TEXAS INSTRUMENTS

FSK Modulation and Demodulation With the MSP430 Microcontroller

Application Report

December 1998

Mixed-Signal Products

SLAA037

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

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

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

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

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Contents

List of Figures

List of Tables

FSK Modulation and Demodulation With the MSP430 Microcontroller

ABSTRACT

This application report describes a software program for performing V.23 FSK modem transceiver functions using an MSP430 microcontroller. It makes use of novel filter architecture to perform DSP functions on a processor with only shift and add capabilities.

1 Introduction

Many measurement applications (for example, electric and gas meters) require a way to communicate electronically with a central office so that measured data can be reported back to the central office and new tariffs can be set in the remote site. Telephony provides a convenient means of data communication.

Frequency shift keying (FSK) and dual tone multi frequency (DTMF) are two popular methods of representing binary data over telephone circuits. This application report describes a V.23-compliant FSK transceiver software module.

Integrating the measurement and communication functions onto the same chip yields cost as well as power-saving benefits. Using the MSP430, a high MIPs ultra low power microprocessor, allows power to be drawn from the telephone line in some cases.

This report describes the mathematical formulas for FSK signal transmission and detection. A list of the software modules is included with a reference schematic for telephone interface and low cost A/D converter. The schematic is only a reference, since the precise implementation can vary from country to country.

2 Demodulation Theory

A quadrature demodulator provides the FSK demodulation. In this type of demodulation, the signal and its delayed version are multiplied together and then low-pass filtered. If the delay, T, is set such that Wcarrier $\times$ T = $\pi/2$, then the low-pass filter result is proportional to the frequency deviation from the carrier and therefore represents the bit value sent.

$$If\ w = Wcarrier \pm Wdelta\ and\ T \times Wcarrier = \pi/2$$

where $w = 2\pi \times f$:

$$cos[wt].cos[w(t–T)] = coswT + cos(2wt–wT) \rightarrow Low\ Pass\ Filter$$
$$\rightarrow coswT = sin[\pm\ Wdelta] = \pm\ sin[Wdelta]$$

2.1 Choosing the Sampling Rate

The sampling is chosen to be Fcarrier $\times$ 4 for the purpose of obtaining the delayed sample without computational overhead. For V.23, the F carrier frequency is 1700 Hz and therefore the sampling rate becomes 6800 Hz. Using a 32768-Hz crystal yields 6793.3 Hz, which is 0.1% out. The sampling frequency is set by the 8-bit interval timer. Because this timer is limited to 256 counts, the interrupt rated is set to twice the sampling rate and the processing is divided into two halves with signal sampling performed every other interrupt.

2.2 Front End Processing

Most A/D converters, including the successive approximation A/D converter in the MSP430C325, need a dc bias; this yields an unsigned integer sample with an offset. Before this sample can be processed further, it needs to go through an unbias filter to take out the dc bias and turn the sample into a signed integer value. This unbias filtering also gives 30 dB or so of rejection for main frequencies.

2.3 FSK Demodulation

The signed integer sample and its delayed version are multiplied together; in this application, an 8×8 signed multiplication loop is used.

The product, made up of two frequency elements, is low-pass filtered to remove the double frequency element. The remainder is a signed integer value representing the original bit value transmitted.

The low-pass filter uses the digital wave filtering technique. This technique gives stable characteristics with very good coefficient tolerance. All multiplication is done through shifts and adds with the number of shift/add operations minimized through rounding off the coefficients. Because the filter has good coefficient tolerance, this rounding off does not affect the filter performance. The Butterworth filter used here gives approximately 40-dB attenuation in the stop band with 1-dB pass and ripple.

2.4 Bit Synchronization

The bit values coming out from demodulation need to be determined and synchronized to produce the incoming data bit stream. This process is also known as bit slicing and clock recovery. Because the sampling rate at 6800 is not an integer multiple of the data rate (baud rate) at 1200, an additional step is needed to consolidate between the two rates. This is done through a count-down counter with a sequence of preload value (5,6,5). Every 17 samples, the sampling rate and the data baud rate are resynchronized. Bit synchronization or clock recovery is done by monitoring bit value transitions. Lead or lag information is then obtained and the count-down counter is adjusted accordingly. Because of the difference between the sampling clock and the data clock, the data bit is never sampled at the middle of the baud period; instead a –5% to 13% variation is introduced. However, this should not have any adverse effect on the accuracy of the system, as it has been verified experimentally.

3 Modulation Theory

FSK modulation involves alternating the value of a delta frequency from a carrier frequency according to the value of the bit to be represented. For V.23, a bit value of 0 = 400 Hz and a bit value of 1 = –400 Hz.

$$FSK\ signal = Amplitude \times cos[t] 2\pi \times (Fcarrier \pm Fdelta)]$$

The sinusoidal signal is generated through a lookup table which contains cosine values from 0 to 2π. A parameter called PHASER (16 bit) represents the current angle: 0=0 degree, 8000 hex = 180 degree 10000 hex = 360 degree. With each sample, this angle is advanced by another parameter DELTA (16 bit) which determines the frequency of the signal (larger DELTA value = higher frequency). Frequency modulation is realized by changing the DELTA value according to the bit value to be transmitted at each baud period, according to the following formula:

$$DELTA = Fdesired/Fsampling \times 65536.$$

The advantage of this method over a digital oscillator method is that this method preserves the phase relationship even when the frequency is shifted from sample to sample.

3.1 Choosing the Sampling Rate

The 8-bit interval timer sets the sampling rate to 19200 samples/s. This rate is subdividable into the data baud rate of 1200. Also, it is sufficiently high to make the D/A process simpler.

3.2 Constructing the Look Up Table

To save ROM space, only the first quadrant (0 to 127 degrees) in Q7 format is coded. This is done by dividing the first quadrant (90 degrees) into 128 steps of approximately 0.7 degrees each. The remaining three quadrants can be worked out from this first quadrant table using additional computation.

