Cika Electrónica

Nota de Aplicación: CAN-045
Título: **Wenshing TRW-2.4G, con Rabbit 3000**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	26/10/05	
1	31/10/05	Ajustes menores de timing

Les presentamos los módulos transceptores TRW-2.4G de Wenshing. Se trata de transceivers que operan en la banda de 2.4GHz, con capacidad de direccionamiento y selección de canal de comunicaciones.

Descripción del TRW-2.4G

Estos módulos poseen una potencia de salida de 0dBm, lo que permite un alcance algo más reducido que una red Wi-Fi. Funcionan a 3,3V, y su consumo es bastante reducido; pero lo realmente interesante está en su interfaz. La interfaz del módulo es netamente digital, si bien nada impide su utilización en la forma tradicional (entra bit - sale bit), estos módulos pueden trabajar con un sistema denominado ShockBurst, que es algo así como un store and forward que permite emplear micros sin UART, y/o con relojes de baja frecuencia y/o poca precisión, comunicándose a una velocidad baja, sin por ello mantener ocupado el canal de comunicaciones.
El micro y el módulo se comunican mediante una interfaz de cinco pines, al ritmo que el micro marca en la señal CLK. Señalizado el fin de paquete a transmitir, el módulo lo transmite a una velocidad mucho mayor (250Kbps), minimizando el riesgo de colisiones con otros módulos. El módulo que recibe un mensaje, informa al micro mediante el pin DR1; el micro procederá entonces a leer la información del módulo a su propio ritmo.
Las señales restantes sirven para informar al módulo que lo estamos accediendo para configurarlo (CS), o para enviar o recibir datos (CE). Los datos viajan en uno u otro sentido por el pin DATA.
Una característica interesante adicional, es la posibilidad de recibir en dos canales simultáneos, teniendo dos interfaces separadas (DR2, CLK2, DOUT2)

Al aplicar alimentación al módulo, éste se encuentra en un estado indefinido y deberá ingresarse la configuración. Entre estos datos, se encuentra la dirección propia del módulo, el canal de operación dentro de la banda, la longitud de los mensajes, y el modo de trabajo, ya que es half-duplex. Tanto recepción como transmisión se realizan por el mismo canal.

Previo al mensaje a transmitir, el micro comunica al módulo la dirección del módulo destinatario, como si fuera parte del mensaje (los primeros cinco bytes). El módulo agregará luego un CRC a la cola y transmitirá el mensaje. Como el módulo destinatario conoce la longitud de lso mensajes, puede validar el CRC y comunicar al micro la presencia de un mensaje sólo cuando éste es válido. Al entregar el mensaje, se eliminan la propia dirección y el CRC, es decir, se obtiene el mensaje solamente.

Desarrollo propuesto

Vamos a implementar una 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:

```
nodebug void trw_write_byte(unsigned char value)
{
  unsigned char i;
  BitWrPortI (TRW_DATA_DIR,&TRW_DATA_DSHADOW,1,TRW_DATA_BIT ); // usa DATA como salida
  for (i=0x80;i>0;i/=2){            //shift bit for masking
    BitWrPortI (TRW_SCK_PORT,&TRW_SCK_SHADOW,0,TRW_SCK_BIT );
    if (i & value)
           BitWrPortI (TRW_DATA_PORT,&TRW_DATA_SHADOW,1,TRW_DATA_BIT );    // DATA=1
    else
           BitWrPortI (TRW_DATA_PORT,&TRW_DATA_SHADOW,0,TRW_DATA_BIT );    // DATA=0
    BitWrPortI (TRW_SCK_PORT,&TRW_SCK_SHADOW,1,TRW_SCK_BIT );              // clk
  }
```

```
  BitWrPortI (TRW_SCK_PORT,&TRW_SCK_SHADOW,0,TRW_SCK_BIT );
  BitWrPortI (TRW_DATA_DIR,&TRW_DATA_DSHADOW,0,TRW_DATA_BIT );      // libera DATA
}

nodebug char trw_read_byte()
{
  unsigned char i,val;

  val=0;
  for (i=0x80;i>0;i/=2){
    BitWrPortI (TRW_SCK_PORT,&TRW_SCK_SHADOW,1,TRW_SCK_BIT );       // CLK
    if (BitRdPortI(TRW_DATA_PORT,TRW_DATA_BIT))
        val=(val | i);                                              // lee bit
    BitWrPortI (TRW_SCK_PORT,&TRW_SCK_SHADOW,0,TRW_SCK_BIT );
  }
  return val;
}
```

Como habrán deducido, la selección de los pines a emplear se realiza mediante macros, las cuales pueden redefinirse dentro del programa. El archivo adjunto tiene un ejemplo, aunque nada reemplaza la cuidadosa observación del código fuente de la biblioteca de funciones.

