Cika Electrónica

Nota de Aplicación: CAN-017
Título: **Control de Acceso de muy bajo costo**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	05/11/03	

Presentamos un control de acceso de muy bajo costo realizado con tarjetas o tags RFID. Utilizamos como lector al módulo GP8F-R2, de muy bajo costo, el que conectamos a un PIC12F629 para validar los RFID. Sin modificaciones, el sistema emplea el módulo, un LED, una resistencia, y el microcontrolador; más el disparo de triac o relé que utilicemos para control directo de la cerradura. Dada la capacidad de memoria EEPROM del 12F629, podemos almacenar hasta 24 RFIDs en memoria.

Descripción del GP8F-R2

El módulo GP8F-R2 lee tarjetas o tags RFID read-only de 64-bits, código Manchester a 125KHz. Posee una salida para conectar un LED, que permanece encendido y aumenta su intensidad al aproximar un RFID. El ID correspondiente se entrega por un terminal en formato 8 bits serie asincrónico, a 9600 bps, sin paridad, a nivel lógico. El usuario puede optar por conectarlo a algun driver RS-232 para leerlo desde una PC, o conectarlo directamente a un microcontrolador o UART. El formato lógico responde al siguiente esquema, en ASCII:

```
<STX> <DATA (10 bytes)> <CR> <LF> <ETX>
```

STX: ASCII Start of Text (02h)
ETX: ASCII End of Text (03h)
CR: ASCII Carriage Return (0Dh)
LF: ASCII Line Feed (0Ah)

El campo de DATA es una representación en ASCII del ID del RFID, representando 5 bytes binarios (40 bits) de la siguiente forma:

byte	ASCII
C1	43 31

Por ejemplo, el ID `60 22 57 C0 31` se transmite de la siguiente forma:

```
02 36 30 32 32 35 37 43 30 33 31 0D 0A 03
```

Recordemos que este tipo de RFID tags de 64 bits 125 Khz utiliza 9 bits para header, 40 para datos, 14 bits de paridad y uno de stop.

Descripción del sistema de acceso propuesto

Se trata de un simple control de acceso donde al acercar un tag autorizado el sistema responde activando un terminal durante un tiempo determinado. El usuario puede utilizar este terminal para controlar un triac, un relé, o lo que prefiera, que a su vez controlarán un solenoide o pasarán información a otro sistema.

La operatoria del sistema es bastante básica, dado que se trata de una nota de aplicación, sin embargo se provee soporte para incorporar o autorizar nuevos IDs, mediante otro tag previamente autorizado, y una purga o borrado total de la base.

La operatoria propuesta es la siguiente:

1. Al arrancar el sistema, verifica la existencia de IDs en memoria.
2. Si no existen IDs precargados, activa momentáneamente la salida indicando esta situación, y permanece a la espera de un ID. Al ingresarse el primer ID, este pasa a ser el “maestro”, es decir, no se utilizará para ingresar sino para autorizar otros IDs.

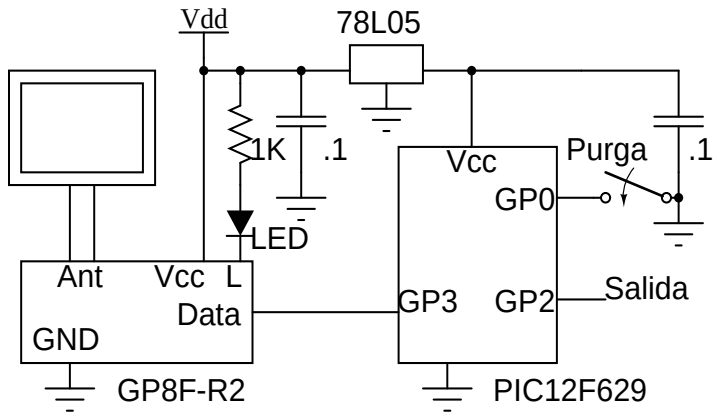
3. Si ya existen IDs precargados (al menos uno), el sistema permanece en espera de un ID que conozca, respondiendo ante cualquiera de ellos diferente del “maestro”, pulsando el terminal de activación por un tiempo prefijado.
4. Si el ID reconocido es el “maestro”, y el pin de purga se encuentra en estado lógico 0, se indica esta condición mediante dos pulsaciones de la salida. El sistema queda entonces a la espera de un ID desconocido a autorizar, el cual memorizará, activando la salida. Si se acerca un tag con un ID conocido, se indicará su rechazo mediante dos pulsaciones de la salida y quedando nuevamente a la espera. Puede salirse de este estado acercando nuevamente la tarjeta “maestra”.
5. Si el ID reconocido es el “maestro”, y el pin de purga se encuentra en estado lógico 1, se procede a borrar toda la base de datos, excepto la tarjeta “maestra”. Se indica esta condición mediante tres pulsaciones de la salida.

