# *A 4800 BPS GTFM MODEM*

# *PART 2–*

# *IMPLEMENTATION AND PERFORMANCE TESTS*

Prepared for
Communications Research Center
Shirley Bay, Ontario

By

W.P. LeBlanc Co–investigator
J.K. Cavers Principal Investigator

July 1986

RELEASABLE

| | |
|---|---|
| UNIVERSITY: | Simon Fraser |
| TITLE: | A 4800 BPS GTFM modem<br>part 1--Analysis and simulation<br>part 2--Implementation and<br>performance tests |
| DOC CR NO: | DOC-CR-SP-86-039 |
| DSS CONTRACT NO: | 36100-5-0088 |
| DOC SC. AUTH.: | Brian Bryden |
| AUTHOR: | W.P. LeBlanc & J.K. Cavers |
| REPORT DATE: | July 1986 |

~~NOW CATALOGUED AS: P 91 C654 L434 1986~~

COMMUNICATIONS RESEARCH CENTRE

RELEASABLE
DOC-CR-SP-86-039

# ORIGINAL DOCUMENT REVIEW AND PUBLICATION RECORD

| | | |
|---|---|---|
| SECTOR<br>DGSTA | BRANCH<br>DSS | DATE<br>January 1987 |

PURPOSE This form is for use during review of the DOC-CR contractor reports.
It is designed to: record decisions for classification,
record reasons for classification and cautionary markin
provide for indexing requirements.

INSTRUCTIONS * 1 copy of the completed form must accompany the contractor report package submitted to the CRC Library.

* Complete the following items as applicable.

| | |
|---|---|
| 1. DOC-CR NO.<br>DOC-CR-SP-86-039 | 2. DSS CONTRACT NO.<br>36100-5-0088 |
| 3. TITLE:<br>A 4800 BPS GTFM modem<br>part 1--Analysis and simulation. part 2--Implementation and performance tests | 4. DATE<br>July 1986 |

5. CONTRACTOR

Simon Fraser University

| | | |
|---|---|---|
| 6. SCIENTIFIC AUTHORITY<br>Brian Bryden | 7. LOCATION<br>DSS/CRC | 8. TEL. NO.<br>998-2515 |

9. CONTRACTOR REPORT CLASSIFICATION:

RELEASABLE ☒ CONDITIONALLY RELEASABLE □ NON-RELEASABLE □

* REASONS FOR CLASSIFICATION:

10. NO. OF COPIES SUBMITTED TO LIBRARY:

EXECUTIVE SUMMARY □ FINAL REPORT ☒ 5 sets

.................................. ....................

Scientific Authority's Signature Date

This form is not official therefore it is not signed.

# A 4800 BPS GTFM MODEM
# PART TWO-
# IMPLEMENTATION AND PERFORMANCE TESTS

Prepared for
Communications Research Center
Shirley Bay, Ontario

Under DSS Contract Number
06ST.36100-5-0088
(OST85-00238)

By
W.P. LeBlanc Co-investigator
J.K. Cavers Principle Investigator

Faculty of Applied Science
School of Engineering Science
Simon Fraser University
. Burnaby, B.C. V5A 1S6

July 1986

# A 4800 BPS GTFM MODEM

## PART 2- IMPLEMENTATION AND PERFORMANCE TESTS

# 1. INTRODUCTION

This report describes the implementation and performance tests of a 4800 bps Generalized Tamed Frequency Modulation, (GTFM), modem intended for the MSAT environment.

A companion report "A 4800 BPS GTFM MODEM, PART 1- ANALYSIS AND SIMULATION", describes the simulation study and GTFM in greater detail than the present report.

The modem is full duplex, interfaces to a radio at a 4.8 kHz IF (intermediate frequency), and to the DTE (Vocoder) through a subset of the RS232 standard.

The major advantages over other modulation formats, demodulation algorithms, and implementations are as follows:

1. The performance of the modem is within 2.1 dB of coherent FFSK, and within 1.8 dB of coherent GTFM, (GTFM(0.62,0.36)). However, no phase reference is required at the demodulator resulting in fast acquisition, and relative insensitivity to the phase noise which is common in the MSAT environment.
2. The bandwidth efficiency of the modem, ($\approx$ 1 bps/Hz), means close channel spacing. Adjacent channels (with 5 kHz channel spacing) 13.5 dB more powerful than the desired channel have only a small effect on the performance, (0.4 dB).
3. The digital implementation reduces parts counts, the effects of aging, and simplifies modifications and enhancements.
4. The modem operates in burst mode, with fast acquisition, and can tolerate up to ± 400 Hz frequency offsets in the received signals center frequency, at signal to noise ratios above 7.0 dB ($E_b/N_o$). Bit tracking is also incorporated allowing transmissions of indeterminate length.

# 2. HARDWARE IMPLEMENTATION

## 2.1 General Organization

Figure 2.1.1 depicts the functional organization of the modem. The modem communicates with the DTE over a synchronous RS232 subset, (RS232 levels). The modem supplies a 4.8 kHz IF output, and has a 4.8 kHz IF input.

The modem is based on the TMS32010 digital signal processor operating at a clock frequency of 19.6608 MHz. For experimental purposes, the modem is full duplex, utilizing two TMS320s. Minor hardware (and software) changes are necessary to create a half duplex modem based on a single TMS320. The processors communicate with the DTE over RTS, RD, TD, RC and TC.

The modulator requires one analog output (4.8 kHz IF out), one digital output (TC), and two digital inputs, (RTS, TD). The demodulator needs only one analog input (4.8 kHz IF in) and two digital outputs (RD, RC).

By utilizing a 19.6608 MHz crystal, and a data rate of 4800 bps, we have exactly 1024 cycles per bit interval. All timing is to be done in software. Thus, it is extremely important that all sections of code between successive inputs from the A/D converter execute in the same number of cycles, (precisely 1024/md cycles, where md is the number of samples per bit).

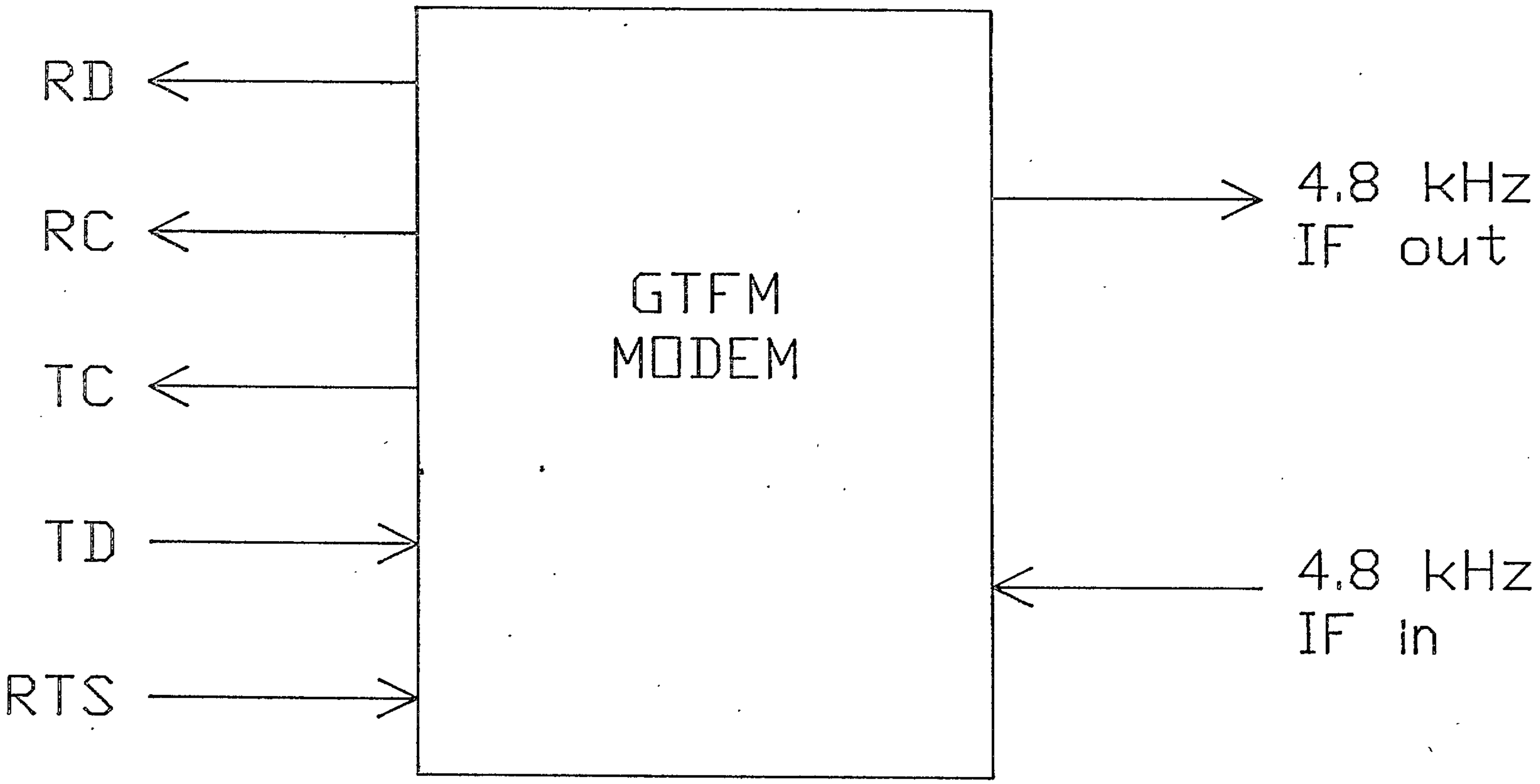


Figure 2.1.1

## 2.2 Modulator

As discussed previously, the modulator consists of one analog output, two digital inputs, and one digital output. The schematic is shown in figure 2.2.1.

The modulator hardware consists of reset hardware, a digital to analog converter, filters, one bit of digital output, two bits of digital input, level shifters, port decoding logic, and an interrupt request line.

For a hardware reset to occur, the RS line on the TMS320 must be held low for a minimum of 5 clock cycles. The reset hardware in figure 2.2.1 ensures this. The analog output port has an 8 bit latch, to hold the current output sample, an 8 bit digital to analog converter, a high pass filter ($f_c$ = 230 Hz), and a sixth order butterworth switched capacitor reconstruction filter ($f_c$ = 10 kHz). Since the switched capacitor filter is a sampled data filter, it also needs an anti-aliasing and reconstruction filter. The output of the switched capacitor filter is amplified and buffered to supply a 10 volt peak to peak, low impedance output.

One bit of digital output is provided by the ALS874, followed by the MC-1488 which level shifts to RS232 compatible levels. The LS125, preceeded by the MC-1489 allows the processor to read the state of two RS232 compatible inputs.

The port decoding logic decodes the address and control bus in order to supply the appropriate control lines to the ports. Address line All is also decoded to allow table writes into program memory locations >800 to >FFF. The table writes (TBLW) are usefull for debugging purposes. The ports and their functions are as in figure 2.2.2.

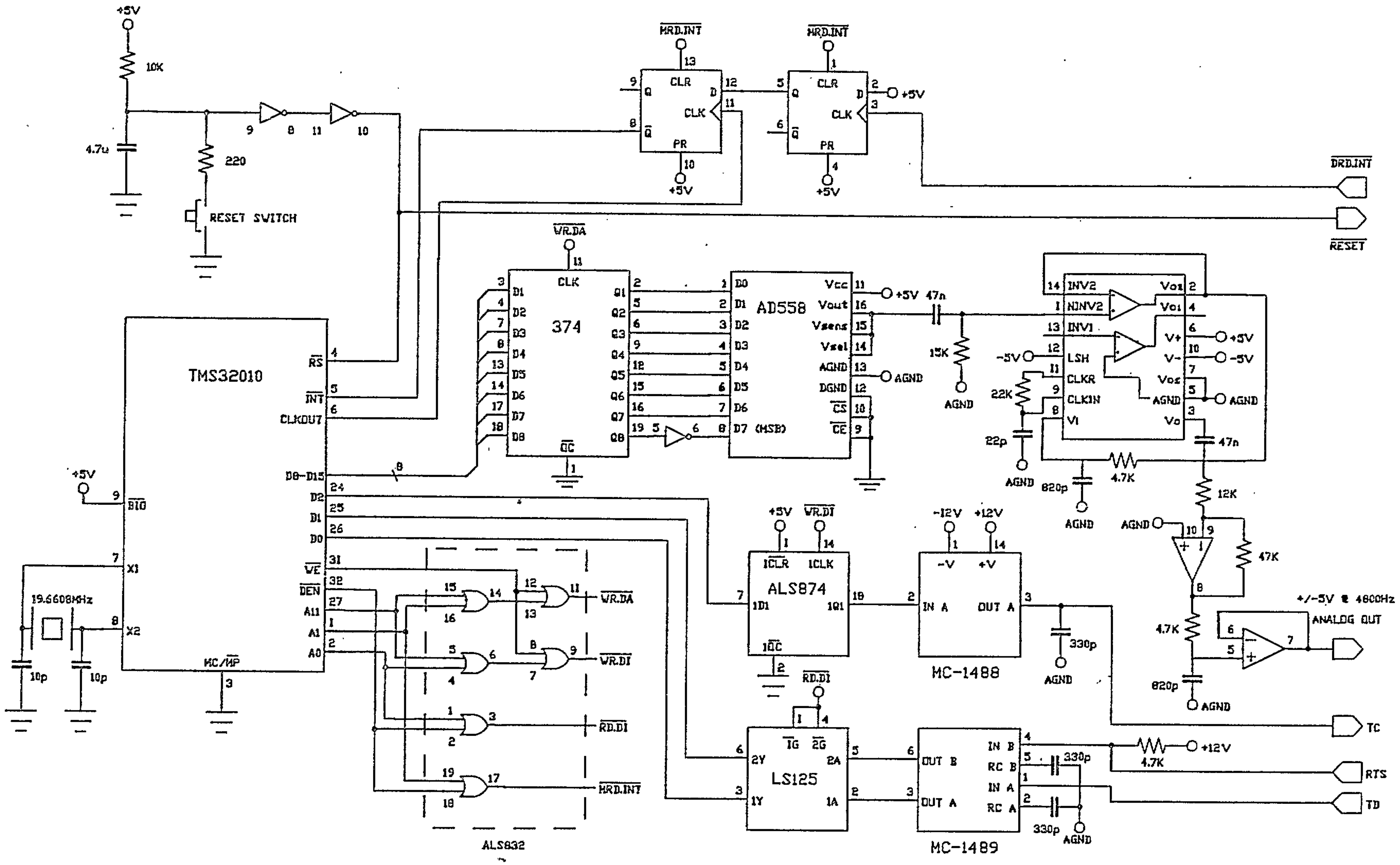

+5V
10K
4.7u
220
RESET SWITCH
HRD.INT
CLR
PR
Q
CLK
D
DRD.INT
RESET
WR.DA
374
AD558
Vcc
Vout
Vsens
Vsel
AGND
DGND
CS
CE
D7 (MSB)
47n
15K
-5V
22K
22p
820p
4.7K
12K
47K
INV2
NINV2
INV1
LSH
CLKR
CLKIN
Vi
Vo1
Vo2
V+
V-
Vos
Vo
TMS32010
RS
INT
CLKOUT
D0-D15
BIO
X1
X2
MC/MP
WE
DEN
A11
A1
A0
19.6608MHz
10p
ALS832
WR.DI
RD.DI
ALS874
1CLR
1CLK
1D1
1Q1
1OC
LS125
1G
2G
2Y
1Y
2A
1A
-12V
+12V
IN A
OUT A
MC-1488
330p
OUT B
IN B
RC B
RC A
MC-1489
+/-5V @ 4800Hz
ANALOG OUT
TC
RTS
TD

Figure 2.2.1

Modulator Input/ Output Ports

| Output Ports | Function |
|---|---|
| 1 | digital to analog converter, D<15\|8> = current analog output |
| 2 | data intertface: D<2> = <TC><br>Transmit Clock |
| Input Ports | |
| 1 | clear interrupt request line |
| 2 | data interface: D<1\|0> = <RTS\|TD><br>Request To Send, Transmit Data |

Figure 2.2.2

A provision was made to allow the modulator to be interrupted on the rising edge of DRD.INT. Reading from port 2 will clear this request. In this way, the demodulator may interrupt the modulator. This facility is useful for debugging purposes.

## 2.3 Demodulator

Provided in the demodulator (figure 2.3.1) is one analog input port, two bits of digital output, and one read line, (for interrupting the modulator).

A reset line (from the modulator) is held low for a minimum of 5 clock cycles during a reset operation.

The ALS874 provides two bits of digital output, and the MC1488 buffers the outputs to provide RS232 compatibility.

The received 4.8 kHz IF signal is first low pass filtered by a sixth order butterworth switch cap filter ($f_c$ = 9 kHz), and then applied to the analog to digital converter hardware, (sample and hold, and A/D converter). A conversion command for the analog to digital converter (A/D) is initiated by executing an IN command from the A/D port (port 1). When an input instruction is executed, the last sample is clocked into the eight bit latch, and read into the processor. The hold line on the sample to hold is asserted, and the instruction to commence a new conversion is applied to the A/D converter, (figure 2.3.2). Once the A/D is ready to begin a conversion, (rising edge of STATUS), CNVT is set to logic one, and the conversion begins. The conversion is complete on the falling edge of STATUS, and the hold command is negated. Thus, each input command (from the A/D port) gets a new sample, and initiates a new conversion.

The port decoding logic decodes the address and control bus (of the TMS320) to provide the necessary handshaking lines which control the input and output ports. The input and output ports and their functions are displayed in figure 2.3.3.

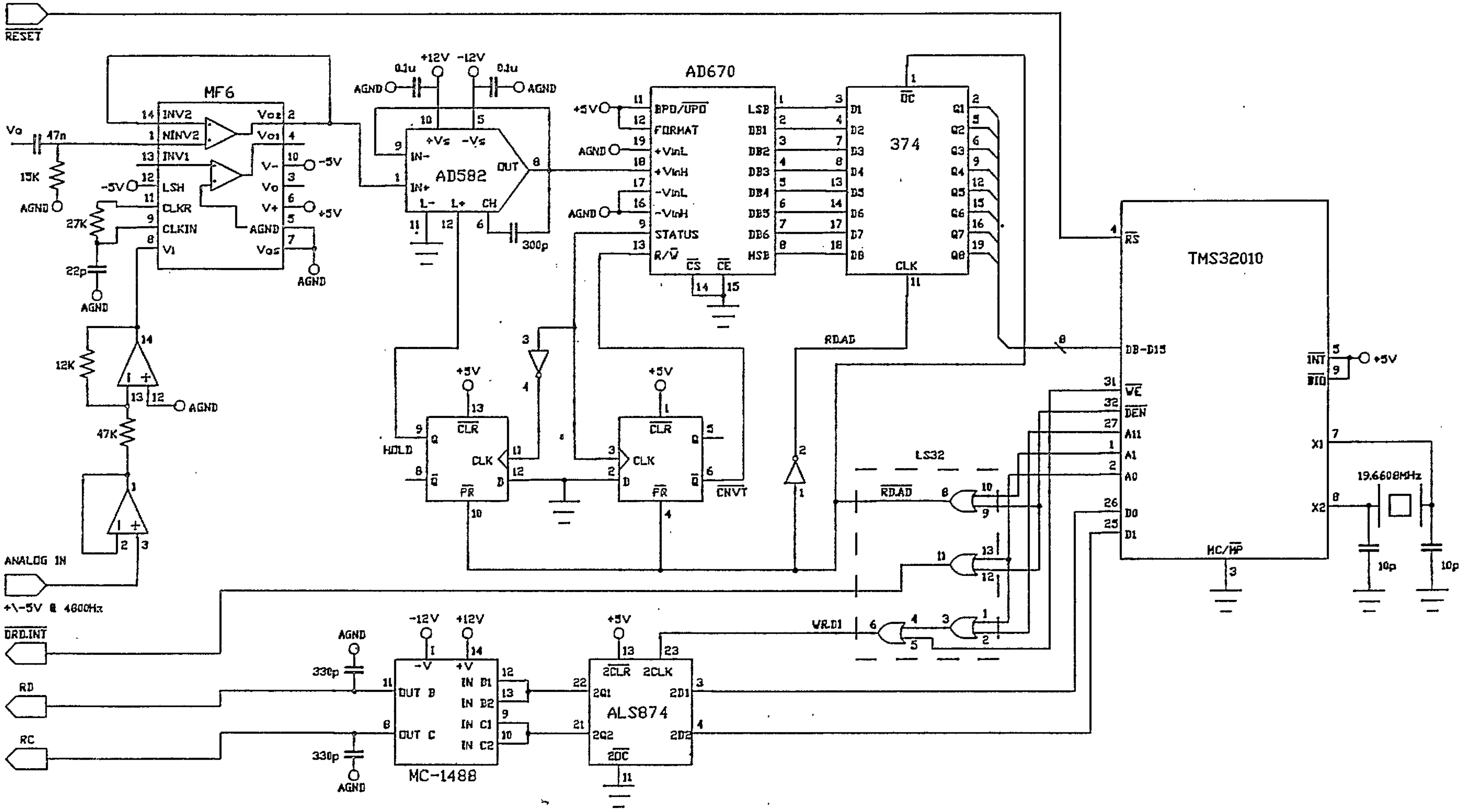

RESET
MF6
INV2
NINV2
INV1
LSH
CLKR
CLKIN
V1
Vo2
Vo1
V-
Vo
V+
AGND
Vos
Vo
47n
15K
AGND
-5V
27K
22p
AGND
-5V
+5V
AGND
0.1u
+12V
-12V
0.1u
AGND
AGND
AD582
IN-
IN+
+Vs
-Vs
OUT
L-
L+
CH
300p
AD670
BPO/UPO
FORMAT
+VinL
+VinH
-VinL
-VinH
STATUS
R/W
CS
CE
LSB
DB1
DB2
DB3
DB4
DB5
DB6
MSB
+5V
AGND
AGND
374
OC
D1
D2
D3
D4
D5
D6
D7
D8
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
CLK
RDAD
TMS32010
RS
DB-D15
WE
DEN
A11
A1
A0
D0
D1
MC/MP
INT
BIO
+5V
X1
X2
19.6608MHz
10p
10p
12K
47K
AGND
ANALOG IN
+\-5V @ 4800Hz
+5V
CLR
CLK
Q
Q
PR
D
HOLD
+5V
CLR
CLK
Q
Q
PR
D
CNVT
LS32
RD.AD
WR.D1
DRD.INT
RD
RC
AGND
330p
330p
AGND
-12V
+12V
-V
+V
OUT B
OUT C
IN B1
IN B2
IN C1
IN C2
MC-1488
+5V
2CLR
2CLK
2Q1
2Q2
2OC
2D1
2D2
ALS874

Figure 2.3.1

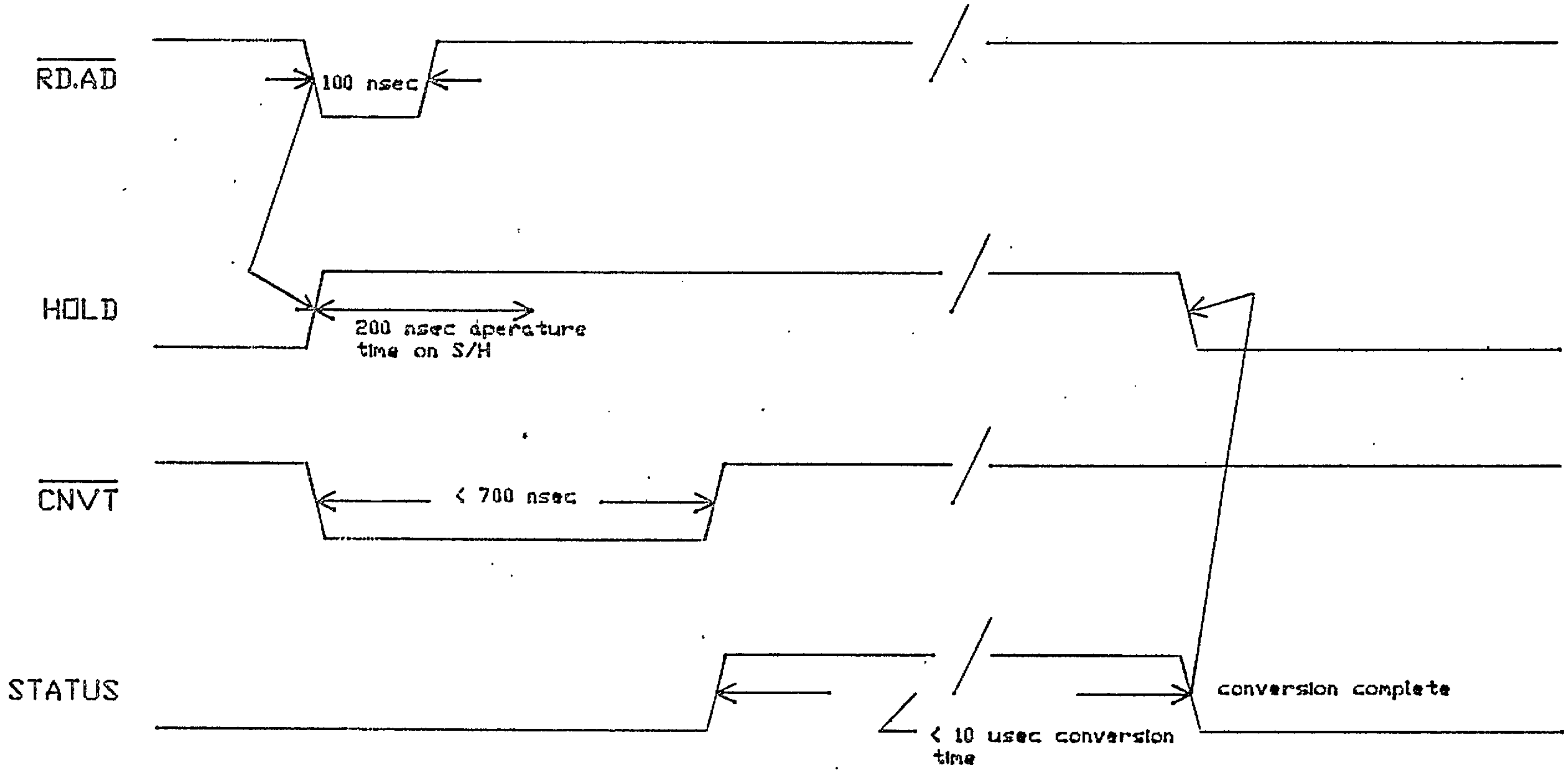

$\overline{\text{RD.AD}}$
100 nsec
HOLD
200 nsec aperature time on S/H
$\overline{\text{CNVT}}$
< 700 nsec
STATUS
conversion complete
< 10 usec conversion time

Figure 2.3.2

### Demodulator Input/ Output Ports

Output Ports    Function

2    data interface: D<1|0> = <RC|RD>

Receiver Clock, Received Data

Input Ports

1    analog to digital converter: D<15|8> = current analog input

also starts a new conversion

2    interrupt request to modulator

set interrupt request flip flop on modulator

Figure 2.3.3

## 2.4 Noise Generator

The noise generator is comprised of a low passed maximal length shift register sequence (m-sequence) driven by clock derived from clock-out on the modulator, see figure 2.4.1.

The m-sequence is 31 bits long, and is low passed (sixth order butterworth switch cap, $f_c$ = 9 kHz), and high passed ($f_c$ = 160 Hz) before being added to the transmitted signal with an arbitrary gain. Resistor $R_n$ determines the signal to noise ratio. The 25 k ohm variable resister provides a means to adjust the input level to the demodulator. The demodulator input, (noise generator output), should be set to approximately 10 Volts peak to peak.

The 31 bit m-sequence, driven by clock-out/4, provides approximately 30 minutes of noise before the m-sequence repeats. This is acceptable given the signal to noise ratios, (and hence BER), the tests will be conducted under.

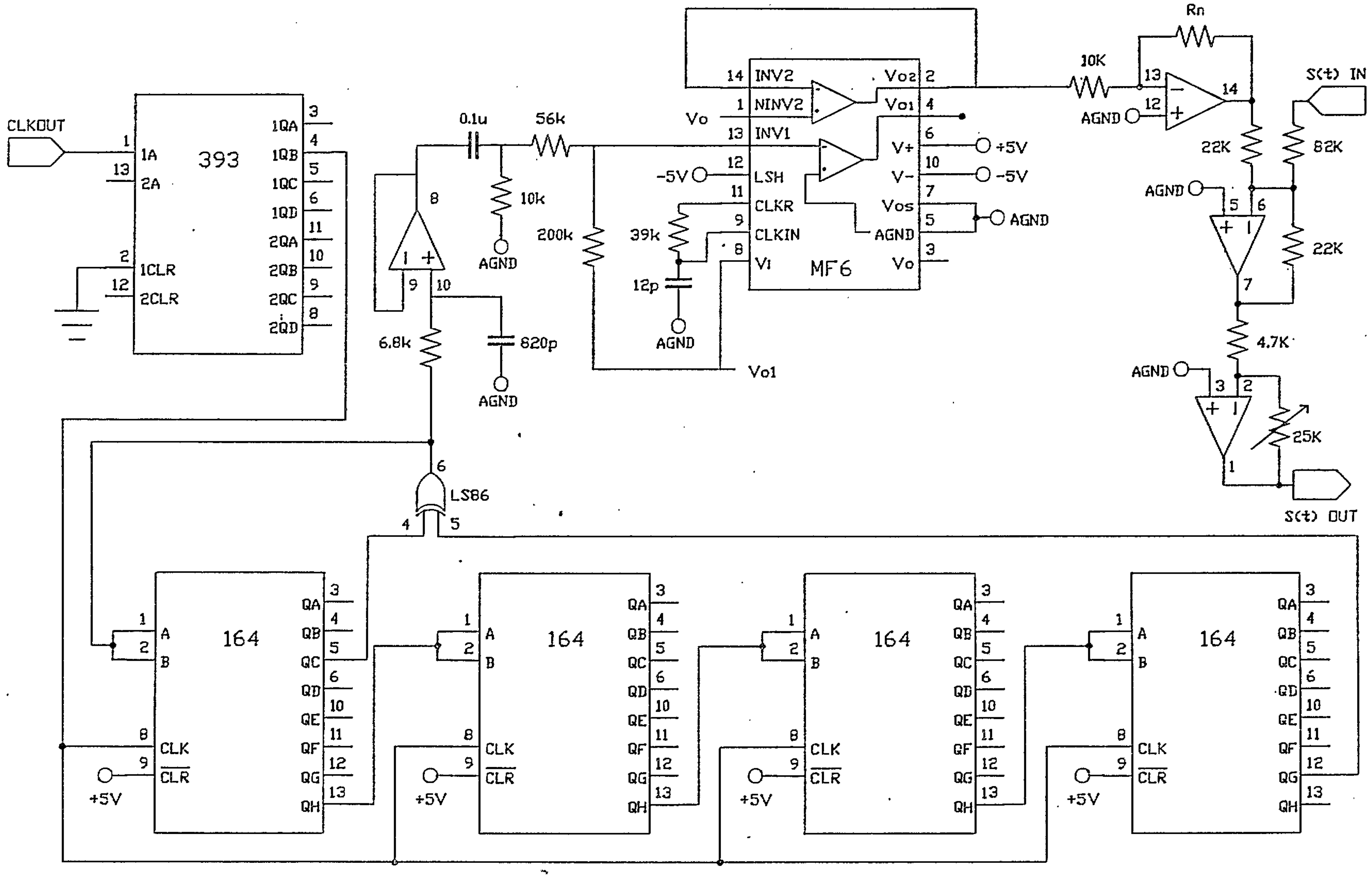

CLKOUT
393
1A
2A
1CLR
2CLR
1QA
1QB
1QC
1QD
2QA
2QB
2QC
2QD
0.1u
56k
10k
AGND
200k
39k
12p
-5V
6.8k
820p
LS86
INV2
NINV2
INV1
LSH
CLKR
CLKIN
V1
MF6
Vo2
Vo1
V+
V-
Vos
AGND
Vo
+5V
-5V
Vo1
Rn
10K
S(t) IN
22K
82K
22K
4.7K
25K
S(t) OUT
164
A
B
CLK
CLR
QA
QB
QC
QD
QE
QF
QG
QH
+5V

Figure 2.4.1

# 3. SOFTWARE IMPLEMENTATION

## 3.1 General Organization

In this section the general organization of the modem software is discussed. For a copy of the assembler listing see appendix A and B.

The modem has the functional organization shown in figure 2.1.1. It is completely full duplex, and the modulation format is GTFM(0.62,0.36). Bits are accepted from the DTE (the source), on the negative edge the transmitter clock (TC), shortly after receipt of request to send (RTS, see figure 3.1.1, 3.1.2). The modem modulates the bits using GTFM(0.62,0.36) up to a center frequency of 4.8 kHz, and provides this signal to the radio (4.8 kHz IF out). The DTE should add a short pad (≈ 8 bits), to the end of the packet, since bits are buffered, and when RTS is negated modulation, is immediately stopped. When RTS is negated, the modem enters a state in which it continually polls RTS until RTS is asserted in which case the above process is repeated.

