Telecommunications Applications With the TMS320C5x DSPs

Edited by Mansoor A. Chishtie

Digital Signal Processing Applications — Semiconductor Group
Texas Instruments Incorporated

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1994, Texas Instruments Incorporated

Preface

This book belongs to a growing series of digital signal processing application books that Texas Instruments has published over the years. Some of these books are broad in content and cover a wide variety of DSP-related technologies and applications. Others are more focused and concentrate on one DSP application area. TI has also published many individual application reports. This particular collection of application reports focuses primarily on a variety of DSP applications that are related to the field of telecommunications and implemented on the ’C5x generation of the TMS320 family.

This book is divided into nine parts, including the introduction and the bibliography:

Part I	Introduction
Part II	Digital Cellular Systems
Part III	Speech Synthesis
Part IV	Error-Correction Coding
Part V	Baseband Modulation and Demodulation
Part VI	Equalization and Channel Estimation
Part VII	Speech and Character Recognition Algorithms
Part VIII	System Design Considerations
Part IX	Bibliography

Part I introduces the TMS320 family and the TMS320C5x generation; it also summarizes various telecommunications applications that use TMS320C5x DSPs. Parts II – VIII discuss major application topics.

Most of the papers presented here are application reports written either by the engineering staff of the TI digital signal processing department (including factory and field personnel and summer students) or by third parties. Some of the papers were contributed by other departments within TI. It is generally assumed that reader is DSP literate and has some exposure to the TMS320 DSP family.

The application reports presented in this book represent practical implementations of DSP algorithms. Source code associated with these reports is not listed in this book because of space constraints. However, most of the papers have associated source code that is publicly available from the TMS320 DSP Bulletin Board System (BBS) at 713–274–2323. The contents of this BBS are also mirrored at an Internet anonymous FTP site *ti.com.* Some technical papers included here present commercial implementations that are licensable from respective organizations. The technical data sheets of these implementations will also be included in a future update of the TMS320 Software Cooperative Library.

The editor would like to thank all the contributors and reviewers of this book. In particular, a special note of appreciation goes to Gene Frantz, Jay Reimer, Raj Chirayil, and Paul Buenaflor for their encouragement and helpful suggestions in improving the overall structure of this book. It is our hope that this book will help you in making the transition to DSP-based telecommunication applications. Lastly, the editor would like to acknowledge the untiring efforts of Ms. Katie Delbridge in planning and coordinating this project.

Mansoor A. Chishtie
Telecom Applications
Digital Signal Processing
Semiconductor Group
Texas Instruments Incorporated

Contents

Title *Page*

Part I: Introduction

Introduction 1
Overview 3
Programmable Versus Hard-Wired Solutions 3
Fixed-Point Versus Floating-point Solutions 4
TMS320 Digital Signal Processors 5
TMS320C5x Architecture 7
Summary of Telecom Applications Topics 9
Bibliographies and Other References 10

Part II: Digital Cellular Systems

Digital Cellular Phone: A Functional Analysis 11
Introduction 13
Transmitter 14
Receiver 24
Summary 29
References 29

IS-54 Simulation 31
Introduction 33
Description 35
Using the Simulation 39
Code Availability 40
References 40

Part III: Speech Synthesis

Theory and Implementation of the Digital Cellular Standard Voice Coder: VSELP on the TMS320C5x 41
Introduction 43
Overview of VSELP 43
Speech Decoder 57
Features of VSELP 59
TMS320C5x Real-Time Implementation 59
A Typical Digital Cellular Vocoder Configuration 60
Code Availability 61
References 61

Contents

Title *Page*

Part IV: Error-Correction Coding

U.S. Digital Cellular Error-Correction Coding Algorithm Implementation on the TMS320C5x 63
Abstract 65
Introduction 65
VSELP Channel Format 66
FACCH Channel Format 73
Code Availability 75
References 75

Viterbi Implementation on the TMS320C5x for V.32 Modems 77
Introduction 79
Standard V.32 Encoder 82
Viterbi Decoder 85
Viterbi Decoder Implementation 90
Performance Analysis 96
Summary 100
Code Availability 100
References 101

A TMS320C53-Based Enhanced Forward Error-Correction Scheme for U.S. Digital Cellular Radio 103
Abstract 105
Introduction 105
Algorithm Description 105
Implementation Details 107
Results 108
Conclusions 109
References 109

Part V: Baseband Modulation and Demodulation

IS-54 Digital Cellular Modem Implementation on the TMS320C5x 111
Introduction 113
Description of π/4-QPSK Modulation Scheme 113
Theory of the π/4-DQPSK Modem 115
Modem Implementation on the TMS320C5x 119
Performance Results 126
Summary 129
Code Availability 129
References 130

Contents

Title *Page*

A DSP GMSK Modem for Mobitex and Other Wireless Infrastructures 131
Abstract 133
Introduction 133
Mobitex DSP Modem Characteristics 135
Modulator Design 137
GMSK Demodulator Design 140
Conclusions 144
Code Availability 144
References 145

Part VI: Equalization and Channel Estimation

Equalization Concepts: A Tutorial 147
Introduction 149
What Is Intersymbol Interference? 149
Equalization 159
LMS Equalization 167
Code Availability 174
References 174

Channel Equalization for the IS-54 Digital Cellular System With the TMS320C5x 177
Introduction 179
Design Considerations 179
Equalizer Design 183
Choosing an Update Algorithm 186
Code Availability 187
References 187

Digital Voice Echo Canceler Implementation on the TMS320C5x 189
Introduction 191
'C5x Device Features Used in This Implementation 191
Conclusion 201
Acknowledgements 201
Code Availability 201
References 201
Appendix: Schematic of the Dual-Telephone Interface for the TMS320C51 SWDS 202

Contents

Title ***Page***

Part VII: Speech and Character Recognition Algorithms

DSP-Based Handprinted Character Recognition 203
Introduction 205
Architecture 206
System-Level software 207
Results 211
References 212

Implementation of an HMM-Based, Speaker-Independent Speech Recognition System on the TMS320C2x and TMS320C5x 213
Abstract 215
Background 215
The TMS320-Based HMM Recognizer 215
System Considerations 218
Conclusion 226

Automated Dialing of Cellular Telephones Using Speech Recognition 229
Introduction 231
The Technology 231
The Human Interface 232
The Implementation 233
Accuracy 234
Code Availability 235
Summary 236

Part VIII: System Design Considerations

The PCMCIA DSP Card: An All-in-One Communications System 237
Introduction 239
System Architecture 240
Operation 242
Conclusion 245

Software Coding Guidelines for ’C5x Developers 247
Introduction 249
Hardware Platform Overview 249
Software Organization 249
Memory Organization 252
Programming Guidelines 253
Source Code Documentation 254
Appendix: **A** Sample Linker Command File for the ’C5x Card 255

Contents

Title *Page*

TCM320AC3x/4x Voice-Band Audio Processors **259**
- Introduction 261
- Principles of Operation 262
- Transmit Channel 262
- Receive Channel 264
- Timing and Clocking 265
- Fixed- and Variable-Data-Rate Modes 266
- Application Information 267

Part IX: Bibliography

Bibliography **271**
- TMS320 Bibliography 273
- Mobile Radio Systems 273
- Modulation and Demodulation 274
- Equalization. Channel Estimation. and Adaptive Filtering 274
- Speech Recognition 275
- Speech Compression 276
- System Design Considerations 280

List of Illustrations

Figure *Title* *Page*

Introduction 1
1. TMS320 Family of Devices 5
2. Key Features of the TMS320C5x Architecture 8

Digital Cellular Phone: A Functional Analysis 11
1. Functional Components of a Dual-Mode (IS-54) Cellular Phone 13
2. Functional Blocks of the Digital Portion of a Dual-Mode Phone 14
3. Front-End Analog Section Converts Audio to a 64-kbps Data Stream 15
4. Full-Rate Speech Coder (VSELP) Reduces a 64-kbps Data Stream to an 8-kbps Data Stream 15
5. A Channel Coder and Its Functional Components With Associated Data Rates 17
6. Error Protection via Convolutional Coding and CRC Computation 18
7. Error Protection Adds 101 Extra Bits per Speech Frame 18
8. Interleaving Adjacent Frames for Error Protection 19
9. Control-Signal Multiplexing 20
10. Burst Generator 21
11. A 4-Level Modulator Groups Two Bits to Form a Symbol 22
12. π/4 Differential Quaternary PSK Modulator States 22
13. π/4 DQPSK Modulator 23
14. Linear RF Amplifiers Are Needed for IS-54 Cellular Phone 23
15. RF Portion of Receiver Section of Dual-Mode Cellular Phone 24
16. An MLSE Adaptive Equalizer 26
17. Channel Decoding and Speech Decoding 27

IS-54 Simulation 31
1. IS-54B Simulation Processing Block Diagram 34
2. IS-54 Error Encoding and Interleaving 35
3. IS-54 Convolutional Encoding Block Diagram 36
4. IS-54 Slot Formats 37

Theory and Implementation of the Digital Cellular Standard Voice Coder: VSELP on the TMS320C5x 41
1. LPC Filter Coefficient Quantization and Interpolation 47
2. Adaptive Code Book Search 48
3. Code Book Search Signal Flow 49
4. Possible Digital Cellular System Configuration 61

U.S. Digital Cellular Error-Correction Coding Algorithm Implementation on the TMS320C5x 63
1. Voice and Control-Channel Multiplexing Over One Time Slot 65
2. Error Protection for VSELP Data 66
3. Convolutional Encoder for VSELP Data 67
4. Representative Trellis Section for VSELP Convolutional Encoder 67
5. Transition Table Organization 71
6. FACCH Rate-1/4 Convolutional Encoder 74

List of Illustrations

Figure *Title* *Page*

Viterbi Implementation on the TMS320C5x for V.32 Modems 77

1. V.32 Modems 81
2. V.32 Encoder 82
3. Viterbi Encoder — Convolutional Encoding Scheme 83
4. V.32 Modem Trellis Diagram 84
5. Viterbi Decoding — Output Tracking and Cost Function 86
6. V.32 Modem — Signal Element Mapping 87
7. Viterbi Decoding — Dynamic Programming 88
8. V.32 Encoder Program Flow 89
9. Decoder Flowchart 90
10. Delay State Linking 93
11. 128-Word Circular Buffers — Format of PAST–PATH and PAST–DLY Tables 94
12. DIST Table Structure 95
13. White-Noise Impairment — Simulation Results 98

A TMS320C53-Based Enhanced Forward Error-Correction Scheme for U.S. Digital Cellular Radio 103

1. IS-54 Voice-Channel GVA Algorithm 106
2. Simulated Bit Error Rate of Serial GVA Versus VA 108
3. State Path History Trace 109

IS-54 Digital Cellular Modem Implementation on the TMS320C5x 111

1. $\pi/4$-Shifted QPSK Signal Constellation 114
2. Modulator Block Diagram 116
3. Demodulator Block Diagram 117
4. Interrupt Organization 119
5. Modem Test Configuration 127
6. BER Versus SNR for a Static AWGN Channel 128

A DSP GMSK Modem for Mobitex and Other Wireless Infrastructures 131

1. Typical Mobitex Terminal Architecture 134
2. Bit Error Rate Versus E_b/N_o Modem Performance 136
3. Idealized GMSK.3 Generation 137
4. Eye Pattern for 8-kbps GMSK.3, $2^{15}-1$ Length Pseudorandom Transmit Data 138
5. GMSK Modulator DSP Implementation 138
6. GMSK Demodulator DSP Implementation 140
7. Mobitex Packet Structure 142
8. Computer-Simulated Eye Pattern for a 19.2-kbps GMSK.5 (Amplitude Versus Time). 143

List of Illustrations

Figure *Title* *Page*

Equalization Concepts: A Tutorial 147

1. A Pulse Train to Be Transmitted 151
2. Component of r(t) 152
3. Contribution Due to x_{-1} 153
4. Contribution Due to x_1 at $t=0$ 154
5. Set of Shifted Pulse Responses 155
6. Odd Symmetry 156
7. Spectral Response at 1/(2T) 156
8. Time Response of the Raised Cosine Signal 158
9. Transmission Process With Example Pulse Responses 159
10. Case 1: Ideal Channel, No Multipath Effects 160
11. Case 2: System With a Single Unattenuated Multipath Channel 161
12. Equalization Process 162
13. Simulated Pulse Response 163
14. ZFE Filter Coefficient 166
15. Filter Output Computation 168
16. Decision-Directed Equalization 170
17. Received Signal Including Additive Noise Effects 171
18. DFE Functional Block Diagram 172

Channel Equalization for the IS-54 Digital Cellular System With the TMS320C5x 177

1. π/4 DQPSK 180
2. Multipath Interference 181
3. Rayleigh Fading 182
4. Intersymbol Interference: Interferer Level -3 dBc 183
5. Block Diagram of a Decision-Feedback Equalizer 185
6. Equalizer Taps Responding to a Fade 186

DSP-Based Handprinted Character Recognition 203

1. Prototype HHC Platform With Pen Input 206
2. DSP Card Memory Organization for HCR 209

Implementation of an HMM-Based, Speaker-Independent Speech Recognition System on the TMS320C2x and TMS320C5x 213

1. Voice Dialer Sentence Hypothesizer Flow Chart 216
2. A Minimal TMS320C53 HMM System 220
3. Example of an HMM Flow 221
4. Block Diagram of the HHM Recognizer 222
5. The Feature Extractor 223
6. Example of $Q_{4/16}$ Notation 225
7. SISR System for Very Large Vocabulary 226

List of Illustrations

Figure	*Title*	*Page*
Automated Dialing of Cellular Telephones Using Speech Recognition		**229**
1.	Flow Diagram of Human Interface	233
The PCMCIA DSP Card: An All-in-One Communications System		**237**
1.	DSP Card Block Diagram	239
2.	DSP Card Architecture	241
3.	Loading and Executing a Single Algorithm	244
Software Coding Guidelines for ’C5x Developers		**247**
1.	Categories of Source Code Files	250
TCM320AC3x/4x Voice-Band Audio Processors		**259**
1.	VBAP Functional Block Diagram	261
2.	VBAP Microphone Connection	262
3.	VBAP Interfaced to a ’C5x DSP	267

List of Tables

Table **Title** Page

Introduction 1
- 1. Benefits of TMS320C5x Features 7

Digital Cellular Phone: A Functional Analysis 11
- 1. Basic Parameters of a VSELP Speech Coder 15
- 2. Bit Allocations Within a Frame of Speech. 16
- 3. Detailed Bit Allocations of Parameters Within a Frame 16
- 4. Interleaving of Two Adjacent Speech Frames, x and y 19

Theory and Implementation of the Digital Cellular Standard Voice Coder: **VSELP on the TMS320C5x** 41
- 1. Primary VSELP Parameters 44
- 2. VSELP Frame Bit Allocation 45
- 3. VSELP Vocoder Processor Requirements 59
- 4. VSELP Vocoder Memory Requirements 59

Viterbi Implementation on the **TMS320C5x** for V.32 Modems 77
- 1. Program Benchmarks 99
- 2. V.32 Encoder Code 99
- 3. V.32 Decoder Code 99

A **TMS320C53-Based** Enhanced Forward Error-Correction Scheme for U.S. Digital Cellular Radio 103
- 1. Algorithm Execution Time on a 35-ns TMS320C53 107
- 2. Memory Requirement 108

IS-54 Digital Cellular Modem Implementation on the **TMS320C5x** 111
- 1. Phase Calculation 115
- 2. Reduced Equations 121
- 3. Odd-Symbol Look-Up 121
- 4. Even-Symbol Look-Up 122
- 5. ModulatorLook-Up 122
- 6. Program Memory and Speed Requirements 128
- 7. Modulator Code Size and Execution Time 129
- 8. Demodulator Code Size and Execution Time 129

A DSP GMSK Modem for **Mobitex** and Other Wireless Infrastructures 131
- 1. Receiver Code Processor Power Requirements 135

Channel Equalization for the IS-54 Digital Cellular System With the **TMS320C5x** 177
- 1. Complexity Comparison of Update Algorithms 186

Digital Voice Echo Canceler Implementation on the **TMS320C5x** 189
- 1. User-Defined System Parameters 197
- 2. Program Module Requirements 198
- 3. 512-Tap Implementation Data Variables 199
- 4. Code Benchmarks 200

List of Tables

Table *Title* *Page*

DSP-Based Handprinted Character Recognition 203

1. Application Command Table for HCR Subsystem 211

Implementation of an HMM-Based, Speaker-Independent Speech Recognition System on the TMS320C2x and TMS320C5x 213

1. Current HMM Vocabulary (49 Words) 217
2. HMM Processor Loading on a TMS320C5x 219
3. Examples of $Q_{n/m}$ Notations (Fixed-Point Representation) 225

The PCMCIA DSP Card: An All-in-One Communications System 237

1. DSP Card Registers 242

TCM320AC3x/4x Voice-Band Audio Processors 259

1. Receive-Channel Volume-Control Bits 265
2. VBAP Master Clock Frequencies 265
3. Power-Down and Standby Procedures 266

List of Examples

Example *Title* *Page*

U.S. Digital Cellular Error-Correction Coding Algorithm Implementation on the TMS320C5x 63

1. Pseudocode for Trellis Expansion 69
2. Trellis Expansion Macro in 'C5x Assembly Code 70
3. Trace-Back Function — Pseudo-C Code 72
4. Trace-Back Implementation in 'C5x Assembly Code 73

Digital Voice Echo Canceler Implementation on the TMS320C5x 189

1. Zero-Overhead Loops UPDATE.ASM 192
2. Echo Estimation Routine FIR.ASM 192
3. Coefficient Update Routine TAPINC.ASM 193
4. Near-End Speech Detection Routine NESPDET.ASM 193
5. Echo Simulation Filter EFILT.ASM 194
6. Use of Delayed Branches NESPDETASM 195
7. Code Excerpt from MULAW.ASM 195
8. Taps Update Routine UPDATE.ASM 196
9. Serial Port ISR ECHOISR.ASM 196

Part I
Introduction

Introduction

Mansoor A. Chishtie
Digital Signal Processing Applications – Semiconductor Group
Texas Instruments Incorporated

Overview

The use of programmable digital signal processors (DSPs) is growing rapidly in telecommunication applications. Conventional wire-line telephony applications were among the earliest adopters of digital signal processing technologies. High-speed telephone-line modem products use more general-purpose DSPs than most other industries, and recent growth of personal and mobile communication services has spurred new interest in high-performance DSPs. With the ongoing integration of mobile communication services and portable computer applications, the role of programmable DSPs in emerging products is expanding. Today, digital signal processors are moving from high-end, low-volume applications to mainstream consumer applications.

Telecommunication applications can be broadly categorized into two classes:

1. **Core Applications.** These applications are the essence of any telecommunication product and include baseband signal processing algorithms, voice and data compression, error correction techniques, and equalization and channel estimation.
2. **Enabler Applications.** These applications provide necessary human interface, improve overall quality of an end-product, and include speech and character recognition, echo cancellation, and noise cancellation.

Programmable Versus Hard-Wired Solutions

DSPs are following the path of microprocessors in terms of performance and on-chip integration. At the same time, users of DSPs are concerned about power consumption. As the communications industry improves portable applications, low power and high integration become key design care-abouts. Generally speaking, a product design is constrained by one or more of the following key design goals, not necessarily with equal importance:

- Power consumption
- Product form factor
- Upgradability
- Cost of product
- Cost of design
- System integration

These design goals play key roles in selecting a programmable versus function-specific or hard-wired DSP solution.

Newer generation DSPs are addressing these concerns. They support various low-power and power-down modes along with clock control options to help meet power goals. System integration and form-factor goals are often interrelated. With high on-chip integration of peripherals and memory, modem DSPs are well-suited for portable applications in which product form factor is extremely important. In Part VIII, "The PCMCIA DSP Card: An All-in-One Communications System", page 237, describes a DSP system based on Personal Computer Memory Card Interface Association (PCMCIA) type II card specifications. Many DSPs are now available in thin low-profile plastic packages, which are ideal for surface-mount applications.

In today's evolving communications world, flexibility and upgradability of design are key factors in longer product cycles. Many personal communication standards are in the early stages of development. Some of these standards must maintain compatibility with older standards. Programmable DSPs are especially suitable for designs that require multiple modes of operation and future upgradability. In a U.S. digital cellular subscriber unit, a programmable DSP engine can easily handle the two-mode operation.

Finally, the traditional distinction between programmable and function-specific DSP designs is fading because of customizable DSP (cDSP) solutions. Now, designers can decide which section of a design is best suited for a hard-wired approach. Code that must maintain upgradability can be downloaded into on-chip RAM. The rest of the program can be masked on on-chip ROM. Algorithm accelerators or custom peripherals can be designed and placed on the same die. These techniques can be implemented through the TI standard cDSP cell design methodology or through the standard gate-array design flow of the TEC320 product line.

Fixed-Point Versus Floating-Point Solutions

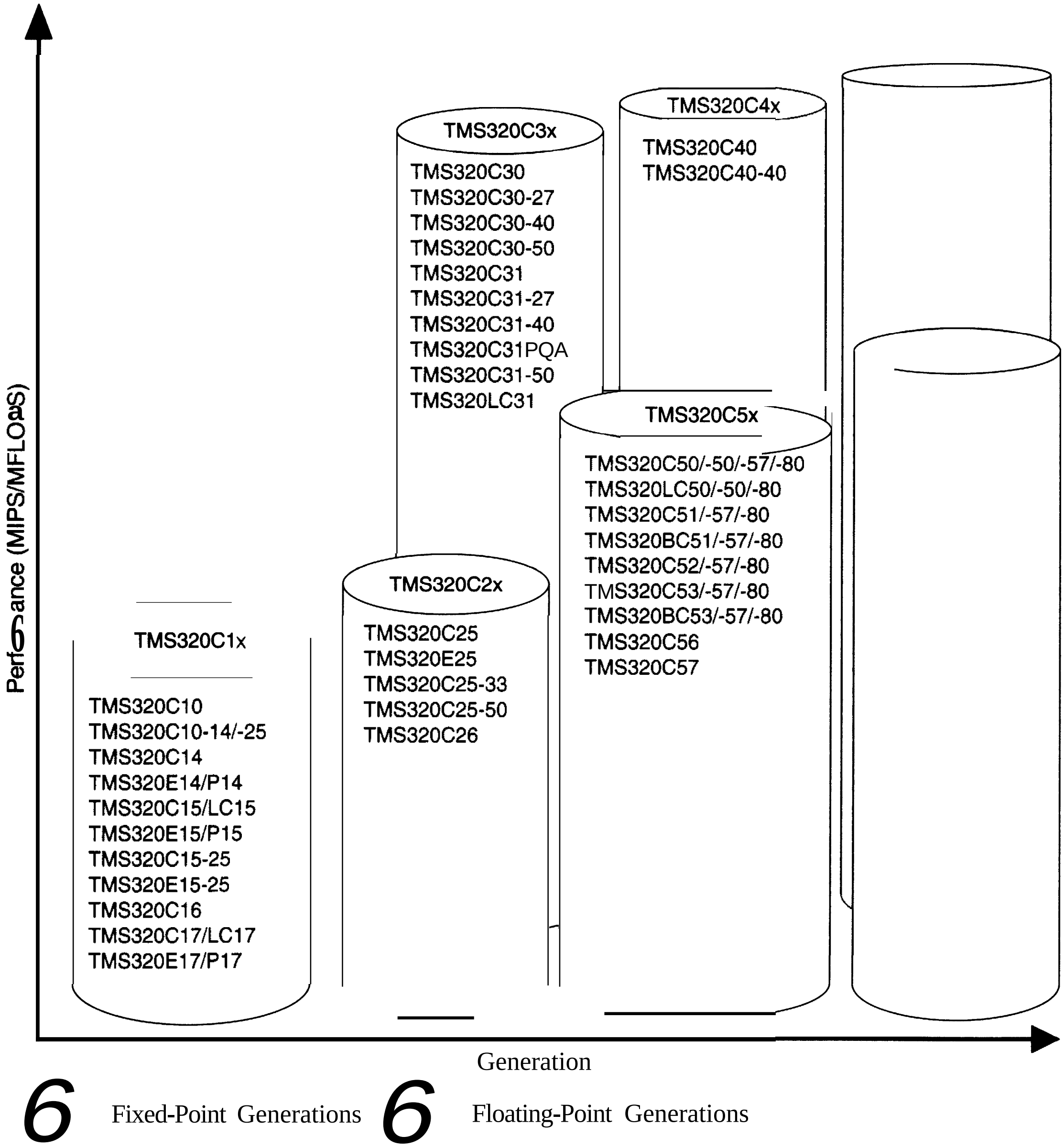
Typically, floating-point DSPs are used in high-end, high-performance telecom applications such as video conferencing, network packet switching, cellular base stations, etc. Floating-point DSPs offer large dynamic range, a fast floating-point computation engine, and large-memory addressability. Due to wider instruction word size, they support more addressing modes and higher execution unit parallelism as well. Floating-point support and large operand dynamic range result in an ease of transition from simulation environment to real-time implementation. **A** more orthogonal instruction set helps in providing efficient high-level language code generation tools.

On the other hand, fixed-point 16-bit DSPs are very popular in high-volume, low-power applications. Generally, they consume less power and cost less because of a smaller die size. They can be operated at faster speeds because of their relatively simple architecture and fewer speed paths. Newer fixed-point DSPs provide application-specific instructions and on-chip power management for portable and mobile communication applications. Due to their prevalence in the mobile communications market, many upcoming industry standards are fine-tuned for 16-bit fixed-point implementations. One such example is the voice compression specification of U.S. Digital Cellular Standard, the IS-54. This algorithm is optimized for 16-bit fixed-point DSP engines. With improved compiler support and a more orthogonal instruction set, the end-product development cycle has also become shorter.

TMS320 Digital Signal Processors

The TMS320 family consists of five generations of fixed-point and floating-point devices (see Figure 1). Members of each generation are object-code and, in some cases, pin compatible. Each generation offers unique features and capabilities, which are optimized for certain types of applications.

Figure 1. TMS320 Family of Devices


Performance (MIPS/MFLOPS)
TMS320C3x
TMS320C30
TMS320C30-27
TMS320C30-40
TMS320C30-50
TMS320C31
TMS320C31-27
TMS320C31-40
TMS320C31PQA
TMS320C31-50
TMS320LC31
TMS320C4x
TMS320C40
TMS320C40-40
TMS320C5x
TMS320C50/-50/-57/-80
TMS320LC50/-50/-80
TMS320C51/-57/-80
TMS320BC51/-57/-80
TMS320C52/-57/-80
TMS320C53/-57/-80
TMS320BC53/-57/-80
TMS320C56
TMS320C57
TMS320C2x
TMS320C25
TMS320E25
TMS320C25-33
TMS320C25-50
TMS320C26
TMS320C1x
TMS320C10
TMS320C10-14/-25
TMS320C14
TMS320E14/P14
TMS320C15/LC15
TMS320E15/P15
TMS320C15-25
TMS320E15-25
TMS320C16
TMS320C17/LC17
TMS320E17/P17
Generation
6 Fixed-Point Generations
6 Floating-Point Generations

TMS320 Fixed-Point DSPs

The three generations of TMS320 fixed-point DSPs — TMS320C1x, TMS320C2x, and TMS320C5x — have a 16-bit architecture with a 32-bit ALU and accumulator. They are based on Harvard architecture with separate buses for program and data, allowing instructions and operands to be fetched in parallel. They also feature a 16 × 16-bit hardware multiplier for single-cycle multiply operations, and a hardware stack for fast interrupt response time. An overflow saturation mode prevents wraparound. Most of the instructions are executed in a single cycle. Performance currently ranges from 3.5 to 40 MIPS (million instructions per second). Even higher performance DSPs will become available in the near future.

The TMS320C1x generation is based on the first DSP, the TMS32010, which was introduced in 1982. 'C1x devices include 1441256 words of on-chip RAM and 4K to 8K words of on-chip ROM. Instruction cycle time is 114 to 280 ns. Members of this generation include the TMS320C10, TMS320C14, TMS320E14 (the EPROM version of the TMS320C14), TMS320C15/E15, TMS320C16, and TMS320C17/E17. The TMS320C14/E14 has been optimized for control applications. The TMS320C16 has an expanded memory address space of 64K words. Low-power versions are also available for 3-volt designs.

The TMS320C2x generation is based on the TMS320C25, featuring 544 words of on-chip RAM and 4K words of on-chip ROM. Total address space is expanded to 64K words for both data and program. The instruction set has been considerably enhanced over the TMS320C1x instruction set, reducing the instruction cycle time to 120/80 ns. Other members of the 'C2x generation include the TMS320E25 (an EPROM version of TMS320C25), the TMS320C26, and the TMS320C28, which expands the on-chip RAM and ROM.

The TMS320C5x generation includes the TMS320C50 (10K words of on-chip RAM, 2K words of on-chip ROM), TMS320C51 (2K words of on-chip RAM, 8K words of on-chip ROM), TMS320C52 (1K words of on-chip RAM, 4K words of on-chip ROM), TMS320C53 (4K words of on-chip RAM, 16K words of on-chip ROM), and TMS320C53SX (4K words of on-chip RAM, 16K words of on-chip ROM). All the devices except the 'C52 have two serial ports; the 'C52 has one. Most of the devices in this generation are available in thin plastic (132- and 100-pin) quad flatpack packages. With an enhanced instruction set, TMS320C5x devices can execute code at the rate of 25 ns per instruction. New architecture features include a bit-manipulation unit, called PLU (parallel logic unit), shadow registers for fast context switch, JTAG serial scan emulation, and zero-overhead loops. Low-power versions are also available.

TMS320 Floating-Point DSPs

The two generations of TMS320 floating-point DSPs — TMS320C3x and TMS320C4x (the first DSP designed for parallel processing) — have a 32-bit architecture with 40-bit extended-precision registers. They are based on Von Neuman architecture. Multiple buses have been added for faster throughput. Features include a hardware floating-point multiplier and a floating-point ALU.

The TMS320C3x generation is based on the TMS320C30 and features 2K × 32 words of on-chip RAM, 4K × 32 words of on-chip ROM, and a 64-word on-chip instruction cache. 'C3x devices include an on-chip DMA controller, two serial ports, two timers, two external 32-bit data buses, and a 16M-word linear address space. Instruction cycle rates are 60 and 50 ns, with peak performance of 40 MFLOPS (million floating-point operations per second). A low-power version of TMS320C31 features special instructions for power management.

The TMS320C4x generation includes the TMS320C40, a parallel digital signal processor. It includes six communications ports, a self-programmable six-channel DMA coprocessor, a developing/debugging analysis module, two independent 32-bit memory interfaces, a 16G-byte address space, and two timers. Other features includes two 4K-byte RAM blocks, one 16K-byte ROM block, and a 512-byte instruction cache. This generation is designed to execute each instruction in 40 ns, perform up to 275 MOPS (million operations per second), and provide 320M-bytelsecond throughput.

TMS320C5x Architecture

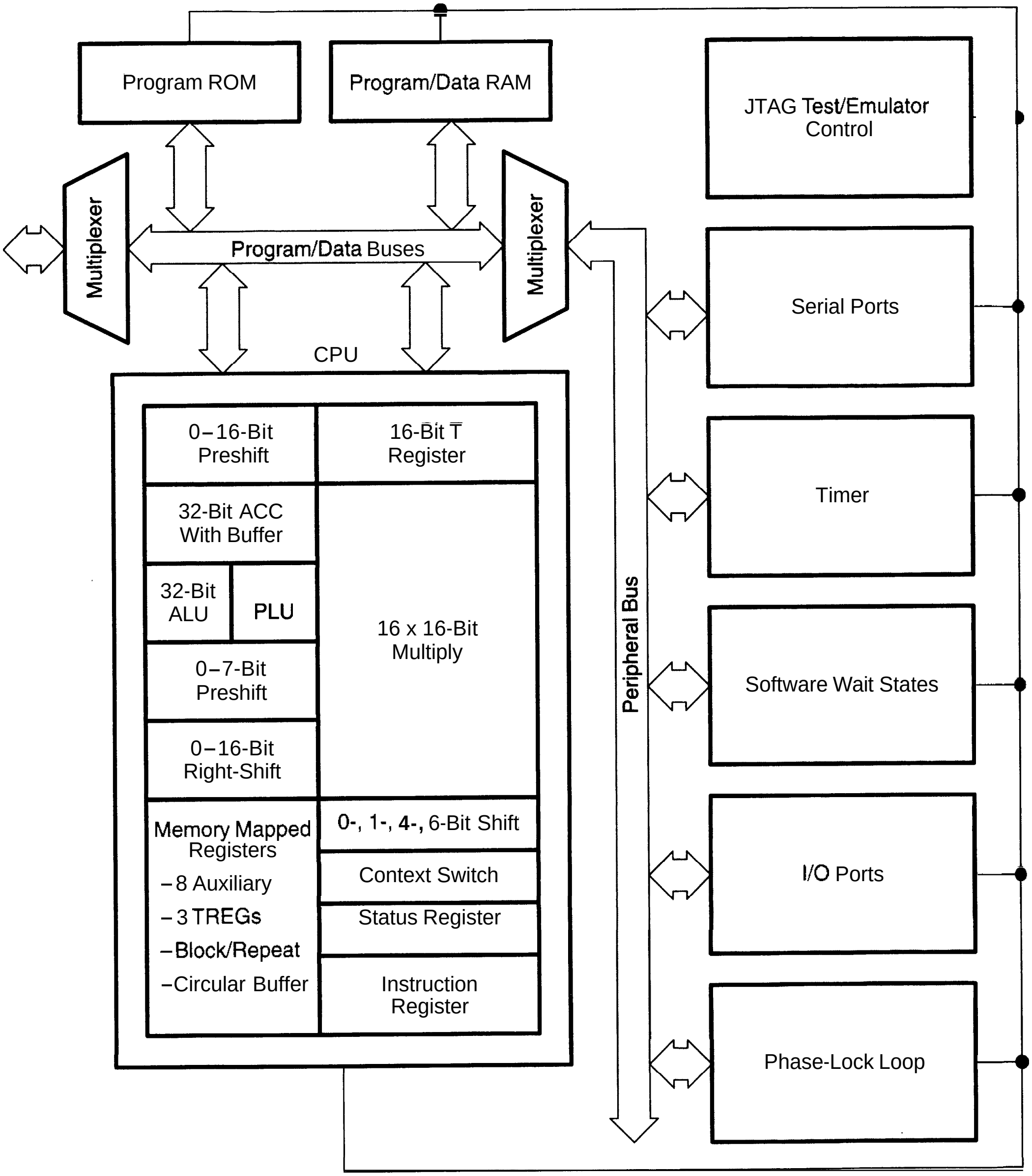
The TMS320C5x generation is designed to perform complex computation-intensive signal processing in real time. It has a high-performance pipelined architecture that enables it to execute each instruction at the maximum rate of 25 ns per instruction. It has a familiar 16132-bit accumulator-based architecture with a 16-bit wide external address bus and a hardware multiplier similar to traditional DSP architectures. It includes a bit-manipulation or parallel logic unit, (PLU), which allows it to efficiently implement traditional microcontroller-type operations. Automatic interrupt context switch and reduced interrupt latency are made possible by on-chip shadow registers and an 8-word deep hardware stack. On-chip peripherals include two serial ports (one of which can be used in the time division multiplex mode), one timer, a wait-state generator, and a phase-locked loop for clock frequency multiplication. Figure 2 on page 8 shows the key features of the TMS320C5x architecture.

The TMS320C5x architecture introduces several new features to make it suitable for telecommunication and related applications. Traditional communication designs (such as modems and cellular radios) use a microcontroller and one or more digital signal processors. Typical microcontroller tasks are system control, general housekeeping, and user interface. These tasks are generally run on a microcontroller because they do not require a high-performance processor. Additionally, these functions are often written in C and involve bit manipulation. The 'C5x bit manipulation unit (PLU), memory-mapped input-output ports, dynamic postscalers and prescalers, and C language support enable these traditional microcontroller tasks to be efficiently implemented. Salient features and benefits of TMS320C5x architecture are shown in Table 1.

Table 1. Benefits of TMS320C5x Features

Feature	Benefit
Harvard architecture	Simultaneously accesses instructions and data operands
Parallel logic unit	Allows direct bit manipulation on memory operands
Shadow registers	Allow zero-overhead context switch for interrupts
Hardware stack	Supports fast interrupt processing
Repeat-block loops	Reduce overhead of looped code
Memory-mapped 110 ports	Efficiently handle peripheral data transfer
Circular buffers	Implement queues. delay lines, circular convolution, etc.
Hardware multiplier	Supports single-cycle signed and unsigned integer multiplication
Power-down modes	Reduce active and idle power consumption
High-speed, single-cycle instruction execution unit	Helps implement advanced signal-processing algorithms in real time

Figure 2. Key Features of the TMS320C5x Architecture


Program ROM
Program/Data RAM
JTAG Test/Emulator Control
Multiplexer
Program/Data Buses
Multiplexer
Serial Ports
CPU
0–16-Bit Preshift
16-Bit T Register
32-Bit ACC With Buffer
32-Bit ALU
PLU
16 x 16-Bit Multiply
0–7-Bit Preshift
0–16-Bit Right-Shift
Memory Mapped Registers
– 8 Auxiliary
– 3 TREGs
– Block/Repeat
– Circular Buffer
0-, 1-, 4-, 6-Bit Shift
Context Switch
Status Register
Instruction Register
Peripheral Bus
Timer
Software Wait States
I/O Ports
Phase-Lock Loop

Summary of Telecom Applications Topics

Digital Cellular Systems

Digital cellular radio designs use general-purpose DSPs to perform speech synthesis, error-correction coding, baseband modem, and system control applications. Where other parts of this book concentrate on these individual applications, Part II focuses primarily on overall system design and highlights tasks suitable for DSP implementation.

Speech Synthesis

Speech compression and coding is one of the earliest and most widely used DSP applications. In both wireline or wireless communications vocoders are used to compress speech signals for limited bandwidth channels. An application paper on U.S. Digital Cellular vocoder implementation is presented in this section.

Error-Correction Coding

Forward error-correction (FEC) schemes are widely used in telecom applications to reduce bit error rate (BER) on noisy channels. The need for improved FEC techniques is becoming more prominent these days as more data is pumped through limited bandwidth channels. Cyclic redundancy check (CRC) and bit parity check are still used for simple error detection. However, more complex forward error-correction schemes such as convolutional encoding with Viterbi decoding and Reed-Solomon (RS) codes are often used to detect and correct multiple bit errors. Often, concatenated coding schemes are used to provide even more protection against bit errors than is possible with a single scheme. One such example is IS-54 voice channel specification, in which Class I bits are protected by both the CRC and convolutional codes. This is described in a paper that is presented in this part. Another conference paper on FEC schemes is also included here, and a third describes an implementation of forward error-correction technique used for V.32 modems.

Baseband Modulation and Demodulation

Programmable digital signal processors can provide necessary performance and throughput to implement baseband modem functions. These functions include symbol timing recovery, automatic gain and frequency control, symbol detection, pulse-shaping, and matched filters. Many of these functions were formerly implemented in hardware. With the advent of high-performance DSPs and the growing need for multipurpose hardware designs, many of these functions are being implemented in DSP software. One such example is the U.S. Digital Cellular IS-54 standard for mobile phones, in which every terminal is required to handle three modulation schemes: FM, FSK, and DQPSK. Two papers are presented in this book on this subject.

Equalization and Channel Estimation

Another computationally intensive DSP task is channel modeling for estimation of echo, noise, or intersymbol interference. Line echo cancellation is a common wireline telephony application suitable for DSP implementation. Acoustic echo and noise cancellation techniques are equally important for wireline and wireless communication links. Equalization is another channel estimation technique for removal of intersymbol interference caused by channel delay spread. The first paper in this section presents a tutorial on equalization techniques. The other two papers present implementation details of an equalizer and a line echo canceller.

Speech and Character Recognition Algorithms

DSPs are often called upon to perform user-interface tasks in addition to core applications. This is a direct consequence of one very important feature of a DSP-based product: flexibility of design. This flexibility allows system designers to load additional tasks on their DSPs to better utilize spare MIPS. A pertinent example is that of a mobile phone; the voice dialing feature can be easily implemented on a DSP without additional DSP horsepower. This is because the phone will be on-hook (or off-air), and the DSP will have many spare MIPS available when the voice dialing feature is enabled. With the onset of personal digital assistant (PDA) technology in which computers and communication applications merge, human interface designs are gaining more importance. Three application papers are presented in this section.

System Design Considerations

Every DSP system engineer deals with several design care-abouts. This part highlights some of these general hardware and software design considerations. The paper "The PCMCIA DSP Card: An All-in-One Communications System" presents an embedded DSP hardware design example. The second paper, "Software Coding Guidelines for 'C5x Developers" outlines general programming guidelines for TMS320C5x assembly language programmers. Finally, the paper "TCM320AC3x/4x Voice-Band Audio Processors" describes DSP applications with voice-band audio processors.

Bibliographies and Other References

To keep TMS320 designers aware of new applications and developments related to the TMS320 DSPs, Texas Instruments has published extensive bibliographies of TMS320-related conference papers and technical articles. Part IX of this book serves as an extension to the previously published bibliographies. It lists only those papers and articles that are generally related to telecommunication applications.

In addition to this collection of telecommunications-related papers on TMS320C5x digital signal processors, Texas Instruments has published related application papers on other TI digital signal processors. For more information, refer to Volumes 1, 2, and 3 of *Digital Signal Processing Applications with the TMS320 Family: Theory, Algorithms, and Implementations.*

Part II

Digital Cellular Systems

Digital Cellular Phone: A Functional Analysis

B. I. (Raj) Pawate
Mansoor A. Chishtie
Digital Signal Processing Applications — Semiconductor Group
Texas Instruments Incorporated

Introduction

This document presents the functional components of a dual-mode cellular phone as specified by the CTIA IS-54 standard. For each functional component, the relevant algorithm, its data structures, if any, and implementation details are given.

A Functional View of a Dual-Mode Cellular Phone

As shown in Figure 1, a dual-mode cellular phone consists of the following:

- Transmitter
- Receiver
- Coordinator
- Antenna assembly
- Control panel

A dual-mode phone is capable of operating in an analog-only cell or a dual-mode cell. Both the transmitter and the receiver support both analog FM and digital time division multiple access (TDMA) schemes. Digital transmission is preferred, so when a cellular system has digital capability, the mobile unit is assigned a digital channel first. If no digital channels are available, the cellular system will assign an analog channel.

The transmitter converts the audio signal to a radio frequency (RF), and the receiver converts an RF signal to an audio signal. The antenna focuses and converts RF energy for reception and transmission into free space. The control panel serves as an input/output mechanism for the end user; it supports a keypad, a display, a microphone, and a speaker. The coordinator synchronizes the transmission and receive functions of the mobile unit.

Figure 1. Functional Components of a Dual-Mode (IS-54) Cellular Phone

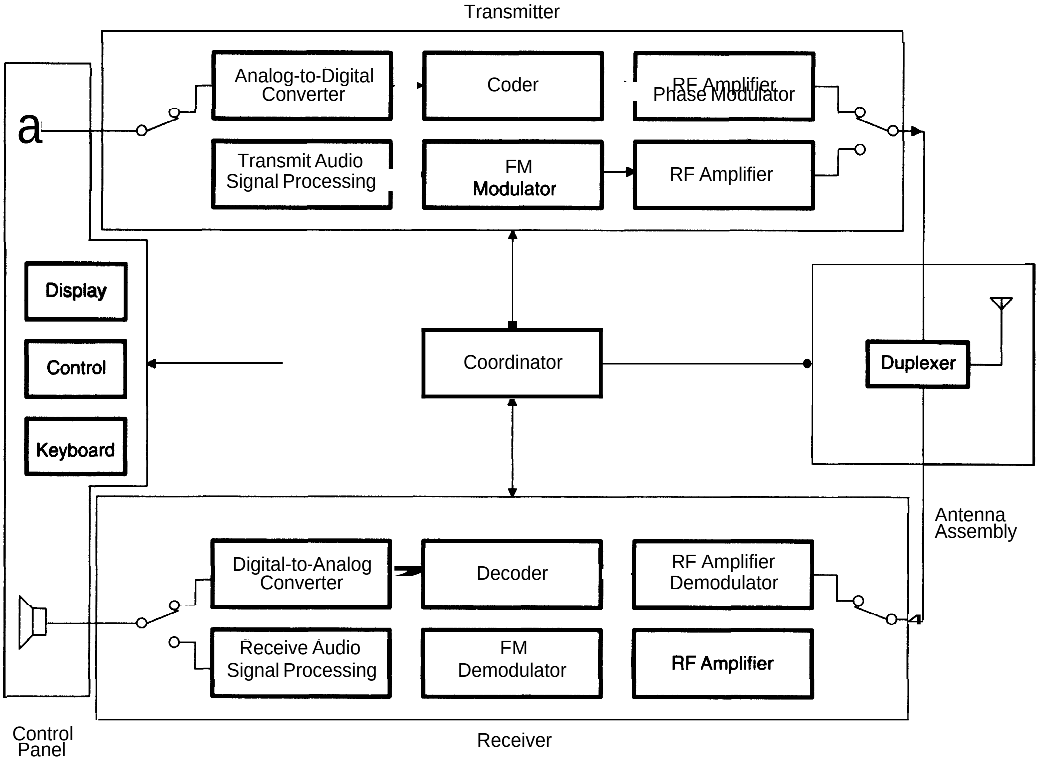
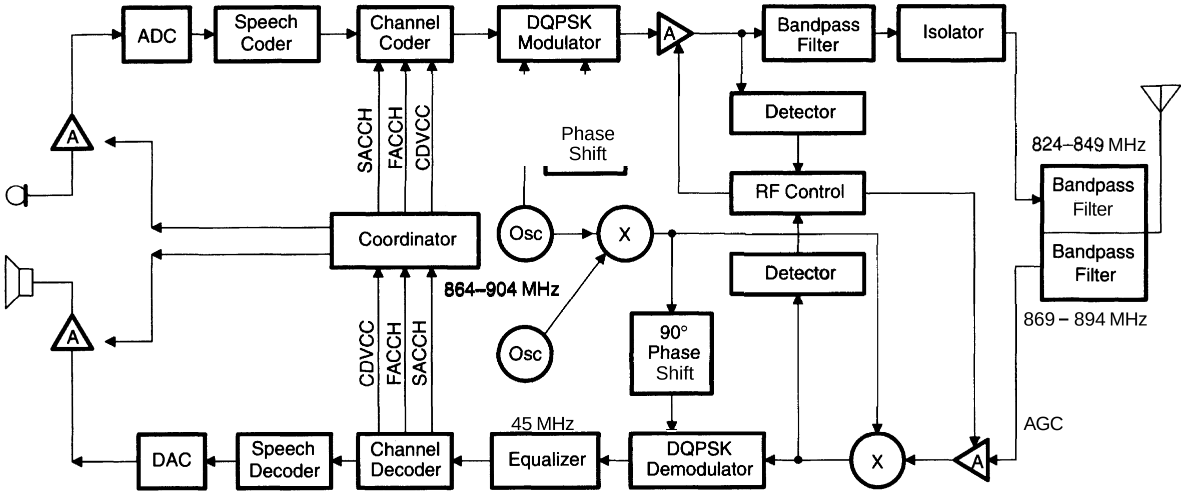

Transmitter
Analog-to-Digital Converter
Coder
RF Amplifier
Phase Modulator
Transmit Audio Signal Processing
FM Modulator
RF Amplifier
Display
Control
Keyboard
Coordinator
Duplexer
Antenna Assembly
Digital-to-Analog Converter
Decoder
RF Amplifier Demodulator
Receive Audio Signal Processing
FM Demodulator
RF Amplifier
Control Panel
Receiver

Figure 2 shows the functional components of the **digital portion** of a dual-mode cellular phone.

Figure 2. Functional Blocks of the Digital Portion of a Dual-Mode Phone


ADC
Speech Coder
Channel Coder
DQPSK Modulator
A
Bandpass Filter
Isolator
Detector
SACCH
FACCH
CDVCC
Phase Shift
824–849 MHz
RF Control
Bandpass Filter
Bandpass Filter
Coordinator
Osc
X
Detector
864–904 MHz
869 – 894 MHz
CDVCC
FACCH
SACCH
Osc
90° Phase Shift
AGC
45 MHz
DAC
Speech Decoder
Channel Decoder
Equalizer
DQPSK Demodulator
X
A

CDVCC = coded digital verification color code
DQPSK = differential quaternary phase-shin keying
FACCH = fast associated control channel
SACCH = slow associated control channel

Transmitter

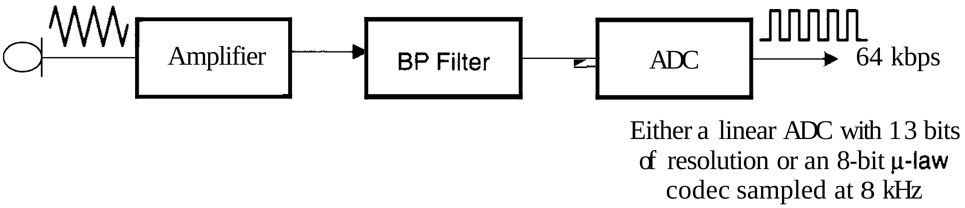
The transmitter converts low-level audio signals from the microphone to digitally coded RF signals by audio processing, digital signal processing, modulation, and RF amplification. The transmitter converts 64-kbps pulse code modulation (PCM) data to a lower data rate, multiplexes control information, error-protects the data, and then passes the data stream to the RF section for modulation, amplification, and transmission. The coordinator inserts system control messages.

Transmit Front-End Processing

Speech signals from the microphone are first amplified, passed through an antiliasing filter, and sampled at a rate of 8 kHz to create a digitized p-law 64-kbps bit stream. Typically, no pre-emphasis is applied. Figure 3 shows the functional blocks of the front-end analog section. The standard does not propose any specific echo canceler; however, it recommends implementing one. The front-end processing includes the following:

- An amplifier. The gain is specified to produce an average signal energy, during a frame, which is 18 dB down from full scale.
- A bandpass filter to avoid antialiasing.
- An analog-to-digital converter. The standard recommends that you either directly convert the analog signal to a uniform PCM format with a minimum resolution of 13 bits or convert the analog signal to an 8-bit p-law codec sample.

Figure 3. Front-End Analog Section Converts Audio to a 64-kbps Data Stream

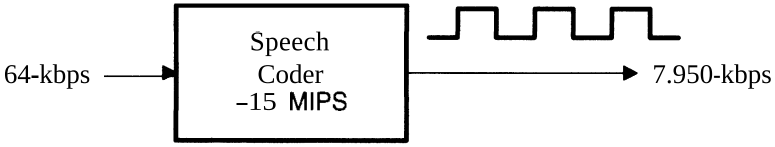

Amplifier
BP Filter
ADC
64 kbps
Either a linear ADC with 13 bits of resolution or an 8-bit µ-law codec sampled at 8 kHz

Speech Coder

The speech coder further reduces the data rate by compressing the 64-kbps data stream input to create a 7.950-kbps data stream. The IS-54 standard accepts a full-rate speech coder called ***vector sum excited linear prediction*** (VSELP). This algorithm belongs to a class of speech coders known as ***code excited linear predictive coders*** (CELP). This class uses code books to vector quantize the excitation (residual) signal. VSELP is a variation on CELP.

The incoming 64 kbps of data are grouped into frames at a frame rate of 50 frames per second. Hence, each frame contains 160 samples and represents a duration of 20 ms. Each frame is coded into 159 bits. Hence, the rate of the conversions is $50 \times 159 = 7950$ bps, as shown in Figure 4.

Figure 4. Full-Rate Speech Coder (VSELP) Reduces a 64-kbps Data Stream to an 8-kbps Data Stream



The speech decoder utilizes two separate code books. Each code book has an independent gain. The two code-book excitations are each multiplied by their corresponding gains and summed to create a combined code-book excitation. The basic parameters are shown in Table 1.

Table 1. Basic Parameters of a VSELP Speech Coder

Parameter	Notation	Specification
Sampling rate	s	8 kHz
Frame length	N_f	160 samples (20 ms)
Subframe length	N	40 samples (5 ms)
Short-term predictor order	N_p	10
Number of taps for long-term predictor	N_L	1
Number of bits in code word 1 (number of basis vectors)	$M1$	7 bits
Number of bits in code word 2 (number of basis vectors)	$M2$	7 bits

NOTE: Within a frame, the 159 bits are allocated as shown in Table 2; detailed bit allocations are shown in Table 3.

Table 2. Bit Allocations Within a Frame of Speech

Parameter	Bits Allocated
Short-term filter coefficients	38
Frame energy, RO	5
Lag, L	28
Code words, I, H	56
Gains beta, gamma1, gamma2	32

Table 3. Detailed Bit Allocations of Parameters Within a Frame

Parameter	Parameter Name	Bits Allocated
Frame energy	RO	5
1st reflection coefficient	LPC1	6
2nd reflection coefficient	LPC2	5
3rd reflection coefficient	LPC3	5
4th reflection coefficient	LPC4	4
5th reflection coefficient	LPC5	4
6th reflection coefficient	LPC6	3
7th reflection coefficient	LPC7	3
8th reflection coefficient	LPC8	3
9th reflection coefficient	LPC9	3
10th reflection coefficient	LPC10	2
Lag for first subframe	LAG-1	7
Lag for second subframe	LAG_2	7
Lag for third subframe	LAG_3	7
Lag for fourth subframe	LAG_4	7
1st code book, I, for first subframe	CODE1_1	7
1st code book, I, for second subframe	CODE1_2	7
1st code book, I, for third subframe	CODE1_3	7
2nd code book, H, for first subframe	CODE2_1	7
2nd code book, H, for second subframe	CODE2_2	7
2nd code book, H, for third subframe	CODE2_3	7
2nd code book, H, for fourth subframe	CODE2_4	7
{GS, PO, P1} code for first subframe	GSP0_1	8
{GS, PO, P1} code for second subframe	GSP0_2	8
{GS, PO, P1} code for third subframe	GSP0_3	8
{GS, PO, P1} code for fourth subframe	GSP0_4	8

Channel Coder

The main function of the channel coder is to protect the data stream against the noise and fading that are inherent to a radio channel. The coder accomplishes this by adding extra or redundant bits. The greater the number of redundant bits, the higher the immunity to interference and the lower the bit-error rate. The tradeoff is an increased data rate.

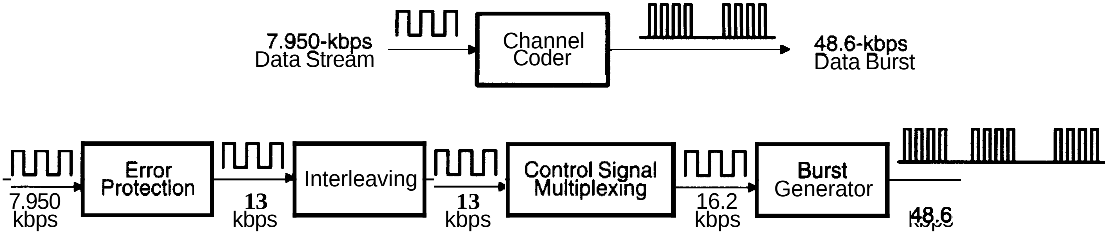
The channel coder protects the data stream in four stages:

1. Convolutional coding
2. Cyclic redundancy check (CRC) generation
3. Interleaving
4. Burst generation

The first two are *mathematical* operations, whereas the last two are *heuristic* approaches. The receiver performs an inverse operation to determine whether errors have occurred during propagation. In radio propagation, it has been found that the fading occurs at localized instances of time and space. As a result, interleaving spreads the information of the data stream across two frames, because it is unlikely that a clustered bit error would occur in successive frames. Finally, data is propagated in bursts.

Between interleaving and burst generation, the channel coder multiplexes control information. Figure 5 shows the functional components of a channel coder.

Figure 5. A Channel Coder and Its Functional Components With Associated Data Rates



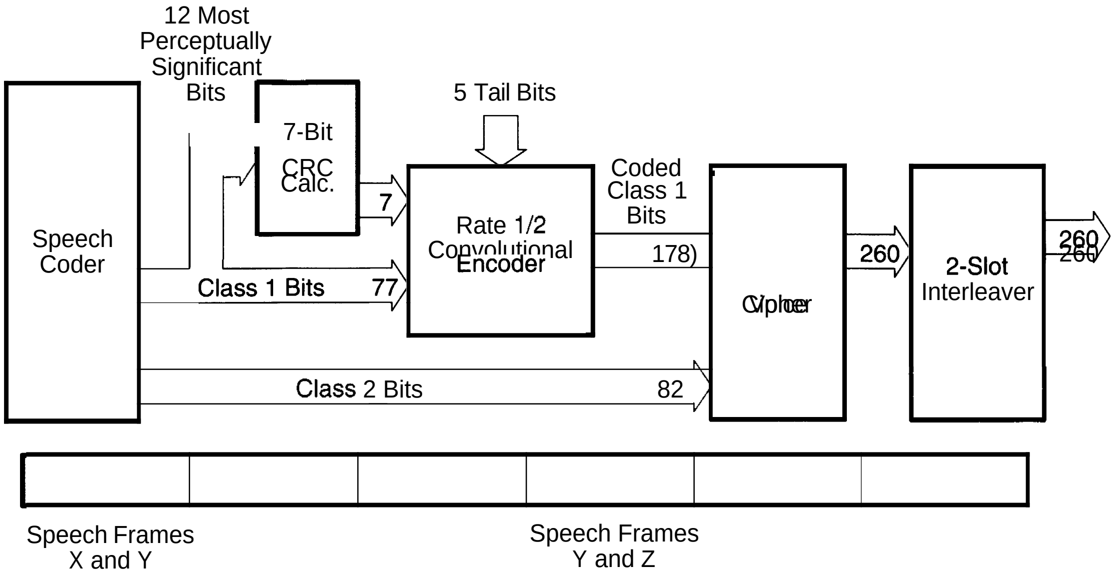
Convolutional Coding

Convolutional coding provides error-correction capability by adding redundancy to the transmitted sequence. Convolutional encoding is implemented by linear feed-forward shift registers.

A convolutional coder is described by the rate at which data enters the coder and the rate at which data leaves the coder. For example, a rate-1/2 convolutional coder implies that for every 1 bit of data entering the coder, 2 bits leave the coder. The smaller the ratio, the greater the redundancy. This improves the error-protection capability.

To reduce the bit rate, not all of the 159 bits in a frame are error-protected. Only 77 of these bits, called class 1 bits, are error-protected. The remaining 82 bits, called class 2 bits, are not error-protected. This is shown in Figure 6.

Figure 6. Error Protection via Convolutional Coding and CRC Computation


12 Most Perceptually Significant Bits
7-Bit CRC Calc.
5 Tail Bits
7
Rate 1/2 Convolutional Encoder
Coded Class 1 Bits
178
Speech Coder
Class 1 Bits
77
Cipher
Voice
260
2-Slot Interleaver
260
Class 2 Bits
82
Speech Frames X and Y
Speech Frames Y and Z

Cyclic Redundancy Check

Of the 77 bits that are error-protected, it has been found that only 12 are perceptually significant. Hence these are protected by using a 7-bit cyclic redundancy computation before they are input to the convolutional coder. **A** 7-bit CRC is computed by dividing the data by a specified constant and transmitting the remainder with the data. The receiver detects errors by comparing the received remainder with what it has calculated.

The following generator polynomial is used for the CRC:

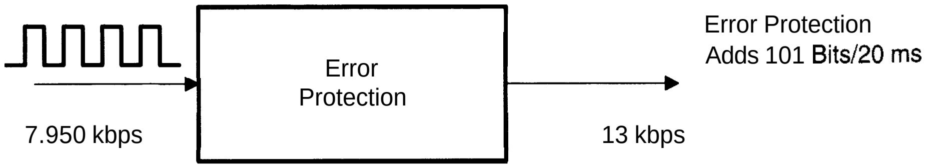
$$gCRC(X) = 1 + X + X^2 + X^4 + X^5 + X^7 \tag{1}$$

The parity polynomial, b(X), is the remainder of the division of the input polynomial by the generator polynomial as shown below:

$$a(X)*X^7 / gCRC(X) = q(X) + b(X)/gCRC(X) \tag{2}$$

where q(X) is the quotient of the division and b(x) is the remainder. The quotient is discarded, and only the parity bits identified in b(X) are encoded for transmission. To facilitate the convolutional coder, these parity bits are placed into the array of class 1 bits.

Figure 7. Error Protection Adds 101 Extra Bits per Speech Frame


Error Protection
7.950 kbps
13 kbps
Error Protection Adds 101 Bits/20 ms

In short, as shown in Figure 7, error protection adds 101 bits every 20 ms, or an additional 5050 bps.

Interleaving

As explained earlier, data from each frame is now divided and spread across two transmit slots. This is done because fading might destroy a frame, but it is unlikely that it will destroy two frames in succession. As a result, not all bits from a speech frame are lost by one bad slot. Figure 8 shows how the data is interleaved when x, y, and z are three speech frames in succession.

Figure 8. Interleaving Adjacent Frames for Error Protection

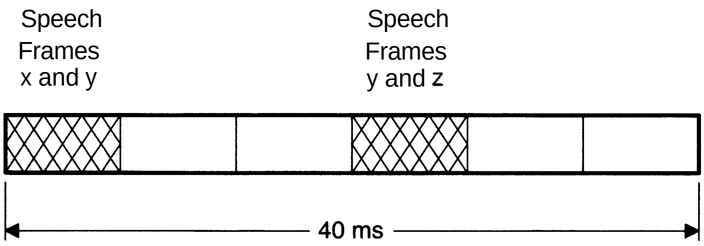


Table 4 shows how the data is interleaved when y is the current frame and x is the previous frame. Note that the speech data is entered into the interleaving array by columns.

Table 4. Interleaving of Two Adjacent Speech Frames, x and y

x0	x26	x52	x78	x104	x130	x156	x182	x208	x234
y1	y27	y53	y79	y105	y131	y157	y183	y209	y235
x2	x28	x54	x80	x106	x132	x158	x184	x210	x236
.	.	.	.	.	.	.	.		
x12	x38	x64	x90	x116	x142	x168	x194	x220	x246
y13	y39	y65	y91	y117	y143	y169	y195	y221	y247
.	.	.	.	.	.	.	.		
.	.	.	.	.	.	.	.		
.	.	.	.	.	.	.	.	.	.
x24	x50	x76	x102	x128	x154	x180	x206	x232	x258
y25	y51	y77	y103	y129	y155	y181	y207	y233	y259

The 159 bits from a speech frame are classified as class 1 and class 2 bits; data is placed into the interleaving array in such a way that class 2 bits are intermixed with class 1 bits. Class 2 bits are sequentially placed into the array and occupy the following numbered locations:

0, 26, 52, 78
93 through 129
130, 156, 182, 208
223 through 259

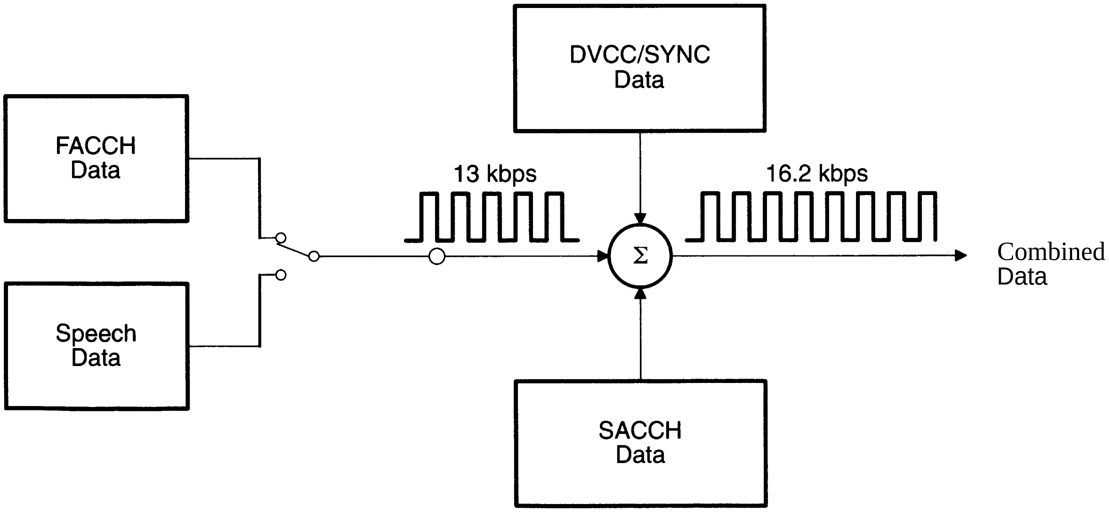
Control Signal Multiplexing

Control signal information is added to the interleaved data. Control information includes

- Slow associated control channel (SACCH)
- Fast associated control channel (FACCH)
- Digital verification color code (DVCC)
- Synchronization word (SYNC)

Figure 9 shows how all this control information is multiplexed.

Figure 9. Control-Signal Multiplexing



Slow associated control channel (SACCH) is a signaling channel in parallel with the speech path used for the transmission of control and supervisory messages between the base station and the mobile unit. SACCH messages are continuously mixed with the channel data; 12 bits are allocated for SACCH.

Fast associated control channel (FACCH) is a signaling channel for the transmission of control and supervisory messages between the base station and the mobile unit. FACCH messages are not mixed with the user information bits; they replace the user information block whenever necessary.

Digital verification color code (DVCC) is an 8-bit code that is sent by the base station to the mobile unit and is used to generate coded digital verification color code (CDVCC). CDVCC is a 12-bit field that includes the 8-bit DVCC; CDVCC is sent in each slot from the base station to the mobile unit and vice versa. The CDVCC is used by the receiver to distinguish the current traffic channel from traffic cochannels.

Synchronization word (SYNC) is a 14-symbol field that is used for slot synchronization, equalizer training, and time slot identification.

Mobile Assisted Handoff

Mobile Assisted Handoff (MAHO) is a new feature of IS-54. The base station can command the mobile unit to perform signal quality measurements on the current forward channel and any other 12 forward channels. The mobile unit can measure two quantities:

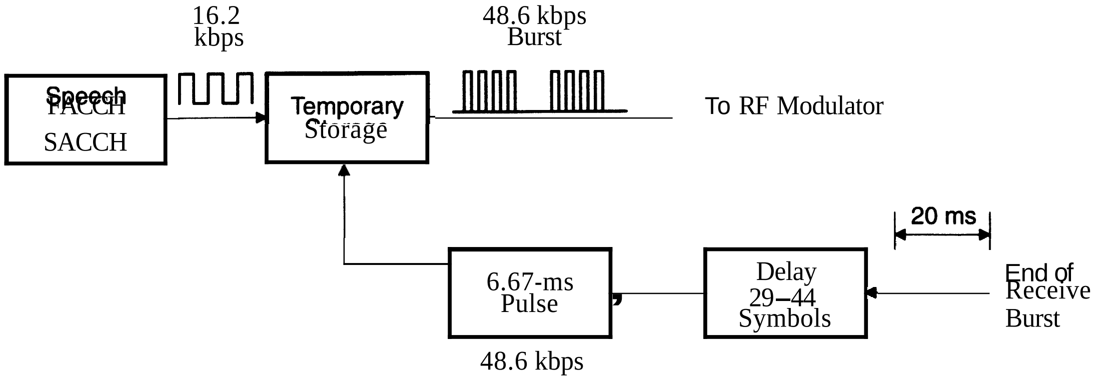
1. Received signal strength indicator (RSSI), which is a measure of the signal strength expressed in dB.
2. Bit error rate (BER), which is an estimate of the bit error information obtained by measuring the correctness of the data stream at the input to the mobile unit's channel decoder.

These channel quality measurements (RSSI and BER) are sent to the base station to assist it in handoff. This reduces the overhead on the base station. RSSI and BER are usually sent via SACCH, although they could be sent via FACCH during discontinuous transmission (DTX). DTX is a mode of operation in which a mobile unit transmitter autonomously switches between two transmitter power levels while the mobile unit is in the conversation state on an analog voice channel or a digital traffic channel.

Burst Generator

After the data has been compressed and error-protected, the bit stream is compressed (in time only) into a burst format. Burst timing offsets may be applied to facilitate dynamic time alignment. Figure 10 shows how the data is compressed and time-aligned to allow the data to be sent using one-third of the 48.6-kbps channel.

Figure 10. Burst Generator



Transmitter π/4 DQPSK Modulator and RF Amplifier

The 48.6-kbps data is now input to a *differential quaternary phase-shift keying* (DQPSK) modulator. This phase modulator groups two bits at a time to create a symbol. This results in four levels of modulation, as shown in Figure 11. Hence, the name quaternary. The term *differential* is used because symbols are transmitted as relative phase changes, rather than absolute phase values.

Figure 11. A 4-Level Modulator Groups Two Bits to Form a Symbol

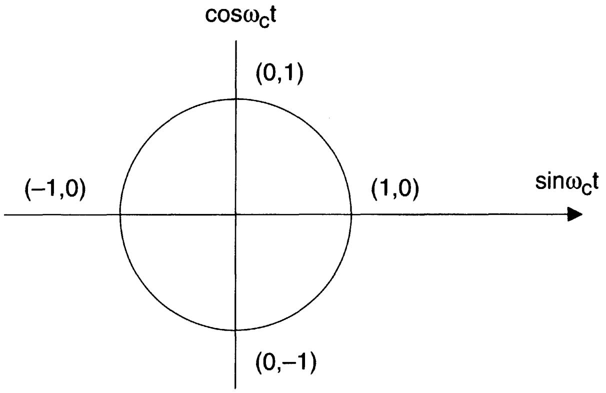


Figure 11 shows that for certain transitions, the origin will have to be crossed. This implies that the power envelope at the decoder will be 0 when the origin is crossed; this can have an undesired impact on the filters. To alleviate this, a π/4 scheme is used. This is shown in Figure 12. The transitions in this scheme are either +/–45 degrees or +/–135 degrees, and the origin is never traversed in transition from one state to another. This results in eight points on the circle, as shown in Figure 12.

Figure 12. π/4 Differential Quaternary PSK Modulator States

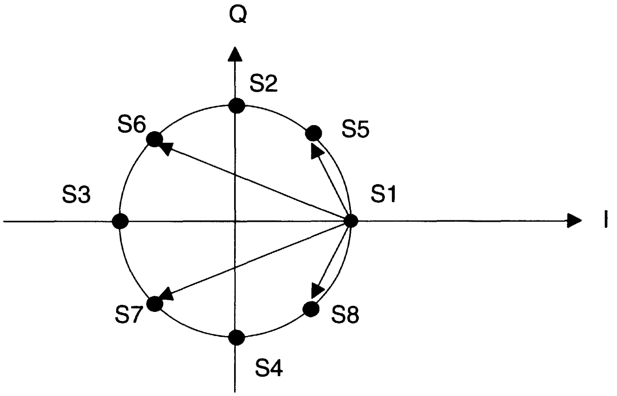
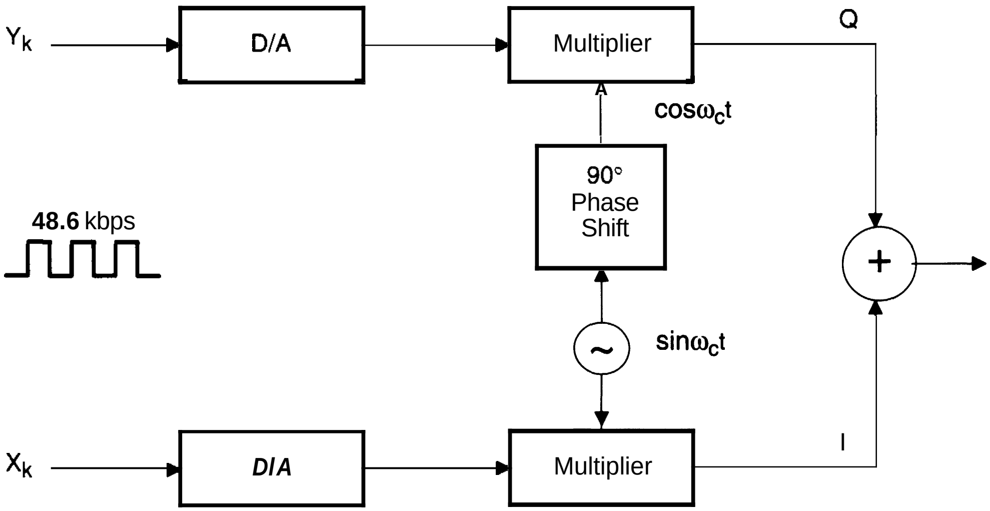


Figure 13 shows how the input serial data is now presented as 2-bit parallel data and is supplied to the multipliers after digital-to-analog conversion. Since two digital-to-analog converters (DACs) are needed, they are sometimes referred to as dual DACs. Binary signals vary the phase-shifted signals via the multipliers. Filters limit the impulse response of the binary signals to ensure that the RF carrier occupies the allocated bandwidth. The two signals are then summed together to form the final phase-shifted carrier. The conversion from baseband to RF (that is, frequency translation of the modulated carrier) is typically carried in several stages in order to reach the 800-MHz range.

Figure 13. π/4 DQPSK Modulator

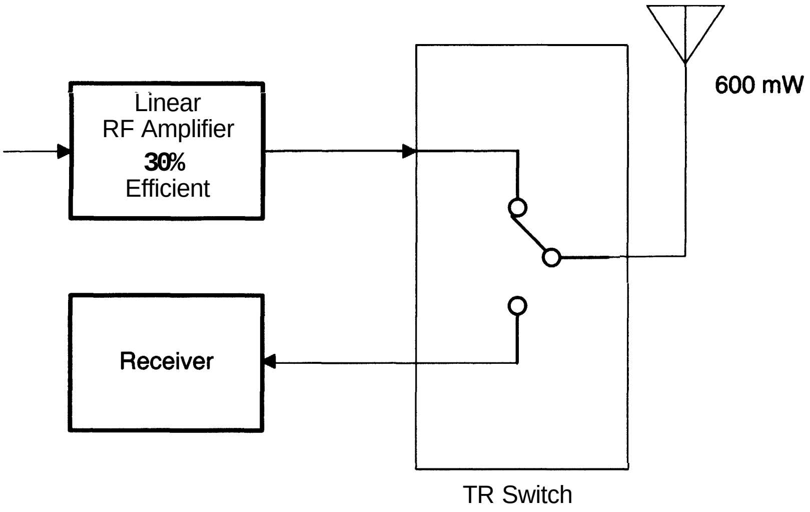


Y_k
D/A
Multiplier
Q
$\cos\omega_c t$
90°
Phase
Shift
48.6 kbps
+
~
$\sin\omega_c t$
X_k
D/A
Multiplier
I

RF Amplifier

The RF amplifier boosts the RF-modulated signal to output levels, as specified by the base station. Unlike analog transmission, which uses FM, the RF amplifier for DQPSK carrier must be linear. In FM, class C push-pull nonlinear amplifiers are used for amplification purposes. These nonlinear amplifiers are efficient (about 50%) in order to conserve power, However, nonlinear amplifiers cannot be used in DQPSK, because they would cause phase distortion. Linear amplifiers used for DQPSK are less efficient (30%). Figure 14 shows an RF amplifier.

Figure 14. Linear RF Amplifiers Are Needed for IS-54 Cellular Phone



Linear
RF Amplifier
30%
Efficient
600 mW
Receiver
TR Switch

While a duplexer is required for the analog section of the dual-mode phone, it is not required for the digital portion, because in this case the transmitter and the receiver do not operate simultaneously. A simple PN switch is enough to isolate the receiver from the transmitter, allowing the duplexer to be removed from the digital portion. Removing the duplexer has added benefits: when DQPSK signals are passed through a duplexer, a phase distortion occurs because of group delay; in addition, there is some power loss, which, in turn, requires a higher-rated power amplifier. Hence, removing the duplexer reduces the rating on the power amplifier, which extends the battery life of the mobile unit.

Receiver

The receiver functions in the following order:

1. Amplifies the received radio signal
2. Superheterodynes the RF signal to a lower workable frequency range
3. Demodulates the signal
4. Equalizes or compensates to mitigate the effects of distortions introduced by the radio channel
5. Detects errors
6. Decodes the speech signal
7. Converts it back into analog form and eventually feeds it to a speaker

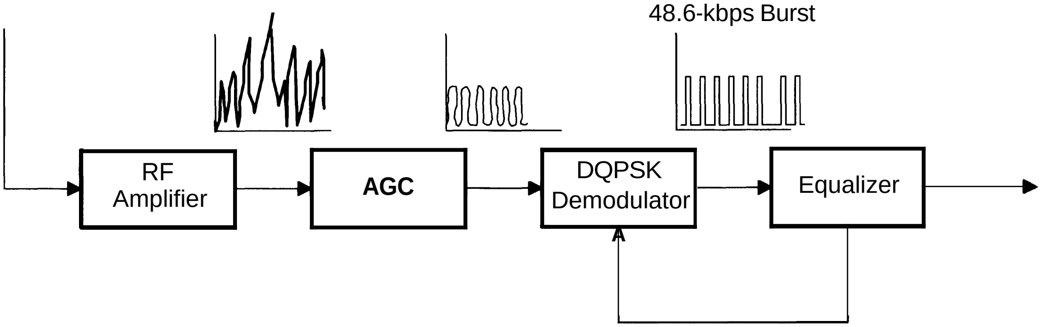
The receiver consists of several functional components:

- Receiver RF amplifier
- Mixer section
- Demodulator
- Channel decoder
- Speech decoder

Receiver RF Amplifier

This section of the receiver amplifies the low-level DQPSK RF carrier, which could be as weak as a few picowatts (–116 dBm). The RF amplifier increases this weak RF signal to a workable range before feeding it to the mixer section. The receiver RF amplifier is a broadband RF amplifier, which has a variable gain controlled by an automatic gain controller (AGC). The AGC compensates for the large dynamic range of the received signal, which is approximately 70 dB. The AGC also reduces the gain of the sensitive RF amplifier so that as the input signal increases, no distortions due to overdriving the receiver occur. Figure 15 shows the RF portion of the receiver.

Figure 15. RF Portion of Receiver Section of Dual-Mode Cellular Phone



Mixer

The frequency of the received carrier is in the range of 869–894 MHz. It is not cost-effective to directly demodulate this RF signal at this frequency range. Typically, the received signal is stepped down to a lower frequency, called the intermediate frequency (IF), by mixing it with a local oscillator (refer to Figure 2). The oscillator source may be varied so that the IF is a constant frequency, which simplifies the IF amplifier design. Typically, a second mixer superheterodynes the first IF with another oscillator source to produce a much lower frequency than the first IF. *A lower frequency enables the design and use of narrow-band filters.*

Demodulator

A DQPSK demodulator extracts data from the IF signal. Typically, a local oscillator with a 90-degree phase-shifted signal is used. The demodulator determines which decision point the phase has moved to; it then determines which symbol is transmitted by calculating the difference between the current phase and the last phase (note that the transmitter is a differential modulator).

Once the symbol has been identified, the next step is to decode the two bits. However, due to noise, Doppler effects, and Rayleigh fading, the signal must be compensated or equalized. Fading occurs when the same RF signal arrives at the receiver at different times because of multiple paths caused by reflections. The Doppler effect is caused by the motion of the transmitter relative to the received signal. The Doppler effect causes the received frequency to vary in proportion to the speed at which the mobile unit is moving; this implies that the equalizer section of a personal communication systems (PCS) unit need not be as complex when it is traveling at pedestrian speeds as when it travels at higher vehicular speeds.

