Cika Electrónica

Nota de Aplicación: CAN-046
Título: **Wenshing TRW-2.4G, con PIC**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	02/11/05	

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 PIC, 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 W
trw_write_byte:
        movwf dat                       ; Guarda en dat
        movlw 8                         ; 8 bits
        movwf bitcntr
        bsf STATUS,RP0                  ; Bank 1
         bcf DATT                       ; DATA = output
        bcf STATUS,RP0                  ; Bank 0
twbl1   rlf dat,f                       ; C = MSB (el MSB va primero)
        bcf SCK                         ; CLK = 0
        bsf DAT                         ; DATA = 1 (será o no será ?)
        btfss STATUS,C                  ; si hay carry, adiviné correctamente
        bcf DAT                         ; sino DATA = 0
        nop
        bsf SCK                         ; CLK = 1
        decfsz bitcntr,f
        goto twbl1
        bcf SCK                         ; CLK = 0
        bsf STATUS,RP0                  ; Bank 1
         bsf DATT                       ; DATA = input
        bcf STATUS,RP0                  ; Bank 0
        return

; Recibe un byte
; Devuelve el dato en W
trw_read_byte:
        clrf dat                        ; DATA = 0
        movlw 8                         ; 8 bits
        movwf bitcntr
trbl1   bsf SCK                         ; CLK = 1
        nop
        rrf DATPORT,W                   ; lee el dato en C
        rlf dat,f                       ; y lo introduce en dat (MSB primero)
        bcf SCK                         ; CLK = 0
        decfsz bitcntr,f
        goto trbl1
        movf dat,W
        return
```

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
```

```
#define DAT    PORTA,0
#define DATPORT PORTA
#define DATT   TRISA,0
#define SCK    PORTA,1
#define CE     PORTA,2
#define DR     PORTA,3
#define CS     PORTA,4

; Packet lenght in bytes
#define TRW_DATA_LEN  24
; Address lenght in bytes
#define TRW_ADDRESS_LEN 5



trwvars UDATA_SHR
dat            RES 1
bitcntr RES 1
bytcntr RES 1
addrptr RES 1
; MUST BE CONTIGUOUS
txbuffer
address RES TRW_ADDRESS_LEN
message RES TRW_DATA_LEN
```

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:
       bsf CS                       ; CS = 1
       clrf bytcntr
til1   movlw HIGH getconfig         ; parte alta de la dirección de la tabla
       movwf PCLATH
       movf bytcntr,W
       call getconfig               ; obtiene byte de configuración en W
       call trw_write_byte          ; lo escribe en el TRW
       incf bytcntr,f
       movlw 18                     ; 18 bytes
       subwf bytcntr,W
       btfss STATUS,Z
       goto til1                    ; repite
       bcf CS                       ; CS = 0
       return

trwdata CODE 0x200
; ...

getconfig:
       addwf PCL,f
       retlw 0x8E
       retlw 0x08
       retlw 0x1C
       retlw 0xC0
       retlw 0xC0
       retlw 0xBB
       retlw 0xBB
       retlw 0xBB
       retlw 0xBB
       retlw 0xBB
       retlw 0x13                   ; LOCAL ADDRESS
       retlw 0x25                   ;
       retlw 0x46                   ;
       retlw 0x79                   ;
       retlw 0x8A                   ;
       retlw 0xA3
       retlw 0x4F
       retlw 0x14
```

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 W, esto nos permitirá realizar otras tareas mientras esperamos un mensaje.

```
; Habilita tranmisión
trw_settx:
        bcf CE                              ; CE = 0
        nop
        bsf CS                              ; CS = 1
        bsf STATUS,RP0                      ; Bank 1
        bcf DATT                            ; DATA = output
        bcf STATUS,RP0                      ; Bank 0
        bcf DAT                             ; DATA = 0
        nop
        nop
sndbit  bsf SCK                             ; CLK = 1
        nop
        bcf SCK                             ; CLK = 0
        bsf STATUS,RP0                      ; Bank 1
        bsf DATT                            ; DATA = input
        bcf STATUS,RP0                      ; Bank 0
        bcf CS                              ; CS = 0
        nop
        bsf CE                              ; CE = 1
        return