When the received carrier is of proper format and power the modem enters the demod state in which case the received carrier is demodulated and bits are passed to the DTE, (the sink), on the rising edge of the receiver clock (RC, see figure 3.1.3, 3.1.4). When the received signal strength falls below a threshold for a period which exceeds the hangover period ($T_h$), the modem drops out of demod and into synch hunt mode where it again searches for an adequate carrier.

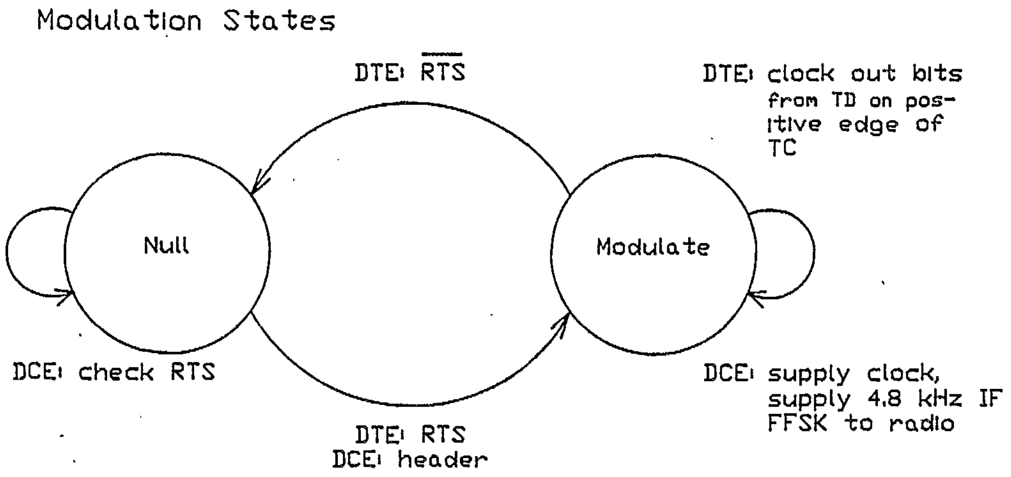

Modulation States
DTE: $\overline{RTS}$
DTE: clock out bits from TD on positive edge of TC
Null
Modulate
DCE: check RTS
DCE: supply clock, supply 4.8 kHz IF FFSK to radio
DTE: RTS
DCE: header

Figure 3.1.1

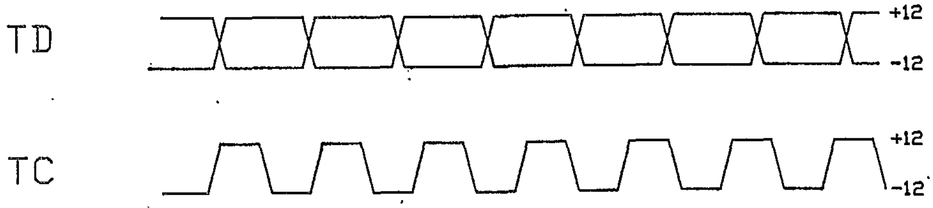


Figure 3.1.2

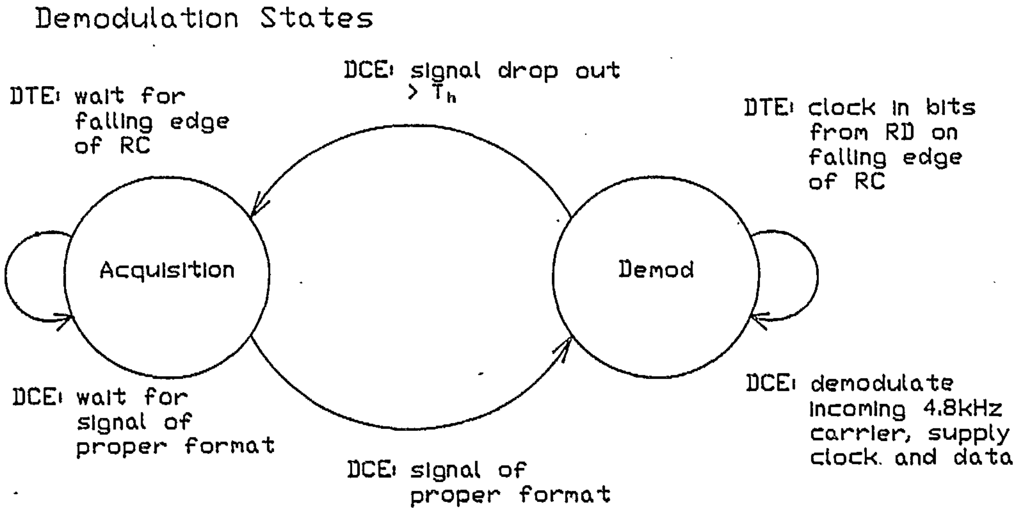

Demodulation States
DCE: signal drop out > $T_h$
DTE: wait for falling edge of RC
DTE: clock in bits from RD on falling edge of RC
Acquisition
Demod
DCE: wait for signal of proper format
DCE: demodulate incoming 4.8kHz carrier, supply clock and data
DCE: signal of proper format

Figure 3.1.3

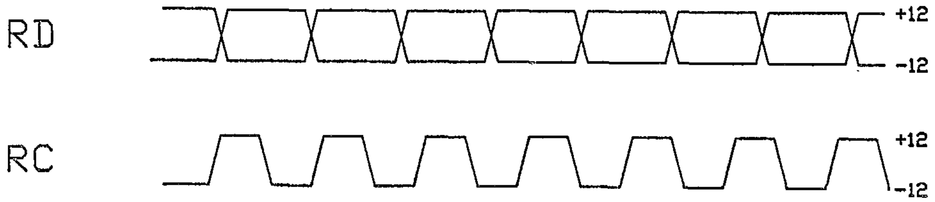


Figure 3.1.4

## 3.2 Modulation

Modulation simply consists of generating the complex baseband envelope, upconverting to a 4.8 kHz IF, and writing the real part to the digital to analog converter.

On the rising edge of RTS (request to send), the modulator outputs the header (-1,-1,1,1,...,-1-1), and enters the modulation mode. During the modulation mode, bits are read in from the data interface on the rising edge of TC (transmit clock), differentially encoded, modulated (using 4.8 kbps GTFM(0.62,0.36)), and upconverted to a 4.8 kHz IF. On the falling edge of RTS the null mode is entered in which RTS is simply polled until the rising edge of RTS is found in which case the above process is repeated, (see figure 3.1.1).

Sixteen samples per bit are used to avoid the problems associated with the 1/sinc($fT_s$) aperature. That is, with a sampling rate of 76.8 kHz, (16 samples per bit, 4800 bits per second), the droop due to the 1/sinc($f_sT$) aperature (across the 5 kHz band) is just:

$$- 10\cdot\log(\mathrm{sinc}((4.8-2.5)/76.8)/\mathrm{sinc}((4.8+2.5)/76.8))$$
$$= - 0.24\ \mathrm{dB}$$

A droop this small is negligable.

A table was constructed (see PMEM location tfmtble) which holds the baseband complex envelope for every possible starting bit pattern, assuming a starting phase of zero. To be consistent with the simulations we choose to truncate the phase pulse to five bits.

Thus, the phase path in any bit is influenced by five bits, and the starting phase (at the beginning of the bit). Our table size is just:

$$(\#\text{ of samples per bit})\cdot\exp_2(\text{length of phase pulse in bits})$$
$$= 16\cdot 2^5 = 512\text{ complex samples}$$
$$= 1024\text{ real samples}$$

By storing only the 8 most significant bits of the envelope, and by storing the real and imaginary parts in one sixteen bit word, our table size is reduced by a factor of two, (512 words of PMEM).

The pointer to the current sample $v_{n,k}$ (bit n, sample k), is:

$$\text{pntr} \leftarrow \text{table base} + \alpha_{n+2}2^0 + \alpha_{n+1}2^1 + \alpha_n 2^2 + \alpha_{n-1}2^3 + \alpha_{n-2}2^4 + k$$

Thus, by storing the past bits as follows:

$<D4|D3|D2|D1|D0> = <\alpha_{n-2}|\alpha_{n-1}|\alpha_n|\alpha_{n+1}|\alpha_{n+2}>$ in a 16 bit word, we simply need to add this quantity onto the table base, and index k samples past this point to get the current samples complex envelope. (See DMEM locations pntr, and bits+[0..2]).

Once the pointer to the first sample is calculated, 16 samples can be output before the above process is repeated. Since we only store the complex envelope for a starting phase of zero, rotation into the appropriate quadrant must be performed. This is just a rotation by 0, $\pi$, or $\pm\pi/2$.

Upconversion consists of computing:

$$\text{Re}[v_k \exp(j2\pi f_c T_s k)]$$

$$= \text{Re}[v_k q^k)$$

where

$$q = \exp(j2\pi f_c T_s)$$

$$T_s = T/md, \text{ (md = number of samples/ bit)}$$

The quantity $q^k$, is computed recursively, and periodically renormalized to unit radius. (See DMEM locations fo+[0,1], phi+[0,1] and procedure Upconv, and Normaliz).

Rotation into the appropriate quadrant is combined with upconversion. That is, $q^k$ is modified depending on the previous bits. At the start of a bit, $q^k$ is multiplied by $\alpha_{k-3}\exp(j\pi/2)$, since the influence of $\alpha_{k-3}$ will remain at $\pm\pi/2$ for the remainder of the transmission.

Once upconversion, and re-normalization is performed, the sample is output to the digital to analog converter, (DAC). Recall that since all timing is done in software, an output must be executed every 64 cycles, (exactly !).

By implementing the modulator in this way, any (reasonable) IF frequency can be accomodated, by simply modifying $q^0$, (PMEM locations consts+1 and consts+2).

## 3.3 Demodulation

It was determined in [1, section 4.3], that differential detection of GTFM with appropriate values of B and r, appropriate front end filters and a four state MLSE (VA) would offer improved performance over other incoherent detectors. However, (as is almost always the case), BER performance is improved at the expense of implementation complexity, (whether hardware or software). The MLSE is by no means simple, but the necessity of computing an Arg function is somewhat discouraging. On first inspection, our 57 point complex FIR filter will almost totally use up the available data memory on the TMS32010. (Sounds like a job for the TMS320C25). However, clever simplifications and approximations allow us to implement the modem with room (albeit small) to spare, as we will now show.

We note that even with an ordinary differential detector, we still require a complex multiply and a hard limiting operation. If we decide to do hard limiting at baseband, this requires a square root and two divisions. IF hard limiting is also possible but with an unknown degradation. Thus, even a simple differential detector (no Arg function) is not so simple. (Note that by using the Arg function hard limiting is not necessary).

During demod mode samples read in (from the analog to digital converter at 8 samples per bit) are first down converted and filtered. At this point we need only one sample per bit. Down conversion involves computing:

$$v_k = r_k \exp(j2\pi(f_d+f_c)t_k)$$

where

$f_d$ = frequency offset

$f_c$ = carrier frequency

Alternatively, (and equivalently), we compute:

$$v_k = r_k q^k$$

where

$$q = \exp(j2\pi(f_d+f_c)T/m_d)$$

$$m_d = 8$$

$$q^0 = 1$$

Thus, for every sample read in we require one half a complex multiply ($r_k$ is real), plus one full complex multiply. Also, since $q^k$ has magnitude one, and we can not truly represent this in fixed point notation, a periodic re-normalization of $q^k$ is necessary. This is done using the recursive formula:

$$q^k = q^k(3 - |q^k|^2)/2$$

Re-normalizing twice per bit should be more than adequate.

In the software implementation we store the input samples in an array of length 8, (buffer), and the derotated samples in an array of length sixteen (secbuf) with the samples stored in real followed by imaginary format. (That is secbuf+0 is the real part of the first sample, secbuf+1 is the imaginary part of the first sample, and so on). Locations phi+[0..1] and fo+[0..1] contain the current phase ($q^k$) and frequency offset measure ($\exp(j2\pi(f_d+f_c)T/m_d)$). Procedure derot handles the derotation of each sample, with auxiliary registers pointing to the two buffers. Procedure normlz handles the periodic normalization of $q^k$ (phi+[0..1]). We recall that each complex multiply (for updating $q^k$) requires 14 cycles, and that each half complex multiply requires 7 cycles resulting in an execution time of 25 cycles for procedure derot (including two cycles for the call and the return statement). Re-normalization (procedure normlz) requires twenty cycles (sixteen for execution plus two for the call and the return). Considering we must downconvert eight samples the total execution time is:

$$8(25) + 2(20) = 240 \text{ cycles}$$

Instead of downconverting, we could simply bandpass filter (with two filters with outputs which are hilbert transforms of each other), differential detect, and subtract the bias due to the frequency offset. (i.e. one cycle instead off 240 since the filtering must be done anyway). However, the center frequency of the bandpass filter should be adjusted depending on the frequency offset. This could be accomplished by multiplying the lowpass filter coefficients ($h_k$) with $q^k$ (= $\exp(j2\pi(f_d+f_c)kT/m_d)$) to obtain the desired bandpass filter. However, the filter coefficients would occupy almost the entire amount of data memory on the TMS32010 (although not on the 32020). That is, the limited data memory on the 32010 costs us approximately 240 cycles, and a fair piece of code space. Since the 32010 is our target machine, we continue with our implementation, but observe that the 32020 implementation does have its advantages.

Now that all eight samples are derotated, low pass filtering must be accomplished. From [1], we have two FIR filters, each 57 points long. There are certain advantages of having the filter length a multiple of 8, no reason the filter for the I and Q channel should be different, and no reason it should not be symetric (in the simulations). To force the filter to be symetric, identical for the I and Q channels, and of length 56 we proceed as follows. We take the sum of the I channel filter impulse response, the Q channel filter impulse response, the I channel filter impulse response reversed ($h_k = h_{57-k}$), and the Q channel filter impulse response reversed. We then linear interpolate between coefficients, and scale to obtain a symetric, 56 point FIR filter, with magnitude response shown in figure 3.3.1. The coefficients in tabular form are shown in figure 3.3.2. We note the similarity with the frequency response of the I and Q filters from [1]. The filter is now length 56, symmetric and identical for the I and Q channels. Recall that the input to the filter is at 8 times

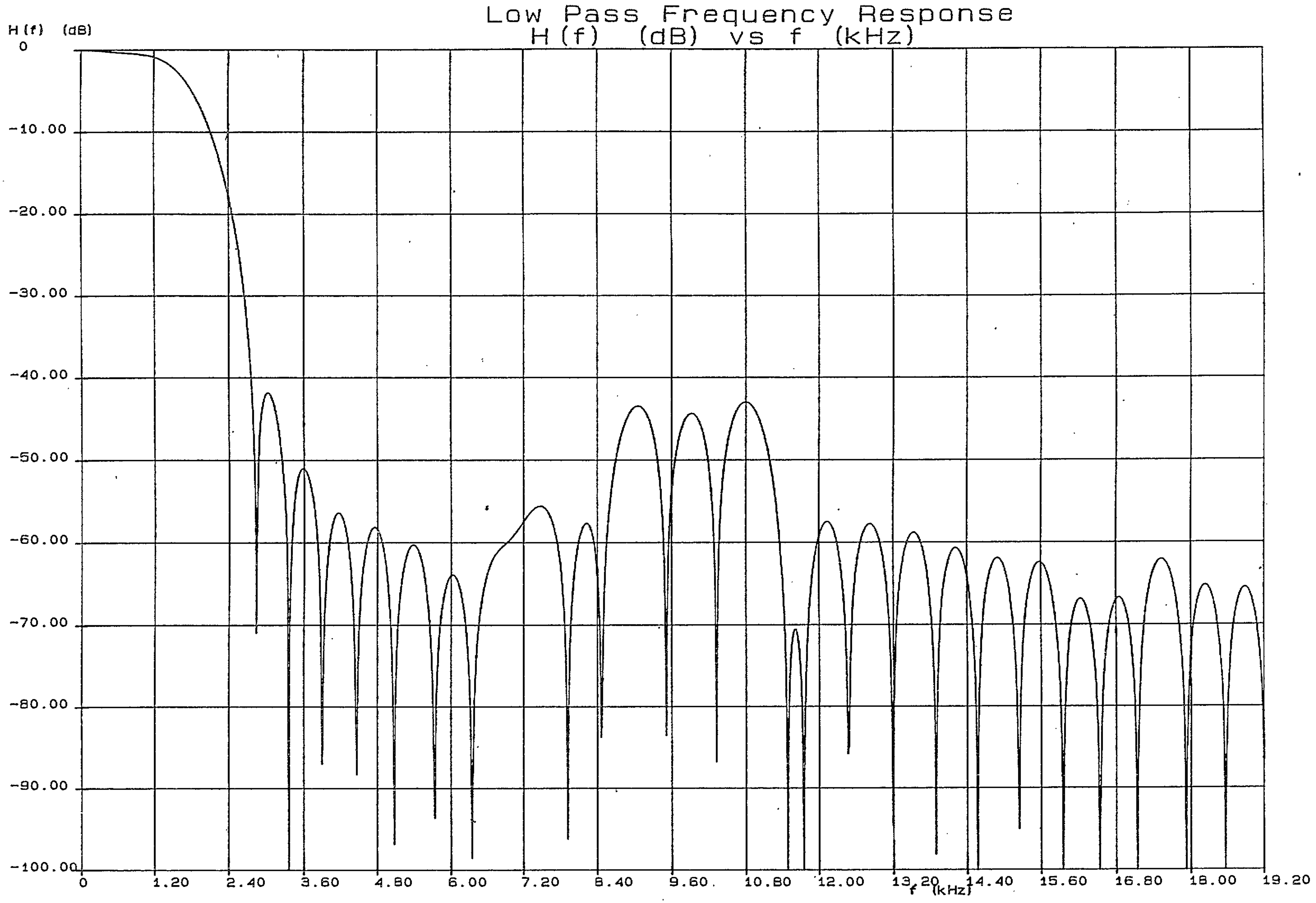

Low Pass Frequency Response
H(f) (dB) vs f (kHz)
H(f) (dB)
0
-10.00
-20.00
-30.00
-40.00
-50.00
-60.00
-70.00
-80.00
-90.00
-100.00
0
1.20
2.40
3.60
4.80
6.00
7.20
8.40
9.60
10.80
12.00
13.20
14.40
15.60
16.80
18.00
19.20
f (kHz)

Figure 3.3.1

```
                       Low Pass Filter Coefficients, (H(0) = 1.0)

 1.0916E-03  1.5688E-03  1.7319E-03  2.7169E-03  3.7650E-03  3.5335E-03  2.5511E-03  1.8285E-03

 7.9060E-04 -1.7568E-03 -5.0255E-03 -7.4927E-03 -9.6059E-03 -1.1722E-02 -1.2621E-02 -1.1203E-02

-8.0409E-03 -3.0060E-03  4.6764E-03  1.4603E-02  2.6041E-02  3.8696E-02  5.1981E-02  6.4559E-02

 7.6034E-02  8.5909E-02  9.2700E-02  9.5698E-02  9.5698E-02  9.2700E-02  8.5909E-02  7.6034E-02

 6.4559E-02  5.1981E-02  3.8696E-02  2.6041E-02  1.4603E-02  4.6764E-03 -3.0060E-03 -8.0409E-03

-1.1203E-02 -1.2621E-02 -1.1722E-02 -9.6059E-03 -7.4927E-03 -5.0255E-03 -1.7568E-03  7.9060E-04

 1.8285E-03  2.5511E-03  3.5335E-03  3.7650E-03  2.7169E-03  1.7319E-03  1.5688E-03  1.0916E-03
```

Figure 3.3.2

the bit rate and the output of the filter is at the bit rate. The structure used (to save on memory requirements) is shown in figure 3.3.3. Processing is done from right to left so the state variables are not destroyed before they are needed. The multiplication symbol implies correlation. The rightmost correlator implements:

$$\sum_{k=0}^{7}(\mathrm{secbuf}_{2k+i})p_{7-k}$$

where:

$i = 0$ for the i channel, 1 for the q channel

In order to maintain full precision the state variables (fmem) are each 32 bits long. In particular fmemih+[0..5] and fmemil+[0..5] are the high and low 16 bits of the state variables for the i channel filter. (Locations fmemqh+[0..5] and fmemql+[0..5] are the q channel state variables, (See procedure Lpf). Also, the 8 inputs to each section are secbuf+0, secbuf+2, to secbuf+14, and secbuf+1, secbuf+3, to secbuf+15 for the i and q channel filters respectively. The (complex) output of the filter is u+[0..1]. Each section can be computed in twenty (although a few sections require less) cycles, resulting in a total execution time of:

$20*7*2 \approx 280$ cycles (7 sections, and two filters)

Once the front end filtering is completed, we must compute the differential phase over one bit. We could, as in a conventional differential detector, compute:

$$\hat{y}_k = u_k u_{k-1}^*$$

where

$u_k$ is the complex low pass filter output at time k.

Then we compute:

$$y_k = \arg(\hat{y}_k).$$

This method has a slight drawback. For a $\delta P$ (dB) drop in the power of $u_k$, the power in y is decreased by $2\delta P$ (dB), which is undesireable.

Instead we compute:

$$y_k = \mathrm{Arg}(u_k) - \mathrm{Arg}(u_{k-1})$$

and ensure that $y_k$ is in $[-\pi,\pi)$

A simplified method to compute the argument for a restricted angle can be accomplished as follows [2]:

$$\mathrm{Arctan}(y/x) = (y/x)/(1 + 0.28(y/x)^2) + \epsilon$$
$$= xy/(x^2 + 0.28y^2)$$

where

$$|y/x| \leq 1$$
$$|\epsilon| \leq 5\cdot 10^{-3}$$

With the aid of this function we can compute the argument of complex numbers with angles in $[0,\pi/4]$, and by appropriate rotations can compute the argument of any complex number. In fixed point notation we can only represent numbers in $[-1,1)$ so instead of computing $\mathrm{Arg}(\cdot)$, we compute $\mathrm{Arg}(\cdot)/\pi$ which is in the range $[-1,1)$.

To compute the differential phase over a bit period we compute the difference in the current phase and the phase in the previous bit period. The difference may take on values in the range $[-2,2)$. For example, if the current phase is $3\pi/4$ and the previous phase was $-3\pi/4$, the differential phase (scaled by 1/pi) is 3/2. Representing this as a 32 bit number we obtain 49152 which is -1/2 as a 16 bit Q15 number, (the correct differential phase). Thus, if we ignore the overflow, (keep only lowest 16 bits), we obtain the actual differential phase in $[-1,1)$.

To compute the argument the complex plane is split up into eight equal size sections, $[0,\pi/4)$, $[\pi/4,\pi/2)$, $[\pi/2,3\pi/2)$, $[3\pi/2,\pi)$, $[-\pi,-3\pi/2)$, $[-3\pi/2,-\pi/2)$, $[-\pi/2,-\pi/4)$ and $[-\pi/4,0)$. Once the section is found, the low pass filter output is rotated into $[0,\pi/4)$, and the arctan is computed. The arctan computation is just a straightforward implementation of the above formula. Procedure argument requires 23 cycles for the derotation, 46 for the arctan function, 5 for scaling and accounting for the derotation, and 4 cycles for the call and return to obtain an execution time of approximately 80 cycles.

Now for the MLSE. We recall (from [1], appendix A) that the MLSE minimizes the distance:

$$\sum_{k=0}^{\infty}(y_k - C(\alpha_k, s_{k-1}))^2$$

where,

$$C(\alpha_k, s_{k-1}) = \sum_{n=0}^{L-1} f_n \alpha_{k-n}$$

where

f is the impulse reponse of the channel, (length L), and $\sum f_n = 1$

Using the above discussed Arg function, the sum of the impulse response, (the maximum differential detector output in the absence of noise), is 1/2. Thus, the output of the differential detector should be either scaled, or the VA should be modified to account for this. Also, the VA should be modified, to ensure the metrics do not exceed ±1 (Q15). Instead we minimize:

$$K_0 \cdot \sum_{k=0}^{\infty}(y_k - K_1 \cdot C(\alpha_k, s_{k-1}))^2$$

The constant $K_0$ ensures the metrics do not overflow, and $K_1$ accounts for the scaling in the differential detector. In fact, $K_0 = K_1 = 1/2$.

Proceding exactly as in [1, appendix 1] we arrive at the rules for implementing the four state MLSE, with the above scaling.

We have:

$$C(\alpha_k, s_{k-1}) = (f_0\alpha_k + f_1\alpha_{k-1} + f_2\alpha_{k-2})$$

and

$$f_0 = f_2, \; f_0 + f_1 + f_2 = 1$$

$$f_0 = 0.235, \; f_1 = 0.53$$

With four states ($N = 2^{L-1} = 4$) we must compute four metrics,

$$\mu_k^0 = \max \quad \beta_0 = \mu_{k-1}^0 + 1/2 \cdot y_k C(+1, s_{k-1}^0) - 1/8 \cdot C(+1, s_{k-1}^0)^2$$

$$\beta_1 = \mu_{k-1}^1 + 1/2 \cdot y_k C(+1, s_{k-1}^1) - 1/8 \cdot C(+1, s_{k-1}^1)^2$$

$$\mu_k^1 = \max \quad \beta_2 = \mu_{k-1}^2 + 1/2 \cdot y_k C(+1, s_{k-1}^2) - 1/8 \cdot C(+1, s_{k-1}^2)^2$$

$$\beta_3 = \mu_{k-1}^3 + 1/2 \cdot y_k C(+1, s_{k-1}^3) - 1/8 \cdot C(+1, s_{k-1}^3)^2$$

$$\mu_k^2 = \max \quad \beta_4 = \mu_{k-1}^0 + 1/2 \cdot y_k C(-1, s_{k-1}^0) - 1/8 \cdot C(-1, s_{k-1}^0)^2$$

$$\beta_5 = \mu_{k-1}^1 + 1/2 \cdot y_k C(-1, s_{k-1}^1) - 1/8 \cdot C(-1, s_{k-1}^1)^2$$

$$\mu_k^3 = \max \quad \beta_6 = \mu_{k-1}^2 + 1/2 \cdot y_k C(-1, s_{k-1}^2) - 1/8 \cdot C(-1, s_{k-1}^2)^2$$

$$\beta_7 = \mu_{k-1}^3 + 1/2 \cdot y_k C(-1, s_{k-1}^3) - 1/8 \cdot C(-1, s_{k-1}^3)^2$$

and four best paths:

$$\Gamma_k^0 = +1 \,||\, \Gamma_{k-1}^0 \quad \text{if } \beta_0 > \beta_1$$

$$+1 \,||\, \Gamma_{k-1}^1 \quad \text{else}$$

$$\Gamma_k^1 = +1 \,||\, \Gamma_{k-1}^2 \quad \text{if } \beta_2 > \beta_3$$

$$+1 \,||\, \Gamma_{k-1}^3 \quad \text{else}$$

$$\Gamma_k^2 = -1 \,||\, \Gamma_{k-1}^0 \quad \text{if } \beta_4 > \beta_5$$

$$-1 \,||\, \Gamma_{k-1}^1 \quad \text{else}$$

$$\Gamma_k^3 = -1 \,||\, \Gamma_{k-1}^2 \quad \text{if } \beta_6 > \beta_7$$

$$-1 \,||\, \Gamma_{k-1}^3 \quad \text{else}$$

but:

$$s_{k-1}^i = (\alpha_{k-1}, \alpha_{k-2})$$

$$s_{k-1}^0 = (+1, +1)$$

$$s_{k-1}^1 = (+1, -1)$$

$$s_{k-1}^2 = (-1, +1)$$

$$s_{k-1}^3 = (-1, -1)$$

and:

$$C(+1, s_{k-1}^0) = +1$$

$$C(+1, s_{k-1}^1) = f_1$$

$$C(+1, s_{k-1}^2) = 2f_0 - f_1$$

$$C(+1, s_{k-1}^3) = -f_1$$

$$C(-1,s^0_{k-1}) = f_1$$

$$C(-1,s^1_{k-1}) = f_1 - 2f_0$$

$$C(-1,s^2_{k-1}) = -f_1$$

$$C(-1,s^3_{k-1}) = -1$$

We need not concern ourselves with keeping old metrics, and old bestpaths. However, we need all current metrics in order to compute the next metrics. That is, $\mu^0_{k-1}$ is needed to compute $\mu^2_k$, and so on. Thus, the current metrics are saved, at the start of the VA. The same argument holds for computing the best paths. Specifically, locations mu+2·i (bp+2·i) holds the current metric $\mu^i$ (bestpath, $\Gamma^i$), and location mu+2·i+1 (bp+2·i+1) holds the copy. This process of saving the best paths and metrics executes in 8 cycles (using DMOV, see procedure Va).

The first metric and bestpath is then computed using the above formula. Re-normalization of the metrics is necessary to avoid overflow, and accomplished by subtracting the maximum metric, (computed below), and adding 1/2. The bestpath is kept in a 16 bit word, with the least significant bit being the most recent. The computation of the metrics comsumes approximately 22 cycles, and the need to compute 4 such metrics means an execution time of 88 cycles.

Strictly speaking, bits should not be released until all bestpaths agree on the same bit. Unfortunately, we can not wait around for bits to agree, since we require a fixed decoding delay. We could just release the most significant bit in one of the bestpaths. Better yet, we release the most significant bit in bestpath with the largest metric. To compute the maximum, differential decode, and release the bit to the sink with the clock (RC) cleared requires about 25 cycles.

Thus, the VA will execute in approximately:

$$8+88+25 \approx 120 \text{ cycles.}$$

For bit tracking and end of transmission determination we require the error between what the differential detector output should have been, given the received bits, and what the differential detector output was. The VA demodulates the bits, although there is a delay involved. Thus, the output of the differential detector must be delayed before the above decision can be made. Once the delay is accomplished, and the error ($\epsilon$) computed, the bit tracking scheme, and end of transmission determination can be done.

The error is just:

$$\begin{aligned}\epsilon &= y_k - \hat{\phi}(k,\underline{\hat{\alpha}}) \\ &= y_k - f_2\hat{\alpha}_{k-2} + f_1\hat{\alpha}_{k-1} + f_0\hat{\alpha}_k\end{aligned}$$

As discussed in [1, section 5.2], a timing error signal was derived by careful examination of the six level eye. The differential phase at the output of the differential detector can be approximated by:

$$\hat{\phi}(k,\underline{\hat{\alpha}}) = f_2\hat{\alpha}_{k-2} + f_1\hat{\alpha}_{k-1} + f_0\hat{\alpha}_k$$

If a change in sign between $\hat{\alpha}_k$ and $\hat{\alpha}_{k-2}$, then we should observe,

$$y_k = f_1\alpha_{k-1} + n_k$$

at the differential detector output.

With a small timing error, ($\tau$), we would observe:

$$y_k = f_1\alpha_{k-1} + \delta(\tau)\alpha_k + n_k$$

The timing error is just:

$$\begin{aligned}t_k &= y_k - f_1\hat{\alpha}_{k-1} \\ &= \delta(\tau)\alpha_k + n_k\end{aligned}$$

For small $\tau$, $\delta(\tau) \approx k\tau$

By averaging the $t_k$'s (in a first order LPF), and bumping the sampling phase (by just delaying, or not delaying), depending on the sign of the LPF output, an adequate timing algorithm can be done, (see procedure Stt).

To determine the end of transmission, the error magnitude ($|\epsilon|$), computed above, is averaged in a first order low pass filter. If the average error rises above a threshold, end of transmission is signalled, and synch hunt is re-entered. The filter memory and threshold determine the hang over period, $T_h$.

Bit tracking and end of transmission determination executes in about 60 cycles.

Summing the approximate execution times for the demodulator we arrive at an approximate total cycle count (per bit):

| | |
|---|---|
| 240 | derotations |
| + 280 | low pass filtering |
| + 80 | Arg function |
| + 120 | VA |
| + 60 | bit tracking, end of transmission determination |
| = 780 cycles | |

This leaves about 240 cycles for all the minor pieces of code we did not mention.

## 3.4 Acquisition

The acquisition section is concerned with determining when a signal of appropriate power and format is present at the receiver, and determining the frequency offset and initial bit phase of the incoming GTFM signal.

The acquisition section is split up into three processes, synch hunt, comb filtering, and analysis of the comb filter.

The synch hunt section is concerned with determining when a signal of appropriate format and power is present at the receiver. When the appropriate signal is deemed to be present, the signal is band pass filtered, differentially detected, and averaged over a number of bit intervals, (comb filtering). Once an adequate number of samples have been processed, the comb filter is then analysed and the initial bit clock phase and frequency offset are estimated.

The synch hunt and comb filtering block diagrams are shown in figures 3.4.1 and 3.4.2.

