# IMPLEMENTATION AND EVALUATION OF VARIOUS STOP AND WAIT TYPE II HYBRID ARQ SCHEMES FOR MOBILE RADIO

by

REMO L. AGOSTINO

B. A. Sc., University of British Columbia, 1990.

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Department of Electrical Engineering

We accept this thesis as conforming

to the required standard

---

THE UNIVERSITY OF BRITISH COLUMBIA

September 1993

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Electrical Eng.

The University of British Columbia
Vancouver, Canada

Date Oct 13/93

# ABSTRACT

This thesis investigates the design and implementation issues involved in the development of various Stop-and-Wait (SW) Type II Hybrid Automatic Repeat reQuest (ARQ) strategies. The modulation scheme utilized is the North American digital cellular standard known as $\pi/4$–shift DQPSK. The general Complementary Punctured Convolutional (CPC) SW Type II ARQ scheme is presented and numerically analyzed in both an AWGN channel and a combined AWGN and Rayleigh fading channel. The three variations of the general scheme implemented are: Rate 3/4 CPC SW Type II ARQ, Rate 3/4 CPC SW Type II ARQ with Code Combining, and an Adaptive CPC SW Type II ARQ scheme. The prototypes are implemented with two Spectrum TMS320C30 Digital Signal Processing (DSP) cards and a host IBM PC. The experimental data for the prototypes were verified and were in good agreement with the numerical results. This validated the prototypes' correct and proper operation along with the DSP software modules used by the prototypes. It is shown that the upgrade of the CPC SW Type II ARQ scheme to a Code Combining and an Adaptive scheme requires small software modifications. It is the versatility and flexibility of the DSP cards which allow these upgrades to be easily accomplished and extremely cost effective. The Code Combining upgrade increased the throughput performance of the general rate 3/4 scheme at low SNR levels. The Adaptive scheme resulted in an increase at both low and high SNR levels with a slight degradation at medium SNR levels with respect to the throughput curve of the general rate 3/4 scheme.

# Contents

# List of Tables

# List of Figures

# Acknowledgments

I would like to thank my mother and aunt, Maddalena and Maria Taddei, for their continuous moral support and constant encouragement throughout my academic career. I would also like to issue a special thanks to my uncle, Tony Bolognese, for having played a major role in my decision to enter the exciting field of communications. I am enormously grateful to my supervisors, Dr. Samir Kallel and Dr. V. C. M. Leung, for their constant guidance, moral support, and invaluable experience which allowed me to complete this thesis. I would also like to thank my fellow students and especially Dimitrios P. Bouras and William Cheung for their insightful and stimulating discussions. Finally, I would like to acknowledge the assistance provided by the B.C. Science Council.

# Chapter 1 Introduction

## Section 1.1 ARQ Schemes

The problem of providing an efficient reliable data communications link in a land mobile radio channel is of great practical importance. Automatic Repeat reQuest (ARQ) protocols or similar custom tailored Radio Data Link Protocols are commonly used to provide a virtually error free data link for the radio channel. The ARQ protocol ensures a consistent data quality under varying channel conditions. The functions the ARQ protocol must accomplish can be divided into two different classes: low level functions involved with encoding and decoding of protocol information in the data packets and high level functions concerned with the request retransmission algorithm to support frame transmission services. The message itself is contained in the data packet of the frame, whereas the destination address and other pertinent information is contained in the header which precedes the data packet. A code with good error detecting capability is used to encode the header and data packet separately. Typically, a Cyclic Redundancy Code (CRC) is used [1]. The header is independently encoded to allow all mobile radio users to decode it in order to distinguish if the frame is addressed to them and decide whether to process the data packet.

### 1.1.1 Stop-and-Wait ARQ

In a Stop-and-Wait ARQ (SW ARQ) scheme, the transmitter sends a single frame and stops to await the reply of the receiver. No other frame can be sent until the receiver's reply arrives at the transmitter. Three possible events may arise once a transmission has taken place. The receiver may send an acknowledgment (ACK) to indicate that the

frame was received error free; or a negative acknowledgment (NACK) if it was received in error; or no reply if the frame was so corrupted by noise as not to be received. To account for this last event, the transmitter is equipped with a timer. Once a frame has been sent, the transmitter awaits for a recognizable reply (ACK or NACK). If no such reply is received during the time-out period, the frame is retransmitted. Therefore, any reply other than an ACK will result in the transmitter retransmitting the same frame again. Figure 1.1 illustrates the SW ARQ scheme.

It is inefficient to utilize a SW ARQ protocol in a single frequency system because the time required for the transmitter to await the receiver's reply is wasted air time. The typical mobile radio system uses a number of frequencies to communicate between the base stations and the mobile users. This configuration allows the SW ARQ protocol to make efficient use of its air time. For example, after the base transmits a message to mobile A, it can send another message to any other mobile while awaiting the reply of mobile A on the return channel. In this respect the SW ARQ protocol can be well suited for mobile radio systems.

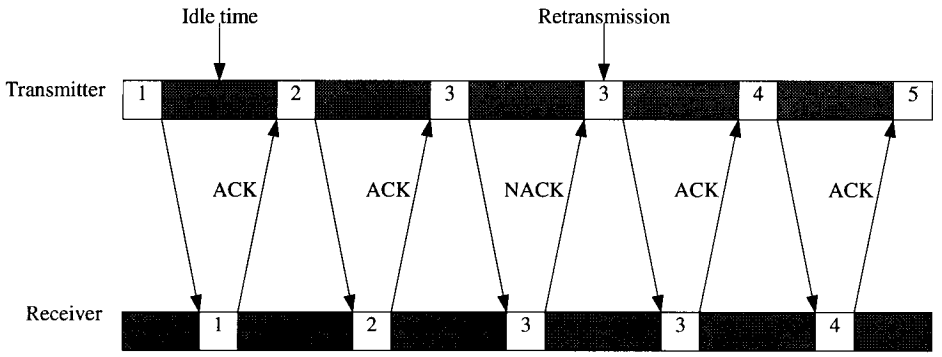


Figure 1.1 Stop-and-Wait ARQ Scheme

### 1.1.2 Type I Hybrid ARQ

A hybrid ARQ system utilizes both Forward Error Correction (FEC) coding and error detection coding (incorporated in the ARQ scheme). The FEC code is used to reduce the number of retransmissions. In a Type I Hybrid ARQ scheme the message and its error detecting parity bits (typically CRC), are further encoded with a FEC code. At the receiver, the FEC parity bits are used to correct channel errors. The FEC decoder (typically a Viterbi Decoder) outputs an estimate of the received message and its error detecting parity bits. This estimate is tested by the error detection decoder (CRC checker) to determine if the message is error free. Figure 1.2 depicts a Type I Hybrid ARQ communication system.

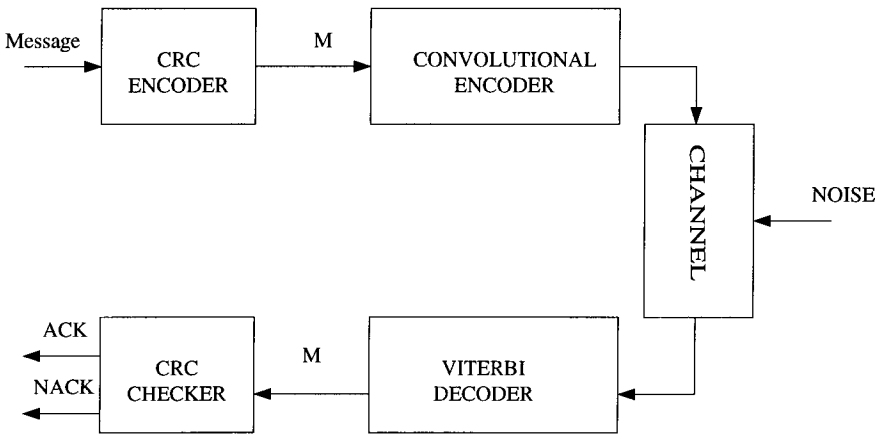


Figure 1.2 Typical Type I Hybrid ARQ System

The efficiency of a Type I ARQ system in comparison to a plain ARQ system depends on the level of noise corrupting the channel. If the Signal-to-Noise Ratio (SNR) is high, the Type I ARQ scheme does not result in any improvement. The FEC parity bits are wasted, as a result of the signal strength being strong enough to deliver error free messages. On the other hand, the Type I system does show an increase in efficiency

at low SNR levels, and since the signal strength is so poor, error free reception is very unlikely and the FEC parity bits are utilized to correct channel errors.

### 1.1.3 Type II Hybrid ARQ

In a Type II ARQ scheme, the FEC parity bits are only sent if the received message contains errors. The transmitter would alternate between sending the message with its error detection parity bits on one transmission, and the FEC parity bits on the next. Note that the FEC parity bits are only sent if the received message contains errors. With this scheme, any error free reception of the message with its error detection parity bits delivers the message. If the FEC parity bits are invertible, any error free reception of the FEC parity bits also delivers the message. Finally, if both the message with its error detection parity bits and the FEC parity bits are in error, combining these two frames for error correction may successfully deliver the message. The Type II system offers the benefit of performing as a plain ARQ scheme at high SNR and performing as a Type I system at low SNR.

## Section 1.2 Thesis Goals

The disadvantage of Type I and Type II hybrid ARQ schemes is the failure to provide a useful throughput at high channel error rates. Application of code combining to hybrid ARQ schemes to achieve a useful throughput has been investigated [2, 3]. Code combining involves taking frames received in error and optimally combining them with their repeated copies. Therefore, the receiver would process a combination of all received sequences for that frame, rather than only the two most recently received ones as in the conventional Type II system.

An adaptive hybrid ARQ system utilizing code combining would be optimal. Adaptive refers to the FEC coding scheme being able to adjust to the channel conditions and data protection needs. Typically, a fixed code with a certain error rate and correction capability matched to the protection requirement of the data and the worst channel conditions is used. Unfortunately, different data (voice, FAX, computer data files, all using the same channel) have different error protection needs and what may be appropriate for one type may be inappropriate for another. Another problem, is the mobile radio channel conditions are constantly changing due to its multipath and time varying characteristics. Therefore, an adaptive code combining hybrid ARQ scheme would generally yield a higher throughput than a non-adaptive scheme in a radio channel [4].

Motivated by the above, this thesis investigates the design, implementation issues, and performance evaluation of various adaptive and non-adaptive FEC coding schemes of a Type II SW ARQ system. The research contributions can be summarized as follows:

1. The Software design, implementation, and test of a Digital Signal Processing (DSP) Module Library for the Spectrum TMS32C30 DSP card housed in an IBM PC platform. The library consists of the following modules:

    - **CRC Encoder/Decoder**
    - **Rate 1/2 Convolutional Encoder**
    - **Puncturing Module**
    - **Rate 1/2 Soft Decision Viterbi Decoder**
    - **Block Interleaver**
    - **Soft Data Deinterleaver**
    - **Queueing Module**

- **$\pi$/4–shift DQPSK Baseband Transmitter/Receiver**

2. The Software implementation and evaluation of a Complementary Punctured Convolutional (CPC) coding scheme for the SW Type II ARQ system with and without code combining utilizing the DSP library in an AWGN channel and a combined AWGN and Rayleigh Fading channel.
3. Software upgrade and performance evaluation of an Adaptive CPC SW Type II ARQ scheme utilizing the DSP library in an AWGN channel and a combined AWGN and Rayleigh fading channel.

## Section 1.3 Thesis Organization

The thesis consists of five chapters and one appendix. It is organized as follows:

- Chapter 2 discusses the $\pi$/4–shift DQPSK modulation system implemented and its theoretical and practical performance.
- Chapter 3 explains the generalized Complementary Punctured Convolutional (CPC) coding scheme for a SW Type II ARQ protocol with and without code combining. It also discusses in detail the DSP prototype CPC SW Type II ARQ scheme implemented. Finally, the prototype's performance is analyzed and evaluated.
- Chapter 4 presents the Adaptive CPC SW Type II ARQ scheme implemented and its performance evaluation. This chapter will also compare the three ARQ schemes implemented and discuss their performances.
- The thesis' conclusions and suggestions for future work are cited in Chapter 5.
- Appendix A contains the software listings for the DSP Module Library, the Adaptive SW Type II ARQ Protocol, the Transmitter DSP card, and the Receiver DSP card.

The CPC scheme's software is a subset of the Adaptive scheme and is therefore not listed.

# Chapter 2 $\pi/4$–Shift DQPSK Modulation Scheme

## Section 2.1 Introduction

The $\pi/4$ shift Differential Quadrature Phase Shift Keying (DQPSK) modulation scheme has become the modulation standard for the North American and Japanese digital cellular communications system [5]. This modulation scheme is used in the implementation of the SW Type II ARQ scheme for mobile radio communications in order to get practical results which are of interest to the cellular industry. The organization of this chapter is as follows. Section 2 will review the $\pi/4$ shift DQPSK modulation technique. Sections 3 to 5 will describe the DSP software and the RF hardware required to construct the system. A performance comparison between the theoretical and implemented modulation scheme is presented in Section 6.

## Section 2.2 Transmitter Model

Figure 2.1 illustrates the transmitter model of the $\pi/4$ shift DQPSK system. The Phase Shift Encoder and Baseband Generator Block produce the unfiltered rectangular pulse waveforms which are denoted as $u(t)$ and $v(t)$ in the Inphase (I) and Quadrature (Q) channels respectively. The waveforms $u(t)$ and $v(t)$ are Nyquist filtered and passed to the RF modulator which mixes the I and Q components to form the RF modulated signal.

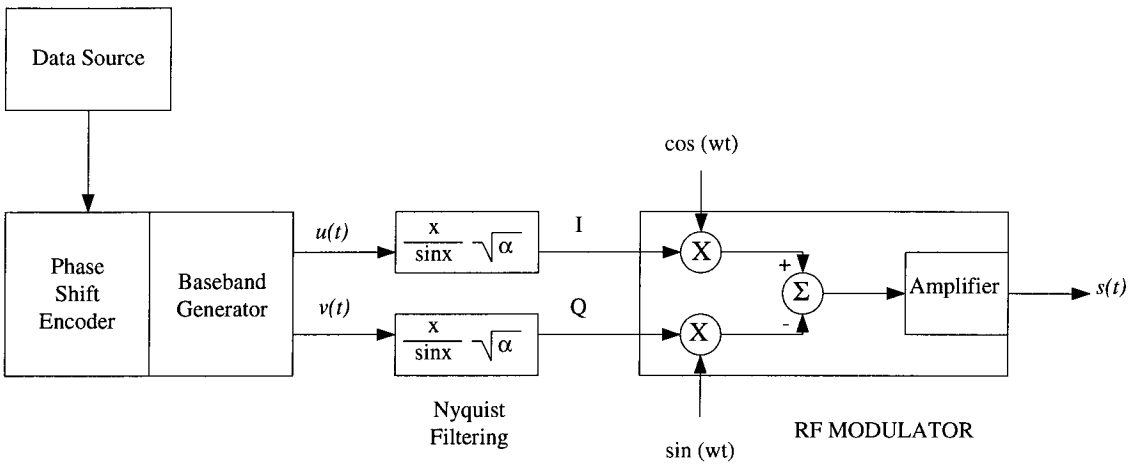


Figure 2.1 Block Diagram of the $\pi/4$ shift DQPSK Transmitter.

Equations 2.1 and 2.2 represent the RF modulated signal.

$$s_i(t) = \sqrt{\frac{2E}{T_s}} \cos\left(w_c t + \frac{2\pi}{8}\right) \; i = 0, 1, \cdots, 7 \tag{2.1}$$

$$s_i(t) = \sqrt{\frac{2E}{T_s}} \left\{ \cos w_c t \cos \frac{\pi}{4} i - \sin w_c t \sin \frac{\pi}{4} i \right\} \quad i = 0, 1, \cdots, 7 \,. \tag{2.2}$$

In Equation 2.2, $E$ repesents the energy per symbol, $T_s$ is the symbol duration, and $w_c$ is the carrier frequency. Figure 2.2 is the state-space signal diagram which illustrates the possible 8 modulated carrier signals at their sampling instants. The state-space diagram shows that the transmitted signals are chosen from two signal groups, the circles (even numbered points {0, 2, 4, 6}) and the crosses (odd numbered points {1, 3, 5, 7}). If the current signal is at one of the four phase states designated by a circle, it shifts to one of the four phase states designated by a cross at the next symbol transition and vice versa. The current signal is not allowed to shift to a fellow member of its phase state at the next symbol transition (i.e., circle to circle or cross to cross). As a result of this constraint, the differential phase shift between two consecutive symbols can only be $k\pi/4$, where

$k = \pm 1$ or $\pm 3$. Consecutive phase shifts of $\pm\pi/2$ and $\pi$ are inhibited. The connections in the state-space diagram indicate the possible phase transitions.

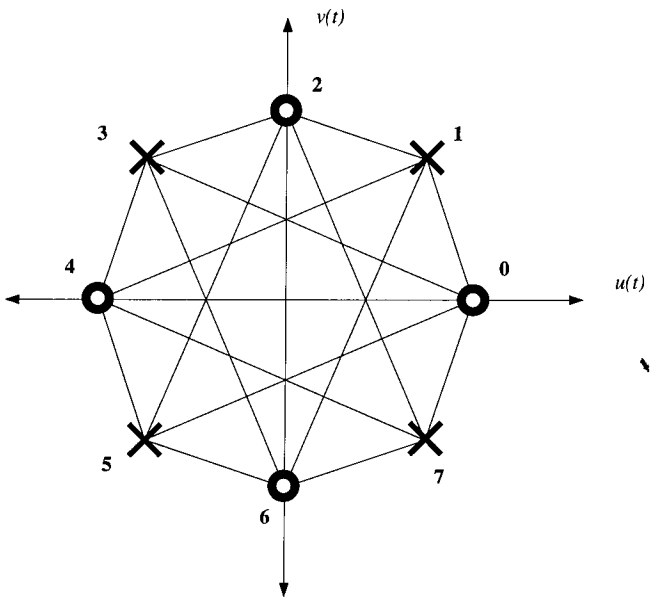


Figure 2.2 State-space diagram of the $\pi/4$ shift DQPSK modulated carrier at sampling points [5].

The differential phase shift encoding operation can be mathematically represented by Equations 2.3, 2.4(a), 2.4(b), and Table 1 [6].

$$s_i(t) = \sqrt{\frac{2E}{T_s}}\{u_k \cos w_c t - v_k \sin w_c t\} \tag{2.3}$$

$$u_k = u_{k-1} \cos\theta_k - v_{k-1} \sin\theta_k \tag{2.4a}$$

$$v_k = u_{k-1} \sin\theta_k + v_{k-1} \cos\theta_k. \tag{2.4b}$$

In Equations 2.4(a) and 2.4(b), $u_k$ and $v_k$ are the signal levels of the pulse amplitudes of

| Information Symbol | $\theta_k$ |
|---|---|
| 11 | $\pi/4$ |
| 01 | $3\pi/4$ |
| 00 | $-3\pi/4$ |
| 10 | $-\pi/4$ |

Table 1 Phase Shift as a function of Information Symbol.

$u(t)$ and $v(t)$ for a period equal to the symbol duration. The signal levels $u_k$ and $v_k$ are determined from the previous signal levels, $u_{k-1}$ and $v_{k-1}$ and the phase shift, $\theta_k$ resulting from the current information symbol. The relationship between the phase shift and the current information symbol is given in Table 1. From Equations 2.4(a) and 2.4(b), it can be seen that the amplitudes of $u(t)$ and $v(t)$ can take the values of 0, $\pm\frac{\sqrt{2}}{2}$, or $\pm 1$. For example, assume the current signal is $s_0(t)$ (i.e., $\theta_0 = 0$, $u_0 = 1$, and $v_0 = 0$ during $0 \leq t \leq T_s$). At time $t = T_s$, the information symbol 11 is sent. Therefore, $\theta_1=\pi/4$ and from Equations 2.4(a) and 2.4(b), $u_1=\frac{\sqrt{2}}{2}$ and $v_1=\frac{\sqrt{2}}{2}$ denoting signal $s_1(t)$.

From the state-space diagram and the mathematical model it follows that the information symbol is contained in the phase difference between two consecutive sampling instants. The receiver only requires the phase difference between two consecutive sampling intervals in order to retrieve the transmitted information symbol. As a result, the receiver does not need to phase synchronize with the transmitter.

## Section 2.3 DSP Implementation of the Phase Shift Encoder and Baseband Generator

The transmitter and receiver is implemented utilizing the Texas Instruments TMS320C30 DSP chip. The DSP platform consists of a Spectrum TMS320C30 card and software development tools for an IBM PC. The TMS320C30 DSP cards were cho-

sen due to their availability and excellent software support. A software based DSP design is more versatile, flexible, and modular than an all hardware design. The DSP system allows the user to make changes and updates to their software algorithms in a fraction of the time required for a hardware update.

The flowchart shown in Figure 2.3 describes the baseband transmission algorithm.

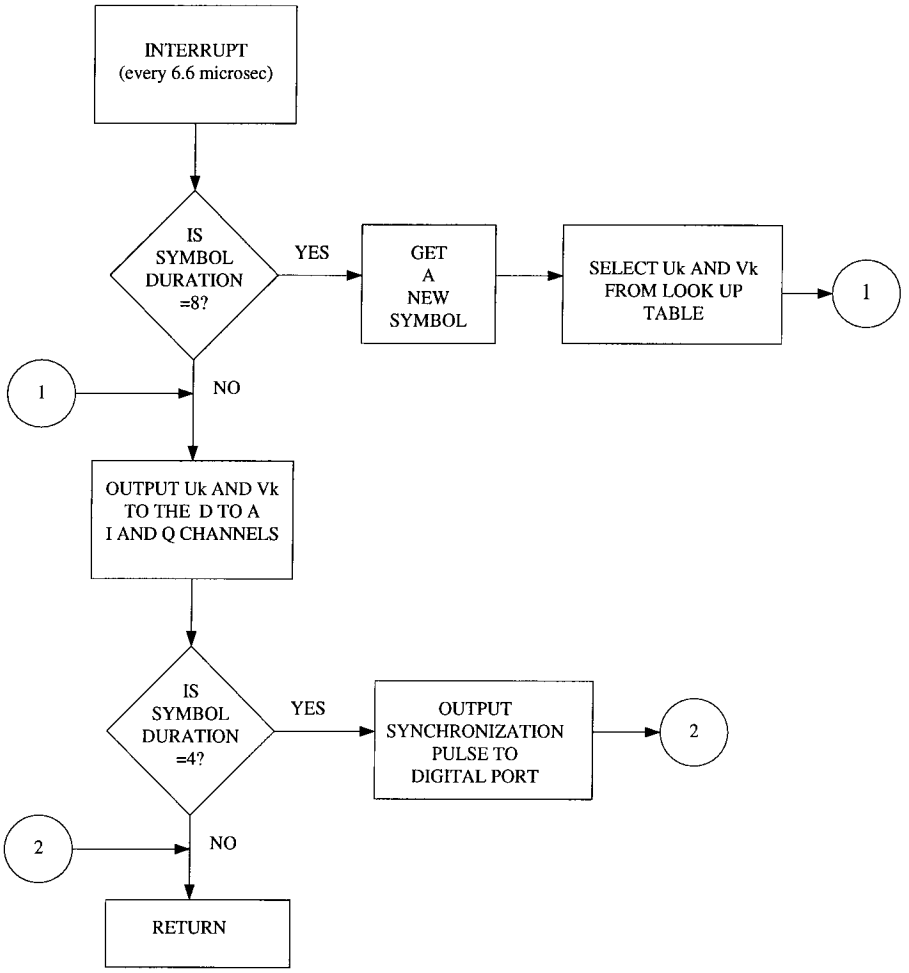


Figure 2.3 Flow chart representing baseband transmission algorithm.

The algorithm is interrupt driven by one of the two timers that the TMS320C30 chip features. The timer is set to 6.6 $\mu$s, which is the upper limit available on the Spectrum card

housing the TMS320C30. The timer value has a direct result on the rate of transmission. The smaller the timer value, the higher the transmission rate. The baseband transmission routine is interrupt driven to allow the DSP chip to encode and construct other frames for transmission while the current frame is being transmitted. Therefore, even though a SW ARQ scheme is being used, the scheme may be upgraded to a Selective Repeat (SR) ARQ with little or no change to the transmission algorithm.

The Baud rate, which is the number of symbols transmitted per second, is determined by the number of times the routine is executed per symbol or dibit. The variable *symbol_duration_count* keeps track of this value, which is compared to a user set limit. In the algorithm shown in Figure 2.3, the limit is set to a value of 8 and gives rise to a baud rate of 18.939kHz according to equation 2.5.

$$\text{Baud rate } = \{(symbol\_duration\_count \text{ Limit}) * 6.6\mu s\}^{-1} . \qquad (2.5)$$

Every time the interrupt routine is executed, the *symbol_duration_count* is checked. If a new symbol or dibit is required, it is fetched from memory and the amplitudes $u_k$ and $v_k$, of the baseband signals *u(t)* and *v(t)*, are chosen from the $\pi/4$ shift DQPSK encoder look up table displayed as Table 2. Table 2 shows all possible state transitions given the

| | **Previous Signal $s_i(t)$** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Current Symbol** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 00 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 |
| 01 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 |
| 10 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 11 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 |

Table 2 $\pi/4$ Shift DQPSK State Encoder Look Up Table.

previous signal $s_i(t)$ and the current symbol or dibit to be transmitted. This table is a direct

result of equations 2.4(a), 2.4(b), and Table 1. Once the values for $u_k$ and $v_k$ are chosen, they are written to the Digital to Analog registers, which in turn outputs an analog voltage on the I and Q channels. Note the transmitter outputs a +5 volt synchronization pulse on the TMS320C30 digital channel at approximately the middle of the symbol duration.

The baseband waveforms *u(t)* and *v(t)* are filtered before being sent to the RF modulator. In the transmitter model discussed in Section 2.2, Nyquist filters were used in order to eliminate Intersymbol Interference (ISI) and maximize the Signal-to-Noise Ratio (SNR). Butterworth filters, which are contained on the Spectrum DSP cards, were used in the prototype implementation. As a consequence of not using Nyquist filters, the received noise power will be greater in the Butterworth filter case.

## Section 2.4 RF Modulator/Demodulator and Channel

A detailed block diagram of the hardware implemented RF modulator/demodulator is shown in Figure 2.4 and presented in [7]. The modulator and demodulator are designed to operate at the relatively low carrier frequency of 1.5 MHz. The carrier frequency enters the modulator to be divided into its I and Q components by a 90° splitter. The carrier's I and Q components are then mixed with the I and Q baseband signals and summed by a signal combiner. The resulting RF modulated carrier is amplified and passed to the channel module, which allows fading to be simulated by the use of the Digital Fading Simulator presented in [8]. White Gaussain noise is also added to the channel from a White Noise Generator whose band coverage is 6 kHz to 25 MHz. The modulated carrier and white noise is filtered by a Band Pass Filter (BPF), which has a 3 dB bandwidth of 200 kHz centered at the carrier frequency of 1.5 MHz. The bandwidth of the BPF

is much greater than that of the Low Pass Filters (LPF) at the demodulator and is used to minimize noise.

The demodulator takes the received RF modulated carrier and splits it into its I and Q components, which are then coherently mixed down to the baseband signals. The baseband I and Q signals are passed through Low Pass Filters (LPF) and fed to the DSP card for Differential Baseband Detection.

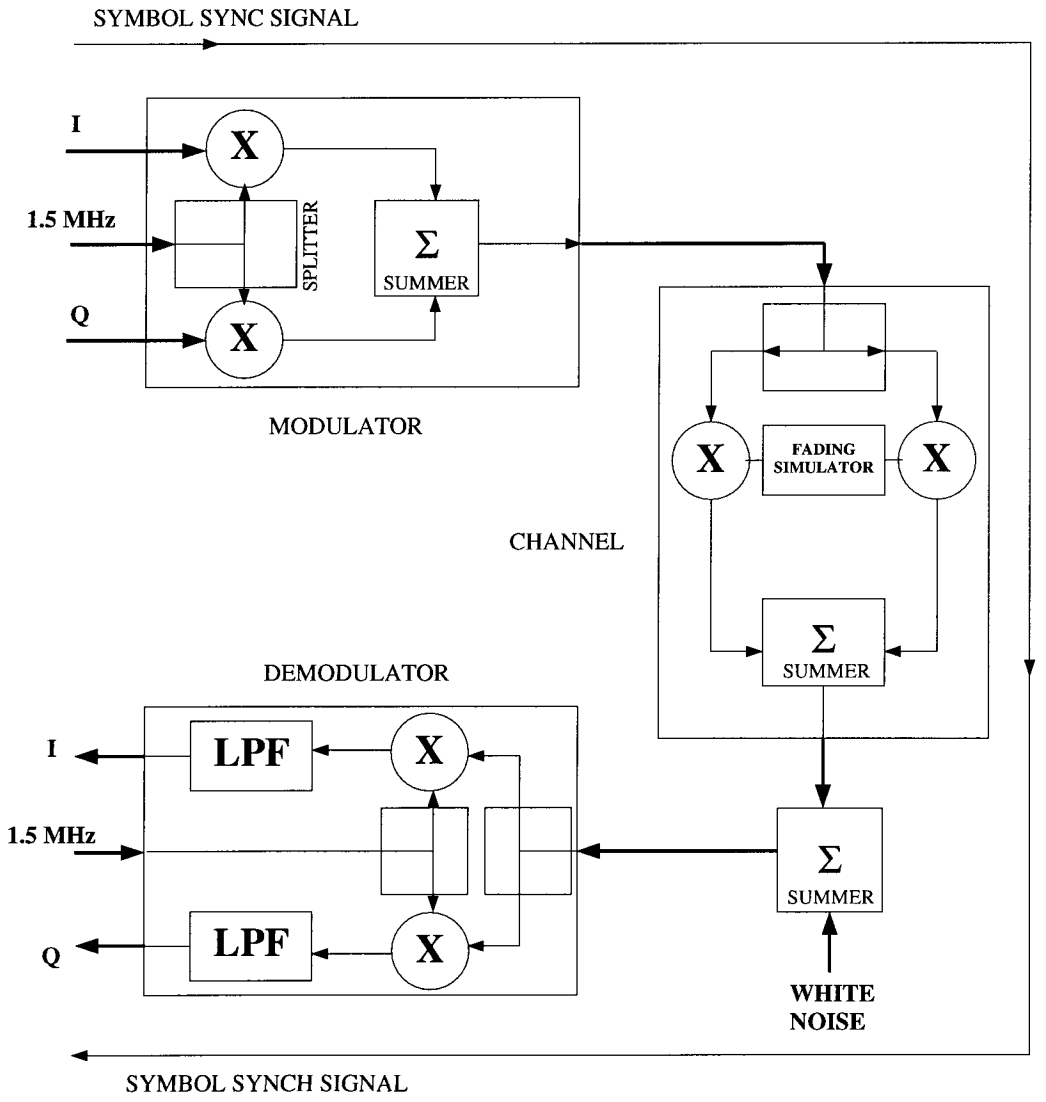


Figure 2.4 Modulator , Demodulator, and Channel simulator.

Note the symbol synchronization pulse is directly connected from the transmitter DSP card to the receiver DSP card. In practice a local oscillator, closely tuned to the symbol rate of the I and Q channels, would trigger the receiver. This procedure was investigated, but it resulted in a synchronization problem. It was observed that approximately 150–200 symbols were correctly received, immediately followed by 50–100 incorrect symbols and then the cycle begins again. The local oscillator drifted in and out of synchronization with the I and Q channels' symbol rate. In order to obtain optimum synchronization, a Phase Locked Loop (PLL) circuit was employed. The PLL worked and the results were encouraging but required further research. Since the investigation of symbol synchronization effects is beyond the scope of this thesis, we opted to use the transmitter DSP card to trigger the receiver.

## Section 2.5 DSP Implemented Baseband Differential Detector

The block diagram of the Differential Detector is shown in Figure 2.5 [6]. Once

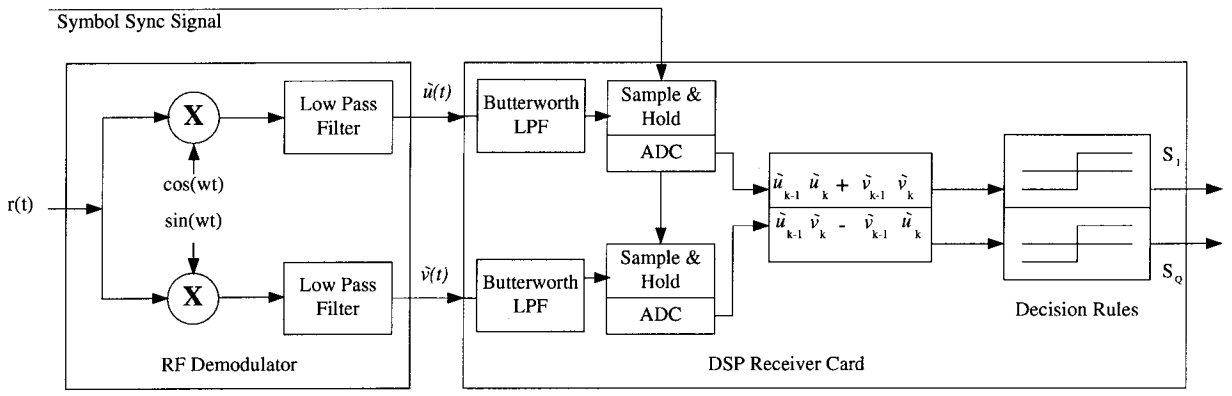


Figure 2.5 DSP Baseband Differential Detector Block Diagram

the RF modulated carrier is converted into its I and Q baseband signals $\grave{u}(t)$ and $\grave{v}(t)$, it is ready to be processed by the DSP Differential Detector. The DSP card drives each baseband signal through a Butterworth filter, a Sample and Hold circuit, and an Analog

to Digital Converter (ADC). It is the digital output of the ADC that the TMS320C30 addresses in order to obtain a real floating point representation of the amplitudes $\grave{u}_k$ and $\grave{v}_k$ of the received baseband signals. The DSP detector samples each symbol and uses equations 2.6(a) and 2.6(b) in order to transform the DQPSK real data $\grave{u}_k$ and $\grave{v}_k$, to QPSK real data $w_k$ and $z_k$ [6].

$$w_k = \grave{u}_{k-1}\grave{u}_k + \grave{v}_{k-1}\grave{v}_k = \cos(\theta_k - \theta_{k-1}) \tag{2.6a}$$

$$z_k = \grave{u}_{k-1}\grave{v}_k - \grave{v}_{k-1}\grave{u}_k = \sin(\theta_k - \theta_{k-1})\,. \tag{2.6b}$$

This transformation of $\pi/4$–shift DQPSK data to QPSK data makes each symbol no longer dependent on the previous symbol for decoding purposes. Note that $w_k$ and $z_k$ are equivalent to $\sin(\theta_k - \theta_{k-1})$ and $\cos(\theta_k - \theta_{k-1})$, where $\theta_k - \theta_{k-1}$ is the phase shift. It follows that, since the phase shift can only be $k\pi/4$, where $k = \pm 1$ or $\pm 3$, $w_k$ and $z_k$ will be approximately $\pm\frac{\sqrt{2}}{2}$. The real floating point values obtained for $w_k$ and $z_k$ may be fed into a soft decision Viterbi decoder or can be hard decoded according to the following decision rules:

$$\begin{aligned} S_I &= 1 \quad \text{if } w_k > 0 \qquad S_I = 0 \quad \text{if } w_k < 0 \\ S_Q &= 1 \quad \text{if } z_k > 0 \qquad S_Q = 0 \quad \text{if } z_k < 0\ , \end{aligned} \tag{2.7}$$

where $S_I$ and $S_Q$ are the least and most significant bit of the symbol respectively. Note the prototype system utilizes the same carrier frequency for both the modulator and demodulator. In practice a local oscillator tuned to the same frequency as the transmitter is used to demodulate the received carrier. This local oscillator will have a constant phase difference but it has been shown that the phase error is cancelled through differential detection [6].

## Section 2.6 Theoretical Analysis and Prototype Performance

The probability of a binary digit error for four-phase DPSK with Gray coding in an AWGN channel is given by [9] as

$$P_{4b}(e) = e^{\frac{-2E_b}{N_o}} \left\{ \sum_{k=0}^{\infty} \left(\sqrt{2}-1\right)^k I_k\left(\frac{\sqrt{2}E_b}{N_o}\right) - \frac{1}{2} I_0\left(\frac{\sqrt{2}E_b}{N_o}\right) \right\}, \qquad (2.8)$$

where $I_k$ is the kth order modified Bessel function of the first kind. The Bit Error Rate (BER) curve based on Equation 2.8 is plotted in Figure 2.6.

Figure 2.6 also shows two experimentally measured curves of the prototype modulation scheme in an Additive White Gaussian Noise (AWGN) channel. The curve labelled as *"Uncoded BER with Butterworth Filtering"* is the actual performance of the prototype implemented. There is a considerable degradation of 6 dB as compared to the theoretical ideal curve. This degradation is primarily due to the substitution of the required Nyquist filters with $4^{th}$ order Butterworth filters. The required Nyquist filters were unavailable and the Butterworth filters are contained on the DSP cards. The effect of the Butterworth filter is to allow more noise to pass through to the receiver and cause ISI in comparison to the Nyquist case. As a result, the prototype will have worse performance since the SNR after the receiver filter will be less than the $E_s/N_0$ which would exist when employing a square root Nyquist filter. Through the use of a computer simulation, which used the Butterworth and Nyquist filters' bandwidths as parameters, it was found that the difference between the Butterworth and Nyquist case is approximately 5 dB. The curve labelled as *"Uncoded BER with Nyquist correction"* is a result of this correction factor. Note that this is an approximation, the true Nyquist correction factor must also account for the added ISI caused by the Butterworth filter. The prototype's corrected performance

is relatively close to the theoretically expected performance with a maximum degradation of 1 dB. This deviation is attributed to the following factors.

- The non-ideal signal space at the demodulator output, due to the imperfect RF components.
- The imperfect timing of the software controlled symbol synchronization signal.
- The ISI caused by the Butterworth filters.

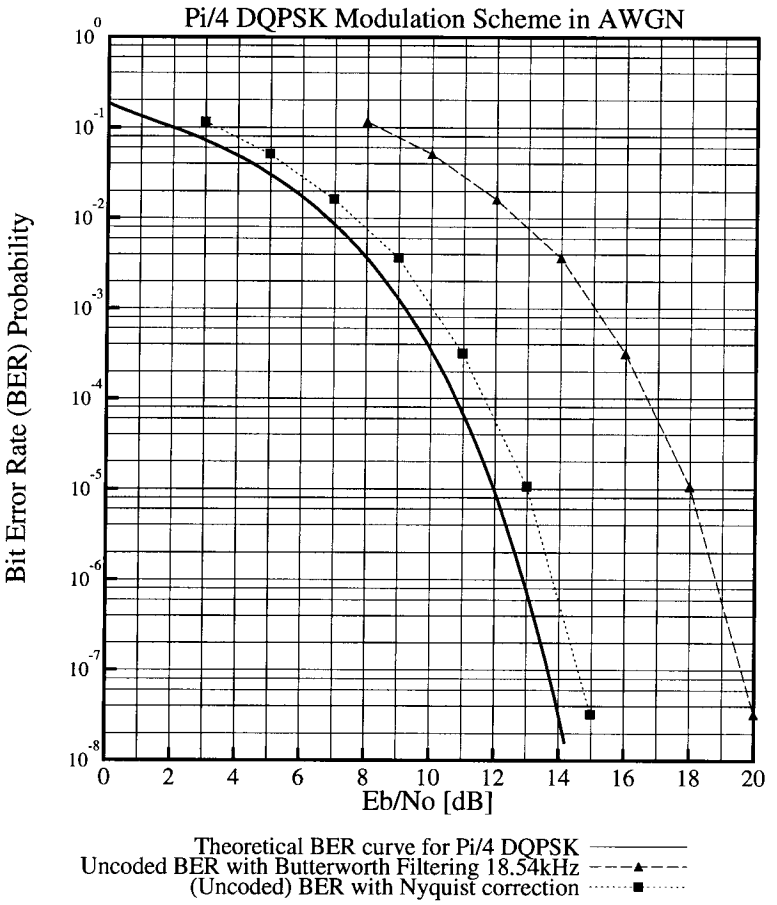


Figure 2.6 BER Performance in AWGN.

