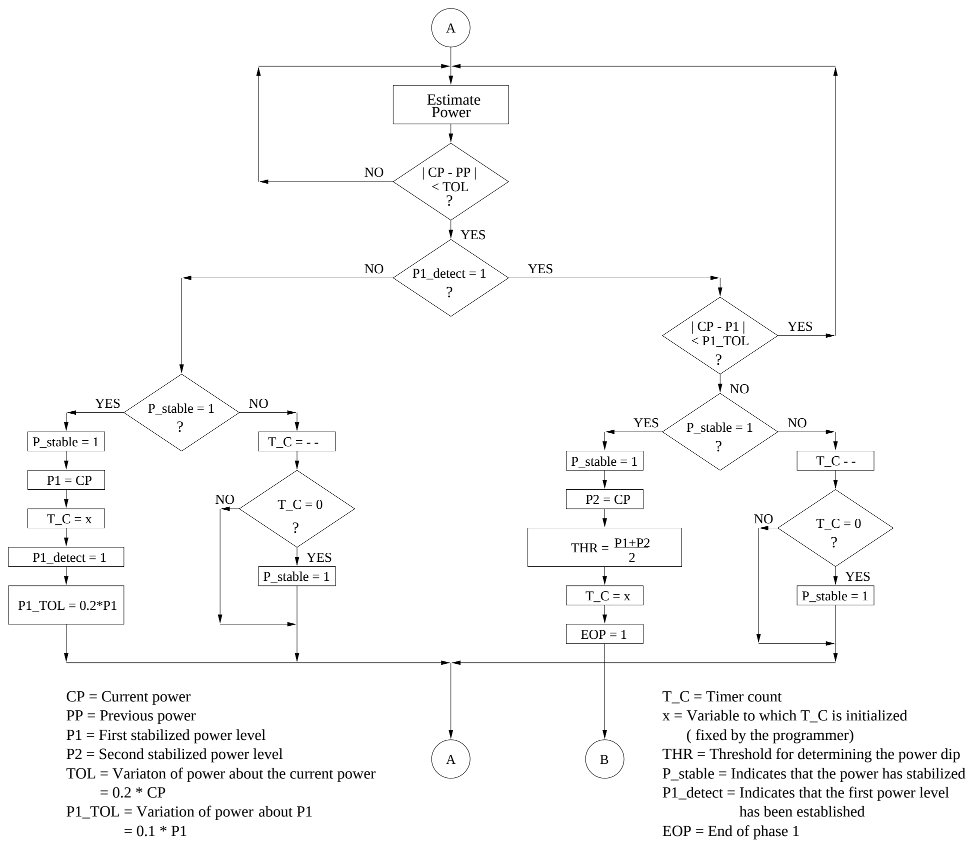

A
Estimate Power
| CP - PP | < TOL ?
NO
YES
P1_detect = 1 ?
NO
YES
| CP - P1 | < P1_TOL ?
YES
NO
P_stable = 1 ?
YES
NO
P_stable = 1
P1 = CP
T_C = x
P1_detect = 1
P1_TOL = 0.2*P1
T_C = - -
T_C = 0 ?
NO
YES
P_stable = 1
P_stable = 1 ?
YES
NO
P_stable = 1
P2 = CP
THR = (P1+P2)/2
T_C = x
EOP = 1
T_C - -
T_C = 0 ?
NO
YES
P_stable = 1
A
B
CP = Current power
PP = Previous power
P1 = First stabilized power level
P2 = Second stabilized power level
TOL = Variaton of power about the current power = 0.2 * CP
P1_TOL = Variation of power about P1 = 0.1 * P1
T_C = Timer count
x = Variable to which T_C is initialized (fixed by the programmer)
THR = Threshold for determining the power dip
P_stable = Indicates that the power has stabilized
P1_detect = Indicates that the first power level has been established
EOP = End of phase 1

Figure 2.6: Program flow for phase 1 of the WWVB time code decoder

set equal to this new level. The program then exits from phase 1.

Once the two power levels have been established, the decoder sets a threshold to determine the dip in power. The threshold is set to half the sum of the two power levels *P1* and *P2*, as shown by the dashed line in Figure 2.5. The program then enters phase 2, which is the start of minute tracking. Figure 2.1 shows that WWVB time code transmits a *position marker* of 0.8 seconds in duration every 10 seconds. It also transmits a *position marker* of 0.8 seconds in duration at the start of each minute. Thus, the start of a minute can be uniquely identified by two consecutive *position markers* of 0.8 seconds in duration. The decoder tracks the power and decodes the bits until it lands on two consecutive *position markers*. This indicates the start of minute.

BCD decoding begins in phase 3. Once the decoder enters this phase, it remains in it until power loss is detected. In phase 3, the decoder decodes the BCD time code sent on the 60 kHz carrier to obtain accurate global time information. If the decoder loses track of the signal, it keeps updating the clock by calculating seconds from the available clock cycles. Every few minutes, the decoder wakes up and tries to track the signal. If it locks onto the signal, then it resumes its normal operation.

Our technique has very low computational complexity. A maximum of three multiplications per sample is required in phase 1, while phase 2 and phase 3 require two multiplications per sample. The program does not require any divisions. We replace the square of the input sample in (2.5) by the absolute value of the input sample. By doing so, the power estimator does not track the exact power of the incoming signal. However, the power estimator still tracks the signal strength accurately. Hence, converting the square of the input sample to the absolute value does not affect the performance of the decoder. It, however, removes a multiplication operation in the data path. Thus, all the multiplications are between a variable and a constant and are optimized for length and run time (See Appendix A.2). The proposed technique is not only computationally efficient, but also requires significantly less program and data memory. Table 2.1 shows the analysis for the required memory and the number of multiply and divide operations.

2.5.2 Obtaining a Frequency Reference

We use the Friedman interpolator to extract the phase information of the 60 kHz signal. The Friedman interpolator accurately estimates the frequency and hence the period of a sinusoid in white noise. This means we can accurately determine the phase, or the distance from the previous zero crossing, of the current sample (refer to Figure 2.2). This information can be used to generate a frequency reference. Next, we describe the method for generating the frequency reference.

The PWM unit of the PIC16C72 microcontroller generates a square wave of specified period and duty cycle. The period and duty cycle counts are stored in two PWM registers. Timer 2 is an 8-bit register associated with the PWM unit. It has a prescaler and a postscaler. Depending on the prescaler value n, the timer is incremented every n cycles, until it matches the PWM period count. The timer starts counting from 0 and the PWM outputs high. When the timer value equals the duty cycle count, the PWM output is made low. When the timer value equals the period count, the PWM output is again made high and the timer is reset to zero.

Once the Friedman interpolator locks onto the signal and the frequency estimate stabilizes, the PWM unit is started at the first sample after a positive going zero crossing. At each positive going zero crossing, $\delta(n)T_s$ (refer to Figure 2.2) is calculated. Since we know the period of the sinusoid accurately, we can calculate the time that this sample occurs from the last zero crossing time. This time can then be converted into an appropriate timer count by diving it by prescaler value n times the instruction cycle speed. This timer value is compared with the actual timer 2 value. Any difference between the two is stored. The difference may not be accurate per cycle because of the phase jitter. Phase jitter occurs due to the propagation delays of the WWVB signal along the path from Ft. Collins. The phase shifts occur when sunrise and sunset occur along the path from the transmitter to receiver [13]. In our analysis, we assume that these phase changes are periodic over a period of 1 day. To remove the effects of phase shift, we keep a moving average of the difference between the calculated timer value and the actual timer value. At the end of one day, the difference is incorporated into the timer count. The whole process is then repeated again.

For a better analysis of the phase shifts, a phase recording from a stable VLF and LF radio station can be used. The phase recording information consists of monthly and weekly notices of the actual phase of the signal (measured in microseconds). NIST publishes data for WWVB, and the U.S. Naval Observatory (USNO) publishes data for a number of LF and VLF stations [13]. We can also place limits on the absolute value of the difference between the calculated timer value and the observed timer value. If the absolute value of the instantaneous difference is greater than the limit, we can disregard it because it most probably will be due to a large phase shift.

