به نام خدا.

دستور کار سری ششم ریزپردازنده.

امیرحسین ادواری ۹۸۲٤۳۰۰٤ – زهرا حیدری ۹۸۲٤۳۰۲۰

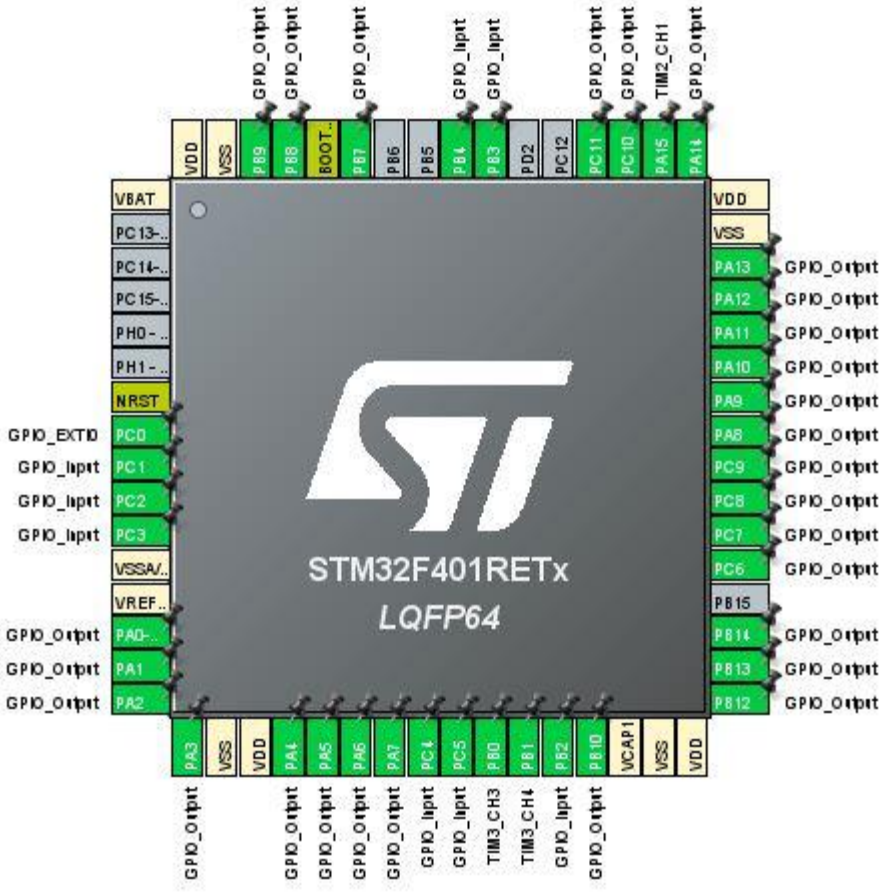
ورودی خروجی‌ها:

pin	PA15	PA14	PA13	PA12	PA11	PA10	PA9	PA8	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
symbol	RESET	D	C	B	A	E	RW	RS	D7	D6	D5	D4	D3	D2	D1	D0
type	IC	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O

PIN	PB14	PB13	PB12	PB10	PB9	PB8	PB7	PB4	PB3	PB2	PB1	PB0
SYM	SG	SF	SE	SD	SC	SB	SA	C3	C2	C1	RED PWM	GREEN PWM
TYPE	O	O	O	O	O	O	O	I	I	I	TIM3-CH3-PWM	TIM3-CH4-PWM

PIN	PC11	PC10	PC9	PC8	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
SYM	RED_PWM_EN	GREEN_PWM_EN	LED_RED	LED_YELLOW	LED_GREEN	LED_BLUE	RESET	RED	YELLOW	GREEN	BLUE	I0
TYPE	o	O	O	O	O	O	I	I	I	I	I	interrupt

به‌طور کلی pinout به شکل زیر است:



توابع مربوط به LCD و SEGMENTدر دستورکارهای مربوطه شرح داده شده اند.

