```
/*
                    _____ ____  _   _____   ____    ______ __
                   / ___// __ \/ | / /   |  / __ \   / __ ) \/ /
                   \__ \/ / / /  |/ / /| | / /_/ /  / __  |\  /
                  ___/ / /_/ / /|  / ___ |/ _, _/  / /_/ / / /
                 /____/\____/_/ |_/_/  |_/_/ |_|  /_____/ /_/

                 _______   _____      _________    __ _   ______
                / ____/ | / /   |    / ____/   |  / /| | / / __ \
               / __/  | |/ / /| |   / /   / /| | / / | |/ / / / /
              / /___  | |/ / ___ |  / /___/ ___ |/ /__| |/ / /_/ /
             /_____/  |___/_/  |_|  \____/_/  |_/_____/___/\____/

                                 ___    _   ______
                                /   |  / | / / __ \
                               / /| | /  |/ / / / /
                              / ___ |/ /|  / /_/ /
                             /_/  |_/_/ |_/_____/

                 _____ _   _________________    ____  ____  _________________
                / /   | | / / ____/ ____/ __ \  / __ \/ __ \/ ____/ ____/ __ \
           __  / / /| | |/ / // // __/ / /_/ / / / / / /_/ / / / / /   / / / /
          / /_/ / ___ |   // // // /___/ _, _/ / /_/ / _, _/ /_/ / /___/ /_/ /
          \____/_/  |_|___/___/_____/_/ |_|  \____/_/ |_/_/ /___/\____/\____/

*/


// Library of the LPC17.xx
#include <LPC17xx.H>

// Own libraries:
#include "modulos/timer05.h"
#include "modulos/keys.h"
#include "modulos/dac.h"
#include "modulos/screen.h"


// Global variables:
struct sonar_status sonar;                          // Struct that contais the state of the sonar.
uint16_t samples[N_SAMPLES];                        // Array that contains the samples of the DAC signal.

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

    Set the priorities of all the
    interruptions that are used in
    the project, except the priority
    of the UART that is configured
    in its own configuration function.

  */

  NVIC_SetPriorityGrouping(3);                      // Only one bit is needed for the subpriority
  NVIC_SetPriority(TIMER3_IRQn,1);                  // UTS        -> (0,1).
  NVIC_SetPriority(TIMER0_IRQn,2);                  // 0.5 Timer  -> (1,0).
  NVIC_SetPriority(EINT0_IRQn, 4);                  // KEY ISP    -> (2,0).
  NVIC_SetPriority(EINT1_IRQn, 6);                  // KEY 1      -> (3,0).
  NVIC_SetPriority(EINT2_IRQn, 7);                  // KEY 2      -> (3,1).
  NVIC_SetPriority(TIMER1_IRQn,8);                  // DAC        -> (4,0).
}

int main(void)
{

  // Initialize the struct:
  sonar.state               = ST_SETUP;             // Sonar starts in Setup mode.
  sonar.distance            = 0;                    // Sonar distancie is initialize with a zero.
```

```
70      sonar.servo_pose          = 0;                    // The servo starts at zero
        degrees.
71      sonar.servo_period        = 1;                    // The servo period is
        initialize with a period of a 0.5 seconds.
72      sonar.servo_resolution    = 10;                   // The servo resolution is
        initialize with a resolution of 10 degrees.
73      sonar.f_block_keys        = 0;                    // The flag f_block_keys is
        initialize with a zero.
74      sonar.f_block_move        = 0;                    // The flag f_block_move is
        initialize with a zero because at beggining
75                                                        // of the automatic mode the
                                                          servo can move.
76      sonar.f_block_measure     = 1;                    // The flag f_block_measure is
        initialize with one because at beggining
77                                                        // of the manual mode the UTS
                                                          can not move.
78      sonar.f_block_transmision = 0;                    // The flag f_block_measure is
        initialize with zero because at beggining
79                                                        // the transmision from the
                                                          board via uart is enable in
                                                          automatic mode.
