Fundamentos de la Transmisión Síncrona

Una visión general de qué es, para qué sirve y cómo se utiliza.

Por: Diego "RedPic" Márquez - http://picmania.garcia-cuervo.net

¿Qué es una Transmisión Síncrona?

Para intentar acercarnos de forma clara a qué es una transmisión **Síncrona** voy a utilizar mi habitual forma indirecta de atacar las cosas y vamos a empezar por su antagonista por naturaleza: la transmisión **Asíncrona**.

Primero démosle un vistazo a su propio nombre y veamos qué significa esa palabreja de Asíncrona. Etimológicamente significa exactamente "sin reloj" o sea que no hay ninguna señal que marque los tiempos en que los datos deben leerse o están disponibles.

Esto significa que en una transmisión Asíncrona tanto la información transmitida como los tiempos en que ésta debe leerse son uno y todo va junto. El mejor ejemplo de este tipo de transmisión es la transmisión serie RS232. En esta forma asíncrona de transmitir información binaria cada bit es representado por un estado Alto o Bajo de la línea de transmisión durante un tiempo predeterminado. Este tiempo debe ser siempre el mismo, dentro de los márgenes de tolerancia normales y que son de aproximadamente de un 2% del valor nominal.

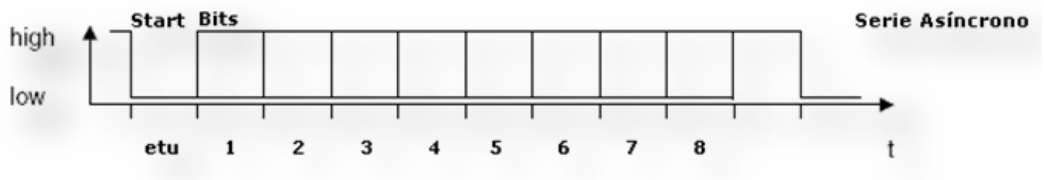
Fijaos por tanto que esto de Asíncrono no significa sin tiempo sino bien al contrario significa con tiempos perfectamente definidos y acordados de antemano ya que de otra forma no habría manera de poner de acuerdo al emisor y al receptor en cuanto a cuando está disponible cada bit para su lectura.

El sistema asíncrono funcionaría entonces así: En cuanto el receptor detecta el primer cambio de estado, una línea que pasa de Alto a Bajo por ejemplo en el RS232, sabe con total seguridad que tras cierto número de microsegundos transcurridos tendrá disponible el primer bit transmitido por el emisor, y tras otro igual número de microsegundos tendrá el segundo bit y... así hasta el último bit que debe recibir.

Se detecta el primer flanco de bajada y a partir de ahí solo debe mirar, cada plazo de tiempo acordado, en qué estado está la línea de transmisión, si alto o bajo, para asignar ese valor a cada uno de los bits a recibir.

De esta forma cuando decimos que una comunicación RS232 es a 8 bits y a 9600 baudios lo que estamos diciendo es que vamos a recibir 8 estados consecutivos de la línea de transmisión, separados cada uno de ellos 1/9600 segundos, o sea un estado cada 104 microsegundos, siendo el primero el estado que tenga tras los primeros 104 microsegundos transcurridos desde el primer flanco de bajada.

A 19.200 baudios en "tiempo" de cada bit será la mitad, 52 microsegundos, y a 4.800 baudios será el doble o sea 208 microsegundos. A esta unidad de tiempo la conocemos como el ETU de una transmisión, iniciales de *elementary time unit*. Abajo podemos ver una representación gráfica de esto que estamos tratando, la transmisión Asíncrona de un byte compuesto por 8 bits (un típico 8N1 9.600)



Una conclusión a la que podemos llegar después de expuesto todo esto sobre la transmisión Asíncrona es que es imprescindible saber a priori a qué velocidad vamos a recibir los distintos bits para ajustar nuestra rutina de recepción a dicha velocidad y mirar así la línea de transmisión en su momento justo, ni antes ni después, para recibir cada uno de los bits en el momento en que realmente les corresponde. Cualquier error en el cálculo dichos tiempos puede hacernos leer bits fantasmas, debido a que leemos dos veces un mismo bit o porque nos saltemos alguno de ellos.