For convenience, all subsequent performance curves of the implemented system will be adjusted by this "Nyquist correction factor". This also holds for the coded case, since

the performance is plotted against the SNR level. The SNR level that would exist with the Nyquist case is just a simple adjustment as above.

Figure 2.7 presents the measured BER performance of the modulation system in a combined AWGN and Rayleigh fading environment with $B_DT$ equal to 0.0043, 0.0022, and 0.00084. The $B_DT$ products correspond to a $\pi/4$ shift DQPSK system operating with a carrier frequency of 900MHz, a baud rate of 19.2kBaud/s, and vehicle velocities of 100, 50, and 20km/hr respectively. Also shown in the graph, is the theoretical BER results for a static multipath fading channel. Static refers to the channel having a constant phase modulation (i.e., the receiver or vehicle is at rest). The experimental results are for vehicles in motion and therefore, are expected to be worse than the theoretical curve for a vehicle at rest. It is evident that the theoretical and experimental results are in close agreement until a residual error floor is established by the experimental curves. This error floor is a result of the random phase modulation caused by the doppler spread obtained from the vehicle being in motion. An increase in the doppler spread results in an increase in the level of the error floor. The experimental results are less than an order of magnitude higher than the computer simulated results of Feher [10] and Bouras [7]. This deviation is due to the imperfections in the modulation scheme and the hardware Rayleigh simulator, as well as the Receiver DSP Card clipping the input voltage waveforms of the I and Q channels to ±3 volts even though the amplitude periodically fluctuates beyond these limits.

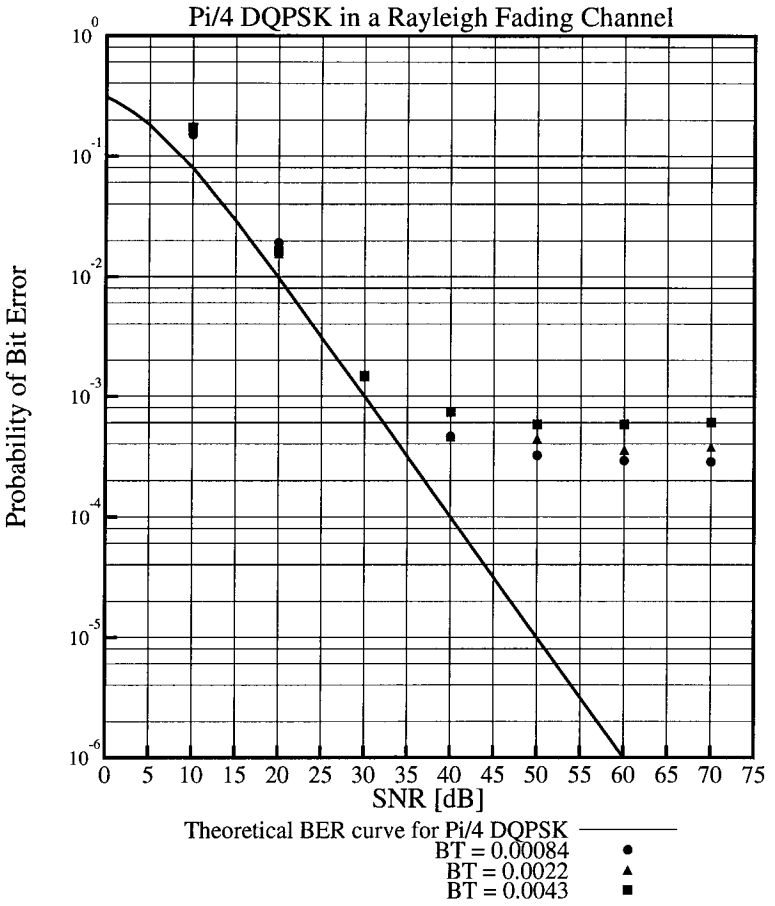


Figure 2.7 BER Performance of $\pi/4$–shift DQPSK in a Rayleigh Fading Channel for Various $B_D T$.

## Section 2.7 Conclusions

The operation of the prototype $\pi/4$ shift DQPSK system was verified through experimental measurements. The BER performance data obtained for the AWGN channel and the combined AWGN and Rayleigh Fading Channel were in very good agreement with the expected theoretical results illustrating the proper operation of the prototype modulation scheme.

# Chapter 3 Application of Complementary Punctured Convolutional Codes to a SW Type II ARQ Scheme

## Section 3.1 Introduction

Recently, Kallel has introduced a new class of punctured convolutional codes which are complementary [11]. In this Chapter we will briefly review Complementary Punctured Convolutional (CPC) Codes and their structure. Section 3 will present the generalized CPC SW Type II Hybrid ARQ algorithm, and Section 4 will discuss its specific implementation using DSP cards housed in an IBM PC. The performance of the implemented prototype will be compared to numerical and computer simulated models in Section 5.

## Section 3.2 Review of Complementary Punctured Convolutional Codes (CPC)

In general, a high rate *(b/N)* punctured convolutional code can be constructed from a rate $1/N_0$ mother code by periodically and selectively deleting $(bN_0–N)$ code bits according to a specific perforation pattern [12]. The function of deleting code bits is usually performed by the use of a perforation matrix which consists of $b$ columns and $N_0$ rows for a rate of $b/N$ punctured code. Each column is associated with one encoding cycle, and each row is associated with each coded bit stream from the $N_0$ modulo–2 adders of the $1/N_0$ encoder. The perforation matrix consists of ones and zeros which corresponds to transmitting and not transmitting code bits. An example, of a rate 3/4 punctured convolutional code of period 3 obtained from a rate 1/2 code is given by

$$P_1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}. \tag{3.1}$$

An equivalent punctured code can be obtained by likewise cyclically shifting the $N_0$ rows. At the most, this will yield $b$ distinct codes which have the same distance properties and error performance capabilities [13]. As a result, $P_2$, which is given by

$$P_2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \tag{3.2}$$

and $P_1$ are equivalent perforation matrices.

### 3.2.1 CPC Codes

Allow $P_i$, $i$=1,2, ..., $p$, to denote the perforation matrices of $p$ equivalent CPC codes of rate $b/N$ obtained from a rate $1/N_0$ mother code, where $p=\lceil \frac{bN_o}{N} \rceil$. The result of perforation matrix $P_i$ is code $CPC_i$. Define the matrix $P_{TOTAL}$ as

$$P_{TOTAL} = \sum_{i=1}^{p} P_i \, . \tag{3.3}$$

The $p$ equivalent codes $CPC_i$, $i$=1,2, ..., $p$, are said to be *Complementary* if every element of $P_{TOTAL}$ is greater than or equal to one. Note that for convenience the $p$ equivalent codes were denoted as $CPC_i$, but if they do not met the above restriction associated with $P_{TOTAL}$, they should not be referred to as $CPC_i$. The rate of $P_{TOTAL}$ is given by $b/(pN)$ which results in two possible cases. If $N = N_0$ , we have $p = b$ and the rate of $P_{TOTAL}$ will be $b/(bN) = 1/N_0$, which is the original mother code. On the other hand, if $N>N_0$ and $p<b$ matrices are chosen to satisfy Equation 3.3, then some elements of $P_{TOTAL}$ will be greater than one and the combined rate is $b/(pN)$. As an example, the two previous matrices $P_1$ and $P_2$ of rate 3/4 are combined to form $P_{TOTAL}$ and yield a resulting code rate of 3/8.

$$P_{TOTAL} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 1 \end{bmatrix}. \tag{3.4}$$

## Section 3.3 Generalized CPC SW Type II Hybrid ARQ Algorithm

Allow $P_i$, $i$=1, 2, ..., $p$, to denote the perforation matrices of $p$ CPC codes of rate $b/N$ obtained from a rate $1/N_0$ mother code, as discussed above. The result of perforation matrix $P_i$ is code $CPC_i$.

The scheme begins by appending $n_{dp}$ detection parity bits and $m$ tail bits, corresponding to the encoder's memory, to each *k-bit* data packet. The resulting sequence is encoded by the rate $1/N_0$ mother code and then punctured and transmitted according to the following algorithm [11].

1. *Level 1:* Puncture the sequence with $P_1$, resulting in packet **A** of code $CPC_1$ which is transmitted. The receiver decodes packet **A** using a rate $1/N_0$ Viterbi decoder and perforation matrix $P_1$. The error detection decoder checks the decoded sequence consisting of data bits and parity bits. If the sequence is declared error free, transmission of **A** is complete. Otherwise, the received sequence is stored for future decoding attempts and the algorithm moves up to the next level.
2. *Level i, 1< i <p:* Transmit packet **A** of code $CPC_i$ resulting from $P_i$. Initially, use Viterbi decoding with perforation matrix $P_i$. If the decoded sequence is declared error free, transmission of A is complete. Otherwise, reapply Viterbi decoding but on the combination of all $i$ sequences, previously stored up to this level, and using perforation matrix $P_{TOTAL}=P_1+P_2+...+P_i$. If the resulting sequence is declared error free, transmission of **A** is complete. Otherwise, the current sequence is stored and the algorithm moves to the next level.
3. *Level p:* Send packet **A** of code $CPC_p$. As above, initially decode using only the received sequence. If unsuccessful, decode using all $p$ sequences. If the resulting

sequence is still in error, discard the received sequence of code $CPC_1$ and the algorithm moves to the next level.

4. *Level (p+j), j=1,2,... :* Send Packet **A** of code $CPC_i$. Decode using the received sequence in conjunction with perforation matrix $P_i$. If unsuccessful, decode using all $p$ sequences. In the event that decoding is still unsuccessful, discard received sequence at *level j+1* and the algorithm moves to the next level.

It should be pointed out, that the above encoding and transmitting strategy did not discuss the implications of appending a flag and a header to packet **A**. In the event that a flag is not found in the implemented prototype, the receiver will time out, and the algorithm will reinitialize at the current level. In practice it is the transmitter which times out if it receives no response from the receiver. If a header failure is detected, the current packet is discarded and the algorithm also reinitializes at the current level. Since the transmitter and receiver DSP cards are contained in the same PC they are initialized and synchronized by the Host ARQ protocol.

## Section 3.4 DSP Implementation of a CPC SW Type II ARQ Scheme

The Stop and Wait Type II Hybrid ARQ Protocol is written in Borland C++ and resident on the host PC. The protocol behaves as discussed above with $p=2$ CPC codes of rate 3/4 from a rate 1/2 mother code. The two perforation matrices used by the DSP transmitter card for encoding the data packet are given by

$$P_1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}. \tag{3.5}$$

Figure 3.1 shows the physical block diagram of the prototype communication system. The protocol constructs the header and random data packet, places them in the Dual Access

Memory (DAM), and strobes the DSP transmitter card to send and the DSP receiver card to listen. The DSP transmitter card retrieves the header and data packets and encodes them according to the information placed in the header. Once the frame is constructed, it is transmitted through the channel to the DSP receiver card, which is contained in the same PC. The DSP receiver card processes the received frame and either places an acknowledgment (ACK) or negative acknowledgment (NACK) in the DAM and strobes the protocol. Once the protocol fetches the DSP receiver's reply, two events may occur. If an ACK was sent, the protocol will construct a new header and a new random data packet to place in the DAM. If a NACK was sent, the protocol will keep the data packet but construct a new header which indicates the new $P_i$ to be used for encoding the data packet. Note that if a frame is lost, the DSP receiver is equipped with a time-out feature which will result in a NACK.

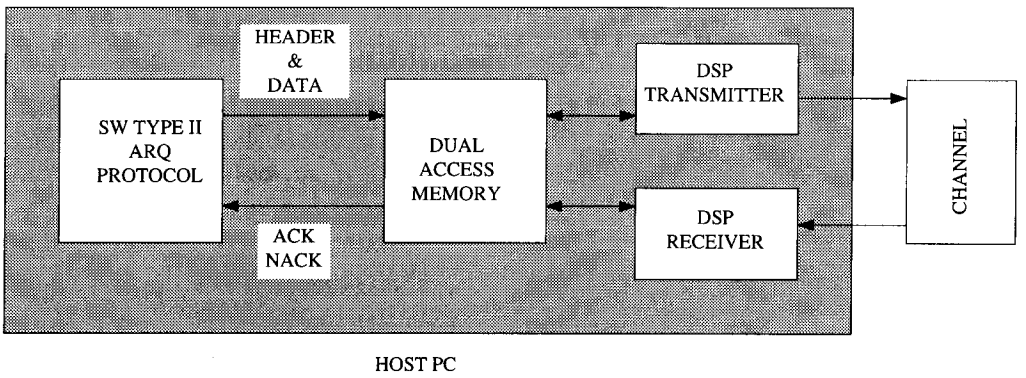


Figure 3.1 Block Diagram of Prototype SW Type II ARQ Scheme.

It is during the construction of the header, that the protocol decides which perforation matrix $P_1$ or $P_2$ to use for encoding the data packet based on the receiver's reply. Along with the NACK, the receiver sends the motive which may be either a *Header CRC Failure* or a *Data CRC Failure.* In the event of a header failure or lost frame (time-out), the protocol will not switch perforation matrices. In the event of a data failure, the protocol

alternates between $P_1$ or $P_2$. The result of this algorithm is to maximize throughput. This algorithm ensures that if a corrupted data sequence of code $CPC_1$ is received, the next data sequence received can only be encoded by $P_2$ and be of code $CPC_2$. If the data sequence of code $CPC_2$ is unsuccessfully decoded, it may be combined with the data sequence of code $CPC_1$ for subsequent decoding. The modulation scheme used by the SW Type II ARQ Protocol for transmission, is the $\pi/4$ Shift DQPSK discussed in detail in Chapter 2.

The following assumptions or simplifications are incorporated in the implemented prototype which consists of the DSP transmitter and receiver cards in the same Host PC under the control of the SW ARQ protocol.

- As a consequence of the transmitter and receiver DSP cards being in the same Host PC, they are initialized and synchronized by the ARQ Protocol running on the Host PC. In practice, there is an initialization and synchronization process to be executed by the independent transmitter and receiver.
- In practice a noisy return channel is used to send the receiver's reply. In the prototype, the receiver's reply is passed internally through the PC via the DAM. This is a noise free return channel.
- As a result of the ARQ protocol controlling both the transmitter and receiver, it is the receiver which times out if a flag is not found. Again, in practice it is the transmitter that times out if it does not get a response from the receiver.
- Symbol Synchronization is accomplished by hard wiring the transmitter and receiver. The actual symbol timing signal is software generated and is not ideal. A practical system would have the receiver utilize a Phase Locked Loop or some other synchro-

nization circuit to obtain symbol synchronization with no link to the transmitter.

These simplifications do not compromise the accuracy of the experimental results. The prototype is used to evaluate various FEC strategies which are unaffected by the above simplifications.

### 3.4.1 Frame Structure

The detailed structure of the frame used for transmission in the prototype system is illustrated in Figure 3.2. Excluding the preamble and flag, the maximum length the encoded frame may attain is 1024 bits. The frame begins with an 8 bit Symbol Sync Preamble. Since a Stop and Wait scheme is implemented, the channel will always be idle before a transmission and the preamble allows the receiver to realize symbol synchronization and stabilize before the remaining portion of the frame arrives. Immediately following the preamble is the Flag or Frame Sync, whose purpose is to present the receiver with a unique bit pattern so that the receiver may synchronize itself with the data stream's frame structure. The receiver is continuously hunting for the flag pattern and the actual procedure and choice of flag is investigated in the next section.

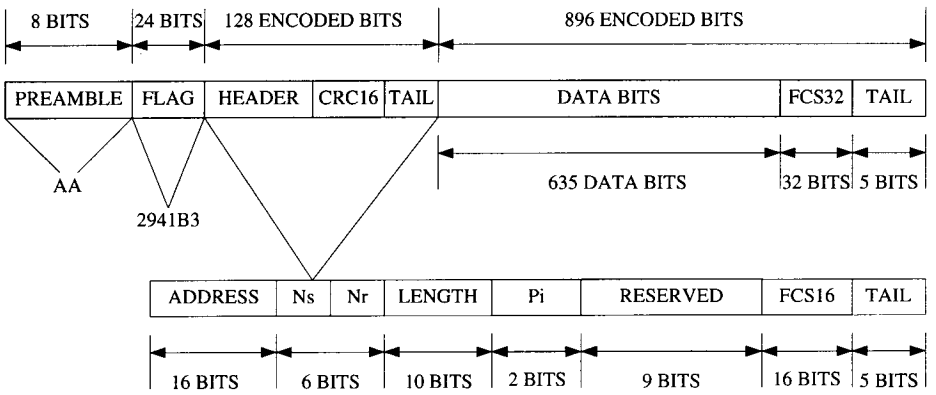


Figure 3.2 Detailed Structure of Frame.

Control information is contained in the 64 bit header, which includes a 16 bit Frame Check Sequence (FCS) and a 5 bit tail for decoding. The header's address field is used to identify the station that is to receive the frame. The next two fields, $N_s$ and $N_r$ are sequence numbers used to number the frames. The sequence numbers are not required for the operation of the prototype but has been included for future upgrading to a Selective Repeat scheme. The next field contains the length of the data packet following the header. The following field consists of two bits which indicate the perforation matrix $P_i$ used in the puncturing operation during the encoding of the data. Reserved is the next field which consists of 9 bits and is not used by the current version of the protocol. The remaining 16 bits represent the FCS which is a result of the generator polynomial CRC–CITT defined as $G_{16}(x) = x^{16} + x^{12} + x^5 + 1$.

The information or data bits are contained in the data packet of the frame. This consists of a maximum of 896 CPC encoded bits. As a result of using a perforation matrix which yields a rate of 3/4, the maximum number of information bits which the data packet can contain is $\frac{3}{4}(896) - 32 - 5 = 635$ bits. The length of the entire frame consisting of preamble, flag, header, and data packet is 1056 bits. The generator polynomial used for the FCS is the CRC32 given as $G_{32}(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$.

### 3.4.2 Frame Synchronization

The 24 bit flag, denoted in hexadecimal as 2941B3, is used by the receiver to synchronize itself with the data stream's frame structure. A good flag sequence has the property that the absolute value of its *correlation sidelobes* is small. A correlation sidelobe is the value obtained by correlating a flag sequence with a time-shifted version

of itself. Therefore, a correlation sidelobe value, $C_k$, for a $k$-symbol shift of a $N$ bit flag sequence $\{F_j\}$, is given by

$$C_k = \sum_{j=1}^{N-k} F_j F_{j+k} \quad , \tag{3.6}$$

where $F_i$ ($1 \leq i \leq N$) is an individual bit taking values of $\pm 1$, and the adjacent bits (associated with index values $i>N$) are assumed to be 0 [14]. The actual flag was found through the use of computer simulations.

Figure 3.3 shows the correlation sidelobes of the flag used in the prototype. The sidelobes are very low when compared to the main lobe of $C_0$, which yields a value of 24. This sidelobe profile ensures a very high probability that the receiver will find the exact starting point of the flag rather than a bit shifted version of it.

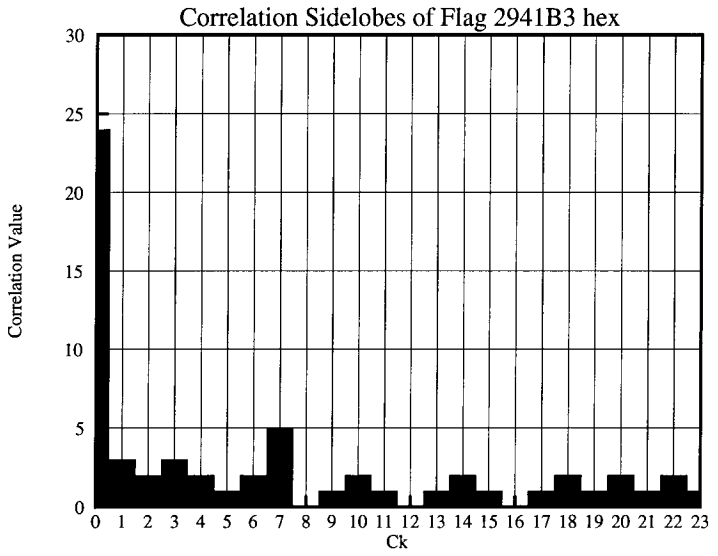


Figure 3.3 Correlation Sidelobes of Flag used in Prototype.

The following procedure is followed to allow the receiver to locate the flag. The receiver correlates the known flag pattern to the incoming data. If the incoming data does not contain a flag, the correlation value will be low. On the other hand, when a flag is

encountered the correlation value will be very high. The correlation value, $C$, for a 24 bit flag pattern $\{F_j\}$ and a 24 bit data sequence $\{D_j\}$ is given by

$$C = \sum_{j=1}^{24} F_j D_j \quad , \tag{3.7}$$

where $F_j$ and $D_j$ take on values of +1 or −1 representing bits 1 and 0 respectively. The maximum value of $C$ is 24 which indicates a flag with 0 bit errors has been located. The prototype compares $C$ to a user set *threshold* value which limits the number of bit errors which will be accepted in the flag and still ensure frame synchronization (i.e., a *threshold* value of 16 indicates that 20 bits of the data sequence match the flag pattern).

The optimum *threshold* value was found through experimentation. For each SNR tested, a 1000 uncoded frames were sent to the receiver whose correlation *threshold* value was altered over the range of 10 to 24. Referring to Figure 3.3, it is seen that the highest sidelobe has a value of 5. A starting point for the *threshold* value is to take twice the highest sidelobe value which is 10. Figure 3.4(a) illustrates that the probability of a bit error is relatively equal for *threshold* values of 24 to 10. However, lowering the *threshold* value below 10 results in the prototype operating very slowly because it must process a large number of false flags. The lower the *threshold* value, the larger the amount of false flags that the prototype must process. Figure 3.4(b) shows the percentage of flags successfully found given the different *threshold* values. It is seen that the lower the *threshold* value, the greater the success of finding the flags. Another observation is that decreasing the *threshold* value below 12 has a marginal affect on the flag success rate. A balance must be found in which a threshold value that gives a good flag success rate does not burden the prototype with false flags. The two curves representing *threshold* values of 10 and 12 give the best success rates and are relatively equal. It is obvious from

the two graphs that a *threshold* value of 10 or 12 is optimum. These values yield the best flag success rate with the least amount of false flags to be processed by the prototype.

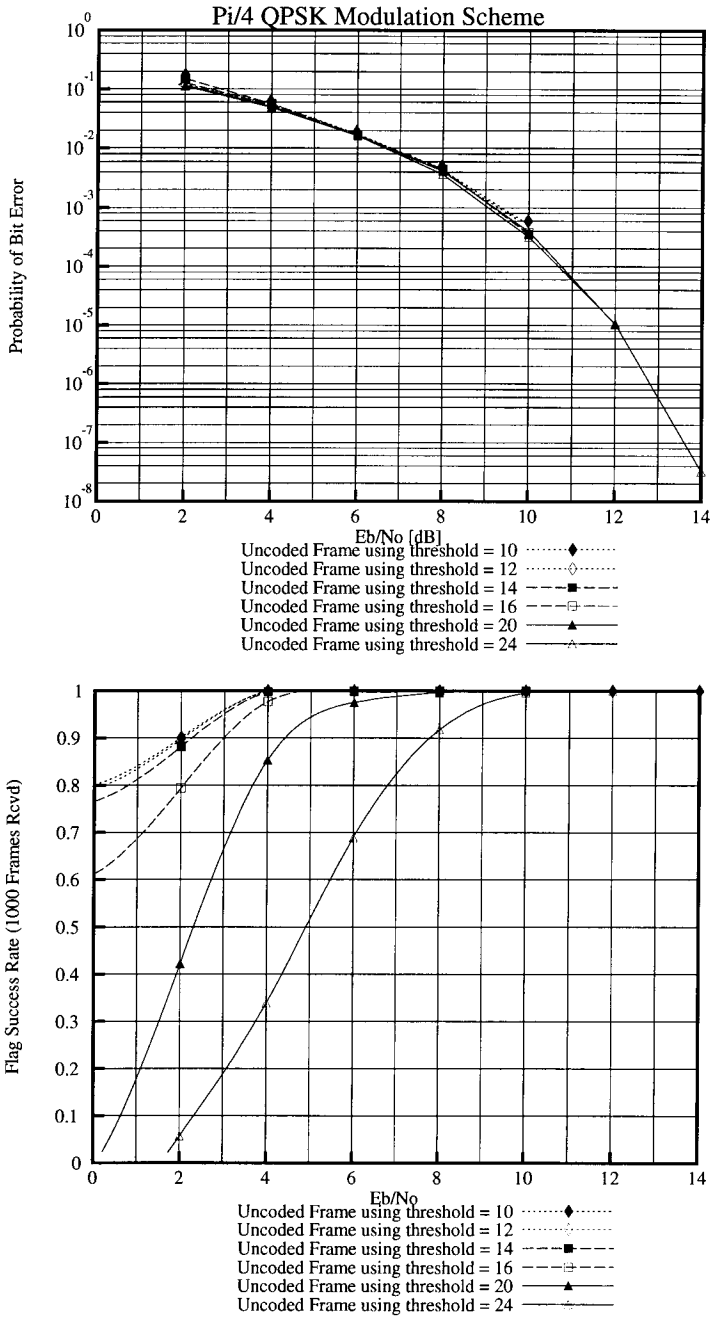


Figure 3.4 (a) and (b) Effects of Changing Threshold value used for Flag Correlation

By comparing the correlation value, $C$, to a *threshold* value, a certain number of false flags will be located. The prototype receiver implemented is "smart" enough to eliminate the majority of false flags. When a flag is located, the header is immediately decoded and two events may occur.

1. If the header fails the CRC check, the next subsequent flag is located and the new header is decoded. If the CRC check fails again, the process repeats itself until the header CRC is passed.
2. If the header passes the CRC check, the length of the frame is obtained and all the false flag occurrences falling within the range of the frame are ignored.

By using this simple procedure a very large majority of the false flags are ignored.

### 3.4.3 Encoder/Transmitter DSP Card

The Encoder/Transmitter DSP Card contains the following C software modules:

- **CRC Encoder** is responsible for calculating the Frame Check Sequence (FCS) bits and is able to use generator polynomials up to 32 bits.
- **Rate 1/2 Convolutional Encoder** outputs two data streams representing the two modulo–2 adders of the encoder. A simple module named **Combine** is required to interleave the two outputs of the adders. The two generator polynomials are $G_1(x) = x^4 + x^3 + 1$ and $G_2(x) = x^4 + x^2 + x^1 + 1$ and are user configurable.
- **Puncture Module** individually punctures the two data stream outputs of the rate 1/2 convolutional encoder. The module punctures according to the perforation matrix $P_i$ which is chosen by the host SW ARQ protocol. **Combine** is required in order to interleave the two punctured outputs of the encoder adders.

- **Block Interleaver** accepts the coded symbols in 128, 256, or 512 bit blocks. The interleaver may be visualized as a rectangular array of $I$ rows and $n$ columns. The encoded symbols are read into the array by rows and read out by columns. The vertical dimension of the array, $I$, is called the interleaving degree and is user configurable by selecting values of 4, 8, and 16. The prototypes tested used an interleaving degree of 16.
- **Queueing Module** manages an 8 slot queue and is responsible for beginning and terminating the operations of the $\pi/4$ shift DQPSK baseband generator.

It is the main program written in DSP Assembly language which utilizes the above software modules and provides the encoding and transmitting services required by the host protocol. Figure 3.5 is a detailed description of the self explanatory procedure followed by the main program to encode and construct a frame. The two final operations not shown would be to interleave the frame and place it in the queue for transmission. The header and data are fetched from the Dual Access Memory.

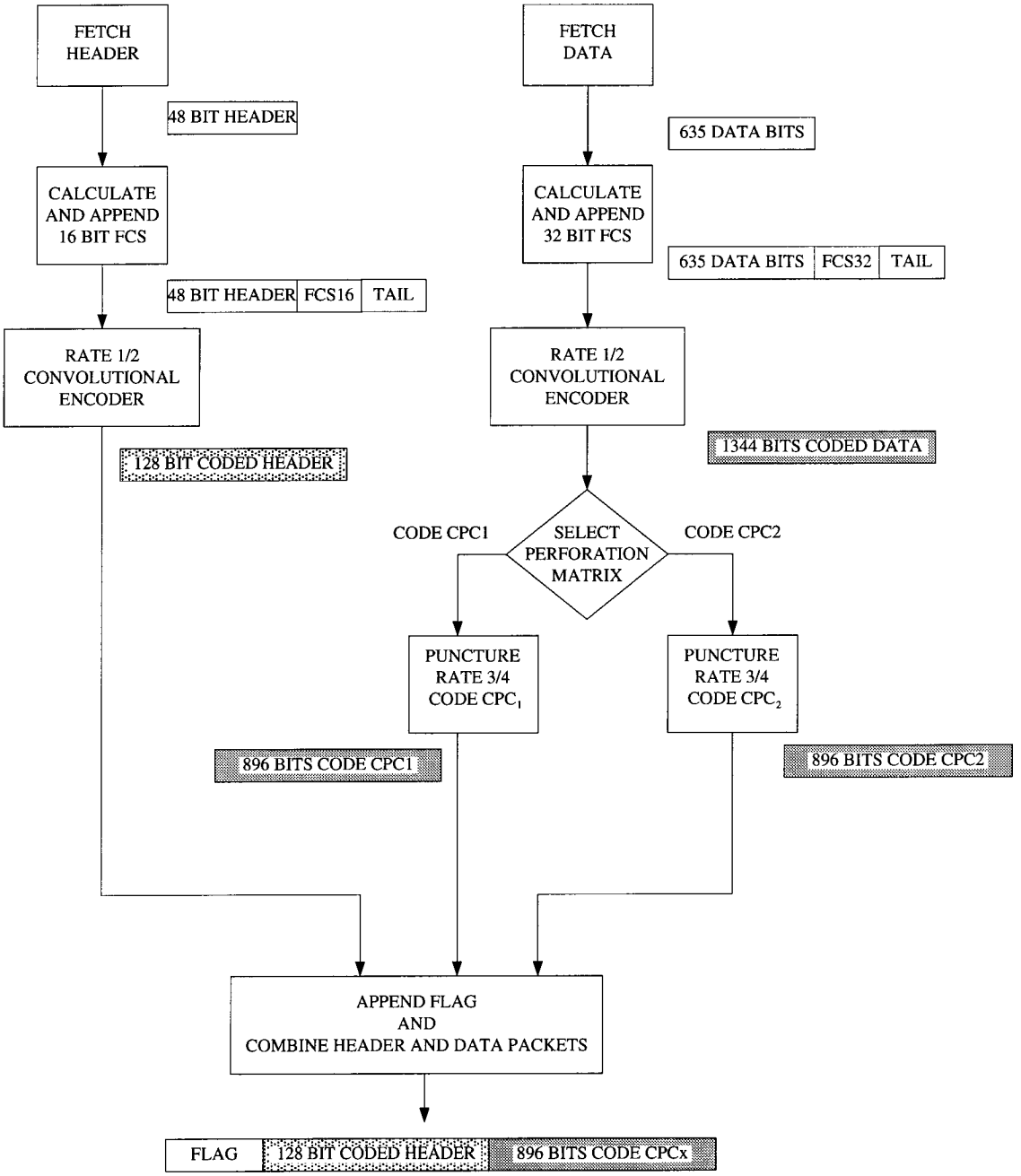


Figure 3.5 Frame Encoding and Construction Algorithm of DSP Transmitter Card.

### 3.4.4 Receiver/Decoder DSP Card

The Receiver/Decoder DSP card contains the following DSP Assembly software modules:

- **Flag Correlator** is used to locate the occurrence of a flag in a data stream according to a user set threshold value. Section 3.4.2 gives a detailed explanation of this software module.
- **Transform** is responsible for transforming the soft $\pi/4$ shift DQPSK data to soft QPSK data and as a result eliminate the dependency between neighboring symbols. Section 2.5 discusses this transformation and its results.
- **Soft Data Deinterleaver** is required to deinterleave the soft QPSK data. This module operates on soft data as compared to its inverse module **Block Interleaver** which operates on hard data.
- **CRC Encoder** is the same module used by the transmitter DSP card. The difference is that the calculated Frame Check Sequence (FCS) is compared to the received FCS in the decoding mode.
- **Data Sequence Combiner** is responsible for combining soft data sequences of different codes, such as $CPC_1$ or $CPC_2$, to form a more powerful code for error correction purposes.
- **Rate 1/2 Soft Decision Viterbi Decoder** is utilized to decode the header and data according to the perforation matrix used in the encoding process.
- In the CPC SW Type II ARQ scheme with *code combining*, an additional module called **Code Combining**, which optimally combines data sequences of equal codes such as $CPC_1$, is required.

Figure 3.6 is a detailed flow chart of the Receiver/Decoder DSP algorithm. The algorithm is a direct result of the general scheme presented in Section 3.3 with $p$=2 CPC codes of $CPC_1$ and $CPC_2$. As shown in Figure 3.6, the replacement of a module is necessary in order to incorporate *code combining*. Rather than simply save the most current corrupted data sequence of code $CPC_1$ or $CPC_2$, the module combines the current sequence with all previous corrupted sequences of the same code for further subsequent decoding.

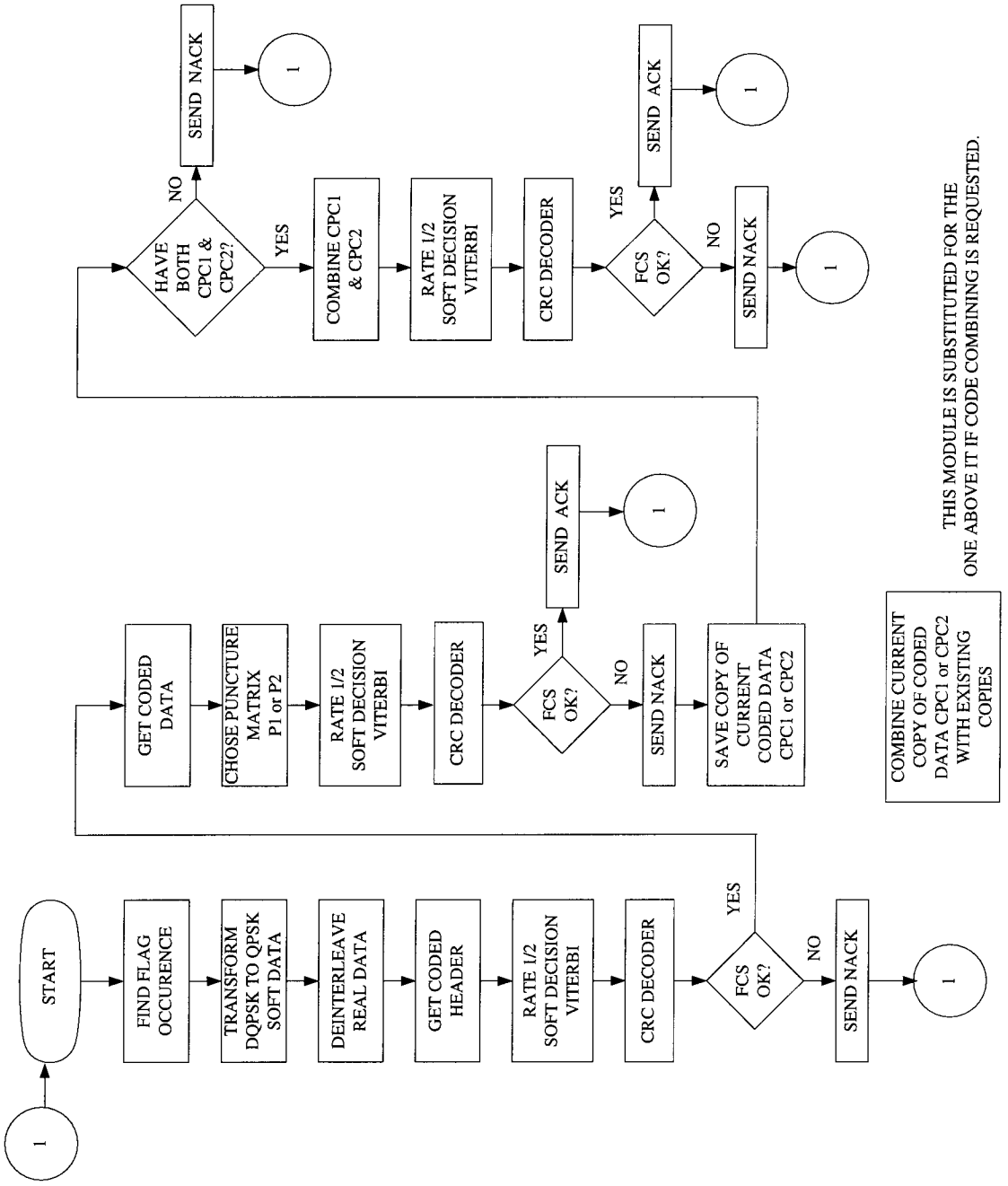


Figure 3.6 Frame Decoding Algorithm of DSP Receiver Card.

**Viterbi Decoder** A 16 state rate 1/2 soft decision maximum likelihood Viterbi Decoder is the heart of the receiver. It is entirely written in DSP Assembly Language for speed and efficiency. The soft decision decoding scheme makes use of past information bit history and a metric function to decode the incoming data. It follows, that the performance of the Viterbi decoder is primarily influenced by the choice of path history length and the metric function. It is common practice to select a path history length equivalent to four or five times the constraint length of the encoder which results in negligible degradation from the optimum decoder performance [14]. In the case of the prototype, the constraint length is 5 and the path history length utilized is 32 information bits. The Viterbi decoder operates on soft QPSK data which is the product of the transformation of soft $\pi/4$ shift DQPSK data. The metric chosen is the Euclidean distance based on the signal constellation of the QPSK signals. The Euclidean distance is defined as

$$D = \sqrt{(X_C - X_R)^2 + (Y_C - Y_R)^2}, \tag{3.8}$$

where $X_C$ and $Y_C$ are the coordinates of the signal on the constellation for QPSK and $X_R$ and $Y_R$ are the coordinates of the received data. Calculating the metric as defined in equation 3.8 is a very tedious and time consuming operation. The square root operation is not performed, and although it is not a linear function, distance values without the square root operation work well because the relationship between x and $\sqrt{x}$ is one-to-one and monotonic. To further simplify 3.8, one may expand the brackets and discard the squared terms to yield

$$D = X_C X_R + Y_C Y_R. \tag{3.9}$$

There is a considerable amount of time saved in calculating 3.9 as opposed to 3.8.

