Texas Instruments

Application Report
SPRA496

TMS320F240 DSP-Solution for High-Resolution Position with Sin/Cos-Encoders

Martin Staebler *Digital Signal Processoring Solutions*

Abstract

This application report offers a solution for obtaining high-resolution position with sin/cos-encoders using the Texas Instrument (TI™) TMS320F240 digital signal processor (DSP) controller.

This is achieved with a minimum of glue logic and software overhead, since the TMS320C24x DSP controller already incorporates an on-chip quadrature encoder pulse (QEP) circuit as well as two analog-to-digital converters (ADCs). The QEP circuit provides a glueless interface to TTL-encoders. The two analog-to-digital converters provide the simultaneous sampling of the two sinusoidal output signals of the sin/cos-encoders.

The software package includes all functions required for initialization and position interpolation with sin/cos-encoders. All subroutines are ANSI C-compatible and can be called from any C program. Position interpolation requires 15µs of CPU execution time. The resolution achieved is approximately 400 times better than with TTL-encoders of equivalent line count.

This gives the user the advantage of having either a higher position resolution or reduced sensor cost, since sin/cos-encoders require less line counts than equivalent TTL-encoders to achieve the same resolution.

Contents

Figures

Tables

Introduction

Incremental rotary encoders are used in many applications to measure angular position and/or speed. Depending on the application, **TTL-encoders** with TTL-output signals or **sin/cos-encoders** with analog sinusoidal output signals (which allow a higher resolution) may be used.

The TMS320F240 DSP controller is designed to meet a wide range of digital motor control applications and thus incorporates the peripherals to support TTL-encoders as well as sin/cos encoders.

The section, *Incremental Rotary Encoders*, describes the principle of sin/cos-encoders and the method to obtain high-resolution position.

The section, *Implementation on the TMS320F240*, explains the implementation on the TMS320F240 with respect to the on-chip QEP unit and the dual analog-to-digital converters. This includes a proposal for the hardware interface, a description of the software routines, which initialize and handle the sin/cos-encoder interface, and the usage of these functions in other application programs.

The section, *Results*, shows the memory requirements and the CPU loading and discusses the position accuracy achieved.

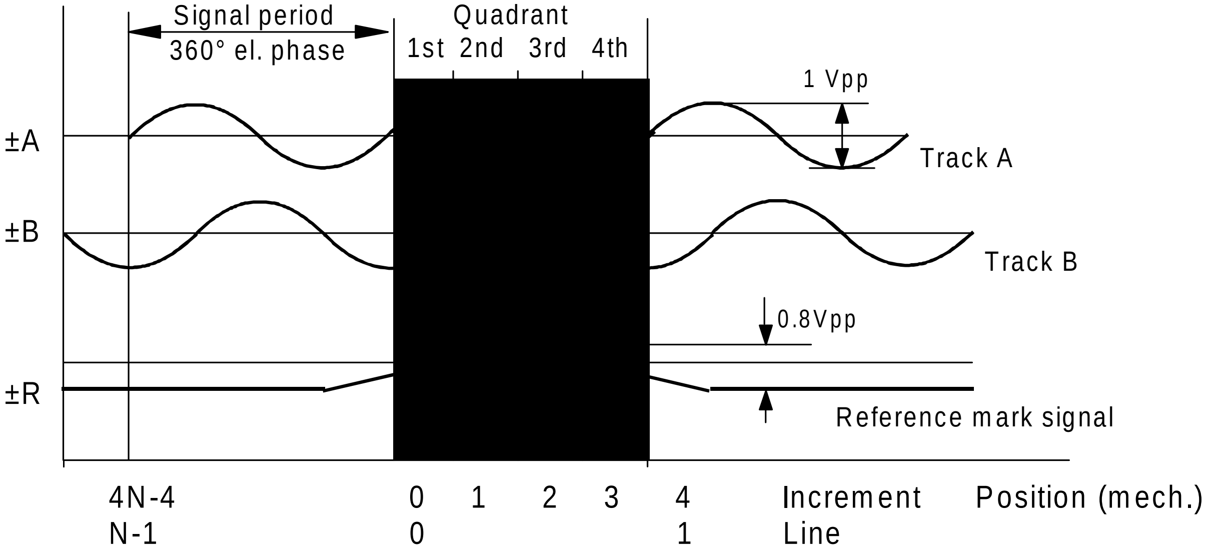
Incremental Rotary Encoders

Output Signals

Incremental rotary encoders operate on the principle of photo-electrically scanning the very fine gratings forming an incremental track. When rotating, the encoders modulate a beam of light whose intensity is sensed by photo-electrical cells, producing two 90° (el.) phase-shifted sinusoidal signals A and B. B lags A with clockwise rotation viewed from the shaft side of the encoder. The number of periods of A and B over one mechanical revolution equals the line count N of the encoder. The frequency is proportional to the revolution speed and line count of the encoder. A second track carries a reference mark that modulates the reference mark signal R at a maximum once per (mech.) revolution.

Sin/cos-encoders provide the differential analog output signals (±A, ±B, and ±R) as shown in Figure 1. **TTL-encoders** already incorporate a circuit that digitizes the sinusoidal scanning signals and provides two 90° phase-shifted square-wave pulse trains U_a, U_b and the reference pulse U_r.

Figure 1. Output Voltage Signals (±A, ±B, ±R) of sin/cos-Encoders with N Line Counts per Revolution as Function of the Mechanical Position



High-Resolution Position Determination

Rotation direction can be determined by detecting which one of the two quadrature encoded signals, A or B, is the leading sequence. Rotation speed can be determined by the frequency of the sinusoidal signals, A or B, with respect to the line count N of the encoder.

According to Figure 1, the angular position can be determined by knowing the incremental count or the line count and, when between two consecutive increments or lines, deriving the phase from the analog signals A and B.

The reference mark signal R provides absolute position determination, if the angle at which the encoder is mounted is known.

Incremental Position/Count

The incremental count and hence the incremental position can be determined by a timer that counts up when A is the leading sequence and counts down when B is the leading sequence. When digitized, both edges of A and B are counted, thus the incremental position Φ_{incr} is given by

$$\Phi_{incr} = \left(\frac{360^\circ}{4 \cdot N}\right) \cdot incr + \Phi_0 ,$$

where

[incr] is the timer count or incremental count

N is the line count of the encoder

Φ_0 is the zero position.

One incremental step is equivalent to a 90° (el.) phase shift of the signals, A and B.

Phase

The phase φ of the sinusoidal signals A and B can be used to interpolate the position between two consecutive line counts or four incremental steps, which are equivalent to each other. It can be calculated as

$$\varphi = \begin{cases} 90° + \arctan\left(\frac{B}{A}\right), & A \geq 0 \\ \\ 270° + \arctan\left(\frac{B}{A}\right), & A < 0 \end{cases} \qquad (2.1)$$

which has the advantage that the absolute amplitudes of A and B, which are a common function of the encoder's rotation speed and supply voltage, do not affect the result. Since the arctan-function is ambiguous, one has to check the sign of the sinusoidal signals A and B to identify the correct quadrant.

Interpolated High-Resolution Position

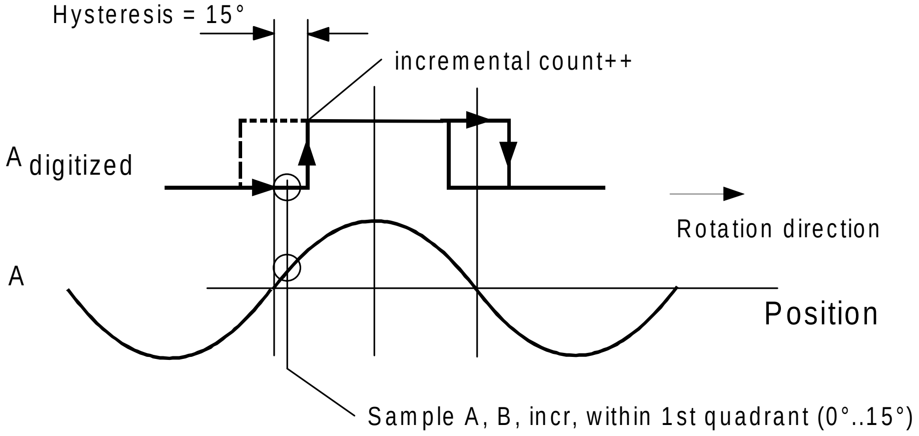
When the incremental count [incr] is matched to the phase φ according to incr = 0 $\Leftrightarrow$ $0° \leq \varphi < 90°$, incr = 1 $\Leftrightarrow$ $90° \leq \varphi < 180°$, etc., the **high-resolution position/angle Φ_a** can then be derived as:

$$\Phi_a(\text{incr}, \varphi) = \frac{360°}{N}\left((\text{incr} >> 2) + \frac{\varphi}{360°}\right) + \Phi_{a0}. \qquad (2.2)$$

Note that the sinusoidal signals A and B and the incremental count [incr] must be sampled **simultaneously.**

Practically, the digitized signals $A_{\text{digitized}}$, $B_{\text{digitized}}$, which edges are counted by the incremental counter, have a phase shift compared to the analog signals due to hysteresis and the propagation delay of the digitizing circuit. At the transition to the next quadrant, the incremental counter is not updated immediately because of the phase lag, e.g., as shown for the first quadrant in Figure 2.

Figure 2. Phase Shift of $A_{digitized}$ to A Due to Hysteresis



However, equation (2.2) can be applied as long as that phase shift is less than ±90° (el.), which is equivalent to a ±1 incremental count. Since only the phase information is used to identify the quadrant, there are only two exceptions (which may occur close to the transition to the next line) to consider when applying equation (2.2):

a) *$0° \leq \varphi < 90°$ AND incr % 4 = 3.*

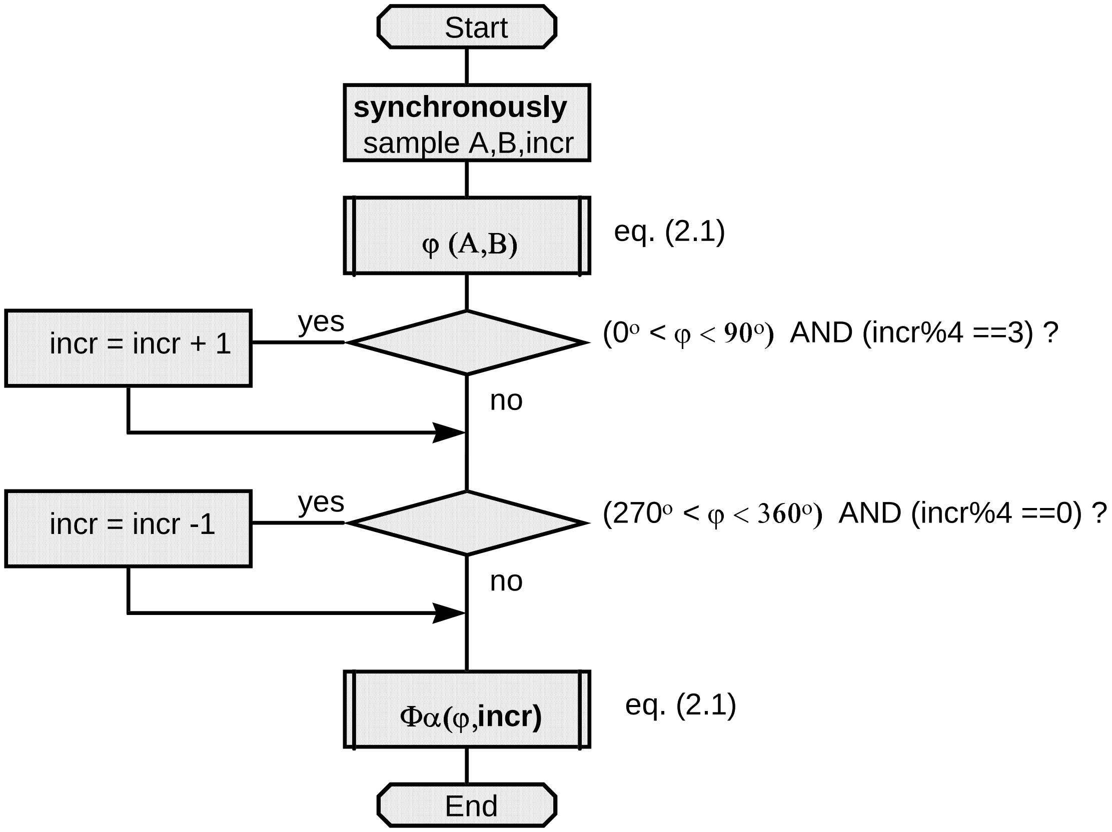
Here the phase φ was obviously sampled before the incremental count was updated due to hysteresis and/or propagation delay. [incr] points to the wrong line count. In that case the incremental count [incr] is increased by one to compensate for that (known) error.

b) *$270° \leq \varphi < 360°$ AND incr % 4 = 0.*

In that case, the incremental count [incr] is decreased by one to compensate that (known) error.