Los puntos más flojos de este sistema, en los que el usuario deberá desarrollar un poco más, son los siguientes:

Si se extravía un ID autorizado, no es posible eliminarlo de forma individual, debiendo purgarse la base de IDs, lo que ocasiona la pérdida de todos los demás, excepto el “maestro”. Implementar un borrado selectivo requiere un gran trabajo sobre la base de datos y alguna interfaz de edición, escapando al alcance de esta nota de aplicación.

La tarjeta o ID “maestra” deberá guardarse en lugar seguro, dado que no puede perderse ni caer en manos de personas no autorizadas. De extraviarse, cosa poco probable si se toman las debidas precauciones, se deberá recargar el firmware. Esto podría resolverse fácilmente mediante un pin extra, que al detectarse activo borre toda la base, incluyendo el ID “maestro”. Sin embargo, preferimos no complicar aún más el desarrollo y dejar pines libres para conectar resonadores o cristales (en caso de tener problemas con el oscilador interno), y una posible comunicación con otro sistema o control de otra salida. Queda a criterio del usuario final la última palabra.

Hardware



Conectamos directamente la salida del módulo GP8F-R2 a un pin del microcontrolador. Aprovechamos el pin GP3 que es entrada solamente. Necesitamos otra entrada para señalizar la entrada en modo purga, para lo cual utilizamos GP0 y aprovechamos el pull-up interno. Tomamos la salida de GP2. Dejamos libre GP1 para comunicarnos con otro sistema o controlar otra salida, y GP4/GP5 para conectar un resonador o cristal en caso que se nos presente algún problema con la estabilidad en frecuencia. En las condiciones de prueba en que se desarrolló esta nota de aplicación, hemos utilizado el oscilador interno sin ningún inconveniente. El circuito se alimenta a 12V, según la especificación del GP8F-R2.

Software

Port serie asincrónico

Tomando como base una de las notas de aplicación de Microchip, y dado que el microcontrolador no tiene otra cosa interesante que hacer mientras espera por el ID, realizamos el port serie asincrónico con retardos por software, como puede verse en el listado.

```
clockrate equ 4000000         ; frecuencia del reloj
baudrate equ    9600          ; velocidad del port
;
fclk  equ     clockrate/4

baudconst      equ     ((fclk/baudrate)/3 - 2)
```

```
baudconst2     equ     ((fclk/baudrate)/2 - 2)

#define _tx     GPIO,GP1
#define _rx     GPIO,GP3

servars UDATA_SHR

count   RES     1    ; cuenta bits a recibir
rxreg   RES     1    ; almacena caracter recibido
delay   RES     1    ; retardo para obtener la velocidad deseada
;
        GLOBAL  rxreg
;
sercode CODE

receive
        GLOBAL receive

        movlw   baudconst2                  ; la primera vez espera 1,5 x tiempo de bit
        movwf   delay
        movlw   8                           ; retorna antes del bit de stop
        movwf   count
stopbit btfss    _rx
        goto     stopbit                    ; espera start bit (fin de byte anterior)
strtbit btfsc    _rx
        goto     strtbit                    ; espera start bit
rxbaudwait
        decfsz  delay, F                    ; demora de tiempo de bit
        goto    rxbaudwait
        movlw   baudconst                   ; recarga constante
        movwf   delay
RecvNextBit
        bcf     STATUS,C
        btfsc   _rx                         ; lee entrada
        bsf     STATUS,C
        rrf     rxreg, F                    ; ingresa bit y desplaza (LSB primero)
        decfsz  count, F                    ; siguiente
        goto    rxbaudwait
        return
```