Once the Viterbi decoder is initialized, it will keep track of 16 surviving paths through the trellis. As depicted in Figure 3.7, at each new decoding instant, each survivor leads to two new states or paths, thereby yielding a total of 32 new paths. The decoder calculates the branch metrics $\beta$ and $\gamma$, related to the two new states, and then adds them to the accumulated metric $\alpha$ resulting in new accumulated metrics of $\alpha$+$\beta$ and $\alpha$+$\gamma$. The smallest new accumulated metric will be chosen as the new surviving path.

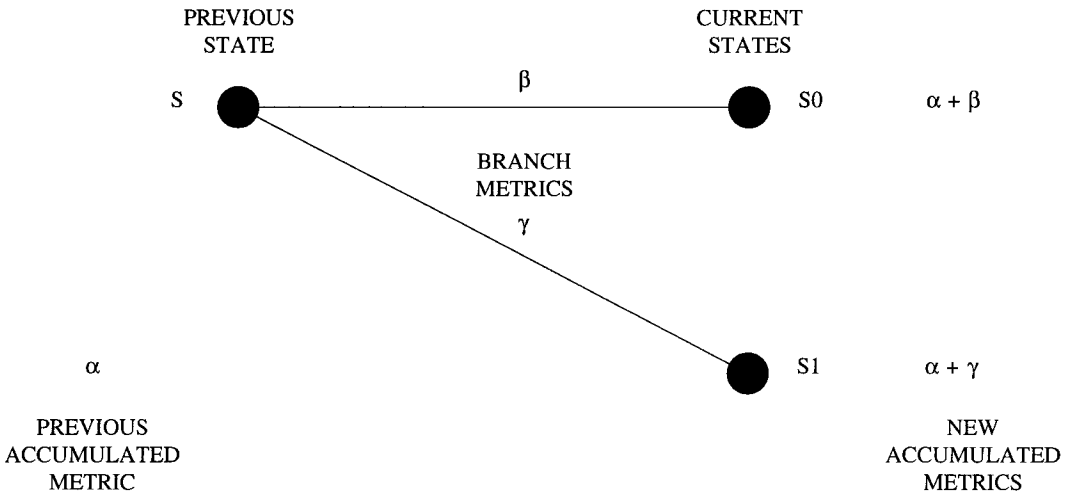


Figure 3.7 Choosing a Path Survivor.

In practice, it is not possible to continue to accumulate the metric distances without encountering an overflow problem. Therefore, a weighted accumulation method is used to determine the accumulated metric and is given as

$$D_{new} = \beta D_{old} + (1 - \beta) D_{branch}, \tag{3.10}$$

where 0<<$\beta$<1 denotes the weighting factor, $D_{branch}$ is the branch metric, and $D_{new}$ and $D_{old}$ are the new and old accumulated distances respectively. This ensures that the new accumulated metric is bound. The value of $\beta$ is a performance parameter which is chosen to be 0.98 in the implemented Viterbi decoder.

**Numerical Analysis** Given the free distance $d_{free}$ and the distance spectra $a_d$ and $c_d$, where $a_d$ is the number of incorrect paths of Hamming weight $d$ that diverge from the correct path and remerge with it sometime later, and $c_d$ denotes the total number of bit errors in all the paths having Hamming weight $d$, the probability of a bit error for Viterbi decoding is upper bounded [15] by

$$P(B) \leq \sum_{d=d_{free}}^{\infty} c_d P_d. \tag{3.11}$$

$P_d$ is the probability that a wrong path at distance $d$ is selected and depends only on the channel and modulation scheme used [9].

For an AWGN channel and $\pi/4$ shift DQPSK, $P_d$ may be obtained as follows. The probability of a binary digit error for four-phase signalling over $L$ statistically independent AWGN channels is given by [16] as

$$P_{4b}(e) = e^{\frac{-2E_bL}{N_o}} \left\{ \begin{array}{l} \sum_{k=0}^{\infty} \left(\sqrt{2}-1\right)^k I_k\left(\frac{\sqrt{2}E_bL}{N_o}\right) - \frac{1}{2} I_0\left(\frac{\sqrt{2}E_bL}{N_o}\right) \\ \\ + \sum_{n=1}^{L-1} C_n \left[\left(\sqrt{2}+1\right)^n - \left(\sqrt{2}-1\right)^n\right] I_n\left(\frac{\sqrt{2}E_bL}{N_o}\right) \end{array} \right\} \tag{3.12}$$

$$\text{where } C_n = \frac{1}{2^{2L-1}} \sum_{k=0}^{\infty} \binom{2L-1}{k}.$$

$P_d$ is the probability that a wrong path at distance $d$ is selected and may be obtained from Equation 3.12 by substituting $d$ for $L$. Using 3.11 and 3.12 with the substitution, an upper bound for the performance of the rate 1/2 Viterbi decoder was calculated. Figure

3.8 depicts the resulting upper bound using a rate 1/2 code with weight spectrum given by Table 3.

| Rate | Generator Polynomials | $d_{free}$ | ($a_{dfree+j}$, j=0, 1, ...4) {$c_{dfree+j}$, j=0, 1, ...4} |
|---|---|---|---|
| 1/2 | 23, 35 | 7 | (2, 3, 4, 16, 37) {4, 12, 20, 72, 225} |

Table 3 Distance Spectrum of Code with Rate 1/2.

**Computer Simulation** A C computer simulation was used to verify the prototype Viterbi decoder's performance. The computer model simulates the prototype which uses a $\pi/4$ shift DQPSK modulation system with the receiver transforming the soft DQPSK data to soft QPSK data for decoding purposes. Figure 3.8 shows the BER curve resulting from the computer simulation. As a result of transmitting $10^6$ bits for each SNR level tested, the BER curve is accurate for points above $10^{-5}$. The simulation BER curve is below the upper bound curve for all accurate SNR levels tested.

**Viterbi Decoder Performance** Figure 3.8 illustrates the probability of a bit error for the Viterbi decoder implemented. For each SNR level tested, the Viterbi decoder processed $10^7$ bits. As is evident, the prototype curve is slightly worse than the simulation curve but close to the upper bound curve. This is expected since the simulation cannot take into account implementation losses. The small deviation between the simulated and prototype curves is due to the imperfect modulation system and synchronization timing. The rate

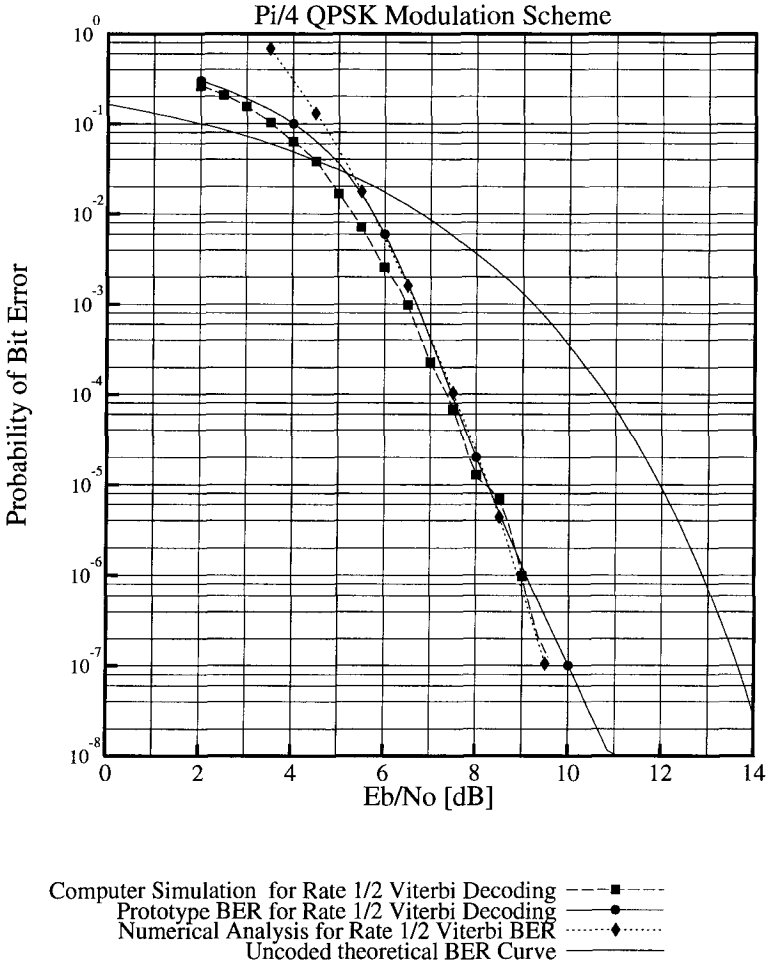


Figure 3.8 Rate 1/2 Soft Decision Viterbi Decoder Performance.

1/2 soft decision Viterbi decoder implemented operates as expected and its performance is verified by the computer simulation and upper bound curves.

## Section 3.5 Prototype Performance

In this section the throughput performance of the prototype CPC SW Type II ARQ system in AWGN is compared to the ideal numerical results. The prototype's throughput performance in a Rayleigh fading channel is also presented and discussed.

### 3.5.1 Throughput Analysis

The throughput $\eta$ is defined as the average number of accepted information bits per transmitted channel symbol and has a maximum possible value of 2 for DQPSK modulation. In general, $\eta$ may be defined as $R/\bar{N}$, where $R$ is the code rate and $\bar{N}$ is the average number of packets transmitted per correctly decoded packet. If the error detection parity bits along with the overhead of the header and flag are taken into account, the resulting throughput is

$$\eta = \frac{R}{\bar{N}} L_{ED} L_{OH} , \tag{3.13}$$

$$\text{where} \quad L_{ED} = \frac{k}{k + n_{dp} + m},$$

$$\text{and} \quad L_{OH} = \frac{(k + n_{dp} + m)\frac{1}{R}}{\frac{1}{R}(k + n_{dp} + m) + h + f}.$$

The factor $L_{ED}$ is the loss in throughput due to the addition of parity bits $n_{dp}$ and the tail of $m$ known bits. The factor $L_{OH}$ is the loss in throughput as a result of the overhead incurred by the frame for appending a rate 1/2 header, $h$, and a flag, $f$, to each block of $k$ information bits. The average number of packets transmitted per correctly decoded packet, $\bar{N}$, for a CPC SW Type II ARQ scheme is given in [11] as

$$\bar{N} \leq \left(1 + \sum_{i=1}^{p-1} Pr\{D_d(i)\}\right) \frac{1}{1 - Pr\{D_d(p)\}} , \tag{3.14}$$

where $D_d(j)$ is the event {decoded sequence obtained by combining $j$ equivalent codes, is detected in error}. As in [11], $Pr\{D_d(j)\}$, assuming the undetected error probability is negligeable, is bounded as

$$Pr\{D_d(j)\} \leq 1 - (1 - P(E))^l , \tag{3.15}$$

where $P(E)$ is the error event probability of Viterbi decoding with a code obtained by combining $j$ equivalent CPC codes (i.e., $CPC_1+CPC_2+...+CPC_j$) and where $l$ is the number of trellis level ($l=(k+n_{dp})/b$).

$P(E)$ is bounded as [15],

$$P(E) \leq \sum_{d=d^j_{free}}^{\infty} a_d^j P_d \quad , \tag{3.16}$$

where $P_d$ is the probability that a wrong path at distance $d$ is selected, and where $d_{free}{}^j$ and $a_d{}^j$ are the free distance and weight spectra of the code obtained by combining $j$ equivalent CPC codes. $P_d$ is dependent on the channel and modulation scheme employed [9].

**Numerical Results** Table 4 contains the distance spectra for the rate 3/4 punctured convolutional code used in the CPC SW Type II Scheme. $P_d$ is given in Equation 3.12,

| Code | Perforation Matrix | $d_{free}$ | $(a_{dfree+j}, j=0,1..5)$ |
|---|---|---|---|
| $CPC_1$ | $\begin{bmatrix}1 & 0 & 1\\ 1 & 1 & 0\end{bmatrix}$ | 3 | (1, 2, 23, 124, 576, 2852) |
| $CPC_2$ | $\begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 1\end{bmatrix}$ | 3 | (1, 2, 23, 124, 576, 2852) |
| $CPC_1 + CPC_2$ | $\begin{bmatrix}2 & 1 & 1\\ 1 & 2 & 1\end{bmatrix}$ | 8 | (1, 4, 3, 11, 18, 38) |

Table 4 Distance Spectra of Rate 3/4 Punctured Convolutional Code of Memory m=4.

where $d$ is substituted for $L$. Using the values in Table 4 and Equations 3.13, 3.14, 3.15,

and 3.16 a lower bound on the throughput for an AWGN channel with $\pi/4$ shift DQPSK modulation can be calculated. The resulting lower bound is plotted in Figure 3.9.

### 3.5.2 Experimental Throughput

The rate 3/4 CPC SW Type II ARQ scheme is tested over several SNR levels by executing the scheme until 1000 frames are successfully delivered. The resulting throughput is plotted in Figure 3.9 along with the previously calculated lower bound. Note that the throughput, which is the average number of information bits accepted per symbol, can be greater than one. This is a consequence of using $\pi/4$ shift DQPSK which has a maximum throughput of 2 information bits per accepted symbol. For medium to

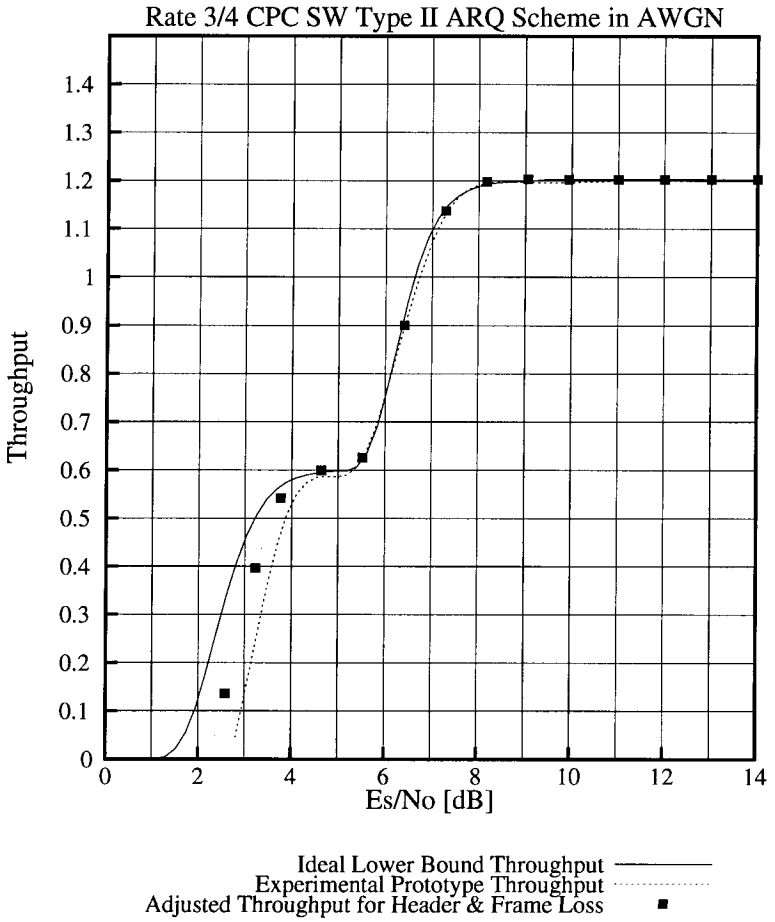


Figure 3.9 Numerical and Experimental Throughputs.

high SNR levels, the experimental curve and the lower bound are in good agreement. This is expected, since the Viterbi BER curve plotted against its upper bound is also in good agreement. At low SNR levels, the prototype throughput has a maximum degradation of 1dB. The calculated lower bound does not take into account header failures or lost frames. Whereas when the prototype encounters a lost frame or header failure, the entire data packet is discarded and taken into consideration for the throughput calculation. If header failures and lost frames are accounted for, the throughput of the system in question will suffer a decrease. To further prove this point, Figure 3.9 also plots a curve labelled as *"Adjusted Throughput for Header & Frame Loss"*. This curve is obtained by ignoring lost and header damaged frames in the prototype system. Recall, that the receiver is capable of transmitting a NACK which indicates whether the frame had a header failure or data failure. The transmitter keeps track of the type of NACKs, as well as the lost frames (time-outs). It is this information which is used to adjust the throughput for header failure and frame loss. It is clear that this adjusted curve is in good agreement with the lower bound with slight degradation at low SNR levels resulting from implementation losses which are critical at lower SNR levels. The scheme is able to correct a certain number of errors. At medium to high SNR levels, the scheme easily corrects the channel errors as well as the errors associated with the implementation losses. At low SNR levels, the number of channel errors in addition to the implementation loss errors places a load on the scheme and results in a negligeable degradation of 0.5dB (maximum) from the lower bound curve. The implementation losses are factors such as:

- imperfect symbol synchronization,
- non-ideal modulator and demodulator, and

- ISI from the Butterworth filtering.

It is clearly evident that since the prototype rate 3/4 CPC SW Type II ARQ scheme is in very good agreement with the lower bound, it is correctly operating and behaves as expected.

### 3.5.3 Rayleigh Fading Channel

The throughput of the prototype rate 3/4 scheme was also investigated in the combined AWGN and Rayleigh fading channel environment. The measurements were obtained for three $B_DT$ products of 0.0043, 0.0022, and 0.00084. These $B_DT$ products correspond to a $\pi/4$ shift DQPSK system operating with a carrier frequency of 900MHz, a baud rate of 19.2kHz, and vehicle velocities of 100, 50, and 20km/hr respectively. The throughput curves are plotted in Figure 3.10.

For comparison purposes, a lower bound on the throughput for a combined AWGN and a static multipath fading channel is also plotted. The lower bound is calculated in the same fashion as before, by using Equations 3.13, 3.14, 3.15, and 3.16. The probability of a binary digit error for four-phase signalling over $L$ statistically independent AWGN with static multipath fading is given by [16] as

$$P_{4b}(e) = \frac{1}{2}\left[1 - \frac{\mu}{\sqrt{2-\mu^2}} \sum_{k=0}^{L-1} \binom{2k}{k} \left(\frac{1-\mu^2}{4-2\mu^2}\right)^k\right],$$

$$\text{where } \mu = \frac{\bar{\gamma}_c}{1+\bar{\gamma}_c}, \tag{3.20}$$

and $\bar{\gamma}_c$ is the average received SNR.

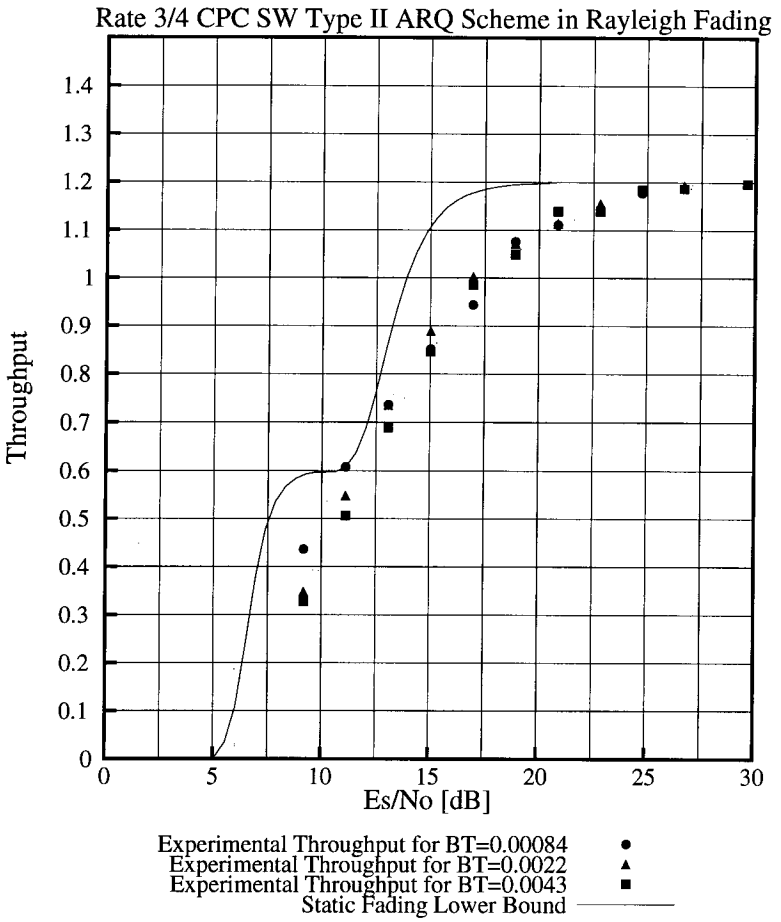


Figure 3.10 Throughput of Prototype in a Rayleigh Fading Channel.

As before, $P_d$ is obtained from equating 3.20 by substituting $d$ for $L$. The resulting lower bound is for a static multipath fading channel. The term static refers to the phase modulation of the multipath channel being constant (i.e., the receiver or vehicle being at rest). It is obvious that the three throughput curves obtained for the various vehicle speeds should be worse than the lower bound since the vehicle is not at rest. When the vehicle is in movement, the Doppler spread causes random phase modulations which in turn is responsible for the existence of residual error floors in the bit error rate as discussed in Section 2.6. In effect, the lower bound may actually be viewed as an *upper bound* when it is being compared to the prototype throughput at various vehicle speeds.

## Section 3.6 CPC SW Type II ARQ Scheme with Code Combining

The upgrading of the CPC SW Type II ARQ scheme to accommodate code combining is very simple. Only the receiver must be modified by the replacement of ten lines of DSP Assembly Language code. The new code or module ensures that the most currently received corrupted data sequence of code $CPC_1$ or $CPC_2$ will be combined with all previous corrupted copies of the same code (if the copies exist). The non–code combining scheme simply discards the previous copy of the corrupted data sequence once a new data sequence is received. It has been shown that code combining will increase the throughput of the scheme at low SNR levels [3].

Figure 3.11 illustrates the experimental results for the rate 3/4 CPC SW Type II ARQ scheme with and without code combining. As expected, the code combining case resulted in an increase in throughput to a maximum of 1dB. If the code combining curve is adjusted for header failure and lost frames, it is expected to perform better than the ideal Type II lower bound curve. Recall, that the Type II lower bound curve does not take into account lost or header damaged frames. Figure 3.11 also displays the *"Adjusted Throughput for Header & Frame Loss with Code Combining"*, which as expected has a substantial performance gain in throughput in comparison to the ideal Type II lower bound. To further verify the code combining scheme, measurements counting the number of frames transmitted to successfully deliver each of the 1000 frames at a certain SNR level were accumulated. Figures 3.12(a) and 3.12(b) are histograms representing the accumulated data for the non–code combining and code combining cases at a SNR level of 3.32dB. In comparing the two histograms, it is evident that the code combining case requires fewer transmitted frames to successfully deliver a frame since it is constantly combining data

sequences. This results in the number of transmitted frames being concentrated toward the lower end of the histogram, as opposed to the non-code combining case where the number of transmitted frames are spread out. These experimental results verify the correct operation of the code combining scheme.

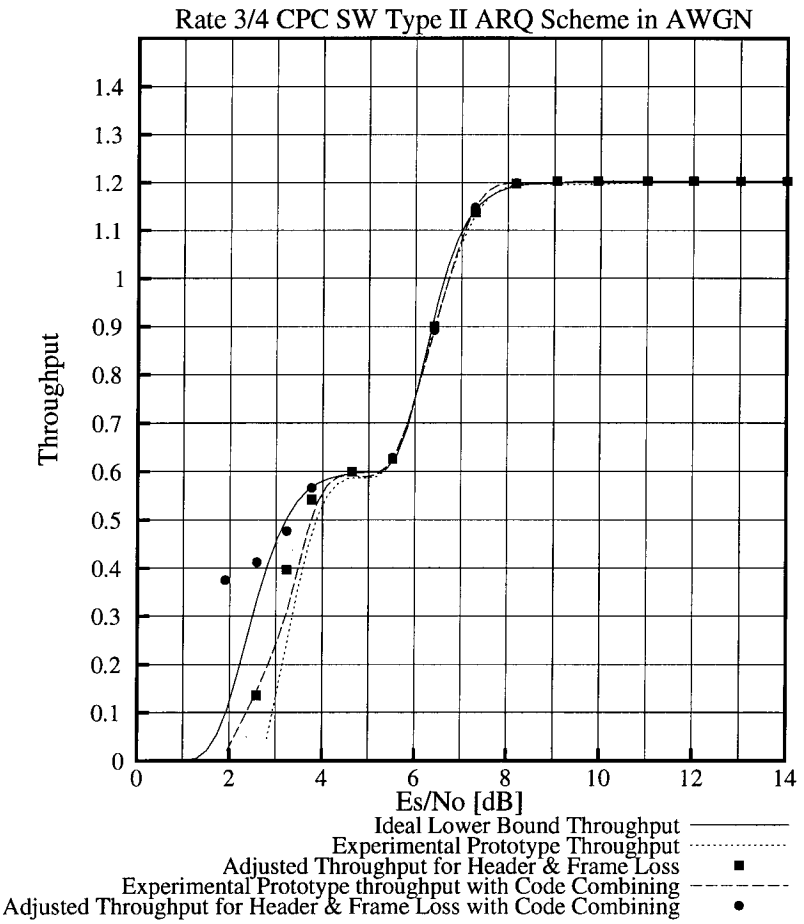


Figure 3.11 Throughput of CPC SW Type II ARQ Scheme with and without Code Combining

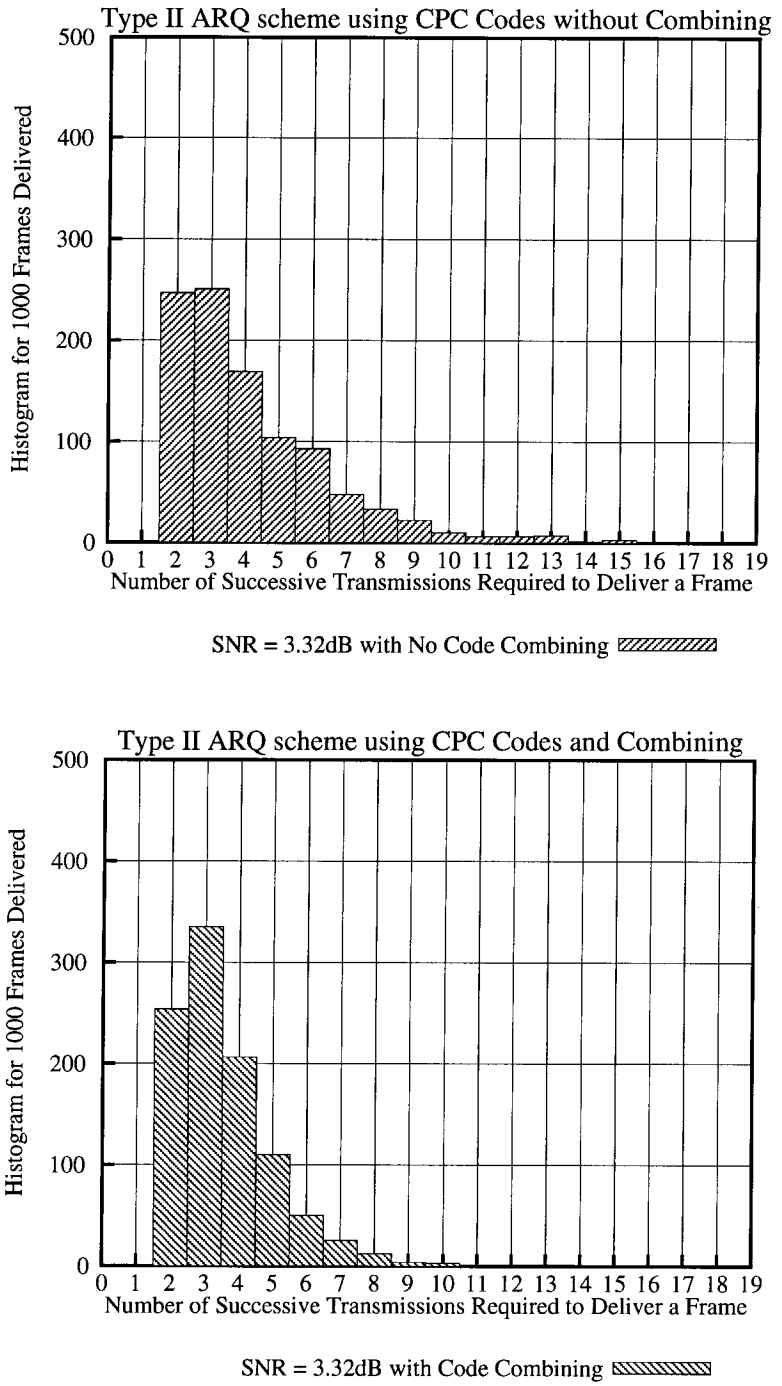


Figure 3.12 Histograms for Rate 3/4 CPC SW Type II ARQ with and without Code Combining

## Section 3.7 Conclusions

A prototype rate 3/4 CPC SW Type II ARQ scheme of memory $m$=4 was implemented utilizing a host IBM PC, two TMS320C30 DSP cards, an existing RF modulator/demodulator, and an existing channel simulator. The rate 1/2 soft decision Viterbi Decoder was thoroughly tested in section 1 and behaved as expected according to both computer simulations and numerical results. The throughput of the prototype was experimentally measured for both an AWGN channel and a combined AWGN and Rayleigh Fading channel. The experimental results for the AWGN channel were in very good agreement with the numerical results. In the case of the combined AWGN and Rayleigh channel, the throughput curves were referenced to numerical results obtained for a static multipath fading channel. The experimental curves behaved as expected indicating proper operation of the prototype.

When code combining was added to the prototype, the throughput at lower SNR levels increased. There is no extra cost associated with upgrading the prototype to a code combining scheme. It only requires the replacement of ten lines of DSP Assembly Language code. The code combining prototype was also verified for proper operation by comparing the histograms at certain SNR levels which counted the number of transmissions required to successfully deliver a frame.

The comparison of the experimental data of the prototype's performance to the numerical results clearly validate the proper and correct operation of the implemented scheme.

# Chapter 4 An Adaptive SW Type II ARQ Scheme

## Section 4.1 Introduction

The previous chapter illustrated how the CPC SW Type II ARQ scheme utilizing code combining achieved an increase in throughput at low SNR levels as compared to the same scheme without code combing. This chapter will focus on increasing throughput at all SNR levels by employing an adaptive coding rate to the CPC SW Type II ARQ scheme. The adaptive scheme uses Channel State Information (CSI) to decide which coding rate is the most appropriate to encode the data packet. Section 2 will present the algorithm used to adapt the coding rate to the AWGN or combined AWGN and Rayleigh channel. Section 3 will discuss the necessary software modifications to the existing DSP Assembly code and host IBM Protocol software. Section 4 will present the performance of the adaptive scheme for both the AWGN and combined AWGN and Rayleigh channel. Finally, all three implemented variations of the CPC SW Type II ARQ scheme will be compared and discussed.

## Section 4.2 The Adaptive Coding Rate Algorithm

A very simple and effective algorithm is used to select the current coding rate of the adaptive prototype. The algorithm calculates the throughput of the most recent N frames transmitted. The throughput is a measure of the channel state condition for the time interval required to transmit N frames. Based on this throughput, the algorithm decides which of the available coding rates to use from a user defined table. A user defined

threshold diagram which utilizes three coding rates is illustrated in Figure 4.1 It follows

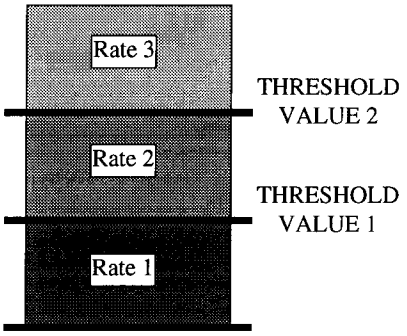


Figure 4.1 Threshold Regions Defining Coding Rates.

that the performance of the adaptive scheme is influenced by the selection of the value N and the threshold values. The smaller the value of N, the quicker the scheme adapts to the changing channel conditions. The threshold values are obtained from the throughput curves of the individual rates. In essence, one would superimpose the throughput curves and select threshold values to maximize the overall throughput of the scheme over all SNR values (i.e., select threshold values that will yield an overall maximum throughput equivalent to the maximum envelope of the individual throughputs).

The generalized adaptive coding rate algorithm is best described by the following procedure.

1. *Level 0:* Select the most powerful coding rate (Rate 1) and transmit using this rate for N frames. The algorithm moves up to the next level.
2. *Level 1:* Calculate the throughput of the last N frames transmitted. If the throughput is less than *THRESHOLD VALUE 1*, continue using Rate 1 to send the N frames and the algorithm remains at this level. Otherwise, if the throughput is greater than *THRESHOLD VALUE 1*, select Rate 2 to transmit the N frames and the algorithm moves up to the next level.

3. *Level i, i>1:* Calculate the throughput for the most recent N frames transmitted. If the throughput is less than *THRESHOLD VALUE i-1* select Rate *i-1*, transmit N frames, and move down to the next level. If the throughput is between *THRESHOLD VALUE i-1* and *THRESHOLD VALUE i*, continue using Rate *i*, transmit N frames, and remain at this level. If the throughput is greater than *THRESHOLD VALUE i*, select Rate *i+1*, transmit N frames, and move up to the next level.

In the prototype, code rate synchronization is obtained by using two bits in the rate 1/2 header to indicate the coding rate of the data packet following.

## Section 4.3 DSP Implementation of the Adaptive Scheme

The adaptive coding algorithm is contained in the SW ARQ protocol running on the host PC. The transmitter and receiver DSP boards require minor software modifications to be able to encode and decode any of the supported coding rates. The other necessary modification is to use 2 of the 9 bits, labelled as RESERVED in the header, to indicate which rate is currently being used to encode the data packet.

In the adaptive prototype scheme, N is chosen to be 5 and the coding rates used are 1/2, 3/4, and 1. The adaptive SW ARQ protocol can also be forced to transmit at one of the three code rates. Figure 4.2 depicts the experimental throughputs obtained for the three individual coding rates. Referencing Figure 4.2, *THRESHOLD VALUE 1* is selected to be 0.77 and *THRESHOLD VALUE 2* is 1.2. From the above algorithm, the rates of 1/2, 3/4, and 1 correspond to the code Rates of 1, 2, and 3 respectively. Notice by selecting *THRESHOLD VALUE 2* to be 1.2, there will be a region of the overall throughput which will be less than the maximum envelope of any of the three individual throughputs. Maximizing the throughput over all SNR levels is not always possible. The

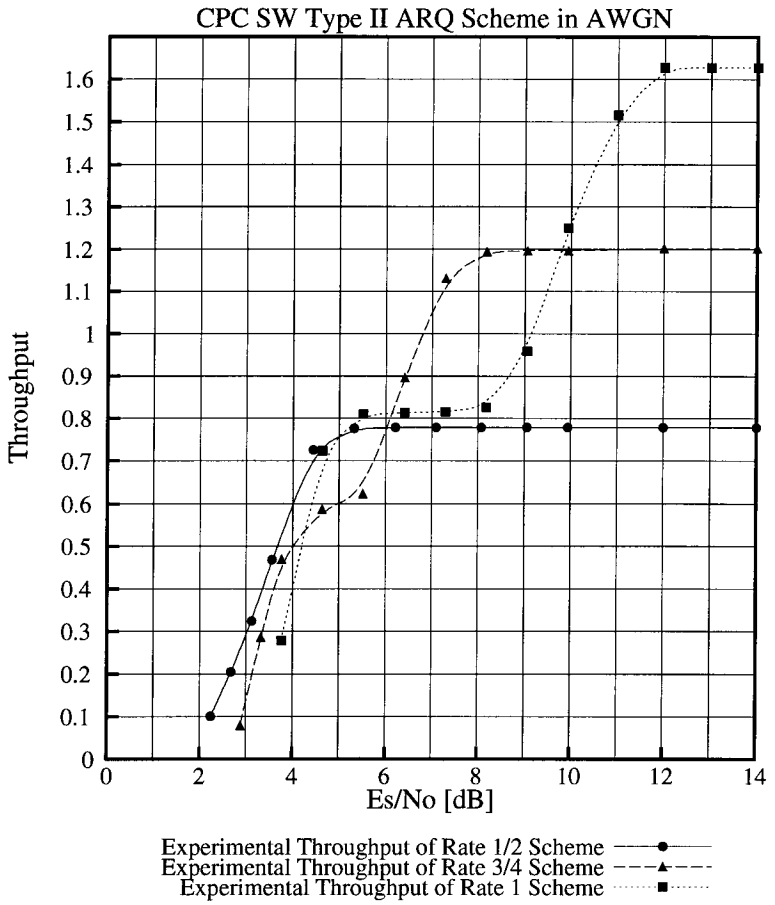


Figure 4.2 Experimental Throughputs of rate 1/2, 3/4, and 1.

specific SNR area is between 8dB and 10.5dB. If 1.2 is selected as a threshold value and the current rate is 3/4, once the throughput reaches 1.2 it switches to rate 1. This takes place at approximately 8dB where the throughput of a rate 3/4 system is 1.2 but the throughput of a rate 1 system is 0.85. As a result the adaptive scheme constantly switches between rate 1 and rate 3/4 within this region and maximum throughput is not obtained. The expected result is to obtain an average between the throughput curves of rate 3/4 and rate 1 in this region.

## Section 4.4 Performance Evaluation

Recall that the goal of the Adaptive CPC SW Type II ARQ protocol is to increase

or equal the throughput at all SNR levels as compared to the rate 3/4 CPC SW Type II ARQ protocol. The prototype is tested over several SNR levels by executing the scheme until 1000 frames are successfully delivered. Figure 4.3 displays the resulting Adaptive CPC SW Type II ARQ throughput in an AWGN channel. As expected, the throughput has increased at all SNR levels excluding the area between 7dB and 10dB. The slight degradation in this area was predicted and is a factor of the selection of *THRESHOLD VALUE 2*. It is observed that the throughput curve has a stair case shape. This is due to rate 1/2 being utilized at low SNR levels, rate 3/4 at medium SNR levels, and rate 1 at high SNR levels. The results in Figure 4.3 clearly validates the operation of the adaptive scheme.

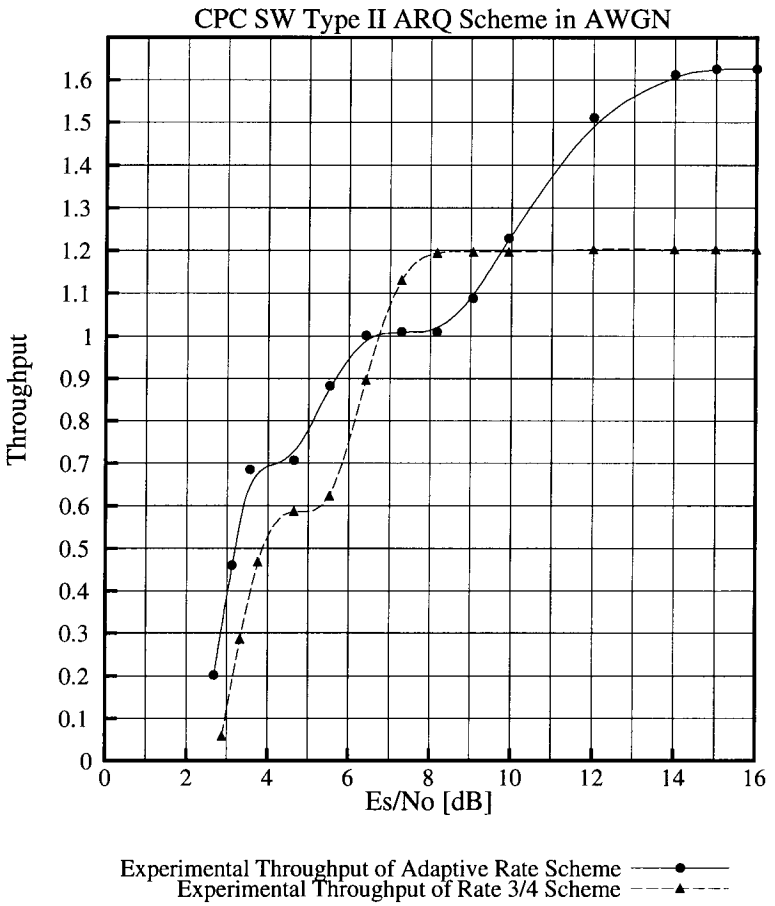


Figure 4.3 Adaptive CPC SW Type II ARQ Throughput.

