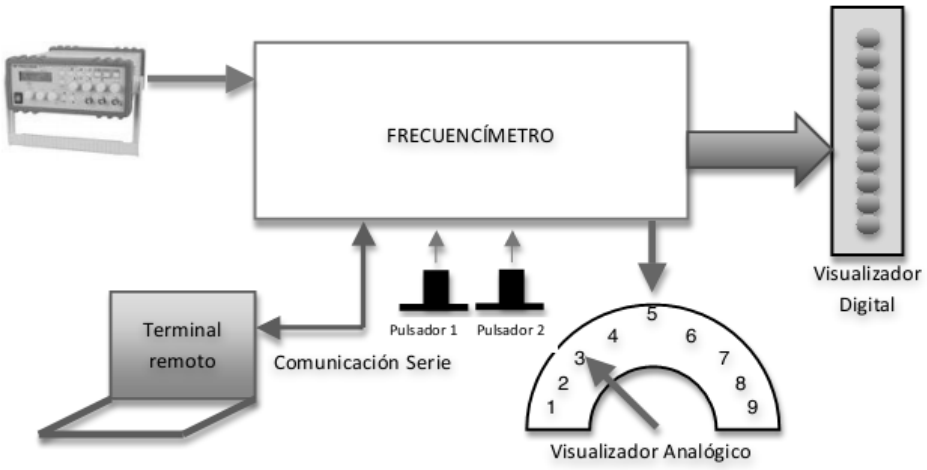
Práctica 8. Diseño de un Frecuencímetro



2ºA GITT UAH

Grupo 5:
Manuel Montoya Catalá.
Álvaro Santos Uriarte.

Índice:

Descripción general del sistema:

El sistema a diseñar es un frecuencímetro digital controlado de forma remota mediante una comunicación serie asíncrona.La entrada del sistema es una señal cuadrada periódica y dos pulsadores y como salida un doble sistema de monitorización de la medida al que además se incluye una mejora añadiendo otro sistema de visualización de la señal,los sistemas son:
-Indicador gráfico de leds.
-Envío remoto de la medida por puerto serie en ASCII.
-Indicador gráfico basado en un display de 7 segmentos(Mejora).

Especificación de la señal de medida
La señal se medirá en un rango superior al pedido como mejora (de o a 10e7 Hz)con posibilidad de que el usuario pueda elegir la escala que prefiera indicándola.Además se incluye una función adicional que detecta automáticamente la escala idónea de medida.

La visualización digital se realizará a modo de “vúmetro digital” utilizando 10 LEDs que visualicen el valor correspondiente en la escala seleccionada.Cuando el valor medido esté por debajo de la primera unidad, parpadeará el primer LED y cuando esté por encima de la décima unidad, parpadeará el último LED.
La visualización con el display de 7 segmentos indicará la primera cifra de la frecuencia medida.Ej:6000HZ--->6
La escala se visualizará con el pulsador 1 y se modificará con el pulsador 2.

El frecuencímetro responderá a una serie de comandos remotos con la posibilidad de enviar cada 500ms por el puerto serie el valor de la medida en Hz de manera que pueda ser visualizado en ASCII en un terminal remoto con una velocidad de comunicación de 19200 baudios.
Hemos diseñado como mejora una manera de poder enviar varias cadenas seguidas sin que sean sobreescritas añadiendo un “0” más al final de la cadena,que es el caracter que queda sobreescrito.También incorporamos una opción para enviar cadenas de caracteres en lugar de únicamente caracteres(opción C).Los caracteres que se van escribiendo también se visualizan por pantalla.
Remotamente se podrán enviar los siguientes comandos:
-M–Iniciar el envío de medidaS por el puerto serie.
-F-Finaliza el envío de medidas por el puerto serie.
-A-Ajuste automático de escala.
-H-Se muestran los comandos.
-L-Modo 10 leds.
-D-Modo 7 segmentos.
-C-Cambia a recepción de cadenas.
-(+/-)-Aumenta o disminuye precisión.

Descripción Hardware:

- 2.[20 al 29] para los 10 leds.
- 2.[20 al 26] para el modo 7 segmentos, siendo 'a' el 20, 'b' el 21 ....
- 2.11 para el pulsador que visualiza la señal.
- 2.12 para la señal.
- 2.13 para el pulsador de cambio de escala.

Descripción Software:

El programa consta de 5 archivos de extensión (.c) y 1 archivo de extensión (.h).Los archivos se explican brevemente a continuación:

Configuraciones.c: Archivo en el que se configuran los distintos pines y elementos que intervendrán en el programa:
void configUART0(int baudrate): Configura la comunicación serie UART0.
void configGPIO(unsigned int leds): Configura los distintos pines como entradas, salidas e interrupciones(EINT2 Y EINT3).
void configRIT(float tiempo_s):Configura el RIT.
int uart0_set_baudrate(unsigned int baudrate):Configura el baudrate.
Herramientas.c: Archivo que consta de distintas funciones que funcionan como herramientas para ser usadas en otros puntos del programa:
int conv10leds(int numero):Devuelve los pines del numero de leds indicado en numero
int conv7s(int numero):Le das el numero a encender y te devuelve los pines que lo referencian.
void tx_cadena_UART0(char *cadena, char *inicio_buffer, char *ptx_completa, int *ppos_cadena):Enviar cadena por UART0.
void conversor_IaS(int numero, char * cadnum):Conviertir entero a cadena y guardar en cadnum.
void conversor_SaI(char * cadnum, int *numero):Conviertir cadena a entero, recibir cadena y puntero a entero.

Interrup.c: Muestra las distintas funciones de interrupción:
void UART0_IRQHandler(void) :Función de interrupción de la comunicación serie.
void RIT_IRQHandler (void) :Incrementa el contador de RIT,LED y UART0.
void EINT2_IRQHandler(void)Función de Interrupción de EINT2 utilizada para aumentar los ciclos de señal.
void EINT3_IRQHandler(void):Función de interrupción de EINT3 utilizada para cambiar la escala.

Funciones.c: Archivo con las funciones del programa:
void config(unsigned int leds, float tiempo_s, int baudrate):Función de configuración general, para los pines, uart0 y RIT.
void estadoflags(int *pcUART0, char *pcomunUART0, int *pcLED, char *pconmuta):Función de activación de los flags de la comunicación asíncrona y los leds.
void obtfrec(float tiempo_s, float *pfrec, int *pcRIT,int *pcSIG, int calidad):Función para obtener la frecuencia.
void visionleds(int escala, float fecuencia, char *pconmuta)Función para configurar el modo de visión en el array de 10 leds.
void vision7seg(int escala, float fecuencia, char *pconmuta): Función para configurar el modo de visión en el display de 7 segmentos..
void controlUART0(char *prx_completa, int *pescala, char *pcomunUART0, char *penviofrec, char *pmodo, int *pcalidad, char *ptipo_rx)

Main.c: Archivo principal.