Obtención de un ID

Como viéramos al describir el GP8F-R2, los 40 bits del ID son transmitidos como 10 bytes en hex, es decir, dos bytes ASCII representan un byte (8 bits) del ID, ordenados nibble alto-nibble bajo. Por consiguiente, leeremos 10 bytes conteniendo 10 nibbles y los convertiremos y ordenaremos, como se ve en el listado a continuación.

```
id_vars UDATA_SHR
aux     RES 1                ; soporte para cálculos
charcnt RES 1                ; caracteres a recibir
IDbuf   RES 5                ; buffer para guardar el ID recibido

id_code CODE

getID   call receive                        ; espera STX
        movlw 2
        subwf rxreg,w                       ; STX ?
        btfss STATUS,Z
        goto getID                          ; No, loop
        movlw 5
        movwf charcnt                       ; esperará 2*5 bytes
        movlw IDbuf                         ; Inicializa puntero
        movwf FSR
rxloop  call receive                        ; siguiente
        call ASCII2bin                      ; a binario
        movwf INDF                          ; al buffer
```

```
        swapf INDF,f                    ; nibble alto
        call receive                    ; siguiente caracter
        call ASCII2bin                  ; a binario
        iorwf INDF,f                    ; al buffer (nibble bajo)
        incf FSR,f                      ; siguiente byte
        decfsz charcnt,f                ; loop
        goto rxloop
        call receive                    ; espera CR, descarta
        call receive                    ; espera LF, descarta
        call receive                    ; espera ETX
        movlw 3
        subwf rxreg,w                   ; ETX ?
        btfss STATUS,Z
        goto getID                      ; No, loop (descarta ID)
        return

ASCII2bin
        movlw 041h                      ; Convierte ASCII a binario
        subwf rxreg,w
        btfss STATUS,C
        addlw 7
        addlw 10
        return
```

Operaciones con IDs

Deberemos detectar si el ID recibido corresponde a alguno almacenado en la base, borrar la base, y almacenar un nuevo ID.

Para agregar un ID, por simpleza, dado que no soportamos borrado selectivo de IDs, incrementamos el indicador de cantidad de IDs almacenados y agregamos el nuevo ID al final de la base. De haberse saturado la capacidad de la base ignoramos el ID:

```
addID   call numID                      ; obtiene NUMIDs en W
        movwf aux                       ; a aux
        sublw MAXIDs-1                  ; Lleno ?
        btfss STATUS,C
        return                          ; ignora
        bsf STATUS,RP0                  ; Bank 1
        incf EEDATA,f                   ; NUMIDs += 1
        bsf EECON1,WREN                 ; Habilita write
        call WriteEE                    ; escribe
        bcf STATUS,C                    ; ID address = 5xNUMIDs
        movf aux,w
        rlf aux,f                       ; x2
        rlf aux,f                       ; x4
        addwf aux,w                     ; x5
        movwf EEADR
        incf EEADR,f                    ; offset NUMIDs (+1)
        movlw 5
        movwf charcnt
        movlw IDbuf                     ; Inicializa puntero
        movwf FSR
wIDloop movf INDF,w                     ; lee del buffer
        movwf EEDATA                    ; pone en EEPROM
        call WriteEE                    ; escribe
        incf FSR,f                      ; byte siguiente en buffer
        incf EEADR,f                    ; en base
        decfsz charcnt,f                ; loop
        goto wIDloop
        bcf EECON1,WREN                 ; Inhabilita write
        goto exit

numID   bsf STATUS,RP0                  ; Bank 1
        clrf EEADR                      ; NUMIDs address
        bsf EECON1,RD                   ; EE Read
        movf EEDATA,W                   ; obtiene NUMIDs en W, setea Z si = 0
```

```
        goto exit
```