Figure 4.4 displays the adaptive scheme's throughput for various values of N. N is a performance parameter which adjusts how quickly the scheme reacts to changes in the channel conditions. It is observed that changing the value of N between 5 and 15 (i.e., approximately 5000 to 15000 bits) has marginal effect on the performance of the scheme in an AWGN channel. This can be accounted to the fact that an AWGN channel's SNR level is constant for all practical purposes as compared to the instantaneous SNR level of the Rayleigh fading channel which fluctuates according to a rayleigh distribution. Changing the value of N for the combined AWGN and Rayleigh channel is expected to affect throughput performance.

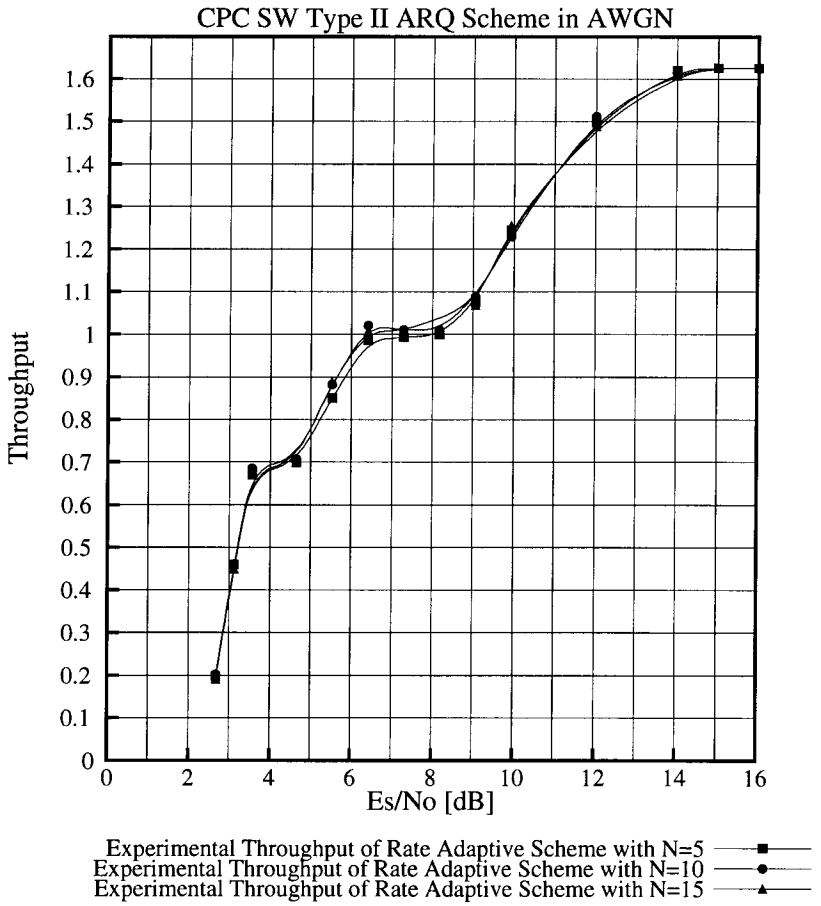


Figure 4.4 Affect of varying N for the Adaptive Scheme's Throughput.

Figure 4.5 illustrates the Adaptive CPC SW Type II ARQ scheme in a combined AWGN and Rayleigh fading channel for a $B_DT$ product of 0.00084. The value for N is 5 and the threshold values chosen are 0.76 and 1.19. The threshold values are slightly lower than those used in the AWGN channel as the fading channel is a very harsh environment and it is more difficult to reach and maintain the threshold values. For comparison purposes, the experimental throughput for the rate 3/4 CPC SW Type II ARQ is also plotted. As in the AWGN channel, the throughput is increased at lower SNR levels,

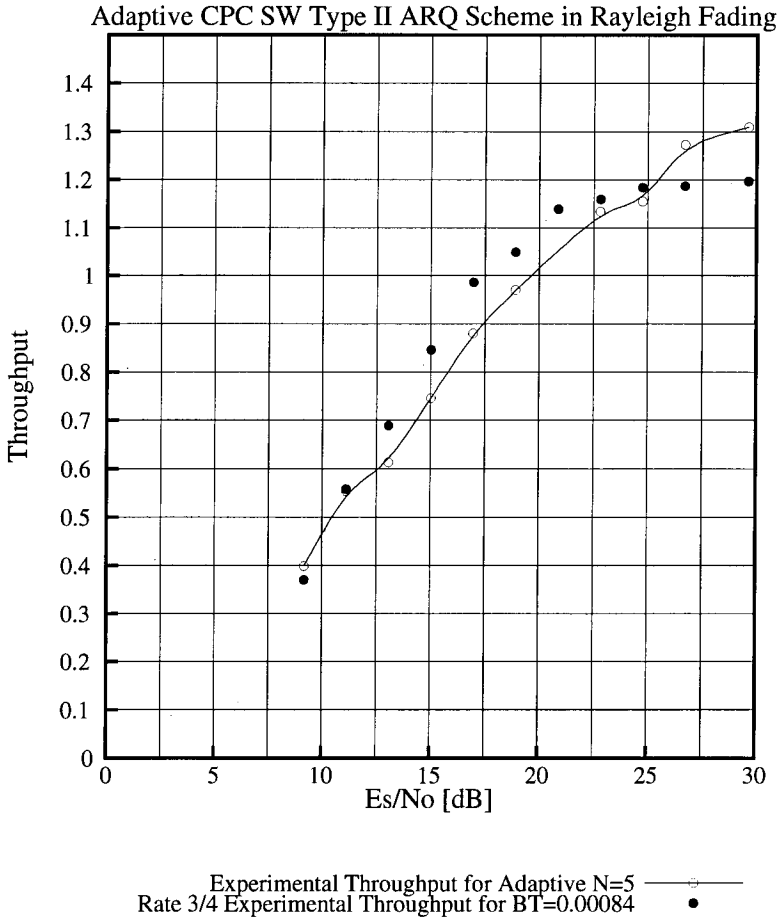


Figure 4.5 Adaptive CPC SW Type II ARQ in Rayleigh Channel.

degraded at medium SNR levels, and increased at high SNR levels. Again the stair case

shape is evident. As in the AWGN channel, it is a result of the rate 1/2 code being used at low SNR levels, rate 3/4 at medium SNR levels, and rate 1 at high SNR levels.

Figure 4.6 depicts the remaining $B_DT$ product curves for the adaptive scheme. It is observed that the slower the vehicle speed (i.e., the smaller the $B_DT$ product) the quicker the maximum throughput is reached at the higher SNR values. This is a very important observation which implies that the set of code rates used must be optimized to the set of $B_DT$ products representing the average vehicle speeds and transmission rate used. The three $B_DT$ products of 0.0043, 0.0022, and 0.00084 correspond to a $\pi/4$ shift DQPSK system operating with a carrier frequency of 900MHz, a baud rate of 19.2kHz, and vehicle velocities of 100, 50, and 20 km/hr respectively. It is observed that the code rates of 1/2, 3/4 and 1 results in a relatively good throughput for the 20km/hr case as compared to the non-adaptive scheme. The same cannot be said about the remaining two speeds of 100 and 50km/hr which will eventually reach the maximum throughput but at a higher SNR level. This implies that a different set of code rates is required to give better performance. The random phase modulation caused by the increase in vehicle speed cannot be overcome by the Rate 1 code (uncoded). It requires higher SNR values to successfully deliver the frame as opposed to the 20km/hr case. In other words, a more powerful code than Rate 1 but weaker than 3/4 is required.

Figure 4.7 shows the effect of varying the value of N which changes the amount of time it requires for the adaptive scheme to react to channel conditions. When larger values of N are chosen, which indicates the adaptive scheme will take longer before reacting to the channel conditions, the performance degrades. This is due to the time varying characteristic of the Rayleigh channel. By selecting a smaller value of N, the

scheme can quickly adapt and maximize its throughput as opposed to a larger value of N which makes the scheme more lethargic. In other words, the smaller the value of N, the more successfully the adaptive scheme can track the channel conditions.

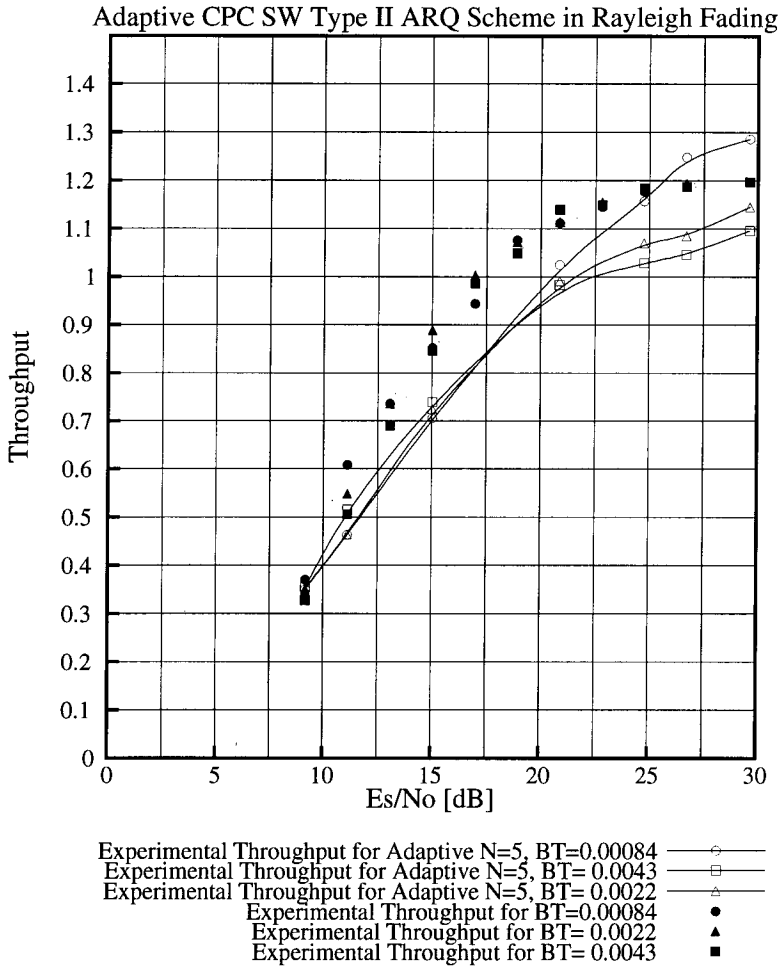


Figure 4.6 Adaptive CPC SW Type II ARQ in a Rayleigh Channel for Various $B_DT$ Products.

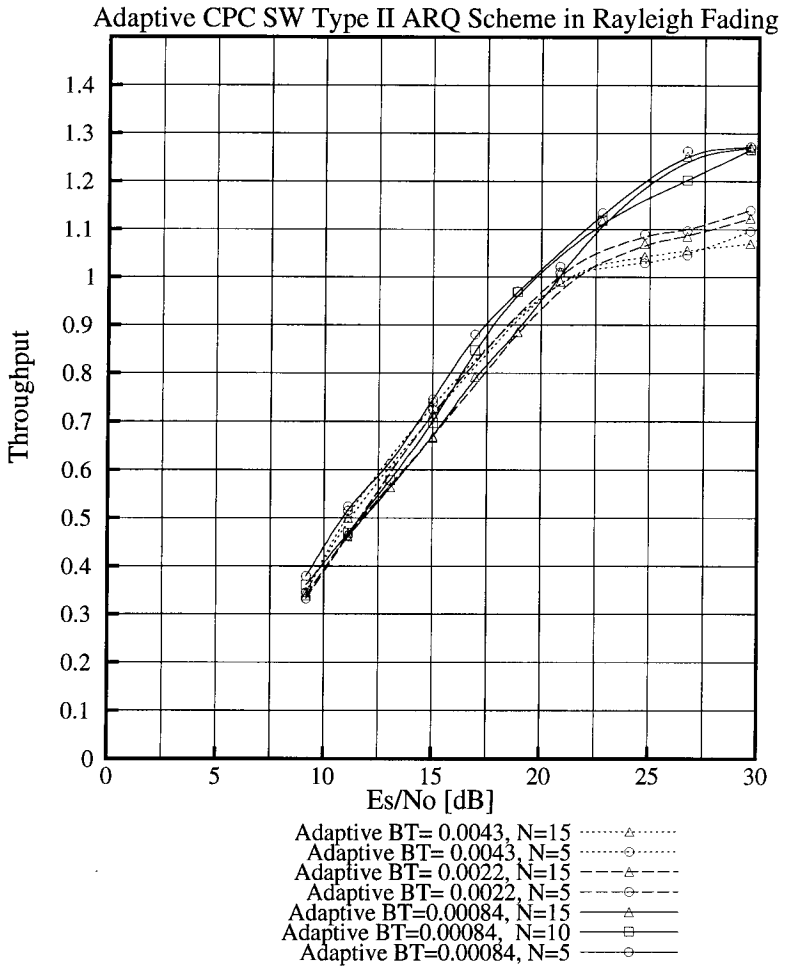


Figure 4.7 Effect of varying N for the Adaptive Scheme in a Fading channel.

## Section 4.5 SW ARQ Scheme Comparisons

The three CPC SW Type II ARQ schemes implemented are listed below and ranked according to throughput performance.

1. Adaptive CPC SW Type II ARQ Scheme
2. Rate 3/4 CPC SW Type II ARQ Scheme with Code Combining
3. Rate 3/4 CPC SW Type II ARQ Scheme

All three schemes are based on the CPC SW Type II ARQ protocol and utilize the identical general DSP software library. The performance of the rate 3/4 scheme was verified by the use of numerical results. The rate 3/4 code combining case resulted in an increase at low SNR levels. In code combining, repeated copies of the identical coded data sequences are optimally combined for subsequent decoding. This upgrade consisted of replacing 10 lines of DSP Assembly code. It does not require any additional memory because the receiver stores the combined data sequence and discards the most recent single copy (i.e. the most recent corrupted data sequence is combined with the previous copies of identical coded data sequences from a certain memory slot and then stored in that same memory slot). In order to obtain a greater increase in throughput over a larger region of SNR levels, the adaptive scheme was implemented. In comparison to the rate 3/4 scheme, the adaptive scheme's throughput increased for low and high SNR levels and decreased for medium SNR levels. As discussed above, the slight degradation ( less than 1dB) in the medium range is the result of the threshold value and the shapes of the individual rate throughput curves. The compromise of a slight degradation is well worth the gain in performance at lower and higher SNR values. It was also observed that the adaptive scheme's performance varied as a result of the system's $B_DT$ product which implies using a set of codes that are optimized for a set of $B_DT$ products. The actual adaptive upgrade consisted of adding case statements in DSP Assembly code to account for the various code rates. The threshold values and selection of the coding rate was added to the Host PC protocol program. The only other modification was to utilize 2 of the 9 Reserved bits of the header to indicate the rate of the data packet.

## Section 4.6 Conclusions

An Adaptive rate CPC SW Type II ARQ scheme was implemented using the existing prototype of Chapter 3 with software modifications. The goal was to utilize the existing general software modules in order to minimize any cost associated with the upgrade. The Adaptive coding rate algorithm was presented and explained. The throughput of the adaptive prototype was experimentally measured for both an AWGN channel and combined AWGN and Rayleigh fading channel. In both channels, the experimental throughput showed a general increase in performance. More specifically, the adaptive scheme's throughput increased for low and high SNR levels and decreased for medium SNR levels in comparison to the rate 3/4 scheme. The compromise of a slight degradation is well worth the gain in performance at lower and higher SNR values. The effect of varying N, which controls the reaction time of the adaptive scheme, was also investigated. It was found that the value of N had marginal affect on the throughput in an AWGN channel. In a combined AWGN and Rayleigh fading channel, as N is decreased the throughput performance increases. The Rayleigh channel is time varying and the smaller the value of N, the more successfully the adaptive coding rate can track the channel conditions.

The experimental results indicate that the upgrade of the CPC SW Type II ARQ protocol to an adaptive scheme was successful.

# Chapter 5 Conclusions and Future Research

## Section 5.1 Conclusions

This thesis investigated the design, implementation issues, and performance evaluation of various adaptive and non-adaptive FEC coding schemes of a Type II SW ARQ system. The research contributions can be summarized as follows:

1. The Software design, implementation, and test of a Digital Signal Processing (DSP) Module Library for the Spectrum TMS32C30 DSP card housed in an IBM PC platform. The library consists of the following modules:

   - **CRC Encoder/Decoder**
   - **Rate 1/2 Convolutional Encoder**
   - **Puncturing Module**
   - **Rate 1/2 Soft Decision Viterbi Decoder**
   - **Block Interleaver**
   - **Soft Data Deinterleaver**
   - **Queueing Module**
   - **$\pi$/4 shift DQPSK Baseband Transmitter/Receiver**

2. The Software implementation and evaluation of a Complementary Punctured Convolutional (CPC) coding scheme for the SW Type II ARQ system with and without code combining utilizing the DSP library in an AWGN channel and a combined AWGN and Rayleigh Fading channel.

3. Software upgrade and performance evaluation of an Adaptive CPC SW Type II ARQ scheme utilizing the DSP library in an AWGN channel and a combined AWGN and Rayleigh Fading channel.

In this thesis a general algorithm for Complementary Punctured Convolutional Coding applied to a Stop-and-Wait ARQ scheme was presented. A rate 3/4 CPC SW Type II ARQ protocol was implemented with the use of two Spectrum TMS320C30 DSP cards and a host IBM PC. The following assumptions or simplifications are incorporated in the implemented prototype which consists of the DSP transmitter and receiver cards in the same Host PC under the control of the SW ARQ protocol.

- As a consequence of the transmitter and receiver DSP cards being in the same Host PC, they are initialized and synchronized by the ARQ Protocol running on the Host PC. In practice, there is an initialization and synchronization process to be executed by the independent transmitter and receiver.
- In practice a noisy return channel is used to send the receiver's reply. In the prototype, the receiver's reply is passed internally through the PC via the DAM. This is a noise free return channel.
- As a result of the ARQ protocol controlling both the transmitter and receiver, it is the receiver which times out if a flag is not found. Again, in practice it is the transmitter that times out if it does not get a response from the receiver.
- Symbol Synchronization is accomplished by hard wiring the transmitter and receiver. The actual symbol timing signal is software generated and is not ideal. A practical system would have the receiver utilize a Phase Locked Loop or some other synchronization circuit to obtain symbol synchronization with no link to the transmitter.

These simplifications do not compromise the accuracy of the experimental results. The prototype is used to evaluate various FEC strategies which are unaffected by the above simplifications.

The rate 3/4 CPC SW Type II ARQ scheme was numerically analyzed for both an AWGN channel and a combined AWGN and Rayleigh fading channel. The experimental data obtained from the prototype was in good agreement with the numerical results validating the implementation and correct operation of the scheme.

The rate 3/4 CPC SW Type II ARQ scheme was upgraded with Code Combining in an effort to gain an increase in the throughput performance. This allows the receiver to optimally combine copies of the same coded sequence for subsequent decoding. The experimental throughput performance increased at low SNR levels as compared to the non-code combining case verifying its proper operation. The upgrade consisted of replacing 10 lines of DSP Assembly Language. The memory requirement remains constant since one data sequence, which consists of the combined copies, is kept rather than the individual copies.

In an effort to further increase the throughput performance of the prototype, the CPC SW Type II ARQ protocol was upgraded with an Adaptive Coding Rate. The resulting experimental throughput showed an increase at low and high SNR levels and a slight degradation at medium SNR levels with respect to the throughput of the original rate 3/4 prototype. The compromise of a slight degradation is well worth the gain in performance at lower and higher SNR values. This degradation is due to the selection of threshold values used in the adaptive coding rate algorithm.

The three implemented schemes behaved as expected and their experimental through-

puts verified their correct operation.

## Section 5.2 Future Research

### 5.2.1 Symbol Synchronization

The $\pi/4$ shift DQPSK modulation system used by the prototypes suffers from imperfect symbol synchronization. As a result, the throughputs of the prototypes are degraded at lower SNR levels. It would be interesting to further investigate the symbol synchronization of the system.

### 5.2.2 Selective Repeat Upgrade

Although a Stop-and-Wait ARQ protocol was used for the prototypes, the software modules and the design of the system were such that an upgrade to a Selective Repeat (SR) Protocol is possible. It would be interesting to have the prototypes upgraded to SR as this would only require software modifications but the majority of DSP library modules do not have to be modified.

### 5.2.3 Adaptive Header

The implemented adaptive scheme varied the coding rate of the data packet while the coding rate of the header remained constant (rate 1/2). If the coding rate of the header is also made adaptive the throughput will increase. At high SNR levels, a powerful code is not required and a larger data packet can be sent resulting in greater throughput. At lower SNR levels, a more powerful coded header will deliver the data packet and reduce the number of retransmissions for header failures. The coding rate for the header should always be more powerful than the coding rate of the data packet. In order to indicate the rate of the header, a miniature header should proceed the header.

### 5.2.4 FEC Schemes

With the existing testbed used for the prototypes and the modular structure of the DSP library software, this leads to endless possible FEC schemes that may be investigated and explored.

## Bibliography


[1] S. Lin and J. D. J. Costello, *Error Control Coding: Fundamentals and Applications.* Prentice Hall, 1983.

[2] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Trans. Commun.*, vol. 36, pp. 389–400, Apr. 1988.

[3] S. Kallel, "Analysis of a type II hybrid ARQ scheme with code combining," *IEEE Trans. Commun.*, vol. 38, pp. 1133–1137, Aug. 1990.

[4] K. J. Guth and T. T. Ha, "An adaptive stop-and-wait ARQ strategy for mobile data communications," in *the Proceedings of IEEE the 40th Vehicular Technology Conference*, pp. 656–661, Apr. 1990.

[5] D. P. C. Wong and P. T. Mathiopoulos, "Nonredundant error correction analysis and evaluation of differentially detected $\pi/4$-shift DQPSK systems in a combined CCI and AWGN Environment," *IEEE Trans. Veh. Tech.*, vol. 41, pp. 35–48, Feb. 1992.

[6] C. L. Liu and K. Feher, "Noncoherent detection of $\pi/4$-QPSK systems in a CCI-AWGN combined environment," in *the Proceedings of the 39th Vehicular Technology Conference*, pp. 83–94, May 1989.

[7] D. P. Bouras, "Optimal decoding of PSK and QAM signals in frequency nonselective fading channels," Master's thesis, University of British Columbia, 1991.

[8] E. Casas and C. S. K. Leung, "A simple digital fading simulator for mobile radio," *IEEE Trans. Veh. Tech.*, vol. 39, pp. 205–212, Aug. 1990.

[9] J. G. Proakis, *Digital Communications.* New York:McGraw-Hill Book Company, 2 ed., 1989.

[10] C. L. Liu and K. Feher, "Performance of Non-coherent $\pi/4$-QPSK in a frequency-selective fast Rayleigh fading channel," in *the Proceedings of SUPERCOM/ICC 90, Atlanta GA*, pp. 335.7.1–335.7.5, Apr. 1990.

[11] S. Kallel, "Complementary Punctured Convolutional (CPC) Codes and their use in hybrid ARQ schemes," in *the Proceedings of IEEE Pacific Rim Conference*, pp. 186–189, May 1993.

[12] S. Kallel and D. Haccoun, "Generalized type II hybrid ARQ scheme using punctured convolutional coding," *IEEE Trans. Commun.*, vol. 38, pp. 1938–1946, Nov. 1990.

[13] G. Begin and D. Haccoun, "High rate punctured convolutional codes: structure properties and construction techniques," *IEEE Trans. Commun.*, vol. 37, pp. 1381–1385, Dec. 1989.

[14] J. A. Heller and I. M. Jacobs, "Viterbi decoding for Sattelite and space communication," *IEEE Trans. Commun.*, vol. 19, pp. 835–848, Oct. 1971.

[15] A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," *IEEE Trans. Commun.*, vol. 19, pp. 751–772, Oct. 1971.

[16] J. G. Proakis, "Probabilities of Error for Adaptive Reception of M-Phase Signals," *IEEE Trans. Commun.*, vol. 16, pp. 71–80, Feb. 1968.

[17] P. F. Driessen, "Performance of frame synchronization in packet transmission using bit erasure information," *IEEE Trans. Commun.*, vol. 39, pp. 567–573, Apr. 1991.

[18] T. Matsumoto and F. Adachi, "BER analysis of convolutional coded DQPSK in digital mobile radio," *IEEE Trans. Veh. Tech.*, vol. 40, pp. 435–442, May 1991.

[19] D. Chase, "Code Combining- a maximum-likelihood decoding approach for combining an arbitrary number of noisy packets," *IEEE Trans. Commun.*, vol. 33, pp. 385–393, May 1985.

[20] J. Hagenauer, "Forward Error Correction coding for fading Compensation in Mobile Sattelite Channels," *IEEE Journal Select. Areas Commun.*, vol. 5, pp. 215–225, Feb. 1987.

[21] N. R. Sollenberger, J. C. I. Chuang et al., "Architecture and implementation of an efficient and Robust TDMA frame structure for digital portable communications," *IEEE Veh. Trans.*, vol. 40, pp. 250–260, Feb. 1991.

[22] J. B. Cain, G. C. Clark Jr., and J. M. Geist, "Punctured Convolutional codes of Rate (n-1)/n and simplified maximum likelihood decoding," *IEEE Trans. Inf. Theory*, vol. 25, pp. 97–100, Jan. 1979.

[23] J. C. I. Chuang, "Comparison of two ARQ protocols in a Rayleigh fading channel," *IEEE Veh. Trans.*, vol. 39, pp. 367–373, Nov. 1990.

[24] C. S. K. Leung and A. Lam, "Forward error correction for an ARQ scheme," *IEEE Trans. Commun.*, vol. 29, pp. 1514–1519, Nov. 1981.

[25] R. W. Lucky, J. Salz, and E. J. Weldon, Jr., *Principles of Data Communication.* McGraw-Hill Book Company, 1968.

[26] W. C. Lindsey and M. K. Simon, *Telecommunication Systems Engineering*. Prentice-Hall Inc., 1973.

[27] P. Bylanski and D. G. W. Ingram, *Digital Transmission Systems*. Peter Peregrinus Ltd., 1976.

[28] K. Feher, *Digital Communications: Satellite/Earth Station Engineering*. Prentice Hall Inc., 1983.

[29] K. Feher, *Digital Communications: Microwave Applications*. Prentice Hall, 1981.

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

[31] S. Haykin, *An Introduction to Analog and Digital Communications*. John Wiley & Sons, 1989.

[32] K. Feher and Engineers of Hewlett Packard Ltd., *Telecommunication Measurements, Analysis, and Instrumentation*. Prentice Hall, 1987.

[33] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Prentice Hall, 1989.

# Appendix A Software Listings

The software listings appear in the following order:

