Scenix Semiconductor, Inc.
V.23 Originate Mode Reference Design

Document V. 0.991

July 5, 1999
Chris Fogelklou
Applications Engineer
Scenix Semiconductor
http://www.scenix.com

TABLE OF CONTENTS

INTRODUCTION

The subject of this overview is the Scenix V.23 modem reference design. Scenix Semiconductor, Inc., has completed a design for a V.23 modem, operating in originate mode, to suit the needs of some of its customers. This document describes the operation of each block of hardware, in the circuit schematics, and the operation of most of the software modules.

Although modem standards have advanced enormously since V.23 was first developed in the 1960's, the standard still exists today. Typically, these applications utilizing V.23 do not require the high data rates provided by modern modems, and significant cost savings can be achieved by choosing a lower-speed, lower-cost design.

The goal of the Scenix V.23 reference design is to allow the end user, typically an engineer, the capability to quickly embed the Scenix solution into any design that requires the use of a low-speed modem. Typical applications that would benefit from a low-cost, low-speed modem include credit-card readers, remote monitoring equipment, alarm systems, set-top-boxes, etc.

This document contains descriptions of the hardware and software used to perform V.23 origination with a Scenix SX processor. This software/hardware has been tested and FCC approved on the Scenix Modem Demo-Board, revision 1.2. The demo board was designed to perform Caller-ID detection, DTMF and Call-Progress detection, Bell 103 and V.23 modes of modem communication. Presently, only the software for V.23 origination has been completed. For a minimal V.23 origination schematic, see Appendix D. The demo-board schematics are contained in Appendix C.

SPECIFICATIONS

This V.23 reference design is designed to meet these specifications:

Universal Asynchronous Receiver/Transmitter

- 1200 bps
- No Parity
- 8 Data Bits
- 1 Stop Bit
- Hardware Flow Control with 16-byte transmit buffer (CTS, RTS)

Compact AT command set

- 64-byte command buffer
- Dial: “ATDTxxxxxxxxx…”
- Switch from data mode to command mode: “+++”
- Switch from command mode to data mode: “ATO”
- Hang up: “ATH”
- Initialize: “ATZ”
- Hybrid Optimization “ATY”

Dual-Tone Multiple-Frequency Generation for Dialing

- Tones generated: 697Hz, 770Hz, 852Hz, 941Hz, 1209Hz, 1336Hz, 1477Hz, 1633Hz, ± 0.5Hz
- On time = 100ms
- Off time = 100ms
- Off-hook delay time before dialing = 4 s
- D/A conversion provided by filtered PPM output

Data transmission and modulation

- FSK transmission data rate at 75bps
- Hardware flow control, 16-byte buffer, and 75bps asynchronous transmitter for data rate conversion from 1200bps to 75bps
- Logic ‘1’ (mark) modulated by 390 Hz
- Logic ‘0’ (space) modulated by 450 Hz
- Transmission power = -15dB
- D/A conversion provided by filtered PPM output

Data reception and demodulation

- FSK reception data rate at 1200bps
- Logic ‘1’ (mark) demodulated from 1300Hz carrier
- Logic ‘0’ (space) demodulated from 2100Hz carrier
- Carrier detection
- Timed-Zero-Cross algorithm
- Carrier Detection

D/A conversion

- Pulse Position Modulation with maximum output frequency of 154kHz

Filtering

- Low pass filter on PPM output
- High pass filter on FSK input

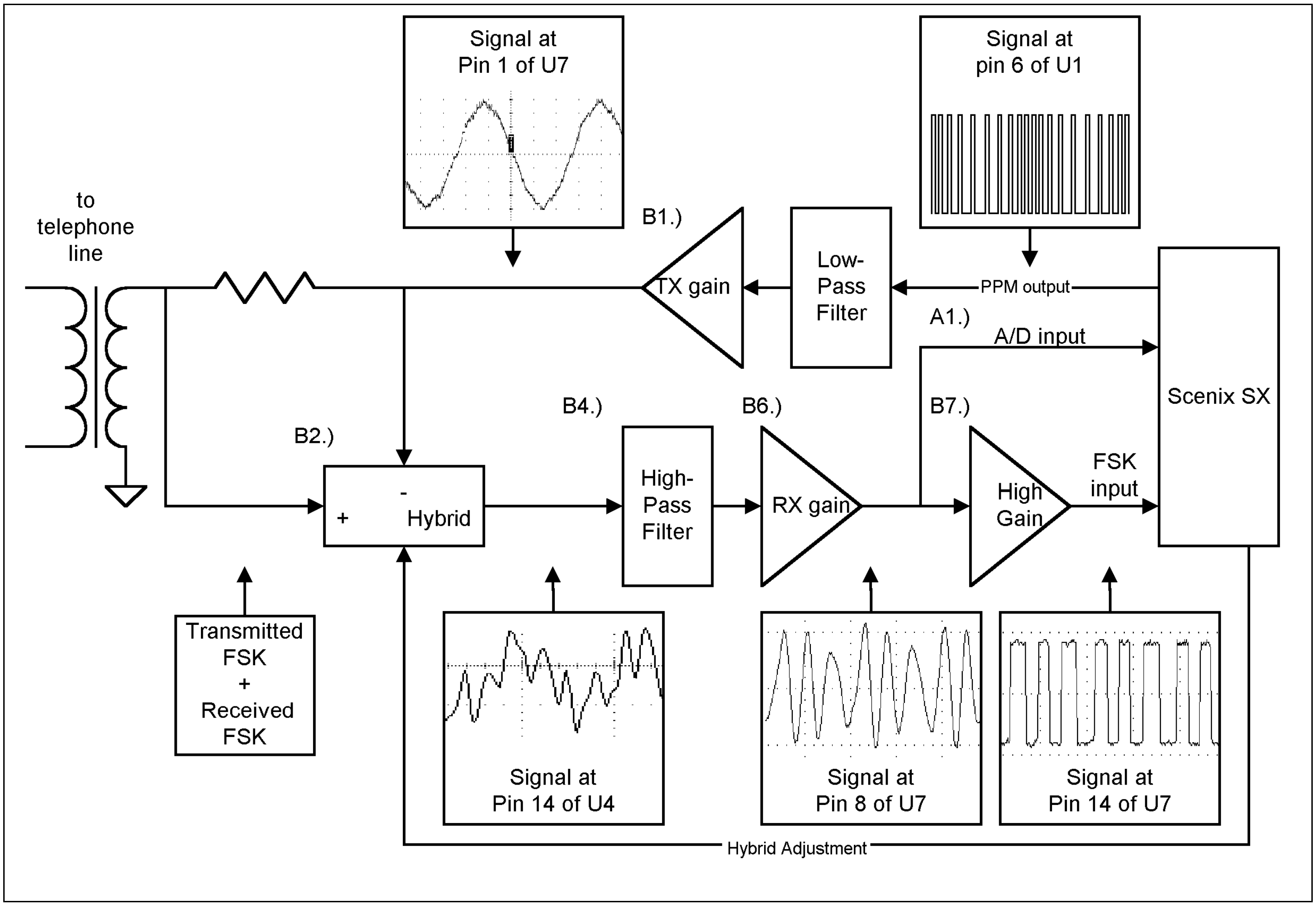
Hybrid

- Four settings provided for automatic hybrid adjustment for various line impedance’s
- Hybrid adjusted by outputting signal onto line and measuring fed-back signal with a low-resolution sigma-delta A/D converter

HARDWARE

<u>Hardware Requirements</u>

According to the V.23 Specification, the originating (dialing) modem transmits a data rate of 75bps using a carrier of 450Hz and 390Hz. The answering modem transmits a data rate of 1200bps using a carrier of 2100Hz and 1300Hz. The block diagram in Figure 1 shows the functional blocks required to implement V.23 origination.



A serial connection to a terminal or PC is required to send commands and data to the modem and to receive data and responses from the modem. The lines the V.23 modem requires to be connected are CTS, TXD, and RXD. The hardware interface to the PC is provided by an RS-232 transceiver such as an ICL232 or MAX232.

The serial connection settings on the terminal or PC should be 1200bps, No Parity, 8 Data Bits, and 1 Stop Bit. Hardware flow control must be enabled, since the V.23 modem's transmit rate is only 75bps yet the PC will transmit data to the modem at 1200bps. Flow control allows the modem to pause the flow of data from the PC when the transmit-buffer is full.

HARDWARE

Modem Schematics

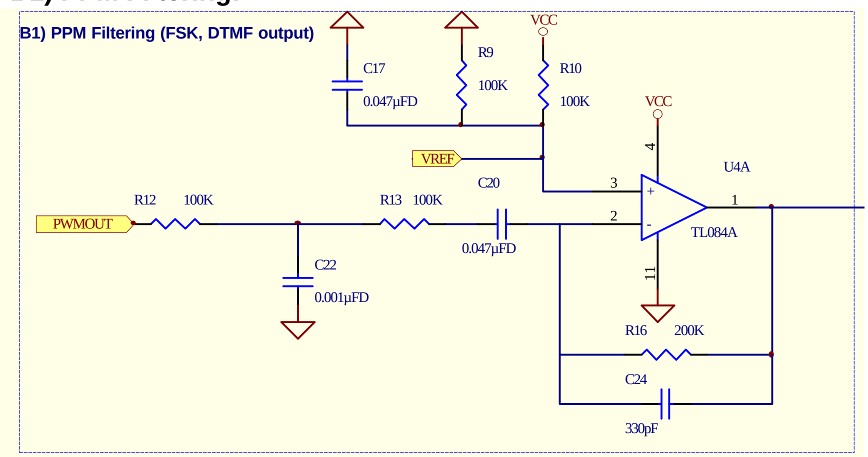
See Appendix A for the complete schematics for the Scenix Modem Demonstration board. Since this application note only covers V.23 origination, several of the components on this schematic are not necessary. The relevant parts of this schematic for V.23 origination include the output filtering, the 4 hybrid resistors, the transformer, the portion of the opto-isolator that allows the modem to go on- or off-hook, the high-pass filtering on the input, and the amplification circuitry on the input. Performing only origination eliminates these components:

- R17, R16, C25, R15, C18, D5, for ring detection
- U6A and its surrounding discrete components for Caller-ID detection (B5)
- U4B and U4C and their discrete components for low-pass filtering FSK in V.23 answer-mode (B3)
- R6, C9, and C10 for DTMF and Call-Progress detection (A2)
- To simplify the circuit even further, the automatic hybrid adjustment may be removed (A2 + B8) and replaced with a fixed resistor value to ground (100kΩ for 600Ω line impedance).

See Appendix D for the schematic used for a minimal implementation of V.23 origination.

Block-by-Block Schematic Descriptions

B1) PPM Filtering:



The filtering on the pulse-position-modulation output of the SX creates the D/A converter for generating FSK and DTMF signals. The cut-off frequency of the filters should be no higher than the highest frequency generated. In this application, a cut-off frequency of 1700Hz could be used, since 1633Hz is the highest frequency that would be generated during normal operation.

This is a dual stage filter. The cutoff frequency for the first stage is

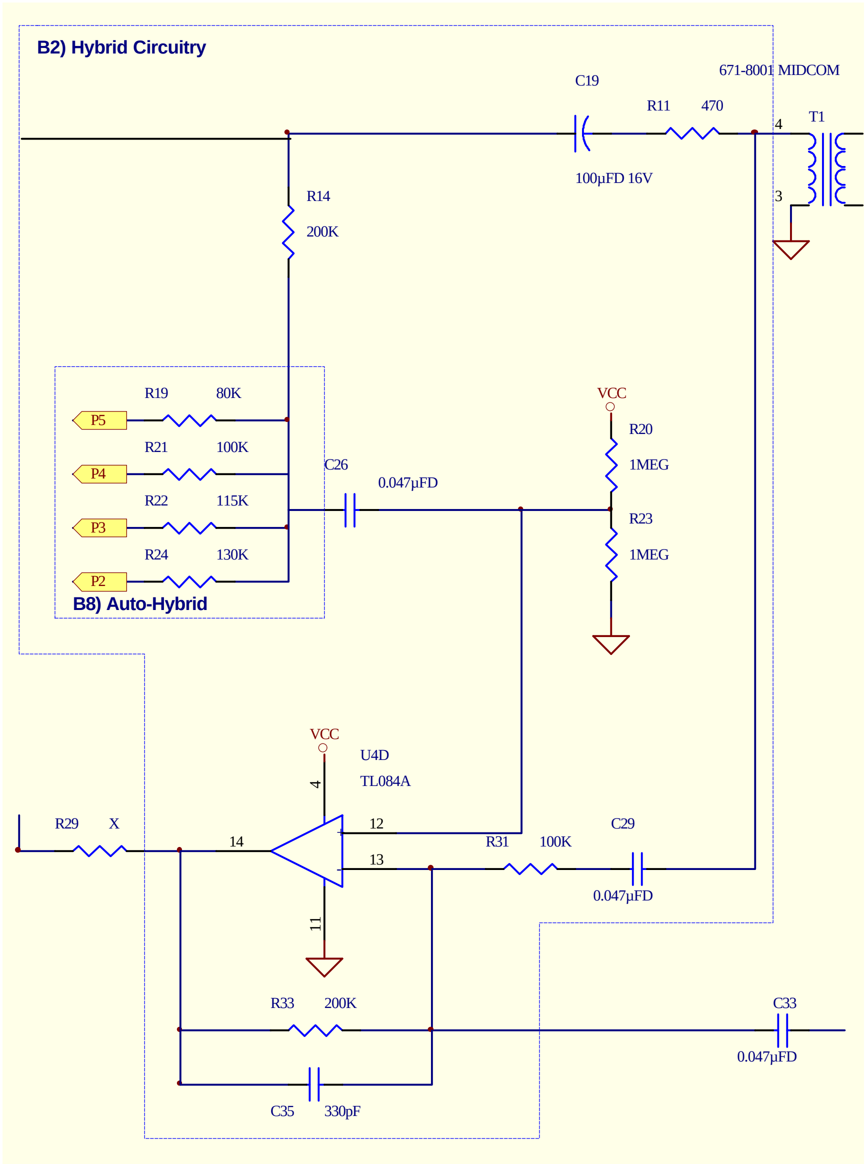
$$Fc = (2 * \pi * ((R12^{-1} + R13^{-1})^{-1}) * C22)^{-1}$$

The cutoff frequency for the second stage is

$$Fc = (2 * \pi * R16 * C24)^{-1}$$

HARDWARE

B2) Hybrid Circuitry



The hybrid circuitry removes some of the transmitted signal from the received signal. R19, R21, R22, and R24 choose the appropriate resistor ratio for the line impedance. To match the hybrid to a specific line impedance, use these resistor values:

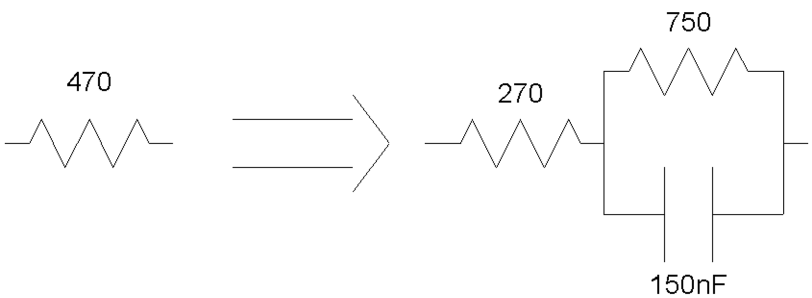
R19 = 450 ohms
R21 = 600 ohms
R22 = 750 ohms
R24 = 900 ohms

For instance, to match the hybrid to a 450-ohm impedance, set P5 as a 0V output and tristate P4, P3, and P2.

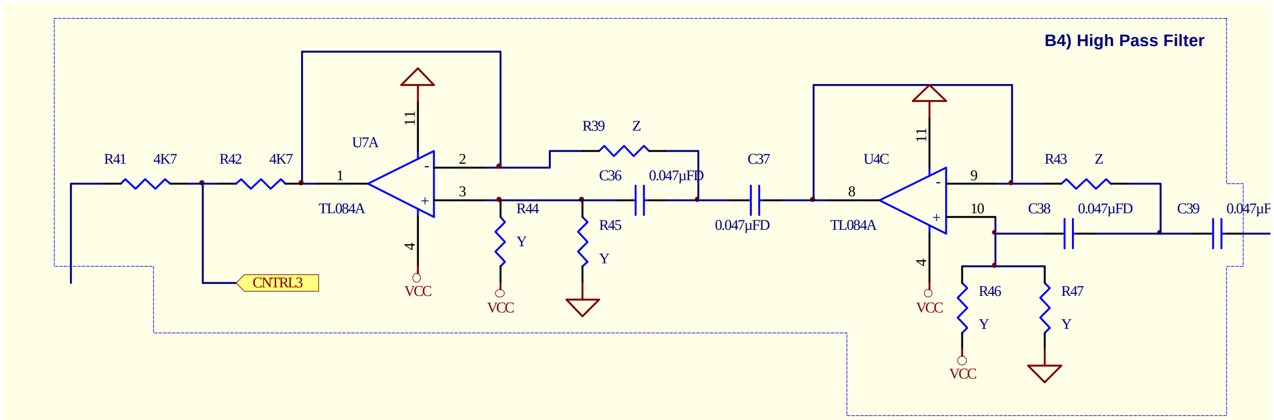
The hybrid adjustment circuitry was included because the Scenix Modem demo-board was designed to demonstrate the modem's operation internationally. Line impedances vary from country to country; however, the hybrid adjustment circuitry and software is not necessary. Without hybrid adjustment, the four resistors, R19, R21, R22, and R24 can be replaced with a single 100k resistor through a capacitor to ground, and C26, R20, and R23 may be eliminated.

<u>CTR-21 Qualifications</u>

To meet CTR-21 specifications, R11 should be replaced by a complex-impedance.