For the synch hunt and comb filtering sections, 8 samples per bit are used on input, with only two samples per bit at the output of the bandpass filter. This is adequate to represent the signal. Since our synchronization pre-amble is the -1,-1,+1,+1,...,-1,-1 pattern, the output of the differential detector is periodic over 4 bits.

During synch hunt, samples are read from the A/D converter at a sampling rate of 38.4 kHz. The real input samples are then bandpass filtered. The complex output is decimated down to a rate of 9.6 kHz (two samples per bit). The signal is filtered with two bandpass filters, with the outputs being hilbert transforms of each other. To create the two filters we take $h_k$, our low pass filter used in demod, and multiply the coefficients by $\exp(j2\pi f_c T_s k)$ to obtain two real, or one complex filter. The two filters have frequency responses shown in figures 3.4.3 and 3.4.4.

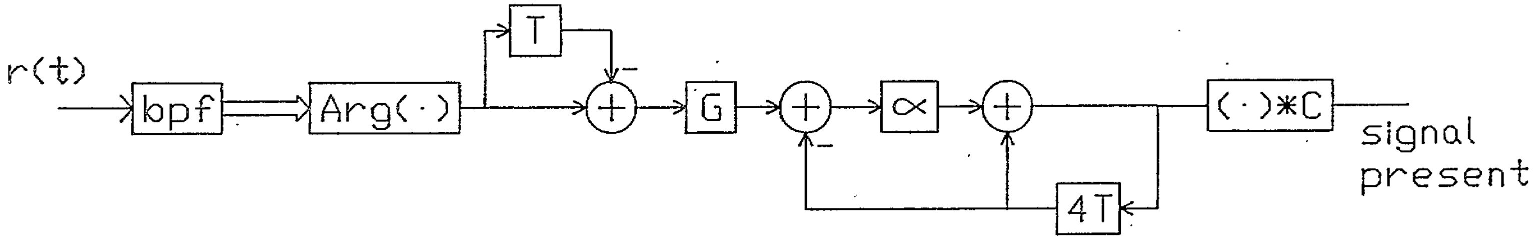


Figure 3.4.1

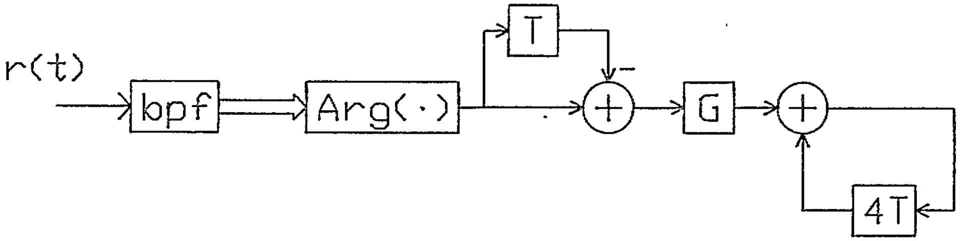


Figure 3.4.2

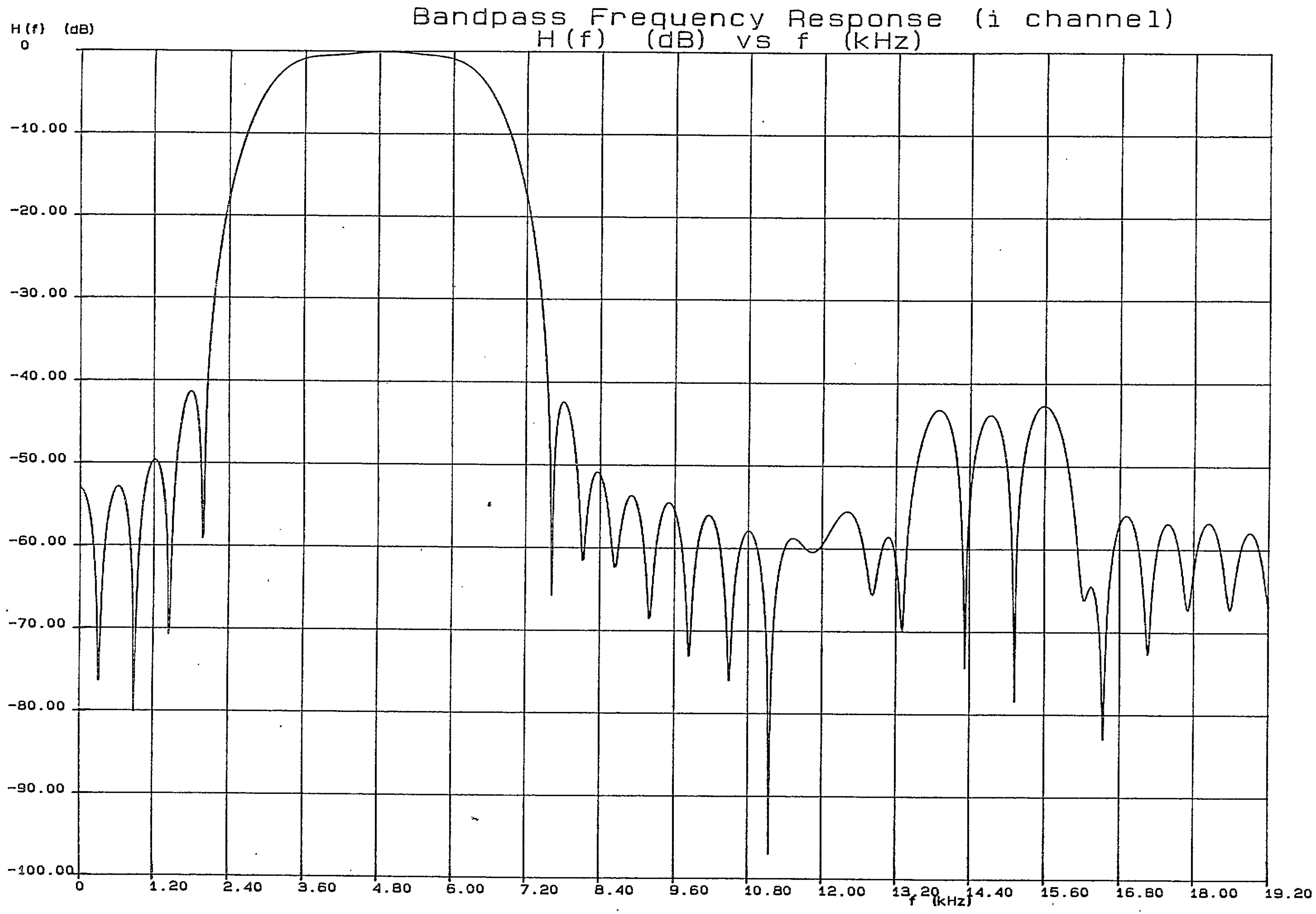

Bandpass Frequency Response (i channel)
H(f) (dB) vs f (kHz)
H(f) (dB)
0
-10.00
-20.00
-30.00
-40.00
-50.00
-60.00
-70.00
-80.00
-90.00
-100.00
0
1.20
2.40
3.60
4.80
6.00
7.20
8.40
9.60
10.80
12.00
13.20
14.40
15.60
16.80
18.00
19.20
f (kHz)

Figure 3.4.3

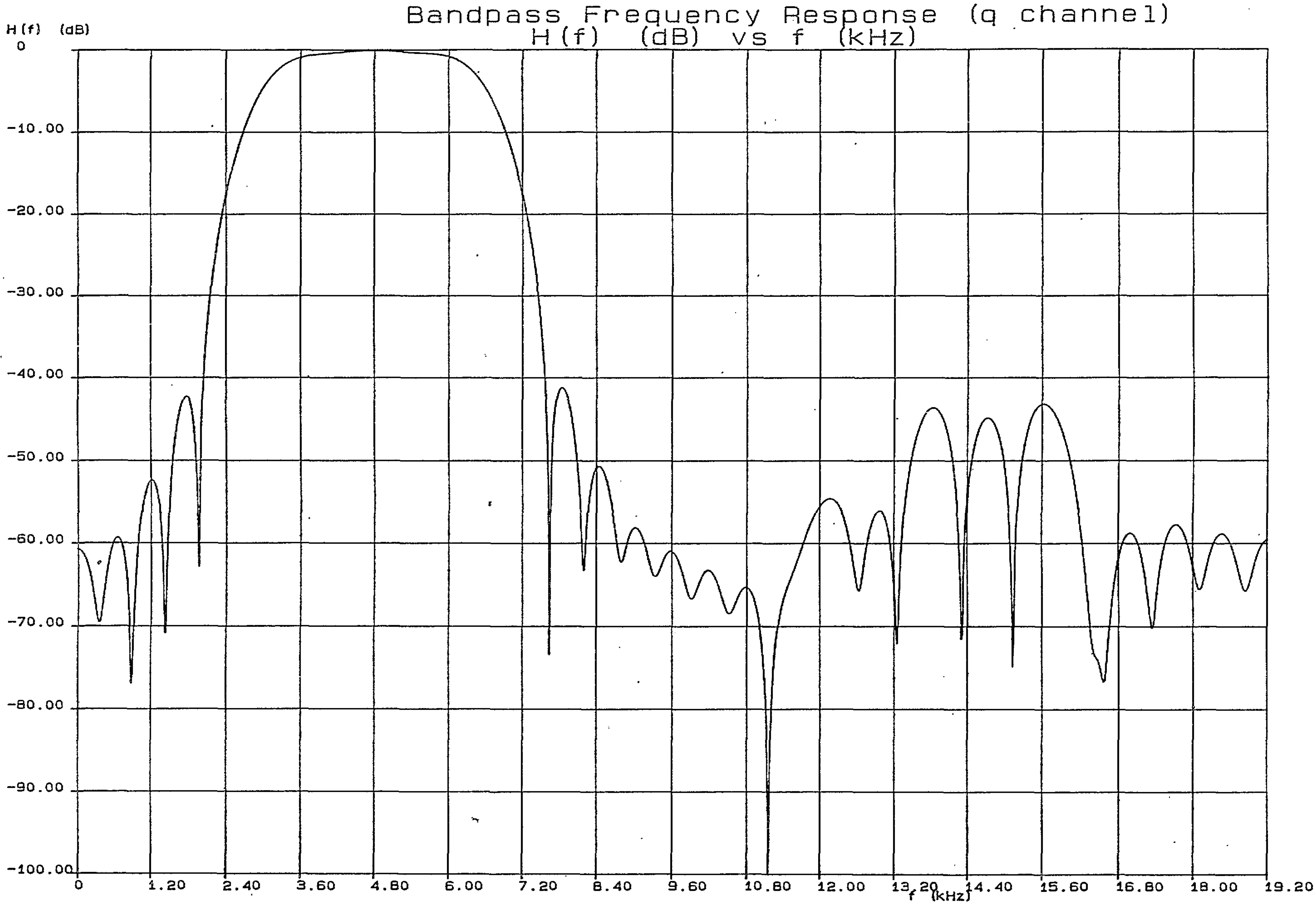

Bandpass Frequency Response (q channel)
H(f) (dB) vs f (kHz)
H(f) (dB)
0
-10.00
-20.00
-30.00
-40.00
-50.00
-60.00
-70.00
-80.00
-90.00
-100.00
0
1.20
2.40
3.60
4.80
6.00
7.20
8.40
9.60
10.80
12.00
13.20
14.40
15.60
16.80
18.00
19.20
f (kHz)

Figure 3.4.4

The difference of their phase responses is shown in figure 3.4.5. The $\pi/2$ difference in the phase shift for positive frequencies between 2.3 and 7.3 kHz, and the $-\pi/2$ phase shift for negative frequencies between -2.3 and -7.3 kHz displays the desired hilbert transformation property. The output of the bandpass filter is then differentially detected, again using the Arg function (same function as discussed in demod). If a signal of proper format is present at the receiver, the received differential phase (in the absence of noise), would be periodic over 4 bit periods, (or eight samples).

We note that in the absence of noise and distortion we receive:

$$v(t)\exp(j2\pi(f_c+f_d)t + j\phi_0)$$

where,

$v(t)$ is the transmitted complex envelope,

$f_c$ is the carrier frequency (4.8 kHz),

$f_d$ is the frequency offset and,

$\phi_0$ is an arbitrary constant phase angle in $[0,2\pi]$.

The differential phase detector output is just:

$$\mathrm{Arg}(v(t)) - \mathrm{Arg}(v(t-T)) + 2\pi(f_c+f_d)t - 2\pi(f_c+f_d)(t-T)$$

$$= \mathrm{Arg}(v(t)v^*(t-T)) + 2\pi f_dT$$

since $2\pi f_cT = 2\pi$

Thus, the presence of a frequency offset causes a level shift (a D.C. offset) in the differential detector output. The differential detector output is then low pass filtered with a first order low pass filter, and since we have periodicity over 8 samples we need 8 such filters. The comb filter is then correlated against a clean signal (that which would be received in the absence of noise), and if the correlator output exceeds a threshold the header is assumed to be present and the comb filtering section is entered.

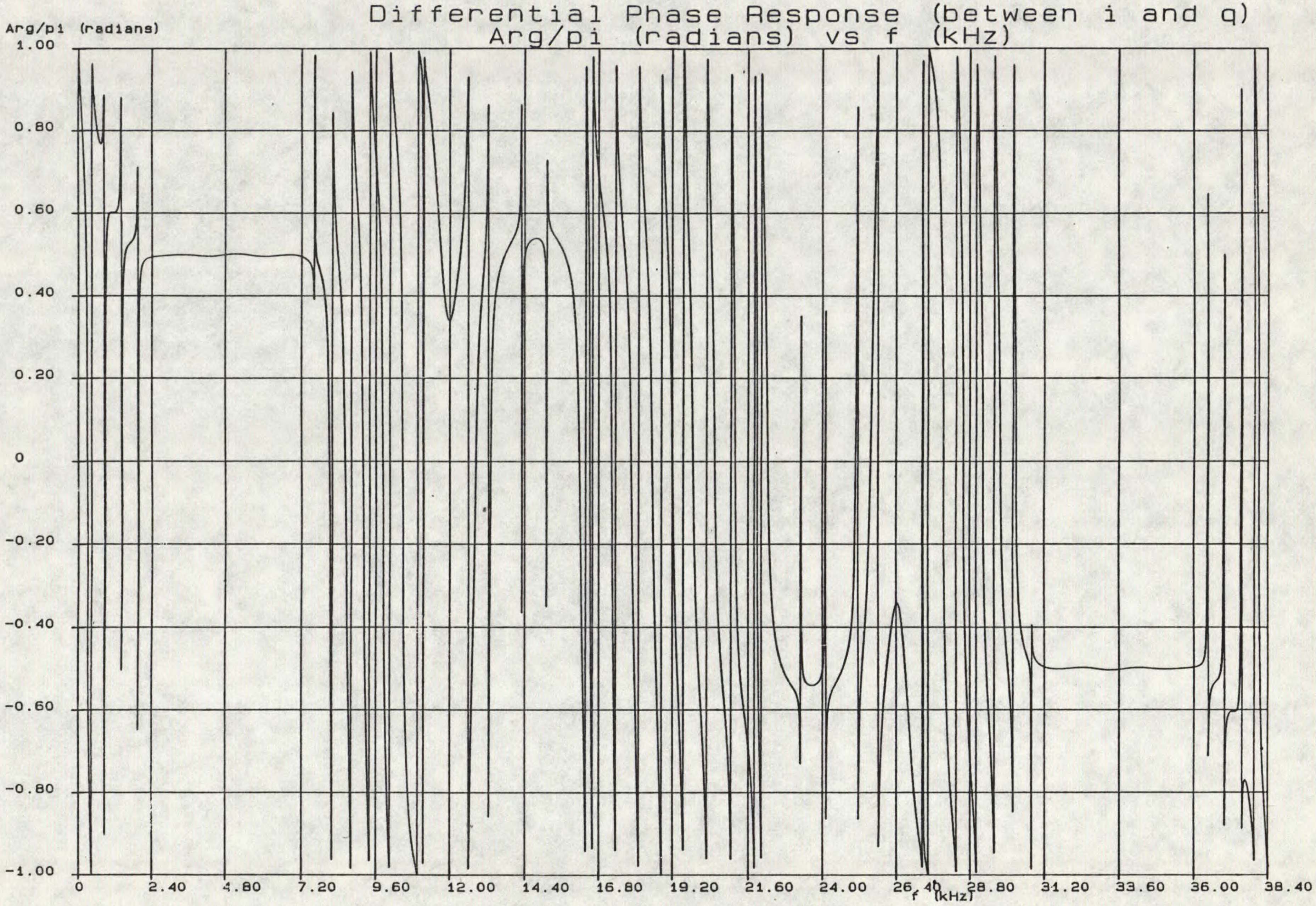

Differential Phase Response (between i and q)
Arg/pi (radians) vs f (kHz)
Arg/pi (radians)
1.00
0.80
0.60
0.40
0.20
0
-0.20
-0.40
-0.60
-0.80
-1.00
0
2.40
4.80
7.20
9.60
12.00
14.40
16.80
19.20
21.60
24.00
26.40
28.80
31.20
33.60
36.00
38.40
f (kHz)

Figure 3.4.4

The first order low pass filter memory is now extended to infinity, giving us a true averager, or true comb filter, and the low pass filter outputs are cleared. The above process, (bandpass filtering, and differential detection) and comb filtering is now performed until 128 bits have been processed. The analysis section is then entered.

Recall that the frequency offset is just the average level in the comb buffer. Thus, the first stage in the analysis section is to sum the comb buffer, to obtain $2f_dT$, (since we compute $\mathrm{Arg}(\cdot)/\pi$). During demod we require $\cos(2\pi f_dT/md)$, and $\sin(2\pi f_dT/md)$. Thus, simple scaling and a complex exponential is only required. Also for a maximum frequency offset of ±1200 Hz, we have a maximum angle of 11.25 degrees.

The zero crossing in the comb buffer, after subtracting the average level, is an indication of the position of the start of a bit. Better yet, by circularly correlating the comb buffer with a clean signal, (that which would be received in the absence of noise), and using the zero crossing of the correlator output, we obtain a better estimate of the initial bit clock. The clean signal is sampled at 8 samples per bit, to obtain better resolution of the zero crossing, (see figure 3.4.6, and 3.4.7 for a diagram and table of the correlator). Thus, the comb buffer is correlated against every fourth sample in the clean signal, the clean signal is shifted by one sample, and the process is repeated. (The actual implementation is slightly different, but the result is the same, see procedures initcorr, freqoff, and corr).

Once the location of the zero crossing is found, (that is, two samples with opposite sign), linear interpolation between the samples is used to obtain a more accurate zero crossing. We then delay the appropriate number of cycles, and demod is entered.

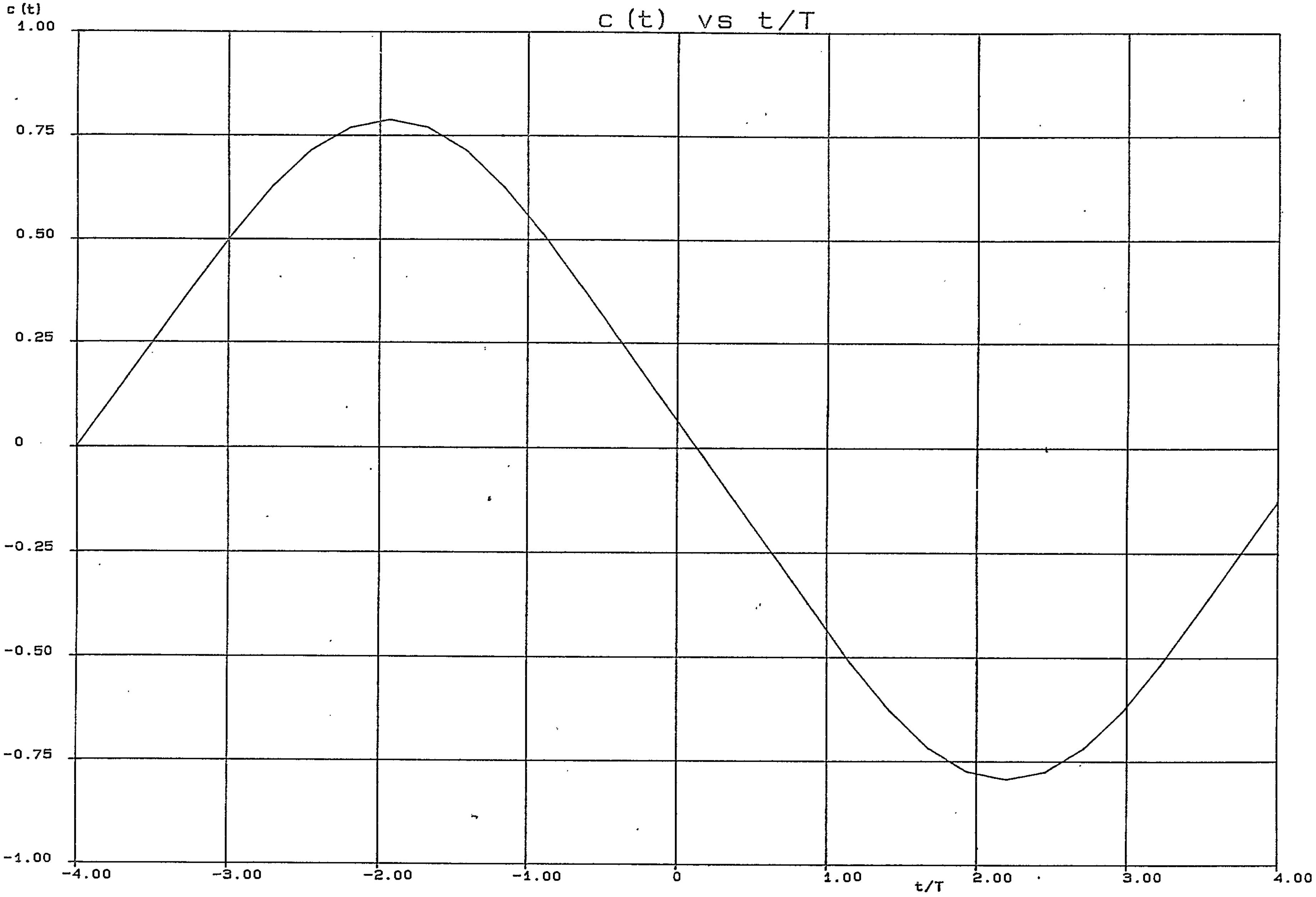

c (t)
c (t) vs t/T
1.00
0.75
0.50
0.25
0
-0.25
-0.50
-0.75
-1.00
-4.00
-3.00
-2.00
-1.00
0
1.00
2.00
3.00
4.00
t/T

Figure 3.4.6

```
                         Correlator (clean differential detector output)

  9.5210E-04  1.2849E-01  2.5803E-01  3.8892E-01  5.1554E-01  6.2793E-01  7.1545E-01  7.7045E-01

  7.8915E-01  7.7055E-01  7.1567E-01  6.2831E-01  5.1615E-01  3.8985E-01  2.5934E-01  1.3019E-01

  3.0092E-03 -1.2446E-01 -2.5450E-01 -3.8645E-01 -5.1462E-01 -6.2872E-01 -7.1775E-01 -7.7382E-01

 -7.9312E-01 -7.7458E-01 -7.1921E-01 -6.3078E-01 -5.1707E-01 -3.8906E-01 -2.5705E-01 -1.2681E-01
```

Figure 3.4.7

# 4. PERFORMANCE TESTS

## 4.1 Nature of the Tests

The circuit discussed in section 2.4 (figure 2.4.1) was used to add noise to the transmitted signal. Frequency offsets were generated in the modulator, by modifying the transmitted center frequency.

During acquisition tests, the modulator was set up to transmit the dotting pattern for three hundred bits, followed by data. Once the demodulator synchronized itself on the header, it saved the frequency offset estimate, (in program memory), and interrupted the modulator. When the modulator was interrupted, it saved the number of bits remaining in the header, delayed, and returned to transmitting the dotting pattern.

The delay ensured that during synch hunt the demodulator would be presented with noise for a short interval before the dotting pattern was transmitted.

During acquisition tests, the acquisition time (in bits) and the frequency offset estimates were collected and a mean and standard deviation calculated.

Bit Error Rate tests were conducted at signal to noise ratios of 6.4, 7.6, 8.1, 9.1, 9.8, 10.6, 11.7 and 12.3 dB using resistors ($R_n$) with values 15, 15||100 (15 in parallel with 100), 12, 12||100, 10, 10||100, 8.2, and 8.2||100 kΩ.

Both acquisition and BER performance were tested at frequency offsets of 0, ± 400, and ± 800 Hz. During aquisition tests a signal to noise ratio of 6.4 dB was used.

## 4.2 Noise Calibration

The noise was calibrated using an FFT analyzer. By computing signal power, or signal amplitude, we can calculate $E_b/N_o$.

Specifically the measured $N_o$ was -40.90 dB (Volts$^2$/Hz), and the measured signal amplitude was 3.7 Volts, (peak to peak). Thus:

$$
\begin{aligned}
E_b/N_o &= A^2/2 \cdot T/N_o \\
&= 20 \cdot \log_{10}(A) + 10 \cdot \log(T/2) - 10 \cdot \log(N_o) \\
&= 20 \cdot \log(3.7/2) - 39.82 + 40.9 \\
&= 6.4 \text{ dB}
\end{aligned}
$$

With the same noise power, and signal level, the EVM was then used to indepedently measure signal power and noise power, at the receiver. The measured values were:

$$P_s = \text{signal power} = 0.5777$$

$$P_n = \text{noise power} = 0.2736$$

Thus:

$$
\begin{aligned}
E_b/N_o &= P_s \alpha / P_n \\
&= 6.4 \text{ dB}
\end{aligned}
$$

$$\alpha_{dB} = 3.15 \text{ dB}$$

and:

$$E_b/N_o = 10 \cdot \log(P_s/P_n) + 3.15 \quad (\text{dB})$$

Now by simply measuring noise power, and signal power at the receiver we can obtain $E_b/N_o$ using the above formula.

## 4.3 Acquisition Tests

Acquisition tests were conducted at a signal to noise ratio ($E_b/N_o$) of 6.4 dB with frequency offsets of 0, ±400, and ±800 Hz. The modulator was set up to provide 500 bits of the dotting pattern followed by random data. Once the demodulator acquired the signal, it interrupted the modulator. On interrupt the modulator delayed (≈ 0.25 seconds), and the demodulator returned to synch hunt. In this way, the demodulator was always presented with noise only, prior to the occurance of the header.

The mean and variance of the measured frequency offset, was computed and is shown in figure 4.3.1. Over 1000 repeats at each frequency offset were made. Assuming a gaussian distribution of the measured frequency offsets, 95% confidence intervals were computed and are also shown in figure 4.3.1.

| Frequency Offset | 800 | 400 | 0 | -400 | -800 |
|---|---|---|---|---|---|
| Mean Error | 99.0 | 13.3 | 2.3 | -7.0 | -81.9 |
| Variance | 52.5 | 18.6 | 9.2 | 17.0 | 42.3 |
| 95% confidence interval | <205 | <51 | <21 | <41 | <167 |

Figure 4.3.1

The mean, variance, and 95% confidence intervals for the acquisition time at each $f_d$ combination are shown in figure 4.3.2.

It is apparent that at a signal to noise ratio of 6.4 dB, a header of length 275 bits is necessary to provide enough bits in the header for acquisition with frequency offsets in the range [-400, 400] Hz. Secondly, with the above conditions, the frequency offset is estimated to within ±50

Hz with more than 95% probability. A header of length 350 bits was used to maintain good performance at poor signal to noise ratios and poor frequency offsets.

With frequency offsets as large as 800 Hz the performance degrades. At a signal to noise ratio of 6.4 dB, the average error in the estimated frequency offset was 99 Hz when the actual frequency offset was 800 Hz. This bias is most likely due to the correlated noise at the output of the differential detector. By re-designing the front end filter so as to eliminate the correlation in the noise samples seperated by T seconds, it is expected that this bias would be removed, see [2].

| | | | | | |
|---|---|---|---|---|---|
| Mean Acquisition time | 291.6 | 234.9 | 228.2 | 235.9 | 286.7 |
| Variance | 35.8 | 17.6 | 16.5 | 17.5 | 32.9 |
| 95% confidence interval | <365 | <271 | <262 | <272 | <353 |

Figure 4.3.2

### 4.4 Bit Error Rate Tests

Bit error rate tests were conducted for various signal to noise ratios and at frequency offsets of 0, ±400, and ±800 Hz. Figure 4.4.1 shows the conditions and results of the tests, in tabular form. Performance was better for positive frequency offsets than for negative frequency offsets. It was concluded that the noise must not be quite flat, and in fact slopes off for larger frequencies. Instead of plotting the BER curves for positive and negative frequency offsets, we simply averaged the BER, and plotted the average. The average BER for each frequency offset magnitude is shown in figure 4.4.2. Even at frequency offsets of ±800 Hz, average BER performance has minimal degradation, from 0 Hz frequency offset.