- CSUB.C — DSP Module Library.
- ADAPT.C — Adaptive SW Type II ARQ Protocol for IBM Host PC.
- XMITADAP.ASM — DSP Assembly code for Transmitter DSP card.
- RCVRADAP.ASM — DSP Assembly code for Receiver DSP card.
- VARSRCVR.ASM — Variables, definitions, and memory locations used by the assembly code for the transmitter and receiver DSP Cards.

```
1     #include <stdlib.h>
2     #include <stdio.h>
3     #include <math.h>
4     main()
5     {
6     }
7
8     /*********************************************************************
      ******
9     *
10    *       polydiv         v1.02           Feb 93
11    *
12    *       This module implements a linear feedback shift register for
13    *       polynomial division.  It is used for CRC calculation and
14    *       the decoding of convolved data chunks.
15    *       This module requires 5 parameters:
16    *        K              - constraint length valid up to 33 decimal
17    *        MESGDATA       - pointer to message or data (dividend)
18    *        POLY           - polynomial to be divisor
19    *        TOTAL          - length of dividend in 32 bit words
20    *        RESULT         - pointer where quotient is to be stored
21    *
22    *       The resulting LFSR will be:
23    *            ___________________________________________
24    *           |        |        |                          |
25    *         --|-->x0--|-->x1--|-->x2---   ...  -->x(K-1)--|--->
26    *
27    *       where the connections are determined by POLY.
28    *
29    *       NOTE: POLY is read from right to left
30    *                     ie 1+x+x^4  ---->  10011 = POLY
31    *
32    *               RESULT
33    *              ------------
34    *     POLY   |  MESGDATA            REMAINDER
35    *            |
36    *
37    *
38    *       The data is feed into the LFSR msb first and lsb last
39    *       ie
40    *       address         data
41    *       300             25a4e845
42    *       301             84fe68dc
43    *
44    *       "548e4a52cd86ef48" ------->> LFSR
45    *
46    *
47    *       Decoder Notes:
48    *           1>  Give all 5 parameters and use quotient stored at
49    *               *RESULT.  Remainder can be discarded.
50    *
51    *       CRC Notes:
52    *           1>  If calculating the CRC it is your responsibility to
      make
53    *               sure the data is premultiplied (padded with zeros) b
      y X^K
54    *           2>  The REMAINDER is the CRC value
55    *
56    *       index1                  -general variable used as loop count
      er &
```



```
57    *                                current 32 bit word
58    *       index2                  -current bit position in 32 bit word
       (0-31)
59    *       infobit                 -input bit to LFSR
60    *       outputbit               -output bit from LFSR
61    *       shift                   -amount to shift LFSR to get output
      bit
62    *       shreg                   -contents of LFSR
63    *       feedback                -feeback value for LFSR
64    *       connections             -bits representing connections of LF
      SR
65    *       temp                    -temp storage of outputs of LFSR
66    *       remainder               -remainder of polynomial division
67    *********************************************************************
      *******/
68
69
70    long unsigned polydiv(int K, long int *MESGDATA, long unsigned int P
      OLY,
71                          int TOTAL, long int *RESULT)
72    {
73
74            int index1, index2, infobit, outputbit, shift;
75            long unsigned int shreg, feedback, connections, temp, remain
      der;
76
77
78            shift=0;
79            temp=0;
80            shreg = 0;
81            connections =0;
82            /*The standard used is to read polynomials from right to lef
      t.
83              This module needs them in left to right format so we rever
      se
84              the bits of POLY  ie 0010 --> 0100.
85              The following code reverse POLY and stores it in connectio
      ns*/
86
87            for (index1 = K-1; index1 > 0; index1--)
88            {
89                    if ( ((POLY) & (1 << index1 -1)) != 0)
90                            connections |= 1 << ( K-1 - index1 );
91
92
93             }    /* end of reversal routine */
94
95
96            /*This section of code determines the amount to shift the
97              register by.  This is needed because if the constraint
98              length is, for example, 7 but we only use the first two
99              registers (ie POLY = 0000011) then we must shift the
100             register by 5 positions to get the output of the 2nd regis
      ter */
101
102           for (index1 = K; index1 >= 0; index1--)
103           {
104                   if ( (POLY & (1<< (index1-1))) !=0)
105                   {
106                           shift=(K-index1);
```

```
107                         index1=-1;
108                 }
109         }/* end of finding shift amount */
110
111         if (K==33)                      /*exceptional case with 32bi
    t CRC*/
112                 shift=0;                /*and we don't require a shi
    ft   */
113
114         outputbit = 0;
115
116         /*This next section implements the LFSR and feeds the entire
117           encoded message through it.  The msb of the polynomial/mes
    sage
118           is feed into the LFSR first.
119           ie encoded message is "F78jNd"    dNj87F---->LFSR
        */
120
121         for (index1 = TOTAL -1; index1 >=0; index1--)
122         {
123                 for (index2 = 31; index2 >= 0; index2--)
124                 {
125                 /*lets get bit to input into LFSR */
126                         infobit=0;
127                         if ( (*(MESGDATA + index1 ) & (1 << index2))
      != 0 )
128                                 infobit=1;
129
130                         /* if output=1 then feed it back */
131                         if (outputbit == 1)
132                                 feedback = 0xffffffff & connections;
133                         else
134                                 feedback = 0;
135
136
137                         shreg = feedback ^ ((infobit << K - 2) | shr
    eg>>1);
138                         outputbit=( shreg>> (shift) ) & 1;
139
140                         /* this shift register SHREG is implemented
    with the */
141                         /* lsb being the msb of the shift register
             */
142                         /*      msb                 lsb
             */
143                         /* ie   . . . . r4 r3 r2 r1 r0
             */
144                         /*              x0 x1 x2 x3 x4 <--- powers o
    f poly   */
145
146
147                         if (index2 == 0)
148                         {
149                                 remainder = shreg;
150
151                                 *( RESULT + index1 ) = temp;
152                                 temp = outputbit << 31;
153                         }
154                         else
155                                 (temp) |= outputbit << (index2 - 1);
```



```
156                 }
157         }
158
159         /*END = 11;*/
160         return(remainder);
161 }
162
163 #define         K       ((Int *)0x809c00)
164 #define         POLY1   ((Int *)0x809c01)
165 #define         POLY2   ((Int *)0x809c02)
166 /*****************************************************************
    *****
167 *       CONV            V1.02           Feb 93
168 *
169 *       This module implements a convolutional encoder of rate
170 *       1/2 with variable length K and generator polynomials
171 *       POLY1 and POLY2.
172 *       This module requires 5 parameters:
173 *       *MESG           - pointer to data to be convolved
174 *       *MESGP1         - pointer to convolved data as a result of P
    OLY1
175 *       *MESGP2         - pointer to convolved data as a result of P
    OLY2
176 *       SIZE            - # of 32 bit words to convolve
177 *
178 *
179 *                       |---------------------------------> P1 bits
180 *                     POLY 1
181 *       ----------------------------------------
182 *       |  1    |   2   | .... |       |   K   |
183 *       ----------------------------------------
184 *                     POLY 2
185 *                       |---------------------------------> P2 bits
186 *
187 *       The encoder is implemented by polynomial multiplication of t
    he
188 *       data and POLY1 and POLY2 polynomials respectively. This is
189 *       achieved by shifting and exclusive ORing.
190 *
191 *       Notes:
192 *       1> When using conv be sure to have a storage area with K ext
    ra
193 *          bits so that none of the convovled data bits are lost.
194 *               ie.
195 *                       16 bits of data
196 *                       K =4                --------> requires 20 bi
    ts
197 *                                                     of storage spa
    ce
198 *
199 *
200 *       vmesg[]         - vector containing data to be convolved
201 *       vmesgP1[]       - vector containing convolved data from POLY
    1
202 *       vmesgP2[]       - vector containing convovled data from POLY
    2
203 *       temp[]          - intermediate storage vector
204 *       index1          - general loop index
205 *       index2          - general loop index
206 *       index3          - general loop index
```

```
207   *       firstbit        - firstbit of current 32 bit word
208   *       lastbit         - 32nd bit of current 32 bit word
209   *       mask            - used to select which term of POLYx is used
       in
210   *                         multiplication
211   *       byte            -intermediate storage variable
212   *
213   *       RETURNS         - nothing
214   ************************************************************************
      /
215
216   void conv( long unsigned int *MESG, long unsigned int *MESGP1,
217                   long unsigned int *MESGP2, int SIZE)
218
219   {
220
221           long unsigned int  temp[33], vmesg[33], vmesgP1[33], vmesgP2
      [33];
222           int index1, index2, index3, firstbit, lastbit, mask;
223           long byte;
224
225           /* get message into vector for processing */
226           for (index1 = 0; index1 < SIZE; index1++)
227                   {
228                           vmesg[index1] = *(MESG + index1);
229                           temp[index1]=0;
230                           vmesgP1[index1]=0;
231                           vmesgP2[index1]=0;
232                   }
233           /* convolution using polynomial multiplication of POLY1 and
      */
234           /* POLY2.  Implemented by shifting and XOR vextors.
      */
235
236           mask = 1l;
237           for(index1 = 0; index1 < *K; index1++)
238           {
239                   firstbit = 0;
240                   lastbit = 0;
241
242                   /* shifting routine which shifts entire contents of
       */
243                   /* vector.  Note that shifting does occur across
       */
244                   /* element boundaries.
       */
245                   for(index2 = 0; index2 < SIZE; index2++)
246                   {
247                           if (index1 == 0)
248                           {
249
250                                   temp[index2]=vmesg[index2];
251                           }
252                           else
253                           {
254                                   byte = temp[index2];
255                                   lastbit=byte & 0x80000000;
256                                   temp[index2]= (byte << 1)| (firstbit
      );
257                                   if (lastbit != 0)
```



```
258                                           firstbit = 1l;
259                                   else
260                                           firstbit = 0l;
261                           }
262                   }/*end of shifting (polynomial multiplication)*/
263
264                   /* add the terms of the polynomial multiplication to
        */
265                   /* the appropriate vector according to their generat
      or*/
266                   /* polnomials POLY1 and POLY2.
        */
267
268                   if ((*POLY1 & mask) != 0)
269                           {
270                                   for(index3=0; index3<SIZE; index3++)
271                                   vmesgP1[index3] ^=temp[index3];
272                           }
273
274                   if ((*POLY2 & mask) != 0)
275                           {
276                                   for(index3=0; index3<SIZE; index3++)
277                                   vmesgP2[index3] ^=temp[index3];
278                           }
279
280                   mask = mask << 1;
281           }/* end of adding up terms */
282
283           /* now place the convolved messages P1 and P2 in the Dual */
284           /* memory so that the PC host can retrieve it.           */
285           for(index1 = 0; index1 < SIZE; index1++)
286           {
287                   *(MESG   + index1) = vmesg[index1];
288                   *(MESGP1 + index1) = vmesgP1[index1];
289                   *(MESGP2 + index1) = vmesgP2[index1];
290           }
291
292       return;
293   }
294   /************************************************************************
      **/
295   /*
296           The parameters chosen are used in interleaving by 128 bit
297           blocks which represent 128/2 = 64 SYMBOLS
       */
298
299   #define ROW                     16    /* BLOCK INTERLEAVING PARAMETE
      RS*/
300   #define COLUMN                  16
301   #define BITS_PER_SYMBOL         2
302   #define SYMBOLS_PER_LINE        16    /*LINE = 32 BIT WORD*/
303   #define FLAG1   0x00000003            /* FLAG1 MUST CORRESPOND TO
304                                            BITS PER SYMBOL
      */
305
306   /************************************************************************
307
308   interleaver     V1.00           Jan 93
309
310           This module will take the data given by pointer DEINT_ADDR
```

```
311          and interleave it according to the parameters above and
312          then place it starting at pointer INT_ADDR.
313
314          This module is a block interleaver.
315          eg
316
317          Given:
318          8 symbols/line
319          4 bits/symbol
320                                       [ 4 X 4]     Result:
321                  12345678            159D                 159D26AE
322                  9ABCDEF0            26AE                 37BF48C0
323                                      37BF
324                                      48C0
325
326
327          The module uses the current row and column of the array to c
     alculate
328          the symbol to be placed in the resulting interleaved data wo
     rd.
329          Once the symbol is known, the module calculates how much to
     increment
330          the data pointer DEINT_ADDR, and how much to shift the FLAG1
     .
331          Next, the module gets the symbol and stores it temporarily i
     n TEMP.
332          When enough symbols have been obtained to write a 32 bit wor
     d,
333          the module places the interleaved word (TEMP) at INT_ADDR an
     d
334          increments the pointer.
335
336          Note: Regardless of the dimensions of the array the module w
     ill
337          always place the result in 32 bit lengths.  As shown above w
     ith
338          the 4X4 array giving rise to 2 32 bit lines of interleaved d
     ata.
339
340
341          INT_ADDR        -pointer to interleaved data
342          DEINT_ADDR      -pointer to deinterleaved data
343
344          cur_row         -current row
345          cur_column      -current column
346          addr            -amount to increment DEINT_ADDR
347          pickbit         -bit amount to shift FLAG1 so that correct
348                           symbol is obtained
349          shift           -bit amount to shift symbol before placing i
     t
350                           in TEMP
351          mod             -intermediate calculation used for pickbit
352          symbol          -current symbol
353          symbol_counter  -used to count symbols and write in 32 bit
354                           lengths
355
356     ************************************************************************
     **********/
357
358     void interleaver(long int *DEINT_ADDR, long int *INT_ADDR)
```



```
359     {
360
361             int cur_row, cur_column, pickbit, addr, mod, shift=0;
362             int symbol=0;
363             int symbol_counter=0;
364             long unsigned int TEMP = 0;
365             for(cur_row=1; cur_row<=ROW; cur_row++)
366             {
367
368                     for(cur_column=0; cur_column<COLUMN; cur_column++)
369                     {
370                             symbol = cur_row + cur_column * ROW;
371                             addr = (symbol - 1) / SYMBOLS_PER_LINE;
372                             mod = (symbol % SYMBOLS_PER_LINE);
373                             if (mod ==0)
374                                     pickbit = (SYMBOLS_PER_LINE - 1) * B
     ITS_PER_SYMBOL;
375                             else
376                                     pickbit = ( mod - 1) *BITS_PER_SYMBO
     L;
377
378                             symbol_counter++;
379
380                             shift = pickbit - (symbol_counter - 1) * BIT
     S_PER_SYMBOL;
381                             shift = shift * -1;
382                             /*The above line ensures the compiler compil
     es the shift
383                             as a LSH rather than an ASH  DO NOT REMOVE*/
384
385                             TEMP = TEMP | (*(DEINT_ADDR + addr) & (FLAG1
      << pickbit))<<shift;
386
387                     /*if enough symbols for 32 bit word then write */
388                     if (symbol_counter == SYMBOLS_PER_LINE)
389                             {
390                             *(INT_ADDR++ ) = TEMP;
391                             symbol_counter=0;
392                             TEMP = 0;
393                             }
394                     }
395             }
396     }
397     /************************************************************************
     ***/
398
399     int     RATES[8][6] ={
400                             {1, 1, 0, 1, 1, 0},          /*import value "0"
     */
401                             {0, 1, 1, 0, 1, 1},          /*import value "1"
     */
402                             {1, 0, 1, 1, 0, 1},          /*import value "2"
     */
403                             {1, 1, 0, 1, 1, 0},          /*import value "3"
     */
404                             {0, 1, 0, 1, 0, 1},          /*import value "4"
     */
405                             {1, 0, 1, 0, 1, 0},         /*import value  "5"
     */
406                             {1, 0, 1, 0, 1, 0},         /*import 6*/
```

```
                        {0, 1, 0, 1, 0, 1}};          /*import 7*/

void puncture(int CHOSENRATE, long int *NEW, long int *OLD, int TOTAL)
{
        int array[6];
        unsigned long int mask=1;

        int index, value, bits, newbits=0;


        for (index=0; index<6; index++)
                array[index]=RATES[CHOSENRATE][index];

        for(bits=1; bits<=TOTAL; bits++)
        {
                value= array[ bits % 6];
                if (value==1)                   /*keep bit*/
                {
                        if ( (*OLD & mask) > 0 )
                                *NEW |= ( 1 << newbits);

                        newbits++;
                }
        /*        else if (value>1)
                {
                        for(index=0; index<value; index++)
                        {
                                if ( (*OLD & mask) > 0 )
                                        *NEW |= ( 1 << newbits);


                                newbits++;
                                if (newbits==32)
                                {
                                        newbits=0;

                                        NEW++;
                                }
                        }
                }  */

                mask<<=1;
                if(mask==0)
                {
                        OLD++;
                        mask=1;
                }
                if (newbits==32)
                {
                        newbits=0;

                        NEW++;
                }
        }
}

/*****************************************************************
***
*
```



```
*       combineheader   v1.02           Feb 93
*
*       This module is used to combine the outputs of the adders of the
*       convolutional encoder to form the header.  The header will either
*       be rate 1/2, 3/4 or 1/3.
*
*/

void combineheader(long int *P1, long int *P2, long int *HEADER, int NEWBITS)
{
        int bits, counter=0;
        unsigned long int templow=0;
        unsigned long int temphigh=0;
        unsigned long int mask=1;
        unsigned long int maskhigh=0x10000;
        /* NEWBITS must be halfed and rounded up since every word you give
           this module it automatically combines it into 2 words.  So if
           NEWBITS=64  the module changes it to 64/2=32 since it works on
           the high (16-31) and low (0-15) bits simultaneously giving rise
           to 2 bits for every one that NEWBITS counts.
        */
        NEWBITS = (NEWBITS/2)+ (NEWBITS%2);

        for (bits=1; bits<=NEWBITS; bits++)
        {
           /*     if ( ((*P1 & mask) | (*P2 & mask)<<1) > 0 )

                        templow |= 1 << counter;

                if ( ((*P1 & maskhigh) | (*P2 & maskhigh)<<1) >0 )
                        temphigh |= 1 << counter;           */


                templow |= ( (*P1&mask) | (*P2&mask)<<1 )<<counter;

                temphigh |=( (*P1&maskhigh)>>1 | (*P2&maskhigh) )>>(15-counter);

                counter +=1;
                mask<<=1;
                maskhigh<<=1;

                if (maskhigh==0)
                {
                        mask=1;
                        maskhigh=0x10000;
                        *HEADER++=templow;
                        *HEADER++=temphigh;
                        counter=0;
                        templow=0;
                        temphigh=0;
                        P1++;
```

```
516                             P2++;
517                     }
518             }
519     if (maskhigh !=0)                   /* Finished in loop but if we we
        re not  */
520     {                                   /* not given an amount of bits d
        ivisable*/
521             *HEADER++=templow;          /* by 32 then we must get the re
        maining */
522             *HEADER++=temphigh;         /* bits that are combined.
                */
523     }
524     }
```

```
1     #include                <stdlib.h>
2     #include                <stdio.h>
3     #include                <graphics.h>
4     #include                <conio.h>
5     #include                "c:\lib\tms30.h"
6
7     #define CONTROL_WORD    (0X30008)
8     #define VIRGIN_HEADER   (0x30010)
9     #define VIRGIN_DATA     (0x30013)
10
11    #define ACK0            (0x3007b)
12    #define STROBE_RCVR     (0x3007c)
13    #define STROBE_HOST     (0x3007d)
14    #define FLAG0P1         (0x300C0)
15    #define MENU_OPTION     (0x30006)
16    #define NOT_READY       01
17    #define READY           11
18    #define TROUBLE         (0x30130)
19    #define RUN             0fffff1
20
21
22    /*****************************************************************
      ****
23    *       ADAPT.C     SEPT 1993
24    *
25    *       Adaptive Complementary Punctured Convolutional Coding Scheme
       for a
26    *       Conventional Type II Stop and Wait ARQ System.
27    *
28    *       Using rate 1, 3/4, and 1/2 codes derived from a rate 1/2
29    *       mother code.
30    *
31    *       See DSP software for actual perforation matrix and generator
32    *       polynomials.
33    *
34    *
35    *  Frame Structure:
36    *
37    *       8 bit preamble + 24 bit flag                              32 b
      its
38    *
39    *       AA              2941B3
40    *
41    * CONVOLVED RATE 1/2 HEADER                                       128 b
      its
42    *
43    *       address |  Ns  |  Nr  | length |
44    *          14      4      4       10 ------------>32 bits
45    *
46    *       P1 or P2 | CSI | Reserved| CRC | Tail |
47    *          2        2      7        16     5 -------->32 bits
48    *
49    * CONVOLVED DATA                                                   864
      bits
50    *                                                                 -----
      ----
51    *                                                                1024
      bits
52    *
53    * UNCONVOLVED DATA - for use with rate 3/4 punctured from a rate 1/2
```



```
54    *
55    *       Data                | CRC | Tail |
56    *  3/4       635                32    5
57    *   1        859
58    *  1/2       411
59    *
60    *****************************************************************
      ****/
61
62    struct  window {
63                    int left;
64                    int right;
65                    int top;
66                    int bottom;
67                    };
68
69    void main(void)
70    {
71            FILE *fp;
72            int header[4], transmit[64], rcvr[64],frame[1000],output[20]
      ;
73            int menu, rate, packet, control, midx, midy, locx, locy;
74            float ber2, value;
75            int index1;
76            char beep = 7;
77            unsigned long int current, index2;
78            unsigned int temp,berword;
79            unsigned long int ber,symbols, oldber;
80            int ack, nack, trans, oldtrans, oldnack=0;
81            int CPC1, CPC2, CPC1CPC2;
82            int headfail, crcfail, lost, tot;
83
84            int     noldtrans, N;
85            float   noldinfobits, ninfo, rat, blockn=0;
86
87
88            int errorcode;
89            short errormsg;
90            long int STRT, temps,response;
91            struct window rcv, xmit, stat;
92
93            int     address, Nr, Ns, length, P1orP2, CSI;
94            int successframe, head1low, head1high, head2low, head2high;
95            float   infobits, totalbits, infoconstant, Current_factor=1.
      2;
96
97
98
99            /*graphics variables*/
100           int gdriver=DETECT, gmode,bkcol=DARKGRAY,maxx, maxy;
101           char msg[120];
102
103           for (index1=0; index1<1000; index1++)
104                   frame[index1]=0;
105
106
107
108           /*Initialize graphics screen*/
109           initgraph(&gdriver, &gmode, "");
110           errorcode = graphresult();
```

```
111           if (errorcode != grOk)
112           {
113                   printf("graphics error:%s\n", grapherrormsg(errorcod
      e));
114                   printf("Press any key to halt:");
115                   getch();
116                   exit(1);
117
118           }
119           cleardevice();
120           setbkcolor(bkcol);
121           maxx=getmaxx()-2;
122           maxy=getmaxy();
123           midx = maxx/2;
124           midy = maxy/2;
125           setlinestyle(SOLID_LINE, 1, THICK_WIDTH);
126           rectangle(0, 0, maxx, maxy*.15);
127           rectangle(0, maxy*.15, maxx*.5, maxy*.5);
128           rectangle(maxx*.5, maxy*.15, maxx, maxy*.5);
129           rectangle(0, maxy*.5, maxx, maxy);
130           setlinestyle(SOLID_LINE, 1, NORM_WIDTH);
131           sprintf(msg, "STATUS WINDOW");
132           outtextxy(10, maxy*.5+5, msg);
133
134           rcv.top = .24*maxy;
135           rcv.bottom = .48*maxy;
136           rcv.left = .83*maxx;
137           rcv.right = .99*maxx;
138           xmit.top = .24*maxy;
139           xmit.bottom = .48*maxy;
140           xmit.left = .33*maxx;
141           xmit.right = .48*maxx;
142           stat.top = maxy*.56;
143           stat.bottom = maxy*.6;
144           stat.left = 10;
145           stat.right = maxx*.98;
146
147           /* Set Text and Headings for the display screen      */
148           settextjustify(CENTER_TEXT, TOP_TEXT);
149           sprintf(msg, "Statistics Module for:");
150           outtextxy(midx, 15, msg);
151           sprintf(msg, "Adaptive Complementary Punctured Convolutional
      Stop and Wait Type II ARQ scheme");
152           outtextxy(midx, 25, msg);
153           sprintf(msg, "PI/4 DQPSK Modulation Scheme");
154           outtextxy(midx, 35, msg);
155           sprintf(msg, "No Code Combining");
156           setcolor(RED);
157           outtextxy(midx, 45, msg);
158           setcolor(WHITE);
159
160
161           sprintf(msg, "Transmitter DSP Board 0x390");
162           outtextxy(maxx/4, 85, msg);
163           settextjustify(LEFT_TEXT, TOP_TEXT);
164           sprintf(msg, "Total Frames sent:");
165           outtextxy(10, 115, msg);
166           sprintf(msg, "Baud Rate");
167           outtextxy(10, 125, msg);
168           sprintf(msg, "# of ACKs arrived:");
```



```
169           outtextxy(10, 135, msg);
170           sprintf(msg, "# of NACKs arrived:");
171           outtextxy(10, 145, msg);
172           sprintf(msg, "Current Frame # (of 1000):");
173           outtextxy(10, 155, msg);
174           sprintf(msg, "Current Rate:");
175           outtextxy(10, 165, msg);
176
177           settextjustify(CENTER_TEXT, TOP_TEXT);
178           sprintf(msg, "Receiver DSP Board 0x290");
179           outtextxy(maxx*.75, 85, msg);
180           settextjustify(LEFT_TEXT, TOP_TEXT);
181           sprintf(msg, "Error Free Rcvd Farmes:");
182           outtextxy(midx+10, 115, msg);
183           sprintf(msg, "# of lost Frames:");
184           outtextxy(midx+10, 125, msg);
185           sprintf(msg, "# of error Frames:");
186           outtextxy(midx+10, 135, msg);
187           sprintf(msg, "Throughput:");
188           outtextxy(midx+10, 145, msg);
189
190           sprintf(msg, "Total ACKs");
191           outtextxy(10, maxy*.5+20, msg);
192           sprintf(msg, "CPC Code 1");
193           outtextxy(210, maxy*.5+20, msg);
194           sprintf(msg, "CPC Code 2");
195           outtextxy(360, maxy*.5+20, msg);
196           sprintf(msg, "CPC 1 & CPC 2");
197           outtextxy(510, maxy*.5+20, msg);
198
199           sprintf(msg, "Total NACKs");
200           outtextxy(10, maxy*.5+50, msg);
201           sprintf(msg, "Header Failure");
202           outtextxy(210, maxy*.5+50, msg);
203           sprintf(msg, "Data CRC Failure");
204           outtextxy(360, maxy*.5+50, msg);
205           sprintf(msg, "Lost Frame");
206           outtextxy(510, maxy*.5+50, msg);
207
208           /*End of text set up */
209
210           errorcode=SelectBoard(0x290);
211           if ( errorcode == 0 )
212                   printerror(1);
213
214           errorcode=LoadObjectFile("RCVRADAP.OUT");
215           if (errorcode != 0)
216                   printerror(2);
217           Reset();
218           setcolor(RED);
219           settextjustify(CENTER_TEXT, TOP_TEXT);
220           sprintf(msg, "Loaded and running...");
221           outtextxy(maxx*.75, 95, msg);
222           for(index1=0; index1<3000; index1++);
223
224
225           errorcode=SelectBoard(0x390);
226           if ( errorcode == 0 )
227                   printerror(3);
228
```

```
229         errorcode=LoadObjectFile("xmitadap.out");
230         if ( errorcode != 0 )
231                 printerror(4);
232         Reset();
233         sprintf(msg, "Loaded and running...");
234         outtextxy(maxx/4, 95, msg);
235         settextjustify(LEFT_TEXT, TOP_TEXT);
236         setcolor(WHITE);
237
238
239         N=0;
240         rat=0.0;
241         noldtrans=0;
242         noldinfobits=0.0;
243         ninfo=0.0;
244         blockn=0.0;
245
246         oldtrans=0;
247         trans=0;
248         ack =0;
249         CPC1=0;
250         CPC2=0;
251         CPC1CPC2=0;
252         nack=0;
253         headfail=0;
254         crcfail=0;
255         infobits=0.0;
256         totalbits=0.0;
257         infoconstant=0.0;
258         tot=0;
259
260         successframe=0;
261         ber = 0;
262         oldber=0;
263         nack =0;
264         lost=0;
265         locy=0;
266         locx=10;
267         current = 0l;
268         response =0l;
269         Put32Bit(ACK0, DUAL, 69691);
270
271         /*************************************************************
    ****/
272         /*  Initialize Header variables*/
273         address = 0x1000;                          /* 14 bit address*/
274         Ns = 0;
275         Nr = 0;
276         length = 608;                             /* full frame */
277         PlorP2  = 1;                               /* which code to us
    e */
278         CSI = 3;                                  /*channel state info
    */
279
280
281
282
283         for (index1=0; successframe<1000; index1++)
284         {
285                 for (index2=0; index2<64; index2++)
```



```
286                         transmit[ index2 ] = 0;
287
288                 /*rate decision must be made here*/
289                 N=trans-noldtrans;
290                 ninfo=infobits-noldinfobits;
291                 if (N>=5)
292                 {
293                         blockn=(2.0*(ninfo/(1056.0*N)));
294                         if (CSI == 1)
295                         {
296                                 if (blockn<1.18)
297                                 {
298                                         CSI = 2;
299                                         goto out;
300                                 }
301                         }
302                         else if (CSI == 2)
303                         {
304                                 if (blockn>1.19)
305                                 {
306                                         CSI = 1;
307                                         goto out;
308                                 }
309                                 else if (blockn<.77)
310                                 {
311                                         CSI = 3;
312                                         goto out;
313                                 }
314                         }
315                         else if (CSI == 3)
316                         {
317                                 if (blockn>.76)
318                                 {
319                                         CSI = 2;
320                                         goto out;
321                                 }
322                         }
323                         ninfo=0.0;
324 out:                    noldinfobits = infobits;
325                         N=0;
326                         noldtrans = oldtrans;
327                         goto out2;
328                 }
329
330
331
332 out2:                    if (CSI == 1)
333                 {
334                         tot=52;
335                         infoconstant=859.0;
336                 }
337                 else if (CSI == 2)
338                 {
339                         tot = 38;
340                         infoconstant=635.0;
341                 }
342                 else if (CSI == 3)
343                 {
344                         tot = 24;
345                         infoconstant = 411.0;
```

```
346                 }
347
348
349
350                 for (index2=0; index2<tot; index2++)
351                 {
352                         transmit[ index2 ] = rand();
353                         if ( (index2 + 1) % 8 == 0)
354                                 transmit[index2] = transmit[index2]
    & 0x07ff;
355
356                         rcvr[ index2 ] = 0;
357                 /*generate 608 random  data bits to be transmitted *
    /
358                 }
359                 /*transmit[37]=0;*/
360
361                 /*************************************************
    ******/
362                 /* Build Header
        */
363                 address = address +1;
364                 PlorP2=1;
365
366                 do{
367                 trans++;
368                 if (response > 1000)              /* if retransmit a
    lternate*/
369                 {
370                         if (response != 9999)
371                         {
372                                 if (PlorP2 == 1 )
373                                         PlorP2 = 2;
374                                 else
375                                         PlorP2 = 1;
376                         }
377                 }
378
379                 for (index2=0; index2<4; index2++)
380                         header[ index2] = 0;
381
382
383                 header[0] = Ns <<14 | address;
384                 header[1] = (Ns >> 2) | (Nr << 2) | (length <<6);
385                 header[2] = (PlorP2) | (CSI <<2);
386
387                 WrBlkInt(VIRGIN_HEADER, DUAL, 2, header);
388                 /* Header built and sent to DSP transmitter
        */
389                 /*************************************************
    ******/
390
391
392                 errorcode = WrBlkInt(VIRGIN_DATA, DUAL, tot/2, trans
    mit);
393                 if (errorcode != 0)
394                         printerror(5);
395
396
397
```



```
398                 menu = 0x1;
399                 rate = 0x4;
400                 packet = 0;
401
402                 control= menu | rate | packet;
403
404                 errorcode = Put32Bit(CONTROL_WORD, DUAL, control);
405                 if (errorcode != 0 )
406                         printerror(6);
407
408                 /*frame transmitted*/
409                 setviewport(xmit.left, xmit.top, xmit.right, xmit.bo
    ttom, 1);
410                 clearviewport();
411                 sprintf(msg, " %d", trans);
412                 outtextxy(10, 0, msg);
413                 sprintf(msg, " 18.93 kHz");
414                 outtextxy(10, 10, msg);
415                 sprintf(msg, " %d", ack);
416                 outtextxy(10, 20, msg);
417                 sprintf(msg, " %d",nack);
418                 outtextxy(10, 30, msg);
419                 sprintf(msg, " %d", index1+1);
420                 outtextxy(10, 40, msg);
421                 if (CSI == 1)
422                         rat=1.0;
423                 else if (CSI == 2)
424                         rat=.75;
425                 else if (CSI == 3)
426                         rat=.5;
427                 sprintf(msg, "%3.3E",rat);
428                 outtextxy(10, 50, msg);
429
430
431
432
433                 errorcode=WarmSelect(0x290);
434                 if ( errorcode == 0 )
435                         printerror(1);
436
437                 menu = 3;
438
439                 Put32Bit(MENU_OPTION, DUAL, menu);
440
441
442                 for(index2=0; index2<30000; index2++);
443                 for(index2=0; index2<30000; index2++);
444                 for(index2=0; index2<30000; index2++);
445                 for(index2=0; index2<30000; index2++);
446
447                 Put32Bit(STROBE_RCVR, DUAL, 0xffffl);
448
449                 for(index2=0; index2<30000; index2++);
450                 for(index2=0; index2<30000; index2++);
451                 for(index2=0; index2<30000; index2++);
452                 for(index2=0; index2<30000; index2++);
453
454
455                 while(Get32Bit(STROBE_HOST, DUAL)==01);
456
```

```
457                     Put32Bit(STROBE_HOST, DUAL, 01);
458                     response = Get32Bit(ACK0, DUAL);
459                     Put32Bit(ACK0, DUAL, 88881);
460
461                     if (response < 1000)
462                     {
463                             ack++;
464                             successframe++;
465                             infobits +=infoconstant;
466                             totalbits +=1056.0;
467
468                             if (response == 100)
469                                     CPC1++;
470                             else if (response == 200)
471                                     CPC2++;
472                             else if (response == 300)
473                                     CPC1CPC2++;
474
475                     }
476                     else
477                     {
478                             nack++;
479                             totalbits +=1056;
480                             if (response == 9999)
481                                     headfail++;
482                             else if (response == 6666)
483                                     crcfail++;
484                             else if (response == 8888)
485                                     lost++;
486                     }
487
488
489                     RdBlkInt(FLAG0P1, DUAL, tot/2+5, rcvr);
490
491
492             }
493
494
495
496
497
498
499
500     last:
501                     setviewport(rcv.left, rcv.top, rcv.right, rcv.bottom
        , 1);
502                     clearviewport();
503                     sprintf(msg, " %d", ack);
504                     outtextxy(10, 0, msg);
505                     sprintf(msg, " %d", lost);
506                     outtextxy(10, 10, msg);
507
508                     sprintf(msg, " %d", crcfail+headfail);
509                     outtextxy(10, 20, msg);
510                     sprintf(msg, " %7.4E",2.0*infobits/totalbits);
511                     outtextxy(10, 30, msg);
512
513                     setviewport(stat.left, stat.top, stat.right, stat.bo
        ttom, 1);
514                     clearviewport();
```



```
515                     setcolor(GREEN);
516                     sprintf(msg, " %d", ack);
517                     outtextxy(15, 5, msg);
518                     sprintf(msg, " %d", CPC1);
519                     outtextxy(215, 5, msg);
520                     sprintf(msg, " %d", CPC2);
521                     outtextxy(365, 5, msg);
522                     sprintf(msg, " %d", CPC1CPC2);
523                     outtextxy(515, 5, msg);
524
525
526                     setviewport(stat.left, maxy*.63, stat.right, maxy*.6
        7, 1);
527                     clearviewport();
528                     setcolor(BLUE);
529                     sprintf(msg, " %d", nack);
530                     outtextxy(15, 5, msg);
531                     sprintf(msg, " %d", headfail);
532                     outtextxy(215, 5, msg);
533                     sprintf(msg, " %d", crcfail);
534                     outtextxy(365, 5, msg);
535                     sprintf(msg, " %d", lost);
536                     outtextxy(515, 5, msg);
537                     setcolor(WHITE);
538
539                     setcolor(WHITE);
540                     WarmSelect(0x390);
541
542                     }while (response > 10001);
543
544             frame[index1] = trans - oldtrans;
545             oldtrans = trans;
546
547             }
548
549             printf("%c",beep);
550             printf("%c",beep);
551             printf("%c",beep);
552             printf("%c",beep);
553
554             for (index1=0; index1<20; index1++)
555                     output[index1]=0;
556
557             for (index1=0; index1<1000; index1++)
558                     output[frame[index1]]++;
559
560             setviewport(stat.left, maxy*.7, stat.right, maxy*.97, 1);
561             sprintf(msg, "Number of Transmissions required for Reception
        ");
562             outtextxy(5, 5, msg);
563
564             for (index1=0; index1<20; index1++)
565             {
566
567                     sprintf(msg, " %d", index1);
568                     outtextxy(30*index1, 20, msg);
569
570                     setcolor(GREEN);
571                     sprintf(msg, " %d ", output[index1]);
572                     outtextxy(30*index1, 30, msg);
```

```
573                     setcolor(WHITE);
574             }
575
576
577     /*
578             for (index2=0; index2<64; index2++)
579                     printf("%x ",transmit[ index2 ]);
580
581             printf("\n");
582             for (index2=8; index2<64; index2++)
583                     printf("%x ",rcvr[ index2 ]);
584     */
585             exit(1);
586
587
588
589
590     }
591
592
593     printerror(int number)
594     {
595             char msg[80];
596             int maxy;
597
598             switch (number)
599             {
600                     case 1:sprintf(msg, "Select Board 290h has failed.")
    ;
601                             break;
602                     case 2:sprintf(msg, "Loading RCVR.out has failed.");
603                             break;
604                     case 3:sprintf(msg, "Select Board 390h has failed.")
    ;
605                             break;
606                     case 4:sprintf(msg, "Loading F.OUT has failed.");
607                             break;
608                     case 5:sprintf(msg, "Downloading transmission frame
    to DSP board has failed.");
609                             break;
610                     case 6:sprintf(msg, "Writing Control Word to transmi
    tter has failed");
611                             break;
612                     default:sprintf(msg, "Problem with error print routi
    ne.");
613
614             }
615             maxy=getmaxy();
616             setcolor(GREEN);
617             setbkcolor(WHITE);
618             outtextxy(10, maxy*.7+15, msg);
619             /*getch();*/
620             exit(1);
621             return(0);
622