80
81      // Configure the hardware:
82      config_timer05();
83      config_keys();
84      config_servo();
85      config_UTS();
86      config_DAC();
87      config_timer_dac();
88      config_priorities();
89      LCD_Init();
90
91      // Initialize the output
92      // and the DAC Signal:
93      generate_samples();                               // Generate the samples of the
        sinusoidal signal of the DAC
94      LCD_Clear(Blue);                                  // Fill the screen with blue
95      set_servo(0);                                     // Initialize the servo pose
96
97      while(1)                                          // Main loop:
98      {
99        sonar.f_block_keys = 0;                         // Clear the flag that blocks
          keys funcionalities.
100       update_screen(&sonar);                          // Update the screen with the
          new status of the sonar.
101       if(sonar.state == ST_AUTOMATIC)                 // If we are in automatic mode
102         update_uart();                                // We update the info via UART
103     }
104
105   }
106
```

```
1   // Preprocessor Directives to include the library only once
2   #ifndef _SONARSTATUS
3   #define _SONARSTATUS
4
5
6   // New defines:
7   #define ST_SETUP       0                                  // Number associated with the
    setup mode.
8   #define ST_MANUAL      1                                  // Number associated with the
    manual mode.
9   #define ST_AUTOMATIC   2                                  // Number associated with the
    automatic mode.
10  #define POSITIVO       0                                  // Number associated with the
    positive direction of the servo.
11  #define NEGATIVO       1                                  // Number associated with the
    negative direction of the servo.
12  #define N_SAMPLES     32                                  // Total number of samples of
    the sinusoidal signal.
13  #define Fpclk 25e6                                        // Frecuency of the
    peripherals by default.
14
15  // New struct:
16  struct sonar_status
17  {
18    /*
19      Structure that contains all
20      the information of the Sonar
21      and allows us to handle it
22      in a simple way.
23    */
24    char  state;                                            // Contains the mode of the
      servo, modes allowed: Setup, Manual and Automatic.
25    float distance;                                         // The distance measure by the
      UTS in cm.
26    int   servo_pose;                                       // The position of the servo
      in degrees.
27    int   servo_period;                                     // How many 0.5 cycles are
      equal to the period of the servo, this parameter can be configured via UART.
28    int   servo_resolution;                                 // Servo motion
      resolution,this parameter can be configured via UART.
29    char  f_block_keys;                                     // Flag to prevent the Eints
      handlers being executed multiple times when the button is pressed.
30    char  f_block_move;                                     // Flag that enables the
      movement of the servo in automatic mode, it can be modified via ISP.
31    char  f_block_measure;                                  // Flag that allows the
      distance measure in manual mode, it can be modified via ISP.
32    char  f_block_transmision;                              // Flag that allows you to
      send information about measurements via UART.
33  };
34
35  #endif
36
```

```
1   // Preprocessor Directives to include the library only once
2   #ifndef _SERVO
3   #define _SERVO
4
5   // Necessary libraries:
6   #include <LPC17xx.H>
7   #include "state.h"
8
9   // New defines:
10  #define Tpwm 15e-3                                      // Period of the PWM signal of
    the servo(15ms)
11
12  // Avaible functions that can be called:
13  void config_servo(void);                                // Enables the PWM, the output
    pin is P1.20.
14  void set_servo(float grados);                           // Moves the servo to the
    position passed in the argument.
15
16  #endif
17
18
```

```
1   // Link this source code with his .h file.
2   #include "servo.h"
3
4   void config_servo(void)
5   {
6     /*
7       config_servo :: void -> void
8
9       Enables the PWM, the output
10      pin is P1.20.
11
12    */
13
14    LPC_PINCON->PINSEL3|=(2<<8);                    // Configure the pin P1.20
      function as a PWM.
15    LPC_SC->PCONP|=(1<<6);                          // Configure power supply.
16    LPC_PWM1->MR0=Fpclk*Tpwm -1;                    // The MR0 is set to the
      equivalent number of TC ticks of PWM's period.
17    LPC_PWM1->PCR|=(1<<10);                         // The PWM is single mode and
      output is enabled.
18    LPC_PWM1->MCR|=(1<<1);                          // When the time counter
      reachs MR0 the time counter is reset.
19    LPC_PWM1->TCR|=(1<<0)|(1<<3);                   // Reset the time counter and
      start to count.
20    LPC_PWM1->LER|= (1<<0);                         // Enables the last changes to
      the MR0.