Equalizer

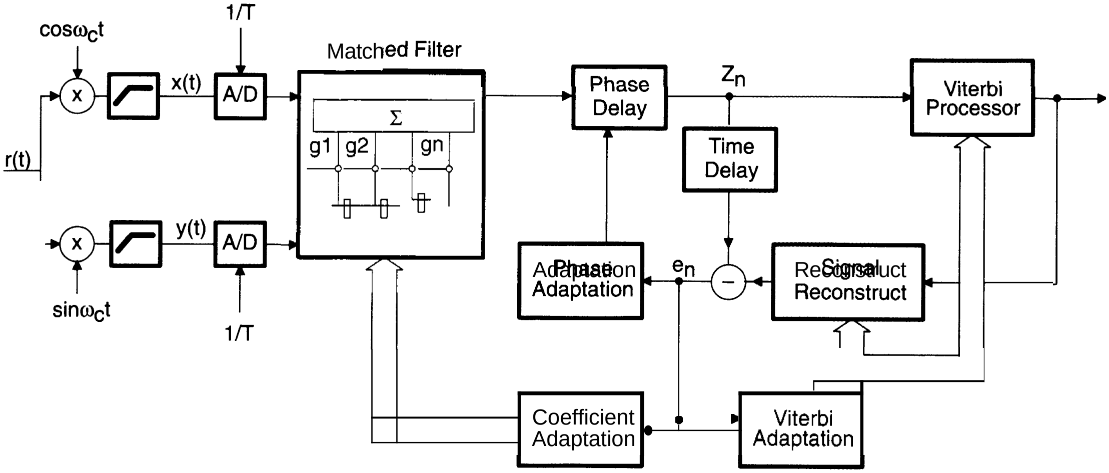
The equalizer is effectively an inverse filter of the channel distortion. Since the RF channel is not constant (as a wireline channel is assumed to be), it is necessary to track or adapt to the changing RF channel. Hence the name *adaptive equalizer.*

The IS-54 specification does not recommend a specific equalizer algorithm. At present, two classes of equalizers are popular:

- The decision feedback equalizer (DFE)
- The maximum likelihood sequence estimator (MLSE)

Figure 16 shows an example MLSE adaptive equalizer [4]. It operates adaptively in a training mode at the beginning of each burst, as well as in a tracking mode during message detection. It includes a matched filter and a modified Viterbi processor. The equalizer in Figure 16 is used by the European GSM system but is similar to the ones used in North America.

Figure 16. An MLSE Adaptive Equalizer

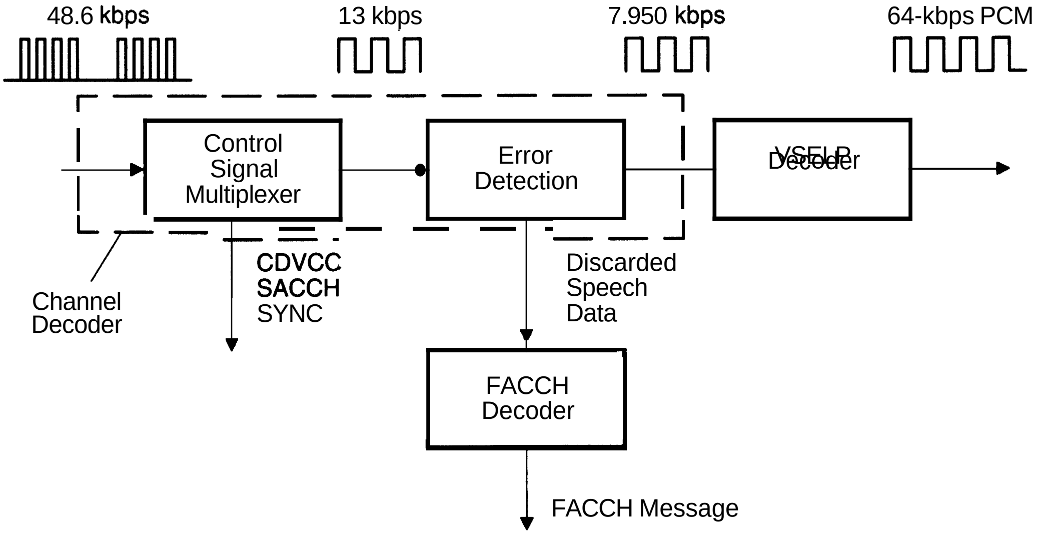

cosωct
1/T
Matched Filter
x(t)
A/D
Σ
g1
g2
gn
r(t)
y(t)
A/D
sinωct
1/T
Phase Delay
Zn
Time Delay
Viterbi Processor
Phase Adaptation
en
Signal Reconstruct
Coefficient Adaptation
Viterbi Adaptation

After demodulation and low-pass filtering of the received signal, the components x(t) and y(t) are sampled and A/D is converted, with a sampling frequency equal to the bit rate. Then the signal samples are filtered through a digital N-tap transversal filter, which approximates the matched filter (MF) shown. Theoretically, an MF makes the receiver insensitive to the carrier and clock phases used to demodulate and sample the received signal, provided that the MF coefficients are properly adjusted and the time span of the MF is long enough to include all the channel impulse responses. To this end, you must choose the number of taps, N, in the MF to comply with the maximum number of echo delays that you expect to observe in the operational environment. Note that the modulator output pulses are spread over three bit periods. Typically, $N = 6$ seems to suffice. The MF output samples are finally processed according to the modified Viterbi processor, which operates on a number of states $S = 2^N - 1$. The complexity of the Viterbi processor varies exponentially with respect to N.

Channel Decoder

The channel decoder detects errors in the bit stream, demultiplexes the control data, and feeds the data to the speech decoder. This is shown in Figure 17. If errors are detected, a masking strategy, explained in Bad *Frame-Masking Strategy* on page 28, is applied.

Figure 17. Channel Decoding and Speech Decoding


48.6 kbps
13 kbps
7.950 kbps
64-kbps PCM
Control Signal Multiplexer
Error Detection
VSELP Decoder
Channel Decoder
CDVCC
SACCH
SYNC
Discarded Speech Data
FACCH Decoder
FACCH Message

The channel decoder works in the following stages:

1. Control signal demultiplexer
2. Error detector

Control Signal Demultiplexer

Speech, SACCH, FACCH, and DVCC data signals from the demodulator are demultiplexed to separate the various signaling information. SACCH and DVCC data are simply demultiplexed by directing the dedicated bits from each burst to their control-processing locations. Speech and FACCH demultiplexing is, however, more challenging. Since FACCH data may replace speech data at any time, FACCH data is extracted by first attempting to detect errors in speech data. If the CRC appears to be correct as decoded for a speech slot, the data is routed to the speech codec section. When the CRC is in error, the data is then decoded as a FACCH message. If the CRC appears to be correct, this FACCH message is routed to its call-processing location.

Error Detector

DVCC words are error-detected, compared to the assigned DVCC to determine cochannel interference, and sent to the transmit section to be echoed back to the base station.

The channel decoder provides BER information and RSSI when commanded by the base station. This feature is called MAHO, which is discussed in the *Mobile Assisted Handoff* section on page 21.

Bad Frame-Masking Strategy

The bad frame-masking strategy is based on a 6-state machine. On every decode of a speech frame, the state machine can change states. State 0 occurs most often and implies that the CRC comparison was successful. State 6 implies that there were at least six consecutive frames that failed the CRC check. The action taken at each of these states varies as well. At state 0, no action is taken. States 1 and 2 are simple frame repeats. States 3, 4, and 5 repeat and attenuate the speech. State 6 completely mutes the speech. A detailed description of the action corresponding to each state follows:

- *State 0*: No CRC error is detected. The received decoded speech data is used.
- *State 1:* A CRC error detected. Parameter values R(0) and the LPC bits from the last frame that was in state 0 are repeated. The remaining decoded bits for the frame are passed to the speech decoder without modification.
- *State 2:* Identical to the action for state 1.
- *State 3:* Similar to the action for state 1, except that the value for R(0) is modified. A 4-dB attenuation is applied to the R(0) parameter: that is, if R(0) of the last state 0 frame is greater than 2, then R(0) is decremented by 2 and repeated at this lower level.
- *State 4:* Similar to state 3. A further attenuation by 4 dB is applied to R(0) so that the level is as much as 8 dB from the original value of R(0).
- *State 5:* Similar to 4. R(0) is further attenuated by 4 dB.
- *State 6:* The frame is repeated; but this time R(0) is cleared to 0, totally muting the output speech. Alternatively, comfort noise could be inserted in place of the speech signal.

Speech Decoder

The speech decoder, VSELP, converts the 7950-bps input data stream into 64-kbps PCM data. In poor radio conditions, the performance of VSELP has been shown to be superior to analog cellular. This is primarily due to the error-protection and error-detection capabilities that are made possible by digital techniques.

When speech frames are lost because of errors and are not correctable, the speech coderrepeats the previous frame information. If the number of consecutive lost speech frames increases, a gradual muting is applied. Thus, gaps are filled by using the characteristics of the human ear.

When the user data is not speech, but computer or facsimile data, then the speech decoder is bypassed.

Adaptive Spectral Postfilter

The perceptual quality of the synthetic speech can be enhanced by using an adaptive spectral postfilter as the final processing step. The form of the postfilter is

$$H(z) = \frac{1 - \sum_{i=1}^{10} n_i z^{-i}}{1 - \sum_{i=1}^{10} v^i \alpha_i z^{-i}} \quad 0 \le v < 1$$

α_i = *Coefficient of synthesis filter*

Audio Interface

The output of the speech coder, a 64-kbps bit stream, is input to the audio interface, which consists of the following stages:

1. Digital-to-analog conversion
2. Reconstruction filter
3. Receive-level adjustment

The reconstruction filter minimizes the step transients caused by the D/A converter. The receive-level sensitivity is defined so that a value of **24** in the R0 field, the frame energy, causes an acoustic level of at least 97 dB at the transducer when measured by an artificial ear. R0 equal to **24** represents the average frame energy during a frame, which is **18** dB down from full scale.

Summary

This report presents a brief functional overview of a digital cellular mobile station. Emphasis is given to the algorithmic description and implementation aspects of each function. The main purpose of this paper is to provide a general introduction to various functional blocks. Refer to the other papers in this book for a detailed implementation description of the individual functions.

References

1. ***Cellular System: Dual-Mode Mobile Station – Base Station Compatibility Standard,*** **IS-54** Project Number **2215,** Electronics Industries Association, December **1989.**

2. **Pawate**, B.I., "Wireless Communication: A Systems Perspective", Texas Instruments (internal document), **1992.**

3. Lin, S., et al., ***Error Control Coding,*** Prentice-Hall, **1983.**

4. Avella, R.D., et al., "An Adaptive MLSE Receiver for TDMA Digital Mobile Radio", IEEE ***Journal on Selected Areas in Communications,*** Vol. ***7*, **pp. **122–129,** January **1989.**

IS-54 Simulation

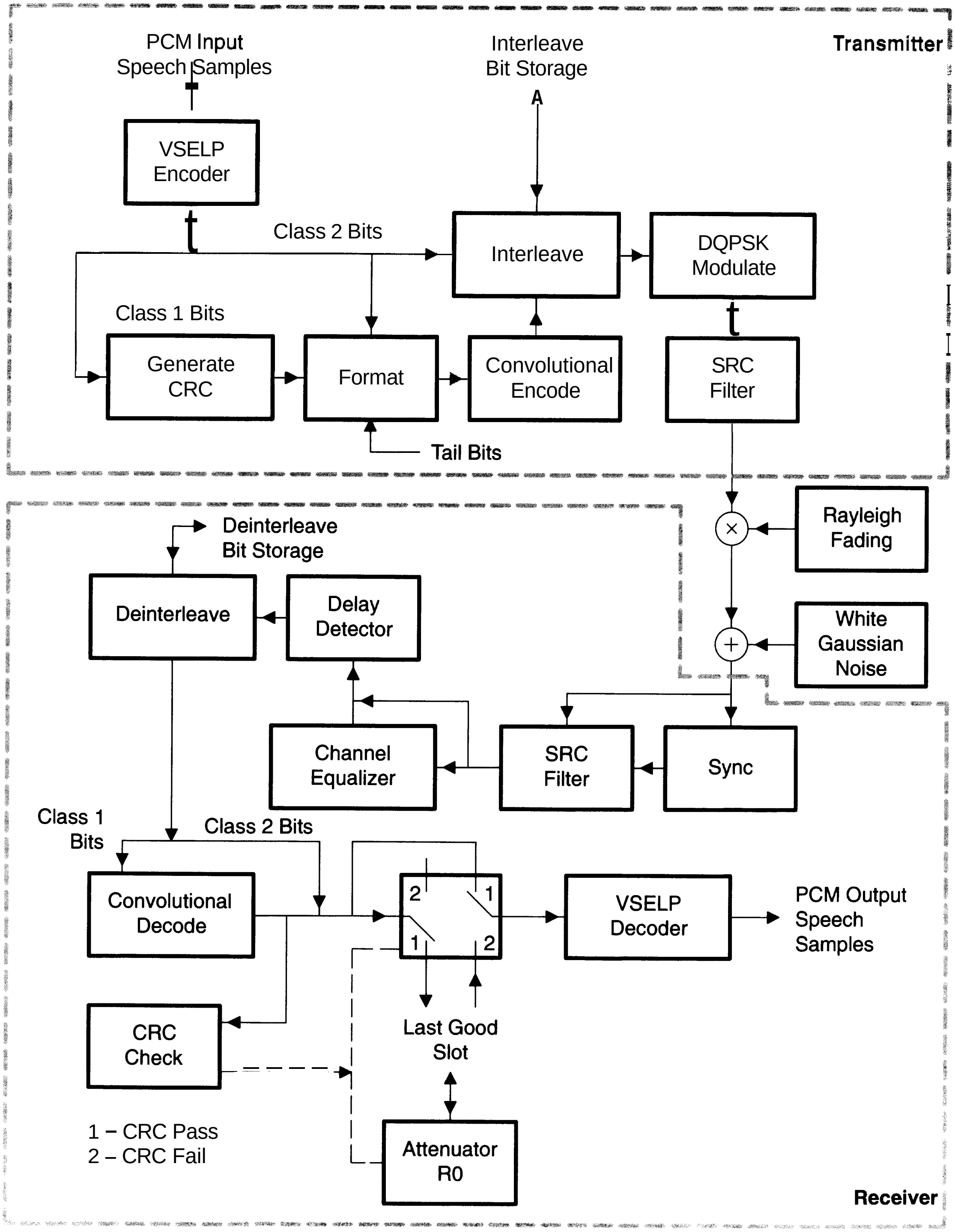
John D. Crockett
Elliott D. Hoole
Thomas Labno
Stephen Popik
Wireless Communications Systems — Semiconductor Group
Texas Instruments Incorporated

Introduction

This paper describes a C language simulation of both the transmit and receive baseband processing for a digital cellular telephone that meets the U.S. digital cellular standard (IS-54B). This simulation is needed for two reasons: first, to gain greater understanding of the IS-54 digital cellular standard and the associated digital signal processing required in a terminal that meets this standard with a vision toward efficient implementation on the TMS320 DSPs; second, to gain the capability to evaluate the effect of bit errors on the speech coder (vector sum excited linear prediction, or VSELP) and IS-54 control functions. This necessitated development of a simulation of the IS-54 processing and RF channel. See Figure 1.

The IS-54 standard separates the data bits into class 1 bits and class 2 bits. The class 2 bits are not protected and have less influence on the speech coder than class 1 bits. The class 1 bits are convolutionally encoded so that errors can be detected and corrected. In addition, a cyclic redundancy check (CRC) is calculated on the 12 class 1 bits designated as most perceptually significant. The CRC is also convolutionally encoded for error detection and correction and is used to signify noncorrectable errors in the most perceptually significant 12 bits for special error handling provisions. Consequently, the evaluation of the effect of bit errors on the voice coder must encompass all IS-54 transmit and receive processing functions.

Figure 1. IS-54B Simulation Processing Block Diagram


Transmitter
PCM Input
Speech Samples
Interleave
Bit Storage
VSELP
Encoder
Class 2 Bits
Interleave
DQPSK
Modulate
Class 1 Bits
Generate
CRC
Format
Convolutional
Encode
SRC
Filter
Tail Bits
Rayleigh
Fading
White
Gaussian
Noise
Deinterleave
Bit Storage
Deinterleave
Delay
Detector
Channel
Equalizer
SRC
Filter
Sync
Class 1
Bits
Class 2 Bits
Convolutional
Decode
2
1
1
2
VSELP
Decoder
PCM Output
Speech
Samples
CRC
Check
Last Good
Slot
Attenuator
R0
1 – CRC Pass
2 – CRC Fail
Receiver

Description

The IS-54 simulation starts with input speech parameters that are organized into 20-millisecond frames. Each frame is processed through the transmit path, the channel simulation, and the receive path.

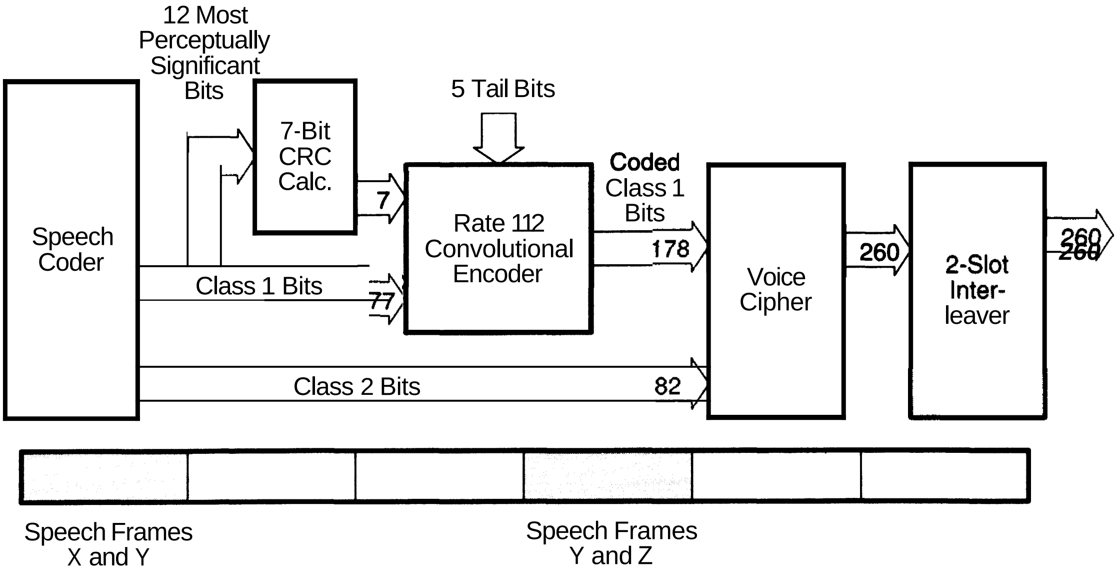
Transmit Path

A block diagram of the IS-54 simulation is shown in Figure 2. The speech data is read into the simulation from an input speech file. This file is binary pulse-code-modulated 16-bit data. The VSELP encoder is the Motorola standard, which is available from the TIA. The VSELP encoder and decoder are not incorporated into this simulation but are run as a separate program. The output from that program is fed to this simulation, whose output is then used to create final PCM speech data. From the output of the VSELP encoder, the most perceptually significant bits of the encoded speech frame are packed into a binary word for generation of the CRC. The CRC is calculated by first multiplying the input word by 2^7 and dividing by a polynomial given in IS-54 as:

$$g_{crc}(X) = 1 + X + X^2 + X^4 + X^5 + X^7 \tag{3}$$

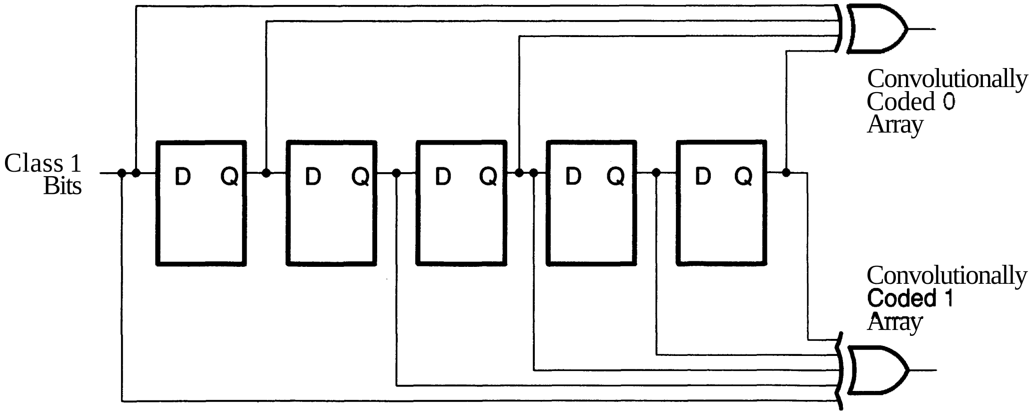
The quotient is discarded and the 7-bit remainder is kept.

Figure 2. IS-54 Error Encoding and Interleaving


12 Most Perceptually Significant Bits
Speech Coder
7-Bit CRC Calc.
7
5 Tail Bits
Rate 1/2 Convolutional Encoder
Class 1 Bits
77
Coded Class 1 Bits
178
Voice Cipher
260
2-Slot Inter-leaver
260
Class 2 Bits
82
Speech Frames X and Y
Speech Frames Y and Z

The CRC, along with the other class 1 bits (IS-54 Table 2.1.3.3.3.4–2) from the VSELP data, is packed into the cl1 array [1] to be encoded for forward error correction. The forward error correction is a rate 1/2 convolutional encoder with an initial state of 0x00. This encoder produces two output bits for each bit input. The last five bits fed into the convolutional encoder are tail bits of state 0 to force the encoder to also return to the zero state. A block diagram of the convolutional encoder is show in Figure 3.

Figure 3. IS-54 Convolutional Encoding Block Diagram



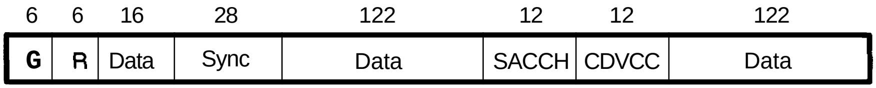
The output from the convolutional encoder, arrays cc0 and cc1 (IS-54 para. 2.1.3.3.3.4), are then packed into a 260-bit slot data array along with the class 2 bits (IS-54 Table 2.1.3.3.4–1). During this packing, the bits are shuffled around within the slot to minimize the probability that a burst error would affect more than one bit in the same vocoder parameter. This is shown in Figure 2 as voice cipher. The 260-bit slot data array is then interleaved with data from the previous frame so that the resultant transmitted burst consists of bits from both the current and previous frames. This interleaving of data across two transmit slots is designed to randomize the burst error across the data bits, thus increasing the probability that bit errors will be detectable and correctable.

The data, which consists of speech and redundant error correction information from two frames, is then formatted to the IS-54 slot format. See Figure 4. This consists of inserting the sync word, the SACCH data, the CDVCC field, and the reserved bits (the CDVCC and the reserved bit fields are filled with 0s in this simulation). The base-to-mobile format is used in order to focus on the processing stream in the handheld terminal.

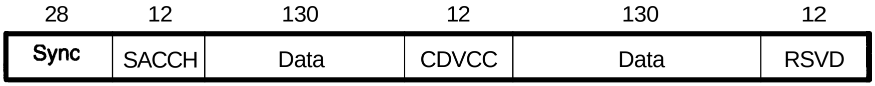
The IS-54 standard specifies the modulation as $\pi/4$ differential quadrature phase shift keying (DQPSK). The input data is paired into dibits, allowing for four symbols that specify a phase change from the previous point on the complex plane. Each dibit corresponds to an odd multiple of $\pi/4$ phase change resulting in an 8-point modulation constellation. These eight points are also called maximum-effect points.

The next function in the transmit path is the square-root-raised-cosine (SRC) filter. This filter is employed on both transmit and receive sections for the composite effect of a raised-cosine filter transfer function. This results in a filter response with nulls at the adjacent symbols in order to minimize intersymbol interference. The transmit SRC filter also includes 4X interpolation. The overall filter response is split between the transmit and receive sections to allow more efficient use of bandwidth due to partial response signaling.

Figure 4. IS-54 Slot Formats



Slot Format: Mobile Station to Base Station



Slot Format: Base Station to Mobile Station

G – Guard Time
R – Ramp Time
Data – User Information or FACCH
RSVD – Reserved (Set to 0)

SACCH – Slow Associated Control Channel
CDVCC – Coded Digital Verification Color Code
Sync – Synchronization and Training

Channel Model

At this point in an actual IS-54 handset, the data would then be input to the RF stage for modulation of the carrier frequency. Because this is a simulation, we chose to substitute simulated fading and noise generation for the transmit and receive RF portions of the IS-54 processing chain.

In a mobile radio environment, signals from many paths combine at the antenna. Depending on the relationship between the phase angles of the signals, the effect of the combination is interference that can be constructive or destructive. As the mobile radio moves, the relationship between the phase angles changes, causing the signals to be combined randomly and providing a challenge for receiver and system designers. The term for this effect is fading, and because the magnitude of the result occurs in a Rayleigh distribution about the mean value, it is called Rayleigh fading.

A simulator for generating Rayleigh fading was proposed by W. C. Jakes [4]:

```
N = 34;                 /* number of simulated signals */
NO = 0.5 *(N/2 – 1);    /* number of oscillators */
alpha = PI / 4;
V = 55;                 /* vehicle speed in MPH */
Fc = 850.0E+6;          /* carrier frequency */
lambda = 3.0E+8 / Fc;   /* carrier wavelength */
wm = 2 * PI * V / lambda;
xc(t) = sqrt(2)*cos(alpha)*cos(wm*t);
xs(t) = sqrt(2)*sin(alpha)*cos(wm*t);
for (n = 1 ; n <= NO ; n++)
{
    wn = wm * cos(2*PI*n/N);
    xc(t) += 2*cos(PI*n/N0)*cos(wn*t);
    xs(t) += 2*sin(PI*n/N0)*cos(wn*t);
}
```

xc(t) is the in-phase (cosine) component, and xs(t) is the quadrature (sine) component. This model provides a very good approximation of theoretical behavior and is excellent for general use.

Another major impairment to wireless communications is within the radio itself. As the received signal gets weaker, the signal-to-noise ratio decreases, and errors caused by thermal noise in the radio receiver can occur. This noise is characterized by a zero-mean, Gaussian probability density function in the time domain. In the frequency domain, the power spectral density of thermal noise is constant and is called white noise. In a real system, there are filters that limit the bandwidth of the noise, but the power spectral density of the noise is still constant in the filter passband, so it can still be called white. In the receiver, the noise is added to the received signal and is therefore termed additive white Gaussian noise (AWGN).

In the simulation, a Gaussian noise generator is used that generates noise of unit variance and then is scaled to the variance required for the desired signal-to-noise ratio.

Receive Path

The receive path (receiver) is also shown in Figure 1. Raised-cosine-filtered samples are fed into the sync detector, which looks for the sync word that occurs at the beginning of the slot. The sync detector looks for this sync word over a 4-symbol window, starting two symbols prior to the expected sync point. When the data matches the proper slot sync word, the data is fed into the SRC filter. This filter is the same as the transmit chain SRC filter described on page 36, except that the receive filter performs 4X decimation.

After it is fed through the SRC filter, the data is input to a channel equalizer. As shown in Figure 2, the channel equalizer can be turned either on or off under command of the cellular base station. The channel equalizer is not included in this simulation and is the subject of a separate paper [9].

The delay detection process, also called differential decoding, is the inverse of the differential encoding process in the transmitter. The delay detector computes the amount of phase change between two successive raised-cosine-filtered maximum-effect points. This can be shown easily with the exponential notation for complex numbers. Let A*exp(j*PI/2) be the current point and B*exp(j*PI/4) be the previous point. Now multiply the current point by the complex conjugate of the previous point:

$$A * \exp(j * PI/2) * B * \exp(-j * PI/4) = A * B * \exp(j * PI/4)$$

The result is an exponential whose angle is the phase change between the previous and the current points. Because it is the phase change that contains the information bits, the magnitude can be disregarded.

The deinterleave function recombines a frame of speech data from data received from two consecutive receive slots. As discussed in the transmit chain description, the data is interleaved to minimize susceptibility to burst errors. At this time, the data is divided back into the encoded class 1 (cc0 and cc1) bits and the unprotected class 2 bits. The class 1 bits are then fed into a convolutional decoder while the unprotected class 2 bits are held to recombine with the class 1 bits once decoded.

The convolutional decode is performed via the Viterbi algorithm. A two-dimensional array is built that is 89 (the number of bits input to the encoder) columns wide and 32 (the possible number of states of the encoder) rows high. This algorithm calculates the probability of possible paths through the array (which represent the sequence of states through which the encoder would have passed). This probability is added to the cumulative probabilities for each of the possible preceding states to give a cumulative probability for a given trellis position. Then, given that the beginning and ending states of the convolutional encoder are 0 (0 is the initial state and five tail bits of 0 force it back to state 0), the path of maximum probability is selected by tracing through the array from ending state to beginning state. With the path through the trellis known, the input bits are easily obtained. The path of maximum probability should produce the original encoded bit stream, even in the presence of low bit errors.

The CRC value and the 12 most perceptually significant bits are extracted from the decoded class 1 bits. A CRC is recalculated on these 12 bits and compared against the received CRC. This is done to detect the presence of errors in these 12 bits. If the CRCs match, the received VSELP speech parameters are sent to the VSELP decoder. If they do not match, a state machine (IS-54 para. 2.2.2.2.3.2.) is employed for handling the errors. This state machine stores the last good set of speech parameters for use in cases of repeated CRC errors. The received speech parameters are then fed into the VSELP decoder for speech synthesis.

Using the Simulation

One of the goals in developing this simulation was to ensure that it is portable across different computing platforms. To this end, every attempt was made to use only ANSI-C compatible calls and syntax. The code was originally developed using Borland C++ 3.1 running on 486133 ISA PCs. It was tested and modified to make it compatible with the Microsoft Visual C++ 1.0 and Zortech C++ 3.0 compilers, which support ANSI-C compliance.

To run the simulation, a command file, IS54SIM.PRM, is utilized to pass all required information to the program. Additionally, another file, SRC_FILT.DAT, is required and contains the square-root cosine filter coefficients necessary for the simulation. These files and the simulation program must all reside in the same working directory.

The format of the command file is simple. It is an ASCII file that contains four lines:

1. The desired SNR
2. The assumed vehicle speed
3. The carrier frequency (used in Fading model)
4. The filename for the input speech data that has already been VSELP processed (This file should also be in the working directory.)

The SRC_FILT.DAT file is also an ASCII file, where each line is a coefficient used by the SRC filter.

After running the simulation (by typing the program name on the system command line), there are seven output files produced, all of which reside in the current working directory. These files are summarized below.

IS54SIM.OUT	An ASCII-Hex version of the 193-bit VSELP data recovered for each frame
RAWTXBIT.OUT	An ASCII-Hex version of the 324-bit formatted TDMA slot prior to transmission
CLTXBIT.OUT	An ASCII-Hex version of the 89 class 1 bits and 82 class 2 bits for the transmit slot
CLRXBITS.OUT	An ASCII-Hex version of the 89 class 1 bits and 82 class 2 bits recovered in the receive slot. Each line of receive data (one per slot) is appended with the current CRC error state (0–7).
RAWRXBITS.OUT	An ASCII-Hex version of the 324-bit formatted TDMA slot prior to decoding. Each line of receive data (one per slot) is appended with the current CRC error state (0–7).

By examining these output files, a user can determine the performance of an IS-54 transmission under varying levels of SNR (degradation in the channel). This program also outputs the number of received frames with valid CRC, the number of frames with invalid CRC, and the bit error rates for each field for CRC-valid frames.

The simulation was compiled and run on IBM-compatible PCs using several compilers. The simulation runs three to six slots per second on a 486DX–33MHz PC.

Code Availability

The associated program files are available from Texas Instruments TMS320 Bulletin Board System (BBS) at (713) 274-2323. Internet users can access the BBS via anonymous ftp at *ti.com*.

References

1. *Cellular System: Dual-Mode Mobile Station – Base Station Compatibility Standard*, IS-54B, Telecommunications Industry Association, April 1992.
2. Chishtie, Mansoor A., "U.S. Digital Cellular Error-Correction Coding Algorithm Implementation on the TMS320C5x", *Telecommunications Applications With the TMS320C5x DSPs,* Texas Instruments Incorporated, 1994, pp. 63–75.
3. Choong, Yong F., *Convolutional Decoder for the IS-54 Error Protected Speech Codes*, Digital Signal Processing Branch, Semiconductor Process and Development Center, Texas Instruments Incorporated, 1992.
4. Jakes, William C., Jr., *Microwave Mobile Communications,* John Wiley & Sons, New York, New York, 1974, pp. 70–76.
5. Proakis, John *G., Digital Communications,* McGraw Hill, New York, New York, 1989.
6. Choong, Yong , and Chishtie, Mansoor *A.*, *Convolutional Encoder for IS-54 Voice Channel* C program, Texas Instruments Incorporated, 1992.
7. Chishtie, Mansoor A., *Viterbi Decoder Algorithm for IS-54 SACCH Control Channel C* program, Texas Instruments Incorporated, 1991.
8. Hartman, Matt, *C Language Version of VSELP Speech Coder,* Systems Research Laboratories, Chicago Corporate Research and Development Center, Motorola, Incorporated, 1990.
9. Hoole, Elliot D., "Channel Equalization for the IS-54 Digital Cellular System With the TMS320C5x", *Telecommunications Applications With the TMS320C5x DSPs,* Texas Instruments Incorporated, 1994, pp. 177–187.

Part III
Speech Synthesis

Theory and Implementation of the Digital Cellular Standard Voice Coder: VSELP on the TMS320C5x

Jason Victor Macres
DSP Software Engineering, Incorporated

Introduction

TIA subcommittee TR45.3 has adopted vector sum excited linear prediction (VSELP) as the voice coding standard for U.S. digital cellular communications. Motorola was responsible for the design and development of the VSELP algorithm. Additionally, Motorola has kept implementation details of the VSELP algorithm proprietary. This paper explains an interoperable VSELP alternative algorithm and the implementation of this algorithm on a TMS320C5x digital signal processor. The interoperable algorithm is developed using reference [1] as a guideline.

The VSELP algorithm is a type of code excited linear predictive coding (CELP) algorithm that has been adopted as the standard for digital cellular communications. The VSELP vocoder encodes speech at a bit rate of 7950 bitslsecond. An additional 5050 bitslsecond are utilized for error protection and synchronization, bringing the total bit rate to 13,000 bitslsecond. This paper describes only the voice coding portion of the vocoder. A brief overview of the VSELP algorithm is presented for background.

Overview of VSELP

Structurally, the VSELP algorithm closely resembles the CELP algorithm. The difference lies in the form and structure of the code books. Whereas CELP uses a stochastically overlapped code book (each entry shares all but two samples with its neighboring entries), VSELP utilizes two sets of basis vectors to generate the space of candidate vectors. Thus, the stochastic code book search of CELP corresponds to two code book searches in VSELP. There are seven basis vectors for each search. Each basis vector contains 40 elements. The selection of the basis vectors is fundamental to deriving fast code book search procedures. The basis vectors chosen provide for fast orthogonalization of the entire space. By orthogonalizing each of the seven vectors with a vector V, the entire 128 (2^7) space, defined by the seven basis vectors, is also orthogonalized.

An open-loop LPC analysis is performed on a frame of speech to derive a set of LPC filter coefficients. These coefficients are bandwidth expanded for use in perceptual error weighting filters, $H(z)$ and $W(z)$, where $H(z) = 1/A(z)$ and $W(z) = A(z)/A(z/\gamma)$. The input frame of speech is filtered through the filter $W(z)$ to obtain a perceptually weighted frame of speech. The analysis by synthesis proceeds with three code books (unlike CELP, which proceeds with two). First, the adaptive code book is searched and the resulting best entry and gain are found. This entry multiplied by its gain factor is orthogonalized with the first set of seven basis vectors. Thus, the second code book search can be performed independently of the first code book search. The new set of basis vectors is used form the code book for the second search. The best entry and gain are found for this code book and orthogonalized with the second set of basis vectors. Finally, the third code book search is performed. The gains of each of the three code book searches are jointly quantized and transmitted with the three code book indices to the receiver.

The basic blocks in the VSELP coder are:

- Tenth-order LPC analysis (spectrum predictor)
- Long term (pitch) predictor
- Adaptive (pitch) code book search
- First basis vector code book search
- Second basis vector code book search
- Vector quantization of the code book gains

The primary VSELP parameters are outlined in Table 1.

Table 1. Primary VSELP Parameters

Symbol	Parameter	Value
SR	Sampling rate	8 kHz
N_F	Samples per frame	160
N_{SF}	Samples per subframe	40
N_P	LPC filter order	10
M_1	No. basis vectors (1)	7
M_2	No. basis vectors (2)	7
BWEXP	Bandwidth expansion	0.8
LTFORD	Long term filter order	1

The VSELP algorithm has been developed from references [1] and [2]. These references contain information pertaining to the high-level description of the algorithm and provide no actual implemented software (high-level or assembly).

Bit Allocations

Table 2 shows the bit allocation for the VSELP frame. The frame energy (RO) and reflection coefficients (LPC1–LPC10) are sent once per frame, while the pitch lag (LAG1–LAG4), code book indices (CODE1_1–CODE1_4, CODE2_1–CODE2_4), and gain indices (GSP0_1–GSP0_4) are sent four times per frame.

The total number of bits per 20-millisecond speech frame is 159, yielding a voice coder bit rate of 7950.

Table 2. VSELP Frame Bit Allocation

Parameter	Bits	Description
R0	5	Frame energy
LPC1	6	1st reflection coefficient
LPC2	5	2nd reflection coefficient
LPC3	5	3rd reflection coefficient
LPC4	4	4th reflection coefficient
LPC5	4	5th reflection coefficient
LPC6	3	6th reflection coefficient
LPC7	3	7th reflection coefficient
LPC8	3	8th reflection coefficient
LPC9	3	9th reflection coefficient
LPC10	2	10th reflection coefficient
LAG1	7	Lag, SF 1
LAG2	7	Lag, SF 2
LAG3	7	Lag, SF 3
LAG4	7	Lag, SF 4
CODE1_1	7	1st CB index, SF 1
CODE1_2	7	1st CB index, SF 2
CODE1_3	7	1st CB index, SF 3
CODE1_4	7	1st CB index, SF 4
CODE2_1	7	2nd CB index, SF 1
CODE2_2	7	2nd CB index, SF 2
CODE2_3	7	2nd CB index, SF 3
CODE2_4	7	2nd CB index, SF 4
GSP0_1	8	Gain index, SF 1
GSP0_2	8	Gain index, SF 2
GSP0_3	8	Gain index, SF 3
GSP0_4	8	Gain index, SF 4

Perceptual Weighting

Perceptual weighting of the input speech signal (or the error signal) improves the performance of the coder. The high-energy formant regions of the speech spectrum mask noise better than lower energy portions of the spectrum. The error signal generated by each synthesizer pass is weighted appropriately to capitalize on this perceptual effect. The filter amplifies the error signal spectrum in nonformant regions of the speech spectrum and attenuates the error signal spectrum in formant regions. Thus, an error signal whose spectral energy is concentrated in formant regions of the speech is considered better than one whose spectral energy is not located under formants.

Open-Loop LPC Analysis

Each incoming speech frame is processed through an open-loop LPC analysis to generate the filter coefficients used in the remaining portions of the algorithm. The input speech is first windowed using a Hamming window, then an autocorrelaion is performed and the result is normalized based on the energy of the first coefficient of the autocorrelation.

The autocorrelation coefficients are then windowed for bandwidth expansion and spectral smoothing using a rectangular (in frequency) window. The smoothed autocorrelations are the input to a Leroux-Guegan routine, which transforms the autocorrelation parameters into reflection coefficients. The Leroux-Guegan algorithm was chosen because it is ideal for fixed-point implementation and is very efficient.

A stability check is performed in the Leroux-Guegan algorithm by monitoring the rms value. If the rms falls below 0, the Leroux-Guegan is terminated, and the previous reflection coefficients are used. This instability can occur from ill-conditioned autocorrelation coefficients.

Interpolation

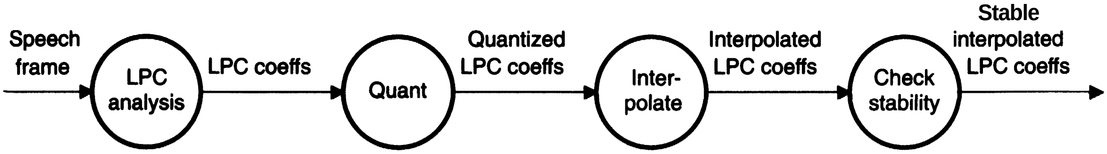
Because the reflection coefficients generated by the LPC analysis represent the spectrum of the speech for one frame centered over the fourth subframe, the coefficients for the remaining subframes are interpolated from the current and the previous frame's coefficients. The direct form-filter coefficients are linearly interpolated. The following table shows the interpolation scheme:

$a_i = (0.75)a_{i(previous)}\ (0.25)a_{i(current)}$	subframe 1 formula
$a_i = (0.50)a_{i(previous)}\ (0.50)a_{i(current}$	subframe 2 formula
$a_i = (0.25)a_{i(previous)}\ (0.75)a_{i(current}$	subframe 3 formula
$a_i = a_{i(current)}$	subframe 4 formula

Interpolating the direct form coefficients can result in an unstable filter; therefore, the resulting coefficients must be checked for stability. For the first, second, and third subframes, the filter coefficients are converted to reflection coefficients. If any of the resulting reflection coefficients' magnitudes are greater than 1, then the interpolation process has produced an unstable filter. To remedy this instability, the filter coefficients for the subframe are replaced by the uninterpolated filter coefficients. For the first subframe, the previous frame's uninterpolated filter coefficients are used. For the third subframe, the current frame's uninterpolated filter coefficients are used. The second subframe uses the uninterpolated filter coefficients from the frame (previous or current) that has the higher energy. For the case when the energies are equal, subframe 2 uses the uninterpolated filter coefficients from the previous frame.

The following data flow illustrates the procedure for quantization and interpolation of the LPC filter coefficients.

Figure 1. LPC Filter Coefficient Quantization and Interpolation


Speech frame
LPC analysis
LPC coeffs
Quant
Quantized LPC coeffs
Inter-polate
Interpolated LPC coeffs
Check stability
Stable interpolated LPC coeffs

Long-Term Predictor

The long-term filtering operation (adaptive code book search) for VSELP is similar to the general CELP long-term filtering operation. The long-term filter is given by:

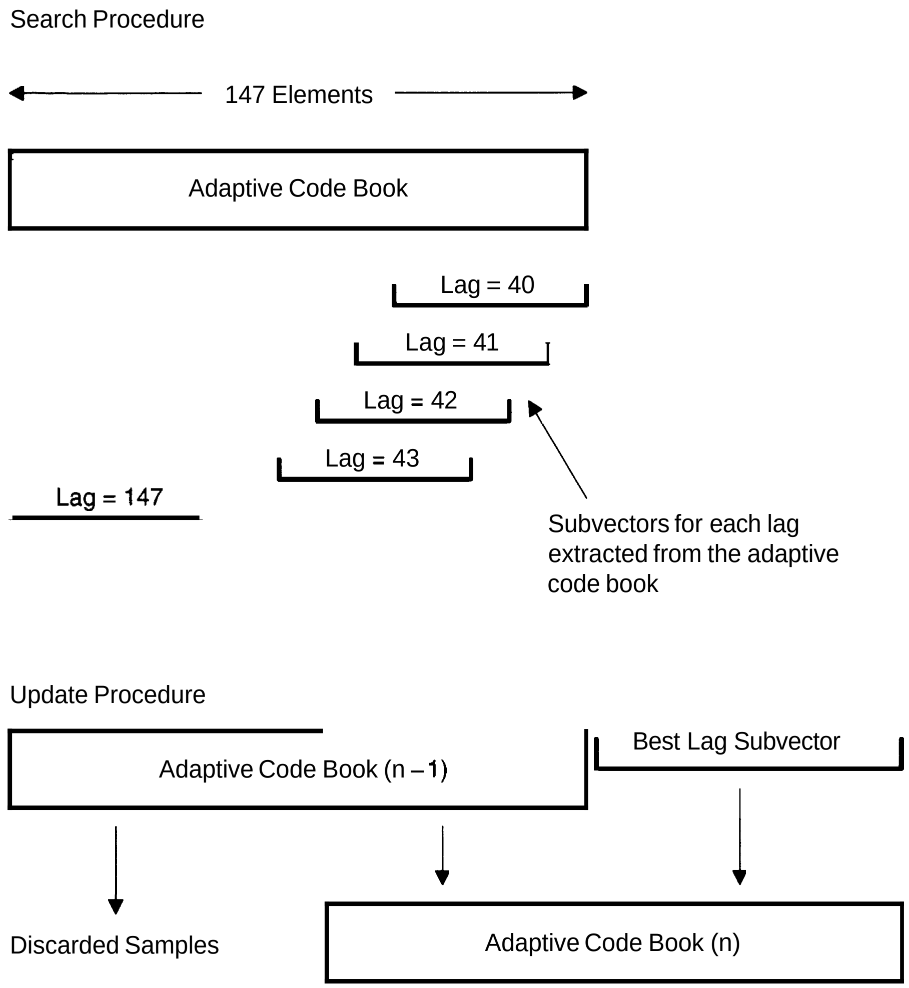
$$B(z) = \frac{1}{1-\beta z^{-L}} \tag{1}$$

To accommodate lags less than the **subframe** size (L < NSF), the equation is modified such that the filter's output is only a function of the filter state at the start of a subframe.

$$B(z) = \frac{1}{1-\beta z^{-\mathrm{flr}\left(\frac{n+L}{L}\right)L}} \tag{2}$$

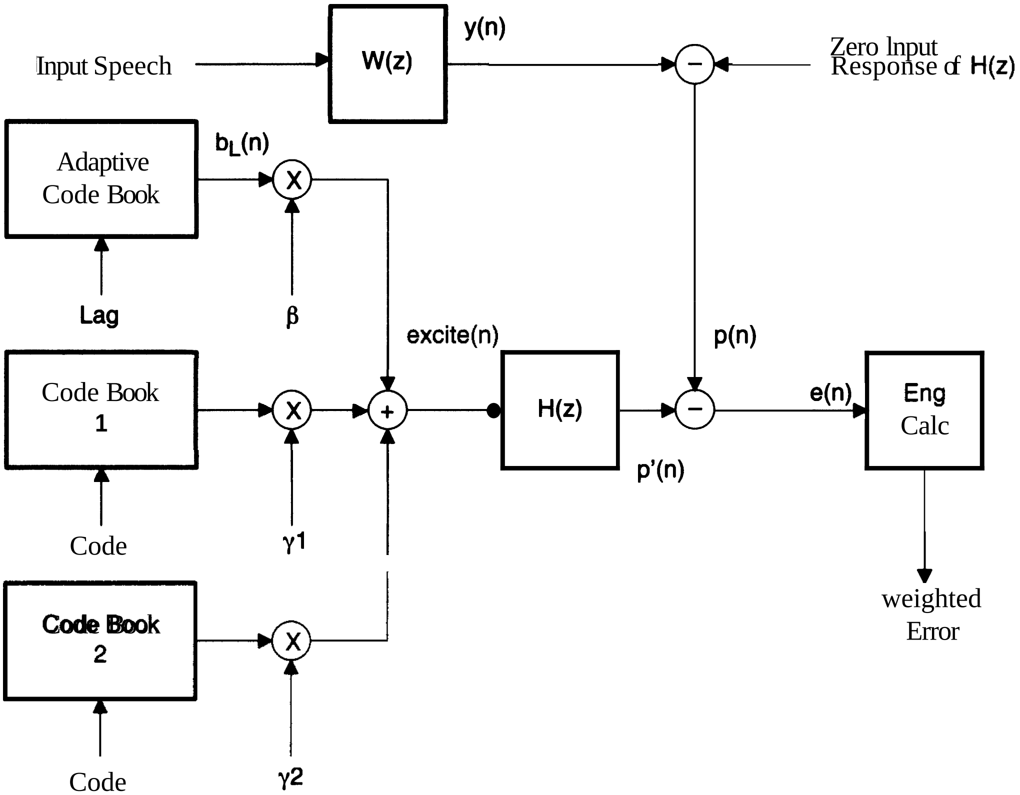
The flr(x) function truncates the fractional portion of x, returning only the integer portion of x. For L ≤ NSF, the equations are identical. For L < NSF, the flr function will evaluate to 2 when n = L, as depicted in Figure 2.

Figure 2. Adaptive Code Book Search


Search Procedure
147 Elements
Adaptive Code Book
Lag = 40
Lag = 41
Lag = 42
Lag = 43
Lag = 147
Subvectors for each lag extracted from the adaptive code book
Update Procedure
Adaptive Code Book (n – 1)
Best Lag Subvector
Discarded Samples
Adaptive Code Book (n)

In Figure 2, the portion of the adaptive code book utilized (call this subvector b_L) is of length NSF and starts at the index defined by the current lag value in the search procedure. For $L \geq$ NSF, this procedure is straightforward because the length of b_L fits (see Figure 2) inside the adaptive code book. The VSELP algorithm supports lags from 20 to 147; therefore, a special situation exists when the lag (L) is less than NSF. In this case, the b_L vector is placed such that a portion of it hangs over the adaptive code book. These elements of the adaptive code book (long-term filter state) do not exist yet. The flr function of equation [2] remedies this by doubling the lag (code book index value). This results in copying the first NSF – L elements of the b_L vector to the ending NSF – L elements.

Figure 3. Code Book Search Signal Flow


Input Speech
W(z)
y(n)
Zero Input Response of H(z)
Adaptive Code Book
$b_L(n)$
X
Lag
β
excite(n)
p(n)
Code Book 1
X
+
H(z)
−
−
e(n)
Eng Calc
p'(n)
Code
γ1
Code Book 2
X
Code
γ2
weighted Error

For each lag (20 I L ≤ 146), a vector called $b_L(n)$ of length NSF is extracted from the adaptive code book. This vector is filtered through the bandwidth-expanded LPC filter H(z). The resulting vector, $b'_L(n)$, is compared to the input vector p(n). The p(n) vector is the perceptually weighted input speech vector minus the zero-input response of H(z). The zero-input response is subtracted from the input speech to remove any of the ringing of the H(z) filter caused by the previous subframe. The b_L vector that produces the minimum mean square error (MSE) (or maximum match score) compared to p(n) is chosen as the best vector from the adaptive code book. The lag L that produced this b_L vector is transmitted to the receiver. The match score is defined as:

$$MS = \frac{(C_L)^2}{G_L} \tag{3}$$

where:

$$G_L = \sum_{n=0}^{NSF-1} (b'_L(n))^2 \tag{4}$$

$$C_L = \sum_{n=0}^{NSF-1} b'_L(n)p(n) \tag{5}$$

In digital cellular VSELP, β is restricted to positive numbers; therefore, only lags with a positive C_L are considered in the search procedure. If no lag with a positive C_L can be found, the adaptive code book is disabled. The lag is coded using seven bits, yielding 128 possible lag values. Since only 127 of these values are valid ($20 \leq L \leq 146$), one lag value is reserved to disable the adaptive code book search in the decoder. It should be noted that the gain coefficient is not coded at this time. After all three code vectors are determined, a joint optimization is performed on the three gain terms, β, $\gamma 1$, and $\gamma 2$.

Our implementation precomputes all of the correlations and energies and stores them. The temporary storing of these parameters is not strictly necessary; however, it allows us to find a scale factor so the search can be performed utilizing maximum dynamic range. Preserving dynamic range is very important for a proper pitch search.

Code Search Algorithm

Each of the two code books is constructed from a set of M basis vectors. These vectors are combined linearly to form a code book of size 2M. The code book vectors are described by:

$$u_i(n) = \sum_{m=1}^{M} \theta_{im} v_m(n) \tag{6}$$

where v_m is the mth basis vector and u_i is the ith code-book vector. The value of θ is either +1 or –1 and is formulated as follows. Each of the code book vectors, u_i, is indexed by i. If the indices are viewed in binary form, M bits are required to represent the index space. If the LSB of the index is defined as bit 1 and the MSB is defined as bit M, then θ_{im} can be defined as:

If (bit m of index i = 1)

then $\theta_{im} = +1$

If (bit m of index i = 0)

then $\theta_{im} = -1$

The following provides an example for the trivial case when M = 2. This defines a code book size of 2^2, or 4. In this case, only two basis vectors are required, namely v_1 and v_2. Each of the four code book vectors is developed below.

$$u_i = \theta_{i1} \times v_1 + \theta_{i2} \times v_2$$

$$u_0 = u_{00} = \theta_{01} \times v_1 + \theta_{02} \times v_2 = v_1 + v_2$$
$$u_1 = u_{01} = \theta_{11} \times v_1 + \theta_{12} \times v_2 = -v_1 + v_2$$
$$u_2 = u_{10} = \theta_{21} \times v_1 + \theta_{22} \times v_2 = v_1 - v_2$$
$$u_3 = u_{11} = \theta_{31} \times v_1 + \theta_{32} \times v_2 = -v_1 - v_2$$

It should be noted that $u_0 = -u_3$ and $u_1 = -u_2$. These are called complementary code book vectors, and this property is exploited in the code book search to reduce computational requirements.

The VSELP code book structure was defined above for a static single code book. The formula below expands the notation to describe a VSELP structure with multiple static code books. From equation *(6):*

$$u_{k,i}(n) = \sum_{m=1}^{M} \theta_{im} v_{k,m}(n) \tag{7}$$

For digital cellular VSELP, k = 1 or 2; that is, two static code books are used. The three code books are searched sequentially. First, the adaptive code book is searched for the optimal vector assuming yl = 0 and $\gamma 2 = 0$. The technique used in searching the adaptive code book is described above. For the stochastic code book searches, it is necessary to generate the zero-state response of each code vector to H(z). This is accomplished by filtering each of the M (M = 7) basis vectors for each code book through H(z) with the history of H(z) set to 0 prior to filtering each vector. The resulting code vectors are defined by equation (8):

$$f_{k,I}(n) = \sum_{m=1}^{M} \theta_{im} q_{k,m}(n) \tag{8}$$

where $q_{k,m}(n)$ is the zero-state response of H(z) to the basis vector $v_{k,m}(n)$.

The result of the first search is the optimal lag value and the optimal $b_L(n)$ vector. The $b_L(n)$ vector times its gain, β, represents the adaptive code book's contribution to the excitation signal. Next, the first stochastic code book is searched, given $b_L(n)$. This results in an optimal code vector and corresponding index (I) for the first code book, $f_{1,I}$. Finally, the second code book is searched given $b_L(n)$ and $f_{1,I}(n)$. This results in an optimal code vector and corresponding index (H) for the second code book, $f_{2,H}(n)$.

All of the searches in this implementation take full advantage of the ’C5x MAC instructions and are optimized for speed.

Orthogonalization of the Code Vectors

The error signal generated after each of the code vectors from each code book is selected is:

$$e(n) = p(n) - \beta b_L(n) - \gamma 1 f_{k,I}(n) - \gamma 2 f_{k,I}(n) \tag{9}$$

and

$$\text{Total weighted error} = \sum_{n=0}^{NSF-1} e^2(n) \tag{10}$$

Given $\gamma 2 = 0$ and $b_L(n)$ for the first code book search, optimal values for β, yl, and $f_{1,I}(n)$ must be found. This however, would be too computationally expensive for real-time performance. If the b'_L vector and the each of the code vectors $f_{1,I}$ are orthogonal, then yl and the code vector can be jointly optimized independent of β. By orthogonalizing each of the basis vectors to the $b'_L(n)$ vector, the entire space of code vectors is orthogonalized. The Grahm-Schmidt algorithm is used to perform this orthogonalization as follows:

$$\Gamma = \sum_{n=0}^{NSF-1} (b'_L(n))^2 \tag{11}$$

and

$$\Psi_m = \sum_{n=0}^{NSF-I} b'_L(n) q_{1,m}(n) \qquad \text{for } 1 \le m \le M \tag{12}$$

The orthogonalized, filtered basis vectors for the first code book are defined by:

$$q'_{1,m}(n) = q_{1,m}(n) - \frac{\Psi_m}{\Gamma} b'_L(n) \qquad (13)$$

The orthogonalized, filtered code vectors for the first code book are defined by:

$$f'_{1,i}(n) = \sum_{m=1}^{M} \theta_{im} q'_{1,m}(n) \quad \text{for } 0 \leq i \leq 2M-I \qquad (14)$$

The new expression for the total weighted error for the first code book search is

$$E'_{1,i} = \sum_{n=0}^{NSF} (p(n) - \gamma_1 f'_{1,i}(n))^2 \qquad (15)$$

This expression is independent of b and b_L and also assumes no contribution from the second code book. The value for the gain is computed for each code vector but is not encoded yet. As stated previously, the value for the gains of each of the vectors contributing to the excitation vector are jointly optimized after all searches are complete.

The second stochastic code book search is identical to the first except that the basis vectors for the second code book are orthogonalized to both the $b_L(n)$ vector and to the optimum code vector from code book 1, $f'_{1,I}(n)$. This orthogonalization can be performed sequentially. The filter basis vectors, $q_{2,m}(n)$, are first orthogonalized to $b_L(n)$. The resulting vectors are then orthogonalized to $f'_{1,I}(n)$.

The orthogonalized, filtered code vectors for the second code book are defined by:

$$f'_{2,i}(n) = \sum_{m=1}^{M} \theta_{im} q'_{2,m}(n) \quad \text{for } 0 \leq i \leq 2^M-1 \qquad (16)$$

The new expression for the total weighted error for the second code book search is

$$E'_{2,i} = \sum_{n=0}^{NSF} (p(n) - \gamma_2 f'_{2,i}(n))^2 \qquad (17)$$

For the implementation of the fixed-point VSELP, a modified Grahm-Schmidt algorithm was used. The difference between this Grahm-Schmidt and the one just presented is that this one is scaled by an energy constant. This scale washes out in the code book search, yet avoids an expensive division and preserves dynamic range.

Gray Code Search

In this section, a fast search procedure for finding the best code vector from the stochastic code book is developed. As with the adaptive code book search, the vector that minimizes the MSE (that is, that maximizes the match score) is sought. Note that the subscript denoting the first or second code book has been dropped for clarity. The code search procedures are identical for each code book. The match score is defined as:

$$MS = \frac{(C_i)^2}{G_i} \tag{18}$$

The search procedure calculates the match score for each vector in the code book. The best code vector (indexed by i) will have the highest match score of all code vectors in the code book. The computational requirements for one subframe search of one code book is 2 × NSF multiply-accumulates (MACS). This results in a code book search computational requirement of:

$$2 \times NSF \times 2^M(\frac{MACS}{code\ book}) \times 2(\frac{codebooks}{subframe}) \text{ x } 4(\frac{subframes}{frame}) \times 50(\frac{frames}{s}) \tag{19}$$

$$= 4.1 \text{ x } 10^6(\frac{MACS}{s})$$

To reduce this complexity, the structure of the VSELP code books is exploited. Defining the correlation between the $p(n)$ vector and the filtered code vector, $f'_i(n)$:

$$C_i = \sum_{n=0}^{NSF} f'_i p(n) \tag{20}$$

Expanding $f'i(n)$ using equation (8) yields:

$$= \sum_{n=0}^{NSF-1} \sum_{m=1}^{M} \theta_{im} q'_m(n)p(n) \tag{21}$$

Rearranging the summations yields:

$$= \sum_{m=1}^{M} \theta_{im} \sum_{n=0}^{NSF-1} q'_m(n)p(n) \tag{22}$$

Defining

$$R_m = 2 \sum_{n=0}^{NSF-1} q'm(n)p(n) \tag{23}$$

then substituting this back into 22 yields:

$$C_i = \frac{1}{2} \sum_{m=1}^{M} \theta_{im} R_m \tag{24}$$

Defining the gain of the filtered code vector, $f'_i(n)$:

$$G_i = \sum_{n=0}^{NSF-I} (f'_i(n))^2 \tag{25}$$

Expanding $f'_i(n)$ using equation (8) yields:

$$= \sum_{n=0}^{NSF-I} \left(\sum_{m=1}^{M} \theta_{im} q'_{m(n)} \right) \left(\sum_{j=1}^{M} \theta_{ij} q'_{j(n)} \right) \tag{26}$$

Rearranging the summations yields:

$$= \sum_{m=1}^{M} \sum_{j=1}^{M} \theta_{im} \theta_{ij} \sum_{n=0}^{NSF-I} q'_j(n) q'_m(n) \tag{27}$$

Defining

$$D_{mj} = 4 \sum_{n=0}^{NSF-1} q'_m(n) q'_j(n) \tag{28}$$

and substituting back into equation (27) yields:

$$G_i = \sum_{m=1}^{M} \sum_{j=1}^{M} \theta_{im} \theta_{ij} \frac{D_{mj}}{4} \tag{29}$$

Because:

$$\theta_{ij} \theta_{im} = \theta_{im} \theta_{ijj}$$

and:

$$\theta_{ij} \theta_{im} = 1 \quad \text{for } j=m$$

the equation can be expanded to:

$$G_i = \frac{1}{2} \sum_{j=2}^{M} \sum_{m=1}^{j-1} \theta_{im} \theta_{ij} D_{mj} + \frac{1}{4} \sum_{j=1}^{M} D_{jj} \tag{30}$$

Given two code words indexed by **i** and u such that u differs from i by only one bit **(that** is, bit position v), then:

$$\theta_{uv} = -\theta_{iv} \tag{31}$$

$$\theta_{um} = \theta_{im} \text{ form } != v \tag{32}$$

The correlations C_i and C, are related by:

$$C_u = C_i + \theta_{uv} R_v \tag{33}$$

The gains G_i and G, are related by:

$$G_u = G_i + \sum_{j=1}^{v-1} \theta_{uj} \theta_{uv} D_{jv} + \sum_{j=v+1}^{M} \theta_{uj} \theta_{uv} D_{vj} \tag{34}$$

If the code book is searched in a sequence such that the code vector index changes by only one bit from the previous code vector index, then the previous set of equations leads to a very efficient method to search the code book. By sequencing the indices using a Gray code, only one bit will change as the indices are generated. In addition, only half of each code book needs to be searched because the other half is the complementary set of code vectors (differing only by sign). The sign of C_i is checked to determine which of the complementary code vectors yields a positive gain γ. The resulting computational requirements are now reduced to:

$$CR = 2 \times 4 \times 50 \times \{[2 \times M1 \times NSF + M1 + 28] + [\frac{2^M}{2} \times (M1 + 2)]\}$$

$$= 0.468 \times 10^6 \text{MACS} \tag{35}$$

Gain Quantization

The gain values for each of the three code book contributions to the excitation vector are jointly optimized using a vector quantization table. The development of the quantization procedure can be found in [1]. The parameters required for the joint vector quantization of the gain values are:

$$R_{cc}(j,k) = \sum_{n=0}^{N-1} c'_k(n)c'_j(n) \quad k = 0,2; \; j = k,2 \tag{36}$$

where $c'_k(n)$ denotes the kth ($k = [0...2]$) excitation contribution vector filtered through the $H(z)$ synthesis filter. Therefore, the upper triangular matrix R_{cc} is the crosscorrelation matrix of the three filtered code book excitation contributions.

$$R_{pc}(k) = \sum_{n=0}^{N-1} p(n)c'_k(n) \quad k = 0,2 \tag{37}$$

where $p(n)$ is the perceptually weighted speech minus the ringing in the synthesis filter from the previous frame. The three-element vector R_{pc} is the crosscorrelation vector of the three filtered code book excitation contributions with the $p(n)$ vector.

$$R_x(k) = \sum_{n=0}^{N-1} c^2_k(n) \quad k = 0,2 \tag{38}$$

where $c_k(n)$ denotes the kth ($k = [0...2]$) excitation contribution vector (not filtered). Thus, the vector $R_x(k)$ denotes the energy in each of the three code book excitation contributions.

Equation (39) defines the parameter RS, the energy in the LPC filter's residual signal.

$$RS = NSF \text{ x } R'_q(0) \text{ x } \prod_{i=1}^{NP}(1 - r_i^2) \tag{39}$$

where $R'_q(0)$ is the average power in the current subframe of speech and the product series is the normalized error power in the synthesis filter. $R'_q(0)$ is interpolated from $R_q(0)$ at the subframe rate using the strategy in Equations 40 – 42.

$$R'_q(0) = R_q(0)_{\text{previous frame}} \qquad \text{for subframe 1} \tag{40}$$

$$R'_q(0) = R_q(0)_{\text{current frame}} \qquad \text{for subframes 3,4} \tag{41}$$

$$R'_q(0) = \sqrt{R_q(0)_{\text{previous frame}} R_q(0)_{\text{current frame}}} \qquad \text{for subframe 2} \tag{42}$$

The error equation used in searching the quantization tables is:

$$\begin{aligned} E = & -a\sqrt{GS\ P0} - b\sqrt{GS\ P1} - c\sqrt{GS\ (1-P0-P1)} \\ & + d\ GS\sqrt{P0\ P1} + e\ GS\ \sqrt{P0(1-P0-P1)} + f\ GS\sqrt{P1(1-P0-P1)} \\ & + g\ GS\ P0 + h\ GS\ P1 + i\ GS\ (1-P0-P1) \end{aligned} \tag{43}$$

where P_0 is the fraction of the coder excitation energy due to the adaptive code book contribution, P_1 is the fraction of the coder excitation energy due to the first stochastic code book, and GS is an energy ***tweak*** parameter ($GS = R/RS$). Note: $(1-PO-P1)$ is the fraction of the coder excitation energy due to the second stochastic code book. The definitions of a through *i* follow:

$$a = 2R_{pc}(0)\sqrt{\frac{RS}{R_x(0)}} \tag{44}$$

$$b = 2R_{pc}(1)\sqrt{\frac{RS}{R_x(1)}} \tag{45}$$

$$c = 2R_{pc}(2)\sqrt{\frac{RS}{R_x(2)}} \tag{46}$$

$$d = \frac{2R_{cc}(0,1)RS}{\sqrt{R_x(0)R_x(1)}} \tag{47}$$

$$e = \frac{2R_{cc}(0,2)RS}{\sqrt{Rx(0)R_x(2)}} \tag{48}$$

$$f = \frac{2R_{cc}(1,2)RS}{\sqrt{R_x(1)R_x(2)}} \tag{49}$$

$$g = \frac{R_{cc}(0,0)RS}{R_x(0)} \tag{50}$$

$$h = \frac{R_{cc}(1, 1)RS}{R_x(1)} \tag{51}$$

$$i = \frac{R_{cc}(2, 2)RS}{R_x(2)} \tag{52}$$

The values PO, P1, and **GS** are vector quantized in a three-column table of length 256. For each subframe, the index of the elements that minimize the error equation (43) is selected. The resulting code book gains are defined by the following equations, where the subscript vq indicates the index of the best table entry.

$$\beta_q = \sqrt{\frac{RS\ GS_{vq}\ P0_{vq}}{R_x(0)}} \tag{53}$$

$$^{th}\gamma_{1q} = \sqrt{\frac{RS\ GS_{vq}\ P1_{vq}}{R_x(1)}} \tag{54}$$

$$\gamma_{2q} = \sqrt{\frac{RS\ GS_{vq}\ (1-P0_{vq}-P1_{vq})}{R_x(1)}} \tag{55}$$

For the fixed-point implementation, the energies are calculated and converted up front to floating-point format. The parameters are then calculated in floating point because of the wide dynamic range. These parameters are then scaled back to the 16-bit integer domain according to the largest of the parameters (hence, the ratios between parameters are maintained.)

Speech Decoder

The speech decoder resembles the encoder with the following exceptions:

- The coefficients for the LPC synthesis filter are not the bandwidth-expanded ones. They are taken from the RC coefficients in the RX bitstream.

  There is no closed-loop search procedure.
- There is an adaptive postfilter in the signal flow.

The coefficients for the filter A(z) are interpolated at the **subframe** rate from the reflection coefficients received at the frame rate. For each frame, the quantized reflection coefficients specified by the bitstream are converted to direct form-filter coefficients. They are then interpolated using the same scheme as defined in the interpolation section. The three code book indices are used to look up the correct vector in each of the code books. Each selected vector is multiplied by its corresponding gain value as calculated using equations (53), (54), and (55). The three scaled code book contributions are then summed to form the excitation signal and applied as input to the LPC synthesis filter A(z). In addition, this excitation signal is fed back into the adaptive code book. The output of the LPC synthesis filter is called the nonpostfiltered speech vector. To mask the effects of quantization in the coder, the speech is filtered through a spectral postfilter.

Adaptive Postfilter

The adaptive postfilter shapes the noise spectrum to match the speech spectrum, thus hiding the effects of quantization in the VSELP coder beneath the formants of the speech signal [12]. Given the speech synthesis filter, $1/A(z)$, the postfilter is defined as:

$$H(z) = \frac{A(\frac{z}{bwf1})}{A(\frac{z}{bwf2})} \tag{56}$$

where $0 \leq bwf1 \leq bwf2 < 1$. With $bwf1$ and $bwf2$ defined as bandwidth expansion factors (like the bandwidth factors used in the perceptual-weighting filter), this filter boosts the formants in the speech signal. Several methods exist for the implementation of the postfilter. Two methods are outlined below.

TIA Postfilter

A problem with the postfilter described above is the accentuation of the speech signal's spectral tilt. This results in the attenuation of the higher frequencies of the speech spectrum. The method described in [1] requires the use of a Levinson-Durbin recursion after the bandwidth expansion of the speech correlation coefficients. The denominator coefficients are converted to autocorrelation coefficients and then bandwidth expanded by $w(i) = 0.923077^{(i \times i)}$. Finally, these autocorrelation coefficients are converted back to filter coefficients via a Levinson-Durbin recursion. This proves to be computationally expensive and provides no quality improvement compared to the method described below. In addition to the spectral shaping filter, a brightness filter is used to boost the high frequencies. The speech, after passing through the filter H(z), is scaled to remove any gain introduced by the filter.

$$Scale = \sqrt{\frac{\sum_{n=0}^{NSF-1} (s_{in}(n))^2}{\sum_{n=0}^{NSF-1} (s_{out}(n))^2}} \tag{57}$$

The scale value is then passed through a first order low-pass filter to remove discontinuities:

$$Scale'(n) = = 0.9875 \times Scale'(n-1) + 0.125 \times Scale \tag{58}$$

Modified Postfilter

Rather than adjusting for the spectral tilt in the postfilter via adjusted numerator coefficients, this method utilizes an adaptive brightness filter. The first reflection coefficient of the numerator filter is used as the coefficient for the brightness filter. This method is described in [14]. This results in the same spectral effect as the specified method, yet it is computationally less expensive. This is the method we used for our implementation.

Features of VSELP

The code book described above allows a fast code book search to be conducted. Memory requirements are also reduced since only the basis vectors are stored (not the entire code book). The selected code book index is robust to channel errors because an error in the index changes only the sign of one of the basis vectors. Most importantly, the gains associated with each of the vectors contributing to the excitation vector are jointly optimized and quantized.

TMS320C5x Real-Time Implementation

The DSPSE implementation of VSELP on the TMS320C5x is written entirely in assembly code so that it can fit on one 'C5x running at 20 MIPS. The two main functions, analysis and synthesis, are completely modular and C callable. The memory and MIPS requirements are listed below.

Processing Requirements

The table below lists the processor utilization requirements for the TMS320C5x VSELP vocoder software.

Table 3. VSELP Vocoder Processor Requirements

Application	MIPS Maximum	Utilization at 20 MIPS†	MIPS Average	Utilization at 20 MIPS†
Analysis	16.10	81%	15.30	77%
Synthesis	3.60	18%	3.32	17%

† Values reflect execution from zero-wait-state external SRAM and use of TMS320C5x internal RAM.

Memory Requirements

The table below lists the memory requirements for the TMS320C5x VSELP vocoder software. All memory specifications are in units of 16-bit words.

Table 4. VSELP Vocoder Memory Requirements

Function	ROM	On-Chip RAM	External RAM	Total RAM
Analyzer	8.2K	1.5K	0.23K	1.73K
Synthesizer	3.32K	1.1K	0.23K	1.33K
Full Duplex VSELP	9.0K	1.55K	0.42K	1.97K

The three on-chip memory blocks are b0, b1, and b2 and are used as follows:

Block b0 is a special block in that it is the only segment of RAM that can be switched into program memory. This feature is useful for filtering operations such as the MACD instruction. Because this memory is dynamically switched as program or data memory, no static variables reside in this block. However, this block is used as temporary memory in the code book searches.

Block b1 is used in two ways. The first 350 locations are used as temporary scratch-pad memory. The remaining locations are used for time-critical buffers such as the intermediate weighted excitation vectors and the stack.

Block b2 is used to overlay local temporary variables. This strategy not only saves memory but also allows all local variables to be placed in fast dual-access RAM for maximum DSP performance.

Speech Coder Quality

Quality measures were used to compare the speech output of the fixed point VSELP (TMS320C5x) with a C model of the TIA reference synthesizer. The input bitstream for each of five speakers (three male and two female) produced five reference files, both postfiltered and nonpostfiltered. This same bitstream was used as input to the 'C5x implementations of the VSELP coder. The resulting speech files were compared to the reference files using the SNR measure described below.

SNR Measurements

To track the progress of algorithmic modification, the *segmental* SNR measure was used. The segmental SNR is the average of the each subframe's SNR over some segment of speech.

$$\mathrm{SegSNR} = \frac{1}{L}\sum_{i=0}^{i=L-1} 10 * \log_{10}\left(\frac{\sum_{n=0}^{NSF-1} s_i(n)^2}{\sum_{n=0}^{NSF-1} (s_i(n)-s_p(n))^2}\right) \qquad (59)$$

where L is the length of the speech segment in subframes, s_i is the input speech, and s_p is the synthetic speech. This measure is used in testing vocoder implementations against the reference vocoder. For the five reference files, the output of the synthesizer was compared to the output of the reference vocoder's synthesizer. All the SNR values for the fixed-point implementation were distributed between 25 and 30 dB.

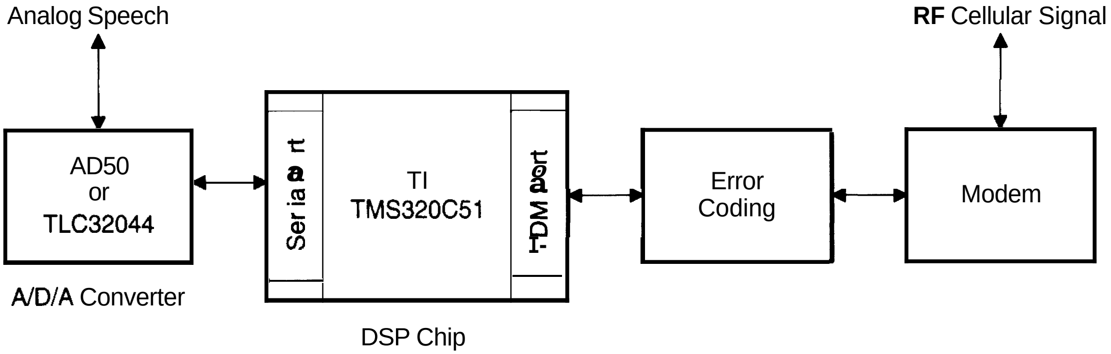
DTMF Performance

The VSELP algorithm must pass the dual-tone multifrequency (DTMF) signals to allow for remote signaling and dialing. Several DTMF files were recorded and processed through the algorithm. The Fourier spectra were analyzed for proper frequency content. In addition, the resulting files were used to signal the central office and correctly initiate a telephone connection.

A Typical Digital Cellular Vocoder Configuration

Figure 4 illustrates a possible digital cellular system configuration. Analog speech sampled by the A/D converter is processed by the TMS320C51 digital signal processor to produce a VSELP coded bitstream. This bitstream is passed through the error-coding block to protect the data against channel errors. Finally, the error-coded VSELP bitstream is modulated and transmitted to the cellular base station. Since the digital cellular telephone is full duplex, incoming RF data is simultaneously processed in the reverse order to produce speech. The incoming signal is demodulated and error corrected before the VSELP synthesis processing and D/A conversion.

Figure 4. Possible Digital Cellular System Configuration



Code Availability

The associated software is available for licensing from DSP Software Engineering Incorporated, 165 Middlesex Turnpike, Suite 206, Bedford, MA 01730

References

1. "Vector Sum Excited Linear Prediction (VSELP) 7950 Bit Per Second Voice Coding Algorithm", *Technical Description*, Motorola, Inc., November 1989.

2. *Cellular System: Dual-Mode Mobile Station – Base Station Compatibility Standard*, IS-54 Project Number 2215, Electronic Industries Association, December 1989.

3. Schroeder, M.R., and Atal, B.S., "Code-Excited Linear Prediction (CELP): High Quality Speech at Very Low Bit Rates", *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing,* March 1985, pp. 937–940.

4. Davidson, G. and Gersho, A., "Complexity Reduction Methods for Vector Excitation Coding", *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing,* April 1986, pp. 3055–3058.

5. Gerson, I.A., and Jasiuk, M., "Vector Sum Excited Linear Prediction (VSELP)", *IEEE Workshop on Speech Coding for Telecommunications,* September 1989, pp. 66–68.

6. Gerson, I.A., "Method and Means of Determining Coefficients for Linear Predictive Coding", U.S. Patent #4,544,919, October 1985.

7. Cumani, A., "On a Covariance-Lattice Algorithm for Linear Prediction", *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing,* May 1982, pp. 651–654.

8. Tohkura, Y., Itakura, F., and Hashimoto, S., "Spectral Smoothing Technique in PARCOR Speech Analysis-Synthesis", *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Volume ASSP–26, December 1978, pp. 587–596.

9. Atal, B.S., Schroeder, M.R., "Predictive Coding of Speech Signals and Subjective Error Criteria", *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Volume ASSP–27, June 1979, pp. 247–254.

10. Kroon, P.. Deptrettere. Ed.F., and Sluyter, R.J., "Regular-Pulse Excitation: A Novel Approach to Effective and Efficient Multipulse Coding of Speech", *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Volume ASSP–34, October 1986, pp. 1054–1063.

11. Linde, Y.. Buzo, A., and Gray, R.M., "An Algorithm for Vector Quantizer Design", *IEEE Transactions*, Communications Volume COM–28, January 1980, pp. 84–95.

12. Chen, Juin-Hwey. and Gersho, Allen, "Real-Time Vector APC Speech Coding at 4800 bps With Adaptive Postfiltering", *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 1987. pp. 51.3.1–51.3.4.

13. Kemp, D.. Sueda. R., and Tremain, T., "An Evaluation of 4800 bps Voice Coders", *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Glasgow. 1989. pp. 200–203.

14. Fenichel, R.. Proposed "Federal Standard 1016 (Second Draft)", National Communications System, Office of Technology and Standards, Washington, DC 20305-2010, November 1989.

15. Campbell, J., Welch, V.. and Tremain, T., "An Expandable Error Protected 4800 bps CELP Coder", *Proceeclings of ICASSP*, Glasgow, 1989, pp. 735-738.

16. Kroon, P., and Atal, B., "On Improving the Performance of Pitch Predictors in Speech Coding Systems", *Abstracts of the IEEE Workshop on Speech Coding for Telecommunications*, 1989, pp. 49–50.

17. Kroon, P., and Atal, B., "Strategies for Improving the Performance of CELP Coders at Low Bit Rates". *Proceedings of ICASSP*, 1988, pp. 151–154.

18. Campbell, J., Tremain, T., and Welch, V., "The DoD 4.8 kbps Standard (Proposed Federal Standard 1016) in Advances in Speech Coding", edited by B. Atal. V. Cuperman, and A Gersho, submitted to Kluwer Academic Publishers, 1990.

