REACTION GAME

Microprocessor based digital systems

G - 69

Delia Sánchez Gómez, 100383339

ÍNDICE

Peripheral initialization:

```
//PA0 configured as digital input (00) and pull-down(10)
GPIOA->MODER &= ~(1<<0*2);
GPIOA->MODER &= ~(1<<(0*2+1));
GPIOA->PUPDR &= ~(1<<0*2);
GPIOA->PUPDR |= (1<<(0*2+1));
EXTI->RTSR |= (1<<0); //rising edge triggers interruption
EXTI->FTSR &= ~(1<<0); //falling edge disable
SYSCFG->EXTICR[0] = 0;//EXTI0 associated with PA0
EXTI->IMR |= 0X01; //unmask exti0
NVIC->ISER[0] |= (1<<6);

//PB7 configured as digital input
GPIOB->MODER &= ~(1<<7*2);
GPIOB->MODER &= ~(1<<(7*2+1)); //PB7 BUTTON PLAYER 2
GPIOB->PUPDR &= ~(1<<(7*2+1)); //Pull up
GPIOB->PUPDR |= (1<<(7*2));
EXTI->FTSR |= (1<<7); //FALLING edge triggers interruption
EXTI->RTSR &= ~(1<<7);  //RISING edge disable
//SYSCFG->EXTICR[1] = 0x1000; //EXTI7 associated with PB7
EXTI->IMR |= 0X0080;  //UNMASK EXTI7
//NVIC->ISER[0] |= (1<<23);


//PB6 configured as digital input
GPIOB->MODER &= ~(1<<6*2);
GPIOB->MODER &= ~(1<<(6*2+1)); //PB6 BUTTON PLAYER 1
GPIOB->PUPDR &= ~(1<<(6*2+1)); //Pull up
GPIOB->PUPDR |= (1<<(6*2));
EXTI->FTSR |= (1<<6); //FALLING edge triggers interruption
EXTI->RTSR &= ~(1<<6); //RISING edge disable
SYSCFG->EXTICR[1] = 0x1100; //EXTI6 associated with PB6 and PB7
EXTI->IMR |= 0X0040;  //UNMASK EXTI6
NVIC->ISER[0] |= (1<<23);

//LEDS configuration (PA12 AND PD2)
// DIGITAL OUTPUT (01)
GPIOA->MODER |= (1<<12*2);
GPIOA->MODER &= ~(1<<(12*2+1));

GPIOD->MODER |= (1<<2*2);
GPIOD->MODER &= ~(1<<(2*2+1));
```

```
//TIM3CH2 TOC WITH INTERRUPTION
   // Internal clock selection: CR1, CR2, SMRC
   TIM3->CR1 = 0x0000; // ARPE = 0 -> Not PWM, it is TOC
   // CEN = 0; Counter off
   TIM3->CR2 = 0x0000; // Always 0x0000 in this subject
   TIM3->SMCR = 0x0000; // Always 0x0000 in this subject
   // Setting up the counter functionality: PSC, CNT, ARR y CCRx
   TIM3->PSC = 31999; // Preescaler=32000 -> F_counter=32000000/32000
= 1000 steps/second
   TIM3->CNT = 0; // Initial value for CNT
   TIM3->ARR = 0xFFFF; // Recommended value = FFFF
   TIM3->CCR2 = 0;
   // IRQ or no-IRQ selection: DIER
   TIM3->DIER = 0x0004; // IRQ enabled only for channel 2 -> CC2IE = 1
                        // Counter output mode
   TIM3->CCMR1 = 0x0000; // CC2S = 0 (TOC)
                       // OC2M = 011 (external output with toggle in
                          channel 2)
                       // OC2PE = 0 (without preload)
   TIM3->CCER = 0x0000; // CC2P = 0 (always for TOC)
   TIM3->SR = 0; // Clear all flags
   // Enabling IRQ source for TIM4 in NVIC (position 30)
   NVIC->ISER[0] |= (1 << 29);

   //TIM4CH1 TOC WITHOUT INTERRUPTION
   // Internal clock selection: CR1, CR2, SMRC
   TIM4->CR1 = 0x0000; // ARPE = 0 -> Not PWM, it is TOC
                       // CEN = 0; Counter off
   TIM4->CR2 = 0x0000; // Always 0x0000 in this subject
   TIM4->SMCR = 0x0000; // Always 0x0000 in this subject
                        // Setting up the counter functionality: PSC,
                           CNT, ARR y CCRx
   TIM4->PSC = 31999; // Preescaler=32000
                      //F_counter=32000000/32000 = 1000 steps/second
   TIM4->CNT = 0; // Initial value for CNT
   TIM4->ARR = 0xFFFF; // Recommended value = FFFF
   TIM4->CCR1 = 0;
   // IRQ or no-IRQ selection: DIER
   TIM4->DIER = 0x0000; // NO IRQ ENABLED
   // Counter output mode
   TIM4->CCMR1 = 0x0000; // CC1S = 0 (TOC)
                         // OC1M = 000
                         // OC1PE = 0 (without preload)
   TIM4->CCER = 0x0000; // CC1P = 0 (always for TOC)
   TIM4->SR = 0; // Clear all flags
```

```
// PA5 INITIALIZATION AS AF
   GPIOA->MODER |=  (1<<(5*2+1));
   GPIOA->MODER &= ~(1<<(5*2));
   // PA5 (TIM2_CH1) alternate function config
   GPIOA->AFR[0] &= ~(0x0F << (4*5));
   GPIOA->AFR[0] |= 1 << (4*5);

   // TIMER 2 CONFIGURATION
   TIM2->CR1 = 0; // clean
   TIM2->CR1 = 0x0080; // 1000 0000 ARPE bit on (0x0080)
   TIM2->CR2 = 0; // zero
   TIM2->SMCR = 0; // zero
   TIM2->PSC = 31;
   // Preescaler=31:f_timer=32000000/32 - 1 = 1000000 steps/s
   TIM2->CNT = 0; // Counter init
   TIM2->ARR = 2;
   TIM2->CCR1 = 0;
   // ARR = (Fclk / (Fpwm * (PSC + 1))) - 1
   TIM2->DIER = 0x0000;
   TIM2->CCMR1 &= ~(0x0FF << (8*0)); // CCMR1 (channel 1)
   TIM2->CCMR1 |= ((6 << 4)|(1<<3));
   TIM2->CCER &= ~(0x0F << 0);
   TIM2->CCER |= (1 << 0);
   // Enable the counter
   TIM2->CR1 |= 0x0001;
   TIM2->EGR |= 0x0001;
   TIM2->SR = 0;

   // ADC CONFIGURATION
   // POTENTIOMETER CONNECTED TO PA4
   // It has to retrieve the voltage value  (0, 1365, 2730)
   GPIOA->MODER |= (1 << 4*2);  // Analog mode (11)
   GPIOA->MODER |= (1 << (4*2+1));
   // We use channel 1
   ADC1->CR2 &= ~(1 << 0); //ADON=0 ADC powered off
   ADC1->CR1 = 0x00000000; // OVRIE = 0 (overrun IRQ disabled)
                           // RES = 00 (resolution = 12 bits)
                           // SCAN = 0 (scan mode disabled)
                           // EOCIE = 0 (EOC IRQ disabled)

   ADC1->CR2 = 0x00000400; // EOCS = 1 (EOC is activated after
each conversion)
                           // DELS = 000 (delay till data is
read) NO DELAY
                           // CONT = 0 (single conversion)
   ADC1->SQR1 = 0x00000000; // 1 channel in the sequence
   ADC1->SQR5 = 0x00000004; // The selected channel is AIN4
```