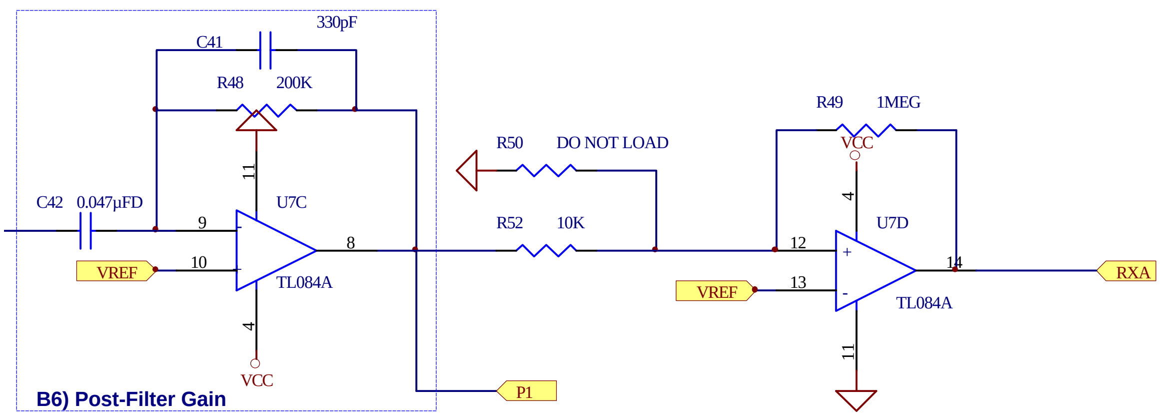


B3 and B4) Input Filtering:



The high-pass filtering removes the remaining low-frequency transmitted signal from the received signal. For V.23 origination, only the high-pass filter is enabled. (Setting CNTRL3 as a tristate pin enables the output of the filter, setting CNTRL3 as an output disables the output of the filter.)
The cut-off frequency for the filter in V.23 originate mode is set to ≅ 1200Hz, as calculated by
$f_c = (2\pi RC)^{-1}$
For this circuitry, these values were chosen:
Y = 5.62kΩ
Z = 2.8kΩ
C36, C37, C38, C39 = 0.047µF
For a calculated cut-off frequency of:
$[2\pi(2.8k\Omega)(0.047\mu F)]^{-1} = 1209Hz$.
Notice that the two 'Y' resistors are double the value of the 'Z' resistors, since they are effectively in parallel and their impedance when combined is ½(Z), producing the same cut-off frequency.

B6 and B7) FSK Amplification:



Since the algorithm for receiving the FSK signal is a zero-cross algorithm, the analog signal is transformed into a digital (+5V and GND) signal by amplifying the received FSK signal into a comparator. The amplification circuitry has a gain of ≅ 20. C41 provides low-pass filtering to eliminate high-frequency noise. R49 adds hysterises to the comparator, reducing the effect of noise on the zero-cross signal. R50 can be used raise the zero-cross point on the input signal, but this is not necessary and is left out of this reference design.

SOFTWARE:

The software of the V.23 origination modem is completed, although a few minor improvements have been suggested:

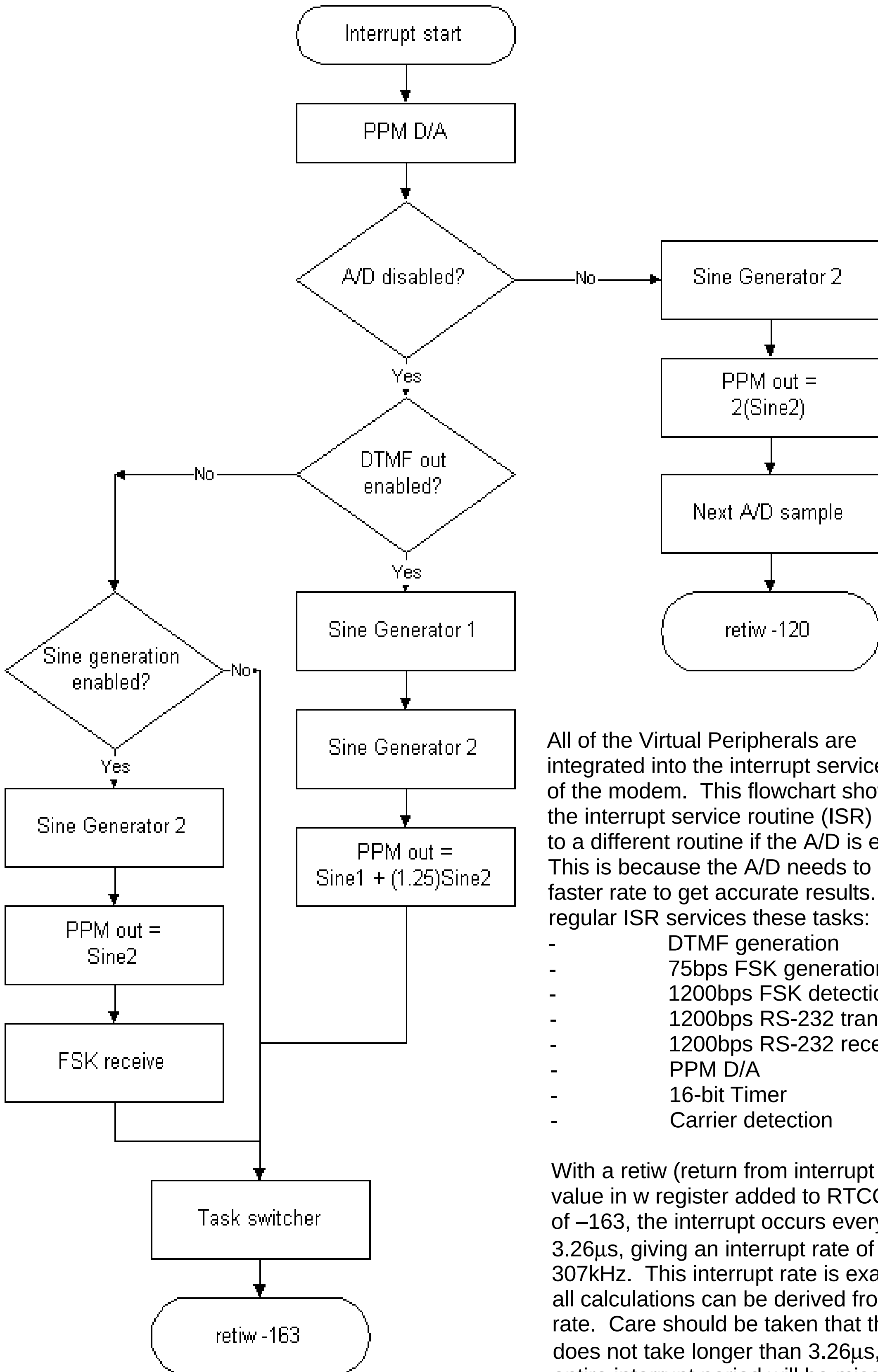
- Once dialing has been initiated, modem should cancel dialing if the user presses a key.

- Once carrier is detected, modem should delay another few seconds, and check again, to ensure no false carrier was detected.

Aside from these improvements, the modem software has been tested to comply with all of the original specifications. (See the specifications at the beginning of this document)

SOFTWARE

Interrupt Service Routine:



All of the Virtual Peripherals are integrated into the interrupt service routine of the modem. This flowchart shows that the interrupt service routine (ISR) splits off to a different routine if the A/D is enabled. This is because the A/D needs to run at a faster rate to get accurate results. The regular ISR services these tasks:

- DTMF generation
- 75bps FSK generation
- 1200bps FSK detection
- 1200bps RS-232 transmitter
- 1200bps RS-232 receiver
- PPM D/A
- 16-bit Timer
- Carrier detection

With a retiw (return from interrupt with value in w register added to RTCC) value of –163, the interrupt occurs every 3.26µs, giving an interrupt rate of 307kHz. This interrupt rate is exact, and all calculations can be derived from this rate. Care should be taken that the ISR does not take longer than 3.26µs, or an entire interrupt period will be missed.

SOFTWARE

Interrupt Service Routine:

Pulse Position Modulation D/A:

This is the assembly code for the PPM D/A, which runs on every pass of the ISR.

```
;*****************************************************************************
PPM_output
              bank    PPM_bank              ; Update the PPM pin
              clc
              add     PPM0_acc,PPM0_out
              snc
              setb    PPM_pin
              sc
              clrb    PPM_pin
;*************************************************************************
```

A simple Pulse Position Modulator is used to perform the Digital to Analog conversion. The resolution of the PPM modulator is 3.26 microseconds, resulting in a maximum output frequency of 154kHz. A low-pass filter with a cutoff of 1.6kHz or greater is used to filter the PPM signal.

On every interrupt, the PPM ISR adds the pwm0_out register to the accumulator register, and the carry flag is moved directly to the PPM pin. A large pwm0_out will cause more frequent carries, producing a larger analog voltage, whereas a small value will produce less frequent carries, producing a smaller analog voltage. An external low-pass filter removes the high-frequency components of the PPM output, producing a steady analog output.

FSK generation and DTMF generation

```
;*************************************************************************
sine_generator1                                    ;(Part of interrupt service routine)
; This routine generates a sine wave with values from the sine table
; at the end of this program.  Frequency is specified by the counter.  To set
; the frequency, put this value into the 16-bit freq_count register:
; freq_count = FREQUENCY * 6.83671552 (@50MHz)
;*************************************************************************
       bank    sine_gen_bank

              clc
              add     freq_acc_low,freq_count_low
              add     freq_acc_high,freq_count_high
              sc
              jmp     :no_change
              inc     sine_index
              mov     w,sine_index
              and     w,#$1f
              call    sine_table
              mov     curr_sine,w           ;1

:no_change
```

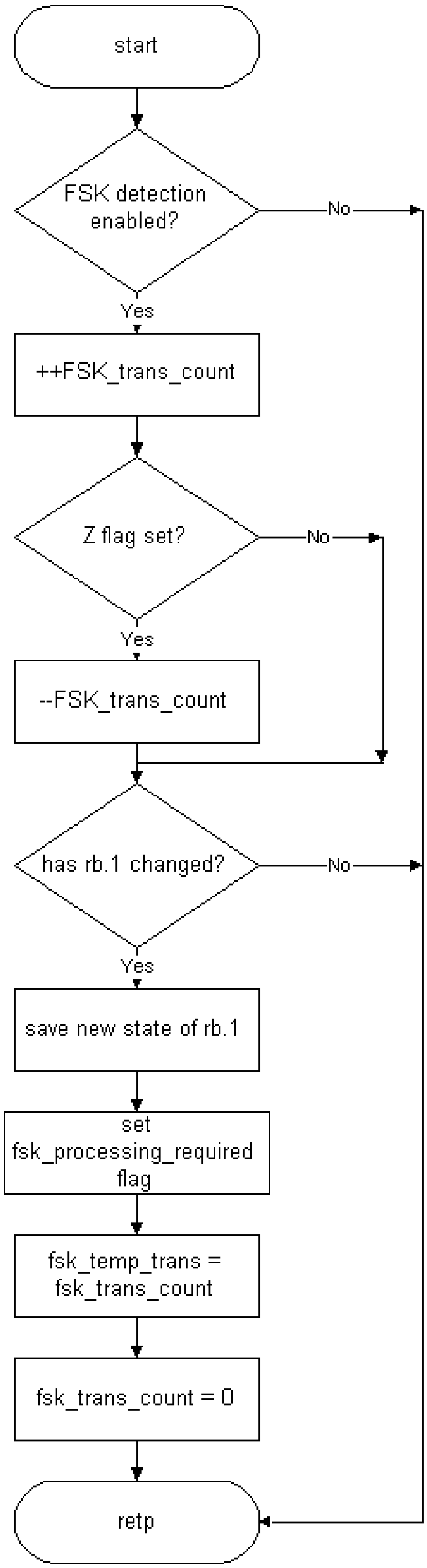
The sine generators use 16-bit phase accumulators, in addition to a table index, to produce a sine wave from a table in the EEPROM of the SX. On each pass of the ISR, the freq_count registers are added to the freq_acc registers, and the table index is incremented if a carry occurs. For very low freq_count values, carries will be less frequent and the index will move through the table at a lower rate. For higher freq_count values, carries will be more frequent and the index will move through the table at a higher rate.

Only one sine generator is used to generate FSK. To generate DTMF, two sine generators are used and their outputs are summed in the ppm0_out register.

SOFTWARE

Interrupt Service Routine:

Receiving FSK:



Data reception and demodulation

- FSK reception data rate at 1200bps
- Logic '1' (mark) demodulated from 1300Hz carrier
- Logic '0' (space) demodulated from 2100Hz carrier
- Carrier detection
- Timed-Zero-Cross algorithm
- Carrier Detection

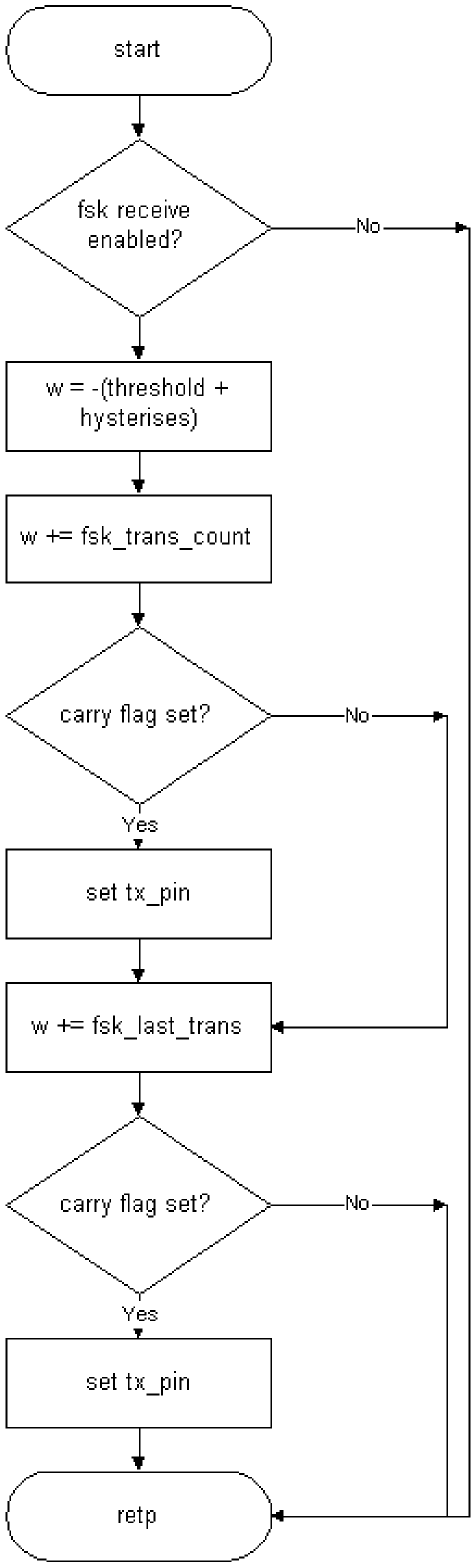
The FSK receive portion of the modem software is performed completely by the Interrupt Service Routine, with no logic required in the mainline routine. FSK receive is enabled via the fsk_rx_en flag in the global flags register. Once enabled, the FSK receive algorithm sets or clears the RS-232 transmit pin, depending on the incoming FSK signal. A timed zero-cross algorithm is used to demodulate the incoming FSK signal. The FSK signal is converted to a square wave by the analog circuitry, and the time elapsed for two transitions is compared to a threshold. The threshold is raised if a low frequency was just detected, and lowered if a high frequency was just detected. This reduces the effects of jitter caused by noise.

The flowchart to the left is for the main part of the FSK-receive algorithm. It runs on every pass of the ISR as long as FSK reception is enabled. This code counts the time between transitions on rb.1. When a transition is detected, the transition count is saved in a temporary register, and a flag is set that indicates to the processing routines that there is data to process. The transition count is re-started from zero.

SOFTWARE

Interrupt Service Routine:

Receiving FSK:

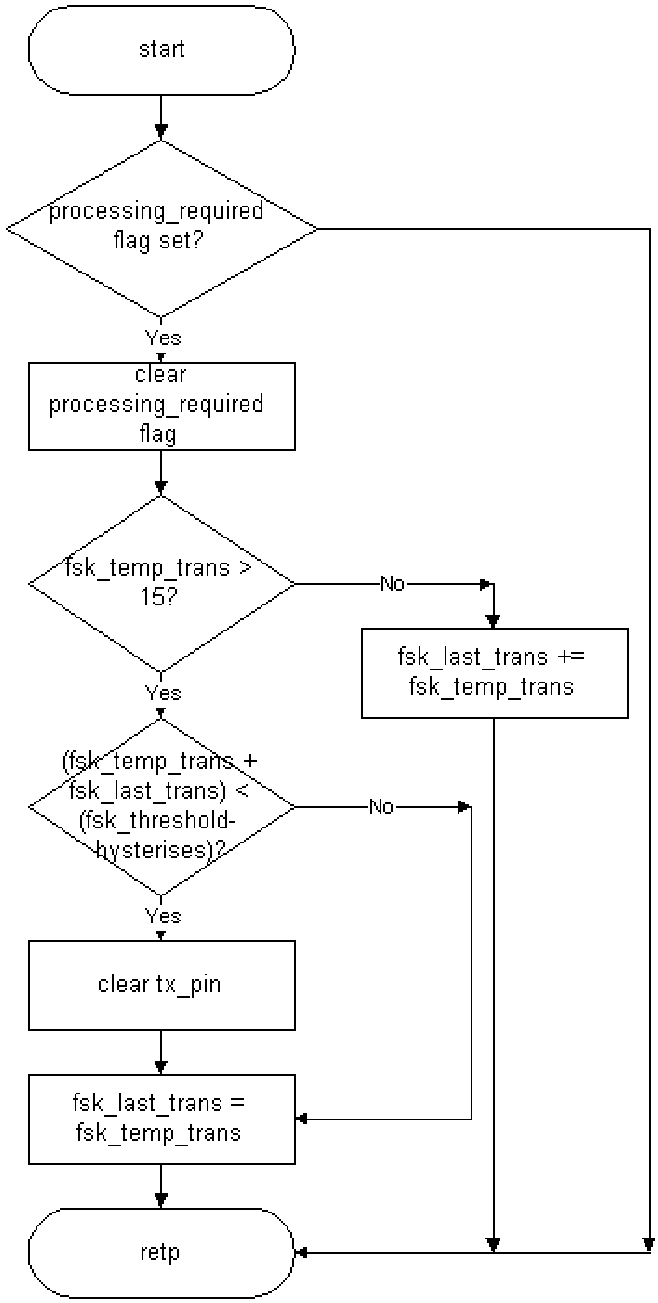


Another part of the FSK receive algorithm is the FSK_receive_main_2 subroutine, which also runs in the ISR. This routine watches for transition counts that are well over the high frequency/low frequency threshold, and automatically sets the RS-232 transmit pin high the instant that this threshold is exceeded. This routine runs in the task manager.

