Cika Electrónica

Nota de Aplicación: CAN-050
Título: **Interrupciones externas en Rabbit 2000, lector ABA (Track II)**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	30/01/06	

Desarrollamos aquí un simple ejemplo de uso de las interrupciones externas en un Rabbit 2000. La aplicación en cuestión es tomar los datos de una lectora de tarjetas RFID con interfaz ABA (Track II).

La interfaz en sí es muy simple, sólo consta de dos líneas: una de clock y una de datos. Los datos son considerados válidos durante el flanco descendente del reloj, por lo que utilizaremos esta señal para generar una interrupción al Rabbit y así observar la señal de datos mediante la lectura de un port de I/O (PE.0 en este caso).

Las interrupciones externas en el Rabbit 2000 vienen de la mano del port E. Si nuestro procesador es R2000C (IQ5T), lo cual es lo más probable debido a que hace ya algunos años que está en el mercado, la forma de seleccionar el puerto de interrupciones es la siguiente:

```
WrPortI(PEDDR, &PEDDRShadow, 0xEE);   // PE0,4=input
WrPortI(PEFR, &PEFRShadow, 0x00);
SetVectExtern3000(0, my_isr);         // setea la dirección
                                      // del handler
WrPortI(I0CR, &I0CRShadow, 0x13);     // habilita INT0 PE4,
                                      // flanco descendente
                                      // prioridad 3
```

Si por el contrario tenemos un R2000 original (IQ2T), deberemos conectar dos de los pines como dice la TN301, e ingresar por uno de ellos tal cual indica la TN301. El software se modifica a :

```
WrPortI(PEDDR, &PEDDRShadow, 0xCE);   // PE0,4,5=input
WrPortI(PEFR, &PEFRShadow, 0x00);
SetVectExtern2000(3, my_isr);         // setea la dirección
                                      // del handler y su
                                      // prioridad
WrPortI(I0CR, &I0CRShadow, 0x13);     // habilita INT0 PE4,
                                      // flanco descendente
WrPortI(I1CR, &I1CRShadow, 0x13);     // habilita INT1 PE5,
                                      // flanco descendente
                                      // PE4 y PE5 unidos
                                      // como indica TN301
```

Dentro del stream de datos recibido, tendremos una cierta cantidad de elementos de 5-bits. Los cinco bits mencionados corresponden a 4 bits de datos y uno de paridad. La paridad es impar para los elementos de datos (número de la tarjeta). Los elementos de señalización rompen este esquema. La transmisión de los bits se realiza MSb primero, y particularmente, la paridad es el MSb. Los dígitos se transmiten MSD primero.

Existen diversas marcas y características de lectoras, por lo que en primera instancia es posible que no sepamos cuántos dígitos vamos a recibir. Ante estas condiciones, lo que haremos es definir un buffer lo suficientemente grande como para recibir la transmisión más larga, e ir guardando los dígitos bit a bit "de izquierda a derecha" (MSb a LSb), de modo que MSb y LSb queden en orden, e iremos contando los pulsos de clock. Una vez terminada la transmisión, lo cual puede hacerse detectando la secuencia de cierre o mediante un timer (aunque en nuestro caso pusimos un pulsador para no complicar el ejemplo), conociendo la cantidad de pulsos de clock tendremos la totalidad de dígitos (dividimos por 5) y podremos extraer el número de tarjeta.

La rutina de interrupciones que es disparada por los flancos de clock y almacena los bits de datos es la siguiente:

```
my_isr::
        push af                       ; salva registros
        push hl
        push bc
        ld a,(count)                  ; counter
```

```
        inc a                           ; incrementa
        ld (count),a                    ; counter
        ioi ld a,(PEDR)                 ; lee port E
        rra                             ; pone PE.0 en carry
        call rotateright
        pop bc
        pop HL
        pop af                          ; recupera registros
        ipres                           ; restablece interrupciones
        ret                             ; retorna
```

La función *rotateright* hace lo que su nombre sugiere, ingresando el nuevo bit por el MSb del primer elemento del buffer y desplazando todos los bits de todos los elementos del buffer a la derecha. De este modo, el primer dígito recibido será el que esté más a la derecha en el buffer.

Como la línea de datos está en estado lógico bajo mientras está inactiva, y la secuencia de inicio corresponde a 10 bits en estado lógico alto, sabiendo que previo a la detección nuestro buffer fue puesto en cero, sabemos que tendremos un uno al detectar el primer bit de la transmisión. Descartamos este bit e inmediatamente hacemos otra rotación, descartando el segundo bit.

```
findstart::
no:     xor a
        call rotateright
        jr nc,no
        xor a
rotateright::
        ld HL,data
        ld b,ABA_BUFSZ
l2:     rr (HL)
        inc HL
        djnz l2
        ret
```

Con la cantidad de pulsos de clock recibidos, conocemos la cantidad de dígitos recibidos, miraremos en la penúltima posición, a modo de descartar los 8 bits restantes del inicio de transmisión. iremos extrayendo los dígitos sucesivamente, realizando un desplazamiento de un dígito (5 bits) de todo el buffer:

```
        findstart();            //  obtiene y descarta start sequence
        nextdigit();            // descarta start sentinel
        num=count[0]/5-7;       // numdigits = clocks/(clocks per digit) - signaling digits
        for(i=0;i<num;i++) {
                digit=(~data[ABA_BUFSZ-2])&0x0F;
                printf("%0x ",(int)digit);
                nextdigit();
      }
#asm
nextdigit::
        ld c,5
l5:     xor a
        call rotateright
        dec c
        jr nz,l5
        ret                     ; retorna
#endasm
```

Por conveniencia, ignoramos la paridad y los señalizadores de inicio y fin (sentinels), así como también el LRC. La confiabilidad de una conexión TTL en un banco de laboratorio es suficientemente alta como para una nota de aplicación e incluso un equipo comercial, aquellos lectores que requieran un cableado extenso, o una confiabilidad mucho mayor, pueden implementar la paridad por dígito y el chequeo de LRC