

Il μC LPC2103 utilizzato come contatore up/down e come tachimetro per encoder ottici incrementali

Abstract: Scopo di questa nota applicativa è l'utilizzo del μC LPC2103 di NXP come contatore per encoder ottico incrementale. Mediante l'uso di un canale del timer del μC e di un input digitale, viene realizzato sia il conteggio di tipo up/down, grazie ai due segnali in quadratura dell'encoder ottico, sia il calcolo della velocità di rotazione.

Introduzione - in molte applicazioni per la gestione dei motori (sia che si tratti di controllo di posizione che di controllo di velocità o di accelerazione) risulta necessario avere un anello di controllo a catena chiusa, con l'utilizzo di filtri più o meno complessi (filtri PID) per una regolazione ottimale.

Il metodo più utilizzato per controllare il movimento è basato sull'encoder ottico incrementale collegato all'organo meccanico, con uscita su due canali aventi segnali ad onda quadra sfasati tra loro di 90° (Channel A e Channel B), tali da fornire contemporaneamente sia l'informazione di conteggio che l'informazione di verso di rotazione.

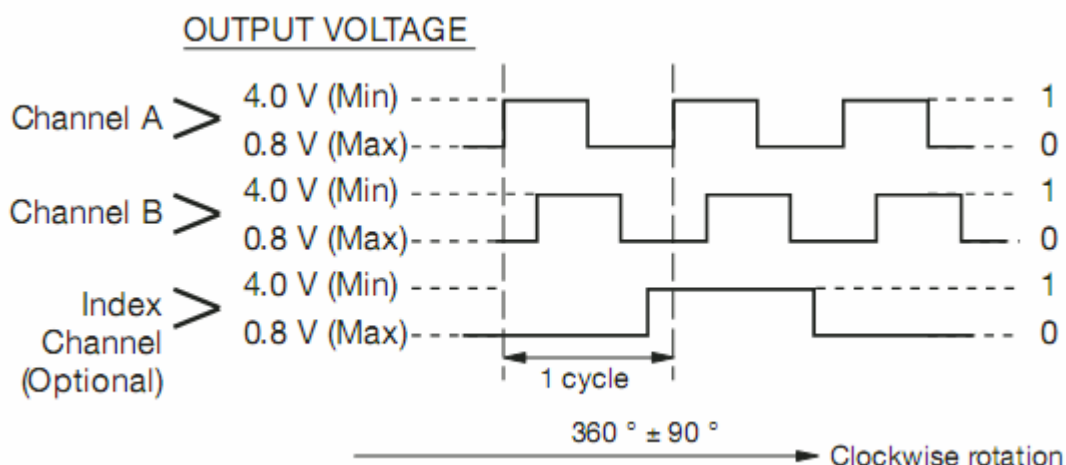


Figura 1

Dalla figura precedente si ricava che se il canale A anticipa di 90° il canale B, la rotazione è in verso orario, viceversa (se il canale B anticipa di 90° il canale A) la rotazione avviene in verso antiorario. Alcuni encoder presentano anche un canale aggiuntivo opzionale (Index) per la segnalazione di rotazione completa (un impulso per ogni rotazione di 360°). I canali A e B, invece, presentano un numero di impulsi per rotazione dipendente dalla risoluzione dell'encoder. Lo scopo dell'applicazione è di contare gli impulsi emessi dall'encoder, verificando contemporaneamente la fase tra i due canali per realizzare un contatore di tipo up/down. Inoltre, mediante la misura di tempo intercorso tra due impulsi successivi si ottiene la misura della velocità di rotazione in impulsi/sec.

Hardware - l'encoder utilizzato è il modello della Bourns con 100 impulsi per ogni rotazione completa (il che determina una risoluzione angolare pari a $360^\circ/100$, ovvero $3,6^\circ$).

Il μC impiegato è lo LPC2103 di NXP, basato su core ARM7. Il timer adoperato è il Timer0 del μC , fornito di registri hardware a 32 bit e in grado, quindi, di registrare conteggi dalla posizione 0 fino alla posizione $2^{32} - 1$ (4294967295), di solito più che sufficiente per un tipico controllo di posizione. In particolare sono stati collegati ai canali B e A il pin *CAP0.1* del *Timer0* ed il pin di I/O

P0.6. Per visualizzare la velocità di rotazione ed il conteggio di posizione è stato utilizzato un LCD 4x16 standard HD44780 collegato con 4 linee dati e 2 linee di controllo.

