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

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

    Configure the Timer3 as a Match to
    generate the trigger signal of the
    UTS and as a Capture to read the
    Echo signal of the UTS.

    The pins used are:
      - P0.10 -> Trigger.
      - P0.23 -> Echo.
  */

  // Basic configuration:
  LPC_SC->PCONP |=(1<<23);                         // Configure the power supply.
  LPC_PINCON->PINSEL1|=(3<<14);                    // Configure the pin P0.23 function as a Capture.
  LPC_PINCON->PINSEL0|=(3<<20);                    // Configure the pin P0.10 function as a Match.
  LPC_TIM3-> PR = 0;                               // No prescale -> 25MHz.

  // Match configuration:
  LPC_TIM3->MR0 = Fpclk * TH_UTS -1;               // Match at 10us -> on/off.
  LPC_TIM3->EMR|=(1<<0)|(3<<4);                    // When the time counter reachs MR0 the P0.10 toggles.
  LPC_TIM3->MCR |=3;                               // When the time counter reachs MR0 interrupts and reset the Timer Counter.

  // Capture configuration:
  LPC_TIM3->CCR=(1<<2)|(1<<0);                     // When the capture detects a rising edge it interrumpts.

  NVIC_EnableIRQ(TIMER3_IRQn);                     // Enables the interruption.
}

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

    The timer 3 start counting to
    start with the measurement sequence.
  */
  LPC_TIM3->MCR |=3;                               // When the time counter reachs MR0 interrupts and reset the Timer Counter.
  LPC_TIM3->TCR &=~(1<<1);                         // Clear the reset bit.
  LPC_TIM3-> TCR|=(1<<0);                          // The TC starts counting.
}

void TIMER3_IRQHandler(void)
{
  /*
    TIMER3_IRQHandler :: void -> void

    Handles the interruption that is
    generated when the timer 3 reaches
    the MR0 or the event capture occurs.

  */

  static float start = 0;                          // Variable used to calculate the width of the echo pulse.


  if(((LPC_TIM3->IR>>0)&1))                        // If the interruption is caused by the Match (First part of the trigger signal):
  {
```

```c
  LPC_TIM3->IR = 1<<0;                               // Clear the flag of the match
  interrupt
  LPC_TIM3-> MCR &= ~(3<<0);                         // When the TC reachs the MR0
  it doesn´t interrupt and does not reset.
 }

 else if((LPC_TIM3->CCR >> 0) & 1)                   // If the interruption is
 caused by a rising edge in the capture (start of the echo signal).
 {
  LPC_TIM3->IR=1<<4;                                 // Clear the flag of the
  capture interrupt
  start = LPC_TIM3->CR0;                             // Save the value of the CR in
  the auxiliary variable.
  LPC_TIM3->CCR=(1<<1)|(1<<2);                       // Next time the Capture
  interrupts if occurs a falling edge.
 }

 else                                                // If the interruption is
 caused by a falling edge in the capture (end of the echo signal).
 {
  LPC_TIM3->IR=1<<4;
  sonar.distance = ((LPC_TIM3->CR0-start)            // Distance calculation in cm.
      * (1/Fpclk)*0.5*340*100);
  LPC_TIM3->TCR &=~(1<<0);                           // Stop the timer.
  LPC_TIM3->TCR |=(1<<1);                            // Reset the timer.
  LPC_TIM3->CCR  =(1<<2)|(1<<0);                     // Next time the Capture
  interrupts if occurs a rising edge.
  start = 0;                                         // Reset the auxiliary variable.


  if(sonar.distance <= THRESHOLD)                    // If the distance it´s below
  the threshold we change the frecuency of the DAC.
  {
   LPC_TIM1->MR0  = (Fpclk                           // New frecuency calculation.
         / (5000 - sonar.distance * 10)
         / N_SAMPLES -1);
   LPC_TIM1->TCR|=(1<<1);
   LPC_TIM1->TCR &=~(1<<1);                          // Clear the reset bit of the
   timer in charge of the DAC.
   LPC_TIM1->TCR|=(1<<0);                            // The TC of the timer in
   charge of the DAC starts counting.
  }

  else                                               // If the distance it´s above
  the threshold we stop the speaker.
  {
   LPC_TIM1->TCR &=~(1<<0);                          // Stop the timer in charge of
   the DAC.
   LPC_TIM1->TCR|=(1<<1);                            // Reset the timer in charge
   of the DAC.
  }

 }
}
```