TMS320 DSP
DESIGNER'S NOTEBOOK

Obtaining Absolute Encoder Position on a TMS320C240

APPLICATION BRIEF: SPRA279

David M. Alter
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
July 1997

![Texas Instruments logo] **TEXAS INSTRUMENTS**

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.

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.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Examples

Obtaining Absolute Encoder Position on a TMS320C240

Abstract

Many servo applications require absolute position information as feedback. This document describes how the encoder count can be processed to obtain absolute position on the TMS320C240. The document includes a theoretical discussion, equations, timing diagrams and a lengthy code example.

Design Problem

How can the encoder count be processed to obtain absolute position on the TMS320C240?

Solution

General Issues: Many servo applications require absolute position information as feedback. Two separate issues must be addressed when using an encoder sensor to obtain this. The first issue stems from a difference between the desired number sequence and the sequence implemented by the 16-bit counter holding the encoder count on the 'C240. Consider for example the sequence from an 8-count-per-revolution encoder, as shown in Figure 1. The hardware counter will only wrap at the FFFFh to 0000h boundary. However, the desired number sequence requires a wrap that corresponds to the number of counts per revolution, in this case 8h. Three methods to overcome this problem will be presented in this technical note.

The second issue involves obtaining absolute, rather than relative (i.e. incremental) position. Interpreting the position count sequence as absolute requires that it be referenced to some known physical encoder angle. This can be achieved in one of two ways. The first approach involves physically locking the encoder at some known angle during startup. A zero can then be written to the encoder counter register thereby referencing the position to the locked position. In permanent magnet motor applications, the entire process can be fully automated in software by statically activating one of the motor commutation paths and waiting for the rotor to rotate into position. This simple method has the advantage of not needing any additional hardware connections. The code required to perform this is straightforward although it does vary depending on the motor application and configuration. It therefore will not be discussed further in this note. The second approach relies on accurate knowledge of the physical angle at which the encoder is mounted. In this case, one can use the encoder index signal, which pulses once per revolution, to trigger a calibration procedure for the encoder count. Various implementations of this second method will be discussed in more detail in the following sections.

Figure 1. Desired and Actual Encoder Number Sequencing for an 8 Count/Revolution Encoder

Desired Binary Sequence (unsigned)   **Actual 16-bit Hardware**
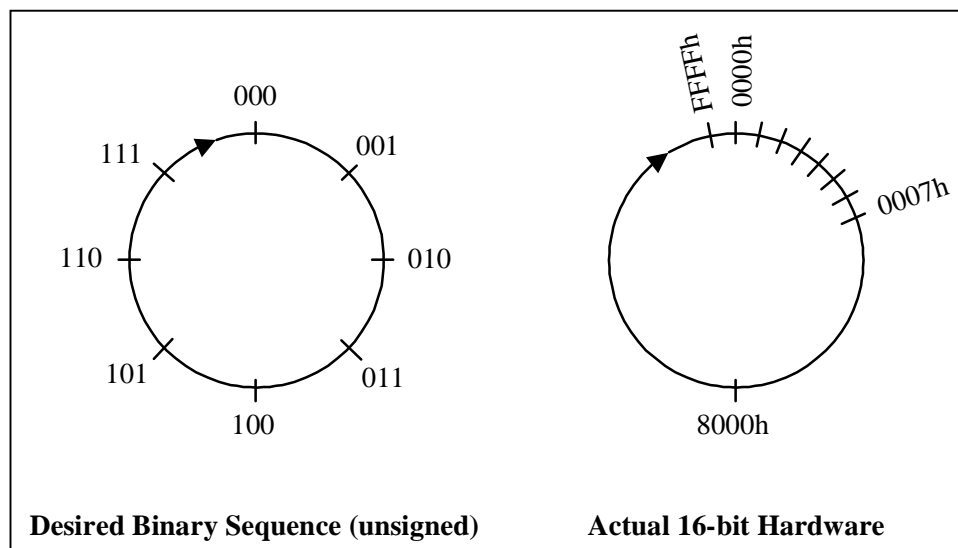
**Method 1: 2^n encoder resolution**