Definiciones.h: Contiene todas las definiciones del programa(estáticas,variables y funciones) para ser usadas por cualquier archivo de el mismo.

Código C:

configuraciones.c

```
#include "definiciones.h"
#include <LPC17xx.h>

void configGPIO(unsigned int leds){

  LPC_GPIO1->FIODIR |= leds;                // Configuracion de P1.xx definidos como salidas
  LPC_GPIO1->FIOPIN &= ~(leds);             //Apagamos todos los leds
  LPC_GPIO2->FIODIR &= ~(1 << 11);           //Configuracion de P2.11 como entrada

  LPC_PINCON->PINSEL4 |= 1 << (12*2);              // Configuracion del P2.12 como EINT2
  LPC_SC->EXTMODE |= 1<< 2;                 // Interrupción activa por flanco de bajada  -->
EXTMODE.2
  NVIC->IP[EINT2_IRQn] = 0x02 << 3;                // Configuramos prioridad 2 a la interrupcion
EINT2 (IRQ20)
  NVIC->ISER[0] = 1 << EINT2_IRQn;           // Habilitar la interrupcion EINT2  --> ISER0.20

  LPC_PINCON->PINSEL4 |= 1 << (13*2);              // Configuracion del P2.13 como EINT3
  LPC_SC->EXTMODE |= 1<< 3;                          // Interrupción activa por flanco de bajada
        --> EXTMODE.3
  NVIC->IP[EINT3_IRQn] = 0x03 << 3;         // Configuramos prioridad 3 a la interrupcion EINT3
(IRQ21)
  NVIC->ISER[0] = 1 << EINT3_IRQn;                 // Habilitar la interrupcion EINT3  --> ISER0.21

}

void configUART0(int baudrate) {

    LPC_PINCON->PINSEL0 |= (1 << 4);        // Configuracion del P0.2 como RX0
       LPC_PINCON->PINSEL0 |= (1 << 6);        // Configuracion del P0.3 como TX0
    LPC_UART0->LCR |= CHAR_8_BIT|STOP_1_BIT|PARITY_NONE;   // Tipo de transmision (8 bits/dato, sin
pariad, y 1 bit de stop)

    uart0_set_baudrate(baudrate);                 // Configura Velocidad transmision

    LPC_UART0->IER |= THRE_IRQ_ENABLE;      // Habilita la interrupcion para TX
       LPC_UART0->IER |= RBR_IRQ_ENABLE;           // Habilita la interrupcion para RX
    NVIC_EnableIRQ(UART0_IRQn);             // Habilita la interrupcion UART0(for Cortex-CM3 NVIC)

}

void configRIT( float tiempo_s ) {

  unsigned int ticks;
  unsigned int velocidadRIT =  25000000;
  ticks = tiempo_s * velocidadRIT;          //Calcula el numero de ciclos necesarios para esperar
tiempo_s segundos
  LPC_SC->PCONP       |= PCONP_RIT_ON;      // Power Control: Alimenta el RIT

  LPC_RIT->RICTRL     &= ~RIT_RITEN;        // Deshabilita RIT para programarlo, pone un 0 en
RIT_RITEN
  LPC_RIT->RICTRL     &= ~RIT_RITENCLR;     // Pone un 0 en RIT_RITENCLR
  LPC_RIT->RICTRL     &= ~RIT_RITINT;      // Bit que cuando a 1 borra la fuente de interrupcion
puesto a 0

  LPC_RIT->RICOUNTER = 0;                       // Inicializa contador del RIT a 0
  LPC_RIT->RICOMPVAL = ticks;       // Da valor al registro de comparacion      RITCOMPVAL
  NVIC_EnableIRQ(RIT_IRQn);                     // Habilita la interrupcion del RIT

  LPC_RIT->RICTRL     |= RIT_RITENCLR;      // Pone un 1 en RIT_RITENCLR
  LPC_RIT->RICTRL     |= RIT_RITINT;
  LPC_RIT->RICTRL     |= RIT_RITEN;         // Habilita RIT, pone un 1 en  RIT_RITEN
}

int uart0_set_baudrate(unsigned int baudrate) {
    int errorStatus = -1; //< Failure

    // UART clock (FCCO / PCLK_UART0)
   // unsigned int uClk = SystemFrequency / 4;
    unsigned int uClk =SystemCoreClock/4;
    unsigned int calcBaudrate = 0;
    unsigned int temp = 0;

    unsigned int mulFracDiv, dividerAddFracDiv;
    unsigned int divider = 0;
    unsigned int mulFracDivOptimal = 1;
    unsigned int dividerAddOptimal = 0;
```

```
    unsigned int dividerOptimal = 0;

    unsigned int relativeError = 0;
    unsigned int relativeOptimalError = 100000;

    uClk = uClk >> 4;  // Divide por 16

     // Ecuacion:  BaudRate = uClk * (mulFracDiv/(mulFracDiv+dividerAddFracDiv) / (16 * DLL)

     // Los valores de mulFracDiv y dividerAddFracDiv deberian estar entre los rangos:
     // 0 < mulFracDiv <= 15, 0 <= dividerAddFracDiv <= 15

    for (mulFracDiv = 1; mulFracDiv <= 15; mulFracDiv++) {
        for (dividerAddFracDiv = 0; dividerAddFracDiv <= 15; dividerAddFracDiv++) {
            temp = (mulFracDiv * uClk) / (mulFracDiv + dividerAddFracDiv);

            divider = temp / baudrate;
            if ((temp % baudrate) > (baudrate / 2))
                divider++;

            if (divider > 2 && divider < 65536) {
                calcBaudrate = temp / divider;

                if (calcBaudrate <= baudrate) {
                    relativeError = baudrate - calcBaudrate;
                } else {
                    relativeError = calcBaudrate - baudrate;
                }

                if (relativeError < relativeOptimalError) {
                    mulFracDivOptimal = mulFracDiv;
                    dividerAddOptimal = dividerAddFracDiv;
                    dividerOptimal = divider;
                    relativeOptimalError = relativeError;
                    if (relativeError == 0)
                        break;
                }
            }
        }

        if (relativeError == 0)
            break;
    }

    if (relativeOptimalError < ((baudrate * UART_ACCEPTED_BAUDRATE_ERROR) / 100)) {

        LPC_UART0->LCR |= DLAB_ENABLE;      // importante poner a 1
        LPC_UART0->DLM = (unsigned char) ((dividerOptimal >> 8) & 0xFF);
        LPC_UART0->DLL = (unsigned char) dividerOptimal;
        LPC_UART0->LCR &= ~DLAB_ENABLE;     // importante poner a 0

        LPC_UART0->FDR = ((mulFracDivOptimal << 4) & 0xF0) | (dividerAddOptimal & 0x0F);

        errorStatus = 0; //< Success
    }

    return errorStatus;
}
```

