```c
/* uart.c
 * contiene las funciones:

 1  UART0_IRQHandler(void)
 2  tx_cadena_UART0(char *ptr)
 3  uart0_set_baudrate(unsigned int baudrate)
 4  uart0_init(int baudrate)

 */

#include "uart.h"

char *ptr_tx;     // puntero de transmisi□n
char tx_completa;   // Flag de transmisi□n de cadena completa
char buffer[30];      // Buffer de recepci□n

/*
 * UART0 interrupt handler
 */
 void tx_cadena_UART0(char *cadena)
{
   ptr_tx=cadena;
   tx_completa=0;
   LPC_UART0->THR=*ptr_tx++;   // IMPORTANTE: Introducir un car□cter al comienzo para iniciar TX o
}                  // activar flag interrupci□n por registro transmisor vacio


void analyze_msg(void)
{
  /*
    analyze_msg :: void -> void

    Analyze the message and answer via
    UART, if the msg is among the valid
    ones, we update the status of the
    internal variables.
  */
  static char first_time = 1;                           // Flag that indicates that it is the first time it is executed.
  int aux = 0;                                          // Auxilary variable.

  if (first_time)                                       // If is the first message received, we respond  with
  {                                                     // the instructions available and stop measuring and moving:
    tx_cadena_UART0("You're in automatic mode.\n"
    "- To set resolution in degrees enter xxg"
    "(where xx are possible resolutions in degrees:"
    "05, 10, 15, 20).\n"
    "-To set the period of each servomotor "
    "movement enter xs (where x are possible"
    "periods in seconds: 1 \n "
    "(for 0.5s), 2 (for 1s), 3 (for 2s).\n"
    "-To show this help message again press h\n"
    "-To stop/start sweep mode press m ");
    first_time = 0;
    sonar.f_block_move = 1;
    sonar.f_block_measure = 1;
    sonar.f_block_transmision = 1;

  }
  else if(buffer[0] == 'h')                             // If the the message is help, we respond with the instructions available
    tx_cadena_UART0("You're in automatic mode.\n"
    "- To set resolution in degrees enter xxg"
    "(where xx are possible resolutions in degrees:"
    "05, 10, 15, 20).\n"
    "-To set the period of each servomotor "
    "movement enter xs (where x are possible"
    "periods in seconds: 1 \n "
    "(for 0.5s), 2 (for 1s), 3 (for 2s).\n"
    "-To show this help message again press h\n"
```

```
69        "-To stop/start sweep mode press m ");
70
71      else if(buffer[0] == 'm')                               // If the the message is move,
        we toggle the flags of the sweep mode.
72      {
73        sonar.f_block_move ^= 1;
74        sonar.f_block_measure ^= 1;
75        sonar.f_block_transmision ^= 1;
76      }
77
78      else if(buffer[1] == 's')                               // If the message is to change
        the servo period
79      {
80        if (buffer[0] > '0' && buffer[0] <= '3')              // We look if is a valid period.
81          sonar.servo_period = buffer[0] - 48;                // Calculate the new period.
82        else                                                  // If isn't a valid period we
          indicate it.
83          tx_cadena_UART0("Unexpected message");
84      }
85
86      else if(buffer[2] == 'g')                               // If the message is to change
        the servo resolution
87      {
88        buffer[2] = 0;                                        // Insert a null for a correct
          cast
89        aux = atoi(buffer);                                   // Do the cast, if an error
          ocurrs it returns a zero.
90        if(aux== 5 || aux== 10 || aux == 15|| aux == 20)      // We look if is a valid
          resolution.
91          sonar.servo_resolution = aux;                       // Assing the value to the
            servo resolution.
92        else                                                  // If isn't a valid resolution
          we indicate it.
93          tx_cadena_UART0("Unexpected message");
94      }
95      else                                                    // If isn't a valid msg we
        indicate it.
96        tx_cadena_UART0("Unexpected message");
97    }
98
99    void update_uart(void)
100   {
101     /*
102     update_uart :: void -> void
103
104     Sent the state of the sonar
105     via UART.
106     */
107     static char cycle = 0;
108     if (cycle == 15 && !sonar.f_block_transmision)        // If its time to send
        information and it's allow to transmit from the board.
109     {
110       char msg [30] = "Automatic mode \n";                // Variable that will contains
          the string with the state of the sonar.
111       tx_cadena_UART0(msg);                               // Sent the msg
112       while(!tx_completa);                                // Wait for the message to be
          sent
113
114       sprintf(msg, "Servo pose  %d \n",                   // Format the string with the
          servo pose.
115         sonar.servo_pose);
116       tx_cadena_UART0(msg);                               // Sent the msg
117       while(!tx_completa);                                // Wait for the message to be
          sent
118
119
120       sprintf(msg, "Measured distance  %3.2f cm \n ",     // Format the string with the
          measured distance.
121         sonar.distance);
122       tx_cadena_UART0(msg);                               // Sent the msg
123       while(!tx_completa);                                // Wait for the message to be
          sent
124       cycle = 0;
```

```
125     }
126     else
127       cycle++;                                          // Increase the number of cycles
128   }
129
130   void UART0_IRQHandler(void) {
131     /*
132       UART0_IRQHandler :: void -> void
133
134       Handles the interruption that is
135       generated when the a msg is
136       recived or sent.
