EIE3810 Microprocessor System Design Laboratory

Lab6 – Mini Ball Project

By *119010046*

Yabin CHENG

23th December, 2021

Table of Contents

1 Word written in the front for notice

1. You may use ST-LINK to download hex file to board without changing any settings! It makes things easier
2. The code used in this project is a mix of my code in previous 5 experiments and also code provided by *Openedv.com* / 正点原子. In detail, the logic structure is written all by myself, while the hardware related code like hardware initialization and some TFTLCD screen print related functions. In reading my code, you will find I use a few interfaces provided by *Openedv.com* (like LED, buzzer control function, and a delay function that call SYSTICK to provide accurate time delay function). I used code provided by *Openedv.com* for initialization for fear of error hidden in my code, which may cause fault that is hard to catch in this not a small project.

2 Experiment Flow

- Experiment A: Implement the basic hit ball game on the development board
- Experiment B: Extension part

3 Experiment A: Implement the basic hit ball game on the development board

3.1 Design of the game

3.1.1 Finish the hardware initialization and check the baud rate

In the beginning of the game, we have to initialize all the hardware used, including clocks, timers, LCD, Buzzer, buttons, joypad and the usart. The code is shown below for reference.

```
    Stm32_Clock_Init(9);    //系统时钟设置->72MHz
    uart_init(72, 9600);//72Mhz clock and the baud rate 9600
    delay_init(72);         //延时初始化 // enable systick timer
    LED_Init();             //初始化与LED 连接的硬件接口

    InitTimers();// 50Hz 40 Hz 40Hz tim2 tim3 tim4

    JOYPAD_Init();
    EXTIX_Init();//init key and the interrupt correesponded with the key
    BEEP_Init();//comment this line for a freindly testing enviroment,
this enable beeper to work
```

```
    LCD_Init();// init the TFTLCD screen
```

The frequency of the UART is set by 9600Hz, which is obtained by analyzing the waveforms of the UART RX signal using the oscilloscope. The screenshot is shown below in Figure 1's line 0.

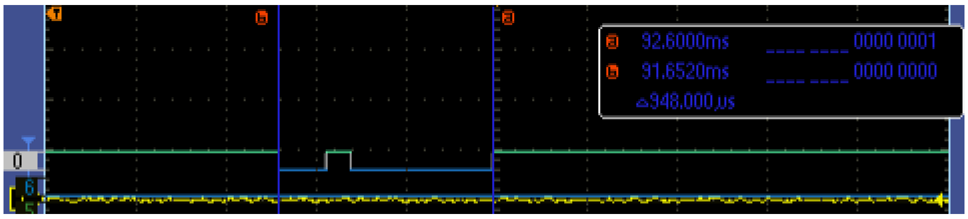


Figure 1. Screenshot of the oscilloscope in capturing the Baud rate of Rx signal as 0 line

The equation is $BaudRate = \frac{1}{period\ of\ 9\ bits} \times 9\ = 9500\ \approx 9600$. The error is due to mismatch of the end measure time which made the period measured a little larger and the baud rate a little smaller than possible choice of baud rates.

3.1.2 Implement the static pages (freeze page)

There are a few static pages in this game, like showing the first page, the first frame of the selection page, the result showing page. These only requires calling the LCD related function, and is not related to the main logic of the program. These functions are listed below.

```
void showWelcome(void);
void showSelect(void);
void showUSARTPage(void); // show the static USART waiting frame
void gameCountDown(void);
void showResult(void);//show result after game finishes
```

3.1.3 Implement active pages (interacting page) except the game

After finish the static page design, we add the control logic in the selection page by adding interrupt handler functions for keys. In the handler, the function changes a global variable to which indicates the last key pressed. Then I implement the live version of the game selection function by actively scanning the last hit key. This function (`LevelSelectFunction`), which runs in the main function, treats the last hit key read from the global variable and update the screen and the flow of the main program. Similarly, there is a `USARTWorking` function that busily waits and reads the global variable to know whether the random number has been passed. Functions mentioned above is listed below.

```
void LevelSelectFunction(void);
void PlayerSelectFunction(void); // used in the bonus part
void USARTWorking(void);
```

3.1.4 Implement the game page function

To finish the basic tasks, we only needed to care the game function to finish this project. In beginning, I wanted to put some work in the game function and some in handlers, then I encounter issue of unexpected. I notice that it could be due to the fact that this function which run in the main function can be interrupted by the handler functions and may somehow bring unexpected bad experience in the game. Then, I decided to put everything about the game in the time handler, and inside the game function only to initialize the game related variables like position of the up/low beater, move unit of the ball (according to the given random number), and the view of the screen. After finish initialization, this program just updates the stage (page) variable by adding one and busily waits for the variable to be changed.
The code for the game function (*gameloop)* is shown below.

```
void gameLoop()
{
    //do the init thing
    speed = Level;
    xMove = (UnitX * (randomNumber%4+1)) * speed;
    yMove = UnitY * speed;
    directionX = (randomNumber/4)? goRightOrDown:goLeftOrUp;
    directionY = goLeftOrUp;
    whichBall = 0; whichColor = 2;
    ballPos[0] = 240; ballPos[1] = 800 - myBallSizes[whichBall] -
beaterShape[1]-1;
    EIE3810_TFTLCD_FillRectangle(0,480, 0, 800, WHITE);
    EIE3810_TFTLCD_FillRectangle(HighBeaterPos[0], beaterShape[0],
HighBeaterPos[1], beaterShape[1],BLACK);
    EIE3810_TFTLCD_FillRectangle(LowBeaterPos[0], beaterShape[0],
LowBeaterPos[1], beaterShape[1], BLACK);

    //let the timer control the whole thing
    gameStatus = 3;//this means we are entering the moving game
    while(gameStatus == 3);
}
```