21  }
22
23  void set_servo(float degrees)
24  {
25    /*
26      set_servo :: float -> void
27
28      Moves the servo to the position
29      passed in the argument.
30    */
31    if(degrees >= 0 && degrees <= 180)              // If the angle doesn´t exceed
      the bounds.
32    {
33      LPC_PWM1->MR2 = (Fpclk * 0.4e-3                 // The MR2 is set to the
        equivalent number of TC ticks of the period
34      +
35      Fpclk *2e-3* degrees/180);                      // that makes the servo move
        to the position passed in the argument
36
37      LPC_PWM1->LER|=(1<<2);                          // Enables the last changes to
        the MR2
38    }
39  }
40
```

```h
// Preprocessor Directives to include the library only once
#ifndef _UTS
#define _UTS


// Necessary libraries:
#include <LPC17xx.H>
#include "state.h"

// New defines
#define TH_UTS 10e-6                                    // Period of the trigger signal.
#define THRESHOLD 100                                   // Measurement threshold.

// Necessary global variable:
extern struct sonar_status sonar;                       // Sonar state is modified by
the UTS measure.

// Avaible functions that can be called:
void config_UTS(void);                                  // Configure the Timer3 to
control the UTS.
void UTS_trigger(void);                                 // Start with the measurement
sequence by triggering the UTS.

#endif


```

```
1   // Link this source code with his .h file.
2   #include "UTS.H"
3
4   void config_UTS(void)
5   {
6     /*
7       config_UTS :: void -> void
8
9       Configure the Timer3 as a Match to
10      generate the trigger signal of the
11      UTS and as a Capture to read the
12      Echo signal of the UTS.
13
14      The pins used are:
15        - P0.10 -> Trigger.
16        - P0.23 -> Echo.
17    */
18
19    // Basic configuration:
20    LPC_SC->PCONP |=(1<<23);                          // Configure the power supply.
21    LPC_PINCON->PINSEL1|=(3<<14);                     // Configure the pin P0.23
      function as a Capture.
22    LPC_PINCON->PINSEL0|=(3<<20);                     // Configure the pin P0.10
      function as a Match.
23    LPC_TIM3-> PR = 0;                                // No prescale -> 25MHz.
24
25    // Match configuration:
26    LPC_TIM3->MR0 = Fpclk * TH_UTS -1;                // Match at 10us -> on/off.
27    LPC_TIM3->EMR|=(1<<0)|(3<<4);                     // When the time counter
      reachs MR0 the P0.10 toggles.
28    LPC_TIM3->MCR |=3;                                // When the time counter
      reachs MR0 interrupts and reset the Timer Counter.
29
30    // Capture configuration:
31    LPC_TIM3->CCR=(1<<2)|(1<<0);                      // When the capture detects a
      rising edge it interrumpts.
32
33    NVIC_EnableIRQ(TIMER3_IRQn);                      // Enables the
      interruption.
34  }
35
36  void UTS_trigger(void)
37  {
38    /*
39      UTS_trigger :: void -> void
40
41      The timer 3 start counting to
42      start with the measurement sequence.
43    */
44    LPC_TIM3->MCR |=3;                                // When the time counter
      reachs MR0 interrupts and reset the Timer Counter.
45    LPC_TIM3->TCR &=~(1<<1);                          // Clear the reset bit.
46    LPC_TIM3-> TCR|=(1<<0);                           // The TC starts counting.
47  }
48
49  void TIMER3_IRQHandler(void)
50  {
51    /*
52      TIMER3_IRQHandler :: void -> void
53
54      Handles the interruption that is
55      generated when the timer 3 reaches
56      the MR0 or the event capture occurs.
57
58    */
59
60    static float start = 0;                           // Variable used to calculate
      the width of the echo pulse.
61
62
63    if(((LPC_TIM3->IR>>0)&1))                         // If the interruption is
      caused by the Match (First part of the trigger signal):
64    {
```

```
65        LPC_TIM3->IR = 1<<0;                                  // Clear the flag of the match
          interrupt
66        LPC_TIM3-> MCR &= ~(3<<0);                            // When the TC reachs the MR0
          it doesn´t interrupt and does not reset.
67      }
68
69      else if((LPC_TIM3->CCR >> 0) & 1)                       // If the interruption is
        caused by a rising edge in the capture (start of the echo signal).