All applications involving microcontrollers or microprocessors usually run on a quartz (crystal) oscillator. Stable crystal oscillators costing \$1000 or more have a relative frequency of 10^{-13} per day. Relative frequency is defined as $\frac{f_{actual} - f_{desired}}{f_{desired}}$. Cheaper crystal oscillators (as low as \$1) have a relative frequency of approximately 10^{-4} to 10^{-6} per day. This means that

	Program Memory	**Data Memory**	**Multiplications per sample**	**Divide per sample**
Decoding the time standard	800 words	22 bytes	3	0
Generating a frequency reference	500 words	15 bytes	1	1
Combined decoder	1300 words	40 bytes	4	1

Table 2.1: Worst case analysis of memory requirements and computational power for implementing the WWVB decoder.

the oscillator frequency variations will be approximately 10^{-4} to 10^{-6} in a day. Once the phase shift error has been compensated by one of the methods described above, a frequency reference with a relative frequency of 10^{-7} to 10^{-9} can be generated.

Table 2.1 shows the approximate analysis of the program and data memory required for generating a frequency reference. It also gives the number of multiplications and divides required. The Friedman interpolator requires one multiply and a divide per sample.

2.6 Conclusions

We present a new approach to decode WWVB broadcast time information and generate an accurate frequency reference calibrated to the NIST primary standard. This reference frequency has a relative frequency of at least 10^{-4} to 10^{-6} and can be used as a secondary standard for calibrating local frequency sources. The key innovations are new, zero-buffering algorithms and microcontroller implementations to decode the WWVB time code information and to use a pulse-width modulator to generate an accurate frequency reference from the WWVB signal. We implement the time information decoder and the frequency reference generator on PIC microcontrollers. The microcontroller based decoder is at least half the cost of the existing WWVB decoders.

The decoders were tested on separate microcontrollers of the same family (PIC16C7X). A combined decoder can be implemented on a PIC16C72. The combined decoder will require 1300 words of program memory and approximately 40 bytes of data memory, as shown in Table 2.1. The PIC16C72 has 2000 words of data, 128 bytes of data memory, 1 PWM module, and 3 timers. Our combined decoder for decoding the broadcast time information and generating an accurate frequency reference is the first to be implemented on a microcontroller.

Chapter 3

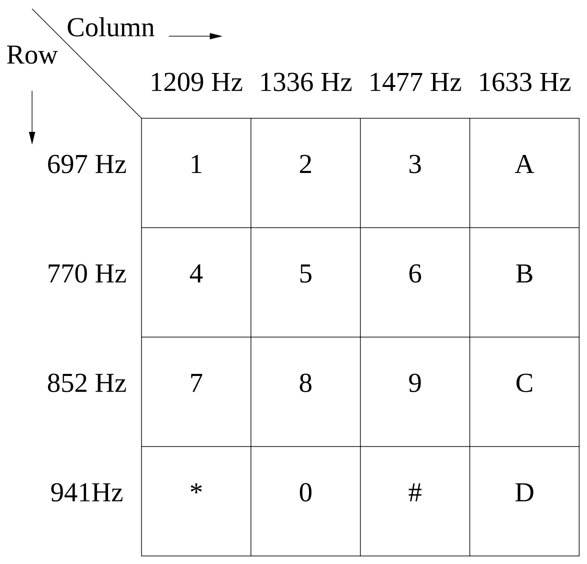
Dual Tone Multiple Frequency (DTMF) Decoding

3.1 Introduction

In the 1940's, engineers at Bell Labs figured out a better alternative than the dial pulse system for long distance dialing, interactive dialing, and reliability. Using pulses, dialing a '0' digit requires 10 pulses which last for a total of 1 second. Their research showed that tones could represent digits that a person was dialing. They came out with a signaling system in which two tones were used to represent a digit. They called this signaling system touchtone a.k.a. Dual Tone Multiple Frequency (DTMF) signaling system. The initial DTMF standard, developed by Bell Labs, supports a maximum dialing rate of 10 pulses per second for DTMF signaling. A more recent standard supports 20 digits per second.

3.1.1 Generating DTMF Signals

Dual Tone Multiple Frequency (DTMF) signaling system enables end-to-end signaling. DTMF signals are commonly used in touchtone dialing applications such as telephone dialing in public and private exchanges, military phone systems, digital answering machines, and interactive banking and reservation systems. The DTMF dialing scheme is shown in Figure 3.1. A DTMF signal is composed of two sinusoidal tones. Whenever a key is pressed, a DTMF



Row \ Column	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941Hz	*	0	#	D

Figure 3.1: Dual-Tone Multiple Frequency (DTMF) scheme for touchtone dialing. When a key is pressed, two sinusoids at the row and column frequencies are added together.

signal consisting of a row frequency tone plus a column frequency tone is transmitted. These frequencies lie in two mutually exclusive frequency groups of four frequencies each. The row group of frequencies lie below 1 kHz and column group of frequencies lie between 1 kHz and 2 kHz. The keys corresponding to the fourth column (1633 Hz) are not implemented on telephone sets, but are used in military applications.

3.1.2 Standards for DTMF Detection

The problem of decoding DTMF signals is essentially a problem of spectral estimation. To decode the DTMF signals, we need to find out the two frequencies in the signal. However, International Telecommunication Union (ITU) has given specifications for decoding DTMF signals, which are shown in

Table 3.1. The ITU requirements state that the DTMF decoder should correctly decode all DTMF signals whose high and low frequency tones have a tolerance of $\pm$ 1.5% or less. If the tolerance is above $\pm$ 3.5%, the DTMF signal should be rejected as invalid. The ITU specifications also have very stringent timing requirements and require that the DTMF detector should operate satisfactorily with a worst-case SNR of 15 dB. The Bellcore specifications, however, require that the detector should have 100% detection at 18 dB SNR and higher. Bellcore also provides digit simulation test tapes to measure the performance of the decoder against *talk-off*. False detection of speech signals as DTMF signals is called *talk-off*. Bellcore specifies the maximum number of false detects that a DTMF detector should not exceed.

• Valid DTMF tones should have a frequency tolerance within $\pm$ 1.5 %. Tones that are offset by $\pm$ 3.5 % should not be detected.
• The DTMF tones should have a minimum length of 40 ms. Tones of length less than 23 ms should be rejected.
• A DTMF signal with an interruption of less than 10 ms should not be detected as two distinct tones.
• DTMF signals separated by a pause time of at least 40 ms must be detected as two distinct digits.
• The receiver should work in a worst-case signal-to-noise ratio (SNR) of 15 dB and with an attenuation of 26 dB.
• The receiver must operate with a maximum of 8 dB normal twist and 4 dB of reverse twist. *Twist* is defined to be the difference in decibels in the amplitudes of the two fundamental tones of the DTMF signal.
• The receiver should operate in the presence of speech without incorrectly identifying the speech signal as a valid DTMF tone. Speech being detected as a DTMF signal is called as *talk-off*.

Table 3.1: ITU requirements for a DTMF decoder

Many algorithms [21] [22] [23] [24] [25] have been reported for decoding DTMF signals. However, only a few of them [24] satisfy all of the ITU requirements. In addition, almost all of them are computationally intensive and necessitate the use of a 16-32 bit digital signal processor (DSP). This chapter describes a new, low-complexity, ITU-compliant method for detecting DTMF signals using an 8-bit microcontroller. The new technique requires zero buffering. A unique feature of this decoder is the decoupling of the design of the signal processing functions from the design of the decision logic functions. Our decoder uses robust decision logic to check whether a DTMF digit satisfies the ITU requirements. We use an extremely accurate timing measurement technique to ensure that our detector performs the best timing validation checks from among the available DTMF decoders. The new method requires about 30 multiplications per sample. Our method is the first ITU-compliant method for detecting DTMF signals on a microcontroller. On a DSP processor such as the Texas Instruments TMS320C54x at 50 MIPS, our algorithm can decode 24 voice channels corresponding to a T1 time-division multiplexed telecommunications line. The new method was developed in collaboration with Crystal Semiconductor Corporation (Austin, TX). The initial development of the algorithm was done by Shawn R. McCaslin when he was at Crystal Semiconductor Corp. He is now with Cicada Semiconductor Corporation (Austin, TX).

3.2 Previous DTMF Decoders

There are many ways to decode DTMF digits. Most integrated circuit decoders use eight sharp-tuned bandpass filters to detect the presence of DTMF tone frequencies [21]. A digital post-processor measures the tone durations and provides correctly coded digital outputs. This scheme is usually implemented on integrated circuits using switched capacitor technology. They also use a 350 Hz and a 440 Hz notch filter to remove the dial tone interference from the DTMF signal. We use these filters in our decoder front-end. Some of the manufacturers of such integrated DTMF receivers are Mitel Corp., Harris Semiconductor Corp., and Teltone [26]. The other complementary approach is to detect the DTMF digits digitally. Again, several techniques for digital DTMF detection have been used. Below, we discuss some of these techniques.