Performance Tests

$N_e$ = number of errors

$N_b$ = number of bits/trial

12 trials at each SNR, and $f_d$

$f_d$ = 0 Hz

| $E_b/N_o$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_b$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6.4 | 2090 | 2004 | 1996 | 2170 | 2116 | 1998 | 1976 | 2032 | 2074 | 2024 | 2032 | 2126 | 100000 |
| 7.6 | 864 | 778 | 812 | 746 | 806 | 804 | 776 | 786 | 780 | 765 | 910 | 818 | 100000 |
| 8.1 | 540 | 542 | 462 | 538 | 516 | 470 | 532 | 526 | 552 | 486 | 510 | 472 | 100000 |
| 9.1 | 174 | 168 | 166 | 186 | 186 | 174 | 206 | 176 | 166 | 192 | 204 | 176 | 100000 |
| 9.8 | 160 | 140 | 132 | 116 | 152 | 136 | 184 | 172 | 164 | 136 | 178 | 158 | 200000 |
| 10.6 | 210 | 224 | 222 | 238 | 216 | 256 | 216 | 216 | 206 | 208 | 194 | 224 | 1000000 |

$f_d$ = 400 Hz

| $E_b/N_o$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_b$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6.4 | 2042 | 1940 | 2024 | 1996 | 2210 | 1990 | 2074 | 2218 | 1885 | 1918 | 2472 | 1982 | 100000 |
| 7.6 | 720 | 744 | 828 | 814 | 782 | 724 | 784 | 745 | 758 | 844 | 796 | 778 | 100000 |
| 8.1 | 548 | 478 | 430 | 484 | 454 | 520 | 494 | 460 | 416 | 490 | 510 | 516 | 100000 |
| 9.1 | 152 | 154 | 170 | 192 | 168 | 160 | 182 | 150 | 176 | 180 | 138 | 150 | 100000 |
| 9.8 | 122 | 106 | 138 | 156 | 120 | 106 | 150 | 142 | 134 | 114 | 134 | 106 | 200000 |
| 10.6 | 154 | 192 | 194 | 164 | 218 | 172 | 182 | 158 | 146 | 152 | 142 | 176 | 1000000 |

$f_d$ = 800 Hz

| $E_b/N_o$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_b$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6.4 | 2256 | 2512 | 2286 | 2338 | 3408 | 1993 | 2126 | 3730 | 2382 | 1900 | 3485 | 2002 | 100000 |
| 7.6 | 882 | 1686 | 844 | 724 | 692 | 666 | 1420 | 690 | 848 | 1042 | 820 | 988 | 100000 |
| 8.1 | 440 | 576 | 426 | 498 | 534 | 586 | 398 | 394 | 596 | 446 | 370 | 590 | 100000 |
| 9.1 | 146 | 152 | 144 | 162 | 254 | 222 | 256 | 128 | 154 | 144 | 140 | 182 | 100000 |
| 9.8 | 108 | 142 | 158 | 150 | 98 | 114 | 126 | 116 | 144 | 122 | 118 | 126 | 200000 |
| 10.6 | 176 | 164 | 166 | 180 | 222 | 212 | 188 | 204 | 186 | 176 | 286 | 348 | 1000000 |

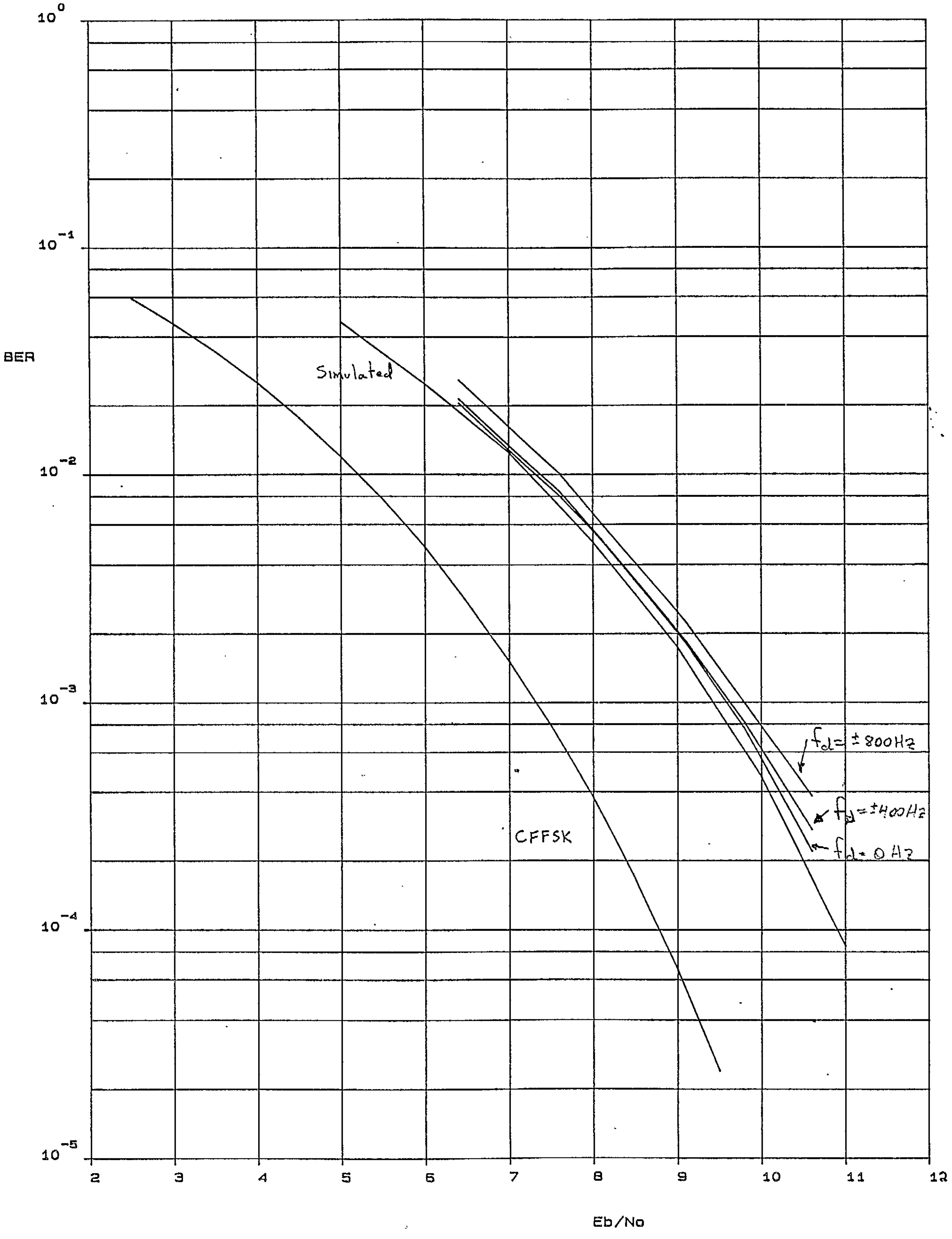
$f_d$ = -400 Hz

| $E_b/N_o$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_b$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6.4 | 2192 | 2283 | 2222 | 2154 | 2234 | 2217 | 2148 | 2376 | 2206 | 2203 | 2144 | 2122 | 100000 |
| 7.6 | 1004 | 878 | 922 | 920 | 882 | 868 | 1040 | 856 | 874 | 916 | 808 | 906 | 100000 |
| 8.1 | 580 | 576 | 530 | 566 | 564 | 536 | 494 | 540 | 566 | 530 | 570 | 542 | 100000 |
| 9.1 | 178 | 184 | 260 | 194 | 244 | 220 | 190 | 204 | 172 | 186 | 216 | 218 | 100000 |
| 9.8 | 216 | 168 | 172 | 184 | 182 | 242 | 158 | 212 | 210 | 214 | 224 | 188 | 200000 |
| 10.6 | 372 | 386 | 408 | 366 | 388 | 338 | 356 | 376 | 394 | 386 | 386 | 330 | 1000000 |

$f_d$ = -800 Hz

| $E_b/N_o$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_e$ | $N_b$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6.4 | 2406 | 3000 | 2866 | 2298 | 2716 | 2396 | 2900 | 2514 | 2166 | 2611 | 3336 | 2596 | 100000 |
| 7.6 | 992 | 910 | 964 | 1051 | 896 | 1232 | 1122 | 1037 | 1118 | 1328 | 1354 | 992 | 100000 |
| 8.1 | 812 | 708 | 562 | 610 | 964 | 762 | 870 | 701 | 694 | 716 | 636 | 642 | 100000 |
| 9.1 | 278 | 230 | 282 | 326 | 264 | 246 | 244 | 236 | 326 | 274 | 280 | 316 | 100000 |
| 9.8 | 268 | 216 | 270 | 254 | 292 | 286 | 302 | 286 | 262 | 252 | 262 | 280 | 200000 |

Figure 4.4.1



10^0
10^-1
10^-2
10^-3
10^-4
10^-5
BER
Simulated
CFFSK
fd= ±800Hz
fd=±400Hz
fd=0 Hz
2
3
4
5
6
7
8
9
10
11
12
Eb/No

Figure 4.4.2

## 5. CONCLUSIONS

This report described the implementation and performance tests of a 4800 bps Generalized Tamed Frequency Modulation, (GTFM), modem intended for the MSAT environment.

The modem was shown to have good performance for the conditions expected in the MSAT channel.

The major advantages over other modulation formats, demodulation algorithms, and implementations are as follows:

1. The performance of the modem is within 2.1 dB of coherent FFSK, and within 1.8 dB of coherent GTFM, (GTFM(0.62,0.36)). However, no phase reference is required at the demodulator resulting in fast acquisition, and relative insensitivity to the phase noise which is common in the MSAT environment.
2. The bandwidth efficiency of the modem, ($\approx$ 1 bps/Hz), means close channel spacing. Adjacent channels (with 5 kHz channel spacing) 13.5 dB more powerful than the desired channel have only a small effect on the performance, (0.4 dB).
3. The digital implementation reduces parts counts, the effects of aging, and simplifies modifications and enhancements.
4. The modem operates in burst mode, with fast acquisition, and can tolerate up to at least ±400 Hz frequency offsets in the received signals center frequency, at signal to noise ratios above 6.4 dB ($E_b/N_o$). Bit tracking is also incorporated allowing transmissions of indeterminate length.

## REFERENCES

[1] W.P. LeBlanc and J.K. Cavers, "A 4800 BPS GTFM Modem, Part One- Analysis and Simulation", under DSS Contract Number 06ST.36100-5-0088, July 1986

[2] W.P. LeBlanc and J.K. Cavers, "A 2400 BPS MSK Modem Based on Digital Signal Processing Techniques- Phase Two: Development of Engineering Prototypes", under DSS Contract Number 25ST.36100-4-4156, May 1985.

## APPENDIX A.
## Assembler Listing, Modulator

```
                                titl    'GTFM(0.62,0.36) Modulator'
                        *--
                        *-- GTFM Modem Code (modulator)
                        *--     GTFM(0.62,0.36) modem implementation using TMS32010
                        *--     digital signal processor
                        *--                                     Wilf LeBlanc
                        *--                                     December 1985-July 1986
                        *--
                        *-- Modulation Section
                        *--     - data rate of 4800 bps
                        *--     - 4.8 kHz IF
                        *--     - 19.6608 Mhz crystal driving TMS320 (1024 cycles/bit)
                        *--     - using 16 samples per bit (76.8 kHz output rate)
                        *--     - use table lookup of i and q channels (first quadrant)
                        *--       rotate into appropriate quadrant, and up to 4.8 kHz IF
                        *--
              = 0000    ar0     equ     0       auxiliary register 0 pointer
              = 0001    ar1     equ     1       auxiliary register 1 pointer
              = 0000    dp0     equ     0       data page 0, (only page used)
              = 0002    diprt   equ     2       data interface port (input, output)
              = 0001    dac     equ     1       digital to analog converter port, (output)
              = 0001    intprt  equ     1       disable interrupt flip flop, (debug) (input)
                        *--
                                dseg
      0000              mask    bss     1          data memory location holding >1F
      0001              one     bss     1          holds the constant one
      0002              aone    bss     1          holds the constant >7FFF
      0003              di      bss     2          di+0 holds the data interface input
                        *--                        di+1 holds the data interface output
      0005              bits    bss     3          bits+0 holds the five past bits
                        *--                        bits+1 holds the six past bits
                        *--                        bits+2 holds the last transmitted bit
                        *--                                  (for differential encoding)
      0008              temp    bss     1          temporary variable
      0009              phi     bss     2          cos(2pifct),sin(2pifct)
      000B              fc      bss     2          cos(2pifcT/md),sin(2pifcT/md)
      000D              pntr    bss     1          pointer to envelope
      000E              v       bss     2          transmitted complex envelope
      0010              dotbits bss     1          number of bits remaining in header
                                dend
                        *== program code, restart location =====================
                                pseg
      0000    7F81      restart dint               no interrupts in this modem (except debug)
      0001    4108              in      temp,intprt clear interrupt flip flop
      0002    7F8B              sovm               saturate, do not overflow
      0003    6880              larp    ar0        arp is ar0 by default
      0004    6E00              ldpk    dp0        page 0
                        *-- clear memory, page 0
      0005    7F89              zac
      0006    707F              lark    ar0,127
```

```
0007   5088    clr     sacl     *
0008   F400            banz     clr
0009   0007
               *-- load constants into DMEM (from PMEM)
000A   7E01            lack     1
000B   5001            sacl     one
000C   6A01            lt       one
000D   8104            mpyk     consts
000E   7F8E            pac
000F   6700            tblr     mask           mask for saving only bits 0 to 4
0010   0001            add      one
0011   670B            tblr     fc+0           center frequency, cos(2pifcT/md)
0012   0001            add      one
0013   670C            tblr     fc+1           sin(2pifcT/md)
0014   0001            add      one
0015   6702            tblr     aone           >7FFF
0016   7F89            zac
0017   500A            sacl     phi+1          phi <- (0.999969482,0.0)
0018   2002            lac      aone
0019   5009            sacl     phi+0
               *-- initially set transmit clock (TC)
001A   2201            lac      one,2
001B   5004            sacl     di+1
001C   4A04            out      di+1,diprt
               *--
               *-- Loop until RTS is asserted
               *--
               *-- check data interface for RTS (request to send) asserted...
001D   4203    waitrts  in      di+0,diprt
               *-- if RTS low, then wait...
001E   2101            lac      one,1
001F   7903            and      di+0
0020   FF00            bz       waitrts
0021   001D
               *-- else get ready to modulate data, (RTS asserted)
               *-- initialize a few DMEM variables
0022   6A01            lt       one
0023   8103            mpyk     numdot
0024   7F8E            pac
0025   6710            tblr     dotbits
0026   7E13            lack     >13
0027   5005            sacl     bits+0
0028   7E33            lack     >33
0029   5006            sacl     bits+1
               *--
               *-- Modulate data
               *--
               *-- determine bit just dropped from "bits"
002A   7E20    mdlt    lack     >20
002B   7906            and      bits+1
```

```
002C   FF00                bz        last0
002D   0034
                   *-- update current phase, if bit dropped was +1 rotate by pi/2...
002E   7F89        last1   zac
002F   100A                sub       phi+1
0030   6909                dmov      phi+0
0031   5009                sacl      phi+0
0032   F900                b         m1
0033   003A
                   *-- ...else rotate by -pi/2
0034   200A        last0   lac       phi+1
0035   6909                dmov      phi+0
0036   5009                sacl      phi+0
0037   7F89                zac
0038   100A                sub       phi+1
0039   500A                sacl      phi+1
                   *-- get pointer to table data, (depends on last five bits)
003A   2405        m1      lac       bits+0,4
003B   6A01                lt        one
003C   8108                mpyk      tfmtble
003D   7F8F                apac
003E   500D                sacl      pntr
                   *-- rotate up to center frequency and output
                   *-- sample 1
003F   F800                call      Upconv
0040   00CF
0041   7E01                lack      1
0042   F800                call      Wdeln6
0043   00FD
0044   490E                out       v+0,dac
                   *-- sample 2
0045   F800                call      Upconv
0046   00CF
0047   F800                call      Normaliz
0048   00ED
0049   7E01                lack      1
004A   F800                call      Wdeln4
004B   00FF
004C   490E                out       v+0,dac
                   *-- sample 3
004D   F800                call      Upconv
004E   00CF
004F   7E07                lack      7
0050   F800                call      Wdeln5
0051   00FE
0052   490E                out       v+0,dac
                   *-- sample 4
0053   F800                call      Upconv
0054   00CF
0055   F800                call      Normaliz
```

```
       0056   00ED
  138  0057   7E01              lack      1
  139  0058   F800              call      Wdeln4
       0059   00FF
  140  005A   490E              out       v+0,dac
  141                    *-- sample 5
  142  005B   F800              call      Upconv
       005C   00CF
  143                    *-- check RTS, clear TC
  144  005D   4203              in        di+0,diprt
  145  005E   2101              lac       one,1
  146  005F   7903              and       di+0
  147                    *-- if RTS is low, branch back
  148  0060   FF00              bz        waitrts
       0061   001D
  149                    *-- clear TC
  150  0062   7F89              zac
  151  0063   5004              sacl      di+1
  152  0064   4A04              out       di+1,diprt
  153  0065   7E04              lack      4
  154  0066   F800              call      Wdeln4
       0067   00FF
  155  0068   490E              out       v+0,dac
  156                    *-- sample 6
  157  0069   F800              call      Upconv
       006A   00CF
  158  006B   F800              call      Normaliz
       006C   00ED
  159  006D   7E01              lack      1
  160  006E   F800              call      Wdeln4
       006F   00FF
  161  0070   490E              out       v+0,dac
  162                    *-- sample 7
  163  0071   F800              call      Upconv
       0072   00CF
  164  0073   7E07              lack      7
  165  0074   F800              call      Wdeln5
       0075   00FE
  166  0076   490E              out       v+0,dac
  167                    *-- sample 8
  168  0077   F800              call      Upconv
       0078   00CF
  169  0079   F800              call      Normaliz
       007A   00ED
  170  007B   7E01              lack      1
  171  007C   F800              call      Wdeln4
       007D   00FF
  172  007E   490E              out       v+0,dac
  173                    *-- sample 9
  174  007F   F800              call      Upconv
```

```
0080  00CF
0081  7E07               lack     7
0082  F800               call     Wdeln5
0083  00FE
0084  490E               out      v+0,dac
                *-- sample 10
0085  F800               call     Upconv
0086  00CF
0087  F800               call     Normaliz
0088  00ED
0089  7E01               lack     1
008A  F800               call     Wdeln4
008B  00FF
008C  490E               out      v+0,dac
                *-- sample 11
008D  F800               call     Upconv
008E  00CF
008F  7E07               lack     7
0090  F800               call     Wdeln5
0091  00FE
0092  490E               out      v+0,dac
                *-- sample 12
0093  F800               call     Upconv
0094  00CF
0095  F800               call     Normaliz
0096  00ED
0097  7E01               lack     1
0098  F800               call     Wdeln4
0099  00FF
009A  490E               out      v+0,dac
                *-- sample 13
009B  F800               call     Upconv
009C  00CF
009D  2010               lac      dotbits
009E  FF00               bz       moddata
009F  00A7
                *-- if in header output --  ...0,0,1,1,0,0,1,1,0,0,1,1...
00A0  1001      header   sub      one
00A1  5010               sacl     dotbits
00A2  2F05               lac      bits+0,15
00A3  5803               sach     di+0
00A4  2001               lac      one
00A5  F900               b        shftadd
00A6  00AC
                *-- else get bit from data interface
00A7  2201      moddata  lac      one,2
00A8  5004               sacl     di+1
00A9  4A04               out      di+1,diprt
                *--
00AA  4203               in       di+0,diprt
```

```
                      *-- differential encode
00AB    2007                    lac       bits+2
00AC    7803          shftadd   xor       di+0
00AD    7901                    and       one
00AE    5007                    sacl      bits+2
00AF    0105                    add       bits+0,1
00B0    5006                    sacl      bits+1
00B1    7900                    and       mask
00B2    5005                    sacl      bits+0
00B3    7E01                    lack      1
00B4    F800                    call      Wdeln6
00B5    00FD
                      *--
00B6    490E                    out       v+0,dac
                      *-- sample 14
00B7    F800                    call      Upconv
00B8    00CF
00B9    F800                    call      Normaliz
00BA    00ED
00BB    7E01                    lack      1
00BC    F800                    call      Wdeln4
00BD    00FF
00BE    490E                    out       v+0,dac
                      *-- sample 15
00BF    F800                    call      Upconv
00C0    00CF
00C1    7E07                    lack      7
00C2    F800                    call      Wdeln5
00C3    00FE
00C4    490E                    out       v+0,dac
                      *-- sample 16
00C5    F800                    call      Upconv
00C6    00CF
00C7    F800                    call      Normaliz
00C8    00ED
00C9    7E01                    lack      1
00CA    F800                    call      Wdeln4
00CB    00FF
00CC    490E                    out       v+0,dac
                      *--
00CD    F900                    b         mdlt
00CE    002A
                      *== Procedure Upconv ===================================
                      *-- Upconvert
                      *--       - get gtfm envelope
                      *--       - upconvert
                      *--       - update phase
                      *-- Input:
                      *--     pntr
                      *--         pointer to gtfm envelope in program memory
```

```
                *--        envelope store as two 8 bit words in a 16 bit
                *--        program memory location. high 8 bits is the real
                *--        part of the envelope and the lower 8 bits are the
                *--        imaginary part
                *--     phi+[0..1]
                *--        current phase of the carrier
                *--     fc+[0..1]
                *--        exp(j2pifcT/md)
                *-- Output:
                *--     v+0
                *--        upconverted sample, (ready to be output to dac)
                *--     v+1
                *--        modified
                *--     phi+[0..1]
                *--        carrier phase for next sample
                *--     registers
                *--        ACC and P are modified. ar0, ar1 and arp are not
                *--        modified
                *--        on return, the T register holds phi+0 (see proc normaliz)
                *-- Other calls:
                *--        none
                *-- Cycle timing
                *--        executes in 35 cycles including call
                *--
                *-- get envelope from pmem
00CF   200D     Upconv   lac       pntr
00D0   670E              tblr      v+0
00D1   0001              add       one
00D2   500D              sacl      pntr
00D3   280E              lac       v+0,8
00D4   500F              sacl      v+1
00D5   580E              sach      v+0
00D6   280E              lac       v+0,8
00D7   500E              sacl      v+0
                *-- rotate by phi, (only need real part of output)
00D8   6A0E              lt        v+0
00D9   6D09              mpy       phi+0
00DA   7F8E              pac
00DB   6A0A              lt        phi+1
00DC   6D0F              mpy       v+1
00DD   7F90              spac
00DE   590E              sach      v+0,1
                *-- update phi, (rotate by fc), phi+1 is in t register
00DF   7F89              zac
00E0   6D0C              mpy       fc+1
00E1   7F90              spac
00E2   6A09              lt        phi+D
00E3   6D0B              mpy       fc+0
00E4   7F8F              apac
00E5   5909              sach      phi+0,1
```

```
00E6   6D0C             mpy       fc+1
00E7   7F8E             pac
00E8   6A0A             lt        phi+1
00E9   6D0B             mpy       fc+0
00EA   6C09             lta       phi+0
00EB   590A             sach      phi+1,1
               *-- return from subroutine upconv, with phi+0 in T register
00EC   7F8D             ret
               *--
               *== Procedure Normaliz ===================================
               *-- normalize
               *--      - the rotation variable, phi (complex) is normalized
               *--        to unit radius using on iteration of a recursive
               *--        formula
               *--                 phi <- phi*(3-|phi|^2)/2
               *--              or
               *--                 phi <- phi/2 + phi*(1-|phi|^2/2)
               *-- Input:
               *--     phi+[0..1]
               *--        phi (complex) contains the rotation variable
               *--        (exp(j*current phase))
               *--     aone
               *--        holds the constant >7FFF
               *--     registers
               *--        the T register holds phi+0 !!!
               *-- Output:
               *--     phi
               *--        phi is normalized using one pass through recursive routine
               *--     temp
               *--        location temp is modified
               *-- Other calls:
               *--        none
               *-- Cycle timing:
               *--        executes in 19 cycles, including call
               *--
               *-- calculate temp = 1 - |phi|^2/2
00ED   6502    Normaliz zalh      aone
00EE   6D09             mpy       phi+0
00EF   7F90             spac
00F0   6A0A             lt        phi+1
00F1   6D0A             mpy       phi+1
00F2   7F90             spac
00F3   5808             sach      temp
               *-- compute im(phi)/2 + im(phi)*temp
00F4   2E0A             lac       phi+1,14
00F5   6D08             mpy       temp
00F6   6C09             lta       phi+0
00F7   590A             sach      phi+1,1
               *-- compute re(phi)/2 + re(phi)*temp
00F8   2E09             lac       phi+0,14
```

```
  352  00F9  6D08             mpy       temp
  353  00FA  7F8F             apac
  354  00FB  5909             sach      phi+0,1
  355  00FC  7F8D             ret
  356                  *--
  357                  *== Procedure Wdeln6 =====================================
  358                  *-- delay = 6 + 3*(acc) cycles
  359  00FD  7F80      Wdeln6   nop
  360                  *== Procedure Wdeln5 =====================================
  361                  *-- delay = 5 + 3*(acc) cycles
  362  00FE  7F80      Wdeln5   nop
  363                  *== Procedure Wdeln4 =====================================
  364                  *-- delay = 4 + 3*(acc) cycles
  365  00FF  1001      Wdeln4   sub  one
  366  0100  FE00               bnz  Wdeln4
       0101  00FF
  367  0102  7F8D               ret
  368                  *--
  369                  *== end of procedures
  370                  *-- number of bits in header
  371  0103  015E      numdot   data      350
  372                  *-- mask
  373  0104  001F      consts   data      >001F
  374                  *-- carrier frequency offset (cos(2pifcT/md),sin(2pifcT/md))
  375                  *-- set for 4800 Hz IF, (modify these to change center frequency)
  376  0105  7640               data      30272,12539
       0106  30FB
  377                  *-- constant 1 (Q15), almost (0.999969482)
  378  0107  7FFF               data      >7FFF
  379                  *-- Table data for modulator, envelope stored as two 8 bit words.
  380                  *-- stored as real part in high 8 bits; im part in low 8 bits
  381                  *-- (a(k-4)...,a(k))
  382                  *-- (-1-1-1-1-1)
  383  0108  8806      tfmtble  data      >8806,>8912,>8C1D,>8F28,>9333,>993E,>A047,>A751
       0109  8912
       010A  8C1D
       010B  8F28
       010C  9333
       010D  993E
       010E  A047
       010F  A751
  384  0110  AF59               data      >AF59,>B960,>C267,>CD6D,>D871,>E374,>EE77,>FA78
       0111  B960
       0112  C267
       0113  CD6D
       0114  D871
       0115  E374
       0116  EE77
       0117  FA78
  385                  *-- (-1-1-1-1+1)
```

```
 386  0118  8806          data     >8806,>8911,>8C1D,>8F28,>9333,>993D,>9F47,>A750
      0119  8911
      011A  8C1D
      011B  8F28
      011C  9333
      011D  993D
      011E  9F47
      011F  A750
 387  0120  AF59          data     >AF59,>B860,>C267,>CC6C,>D771,>E374,>EE77,>FA78
      0121  B860
      0122  C267
      0123  CC6C
      0124  D771
      0125  E374
      0126  EE77
      0127  FA78
 388                *-- (-1-1-1+1-1)
 389  0128  8806          data     >8806,>8911,>8B1C,>8E27,>9231,>973A,>9C42,>A149
      0129  8911
      012A  8B1C
      012B  8E27
      012C  9231
      012D  973A
      012E  9C42
      012F  A149
 390  0130  A64F          data     >A64F,>AB55,>B059,>B45D,>B75F,>BA61,>BC63,>BC63
      0131  AB55
      0132  B059
      0133  B45D
      0134  B75F
      0135  BA61
      0136  BC63
      0137  BC63
 391                *-- (-1-1-1+1+1)
 392  0138  8806          data     >8806,>8911,>8B1C,>8E27,>9230,>9739,>9B42,>A149
      0139  8911
      013A  8B1C
      013B  8E27
      013C  9230
      013D  9739
      013E  9B42
      013F  A149
 393  0140  A64F          data     >A64F,>AB54,>AF59,>B35C,>B75F,>BA61,>BC63,>BC63
      0141  AB54
      0142  AF59
      0143  B35C
      0144  B75F
      0145  BA61
      0146  BC63
      0147  BC63
 394                          *-- (-1-1+1-1-1)
 395   0148   9DBD                     data      >9DBD,>9DBC,>9EBA,>A0B8,>A2B6,>A4B3,>A7B0,>AAAD
       0149   9DBC
       014A   9EBA
       014B   A0B8
       014C   A2B6
       014D   A4B3
       014E   A7B0
       014F   AAAD
 396   0150   ADAA                     data      >ADAA,>B0A7,>B3A4,>B6A2,>B8A0,>BA9E,>BC9D,>BD9D
       0151   B0A7
       0152   B3A4
       0153   B6A2
       0154   B8A0
       0155   BA9E
       0156   BC9D
       0157   BD9D
 397                          *-- (-1-1+1-1+1)
 398   0158   9DBC                     data      >9DBC,>9DBC,>9EBA,>A0B8,>A2B6,>A4B3,>A7AF,>AAAC
       0159   9DBC
       015A   9EBA
       015B   A0B8
       015C   A2B6
       015D   A4B3
       015E   A7AF
       015F   AAAC
 399   0160   ADA9                     data      >ADA9,>B0A6,>B3A4,>B6A1,>B9A0,>BA9E,>BC9D,>BD9D
       0161   B0A6
       0162   B3A4
       0163   B6A1
       0164   B9A0
       0165   BA9E
       0166   BC9D
       0167   BD9D
 400                          *-- (-1-1+1+1-1)
 401   0168   9DBC                     data      >9DBC,>9DBC,>9FBA,>A1B7,>A3B4,>A7B0,>ABAB,>B1A6
       0169   9DBC
       016A   9FBA
       016B   A1B7
       016C   A3B4
       016D   A7B0
       016E   ABAB
       016F   B1A6
 402   0170   B7A1                     data      >B7A1,>BE9C,>C697,>CF92,>D98E,>E48B,>EF89,>FA88
       0171   BE9C
       0172   C697
       0173   CF92
       0174   D98E
       0175   E48B
       0176   EF89
```

```
      0177   FA88
 403                       *-- (-1-1+1+1+1)
 404  0178   9DBC                 data       >9DBC,>9DBC,>9FBA,>A1B7,>A4B3,>A7AF,>ACAB,>B1A6
      0179   9DBC
      017A   9FBA
      017B   A1B7
      017C   A4B3
      017D   A7AF
      017E   ACAB
      017F   B1A6
 405  0180   B7A1                 data       >B7A1,>BE9B,>C797,>D092,>D98E,>E48B,>EF89,>FA88
      0181   BE9B
      0182   C797
      0183   D092
      0184   D98E
      0185   E48B
      0186   EF89
      0187   FA88
 406                       *-- (-1+1-1-1-1)
 407  0188   63BC                 data       >63BC,>63BC,>61BA,>5FB7,>5DB4,>59B0,>55AB,>4FA6
      0189   63BC
      018A   61BA
      018B   5FB7
      018C   5DB4
      018D   59B0
      018E   55AB
      018F   4FA6
 408  0190   49A1                 data       >49A1,>429C,>3A97,>3192,>278E,>1C8B,>1189,>0688
      0191   429C
      0192   3A97
      0193   3192
      0194   278E
      0195   1C8B
      0196   1189
      0197   0688
 409                       *-- (-1+1-1-1+1)
 410  0198   63BC                 data       >63BC,>63BC,>61BA,>60B7,>5DB4,>59B0,>55AB,>50A7
      0199   63BC
      019A   61BA
      019B   60B7
      019C   5DB4
      019D   59B0
      019E   55AB
      019F   50A7
 411  01A0   4AA1                 data       >4AA1,>429C,>3A97,>3193,>278F,>1C8B,>1189,>0688
      01A1   429C
      01A2   3A97
      01A3   3193
      01A4   278F
      01A5   1CBB
```

```
         01A6   1189
         01A7   0688
  412                       *-- (-1+1-1+1-1)
  413    01A8   63BC                data     >63BC,>63BC,>62BA,>60B8,>5EB6,>5CB3,>59B0,>56AD
         01A9   63BC
         01AA   62BA
         01AB   60B8
         01AC   5EB6
         01AD   5CB3
         01AE   59B0
         01AF   56AD
  414    01B0   53AA                data     >53AA,>50A7,>4DA4,>4AA2,>48A0,>469E,>449D,>449D
         01B1   50A7
         01B2   4DA4
         01B3   4AA2
         01B4   48A0
         01B5   469E
         01B6   449D
         01B7   449D
  415                       *-- (-1+1-1+1+1)
  416    01B8   63BD                data     >63BD,>63BC,>62BA,>60B9,>5FB6,>5CB3,>5AB0,>57AD
         01B9   63BC
         01BA   62BA
         01BB   60B9
         01BC   5FB6
         01BD   5CB3
         01BE   5AB0
         01BF   57AD
  417    01C0   54AA                data     >54AA,>51A7,>4DA4,>4AA2,>48A0,>469E,>449D,>449D
         01C1   51A7
         01C2   4DA4
         01C3   4AA2
         01C4   48A0
         01C5   469E
         01C6   449D
         01C7   449D
  418                       *-- (-1+1+1-1-1)
  419    01C8   7806                data     >7806,>7711,>751C,>7227,>6E31,>693A,>6442,>5F49
         01C9   7711
         01CA   751C
         01CB   7227
         01CC   6E31
         01CD   693A
         01CE   6442
         01CF   5F49
  420    01D0   5A4F                data     >5A4F,>5555,>5059,>4C5D,>495F,>4661,>4463,>4463
         01D1   5555
         01D2   5059
         01D3   4C5D
         01D4   495F
```

```
          01D5   4661
          01D6   4463
          01D7   4463
     421                    *-- (-1+1+1-1+1)
     422  01D8   7806                data      >7806,>7711,>751C,>7127,>6D31,>693A,>6442,>5F4A
          01D9   7711
          01DA   751C
          01DB   7127
          01DC   6D31
          01DD   693A
          01DE   6442
          01DF   5F4A
     423  01E0   5950                data      >5950,>5555,>5059,>4C5D,>4960,>4661,>4463,>4463
          01E1   5555
          01E2   5059
          01E3   4C5D
          01E4   4960
          01E5   4661
          01E6   4463
          01E7   4463
     424                    *-- (-1+1+1+1-1)
     425  01E8   7806                data      >7806,>7711,>741D,>7128,>6D33,>673E,>6047,>5951
          01E9   7711
          01EA   741D
          01EB   7128
          01EC   6D33
          01ED   673E
          01EE   6047
          01EF   5951
     426  01F0   5159                data      >5159,>4760,>3E67,>336D,>2871,>1D74,>1177,>0678
          01F1   4760
          01F2   3E67
          01F3   336D
          01F4   2871
          01F5   1D74
          01F6   1177
          01F7   0678
     427                    *-- (-1+1+1+1+1)
     428  01F8   7806                data      >7806,>7712,>741D,>7129,>6C34,>673E,>6048,>5951
          01F9   7712
          01FA   741D
          01FB   7129
          01FC   6C34
          01FD   673E
          01FE   6048
          01FF   5951
     429  0200   5059                data      >5059,>4761,>3D67,>336D,>2871,>1D74,>1177,>0678
          0201   4761
          0202   3D67
          0203   336D
          0204   2871
          0205   1D74
          0206   1177
          0207   0678
    430                       *--- (+1-1-1-1-1)
    431   0208   78FA                   data       >78FA,>77EE,>74E3,>71D7,>6CCC,>67C2,>60B8,>59AF
          0209   77EE
          020A   74E3
          020B   71D7
          020C   6CCC
          020D   67C2
          020E   60B8
          020F   59AF
    432   0210   50A7                   data       >50A7,>479F,>3D99,>3393,>288F,>1D8C,>1189,>0688
          0211   479F
          0212   3D99
          0213   3393
          0214   288F
          0215   1D8C
          0216   1189
          0217   0688
    433                       *-- (+1-1-1-1+1)
    434   0218   78FA                   data       >78FA,>77EF,>74E3,>71D8,>6DCD,>67C2,>60B9,>59AF
          0219   77EF
          021A   74E3
          021B   71D8
          021C   6DCD
          021D   67C2
          021E   60B9
          021F   59AF
    435   0220   51A7                   data       >51A7,>47A0,>3E99,>3393,>288F,>1D8C,>1189,>0688
          0221   47A0
          0222   3E99
          0223   3393
          0224   288F
          0225   1D8C
          0226   1189
          0227   0688
    436                       *-- (+1-1-1+1-1)
    437   0228   78FA                   data       >78FA,>77EF,>75E4,>71D9,>6DCF,>69C6,>64BE,>5FB6
          0229   77EF
          022A   75E4
          022B   71D9
          022C   6DCF
          022D   69C6
          022E   64BE
          022F   5FB6
    438   0230   59B0                   data       >59B0,>55AB,>50A7,>4CA3,>49A0,>469F,>449D,>449D
          0231   55AB
          0232   50A7
          0233   4CA3
          0234   49A0
          0235   469F
          0236   449D
          0237   449D
 439                          *-- (+1-1-1+1+1)
 440      0238   78FA                data      >78FA,>77EF,>75E4,>72D9,>6ECF,>69C6,>64BE,>5FB7
          0239   77EF
          023A   75E4
          023B   72D9
          023C   6ECF
          023D   69C6
          023E   64BE
          023F   5FB7
 441      0240   5AB1                data      >5AB1,>55AB,>50A7,>4CA3,>49A1,>469F,>449D,>449D
          0241   55AB
          0242   50A7
          0243   4CA3
          0244   49A1
          0245   469F
          0246   449D
          0247   449D
 442                          *-- (+1-1+1-1-1)
 443      0248   6343                data      >6343,>6344,>6246,>6047,>5F4A,>5C4D,>5A50,>5753
          0249   6344
          024A   6246
          024B   6047
          024C   5F4A
          024D   5C4D
          024E   5A50
          024F   5753
 444      0250   5456                data      >5456,>5159,>4D5C,>4A5E,>4860,>4662,>4463,>4463
          0251   5159
          0252   4D5C
          0253   4A5E
          0254   4860
          0255   4662
          0256   4463
          0257   4463
 445                          *-- (+1-1+1-1+1)
 446      0258   6344                data      >6344,>6344,>6246,>6048,>5E4A,>5C4D,>5950,>5653
          0259   6344
          025A   6246
          025B   6048
          025C   5E4A
          025D   5C4D
          025E   5950
          025F   5653
 447      0260   5356                data      >5356,>5059,>4D5C,>4A5E,>4860,>4662,>4463,>4463
          0261   5059
```

```
      0262  4D5C
      0263  4A5E
      0264  4860
      0265  4662
      0266  4463
      0267  4463
 448                  *-- (+1-1+1+1-1)
 449  0268  6344              data      >6344,>6344,>6146,>6049,>5D4C,>5950,>5555,>5059
      0269  6344
      026A  6146
      026B  6049
      026C  5D4C
      026D  5950
      026E  5555
      026F  5059
 450  0270  4A5F              data      >4A5F,>4264,>3A69,>316D,>2771,>1C75,>1177,>0678
      0271  4264
      0272  3A69
      0273  316D
      0274  2771
      0275  1C75
      0276  1177
      0277  0678
 451                  *-- (+1-1+1+1+1)
 452  0278  6344              data      >6344,>6344,>6146,>5F49,>5D4C,>5950,>5555,>4F5A
      0279  6344
      027A  6146
      027B  5F49
      027C  5D4C
      027D  5950
      027E  5555
      027F  4F5A
 453  0280  495F              data      >495F,>4264,>3A69,>316E,>2772,>1C75,>1177,>0678
      0281  4264
      0282  3A69
      0283  316E
      0284  2772
      0285  1C75
      0286  1177
      0287  0678
 454                  *-- (+1+1-1-1-1)
 455  0288  9D44              data      >9D44,>9D44,>9F46,>A149,>A44D,>A751,>AC55,>B15A
      0289  9D44
      028A  9F46
      028B  A149
      028C  A44D
      028D  A751
      028E  AC55
      028F  B15A
 456  0290  B75F              data      >B75F,>BE65,>C769,>D06E,>D972,>E475,>EF77,>FA78
```

```
          0291   BE65
          0292   C769
          0293   D06E
          0294   D972
          0295   E475
          0296   EF77
          0297   FA78
     457                  *-- (+1+1-1-1+1)
     458  0298   9D44            data      >9D44,>9D44,>9F46,>A149,>A34C,>A750,>AB55,>B15A
          0299   9D44
          029A   9F46
          029B   A149
          029C   A34C
          029D   A750
          029E   AB55
          029F   B15A
     459  02A0   B75F            data      >B75F,>BE64,>C669,>CF6E,>D972,>E475,>EF77,>FA78
          02A1   BE64
          02A2   C669
          02A3   CF6E
          02A4   D972
          02A5   E475
          02A6   EF77
          02A7   FA78
     460                  *-- (+1+1-1+1-1)
     461  02A8   9D44            data      >9D44,>9D44,>9E46,>A048,>A24A,>A44D,>A751,>AA54
          02A9   9D44
          02AA   9E46
          02AB   A048
          02AC   A24A
          02AD   A44D
          02AE   A751
          02AF   AA54
     462  02B0   AD57            data      >AD57,>B05A,>B35C,>B65F,>B960,>BA62,>BC63,>BD63
          02B1   B05A
          02B2   B35C
          02B3   B65F
          02B4   B960
          02B5   BA62
          02B6   BC63
          02B7   BD63
     463                  *-- (+1+1-1+1+1)
     464  02B8   9D43            data      >9D43,>9D44,>9E46,>A048,>A24A,>A44D,>A750,>AA53
          02B9   9D44
          02BA   9E46
          02BB   A048
          02BC   A24A
          02BD   A44D
          02BE   A750
          02BF   AA53
```

```
  465   02C0   AD56             data      >AD56,>B059,>B35C,>B65E,>B860,>BA62,>BC63,>BD63
        02C1   B059
        02C2   B35C
        02C3   B65E
        02C4   B860
        02C5   BA62
        02C6   BC63
        02C7   BD63
  466                     *-- (+1+1+1-1-1)
  467   02C8   88FA             data      >88FA,>89EF,>8BE4,>8ED9,>92D0,>97C7,>9BBE,>A1B7
        02C9   89EF
        02CA   8BE4
        02CB   8ED9
        02CC   92D0
        02CD   97C7
        02CE   9BBE
        02CF   A1B7
  468   02D0   A6B1             data      >A6B1,>ABAC,>AFA7,>B3A4,>B7A1,>BA9F,>BC9D,>BC9D
        02D1   ABAC
        02D2   AFA7
        02D3   B3A4
        02D4   B7A1
        02D5   BA9F
        02D6   BC9D
        02D7   BC9D
  469                     *-- (+1+1+1-1+1)
  470   02D8   88FA             data      >88FA,>89EF,>8BE4,>8ED9,>92CF,>97C6,>9CBE,>A1B7
        02D9   89EF
        02DA   8BE4
        02DB   8ED9
        02DC   92CF
        02DD   97C6
        02DE   9CBE
        02DF   A1B7
  471   02E0   A6B1             data      >A6B1,>ABAB,>B0A7,>B4A3,>B7A1,>BA9F,>BC9D,>BC9D
        02E1   ABAB
        02E2   B0A7
        02E3   B4A3
        02E4   B7A1
        02E5   BA9F
        02E6   BC9D
        02E7   BC9D
  472                     *-- (+1+1+1+1-1)
  473   02E8   88FA             data      >88FA,>89EF,>8CE3,>8FD8,>93CD,>99C3,>9FB9,>A7B0
        02E9   89EF
        02EA   8CE3
        02EB   8FD8
        02EC   93CD
        02ED   99C3
        02EE   9FB9
```

```
       02EF   A7B0
  474  02F0   AFA7             data      >AFA7,>B8A0,>C299,>CC94,>D78F,>E38C,>EE89,>FA88
       02F1   B8A0
       02F2   C299
       02F3   CC94
       02F4   D78F
       02F5   E38C
       02F6   EE89
       02F7   FA88
  475                     *-- (+1+1+1+1+1)
  476  02F8   88FA             data      >88FA,>89EE,>8CE3,>8FD8,>93CD,>99C2,>A0B9,>A7AF
       02F9   89EE
       02FA   8CE3
       02FB   8FD8
       02FC   93CD
       02FD   99C2
       02FE   A0B9
       02FF   A7AF
  477  0300   AFA7             data      >AFA7,>B9A0,>C299,>CD93,>D88F,>E38C,>EE89,>FA88
       0301   B9A0
       0302   C299
       0303   CD93
       0304   D88F
       0305   E38C
       0306   EE89
       0307   FA88
  478                          pend
  479                          end
```

```
User Defined Symbols


   AR0       0000   AR1       0001   Normaliz  00ED   PA0       0000
   PA1       0001   PA2       0002   PA3       0003   PA4       0004
   PA5       0005   PA6       0006   PA7       0007   Upconv    00CF
   Wdeln4    00FF   Wdeln5    00FE   Wdeln6    00FD   aone      0002
   ar0       0000   ar1       0001   bits      0005   clr       0007
   consts    0104   dac       0001   di        0003   diprt     0002
   dotbits   0010   dp0       0000   fc        000B   header    00A0
   intprt    0001   last0     0034   last1     002E   m1        003A
   mask      0000   mdlt      002A   moddata   00A7   numdot    0103
   one       0001   phi       0009   pntr      000D   restart   0000
   shftadd   00AC   temp      0008   tfmtble   0108   v         000E
   waitrts   001D
```