SOFTWARE

Interrupt Service Routine:

Receiving FSK:



When a transition occurs on the FSK receive pin (rb.1), the fsk_processing_required flag gets set. The fsk_receive_processing routine (left) then checks the latest transition count for a high frequency. If the current transition count, when added to the previous transition count, does not exceed the threshold, then the current input frequency is high and the rs-232 transmit pin is set low.

SOFTWARE

Interrupt Service Routine:

Multitasking the Interrupt Service Routine:

To save on processing time, a task manager runs any tasks that don't need to run at the full interrupt speed. The task manager runs one task per interrupt from a table. These tasks include the RS-232 transmitters and receivers, the 75bps FSK synchronizer, the 16-bit timers, and some of the FSK detection processing. The task_switcher variable is a global variable used to keep track of the next task to run.

```
;***************************************************************************
task_manager
; This portion of the ISR allows 1 of 16 separate tasks to run in each
; interrupt.
;***************************************************************************
               inc     task_switcher
               mov     w,task_switcher
               and     w,#$0f
               clc
               jmp     pc+w

               ;*** TASKS ***
               jmp     fsk_receive_main_2     ;0
               jmp     transmit               ;1
               jmp     receive                ;2
               jmp     fsk_transmit_uart      ;3
               jmp     fsk_receive_main_2     ;4
               jmp     transmit_fsk           ;5
               jmp     do_timers              ;6
               jmp     fsk_receive_processing1 ;7
               jmp     fsk_receive_main_2     ;8
               jmp     carrier_detect         ;9
               retp                           ;10
               retp                           ;11
               jmp     fsk_receive_main_2     ;12
               retp                           ;13
               retp                           ;14
               retp                           ;15
               jmp     fsk_receive_main_2     ;16
               retp                           ; (just in case)

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

SOFTWARE

Interrupt Service Routine:

Universal Asynchronous Receiver/Transmitter:

- 1200 baud
- No Parity
- 8 Data Bits
- 1 Stop Bit
- Hardware Flow Control (CTS, RTS)

The UART is integrated into the Interrupt Service Routine of the software, which runs every 3.26us. The UART runs on every 16th pass of the ISR, or every 52.16 microseconds. The bit time for a 1200bps UART is 83.33 milliseconds. Dividing 83.33 milliseconds by 52.16us gives a result of 15.97, or 16, allowing for an easy divide ratio for the UART timing.

```
;*************************************************************************
transmit
; This is an asynchronous RS-232 transmitter
; INPUTS:
;       tx_divide.baud_bit  -  Transmitter only executes when this bit is = 1
;       tx_high             -  Part of the data to be transmitted
;       tx_low              -  Some more of the data to be transmitted
;       tx_count            -  Counter which counts the number of bits transmitted.
; OUTPUTS:
;       tx_pin              -  Sets/Clears this pin to accomplish the transmission.
;*************************************************************************
               bank    serial
               clrb    tx_divide.baud_bit      ;clear xmit timing count flag
               inc     tx_divide               ;only execute the transmit routine
               STZ                             ;set zero flag for test
               SNB     tx_divide.baud_bit      ; every 2^baud_bit interrupt
               test    tx_count                ;are we sending?
               snz
               retp                           ;if not, go to :receive
               clc                             ;yes, ready stop bit
               rr      tx_high                 ; and shift to next bit
               rr      tx_low                  ;
               dec     tx_count                ;decrement bit counter
               movb    tx_pin,/tx_low.6        ;output next bit
               retp
;*************************************************************************
```

```
;*************************************************************************
receive
; This is an asynchronous receiver for RS-232 reception
; INPUTS:
;       rx_pin             -    Pin which RS-232 is received on.
; OUTPUTS:
;       rx_byte            -    The byte received
;       rx_flag            -    Set when a byte is received.
;*************************************************************************
               bank    serial
               movb    c,rx_pin                ;get current rx bit
               test    rx_count                ;currently receiving byte?
               jnz     :rxbit                  ;if so, jump ahead
               mov     w,#9                    ;in case start, ready 9 bits
               sc                              ;skip ahead if not start bit
               mov     rx_count,w              ;it is, so renew bit count
               mov     rx_divide,#start_delay  ;ready 1.5 bit periods
:rxbit         djnz    rx_divide,:rxdone       ;middle of next bit?
               setb    rx_divide.baud_bit      ;yes, ready 1 bit period
               dec     rx_count                ;last bit?
               sz                              ;if not
               rr      rx_byte                 ;  then save bit
               snz                             ;if so
               setb    rx_flag                 ;  then set flag
:rxdone
               retp
;*************************************************************************
```

SOFTWARE

Interrupt Service Routine:
Transmitting FSK:
The timer for the FSK transmitter uses the same timing scheme used by the RS-232 transmitter, but it divides the timers by 16 again to accomplish 75bps transmission.

```
;**************************************************************************
fsk_transmit_uart
; This is an asynchronous RS-232 transmitter
; INPUTS:
;       tx_divide.baud_bit  -  Transmitter only executes when this bit is = 1
;       tx_high             -  Part of the data to be transmitted
;       tx_low              -  Some more of the data to be transmitted
;       tx_count            -  Counter which counts the number of bits transmitted.
; OUTPUTS:
;       tx_pin              -  Sets/Clears this pin to accomplish the transmission.
;**************************************************************************
               bank    fsk_serial_bank
               sb      fsk_answering
               inc     fsk_tx_divide_2
               and     fsk_tx_divide_2,#$0f      ; Divide the 1200bps UART by 16 to
                                                 ; achieve 75bps
               sz
               retp
               clrb     fsk_tx_divide.baud_bit      ;clear xmit timing count flag
               inc      fsk_tx_divide               ;only execute the transmit routine
               STZ                              ;set zero flag for test
               SNB      fsk_tx_divide.baud_bit      ; every 2^baud_bit interrupt
               test     fsk_tx_count                ;are we sending?
               snz
               retp                            ;if not, go to :receive
               clc                              ;yes, ready stop bit
               rr       fsk_tx_high                 ; and shift to next bit
               rr       fsk_tx_low                  ;
               dec      fsk_tx_count                ;decrement bit counter
               movb     fsk_tx_bit,/fsk_tx_low.6        ;output next bit
               retp
;**************************************************************************
```

This routine is tied into the transmit_fsk routine, which loads the sine generator’s registers with a high frequency when the fsk_tx_bit is low, and vice-versa for a high fsk_tx_bit.

```
;**************************************************************************
transmit_fsk
;**************************************************************************
               bank    fsk_transmit_bank
               sb      fsk_tx_en
               retp
               jb      fsk_answering,transmit_answer_tones
transmit_originate_tones
               jnb     fsk_tx_bit,:low_bit
:high_bit
               bank    sine_gen_bank
               mov     freq_count_high2,#f390_h
               mov     freq_count_low2,#f390_l
               retp
:low_bit
               bank    sine_gen_bank
               mov     freq_count_high2,#f450_h
               mov     freq_count_low2,#f450_l
               retp
transmit_answer_tones
               jnb     fsk_tx_bit,:low_bit
:high_bit
               bank    sine_gen_bank
               mov     freq_count_high2,#f1300_h
               mov     freq_count_low2,#f1300_l
               retp
:low_bit
               bank    sine_gen_bank
               mov     freq_count_high2,#f2100_h
               mov     freq_count_low2,#f2100_l
               retp
```

SOFTWARE

Main Program:

Compact AT command set:

- 64-byte command buffer
- Dial: “ATDTxxxxxxxxx…”
- Switch from data mode to command mode: “+++”
- Switch from command mode to data mode: “ATO”
- Hang up: “ATH”
- Initialize: “ATZ”
- Hybrid Optimization “ATY”

The AT-commands were chosen to provide enough functionality for a very simple modem design. Since the SX originate-only modem can only originate a data call, no answer functions are implemented. Incoming AT-commands are stored in a 64-byte buffer, and compared to software lookup tables on reception of a carriage return.

