Cika Electrónica

Nota de Aplicación: CAN-056
Título: **Wenshing TRW-2.4G, con Holtek**
Autor: Sergio R. Caprile, Senior Engineer

| Revisiones | Fecha | Comentarios |
|---|---|---|
| 0 | 07/02/07 | port de CAN-046 |
| | | |
| | | |

En la CAN-045 les presentamos los módulos transceptores TRW-2.4G de Wenshing, transceivers que operan en la banda de 2.4GHz, con capacidad de direccionamiento y selección de canal de comunicaciones. En esta oportunidad, portamos el código a Holtek, con lo cual podemos emplearlos para implementar económicos remotos que se encargan de tomar una medición y reportarla al master cuando éste lo requiere, por ejemplo.

**Desarrollo propuesto**

Vamos a implementar una pequeña especie de biblioteca de funciones que se ocupe de enviar y recibir mensajes utilizando estos módulos, para lo cual comenzaremos por desarrollar rutinas para escribir y leer un byte en el módulo:

```
; Envía un byte
; dato en A
trw_write_byte:
               mov datas,a                  ; guarda dato a transmitir
               mov a,8                      ; 8 bits
               mov bitcntr,a
               clr DATT                     ; DATA = output
               nop
twbl1:         rlc datas                    ; C = MSB (shift bit for output)
               clr SCK                      ; CLK = 0
               set DAT                      ; DATA = 1 (adivino)
               snz C                        ; si hay carry, tuve suerte
               clr DAT                      ; sino DATA = 0
               nop
               set SCK                      ; CLK = 1
               sdz bitcntr
               jmp twbl1
               clr SCK                      ; CLK = 0
               nop
               set DATT                     ; DATA = input
               ret

; Recibe un byte
; Retorna con dato en A
trw_read_byte:
               clr datas                    ; DATA = 0
               mov a,8                      ; 8 bits
               mov bitcntr,a
trbl1:         set SCK                      ; CLK = 1
               nop
               rrca DATPORT                 ; lee dato en C
               rlc datas                    ; y lo pone en el registro (MSB primero)
               clr SCK                      ; CLK = 0
               sdz bitcntr
               jmp trbl1
               mov a,datas
               ret
```

Como habrán deducido, la selección de los pines a emplear se realiza mediante macros, las cuales pueden redefinirse dentro del programa:

```
; DAT debe ser el LSB, de otro modo hay que modificar el programa
DAT            equ     PG.0
DATPORT        equ     PG
```

```
DATT            equ     PGC.0
SCK             equ     PC.5
CE              equ     PB.3
DR              equ     PB.7
CS              equ     PB.2

; Tamaño de paquete en bytes
TRW_DATA_LEN    equ     24
; Cantidad de bytes de dirección
TRW_ADDRESS_LEN equ     5

trwvars .section 'data'
datas           db ?
bitcntr         db ?
bytcntr         db ?
addrptr         db ?
; DEBE SER ESPACIO CONTIGUO
txbuffer        LABEL BYTE
address         db TRW_ADDRESS_LEN DUP(?)
messge          db TRW_DATA_LEN DUP(?)
```

Luego seguimos por el proceso de inicialización. La configuración deseada la guardamos como una tabla. Como las tablas no deben cruzar un espacio de 256 program words, en realidad la definimos en una posición fija y segura de memoria.

```
; Inicializa el TRW-2.4G
trw_init:
                set CS                          ; CS = 1
                clr bytcntr
til1:           mov a,bytcntr
                call getconfig                  ; obtiene byte de tabla en A
                call trw_write_byte             ; envía byte en A
                inc bytcntr
                mov a,18                        ; 18 bytes
                sub a,bytcntr
                sz ACC
                jmp til1                        ; repite
                clr CS                          ; CS = 0
                ret

trwdata .section at LASTPAGE-0100h 'code'
; ...
getconfig:
                addm a,PCL
                ret a,08Eh
                ret a,008h
                ret a,01Ch
                ret a,0C0h
                ret a,0C0h
                ret a,0BBh
                ret a,0BBh
                ret a,0BBh
                ret a,0BBh
                ret a,0BBh
                ret a,013h                      ; LOCAL ADDRESS
                ret a,025h                      ;
                ret a,046h                      ;
                ret a,079h                      ;
                ret a,08Ah                      ;
                ret a,0A3h
                ret a,04Fh
                ret a,014h
```