## APPENDIX B.
### Assembler Listing, Demodulator

```
                      titl    'GTFM(0.62,0.36) Demodulator
              *--
              *-- GTFM Modem Code (demodulator)
              *--     GTFM(0.62,0.36) modem implementation using the TMS32010
              *--     digital signal processor
              *--                                             Wilf LeBlanc
              *--                                             December 1985-July 1986
              *--
              *-- Demodulation Section:
              *--     - data rate of 4800 bps, 5 kHz channel spacing
              *--     - 19.6608 Mhz crystal driving TMS320 (1024 cycles/bit)
              *--     - using md = 8 samples/bit on input
              *--     - decimating down to 2 samples per bit after front end
              *--       filters (FEF) for acquisition, and down to 1 sample per bit
              *--       after FEF for demod.
              *-- Interrupts:
              *--       not used
              *-- BIO pin:
              *--       not used
              *-- Timing:
              *--       all done in software
              *-- Ports:
              *--       8 bit analog to digital converter - port 1, (D8-D15)   (input)
              *--       data interface - port 2, (RD,RC)=(D0,D1)               (output)
              *--       modulator interrupt - port 2 (debug mode)              (input)
              *-- IF interface:
              *--       nominal 4.8 kHz IF interface (fc)
              *--       can handle center frequency offset errors up to 800 Hz
              *--
              *-- Applicable documents:
              *--       A 4800 BPS GTFM Modem, Part 1 - Analysis and Simulation
              *--       A 4800 BPS GTFM Modem, Part 2 - Implementation and Performance Tests
              *--
              *-- Constants
      = 0000  ar0     equ     0       auxiliary register 0 pointer
      = 0001  ar1     equ     1       auxiliary register 1 pointer
      = 0000  dp0     equ     0       data page 0
      = 0001  adc     equ     1       analog to digital converter port (input)
      = 0002  intprt  equ     2       port for interrupting modulator (debug) (input)
      = 0002  diprt   equ     2       data interface port (input/output)
              *--
                      dseg
              *-- global memory
              *--       the following variables are used in both acquisition and demod.
              *--       one, aone, and root2 hold constants (initialized after power up).
              *--       the others hold variables used in both processes.
              *--       only page zero of data memory is used.
0000          one     bss     1       constant 1
0001          aone    bss     1       constant >7FFF = 0.999969482 (Q15) (almost 1)
0002          root2   bss     1       constant 23170 = 0.707092285 (1/sqrt(2))
```

```
0003               pmemads  bss      1         points to pmem above >800 (debug)
0004               temp     bss      1         temporary location
0005               buffer   bss      8         buffer for holding input a/d converter values
000D               z        bss      2         complex variable used in argument function (re, im)
000F               u        bss      2         complex front end filter output (re, im)
0011               arg      bss      3         used in arg function and differential detection
0014               y        bss      1         output of differential detector
0015               di       bss      2         input/output from/to data interface
                   *--
       = 0017      dmdorg   equ      $
                   *-- data memory for acquisition
                   *--         acquisition memory is split up into 3 sections, one global
                   *--         section (the comb filter), one section for synch hunt and
                   *--         estimation, and one section for the analysis process.
                   *--         the latter two sections are overlapping
0017               comb     bss      16        the comb filter buffer
                   *--
       = 0027      cororg   equ      $
                   *-- data memory for the synch hunt and estimation section of acquisition
0027               afmemih  bss      13        state variables for filter
0034               afmemil  bss      13
0041               afmemqh  bss      13
004E               afmemql  bss      13
005B               scorr    bss      8         short correlator (clean signal)
0063               alpha    bss      2         comb memory, and gain
0065               thresh   bss      1         threshold for acquisition
                   *-- data memory for analysis section of acquisition
0027                        dorg     cororg
0027               lcorr    bss      32        long correlator (clean signal)
0047               tcorr    bss      3         correlator output, and zero crossing
004A               corrmax  bss      1         max correlator output
004B               corrthr  bss      1         correlator threshold
004C               zcflag   bss      1         zero crossing flag
004D               count    bss      2         loop counter, zero crossing pointer
004F               sin      bss      2         for sine
0051               cos      bss      2             and cosine routine
                   *-- data memory for demod
                   *--      one section is needed for demod.
                   *--      several data memory locations hold constants (fo, c, dmdthr, and
                   *--      f). the others are variables.
                   *--
0017                        dorg     dmdorg
0017               phi      bss      2         the current phase (exp(jphi)) of the carrier (re,im)
0019               fo       bss      2         the frequency offset and center frequency  (re,im)
                   *--                                           (exp(j2pi(fo+fc)/md))
001B               c        bss      4         constants for MLSE
001F               lbits    bss      3         last received bits (+/- 32767)
0022               dmdthr   bss      1         threshold for hang over period
0023               f        bss      2         constants for hang over period determination
                   *--                             and bit tracking
```

```
0025           dfmemih  bss      6          filter state variables
002B           dfmemil  bss      6
0031           dfmemqh  bss      6
0037           dfmemql  bss      6
003D           secbuf   bss      16         down converted samples
004D           mumax    bss      1          MLSE variables
004E           mu       bss      8
0056           bp       bss      8
005E           recd     bss      2          received and last received bit
0060           ly       bss      15         last 15 outputs of differential detector
006F           yavg     bss      1          low pass error output (for hang over period)
0070           tsum     bss      2          timing error averager (first order lpf)
                        dend
               *-- program start
                        pseg
0000   7F81    restart  dint                        no interrupts in this modem !!!
0001   F800             call     Initglob           initialize global constants, and clear mem
0002   03FB
               *--
               *///////////////////////////////
               *-- Synch hunt section
               *--      - front end filter (FEF)
               *--      - differential detect (DD)
               *--      - comb filter (CF)
               *--      - correlate against clean signal
               *--      - threshold
               *--
0003   F800    synch    call     Initsync           initialize constants for synch hunt and clear mem
0004   0410
               *--
               *-- FEF, DD and CF
0005   F800    syncstrt call     Bpfilt
0006   0106
0007   4109             in       buffer+4,adc
               *-- correlate and threshold
0008   F800             call     Hcorr
0009   0375
000A   FC00             bgz      setcomb
000B   0012
               *-- .delay
000C   7E1E             lack     30                 delay
000D   F800             call     Wdeln5
000E   064A
000F   410B             in       buffer+6,adc
0010   F900             b        syncstrt
0011   0005
               *--
               *///////////////////////////////
               *-- Comb Filtering section
               *--      - set comb filter memory to infinity (true averager)
```

```
              *--      - clear comb
              *--      - repeat above process for 128 bits
              *--      - the delay before the next call to Bpfilt (below, at label combfilt)
              *--        is the same as the time delay which would have elapsed before the
              *--        next call to Bpfilt (above, at label syncstrt) if the threshold had
              *--        not been met. thus, the sampling period remains constant and the
              *--        differential detector output is valid immediately.
              *--
              *-- set comb memory to infinity, clear comb filter
0012   F800   setcomb  call      Initcomb
0013   038E
0014   410B            in        buffer+6,adc
0015   7F80            nop
0016   70FF            lark      ar0,>FF
              *-- now repeat synch hunt code for 128 bits (256 samples)
              *-- (recall comb memory is infinity, and no correlation or thresholding
              *-- is done here)
0017   F800   combfilt call      Bpfilt
0018   0106
0019   4109            in        buffer+4,adc
001A   F800            call      Eshft
001B   0288
001C   7E23            lack      35             delay
001D   F800            call      Wdeln6
001E   0649
001F   410B            in        buffer+6,adc
0020   F400            banz      combfilt
0021   0017
              *--
              *////////////////////////////
              *-- Analysis section
              *--      - calculate frequency offset and bit phase from comb buffer
              *-- initialize correlator (long), (and other variables)
0022   F800            call      Initcorr
0023   039E
              *-- estimate frequency offset and bit phase, delay till start of bit
0024   F800            call      Freqoff
0025   0293
              *-- if max correlator output is not large enough, no signal present
0026   FC00            bgz       synch
0027   0003
              *--
              *////////////////////////////
              *-- Demodulation section
              *--   - down convert from 4.8 kHz IF
              *--   - filter (i and q channels)
              *--   - differential detection
              *--   - Maximum Likelihood Sequence Estimation
              *--   - bit tracking
              *--
              *-- clear memory, from dfmemih to tsum+1
              *-- initialize constants
0028  F800              call      Initdmd
0029  03D2
              *--
              *-- derotate input samples
              *-- includes down conversion plus frequency offset correction
002A  7005    demod     lark      ar0,buffer+0
002B  713D              lark      ar1,secbuf+0
002C  F800              call      Derot
002D  0272
002E  F800              call      Derot
002F  0272
0030  F800              call      Derot
0031  0272
0032  F800              call      Derot
0033  0272
0034  F800              call      Normlz
0035  0546
0036  4105              in        buffer+0,adc
0037  F800              call      Derot
0038  0272
0039  F800              call      Derot
003A  0272
003B  F800              call      Derot
003C  0272
003D  F800              call      Derot
003E  0272
003F  F800              call      Normlz
0040  0546
              *-- low pass filter, both I and Q channels
0041  F800              call      Lpf
0042  0429
              *-- compute current phase
0043  F800              call      Argument
0044  005D
              *-- differential detect
              *-- (compute differential phase over one bit period)
              *--       - we want the differential phase over one bit period,
              *--         thus, we take the difference between the current phase
              *--         and the last phase.
              *--       - however, this would yield (arg) values in the range [-2..2]
              *--         (recall the Argument function returns Arg/pi)
              *--       - by taking the difference, and throwing away the overflow
              *--         we obtain the desired result
0045  2011              lac       arg+0
0046  1012              sub       arg+1
0047  5014              sacl      y              forget about high order ACC
0048  6911              dmov      arg+0          differential delay
0049  7E03              lack      3              delay
```

```
   222  004A  F800         call      Wdeln6
        004B  0649
   223                *-- read in sample 4
   224  004C  4109         in        buffer+4,adc
   225                *-- Maximum likelihood sequence estimation (Viterbi Algorithm)
   226  004D  F800         Call      Va
        004E  0596
   227  004F  410A         in        buffer+5,adc           sample 5
   228                *-- delay before next sample is read
   229  0050  7E28         lack      40             delay
   230  0051  F800         call      Wdeln5
        0052  064A
   231  0053  410B         in        buffer+6,adc           sample 6
   232                *-- bit timing recovery algorithm
   233  0054  F800         call      Stt
        0055  0557
   234  0056  410C         in        buffer+7,adc
   235                *-- check to see if transmission is finished
   236  0057  206F         lac       yavg
   237  0058  1022         sub       dmdthr
   238  0059  FA00         blz       demod
        005A  002A
   239                *-- transmission is finished, loop back to synch hunt mode
   240  005B  F900         b         synch
        005C  0003
   241                *--
   242                *-- Include Files
   243                *-- bandpass filter, and lowpass filter definitions (equate statements)
   244                *--
   245                     page
```

```
                        copy      h.def
                   *-- low pass coefficients
      = 0024  p0        equ       36
      = 0033  p1        equ       51
      = 0039  p2        equ       57
      = 0059  p3        equ       89
      = 007B  p4        equ       123
      = 0074  p5        equ       116
      = 0054  p6        equ       84
      = 003C  p7        equ       60
      = 001A  p8        equ       26
      = FFC6  p9        equ       -58
      = FF5B  p10       equ       -165
      = FF0A  p11       equ       -246
      = FEC5  p12       equ       -315
      = FE80  p13       equ       -384
      = FE62  p14       equ       -414
      = FE91  p15       equ       -367
      = FEF9  p16       equ       -263
      = FF9D  p17       equ       -99
      = 0099  p18       equ       153
      = 01DF  p19       equ       479
      = 0355  p20       equ       853
      = 04F4  p21       equ       1268
      = 06A7  p22       equ       1703
      = 0843  p23       equ       2115
      = 09BB  p24       equ       2491
      = 0AFF  p25       equ       2815
      = 0BDE  p26       equ       3038
      = 0C40  p27       equ       3136
      = 0C40  p28       equ       3136
      = 0BDE  p29       equ       3038
      = 0AFF  p30       equ       2815
      = 09BB  p31       equ       2491
      = 0843  p32       equ       2115
      = 06A7  p33       equ       1703
      = 04F4  p34       equ       1268
      = 0355  p35       equ       853
      = 01DF  p36       equ       479
      = 0099  p37       equ       153
      = FF9D  p38       equ       -99
      = FEF9  p39       equ       -263
      = FE91  p40       equ       -367
      = FE62  p41       equ       -414
      = FE80  p42       equ       -384
      = FEC5  p43       equ       -315
      = FF0A  p44       equ       -246
      = FF5B  p45       equ       -165
      = FFC6  p46       equ       -58
      = 001A  p47       equ       26
```

```
        = 003C    p48      equ      60
        = 0054    p49      equ      84
        = 0074    p50      equ      116
        = 007B    p51      equ      123
        = 0059    p52      equ      89
        = 0039    p53      equ      57
        = 0033    p54      equ      51
        = 0024    p55      equ      36
                  *-- bandpass coefficients (real part)
        = FFDC    pi0      equ      -36
        = FFDC    pi1      equ      -36
        = 0000    pi2      equ      0
        = 003F    pi3      equ      63
        = 007B    pi4      equ      123
        = 0052    pi5      equ      82
        = 0000    pi6      equ      0
        = FFD6    pi7      equ      -42
        = FFE6    pi8      equ      -26
        = 0029    pi9      equ      41
        = 0000    pi10     equ      0
        = FF52    pi11     equ      -174
        = FEC5    pi12     equ      -315
        = FEF0    pi13     equ      -272
        = 0000    pi14     equ      0
        = 0104    pi15     equ      260
        = 0107    pi16     equ      263
        = 0046    pi17     equ      70
        = 0000    pi18     equ      0
        = 0152    pi19     equ      338
        = 0355    pi20     equ      853
        = 0381    pi21     equ      897
        = 0000    pi22     equ      0
        = FA28    pi23     equ      -1496
        = F645    pi24     equ      -2491
        = F839    pi25     equ      -1991
        = 0000    pi26     equ      0
        = 08A9    pi27     equ      2217
        = 0C40    pi28     equ      3136
        = 0864    pi29     equ      2148
        = 0000    pi30     equ      0
        = F91E    pi31     equ      -1762
        = F7BD    pi32     equ      -2115
        = FB4C    pi33     equ      -1204
        = 0000    pi34     equ      0
        = 025B    pi35     equ      603
        = 01DF    pi36     equ      479
        = 006C    pi37     equ      108
        = 0000    pi38     equ      0
        = 00BA    pi39     equ      186
        = 016F    pi40     equ      367
        = 0124    pi41     equ       292
        = 0000    pi42     equ       0
        = FF21    pi43     equ       -223
        = FF0A    pi44     equ       -246
        = FF8C    pi45     equ       -116
        = 0000    pi46     equ       0
        = FFEE    pi47     equ       -18
        = FFC4    pi48     equ       -60
        = FFC5    pi49     equ       -59
        = 0000    pi50     equ       0
        = 0057    pi51     equ       87
        = 0059    pi52     equ       89
        = 0028    pi53     equ       40
        = 0000    pi54     equ       0
        = FFE7    pi55     equ       -25
                  *-- band pass coefficients (im part)
        = 0000    pq0      equ       0
        = FFDC    pq1      equ       -36
        = FFC7    pq2      equ       -57
        = FFC1    pq3      equ       -63
        = 0000    pq4      equ       0
        = 0052    pq5      equ       82
        = 0054    pq6      equ       84
        = 002A    pq7      equ       42
        = 0000    pq8      equ       0
        = 0029    pq9      equ       41
        = 00A5    pq10     equ       165
        = 00AE    pq11     equ       174
        = 0000    pq12     equ       0
        = FEF0    pq13     equ       -272
        = FE62    pq14     equ       -414
        = FEFC    pq15     equ       -260
        = 0000    pq16     equ       0
        = 0046    pq17     equ       70
        = FF67    pq18     equ       -153
        = FEAE    pq19     equ       -338
        = 0000    pq20     equ       0
        = 0381    pq21     equ       897
        = 06A7    pq22     equ       1703
        = 05D8    pq23     equ       1496
        = 0000    pq24     equ       0
        = F839    pq25     equ       -1991
        = F422    pq26     equ       -3038
        = F757    pq27     equ       -2217
        = 0000    pq28     equ       0
        = 0864    pq29     equ       2148
        = 0AFF    pq30     equ       2815
        = 06E2    pq31     equ       1762
        = 0000    pq32     equ       0
        = FB4C    pq33     equ       -1204
```

```
  396       = FB0C    pq34     equ      -1268
  397       = FDA5    pq35     equ      -603
  398       = 0000    pq36     equ      0
  399       = 006C    pq37     equ      108
  400       = FF9D    pq38     equ      -99
  401       = FF46    pq39     equ      -186
  402       = 0000    pq40     equ      0
  403       = 0124    pq41     equ      292
  404       = 0180    pq42     equ      384
  405       = 00DF    pq43     equ      223
  406       = 0000    pq44     equ      0
  407       = FF8C    pq45     equ      -116
  408       = FFC6    pq46     equ      -58
  409       = 0012    pq47     equ      18
  410       = 0000    pq48     equ      0
  411       = FFC5    pq49     equ      -59
  412       = FF8C    pq50     equ      -116
  413       = FFA9    pq51     equ      -87
  414       = 0000    pq52     equ      0
  415       = 0028    pq53     equ      40
  416       = 0033    pq54     equ      51
  417       = 0019    pq55     equ      25
  418                 *--
  419                 *--
  420                 *-- Procedures (in alphabetical order)
  421                 *--
  422                          page
```

```
                              copy      argument.prc
                    *--
                    *== Procedure Argument ===================================
                    *-- Argument of a complex number in [-pi,pi]
                    *--      - value returned is actually Arg(u)/pi
                    *--      - rotate complex number into [0,pi/4]
                    *--      - compute arctangent in [0,pi/4] using formula:
                    *--           arctan(y/x) = xy/(x^2 + 0.28y^2) + epsilon
                    *--           where |epsilon| < 5E-03
                    *--      - scale and rotation value to get arg(u)/pi
                    *-- Input:
                    *--   Variables
                    *--     u+[0..1]
                    *--        complex number (re,im), we want arg(u)/pi
                    *--     root2
                    *--        holds the constant sqrt(2)/2 = 0.707106781 (Q15)
                    *-- Output:
                    *--     arg+[0]
                    *--        contains arg(u)/pi
                    *--     temp
                    *--        location temp is modified
                    *--     z+[0,1]
                    *--        locations z+0, and z+1 are modified
                    *--        (holds rotated "u" value, Arg(z) is in [0,pi/4])
                    *--     registers
                    *--        ACC, P, and T registers are modified
                    *--        AR0, AR1, and ARP are not modified
                    *-- Other calls:
                    *--        calls to delay procedures are made
                    *--        two elements must be available on stack
                    *--        before argument is called
                    *-- Cycle timing:
                    *--     executes in 78 cycles (including call)
                    *--
 005D   6A02        Argument lt       root2         the t register holds root2
                    *--                               for the rotation section
                    *-- if im(u) < 0 then in quadrant 3 or 4
 005E   2010                 lac      u+1
 005F   FA00                 blz      Aquad34
 0060   0097
                    *-- if re(u) < 0 (and since im(u) > 0) then in quadrant 2
 0061   200F                 lac      u+0
 0062   FA00                 blz      Aquad2
 0063   007D
                    *-- in quadrant 1, if im(u) > re(u) then between pi/4 and pi/2
 0064   1010        Aquad1   sub      u+1
 0065   FA00                 blz      Ah2
 0066   0070
                    *-- in [0,pi/4]
 0067   200F        Ah1      lac      u+0
```

```
0068  500D              sacl    z+0
0069  2010              lac     u+1
006A  500E              sacl    z+1
006B  F800              call    Wdel6
006C  0646
                  *-- rotated by 0, so load ACC with 0
006D  7F89              zac
006E  F900              b       Aendrot
006F  00D2
                  *-- in [pi/4,pi/2]
0070  6D10        Ah2   mpy     u+1
0071  7F8E              pac
0072  6D0F              mpy     u+0
0073  7F90              spac
0074  590E              sach    z+1,1
0075  7F8F              apac
0076  7F8F              apac
0077  590D              sach    z+0,1
0078  7F80              nop
0079  7F80              nop
                  *-- rotated by -pi*8192/32768, so load ACC with 8192 = 0.25 Q15
007A  2D00              lac     one,13
007B  F900              b       Aendrot
007C  00D2
                  *-- quadrant 2, if im(u) < - re(u) then in [3pi/4,pi]
007D  0010        Aquad2 add    u+1
007E  FA00              blz     Ah4
007F  008A
                  *-- in [pi/2,3pi/4]
0080  2010        Ah3   lac     u+1
0081  500D              sacl    z+0
0082  7F89              zac
0083  100F              sub     u+0
0084  500E              sacl    z+1
0085  F800              call    Wdel5
0086  0647
                  *-- rotated by -pi*16384/32768, so load ACC with 16384 = 0.50 Q15
0087  2E00              lac     one,14
0088  F900              b       Aendrot
0089  00D2
                  *-- in [3pi/4,pi]
008A  7F89        Ah4   zac
008B  6D0F              mpy     u+0
008C  7F90              spac
008D  6D10              mpy     u+1
008E  7F8F              apac
008F  590D              sach    z+0,1
0090  7F90              spac
0091  7F90              spac
0092  590E              sach    z+1,1
```

```
                     *-- rotated by -pi*24576/32768 so load ACC with 24576 = 0.75 Q15
0093   2E00                  lac       one,14
0094   0D00                  add       one,13
0095   F900                  b         Aendrot
0096   00D2
                     *-- in quadrant 3 or 4, if re(u) > 0 then in quadrant 4
0097   200F          Aquad34 lac       u+0
0098   FC00                  bgz       Aquad4
0099   00B7
                     *-- in quadrant 3, if im(u) < re(u) then in [-pi/2,-3pi/4]
009A   1010          Aquad3  sub       u+1
009B   FC00                  bgz       Ah6
009C   00AA
                     *-- in [-3pi/4,-pi]
009D   7F89          Ah5     zac
009E   100F                  sub       u+0
009F   500D                  sacl      z+0
00A0   7F89                  zac
00A1   1010                  sub       u+1
00A2   500E                  sacl      z+1
00A3   7F80                  nop
00A4   7F80                  nop
00A5   7F80                  nop
                     *-- rotated by pi*32768/32768, so load ACC with -32768 = -1 Q15
00A6   7F89                  zac
00A7   1F00                  sub       one,15
00A8   F900                  b         Aendrot
00A9   00D2
                     *-- in [-pi/2,-3pi/4]
00AA   7F89          Ah6     zac
00AB   6D10                  mpy       u+1
00AC   7F90                  spac
00AD   6D0F                  mpy       u+0
00AE   7F8F                  apac
00AF   590E                  sach      z+1,1
00B0   7F90                  spac
00B1   7F90                  spac
00B2   590D                  sach      z+0,1
                     *-- rotated by pi*24576/32768, so load ACC with -24576 = -0.75 Q15
00B3   2D00                  lac       one,13
00B4   1F00                  sub       one,15
00B5   F900                  b         Aendrot
00B6   00D2
                     *-- in quadrant 4, if re(u) > - im(u) then in [0,-pi/4]
00B7   0010          Aquad4  add       u+1
00B8   FC00                  bgz       Ah8
00B9   00C5
                     *-- in [-pi/4,-pi/2]
00BA   7F89          Ah7     zac
00BB   1010                  sub       u+1
```

```
00BC  500D          sacl     z+0
00BD  200F          lac      u+0
00BE  500E          sacl     z+1
00BF  F800          call     Wdel4
00C0  0648
              *-- rotated by pi*16384/32768, so load ACC with -16384 =--0.50 Q15
00C1  7F89          zac
00C2  1E00          sub      one,14
00C3  F900          b        Aendrot
00C4  00D2
              *-- in [0,-pi/4]
00C5  6D0F    Ah8   mpy      u+0
00C6  7F8E          pac
00C7  6D10          mpy      u+1
00C8  7F90          spac
00C9  590D          sach     z+0,1
00CA  7F8F          apac
00CB  7F8F          apac
00CC  590E          sach     z+1,1
              *-- rotated by pi*8192/32768, so load ACC with -8192 = -0.25 Q15
00CD  7F89          zac
00CE  1D00          sub      one,13
00CF  7F80          nop
00D0  7F80          nop
00D1  7F80          nop
              *-- Arg(u) = pi*acc + arctan(z), where z is in [0..pi/4]
              *-- save ACC, and compute arctan(z)
00D2  5011    Aendrot  sacl  arg+0
              *-- compute temp = im(z)*0.28125
00D3  2E0E          lac      z+1,14
00D4  0B0E          add      z+1,11
00D5  5804          sach     temp
              *-- compute re(z)^2 + 0.28125*im(z)^2
00D6  6A0D          lt       z+0
00D7  6D0D          mpy      z+0
00D8  7F8E          pac
00D9  6A0E          lt       z+1
00DA  6D04          mpy      temp
00DB  7F8F          apac
00DC  590E          sach     z+1,1
              *-- compute re(z)*im(z)
00DD  6D0D          mpy      z+0
00DE  7F8E          pac
00DF  590D          sach     z+0,1
              *-- compute re(z)*im(z)/(re(z)^2 + 0.28*im(z)^2)
              *-- i.e. do the division portion
              *-- z+0 holds re(z)*im(z), z+1 holds (re(z)^2 + 0.28*im(z)^2)
00E0  650D          zalh     z+0
00E1  640E          subc     z+1
00E2  7F80          nop                 instruction following subc cannot be subc
```

```
00E3  640E                subc    z+1           (can be with 32020)
00E4  7F80                nop
00E5  640E                subc    z+1
00E6  7F80                nop
00E7  640E                subc    z+1
00E8  7F80                nop
00E9  640E                subc    z+1
00EA  7F80                nop
00EB  640E                subc    z+1
00EC  7F80                nop
00ED  640E                subc    z+1
00EE  7F80                nop
00EF  640E                subc    z+1
00F0  7F80                nop
00F1  640E                subc    z+1
00F2  7F80                nop
00F3  640E                subc    z+1
00F4  7F80                nop
00F5  640E                subc    z+1
00F6  7F80                nop
00F7  640E                subc    z+1
00F8  7F80                nop
00F9  640E                subc    z+1
00FA  7F80                nop
00FB  640E                subc    z+1
00FC  7F80                nop
00FD  640E                subc    z+1
00FE  7F80                nop
00FF  500D                sacl    z+0
                   *-- have arg(z) in [0..pi/4], now derotate and scale by 1/pi
                   *-- arg+0 contains (arg(u)-arg(z))/pi
                   *-- thus arg(u)/pi = arg(z)/pi + (arg(u)-arg(z)/pi
0100  2C11                lac     arg+0,12
0101  6A0D                lt      z+0
0102  8518                mpyk    1304          (1/pi, Q12 notation)
0103  7F8F                apac
0104  5C11                sach    arg+0,4
0105  7F8D                ret
                   *--
                          page
```

```
                copy      bpfilt.prc
*--
*== Procedure Bpfilt ======================================
*-- Bandpass Filter differential detection, comb filtering
*--         - called from acquisition section of demod
*--         - input is at 8 times the bit rate
*--         - output is at twice the bit rate
*--         - filter is FIR, length = 56
*--         - filter is broken up into 14 sections, each of
*--           length 4. this structure allows us to keep only
*--           the past 4 input samples and a few state variables
*--           in order to compute the filter output
*--         - have a single real input and a single complex output.
*--           the output is the complex bandpass envelope
*--         - after filtering, the signal is differentially detected
*--           (delay of one bit period). down conversion is redundant
*--           (and not done) since samples seperated by one bit period
*--           have an associated phase rotation of 2pifcT = 2pi
*--         - after differential detection comb filtering is performed.
*--           the comb buffer is cyclical over four bits and thus has
*--           length 8. shifting of the comb buffer is performed
*--           elsewhere. thus position comb+0 is always updated
*-- Input:
*--     buffer+[0..7]
*--           the 4 input samples are contained in locations buffer+0,
*--           buffer+2, buffer+4, buffer+6
*--           at the start of the procedure buffer+0 is copied into
*--           buffer+1, buffer+2 is copied into buffer+3 and so on.
*--           thus we may input a sample from the a/d into buffer+0
*--           since we have a copy of it in buffer+1
*--     afmemih+[0..12], afmemil+[0..12], afmemqh+[0..12], afmemql+[0..12]
*--           filter state variables, 32 bits in length. the low order 16 bits
*--           of the i channel filter are kept in location afmemil+[0..12],
*--           and the high order 16 bits are kept in location afmemih+[0..12]
*--     alpha+[0..1]
*--           comb memory (alpha+0) and gain (alpha+1)
*--     pi0..pi55, pq0..pq55 (constants defined in equate statements)
*--           filter coefficients for i and q channels
*-- Output:
*--     comb+0
*--           comb filter output, (only output needed and used elsewhere)
*--     u+0,u+1
*--           filter outputs
*--     afmemih+[0..12], afmemil+[0..12], afmemqh+[0..12], afmemql+[0..12]