The most popular method for digital DTMF detection is based on the discrete Fourier transform (DFT) which is typically implemented using the Goertzel DFT algorithm. The Goertzel algorithm formulates the computation of DFT as an infinite impulse response (IIR) filtering operation. A bank of M second-order IIR filters can be used to calculate M DFT coefficients [1]. For DTMF detection, we are interested in eight frequencies, so there will be eight parallel IIR resonators to calculate the eight values of the DFT.

The general approach of a Goertzel-based DTMF detector is to examine the energy of the received signal at the eight DTMF frequencies in order to determine whether a valid DTMF tone pair was received [27]. The Goertzel algorithm requires one real coefficient and one complex coefficient. For an N length DFT, the Goertzel algorithm only uses the complex coefficient on the Nth input sample. By taking the absolute value of the square of the Goertzel output, we can compute the power at a DTMF frequency and eliminate the use of the complex coefficient [22]. The Goertzel algorithm computes one DFT coefficient using N multiplications and 2 N additions [28].

Many modifications to the Goertzel algorithm have been applied to DTMF detection. Most methods use a fixed window size of N samples for all eight Goertzel algorithms. The choice of window length is a tradeoff between the frequency resolution and timing requirements. Most DTMF decoders use a window length that yields a window bandwidth that is too large to satisfy the ITU frequency tolerance requirements. The window length is also used to perform the timing checks. A typical window size used is 13.33 ms [28]. If a DTMF digit is valid across two or more windows a valid DTMF digit is signaled. This approach, however, does not guarantee that the DTMF digit meets the timing requirements [24].

Another DFT based approach is to use the non-uniform discrete Fourier transform (NDFT) to decode the DTMF digits [23]. The NDFT algorithm calculates the energy of a DTMF signal at the exact DTMF frequencies. The original approach, however, does not meet the ITU frequency tolerance and timing requirements. A modification of the NDFT algorithm is applied in [24]. The modified NDFT method requires about 30 multiplications per sample. It meets all the ITU and Bellcore recommendations. A US patent has been submitted for the modified NDFT method.

A different approach for DTMF detection is to use adaptive digital filtering [21]. The DTMF signal is split into a low frequency tone and high frequency tone. Two adaptive IIR filters estimate the frequency and amplitude parameters of these two tones. Another adaptive technique, an LMS-based normalized direct adaptive frequency estimation technique, has been tested for DTMF detection in [25]. Although the method is fast and easy to implement, it has a poor frequency resolution. The method is not robust in noise and does not satisfy the ITU SNR requirements.

In another linear prediction (LP) based algorithm, the DTMF signal is split into two bands. The detector analyzes each band separately and tries to adaptively estimate the roots of the second-order polynomial $1 - a_1 z^{-1} - a_2 z^{-2}$. The roots reveal the frequency content [22]. The algorithm, however, fails to meet the ITU timing requirements since it uses a fixed frame length of 13.33 ms. The algorithm needs the computation of the covariance matrix. The resolution of the LP algorithm is a function of the number of samples in the covariance matrix [22]. Thus, frequency resolution increases with an increase in the number of samples, but at the cost of a quadratic increase in the computational complexity.

An entirely different approach to DTMF detection uses subspace techniques like Multiple Signal Classification (MUSIC). MUSIC was tested for DTMF detection in [25]. Although subspace techniques could meet ITU specifications, they are computationally intensive and difficult to implement in real-time. An excellent comparison between the subspace, adaptive, and Goertzel DFT techniques for DTMF detection can be found in [25].

3.3 Proposed Method

The new, low-complexity DTMF detector is shown in Figure 3.2. A unique feature of this decoder is the separation of signal processing functions from the decision functions. The detector consists of two distinct sections– a signal processing front-end and a decision logic back-end. The signal processing

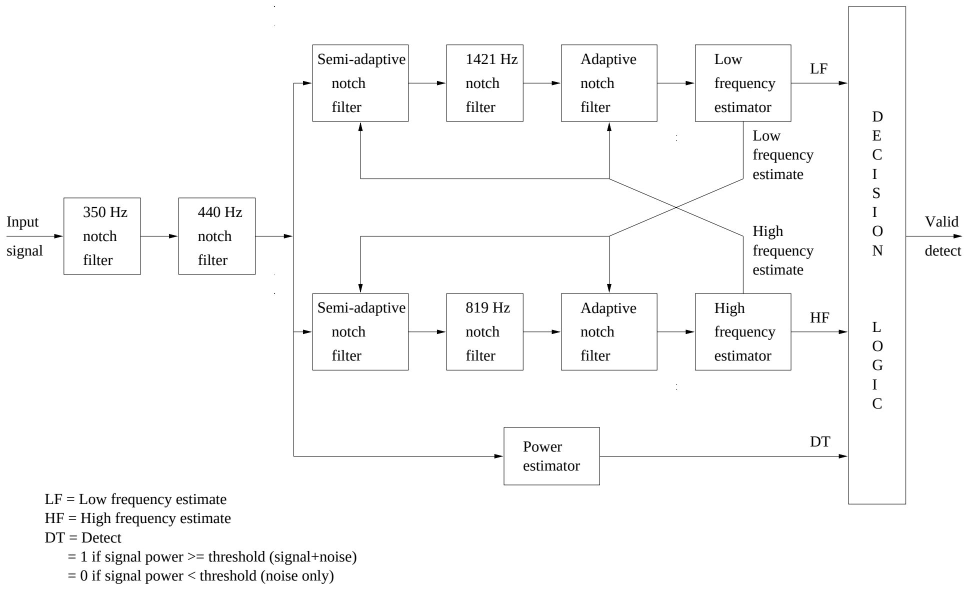


Figure 3.2: Block diagram of the DTMF detector

front-end estimates the frequencies in the incoming signal and also estimates the signal and noise power. The decision logic back-end includes the decision logic which determines whether a DTMF signal is ITU-compliant or not.

3.3.1 Signal Processing Front-end

The incoming signal on the input channel is sampled at the standard telecommunications sampling rate of 8 kHz. The sampled signal is then passed through a 350 Hz notch filter and a 440 Hz notch filters. These two notch filters remove the dial tone interference from the incoming signal. In the United States, the dial tone frequencies are standardized at 350 Hz and 440 Hz. The dial tone filtered signal is passed through three distinct data paths – the low frequency data path, the high frequency data path, and the power estimator data path.

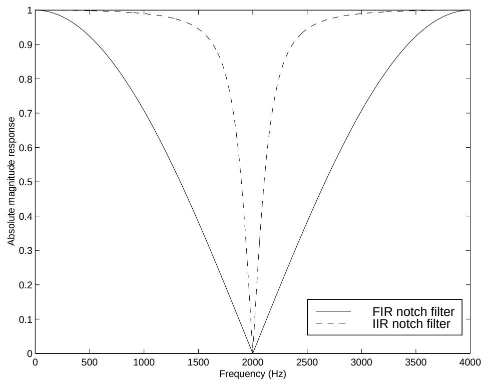


Figure 3.3: Frequency response characteristics of FIR and IIR notch filters

A notch filter is a filter having a perfect null at the desired frequency in the frequency response characteristics. A notch filter can be created by placing a pair of complex conjugate zeros on the unit circle at the desired frequency. The system function for a finite impulse response (FIR) notch filter [1] is

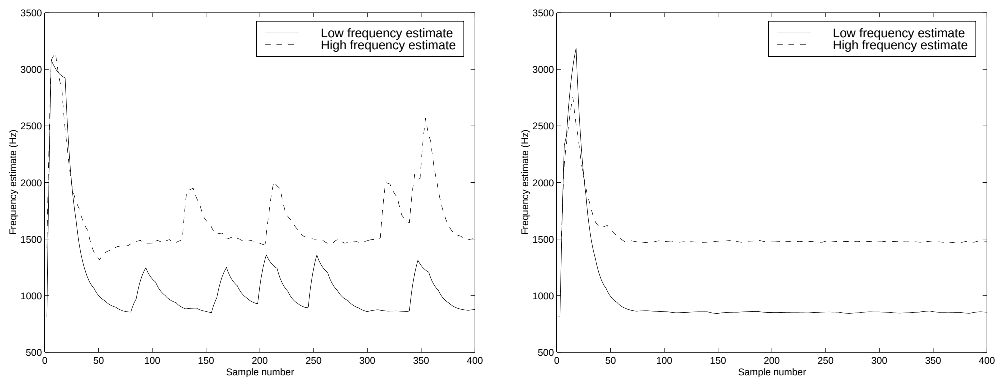
$$H(z) = b_0(1 - 2\cos w_0 z^{-1} + z^{-2}) \tag{3.1}$$

The FIR filter has a relatively large bandwidth which attenuates other frequency components near the null. On the other hand, an IIR notch filter has a very sharp notch. The transfer function for the IIR notch filter is

$$H(z) = b_0 \frac{1 - 2\cos w_0 z^{-1} + z^{-2}}{1 - 2r\cos w_0 z^{-1} + r^2 z^{-2}} \tag{3.2}$$

where $\mid r \mid < 1$. The IIR filter has poles near the zeros which introduces resonance near the null. The FIR filter has poles at the origin. Figure 3.3 shows a comparison of the bandwidths of a FIR and an IIR notch filter.