3.1.5 Implement TIMx Handlers to finish the ball game

Continue from designing the game loop function, the program needs to treat the interrupt handler well to make the program flow. The program related includes TIM2_IRQHandler(void), TIM3_IRQHandler(void), TIM4_IRQHandler(void). The frequency of each handler is 50Hz, 40Hz and 40Hz. The TIM2_IRQHandler

a) The TIM4_IRQHander reads the left right key in the development board and update control info for the lower beater.
b) The TIM3_IRQHander reads the JOYPAD and control the game (pause and restart) and the update the upper bar/beater movement info.

c) **The TIM2_IRQHander** has an inner static time variable which increases after each time it is called. A beepTime variable which indicates the rest lasting time for the beeper to voice. It also moves the ball and update the logic related with the ball's movement, like touchNumber, beeper on time, and other added infos. It handles the up and low beater movement by global variable which is updated by TIM4 and TIM3 handler accordingly. Moreover, it controls the data info display refresh in the screen every .5 seconds.
In summary, this timer is the main body of the game which controls the movement of ball, beaters, on and off of buzzer, touch logic of the ball, time, and display. This handler is the most important among all other codes.

You can check the code in time.c and time.h file to see the implementation detail.

3.2 Some details in implement the game

3.1 sections mainly the main body and logic flow of this game, and in this section, we will talks about some details in implementation, including how was the ball direction controlled, priority management, move function for the beater, and more. Some of them is elaborated with code.

3.2.1 How to translate the random number received to ball direction

In the design, the smaller ball movement unit in x direction is set as a variable XUnit = 3 pixels, and that in y direction is 5. If the random number received is 0 or 4 each move in x direction is one XUnit, 1 or 5 then two XUnits each step, 2 or 6 to three XUnits, and 3 or 7 to four XUnits. The moveUnit for Y direction is fixed as 5 pixels.
Also, the ball's first movement is to left or right is determined by whether the random number is smaller than 4 or not.
The relevant code is shown below inside the *gameLoop*() function.

```
    xMove = (UnitX * (randomNumber%4+1)) * speed;//
    yMove = UnitY * speed;
    directionX = (randomNumber/4)? goRightOrDown:goLeftOrUp;//go left if
random number is smaller than 4 otherwise go right
```

3.2.2 How to manage different interrupts' priority?

I think in my program, there is no interrupt that worth a higher priority than the other, so I set the priority group index to 7 so that no preempt priority is used. Under this setting, every interrupt is free of other's interrupt, and our time logic can be easier managed.

3.2.3 How was the beater movement function designed?

The movement function is called in the TIM2 handler. The move function is called when its corresponded movement variable which indicate direction or direction plus move pixel numbers is not zero. The variables are int variables named as UpperMove and LowerMove.
LowerMove is updated in the TIM4 handler that change it to 1 once it finds right key is pressed, while -1 if left is pressed.
As for the UpperMove is updated in the TIM3 handler in the game state, and is increased or decreased by 10 if corresponded key on joypad is pressed.
The UpperBarMove function when called will move the up beater according to the UpperMove variable, and will move a fix step according to the variable. The lowerBarMove function is written in a similar way. The LowerBarMove function is attached here for reference.

```
void lowerBarMove(int* pos, int* size, int* direction)
{
    int posX = *pos, posY = *(pos+1);int moveUnit = 5;

    if (*direction == goLeftOrUp)
    {
        if (posX >= moveUnit)
        {
            EIE3810_TFTLCD_FillRectangle(posX+beaterShape[0]-moveUnit, 
moveUnit, posY, beaterShape[1], WHITE);
            posX -= moveUnit; *pos = posX;
            EIE3810_TFTLCD_FillRectangle(posX, moveUnit, posY, 5, BLACK);
        }
        *direction = 0;
    }
    else if (*direction == goRightOrDown)
    {
        if (posX <= 480-beaterShape[0]-moveUnit)
        {
            EIE3810_TFTLCD_FillRectangle(posX, moveUnit, posY, beaterShape[1], 
WHITE);
            posX += moveUnit; *pos = posX;
            EIE3810_TFTLCD_FillRectangle(posX+beaterShape[0]-moveUnit, 
moveUnit, posY, beaterShape[1], BLACK);
        }
        *direction = 0;
    }
}
```

3.2.4 How to avoid pollution of the up or down beater's display due to ball touch?

In running the program, I find that sometimes, the ball covers some region of the beater, and when the ball leaves, the white color made by the ball movement could last on the touched beater.The beater can recover in good shape only when the user make it moves.
To fix this situation, I let the beater refresh for a few times after a beater kick happened. If at least one refresh happens after the ball has entirely not touch the beater anymore, the beater is made sure to be in a good figure. In testing, the number is 3. This means the ball can always leave the beater in three steps at least.