623     }
```

```
1     ;***************************************************************
      ******
2     ;       XMITADAP.asm  V1.00         Dec 92
3     ;                     V1.01         Jan 93
4     ;                     V1.02         Mar 93
5     ;                     v1.03         Apr 93
6     ;                     V2.00         Aug 93
7     ;                     V2.01         Sept 93
8     ;       The purpose of this code is to set up the dsp board environm
      ent,
9     ;       variables, and memory.  This code is used as the main interf
      ace
10    ;       between the PC and the dsp board. It places all necessary
11    ;       assembler and C routines in memory and then awaits in a simp
      le
12    ;       loop, where the ARQ shell can poke the appropriate info into
13    ;       DSP memory and then run the appropriate routine.
14    ;
15
16
17
18                    .include VARS.ASM
19                    .global .bss
20                    .global cinit                   ;init table (from li
      nker)                                                   .global _c_in
      t00                ;starting address (C standard)
21                    .global _interleaver
22                    .global _puncture
23                    .global _combineheader
24                    .global  _conv                ;the convolutional en
      coding
25                                                    ;routine
26                    .global  _polydiv             ;polnomial division r
      outine
27
28
29                    .sect    ".init"              ;interrupt section
30    RESET           .word    _c_int00             ;RESET -> start addr
      ess
31    INT0            .word    NO                   ;all others to dummy
       reti
32    INT1            .word    INT_TRANSMISSION     ;except the sync int
33    INT2            .word    NO
34    INT3            .word    NO
35    XINT0           .word    NO
36    RINT0           .word    NO
37    XINT1           .word    NO
38    RINT1           .word    NO
39    TINT0           .word    NO
40    TINT1           .word    NO
41    DINT            .word    NO
42
43
44    ;***************************************************************
45    ;       Data section to initially be loaded at $30000h but then
46    ;       moved to $809c00 (on chip ram).
47
48                    .data
49                    .word    5              ;constraint length
50                    .word    19h            ;polynomial1 =11001=1+x^3+x^
```



```
      4
51                    .word    17h            ;polynomial2 =10111=1+x+x^2+
      x^4
52                    .word    69665          ;CRC-CCITT
53                    .word    4374732215     ;CRC-32
54                    .word    3B1492AAh      ;flag for packet
55                    .word    0              ;Q_START --> 809c06
56                    .word    0              ;Q_START(1)
57                    .word    0              ;Q_START(2)
58                    .word    0              ;Q_START(3)
59                    .word    0              ;Q_START(4)
60                    .word    0              ;Q_START(5)
61                    .word    0              ;Q_START(6)
62                    .word    0              ;Q_START(7)
63                    .word    0              ;Q_END --> 809c0e
64                    .word    0              ;Q_END(1)
65                    .word    0              ;Q_END(2)
66                    .word    0              ;Q_END(3)
67                    .word    0              ;Q_END(4)
68                    .word    0              ;Q_END(5)
69                    .word    0              ;Q_END(6)
70                    .word    0              ;Q_END(7)
71                    .word    809c06h        ;Q_START --->809c16
72                    .word    809c0Eh        ;Q_END --->809c17
73                    .word    808042H        ;DIGITAL PORT ADDRESS --->80
      9C18
74    TAB_ENC         .WORD    5              ;pi/4 QPSK encoding table
75                    .WORD    6              ;---> 809c19
76                    .WORD    7
77                    .WORD    0
78                    .WORD    1
79                    .WORD    2
80                    .WORD    3
81                    .WORD    4
82                    .WORD    3
83                    .WORD    4
84                    .WORD    5
85                    .WORD    6
86                    .WORD    7
87                    .WORD    0
88                    .WORD    1
89                    .WORD    2
90                    .WORD    7
91                    .WORD    0
92                    .WORD    1
93                    .WORD    2
94                    .WORD    3
95                    .WORD    4
96                    .WORD    5
97                    .WORD    6
98                    .WORD    1
99                    .WORD    2
100                   .WORD    3
101                   .WORD    4
102                   .WORD    5
103                   .WORD    6
104                   .WORD    7
105                   .WORD    0
106   ICHAN           .WORD    7fff0000H      ;IBIT CHAN   1 volt
107                   .WORD    5a780000H      ;0.707
```

```
                .WORD   0               ;0.00
                .WORD   -5a780000H      ;-0.707
                .WORD   -7fff0000H      ;-1.00
                .WORD   -5a780000H      ;-0.707
                .WORD   0
                .WORD   5a780000H
QCHAN           .WORD   0
                .WORD   5a780000H
                .WORD   7fff0000H
                .WORD   5a780000H
                .WORD   0
                .WORD   -5a780000H
                .WORD   -7fff0000H
                .WORD   -5a780000H
                .WORD   809C19H
                .WORD   809C39H
                .WORD   809C41H


;***************************************************************************
;       Variables to be ued for initialization.

STACK           .usect   ".stack",STACK_SIZE

                .text
STACK_ADDR      .word   STACK                   ;address of stack
INIT_ADDR       .word   cinit                   ;address of init tables
PRIMCTRL        .word   00808064h               ;primary bus control address
EXPCTRL         .word   00808060h               ;expansion bus control address
TIMECTL1        .word   808030H                 ;timer 1 control
SERIAL0         .word   808042h                 ;FSX/DX/CLKX port control
TIMECTL2        .word   808020h                 ;timer 2 control
RSTCTRL         .word   601h                    ;reset value for timers
PERIOD          .word   808038h                 ;timer 1 period
COUNT           .word   55                      ;period value for timer 1
SETCTRL         .word   6c1h                    ;set value for timers
RAM1            .word   809c00h                 ;on chip ram area
DUALSTART       .word   30000h                  ;temp variables
DUALEND         .word   33300h                  ;change for 64k
DUALMEM         .word   300BFh
;***************************************************************************
;       The following code sets up the stack pointer and then
;       initializes the DSP hardware as outlined in the Technical
;       Reference Manual.
;
_c_int00:
                LDP     CODES                   ;get page of stored address
                LDI     @STACK_ADDR, SP         ;load the address in to SP
                LDI     SP, FP                  ;and into FP too
```



```

                LDI     @PRIMCTRL, AR0          ;Hardware specific init
                LDI     INITIAL, R0
                STI     R0, *AR0
                LDI     @EXPCTRL, AR0
                LDI     NULL, R0
                STI     R0, *AR0
                LDI     @SERIAL0, AR0           ;SET DIGITAL OUTPUT TO 0
                LDI     2H, R0
                STI     R0, *AR0

;***************************************************************************
;       This portion of code is absolutely necessary when mixing C
;       modules with assembly language.  It ensures that the
;       variables defined in the C module are properly initialized.

                LDP     CODES                   ;get page of stored address
                LDI     @INIT_ADDR, AR0         ;get address of init tables
                CMPI    -1, AR0                 ;if RAM model, skip init
                BEQ     init_done
                LDI     *AR0++, R1              ;get first count
                BZD     init_done               ;if 0, nothing to do
                LDI     *AR0++, AR1             ;get dest address
                LDI     *AR0++, R0              ;get first word
                SUBI    1, R1                   ;count - 1

do_init:        RPTS    R1                      ;block copy
                STI     R0, *AR1++
        ||      LDI     *AR0++, R0
                LDI     R0, R1                  ;move next count into R1
                BNZD    do_init                 ;if there is more, repeat
                LDI     *AR0++, AR1             ;get next dest address
                LDI     *AR0++, R0              ;get next first word
                SUBI    1, R1                   ;count - 1

;***************************************************************************
;       This code block copies all of the variables placed at $30000h
;       and moves them to the on chip memory area at $809c00 - $80a000

init_done:
        LDI     @DUALSTART, AR0
        LDI     @RAM1, AR1
        LDI     *AR0++, R0              ;since parallel instruction
                                        ;coming up must initialize R0

        RPTS    DATALENGTH
        LDI     *AR0++, R0
```

```
204         ||STI     R0, *AR1++
205
206
207     ;***************************************************************
        ****
208     ;
209             LDI @DUALSTART, R6              ;clear DUAL memory
210             LDI @DUALEND, R7                ;$30000 -->$33300
211             CALL CLEAR
212
213             LDP ONCHIP                      ;initialize variables
214             LDI 0, R0                   ;used in transmission interrupt
215             LDI 0, R1                       ;routine
216             LDI 32, R2
217             STI R0, @SINE_POINTER           ;sine_pointer = $809c08
218             STI R0, @COSINE_POINTER         ;cosine_pointer=$809c08
219             STI R2, @POINT_COUNT            ;point_count = 32
220             STI R1, @DATA_WORD              ;data_word = 0
221             STI R1, @CURRENT_ADDRESS        ;current_address = 0
222             STI R1, @END_ADDRESS            ;end_address = 0
223             STI R1, @Q_OFFSET               ;q_offset = 0
224             STI R1, @TRANSMISSION           ;transmission = 0 (not busy)
225             STI R1, @Q_OFF_TRANS            ;q_off_trans = 0
226             STI R1, @GET_NEWFRAME_FLAG      ;flag = 0
227             LDI 40H, BK                     ;BK = 40H
228
229
230     ;***************************************************************
        ****
231     ;       This section places the flag for the frames in the appropria
        te
232     ;       memory locations.
233
234     ADD_FLAG:
235             LDP ONCHIP
236             LDI @FLAG, R0
237             LDP CODES
238             LDI @FLAG0P1, AR1
239             ;LDI 15, RC                      ;16 packets
240             ;RPTB ENDLOOP1
241             addi 63h, ar1                   ;cut it out because interef
        eres
242                                             ;with combine area must be c
        lean!!
243
244             STI R0, *AR1
245     ENDLOOP1:
246             ;ADDI 21h, AR1                  ;get next flag address
247
248
249
250
251     ;strip information from CONTROL_WORD
252
253     START_OF_MAIN_ROUTINE:
254
255             LDP DUAL
256     MENU:   LDI @CONTROL_WORD, R0
257             LDI MENU_MASK, R1               ;get MENU_OPTION
258             AND R0, R1, R2
```



```
259             STI R2, @MENU_OPTION
260             BZ MENU                         ;if no choice loop back
261
262
263             ;LSH -2, R0
264             ;AND R0, R1, R3                  ;get  data RATE
265             ;STI R3, @RATE
266
267             LSH -2, R0
268             LDI PACKET_MASK, R1             ;get PACKET_NUM
269             AND R0, R1, R4
270             STI R4, @PACKET_NUM
271
272             LDP CODES
273             LDI @VIRGIN_HEADER, AR1
274             LDI *AR1++, R0
275             LDI LENGTH_MASK, R1             ;get LENDATA0
276             LSH -22, R0
277             AND R0, R1, R5
278             ldi*ar1, r0
279             and 3, r0                       ;get puncture matrix#
280             ldi *ar1, r1
281             and 12, r1
282             lsh -2, r1                      ;get rate to be used
283             LDP DUAL
284             sti r0, @CODE
285             sti r1, @RATE
286             STI R5, @LENDATA0
287
288
289             CMPI 1, R2
290             BZ OPTION1
291             CMPI 2, R2
292             BZ OPTION2
293             CMPI 3, R2
294             BZ OPTION3
295             LDI 2, R7
296             BR ERROR
297     DEAR:       BR DEAR
298     ;***************************************************************
        ***
299     ;       OPTION 1 - ADAPTIVE SCHEME
300     ;                  Construct frame from given header and data. Use
301     ;                  rate 1/2 for header and appropriate CPC matrix
302     ;                  chosen rate of data packet.
303     OPTION1:
304
305             ;*******************************************************
        ****
306             ;CRC calculation and appending to header information stored
307             ;at VIRGIN_HEADER
308
309             LDP CODES
310             LDI @VIRGIN_HEADER, AR1         ;get original header and place
311             LDI *AR1++, R0                  ;at HEADBUF1 padded with x^16
312             LDI *AR1, R1                    ;zeros
313             LDI @HEADBUF1, AR5
314
315             LDI @HIGH_MASK, R2
316             LSH 16, R1
```

```
317             AND R0, R2, R3
318             LSH -16, R3
319             OR R1, R3
320             LSH 16, R0
321             STI R0, *AR5++
322             STI R3, *AR5--
323
324             LDI 17, R0                      ; K constraint length
325             LDI @CRC_CCITT, R1              ; POLY divisor
326             LDI 2, R2                       ; length of header in 32 bit
        words
327             LDI @HEADBUF2, R3               ; where to place RESULT (not
        needed
328                                             ; in this CRC case)
329
330             PUSH AR1                        ; save VIRGIN_HEADER+1
331             PUSH R3                         ; *RESULT = HEADBUF2
332             PUSH R2                         ; TOTAL = 2
333             PUSH R1                         ; POLY = CRC_CCITT
334             PUSH AR5                        ; MESGDATA = HEADBUF1
335             PUSH R0                         ; K = 17
336
337
338             CALL _polydiv                   ; Do CRC calculation
339                                             ; CRC checksum returned in R
        0
340
341
342             SUBI 5,SP                       ; clean stack
343             POP AR1                         ; get back VIRGINHEADER+1
344             LSH 5, R0                       ; shift CRC before placing
345             LDI *AR1, R1                    ; in VIRGIN_HEADER+1
346             OR R1, R0
347             STI R0, *AR1
348
349             ;End of CRC calculation and appending to header info
350
351             ;***************************************************************
        **
352             ;Convolutional encoding of header information stored at
353             ;VIRGIN_HEADER.  Resulting encoded header is stored at
354             ; Adder1 output encoded header bits-----> HEADBUF1
355             ; Adder2 output encoded header bits-----> HEADBUF2
356
357
358             LDI @VIRGIN_HEADER, AR0         ; data to be convolved
359             LDI @HEADBUF1, AR1              ; address of adder1 bits
360             LDI @HEADBUF2, AR2              ; address of adder2 bits
361             LDI 2, R0                       ; # of 32 bit words to convo
        lve
362
363             PUSH R0                         ; SIZE
364             PUSH AR2                        ; MESGP2
365             PUSH AR1                        ; MESGP1
366             PUSH AR0                        ; MESG
367
368             CALL _conv
369
370             SUBI 4, SP
371             ;End of convolutional encoding of header
```



```
372
373
374             LDP CODES
375             LDI @FRAMEBUFP1, AR3            ;where to place result
376             LDI 64, R0                      ;bit length of adder1
377             LDI @HEADBUF1, AR0              ;adder 1 output bits
378             LDI @HEADBUF2, AR1              ;adder 2 output bits
379
380             PUSH R0
381             PUSH AR3
382             PUSH AR1
383             PUSH AR0
384             CALL _combineheader
385             SUBI 4, SP
386
387             ;End of header construction
388
389             ;***************************************************************
        ******
390             ;  Data CRC calculation
391
392
393
394             LDI @VIRGIN_DATA, AR0           ;start of data
395
396             LDI 33, R0                      ;K constraint length
397             LDI @CRC_32, R1                 ;POLY
398             LDI @DATABUFP1, R3              ;address to store RESULT
399             LDI @VIRGIN_DATA, AR3           ;*MESGDATA
400             SUBI 1, AR3                     ; same as premultiply by x^K
401
402             LDP DUAL
403             LDI @RATE, R7
404             LDP CODES
405             CMPI 1,R7
406             LDIZ 27, R2                     ;26 DATA WORDS + BLANK CRC
407             CMPI 2, R7
408             LDIZ 20, R2                     ;19 DATA WORDS + BLANK CRC
409             CMPI 3, R7
410             LDIZ 13, R2                     ;12 DATA WORDS + BLANK CRC
411             CMPI 0, R7
412             BZ ERROR
413
414
415
416             LDI R2, R7
417             PUSH R7
418             PUSH R3                         ;*RESULT (not used)
419             PUSH R2                         ;TOTAL = length in 32 bit wo
        rds +
420                                             ; 1 for CRC32
421             PUSH R1                         ;POLY = CRC_32
422             PUSH AR3                        ;MESGDATA = VIRGIN_DATA
423             PUSH R0                         ;K = constraint length
424
425             CALL _polydiv
426
427             SUBI 5, SP
428             ADDI 1, AR3                     ;Point to @VIRGIN_DATA
429
```

```
430           POP R7
431           SUBI 1, R7
432           ADDI R7, AR3
433           STI R0, *AR3                    ;place CRC at end of data
434
435
436           ;At this point the data exists @VIRGIN_DATA with a 32 bit CR
      C
437           ;appended to it right after the last data bit
438           ;0X30013 - 0X30025 DATA    CRC  @ 30026
439
440
441           ;*************************************************************
      ******
442           ;Convolutional encoding of data information @VIRGIN_DATA.
443           ;Resulting encoded data is stored at:
444           ; P1 encoded data ---> DATABUFP1
445           ; P2 encoded data ---> DATABUFP2
446
447
448           ;need to get number of 32 bit words to convolve and also add
       K
449           ; (constraint length) bits to
450           ;                 1> rounded up word count, and
451           ;                 2> bit count of encoded data
452
453           LDP CODES
454           LDI @VIRGIN_DATA, AR0
455           LDI @DATABUFP1, AR1
456           LDI @DATABUFP2, AR2
457           LDP DUAL
458           LDI @RATE, R7
459           LDP CODES
460           CMPI 1,R7
461           LDIZ 28, R4                     ;26 DATA WORDS + BLANK CRC
462           CMPI 2, R7
463           LDIZ 21, R4                     ;19 DATA WORDS + BLANK CRC
464           CMPI 3, R7
465           LDIZ 15, R4                     ;12 DATA WORDS + BLANK CRC
466
467           PUSH R4                         ;SIZE
468           PUSH AR2                        ;MESGP2
469           PUSH AR1                        ;MESGP1
470           PUSH AR0                        ;MESG
471
472           CALL _conv
473
474           SUBI 4, SP
475           ;End of convolutional encoding of data
476           ;*************************************************************
      ****
477           ; Construct frame P1 and P2 and place at:
478           ; frame P1 ---> FRAMEBUFP1
479           ; frame P2 ---> FRAMEBUFP2
480           ; This procedure is made a bit tedious because if the header
481           ; length is not a multiple of 32 then we must shift and OR
482           ; all the data bits so that they can be appended directly
483           ; after the header bits!!
484
485
```



```
486           LDI @DATABUFP1, AR0             ;pointer to data bits P1
487           LDI @DATABUFP2, AR1             ;pointer to data bits P2
488           LDI @FRAMEBUFP1, AR2            ;pointer to frame P1
489           LDI @FRAMEBUFP2, AR3            ;pointer to frame P2
490
491
492           ;*************************************************************
      ******
493           ;Puncture and combine bits of adder1 and adder2
494
495
496     PUNCTURE_AND_COMBINE:
497           LDP DUAL
498           LDI @RATE, R7
499           LDP CODES
500           CMPI 1,R7
501           BZ RATE1
502           CMPI 2, R7
503           BZ RATE75
504           CMPI 3, R7
505           BZ RATE5
506           CMPI 0, R7
507           BZ ERROR
508
509           ;data bits of adder 1 are @DATABUFP1
510           ;data bits of adder 2 are @DATABUFP2
511
512     RATE1:
513           LDP DUAL
514           LDI @CODE, R7
515           ldp CODES
516           CMPI 1, R7
517           BZ CODE1
518           CMPI 2, R7
519           BZ CODE2
520           BZ ERROR
521
522     RATE75:
523           LDP DUAL
524           LDI @CODE, R7
525           ldp CODES
526           CMPI 1, R7
527           BZ CODE75_CPC_ONE
528           CMPI 2, R7
529           BZ CODE75_CPC_TWO
530           BZ ERROR
531     RATE5:
532           LDP CODES
533
534           LDI @DATABUFP1, AR0
535           LDI @DATABUFP2, AR1
536           BR RATE_HALF
537
538
539
540
541     PUSH_AND_CALL:
542           PUSH R0                 ; Routine used by all rates to push
543           PUSH AR1                ; parameters and call puncture modul
      e
```

```
544             PUSH AR3
545             PUSH R1
546
547             CALL _puncture
548
549             SUBI 4, SP
550             RETS
551
552     CODE1:
553             LDI 896, R0
554             LDI 4, R1                       ; Puncture Adder1 bits by 1/
        2
555             LDI @DATABUFP1, AR1
556             LDI @DATABUFP3, AR3
557             CALL PUSH_AND_CALL
558
559             LDI 896, R0
560             LDI 5, R1                       ;Puncture Adder2 bits by 1/2
561             LDI @DATABUFP4, AR3
562             LDI @DATABUFP2, AR1
563             CALL PUSH_AND_CALL
564             BR COMBINE1
565     CODE2:
566             LDI 896, R0
567             LDI 6, R1                       ; Puncture Adder1 bits by 1/
        2
568             LDI @DATABUFP1, AR1
569             LDI @DATABUFP3, AR3
570             CALL PUSH_AND_CALL
571
572             LDI 896, R0
573             LDI 7, R1                       ;Puncture Adder2 bits by 1/2
574             LDI @DATABUFP4, AR3
575             LDI @DATABUFP2, AR1
576             CALL PUSH_AND_CALL
577             BR COMBINE1
578
579
580     CODE75_CPC_ONE:
581             LDI 672, R0
582             LDI 0, R1                       ; Puncture Adder1 bits by 3/
        4
583             LDI @DATABUFP1, AR1
584             LDI @DATABUFP3, AR3
585             CALL PUSH_AND_CALL
586
587             LDI 672, R0
588             LDI 1, R1                       ;Puncture Adder2 bits by 3/4
589             LDI @DATABUFP4, AR3
590             LDI @DATABUFP2, AR1
591             CALL PUSH_AND_CALL
592             BR COMBINE
593
594     CODE75_CPC_TWO:
595             LDI 672, R0
596             LDI 2, R1               ;Puncture Adder1 by 3/4
597             LDI @DATABUFP1, AR1
598             LDI @DATABUFP3, AR3
599             CALL PUSH_AND_CALL
600
```



```
601             LDI 672, R0
602             LDI 3, R1               ;Puncture Adder2 by 3/4
603             LDI @DATABUFP4, AR3
604             LDI @DATABUFP2, AR1
605             CALL PUSH_AND_CALL
606             BR COMBINE
607
608     RATE_HALF:
609             LDI 448, R0                     ;14 WORDS * 32 BITS
610             LDI @FRAMEBUFP2, AR3
611             PUSH R0
612             PUSH AR3
613             PUSH AR1
614             PUSH AR0
615             CALL _combineheader
616             SUBI 4, SP
617             BR CONSTRUCT
618     COMBINE1:
619             LDI @DATABUFP3, AR0
620             LDI @DATABUFP4, AR1
621
622             LDP CODES
623
624
625             LDI @FRAMEBUFP2, AR3
626             LDI 896, R0                     ;21 WORDS * 32 BITS
627             PUSH R0
628             PUSH AR3
629             PUSH AR1
630             PUSH AR0
631             CALL _combineheader
632             SUBI 4, SP
633             BR CONSTRUCT
634
635
636     COMBINE:
637             LDI @DATABUFP3, AR0
638             LDI @DATABUFP4, AR1
639
640             LDP CODES
641
642
643             LDI @FRAMEBUFP2, AR3
644             LDI 672, R0                     ;21 WORDS * 32 BITS
645             PUSH R0
646             PUSH AR3
647             PUSH AR1
648             PUSH AR0
649             CALL _combineheader
650             SUBI 4, SP
651
652             ;End of header construction
653
654     CONSTRUCT:
655             LDP CODES
656             LDI @FRAMEBUFP1, AR1
657             ADDI 4,AR1
658             LDI @FRAMEBUFP2, AR0
659             ldp DUAL
660             LDI 27, R0                      ;copy possible 27 words if r
```

```
     ate 1/2
661           LDI *AR0++, R1
662           RPTS R0
663
664           LDI *AR0++, R1
665          ||STI R1, *AR1++
666
667
668
669     ;      BR TRANY
670
671           ;End of constructing Frames P1 and P2
672           ;*************************************************************
     *******
673           ;       Now we must save the frame P2 for future possible
674           ;       retransmission and interleave frame P1 which will
675           ;       also be save and then queued for transmission.
676           ;
677           ;       Save frame P2          ---> @PACKETxHARDP2
678           ;       Interleave P1 and save ---> @PACKETxHARDP1
679
680
681
682           nop
683           nop
684           nop
685           ldi BLOCKS, r2
686
687     INTERLEAVE_MORE:
688           PUSH R2
689           PUSH AR2
690           PUSH AR0
691
692           CALL _interleaver
693           POP AR0
694           POP AR2
695           POP R2
696
697
698           ;ADDI 4, AR2                    ;not required, subroutine au
     tomatically
699           ADDI DEINT_ROW, AR0                     ;increments AR2 cont
     ents
700           SUBI 1, R2
701           BNZ INTERLEAVE_MORE
702
703           ;Completed interleaving of frame P1 and saving of frame P1 &
      P2
704           ;*************************************************************
     ****
705
706          ;Now set up for transmission
707
708          LDI @PACKET3HARDP1, AR0
709          SUBI 1, AR0
710          LDI AR0, AR3
711          ADDI 34, AR3
712          LDI AR0, R7
713          LDI AR3, R6
714          CALL QUEUE
```



```
715
716
717          LDP DUAL
718          LDI 0, R6
719          STI R6, @CONTROL_WORD
720
721          LDP CODES
722          LDI @FRAMEBUFP1, R6
723          LDI R6, R7
724          ADDI 40H, R7
725          CALL CLEAR
726          LDI @DATABUFP3, R6
727          LDI R6, R7
728          ADDI 200h, R7
729          CALL CLEAR
730
731          BR START_OF_MAIN_ROUTINE
732
733
734
735     TRANY:
736           LDI 31, R0
737           LDP CODES
738           LDI @FRAMEBUFP1, AR0
739           LDI @PACKET3HARDP1, AR1                ;COPY FRAME TO SLOT
     3
740           LDI *AR0++, R1
741           RPTS R0
742           LDI *AR0++, R1
743          ||STI R1, *AR1++
744
745          ;Now set up for transmission
746
747          LDI @PACKET3HARDP1, AR0
748          SUBI 1, AR0
749          LDI AR0, AR3
750          ADDI 33, AR3
751          LDI AR0, R7
752          LDI AR3, R6
753          CALL QUEUE
754
755
756          LDP DUAL
757          LDI 0, R6
758          STI R6, @CONTROL_WORD
759
760          LDP CODES
761          LDI @FRAMEBUFP1, R6
762          LDI R6, R7
763          ADDI 40H, R7
764          CALL CLEAR
765          LDI @DATABUFP3, R6
766          LDI R6, R7
767          ADDI 200h, R7
768          CALL CLEAR
769
770          BR START_OF_MAIN_ROUTINE
771
772
773
```

```
774   ;*****************************************************************
      ******
775   ;
776   ;       The purpose of this option is to encode the data using a rat
      e
777   ;       1/2 convolutional encoder and the two polynomials defined in
778   ;       VARS.ASM. The encoded data is placed in a frame and sent thr
      ough
779   ;       the channel.
780   ;       Main purpose for this option is to test perfomance of the Vi
      terbi
781   ;       Decoder in the RCVRCPC.ASM module.
782
783   OPTION2:
784           LDP CODES
785           LDI @VIRGIN_HEADER, AR0                 ;address of data to
      convolve
786           LDI @FRAMEBUFP1, AR1                    ;P1 bit buffer
787           LDI @FRAMEBUFP2, AR2                    ;P2 bit buffer
788           LDI 16, R0                              ;# of 32 bit words t
      o encode
789
790           PUSH R0                                 ;SIZE
791           PUSH AR2                                ;MESGP2
792           PUSH AR1                                ;MESGP1
793           PUSH AR0                                ;MESG
794
795           CALL _conv                              ;convolve data
796           SUBI 4, SP
797
798           ;Now the P1 bits and P2 bits must be combined
799
800           LDI 512, R0
801           LDI @FRAMEBUFP1, AR0
802           LDI @FRAMEBUFP2, AR1
803           LDI @DATABUFP1, AR3
804
805           PUSH R0                                 ;# BITS
806           PUSH AR3                                ;address of convolve
      d data
807           PUSH AR1                                ;P1 bits
808           PUSH AR0                                ;P2 bits
809
810           CALL _combineheader
811           SUBI 4, SP
812
813
814           ;Now set up to interleave & transmit frame through channel
815
816           LDI @PACKET3HARDP1, AR2         ;slot address of frame
817           LDI @DATABUFP1, AR0         ;random data start address
818
819
820
821           ldi BLOCKS, r2
822
823   INTERLEAVE_MORE2:
824           PUSH R2
825           PUSH AR2
826           PUSH AR0
```



```
827
828           CALL _interleaver
829           POP AR0
830           POP AR2
831           POP R2
832
833
834           ;ADDI 4, AR2                    ;not required, subroutine au
      tomatically
835           ADDI DEINT_ROW, AR0                     ;increments AR2 cont
      ents
836           SUBI 1, R2
837           BNZ INTERLEAVE_MORE2
838
839           ;frame is now interleaved and placed into the slot 30123
840           ;and it also has a flag appended to it
841
842           ldi @PACKET3HARDP1, ar2
843           SUBI 1, AR2                     ;AR2 points to flag @ start
      of frame
844           ldi ar2, ar0
845           ADDI 34, AR0
846           LDI AR2, R7                     ;start address of frame
847           LDI AR0, R6                     ;end address + 2
848           CALL QUEUE                      ;transmit frame
849
850           LDP CODES                       ;clear memory @$30000 to
851           LDI @DUALSTART, R6              ;$300bf and branch back for
      next
852           LDI @DUALMEM, R7                ;option
853           CALL CLEAR
854           BR START_OF_MAIN_ROUTINE
855
856
857
858
859
860   ;*****************************************************************
      *
861   ;       OPTION 3 - Take 992 random data bits delivered @VIRGIN_HEADE
      R
862   ;                  and look at control word in order to determine
863   ;                  which P1 frame slot to place data in.  Then trans
      mit
864   ;                  random UNCODED data with a flag.
865   ;
866   ;                  The purpose of this option is to allow the user t
      o
867   ;                  obtain the Bit Error Rate (with a PC host Program
      )
868   ;                  of the modulation scheme being used.  This option
869   ;                  takes the random data, adds a flag and just
870   ;                  transmits the 1024 bit frame through the channel.
871
872   OPTION3:
873
874           LDP CODES
875           LDI @PACKET3HARDP1, AR2         ;slot address of frame
876           LDI @VIRGIN_HEADER, AR0         ;random data start address
877
```

```
878
879
880           ldi BLOCKS, r2
881
882   INTERLEAVE_MORE3:
883           PUSH R2
884           PUSH AR2
885           PUSH AR0
886
887           CALL _interleaver
888           POP AR0
889           POP AR2
890           POP R2
891
892
893           ;ADDI 4, AR2                    ;not required, subroutine au
      tomatically
894           ADDI DEINT_ROW, AR0                     ;increments AR2 cont
      ents
895           SUBI 1, R2
896           BNZ INTERLEAVE_MORE3
897
898           ;frame is now interleaved and placed into the slot 30123
899           ;and it also has a flag appended to it
900
901           ldi @PACKET3HARDP1, ar2
902           SUBI 1, AR2                     ;AR2 points to flag @ start
      of frame
903           ldi ar2, ar0
904           ADDI 34, AR0
905           LDI AR2, R7                     ;start address of frame
906           LDI AR0, R6                     ;end address + 2
907           CALL QUEUE                      ;transmit frame
908
909           LDP CODES                       ;clear memory @$30000 to
910           LDI @DUALSTART, R6              ;$300bf and branch back for
      next
911           LDI @DUALMEM, R7                ;option
912           CALL CLEAR
913           BR START_OF_MAIN_ROUTINE
914
915   ;*****************************************************************
      ***
916   ;       This section of code places the start and end address of a
917   ;       frame to be transmitted in the queue.
918   ;       Note: before calling this routine ensure that
919   ;       R6 = end address of frame + 2 locations
920   ;       R7 = start address of frame
921
922   QUEUE:
923
924           LDP ONCHIP
925           LDI @Q_START, AR0
926   KEEPGOING:
927           LDI @Q_OFFSET, IR0
928           LDI *+AR0(IR0), R1              ;QUEUE is full so keep loopi
      ng until
929           BNZ KEEPGOING                   ;this slot is available
930
931           STI R7, *+AR0(IR0)              ;Place start and end address
```



```
       in
932           LDI @Q_END, AR1                 ;QUEUE
933           STI R6, *+AR1(IR0)
934
935           LDI  IR0, R0                    ;Adjust offset value which i
      s used
936           ADDI 1, R0                      ;to make the QUEUE a circula
      r
937           CMPI 8, R0                      ;buffer
938           LDIZ 0, R0
939           STI R0, @Q_OFFSET
940
941           LDI @TRANSMISSION, R0           ;Check if busy transmitting
      if not
942           BZ BEGIN_TRANS                  ;begin transmission
943           RETS
944
945   ;*****************************************************************
      ******
946   ;       This section of code is used to begin transmission of a fram
      e
947   ;       and initialize various parameters for transmission interrupt
948   ;       routine.
949   ;       It is used for the first frame transmission as well as any
950   ;       subsequent frame transmission when the interrupt has disable
      d
951   ;       its self.
952
953   BEGIN_TRANS:
954   ;       Set up timer 1 which will be used to trigger interrupt 1
955   ;       every 6.6 mircoseconds.  55/.12 = 6.6mircoseconds
956                 LDP CODES
957                 LDI     @TIMECTL2, AR1
958                 LDI     @TIMECTL1, AR0
959                 LDI     @RSTCTRL, R0
960                 STI     R0, *AR0
961                 STI     R0, *AR1                ;Reset timers 0, and
       1
962
963                 LDI @PERIOD, AR1                ;Set timer 1 to trig
      ger
964                 LDI @COUNT, R0                  ;every 6.6microsecon
      ds
965                 STI R0, *AR1
966                 LDI @SETCTRL, R0
967                 STI R0, *AR0
968           LDP ONCHIP
969           LDI 0FFFFH, R0
970           LDI @Q_START,AR0
971           STI R0, @TRANSMISSION           ;set transmission flag to bu
      sy
972
973           LDI @Q_OFF_TRANS, IR0
974           LDI *+AR0(IR0), AR3             ;AR3 = address of frame star
      t
975
976   KEEP_TRANS:
977
978
979
```

```
980           ;LDI @SINE0, R0
981           LDI 0, R0
982           STI R0, @SINE_POINTER          ;sine_pointer = 809c08
983           STI R0, @COSINE_POINTER        ;cosine_pointer = 809c08
984           LDI 8, R0
985           STI R0, @POINT_COUNT           ;point_count = 8
986           ldi 16, r0
987           sti r0, @DIBIT_COUNT
988           LDI *AR3++, R0                 ;place first data in DATA_WO
      RD
989           STI R0, @DATA_WORD
990
991           LDI AR3, R0
992           STI R0, @CURRENT_ADDRESS       ;save incremented pointer
993           LDI @Q_END, AR0
994           LDI *+AR0(IR0), R0
995           STI R0, @END_ADDRESS           ;save end address
996
997           ldi 0, r0
998           sti r0, *+ar0(ir0)
999           ldi @Q_START, ar0
1000          sti r0, *+ar0(ir0)
1001
1002
1003          LDI IR0, R0                    ;Adjust offset used to make
      QUEUE
1004          ADDI 1, R0                     ;a circular buffer
1005          CMPI 8, R0
1006          LDIZ 0, R0
1007          STI R0, @Q_OFF_TRANS
1008
1009          LDI @GET_NEWFRAME_FLAG, R0
1010          BZ ENABLE_RET
1011          LDI 0, R0
1012          STI R0, @GET_NEWFRAME_FLAG
1013          RETS
1014
1015    ENABLE_RET:
1016          OR 2H, IE                       ;ENABLE INTERRUPT 1
1017          OR 2000H, ST                    ;ENABLE GLOBAL INTERRUPTS
1018          RETS
1019
1020
1021    ;*************************************************************************
      ***
1022    ;This section clears memory chunks specified by AR0 --> AR1
1023
1024    CLEAR:
1025          SUBI R6, R7
1026          LDIN 1, R7
1027          BN ERROR
1028          LDI NULL, R0
1029          LDI R6, AR0
1030          RPTS R7
1031          STI R0, *AR0++
1032          RETS
1033    ;*************************************************************************
      ****
1034    ;This section is used for debugging various errors
1035
```



```
1036    ERROR:
1037          LDP DUAL
1038          STI R7, @ERROR_NUM
1039    DEAD  BR DEAD
1040
1041
1042
1043    ;*************************************************************************
      ****
1044    ;       This interrupt 1 is responsible for the actual transmission
      of
1045    ;       the frame.  It consists of 30 instructions (if it does not j
      ump
1046    ;       to get a NEWDIBIT).
1047    ;               30 X 60ns = 1800ns
1048    ;
1049    ;       Interrupt 1 occurs every  55 (count value)/.12 = 6.6 microse
      c
1050    ;               6.6microsec/60ns = 110 instructions
1051    ;
1052    ;               110 - 30 = 80 instruction between interrupts
1053    ;
1054    ;       Note: NEWDIBIT is called after 32 points have been output fo
      r
1055    ;             each symbol.  NEWDIBIT is very time consuming and woul
      d
1056    ;             practically take the entire 6.6 microseconds.
1057
1058
1059    INT_TRANSMISSION:
1060
1061          PUSH ST
1062          XOR 2000H, ST
1063          PUSH R0                              ;save registers before using
       them
1064          PUSH R1                              ;in interrupt routine
1065          PUSH R2
1066          PUSH AR0
1067          PUSH AR1
1068          PUSH AR3
1069
1070          push ir0
1071          PUSH DP
1072
1073          LDP ONCHIP
1074          LDI @IBIT_POINTER, AR0          ;ar0 = sine table pointer
1075          LDI @QBIT_POINTER, AR1        ;ar1 = cosine table pointer
1076
1077          LDI @SINE_POINTER, IR0
1078
1079
1080          LDI *+AR0(IR0), R1          ;output value to channel and upda
      te
1081          LDI *+AR1(IR0), R2          ;table pointers respectively
1082          STI R1, @ADCHANA
1083          STI R2, @ADCHANB
1084
1085          ldi     @SERIALO, AR3
1086          LDI     2H, R2
1087          STI     R2, *AR3
```

```
1088
1089          LDI @POINT_COUNT, R0            ;check if 32 points per symb
      ol
1090          ;SUBI 1, R0                      ;has been output to channel
1091          CMPI 4, R0
1092
1093
1094          BNZ NOPULSE
1095    PULSE_RCVR:
1096
1097                                          ;if 16th point then output
1098          LDI     6H, R2                  ;a pulse on the serial 0 por
      t
1099          STI     R2, *AR3
1100    NOPULSE:
1101
1102          SUBI 1, R0
1103          STI R0, @POINT_COUNT
1104          BZ NEWDIBIT                     ;if so get another dibit
1105
1106    FINISH:
1107          ;STI AR0, @SINE_POINTER          ;save new pointers
1108          ;STI AR1, @COSINE_POINTER
1109
1110    FINISH2:
1111          POP DP                          ;restore registers before re
      turning
1112          pop ir0
1113          POP AR3                         ;from interrupt routine
1114          POP AR1
1115          POP AR0
1116          POP R2
1117          POP R1
1118          POP R0
1119          POP ST
1120          OR 2000h, ST
1121          RETI
1122
1123
1124    NEWDIBIT:
1125          LDI @GET_NEWFRAME_FLAG, R0      ;if newframe flag set
1126          BN NEWFRAME                    ;get newframe
1127          LDI 8, R0
1128          LDI 3, R1                       ; R1 = 3 mask for 2 lsb's
1129          STI R0, @POINT_COUNT            ;initialize POINT_COUNT = 32
1130          LDI @DATA_WORD, R0              ; R0 = DATA_WORD
1131          AND3 R0, R1, R2                 ; R2 = DATA_WORD & 3
1132          LSH -2, R0                      ;shift DATA_WORD by 2
1133          STI R0, @DATA_WORD
1134          LDI @DIBIT_COUNT, R0
1135          SUBI 1, R0
1136          STI R0, @DIBIT_COUNT
1137
1138          bnz NEXT
1139
1140          ;ldi @COSINE_POINTER, ar0         ;random bit stream output
1141          ;ldi *ar0++, r0
1142          ;sti r0, @DATA_WORD
1143          ;ldi ar0, r0
1144          ;cmpi 0ffffh, r0
```



```
1145          ;ldiz 0, r0
1146          ;sti r0, @COSINE_POINTER
1147          ;ldi 16, r0
1148          ;sti r0, @DIBIT_COUNT
1149
1150          ;ldiz @FLAG, r0                 ;continuous flag output
1151          ;sti r0, @DATA_WORD
1152          ;LDI 16, R0
1153          ;STI R0, @DIBIT_COUNT
1154
1155          BZ NEWWORD                      ;frame output
1156
1157    NEXT:   LDI @SINE_POINTER, R0                    ;use DIBIT to dec
      ide how much
1158          MPYI 8, R2
1159          ADDI R2, R0
1160          LDI R0, IR0
1161          LDI @TABLE_ENC, AR0
1162          LDI *+AR0(IR0), R1
1163          STI R1, @SINE_POINTER
1164
1165
1166
1167
1168
1169
1170
1171          BR FINISH2
1172
1173    NEWWORD:
1174          LDI 16, R0
1175          STI R0, @DIBIT_COUNT
1176
1177          LDI @CURRENT_ADDRESS, AR3       ;get pointer to data to tran
      smit
1178          LDI *AR3++, R0                  ;get data pointed to
1179          STI AR3, @CURRENT_ADDRESS       ;save incremented pointer
1180          STI R0, @DATA_WORD              ;save data
1181          CMPI @END_ADDRESS, AR3          ;check if end of frame reach
      ed
1182          BZ SETFLAG
1183          BR NEXT                         ;if not do NEXT
1184
1185    SETFLAG:
1186          LDI -1, R0
1187          STI R0, @GET_NEWFRAME_FLAG
1188          BR NEXT
1189
1190    NEWFRAME:
1191          ;MIGHT WANT TO ADD TIME DELAY BETWEEN FRAMES
1192          LDI @Q_START, AR0
1193          LDI @Q_OFF_TRANS, IR0
1194          LDI *+AR0(IR0), AR3
1195          LDI AR3, R0                     ;R0 will set ST flag
1196          BZ STOP_TRANS
1197          CALL KEEP_TRANS
1198          BR FINISH2
1199
1200    STOP_TRANS:
1201          LDI 0, R0
```

```
1202
1203
1204          STI R0, @TRANSMISSION
1205          STI R0, @GET_NEWFRAME_FLAG
1206
1207          POP DP                          ;restore registers before re
      turning
1208          pop ir0
1209          POP AR3                         ;from interrupt routine
1210          POP AR1
1211          POP AR0
1212          POP R2
1213          POP R1
1214          POP R0
1215          POP ST
1216          XOR 2000H, ST
1217          RETI
1218
1219    UPDATE1:
1220          LDI *AR0--(DELTA)%, R0          ;output value to channel and
      update
1221          LDI *AR1++(DELTA)%, R1          ;table pointers respectively