Infinite loop:

For this project I have decided to implement functions for both Game 1 and Game 2. In the piece of code below, it is possible to see the switch I used to choose one game or another. Since Milestone 4 has been completed, USART is being used in order to select the game to be played.

```
while (1)
{
  change = false;
  initial();
  input[1] = 0;
    switch(game){
    case 1:

      BSP_LCD_GLASS_Clear();
      BSP_LCD_GLASS_DisplayString((uint8_t *)" GAME1");
      for(int j=0;j<10;j++){
        HAL_UART_Transmit(&huart3, &(g1[j]), 1, 10000);
      }
      game1();
      break;
     case 2:
      ADC1->CR2 |= 0x40000000; // When ADONS = 1, I start
                                  conversion (SWSTART = 1)
      while((ADC1->SR&0x0002)==0); //waits till conversion is
                                    finished (ADONS=1)
      ADC_value = (int)ADC1->DR; //store the value
      mode = game2_mode(ADC_value);
      BSP_LCD_GLASS_Clear();
      BSP_LCD_GLASS_DisplayString((uint8_t *)" GAME2");
      for(int j=0;j<10;j++){
        HAL_UART_Transmit(&huart3, &(g2[j]), 1, 10000);
      }
      espera(2000000);
      game2(mode);
      break;
    }
```

Private variables:

```
unsigned char game;

// game 1
unsigned char pressed = 0;
unsigned char display[6];
bool active1 = false;
bool active2 = false;
bool change = false;

// game 2
unsigned short numero;
unsigned char numero_ant = 0;
unsigned int random_count;
signed int player1_time = 0;
signed int player2_time = 0;
unsigned int game2_winner = 0;
bool end_game2 = false;
unsigned int ADC_value = 0;
unsigned int mode = 0;
bool isNegative = false;
unsigned short before[3] = {1668, 2000, 2500};
unsigned short after[3] = {2500, 2000, 1668};
unsigned short current_note = 0;
unsigned short prev_note = 0;
bool sound = false;
bool end_count = false;

// usart
uint8_t input[2] = " ";
//bool init = true;
uint8_t message[103] = "\n\rWelcome to the game of reflexes, select:
game 1 REACTION TIME (press 1), game 2 COUNTDOWN (press 2)\n\r";
uint8_t g1[10] = "\n\rGAME 1\n\r";
uint8_t g2[10] = "\n\rGAME 2\n\r";
uint8_t winner_p1[29] = "\n\rThe winner is Player 1!!\n\r";
uint8_t winner_p2[29] = "\n\rThe winner is Player 2!!\n\r";
bool listen = false;
```

game

This variable was used to store the selected game. It takes the value “1” for Game 1 and “2” for Game 2.

