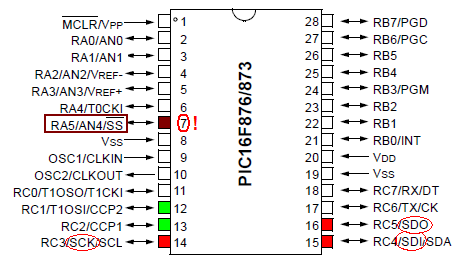
Como controlaremos el SPI del 16F876 por hardware con el nRF24L01, has de saber que hay tres patillas que no se puede modificar de lugar y son:

https://sites.google.com/site/proyectosroboticos/_/rsrc/1353108561579/nrf24l01/16f876-nrf24l01/SPI%20%26%20nRF24L01.PNG

Se han de corresponder con el hardware del PIC que estés usando, en este caso es el 16F876.



**Partes importantes del programa emisor:**

* La interrupción #int\_ext en realidad no llega a usarse, sólo la he puesto en el programa para futuros proyectos en el que quieras hacer recepción de datos además de emitir. Si le llegara algún dato, tal y como está en el programa, lo único que haría sería vaciar el buffer de recepción, nada más; no afecta al programa principal. Pese a lo dicho, no elimines esta parte del programa principal. Luego, cuando todo te funcione bien, haz los cambios que quieras.
* RF\_INT\_EN(); Ese comando habilita la recepción de datos por interrupción de la RB0/INT, pero como sólo vamos a emitir, está puesta simbólicamente para futuros proyectos en el que sí haya recepción de datos. No elimines esta línea hasta que todo te vaya bien.
* RF\_CONFIG\_SPI();  Configura el SPI entre el PIC y el nRF24L01, definiendo los pines, TRIS e interrupción. Estas definiciones están en la librería  "lib\_rf2gh4\_10.h".
* RF\_CONFIG(0x40,0x01); Se refiere "0x40" al canal de comunicación; este siempre ha de ser el mismo para todos los PICs que se quieran comunicar entre sí. "0x01" es la dirección de recepción. Aunque en este caso, por ser exclusivamente emisor, no va a recibir datos, de todas formas hay que ponerle una dirección de recepción.
* RF\_ON(); Activa el módulo RF. Después de este comando conviene que transcurra unos 2.5 milisegundos antes de hacer envíos o recepciones de datos.
* set\_tris\_a(0b111111); Todo el puerto A lo convertimos en entradas de datos. Sólo tiene 6 bits, pero para ver el funcionamiento de este ejercicio es más que suficiente.
* RF\_DATA[0]=porta; Carga lo que hay en el puerto A en el buffer RF\_DATA[0]. Hay 8 bytes: de RF\_DATA[0] a RF\_DATA[7]. Si quieres aprovechar todo el ancho de banda que da el nRF24L01 tendrías que usar los 8 bytes disponibles para enviar información. También lo puedes ver como la posibilidad de enviar 8 bytes de una vez.
* RF\_DIR=0x08; Este comando le está diciendo a qué dirección ha de ir la información que quieras enviar. Piensa que puedes tener varios PICs de recepción (127 teóricos y 32 prácticos), pues irá al PIC que tenga configurado como recepción la dirección 0x08.
* ret2=RF\_SEND(); Envía la información. "ret2" nos sirve para saber qué ha sucedido al intentar enviar. Nos dará '0' si se ha enviado y ha recibido ACK (confirmación de recepción); '1' si ha enviado pero no ha recibido el ACK; y '2' si no ha podido ser enviado (fallo del hardware del emisor). Se podría poner "RF\_SEND();" a secas para enviar, pero ortodoxamente lo correcto es hacerlo tal como lo he puesto.
* La carga de datos en RF\_DATA[] y la dirección de envío RF\_DIR y RF\_SEND(), han de estar dentro del programa juntos, como si fuese un bloque, porqueRF\_DATA y RF\_DIR no contendrán valores constantes todo el tiempo, sino que puede variar, por ello has de cargarlos cada vez que vayas a hacer un envío de datos.

**Partes importantes del programa receptor:**

La interrupción #int\_ext se encargará de recibir los datos.