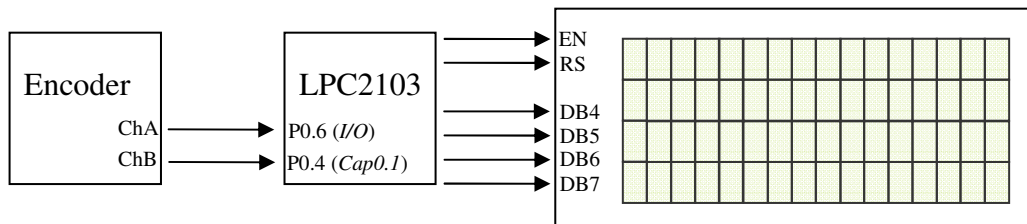


Figura 2

Software - le impostazioni dei registri del μC sono state le seguenti: il *Timer0* è stato impostato in modalità *Timer Mode* (funzionalità di default al reset), il canale *CAP0.1* del *Timer0* è stato collegato all'input *P0.4* e l'azione di cattura è stata sincronizzata con il fronte di discesa del segnale. In tale situazione avviene il caricamento del valore corrente del *Timer0* nel registro *CR1* e l'attivazione dell'interrupt per segnalare tale evento.

Inoltre sono stati attivati gli interrupt relativi al raggiungimento da parte del *Timer0* dei valori caricati nei *Match Register MR0* e *MR1*, utilizzati rispettivamente per aggiornare la misura di velocità in caso di encoder fermo e per segnalare la condizione di overflow del *Timer0*.

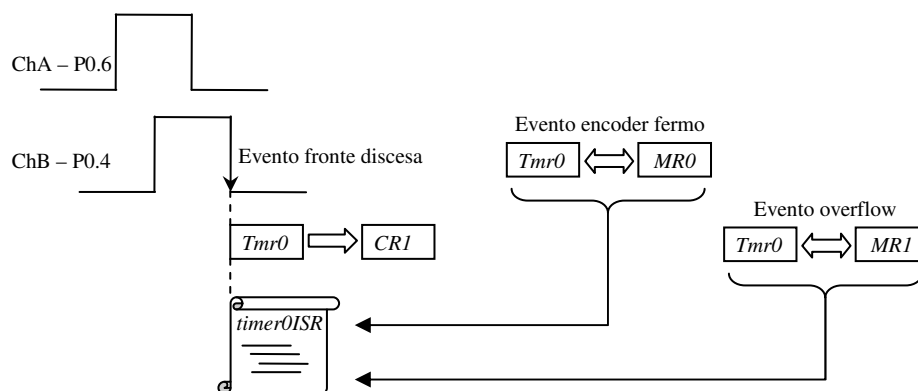


Figura 3

```

PINSEL0 |= (1<<9); // set P0.4 as CAP0.1 (Timer0)

T0CCR |= (1<<4); // capture on CAP0.1 falling edge CR1 = TC
T0CCR |= (1<<5); // interrupt on CAP0.1 event

T0MCR |= 1; // interrupt on MR0 (when MR0 matches TC)
T0MCR |= (1<<3); // interrupt on MR1 (when MR1 matches TC)

T0MR0 = SPEED_UPDT_INT; // MR0 is loaded with FOSC
T0MR1 = 0xFFFFFFFF; // MR1 is loaded with max int val (overflow condition)

enIRQ(TIMERO0_VIC_Ch, timer0ISR, 15); // Timer0 IRQ configuration

T0TCR = TCR_CNT_ENBL; // enable Timer0

