TMS320 DSP
DESIGNER'S NOTEBOOK
Number 83
TEXAS INSTRUMENTS

Using the Capture Units for Low Speed Velocity Estimation on a TMS320C240

Contributed by David Alter

Design Problem

How are the capture units used for low speed velocity estimation on the TMS320C240?

Solution

General Issues: Estimating velocity from a digital position sensor is a cost-effective strategy in motor control. Two different first order approximations for velocity may be written as:

$$v(k) \cong \frac{x(k) - x(k-1)}{T} \qquad \text{Equation 1}$$

$$v(k) \cong \frac{X}{t(k) - t(k-1)} \qquad \text{Equation 2}$$

where v is velocity, x is position, t is time, T is a fixed sampling period, X is a fixed position interval, and k is the discrete-time index. Equation (1) is the conventional approach to velocity estimation and can be implemented on the TMS320C240 by, for example, a glueless interface between the on-chip QEP logic and a quadrature encoder. The encoder count (position) is read at the beginning of each velocity loop calculation, the difference x(k) - x(k-1) formed, and a new velocity estimate computed by multiplying by the known constant 1/T.

Estimation based on equation (1) has an inherent accuracy limit directly related to the resolution of the position sensor and the sampling period T. For example, consider a 500 count/revolution quadrature encoder with a velocity loop sample rate of 400 Hz. When used for position the quadrature encoder gives a fourfold increase in resolution, in this case 2000 counts/rev. The minimum rotation that can be detected is therefore 0.0005 revolutions, which gives a velocity resolution of 12 rpm when sampled at 400 Hz. While this resolution may be satisfactory at moderate or high speeds, e.g. 1% error at 1200 rpm, it would clearly prove inadequate at low speeds. In fact at speeds below 12 rpm, the speed estimate would erroneously be zero much of the time.

At low speed, equation (2) provides a more accurate approach. It requires a position sensor that outputs a fixed interval pulse train, such as the aforementioned quadrature encoder. The width of each pulse is defined by the sensor resolution, and by measuring the elapsed time between successive pulse edges, equation (2) can be implemented on a DSP by performing a division. Note that the velocity estimate is no longer updated at

regular time intervals, but rather the update rate is proportional to the motor speed. The accuracy of this method is directly related to both the counter bit-width and the speed of the motor. At the minimum designed for speed (to be discussed later), the 16-bit timers on the TMS320C240 give an achievable accuracy of 1 part in 2^{16}, or 0.0015%! However, this method suffers from the opposite limitation as does equation (1). A combination of relatively large motor speeds and high sensor resolution makes the time interval t(k) - t(k-1) small, and thus more greatly influenced by the timer resolution. This can introduce considerable error into high speed estimates.

This technical brief presents the various design issues associated with implementing equation (2) on a TMS320C240 DSP, gives a design example, and also provides example code for an interrupt driven velocity estimator based upon the example design.

TMS320C240 Hardware Issues: The (time) width of the incoming position pulse is recorded by a capture unit working with either GP Timer 2 or 3. The selected 16-bit timer is run in the continuous-up count mode, which allows automatic rollover from 0xFFFF to 0x0000 on overflow. The correct value for t(k) - t(k-1) will be obtained only if no more than 65,535 counts (2^{16} - 1) have occurred between t(k) and t(k-1). The time base prescaler for the selected timer must therefore be carefully chosen based upon the expected minimum motor velocity and the known position interval X.

Sensor Issues: The position interval X is solely determined by the sensor resolution. It is advantageous to have X relatively small so that t(k) - t(k-1) (in timer counts) does not overflow 16-bit width. However, X must not be too small relative to the GP Timer clock rate or t(k) - t(k-1) will be small and more greatly influenced by the timer resolution. A quadrature encoder can be utilized so as to provide any of three different values for X, as shown in Figure 1. Consider an encoder rated at 500 pulses/rev. The base pulse width is the reciprocal of this, $X_0 = 0.002$ rev. To obtain $X = X_0$, one captures the time between two successive rising edges (or two successive falling edges) on a single encoder channel. Alternately, by capturing the time between successive rising and falling edges on a single channel, one can reduce this value by half: $X = X_0/2 = 0.001$ rev. Finally, since the two encoder channels are 90 degrees apart in phase, one can also capture the time between successive edges on alternating channels. This yields $X = X_0/4 = 0.0005$ rev. This last approach requires the use of two of the capture units on the TMS320C240, one for each channel.