pressed

This variable will store the player who has won Game 1. Its value changes within the interruption set on the external buttons connected to PB6 and PB7.

display

This variable is used in Game 1 and Game 2, and it is used in order to display text or numbers on the LCD. Depending on the outcome of the match, its value may change to show one result or another.

active1, active2

These boolean variables are intended to check the course of the games.
active1 will only be *true* if the game LED is on and it is valid for the players to press their buttons.
active2 will only be *true* if Game 2 is already running. It is used to check whether the pressed button should store the time when it has been hit.

change

This boolean variable changes to *true* anytime the player wants to switch games. At first, this was managed by user button (PA0), but with Milestone 4 it is possible to return to the initial state just by pressing “X” or “x” on the PC.
In order to exit games when *change* turns out to be *true,* every time there is a *while* loop waiting for an event to happen (mostly a variable to change values) or even when it is necessary to wait some time (with timers or with *espera* function), *change* boolean is checked and, if true, loops will be exited and waiting times will be zero.

numero, numero_ant

These variables are used in order to implement the countdown in Game 2. This functionality will be explained in countdown function.

player1_time, player2_time

These variables will store the time in which each player presses their button when Game 2 is running. Since the value will be the difference between 10 (the starting point of countdown) and the pressing time, it can be negative if the player hits the button before countdown ends or postive if it happens after countdown reaches zero.

game2_winner

This variable will store the number of the winner of Game 2.

end_game2, end_count

The first boolean variable will be checked after the countdown of Game 2 ends. Its change to *true* will be triggered by a timer, 2s after reaching zero, and will quit Game 2 with an “END” message on the LCD.

end_count will be used in order to state that the countdown has ended and it allows the timer to start counting 2s for Game 2 to finish.

ADC_value, mode

These two variables have been declared so as to choose the mode of Game 2 within the potentiometer connected to PA4.

ADC_value will store the voltage depending on the resistance value of the variable resistor. Three ranges of values have been set in order to assign a different time regarding the obtained number.

mode will be the number of steps set for the countdown to change values.

isNegative

This boolean variable will change its value to *true* if the winner of Game 2 pressed the button after the countdown ended. It will be checked in order to play melody *before* or *after*.

before, after, current_note, prev_note

before and *after* are the sequences of notes to be played if there is a winner in Game 2. *before* will be played if the winner has pressed the button before ending the countdown. Otherwise *after* will be played.

current_note and *prev_note* are the variables used to go through the sequences. Once the sequence starts playing, each note should sound for a quarter of second and then timer will change the note.

input

This variable is used to store the value obtained through USART.

message, g1, g2, winner_p1, winner_p2

These are the messages to be displayed on the terminal when using USART.

listen

This boolean variable will allow to set a situation in which it is necessary to receive an input from the player. It will enable or disable the character reception when choosing a game at the initial state.

ISR and additional functions:

initial

This function implements the game election for this project using reception within USART.

The initial message is shown in the terminal and listen boolen enables input from keyboard.

```
void initial(){
    for(int j=0;j<103;j++){
      HAL_UART_Transmit(&huart3, &(message[j]), 1, 10000);
    }
    listen = true;
    while(input[1]!=49&&input[1]!=50&&!change);
    listen = false;
    switch(input[1]){
    case 49:
      game = 1;
      break;
    case 50:
      game = 2;
      break;
    }
    input[1] = 32;
}
```

min

The aim of this function is to get the winner of Game 2, obtaining the player who has hit the button the closest to the end of the countdown.

```
int min(int a, int b){
  int c = abs(a);
  int d = abs(b);
  if(!change&&(a!=b)){
    return (c > d) ? 2 : 1; // returns 1 if abs(player1_time)
                               is smaller than abs(player2_time)
  }else{
    return 3;
  }
}
```

Bin2Ascii

This function is the same as the provided by the teachers of the subject in class.

IRQ Handler (EXTI0)

If the *user button* (PA0) is pressed, *change* variable becomes *true* and it skips all waiting times and loops. *game* variable also changes and infinite loop restarts, entering the other game.

```
void EXTI0_IRQHandler(void){
  if (EXTI->PR!=0){
    change = true;
      if(game == 2){
        game=1;
      }
      else{
        game = 2;
      }
      EXTI->PR = 0x01;
  }
}
```

IRQ Handler (EXTI9_5)

Depending on the game that is running, its behaviour is different.

If the current game is Game 1 and the LED has already been lit, pressed variable will store the value of the player who hits the button first.

```
void EXTI9_5_IRQHandler(void){
  switch(game){
  case 1: // GAME 1
    if((EXTI->PR)==0x0040){ // PB6, BUTTON 1
        if(active1){
          pressed = 1;
          Bin2Ascii((TIM4->CNT),display);
        }
        EXTI->PR = 0x0040;
    }
    if((EXTI->PR)==0x0080){ // PB7, BUTTON 2
      if(active1){
        pressed = 2;
        Bin2Ascii((TIM4->CNT),display);
      }
      EXTI->PR = 0x0080;
    }
    break;
```

If Game 2 is the one running, *player1_time* and *player2_time* will store the moment in which each player hits the button minus the time measured by the countdown.

```
  case 2: // GAME 2
    if((EXTI->PR)==0x0040){ //PB6, BUTTON 1
        if(active2){
          player1_time = (TIM4->CNT)-(10*mode); // stores the time
                                                the button is pressed
        }
        EXTI->PR = 0x0040;
      }
    if((EXTI->PR)==0x0080){ // PB7, BUTTON 2
        if(active2){
          player2_time = (TIM4->CNT)-(10*mode); // stores the time
                                                  the button is pressed
        }
        EXTI->PR = 0x0080;
      }
    break;
  }
}
```

Timer 3 IRQ Handler (TIM3)

Timer 3 is used several times in this project. Since it has been configured as TOC with interruption enabled, depending on the value stored in TIM3->CCR1, IRQ will have different effects.