```

In fine è stata collegata la sorgente di interrupt del *Timer0* al canale 15 del *VIC (Vectored Interrupt Controller)* con priorità 15 e routine di gestione dell'evento *timer0ISR*.

```
void timer0ISR(void)
{
    unsigned int regValue;

    regValue = TIMER0_IR; // read Timer0 interrupt register
    if(regValue & CR1_INT_MASK) // CR1 interrupt
    {
        T0TCR |= TCR_CNT_RST; // reset
        T0TCR &= ~TCR_CNT_RST; // timer
        encSpeed = SPEED_CONST/T0TCR1; // encoder speed evaluation
        T0MR0 = SPEED_UPDT_INT; // MR0 is re-loaded with SPEED_UPDT_INT

        // read P0.6 (ChA) to compare with CAP 0.1 (ChB) (now ChB is low)
        if(IOPIN & 0x40) // ChA is high
            encPos--; // ChB leads ChA (counterclockwise)
        else // ChA is low
            encPos++; // ChA leads ChB (clockwise)
        TIMER0_IR = regValue; // reset Timer0 interrupt
    }
    else if((regValue&MR0_INT_MASK) && !(regValue&MR1_INT_MASK)) // MR0 interrupt (refresh time)
    {
        T0MR0 += SPEED_UPDT_INT; // MR0 is added with SPEED_UPDT_INT
        encSpeed = SPEED_CONST/T0TC; // encoder speed evaluation (can't use T0TCR1)
        TIMER0_IR = regValue; // reset Timer0 interrupt
    }
    else // MR1 interrupt (overflow condition)
        encSpeed = 0; // speed is zero

    VICVectAddr = 0x00000000; //dummy write to VIC to signal end of interrupt
}
```

La routine di gestione interrupt deve gestire tre possibili cause di origine dell'evento: la presenza di un fronte di discesa su *CAP0.1* ed il conseguente caricamento del *Timer0* in *CR1*, il raggiungimento da parte del *Timer0* del valore caricato nel *Match Register MR0* ed il raggiungimento da parte del *Timer0* del valore caricato nel *Match Register MR1*.

- a) **fronte di discesa su CAP0.1:** l'encoder sta ruotando e (poiché siamo già nella routine di servizio) il valore corrente del *Timer0* è stato già caricato in *CR1*. Possiamo azzerare il *Timer0* e valutare la velocità di rotazione in impulsi/sec facendo il rapporto tra la costante di velocità (frequenza del clock interno) ed il valore del *Timer0* accumulato in *CR1*. Il registro *MR0* viene ricaricato con il valore predefinito per l'aggiornamento di velocità in caso di encoder fermo. In seguito dobbiamo aggiornare la variabile contatore up/down *encPos* valutando la fase relativa tra ChA e ChB; dato che siamo nella routine di servizio ChB è sicuramente basso (sincronismo sul fronte di discesa), quindi basta verificare il valore corrente di ChA per dedurre se siamo in rotazione oraria (ChA basso) o antioraria (ChA alto). Nel primo caso incrementiamo la variabile a 32 bit *encPos*, nel secondo la decrementiamo. In ultimo viene cancellata la flag di interrupt.
- b) **Timer0 uguale a MR0:** in questo caso l'encoder è fermo da un intervallo di tempo proporzionale al valore caricato in MR0 (nel nostro caso 1/10 di secondo) e non siamo in caso di overflow. Sommiamo al valore presente in MR0 un ulteriore intervallo di tempo per

l'aggiornamento successivo della velocità, quindi calcoliamo la velocità di rotazione in impulsi/sec con il rapporto tra la costante di velocità (frequenza del clock interno) ed il valore corrente del *Timer0*. La variabile *encPos* non è modificata e viene comunque cancellata la flag di interrupt.

- c) ***Timer0* uguale a *MR1***: in questo caso l'encoder è fermo da un intervallo di tempo proporzionale al valore caricato in *MR1* (nel nostro caso $2^{32} - 1$, poco meno di 5 min) e siamo in overflow. La velocità viene impostata a zero e la flag di interrupt non viene cancellata.

In tutti i casi precedenti è comunque necessario successivamente azzerare il registro VIC (*Vectored Interrupt Controller*), con una scrittura fittizia per segnalare la fine della routine di servizio interrupt.

IT, Roma, 15/12/07



Questa opera è pubblicata sotto una licenza Creative Commons
<http://creativecommons.org/licenses/by-nc-sa/2.5/it/>