3.3 FSK Modulation

The parameter PHASER is advanced by the amount DELTA at every interrupt. The first 9 bits of the PHASER is used to look up the cosine value. For the cosine function, the third and fourth quadrant are the same as the second and first quadrant, and so only the absolute value of the first 9 bits of PHASER is used.

Next, all second quadrant values are derived from the first quadrant ROM table.

The 8-bit result value is stored onto P0.OUT.

Every 16 interrupts, the parameter DELTA is updated with the next frequency by looking at the next bit to be transmitted.

4 Data Conversion

This section describes the required digital-to-analog (D/A) and analog-to-digital (A/D) data conversions.

4.1 A/D Conversion

The most straightforward way to digitize the incoming FSK signal is to use the 12-bit mode of the internal 14-bit A/D converter of the MSP430C325. However, not all of the 12 bits are needed to achieve good dynamic range for the FSK demodulation. Simulation results indicate that an 8-bit A/D stage gives good dynamic range up to 25 dB using internal AGC software. With an additional external AGC stage, the dynamic range can be further widened. As economical means of building 8-bit single slope A/D exists, this extends the application of this module to the rest of the MSP430 family. The application software included here uses a single slope A/D (universal timer with external comparator) for the demodulator. This makes the software universally applicable for the whole family.

4.2 D/A Conversion

A 6-bit external R–2R ladder is used to construct the D/A converter. Because the carrier frequency of 19200 Hz is nine times the highest frequency of the FSK of 2100 Hz, the post filtering stage should be relatively simple. In the application circuit, a single capacitor forms a single pole low pass filter but more poles can be realized using additional passive networks.

5 Power Consumption

The FSK concept is designed with low power in mind. The FSK demodulator takes less than 2 MIPs. With a low power op-amp as a front-end, total power consumption of less that 1.5 mA should be achievable. Thus, it is possible that the power can be derived entirely from the telephone line. A schematic is included for a suggested telephone line interface. The precise configuration may vary from country to country.

6 Exercising the Software

This section describes operation of the software.

6.1 FSK Receiver

The FSK signal is derived from the telecom interface circuit. This signal should have a dc bias of 1.2 V and a peak-to-peak level of 400 mV. The software decodes this FSK signal and produces three outputs which lets the user monitor the demodulated data.

- TP.3. This is the clock signal recovered from the input FSK.
- TP.5. This is the data recovered from the input FSK; data is latched out every rising edge of TP.3.
- P0.2–P0.7. These six bits output the low pass filtered result. With an external R–2R ladder this becomes very useful in monitoring the analogue FSK demodulator output level. It is hard limited to 8 bits with the MSB 6 bits loaded to port P0

6.2 FSK Transmitter

The transmitter software outputs an FSK signal according to the BIT MAP data defined in TX_DATA_TABLE. The bitmap pattern starts with a preamble followed by a long MARK period. Then the actual data is transmitted. This table uses a zero word as an end marker, and the software restarts the whole data sequence upon reaching a zero value in the bit map data.

7 Example Circuits

This section shows and describes example circuits.

7.1 Using the MSP430C325 as Main Processor

Figure 1 shows an example circuit using the MSP430C325 as the main processor. The circuit is tested with 400 mV peak-to-peak FSK input. To obtain the same results, Rx needs to be biased at 1.2 V with a 400 mV peak-to-peak FSK signal superimposed.