137     */
138     switch(LPC_UART0->IIR&0x0E) {
139       static int index = 0;
140       case 0x04:                                        // RBR, Receiver Buffer Ready
141         buffer[index] = LPC_UART0->RBR;                 // Stores the data in the
          correspondent index
142         if (buffer[index] == 13)                        // Return --> Complete String,
143         {
144           analyze_msg();                                // Analyze the chain.
145           index = 0;                                    // Reset the index.
146         }
147         else
148           index++;                                      // Increase index.
149       break;
150
151      case 0x02:                                         //  THRE, Transmit Holding
         Register empty.
152       if (*ptr_tx!=0) LPC_UART0->THR=*ptr_tx++;         //  Loads a new value for
         being transmited.
153       else tx_completa=1;
154       break;
155       }
156   }
157
158   // Funci□n para enviar una cadena de texto
159   // El argumento de entrada es la direcci□n de la cadena, o
160   // directamente la cadena de texto entre comillas
161
162   static int uart0_set_baudrate(unsigned int baudrate) {
163       int errorStatus = -1; //< Failure
164
165       // UART clock (FCCO / PCLK_UART0)
166      // unsigned int uClk = SystemCoreClock / 4;
167       unsigned int uClk =SystemCoreClock/4;
168       unsigned int calcBaudrate = 0;
169       unsigned int temp = 0;
170
171       unsigned int mulFracDiv, dividerAddFracDiv;
172       unsigned int divider = 0;
173       unsigned int mulFracDivOptimal = 1;
174       unsigned int dividerAddOptimal = 0;
175       unsigned int dividerOptimal = 0;
176
177       unsigned int relativeError = 0;
178       unsigned int relativeOptimalError = 100000;
179
180       uClk = uClk >> 4; /* div by 16 */
181
182       /*
183        *  The formula is :
184        * BaudRate= uClk * (mulFracDiv/(mulFracDiv+dividerAddFracDiv) / (16 * DLL)
185        *
186        * The value of mulFracDiv and dividerAddFracDiv should comply to the following
          expressions:
187        * 0 < mulFracDiv <= 15, 0 <= dividerAddFracDiv <= 15
188        */
189       for (mulFracDiv = 1; mulFracDiv <= 15; mulFracDiv++) {
190           for (dividerAddFracDiv = 0; dividerAddFracDiv <= 15; dividerAddFracDiv++) {
191               temp = (mulFracDiv * uClk) / (mulFracDiv + dividerAddFracDiv);
192
193               divider = temp / baudrate;
```

```
194                 if ((temp % baudrate) > (baudrate / 2))
195                     divider++;
196 
197                 if (divider > 2 && divider < 65536) {
198                     calcBaudrate = temp / divider;
199 
200                     if (calcBaudrate <= baudrate) {
201                         relativeError = baudrate - calcBaudrate;
202                     } else {
203                         relativeError = calcBaudrate - baudrate;
204                     }
205 
206                     if (relativeError < relativeOptimalError) {
207                         mulFracDivOptimal = mulFracDiv;
208                         dividerAddOptimal = dividerAddFracDiv;
209                         dividerOptimal = divider;
210                         relativeOptimalError = relativeError;
211                         if (relativeError == 0)
212                             break;
213                     }
214                 }
215             }
216 
217             if (relativeError == 0)
218                 break;
219         }
220 
221         if (relativeOptimalError < ((baudrate * UART_ACCEPTED_BAUDRATE_ERROR) / 100)) {
222 
223           LPC_UART0->LCR |= DLAB_ENABLE;  // importante poner a 1
224           LPC_UART0->DLM = (unsigned char) ((dividerOptimal >> 8) & 0xFF);
225           LPC_UART0->DLL = (unsigned char) dividerOptimal;
226           LPC_UART0->LCR &= ~DLAB_ENABLE; // importante poner a 0
227 
228           LPC_UART0->FDR = ((mulFracDivOptimal << 4) & 0xF0) | (dividerAddOptimal & 0x0F);
229 
230           errorStatus = 0; //< Success
231         }
232 
233         return errorStatus;
234     }
235 
236     void uart0_init(int baudrate)
237     {
238       LPC_PINCON->PINSEL0|=(1<<4)|(1<<6);              // Change P0.2 and P0.3 mode to
          TXD0 and RXD0
239       LPC_UART0->LCR &= ~STOP_1_BIT & ~PARITY_NONE;    // Set 8N1 mode (8 bits/dato,
          sin pariad, y 1 bit de stop)
240       LPC_UART0->LCR |= CHAR_8_BIT;
241       uart0_set_baudrate(baudrate);                    // Set the baud rate
242       LPC_UART0->IER = THRE_IRQ_ENABLE|RBR_IRQ_ENABLE; // Enable UART TX and RX
          interrupt (for LPC17xx UART).
243       NVIC_EnableIRQ(UART0_IRQn);                      // Enable the UART interrupt
          (for Cortex-CM3 NVIC).
244       NVIC_SetPriority(UART0_IRQn, 0);                 // Assign priority 0 to the UART.
245     }
246 
247 
```