* No debemos de confundir "ret2" (del emisor) con "ret1" (del receptor), porque en esas variables se guardará también valores "0, 1, 2...". Tienen significados distintos para emitir que para recibir. En el receptor, cuando vemos esto: ret1 = RF\_RECEIVE(); "ret1" nos estará diciendo que, si el valor es '0' hay entrada simple de datos (una sola recepción, o dicho de otra manera, los bytes RF\_DATA[0] a RF\_DATA[7] están listos para ser leídos); si nos da '1' significa que hay más de una entrada (cada entrada es de 8 bytes) para ser leída, es decir, una segunda o tercera tanda, y no soporta más de tres niveles, el resto se perderían. Cuando "ret1" nos dé el valor '2', significa que no hay entradas para leer, o sea, no ha habido entrada nueva de datos. En el programa, tanto si hay entrada simple como múltiple, me quedo con lo último que ha llegado. El buffer (compuesto de 8 bytes por 3 niveles) queda vacío una vez que sale del "while ( (ret1 == 0) || (ret1 == 1) );". Hay que procurar que el emisor sea un poco más lento que el receptor para evitar saturar al receptor y eso puede suceder cuando ret1 nos devuelve el valor '1' (recepción múltiple).
* RF\_INT\_EN(); Este comando habilita la recepción de datos por interrupción de la RB0/INT. Una vez que llegan los datos los puedes tratar inmediatamente, en la misma interrupción, en el ejemplo propuesto lo hacemos de esta forma.
* RF\_CONFIG\_SPI();  Configura el SPI entre el PIC y el nRF24L01, definiendo los patillajes, TRIS e interrupción. Estas definiciones están en la librería "lib\_rf2gh4\_10.h" y las puedes modificar si tu hardware es diferente.
* RF\_CONFIG(0x40,0x08); Se refiere "0x40" al canal de comunicación, este siempre ha de ser el mismo para todos los PICs que quieras comunicar entre sí. "0x08" es la dirección de recepción. Recuerda que en el emisor pusimos: RF\_DIR=0x08; como dirección de envío, así que si el transceptor receptor está configurado con esa dirección podrá recibir esos datos, si no fuese esa dirección el transceptor entenderá que se refiere a otro y no a él, por tanto no atendería a esos datos.
* RF\_ON(); Activa el módulo RF. Después de este comando conviene que transcurra unos 2.5 milisegundos antes de hacer envíos o recepciones de datos.
* set\_tris\_a(0b000000); Todo el puerto A lo convertimos en salidas de datos. Sólo tiene 6 bits, pero para ver el funcionamiento de este ejercicio es más que suficiente. Esas salidas son a través de LEDs. Recuerda que RA4 es de colector abierto, por tanto hemos de poner una resistencia a positivo para poder alimentar ese LED; en el esquema está detallado.
* do {...} while( (ret1 == 0) || (ret1 == 1) ); Mientras haya datos por leer eso estará haciendo. Pero aquí hay que entender bien una cosa: el nRF24L01 envía y recibe 8 bytes de un vez (es un array de 8 bytes y nosotros sólo usamos el primero), esto significa que si hay más datos por leer, leerá otros 8 bytes y hay que cargar esos datos en variables o un array que hayamos declarado nosotros antes de hacer ret1 = RF\_RECEIVE(); Esto sucede cuando hay entrada múltiple de datos y sólo soporta 3 niveles, si hubiese más de tres tandas, el resto se perderían. Es por ello que para un funcionamiento simple y sin problemas se recomienda que el emisor emita un poco más lento de lo que el receptor puede asimilar. En el ejemplo que pongo aquí lo que hace es quedarse con lo último que haya recibido.
* En la recepción de datos, si esos datos no son tratados inmediatamente los puedes perder. Es decir, que una vez que le lleguen datos al receptor has de cargarlos en una variable o matriz (según el caso) antes de que pase al programa principal. En el ejemplo que pongo de recepción no hace falta porque en la misma recepción (dentro de la interrupción) hago que ese dato salga por el puerto A.