19. Macres, J.. "The First Real-Time Implementation of U.S. Federal Standard 4800 bps CELP version 3.1", submitted to the *Proceeding on Speech Technology '90*, 1990.

20. Parsons, T.W., *Voice and Speech Processing,* McGraw-Hill, New York , New York, 1987.

21. Kemp. D.. Sueda. R., and Tremain, T., "Processing of Military and Government Speech Technology '89". *Media Dirnensions*, 1989, pp. 86–90.

Part IV
Error-Correction Coding

U.S. Digital Cellular Error-Correction Coding Algorithm Implementation on the TMS320C5x

Mansoor A. Chishtie
Digital Signal Processing Applications — Semiconductor Group
Texas Instruments Incorporated

Abstract

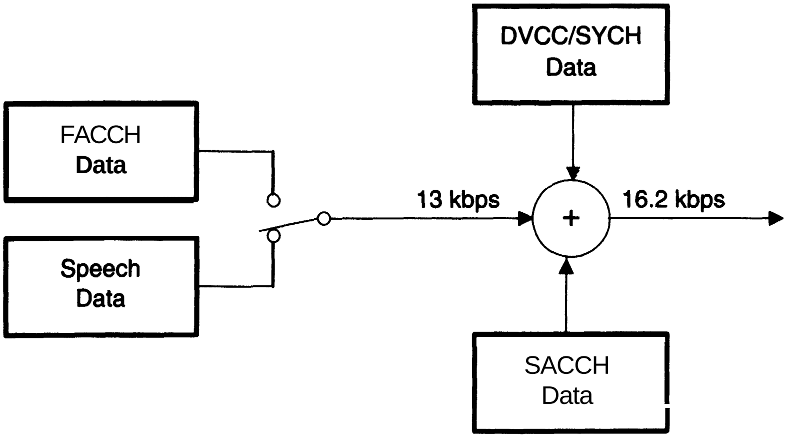
Programmable digital signal processors are commonly used in U.S. digital cellular terminal designs. All digital cellular transmitters employ convolutional and CRC codes to protect against channel-induced errors. Receivers typically use Viterbi decoders and CRC syndrome checks to verify that the decoded data contains no errors. This paper presents selected implementation examples of the error-protection and correction functions of various cellular data channels using the TMS320C5x digital signal processor family.

Introduction

Programmable DSPs are widely used in the new U.S. digital cellular (USDC) radio designs. The primary function of the DSPs in these designs is baseband signal processing. However, many designs are also using the newer DSPs as the system coordinator in the radio, a task typically performed by a microcontroller. This trend is caused by a) system care-abouts of low cost, low power, and small form factor, and b) newer generations of DSPs (such as the TI TMS320C5x family) that have architectures suitable for microcontroller-type functions.

One of the several signal-processing-intensive tasks that a digital cellular radio needs to perform is error protection and correction. The IS-54 voice channels transmit voice and control information in digital form. Although these radio links are primarily used for digital voice transmission (VSELP), a portion of the channel capacity is reserved for control information. This relatively slow bit-rate link is used for background control information such as broadcast messages, mobile-assisted handoffs, etc. This is called slow associated control channel (SACCH) in IS-54 terminology. Another type of signaling channel is called fast associated control channel (FACCH). However, FACCH messages are not sent simultaneously with the voice data. They replace the compressed voice data whenever necessary. Figure 1 shows how these messages are multiplexed with voice data.

Figure 1. Voice and Control-Channel Multiplexing Over One Time Slot



These three digital data channels employ extensive error-protection and correction mechanisms to protect all or most of the transmitted information. Convolutional codes, CRC codes, and bit/frame interleaving techniques are used for this purpose. The convolutional coding schemes used by these three channels are not identical and require slightly different decoding methods to be employed by the receivers. Despite these minor differences, the basic decoding algorithm used by the three channels is usually a Viterbi algorithm. In the rest of this paper, these channel formats are explained separately, a suitable decoding scheme is presented, and its implementation details are discussed.

VSELP Channel Format

The VSELP encoder compresses the digitized speech from 64 kbps to 7.950 kbps. Additional information is added for error protection to increase the total data transfer rate to 13 kbps. The VSELP algorithm operates on a frame-by-frame basis in which each speech frame is 20 ms in duration. The VSELP encoder generates 159 bits of compressed speech for each speech frame. These bits are grouped into two classes: 77 class-I bits that need error protection and 82 class-I1 bits that are sent without any error protection. Class-I bits are protected from channel-induced errors by applying convolutional encoding. Furthermore, error detection is also provided by applying a 7-bit CRC code to the 12 most perceptually significant class-I bits. Finally, this 260-bit speech frame is interleaved over two time slots to protect against burst errors.

Figure 2. Error Protection for VSELP Data

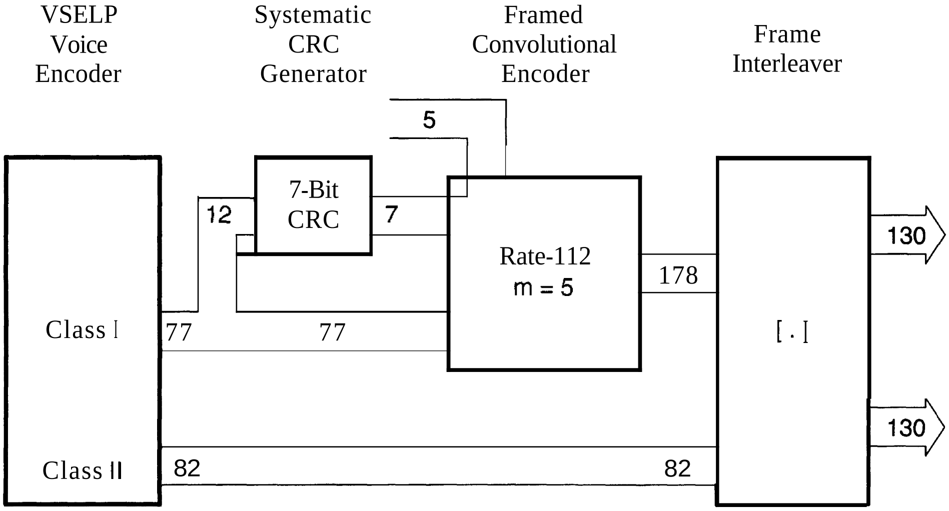
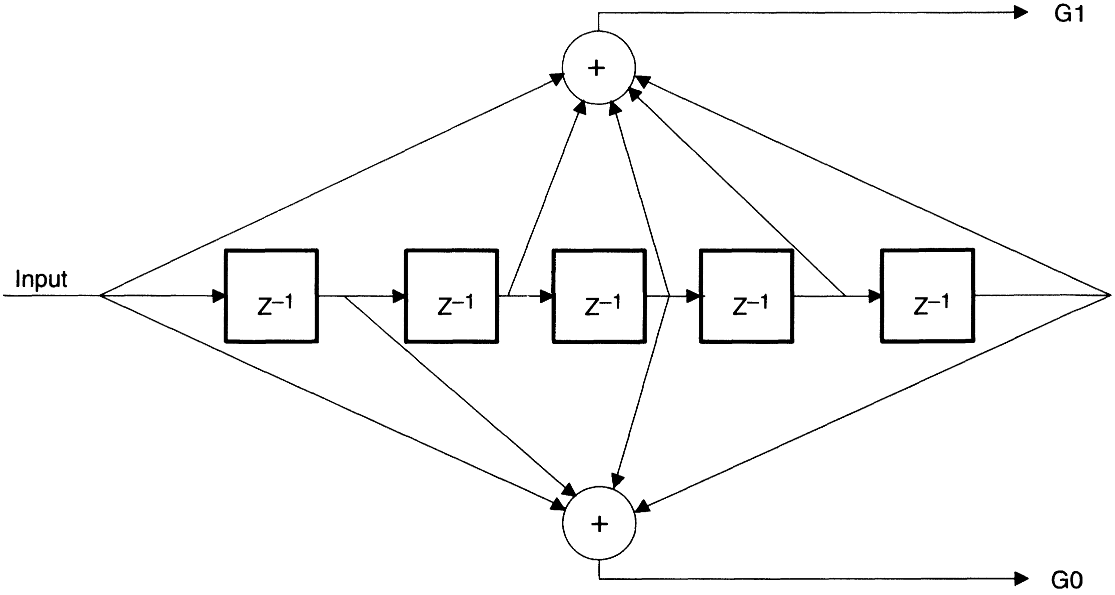
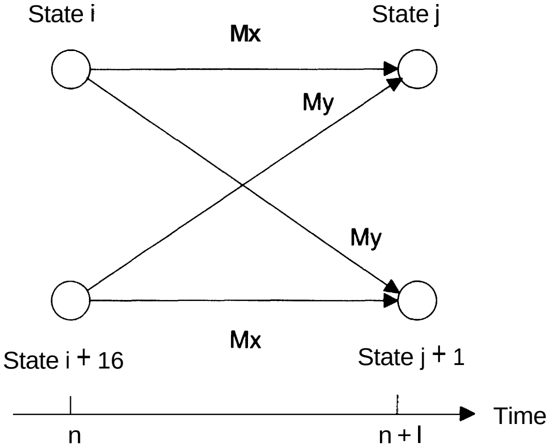


Figure 3. Convolutional Encoder for VSELP Data



The VSELP convolutional encoder is a rate-1/2 framed encoder with a constraint length (denoted by M) of 5. The frame size is 89 bits (see Figure 2), which consists of 77 class-I bits, a 7-bit CRC, and 5 tail bits. Both the initial and the final states are 0. The trellis diagram for this encoder consists of 32 states (that is, 2M) with each state in symbol interval n connected to two states in the next time interval n + 1. The basic building block of this trellis is shown in Figure 4.

Figure 4. Representative Trellis Section for VSELP Convolutional Encoder



Due to the rate-112 encoding scheme, each state is linked to two states in the previous time interval, as shown in Figure 4. The Viterbi algorithm operates on received data by expanding the trellis over a frame length of 89 symbol intervals. Refer to [4] and [5] for general Viterbi algorithm descriptions. A 32-element accumulated cost metric is set up where each element corresponds to one state. Each link from an old state to a new state has a transition cost associated with it. For instance, Mx is the transition cost from state i to state j in Figure 4. These transition costs. which are computed at the symbol rate, reflect the current channel conditions. Each transition cost indicates the probability of a state i to state j transition over one symbol interval. Consider Figure 4 where state j at time interval n + 1 can be reached from either state i or state i + 16 in time interval n. The Viterbi algorithm selects the more likely transition into state j by comparing the total accumulated cost of the two possible links. The accumulated cost of each link is computed by adding the current transition cost to its previous accumulated cost. For example, new accumulated costs of the two links entering state j in Figure 4 are:

$$\text{new–acc–cost}\,[j] = \text{old–acc–cost}\,[i] + Mx \tag{1}$$

$$\text{new–acc–cost}\,[j] = \text{old–acc–cost}\,[i+16] + My \tag{2}$$

The smaller of the two values is selected and the corresponding link is retained for further processing in the next time interval. The other candidate is discarded. This process of selecting one transition entering a state is performed on all 32 states at each symbol interval. Path history of every state is maintained for the entire 89-symbol-long frame. When one frame is processed completely, the state 0 in the last time interval is selected, and its associated path is considered the most likely received path. This path is traced to find the most likely received bit sequence. As shown in Figure 2, the encoder pads five tail bits (all 0s) at the end of each message frame. This ensures that the last encoder state is always 0. Additionally, the initial state of the encoder is also 0 by definition. This requires special consideration by the decoder during initialization of the accumulated cost metric at the beginning of each frame. To assure that state 0 is selected by the algorithm at the beginning of each frame, it is initialized with a lower cost value than that of the other 31 states.

This algorithm can be implemented more efficiently if the underlying symmetry of the trellis structure used is considered. As shown in Figure 4, a pair of states in a symbol interval are connected to another pair of states in the next interval with no other connections to the rest of the trellis. Therefore, all state transitions during one symbol interval can be uniquely broken down into 16 butterfly-like structures similar to Figure 4. Furthermore, only two transition cost values are associated with the four links of each butterfly (Mx and My in Figure 4; in some implementations, Mx is always equal to –My, which leads to further simplification of the structure). This symmetrical structure allows a subroutine that will operate on one butterfly at a time, computing new accumulated cost metrics, selecting the best transition, and storing the path history. This subroutine (or a macro) is invoked 16 times at each symbol interval to update 32 state transitions. Example 1 lists the pseudocode for this function.

Example 1. Pseudocode for Trellis Expansion

```
Acc_Metric1[n]    + Curr_M[x] -> AccB
Acc_Metric1[n+16] + Curr_M[y] -> Acc
min(Acc,AccB) -> Acc_Metric2[m]
If (Acc > AccB) then
   shift 1 in Trans_Tbl[i]
else
   shift 0 in Trans_Tbl[i]
Acc_Metric1[n]    + Curr_M[y] -> AccB
Acc_Metric1[n+16] + Curr_M[x] -7 Acc
min(Acc,AccB) -> Acc_Metric2[m+1]
If (Acc > AccB) then
   shift 1 in Trans_Tbl[i+1]
else
   shift 0 in Trans_Tbl[i+1]
```

The pseudocode shown above performs necessary computations for two states. similar to the butterfly structure shown in Figure 4. There are two accumulated cost metrics used by the code, Acc_Metric1[] and Acc_Metric2[]. One contains previous cost metrics and the other is used to store new accumulated cost metrics. At each symbol interval, roles of the two arrays are reversed. Only two array elements need to be accessed by the subroutine. The offsets between those two elements are always 16 and 1 for the two arrays, respectively. This allows for simple indexing of these arrays regardless of which state is currently being accessed. Similarly, only two current metric values Curr_M[] are accessed by the function. The offset between these two elements can also be made equal to 1 if this array is set up in the form of a circular buffer. Finally, since the path history is stored for the two states j and j + 1 in time n + 1, the two elements of the transition table Trans_Tbl[] that need to be accessed are also offset by 1.

Considerable coding efficiency is gained by taking into account these structural symmetries of the trellis butterfly. As shown in pseudocode above, the accumulator and the accumulator buffer are used to hold total accumulated cost of the two links. The TMS320C5x DSPs support special instructions to select the smaller (or larger) of the two values. The CRLT instruction and the conditional-execute instruction (XC) are used in this implementation to select the lower cost link and update the accumulated cost array and the transition table. Since the accumulated cost arrays are accessed only in steps of 1 or 16, indirect addressing modes of postincrement and postmodification by an index of 16 are used to step efficiently through the table. The current transition cost array, Curr_M[], consists of four elements representing four symbols of the rate-1/2 encoder. It is set up as two circular buffers, each containing two elements. In Example 2, the code listing shows the function implemented in 'C5x assembly code. It is set up as a macro that is invoked 16 times to update all 32 states per time interval.

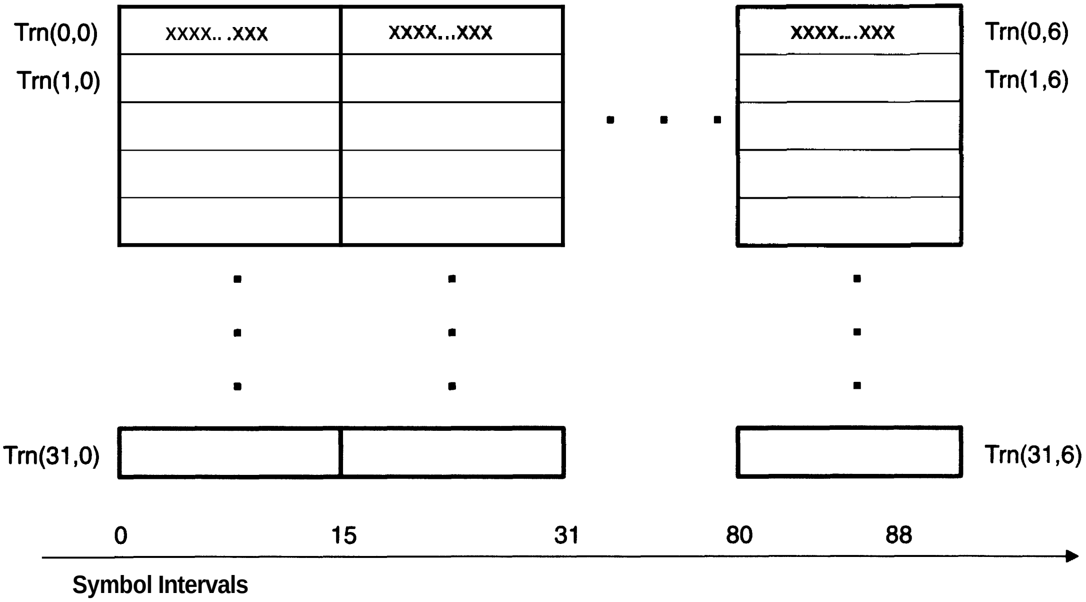
Example 2. Trellis Expansion Macro In ’C5x Assembly Code

```
*
* Entry Conditions:
*
* ARP =  AR1
* INDX=  16
* AR1 -> AccMa[n] ;n=0..31
* CurrMPtr -> CurrM[i] ;i=0..3
* Circ.buffers: CurrM[0..1] and CurrM[2..3]
* AR3 -> AccMb[m] ;m=0..31
* AR4 -> Trn[k]   ;k=0..(6*32)
*
* Exit Conditions:
*
* AR1 -> AccMa[n+1]
* CurrMPtr -> CurrM[i]
* AR3 -> AccMb[m+2]
* AR4 -> Trn[k+2]
*
Texpand .macro  CurrMPtr
        lacc    *0+,CurrMPtr    ; load AccM1[n]
        add     *+,ar1          ; add CurrM[x]
        sacb                    ,
        lacc    *0-,CurrMPtr    ; load AccM1[n+16]
        add     *,ar3           ; add CurrM[y]
        crlt                ; change to crgt for correlation type metric
        sacl    *+,ar4          ; min(path1,path2) -> AccM2[m]
        lacc    *,1             ; load Trn[i]
        xc      1,c             ; if path1>path2
        add     #1              ; shift 1 in Trn[i]
        sacl    *+,ar1          ; save Trn[i]
*
        lace    *0+,CurrMPtr    ; load AccM1[n]
        add     *+,ar1          ; add CurrM[y]
        sacb
        lacc    *0-,CurrMPtr    ; load AccM1[n+16]
        add     *,ar3           ; add CurrM[x]
        crlt                ; change to crgt for correlation type metric
        sacl    *+,ar4          ; min(path1,path2) -> AccM2[m+1]
        lacc    *,1             ; load Trn[i+1]
        xc      1,c             ; if path1>path2
        add     #1              ; shift 1 in Trn[i+1]
        sacl    *+,ar1          ; save Trn[i+1]
*
        mar     *+
        .endm
```

Path History Memory Organization

Path history is generated by the decoder during the forward pass as it expands the trellis. Given that each encoder state can only be reached from one of the two possible states in the previous symbol interval, a single bit can be used to store this information. The state transition table Trn[x,y] is a **32** × 6 word matrix in which each bit position in a row of elements corresponds to one symbol interval. Each row element in a column corresponds to one of the **32** encoder states. In other words, if Trn[x,y] is the matrix where x = 0 ... 31, and y = 0 ... 5, then x corresponds to the encoder state and (16y + bit position) corresponds to the symbol interval.

Figure 5. Transition Table Organization



Trace-Back

Trace-back starts from state 0 in the 89th symbol interval. The corresponding bit in the transition table indicates which state is linked to it in the 88th symbol interval. This bit is the decoder output in the 89th symbol interval. Next, the decoder jumps to the selected state in the 88th symbol interval, generating the next output bit. This procedure is repeated until all 89 symbol intervals are traced back, producing one frame of decoded output. Example **3** shows this algorithm in pseudo-C code.

Example 3. Trace-Back Function — Pseudo-C Code

```
state = 0;
n = 0;
for (word=6; word>=0; word--) {
    for (bitno=15; bitno>=0; bitno--) {
        if (Trn[state,word].bitno == 0) {
            store 0 in Output[n++];
            state = state>>1;
        }
        else {
            store 1 in Output[n++];
            state = (state>>1) + 16;
        }
    }
}
```

This trace-back function is implemented on the ’C5x using its zero-overhead loop structure. Indirect index addressing is used to step through the transition table efficiently. The INDX register holds the current state ID (0 to 15). Bit-reversed addressing is used to left-shift the INDX register for each iteration. Dynamic bit testing is done by using the TREG2 register as a bit pointer to each element of the transition table. Example 4 lists this ’C5x assembly routine.

Example 4. Trace-Back Implementation in ’C5x Assembly Code

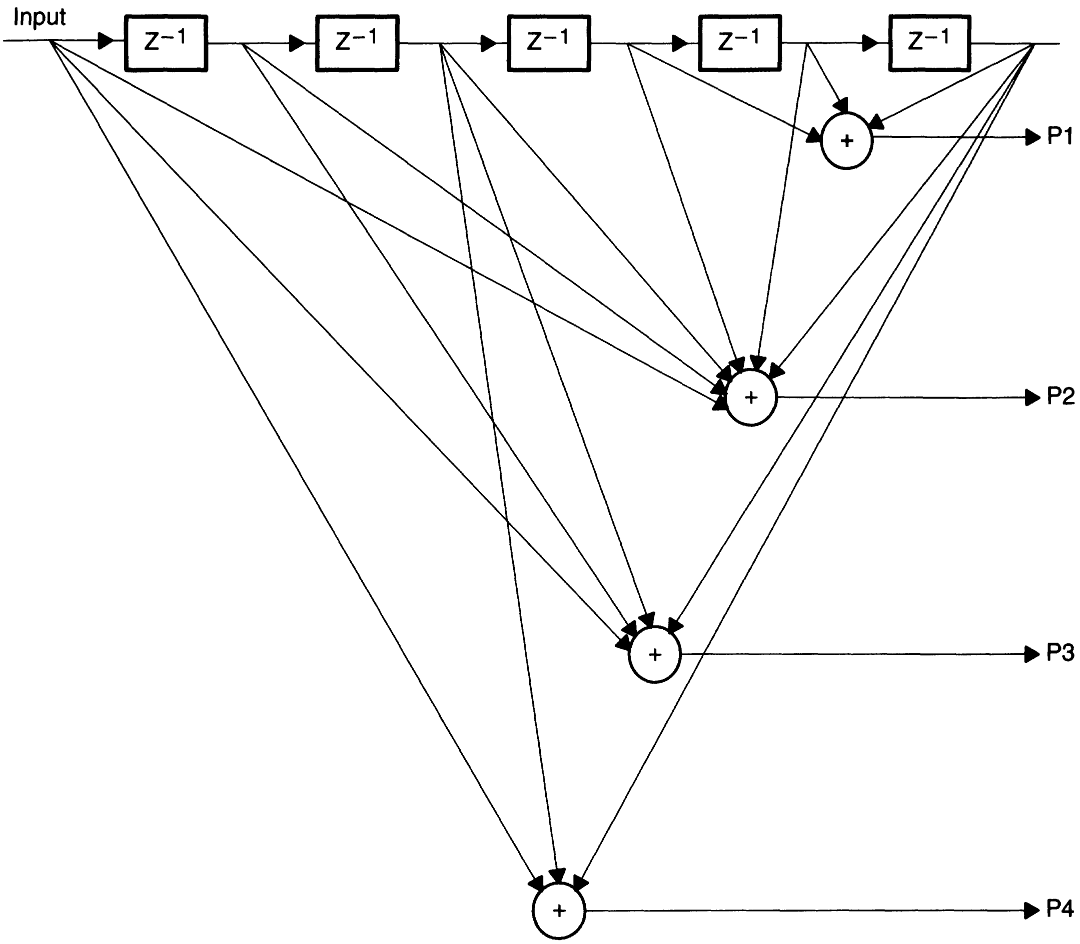
```
TraceBack:
        lar     ar0,#0          ; ar0 is n of Trn[n] (0..31)
        lar     ar3,#OutBuf     ; ar3 -> OutBuf[0]
        lar     ar4,#Trn        ; ar4 -> Path history table
        lacl    #1
        samm    dbmr            ; initialize mask
        lacl    #16-1
        samm    brcr            ; initialize loop count
        lacl    #15
        samm    treg2           ; initialize bit pointer
*
        rptb    trace--1        ; loop 16 times
        mar     *,ar3
        sar     ar0,*
        apl     *+,ar4          ; save OutBuf[i]
        mar     *0+
        bitt    *0-,ar0         ; test bit(i) of Trn[state,word]
        mar     *br0+           ; right-shift INDX by one
        xc      1,tc            ; if bit(i) == 1
        adrk    16              ; add 16 to INDX
        sub     #1              ;
        samm    treg2           ; decrement bit pointer
trace:
        ret
```

FACCH Channel Format

The FACCH is a signaling channel in parallel with the speech path used for transmission of control and supervision messages between the base station and the mobile station. The FACCH replaces the user information block (that is, speech data) whenever necessary [1]. An FACCH message block consists of a 48-bit message frame, a 1-bit continuation flag, and a 16-bit CRC. The standard CCITT CRC-16 code is generated for 49 information bits (1 continuation and 48 message) and eight bits of DVCC color code. The FACCH data (48-bit message, 1-bit continuation, 16-bit CRC) is error protected by means of a rate-114 convolutional code. The resulting 260-bit frame is interleaved over two consecutive bursts in the same manner as the VSELP speech frame.

The rate-114 convolutional encoder has a constraint length of 5. In other words, it operates as a shift register of length 5. Each new bit shifted in results in four parity bits being shifted out of the encoder that are designated P1, P2, P3, and P4. Figure 6 illustrates the encoder shift register.

Figure 6. FACCH Rate-114 Convolution Encoder



The 65-bit input frame to the encoder consists of 48 bits of data, a 1-bit continuation flag indicating whether this is the first word of a message, and 16 bits of CRC code. The encoder does not require five explicit tail bits, as was the case with the VSELP rate-112 encoder. It treats each input frame as a 65-bit circular buffer. The first five bits in each input frame constitute the initial encoder state (that is, C[4], C[3], C[2], C[1], C[0]). The first output bit quadruple (P1, P2, P3, P4) is generated when the sixth bit is shifted in. After shifting the 65th bit in, bit 0 is input to the encoder, creating the circular buffer. The final encoder state is (C[3], C[2], C[1], C[0], C[64]). Note that after one more shift, the encoder state would return to its initial state. In terms of the corresponding trellis structure, this means that there is always a wraparound from the final encoder state to its initial state.

The FACCH decoder is similar to the VSELP decoder except for the following considerations:

- It decodes rate-114 code instead of rate-112 code.
- The encoder frame is 65 bits long.
- Each encoder frame is treated as a circular buffer.

The basic Viterbi algorithm in this case remains identical to the VSELP rate-112 algorithm. There are two paths entering each state from the previous symbol interval. The decoder selects the lower cost link, based on its accumulated cost. However, since each output symbol consists of four bits, there are possibly 16 distinct transition costs that need to be updated at every symbol interval. The rest of the algorithm is similar to the speech decoder algorithm except that the frame size is 65 bits instead of 89 bits.

Since the encoder initial state is not previously known in this case, all states are equally likely in the first symbol interval. Hence, all accumulated costs are initialized to 0 at the beginning of each frame. This can result in poor initial performance of the decoder under low signal-to-noise (SNR) conditions. According to Forney [4], the Viterbi decoder output is unreliable until a path history of four or five times the encoder-constraint length is available. Therefore, the first 20 to 25 decoded bits can contain errors. This problem can be alleviated by considering the final encoder state (in the 65th symbol interval) and the initial encoder state (in the first symbol interval) wraparound. Each received frame is treated as a 65-symbol-long circular buffer, and the decoder is fed with a total of 85 symbols (composed of a 65-symbol frame and 20 repeated initial symbols), thereby generating an artificially long path history. Since 20 initial symbols are repeated, a portion of the path history is redundant. Ideally, path history that corresponds to the first 20 symbol intervals and the last 20 symbol intervals should be identical because it corresponds to the same 20 symbols. However, the trellis generated for the last 20 symbol intervals is more reliable because it takes into account the path history of the previous 65 symbols. Accordingly, the path history of the first 20 symbols is pruned. This approach is taken to avoid the uncertainty of the decoder decisions during the first 20 input symbols. After all the symbols are input to the decoder, the best path (of the possible 32 paths) is selected based on least accumulated cost. This path is traced back to yield the output bit sequence.

Code Availability

The associated program files are available from the Texas Instruments TMS320 Bulletin Board System (BBS) at (713) 274-2323. Internet users can access the BBS via anonymous ftp at *ti.com.*

References

1. *Cellular System: Dual-Mode Mobile Station – Base Station Compatibility Standard,* IS-54 Project Number 2215, Electronic Industries Association, December 1989.

2. *TMS320C5x User's Guide,* Texas Instruments, 1993

3. Pawate, B. I., "Wireless Communications: A Systems Perspective", Texas Instruments, 1992.

4. Forney, G. D., Jr., "The Viterbi Algorithm", *Proceedings of the IEEE,* March 1973.

5. Viterbi, A. J., "Error Bounds for Convolutional Codes and an Asymptomatically Optimum Decoding Algorithm", *IEEE Transactions, Infinity Theory,* April 1967.

Viterbi Implementation on the TMS320C5x for K32 Modems

Mansoor A. Chishtie
Digital Signal Processing Applications — Semiconductor Group
Texas Instruments Incorporated

Introduction

Error-control coding plays an increasingly important role in today's communication systems. Described concisely, *error-control coding involves the addition of redundancy to transmitted data so as to provide the means for detecting and correcting errors that inevitably occur in any real communications process* [1].

Such coding techniques are particularly useful for transmission over limited-power channels like general-switched telephone network (GSTN). Adding redundancy to the transmitted data and making use of soft-decision decoding, the bit-error rate can be reduced considerably without increasing transmission power. These coding techniques have proved very useful in the past decade, and many of them have been standardized for modems and other communication devices.

CCITT recommendation V.32 is one such standard that uses trellis-coded modulation and Viterbi decoding to achieve forward error correction at a data transmission rate of 9600 bits per second (bps). This application report deals with the general theory and implementation of the encoding and decoding algorithms required for the V.32 family of modems.

The architecture of the fifth generation of Texas Instruments digital signal processors (DSPs) is especially suited for soft-decision encoding and decoding algorithms. These dynamic programming algorithms often make use of looped code, conditional execution, min-max searches, and pointer-addressing techniques. The enhanced TMS320C5x core CPU allows zero-overhead looping, multiple-condition branches, delayed jumps and calls to minimize execution time, min-max instructions to implement efficient search algorithms, and postmodified indirect addressing (which includes indexed, circular, and bit-reversed addressing modes). These algorithms can be executed very rapidly since almost all ’C5x instructions take only one machine cycle (25 ns) to execute.

Introduction to the V.32 Standard

V.32 modems are designed for use on connections on GSTNs and on point-to-point 2-wire leased telephone-type circuits. The full-duplex mode of operation is supported using echo-cancelation techniques for channel separation. Each channel uses quadrature amplitude modulation (QAM) with a synchronous line-transmission rate of 2400 symbols per second (baud).

QAM is a modulation technique that allows two independent information channels to be modulated into a single carrier signal. These two channels are commonly referred to as *real* and *imaginary* (or I and *Q*)[1] components of the signal. A constellation diagram illustrates this concept (see Figure 1). Each point on the constellation has a unique set of real and imaginary components. For a 16-point constellation, four bits are required to uniquely represent each point.

If the input data stream is grouped into quad bits (also called symbols), each quad bit can be mapped to a constellation point, and corresponding I and Q values are modulated into a QAM signal. V.32 modems have a data-transmission rate of either 4800 bps or 9600 bps. At the rate of 9600 bps, either a 16-point or a 32-point constellation can be used (see Figure 1). Obviously, 5-bit-long symbols are required to map each point of a 32-point constellation.

---

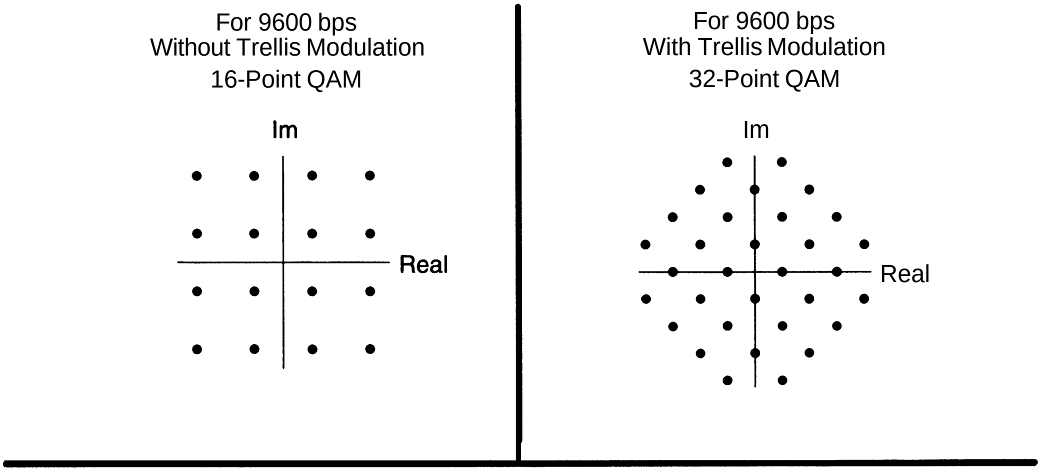
[1] I and Q components are also referred to as X and Y in literature. Both notations are used interchangeably in this paper.

The V.32 standard recommends two alternative modulation schemes at 9600 bps: one using a 16-point constellation, and the other using trellis (convolutional) coding with a 32-point constellation. When using the trellis coding, the input data stream to be transmitted is divided into groups of four consecutive data bits. The first two bits of each group are first differentially encoded and then convolutionally encoded to generate a set of three bits. The other two bits are not encoded but are passed to the output stage. Thus, each output group consists of five bits. These five bits are then mapped into a 32-point (diamond-type) constellation. On the receiver end, a maximum-likelihood decoding algorithm (due to Viterbi) is used to estimate the transmitted data.

This report deals with the encoding and decoding algorithms as required for the 9600-bps 32-point constellation transmission. The basic encoding algorithm is known as a convolutional encoding scheme, and the decoding algorithm scheme is based on the Viterbi algorithm. Although the 32-point constellation is used extensively to help decode the signals, the actual modulation/demodulation scheme is not implemented in software.

Figure 1. V.32 Modems

(a) V.32 Modems Constellations

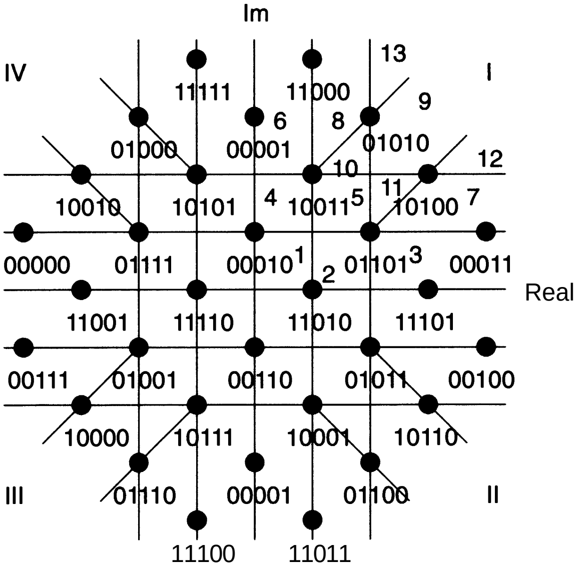


For 9600 bps
Without Trellis Modulation
16-Point QAM

For 9600 bps
With Trellis Modulation
32-Point QAM

Im
Real
Im
Real

(b) V.32 Modems Constellation Regions



Im
IV
11111
11000
13
I
9
6
8
01000
00001
01010
12
10
10010
10101
4
10011
5
11
10100
7
00000
01111
00010
1
2
01101
3
00011
Real
11001
11110
11010
11101
00111
01001
00110
01011
00100
10000
10111
10001
10110
III
01110
00001
01100
II
11100
11011

Each point corresponds to a 5-bit output symbol.

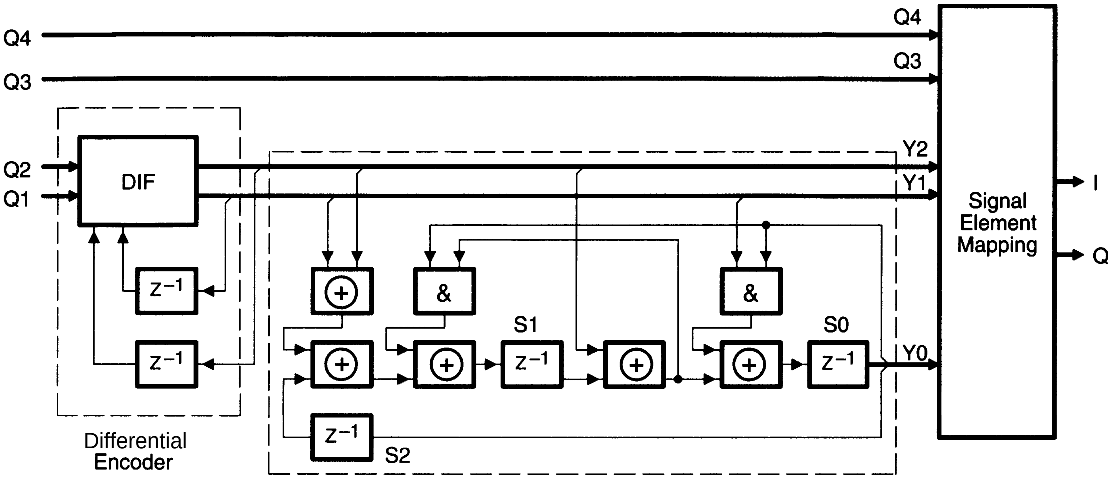
Standard V.32 Encoder

The V.32 encoder (see Figure 2) is divided into two functional blocks:

- Differential encoder
- Convolutional encoder

The input data stream to the encoder is divided into 4-bit long symbols (Q1, Q2, Q3, Q4). Each symbol is processed by the encoder, and the resulting output symbol is 5 bits long (YO, Y1, Y2, Q3, Q4). The output symbol is larger than the input symbol because it contains error-correction information in addition to the transmit data.

Figure 2. V.32 Encoder



$Y1_n = Q1_n \oplus Y1_{n-1}$

$Y2_n = (Q1_n \bullet Y1_{n-1}) \oplus Y2_{n-1} \oplus Q2_n$

The V.32 standard recommends the QAM technique to transmit data over the channel. Without any error correction information, each symbol has four bits, requiring a 16-point constellation as shown in Figure 2. If a convolutional encoding scheme is employed, each symbol has five bits, and a 32-point constellation is required.

In general, for the same average power, a modulation scheme using a 32-point constellation has higher bit-error rate (BER) when compared with a 16-point constellation scheme. This is because the minimum Euclidean distance between any two points on a 32-point constellation is relatively small, which decreases the noise margin. However, convolutional encoding introduces constraints in transforming an input symbol to a 5-bit output symbol. Specifically, it does not allow two consecutive output symbols to be in the eight neighborhood positions of each other, as seen on the constellation diagram. The minimum distance between two consecutive output symbols is thereby increased, thus providing an overall performance gain of 3 dB.

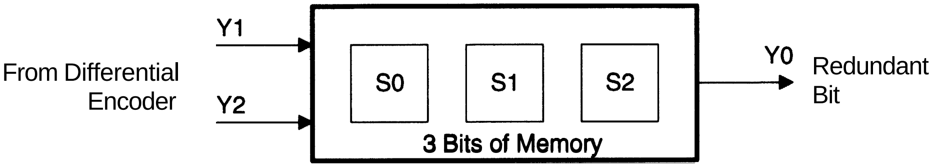
The differential encoder provides protection against 180° phase ambiguity in the channel. The following two equations describe the differential encoding algorithm:

$$Y1_n = Q1_n \oplus Y1_{n-1} \tag{1}$$

$$Y2_n = (Q1_n \bullet Y1_{n-1}) \oplus Y2_{n-1} \oplus Q2_n \tag{2}$$

Notice in Figure 3 that only two input bits are differentially encoded. Because of differential encoding, errors caused by phase reversal in the channel are not allowed to propagate, and the information sequence is reconstructed by the receiver except for the errors at points where phase reversal has occurred [1].

Figure 3. Viterbi Encoder — Convolutional Encoding Scheme



Definitions:

Convolutional Encoder

- SO, S1, and S2 are called delay states
- YO, Y1, and Y2 are called path states

Constraint condition:

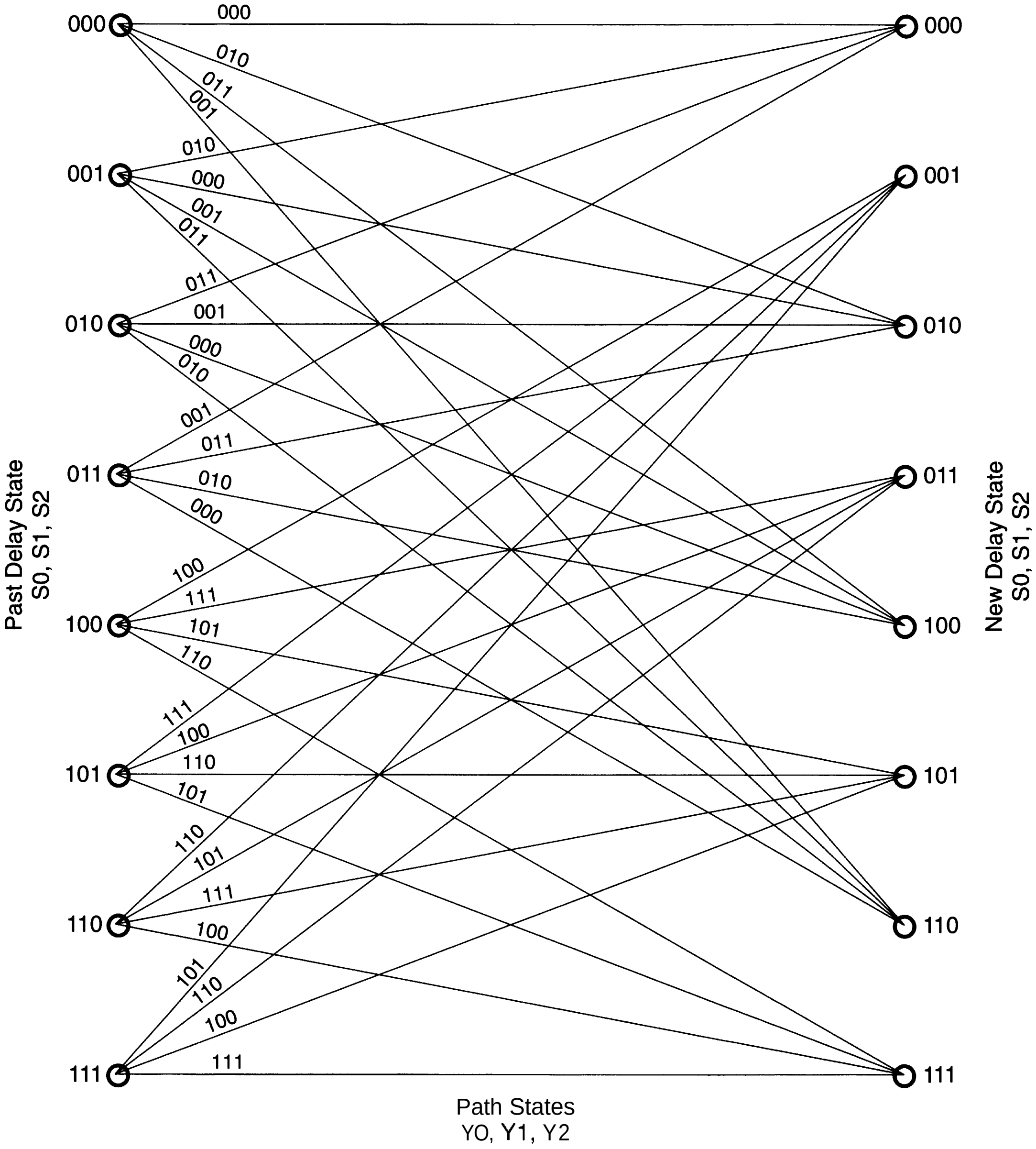
- Given a particular set of delay states (SO, S1, S2), not all path states (YO, Y1, Y2) are possible.

The convolutional encoder takes the two differentially encoded bits (Y1, Y2) and generates an output bit YO. YO is often called the redundant bit because it carries only the forward error-correction information. Functionally, the convolutional encoder is a 3-bit shift register interconnected by AND and XOR logic. A simplified diagram of a convolutional encoder is shown in Figure 3. By convention, the three bits of encoder memory (SO, S1, and S2) are called delay states, and the set of output bits (YO, Y1, and Y2) are known as path states. The idea behind this terminology will become obvious later when the trellis structure is considered. The size of encoder memory is sometimes referred to as its constraint length.

One important constraint is imposed by the encoder. Given a particular set of delay states (SO, S1, and S2), not all path states are possible in that time interval. For instance, given a delay state (0, 0, 1) for the encoder, only four path states (0, 0, 0), (0, 1, 0), (1, 0, 0), and (1, 1, 0) are allowed in next time interval.

This leads to the concept of trellis structure. Since the encoder is essentially a finite-state machine, a finite-state diagram may be used to represent it. There are eight possible delay states of the encoder. At any given time, only one delay state (SO, S1, or S2) represents the encoder. In the next instant, only four delay states are possible instead of eight. The particular path chosen at that time depends on the current path state of the encoder (hence, the name path state). The trellis diagram (Figure 4) concisely illustrates all possible transformations from one delay state to another, along with their corresponding path states.

Figure 4. V.32 Modem Trellis Diagram


000
010
011
001
001
010
000
001
011
010
011
001
000
010
011
001
011
010
000
100
100
111
101
110
101
111
100
110
101
110
101
111
110
111
100
101
110
100
111
111
000
001
010
011
100
101
110
111
Past Delay State
S0, S1, S2
New Delay State
S0, S1, S2
Path States
YO, Y1, Y2

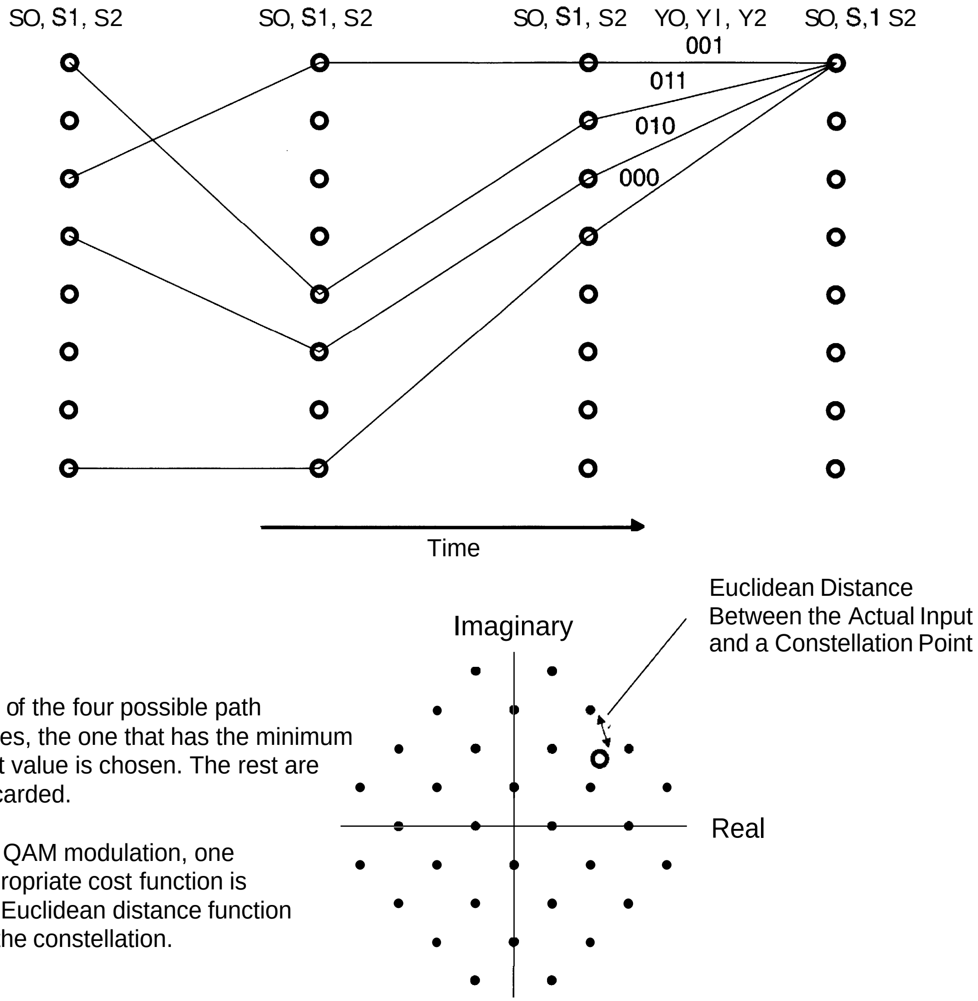
NOTE: Finite-state diagram for the convolutional encoder showing the relationship between delay and path states. Not all delay states can be reached from a previous delay state.

Viterbi Decoder

The Viterbi algorithm is based on a soft-decision maximum-likelihood decoding technique. The main function of any decoder is to select the most likely output. A simple hard-decision decoder selects a code word that differs from the received sequence in the smallest number of positions. In other words, the code word is chosen that minimizes distance between the received signal and the code word. A soft-decision decoding scheme makes use of past history and reliability information to decode incoming data. A necessary ingredient of any soft-decision decoder is a suitable distance (or cost) function.

A cost function may be unique to each modulation technique. Two widely used cost functions are the Hamming distance and the Euclidean distance functions [2]. The standard Viterbi algorithm does not specify any particular cost function. The Hamming distance function is suitable for binary signals. For PSK and QAM signals, the Euclidean distance function on their respective constellations is appropriate. For an added white gaussian noise (AWGN) channel, the farther the received signal from a point on the constellation, the less likely that it corresponds to that point. Therefore, the distance between the received signal (as it is mapped on the constellation) and a hypothesized output point on the constellation makes a good cost function for any QAM signal. Since V.32 uses QAM modulation, the distance estimate on its constellation is used as the cost function.

Figure 5. Viterbi Decoding — Output Tracking and Cost Function



Euclidean Distance Between the Actual Input and a Constellation Point

Out of the four possible path states, the one that has the minimum cost value is chosen. The rest are discarded.

For QAM modulation, one appropriate cost function is the Euclidean distance function on the constellation.

Figure 7 shows an expanded trellis diagram over several symbol time intervals with the x axis representing time and the y axis representing the eight possible delay states of the encoder. The encoder may attain only one delay state at any given time, but the decoder keeps track of all the possible states until it decides which one to select. This is the essence of soft-decision algorithms in which the actual decision is delayed until more information is available. Ideally, the maximum-likelihood method looks at the entire stream of input before making any decision about the output. Clearly, this approach is not feasible for real-time applications due to two factors:

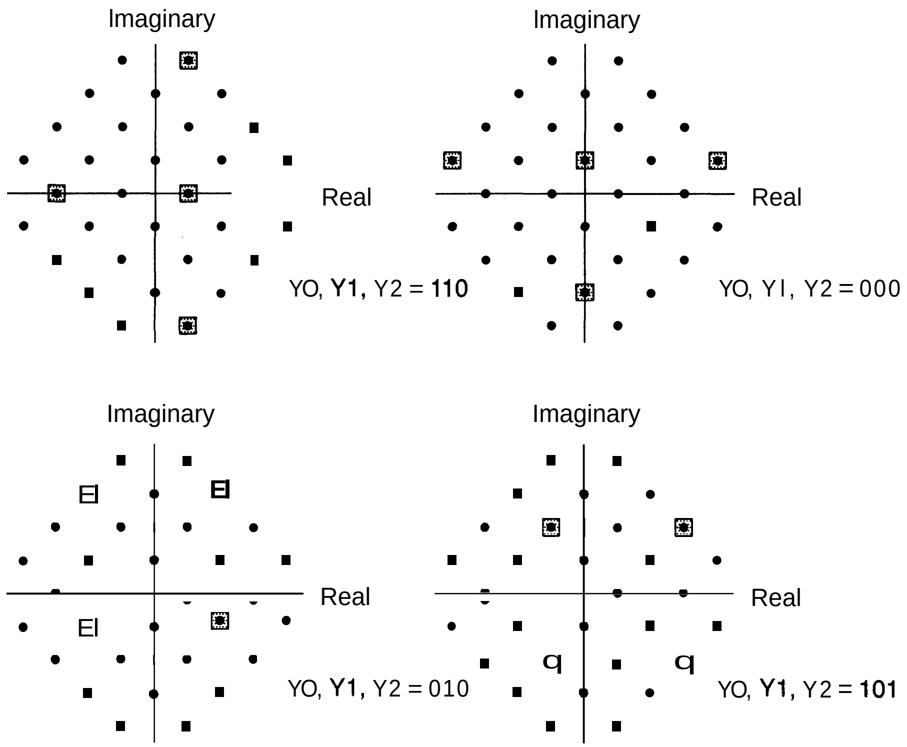
- Prohibitive memory requirements, even for relatively small blocks of data
- Inherent time delay before the decoder selects an output

The more practical approach taken by Viterbi is to consider only a finite length of input data before making a decision about the output. The decision-making process relies heavily on the cost function.

To understand this algorithm, consider the expanded trellis diagram as shown in Figure 7. At each time interval, there are eight possible delay states. Since the decoder must keep an "open mind" until it is time to select the most likely output, all eight states are considered as possible representations of the encoder in that time interval. A particular delay state can be approached only by four states from the previous time interval (see Figure 5). The decoder selects only one of these four states so as to establish a link between the previous time interval and the current one. Note that each link is identified by the path state it represents.

Each path state consists of three bits of a 5-bit symbol. Therefore, one path state uniquely identifies a set of four constellation points. The V.32 signal space mapping is defined in such a way that each set of four points is symmetrically arranged and equally spaced on the constellation, as shown in Figure 6. Furthermore, each set of points is spaced as far apart as possible on the constellation. At the beginning of each sample interval, the decoder compares the received signal with each set and selects the point from each set that is closest to the signal. Essentially, this is a form of hard decoding, but its effect on the quality of the decoder performance is not significant. This is because each set of four points is widely spaced on the constellation so that any noise perturbation is less likely to affect these estimates.

Figure 6. V.32 Modem — Signal Element Mapping

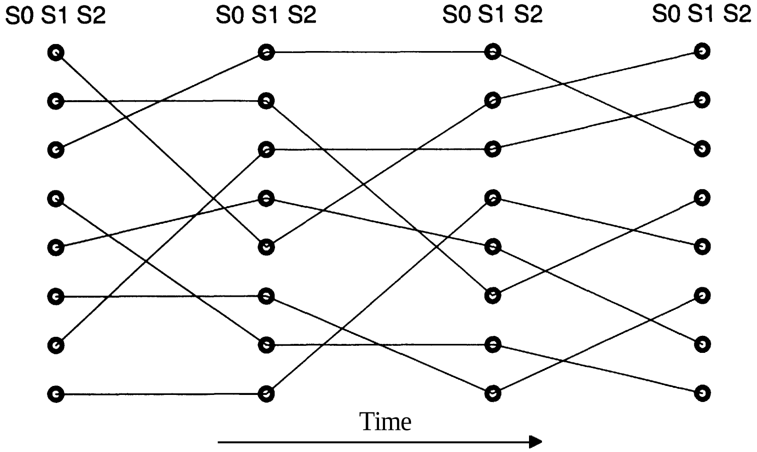


Bit Sequence = YO, Y1, Y2, Q3, Q4

Eight constellation points are selected, and their respective distances from the received signal are computed. Each point corresponds to a different path state. Since each link in Figure 5 is identified by a path state, these computed distance values are associated with each link.

By selecting all eight links, connections are established between the delay states at the current time and the previous time (see Figure 7). In this way, eight independent path traces are stored in memory. The cost function is now updated for each of these path traces. The cost function is the sum of distances associated with each link of a path trace.

Figure 7. Viterbi Decoding — Dynamic Programming



- For every time increment, the minimum cost line is chosen for each of the eight delay states.
- Eight independent path traces are stored in memory.
- For each track, current cost is accumulated as it hops over the delay states.
- The state with the minimum accumulated distance is selected to receive output.

Of the eight path traces, the one that has minimum cost (or accumulated distance) is selected as the most likely path to receive the output. The selected path is traced back, and the 3-bit path state value (YO, Y1, Y2) that is associated with the last link stored in memory is the result of the Viterbi algorithm. Note that this 3-bit result does not uniquely identify a 5-bit output symbol. The four constellation points that correspond to the 3-bit result are compared with the input corresponding to that time interval, and the 5-bit value associated with the point that is closest to the input is the output of the decoder. Since the output of the decoder corresponds to the time period of the last link, it lags the input of the decoder by the length of the path history maintained by the decoder. It is experimentally determined that the optimal length of a Viterbi decoder is four or five times the constraint length of the convolutional encoder [1]. The V.32 encoder has a constraint length of 3, and the decoder keeps a path history of the past 16 time intervals.

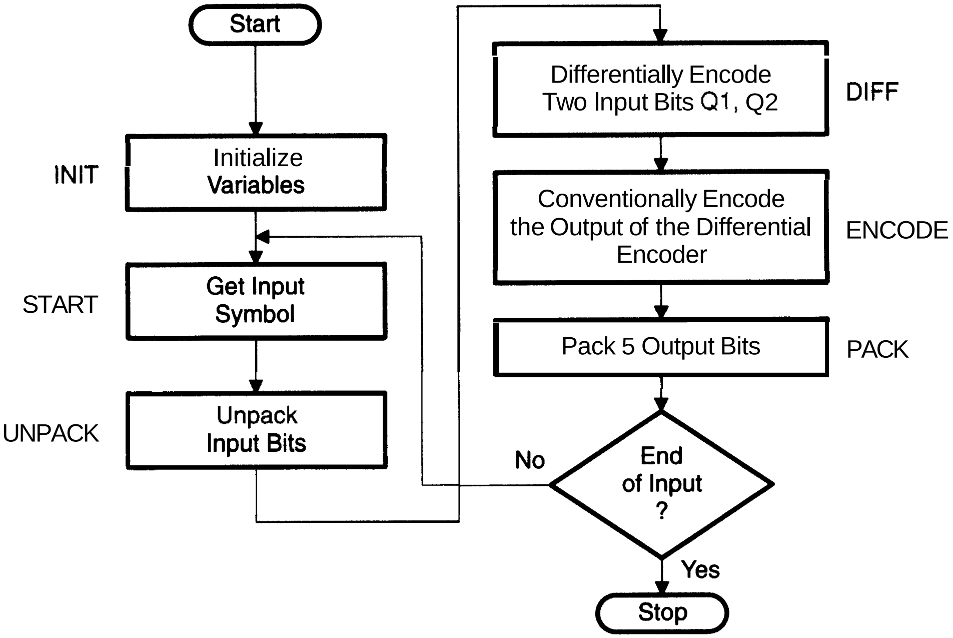
Algorithm Implementation on the TMS320C5x

The three most useful features of TMS320C5x for the Viterbi algorithm are circular buffers, minimum-maximum instructions, and zero-overhead loops. Circular addressing is used extensively throughout the decoder algorithm to access the distance tables, stepping through the path and delay states, and tracing back the past path states to get output. Minimum-maximum value instructions are used in search algorithms to compute minimum Euclidean distance for each state and to find minimum accumulated distance at each time interval. Since the algorithm is based on a dynamic programming technique, it tends to have a multiple looped structure. The zero-overhead loops of TMS320C5x are frequently used by the decoder program.

Encoder Implementation

The V.32 encoder block diagram is shown in Figure 2. As previously explained, it has two functional blocks: the differential encoder and the convolutional encoder. The encoder program flow is shown in Figure 8.

Figure 8. V.32 Encoder Program Flow


Start
INIT
Initialize Variables
START
Get Input Symbol
UNPACK
Unpack Input Bits
Differentially Encode Two Input Bits Q1, Q2
DIFF
Conventionally Encode the Output of the Differential Encoder
ENCODE
Pack 5 Output Bits
PACK
No
End of Input ?
Yes
Stop

The initialization routine INIT sets up auxiliary registers to point to input and output tables and resets the delay states (SO, S1, S2) to 0. This ensures that the initial state of the encoder is known beforehand. It is useful from the decoder point of view because the decoder initializes the cost of delay state 0 to 0 so that this state is always selected in the very first time interval.

The encoder expects the input symbols to be stored in the table PCKD_IP with each element of the table containing a right-justified 4-bit symbol. The table input method is employed because of its simplicity. For real-time applications, other techniques can easily replace the default method. If the input data is coming from an ADC, a simple approach is to create two buffers. One is read by the encoding algorithm, while the other is filled with incoming data by an interrupt service routine. In case the encoding process is required to be synchronous with incoming data, no data buffer is needed. At every symbol time, the input symbol is read from a peripheral device, and the resulting 5-bit output symbol is sent to another external device.

The encoding algorithm operates on binary inputs. Therefore, each input symbol is unpacked into four words (which correspond to each bit) before any processing is done. The UNPACK section uses a zero-overhead block-repeat loop and PLU instructions to perform the unpacking operation.

```
UNPACK:     LACC    LOCATE          ;Get packed input bits
            RPTB    LOOP1           ;For i=0;i<=3;++i
            SACL    *               ;Save the word
            APL     *-              ;Keep LSB only
LOOP1:      SFR                     ;Shift right to get next bit
```

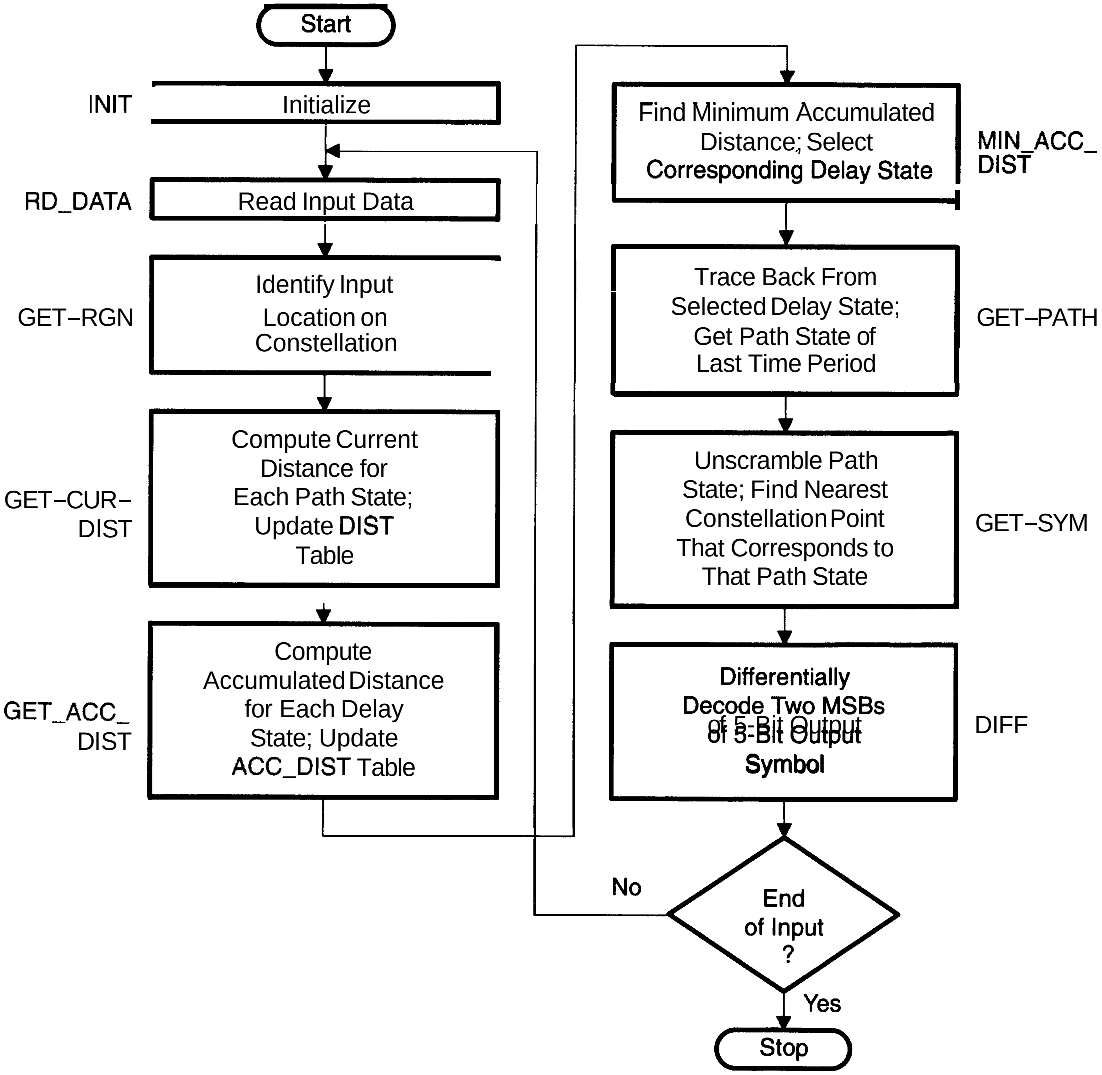
The DIFF function differentially encodes two input bits according to Equation (1) and Equation (2) on page 83. Its output overwrites the original two input bits located in INPUT table. Next, the convolutional encoder processes these two bits and generates a redundant bit YO. The encoder state (SO, S1, S2) is stored in the STATMEM table, and it is updated each time a new redundant bit is generated.

Finally, the resulting five output bits (OUTPUT + INPUT) are packed into a single word by the PACK function. This output word contains five right-justified bits (YO, Y1, Y2, Q3, Q4), and it is stored in the output buffer PCKD_OP in sequential order. Note that these five output bits could be sent to a DAC or a front-end modulator instead.

Viterbi Decoder Implementation

In contrast with the convolutional encoding algorithm, the Viterbi decoding algorithm is computationally more complex and numerically more intensive. In general, the execution time of the decoding algorithm is significantly greater than the execution time of the encoder algorithm. This section describes the algorithm in detail as it is implemented on the TMS320C5x. Although the code presented here is designed for the V.32 modem standard, it could easily be transformed for any other application of the Viterbi algorithm.

Figure 9. Decoder Flowchart


Start
INIT
Initialize
RD_DATA
Read Input Data
GET–RGN
Identify Input Location on Constellation
GET–CUR–DIST
Compute Current Distance for Each Path State; Update DIST Table
GET_ACC_DIST
Compute Accumulated Distance for Each Delay State; Update ACC_DIST Table
Find Minimum Accumulated Distance; Select Corresponding Delay State
MIN_ACC_DIST
Trace Back From Selected Delay State; Get Path State of Last Time Period
GET–PATH
Unscramble Path State; Find Nearest Constellation Point That Corresponds to That Path State
GET–SYM
Differentially Decode Two MSBs of 5-Bit Output Symbol
DIFF
No
End of Input ?
Yes
Stop

The decoder program flowchart is shown in Figure 9. Each process block in the flowchart corresponds to an independent function. The modularity of each block is sacrificed somewhat to gain execution efficiency. In other words, each block is integrated, to a certain extent, with the block that precedes it. The results of a block are frequently passed in internal registers to the next block. However, all system variables are defined explicitly in the beginning, and the line-by-line comments in the source code help identify where the results are being stored.

The initialization routine INIT is called to set up tables and variables. The ACCDIST table, which holds eight accumulated distance values for each delay state, is initialized by this function. As discussed in the ***Standard*** V32 ***Encoder*** section on page 82, the first state of the encoder is always (0,0,0) (that is, state 0). To ensure that the decoder always chooses state 0 in the first time interval, the initial accumulated cost of state 0 is set to 0 while the rest of the states are set to a cost of 0.5.

The routine RD_DATA is called once every symbol interval to read new data. This is the only routine that needs to be rewritten to suit each application. The code presented here is not designed for any specific hardware. It assumes that some test data has already been stored in the TST_INP table before the decoder is invoked. The input is in the form of 5-bit symbols output by the encoder. Two look-up tables, XLOC and YLOC, convert each symbol to its equivalent real and imaginary axis values (also called XY or IQ values). The channel noise and distortion effects may be added to the I and Q channels independently. The resulting values are saved in variables CURR_X and CURR_Y for later use. This approach is taken so that test data and channel noise data may be computed independently of each other and stored in respective tables before the decoder is invoked. Obviously, this is not a real-time approach. The front-end demodulator can provide I and Q values directly to the device. In that case, RD_DATA is required to save only those values in CURR_X and CURR_Y locations. Each I and Q (or X and Y) input can have a maximum resolution of 16-bits.

Once the current input is located on the constellation by X and Y values, eight constellation points corresponding to the eight path states that are closest to this input point must be identified. Note that each path state corresponds to four unique constellation points (see Figure 6). The brute force method of determining these constellation points is to consider each group of four points individually, compute the distance from each point to the input, and select the closest one. This requires all 32 points that compose the **V.32** constellation to be considered for each input symbol. Another way to make the selection is to use a look-up table. Since the locations of the constellation points are known beforehand, it is simpler to identify the region where the input lies and use a table to determine the eight points that are closest to that region. As shown in quadrant I in Figure 1(b), there are 13 distinct regions in each quadrant of the constellation. Each region has a unique set of eight constellation points (corresponding to eight path states). A table called REGION is set up in data memory that contains 13 macro elements, each element having four subelements corresponding to four quadrants of the constellation. Each subelement is a set of eight pointers to the closest constellation points.

To identify the region where the current input lies, the following decision algorithm is used, where X,Y is the location of the current input on the constellation shown in Figure 1(b).

```
If |X| <= 1 Then
   If |Y| <= 1 Then
      Region#1
   Else
      If |Y|<= 2 Then
         Region#4
      Else
         Region#6
Else
   If |X| <= 2 Then
      If |Y| <=1 Then
         Region#2
      Else
         If |Y| <=2 Then
            Region#5
         Else
            If |Y| <= |X|+1 Then
               Region#10
            Else
               Region#8
   Else
      If |Y| <= 1 Then
         Region#3
      Else
         If |Y| > |X|+1 Then
            Region#13
         Else
            If |Y| <= |X|-1 Then
               If |Y| <= 2 Then
                  Region#7
               Else
                  Region#12
            Else
               If |Y| <= 2 Then
                  Region#11
               Else
                  Region#9
```

After identifying a region, a quadrant is selected according to the polarities of X and Y.

Refer to the GET−RGN function of the decoder source code for implementation details. Note the use of delayed conditional branches and the XC instruction to avoid flushing the pipeline. The result of the GET−RGN function is a pointer to the REGION table.

The current cost of each path state is defined as the distance from the current input to the respective constellation point. The result of the GET−RGN function points to a set of eight constellation points. If (X,Y) is the input for a given time interval, and (Xk,Yk) are eight constellation points that correspond to state k (where k = 0...7), then the current distance table is defined as:

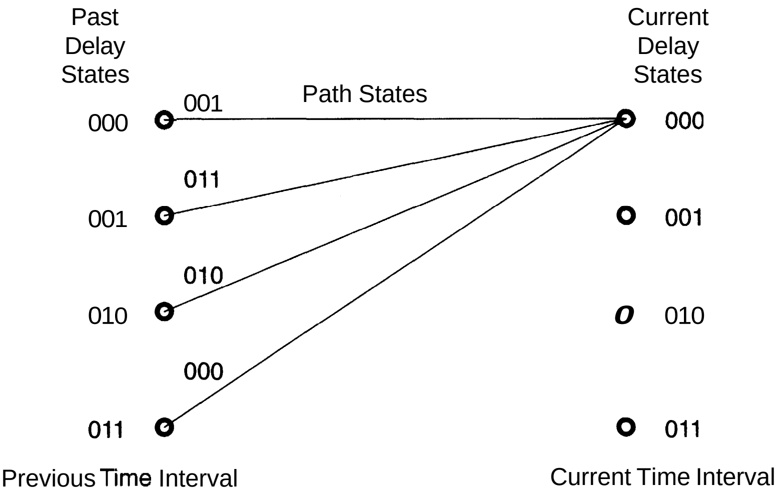
$$\text{DIST } [k] = (Xk-X)^2 + (Yk-Y)^2 \; ; \quad k = 0 \ldots 7 \tag{3}$$

The square root operation is not performed because it is time-consuming. Although the square root function is not linear, distance values without the square root operation work well because the relationship between x and sqrt(x) is one-to-one and monotonic. The GET–CUR–DIST routine performs this computation for each path state.

```
STATE0:
       LAR    AR2,*+,AR2           ;Get address of 1st point out of 8
       MAR    *o+                  ;Add XLOC, AR2 points inside XLOC
       LACC   *                    ;Get x value of 1st point
       SUB    CURR_X               ;Subtract current x value
       SACL   DIFF_X               ;Save (Xc-Xi)
       SQRA   DIFF_X               ;P=(Xc-Xi) ^2
       ADRK   #32                  ;Now AR2 points inside YLOC
       LACC   *,0,AR0              ;Get Y value of 1st point
       SUB    CURR_Y               ;Subtract current y value
       SACL   DIFF_Y               ;Save (Yc-Yi)
       LACL   #0
       SQRA   DIFF_Y               ;P=(Yc-Yi) ^2, ACC=(Xc-Xi) ^2
       LTA    SMALL                ;ACC=(Xc-Xi) ^2+(Yc-Yi) ^2
       SACH   DIST,4               ;Save acc. distance*2^4
       MPY    DIST
       SPH    DIST                 ;Save distance*0.1 in 1st location
```

The distance or cost values are stored in an 8-word DIST table. Each element of the DIST table corresponds to a path state. The order of storage in the table shown in Figure 12 is not a simple ascending or descending form. The reason for this scrambled order is explained later.

Figure 10. Delay State Linking



The next step is to accumulate the cost (or distance) for each delay state at the current time. As previously explained, at every time interval there are eight delay states (SO, S1, S2). Each delay state at the current time interval is linked to four delay states from the previous time interval, as shown in Figure 10. The minimum cost link is identified, and the distance value of the selected link is added to the accumulated cost of the delay state from which it originates. This gives the accumulated cost of the current delay state.

In addition to the accumulated cost, the following information needs to be stored for each delay state:

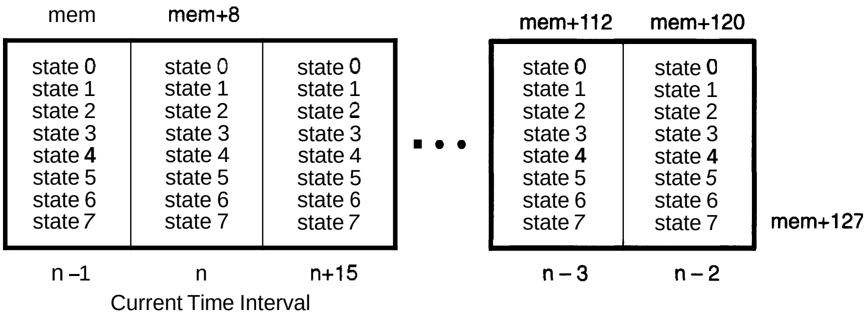
The path state that identifies the link selected
The delay state of the previous time interval that is linked to the current delay state

The code to perform these functions is:

```
STATE0:
       RPTB   ENDB0—1           ;For (i=0;i<=3;++i)
       LACC   *,0,AR2           ;Get prev. accumulated distance
       ADD    *,AR5             ;Add current distance
       CRLT                     ;If acc < prev. largest
       NOP                      ;Then
       XC     2,C               ;Update PAST-DLY & PAST-PTH locations
       SAR    AR1,*,AR6         ;Pointer to ACCDIST —> PAST-DLY
       SAR    AR2,*,AR1         ;Pointer to DIST —> PAST-PTH
       MAR    *,AR1             ;ARP = AR1
       MAR    *+,AR2            ;AR1++ (circular addressing)
       MAR    *+,AR1            ;AR2++ (circular addressing)
ENDB0:
```

Pointers to the past path and delay states are stored in the PAST–PTH and the PAST–DLY tables. Since the decoder bases its decision on the path history of the previous 15 time periods, these two tables span 16 time periods (including the current time period). The length of each table is 128 words (16 time periods x 8 states). At each time interval, the GET_ACC_DIST routine adds new information to the tables and discards the oldest eight states. The format of these tables is shown below.

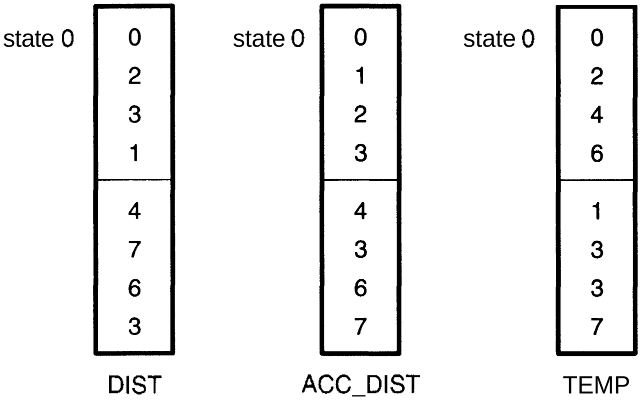
Figure 11. 128-Word Circular Buffers — Format of PAST-PATH and PAST-DLY Tables


mem
mem+8
mem+112
mem+120
state 0 state 1 state 2 state 3 state 4 state 5 state 6 state 7
state 0 state 1 state 2 state 3 state 4 state 5 state 6 state 7
state 0 state 1 state 2 state 3 state 4 state 5 state 6 state 7
state 0 state 1 state 2 state 3 state 4 state 5 state 6 state 7
state 0 state 1 state 2 state 3 state 4 state 5 state 6 state 7
mem+127
n–1
n
n+15
n – 3
n – 2
Current Time Interval

Both tables are set up as 128-word circular buffers. Each of them is divided into 16 macro elements corresponding to 16 time intervals. Each macro element stores the state history of one time interval. A pointer is set up to indicate the location of the current time interval. By stepping through each macro element, a path can be traced backward in time.

Consider the V.32 trellis diagram again (see Figure 4). Notice that all even-numbered delay states of the current time interval have links to the first four delay states of the previous time interval. Similarly, all odd-numbered new delay states have links to the last four delay states. For instance, the new delay state 0 can be reached from the past delay states 0 – 3, and the new delay state 1 can be reached from the past delay states 4 – 7. So it is relatively simple to process even- and odd-numbered states in two groups. Furthermore, even-numbered delay states can be reached only by the first four path states, and odd-numbered delay states can be reached only by the last four path states.

Figure 12. DIST Table Structure



If the elements of the DIST table are set up as shown in Figure 12, all the path-state sequences can be generated from the same table. Four-word circular buffers are set up, comprising upper and lower halves of the DIST and ACC–DIST tables. By incrementing or decrementing through these circular buffers, path and delay-state sequences can be generated for each new delay state. (See the GET_ACC_DIST routine in the source code.) For each new delay state, only four past delay states and path states need to be accessed. The table for past delay states (ACC–DIST) is set up as a circular buffer so that after accessing four elements of the table, the pointer is automatically reset to the first element for the next iteration.

Once least-cost links to the eight delay states are identified and stored in appropriate tables by the MIN_ACC_DIST routine, the accumulated distance table ACC–DIST is updated with new accumulated distances. To avoid overflow, new accumulated distance is computed according to the following equation:

$$\text{new acc dist} = 0.9 \times \text{old acc dist} + 0.1 \times \text{dist} \tag{4}$$

Note that this is a simple IIR implementation of a low-pass filter. The coefficients of Equation (4) can be modified to control the decay time of this low-pass filter.