The large bandwidth of the FIR filters creates problem for the frequency estimators. We use IIR notch filters in our decoder. The location of



FIR adaptive notch filters

IIR adaptive notch filters

Figure 3.4: Frequency estimator outputs for DTMF digit '9'

the poles of the IIR filter in (3.2) can be varied by changing the parameter r. In our implementation, we have chosen the r value to be 0.866.

The adaptive notch filters in each data path do not change their coefficients to do adaptive filtering. They take the frequency estimate from the frequency estimator in the opposite data path and put a notch at that frequency. Since the notch frequency is being adaptively changed depending on the frequency estimate, we call the filter an adaptive notch filter. Once these filters converge, the adaptive filter in the low frequency data path provides a low frequency signal to the low frequency estimator. Similarly, the adaptive filter in the high frequency data path provides a high frequency signal to the frequency estimator in the same data path.

We use IIR filters for the adaptive notch filters. Although the FIR notch filters converge faster than the IIR filters, the large bandwidth of the FIR filters creates a lot of interference for the frequency estimators. Figure 3.4 shows the outputs of the frequency estimators in two cases, one using FIR adaptive notch filters and the other using IIR adaptive notch filters. In both cases, we passed a DTMF digit '9' (852 Hz + 1477 Hz) with 20 dB signal-to-noise ratio (SNR) through the detector. The FIR filters fail to converge and hence the frequency estimate does not stabilize. The IIR filters converge, although slowly, and hence the frequency estimate stabilizes to the desired frequencies.

Suppose the high frequency component of the DTMF signal lies in the lower half of the high frequency group (less than 1400 Hz). If the high frequency estimate of the frequency estimator is in the upper half of the high frequency group (above 1400 Hz), the IIR adaptive filter in the low frequency data path will take a long time to converge to the actual frequency. Due to this the detector will not be able to meet the ITU timing requirements. We eliminated this drawback by introducing a semi-adaptive notch filter in each data path.

Since the low frequency data path and the high frequency data path are essentially the same, we will explain only the low frequency data path. The semi-adaptive notch filter in the low frequency data path has a notch at one of the corner frequencies of the DTMF high frequency group. The semi-adaptive notch filter takes the frequency estimate of the high frequency estimator as a parameter. If the estimate is above 1400 Hz (approximately the center of the high frequency group), the filter sets its notch at 1209 Hz (low end of the high frequency group). If the high frequency estimate is below 1400 Hz, the semi-adaptive filter sets the notch at 1633 Hz (high end of the high frequency group). The semi-adaptive notch filter aids the adaptive notch filter in notching the high frequency component from the DTMF signal. By putting a notch in the opposite half of the high frequency group, the semi-adaptive filter increases the rate of convergence of the adaptive filter. The semi-adaptive filter in the high frequency data path does a similar job in the DTMF low frequency group.

We introduced an additional filter in each data path at the center frequency of the opposite DTMF frequency group. Thus the low frequency data path has a notch filter at the center of the DTMF high frequency group (1421 Hz). The high frequency data path has a notch filter at the center of the DTMF low frequency group (819 Hz). These filters increase the convergence rate of the adaptive notch filters. By using these filters we ensured that our decoder meets the ITU timing requirements.

The frequency estimators use a highly accurate, frequency estimation algorithm [16] based on zero crossings. The algorithm is explained briefly in Section 2.4. The spectrum of the estimated period of the incoming sinusoid contains a DC component which is equal to the actual period of the signal. To obtain this DC component, we need to lowpass filter the period estimate. In order to decrease the order of the lowpass filter, we implemented the filter in two stages. The first stage is a simple *averaging by two* filter, implemented implicitly in the frequency estimation algorithm. The second stage uses a first order IIR lowpass filter of the form given in (3.3):

$$FE(n) = \alpha \times FE(n-1) + (1-\alpha) \times FE(n). \tag{3.3}$$

where FE stands for *frequency estimate*. In our implementation, we have used a value of 0.875 for α. Experimentally, this value of α gives an accurate estimate of the period and hence the frequency of the sinusoid.

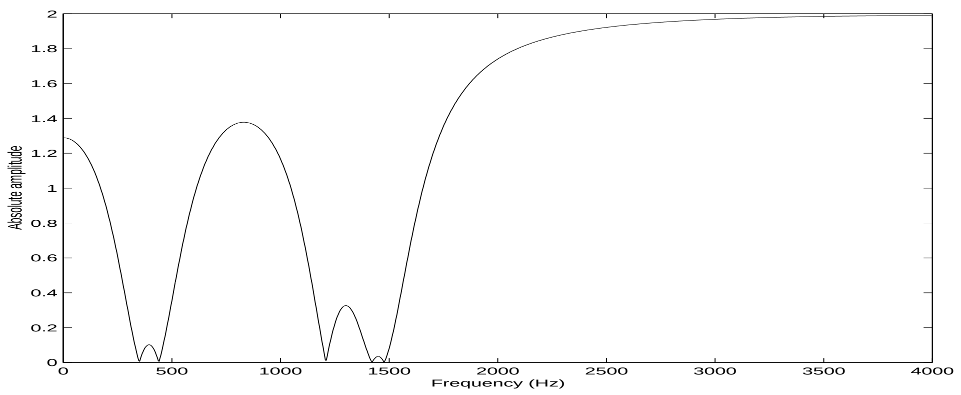


Figure 3.5: Transfer function of the low frequency data path

Figure 3.5 shows the transfer function of the low frequency data path after the frequency estimates have stabilized. The decoder was passed the DTMF digit '8' which consists of 852 Hz and 1477 Hz tones. There are notches at 350 Hz and 440 Hz and three more at 1209 Hz (semi-adaptive notch filter), 1421 Hz and 1477 Hz (adaptive notch filter). As can be seen from the transfer function, the DTMF high frequency group is severely attenuated as compared to the DTMF low frequency group. This emphasizes the low frequency signal being fed to the low frequency estimator.

The power estimator tracks the power of the signal component and the power of the noise component of the incoming noisy signal. It is difficult to separate the signal component from the noise component. However, since the DTMF signal occurs in bursts, the signal power can be tracked as a short term average of the power. The power estimator uses a first-order IIR filter to track the signal power.

$$P(n) = \alpha \times P(n-1) + (1-\alpha) \times \mid s(n) \mid \tag{3.4}$$

where $P(n)$ is the current power and $s(n)$ is the current signal sample. In

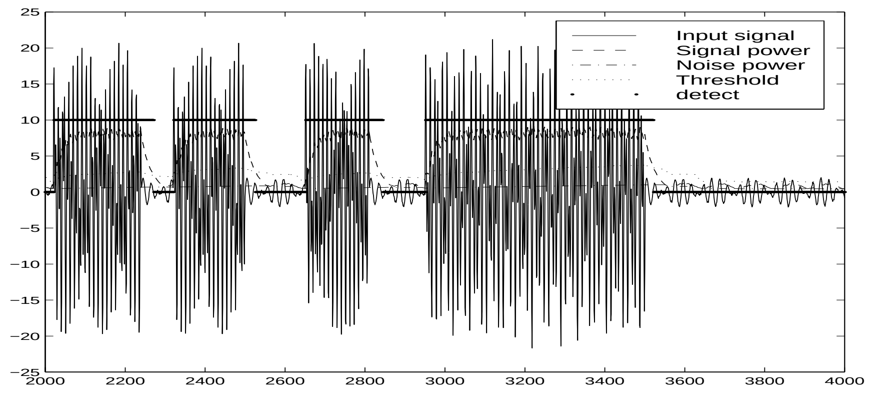


Figure 3.6: The power estimator in the DTMF detector

our implementation, we have used a value of 0.95 for α. The noise power is computed as the long-term average of the power.