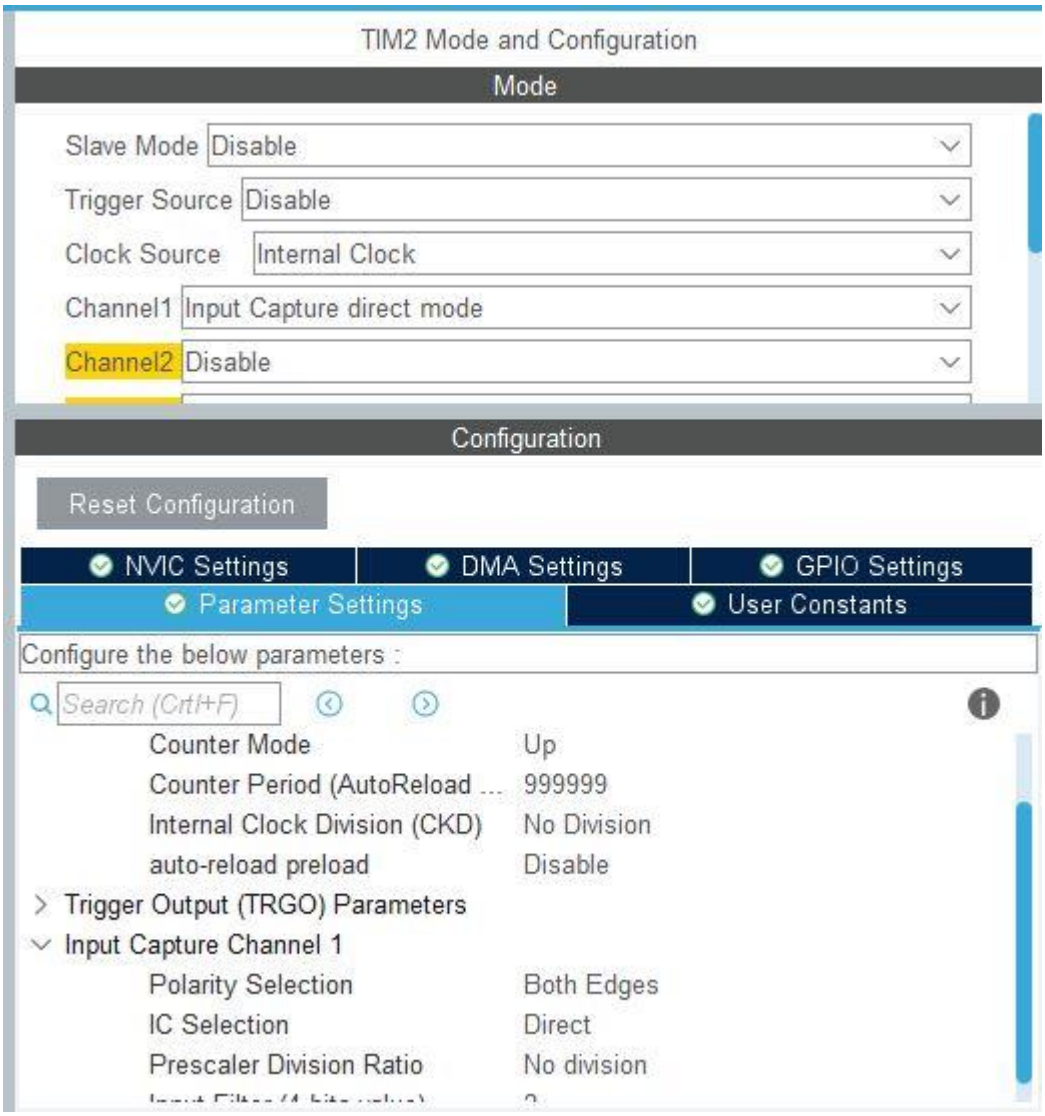
از طرف دیگر شیوه کار و راه‌اندازی Keypad در دستورکار مربوطه به تفصیل آورده شده است در اینجا صرفا توابع و روال‌های نو را بررسی می‌کنیم.

تایمر ٤ را برای محاسبه فاصله زمانی میان toggle کردن نام استفاده میکنیم.