In Game 1, channel 2 is used and it first manages the random time before the LED is turned on and, after one of the players has pressed a button, counts 3s before restarting the process.

```
void TIM3_IRQHandler(void) {
 if ((TIM3->SR & 0x0004)!=0){ // Is the event channel 2 from TIM3?
     switch(game){
       case 1:
         if((pressed==0)&&(!change)&&(!active1)){
           GPIOD->BSRR = (1<<2);
           TIM4->CNT = 0;
           TIM4->CCR1 = 50*1000;
           TIM4->CR1 |= 0x0001; // CEN = 1 -> Starting CNT
           TIM4->EGR |= 0x0001; //Update registers
           TIM3->CR1 = 0x0000;
           TIM3->CNT = 0;
           TIM3->EGR |= 0x0001;
           active1 = true;
         }else if((pressed!=0)&&(!change)&&(active1)){
           active1 = false;
         }
         break;
```

In Game 2, channel 2 is also being used. Timer 3 is in charge of the countdown, regarding the steps set on the Game 2 mode (potentiometer). It is also responsible of changing notes when playing a sequence, and the one counting up to 2s after the countdown reaches zero for the game to end.

```
      case 2:
        if ((TIM3->SR & 0x0004)!=0){ // If the comparison is
                                      successful, then the IRQ is launched
                                     // and this ISR is executed. This line
                                        check which event
                                     // launched the ISR
          if(end_count&&player1_time==0&&player2_time==0&&!sound){ //if none of
                                     the players has pressed the button after 2s
            end_game2 = true;
          }
          if(numero==0&&!end_count){
            end_count = true;
            TIM3->CR1 = 0x0000; // CEN = 1 -> stop counter
            TIM3->EGR |= 0x0001; // UG = 1 -> Generate an update
                                    event to update all registers
            TIM3->CCR2 = 2000;
            TIM3->CNT = 0;
            TIM3->CR1 |= 0x0001;
            TIM3->EGR |= 0x0001;
          }else{
            if(numero>=0){
              numero--; // Decrease in 1 the number to be shown in the LCD
            }
            TIM3->CCR2 += mode; // Update the comparison value, adding 1000
                                   steps = 1 second
          }
          if((game2_winner!=0)&&sound){
            if(current_note<2){
              current_note++;
            }else{
              sound = false;
              current_note = 0;
              TIM3->CR1 = 0x0000; // CEN = 1 -> Stopping CNT
              TIM3->CNT = 0;
              TIM3->EGR = 0x0001; // UG = 1 -> Update event*/
            }
          }
          if(((game2_winner!=0))&&active2){//wait 3s until the game restarts
            active2 = false;
          }
        break;
     }
   }
 TIM3->SR = 0x0000; // Clear all flags
 }
}
```

espera function

This auxiliar function waits a determined amount of time. If user button or “X” is pressed, the function will be exited and the program will restart.

```
void espera(int tiempo){
  while (tiempo>0&&(!change)){
    tiempo--;
  }
}
```

game1

When Game 1 starts, TIM3_CH2 starts counting a random time (TIM3->CCR2). When the LED is lit, the first player to press the button is stored in *pressed* variable and the result is displayed on the LCD and Terminal

```
void game1(){ //reaction time game implementation function
  TIM3->CNT = 0;
  TIM3->CCR2 = ((rand() % (8 - 3 + 1)) + 3)*1000;
  TIM3->CR1 |= 0x0001; // CEN = 1 -> Starting CNT
  TIM3->EGR |= 0x0001; // UG = 1 -> Update event
  while(pressed == 0&&(!change)); //waiting
  switch (pressed){
    case 1:
      BSP_LCD_GLASS_Clear();
      BSP_LCD_GLASS_DisplayString((uint8_t *)" P1");
      for(int j=0;j<29;j++){
        HAL_UART_Transmit(&huart3, &(winner_p1[j]), 1, 10000);
      }
      espera(2000000);
      BSP_LCD_GLASS_Clear();
      BSP_LCD_GLASS_DisplayString((uint8_t *)display);
      for(int j=0;j<6;j++){
        HAL_UART_Transmit(&huart3, &(display[j]), 1, 10000);
        }
      break;
    case 2:
      BSP_LCD_GLASS_Clear();
      BSP_LCD_GLASS_DisplayString((uint8_t *)" P2");
      for(int j=0;j<29;j++){
        HAL_UART_Transmit(&huart3, &(winner_p2[j]), 1, 10000);
      }
      espera(2000000);
      BSP_LCD_GLASS_Clear();
      BSP_LCD_GLASS_DisplayString((uint8_t *)display);
      for(int j=0;j<6;j++){
        HAL_UART_Transmit(&huart3, &(display[j]), 1, 10000);
        }
    default:
      break;
    }
```

TIM4 has also been configured as TOC. However, it will not trigger any interruption, and will only be used to check the time in which a button has been pressed.