**COMUNICACIÓN SPI**

El estándar SPI (Syncronous Peripheral Interface) es utilizado para la comunicación serial entre dispositivos. El SPI fue inicialmente creado por Motorola y adoptado posteriormente por diferentes fabricantes, como Microchip y Atmel. Se trata de un enlace de datos en serie, síncrono,y que opera en modo full dúplex, es decir, las señales de datos viajan en ambas direcciones en forma simultánea.

El canal SPI fue diseñado para aplicaciones de trasmisión de datos a velocidades altas (10 Mbps) y distancias cortas, del orden de 10 a 20 cms, ó bien dentro de un mismo PCB (circuito impreso), entre 2 circuitos integrados como podrían ser un  microcontrolador y otro dispositivo, por ejemplo, un circuito integrado con la función RFID. Las señales de trasmisión de datos y control del canal SPI, usan niveles de voltaje TTL ó bien 3.3 volts, dependiendo de la tecnología de fabricación del dispositivo.

Los dispositivos SPI se comunican entre sí utilizando un bus de 4 señales (MOSI, MISO, SCK, SS) y un esquema maestro/esclavo, en el cual el maestro inicia el protocolo de trasmisión de los datos. En ocasiones, las interfaces SPI son circuitos que están ya disponibles como parte del hardware en los microcontroladores como por ejemplo el [18F2550](http://www.puntoflotante.net/18F2550-BIS.htm) (Microchip) ó Atmega8 (Atmel-AVR) ó bien en dispositivos [como módulos lectores RFID](http://www.puntoflotante.net/RFID-MF522-MFRC522-SPI-BLUETOOTH.htm), tarjetas de memoria, convertidores A/D, etc.

Sin embargo, es también posible implementar un esquema de comunicaciones SPI, utilizando 4 bits de entrada/salida de un microcontrolador junto con un firmware adecuado que maneje el protocolo SPI. A este método se le conoce como 'bit-banging'.  
  
Las señales denominadas MOSI y MISO son portadoras de los datos en ambas direcciones mientras la señal SCK es la señal de reloj la cual sincroniza la recepción de los datos. Por cada pulso de reloj, un bit de información ingresa al dispositivo Master ó al Slave. De tal manera que se requieren 8 pulsos para transferir 1 byte.

La señal SS (Slave Select) habilita el esclavo correspondiente. En la Figura 1 se muestra un ejemplo con solamente un circuito master y un esclavo. Sin embargo, añadiendo varias líneas SS, por ejemplo SS1 y SS2 y SS3, se puede implementar una red de varios circuitos SPI, controlados por el mismo dispositivo Master, como se muestra en la Figura 2.

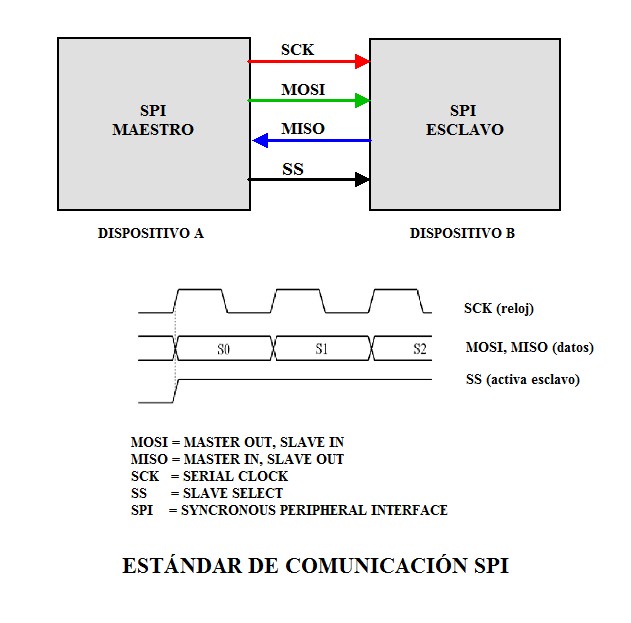
Cuando el maestro selecciona un dispositivo esclavo y genera una señal de reloj (SCK), los datos pueden fluir en ambas direcciones simultáneamente (full dúplex), ya que el mismo reloj funciona para los 2 dispositivos, maestro y esclavo. De hecho, en lo que se refiere a la comunicación SPI, los datos se transfieren siempre en ambas direcciones. Corresponde a los dispositivos maestros y esclavo saber si un byte recibido contiene información útil o no. Así, un dispositivo debe descartar el byte recibido en un modo "sólo transmitir".