```
;*************************************************************************
command_1                                    ; Dial command
               mov     w,pop_index
               add     PC,w
               retw    'A'
               retw    'T'
               retw    'D'
               retw    'T'
               jmp     DIAL_MODE
;*************************************************************************
```

The AT-Commands are stored in a series of jump tables, like the above. The last table entry is a jump to the routine that handles the command.

Hybrid:

- Four settings provided for automatic hybrid adjustment for various line impedance’s
- Hybrid adjusted by outputting signal onto line and measuring fed-back signal with a low-resolution sigma-delta A/D converter

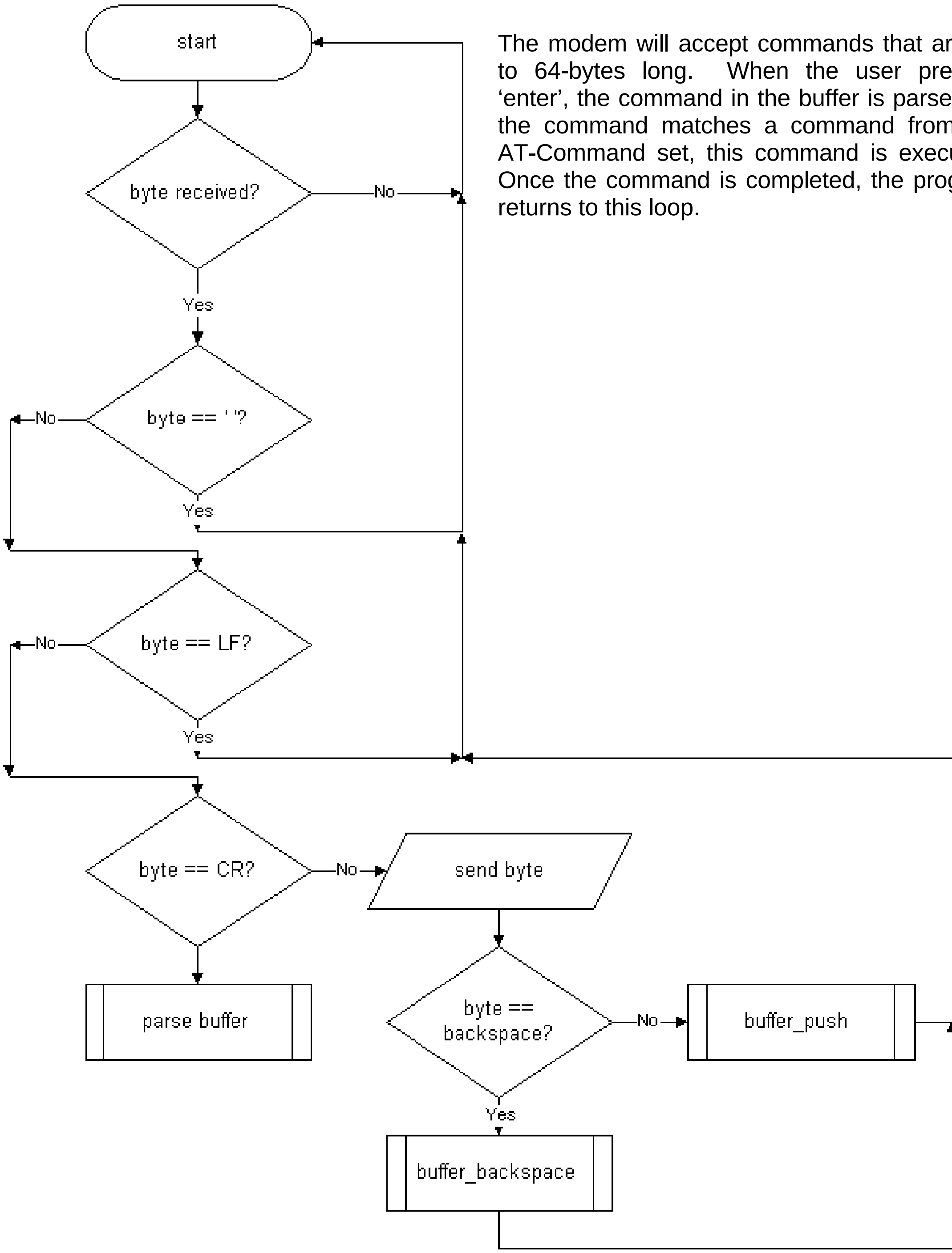
Because of the high attenuation at the filtering stages, it is not necessary for the hybrid to be perfectly matched to the line impedance. Four impedance-match settings are provided by the V.23 reference design. On initialization, the modem outputs a DTMF digit to quiet the line. It then outputs a 2100Hz tone to disable the line equalizers, and measures the amplitude of the signal being fed-back. Each setting is tried, and the setting that produces the most attenuated feedback is saved and used. This allows the SX reference design to be optimized in software for each individual telephone line. The command to optimize the hybrid is “ATY.” The optimization process takes about 10 seconds. Optimization needs to be performed each time the modem is powered down, since there is no NVRAM on the board to remember the result of the last optimization.

SOFTWARE

Main Program:

Modem Command Line:

The modem will accept commands that are up to 64-bytes long. When the user presses 'enter', the command in the buffer is parsed. If the command matches a command from the AT-Command set, this command is executed. Once the command is completed, the program returns to this loop.



SOFTWARE

V.23 Operation : Main Loop

FSK De-Modulation

V.23 Data Reception is performed completely in the interrupt service routine. The main-line routine simply needs to set the fsk_rx_en flag to enable FSK reception.

Data Transmission/FSK Modulation

Also in the main loop, the program continually waits for the fsk_transmitter to be idle, indicated by the fsk_tx_count register equaling zero. When the fsk transmitter is idle, calling fsk_send_byte will send the next byte from the transmit-buffer.

Another task performed in the main loop of the modem is to load any received RS-232 characters into the transmit buffer. When a byte has been received (indicated by the rx_flag) the program calls the buffer_push routine, loading the received byte into the transmit buffer. The program checks to position of the push and pop indexes into the buffer. If the buffer is within 5 bytes of being full, the program will set the CTS pin of the SX, disabling data transmission from the PC. The CTS pin will not be re-enabled until the entire transmit-buffer is empty. The pop_index register equaling the push_index register indicates an empty buffer.

Carrier Detection

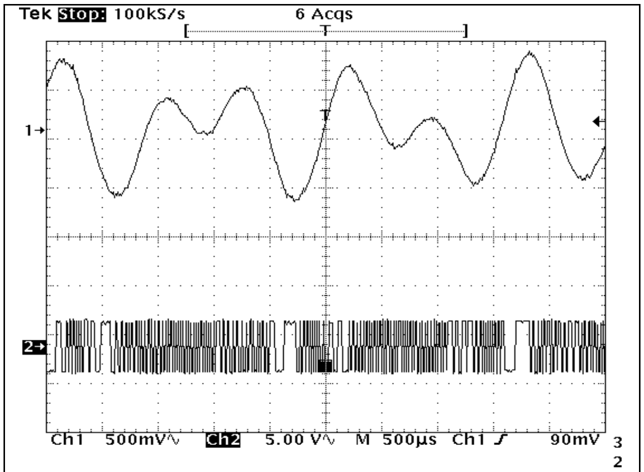
The main loop constantly tests the carrier_detected flag, which is set and cleared by the interrupt service routine. If the interrupt service routine detects no input, or a frequency that does not fall within V.23 ranges, it will clear the carrier_detected flag. When the main loop detects that this flag is cleared, it will jump to a routine that waits another 8 seconds and re-checks for carrier. If the carrier is still not detected, the program jumps to the hang-up routine, which then returns to the command line routine.

Exiting Data Mode (Detecting “+++”)

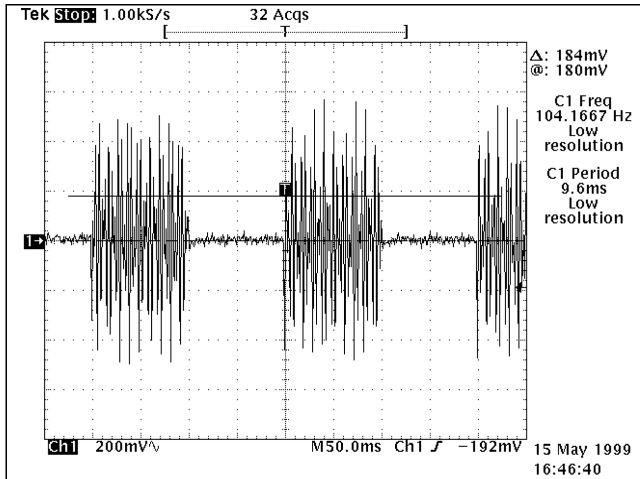
The escape code sequence forces the modem to the command-state from the on-line state. It consists of a three-character escape code sequence surrounded by escape guard times. The delay between issuance of each escape character must not exceed the escape guard time. The escape guard time is defined as the time delay required between the last character transmitted and the first character of the escape code. The guard time is 2 seconds, and the escape code sequence is “+++”.

To detect the escape code sequence, the program resets a 2-second timer every time a character is received. If the input character is a “+” and the 2-second timer has expired, the program increments the plus_count register and resets the timer for 2 seconds. When the next “+” is received, the program ensures that the timer has NOT timed out before incrementing the plus_count register. If the timer expires or the character is not a “+”, the timers and plus_count register are reset again. In another part of the main loop, the program will exit data mode under the condition that the timer_flag is set and the plus_count register is equal to 3, a condition that will only occur when the escape code has been received, surrounded by the guard time.

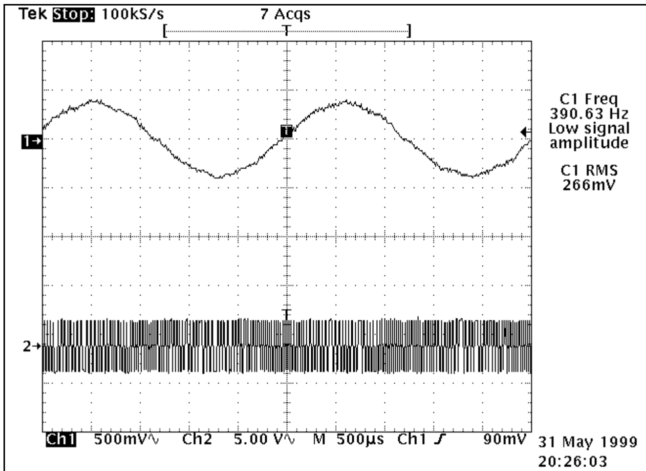
APPENDIX A: Signals in the Modem



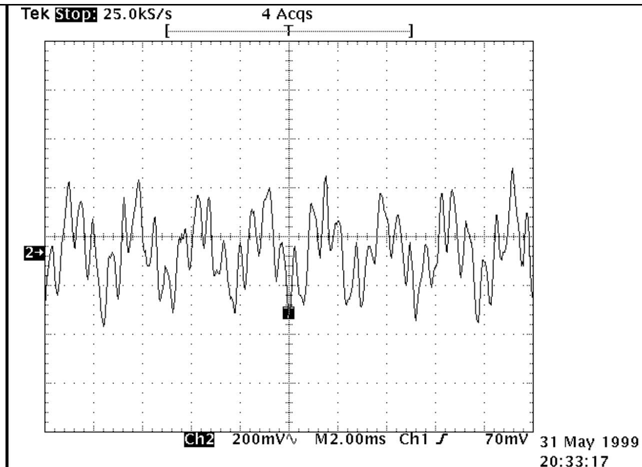
Top: U4 pin 1 Filtered DTMF output ('1')
Bottom: U1 pin 6 PPM output



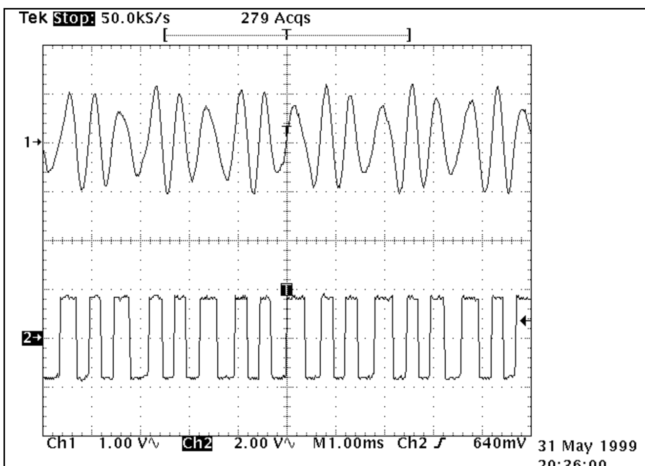
Dial Times: 100ms ON
100ms OFF



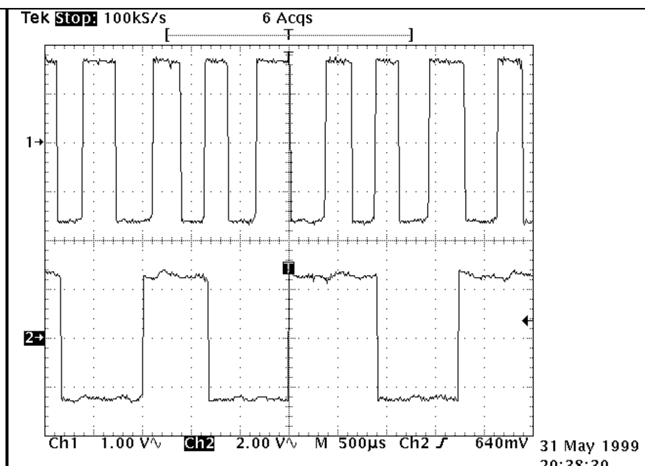
Top: U4 pin 1 Filtered 390Hz output
Bottom: U1 pin 6 PPM output



Hybrid Output: U4 pin 14
1300Hz received on 390Hz transmitted.



Top: U7 pin 8 Filtered FSK input
Bottom: U7 pin 14 Zero-cross FSK input



Top: U7 pin 14 Zero-cross FSK input
Bottom: U1 pin 8 De-modulated data stream

APPENDIX B: CD-R Contents

Quickstart.PDF : Quick guide to running the modem demonstration

read_me.txt : The CD-R's index

max232.pdf : MAX232 RS-232 Driver datasheet

st_tl084cn.pdf : TL084 Op-Amp datasheet

ts117.pdf : TS117 Multifunction Telecom Switch datasheet

sx_datasheet.pdf : SX18AC/SX28AC datasheet

sx28_addendum.pdf : Addendum to sx_datasheet.pdf

SX_User's_Manual.pdf :User's manual for Scenix SX devices

ar40eng.exe : Adobe Acrobat Reader V.4.0

SXKey28L.exe : Parallax Assembler for SX28L devices

V_23_Schematic_2_2.pdf : .PDF's of the modem schematics

V.23 Source Code\ : Folder containing all V.23 source code up to June 1, 1999.

I.D.C\ : Folder containing all files provided to Scenix from I.D.C., including ORCAD schematics, PCB layouts, netlists, Bills of Material, etc. Some component values may differ, but the netlist and layout is the same as the final design.

Protel Stuff\Protel Trial Version\Setup.exe : Trial version of Protel 99 (Also downloadable from Protel's website). Opens all Protel files included in the Protel Stuff directory.

Protel Stuff\Scenix2.DDB : The Scenix version of the V.23 modem schematic. Includes all changes made after I.D.C. handed the design over. This file can be opened with the trial version of Protel 99.

Protel Stuff\Scenix2_Cache : Protel '98 format parts cache

Protel Stuff\Scenix2_Library : Protel '98 format parts library

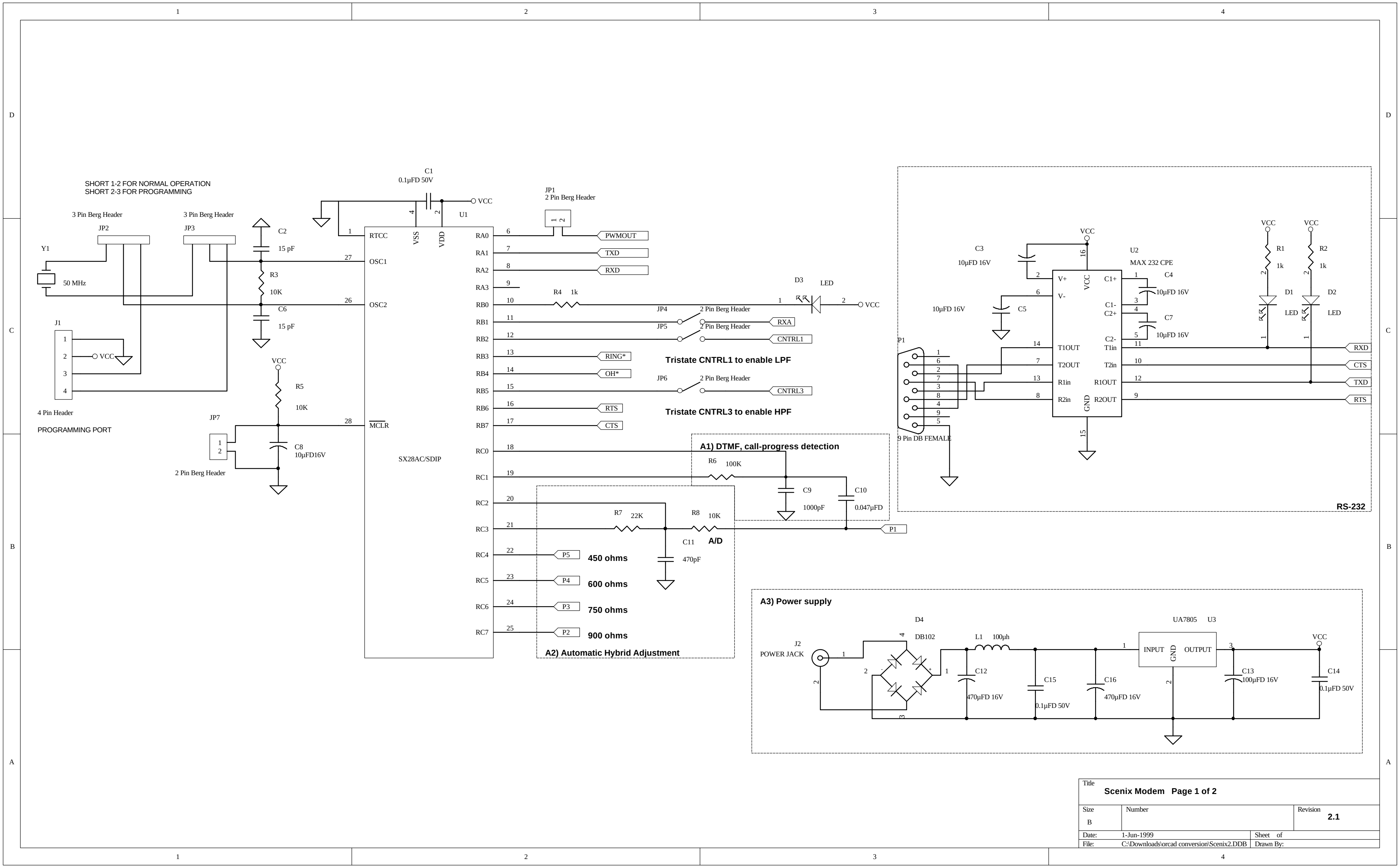
Protel Stuff\SCHEMATIC1 : Protel '98 format master schematic (links page 1 and page 2 of schematic)

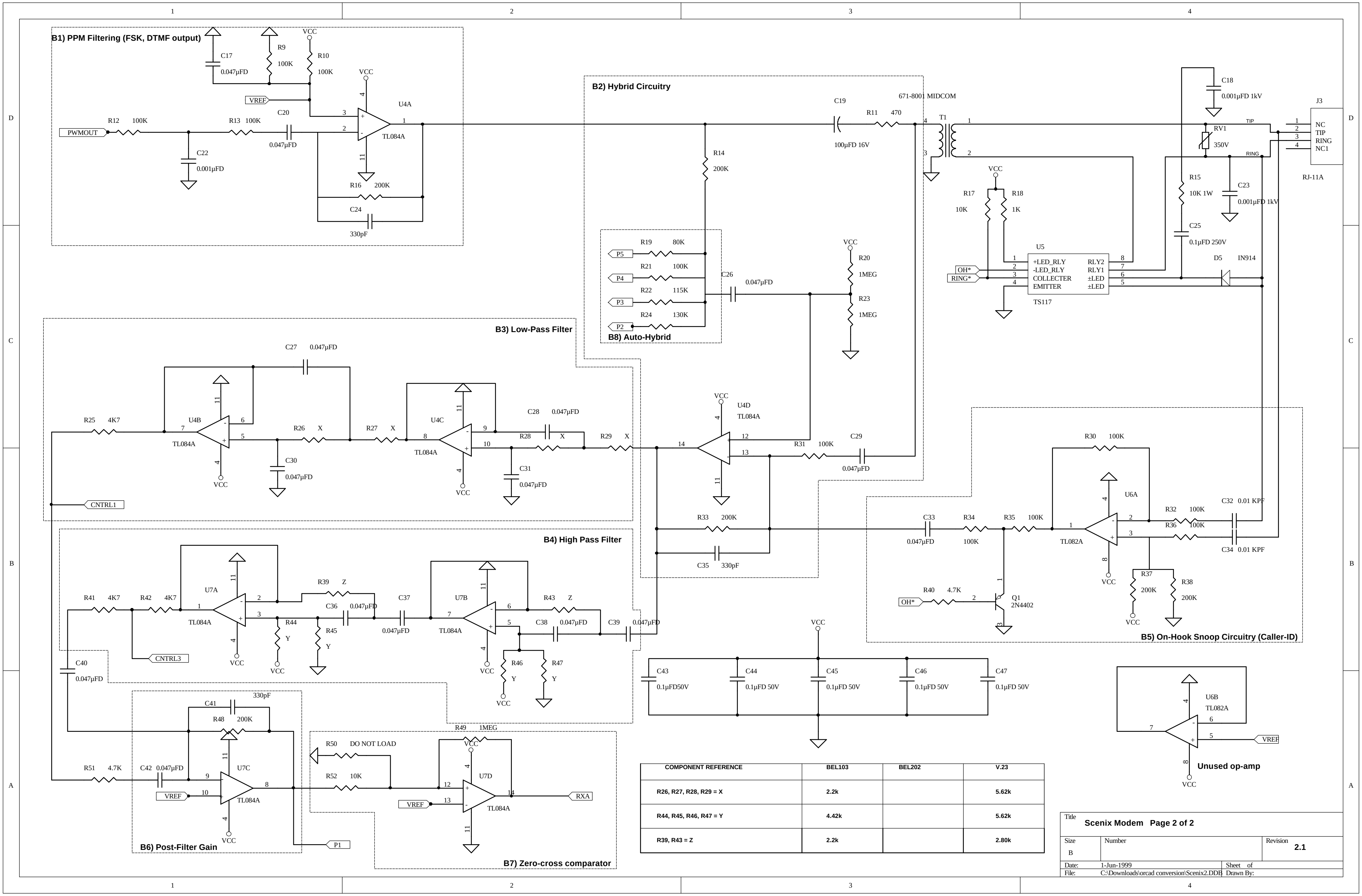
Protel Stuff\Page1_2.sch : Protel '98 format page 1 of schematic

Protel Stuff\Page2_2.sch : Protel '98 format page 2 of schematic

Protel Stuff\SCHEMATIC1_BOM.CSV : Bill of Materials with most up-to-date component values (June 1, 1999)

APPENDIX C: V.23 MODEM DEMO-BOARD SCHEMATICS

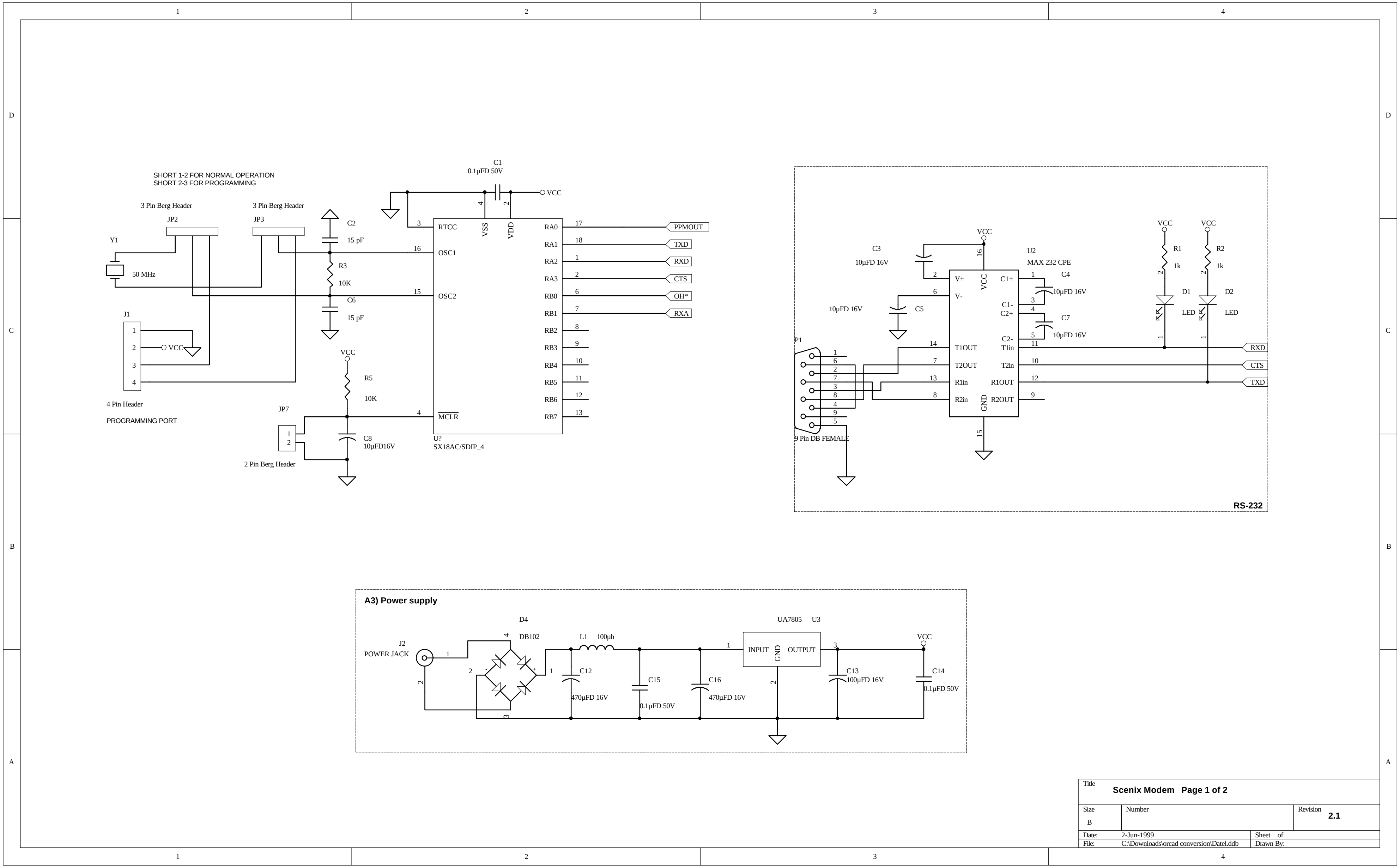

SHORT 1-2 FOR NORMAL OPERATION
SHORT 2-3 FOR PROGRAMMING
3 Pin Berg Header
JP2
3 Pin Berg Header
JP3
Y1
50 MHz
C2
15 pF
R3
10K
C6
15 pF
J1
4 Pin Header
PROGRAMMING PORT
VCC
R5
10K
JP7
C8
10µFD16V
2 Pin Berg Header
C1
0.1µFD 50V
VCC
U1
JP1
2 Pin Berg Header
RTCC
OSC1
OSC2
MCLR
VSS
VDD
SX28AC/SDIP
RA0
RA1
RA2
RA3
RB0
RB1
RB2
RB3
RB4
RB5
RB6
RB7
RC0
RC1
RC2
RC3
RC4
RC5
RC6
RC7
PWMOUT
TXD
RXD
R4 1k
D3 LED
JP4 2 Pin Berg Header
JP5 2 Pin Berg Header
RXA
CNTRL1
RING*
OH*
Tristate CNTRL1 to enable LPF
JP6 2 Pin Berg Header
CNTRL3
RTS
CTS
Tristate CNTRL3 to enable HPF
A1) DTMF, call-progress detection
R6 100K
C9 1000pF
C10 0.047µFD
R7 22K
R8 10K
C11 470pF
A/D
P1
P5 450 ohms
P4 600 ohms
P3 750 ohms
P2 900 ohms
A2) Automatic Hybrid Adjustment
C3 10µFD 16V
C5 10µFD 16V
U2
MAX 232 CPE
V+
V-
C1+
C1-
C2+
C2-
T1OUT
T2OUT
R1in
R2in
T1in
T2in
R1OUT
R2OUT
GND
C4 10µFD 16V
C7 10µFD 16V
R1 1k
R2 1k
D1 LED
D2 LED
P1
9 Pin DB FEMALE
RXD
CTS
TXD
RTS
RS-232
A3) Power supply
J2
POWER JACK
D4
DB102
L1 100µh
C12 470µFD 16V
C15 0.1µFD 50V
C16 470µFD 16V
UA7805 U3
INPUT
OUTPUT
GND
C13 100µFD 16V
C14 0.1µFD 50V
Title
Scenix Modem Page 1 of 2
Size B
Number
Revision 2.1
Date: 1-Jun-1999
Sheet of
File: C:\Downloads\orcad conversion\Scenix2.DDB
Drawn By:

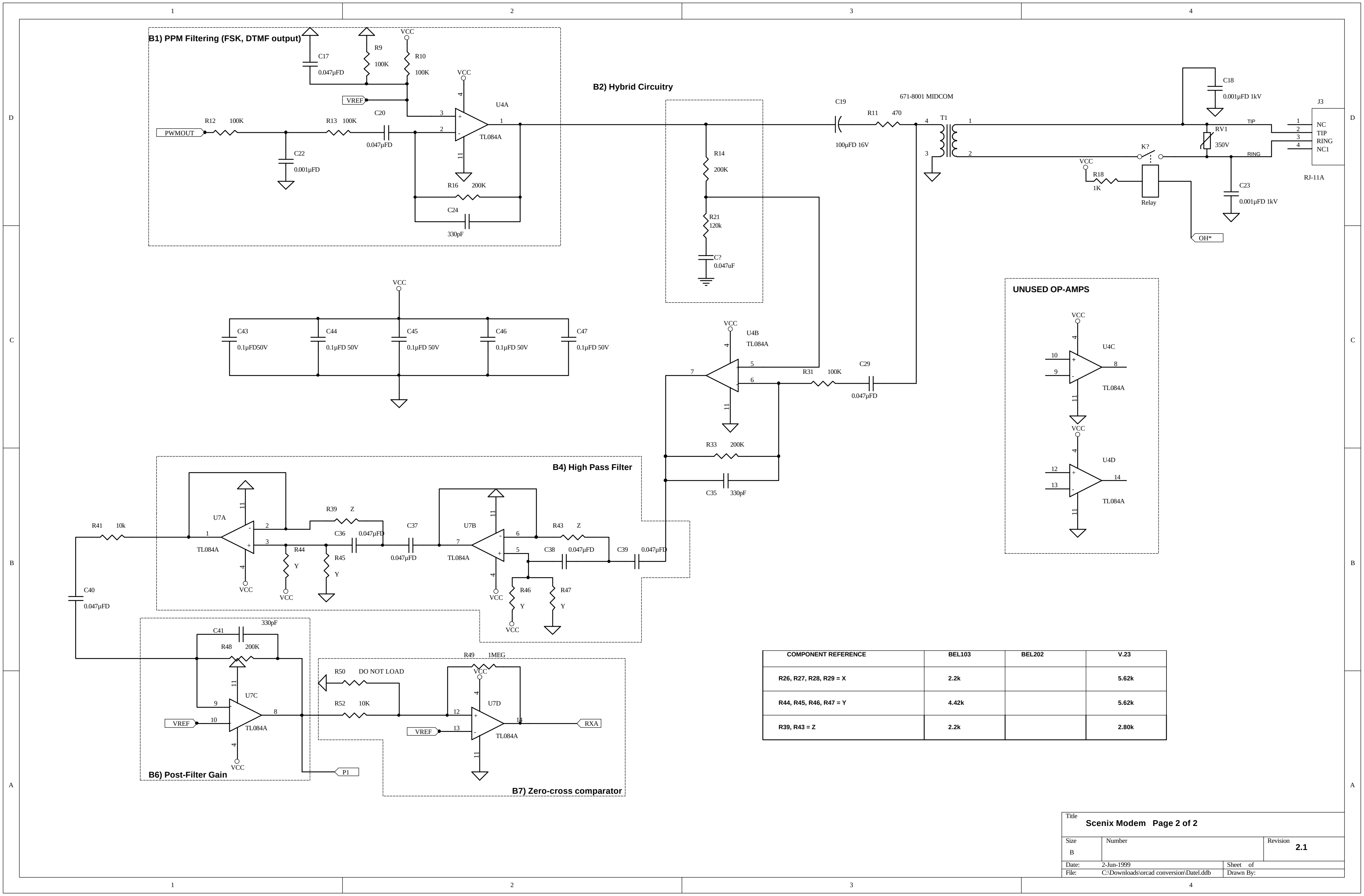


COMPONENT REFERENCE	BEL103	BEL202	V.23
R26, R27, R28, R29 = X	2.2k		5.62k
R44, R45, R46, R47 = Y	4.42k		5.62k
R39, R43 = Z	2.2k		2.80k

Title	**Scenix Modem Page 2 of 2**	
Size B	Number	Revision 2.1
Date: 1-Jun-1999	Sheet of	
File: C:\Downloads\orcad conversion\Scenix2.DDB	Drawn By:	

APPENDIX D: MINIMAL V.23 ORIGINATE MODE SCHEMATICS


SHORT 1-2 FOR NORMAL OPERATION
SHORT 2-3 FOR PROGRAMMING
3 Pin Berg Header
JP2
3 Pin Berg Header
JP3
Y1
50 MHz
J1
VCC
4 Pin Header
PROGRAMMING PORT
C1
0.1µFD 50V
VCC
C2
15 pF
R3
10K
C6
15 pF
RTCC
OSC1
OSC2
VSS
VDD
MCLR
RA0
RA1
RA2
RA3
RB0
RB1
RB2
RB3
RB4
RB5
RB6
RB7
PPMOUT
TXD
RXD
CTS
OH*
RXA
U?
SX18AC/SDIP_4
VCC
R5
10K
JP7
C8
10µFD16V
2 Pin Berg Header
VCC
C3
10µFD 16V
U2
MAX 232 CPE
V+
V-
C1+
C1-
C2+
C2-
C4
10µFD 16V
C7
10µFD 16V
10µFD 16V
C5
P1
T1OUT
T2OUT
R1in
R2in
T1in
T2in
R1OUT
R2OUT
GND
9 Pin DB FEMALE
VCC
VCC
R1
1k
R2
1k
D1
LED
D2
LED
RXD
CTS
TXD
RS-232
A3) Power supply
D4
DB102
L1 100µh
UA7805 U3
INPUT
GND
OUTPUT
J2
POWER JACK
C12
470µFD 16V
C15
0.1µFD 50V
C16
470µFD 16V
C13
100µFD 16V
C14
0.1µFD 50V
VCC
Title
Scenix Modem Page 1 of 2
Size
B
Number
Revision
2.1
Date: 2-Jun-1999
Sheet of
File: C:\Downloads\orcad conversion\Datel.ddb
Drawn By:



COMPONENT REFERENCE	BEL103	BEL202	V.23
R26, R27, R28, R29 = X	2.2k		5.62k
R44, R45, R46, R47 = Y	4.42k		5.62k
R39, R43 = Z	2.2k		2.80k

Title: **Scenix Modem Page 2 of 2**			
Size: B	Number		Revision: **2.1**
Date: 2-Jun-1999		Sheet of	
File: C:\Downloads\orcad conversion\Datel.ddb		Drawn By:	

APPENDIX E: v_23_originate_1_35_rev_2_1.src

