Cika Electrónica

Nota de Aplicación: CAN-070
Título: **Comunicación serie asincrónica con Ramtron VRS51L3074**
Autor: Sergio R. Caprile, Senior Engineer

Revisiones	Fecha	Comentarios
0	21/02/07	
1	28/02/07	Ampliación y corrección uso de UARTxINT

En esta nota de aplicación desarrollaremos un simple y práctico sistema para utilizar comunicación serie asincrónica por las UARTs del VRS51L3074.

Dado que la mayoría de las veces la comunicación con un dispositivo de orden superior o inferior es del tipo comando-respuesta, y generalmente se trata de mensajes cortos con una longitud medianamente predecible y un funcionamiento pausado, desarrollaremos un esquema con buffers lineales, muy simple, que nos permita esperar un comando y contestar, o enviar un comando y esperar una respuesta.

El esquema a desarrollar presenta al usuario las siguientes funciones:

- ➔ *serXpeek()*: devuelve el último caracter en el buffer (o -1 si el buffer está vacío), y sirve para detectar fin de mensaje en protocolos de comunicaciones con señalización por caracter especial.
- ➔ *serXflush()*: borra el buffer de recepción, se utiliza luego de procesado el mensaje, para hacer lugar para el siguiente.
- ➔ *serXsend()*: inicia el proceso de envío de un mensaje

y las siguientes variables:

- ➔ *serXrxbytes*: indica la cantidad de caracteres recibidos, se utiliza para detectar fin de mensaje en protocolos de comunicaciones con mensajes de longitud conocida.
- ➔ *serXtxbytes*: contiene la cantidad de caracteres a ser transmitidos, será decrementada por la rutina de transmisión y se leerá como cero cuando se haya finalizado.
- ➔ *serXrxbuf*: contiene el mensaje recibido.

Tanto transmisión como recepción se realizan mediante una rutina de interrupciones, muy simple, que resulta en un código compilado compacto. Para esta aplicación genérica no fue necesario escribirla en assembler. Para buffers de menos de 256 caracteres es posible generar código algo más compacto en assembler.

```
void ser0Interrupt(void) interrupt 5
{
register unsigned char st,dat;

    st=UART0INT;
    if(st&0x6){                       // RXAVENF o RXOVF
      dat=UART0BUF;                   // lee caracter
      if(ser0rxbytes<RXBUF0SZ){       // buffer overflow: descarta
        *(rxptr++)=dat;               // pone en el buffer
        if(st&(1<<2))
                rxptr--;              // (excepto si es overrun)
        else
                ser0rxbytes++;        // incrementa cantidad recibida
      }
    }
    else {                            // TXEMPTYF
      if(ser0txbytes){
        UART0BUF=*(txptr++);           // si hay algo que enviar, lo hace
        --ser0txbytes;                // decrementa cuenta
      }
      if(!ser0txbytes)
        UART0INT=0x62;                // si no hay más, no más interrupciones
    }
}
```

A continuación, el set de rutinas utilizado

```
#define RXBUF0SZ 16

__xdata unsigned char ser0rxbuf[RXBUF0SZ];
static __xdata unsigned char *txptr,*rxptr;
unsigned char ser0txbytes,ser0rxbytes;

int ser0peek(void)
{
  if(ser0rxbytes)                   // si hay algo en el buffer
    return(*(rxptr-1));             // devuelve el último caracter
  else
    return(-1);                     // sino, devuelve -1
}

void ser0flush(void)
{
register unsigned char aux;

  aux=UART0INT;                     // lee registro de configuración de interrupciones
  aux&=0x70;                        // mantiene sólo los bits de interrupciones (4,5,6)
  aux|=0x02;                        // mantiene habilitada la recepción
  UART0INT=(aux&~0x60);             // inhabilita interrupciones de rx (RXAVENF + RXOVF)
  rxptr=ser0rxbuf;                  // inicializa puntero de recepción
  ser0rxbytes=0;                    // resetea cuenta de caracteres recibidos
  UART0INT=aux;                     // restablece estado de interrupciones
}

void ser0send(__xdata unsigned char *ptr,unsigned char bytes)
{
  UART0INT=0x62;                    // inhabilita interrupción de tx (TXEMPTY)
  txptr=ptr;                        // inicializa puntero de transmisión
  ser0txbytes=bytes;                // y cuenta de caracteres a transmitir
  UART0INT=0x72;                    // habilita TXEMPTY
}
```