1222
1223         ; SUBI DELTA, AR0
1224         ; ADDI DELTA, AR1
1225         ; BR UPDATE2
1226    ;*****************************************************************
      ***
1227    ;         All other interrupts simply return
1228
1229    NO:       RETI
1230
1231
1232
1233          .end
```

```
1     ;***************************************************************
      ******
2     ;       RCVRADAP.asm      V1.00         Dec 92
3     ;                         V1.01         Apr 93
4     ;                         V2.00         Sept 93
5     ;
6     ;       The purpose of this code is to set up the dsp board environm
      ent,
7     ;       variables, and memory.  This code is used as the main interf
      ace
8     ;       between the PC and the dsp board. It places all necessary
9     ;       assembler and C routines in memory and then awaits in a simp
      le
10    ;       loop, where the ARQ shell can poke the appropriate info into
11    ;       DSP memory and then run the appropriate routine.
12    ;
13
14
15
16
17                    .include VARSRCVR.ASM
18                    .global .bss
19                    .global cinit                    ;init table (from li
      nker)
20                    .global _c_int00                 ;starting address (C
       standard)
21                    .global _interleaver
22                    .global _puncture
23                    .global _combineheader
24                    .global  _conv                  ;the convolutional en
      coding
25                                                     ;routine
26                    .global  _polydiv               ;polnomial division r
      outine
27
28
29
30                    .sect    ".init"                 ;interrupt section
31    RESET           .word    _c_int00                ;RESET -> start addr
      ess
32    INT0            .word    NO                      ;all others to dummy
       reti
33    INT1            .word    RCV                     ;except the sync int
34    INT2            .word    NO
35    INT3            .word    NO
36    XINT0           .word    NO
37    RINT0           .word    NO
38    XINT1           .word    NO
39    RINT1           .word    NO
40    TINT0           .word    NO
41    TINT1           .word    NO
42    DINT            .word    NO
43
44
45    ;***************************************************************
46    ;       Data section to initially be loaded at $30000h but then
47    ;       moved to $809c00 (on chip ram).
48
49                    .data
50                    .word    5              ;constraint length
```



```
51                    .word    19h            ;polynomial1 =11001=1+x^3+x^
      4
52                    .word    17h            ;polynomial2 =10111=1+x+x^2+
      x^4
53                    .word    69665          ;CRC-CCITT
54                    .word    4374732215     ;CRC-32
55                    .word    3B1492AAh      ;flag for packet
56                    .word    0              ;Q_START --> 809c06
57                    .word    0              ;Q_START(1)
58                    .word    0              ;Q_START(2)
59                    .word    0              ;Q_START(3)
60                    .word    0              ;Q_START(4)
61                    .word    0              ;Q_START(5)
62                    .word    0              ;Q_START(6)
63                    .word    0              ;Q_START(7)
64                    .word    0              ;Q_END --> 809c0e
65                    .word    0              ;Q_END(1)
66                    .word    0              ;Q_END(2)
67                    .word    0              ;Q_END(3)
68                    .word    0              ;Q_END(4)
69                    .word    0              ;Q_END(5)
70                    .word    0              ;Q_END(6)
71                    .word    0              ;Q_END(7)
72                    .word    809c06h        ;Q_START --->809c16
73                    .word    809c0Eh        ;Q_END --->809c17
74                    .word    808042H        ;DIGITAL PORT ADDRESS --->80
      9C18
75    TAB_ENC         .WORD    5              ;pi/4 QPSK encoding table
76                    .WORD    6              ;---> 809c19
77                    .WORD    7
78                    .WORD    0
79                    .WORD    1
80                    .WORD    2
81                    .WORD    3
82                    .WORD    4
83                    .WORD    3
84                    .WORD    4
85                    .WORD    5
86                    .WORD    6
87                    .WORD    7
88                    .WORD    0
89                    .WORD    1
90                    .WORD    2
91                    .WORD    7
92                    .WORD    0
93                    .WORD    1
94                    .WORD    2
95                    .WORD    3
96                    .WORD    4
97                    .WORD    5
98                    .WORD    6
99                    .WORD    1
100                   .WORD    2
101                   .WORD    3
102                   .WORD    4
103                   .WORD    5
104                   .WORD    6
105                   .WORD    7
106                   .WORD    0
107   ICHAN           .WORD    2AA50000H      ;IBIT CHAN   1 volt
```

```
                .WORD   1E280000H       ;0.707
                .WORD   0               ;0.00
                .WORD   ~1E280000H      ;-0.707
                .WORD   -2AA50000H      ;-1.00
                .WORD   -1E280000H      ;-0.707
                .WORD   0
                .WORD   1E280000H
QCHAN           .WORD   0
                .WORD   1E280000H
                .WORD   2AA50000H
                .WORD   1E280000H
                .WORD   0
                .WORD   -1E280000H
                .WORD   -2AA50000H
                .WORD   -1E280000H
                .WORD   809C19H          ;TABENC --->
                .WORD   809C39H         ;IBIT CHAN      --->
                .WORD   809C41H          ;QBIT CHAN    --->


;***********************************************************************
******
;       Variables to be used for initialization.

STACK           .usect   ".stack",STACK_SIZE

                .text
STACK_ADDR      .word   STACK                   ;address of stack
INIT_ADDR       .word   cinit                   ;address of init tab
les
PRIMCTRL        .word   00808064h               ;primary bus control
 address
EXPCTRL         .word   00808060h               ;expansion bus contr
ol address
TIMECTL1        .word   808030H                 ;timer 1 control
SERIAL0         .word   808042h                 ;FSX/DX/CLKX port co
ntrol
TIMECTL2        .word   808020h                 ;timer 2 control
RSTCTRL         .word   601h                    ;reset value for tim
ers
PERIOD          .word   808038h                 ;timer 1 period
COUNT           .word   55                      ;period value for ti
mer 1
SETCTRL         .word   6c1h                    ;set value for timer
s
RAM1            .word   809c00h                 ;on chip ram area
DUALSTART       .word   30000h                  ;temp variables
DUALEND         .word   33300h                  ;change for 64k
DUALMEM         .word   300BFh
REALSTART       .word   31000H
REALEND         .word   32fffH
;***********************************************************************
***
;       The following code sets up the stack pointer and then
;       initializes the DSP hardware as outlined in the Technical
;       Reference Manual.
;
_c_int00:
                LDP     CODES                   ;get page of stored
```



```
address
                LDI     @STACK_ADDR, SP         ;load the address in
to SP
                LDI     SP, FP                  ;and into FP too

                LDI     @PRIMCTRL, AR0          ;Hardware specific i
nit
                LDI     INITIAL, R0
                STI     R0, *AR0
                LDI     @EXPCTRL, AR0
                LDI     NULL, R0
                STI     R0, *AR0
                ;LDI     @SERIAL0, AR0           ;SET DIGITAL OUTPUT
 TO 0
                ;LDI     2H, R0
                ;STI     R0, *AR0

;***********************************************************************
****
;       This portion of code is absolutely necessary when mixing C
;       modules with assembly language.  It ensures that the
;       variables defined in the C module are properly initialized.

                LDP     CODES                   ;get page of stored
address
                LDI     @INIT_ADDR, AR0         ;get address of init
 tables
                CMPI    -1, AR0                 ;if RAM model, skip
init
                BEQ     init_done
                LDI     *AR0++, R1              ;get first count
                BZD     init_done               ;if 0, nothing to do
                LDI     *AR0++, AR1             ;get dest address
                LDI     *AR0++, R0              ;get first word
                SUBI    1, R1                   ;count - 1

do_init:        RPTS    R1                      ;block copy
                STI     R0, *AR1++
        ||      LDI     *AR0++, R0
                LDI     R0, R1                  ;move next count int
o R1
                BNZD    do_init                 ;if there is more, r
epeat
                LDI     *AR0++, AR1             ;get next dest addre
ss
                LDI     *AR0++, R0              ;get next first word
                SUBI    1, R1                   ;count - 1

;***********************************************************************
*****
;       This code block copies all of the variables placed at $30000
h
;       and moves them to the on chip memory area at $809c00 - $80a0
00

init_done:
        LDI     @DUALSTART, AR0
        LDI     @RAM1, AR1
        LDI     *AR0++, R0              ;since parallel instruction
                                        ;coming up must initialize R
```

```
     0
205
206           RPTS    DATALENGTH
207           LDI     *AR0++, R0
208         ||STI     R0, *AR1++
209
210
211   ;*************************************************************************
      ****
212   ;
213           LDI @DUALSTART, R6              ;clear DUAL memory
214           LDI @DUALEND, R7                ;$30000 -->$33300
215           CALL CLEAR
216
217           ldf 0, r0
218           ldi @RCVD_SIGNAL_ENERGY,ar0
219           stf r0, *ar0
220
221           LDI @CPC1I, R6
222           LDI @REALEND, R7
223           CALL CLEARFLOAT
224
225           ldi @CPC2Q, r6
226           ldi r6, r7
227           addi 300h, r7
228           call CLEARFLOAT
229
230
231           LDI 0FFFH, BK                   ;set circular length of inpu
      t
232                                           ;buffers
233           OR 2H, IE                       ;enable interrupt 1
234           OR 2000H, ST                    ;enable global interrupts
235
236   BOSS:
237
238           ldp DUAL
239           ldi @STROBE_RCVR, r0
240           bnz GOAHEAD
241           br BOSS
242
243   STROBETHEHOST:
244           ldp DUAL
245           ldi 0, r0
246           sti r0, @STROBE_RCVR
247           ldi 255, r0
248           sti r0, @STROBE_HOST
249           br BOSS
250
251
252   GOAHEAD:
253           LDP CODES
254           LDI @CURRENT_FLAG, AR0          ;get latest found flag point
      er
255
256   BACKUP: LDI *AR0, R0
257           BZ STROBETHEHOST          ;check if flag found
258           LDI 0, R1
259           STI R1, *AR0                    ;if found reset pointer to z
      ero
```



```
260           LDI AR0, R1                     ;& increment flag pointer to
       next
261           ADDI 1, R1                      ;position in table.  If BOTT
      OM of
262           CMPI @TABLE_BOTTOM, R1          ;table reached reset pointer
263           BNZ NORESET                     ;to TOP
264           LDI @TABLE_TOP, R1
265   NORESET:
266           STI R1, @CURRENT_FLAG
267           ;At this point R0 contains address of real data value of the
268           ;very last dibit of FLAG
269
270           SUBI 17, R0                      ;point to first dibit of FL
      AG
271
272           cmpi @CIRC_BOTTOM, r0
273           bge NO_ADJUST
274           addi 0fffh, r0
275           ;This next section of code is used to check the validity of
      the
276           ;FLAG found.  That is, it checks if the flag found has occur
      red
277           ;in a frame of data thus resulting in a false flag.
278
279   NO_ADJUST:
280           LDI @START_FRAME, R1            ;start add of last frame dec
      oded
281           LDI @STOP_FRAME, R2             ;stop add of last frame deco
      ded
282           LDI R0, R3                      ;R3 = flag address
283
284           CMPI R1, R2                     ;STOP > START?
285           BP NO_CIRC_ADJUST               ;yes, no adjustment required
286           ldi r2, r4
287           ADDI 1000H, R2                  ;NO, adjust for circular
288           cmpi r4, r3
289           bp NO_CIRC_ADJUST
290           ADDI 1000H, R3                  ;buffer by adding circular
291                                           ;length
292
293   NO_CIRC_ADJUST:
294           ;IF ( FLAG_ADD > START_FRAME  &  FLAG_ADD < STOP_FRAME )
295           ;       FLAG IS INVALID
296           ;OTHERWISE PROCESS DATA
297
298           CMPI R1, R3                     ;FLAG - START
299           BLE VALID
300           CMPI R2, R3                     ;FLAG - STOP
301           BGE VALID
302           BR BOSS
303
304
305   ;*********************************************************
306   ;       A this point we have a valid flag and will now select
307   ;       which mode the user has chosen for the RCVR.
308
309   VALID:
310           LDP DUAL
311           LDI @MENU_OPTION, R1
312           AND 3h, R1
```

```
313           CMPI 1, R1
314           BZ MODE1
315           CMPI 2, R1
316           BZ MODE2
317           CMPI 3, R1
318           BZ MODE3
319           BR VALID
320
321
322     ;*****************************************************************
      *
323     ; MODE 1
324     ;       This mode makes the rcvr simply hard decode the data rcvd,
325     ;       place it at slot P1 0, and send an ACK.
326     ;       No decoding is done since, it is assumed that no coding
327     ;       was performed.  This mode allows the ser to check the channe
      l
328     ;       conditions of the system with no coding.
329     ;
330     ;       Requires:
331     ;               R0 = real data flag start
332     ;       Modifies:
333     ;               R0, R1, R2, R3, R4, R5, R6, R7
334     ;               AR0, AR1, AR2
335     ;       Returns:
336     ;               Nothing
337     ;
338     MODE1:
339           LDP DUAL                        ;insure ACK0 is clear
340           LDI 0, R7
341           STI R7, @ACK0
342           LDP CODES
343           LDI @FLAG0P1, R6                ;clear slot 0 area of P1
344           LDI @PACKET1HARDP1, R7
345           CALL CLEAR
346
347           sti r0, @CURRENT_START
348
349
350
351           LDI 16, R1                      ;hard decode flag
352           LDI @FLAG0P1, AR2               ;slot to store decoded data
353           CALL HARDDECODE_CHUNK           ;decode chunk of length = R1
354                                           ;and place starting at AR2
355
356           LDI @CURRENT_START, R0
357           ADDI 10H, R0                    ;transform DQPSK data to QPS
      K data
358           cmpi @CIRC_TOP, r0
359           BLT NO_ADJ1
360           SUBI 0FFFh, R0
361     NO_ADJ1:
362           CALL DQPSK_DEINT
363
364           LDI @BUFP1, AR0                   ;Zk value
365           LDI @BUFP2, AR1                   ;Wk value
366
367           LDI 512, R1                     ;hard decode flag
368           LDI @PACKET0HARDP1, AR2               ;slot to store decoded
       data
```



```
369           CALL QPSK1                              ;decode chunk of length =
       R1
370                                                   ;and place starting at AR2
371
372
373
374
375
376
377           ;Refresh start and stop frame pointers used in flag validati
      on
378           LDI @CURRENT_START, R0
379           STI R0, @START_FRAME
380
381           ADDI 210H, R0
382           LDI @CIRC_TOP, R1
383           CMPI R1, R0
384           BLT NO_SUB1
385           SUBI 1000H, R0
386
387     NO_SUB1:STI R0, @STOP_FRAME
388
389           LDI 1, R1               ;send an ACK0 to HOST PROGRAM
390           LDP DUAL
391           STI R1, @ACK0
392           LDP CODES
393           BR BOSS
394
395     ;*****************************************************************
      *****
396     ;
397     ; MODE VITERBI DECODE CHANNEL
398     ;       This mode assumes that the frame rcvd contains a flag with 4
      96 bits
399     ;       convolved with a rate 1/2 code given by polynomials in VARS.
      ASM
400     ;       and constraint length K=5.
401     ;       The frame     32 bits --> flag
402     ;                    992 bits --> convolved data
403     ;                   ------------
404     ;                    1024 bit frame
405
406     MODE2:
407           LDP DUAL                                ;insure ACK0 is clear
408           LDI 0, R7
409           STI R7, @ACK0
410           LDP CODES
411           LDI @FLAG0P1, R6                        ;clear slot 0 area of P1
412           LDI @PACKET1HARDP1, R7
413           CALL CLEAR
414
415           sti r0, @CURRENT_START
416
417
418
419           LDI 16, R1                              ;hard decode flag
420           LDI @FLAG0P1, AR2                       ;slot to store decoded data
421           CALL HARDDECODE_CHUNK                   ;decode chunk of length = R1
422                                                   ;and place starting at AR2
423
```

```
424
425             LDI @CURRENT_START, R0
426             ADDI 10H, R0                    ;transform DQPSK data to QPS
    K data
427             cmpi @CIRC_TOP, r0
428             BLT NO_ADJ2
429             SUBI 0FFFh, R0
430     NO_ADJ2:
431             CALL DQPSK_DEINT
432
433             LDI @BUFP1, AR0                   ;Zk value
434             LDI @BUFP2, AR1                   ;Wk value
435
436             LDI 512, R1
437             LDI @FLAG0P1, AR2
438             ADDI 1, AR2
439             LDI 7, R0
440             STI R0, @ADDER_ONE_PUNC
441             STI R0, @ADDER_TWO_PUNC
442
443             CALL START_VITB
444
445
446
447
448
449             ;Refresh start and stop frame pointers used in flag validati
    on
450             LDI @CURRENT_START, R0
451             STI R0, @START_FRAME
452
453             ADDI 210H, R0
454             LDI @CIRC_TOP, R1
455             CMPI R1, R0
456             BLT NO_SUB2
457             SUBI 1000H, R0
458
459     NO_SUB2:STI R0, @STOP_FRAME
460
461             LDI 1, R1               ;send an ACK0 to HOST PROGRAM
462             LDP DUAL
463             STI R1, @ACK0
464             LDP CODES
465             BR BOSS
466
467
468             LDP DUAL                        ;insure ACK0 is clear
469             LDI 0, R7
470             STI R7, @ACK0
471             LDP CODES
472             LDI @FLAG0P1, R6                ;clear slot 0 area of P1
473             LDI @PACKET1HARDP1, R7
474             CALL CLEAR
475
476             STI R0, @CURRENT_START
477             LDI 496, R1                     ;496 dibits = 992 bits
478
479             ;CALL DATA_CHUNK_FULL            ;insure all 496 dibits rcvd
480                                             ;before any further processi
    ng
```



```
481
482             LDI 16, R1                      ;hard decode flag
483             LDI @FLAG0P1, AR2
484
485             CALL HARDDECODE_CHUNK
486
487             LDI @CURRENT_START, R0
488             ADDI 10H, R0                    ;transform DQPSK data to QPS
    K data
489             cmpi @CIRC_TOP, r0
490             BLT NO_ADJ
491             SUBI 0FFFh, R0
492
493
494     ;NO_ADJ: LDI R0, AR0                     ;real I values
495             LDI R0, AR1
496             ADDI 1000H, AR1                 ;real Q values
497
498             LDF *AR0++(1)%, R2
499             STF R2, @OLDI                   ;initialize OLDI & OLDQ
500             LDF *AR1++(1)%, R2
501             STF R2, @OLDQ
502
503             LDI @BUFP1, AR3                   ;I
504             LDI @BUFP2, AR4                   ;Q
505             LDI 496, R4                       ;length to decode
506
507     ;MORE:
508             LDF @OLDI, R0
509             LDF @OLDQ, R1
510             LDF *AR0++(1)%, R2
511             LDF *AR1++(1)%, R3
512             PUSH R4
513             CALL DIFFERENTIAL_PHASE_DECODING
514             POP R4
515             STF R6, *AR4++(1)%                    ;Wk LSB
516             STF R7, *AR3++(1)%                    ;Zk MSB
517                                                   ;branch symbol ZkWk
518                                                   ;or IQ
519             STF R2, @OLDI
520             STF R3, @OLDQ
521             SUBI 1, R4
522             BP MORE
523             nop
524             LDI @BUFP1, AR0                   ;Zk value
525             LDI @BUFP2, AR1                   ;Wk value
526             LDI 64, R1
527             LDI @FLAG0P1, AR2
528             ADDI 1, AR2
529             CALL START_VITB
530
531             ;Refresh start and stop frame pointers used in flag validati
    on
532             LDI @CURRENT_START, R0
533             STI R0, @START_FRAME
534             addi 200h, r0
535             cmpi 3200h, r0
536             blt NO_SUB
537             subi 1000h, r0
538     ;NO_SUB: sti r0, @STOP_FRAME
```

```
539           ;SUBI 1, AR2
540           ;LDI AR2, R6
541           ;STI R6, @STOP_FRAME
542
543           LDI 1, R1               ;send an ACK0 to HOST PROGRAM
544           LDP DUAL
545           STI R1, @ACK0
546           LDP CODES
547
548
549
550           BR BOSS
551
552     ;*************************************************************
      *****
553     ;
554     ; MODE ADAPTIVE CPC DECODING
555     ;       This mode allows the receiver to decode a rate 1/2 header
556     ;       either a rate 1, 3/4 or 1/2 data packet.
557
558
559
560     MODE3:
561           LDP DUAL                          ;insure ACK0 is clear
562           LDI 8888, R7
563           STI R7, @ACK0
564           LDP CODES
565           LDI @FLAG0P1, R6                  ;clear slot 0 area of P1
566           LDI @PACKET1HARDP1, R7
567           CALL CLEAR
568
569           STI R0, @CURRENT_START
570
571           LDI 16, R1                        ;hard decode flag
572           LDI @FLAG0P1, AR2
573
574           CALL HARDDECODE_CHUNK
575
576
577           LDI @CURRENT_START, R0
578           ADDI 10H, R0                      ;transform DQPSK data to QPS
      K data
579           cmpi @CIRC_TOP, r0
580           BLT NO_ADJ
581           SUBI 0FFFh, R0
582     NO_ADJ:
583           CALL DQPSK_DEINT
584
585           LDI @BUFP1, AR0                     ;Zk value
586           LDI @BUFP2, AR1                     ;Wk value
587           LDI 64, R1
588           LDI @FLAG0P1, AR2
589           ADDI 1, AR2
590           LDI 3fh, R0
591           STI R0, @ADDER_ONE_PUNC
592           STI R0, @ADDER_TWO_PUNC
593
594           CALL START_VITB
595
596           ;HEADER SOFT DECODED AND PLACED @ 300C1--300C2
```



```
597
598           LDP CODES
599           LDI @PACKET0HARDP1, AR1
600           LDI *AR1++, R0
601           LDI *AR1, R1
602           LDI R1, R4
603           LDI @CRC_MASK, R2
604           AND R2, R4
605           LSH -5, R4                                ;R4 = CRC
606           LDI 31, R2
607           AND R2, R1                          ;mask out TAIL & CRC from he
      ader
608
609           LDI @HEADBUF1, AR5
610           LDI @HIGH_MASK, R2
611           LSH 16, R1
612           AND R0, R2, R3
613           LSH -16, R3
614           OR R1, R3
615           LSH 16, R0
616           STI R0, *AR5++
617           STI R3, *AR5--
618
619
620
621           LDP CODES
622           LDI 17, R0
623           LDI @CRC_CCITT, R1
624           LDI 2, R2
625           LDI @HEADBUF2, R3
626
627
628           PUSH R3
629           PUSH R2
630           PUSH R1
631           PUSH AR5
632           PUSH R0
633           CALL _polydiv
634           SUBI 5, SP
635
636           CMPI R0, R4
637           LDINZ 9999, R1
638           BNZ NACK                  ;if CRC fails send NACK
639                                     ;otherwise strip info from header
640                                     ;and decode data packet
641
642           ldi @PACKET0HARDP1, AR0
643           LDI *++AR0, R0            ;get 2nd word of header
644           ldi r0, r1
645           AND 3, R0
646           and 12, r1
647           LDIZ 9999, R1
648           BZ NACK                   ;if rate is wrong header is NFG
649           LDP DUAL
650           lsh -2, r1
651           STI R1, @RATE             ;get rate of data packet
652           sti r0, @CODE             ;get CPCi code to use i=1, 2
653
654           LDP CODES                 ; 1 - CPC1
655                                     ; 2 - CPC2
```

```
656
657
658
659           LDI @BUFP1, AR0
660           LDI @BUFP2, AR1
661
662           ADDI 64, AR0
663           ADDI 64, AR1
664
665           CMPI 1, R1              ;rate 1
666           BZ RATE1
667           CMPI 2, R1              ;rate 3/4
668           BZ RATE75
669           CMPI 3, R1              ;rate 1/2
670           BZ RATE50
671           LDI 9999, R1
672           BR NACK
673
674   RATE1:
675           LDI 896, R3
676           LDI 27, R4
677
678           CMPI 1, R0
679           LDIZ 15H, R1
680           LDIZ 2AH, R2
681           LDIZ 100, R7
682
683           CMPI 2, R0
684           LDIZ 2AH, R1
685           LDIZ 15H, R2
686           LDIZ 200, R7
687           BR DECODE
688   RATE75:
689           LDI 672, R3
690           LDI 20, R4
691
692           CMPI 1, R0
693           LDIZ 2dh, R1
694           LDIZ 1bh, R2
695           LDIZ 100, R7
696
697           CMPI 2, R0
698           LDIZ 36h, R1
699           LDIZ 2dh, R2
700           LDIZ 200, R7
701           BR DECODE
702   RATE50:
703           LDI 448, R3
704           LDI 13, R4
705
706           CMPI 1, R0
707           LDIZ 3fh, R1
708           LDIZ 3fh, R2
709           LDIZ 100, R7
710
711           CMPI 2, R0
712           LDIZ 3fh, R1
713           LDIZ 3fh, R2
714           LDIZ 200, R7
715           BR DECODE
```



```
716
717   DECODE:
718           LDI @PACKET0HARDP1, AR2
719           ADDI 3, AR2             ;leave blank word for CRC calc
720
721
722
723           STI R1, @ADDER_ONE_PUNC
724           STI R2, @ADDER_TWO_PUNC
725           LDI R3, R1              ;BIT LENGTH TO DECODE
726           PUSH R7
727           PUSH R4
728           CALL START_VITB
729
730           ;CRC calculation for data packet
731
732           LDI @PACKET0HARDP1, AR3
733           ADDI 2, AR3
734           LDI 33, R0
735           LDI @CRC_32, R1
736           LDI @DATABUFP1, R3
737           POP R4                            ;# OF DATA WORDS
738           LDI R4, R2
739           PUSH R4
740
741           PUSH R3
742           PUSH R2
743           PUSH R1
744           PUSH AR3
745           PUSH R0
746
747           CALL _polydiv
748
749           SUBI 5, SP
750           LDI @PACKET0HARDP1, AR0
751           POP R4
752           ADDI R4, AR0
753           ADDI 2, AR0
754           LDI *AR0, R1
755           POP R7
756           CMPI R0, R1
757           LDIZ R7, R1
758           BZ ACK
759
760           LDP DUAL
761           LDI @CODE, R0
762
763           LDP CODES
764           LDI @SEQUENCES, R1
765
766
767           CMPI 1, R0                        ;if rate CPC1 load pointers
768           LDIZ @CPC1I, AR0                  ;and update sequence count
769           LDIZ @CPC1Q, AR1
770           LDIZ 1, R2
771
772           CMPI 2, R0                        ;if rate CPC2 load pointers
773           LDIZ @CPC2I, AR0                  ;and update sequence count
774           LDIZ @CPC2Q, AR1
775           LDIZ 2, R2
```

```
776
777           OR R2, R1
778           STI R1, @SEQUENCES              ;write sequence count
779
780           ;save CPCx code to appropriate slot for possible combination
781
782           LDI @BUFP1, AR2
783           LDI @BUFP2, AR3
784           ADDI 64, AR2                    ;point to real values of dat
      a packet
785           ADDI 64, AR3
786           LDF *AR2++, R0                  ;preload registers for copyi
      ng
787           LDF *AR3++, R1
788           LDI 1ffh, RC                    ;copy 512 WORDS
789           RPTB COPY_SEQUENCE
790
791           ;ldf *ar2++, r0
792           ;ldf *ar0, r1
793           ;addf r0, r1
794           ;stf r1, *ar0++
795
796           ;ldf *ar3++, r0
797           ;ldf *ar1, r1
798           ;addf r0, r1
799     ;COPY_SEQUENCE:        stf r1, *ar1++
800
801           ;The above commented out instructions are used in Code Combi
      ning
802
803           LDF *AR2++, R0
804          ||STF R0, *AR0++
805     COPY_SEQUENCE:
806          LDF *AR3++, R1
807          ||STF R1, *AR1++
808
809
810           ;currently rcvd data packet placed in CPC1 or CPC2 slot
811
812           LDI @SEQUENCES, R0
813
814           CMPI 3, R0
815           LDINZ 6666, R1                  ;do we have at least two seq
      uences
816           BNZ NACK                        ;to combine
817                                           ;NO - send NACK
818                                           ;YES - combine sequences and
       decode
819           ;combine CPC1 and CPC2
820           LDI 32, R0                       ;initialize all necessary p
      ointers
821           STI R0, @PUNC_COLUMN            ;and variables for combining
822
823           LDI 1bh, R0                       ;CPC1 and CPC2 I and Q val
      ues
824           LDI 2dh, R1
825           LDI 36h, R2
826           LDI 15h, R3
827           LDI 2ah, R4
828           ldi 3fh, R5
```



```
829
830           LDP DUAL
831           LDI @RATE, R7
832           ldp CODES
833           CMPI 1, R7
834           BZ RATE1A
835           CMPI 2, R7
836           BZ RATE75A
837           CMPI 3, R7
838           BZ RATE50A
839
840     RATE1A:
841           STI R3, @ADDER_ONE_PUNC
842           STI R4, @ADDER_TWO_PUNC
843           STI R4, @ADDER_1PRIME_PUNC
844           STI R3, @ADDER_2PRIME_PUNC
845           LDI 896, RC
846           LDI 27, R2
847           BR CONTINUE
848     RATE75A:
849           STI R1, @ADDER_ONE_PUNC
850           STI R0, @ADDER_TWO_PUNC
851           STI R2, @ADDER_1PRIME_PUNC
852           STI R1, @ADDER_2PRIME_PUNC
853           LDI 672, RC
854           LDI 20, R2
855           BR CONTINUE
856     RATE50A:
857           STI R5, @ADDER_ONE_PUNC
858           STI R5, @ADDER_TWO_PUNC
859           STI R5, @ADDER_1PRIME_PUNC
860           STI R5, @ADDER_2PRIME_PUNC
861           LDI 448, RC
862           LDI 13, R2
863           BR CONTINUE
864     CONTINUE:
865           PUSH R2
866           LDI @CPC1I, AR0
867           LDI @CPC1Q, AR1
868           LDI @CPC2I, AR2
869           LDI @CPC2Q, AR3
870           LDI @BUFP1, AR4
871           LDI @BUFP2, AR5
872                                   ;combine 512 words for each I and Q
873           RPTB COMBINE
874
875           LDI @PUNC_COLUMN, R1
876           MPYI 2, R1
877           CMPI 64, R1
878           LDIZ 1, R1
879           sti r1, @PUNC_COLUMN
880
881           TSTB @ADDER_ONE_PUNC, R1
882           LDFZ 0, R2
883           BZ OVER1
884           LDF *AR0++, R2                  ;R2 = I1
885
886     OVER1:  TSTB @ADDER_1PRIME_PUNC, R1
887           LDFZ 0, R3
888           BZ OVER2
```

```
889             LDF *AR2++, R3                  ;R3 = I2
890
891     OVER2:  TSTB @ADDER_TWO_PUNC, R1
892             LDFZ 0, R4
893             BZ OVER3
894             LDF *AR1++, R4                  ;R4 = Q1
895
896     OVER3:  TSTB @ADDER_2PRIME_PUNC, R1
897             LDFZ 0, R5
898             BZ OVER4
899             LDF *AR3++, R5                  ;R5 = Q2
900
901
902     OVER4:
903             ADDF R2, R3
904             ADDF R4, R5
905
906             STF R3, *AR4++
907     COMBINE:STF R5, *AR5++
908
909             ;clear memory at PACKET0HARDP1
910             LDI @PACKET0HARDP1, R6
911             ADDI 3, R6
912             LDI @PACKET1HARDP1, R7
913             CALL CLEAR
914
915             ;viterbi decode rate 1/2 combined sequences
916
917             ldp CODES
918             ldi @BUFP1, ar0
919             ldi @BUFP2, AR1
920             ldi @PACKET0HARDP1, ar2
921             addi 3, ar2
922
923
924             LDI 3fh, R0
925             STI R0, @ADDER_ONE_PUNC
926             STI R0, @ADDER_TWO_PUNC
927
928             LDP DUAL
929             LDI @RATE, R7
930             ldp CODES
931             CMPI 1, R7
932             ldiz 896, r1
933             CMPI 2, R7
934             ldiz 672, r1
935             CMPI 3, R7
936             ldiz 448, r1
937
938
939             CALL START_VITB
940
941             ;check CRC again
942
943
944             ;CRC calculation for data packet
945
946             LDI @PACKET0HARDP1, AR3
947             ADDI 2, AR3
948             LDI 33, R0
```



```
949             LDI @CRC_32, R1
950             LDI @DATABUFP1, R3
951             POP R2
952             PUSH R2
953
954             PUSH R3
955             PUSH R2
956             PUSH R1
957             PUSH AR3
958             PUSH R0
959
960             CALL _polydiv
961
962             SUBI 5, SP
963             LDI @PACKET0HARDP1, AR0
964             POP R2
965             ADDI R2, AR0
966             ADDI 2, AR0
967             LDI *AR0, R1
968             and @MASK1, r0
969             and @MASK1, r1
970
971             CMPI R0, R1
972             BZ CLEAN_COMBINE
973             LDI 6666, R1
974             BR NACK
975
976     CLEAN_COMBINE:
977             LDP CODES
978             LDI 0, R1
979             STI R1, @SEQUENCES
980             LDI 300, R1
981             BR ACK
982
983
984             ;Refresh start and stop frame pointers used in flag validati
        on
985     ACK:
986             ;LDI 1, R1
987             LDP DUAL
988             STI R1, @ACK0
989             LDP CODES
990             ldi @BUFP1, r6
991             ldi r6, r7
992             addi 0effh, r7
993             call CLEARFLOAT
994
995             ldi @CPC2Q, r6
996             ldi r6, r7
997             addi 300h, r7
998             call CLEARFLOAT
999
1000    CLEANER:
1001            LDI @CURRENT_START, R0
1002            STI R0, @START_FRAME
1003            addi 210h, r0
1004            ldi @CIRC_TOP, r1
1005            cmpi r1, r0
1006            blt NO_SUB
1007            subi 1000h, r0
```

```
1008  NO_SUB: sti r0, @STOP_FRAME
1009
1010
1011          BR BOSS
1012
1013  NACK:
1014          ;LDI 9999, R1               ;send an NACK0 to HOST PROGRAM
1015          LDP DUAL
1016          STI R1, @ACK0
1017          LDP CODES
1018          cmpi 6666, r1
1019          bz CLEANER
1020          ;BR POLIZIA
1021          BR BOSS
1022
1023
1024  ;*************************************************************************
      *
1025  ;       This section of code is responsible for hard decoding a chun
      k
1026  ;       of data given:
1027  ;       Requires:
1028  ;               R0  = start of real data I channel
1029  ;               R1  = length to decode in Dibits
1030  ;               AR2 = where to place hard data
1031  ;       Modifies:
1032  ;               R1, R2, R3, R4, R5, R6, R7
1033  ;               AR0, AR1, AR2
1034  ;       Returns:
1035  ;                Nothing
1036
1037  HARDDECODE_CHUNK:
1038          LDF 0, R2
1039          STF R2, @OLDI                   ;initialize OLDI & OLDQ
1040          STF R2, @OLDQ
1041
1042          LDI R0, AR7                     ;store flag start for future
       ref
1043          LDI R0, AR0                     ;AR0 = I channel pointer
1044          LDI R0, AR1                     ;add offset to get Q channel
       ptr
1045          ADDI 1000H, AR1                 ;AR1 = Q channel pointer
1046          LDI 16, R4                      ;R4 = dibit count
1047
1048          ldf *ar0++(1)%, r2
1049          stf r2, @OLDI
1050          ldf *ar1++(1)%, r2
1051          stf r2, @OLDQ
1052
1053  MORE_DIBITS:
1054
1055
1056          ldi @TROUBLE2, ar3
1057          LDI AR0, R2
1058          LDI AR1, R3
1059          sti r2, *ar3++
1060          sti r3, *ar3++
1061          ldf *ar0, r2
1062          ldf *ar1, r3
1063          stf r2, *ar3++
```



```
1064          stf r3, *ar3++
1065
1066
1067          LDF *AR0++(1)%, R2                ;load real data I
1068        ||LDF *AR1++(1)%, R3                ;and Q
1069          ;ldi @TROUBLE2, ar3
1070          stf r2, *ar3++
1071          stf r3, *ar3++
1072          ldf @OLDI, r5
1073          stf r5, *ar3++
1074          ldf @OLDQ, r5
1075          stf r5, *ar3
1076
1077          PUSH R1
1078          PUSH R4
1079          ldf @OLDI, r0
1080          ldf @OLDQ, r1
1081          CALL HARDDECODE                   ;hard decode it
1082
1083          POP R4
1084          POP R1
1085          STF R2, @OLDI
1086          STF R3, @OLDQ                     ;refresh OLDI & OLDQ
1087
1088          LDI *AR2, R2                      ;place dibit @ current locat
      ion
1089          LSH -2, R2
1090          LSH 30, R0
1091          OR R0, R2
1092          STI R2, *AR2
1093
1094          SUBI 1, R4                        ;decrement dibit count
1095          BNZ NO_MEM_INC
1096          ADDI 1, AR2                       ;increment memory pointer
1097          LDI 16, R4                         ;reset dibit count
1098
1099  NO_MEM_INC:
1100          SUBI 1, R1                        ;length = 0 ?
1101          BNZ MORE_DIBITS                   ;no, branch back
1102          RETS                              ;yes, return
1103
1104  ;*************************************************************************
      *
1105  ;
1106  ;       This section of code checks to insure that the necessary dat
      a
1107  ;       chunk has ALL been rcvd (real data) before any further proce
      ssing
1108  ;       done
1109  ;       Requires:
1110  ;               R0 = start of flag pointer
1111  ;               R1 = # of dibits that chunk consists of
1112  ;       Modifies:
1113  ;               R1, R2
1114  ;       Returns:
1115  ;                Nothing
1116
1117
1118  DATA_CHUNK_FULL:
1119          LDI R0, R2
```

```
1120            ;ADDI 16, R2                    ;R2 = start of data chunk
1121
1122            ADDI R1, R2                     ;R2 now contains address poi
    nter
1123                                            ;for last dibit
1124    GO:
1125            ;cmpi @CIRC_TOP, r2
1126            cmpi @CIRCLESS1, r2
1127            blt WAIT2
1128            subi 0FFFh, r2
1129    WAIT2:
1130            ldi @REAL_IBIT_POINTER, r1
1131            cmpi r2, r1
1132            bnz WAIT2
1133
1134
1135
1136    WAIT:
1137            ;LDI @REAL_IBIT_POINTER, R1
1138            ;CMPI R2, R1                    ;reached last dibit ?
1139            ;BLT WAIT                       ;no, then wait
1140    GETOUT:
1141            ldi @TROUBLE3, ar0              ;debug code to check when l
    eft loop
1142            sti r1, *ar0++
1143            ldi r1, ar1
1144            ldf *ar1, r1
1145            sti r1, *ar0
1146            RETS                            ;yes, then return
1147
1148
1149
1150    ;*****************************************************************
    ***
1151    ;       This section of code is responsible for decoding the differe
    ntial
1152    ;       phase from the real data.  It operates on the real data from
1153    ;       the I and Q channel and outputs real data which is no longer
1154    ;       dependent on the previous real data.  That is it transforms
1155    ;       pi/4 - DQPSK real data to QPSK real data for hard decoding.
1156    ;       The mapping of this transform is given by Wk and Zk.
1157    ;
1158    ;       Requires
1159    ;               R0 = oldI
1160    ;               R1 = oldQ
1161    ;               R2 = newI
1162    ;               R3 = newQ
1163    ;       Modifies:
1164    ;               R4, R5, R6, R7
1165    ;       Returns:
1166    ;               R6 = Wk = oldI * newI + oldQ * newQ
1167    ;               R7 = Zk = oldI * newQ - oldQ * newI
1168    ;
1169
1170    DIFFERENTIAL_PHASE_DECODING:
1171            MPYF3 R0, R2, R4                ;R4 = oldI * newI
1172            MPYF3 R1, R3, R5                ;R5 = oldQ * newQ
1173            ADDF R4, R5, R6                 ;R6 = R4 + R5
1174
1175            MPYF3 R1, R2, R4                ;R4 = oldQ * newI
```



```
1176            MPYF3 R0, R3, R5                ;R5 = oldI * newQ
1177            SUBF R4, R5, R7                 ;R7 = R5 - R4
1178
1179            ;At this point R6 = Wk and R7 = Zk
1180
1181            RETS
1182
1183    ;*****************************************************************
    ***
1184    ;       This section of code is responsible for the actual hard deco
    ding
1185    ;       It references Wk and Zk and uses these real values to decide
1186    ;       which dibit was sent.
1187    ;
1188    ;       Requires
1189    ;               R0 = oldI
1190    ;               R1 = oldQ
1191    ;               R2 = newI
1192    ;               R3 = newQ
1193    ;               R6 = Wk
1194    ;               R7 = Zk
1195    ;       Modifies:
1196    ;               R0, R4, R5, R6, R7
1197    ;       Returns:
1198    ;               R0 = dibit received
1199    ;
1200    HARDDECODE:
1201            CALL DIFFERENTIAL_PHASE_DECODING
1202
1203            ;if (Wk > 0 & Zk > 0)
1204            CMPF 0, R6
1205            BLE L1
1206            CMPF 0, R7
1207            BLE L1
1208            ; dibit is decoded as 3
1209            LDI 3, R0
1210            RETS
1211
1212    L1:     ;else if (Wk > 0 & Zk < 0)
1213            CMPF 0, R6
1214            BLE L2
1215            CMPF 0, R7
1216            BGE L2
1217            ;dibit is decoded as a 2
1218            LDI 2, R0
1219            RETS
1220
1221    L2:     ;else if (Wk < 0 & Zk > 0)
1222            CMPF 0, R6
1223            BGE L3
1224            CMPF 0, R7
1225            BLE L3
1226            ;dibit is decoded as a 1
1227            LDI 1, R0
1228            RETS
1229
1230    L3:     ;Otherwise dibit is decoded as a 0
1231            LDI 0, R0
1232            RETS
1233    ;*****************************************************************
```

```
     ***
1234 ;       This section of code uses a shift register which is shifted
     and ORed
1235 ;       with the most recent decoded dibit.  This register is then
1236 ;       compared to the flag and if it matches the address which poi
     nts
1237 ;       to the occurrence of this flag (in real data) is saved in
1238 ;       the FLAG_ADDRESS_TABLE.
1239 ;
1240 ;       Requires:
1241 ;               R0 = contains most recent decoded dibit
1242 ;       Modifies:
1243 ;               R0, R1, AR2
1244 ;       Returns:
1245 ;               Nothing
1246
1247 FLAG_CHECKER:
1248         LDI @FLAG_TO_BE, R1             ;load current decoded word
1249         LSH -2, R1                      ;shift word right by 2
1250         LSH 30, R0                      ;shift dibit left by 30
1251         OR R0, R1                       ;OR dibit with current word
1252         STI R1, @FLAG_TO_BE
1253         lsh -8, R1
1254
1255         CMPI @FLAG_COMP, R1             ;compare word to flag patter
     n
1256         ;BNZ ENDS
1257         BNZ     CORRELATE               ;if no match, then return
1258
1259 FLAG_FOUND:
1260         LDI @FLAG_ADDRESS_TABLE, AR2    ;yes, flag found
1261
1262         LDI @REAL_IBIT_POINTER, R0      ;get address location of fla
     g end
1263         STI R0, *AR2++                  ;store in FLAG_ADDRESS_TABLE
1264         LDI AR2, R0
1265         CMPI @TABLE_BOTTOM, R0          ;if FLAG_ADDRESS_TABLE reach
     ed
1266         BNZ UPDATE_TABLE                ;bottom of buffer
1267         LDI @TABLE_TOP, R0              ;reset to top of buffer
1268 UPDATE_TABLE:
1269         STI R0, @FLAG_ADDRESS_TABLE
1270
1271
1272 ENDS:   RETS
1273
1274 CORRELATE:
1275         LDI @FLAG_COMP, R2
1276         XOR R1, R2, R3                  ;R3 = negative 1's
1277         NOT R3, R4                      ;R4 = positive 1's
1278         LDI 0, R5                       ;R5 = number of negative 1 b
     its
1279         LDI 0, R6                       ;R6 = number of positive 1 b
     its
1280         LDI 24, R7                      ;correlate for 24 bit length
1281 TEST:
1282         TSTB 1, R3
1283         BZ LOOK_POS
1284         ADDI 1, R5
1285 LOOK_POS:
```



```
1286         TSTB 1, R4
1287         BZ OVER
1288         ADDI 1, R6
1289 OVER:
1290         LSH -1, R3
1291         LSH -1, R4
1292         SUBI 1, R7
1293         BP TEST
1294
1295         SUBI R5, R6
1296         CMPI @THRESHOLD, R6
1297         BGE FLAG_FOUND
1298         BR ENDS
1299 ;*************************************************************
     *
1300 ;       This section of code is responsible for hard decoding a chun
     k
1301 ;       data given:
1302 ;       Requires:
1303 ;               AR0  = start of MSB data Zk Q
1304 ;               AR1  = start of LSB data Wk I
1305 ;               R1  = length to decode in Dibits