Figure 3 shows the flow chart of the position measurement algorithm, utilizing equations (2.1) and (2.2).

Figure 3. High-Resolution Position Measurement Algorithm



Maximum Tracking Speed n_{max}

The maximum revolution speed at which the algorithm tracks the high resolution position depends on the following:

- ❑ Line count, N, of the encoder
- ❑ Hysteresis angle α of the digitizing circuit
- ❑ Propagation delay between the analog and the digitized signals
- ❑ Delay time between sampling the analog signals and capturing the incremental counter

$$n_{max}[\mathrm{rpm}] = \frac{60}{N \cdot t_{delay}} \left(\frac{90° - \alpha_{Hysteresis}}{360°} \right)$$

Implementation on the TMS320F240

TMS320F240 QEP-Unit and ADC Module

The TMS320F240 incorporates a QEP circuit as well as two 10-bit analog-to-digital converters with two built-in sample and hold circuits. This minimizes the glue logic required to interface incremental encoders with analog sin/cos-output to the TMS320F240.

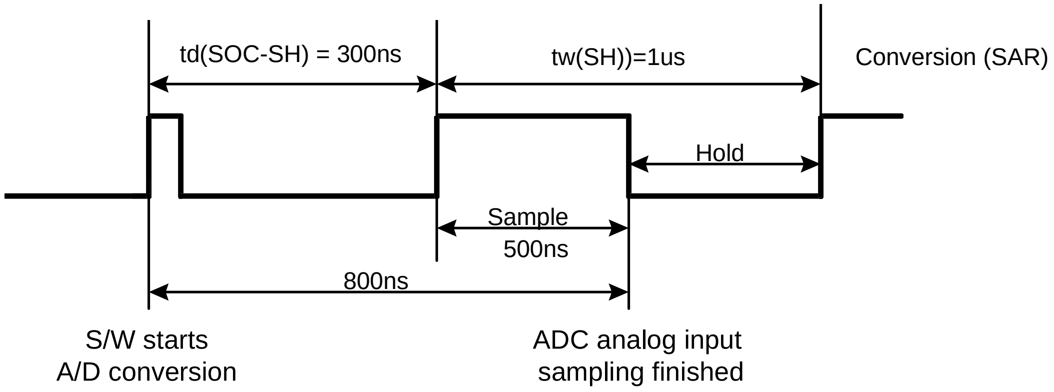
The QEP circuit decodes and counts the quadrature encoded input pulses on the TTL-compatible input pins, CAP1/QEP1 and CAP2/QEP2. The selected timer counts up, if CAP1/QEP1 is the leading sequence, and counts down, if CAP2/QEP2 is the leading sequence. Both edges are counted. However, a synchronization circuit suppresses spikes, since the input is sampled twice at two consecutive falling edges of the CPU instruction clock and is only recognized when its level is constant during that time. Hence, for a nominal CPU instruction cycle time **tc(CO)** = 50ns, any spike with a pulse width less than 50ns is suppressed. The maximum input frequency is given by

$$f_{\max(QEP)} \le \frac{1}{2t_{c(CO)} + 12ns} = 8.9 \text{ Mhz}$$

and is quite enough to meet the maximum output frequency of incremental rotary encoders.

The ADC module comprises two 10-bit analog-to-digital converters with eight multiplexed analog input channels per converter and has a minimum conversion time of $t_{W(SHC)}$ = 6.2us. Both ADCs can be started **simultaneously** by software, by external hardware pin, or by the event manager, allowing synchronization, e.g., to the PWM.

Figure 4. ADC Module Timing Diagram (ADC Clock = 1MHz)



After the ADCs are started, e.g., by software, the sampling of the two selected input channels starts with a delay of

$$t_{d(SOC\text{-}SH)} = 3\ t_{c(SYS)} = 300ns \text{ @SYSCLK of 10 MHz.}$$

The sample-and-hold time is tw(SH) = 1us (ADC clock = 1MHz), where the internal switched capacitor samples the analog input for 500ns and holds it for another 500ns before starting successive approximation conversion. Note that the external circuitry should be capable of charging the capacitor within ±1/2 LSB during sampling time.

To achieve **simultaneous sampling** of the two sinusoidal input signals, A and B, (ADC module) **and** the incremental counter (QEP module), the incremental counter has to be read/captured by software with a delay of td(SOC-SH) + ½ tw(SH) = 800ns (16 DSP instruction cycles), after writing a start command to the ADC control register.

For more information on the TMS320F240 peripherals, refer to [1],[2].

Hardware Interface

Any sin/cos-encoder, with analog output signals as shown in Figure 1, can be interfaced to the TMS320F240 DSP controller as shown in Figure 5. Note that the components and values used are applicable to an incremental encoder with a line count N = 2048 and a maximum speed of 12000 rpm, hence the maximum frequency is $f_{A,B,max}$ = 410 kHz. The main aspects of the circuit are discussed below.

Supply Voltage

The total circuit is supplied with only a single +5V supply for analog (VCCA) and digital part (VCC), respectively. Analog and digital ground should be connected close to the TMS320F240 AGND and GND input pins. AGND can be used to shield the differential signals ±A and ±B of the incremental rotary encoder.

Virtual Ground

The TL2425, a precision by-two voltage divider, generates the virtual ground at VREFH/2, which is exactly the mid-voltage of the ADC input and is interpreted as digital offset 0x8000. This eliminates the influence of the reference voltage to that digital offset.

ADC Input Voltage Range

The TMS320F240 VREFH input is connected to VCCA = +5V. VREFL is connected to AGND and decoupled with a 10uF tantalum capacitor. A 1.5 ohm resistor may be inserted to filter high frequency noise. The input range is VREFL = 0V . . +5V = VREFH.

Driving the ADC Input

The differential input signals ±A and ±B first pass through a high impedance unity-gain buffer (TLV2772) to eliminate the influence of the line impedance. The buffered differential signals are input to a differential amplifier (TLV2772) with a gain of 22/10. The resistors, which set the non-inverting and inverting gain, should be 1%, to keep the common mode rejection CMRR > 60dB. The non-inverting input network is referenced to the virtual ground VREFH/2 = 2.5V, hence the output voltage swing is:

$$0.3\mathrm{V} \le u_{A,B}(\varphi) \le 4.7\mathrm{V}\,.$$

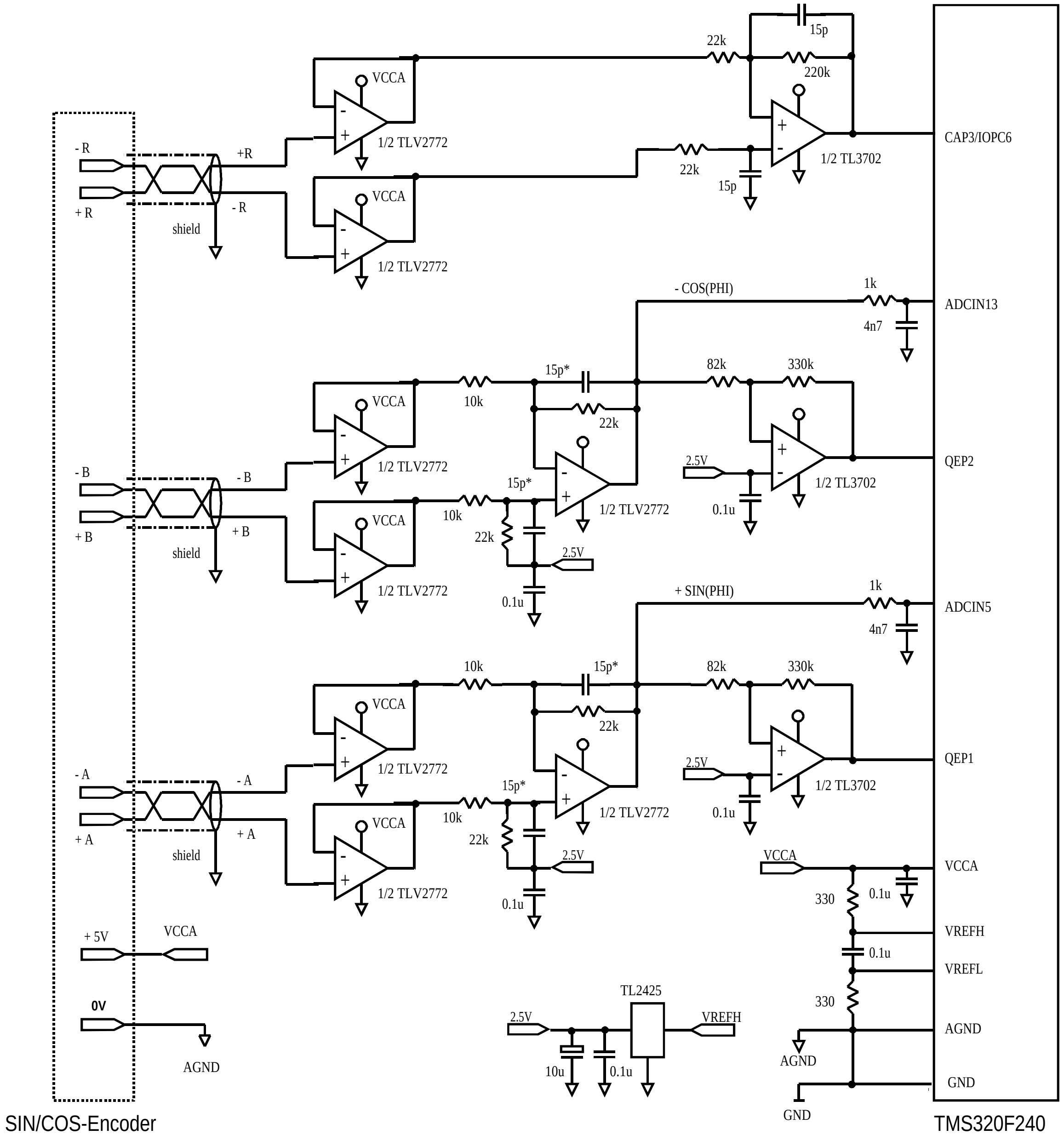
The output of the differential amplifier is filtered with an RC network, providing good noise rejection and accuracy at lower revolution speeds, where a precise analog phase information is required to increase position and/or speed resolution. The cut-off frequency is approximately 40kHz, which is, for the sin/cos-encoder, used in that report, with a line count N = 2048, equivalent to a rotation speed of approximately 1200rpm, and has to be changed according to the encoders parameter and other requirements. However, the capacitor should be as close as possible to the ADC's input and will charge the ADC's internal switched-capacitor rather than the op amp, which increases noise immunity and relaxes the op amp requirements for settling time and output current capability.

The TLV2772 LinCMOS dual rail-to-rail op amp is used because it allows single-supply operation and provides a gain bandwidth of 5MHz and a slew rate of 9V/us, hence operating up to the maximum encoders frequency $f_{A,B,max}$ = 410 kHz. Since 1/2 LSB = 2.5mV, offset voltage drift over temperature of 2uV/K does not affect system accuracy. Any initial offset voltage is removed digitally by software.

Driving the QEP Unit

The sinusoidal output of the differential amplifier is also input to a high-speed comparator (TLC3702). The threshold voltage is adjusted to 2.5V and has a hysteresis of approximately ±0.6V to increase noise immunity and prevent the comparator from toggling. The hysteresis of ±0.6V is equivalent to a phase hysteresis of ±15°. The TLC3702 LinCMOS dual-voltage comparator allows single-supply operation and provides TTL-compatible outputs with a rise time of 50ns; thus, it can be directly interfaced to the TMS320F240 QEP1,2 inputs without an additional Schmitt-trigger. The propagation delay [t_P] for a 40mV overdrive is typically 1us.