3.2.5 How was pause realized (Key & Joypad)?

1. For the Key pause. This is TIM2 handler. If it finds the down key is pressed, then it enters a while loop that detect whether the down key is pressed again. If down key is found to be pressed again, then just leaves the while loop and keep going to the rest functions of the TIM2 handler.
2. As for the Joypad stops function, it is inside the TIM3 handler. When the start key is detected and the program is in the game state, then the handler will enter a subroutine that first waits for the joypad to read a value other than start, which indicates the start button is released, then a while loop that periodically asked for the Joypad value until another START value is read, then out the while loop. Then another while loop to waits for the start button to be released. This part is a little than the KEY part because KEY is triggered by the falling edge while the JOYPAD is triggered by actively detecting. The overall difference is not much. The relevant code used in the joypad pause is attached below.

```
if (gameStatus == 3 && mode == PLAYER)//do the pause thing by busy waiting
{
    BACK_COLOR = WHITE;POINT_COLOR = RED;
    LCD_ShowString(50, 300, 400, 40, 24, "You are pasued");
    BEEP = 0;//stop the beeper anyway
    while(JOYPAD_Read() == 8);//wait for the first push stops
    while(1){
        KeyRead = JOYPAD_Read();
        if (KeyRead != 0)
        {
            printf("The receive keyread is %d\r\n",KeyRead);

        }
        if (KeyRead == 8)
        {
            printf("Your game now returns! Enjoy! \r\n");
            LCD_ShowString(50, 300, 400, 40, 24, "Wait for release");//clear
            KeyRead = 0;// if key read == 4 -> it will catch error in the
another thing
            while(JOYPAD_Read() == 8);//wait for the second push stops
            LCD_ShowString(50, 300, 400, 40, 24, "                 ");//clear
            break;
        }
```

3.2.6 Use a new version of draw ball function to provide better game experience by faster drawing

By looking at the provided draw ball function, I find it is not very efficient because it has to draws by calling drawing points. So, I update the function to let it print a ball by calling draw line functions.

In my draw ball function, I use a static variable to store last called draw ball radius and an array to store the line length info needed for drawling the ball. If the new given ball radius is not the same with the old, it updates the array and the static radius. The draw ball function just draws the ball from left to right.

To clear the ball faster, I use a white square to clear the ball. Whenever, the transmitted ball color variable is WHITE, the clear ball function will be called and the ball will be covered by white rectangles to save time.

The whole code for drawing a circle is shown below.

```
void EIE3810_TFTLCD_DrawCircle(u16 x0,u16 y0, u8 r, u8 full,u16 color)
{
    static int lastPlotR = 0;
    static int BallInfo[200];
    int x1, y1, x2, y2, iterator;
    if (x0>480 || y0 > 800) return;
    // copied
    if (lastPlotR!=r)
    {
        //generate half plot points
        int i, j;
        BallInfo[0] = r;
        lastPlotR = r;
        //(" starts generation ball info\r\n");
        for (i = 1; i <= r; i++)
        {
            for (j=r;j>=0;j--)
            {
                if (j*j + i*i <= (r+1)*(r+1))
                {
                    BallInfo[i] = j;//y pixelNumber on x pos i is 2*j + 1
                    //printf("%d\r\n",j);
                    break;
                }
            }
        }
    }
    if (color == BACK_COLOR)
    {
        hideCircle(x0, y0, r);
        return;
    }
    for (iterator = 0; iterator <= r; iterator++)//plot the right part of the
ball
    {
        x1 = (int) x0 + (int) iterator;
        if (x1>=480) break;
        x2 = x1;
        y1 = (int) y0 + (int) BallInfo[iterator];
        y1 = (y1>799)? 799 : y1;
        y2 = (int) y0 - (int) BallInfo[iterator];
        y2 = (y2<0)? 0 : y2;
        //printf("%d x1 %d y1 %d x2 %d y2 %d \r\n",iterator,x1, y1, x2, y2);
        LCD_DrawLineColor(x1,y1,x2,y2,color);
    }
    for (iterator = 1; iterator <= r; iterator++) // plot the right part of
the ball
    {
        x1 = (int) x0 -iterator;
        if (x1<0) break;
        x2 = x1;
        y1 = (int) y0 + BallInfo[iterator];
        y1 = (y1>799)? 799 : y1;
        y2 = (int) y0 - BallInfo[iterator];
        y2 = (y2<0)? 0 : y2;
        //printf("%d x1 %d y1 %d x2 %d y2 %d \r\n",iterator,x1, y1, x2, y2);
        LCD_DrawLineColor(x1,y1,x2,y2,color);
    }
}
```