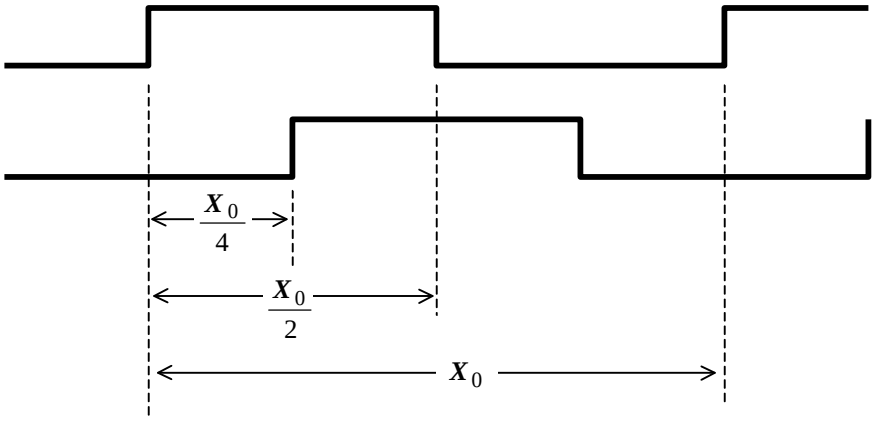

Figure 1: Position interval determination from quadrature encoder output

Motor Speed Issues: The motor speeds for which the estimation calculation will produce valid results are bounded by both upper and lower limits. These limits are determined by the pulse width X, the GP Timer prescale value, and the numerical scaling employed in the software. The lower limit is the speed where a rotation of X revolutions takes exactly 65,535 GP timer counts, as discussed previously. The upper limit results from the nature of fractional division. On the TMS320C240, division is performed using multiple conditional subtraction instructions (SUBC). The number of SUBC instructions executed determines the Q format by which the quotient should be interpreted, and the maximum allowable value for this Q format is the upper speed limit (see Appendix for a Q format review). The maximum value of the quotient must be determined a priori, and a Q format selected (and coded for) that accomodates this worst case. A trade-off exists between speed range and accuracy. For example, selecting Q15 format allows a maximum speed value of ~1. Q14 format allows values up to ~2, but will be one bit less accurate than Q15 for values less than 1. One approach to designing a system with a large speed range is to keep the upper estimation limit relatively small so that good accuracy is obtained at low speed, and have the software switch to equation (1) when speeds reach higher values. An encoder can easily meet the needs of both equations on the TMS320C240 by connecting the two encoder channels to Capture Units 1 and 2, and switching between QEP decode and capture functions as needed.

Design Example: A particular control application must meet the following requirements:

- quadrature encoder resolution = 500 pulses/rev
- minimum speed = 1 rpm (0.01667 rps)
- TMS320C240 DSP clock rate = 50 ns (20 MIPS)

For this example, equation (2) will be implemented using a single encoder channel with rising and falling edges captured. Therefore, $X = X_0/2 = 0.001$ rev. as previously discussed. This simplifies the code compared to the $X_0/4$ implementation since it requires only a single capture unit. At the minimum speed of 1 rpm, the pulse width will be (0.001 rev) / (0.01667 rev/s) = 0.06 sec. With a 50 ns DSP clock, 1,200,000 clock counts will occur for each pulse. This exceeds the 65,535 count limit, and therefore a suitable prescaler should be used to clock the desired GP timer. A minimum prescale of 18.3 is needed (1,200,000 / 65,535). A prescale of 32 is selected on the TMS320C240 since that is the nearest available larger prescale. Equation (2) can now be represented numerically as:

$$v \cong \frac{X}{counts * prescale * CPUCLK} = \frac{0.001\,\mathbf{rev}}{counts * 32 * 50\,\mathbf{ns}} = \frac{625}{counts}\,\mathbf{rev/s} \qquad \text{Equation 3}$$

where counts is the number of timer counts that occurred during a rotation of magnitude X. Note that this prescale actually allows for a minimum motor speed of 0.572 rpm.