Figure 5. TMS320F240 Interface to sin/cos-Encoder



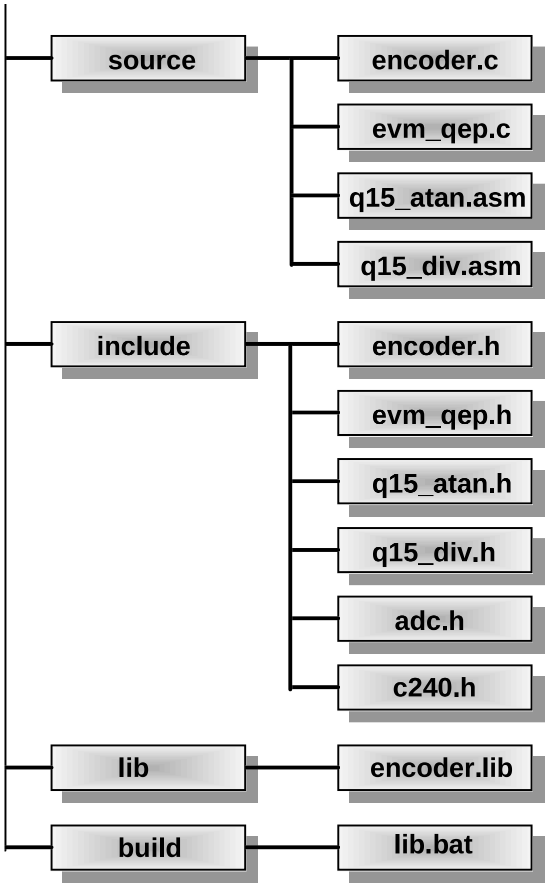
Software Implementation

Overview

The main part of the application software is written in ANSI C language to provide well-structured and readable software and ease the usage of the subroutines in own applications. However, time critical functions, such as fractional division or arctan-function are written in assembler but provide a C-compatible interface, thus can be called from any C program.

Figure 6 gives an overview of the software modules, which are structured into four directories. Each source module has its own header file, declaring the global variables, functions and constants.

Figure 6. Software Organization



The source files provide the following functions:

- ❑ encoder.c contains subroutines for initialization and position determination of the added sin/cos-encoders.
- ❑ evm_qep.c contains functions to initialize and serve the TMS320F240 QEP unit.
- ❑ q15_atan.asm and q15_div.asm provide fractional Q15 math functions arc tangents and division, respectively, and can be called from C.

The include directory contains the header files (which declare the global variables, functions, and constants) and TMS320F240 memory-mapped register as well as the C-macros to serve the TMS320F240 ADC module. The batch file lib.bat can be used to compile all source files and build the object library encoder.lib.

Functional Description

Global Variables

```
int  qep_diff;
unsigned qep_rollover;
unsigned encoder_position[2];
```

The encoder’s incremental count per revolution is held in `qep_rollover;` `qep_diff` is used as incremental offset. Both variables are required to handle the QEP-unit.

The array `encoder_position[]` is used to store the actual high-resolution position of the sin/cos-encoder. A 16-bit integer array was used rather than a 32-bit long variable because it better fits to the DSP architecture and results in lower code size. `encoder_position[0]` equals the (scaled) phase, 0° - 360°, between two consecutive incremental lines (four incremental steps); `encoder_position[1]` equals then the incremental line position. The high-resolution position angle is given by

$$\Phi_a = \frac{360^\circ}{\left(\frac{qep\_rollover}{4}\right)} \cdot \left(encoder\_position[1] + \frac{encoder\_position[0]}{2^{16}} \right)$$

Initialization Routines

The following functions are used for initialization. Note that the functions of level 2 or 3 are called from ‘higher’ level functions of level 1 and 2, respectively.

Table 1. Initialization Functions

Function Declaration	Level
`void Encoder_Init(unsigned encoder_rollover)`	1
`void QEP_Init(void)`	2
`void Encoder_ZeroPosition(void)`	2
`void Encoder_MatchIncrPhase(void)`	2
`void QEP_GetIncr(void)`	3

```
void Encoder_Init(unsigned encoder_rollover)
```

This is the main (level 1) initialization routine, which calls the other initialization functions of level 2, to initialize the ADC module,

□ enable both ADCs, ADC clock = 1 MHz,

the QEP unit,

□ 16-bit timer 2 counter, enable QEP inputs

and the global variables, where `qep_rollover` is initialized with the parameter encoder_rollover, as explained above. `void Encoder_ZeroPosition(void)`, as shown in Figure 7, is called to obtain absolute encoder position information.

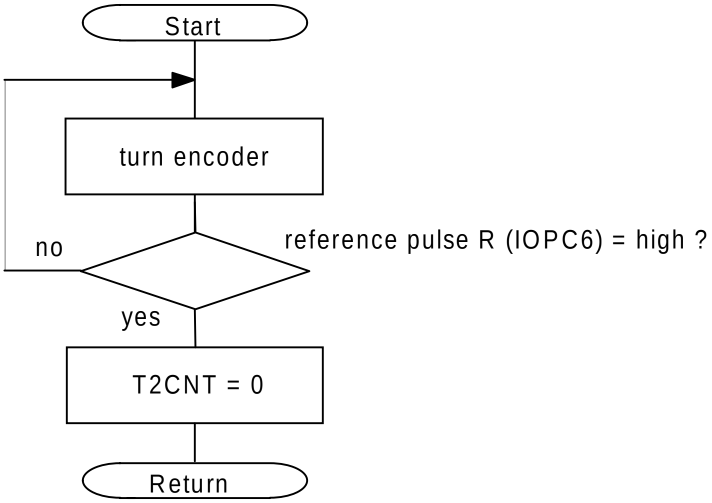
The encoder has to be turned until the reference mark signal gets high, indicating its zero position. `void Encoder_MatchIncrPhase(void)` (see Figure 8) then modifies the variable qep_diff, so that the phase information, stored in `encoder_position[0]` and the incremental count, which is returned from the function unsigned QEP_GetIncr(void), explained later, match according to equation (3.1):

$$\mathrm{QEP\_GetIncr()}\ \%\ 4 = \begin{cases} 0, & \text{if } 0\mathrm{x}0000 \le \mathrm{encoder\_position}[0] < 0\mathrm{x}4000 \\ 1, & \text{if } 0\mathrm{x}4000 \le \mathrm{encoder\_position}[0] < 0\mathrm{x}8000 \\ 2, & \text{if } 0\mathrm{x}8000 \le \mathrm{encoder\_position}[0] < 0\mathrm{xC}000 \\ 3, & \text{if } 0\mathrm{xC}000 \le \mathrm{encoder\_position}[0] < 0\mathrm{xFFFF} \end{cases} \qquad (3.1)$$

void Encoder_ZeroPosition(void)

To obtain absolute encoder position, the encoder is turned until the reference index gets high. The reference index is high only within its zero line, which is equal to the incremental counts 0,1,2 and 3. On the transition to high the software immediately initializes the timer 2 counter T2CNT = 0, indicating it's 'zero' line. The quadrant information is adjusted later by the function Encoder_MatchIncrPhase(). A second method would be turning the encoder into its zero position, lock it and then initialize timer 2 accordingly. The advantage of the first method is that the encoder does not have to be locked, thus it can be turned during initialization, since any quadrant offset error is compensated later.

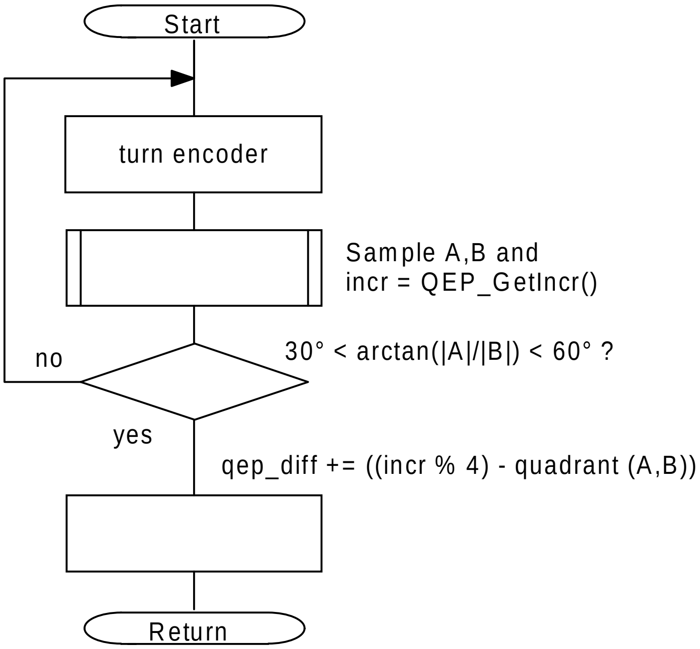
Figure 7. Flow Chart of Encoder_ZeroPostion()



void Encoder_MatchIncrPhase(void)

This function aligns the incremental count and the phase according to equation (3.1). Therefore, the encoder has to be turned slowly. Matching is done only in the ‘middle’ of a quadrant to exclude hysteresis effects.

Figure 8. Flow Chart of Encoder_MatchIncrPhase()


Start
turn encoder
Sample A,B and
incr = QEP_GetIncr()
30° < arctan(|A|/|B|) < 60° ?
no
yes
qep_diff += ((incr % 4) - quadrant (A,B))
Return

Routines for Obtaining Incremental Position

Obtaining the encoder's incremental position requires the two global variables, qep_rollover and qep_diff, which are derived from the sinusoidal signals A and B, described earlier. qep_rollover is equal to the incremental counts per revolution. qep_diff is used to match the incremental position to the phase as well as to handle rollover. Here Timer 2 is the selected QEP counter. Since it wraps at 0xFFFF to 0x0000, software has to take care of wrapping at the desired number, which corresponds to the number of incremental counts per revolution.

Two methods are used, depending on whether an encoder with 2^n counts, n = integer, per revolution is used or not.

*For encoders with **2^n counts**, it can be done by simply masking the upper 16-n unwanted bits of timer 2 counter:*

```
(T2CNT - qep_diff) & (qep_increments-1).
```

*For **non-2^n** encoders the function:*

```
unsigned QEP_GetIncr(void)
```

returns the (absolute) incremental count or position.

```
unsigned QEP_GetIncr(void).
```

This function checks timer 2 counter [T2CNT] for upper and lower limit, where qep_diff points to the lower and qep_diff + qep_rollover to the upper limit. If out of limit qep_diff is incremented or decremented by qep_rollover, as shown below:

```
        volatile unsigned buffer;
        buffer = T2CNT - qep_diff;
        while(buffer < 0)
        {
           buffer += qep_rollover;
           qep_diff -= qep_rollover;
        }
        while(buffer ≥ qep_rollover)
        {
           buffer -= qep_rollover;
           qep_diff += qep_rollover;
        }
        return buffer;    /* 0 ≤ buffer < qep_rollover */
```

Note: The subroutine is written in assembler to optimize speed and guarantee simultaneous capturing of the timer 2 counter (T2CNT) and the sinusoidal signals, A and B.

Routines for High-Resolution Position Interpolation

The following functions are used to obtain high-resolution position, which is stored in the array encoder_position[]. There are only two level 1 functions, which call functions of level 2 or 3.