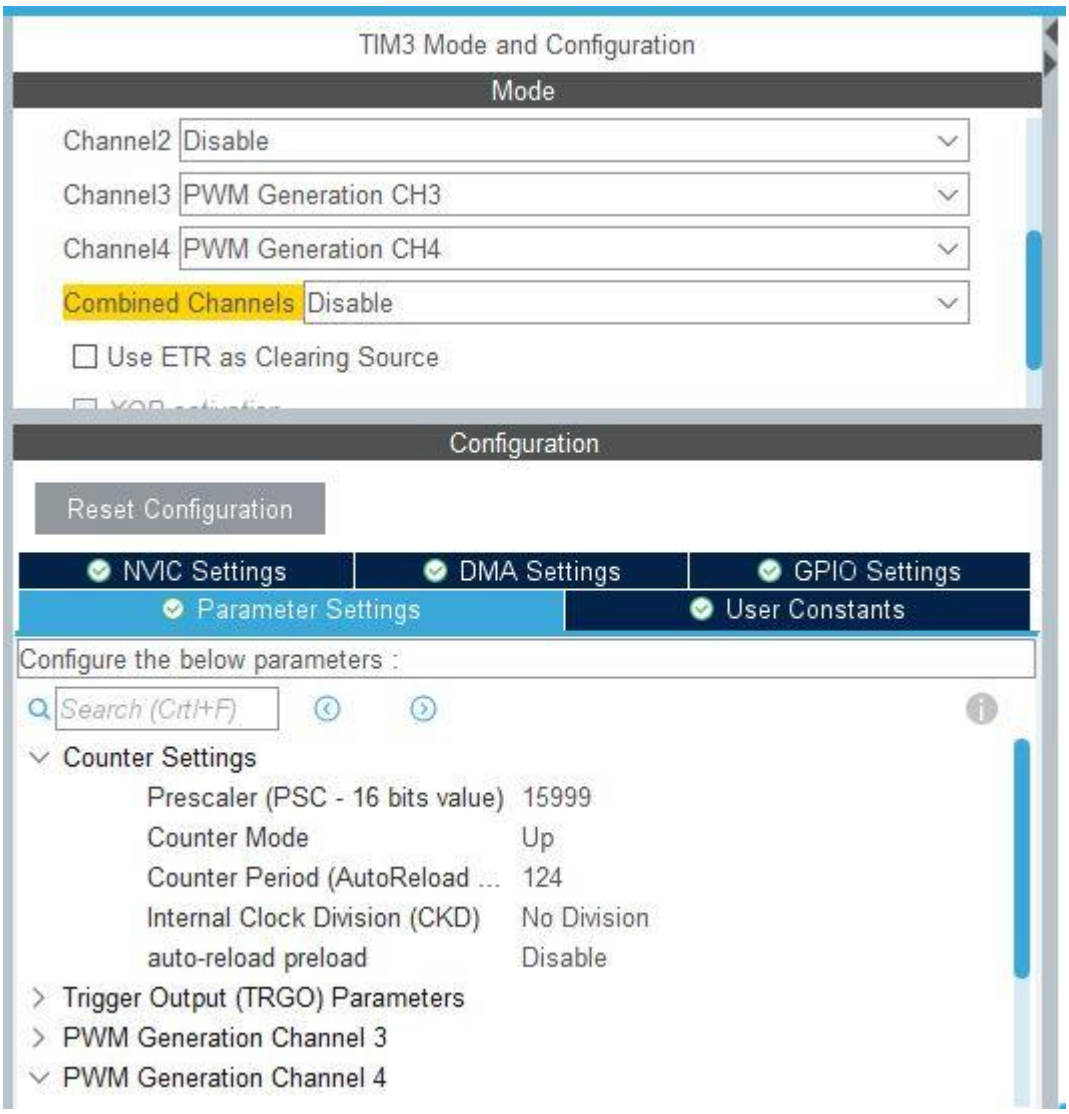
از چنل اول تایمر ۲ برای input-capture روی خط reset و از چنلهای ۳ و ٤ تایمر را برای pwm استفاده میکنیم.

تنظیمات برای ست کردن این تایمرها به شرح زیر در cubemx انجام میشود.