```
;*****************************************************************************************
; Copyright © [05/15/1999] Scenix Semiconductor, Inc. All rights reserved.
;
; Scenix Semiconductor, Inc. assumes no responsibility or liability for
; the use of this [product, application, software, any of these products].
; Scenix Semiconductor conveys no license, implicitly or otherwise, under
; any intellectual property rights.
; Information contained in this publication regarding (e.g.: application,
; implementation) and the like is intended through suggestion only and may
; be superseded by updates. Scenix Semiconductor makes no representation
; or warranties with respect to the accuracy or use of these information,
; or infringement of patents arising from such use or otherwise.
;*****************************************************************************************
;
; Filename:     v_23_originate_1_35.src
;
; Author:       Chris Fogelklou
;               Applications Engineer
;               Scenix Semiconductor Inc.
;
; Revision:     1.35
;
; Date:         May 23, 1999.
;
; Part:         SX28AC rev. 2.5
;
; Freq:         50Mhz
;
; Compiled using Parallax SX-Key 28L software v1.05
;
; Program Description:
;               This program performs V.23 origination on the Scenix/IDC
;               modem boards V.2.1.  These specifications are followed:
;
;       User Interface
;               - Software UART will provide the modem's interface.
;                       - 1200 baud
;                       - No Parity
;                       - 8 Data Bits
;                       - 1 Stop Bit
;                       - Hardware Flow Control (CTS, RTS)
;                       - Compact AT command set
;                       - 64-byte command buffer
;                       - Dial: "ATDTxxxxxxxxx…"
;                       - Switch from data mode to command mode: "+++"
;                               To escape, wait at least 3 seconds from the last transmitted
;                               character, and type +++ with less than 1 second between each
;                               character.  The modem will return to command mode if another
;                               character is not received in 3 seconds.
;                       - Switch from command mode to data mode: "ATO"
;                       - Hang up: "ATH"
;                       - Initialize: "ATZ"
;                       - Automatic Hybrid Adjustment: "ATY"
;
;       Signal Generation/Detection Software
;               - DTMF Generation for Dialing
;                       - Tones generated: 697Hz, 770Hz, 852Hz, 941Hz, 1209Hz, 1336Hz, 1477Hz,1633Hz
;                       - On time = 100ms
;                       - Off time = 100ms
;                       - Off-hook delay time before dialing = 4 s
;                       - D/A conversion provided by filtered PPM output
;                       - Data transmission and modulation
;               - FSK transmission data rate at 75bps
;                       - Hardware flow control, 16-byte buffer, and 75bps asynchronous transmitter for
;                         data rate conversion from 1200bps to 75bps
;                       - Logic '1' (mark) modulated by 390 Hz
;                       - Logic '0' (space) modulated by 450 Hz
;                       - Transmission power = -15dB
;                       - D/A conversion provided by filtered PPM output
;                       - Data reception and demodulation
;               - FSK reception data rate at 1200bps
;                       - Logic '1' (mark) demodulated from 1300Hz carrier
;                       - Logic '0' (space) demodulated from 2100Hz carrier
;                       - Carrier detection
;                       - Timed-Zero-Cross algorithm
;               - D/A conversion
;                       - Pulse Position Modulation with maximum output frequency of 307kHz
;
;       Hardware Specifications
;               - Filtering
;                       - Low pass filter on PPM output (fc = 1633Hz)
;                       - High pass filter on FSK input (fc = 1300Hz)
;               - Hybrid (removes tx signal from rx signal)
;                       - Four settings provided for automatic hybrid adjustment for various line
;                         impedance's
;                       - Hybrid adjusted by outputting signal onto line and measuring fed-back signal
;                         with a low-resolution sigma-delta A/D converter
;                       - FSK input sensitivity = -30dB
;               - UART
;                       - RS-232 interface provided through MAX232 or similar IC
;                       - Interface provided through RXD, TXD, RTS, and CTS lines
;
;       Testing Specifications
;                       - Initial tests using function generator and off-the-shelf V.23 modems
;                       - Second round of testing performed with IDC's modem test equipment
;                       - Tests performed:
;                       - Input Sensitivity
;                       - DTMF output level
```

```
;                       - FSK output level
;                       - Error rate
;                       - FCC part 68 and FCC part 15 qualified
;                       - CTR-21 ready
;                       - All test results will be documented
;
; Program Instructions;
;               To use this program, the modem board must be connected to a serial port at
;               these settings:
;                       1200 bps
;                       No parity
;                       8 Data Bits
;                       1 stop bit
;                       Hardware flow control ON!!! (CTS/RTS)
;
;               These AT commands can be used:
;
;                       ATDT - Used to dial into a remote modem
;                       ATH - Used to hang up a call
;                       ATZ - Used to initialize the modem settings
;                       ATO - Switches back to data mode from command mode
;                       ATY - Performs automatic hybrid adjustment
;                       +++ - Switches from data mode to command mode.
;                       ?   - Re-prints the help screen to the terminal.
;
; Revision History:
;               1.10 Took semi-working V.23 code and cleaned it up.  Kept it working, but made
;               few improvements to the operation.
;               1.15 Finally got FSK receive to work error free!!! Whooopeee!!!
;               1.17 Added documentation.
;               1.20 Added carrier detection to the software
;               1.30 Added automatic hybrid adjustment to the software.
;               1.32 Automatic hybrid adjustment tweaked until working.  Component values for A/D:
;                    C = 470pF, R1 = 22k, R2 = 10k
;               1.35 Added guard times around the "+++" coming in.
;
;
;       RESOURCES:
;       Program memory: TBD
;       Data memory:    TBD
;       I/O Count:      TBD
;
;*****************************************************************************
;       PROGRAM DEFINES
;*****************************************************************************
 SX28L_assembler; Uncomment this line to use this program with the SX28L assembler
 rev_2_0        ; Uncomment this line to use on the IDC boards (rev 2.0)
;
;*****************************************************************************
; Device Directives
;*****************************************************************************
IFDEF SX28L_assembler
                device  SX28L,oscxt4,carryx             ; 28-pin device, 4 pages, 8 banks of RAM
                device  turbo,stackx_optionx            ; High speed oscillator, turbo mode,
                                                        ; option register extend, 8-level stack
ELSE
                device  pins28,pages4,banks8,carryx     ; 28-pin device, 1 pages, 8 banks of RAM
                device  oschs,turbo,optionx,stackx      ; High speed oscillator, turbo mode,
ENDIF                                                   ; option register extend, 8-level stack
                freq    50_000_000                      ; default run speed = 50MHz
                ID      'v23org11'                      ; Version = 1.1

                reset   reset_entry                     ; JUMP to reset_entry label on reset

;**************************************************************
; Equates for the FSK receive part of the modem
;**************************************************************
threshold       =       180     ; How many counts to look for for a transition from high frequency to low frequency
fsk_hysterises  =       6       ; The number of counts over/under the threshold to allow an actual transition
                                ; from high to low on RX-bit
;**************************************************************
;*****************************************************************************
; Watches (For Debug in SX_Key software V.1.0 +)
;*****************************************************************************
watch   plus_count,8,udec
watch   freq_acc_low,16,udec
watch   freq_count_low,16,uhex
watch   freq_acc_low2,16,udec
watch   freq_count_low2,16,uhex
watch   sine_index,8,udec
watch   tx_pin,1,ubin
watch   wreg,8,uhex
watch   buffer,16,fstr
watch   pop_index,8,udec
watch   push_index,8,udec
watch   fsk_trans_count,8,udec
watch   fsk_rb_past_state,8,uhex
watch   fsk_temp_trans,8,udec
watch   fsk_last_trans,8,udec
watch   wreg,8,udec
watch   cd_trans_count,8,udec
watch   cd_trans_avg_l,8,udec
watch   cd_trans_avg_h,8,udec
watch   cd_avg_count,8,udec
watch   cd_rb_past_state,8,udec
watch   carrier_detected,1,ubin
watch   A_to_D_val,8,sdec
watch   A_to_D_count,8,udec
```

```
watch   A_to_D_sample,8,sdec
watch   fdbk_l,16,udec
watch   fdbk_h,8,udec
watch   fdbk_l,8,udec
watch   fdbk_count,16,udec
watch   lowest,8,udec
watch   temp,8,udec


;***************************************************************************
; Baud rate defines
;***************************************************************************
        ; *** 150 baud
;       baud_bit        =       7                       ;for 2400 baud
;       start_delay     =       128+64+1                ; "    "    "
;       int_period      =       163                     ; "    "    "
        ; *** 600 baud
        baud_bit        =       5
        start_delay     =       32+16+1
        int_period      =       163
        ; *** 1200 baud
        baud_bit        =       4
        start_delay     =       16+8+1
        int_period      =       163
;***************************************************************************
; Equates for hybrid set-up
;***************************************************************************
f2100_h2        equ     $029    ; 2100 Hz for when A/D is on
f2100_l2        equ     $049

f450_h2         equ     $008    ; 450 Hz for when A/D is on
f450_l2         equ     $0d8

rc_450_mask     equ     %11100101                       ; Hybrid set-up for 450 ohms
rc_600_mask     equ     %11010101                       ; Hybrid set-up for 600 ohms
rc_750_mask     equ     %10110101                       ; Hybrid set-up for 750 ohms
rc_900_mask     equ     %01110101                       ; Hybrid set-up for 900 ohms

;***************************************************************************
; Equates for common data comm frequencies
;***************************************************************************
f697_h          equ     $012    ; DTMF Frequency
f697_l          equ     $09d

f770_h          equ     $014    ; DTMF Frequency
f770_l          equ     $090

f852_h          equ     $016    ; DTMF Frequency
f852_l          equ     $0c0

f941_h          equ     $019    ; DTMF Frequency
f941_l          equ     $021

f1209_h         equ     $020    ; DTMF Frequency
f1209_l         equ     $049

f1336_h         equ     $023    ; DTMF Frequency
f1336_l         equ     $0ad

f1477_h         equ     $027    ; DTMF Frequency
f1477_l         equ     $071

f1633_h         equ     $02b    ; DTMF Frequency
f1633_l         equ     $09c
;*******************************************************************************
; Equates for FSK generation
;*******************************************************************************
f390_h          equ     $00a    ; V.23 backchannel logic '1' (mark)
f390_l          equ     $06a

f450_h          equ     $00c    ; V.23 backchannel logic '0' (space)
f450_l          equ     $004

f1300_h         equ     $022    ; V.23 forward channel logic '1' (mark)
f1300_l         equ     $0b7

f2100_h         equ     $038    ; V.23 forward channel logic '0' (space)
f2100_l         equ     $015

f2225_h         equ     $03b    ; Bell 103 forward channel logic '1' (mark)
f2225_l         equ     $06b

f2025_h         equ     $036    ; Bell 103 forward channel logic '0' (space)
f2025_l         equ     $014

f1070_h         equ     $01c    ; Bell 103 backward channel logic '1' (mark)
f1070_l         equ     $093

f1270_h         equ     $021    ; Bell 103 backward channel logic '0' (space)
f1270_l         equ     $0ea


;*******************************************************************************
; Pin Definitions (These definitions are for SX DTMF DEMO boards)
;*******************************************************************************
        PPM_pin         equ     ra.0    ; PPM output pin
        rx_pin          equ     ra.1    ; RS-232 Input pin
        tx_pin          equ     ra.2    ; RS-232 Output pin
```

```
                                        ; and input on SX DTMF DEMO boards.
        led_pin         equ     rb.0    ; Flashes continually while program is running
        cntrl_1         equ     rb.2    ; Tristate cntrl_1 and set cntrl_3 low to enable the low-pass
        cntrl_3         equ     rb.5    ; filter on input.  Vice-versa for enabling the high-pass filter.
        hook            equ     rb.4    ; Goes on/off-hook.
        rts             equ     rb.6    ; RTS indicates that the PC is ready to send something.
        cts             equ     rb.7    ; CTS indicates that to the PC that the modem is ready
                                        ; to receive something.
        gain_booster    equ     ra.3    ; Setting this adds 20dB gain to the low frequency input.