Once the main part of Game 1 has finished, TIM3 starts counting up to 3s for the program to restart. Then, *active1 = false*, the LED will go off and TIM3 and TIM4 will reset.

```
  TIM3->CNT = 0; // 3s for the game to restart
  TIM3->CR1 |= 0x0001;
  TIM3->EGR |= 0x0001;
  TIM3->CCR2 = 3000; //añado 3s
  while(active1&&(!change)); //wait some time to restart the game
in case pa0 not pressed
  pressed = 0; //reset the game
  GPIOD->BSRR = (1 << 2)<<16;
  BSP_LCD_GLASS_Clear(); //clear lcd
  TIM3->CR1 = 0x0000; // CEN = 1 -> Stopping CNT
  TIM3->CNT = 0;
  TIM3->EGR = 0x0001; // UG = 1 -> Update event*/
  TIM3->SR = 0x0000;
  TIM4->CR1 = 0x0000; // CEN = 1 -> Stopping CNT
  TIM4->CNT = 0;
  TIM4->EGR = 0x0001; // UG = 1 -> Update event*/
  TIM4->SR = 0x0000;
}
```

countdown

This function will perform the countdown of Game 2. If *numero* is still greater than zero (countdown has not ended yet) and Game 2 is running (active2), TIM3 will decrease its value regarding the step value set by the potentiometer. *numero* will be displayed on the LCD until it reaches the value of *random_count*.

```
void countdown(){
  while((numero>0)&&(active2)&&(!change)){
    if ((numero_ant != numero)&&(numero>=random_count)) {
    // If the ISR has changed the number to show …
        numero_ant = numero;
    // Store new number to be used next time
        Bin2Ascii(numero, display); // Convert number to text
        BSP_LCD_GLASS_Clear();
        BSP_LCD_GLASS_DisplayString((uint8_t *)display);
    }else if((numero<random_count)){
      BSP_LCD_GLASS_Clear();
    }
  }
}
```

playSequence

This function receives the boolean variable which tells whether the winner has pressed the button before the countdown ends (time will be negative). Before it starts playing any sequence sound is set to true. This will not allow the game to end if the sequence is playing.

When TIM3_CH2 counts 250ms, the note will change to the following one in the sequence and this will happen until the melody has finished.

To generate the sound TIM2 has been used, configured as PWM.

```
void playSequence(bool n){
  sound = true;
  if(n){ //if the winner pressed the button before the countdown ended
    while(sound&&!change&&!end_game2){
      prev_note = current_note;
      TIM2->ARR = before[current_note];
      TIM2->CCR1 = before[current_note]/2;
      TIM3->CNT = 0;
      TIM3->CCR2 = 250;
      TIM3->CR1 |= 0x0001;
      TIM3->EGR |= 0x0001;
      while(current_note==prev_note&&!change); //while note is playing
    }
  }else{
    while(sound&&!change&&!end_game2){
      prev_note = current_note;
      TIM2->ARR = after[current_note];
      TIM2->CCR1 = after[current_note]/2;
      TIM3->CNT = 0;
      TIM3->CCR2 = 250;
      TIM3->CR1 |= 0x0001;
      TIM3->EGR |= 0x0001;
      while(current_note==prev_note&&!change); //while note is playing
  }
}
  TIM2->ARR = 2;
  TIM2->CCR1 = 0;
  sound = false;
}
```

game2_mode

This function returns the steps used in order to decrease *numero* in the countdown.

It has been determined three ranges depending on the voltage on potentiometer connected to PA4.

```
int game2_mode(int value){
  unsigned int output = 0;
  if (value<1365){
    output = 500; //TIM3->CCR2 500ms
  }else if(value>=1365&&value<2730){
    output = 1000; // TIM3->CCR2 1s
  }else if(value>=2730){
    output = 2000; // TIM3->CCR2 2s
  }
  return output;
}
```

game2

This function is the responsible of running Game 2. It makes use of the functions stated before: *countdown*, *playSequence* and *min*.

After players have pressed the buttons, the winner is computed thanks to *min* function, which returns the player who has hit the button the nearest in time from the end of the countdown. The result is displayed on the LCD and Terminal and, depending on whether the winner pressed the button before or after the countdown reached zero, it will sound melody *before* or *after* thanks to the function *playSequence*, and the corresponding LED will be on.