SPI no especifica un protocolo de alto nivel para el diálogo maestro-esclavo, y no cuenta con un mecanismo de hardware para la confirmación (acknowledge) ó validación de la recepción de los datos. Por ejemplo, sin un protocolo adecuado en el firmware, el dispositivo SPI maestro no tendría conocimiento de si existe un esclavo conectado y listo para la transferencia de datos.

De acuerdo a lo mencionado en el párrafo anterior, el usuario debe entonces adoptar ó diseñar un protocolo especial para la comunicación SPI, y que permita, tanto el diagnóstico de fallas, como la validación de los bloques de datos trasmitidos, de acuerdo a su aplicación.

Tratándose de un canal de trasmisión síncrono, la velocidad de trasmisión de SPI depende de la frecuencia de generación del reloj (señal SCK), por lo que puede tomar cualquier valor. No existen velocidades estándar de trasmisión, [como en el caso del RS232](http://www.puntoflotante.net/RS485.htm). La velocidad máxima típica de SPI puede llegar a 10 Mbps.

Figura 1: señales presentes en el estándar SPI



|  |
| --- |
| Figura 2: Ejemplo de un diseño de red SPI con un maestro y varios circuitos esclavo. La línea de activación SS permite al maestro seleccionar el circuito con el cual desea comunicarse:  http://www.puntoflotante.net/SPI_three_slaves.jpg |

**CÓDIGO RECEPTOR**

#include <16F876A.h>

#OPT 0 //molestaba para la comunicacion spi, pone el optimizador del código más bajo

#FUSES NOWDT, XT, PUT, NOPROTECT, NODEBUG, NOBROWNOUT, NOLVP, NOCPD, NOWRT

#use delay(clock=4000000) //oscilador comentario

#include "lib\_rf2gh4\_10.h"

#use fast\_io(A)

#use fast\_io(B)

#byte porta=0x05

#byte portb=0x06

INT8 LAMPARA; //39 = ENCENDER //68 = APAGAR

int8 const ENCENDER=39;

int8 const APAGAR=68;

#INT\_EXT

void int\_RB0()

{

int8 ret1;

LAMPARA=0;

ret1 = RF\_RECEIVE();

if ( (ret1 == 0) || (ret1 == 1) )

{

do

{

LAMPARA=RF\_DATA[0]; // El puerto A contendrá el valor que le llege del emisor, a través de RF\_DATA[0].

ret1 = RF\_RECEIVE();

} while ( (ret1 == 0) || (ret1 == 1) );

if(LAMPARA==ENCENDER)

{

output\_high(PIN\_A2);

output\_low(PIN\_A3);

}

if(LAMPARA==APAGAR){

output\_low(PIN\_A2);

output\_low(PIN\_A3);

}

if (LAMPARA!=APAGAR&&LAMPARA!=ENCENDER) output\_high(PIN\_A3);

}

}

#INT\_RB

void interrupcion(){

while(!input(PIN\_B4));

output\_toggle(PIN\_A2);

#asm movf portb,0 #endasm

}

void main()

{

RF\_INT\_EN(); // Habilitar interrupción RB0/INT.

RF\_CONFIG\_SPI(); // Configurar módulos SPI del PIC.

RF\_CONFIG(0x40,0x0A);//0x0A // Configurar módulo RF (canal y dirección).

RF\_ON(); // Activar el módulo RF.

set\_tris\_b(0x11); // Rb0,Rb4 COMO ENTRADA

set\_tris\_a(0x00);

porta=0;

while(true){

SLEEP();

}

}

**CÓDIGO EMISOR**

#include <16F876A.h>//ola k ase

#device adc=10

#OPT 0 //molestaba para la comunicacion spi, pone el optimizador del código más bajo

#FUSES NOWDT, XT, PUT, NOPROTECT, NODEBUG, NOBROWNOUT, NOLVP, NOCPD, NOWRT

#use delay(clock=4000000)

#include "lib\_rf2gh4\_10.h"