herramientas.c

```
#include <LPC17xx.h>
#include "definiciones.h"
#include <string.h>


 int conv10leds(int numero){                //Devuelve los pines del numero de leds indicado en
numero
       int i;
              int result = 0 ;
              int exp = 1;
              for (i=0;i<numero;i++){              //Calcula el numero equivalente para "numero"
de 1's seguidos
                 result += exp;
                 exp *=2; }
              return (result<<20);
   }


int conv7s(int numero){                                          //Le das el numero a encender y
te devuelve los pines k lo referencian
    switch (numero){
```

```
          case 0:
                 return (0x3F<<20);
           break;
          case 1:
                 return (0x06<<20);
           break;
                case 2:
                 return (0x5B<<20);
           break;
                case 3:
                 return (0x4F<<20);
           break;
                case 4:
                 return (0x66<<20);
           break;
                case 5:
                 return (0x6D<<20);
           break;
                case 6:
                 return (0x7D<<20);
           break;
                case 7:
                 return (0x07<<20);
           break;
                case 8:
                 return (0x7F<<20);
           break;
                case 9:
                 return (0x67<<20);
           break;
                case 10:                                    //Selecciona todos
                 return (0x7F<<20);
           break;

       };
}


 void tx_cadena_UART0(char *cadena, char *inicio_buffer, char *ptx_completa, int *ppos_cadena){
         //Envia cadena por UART0
  int i=0;
  for(i=0;i<=strlen(cadena);i++){ // Por cada caracter que se quiere enviar
  *(inicio_buffer + (*ppos_cadena) + i)=cadena[i];}    //Se guarda la cadena en la posicion dada por
ppos_cadena para que no se sobreescriban
  *(inicio_buffer + (*ppos_cadena) + i)=0;                    //Finalizamos cadena con un 0 para
asegurarnos
  *(inicio_buffer + (*ppos_cadena) + i + 1)=0;                //Ponemos un segundo 0 para nuestro
protocolo
  (*ppos_cadena)+= strlen(cadena) + 1;             //Indicamos que la primera posicion libre esta
despues del 0 final de
                                                                                    //la ultima cadena
escrita, asi como indica el protocolo, con la siguiente cadena
                                                                                    //se sobreescribira
el segundo 0 puesto por la ultima cadena
  if((*ptx_completa)==1){         //Si todas las cadenas del buffer se han enviado habra que
empezar el envio
  LPC_UART0->THR=(*inicio_buffer);}  // Pone el primer dato en el buffer de tranmision TX para que
cuando se transmita
                                  // segenere la interrupcion de dato transmitido y la atencion a la
interrupcion
                                                             // se ocupara de enviar el resto de la
cadena
  (*ptx_completa)=0;                 // Indicamos que se estan transmitiendo datos
}
//Protocolo:    cadena1..
//                           cadena1.cadena2..
//              cadena1.cadena2.cadena3..
// Cuando se envien todas las cadenas pendientes, el puntero volvera a apuntar al principio del
buffer
// Sobreescribiendo asi las cadenas ya enviadas




  void conversor_IaS(int numero, char * cadnum){                    //Convierte entero a cadena y la
guarda en cadnum
               int i = 0, j = 0;
```

```
                char aux[10];
                char convertir = 1;
                while(convertir){                                       //Mientras quede cifra por
convertir
                   aux[i] = numero % 10 + 48;                           //Igualamos la posicion i del
array a la unidad mas pequeña de la cifra
                   numero = numero / 10;                                //Dividimos por 10 la cifra
                   i++;
//Aumentamos la posicion donde gruardarmos la proxima cifra
                   if (numero == 0 ){                                          //Si ya no hay mas
numero por corvertir se indica pasa salir
                   convertir = 0; }}

                for (j=0; j<i ;j++){                                            //Copiamos en la
cadena dada, la obtenida dada la vuelta
                    *(cadnum+j) = aux[i-j-1];}
                     *(cadnum+i) = 0;                          //Ponemos un 0 en la ultima como fin
de cadena
             }

   void conversor_SaI(char * cadnum, int *numero){                  //Convierte cadena a entero,
recibe cadena y puntero a entero
            int i = 0;
               int aux = *numero;                    //Auxiliar por si acaso no podemos traducir
el valor
               int n = strlen(cadnum);
       //Cifras que tiene el caracter
               int mul = 1;
       //Multiplicador de la cifra
               *numero = 0;                                                    //Igualamos a 0
el numero que nos dan por si acaso tenia otro valor
               for(i=0; i < n ; i++) {
                  if((*(cadnum +(n-1)-i) >='0' )&&(*(cadnum +(n-1)-i) <= '9')){  //Si el caracter
va del 0 al 9
                    *numero += (*(cadnum +(n-1)-i)-48)*mul;                  //Le sumamos al
numero esa cifra elevada a lo que toke
                    mul *= 10;   }                                      //Recordar que 48 es el
caracter 0
                     else {
       //Si algun graciosete nos da un caracter no valido
                       *numero = aux;                                 // Le devolvemos al int dado
su valor inicial
                       tx_cadena_UART0("Da un numero valido graciosete\n\r", buffer_tx[0],
&tx_completa, &pos_cadena);          //Retorno de carro
                       return; }

                    }
             }
```