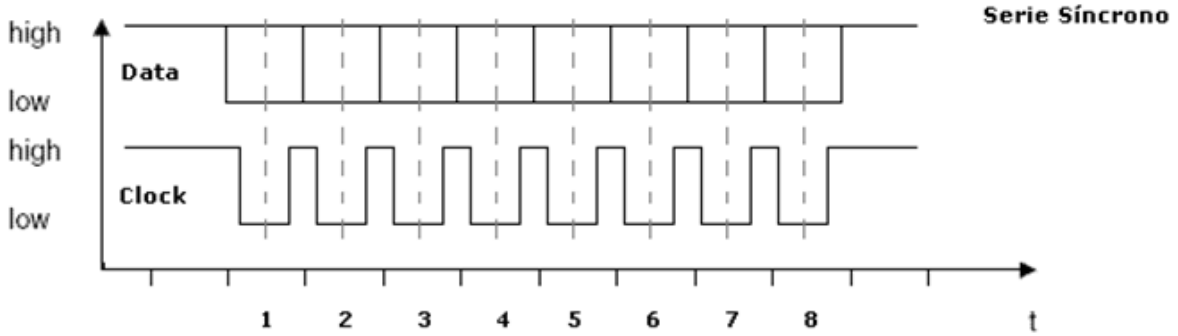
Y por fin llegamos a nuestra transmisión Síncrona de datos.

Síncrono significa "con reloj" y exactamente eso es lo que necesitamos, un reloj (o dicho en inglés un Clock). La transmisión síncrona necesita de dos líneas, una de datos sobre la que se van a representar los distintos estados de los bits a transmitir y una de reloj donde vamos indicando cuando está disponible cada bit en la línea de datos. Esta línea de reloj es la de "sincronización" entre ambos dispositivos, el emisor y el receptor de la transmisión.

De esta forma una transmisión síncrona consiste exactamente en poner el estado de un bit en la línea de datos, generamos un pulso de subida y bajada en la línea del reloj, ponemos otro estado de bit en los datos, volvemos a dar un pulso de subida y bajada en la del reloj... y así hasta completar el número de bits que deseemos transmitir.

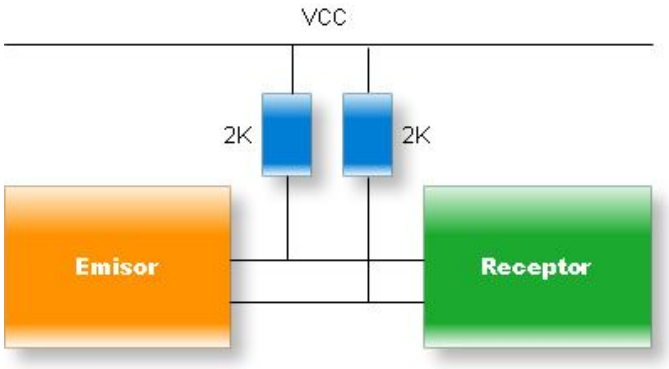
Esta forma de transmisión tiene una clara ventaja, y es que no es necesario poner de acuerdo en velocidad alguna a emisor y receptor de la transmisión. El emisor coloca su bit y genera el pulso en el reloj, el receptor detecta el reloj y mira el estado del bit, y así uno tras otro, a cualquier velocidad, a distinta velocidad cada bit, a toda la velocidad posible. Hay pulso significa hay dato, leo y a esperar otro pulso, más lento o más rápido es irrelevante solo es importante aquello de pulso-dato y a empezar de nuevo.