La inicialización de la UART la hacemos mediante una rutina como la siguiente. En este caso asumimos siempre 8-bits por caracter y 1-bit de stop.

```
void ser0init(unsigned long clockspeed)
{
register unsigned char aux;
register unsigned int aux2;

        PERIPHEN1 |= (1<<3);                // Habilita UART0

        aux=(unsigned char)(clockspeed>>1)&0x0F; // Ajuste fino
        UART0CFG = (aux<<4);                // clock interno, 8-bits, 1 stop

        UART0INT = 0x62;                    // RX AV + RXOV int + Enable Reception
        UART0EXT = 0x00;                    // sin extensiones

        aux2=((unsigned int)(clockspeed>>5)&0xFFFF)-1;   // BRG = (clock/speed)/32 -1
        UART0BRL = (unsigned char) aux2&0xFF;
        UART0BRH = (unsigned char)(aux2>>8)&0xFF;
        ser0flush();
        INTSRC1 &= ~(1<<5);                  // UART0 module interrupt
        INTEN1 |= (1<<5);                    // interrupt 5 enable
}
```

Finalmente, un programa de ejemplo para ver en forma conceptual el uso de las rutinas. Para la transmisión, le pasamos a la rutina que inicia el proceso la dirección de un buffer en memoria de datos externa.

```
#include "serial0.h"

__xdata unsigned char txbuf[10];

void main()
{
        ser0init(40000000/9600);                    // clock/speed
        GENINTEN = 0x03;                            // Habilita interrupciones
        while(1){
                ...
```

```
                while(ser0peek()!=0x0D);              // espera recibir <enter>
// o bien:      while(ser0rxbytes<14);                // espera recibir 14 caracteres

                // procesa el mensaje en ser0rxbuf
                ser0flush();                          // libera buffer para nuevo mensaje
                ...
                memcpy(txbuf,"OK\r\n",4);             // pone "OK<CR><LF>" en el buffer
                ser0send(txbuf,4);                    // envía mensaje
                while(ser0txbytes);                   // espera a que se envíe
                ...
        }
}
```

En la mayoría de los casos no nos podemos dar el lujo de esperar, por lo que dentro de un loop, la forma de utilización es más o menos como la siguiente:

```
                if(ser0peek()==0x0D){                 // recibió <enter> ?
// o bien:      if(ser0rxbytes<14){                   // recibió 14 caracteres ?

                        // procesa el mensaje en ser0rxbuf
                        ...
                }
                // sigue en el loop
        }
}
```

El esquema desarrollado nos permite implementar timeouts dentro de la aplicación, de forma externa a las rutinas, y resulta de fácil inclusión en máquinas de estados.

En algunos casos, nuestro dispositivo debe estar atento a mensajes no solicitados, es decir, es el otro extremo el que inicia la comunicación cuando desea, y es necesario detectar mensajes incompletos y eliminarlos del buffer. Por ejemplo, para implementar una simple limpieza de buffer por timeout, es decir, cuando se está en espera de posibles mensajes y no se recibe un mensaje completo dentro de un cierto tiempo, podemos usar un esquema de timers como el descripto en CAN-069:

```
__bit  rxing0=0;                            // flag genérico para manejar el timer

#define r0timer CS_TIMERS[0]
// el timeout debería ser bastante mayor al tiempo razonable de transmisión de un paquete
// completo, digamos 300ms
#define TIMEOUT 30

        ...
        if(ser0rxbytes&&!rxing0){           // hay algo en el buffer y todavía no lo vi ?
          r0timer=TIMEOUT;                  // inicio timer
          rxing0=1;                         // ya lo vi
        }
        else if(rxing0&&!r0timer){          // el timer expiró, es un paquete incompleto
          ser0flush();                      // descarta lo que haya en el buffer
          rxing0=0;                         // reinicia el proceso de detección por timeout
        }

        // verifica y procesa el mensaje como en el ejemplo anterior
                ser0flush();                // procesado el mensaje, y borrado el buffer
                rxing0=0;                   // reinicia el proceso de detección por timeout
        ...
```

Las rutinas para la UART1 se encuentran en el archivo que acompaña a esta nota de aplicación.