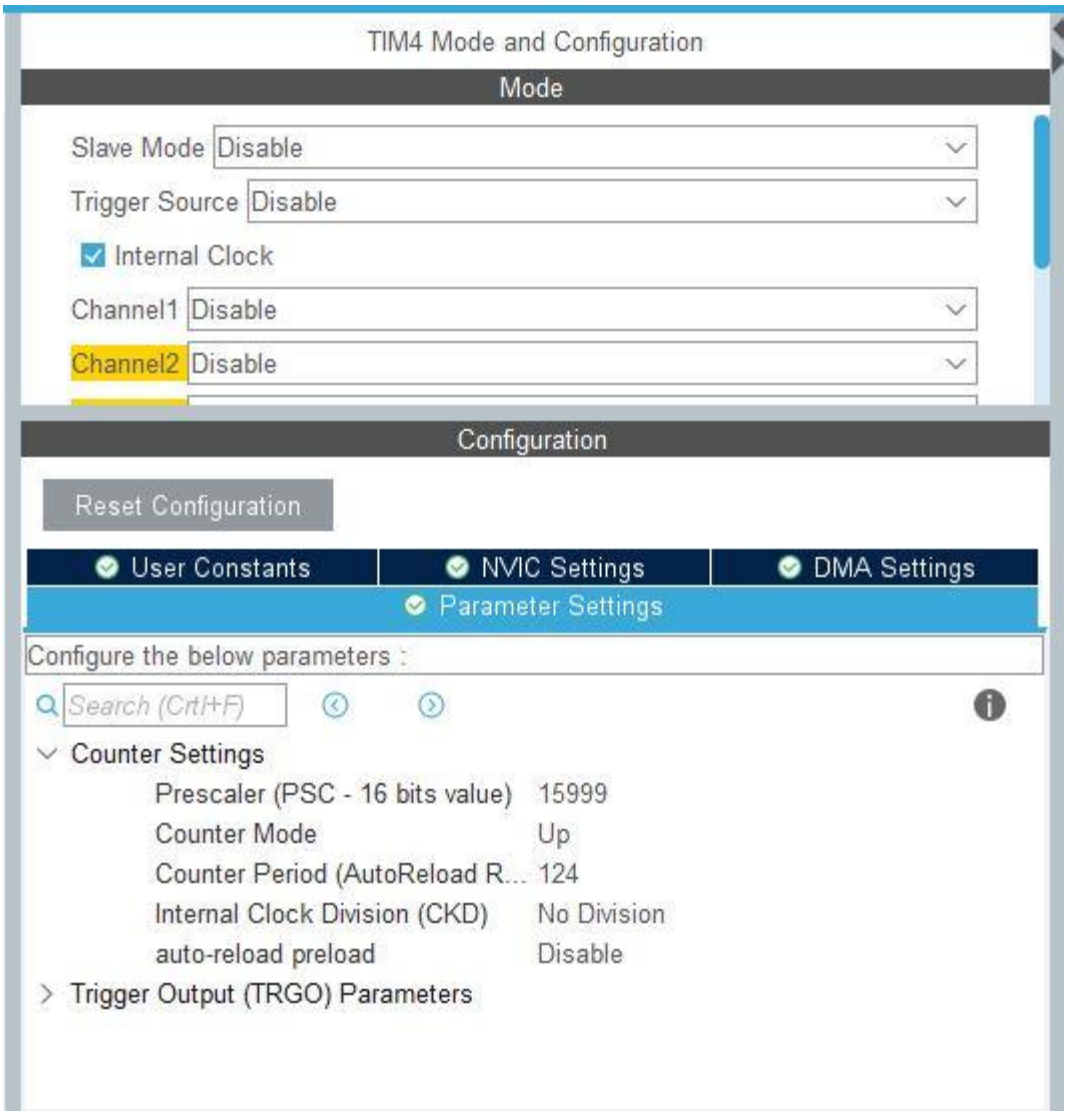
تایمر ۲:



تایمر ۳:


TIM3 Mode and Configuration
Mode
Channel2 Disable
Channel3 PWM Generation CH3
Channel4 PWM Generation CH4
Combined Channels Disable
Use ETR as Clearing Source
Configuration
Reset Configuration
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Configure the below parameters :
Search (Crtl+F)
Counter Settings
Prescaler (PSC - 16 bits value) 15999
Counter Mode Up
Counter Period (AutoReload ... 124
Internal Clock Division (CKD) No Division
auto-reload preload Disable
Trigger Output (TRGO) Parameters
PWM Generation Channel 3
PWM Generation Channel 4