3.3 Test result for the basic game

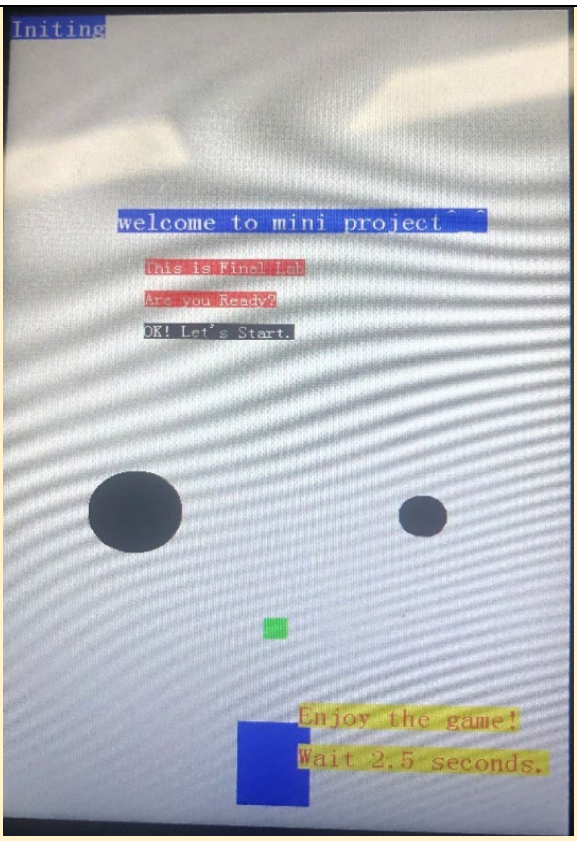
3.3.1 Quick view of the main function

After implementing each function, in the main function, we let the program go through each page / state of the game in sequence. The code in main function is shown below for reference.

```
int main(void)
{
    Stm32_Clock_Init(9);    //系统时钟设置->72MHz
    uart_init(72, 9600);//seems good
    delay_init(72);         //延时初始化 // enable systick timer
    LED_Init();             //初始化与 LED 连接的硬件接口

    InitTimers();// 50Hz 40 Hz 40Hz tim2 tim3 tim4

    JOYPAD_Init();
    EXTIX_Init();//init key and the interrupt correesponded with the key
    BEEP_Init();//comment this line for a freindly testing enviroment, this
enable beeper t o work
    LCD_Init();// init the TFTLCD screen

/**** The front page 0***/
    showWelcome();

/**** The game Level selection page 1***/
    gameStatus = 1;//used to update info (not meangingful)
    showSelect();   //hard level (actually the speed issue)
    LevelSelectFunction();//let the user select by using joypad or just the
up/down/right button
    PlayerSelectFunction();// let the user to select the player mode ->1. Play
with another player or play with chip controlled up beater

    /**** The USART page 2***/
    gameStatus = 2;//used to update info (not meangingful)
    showUSARTPage();//print out the usart page
    USARTWorking();// the working subroutine in using the usart to receive the
random bumber

    gameCountDown();// 7 segment code to let the thing count down from 3 to 0
with each takes 1 second
/**** counting down then the game starts***/
    gameLoop();//a while loop in the game

/*** the last page ***/
    showResult();// show the result on the screen based on the result (take it
longer there will be 彩蛋)

    while(1);//stuck in the loop nowhere else should be going
}
```

3.3.2 Screenshots during the game

Below are screenshots obtained in running the game, and is arranged in time order. Some of them is related to the extension part, and will be mentioned later.


Initing

welcome to mini project^ ^

This is Final Lab

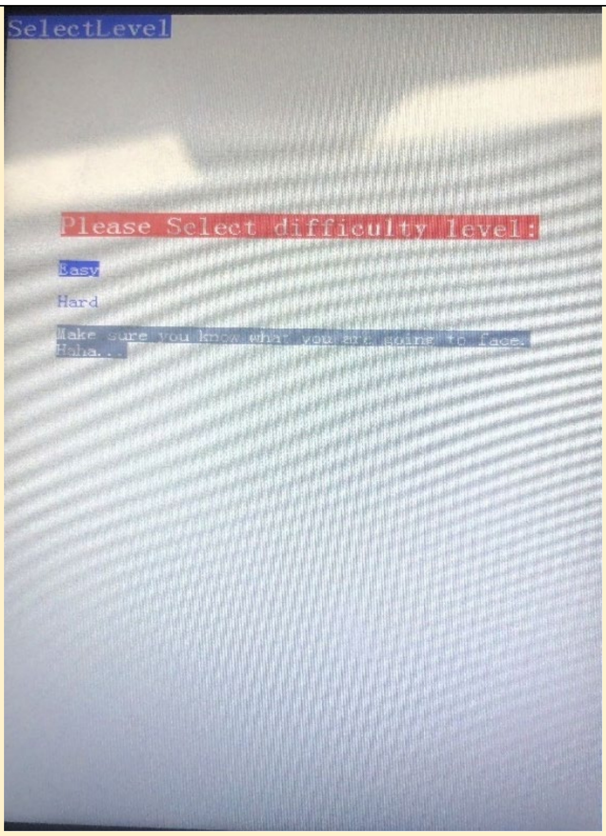
Are you Ready?

OK! Let's Start.

Enjoy the game!

Wait 2.5 seconds.

Figure 2. Front page of the game


SelectLevel

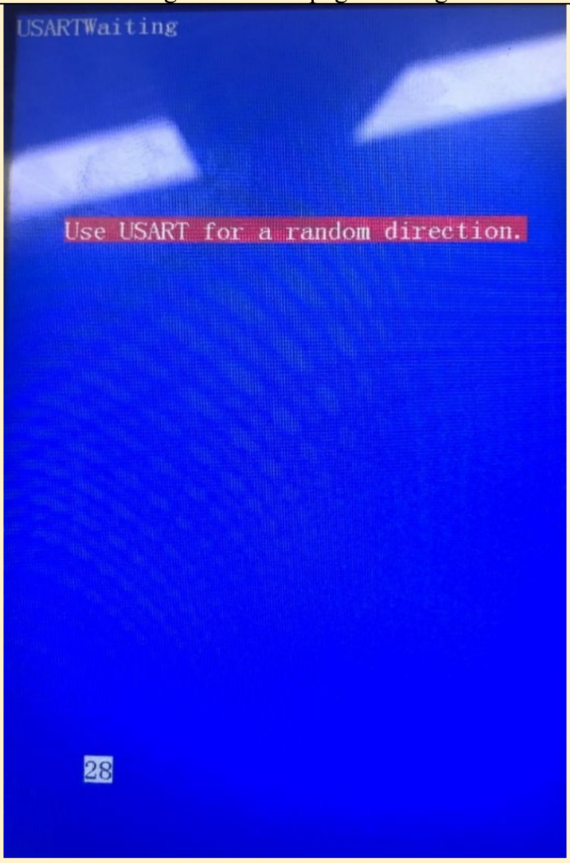
Please Select difficulty level:

Easy

Hard

Make sure you know what you are going to face.
Haha...