Table 2. Functions for Obtaining High-Resolution Position

Function Declaration	Level
unsigned Encoder_SamplePosition(void)	1
void Encoder_CalcPosition(unsigned incr)	1
unsigned Encoder_CalcPhase(int sin, int neg_cos)	2
unsigned QEP_GetIncr(void)	2
unsigned q15p_atan(unsigned)	3
unsigned q15_div(unsigned nom, unsigned denom)	3

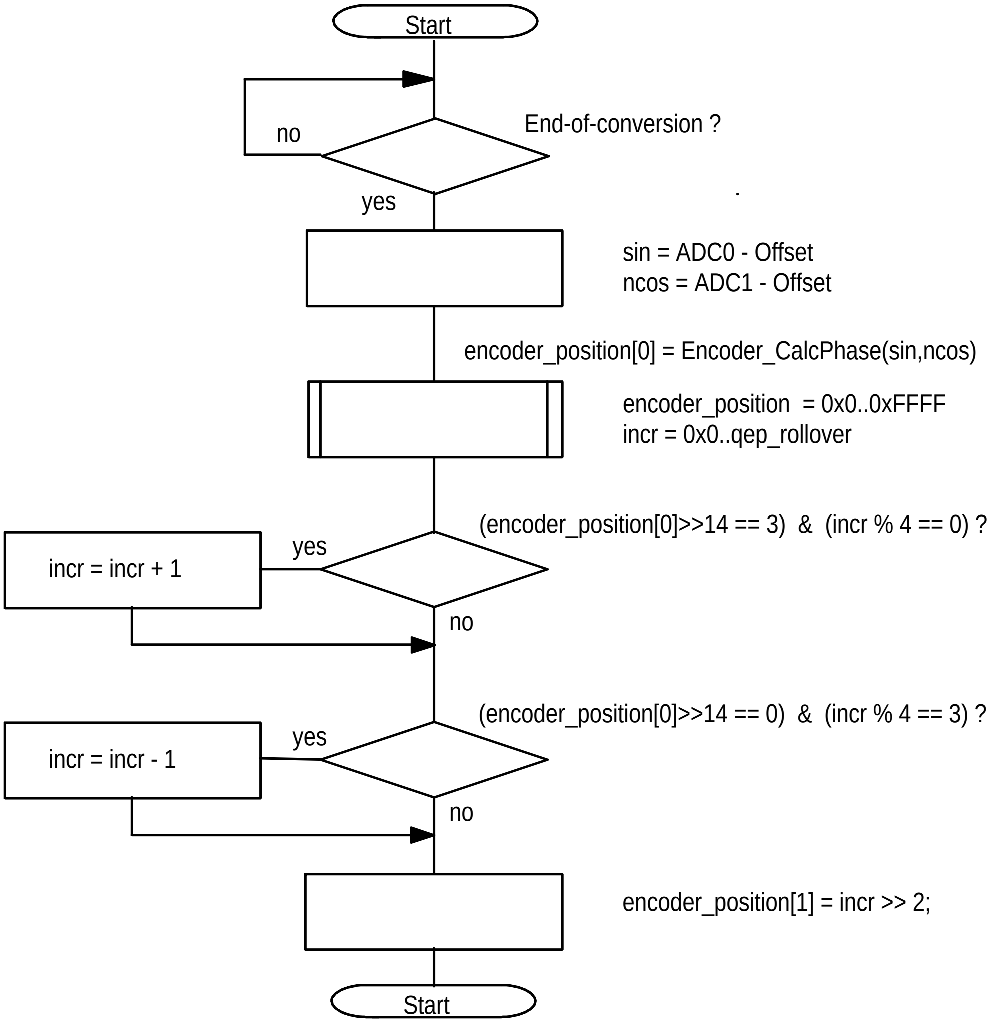
unsigned Encoder_SamplePosition(void),

This command initiates an analog-to-digital conversion of the sinusoidal input signals at the TMS320F240 analog input pins, ADCIN5 and ADCIN13. The command calls `QEP_GetIncr()`, which returns the modified incremental count [incr] that was sampled at the same time as the ADC inputs, hence **simultaneously.** This is achieved by reading the timer 2 counter (T2CNT), which holds the (not processed) incremental count exactly sixteen DSP clock cycles after writing a start command to the ADC control register. During that time slot interrupts must be disabled. (Refer to the section, *TMS320F240 QEP-Unit and ADC Module*.)

void Encoder_CalcPosition(unsigned incr).

This function calculates the interpolated high-resolution position and stores it to the global array encoder_position[] as defined in the section, *Global Variables*. The flow chart is outlined in Figure 9.

Figure 9. Flow Chart of Encoder_CalcPosition()


Start
End-of-conversion ?
no
yes
sin = ADC0 - Offset
ncos = ADC1 - Offset
encoder_position[0] = Encoder_CalcPhase(sin,ncos)
encoder_position = 0x0..0xFFFF
incr = 0x0..qep_rollover
(encoder_position[0]>>14 == 3) & (incr % 4 == 0) ?
yes
incr = incr + 1
no
(encoder_position[0]>>14 == 0) & (incr % 4 == 3) ?
yes
incr = incr - 1
no
encoder_position[1] = incr >> 2;
Start

The function passes the incremental count, which was sampled at the same time as the analog signals A and B, polls the ADC for end-of-conversion, reads the ADC FIFOs and adds an offset to get the 2's complement of the signals A = sin(φ) and B = -cos(φ). Optionally, software performs gain compensation. The two variables A = sin(φ) and B = -cos(φ) are passed to the function `unsigned Encoder_CalcPhase()`, which returns the scaled phase φ to encoder_position[0].

The next step is to ensure that the incremental count (the two LSBs, or least significant bits) and the phase (the two MSBs, or most significant bits) match as outlined in the section, *High-Resolution Position Determination*. This is done to compensate any hysteresis and propagation delay of the digitizing circuit, as shown in Figure 2 . This is required only for the 1st and 4th quadrant. The algorithm checks for these two cases and compensates the incremental count, if required. After that it performs the by-two right-shift of the incremental count (removing the redundant information) before storing the now correct incremental position to encoder_position[1]. (Refer also to *High-Resolution Position Determination*.)

```
unsigned Encoder_CalcPhase(int sin, int negcos)
```

This function returns the scaled phase φ_{scaled}, which is derived from the quotient of the two arguments sin = sin(φ) and negcos = -cos(φ). First it checks for sign to identify the correct quadrant and then it utilizes fractional division and arctan routines to derive the scaled phase according to: 0x0 ⇔ 0°, …, 0xFFFF⇔ 360°.

Fractional Math Routines

Two mathematical routines for division and inverse tangents support (positive) fractional numbers (Q15).

```
unsigned _q15_div(unsigned nom, unsigned denom),
```

with nom [0..+1], denom [0..+1] **> nom,** and return value = nom/denom [0..+1].

```
unsigned q15p_atan(unsigned arg)
```

with arg [0..+1] and the return value the (by PI) scaled angle according to [0 ⇔ 0°, 0x2000 ⇔ PI/4 = 45°]. The function uses a 128 point lookup table with interpolation, the error of the scaled angle is less that 2LSBs.

Application Interface (API)

All subroutines discussed in the previous paragraph are archived in the object library ENCODER.LIB and can be called from any C program.

The following steps are required when using sin/cos-encoders interfaced to the TMS320F240, as shown in Figure 5.

Step 1: Modify ENCODER.H

The encoder's incremental count per revolution and the analog input offset voltage of the sinusoidal signals, A and B, are defined in the header file ENCODER.H, as shown below for an sin/cos-encoder with a line count N=2048, equal to an incremental count of 8192. Note that A and B are interfaced to ADCIN5 and ADCIN13, respectively.

```
#define QEP_ROLLOVER    8192  /*incremental counts/rev.*/
asm("QEP_ROLLOVER  .set 8192 "); /*dito, assembler
support */
#define QEP_POWER2            /*undefine if incremental*/
                               /*count isn't power of 2*/

#define ENC_U0_OFFSET  (0x8000 + 0)  /* A (sin)  offset*/
#define ENC_U90_OFFSET (0x8000 + 300) /* B (-cos)
offset*/
```

Step 2: Run LIB.BAT

This recompiles the library ENCODER.LIB

```
dspcl -v2xx -as ..\source\*.asm
dspcl -v2xx -gs -mn -o2 -i..\include  ..\source\*.c
dspar -r encoder.lib  *.obj
copy *.lib ..\lib
```

Step 3: Calling Conventions when Using Sin/Cos-Encoders

Three function calls and one local variable are required within your own application software to initialize the TMS320F240 sin/cos-encoder interface and obtain high-resolution position, stored into the global array encoder_position[], declared in ENCODER.H.

```
#include "encoder.h"

/*-----------------------------------------------*/
/* Initialize TMS320F240 sin/cos-encoder interface */
/*-----------------------------------------------*/

Encoder_Init(QEP_ROLLOVER);
```

The two functions required for obtaining high-resolution position can be called from any C program, in many cases the interrupt service routine, e.g., current controller in field-oriented controlled AC drives.

```
#include "encoder.h"
volatile unsigned  buffer; /* local variable */
/*-------------------------------------------------------*/
/*Get position: encoder_position[0]= 0. FFFFh = 0°.360°*/
/*              encoder_position[1]= 0..QEP_ROLLOVER/4 */
/*-------------------------------------------------------*/
buffer = Encoder_SamplePosition();

/* insert any signal processing during the 6us a/d
   conversion time  */
Encoder_CalcPosition(buffer); /* writes to
encoder_position*/
```

Step 4: Include the Library ENCODER.LIB in Your Linker Command File

```
 -i .\encoder\lib  /* library search path */
 -l encoder.lib
```

Step 5: Compile All

Monitor and Test Program

The monitor program allows testing sin/cos-encoders interfaced as shown in the schematics, Figure 2, via any VT100 terminal program, without the need of writing your own application software.

All files necessary are packed into the file, ENCODER.ZIP, which can be downloaded from the Internet [8]. After unzipping you will get the following structure:

Table 3. Software Organization

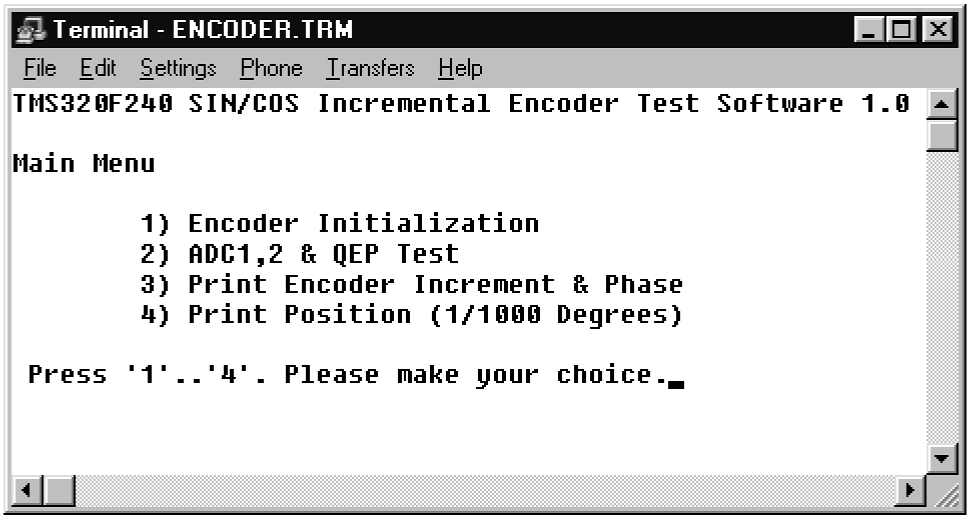
File/Directory	Description
MAIN.C	Main program, TMS320F240 initialization
MONITOR.C	Monitor program
MONITOR.H	Terminal settings (Baudrate)
RS232\LIB\RS323.LIB	RS232 terminal handler object library
RS232\INCLUDE	Directory of header files for RS232.LIB
ENCODER\LIB\ENCODER.LIB	Encoder functions object library
ENCODER\INCLUDE	Directory of header files for ENCODER.LIB
LINK.CMD	TMS320F240 Linker command file
CC.BAT	C compiler & linker batch file
ENCODER.TRM	Windows terminal settings (TERMINAL.EXE)

❑ If you do not have a 10 MHz external oscillator (such as TI’s EVM-240), open MAIN.C and modify the clock option as shown in MAIN.C

- ❑ Modify ENCODER.LIB according to steps 1 and 2 in the section, *Application Interface (API)*.
- ❑ Run CC.BAT, make sure you’ve set the DOS path for the TI code generation tools.
- ❑ Load MAIN.OUT with the JTAG emulator to your target or program it into the on-chip flash memory.
- ❑ Open any VT100 terminal program, e.g., terminal.exe, where encoder.trm defines the terminal settings: 9600 baud, 8 data bits, no parity, append LF (CR ⇒ CR/LF).
- ❑ Run the program.

You will see the following communication interface.