There are eight independent tracks whose path histories are maintained in the PAST–PTH and PAST–DLY tables. The track that has the least accumulated cost (or distance) at this point is traced back for 16 time periods to determine the decoder output at that time. This task is performed by the GET–PATH routine as shown below. After 15 iterations, the delay state that corresponds to oldest link of the track is found.

```
    RPTB     TLOOP-1          ;for i=0,i<=15,i++
    MAR      *0+              ;offset by state for prev. time period
    LACC     *0-              ;get next pointer & reset ARO to state 0
    SUB      #ACCDIST         ;subtract #ACCDIST to get next state
    SAMM     INDX             ;save next state
    SBRK     7                ;move AR0 7 locs back to avoid skipping CBER1
    SBRK     1                ;now AR0 is correctly positioned 1 time period
TLOOP:                        ;back (circular addressing)
```

The format of the PAST–PTH table is identical to the PAST–DLY table except that it contains previous path states instead of previous delay states. Also, the two tables are contiguous in data memory. Hence, by adding 128 to the pointer of the PAST–DLY table, corresponding path states can be accessed in the PAST–PTH table. The 3-bit path state (YO, Y1, Y2) that corresponds to the oldest link is the output of the decoder. Since the path-state table DIST is not in a simple order, a short table look-up routine performs the descrambling of the output.

The 3-bit path state output by the Viterbi algorithm identifies a set of four points on the **V.32** constellation. Of these four points, the one that is closest to the actual input (at that time period) should be selected. A table must be set up in memory that stores the decoder input for the last 16 time periods so that the oldest input can be compared with these four constellation points. Fortunately, this cycle-consuming function can be avoided entirely by recalling that this comparison operation was done earlier (16 time periods back, to be exact) using the REGION table. If the pointer to the REGION table that identifies the eight closest constellation points (for each one of the path states) is available for that time interval, it is a simple matter to select a constellation point according to the path state number 0–7.

A 16-word circular table PATH–TBL is set up that stores pointers to the REGION table for the last 16 time periods. Since this table is always accessed sequentially (as opposed to randomly), the bit-reversed addressing mode is used to implement this circular buffer. The resulting 5-bit symbol (YO, Y1, Y2, Q3, Q4) is the actual output. Obviously, YO, the redundant bit, does not contain useful information (as it has already served its purpose) and can be discarded now.

Finally, the differential decoding algorithm (DIFF routine) converts Y1 and Y2 to Q1 and Q2. The following equations describe this decoding process:

$$Q1_n = Y1_n \oplus Y1_{n-1} \tag{5}$$

$$Q2_n = (Q1_n \bullet Y1_{n-1}) \oplus Y2_{n-1} \oplus Y2_n \tag{6}$$

A table look-up approach is taken here to decrease the execution time of this routine. A 16-word table DIFF_TBL is set up in memory. Each element of this table corresponds to a unique combination of bits [$Y1_{n-1}$ $Y2_{n-1}$ Y1, $Y2_n$], and it contains resulting decoded bits $Q1_nQ2_n$. Refer to the source code listing; see the ***Code*** Availability section on page 100. These two bits combined with $Q3_n$ and $Q4_n$ result in a 4-bit output symbol (Q1, Q2, Q3, Q4).

Performance Analysis

The V.32 encoder/decoder performance is evaluated on the TMS320C5x Software Development System (SWDS)[2]. The code benchmarks are also computed with the help of TMS320C5x SWDS. The transmission channel characteristics are simulated using the MATLAB software.

The input to the V.32 encoder is a binary data stream. As previously discussed, the stream is divided into 4-bit contiguous blocks called symbols. From the encoder standpoint, the input data is random, but the resulting 5-bit output symbols are not entirely random. Due to the convolutional encoding done on two bits of each 4-bit input symbol, output symbols are restricted within a subset of 32 symbols, depending on past symbol history.

The QAM modulator modifies the amplitude and the phase angle of the transmitted carrier signal according to each 5-bit symbol it receives. The communication channel imperfections distort the transmitted signal. White noise, impulse noise, and phase reversals are the most commonly encountered sources of channel distortion in telephony.

---

[2] Since the writing of this paper, the 'C5x SWDS has been replaced with the 'C5x evaluation module (EVM) for code development.

The information is carried by the amplitude/phase of the transmitted carrier or, equivalently, by the I and Q components of it.

$$S(t) = amplitude \times \cos(wt + phase) \tag{7}$$

$$= I \times \cos(wt) + Q \times \sin(wt) \tag{8}$$

The I and Q components of the signal received by a V.32 modem are corrupted with channel noise. If the channel is modeled as an AWGN-type channel, it is simple to simulate its effect on the signal by adding controlled Gaussian noise to the I and Q components independently. If $N(t)$ is the zero-mean white noise signal, the signal-to-noise ratio (SNR) of QAM modulated signal $S(t)$ is given by

$$SNR\ (dB) = 10 \times \log_{10} \left[\frac{variance\ of\ S(t)}{variance\ of\ N(t)}\right] \tag{9}$$

$$= 10 \times \log_{10} \left[\frac{E\ [S^2(t)]}{E\ [N^2(t)]}\right] \tag{10}$$

With the assumption that the I and Q inputs are statistically independent of each other, the SNR equation for the QAM modulated signal can be simplified as

$$SNR\ (dB) = 10 \times \log_{10} \left[\frac{variance\ of\ I}{variance\ of\ N_i}\right] \tag{11}$$

$$= 10 \times \log_{10} \left[\frac{variance\ of\ Q}{variance\ of\ N_q}\right] \tag{12}$$

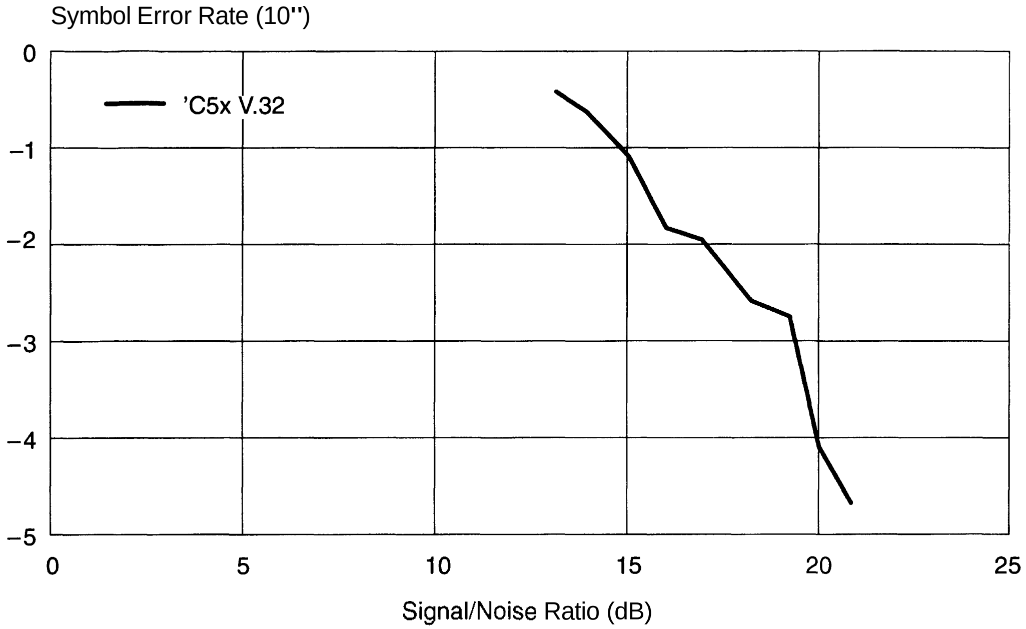
where N_i and N_q are additive noise signals for the I and Q input signals, respectively. Fixed-length sequences of I and Q are generated, and their sample variances are computed using the MATLAB software. For each desired value of SNR, required variances of N_i and N_q are calculated using Equations (9) through (12). Once the variances of N_i and N_q are determined, zero-mean Gaussian noise sequences N_i and N_q are generated by MATLAB. The input to the decoder program consists of I and Q data added to the respective noise sequences, N_i and N_q. This allows measuring the SNR performance of the decoder.

Figure 13 illustrates the performance of V.32 encoder/decoder code for various SNRs. These results are based on an input data sequence length of 4000 symbols. The yardstick for the performance measurement is symbol error rate (SER), which is defined as:

$$SER = \frac{total\ number\ of\ symbol\ errors}{total\ number\ of\ input\ symbols\ received} \tag{13}$$

Note that each input symbol consists of four bits.

Figure 13. White-Noise Impairment — Simulation Results



There are several factors that affect the performance of a Viterbi decoder in the presence of noise. One is the length of the path history analyzed by the decoder before selecting the most likely output. In general, it should be four or five times the encoder constraint length. Further increase in path history length gives only marginal improvement in performance.

Another performance factor is the decay time of the low-pass filter that is used to accumulate distance. By decreasing its time constant, the decoder can be made to respond to short noise bursts in the channel.

The table of eight accumulated distance values provides a convenient way of monitoring the performance of the decoder (and noise activity in the channel) in the absence of any prior knowledge of incoming data. Recall that these eight accumulated distance values allow the selection of minimum cost path at every symbol time interval. These values are also updated as new data is processed. During the relatively noise-free periods of transmission, it is observed that only one of the eight distance values remains significantly smaller than the rest. This in turn forces the decoder to select one particular path at every time interval. As the signal deteriorates, the difference between the minimum value and the rest of the table contents decreases. At some point, all distance values become so much alike that the decoder can no longer identify the correct path. This is the stage in which the BER increases considerably.

Table 1. Program Benchmarks

	Speed And Memory Requirements		
	Code Size (in Words)	Data Size (in Words)	CPU Loading per Symbol, Excluding Initialization (in Machine Cycles)
V.32 Encoder	79	10	90
V.32 Decoder	768	837	963-973

Table 1 shows the code size, data size, and CPU loading of the V.32 encoder/decoder program. This is by no means a fully optimized implementation of V.32 on the TMS320C5x. This code is written with the basic aims of demonstrating the capabilities of the TMS320C5x digital signal processor family and providing system designers with a head start on V.32 modem design. Table 2 and Table 3 present memory and speed requirements for various modules of the encoder and decoder. There are several speed-vs.-memory issues that can best be resolved by the system designer. The following paragraphs highlight some of them.

Table 2. V.32 Encoder Code

No	Function Name	Code Words	Machine Cycles
1	START	8	9
2	UNPACK	6	15
3	DIFF	11	12
4	ENCODE	20	21
5	PACK	12	20

Table 3. V.32 Decoder Code

No	Function Name	Code Words	Machine Cycles
1	RD_DATA	17	22
2	GET–RGN	108	80 - 112
3	GET–CUR–DIST	136	142
4	GET_ACC_DIST	228	489
5	MIN_ACC_DIST	36	65
6	GET–PATH	12	132
7	GET_SYM	11	15
8	DIFF	21	24

The approach that should be taken wherever speed-vs.-memory tradeoffs exist is to optimize for speed. For instance, the GET–RGN function uses a 416-word table to identify the eight closest constellation points. As discussed in the *Algorithm Implementation on the TMS320C5x* section on page 88, an alternate approach is to compute the distance between each constellation point and the current input and select the minimum distance point.

In the GET–CUR–DIST routine, distances corresponding to eight path states are computed by inline code, as opposed to looped code. This is done to facilitate the scrambled order of storage in the DIST table (see Figure 12). A considerable amount of program space may be released (approximately 100 words) if looped code is used here at the cost of additional machine cycles required to set up the loop and to access the DIST table.

In contrast with the GET–CUR–DIST routine, the GET_ACC_DIST routine is very difficult to implement in loop form. Each delay state computation itself makes use of iterative code. Furthermore, path-state sequences are unique for each delay state.

Summary

The TMS320C5x provides a powerful DSP engine for data-communication applications. This application report presents an efficient implementation of data encoding and decoding algorithms for V.32 modems on the TMS320C5x.

The encoder and decoder source code is designed with a generic hardware interface in mind. System designers can modify the input/output modules to suit their hardware requirements. The encoder algorithm is fairly straightforward. Most of the number crunching is required by the decoder algorithm. Although the code is written for the V.32 modem standard, a conscious effort is made to point out the V.32-specific and general-purpose Viterbi functions for adaptation of the code to any other Viterbi decoding scheme. For the same reason, the program flow is discussed in considerable detail.

Assembly code can be run on TMS320C50/1 in real time, without requiring any external memory. On a 35-ns TMS320C5x, the entire code only takes approximately 8% of the CPU time.

Code Availability

The associated program files are available from Texas Instruments TMS320 Bulletin Board System (BBS) at (713) 274-2323. Internet users can access the BBS via anonymous ftp at *ti.com.*

References

Michelson, A. M., and Levesque, A. H., *Error-Control Techniques for Digitul Communications*, John Wiley & Sons, 1985.

Clark, G. C., and Cain, J. B., *Error Correction Coding for Digital Communications*, Plenum Press, 1981.

3. Proakis, J. G., *Digital Communications,* McGraw-Hill Book Company, 1983.

4. Forney, G. D., Jr., "The Viterbi Algorithm", *Proceedings Of The IEEE,* March 1973.

5. Viterbi, A. J., "Error Bounds for Convolutional Codes and An Asymptomatically Optimum Decoding Algorithm", *IEEE Transactions, Infinity Theory,* April 1967.

Lin, S., and Costello, D., *Error-Control Coding: Fundamentals and Applications,* Prentice-Hall, 1983.

TMS320C5x User's Guide, Texas Instruments, 1993

8. "Report on the Work of Study Group XVII During the Period 1981–1984 Part III: Proposed New or Revised Recommendations in V-Series", *CCITT*, Malaga-Torremolinos, 1984.

MATLAB User's Guide, The Math Works, Inc., 1989.

A TMS320C53-Based Enhanced Forward Error-Correction Scheme for U.S. Digital Cellular Radio

Mansoor A. Chishtie
Digital Signal Processing Applications — Semiconductor Group
Texas Instruments Incorporated

Abstract

This report presents an enhanced forward error-correction scheme that complies with the U.S. Digital Cellular (USDC) standard. The proposed scheme uses a generalized Viterbi algorithm (GVA) based on N. Seshadri and C-E. W. Sundberg's, "Generalized Viterbi Algorithm for Error Detection with Convolutional Codes" [1] that produces an ordered list of N globally best estimates of the transmitted sequence. The scheme uses the GVA to enhance performance of the USDC voice channel decoder and is implemented on the TMS320C53 fixed-point digital signal processor (DSP). This paper shows that the 'C53 implementation of the algorithm does not require significant increase in computational overhead when compared to a standard Viterbi algorithm.

Introduction

The second-generation U.S. cellular radio telephone system (IS-54 standard) is based on digital technology. To increase system capacity and improve speech quality, the voice channels use digital transmission for both forward and reverse radio links. Each radio channel is shared by at least three mobile units through a time-division multiple access (TDMA) scheme. Elaborate forward error-correction (FEC) techniques are employed to operate these radio links reliably under low carrier-to-interference (C/I) ratio and high data-transfer rate. The IS-54 standard combats channel noise by using systematic cyclic redundancy check (CRC) codes, convolutional encoding, and frame interleaving techniques on transmitted data. Although the standard does not recommend any particular decoding algorithm, the Viterbi algorithm (VA) is most commonly used by system designers.

Various generalizations of the original Viterbi algorithm have been presented in the literature [1, 2, 5, 6]. These schemes provide enhanced performance over the conventional Viterbi algorithms for a number of applications, including automatic repeat request (ARQ) schemes, concatenated codes, coded Viterbi equalization, etc. One scheme [2] modifies the VA to deliver a reliability value for each bit in the most likely path sequence. This algorithm is useful for Viterbi equalization on IS-54 radio channels, leading to an improved performance of the outer VA performing the FEC. Another scheme [1] generalizes the VA to find N globally best estimates of the transmitted sequence. It is shown here that this algorithm is particularly appropriate for the rate-112 framed convolutional encoder used by the IS-54 voice channel. This GVA is implemented on the TMS320C53 fixed-point DSP.

Algorithm Description

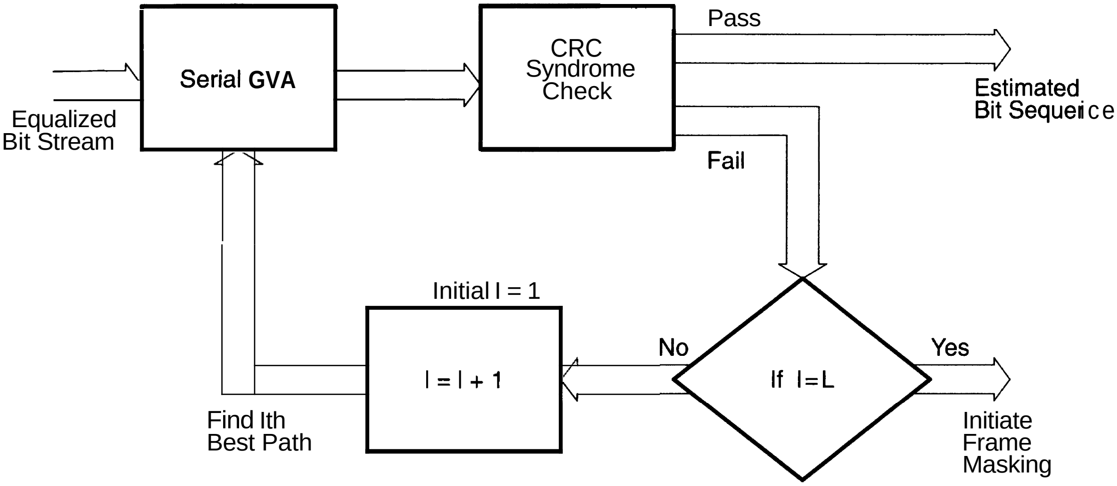
It is well known that bit errors usually occur in bursts in Viterbi decoders. If you know the globally second best path, the third, etc., you can use this information to reduce the burst error rate under noisy conditions. To select N best paths simultaneously, N best survivors (out of 2N for a rate-llm code) at each state must be retained during the forward pass through the trellis. This is referred to as *parallel* GVA in [1]. However, for this application, only one path is estimated at a time. If requested, the *serial* GVA [1] produces the nth best path on the basis of the previous $n-1$ best paths.

The serial GVA algorithm, as shown in Figure 1, is selected for the voice channel FEC because it results in reduced memory requirement and less computational overhead, as discussed in the next section.

The IS-54 voice channel uses a concatenated coding scheme in which each message frame is divided into two bit classes: class I and II. For the twelve perceptually most important bits of class I, a 19-bit systematic CRC code is generated. These bits, along with the rest of the class I bits. form input to a rate-112 framed convolutional encoder with a constraint length of 5. The class II bits remain uncoded.

The frame size of the encoder is 89 bits, including five tail bits. The encoder always starts and ends in state 0. The serial **GVA** decoder maintains a state path history as it expands the trellis in the forward pass. It also sets up two accumulated metric tables, accuml and accum2, for the two globally best paths. Since the trellis expands from initial state 0, the first element of the accum1 table is initialized to 0, and the rest of the table is set to a large positive number for a distance-type metric (or a large negative number for a correlation-type metric). Similarly, the second table, accum2, is also initialized to a large positive number, except for the first element, which, in this case, is initialized with a positive integer N.

Figure 1. IS-54 Voice-Channel GVA Algorithm



The first pass of the algorithm produces the globally best estimate of the transmitted sequence. The algorithm, in this case, is identical to a conventional **VA,** with one exception: it updates the state count array as it traces the best path back in time. This state count array is used for any subsequent invocations of the **GVA.** Each element h_{ij} of the state count array uniquely represents state **i** in time interval **j** and indicates how many of the previously identified n–1 best paths pass through state i in time j. When the **GVA** is invoked for the first time, the state count array is initialized to 0. During trace-back of the best path, corresponding elements of the array are incremented by 1.

To find the second best path, the trellis is expanded again; however, this time, the second best path (out of four possible survivors) that enters state i in time j whose h_{ij} != 0 is retained. For the states whose corresponding h_{ij}s are 0 (that is, states that are not included in the globally best path), the best survivor is retained. Note that, in this case, no processing is required because the state path table already contains the history of the best path. During the trace-back phase for the second best path, elements of the state count array that corresponds to the path are incremented by 1. This procedure is repeated for n best paths.

For the rate-112 voice channel coding, two survivors normally leave and enter state i at any given instant. The better of the two paths entering state i is retained for further expansion. However, for the second best path estimation, four links are considered. Two links each from state i and state i+16 in time j-1 enter state **i** in time j. Two links are retained for further expansion. The accumulated metric tables, accuml and accum2, represent the two survivors for state i. The difference between the initial state 0 metrics of the two best paths (that is, accum2 – accuml = N) serves to maintain an initial offset between the two most likely paths. This allows the two paths to possibly diverge later in time. The actual value of N is system-dependent and can be determined experimentally.

The knowledge of the second best path, the third, etc., is utilized by the voice channel decoder in this way: if the CRC syndrome is nonzero for the best path, the decoder output contains errors. In this case, the second best estimate of the transmitted data is considered. If the CRC check is successful on this estimated sequence, then it is selected as the decoder output. Otherwise, the next best path is considered. This procedure is repeated either until an estimated sequence with zero syndrome is found or until the L best candidates fail. In case of failure, the current speech frame is marked bad, and a frame-masking procedure is initiated as specified by the IS-54 standard.

Implementation Details

Programmable DSPs are widely used in digital cellular mobile unit and base station designs. The high-performance TMS320C5x is especially designed for digital cellular applications. The newest member of this generation, the TMS320C53, provides a low-cost, low-power DSP engine with more than 20K words of on-chip memory. Its 35-ns fast instruction cycle time, large on-chip memory, and programmable power-down modes make it especially suitable for hand-held telephone designs.

The GVA is implemented on a TMS320C53. The 'C53 min/max instructions facilitate a search algorithm for trellis expansion. Its dynamic bit testing and zero-overhead loops efficiently implement a trace-back routine.

The serial GVA algorithm is chosen for two primary reasons:

- The relatively insignificant increase in computational overhead when compared to a conventional VA
- Less memory usage compared to other types of GVAs

The first pass of the GVA algorithm (that is, search for the best path) is identical to a conventional VA. The only additional overhead is the update of the state count array during the trace-back stage. The second pass of the GVA (if required) is more complicated. In this case, two out of four possible survivors are selected. This normally requires a binary search of the accum1 and accum2 tables (for a total of five comparisons). However, when an ordered list of accumulated metric tables is maintained, only two comparisons are required. Moreover, comparison is required only for the trellis points for which h_{ij} is nonzero, as previously discussed. Table 1 summarizes the result of a TMS320C53 implementation of conventional VA and serial GVA algorithms. Although the serial GVA takes longer to find the second best path, it is required to do so only if the CRC syndrome fails on the best path. Therefore, the computational requirement of the serial GVA averages out over varying channel conditions.

Table 1. Algorithm Execution Time on a 35-ns TMS320C53

		Serial GVA	
	Conventional VA	**Best Path**	**Second Best Path**
Trellis Expansion	1.15 ms	1.15 ms	3.1 ms
Trace-back	31.15 µs	46.7 µs	46.7 µs

The other advantage of this algorithm is its conservative memory requirement. The two main system design constraints of a portable dual-mode phone are small form factor and low power consumption. Both preclude a design from having a large amount of expensive static RAMs. Since the algorithm serially finds the globally best estimates, there is no need to save path histories of the previously found paths. Therefore, one state path history buffer suffices for this application. Table 2 compares the memory requirement of a serial GVA that finds two best paths with the memory requirement of a conventional VA. Both algorithms are implemented on a TMS320C53 processor.

Table 2. Memory Requirement

	Conventional VA[†]	GVA[†]
State Path History	192	192
State Count	–	192
Accumulated Metric	32	64[‡]

[†] Number of 16-bit words required
[‡] For best and second best paths

Results

The performance of the GVA in comparison with the conventional VA is shown in Figure 2. The modulation scheme used is phase-shift keying (PSK). The results are measured over a simulated additive white Gaussian noise (AWGN) channel. Figure 3 shows the path history of the voice channel encoder for a sample input sequence. It also shows the best estimated path and the second best path traces. Note how the second best path diverges from the best path briefly and remerges with it subsequently. If the best path diverges only once from the actual encoder path, it is likely that the second best path will match the encoder path.

Figure 2. Simulated Bit Error Rate of Serial GVA Versus VA

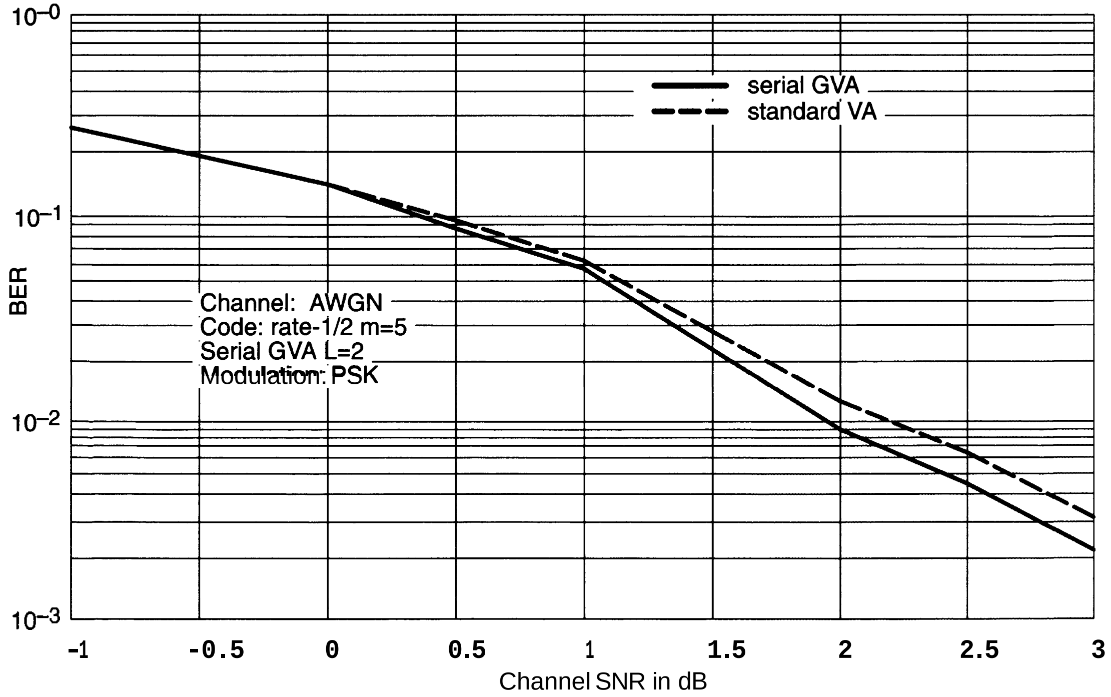
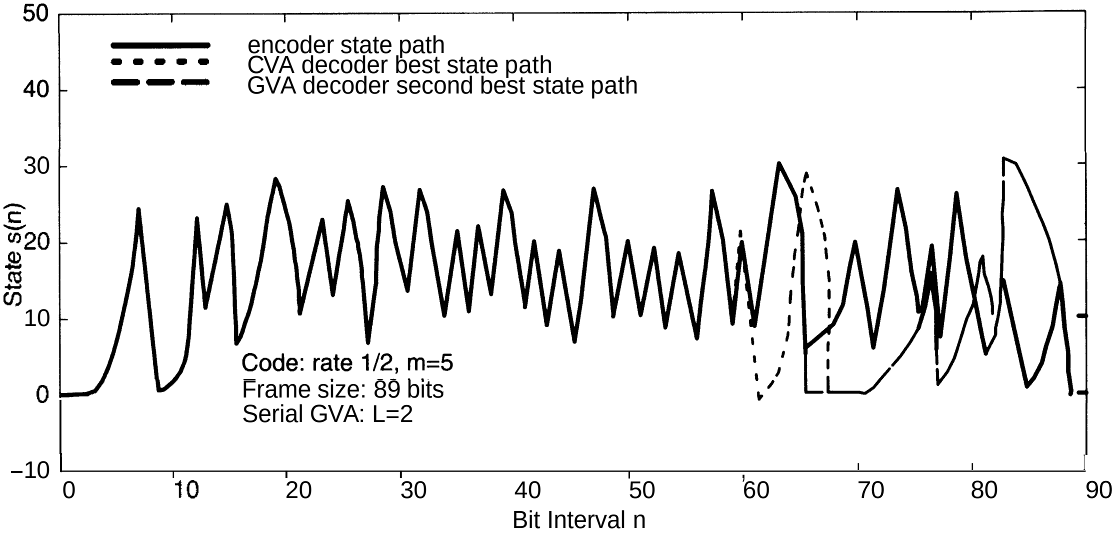


Figure 3. State Path History Trace



Conclusions

In general, modified Viterbi algorithms offer improved performance of a forward error-correction design at the expense of more computational overhead and added complexity. This paper presents an FEC subsystem for a USDC voice channel that uses a generalized Viterbi algorithm [1] to combat bit errors under noisy channel conditions. It shows that the proposed FEC design performs better than a standard Viterbi-based design. Furthermore, the FEC design does not require significant increase in memory space and processing power. The algorithm is implemented on a digital signal processor, the TMS320C53. The experimental results indicate that even when the proposed algorithm is restricted to two best estimates of the transmitted sequence (that is, L = 2), its bit error rate is less than that of a standard Viterbi algorithm operating under similar channel conditions. Further performance improvement is achievable if more than two estimated sequences are generated.

References

1. Seshadri, N., and Sundberg, C-E. W., "Generalized Viterbi Algorithm for Error Detection With Convolutional Codes", *GLOBECOMM '89 Conference Records,* pp. 43.3.1 – 43.3.4.

2. Hagenauer, J., and Hoeher, P., "A Viterbi Algorithm With Soft-Decision Outputs and Its Applications", *GLOBECOMM '89 Conference Records,* pp. 47.1.1 – 47.1.7.

3. *Cellular System: Dual-Mode Mobile Station – Base Station Compatibility Standard,* IS-54 Project Number 2215, Electronic Industries Association, December 1989.

4. *TMS320C5x User's Guide,* Texas Instruments, 1993.

5. Hashimoto, T., "A List-Type Reduced-Constraint Generalization of the Viterbi Algorithm", *IEEE Transactions on Infinity Theory,* Volume IT–33, November 1987, pp. 866–876.

6. Schaub, T., and Modestino, J.W., "An Erasure Declaring Viterbi Decoder and its Applications to Concatenated Coding Systems", *ICC '86,* 1986, pp. 1612–1616.

Part V
Baseband Modulation and Demodulation

IS-54 Digital Cellular Modem Implementation on the TMS320C5x

Balaji Srinivasan
Digital Signal Processing Applications — Semiconductor Group
Texas Instruments Incorporated

Introduction

Digital cellular and digital mobile radio communication are today's key topics in the communications field. Digital mobile cellular communication systems are being introduced in the U.S., Canada, Europe, Japan, and many other countries. Various standards like the U.S. Digital Cellular (USDC), Global System for Mobile Communications (GSM), and Personal Digital Cellular (PDC) have been proposed in different countries for the development of a mobile cellular communication system. The U.S. Digital Cellular standard is specified by the Telecommunications Industry Association (TIA). The TIA has specified π/4-DQPSK as the new modulation standard for the emerging U.S. digital cellular communication systems. The focus of this report is on the theory and implementation of the π/4-DQPSK modem on the TMS320C5x DSP. The TMS320 family of DSPs is well suited for such modem applications. The advanced features of the 'C5x have made the high-data-rate modem implementation possible. This report is organized into the following topics.

- Description of π/4-QPSK modulation scheme
- Theory of the π/4-DQPSK modem
- Modem implementation on the TMS320C5x
- Performance results
- Summary

The key features of the TMS320C5x that provide excellent code efficiency and ease of implementation are discussed in the *Modem Implementation on the TMS320C5x* section on page 119.

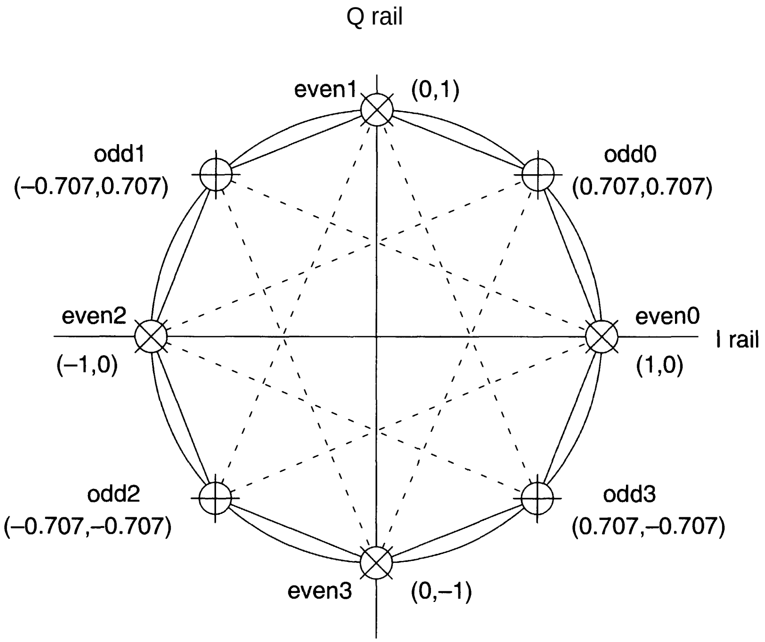
Description of d4-QPSK Modulation Scheme

A study of various modulation schemes like QPSK, OQPSK, GMSK, and TFM have been made, and attention has been focused on the use of linear modulation techniques for nonlinearly amplified systems to meet both the power and spectral efficiency requirements of mobile cellular systems. There has been a search for alternative unstaggered linear modulation systems that have low envelope fluctuation. After a thorough analysis, π/4-QPSK was proposed as the standard modulation technique to be used in the digital cellular environment. π/4-QPSK is an unstaggered modified version of QPSK with two sets of constellations totaling eight constellation points. This modification to QPSK has carrier phase transitions that are restricted to ± π/4 and ± 3π/4. Since the phase does not undergo instantaneous ± π transitions as in QPSK, the envelope fluctuation at the output is significantly reduced. Also, as this is not a staggered/offset scheme, coherent as well as noncoherent detection can be applied to π/4-QPSK. It has been shown that the spectral efficiency obtained is twice that obtained by two-level digital FM, GMSK, or TFM, which are constant envelope modulation techniques.

The π/4-shifted-QPSK signal constellation can be viewed as the superposition of two QPSK signal constellations offset by 45° relative to each other, resulting in eight signal phases. Symbol phases are alternately selected from one of the two QPSK constellations, and as a result, successive symbols have a relative phase difference that is one of the four angles, ± π/4 and ± 3π/4. Figure 1 illustrates the π/4-shifted-QPSK signal constellation and the various possible phase transitions. As Figure 1 shows, two constellation sets, one with four possible phases (0, π/2, π, and ±π/2) and the other with another four possible phases (π/4, 3π/4, –3π/4, and ±π/4) are used in the actual modulation. There is a relative n14 shift between the two constellation sets; hence, the name π/4-shifted QPSK.

First, the input data is buffered into one of the four possible dibit symbols (namely, 00, 01, 10, or 11). Then. for odd numbered symbols, the output signal phase is chosen from one of four possible phases of the constellation set ⊕; for even numbered symbols, the output signal phase is chosen from one of four possible phases of the constellation set ⊗. The choice of the particular phase within a constellation set depends on the dibit input. As usual, to reduce dibit errors in the receiver, Gray coding of dibits is done prior to phase selection from a chosen constellation set. This alternate selection of a constellation set can be reversed for odd and even numbered symbols. In conventional QPSK, only one of the constellation sets is chosen. Due to the change of constellation sets in $\pi/4$-shifted QPSK, eight signal constellation points are possible. Although eight constellation points are seen in the constellation diagram and they look like the 8-PSK signal constellation, the choice of signal phases for every symbol is only four; hence, it is still a 4-phase QPSK. In conventional QPSK, the possible phase transitions were 0, $\pm \pi/2$, and π. Here, the possible phase transitions are only $\pm \pi/4$ and ± 3x14, thereby reducing the envelope fluctuations of the modulated output signal. Envelope fluctuations are very important since demodulation becomes difficult when the signal is amplified by nonlinear amplifiers (which are common in cellular systems). An OQPSK (offset QPSK) scheme reduces the fluctuations but restricts the type of demodulation scheme to be coherent. Noncoherent demodulation has certain advantages in the cellular systems, and $\pi/4$-shifted QPSK allows the flexibility to use either coherent or noncoherent demodulation. If differential encoding is also performed prior to signal mapping, the scheme becomes $\pi/4$ DQPSK.

Figure 1. $\pi/4$-Shifted QPSK Signal Constellation



Theory of the π/4-DQPSK Modem

Basic Modem Specifications

The specifications for the U.S. digital cellular modem were set by the TIA. A few of the specifications that are relevant to this application are:

- **Mode of operation**
  - 30-kHz channel structure, each channel operating on TDMA burst mode
  - Gross bit rate of 48.6 kbps
- **Modulation**
  - π/4-shifted differentially encoded quadrature phase shift keying
  - Gray coding used in signal mapping to reduce dibit errors
  - Spectral shaping to limit adjacent channel interference
  - No specific implementation method
- **Baseband filtering**
  - Square-root raised-cosine pulse-shape frequency response
  - Linear phase response
  - Roll-off factor for square-root pulse shaping filter to be 0.35
  - No specific implementation method
- **Demodulation**
  - Any coherent or noncoherent demodulation method
  - No carrier-related specifications (TMS320C5x implementation is a baseband modulation and demodulation)

Modulator

The theory behind signal mapping and baseband filtering for modulation is reproduced from the TIA document [7] here. The block diagram of the π/4-shifted DQPSK modulator is shown in Figure 2. The input 48.6-kbps data stream is converted into symbols as dibits A_k (odd bit) and B_k (even bit). Then the information is differentially encoded (symbols are transmitted as changes in phase between two successive symbols rather than as absolute phases) and mapped into one of the signal phases from either of the two signal constellations described in the ***Description of π/4-QPSK Modulation Scheme*** section on page 113. The symbols can be first differentially encoded and then mapped into a signal phase as a two-step process, or they can be combined into a single step with a set of equations. The digital data sequences A_k and B_k are encoded as I_k and Q_k according to the following set of equations.

$$I_k = I_{k-1} \cos[\ \Delta\phi(A_k, B_k)\ ] - Q_{k-1} \sin[\ \Delta\phi(A_k, B_k)\ ] \tag{1}$$

$$Q_k = I_{k-1} \sin[\ \Delta\phi(A_k, B_k)\ ] + Q_{k-1} \cos[\ \Delta\phi(A_k, B_k)\ ] \tag{2}$$

I_{k-1} and Q_{k-1} are the previous symbol's I and Q values. $\Delta\phi(A_k, B_k)$ is the phase change in the kth symbol interval and is determined according to Table 1. The phase change values are Gray coded.

Table 1. Phase Calculation

A_k	B_k	$\Delta\phi$	$\cos(\Delta\phi)$	$\sin(\Delta\phi)$
0	0	$+\pi/4$	+	+
0	1	$\pm\pi/4$	+	–
1	0	$+3\pi/4$	–	+
1	1	$-3\pi/4$	–	–

Simple trigonometric manipulation easily shows that Equations (1) and *(2)* are derived from

$$I_k = \cos[\phi_k] = \cos[\phi_{k-1} + \Delta\phi(A_k,B_k)] \tag{3}$$

$$Q_k = \sin[\phi_k] = \sin[\phi_{k-1} + \Delta\,\phi(A_k,B_k)] \tag{4}$$

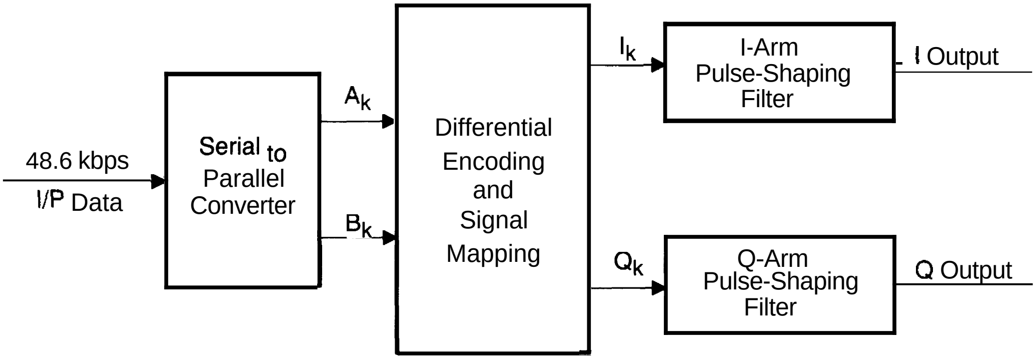
where ϕ_k and ϕ_{k-1} are the absolute phase angles corresponding to the kth and (k–1)th symbol intervals, respectively.

The signals I_k, Q_k at the output of the differential phase encoding block can take one of the five values 0, ± 1, or $\pm\frac{1}{\sqrt{2}}$, as seen from the constellation of Figure 1. Impulses I_k, Q_k are applied to the I and Q baseband pulse-shaping filters. The baseband filters have linear phase and square-root raised-cosine frequency response of the form:

$$|H(f)| = \begin{cases} 1 & : f \le (1-\alpha)/2T \\ \sqrt{0.5\left(1-\sin\left[\pi(2f-1)/2a\right]\right)} & : (1-\alpha)/2T \le f \le (1+\alpha)2T \\ 0 & : f > (1+\alpha)/2T \end{cases} \tag{5}$$

where T is the symbol period. The roll-off factor, a, determines the width of the transition band and is 0.35 as per the specifications.

Figure 2. Modulator Block Diagram


48.6 kbps
I/P Data
Serial to Parallel Converter
A_k
B_k
Differential Encoding and Signal Mapping
I_k
Q_k
I-Arm Pulse-Shaping Filter
Q-Arm Pulse-Shaping Filter
I Output
Q Output

The baseband-filtered I and Q signals are then multiplied by the carrier and transmitted over the channel. The implementation on the TMS320C5x is a baseband modem, hence; the carrier is not included as part of the modulator block diagram.

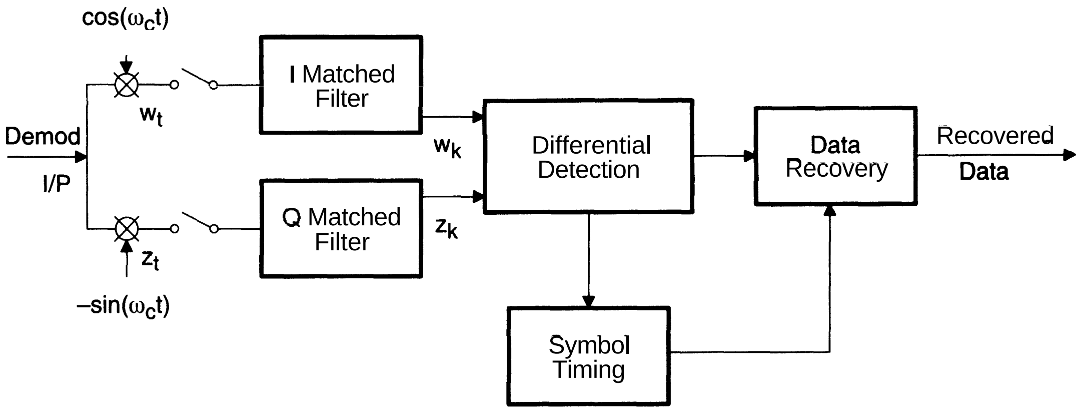
Demodulator

Digital communication systems that operate in power- and bandwidth-limited channels generally employ coherent detection that involves carrier-recovery technique. In a Rayleigh-faded mobile channel with

AWGN, coherent systems have a significant advantage in power efficiency and performance over the noncoherent demodulation involving differential or delay detection techniques. But in a mobile environment, disturbances such as multipath fading, Doppler frequency shifts, and phase noise are present. Coherent detection, which is based on the carrier frequency and phase lock, may suffer disadvantages over the noncoherent detection, though coherent detection has 3-dB power efficiency. Additionally, the noncoherent detection makes the receiver design simpler.

Since the current implementation on the 'C5x is a baseband modem that does not involve the carrier, and since noncoherent demodulation offers significent advantages, baseband differential detection has been chosen as the implementation technique on the 'C5x. Note that the TIA has not recommended any specific demodulation method.

Figure 3. Demodulator Block Diagram



The block diagram of the demodulator is shown in Figure **3.** The theory of baseband differential detection [5] is discussed in the following sections.

Since no carrier multiplication is performed in this 'C5x implementation, the signals w_t and z_t are directly available at the demodulator without any cosine/sine multiplication. At this time, w_t and z_t are sampled, and the filtering, differential detection, data recovery, and symbol timing operations are performed. In this implementation, the samples w_k and z_k are made directly available to the demodulator in order to test the modem in the loop-back mode.

Filtering

The samples of w_t and z_t are passed through the matched filters in the receiver. Since the baseband I and Q signals at the transmitter are filtered by square-root raised-cosine pulse-shaping filters, the matched filters at the front end of the receiver are designed to give the same frequency response so that the combined receiver/transmitter response becomes raised cosine.

Differential Detection

Differential detection (delay and multiply) is performed with the filtered samples w_k and z_k according to the following equations.

$$w_k = \cos(\phi_k - \mathbf{0}) \quad \text{and} \quad z_k = \sin(\phi_k \pm \mathbf{0}) \tag{6}$$

where ϕ_k is the phase of the carrier at the sampling instant and **0** is an arbitrary phase shift that is canceled in the differential operation.

After the detection operation:

$$x_k = w_k w_{k-1} + z_k z_{k-1} = \cos(\phi_k - \theta_{k-1}) \tag{7}$$

$$y_k = z_k w_{k-1} - w_k z_{k-1} = \sin(\phi_k - \theta_{k-1}) \tag{8}$$

where w_{k-1} and z_{k-1} are the one-symbol. time-delayed values of w_k and z_k, respectively.

Data Recovery

Equations (7) and (8) retrieve the phase change between two successive symbol intervals. Using Table 1 and the values of x_k and y_k, it is simple to decode the dibit information transmitted according to the following hard decision rule.

$$a_k = 0 \quad \text{if } x_k > 0\,; \; a_k = 1 \quad \text{if } x_k < 0 \tag{9}$$

$$b_k = 0 \quad \text{if } y_k > 0\,; \; b_k = 1 \quad \text{if } y_k < 0 \tag{10}$$

Symbol Timing

Symbol timing is one of the most important aspects of the demodulator because the hard-decision decoding has to be performed for data recovery in the appropriate sample so that, in the presence of noise, the recovered information is without error. In a TDMA environment where fast synchronization is required, differential detection is more advantageous, as it does not depend on the carrier recovery and phase lock in the beginning. The theory of symbol timing is based on a simple squaring/energy comparison technique [6]. Assuming four samples per symbol, the energy is calculated at every sample as

$$e = {x_k}^2 + {y_k}^2 \tag{11}$$

In the beginning of timing acquisition, an assumed mid-baud sample (say sample 3) is used for data recovery. Sample 2 and sample 3 energies are designated as e_p and e_n, respectively. At the assumed sample 3, the value of $e_n - e_p$ is calculated. The symbol timing is varied according to the following algorithm.

```
Let thresh = a threshold value ; counter = a count value
   begin:
      If | $e_n - e_p$ |  > thresh then goto ' correct '
      else goto ' done '
   correct: If  $e_n - e_p > 0$ then goto ' checkm '
      else goto ' checkl '
   checkm: countl = 0
        If countm - counter = 0 then goto ' advance '
        else
         {  countm = countm + 1
            goto ' done '  }
   checkl: countm = 0
       If countl - counter = 0 then goto ' retard '
       else
         { countl = countl + 1
           goto ' done '}
   advance: " Process to advance the timing by one sample "
          goto ' done '
   retard: " Process to retard the timing by one sample "
   done:
```

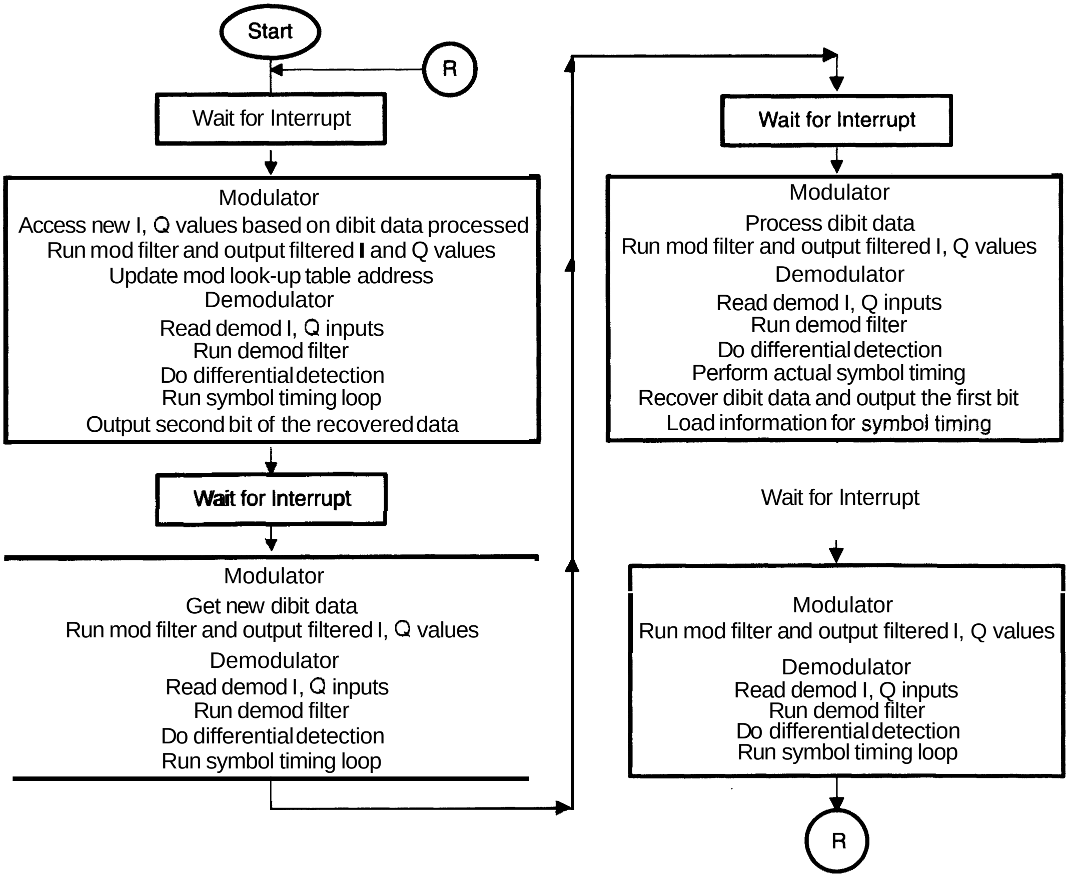
In this algorithm, the values of counter and threshold are initialized in the beginning to estimated values by trial and error. The value of counter can be kept small in the beginning of timing acquisition and later changed to a larger value so that the timing lock is maintained. This method is more stable with phase errors and small frequency shifts, as it does not depend on carrier recovery.

Modem Implementation on the TMS320C5x

Interrupt Organization

The data rate for the modem is 48.6 kbps, per the TIA specifications. The symbol rate for QPSK, then, is 24.3 kbaud/s, as every symbol comprises two bits. The number of sampleshaud chosen is four, both for the modulator and demodulator. This means the baseband filters at the modulator need to generate at least four filtered samples/baud; hence, the minimum sampling frequency that is required is 97.2 kHz. The time available to complete the entire modem operation is quite critical, due to this high sampling frequency. For real-time operation, interrupts are generated at this rate. The consecutive interrupt routines are organized in a particular way for ease of implementation and code efficiency. Figure 4 details the operations performed in the consecutive interrupts.

Figure 4. Interrupt Organization


Start
R
Wait for Interrupt
Modulator
Access new I, Q values based on dibit data processed
Run mod filter and output filtered I and Q values
Update mod look-up table address
Demodulator
Read demod I, Q inputs
Run demod filter
Do differential detection
Run symbol timing loop
Output second bit of the recovered data
Wait for Interrupt
Modulator
Get new dibit data
Run mod filter and output filtered I, Q values
Demodulator
Read demod I, Q inputs
Run demod filter
Do differential detection
Run symbol timing loop
Wait for Interrupt
Modulator
Process dibit data
Run mod filter and output filtered I, Q values
Demodulator
Read demod I, Q inputs
Run demod filter
Do differential detection
Perform actual symbol timing
Recover dibit data and output the first bit
Load information for symbol timing
Wait for Interrupt
Modulator
Run mod filter and output filtered I, Q values
Demodulator
Read demod I, Q inputs
Run demod filter
Do differential detection
Run symbol timing loop
R

Modulator Implementation

Pulse-Shaping Filter

The pulse-shaping filters are designed using a commercial filter design package [9]. A 20-tap pulse-shaping FIR filter with a roll-off factor of 0.35 is designed, and the coefficients are stored in the program memory. The same set of coefficients are used for both I and **Q** filtering. The **I** and Q values (the filter inputs) do not change over a complete symbol period. This means once the modulator look-up table is read in the first interrupt, these values remain unchanged for the next three interrupts. Therefore, the interpolation technique is employed in filtering. An interpolation factor of 4 is achieved. Thus, the number of coefficients used in multiplication is reduced to 2014 = 5. The number of filter delays used is also 5. This interpolation technique saves three-fourths of the time required to run the normal filter. The delays are updated once in four interrupts; specifically, in the interrupt just before the table look-up is done. The five I delays are immediately followed by five Q delays in the internal dual-access random-access memory (DARAM). The MADS instruction is used in the first three interrupts for multiply and accumulate. The MADD instruction is used in the fourth interrupt because the delays are also updated so that the new I and Q values can be loaded. The BMAR register is loaded with the appropriate address before the modulator filter is called in the main routine. Since it is an interpolation filter, the filter coefficients are rearranged in different blocks of five consecutive locations in program memory, so that the appropriate set of coefficients is used by filters in the four consecutive interrupts.

The modulator filter is implemented using one of the circular buffers in the ’C5x. The circular buffer is initialized in the beginning of the program as a decrement-type buffer. The circular buffer **I** and Q delays with the auxiliary register pointer (ARP) are as shown below.

Q_k → Circular buffer end low address
Q_{k-1}
Q_{k-2}
Q_{k-3}
Q_{k-4}
Auxiliary register pointer → ⇒ I_k
I_{k-1}
I_{k-2}
I_{k-3}
I_{k-4} → Circular buffer start high address

The filter code is of the general format

```
rptz       #coefnum
mads/madd        *-
apac
sach
```

The preceding filter code does not involve overhead such as loading scaled filter inputs, loading the filter pointer with the appropriate address, etc. Both I and Q filtering are performed using a single circular buffer with contiguous filter delay locations.

The modulator circular buffer pointer points to location I_k at the start of the first interrupt. As shown in the modulator code, the new value of I_k is accessed from the look-up table and loaded, then decremented in such a way that it points to the Q_k location. The new Q_k value is then loaded and the pointer is modified so that it is reset to the start address. The BMAR register is loaded with the appropriate address so that the filter operates on the appropriate coefficients. The BMAR register allows the dynamic addressing for the filter instructions MADS and MADD. There is no data move involved in the filter for the first three interrupts. In the last interrupt, MADD is used so that a data delay creates the space for the next I and Q values.

Differential Encoding and Signal Mapping

As discussed in the Modulator subsection on page 115, Equations (1) and (2) implement differential encoding and signal mapping as a direct one-step process. Those equations can be further reduced and tabulated as shown in Table 2.

Table 2. Reduced Equations†

A_k	B_k	I_k	Q_k
0	0	sincos x $(I_{k-1} - Q_{k-1})$	sincos x $(I_{k-1} + Q_{k-1})$
0	1	–sincos x $(I_{k-1} + Q_{k-1})$	sincos x $(I_{k-1} - Q_{k-1})$
1	0	sincos x $(I_{k-1} + Q_{k-1})$	–sincos x $(I_{k-1} - Q_{k-1})$
1	1	–sincos x $(I_{k-1} - Q_{k-1})$	–sincos x $(I_{k-1} + Q_{k-1})$

† sincos $= \frac{1}{\sqrt{2}} = 0.707$

The following tables for odd and even symbols are generated from the equations in Table 2 and the naming pattern of the constellation in Figure 1. The values inside the parentheses (within the table entry) are the corresponding (I_k, Q_k) values. The column headings of the table represent A_k, B_k and the constellation point per the constellation of Figure 1.

Table 3. Odd-Symbol Look-Up

A_k	B_k	even0 (0)	even1 (1)	even2 (2)	even3 (3)
0	0	(+0.707, +0.707)	(–0.707, +0.707)	(–0,707, –0.707)	(+0,707, –0.707)
0	1	(+0.707, –0.707)	(+0.707, +0.707)	(–0,707, +0.707)	(–0,707, –0.707)
1	0	(–0.707, +0.707)	(–0,707, –0.707)	(+0,707, –0.707)	(+0,707, +0.707)
1	1	(–0.707, –0.707)	(+0,707, –0.707)	(+0,707, +0.707)	(–0,707, +0.707)

Table 4. Even-Symbol Look-Up

A_k	B_k	even0 (0)	even1 (1)	even2 (2)	even3 (3)
0	0	(0,1)	(−1 ,0)	(0, −1)	(1, 0)
0	1	(1,0)	(0 ,1)	(−1, 0)	(0, −1)
1	0	(−1 ,0)	(0, −1)	(1, 0)	(0, 1)
1	1	(0, −1)	(1, 0)	(0, 1)	(−1, 0)

Modulator Look-Up Table

The constellation points are named 0, 1, 2, and 3, whether for an odd symbol or an even symbol. The odd symbols and even symbols are designated as sym0 and sym1, respectively, and pcn stands for previous constellation. For example, pcn0sym0 means that the previous constellation was numbered 0 and the present symbol is odd. The organization of the table is as follows.

Table 5. Modulator Look-Up

Address Naming	Description	Table Entry
Lookup Table Main Base Address		
SetI's Base Address: pcn0sym0	A_k0B_k0	I_k value
		Q_k value
		Next Address†
	A_k0B_k1	- do -
	A_k1B_k0	- do -
	A_k1B_k1	- do -

† This value is the next set's base address for the new symbol, and it is calculated relative to the look-up table's main base address.

There are eight sets of table entries as shown above (pcn0sym1, pcn1sym0, etc.) with each set having entries for A_k0B_k0, A_k0B_k1, A_k1B_k0, and A_k1B_k1, and each A_kB_k having three entries, totaling 96 entries.

Updating the Look-Up Table

An ARP pointer (for example, ar4) is used to point to the look-up table address. The following code excerpt gets new values for I_k and Q_k from the modulator look-up table and updates the table address pointer.

Modulator Table Manipulation Code

```
Interrupt
1st:       lmmr   bmar,#bmar1        ; bmar reg = base address of 1st set
                                     ; interpolation coeffts in prog mem
           lacc   *+, 13, ar2        ; load ik value
           sach   *-, 0, ar4         ; store in filter's ik input location,
           lacc   *+, 13, ar2        ; load qk value
           sbrk   #coefnum           , sub coef. no (19) to point the qk
                                     , filter input location
           calld  Mod_fltr, *, ar2   ; call delayed filter
           sach   *-                 ; store acc in filter's qk i/p location
           zap                       ; clear acc. & p reg.
           mar    *,ar4              ; after filtering, arp=ar4
           lar    *,ar4,ar5          ; ar4=next set's base address
2nd:       lacl   *+                 ; load 1st bit of dibit data
           sacl   data               ; store in var "data"
           lacl   *+                 ; load 2nd bit of dibit data
           sacl   data1              ; store in var "data1"
           lmmr   bmar, #bmar2       ; bmar=interpolation coeff. address
           calld  Mod-fltr, *, ar2   ; call delayed filter
           zap                       ; clear acc. & p reg.
           nop                         nop to fill up delayed call
3rd:       lacl   scrdata            ; load 1st scrambled bit
           nop                       ; no operation
           xc     1, gt              ; if that bit is a 1 execute foll ins'n
           adrk   #6                 ; add 6 to lookup table pointer
           lmmr   bmar, #bmar3       ; bmar=interpolation coeff. address
           calld  Mod_fltr, *, ar2   ; call delayed filter
           dmov   data1              ; move data1 into data
           zap                       ; clear acc. & p reg.
4th:       lacl   scrdata            ; load 2nd scrambled bit
           nop                       ; no operation
           xc     1, gt              ; if that bit is a 1 execute foll ins'n
           adrk   #3                 ; add 3 to lookup table pointer
           lmmr   bmar, #bmar4       ; bmar=interpolation coeff. address
           calld  Mod-fltr, *, ar2   ; call delayed filter
           zap                       ; clear acc. & p reg.
           nop                       ; nop to fill up delayed call
```

In the first interrupt, after the I and Q values are accessed and loaded into the appropriate filter input locations, the filter is executed. The pointer ar4 now points to the location where the next set's base address is available, and this address value is loaded into ar4. In the second interrupt, the new dibit data is read. In the third interrupt, ar4 is incremented by 6 if the first bit of the dibit data is a 1; otherwise it is unchanged. In the last interrupt, ar4 is incremented by 3 if the second bit of the dibit data is a 1; otherwise it is unchanged. This way, ar4 is modified so that it points to the appropriate subset base address in the set chosen in the first interrupt.

The differential encoding and signal mapping only takes three cycles (max) for the ARP modification in any interrupt. The modulator code is found to be highly efficient with this implementation. This is made possible with the powerful features of the ’C5x. The circular buffer feature enables absolute zero-overhead filtering. Dynamic addressing with MADS and MADD makes interpolation filtering easier. Single-cycle decision-making instructions like XC make look-up table pointer modification simpler. The instructions for delayed call, return and branching, and special instructions like ZAP and RPTZ reduce the various branch overheads.

Demodulator Implementation

The demodulator performs I and Q matched filtering, differential detection, data recovery, and symbol timing. Unlike the modulator, the operations performed by the demodulator in four interrupts are the same except for the symbol timing loop.

Input Filtering

The input I and Q matched filters have square-root raised-cosine frequency response. They are 20-tap FIR pulse-shaping filters similar to the modulator. But these filters cannot be implemented as interpolation filters because the sampled I and Q values are always different. Again, four samples/baud are chosen for the demodulator implementation. The I and Q filters are implemented using the second circular buffer, similar to the modulator I and Q circular buffer. The only difference is that MACD is used instead of MADD or MADS because the inputs are updated with every interrupt.

Differential Detection

Every time the demodulator filter is executed, the filtered I_k sample is made available in the accumulator buffer ACCB, and the filtered Q_k sample is made available in the accumulator. This format is used for code-efficient differential detection. The accumulator buffer feature of the ’C5x is very useful as an accumulator backup, and data transfer between the accumulator and its buffer enhances its usage. Differential detection and energy calculation are performed by the following short code excerpt.

```
sach      zkl, 2           ; immly after i/p filtering,
                           ; store acc. in wk1
lacb                       ; load acc. with acc. buffer
sach      wkl, 2           ; store it in wk1
lt        wkl              ; t reg = wk1
mpy       wkpl             ; P reg = wk1·wk1–1
ltp       zkl              ; t reg = zk1, acc = wk1·wk1–1
mpy       zkp1             ; p reg = zk1·zk1–1
mpya      wkp1             ; P reg = zk1·wk1–1
                           ; acc = wk1·wk1–1 + zk1·zk1–1
sach      xk               ; store acc. in xk
ltp       wkl              ; t reg = wk1, acc = zk1·wk1–1
mpy       zkp1             ; P reg = wk1·zk1–1
sqrs      xk               ; p reg = xk²,
                           ; acc = zk1·wk1–1 – wk1·zk1–1
sach      yk               ; store acc. in yk
lacc      #zero            ; clear acc.
sqra      yk               ; P reg = yk²
apac                       ; acc = xk² + yk²
sach      energy           ; store acc. in energy
lacc      addr             ; load symbol timing address
calad                      ; call delayed with address in acc.
dmov      wkl              ; move wk1 into wk1–1
dmov      zk1              ; move zk1 into zk1–1
```

Notice from Equations (7) and (8) that every new filtered sample w_k and z_k is multiplied by w_{k-1} and z_{k-1}, which are one symbol (that is, four samples) delayed. The segregation of interrupts facilitates efficient implementation. There are four sets of w_k, w_{k-1} and z_k, z_{k-1} used for four interrupts. As far as the first interrupt is concerned, w_{k1} and z_{k1} are the current filtered I and Q values and w_{kp1} and z_{kp1} are the one-symbol delayed values. Similarly, w_{kp2} and z_{kp2} are the one-symbol delayed values for the second interrupt, and so on. Hence, after performing the differential detection and energy calculation, two DMOV instructions move w_{k1} and z_{k1} into w_{kp1} and z_{kp1} to be used next time in that particular interrupt. The w_{kp} and z_{kp} values are allocated proper memory locations to perform this. Once differential detection is done, the symbol timing loop is called using CALAD, accommodating the two DMOV cycles. The main differential detection and energy calculation takes just 17 cycles.

Symbol Timing

Symbol timing is performed using a program address jump with instruction CALA. The organization of the symbol timing loop is as follows. The variable Addr is initialized to Samplel at the beginning of the program.

```
Samplel  :  Output the second bit of the recovered dibit
            information ;
            Addr = Sample2 ;

Sample2  :  energyqrev = energy ;
            Addr = Sample3 ;

Sample3  :  energy-next = energy ;
            Recover dibit data and output first bit of the
            dibit information
            Run symbol timing algorithm
            If no correction : Addr = Sample4
            If advance correction : Addr = Samplel
            If retard correction : Addr = Sampld

Sample4  :  Addr = Samplel

 Sampld  :  Addr = Sample4
```

As seen, if the timing is to be advanced, one sample is skipped. If the timing is to be delayed, one extra dummy sample address jump is inserted.

Performance Results

The performance of the modem implemented on the ’C5x under the AWGN environment is summarized here.

Theory

The theory of noise generation and addition is as follows. Note that VAR( ) and Std ( ) represent the variance and the standard deviation functions.

$$SNR_{dB} = 10\log_{10}\left(\frac{Signal\ Power}{Noise\ Power}\right) = 10\log_{10}\left(\frac{Var(signal)}{Var(noise)}\right) \tag{12}$$

$$\therefore\ Var(noise) = \frac{Var(signal)}{10^{(SNR_{dB}/10)}} \tag{13}$$

For I & Q arms:

$$Var\_i = \frac{Var(I\ \text{signal})}{10^{(SNR_{dB}/10)}} - \frac{[Std(I\ signal)]^2}{10^{(SNR_{dB}/10)}} \tag{14}$$

$$Var\_q = \frac{Var(Q\ \text{signal})}{10^{(SNR_{dB}/10)}} = \frac{[Std(Q\ signal)]^2}{10^{(SNR_{dB}/10)}} \tag{15}$$

Also

$$Std\_i = \sqrt{Var\_i}\ ;\quad Std\_q = \sqrt{Var\_q} \tag{16}$$

Two independent Gaussian-distributed random-noise sequences, **I_noise[k]** and **Q_noise[k]**, are generated using the **Matlab** software. The noise is added to the I and Q signals as shown below.

$$I\_noise[k] = I\_noise[k] \times Std\_i \tag{17}$$

$$Q\_noise[k] = Q\_noise[k] \times Std\_q \tag{18}$$

$$Id[k] = I[k] + I\_noise[k] \tag{19}$$

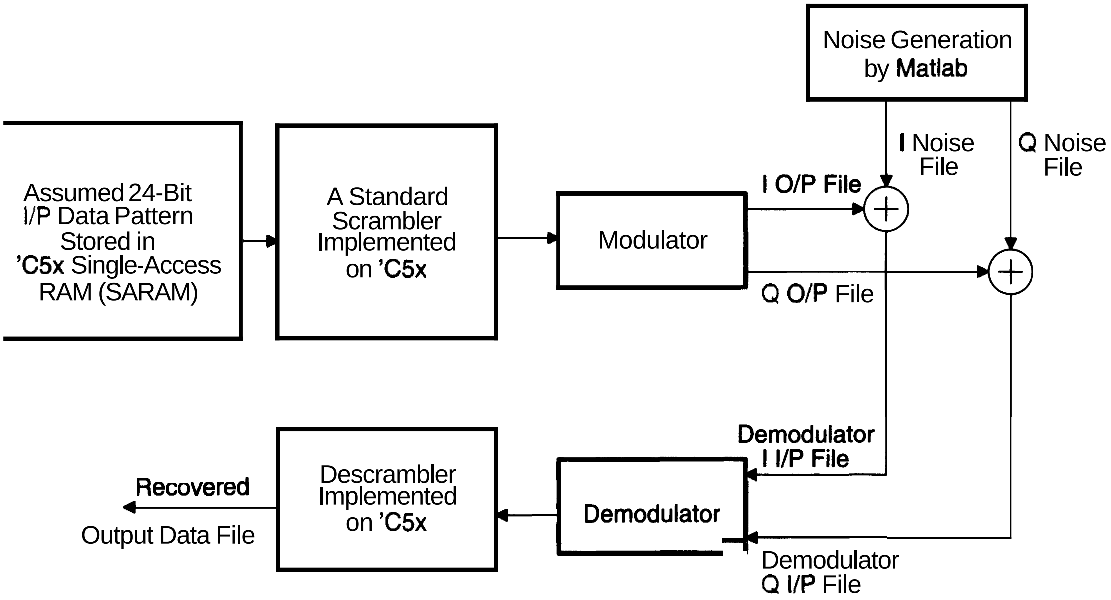
$$Qd[k] = Q[k] + Q\_noise[k] \tag{20}$$

I_d and Q_d are the two new demodulator input points that are generated using **Matlab**.

Testing

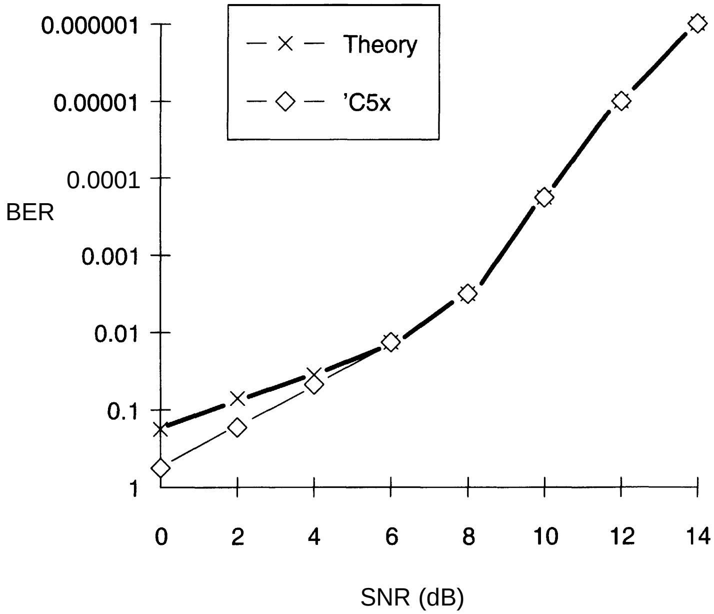
The modem implementation is tested using a file I/O scheme in which the modem runs on a **'C5x** EVM card and communicates with the host **PC** for a nonreal-time file transfer. The testing is performed with the setup shown in Figure **5.**

Figure 5. Modem Test Configuration


Noise Generation by Matlab
Assumed 24-Bit I/P Data Pattern Stored in 'C5x Single-Access RAM (SARAM)
A Standard Scrambler Implemented on 'C5x
Modulator
I Noise File
Q Noise File
I O/P File
Q O/P File
Demodulator I/P File
Recovered Output Data File
Descrambler Implemented on 'C5x
Demodulator
Demodulator Q I/P File

As shown in Figure 5, the assumed input data pattern is scrambled in the **'C5x** to generate randomness in the input data. About 60,000 samples each of I and Q are generated by the modulator and stored in files. The **'C5x** EVM talks to the **PC** through **DSP-PC** interface software for file transfer. The I and Q noise files are generated by **Matlab** and added with modulator output files. The demodulation and descrambling is done, and recovered data of 30,000 bits is stored in a file. The number of errors in the demodulator output file are counted in trials with various **SNR** values. The modem performance for the AWGN channel is shown in Figure 6.

Figure 6. BER Versus SNR for a Static AWGN Channel



Performance

As seen in Figure 6, the performance in an AWGN environment closely follows the theoretical performance. Since this is a cellular modem, its performance also needs to be tested under fading conditions with Doppler shifts due to vehicle speed. The performance of this modem under such conditions is expected to be of moderate standards because the implementation involves restrictions such as fewer samples/baud, etc. The performance could be improved by employing more samples/baud, a sophisticated symbol timing scheme, an automatic AGC, and an equalizer at the front end of the demodulator. As the number of cycles taken by the entire modem function is about 160, other extra features listed above could be accommodated to improve the performance under fading and Doppler-shift conditions.

Speed and Memory Requirements

Table 6 lists the number of ’C5x program and data memory words required by the core modulator/demodulator algorithm. It also provides the maximum number of cycles needed to run the modulator and demodulator. The hardware, I/O interface, and program initialization requirements are not included here, as they do not fall within the time-critical loop of the modem implementation. Note that this table does not include the interrupt handling overheads.

Table 6. Program Memory and Speed Requirements

Module Name	Program Memory	Data Memory	Cycles (Max)
Modulator	76 + 116†	114	32
Demodulator	246 + 20t	68	126

† This is the size of program memory used for loading tables, etc.

The maximum number of words and cycles used by the various modules of the modulator and demodulator, including the different overheads, are shown in the following tables.

Table 7. Modulator Code Size and Execution Time

Module Name	Size in Words	Cycles (Max)
Mod–Main	55 + 96†	13
Mod–Fltr	21 + 20†	19

† This is the size of program memory used for loading tables, etc.

Table 8. Demodulator Code Size and Execution Time

Module Name	Size in Words	Cycles (Max)
Dmd_Fltr	11 + 20†	52
Dmd_Main	124	28
Sym_Time	111	46

† This is the size of program memory used for loading tables, etc.

As the above tables show, both the modulator and demodulator have been well optimized to accommodate future addition of modules, if necessary, for performance improvements. There is also a large portion of unused internal RAM for future memory requirements.

Summary

The IS-54 U.S. digital cellular modem concepts are introduced and the theory of π/4-QPSK with signal constellation is discussed. The modem implementation on the TMS320C5x is explained and the performance of the modem with AWGN is summarized. Also, the requirements of the modem regarding speed and memory are tabulated. The efficiency and capabilities of the TMS320C5x for the high-bit-rate cellular modem application are clearly visible from the modem implementation. This implementation needs to be further studied under Rayleigh fading with co-channel interference and Doppler shift. Improvements for the demodulator are suggested. The modem program is made highly modular and is developed according to the TI Communication Software Library (CSP) developer's guidelines[1]. This π/4-QPSK cellular modem implementation on the TMS320C5x family of DSPs provides guidelines for cellular-systems designers to employ in using the 'C5x DSP for all cellular and related applications.

Code Availability

The associated program files are available from Texas Instruments TMS320 Bulletin Board System (BBS) at (713) 274-2323. Internet users can access the BBS via anonymous ftp at *ti.com.*

---

[1] Refer to "Software Coding Guidelines for 'C5x Developers", p. 247 of this book.

References

1. Lee, W. C.Y., *Mobile Cellular Telecommunications Systems*, McGraw-Hill, 1989.

2. Proakis, J.G., *Digital Communications*, McGraw-Hill, 1989.

3. Crochiere, R.E., and Rabiner, L.R., *Multirate Digital Signal Processing*, Prentice-Hall, Inc., 1983.

4. Chennakeshu, Sandeep, and Saulnier, Gray J., "Differential Detection of π/4-Shifted-DQPSK for Digital Cellular Radio", IEEE *Transactions on Vehicular Technology*, Vol. 42., No. 1, February 1993.

5. Feher, Kamilo, "MODEMS for Emerging Digital Cellular-Mobile Radio System", IEEE *Transactions on Vehicular Technology*, Vol. *40*, No. 2, May 1991.

6. Troullinos, George, et al., "Theory and Implementation of a Splitband Modem Using the TMS32010", *Digital Signal Processing Applications with the TMS320 Family*, Vol. 2, Prentice-Hall, Inc, 1991.

7. *Cellular System: Dual-Mode Mobile Station – Base Station Compatibility Standard*, IS-54 Project Number 2215, Electronic Industries Association, December 1989.

8. *TMS320C5x User's Guide*, Texas Instruments, 1990.

9. *DFDP3/Plus Digital Filter Design Package Release 1.00*, Atlanta Signal Processors, Inc., 1992.

10. *MATLAB User's Guide*, The Math Works, Inc., 1989.

11. Chishtie, Mansoor A., "Software Coding Guidelines for 'C5x Developers", *Telecommunications Applications With the TMS320C5x DSPs Application Book*, Texas Instruments, 1994, pp. 247–258.

A DSP GMSK Modem for Mobitex and Other Wireless Infrastructures

Etienne J. Resweber
Synetcom Digital Incorporated

Abstract

Mobitex is a packetized wireless 900-MHz wide area network (WAN) that allows mobile/portable subscribers to transfer data, including e-mail, through the growing national and international network infrastructure. The network operates with an 8-kbps data rate using GMSK.3 modulation. User terminals are typically sophisticated portable or mobile devices that encompass one or more applications and all additional OSI protocol layers necessary to send and receive data on the network. Within the user terminal, the interface between the radio (physical layer) and other layers is a high-performance Gaussian minimum shift-keying (GMSK) modem. During transmission, the modem converts packets of network data into transmit baseband. For receiving, it demodulates similar waveforms into data decisions. The typical Mobitex modem produces at least part of the physical-layer processing necessary for radio interface.

The cellular industry solution for packetized data is called cellular digital packet data (CDPD). The modem waveforms used for Mobitex are similar (GMSK), though CDPD uses 19.2 kbps. Core GMSK concepts, however, still apply; therefore, the modem design described herein can also be used as a basis for CDPD modem development in the future.

Synetcom Digital Incorporated has developed a DSP-based Mobitex modem that accomplishes the radio interface. Transmit data in packet form is level shifted and Gaussian filtered digitally within the modem algorithm so that it is ready for transmitter baseband interface, either via D/A converter or by direct digital modulation. Receive data at either baseband or intermediate frequency (IF) from the radio receiver is digitized and processed by the modem — nearly optimally — into data decisions. Packet synchronization is also handled by the modem, assuring that the next layer sees only valid Mobitex packets. Received signal degradation from frequency offsets, multipath (Rayleigh) fading, and other effects is anticipated and addressed in the modem design.

Introduction

About Mobitex

Mobitex is a packetized narrow-band data service operating near 900 MHz (450 MHz in the United Kingdom), originally conceived by Swedish Telecom and further developed by Eritel, a joint venture of Swedish Telecom and Ericsson. The service is being offered in the United States by RAM Mobile Data/Bell South. Base stations, which typically cover 5–15 mile radii, are arranged in a cellular-like fashion. Network roll-out has proceeded to the extent that coverage within the top 200 U.S. metropolitan areas is advertised. At Synetcom Digital Incorporated's Redondo Beach, California office, five base stations are audible on an indoor cellular whip, four of which have usable signals.