If none of the players has pressed the button 2s after the end of the countdown, “END” message will be shown and the program will restart.

```
void game2(int mode){ // countdown game implementation function
  numero = 10;
  end_game2 = false;
  end_count = false;
  random_count = ((rand() % (8 - 3 + 1)) + 3); //(rand() % (upper -
lower + 1)) + lower;
  TIM3->CNT = 0;
  TIM3->CCR2 = mode; // Record that stores the value for the
comparison.
  TIM3->CR1 |= 0x0001; // Enable the counter
  TIM4->CNT = 0;
  TIM4->CR1 |= 0x0001;
  TIM4->EGR |= 0x0001;
  TIM3->EGR |= 0x0001; //Update registers
  active2 = true;
  countdown(); // countdown starts

while(((player1_time==0)||(player2_time==0))&&(!change)&&(!end_game2));
// wait until both players have pressed their buttons
  game2_winner = min(player1_time, player2_time);
```

```
  switch(game2_winner){
  case 1:
    if(player1_time<0){ // if player pressed the button before the
countdown ended
      isNegative = true;
      player1_time = abs(player1_time);
      Bin2Ascii(player1_time, display);
      for(int i = 0; i<6;i++){
        if(display[i]!=0x20&&display[i]!=0x30){
          break;
        }else if(display[i+1]!=0x30){
          display[i] = 0x2D;  // write '-' before the absolute value
number
          if(i>0&&display[i-1]==0x30){
            display[i-1] = 0x20; // write space before '-'
          }
          break;
        }
      }
    }else{
      Bin2Ascii(player1_time, display);
      isNegative = false;
    }
    GPIOD->BSRR |= (1 << 2);
    BSP_LCD_GLASS_Clear();
    BSP_LCD_GLASS_DisplayString((uint8_t *)" P1");
    for(int j=0;j<29;j++){
      HAL_UART_Transmit(&huart3, &(winner_p1[j]), 1, 10000);
    }
    espera(2000000);
    BSP_LCD_GLASS_Clear();
    BSP_LCD_GLASS_DisplayString((uint8_t *)display); // Display number
in the LCD
    for(int j=0;j<6;j++){
      HAL_UART_Transmit(&huart3, &(display[j]), 1, 10000);
      }
    break;
```

```
case 2:
    if(player2_time<0){ // if player pressed the button before the countdown
ended
      isNegative = true;
      player2_time = abs(player2_time);
      Bin2Ascii(player2_time, display);
      for(int i = 0; i<6;i++){
        if(display[i]!=0x20&&display[i]!=0x30){
          break;
        }else if(display[i+1]!=0x30){
          display[i] = 0x2D;
          if(i>0&&display[i-1]==0x30){
            display[i-1] = 0x20; // write space before '-'
          }
          break;
        }
      }
    }else{
      Bin2Ascii(player2_time, display);
      isNegative = false;
    }
    GPIOA->BSRR |= (1 << 12);
    BSP_LCD_GLASS_Clear();
    BSP_LCD_GLASS_DisplayString((uint8_t *)" P2");
    for(int j=0;j<29;j++){
      HAL_UART_Transmit(&huart3, &(winner_p2[j]), 1, 10000);
    }
    espera(2000000);
    BSP_LCD_GLASS_Clear();
    BSP_LCD_GLASS_DisplayString((uint8_t *)display); // Display number in the LCD
    for(int j=0;j<6;j++){
      HAL_UART_Transmit(&huart3, &(display[j]), 1, 10000);
      }
    break;
  default:
    break;
  }
```

```
  playSequence(isNegative);
  if(end_game2){
    BSP_LCD_GLASS_Clear();
    BSP_LCD_GLASS_DisplayString((uint8_t *)" END");
  }
  TIM3->CR1 = 0;
  TIM3->EGR |= 0x0001;
  TIM3->CNT = 0;
  TIM3->CR1 |= 0x0001;
  TIM3->EGR |= 0x0001;
  TIM3->CCR2 = 3000; // set 3s to restart
  while((active2)&&(!change));
  TIM4->CNT = 0;
  TIM3->CNT = 0;
  TIM4->CR1 = 0x0000;
  TIM3->CR1 = 0x0000;
  TIM4->EGR |= 0x0001;
  TIM3->EGR |= 0x0001; // stop game 2

  game2_winner = 0;
  player1_time = 0;
  player2_time = 0; // reset game 2

  GPIOA->BSRR |= (1 << 12)<<16;
  GPIOD->BSRR |= (1 << 2)<<16;

  BSP_LCD_GLASS_Clear();
}
```

RX IRQ Handler

Stores the received value from keyboard input.

“X” or “x” is processed anytime. Game selection is only processed when the question is on screen.