Figure 10. TMS320F240 Monitor Communications Interface



```
TMS320F240 SIN/COS Incremental Encoder Test Software 1.0

Main Menu

        1) Encoder Initialization
        2) ADC1,2 & QEP Test
        3) Print Encoder Increment & Phase
        4) Print Position (1/1000 Degrees)

 Press '1'..'4'. Please make your choice._
```

Results

Processor Utilization

For the following benchmarking results, all files compiled with the Texas Instruments Fixed-Point C Compiler 6.60, optimization level 2. [4]

Table 4 shows the required program and data memory for the library ENCODER.LIB and the cycle time required for obtaining high-resolution position.

Table 4. TMS320F240 Utilization for ENCODER.LIB

Program memory (ROM)	2ABh
Data memory	5 + 20h (stack)
CPU cycles / time	325 / 16.25 us

Table 5 shows the CPU cycles and memory requirements for the subroutines, which calculate the sin/cos-encoders high-resolution position. Note that all functions with level 2 or 3 are called by functions of level 1 or 2, respectively.

Table 5. TMS320F240 Utilization on Functional Level

Function	**Level**	**Program Memory (FLASH)**	**Data Memory RAM**	**CPU Cycles**
Encoder_SamplePosition()	1	15h	-	40
Encoder_CalcPosition()	1	6Ch	2	285
Encoder_CalcPhase()	2	71h	-	164
QEP_GetIncr()	2	24h	3	19 -
q15p_atan()	3	29h + 80h	-	48
q15_div()	3	Bh	-	30

NOTE:
CPU cycles are expected to be reduced by 20-50% when software routines are written in assembler language and calls are minimized.

Accuracy Analysis

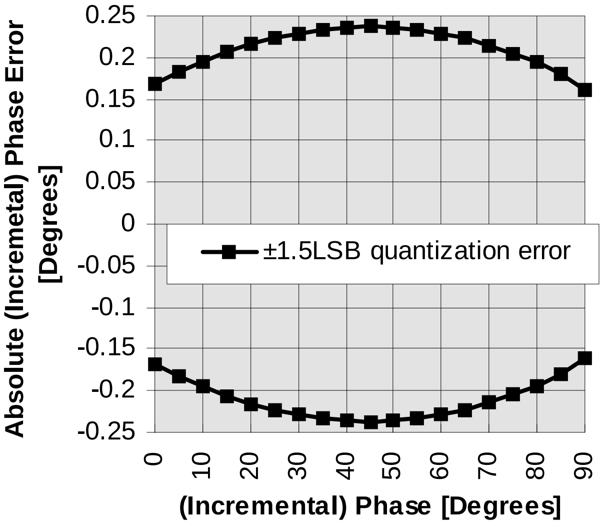
Parameters that influence the position resolution or accuracy are – besides the encoders accuracy itself – especially rotation speed, analog gain and offset (signal conditioning) and quantization (ADC, DSP), The equation below shows the absolute phase error as a function of quantization, gain and offset errors.

$$\Delta\varphi = \arctan\left(\frac{\left(1 \pm \Delta_{\mathrm{GAIN}}\right)\cdot \mathrm{A} \ \pm\Delta_{\mathrm{OFFSET}} \ \pm\Delta_{\mathrm{ADC}}}{\left(1 \mp \Delta_{\mathrm{GAIN}}\right)\cdot \mathrm{B} \ \mp\Delta_{\mathrm{OFFSET}} \ \mp\Delta_{\mathrm{ADC}}}\right) - \varphi$$

Quantization

Quantization is limited by the TMS320F240 10-bit ADC converter, which has a total error of less than ±1.5LSB. Digital truncation and rounding errors, as well as the accuracy of the inverse tangent function are negligible. Figure 11 shows the maximum absolute phase error (in degrees) within one incremental step for TMS320F240 implementation, which is independent of the sin/cos-encoders line count. The x-axis corresponds to the (el.) phase over one incremental step, the y-axis to the absolute phase error. The error shows a maximum at 45° and repeats each incremental step, respectively.

Figure 11. Maximum Absolute Phase Error over One Incremental Step



For an accuracy of ±1.5LSB of the ADC, the minimum position resolution for sin/cos-encoders is shown in Table 6. Compared to TTL-encoders with identical line count N, the resolution achieved is approximately 400-times better.

Table 6. Minimum Position Resolution with the TMS320F240 10-bit ADC for Encoders with Different Line Count

Line count N	***500***	***1024***	***1024***
Min. position resolution [arc seconds]	1.75	0.87	0.43

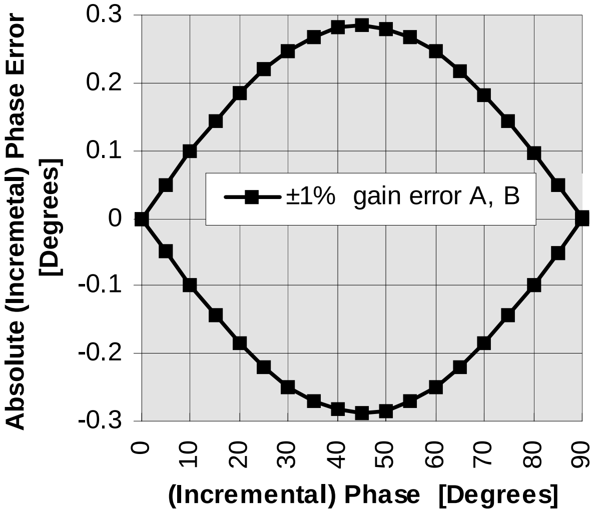
Signal Conditioning

Signal conditioning should not introduce additional error, therefore, the hardware parameter's accuracy should be within ½ LSB, equivalent to 5mV.

Any initial offset voltage is corrected by software and the op amps offset voltage temperature drift of 50uV/K can be neglected, since 1 LSB = 5mV.

The gain accuracy of the op amps, amplifying the two sinusoidal signals, A and B, naturally affects position accuracy, as shown in Figure 12, where the absolute phase error is shown over one quadrant for gain errors of 1%.

Figure 12. Absolute Phase Error over One Quadrant for A/B Gain Errors of ±1%



The error has a maximum at 45° and is in the range of the ADC. Therefore either the resistors, which set the gain, must match better than 1%, or software has to correct the gain accordingly, which is the preferred method. It is recommended to place resistors close together, however even with resistors' temperature drifts of 50ppm, gain keeps within ±0.1% for 20K temperature drop. Thus, this effect can be neglected.

Rotation Speed

Rotation speed does not affect the high-resolution position determination, as long as the phase shift between the analog sinusoidal signals and digitized signals, triggering the incremental counter is less than ±90°. If the analog signals and the incremental counter are sampled with a delay, this does increase or decrease the phase shift, respectively, and has to be considered. The maximum rotation speed allowing high-resolution position determination is given by:

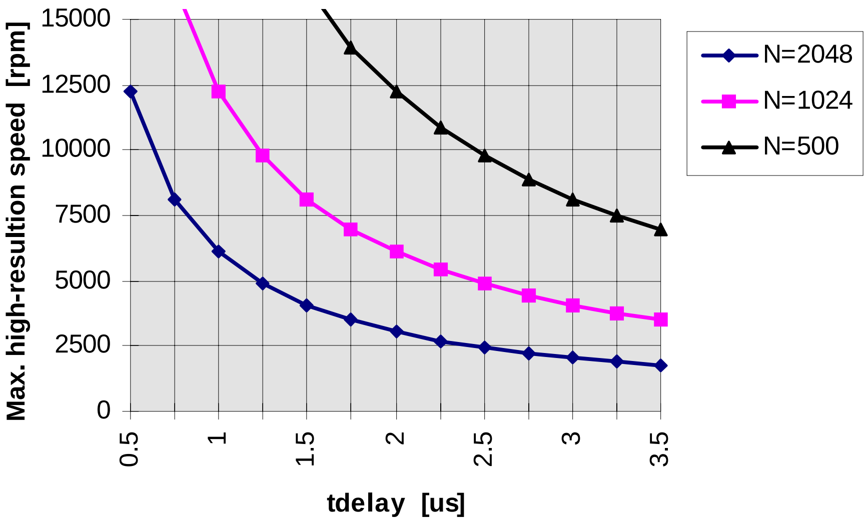
$$n_{max}\left[\text{rpm}\right] \approx \frac{60}{N \cdot t_{delay}}\left(\frac{90^\circ - \alpha_{Hysteresis}}{360^\circ}\right)$$

where [α] is the hysteresis of the comparator and [t_{delay}] the sum of propagation delay between the analog and digitized signal plus any delay time between sampling the analog signals and capturing the incremental counter.

Figure 13 shows the maximum speed to determine high-resolution position according to the above equation for a hysteresis α= 15° as function of the delay time, tdelay, for encoders with different line count N. To achieve these maximum ratings, the RC filter at the analog input of the 'F240 has to be modified accordingly.

Note that the 'F240 incremental counter runs up to 9MHz, thus there is no speed limitation regarding the incremental resolution.

Figure 13. Maximum Rotation Speed allowing High-Resolution Position Determination as Function of the Delay Time Ddelay



Conclusion

The TMS320F240 DSP controller is designed to meet a wide range of digital motor control applications, and thus also incorporates the peripherals to support TTL-encoders as well as sin/cos encoders.

This application report should help the user use sin/cos-encoders interfaced to the TMS320F240 DSP controller, which have the advantage of either a higher position resolution or reduced sensor cost, since sin/cos-encoders require less line counts as equivalent TTL-encoders to achieve the same resolution.

The sin/cos-encoder hardware interface requires only a few active components. A careful hardware design is also important, as shown in the previous section, to achieve the optimum results. The software to initialize the TMS320F240 and obtain high-resolution position is fully C-compatible and allows a quick and easy implementation within own digital control application programs, as shown in the section, *Implementation on the TMS320F240*.

The position resolution achieved is approximately 400 times better than with TTL-encoders of identical line count. The CPU execution time for obtaining high-resolution position is approximately 15us. The total program code size is approximately 680 words.

Note CPU cycles are expected to be reduced by 30-50%, when all software routines are written in assembler language and calls are minimized.

References

[1] *TMS320C24x DSP Controllers - Reference Set*: Vol.1 + 2, Texas Instruments Inc., 1997.

[2] *TMS320F240* Data Sheet, Texas Instruments Inc., 1997.

[3] *TMS320C2x/C2xx/C5x Optimizing C Compiler User’s Guide*, Texas Instruments Inc., 1995.

[4] *Obtaining Absolute Encoder Position on a TMS320F240*, Designer's Notebook #82, Texas Instruments Inc.,1997.

[5] *TLV2772 Advanced LinCMOS Rail-to-Rail Dual Op Amp* Data Sheet, Texas Instruments Inc.,1998.

[6] *TLC3702 Dual Micropower LinCMOS Voltage Comparator* Data Sheet, Texas Instruments Inc.,1991.

[7] *Selection Guide for Incremental Rotary Encoders*, Heidenhain GmbH, 1997.

[8] INTERNET: **http://www.ti.com**

[9] FTP Server: ftp://ftp.ti.com/pub/tms320bbs

Appendix A. Source Code

Appendix A provides the listing the C and assembler source code discussed in the section, *Software Implementation*.