;*****************************************************************************************
; Global Variables
;*****************************************************************************************
                org     $8              ; Global registers

        flags   ds      1
                dtmf_gen_en     equ     flags.0 ; Signifies whether or not DTMF output is enabled
                sine_gen_en     equ     flags.1 ; Enables the sine generator(s) for DTMF generation and
                                                ; FSK generation
                timer_flag      equ     flags.2 ; Set every time the timers roll over.
                fsk_tx_en       equ     flags.3 ; enables the fsk transmission portion of the ISR
                fsk_rx_en       equ     flags.4 ; enables the fsk reception portion of the ISR
                rx_flag         equ     flags.5 ; this flag is set when a byte is received via the UART
                fsk_rx_bit      equ     flags.6 ; this bit indicates the current state of the FSK being
                carrier_detected equ    flags.7 ; indicates the presence of a carrier
                                                ;  received.
        flags2          ds      1
                A_to_D_en       equ     flags2.0
                sample_ready    equ     flags2.1

        temp            ds      1               ; Temporary register
        temp2           ds      1               ; Temporary register
        task_switcher   ds      1               ; Used in the ISR to switch between tasks.
        push_index      ds      1               ; Used by the 64-byte buffer to store bytes.
        pop_index       ds      1               ; Used by the 64-byte buffer to retrieve bytes.
        command_index   ds      1               ; Used by the string parser to remember the current
                                                ; command being checked.

;*****************************************************************************************
; Bank 0 Variables
;*****************************************************************************************
                org     $10

sine_gen_bank           =       $
        freq_acc_low            ds      1       ; 16-bit accumulator which decides when to increment the sine wave
        freq_acc_high           ds      1       ;
        freq_count_low          ds      1       ; 16-bit counter which decides which frequency for the sine wave
        freq_count_high         ds      1       ; freq_count = Frequency * 6.83671552
        sine_index              ds      1       ; Index into the sine table for sine wave 1
        sine_index2             ds      1       ; Index into the sine table for sine wave 2
        freq_count_low2         ds      1       ; 16-bit counter which sets the sine wave frequency
        freq_count_high2        ds      1       ; freq_count = Frequency * 6.83671552
        freq_acc_high2          ds      1       ;
        freq_acc_low2           ds      1       ; 16-bit accumulator which decides when to increment the sine wave
        curr_sine               ds      1       ; The current value of the sine wave
        curr_sine2              ds      1       ; The current value of sine wave 2
        sine2_temp              ds      1       ; This register is used to do a temporary shift/add register

PPM_bank                =       $

PPM0_acc                ds      1               ; PPM accumulator
PPM0_out                ds      1               ; current PPM output (D/A)

;*****************************************************************************************
; Bank 1 Variables
;*****************************************************************************************
                org     $30
timers                  =       $
        timer_l         ds      1               ; The low byte of the 24-bit timer
        timer_h         ds      1               ; the middle byte of the 24-bit timer
        timer_hh        ds      1               ; the high byte of the 24-bit timer

serial          =       $                       ;UART bank

tx_high         ds      1                       ;hi byte to transmit
tx_low          ds      1                       ;low byte to transmit
tx_count        ds      1                       ;number of bits sent
tx_divide       ds      1                       ;xmit timing (/16) counter
rx_count        ds      1                       ;number of bits received
rx_divide       ds      1                       ;receive timing counter
rx_byte         ds      1                       ;buffer for incoming byte
rx_count2       ds      1                       ;number of bits received
rx_divide2      ds      1                       ;receive timing counter
rx_byte2        ds      1                       ;buffer for incoming byte
string          ds      1                       ;the address of the string to be sent
byte            ds      1                       ;semi-temporary serial register
plus_count      ds      1                       ;stores the number of consecutive '+''s received during
                                                ; FSK i/o mode.

;*****************************************************************************************
;       Bank 2 Variables
;*****************************************************************************************
                org     $50                     ;bank3 variables
        fsk_transmit_bank       =       $
        fsk_receive_bank        =       $
        fsk_serial_bank         =       $

        fsk_tx_high             ds      1                       ;hi byte to transmit
```

```
	fsk_tx_low		ds	1			;low byte to transmit
	fsk_tx_count		ds	1			;number of bits sent
	fsk_tx_divide		ds	1			;xmit timing (/16) counter
	fsk_tx_divide_2		ds	1

	fsk_trans_count		ds	1		; This register counts the number of counts
							; between transitions at the pin
	fsk_last_trans		ds	1
	fsk_rb_past_state	ds	1		; This register keeps track of the previous
							; state of port RB, to watch for transitions
	fsk_temp_trans		ds	1		; Temporarily stores the transition count after
							; a transition has occurred, until it can be processed.
	fsk_flags		ds	1
	fsk_answering		equ	fsk_flags.0
	fsk_tx_bit		equ	fsk_flags.1
	fsk_processing_required_1 equ	fsk_flags.2

;*****************************************************************************
;	Bank 3 Variables
;*****************************************************************************
		org	$70
	carrier_detect_bank	=	$
	cd_trans_count		ds	1
	cd_trans_avg_l		ds	1
	cd_trans_avg_h		ds	1
	cd_avg_count		ds	1
	cd_rb_past_state	ds	1

	A_to_D_bank		=	$
	A_to_D_val		ds	1
	A_to_D_count		ds	1
	A_to_D_sample		ds	1
	fdbk_l			ds	1
	fdbk_h			ds	1
	fdbk_count		ds	1
	fdbk_count_2		ds	1
	lowest			ds	1
;*************************************************************
; Bank 4, 5, 6, 7 (for 64-byte buffer, but can be reused.)
;*************************************************************
		org	$90
	buffer		=	$
		org	$b0
	buffer2		=	$
		org	$d0
	buffer3		=	$
		org	$f0
	buffer4		=	$

;************************ Beginning of program space ***************************
		org	$0
;*****************************************************************************
; Interrupt
;
; With a retiw value of -163 and an oscillator frequency of 50MHz, this
; code runs every 3.26us.
;*****************************************************************************
PPM_output
		bank	PPM_bank		; Update the PPM pin
		clc
		add	PPM0_acc,PPM0_out
		snc
		setb	PPM_pin
		sc
		clrb	PPM_pin
;*************************************************************************
		sb	A_to_D_en
		jmp	@ISR	; The ISR is in the second page, so go there.
		call	@A_to_D
		mov	w,#-120
		break
		retiw
;*****************************************************************************
reset_entry
; Program Starts Here on Power Up
;*****************************************************************************

	;*********************************************************************
	; First, call init to initialize the program
	;*********************************************************************
		call	@init

		mov	!option,#%00011111	; enable wreg and rtcc interrupt
		setb	tx_pin			; set the RS-232 tx_pin
		setb	CTS			; Don't allow PC to transmit
		mov	w,#25			; delay 250 milliseconds
		call	@delay_10n_ms

	;*********************************************************************
	; Send "hello" string
	;*********************************************************************
		mov	w,#_hello		; say 'hello'
		call	@send_string
		mov	w,#_help
		call	@send_string

	;*********************************************************************
	; Send prompt
```

```
        ;*********************************************************************
main_2
_send_prompt    mov     w,#_prompt              ; send prompt
                call    @send_string
                clrb    CTS                     ; Allow PC to transmit
                clr     push_index              ; Clear the buffer_push pointer
                clr     pop_index               ; Clear the buffer_pop pointer

        ;*********************************************************************
        ; Fill the command buffer with input characters.  Backspace will delete
        ; the last value entered.
        ;*********************************************************************
_cmd_loop
                jnb     rx_flag,$                       ; Wait until we receive a byte via. RS-232
                clrb    rx_flag                         ; clear the flag
                bank    serial
                mov     byte,rx_byte                    ; Move the received byte to 'byte' and
                call    @uppercase                      ; convert it to uppercase
                mov     w,#$20                          ; compare the byte to ' '
                xor     w,byte
                jz      _cmd_loop                       ; If byte == space, ignore it.
                mov     w,#$0a                          ; compare the byte to LF
                xor     w,byte
                jz      _cmd_loop                       ; If byte == line feed, ignore it.
                mov     w,#$0d                          ; compare the byte to CR
                xor     w,byte
                jz      :enter                          ; if byte == CR, parse the string.
                mov     w,byte                          ; if it does not resemble the above characters, echo it.
                call    @send_byte                      ; send via. RS-232
                mov     w,#$08                          ; compare the byte to a backspace.
                xor     w,byte
                jz      :backspace                      ; if it equals a backspace, delete one character in the buffer.
                call    @buffer_push                    ; otherwise, store it
                jmp     _cmd_loop                       ; and come back for more.
:backspace
                call    @buffer_backspace
                jmp     _cmd_loop

:enter                                                  ; If the user presses enter, then parse the string.


        ;*************************************************************************
        ; String parser (Checks to see if buffer = any commands)
        ; -Checks contents of ascii buffer against any commands stored in ROM
        ; -If a command = the contents of the ascii buffer, a routine will be called
        ; -Each routine MUST perform a retw 0 on exit, or parse_string will not
        ;  know that a routine has run and it should exit back to command mode.
        ; -Exits back to command mode when it detects a zero after the table look-up.
        ; -Outputs 'OK' if no commands are matched.
        ;*************************************************************************
parse_string
                clr     pop_index                       ; Clear the index into the ascii buffer
                clr     command_index                   ; And the index into the commands

:loop           call    @buffer_get                     ; Get a vale from the buffer at ascii_index
                call    command_table                   ; Get a character from one of the commands
                test    wreg                            ; If the return value is 0, then this matched
                jz      :nothing                        ; the command and ran a routine.  Exit.
                bank    serial
                xor     w,byte                          ; compare the command's character with the
                jnz     :not_equal                      ; buffer's character.
                call    @inc_pop_index                  ; Increment the index into the buffer.
                jmp     :loop
:not_equal
                inc     command_index                   ; If the buffer did not equal the command,
                clr     pop_index                       ; start from the beginning of a new command
                stc
                cjne    command_index,#6,:loop          ; compare command_index with 5  (This number = # of commands)
:nothing        clrb    fsk_rx_en
                setb    tx_pin
                mov     w,#20
                call    @delay_10n_ms
                mov     w,#_CR
                call    @send_string
                mov     w,#_OK                          ; If we have checked all 5 commands, then this
                call    @send_string                    ; did not equal any so send an 'OK' message.
:done
                bank    buffer
                clr     pop_index
                clr     push_index
                clr     buffer
                jmp     _send_prompt

;*************************************************************************
command_table
                mov     w,command_index
                clc
                add     pc,w
                jmp     command_1
                jmp     command_2
                jmp     command_3
                jmp     command_4
                jmp     command_5
                jmp     command_6
;*************************************************************************
command_1                                       ; Dial command
                mov     w,pop_index
                add     PC,w
```

```
		retw	'A'
		retw	'T'
		retw	'D'
		retw	'T'
		jmp	DIAL_MODE
;*************************************************************************
command_2					; Hang up command
		mov	w,pop_index
		add	PC,w
		retw	'A'
		retw	'T'
		retw	'H'
		jmp	HANG_UP
;*************************************************************************
command_3					; Initialize
		mov	w,pop_index
		add	PC,w
		retw	'A'
		retw	'T'
		retw	'Z'
		jmp	INITIALIZE
;*************************************************************************
command_4					; Data mode
		mov	w,pop_index
		add	PC,w
		retw	'A'
		retw	'T'
		retw	'O'
		jmp	FSK_IO
;*************************************************************************
command_5					; Hybrid Set-up
		mov	w,pop_index
		add	PC,w
		retw	'A'
		retw	'T'
		retw	'Y'
		jmp	HYBRID
;*************************************************************************
command_6					; Help
		mov	w,pop_index
		add	PC,w
		retw	'?'
		jmp	HELP
;*************************************************************************
; END of String parser (Checks to see if buffer = any commands)
;*************************************************************************
;*************************************************************************
HYBRID
;	This routine is jumped to when the string parser decodes the
;	string "ATY".  This routine finds the closest hybrid match for
;	the line to which the modem is connected.  The modem must be
;	connected to the telephone line for this routine to adjust properly.
;	This routine uses the sigma-delta A/D to average many low-resolution
;	samples, obtaining the average input voltage.  The routine samples
;	the returned signal for each hybrid setting, 450 ohms, 600 ohms,
;	and 900 ohms.  The setting which produces the smallest return signal
;	is used.
;*************************************************************************
	;*****************************************************************
	; Go off-hook, wait 3 seconds, and dial a '1' to quiet the line
	;*****************************************************************
		clrb	hook			; go off hook
		mov	w,#30
		call	@delay_100n_ms		; wait 3 seconds before dialing
		mov	w,#'1'
		call	@load_frequencies
		call	@dial_it		; send out a '1'
:loop		clr	flags			; disable everything
		clr	flags2			; disable everything
	;*****************************************************************
	; Wait 1 second, and output a signal of 2100Hz to disable the line
	; equalizers.
	;*****************************************************************
		mov	w,#10			; wait 1 second
		call	@delay_100n_ms
		call	@answer_tone
	;*****************************************************************
	; Output a signal, try each hybrid setting, and sample the returned signal
	; to get the returned amplitude.
	;*****************************************************************
		bank	sine_gen_bank		; now, since interrupt rate is changing, output another frequency which will
		mov	freq_count_high2,#f1633_h ; make it through the output low-pass filter.
		mov	freq_count_low2,#f1633_l
		setb	sine_gen_en

		mov	m,#$0f			; set up hybrid to match a 450 ohm impedance
; Is 450 ohms the best match?
		mov	!rc,#rc_450_mask
		clrb	rc.4
		call	@get_signal_amplitude	; get the amount of signal sent back
		mov	lowest,fdbk_h
		clr	temp
; Is 600 ohms the best match?
		mov	!rc,#rc_600_mask	; ditto for 600 ohms
		clrb	rc.5
		call	@get_signal_amplitude
		stc
		cjb	lowest,fdbk_h,:lower_1
```

```
                mov     temp,#1                 ; if the amount of signal is less, then save this one
                mov     lowest,fdbk_h

:lower_1
; Is 750 ohms the best match?
                mov     !rc,#rc_750_mask        ; ditto for 750 ohms
                clrb    rc.6
                call    @get_signal_amplitude
                stc
                cjb     lowest,fdbk_h,:lower_2
                mov     temp,#2
                mov     lowest,fdbk_h
:lower_2
; Is 900 ohms the best match?
                mov     !rc,#rc_900_mask        ; ditto for 900 ohms
                clrb    rc.7
                call    @get_signal_amplitude
                stc
                cjb     lowest,fdbk_h,:lower_3
                mov     temp,#3
                mov     lowest,fdbk_h
:lower_3
        ;*****************************************************************
        ; Disable the sine generators and go back on-hook.
        ;*****************************************************************
                call    @disable_o
                setb    hook

        ;*****************************************************************
        ; Use the best setting, and output the setting used.
        ;*****************************************************************
                mov     w,temp
                clc
                jmp     pc+w                    ; depending on which one gave the lowest feedback, set the port for this hybrid.
                jmp     :imp_450
                jmp     :imp_600
                jmp     :imp_750
                jmp     :imp_900
:imp_450
                mov     !rc,#rc_450_mask        ; set up hybrid for 450 ohms
                clrb    rc.4
                mov     w,#_450
                jmp     :out
:imp_600
                mov     !rc,#rc_600_mask        ; set up hybrid for 600 ohms
                clrb    rc.5
                mov     w,#_600
                jmp     :out
:imp_750
                mov     !rc,#rc_750_mask        ; set up hybrid for 750 ohms
                clrb    rc.6
                mov     w,#_750
                jmp     :out
:imp_900
                mov     !rc,#rc_900_mask        ; set up hybrid for 900 ohms
                clrb    rc.7
                mov     w,#_900
                jmp     :out
:out
                call    @send_string            ; output the hybrid setup
                mov     w,rc
                and     w,#%00001111
                mov     rc,w
        ;*****************************************************************
        ; Use the best setting, and output the setting used.
        ;*****************************************************************
                retw    0                       ; return from hybrid set-up
;*************************************************************************
; DONE HYBRID SET-UP
;*************************************************************************
        ;*********************************************************************
        ; Hang Up
        ;*********************************************************************
HANG_UP
                call    @disable_o
                clrb    fsk_rx_en       ; Disable fsk detection
                mov     w,#50
                call    @delay_100n_ms  ; Pause for 5 seconds.
                setb    hook            ; hang-up
                retw    0
        ;*********************************************************************
        ; Initialize
        ;*********************************************************************
INITIALIZE
                mov     w,#10
                call    @delay_100n_ms  ; Pause for 1 second
                call    @init
                clr     flags
                retw    0


        ;*********************************************************************
        ; Send Help string
        ;*********************************************************************
HELP            mov     w,#_HELP
                call    @send_string
                retw    0
```