تايمر ٤:



شرح کد و عملکرد :

```c
#include <stdbool.h>
#include <time.h>
#include <stdlib.h>

// GPIOA
#define RS (8)
#define RW (9)
#define E  (10)
#define A (11)
#define B (12)
#define C (13)
#define D (14)
#define RESET_IC (15)

// GPIOB
#define LED_GREEN_PWM (0)
#define LED_RED_PWM (1)
#define C1 (2)
#define C2 (3)
#define C3 (4)
#define SA (7)
#define SB (8)
#define SC (9)
#define SD (10)
#define SE (12)
#define SF (13)
#define SG (14)

//GPIOC
#define BLUE (1)
#define GREEN (2)
#define YELLOW (3)
#define RED (4)
#define RESET (5)
#define LED_BLUE (6)
#define LED_GREEN (7)
#define LED_YELLOW (8)
#define LED_RED (9)
#define GREEN_PWM_ENABLE (10)
#define RED_PWM_ENABLE (11)

#define MASK(x)   (1UL << (x))
```

طبق pinout نشان‌داده شده به منظور سهولت برخی از سیگنال‌هارا دیفاین میکنیم.

```
const uint16_t allowedMisses = 3;
const uint16_t gameTime = 60;

volatile uint16_t currentRow = 0;
volatile char inputID[8];
volatile uint16_t inputIDIndex = 0;
volatile uint16_t misses = 0;
char showingName[30];
enum State { STARTUP, SHOW_NAME, RUNNING, RESULT };
volatile enum State systemState = STARTUP;
volatile uint16_t remainingTime = gameTime;

enum ResultType { WIN, LOOSE, UNKNOWN};
volatile enum ResultType resultType = UNKNOWN;

// For IC
 uint32_t count = 0;
 bool errorPassed = false;
 uint32_t lastCaptured = 0;
```

allowedMisses و gameTime به ترتیب تعداد خطای مجاز و تایم کلی بازی را نشان میدهند. currentRow برای کار با keypad است که قبلا شرح داده شده است.

آرایه inputID قرار است شماره دانشجویی ورودی را حفظ کند. متغیر بعدی ایندکس فعلی آن را نگاه میدارد.

تعدادخطاها، متغیری برای نشاندادن و حفظ اسم متناظر با شماره دانشجویی ورودی، حالت برنامه، تایم باقیمانده، و شکل نتیجه نیز تعریف میشوند.

متغیرهای FOR IC برای شناسایی پالس مربوطه تعریف شده اند. Count شماره لبه فعلی را نشان میدهد و lastcaptured مقدار آخر capture شده.

```
void setName(char* name){
      uint16_t index = 0;
        while(*name != '\0'){
        showingName[index++] = (char) (*name);
        name++;
    }
}
```

این تابع صرفا اسم ورودی را در showingName قرار میدهد.

```
char clickedButton(){
        if( HAL_GPIO_ReadPin( GPIOB, MASK(C1)) == GPIO_PIN_SET) {
            while(( HAL_GPIO_ReadPin( GPIOB, MASK(C1)) == GPIO_PIN_SET));
            switch(currentRow) {
                case 0:
                    return '1';
                case 1:
                    return '4';
                case 2:
                    return '7';
                case 3:
                    return '*';
                default:
                    return '!';
            }
        }
        else if(HAL_GPIO_ReadPin( GPIOB, MASK(C2)) == GPIO_PIN_SET){
            while(( HAL_GPIO_ReadPin( GPIOB, MASK(C2)) == GPIO_PIN_SET));
            switch(currentRow) {
                case 0:
                    return '2';
                case 1:
                    return '5';
                case 2:
                    return '8';
                case 3:
                    return '0';
                default:
                    return '!';
            }
        }
        else if(HAL_GPIO_ReadPin( GPIOB, MASK(C3)) == GPIO_PIN_SET){
            while(( HAL_GPIO_ReadPin( GPIOB, MASK(C3)) == GPIO_PIN_SET));
            switch(currentRow) {
                case 0:
                    return '3';
                case 1:
```

