Cika Electrónica

Nota de Aplicación: CAN-027
Título: **Codificadores en cuadratura**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	16/03/05	

Ya sea como alternativa interesante para la interfaz con el usuario, o como eficaz solución al problema de la detección del movimiento y sentido de giro, les proponemos en esta oportunidad la utilización de encoders rotativos, codificadores en cuadratura.

Un encoder rotativo codificador en cuadratura produce, a partir de su giro, un par de señales en cuadratura. La frecuencia de ambas depende de la velocidad de giro, y el sentido se identifica observando el signo de la diferencia de fase. Si bien existen varios tipos de encoders de esta clase, los que proponemos tienen una característica interesante que simplifica notablemente la detección: estos encoders producen un pulso en ambas salidas a cada movimiento del eje entre los descansos, según puede observarse en la hoja de datos. De este modo, podemos utilizar una de las señales como disparador y la otra para detectar el sentido de giro. La técnica convencional consiste en incrementar un contador con el giro en un sentido y decrementarlo con el giro en el otro, de modo de poder utilizar el valor del contador para avanzar en un sistema de menúes o detectar posición del eje, respecto de una referencia; aunque existe otra clase de encoders que resuelve directamente la posición en forma absoluta, entregando código gray o binario correspondiente a la posición de descanso del cursor.
A grandes rasgos, tendremos dos formas de leer el encoder: polling, mediante un handler que periódicamente chequea el estado de una señal y ante cambios verifica la otra; e interrupciones, configurando al micro para ser interrumpido por un flanco de una señal y verificando el estado de la otra. El esquema de interrupciones es bastante más estricto en cuanto a la limpieza de la señal que genera las interrupciones, la cual deberá ser filtrada correctamente a fin de evitar falsos disparos. El esquema de polling es más permisivo, según la frecuencia de operación del handler en cuestión.
Entregamos a continuación unos sencillos ejemplos de uso con Microchip PIC y Rabbit 2000

Microchip

A continuación, presentamos una versión de handler por IOC (interrupt-on-change) para PIC con GPIO e IOC, por ejemplo el 12F629. Encontrará un archivo ejemplo en el software que acompaña a esta nota de aplicación. Por claridad, asumimos que no hay otro dispositivo generando interrupciones; de no ser así, deberá manejar esta situación.

```
counter equ 0x20

                org 0x04

                bcf INTCON,GPIF             ; ACK
                btfss   GPIO,3              ; Flanco
                goto falling
                btfss   GPIO,0              ; A sube, check B
                goto trescero
cerotres        decf counter,f              ; B antecede a A
falling         retfie                      ; descendente, descartar
                goto cerotres
trescero        incf counter,f              ; A antecede a B
                retfie

Main            <inicialización>
                movf GPIO,w                 ; Lee GPIO por primera vez
                clrf counter
                <permite interrupciones>
```

A continuación, presentamos una versión de handler por polling para PIC12F629. Por simpleza, comodidad y claridad, hemos utilizado todo un byte como memoria del estado anterior para la detección de flancos. El avezado lector en desesperada avidez de data memory puede, si así lo desea, emplear algún bit no utilizado de alguna palabra de flags para este propósito.

```
counter equ 0x20
oldgpio equ 0x21


loop:           call encoder
                <haga algo>
                goto loop


encoder         movf GPIO,w                 ; Lee port
                andlw B'00001000'           ; "change-detect" bit
                subwf oldgpio,f             ; Compara
                movwf oldgpio               ; Guarda nuevo estado (preserva Z)
                btfsc STATUS,Z              ; Igual ?
                return                      ; Sí, vuelve
                btfss   GPIO,3              ; No, flanco ?
                goto falling
                btfss   GPIO,0              ; A sube, check B
                goto trescero
cerotres        decf counter,f              ; B antecede a A
                bcf GPIO,1
                bsf GPIO,5
falling         retfie                      ; Descendente, descartar
                goto cerotres
trescero        incf counter,f              ; A antecede a B
                bsf GPIO,1
                bcf GPIO,5
                return
```

Rabbit

En el caso de Rabbit, la literatura recomienda reservar las interrupciones para aquellas tareas que tienen un timing crítico (lo cual no es este caso, por lo general), dado que se trata de un sistema con soporte multitarea fácilmente administrable. No obstante, presentaremos un handler por interrupciones para quien guste utilizarlas. En este caso, deberemos discernir entre R2000C (la última revisión del micro, rotulada IQ5T) y las revisiones anteriores, debido a que Dynamic C corrige automáticamente el problema comentado en TN301
A continuación, presentamos una versión de handler por interrupciones externas para Rabbit 2000:

```
static int counter;

#asm
quadenc_isr::
                push AF                     ; salva registros
                push HL
                push DE
                ld DE,(counter)             ; lee contador (16-bits) (preserva Z)
                ld HL,PBDR                  ; apunta a port B
                ioi bit 0,(HL)              ; A sube, check B
                jr z, AlB
BlA:            dec DE                      ; B antecede a A
                jr qidn
AlB:            inc DE                      ; A antecede a B
                jr qidn
qidn:           ld (counter),DE             ; actualiza contador
                pop DE
                pop HL
                pop af
                ipres                       ; restablece prioridad e interrupciones
                ret                         ; return
#endasm


main()
```

```
{

		WrPortI ( PEDDR,&PEDDRShadow,'\B10001010' );		// PE1,3,7 = output
		counter=0;
//		SetVectExtern3000(0, quadenc_isr);			// set up ISR R2000C
//		WrPortI ( I0CR,&I0CRShadow,'\B00001001' );		// ascendente, prioridad 1
		SetVectExtern2000(1, quadenc_isr);			// set up ISR R2000
		WrPortI ( I0CR,&I0CRShadow,'\B00001010' );		// ascendente, prioridad 2
		while(1){
			printf("%05d \r",counter);
		}
}
```

A continuación, presentamos una versión de handler por polling para Rabbit 2000:

```
int quadenc_poll()
{
static int counter;
static char oldA;

#GLOBAL_INIT {
	counter=0;
	oldA=BitRdPortI(PBDR,0);
}

#asm
		ioi ld A,(PEDR)			; Lee
		ld HL,oldA			; Apunta a estado anterior
		and 0x1				; "change-detect" bit
		cp (HL)				; Compara
		ld (HL),A			; Almacena nuevo estado (preserva Z)
		ld HL,(counter)			; lee contador (preserva Z)
		ret z				; Sale si no hubo cambios
		ex DE,HL
		ld HL,PBDR			; Cambio, apunta a port B
		and A				; Flanco ?
		jr z,falling
		ioi bit 0,(HL)			; A sube, check B
		jr z, AlB
BlA:		dec DE				; B antecede a A
		jr qpdn
AlB:		inc DE				; A antecede a B
		jr qpdn
falling:					; descendente, descartar
qpdn:		ex DE,HL
		ld (counter),HL			; vuelve con el valor del contador
		ret
#endasm
}


main()
{

		WrPortI ( PEDDR,&PEDDRShadow,'\B10001010' );		// PE1,3,7 = output
		while(1){
			printf("%05d \r",quadenc_poll());
		}

}
```

Para Rabbit3000, existe soporte por hardware en la forma de un decodificador en cuadratura. Encontrará información en las samples que acompañan la instalación de Dynamic C.