#use fast\_io(A)

#use fast\_io(B)

#byte porta=0x05

#byte portb=0x06

int8 inter\_no\_hay\_luz=0;

int8 inter\_boton=0;

int8 inter\_temp=0;

int8 const ENCENDER=39;

int8 const APAGAR=68;

int16 Sensor\_Luz;

#INT\_EXT

void respuesta(){

}

#INT\_RB

void interrupcion(){

if(input(PIN\_B4))

{

inter\_no\_hay\_luz = 1;

if(!input(PIN\_B5)) {

while(!input(PIN\_B5));

inter\_boton=1;

}

if(!input(PIN\_B6)) {

while(!input(PIN\_B6)); //depende del temporizador

inter\_temp=1;

}

}else{

inter\_no\_hay\_luz = 0;

// output\_low(pin\_a1);

}

#asm movf portb,0 #endasm

}

void main()

{

int8 ret2;

set\_tris\_b(0xFF); // Todo el puerto B como entradas.

set\_tris\_a(0x01);

PORT\_B\_PULLUPS(true);

setup\_adc\_ports(AN0); //convertidor analógico-digital / Sensor de luz

setup\_adc(ADC\_CLOCK\_INTERNAL);

RF\_INT\_EN(); // Habilitar interrupción RB0/INT.

RF\_CONFIG\_SPI(); // Configurar módulo SPI del PIC.

RF\_CONFIG(0x40,0x01); // Configurar módulo RF canal y dirección.

RF\_ON(); // Activar el módulo RF.

output\_low(pin\_a1);

output\_low(PIN\_A2);

set\_adc\_channel(0);

delay\_ms(1);

while(true)

{

inter\_no\_hay\_luz=0;

inter\_boton=0;

inter\_temp=0;

SLEEP();

Sensor\_Luz=read\_adc();

delay\_ms(1);

if(Sensor\_Luz > 900)

{

if(inter\_no\_hay\_luz==1){

//cuando no hay luz

//primero envio la señal de encendido = 39 a los dos transceptores

RF\_DATA[0]=ENCENDER;

RF\_DIR=0x08; // Dirección del receptor.

ret2=RF\_SEND(); // Enviar datos.

delay\_ms(25);

RF\_DATA[0]=ENCENDER;

RF\_DIR=0x0A; // Dirección del receptor.

ret2=RF\_SEND(); // Enviar datos.

delay\_ms(25);

output\_high(pin\_a1);

while(inter\_boton==0 && inter\_temp==0 && inter\_no\_hay\_luz == 1)

{ //señal de encendido

delay\_ms(500);

output\_high(PIN\_A2);

delay\_ms(500);

output\_low(PIN\_A2);

}

//señal de apagado

output\_low(PIN\_A2);

RF\_DATA[0]=APAGAR;

RF\_DIR=0x08; // Dirección del receptor.

ret2=RF\_SEND(); // Enviar datos.

delay\_ms(25);

RF\_DATA[0]=APAGAR;

RF\_DIR=0x0A; // Dirección del receptor.

ret2=RF\_SEND(); // Enviar datos.

delay\_ms(25);

output\_low(pin\_a1);

}

}

}

}

**LIBRERÍA lib\_rf2gh4\_10.h**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* DEFINICIONES \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// PORTB

#define RF\_IRQ PIN\_B0

#define RF\_IRQ\_TRIS TRISB,0

// PORTC

#define RF\_CS PIN\_C1

#define RF\_CE PIN\_C2

#define SCK PIN\_C3

#define SDI PIN\_C4

#define SDO PIN\_C5

#define RF\_CS\_TRIS TRISC,1

#define RF\_CE\_TRIS TRISC,2

#define SCK\_TRIS TRISC,3

#define SDI\_TRIS TRISC,4

#define SDO\_TRIS TRISC,5

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* VARIABLES \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#BYTE TRISA = 0x85

#BYTE TRISB = 0x86

#BYTE TRISC = 0x87

#BYTE INTCON = 0x0B

//Variables internas

static int1 interRF;

static int16 noRF;

static int1 RCVNW=0;

static int8 DATA\_N\_SND=0;

static int8 DATA\_N\_RCV=0;