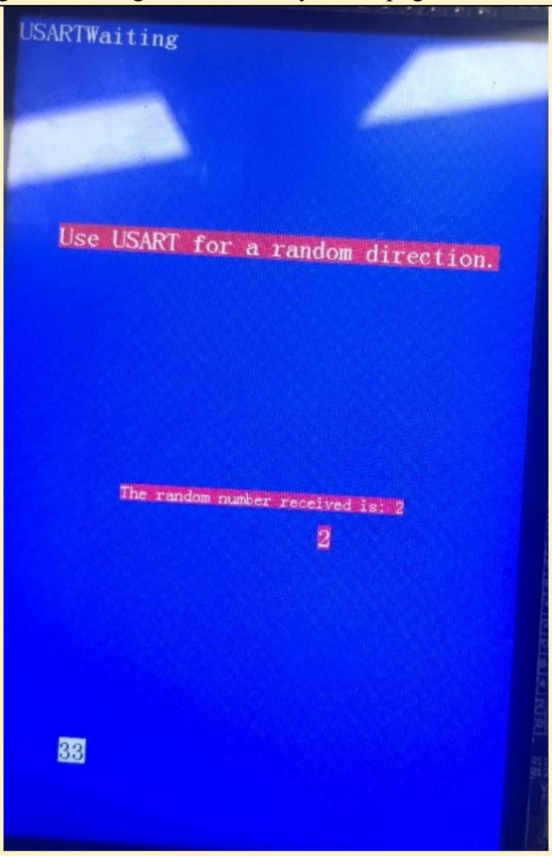
Figure 3. The game hard / easy select page


USARTWaiting

Use USART for a random direction.

28

Figure 4. USART page before receiving


USARTWaiting

Use USART for a random direction.

The random number received is: 2

2

33

Figure 5. USART page after number received

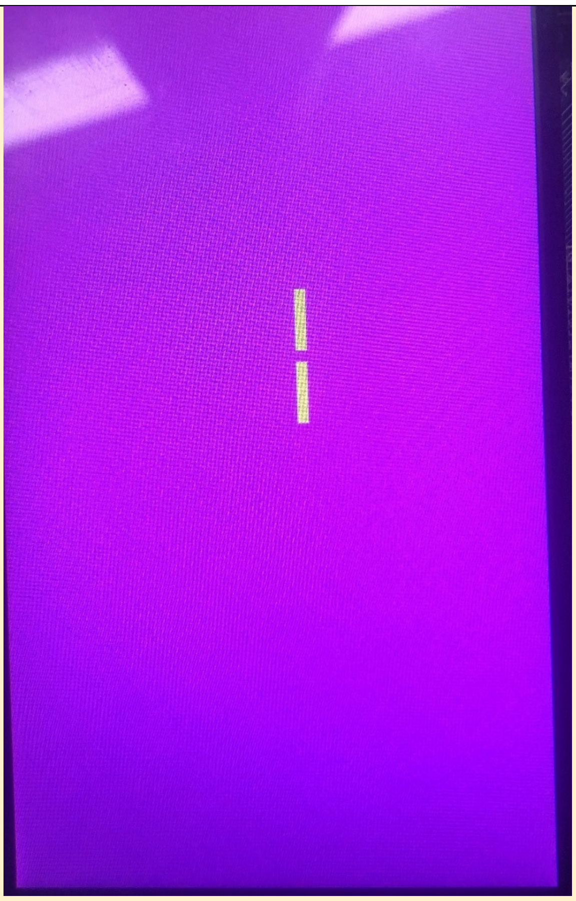


Figure 6. Count down page (number 1 in display)