ENCODER.C

```
/*==================================================================
  Name:             ENCODER.C
  Project:          ENCODER
  Originator:       Martin Staebler
  Description:      Provides functions for handling incremental
                    encoder with analog sin/cos output signal
                    to achieve higher position resolution
 ====================================================================

       Function List:   void   Encoder_Init(unsigned rollover)
                        void   Encoder_ZeroPosition(void)
                        void   Encoder_MatchIncrPhase(void)
                        void   Encoder_CalcPhase(int sin,int ncos)
                        unsigned Encoder_SamplePosition(void)
                        void   Encoder_CalcPosition(unsigned
qep_incr_sample)

        Status:         OK

        Target:         TMS320C240

        History:        (Date, Revision, Who, What)
        ------------------------------------------------------------
        02/11/97        1.0     STAE    Preliminary

===================================================================*/
/*--------------*/
/* Header Files */
/*--------------*/
#include <stdlib.h>
#include <math.h>
#include <q15_div.h>            /* q15.lib */
#include <q15_atan.h>           /* q15.lib */
#include "c240.h"            /* TMS320C240 memory mapped registers */
```

```
#include "adc.h"             /* ADC macros */
#include "evm_qep.h"         /* QEP unit */
#include "monitor.h"         /* test program & monitor */
#include "encoder.h"         /* encoder definition */

/*----------------------*/
/* Variable Declaration */
/*----------------------*/
volatile unsigned  encoder_position[2];


/*=================================================================*/
/* void Encoder_Init(unsigned rollover)                            */
/*=================================================================*/
/* Function:      Incremental encoder initialization              */
/*                                                                */
/* Arguments:   Incremental counts/revolution = 4-times line count */
/*                                                                */
/* Return value:  None                      */
/*=================================================================*/
void Encoder_Init(unsigned rollover)
{
    /*---------------------*/
    /* Initialize F240 ADC */
    /*---------------------*/
    ADCTRL1  = 0x1800;           /* enable both ADC's */
    ADCTRL1 |= 0x0100;           /* clear interrupt flag */
    ADCTRL2  = 0x0003;           /* ADC_CLOCK = SYSCLK/10 = 1MHz */

    /*------------------------------*/
    /* Initialize F240 QEP Circuit  */
    /*------------------------------*/
    QEP_Init(0x0,rollover);      /* QEP counts Timer 2 */

    /*------------------------------------------------*/
    /* Get zero incremental position (via index) and  */
    /* synchronize incremental count and phase        */
    /*------------------------------------------------*/
    Encoder_MSG();          /* turn encoder into zero position */
    Encoder_ZeroPosition(); /* detect zero position adjust counter */
    Encoder_MatchIncrPhase();/* match counter (quadrant) and phase */

} /* end Encoder_Init */


/*=================================================================*/
/* void Encoder_ZeroPosition(void)                                */
/*=================================================================*/
/* Function: Modify/adjust qep_diff, so that the incremental count */
/*        of the encoder (using function QEP_GetIncr() matches with*/
/*        phase derived from the analog sin/-cos signal, as shown  */
/*            below.                                               */
/*=================================================================*/
void Encoder_ZeroPosition(void)
{
   /* Configure IOPC6/CAP3 to detect level */
   OCRB &= 0xFFBF;                      /* clear bit 6 */
   PCDATDIR &= 0xBFFF;                  /* clear bit 14 */
   while ( !(PCDATDIR & 0x0040));      /* poll bit 6 */
   T2CNT = 0x0;
}

/*=================================================================*/
/* void Encoder_MatchIncrPhase(void)                               */
/*=================================================================*/
/* Function:  Modify/adjust qep_diff, so that the incremental count*/
/*     of the encoder (using function QEP_GetIncr() matches with   */
/*       phase derived from the analog sin/-cos signal, as shown   */
/*       below.                                                    */
/*=================================================================*/
void Encoder_MatchIncrPhase(void)
{
    /*-----------------*/
    /* local variables */
    /*-----------------*/
    char  c;
    volatile int      buffer[2];
    volatile unsigned    ubuffer[2];

    /*----------------------------------------------------------*/
    /* Synchronize incremental count and encoder phase         */
    /*----------------------------------------------------------*/
    /* Assumption: ADCIN5  <--> QEP1/sin(x)                    */
    /*             ADCIN13 <--> QEP2/negcos(x)                 */
    /*                                                         */
    /* (T2CNT % QEP_ENCODER) % 4 + qep_diff =  Quadrant 0,1,2,3 */
    /*----------------------------------------------------------*/
    do
    {
       /*------------------------*/
       /* aquire encoder signals */
       /*------------------------*/
       ubuffer[1] = Encoder_SamplePosition();

       ADC_READ2(buffer[0],buffer[1]);

       /* Correct U0 and U90 input signals offset */
       buffer[0] = buffer[0] - ENC_U0_OFFSET;
       buffer[1] = buffer[1] - ENC_U90_OFFSET;

       /*---------------------------------------------*/
       /* check, if nearly in the middle of a quadrant */
       /*---------------------------------------------*/
    }  while ( abs(abs(buffer[0])-abs(buffer[1])) > 0x4000);

    /*----------------------------------------*/
    /* init qep_diff for phase/count matching */
    /*----------------------------------------*/
    ubuffer[0] = Encoder_CalcPhase(buffer[0],buffer[1]);
    ubuffer[0] = (ubuffer[0] >> 14) & 0x0003;  /* extract quadrant */
    ubuffer[1] = ubuffer[1] & 0x0003;

    qep_diff = qep_diff + ((int) ubuffer[1] - (int) ubuffer[0]);
}

/*=================================================================*/
```

```
/* unsigned Encoder_SamplePosition(void);                          */
/*=================================================================*/
/* Function:      Sample encoder signals SIMULTANEOUSLY           */
/*                SIN, -COS  --> Channel 5, 13 (hardcoded)        */
/*                INCREMENTS --> Timer 2 counter (T2CNT)          */
/*                                                                */
/* Arguments:     Buffer for increments sample                    */
/*                                                                */
/* Return value:  Incremental count                               */
/*================================================================*/
asm("ADCTRL1    .set    7032h                                   ");
asm("T2CNT      .set    7405h                                   ");
asm("           .ref    _qep_diff                               ");
asm("           .ref    _QEP_GetIncr                            ");
asm("           .def    _Encoder_SamplePosition                 ");
asm("_Encoder_SamplePosition:                                   ");
asm("           ldp     #ADCTRL1/128                            ");
asm("           lacl    ADCTRL1                                 ");
asm("           and     #0FF81h         ;clear channels         ");
asm("           or      #005Bh          ;select channel 5 and 13");
asm("           sacl    ADCTRL1         ;(1) start ADC's        ");

#ifdef QEP_POWER2
asm("           ldp     #_qep_diff                              ");
asm("           lacl    #0                                      ");
asm("           sub     _qep_diff                               ");
asm("           ldp     #(T2CNT/128)                            ");
asm("           rpt     #6                                      ");
asm("           nop                                             ");
asm("     ;capture Timer 2 800ns after ADC start (1)            ");
asm("     ;-----------------------------------------            ");
asm("           add     T2CNT                                   ");
asm("           and     #(QEP_ROLLOVER-1)                       ");
asm("           ret                                             ");

#else
asm("           rpt     #5                                      ");
asm("           nop                                             ");
asm("     ;capture Timer 2 800ns after ADC start (1)            ");
asm("     ;-----------------------------------------            ");
asm("           call    _QEP_GetIncr   ; 500ns (NOP) + 200ns (call)
");
asm("                                  ; + 100ns in subroutine   ");
asm("           ret                                              ");

#endif


/*================================================================*/
/* void Encoder_CalcPosition(unsigned qep_incr_sample);           */
/*================================================================*/
/* Function:      Calc encoder position                           */
/*                               increments (Timer 2)             */
/*                                                                */
/* Arguments:     increments sample                               */
/*                                                                */
/* Global var's:  unsigned qep_position[2]                        */
/*                                                                */
/* Return value:  None                                            */
/*================================================================*/
void Encoder_CalcPosition(unsigned qep_incr_sample)
{
    /*--------------*/
    /* locals var's */
    /*--------------*/
    volatile int sin_sample;
    volatile int ncos_sample;
    volatile unsigned incr;
    volatile unsigned buffer;

    incr = qep_incr_sample;

    /*---------------------------------------------*/
    /* read converted sin and -cos encoder signals */
    /*---------------------------------------------*/
    ADC_READ2(sin_sample,ncos_sample);

    /* Correct U0 (sin_sample and U90 (ncos_sample) offset */
    sin_sample  = sin_sample - ENC_U0_OFFSET;
    ncos_sample = ncos_sample - ENC_U90_OFFSET;

    /*-------------------*/
    /* phase calculation */
    /*-------------------*/
    encoder_position[0] = Encoder_CalcPhase(sin_sample, ncos_sample);

    /*---------------------------------------------------------*/
    /* correct incremental steps according to phase information */
    /*---------------------------------------------------------*/
    buffer = ((encoder_position[0] >> 14) & 0x0003);
    switch (buffer)
    {
       case 0:  if ((incr & 0x0003) == 3)
                   incr = (incr + 1) & (qep_rollover-1);
                break;

       case 3:  if ((incr & 0x0003) == 0)
                   incr = (incr - 1) & (qep_rollover-1);
                break;
    } /* switch */

    /* remove (redundant) quadrant information (last two bits) */
    encoder_position[1] = (incr >> 2);
}

/*=============================================================*/
/* int Encoder_CalcPhase(int qep_sin,int qep_negcos);          */
/*=============================================================*/
/* Function:      Incremental encoder initialization           */
/*                                                             */
/* Arguments:     None                                         */
/*                                                             */
/* Return value:  None                                         */
/*=============================================================*/
int  Encoder_CalcPhase(int qep_sin, int qep_negcos)
{
    int  phase;
    int  buffer;
   /*------------------------------------------*/
   /* general calculation, within 1st quadrant */
   /*------------------------------------------*/
   if (abs(qep_sin) == abs(qep_negcos))
      phase = (PI/4);
   else if (abs(qep_sin) < abs(qep_negcos))
   {
      buffer = q15_div(abs(qep_sin),abs(qep_negcos));
      /* phase  = q15_atan(buffer);  */
      phase  = q15p_atan(buffer);
   }
   else
   {
      buffer = q15_div(abs(qep_negcos),abs(qep_sin));
      /* phase  = (PI/2) - q15_atan(buffer); */
      phase  = (PI/2) - q15p_atan(buffer);
   }

   /*------------------------------*/
   /* get 2nd, 3rd to 4th quadrant */
   /*------------------------------*/
   if (qep_sin >= 0)
   {
      if (qep_negcos > 0)
         phase = PI - phase;    /* 2nd quadrant */
   } /* end if */

   else
   {
      if ( qep_negcos > 0 )
         phase = PI + phase;    /* 3rd quadrant */
      else
         phase = -phase;        /* 4th quadrant */
   } /* end else */

   return  phase;
}
```

EVM_QEP.C

```
/*==================================================================
  Name:           EVM_QEP.C
  Project:        C240.LIB
  Originator:     Martin Staebler
  Description:    Initializes the QEP circuit module
                  Provides functions for serving module
===================================================================

        Function List:  void     QEP_Init(unsigned zero_position)
                        unsigned QEP_GetIncr(void)
        Status:

        Target:         TMS320C240

        History:        (Date, Revision, Who, What)
        ------------------------------------------------------------
        02/11/97        1.0     STAE    Preliminary
====================================================================*
/

/*===============*/
/* Include Files */
/*===============*/
#include        "c240.h"
#include        "evm_qep.h"

/*==============*/
/* Global Var's */
/*==============*/
asm("       .bss   _qep_rollover,1 ");  /* align on ONE PAGE */
asm("       .bss   _qep_diff,1  ");
asm("       .bss   _qep_temp,1  ");


/*=================================================================*/
/* void QEP_Init(unsigned zero_position, unsigned rollover)        */
/*=================================================================*/
/*Function:       Incremetal encoder (QEP Circuit) initialization  */
/*                based on timer 2                                 */
/*                                                                 */
/* Arguments:     unsigned zero_position                           */
/*                unsigned rollover                                */
/* Return value:  none                                             */
/*=================================================================*/

void QEP_Init(unsigned zero_position, unsigned rollover)
{
   OCRB |= 0x30;        /* enable qep1/2 mux'd inputs */

   T2CON = 0xD83A;      /* enable qep circuit using timer2 */
               /* bit 15,14: timer 2 doesn't on emulation suspend */
               /*   bit 13-10: directional up/down                */
               /*   bit 5-4:   qep pulse circuit triggers timer 2 */
               /*   bit 3-2:   compare no shadowed                */
               /*   bit 1:     enable compare                     */
   T2CON |= 0x0040;     /* enable timer 2 */

   CAPCON &= 0x1D0F;
   CAPCON |= 0xE000;    /* enable qep decoder circuit */

   /* init rollover value and zero position for qep timer 2 */
   T2CNT = zero_position;
   qep_rollover = rollover;
   qep_diff = 0;
}

/*=================================================================*/
/* unsigned QEP_GetIncr(void)                                      */
/*=================================================================*/
/* Funtion       Returns incremental step of encoder               */
/*               4-times resolution                                */
/* Arguments:    None                                              */
/*                                                                 */
/* Return value: unsigned incremental value module qep_rollover    */
/*=================================================================*/
asm("           .text                                           ");
asm("           .def    _qep_rollover                           ");
asm("           .def    _qep_diff                               ");
asm("           .def    _qep_temp                               ");
asm("           .def    _QEP_GetIncr                            ");
asm("T2CNT      .set    7405h                                   ");

asm("_QEP_GetIncr:                                              ");
asm("       ;read timer 2 counter                               ");
asm("       ;--------------------                               ");
asm("           ldp     #T2CNT/128                              ");
asm("           lacc    T2CNT                                   ");
asm("           ldp     #_qep_temp                              ");
asm("           sacl    _qep_temp   ;save temporarely       ");
asm("                                                           ");
asm("       ;modulo calculation                                 ");
asm("       ;------------------                                 ");
asm("check:      lacc    _qep_temp,16                           ");
asm("            sub     _qep_diff,16                           ");
asm("            bcnd    Nega,lt         ; ACC negative         ");
asm("            sub     _qep_rollover,16                       ");
asm("            bcnd    OK, lt          ; ACC is OK            ");
asm("            lacc    _qep_diff,16                           ");
asm("            add     _qep_rollover,16                       ");
asm("            sach    _qep_diff                              ");
asm("            b       check;                                 ");
asm("                                                           ");
asm("Nega:       lacc    _qep_diff,16                           ");
asm("            sub     _qep_rollover,16                       ");
asm("            sach    _qep_diff                              ");
asm("            b       check                                  ");
asm("                                                           ");
asm("OK:         add     _qep_rollover,16                       ");
asm("                                                           ");
asm("EPI:        sach    _qep_temp                              ");
asm("            lacc    _qep_temp    ;ACC = Return Value       ");
asm("            ret                                            ");
```