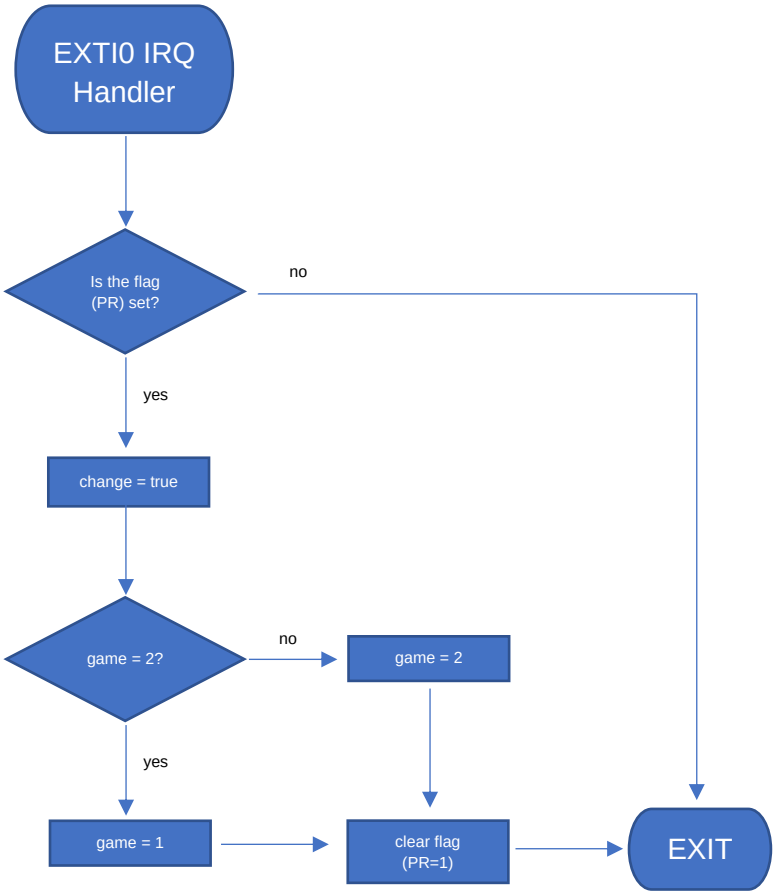
```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
  if(input[1]==88||input[1]==120){
    change = true;
  }
  if((input[1]==49||input[1]==50)){
    if(!listen){
      input[1] = 32;
    }
  }
  HAL_UART_Receive_IT(huart, &(input[1]), 1);
// Vuelve a activar Rx por haber acabado el buffer
}
```

PERIPHERALS SUMMARY:

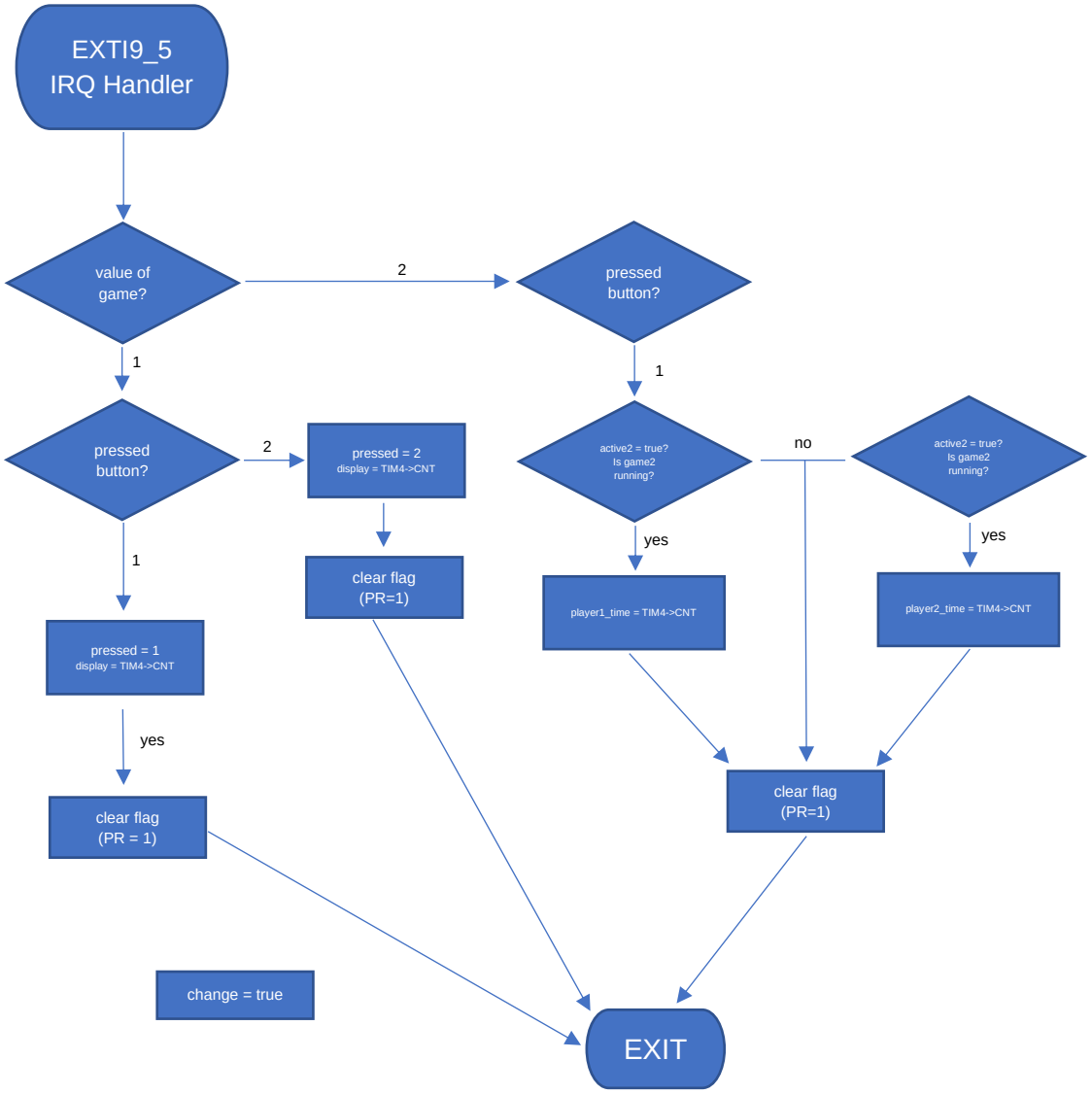
PERIPHERAL	CONFIGURATION
PB6, PB7 (buttons)	GPIO Input (00) Pull-up (01) Falling Edge enabled Rising Edge disabled EXTIs enabled IRQ enabled for each EXTI
PA12, PD2 (LEDs)	GPIO Output (00)
TIM2 (speaker)	Configured as PWM ARR = (Fclk / (Fpwm × (PSC + 1))) – 1
TIM3	Configured as TOC Channel 2 IRQ enabled
TIM4	Configured as TOC Channels 1 and 2 No IRQ enabled
PA4 (potentiometer)	ADC input GPIO as AF Single conversion
USART	RX interruptions enabled