```
	;**************************************************************************
	; Dial Mode:
	; -Dials contents of ascii buffer, starting from location pointed
	;  to by ascii_index.
	; -Responds to these commands:
	;	0-9, *, #	- Dials the specified number
	;	,		- Pause for 2 seconds
	; -Jumps to data mode after dialing.
	;**************************************************************************
DIAL_MODE
		clrb	sine_gen_en			; Disable sine generation
		clrb	fsk_tx_en			; Disable fsk generation
		clrb	dtmf_gen_en			; Disable dtmf generation
		clrb	hook				; go off-hook
		mov	m,#$0f
		mov	w,#%01101010			; rb.5 (cntrl_3) is tristate, rb.2 (cntrl_1) is low
		mov	!rb,w
		clrb	cntrl_1				; Enable lowest low-pass filter on output
		mov	w,#40				; delay 4 seconds before dialing
		call	@delay_100n_ms
		mov	w,#_CR				; Send a carriage return
		call	@send_string
		mov	w,#_DIALING			; send "Dialing" to screen
		call	@send_string
		bank	serial

:dial_loop	call	@buffer_get			; Get a character from the buffer
		call	@uppercase			; convert it to uppercase
		mov	w,byte				; test byte for zero
		snz
		jmp	:originate_mode			; If byte is zero, dialing is done.
		call	@send_byte
		stc
		cje	byte,#',',:pause		; if the character = ',', pause for 2s
		call	@digit_2_index			; convert the ascii digit to an
							; index value
		call	@load_frequencies		; load the frequency registers
		call	@dial_it			; dial the number for 60ms and return.
:inc		call	@inc_pop_index			; increment the index into the table
		jmp	:dial_loop
:pause
		mov	w,#20				; delay 2s
		call	@delay_100n_ms
		jmp	:inc


:originate_mode
	;*****************************************************************
	; Set/clear proper flags for origination
	;*****************************************************************
		bank	fsk_transmit_bank
		clr	fsk_tx_divide_2		; clear the transmit-divider
		clr	flags			; clear all flags.
		clrb	fsk_answering		; we are not answering.
		setb	fsk_tx_bit		; set the transmit bit to logic '1'
		setb	sine_gen_en		; enable the sine generators
		setb	fsk_tx_en		; enable the fsk transmitter
		mov	w,#50			; delay 5 seconds after dialing to wait for carrier
		call	@delay_100n_ms
		jb	carrier_detected,FSK_IO ; if there still is no carrier, exit
		mov	w,#50			; delay 5 seconds after dialing to wait for carrier
		call	@delay_100n_ms
		jb	carrier_detected,FSK_IO ; if there still is no carrier, exit
		mov	w,#100			; delay 10 seconds after dialing to wait for carrier
		call	@delay_100n_ms
		jb	carrier_detected,FSK_IO
		mov	w,#150			; delay 15 seconds after dialing to wait for carrier
		call	@delay_100n_ms
		jb	carrier_detected,FSK_IO
no_carrier
		clrb	fsk_rx_en
		setb	tx_pin
		mov	w,#80			; give carrier 8 more seconds to re-appear
		call	@delay_100n_ms
		jb	carrier_detected,FSK_IO_AGAIN
		mov	w,#_no_carrier
		call	@send_string
		jmp	INITIALIZE


	;*****************************************************************
	; Once at FSK I/O mode, the program sends/receives data.  In
	; originate mode, the send is at 75bps and the receive is at 1200bps.
	; Because of the difference in baud rates, hardware flow control
	; is used.  CTS is disabled when the buffer is close to capacity,
	; and re-enabled when the buffer is completely empty.
	;*****************************************************************
FSK_IO


		mov	w,#_DATA_MODE		; Send "connect" message
		call	@send_string
FSK_IO_AGAIN	clr	plus_count		; clear the plus count
		clr	push_index		; clear the push pointer to buffer
		clr	pop_index		; clear the pop pointer to buffer
		setb	fsk_rx_en		; enable the FSK reception part of ISR
		mov	m,#$0f
		mov	w,#%01101010		; rb.5 (cntrl_3) is tristate, rb.2 (cntrl_1) is low
```

```
                mov     !rb,w
                clrb    cntrl_1                 ; Enable lowest low-pass filter on output
                clrb    cts                     ; clear CTS to tell PC "ready for data"
        ;*****************************************************************
        ; This is the main loop for FSK I/O.  Sends FSK bytes, and receives
        ; bytes from the UART.  The FSK receive portion of FSK I/O is completely
        ; handled by the ISR
        ;*****************************************************************

:loop2

                jnb     timer_flag,:no_timeout  ; if (timer_flag)
                bank    serial
                test    plus_count              ;       if (plus_count)
                jz      :no_timeout
                mov     w,plus_count            ;               if (plus_count==3)
                xor     w,#3                    ;                       return;
                snz
                retw    0
                jmp     :clr_plus_count         ;               else clr_plus_count();
                                                ;       else no_timout();
                                                ; else no_timeout();
:no_timeout     jnb     carrier_detected,no_carrier
                jb      rx_flag,:got_byte       ; Received a byte of data.  Handle it.
                bank    fsk_transmit_bank       ; If no byte, check to see if we need to transmit
                test    fsk_tx_count            ; Are we transmitting anything?
                sz                              ;       if no, then send next byte.
                jmp     :loop2                  ;       else jump here forever (ISR does all the work)

                mov     w,pop_index             ; If pop_index == push_index, everything in the buffer has been sent.
                xor     w,push_index
                sz
                jmp     :not_empty_yet
        ;*****************************************************************
        ; The buffer is empty: initialize the buffer and enable CTS.
        ;*****************************************************************

:empty
                clr     push_index              ; so clear the buffer indexes
                clr     pop_index
                clrb    cts                     ; and clear cts to allow more data from DCE
                jmp     :loop2
        ;*****************************************************************
        ; The buffer is not empty, keep sending stuff..
        ;*****************************************************************
:not_empty_yet
                call    @buffer_get             ; if the buffer is not empty, get the next byte
                call    @fsk_send_byte          ; from the buffer and send it via. FSK
                call    @inc_pop_index          ; and increment the pop index
                and     pop_index,#$0f
                jmp     :loop2
        ;*****************************************************************
        ; The we just received a byte, so put it on the buffer.
        ;*****************************************************************
:got_byte

                bank    serial
                clrb    rx_flag
                mov     byte,rx_byte
                call    @buffer_push
        ;*****************************************************************
        ; Check to see if the pop index is at (push index + 5)
        ;*****************************************************************
                and     push_index,#$0f
                mov     w,#5
                clc
                add     w,push_index
                and     w,#$0f                  ; keep push index < 16
                xor     w,pop_index             ; if (push_index + 5 == pop_index, the buffer is almost full so indicate this)
                snz
                setb    cts                     ; If push index == pop index, disable CTS
                bank    serial
                mov     w,#'+'                  ; If the byte = '+', increment plus_count, otherwise, plus_count == 0
                xor     w,rx_byte               ; If byte = '+'
                jz      :plus_received          ;       plus_received();

:clr_plus_count                                 ; Else
                clr     plus_count              ;       clr_plus_count();
                mov     w,#255                  ;
                call    @reset_timers           ;
                jmp     :loop2                  ;

:plus_received                                  ; plus_received();
                test    plus_count              ; If !(plus_count)
                jz      :zero_plus_count        ;       zero_plus_count();
:some_pluses    jb      timer_flag,:clr_plus_count; else if (timer_flag)
:inc_plus_count inc     plus_count              ;               clr_plus_count();
                mov     w,#200                  ;       else
                call    @reset_timers           ;               inc_plus_count();
                jmp     :loop2

:zero_plus_count
                sb      timer_flag              ; If (timer_flag)
                jmp     :clr_plus_count         ;       clr_plus_count();
                jmp     :inc_plus_count         ; else  inc_plus_count

;*****************************************************************
; Miscellaneous subroutines....
```

```
;*************************************************************************
org     $200
;*************************************************************************
reset_timers
; This subroutine times 'w' ticks, and returns with a '1' in w when
; the specified time has timed out.  Each tick is 13.35296 ms.
; This subroutine uses the TEMP2 register.  Call this routine with w = 0
; to poll for a time_out.
;*************************************************************************
        bank    timers
        not     w
        inc     wreg
        mov     timer_h,w
        clr     timer_l
        clrb    timer_flag
        retp

;*************************************************************************
buffer_push
; This subroutine pushes the contents of byte onto the 64-byte ascii buffer.
;*************************************************************************
        bank    serial                  ; Move the byte into the buffer
        mov     temp,byte
        mov     fsr,#buffer
        clc
        add     fsr,push_index
        mov     indf,temp
                                        ; Increment index and keep it in range
        call    @inc_push_index
        mov     fsr,#buffer     ; Null terminate the buffer.
        clc
        add     fsr,push_index
        clr     indf
        bank    serial
        retp
;*************************************************************************
;*************************************************************************
buffer_backspace
; This subroutine deletes one value of the buffer and decrements the index
;*************************************************************************
        dec     push_index
        and     push_index,#%01101111

        mov     fsr,#buffer
        clc
        add     fsr,push_index
        clr     indf
        bank    serial
        retp
;*************************************************************************
inc_pop_index
;*************************************************************************
        mov     fsr,#pop_index
        jmp     inc_index
;*************************************************************************
inc_push_index
;*************************************************************************
        mov     fsr,#push_index
;*************************************************************************
inc_index
; This subroutine increments the index into the buffer
;*************************************************************************
        mov     w,indf
        and     w,#%00001111
        xor     w,#%00001111
        jnz     :not_on_verge
        inc     indf
        mov     w,#16
        clc
        add     w,indf
        and     w,#$7f
        mov     indf,w
        retp
:not_on_verge
        inc     indf
        retp
;*************************************************************************
buffer_get
; This subroutine retrieves the buffered value at index
;*************************************************************************
        mov     fsr,#buffer
        clc
        add     fsr,pop_index
        mov     w,indf
        bank    serial
        mov     byte,w

        retp
;*************************************************************************
delay_10n_ms
; This subroutine delays 'w'*10 milliseconds. (not exactly, but pretty close)
; This subroutine uses the TEMP register
; INPUT         w       -       w/10 milliseconds to delay for.
; OUTPUT        Returns after 10 * n milliseconds.
;*************************************************************************
        mov     temp,w
        bank    timers
:loop   clrb    timer_flag      ; This loop delays for 10ms
```

```
        mov     timer_h,#$0ff
        mov     timer_l,#$041
        jnb     timer_flag,$
        dec     temp            ; do it w-1 times.
        jnz     :loop
        clrb    timer_flag
        retp
;*************************************************************************
delay_100n_ms
; This subroutine delays 'w'*100 milliseconds. (not exactly, but pretty close)
; This subroutine uses the TEMP register
; INPUT         w       -       w/100 milliseconds to delay for.
; OUTPUT        Returns after 100 * n milliseconds.
;*************************************************************************
        mov     temp,w
        bank    timers
:loop   clrb    timer_flag      ; This loop delays for 10ms
        mov     timer_h,#$0f8
        mov     timer_l,#$083
        jnb     timer_flag,$
        dec     temp            ; do it w-1 times.
        jnz     :loop
        clrb    timer_flag
        retp
;*************************************************************************
zero_ram
; Subroutine - Zero all ram.
; INPUTS:       None
; OUTPUTS:      All ram locations (except special function registers) are = 0
;*************************************************************************
                CLR     FSR
:loop           SB      FSR.4                   ;are we on low half of bank?
                SETB    FSR.3                   ;If so, don't touch regs 0-7
                CLR     IND                     ;clear using indirect addressing
                IJNZ    FSR,:loop               ;repeat until done
                retp
;*************************************************************************
; Subroutine - Disable the outputs
; Load DC value into PPM and disable the output switch.
;*************************************************************************
disable_o
                bank    PPM_bank        ; input mode.
                mov     PPM0_out,#128   ; put 2.5V DC on PPM output pin
                clrb    sine_gen_en
                clrb    dtmf_gen_en
                clrb    fsk_tx_en
                retp
;*************************************************************************
init
;       Initializes the program.
;*************************************************************************
                mov     m,#$0d                  ; Enable CMOS inputs
                mov     !ra,#$00
                mov     !rb,#$00
                mov     !rc,#$00
                mov     m,#$0c                  ; Enable Schmitt triggers
;               mov     !rb,#%11111101
                mov     m,#$0f
                mov      ra,#%0110              ; init ra
                mov     !ra,#%0010              ;
                mov      rb,#%11000000          ; init rb
                mov     !rb,#%01001010          ;
                mov     w,#%01101010                    ; rb.5 (cntrl_3) is tristate, rb.2 (cntrl_1) is low
                mov     !rb,w
                clrb    cntrl_1                         ; Enable lowest low-pass filter on output
                mov     w,#%11010101                    ; Hybrid set-up for 600 ohms
                mov     !rc,w
                setb    hook                    ; go on hook.
                clrb    cts
                setb    led_pin                 ; turn on LED
                clr     flags                   ; Clear all flags
                call    zero_ram
                call    @disable_o

                retp
;*************************************************************************
; Subroutine - Get byte via serial port and echo it back to the serial port
; INPUTS:
;       -NONE
; OUTPUTS:
;       -received byte in rx_byte
;*************************************************************************
get_byte        jnb     rx_flag,$               ;wait till byte is received
                clrb    rx_flag                 ;reset the receive flag
                bank    serial                  ;switch to serial bank
                mov     byte,rx_byte            ;store byte (copy using W)
                                                ; & fall through to echo char back
;*************************************************************************
; Subroutine - Send byte via serial port
; INPUTS:
;       w       -       The byte to be sent via RS-232
;*************************************************************************
send_byte       bank    serial

:wait           test    tx_count                ;wait for not busy
                jnz     :wait                   ;

                not     w                       ;ready bits (inverse logic)
```

```
                mov     tx_high,w               ; store data byte
                setb    tx_low.7                ; set up start bit
                mov     tx_count,#10            ;1 start + 8 data + 1 stop bit
                RETP                            ;leave and fix page bits

;**************************************************************************
; Subroutine - Send byte via serial port
; INPUTS:
;       w       -       The byte to be sent via RS-232
;**************************************************************************
fsk_send_byte   bank    fsk_serial_bank

:wait           test    fsk_tx_count                ;wait for not busy
                jnz     :wait                   ;

                not     w                       ;ready bits (inverse logic)
                mov     fsk_tx_high,w               ; store data byte
                setb    fsk_tx_low.7                ; set up start bit
                mov     fsk_tx_count,#10            ;1 start + 8 data + 1 stop bit
                RETP                            ;leave and fix page bits

;**************************************************************************
; Subroutine - Send string pointed to by address in W register
; INPUTS:
;       w       -       The address of a null-terminated string in program
;                       memory
; OUTPUTS:
;       outputs the string via. RS-232
;**************************************************************************
send_string     bank    serial
                mov     string,w                ;store string address
:loop           mov     w,string                ;read next string character
                mov     m,#3                    ; with indirect addressing
                iread                           ; using the mode register
                mov     m,#$F                   ;reset the mode register
                test    w                       ;are we at the last char?
                snz                             ;if not=0, skip ahead
                RETP                            ;yes, leave & fix page bits
                call    send_byte               ;not 0, so send character
                inc     string                  ;point to next character
                jmp     :loop                   ;loop until done

;**************************************************************************
; Subroutine - Make byte uppercase
; INPUTS:
;       byte    -       The byte to be converted
; OUTPUTS:
;       byte    -       The uppercase byte
;**************************************************************************
uppercase       stc
                csae      byte,#'a'             ;if byte is lowercase, then skip ahead
                RETP
                stc
                sub     byte,#'a'-'A'           ;change byte to uppercase
                RETP                            ;leave and fix page bits
;**************************************************************************

org     $300
;**************************************************************************
; String data (for RS-232 output) and tables
;**************************************************************************
_hello          dw      13,10,'V.23 Originate V.1.34',13,10,0
_instructions   dw      '- ? For Help',0
_DIALING        dw      'DIAL ',0
_PROMPT         dw      13,10,'>',0
_OK             dw      'OK',13,10,0
_CR             dw      13,10,0
_DATA_MODE      dw      13,10,'CONNECT 1275',13,10,0
_no_carrier     dw      13,10,'NO CARRIER',0
_HELP           dw      13,10,'ATDT- Dial',13,10,'ATH - Hang Up',13,10,'ATO - Data Mode',13,10,'ATZ - Init',13,10,'ATY - Hybrid',13,10,'+++
_450            dw      13,10,'450 ohms',13,10,0
_600            dw      13,10,'600 ohms',13,10,0
_750            dw      13,10,'750 ohms',13,10,0
_900            dw      13,10,'900 ohms',13,10,0

;**************************************************************************
org     $400    ; FSK subroutines and the Interrupt Service Routine.
;**********************************************************************
get_signal_amplitude
;       This routine samples the signal coming on the Sigma-Delta A/D
;       and returns with an 8-bit value in fdbk_h which indicates the
;       level found.
;**********************************************************************
                bank    A_to_D_bank
                mov     fdbk_count_2,#%00001111
                clr     fdbk_count
                clr     fdbk_h
                clr     fdbk_l
                clr     A_to_D_count
                clrb    sample_ready

                setb    A_to_D_en
:init_loop      jnb     sample_ready,$          ; wait for the A/D to settle
                clrb    sample_ready            ; (Cap value should be close to 2.5V)
                dec     fdbk_count
                sz
                jmp     :init_loop
```

```
:loop           jnb     sample_ready,$          ; Okay, now start getting A/D samples.
                clrb    sample_ready
                mov     w,A_to_D_sample         ; move the sample to w

                jnb     wreg.7,:pos             ; if it is negative, make it positive
:neg            not     w
                inc     wreg