Q15_DIV.ASM

```
;====================================================================
; Name:          Q15_DIV.ASM
; Project:       Q15.LIB
; Originator:    Martin Staebler
;====================================================================

;-------------------------------------------------------------------
; unsigned _q15_div(unsigned nom,unsigned denom)
;-------------------------------------------------------------------
; Function:     positive fractional q15 division
;               range from 0000h .. 7FFFF
;
; Arguments:    unsigned nominator:    q15 [0000h - 7FFFFh]
;               unsigned denominator:  q15 > nominator
;               --> !!! check beforehand  !!!
;
```

```
; Return value:  dividend q15 <--> 0000h - 7FFFh
;------------------------------------------------------------------
                .def    _q15_div
                .text
_q15_div:
        ;context save
        ;------------
                popd    *+              ;push return address
                                        ;no local var's
                sar     AR1,*
                lar     AR2,*,AR2
                sbrk    #2           ;AR2 points to first parameter

        ;division for POSITIV fractional numbers
        ;---------------------------------------

                lacc    *-,16
                rpt     #14
                subc    *
                                ;ACCL = quotient

        ;context restore
        ;---------------
                mar     *,AR1
                sbrk    #1           ;pop local var's + 1 from stack
                pshd    *            ;restore return address
                ret
```

Q15_ATAN.ASM

```
;====================================================================
; Name:           Q15_ATAN.ASM
; Project:        Q15.LIB
; Originator:     Martin Staebler
;====================================================================
;            Function:   int q15_atan(int)
;                  unsigned q15p_atan(unsigned)
;        Status:
;
;        Target:          TMS320C240
;
;        History:         (Date, Revision, Who, What)
;        -----------------------------------------------------------
;        02/11/97         1.0     STAE    Preliminary
;====================================================================


;-------------------------------------------------------------------
; Include files
;-------------------------------------------------------------------
               .include        "q15_atan.inc"   ;lookup table

;-------------------------------------------------------------------
; int q15_atan(int);
;-------------------------------------------------------------------
```

```
; Function:      arcus tanges, for fractional q15 format
;
; Arguments:     fractional q15
;                min: -1.0   <--> 8000h
;                max: 0.9999 <--> 7FFFh
;
; Return value:  scaled angle (-PI/4 .. PI/4)
;                scaling: PI  (e.g. atan(1.0) = 0.25 or 2000h)
;
; Error:         < 2 LSB  (128 point lookup table)
;-------------------------------------------------------------------
                .def    _q15_atan       ;define global
                .text
_q15_atan:
        ;context save
        ;------------
                popd    *+              ;push return address
                sar     AR0, *+         ;push old frame pointer
                sar     AR1, *
                lar     AR0,*           ;init new frame pointer
                adrk    #3              ;alocate space for two
                                        ;local variables

                mar     *,AR2
                lar     AR2,#-3
                mar     *0+             ;AR2 = &parameter

         ;check if negative or -1
         ;-----------------------
                lacc    *
                adrk    #3
                sacl    *               ;local #1 = abs(parameter)
                bcnd    OK,GEQ
                sub     #8000h
                bcnd    MINUS_1,EQ
                lacc    *
                neg
                sacl    *               ;local #1 = abs(parameter)

        ;calculte atan for POSITIV fractional numbers
        ;--------------------------------------------
OK:             lacc    *+,tablen_lg2+1
                                 ;lookup table length = 2^tablen_lg2
                sach    *        ;local #2 = first table address
                lacc    #table
                add     *
                tblr    *+       ;local #2 = first value
                add     #1
                tblr    *        ;local #3 = second value
                lacc    *-
                sub     *+       ;ACC = difference = local #3 - #2
                sacl    *        ;local #3 = difference
                lt      *        ;T = difference
                sbrk    #2       ;AR2 points to local #1
                lacc    *,tablen_lg2    ;ACC = local #1 << tablen_lg2
                and     #7FFFh          ;make distance positiv value
                sacl    *               ;local #1 = distance
                mpy     *+              ;differnce * distance
                spm     1
                lacc    *-,16        ;ACCH = local #2 = first value
                apac                 ;ACC += distance * difference
                sach    *            ;local #1 = 'positiv' result

        ;correct sign, if necessary
        ;--------------------------
                lar     AR2,#-3
                mar     *0+             ;AR2 = &parameter
                lacc    *
                adrk    #3
                bcnd    POSITIV,GEQ
                lacc    *
                neg                     ;2's complement
                b       EPIO

MINUS_1:        lacc    #-4000h
                b       EPIO

POSITIV:        lacc    *               ;ACCL = fractional result
                ;b       EPIO           ;does already


        ;context restore
        ;---------------
EPIO:           spm     0               ;default 'C' setting
                mar     *,AR1
                sbrk    #(3+1)          ;pop local var's+1 from stack
                lar     ar0, *-         ;restore old frame pointer
                pshd    *               ;restore return address
                ret


;--------------------------------------------------------------------
; unsigned q15p_atan(unsigned);
;--------------------------------------------------------------------
; Function:      arcus tanges, for positiv fractional q15 format
;
; Arguments:     fractional q15
;                min: 9.0    <--> 0000h
;                max: 0.9999 <--> 7FFFh
;
; Return value:  scaled angle (0 .. PI/4)
;                scaling: PI  (e.g. atan(1.0) = 0.25 or 2000h)
;
; Error:         < 2 LSB  (128 point lookup table)
;--------------------------------------------------------------------
                .def    _q15p_atan       ;define global
                .text
_q15p_atan:
        ;context save
        ;------------
                popd    *+         ;push return address
                sar     AR0, *+    ;push old frame pointer
                sar     AR1, *
                lar     AR0,*      ;init new frame pointer
                adrk    #3   ;alocate space for three local variables
                mar     *,AR2
                lar     AR2,#-3
                mar     *0+             ;AR2 = &parameter

         ;local #1 = parameter
         ;--------------------
                lacc    *
                adrk    #3
                sacl    *               ;local #1 = parameter

        ;calculte atan for POSITIV fractional numbers
        ;--------------------------------------------
                lacc    *+,tablen_lg2+1
                                ;lookup table length = 2^tablen_lg2
                sach    *      ;local #2 = first table address
                lacc    #table
                add     *
                tblr    *+          ;local #2 = first value
                add     #1
                tblr    *           ;local #3 = second value
                lacc    *-
                sub     *+          ;ACC = difference = local #3 - #2
                sacl    *           ;local #3 = difference
                lt      *           ;T = difference
                sbrk    #2          ;AR2 points to local #1
                lacc    *,tablen_lg2  ;ACC = local #1 << tablen_lg2
                and     #7FFFh        ;make distance positiv value
                sacl    *             ;local #1 = distance
                mpy     *+            ;differnce * distance
                spm     1
                lacc    *-,16         ;ACCH = local #2 = first value
                apac                  ;ACC += distance * difference
                sach    *             ;local #1 = 'positiv' result
                lacc    *             ;put into lowwer ACC

        ;context restore
        ;---------------
                spm     0               ;default 'C' setting
                mar     *,AR1
                sbrk    #(3+1)          ;pop local var's+1 from stack
                lar     ar0, *-         ;restore old frame pointer
                pshd    *               ;restore return address
                ret
```

Q15_ATAN.INC

```
;---------------------------
; atan(x), with 0 <= x <= 1
;---------------------------
; output scalingfactor = PI
;---------------------------
tablen_lg2      .set    7
                .text
;-------------------
; atan lookup table
;-------------------
table:
```

```
            .word    0
            .word    81
            .word    163
            .word    244
            .word    326
            .word    407
            .word    489
            .word    570
            .word    651
            .word    732
            .word    813
            .word    894
            .word    975
            .word    1056
            .word    1136
            .word    1217
            .word    1297
            .word    1377
            .word    1457
            .word    1537
            .word    1617
            .word    1696
            .word    1775
            .word    1854
            .word    1933
            .word    2012
            .word    2090
            .word    2168
            .word    2246
            .word    2324
            .word    2401
            .word    2478
            .word    2555
            .word    2632
            .word    2708
            .word    2784
            .word    2860
            .word    2935
            .word    3010
            .word    3085
            .word    3159
            .word    3233
            .word    3307
            .word    3380
            .word    3453
            .word    3526
            .word    3599
            .word    3670
            .word    3742
            .word    3813
            .word    3884
            .word    3955
            .word    4025
            .word    4095
            .word    4164
            .word    4233
            .word    4302
            .word    4370
```

```
          .word   4438
          .word   4505
          .word   4572
          .word   4639
          .word   4705
          .word   4771
          .word   4836
          .word   4901
          .word   4966
          .word   5030
          .word   5094
          .word   5157
          .word   5220
          .word   5282
          .word   5344
          .word   5406
          .word   5467
          .word   5528
          .word   5589
          .word   5649
          .word   5708
          .word   5768
          .word   5826
          .word   5885
          .word   5943
          .word   6000
          .word   6058
          .word   6114
          .word   6171
          .word   6227
          .word   6282
          .word   6337
          .word   6392
          .word   6446
          .word   6500
          .word   6554
          .word   6607
          .word   6660
          .word   6712
          .word   6764
          .word   6815
          .word   6867
          .word   6917
          .word   6968
          .word   7018
          .word   7068
          .word   7117
          .word   7166
          .word   7214
          .word   7262
          .word   7310
          .word   7358
          .word   7405
          .word   7451
          .word   7498
          .word   7544
          .word   7589
          .word   7635
```

```
     .word    7679
     .word    7724
     .word    7768
     .word    7812
     .word    7856
     .word    7899
     .word    7942
     .word    7984
     .word    8026
     .word    8068
     .word    8110
     .word    8151
     .word    8192
```