La única limitación es que al receptor le debe dar tiempo a leer el estado de cada bit tras detectar el pulso de reloj antes de que aparezca un nuevo pulso.



Notad que en estos ejemplo estamos utilizando la "lógica negativa" es decir que detectamos los pulsos estando la línea en alto cuando cae a bajo, o sea recibiendo primero un un flanco de bajada y después uno de subida para conformar un pulso.

Todo lo que estamos tratando sería exactamente igual con los pulsos al revés, en "lógica positiva" con el flanco de subida primero y el de bajada después. Esta configuración con las líneas en alto y dando pulsos negativos es la mas utilizada debido a la estabilidad y resistencia al "ruido" que tienen. Se consigue conectando una resistencia a VCC para que mantenga la línea a estado alto y nuestro emisor genera los pulsos poniendo la línea a GND. El receptor está constantemente recibiendo el estado alto y detecta cada pulso cuando pasa a bajo.



Bueno, y ahora vamos a ver cómo podemos implementar una simple comunicación Síncrona en C.

Funciones para Transmisión Síncrona

Las funciones para Transmitir de forma Síncrona que vamos a implementar son dos: Transmite_Bit_Clock_and_Data y Transmite_Byte_Clock_and_Data. La primera de ellas coloca el estado de un bit en la línea Data y genera un pulso en Clock, la segunda se encarga de extraer, bit a bit, el contenido de un byte (de 8 bits) y llamar a la función anterior.

```
#define OUT_CLOCK PIN_B0
#define OUT_DATA  PIN_B1

void Transmite_Bit_Clock_and_Data(int1 bit){

   // Coloca Data
   if( bit==0){
      output_high(OUT_DATA);
   }
   else{
      output_low(OUT_DATA);
   }
   // Genero pulso en Clock (500 microsegundos ó 2 Khz)
   delay_us(250);
   output_low(OUT_CLOCK);
   delay_us(250);
   output_high(OUT_CLOCK);
}

void Transmite_Byte_Clock_and_Data(char c){

   int8 i;
   int1 b;

   for(i=0;i<8;i++){
      b = bit_test(c,i);
      Transmite_Bit_Clock_and_Data(b);
   }
}
```

Para las funciones de recepción Síncrona vamos a usar el recurso de la Interrupción Externa de los PIC's, eligiendo estratégicamente el PIN del reloj (CLOCK) de forma que tengamos disponible una de estas interrupciones.

La interrupción externa la configuramos para detectar los flancos de bajada (ved la nota anterior sobre la "lógica negativa") De esta forma cada vez que se dispara la interrupción sabemos que tenemos disponible un bit en la línea de los datos (DATA). Lo recogemos sobre nuestro **recByte** y contamos uno más. Cuando lleguemos a 8 bits recogidos tenemos nuestro Byte completo y podemos indicarlo convenientemente poniendo a uno el flag **reccomplete**.

Cuando en **main** detectamos este **reccomplete** lo monitorizamos por el puerto serie y reiniciamos todo para recoger el siguiente byte.

Funciones para Recepción Síncrona

```
#define IN_CLOCK PIN_B0
#define IN_DATA  PIN_B1

char recByte=0;
int8 nextBit=0;
int1 reccomplete=0;

// INTERRUPCIÓN por EXT0 Clock CK (Data - Clock) --------

#int_ext
ext_isr() {

   int1 bit;

   bit=!input(IN_DATA);
   bit_clear(recByte,nextBit);
   if(bit==1) bit_set(recByte,nextBit);
   if(++nextBit=8){
      nextBit=0;
      reccomplete=1;
   }
}

// MAIN ------------------------------------------------

void main(void){

   ext_int_edge(0,H_TO_L);
   enable_interrupts(int_ext);
   enable_interrupts(global);

   do {
      // Lectura Completa
      if(reccomplete==1){
         readcomplete=0;
         putc(recByte);
      }
   } while (TRUE);
}
```