این تابع همانطور که در دستورکار های پیش مشابه آن استفاده شده، کلید وارد شده را بعنوان کاراکتر برمیگرداند. لازم بذکر است میان فشردن 3 ثانیه‌ای یا لحظه‌ای RESET تفاوت قائل شده-ایم.

```
void resetAll(){
    resultType = UNKNOWN;
    HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_4);
    HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_3);
    HAL_GPIO_WritePin(GPIOC, MASK(RED_PWM_ENABLE), GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOC, MASK(GREEN_PWM_ENABLE), GPIO_PIN_RESET);
    systemState = STARTUP;
    clear_line_2();
    clearLEDs();
    remainingTime = gameTime;
    inputIDIndex = 0;
    count = 0;
    misses = 0;
    showMisses();
    commitCommand(0x80); // Move Pointer to firstLine

    MX_TIM3_Init();
    HAL_TIM_Base_Start_IT(&htim3);
}
```

این تابع همه شرایط برنامه را به حالت اولیه برمیگرداند.

```
void showColor(char color){
    switch(color){
        case 'b':
            HAL_GPIO_WritePin(GPIOC, MASK(LED_GREEN) | MASK(LED_RED) |MASK(LED_YELLOW) , GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOC, MASK(LED_BLUE), GPIO_PIN_SET);
            return;
        case 'y':
            HAL_GPIO_WritePin(GPIOC, MASK(LED_BLUE) | MASK(LED_RED) |MASK(LED_GREEN) , GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOC, MASK(LED_YELLOW), GPIO_PIN_SET);
            return;
        case 'r':
            HAL_GPIO_WritePin(GPIOC, MASK(LED_BLUE) | MASK(LED_GREEN) |MASK(LED_YELLOW) , GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOC, MASK(LED_RED), GPIO_PIN_SET);
            return;
        case 'g':
            HAL_GPIO_WritePin(GPIOC, MASK(LED_BLUE) | MASK(LED_RED) |MASK(LED_YELLOW) , GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOC, MASK(LED_GREEN), GPIO_PIN_SET);
            return;
    }
}
```

صرفا LED با رنگ ورودی را روشن و سایرین را خاموش میکند.

```
void showMisses(){
    HAL_GPIO_WritePin( GPIOB, 0xFF << 7, GPIO_PIN_RESET);
    HAL_GPIO_WritePin( GPIOB, segment(misses), GPIO_PIN_SET);
}
```

تعداد خطاها را روی SEGMENTنشان میدهد (تابع SEGMENT در دستور کار مربوط به سون سگمنت (چراغ راهنمایی) به تفصیل شرح داده شده)

```
char randomColor(){
```

صرفا یک کاراکتر رندم هرکدام به نمایندگی از یک رنگ برمیگرداند.

طبق شرح دستورکار، در تابع runGame ۳ مرتبه ۳ ثانیه‌ای LED روشن و ورودی چک می-شود.

سپس ۳ مرتبه ۲ثانیه‌ای و پس آن طبق دستورکار ۱۰ مرتبه ۱۰ مرتبه با کسر ۱۰۰ میلی‌ثانیه روال انجام می‌شود. در تابع showAndCheck نیز یک رنگ رندم ایجاد میکنیم، LED مربوطه را روشن میکنیم و زمان مشخصی مرتبا ورودی کاربر را چک میکنیم درصورتیکه کاربر کلید درست در تایم مربوطه وارد نکند MISSES اضافه میشود، اگر به تعداد مجاز خطا برسد، بازی تمام شده و loose کال می‌شود.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    if(GPIO_PIN_0) {
            char clickedBtn = clickedButton();