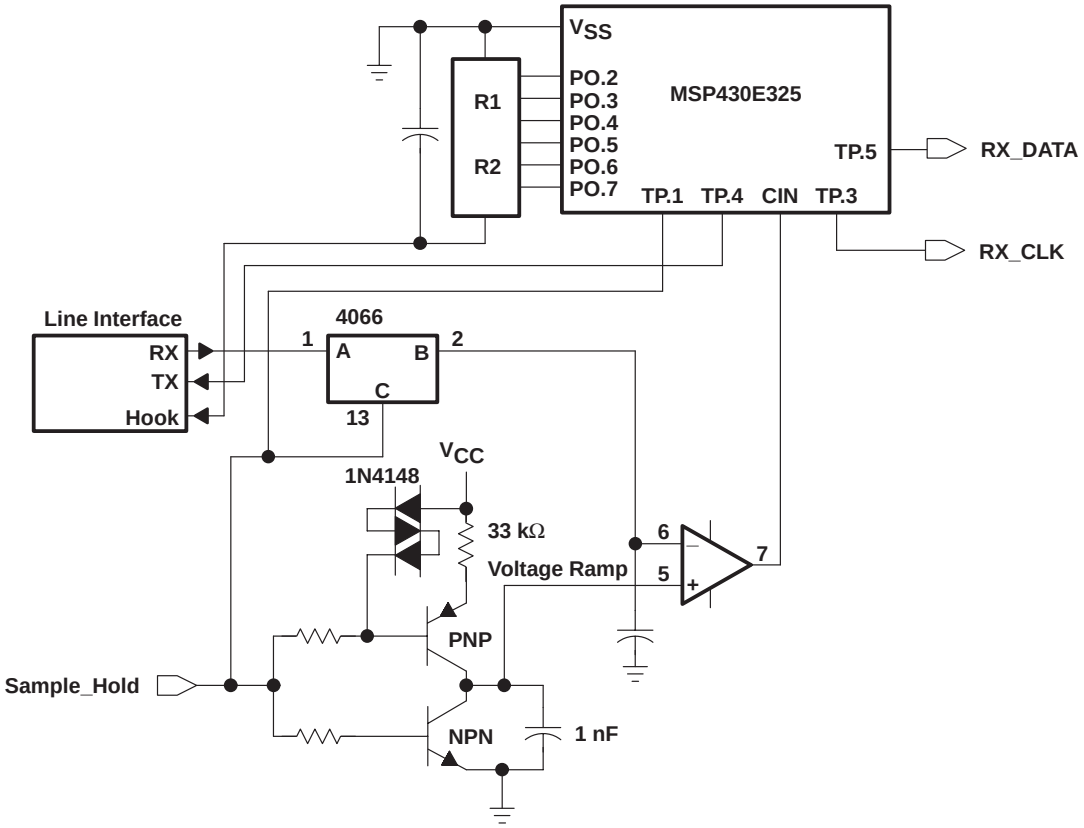
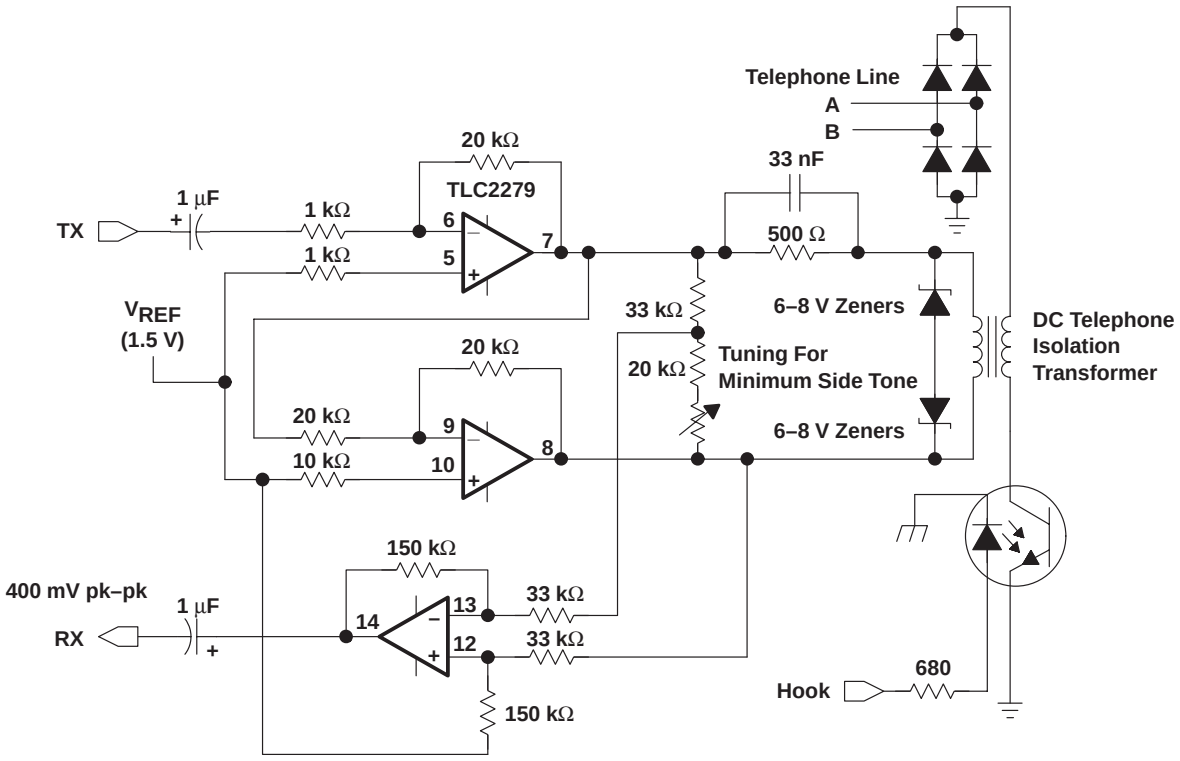


Figure 1. Main Processor and A/D Converter

7.2 Example Telephone Interface

Figure 2 shows an example telephone interface, and Table 1 lists FSK transceiver performance data.



This is a reference circuit only and may not be applicable under some circumstances.

Figure 2. Telephone Interface

Table 1. FSK Transceiver Performance

	RAM (BYTES)	ROM (BYTES)	MIPS (APPROX.)
FSK Receiver	18	512	2
FSK Transmitter	12	400	1.4

8 Summary

FSK transceivers are normally realized by either analog means or by the use of DSPs with hardware MAC units. Using an MSP430 RISC processor without a hardware MAC to achieve the transceiver function is a very unusual approach. The ability to create filters using digital wave filtering techniques, together with the orthogonal instruction set and the 16 bit architecture of the MSP430, makes the code very ROM and MIPs efficient. Moreover, the ultra low power capability of the MSP430 means that power can readily be derived from the phone line. This leads to component-efficient designs. The author has conducted other tests to conclude that, with some enhancements, the FSK receiver can work with an 8-bit A/D converter with enough sensitivity. Therefore the FSK transceiver can be implemented economically across the whole MSP430 family.

9 References

1. Texas Instruments: MSP430 Family, Architecture User’s Guide and Module Library.
2. Texas Instruments Digital Signal Processing Application with the TMS320 Family Volume 2.
3. Gaszi, L: *Explicit Formulas for Lattice Wave Digital Filters*; IEEE Trans. On Circuits and Systems VOL. CAS-32, NO. 1, January 1985