ISR flowcharts:

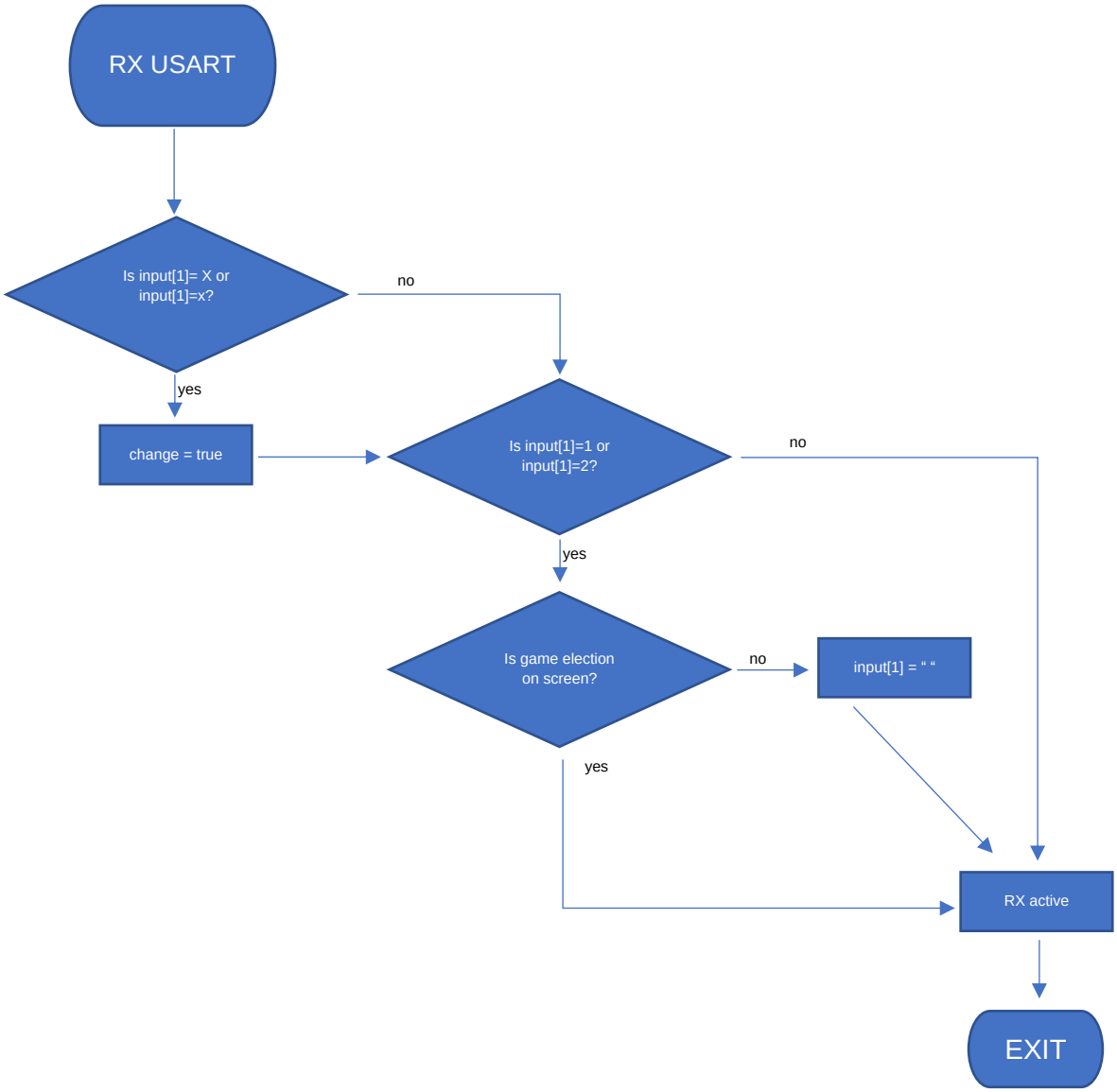
EXTI0 IRQ Handler:



EXTI9_5 IRQ Handler:



HAL_UART_RxCpltCallback:



TIM3 IRQ Handler:

Conclusions:

Since requirements for Milestone 4 have been achieved, the project is completely finished. No improvements have been carried out.
The code is not fully optimised since I did not have time.