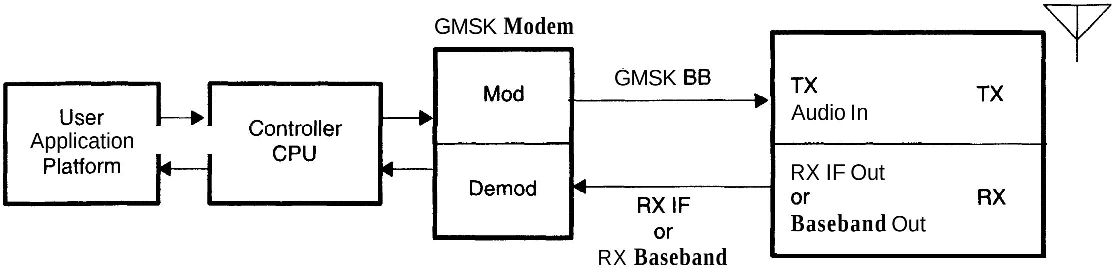
Other Networks

Mobitex falls into the class of wireless WANs. There is at least one other operational infrastructure, called Ardis (IBM/Motorola), and several more are anticipated, including CDPD from McCaw Cellular and its partners.

Mobitex Terminal Hardware Architecture

Figure 1 shows a typical terminal architecture. Controller CPU functions typically handle higher OSI layers, which form packets, provide error coding and scrambling, handle acknowledgments, and control transmitter and receiver operation.

Figure 1. Typical Mobitex Terminal Architecture



WAN Modems and the Radio Channel

WAN modems are designed to operate with signal distortions produced by multipath frequency offsets and nonideal radio IF filters. Multipath distortion occurs when a signal reflection causes propagation along several paths across the link. Different path lengths and reflections produce signal components with unequal amplitude and delay, which vector sum at the receiver. For fixed links, the vector sum looks like a superposition of comb filters in the frequency domain. In the time domain with long delays, symbol energy is *smeared;* this smearing is known as intersymbol interference (ISI). A null (cancellation) or significant slope at or near the carrier frequency causes severe distortion to the received signal, which can degrade bit error rate (BER) performance.

The actual multipath parameters vary spatially for mobile links. The receiver sees time-varying comb functions with nulls that traverse the spectrum and momentarily align with the signal frequency, causing deep fades. Under these conditions, the received carrier-envelope amplitude has been shown theoretically and experimentally to conform to a Rayleigh distribution. Based on this model, it has been shown that 99.9% of fluctuation occurs within a dynamic range of 40 dB [1].

Typical radio systems allow for some frequency error (tight frequency tolerance is expensive), which may degrade modem receive performance. Receiver IF and baseband filtering is also never ideal and can introduce additional waveform distortion from ISI.

The Mobitex modem design described herein anticipates these and other distortions and has been shown to operate satisfactorily in laboratory simulations of the degradations. Mobile field tests are anticipated to further qualify modem performance.

Advantages of DSP Modems

Modem DSP code is written to closely approximate the ideal modem architecture — typically, more closely than an analog implementation approximates it — potentially realizing outstanding modem performance that is repeatable over time and temperature. The approach is flexible because all modem parameters can be trimmed in software.

A DSP can assume other chores in the user terminal and may become the platform for additional protocol layers required for a given network, assuming enough spare MIPS are available, and it may even be reconfigured to interface with other networks on multiple layers.

DSP chips are on the same fast track as CPUs, with smaller feature size, higher speed, lower power, and lower voltage required with each new generation. Competition among several major corporations has brought pricing down to levels that compete favorably with discrete analog and ASIC implementations.

Mobitex DSP Modem Characteristics

Code Size and DSP MIPS Requirement

The Mobitex modem code is actually two distinct algorithms associated with half-duplex transmit and receive functions. The receive (digital demodulator) algorithm is more complex and embodies most of the important features necessary for a successful modem design. As with all modems, receiver code requires more processor power, as shown in Table 1.

Table 1. Receiver-Code Processor Power Requirements

Function	Code Size	TMS320C25 MIPS Requirement
Transmit GMSK Modulator	256 words	3
Transmit PN Generator	128 words	1
Receiver Digital Demodulator	500 words	6
Receiver Discriminator†	128 words	4

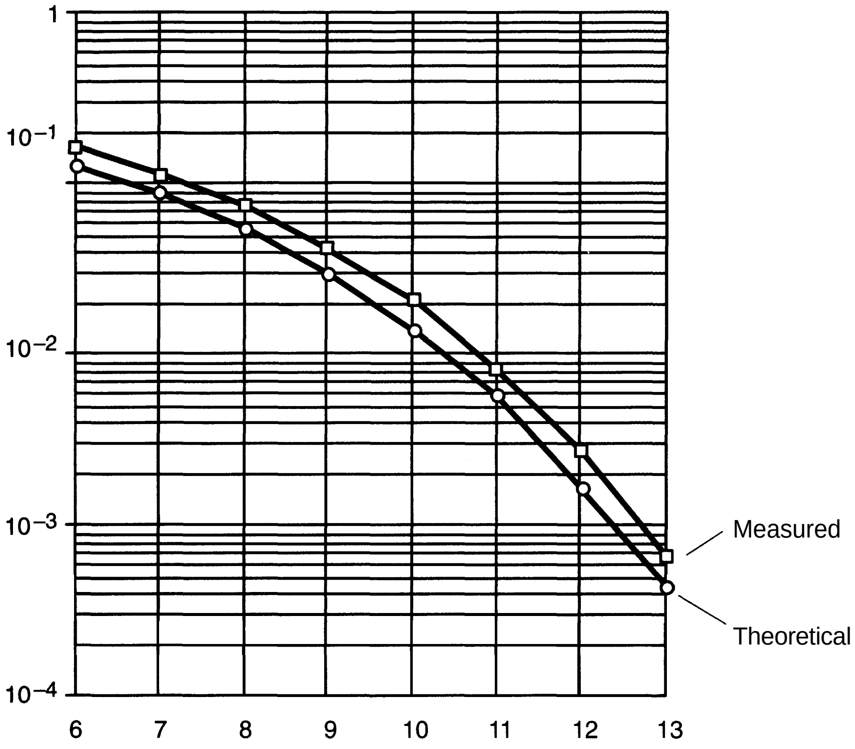
† Discriminator code is required if the A/D interface is receiver IF.

Bit-Error-Rate Performance

The BER performance of a pair of the Mobitex modems was measured in the laboratory. GMSK IF and Gaussian noise are summed to create an approximation of the noisy radio channel, representative of weak receive signals. Signal and noise power levels are calibrated relative to each other and converted to E_b and N_0 values through bit rate and equivalent bandwidth normalization. The test scenario increments noise in 1-dB steps and captures BER data.

Results are plotted against theoretical performance in Figure 2. Performance is quite close to ideal (<0.5 dB) over the range of data shown. Transmit GMSK is a continuous 2^9-1 pseudorandom noise (PN) code.

Figure 2. Bit Error Rate Versus E_b/N_o Modem Performance



Modulator Design

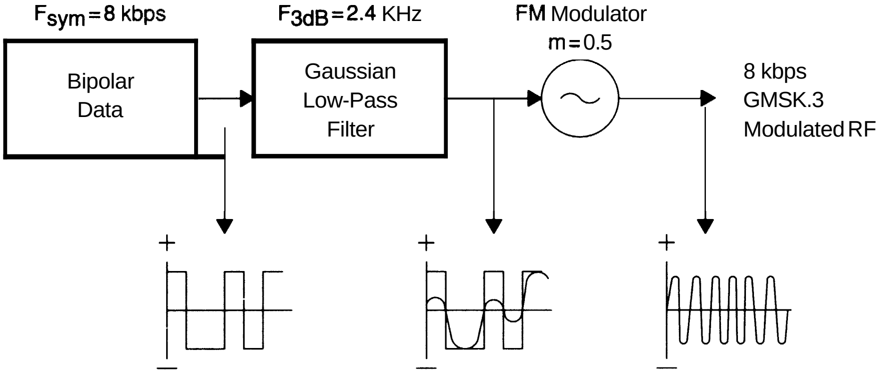
GMSK.3 Modulation

GMSK has been widely proposed and utilized for mobile radio data communications. In addition to Mobitex, GMSK is used for GSM (European digital cellular) and CDPD in the U.S. Several characteristics that make it especially attractive for these applications are:

- Spectral efficiency (12.5-kHz channels for 8-kbps GMSK.3)
- Constant RF envelope (efficient class-C amplifiers and hard-limiting receivers)
- Compatibility with analog FM techniques
- Reasonable performance (assuming proper modem techniques) in multipath environment

As illustrated in Figure 3, GMSK.3 is generated with Gaussian low-pass filtered bipolar data, applied to a DC coupled FM modulator, set to a modulation index of 0.5.

Figure 3. Idealized GMSK.3 Generation



The .3 suffix on GMSK refers to the BT, or bandwidth, symbol time product. Alternatively, BT can be expressed as the ratio:

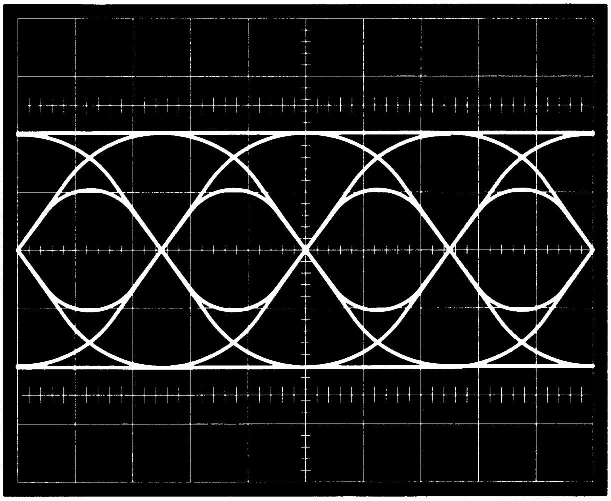
$$F_{tx} / F_s = 0.3 \text{ for GMSK.3}$$

where F_{tx} is the transmit filter with a 3-dB bandwidth and 2.4-kHz frequency, and F_s is the symbol rate.

As the ratio increases, more energy at higher frequencies is transmitted, occupying more radio spectrum. A decrease in ratio below 0.2 attenuates higher frequencies significantly, compromising obtainable performance.

The eye pattern for GMSK.3 baseband signals is shown in Figure 4. An eye pattern conveys every possible trajectory in the transmit/receive data baseband waveform synchronized to symbol timing. It is useful because it can very quickly convey the *fidelity* of transmit and receive data and is a strong diagnostic tool in the wireless development environment.

Figure 4. Eye Pattern for 8-kbps GMSK.3, 2^{15}–1 Length Pseudorandom Transmit Data†

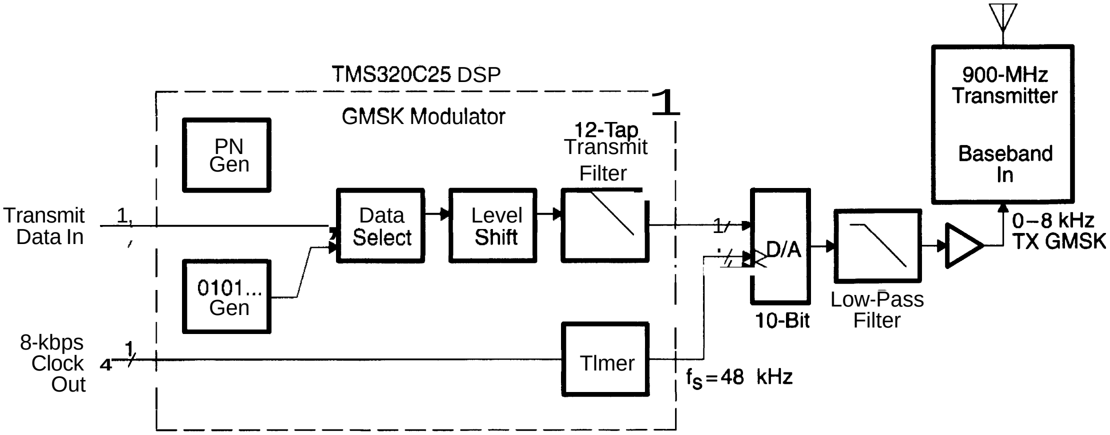


† Signal observed at the output of the transmit filter

GMSK Modulator Architecture

A block diagram of the modulator DSP implementation is shown in Figure 5.

Figure 5. GMSK Modulator DSP Implementation


TMS320C25 DSP
GMSK Modulator
PN Gen
Transmit Data In
1
0101... Gen
Data Select
Level Shift
12-Tap Transmit Filter
8-kbps Clock Out
1
Timer
$f_S = 48$ kHz
1
D/A
10-Bit
Low-Pass Filter
900-MHz Transmitter
Baseband In
0–8 kHz TX GMSK

The present GMSK modulator algorithm accepts data from upper OSI layers that has been packetized, error encoded, and scrambled according to Mobitex specifications. In most systems, this is accomplished on a CPU in the application computer or in a separate microcontroller. Ultimately, these functions can occur on the DSP.

The modulator algorithm either accepts external data or can generate pseudorandom (PN) data with 2^7–1, 2^9–1, and 2^{15}–1 length codes for transmit test purposes. This feature enables easier bit-error-rate measurements, eye-pattern checks, and other system measurements during integration with radio gear.

The DSP algorithm implements a level shift and digital low-pass filter function on the square data provided by the other OSI layers or the algorithmic PN generator. A 12-tap (two symbol length) linear-phase FIR structure forms the transmit filter, which is designed to approximate the ideal Gaussian transmit filter very closely. The FIR 3-dB point is set to 2.4 kHz for BT = 0.3. The modulator sample rate is 48 kHz, producing a baseband bandwidth with significant energy out to approximately 5 kHz and virtually no energy beyond 10 kHz.

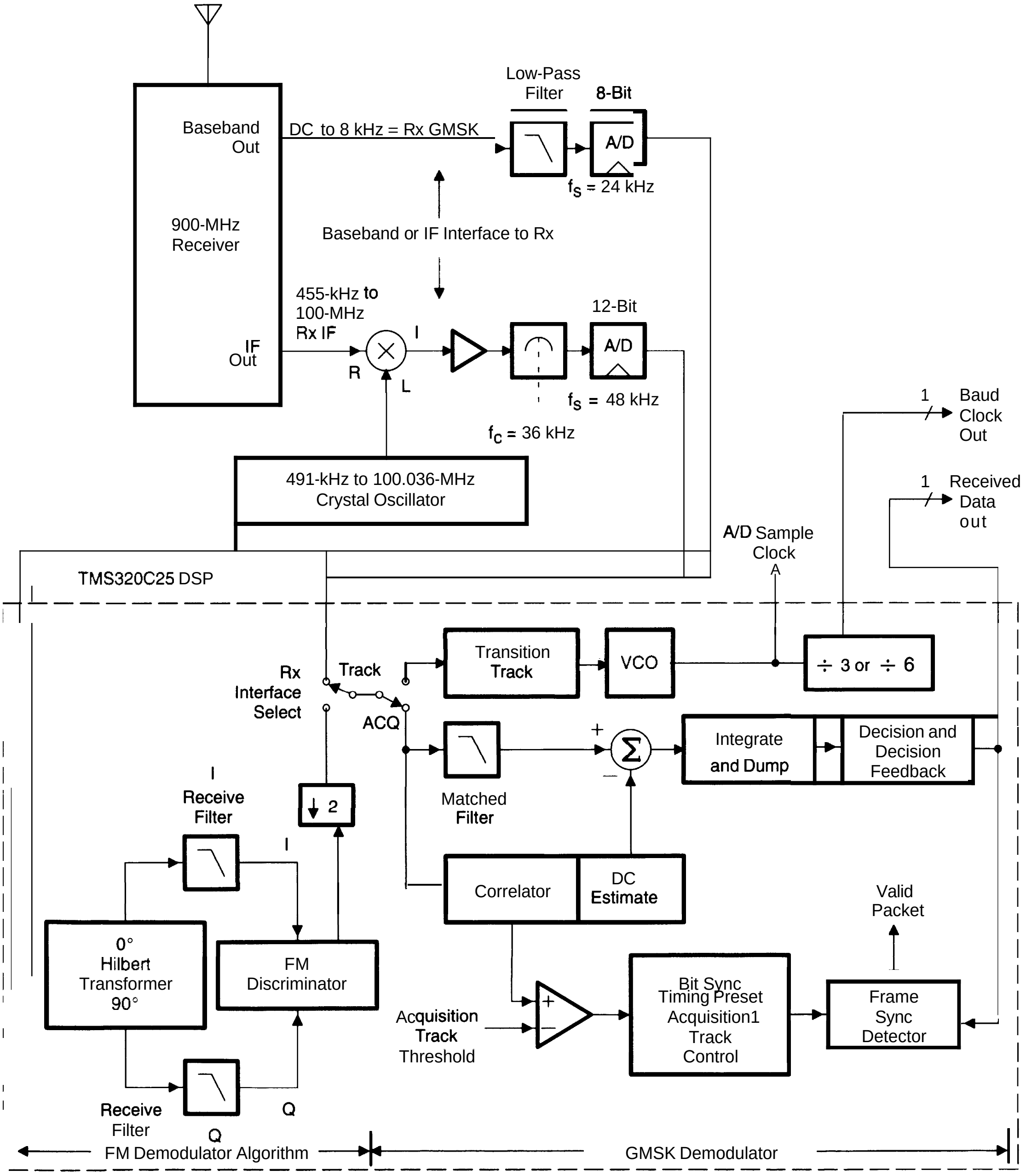
The modem exists on an evaluation board that contains a 16-bit D/A converter and low-pass reconstruction filter that attenuates digital spectra beyond $f_S/2$ (24 kHz) to levels near the noise floor. Other implementations can exploit the latest single-chip CODEC or analog interface circuits, which combine several D/A and reconstruction filter blocks with A/D converters. A single chip can thus furnish the entire radio-analog interface. Ten-bit precision D/A converters are adequate for this application.

GMSK Demodulator Design

GMSK Demodulator Architecture

A block diagram of the demodulator structure is shown in Figure 6. The upper half of the figure shows an external interface to a 900-MHz radio receiver. Either a baseband or an IF interface is possible with this algorithm. The IF interface includes an FM discriminator function in the DSP code.

Figure 6. GMSK Demodulator DSP Implementation



Baseband Out
900-MHz Receiver
IF Out
DC to 8 kHz = Rx GMSK
Low-Pass Filter
8-Bit
A/D
f_S = 24 kHz
Baseband or IF Interface to Rx
455-kHz to 100-MHz Rx IF
R
L
I
12-Bit
A/D
f_S = 48 kHz
f_C = 36 kHz
491-kHz to 100.036-MHz Crystal Oscillator
1
Baud Clock Out
1
Received Data out
A/D Sample Clock
A
TMS320C25 DSP
Rx Interface Select
Track
ACQ
Transition Track
VCO
÷ 3 or ÷ 6
+
–
Σ
Integrate and Dump
Decision and Decision Feedback
I
Receive Filter
↓ 2
Matched Filter
I
0° Hilbert Transformer 90°
FM Discriminator
Correlator
DC Estimate
Valid Packet
Acquisition Track Threshold
+
–
Bit Sync Timing Preset Acquisition1 Track Control
Frame Sync Detector
Receive Filter
Q
Q
FM Demodulator Algorithm
GMSK Demodulator

The demodulator algorithm employs noncoherent techniques to arrive at each data decision. Two entry points for digitized data from the receiver are shown in Figure 6.

Digitized IF Processing

As the cost and power consumption of DSP MIPS and associated A/D converters decrease, it will make sense to locate the A/D converter closer to the antenna, somewhere in the radio IF strip. Traditionally, digital processing at IF has been applied to expensive military systems in which the highest possible receiver performance is required. As DSP costs decrease and techniques improve, IF processing may become standard in wireless applications, where both benefits — cost and performance — are possible. In anticipation of this next step, a radio IF interface to the DSP demodulator algorithm was created.

Band-limited radio IF (presumed to be at 36 kHz center, 12.5 kHz wide for Mobitex) is digitized at a sample rate of 48 kHz, realizing a digital down-conversion to a center frequency of 12 kHz. The DSP algorithm then implements a close approximation of a $0°/90°$ splitter that feeds a pair of identical, 7-tap low-pass FIR receive filters, carefully bandwidth optimized under noise conditions for best overall demodulator performance.

Digital FM Discriminator

The FM discrimination algorithm maps the frequency of complex IQ samples to a voltage using a differential estimation technique. Sample-rate decimation by a factor of 2 is also used, yielding subsequent processing that executes only on every other input IF sample. After decimation, the discriminator normalizes each sample by $I^2 + Q^2$ to wipe off any IF energy variation, due to radio channel fades that fall out of the receiver's hard limiting or AGC range. The dynamic range of the normalization algorithm approaches 40 dB when used with a 12-bit A/D converter.

Normalization becomes a significant issue if the receiver RF/IF chain must have linear or AGC loop-controlled gain. Certain modulation types require linear receiver performance. In a multinetwork/infrastructure environment, linearity may be a requirement. The normalization algorithm exists to cover that eventuality, even though most implementations to date have used hard limiting and traditional FM receiver techniques.

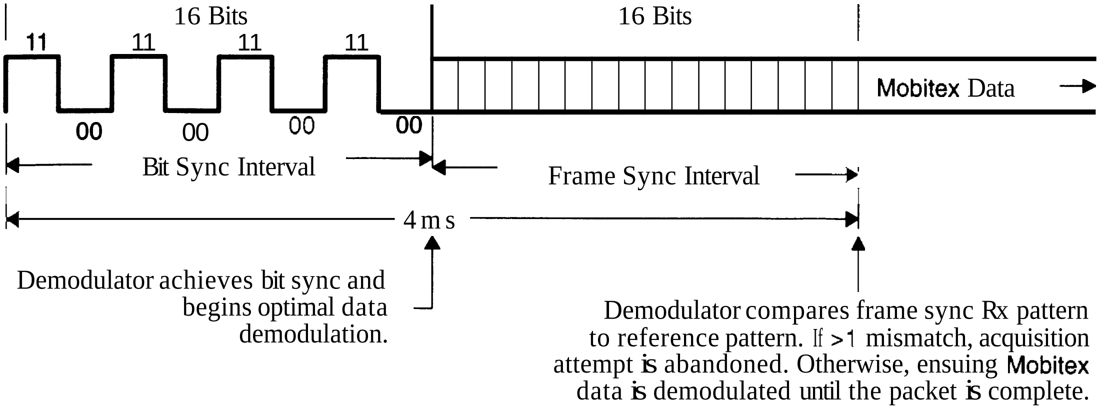
Baseband Processing

A second entry point to the demodulator algorithm can be selected just after the digital FM discriminator of Figure 6. The receiver baseband (audio DC to 8 kHz) that carries the data waveform is digitized by at least an 8-bit A/D converter at a sample rate of 24 kHz. Less precision is required because the receiver hard limiting and discriminator mitigate most of the envelope fluctuation due to flat signal fading. Processing beyond this point is identical regardless of which input is selected.

Packet Acquisition

All received Mobitex packets are qualified by an acquisition process that recognizes and exploits information in the first two data structures of the Mobitex packet, which is shown in Figure 7.

Figure 7. Mobitex Packet Structure



Demodulator achieves bit sync and begins optimal data demodulation.

Demodulator compares frame sync Rx pattern to reference pattern. If >1 mismatch, acquisition attempt is abandoned. Otherwise, ensuing Mobitex data is demodulated until the packet is complete.

When the demodulator is not tracking and demodulating a qualified packet, an FIR filter-based structure that implements pattern specific correlation is executed. The correlator searches for the bit sync pattern. When correlator output exceeds a preset threshold, demodulation begins and frame sync, which is a fixed, country-specific pattern 16 bits long, is expected. If frame sync does not occur within the next 16 bits with one bit error or less, the packet acquisition attempt is abandoned and the correlation process is begun again. In this manner, probability of false acquisition is kept very small, and higher OSI layers in the user terminal receive data only when qualified packets are present.

Simultaneous to successful correlation, a low-bandwidth tracking-loop algorithm is invoked. Data transitions (zero crossings) are extracted, and the algorithm attempts to keep crossings aligned by adjusting the DSP timer register, which ultimately generates sample pulses to the A/D converter. The resulting servo loop is invoked as long as the qualified packet data is present. This feature is especially important for long packets and operates reliably even with very weak receive signals.

Also, after each successful correlation, a DC estimate (which is proportional to receiver frequency offset relative to base station) is extracted from the bit sync sequence and is used to cancel DC offsets in the baseband demodulation (track) path. The modem performance is made tolerant of frequency offsets in this manner.

Finally, the correlator triggers an A/D sample timing preset. Correlator output information is examined, and a precise estimate of correct initial A/D sample phase and frequency is made. The preset timing is subsequently updated very slowly at each zero crossing with the aforementioned servo loop.

Data Demodulation

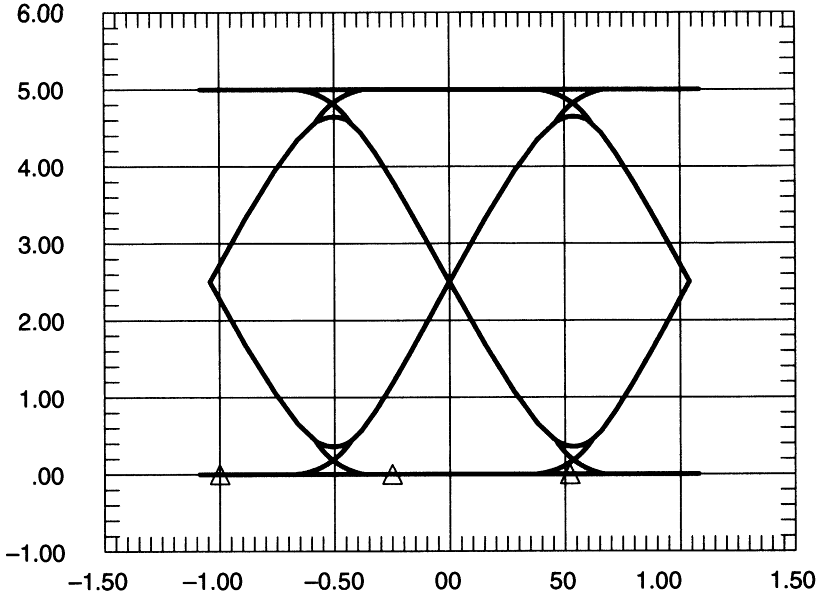
After correlation to the packet bit sync pattern occurs, the data demodulation/decision process begins. Conceptually, the goal of the decision process is simple: every three samples (at 24 kHz) produce either a zero or one data decision such that the original packet data, prior to modulation, is recovered.

The decision process employs matched filtering (which is identical to transmit filtering), integrate-and-dump, and decision feedback techniques to minimize the probability of bit errors. The integrate-and-dump and decision feedback algorithms are especially effective under disturbed conditions, such as with either fixed or time-varying multipaths, and they also reduce modem sensitivity to ISI induced by receiver filters.

Design Adaptations for CDPD

The CDPD modem requirement is for GMSK.5 radio waveforms at 19.2 kbps. CDPD utilizes cellular channels that are full-duplex; the packetized protocol can use this characteristic, though a half-duplex CDPD implementation is also possible. A computer simulation of the transmit eye pattern for GMSK.5 is shown in Figure 8.

Figure 8. Computer-Simulated Eye Pattern for 19.2 kbps GMSK.5 (Amplitude Versus Time)



As compared to Mobitex, the higher baud of CDPD dictates use of a more powerful DSP chip, such as one from TI's TMS320C5x family, to support the modem function. Generally speaking, a good estimate for half-duplex CDPD MIPS required for the GMSK demodulator can be obtained by simply scaling the 6-MIPS benchmark for the baseband-interfaced Mobitex demodulator. A conservative approximation is based on the ratio of bauds (19.2 / 8 = 2.4). CDPD, therefore, can require up to 14.4 MIPS peak for the receive modem function.

Digital demodulators can operate with fewer samples per baud than were assumed above. The Mobitex modem uses an A/D converter to sample IF at 48 kHz or baseband at 24 kHz. The algorithm ultimately uses three samples per 8-kHz symbol in the data-decision section.

For CDPD, it is estimated that if two samples per baud are used, approximately 0.7 dB of performance is sacrificed. The associated baseband sample rate is 38.4 kHz, and the corresponding MIPS requirement is approximately 10 (33% less than the 3 samples-per-baud case).

CDPD's GMSK.5 uses a higher BT factor (0.5). The immediate result is an eye pattern that is less filtered than shown in Figure 4. Overall modem receive performance is correspondingly improved. Adjustments of constants in the current decision feedback algorithm are necessary to optimize performance, though the current constants (based on GMSK.3) will operate surprisingly well.

CDPD transmit baseband eye pattern has been simulated and is shown in Figure 8. The Gaussian transmit filter 3-dB frequency is 9.6 kHz. The transmit and receive Gaussian digital filter is adjusted for the new bandwidth.

Transition of GMSK Modem to TMS320C5x

Work has begun to translate the existing ’C2x code to a ’C5x processor. The GMSK modulator and portions of the demodulator algorithm are currently able to execute successfully on TI’s EVM system. The translation is very straightforward, using TI’s DSP assembly conversion utility (DSPCV.EXE), and the utility is able to convert ’C2x source code (.ASM) files directly to ’C5x source code files. A minor amount of manual intervention is necessary after running the utility. This intervention is associated with memory directives that do not have exact equivalents between the two processor families.

Conclusions

Packet networks such as Mobitex or CDPD generally operate with a sophisticated protocol that allows for error detection, limited error correction, and, if all else fails, packet retransmission. All data is eventually received successfully across the link. High-performance modem techniques are employed to meet overall network performance requirements because inferior modems can generate unnecessary traffic, requiring repetition of missed data.

The Mobitex modem code exists on a 16-bit fixed-point TMS320C25, which is an entirely adequate platform for the core modulation/demodulation algorithms implemented. No issues associated with the 16-bit fixed-point precision were encountered. In general, no applications are envisioned in which floating-point processors or wider fixed-point registers are necessary for wireless modems anticipated for future implementation.

The existing code is portable to the Texas Instruments TMS320C5x family, which will ultimately offer 3.3-volt, 40-MIPS operation, suitable for battery-powered portable operation. The fully implemented IF interface Mobitex modem algorithm requires 10 MIPS for demodulation. The ’C5x family and similar processors from other manufacturers open prospects for other layers of wireless protocol executing on the same DSP, with ultimate partitioning of DSP and controller-processing responsibilities dictated by DSP/processor cost, memory requirements, speed and power consumption, and interface issues. All new designs should weigh these issues carefully.

The DSP chip offers flexibility beyond Mobitex. Multiple wireless infrastructures, including CDPD, can ultimately be accommodated on the same processor, which, in fact, may be necessary for long-term product survival. As wireless/PCN industries take shape, the emphasis will likely be on flexibility. Systems that are incompatible starting at the lowest link/physical layers will dictate that user radio/modem devices be capable of loading and executing new modem and control (protocol) code as needed. A single user terminal can thus interface with multiple infrastructures.

Code Availability

The associated software is available for licensing from Synetcom Digital Incorporated, 1426 Aviation Boulevard, Suite #203, Redondo Beach, California 90278.

References

1. Feher, Kamilo, *Advanced Digital Communications Systems and Signal Processing Techniques*, Prentice-Hall, 1987.

2. Hirono, Masahiko, Miki, T., and Murota, K., "Multilevel Decision Method for Band-Limited Digital FM with Limiter-Discriminator Detection", IEEE *Transactions on Vehicular Technology,* August 1984, pp. 114–122.

3. *Mobitex Interface Specification*, Revision 2A, RAM Mobile Data, Woodbridge, New Jersey, February 1993.

4. *Cellular Digital Packet Data System Specification,* Release 1.0, Book III, Volume 4, July 19, 1993.

Part VI
Equalization and Channel Estimation

Equalization Concepts:
A Tutorial

David Smalley
Atlanta Regional Technology Center
Texas Instruments Incorporated

Introduction

DSP-based equalizer systems have become ubiquitous in many diverse applications including voice, data, and video communications via various transmission media. Typical applications range from acoustic echo cancelers for full-duplex speakerphones to video deghosting systems for terrestrial television broadcasts to signal conditioners for wireline modems and wireless telephony. The effect of an equalization system is to compensate for transmission-channel impairments such as frequency-dependent phase and amplitude distortion. Besides correcting for channel frequency-response anomalies, the equalizer can cancel the effects of multipath signal components, which can manifest themselves in the form of voice echoes, video ghosts or Rayleigh fading conditions in mobile communications channels. Equalizers specifically designed for multipath correction are often termed ***echo-cancelers*** or ***deghosters.*** They may require significantly longer filter spans than simple spectral equalizers, but the principles of operation are essentially the same.

The literature is rich with practical and theoretical treatments of the various equalization schemes. This article attempts to familiarize you with some basic concepts associated with channel equalization and data communication in general. It is hoped that the liberal use of signal plots will lead to an intuitive understanding of such concepts as intersymbol interference and multipath effects. To this end, the Mathcad 4.0 [15] files used to create the figures have been made available. See the ***Code Availability*** section on page 174. You are encouraged to experiment further with these files. For a more rigorous mathematical treatment, refer to the numerous books and articles cited on page 174. Of particular note is the excellent tutorial by Shahid Qureshi [1], after which this article is loosely patterned.

Of particular interest today is the area of digital cellular communications, which has seen wide use of fixed-point DSPs such as the TMS320C5x. This family of processors provides the processing power to perform the requisite adaptive equalization while at the same time handling such tasks as channel coding, error correction (Viterbi algorithm), and vocoding functions (VSELP), thus providing a highly integrated and yet flexible solution to baseband processing. The last section of this paper provides a brief survey of adaptive equalization for digital cellular systems. For a detailed application example, please see the application report ***Channel Equalization for the IS-54 Digital Cellular System With the*** *TMS320C5x* on page ***177.***

What Is Intersymbol Interference?

Consider what happens when pulsed information is transmitted over an analog channel such as a phone line or airwaves. Even though the original signal is a discrete time sequence (or a reasonable approximation). the received signal is a continuous time signal. Heuristically, one can consider that the channel acts as an analog low-pass filter, thereby spreading or smearing the shape of the impulse train into a continuous signal whose peaks relate to the amplitudes of the original pulses. Mathematically, the operation can be described as a convolution of the pulse sequence by a continuous time channel response.

The operation starts with the convolution integral:

$$r(t) = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \tag{1}$$

where r(t) is the received signal, h(t) is the channel impulse response, and x(t) is the input signal. The second half of the equation above is a result of the fact that convolution is a commutative operation.

Component x(t) is the input pulse train, which consists of periodically transmitted impulses of varying amplitudes. Therefore,

$$x(t) = 0 \text{ for } t \neq kT \tag{2}$$

$$x(t) = X_k \text{ for } t = kT \tag{3}$$

where T represents the symbol period. This means that the only significant values of the variable of integration in the above integral are those for which $\tau = kT$. Any other value of τ amounts to multiplication by 0. Therefore r(t) can be written as

$$r(t) = \sum_{k=-\infty} x_k h(t - kT) \tag{4}$$

This representation of r(t) more closely resembles the convolution sum familiar to DSP engineers. Note, however, that it still describes a continuous time system. It shows that the received signal consists of the sum of many scaled and shifted continuous time system impulse responses. The impulse responses are scaled by the amplitudes of the transmitted pulses of x(t).

As an example, consider the calculation for r(t) at some noninteger time index (t = 1.1)

$$r(1.1) = \cdots + x_{-2}h(1.1 + 2T) + x_{-1}h(1.1 + T) + x_0 h(1.1) + x_1 h(1.1 - T) + x_2 h(1.1 - 2T) \cdots \tag{5}$$

One can see how received values for any time t are computed. Each pulse value of the input sequence, x_k, contributes a component of the output summation.

Because you are interested in processing the received signal on digital hardware, you must represent the received signal as a difference equation. Physically, you are periodically sampling the received waveform. For the case of pulse-amplitude modulation, it is sufficient to sample the received signal at the symbol transmit rate, $1/T^2$. (In some instances it can be advantageous to sample at a multiple of the symbol rate to implement a *fractionally* spaced signal processing system.) To represent the sampling mathematically, replace t with nT, where, again, T is the symbol transmit rate:

$$r(nT) = \sum_{k=-\infty} x_k h(nT - kT) \tag{6}$$

which can also be written as

$$r(nT) = x_n h(0) + \sum_{k \neq \infty} x_k h(nT - kT) \tag{7}$$

One last factor to account for is sampling phase. Unless the sample clock is perfectly synchronized with the transmit clock, the sample-phase offset will be nonzero. To account for an arbitrary phase offset in the equation above, add an offset t_0 to the time index.

$$r(nT + t_0) = x_n h(t_0) + \sum_{k \neq \infty}^{N} x_k h(t_0 + nT - kT)$$

In the equation above, the first term is the component of r(t) due to the Nth symbol. It is multiplied by the center tap of the channel-impulse response. The other product terms in the summation are intersymbol interference (ISI) terms. The input pulses in the neighborhood of the Nth symbol are scaled by the appropriate samples in the tails of the channel-impulse response. Below are numerical examples for various values of n with $t_0 = 0.1$ for values of k spanning the five sample neighborhoods around n.

$$r(0.1) = x_0h(0.1) + x_{-2}h(2.1) + x_{-1}h(1.1) + x_1h(-0.9) + x_2h(-1.9)\cdots (n = 0) \tag{9}$$

$$r(1.1) = x_1h(0.1) + x_{-1}h(2.1) + x_0h(1.1) + x_2h(-0.9) + x_3h(-1.9)\cdots (n = 1) \tag{10}$$

$$r(2.1) = x_2h(0.1) + x_0h(2.1) + x_1h(1.1) + x_3h(-0.9) + x_4h(-1.9)\cdots (n = 2) \tag{11}$$

Figure 1 illustrates a pulse train to be transmitted. The center pulse is x_0, the pulse at 1 is x_1, the pulse at –1 is x_{-1}, etc. If you assume an arbitrary impulse response for the transmission channel, you can construct the received signal r(t). This signal is shown superimposed on the transmit waveform x(t). In actuality, the received waveform would be time shifted because of the channel delay, but for clarity r(t) is shown with no delay relative to x(t). Note that the peaks of r(t) roughly relate to the sense of the corresponding transmit pulses; however, the value of r(t) at the sample instants can be quite different from those transmitted. This is because of ISI effects.

Figure 1. A Pulse Train to Be Transmitted

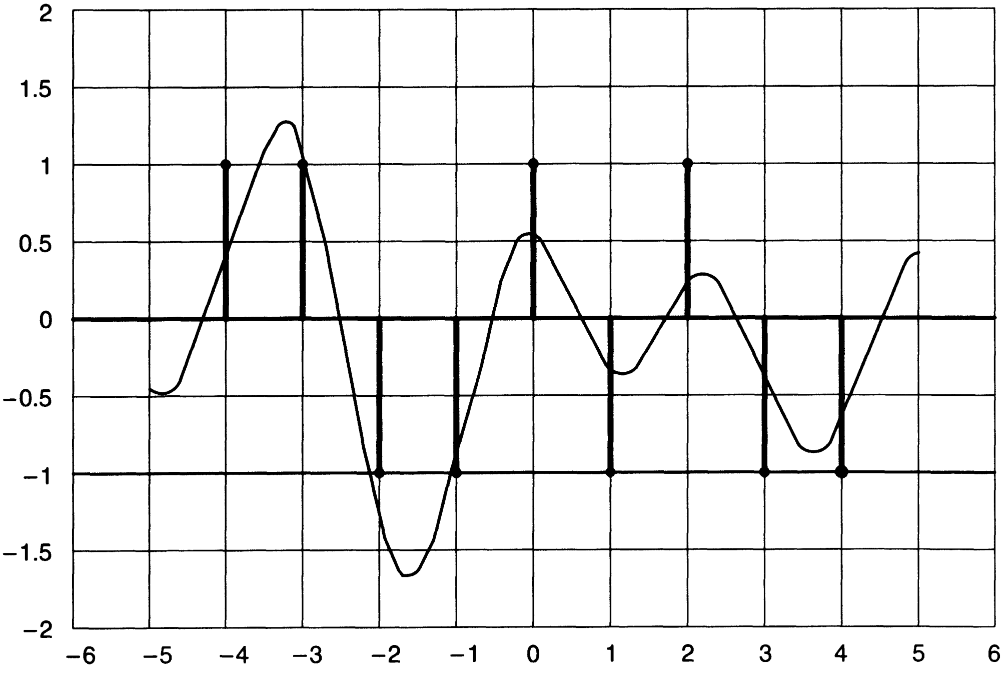
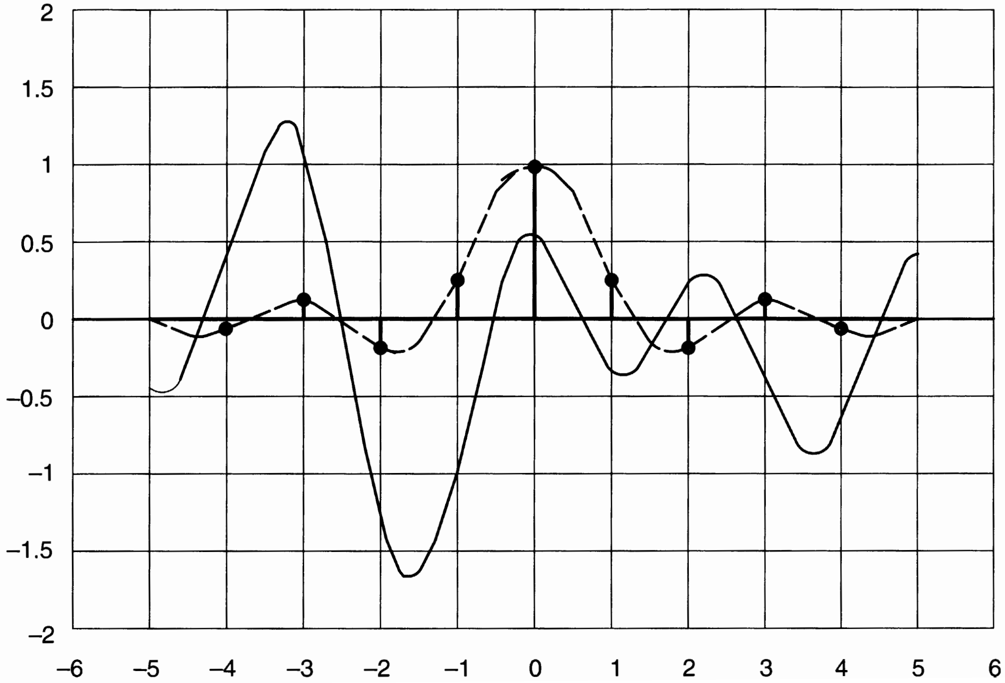


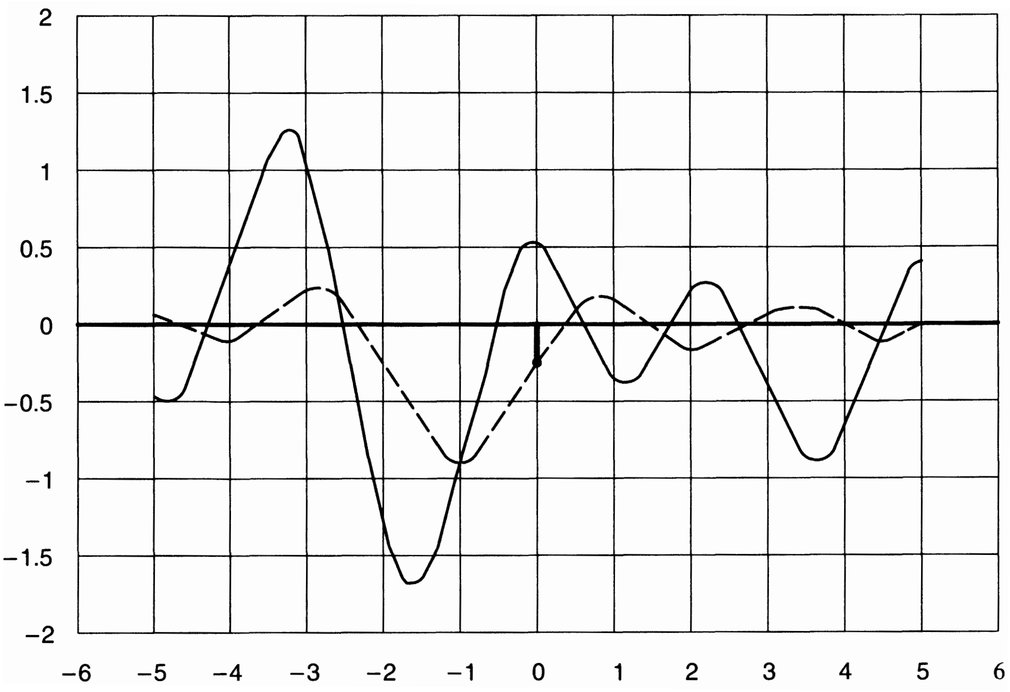
Figure 2 shows the component of r(t) due to a single input pulse x_1, which is superimposed on the received signal r(t). Recall that the shape of this component is the same as that of the transmit-channel impulse response. The values of this individual pulse response at the sample periods (which are multiples of T) are indicated by the black dots. Note that although the signal component in this example is sinc shaped, the nulls *do not* occur at the sample interval. Therefore, the pulse response centered at t=0 makes undesirable contributions to the neighboring received samples of r(t). The contribution of the x_0 symbol to r(0) is the value + 1.

Figure 2. Component of r(t)



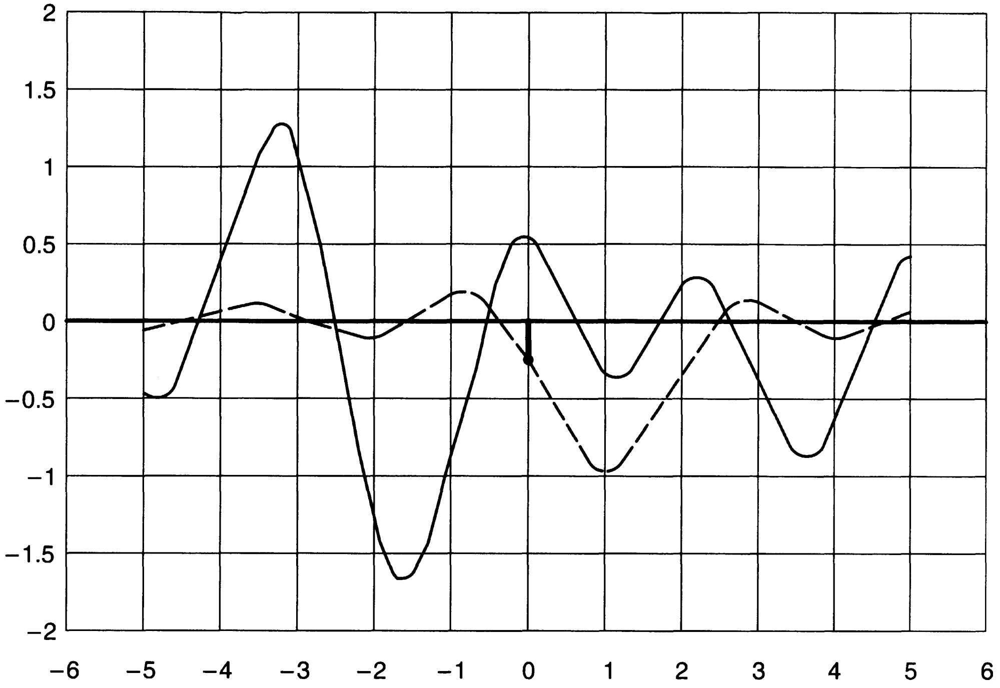
To determine the value of the received signal at t=0, r(0), sum the contributions of the received impulse responses due to x_0, x_{-1}, x_1, x_{-2}, x_2... .

Figure 3. Contribution Due to x_{-1}



As shown in Figure 3, the contribution due to x_{-1} is the value at t=O of the scaled and shifted impulse response corresponding to the x_{-1} transmit pulse. In this case the impulse response is scaled by −1, which is the value of x_{-1} and is advanced by one sample period because x_{-1} is transmitted one period prior to x_0. Therefore, the x_{-1} symbol results in a small negative component of r(0).

Figure 4. Contribution Due to x_1 at t=O



Similar reasoning explains the contribution due to x_1 , except this time use the value of the time-delayed impulse response at t=0 as illustrated in Figure 4.

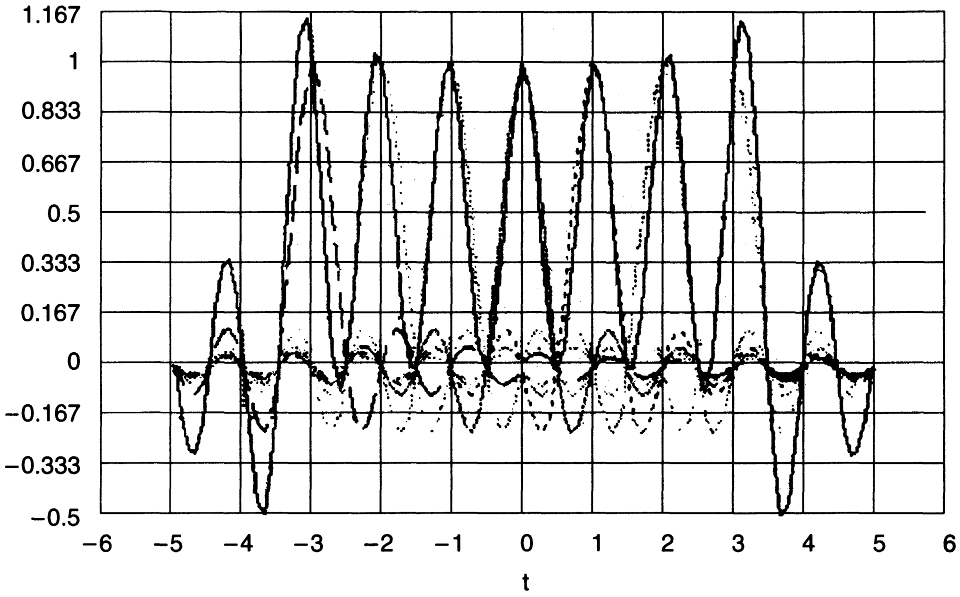
The received value of r(t=0) is computed by summing the contribution of x_0 plus all of the ISI terms; that is, $x_{+/-1}$, $x_{+/-2}$, $x_{+/-3}$, ... Theoretically, this is an infinite sum, but as shown here, the channel response is typically a decaying exponential. Therefore, in practice, an FIR system can be used to model and compensate the system.

From the example above you can see that the nth received sample is primarily influenced by the nth symbol transmitted; however, there are ISI components contributed by prior and subsequent transmit symbols. The terms due to prior symbols (x_{n-1} and before) are termed ***postcursor*** ISI [3] because the nth transmitted symbol affects on symbols following the nth received symbol. The nature of this ISI can be determined by examining the right-hand portion of the system impulse response. Alternately, the ISI terms due to subsequent transmit symbols (x_{n+1} and beyond) exert ***precursor*** ISI *[3]*because the nth transmit symbol influences received symbols prior to the nth. These ISI terms are determined by the shape of the left-hand portion of the system impulse response.

Pulse Shaping

From the preceding figures, it is apparent that ISI is caused when the tails of the received pulses overlap at the sample points, causing uncertainty in the received pulse amplitude. It is possible to shape the transmit pulses in a manner designed to minimize the effects of ISI on the received waveform. As shown in Figure 5, the set of shifted pulse responses overlap, but their tails all possess nulls at the sample instants. Therefore, the only contribution to r(nT) is due to the nth transmit pulse. As shown below, the received signal r(t) equals the amplitude of the individual sinc functions at the sample instants. Compare this with the previous example in which r(t) has a more ambiguous relationship to the individual pulse responses.

Figure 5. Set of Shifted Pulse Responses



If a received pulse shape can meet the following property, zero ISI can be achieved:

$$p_r(nT) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} \tag{12}$$

Equation (12) simply means that there are zero crossings at the sample rate. It can be shown that this results in a spectrum possessing *vestigial symmetry*. That is, the frequency response exhibits odd symmetry about 1/2T, causing the sum of repeat spectra to equal a constant. It is important to note that this spectrum may be closely approximated by a realizable filter having a gradual rolloff around 1/2T.

Figure 6. Odd Symmetry

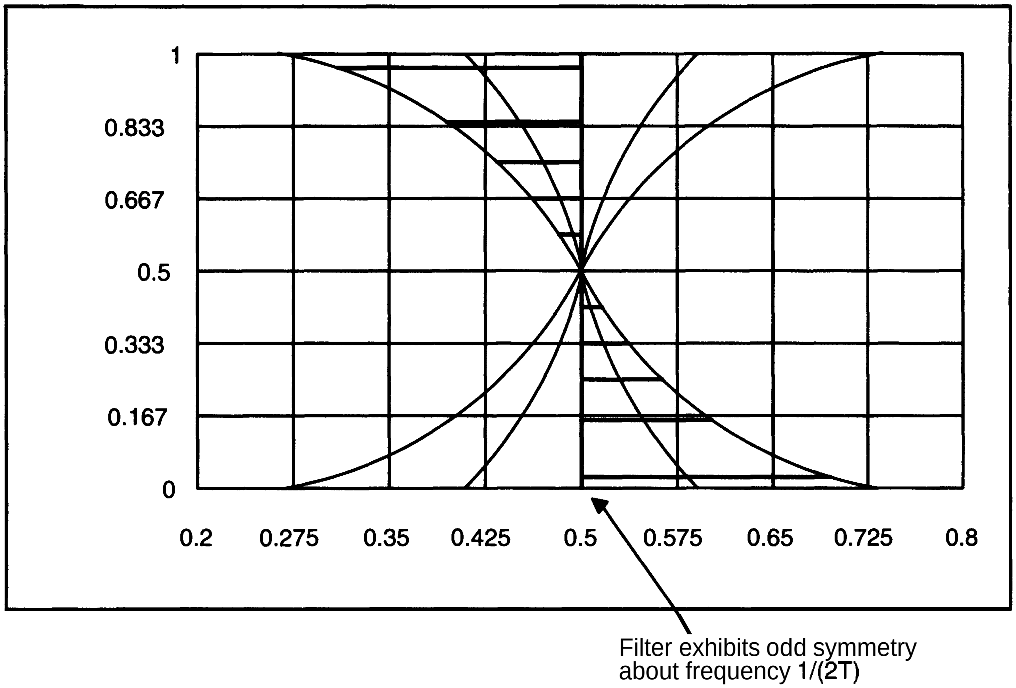


Figure 6 illustrates the notion of odd symmetry.

Figure 7. Spectral Response at 1/(2T)

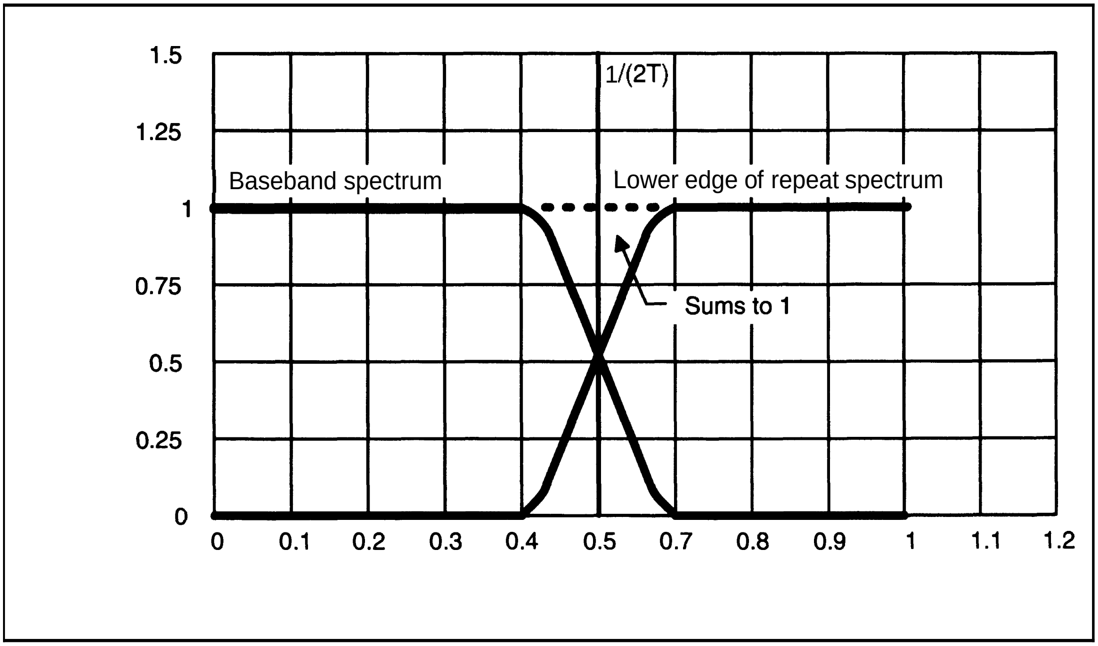


Figure 7 shows the spectral response around 1/(2T). The repeat spectra centered at 1/T actually overlaps the baseband spectrum, but as long as the sum of the two responses is constant, the criterion for zero ISI is met.

One class of linear phase filters possessing vestigial symmetry is the ***raised cosine*** family:

$$P_{rc}(f) = \begin{cases} T & |f| \leq 1/(2T)-\beta \\ \frac{1}{2}T\left[1 + \cos\left(\frac{\pi|f|-1/(2T)+\beta}{2\beta}\right)\right] & 1/(2T)-\beta < |f| \leq 1/(2T)+\beta \\ 0 & |f| > 1/(2T)+\beta \end{cases} \tag{13}$$

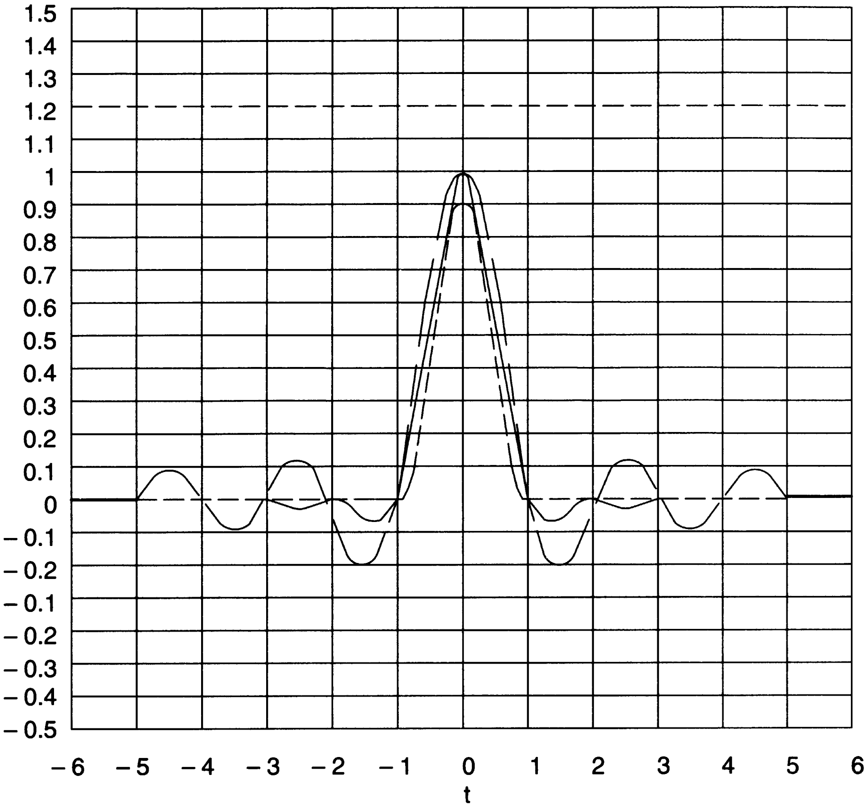
This filter is flat up to $1/(2T)-\beta$ and 0 beyond $1/(2T)+\beta$. The complicated part of the equation above describes the shape of the odd symmetric transition band. Closer inspection of the equation for the transition band quickly reveals the shape of the signal. It is really the cosine of an argument ranging from 0 to π with a DC offset of +1, hence ***raised cosine.*** The other variables scale the backwards S shape in the x and y dimension to fit the curve into the flat portions of the response.

The impulse response of the signal possessing the raised cosine spectrum is as follows:

$$P_{rc}(t) = \frac{\cos 2\pi\beta t}{1-(4\beta t)^2}\,\mathrm{sinc}\left(\frac{t}{T}\right) \tag{14}$$

Note that the Equation above can be broken into two parts: the familiar sinc function, which insures that the product will have nulls at multiples of T, and a second term that is an exponentially decaying sinusoid whose rate of decay is proportional to β. The time response of the raised cosine signal for various values of β is shown in Figure 8.

Figure 8. Time Response of the Raised Cosine Signal



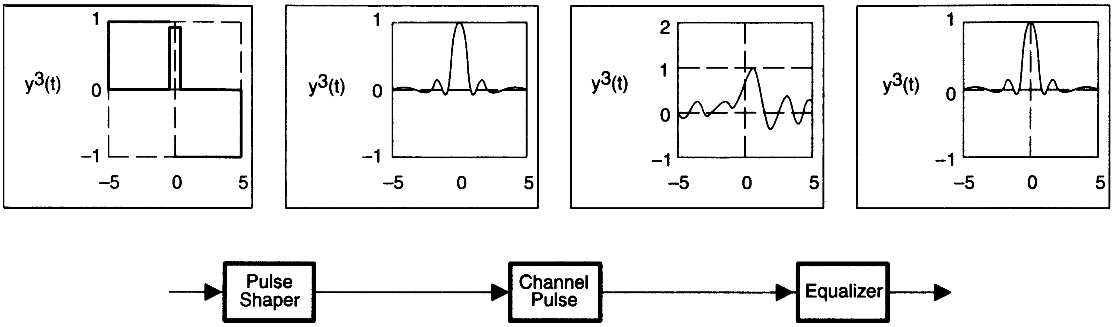
It is common practice to filter the signal pulses at the transmitter with the frequency characteristic described above in Figure 8. Having performed this operation, if the signals are sent down an ideal channel (that is, a channel with a channel-impulse response of δ function and no noise), the received signal should exhibit no ISI. Note that in general this condition will not be met, as the channel will have its own shaping effect on the transmitted signals. The effects of noise are considered in the next section of this paper.

In summary, the obvious problem caused by ISI is uncertainty in the received data samples. Instead of receiving the discrete levels that were transmitted, the receiver finds a continuous signal whose samples can take on any value. The receiver must then form an estimate from the received values to decide on the transmitted signal.

Equalization

As discussed in the ***Pulse Shaping*** subsection on page 155, a properly shaped transmit pulse resembles a sinc function, and direct superposition of these pulses results in no ISI at properly selected sample points. In practice, however, the received pulse response is distorted in the transmission process and may be combined with additive noise. Because the raised cosine pulses are distorted in the time domain, you may find that the received signal exhibits ISI. If you can define the channel impulse response, you can implement an inverse filter to counter its ill effect. This is the job of the equalizer. See Figure 9 below, which depicts the response to a single transmit pulse at various points in the system.

Figure 9. Transmission Process With Example Pulse Responses

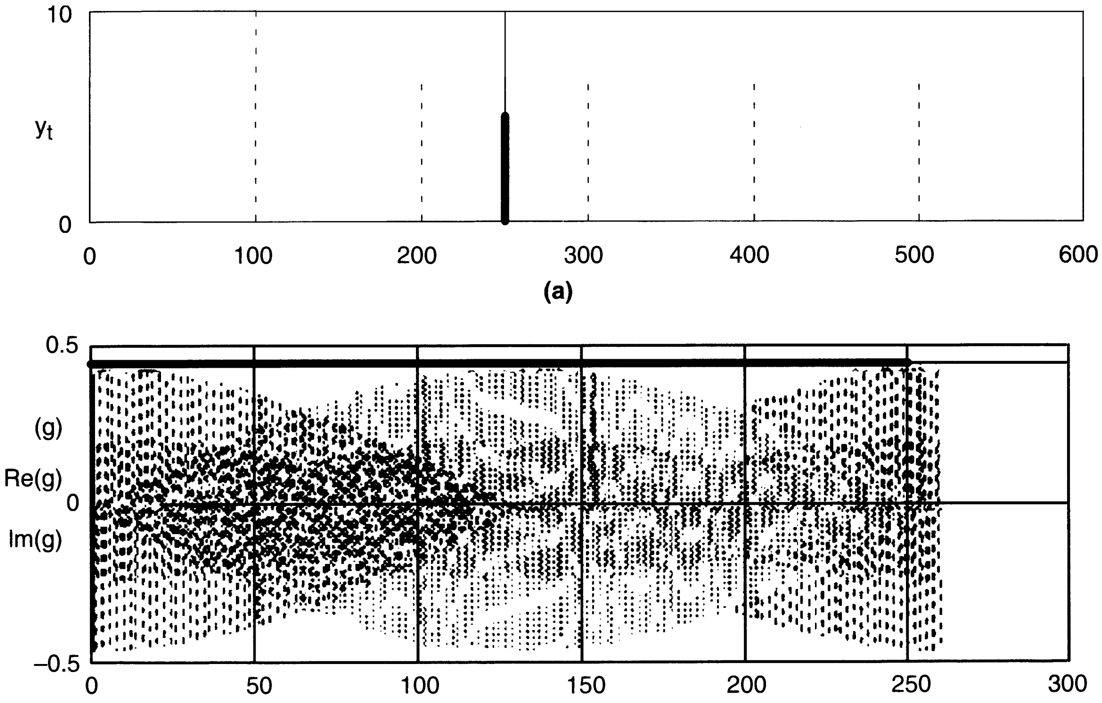


The original rectangular transmit pulse is shaped by the raised cosine filter. This ensures that the sampled spectra do not alias and therefore there is no ISI. The next waveform portrays the distorted impulse response received at the input of the equalizer. This distortion can be caused by spectral shaping due to a nonflat frequency response or multipath reception of the channel. This distortion can be removed by applying a filter that is the exact inverse (multiplicative inverse in spectral domain) of the channel frequency response.

Multipath Effects on Frequency Response

Multipath effects describe the situation in which there are several propagation paths from transmitter to receiver. Most commonly, this results when there are reflected signals detected at the receiver following the direct path. The multipath phenomenon can be modeled by an FIR system. The center tap represents the direct path, while the succeeding tap weights represent the amplitudes, delays, and phases of the reflected paths. What does this look like in the spectral domain? For simple examples, see the two cases described in Figure 10 and Figure 11.

Figure 10. Case 1: Ideal Channel, No Multipath Effects



(a)

(b)

Figure 10(a) above shows the time response of an ideal transmission path, which is a δ function. Such a channel exerts no spectral distortion or delayed signals. Figure 10(b) shows the spectral response of such a system. Note that the frequency magnitude response is perfectly flat, as indicated by the solid horizontal line.

Figure 11. Case 2: System With a Single Unattenuated Multipath Channel

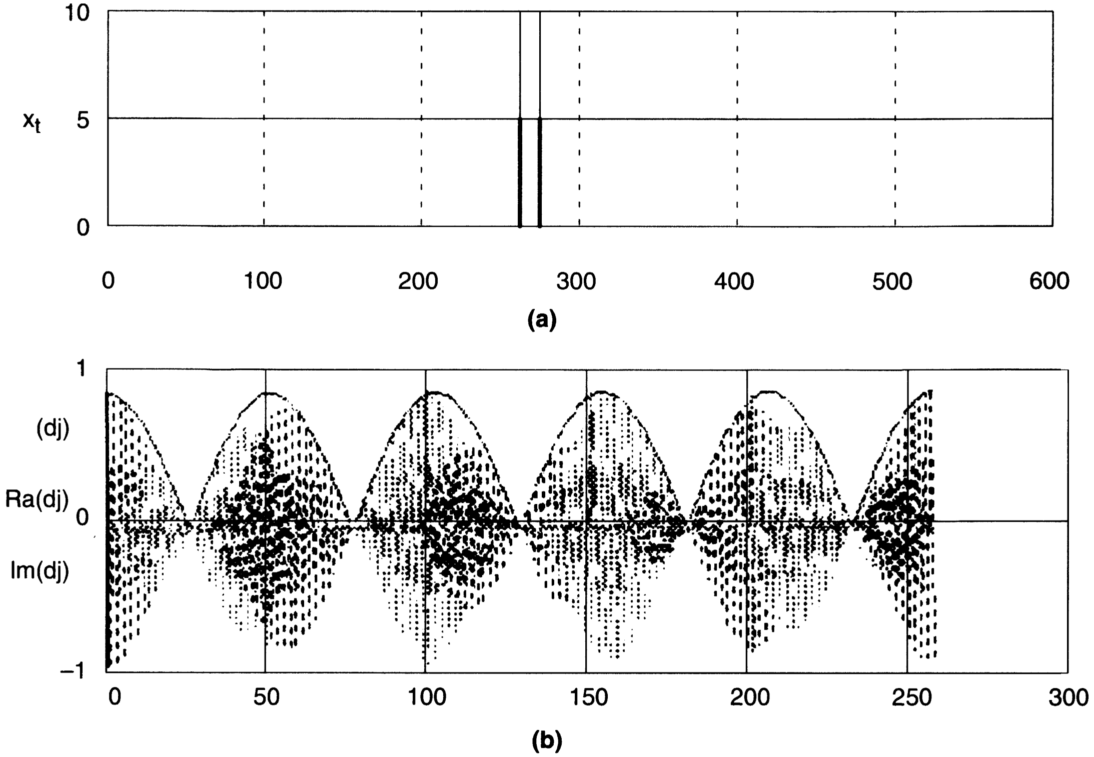
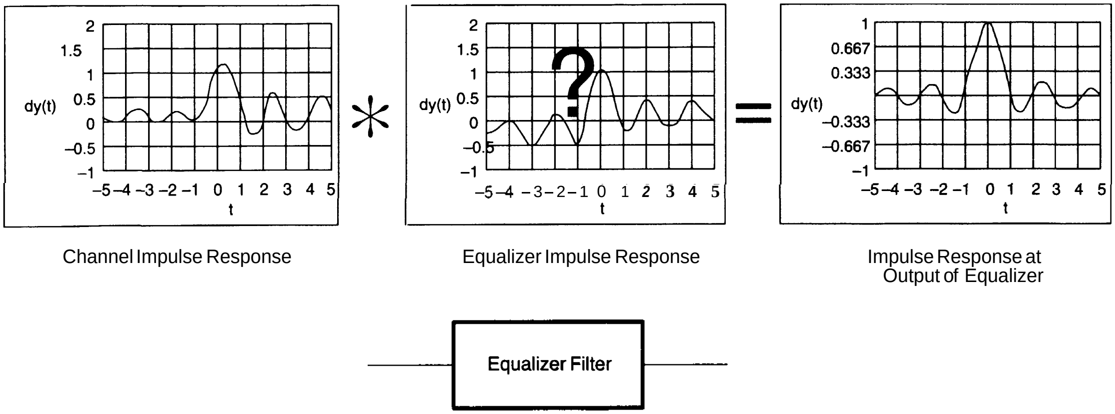


Figure 11(a) shows the time response of a system that contains a single multipath channel. The first nonzero sample of the response represents the direct path, while the second represents a delayed path to the receiver. In this instance, the pulses are identical in amplitude and phase and are separated by ten sample intervals. Notice in Figure 11(b) that the magnitude response exhibits $t_0/2$ nulls, where t_0 represents the sample delay. Even though you are effectively adding two identical flat spectra (as shown in Figure 10(b)), the time delay results in a phase delay in the spectral domain. This phase delay results in nulls where the two signals are of equal amplitude but opposite phase.

Obviously, multipath effects can have major effects on the system spectral response, thereby providing another justification for channel equalization.

Figure 12. Equalization Process



Channel Impulse Response

Equalizer Impulse Response

Impulse Response at Output of Equalizer

As depicted in Figure 12, the task of the equalization system is to determine and apply a filter that results in an equalized impulse response having zero ISI and channel distortion. This means that convolution of the channel impulse response and the equalizer impulse response must equal 1 at the center tap and have nulls at the other sample points within the filter span.

Two main techniques are employed to formulate the filter coefficients: *automatic synthesis* and *adaptation*. In automatic-synthesis methods, the equalizer typically compares a received time-domain reference signal to a stored copy of the undistorted training signal. By comparing the two, a time-domain error signal is determined that may be used to calculate the coefficient of an inverse filter. The formulation of this inverse filter may be accomplished strictly in the time domain, as is done in ZFE and LMS systems, which are examined in more detail in following sections. Other methods involve conversion of the received training signal to a spectral representation. A spectral inverse response can then be calculated to compensate for the channel response. This inverse spectrum is then converted back to a time-domain representation so that filter tap weights may be extracted.

The second method of filter synthesis is adaptation. In adaptation the equalizer attempts to minimize an error signal based on the difference between the output of the equalizer z_k and the *estimate* of the transmitted signal $\hat{x}_k$, which is generated by a decision device. In other words, the equalizer filter outputs a sample. The predictor or decision device determines what value was most likely transmitted. The adaptation logic endeavors to keep the difference between the two small. The main idea is that the receiver takes advantage of the knowledge of the discrete levels possible in the transmitted pulses. When the decision device quantizes the equalizer output, it is essentially throwing away received noise.

The main drawback of automatic synthesis is the overhead associated with the transmission of a training signal, which must be at least as long as the filter tap length. Typically, training is used to converge a filter at startup as part of the initialization overhead. Adaptation techniques can then be employed to track and compensate for minor variations in channel response on the fly [1].

Zero-Forcing Equalization

One computationally efficient method of forming an inverse filter is the *zero-forcing* technique. To formulate a set of FIR inverse filter coefficients, a training signal consisting of an impulse is transmitted over the channel. By solving a set of simultaneous equations based on the the received sample values, a set of coefficients can be determined to force all but the center tap of the filtered response to 0. This means the $N-1$ samples surrounding the center tap will not contribute ISI. The main advantage of this technique is that the solution to the set of equations is reduced to a simple matrix inversion.

The major drawback of ZFE is that the channel response may often exhibit attenuation at high frequencies around one-half the sampling rate (the folding frequency). Since the ZFE is simply an inverse filter, it applies high gain to these upper frequencies, which tends to exaggerate noise. **A** second problem is that the training signal, an impulse, is inherently a low-energy signal, which results in a much lower received signal-to-noise ratio than could be provided by other training signal types [1, 4].

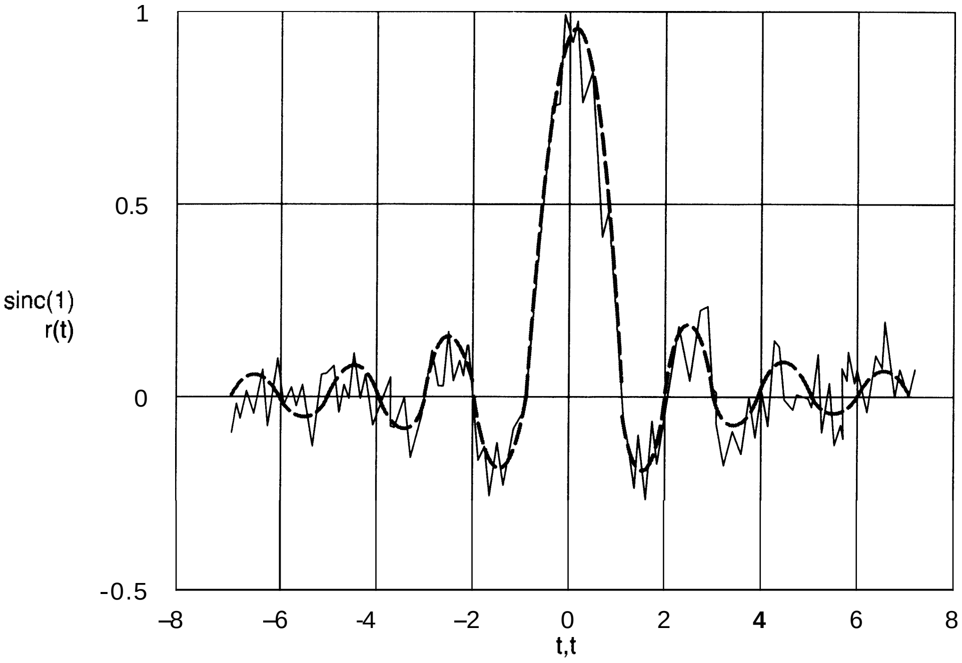
Example of 7-Tap ZFE Computation

First, create a simulated received pulse response. Begin with the equation of a sinc function, which is a simplification of the raised cosine pulse. Then simulate additive noise by the addition of random thermal noise. Finally, simulate sampling phase jitter with the random jitter term added to the time index. The simulated pulse response is plotted in Figure 13. The dotted trace represents the ideal noiseless channel response.

Figure 13. Simulated Pulse Response

$sinc(t) := \frac{\sin(\pi \cdot t)}{\pi \cdot t}$ $\qquad noise(t) := \mathrm{rnd}(0.2) - 0.1$ $\qquad jitter(t) := \mathrm{rnd}(0.10) - 0.05$

$t := -7, -6.9, 7.0$ $\qquad r(t) := sinc(t + jitter(t)) + \mathrm{noise}(t)$



A vector is formed from the received samples. 2N−1 samples are required to implement an N-tap filter. For the example 7-tap ZFE, you must collect 13 samples. Therefore, 13 equally spaced samples of **r(t)** are formed into the column vector V.

$$i := 0 \ldots 12 \quad v_i := r(i-6) \qquad v = \begin{bmatrix} 0.083 \\ -0.032 \\ 0.042 \\ 0.026 \\ -0.072 \\ 0.002 \\ 0.915 \\ 0.011 \\ -0.105 \\ 0.031 \\ -0.076 \\ 0.002 \\ 0.04 \end{bmatrix} \tag{15}$$

Next, form a matrix, PR, from the received samples. Each row consists of seven adjacent samples in time-reversedorder. The first element of the top row is the center tap of the pulse response. The first element of the second row is the sample following the center tap, etc.

$$i := 0 \ldots 6 \quad PR_{0,i} := v_{6-i} \quad PR_{1,i} := v_{7-i} \quad PR_{2,i} := v_{8-i}$$

$$PR_{3,i} := v_{0-i} \quad PR_{4,i} := v_{10-i} \quad PR_{5,i} := v_{11-i} \quad PR_{6,i} := v_{12-i}$$

$$PR = \begin{bmatrix} 0.915 & 0.002 & -0.072 & 0.026 & 0.042 & -0.032 & 0.083 \\ 0.011 & 0.915 & 0.002 & -0.072 & 0.026 & 0.042 & -0.032 \\ -0.105 & 0.011 & 0.915 & 0.002 & -0.072 & 0.026 & 0.042 \\ 0.031 & -0.105 & 0.011 & 0.915 & 0.002 & -0.072 & 0.026 \\ -0.076 & 0.031 & -0.105 & 0.011 & 0.915 & 0.002 & -0.072 \\ 0.002 & -0.076 & 0.031 & -0.105 & 0.011 & 0.915 & 0.002 \\ 0.04 & 0.002 & -0.076 & 0.031 & -0.105 & 0.011 & 0.915 \end{bmatrix} \tag{16}$$

Next, compute the inverse of the channel response matrix PREQ.

$$PREQ := PR^{-1} \qquad PREQ = \begin{bmatrix} 1.101 & -9.62\times10^{-4} & 0.07 & -0.023 & -0.057 & 0.036 & -0.106 \\ -0.02 & 1.099 & -0.004 & 0.08 & -0.026 & -0.045 & 0.036 \\ 0.136 & -0.019 & 1.108 & -0.01 & 0.075 & -0.026 & -0.057 \\ -0.042 & 0.135 & -0.023 & 1.114 & -0.01 & 0.08 & -0.023 \\ 0.107 & -0.043 & 0.141 & -0.023 & 1.108 & -0.004 & 0.07 \\ -0.015 & 0.108 & -0.043 & 0.135 & -0.019 & 1.099 & -9.62\times10^{-4} \\ -0.023 & -0.015 & 0.107 & -0.042 & 0.136 & -0.02 & 1.101 \end{bmatrix} \tag{17}$$

The center column of PREQ contains the coefficients of the ZFE.