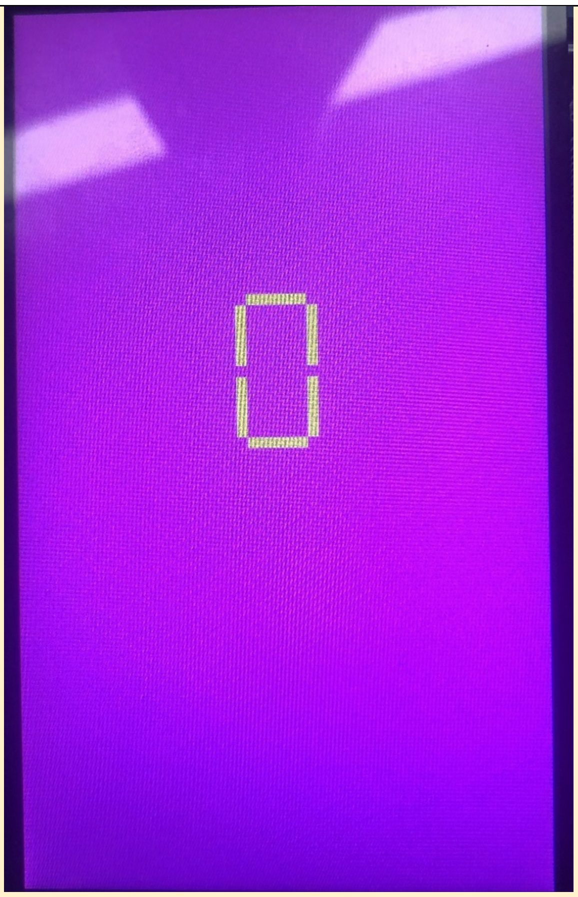
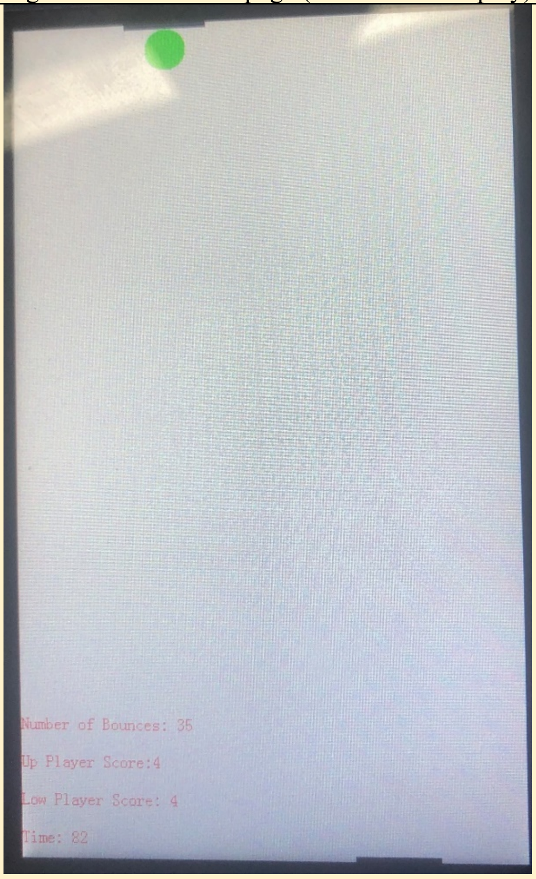
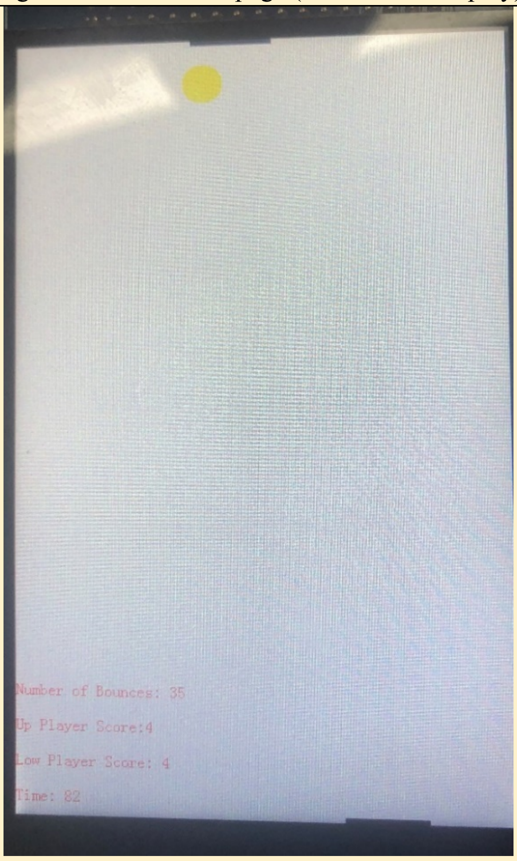


Figure 7. Count down page (number 0 in display)