1306 ;               AR2 = where to place hard data
1307 ;       Modifies:
1308 ;               R1, R2, R3, R4, R5, R6, R7
1309 ;               AR0, AR1, AR2
1310 ;       Returns:
1311 ;                Nothing
1312
1313 QPSK1:
1314         LDI 16, R4                      ;R4 = dibit count
1315 MORE_DIBITS1:
1316         LDF *AR0++, R7                  ;load real data MSB
1317       ||LDF *AR1++, R6                  ;and LSB
1318
1319         PUSH R1
1320         PUSH R4
1321         CALL QPSK2                      ;hard decode it
1322
1323         POP R4
1324         POP R1
1325
1326         LDI *AR2, R2                    ;place dibit @ current locat
     ion
1327         LSH -2, R2
1328         LSH 30, R0
1329         OR R0, R2
1330         STI R2, *AR2
1331
1332         SUBI 1, R4                      ;decrement dibit count
1333         BNZ NO_MEM_INC1
1334         ADDI 1, AR2                     ;increment memory pointer
1335         LDI 16, R4                       ;reset dibit count
1336
1337 NO_MEM_INC1:
1338         SUBI 1, R1                      ;length = 0 ?
1339         BNZ MORE_DIBITS1                 ;no, branch back
1340         RETS                            ;yes, return
1341
1342 ;*************************************************************
```

```
     ***
1343 ;       This section of code is responsible for the actual hard deco
     ding
1344 ;       It references Wk and Zk and uses these real values to decide
1345 ;       which dibit was sent.
1346 ;
1347 ;       Requires
1348 ;               R6 = Wk
1349 ;               R7 = Zk
1350 ;       Modifies:
1351 ;               R6, R7
1352 ;       Returns:
1353 ;               R0 = dibit received
1354 ;
1355 QPSK2:
1356         ;if (Wk > 0 & Zk > 0)
1357         CMPF 0, R6
1358         BLE L11
1359         CMPF 0, R7
1360         BLE L11
1361         ; dibit is decoded as 3
1362         LDI 3, R0
1363         RETS
1364
1365 L11:     ;else if (Wk > 0 & Zk < 0)
1366         CMPF 0, R6
1367         BLE L12
1368         CMPF 0, R7
1369         BGE L12
1370         ;dibit is decoded as a 2
1371         LDI 2, R0
1372         RETS
1373
1374 L12:     ;else if (Wk < 0 & Zk > 0)
1375         CMPF 0, R6
1376         BGE L13
1377         CMPF 0, R7
1378         BLE L13
1379         ;dibit is decoded as a 1
1380         LDI 1, R0
1381         RETS
1382
1383 L13:     ;Otherwise dibit is decoded as a 0
1384         LDI 0, R0
1385         RETS
1386 ;*************************************************************************
     **
1387 ;
1388 ;       This code transforms real DQPSK data to real QPSK data and
1389 ;       then deinterleaves the data placing it @BUFP1 and @BUFP2
1390 ;       for either hard decoding by QPSK or soft decoding.
1391 ;
1392 ;
1393 ;       Requires:
1394 ;               R0 = start address of real I data in receiver buffer
1395 ;
1396 ;       Returns:
1397 ;               BUFP1 contains Zk real values MSB
1398 ;               BUFP2 contains Wk real values LSB
1399 ;
```



```
1400 ;
1401 ;       Deinterleaver currently set for 256 bit blocks
1402 ;       which is 8 ROWS by 16 COLUMNS of SYMBOLS
1403 ;       can handle up to 256 bit block if a larger
1404 ;       size is required the DSP board requires more memory
1405 ;       to handle the operation and FREE1 and FREE2 should be
1406 ;       changed to reflect the increase in memory as well as
1407 ;       the MAP.CMD file used for compiling and linking.
1408
1409 DQPSK_DEINT:
1410         LDI R0, AR0                     ;real I values
1411         LDI R0, AR1
1412         ADDI 1000H, AR1                 ;real Q values
1413
1414         LDF *AR0++(1)%, R2
1415         STF R2, @OLDI                   ;initialize OLDI & OLDQ
1416         LDF *AR1++(1)%, R2
1417         STF R2, @OLDQ
1418
1419         LDI @BUFP1, AR3                    ;I
1420         LDI @BUFP2, AR4                    ;Q
1421         ldi 512, r4
1422         ;LDI 496, R4                        ;length to decode
1423
1424 MORE:
1425         LDF @OLDI, R0
1426         LDF @OLDQ, R1
1427         LDF *AR0++(1)%, R2
1428         LDF *AR1++(1)%, R3
1429         PUSH R4
1430         CALL DIFFERENTIAL_PHASE_DECODING
1431         POP R4
1432         STF R6, *AR4++(1)%                      ;Wk LSB
1433         STF R7, *AR3++(1)%                      ;Zk MSB
1434                                                 ;branch symbol ZkWk
1435                                                 ;or IQ
1436         STF R2, @OLDI
1437         STF R3, @OLDQ
1438         SUBI 1, R4
1439         BP MORE
1440
1441
1442         ;AT THIS POINT Zk VALUES ARE AT @BUFP1 AND Wk AT @BUFP2
1443
1444         ;DEINTERLEAVE THE DATA
1445
1446         LDI @FREE1, R6
1447         LDI R6, R7
1448         ADDI 600H, R7
1449         CALL CLEARFLOAT
1450
1451
1452         LDI @BUFP1, AR0
1453         LDI @BUFP2, AR1
1454         LDI @FREE1, AR2
1455         LDI @FREE2, AR3
1456
1457         LDI 0, R0                       ;R0 is block count <= 8
1458 KEEP_DEINT:
1459         LDI 0, R1                       ;R1 = ROW = 0,1,2,3
```

```
1460          LDI 0, R2                       ;R2 = COLUMN = 0,1,2,3,..,15
1461
1462    SYMBOL:
1463          LDI R2, R3                      ;R3 = SYMBOL
1464          MPYI DEINT_ROW, R3
1465          ADDI R1, R3
1466          LDI R3, IR0
1467
1468          LDF *AR0++, R4              ;pick the symbol from interleav
      ed
1469          LDF *AR1++, R5              ;buffer and place in deinterlea
      ved
1470          STF R4, *+AR2(IR0)                ;buffer
1471          STF R5, *+AR3(IR0)
1472
1473          ADDI 1, R2                      ;COLUMN++
1474          CMPI 16, R2                     ;IF COLUMN<=15 GOTO SYMBOL
1475          BNZ SYMBOL
1476
1477          ADDI 1, R1                      ;ROW++
1478          LDI 0, R2                       ;COLUMN = 0
1479          CMPI DEINT_ROW, R1
1480          BNZ SYMBOL                      ;IF ROW<=3 GOTO SYMBOL
1481
1482          ;1 128BIT BLOCK DEINTERLEAVED AND READY TO BE COPIED BACK
1483          ;TO THE BUFFER IT CAME FROM
1484
1485
1486          LDI @BUFP1, AR0
1487          LDI @BUFP2, AR1
1488          LDI R0, R1
1489          MPYI DEINT_BLOCK, R1
1490          ADDI R1, AR0                    ;Adjust the pointers @BUFP1,
       @BUFP2
1491          ADDI R1, AR1
1492
1493          LDF *AR2++, R6
1494          LDF *AR3++, R7                  ;preload registers before bl
      ock below
1495                                          ;is executed
1496          ldi DEINT_BLOCK, RC
1497          SUBI 1, RC
1498          RPTB DEINT
1499
1500          LDF *AR2++, R6
1501        ||STF R6, *AR0++
1502    DEINT:
1503          LDF *AR3++, R7
1504        ||STF R7, *AR1++
1505
1506          ;clear @FREE1 and @FREE2
1507          LDI @FREE1, R6
1508          LDI R6, R7
1509          ADDI 600H, R7
1510          CALL CLEARFLOAT
1511
1512          LDI @BUFP1, AR0
1513          LDI @BUFP2, AR1
1514          LDI @FREE1, AR2
1515          LDI @FREE2, AR3
```



```
1516
1517          ADDI 1, R0                      ;BLOCK COUNT++
1518          LDI R0, R3
1519          MPYI DEINT_BLOCK, R3
1520          ADDI R3, AR0
1521          ADDI R3, AR1
1522
1523          CMPI BLOCKS, R0                       ;BLOCK COUNT<=7 KEEP DE
      INT
1524          BNZ KEEP_DEINT
1525
1526
1527
1528    ;*****************************************************************
      **
1529    ;       This is the beginning of the Viterbi decoding algorithm.
1530
1531
1532
1533
1534    START_VITB:
1535          LDP CODES
1536          LDI 32, R0
1537          STI R0, @PUNC_COLUMN
1538
1539          LDI 2, R0
1540          STI R0, @DEEP                                 ;used to intial
      ize trellis
1541          LDI 0, R0
1542          STI R0, @FORCE_END_ZEROS
1543          subi 4, r1                      ;length-4
1544    PUNC:
1545          LDI 0, R0
1546          CMPI 5, R1
1547          LDILE 0FFFFH, R0
1548          STI R0, @FORCE_END_ZEROS
1549
1550
1551          LDI @PUNC_COLUMN, R0
1552          MPYI 2, R0
1553          CMPI 64, R0
1554          LDIZ 1, R0
1555          STI R0, @PUNC_COLUMN
1556          TSTB @ADDER_ONE_PUNC, R0
1557          LDFZ 0, R2                      ;stuff zero as I value
1558          BZ GET_Q_VALUE
1559          LDF *AR0++, R2                  ;get I value from data
1560
1561    GET_Q_VALUE:
1562          TSTB @ADDER_TWO_PUNC, R0
1563          LDFZ 0, R3                      ;stuff zero as Q value
1564          BZ SKIP_OVER
1565          LDF *AR1++, R3                  ;get real Q value from data
1566
1567
1568
1569
1570          ;LDF *AR0++(IR0), R2             ;get real I value
1571          ;LDF *AR1++(IR1), R3             ;get real Q value
1572    SKIP_OVER:
```

```
1573          LDI 16, R0
1574          LDI @FORCE_END_ZEROS, R4        ;force zero for last 5 symbo
      ls
1575
1576          LDINZ 16, R0
1577          STI R0, @PATH
1578          ;LDFNZ -7.07E-1, R3
1579
1580          LDI @DEEP, R0
1581          CMPI 16, R0                  ;if trellis is not initialized
1582          BLE INIT_VITB                ;4 branches deep then call init
      _vitb
1583
1584          ;At this point the Viterbi Decoder is initialized 4 branches
       deep.
1585          ;That is there are 16 survivors and now we can go through th
      e
1586          ;repeat process of looking at all 32 paths, calculating part
      ial
1587          ;metrics, and decoding.
1588
1589
1590          LDI 0, R0
1591         ; SUBI 4, R1                    ;R1 = length - 4
1592    TOP:
1593          LDI r0, R4                     ;calculate offset to add to
      base addr
1594          MPYI 7, R4
1595          LDI @STATE_TABLE, AR3
1596          ADDI R4, AR3                   ;add offset to base addr
1597                                         ;               Ri'Rq'
1598          LDF *++AR3, R4                 ;R4 = Ri'    O-------O
1599          LDF *++AR3, R5                 ;R5 = Rq'            /
1600                                         ;                   /
1601          LDF *++AR3(2), R6              ;R6 = Ri"          /  Ri"Rq"
1602          LDF *++AR3, R7                 ;R7 = Rq"    O/
1603    ;This is the short cut metric
1604    ;         MPYF R2, R4
1605    ;         MPYF R3, R5
1606    ;         ADDF R4, R5                 ;R5 = top partial metric
1607    ;         negf r5
1608
1609    ;         MPYF R2, R6
1610    ;         MPYF R3, R7
1611    ;         ADDF R6, R7                 ;R7 = bottom partial metric
1612    ;         negf r7
1613    ;This is the distance squared
1614          subf r2, r4
1615          subf r3, r5
1616          mpyf r4, r4
1617          mpyf r5, r5
1618          addf r4, r5
1619
1620          subf r2, r6
1621          subf r3, r7
1622          mpyf r6, r6
1623          mpyf r7, r7
1624          addf r6, r7
1625
1626
```



```
1627          ldi *-ar3(2), r4              ;top partial metric previous
      state
1628          ldi *+ar3, r6                  ;lower partial metric previo
      us state
1629
1630
1631          PUSHF R3
1632          PUSH R3
1633          PUSHF R2
1634          PUSH R2
1635
1636          CALL FIND_CORRECT_SURV
1637
1638          ;CMPF R5, R7                    ;R7 - R5
1639          ;BGT UPPER_BRANCH               ;R7 > R5 choose upper branc
      h
1640
1641          ;LDI *++AR3, R4                 ;R7 < R5  choose lower bran
      ch
1642          ;CALL FIND_CORRECT_SURV         ;R5 >= R7 choose lower bran
      ch
1643
1644    ONWARD: POP R2
1645          POPF R2
1646          POP R3
1647          POPF R3
1648
1649          ADDI 1, R0
1650          CMPI @PATH, R0                    ;repeat for all 32 paths
1651          BN TOP
1652
1653
1654          LDI @SURV_STATE_TABLE, AR4      ;update survivors
1655          LDI @NEXT_16_SURV, AR5
1656          LDI 15, RC
1657          RPTB BLOCK5
1658
1659          LDI *AR5++, R2
1660          STI R2, *AR4++
1661
1662          LDF *AR5++, R2
1663          STF R2, *AR4++
1664
1665          LDI *AR5++, R2
1666          STI R2, *AR4++
1667
1668    BLOCK5: ADDI 1, AR4
1669
1670
1671
1672
1673          LDI @SURV_STATE_TABLE, AR3      ;find smallest accum metric
1674          LDI 15, RC
1675          LDF *++AR3, R2                  ;R2 = accumulated metric
1676          ldi ar3, ar4                    ;ar4 = address of min metric
1677          RPTB BLOCK6
1678          LDF *++AR3(4), R3
1679          CMPF R3, R2                     ;R2-R3
1680          LDFGT R3, R2                    ;R2 > R3 so take r3 as min
1681    BLOCK6: LDIGT AR3, AR4
```

```
1682
1683          ;At this point R2 = minimum metric & AR4 the address of this
     metric
1684
1685          LDI *--AR4, R3                  ;R3 = output bit path histor
     y
1686          AND 1, R3                       ;R3 = output bit
1687          LDI @BIT_COUNT, R4
1688          LSH R4, R3
1689          ldi *ar2, r5
1690          OR R3, r5
1691          sti r5, *ar2
1692          ADDI 1, R4
1693          CMPI 32, R4
1694          BNZ NO_BIT_COUNT_RESET
1695          LDIZ 0, R4
1696          ADDI 1, AR2
1697    NO_BIT_COUNT_RESET:
1698          STI R4, @BIT_COUNT
1699
1700          SUBI 1, R1                      ;length -1
1701                                          ;could add force to zero for
     last
1702                                          ;5 data bits????
1703
1704
1705          BNZ PUNC
1706
1707          ;Input data is finished so clean up and wrap up Viterbi deco
     ding
1708
1709          LDI *AR4, R3                    ;get path history
1710          LSH -1, R3                      ;lose first bit which was
1711                                          ;output just above
1712
1713          ;NEGI R4, R5                     ;negate bit count
1714          LDI R3, R6
1715          ;addi 1, r4
1716          LSH R4, R3                      ;shift path history
1717          ldi *ar2, r7                    ;before writing to buffer
1718          OR R3, r7
1719          sti r7, *ar2++
1720
1721          LDI 33, r5                      ;check if any more bits to o
     utput
1722          SUBI R5, R4
1723          bp CLOSE
1724          ldi *ar2, r7
1725
1726          LSH R4, R6
1727          OR R6, r7
1728          sti r7, *ar2
1729    CLOSE:
1730          ;reset the survivor table to original values
1731          ;ldi *+ar4, r6
1732          ;ldi @TROUBLE6, ar4
1733          ;sti r6, *ar4
1734
1735          LDI @SURV_STATE_TABLE, AR3
1736          LDI 15, RC
```



```
1737          LDI 0, R2
1738          LDF 0, R3
1739          RPTB BLOCK7
1740
1741          STI R2, *AR3++
1742          STF R3, *AR3++
1743          STI R2, *AR3++
1744    BLOCK7: STI R2, *AR3++
1745
1746          LDI -27, R2
1747          STI R2, @BIT_COUNT
1748
1749
1750          RETS
1751
1752
1753
1754          ;This section of code is usedto initialize the viterbi decod
     er
1755
1756    INIT_VITB:
1757          LDI @STATE_TABLE, AR4
1758          LDI @SURV_STATE_TABLE, AR3
1759          LDI 0, R0
1760    TOP2:
1761          ;ADDI 1, AR2
1762          LDI *+AR3(2), R6                ;get last state of current s
     urvivor
1763          LDI *++AR4(3), R7               ;compare to state table for
     a match
1764          CMPI R6, R7
1765          CALLZ INIT_METRIC               ;if matches calculate branch
      metric
1766          LDI *++AR4(3), R7               ;repeat for lower branch
1767          CMPI R6, R7
1768          CALLZ INIT_METRIC
1769
1770          ADDI 1, R0
1771          CMPI 16, R0
1772          addi 1, ar4
1773          BNZ TOP2
1774
1775          LDI @SURV_STATE_TABLE, AR3      ;copy next state fields to
1776          LDI 0, R3                       ;last state fields of
1777          LDI 15, RC                      ;SURV_STATE_TABLE
1778          RPTB BLOCK2
1779          LDI *++AR3(3), R2
1780          STI R3, *AR3
1781          STI R2, *-AR3
1782    BLOCK2: ADDI 1, AR3
1783
1784          LDI @DEEP, R2                   ;make sure the proper number
1785
1786          MPYI 2, R2                      ;of survivors is initialized
      for
1787          STI R2, @DEEP                   ;the first 4 branches
1788                                          ;will be (2, 4, 8, 16)
1789
1790          CMPI 16, R2
1791          BGT PUNC
```

```
1792          MPYI 2, R2                        ;copy all 5 fields for each
    survivor
1793          LDI @SURV_STATE_TABLE, AR3
1794          LDI AR3, AR4
1795          ADDI R2, AR4
1796          LDI *AR3++, R3
1797          SUBI 1, R2
1798          RPTS R2
1799          LDI *AR3++, R3
1800        ||STI R3, *AR4++
1801          BR PUNC                           ;go back for more data
1802
1803
1804    INIT_METRIC:
1805          LDF *-AR4(2), R4                          ;R4 = state table I
1806          LDF *-AR4, R5                             ;R5 = state table Q
1807          ;MPYF R2, R4
1808          ;MPYF R3, R5
1809          ;ADDF R4, R5                               ;R5 = partial metri
    c
1810          ;negf r5
1811
1812          subf r2, r4
1813          subf r3, r5
1814          mpyf r4, r4
1815          mpyf r5, r5
1816          addf r4, r5
1817
1818          LDF *++AR3, R4                            ;get accumulated met
    ric of
1819                                                    ;current survivor
1820          mpyf BETA, r4
1821          mpyf ONE_MINUS_BETA, r5
1822
1823          ADDF R4, R5                               ;add branch metric
1824          STF R5, *AR3                              ;update accumulated
    metric
1825
1826          LDI *--AR3, R4                            ;get output history
1827          LSH -1, R4                                ;R4 >> 1
1828          LDI 0, R5                                 ;output "0"
1829          CMPI 8, R0                                ;if R0=current state
     >=8
1830          LDIGE @BIT_MASK, R5                       ;then output "1"
1831          OR R5, R4
1832          STI R4, *AR3                              ;update path history
1833
1834          STI R0, *++AR3(3)                         ;save next state
1835
1836          ADDI 1, AR3                               ;move to next surviv
    or
1837          RETS
1838
1839          ;This next section is used to find the correct survivor
1840    FIND_CORRECT_SURV:
1841
1842
1843          ;R4 last state for top partial metric
1844          ;R5 top partial metric
1845          ;R6 last state for lower partial metric
```



```
1846          ;R7 lower partial metric
1847
1848          LDI 15, RC
1849          LDI @SURV_STATE_TABLE, AR4
1850          LDI *++AR4(2), R3                 ;load last state of SURV
1851          RPTB BLOCK4
1852
1853
1854          CMPI R4, R3                       ;cmp to last state of top br
    anch
1855          ldiz ar4, ar5                     ;top branch
1856          cmpi r6, r3                       ;cmp to last state of lower
    branch
1857          ldiz ar4, ar6                     ;lower branch
1858
1859          NOP
1860    BLOCK4: LDI *++AR4(4), R3               ;get next last state
1861
1862
1863          ;AR5 address of last state for top branch
1864          ;AR6 address of last state for lower branch
1865
1866          LDF *-AR5, R4
1867          mpyf BETA, r4
1868          mpyf ONE_MINUS_BETA, r5
1869
1870          ADDF R4, R5                       ;r5=accum metric with top br
    anch met
1871
1872          LDF *-AR6, R6
1873          mpyf BETA, r6
1874          mpyf ONE_MINUS_BETA, r7
1875
1876          ADDF R6, R7                       ;R7=accum metric with lower
    branch met
1877
1878          CMPF R5, R7                       ; R7 - R5
1879
1880          BGT UPPER_BRANCH                  ;R7 > R5 choose top branch R
    5
1881
1882          ldi ar6, ar4                      ;R5 > R7 choose lower branch
     R7
1883          CALL UPDATE_SURV
1884          RETS
1885
1886    UPDATE_SURV:
1887          ;we have the correct survivor so now we update it
1888
1889          ;R7 = accumulated metric
1890          ;ar4= last state SURV_TABLE
1891
1892          LDI @NEXT_16_SURV, AR5
1893          LDI R0, R3
1894          MPYI 3, R3                                ;get offset to add t
    o base add
1895          ADDI R3, AR5
1896
1897          STF R7, *+AR5                             ;save accum metric
1898
```

```
1899          ;save bit
1900          LDI *-AR4(2), R5
1901          LSH -1 , R5
1902          LDI 0, R6
1903          CMPI 8, R0
1904          LDIGE @BIT_MASK, R6
1905          OR R6, R5
1906          STI R5, *AR5                        ;save past history
1907          STI R0, *+AR5(2)                          ;save last state
1908          RETS
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920    UPPER_BRANCH:
1921          ldf r5, r7                             ;R5 metric in R7
1922          LDI AR5, AR4                          ;choose upper branch
1923          CALL UPDATE_SURV
1924          RETS
1925
1926
1927
1928
1929
1930    ;*************************************************************
      ***
1931    ;       Interrupt 1 is responsible for obtaining real data from the
1932    ;       I and Q channels and then hard decoding each dibit while
1933    ;       simultaneously searching for the occurence of a flag.
1934    ;
1935    ;       Interrupt 1 occurs once every symbol duration time period.
1936    ;       Currently a symbol lasts for:
1937    ;
1938    ;                       6 * 6.6micros = 39.6 micros
1939    ;
1940    ;                       39.6ms/60ns   = 660 instructions
1941    ;
1942    ;       This allows for the execution of 660 instructions between
1943    ;       interrupt trigger times.
1944    ;
1945    ;       Currently this interrupt consists of  xx instructions giving
1946    ;       rise to :
1947    ;                       660 - xx = yy instructions of main code.
1948    ;
1949    ;       Requires:
1950    ;               Nothing
1951    ;       Modifies:
1952    ;               R0, R1, R2, R3, R4, R5, R6, R7
1953    ;               AR0, AR1, AR2
1954    ;       Returns:
1955    ;               Nothing
1956    ;
1957    ;
```



```
1958    ;*************************************************************
      ***
1959
1960    RCV:
1961          PUSH ST                 ;IMPORTANT MUST USE OTHERWISE REST O
      F
1962                                  ;PROGRAM WILL NOT WORK PROPERLY
1963          XOR 2000H, ST           ;disable interrupts
1964
1965          PUSH DP                 ;save register contents
1966
1967          push ir0
1968          push ir1
1969          push bk
1970          push ie
1971          push if
1972          push iof
1973          push rs
1974          push re
1975          push rc
1976
1977          PUSH R0
1978          pushf r0
1979          PUSH R1
1980          pushf r1
1981          PUSH R2
1982          pushf r2
1983          PUSH R3
1984          pushf r3
1985          PUSH R4
1986          pushf r4
1987          PUSH R5
1988          pushf r5
1989          PUSH R6
1990          pushf r6
1991          PUSH R7
1992          pushf r7
1993          PUSH AR0
1994          PUSH AR1
1995          PUSH AR2
1996
1997          LDP CODES
1998          LDI @ADCHANA1, AR0              ;read I channel
1999          LDI @ADCHANB1, AR1              ;read Q channel
2000          LDI *AR0, R0
2001        ||LDI *AR1, R1
2002
2003
2004          ash -16, r0
2005          ash -16, r1
2006          FLOAT R0, R2                    ;convert A to D hex value to
       float
2007          FLOAT R1, R3
2008
2009          MPYF @SCALE, R2                 ;scale float value down
2010          MPYF @SCALE, R3
2011
2012
2013          ;At this point R2 and R3 contain a scaled down floating poin
      t
```

```
2014          ;representaion of the I and Q channel just read
2015
2016          LDI @REAL_IBIT_POINTER, AR0     ;get current pointer to I an
    d Q
2017          LDI @REAL_QBIT_POINTER, AR1     ;real data
2018
2019          STF R2, *AR0++(1)%              ;save real value I and Q in
2020          STF R3, *AR1++(1)%              ;circular memory
2021
2022          STI AR0, @REAL_IBIT_POINTER     ;update pointers to real dat
    a
2023          STI AR1, @REAL_QBIT_POINTER
2024
2025          ldf @OLDI_INT, r0
2026          ldf @OLDQ_INT, r1
2027          CALL HARDDECODE                 ;decode current dibit
2028          STF R2, @OLDI_INT               ;save current dibit for refe
    rence
2029          STF R3, @OLDQ_INT               ;by the next future dibit
2030
2031          CALL FLAG_CHECKER               ;checks if flag has been enc
    ountered
2032
2033          POP AR2
2034          POP AR1
2035          POP AR0
2036          popf r7
2037          POP R7
2038          popf r6
2039          POP R6
2040          popf r5
2041          POP R5
2042          popf r4
2043          POP R4
2044          popf r3
2045          POP R3
2046          popf r2
2047          POP R2
2048          popf r1
2049          POP R1
2050          popf r0
2051          POP R0
2052
2053          pop rc
2054          pop re
2055          pop rs
2056          pop iof
2057          pop if
2058          pop ie
2059          pop bk
2060          pop ir1
2061          pop ir0
2062
2063          POP DP
2064          POP ST
2065          OR 2000h, ST
2066          RETI
2067
2068  ;*************************************************************
    ********
```



```
2069  ;       All other interrupts simply return
2070
2071  NO:     RETI
2072
2073  ;*************************************************************
    *******
2074  ;       This section clears memory chunks specified by AR0 --> AR1
2075  CLEAR:
2076          SUBI R6, R7
2077          LDIN 1, R7
2078          BN ERROR
2079          LDI R6, AR0
2080          LDI 0, R6
2081          RPTS R7
2082          STI R6, *AR0++
2083          RETS
2084  ;*************************************************************
    *******
2085  ERROR:
2086          LDP DUAL
2087          STI R7, @ERROR_NUM
2088  DEAD:   BR DEAD
2089  ;*************************************************************
    ******
2090  ;       This section clears memory chunks specified by AR0 --> AR1
2091  CLEARFLOAT:
2092          SUBI R6, R7
2093          LDIN 1, R7
2094          BN ERROR
2095          LDI R6, AR0
2096          LDF 0, R6
2097          RPTS R7
2098          STF R6, *AR0++
2099          RETS
2100
2101
2102
2103          .end
```

```
1     ;       VARS.asm        V1.00           JAN 93
2     ;
3     ;       This file is used to set variables which are constantly
4     ;       used throughout the assembly section of the code.
5     ;
6                     .text
7     STACK_SIZE      .set    400h    ;size of system stack
8     FP              .set    AR3     ;frame pointer
9     DELTA           .set    2       ;amount to jump in SIN table
10    INITIAL         .set    800h
11    NULL            .set    0
12
13    DATALENGTH      .set    100h
14    CODES           .set    0
15    DUAL            .set    30000h
16    ONCHIP          .set    809C00h
17
18    DEINT_ROW        .set    16
19    DEINT_BLOCK      .set    256     ;symbols per block
20    BLOCKS          .set    2
21
22    K               .set    809c00h
23    POLY1           .set    809c01h
24    POLY2           .set    809c02h
25    CRC_CCITT       .word   69665
26    CRC_32          .word   4374732215    ;problem with length
27    FLAG            .set    809c05h
28
29    MENU_MASK       .set    3h
30    PACKET_MASK     .set    7h
31    LENGTH_MASK     .set    3ffh
32    CRC_MASK        .word   01FFFE0H
33    HIGH_MASK       .word   0ffff0000h
34
35    CPC_ONE_ADDER1              .set    0h
36    CPC_ONE_ADDER2              .set    1h
37    CPC_TWO_ADDER1              .set    2h
38    CPC_TWO_ADDER2              .set    3h
39
40
41    ADCHANA         .set    804000H
42    ADCHANB         .set    804001H
43    TROUBLEA        .WORD   31000H
44
45    Q_START         .set    809c16H         ;contains add of Q table
46    Q_END           .set    809c17H           ;contains add of Q end tabl
      e
47    SERIALO         .set    809C18H
48    TABLE_ENC       .set    809C49H         ;CONTAINS POINTER TO TAB_ENC
49    IBIT_POINTER    .set    809C4AH         ;contains IBIT pointer
50    QBIT_POINTER    .set    809C4BH         ;contains Qbit pointer
51
52
53    SINE_POINTER    .set    809D00H
54    COSINE_POINTER  .set    809D01H
55    POINT_COUNT     .set    809D02H
56    DATA_WORD       .set    809D03H
57    CURRENT_ADDRESS .set    809D04H
58    END_ADDRESS     .set    809D05H
59    Q_OFFSET        .set    809D06H
```



```
60    TRANSMISSION    .set    809d07H
61    Q_OFF_TRANS     .set    809D08H
62    GET_NEWFRAME_FLAG .set  809D09H
63    DIBIT_COUNT     .set    809D0AH
64
65
66    ;RCVR VARIABLES
67    ADCHANA1                .word   804000h
68    ADCHANB1                .word   804001h
69    ;SCALE                   .float  3.052e-5
70    SCALE                   .float  9.155553e-5
71
72
73    REAL_IBIT_POINTER       .word   31000h
74    REAL_QBIT_POINTER       .word   32000h
75    FLAG_ADDRESS_TABLE      .word   33000h
76    TABLE_TOP               .word   33000h
77    TABLE_BOTTOM            .word   330ffh
78    FLAG_COMP               .word   3B1492H
79    MASK                    .word   0FFFFFF00H
80    MASK1                   .WORD   00ffffffh
81    OLDI                    .float  0
82    OLDQ                    .float  0
83    OLDI_INT                .float  0
84    OLDQ_INT                .float  0
85    FLAG_TO_BE              .word   0
86    START_FRAME             .word   32000h          ;init to dummy value
      s
87    STOP_FRAME              .word   32fffh          ;init to dummy value
      s
88    CURRENT_FLAG            .word   33000h          ;init to table_top
89    CIRC_BOTTOM             .word   31000h
90    CIRC_TOP                .word   32000H
91    TROUBLE                 .word   30100h
92    TROUBLE2                .word   300F0h
93    TROUBLE3                .word   30110h
94    TROUBLE4                .word   30120h
95    TROUBLE6                .word   30130h
96    HD_LENGTH               .word   0
97    DIBIT                   .word   0
98    THRESHOLD               .word   12
99    CIRCLESS1               .word   31fffh
100   RCVD_SIGNAL_ENERGY      .word   3000fh
101   SYMBOLS                 .word   30010h
102   buf                     .word   31000h
103   buf2                    .word   32000h
104   CURRENT_START           .WORD   0
105   ;VITERBI DECODER TABLES & VARIABLES
106
107   BETA                    .set  9.9e-1
108   ONE_MINUS_BETA          .set  1.0e-2
109   BIT_MASK                .word   80000000h
110   BIT_COUNT               .word   -27
111   DEEP                    .word   2
112   PATH                    .word   16
113   FORCE_END_ZEROS         .word   0
114   PUNC_COLUMN             .word   1
115   ADDER_ONE_PUNC          .word   7
116   ADDER_TWO_PUNC          .word   7
117   ADDER_1PRIME_PUNC       .word   7
```

```
118   ADDER_2PRIME_PUNC       .word   7
119   SEQUENCES               .word   0
120
121                           .label  TABLE_STATE
122                           .word   0               ;state 0
123                           .float  -7.07e-1        ;I(s)
124                           .float  -7.07e-1        ;Q(s)
125                           .word   0               ;s
126                           .float  7.07e-1          ;I(s+1)
127                           .float  7.07e-1          ;Q(s+1)
128                           .word   1               ;s+1
129
130                           .word   1               ;state 1
131                           .float  7.07e-1         ;I(s)
132                           .float  -7.07e-1        ;Q(s)
133                           .word   2               ;s
134                           .float  -7.07e-1        ;I(s+1)
135                           .float  7.07e-1         ;Q(s+1)
136                           .word   3               ;s+1
137
138                           .word   2               ;state 2
139                           .float  -7.07e-1        ;I(s)
140                           .float  7.07e-1         ;Q(s)
141                           .word   4               ;s
142                           .float  7.07e-1         ;I(s+1)
143                           .float  -7.07e-1        ;Q(s+1)
144                           .word   5               ;s+1
145
146                           .word   3               ;state 3
147                           .float  7.07e-1         ;I(s)
148                           .float  7.07e-1         ;Q(s)
149                           .word   6               ;s
150                           .float  -7.07e-1        ;I(s+1)
151                           .float  -7.07e-1        ;Q(s+1)
152                           .word   7               ;s+1
153
154                           .word   4               ;state 4
155                           .float  -7.07e-1        ;I(s)
156                           .float  7.07e-1         ;Q(s)
157                           .word   8               ;s
158                           .float  7.07e-1         ;I(s+1)
159                           .float  -7.07e-1        ;Q(s+1)
160                           .word   9               ;s+1
161
162                           .word   5               ;state 5
163                           .float  7.07e-1         ;I(s)
164                           .float  7.07e-1         ;Q(s)
165                           .word   10              ;s
166                           .float  -7.07e-1        ;I(s+1)
167                           .float  -7.07e-1        ;Q(s+1)
168                           .word   11              ;s+1
169
170                           .word   6               ;state 6
171                           .float  -7.07e-1        ;I(s)
172                           .float  -7.07e-1        ;Q(s)
173                           .word   12              ;s
174                           .float  7.07e-1         ;I(s+1)
175                           .float  7.07e-1         ;Q(s+1)
176                           .word   13              ;s+1
177
```



```
178                           .word   7               ;state 7
179                           .float  7.07e-1         ;I(s)
180                           .float  -7.07e-1        ;Q(s)
181                           .word   14              ;s
182                           .float  -7.07e-1        ;I(s+1)
183                           .float  7.07e-1         ;Q(s+1)
184                           .word   15              ;s+1
185
186                           .word   8               ;state 8
187                           .float  7.07e-1         ;I(s)
188                           .float  7.07e-1         ;Q(s)
189                           .word   0               ;s
190                           .float  -7.07e-1        ;I(s+1)
191                           .float  -7.07e-1        ;Q(s+1)
192                           .word   1               ;s+1
193
194                           .word   9               ;state 9
195                           .float  -7.07e-1        ;I(s)
196                           .float  7.07e-1         ;Q(s)
197                           .word   2               ;s
198                           .float  7.07e-1         ;I(s+1)
199                           .float  -7.07e-1        ;Q(s+1)
200                           .word   3               ;s+1
201
202                           .word   10              ;state 10
203                           .float  7.07e-1         ;I(s)
204                           .float  -7.07e-1        ;Q(s)
205                           .word   4               ;s
206                           .float  -7.07e-1        ;I(s+1)
207                           .float  7.07e-1         ;Q(s+1)
208                           .word   5               ;s+1
209
210                           .word   11              ;state 11
211                           .float  -7.07e-1        ;I(s)
212                           .float  -7.07e-1        ;Q(s)
213                           .word   6               ;s
214                           .float  7.07e-1         ;I(s+1)
215                           .float  7.07e-1         ;Q(s+1)
216                           .word   7               ;s+1
217
218                           .word   12              ;state 12
219                           .float  7.07e-1         ;I(s)
220                           .float  -7.07e-1        ;Q(s)
221                           .word   8               ;s
222                           .float  -7.07e-1        ;I(s+1)
223                           .float  7.07e-1         ;Q(s+1)
224                           .word   9               ;s+1
225
226                           .word   13              ;state 13
227                           .float  -7.07e-1        ;I(s)
228                           .float  -7.07e-1        ;Q(s)
229                           .word   10              ;s
230                           .float  7.07e-1         ;I(s+1)
231                           .float  7.07e-1         ;Q(s+1)
232                           .word   11              ;s+1
233
234                           .word   14              ;state 14
235                           .float  7.07e-1         ;I(s)
236                           .float  7.07e-1         ;Q(s)
237                           .word   12              ;s
```

```
                        .float  -7.07e-1        ;I(s+1)
                        .float  -7.07e-1        ;Q(s+1)
                        .word   13              ;s+1

                        .word   15              ;state 15
                        .float  -7.07e-1        ;I(s)
                        .float  7.07e-1         ;Q(s)
                        .word   14              ;s
                        .float  7.07e-1         ;I(s+1)
                        .float  -7.07e-1        ;Q(s+1)
                        .word   15              ;s+1

                        .label  TABLE_SURV
SURV0                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV1                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV2                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV3                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV4                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV5                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV6                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV7                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state
```



```

SURV8                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV9                   .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV10                  .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV11                  .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV12                  .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV13                  .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV14                  .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

SURV15                  .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
                        .word   0               ;next state

                        .label  TABLE_NEXT_16
NEXTSURV0               .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state

NEXTSURV1               .word   0               ;past history
                        .float  0               ;accumulated metric
                        .word   0               ;last state
```

```
358
359
360   NEXTSURV2             .word   0               ;past history
361                         .float  0               ;accumulated metric
362                         .word   0               ;last state
363
364
365
366   NEXTSURV3             .word   0               ;past history
367                         .float  0               ;accumulated metric
368                         .word   0               ;last state
369
370
371
372   NEXTSURV4             .word   0               ;past history
373                         .float  0               ;accumulated metric
374                         .word   0               ;last state
375
376
377
378   NEXTSURV5             .word   0               ;past history
379                         .float  0               ;accumulated metric
380                         .word   0               ;last state
381
382
383
384   NEXTSURV6             .word   0               ;past history
385                         .float  0               ;accumulated metric
386                         .word   0               ;last state
387
388
389
390   NEXTSURV7             .word   0               ;past history
391                         .float  0               ;accumulated metric
392                         .word   0               ;last state
393
394
395
396   NEXTSURV8             .word   0               ;past history
397                         .float  0               ;accumulated metric
398                         .word   0               ;last state
399
400
401
402   NEXTSURV9             .word   0               ;past history
403                         .float  0               ;accumulated metric
404                         .word   0               ;last state
405
406
407
408   NEXTSURV10            .word   0               ;past history
409                         .float  0               ;accumulated metric
410                         .word   0               ;last state
411
412
413
414   NEXTSURV11            .word   0               ;past history
415                         .float  0               ;accumulated metric
416                         .word   0               ;last state
417
```



```
418
419
420   NEXTSURV12            .word   0               ;past history
421                         .float  0               ;accumulated metric
422                         .word   0               ;last state
423
424
425
426   NEXTSURV13            .word   0               ;past history
427                         .float  0               ;accumulated metric
428                         .word   0               ;last state
429
430
431
432   NEXTSURV14            .word   0               ;past history
433                         .float  0               ;accumulated metric
434                         .word   0               ;last state
435
436
437
438   NEXTSURV15            .word   0               ;past history
439                         .float  0               ;accumulated metric
440                         .word   0               ;last state
441
442
443
444   STATE_TABLE           .WORD   TABLE_STATE
445   SURV_STATE_TABLE      .WORD   TABLE_SURV
446   NEXT_16_SURV          .WORD   TABLE_NEXT_16
447
448
449   ;*****************************************************************
450   ;
451   ;       Memory Map of On chip memory $30000 - $3ffff
452   ;
453   ;       v1.00           Feb 93
454   ;
455   LENHEAD0        .set    30000h          ;length of unencoded header
456   LENDATA0        .set    30001h          ;length of unencoded data
457   LENHEADENC      .set    30002h          ;length of encoded header
458   LENDATAENC      .set    30003h          ;length of encoded data
459   PACKET_NUM      .set    30004h          ;packet number Ns
460   RATE            .set    30005h          ;rate to be used for encodin
      g
461   MENU_OPTION     .set    30006h          ;menu option 1, 2, or 3
462   ERROR_NUM       .set    30007h          ;error number
463   CONTROL_WORD    .set    30008h          ;control info from protocol
464   LENHEADP1       .set    3000ah          ;bit length of P1 header bit
      s
465   LENHEADP2       .set    3000bh          ;bit length of P2 header bit
      s
466   LENDATA_WORD    .set    3000ch          ;word length of data chunk
467   LENDATA_BIT     .set    3000dh          ;# of left over bits from da
      ta chunk
468   TOTAL_WORDS     .set    3000eh          ;word length of current fram
      e
469   CODE            .set    3000fh
470   ;unused 30009-3000f
471   ;
472   ; Careful !!   these are pointers to specific memory locations
```

```
473
474                     .text
475   VIRGIN_HEADER     .word   30010h          ;start of virgin header
476   TAIL1             .set    30012h          ;used for header CRC calc
477   VIRGIN_DATA       .word   30013h          ;start of virgin data
478   TAIL2             .set    30030h          ;space to store data CRC
479   HEADBUF1          .word   30031h          ;header buffer 1
480   HEADBUF2          .word   30033h          ;header buffer 2
481   HEADBUF3          .word   30035h          ;header buffer 3
482   HEADBUF4          .word   30039h          ;header buffer 4
483
484   DATABUFP1         .word   3003dh          ;data buffer 1
485   DATABUFP2         .word   3005ch          ;data buffer 2
486   ;7bh--->7fh Unused
487   ACK0              .set    3007Bh          ;ACK0
488   STROBE_RCVR       .set    3007ch
489   STROBE_HOST       .set    3007dh
490
491   FRAMEBUFP1        .word   30080h          ;frame buffer 1
492   FRAMEBUFP2        .word   300a0h          ;frame buffer 2
493
494   FLAG0P1           .word   300c0h          ;flag for frame 0, P1
495   PACKET0HARDP1     .word   300c1h
496   PACKET1HARDP1     .word   300e2h
497   BUFP1             .word   30100H
498   BUFP2             .word   30500H
499   CPC1I             .word   30900H
500   CPC1Q             .word   30b00H
501   CPC2I             .word   30d00H
502   CPC2Q             .word   33800H
503   FREE1             .WORD   33100H
504   FREE2             .WORD   33400H
505
506
507
```