Appendix A FSK Receiver Routine

```
CPUOFF                  .equ   010h
SCG0                    .equ   040h
SCG1                    .equ   080h
IE1                     .equ   0h
IE1_P0IE1               .equ   08h
IE1_P0IE0               .equ   04h
IE1_OFIE                .equ   02h
IE1_WDTIE               .equ   01h
IE2                     .equ   01h
IE2_BTIE                .equ   80h
IE2_TPIE                .equ   08h
IE2_ADIE                .equ   04h
IE2_UTXRIE              .equ   02h
IE2_URXIE               .equ   01h
IFG1                    .equ   02h
IFG1_2                  .set   04h
IFG2                    .equ   03h
ME1                     .equ   04h
ME2                     .equ   05h
P0IN                    .equ   010h
P0OUT                   .equ   011h
P0DIR                   .equ   012h
P0FLG                   .equ   013h
P0IES                   .equ   014h
P0IE                    .equ   015h
LCDCTL                  .equ   030h
LCDM                    .equ   030h   ; LCD control & mode register address
BTCTL                   .equ   040h
BTCTL_SSEL              .equ   80h
BTCTL_Hold              .equ   40h
BTCTL_DIV               .equ   20h
BTCTL_FREQ1             .equ   10h
BTCTL_FREQ0             .equ   08h
BTCTL_IP2               .equ   04h
BTCTL_IP1               .equ   02h
BTCTL_IP0               .equ   01h
BTCNT1                  .equ   046h
BTCNT2                  .equ   047h
BTIFG                   .equ   080h   ; BT intrpt flag
TCCTL                   .equ   042h   ; Address of Timer/Counter control register
TCCTL_SSEL1             .equ   080h
TCCTL_SSEL0             .equ   040h
TCCTL_ISCTL             .equ   020h
TCCTL_TXE               .equ   010h
TCCTL_ENCNT             .equ   008h
TCCTL_RXACT             .equ   004h
TCCTL_TXD               .equ   002h
TCCTL_RXD               .equ   001h
TCPLD                   .equ   043h   ; Address of Timer/Counter pre–load register
TCDAT                   .equ   044h   ; Address of Timer/Counter
TPD                     .equ   04eh
TPD_B16                 .equ   080h
TPD_CPON                .equ   040h
TPE                     .equ   04fh
TPE_0                   .equ   01h
TPE_1                   .equ   02h
TPE_2                   .equ   04h
TPE_3                   .equ   08h
```

```
TPE_4                   .equ   10h
TPE_5                   .equ   20h
TPE_TPSSEL2             .equ   40h
TPE_TPSSEL3             .equ   80h
TPCTL                   .equ   04Bh
TPCTL_EN1FG             .equ   01h
TPCTL_RC1FG             .equ   02h
TPCTL_RC2FG             .equ   04h
TPCTL_EN1               .equ   08h
TPCTL_ENA               .equ   10h
TPCTL_ENB               .equ   20h
TPCTL_TPSSEL0           .equ   40h
TPCTL_TPSSEL1           .equ   80h
TPCNT1                  .equ   04Ch
TPCNT2                  .equ   04Dh
SCFI0                   .equ   050h
SCFI1                   .equ   051h
SCFQCTL                 .equ   052h
CBCTL                   .equ   053h
CBE                     .set   1
AIN                     .equ   0110h
AEN                     .equ   0112h
ACTL                    .equ   0114h
ACTL_CSTART             .equ   0001h
ACTL_SVCC_OFF           .equ   0000h
ACTL_SVCC_ON            .equ   0002h
ACTL_2                  .equ   0004h
ACTL_3                  .equ   0008h
ACTL_4                  .equ   0010h
ACTL_5                  .equ   0020h
ACTL_SEL_A0             .equ   0000h
ACTL_SEL_A1             .equ   0004h
ACTL_SEL_A2             .equ   0008h
ACTL_I_SRC_A0           .equ   0000h
ACTL_I_SRC_A1           .equ   0040h
ACTL_RNG_B              .equ   0200h
ACTL_RNG_AUTO           .equ   0800h
ACTL_POWER_UP           .equ   0000h
ACTL_POWER_DOWN         .equ   1000h
ACTL_CLK_MCLK           .equ   0000h
ACTL_CLK_MCLK_2         .equ   2000h
ACTL_CLK_MCLK_3         .equ   4000h
ADAT                    .equ   0118h
ADIFG                   .equ   04h
WDTCTL                  .equ   0120h
WDTHold                 .equ   80h
WDT_wrkey               .equ   05A00h
STACK                   .set   300h   ;280h start of system stack

*********************************************
*      Filters
*********************************************
WDF_PARMS               .usect "FILTMEM",10,0200h
IN_Z                    .set   0
Z1_1                    .set   2
Z1_2                    .set   4
Z3_1                    .set   6
Z3_2                    .set   8
end_of_parms            .usect "FILTMEM",2
data_word               .usect "FILTMEM",2
last_sample             .usect "FILTMEM",2
```

```
bit_lead_lag              .usect ”FILTMEM”,2
cycle_counter             .usect ”FILTMEM”,1   ;user service routine

;***************************************************************
;      These are used during the 8 bit timer interrupt
;       for FSK demodulation
;***************************************************************
;
************************
* DYNAMIC: The Registers Marked (used by WDF) must not be used moved.
************************
currenty                  .set   R6            ;used by WDF
currentx                  .set   R7            ;used by WDF
IROP1                     .set   currenty      ;used by WDF
IROP2L                    .set   R7            ;used by WDF
IRACL                     .set   R8            ;used by WDF
IRBT                      .set   R9            ;used by WDF
lastx                     .set   R10
bit_data                  .set   R11           ;used by WDF
mem_ptr                   .set   R15           ;used by WDF
************************
* STATIC
************************
bit_sync_timer            .set   R12           ;
global_status             .set   R13           ;
bits_count                .set   R14           ;
INTERRUPT_TOGGLE          .set   1
FALLING                   .set   2
CLOCK                     .set   4

;***********************************************
;System Init
;***********************************************
;