Luego seguimos por el proceso de inicialización, en el cual escribiremos la configuración deseada.

```
nodebug void trw_init()
{
int i;
unsigned char *cfgptr;
const static unsigned char config[18]={
0X8E,0X08,0X1C,                             // TEST, PLL
TRW_DATA_LENb,                              // DATA2_W  longitud de datos 2do receptor
TRW_DATA_LENb,                              // DATA1_W  longitud de datos
TRW_DUMMY_ADDRESS,                          // ADDR2 dirección segundo receptor
TRW_LOCAL_ADDRESS,                          // ADDR1 dirección
TRW_ADDR_W,                                 // ADDR_W/CRC tipo de CRC
TRW_RF_CONFIG,                              // RF config
(TRW_RF_CHANNEL<<1)                         // frecuencia del canal, modo Tx/Rx
};

   BitWrPortI (TRW_CS_PORT,&TRW_CS_SHADOW,1,TRW_CS_BIT);    // sube CS
   i=18;
   cfgptr=config;
   while(i--)                                               // envía config
        trw_write_byte(*cfgptr++);
   BitWrPortI (TRW_CS_PORT,&TRW_CS_SHADOW,0,TRW_CS_BIT);    // baja CS
}
```

A continuación, un par de rutinas para enviar y recibir un mensaje. Nótese como insertamos la dirección antes del mensaje, al transmitir.

```
nodebug void trw_settx(void)
{
   BitWrPortI (TRW_CS_PORT,&TRW_CS_SHADOW,1,TRW_CS_BIT);
   BitWrPortI (TRW_DATA_DIR,&TRW_DATA_DSHADOW,1,TRW_DATA_BIT );     // usa DATA como salida
   BitWrPortI (TRW_DATA_PORT,&TRW_DATA_SHADOW,0,TRW_DATA_BIT );     // DATA=0
   BitWrPortI (TRW_SCK_PORT,&TRW_SCK_SHADOW,1,TRW_SCK_BIT );        // clk
   BitWrPortI (TRW_SCK_PORT,&TRW_SCK_SHADOW,0,TRW_SCK_BIT );
   BitWrPortI (TRW_CS_PORT,&TRW_CS_SHADOW,0,TRW_CS_BIT);
   BitWrPortI (TRW_DATA_DIR,&TRW_DATA_DSHADOW,0,TRW_DATA_BIT );     // libera DATA
}

nodebug void trw_setrx(void)
{
   BitWrPortI (TRW_CS_PORT,&TRW_CS_SHADOW,1,TRW_CS_BIT);
   BitWrPortI (TRW_DATA_DIR,&TRW_DATA_DSHADOW,1,TRW_DATA_BIT );     // usa DATA como salida
   BitWrPortI (TRW_DATA_PORT,&TRW_DATA_SHADOW,1,TRW_DATA_BIT );     // DATA=1
   BitWrPortI (TRW_SCK_PORT,&TRW_SCK_SHADOW,1,TRW_SCK_BIT );        // clk
   BitWrPortI (TRW_SCK_PORT,&TRW_SCK_SHADOW,0,TRW_SCK_BIT );
   BitWrPortI (TRW_CS_PORT,&TRW_CS_SHADOW,0,TRW_CS_BIT);
   BitWrPortI (TRW_DATA_DIR,&TRW_DATA_DSHADOW,0,TRW_DATA_BIT );     // libera DATA
   BitWrPortI (TRW_CE_PORT,&TRW_CE_SHADOW,1,TRW_CE_BIT);
}
```

```
nodebug cofunc void trw_sendpacket(unsigned char *address,unsigned char *buf)
{
auto int len;


   trw_settx();                                                    // configura tx
   BitWrPortI (TRW_CE_PORT,&TRW_CE_SHADOW,1,TRW_CE_BIT);           // sube CE
   len=TRW_ADDRESS_LEN;
   while(len--){                                                   // manda dirección remota
        trw_write_byte(*address++);
        yield;
   }
   len=TRW_DATA_LEN;
   while(len--){                                                   // manda datos
        trw_write_byte(*buf++);
        yield;
   }
   BitWrPortI (TRW_CE_PORT,&TRW_CE_SHADOW,0,TRW_CE_BIT);           // baja CS
}

nodebug cofunc int trw_getpacket(unsigned char *buf,int timeout)
{
long taimaut;
int len;

   trw_setrx();                                                    // configura rx
   BitWrPortI (TRW_CE_PORT,&TRW_CE_SHADOW,1,TRW_CE_BIT);           // habilita Rx (CE=1)
   waitfor(DelayMs(2));
   if (timeout) {                                                  // espera indicación
        taimaut=MS_TIMER+timeout;
        while(!BitRdPortI(TRW_DR_PORT,TRW_DR_BIT)){                // de recepción DR1=1
                if(MS_TIMER<taimaut)
                        yield;
                else {
                        BitWrPortI (TRW_CE_PORT,&TRW_CE_SHADOW,0,TRW_CE_BIT);
                        return(-1);                                // o sale por timeout
                }                                                  // si se indica
        }
   }
   else {
        while(!BitRdPortI(TRW_DR_PORT,TRW_DR_BIT))
                yield;
   }
   BitWrPortI (TRW_CE_PORT,&TRW_CE_SHADOW,0,TRW_CE_BIT);           // baja CE
   len=TRW_DATA_LEN;
   while(len--){
        *buf++ = trw_read_byte();                                  // lee datos
        yield;
   }
   return(TRW_DATA_LEN);
}
```