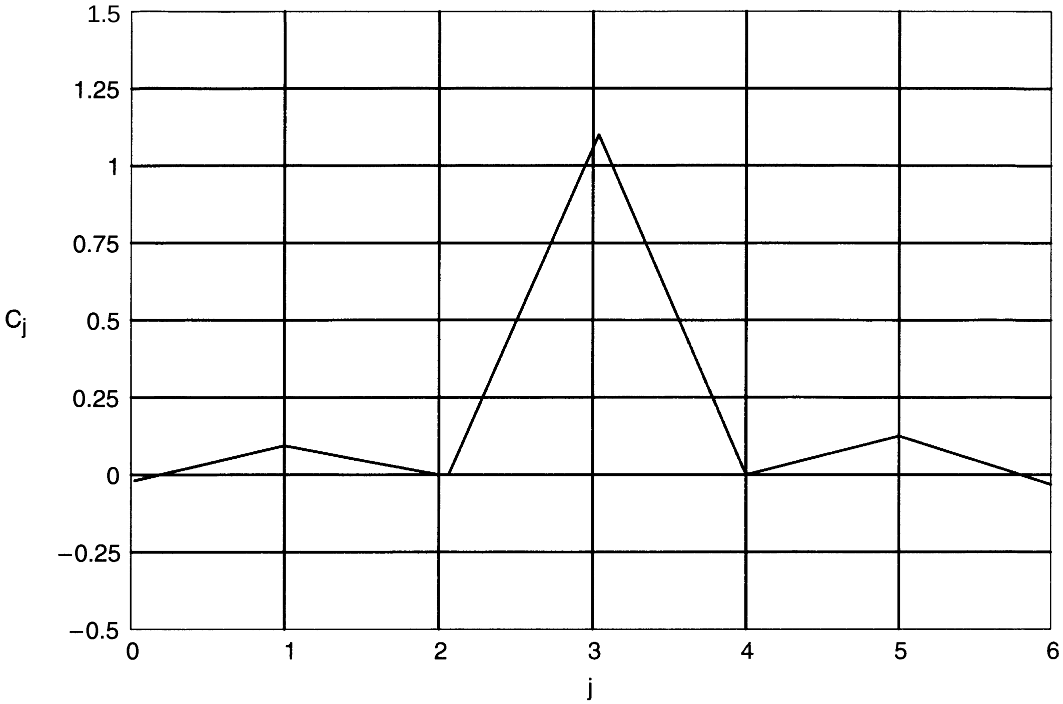
$$j := 0...6 \qquad C_j := PREQ_{j,3} \qquad C = \begin{bmatrix} -0.023 \\ 0.08 \\ -0.01 \\ 1.114 \\ -0.023 \\ 0.135 \\ -0.042 \end{bmatrix} \tag{18}$$

Check the results by multiplying the coefficient vector by the row vectors of the received sample matrix. The dot products should result in the ideal channel response for the filter span, that is, 0, 0, 0, 1, 0, 0, 0. As shown below, the results check.

$$\begin{array}{lll} ROW0_j := PR_{0,j} & z := ROW0 \bullet C & z = 0 \\ ROW1_j := PR_{1,j} & z := ROW1 \bullet C & z = 0 \\ ROW2_j := PR_{2,j} & z := ROW2 \bullet C & z = 0 \\ ROW3_j := PR_{3,j} & z := ROW3 \bullet C & z = 1 \\ ROW4_j := PR_{4,j} & z := ROW4 \bullet C & z = 0 \\ ROW5_j := PR_{5,j} & z := ROW5 \bullet C & z = 0 \\ ROW6_j := PR_{6,j} & z := ROW6 \bullet C & z = 0 \end{array} \tag{19}$$

The coefficients for the ZFE filter response are shown plotted in Figure 14.

Figure 14. ZFE Filter Coefficient



Because the Japanese television broadcasters employed an impulse-like training signal, many of the first video deghosters for use in Japan employed ZFEs. To provide a higher signal-to-noise ratio (SNR) for the received training signal, these systems averaged the training signal over several training intervals. To further improve SNR, the U.S. broadcast industry has selected a chirp-like training signal, which has inherently higher energy. This signal, transmitted during the vertical blanking interval, allows suitably equipped receivers to automatically synthesize filters to alleviate the effects of multipath interference; that is, visible *ghost* images.

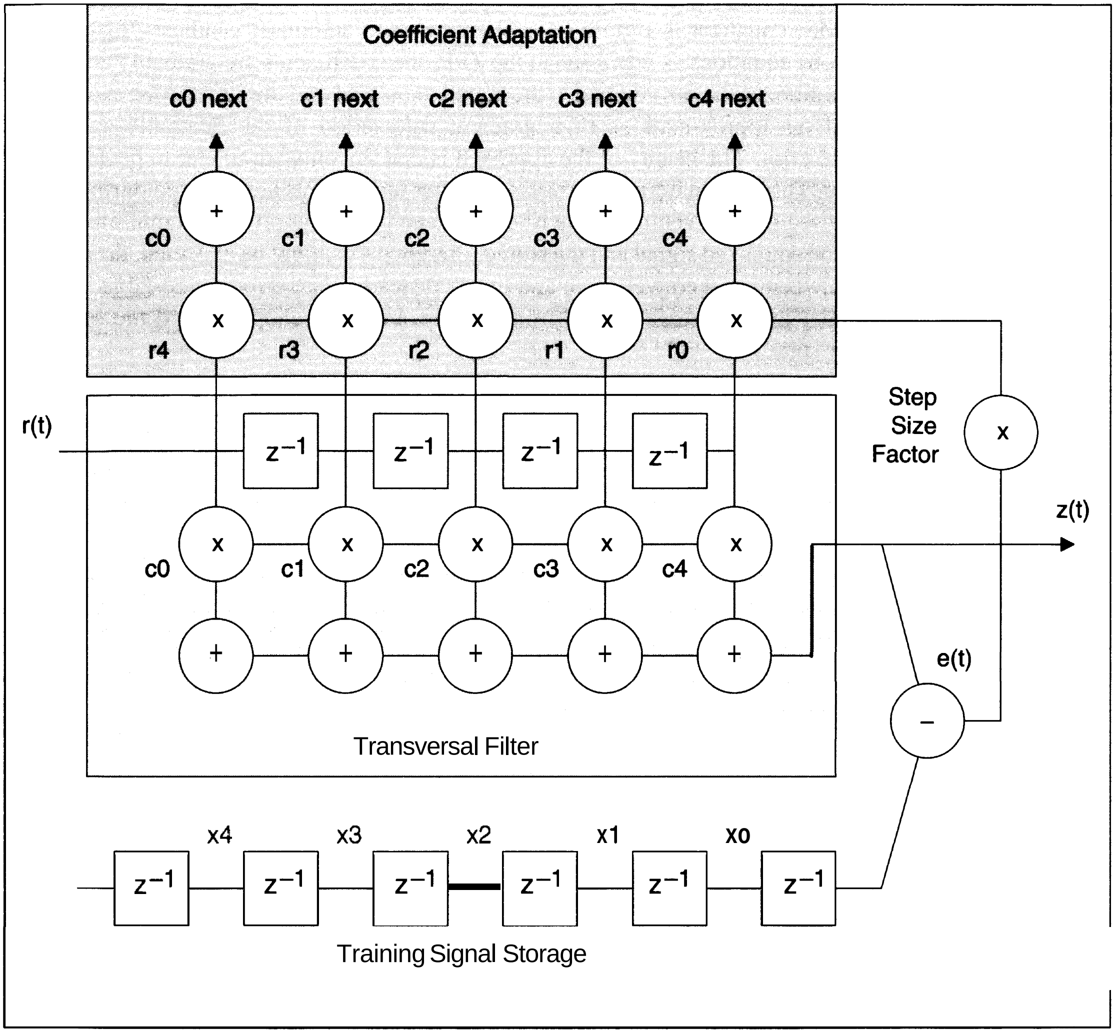
LMS Equalization

The least mean squared (LMS) equalizer is a more general approach to automatic synthesis. Instead of solving a set of N simultaneous equations as was done in the ZFE, the coefficients are gradually adjusted to converge to a filter that minimizes the error between the equalized signal and the stored reference. The filter convergence is based on approximations to a gradient calculation of the quadratic equation representing the mean square error. The beauty of the approach is that the only parameter to be adjusted is the adaptation step size a a . Through an iterative process, all filter tap weights are adjusted during each sample period in the training sequence. Eventually, the filter will reach a configuration that minimizes the mean square error between the equalized signal and the stored reference. As might be expected, the choice of a a involves a tradeoff between rapid convergence and residual steady-state error. A too-large setting for a a can result in a system that converges rapidly on start-up, but then chops around the optimal coefficient settings at steady state.

The LMS equalizer can also be shown to have better noise performance than the ZFE. Heuristically, the ZFE calculates coefficients based upon the received samples of one training signal. Since the captured data will always contain some noise, the calculated coefficients will be noisy — noise in / noise out. On the other hand, the LMS algorithm gradually adapts a filter based on many cycles of the training signal. If the noise is zero mean and is averaged over time, its effect will be minimized — noise integrates to 0.

A second major benefit to this approach is that you can employ any arbitrary training sequence. In general, you would prefer to use a high-energy signal to improve the received signal-to-noise ratio of the training sequence. In contrast, the unit impulse training signal required by the ZFE is probably the lowest energy flat-spectrum signal possible. Typical training sequences employed for LMS equalization include pseudorandom noise sequences and chirp-type signals.

Figure 15. Filter Output Computation



Coefficient Adaptation
c0 next
c1 next
c2 next
c3 next
c4 next
+
c0
c1
c2
c3
c4
x
r4
r3
r2
r1
r0
r(t)
z−1
Step Size Factor
z(t)
e(t)
−
Transversal Filter
x4
x3
x2
x1
xo
Training Signal Storage

In Figure 15, the portion in the lower shaded rectangle is a standard transversal filter (FIR). The lower set of delays represents storage for the reference version of the training signal. Each time a sample is received, a new filter output is computed and compared to the corresponding reference signal, thereby forming an error signal. This error signal is then used to scale the received sample values contained in the filter storage elements. These scaled sample values are then added to the current filter coefficients to form the updated coefficients to be used at the next sample time.

The coefficients are updated according to the following equation:

$$C_n(k+1) = C_n(k) - \alpha e_k r_{k-n}, n = 0, 1\ldots, N-1 \qquad \textbf{(20)}$$

As an example, consider the calculation for the third tap weight (n = 2) at time k = 5:

$$C_2(5) = C_2(4) - \alpha e_4 r_2 \qquad \textbf{(21)}$$

This means that the C_2 coefficient for the next sample period equals the current C_2 coefficient minus a correction term. The correction term is simply the current input sample corresponding to the C_2 tap multiplied by the current error value scaled by the adaptation rate term α.

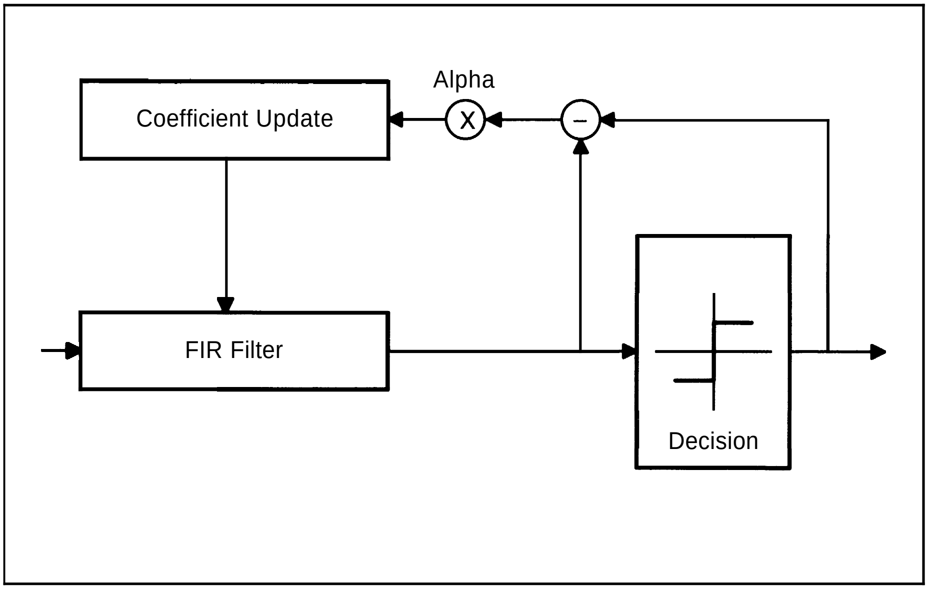
If the filtered sample output is much smaller than the actual value for the training signal, the error is a large negative value. The received sample values are scaled by this relatively large value, and the product is used to adjust the individual coefficients up or down (depending on the sign of the stored sample values) by a relatively large amount. For a smaller discrepancy between the filter output and training signal, the error sample and, hence, the amount of adjustment, will be smaller. The fact that each coefficient is changed by a different adjustment term (based on distinct received samples) allows the filter coefficients to converge from any initial state to one that minimizes the mean square error between the received training signal and the reference.

Decision-Directed Equalization

The previous equalizer systems are linear in that they employ linear transversal filter structures. The filters implement a convolution sum of a computed impulse response with the input sequence. Often with data communication systems, one can take advantage of prior knowledge of the transmit signal characteristics to deduce a more accurate representation of the transmit signal than can be afforded by the linear filter. For instance, a bipolar transmit signal consists of pulses with amplitudes of ±1. This signal is then pulse shaped, distorted by the analog channel, and filtered by a linear FIR filter. The processed signal is no longer a bipolar sequence. Instead, the output values span the range of values representable by the hardware, for example, the range of numbers specified by Q15 notation [5]. It is possible to devise a ***decision device*** (a predictor or a slicer) that estimates what symbol value was most likely transmitted, based on the linear filter continuous output. For example, in the case of the bipolar sequence transmission scheme, a very simple decision device could replace all positive values with a positive 1 and all negative values with a negative 1. The difference between the decision device input and output forms an error term which can then be minimized to adapt the filter coefficients. This is true because a perfectly adapted filter would produce the actual transmitted symbol values, and, therefore, the slicer error term would go to 0. In practice, the error is never 0, but if the adapted filter is near ideal, the decisions are perfect. In this case, the slicer is effectively throwing away received noise with each decision made.

Coefficients can be updated in a manner similar to that employed by the LMS equalizer. There is, however, one important distinction. In Figure 16, the error term is computed as the difference between the input and the output of the decision device, as opposed to the LMS error term, which was based on a stored reference training signal. This means that the decision-directed equalizers do not require a training sequence. This is a major distinction between automatic synthesis (which requires a training signal) and adaptive techniques (which do not require a training signal).

Figure 16. Decision-Directed Equalization


Alpha
Coefficient Update
X
–
FIR Filter
Decision

Decision-Feedback Equalization

Another nonlinear adaptive equalizer should be considered: the decision feedback equalization (DFE). DFE is based on the principle that once you have determined the value of the current transmitted symbol, you can exactly remove the ISI contribution of that symbol to future received symbols (see Figure 17). The nonlinear feature is again due to the decision device, which attempts to determine which symbol of a set of discrete levels was actually transmitted. Once the current symbol has been decided, the filter structure can calculate the ISI effect it would tend to have on subsequent received symbols and compensate the input to the decision device for the next samples. This postcursor ISI removal is accomplished by the use of a feedback filter structure.

Figure 17. Received Signal Including Additive Noise Effects

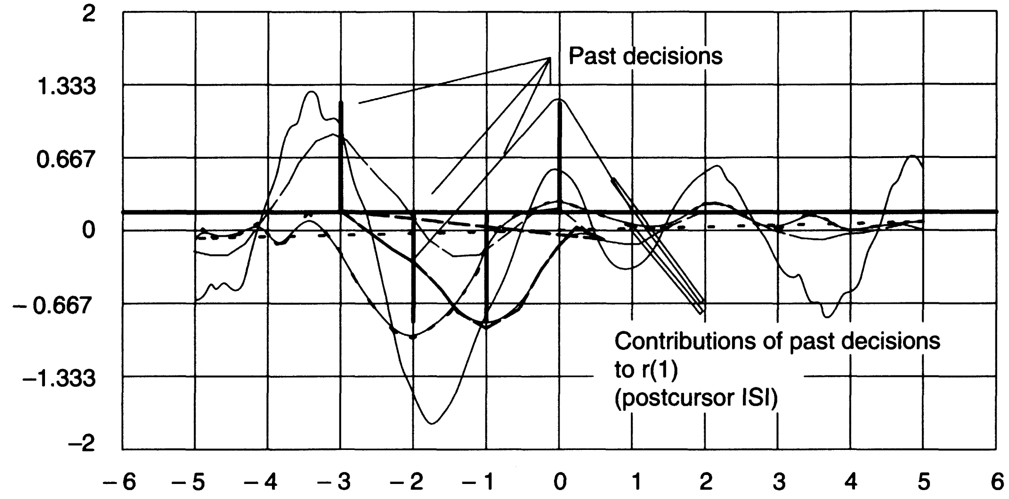
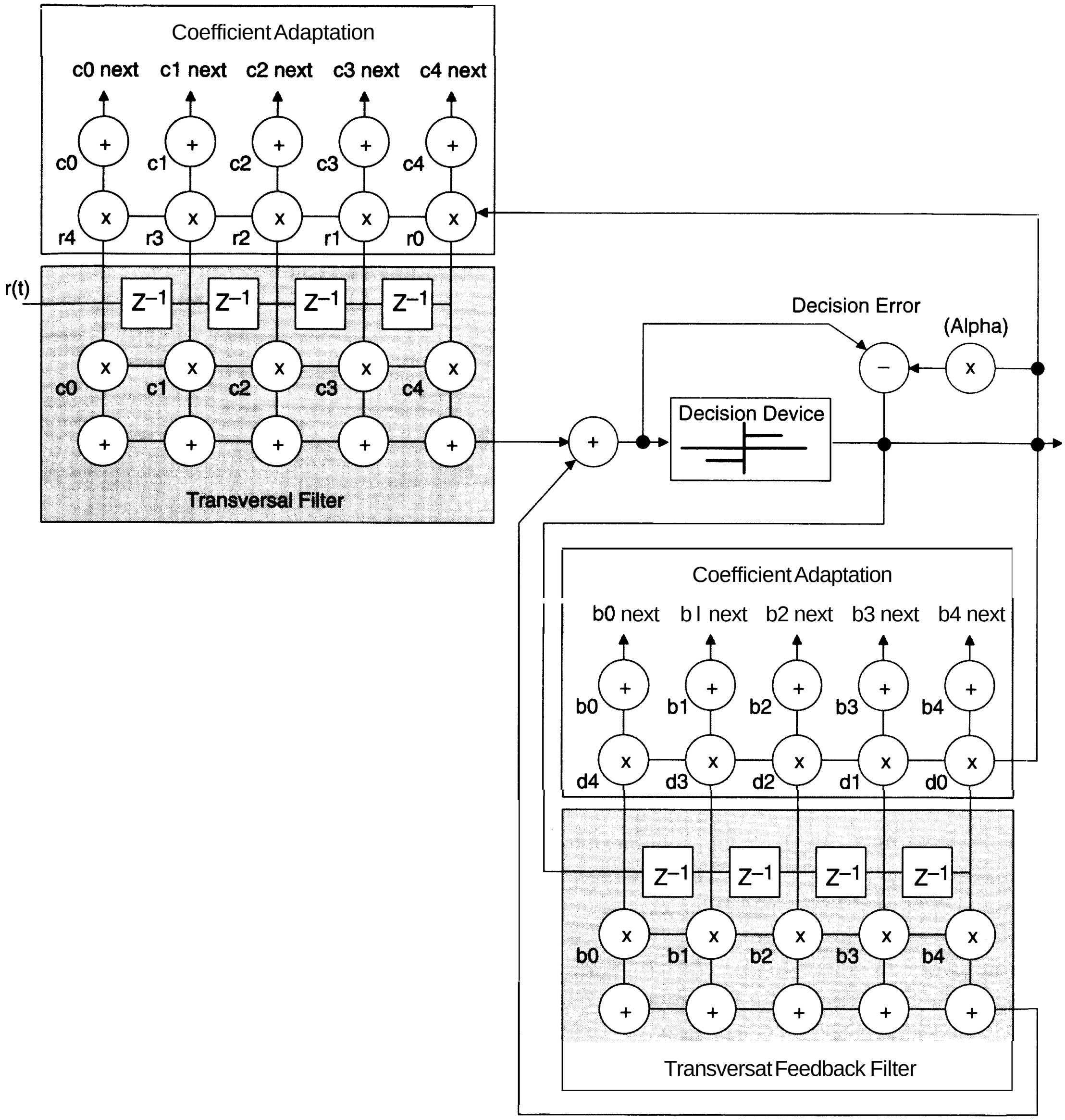


Figure 17 shows the received signal, including the effects of additive noise. Superimposed are the past four decisions (r(0), r(–1), r(–2), r(–3)) and the traces corresponding to the channel response for each pulse. Because you have a transverse filter that mimics the the system response, you can subtract the ISI contributions of the past symbols (as decided) from the next received symbol (t = 1). You can see that the decision values are sliced to ± 1, thereby tossing away noise that would have otherwise improperly influenced the compensation for the postcursor ISI. As shown in Figure 17, you can see that the coefficients of the feedback filter should converge to the right half of the channel impulse response. That is because the output value at any time t consists of the current sample times the center tap weight, plus the previous samples times the right half of the impulse response, plus the subsequent samples times the left half of the impulse response. It is the previous samples times the right half impulse response that will be subtracted by the feedback filter.

Figure 18. DFE Functional Block Diagram


Coefficient Adaptation
c0 next
c1 next
c2 next
c3 next
c4 next
c0
c1
c2
c3
c4
r4
r3
r2
r1
r0
r(t)
Z-1
Z-1
Z-1
Z-1
c0
c1
c2
c3
c4
Transversal Filter
Decision Error
(Alpha)
Decision Device
Coefficient Adaptation
b0 next
b1 next
b2 next
b3 next
b4 next
b0
b1
b2
b3
b4
d4
d3
d2
d1
d0
Z-1
Z-1
Z-1
Z-1
b0
b1
b2
b3
b4
Transversal Feedback Filter

In Figure 18, you can see that the DFE contains all of the same functional blocks as the previously described decision-directed equalizer. In addition, there is a second adaptive filter structure fed by the output of the decision device. This second filter is the feedback stage that cancels the postcursor ISI. Its inputs are the symbol decisions, and the tap weights converge through the LMS process to resemble the tail of the channel impulse response (taps beyond the center tap).

The adaptation formula for the feedback tap coefficients can be the same as for the feed forward section. For the LMS approximation [1, 4]:

$$c_n(k+1) = c_n(k) - \alpha e_k r_{k-n}, n = 0, 1 \ldots, N-1 \tag{21}$$

$$b_n(k+1) = b_n(k) - \alpha e_k d_{k-n}, n = 0, 1 \ldots, N-1 \tag{22}$$

Adaptive Equalization for Digital Cellular Telephony

The direct sequence spreading employed by CDMA (IS-95) obviates the need for a traditional equalizer. The TDMA systems (for example, GSM and IS-54), on the other hand, make great use of equalization to contend with the effects of multipath-induced fading, ISI due to channel spreading, additive received noise, and channel-induced spectral distortion, etc. Because the RF channel often exhibits spectral nulls, the linear equalizers are not optimal due to their tendency to boost noise at the null frequencies. Of the nonlinear equalizers, the DFE is currently the most practical system to implement in a consumer system. As discussed below, there are other designs that outperform the DFE in terms of convergence or noise performance, but these generally come at the expense of greatly increased system complexity. Today, most TDMA phones employ DFE running on fixed-point DSPs such as those in the TMS320C5x [6] family. For a detailed look at some representative systems, consult *A Low-Effort DSP Equalization Algorithm for Wideband Digital TDMA Mobile Radio Receivers [7]* and *Channel Equalizer for a Digital Mobile Telephone Using Narrow-Band TDMA Transmission* [8].

Advanced Adaptive Equalizer Structures

Several adaptation schemes and alternate filter structures offer better performance in some respects than those described above. Usually this performance improvement comes at the cost of increased complexity in terms of DSP CPU loading or logic gate count. For the most part, these are well understood algorithms whose system performance is still being evaluated in various applications. In any case, their treatment is beyond the scope of this tutorial in equalization concepts, and references are cited on page 174 for the interested reader.

Lattice Filter Structures

In general, the well-known lattice filter structure [9] can be substituted for the FIR sections in the DFE system. The lattice DFE has been shown to be less sensitive to roundoff errors than the transverse filter DFE, though it has comparable convergence properties. Special forms of LMS and RLS adaptation for lattice structures are summarized in *Adaptive Equalization for TDMA Digital Mobile Radio* [10]. For a detailed discussion of the implementation of lattice DFE for digital cellular radio, refer to *An Adaptive Lattice Decision Feedback Equalizer for Digital Cellular Radio* [11].

RLS Adaptation

RLS adaptation refers to the recursive least squares algorithm. The RLS algorithm can be designed to converge significantly faster than the LMS technique converges. Recall that the LMS coefficients are adjusted during each sample period by the product of the current error multiplied by the appropriate signal sample scaled by a. In the case of RLS, the adaptation is similar, but instead of scaling the adjustment by a, a value derived from the inverse of the sample autocorrelation matrix is used to scale the error/sample product. As a comparison of complexity, a 20-tap (10 forward, 10 feedback) LMS update system requires about 40 operations. A standard RLS update, on the other hand, requires on the order of 1000 operations [10]. For a more detailed look at RLS in digital cellular systems, see *A Decision Feedback Equalizer With a Frequency Offset Compensating Circuit for Digital Cellular Radio* [12] and *Bidirectional Equalization Technique for TDMA Communication Systems Over Land Mobile Radio Channels* [13].

Probabilistic Detection Algorithms

Two more advanced adaptation techniques that employ stochastic principles to minimize the probability of error are *maximum a posteriori probability* (MAP) and *maximum likelihood sequence estimation* (MLSE). These techniques require knowledge of the channel characteristics and the probability distribution of the additive noise. MAP is a symbol-by-symbol detector; whereas, the MLSE algorithm employs the Viterbi algorithm (VA) to minimize the probability of sequence error. Both approaches provide comparable performance and are still regarded as prohibitively complex for channels with a long impulse response, because complexity is exponentially related to the ISI span. For a further study, consult references [10] and [14].

Code Availability

The associated program files are available from the Texas Instruments TMS320 Bulletin Board System (BBS) at (713) 274-2323. Internet users can access the BBS via anonymous ftp at *ti.com.*

References

1. Qureshi, S.,"Adaptive Equalization", *IEEE Communications Magazine,* March 1992, pp. 9–16.

2. Peebles, P.Z., *Communication System Principles,* Addison-Wesley, 1976.

3. Samueli, H., Daneshrad, B., Joshi, R., Wong, B., and Nicholas, H., "A 64-Tap CMOS Echo Canceller/Decision Feedback Equalizer for 2B1Q HDSL Transceivers", *IEEE Journal on Selected Areas in Communications,* Vol. 9, Iss: 6 , August 1991, pp. 839–847.

4. Ziemer, R.E., and Peterson, R.L., *Introduction to Digital Communication,* Macmillan, 1992.

5. Lovrich, A. and Simar, R., "Implementation of FIR/IIR Filters with the TMS32010/TMS32020", *Digital Signal Processing Applications with the TMS320 Family,* Volume 1, Texas Instruments, 1989.

6. *TMS320C5x User's Guide,* Texas Instruments, 1993.

7. Bune, P., "A Low-Effort DSP Equalization Algorithm for Wideband Digital TDMA Mobile Radio Receivers", *International Conference on Communications Conference Record,* June 1991, pp. 763–767.

8. Svensson, L., "Channel Equalizer for a Digital Mobile Telephone Using Narrow-Band TDMA Transmission", *39th IEEE Vehicular Technology Conference,* Volume 1, 1989, pp. 155–158.

9. Oppenheim, A.V.,. and Schafer, R.W., *Discrete Time Signal Processing,* Prentice-Hall, 1989.

10. Proakis, J.G., "Adaptive Equalization for TDMA Digital Mobile Radio", *IEEE Transactionson Vehicular Technology,* Volume 40, No. 2, May 1991.

11. Narasimhan, A., Chennnakeshu, S., and Anderson, J.B., "An Adaptive Lattice Decision Feedback Equalizer for Digital Cellular Radio", *40th IEEE Vehicular Technology Conference,* 1990, pp. 662–667.

12. Shimizaki, Y., Nakai, T., Ono, S., and Kondoh, N., "A Decision Feedback Equalizer With a Frequency Offset Compensating Circuit for Digital Cellular Radio", *Vehicular Technology Society 42nd VTS Conference*, Volume 2, 1992, pp. 596–599.

13. Liu, Yow-Jong, "Bidirectional Equalization Technique for TDMA Communication Systems Over Land Mobile Radio Channels", *Proceedings,* GLOBECOM, 1991.

14. Jung P. and Baier, P.W., "VLSI Implementation of Soft Output Viterbi Equalizers for Mobile Radio Applications", *Vehicular Technology Society 42nd VTS Conference,* Volume 2, 1992, pp. 577–85.

15. *Mathcad 4.0,* MathSoft Inc., 210 Broadway, Cambridge, MA 02139.

Channel Equalization for the IS-54 Digital Cellular System With the TMS320C5x

Elliott D. Hoole
Wireless Communications Systems — Semiconductor Group
Texas Instruments Incorporated

Introduction

Transmitting digital information on a radio frequency carrier is not a new concept, but it continues to attract attention because of the need to utilize the radio spectrum more efficiently through multiple access techniques that are available only with digital links. Digital signal processors (DSPs) are required by today's communications equipment to perform complicated algorithms in a limited amount of time. One such algorithm in the digital receiver is an equalizer, which is a filter that removes the distortion caused by the communications link between the transmitting antenna and the receiving antenna.

This paper's two sections discuss ways to successfully implement an equalizer for the IS-54 standard on the fixed-point TMS320C5x. The first section gives background on the digital modulation used in the IS-54 and in the radio environment that should be taken into consideration when designing an IS-54 receiver. The second section describes the design of an equalizer for the IS-54.

Design Considerations

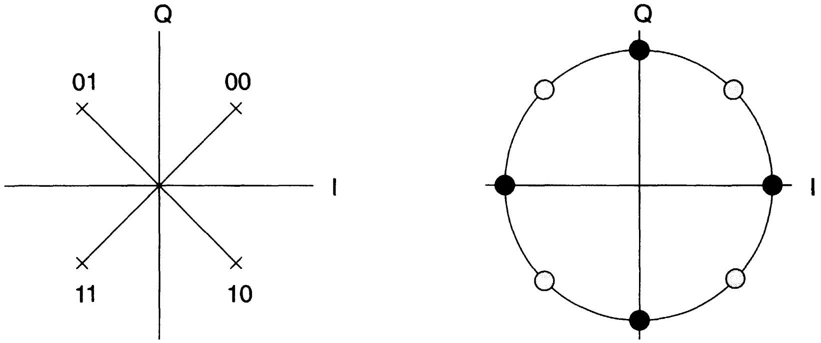
Many conditions affect a system's design. The type of digital modulation and the types of distortion and their limits influence how the receiver is structured.

Maximum-Effect Points

The IS-54 standard uses $\pi/4$ differential quaternary phase-shift keying (DQPSK) to encode a pair of bits into a phase change between two points in the complex plane. The resulting phase change is called a symbol. The points between which the change is made are known as maximum-effect points (MEPs), which are recovered by the receiver. The changes between them are investigated to decode the digital information.

Figure I shows the corresponding phase changes for the four dibits. The encoding process produces an eight-point constellation around the unit circle in the complex plane. Notice that these eight points can be divided into two subsets of four points. One subset is composed of the four points that are located on the axes of the complex plane. The other subset consists of the points that are in each of the four quadrants. For a given allowable phase change, if the starting point is an axis point, the end point will be a quadrant point. Similarly, if the starting point is a quadrant point, the end point will be an axis point. When a sequence of bits is encoded into a sequence of phase changes, the result is a sequence of points in the complex plane, which alternates between the subset of axis points and the subset of quadrant points.

Figure 1. π/4 DQPSK



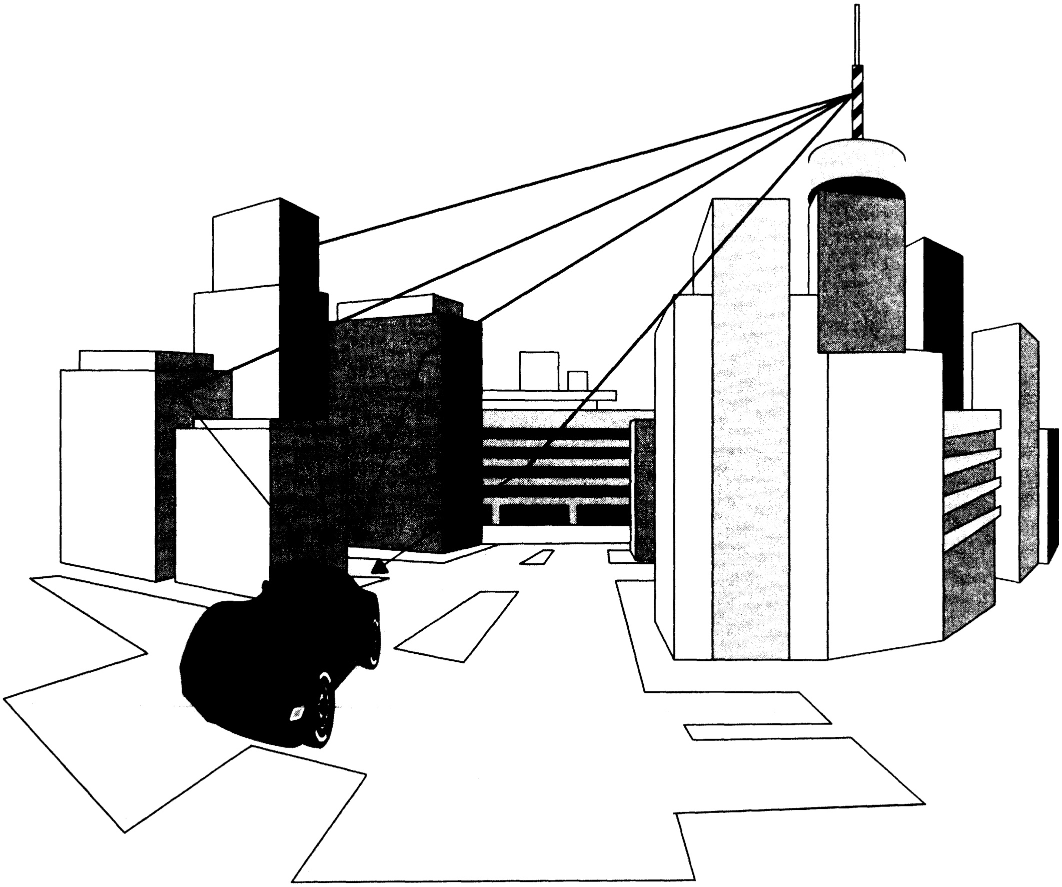
Relationship Between Phase Changes and Dibits

Constellation of Maximum-Effect Points

Multipath Interference

When a radio signal is transmitted, it can propagate along many paths to reach the receiver. At the receiver antenna, the received signal can be viewed as a complex sum of vectors with independent gains and phases. Multipath interference is the effect of multiple versions of a transmitted signal arriving at a radio receiver and combining in a way that produces distortion of the original signal. Figure 2 shows how multipath interference is produced by reflections from buildings or other objects.

Figure 2. Multipath Interference

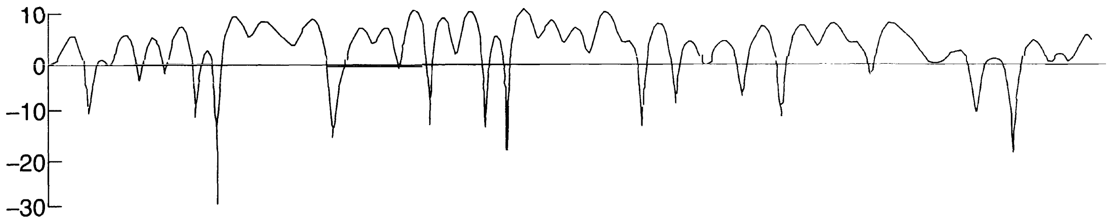


When multiple versions of the same transmitted signal arrive simultaneously, an interference pattern is formed with the various signals combining according to their amplitudes and phases. **As** the mobile receiver moves, the relationship between the amplitudes and phases of the signals from the various paths changes, causing undulations in both the composite amplitude and the composite phase. This effect is known as Rayleigh fading because the magnitude envelope has a Rayleigh probability distribution. Figure **3** shows magnitude and phase plots of fading for 0.64 seconds of a signal received by a mobile unit traveling at 25 MPH.

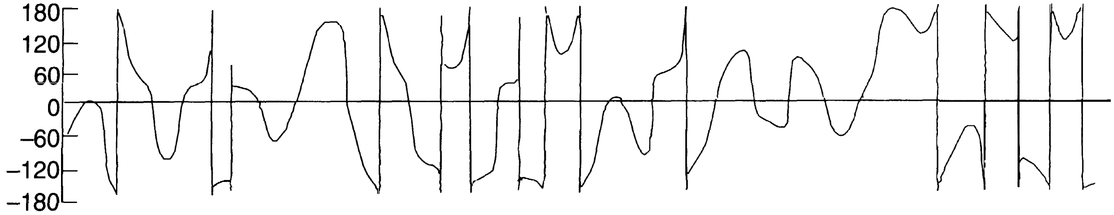
The accepted limits [2] on the amount of gain and attenuation provided by fading are +10 dB and –30 dB, respectively. Statistically, there is a 0.01% probability that the faded signal will be above 10 dB and a 99.9% probability that it will be above –30 dB.

Figure 3. Rayleigh Fading

Magnitude in dB



Phase Angle in Degrees



Intersymbol Interference

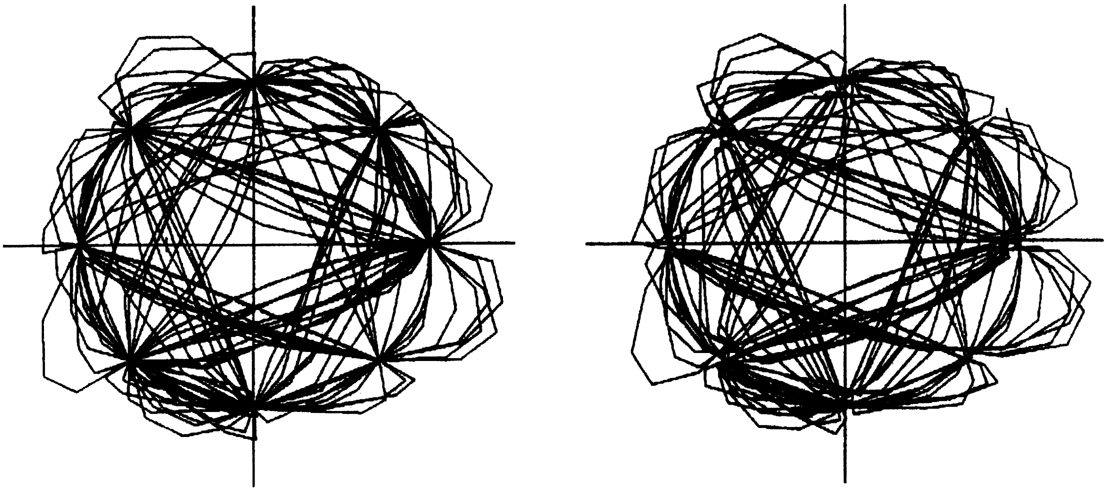
Occasionally, the arrival of some signals can be significantly delayed. This situation can result in intersymbol interference (ISI). In other words, the received signal components of previous symbols smear into later ones, thus producing distortion. In small cells, the propagation times of the different paths are nearly equal, so intersymbol interference due to multiple paths is minimal. However, in large cells, the intersymbol interference due to multiple paths can be significant. The difference in time (or symbols) between two rays' arrivals is called the *delay spread* of the channel.

Another cause of ISI is simply the bandlimited nature of the communications channel. A bandlimited channel disperses pulses going through it. This is a result of the nonideal amplitude and phase characteristics of the communications channel.

Severe ISI from one or multiple sources can render the received signal unrecoverable. For situations in which ISI is a problem, you can use an adaptive filter called an equalizer to compensate. The channel characteristics change considerably over the slot length of the IS-54 system; thus, adaptation of the equalizer coefficients is required, and the adaptive equalizer's taps must change while it is filtering the data sequence.

IS-54 defines the limit for the amount of ISI that must be compensated for by an equalizer. The IS-54 channel model was chosen to consist of a faded main ray and an independently faded delayed ray. The limit on the amount of delay is one symbol time (41.17 µs). The delayed ray can also be of equal nominal magnitude to the main ray. Figure 4 shows the effect of ISI on the π/4 DQPSK constellation. with the delayed ray three dB below the main ray.

Figure 4. Intersymbol Interference: Interferer Level –3 dBc

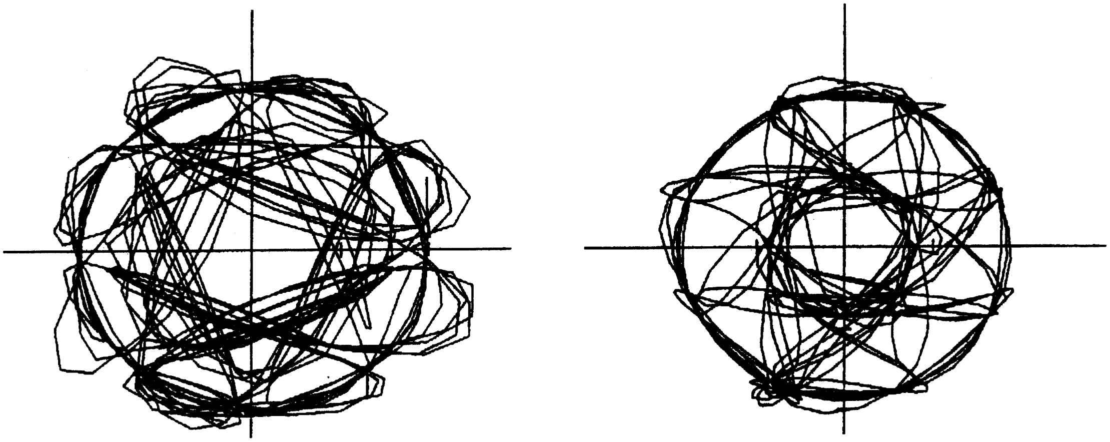


No Intersymbol Interference

1/4 Symbol Interference

1/2 Symbol Interference

1 Symbol Interference



Equalizer Design

Set Data 12 dB Below Full Scale

Figure 3 shows that Rayleigh fading can cause the magnitude of the data to be amplified by 10 dB. To prevent the sampled representations of the I and Q baseband signals from clipping, the nominal point (0 dB) should be kept at least 10 dB below full scale. For sampled signals, the amount of dynamic range represented by each bit is 6 dB. **A** convenient figure to work with for a fade margin is 12 dB, which corresponds to 2 bits in sampled form.

Software AGC

To maintain maximum resolution in the presence of fading, some mechanism is required to keep the nominal value of the data at 12 dB below full scale. An RF section may contain an automatic gain control (AGC) circuit that prevents strong signals from overloading the receiver and boosts weak signals for better recovery. However, building an AGC with a large amount of dynamic range (–30 to –115 dBm) can be an issue. An alternative to a full-dynamic range hardware AGC is a combination of hardware to attenuate the large signals and software to boost signals below the desired nominal value.

An algorithm for the software AGC can be based on signal strength measurements of the incoming data. Often, the software must perform measurements for reporting the received signal strength indication (RSSI) of the cell to the cellular system. The software AGC could use the RSSI directly or in some derivative form, but a single measurement would look like:

$$m(n) = \frac{1}{N}\sum_{i=0}^{N}\left[\left(real\{r(i)\}\right)^2 + \left(imag\{r(i)\}\right)^2\right]$$

and the RSSI value would be the weighted average of individual measurements:

$$RSSI_{sw}(n) = \lambda m(n) + (1-\lambda)\, RSSI_{sw}(n-1)$$

where λ is an exponential weighting factor that is less than 1. The final RSSI report would include the amount of attenuation provided by the hardware AGC portion, which could simply be a resistive pad switched in to prevent overloading the A/Ds. $RSSI_{sw}(n)$ represents the software scale factor. Ideally

$$RSSI_{sw}(n) = 0.0625 = -12dB$$

where 12 dB is the fade margin discussed above. So the scale factor will be

$$sc\_fact(n) = 0.0625 / RSSI_{sw}(n)$$

An assembly language example is included in the source code. See the Code *Availability* section on page 187.

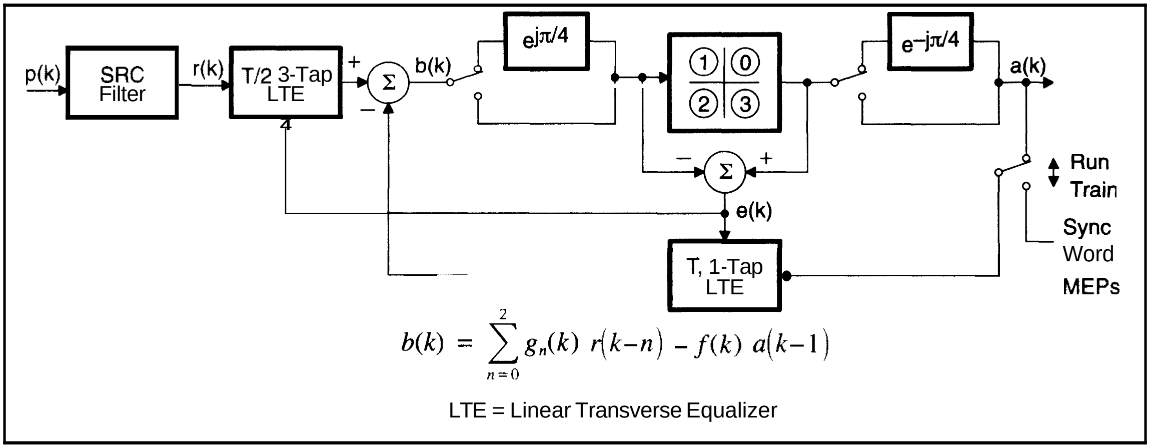
Equalization and Estimating Maximum-Effect Points

To recover the maximum-effect points in the presence of ISI, an adaptive equalizer is used in mobile stations that are compatible with IS-54. The equalizer inverts the distortion of the communications channel. Amplitude and phase distortion from fading is compensated, and components of previous symbols are removed. Figure 5 shows a block diagram of a decision-feedback equalizer (DFE). The received data is square-root raised cosine (SRC) filtered and fed into a feed-forward filter section. In this example, the feed-forward filter is a linear transversal equalizer (LTE) composed of three taps that are spaced at 1/2 symbol, or T/2. It is well-known [5] that equalizers with taps spaced in fractions of a symbol perform better than those with taps spaced at the symbol interval. Three taps spaced at T/2 provide the capability to compensate for up to one symbol of ISI, which is the upper limit specified in IS-54. The feedback filter contains a single adaptive tap. Previous decisions are filtered by the feedback filter and subtracted from the output of the feed-forward filter. This result is the estimated maximum-effect point at time k. The estimate is then fed into a data slicer, which decides which maximum-effect point is being estimated on the basis of its phase. The error vector is the difference between the estimate and the decision and drives the filter tap adaptation.

The slicer makes its decision on the basis of the phase of the estimate. Recall that there are two subsets of maximum-effect points that the encoded sequence alternates between in $\pi/4$ DQPSK. In both subsets, the points are offset by 90 degrees. In the case of the subset of quadrant points, the decision regions are trivial: determine which quadrant the estimate is in from the signs of the real and imaginary components of the estimate. In the case of the subset of axis points, the decision region boundaries are at odd multiples of π/4 in phase. However, if the estimates for the subset of axis points were rotated by π/4 in phase, the decision regions would be the quadrants of the complex plane. In Figure 5, there are two paths into and out of the slicer. The upper path is for the estimates of axis points, and the lower path is for the estimates of quadrant points.

For the equalizer to be able to track changes in the communications channel, it must first be trained to the channel's characteristics. At the beginning of every TDMA slot received by the mobile unit is a 14-symbol, 15-maximum-effect point synchronization word. The mobile unit uses this known sequence of phase changes to synchronize its receiver to the base station's transmitter. When an arbitrary starting point on the unit circle is chosen, this known sequence of phase changes can be encoded into a sequence of maximum-effect points and stored in the memory of the DSP. The stored MEPs drive the error, and the equalizer taps converge to a state in which the error is minimized; thus, the equalizer adapts to the channel's characteristics. The equalizer compensates for the phase difference, which can be completely arbitrary, between the stored MEP sequence and the received one. The taps take on whatever values are required to produce estimates of the stored MEPs.

Figure 5. Block Diagram of a Decision-Feedback Equalizer



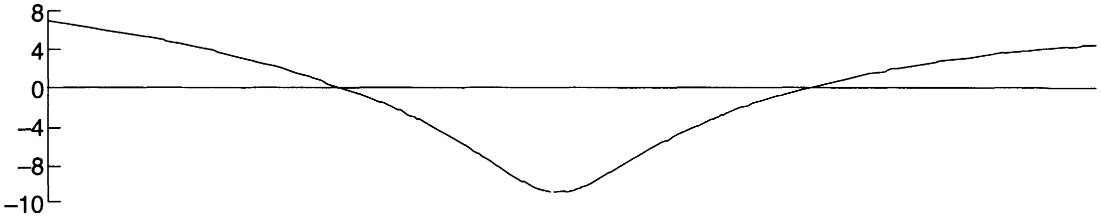
$$b(k) = \sum_{n=0}^{2} g_n(k)\, r(k-n) - f(k)\, a(k-1)$$

LTE = Linear Transverse Equalizer

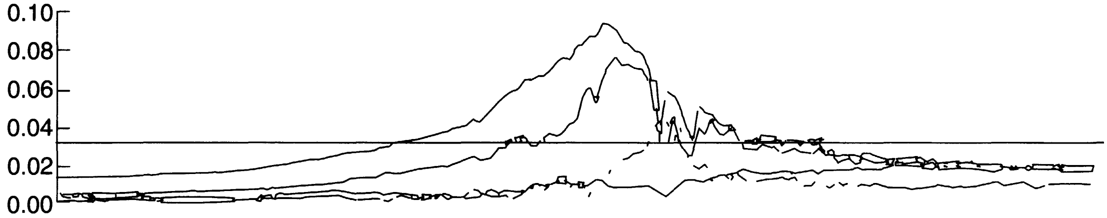
The ’C5x is a fixed-point machine; to prevent the equalizer taps from exceeding 1.O, it is necessary to scale the decision points. Since the equalizer taps adapt to the inverse of the channel, an amplification by the equalizer tap compensates for attenuation of the signal. This is illustrated in Figure 6. Since fading can attenuate the received signal by 30 dB from its nominal value, the same amount of amplification could be applied by the equalizer. The desired magnitude for I and Q is 0.25, which is 12 dB below full scale. This is also the value of the decision used in the feedback path in Figure 5. The decision points must be scaled by another 30 dB (42 dB altogether). It was verified by simulation that an attenuation of the signal by 30 dB (corresponding to a constant 30-dB fade) produced a main tap magnitude equal to 1.0 when the decision points were scaled by 42 dB.

Figure 6. Equalizer Taps Responding to a Fade

Magnitude of a Fade During a TDMA Time in a Slot (dB)



Magnitude of Equalizer Steps



Choosing an Update Algorithm

To track the changing communications channel, the adaptive equalizer uses an algorithm that updates the taps according to the error signal. Because of the requirement for tracking a fast-fading channel in a fixed-point implementation, the update algorithm should be chosen carefully.

Table 1 compares the best possible candidates. Using Table 6.8.5 in [1], a complexity comparison can be made for an equalizer with N1 = **3** feed-forward taps and N2 = 1 feedback tap. Assuming 4 **DSP** operations for a complex multiply and 40 **DSP** operations for a complex divide provides a comparative figure for the number of **DSP** operations required for each algorithm. These **DSP** operations are in the parentheses in the middle two columns of the table.

Table 1. Complexity Comparison of Update Algorithms

	Number of Complex Operations	Number of Complex Divisions	Number of Complex Multiplications	Number of DSP Operations
LMS	9	0 (0)	9 (36)	36
Fast Kalman	85	3 (120)	82 (328)	448
Conventional Kalman	58	2 (80)	56 (224)	304
Square-Root Kalman	50	4 (160)	46 (184)	344
Gradient Lattice	30	6 (240)	24 (96)	336
RLS Lattice	54	6 (240)	48 (192)	432

Of the six choices, one *must* be disqualified, and one *will* be disqualified. The LMS algorithm, although overwhelmingly simpler than the others, has insufficient convergence properties (tracking ability) for the types of channels that must be dealt with. It was included to show that the price to be paid for enough convergence is an order-of-magnitude increase in complexity. The conventional Kalman is known to have stability issues [1] and therefore should be used with caution — especially in a fixed-point implementation. For this discussion it is disqualified, as well. Of the remaining four candidates, two are clearly more complex for the desired number of taps, so the final choice is between the square-root Kalman and the gradient lattice. According to [1], the gradient lattice is a suboptimum derivative of the RLS lattice with reduced complexity and processing requirements. The square-root Kalman, however, maintains the optimal convergence properties of the conventional Kalman but uses a more stable method for updating the Kalman gain vector. It seems worthwhile to choose a slightly more complex algorithm that has significantly better convergence properties.

The list of algorithms in Table 1 is by no means comprehensive. There is a multitude of algorithms to choose from. This discussion considers only a few well-known and proven options.

Code Availability

The associated program files are available from the Texas Instruments TMS320 Bulletin Board System (BBS) at (713) 274-2323. Internet users can access the BBS via anonymous ftp at *ti.com*.

References

1. Proakis, John G., *Digital Communications*, McGraw-Hill, New York, 1989.
2. Jakes, W.C., *et al.*, Editors, *Microwave Mobile Communications*, Wiley-Interscience, New York, 1974.
3. *Cellular System: Dual-Mode Mobile Station – Base Station Compatibility Standard*, IS-54B, Telecommunications Industry Association, April 1992.
4. *TMS320C5x User's Guide*, Texas Instruments, 1993.
5. Qureshi, S. U. H., "Adaptive Equalization," *Proceedings of the IEEE*, Vol. 53, September 1985, pp. 1349–1387.

Digital Voice Echo Canceler Implementation on the TMS320C5x

Kevin McCoy
DNA Enterprises

Manswr A. Chishtie
Digital Signal Processing Applications — Semiconductor Group
Texas Instruments Incorporated

Introduction

This voice echo canceler implementation on the TMS320C5x is based on a similar implementation on the TMS320C2x [1]. This application report outlines the differences between the two implementations and highlights the specific ’C5x features that support an efficient echo canceler implementation.

This application report extends the ’C2x report with a description of the ’C5x implementation of the algorithm. It is highly recommended that you read both reports to get complete details on the theory and the algorithm used for adaptive filtering and echo cancellation. Although the basic algorithm is the same, the ’C5x implementation is considerably different from that of the ’C2x to take advantage of the ’C5x architecture. These performance improvement techniques are discussed in detail in this application report.

The hardware platform used for testing the ’C5x echo canceler software consists of a ’C5x software development system (SWDS) and an analog front end (AFE) board. The SWDS is a plug-in IBM PC AT card, which is used to debug and run ’C5x code in real time. It has all the necessary hardware hooks to allow an efficient message-passing scheme between the ’C5x and the host PC. The AFE board acts as an analog interface to the ’C5x SWDS. It is made up of two codecs, two telephone hybrid transformers, and clock generation logic for the near-end and the far-end line interfaces.

Although the software is designed to run on an SWDS-AFE platform, very little modification is required to adapt the program to a different target board[1]. The current implementation simulates the following functions in software:

- Near-end round-trip delay
- Far-end round-trip delay
- Near-end echo generation

The near-end round-trip delay directly affects the performance of the echo canceler. This is the time delay of the tail circuit (see Table 3 for details) and is simulated in software in order to analyze the echo canceler performance. The far-end round trip delay is the delay of the forward circuit. The echo generation is implemented in software.

In addition to these simulations, a message-passing scheme is supported by the ’C5x to interface to the host PC via the SWDS hardware. This allows you to monitor the echo canceler performance in real time.

These features are provided to fine-tune the software performance according to each applications requirement. They can be turned off by using software switches (see Table 1 on page 197) during assembly time.

’C5x Device Features Used in This Implementation

The ’C5x architecture is based on the industry-standard TMS320C25 architecture. The ’C5x assembly language is a superset of the TMS320C25 assembly language. However, the ’C5x has an enhanced pipelined architecture that allows it to execute instructions at 50 ns or 25 ns — more than twice the speed of the ’C2x. In addition, the ’C5x has a more powerful set of instructions that allows highly efficient algorithm implementation. Many of these enhanced features are used in this echo canceler implementation.

The rest of this section highlights various features of the ’C5x architecture that distinguish it from the ’C2x family. All code examples are taken from the echo canceler software, but the general comments are equally applicable to any DSP algorithm.

---

[1] Editor's note: This may be necessary since the ’C5x SWDS is no longer available from Texas Instruments Incorporated. An alternative development platform is the ’C5x evaluation module (EVM).

Dual Mapping of On-Chip Memory

The ’C5x has 1056 words of on-chip dual-access memory, 512 words more than the ’C25. While this type of memory is more efficient to use, it is expensive in terms of silicon real estate. Another type of on-chip memory available on ’C5x devices is single-access memory. The ’C53 and ’C51 have 3K/1K words of single-access memory, while the ’C50 has 8K words. This memory block can be mapped simultaneously in program and data spaces. This dual-mapping feature is very useful for adaptive FIR filters, such as the echo path transversal filter. The multiply/accumulate loops require FIR coefficients in the program space, but the same coefficient table is also accessed in data space to update the transversal filter coefficients. Placing this coefficient table in single-access memory and utilizing its dual-mapping feature make the transversal filter implementation more efficient. Note that the data-move operation (DMOV instruction) works on the single-access RAM (SARAM) block, as well.

Zero-Overhead Loops

The ’C5x features zero-overhead loops, as opposed to the 3-cycle overhead of the ’C25 BANZ (branch on AR not zero) loops. This makes ’C5x looped code as efficient as inline implementation. The code in Example 1 illustrates the use of block repeats in the filter taps update algorithm:

Example 1. Zero-Overhead Loops UPDATE.ASM

```
                lacl          num_a_iter_2
                samm          brcr              ;no. of iterations
                rptb          $block-end-1
                       lacc   *,16,arl          ;start of loop
                       mpya   *+,ar2
                       sach   *0-               ;end of loop
     $block-end:
```

In the 'C25 implementation, the same algorithm was coded inline.

Dynamic Addressing of Coefficient Tables

The multiply/accumulate instruction (MAC) on ’C25/’C5x devices fetches input samples of an FIR filter from data memory and takes the filter coefficients from the program memory. This achieves single-cycle, multiply/accumulate operation by simultaneously fetching two operands from memory. Most ’C25/’C5x FIR computations are carried out this way. On the ’C25, the coefficient table address can be specified only in the direct addressing mode. This is adequate for most applications, except where the coefficient table address is determined in runtime. For such cases, the ’C5x provides a register-indirect mode of addressing on multiply/accumulate operations.

Example 2. Echo Estimation Routine FIR.ASM

```
                lacl          last-a            ;update coefficient
                samm          bmar              ;  table address
                lacc          one,14
                zpr                             ;clear preg
                rpt           num_a_1           ;repeat
                       madd   *_                ;multiply/accumulate
                apac                            ;last product
                sach          est_echo,1        ;save echo estimate
```

This feature is used in the echo estimation routine, as shown in Example 2. The block-move-address register (BMAR), a dedicated CPU register, points to the location of the coefficient table in program memory. This feature is useful when code reuse is a consideration. For the code shown in Example 2, it is particularly important because the length and the location of the transversal filter coefficients are determined in runtime.

Use of Nested Loops

Complex applications like voice echo cancellation often need nested loops. For instance, the block update algorithm for echo filter taps requires two nested loops: an inner loop to compute a time-averaged correlation error for each coefficient in the block and an outer loop to update the coefficient. This can easily be accomplished on the ’C5x by nesting a single-instruction repeat (RPT) inside the block-repeat (RPTB) loop.

Example 3. Coefficient Update Routine TAPINC.ASM

```
          lt          cun0   ;
          lar         ar2, #inc0
          rptb        $calc_INCs–1          :outer  loop
             lacc     one, 15
             mpy      *+
             rpt      #14                   ;inner loop
                mac   pun0+1,*+             ;compute error
             mar      *,ar2
             lta      cun0
             sach     *+,0,arl              ;save coeff update
$calc_INCs:
```

When a single-instruction repeat (RPT) loop cannot be used, block-repeat loops can be nested with delayed-branch loops such as branch-on-AR-not-zero-delayed (BANZD). Up to eight such BANZD loops can be nested, each using an auxiliary register as the loop counter. In ’C25 implementation, the same algorithm is coded in-line.

Maxima/Minima Search

The ’C5x features special instructions to efficiently find minimum (or maximum) value in a data array. Each element in the array can be 32 or fewer bits wide. A signed comparison is made between the accumulator and the accumulator buffer, and the smaller (or greater) of the two values updates the accumulator buffer. This feature is advantageous in the near-end speech detection algorithm.

Example 4. Near-End Speech Detection Routine NESPDET.ASM

```
          lacl        num_m_1   ;
          samm        brcr                ;repeat count
          zap
          sacb                            ;initialize accb for search
          rptb        $max
             lacc     *–, 0,ar2           ;get partial maxima M(k)’s
             sacl     *–,0,arl
             crgt                         ;save largest M(k) in accb
$max:
          sacl        max_m               ;largest M(k) –> max_m
```

The code loop shown in Example 4 performs two functions:

- It finds the largest far-end speech sample (or its power estimate) from a set of the num_m most recent samples.
- It implements a time window spanning the echo path delay range.

On the TMS320C2x, the same algorithm must be implemented with conditional branches. The built-in 'C5x support for search algorithms generates faster and more elegant code.

Circular Buffers

Another 'C5x advantage over the 'C2x is its support for circular addressing. Two independent circular buffers of any size are supported by the 'C5x address generation unit. They can be used to implement FIFO buffers and queues. In this echo canceler application, the two circular buffers are used to hold far-end and near-end receive samples and implement variable delay for near-to-far and far-to-near signal paths.

Another important use of circular addressing is in FIR filter implementations. The conventional way of performing FIR computation on 'C2x/'C5x devices is via a multiply/accumulate with data-move (MACD) operation. In the case of a 'C5x, circular addressing can replace a data-move operation to update filter taps. This is a faster implementation if the filter taps reside in the on-chip single-access memory or the external data memory. The echo simulation filter employs this technique, as shown in Example 5.

Example 5. Echo Simulation Filter EFILT.ASM

```
mar      *,ar5;
lar      ar5,efilt_ptr    ;get echo filter taps address
zap
rpt      #(filt_len–1)    ;multiply/accumulate
   mac   echo_filt_end,*+ ;  with circular addressing
apac                      ;add final product
add      one,14           ;round output
sach     sim_echo_out,1   ;save as Q15 result
```

Delayed Branches and Conditional Execution

The 'C2x has a three-deep instruction pipeline. This allows it to perform more operations in parallel by overlapping various phases of instructions. The 'C5x features a four-deep instruction pipeline to attain even higher performance. Since deeper pipelines take more cycles to flush, the 'C5x supports special types of branches and calls to avoid this overhead. Normal 'C5x branches take four machine cycles, while a similar instruction on a 'C2x takes only three cycles. However, all 'C5x instructions that cause a pipeline flush support a delayed option that reduces the overhead to only two machine cycles. Moreover, in the special case in which only one or two instructions are skipped over, you can use an even faster instruction, XC (conditional execute), which takes only one machine cycle.

The code shown in Example 6 illustrates the use of delayed branches and conditional execute instructions.

Example 6. Use of Delayed Branches NESPDET.ASM

```
bd         $chk_hang          ;delayed branch
    sacl   max_m
    lacc   absy0f             ;branch executes here
sub        max_m
lar        arl,last_m_1
xc         2,gt               ;if acc<=0 then skip next two
    lacc   absy0f             ;   instructions
    sacl   max_m
lacc       num_m_1
samm       brcr
```

Barrel Shifters

Both the ’C2x and ’C5x DSP families support a 16-bit input prescalar and an 8-bit output postscalar in hardware. This is necessary for efficient fractional arithmetic and bit manipulation. In addition to these barrel-shifters at the input and output paths, the ’C5x family also features a 16-bit right barrel shifter on the accumulator. This complements left barrel shifting provided by the input prescalar. The code in Example 7 illustrates the use of barrel shifters.

Example 7. Code Excerpt From MULAW.ASM

```
lact   temp-B2    ;Shift left biased linear into ACC
bsar   16         ;Shift right ACC by 16
add    #0E0h
sub    treg1,4    ;Shift left by 4 and subtract
```

The lact instruction uses the left barrel shifter to transfer data to the accumulator, and the input shift is determined by the tregl register. The following instruction, bsar, performs a 16-bit right barrel shift on the accumulator contents.

Memory-Mapped Registers

Both the ’C2x and the ’C5x have accumulator-based internal architecture. In ’C2x devices, all arithmetic operations are performed on the accumulator. There is no data path between the accumulator and other CPU registers, including the auxiliary register set. Therefore, a temporary data memory location must be used to transfer data between the arithmetic logic unit (ALU) and the address generation unit (AGU).

The ’C5x architecture is considerably enhanced; it provides a direct data path between the accumulator and the rest of the CPU registers by mapping them into local data memory. It also supports direct memory-to-register data transfer on all its internal registers. The code in Example 8 illustrates the use of ’C5x memory-mapped registers.

Example 8. Taps Update Routine UPDATE. ASM

```
update taps:
   splk      #16,indx      ;init. index register
   lar       AR1,#INCO     ;init. aux register 1
   lacc      ADA0
   sub       H
   sacl      ar2           ;init. aux register 2
   lacc      beta-gain     ;get variable beta-gain factor
   samm      tregl         ;init. temp register 1
   lacl      num_a_2
   samm      brcr          ;init. repeat count
   lact      IABSY
   samm      treg0         ;init. temp register 2
   mpy       *+,ar2
   rptb      $block-end-1
      lacc   *,16,arl
      mpya   *+,ar2
      sach   *0-
$block-end
```

Parallel Logic Unit

The ’C5x bit manipulation unit runs independently from its arithmetic logic unit. It allows logical operations on any on-chip or off-chip memory location (including memory-mapped registers) without modifying the accumulator (ACC) or accumulator buffer (ACCB). This feature, in conjunction with the memory mapping of the CPU registers, provides ’C5x programmers more flexibility to modify auxiliary registers to implement software queues and FIFOs. Additionally, the read-modify-write operation performed by the parallel logic unit (PLU) instructions may also be used for semaphore update. The section of code in Example 9 is taken from the echo canceler program. It services the serial port receive interrupt by reading the received data, transmitting new data and setting appropriate flags to communicate with the background program. Notice in particular the use of PLU instructions for setting software flags.

Example 9. Serial Port ISR ECHOISR.ASM

```
rint_isr:
   ldp    #DRR_data
   smmr   drr,#DRR_data     ;get serial receive data
   lmmr   drr,#DXR_data     ;send serial transmit data
   opl    #RXDATA,sp_flag   ;mark serial data received
   apl    #TXDATA,sp_flag   ;mark serial port data sent
   opl    #ERINT,intr_flag  ;mark rint in intr_flag
   reti
```

Code and Data Requirement

The echo canceler software implementation gives you maximum control over its performance and behavior. Various system parameters, such as the echo filter length, echo cancellation enableldisable mode, and filter adaption enableldisable mode, are represented by memory variables rather than by hard-coding in software. This lets you either:

- Modify these parameters in realtime by the use of supervisory software, as illustrated in the SWDS demo program, or
- Set up these parameters in the initialization stage.

Table 1 lists these user-defined system parameters along with their default values. To modify the default value parameters, edit the echoequ.inc file.

Table 1. User-Defined System Parameters

Number	Variable Name	Description	Type	Default [range]
1	pd_wait	Program/data wait states	const	0h
2	echo–taps	Transversal echo filter taps	const	512 [16-512]
3	sim_echo	Simulated echo disable/enable	const	1 [0/1]
4	host–comm	Host PC communications disable/enable	const	1 [0/1]
5	control_flags[†]	Bit 0: echo cancellation disable/enable Bit 1: residual suppression disable/enable Bit 2: coeff adaptation disable/enable	variable	1 [0/1] 1 [0/1] 1 [0/1]

[†] The control–flags variable is active only when host–comm is set to 1. Edit the echoinit.asm file to modify this memory variable.

Table 2 indicates the processor loading and the code size of each software module for a 512-tap implementation. It also indicates where each module is located in program memory. Most of the time-critical subroutines are located in the on-chip single-access random-access memory (SARAM). The auxiliary functions, such as the host PC mailbox, are executed from external memory.

Table 2. Program Module Requirements

Number	Module Name	Description	CPU Cycles†	Code Size	Code Location*
1	ECHO.ASM	Main module — variable declarations.	–	2	ROM
2	ECHOINIT.ASM	Initialization module.	–	218	ROM
3	ECHOISR.ASM	Interrupt services routines.	17	56	ROM
4	CYCLE.ASM	Get new samples. Convert μ-law to linear. Poll host PC mailbox.	67	71	SARAM
5	EFILT.ASM	AR for the echo simulation. Update delay buffers.	50	21	SARAM
6	FIR.ASM	Estimate echo. Compute error.	546	21	SARAM
7	RESID.ASM	Residual error suppressor.	17	16	SARAM
8	MULAW.ASM	Linear-to-PCM conversion.	41	28	SARAM
9	PCALC.ASM	Power estimate of $y(n)$ and $o(n)$.	39	19	SARAM
10	NESPDET.ASM	Near-end speech detection.	47	83	SARAM
11	ONORM.ASM	Output normalization for coefficient update.	55	32	ROM
12	TAPINC.ASM	Tap increment.	791	32	ROM
13	UPDATE.ASM	FIR filter tap update.	153	27	ROM
14	UTIL.ASM	Process host PC commands. Write monitored variables.	–	233	ROM
15	MAILBOX.ASM	Host PC mailbox.	–	41	ROM
			Total cycles for 512-tap filter = 1825	Total code size = 900 words	

† Only for the modules that are in the main cycle. Cycle count given for 512 taps transversal echo filter.

‡ ROM = 'C51 on-chip, read-only memory or external memory.
SARAM = 'C51 on-chip, single-access RAM.

Data Allocation

The 'C51 has 1056 words of dual-access and 1024 words of single-access on-chip memory. It also has 8K words of on-chip, read-only memory. The on-chip data memory is allocated to various modules of the echo canceler software according to their specific requirements. Table 3 lists the size and the location of various data variables for a 512-tap implementation.

The coefficients of the echo transversal filter are placed in the on-chip, single-access memory because of its dual-mapping capability. Note that these coefficients are accessed in both program and data spaces by two different modules.

The 1024 words of dual-access memory are used for data storage. Reference samples of the far-end talker reside in this memory block. This makes efficient use of multiply-accumulate-with-data-move-type operations.

To simulate delay paths between near-end and far-end speakers, two long buffers of 2K words each are maintained in external data memory. Another buffer that holds host PC messages resides in external memory. Since all three buffers are in noncritical paths and would eventually be deleted from the final implementation, they are placed in external memory.

Table 3. 512-Tap Implementation Data Variables

On-Chip Single-Access Memory: 528 Words	
16 words	Normalized outputs Un0 – Un15
512 words	Transversal echo filter coefficients A0 – A15
On-Chip Dual-Access Memory: 655 Words	
62 words	System variables
33 words	Local maxima M(k) for near-end speech detection
32 words	Coefficient increment INC(k)
528 words	Reference samples Y(k)
External Data Memory	
2304 words	Near-to-far sample delay buffer (optional)
2304 words	Far-to-near sample delay buffer (optional)
2048 words	Message buffer for PC communications (optional)

Code Benchmarks

The two most computationally intensive routines of this echo canceler application are:

- The transversal echo filter routine FIR.ASM, and
- The mean square error (MSE) computation routine TAPINC. ASM.

The computational requirement for these two routines depends on the length of the echo transversal filter.

Table 4 shows the relationship between the processor loading and the length of the transversal filter. For a 512-tap filter, the ’C5x takes only 92 microseconds to process each sample. With an input sampling rate of 128 microseconds, this leaves the processor with ample time for system overhead. In fact, a 50-ns ’C5x processor can implement about 750 echo filter taps within a 128-microsecond sampling period. In other words, one 50-ns ’C5x DSP can handle 96 ms of the tail-end circuit delay.

Table 4 shows code benchmarks for a hardware platform that consists of the ’C51 software development system (SWDS) with an analog front end (AFE) board, a zero-wait-state external data/program memory, a 50-ns instruction cycle rate, a 128-ys input sampling period, and PC communication disabled.

Table 4. Code Benchmarks

Number	Echo Filter Taps	Time Required to Process One Sample
1	32	26.0 µs
2	48	28.1 µs
3	64	30.0 µs
4	80	32.4 µs
5	96	34.6 µs
6	128	38.9 µs
7	256	56.7 µs
8	512	91.6 µs

Echo Canceler Demonstration on a ’C5x SWDS

The primary hardware platform for testing the ’C5x echo canceler software (for code benchmarks) was a ’C5x SWDS. The AFE board communicates with the ’C5x DSP via its serial port and has codecs and hybrid transformers for near-end and far-end telephone interfaces. An AFE board schematic is shown in the appendix of this report.

You can run the demonstration software on any ’C5x SWDS board by downloading the echo.out file to the board and running the echodemo.exe file on the host PC. To do this, type the following two commands at the DOS prompt:

```
c51load echo.out
```

```
echodemo.exe
```

You can control the various system parameters — such as tail-circuit delay, transversal filter taps, echo cancellation mode, and adaptation mode — in real time by running the echodemo.exe program.

Conclusion

This implementation of a single-channel voice echo canceler on a TMS320C51 highlights the powerful and versatile architecture of that DSP. This particular algorithm was first coded on a TMS32020. Coding the same algorithm on a TMS320C51 shows that the resulting performance improvement is not merely due to the faster instruction rate on the 'C5x. Performance is improved by more than a factor of two when enhanced 'C5x architecture is fully utilized. The 'C5x features used in this implementation are discussed in detail. The processor loading and the code and data size of each software module are listed. Several auxiliary functions that are used for testing and evaluation purposes are discussed. The details of a demonstration package that consists of a 'C51 SWDS, an analog front-end board, a 'C5x DSP, and PC software are given.

Acknowledgements

Texas Instruments acknowledges the efforts of the DNA Enterprises' project team: Kevin McCoy, Mark Sissom, and Paul Kniffen.

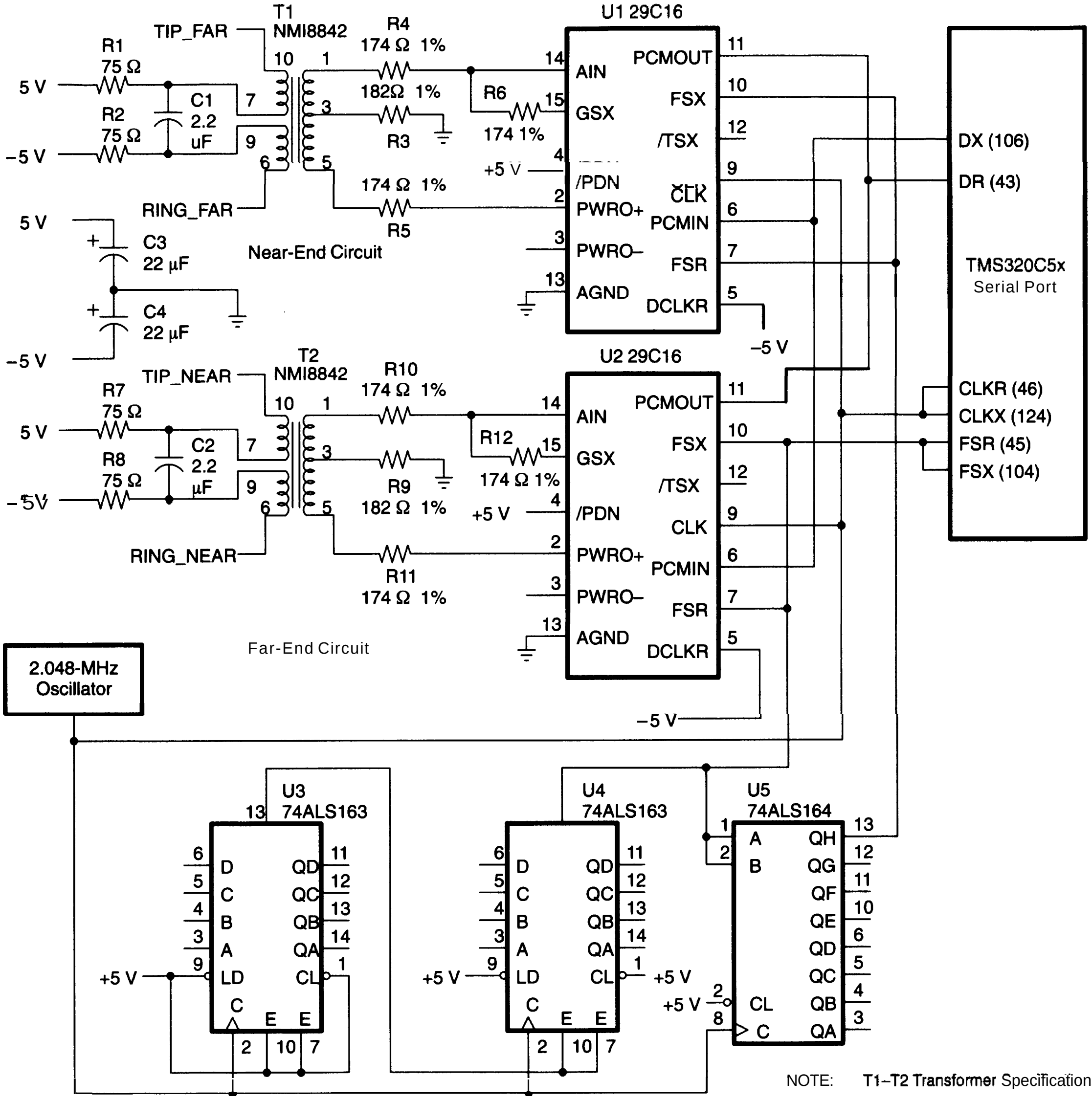
Code Availability

The associated program files are available from the Texas Instruments TMS320 Bulletin Board System (BBS) at (713) 274-2323. Internet users can access the BBS via anonymous ftp at *ti.com*.

References

1. "Digital Voice Echo Canceler With a TMS32020", *Digital Signal Processing Applications,* Volume 1, Texas Instruments, 1986, pp. 415–454.
2. *TMS320C5x User's Guide,* Texas Instruments, 1991.
3. *TMS320C5x Software Development System Technical Reference,* Texas Instruments, 1990.
4. *TMS320C5x C Source Debugger User's Guide,* Texas Instruments, 1991.
5. *TMS320 Fixed-Point DSP Assembly Language Tools User's Guide,* Texas Instruments, 1990.