$$v_{\mathbf{min}} = \frac{625}{65535}\,\mathbf{rev/s} = 0.00954\,\mathbf{rev/s} = 0.572\,\mathbf{rpm} \qquad \text{Equation 4}$$

The final step is to select the Q format for *v*, which will define the maximum allowable motor speed. Using the Q15 format, the minimum number of counts that must elapse between captured pulse edges is 626 (maximum Q15 number is just less than 1). Therefore, the maximum velocity becomes 1 rev/s = 60 rpm. The timer error at 60 rpm is essentially one clock count out of 626 (0.16%), or 0.1 rpm. The timer error at 0.572 rpm is one clock count out of 65535 (0.0015%), or 8.6E-6 rpm. Of course, Q15 format only allows 15-bit accuracy for positive numbers, so the minimum achievable error after

the division will actually be 0.003%. Selecting different units for *v* can help scale the problem. For example in radians/sec, *v* = 3927/counts, which allows a maximum speed of 9.55 rpm using the Q15 format.

Application Code: The following TMS320C240 assembly code implements the velocity estimator for the design example, i.e. equation (3). It utilizes Capture Unit 3 and GP Timer 2. There are three different parts to the code. Section 1 defines some register addresses and allocates memory for variables. Section 2 contains initialization code for GP Timer 2 and Capture Unit 3. It should be incorporated into the initialization routine at the beginning of the main program. In addition to the initializations shown, the main program must also perform the following setup tasks:

1. Set bit 6 in the OCRB register to select the capture function for the CAP3/IOPC6 pin.
2. Set bit 3 in the IMR register to enable core interrupt #4.
3. Clear the INTM bit in the ST0 register to enable global interrupts.

Note that the old timer count, t_old, is initialized to zero. Therefore, the first pass through the estimator will produce an erroneous result. This problem can be handled as desired by modifying the code.

Section 3 contains the Capture Unit 3 interrupt service routine that computes the velocity estimation. The interrupt branching sequence, as designed in the main program, should ultimately branch to the starting address of this routine, here labeled "CAP3INT." If none of the other three capture unit interrupts are being used, the interrupt vector at program address 0x0008 can be changed to directly branch here, e.g. B CAP3INT. See "Interrupts" in the *TMS320C24x DSP Controllers Reference Set, Volume 1* for more information on interrupt management.

Example 1. Code Listing

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; Section 1: Definitions and memory allocations
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

T2PR         .set   7407h    ;GP Timer 2 Period Register
T2CON        .set   7408h    ;GP Timer 2 Control Register
CAPCON       .set   7420h    ;Capture Unit Control
CAP3FIFO     .set   7425h    ;Capture Channel 3 FIFO Top
EVIMRC       .set   742Eh    ;Group C Int. Mask Register
EVIFRC       .set   7431h    ;Group C Int. Flag Register

             .bss   t_new,5,1    ;new timer count
t_old        .set   t_new+1      ;old timer count
delta_t      .set   t_new+2      ;change in timer count
dividend     .set   t_new+3      ;dividend of velocity
                                 ; calculation
velocity     .set   t_new+4      ;velocity

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; Section 2: GP Timer 2 and Capture 3 Initializations
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

;setup GP TIMER 2 for CAPTURE time base
             LDP    #232            ;DP set for 7400-747Fh
             SPLK   #0FFFFh,T2PR    ;set Timer 2 period
                                    ; register

             SPLK    #1001010101001000b,T2CON
*                     ||||||||||||||||
*                     FEDCBA9876543210
* bit 15-14  10:    operation not affected by emulation suspend
* bit 13-11  010:   continuous-up count mode
* bit 10-8   101:   prescale = /32
* bit 7      0:     use own TENABLE bit
* bit 6      1:     enable timer
* bit 5-4    00:    clock source = CPUCLK
* bit 3-2    10:    reload timer compare immediately
* bit 1      0:     disable timer compare
* bit 0      0:     use own period register