interrup.c

```
#include "definiciones.h"
#include <LPC17xx.h>

void UART0_IRQHandler(void) {

   switch(LPC_UART0->IIR&0x0E) {     //Cuando hay una interrupcion UART0
                                                                   //Hace un AND entre los flags de
interrupciones y el numero en binario 1111
                                                                   //Para ver si alguno esta
activado y por tanto atender a esa interrucion
                                                                   //RECEPCION
   case 0x04:                                 //Si se ha recibido un dato en el buffer

    if(tipo_rx == 0){                        //Si la transmision es caracter a caracter
         pcad_rx =buffer_rx;                 //Se pone el puntero de recepcion pcad_rx al comienzo
del array buffer
     *pcad_rx=LPC_UART0->RBR;        //lee el dato recibido y lo almacena en la direccion en memoria
a la que apunte
                                     //pcad_rx quien apunta al array buffer_rx por lo que se
guarda en el mismo
         rx_completa = 1; }              //Indica que se ha recibido el comando y hay que atenderlo



       if(tipo_rx == 1){                         //Si la transmision es por cadena, guardaremos los
caracter recibidos palabra
                                                                   //por palabra, es decir, cada
palabra sera guardada como una cadena
```

```
              *pcad_rx = LPC_UART0->RBR;     //Guardamos el caracter transmitido en la posicion que
apunta pcad_rx

              if(*pcad_rx == 32){                  //Si el caracter recibido es un espacio (32),
indicando el final de una palabra
                                          //y por tanto el final de la cadena que estamos
guardando
           *pcad_rx = 0; }                   //Sobreescrimos el espacio " " con un 0, asi se
guardara la palabra como cadena

                                                                 //Codigo que se ocupa de que veas lo
que escribes
                  *(pcad_rx + 1) = 0;     //Guardamos en la siguiente posicion el caracter 0 para
finalizar la cadena
                                                          //y poder enviar el caracter asi vemos lo
que escribimos
                                                                    //El siguiente caracter si lo hay
reescribira el 0
                tx_cadena_UART0(pcad_rx, buffer_tx[0], &tx_completa, &pos_cadena);       //La envia
el caracter    que se ha escrito
                                                                                        //Asi el tio lo
podra ver

             if(*pcad_rx == 8){              //El caracter 8 es el de borrar un caracter, aunque en
verdad ese solo mueve
                                          //el puntero de escribir a la salida hacia la izquierda
                                                                    //Por lo que tendremos que borrar
el caracter nosotros mismos sobreescribiendolo
                                                                    //con un espacio y despues volver
atras
                  if(pcad_rx > buffer_rx){   //Si no estamos al principio del buffer de reccepcion
lo que quiere decir
                                                 //que se ha enviado un caracter con anteriorridad
                  tx_cadena_UART0(" ", buffer_tx[0], &tx_completa, &pos_cadena);       //Enviamos
un espacio que simplemente
                                                                         //sobreescribira el ultimo
caracter leido en la pantalla

                //Esta cacter no esta logicamente en el buffer de recepcion
                     tx_cadena_UART0(pcad_rx, buffer_tx[0], &tx_completa, &pos_cadena); //Hace k
el puntero vuelva atras
                      pcad_rx -= 2;}}             //Atrasamos el puntero de recepcion dos veces,
una por el caracter que queremos
                                            //borrar y otro por el caracter (8) Asi el nuevo
cacter sobreescribira
                                                                    //al caracter que hemos kerido
borrar



           if(*pcad_rx ==13){                   //Si el caracter recibido es [ENTER] (ya esta
guardado en buffer)
                *(pcad_rx) = 0;                 //Sobreescribimos el [ENTER] con un cero indicando
el fin de la cadena
              pcad_rx = buffer_rx;        //Recolocamos el puntero al principio del buffer
                 rx_completa = 1;                  //Indica que se ha recibido el comando y hay
que atenderlo
                 tx_cadena_UART0("\r\n", buffer_tx[0], &tx_completa, &pos_cadena);} //Escribimos un
salto de linea y retorno de carro

           else{                             //Si no se ha acabado la cadena
               pcad_rx++; }                      //Aumentamos en 1 el puntero para que el siguiente
caracter se guarde
       }                                                        //en la siguiente posicion
       break;


                                                                          //TRANSMISION\\ (ya
iniciada por la funcion tx_cadena_UART0 )
   case 0x02:                                       //Si el registro de transmision esta vacio por
lo que se ha enviado un dato
        pcad_tx++;                                      //Se incrementa el puntero de la cadena para
que apunte al siguiente dato a transmitor
     if((*pcad_tx)!=0){                                 //Si la cadena a enviar no ha acabado por lo
que hay mas datos por transmitir
       LPC_UART0->THR=(*pcad_tx);        //Se carga el nuevo dato (caracter) en el registro de
transmision,
                                        //cuando se transmita generara otra interrupcion del
mismo tipo al haberlo enviado
```

```
            }                                                       //y esta funcion seguira
asi hasta que sea el fin de la cadena

          else {                                                  //Si se llega al final de una
cadena (un 0 )
           pcad_tx++;                          //Incrementamos al siguiente punto
               if((*pcad_tx)!=0){                          //Si el siguiente caracter no es 0,
significa que hay mas cadenas por enviar
                                        //pcad_tx ya apunta al principio de la siguiente cadena
               LPC_UART0->THR=(*pcad_tx); }       //Cargamos el nuevo dato para que continue el
ciclo de interrupciones

               else {                                              //Si tambien es 0 incica segun
nuestro protocolo el final de las cadenas
                                             //Por lo que todas las cadenas en el buffer han
sido enviadas
               tx_completa=1;                              //Indicamos que no se esta
transmitiendo nada
           pcad_tx = buffer_tx[0];                  // Ponemos el puntero al principio del bufer
de salida
               pos_cadena = 0;       }                           //Inicializamos posiciones para
que las nuevas cadenas
               }                                        //Sobreescriban a las anteriores


     break;
   };                                    //Fin del Switch
  }



void RIT_IRQHandler (void) {
  LPC_RIT->RICTRL |= RIT_RITINT;     // Borra fuente de la interrupción (sino se ejecutaria
permanentemente)
  (*pcRIT)++;                                       //Incrementa en 1 el contador del RIT
  (*pcLED)++;                                       //Incrementa en 1 el contador del conmutador
  (*pcUART0)++;                                      //Incrementa en 1 el contador del UART0
}                                                       //No poner mas codigo para no sobrecargar ya
que es una interrupcion muy usual



void EINT2_IRQHandler(void)   {           //Funcion de interrupcion EINT2
                                                        //Conectado a la señal para contar
ciclos
  LPC_SC->EXTINT |= (1 << 2);         // Borrar el flag de la EINT2 --> EXTINT.2
  (*pcSIG)++;                                           //Aumenta en 1 el contador de ciclos
}

void EINT3_IRQHandler(void){         //Funcion de interrucion EINT3
                                                        //Utilizada para cambiar la escala
  LPC_SC->EXTINT |= (1 << 3);        // Borrar el flag de la EINT3 --> EXTINT.3
  if(*pescala == 10000000){  *pescala = 100;}       //Si se ha llegado a la escala final, se
inicializa
  else {(*pescala) *= 10;}                        //Sino se pasa a la siguiente escala
}
```