This method requires an encoder with 2^n counts/revolution, where n is an integer. The key is to notice that the hardware counter will adhere to the desired number sequence in the least significant n bits of the count, as illustrated in Figure 2. Software must mask out the upper 16-n unwanted bits. The following code segment illustrates the process on the 'C240 using GP Timer 3 to hold the counts from an 8 count/revolution encoder (n = 3).

```
T3CNT .set     7409h    ;GP Timer 3 counter register
      .bss     position,1 ;position variable

      .text
      LDP      #232 ;data page set to event manager
      LACC     T3CNT    ;ACC = count
      AND      #111b    ;mask off unwanted upper bits
      LDP      #position  ;data page set
      SACL     position   ;store position to memory
```

One can easily modify this code for more realistic encoder resolutions. For example, with a 1024 count/revolution encoder (n=10) the AND instruction should be changed to read `AND #2ffh`. An additional feature of this method is the ability to discriminate positions over more than one revolution by masking off fewer bits (e.g. 0 and 360 degrees will give different count values). This is often required in servo applications. For example, to obtain position over 2 complete revolutions of the example 8 count encoder, replace `AND #111b` with `AND #1111b`.

Figure 2. Binary number sequence of counter for a 2^n encoder with n=3
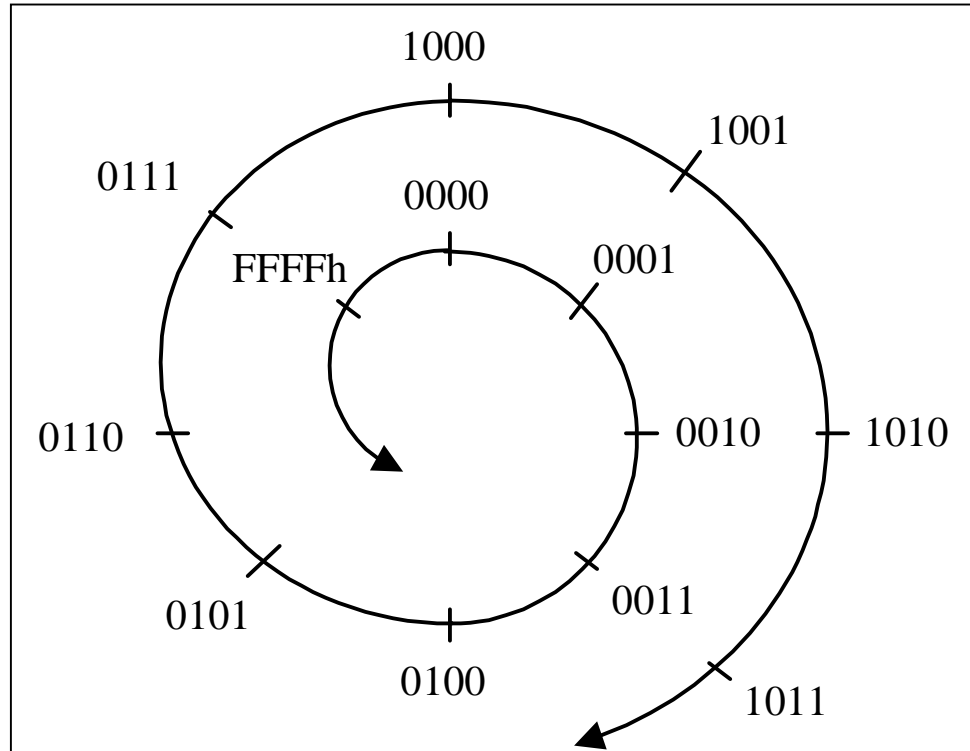
The encoder index signal can be used to reference the absolute position by connecting it to one of the external interrupt pins on the DSP (e.g. XINT1, XINT2, or XINT3). During the startup sequence, the motor should be slowly rotated in some controlled fashion. When the index pulse triggers an interrupt, the GP Timer holding the encoder count is zeroed in the interrupt service routine (e.g. T2CNT or T3CNT registers). After zeroing, the external interrupt should be disabled (i.e. masked) so that calibration is only performed once. The position will now be referenced to the physical angle at which the index pulse occurred. Interrupt latency is normally not a problem. For example, a 1024-count encoder rotating at 1-revolution-per-second has 0.977 ms between counts. This allows for over 19,500 instruction cycles on a 20 MIPS DSP!