*--           filter state variables are updated
*--     buffer+0, buffer+2 (needed for future filtering operations)
*--           two samples are read in from the a/d converter
*--     arg+0, arg+1, arg+2
*--           needed for differential detection, delay of T
*--     temp, z+0, z+1
```

```
             *--        needed for argument function
             *-- Other calls:
             *--     procedure Argument is called, which needs to locations on the
             *--     stack. thus, prior to calling Bpfilt, 3 values on the stack must
             *--     be available
             *-- Cycle timing:
             *--     after the call instruction 4 cycles have been used up
             *--     since the last input from the a/d. after the return
             *--     126 cycles have been utilized (since the last input)
             *--
             *--
             *-- get samples into correct location
             *-- (so future input instructions from the a/d have space for storage
             *--  without modifying the current output of the filters)
0106  6905   Bpfilt   dmov     buffer+0
0107  6907            dmov     buffer+2
0108  6909            dmov     buffer+4
0109  690B            dmov     buffer+6
             *-- i channel filtering
010A  6A06            lt       buffer+1
             *-- section 14
010B  6533            zalh     afmemih+12
010C  7A40            or       afmemil+12
010D  803F            mpyk     pi3
010E  6C0A            lta      buffer+5
010F  9FDC            mpyk     pi1
0110  6C0C            lta      buffer+7
0111  9FDC            mpyk     pi0
0112  6C06            lta      buffer+1
0113  590F            sach     u+0,1
             *-- section 13
0114  6532            zalh     afmemih+11
0115  7A3F            or       afmemil+11
0116  9FD6            mpyk     pi7
0117  6C0A            lta      buffer+5
0118  8052            mpyk     pi5
0119  6C0C            lta      buffer+7
011A  807B            mpyk     pi4
011B  6C06            lta      buffer+1
011C  5833            sach     afmemih+12
011D  5040            sacl     afmemil+12
             *-- section 12
011E  6531            zalh     afmemih+10
011F  7A3E            or       afmemil+10
0120  9F52            mpyk     pi11
0121  6C0A            lta      buffer+5
0122  8029            mpyk     pi9
0123  6C0C            lta      buffer+7
0124  9FE6            mpyk     pi8
0125  6C06            lta      buffer+1
```

```
 746   0126   5832              sach     afmemih+11
 747   0127   503F              sacl     afmemil+11
 748                   *-- section 11
 749   0128   6530              zalh     afmemih+9
 750   0129   7A3D              or       afmemil+9
 751   012A   8104              mpyk     pi15
 752   012B   6C0A              lta      buffer+5
 753   012C   9EF0              mpyk     pi13
 754   012D   6C0C              lta      buffer+7
 755   012E   9EC5              mpyk     pi12
 756   012F   6C06              lta      buffer+1
 757   0130   5831              sach     afmemih+10
 758   0131   503E              sacl     afmemil+10
 759                   *-- section 10
 760   0132   652F              zalh     afmemih+8
 761   0133   7A3C              or       afmemil+8
 762   0134   8152              mpyk     pi19
 763   0135   6C0A              lta      buffer+5
 764   0136   8046              mpyk     pi17
 765   0137   6C0C              lta      buffer+7
 766   0138   8107              mpyk     pi16
 767   0139   6C06              lta      buffer+1
 768   013A   5830              sach     afmemih+9
 769   013B   503D              sacl     afmemil+9
 770                   *-- section 9
 771   013C   652E              zalh     afmemih+7
 772   013D   7A3B              or       afmemil+7
 773   013E   9A28              mpyk     pi23
 774   013F   6C0A              lta      buffer+5
 775   0140   8381              mpyk     pi21
 776   0141   6C0C              lta      buffer+7
 777   0142   8355              mpyk     pi20
 778   0143   6C06              lta      buffer+1
 779   0144   582F              sach     afmemih+8
 780   0145   503C              sacl     afmemil+8
 781                   *-- section 8
 782   0146   652D              zalh     afmemih+6
 783   0147   7A3A              or       afmemil+6
 784   0148   88A9              mpyk     pi27
 785   0149   6C0A              lta      buffer+5
 786   014A   9839              mpyk     pi25
 787   014B   6C0C              lta      buffer+7
 788   014C   9645              mpyk     pi24
 789   014D   6C06              lta      buffer+1
 790   014E   582E              sach     afmemih+7
 791   014F   503B              sacl     afmemil+7
 792                   *-- section 7
 793   0150   652C              zalh     afmemih+5
 794   0151   7A39              or       afmemil+5
 795   0152   991E              mpyk     pi31
 796   0153   6C0A            lta     buffer+5
 797   0154   8864            mpyk    pi29
 798   0155   6C0C            lta     buffer+7
 799   0156   8C40            mpyk    pi28
 800   0157   6C06            lta     buffer+1
 801   0158   582D            sach    afmemih+6
 802   0159   503A            sacl    afmemil+6
 803                  *-- section 6
 804   015A   652B            zalh    afmemih+4
 805   015B   7A38            or      afmemil+4
 806   015C   825B            mpyk    pi35
 807   015D   6C0A            lta     buffer+5
 808   015E   9B4C            mpyk    pi33
 809   015F   6C0C            lta     buffer+7
 810   0160   97BD            mpyk    pi32
 811   0161   6C06            lta     buffer+1
 812   0162   582C            sach    afmemih+5
 813   0163   5039            sacl    afmemil+5
 814                  *-- section 5
 815   0164   652A            zalh    afmemih+3
 816   0165   7A37            or      afmemil+3
 817   0166   80BA            mpyk    pi39
 818   0167   6C0A            lta     buffer+5
 819   0168   806C            mpyk    pi37
 820   0169   6C0C            lta     buffer+7
 821   016A   81DF            mpyk    pi36
 822   016B   6C06            lta     buffer+1
 823   016C   582B            sach    afmemih+4
 824   016D   5038            sacl    afmemil+4
 825                  *-- section 4
 826   016E   6529            zalh    afmemih+2
 827   016F   7A36            or      afmemil+2
 828   0170   9F21            mpyk    pi43
 829   0171   6C0A            lta     buffer+5
 830   0172   8124            mpyk    pi41
 831   0173   6C0C            lta     buffer+7
 832   0174   816F            mpyk    pi40
 833   0175   6C06            lta     buffer+1
 834   0176   582A            sach    afmemih+3
 835   0177   5037            sacl    afmemil+3
 836                  *-- section 3
 837   0178   6528            zalh    afmemih+1
 838   0179   7A35            or      afmemil+1
 839   017A   9FEE            mpyk    pi47
 840   017B   6C0A            lta     buffer+5
 841   017C   9F8C            mpyk    pi45
 842   017D   6C0C            lta     buffer+7
 843   017E   9F0A            mpyk    pi44
 844   017F   6C06            lta     buffer+1
 845                  *-- get sample
```

```
0180  4105              in      buffer+0,adc
            *--
0181  5829              sach    afmemih+2
0182  5036              sacl    afmemil+2
            *-- section 2
0183  6527              zalh    afmemih+0
0184  7A34              or      afmemil+0
0185  8057              mpyk    pi51
0186  6C0A              lta     buffer+5
0187  9FC5              mpyk    pi49
0188  6C0C              lta     buffer+7
0189  9FC4              mpyk    pi48
018A  6C06              lta     buffer+1
018B  5828              sach    afmemih+1
018C  5035              sacl    afmemil+1
            *-- section 1
018D  7F89              zac
018E  9FE7              mpyk    pi55
018F  6C0A              lta     buffer+5
0190  8028              mpyk    pi53
0191  6C0C              lta     buffer+7
0192  8059              mpyk    pi52
0193  6C06              lta     buffer+1
0194  5827              sach    afmemih+0
0195  5034              sacl    afmemil+0
            *-- q channel
            *-- section 14
0196  654D              zalh    afmemqh+12
0197  7A5A              or      afmemql+12
0198  9FC1              mpyk    pq3
0199  6C08              lta     buffer+3
019A  9FC7              mpyk    pq2
019B  6C0A              lta     buffer+5
019C  9FDC              mpyk    pq1
019D  6C06              lta     buffer+1
019E  5910              sach    u+1,1
            *-- section 13
019F  654C              zalh    afmemqh+11
01A0  7A59              or      afmemql+11
01A1  802A              mpyk    pq7
01A2  6C08              lta     buffer+3
01A3  8054              mpyk    pq6
01A4  6C0A              lta     buffer+5
01A5  8052              mpyk    pq5
01A6  6C06              lta     buffer+1
01A7  584D              sach    afmemqh+12
01A8  505A              sacl    afmemql+12
            *-- section 12
01A9  654B              zalh    afmemqh+10
01AA  7A58              or      afmemql+10
```

```
 896   01AB   80AE             mpyk    pq11
 897   01AC   6C08             lta     buffer+3
 898   01AD   80A5             mpyk    pq10
 899   01AE   6C0A             lta     buffer+5
 900   01AF   8029             mpyk    pq9
 901   01B0   6C06             lta     buffer+1
 902   01B1   584C             sach    afmemqh+11
 903   01B2   5059             sacl    afmemql+11
 904                   *-- section 11
 905   01B3   654A             zalh    afmemqh+9
 906   01B4   7A57             or      afmemql+9
 907   01B5   9EFC             mpyk    pq15
 908   01B6   6C08             lta     buffer+3
 909   01B7   9E62             mpyk    pq14
 910   01B8   6C0A             lta     buffer+5
 911   01B9   9EF0             mpyk    pq13
 912   01BA   6C06             lta     buffer+1
 913   01BB   584B             sach    afmemqh+10
 914   01BC   5058             sacl    afmemql+10
 915                   *-- section 10
 916   01BD   6549             zalh    afmemqh+8
 917   01BE   7A56             or      afmemql+8
 918   01BF   9EAE             mpyk    pq19
 919   01C0   6C08             lta     buffer+3
 920   01C1   9F67             mpyk    pq18
 921   01C2   6C0A             lta     buffer+5
 922   01C3   8046             mpyk    pq17
 923   01C4   6C06             lta     buffer+1
 924   01C5   584A             sach    afmemqh+9
 925   01C6   5057             sacl    afmemql+9
 926                   *-- section 9
 927   01C7   6548             zalh    afmemqh+7
 928   01C8   7A55             or      afmemql+7
 929   01C9   85D8             mpyk    pq23
 930   01CA   6C08             lta     buffer+3
 931   01CB   86A7             mpyk    pq22
 932   01CC   6C0A             lta     buffer+5
 933   01CD   B381             mpyk    pq21
 934   01CE   6C06             lta     buffer+1
 935   01CF   5849             sach    afmemqh+8
 936   01D0   5056             sacl    afmemql+8
 937                   *-- section 8
 938   01D1   6547             zalh    afmemqh+6
 939   01D2   7A54             or      afmemql+6
 940   01D3   9757             mpyk    pq27
 941   01D4   6C08             lta     buffer+3
 942   01D5   9422             mpyk    pq26
 943   01D6   6C0A             lta     buffer+5
 944   01D7   9839             mpyk    pq25
 945   01D8   6C06             lta     buffer+1
```

```
01D9  5848           sach    afmemqh+7
01DA  5055           sacl    afmemql+7
              *-- section 7
01DB  6546           zalh    afmemqh+5
01DC  7A53           or      afmemql+5
01DD  86E2           mpyk    pq31
01DE  6C08           lta     buffer+3
01DF  8AFF           mpyk    pq30
01E0  6C0A           lta     buffer+5
01E1  8864           mpyk    pq29
01E2  6C06           lta     buffer+1
01E3  5847           sach    afmemqh+6
01E4  5054           sacl    afmemql+6
              *-- section 6
01E5  6545           zalh    afmemqh+4
01E6  7A52           or      afmemql+4
01E7  9DA5           mpyk    pq35
01E8  6C08           lta     buffer+3
01E9  9B0C           mpyk    pq34
01EA  6C0A           lta     buffer+5
01EB  9B4C           mpyk    pq33
01EC  6C06           lta     buffer+1
01ED  5846           sach    afmemqh+5
01EE  5053           sacl    afmemql+5
              *-- section 5
01EF  6544           zalh    afmemqh+3
01F0  7A51           or      afmemql+3
01F1  9F46           mpyk    pq39
01F2  6C08           lta     buffer+3
01F3  9F9D           mpyk    pq38
01F4  6C0A           lta     buffer+5
01F5  806C           mpyk    pq37
01F6  6C06           lta     buffer+1
01F7  5845           sach    afmemqh+4
01F8  5052           sacl    afmemql+4
              *-- section 4
01F9  6543           zalh    afmemqh+2
01FA  7A50           or      afmemql+2
01FB  80DF           mpyk    pq43
01FC  6C08           lta     buffer+3
01FD  8180           mpyk    pq42
01FE  6C0A           lta     buffer+5
              *-- input a sample
01FF  4107           in      buffer+2,adc
              *--
0200  8124           mpyk    pq41
0201  6C06           lta     buffer+1
0202  5844           sach    afmemqh+3
0203  5051           sacl    afmemql+3
              *-- section 3
```

```
 996   0204   6542          zalh    afmemqh+1
 997   0205   7A4F          or      afmemql+1
 998   0206   8012          mpyk    pq47
 999   0207   6C08          lta     buffer+3
1000   0208   9FC6          mpyk    pq46
1001   0209   6C0A          lta     buffer+5
1002   020A   9F8C          mpyk    pq45
1003   020B   6C06          lta     buffer+1
1004   020C   5843          sach    afmemqh+2
1005   020D   5050          sacl    afmemql+2
1006                  *-- section 2
1007   020E   6541          zalh    afmemqh+0
1008   020F   7A4E          or      afmemql+0
1009   0210   9FA9          mpyk    pq51
1010   0211   6C08          lta     buffer+3
1011   0212   9F8C          mpyk    pq50
1012   0213   6C0A          lta     buffer+5
1013   0214   9FC5          mpyk    pq49
1014   0215   6C06          lta     buffer+1
1015   0216   5842          sach    afmemqh+1
1016   0217   504F          sacl    afmemql+1
1017                  *-- section 1
1018   0218   7F89          zac
1019   0219   8019          mpyk    pq55
1020   021A   6C08          lta     buffer+3
1021   021B   8033          mpyk    pq54
1022   021C   6C0A          lta     buffer+5
1023   021D   8028          mpyk    pq53
1024   021E   7F8F          apac
1025   021F   5841          sach    afmemqh+0
1026   0220   504E          sacl    afmemql+0
1027                  *-- calculate argument
1028   0221   F800          call    Argument
       0222   005D
1029                  *-- differential detection
1030   0223   2011          lac     arg+0
1031   0224   1013          sub     arg+2
1032   0225   5014          sacl    y
1033                  *-- delay by T
1034   0226   6912          dmov    arg+1
1035   0227   6911          dmov    arg+0
1036                  *-- comb filter
1037   0228   6517          zalh    comb+0
1038   0229   6A17          lt      comb+0
1039   022A   6D63          mpy     alpha+0
1040   022B   7F90          spac
1041   022C   6A14          lt      y
1042   022D   6D64          mpy     alpha+1
1043   022E   7F8F          apac
1044   022F   5817          sach    comb+0
```

```
  1045   0230   7F8D             ret
  1046                   *--
  1047                           page
```

```
         copy      corr.prc
*--
*== Procedure Corr ========================================
*-- Correlate (circular)
*--      - perform one step of the circular correlation and
*--        search for maximum and zero crossing
*--
*-- Input:
*--     registers
*--        ar0 points to the current sample in the comb buffer
*--        ar1 points to the clean signal (lcorr)
*--     tcorr+1
*--        the last correlator output if zero crossing has not been found
*--        otherwise it is the right side of the zero crossing
*--     tcorr+2
*--        the left side of the zero crossing, if the zero crossing
*--        has been found, zero otherwise
*--     corrmax
*--        the previous maximum correlator output
*--     zcflag
*--        -1 if on sample zero
*--        0 if zero crossing not found
*--        1 if zero crossing has been found
*--        if we are on sample zero, just evaluate tcorr+1 and return
*--        if we have found the zero crossing we return
*--        if we have not found the zero crossing we determine if there
*--        was a zero crossing.
*-- Output:
*--     corrmax
*--        if the correlator output exceeded the maximum then corrmax
*--        is updated
*--     count+1
*--        sample of zero crossing, incremented if zero crossing not
*--        found
*--     temp
*--        temp is modified
*--     tcorr+0
*--        the correlator output
*--     tcorr+[1..2]
*--        if the zero crossing was previously found, then tcorr+1
*--        is one sample in the zero crossing and tcorr+2 is the other.
*--        otherwise tcorr+1 = tcorr+0. if the zero crossing was just
*--        found tcorr+2 = a sample in zero crossing, otherwise it is 0
*--     zcflag
*--        = 0 if zero crossing not found
*--        = 1 otherwise
*--     registers
*--        ar0 is saved and restored in the procedure. thus, it is not
*--        modified. ar1 is decremented by 8 in the procedure. thus,
*--        apon returning it points to the next clean signal slipped
```

```
 1098                    *--       1/8th of a bit from the current.
 1099                    *--       arp = 0 after returning
 1100                    *--       the ACC, T and P registers are also modified
 1101                    *-- Other calls:
 1102                    *--       one call is made (delay proc), thus two stack positions
 1103                    *--       are required before call is made
 1104                    *-- Cycle timing
 1105                    *--       executes in 45 cycles, (including call)
 1106                    *--
 1107   0231   3004      Corr     sar      ar0,temp       save ar0
 1108                    *-- do correlation
 1109   0232   7F89               zac
 1110   0233   6AA1               lt       *+,ar1
 1111   0234   6D90               mpy      *-,ar0
 1112   0235   6CA1               lta      *+,ar1
 1113   0236   6D90               mpy      *-,ar0
 1114   0237   6CA1               lta      *+,ar1
 1115   0238   6D90               mpy      *-,ar0
 1116   0239   6CA1               lta      *+,ar1
 1117   023A   6D90               mpy      *-,ar0
 1118   023B   6CA1               lta      *+,ar1
 1119   023C   6D90               mpy      *-,ar0
 1120   023D   6CA1               lta      *+,ar1
 1121   023E   6D90               mpy      *-,ar0
 1122   023F   6CA1               lta      *+,ar1
 1123   0240   6D90               mpy      *-,ar0
 1124   0241   6CA1               lta      *+,ar1
 1125   0242   6D90               mpy      *-,ar0
 1126   0243   7F8F               apac
 1127   0244   5847               sach     tcorr+0
 1128                    *-- does current output exceed maximum
 1129   0245   2047               lac      tcorr+0
 1130   0246   104A               sub      corrmax
 1131   0247   FC00               bgz      Cswtch
        0248   024B
 1132   0249   F900               b        C0
        024A   024D
 1133                    *-- have new maximum, make assignment
 1134   024B   2047      Cswtch   lac      tcorr+0
 1135   024C   504A               sacl     corrmax
 1136                    *--
 1137   024D   3804      C0       lar      ar0,temp       restore ar0
 1138                    *-- zero crossing not found ?
 1139   024E   204C               lac      zcflag
 1140   024F   FF00               bz       Czcross
        0250   0260
 1141                    *-- on first sample ?
 1142   0251   FA00               blz      Czc0
        0252   0257
 1143                    *-- zero crossing already found, delay and exit
```

```
0253  F800          call    Wdel7
0254  0645
0255  F900          b       Corrend
0256  0271
                *-- on first sample, calculate new flag, save tcorr+0, increment count
0257  7F89  Czc0    zac
0258  504C          sacl    zcflag
0259  6947          dmov    tcorr+0
025A  204E          lac     count+1
025B  0000          add     one
025C  504E          sacl    count+1
025D  7F80          nop
025E  F900          b       Corrend
025F  0271
                *-- look for sign change,
0260  6A47  Czcross lt      tcorr+0
0261  6D48          mpy     tcorr+1
0262  7F8E          pac
0263  FC00          bgz     Czcnfnd
0264  026B
                *-- zero crossing found
                *-- save the autocorrelation outputs at the zero crossing
                *-- set flag
0265  2000          lac     one
0266  504C          sacl    zcflag
0267  6948          dmov    tcorr+1
0268  6947          dmov    tcorr+0
0269  F900          b       Corrend
026A  0271
                *-- zero crossing not found, save correlator output, increment count
026B  6947  Czcnfnd dmov    tcorr+0
026C  204E          lac     count+1
026D  0000          add     one
026E  504E          sacl    count+1
026F  7F80          nop
0270  7F80          nop
                *--
0271  7F8D  Corrend ret
                *--
                    page
```

```
                  copy      derot.prc
        *..
        *== Procedure derot ====================================
        *-- Derotate
        *--      - derotate input samples (i.e. downconvert)
        *--            r(k) <- r(k)*phi
        *--      - update derotation variables
        *--      - input sample is multiplied by phi(k)
        *--        where phi(k) = phi(k-1)*conj(fo)
        *--            phi(0) = 1 + j0
        *--            fo = exp(j2pifdT/md)*exp(j2pifcT/md)
        *--            fd = frequency offset
        *--            fc = carrier frequency = 4.8kHz
        *--            md = samples/bit = 8
        *-- Input:
        *--     phi+[0,1]
        *--        phi (complex) holds the current phase of the
        *--        carrier.
        *--     fo+[0,1]
        *--        fo (complex) holds the frequency offset and
        *--        the carrier frequency fo = exp(j2pi(fd+fc)T/md)
        *--     buffer+[0..7]
        *--        buffer is an array of real input samples (i.e. the
        *--        samples read from the a/d converter. it is addressed
        *--        by auxiliary register 0
        *--     registers
        *--        auxiliary register 0 points to 1 of 8 positions in
        *--        the array "buffer"
        *--        auxiliary register 1 points to one of 8 complex samples
        *--        in the array secbuf
        *--        auxiliary register pointer (arp) = 0 on input
        *-- Output:
        *--     phi+[0,1]
        *--        phi is multiplied by fo in order to be set up
        *--        for the next call
        *--     secbuf+[0..15]
        *--        secbuf is an array of complex de-rotated samples
        *--        format is (re,im,re,im,...re,im)
        *--        the array is addressed using auxiliary register 1
        *--     registers
        *--        on return auxiliary register zero points to the next
        *--        element in the array "buffer". auxiliary register one
        *--        points to the next complex sample in the array "secbuf"
        *--        (i.e. aux reg 1 is incremented by 2 in the proc)
        *--        arp = 0 on output
        *--        the accumulator, T and P registers are also modified
        *-- Other calls:
        *--        none
        *-- Cycle timing:
        *--        procedure executes in 25 cycles (including call)
```

```
                    *--
                    *-- derotate input sample
0272     6AA1       Derot    lt       *+,ar1
0273     6D17                mpy      phi+0
0274     7F8E                pac
0275     59A8                sach     *+,1
0276     6D18                mpy      phi+1
0277     7F8E                pac
0278     59A0                sach     *+,1,ar0
                    *-- update derotation variable (phi), phi <- phi*conj(fo)
0279     7F89                zac
027A     6A18                lt       phi+1
027B     6D1A                mpy      fo+1
027C     6C17                lta      phi+0
027D     6D19                mpy      fo+0
027E     7F8F                apac
027F     5917                sach     phi+0,1
0280     7F89                zac
0281     6D1A                mpy      fo+1
0282     7F90                spac
0283     6A18                lt       phi+1
0284     6D19                mpy      fo+0
0285     7F8F                apac
0286     5918                sach     phi+1,1
0287     7F8D                ret
                    *--
                             page
```

```
                      copy      eshft.prc
            *--
            *== Procedure Eshft========================================
            *-- circular shift of Comb buffer
            *-- Input:
            *--     comb+[0..8]
            *--        the comb filter outputs
            *-- Output:
            *--     comb+0..8
            *--        the comb filter outputs circularly shifted (one position)
            *--     registers
            *--        only ACC is modified
            *-- Other calls:
            *--        none
            *-- Cycle timing:
            *--        executes in 14 cycles (including call)
0288  691E  Eshft     dmov      comb+7
0289  691D            dmov      comb+6
028A  691C            dmov      comb+5
028B  691B            dmov      comb+4
028C  691A            dmov      comb+3
028D  6919            dmov      comb+2
028E  6918            dmov      comb+1
028F  6917            dmov      comb+0
0290  201F            lac       comb+8
0291  5017            sacl      comb+0
0292  7F8D            ret
            *--
                      page
```

```
         copy      freqoff.prc
*--
*== Procedure Freqoff ====================================
*-- estimate the Frequency Offset and bit phase
*--      - estimate the frequency offset and bit clock phase
*--        from the contents of the comb buffer
*--      - sum the comb buffer, this gives 2fdT
*--        (fd = frequency offset, T = bit period)
*--      - for demod we require cos(2pifdT/md), sin(2pifdT/md) (md = 8)
*--        thus, we must compute a sin and cos function. however
*--        2pifdT/md has a maximum value of 0.19635 = 11.25 degrees
*--        for fd = 1200 Hz. thus, the sin and cos table need not be
*--        complete.
*--      - the bit clock phase is estimated by inserting zero's in the
*--        comb buffer (i.e. back to 8 samples per bit), correllating
*--        against a clean signal (comb buffer in absence of noise),
*--        and finding the zero crossing. using this information we
*--        then delay until the start of the bit and enter demod.
*--        the maximum correlation is also computed, and if this
*--        does not exceed a threshold, it is assumed demod was
*--        entered prematurely.
*--
*-- Input:
*--     comb+[0..7]
*--        the comb buffer
*--
*-- Output:
*--     cos+0
*--        holds cos(2pifdT/md)
*--     sin+0
*--        holds sin(2pifdT/md)
*--     fo+[0..1]
*--        holds exp(j2pi(fd+fc)T/md) (fc = carrier frequency)
*--        notice that fo is a demod variable, care must be taken
*--        to ensure that fo does not overlap any vital acquisition
*--        variables. we chose to overlap fo and the comb buffer,
*--        and fo is only assigned a value at the end of this proc.
*--     comb+[8..15]
*--        comb+[0..7] is copied to comb+[8..15]. needed for
*--        correlation computation.
*--     temp, arg+[0..2]
*--        locations temp and arg+[0..2] are modified
*--     zcflag
*--        flag indicating when zero crossing is found
*--     count+[0..1]
*--        count+0 is a loop counter for the correlation
*--        computation section, and count+1 is a pointer to the
*--        zero crossing of the correlation
*--     tcorr+[0..2]
*--        tcorr+0 holds the most recent output of correlation
```

```
                    *--       tcorr+[1..2] holds each value at the zero crossing
                    *--       (i.e. a positive and negative value)
                    *--    registers
                    *--       T, P, ar0, ar1 are modified
                    *--       arp = 0 on entry and on exit
                    *--       ACC, if ACC <= 0 on exit, the correlation produced
                    *--       a valid maximum. otherwise (ACC > 0) the maximum
                    *--       correlation was to small and synch hunt should be
                    *--       re-entered
                    *-- Other calls:
                    *--       procedure Corr is called
                    *--       (two stack positions must be available before Freqoff
                    *--       is called)
                    *-- Cycle timing:
                    *--       the actual value is not important, but must
                    *--       execute in the same number of cycles regardless
                    *--       of branches taken.
                    *--
                    *--
0293    2D17        Freqoff  lac       comb+0,13
0294    0D18                 add       comb+1,13
0295    0D19                 add       comb+2,13
0296    0D1A                 add       comb+3,13
0297    0D1B                 add       comb+4,13
0298    0D1C                 add       comb+5,13
0299    0D1D                 add       comb+6,13
029A    0D1E                 add       comb+7,13
029B    5811                 sach      arg+0
                    *--
                    *-- Sine and Cosine computation
                    *--    compute sin(pi*arg/md) and cos(pi*arg/md) where arg is an element of
                    *--    [-1/2,1/2]
                    *--    computed using table look-up combined with linear interpolation
                    *--
                    *-- determine sign, store sign in location arg+1, take |arg+0|
029C    FA00                 blz       Fnegarg
029D    02A3
029E    2000                 lac       one
029F    5012                 sacl      arg+1
02A0    7F80                 nop
02A1    F900                 b         Fcoarse
02A2    02A8
02A3    7F88        Fnegarg  abs
02A4    5811                 sach      arg+0
02A5    7F89                 zac
02A6    1000                 sub       one
02A7    5012                 sacl      arg+1
                    *-- coarse evaluation of sine and cosine
                    *-- arg+0 is in [0..0.499969) (Q15), (or 0..16383)
02A8    2711        Fcoarse  lac       arg+0,7
```

```
02A9   5813              sach      arg+2
                 *-- arg+2 is in [0..31], use arg+2 to index into a 33 word table
                 *-- get two values, upper and lower so we can linear interpolate
02AA   2013              lac       arg+2
02AB   6A00              lt        one
02AC   8333              mpyk      Fsintbl
02AD   7F8F              apac
02AE   674F              tblr      sin+0
02AF   0000              add       one
02B0   6750              tblr      sin+1
                 *-- sin and cos tables are one after the other
                 *-- both are 33 words long
02B1   0500              add       one,5
02B2   6751              tblr      cos+0
02B3   0000              add       one
02B4   6752              tblr      cos+1
                 *-- fine evaluation, (linear interpolate)
                 *-- load the fraction discarded in the coarse evaluation
02B5   2611              lac       arg+0,6
02B6   1F13              sub       arg+2,15
02B7   5013              sacl      arg+2
                 *-- do linear interpolation, sin
02B8   2050              lac       sin+1
02B9   104F              sub       sin+0
02BA   5004              sacl      temp
02BB   2F4F              lac       sin+0,15
02BC   6A04              lt        temp
02BD   6D13              mpy       arg+2
02BE   7F8F              apac
02BF   594F              sach      sin+0,1
                 *-- multiply by sign (contained in arg+1)
02C0   6A12              lt        arg+1
02C1   6D4F              mpy       sin+0
02C2   7F8E              pac
02C3   504F              sacl      sin+0
                 *-- do linear interpolation, cos
02C4   2052              lac       cos+1
02C5   1051              sub       cos+0
02C6   5004              sacl      temp
02C7   2F51              lac       cos+0,15
02C8   6A04              lt        temp
02C9   6D13              mpy       arg+2
02CA   7F8F              apac
02CB   5951              sach      cos+0,1
                 *--
                 *-- compute circular correlation of comb buffer with clean comb buffer
                 *-- (i.e. the contents of the comb buffer in the absence of noise, and
                 *--  sampled at 8 samples per bit). thus, we could insert three zeros
                 *--  between each sample in the comb buffer, and correlate against
                 *--  the clean signal for the 32 possible time slips. alternatively, (and
 1433                  *--  equivalently) we correlate the comb buffer against every fourth
 1434                  *--  sample in the clean signal, (again 32 possible time slips)
 1435                  *--  the correlator is stored (in lcorr+[0..31] as follows (see proc Initcorr)
 1436                  *--      corr(3), corr(7), ... corr(31), corr(2), corr(6),... corr(30),
 1437                  *--      corr(1), corr(5), ... corr(29), corr(0), corr(4),... corr(28)
 1438                  *--  (i.e. lcorr+0 = corr(3) and lcorr+31 = corr(28)
 1439                  *--  the comb buffer is duplicated to prepare for circular correlation