The estimator uses a variable threshold to detect the presence of the signal component. The estimator weights the signal power and the noise power to set the threshold for each iteration [29]. Figure 3.6 shows the working of the power estimator. If the signal power is more than the threshold, *detect* is made high to indicate the presence of a signal component. If the signal power is less than the threshold, the *detect* signal is low indicating the presence of noise. Since we adaptively set the threshold at each iteration, our power estimator is extremely robust for detecting the presence of a signal component in a noisy signal. By including this ability of detecting the presence of a signal component in a noisy signal, we ensured that our detector has an outstanding *talk-off* performance as Section 3.5 explains.

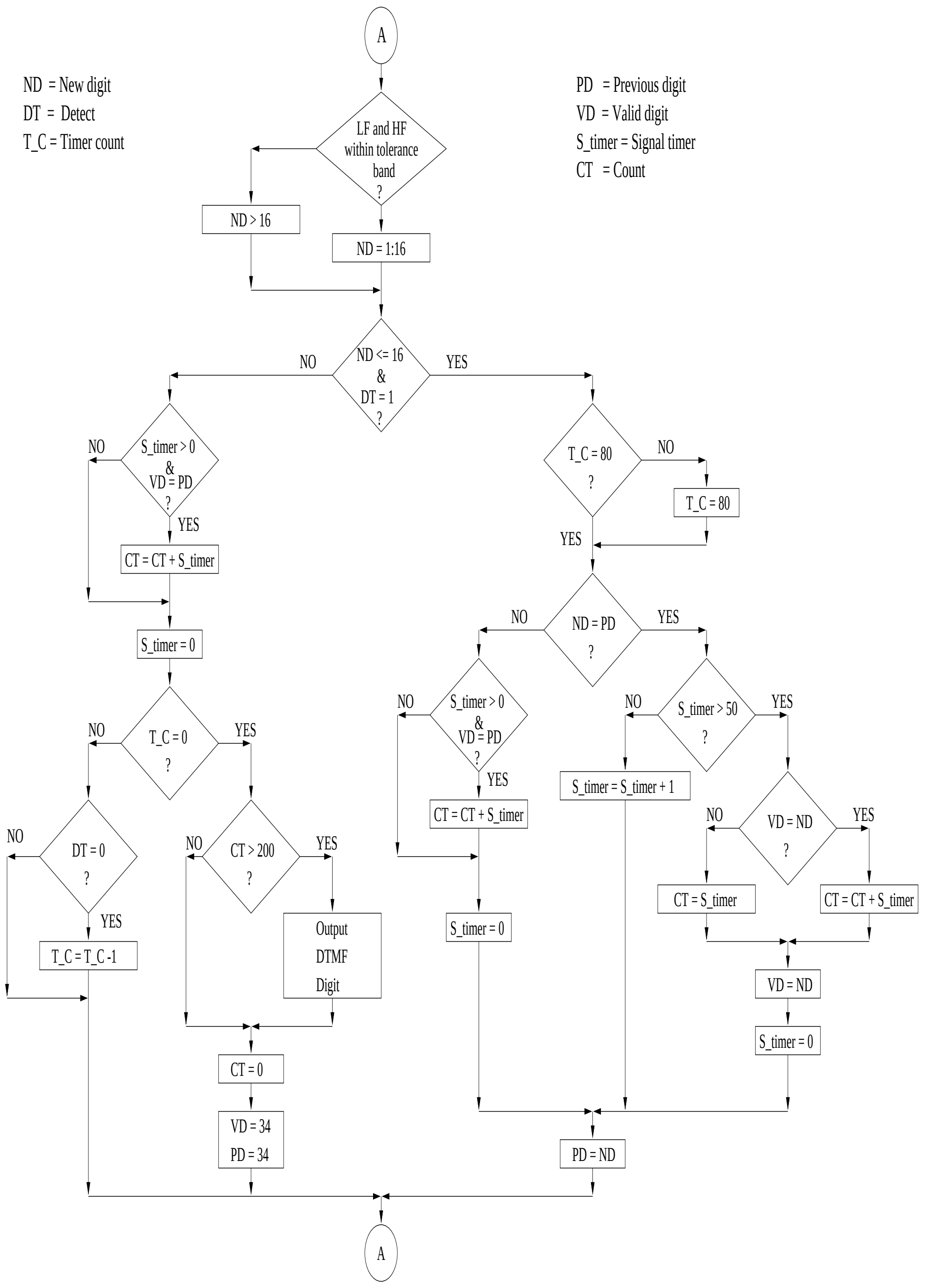


Figure 3.7: Decision logic for DTMF detection

3.3.2 Decision Logic Back-end

The decision logic back-end performs decision checks to ensure that the detected DTMF digit satisfies all the ITU requirements. We use a robust decision logic for this purpose. Our key contribution in the decision logic is the use of a counter which keeps track of the DTMF digit timing. Most DTMF decoders use a fixed window size (such as 13.33 ms) [22] to perform the timing check. If a DTMF digit is valid for 2 or more windows lengths, the DTMF digit is assumed to have satisfied the timing requirements. The time resolution used here is very large and may lead to erroneous results. In our decision logic, we start a counter as soon as we detect a DTMF digit. The counter is updated every cycle if the same DTMF digit is detected. This approach ensures that our timing estimate uses the smallest resolution available (125 μs). The decision logic performs the best timing validation checks from among the available DTMF decoders.

The decision logic takes the low frequency estimate (*LF*), the high frequency estimate (*HF*), and the *detect* signal as its inputs as shown in Figure 3.2. Figure 3.7 shows the control logic that it uses to determine whether or not a DTMF digit is valid. *DT* is the *detect* signal coming from the power estimator. *ND* represents the most recently detected DTMF digit. *PD* stores the previously detected DTMF digit. If *ND* and *PD* are same for a pre-determined amount of time, we can conclude that the frequency estimates have stabilized. *VD* stores the detected DTMF digit after the frequency estimates have stabilized. *T_C* is the timer count. It is used to determine if the tone interruption lasted more than 10 ms which indicates the presence of two DTMF digits. *T_C* is initialized to 80 samples since it corresponds to 10 ms with a sampling frequency of 8 kHz. *CT* holds the actual time for which a DTMF digit is present. It is updated every time *S_timer* counts from 0 to 50.

Initially, the decision logic checks whether the low frequency estimates *LF* and *HF* are within the DTMF tolerance band. If *LF* and *HF* satisfy the frequency tolerance requirements, *ND* is assigned a DTMF digit between 1 and 16, corresponding to the detected frequencies. If *detect* is high, *T_C* is reinitialized and the previously detected DTMF digit (*PD*) is compared with the recently detected DTMF digit (*ND*). If *PD* does not equal *ND*, the time count *CT* is updated and *S_timer* is reset to 0. If *PD* is equal to *ND*, the time count *CT* is updated and *VD* is equated to *ND*. The logic assigns the most recently detected DTMF digit *ND* to *PD* before exiting.

Whenever there is a pause greater than 10 ms after a DTMF tone burst, the logic checks the actual time (*CT*) for which the DTMF digit was detected. If the DTMF digit meets the ITU requirements (see Table 3.1), then a valid DTMF digit detection is signaled.

3.4 Design and Implementation

Existing DTMF receivers are computationally intensive. The Goertzel-based DFT algorithm requires a real coefficient and a complex coefficient. As discussed previously, we can eliminate the complex coefficient by taking the absolute value of the energy. Still, the Goertzel algorithm requires N real multiplications, $2N$ real additions and 3 words of memory [24]. The DFT-based DTMF detector uses 8 banks of Goertzel filters. Thus, the detector requires

$8N$ real multiplications, $16N$ real additions and 24 words of memory only for the Goertzel filters. The processing power required to decode DTMF signals on one channel is of the order of 1 MIPS [24]. Our DTMF detector is efficient from the point of view of memory and complexity. Below we present an analysis of the memory requirement and the computational complexity of our detector. The analysis is based on hand coding some of the key routines of the DTMF detector such as the multiplications for the notch filters and cosine value computation on a PIC microcontroller. We present a separate analysis for microcontrollers, primarily because multiplication is an expensive operation on microcontrollers. We also present an analysis of the memory and processing power required for a DSP processor implementation.