Reconocemos un ID almacenado mediante comparaciones sucesivas, buscando en forma secuencial desde el primero hasta el último:

```
matchID bsf STATUS,RP0                  ; Bank 1
        clrf EEADR                      ; NUMIDs address
        bsf EECON1,RD                   ; EE Read
        movf EEDATA,W                   ; en W
        movwf aux                       ; en contador
        incf EEADR,f                    ; apunta a base
mIDl2   movlw 5                         ; longitud de ID
        movwf charcnt
        movlw IDbuf                     ; puntero en buffer de recepción
        movwf FSR
mIDl1   bsf EECON1,RD                   ; EE Read
        movf EEDATA,w                   ; lee EEPROM
        subwf INDF,w                    ; compara con buffer
        btfss STATUS,Z                  ; iguales ?
        goto mIDnext                    ; No, siguiente ID
        incf FSR,f                      ; Sí, siguiente byte en buffer
        incf EEADR,f                    ; y en base
        decfsz charcnt,f                ; loop
        goto mIDl1
        movf aux,w                      ; IGUALES! Devuelve número de ID en W, Z reset
exit    bcf STATUS,RP0                  ; Bank 0
        return
mIDnext movf charcnt,w                  ; siguiente ID
        addwf EEADR,f
        decfsz aux,f
        goto mIDl2
        bsf STATUS,Z
        goto exit                       ; No la encuentra, retorna Z set
```

Para borrar la base, simplemente hacemos que la cantidad de IDs almacenados sea 1:

```
purgeDB bsf STATUS,RP0                  ; Bank 1
        bsf EECON1,WREN                 ; Permite write
        clrf EEADR                      ; apunta a NUMIDs
        movlw 1                         ; borra todas excepto la primera
        movwf EEDATA
        call WriteEE                    ; ahora
        bcf EECON1,WREN                 ; Inhibe write
        goto exit
```

La función WriteEE implementa la escritura de un byte en EEPROM, esperando a su finalización:

```
WriteEE ; ints are disabled
        movlw 055h                      ; Permite write
        movwf EECON2
        movlw 0AAh
        movwf EECON2
        bsf EECON1,WR                   ; Inicia write
WEl1    btfsc EECON1,WR                 ; Espera a que termine
        goto WEl1
        return
```

Control de Acceso

Ya estamos entonces en condiciones de realizar el control de acceso, el cual implementa el protocolo que propusimos al comienzo de este artículo:

```
#define Purge   GPIO,GP0

vars    UDATA_SHR       0x20
```

```
IDs     RES    1                        ; # IDs en base

master  CODE   0
start
        clrf   GPIO
        movlw   B'00000111'
        movwf   CMCON                   ; selecciona GPIO en vez de comparador
        bsf STATUS,RP0                  ; Bank 1
        call 3FFh                       ; Valor de calibración del oscilador interno
        movwf OSCCAL                    ; Calibra osc interno
        movlw   B'00001111'             ; Habilita pull-ups
        movwf   OPTION_REG
        movlw   B'00001001'             ; GP1,GP2 = output, GP0,GP3 = input
        movwf   TRISIO
        bcf STATUS,RP0                  ; Bank 0
        call numID                      ; # IDs en W
        movwf IDs
        btfsc STATUS,Z                  ; Z = ninguno, debo obtener master
        goto GetMaster
main    call getID                      ; espera ID
        call matchID                    ; busca en base
        btfsc STATUS,Z                  ; Encuentra ?
        goto main                       ; No, loop
        subwf IDs,w                     ; Mira si es Master
        btfsc STATUS,Z                  ; Master ?
        goto Special                    ; Sí, modo autorización o purga
open    call OpenLock                   ; No, abre
        goto main

Special call pulseLock                  ; Acknowledge (2 pulsos cortos)
        call HSdelay
        call pulseLock
        btfsc Purge                     ; purga ?
        goto DOPurge
        call getID                      ; No, espera ID
        call matchID                    ; Busca en base
        btfsc STATUS,Z                  ; Está ?
        goto learn                      ; No, aprende
        subwf IDs,w                     ; Sí, mira si es Master
        btfss STATUS,Z                  ; Master ?
        goto Special                    ; No, ignora
        goto open                       ; Sí, sale del modo autorización (abre)
learn   call addID                      ; agrega a la base
        call OpenLock                   ; abre (ignora errores)
        goto start

DOPurge call purgeDB                    ; Purga base
        call HSdelay                    ; Acknowledge (3 pulsos cortos)
        call pulseLock
        goto start                      ; restart

GetMaster
        call pulseLock                  ; Acknowledge (1 pulso corto)
        call HSdelay
        call getID                      ; espera ID
        call addID                      ; Guarda en base
        call pulseLock                  ; Acknowledge (1 pulso corto)
        call HSdelay
        goto start                      ; restart
```