Appendix: Schematic of the Dual-Telephone Interface for the TMS320C51 SWDS



T1 NMI8842
TIP_FAR
RING_FAR
R1 75 Ω
R2 75 Ω
C1 2.2 uF
5 V
−5 V
R4 174 Ω 1%
R3 182Ω 1%
R5 174 Ω 1%
R6 174 1%
C3 22 µF
C4 22 µF
Near-End Circuit
U1 29C16
AIN
GSX
/PDN
PWRO+
PWRO−
AGND
PCMOUT
FSX
/TSX
CLK
PCMIN
FSR
DCLKR
+5 V
T2 NMI8842
TIP_NEAR
RING_NEAR
R7 75 Ω
R8 75 Ω
C2 2.2 µF
R10 174 Ω 1%
R9 182 Ω 1%
R11 174 Ω 1%
R12 174 Ω 1%
U2 29C16
Far-End Circuit
TMS320C5x Serial Port
DX (106)
DR (43)
CLKR (46)
CLKX (124)
FSR (45)
FSX (104)
2.048-MHz Oscillator
U3 74ALS163
U4 74ALS163
U5 74ALS164
D C B A LD C E E
QD QC QB QA CL
A B CL C
QH QG QF QE QD QC QB QA

Codec Frame Generator

DNA Enterprises. Inc.
Dual-Telephone Interface for the TMS320C51 SWDS

NOTE: T1–T2 Transformer Specifications

1. Dual primary, center-tapped secondary
2. Primary inductance = 1.1 H @ 1 kHz. 65 mA
3. Saturation current > 70 mA
4. 1 : 1 : 1 : 1 turns ratio
5. Average winding resistance = 62.5 Ω
6. Primary winding resistances should match within
7. Wind 1/2 secondary first; tape, then wind primary bifilar; tape, then wind rest of secondary.
8. Coefficient of coupling (K) >= 0.998
9. Pri-to-pri and pri-to-sec Hipot = 500 V (This is a single-coil, hybrid transformer.)

Part VII
Speech and Character Recognition Algorithms

DSP-Based Handprinted Character Recognition

Alan Josephson
Information Technology Group
Texas Instruments Incorporated

Introduction

The market for pen-based computers is growing. Pen-based computers include notebook-sized tablets, pocket organizers, and handheld computers (HHC). Most pen-based computers offer handprinted character recognition (HCR), and some are beginning to offer cursive handwriting recognition. Most implementations of pen-based computers with HCR suffer from slow response times and inaccurate recognition. The HCR algorithm is typically implemented on the CPU of the pen-based computer.

This report describes how to implement HCR for real-time applications. Such applications are found in a variety of industries, including financial trading, healthcare, and transportation.

Users of handheld computers require fast response and high accuracy recognition rates. To meet these requirements, the execution of HCR tasks in a handheld computer is shared among a pen-input processor, a TMS320C5x digital signal processor (DSP) residing on a Type II PCMCIA-compatible card, and the main CPU. The input processor digitizes and filters handprinted input written on a resistive pad. The DSP manages pen-stroke and character libraries. The DSP performs character-based matching by using these libraries and digitized strokes provided in real-time by the input processor. The DSP provides the character string with the best match to the input data, along with a set of possible alternatives, to the main CPU. In addition, the DSP can handle high-level verification of character recognition, such as constraining the matching to a dictionary of valid character string inputs. The main CPU handles inking of data to the LCD, establishing a recognition context, and communicating pen-stroke data to the DSP.

The prototype HHC platform described in this paper was developed in conjunction with Commodity Exchange, Inc. of New York (COMEX) as a means for traders and brokers to input trades to the exchange and electronically receive price and trade order information. The HHC platform system can also be used in other industries that require communicating with pen-input computers and wireless LANs.

The current implementation of HCR is integral to the HHC and runs on an MC68000 processor, along with other system and application software. This paper shows how a PCMCIA card containing static memory and a general-purpose DSP can be used to implement HCR in a multiprocessor setting. A Type II PCMCIA-compatible card containing a TMS320C5x DSP and 256K bytes of SRAM (referred to as a ***DSP/memory card*** or ***DSP card)*** is under development by the Texas Instruments Semiconductor Division [1, 2, 3]. The architectural and functional aspects of this card that are relevant to the implementation of HCR are discussed.

Architecture

The prototype HHC is shown in Figure 1. The processors and I/O devices for implementing HCR are described in the following paragraphs.

Figure 1. Prototype HHC Platform With Pen Input

Host Processor

The MC68302 is the main processor used in the HHC. The 68302 is an integrated multiprotocol processor consisting of an MC68000 core, a system integration block (SIB), and a communications processor (CP). The CP is connected to the core through the SIB, not the data bus, and can operate independently of the core. This feature allows multiple tasks to be implemented in hardware, providing increased system speed and better power management. The 68302 has the ability to put the core and the CP to sleep independently, allowing large power savings. The return to a normal operating state is very quick and undetectable by the user. The MC68000 core is referred to as the *host processor.*

Input Processor

Any standard ball-point pen, pencil, or stylus can be used to enter handprinted input and signatures onto a resistive, opaque X-Y digitizing pad, located between the LCD and the elastomeric keyboard. An Intel 8051-like Signetics S87C552 microprocessor performs input preprocessing and provides low power modes of operation used in power management. This processor is awakened through hardware whenever a key is pressed, the discrete matrix touchscreen is touched, the digitizing pad is touched, or data is received over its serial line from the host processor. The firmware drives the A/D circuitry, which biases the digitizing pad and gets X-Y and touch-detect readings. Higher level software averages the X-Y points and reports the filtered ***strokes*** (a pen down, followed by a stream of points, followed by a pen up) over the serial line to the host processor for recognition or for signature compression. As information is sent to the main processor for recognition and/or storage, it is presented, or ***inked,*** on the LCD to provide feedback to the user. Key presses and touchscreen touches are reported similarly.

DSP/Memory Card

The DSP/memory card can be used either as standard memory or as a multifunction peripheral device. The HCR (and other) DSP algorithms can be loaded into the card by a host processor in the same way it writes to any PCMCIA memory. Once the program is loaded, the host can command the DSP to execute the algorithm as a CP. Among the key features of the DSP/memory card used in this implementation of HCR are on-board logic to arbitrate the memory bus between the DSP and the host, direct interrupt control and handshake between the host and DSP, and host control of DSP operating speeds for power management.

System-Level Software

A real-time operating system (RTOS) with facilities for multitasking and interprocess communications runs on the host. The application program interface (API) implements the application programmers' view of the operating system. Included in this interface are functions to accept input from the keyboard, the touch screen, the digitizing pad, or the communications system. Also included are functions to output data to the liquid crystal display (LCD), to handle communications, to access RAM, and to access the PCMCIA-compatible card.

The input subsystem routines allow application programmers to manage the input queue that records user inputs from the touch screen, keyboard, and HCR subsystem. The modularity of the HCR subsystem makes porting it to the DSP/memory card straightforward. Initialization routines for the HCR subsystem are available from within applications. Communication of recognition parameters between applications and the HCR subsystem occurs through APIs that manipulate the recognition context, which is composed of inking parameters, active model databases, active gesture sets, and active constraint dictionaries. The HCR subsystem is activated asynchronously when serial data from the digitization subsystem is received. After processing the digitizer data, the HCR subsystem sends a notification to the application (similar to those sent for keyboard and touch screen events) that the recognition results are awaiting processing. The application is then responsible for invoking final translation and constraint of the recognition result through function calls to the HCR subsystem running on the DSP/memory card.

To conserve power when the HHC is not in use, it can be placed, under software control, into one of two sleep modes: shallow sleep or deep sleep. In the shallow sleep mode, the processor is active, but some of the nonessential services have been turned off. The application is unaware of the shallow sleep mode that is managed by the HHC system software. In the deep sleep mode, almost all services are turned off, the internal status of the processor is saved, and the HHC uses the minimum power required to wake up automatically when an interrupt occurs from the keyboard, touch screen, digitizing pad, communications system, or DSP.

HCR Subsystem Description

The HCR algorithm embodies an operator-trainable stroke-based approach. The operator can enter models for individual characters (alphabetic and numeric) during interactive training sessions. These models make up *model databases* that are employed by the recognition software as a basis for translating the operator's handwritten input from within applications. As the operator writes on the digitizing pad, strokes are digitized into a set of discrete points that are used as input to the recognition subsystem. Strokes are thinned so that not all points are retained; strokes are normalized with respect to a common scaling factor. After normalization, the stroke is compared to all strokes in the currently active stroke database, and a *degree of match* or *penalty* is determined for each. Once this process has been repeated for all strokes in the current input, the sequence of strokes is *parsed* into a set of potential symbol matches, utilizing the models in the currently active model database as references. As possible recognition results are computed, they are stored — along with their associated penalties — to be reported to the application. The HCR approach does not require the operator to write within a specified grid. Spatial separation between characters is not essential for recognition, and the user may overlap and overwrite characters.

Application-generated contexts allow the recognition software to disambiguate between otherwise indistinguishable models from different databases (for example, numeric 0, 1, and 5 from alphabetic O, I, and S). Using context-sensitive dictionaries to constrain fields *before* the recognized result is reported to the user causes the perceived recognition accuracy to be higher than it would be if only character-based recognition were being used, and it causes it to be *much* higher when alphabetic and numeric contexts are available. Additionally, context provides a means for increasing recognition speed, because the database of models to be searched is smaller.

HCR Subsystem Implementation

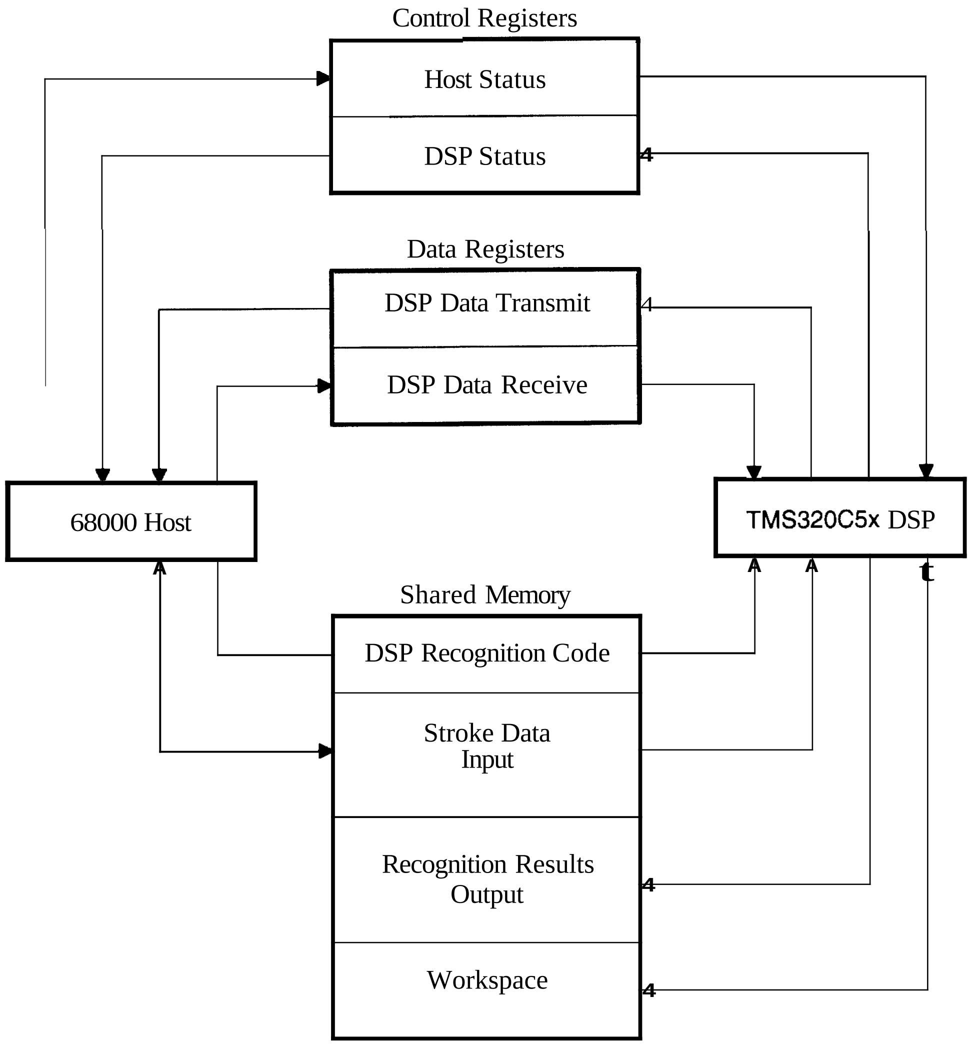
The incremental nature of the HCR algorithm makes it a natural candidate for exploiting the parallelism offered by the DSP CP. The host processor is responsible for receiving strokes from the input processor, optionally inking the digitized points to the LCD, performing high-level filtering of the digitized strokes, and communicating the strokes to the DSP for processing. Additionally, application-level software running on the host processor communicates contextual recognition parameters and requests for recognition results to the DSP. The DSP incrementally processes the strokes as they arrive from the host by forming partial recognition results. Also, the DSP — in response to the host's requests — sets recognition parameters and generates final translations of pen input on the basis of the recognition parameters.

Memory Organization of the DSP Card

Figure 2 shows the DSP card's memory map as used by the HCR subsystem. The shared memory is partitioned into code memory, stroke data (input), recognition results (output), and a workspace for the HCR subsystem running on the DSP. The arrows in the diagram indicate read/write privileges for the host processor and DSP. This partition of the shared memory is a design choice, based on the read/write privileges. If some segment of the card memory requires write privileges for both the host processor and DSP, card logic in an onboard FPGA/ASIC handles this memory contention by giving precedence to the host.

The DSP code is loaded into the shared card memory by the host during initialization of the HHC. The host then switches the DSP card from *standard mode* (in which the DSP is inactive) to *smart mode*, at which time the DSP begins execution from the code segment. The DSP initializes variables in its workspace and enters a processing loop, awaiting commands from the host processor.

Figure 2. DSP Card Memory Organization for HCR


Control Registers
Host Status
DSP Status
Data Registers
DSP Data Transmit
DSP Data Receive
68000 Host
TMS320C5x DSP
Shared Memory
DSP Recognition Code
Stroke Data Input
Recognition Results Output
Workspace

Interprocessor Communication

The host processor and DSP communicate through dedicated 16-bit data, status, and control registers in the FPGA/ASIC on the DSP/memory card (see Figure 2). Read and write access to the DSP data transmit and DSP data receive registers is enforced by the onboard logic.

Host-to-DSP Communication

The host communicates to the DSP by writing commands to the DSP data receive register. The host has only write access to this register; any host read from this register causes invalid data to be read. The DSP has only read access to the DSP data receive register; any DSP write to this register is ignored. A host write to this register generates an interrupt to the DSP. Similarly, a DSP read from this register generates an interrupt to the host. The host status register is accessed by the DSP to determine the status of host communication registers.

The DSP recognition code is a stand-alone application that does not run under an operating system. The top-level DSP recognition code consists of a command processing loop that interprets commands written to the DSP data receive register. Thus, when a stroke becomes available to the host from the input processor, it is communicated to the DSP by first copying it to the card's shared memory. The host then issues a command to the DSP to process the stroke. Commands to initialize and reset HCR parameters are communicated similarly. Whenever the DSP has no pending commands from the host, it enters its lowest power mode [4]. Writes to the DSP data receive register awaken the DSP for further processing.

DSP-to-Host Communication

The DSP communicates to the host by writing commands to the host data receive register. The DSP has only write access to this register; any DSP read from this register causes invalid data to be read. The host has only read access to the host data receive register; any host write to this register is ignored. A DSP write to this register generates an interrupt to the host. Similarly, a host read from this register generates an interrupt to the DSP. The DSP status register is accessed by the host to determine the status of DSP communication registers.

The RTOS running on the host uses interprocess communication primitives to provide race-free synchronization and communication mechanisms. The RTOS supports message queues that are not bound to any task. Tasks may send messages to a queue, and several tasks may request messages from the RTOS. When the DSP writes to the host data receive register, an interrupt service routine on the host places the DSP message on the input subsystem queue. This implementation provides a seamless interface between APIs running on the host processor and the HCR subsystem running on the DSP.

Application Command Protocols

The application command table (ACT) for the HCR subsystem is shown in Table 1. The INITIALIZE, RESET–HCR, and BUILD–RESULT commands have no parameters and are sent directly to the DSP data receive register by the host's APIs. The SET–PARMS and PROCESS–STROKE commands are sent after their parameters have been written to the input section of the shared memory. The DSP sends a RESULT–READY after it has generated the recognition results and written them to the output section of the shared memory.

Table 1. Application Command Table for HCR Subsystem

Command Name	Host to DSP?	Parameters	Command ID	Function
INITIALIZE	True	None	00	Initialize variables on DSP for HCR
RESET–HCR	True	None	01	Reset HCR context for new entry
SET–PARMS	True	Database, dictionary, # of return strings	02	Set HCR parameters
PROCESS_STROKE	True	Stroke data	03	Perform incremental recognition on next stroke
BUILD–RESULT	True	None	04	Generate recognition result(s) based on parameters
RESULT_READY	False	Recognition results	10	Signal that the HCR results are ready

Results

As of this writing, the implementation of HCR using the DSP/memory card is not yet complete. Porting of the HCR recognition software to a PC-resident EVM board containing a TMS320C5x DSP and sufficient memory to emulate the DSP/memory card is in progress. Initial results indicate that the overhead in transferring data between the two processors is minimal and that a high degree of parallelism is possible. The final porting of the code depends on availability of the DSP/memory card.

References

1. Pawate, Basavaraj, Frantz, G.A., and Chirayil, Raj, "System Design Using Memory With a Processor Having Communication With Host Processor", Texas Instruments (patent pending), 1993.

2. Pawate, Basavaraj, Frantz, G.A., and Chirayil, Raj, "System Design Using Memory With a Host Processor for Activating a CP", Texas Instruments (patent pending), 1993.

3. Chirayil, Raj, and Pawate, Basavaraj, "PCMCIA DSP/Memory Card Specification", Texas Instruments, November, 1992.

4. *TMS320C5x* User's Guide, Texas Instruments, 1993.

5. Pawate, Raj, "BASAVA Concept", Texas Instruments, November, 1991.

Implementation of an HMM-Based, Speaker-Independent Speech Recognition System on the TMS320C2x and TMS320C5x

B. I. (Raj) Pawate
Peter D. Robinson
Speech and Image Understanding Laboratory
Computer Sciences Center
Texas Instruments Incorporated

Abstract

In the years to come, speaker-independent speech recognition (SISR) systems based on digital signal processors(DSPs) will find their way into a wide variety of military, industrial, and consumer applications. This paper presents an implementation of a hidden Markov model (HMM) speech recognition system based on the 16-bit fixed-point TMS320C2x or TMS320C5x DSP from Texas Instruments. It includes a description of a minimal TMS320C5x-based system and shows how the HMM algorithm and the system algorithm interact. The report also presents system loading, along with a current list of the speech templates. In addition, it presents data showing the relative performance of the algorithm in a controlled environment. A discussion is included on future algorithm enhancements and reduction of the physical hardware system. The paper concludes with a description of a very large vocabulary SISR system based on the TMS320C3x and TMS320C4x floating-point DSPs.

Background

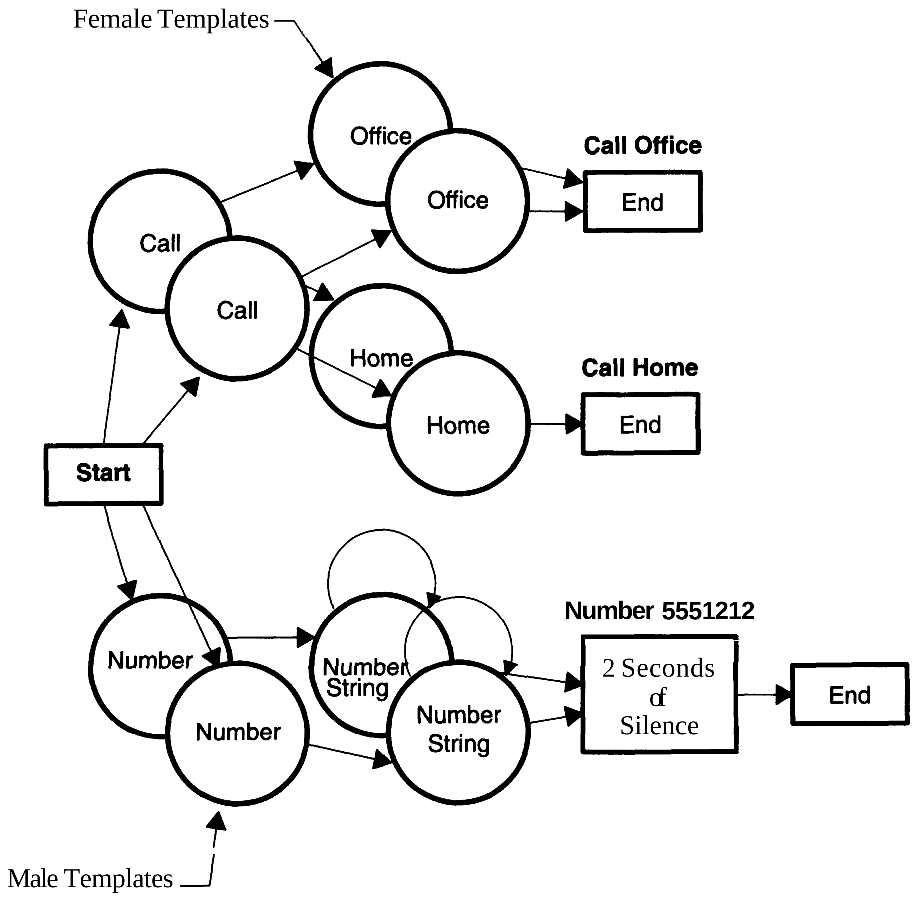
Prior to the introduction of the HMM word recognizer, speech recognition was based almost entirely on dynamic time warping (DTW) techniques. These DTW recognizers are limited in that they are speaker dependent and can operate only on discrete words or phrases (pseudoconnected word recognition). They employ a traditional bottoms-up approach to recognition in which isolated words or phrases are recognized by an autonomous or unguided word hypothesizer. The recognition technique employed in DTW is straightforward. A set of speech templates are maintained in memory for each word/phrase in the vocabulary. These templates are based on linear predictive coding (LPC) models. As a new word or phoneme is acquired and placed in the speech queue, its features or characteristics are compared to each memory-resident template one word/frame at a time. As the acquired speech frame is compared, it is stretched or compressed in time to optimize the correlation between the memory-resident templates and the queued speech frame (hence, the term dynamic time warping). As this warping process progresses, the result of the optimized correlation is logged, and the score is updated. This is repeated for each template in the current vocabulary list. A process is then run on all the collected scores, yielding a best-guess estimate or hypothesis of the recognized word. These DTW systems have been implemented on DSP platforms with a throughput as small as five million instructions per second (MIPS).

The TMS320-Based HMM Recognizer

The Texas Instruments speaker-independent continuous word recognizer provides a top-down approach to speech recognition using the continuous-density hidden Markov model. The Markov model (or the Markovnikov rule) was introduced by a Russian organic chemist, Vladimir Vasilyevich Markovnikov in 1870. HMMs are statistical or stochastic processes, which, when applied to speech recognition, bring machine-based voice recognizers to new levels of performance. However, this increase in performance has its price. HMM-based speech recognizers require a digital signal processor, such as the TMS320C25, that can execute a minimum of 10 MIPS. As this report shows, the improved accuracy and system flexibility provided by the HMM-based system outweighs the added cost of the ’C25 or ’C5x over the 'C1x (5 MIPS).

The Texas Instruments Speech Research Group in Dallas implemented the HMM-based speech recognizer described in this paper on a ’C25 in June 1988. This original application, which contained a vocabulary of 15 words (15 male and 15 female templates), implemented a voice dialer to show proof of concept. Figure 1 shows the grammar rules or vocabulary flowchart for this application.

Figure 1. Voice Dialer Sentence Hypothesizer Flowchart


Female Templates
Office
Office
Call Office
End
Call
Call
Home
Home
Call Home
End
Start
Number 5551212
Number
Number
Number String
Number String
2 Seconds of Silence
End
Male Templates

The HMM voice dialer can currently run on three platforms:

- A stand-alone TMS320C25-based voice dialer demonstration box
- A custom dual TMS320C25-based development platform named Calypso
- The TMS320C5x Evaluation Module (EVM) with analog front-end board

In addition to the 15 words used in the voice dialer application, a total of 49 voice templates (male and female) are available for a user's unique end application. Table 1 lists the 49-word HMM vocabulary. Example sentences follow the table.

Table 1. Current HMM Vocabulary (49 Words)

ADD	AREA–CODE	BACK	BLOCK
CALL	CANCEL	CONFERENCE	CREATE
DELETE	DISABLE	DISTURB	DO
EMERGENCY	ENABLE	ENTER	EXTENSION
FORWARD	FROM	HOLD	HOME
LAST	MAIL–LIST	MESSAGE	NO
NOT	NUMBER	OFFICE	PLAY
PROGRAM	RECORD	REDIAL	REVIEW
SEND	STOP	TO	TRANSFER
WAITING	YES	ZERO	OH
ONE	TWO	THREE	FOUR
FIVE	SIX	SEVEN	EIGHT
NINE			

Example sentences from the vocabulary include:

- Call home
- Call office
- Call number five five five one two one two send
- Call extension two three enter
- Delete extension three five enter
- Create extension seven seven enter
- Disable do not disturb
- Disable call waiting
- Enable call back
- Block last call

Voice-Dialer Performance Testing

In 1990, a test was conducted on the HMM recognizer using the standalone voice-dialer demonstration box. **A** total of 2,272 sentences were tested in a closed-set experiment utilizing word templates from the vocabulary database noted above. Test sentences included the words ***call,*** *office,* ***home, area–code,*** *number,* ***extension, entel;*** and ***cancel,*** in addition to the normal digits. Speed-dialed sentences included the words ***home,*** *office,* and ***emergency.***

Sentence Recognition Performance

Total number of sentences	2,272	
Total sentence errors	133	(5.9%)
With substitutions	73	(3.2%)
With deletions	41	(1.8%)
With insertions	19	(0.8%)

Word Recognition Performance

Total number of words	12,785	
Total word errors	148	(1.2%)
With substitutions	84	(0.7%)
With deletions	45	(0.4%)
With insertions	19	(0.1%)

System Considerations

The system considerations or desirable objectives for a recognizer can be broken into two categories — functional (or ergonomic) and technical:

Functional Requirements

- Speaker-independent recognition (no training required)
- Recognition of connected or continuous words (natural speech)
- High level of accuracy
- Ability to work on a wide cross section of dialects
- Reasonable size of vocabulary
- Affordability

The functional criteria are straightforward, and the system should perform well enough to make it usable in a quiet environment by a majority of the population. Current low-cost, machine-based recognizers are not sufficiently robust to recognize all people at all times. Therefore, it is important to set limits on the level of performance. These limits or restrictions can be determined only through experimentation and test marketing.

Technical Requirements

The technical objectives are much easier to define because they are price- and performance-driven.

- Utilize as little memory as possible
- Work on a 16-bit fixed-point microcontroller/DSP
- Incorporate minimal chip count for a small system form factor
- Use single voltage and low power for battery operation

Things to Come

As the technology progresses, speech recognition will find its way into a wider base of applications. These developments are currently under way at Texas Instruments:

- Adaptation of a microphone array for acoustic beam forming
- Active creation or modeling of background noise for noise templates
- Speaker-adaptive speech recognition
- A mix of speaker-dependent and speaker-independent recognition

Each of the listed techniques may or may not increase the perceived performance. However, they all show promise. The hardest problem to overcome is background noise management, or the art of listening in the presence of noise. Noise management algorithms require an extensive amount of processing power to implement. As an example, adaptive noise cancellation deals with the problem of removing correlated noise (that is, noise that has some redundancy associated with it). This process requires large amounts of data memory, and, as noted, it is computationally intensive. Another technique that shows promise is the use of a microphone array. The array can focus or listen in a specific direction while subtracting the noise in all other directions. Another noise-related enhancement is the real-time creation of a template that matches the current background noise. This technique tries to cancel noise by ignoring it; hence, if the noise is known, a null set is returned when the noise is detected.

In addition to enhancing noise performance, it is also desirable to increase the flexibility of the machine-based recognizer. One technique currently under development at Texas Instruments is the inclusion of a speaker-adaptive algorithm. In this algorithm, the SISR routine comes with a set of general-purpose RAM-based templates that are initialized during runtime from some nonvolatile storage media. As a user interfaces with the machine, the machine modifies or optimizes the templates for that user. This technique is useful when there is only one user per session, such as with a PC-based SISR system.

In the area of speaker-dependent and speaker-independent recognition, a provision will be made so that a user ***can*** supplement the existing speech library with user-recorded templates, such as a trade or personal name: for example, CALL JIM.

Example Platform

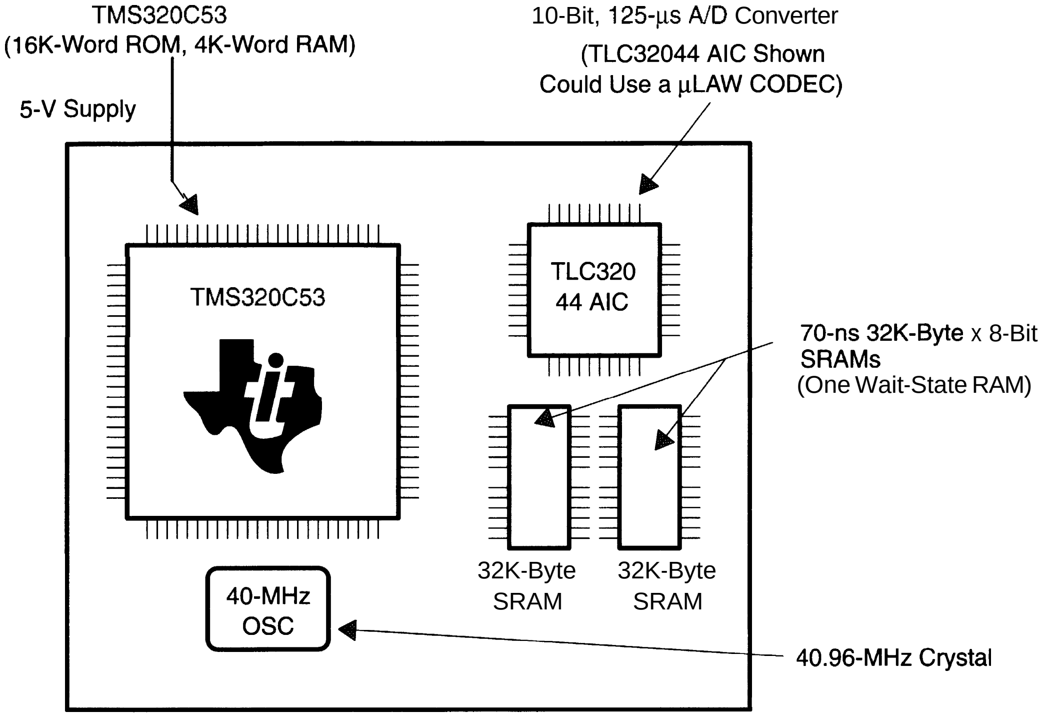
The current fixed-point HMM recognizer, running the voice dialer vocabulary shown in Figure 1, requires a little over 6K words of program and around 40K words of data memory. Table 2 breaks out memory loading on a module-by-module basis and reflects performance on a ’C5x platform running at 20 MIPS.

Table 2. HMM Processor Loading on a TMS320C5x

Module	Data Memory (Words)	Program Memory (Words)	CPU Loading at 20 MIPS
Feature Extractor	5K	1.8K	7%
Compute Word	16K + 0.75K/word (est.)	0.6K	21%
Compute Sentence	5K	1.2K	12%
HMM Executive	0.1K	1.8K	7%
Initialization and I/O	0.1K	0.5K	2%
Totals	**26.2K + Compute Word Templates**	**5.9K**	**49%**

Given the total system memory requirements, this algorithm could be packaged in a single 'C53 with one external A/D converter and two external 70-ns 32K-byte x 8-bit SRAMs. Note that the entire program memory (5.9K words) can reside in ROM. However, all data memory except the compute word templates (0.75K bytes × 16 bits per word) must be of the read-write type.

Figure 2. A Minimal TMS320C53 HMM System


TMS320C53
(16K-Word ROM, 4K-Word RAM)
5-V Supply
10-Bit, 125-μs A/D Converter
(TLC32044 AIC Shown
Could Use a μLAW CODEC)
TMS320C53
TLC320
44 AIC
70-ns 32K-Byte x 8-Bit
SRAMs
(One Wait-State RAM)
32K-Byte
SRAM
32K-Byte
SRAM
40-MHz
OSC
40.96-MHz Crystal

The system shown in Figure 2 provides 16K words of program ROM and up to 36K words of data RAM (it is assumed that there is a host interface for template upload; if not, an additional 1M words of ROM is needed). Further integration is possible with Texas Instruments customized DSP (cDSP) devices. A cDSP implementation will reduce this design to two chips: a monolithic DSP, including an A/D converter with system interface logic, and an external 70-ns 32K-byte × 16-bit SRAM (1/2M word SRAM).

How the Texas Instruments HMM Implementation Works

The Texas Instruments HMM speech recognizer consists of three computational processes running together: a feature extractor, a word hypothesizer, and a sentence hypothesizer. The feature extractor, as its name implies, reduces the continuous speech to a series of 20-ms frames or states whose features are reduced to a finite feature set called a generalized set feature, or GSF. The HMM processes *compute word* and *compute sentence* guide the recognition — first at the sentence level, then at the word level. These three processes interact so that the feature extractor feeds the word hypothesizer, which is no longer autonomous, but guided by a sentence hypothesizer. Hence, recognition is now accomplished on a state-by-state basis.

The HMM processes, at any level, *can be* expressed in terms of mathematical probabilities as the likelihood that one state follows another. If the vocabulary is known and the sentence structure is known and finite, then it is a simple process to predict the next state, given the present and past states. This is done by scoring frames of extracted features along paths that terminate at unique solution end points. Hence, paths scored at the state level point to word level, which points to sentence-level solution sets. All along the way, probabilities are calculated and assigned in guiding the process.

Figure 3 shows graphically how the HMM word hypothesizer works. Within the voice dialer system, after the word CALL is recognized, the sentence hypothesizer has only two paths from which to select: HOME or OFFICE. The lower portion of Figure 3 shows the path selection resulting in OFFICE.

Figure 3. Example of an HMM Flow

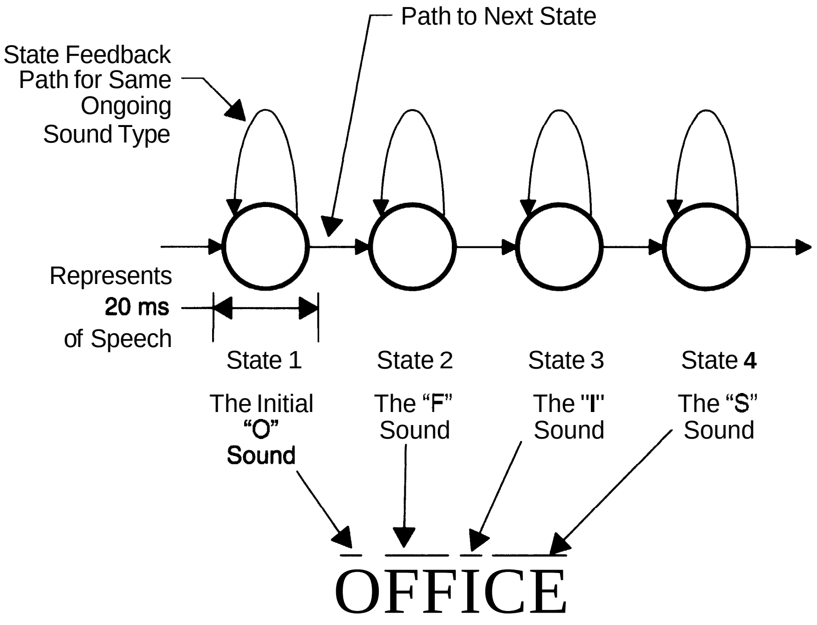

Path to Next State
State Feedback Path for Same Ongoing Sound Type
Represents 20 ms of Speech
State 1
State 2
State 3
State 4
The Initial "O" Sound
The "F" Sound
The "I" Sound
The "S" Sound
OFFICE

Figure 4 on page 222 shows how this application of the hidden Markov model continuous word recognizer is implemented on the 'C2x or 'C5x. The speech data flows through the model from left to right, while the recognition is driven from right to left.

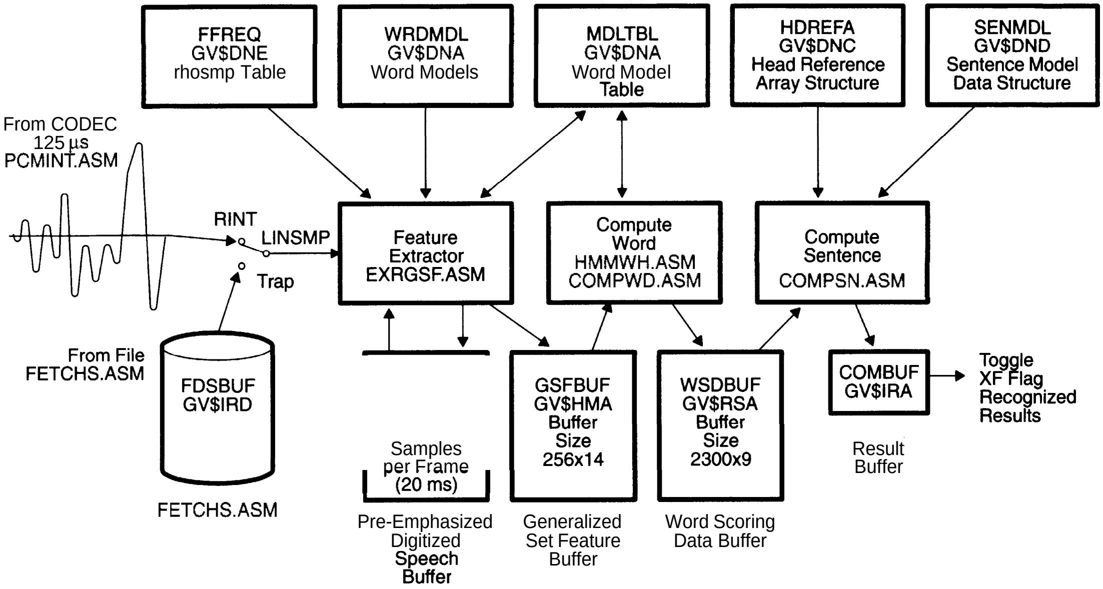
The process is started and sustained as follows. Time samples, which are taken every 125 µs, are queued in the pre-emphasis digitized speech sample buffer (PDSBUF). These samples are then operated upon by the feature extractor on a frame-by-frame basis (a frame is equal to 160 samples). The feature extractor interfaces to five data structures:

- The LPC filter coefficient table, or rhosmp table, as noted in Figure 4
- The pre-emphasis data structure buffer
- The word models
- The word model table
- The generalized set features buffer (GSFBUF)

These data structures and their contents are discussed on the following pages. In general, the feature extractor performs two functions:

1. It reduces a frame of speech data to a finite data set that describes the speech type. This reduced data is called a state, which is the smallest unit of time in the algorithm (20 ms).
2. Next, it expresses the state so it can be approximated by a Gaussian distribution.

Figure 4. Block Diagram of the HHM Recognizer

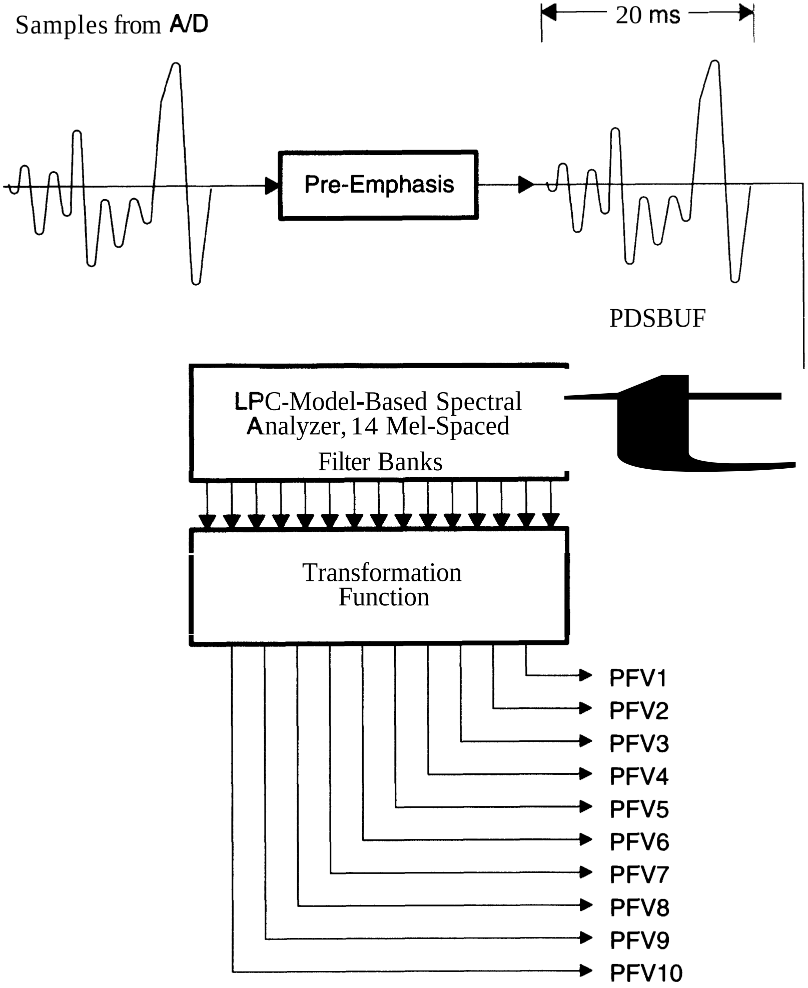

FFREQ
GV$DNE
rhosmp Table
WRDMDL
GV$DNA
Word Models
MDLTBL
GV$DNA
Word Model Table
HDREFA
GV$DNC
Head Reference Array Structure
SENMDL
GV$DND
Sentence Model Data Structure
From CODEC
125 µs
PCMINT.ASM
RINT
LINSMP
Trap
Feature Extractor
EXRGSF.ASM
Compute Word
HMMWH.ASM
COMPWD.ASM
Compute Sentence
COMPSN.ASM
From File
FETCHS.ASM
FDSBUF
GV$IRD
FETCHS.ASM
Samples per Frame (20 ms)
Pre-Emphasized Digitized Speech Buffer
GSFBUF
GV$HMA
Buffer Size
256x14
Generalized Set Feature Buffer
WSDBUF
GV$RSA
Buffer Size
2300x9
Word Scoring Data Buffer
COMBUF
GV$IRA
Result Buffer
Toggle XF Flag Recognized Results

Once a frame of speech is processed by the feature extractor, the results are queued in the GSFBUF. At this point, the 160-element frame has been reduced to a 14-element state vector in the GSFBUF. These 14 memory elements contain the following information:

- Frame energy
- Inner product
- 10 principal feature vectors (PFVs), PFV1 through PFV10
- Utterance flag (voiced or unvoiced speech)
- Ongoing-word flag (still in same utterance)

Once a new frame is added to the GSFBUF, the HMM process takes over (compute word and compute sentence). The function of the HMM is to present a hypothesis on an optimal path for the frame. Hence, the contents of the GSFBUF are continuously being interrogated by the word hypothesizer to determine the best path score to a unique end point (word), given the current state and previous states observed. In addition to the RAM-based buffers, there are three ROM data structures that the feature extractor accesses. The rhosmp table contains all the coefficients used in the various data reduction routines within the feature extractor, the 14 5-tap filters, and the LPC-10 coefficients.

Figure 5. The Feature Extractor


Samples from A/D
20 ms
Pre-Emphasis
PDSBUF
LPC-Model-Based Spectral Analyzer, 14 Mel-Spaced Filter Banks
Transformation Function
PFV1
PFV2
PFV3
PFV4
PFV5
PFV6
PFV7
PFV8
PFV9
PFV10

Next, the word model (WRDMDL) data structure contains all the word model templates in the vocabulary. This buffer is typically the largest memory array within the recognizer. The word hypothesizer indexes into this data structure via the word model table (WDLTBL). This table contains the starting address, length, and word ID for each word model. As noted above, extracted features or states are queued in the GSFBUF. They are correlated against the valid word models, as determined by the word and sentence hypothesizer for that state.

Once a word is processed, all associated state vectors are removed from the GSFBUF and transferred to a slot of a buffer in memory called the word scoring data buffer (WSDBUF). Each slot in the WSDBUF stores the following:

- Level index — sentence-level, word-level, or states
- Model index — current index pointer into the word model data structure
- State index — what index within the model
- Path–scr — best-path score for the current frame
- Path–ptr — scoring data structure (SDS) index of the previous frame in best path
- Time — input frame time index
- Last–time — time index of last path through this point
- Next–state — SDS index of next state for this model
- Next–word — SDS index of next word

The data is now reduced from 160 words (PDSBUF) to 14 words (GSFBUF) and to a 9-word WDS buffer. However, as multiple state vectors were required for each word in the GSFBUF, multiple WDSBUF frames are needed for best path determination, resulting sometimes in an increase in total memory requirement to sustain the HMM process. The final phase in the HMM process is the reduction and subsequent linking of the WSDBUF vectors (based on their path score) so that an optimal path vector set remains. This vector set points to a unique word ID determined by the read-only sentence model (SENMDL) data structure. This array maps vocabulary words to model IDs within the sentence IDs or compute sentence, in conjunction with the head reference array structure (HDREFA). This read-only array maps a vocabulary word ID to its first model ID. The subsequent fields in the HDREFA model table are used to link multiple models for a given word ID when that word ID is used more then once in a sentence model. With the IDs known, the word IDs are passed to the COMBUF, where the host system reads the results. Hence, the COMBUF contains the recognized words that are to be returned to the host. The COMBUF is organized as follows:

- ID of the recognized word model
- The error (32 bits) in the word model
- Frame index of the word model's beginning
- Frame index of the word model's ending
- Frame index at which the word model was created

Fixed Point Versus Floating Point

Thus far, the discussion has focused on implementing the HMM algorithm on a fixed-point DSP. A floating-point processor such as the TMS320C3x, with its vast 16M-word address range, DMA controller, and inherent floating-point attributes, makes coefficient representation a nonissue. The elimination of numerical concerns can significantly reduce development time, but this is not necessary for implementing the HMM algorithm. As shown, a fixed-point processor performs the algorithm equally well and can significantly reduce system cost. However, executing a fixed-point system requires a thorough understanding of the complex numerical issues. Typical fractional variables, such as the features used to represent the acoustic data (GSFBUF), are represented on a fixed-point DSP by using a $Q_{n/m}$ format. With this format, the 16-bit 2s-complement field is evaluated with a sign bit, n integer bits, and 15-n fractional bits.

Figure 6. Example of Q 4/16 Notation

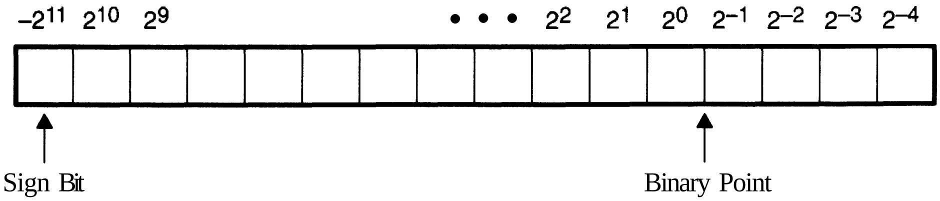


Figure 6 shows the dynamic range of a $Q_{4/16}$ number is: 2^{11} – 2–4 = 2047.9375 to –2^{11} = –2048, where a $Q_{15/16}$ number would range from 2^0 – 2–15 = 0.99996948 to –2^0 = –1.

Table 3 shows several examples of $Q_{n/m}$ notation, as used in the implementation of the fixed-point Texas Instruments HMM recognizer.

Table 3. Examples of $Q_{n/m}$ Notations (Fixed-Point Representation)

Variable	$Q_{n/m}$ Notation
Feature vector	$Q_{4/16}$
Cumulative pathscores	$Q_{15/16}$
Log of transition probabilities	$Q_{15/16}$
Log of observation probability	$Q_{15/16}$

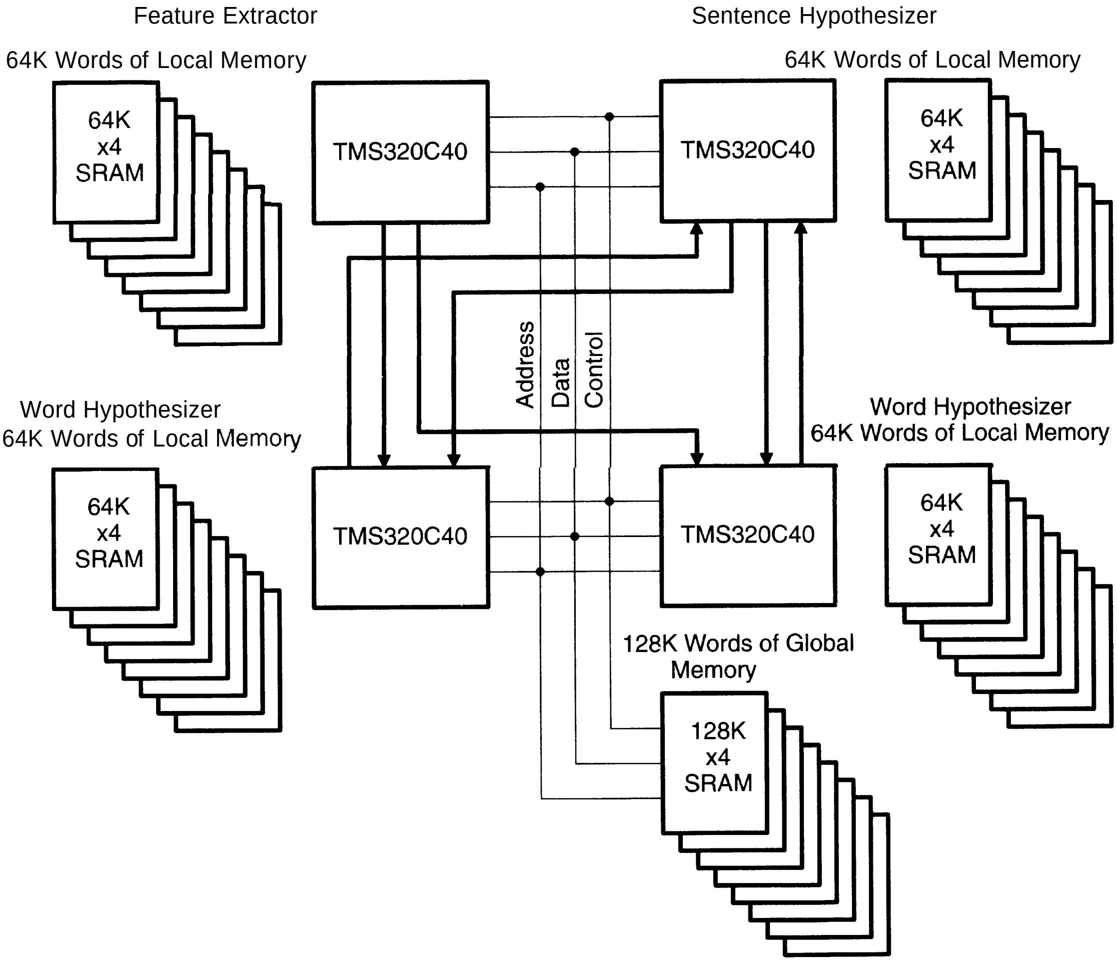
As noted, a fixed-point DSP can reduce system cost. However, in SISR systems, size of vocabulary is all-important. SISRs need a floating-point system, not only for its ability to represent values in floating-point format, but also for its memory reach. The ’C2x and ’C5x can access only 64K words of linear data memory, while the ’C3x can access 16M words, and the ’C4x 4G words. The size of the vocabulary is limited only by processing power, as opposed to accessible system memory. To implement a very large vocabulary recognition system via the HMM technique presented here, the following must be accomplished.

- The feature extractor must be improved to increase its granularity and to make it more robust. As more and more words are added to the vocabulary data base, it becomes increasingly more difficult to distinguish similar sounding words.
- A sentence hypothesizer must be developed that can track and predict words according to grammar rules for the English language. In addition, the sentence hypothesizer must be adaptive in that it must be able to learn user-specific grammar rules (slang).
- A word hypothesizer must be developed that is speaker adaptive (work ongoing) and allows the addition of user-defined vocabulary (again, work ongoing).
- A technique must be developed for creating templates from text-based descriptions. Optimally, these descripters should be based on a published standard, such as the symbols used in the respelling for pronunciation, as found in dictionary pronounciation guides.

Example: **elephant** _(ĕl′ə -**fa** nt)

Figure 7 shows a very large vocabulary SISR system based on the 'C4x parallel processor development system (PPDS).

Figure 7. SISR System for Very Large Vocabulary


Feature Extractor
Sentence Hypothesizer
64K Words of Local Memory
64K Words of Local Memory
64K x4 SRAM
TMS320C40
TMS320C40
64K x4 SRAM
Address
Data
Control
Word Hypothesizer
64K Words of Local Memory
Word Hypothesizer
64K Words of Local Memory
64K x4 SRAM
TMS320C40
TMS320C40
64K x4 SRAM
128K Words of Global Memory
128K x4 SRAM

The feature extractor, compute sentence, and word hypothesizer are distributed over the four 'C40s. The word hypothesizer uses two 'C40s because it is the most computationally intensive task. The feature extractor feeds output (frame or state data) to the two word hypothesizers via two 8-bit parallel ports. In addition, the sentence hypothesizer feeds both word hypothesizers, which, in turn, feed their results back to compute sentence. Although the above system has not been implemented. it demonstrates a logical progression of the technology.

Conclusion

In summary, one TMS320C53 DSP can implement a robust HMM speaker-independent speech-recognition system with just under 50% processor loading. This. with future enhancements to the existing HMM SISR algorithm and hardware systems, makes a single-chip DSP-based recognizer in a noisy environment a reality. This paper discusses the system resource requirements, vocabulary flexibility. and possible future enhancements. The data presented shows how a fixed-point processor is ideal for small vocabulary systems in which expense and power are a concern. The paper also shows how this HMM-based algorithm can be adapted to a floating-point processor, allowing for a very large vocabulary system.

Automated Dialing of Cellular Telephones Using Speech Recognition

Frank Henry Dearden III
Voice Control Systems, Incorporated

Introduction

The cellular telephone industry has experienced tremendous growth since its beginning more than ten years ago. What was once considered to be a toy for high-profile executives has now become an integral communications tool for over 14 million subscribers in the U.S. alone. Growth rates are expected to accelerate during the next few years.

Automated speech recognition (ASR) technology has been a bedfellow of cellular telephone technology for many years. Most of the large cellular subscriber unit manufacturers have developed their own ASR systems to facilitate hands-free dialing. The benefits of combining these two technologies are obvious: the less time and focus a driver gives to placing a call, the more attentive he is to operating the vehicle. Hands-free kits that include a far-talk microphone and speaker are now required by law in some European countries for conversing once a call is connected. Various states are currently considering similar requirements. Similarly, requirements for hands-free dialing capability via speech recognition are not too far off.

This paper explains how ASR-enabled dialing capability can be implemented with DSP technology from Texas Instruments. Speech recognition technology has never been as accurate, user-friendly, and inexpensive as it is today, or as easy to integrate into state-of-the-art cellular subscriber systems.

The Technology

Most of the past and existing ASR units on the market are limited to what is known as ***speaker-dependent*** (SD) technology. This technology has exhibited some rather fundamental performance limitations. SD systems work by comparing a whole word input with a user-supplied ***template.*** Templates are developed by each user during a rather cumbersome training exercise, which usually takes place in a quiet, stationary environment. Since the systems are used in a moving car environment, the increase in background noise, coupled with a user's inflection change (people usually shout slightly, and unconsciously, when a car is in motion) confuse most SD systems. Accuracy rates are typically less than 90%.

Speaker-dependent ASR systems are steadily being replaced with ***speaker-independent*** (SI) systems. SI-capable systems approach the recognition problem in a fundamentally different manner than SD-only systems. Once an input command is captured and digitized, an SI system will parse it into phonetic-like pieces, or ***features.*** These speech features are then compared with supplied target data, not with templates supplied by the user.

The training procedure for a speaker-independent recognizer is both processing and data intensive. Speech variations due to sex, age, accent, and speaking habits must be considered, along with the great variety of noise sources, internal and external to the car, that have a tremendous effect on the signal-to-noise ratio. This implies that an application-specific speech data base is required for the vocabulary ***training*** process. Consequently, each SI vocabulary is essentially hand-crafted for the particular word list and the environment of use. The diversity of the training data helps account for the robustness of the resultant recognizer in the presence of real users and all types of automobile noise.

Usually, speech-independent reference data is derived from a large data base of speech ***tokens*** collected inside several cars, from hundreds of speakers, over a variety of road conditions, and with high-quality digital recording equipment. The computer-controlled recording equipment has a display screen that automatically prompts the ***donor*** to speak through a given vocabulary. The incoming speech sample is transduced by a noise-canceling microphone placed on the windshield and is recorded on a remotely controlled digital audio tape (DAT).

The result is a scheme that is extremely robust. Matching pieces of sound to feature templates derived from rigorously collected data reduces the amount of computational power required and is more forgiving of inflection change than an SD scheme. For example, a cold will make John sound less like John specifically, but his speech will continue to exhibit feature characteristics consistent with the statistical samples derived from the database. Additionally, technologists at Voice Control Systems Incorporated (VCS) have done empirical analysis on SI feature recognition and have even identified some features that occur often but are irrelevant to recognition. The complexity of the task can be reduced and the odds of a successful recognition increased if some of these redundant features are disregarded.

The Human Interface

All recognition systems consist of two basic components: the core recognition engine and the human interface. The adage "you're only as good as your presentation" is very apropos when designing ASR systems. Technologists tend to devote most of their time to enhancing a system's raw recognition power, bandwidth, and memory allocation, etc. This is all well and good. Marketers however, should make sure that the interface gets an equal amount of attention.

Besides high accuracy, the major benefit of a speaker-independent capable system is its intuitive, user-friendly presentation. Acceptance by the user is critical, especially during the first use. The system should prompt the user with high-quality, stored human speech and should respond quickly to each input. The result should be a semiconversational experience, such as the following (the user input is in bold **CAPITALS** and the response is in lowercase):

"VOICE CONTROL"
    "ready"
"CALL"
    "calling?"
"OFFICE"
    "calling office, correct?"
"YES"
    "dialing..."

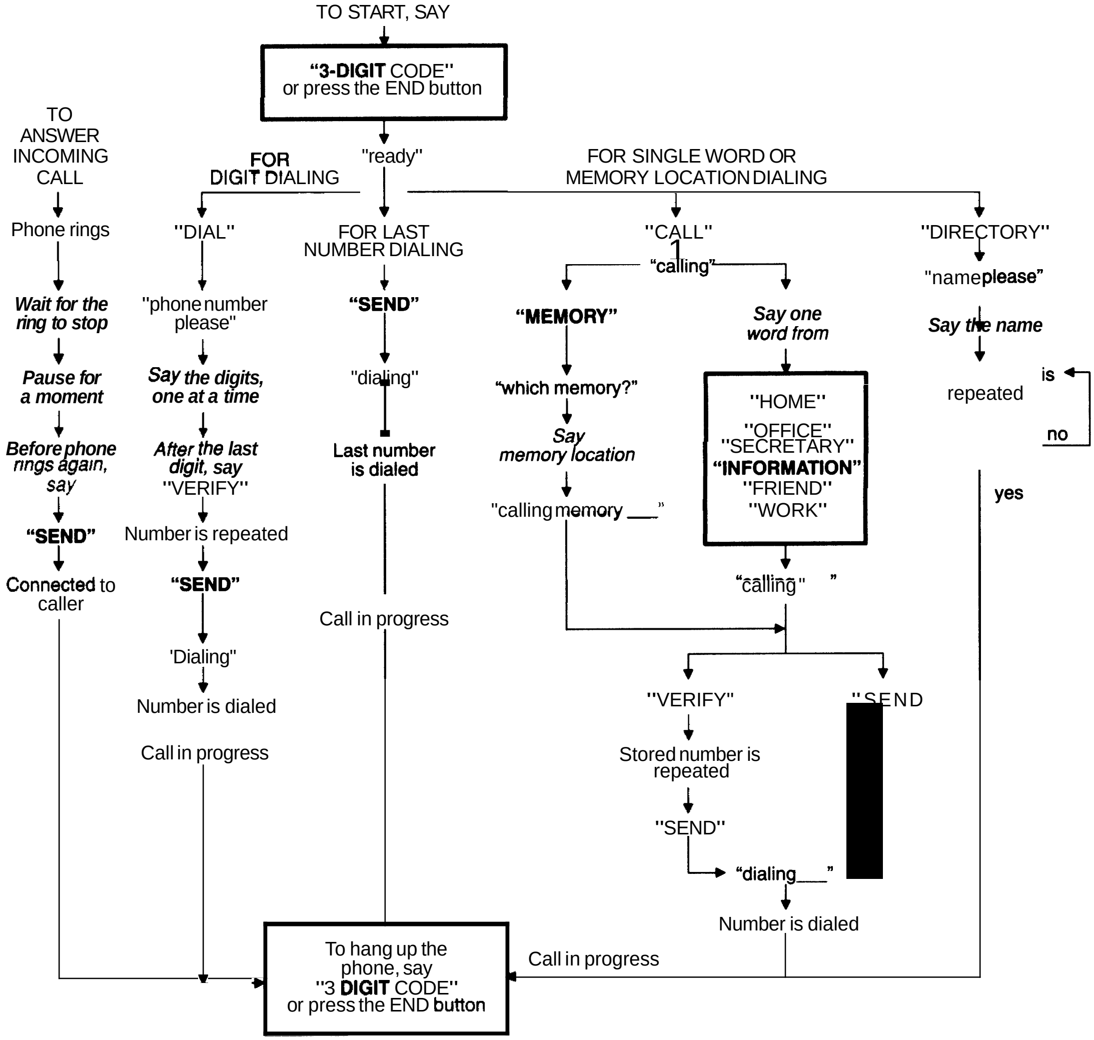
In this example, the user accesses a memory location by using one of many possible predetermined name tags (for example, office, home, school, information, doctor, etc.).

A user should also be able to place a random phone call by using a speaker-independent digit dialing sequence, like this:

"DIAL"
    "phone number, please"
"THREE"
    [beep, display 3]
"SEVEN"
    [beep, display 7]
        etc.
"VERIFY"
    "three, seven, (etc.)"
"SEND"
    "dialing..."

Figure 1 shows a flowchart, or decision tree, of a well-tested human interface.

Figure 1. Flow Diagram of Human Interface


TO START, SAY
"3-DIGIT CODE" or press the END button
"ready"
TO ANSWER INCOMING CALL
FOR DIGIT DIALING
FOR SINGLE WORD OR MEMORY LOCATION DIALING
Phone rings
Wait for the ring to stop
Pause for a moment
Before phone rings again, say
"SEND"
Connected to caller
"DIAL"
"phone number please"
Say the digits, one at a time
After the last digit, say "VERIFY"
Number is repeated
"SEND"
'Dialing"
Number is dialed
Call in progress
FOR LAST NUMBER DIALING
"SEND"
"dialing"
Last number is dialed
Call in progress
"CALL"
"calling"
"MEMORY"
"which memory?"
Say memory location
"calling memory ___"
Say one word from
"HOME"
"OFFICE"
"SECRETARY"
"INFORMATION"
"FRIEND"
"WORK"
"calling"
"VERIFY"
Stored number is repeated
"SEND"
"dialing___"
"SEND
Number is dialed
Call in progress
"DIRECTORY"
"name please"
Say the name
repeated
is
no
yes
To hang up the phone, say "3 DIGIT CODE" or press the END button

NOTE: User input is in bold CAPITALS, the response is in lowercase, and directions are in *italics.*

Note that a system can be both speaker-independent capable and speaker-dependent capable. SD technology allows a user to assign personal name tags to memory locations in addition to the SI locations mentioned above. Depending on the memory available, a user can program phone numbers into memory locations labeled "John Smith", "Fred's Office", "Pizza", etc. For the greatest recognition accuracy, it is best to limit the number of customizable name tags to about ten. VCS uses its feature-matching algorithm for SD comparisons as well as for SI comparisons, resulting in high accuracy rates.

The Implementation

VCS has focused solely on developing ASR technology for the past 14 years. Most of VCS's more than 90,000 fielded systems are multichannel telephone network-based installations, which allow random callers to utilize voice mail or other interactive response functions without the need for touch-tone input. The recognition algorithms in these applications are handled by dedicated TI DSP hardware.

VCS began to migrate its ASR expertise into single-channel applications about four years ago. The goal was to maintain high functionality while minimizing hardware cost and space requirements. These first products used custom interface circuitry, standard X86 microprocessors, a CODEC, and memory for the core recognition hardware. The custom chip has been redesigned to reduce cost for the recognition core to about $30 in quantities of 10,000. Manufacturing tooling, testing, packaging, and labor expenses can easily lead to a total cost per unit of twice this amount. Although the circuit can be made quite small, adequate space must be allowed in a transceiver unit, a 3-watt booster, an external enclosure, or even within a handset cradle. Building this chipset directly into a portable cellular telephone remains impractical at this time.

ASR can also be used with digital cellular phones because VCS code can take advantage of the hardware already resident within the handset. This hardware includes a CODEC, memory, and digital signal processing capability. Consequently, adding ASR code may require some additional memory capacity but does not require the design and manufacture of an entire circuit board. The total cost is greately reduced — from about $60 to about $15 — which includes additional memory and software licensing.

Since VCS's ASR code uses only a small amount of DSP bandwidth, the cellular telephone can execute recognition operations in parallel with being enabled for incoming calls. For example, our discrete speaker-independent and speaker-dependent capability utilizes about 25% of the bandwidth for one channel of recognition on a TMS320C25 operating at 40 MHz. Additionally, the ASR code does not compete with the digitization and companding exercises undertaken during a conversation, because the recognition task for dialing precedes the actual placement of a call.

In this scheme, the cellular telephone task master communicates with VCS ASR object code via applications programming interface (API) commands. This involves a reasonable level of integration, but the end result is the lowest incremental cost option for adding ASR to a cellular telephone. An API for the Texas Instruments TMS320C2x DSPs can be acquired directly from VCS.

Accuracy

VCS has designed a tape test exercise to systematically determine the recognition accuracies of a newly designed voice recognition unit (VRU); the procedures for quantifying the performance of speaker-independent and speaker-dependent commands are different. A properly designed VRU will utilize these two technologies to maximize the acceptability of the system by the operator.

Tape testing is conducted under laboratory conditions and with a direct audio path between the tape and the VRU. The total number of SI commands a system is capable of recognizing is simply a function of available memory. However, at any given time, only a specific subset of the total SI vocabulary should be active. In general, each subvocabulary should be limited to about 12 elements, even though larger subvocabularies are possible. Smaller subvocabularies maximize the performance of the technology and minimize operator choice and confusion. Each speaker-independent subvocabulary (that is, each path in the tree) should be tested.

The test data includes 50 speakers, of which half are male and half are female. The data is obtained from a data collection of every recognizable word in the vocabulary, as described above. These data are reserved for testing purposes only and are not to be used to train the VRU.

Each response is recorded as the source tape is played. Twice, the tape plays each person speaking the entire speaker-independent vocabulary, divided into the designated subvocabularies. The expected error rates for VCS speaker-independent technology are:

Average rejection error rate	< 3.0%
Average substitution error rate	< 1.5%

A rejection error occurs when the system rejects a valid word input on the basis of insufficient *class* distinction. *A* substitution error occurs when the system substitutes another word from the active vocabulary in response to a valid word input. On occasion (less than 1% of the time), the system may not respond to a spoken input, because the word was not spoken loud enough. These cases should be ignored when the rejection and substitution error rates are computed.

Softening the impact of an error is the job of the user-friendly interface. For example, if the VRU responds with a polite *"pardon?"* following a rejection error, most people will patiently repeat the input (at least once) and enunciate a bit more clearly. The system typically accepts the next attempt, and the user proceeds, sometimes unaware that an error has occurred. For this reason, an SI rejection error rate under 4% is perfectly acceptable for most users.

VCS systems have the capability to handle at least one SD vocabulary, although with enough memory, more are possible. However, only one vocabulary should be active at any given time. During testing, this speaker-dependent memory should initially be cleared. A representative group of ten people, five male and five female, should participate, with a minimum of three passes. Words not easily confused should be used for this test.

home	office	Steve	Bob	Mary Jones
Sears	Jill Miller	weather	voice mail	John Smith

Each member of the group then rotates through the above list ten times, trying to recall the correct command. On average, the expected substitution error rates for VCS speaker-dependent vocabularies are less than 5%. SD vocabularies are not prone to rejection errors.

It is extremely difficult to combine technologies, (that is, to have a speaker-independent vocabulary simultaneously active with a speaker-dependent vocabulary). Situations like this should be avoided, if for no other reason than to minimize the confusion of the operator.

Code Availability

The associated software is available for licensing from Voice Control Systems, 14140 Midway Road, Dallas, Texas 75244. Relevant data sheets are also included in the *TMS320 Software Cooperative Data Sheet Folder,* Texas Instruments literature number SPRT111.

Summary

With a PC, multimedia hardware, and a relevant technical paper in the public domain, an engineer can design a reasonable speaker-dependent ASR system. The accuracy is usually in the mid-80% range, as long as the environment is quiet. Improving this capability to handle speaker-independent input, achieve a 97%+ accuracy in noisy environments, and cost as little as $15 per unit is quite another challenge.

VCS has worked for more than a decade in tedious research and testing to incrementally improve its technology to these levels. It is predicted that the features and benefits offered by ASR will greatly influence subscriber unit purchases.

Part VIII
System Design Considerations

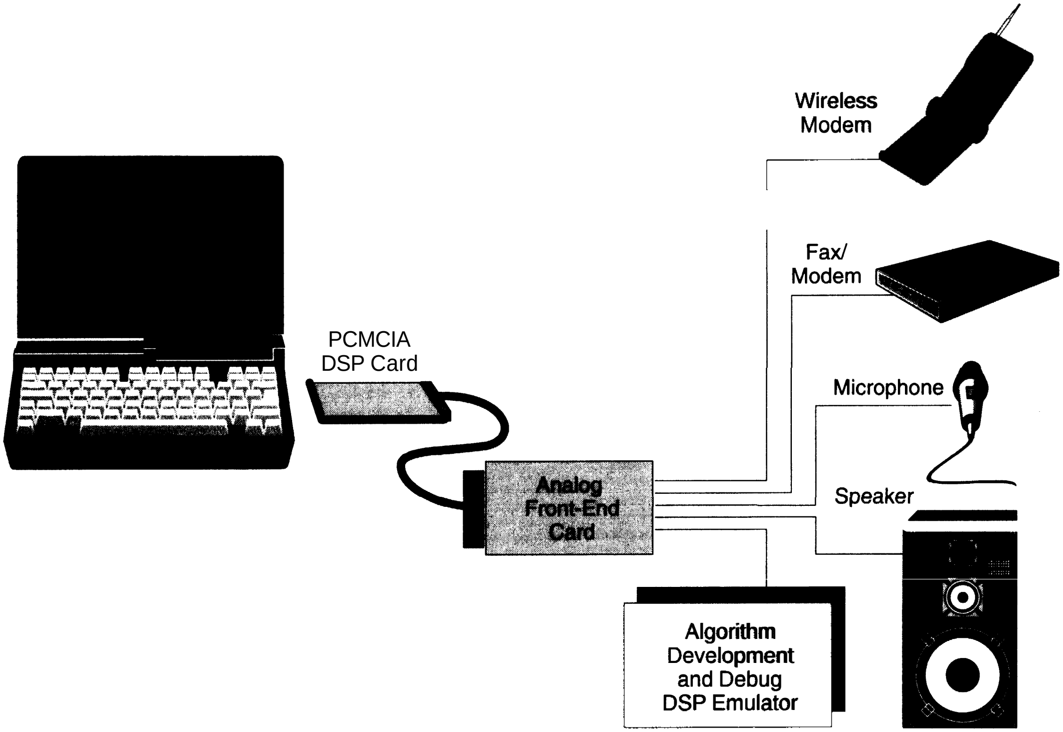
The PCMCIA DSP Card: An All-in-One Communications System

Raj Chirayil
Digital Signal Processing Applications — Semiconductor Group
Texas Instruments Incorporated

Introduction

With the advent of subnotebook computers and personal digital assistants (PDA), there is an ever increasing need for a universal communications engine that is compact, simple to use, and dynamically configurable to suit various operating environments. In the desktop computer world, there is rarely a need for portability, whereas in the world of PDAs, portability is everything. This includes not only the computer itself but also any peripherals that go with it. The Personal Computer Memory Card Interface Association (PCMCIA) standard has made a significant contribution toward meeting this requirement.

Figure 1. DSP Card Block Diagram



All of the PCMCIA cards available today are single-function products and lack the flexibility to be dynamically reconfigured to support multiple applications. The PCMCIA DSP card described here was defined to be used by the host CPUs as a generic coprocessor or as a storage device. For larger data storage applications, the on-board SRAM can be replaced by lower cost, low-power DRAM devices. However, advanced digital signal processing applications such as V.Fast modems and digital cellular phones require higher speed SRAMs to allow full-speed DSP operation. When appropriate algorithms are loaded, the host can transfer data to the on-board memory and command the DSP to perform specific tasks, such as handwriting recognition, image or voice data compression, or music synthesis. An external analog front-end (AFE) card can be connected to the DSP card if the application requires external analog input/output capability.

The architecture and design described here allow users to configure the card as a data/fax modem, speakerphone, telephone answering machine, note taker, character recognition system, or business audio card by merely downloading the appropriate DSP algorithm to the card. Replacing the wireline telephone interface circuit on the AFE card with an RF circuit and antenna allows the same DSP card to support wireless data or voice communications.

A key requirement for any portable system is low power consumption. This DSP card uses a TMS320C51 DSP, which is ideal for the PCMCIA application because of its very low power consumption, high MIPS rate, and very low cost. Another important system requirement for a portable multifunction DSP card is the ability to provide processing power on demand. A card running a simple speech compression algorithm for a note taker may need less than 5 MIPS, whereas a voice-over-data system running a V.Fast or V.32bis modem and higher quality speech compression algorithm may need 40 MIPS or more. Because of the flexibility of the C5x DSP's clock input scheme, this design allows the host PC to configure the DSP to run faster or slower via s/w commands.

System Architecture

The DSP card system architecture is defined so that any algorithm developed for a particular DSP could be run on non-PCMCIA platforms, provided that particular DSP is available. The host system can treat the DSP card as a programmable function coprocessor. The system applications software needs only to know which DSP a particular algorithm is developed for. This allows the DSP card system to be integrated onto a notebook PC or PDA motherboard by merely replacing the PCMCIA interface with the appropriate host system interface.

The PCMCIA interface logic and additional logic needed for the communications and control are implemented in less than 5,000 gates in an FPGA. For highly integrated systems and motherboard applications, these functions can be easily integrated with the DSP as a single device through TI's TEC320 cDSP approach. The hardware- and host-independent architecture supports a Windows™[1] application, using the DSP Resource Manager to run the same application and DSP algorithm on multiple platforms and hosts.

The architecture defines any host memory provided on the card to be shared between the host PC and the local DSP. This eliminates data bandwidth limitations and facilitates fast block transfer of data or downloading of DSP algorithms. Because of this dynamic algorithm loading capability, the host can treat the on-board DSP as a programmable function coprocessor. A real-time memory paging scheme makes it possible to load different application algorithms into different pages for fast reconfiguration and task switching.

The on-board FPGA arbitrates any conflicts for access of shared memory between the host PC and the DSP, with the host access having higher priority. The FPGA also implements all necessary host system interface and control logic. Several dedicated communication registers are provided in the FPGA to allow the host PC and the DSP to communicate without interrupting DSP operation. Buffered registers are provided in the FPGA for the required programmable bit I/O.

The PCMCIA DSP card interfaces to the host as a PCMCIA memory card and an I/O card. The PCMCIA specification supports up to 64MB of PCMCIA common memory in addition to a separate attribute memory space. For a 16-bit fixed-point DSP such as the TMS320C5x, this translates into 32M (16-bit) words of external program/data space. The attribute memory can be used by the DSP as 32M (8-bit) bytes of global data space. Both memories must obviously be paged by a DSP with only a 16-bit address.

---

[1] Windows is a trademark of Microsoft Corp.

The DSP card architecture is expandable to support the full extent of PCMCIA memory, which the DSP can access in paged mode under software control. The paging feature allows users to load different application algorithms into different pages for dynamic reconfiguration and task switching. The PCMCIA common memory is mapped into the DSP's data/program space, and the PCMCIA attribute memory is mapped into the DSP's global data space.

This particular implementation limits the DSP's paged external data and program space to 3M words and global data space to 128K bytes. The DSP card is populated with two sets of fast (15-ns) SRAMs — one 64K-byte × 16-bit SRAM in one set and two 128K-byte × 8-bit SRAMs in the other. When the 64K-byte × 16-bit SRAM is enabled, it is used by the DSP as combined data and program memory. When the 128K-byte × 8-bit SRAM is enabled, it is used as separate 64K words of data and 64K words of program memory. The entire 256K bytes of memory are accessible by the PC in byte mode or word mode. However, the DSP can access only one of these memories at a time, as enabled by the system configuration register. The card also has 128K bytes of flash memory, which can be programmed by the PC. The DSP can be configured to boot load from this flash memory upon reset. Although the entire flash memory is mapped into the PC's attribute memory space, only 32K bytes are mapped into the DSP global data space at any given time.

The host PC can access the card as a 16-bit I/O device by writing to the configuration registers. The I/O address for the card is selectable by the PC in the card configuration registers. When the DSP card is configured as an VO-mapped peripheral, the host communication registers are dual-mapped into the PC's common memory and I/O space.