funciones.c

```
#include <LPC17xx.h>
#include "definiciones.h"
#include <string.h>

void config(unsigned int leds,  float tiempo_s, int baudrate){

 configGPIO(leds);
 configRIT(tiempo_s);
 configUART0(baudrate);
 tx_cadena_UART0("Bienvenidos al frecuencimetro: \n\r", buffer_tx[0], &tx_completa, &pos_cadena);
 tx_cadena_UART0("HAPPY XMAS !!!\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
 tx_cadena_UART0("Teclear H para ver los comandos \n\r", buffer_tx[0], &tx_completa, &pos_cadena);

 }



void estadoflags(int *pcUART0, char *pcomunUART0, int *pcLED, char *pconmuta){
  if((*pcUART0) >= 5000){                                 //Si han pasado 500ms = 0,5s
 (*pcomunUART0) = 1;                                       //Activamos el indicador de
comunicacion asincrona
  (*pcUART0) = 0; }                                       //Ponemos a 0 el contador del UART0
```

```
  if((*pcLED) >= 1000){                                          //Si han  pasado 200ms = 0,2s
  (*pconmuta) = 1;                                               //Activamos el indicador de
comunicacion asincrona
  (*pcLED) = 0; }                                              //Ponemos a 0 el contador del UART0
 }

void obtfrec(float tiempo_s, float *pfrec, int *pcRIT,int *pcSIG, int calidad){
        int nc = 1;                                                    //Numero de ciclos de la señal
a la que esperamos
        float amplitud = 0;                              //Variable en la que pondremos la amplitud de
la señal
                                                                                    //Iniciamos a 1
como base para la siguiente instruccion
              while(amplitud == 0){                 //De esta funcion no se sale hasta que no
calcula la amplitd de la señal
                                                                                  //con un minimo
de 5 ciclos de accuaricy
       (*pcRIT) = 0;                                    //Ponemos nuestro contador del RIT a 0
       (*pcSIG) = 0;                            //Ponemos nuestro contador de la señal a 0


           while ((*pcSIG) < nc){               //Esperamos a "nc" interrupciones de la señal
                       if (*pcRIT >= 5000*nc){         //Si la frecuencia va a ser menor que 5 HZ
                          *pfrec = 0;                //Damos por hecho que no hay frecuencia
                    tx_cadena_UART0("N0 SIGNAL\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
//Decimos por UART0 que no hay señal
                                return; }}                                  //salimos de la funcion

          amplitud = 1000*(tiempo_s * (*pcRIT)/nc);//Obtenemos el periodo a partir de los datos
                                                                                                  //
EL 1000 es para evitar numeros decimales tan pequeños
                                                                                                  //
que no se guardarian en el float
          if((*pcRIT) <= calidad){                                 //Si no tenemos "calidad" ciclos
de exactitud
             nc *= 10;                                                        //Esperamos 10 veces
mas interrupciones de la señal
                                                       //A tan alta frecuencia dicho tiempo es
despreciable
                amplitud = 0;       }                                              //Indicamos que la
acuare no la pedida

           }
       *pfrec = 1000/amplitud;                            //Escribimos el resultado en la variable
externa "frecuencia"
       }


 void visionleds(int escala, float fecuencia, char *pconmuta){  //Le damos la escala, la frecuencia
y el puntero al flag
       int i = 0;
          int div = 100;                                   //variable para ver en que escala estamos

          if ((LPC_GPIO2->FIOPIN & (1<<11))==0){        //Si esta pulsado el interruptor que muestra
la escala
              for(i=0; escala%div==0; i++){                 //Mientras el resto de dividir por 10
sea 0
                   div *= 10;
//Multiplicamos el divisor por 10
                    }
//Cuando el resto sea 1, nos dara el numero de la escala (1 al 7 )
                                                                  //Siendo 100 -> i = 1
                    LPC_GPIO1->FIOPIN  &= ~ (leds - conv10leds(i));                    //Apaga
los leds que no usaremos
              LPC_GPIO1->FIOPIN  |= conv10leds(i);                                    //Enciende
leds correspondientes
                    LPC_GPIO1->FIOPIN  &= ~ (conv10leds(i-1));             //Apaga los leds por
detras del ultimo

                            //Para que sepamos que estamos viendo la escala
                    }



          else{                                                    //Si el pulsador no esta
pulsado
       int numactivos =  frecuencia / (escala/10 );         //Calculamos numero de leds a encender

          //En numacticos solo estara la parte entera de la division
```

```
          if((numactivos==0) && (*pconmuta == 1)){             //Si por debajo de frecuencia minima
y ha pasado el tiempo necesario
                LPC_GPIO1->FIOPIN  &= ~ (leds - conv10leds(1));            //Apaga los leds que no
usaremos
                LPC_GPIO1->FIOPIN ^= conv10leds(1);                                      //Conmuta
el led 1
                 (*pconmuta)=0;                                           //Apagamos el flag de
conmutacion
                                }
          if((numactivos>10) && (*pconmuta == 1)){               //Si por encima de frecuencia
maxima y ha pasado el tiempo necesario
                     LPC_GPIO1->FIOPIN  |= (conv10leds(9));          //Encendemos todos los leds
menos el ultimo que conmutara
                     LPC_GPIO1->FIOPIN ^= 1<<29;                          //Conmuta led 10
                     *pconmuta=0;
//Apagamos el flag de conmutacion
                            }

          if((numactivos<= 10) && (numactivos>0)){               //Si la frecuencia esta en el
rango
          LPC_GPIO1->FIOPIN  &= ~ (leds - conv10leds(numactivos));                //Apaga los leds
que no usaremos
          LPC_GPIO1->FIOPIN  |= (conv10leds(numactivos));         //Enciende los leds que indican la
frecuencia

          }
        }
        }




 void vision7seg(int escala, float fecuencia, char *pconmuta){  //Le damos la escala, la frecuencia,
el puntero al flag, y la variable con los pines

          int i = 0;
          int div = 100;                                  //variable para ver en que escala estamos

          if ((LPC_GPIO2->FIOPIN & (1<<11))==0){        //Si esta pulsado el interruptor que muestra
la escala
               for(i=0; escala%div==0; i++){                 //Mientras el resto de dividir por 10
sea 0
                    div *= 10;
//Multiplicamos el divisor por 10
                     }
//Cuando el resto sea 1, i contentra el numero de la escala
                     LPC_GPIO1->FIOPIN  &= ~ (segment7 - conv7s(i));  //Apaga los leds que no
queremos encender
               LPC_GPIO1->FIOPIN  |=conv7s(i);           //Enciende leds necesarios
                  }

          else{                                                    //Si el pulsador no esta
pulsado
       int numactivos =  frecuencia / (escala/10 );         //Calculamos numero de leds a encender

          //En numacticos solo estara la parte entera de la division
          if((numactivos==0) && (*pconmuta == 1)){             //Si por debajo de frecuencia minima
y ha pasado el tiempo necesario
           LPC_GPIO1->FIOPIN  &= ~ (segment7 - conv7s(0));  //Apaga los leds que no queremos
encender
                 LPC_GPIO1->FIOPIN ^= conv7s(0);                                    //Conmuta el  0
                 (*pconmuta)=0;                                          //Apagamos el flag de
conmutacion
                                }
          if((numactivos>10) && (*pconmuta == 1)){               //Si por encima de frecuencia
maxima y ha pasado el tiempo necesario
                LPC_GPIO1->FIOPIN  &= ~ (segment7 - conv7s(9));  //Apaga los leds que no queremos
encender
                LPC_GPIO1->FIOPIN ^= conv7s(9);                              //Conmuta el 9
                     *pconmuta=0;
//Apagamos el flag de conmutacion
                            }

          if((numactivos<= 10) && (numactivos>0)){               //Si la frecuencia esta en el
rango
           LPC_GPIO1->FIOPIN  &= ~ (segment7 - conv7s(numactivos));  //Apaga los leds que no
queremos encender
               LPC_GPIO1->FIOPIN  |=conv7s(numactivos);              //Enciende el numero
          }
```

```
        }
        }



 void controlUART0(char *prx_completa, int *pescala, char *pcomunUART0, char *penviofrec, char
*pmodo, int *pcalidad, char *ptipo_rx){
               char cadconv[10];                                                  //Cadena para usar con
la funcion conversor de int a cadena
        int i = 0;
        int mul = 1;                                                              //Variable para
modificar la escala
               char *pcadconv = cadconv;                                          //Puntero a la cadena
para enviar a la funcion

               if ((*prx_completa)==1)      {                //Si se ha recivido un nuevo comando,
le atendemos
               (*prx_completa)=0;                //Apagamos el flag de atencion al comando

        if(*ptipo_rx == 0) {                                                      //Si recibimos comandos
por caracteres
             if((buffer_rx[0]>='1')&&(buffer_rx[0]<='7')){      //Si el dato dado es un numero del 1
al 7
               for(i=0; i<=(buffer_rx[0]-48); i++){               //Mientras no encontremos la
escala (48 = '0')
                    mul *= 10;
//Multiplicamos el divisor por 10
                   }
                *pescala  = mul;                  //Igualamos la escala al valor querido

                 conversor_IaS(*pescala, pcadconv);                                    //Convierte
la escala a Cadena
                 tx_cadena_UART0("Valor Maximo de Escala: ", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0(pcadconv, buffer_tx[0], &tx_completa, &pos_cadena);        //La
envia (la escala)
                 tx_cadena_UART0("\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
//Salto de linea
                 return;}