//Variables configurables

static int8 RF\_DATA[8];

static int8 RF\_DIR;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* CÓDIGO \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* RF\_CONFIG\_SPI() \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Descripción: La función configura el módulo SPI del\*

//\*microcontrolador.En ella se especifica como salida \*

//\*SDO y como entrada SDI entre otros parámetros del \*

//\*protocolo SPI. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Variables de entrada: \*

//\*Variables de salida: \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void RF\_CONFIG\_SPI()

{

//Configuración I/O.

bit\_clear(SCK\_TRIS);

bit\_set(SDI\_TRIS);

bit\_clear(SDO\_TRIS);

//Configuración módulo comunicaciones.

setup\_spi(SPI\_MASTER|SPI\_L\_TO\_H|SPI\_XMIT\_L\_TO\_H

|SPI\_CLK\_DIV\_4|SPI\_SAMPLE\_AT\_END);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* RF\_INT\_EN() \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Descripción:Se encarga de habilitar la interrupción\*

//\*externa (RB0) utilizada por el módulo de RF en la \*

//\*recepción de datos. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Variables de entrada: \*

//\*Variables de salida: \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void RF\_INT\_EN()

{

//Habilitar interrupciones externas con flanco de

//bajada.

disable\_interrupts(global);

enable\_interrupts(int\_ext);

ext\_int\_edge( H\_TO\_L );

bit\_set(RF\_IRQ\_TRIS);

ENABLE\_INTERRUPTS(INT\_RB);

enable\_interrupts(global);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* RF\_CONFIG(int canal, int dir) \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Descripción:Esta función se encarga de configurar \*

//\*el transceptor habilitando su propia dirección de \*

//\*escucha y el canal entre otros parámetros. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Variables de entrada:- Canal \*

//\* - Direccion \*

//\*Variables de salida: \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void RF\_CONFIG(int canal, int dir)

{

bit\_clear(RF\_CS\_TRIS);

bit\_set(RF\_IRQ\_TRIS);

bit\_clear(RF\_CE\_TRIS);

output\_low(RF\_CE);

// TX\_ADDR (0xFF)

//Configuración de la dirección de envio aleatoria.

//En la función de enviar se configura la direccion

//deseada por el usuario.

output\_low(RF\_CS);

spi\_write(0x30);

spi\_write(0xFF);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

output\_high(RF\_CS);

// RX\_ADDR\_P0 (0xFF) ACK

//Configuración de la direccióndel Pipe0 para la

//recepción de ACK.

output\_low(RF\_CS);

spi\_write(0x2A);

spi\_write(0xFF);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

output\_high(RF\_CS);

// RX\_ADDR\_P1 (dir)

//Configuración de la direccióndel Pipe1 para la

//recepción de tramas.

output\_low(RF\_CS);

spi\_write(0x2B);

spi\_write(dir);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

output\_high(RF\_CS);

// RX\_ADDR\_P2 (0x00) BROADCAST

//Configuración de la direccióndel Pipe2 para la

//recepción de tramas

output\_low(RF\_CS);

spi\_write(0x2C);

spi\_write(0x00);

output\_high(RF\_CS);

// EN\_AA

//Habilitar AutoAck en los Pipe0,Pipe1 y Pipe2.

output\_low(RF\_CS);

spi\_write(0x21);

spi\_write(0x07);

output\_high(RF\_CS);

// EN\_RXADDR

//Habilitar los Pipe0,Pipe1 y Pipe2.

output\_low(RF\_CS);

spi\_write(0x22);

spi\_write(0x07);

output\_high(RF\_CS);

// SETUP\_AW

//Configuración de la longitud de las direcciones.

//Direcciones de 5 bytes.

output\_low(RF\_CS);

spi\_write(0x23);

spi\_write(0x03);

output\_high(RF\_CS);

//SETUP\_RETR

//Configuración de las retrasmisiones en la transmisión.

//Diez retransmisiones cada 336us.

output\_low(RF\_CS);

spi\_write(0x24);

spi\_write(0x0A);

output\_high(RF\_CS);

//RF\_CH

//Configuración del canal.