ENCODER.H

```
/*==============================================================*/
/* Name:           ENCODER.H             */
/* Project:        ENCODER                                      */
/* Originator:     Martin Staebler          */
/* Description:    Header File for ENCODER.C      */
/*==============================================================*/
#ifndef  __ENCODER_H_
#define  __ENCODER_H_

/*===============================================================*/
/* USER SETTABLE ENCODER CONSTANTS                               */
/*===============================================================*/
/*---------------------------*/
/* 1) Incremental resolution */
/*---------------------------*/
#define QEP_ROLLOVER    8192      /* incremental counts          */
asm("QEP_ROLLOVER  .set 8192 ");  /* dito, assembler support     */
/* #define QEP_POWER2 */          /* undefine if incremental     */
                                  /*   count isn't a power of 2  */

/*-------------------------------------------------*/
/* 2) 'F240 ADC's input channel selection & offset */
/*-------------------------------------------------*/
/* Pin ADCIN5  (ADC0, channel 5)  selected for U0  */
/* Pin ADCIN13 (ADC1, channel 5)  selected for U90 */
#define ENC_U0_OFFSET   (0x8000 + 0)    /* U0  (sin) offset      */
#define ENC_U90_OFFSET  (0x8000 + 300)  /* U90 (-cos) offset     */


/*==================*/
/* OTHER CONSTANTS  */
/*==================*/
#define PI              32768
#define ENABLE     1
#define DISABLE    0

/*--------------------*/
/* Global Variables   */
/*--------------------*/
extern volatile unsigned encoder_position[2];

/*----------------------*/
```

```
/* Function Declaration */
/*----------------------*/
extern void Encoder_Init(unsigned rollover);
extern void Encoder_ZeroPosition(void);
extern void Encoder_MatchIncrPhase(void);
extern int  Encoder_CalcPhase(int sin,int ncos);
extern unsigned Encoder_SamplePosition(void);
extern void Encoder_CalcPosition(unsigned qep_incr_sample);

#endif
```

EVM_QEP.H

```
/*==============================================================*/
/* Name:           EVM_QEP.H                                    */
/* Project:        ENCODER                                      */
/* Originator:     Martin Staebler                              */
/* Description:    QEP module functions header                  */
/*==============================================================*/

#ifndef __EVM_QEP_H_
#define __EVM_QEP_H_

extern volatile unsigned  qep_rollover;
extern volatile int       qep_diff;

unsigned QEP_GetIncr(void);
void     QEP_Init(unsigned zero_position, unsigned rollover);

#endif
```

Q15_ATAN.H

```
/*==============================================================*/
/* Name:           Q15_ATAN.H                                   */
/*==============================================================*/
  int q15_atan(int);
 -----------------------------------------------------------------
  Function:     arcus tanges, for unsigned fractional q15 format

  Arguments:    unsigned fractional q15
                min: 0      <--> 0000h
                max: 0.9999 <--> 7FFFh

  Return value: scaled angle (0.. PI/4)
                scaling: PI  (e.g. atan(1.0) = 0.25 or 2000h)

  Error:         < 2 LSB  (128 point lookup table)
 ----------------------------------------------------------------*/

#ifndef  __Q15_ATAN_H_
#define  __Q15_ATAN_H_

unsigned q15p_atan(unsigned value); /* dito, but only positive q15 */
#endif
```

Q15_DIV.H

```
/*-----------------------------------------------------------------
  unsigned _q15_div(unsigned nom,unsigned denom)
 ------------------------------------------------------------------
  Function:     positive fractional q15 division
                range from 0000h .. 7FFFF

  Arguments:    unsigned nominator:    q15 [0000h - 7FFFFh]
                unsigned denominator:  q15 > nominator
                --> !!! check beforehand  !!!

  Return value:  dividend q15 <--> 0000h - 7FFFh
 ---------------------------------------------------------------*/

#ifndef __Q15_DIV_H_
#define __Q15_DIV_H_

unsigned q15_div(unsigned nom, unsigned denom);
#endif
```

ADC.H

```
/*==============================================================*/
/* Name:           ADC.H              */
/* Project:        ENCODER                                      */
/* Originator:     Martin Staebler          */
/* Description:    ADC Converter 'C' Marcos       */
/*==============================================================*/

#ifndef __ADC_H_
#define __ADC_H_

/*---------------*/
/* Include files */
/*---------------*/
#include        "c240.h"

/*--------*/
/* Macros */
/*--------*/

/*---------------------------------------*/
/* select ADC0/1 channel and start ADC's */
/*---------------------------------------*/
#define ADC_START(ch_0_7,ch_8_15)          \
        {                                  \
           ADCTRL1 &= 0xFF81;              \
           ADCTRL1 |= ( (ch_0_7<<1) | ((ch_8_15-8)<<4) | 0x0001); \
        }

/*---------------------*/
/* read ADC0 & 1 value */
/*---------------------*/
#define ADC_READ2(val0,val1)                 \
        {                                    \
           while(!(ADCTRL1 & 0x0100));       \
           val0 = ADCFIFO1;                  \
           val1 = ADCFIFO2;                  \
           ADCTRL1 |= 0x0100;                \
        }

#endif
```

C240.H

```
/*=============================================================*/
/* Name:           C240.H                                      */
/* Project:        C240.LIB                                    */
/* Description:    TMS320C240 memory mapped registers          */
/*=============================================================*/

#ifndef __C240_H_
#define __C240_H_

/******************************************************************/
/* Memory Mapped Register                                         */
/******************************************************************/
#define IMR  *(volatile unsigned int*) 0x0004
#define IFR  *(volatile unsigned int*) 0x0006


/******************************************************************/
/* Watchodg and Real time Interrupt Control Registers             */
/******************************************************************/
#define RTICNTR *(volatile unsigned int*) 0x7021
#define WDCNTR  *(volatile unsigned int*) 0x7023
#define WDKEY   *(volatile unsigned int*) 0x7025
#define RTICR   *(volatile unsigned int*) 0x7027
#define WDCR    *(volatile unsigned int*) 0x7029

/******************************************************************/
/* PLL Clock Registers                                            */
/******************************************************************/
#define CKCR0   *(volatile unsigned int*) 0x702B
#define CKCR1   *(volatile unsigned int*) 0x702D

/******************************************************************/
/* Output Logic                                                   */
/******************************************************************/
#define OCRA    *(volatile unsigned int*) 0x7090
#define OCRB    *(volatile unsigned int*) 0x7092
#define PADATDIR *(volatile unsigned int*) 0x7098
#define PBDATDIR *(volatile unsigned int*) 0x709A
#define PCDATDIR *(volatile unsigned int*) 0x709C

/******************************************************************/
/*     Definitions for SCI Module                                 */
/******************************************************************/
#define SCICCR   *(volatile unsigned int*)0x7050   /* SCI comms.
                                                  ctrl. reg */
#define SCICTL1  *(volatile unsigned int*)0x7051   /* SCI control
                                                  register */
#define SCIHBAUD *(volatile unsigned int*)0x7052   /* Baud rate
                                                  select MSB */
#define SCILBAUD *(volatile unsigned int*)0x7053   /* Baud rate
                                                  select LSB */
#define SCICTL2  *(volatile unsigned int*)0x7054   /* Xmit int. ctrl
                                                  & status reg*/
```

```
#define SCIRXST  *(volatile unsigned int*)0x7055   /* RCV ctrl and
                                                  status reg*/
#define SCIRXEMU *(volatile unsigned int*)0x7056   /* Receiver data
                                                  buffer   */
#define SCIRXBUF *(volatile unsigned int*)0x7057   /* Transmit data
                                                  buffer   */
#define SCITXBUF *(volatile unsigned int*)0x7059  /* Transmit data
                                                  buffer*/
#define SCIPC2   *(volatile unsigned int*)0x705E   /* Port ctrl
                                                  register #2  */
#define SCIPRI   *(volatile unsigned int*)0x705F   /* Int. priority
                                                  ctrl reg */


/*******************************************************************/
/*     Definitions for ADC Module                                  */
/* structures are computed uncorrectly by the C Compiler v6.60 !  */
/* therefore following approach has to be used                     */
/******************************************************************/
#define ADCTRL1  *(volatile unsigned int*) 0x7032   /* ADC Control
                                                      reg 1 */
#define ADCTRL2  *(volatile unsigned int*) 0x7034   /* ADC Control
                                                      reg 2 */
#define ADCFIFO1 *(volatile unsigned int*) 0x7036   /* ADC1  result
                                                      FIFO */
#define ADCFIFO2 *(volatile unsigned int*) 0x7038   /* ADC2 result
                                                      FIFO */


/*******************************************************************/
/*      Definitions for EV Module                                  */
/* structures are computed uncorrectly by the C Compiler v6.60 !  */
/*******************************************************************/
#define GPTCON  *(volatile unsigned int*) 0x7400
#define T1CNT   *(volatile unsigned int*) 0x7401
#define T1CMPR  *(volatile unsigned int*) 0x7402
#define T1PR    *(volatile unsigned int*) 0x7403
#define T1CON   *(volatile unsigned int*) 0x7404
#define T2CNT   *(volatile unsigned int*) 0x7405
#define T2CMPR  *(volatile unsigned int*) 0x7406
#define T2PR    *(volatile unsigned int*) 0x7407
#define T2CON   *(volatile unsigned int*) 0x7408
#define T3CNT   *(volatile unsigned int*) 0x7409
#define T3CMPR  *(volatile unsigned int*) 0x740A
#define T3PR    *(volatile unsigned int*) 0x740B
#define T3CON   *(volatile unsigned int*) 0x740C

#define COMCON  *(volatile unsigned int*) 0x7411
#define ACTR    *(volatile unsigned int*) 0x7413
#define SACTR   *(volatile unsigned int*) 0x7414
#define DBTCON  *(volatile unsigned int*) 0x7415
#define CMPR1   *(volatile unsigned int*) 0x7417
#define CMPR2   *(volatile unsigned int*) 0x7418
#define CMPR3   *(volatile unsigned int*) 0x7419
#define SCMPR1  *(volatile unsigned int*) 0x741A
#define SCMPR2  *(volatile unsigned int*) 0x741B
#define SCMPR3  *(volatile unsigned int*) 0x741C
```

```
#define CAPCON   *(volatile unsigned int*) 0x7420
#define CAPFIFO  *(volatile unsigned int*) 0x7422
#define CAP1FIFO *(volatile unsigned int*) 0x7423
#define CAP2FIFO *(volatile unsigned int*) 0x7424
#define CAP3FIFO *(volatile unsigned int*) 0x7425
#define CAP4FIFO *(volatile unsigned int*) 0x7426

#define EVIMRA  *(volatile unsigned int*) 0x742C
#define EVIMRB  *(volatile unsigned int*) 0x742D
#define EVIMRC  *(volatile unsigned int*) 0x742E
#define EVIFRA  *(volatile unsigned int*) 0x742F
#define EVIFRB  *(volatile unsigned int*) 0x7430
#define EVIFRC  *(volatile unsigned int*) 0x7431
#define EVIVRA  *(volatile unsigned int*) 0x7432
#define EVIVRB  *(volatile unsigned int*) 0x7433
#define EVIVRC  *(volatile unsigned int*) 0x7434

#endif
```

INTERNET

www.ti.com

Register with TI&ME to build custom information pages and receive new product updates automatically via email.

TI Semiconductor Home Page
http://www.ti.com/sc

TI Distributors
http://www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

US TMS320

Hotline	(281) 274-2320
Fax	(281) 274-2324
BBS	(281) 274-2323
email	dsph@ti.com

Americas

Phone	+1(972) 644-5580
Fax	+1(972) 480-7800
Email	sc-infomaster@ti.com

Europe, Middle East, and Africa

Phone	
Deutsch	+49-(0) 8161 80 3311
English	+44-(0) 1604 66 3399
Francais	+33-(0) 1-30 70 11 64
Italiano	+33-(0) 1-30 70 11 67
Fax	+33-(0) 1-30-70 10 32
Email	epic@ti.com

Japan

Phone	
International	+81-3-3457-0972
Domestic	+0120-81-0026
Fax	
International	+81-3-3457-1259
Domestic	+0120-81-0036
Email	pic-japan@ti.com

Asia

Phone	
International	+886-2-3786800
Domestic	
Australia	1-800-881-011

Asia (continued)

TI Number	-800-800-1450
China	10811
TI Number	-800-800-1450
Hong Kong	800-96-1111
TI Number	-800-800-1450
India	000-117
TI Number	-800-800-1450
Indonesia	001-801-10
TI Number	-800-800-1450
Korea	080-551-2804
Malaysia	1-800-800-011
TI Number	-800-800-1450
New Zealand	+000-911
TI Number	-800-800-1450
Philippines	105-11
TI Number	-800-800-1450
Singapore	800-0111-111
TI Number	-800-800-1450
Taiwan	080-006800
Thailand	0019-991-1111
TI Number	-800-800-1450

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current and complete. TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements. Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office. In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards. TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.