                                     //Salimos de la funcion

        switch (buffer_rx[0]){                                                   //Vemos el valor del
primer dato del buffer


         case 'H':                                                                //Si es H damos las
instrucciones disponibles
                 tx_cadena_UART0("Listado de Comandos:\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0("(1 al 7) -> Establece Escala: min(10 - 100 Hz) \n\r",
buffer_tx[0], &tx_completa, &pos_cadena);
                 tx_cadena_UART0("H -> Muestra lista Comandos \n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0("M -> Envia Frecuencia cada 0,5s\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0("F -> Para de Enviar Frecuencia \n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0("L -> Establece Modo 10 leds\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0("D -> Establece Modo Display 7 segmentos\n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
                 tx_cadena_UART0("I -> Envia Estado Frecuencimetro\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0("A -> Autoajusta la Escala\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0("C -> Cambia a Recepcion de Cadena \n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
                 tx_cadena_UART0("+ -> Aumenta Precision (10 - 1000)\n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
                 tx_cadena_UART0("- -> Disminuye Precision (10 - 1000) \n\r", buffer_tx[0],
&tx_completa, &pos_cadena);

           break;
                case 'M':                                                                 //Si es M
enviamos fecuencia
           *penviofrec = 1;                                          //Enviamos la frecuencia
                  tx_cadena_UART0("Medidas Frecuencia\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
           break;
```

```
              case 'F':                                                    //Si es F paramos de
enviar frecuencia
           *penviofrec = 0;                                              //Paramos la frecuencia
                 tx_cadena_UART0("Envios de Frecuencia Parados\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
           break;
                case 'L':                                                  //Si es L lo ponemos
en modo leds
           *pmodo = 1;
                 tx_cadena_UART0("Modo 10 leds\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
                 break;
         case 'D':                                                  //Si es D lo ponemos en modo
display 7s
           *pmodo = 2;
                 tx_cadena_UART0("Modo Display 7 segmentos\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
           break;
            case 'C':                                                  //Si es C cambiamos a
modo de recepcion de cadenas
           *ptipo_rx = 1;
                 tx_cadena_UART0("--------------------------------------\n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
                 tx_cadena_UART0("Recepcion de Comandos por Cadenas. \n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
           tx_cadena_UART0("Teclear: Comandos + [ENTER] Para ver comandos\n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
                 break;
                case 'A':                                                  //Si es A autoajusta
la escala para que la frecuencia este en el rango
                 *pescala = 100;                                       //Empezamos por la escala
mas baja
           while( frecuencia >= *pescala){      //Mientras la frecuencia sea mayor que la escala
                       *pescala *= 10; }                     //Multiplicamos por 10 la escala
                 conversor_IaS(*pescala, pcadconv);
        //Convierte la escala a Cadena
                 tx_cadena_UART0("Valor Maximo de Escala: ", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0(pcadconv, buffer_tx[0], &tx_completa, &pos_cadena);        //La
envia (la escala)
                 tx_cadena_UART0("\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
//Salto de linea
                 break;

                 case '+':                                                 //Si es + Aumentamos
la calidad
                  if(*pcalidad< 1000 ){                                //Si la calidad       es
menor que la maxima
          (*pcalidad)*=10;    }                                    //Aumentamos la calidad, aumenta
tambien el tiempo de espera
                 else {
                   tx_cadena_UART0("!!!Precision Maxima!!!\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);  }
                 break;

                 case '-':                                                 //Si es + Aumentamos
la calidad
                  if(*pcalidad> 1 ){                                   //Si la calidad       es mayor que
la miniama
          (*pcalidad)/=10;    }                                    //Disminuimos
                  else {
                   tx_cadena_UART0("!!!Precision Minima!!!\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);  }
                 break;

               case 'I':                                                   //Si es I nos da la
informacion del estado
            tx_cadena_UART0("Informacion del Frecuencimetro: \n\r", buffer_tx[0], &tx_completa,
&pos_cadena);

                 conversor_IaS(*pescala, pcadconv);
        //Convierte la escala a Cadena
                 tx_cadena_UART0("Valor Maximo de Escala: ", buffer_tx[0], &tx_completa,
&pos_cadena);
                 tx_cadena_UART0(pcadconv, buffer_tx[0], &tx_completa, &pos_cadena);        //La
envia (la escala)
                 tx_cadena_UART0("\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
//Salto de linea

                 switch(modo){
                case 1:
```

```
                  tx_cadena_UART0("Modo de Trabajo: 10 leds\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
                break;
                  case 2:
                   tx_cadena_UART0("Modo de Trabajo: Display 7 segmentos \n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
                break;
               };
                  conversor_IaS(frecuencia, pcadconv);
                        //Convierte la frecuencia a Cadena
                  tx_cadena_UART0("Frecuencia: ", buffer_tx[0], &tx_completa, &pos_cadena);
                  tx_cadena_UART0(pcadconv, buffer_tx[0], &tx_completa, &pos_cadena);        //La
envia (la frecuencia)
                  tx_cadena_UART0("\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
        //Retorno de carro

                  conversor_IaS(*pcalidad, pcadconv);
                        //Convierte la frecuencia a Cadena
                  tx_cadena_UART0("Calidad: ", buffer_tx[0], &tx_completa, &pos_cadena);
                  tx_cadena_UART0(pcadconv, buffer_tx[0], &tx_completa, &pos_cadena);        //La
envia (la frecuencia)
                  tx_cadena_UART0("\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
        //Retorno de carro


            break;
                  default:
                   tx_cadena_UART0("Comando desconocido\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);   //Si no coincide
                   break;

                  };                              //Fin del Switch    de recepcion
          return;   }                                    //Fin de la recepcion de comandos por
caracteres

   if(*ptipo_rx == 1) {                                                  //Si recibimos comandos
por cadenas
                  if(strcmp(buffer_rx, "Caracteres") == 0){ //Si la cadena enviada es lo que pone
entre parentesis
                        *ptipo_rx = 0;                                                  //Iniciamos
recepcion por caracteres
                tx_cadena_UART0("--------------------------------------\n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
                   tx_cadena_UART0("Recepcion de Comandos por Caracteres. \n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
             tx_cadena_UART0("Teclear H para ver los comandos \n\r", buffer_tx[0], &tx_completa,
&pos_cadena); }

               else if(strcmp(buffer_rx, "Escala") == 0){
                  conversor_SaI((buffer_rx + strlen(buffer_rx) +1),pescala);}     //  (buffer_rx +
strlen(buffer_rx)+1) contiene la direccion

                                  // a la siguiente palabra enviada por la terminal

                  else if(strcmp(buffer_rx, "Precision") == 0){
                   conversor_SaI((buffer_rx + strlen(buffer_rx) +1),pcalidad);}   //  (buffer_rx +
strlen(buffer_rx)+1) contiene la direccion

                                  // a la siguiente palabra enviada por la terminal

                  else if(strcmp(buffer_rx, "Comandos") == 0){                      //Si la cadena es
comandos, damos los comandos
                    tx_cadena_UART0("Lista de comandos:\n\r", buffer_tx[0], &tx_completa,
&pos_cadena);
                    tx_cadena_UART0("Comandos -> Muestra listado de comandos \n\r", buffer_tx[0],
&tx_completa, &pos_cadena);
                    tx_cadena_UART0("Caracteres -> Cambia la recepcion de comandos a caracteres.
\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
             tx_cadena_UART0("Escala <Valor maximo de la escala> -> Personaliza Tu Escala !!! \n\r",
buffer_tx[0], &tx_completa, &pos_cadena);
                    tx_cadena_UART0("Prescision <Valor Precision> -> Personaliza Tu Precision (1 -
1000) !!! \n\r", buffer_tx[0], &tx_completa, &pos_cadena);}

                  else{
                   tx_cadena_UART0("Comando desconocido \n\r", buffer_tx[0], &tx_completa,
&pos_cadena); }
                 }
        }
```

```
        if((*pcomunUART0)&&(*penviofrec==1)){         //Si ha pasado el tiempo necesario para
volver transmitir y hay que hacerlo
             (*pcomunUART0)=0;
                conversor_IaS(frecuencia, pcadconv);
                      //Convierte la frecuencia a Cadena
                tx_cadena_UART0("Frecuencia: ", buffer_tx[0], &tx_completa, &pos_cadena);
//La envia (la frecuencia)
                tx_cadena_UART0(pcadconv, buffer_tx[0], &tx_completa, &pos_cadena);         //La
envia (la frecuencia)
                tx_cadena_UART0("\n\r", buffer_tx[0], &tx_completa, &pos_cadena);
       //Retorno de carro
                }
              }
```