//Canal elegido por el usuario (0x01 - 0x7F).

output\_low(RF\_CS);

spi\_write(0x25);

spi\_write(canal);

output\_high(RF\_CS);

//RF\_SETUP

//Configuración aspectos RF.

//Ganancia máxima de LNA, 0dBm potencia de salida y 1Mbps de velocidad.

output\_low(RF\_CS);

spi\_write(0x26);

spi\_write(0x07);

output\_high(RF\_CS);

//STATUS

//Reseteo del registro STATUS

output\_low(RF\_CS);

spi\_write(0x27);

spi\_write(0x70);

output\_high(RF\_CS);

//RX\_PW\_P0

//Nº de bytes en Pipe0.

//1 byte (ACK).

output\_low(RF\_CS);

spi\_write(0x31);

spi\_write(0x01);

output\_high(RF\_CS);

//RX\_PW\_P1

//Nº de bytes en Pipe1.

//10 byte (Direccion emisor y trama).

output\_low(RF\_CS);

spi\_write(0x32);

spi\_write(0x0A);

output\_high(RF\_CS);

//RX\_PW\_P2

//Nº de bytes en Pipe2.

//10 byte (Direccion emisor y trama).

output\_low(RF\_CS);

spi\_write(0x33);

spi\_write(0x0A);

output\_high(RF\_CS);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* RF\_ON() \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Descripción:Esta rutina activa el módulo de \*

//\*radiofrecuencia en modo escucha para poder recibir \*

//\*datos enviados a su dirección. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Variables de entrada: \*

//\*Variables de salida: \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void RF\_ON()

{

output\_low(RF\_CE);

// CONFIG

//Se activa el modulo, se pone en recepción,

//se activa el CRC para que utilice 2 bytes.

output\_low(RF\_CS);

spi\_write(0x20);

spi\_write(0x0F);

output\_high(RF\_CS);

delay\_ms(2);

output\_high(RF\_CE);

delay\_us(150);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* RF\_OFF() \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Descripción:Este procedimiento desactiva el módulo \*

//\*de radiofrecuencia. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Variables de entrada: \*

//\*Variables de salida: \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void RF\_OFF()

{

output\_low(RF\_CE);

// CONFIG

//Se desactiva el modulo

output\_low(RF\_CS);

spi\_write(0x20);

spi\_write(0x0C);

output\_high(RF\_CS);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* RF\_SEND() \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Descripción:Esta función envía 8 Bytes de datos a \*

//\*la dirección indicada informando de la correcta \*

//\*recepción en el destinatario. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Variables de entrada:- RF\_DATA[] \*

//\* - RF\_DIR

//\*Variables de salida: - \*

//\*Salida: - 0: Envío correcto (ACK OK) \*

//\* - 1: No recepcibido (NO ACK) \*

//\* - 2: No enviado \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int RF\_SEND()

{

int i;

int estado;

if(bit\_test(INTCON,7))

interRF=1;

else

interRF=0;

disable\_interrupts(GLOBAL);

// INICIO

output\_low(RF\_CE);

//STATUS

//Reseteo del registro STATUS

output\_low(RF\_CS);

spi\_write(0x27);

spi\_write(0x70);

output\_high(RF\_CS);

// EN\_RXADDR

//Se habilita el Pipe0 para la recepción del ACK

output\_low(RF\_CS);

spi\_write(0x22);

spi\_write(0x01);

output\_high(RF\_CS);

// TX\_ADDR

//Se configura la dirección de transmisión=RF\_DIR

output\_low(RF\_CS);

spi\_write(0x30);

spi\_write(RF\_DIR);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

output\_high(RF\_CS);

// RX\_ADDR\_P0

//Para la recepción del ACK se debe configurar el Pipe0 con

//la misma dirección a trasmitir.

output\_low(RF\_CS);

spi\_write(0x2A);

spi\_write(RF\_DIR);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

spi\_write(0xC2);

output\_high(RF\_CS);

// RX\_ADDR\_P1

//Se mete en RF\_DIR la direccion propia.