 1440                  *--  (i.e. comb+8 = comb +0, ... , comb+15 = comb + 7)
 1441                  *--
 1442                  *--  the zero crossing in the correlation output is found (see procedure
 1443                  *--  Corr) and the maximum is also found. the zero crossing determines the
 1444                  *--  bit phase, and the maximum determines if the incoming signal was of
 1445                  *--  appropriate power
 1446                  *--
 1447                  *-- get comb buffer ready for circular correlation
 1448  02CC  2017            lac       comb+0
 1449  02CD  501F            sacl      comb+8
 1450  02CE  2018            lac       comb+1
 1451  02CF  5020            sacl      comb+9
 1452  02D0  2019            lac       comb+2
 1453  02D1  5021            sacl      comb+10
 1454  02D2  201A            lac       comb+3
 1455  02D3  5022            sacl      comb+11
 1456  02D4  201B            lac       comb+4
 1457  02D5  5023            sacl      comb+12
 1458  02D6  201C            lac       comb+5
 1459  02D7  5024            sacl      comb+13
 1460  02D8  201D            lac       comb+6
 1461  02D9  5025            sacl      comb+14
 1462  02DA  201E            lac       comb+7
 1463  02DB  5026            sacl      comb+15
 1464                  *-- pointer to comb buffer
 1465  02DC  7017            lark      ar0,comb+0
 1466                  *-- flag for zero crossing
 1467  02DD  7F89            zac
 1468  02DE  1000            sub       one
 1469  02DF  504C            sacl      zcflag
 1470  02E0  504E            sacl      count+1
 1471                  *-- count+0 is loop counter
 1472  02E1  7E08            lack      8
 1473                  *-- repeat loop 8 times, (get 32 correlator outputs)
 1474  02E2  504D      Facorrzc sacl      count+0
 1475  02E3  7146            lark      ar1,lcorr+31
 1476                  *-- correlate against lcorr+31...lcorr+24
 1477  02E4  F800            call      Corr
       02E5  0231
 1478                  *-- correlate against lcorr+23...lcorr+16
 1479  02E6  F800            call      Corr
       02E7  0231
 1480                  *-- correlate against lcorr+15...lcorr+8
 1481  02E8  F800             call      Corr
       02E9  0231
 1482              *-- correlate against lcorr+7...lcorr+0
 1483  02EA  F800             call      Corr
       02EB  0231
 1484              *-- point to next comb sample
 1485  02EC  68A8             mar       *+
 1486              *-- check count
 1487  02ED  204D             lac       count+0
 1488  02EE  1000             sub       one
 1489  02EF  FE00             bnz       Facorrzc
       02F0  02E2
 1490              *-- linear interpolate to find more exact zero crossing
 1491  02F1  2048             lac       tcorr+1
 1492  02F2  FF00             bz        Fztc1
       02F3  0306
 1493  02F4  2049             lac       tcorr+2
 1494  02F5  FF00             bz        Fztc2
       02F6  030D
 1495              *-- tcorr+1 is the most recent output sample (right side of zero crossing)
 1496  02F7  2048             lac       tcorr+1
 1497  02F8  1049             sub       tcorr+2
 1498  02F9  7F88             abs
 1499  02FA  5004             sacl      temp
 1500  02FB  2049             lac       tcorr+2
 1501  02FC  7F88             abs
 1502  02FD  5049             sacl      tcorr+2
 1503              *..
 1504  02FE  6549             zalh      tcorr+2
 1505  02FF  700E             lark      ar0,14
 1506              *..
 1507  0300  6404  Fzcdiv     subc      temp
 1508  0301  F400             banz      Fzcdiv
       0302  0300
 1509  0303  5004             sacl      temp
 1510  0304  F900             b         Fcalcdel
       0305  0315
 1511              *..
 1512  0306  7F89  Fztc1      zac
 1513  0307  5004             sacl      temp
 1514  0308  7E11             lack      17
 1515  0309  F800             call      Wdeln4
       030A  064B
 1516  030B  F900             b         Fcalcdel
       030C  0315
 1517              *..
 1518  030D  204E  Fztc2      lac       count+1
 1519  030E  0000             add       one
 1520  030F  504E             sacl      count+1
 1521  0310  7F89             zac
 1522  0311  5004              sacl     temp
 1523  0312  7E0F              lack     15
 1524  0313  F800              call     Wdeln6
       0314  0649
 1525                   *-- calculate the delay before demod commences
 1526                   *-- want delay in cycles
 1527                   *-- delay eigths of a bit
 1528  0315  7E20       Fcalcdel lack     32
 1529  0316  104E              sub      count+1
 1530  0317  504E              sacl     count+1
 1531                   *-- delay in fractions of a bit (due to linear interpolation)
 1532  0318  7F89              zac
 1533  0319  1604              sub      temp,6
 1534  031A  584D              sach     count+0
 1535                   *-- add together                 get delay in multiple of clock cycles
 1536  031B  204D              lac      count+0        1 = delay of 4 clock cycles
 1537  031C  054E              add      count+1,5      maximum value is 32*32 = 1024
 1538  031D  504D              sacl     count+0
 1539                   *-- take modulus, so delay is less than one bit period
 1540                   *-- one bit period = 256 = 1024 cycles, thus take mod 256
 1541                   *-- add fudge factor so demod starts at beginning of bit
 1542                   *-- (fudge factor computed by trial and error)
 1543  031E  7EFF              lack     >FF
 1544  031F  504E              sacl     count+1
 1545  0320  7EBF              lack     191            fudge
 1546  0321  004D              add      count+0
 1547  0322  794E              and      count+1        modulo 256
 1548                   *-- delay
 1549  0323  1000       Fcldel  sub      one
 1550  0324  7F80              nop
 1551  0325  FE00              bnz      Fcldel
       0326  0323
 1552                   *-- fo <- exp(2pifdT/md)*exp(2pifcT/md)
 1553  0327  6A02              lt       root2
 1554  0328  6D51              mpy      cos+0
 1555  0329  7F8E              pac
 1556  032A  6D4F              mpy      sin+0
 1557  032B  7F90              spac
 1558  032C  5919              sach     fo+0,1
 1559  032D  7F8F              apac
 1560  032E  7F8F              apac
 1561  032F  591A              sach     fo+1,1
 1562                   *-- compute threshold - maximum correlation, return result in accumulator
 1563  0330  204B              lac      corrthr
 1564  0331  104A              sub      corrmax
 1565  0332  7F8D              ret
 1566                   *--
 1567  0333  0000       Fsintbl data     0,201,402,603,804,1005,1206,1407
       0334  00C9
       0335  0192
```

```
      0336  025B
      0337  0324
      0338  03ED
      0339  04B6
      033A  057F
1568  033B  0648            data    1608,1809,2009,2210,2410,2611,2811,3012
      033C  0711
      033D  07D9
      033E  08A2
      033F  096A
      0340  0A33
      0341  0AFB
      0342  0BC4
1569  0343  0C8C            data    3212,3412,3612,3811,4011,4210,4410,4609
      0344  0D54
      0345  0E1C
      0346  0EE3
      0347  0FAB
      0348  1072
      0349  113A
      034A  1201
1570  034B  12C8            data    4808,5007,5205,5404,5602,5800,5998,6195
      034C  138F
      034D  1455
      034E  151C
      034F  15E2
      0350  16A8
      0351  176E
      0352  1833
1571  0353  18F9            data    6393
1572              *-- cos table must follow sin table (immediately)
1573  0354  7FFF  Fcostbl   data    32767,32766,32765,32761,32757,32752,32745,32737
      0355  7FFE
      0356  7FFD
      0357  7FF9
      0358  7FF5
      0359  7FF0
      035A  7FE9
      035B  7FE1
1574  035C  7FD8            data    32728,32717,32705,32692,32678,32663,32646,32628
      035D  7FCD
      035E  7FC1
      035F  7FB4
      0360  7FA6
      0361  7F97
      0362  7F86
      0363  7F74
1575  0364  7F61            data    32609,32589,32567,32545,32521,32495,32469,32441
      0365  7F4D
      0366  7F37
      0367  7F21
      0368  7F09
      0369  7EEF
      036A  7ED5
      036B  7EB9
 1576 036C  7E9C          data     32412,32382,32351,32318,32285,32250,32213,32176-
      036D  7E7E
      036E  7E5F
      036F  7E3E
      0370  7E1D
      0371  7DFA
      0372  7DD5
      0373  7DB0
 1577 0374  7D89          data     32137
 1578                *..
 1579                     page
```

```
                        copy      hcorr.prc
               *--
               *== Procedure Hcorr ======================================
               *-- short Correlation during synch Hunt
               *--      - correlate the comb filter against the clean signal
               *--      - the comb filter output is then circularly shifted
               *--      - the correlation is compared to a threshhold and
               *--      - the difference is returned in location temp
               *--        and in the ACC
               *--      - executes in 28 cycles, including call
               *--
               *-- Input:
               *--     comb+[0..8]
               *--        the most recent comb filter outputs
               *--        (comb+0 is the most recent, comb+7 is the oldest)
               *--     scorr+[0..7]
               *--        the "clean signal", the comb filter is correlated
               *--        with this buffer, and compared to a threshold
               *--     thresh
               *--        holds a constant, constant is subtracted from
               *--        the correlator output magnitude (when the difference
               *--        is greater than 0, the signal present decision
               *--        is made)
               *-- Output
               *--     temp
               *--        hold the magnitude of the correlator output less the
               *--        threshold (thresh)
               *--     comb+[0..8]
               *--        the comb filter is circularly shifted during the correlation
               *--        process
               *--        for this reason, when computing the comb filter output, only
               *--        comb+0 is filtered
               *--     registers
               *--        ACC holds temp after returning
               *--        the P and T registers are also modified
               *--        ar0, ar1, and arp are not modified
               *-- Other calls:
               *--        none
               *-- Cycle timing:
               *--        executes in 28 cycles (including call)
               *--
0375   6B1E    Hcorr    ltd       comb+7
0376   7F89             zac
0377   6D62             mpy       scorr+7
0378   6B1D             ltd       comb+6
0379   6D61             mpy       scorr+6
037A   6B1C             ltd       comb+5
037B   6D60             mpy       scorr+5
037C   6B1B             ltd       comb+4
037D   6D5F             mpy       scorr+4
```

```
 1630   037E   6B1A              ltd      comb+3
 1631   037F   6D5E              mpy      scorr+3
 1632   0380   6B19              ltd      comb+2
 1633   0381   6D5D              mpy      scorr+2
 1634   0382   6B18              ltd      comb+1
 1635   0383   6D5C              mpy      scorr+1
 1636   0384   6B17              ltd      comb+0
 1637   0385   6D5B              mpy      scorr+0
 1638   0386   7F8F              apac
 1639   0387   7F88              abs
 1640   0388   6265              subh     thresh
 1641   0389   5804              sach     temp
 1642   038A   201F              lac      comb+8
 1643   038B   5017              sacl     comb+0
 1644   038C   2004              lac      temp
 1645   038D   7F8D              ret
 1646                   *..
 1647                            page
```

```
                       copy      initcomb.prc
              *--
              *== Procedure Initcomb ====================================
              *-- Initialize comb buffer
              *--      - clear comb buffer (comb+[0..7])
              *--      - set comb memory to infinity (true averager)
              *--        alpha+0 = 0, alpha+1 = 1/16 Q15
              *--      - executes in 96 cycles
              *--      - need two locations free on stack before call
              *--
038E   7F89   Initcomb zac
038F   5063            sacl      alpha+0
0390   5017            sacl      comb+0
0391   5018            sacl      comb+1
0392   5019            sacl      comb+2
0393   501A            sacl      comb+3
0394   501B            sacl      comb+4
0395   501C            sacl      comb+5
0396   501D            sacl      comb+6
0397   501E            sacl      comb+7
0398   2B00            lac       one,11
0399   5064            sacl      alpha+1
039A   7E19            lack      25
039B   F800            call      Wdeln4
039C   064B
039D   7F8D            ret
              *--
                       page
```

```
 1675                               copy      initcorr.prc
 1676                     *..
 1677                     *== Procedure Initcorr ===================================
 1678                     *..       - initialize variables used during the analysis of
 1679                     *..         the comb buffer
 1680                     *..       - initialize the long correlator (used to estimate
 1681                     *..         initial clock phase) (lcorr+[0..31])
 1682                     *..       - initialize maximum (for correlator output) (corrmax)
 1683                     *..       - initialize correlator output threshold (corrthr)
 1684                     *..       - arp = 0 on return
 1685                     *-- get corelator
 1686   039E   6A00       Initcorr lt         one
 1687   039F   83AE                mpyk       Icorltr
 1688   03A0   7F8E                pac
 1689   03A1   701F                lark       ar0,31
 1690   03A2   7127                lark       ar1,lcorr+0
 1691   03A3   6881       Igetlc   larp       ar1
 1692   03A4   67A0                tblr       *+,ar0
 1693   03A5   0000                add        one
 1694   03A6   F400                banz       Igetlc
        03A7   03A3
 1695                     *-- get correlator threshold
 1696   03A8   83CE                mpyk       Icorthr
 1697   03A9   7F8E                pac
 1698   03AA   674B                tblr       corrthr
 1699                     *-- initialize correlator maximum
 1700   03AB   7F89                zac
 1701   03AC   504A                sacl       corrmax
 1702   03AD   7F8D                ret
 1703                     *-- correlator constants, (not stored in order)
 1704                     *--   corr(3,7,11,15,...,31)
 1705   03AE   26E4       Icorltr  data       9956,19725,16084,3333,-9893,-19809,-16147,-3246
        03AF   4D0D
        03B0   3ED4
        03B1   0D05
        03B2   D95B
        03B3   B29F
        03B4   C0ED
        03B5   F352
 1706                     *--   corr(2,6,10,14,...,30)
 1707   03B6   19CD                data       6605,18315,18320,6639,-6515,-18373,-18411,-6580
        03B7   478B
        03B8   4790
        03B9   19EF
        03BA   E68D
        03BB   B83B
        03BC   B815
        03BD   E64C
 1708                     *--   corr(1,5,9,13,...,29)
 1709   03BE   0CD9                data       3289,16074,19725,9980,-3186,-16094,-19828,-9959
```

```
        03BF    3ECA
        03C0    4D0D
        03C1    26FC
        03C2    F38E
        03C3    C122
        03C4    B28C
        03C5    D919
 1710                   *--  corr(0,4,8,12,...,28)
 1711   03C6    0018            data      24,13197,20201,13213,77,-13174,-20303,-13236
        03C7    338D
        03C8    4EE9
        03C9    339D
        03CA    004D
        03CB    CC8A
        03CC    B0B1
        03CD    CC4C
 1712                   *-- correlator threshold during analysis section
 1713   03CE    1388    Icorthr  data      5000
 1714                   *-- threshold (during sync hunt), comb memory and gain
 1715   03CF    32C8    Isthr    data      13000,1638,3277
        03D0    0666
        03D1    0CCD
 1716                   *--
 1717                           page
 1718                           copy      initdmd.prc
 1719                    *--
 1720                    *== Procedure Initdmd ===================================
 1721                    *-- Initialize demod
 1722                    *--        - clear memory allocated for demod
 1723                    *--        - set up last demodulated bits (for bit tracking)
 1724                    *--          lbits+[0..2]
 1725                    *--        - initialize rotation variable, phi <- aone + j0
 1726                    *--        - initialize demod threshold (hangover time) (dthr)
 1727                    *--        - initialize constants for hang over time
 1728                    *--          (dthr, f+[0..1])
 1729                    *--        - read in constants for MLSE
 1730                    *--        - arp = 0 on return
 1731                    *-- clear memory
 1732  03D2  7171   Initdmd  lark      ar1,tsum+1
 1733  03D3  704C            lark      ar0,tsum+1-dfmemih
 1734  03D4  7F89            zac
 1735  03D5  6881   Idclr    larp      ar1
 1736  03D6  5090            sacl      *-,0,ar0
 1737  03D7  F400            banz      Idclr
       03D8  03D5
 1738                    *-- load constants into DMEM
 1739  03D9  2F00            lac       one,15
 1740  03DA  501F            sacl      lbits+0
 1741  03DB  5020            sacl      lbits+1
 1742  03DC  5021            sacl      lbits+2
 1743                    *-- initialize current phase, (phi <- 1 + j0)
 1744  03DD  2001            lac       aone
 1745  03DE  5017            sacl      phi+0
 1746  03DF  7F89            zac
 1747  03E0  5018            sacl      phi+1
 1748                    *-- initialize constants for hang-over time estimation
 1749  03E1  6A00            lt        one
 1750  03E2  83F4            mpyk      Idthr
 1751  03E3  7F8E            pac
 1752  03E4  6722            tblr      dmdthr
 1753  03E5  0000            add       one
 1754  03E6  6723            tblr      f+0
 1755  03E7  0000            add       one
 1756  03E8  6724            tblr      f+1
 1757                    *-- read in constants for MLSE
 1758  03E9  6A00            lt        one
 1759  03EA  83F7            mpyk      Ivatble
 1760  03EB  7F8E            pac
 1761  03EC  671B            tblr      c+0
 1762  03ED  0000            add       one
 1763  03EE  671C            tblr      c+1
 1764  03EF  0000            add       one
 1765  03F0  671D            tblr      c+2
 1766  03F1  0000            add       one
```

```
1767   03F2  671E             tblr     c+3
1768   03F3  7F8D             ret
1769                   *-- demod threshold, constants for hangover period estimation (channel impulse response)
1770   03F4  36B0     Idthr   data     14000
1771   03F5  43D7             data     17367,7700     (f0 = 0.53 Q15, f1 = 0.235 Q15)
       03F6  1E14
1772                   *-- table for MLSE
1773   03F7  43D7     Ivatble data     17367,1966,9205,118
       03F8  07AE
       03F9  23F5
       03FA  0076
1774                   *--
1775                          page
```

```
 1776                            copy      initglob.prc
 1777                    *--
 1778                    *== Procedure Initglob ===================================
 1779                    *-- Initialize global variables on restart
 1780                    *--       - called only after restart
 1781                    *--       - set arp = 0, dp = 0, and set overflow mode (saturate)
 1782                    *--       - clear global memory
 1783                    *--       - initialize global variables (one, aone, root2)
 1784                    *--         one = 1
 1785                    *--         root2 = 23170 = 0.707092285 Q15
 1786                    *--         aone = 32767 = >7FFF = 0.999969482 Q15
 1787                    *--
 1788  03FB   7F8B       Initglob sovm
 1789  03FC   6880                larp      ar0
 1790  03FD   6E00                ldpk      dp0
 1791                    *-- clear global memory
 1792  03FE   7F89                zac
 1793  03FF   7016                lark      ar0,dmdorg-1
 1794  0400   5088       Igclr    sacl      *
 1795  0401   F400                banz      Igclr
       0402   0400
 1796                    *-- initialize global variables
 1797  0403   7E01                lack      1
 1798  0404   5000                sacl      one
 1799                    *--
 1800  0405   6A00                lt        one
 1801  0406   840E                mpyk      Iconsts
 1802  0407   7F8E                pac
 1803  0408   6701                tblr      aone
 1804  0409   0000                add       one
 1805  040A   6702                tblr      root2
 1806  040B   2B00                lac       one,11
 1807  040C   5003                sacl      pmemads
 1808  040D   7F8D                ret
 1809                    *-- the constant 0.99996948 (Q15, almost one), and sqrt(2)/2 Q(15)
 1810  040E   7FFF       Iconsts  data      >7FFF,23170
       040F   5A82
 1811                    *--
 1812                             page
```

```
                          copy      initsync.prc
                 *--
                 *== Procedure Initsync
                 *-- Initialize before synch hunt
                 *--      - clear memory used for synch hunt
                 *--      - initialize comb memory and gain (alpha+0, alpha+1)
                 *--      - initialize short correlator (scorr+[0..7])
                 *--      - initialize threshold
                 *--      - arp = 0 on return
                 *-- clear non global memory
0410   717F      Initsync lark      ar1,>7F
0411   7068               lark      ar0,>7F-dmdorg
                 *--
0412   7F89               zac
0413   6881      Isclr    larp      ar1
0414   5090               sacl      *-,0,ar0
0415   F400               banz      Isclr
0416   0413
                 *-- get constants from pmem (see proc Initcorr for pmem definitions)
0417   6A00               lt        one
0418   83AE               mpyk      Icorltr
0419   7F8E               pac
041A   7007               lark      ar0,7
041B   715B               lark      ar1,scorr+0
041C   6881      Igetsc   larp      ar1
041D   67A0               tblr      *+,ar0
041E   0000               add       one
041F   F400               banz      Igetsc
0420   041C
                 *-- read in threshold and comb memory and gain (from pmem)
0421   83CF               mpyk      Isthr
0422   7F8E               pac
0423   6765               tblr      thresh
0424   0000               add       one
0425   6763               tblr      alpha+0
0426   0000               add       one
0427   6764               tblr      alpha+1
0428   7F8D               ret
                 *--
                          page
```

```
                         copy      lpf.prc
                  *--
                  *== Procedure Lpf ========================================
                  *-- Low Pass Filter
                  *--      - input is at 8 times the bit rate
                  *--      - output is at the bit rate
                  *--      - filter is FIR, length = 56
                  *--      - filter is broken up into seven sections, each of length 8.
                  *--        This structure allows us to keep only the past 8 input samples
                  *--        and a few state variables in order to compute the filter output
                  *--
                  *-- Input:
                  *--     secbuf+[0..15]
                  *--        the past 8 inputs to the filter are held in secbuf, an array of
                  *--        8 real and complex samples. The values are stored in re, im format.
                  *--        i.e. the first value in the array (secbuf+0) is the real part of the
                  *--        first input, and the second value in the array (secbuf+1) is the
                  *--        complex part of the first input value
                  *--     dfmemil[0..5], dfmemql[0..5], dfmemih[0..5], dfmemqh[0..5]
                  *--        the state variables are 32 bits in length. The state variables are
                  *--        kept in locations dfmemih (real part, upper 16 bits), dfmemil (real part
                  *--        lower 16 bits) and dfmemqh (im part, upper 16 bits), dfmemql (im part,
                  *--        lower 16 bits). Each state variable has 6 components (since the
                  *--        filter has 7 sections).
                  *--     p0..p55 (constants defined by equates)
                  *--        constants p0-p55 are the filter coefficients (see file h.def)
                  *--
                  *-- Output:
                  *--     u+[0..1]
                  *--        the (complex) filter output is stored u+0 (real part) and
                  *--        u+1 (im part)
                  *--     dfmemil[0..5], dfmemql[0..5], dfmemih[0..5], dfmemqh[0..5]
                  *--        the state variables are updated during this procedure
                  *--     buffer+[1..3]
                  *--        3 input samples are read in during the procedure
                  *--     di+1
                  *--        Receiver clock (RC, bit 1) is set
                  *--     registers
                  *--        the accumulator, T and P registers are modified
                  *--        arp, ar0, and ar1 are not modified
                  *-- Other calls
                  *--        none
                  *-- Cycle timing
                  *--        on input, 122 cycles have elapsed since the last "in" (after
                  *--        the call statement
                  *--        on output, 28 cycles will have elapsed since the last "in", (after
                  *--        the return statement)
                  *--
                  *-- set RC (receiver clock)
0429    2100      Lpf      lac       one,1
```

```
042A  7A16              or       di+1
042B  5016              sacl     di+1           out instruction is done below
                *-- seventh (last) section (i channel)
                *-- load high 16 bits of ACC
042C  652A              zalh     dfmemih+5
                *-- input sample, set RC (do out instruction)
042D  4106              in       buffer+1,adc
042E  4A16              out      di+1,diprt
                *-- load low 16 bits of ACC
042F  7A30              or       dfmemil+5
0430  6A3D              lt       secbuf+0
0431  803C              mpyk     p7
0432  6C3F              lta      secbuf+2
0433  8054              mpyk     p6
0434  6C41              lta      secbuf+4
0435  8074              mpyk     p5
0436  6C43              lta      secbuf+6
0437  807B              mpyk     p4
0438  6C45              lta      secbuf+8
0439  8059              mpyk     p3
043A  6C47              lta      secbuf+10
043B  8039              mpyk     p2
043C  6C49              lta      secbuf+12
043D  8033              mpyk     p1
043E  6C4B              lta      secbuf+14
043F  8024              mpyk     p0
0440  6C3D              lta      secbuf+0
0441  590F              sach     u+0,1
                *-- sixth section (i channel)
0442  6529              zalh     dfmemih+4
0443  7A2F              or       dfmemil+4
0444  9E91              mpyk     p15
0445  6C3F              lta      secbuf+2
0446  9E62              mpyk     p14
0447  6C41              lta      secbuf+4
0448  9E80              mpyk     p13
0449  6C43              lta      secbuf+6
044A  9EC5              mpyk     p12
044B  6C45              lta      secbuf+8
044C  9F0A              mpyk     p11
044D  6C47              lta      secbuf+10
044E  9F5B              mpyk     p10
044F  6C49              lta      secbuf+12
0450  9FC6              mpyk     p9
0451  6C4B              lta      secbuf+14
0452  801A              mpyk     p8
0453  6C3D              lta      secbuf+0
0454  582A              sach     dfmemih+5
0455  5030              sacl     dfmemil+5
                *-- fifth section (i channel)
```

```
0456  6528            zalh     dfmemih+3
0457  7A2E            or       dfmemil+3
0458  8843            mpyk     p23
0459  6C3F            lta      secbuf+2
045A  86A7            mpyk     p22
045B  6C41            lta      secbuf+4
045C  84F4            mpyk     p21
045D  6C43            lta      secbuf+6
045E  8355            mpyk     p20
045F  6C45            lta      secbuf+8
0460  81DF            mpyk     p19
0461  6C47            lta      secbuf+10
0462  8099            mpyk     p18
0463  6C49            lta      secbuf+12
0464  9F9D            mpyk     p17
0465  6C4B            lta      secbuf+14
0466  9EF9            mpyk     p16
0467  6C3D            lta      secbuf+0
0468  5829            sach     dfmemih+4
0469  502F            sacl     dfmemil+4
                *-- fourth section (i channel)
046A  6527            zalh     dfmemih+2
046B  7A2D            or       dfmemil+2
046C  89BB            mpyk     p31
046D  6C3F            lta      secbuf+2
046E  8AFF            mpyk     p30
046F  6C41            lta      secbuf+4
0470  8BDE            mpyk     p29
0471  6C43            lta      secbuf+6
0472  8C40            mpyk     p28
0473  6C45            lta      secbuf+8
0474  8C40            mpyk     p27
0475  6C47            lta      secbuf+10
0476  8BDE            mpyk     p26
0477  6C49            lta      secbuf+12
0478  8AFF            mpyk     p25
0479  6C4B            lta      secbuf+14
047A  89BB            mpyk     p24
047B  6C3D            lta      secbuf+0
047C  5828            sach     dfmemih+3
047D  502E            sacl     dfmemil+3
                *-- third section (i channel)
047E  6526            zalh     dfmemih+1
047F  7A2C            or       dfmemil+1
0480  9EF9            mpyk     p39
0481  6C3F            lta      secbuf+2
0482  9F9D            mpyk     p38
0483  6C41            lta      secbuf+4
0484  8099            mpyk     p37
0485  6C43            lta      secbuf+6
```

```
0486  81DF             mpyk     p36
0487  6C45             lta      secbuf+8
0488  8355             mpyk     p35
0489  6C47             lta      secbuf+10
048A  84F4             mpyk     p34
048B  6C49             lta      secbuf+12
048C  86A7             mpyk     p33
048D  6C4B             lta      secbuf+14
048E  8843             mpyk     p32
048F  6C3D             lta      secbuf+0
0490  5827             sach     dfmemih+2
0491  502D             sacl     dfmemil+2
              *-- second section (i channel)
0492  6525             zalh     dfmemih+0
0493  7A2B             or       dfmemil+0
0494  801A             mpyk     p47
0495  6C3F             lta      secbuf+2
0496  9FC6             mpyk     p46
0497  6C41             lta      secbuf+4
0498  9F5B             mpyk     p45
0499  6C43             lta      secbuf+6
049A  9F0A             mpyk     p44
049B  6C45             lta      secbuf+8
049C  9EC5             mpyk     p43
049D  6C47             lta      secbuf+10
049E  9E80             mpyk     p42
049F  6C49             lta      secbuf+12
04A0  9E62             mpyk     p41
04A1  6C4B             lta      secbuf+14
04A2  9E91             mpyk     p40
04A3  6C3D             lta      secbuf+0
04A4  5826             sach     dfmemih+1
04A5  502C             sacl     dfmemil+1
              *-- first section (i channel)
04A6  7F89             zac
04A7  8024             mpyk     p55
04A8  6C3F             lta      secbuf+2
04A9  8033             mpyk     p54
04AA  6C41             lta      secbuf+4
              *- input sample 2
04AB  4107             in       buffer+2,adc
              *-
04AC  8039             mpyk     p53
04AD  6C43             lta      secbuf+6
04AE  8059             mpyk     p52
04AF  6C45             lta      secbuf+8
04B0  807B             mpyk     p51
04B1  6C47             lta      secbuf+10
04B2  8074             mpyk     p50
04B3  6C49             lta      secbuf+12
```

```
04B4  8054              mpyk    p49
04B5  6C4B              lta     secbuf+14
04B6  803C              mpyk    p48
04B7  6C3E              lta     secbuf+1
04B8  5825              sach    dfmemih+0
04B9  502B              sacl    dfmemil+0
                *-- seventh (last) section (q channel)
04BA  6536              zalh    dfmemqh+5
04BB  7A3C              or      dfmemql+5
04BC  803C              mpyk    p7
04BD  6C40              lta     secbuf+3
04BE  8054              mpyk    p6
04BF  6C42              lta     secbuf+5
04C0  8074              mpyk    p5
04C1  6C44              lta     secbuf+7
04C2  807B              mpyk    p4
04C3  6C46              lta     secbuf+9
04C4  8059              mpyk    p3
04C5  6C48              lta     secbuf+11
04C6  8039              mpyk    p2
04C7  6C4A              lta     secbuf+13
04C8  8033              mpyk    p1
04C9  6C4C              lta     secbuf+15
04CA  8024              mpyk    p0
04CB  6C3E              lta     secbuf+1
04CC  5910              sach    u+1,1
                *-- sixth section (q channel)
04CD  6535              zalh    dfmemqh+4
04CE  7A3B              or      dfmemql+4
04CF  9E91              mpyk    p15
04D0  6C40              lta     secbuf+3
04D1  9E62              mpyk    p14
04D2  6C42              lta     secbuf+5
04D3  9E80              mpyk    p13
04D4  6C44              lta     secbuf+7
04D5  9EC5              mpyk    p12
04D6  6C46              lta     secbuf+9
04D7  9F0A              mpyk    p11