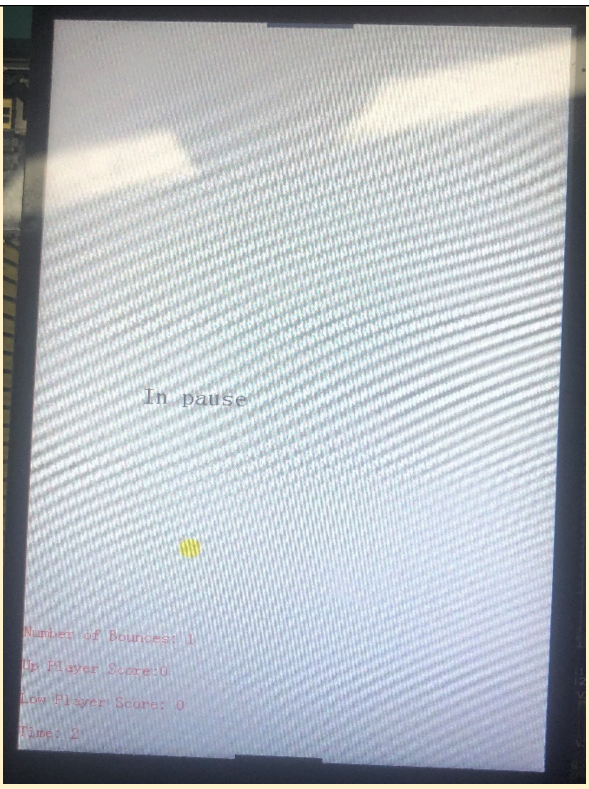
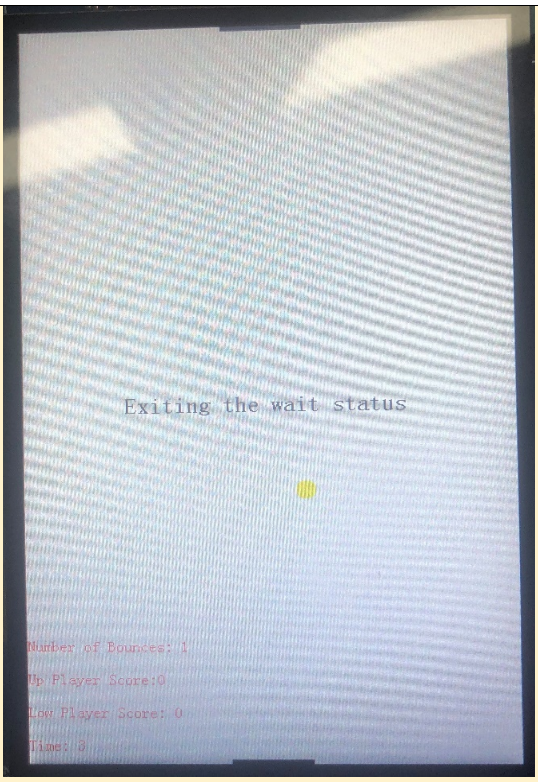
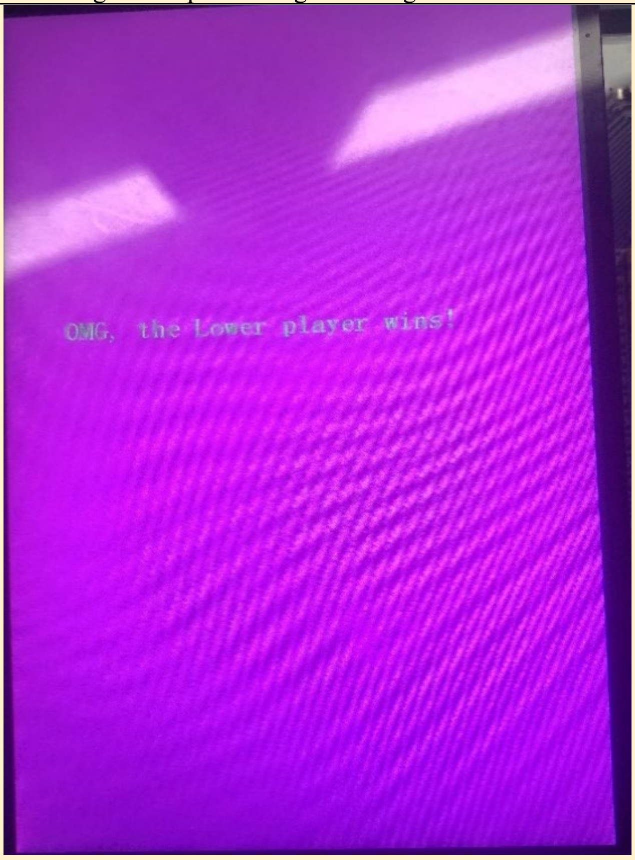
```
Number of Bounces: 35
Up Player Score:4
Low Player Score: 4
Time: 82
```

Figure 8. Before hitting the up beater



```
Number of Bounces: 35
Up Player Score:4
Low Player Score: 4
Time: 82
```

Figure 9. After hitting the up beater (bounces in display will be updated in 1/2 seconds)

 In pause Number of Bounces: 1 Up Player Score:0 Low Player Score: 0 Time: 2 Figure 10. pause the game using KEY DOWN	 Exiting the wait status Number of Bounces: 1 Up Player Score:0 Low Player Score: 0 Time: 3 Figure 11. Just after pause is finished
 OMG, the Lower player wins! Figure 12. The result page when one player wins	

4 Experiment B: Extension part

4.1 Improvement in what things?

1. In the game, the player can hit the UP-KEY button to adjust the ball speed to HARD if the current level is EASY.
2. In choosing page player can use JOYPAD to choose.
3. Add the single player mode (**Computer mode**) that the lower player can play with a chip controlled upper beater
4. The **ball's color changes** after each hit, and one of the colors is white. In white color, the human player cannot know the position of the ball until next hit or when the ball is very close to be hit (200pixels). This adds more challenge to the human player. Also, **the size of ball changes** after every 6 times hit by beaters. The size range includes radius 10, radius 20 and radius 40.
5. The draw **circle function is re-written** to provide a faster drawing speed of the ball and better game view experience.
6. After the result page the you will see a colorful final end page. One Chinese character "完" that is print out by moving a black balls. A circle in circle is shown in the end page. After about 10 seconds, the final page will be cleared by colorful rectangles from up to down, each occupy 10 pixels in y direction, and each takes .8 seconds.

4.2 How was each improvement implemented?

1. A speed up button (in easy mode):
   This is implemented by adding a if statement in the TIM2 handler and change the variable speed based if condition meets.

2. Choose hardness or player level using joypad & KEYS:
   This is achieved because in the scan joy pad timer (TIM3), it updates the KeyRead variable before entering the game (which is also what KEY interrupts update). So, pressing in joypad is equivalent to repeat pressing the corresponded key.