//De esta manera el receptor sabe la dirección

//del transmisor.

output\_low(RF\_CS);

spi\_write(0x0B);

RF\_DIR=spi\_read(0);

spi\_read(0);

spi\_read(0);

spi\_read(0);

spi\_read(0);

output\_high(RF\_CS);

// W\_TX\_PAYLOAD

//Se manda los datos al transductor

output\_low(RF\_CS);

spi\_write(0xA0);

DATA\_N\_SND++;

spi\_write(DATA\_N\_SND);

spi\_write(RF\_DIR);

for (i=0;i<8;i++)

spi\_write(RF\_DATA[i]);

output\_high(RF\_CS);

// CONFIG

//Se pasa a modo transmisión.

output\_low(RF\_CS);

spi\_write(0x20);

spi\_write(0x0E);

output\_high(RF\_CS);

// Pulso de comienzo de envío

output\_high(RF\_CE);

delay\_us(15);

output\_low(RF\_CE);

noRF=0;

while (input(RF\_IRQ)==1) {

noRF++;

//Si no da respuesta en 7ms, no se ha enviado.

if(noRF==500){

break;

}

}

// STATUS

//Lectura del estado en el registro estatus.

output\_low(RF\_CS);

estado=spi\_read(0x27);

spi\_write(0x70);

output\_high(RF\_CS);

// EN\_RXADDR

//Habilitar los Pipe0,Pipe1 y Pipe2.

output\_low(RF\_CS);

spi\_write(0x22);

spi\_write(0x07);

output\_high(RF\_CS);

// TX\_FLUSH

//Limpieza de la FIFO de salida

output\_low(RF\_CS);

spi\_write(0xE1);

output\_high(RF\_CS);

// CONFIG

//Paso a modo recepción

output\_low(RF\_CS);

spi\_write(0x20);

spi\_write(0x0F);

output\_high(RF\_CS);

// FIN

output\_high(RF\_CE);

delay\_us(150);

//Si no da respuesta en 7ms, no se ha enviado.

if(noRF==500){

if(interRF==1)

enable\_interrupts(GLOBAL);

clear\_interrupt(int\_ext);

return(2);

}

//estado

//Chequeo de los bit del registro STATUS que indican si se ha recibido

//ACK y si se ha terminado las retrasmisiones sin ningun ACK.

if ((bit\_test(estado,4)==0) && (bit\_test(estado,5)==1)){

if(interRF==1)

enable\_interrupts(GLOBAL);

clear\_interrupt(int\_ext);

return(0);

}

else{

if(interRF==1)

enable\_interrupts(GLOBAL);

clear\_interrupt(int\_ext);

return(1);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* RF\_RECEIVE() \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Descripción: Esta rutina se encarga de comprobar si\*

//\*se ha producido una recepción y de ser así, \*

//\*devuelve la trama recibida. \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*Variables de entrada:- \*

//\*Variables de salida: - RF\_DATA[] \*

//\* - RF\_DIR \*

//\*Salida: - 0: Recepción correcta y única \*

//\* - 1: Recepción correcta y múltiple \*

//\* - 2: No se ha producido recepción \*

//\* - 3: No se ha producido recepción \*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int RF\_RECEIVE()

{

int i;

int mas;

int estado;

if (input(RF\_IRQ)==1 && RCVNW==0){

return (2);

}

//STATUS

//Lectura y reseteo del registro STATUS

output\_low(RF\_CS);

estado=spi\_read(0x27);

spi\_write(0x70);

output\_high(RF\_CS);

//estado

//Chequeo de la interrupción de recepción.

if (bit\_test(estado,6)==0 && RCVNW==0){

return(3);

}

//R\_RX\_PAYLOAD

//Lectura de los datos recibidos.

output\_low(RF\_CS);

spi\_write(0x61);

DATA\_N\_RCV=spi\_read(0);

RF\_DIR=spi\_read(0);

for (i=0;i<8;i++)

{

RF\_DATA[i]=spi\_read(0);

}

output\_high(RF\_CS);

//FIFO\_STATUS

//Comprobación del estado de la FIFO de

//recepción para comprobar si hay más datos

output\_low(RF\_CS);

spi\_write(0x17);

mas=spi\_read(0);

output\_high(RF\_CS);

if (bit\_test(mas,0)==0){

RCVNW=1;

return(1);

}

RCVNW=0;

return(0);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*