Each notch filter requires 3 multiplications, except for the two fully adaptive notch filters which require 4 multiplications. We have 8 notch filters in our decoder. Each frequency estimator requires a divide operation which can be converted to a multiplication operation in the following manner. We need to calculate the fraction $\left(\frac{\mathrm{a}}{\mathrm{b}}\right)$ in the frequency estimators where $b \geq a$. Both a and b are positive numbers. Hence, we use a 0.8 binary fractional format for this division. The binary 0.8 format puts the decimal point to the left of the most significant bit (MSB). Thus, all numbers range between 0 and 1. The division $\left(\frac{\mathrm{a}}{\mathrm{b}}\right)$ can be expressed as $a \times \left(\frac{1}{\mathrm{b}}\right)$. In 0.8 binary format, 1 is represented as the number 255 (hex 0×ff). We propose to implement the operation $\left(\frac{1}{\mathrm{b}}\right)$, which is equivalent to $\left(\frac{255}{\mathrm{b}}\right)$ in binary 0.8 format, as a lookup table. Once we have a lookup table, the binary number $\left(\frac{255}{\mathrm{b}}\right)$ can be obtained and multiplied by a to implement the division operation $\left(\frac{\mathrm{a}}{\mathrm{b}}\right)$.

Since we implement the division operations as multiply operations, each frequency estimator requires only one multiply operation. The power estimator and the lowpass filters for filtering the low frequency estimate and the high frequency estimate as given in (3.3) require about 4 multiplications each. The algorithm requires a total of about 35 multiplications per sample. However, some of the multiplications can be implemented as shifts (e.g. the multiplication by α (0.875) in (3.4)). On average, we estimate 30 multiplications per sample.

Besides the multiplication operations, each notch filter requires about 15 microcontroller instructions. We assume a RISC microcontroller having single cycle instruction execution, bit manipulation instructions such as *bit set*, *bit clear*, and bit control instruction such as *branch on bit change*. Whenever there is a zero crossing, each frequency estimator requires about 20 microcontroller instructions. Implementing two lowpass filters as given in (3.4) requires 16 microcontroller instructions. The decision logic requires about 200 microcontroller instructions. Based on the implementation of some of key kernels required for the DTMF detector, we estimate that an additional 400 instructions per sample (without taking the multiply operations into account) will be required.

Our analysis shows that the DTMF detector requires a maximum of 50 bytes of data memory. The detector requires the computation of a cosine value which we implement as a lookup table. We propose to implement the lookup tables for the divide operation and the cosine value computation in program memory. Accordingly, the proposed DTMF detector requires about

2000 words of program memory.

The DTMF detector can be implemented on a high-end microcontroller such as the one from PIC17CXX series of microcontrollers from Microchip Technology Incorporation. PIC17C43 has 4000 words of erasable program memory and 454 bytes of data memory. The instruction execution speed is 121 ns at 33 MHz or 160 ns at 25 MHz. PIC17C43 has a hardware multiplier which computes an 8-by-8 signed multiplication operation in about 1 μs [12]. We can use PIC17C43 for single-channel DTMF detection. The cost of PIC17C43 is $10 in volumes of 100 units.

For the DSP processor implementation, we assume a 16-bit fixed point processor with single cycle multiply-accumulate capability and having simultaneous instruction execution and data access ability (such as Texas Instruments TMS320C54x digital signal processors). Since the DSP processor can perform the multiplication operation in a single cycle, we do not consider multiplication operations separately as we did for microcontroller implementation. The detector requires about 1.6 MIPS per sample. We propose to implement the lookup tables for division and cosine value computation in data memory. Sinusoid lookup tables come with many DSP processors. The remaining program requires only 50 bytes of data memory. The estimated data memory and program memory size is 600 words and 1000 words respectively. With a fixed point processor such as the TMS320C54x running at 50 MIPS, our decoder can decode 24 channels corresponding to a single T1 line.

3.5 Experimental Results

The ITU recommendations for a DTMF detector are given in Table 3.1. We tested our detector performance for all the requirements using floating-point precision. Below, we explain the tests performed and the detector performance.

The ITU requirements state that the DTMF detector should detect all DTMF digits having a frequency tolerance of ±1.5% or less. All digits having a frequency tolerance of ±3.5% or greater should be rejected. If the frequency tolerance is between ±1.5% and ±3.5%, the digit may be detected as valid or invalid. For each digit the high frequency tolerance was held at -4% and the low frequency tolerance was varied from -4% to +4% in steps of 0.5%. For each combination of frequency tolerance, the DTMF detector performance was tested. Next, the high frequency tolerance was increased by 0.5% to -3.5% and the whole process was repeated. The test was continued until the high frequency tolerance was +4%. Our detector detected all digits having a frequency tolerance of ±1.5% or less. The detector showed 100% rejection for digits with a frequency tolerance of ±3.5%.

According to the ITU recommendations, a valid DTMF digit should have a minimum tone length of 40 ms. All digits having a tone length of 23 ms or less should be considered invalid. If the tone length is between 23 ms and 40 ms, it is up to the detector to consider the digit as valid or invalid. A DTMF signal interrupted for less than 10 ms should not be detected as two DTMF digits. Five tests were carried out for timing requirements. *Guard time* is the minimum tone length of the DTMF digits that the detector accepts as

	0 dBm	**-20 dBm**	**15 dB SNR**	**40 dB SNR**
Minimum accepted tone length	36 ms	36 ms	36 ms	36 ms
Maximum rejected tone length	35 ms	35 ms	33 ms	33 ms
Minimum tone interruption causing 2 detects	15 ms	15 ms	15 ms	15 ms
Maximum pause time not causing 2 detects	14 ms	14 ms	14 ms	14 ms

Table 3.2: Timing test results for our DTMF decoder

valid. The proposed DTMF detector has a guard time of 35.6 ms. Table 3.2 shows the results of the other timing tests. DTMF digit '1' was used for all the timing tests.

Twist is defined as the difference in decibels in the amplitudes of the two fundamental tones of the DTMF signal. Usually, due to the lowpass characteristics of the telephone channel, the high frequency tone will be received at an amplitude lower than the low frequency tone. This is called *normal twist*. When the low frequency tone is received at an amplitude lower than the high frequency tone, the condition is called *reverse twist*. ITU recommendations state that the decoder must operate properly with a maximum of 8 dB normal twist and 4 dB reverse twist. All DTMF digits were tested by varying the twist from -4 dB to +8 dB in steps of 0.1 dB. Table 3.3 shows results of the test.

ITU requires that the DTMF detector should should detect DTMF digits with a worst-case signal-to-noise ratio (SNR) of 15 dB. Bellcore, requires

Digits	**Detector accuracy from -4dB to +7.8 dB twist**	**Detector accuracy at +8 dB twist**
1	100%	92%
2	100%	100%
3	100%	93%
4	100%	70%
5	100%	95%
6	100%	100%
7	100%	98%
8	100%	100%
9	100%	100%
0	100%	100%
*	100%	53%
#	100%	100%
A	100%	61%
B	100%	93%
C	100%	88%
D	100%	61%

Table 3.3: Twist test results for our DTMF decoder

the detector to have 100% detection for SNR of 18 dB or more. Our detector showed 100% detection down to 16 dB. After that the percentage detection starts decreasing. However, with Gaussian noise, 100% detect reliability cannot be expected. Nonetheless, the error probability for the detector will be very small (10^{-4} to 10^{-5}).

Bellcore power level test measures the dynamic range and sensitivity of the DTMF decoder. The test starts with the DTMF tone level at 0 dBm (dBm is defined as $10 \log_{10} \frac{signal\ power}{1 \times 10^{-3}}$, where the denominator is the reference power). Every iteration the DTMF power level is decreased by 1 dBm.

Digits	**Allowed Detects**	**Our Detector Performance**
0-9	333	2
0-9,*,#	500	2
0-9,*,#,A-D	600	2

Table 3.4: Talk-off test results for our DTMF decoder

The test is stopped just before the DTMF detector fails for the first time. This dBm level is defined as the sensitivity of the detector. The Bellcore requirements state that the decoder should have a minimum sensitivity of -25 dBm. Our detector showed a sensitivity of -36 dBm.

The Bellcore decode test measures the detector performance for each digit. For each DTMF digit, the test determines the minimum length for which the detector has 100% detection. We passed 10 pulses of each digit to the detector, starting with a 50 ms ON time and 50 ms OFF time. The ON time was decreased by 1 ms till 100% detection was observed. The proposed detector has 100% detection for each digit up to 38 ms ON time.

Speech being detected as a DTMF signal is called as *talk-off*. Bellcore provides audio test tapes containing 1 million calls to a central office, including 50,000 speech samples to test the DTMF receiver performance against *talk-off*. The maximum allowable false detects are specified by Bellcore. Our detector showed excellent performance against *talk-off*. Table 3.4 shows the detector performance.