:pos            clc                             ; add the sample to the fdbk register
                add     fdbk_l,w
                snc
                inc     fdbk_h
                dec     fdbk_count              ; do this until fdbk_count = 0
                snz
                dec     fdbk_count_2
                sz
                jmp     :loop                   ; if fdbk_count != 0 , do again
                clrb    A_to_D_en
                retp                            ; Return with amplitude in fdbk_h register
;**************************************************************************
answer_tone
;       This subroutine sends out an answer tone of 2100Hz for 3 seconds.
;**************************************************************************
                bank    sine_gen_bank           ; send out the answer tone for 3 seconds
                clr     curr_sine
                mov     freq_count_high2,#f2100_h
                mov     freq_count_low2,#f2100_l
                setb    sine_gen_en             ; enable the FSK transmitter
                mov     w,#30
                call    @delay_100n_ms
                retp
;**************************************************************************
; THESE ROUTINES ARE RUN WITHIN THE ISR... DO NOT CALL THEM FROM THE MAINLINE.
;**************************************************************************
;******************************************************************************
A_to_D
                mov     w,<<rc
                not     w
                and     w,#%00001111
                mov     rc,w

                jnb     sine_gen_en,:no_sine
                call    @sine_generator2
                bank    sine_gen_bank
                rl      curr_sine2
                call    @SINE_out                  ; Output each discrete value of the sine table
:no_sine
                bank    A_to_D_bank
                sb      rc.3
                dec     A_to_D_val
                snb     rc.3
                inc     A_to_D_val

                inc     A_to_D_count
                and     A_to_D_count,#%00001111
                sz
                retp
                mov     w,A_to_D_val
                mov     A_to_D_sample,w
                clr     A_to_D_val
                setb    sample_ready
                retp
;******************************************************************************
carrier_detect
                bank    carrier_detect_bank
                inc     cd_trans_count
                jnz     :no_rollover
                dec     cd_trans_count
                jmp     :sample
:no_rollover
                mov     w,rb
                xor     w,cd_rb_past_state
                and     w,#%00000010
                snz
                retp
                xor     cd_rb_past_state,w
                sb      cd_rb_past_state.1
                retp
:sample
                clc
                mov     w,cd_trans_count
                add     cd_trans_avg_l,w
                snc
                inc     cd_trans_avg_h
                clr     cd_trans_count
                inc     cd_avg_count
                sz
                retp
                setb    carrier_detected
                mov     w,#-8
                clc
                add     w,cd_trans_avg_h
                sc
                clrb    carrier_detected
                mov     w,#-16
                clc
                add     w,cd_trans_avg_h
                snc
```

```
                clrb    carrier_detected
                clr     cd_trans_avg_h
                clr     cd_trans_avg_l
                retp
;*************************************************************************
fsk_receive_main        ; This code is speed critical and runs in every
                        ; ISR.  It increments the FSK transition couters
                        ; and checks for a transition.  If a transition
                        ; has occured, it sets a flag, and saves the
                        ; transition count for later processing by the
                        ; fsk_receive_processing1 subroutine.
;*************************************************************************
                bank    fsk_receive_bank
                sb      fsk_rx_en
                retp
                inc     fsk_trans_count
                snz
                dec     fsk_trans_count
                mov     w,fsk_rb_past_state
                xor     w,rb
                and     w,#%00000010
                snz
                retp
                xor     fsk_rb_past_state,w
                setb    fsk_processing_required_1
                mov     fsk_temp_trans,fsk_trans_count
                clr     fsk_trans_count
                retp
;*************************************************************************
fsk_receive_main_2      ; This code removes some of the jitter away from
                        ; the low frequency detection algorithm by
                        ; continuously checking the transition count
                        ; to see if it has now reached a point where it
                        ; is safe to say that there is no high frequency
                        ; present.
;*************************************************************************
                bank    fsk_receive_bank
                sb      fsk_rx_en
                retp
                clc
                mov     w,#-(threshold+fsk_hysterises)
                add     w,fsk_trans_count
                snc
;               setb    fsk_rx_bit
;               setb    test_pin
                setb    tx_pin
                add     w,fsk_last_trans
                snc
;               setb    test_pin
;               setb    fsk_rx_bit
                setb    tx_pin
                retp
;*************************************************************************
fsk_receive_processing1 ; This subroutine runs only when a transition has
                        ; occurred.  It adds the last transition count
                        ; to the current one and checks this against the
                        ; high/low frequency threshold.  If the transition
                        ; count is below the threshold, the fsk_rx_bit
                        ; flag is cleared.
;*************************************************************************
        bank    fsk_receive_bank
        sb      fsk_processing_required_1
        retp                                ; Exit if disabled
        clrb    fsk_processing_required_1
        mov     w,#-25                      ; compare the transition count with 5
        clc
        add     w,fsk_temp_trans
        jnc     :glitch                     ; If the transition count is less than 5, handle the glitch.
        mov     w,#-(threshold-fsk_hysterises)  ; compare the transition count with
        add     w,fsk_temp_trans            ; the threshold (-hysterises)
        snc
        mov     w,#$ff
        clc
        add     w,fsk_last_trans
        sc
;       clrb    test_pin
;       clrb    fsk_rx_bit
        clrb    tx_pin                      ; Clear the TX_PIN if the transition count is less than the threshold
        mov     fsk_last_trans,fsk_temp_trans   ; save the last transition count.
        retp
:glitch
        clc                                 ; If this transition count was a glitch, then save the result as part
        mov     w,fsk_last_trans            ; of the last transition
        add     w,fsk_temp_trans
        snc
        mov     w,#$ff
        mov     fsk_last_trans,w
        retp
;*************************************************************************
task_manager
; This portion of the ISR allows 1 of 16 separate tasks to run in each
; interrupt.
;*************************************************************************
                inc     task_switcher
                mov     w,task_switcher
                and     w,#$0f
                clc
                jmp     pc+w
```

```
                ;*** TASKS ***
                jmp     fsk_receive_main_2      ;0
                jmp     transmit                ;1
                jmp     receive                 ;2
                jmp     fsk_transmit_uart       ;3
                jmp     fsk_receive_main_2      ;4
                jmp     transmit_fsk            ;5
                jmp     do_timers               ;6
                jmp     fsk_receive_processing1 ;7
                jmp     fsk_receive_main_2      ;8
                jmp     carrier_detect          ;9
                retp                            ;10
                retp                            ;11
                jmp     fsk_receive_main_2      ;12
                retp                            ;13
                retp                            ;14
                retp                            ;15
                jmp     fsk_receive_main_2      ;16
                retp                            ; (just in case)

;**************************************************************************
fsk_transmit_uart
; This is an asynchronous RS-232 transmitter
; INPUTS:
;       tx_divide.baud_bit  -   Transmitter only executes when this bit is = 1
;       tx_high             -   Part of the data to be transmitted
;       tx_low              -   Some more of the data to be transmitted
;       tx_count            -   Counter which counts the number of bits transmitted.
; OUTPUTS:
;       tx_pin              -   Sets/Clears this pin to accomplish the transmission.
;**************************************************************************
                bank    fsk_serial_bank
                sb      fsk_answering
                inc     fsk_tx_divide_2
                and     fsk_tx_divide_2,#$0f        ; Divide the 1200bps UART by 16 to
                                                    ; achieve 75bps
                sz
                retp
                clrb    fsk_tx_divide.baud_bit      ;clear xmit timing count flag
                inc     fsk_tx_divide               ;only execute the transmit routine
                STZ                             ;set zero flag for test
                SNB     fsk_tx_divide.baud_bit      ; every 2^baud_bit interrupt
                test    fsk_tx_count                ;are we sending?
                snz
                retp                            ;if not, go to :receive
                clc                             ;yes, ready stop bit
                rr      fsk_tx_high                 ; and shift to next bit
                rr      fsk_tx_low                  ;
                dec     fsk_tx_count                ;decrement bit counter
                movb    fsk_tx_bit,/fsk_tx_low.6        ;output next bit
                retp
;**************************************************************************
transmit
; This is an asynchronous RS-232 transmitter
; INPUTS:
;       tx_divide.baud_bit  -   Transmitter only executes when this bit is = 1
;       tx_high             -   Part of the data to be transmitted
;       tx_low              -   Some more of the data to be transmitted
;       tx_count            -   Counter which counts the number of bits transmitted.
; OUTPUTS:
;       tx_pin              -   Sets/Clears this pin to accomplish the transmission.
;**************************************************************************
                bank    serial
                clrb    tx_divide.baud_bit      ;clear xmit timing count flag
                inc     tx_divide               ;only execute the transmit routine
                STZ                             ;set zero flag for test
                SNB     tx_divide.baud_bit      ; every 2^baud_bit interrupt
                test    tx_count                ;are we sending?
                snz
                retp                            ;if not, go to :receive
                clc                             ;yes, ready stop bit
                rr      tx_high                 ; and shift to next bit
                rr      tx_low                  ;
                dec     tx_count                ;decrement bit counter
                movb    tx_pin,/tx_low.6        ;output next bit
                retp
;**************************************************************************
receive
; This is an asynchronous receiver for RS-232 reception
; INPUTS:
;       rx_pin             -    Pin which RS-232 is received on.
; OUTPUTS:
;       rx_byte            -    The byte received
;       rx_flag            -    Set when a byte is received.
;**************************************************************************
                bank    serial
                movb    c,rx_pin                ;get current rx bit
                test    rx_count                ;currently receiving byte?
                jnz     :rxbit                  ;if so, jump ahead
                mov     w,#9                    ;in case start, ready 9 bits
                sc                              ;skip ahead if not start bit
                mov     rx_count,w              ;it is, so renew bit count
                mov     rx_divide,#start_delay  ;ready 1.5 bit periods
:rxbit          djnz    rx_divide,:rxdone       ;middle of next bit?
                setb    rx_divide.baud_bit      ;yes, ready 1 bit period
                dec     rx_count                ;last bit?
                sz                              ;if not
```

```
		rr	rx_byte			;  then save bit
		snz				;if so
		setb	rx_flag			;  then set flag
:rxdone
		retp
;**************************************************************************
do_timers
; The 24-bit timer increments every 52.16us when called by task_manager.
;**************************************************************************
		bank	timers			; Update the timers
		inc	timer_l
		snz
		inc	timer_h
		snz
		setb	timer_flag
		snz
		inc	timer_hh
		snz
		dec	timer_hh
		setb	led_pin
		sb	timer_h.2
		clrb	led_pin
		retp
;**************************************************************************
transmit_fsk
;**************************************************************************
		bank	fsk_transmit_bank
		sb	fsk_tx_en
		retp
		jb	fsk_answering,transmit_answer_tones
transmit_originate_tones
		jnb	fsk_tx_bit,:low_bit
:high_bit
		bank	sine_gen_bank
		mov	freq_count_high2,#f390_h
		mov	freq_count_low2,#f390_l
		retp
:low_bit
		bank	sine_gen_bank
		mov	freq_count_high2,#f450_h
		mov	freq_count_low2,#f450_l
		retp
transmit_answer_tones
		jnb	fsk_tx_bit,:low_bit
:high_bit
		bank	sine_gen_bank
		mov	freq_count_high2,#f1300_h
		mov	freq_count_low2,#f1300_l
		retp
:low_bit
		bank	sine_gen_bank
		mov	freq_count_high2,#f2100_h
		mov	freq_count_low2,#f2100_l
		retp
;******************************************************************************
; Interrupt
;
; With a retiw value of -163 and an oscillator frequency of 50MHz, this
; code runs every 3.26us.
;******************************************************************************
ISR
;******************************************************************************
FSK_output
		jnb	dtmf_gen_en,:dtmf_disabled
		call	@sine_generator1
		call	@DTMF_twist
		jmp	:task_switcher
:dtmf_disabled
		jnb	sine_gen_en,:task_switcher  ; Output the frequencies set by the freq_count registers
		call	@sine_generator2
		call	@SINE_out		; Output each discrete value of the sine table
		call	fsk_receive_main
:task_switcher
		call	task_manager
;******************************************************************************
ISR_DONE
; This is the end of the interrupt service routine.  Now load -163 into w and
; perform a retiw to interrupt 163 cycles from the start of this one.
; (3.26us@50MHz)
;******************************************************************************
		mov	w,#-163		;1	; interrupt 163 cycles after this interrupt
		retiw			;3	; return from the interrupt
;******************************************************************************
; End of the Interrupt Service Routine
;******************************************************************************

;**************************************************************************
org	$600			; These subroutines are on page 3.
;**************************************************************************
; DTMF transmit functions/subroutines
;**************************************************************************
;**************************************************************************
DTMF_TABLE	; DTMF tone constants
; This routine returns with the constant used for each of the frequency
; detectors.
; INPUT:	w	-	Index into the table (0-15 value)
; OUTPUT:	w	-	Constant at that index
;**************************************************************************
```

```
                clc
                jmp     PC+w
                retw    f697_l
                retw    f697_h
                retw    f770_l
                retw    f770_h
                retw    f852_l
                retw    f852_h
                retw    f941_l
                retw    f941_h
                retw    f1209_l
                retw    f1209_h
                retw    f1336_l
                retw    f1336_h
                retw    f1477_l
                retw    f1477_h
                retw    f1633_l
                retw    f1633_h
;**************************************************************************
ASCII_TABLE     ; Ascii value at index (0-15)
; INPUT:        w       -       Index into the table (0-15 value)
; OUTPUT:       w       -       Constant at that index
;**************************************************************************
                clc
                jmp     PC+w
                retw    '1'
                retw    '2'
                retw    '3'
                retw    'A'
                retw    '4'
                retw    '5'
                retw    '6'
                retw    'B'
                retw    '7'
                retw    '8'
                retw    '9'
                retw    'C'
                retw    '*'
                retw    '0'
                retw    '#'
                retw    'D'
;**************************************************************************
index_2_digit
; This subroutine converts a digit from 0-9 or a '*' or a '#' to a table
; lookup index which can be used by the load_frequencies subroutine.  To use
; this routine, pass it a value in the 'byte' register.  No invalid digits
; are used. (A, B, C, or D)
;**************************************************************************
                call    ASCII_TABLE
                retp
;**************************************************************************
digit_2_index
; This subroutine converts a digit from 0-9 or a '*' or a '#' to a table
; lookup index which can be used by the load_frequencies subroutine.  To use
; this routine, pass it a value in the 'byte' register.  No invalid digits
; are used. (A, B, C, or D)
;**************************************************************************
                bank    serial
                clr     temp
:loop
                mov     w,temp
                call    ASCII_TABLE
                xor     w,byte
                jz      :done
                inc     temp
                jb      temp.4,:done
                jmp     :loop

:done           mov     w,temp
                retp

;**************************************************************************
load_frequencies
; This subroutine loads the frequencies using a table lookup approach.
; The index into the table is passed in the byte register.  The DTMF table
; must be in the range of $400 to $500.
;**************************************************************************
                mov     temp,w
                bank    sine_gen_bank

                mov     w,>>temp
                and     w,#%00000110
                call    DTMF_TABLE
                mov     freq_count_low,w

                mov     w,>>temp
                and     w,#%00000110
                inc     wreg
                call    DTMF_TABLE
                mov     freq_count_high,w

                rl      temp
                setb    temp.3
                mov     w,temp
                and     w,#%00001110
                mov     temp,w
                call    DTMF_TABLE
                mov     freq_count_low2,w
```

```
                mov     w,temp
                inc     wreg
                call    DTMF_TABLE
                mov     freq_count_high2,w
                retp

;*****************************************************************************
dial_it         ; This subroutine puts out whatever frequencies were loaded
                ; for 100ms, and then stops outputting the frequencies.
;*****************************************************************************
                bank    sine_gen_bank
                clr     sine_index
                clr     sine_index2
                enable_o                        ; enable the output
                mov     w,#1
                call    @delay_100n_ms          ; delay 100ms
                setb    dtmf_gen_en
                mov     w,#1
                call    @delay_100n_ms          ; delay 100ms
                clrb    dtmf_gen_en
                call    @disable_o              ; now disable the outputs
:end_dial_it    retp
;*****************************************************************************
sine_generator1                                         ;(Part of interrupt service routine)
; This routine generates a sine wave with values from the sine table
; at the end of this program.  Frequency is specified by the counter.  To set
; the frequency, put this value into the 16-bit freq_count register:
; freq_count = FREQUENCY * 6.83671552 (@50MHz)
;*****************************************************************************
                bank    sine_gen_bank

                clc
                add     freq_acc_low,freq_count_low
                add     freq_acc_high,freq_count_high
                sc
                jmp     :no_change
                inc     sine_index
                mov     w,sine_index
                and     w,#$1f
                call    sine_table
                mov     curr_sine,w             ;1

:no_change

;*****************************************************************************
sine_generator2                                         ;(Part of interrupt service routine)
; This routine generates a sine wave with values from the sine table
; at the end of this program.  Frequency is specified by the counter.  To set
; the frequency, put this value into the 16-bit freq_count register:
; freq_count = FREQUENCY * 6.83671552 (@50MHz)
;*****************************************************************************
                bank    sine_gen_bank
                clc
                add     freq_acc_low2,freq_count_low2
                add     freq_acc_high2,freq_count_high2
                sc
                retp
                inc     sine_index2
                mov     w,sine_index2
                and     w,#$1f
                call    sine_table
                mov     curr_sine2,w
:no_change
                retp
;*****************************************************************************
SINE_out
; This subroutine moves the FSK output to the PPM register
;*****************************************************************************
                bank    sine_gen_bank
                clc
                mov     w,#127
                add     w,curr_sine2
                mov     PPM0_out,w
                retp
;*****************************************************************************
DTMF_twist
; This subroutine adds twist to the high frequency of the DTMF output.
;*****************************************************************************
                bank    sine_gen_bank
                mov     PPM0_out,curr_sine2                     ; mov sin2 into PPM0
                rr      wreg
                rr      wreg
                and     w,#$3f
                snb     wreg.5
                or      w,#$C0
                clc
                add     PPM0_out,w                              ; (1.25)(sin2) = sin2 + (0.25)(sin2)
                clc
                add     PPM0_out,curr_sine                      ; add the value of SIN into the PPM output
                clc
                add     PPM0_out,#128                   ; for result = PPM0 = 1.25*sin2 + 1*sin
                retp                                    ; return with page bits intact
;*********************************************************************************
sine_table
; The values in this table can be changed to increase/decrease the amplitude of
; the output sine wave.
; This sine table gives an output level of approximately -15dB into a 600 ohm
```

```
; impedance
;******************************************************************************
        clc
        jmp     pc+w
        retw    0
        retw    4
        retw    8
        retw    11
        retw    14
        retw    16
        retw    18
        retw    19
        retw    20
        retw    19
        retw    18
        retw    16
        retw    14
        retw    11
        retw    8
        retw    4
        retw    0
        retw    -4
        retw    -8
        retw    -11
        retw    -14
        retw    -16
        retw    -18
        retw    -19
        retw    -20
        retw    -19
        retw    -18
        retw    -16
        retw    -14
        retw    -11
        retw    -8
        retw    -4
```