;RAM_NORMAL_DEMOD          .sect ”RAM_CODE”,02f0h
Start                     .sect ”ADC”,0f000h   ;0214h ;0F000h
       mov    #STACK,SP                        ;initialize system stack pointer
       mov    #(WDTHold+WDT_wrkey),&WDTCTL     ; Stop Watchdog Timer
       clr.b  &IFG1                            ;clear all interrupt flags
       clr.b  &IFG2
       mov.b  #(17*5)-1,&SCFQCTL               ;MCLK=32768*17*5 gives 2.8 MIPs
       mov.b  #4,&SCFI0                        ;Set RC oscillator to 2xFreq range
       mov.b  #-205,&TCPLD                     ;328/2 ;reset is lo/hi edge
       bis.b  #TPD_B16,&TPD                    ;op-amp
       bis.b  #11111110b,&P0DIR                ;set P0.7-P0.1 to output
                                               ;P0.0 used for RING
       mov.b  #(TCCTL_SSEL1+TCCTL_ISCTL+TCCTL_ENCNT),&TCCTL
       bic.b  #IE2_BTIE,&IE2                   ;disable Basic Timer Interrupt
       bic.b  #IE1_P0IE0,&IE1                  ;disable P0.0 interrupt
       bic.b  #IE2_TPIE,&IE2                   ;disable Universal timer Interrupt
       bis.b  #TPE_0+TPE_1+TPE_2+TPE_3+TPE_5,&TPE
                        ;TP_0 is used for Hook
                        ;TP_1 is used for RAMP generator control
                        ;TP_2 is used for test circuit
                        ;TP_3 is the recovered clock bit
                        ;TP_5 is the data bit
       mov    #1,bit_sync_timer
       mov.b  #3,cycle_counter
       call   #clear_all_parameters
                    ;<------------ enable FSK DEMO !!!!
       bis.b  #IE1_P0IE1,&IE1
       eint
```

```
WAIT_LOOP:                      ;wait in loop indefinitely,
                                ;Let Interrupt do the job
       jmp    WAIT_LOOP
TIM8_Int:
;*****************************************
;      Do input sampling
;*****************************************
NORMAL_DEMOD:
       bit    #INTERRUPT_TOGGLE,global_status         ;!!!!!
       jz     filters
do_ADC
       bic    #INTERRUPT_TOGGLE,global_status
SLP_A_D:
       mov.b  #(TPCTL_TPSSEL1+TPCTL_TPSSEL0),&TPCTL ;use MCLK as input,stop counting
       bis.b  #TPE_2+TPE_1,&TPD                ;reset RAMP
                        ;TPE_2 is used for test circuit only can be taken out.
       mov.b  &TPCNT1,currenty                 ;MSByte is zeroed
       mov.b  &TPCNT2,currentx                 ;MSByte is zeroed
       swpb   currentx
       bis    currenty,currentx                ;combined to form 16 bit results
                                               ;echo sample to output
       clr.b  &TPCNT1
       clr.b  &TPCNT2
                                               ;enable counter
       mov.b  #TPCTL_ENA+TPCTL_ENB+TPCTL_TPSSEL1+TPCTL_TPSSEL0,&TPCTL
       bic.b  #TPE_2+TPE_1,&TPD                ;re-start RAMP
_0DB:
       rla    currentx
       rla    currentx
       rla    currentx
;*************************************************
;      Make sure input voltage range < VCC/4
;*************************************************
*******************************************************************************
*      Anti-bias results
*                answer = input + (0.875 * lasty) - lastx;
*                lasty  = answer;
*                lastx  = input;
*******************************************************************************
       mov    currentx,currenty
       sub    lastx,currenty
       mov    currentx,lastx
;***************************************
; The gain readjustments here is optimal for 1V pk-pk
;    with a higher setting, may need to scale it down by
;       putting in the extra rra's
;***************************************
;      rra    currenty
       rra    currenty
       rra    currenty
       mov    &last_sample,IROP2L
       mov    currenty,&last_sample
       and    #0ffh,currenty
       and    #0ffh,IROP2L
```

```
************************************************
* signed 8 x 8 routine
************************************************
       clr    IRACL
       tst.b  currenty
       jge    L$101
       swpb   IROP2L
       sub    IROP2L,IRACL
       swpb   IROP2L
L$101
       tst.b  IROP2L
       jge    MACU8
       swpb   currenty
       sub    currenty,IRACL
       swpb   currenty
MACU8
       mov    #1,IRBT
L$002  bit    IRBT,currenty
       jz     L$01
       add    IROP2L,IRACL
L$01   rla    IROP2L
       rla.b  IRBT
       jnc    L$002
       mov    IRACL,R9                  ;prepare for LP filter in next cycle

************************************************
*      Do edge synchronisation
************************************************
       bit    #FALLING,global_status    ;!!!!!!!!!
       jz     detect_rising
detect_falling:
       tst    bit_data                  ;last edge was rising so test for
falling
       jge    update_bit_sync_timer     ;not falling, do next stage
       bic    #FALLING,global_status    ;falling has been detected, next cycle should
                                        ;look for rising
       jmp    new_edge_detected

detect_rising:
       tst    bit_data
       jn     update_bit_sync_timer
       bis    #FALLING,global_status

;*******************************************************************************
;  Edge change has been detected, now synchronise to bit synch timer
;  by determining, leading or lagging
;                    ____ Sample at 3
;                   /
;   Lead      Lag
;    __/|\__   /|\   ______
;  |   |   |   |   |
;  |   |   |   |   |
; _|   |   |___|___|
;  A new edge has been confirmed, now we need to synchronise the software's internal
;  data clock running at 1200 BAUD with the external data edge. The internal data
;  clock is represented by BIT2 of TPD and so it is used to determine leading or
;  lagging. Since there is fare amount of jigger in the incoming clock, we do not
;  use the lead or lag informative to update the internal straight away. Instead, we
;  try to 'low pass' the results by adding or taking it from a counter, only when the