3. **The ball's color and shape changes:**
   This is achieved by using a global variable to point to an array of colors, also a global variable whichBall to point to an array of radiuses. We can update the ball by changing the color and size "pointer" which is implemented in the TIM2 Handler.
   Notice that one of the colors is white, you may experience a small period where you cannot find the ball. Player has to guess where it is to catch the ball without risking. However, if the player couldn't know, I still add a protection rule to let the ball shows in black when the ball was in white and is very close to the next beater (200 pixels in y direction). So, the player can always see the ball before hitting.

4. **The Single mode (Computer mode):**

This is achieved by adding an auto update upperMove variable subroutine in the TIM2 interrupt handler. The rule is that, when the ball is away from the upper bar, upperMove is set to the direction and distance to the center of the up screen. However, when the ball is going upward, the upperMove variable will be updated to the direction and distance to the x position where the center of the bar is the same as or very close to the center of the ball after next move.

5. A more colorful end page
   This is quite easy to implement by using function like draw ball, draw rectangle, and a lot color values.

4.3 Test result for the extension part

For part 1 and 2 of my improvement, it is not easy to show in pictures, you may find it in the **video** or testing.

As for the change of color and size of the ball, you may refer it in Figure 8., Figure 9. , and Figure 10. The ball color before hitting the beater is shown in Figure 8, then after hitting to Figure 9. And you may find the ball size is different between Figure 9 and Figure 10.

As for the single mode (4), you may see its behavior in the **video** or direct test it on the board.

The colorful end page is shown below for reference.

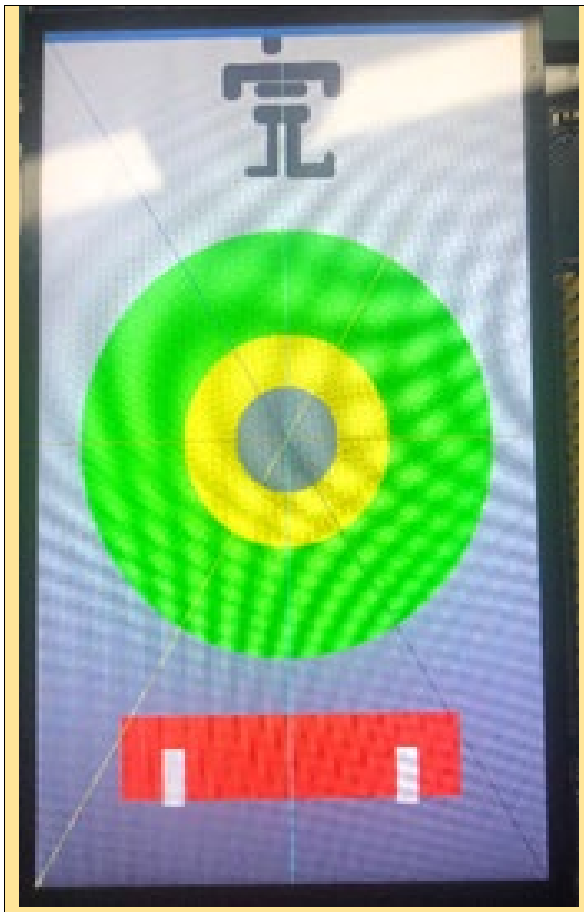


Figure 13. The final page after game over, then "完" is written by drawing with balls

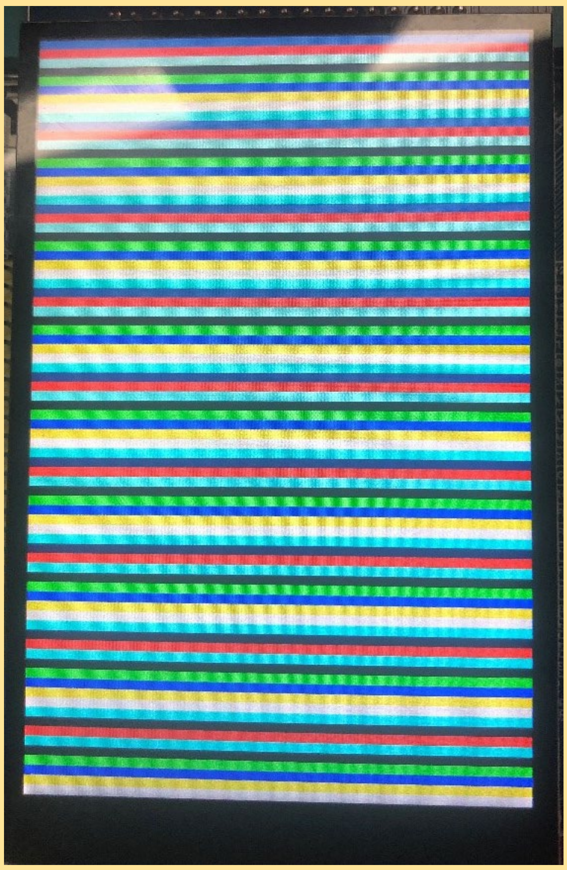


Figure 14. The final page after 100 seconds after game over

5 Conclusions

In this lab, we have finished the small ball game on the development board. In designing, I learnt a lot knowledge about

treating with unsigned type,
debugging with ST-Link,
Multi interrupts program design,
make use of Timer and external interrupts and possible error related with using interrupts,
using vscode in programming for better coding experience
and capture the baud rate using an oscilloscope.

I think I learnt quite a lot in this experiment. Good experience.