;setup CAPTURE #3
             SPLK   #1001000000001100b,CAPCON
*                    ||||||||||||||||
*                    FEDCBA9876543210
* bit 15     1:     no action
* bit 14-13  00:    disable CAP1-2 and QEP
* bit 12     1:     enable CAP3
* bit 11     0:     disable CAP4
* bit 10     0:     CAP3-4 use TIMER 2
* bit 9      0:     CAP1-2 use TIMER 2
* bit 8      0:     CAP4 does not start ADC
* bit 7-6    00:    CAP1 no detection
* bit 5-4    00:    CAP2 no detection
* bit 3-2    11:    CAP3 both edges
* bit 1-0    00:    CAP4 no detection

;enable CAP3INT in EVIMRC          ;load ACC with mask
             OR     EVIMRC         ;OR with EV mask register C
             SACL   EVIMRC         ;save EV mask register C

;initialize relevant variables
             LDP    #t_new         ;set data page
             SPLK   #625,dividend  ;initialize dividend
             SPLK   #0,t_old       ;initialize old count
```

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; Section 3: Capture 3 Interrupt Service Routine
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

STATUS4      .usect "BLOCKB2",2     ;must be linked to DP 0
CONTEX4      .usect "CONTEXT",2,1   ;link anyplace

             .text
CAP3INT:

;context save
             SST    #0, STATUS4     ;store ST0
             SST    #1, STATUS4+1   ;store ST1
             LDP    #CONTEX4        ;set data page
             SACH   CONTEX4         ;save ACCH
             SACL   CONTEX4+1       ;save ACCL

;compute delta_t
             LDP    #232               ;data page set to
                                       ; event manager regs
             SPLK   #0100b,EVIFRC      ;clear CAP3INT
                                       ; flag in EVIFRC reg
             LACC   CAP3FIFO           ;read new time
             LDP    #t_new             ;set data page
             SACL   t_new              ;store to t_new
             SUB    t_old              ;subtract
             SACL   delta_t            ;store to delta_t
             LACC   t_new              ;move t_new to t_old
             SACL   t_old

;perform the division
             LACC   dividend,16        ;load dividend
             RPT    #14                ;do SUBC 15 times
             SUBC   delta_t
             SACL   velocity           ;store Q15 result

;context restore
             LDP    #CONTEX4           ;set data page
             LACL   CONTEX4+1          ;restore ACCL
             ADD    CONTEX4,16         ;restore ACCH
             LDP    #0                 ;set data page
             LST    #1, STATUS4+1      ;restore ST1
             LST    #0, STATUS4        ;restore ST0
             CLRC   INTM               ;global interrupts
             RET                       ;return from ISR
```

Appendix: Q Format Review

Fractional numbers are represented in the TMS320C240 DSP in a fixed-point 2's complement form called Q format. In Q*n* format, the *n* signifies the number of bits to the right of the binary point. Thus, a 16-bit word has one sign bit, 15-*n* integer bits, and *n* fractional bits. The binary number `1101000000000000b` can be interpreted, for example, as follows:

Q0: $1101000000000000.b = -2^{15} + 2^{14} + 2^{12} = -12288$

Q14: $11.01000000000000b = -2^{1} + 2^{0} + 2^{-2} = -0.75$

Q15: $1.1010000000000000b = -2^{0} + 2^{-1} + 2^{-3} = -0.375$

Q0 is seen equivalent to the standard representation for signed integers. In any 16-bit Q format, the minimum (most negative) value that can be represented is 1000000000000000b (0x8000), and the maximum (most positive) value is 0111111111111111b (0x7FFF). Table 1 shows numerical ranges for three common Q formats.

Table 1. Decimal ranges of fractional formats

	minimum value	maximum value
Q0	-32768	32767
Q14	-2	1.999939
Q15	-1	0.999969