;  total value is greater or smaller than a certain threshold that we take action.
;*******************************************************************************
```

```
new_edge_detected
       bit.b  #CLOCK,global_status      ;!!!!!!!!!
       jz     lagging
leading:
       add    #1,bit_lead_lag
       jmp    update_bit_sync_timer
lagging:
       sub    #1,bit_lead_lag

;*****************************************************************************
;  Do bit timing generation
;  6/6800+5/6800+6/6800=3/1200
;  This is the internal clock and it should run at an average of 1200 BAUD but it
;  does have jigger because you cannot generate a 1200 signal from a 6800 Hz signal
;  The bit_sync_timer is loaded with 6 and then 5 and then 6 cylcically
;*****************************************************************************

update_bit_sync_timer:
       sub    #1,bit_sync_timer
       jz     load_new_timer_value      ; if timer reaches zero, time to load
new value
                                        ; the clock edge is reset
       cmp    #3,bit_sync_timer         ; eye is more open at 4
       jnz    done_bit_sync
       bis.b  #TPE_3,&TPD
       bis.b  #CLOCK,global_status      ; the clock edge is set in the middle of the
                                        ; cycle
                                        ; should do sampling of data in here but not
       cmp    #20,bit_data
       jge    data_is_space             ; used in this program.
                                        ; C is set at this point
data_is_mark
       bic.b  #TPE_5,&TPD
       clrc
       jmp    count_down_bits
done_bit_sync:
       reti
data_is_space
       bis.b  #TPE_5,&TPD
       setc
count_down_bits
       rrc    data_word
       reti
load_new_timer_value:
       bic.b  #TPE_3,&TPD
       bic.b  #CLOCK,global_status      ;clear the internal synch clock bit.
       mov    #6,bit_sync_timer
       sub.b  #1,cycle_counter          ;determine whether next count should
                                        ;be 5 or 6
       jnz    do_6_counts
do_5_counts
       mov.b  #3,cycle_counter
       mov    #5,bit_sync_timer         ;if count is 5 see if we need to
                                        ;compensate for
       cmp    #-7,bit_lead_lag          ;leading
       jl     compensate_lag
       reti
do_6_counts:
       cmp    #7,bit_lead_lag           ;if count is 6 see if we need to
                                        ;compensate for
       jge    compensate_lead                ;lagging
```

```
       reti
compensate_lag
       add    #1,bit_sync_timer
       mov    #0,bit_lead_lag

       reti
compensate_lead
       sub    #1,bit_sync_timer
       mov    #0,bit_lead_lag
       reti
************************************************
*      Running this filter takes 113 cycles
************************************************

;*****************************************************
; New simpler filter at following specification
; Freq_Stop: 2.5KHz, Attenuation_Stop: 40dB
; Freq_Pass: 1.4KHz, Attenuation_Pass: 1dB
; Order of filter = 5
;*****************************************************
filters:
       bis    #INTERRUPT_TOGGLE,global_status
       mov    #WDF_PARMS,mem_ptr
       .word  4f16h
       .word  0000h
       .word  498fh
       .word  0000h
       .word  4f17h
       .word  0008h
       .word  8607h
       .word  4708h
       .word  1108h
       .word  4806h
       .word  1108h
       .word  1108h
       .word  1108h
       .word  1108h
       .word  1108h
       .word  8806h
       .word  8f16h
       .word  0008h
       .word  4f9fh
       .word  0006h
       .word  0008h
       .word  468fh
       .word  0006h
       .word  8706h
       .word  4f17h
       .word  0004h
       .word  8907h
       .word  4708h
       .word  1108h
       .word  1108h
       .word  1108h
       .word  4809h
       .word  1108h
       .word  1108h
       .word  1108h
       .word  8809h
       .word  5f19h
       .word  0004h
```

```
       .word  8907h
       .word  4f9fh
       .word  0002h
       .word  0004h
       .word  478fh
       .word  0002h
       .word  8906h
       mov    R6,bit_data
;*************************************************************
;      Low pass filter output stored in R6
;      R6 get turned into a analogue value after some hard limiting
;*************************************************************
       add    #80h,R6
       tst    R6
       jge    non_negative
       mov    #0,R6
non_negative
       cmp    #0ffh,R6
       jlo    non_ceiling
       mov    #0ffh,R6
non_ceiling
       mov.b  R6,&P0OUT
exit_D_A
       reti
;************************************************
;      clear all parameters excess comb filter
;************************************************
clear_all_parameters:
       mov    #0,r6
       mov    #0,r7
       mov    #0,r8
       mov    #0,r9
       mov    #0,r10
       mov    #0,r11
       mov    #0,r12
       mov    #0,r13
       mov    #0,r14
       mov    #0,r15
       mov    #WDF_PARMS,r4
clear_parms_loop
       mov    #0,0(r4)
       incd   r4
       cmp    #end_of_parms,r4
       jnz    clear_parms_loop
       ret
;*************************************************************
;**** Interrupt Vector Addresses:
       .sect ”Int_Vect”,0ffe0h    ;03e0h              ; 0FFE0h

       .word  Start               ;P0.27
       .word  Start               ;BT
       .word  Start               ;
       .word  Start               ;
       .word  Start               ;Universal Timer
       .word  Start               ;ADC
       .word  Start               ;
       .word  Start               ;
       .word  Start               ;
       .word  Start               ;
       .word  Start               ;WDT
```

```
.word  Start              ;
.word  TIM8_Int           ;P0.1
.word  Start              ;P0.0
.word  Start              ;RSTI/OF
.word  Start              ;PUC/WDT
```