Methods 2 and 3 are presented next for non-2^n encoder resolutions. Although they can be used with encoders of 2^n resolution, they are inefficient when compared with method 1.

Method 2: Non-2^n encoder resolution, discrimination over a single rotation

This method can only discriminate position over a single rotation. The encoder index signal is connected to Capture #3, and the CAPCON register configured so that Capture #3 records the count of the GP Timer being used to hold the encoder count. This will occur once every 360-degree rotation of the encoder. In addition, the Capture #3 interrupt is enabled and its service routine is designed to copy the captured value off the CAP3FIFO results stack and store it in some data memory location (e.g. call it position_ref). When the encoder count is processed, position_ref is first subtracted from it. If the resultant is positive, processing is complete. If the resultant is negative, the number of counts per revolution should be added. These operations result in a desired number sequence similar to that shown in Figure 1 but with a different number of counts per revolution, and referenced to the index pulse position. Note that Capture #4 could be used instead of Capture #3 if desired. The following code segments illustrate the process on the 'C240 with the index signal connected to Capture #3, and GP Timer 3 holding the counts for a 9 count/revolution encoder.

Example 1. Code Listing #1

```
T3CNT             .set  7409h      ;GP Timer 3 counter register
CAP3FIFO          .set  7425h      ;Capture #3 results stack
EVIFRC            .set  7431h      ;Event Manager Interrupt Flag
                                   ; Reg C

                  .bss  position,2,1  ;position
position_ref      .set  position+1    ;position reference
count_rev         .set  9             ;counts per revolution


********************************************
*      Encoder Processing Code Segment
********************************************
                  .text
                  CLRC  OVM        ;overflow mode off
                  SETC  SXM        ;use signed math
                  LDP   #232       ;DP set to event manager
                  LACC  T3CNT      ;ACC = count
                  LDP   #position  ;data page set
                  SUB   position_ref ;subtract stored
                                   ; capture value
                  BCND  STORE, GEQ ;branch if positive
                  ADD   #count_rev ;add counts per
                                   ; revolution
STORE:            SACL  position   ;store position to memory


********************************************
*     Capture #3 Interrupt Service Routine
********************************************
```

```
STATUS             .usect   "BLOCKB2", 2       ;must be located on
                                               ; data page 0


                   .text
cap3int:           SST      #0, STATUS         ;save ST0
                   SST      #1, STATUS+1       ;save ST1
                   LDP      #232               ;DP set to event manager
                   SPLK     #0100b, EVIFRC     ;clear CAP3INT flag
                   BLDD     CAP3FIFO, #position_ref
                                  ;save captured value to data memory
                   LDP      #0                 ;set data page
                   LST      #1, STATUS+1       ;restore ST1
                   LST      #0, STATUS         ;restore ST0
                   CLRC     INTM               ;global interrupts
                   RET                         ;return
```

Several points are worth noting about the code. First, the position obtained is inherently absolute in that it is always referenced to the index pulse absolute position. No further steps need be taken to provide for this. Second, the CAP3INT service routine performs the required context save-and-restore using direct addressing. Therefore, the uninitialized data section "BLOCKB2" must be linked to data page 0 since SST always saves to this page when using direct addressing. Also, the use of the BLDD instruction in the Capture #3 interrupt service routine rather than a LACC and SACL combination eliminates the need to context save and restore the accumulator. Finally, note that the encoder processing code segment of Method 2 is more computationally intensive than Method 1, although it is still arguably of negligible length. However, additional CPU cycles are needed once per revolution to execute the Capture #3 interrupt service routine.