3.6 Conclusions and Future Work

We present a new, low-complexity algorithm for DTMF detection. Our algorithm is one of the two known algorithms that satisfies all the ITU recommendations. A unique feature of this detector is the decoupling of the design of the signal processing functions from the design of the decision logic functions. In the signal processing front-end, key innovations include the use of adaptive notch filtering and accurate frequency estimation techniques for DTMF detection. Another key contribution in the signal processing front-end is the use of semi-adaptive and fully adaptive notch filters to increase the convergence rate of the combined notch filter. Our contribution in the decision logic includes a new, accurate timing measurement technique that uses the smallest available resolution and a new sophisticated decision logic to check whether a DTMF digit satisfies the ITU requirements. The proposed detector has excellent *talk-off* performance. The proposed method is the first ITU-compliant method that can be implemented on an 8-bit microcontroller.

The detector requires only 50 words of data memory. It requires about 2000 words of program memory for a microcontroller implementation and about 600 words of program memory for a DSP processor implementation. A single channel DTMF decoder can be implemented on a high-end microcontroller such as PIC17C43 from Microchip Technology Incorporation. On a DSP processors, our technique requires 1.6 MIPS per channel. On TMS320C54X running at 50 MIPS, we predict that the new method can detect 24 simultaneous telephone channels corresponding to a T1 time-division multiplexed telecommunications line.

In the future, we intend to test the algorithm at a decimated rate of 4 kHz. If the algorithm satisfies all the ITU recommendations at the decimated rate of 4 kHz, we plan to implement the algorithm on a cheaper microcontroller (not having a hardware multiplier) like the PIC16C63 mid-range microcontroller from Microchip Technology Incorporation.

Appendix A

Multiplication Routines for PIC Microcontroller

Appendix A.1 shows a PIC16C71 microcontroller code for multiplication between two 8-bit variables. The 16-bit result is stored in 2 bytes. Appendix A.2 shows a modification of the PIC16C71 microcontroller code for multiplication between an 8-bit variable and an 8-bit constant. The routine has been optimized for code length and run time. A division routine for dividing a 16-bit number by an 8-bit number is shown in Appendix B. The example codes in Appendix A.1 and Appendix B are taken from CD-ROM technical library provided my Microchip Technology Inc. The codes can also be found at Microchip website (http://www.microchip.com/10/Appnote/Listing/index.htm).

A.1 Multiplication between two Variables

```
; LIST    P = 16C71,  F = INHX8M, n = 66
;
;*************************************************
; 8x8 Software Multiplier
; ( Fast Version : Straight Line Code )
;*************************************************
;
; The 16 bit result is stored in 2 bytes
```

```
;
; Before calling the subroutine " mpy ", the multiplier should
; be loaded in location " mulplr ", and the multiplicand in
; "mulcnd". The 16 bit result is stored in locations
; H_byte & L_byte.
;
; Performance :
;                      Program Memory  :  35 locations
;                      # of cycles     :  37
;                      Scratch RAM     :   0 locations
;
;
;       Program:          MULT8x8F.ASM
;       Revision Date:    12-12-95
;
;       Compatibility with MPASMWIN 1.30
;
; This routine is optimized for speed efficiency
; ( straight line code )
;***************************************************
;
Same equ 1
;
;
; include         "mpreg.h"
```

```
;
; Define a macro for adding & right shifting
;
mult    MACRO   bit             ; Begin macro
btfsc   mulplr,bit
addwf   H_byte,Same
rrf     H_byte,Same
rrf     L_byte,Same
ENDM                    ; End of macro
;
; *****************************   Begin Multiplier Routine
; Begin macro
;
mpy_F   MACRO mulplr,mulcnd,H_byte,L_byte
clrf    H_byte
clrf    L_byte
movf    mulcnd,w        ; move the multiplicand to W reg.
bcf     STATUS,C ; Clear the carry bit in the status Reg.
mult    0
mult    1
mult    2
mult    3
mult    4
mult    5
mult    6
mult    7
;
ENDM
;
; END
;
;************************************************
```

A.2 Multiplication between a Variable and a Constant

```
; LIST    P = 16C71,  F = INHX8M, n = 66
;
;*********************************************
; 8x8 Software Multiplier
; (Fast Optimized Version : Straight Line Code )
;*********************************************
;
;    The 16 bit result is stored in 2 bytes
;
;  Before calling the subroutine " mpy ", the multiplier should
;  be loaded in location " mulplr ", and the multiplicand in
;  "mulcnd". The 16 bit result is stored in locations
;  H_byte & L_byte.
;
;       Performance :
;                      Program Memory  :  14 locations
;                      # of cycles     :  16
;                      Scratch RAM     :   0 locations
;
;
;       Program:          MULT_OPT.ASM
;       Revision Date:    2-10-98
;       Author:           Amey Deosthali
;       Copyright:        The University of Texas at Austin
```

```
;
; Compatibility with MPASMWIN 1.30
;
; This routine is optimized for speed
; efficiency ( straight line code )
;*********************************************
;
Same equ 1
;
;
;
; ********************************************
; Begin Multiplier Routine
mpy_48  MACRO mulcnd ;Begin macro
clrf    H_byte
clrf    L_byte
movf    mulcnd,W ; move the multiplicand to W reg.
bcf     STATUS,C ; Clear the carry bit in the status Reg.
;
; Multiply by 0.875. Using fixed point arithmetic
; (0.8 format) for fractional numbers, 0.875 maps the hex value
; of 0xE0.=> 0.875/LSB = 0.875/(1/256) = 224 => 0xE0
;
addwf H_byte,Same ; bit 4
rrf H_byte,Same
rrf  L_byte,Same
;
addwf H_byte,Same ; bit 5
rrf H_byte,Same
rrf L_byte,Same
;
rrf H_byte,Same ; bit 6
rrf L_byte,Same
;
rrf H_byte,Same ; bit 7
rrf L_byte,Same
;
ENDM
******************************************************
```

Appendix B

Division Routine for PIC Microcontroller

```
; 16/8 Bit Signed Fixed Point Divide 16/8 -> 16.08
;
; Input: 16 bit signed fixed point dividend in AARG+B0, AARG+B1
; 8 bit signed fixed point divisor in BARG+B0
;
; Use: CALL FXD1608S
;
; Output: 16 bit signed fixed point quotient in AARG+B0, AARG+B1
; 8 bit signed fixed point remainder in REM+B0
; Result: AARG, REM < AARG / BARG
;
; Max Timing: 10+163+3 = 176 clks A > 0, B > 0
; 11+163+11 = 185 clks A > 0, B < 0
; 14+163+11 = 188 clks A < 0, B > 0
; 15+163+3 = 181 clks A < 0, B < 0
;
; Min Timing: 10+163+3 = 176 clks A > 0, B > 0
; 11+163+11 = 185 clks A > 0, B < 0
; 14+163+11 = 188 clks A < 0, B > 0
; 15+163+3 = 181 clks A < 0, B < 0
;
; PM: 15+42+10 = 67 DM: 6
;
;****************************************************
;
list r=dec,x=on,t=off,p=16C71
include <PIC16.INC>
;
; Define divide register variables

ACC  equ 0x0D  ; most significant byte of
; contiguous 4 byte accumulator
SIGN  equ 0x13  ; save location for sign in MSB
TEMP  equ 0x19  ; temporary storage

; Define binary operation arguments
;
AARG  equ 0x0D  ; most significant byte of argument A
BARG  equ 0x16  ; most significant byte of argument B
REM  equ 0x11  ; most significant byte of remainder
LOOPCOUNT  equ 0x14  ; loop counter

;  Note:  ( AARG+B0, AARG+B1 ) and ( ACC+B0, ACC+B1)
;  reference the same storage locations, and similarly for
;  ( REM+B0, REM+B1 ) and ( ACC+B4, ACC+B5 )
;*****************************************************
;
; 16/08 BIT Division Macros
SDIV1608L  macro
;
; Max Timing: 3+5+2+5*11+10+10+6*11+10+2 = 163 clks
; Min Timing: 3+5+2+5*11+10+10+6*11+10+2 = 163 clks
; PM: 42 DM: 5
;
MOVF  BARG+B0,W
SUBWF REM+B0
RLF  ACC+B0
RLF  ACC+B0,W
RLF  REM+B0
MOVF  BARG+B0,W
ADDWF REM+B0
RLF  ACC+B0
MOVLW 6
MOVWF LOOPCOUNT
;
LOOPS1608A
;
RLF  ACC+B0,W
RLF  REM+B0
```

```
MOVF  BARG+B0,W
BTFSC ACC+B0,LSB
SUBWF REM+B0
BTFSS ACC+B0,LSB
ADDWF REM+B0
RLF  ACC+B0
DECFSZ LOOPCOUNT
GOTO  LOOPS1608A
RLF  ACC+B1,W
RLF  REM+B0
MOVF  BARG+B0,W
BTFSC ACC+B0,LSB
SUBWF REM+B0
BTFSS   ACC+B0,LSB
ADDWF REM+B0
RLF  ACC+B1
MOVLW 7
MOVWF LOOPCOUNT
;
LOOPS1608B
;
RLF  ACC+B1,W
RLF  REM+B0
MOVF  BARG+B0,W
BTFSC ACC+B1,LSB
```

```
SUBWF   REM+B0

BTFSS ACC+B1,LSB

ADDWF REM+B0

RLF  ACC+B1

DECFSZ   LOOPCOUNT

GOTO  LOOPS1608B

BTFSS ACC+B1,LSB

ADDWF REM+B0

;

ENDM

;

;*******************************************************

;

FXD1608S

MOVF  AARG+B0,W

XORWF BARG+B0,W

MOVWF SIGN

BTFSS BARG+B0,MSB  ; if MSB set go & negate BARG

GOTO  CA1608S

COMF  BARG+B0

INCF  BARG+B0

;

CA1608S

;

BTFSS AARG+B0,MSB  ; if MSB set go & negate ACCa
```

```
GOTO  C1608S
COMF  AARG+B1
INCF  AARG+B1
BTFSC _Z
DECF  AARG+B0
COMF  AARG+B0
;
C1608S
;
CLRF  REM+B0
;
SDIV1608L
;
BTFSS SIGN,MSB  ; negate (ACCc,ACCd)
RETLW 0x00
COMF  AARG+B1
INCF  AARG+B1
BTFSC _Z
DECF  AARG+B0
COMF  AARG+B0
COMF  REM+B0
INCF  REM+B0
RETLW 0x00

;*******************************************************
```