70      {
71        LPC_TIM3->IR=1<<4;                                    // Clear the flag of the
          capture interrupt
72        start = LPC_TIM3->CR0;                                // Save the value of the CR in
          the auxiliary variable.
73        LPC_TIM3->CCR=(1<<1)|(1<<2);                          // Next time the Capture
          interrupts if occurs a falling edge.
74      }
75
76      else                                                    // If the interruption is
        caused by a falling edge in the capture (end of the echo signal).
77      {
78        LPC_TIM3->IR=1<<4;
79        sonar.distance = ((LPC_TIM3->CR0-start)               // Distance calculation in cm.
80            * (1/Fpclk)*0.5*340*100);
81        LPC_TIM3->TCR &=~(1<<0);                              // Stop the timer.
82        LPC_TIM3->TCR |=(1<<1);                               // Reset the timer.
83        LPC_TIM3->CCR  =(1<<2)|(1<<0);                        // Next time the Capture
          interrupts if occurs a rising edge.
84        start = 0;                                            // Reset the auxiliary variable.
85
86
87        if(sonar.distance <= THRESHOLD)                       // If the distance it´s below
          the threshold we change the frecuency of the DAC.
88        {
89          LPC_TIM1->MR0  = (Fpclk                             // New frecuency calculation.
90                / (5000 - sonar.distance * 10)
91                / N_SAMPLES -1);
92          LPC_TIM1->TCR|=(1<<1);
93          LPC_TIM1->TCR &=~(1<<1);                            // Clear the reset bit of the
            timer in charge of the DAC.
94          LPC_TIM1->TCR|=(1<<0);                              // The TC of the timer in
            charge of the DAC starts counting.
95        }
96
97        else                                                  // If the distance it´s above
          the threshold we stop the speaker.
98        {
99          LPC_TIM1->TCR &=~(1<<0);                            // Stop the timer in charge of
            the DAC.
100         LPC_TIM1->TCR|=(1<<1);                              // Reset the timer in charge
            of the DAC.
101       }
102
103     }
104   }
105
106
107
```

```
// Preprocessor Directives to include the library only once
#ifndef _DAC
#define _DAC

// Necessary libraries:
#include <LPC17xx.H>
#include <math.h>
#include "state.h"

// Necessary global variable
extern uint16_t samples[N_SAMPLES];

// New defines:
#define PI 3.14                                          // Pi Number

// Avaible functions that can be called:
void config_DAC(void);                                   // Enables the DAC, the output pin is P0.26.
void config_timer_dac(void);                             // Configure the Timer1 to change the sample of the DAC.
void generate_samples(void);                             // Generate the samples of the sinusoidal signal.

#endif

```

```c
// Link this source code with his .h file.
#include "dac.h"


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

    Enables the DAC, the output
    pin is P0.26.
  */

  LPC_PINCON->PINSEL1  |= (2<<20);                    // Configure the pin P0.26 function as AOUT.
  LPC_PINCON->PINMODE1 |= (2<<20);                    // Pull-up-pull-down not enabled.
  LPC_DAC->DACCTRL      = 0;                          // DMA not enabled
}

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

    Configure the Timer1 to
    change the sample of
    the DAC.
  */

  LPC_SC->PCONP |=(1<<1);                             // Configure the power supply.
  LPC_TIM1->PR   = 0;                                 // No prescale -> 25MHz.
  LPC_TIM1->MCR |= 3;                                 // When the time counter reaches MR0 interrupts and reset the Timer Counter.
  NVIC_EnableIRQ(TIMER1_IRQn);                        // Enables the interruption.
}

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

    Generate the samples of
    the sinusoidal signal.
  */

  int t;
  for(t=0; t < N_SAMPLES; t++)
    samples[t]= (uint16_t)(1023 *                     // Calculate the corresponding sample of the sine
        (0.5 + 0.5 * sin(2*PI*t/N_SAMPLES)));
}

void TIMER1_IRQHandler(void)
{
  /*
    TIMER1_IRQHandler :: void -> void

    Handles the interruption that is
    generated when the Timer1 matchs
    the sample period. This handles
    changes the value of the DAC.