Method 3: Non-2^n encoder resolution, discrimination over multiple rotations

Unlike method 2, this method can discriminate position over multiple rotations. The encoder is used in an incremental fashion, where the change in position from the last sample period is computed as the difference of the new count minus the old count, i.e.:

position = position + (new_count – old_count).

The old_count is then replaced with the new_count in preparation for the next sample period, i.e.:

`old_count = new_count.

Finally, the position value must be manually rolled over in order to obtain the desired number sequence. Figure 3 illustrates the adjustment for a 9-count-per-revolution encoder.

Figure 3. Software rollover adjustment for a 9 count/revolution encoder
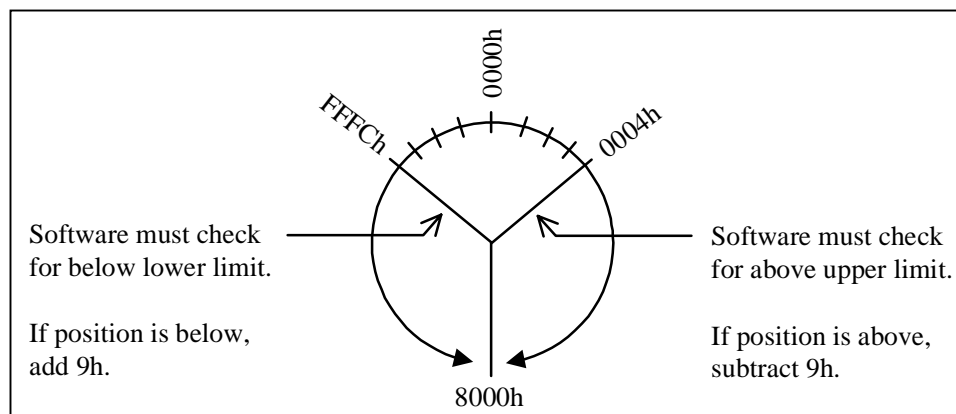
The following code illustrates this method using GP Timer 3 to hold the counts from a 9-count-per-revolution encoder.

Example 2. Code Listing #2

```
T3CNT               .set     7409h
CAP3FIFO            .set     7425h
EVIFRC              .set     7431h
EVIMRC              .set     742Eh


                    .bss     position,3,1
temp                .set     position+1
old_count           .set     position+2

                    .data
count_rev           .int     9        ;encoder counts per revolution
upper_lim           .int     4        ;upper limit of desired sequence
lower_lim           .int     -4       ;lower limit of desired sequence


*********************************************
* Encoder Processing Code Segment
*********************************************
                    .text
;position = position + (new_count - old_count)
                    CLRC     OVM         ;overflow mode off
                    SETC     SXM         ;use signed math
                    LDP      #232        ;data page set to event manager
                    LACC     T3CNT       ;ACC = new_count
                    LDP      #position   ;set data page
                    SACL     temp        ;temporarily store new count
                    SUB      old_count   ;subtract old count
                    ADD      position    ;add previous position
                    SACL     position    ;store new position to memory
                    LACC     temp        ;ACC = new_count
                    SACL     old_count   ;store new_count as old_count
;check for rollover and adjust position if needed
                    LACC     position    ;ACC = new position
```

```
                 LDP       #upper_lim     ;set data page
                 SUB       upper_lim      ;subtract upper limit
                 BCND      over, GT       ;branch if above upper limit
                 LDP       #position      ;set data page
                 LACC      position       ;ACC = new position
                 LDP       #lower_lim     ;set data page
                 SUB       lower_lim      ;subtract lower limit
                 BCND      done, GEQ      ;done if above lower limit
under:           ADD       lower_lim      ;ACC = new position
                 ADD       count_rev      ;add count_rev
                 LDP       #position      ;set data page
                 SACL      position       ;store adjusted position
                 B         done           ;done
over:            ADD       upper_lim      ;ACC = new position
                 SUB       count_rev      ;subtract count_rev
                 LDP       #position      ;set data page
                 SACL      position       ;store adjusted position
done:                                     ;main code continues


*********************************************
*      Capture #3 Interrupt Service Routine
*********************************************
STATUS           .usect    "BLOCKB2", 2     ;must be located on DP 0
                 .bss  CONTEXT, 2, 1    ;context saving area

                 .text
cap3int:         SST       #0, STATUS     ;save ST0
                 SST       #1, STATUS+1   ;save ST1
                 LDP       #CONTEXT       ;set data page
                 SACH      CONTEXT        ;save high accumulator
                 SACL      CONTEXT+1      ;save low accumulator
                 LDP       #232           ;DP set to event manager
                 LACC      CAP3FIFO       ;ACC = captured value
                 LDP       #position      ;set data page
                 SACL      old_count      ;init old_count with
                                          ; captured value
                 SPLK      #0, position   ;zero the position
                 LACC      EVIMRC         ;ACC loaded with
                                          ; EV Mask Register C
                 AND       #1011b         ;mask off CAP3INT bit
                 SACL      EVIMRC         ;disable CAP3INT
                 SPLK      #0100b, EVIFRC ;clear CAP3INT flag
                 LDP       #CONTEXT       ;set data page
                 LACL      CONTEXT+1      ;restore low accumulator
                 ADD       CONTEXT,16     ;restore high accumulator
                 LDP       #0             ;set data page
                 LST       #1, STATUS+1   ;restore ST1
                 LST       #0, STATUS     ;restore ST0
                 CLRC      INTM           ;re-enable global interrupts
                 RET                      ;return
```