main.c

```
#include <LPC17xx.H>
#include <string.h>
#include "definiciones.h"

                         /////   VARIABLES  \\\\


//Contadores con sus punteros
int contRIT = 0;                            //Contador del RIT (Cada vez que RIT = RITCOMPVAL)
int *pcRIT = &contRIT;               //Puntero al contador del RIT para poder modificarlo desde
funciones
int contSIG = 0;                            //Contador de la señal (Cada vez que hay un flanco de
bajada de la señal)
int *pcSIG = &contSIG;               //Puntero al contador de señal para poder modificarlo desde
funciones
int contLED;                             //Contador del LED para conmutarlo
int *pcLED = &contLED;                                  //Puntero al contador del LED para poder
modificarlo desde funciones
int contUART0;                              //Contador del UART0 para enviar la informacion
int *pcUART0 = &contUART0;                              //Puntero al contador del UART0 para
poder modificarlo desde funciones

//Variables generales
int leds = (0x3FF<<20);                          //Pines a configurar como salidas para los leds
int segment7 = (0xFF<<20);                     //Pines para configurar como salidas para el 7
segmentos
char modo = 1;                                        //Modo de enseñar frecuencia
float tiempoRIT = 0.00001;                  //Tiempo en segundos que tarda para que RIT = RITCOMPVAL
unsigned int baudrate = 19200;          //Velocidad de transmision asincrona en bits/seg
int escala = 1000;                             //Escala en la que estamos (inicializada 10 - 100)
int *pescala = &escala;                        //Puntero a la escala para usarla con otras funcines
float frecuencia;                    //Frecuencia de la señal
char conmuta = 0;                           // Flag que indica si hay que conmutar el led
char comunUART0 = 0;                        // Flag que indica si se envia la informacion por el
UART0
char enviofrec = 0;                               //Flag que indica si hay que enviar la frecuencia
int calidad = 10;                                  //Indica el numero minimo de periodos de la
señal para obtener su frec
char tipo_rx = 0;                             //Flag que indica si se espera recibir un caracter solo
o una cadena


//Variables para UART0
char buffer_rx[40];                     // Buffer de recepción de 40 caracteres
char buffer_tx[m1][m2];                   // Buffer de envio de m1*m2 caracteres
int pos_cadena = 0;                    //Posicion de la ultima cadena enviada + 1, Posicion donde de
escribira la siguiente
char *pcad_rx = buffer_rx;                                 // Puntero de recepción
char rx_completa = 0;                   // Flag de recepción de cadena que se activa al recivir un
caracter
char *pcad_tx = buffer_tx[0];                 // Puntero de transmisión
char tx_completa = 1;           // Flag de transmisión que es 1 cuando se puede transmitir

int main(void){

  config(leds,tiempoRIT,baudrate);                                                  //Configura todo

  while (1){

  estadoflags(pcUART0, &comunUART0 , pcLED, &conmuta);                          //Comprueba si hay
que activar algun flag

  obtfrec(tiempoRIT, &frecuencia, pcRIT, pcSIG, calidad);                              //Obtiene
frecuencia
```

```
  switch(modo){
         //Veamos en que modo queremos mostrar los leds
    case 1:
        //Muestra los leds, en cualquier situacion
    visionleds(escala, frecuencia, &conmuta);
    break;
       case 2:
        //Muestra el display, en cualquier situacion
    vision7seg(escala, frecuencia, &conmuta);
    break;
};

  controlUART0(&rx_completa, pescala, &comunUART0, &enviofrec, &modo, &calidad, &tipo_rx);
//Ejecuta las operaciones del UART0                                          //Se ocupa de la
comunicacion asincrona
}
}
```