A continuación, un par de rutinas para enviar y recibir un mensaje. Nótese como para insertar la dirección antes del mensaje a transmitir, hemos definido los dos buffers contiguos. El proceso de direccionamiento lo hacemos copiando la dirección desde una tabla, la cual también debe cumplir los requisitos mencionados en el párrafo anterior. La rutina de recepción indica si vuelve con las manos vacías o trayendo un paquete, según el valor de A, esto nos permitirá realizar otras tareas mientras esperamos un mensaje.

```
; Habilita tranmisión
trw_settx:
                clr CE                          ; CE = 0
```

```
                nop
                set CS                          ; CS = 1
                nop
                clr DATT                        ; DATA = output
                nop
                clr DAT                         ; DATA = 0
                nop
                nop
sndbit:         set SCK                         ; CLK = 1
                nop
                clr SCK                         ; CLK = 0
                nop
                set DATT                        ; DATA = input
                nop
                clr CS                          ; CS = 0
                nop
                set CE                          ; CE = 1
                ret

; Habilita recepción
trw_setrx:
                clr CE                          ; CE = 0
                nop
                set CS                          ; CS = 1
                nop
                clr DATT                        ; DATA = output
                nop
                set DAT                         ; DATA = 1
                jmp sndbit

; Envía un mensaje
; Cargar previamente la dirección en address y el mensaje en message
trw_sendpacket:
                call trw_settx
                mov a,(TRW_ADDRESS_LEN+TRW_DATA_LEN)  ; # bytes
                mov bytcntr,a                   ; Inicializa # bytes a enviar
                mov a,OFFSET txbuffer           ; Inicializa puntero
                mov MP0,a
tsl1:           mov a,IAR0                      ; obtiene byte en A
                call trw_write_byte             ; envía byte en A
                inc MP0                         ; siguiente
                sdz bytcntr                     ; loop
                jmp tsl1
                clr CE                          ; CE = 0
                ret


; Obtiene un mensaje
; Devuelve:
;       A=0 => no recibió nada
;       A=0xFF => hay un mensaje en el buffer
trw_getpacket:
                snz DR                          ; packet ?
                ret a,0                         ; NO
                clr CE                          ; CE = 0
                mov a,TRW_DATA_LEN              ; # bytes
                mov bytcntr,a                   ; Inicializa # bytes a obtener
                mov a,OFFSET messge             ; Inicializa puntero
                mov MP0,a
tgl1:           call trw_read_byte              ; obtiene byte en A
                mov IAR0,a                      ; guarda byte en A en buffer
                inc MP0                         ; siguiente
                sdz bytcntr                     ; loop
                jmp tgl1
                ret a,0FFh


; Setea dirección a quien transmitir
; Llamar con el número de remoto en A (0 en este caso, hay uno sólo cargado... el master...)
trw_address:
                call addresses                  ; Obtiene posición donde está la dirección del
                                                ; remoto solicitado
                mov addrptr,a                   ; la guarda
                mov a,TRW_ADDRESS_LEN           ; # bytes
```

```
                mov bytcntr,a
                mov a,OFFSET address          ; Inicializa puntero
                mov MP0,A
tal1:           call getaddr                  ; obtiene address byte en A
                mov IAR0,a                    ; lo guarda en el buffer
                inc MP0                       ; siguiente
                inc addrptr
                sdz bytcntr                   ; byte count
                jmp tal1                      ; loop
                ret

trwdata .section at LASTPAGE-0100h 'code'

getaddr: mov a,addrptr
addresses:
                addm a,PCL
                ret a,LOW (addrs1-2)          ; 2 bytes offset
;               ret a,LOW (addrs2-2)
; ...
addrs1: ret a,021h
                ret a,043h
                ret a,065h
                ret a,087h
                ret a,0A9h
; addrs2
; ...
getconfig:
; ...
```

Dado que se trata de un medio de acceso múltiple en el cual no tenemos detección de portadora ni de colisiones, es menester arbitrar de algún modo la comunicación para minimizar retransmisiones y lograr confirmar que la información llega a destino. La forma más simple suele ser tener un sistema configurado como master, el cual interroga a los remotos y obtiene respuesta de ellos. Si la pregunta del master no llega a destino, éste la retransmitirá. Si la respuesta del remoto no llega, el master retransmitirá la pregunta. Los remotos deberán estar preparados para recibir varias veces la misma pregunta, sin ofenderse. Con esta filosofía desarrollamos un pequeño programita de ejemplo que ante una petición contesta. El master (quien pregunta) en nuestro caso se realizó con un Rabbit, aprovechando la CAN-045, con la feliz y pacífica coexistencia de ambos remotos, Rabbit y Holtek. Nada impide, empero, que el avezado lector construya su master con Holtek, si así lo desea.

```
                call TXdelay                  ; Esperar al arrancar antes de
                call trw_init                 ; inicializar el módulo (devuelve A=0)
                mov a,0                       ; selecciona master
                call trw_address              ; Setea master address (para responder)

main:           call trw_setrx                ; Habilita rx
mw4msg:         call trw_getpacket            ; Obtiene msg
                snz ACC.0                     ; A!=0, sólo miro bit0
                                              ; (trw_getpacket devuelve 00 o FF)
                jmp mw4msg                    ; no, loop
                call TXdelay                  ; El master necesita un tiempo para habilitar Rx
                                              ; (Rabbit CAN-045 espera 3ms luego de Tx)
        ; acá va el proceso necesario para obtener la respuesta, si >3ms, no hace falta demora
                mov a,041h                    ; sí, prepara ACK
                mov messge,a
                mov a,043h
                mov messge[1],a
                mov a,04Bh
                mov messge[2],a
                clr messge[3]
                call trw_sendpacket           ; y contesta
                call TXdelay                  ; espera >3ms para que salga Tx
                jmp main
```

Las rutinas desarrolladas fueron escritas teniendo en cuenta las especificaciones del fabricante, para 4MHz de clock. La cantidad de calls anidadas se comprobó para un HT48E30. Para otras aplicaciones, se sugiere al interesado revisar que se cumplan los tiempos detallados en el manual del usuario del TRW-2.4G y la cantidad de calls permitidos por el hardware stack del micro de Holtek utilizado.
El archivo adjunto contiene la totalidad de los listados