  */
  static char index = 0;
  LPC_TIM1->IR|=(1<<0);                               // Clear the interruption flag of the timer.
  LPC_DAC->DACR=samples[index++]<<6;                  // Change the value of the DAC.

  if(index == N_SAMPLES -1 )                          // If we go through all the samples
    index = 0;                                        // Restart from the begining
}

```

```
// Preprocessor Directives to include the library only once
#ifndef _KEYS
#define _KEYS

// Necessary libraries:
#include <LPC17xx.H>
#include "uart.h"
#include "servo.h"

// Necessary global variable
extern struct sonar_status sonar;                    // Sonar state is modified by
Eint_Handlers

// Avaible functions that can be called:
void config_keys(void);                              // Enables the buttons and
their interruptions

#endif

```

```c
// Link this source code with his .h file.
#include "keys.h"

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

    Enables the buttons
    and their interruptions.
  */

  LPC_PINCON->PINSEL4|=(1<<20)|(1<<22)|(1<<24);       // Configuration of pin functionality.
  NVIC_EnableIRQ(EINT0_IRQn);                         // Enable the interruption of ISP.
  NVIC_EnableIRQ(EINT1_IRQn);                         // Enable the interruption of KEY1.
  NVIC_EnableIRQ(EINT2_IRQn);                         // Enable the interruption of KEY2.
}


void EINT0_IRQHandler()
{
  /*
  EINT0_IRQHandler :: void -> void

  Handles the interruption that is
  generated when the ISP button
  is pressed.

  If the sonar is in automatic
  mode the flag f_block_move toggles,
  When this flag is high the servo
  stops moving.

  Instead, if the sonar is in manual
  mode the flag f_block_measure is
  the flag that toggles. When this
  flag is rised the UTS stops
  measuring.

  If the sonar is in another mode
  this function does not do
  anything.
  */

  LPC_SC->EXTINT |= (1<<0);                           // Clear the interruption flag.
  if(!sonar.f_block_keys)                             // Is the firt time that interrupts this cycle?
  {
    switch(sonar.state)
    {
      case(ST_AUTOMATIC):                             // If we are in aumatic mode:
        sonar.f_block_move ^= 1;                      // Toggle the f_block_move flag.
        break;
      case(ST_MANUAL):                                // If we are in manual mode:
        sonar.f_block_measure ^= 1;                   // Toggle the f_block_measure flag.
        break;
    }
    sonar.f_block_keys = 1;                           // Raise the flag to indicate that we have alredy
  }                                                   // interrupted this cycle.
}

void EINT1_IRQHandler()
{
   /*
    EINT1_IRQHandler :: void -> void

    Handles the interruption that is
```

```
67        generated when the KEY1 button
68        is pressed.
69
70        If the sonar is in Setup mode
71        the mode is changed to automatic,
72        and the UART is configured with
73        a baudrate of 9600 bauds.
74
75        Instead, if the sonar is in manual
76        mode the servo moves 10 degrees
77        in positive direction as long as
78        it not exceeds the maximun angle,
79        in our case 180 degrees.
80
81        If the sonar is in another mode
82        this function does not do
83        anything.
84      */
85
86      LPC_SC->EXTINT |= (1<<1);                       // Clear the interruption flag.
87      switch(sonar.state)
88      {
89        case(ST_SETUP):                               // If we are in Setup mode:
90          sonar.state = ST_AUTOMATIC;                 // Change the mode to
            automatic mode.
91          uart0_init(UART_BAUDRATE);                  // Configure the UART protocol.
92          sonar.f_block_measure  = 0;                 // Initialize the flag for
            automatic mode.
93
94          break;
95
96        case(ST_MANUAL):                              // If we are in Manual mode:
97          if(
98            (!((sonar.servo_pose + 10) > 180))        // If the sonar does not
              exceed the maximun angle
99            &&                                        // in the next move AND is the
              first time that
100           !(sonar.f_block_keys))                    // interrupts this cycle?
101         {
102           set_servo(sonar.servo_pose += 10);        // Increase the servo pose by
              10 degrees.
103           sonar.f_block_keys = 1;                   // Raise the flag to indicate
              that we have alredy
104         }                                           // interrupted this cycle.