Appendix B FSK Transmitter Routine

```
CPUOFF                  .equ   010h
SCG0                    .equ   040h
SCG1                    .equ   080h
IE1                     .equ   0h
IE1_P0IE1               .equ   08h
IE1_P0IE0               .equ   04h
IE1_OFIE                .equ   02h
IE1_WDTIE               .equ   01h
IE2                     .equ   01h
IE2_BTIE                .equ   80h
IE2_TPIE                .equ   08h
IE2_ADIE                .equ   04h
IE2_UTXRIE              .equ   02h
IE2_URXIE               .equ   01h
IFG1                    .equ   02h
IFG2                    .equ   03h
ME1                     .equ   04h
ME2                     .equ   05h
P0IN                    .equ   010h
P0OUT                   .equ   011h
P0DIR                   .equ   012h
P0FLG                   .equ   013h
P0IES                   .equ   014h
P0IE                    .equ   015h
LCDCTL                  .equ   030h
LCDM                    .equ   030h   ;LCD control & mode register address
BTCTL                   .equ   040h
BTCTL_SSEL              .equ   80h
BTCTL_Hold              .equ   40h
BTCTL_DIV               .equ   20h
BTCTL_FREQ1             .equ   10h
BTCTL_FREQ0             .equ   08h
BTCTL_IP2               .equ   04h
BTCTL_IP1               .equ   02h
BTCTL_IP0               .equ   01h
BTCNT1                  .equ   046h
BTCNT2                  .equ   047h
BTIFG                   .equ   080h   ; BT intrpt flag
TCCTL                   .equ   042h   ; Address of Timer/Counter control
                                      ; register
TCCTL_SSEL1             .equ   080h
TCCTL_SSEL0             .equ   040h
TCCTL_ISCTL             .equ   020h
TCCTL_TXE               .equ   010h
TCCTL_ENCNT             .equ   008h
TCCTL_RXACT             .equ   004h
TCCTL_TXD               .equ   002h
TCCTL_RXD               .equ   001h
TCPLD                   .equ   043h   ; Address of Timer/Counter preload
                                      ; register
TCDAT                   .equ   044h   ; Address of Timer/Counter
TPD                     .equ   04eh
TPD_B16                 .equ   080h
TPD_CPON                .equ   040h
TPE                     .equ   04fh
TPE_0                   .equ   01h
TPE_1                   .equ   02h
TPE_2                   .equ   04h
TPE_3                   .equ   08h
```

```
TPE_4                   .equ   10h
TPE_5                   .equ   20h
TPE_TPSSEL2             .equ   40h
TPE_TPSSEL3             .equ   80h
TPCTL                   .equ   04Bh
TPCTL_EN1FG             .equ   01h
TPCTL_RC1FG             .equ   02h
TPCTL_RC2FG             .equ   04h
TPCTL_EN1               .equ   08h
TPCTL_ENA               .equ   10h
TPCTL_ENB               .equ   20h
TPCTL_TPSSEL0           .equ   40h
TPCTL_TPSSEL1           .equ   80h
TPCNT1                  .equ   04Ch
TPCNT2                  .equ   04Dh
SCFI0                   .equ   050h
SCFI1                   .equ   051h
SCFQCTL                 .equ   052h
CBCTL                   .equ   053h
AIN                     .equ   0110h
AEN                     .equ   0112h
ACTL                    .equ   0114h
ACTL_CSTART             .equ   0001h
ACTL_SVCC_OFF           .equ   0000h
ACTL_SVCC_ON            .equ   0002h
ACTL_2                  .equ   0004h
ACTL_3                  .equ   0008h
ACTL_4                  .equ   0010h
ACTL_5                  .equ   0020h
ACTL_SEL_A0             .equ   0000h
ACTL_SEL_A1             .equ   0004h
ACTL_SEL_A2             .equ   0008h
ACTL_I_SRC_A0           .equ   0000h
ACTL_I_SRC_A1           .equ   0040h
ACTL_RNG_B              .equ   0200h
ACTL_RNG_AUTO           .equ   0800h
ACTL_POWER_UP           .equ   0000h
ACTL_CLK_MCLK           .equ   0000h
ACTL_CLK_MCLK_2         .equ   2000h
ACTL_CLK_MCLK_3         .equ   4000h
ADAT                    .equ   0118h
ADIFG                   .equ   04h
WDTCTL                  .equ   0120h
WDTHold                 .equ   80h
WDT_wrkey               .equ   05A00h
STACK                   .set   3d0h   ;280h start of system stack
*************************************************************************
*      constants:
*      delta_phase = (freq/sam_freq)*65536
*************************************************************************
_1300_Hz                .equ   01155h
_2100_Hz                .equ   01c00h
DELTA_PHASE             .equ   1
*************************************************************************
*      Filters
*************************************************************************
sinne_value                          .usect ”FILTMEM”,2,200h
tx_cycle_counter                     .usect ”FILTMEM”,2
tx_cycle_ptr                         .usect ”FILTMEM”,2
                                     ;user service routine
```

```
delta_phase                     .set   R6
phase_ptr                       .set   R7
tx_data_ptr                     .set   R8
tx_data_mask                    .set   R9
DELAY_COUNTER                   .set   R10
global_status                   .set   R11
reg_1                           .set   R12
reg_2                           .set   R13
reg_3                           .set   R14
TX_DONE                         .set   1
FALLING                         .set   2
CLOCK                           .set   4
HUNT                            .set   8
_20MS                           .set   136
_1PT5S                          .set   28800

;***********************************************
;System Init
;***********************************************
Start         .sect ”ADC”,0f000h                ;0214h ;0F000h
       mov    #STACK,SP                 ;initialize system stack pointer
       mov    #(WDTHold+WDT_wrkey),&WDTCTL  ; Stop Watchdog Timer
       clr.b  &IFG1                     ;clear all interrupt flags
       clr.b  &IFG2
       mov.b  #(75)–1,&SCFQCTL          ;MCLK=32768*17*4 gives 2.45 MIPs
       mov.b  #4,&SCFI0                 ;Set RC oscillator to 2xFreq range
       mov.b  #–128,&TCPLD              ;32768*75/128 = 19200 smps/s
       mov.b  #(TCCTL_SSEL1+TCCTL_ISCTL+TCCTL_ENCNT),&TCCTL
       mov.b  #IE1_P0IE1,&IE1           ;enable 8 bit Timer
       bis.b  #11111111b,&P0DIR         ;set P0.7–P0.0 to output
       eint
;**** Main Program:
Loop                                    ;wait for Interrupt to do its work
       mov    #TX_DATA_TABLE,tx_data_ptr
       mov    #08000h,tx_data_mask
       mov    #0,phase_ptr
       mov    #1,tx_cycle_counter
       bic    #TX_DONE,global_status
       call   #fetch_new_output_bit
       call   #fsk_modulation
wait_for_tx_done
       bit    #TX_DONE,global_status
       jz     wait_for_tx_done
Loop2:
       jmp    Loop2
TIM8_Int:
NORMAL_MOD:
       call   #fsk_modulation

;***************************************************************************
;       This part will output to P0OUT which should have an 8 bit