Finalmente, una simple rutina adicional que se encarga de retransmitir el mensaje si no se obtiene una respuesta dentro de un cierto tiempo. 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, que coincide con la propuesta, 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.

```
nodebug cofunc int trw_poll(unsigned char *raddr,char *msg,int timeout)
{
int i,retries;

   retries=0;
   do {
        wfd trw_sendpacket(raddr,msg);                    // envía mensaje
        waitfor(DelayMs(TRW_TXGUARDTIME));                // espera transmisión
        wfd i=trw_getpacket(msg,timeout);                 // espera respuesta
   } while ((i<0) && (retries++ < TRW_NUM_RETRIES));      // repite si no la hay
```

```
   return(i);                                          // hasta que desiste
}
```

Las rutinas desarrolladas fueron escritas teniendo en cuenta las especificaciones del fabricante, pero aprovechando los retardos normales de un programa en C, para evitar introducir demoras, funcionando correctamente aun en cores rápidos como el RCM-3360. Para otras aplicaciones, se sugiere al interesado revisar que se cumplan los tiempos detallados en el manual del usuario del TRW-2.4G

A continuación, un par de programas de ejemplo, uno para master (pregunta) y otro para slave (responde), respectivamente:

```
void main()

{
unsigned int num;
int i;
unsigned char msg[TRW_DATA_LEN+1];


        trw_init();                                     // inicializa módulo

        num=0;
        while(1){
                costate {
                        sprintf(msg,"%d:Mensaje",num);  // prepara pregunta
                        wfd i=trw_poll(raddr1,msg,100); // envía y espera respuesta
                        if(i>0){
                                // procesa respuesta
                        }
                        num++;                          // otro mensaje...
                }
                // otras tareas
        }
}

void main()

{
int i,num;
unsigned char msg[TRW_DATA_LEN+1];

        trw_init();                                     // inicializa módulo

        while(1){
                costate {
                        wfd i=trw_getpacket(msg,5000);  // espera pregunta (5 segundos)
                        if(i>0){
                                sprintf(msg,"%d: ACK",atoi(msg));     // prepara respuesta
                                // permitir que el master ingrese en recepción
                                wfd trw_sendpacket(raddr1,msg);       // envía respuesta
                                waitfor(DelayMs(TRW_TXGUARDTIME));    // espera transmisión
                        }
                        else {
                                // no hay preguntas por 5 segundos (me voy a dormir ?)
                        }
                // otras tareas
                }
        }
}
```

El archivo adjunto contiene la totalidad del listado, en la forma de library, con dos programas ejemplo de uso, uno para RCM-3360 (master) y otro para RCM-3720 (slave). Ambos comentan la operación por el port serie A a 57600 bps. Por último, recordemos que para poder compilar estos ejemplos, se debe incluir la library en el archivo LIB.DIR