105         break;
106     }
107   }
108
109   void EINT2_IRQHandler()
110   {
111     /*
112       EINT2_IRQHandler :: void -> void
113
114       Handles the interruption that is
115       generated when the KEY1 button
116       is pressed.
117
118       If the sonar is in manual
119       mode the servo moves 10 degrees
120       in negative direction as long as
121       it not exceeds the minimun angle,
122       in our case 0 degrees.
123
124       If the sonar is in another mode
125       this function does not do
126       anything.
127     */
128
129     LPC_SC->EXTINT |= (1<<2);                       // Clear the interruption flag.
130     if(
131       (sonar.state == ST_MANUAL)                    // If we are in Manual mode
132       &&                                            // AND
133       (!((sonar.servo_pose - 10) < 0))              // the sonar does not exceed
```

```
        the minimun angle in the next move
134       &&                                                  // AND
135       !(sonar.f_block_keys)                               // is the first time that
          interrupts this cycle
136     )
137     {
138       set_servo(sonar.servo_pose -= 10);                  // Decrease the servo pose by
          10 degrees.
139       sonar.f_block_keys = 1;                             // Raise the flag to indicate
          that we have alredy
140     }                                                     // interrupted this cycle.
141   }
142
```

```h
// Preprocessor Directives to include the library only once:
#ifndef _TIMER05
#define _TIMER05

// Necessary libraries:
#include <LPC17xx.H>
#include "UTS.h"
#include "uart.h"
#include "servo.h"

// Necessary global variable:
extern struct sonar_status sonar;                    // Sonar state is modified by timer 0.5 handler.

// Avaible functions that can be called:
void config_timer05(void);                           // Configure the Timer0 to interrupt every 0.5 seconds.

#endif

```

```
1   // Link this source code with his .h file.
2   #include "timer05.h"
3
4   void config_timer05()
5   {
6     /*
7       config_UTS :: void -> void
8
9       Configure the Timer0 to interrupt
10      every 0.5 seconds.
11    */
12
13    LPC_SC->PCONP |=(1<<1);                                  // Configure the power supply.
14    LPC_TIM0->PR   = 0;                                      // No prescale -> 25MHz.
15    LPC_TIM0->MR0  = (Fpclk*0.5-1);                          // Match at 0.5s.
16    LPC_TIM0->MCR  = 3;                                      // When the time counter
      reachs the match interrupt,stop the TC and reset TC.
17    LPC_TIM0->TCR |= (1<<0);                                 // Start count.
18    NVIC_EnableIRQ(TIMER0_IRQn);                             // Enables the interruption of
      Timer0.
19  }
20
21  void TIMER0_IRQHandler()
22  {
23    /*
24      TIMER0_IRQHandler :: void -> void
25
26      Handles the interruption that is
27      generated when the timer count up
28      to 0.5 seconds.
29
30      If the sonar is in Setup mode
31      the mode is changed to manual.
32
33      If the sonar is in automatic
34      mode the UTS takes a measure,
35      also if the servo is not blocked
36      by the flag f_block_move and
37      it's time to move servo, we
38      move the servo.
39
40      Instead, if the sonar is in the
41      manual mode and the UTS is not
42      blocked by the flag f_block_measure
43      the UTS takes a measure.
44    */
45
46    static char sentido = POSITIVO;                          // Static variable that
      indicates the direction of the move.
47    static int  cycle = 0;                                   // Static variable that
      indicates how much cycles have
48                                                             // passed since the last time
                                                               the servo was turned.
49
50    LPC_TIM0->IR=1<<0;                                       // Clear the flag of the match
      interrupt,
51
52    switch(sonar.state)
53    {
54      case(ST_SETUP):                                        // If the sonar is in Setup mode
55        sonar.state = ST_MANUAL;                             // Change the mode to manual
          mode.
56        break;
57
58      case(ST_AUTOMATIC):                                    // If the sonar is in
        Automatic mode:
59        cycle++;                                             // Increase the number of cycles
60        if(!sonar.f_block_measure)                           // If the UTS is allowed to
          measure:
61          UTS_trigger();                                     // Make a measure with the UTS
62
63        if(!sonar.f_block_move                               // If the servo is allowed to
          move
```

```
64            &&                                    // AND
65           (cycle >= sonar.servo_period))         // the cycle coincides with
             the servo's period.