;       R–2R ladder attached to it. This is used for monitoring
;       the filtered value and should be taken off if we need to
;       use the port to do FSK TX function
;***************************************************************************
D_A
       mov    sinne_value,reg_1
       mov.b  reg_1,&P0OUT        ;MSB is in bit 7.
       reti
```

```
;***************************************************************************
;      Table look up
;***************************************************************************
fsk_modulation:
       add    delta_phase,phase_ptr

                           ;table has 128 elements, 4*128 = 512 =+/- 256
                           ;extract the top most 9 bits
       mov    phase_ptr,reg_1
                           ;cos table 3rd and 4th quad maps into 1st and
                           ;2nd quad
       tst    reg_1
       jge    no_tx_abs
       xor    #0ffffh,reg_1
       add    #1,reg_1
no_tx_abs:
       swpb   reg_1        ;LSB of result in MSB of reg_1, MSB 8bits in
                           ;LSByte of reg_1
       bit    #8000h,reg_1
       rlc    reg_1        ;C into bit0 MSB 8 bits in bit1-8
       bic    #0fe00h,reg_1
                           ;Q8 format
first_two_quadrant:
       cmp    #128,reg_1
       jn     first_quadrant
second_quadrant
       mov    #256,reg_2
       sub    reg_1,reg_2
       mov.b  cos_table(reg_2),reg_2
       xor    #0ffffh,reg_2
       add    #1,reg_2
       jmp    output_sample
first_quadrant             ;0-127 degrees
       mov.b  cos_table(reg_1),reg_2
output_sample:
       add    #80h,reg_2
       mov    reg_2,sinne_value          ;results in sinne_value

;***************************************************************************
;fetch_new_out_bit service routine, begin
;***************************************************************************
fetch_new_output_bit:
       dec    tx_cycle_counter
       jnz NO_RESET_TX_PTR
                                         ;@19200 smps/s div 16 = 1200 BAUD
       mov    #16,tx_cycle_counter
load_next_cycle:
       mov    #_2100_Hz,delta_phase      ;assume this first
       mov    tx_data_ptr,reg_1
       bit    @reg_1,tx_data_mask
       jnz    TX_BIT_IS_1
TX_BIT_IS_0:
       mov    #_1300_Hz,delta_phase
TX_BIT_IS_1:
       rra    tx_data_mask
       BIC    #8000h,tx_data_mask
       jnc    NO_TX_PTR_UPDATE
       mov    #8000h,tx_data_mask
       add    #2,tx_data_ptr
       mov    tx_data_ptr,reg_1
       tst    0(reg_1)
```

```
        jnz    NO_RESET_TX_PTR
        mov    #TX_DATA_TABLE,tx_data_ptr
        bis    #TX_DONE,global_status
NO_TX_PTR_UPDATE
NO_RESET_TX_PTR
        ret
;-------------------------------------------------------------------------
;  fetch_new_out_bit service routine, end
;-------------------------------------------------------------------------
TX_DATA_TABLE
        .word  05555h
        .word  05555h
        .word  05555h
        .word  05555h
        .word  05555h
        .word  05555h
        .word  0FFFFh
        .word  0FFFFh
        .word  0FFFFh
        .word  0FFFFh
        .word  0FC07h
;-------------------------------------------------------------------------
;  this has plenty of mark bits
;  contents: 5 bytes, 1 3 5 7 9
;-------------------------------------------------------------------------
        .word  0FC0Bh
        .word  0FC03h
        .word  0FC07h
        .word  0FC0Bh
        .word  0FC0Fh
        .word  0FC13h
        .word  0
cos_table:
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07fh
        .byte  07eh
        .byte  07eh
        .byte  07eh
        .byte  07eh
        .byte  07dh
        .byte  07dh
        .byte  07dh
        .byte  07ch
        .byte  07ch
        .byte  07ch
        .byte  07bh
        .byte  07bh
        .byte  07ah
        .byte  07ah
        .byte  07ah
        .byte  079h
```

```
        .byte  079h
        .byte  078h
        .byte  077h
        .byte  077h
        .byte  076h
        .byte  076h
        .byte  075h          ;cos 23.2031
        .byte  075h
        .byte  074h
        .byte  073h
        .byte  073h
        .byte  072h
        .byte  071h
        .byte  070h
        .byte  070h
        .byte  06fh
        .byte  06eh
        .byte  06dh
        .byte  06ch
        .byte  06ch
        .byte  06bh
        .byte  06ah
        .byte  069h
        .byte  068h
        .byte  067h
        .byte  066h
        .byte  065h
        .byte  064h
        .byte  063h
        .byte  062h
        .byte  061h
        .byte  060h
        .byte  05fh
        .byte  05eh
        .byte  05dh
        .byte  05ch
        .byte  05bh
        .byte  05ah
        .byte  059h
        .byte  058h
        .byte  057h
        .byte  055h
        .byte  054h
        .byte  053h
        .byte  052h
        .byte  051h
        .byte  04fh
        .byte  04eh
        .byte  04dh
        .byte  04ch
        .byte  04ah
        .byte  049h
        .byte  048h
        .byte  047h
        .byte  045h
        .byte  044h
        .byte  043h
        .byte  041h
        .byte  040h
        .byte  03fh
```

```
        .byte  03dh         ;cos 61.1719
        .byte  03ch
        .byte  03ah
        .byte  039h
        .byte  038h
        .byte  036h
        .byte  035h
        .byte  033h
        .byte  032h
        .byte  030h
        .byte  02fh
        .byte  02eh
        .byte  02ch
        .byte  02bh
        .byte  029h
        .byte  028h
        .byte  026h
        .byte  025h
        .byte  023h
        .byte  022h
        .byte  020h
        .byte  01fh
        .byte  01dh
        .byte  01ch
        .byte  01ah
        .byte  018h
        .byte  017h
        .byte  015h
        .byte  014h
        .byte  012h
        .byte  011h
        .byte  00fh
        .byte  00eh
        .byte  00ch
        .byte  00ah
        .byte  009h
        .byte  007h
        .byte  006h
        .byte  004h
        .byte  003h
        .byte  001h         ;cos 89.2969
;***************************************************************
;**** Interrupt Vector Addresses:
        .sect ”Int_Vect”,0ffe0h    ;03e0h       ; 0FFE0h
        .word  Start               ;P0.27
        .word  Start               ;BTIM_Int    ;BT
        .word  Start               ;
        .word  Start               ;
        .word  Start               ;UTIM_Int    ;Universal Timer
        .word  Start               ;ADC
        .word  Start               ;
        .word  Start               ;
        .word  Start               ;
        .word  Start               ;
        .word  Start               ;WDT
        .word  Start               ;
        .word  TIM8_Int            ;P0.1
        .word  Start               ;P0.0
        .word  Start               ;RSTI/OF
        .word  Start               ;PUC/WDT
```