BIBLIOGRAPHY

[1] J. G. Proakis and D.G.Manolakis, *Digital Signal Processing Principles, Algorithms, and Applications.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[2] J. Bier, "The evolution of DSP processors." Berkeley Design Technology Inc., presentation to the U.C. Berkeley CS 152 class, Nov. 1997. http://www.bdti.com/articles/evolution.htm.

[3] T. Engibous. "Focus on Digital Signal Processing solutions," Bernstein Strategic Decisions Conference, June 1997. Texas Instruments, http://www.ti.com/corp/docs/library/bern/bern.htm.

[4] C. Nguyen, "Designing low voltage microcontrollers into portable products," in *Wescon Conf. Record Proc.*, (San Jose, CA), pp. 222–227, Nov. 1997.

[5] B. L. Evans, "*EE382C Embedded Software Systems.*" Spring 1998 course, The University of Texas at Austin, http://www.ece.utexas.edu/∼bevans/courses/ee382c/.

[6] P. K. Mishra, S. Paul, S. Venkataraman, and R. Gupta, "Hardware/software co-design of a high-end mixed signal microcontroller," in *Proc. IEEE Int. Conf. VLSI Design*, (San Jose, CA), pp. 91–95, Jan. 1998.

[7] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features.* Berkeley, CA: Berkeley Design Technology Inc., 1994.

[8] D. Clark, "New era for digital signal processors," *IEEE Computer*, vol. 30, pp. 10–11, Jan. 1998.

[9] R. Hersch, "Microcontroller-faq/primer." ftp://rtfm.mit.edu/pub/usenet-by-hierarchy/news/answers/microcontroller-faq/primer.

[10] B. Shackleford, M. Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, A. Inoue, and H. Yasuura, "Embedded system cost optimization via data path width adjustment," *IEEE Trans. Inform. and Syst.*, vol. E80-D, pp. 974–981, Oct. 1997.

[11] C. Nguyen, "Fixed-point math in time critical applications," in *Wescon Conf. Record Proc.*, (Los Angeles, CA), pp. 587–593, Oct. 1996.

[12] "Microchip PICmicro devices, Microchip Techonology Incorporation." http://www.microchip.com/10/Datasheet/PICmicros/index.htm.

[13] G. Kamas and M. A. Lombardi, *NIST Time and Frequency Users Manual.* Boulder, CO: National Institute of Science and Technology, 1990.

[14] R. E. Beehler and M. A. Lombardi, *NIST Time and Frequency Services.* Boulder, CO: National Institute of Science and Technology, 1991.

[15] T. E. Parker and J. Levine, "Impact of New High Stability Frequency Standards on the Performance of the NIST AT1 Time Scale," *IEEE Trans-*

actions on Ultrasonics, Ferroelectrics, and Frequency Control, vol. 44, pp. 1239 – 1244, Nov 1997.

[16] V. Friedman, "A zero crossing algorithm for the estimation of the frequency of a single sinusoid in white noise," *IEEE Trans. Signal Processing*, vol. 42, pp. 1565 – 1569, June 1994.

[17] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing.* Englewood Cliffs, NJ: Prentice Hall, 1989.

[18] P. E. Wellstead, "Aliasing in system identification," *Int. Journal of Control*, vol. 22, pp. 363 – 375, Sep. 1975.

[19] Z. Dusan, "Mean power estimation with a recursive filter," *IEEE Trans. Aerospace Electronic Systems*, vol. 13, pp. 281 – 289, May 1977.

[20] T. N. Osterdock and J. A. Kusters, "Using a new GPS frequency reference in frequency calibration operations," in *Proc. Annual Int. Frequency Control Symp.*, pp. 33–39, June 1993.

[21] S. Park and D. M. Funderburk, "DTMF detection having sample rate decimation and adaptive tone detection." United States Patent, Feb. 1995. Patent Number: 5,392,348.

[22] S. L. Gay, J. Hartung, and G. L. Smith, "Algorithms for multi-channel DTMF detection for the WEDSP32 family," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Processing*, pp. 1134–1137, May 1989.

[23] S. Bagchi and S. K. Mitra, “An efficient algorithm for DTMF decoding using the subband NDFT,” in *Proc. IEEE Int. Symp. Circ. Syst.*, pp. 1936–1939, May 1995.

[24] M. D. Felder, J. C. Mason, and B. L. Evans, “Efficient dual-tone multi-frequency detection using the non-uniform discrete Fourier transform,” *IEEE Signal Processing Letters*, Jul. 1998. to appear.

[25] G. Arslan, B. L. Evans, F. A. Sakarya, and J. L. Pino, “Performance evaluation and real-time implementation of subspace, adaptive, and DFT algorithms for multi-tone detection,” in *Proc. IEEE Int. Conf. Telecommunications*, (Istanbul, Turkey), pp. 884–887, Apr. 1996.

[26] T. H. Tsim, “DTMF FAQ - Telephone Tone Dialing Chips V1.20.” http://margo.student.utwente.nl/el/phone/dtmf.htm, Aug. 1994.

[27] J. H. Beard, S. P. Given, and B. J. Young, “A discrete Fourier transform based digital DTMF detection algorithm,” in *MS State DSP Conf.*, (Mississippi), 1995.

[28] P. Mock, “Add DTMF generation and decoding to DSP-μp designs,” in *Electronic Design News*, vol. 30, pp. 205–220, Mar. 1985.

[29] H. M. Mesiwala and S. R. McCaslin, “Persistence and dynamic threhold based signal detector.” United States Patent, July 1996.

Vita

Amey Arun Deosthali was born in Pune, Maharashtra, India, on April 15, 1975, the son of Arun Sadashiv Deosthali and Vrinda Arun Deosthali. After completing his studies at the Seth Dagduram Kataria High School, Pune, India, in 1990, he entered Sir Parshurambhau College where he played for the college badminton team. After completing his higher secondary school, he entered Government College of Engineering, Pune, India, in August, 1992. During the subsequent years, he actively participated in various cultural activities. He also pursued his interests in rowing during his undergraduate studies. He received the degree of Bachelor of Engineering from Government College of Engineering in May 1996. In September, 1996, he started his studies towards Master's degree at The University of Toledo, Ohio. After studying there for one semester, he transferred to the Graduate School at The University of Texas at Austin in January 1997.

Permanent address: 3110 Red River, #208
Austin, Texas 78705

This report was typeset[1] with LaTeX by author.

[1] LaTeX document preparation system was developed by Leslie Lamport as a special version of Donald Knuth's TeX program for computer typesetting. TeX is a trademark of the American Mathematical Society. The LaTeX macro package for The University of Texas at Austin report format was written by Khe-Sing The.