66         {
67           if(sentido == POSITIVO)                // If the direction is positive.
68           {
69             if((sonar.servo_pose                 // If the next move exceeds
               the bounds.
70                 +
71                 sonar.servo_resolution) > 180)
72               sentido = NEGATIVO;                // Change the direction of the
                 movement.
73
74             else
75               set_servo(sonar.servo_pose         // Increse the angle of the
                 servo.
76                 += sonar.servo_resolution);
77           }
78
79           else                                   // If the direction is negative.
80           {
81             if((sonar.servo_pose                 // If the next move exceeds
               the bounds.
82                 -
83                 sonar.servo_resolution) < 0)
84             {
85               sentido = POSITIVO;                // Change the direction of the
                 movement.
86               set_servo(sonar.servo_pose         // Increase the angle.
87                 += sonar.servo_resolution);
88             }
89             else
90               set_servo(sonar.servo_pose         // Decrease the angle of the
                 servo.
91                 -= sonar.servo_resolution);
92           }
93           cycle = 0;                             // Reset the cycle counter.
94         }
95         break;
96
97       case(ST_MANUAL):
98         if(!sonar.f_block_measure)               // If the UTS is allowed to
           measure:
99           UTS_trigger();                         // Make a measure with the UTS.
100        break;
101    }
102  }
103
```

```
/*
 * uart_.h
 *
 *  Created on: 1-Oct-2011
 *  Author: J.M.V.C.
 *  Modified: 12_Dec-2020
 *  Authotr E.C.R and J.O.P-J
 */

///////////////////////////////////
////////////// Original ///////////
///////////////////////////////////
// Preprocessor Directives to include the library only once:
#ifndef UART_H_
#define UART_H_

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

#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)

extern void uart0_init(int baudrate);
extern void tx_cadena_UART0(char *ptr);

///////////////////////////////////
// Modified for the proyect:  ///
///////////////////////////////////

// Necessary libraries:
#include <LPC17xx.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "state.h"


// New defines
#define UART_BAUDRATE  9600                             // Selected baudrate

// Necessary global variable:
extern struct sonar_status sonar;                       // Sonar state is modified by
timer 0.5 handler.

// Avaible functions that can be called:
void update_uart(void);                                 // Sent the state of the sonar
via UART.

#endif /* UART_H_ */

```

```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 
```

```c
// Link this source code with his .h file.
#include "screen.h"

void update_screen(struct sonar_status *sonar)
{
  /*
    update_screen :: *sonar -> void

    Update the screen with the information
    of the sonar state.

  */
  char msg [50];                                    // Variable that will contains
  the string with the information of the sonar.

  switch(sonar->state)
  {
    case(ST_SETUP):
      sprintf(msg, "Sonar mode = Setup");
      GUI_Text(20,70,(uint8_t *)msg,White,Black);     // Puts the info in the screen.
      break;
    case(ST_MANUAL):
      sprintf(msg, "Sonar mode = Manual");
      GUI_Text(20,70,(uint8_t *)msg,White,Black);     // Puts the info in the screen.
      break;
    case(ST_AUTOMATIC):
      sprintf(msg, "Sonar mode = Automatic");
      GUI_Text(20,70,(uint8_t *)msg,White,Black);     // Puts the info in the screen.
      break;

  }

  sprintf(msg, "Sonar pose = %d ",                  // Format the string with the
  servo pose.
    sonar->servo_pose);
  GUI_Text(20,100,(uint8_t *)msg,White,Black);        // Puts the info in the screen.

  sprintf(msg, "Measured distance = %3.2f ",        // Format the string with the
  measured distance.
    sonar->distance);
  GUI_Text(20,130,(uint8_t *)msg,White,Black);        // Puts the info in the screen.

}

```

```c
// Preprocessor Directives to include the library only once
#ifndef _SCREEN
#define _SCREEN

// Necessary libraries:
#include <stdio.h>
#include "state.h"
#include "GLCD/GLCD.h"

// Avaible functions that can be called:
void update_screen(struct sonar_status *sonar);       // Update the screen with the information of the sonar state.

#endif
```