; Habilita recepción
trw_setrx:
        bcf CE                              ; CE = 0
        nop
        bsf CS                              ; CS = 1
        bsf STATUS,RP0                      ; Bank 1
        bcf DATT                            ; DATA = output
        bcf STATUS,RP0                      ; Bank 0
        bsf DAT                             ; DATA = 1
        goto sndbit

; Envía un mensaje
; Cargar previamente la dirección en address y el mensaje en message
trw_sendpacket:
        call trw_settx
        movlw (TRW_ADDRESS_LEN+TRW_DATA_LEN)  ; # bytes (direción + datos)
        movwf bytcntr
        movlw txbuffer                      ; Inicializa puntero (buffer conjunto)
        movwf FSR
tsl1    movf INDF,W                         ; lee byte del buffer
        call trw_write_byte                 ; envía al módulo
        incf FSR,f                          ; siguiente
        decfsz bytcntr,f                    ; loop
        goto tsl1
        bcf CE                              ; CE = 0
        return


; Obtiene un mensaje
; Devuelve:
;       W=0 => no recibió nada
;       W=0xFF => hay un mensaje en el buffer
trw_getpacket:
        btfss DR                            ; packet ?
        retlw 0                             ; NO
        bcf CE                              ; CE = 0
        movlw TRW_DATA_LEN                  ; # bytes (mensaje)
        movwf bytcntr
        movlw message                       ; Inicializa puntero
        movwf FSR
tgl1    call trw_read_byte                  ; obtiene byte en W
        movwf INDF                          ; almacena en buffer
        incf FSR,f                          ; siguiente
        decfsz bytcntr,f                    ; loop
        goto tgl1
        retlw 0xFF
```

```
; Setea dirección a quien transmitir
; Llamar con el número de remoto en W (0 en este caso, hay uno sólo cargado... el master...)
trw_address:
        movwf bytcntr                   ; salva W
        movlw HIGH addresses            ; parte alta de la tabla
        movwf PCLATH
        movf bytcntr,W                  ; reupera W
        call addresses                  ; obtiene direción en memoria de la dirección del remoto
        movwf addrptr                   ; la salva
        clrf bytcntr
        movlw address                   ; Inicializa puntero
        movwf FSR
tal1    movlw HIGH addresses            ; parte alta de la tabla
        movwf PCLATH
        call getaddr                    ; obtiene byte de la dirección en W
        movwf INDF                      ; lo guarda en el buffer
        incf FSR,f                      ; siguiente posición en buffer
        incf addrptr,f                  ; siguiente byte de la direción
        incf bytcntr,f                  ; cuenta número de bytes
        movlw TRW_ADDRESS_LEN           ; # bytes
        subwf bytcntr,W
        btfss STATUS,Z
        goto tal1                       ; loop
        return

; debe ser 0xY00, sino hay que modificar el código
trwdata CODE 0x200

getaddr movf addrptr,W
addresses:
        addwf PCL,f
        retlw LOW (addrs1-2)            ; -2: addresses es 0x201, suma a PCL a partir de 0x202
;       retlw LOW (addrs2-2)
; ...
addrs1  retlw 0x21
        retlw 0x43
        retlw 0x65
        retlw 0x87
        retlw 0xA9
; 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 PIC. Nada impide, empero, que el avezado lector construya su master con PIC, si así lo desea.

```
        call TXdelay                    ; Espera al inicio antes de
        call trw_init                   ; inicializar el módulo
        call trw_address                ; dirección del master

main    call trw_setrx                  ; Habilita recepción
mw4msg  call trw_getpacket              ; Hay mensajes ?
        iorlw 0                         ; chequea respuesta
        btfsc STATUS,Z
        goto mw4msg                     ; no, loop
        call TXdelay                    ; da al master tiempo de habilitar Rx (el Rabbit de la
                                        ; CAN-045 espera 3ms luego de Tx)
        ; acá va el proceso necesario para obtener la respuesta, si >3ms, no hace falta demora
        movlw 'A'                       ; prepara ACK
        movwf message
        movlw 'C'
        movwf message+1
```

```
movlw 'K'
movwf message+2
clrf message+3
call trw_sendpacket           ; contesta
call TXdelay                  ; espera para que salga el paquete de Tx
goto 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 PIC16F630. 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 PIC utilizado.

El archivo adjunto contiene la totalidad de los listados