Absolute position referencing has been accomplished through proper initialization of `position` and `old_count`. The above code uses the encoder index signal in conjunction with Capture #3 to perform the initialization in the Capture #3 interrupt service routine. During the interrupt service routine, `position` is set to zero and `old_count` is initialized with the captured count value. The Capture #3 interrupt is then disabled prior to returning to the main code so that encoder initialization is only performed once.

The code is easily modified for other encoder resolutions and for position discrimination over multiple revolutions by changing the assigned values for `count_rev`, `upper_lim`, and `lower_lim`. The assigned values must satisfy (`N*count_rev - 1`) = (`upper_lim - lower_lim`), where N equals the number of revolutions. For example, to obtain position over two complete revolutions of the example 9 count/revolution encoder, one could set `upper_lim` to 8 and `lower_lim` to -9. Alternately, one could set `upper_lim` to 9 and `lower_lim` to −8.

Note that `position` must never cross the boundary at 8000h in either direction. Normally, this is not a problem except in the uncommon situation where `upper_lim` and `lower_lim` begin to approach 8000h, or when the sample rate is unusually low so that `new_count` minus `old_count` becomes large. The code would need to be modified if crossing this boundary becomes a possibility.

Obtaining Signed Position:

The desired number sequence shown in Figure 1 is an unsigned sequence representing 0 to 360 degrees, as opposed to a signed sequence representing −180 to +180 degrees. As illustrated in this paper, Methods 1 and 2 always produce such an unsigned sequence, while Method 3 can directly produce either a signed or an unsigned sequence depending on the values assigned to `upper_lim` and `lower_lim`. If a signed sequence is desired when using Methods 1 or 2, first obtain the unsigned sequence as previously shown. A 2's compliment sequence that is properly sign-extended to 16 bits can then be obtained by subtracting half the number of counts per revolution from the unsigned position. For example, with a 1024 count/revolution encoder subtract 512 = 200h. With a 1000 count/revolution encoder subtract 500 = 1F4h. Figure 4 depicts a subtraction of 4 with an 8-count-per-revolution encoder. Note that the absolute position is now referenced to a position 180 degrees away from the original reference position.

When discriminating position over multiple revolutions, the subtracted value should equal half the total number of counts for the multiple revolutions. For example, when discriminating four revolutions with an 8-count-per-revolution encoder, the subtracted value should be 16 = 10h.

Figure 4. Position Sequence After Sign Adjustment for an 8 Count/Revolution Encoder

Unsigned Sequence   **Signed Sequence after Subtracting 4h**