Figure 2. DSP Card Architecture

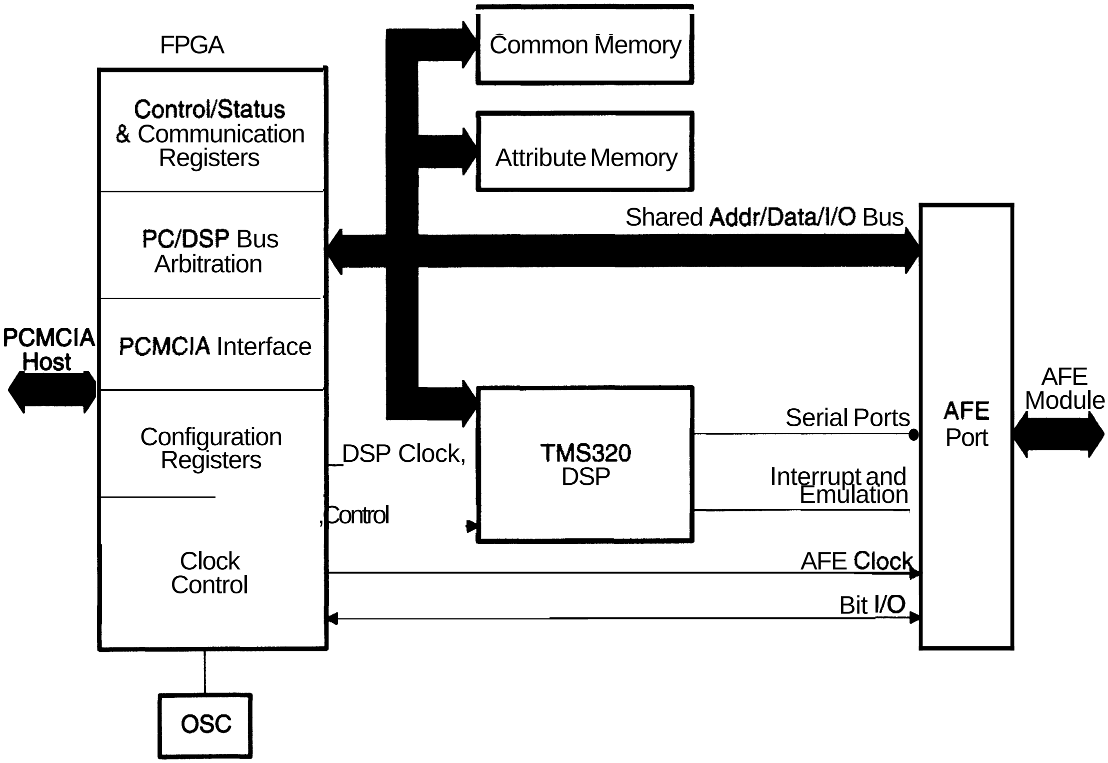

FPGA
Control/Status & Communication Registers
PC/DSP Bus Arbitration
PCMCIA Interface
Configuration Registers
Clock Control
PCMCIA Host
OSC
Common Memory
Attribute Memory
Shared Addr/Data/I/O Bus
DSP Clock,
Control
TMS320 DSP
Serial Ports
Interrupt and Emulation
AFE Clock
Bit I/O
AFE Port
AFE Module

Figure 2 shows a block diagram of the DSP card. The PCMCIA connector appears on the left side of the board. The FPGA integrates all discrete logic in the system. The system clock to the DSP is provided by the FPGA for the control of the DSP's operating speeds. A 48-pin analog front-end (AFE) connector provides an external interface as well as system development and debug. The DSP serial port signals are available at this connector along with programmable input and output pins and DSP interrupt pins to monitor, configure, and control A/D converters, D/A converters, and other external devices. The connector also provides DSP emulation control pins to help DSP algorithm development on the card using the TI XDS510 emulation system.

Operation

The card's operating mode is controlled via the control, status, and communication registers in the FPGA. Some of these registers are accessible only by the PC, some only by the DSP, and some by both the PC and the DSP. These registers are mapped in the common memory space of the PC and I/O space of the DSP.

The DSP card can operate in two modes, the standard mode and the smart mode. In the standard mode, only the signature register (SIGR) is accessible to the PC. Other registers exist only when the card is in the smart mode.

Table 1. DSP Card Registers

	Memory Address		Access Type		
Register Names	**PCMCIA Common Memory**	**C5x (I/O)**	**PC**	**DSP**	**Register Definitions**
SIGR	000400h	----	R/W	----	Signature register — shadowed in the FPGA
Resewed	000000h		----	----	
DSPCR	000002h	----	R/W	----	PC writes to DSPCR to control and configure the DSP card
DSPSR	000004h	----	R/W	----	DSP status register holds DSP operation and communication status
DSPTXD	000006h	0050h	R	W	PC/DSP communication register — buffers DSP's transmit data
DSPRXD	000008h	0051h	W	R	PC/DSP communication register — buffers data to be received by DSP
Resewed	0Ah–0Fh	----	----	----	
PCSR	----	0052h	----	R	DSP reads status of host communication from this PC status register
BIOR	----	0053h	----	R/W	DSP reads/writes this buffered register to create up to 16 bits of I/O
SYSCFG	----	0054h	----	R/W	DSP selects memory pages and clock speeds by writing to SYSCFG
Resewed	----	055h–058h	----	----	

Standard Mode

In the standard mode of operation, the DSP is reset and the clocks are turned off, disabling the DSP. This reduces the standby power and also gives the PC uncontrolled access to the shared memory. In this default mode, the card appears to the PC and is used by the PC as a standard memory card only. The host can download various communications signal processing (CSP) algorithms to the card without enabling the DSP. The host can also program the flash memory with a DSP initialization code or even a real-time DSP operating system before enabling the DSP. The DSP does not become active until it is specifically made active by the host PC.

Smart Mode

In the smart mode, the communications registers become active and available to the host and DSP. The host continues to have full access to the entire memory on the card. However, when the host PC accesses the shared memory, the DSP operation is temporarily halted because the arbitration logic must put the DSP in a hold condition to give the PC access to the memory bus. Control and communication between the DSP and the PC are implemented via the host communication registers. Although the host PC accesses these registers as regular shared memory, they are physically located in the FPGA. This allows the PC and DSP to access these registers without halting the DSP operation.

Switching Between Standard Mode and Smart Mode

When the PC writes the DSP signature pattern (A320), the DSP is activated and the card is switched from standard mode to smart mode. Once a valid signature is detected, the corresponding bit is set in the DSP control register, DSPCR. Resetting this bit automatically deactivates the DSP and switches the card to standard mode. An alternate method of switching modes is writing to a user-defined register in the PCMCIA attribute memory space.

Memory Organization

The PCMCIA DSP cards provide two separate memory spaces for the common memory and attribute memory. Both memory spaces are accessible by the DSP and the PC. The DSP accesses the common memory in its program and data space and the attribute memory in its global data memory space.

The ability to switch efficiently between various DSP tasks without having to reinitialize or reload is critical for any multifunction communications system. Such a system needs a common memory area that DSP operating systems and the host applications can always access to save system parameters and the operating system itself. Page 0 of the DSP data and program memory is defined to be always active. Thus, DSP operating systems can use page 0 as system memory and additional pages as application-specific memory.

Bus Arbitration

Both the DSP and the PC can access the shared memory on the card. The PC always has higher priority for accessing the memory bus on the card. During PC accesses to the memory bus, the DSP operation is halted. The arbitration logic in the FPGA asserts the $\overline{\text{HOLD}}$ signal to the DSP and extends the PC memory bus access cycle by asserting the $\overline{\text{WAIT}}$ signal. Once the DSP acknowledges the hold by asserting $\overline{\text{HOLDA}}$, the PC $\overline{\text{WAIT}}$ is released and access to shared memory is completed. As soon as the PC completes its access, control of the shared memory is returned to the DSP. Since communication, control, and control registers are not resident in the shared memory, any PC access to these registers will not halt the DSP operation.

Memory Access by the PC

When the PC accesses the shared memory, the DSP is put on hold to grant control of the bus to the PC. The PC's memory access is extended by using the $\overline{\text{WAIT}}$ signal until the DSP puts its bus in the high-impedance state, as indicated by $\overline{\text{HOLDA}}$ signal. There is a time-out if $\overline{\text{HOLDA}}$ is not granted in time. When the card is in smart mode, the PC cannot access the first 16 bytes of the shared memory (also note that the PC cannot access DSP internal memory). This could be used as protected memory for the DSP. PC accesses to this block do not cause the DSP to be put on hold. The PC must load the DSP reset and interrupt vectors and the application algorithm before switching the card into smart mode. Since the PC can access the entire memory on the card without consideration of the DSP page sizes, memory pages not used by the DSP can be dedicated exclusively for the PC.

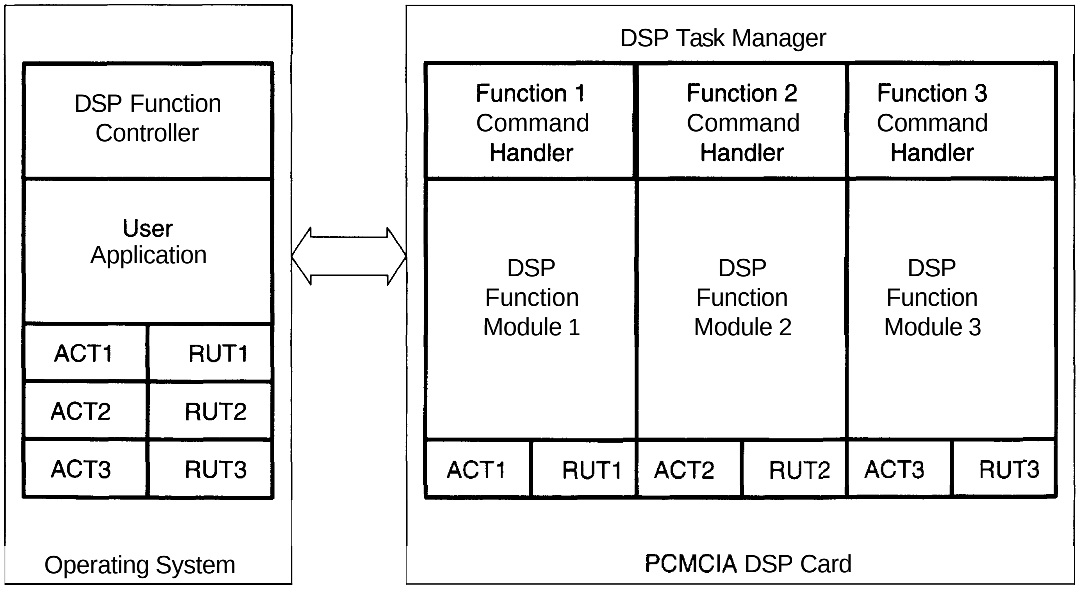
Memory Access by the DSP

The ’C5x versions of the DSP cards can address a maximum of 3M words of common memory. The DSP address range is expanded by using the page selects. Page size for the ’C5x DSP is 32K (x 16) words. Page 0 (both program and data memory for the ’C5x) is always enabled and cannot be deselected via page select bits in the SYSCFG register. This allows DSP operating systems to use this memory without affecting any memory dedicated for DSP applications.

Loading and Executing a Single Algorithm

Initially, the PC loads the desired algorithm to the DSP memory and initializes the DSP. Then the PC enables itself to be interrupted by setting the appropriate enable bits in the DSP control register (DSPCR). This interrupt can be generated by the AFE card (voice activated switch, ring detect, etc.), depending on the application.

Figure 3. Loading and Executing a Single Algorithm


DSP Function Controller
User Application
ACT1
RUT1
ACT2
RUT2
ACT3
RUT3
Operating System
DSP Task Manager
Function 1 Command Handler
Function 2 Command Handler
Function 3 Command Handler
DSP Function Module 1
DSP Function Module 2
DSP Function Module 3
ACT1
RUT1
ACT2
RUT2
ACT3
RUT3
PCMCIA DSP Card

Once the algorithm is loaded and the system is initialized, the host PC can reduce power consumption by turning off the DSP clock, which automatically puts the DSP in a hold condition, placing its buses into the high-impedance state and allowing the PC quicker access to the remaining unused memory on the card.

When the desired external event occurs (indicated by the interrupt), the PC turns the DSP clock on, and the DSP starts executing the algorithm. Since the algorithm is already loaded into DSP memory, there is no delay in loading the algorithm; this makes fast system response time possible.

Note that the code may also be written into global data memory, and the DSP may be bootloaded by the PC to force the DSP to run any preselected default application.

Loading and Executing Multiple Algorithms

First, the host PC initializes the DSP card and loads the DSP operating system. Now the operating system can load multiple DSP algorithms into the DSP's local memory by using the paging scheme. Each 32K-word page could be used for a specific application. Algorithms that require more than 32K words of memory can use multiple pages. Since the paging scheme is needed only for DSPs with a 16-bit address reach, the host PC or other 32-bit DSP, such as the TMS320C3x, can ignore the paging scheme. Also note that a real-time DSP operating system, such as SPOX 2.0, can be loaded into the DSP's on-chip RAM or mask-programmable ROM, freeing the entire external memory for an applications program or data.

The PC and the DSP must follow a predetermined handshake protocol. Commands and data can be passed easily by using the communication registers without halting DSP operation. The DSP operating system controls enabling of DSP program/data pages and transmission on processed data to the PC.

Host Communication

The host PC and the DSP communicate to each other via dedicated host communications registers. These registers are dual-mapped into the common memory space and I/O space of the of the host PC. They are always mapped into the I/O space for the DSP.

The appropriate control and status registers can be programmed to allow an interrupt-based handshake between the host and the DSP. Both the DSP and the host PC can also poll the appropriate bits in the status registers, where interrupts are not available. This could be true in some motherboard applications, where a single integrated device may share the local memory with the host CPU.

Conclusion

With a real-time DSP operating system such as SPOX 2.0, which is small enough to be executed from a DSP's on-chip ROM, and application algorithm modules loaded into the shared memory as needed by the host PC, the PCMCIA DSP card could become the universal communications system for the emerging mobile office environment. With the TEC320 cDSP available today, the same set of CSP software modules could run on a PDA motherboard or PC add-on card if the required analog interface is provided. With such a universal communications platform and standardized user applications interface, the hardware dependencies and porting nightmares should be a thing of the past.

Software Coding Guidelines for ’C5x Developers

Mansoor A. Chishtie
Digital Signal Processing Applications — Semiconductor Group
Texas Instruments Incorporated

Introduction

This report furnishes guidelines to DSP application software developers on how to organize and structure their software to facilitate its maintenance and ease its porting to any custom-defined DSP hardware platform. The model DSP platform used here is a PCMCIA-based 'C5x DSP card with an external connector for an analog interface. (For details on the card, see the preceding report, ***The PCMCIA DSP Card: An All-in-One Communications System.)***

The guidelines in this report should be used in conjunction with the following documents:

- ***TMS320 Fixed-Point DSP Assembly Language Tools User's Guide***
- ***TMS320C2x/C5x Optimizing C Compiler User's Guide***

Hardware Platform Overview

A model DSP hardware platform that will be used as a test and demonstration bed for various DSP applications consists of a PCMCIA type II card with an embedded 25-ns 'C51 digital signal processor and memory. This card complies with the PCMCIA I/O card specifications. This card is capable of running in either standard or smart mode. In standard mode, the DSP is nonfunctional, and the card behaves like any other PCMCIA memory card. The host can switch the card into smart mode by writing a predetermined ***signature*** sequence to a memory location. In smart mode, the embedded DSP is active and executes code from the card memory. Memory available on the first version of this card is 192K words, mapped as multiple 64K pages in data and program spaces.

There are two standard methods for data transfer and command handshake between the host and the DSP: the shared PCMCIA memory and a pair of dual-ported memory-mapped registers. The shared PCMCIA memory, when properly initialized by a PCMCIA card controller, acts like extended memory to the PC memory map. This is the preferred way of transfemng large blocks of code or data to and from the embedded DSP. Note that this mode of access may impose additional time constraints on the real-time execution of an application because the DSP halts while the PC is accessing the shared memory.

Both the host and the DSP can read or write to the dual-ported memory-mapped registers that provide the other host-DSP interface. Access to these registers does not affect the normal operation of the DSP or the host processor. Both sides can poll special bit flags or enable themselves to be interrupted whenever the other side accesses these registers. This register-based communication link is especially suited for sending commands and occasional data parameters to the other end. This feature should be fully utilized by applications to pass results back to the host and let the host apply real-time control functions (such as mode change, start, stop, etc.) to the applications.

For applications that require an analog interface to the outside world, a special connector is provided at the back end of the PCMCIA card; the connector can interface special peripherals to the DSP serial port or bit I/O. Additionally, digital data can be sent over the serial link from an external processor or controller. The connector also supports a TI JTAG emulator (XDS-510) that facilitates application software debug directly on the card.

This hardware platform overview is provided for illustration purposes only. The following discussion is equally applicable to any other 'C5x-based hardware platform.

Software Organization

It is strongly recommended that the following guidelines be observed to organize DSP application software. This will not only result in well-structured code, but it will also make the application easier to port to any other hardware platform.

Organize each software application as a collection of modules or files that belongs to one of the following categories:

- ***Source Modules (*.c, *.asm):*** C or assembly source code files should not define any global constants or macros.

- ***Include Modules (*.h, *.inc):*** All include files for C modules must use file extension *.h, and all such files for assembly modules must use extension *.inc. Include files should define all global constants, macros, or variable types. They should not allocate memory or define functions, because this prevents them from being included by multiple source files. All functions and variables that form part of the overall interface to a *.c or *.asm file should be declared in a *.h or *.inc file. This provides a convenient overview of the interface and allows the compiler or assembler to check for errors.

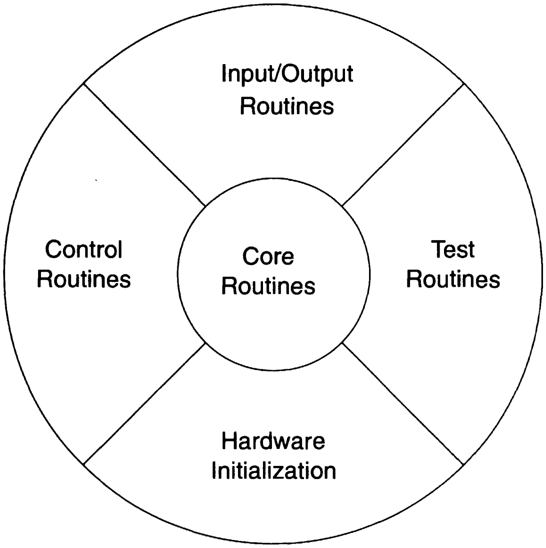
- ***Linker Command File (*.cmd):*** This command file is used by the TI COFF linker to link multiple modules into a single executable COFF output file.

- ***Data Vectors (*.dat):*** These files should contain only data to be used for tests or algorithms. There must not be any code in these data files. These files, if used, will probably be included or copied (.include or .copy directives) in other source files or assembled as stand-alone modules.

- ***Make File (*.mak, *.prj):*** It is strongly recommended that you maintain a project make file that checks for any out-of-date target files and builds them automatically. Note that both Microsoft and Borland make-file utilities use mutually compatible file syntax.

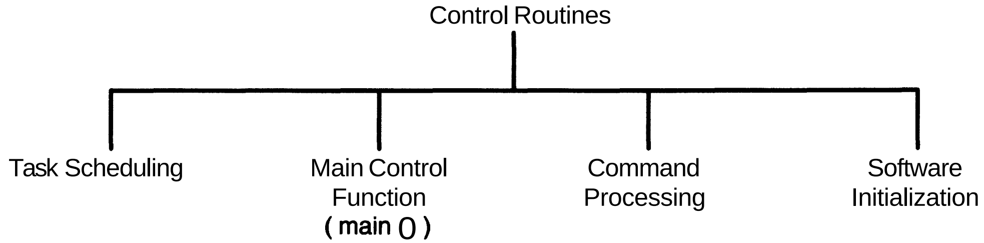
Organize source code files so that each file will fall under one of the categories shown in Figure 1:

Figure 1. Categories of Source Code Files

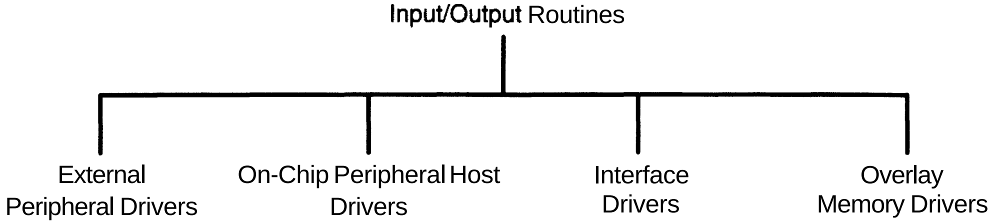


- ***Core Routines:*** Include all software modules that implement the core algorithm. These routines should be independent of hardware-specific implementations. The only target-specific information that these routines should contain is the knowledge of the target DSP processor, in case the modules are in assembly language. Developers of independent applications may want to group these routines into additional categories on the basis of their functionality.

- ***Control Routines:*** These routines consist of all software modules that implement control functions. These control functions may include a C-like main function for program flow control, task handling and scheduling functions, interrupt service routines that pass control to core routines, a command handler that interprets host commands, and routines that initialize variables and tables. Some of these modules may contain some hardware-specific information, but their primary task is to control the program flow. They must ***not*** handle any input/output functions or external peripheral accesses. Note that interrupt service routines (ISRs) that handle on-chip or external peripherals must not be grouped here. The intention is to keep any modifications to these routines at a minimum when the software is ported to a new platform.



- ***Input/Output Routines:*** These routines should handle all the input/output activities of the application, including accesses to any on-chip or external peripherals and I/O ports. As an example, DSP code that handles host communication protocol falls under this category. A serial port ISR and other functions that access an 110-mapped external peripheral also belong to this category. It is recommended that each peripheral driver be arranged as one source file.



- ***Hardware Initialization Routines:*** In general, most nonhardware-specific initialization routines belong to the control routines category. However, since core routines must not have hardware-specific implementations, all functions that initialize external hardware such as external peripherals, host processor, etc., must be grouped separately. Note that these routines will differ from inputloutput routines in that they are invoked only once during system initialization.

- ***Test Routines:*** Application developers should provide a test procedure to verify functionality of their applications. This is especially important when an application is ported (or modified) to a different hardware platform. This test procedure can be in the form of a test program that calls different modules of an application separately to determine their integrity, or it can be in the form of input data vectors that can be processed by the application and output data vectors to be used for verification of the results.

Memory Organization

Proper memory organization is essential for application portability and maintenance. The following guidelines are mandatory:

- Addresses of data variables and tables should not be hard-coded. For example, you cannot use the .set directive to equate a label to an address. This is effectively a form of hard-coded memory allocation because variable addresses are determined during assembly time. The .usect, .sect, and other similar assembler directives should be used to allocate uninitialized and initialized variables. It is recommended that all variable definitions and allocations be done in separate files, as in the following examples:

  ```
  Var_addr    .set    0800h                    **Hard-Coded Addr**
  Var_addr    .usect "Section-name",1          **Addr Def. by Linker**
  ```

  If a peripheral is mapped to a unique address, then this mapping should be clearly identified in the linker command file.

- No assumption should be made about the type of COFF loader available to a host. In many cases, the host would not have access to a smart loader that can autoinitialize global variables during loading (similar to the –c option in the COFF linker). In other cases, an application can be preloaded in nonvolatile memory so that a loader is unnecessary. Therefore, an application should initialize all data variables during system initialization. One side effect of this restriction is that no initialized data can exist in data memory; all initialized tables and variables must be in program memory. They can be later copied to data memory, if necessary, by the software initialization module. This, however, implies that the total code size of an application will become larger than necessary. If the program size is getting unreasonably large because of this restriction, you can choose to ignore this restriction if your system loader can initialize data memory directly. In this case, all initialized data sections must be clearly identified in the linker command file.

- Avoid any restrictions on placement of variables and tables in memory, if possible. Occasionally, an application may require that restrictions be imposed on where a table can be placed in memory. This may happen because 1) a particular DSP feature (for example, bit-reversed addressing) demands it, or 2) it makes an algorithm implementation easier. Any such restriction should be clearly defined in the COFF linker command file in the form of extended comments.

- Global variables and local variables should be defined in separate sections. However, memory can be reused, and local variables of independent functions can occupy the same physical memory space when you use the GROUP and UNION linker directives (see the appendix for a sample linker command file).

- All code and data sections should be mapped to physical addresses during link time. In other words, the *linker command file should be the only module in which absolute addresses are defined.*

- If your application uses overlays or multiple memory pages, you should use the TI COFF linker syntax to define these overlays (see the appendix for an example linker command file). Additionally, you should write a driver module to be a part of the inputloutput routines that will handle the custom-defined memory overlay/page control implementation. This driver module should comply with the following restrictions:
  - The module must be located in on-chip memory. This restriction is intended to guarantee that the DSP will not be accessing off-chip memory when bank-switching occurs.
  - Due to pipelining of instructions by the DSP, the next three instructions following a bank-switch instruction can still access the previous bank. To avoid this, you must make sure that the three instructions immediately following a bank-switch must not access the address range that corresponds to the switched memory bank. Note that if this driver module is called as a subroutine, then a return (RET) instruction immediately after the bank-switch will guarantee that the switch has occurred before the DSP fetches instructions from the new bank:

```
Bank-Switch:  ; bank switch routine
   ...
out *,PA0     ; switch in new memory bank
ret           ; return to new bank
```

Programming Guidelines

- Many DSP applications use mixed-mode (C and assembly) programming techniques to compromise between the need for efficient code and ease of programming. However, in some cases, an application may completely be written in DSP assembly language. In such cases, it is highly recommended that at least a dummy C main() function be written that simply transfers control to an assembly function. In this way, a basic C environment is automatically set up by main(), which leads to easier integration of any C functions in the future. If main() is the only C function in an application, then the rest of the functions need not adhere to C calling conventions.

- Many mixed-mode applications strictly follow the C convention for function calls, parameter passing, and variable allocation. However, you may need to avoid these constraints to efficiently implement some assembly-level functions. All such exceptions must be clearly identified and described in corresponding documentation. In some cases, when an assembly language function is called only by other assembly functions, context is not maintained across the function calls. These functions, although legal, must be clearly identified as non-C-callable functions to avoid any future maintenance problems.

- Self-modifying code should not be written. Such code is commonly used in interrupt vector tables (IVT), where one ISR can be patched for another during runtime. You can avoid this by using a software semaphore in ISR or by using an LAMM/BACC sequence to replace a more conventional B address sequence in IVT. The following interrupt vector table code example illustrates the use of an LAMM/BACC instruction to fetch the address of an ISR from a data memory location (in data page 0):

```
INT1:  lamm   INT1_Addr
       bacc
```

- For relocatable sections of code, do not use the .asect directive. Instead, use the runtime and load-time address options of the TI linker. This emphasizes our strategy of not allowing absolute addresses in assembly modules. Note that the .asect directive requires an absolute address to be specified as a parameter.

- Avoid using numerical constants as instruction parameters. Code listings are more readable when constants are replaced by meaningful labels. You can do this with the .set directive, as shown in the following example:

  replace:

  ```
  add       #07FFFh
  ```

  with:

  ```
  One-Q15   .set    07FFFh
  add       #One-Q15
  ```

Source Code Documentation

- All source modules, whether in assembly or C, must maintain a modification history table that lists the date and time of each modification in chronological order, the person who made the change, and a brief description of the change.

- Line-by-line comments are highly recommended, especially for assembly language modules. All functions in a module, whether assembly or C, must clearly describe the implementation-specific details of the function.

- All functions should be preceded by a function header that gives the function description. input and output parameter lists, global variables used, a list of nested function calls, a list of functions that can call this function, and entry/exit conditions. Note that entry and exit conditions are especially important for assembly functions because processor context is not often maintained across function calls.

Appendix: A Sample Linker Command File for the 'C5x Card

The following linker command file is listed here to illustrate how to use the TI COFF linker syntax to define overlays and multiple code/data pages for the 'C5x PCMCIA card version 1.0. This command file would require minimum modifications to adapt to any 'C5x application running on this PCMCIA card.

```
f1.obj f2.obj f3.obj f4.obj f5.obj f6.obj

-o f.out

-m f.map

/*************************************************************************

PCMCIA 'C5x Card Memory Map: version 1.0

At reset, page 0 is in the 'C5x program space and page 1 is in data space.

If page 2 is enabled, it is dual-mapped in both program and data spaces. Each
application must carefully divide page 2 into two or more sections, and each
section must be considered as either program or data, but not both. In the
following example, the PRAM2 section is mapped as program, and the RAMEXT2
section is mapped as data, but this can be modified by an application.

The RAMSA and RAMDA memory blocks (in both page 0 and page 1) are defined as
overlays. This means that runtime addresses of multiple code and data
sections can be bound to these overlay sections. Note, however, that you must
copy any initialized section to an overlay area before it can be used.

All pages are 64K words in length.

           Program                    Data                     Program/Data
                              0000 ---------------       0000 ---------------
0000 ---------------               | XXXXXXXXXX  |            |             |
     |             |          0060 |-------------|            |             |
     |             |               |    RAMB2    |            |    PRAM2    |
     |    PRAM     |          007F |-------------|            |             |
     |             |               | XXXXXXXXXX  |            |             |
     |             |          0100 |-------------|            |             |
2000 |-------------|               |    RAMDA    |            |             |
     |    RAMSA    |          0500 |-------------|       2000 |-------------|
2400 |-------------|               | XXXXXXXXXX  |            |             |
     |             |          0800 |-------------|            |  RAMTEXT2   |
     |    RAMEXT   |               |    RAMSA    |            |             |
     |             |          0C00 |-------------|            |             |
FE00 |-------------|               |    RAMEXT   |            |             |
     |    RAMDA    |          FFFF ---------------       FFFF ---------------
FFFF ---------------                   Page 1                    Page 2
          Page 0

*************************************************************************/
MEMORY


    page 0 : /* Program Only */
        PRAM  : origin = 00000h,  length = 02000h
        RAMSA : origin = 02000h,  length = 00400h  /* Overlay Section */
        RAMEXT: origin = 02400h,  length = 0DA00h
        RAMDA : origin = 0FE00h,  length = 00200h  /* Overlay Section */
    page 1 : /* Data Only */
```

```
        RAMB2 : origin = 00060h,  length = 00020h
        RAMDA : origin = 00100h,  length = 00400h  /* Overlay Section */
        RAMSA : origin = 00800h,  length = 00400h  /* Overlay Section */
        RAMEXT: origin = 00C00h,  length = 0F400h
    page 2 : /* Dual-Mapped in Program and Data */
        PRAM2  :origin = 00000h,  length = 02000h  /* Contains Code  */
        RAMEXT2:origin = 02000h,  length = 0E000h  /* Contains Data  */
}
SECTIONS
{
      PROG1:  load = PRAM     page 0
      {
              f1.obj(.text)
      }
      PROG2:  load = PRAM2    page 2

              f2.obj(.text)
      }
      DATA1:  load = RAMEXT   page 1
      {
              f1.obj(.data)
      }
      DATA2:  load = RAMEXT2  page 2
      {
              f2.obj(.data)
      }
      UNION : run  = RAMSA    page 0          /* Overlay Section:     */
      {                                       /* f3 and f4 functions  */
              .text1 : load = RAMEXT page 0   /* will be copied and   */
              {                               /* run from RAMSA page0 */
                      f3.obj(.text)
              }
              .text2  : load = RAMEXT page 0
              {
                      f4.obj(.text)
              }
      }
      UNION : run  = RAMDA    page 0          /* Overlay Section:     */
      {                                       /* f5 and f6 functions  */
```

```
            .text3  : load = RAMEXT page 0    /* will be copied and   */
            {                                 /* run from RAMDA page0 */
                    f5.obj(.text)
            }
            .text4  : load = PRAM    page 0
            {
                    f6.obj(.text)
            }
        }
        UNION : run  = RAMSA    page 1        /* Overlay Section:     */
        {                                     /* local variables of   */
            .bss1  :                          /* f3 and f4 functions  */
            {                                 /* overlay each other   */
                    f3.obj(.bss)              /* in RAMSA page 1      */
            }
            .bss2  :
            {
                    f4.obj(.bss)
            }
        }
        UNION : run  = RAMDA    page 1        /* Overlay Section:     */
        {                                     /* local variables of   */
            .bss3  :                          /* f5 and f6 functions  */
            {                                 /* overlay each other   */
                    f5.obj(.bss)              /* in RAMDA page 1      */
            }
            .bss4  :
            {
                    f6.obj(.bss)
            }
        }
}
```

TCM320AC3x/4x
Voice-Band Audio Processors

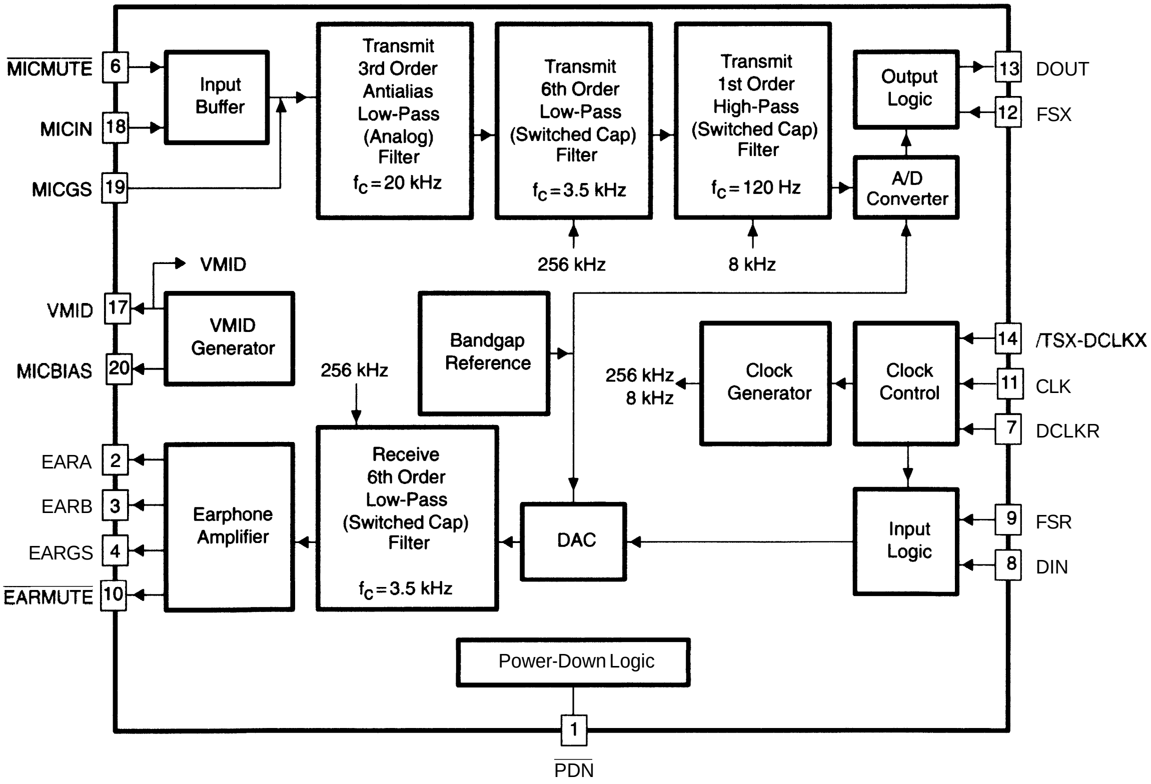
Greg Davis
Russ MacDonald
Advanced Linear Applications — Semiconductor Group
Texas Instruments Incorporated

Introduction

The voice-band audio processor (VBAP) family of devices is a line of highly specialized single-supply voice codecs specifically designed for use in battery-powered personal communications systems. The VBAP uses the TI LinEPICZ1 1-ym semiconductor process, which results in very low power consumption. In addition, a patented TI process is used to maintain extremely low noise specifications. The VBAP device serves as an interface between a voice and a DSP and incorporates three major functions: transmit encoding (A/D conversion), receive decoding (D/A conversion), and transmit and receive filtering. The VBAP family supports a serial data connection in either 8-bit companded y-Law or A-law mode, and a pin-selectable 13-bit linear conversion mode. The VBAP utilizes sophisticated switched capacitor filters to provide filtering that is compatible with most personal communication specifications, including the EIA/TIA/IS-54 for U.S. digital cellular telephones and the CCITT G.711 and G.712 y-law and A-law filtering requirements. The VBAP also provides direct microphone and speaker interface.

VBAP devices are available in 20-pin N (dual in-line plastic) and DW (surface mount) packages, as well as soon-to-be-introduced QFP (quad flat pack <20-mm) packages.

Figure 1. VBAP Functional Block Diagram


MICMUTE
6
MICIN
18
MICGS
19
Input Buffer
Transmit 3rd Order Antialias Low-Pass (Analog) Filter
f_C = 20 kHz
Transmit 6th Order Low-Pass (Switched Cap) Filter
f_C = 3.5 kHz
Transmit 1st Order High-Pass (Switched Cap) Filter
f_C = 120 Hz
Output Logic
A/D Converter
13
DOUT
12
FSX
256 kHz
8 kHz
VMID
VMID
17
VMID Generator
MICBIAS
20
Bandgap Reference
256 kHz
256 kHz
8 kHz
Clock Generator
Clock Control
14
/TSX-DCLKX
11
CLK
7
DCLKR
EARA
2
EARB
3
EARGS
4
EARMUTE
10
Earphone Amplifier
Receive 6th Order Low-Pass (Switched Cap) Filter
f_C = 3.5 kHz
DAC
Input Logic
9
FSR
8
DIN
Power-Down Logic
1
PDN

NOTE: f_C is the –3-dB cutoff frequency.

Principles of Operation

To minimize crosstalk, the VBAP design utilizes independent converters, filters, and voltage references for the transmit and receive channels. Figure 1 shows a typical VBAP functional diagram with these features.

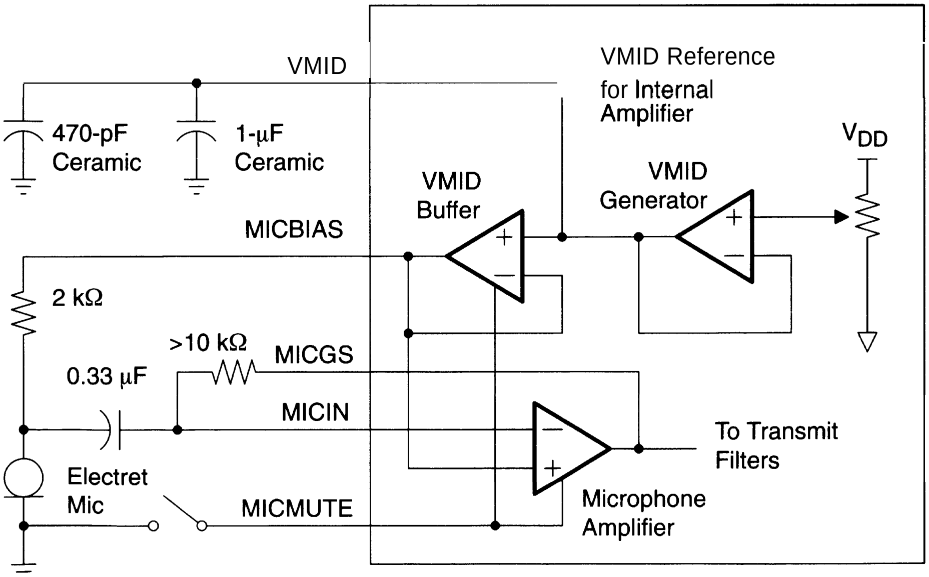
Transmit Channel

Microphone interface

A reference voltage equal to $V_{DD}/2$, called VMID, is used to develop the midlevel virtual ground for all amplifier circuits and the microphone bias circuits. Any power supply noise on VMID would normally be detected on the output of the VBAP; therefore, VMID is brought to an external pin so that the voltage can be filtered by using an external capacitor. The optimum capacitor combination is a 1-µF ceramic type in parallel with a 470-pF ceramic chip cap. A reference voltage at the MICBIAS pin can be used to supply bias current for the microphone. Because MICBIAS is also used internally to bias the microphone amplifier, the common-mode rejection results in a quiet bias voltage. For this reason, it is recommended that you use MICBIAS to bias only an electret microphone, as shown in Figure 2.

The microphone input signal (MICIN) is buffered and amplified with provision for setting the amplifier gain to accommodate a range of signal input levels. This is accomplished by changing the value of the series capacitor and feedback resistor of the (uncommitted) microphone-inverting amplifier. While the configuration shown in Figure 2 will suit most applications, the steady-state impedance of the electret microphone and the 2-kΩ microphone bias resistor can be converted to a Thevenin's equivalent voltage source with a series impedance to calculate the microphone amplifier gain. A resistor can also be added in series with the 0.33-µF capacitor at the amplifier input to decrease the amplifier gain. Note that the 0.33-µF capacitor, along with the 2-kΩ resistor, yields a high-pass filter with a –3-dB cutoff of 240 Hz and –0.6 dB cutoff at 300 Hz, which is acceptable for voice-band communications.

Figure 2. VBAP Microphone Connection



Microphone Mute

The microphone mute function disables the microphone amplifier, and the input to the transmit filters is placed in a high-impedance state. With $\overline{\text{MICMUTE}}$ enabled, the output of the microphone amplifier is more than 80 dB down from the signal on MICIN (microphone input), and the digital circuitry will transmit zero code on DOUT. In addition, the VMID buffer is disabled, and the MICBIAS output is zero.

Transmit Filters

The amplified signal is passed through antialiasing and band-pass filters. The antialiasing filter is an analog (continuous time) first-order low-pass filter with a cutoff of 20 kHz and is used to attenuate any modulation components above half the sampling frequency *of the next stage* to avoid aliasing artifacts (Nyquist sampling theorem). The next stage is a switched capacitor filter with a sampling rate of 256 kHz, so the antialiasing filter provides a greater than 35-dB attenuation at half that sampling frequency, or 128 kHz.

The band-pass filters are composed of oversampled switched capacitor filters to avoid the effects of aliasing. The first band-pass filter is a sixth-order low-pass filter with a cutoff of 3.5 kHz, and the second is a first-order high-pass filter with a cutoff of 100 Hz, sampled at 256 kHz and 8 kHz, correspondingly. *The effective 0-dB bandpass of these filters is from 300 Hz to 3.4 kHz.* Because of the oversampling and because the clocks used by both these filters are synchronous, antialiasing products can be easily controlled and virtually eliminated.

Encoding (A/D Conversion)

The encoded data word structure is available in two formats: companded and linear conversion. The formats are pin selectable. When the device is in the companded mode, the analog signal is sent to the transmit filters and then input to a compressing analog-to-digital converter (COADC). The analog signal is encoded into 8-bit digital representation via the μ-Law and A-Law encoding scheme according to CCITT G.711; this equates to 12 bits of resolution for low-amplitude signals. When the linear conversion mode is selected, 13 bits of data are sent, padded with 0s to provide a 16-bit word. Both companded and linear conversion modes use 2s-complement words.

Data can be transmitted in either a fixed or variable data rate mode. See *Fixed and Variable Data Rate Modes* on page 266 for more detail.

The encoder internally samples the output of the transmit filter *at the middle of the frame* and holds each sample on an internal sample-and-hold capacitor. The encoder performs an analog-to-digital conversion (on a switched capacitor array), also starting in the *second half of the frame*. To minimize the delay across the VBAP, the actual conversion process does not complete until just *before* the next frame. Digital data representing the sample is then transmitted at the start of the *next* frame. The transmit data is output on the DOUT pin. Transmit data is clocked out on consecutive *positive* transitions of the transmit data clock, which is CLK in the fixed-data-rate mode and DCLKR in the variable-date-rate mode.

The master-clock-to-frame-sync ratio is critical and cannot be violated. Refer to *Timing and Clocking* on page 265 for more detail.

For both companded and linear modes, the sign bit is transmitted first, followed by the MSB, with the LSB transmitted last.

Since the A/D conversion rate is the master clock, and the band-pass switched capacitor filter clocks are integer submultiples of the master clock, unwanted aliasing products are prevented.

Transmit Auto Zero

The auto zero circuit corrects for any DC offset on the input signal to the encoder by using a sign-bit averaging technique. The sign bit from the encoder output is long-term averaged and subtracted from the input to the encoder. This acts as a form of feedback to track and correct for changing DC offsets. The auto zero circuitry is implemented after the high-pass transmit filter so that it will not mistakenly track low-frequency audio signals. The response time of the auto zero circuitry is about five frames from device power-up, or from standby to active.

Noise-Reduction Algorithm

The VBAP transmit circuitry incorporates patented TI circuits to reduce transmit noise to extremely low levels. These circuits reduce the transmit audio when the analog input falls below a set level; they are used in the companded mode *only*. The levels at which the noise reduction circuits are enabled include hysteresis for further improved performance; these levels are about –55 and –60 dB. When the VBAP detects these low audio input conditions, it puts out a zero code (1111 1111 in μ-Law and 0101 0101 in A-Law, according to CCITT G.711 specifications). This is different from the normal output under idle channel noise conditions, which typically consists of a random sequence of codes around 0 (LSB and/or second LSB and MSB sign bit toggling arbitrarily).

Receive Channel

Decoding (D/A Conversion)

Data can also be received in either a fixed or variable data rate mode. See *Fixed- and Variable-Data-Rate Modes* on page 266 for more detail.

In the companding modes, the serial data word is received at DIN on the first eight clock cycles in the fixed-data-rate mode or the last eight clock cycles in the variable-data-rate mode. The decoding section converts the 8-bit PCM data into an analog signal with 12 bits of dynamic range, according to CCITT G.711 specifications. In the linear mode, the serial data word is received in the first 13 clock cycles. In both the companded and linear modes, input data is clocked in on consecutive *negative* transitions of the receive clock, which is CLK in the fixed-data-rate mode and DCLKR in the variable-date-rate mode. Digital-to-analog conversion is performed, and the corresponding analog sample is held on an internal sample-and-hold capacitor. The sample is then transferred to the receive filter during the next frame.

Receive Filters

The receive filter is a switched capacitor sixth-order low-pass filter with a cutoff of 20 kHz; it provides pass-band flatness and stop-band rejection that fulfills both the AT&T D3/D4 specifications and the CCITT recommendation for G.712. The filter also contains the (sinx)/x correction response of such decoders.

Receive Buffer/Volume Control

The receive buffer contains the volume control circuitry. When data is received in the linear mode, the 13 bits are read as data, and the remaining 3 bits are used as programmable volume control of the analog output. These volume control bits originate from a DSP or other device that is interfaced with the VBAP, and they serve to attenuate the speaker output of the VBAP in seven 3-dB steps. The volume control bits are *not* latched into the VBAP, so they must be present in each received data word. If they are missing, the VBAP circuitry will assume that the three volume control bits are 0 (0-dB attenuation). In the companded mode, programmable gain is not used. Table 1 illustrates the volume control bits required for a given attenuation.

Table 1. Receive-Channel Volume-Control Bits

Bits 14–16 in DIN Input Data Stream	Resulting Receive Channel Attenuation
000	0 dB
001	-3 dB
010	-6 dB
011	-9 dB
100	-12 dB
101	-15 dB
110	-18 dB
111	-21 dB

NOTE: The first bit is the MSB.

Speaker Amplifier Overview

The VBAP incorporates an analog output power amplifier. This amplifier can drive transformer hybrids or low-impedance loads directly in either a differential or single-ended configuration. In addition, the VBAP speaker output stage (in its differential configuration) allows for further volume control (in addition to the volume control bits), by connection of a resistor chain to the output terminal of the device.

The speaker amplifier output will typically assume a DC offset of approximately 40 mV. This is a normal consequence of using switched capacitors in the VBAP design. Potential biasing problems can be avoided by the use of an AC coupling capacitor.

Timing and Clocking

Master Clock and Frame Sync

The VBAP requires a master clock and frame sync. The master clock is used for many internal functions, most notably to clock the switched capacitor filters and the A/D-D/A conversion process in both the transmit and receive directions. The VBAP family (TCM320ACxx) accommodates a variety of master clock frequencies, as shown in Table 2.

Table 2. VBAP Master Clock Frequencies

Device Suffix (xx)	Master Clock (MHz)
36,37,46	2.048
39	2.6
41	1.152
42	1.944
44	1.536

Power-Down and Standby Operations

To minimize power consumption, a power-down mode and three standby modes are provided.

For power-down, an external low signal is applied to PDN. In the absence of a signal, PDN is internally pulled up to a high logic level, and the device remains active. In the power-down mode, the average power consumption is reduced to 1.25 mW.

The standby modes give you the option of putting the entire device on standby or putting only the transmit or receive channels on standby. The standby modes are entered by removing one or both of the frame syncs. Table 3 illustrates all VBAP modes of operation.

Table 3. Power-Down and Standby Procedures

Device Status	Procedure	Typical Power Consumption	Digital Output Status
Power on	PDN = high FSX = pulses FSR = pulses	40 mW	Active
Power down	PDN = low FSX/FSR = X/X	1.25 mW	TSX and DOUT in a high-impedance state
Entire device on standby	FSX = low FSR = low PDN = high	5 mW	TSX and DOUT in a high-impedance state
Receive only (transmit standby)	FSX = low FSR = pulses PDN = high	20 mW	TSX and DOUT in a high-impedance state within 5 frames
Transmit only (receive standby)	FSR = low FSX = pulses PDN = high	20 mW	Active

Fixed- and Variable-Data-Rate Modes

The VBAP is designed to operate in both the fixed and variable-data-rate modes. The mode of operation is pin selectable. In the fixed mode, the data is transmitted (or burst) and received at the rate of the master clock frequency and is sampled every frame. In the variable-data-rate mode, the data is transmitted or received at a rate slower than the master clock frequency and uses the data clock input DCLKR.

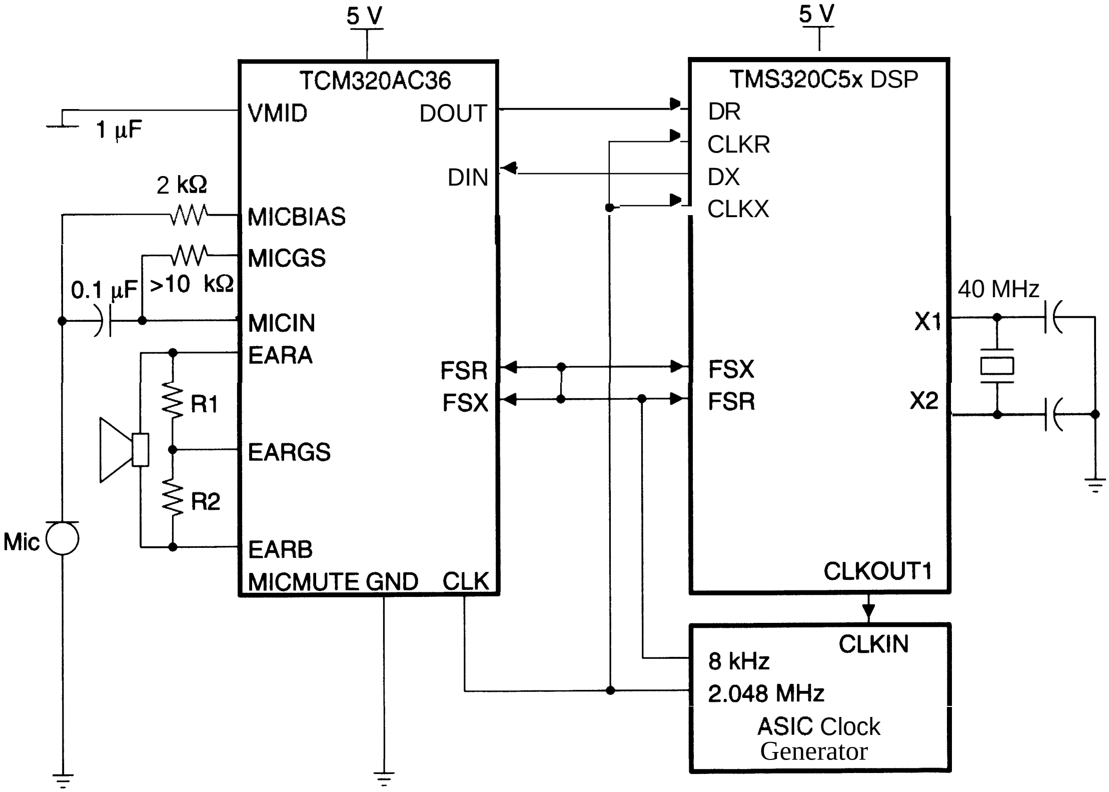
For example, suppose you are using the TCM320AC36 VBAP in the 8-bit companded mode and variable-data-rate configuration. This VBAP has a master clock frequency of 2.048 MHz and must use a frame sync of 8 kHz to maintain a 256 master-clock-to-frame-sync ratio. The data is sampled every 125 µs, but the speed at which the data is transmitted (or burst) and received, each 125 µs, can vary from 2.048 MHz to 64 kHz. Notice that the slowest speed of the data clock is 64 kHz; any slower speed would not allow a full 8-bit sample to be performed before the next frame begins. At 64 kHz, the complete frame is used to transmit or receive the data (8 bits × 8000 = 64 kbps). Likewise, the minimum variable-data-rate speed for the 16-bit linear mode would be 128 kHz (16 kHz X 8000).

Application Information

VBAP interfaced to a DSP

The most common application for the VBAP is as an interface to a DSP. The VBAP performs the analog-to-digital and digital-to-analog conversions, along with filtering, while the DSP performs more complex functions with the encoded speech. For example, in a cellular telephone application, the DSP would typically perform equalization and speech coding through the use of algorithms (code) executed by the DSP. The circuit in Figure 3 illustrates a typical VBAP-to-DSP interface.

Figure 3. VBAP Interfaced to a 'C5x DSP


5 V
TCM320AC36
VMID
DOUT
1 µF
DIN
2 kΩ
MICBIAS
MICGS
0.1 µF
>10 kΩ
MICIN
EARA
FSR
FSX
R1
EARGS
R2
Mic
EARB
MICMUTE GND CLK
5 V
TMS320C5x DSP
DR
CLKR
DX
CLKX
40 MHz
X1
FSX
FSR
X2
CLKOUT1
CLKIN
8 kHz
2.048 MHz
ASIC Clock Generator

Device Power-Up Sequence

The VBAP should be powered up and initialized as follows:

1. Apply GND
2. Apply V_{DD}
3. Apply low to PDN bar
4. Connect master clock
5. Connect data clock (if used)
6. Remove low to PDN bar
7. Apply FSX and/or FSR synchronization pulses

Grounding and Decoupling

Use a ground plane on the PCB, covering as much unused area as possible.

Bypass the VBAP with a high-quality 0.1-µF ceramic capacitor (such as a CK05) *directly across the VBAP power supply pins.* Ceramic capacitors have a low ESR (equivalent series resistance) or high Q; they are able to react to fast changes in voltage and are used to suppress high-frequency transients. High-frequency voltage transients result from instantaneous high current consumption during digital device switching. Since all power supplies have an internal impedance that prevents infinite current sourcing, power supply voltage ripple, or noise, will result. Capacitive loading on the power supply rail regulates the voltage of the supply.

Any power supply noise on VMID would normally be detected on the output of the VBAP; therefore, VMID is brought to an external pin so that the voltage can be filtered by using an external capacitor. The optimum capacitor combination is a 1-µF ceramic in parallel with a 470-pF ceramic chip cap.

Power Supply

A voltage regulator should always be used, even with battery power. Batteries in particular have a high internal impedance that allows the DC voltage to vary under instantaneous current consumption during digital switching. The resultant change in voltage manifests itself as noise on the power supply rail.

Use a 10-µF capacitor across the power supply rails on the PCB. This serves the same purpose as the ceramic capacitor, except that it responds well to lower frequency transients.

All power supply traces should be as close as possible to the ground plane. Proximity to the ground plane adds parallel capacitance.

Variable Data Rate at Master Clock Frequency

In some applications, it is desirable to run the VBAP in the variable-data-rate mode at a data rate equal to the master clock speed. This gives you the advantage of using the variable-data-rate mode (as with repeated data while frame sync is high) while still running the maximum data rate as in the fixed-data-rate mode (in fixed-data-rate mode, the data clock is internally run at the master clock speed).

If the device is operated in the variable-data-rate mode with the data clock run at the master clock frequency, the DCLKX and MCLK pins cannot be directly connected externally. If you choose to use the master clock as the DCLKX, you must buffer the output of the master clock before connecting it to DCLKX. This is necessary because the VBAP always powers up in the fixed-data-rate mode, and for the first several clock cycles, the DCLKX pin is actually an output ($\overline{\text{TSK}}$) as defined in the data sheet. The $\overline{\text{TSX}}$ output is a transmit time strobe that will pull the MCLK pin low; this will corrupt the MCLK input, if MCLK and DCLKX are directly connected externally to the device. Only after the first several master clock cycles does the device assume a fixed-data-rate mode and the DCLKX pin become an input. Therefore, the suggested method is to join MCLK and DCLKX *before* a buffering stage for the DCLKX line.

Typical PCM Output Expected From a Transmit VBAP

In an ideal situation, the 8- (and 13-bit) A/D converter in the VBAP is designed with a noise floor that equates to the transition of half the LSB. In the linear mode, a half bit represents approximately –75 dB, as shown below:

$$20 \times \log\left[\frac{2^{0.5}}{2^{13}}\right] = -75\ \text{dB} \tag{1}$$

This corresponds to the VBAP data sheet, which specifies the transmit noise in linear mode to be –74 dB. Therefore, using a VBAP in the receive mode, configured for a maximum output signal of 4 volts peak-to-peak (V_{P-P} which is equal to 1.414 V_{rms}), the VBAP would encode this half bit of noise and experience about 250 μV_{rms} of noise on the speaker output terminals, as in this equation:

$$-75\ \text{dB} = 20 \times \log \frac{X}{1.414 V_{rms}} \tag{2}$$

where X = output that is 75 dB down from 1.414 V_{rms} (that is, X = 250 μV_{rms}).

Part IX
Bibliography

Bibliography

TMS320 Bibliography

Since TMS32010 was disclosed in 1982, the TMS320 family has received ever-increasing recognition. The number of outside parties contributing to the extensive development support offered by Texas Instruments has grown significantly. Many technical articles are being written about TMS320 applications in the field of digital signal processing.

To keep TMS320 designers aware of new applications and developments related to the TMS320 DSPs, Texas Instruments has published extensive bibliographies of TMS320-related conference papers and technical articles in the *Digital Signal Processing Applications with the TMS320 Family,* Volumes 1, 2, and 3 and in *Digital Control Applications with the TMS320 Family*. The following TMS320 bibliography serves as an extension of the previously published bibliographies. It lists only those papers and articles that are generally related to telecommunication applications. For additional papers on this subject, please refer to the appropriate sections of the above-mentioned bibliographies. Readers who are interested in gaining further information about these applications may obtain copies of these articles/papers from their local or university library.

The articles are organized into the following six categories:

1. Mobile Radio Systems
2. Modulation and Demodulation
3. Equalization, Channel Estimation, and Adaptive Filtering
4. Speech Recognition
5. Speech Commpression
6. System Design Considerations

Mobile Radio Systems

1. An, J.F., Turkmani, A.M.D., and Parsons, J.D., "Implementation of a DSP-Based Frequency Non-Selective Fading Simulator", *Fifth International Conference on Radio Receivers and Associated Systems*, Conference Publication No. 325, 1990, pp. 20–24.

2. Cullen, P. J., Fannin, P. C., and Molina, A., "Wide-band Measurement and Analysis Techniques for the Mobile Radio Channel", *IEEE Transactions on Vehicular Technology,* Volume 42, No. 4, November 1993, pp. 589–603.

3. Leung, P. S.K., and Zhu, M., "A Simple DSP Rayleigh Fading Simulator for Mobile Radio", IREECON '91, *Australia's Electronics Convention Proceedings,* Volume 1, 1992, pp. 49–52.

4. Lim, M.S., and Park, H.K., "The Implementation of the Mobile Channel Simulator in the Baseband and Its Application to the Quadrature Type GMSK Modem Design", *40th IEEE Vehicular Technology Conference: On the Move in the 90's*, IEEE Cat. No. 90CH2846-4, 1990, pp. 496–500.

5. Peterson, B., Gross, K., Chamberlin, E., Montague, T., and Jones, W., "Integrated CIS VLF/Omega Receiver Design", *IEEE Aerospace and Electronics Systems Magazine,* Volume 8, No. 1, January 1993, pp. 9–20.

6. Wu Bo, Y.Y., and Wang, Jing, "A Direct Conversion Transceiver for GMSK in Digital Mobile Communications", *Proceedings of 1992 International Conference on Communication Technology,* Volume 2, 1992, pp. 28.05/1–4.

Modulation and Demodulation

1. Boudreau, D., "2400 BPS TMS 2010 Modem Implementation for Mobile Satellite Applications", *Proceedings of the Thirteenth Biennial Symposium on Communications,* 1986, pp. B3/1–4.
2. Gott, G.F., Darbyshire, E.P. , Brydon, A.N., Oodit, B.P. , and Rubenstein, R.H., "Robust Slow and Medium Rate Data Transmission", *Fifth International Conference on HF Radio Systems and Techniques,* Conference Publication No. 339, 1991, pp. 212–216.
3. Perl, J.M., Bar, A., and Cohen, J., "TMS-320 Implementation of a 2400 bps V.26 Modem", *Signal Processing III: Theories and Applications. Proceedings of EUSIPCO—86: Third European Signal Processing Conference,* Volume 2, 1986, pp. 1121–1124.
4. Tavares, G., Henriques, J., Piedade, M.S., Goncalves, V., Costa, T., and Gerald, J., "High Speed Data Modem Implementation Using the TMS320C25”, *1990 IEEE International Symposium* on *Circuits and Systems,* IEEE Cat. No. 90CH2868-8, Volume 4, 1990, pp. 2889–2892.
5. Yim, W.H., Kwan, C.C.D., Coakley, F.P., and Evans, B.G., "On-Board Multicarrier Demodulator for Mobile Applications Using DSP Implementation", *Space Communications,* Volume 7, No. 4–6, November 1990, pp. 543–548.

Equalization, Channel Estimation, and Adaptive Filtering

1. Ahmed, I., and Lovrich, A., "Adaptive Line Enhancer Using the TMS320C25”, Northcon/86, *Conference Record,* 1986, pp. 141311–10.
2. Bateman, S.C., and Hale, R.G., "Real Time Implementation of Adaptive Filter Algorithms for High-Speed Data Transmission", 1989.
3. Clarkson, P. M., and Dokic, M.V., "Real-Time Adaptive Filters for Time-Delay Estimation", *Proceedings of the 32nd Midwest Symposium on Circuits and Systems,* IEEE Cat. No. 89CH2785-4, Volume 2, 1990, pp. 891–894.
4. Er, M.H., Ooi, T.H., Li, L.S., and Liew, C.J., "A DSP-based Acoustic Feedback Canceler for Public Address Systems", *Microprocessors and Microsystems,* Volume 18, No. 1, January 1994, pp. 39-47.
5. Eriksson. L.J., and Allie, M.C., “System Considerations for Adaptive Modelling Applied to Active Noise Control", *1988 IEEE International Symposium on Circuits and Systems: Proceedings,* IEEE Cat. No. 88CH2458-8, Volume 3, 1988, pp. 2387–2390.
6. Kang, G.S., and Fransen, L.J., "Experimentation with An Adaptive Noise-Cancelation Filter", *IEEE Transactions on Circuits and Systems,* Volume CAS-34, No. 7, July 1987, pp. 753–758.

Equalization, Channel Estimation, and Adaptive Filtering (Continued)

7. Kloos, M.N., and Jenkins, W.K., "The Investigation of New Adaptive Filtering Algorithms for Telecommunications Echo Cancelation Implemented in TMS32010 Fixed-Point Assembly Code", *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*, IEEE Cat. No. 90CH2819-1, Volume 2, 1991, pp. 1034–1037.

8. Lal Sharma, P., “Implementation of Adaptive Recursive Echo Canceler Using the TMS320C25 Digital Signal Processor", *Proceedings of the 34th Midwest Symposium on Circuits and Systems*, IEEE Cat. No. 91CH3143-5, Volume 1, 1992, pp. 494–496.

9. Mirchandani, G., Gaus, R.C., Jr., and Bechtel, L.K., “Performance Characteristics of a Hardware Implementation of the Cross-Talk Resistant Adaptive Noise Canceler", *ICASSP 86 Proceedings*, International Conference on Acoustics, Speech and Signal Processing, IEEE Cat. No. 86CH2243-4, Volume 1, 1986, pp. 93–96.

10. Muller, G.S., and Pauw, C.K., "Acoustic Noise Cancellation", *ICASSP 86 Proceedings*, International Conference on Acoustics, Speech and Signal Processing IEEE Cat. No. 86CH2243-4, Volume 2, 1986, pp. 913–916.

11. Yeh, H., "Adaptive Noise Cancellation for Speech With a TMS32020”, *Proceedings: ICASSP 87*, 1987 International Conference on Acoustics, Speech, and Signal Processing, IEEE Cat. No. 87CH2396-0, Volume 2, 1987, pp. 1171–1174.

12. Young, M.C.S., Grant, P. M., and Cowan, C.F.N., "Block LMS Adaptive Equaliser Design for Digital Radio", *Signal Processing IV: Theories and Applications. Proceedings of EUSIPCO-88*, Fourth European Signal Processing Conference, Volume 3, 1988, pp. 1349–1352.

13. Zhuang, J.D., Zhu, X.L., and Lu, D.J., "Design and Implementation of a Programmable Blind Equalizer for High-Speed Multilevel Data Transmission", *IEEE TENCON '90: 1990 IEEE Region 10 Conference on Computer and Communication Systems*, Cat. No. 90CH2866-2, Volume 2, 1990, pp. 701–704.

Speech Recognition

1. Aktas, A., and Zunkler, K., "Speaker-Independent Continuous HMM-Based Recognition of Isolated Words On A Real-Time Multi-DSP System", *EUROSPEECH 91. 2nd European Conference on Speech Communication and Technology Proceedings*, Volume 3, 1991, pp. 1345–1348.

2. Attili, J.B., Savic, M., and Campbell, J.P. Jr., "A TMS32020-based Real Time, Text-Independent, Automatic Speaker Verification System", *ICASSP 88: 1988 International Conference on Acoustics, Speech, and Signal Processing*, IEEE Cat. No. 88CH2561-9, Volume 1, 1988, pp. 599–602.

Speech Recognition (Continued)

3. Ciaramella, A., and Venuti, G., "Dynamic Programming With Hidden Markov Models on a TMS32020 Digital Signal Processor", *Signal Processing IV: Theories and Applications. Proceedings of EUSIPCO-88: Fourth European Signal Processing Conference*, Volume 2, 1988, pp. 751–754.
4. Sedivy. J., Filcev, J., Uhlir, J., Vanek, T., Hanzl, V., Oliva, Z., and Kotek, P. , "The One-Chip Speech Recognition System", *EUROSPEECH 91: 2nd European Conference on Speech Communication and Technology Proceedings,* Volume 3, 1991, pp. 1357–1361.

Speech Compression

1. "A Real-Time French Text-to-Speech System Generating High-Quality Synthetic Speech, ICASSP 90, *1990 International Conference on Acoustics, Speech and Signal Processing,* IEEE Cat. No. 90CH2847-2, Volume 1, 1990, pp. 309–312.
2. Ancin, F.J., Burrows, B.L., and Carrasco, R.A., "Pitch Detection of Speech Signals Using the Wavelet Transform" *Fifth Bangor Symposium on Communications,* 1993, pp. 239–242.
3. Andreotti, F.G., Maiorano, V., and Vetrano, L., "A 6.3 kb/s CELP Codec Suitable for Half-Rate System", ICASSP 91, *1991 International Conference on Acoustics, Speech and Signal Processing,* IEEE Cat. No. 91CH2977-7, Volume 1, 1991, pp. 621–624.
4. Zhigang, C., Wei, Z., and Wanjun, Z., "An Improved Formant Synthesis System Using TMS320C25", *1991 IEEE International Symposium on Circuits and Systems,* IEEE Cat. No. 91CH3006-4, Volume 1, 1991, pp. 53–56.
5. Casale, S., Giarrizzo, C., and La Corte, A., "A DSP Implemented Speech/Voiceband Data Discriminator", GLOBECOM '88: IEEE Global Telecommunications Conference and Exhibition — Communications for the Information Age, *Conference Record,* IEEE Cat. No. 88CH2535-3, Volume 3, 1988, pp. 1419–1427.
6. Clarkson, P.M., and Bahgat, S.F., "Envelope Expansion Methods for Speech Enhancement", *Journal of the Acoustical Society of America,* Volume 89, No. 3, March 1991, pp. 1378–1382.
7. Cross, T.E., and Loasby, J.M., "Modelling the Vocal Tract Using Multiple Digital Signal Processors", *Sixth Intemational Conference on Digital Processing .of Signals in Communications.* Conf. Publ. No. 340, 1991, pp. 219–225.
8. Dankberg, M., Iltis, R., Saxton, D., and Wilson, P., "Implementation of the RELP Vocoder Using the TMS320", *ICASSP 84. Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing,* Volume 2, 1984, pp. 27.8/1–4.

9. Dixon, J.L., Varma, V.K., Sollenberger, N.R., and Lin, D.W., "Single DSP Implementation of a 16-kbps Sub-Band Speech Coder for Portable Communications", *ICASSP-89: 1989 International Conference on Acoustics, Speech and Signal Processing,* IEEE Cat. No. 89CH2673-2, Volume 1, 1989, pp. 184–187.

10. Frantz, G.A., and Lin, K.S., "A Low-Cost Speech System Using the TMS320C17", *IES Journal,* Volume 29, No. 3, October 1989, pp. 41-44.

11. Greenwood, M., and Dent, P., "A Full-Duplex ADPCM Voice Coder for Use in the Ferranti Zonephone", *IEEE Colloquium on VLSI Implementations for Second Generation Digital Cordless and Mobile Telecommunication Systems,* Digest No. 043, 1990, pp. 711–5.

12. Hill, P. D., and Mikael, W.B., "Real-Time Implementation of a Variable Stepsize Adaptive Algorithm", *Proceedings of the Twenty-Seventh Midwest Symposium on Circuits and Systems,* Volume 1, 1984, pp. 181–184.

13. Holck, A.W., and Anderson, W.W., "A Single-Processor LPC Vocoder", *ICASSP 84. Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing,* Volume 3, 1984, pp. 44.13/1–4.

14. Holck, A.W., "Low-Cost Speech Processing With TMS32010", *Midcon 83 Conference Record,* 1984, pp. 16.2/1–6.

15. Jain, V.K., Skrzypkowiak, S.S., and Heathcock, R.B., "Full Duplex Speech and Data Coder: Algorithm Enhancement Test Bed", GLOBECOM '88. IEEE Global Telecommunications Conference and Exhibition — Communications for the Information Age, *Conference Record* IEEE Cat. No. 88CH2535-3, Volume 3, 1988, pp. 1409–1413.

16. Karjalainen, M., and Laine, U.K., "A Model for Real-Time Sound Synthesis of Guitar on a Floating-Point Signal Processor", *ICASSP 91: 1991 International Conference on Acoustics, Speech and Signal Processing,* IEEE Cat. No. 91CH2977-7, Volume 5, 1991, pp. 3653–3656.

17. Kelly, D.P., and Melsa, J.L., "Syllabic Companding and 32 kb/s ADPCM Performance", *IEEE International Conference on Communications 1985,* IEEE Cat. No. 85CH2175-8, Volume 1, 1985, pp. 414–417.

18. Kitson, F.L., and Zeger, K.A., "A Real-Time ADPCM Encoder Using Variable Order Prediction Speech", *ICASSP 86 Proceedings: International Conference on Acoustics, Speech and Signal Processing,* IEEE Cat. No. 86CH2243-4, Volume 2, 1986, pp. 825–828.

19. Lazzari, V., Quacchia, M., Sereno, D., and Turco, E., "Implementation of a 16-kbit/s Split Band-Adaptive Predictive Codec for Digital Mobile Radio Systems", *CSELT Technical Reports,* Volume 16, No. 5, August 1988, pp. 443–447.

Speech Compression (Continued)

20. Lazzari, V., Quacchia, M., Sereno, D., and Turco, E., "SB-APC Codec for Digital Mobile Radio Applications", *Signal Processing IV: Theories and Applications. Proceedings of EUSIPCO-88. Fourth European Signal Processing Conference,* Volume 2, 1988, pp. 615–617.

21. Leung, S.H., Chung, C.Y., and Luk, A., "A Low Noise Fixed-Point Implementation of GSM Speech Codec On TMS320C25", *Proceedings of the Fourth Australian International Conference on Speech Science and Technology,* 1992, pp. 381–386.

22. Liu, Z.Y., and Zhang, L.P., "Implementation of Modified Regular-Pulse Excited Linear Predictive Codec on TMS320C25", *41st IEEE Vehicular Technology Conference. Gateway to the Future Technology in Motion,* IEEE Cat. No. 91CH2944-7, 1991, pp. 280–284.

23. Liu, Z.Y., and Zhu, W.M, "Realtime Implementation Algorithm of CELP at 4.8 kb/s", *ICC 91 International Conference on Communications, Conference Record,* IEEE Catalog No. 91CH2984-3, Volume 3, 1991, pp. 1731–1735.

24. Macres, J.V., "Real-Time Implementations and Applications of the US Federal Standard CELP Voice Coding Algorithm", *Proceedings of the Tactical Communications Conference. Tactical Communications: Technology in Transition,* Volume 1 (Unclassified Papers), IEEE Cat. No. 92TH0467-1, 1992, pp. 41–45.

25. Marks, J.A., "Real Time Speech Classification and Pitch Detection", COMSIG 88. Southern African Conference on Communications and Signal Processing, *Proceedings,* IEEE Cat. No. 88TH0219-6, 1988, pp. 1–6.

26. Minin, A.V., and Deryugin, S.N., "The Adaptive Differentia1 PCM Codec of the DKD-400 Digital Satellite Communications Equipment", *Elektrosvyaz,* October 1992, pp. 24–26.

27. Mumolo, E., Riccio, A., and Abbattista, G., "An Efficient Algorithm for Real-Time Voiced/Unvoiced Decision", *EUROSPEECH '91 — 2nd European Conference on Speech Communication and Technology Proceedings,* Volume 3, 1991, pp. 1305–1308.

28. Nieminen, T., and Simola, A., "A Gateway Between a EUROCOM D/1-Network and a Private PTT-Type CCITT SS7-Network", MILCOM '92 — Communications: Fusing Command, Control and Intelligence, *Conference Record,* IEEE Cat. No. 92CH3131-0, Volume 1, 1992.

29. Hongwen, P., and Fengji, S., "Research and Implementation of Linear Predictive Speech Analysis and Synthesis", China 1991 International Conference on Circuits and Systems, *Conference Proceedings,* IEEE Cat. No. 91TH0387-1,Volume 1, 1991, pp. 22–25.

30. Perosino, F., and Quacchia, M., "DSP-Based Implementation of SB-ADPCM Audio Codec for ISDN Terminals", *CSELT Technical Reports,* Volume 18, No. 2, 1990, pp. 83–87.

Speech Compression (Continued)

31. Rajugopal, G.R., and Paulraj, A., "Multichannel All-Digital PCM-ADM Transcoder", *IETE Technical Review,* Volume 9, No. 3, May 1992, pp. 221–217.

32. Raviraj, C.R., and Jones, E.V., "Adaptive Coding for Conversational Speech Communication", *Second IEEE National Conference on Telecommunications,* (Conference Publication No. 300), 1989, pp. 344–348.

33. Reilly, M.T., "A Hybridized Linear Prediction Code Speech Synthesizer", *MILCOM 86: 1986 IEEE Military Communications Conference. Communications-Computers: Teamed for the '90's. Conference Record,* IEEE Cat. No. 86CH2323-4, Volume 2, 1986, pp. 32.511–5.

34. Rose, C., and Donaldson, R.W., "Real-Time Implementation and Evaluation of An Adaptive Silence Deletion Algorithm for Speech Compression", *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing,* IEEE Cat. No. 91CH2954-6, Volume 2, 1991, pp. 461–468.

35. Rothweiler, J., "Noise-Robust 1200-bps Voice Coding", *Proceedings of the Tactical Communications Conference. Tactical Communications: Technology in Transition,* Volume 1 (Unclassified Papers), IEEE Cat. No. 92TH0467-1, 1992, pp. 65-69.

36. So, J.L., "Implementation of an NIC Nearly Instantaneous Companding 32 kbps Transcoder Using the TMS320C25 Digital Signal Processor", GLOBECOM '88, IEEE Global Telecommunications Conference and Exhibition — Communications for the Information Age, *Conference Record,* IEEE Cat. No. 88CH2535-3, Volume 3, 1988, pp. 1414–1418.

37. Stone, R.E., "Speech Processing Using the TMS32010 — A Case Study", *Digital Signal Processing: Principles, Devices and Applications,* 1990, pp. 354–370.

38. Tsakalos, N., and Zigouris, E., "Autocorrelation-Based Pitch Determination Algorithms for Real-Time Vocoders with the TMS32020/C25", *Microprocessors and Microsystems,* Volume 14, No. 8, October 1990, pp. 511–516.

39. Wong, O.Y., Law, K.W., Leung, S.H., Chan, C.F., and Luk, A., "A Novel Pulse-Excitation Using Coded Locations for Linear Predictive Speech Coding", *Sixth International Conference on Digital Processing of Signals in Communications,* Conf. Publ. No. 340, 1991, pp. 300–304.

40. Wei, Z., and Zhigang, C., "Real-Time Formant Speech Synthesis Using the TMS320C25", *China 1991 International Conference on Circuits and Systems. Conference Proceedings,* IEEE Cat. No. 91TH0387-1, Volume 1, 1991, pp. 41-44.

41. Zinser, R.L., "An Efficient, Pitch-Aligned High-Frequency Regeneration Technique for RELP Vocoders", ICASSP 85, *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing,* IEEE Cat. No. 85CH2118-8, Volume 3, 1985, pp. 969–972.

System Design Considerations

1. Chen, D.C.C., and Price, R.H., "A Real-Time TMS320C40-Based Parallel System for High Rate Digital Signal Processing", ICASSP 91, *1991 International Conference on Acoustics, Speech and Signal Processing,* IEEE Cat. No. 91CH2977-7, Volume 3, 1991, pp. 1573–1576.

2. Culloch, A.D., "3L Parallel C for the Texas Instruments TMS320C40: Initial Applications", *Parallel Computing and Transputer Applications*, Volume 2, 1992, pp. 1080–1088.

3. Dunn, S.M., Peters, J.E., Finkel, B., and Neafsey, L., "DISIPLE: Digital Signal Processor Programming Language and Environment", *IEEE Transactions on Acoustics, Speech and Signal Processing,* Volume 38, No. 11, November 1990, pp. 2001–2003.

4. Marshall, T.G., Jr., "A Matlab/TMS320/TMS340 Image Processing Environment", *1992 IEEE International Symposium on Circuits and Systems,* IEEE Cat. No. 92CH3139-3, Volume 5, 1992, pp. 2505–2508.

5. Morgado, A.M.L.S., Domingues, J.P.P., Loureiro, C.F.M., Assuncaao, J.M.V., and Correia, C.M.B.A., "Data Acquisition and Signal Processing System Based on TMS320C50 and on a IMSA100 Processing Cascade", *6th Mediterranean Electrotechnical Conference Proceedings*, IEEE Cat. No. 91CH2964-5, Volume 1, 1991, pp. 340–342.

6. Pfeiffer, E., and Disch, J., "Using Ada with Embedded DSPs", *Embedded Systems Programming,* Volume 6, No. 12, December 1993, pp. 32–35, 38, 40–41.

7. Prandolini, R., and Sridharan, S., "VLSI Implementation of a Block Floating-Point Coprocessor for the TMS320 Fixed-Point Digital Signal Processor", *7th Australian Microelectronics Conference Proceedings*, 1988, pp. 33-40.

8. Reichler, T., Hartimo, I., and Jaatinen, J., "Automatic Signal Processor Code Generation: Matrix Reduction-Based Module Optimization", *1990 IEEE International Symposium on Circuits and Systems,* IEEE Cat. No. 90CH2868-8, Volume 4, 1990, pp. 2901–2904.

9. Robillard, J., "Telecommunications Interfacing to the TMS32010", *Digital Signal Processing Applications With the TMS320 Family,* Volume 1, 1987, pp. 383–413.

10. Tomasovic, L., and Marcek, **A.,** "Experimental Module with TMS320C25 Processor", *Elektrotechnicky Casopis,* Volume 44, No. 12, 1993, pp. 373–375.