04D8  6C48              lta     secbuf+11
04D9  9F5B              mpyk    p10
04DA  6C4A              lta     secbuf+13
04DB  9FC6              mpyk    p9
04DC  6C4C              lta     secbuf+15
04DD  801A              mpyk    p8
04DE  6C3E              lta     secbuf+1
04DF  5836              sach    dfmemqh+5
04E0  503C              sacl    dfmemql+5
                *-- fifth section (q channel)
04E1  6534              zalh    dfmemqh+3
04E2  7A3A              or      dfmemql+3
```

```
 2101   04E3   8843               mpyk      p23
 2102   04E4   6C40               lta       secbuf+3
 2103   04E5   86A7               mpyk      p22
 2104   04E6   6C42               lta       secbuf+5
 2105   04E7   84F4               mpyk      p21
 2106   04E8   6C44               lta       secbuf+7
 2107   04E9   8355               mpyk      p20
 2108   04EA   6C46               lta       secbuf+9
 2109   04EB   81DF               mpyk      p19
 2110   04EC   6C48               lta       secbuf+11
 2111   04ED   8099               mpyk      p18
 2112   04EE   6C4A               lta       secbuf+13
 2113   04EF   9F9D               mpyk      p17
 2114   04F0   6C4C               lta       secbuf+15
 2115   04F1   9EF9               mpyk      p16
 2116   04F2   6C3E               lta       secbuf+1
 2117   04F3   5835               sach      dfmemqh+4
 2118   04F4   503B               sacl      dfmemql+4
 2119                      *-- fourth section (q channel)
 2120   04F5   6533               zalh      dfmemqh+2
 2121   04F6   7A39               or        dfmemql+2
 2122   04F7   89BB               mpyk      p31
 2123   04F8   6C40               lta       secbuf+3
 2124   04F9   8AFF               mpyk      p30
 2125   04FA   6C42               lta       secbuf+5
 2126   04FB   8BDE               mpyk      p29
 2127   04FC   6C44               lta       secbuf+7
 2128   04FD   8C40               mpyk      p28
 2129   04FE   6C46               lta       secbuf+9
 2130   04FF   8C40               mpyk      p27
 2131   0500   6C48               lta       secbuf+11
 2132   0501   8BDE               mpyk      p26
 2133   0502   6C4A               lta       secbuf+13
 2134   0503   8AFF               mpyk      p25
 2135   0504   6C4C               lta       secbuf+15
 2136   0505   89BB               mpyk      p24
 2137   0506   6C3E               lta       secbuf+1
 2138   0507   5834               sach      dfmemqh+3
 2139   0508   503A               sacl      dfmemql+3
 2140                      *-- third section (q channel)
 2141   0509   6532               zalh      dfmemqh+1
 2142   050A   7A38               or        dfmemql+1
 2143   050B   9EF9               mpyk      p39
 2144   050C   6C40               lta       secbuf+3
 2145   050D   9F9D               mpyk      p38
 2146   050E   6C42               lta       secbuf+5
 2147   050F   8099               mpyk      p37
 2148   0510   6C44               lta       secbuf+7
 2149   0511   81DF               mpyk      p36
 2150   0512   6C46               lta       secbuf+9
```

```
0513   8355            mpyk     p35
0514   6C48            lta      secbuf+11
0515   84F4            mpyk     p34
0516   6C4A            lta      secbuf+13
0517   86A7            mpyk     p33
0518   6C4C            lta      secbuf+15
0519   8843            mpyk     p32
051A   6C3E            lta      secbuf+1
051B   5833            sach     dfmemqh+2
051C   5039            sacl     dfmemql+2
               *-- second section (q channel)
051D   6531            zalh     dfmemqh+0
051E   7A37            or       dfmemql+0
051F   801A            mpyk     p47
0520   6C40            lta      secbuf+3
0521   9FC6            mpyk     p46
0522   6C42            lta      secbuf+5
0523   9F5B            mpyk     p45
0524   6C44            lta      secbuf+7
0525   9F0A            mpyk     p44
0526   6C46            lta      secbuf+9
0527   9EC5            mpyk     p43
0528   6C48            lta      secbuf+11
0529   9E80            mpyk     p42
               *- input sample 3
052A   4108            in       buffer+3,adc
               *-
052B   6C4A            lta      secbuf+13
052C   9E62            mpyk     p41
052D   6C4C            lta      secbuf+15
052E   9E91            mpyk     p40
052F   6C3E            lta      secbuf+1
0530   5832            sach     dfmemqh+1
0531   5038            sacl     dfmemql+1
               *-- first section (q channel)
0532   7F89            zac
0533   8024            mpyk     p55
0534   6C40            lta      secbuf+3
0535   8033            mpyk     p54
0536   6C42            lta      secbuf+5
0537   8039            mpyk     p53
0538   6C44            lta      secbuf+7
0539   8059            mpyk     p52
053A   6C46            lta      secbuf+9
053B   807B            mpyk     p51
053C   6C48            lta      secbuf+11
053D   8074            mpyk     p50
053E   6C4A            lta      secbuf+13
053F   8054            mpyk     p49
0540   6C4C            lta      secbuf+15
```

```
0541   803C             mpyk      p48
0542   7F8F             apac
0543   5831             sach      dfmemqh+0
0544   5037             sacl      dfmemql+0
0545   7F8D             ret
                   *..
                        page
```

```
                     copy     normlz.prc
            *--
            *== Procedure Normlz ====================================
            *-- Normalize
            *--       - the derotation variable (phi) is normalized
            *--         to unit radius using the formula
            *--            phi <- phi*(3 - |phi|^2)/2
            *--         or
            *--            phi <- phi/2 + phi*(1.0 - |phi|^2/2)
            *-- Input:
            *--      phi+[0..1]
            *--         phi contains the derotation variable
            *--      aone
            *--         contains the constant >7FFF = 32767 = 0.999969482 Q15
            *-- Output
            *--      phi+[0..1]
            *--         normalized using one pass through recursive routine
            *--      temp
            *--         location temp is modified
            *-- Cycle timing
            *--         executes in 20 cycles, (including call)
            *--
            *-- compute temp = 1.0 - |phi|^2/2
0546  6501  Normlz   zalh     aone
0547  6A18           lt       phi+1
0548  6D18           mpy      phi+1
0549  7F90           spac
054A  6A17           lt       phi+0
054B  6D17           mpy      phi+0
054C  7F90           spac
054D  5804           sach     temp
            *-- compute re(phi)/2 + re(phi)*temp
054E  2E17           lac      phi+0,14
054F  6D04           mpy      temp
0550  6C18           lta      phi+1
0551  5917           sach     phi+0,1
            *-- compute im(phi)/2 + im(phi)*temp
0552  2E18           lac      phi+1,14
0553  6D04           mpy      temp
0554  7F8F           apac
0555  5918           sach     phi+1,1
0556  7F8D           ret
            *--
                     page
```

```
                        copy     stt.prc
                  *--
                  *== Procedure Stt ========================================
                  *-- Symbol Transition Tracker, and end of transmission determination
                  *--      - bit tracking in demod mode
                  *--      - determine end of transmission
                  *--      - compute error between received samples and samples
                  *--        received in the absence of noise (using decision
                  *--        feedback)
                  *--      - average error in first order low pass
                  *--        (if average error exceeds a threshold, transmission
                  *--         is over, see demod.tms)
                  *--      - if transition is observed (lrecd+0 not equal to lrecd+2)
                  *--        then step bit tracking algorithm
                  *--      - shift last received differential detector outputs (ly+[0..13]
                  *--
                  *-- Input:
                  *--     recd+0
                  *--        the bit just released from the MLSE (0 or 1)
                  *--     lbits+[0..2]
                  *--        the last received bits, (+/- 32767)
                  *--        lbits+0 <=> recd+0
                  *--     ly+[0..14]
                  *--        the last outputs of the differential detector
                  *--        (ly+14 corresponds to recd+1)
                  *--     f+[0..1]
                  *--        channel impulse response coefficients
                  *--        (scaled by factor of two)
                  *-- Output:
                  *--     temp
                  *--        location temp is modified
                  *--     yavg
                  *--        average error is updated
                  *--     tsum+[0..1]
                  *--        32 bit timing error average is updated if
                  *--        a transition is observed
                  *--     lbits+[0..2]
                  *--        lbits+0 is updated, and the old values are shifted
                  *-- Other calls:
                  *--        none
                  *-- Cycle Timing
                  *--        executes in 126 cycles (including call)
                  *--
                  *-- lbits+0 <- +/- 32767
0557    2F00      Stt       lac      one,15
0558    0000                add      one
0559    115E                sub      recd+0,1
055A    501F                sacl     lbits+0
                  *-- calculate error given last bits and received sample
055B    656E                zalh     ly+14
```

```
 2302   055C   6A23              lt       f+0           (recall: scale of two in f+0 and f+1)
 2303   055D   6D20              mpy      lbits+1
 2304   055E   7F90              spac
 2305   055F   6A24              lt       f+1
 2306   0560   6D1F              mpy      lbits+0
 2307   0561   7F90              spac
 2308   0562   6D21              mpy      lbits+2
 2309   0563   7F90              spac
 2310   0564   5804              sach     temp
 2311                  *-- average error, (for hang over determination)
 2312   0565   2B04              lac      temp,11
 2313   0566   7F88              abs
 2314   0567   606F              addh     yavg
 2315   0568   1A6F              sub      yavg,10
 2316   0569   586F              sach     yavg
 2317                  *-- determine if a transition was observed
 2318   056A   201F              lac      lbits+0
 2319   056B   1021              sub      lbits+2
 2320   056C   FE00              bnz      Strans
        056D   0571
 2321                  *-- no transition branch to delay below
 2322   056E   7026              lark     ar0,38
 2323   056F   F900              b        Sdel
        0570   0581
 2324                  *-- temp holds the timing error
 2325                  *-- calculate timing error
 2326   0571   6A04    Strans    lt       temp
 2327   0572   6D21              mpy      lbits+2
 2328   0573   7F8E              pac
 2329   0574   5904              sach     temp,1
 2330                  *-- first order lopass
 2331   0575   6570              zalh     tsum+0
 2332   0576   7A71              or       tsum+1
 2333   0577   1E70              sub      tsum+0,14
 2334   0578   0E04              add      temp,14
 2335   0579   5071              sacl     tsum+1
 2336   057A   5870              sach     tsum+0
 2337                  *-- advance or retard phase according to filter output
 2338   057B   7F80              nop
 2339   057C   FA00              blz      Sbmp
        057D   0580
 2340   057E   7F80              nop
 2341   057F   7F80              nop
 2342   0580   7020    Sbmp      lark     ar0,32
 2343                  *-- delay
 2344   0581   F400    Sdel      banz     $
        0582   0581
 2345                  *-- save last bits
 2346   0583   6920              dmov     lbits+1
 2347   0584   691F              dmov     lbits+0
```

```
                 *-- save last received values
0585   696D             dmov      ly+13
0586   696C             dmov      ly+12
0587   696B             dmov      ly+11
0588   696A             dmov      ly+10
0589   6969             dmov      ly+9
058A   6968             dmov      ly+8
058B   6967             dmov      ly+7
058C   6966             dmov      ly+6
058D   6965             dmov      ly+5
058E   6964             dmov      ly+4
058F   6963             dmov      ly+3
0590   6962             dmov      ly+2
0591   6961             dmov      ly+1
0592   6960             dmov      ly+0
                 *-- first value
0593   2014             lac       y
0594   5060             sacl      ly+0
                 *-- c'est finit
0595   7F8D             ret
                        page
```

```
         copy      va.prc
*--
*== Procedure Va ==========================================
*-- Viterbi Algorithm- Maximum Likelihood Sequence Estimator (MLSE)
*--      - the "brains" of the modem
*--      - compute new metrics from old metrics and received
*--        differential phase.
*--      - compute bestpaths (need maximum of metrics)
*--      - compute most likely received data (assuming gaussian
*--        noise at output of differential detector)
*--      - re-normalize metrics (need maximum of metrics)
*--      - differential decoding
*--      - see document ' ' for a detailed description
*--      - channel impulse response f
*--                y = (f(-1)a(k+1) + f(0)a(k) + f(1)a(k-1))/2
*--                f(1) = f(-1)
*--                f(0) + 2*f(1) = 1
*--
*-- Input:
*--     c+[0..3]
*--        c+0 = f(0)              Q15
*--        c+1 = f(0)-2f(1)        Q15
*--        c+2 = f(0)^2            Q15
*--        c+3 = (f(0)-2f(1))^2    Q15
*--     mu+[0..7]
*--        state metrics, (or weights)
*--        two state viterbi, only need four metrics
*--        mu+1 is a backup for mu+0, mu+3 for mu+2,.., mu+7 for mu+6
*--     bp+[0..7]
*--        best paths corresponding to the metrics
*--        bp+1 is a backup for bp+0, bp+3 for bp+2,.., bp+7 for bp+6
*--     mumax
*--        the maximum of the metrics
*--     recd+1
*--        last demodulated bit
*--     y
*--        y holds the differential detector output (the input to the VA)
*-- Output:
*--     mu+[0..7]
*--        holds the metrics and copies
*--     mumax
*--        holds the new maximum of the metrics
*--     bp+[0..7]
*--        best paths
*--     recd+[0,1]
*--        demodulated bit
*--     di+1
*--        di+1 holds the differentially decoded demodulated bit
*--     registers
*--        ar0 and ar1 are modified
```

```
 2419                   *--        arp = 0 on exit
 2420                   *--     ports
 2421                   *--        the demodulated differentially decoded bit is output to
 2422                   *--        the data interface (bit 0). the receiver clock is cleared
 2423                   *--        (bit one)
 2424                   *-- Other calls:
 2425                   *--        none
 2426                   *-- Cycle timing:
 2427                   *--        executes in 126 cycles (including call)
 2428                   *--
 2429                   *-- load T register with differential detector output
 2430  0596  6A14       Va        lt        y
 2431                   *-- keep copy of best paths, and last metrics
 2432  0597  6956                 dmov      bp+0
 2433  0598  6958                 dmov      bp+2
 2434  0599  695A                 dmov      bp+4
 2435  059A  695C                 dmov      bp+6
 2436  059B  694E                 dmov      mu+0
 2437  059C  6950                 dmov      mu+2
 2438  059D  6952                 dmov      mu+4
 2439  059E  6954                 dmov      mu+6
 2440                   *-- a(k),a(k-1) a(k-2)
 2441                   *-- the new first metric (mu+0) has two possible predecessors
 2442                   *-- +1,+1,+1
 2443  059F  654F                 zalh      mu+1
 2444  05A0  624D                 subh      mumax
 2445  05A1  0F14                 add       y,15
 2446  05A2  1D01                 sub       aone,13
 2447  05A3  580D                 sach      z+0
 2448                   *-- +1,+1,-1
 2449  05A4  6551                 zalh      mu+3
 2450  05A5  624D                 subh      mumax
 2451  05A6  6D1B                 mpy       c+0
 2452  05A7  7F8F                 apac
 2453  05A8  1D1D                 sub       c+2,13
 2454  05A9  580E                 sach      z+1
 2455                   *-- compute metric 0
 2456  05AA  620D                 subh      z+0
 2457  05AB  FA00                 blz       Vsw0
       05AC  05B0
 2458                   *--
 2459  05AD  2159                 lac       bp+3,1
 2460  05AE  F900                 b         Vendmu0
       05AF  05B3
 2461                   *--
 2462  05B0  690D       Vsw0      dmov      z+0
 2463  05B1  2157                 lac       bp+1,1
 2464  05B2  7F80                 nop
 2465                   *--
 2466  05B3  7A00       Vendmu0   or        one
```

```
05B4  5056                  sacl    bp+0
05B5  200E                  lac     z+1
05B6  504E                  sacl    mu+0
                *-- the new second metric (mu+2) has two possible predecessors
                *-- +1,-1,+1
05B7  6553                  zalh    mu+5
05B8  624D                  subh    mumax
05B9  6D1C                  mpy     c+1
05BA  7F90                  spac
05BB  1D1E                  sub     c+3,13
05BC  580D                  sach    z+0
                *-- +1,-1,-1
05BD  6555                  zalh    mu+7
05BE  624D                  subh    mumax
05BF  6D1B                  mpy     c+0
05C0  7F90                  spac
05C1  1D1D                  sub     c+2,13
05C2  580E                  sach    z+1
                *-- compute metric 1
05C3  620D                  subh    z+0
05C4  FA00                  blz     Vsw1
05C5  05C9
                *--
05C6  215D                  lac     bp+7,1
05C7  F900                  b       Vendmu1
05C8  05CC
                *--
05C9  690D      Vsw1        dmov    z+0
05CA  215B                  lac     bp+5,1
05CB  7F80                  nop
                *--
05CC  7A00      Vendmu1     or      one
05CD  5058                  sacl    bp+2
05CE  200E                  lac     z+1
05CF  5050                  sacl    mu+2
                *-- the new third metric (mu+4) has two possible predecessors
                *-- -1,+1,+1
05D0  654F                  zalh    mu+1
05D1  624D                  subh    mumax
05D2  6D1B                  mpy     c+0
05D3  7F8F                  apac
05D4  1D1D                  sub     c+2,13
05D5  580D                  sach    z+0
                *-- -1,+1,-1
05D6  6551                  zalh    mu+3
05D7  624D                  subh    mumax
05D8  6D1C                  mpy     c+1
05D9  7F8F                  apac
05DA  1D1E                  sub     c+3,13
05DB  580E                  sach    z+1
```

```
                    *-- compute metric 2
05DC   620D                 subh      z+0
05DD   FA00                 blz       Vsw2
05DE   05E2
                    *--
05DF   2159                 lac       bp+3,1
05E0   F900                 b         Vendmu2
05E1   05E5
                    *--
05E2   690D         Vsw2    dmov      z+0
05E3   2157                 lac       bp+1,1
05E4   7F80                 nop
                    *--
05E5   505A         Vendmu2 sacl      bp+4
05E6   200E                 lac       z+1
05E7   5052                 sacl      mu+4
                    *-- the new fourth metric (mu+6) has two possible predecessors
                    *-- -1,-1,+1
05E8   6553                 zalh      mu+5
05E9   624D                 subh      mumax
05EA   6D1B                 mpy       c+0
05EB   7F90                 spac
05EC   1D1D                 sub       c+2,13
05ED   580D                 sach      z+0
                    *-- -1,-1,-1
05EE   6555                 zalh      mu+7
05EF   624D                 subh      mumax
05F0   1F14                 sub       y,15
05F1   1D01                 sub       aone,13
05F2   580E                 sach      z+1
                    *-- compute metric 3
05F3   620D                 subh      z+0
05F4   FA00                 blz       Vsw3
05F5   05F9
                    *--
05F6   215D                 lac       bp+7,1
05F7   F900                 b         Vendmu3
05F8   05FC
                    *--
05F9   690D         Vsw3    dmov      z+0
05FA   215B                 lac       bp+5,1
05FB   7F80                 nop
                    *--
05FC   505C         Vendmu3 sacl      bp+6
05FD   200E                 lac       z+1
05FE   5054                 sacl      mu+6
                    *-- - find maximum values of metrics
05FF   204E                 lac       mu+0
0600   1050                 sub       mu+2
0601   FA00                 blz       Vmu1lgr
```

```
      0602  0623
2561              *-- mu+0 > mu+2
2562  0603  2052  Vmu0lgr  lac      mu+4
2563  0604  1054           sub      mu+6
2564  0605  FA00           blz      Vmu03lgr
      0606  0615
2565              *-- & mu+4 > mu+6
2566  0607  204E  Vmu04lgr lac      mu+0
2567  0608  1052           sub      mu+4
2568  0609  FA00           blz      Vmu2max
      060A  0610
2569              *-- mu+0 is the max
2570  060B  6656           zals     bp+0
2571  060C  595E           sach     recd+0,1
2572  060D  204E           lac      mu+0
2573  060E  F900           b        Vrelbit
      060F  0639
2574              *-- mu+4 is the max
2575  0610  665A  Vmu2max  zals     bp+4
2576  0611  595E           sach     recd+0,1
2577  0612  2052           lac      mu+4
2578  0613  F900           b        Vrelbit
      0614  0639
2579              *-- mu+6 > mu+4
2580  0615  204E  Vmu03lgr lac      mu+0
2581  0616  1054           sub      mu+6
2582  0617  FA00           blz      Vmu3max
      0618  061E
2583              *-- mu+0 is the max
2584  0619  6656           zals     bp+0
2585  061A  595E           sach     recd+0,1
2586  061B  204E           lac      mu+0
2587  061C  F900           b        Vrelbit
      061D  0639
2588              *-- mu+6 is the max
2589  061E  665C  Vmu3max  zals     bp+6
2590  061F  595E           sach     recd+0,1
2591  0620  2054           lac      mu+6
2592  0621  F900           b        Vrelbit
      0622  0639
2593              *-- mu+2 > mu+0
2594  0623  2052  Vmu1lgr  lac      mu+4
2595  0624  1054           sub      mu+6
2596  0625  FA00           blz      Vmu13lgr
      0626  0630
2597              *-- mu+4 > mu+6
2598  0627  2050  Vmu12lgr lac      mu+2
2599  0628  1052           sub      mu+4
2600  0629  FA00           blz      Vmu2max
      062A  0610
```

```
 2601                    *-- mu+2 is the max
 2602   062B   6658              zals      bp+2
 26D3   062C   595E              sach      recd+0,1
 2604   062D   2050              lac       mu+2
 2605   062E   F900              b         Vrelbit
        062F   0639
 2606                    *--
 2607   0630   2050      Vmu13lgr lac      mu+2
 2608   0631   1054              sub       mu+6
 2609   0632   FA00              blz       Vmu3max
        0633   061E
 2610                    *-- mu+2 is the max
 2611   0634   6658              zals      bp+2
 2612   0635   595E              sach      recd+0,1
 2613   0636   2050              lac       mu+2
 2614   0637   F900              b         Vrelbit
        0638   0639
 2615                    *-- recalculate maximum
 2616   0639   1E00      Vrelbit  sub      one,14
 2617   063A   504D              sacl      mumax
 2618                    *-- differential decode
 2619   063B   205E              lac       recd+0
 2620   063C   785F              xor       recd+1
 2621   063D   695E              dmov      recd+0
 2622   063E   5016              sacl      di+1
 2623   063F   4A16              out       di+1,diprt
 2624                    *-- waste a few cycles before exit
 2625   0640   7F80              nop
 2626   0641   7F80              nop
 2627   0642   7F80              nop
 2628   0643   7F80              nop
 2629   0644   7F8D              ret
 2630                            page
```

```
                      copy     wdel.prc
             *-- title: wdel.tms
             *--
             *--  Delay Procedures
             *--      - delay a number of cycles
             *--      - some procedures require a repeat count in the accumulator
             *--        on invocation (Wdeln4, Wdeln5 and Wdeln6)
             *--      - the accumulator is also modified in Wdeln4, Wdeln5 and Wdeln6
             *--
             *-- Procedures to delay N cycles where N is in [4..7]
             *--
             *== Procedure Wdel7 ======================================
             *--      - delay 7 cycles, (including call)
             *--      - algorithm: waste one cycle and fall through to Wdel6
             *--        (which wastes 6 cycles)
0645  7F80   Wdel7    nop
             *--
             *== Procedure Wdel6 ======================================
             *--      - delay 6 cycles, (including call)
             *--      - algorithm: waste one cycle and fall through to Wdel5
             *--        (which wastes 5 cycles)
0646  7F80   Wdel6    nop
             *--
             *== Procedure Wdel5 ======================================
             *--      - delay 5 cycles, (including call)
             *--      - algorithm: waste one cycle and fall through to Wdel4
             *--        (which wastes 4 cycles)
0647  7F80   Wdel5    nop
             *--
             *== Procedure Wdel4 ======================================
             *--      - delay 4 cycles, (including call)
             *--      - algorithm: just return
0648  7F8D   Wdel4    ret
             *--
             *--
             *-- Procedures to delay N cycles where N is in [8...]
             *--
             *--
             *== Procedure Wdeln6 =====================================
             *--      - delay 6 + ACC*3 cycles (including call), (ACC > 0)
             *--      - algorithm: delay one cycle, and fall through to Wdeln5
             *--        (which wastes 5+ACC*3 cycles)
             *--      - only ACC is modified
             *--
0649  7F80   Wdeln6   nop
             *--
             *== Procedure Wdeln5 =====================================
             *--      - delay 5 + ACC*3 cycles, (ACC > 0)
             *--      - algorithm: delay one cycle, and fall through to Wdeln5
             *--        (which wastes 5+ACC*3 cycles)
```

```
                *--      - only ACC is modified
                *--
064A   7F80     Wdeln5   nop
                *--
                *== Procedure Wdeln4 ====================================
                *--      - delay 4 + ACC*3 cycles, (ACC > 0)
                *--      - algorithm: count down ACC to zero and return
                *--      - only ACC is modified
                *--
064B   1000     Wdeln4   sub       one
064C   FE00              bnz       Wdeln4         \3*ACC
064D   064B
064E   7F8D              ret                      \2+2 (for call) = 4+3*ACC, (ACC > 0)
                *--
                *--
                         pend
                         end
```

```
*IDT*     GTFM(0.62,0.36) Demodulator     TMS320 Assembler vers 1.36

User Defined Symbols


    AR0         0000    AR1         0001    Aendrot     00D2    Ah1         0067
    Ah2         0070    Ah3         0080    Ah4         008A    Ah5         009D
    Ah6         00AA    Ah7         00BA    Ah8         00C5    Aquad1      0064
    Aquad2      007D    Aquad3      009A    Aquad34     0097    Aquad4      00B7
    Argument    005D    Bpfilt      0106    C0          024D    Corr        0231
    Corrend     0271    Cswtch      024B    Czc0        0257    Czcnfnd     026B
    Czcross     0260    Derot       0272    Eshft       0288    Facorrzc    02E2
    Fcalcdel    0315    Fcldel      0323    Fcoarse     02A8    Fcostbl     0354
    Fnegarg     02A3    Freqoff     0293    Fsintbl     0333    Fzcdiv      0300
    Fztc1       0306    Fztc2       030D    Hcorr       0375    Iconsts     040E
    Icorltr     03AE    Icorthr     03CE    Idclr       03D5    Idthr       03F4
    Igclr       0400    Igetlc      03A3    Igetsc      041C    Initcomb    038E
    Initcorr    039E    Initdmd     03D2    Initglob    03FB    Initsync    0410
    Isclr       0413    Isthr       03CF    Ivatble     03F7    Lpf         0429
    Normlz      0546    PA0         0000    PA1         0001    PA2         0002
    PA3         0003    PA4         0004    PA5         0005    PA6         0006
    PA7         0007    Sbmp        0580    Sdel        0581    Strans      0571
    Stt         0557    Va          0596    Vendmu0     05B3    Vendmu1     05CC
    Vendmu2     05E5    Vendmu3     05FC    Vmu03lgr    0615    Vmu04lgr    0607
    Vmu0lgr     0603    Vmu12lgr    0627    Vmu13lgr    0630    Vmu1lgr     0623
    Vmu2max     0610    Vmu3max     061E    Vrelbit     0639    Vsw0        05B0
    Vsw1        05C9    Vsw2        05E2    Vsw3        05F9    Wdel4       0648
    Wdel5       0647    Wdel6       0646    Wdel7       0645    Wdeln4      064B
    Wdeln5      064A    Wdeln6      0649    adc         0001    afmemih     0027
    afmemil     0034    afmemqh     0041    afmemql     004E    alpha       0063
    aone        0001    ar0         0000    ar1         0001    arg         0011
    bp          0056    buffer      0005    c           001B    comb        0017
    combfilt    0017    cororg      0027    corrmax     004A    corrthr     004B
    cos         0051    count       004D    demod       002A    dfmemih     0025
    dfmemil     002B    dfmemqh     0031    dfmemql     0037    di          0015
    diprt       0002    dmdorg      0017    dmdthr      0022    dp0         0000
    f           0023    fo          0019    intprt      0002    lbits       001F
    lcorr       0027    ly          0060    mu          004E    mumax       004D
    one         0000    p0          0024    p1          0033    p10         FF5B
    p11         FF0A    p12         FEC5    p13         FE80    p14         FE62
    p15         FE91    p16         FEF9    p17         FF9D    p18         0099
    p19         01DF    p2          0039    p20         0355    p21         04F4
    p22         06A7    p23         0843    p24         09BB    p25         0AFF
    p26         0BDE    p27         0C40    p28         0C40    p29         0BDE
    p3          0059    p30         0AFF    p31         09BB    p32         0843
    p33         06A7    p34         04F4    p35         0355    p36         01DF
    p37         0099    p38         FF9D    p39         FEF9    p4          007B
    p40         FE91    p41         FE62    p42         FE80    p43         FEC5
    p44         FF0A    p45         FF5B    p46         FFC6    p47         001A
    p48         003C    p49         0054    p5          0074    p50         0074
    p51         007B    p52         0059    p53         0039    p54         0033
    p55         0024    p6          0054    p7          003C    p8          001A
    p9         FFC6   phi        0017   pi0        FFDC   pi1        FFDC
    pi10       0000   pi11       FF52   pi12       FEC5   pi13       FEF0
    pi14       0000   pi15       0104   pi16       0107   pi17       0046
    pi18       0000   pi19       0152   pi2        0000   pi20       0355
    pi21       0381   pi22       0000   pi23       FA28   pi24       F645
    pi25       F839   pi26       0000   pi27       08A9   pi28       0C40
    pi29       0864   pi3        003F   pi30       0000   pi31       F91E
    pi32       F7BD   pi33       FB4C   pi34       0000   pi35       025B
    pi36       01DF   pi37       006C   pi38       0000   pi39       00BA
    pi4        007B   pi40       016F   pi41       0124   pi42       0000
    pi43       FF21   pi44       FF0A   pi45       FF8C   pi46       0000
    pi47       FFEE   pi48       FFC4   pi49       FFC5   pi5        0052
    pi50       0000   pi51       0057   pi52       0059   pi53       0028
    pi54       0000   pi55       FFE7   pi6        0000   pi7        FFD6
    pi8        FFE6   pi9        0029   pmemads    0003   pq0        0000
    pq1        FFDC   pq10       00A5   pq11       00AE   pq12       0000
    pq13       FEF0   pq14       FE62   pq15       FEFC   pq16       0000
    pq17       0046   pq18       FF67   pq19       FEAE   pq2        FFC7
    pq20       0000   pq21       0381   pq22       06A7   pq23       05D8
    pq24       0000   pq25       F839   pq26       F422   pq27       F757
    pq28       0000   pq29       0864   pq3        FFC1   pq30       0AFF
    pq31       06E2   pq32       0000   pq33       FB4C   pq34       FB0C
    pq35       FDA5   pq36       0000   pq37       006C   pq38       FF9D
    pq39       FF46   pq4        0000   pq40       0000   pq41       0124
    pq42       0180   pq43       00DF   pq44       0000   pq45       FF8C
    pq46       FFC6   pq47       0012   pq48       0000   pq49       FFC5
    pq5        0052   pq50       FF8C   pq51       FFA9   pq52       0000
    pq53       0028   pq54       0033   pq55       0019   pq6        0054
    pq7        002A   pq8        0000   pq9        0029   recd       005E
    restart    0000   root2      0002   scorr      005B   secbuf     003D
    setcomb    0012   sin        004F   synch      0003   syncstrt   0005
    tcorr      0047   temp       0004   thresh     0065   tsum       0070
    u          000F   y          0014   yavg       006F   z          000D
    zcflag     004C
```