definiciones.h

```
#ifndef _definiciones_h_
#define _definiciones_h_
                          ////////       UART0          \\\\\\\


#define UART_ACCEPTED_BAUDRATE_ERROR    3                   // Accepted Error baud rate value (in
percent unit)

#define CHAR_8_BIT                      (3 << 0)
#define STOP_1_BIT                      (0 << 2)
#define PARITY_NONE                     (0 << 3)
#define DLAB_ENABLE                     (1 << 7)
#define FIFO_ENABLE                     (1 << 0)
#define RBR_IRQ_ENABLE                  (1 << 0)
#define THRE_IRQ_ENABLE                 (1 << 1)
#define UART_LSR_THRE                                (1 << 5)
#define RDA_INTERRUPT                   (2 << 1)
#define CTI_INTERRUPT                   (6 << 1)

                  ////////       RIT         \\\\\\

#define RIT_RITINT   (1<<0)                                 // Bit RITINT del registro RICTRL    (Bit
que borra la fuente de la interrupcion con un 1 )
#define RIT_RITENCLR (1<<1)                                 // Bit RITENCLR del registro RICTRL
(Hace que RIT = 0 cuando RIT = RITCOMPVAL)  )
#define RIT_RITEN    (1<<3)                                 // Bit RITEN del registro RICTRL
(Bit de habilitacion, 1->habilitado)
#define PCONP_RIT_ON (1<<16)                          // Bit de habilitación alimentación RIT
(Bit de alimentacion)

               ///VARIABLES\\\\

#define m1 20
#define m2 50


extern char conmuta;                    //Flag para conmutar el led
extern char comunUART0;                        //Flag para saber cuando enviar la comunicacion
asincrona

extern float frecuencia;      //Frecuencia de la señal
extern int contRIT;                   //Contador de RIT (Cada vez que RIT = RITCOMPVAL)
extern int *pcRIT;               //Puntero al contador del RIT para poder modificarlo desde
funciones

extern int contSIG;                     //Contador de la señal (Cada vez que hay un flanco de bajada
de la señal)
extern int *pcSIG;            //Puntero al contador de señal para poder modificarlo desde funciones

extern int contLED;                           //Contador del LED (Cada vez que RIT = RITCOMPVAL)
extern int *pcLED;                            //Puntero al contador del LED para poder modificarlo
desde funciones

extern int contUART0;                 //Contador del UART0 (Cada vez que RIT = RITCOMPVAL)
extern int *pcUART0;                  //Puntero al contador del UART0 para poder modificarlo desde
funciones

extern int leds;                             //Pines para configurar como salidas en el modo 10 leds
extern int segment7;            //Pines para configurar como salidas para el 7 segmentos
extern char modo;                      //Sirve para diferenciar el modo de la visualizacion
extern int calidad;                    //Contiene la precision de las medidas
```

```
extern float tiempoRIT;                     //Intervalo que dice cada cuanto tiempo sera  RIT =
RITCOMPVAL
extern unsigned int baudrate;//Velocidad de transmision asincrona
extern int escala;                          //Escala de la visualizacion
extern int *pescala;                //Puntero a la escala de la visualizacion para poder
modificarlo desde funciones

extern char buffer_rx[40];              // Buffer de recepción de 40 caracteres
extern char buffer_tx[m1][m2];      // Buffer de salida de m * n  caracteres

extern int pos_cadena;        //Posicion de la ultima cadena enviada + 1, Posicion donde de
escribira la siguiente
extern char *pcad_rx;               // Puntero de recepción
extern char rx_completa;            // Flag de recepción de cadena que se activa al recibir la
tecla return CR(ASCII=13)
extern char *pcad_tx;               // Puntero de transmisión
extern char tx_completa;            // Flag de transmisión de cadena que se activa al transmitir el
caracter null
extern char enviofrec;               //Flag que indica si hay que enviar la frecuencia
extern char tipo_rx ;                   //Flag que indica si se espera recibir un caracter solo o
una cadena

//FUNCIONES
//Principales
void config(unsigned int leds,  float tiempo_s, int baudrate);
//Establece la configuracion
void obtfrec(float tiempo_s, float *pfrec, int *pcRIT,int *pcSIG, int calidad);    //Obtiene la
frecuencia
void visionleds(int escala, float fecuencia, char *pcon);
//Enciende los leds en version 10 leds
void vision7seg(int escala, float fecuencia, char *pconmuta);
//Enciende los leds en version display 7 segmentos
void controlUART0(char *prx_completa, int *pescala, char *pcomunUART0, char *penviofrec,  char
*pmodo, int *pcalidad, char *ptipo_rx);       //Se encarga de gestionar los comandos recibidos
void estadoflags(int *pcUART0, char *pcomunUART, int *pcLED, char *pconmuta);     //Controla el
estado de las flags

//Configuraciones
void configRIT( float tiempo_s ) ;                                  //Configura el RIT
void configGPIO(unsigned int leds);                                 //Configura los pines exepto
los del UART0
void configUART0(int baudrate);                                         //Configura el UART0
int uart0_set_baudrate(unsigned int baudrate);              //Establece el baudrate dado

//Herramientas

void conversor_IaS(int numero, char * cadnum);                          //Convierte entero a
cadena y la guarda en cadnum
int conv7s(int numero);                                                         //Le das el
numero que quieres que se vea y devuelve los pines
int conv10leds(int numero);                                                 //Le das los leds
que quires que se enciendan y devuelve los pines
void tx_cadena_UART0(char *cadena, char *inicio_buffer, char *ptx_completa, int *ppos_cadena);
void conversor_SaI(char * cadnum, int *numero);         //Convierte cadena a int, guarda el int en
*numero
       //Envia por UART0 la cadena dada por "cadena"

#endif
```