            if( systemState == STARTUP ){
                if( clickedBtn != 'e')
                    inputID[inputIDIndex++] = clickedBtn;
                if(inputIDIndex == 8){
                    clear_line_1();
                    commitCommand(0xC0);
                    if( inputID[7] == '4')
                        setName("Advari");
                  else
                        setName("Heidari");
                    systemState = SHOW_NAME;
                }
                else if( clickedBtn != 'e')
                    print(clickedBtn);
            }
            else if (systemState == RUNNING || systemState == RESULT || systemState == SHOW_NAME){
                if(clickedBtn == '*' || clickedBtn == '1')
                    resetAll();
            }
    }
}
```

اینتراپت درصورت فشردن کیپد یا Reset صادر می‌شود. در حالت STARTUP صرفا ورودی-هارا نشان میدهیم وقتی ۸ کاراکتر وارد شد، نام متناظر با آن را متعاقب ورود به حالت SHOW_NAME به‌صورت چشمک زن نشان می‌دهیم. اگر کلیدهای مربوط به ریست وارد شوند روتین resetCall را صدا می‌زنیم.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){


    if( !errorPassed){
        errorPassed = true;
        return;
    }
    if( systemState != SHOW_NAME)
        return;

    count++;
    // first and third edges (rising)
    if(count == 1 || count == 3){
        lastCaptured = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
    }
    // second edge  (falling)
    else if(count == 2){
        uint32_t currentCaptured = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
        if( currentCaptured - lastCaptured < 2000/16)
            count = lastCaptured = 0;
    }
    // check wether appropriate pulse has been s een or not.
    else if(count == 4) {
        uint32_t currentCaptured = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
        if( currentCaptured - lastCaptured >= 2000/16)
         {
            count = lastCaptured = 0;
            systemState = RUNNING;
            remainingTime = gameTime;
        }
        else
            count = lastCaptured = 0;
    }


}
```

در این تابع(که همان هندلر مربوط به input capture میباشد) زمانیکه ٤ لبه (پایین رونده و بالارونده) دیده شود، به‌طوریکه فاصله بین لبه‌های بالارونده بیش از ۲ ثانیه باشد، سیستم وارد حالت RUNNING شده و بازی اجرا می‌شود (()runGame در main اجرا می‌شود)

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if( htim->Instance == TIM4) {
        if( systemState == SHOW_NAME){
            static bool printed = false;
            if(printed)
                clear_line_2();
            else
                printInLine(showingName);
            printed = !printed;
        }
    }
    if( htim->Instance == TIM3){
        if( systemState == RUNNING) {
            clear_line_2();
            remainingTime--;
            if(remainingTime == 0){
                if(misses < allowedMisses )
                    win();
                else
                    loose();
            }
            else
                printDigit(remainingTime);
        }
    }
}
```

تایمر 4 که تایمر Base است مسئولیت چشمک زدن نام را دارد.

تایمر 3 نیز در حالت Base اگر سیستم درحالت RUNNING باشد و تایم تمام نشده باشد، در هرثانیه آنرا Decrement میکند و نشان می‌دهد. اگر تایم تمام شود و تعداد خطاها کمتر از آستانه تعریف شده باشد؛ بازی با برد و در غیر اینصورت با باخت خاتمه یافته است.

```
void loose(){
    if(resultType == WIN)
        return;
    resultType = LOOSE;
    clearLEDs();
    systemState = RESULT;
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
    HAL_GPIO_WritePin(GPIOC, MASK(RED_PWM_ENABLE), GPIO_PIN_SET);
    clear_line_2();
    printInLine("Looser");
}
void win(){
    if(resultType == LOOSE)
        return;
    resultType = WIN;
    clearLEDs();
    systemState = RESULT;
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);
    HAL_GPIO_WritePin(GPIOC, MASK(GREEN_PWM_ENABLE), GPIO_PIN_SET);
    clear_line_2();
    printInLine("Winner");
}
```

در تابع loose() اگر بازی به حالت WIN باشد loose نبایستی اجرا شود (برای جلوگیری از اختلال زمانی)

در غیر اینصورت این تابع حالت نتیجه را LOOSE و حالت سیستم را در حالت RESULT میگذارد. سپس PWM مربوط به باخت (قرمز) را فعال میکن و عبارت لوزر را چاپ میکند.

متقارن همین تابع WIN وجود دارد که با توضیحات بالا کد آن واضح است.

```
void clearLEDs(){
    HAL_GPIO_WritePin(GPIOC, MASK(LED_GREEN) | MASK(LED_RED) |MASK(LED_YELLOW) | MASK(LED_BLUE) , GPIO_PIN_RESET);
}
```

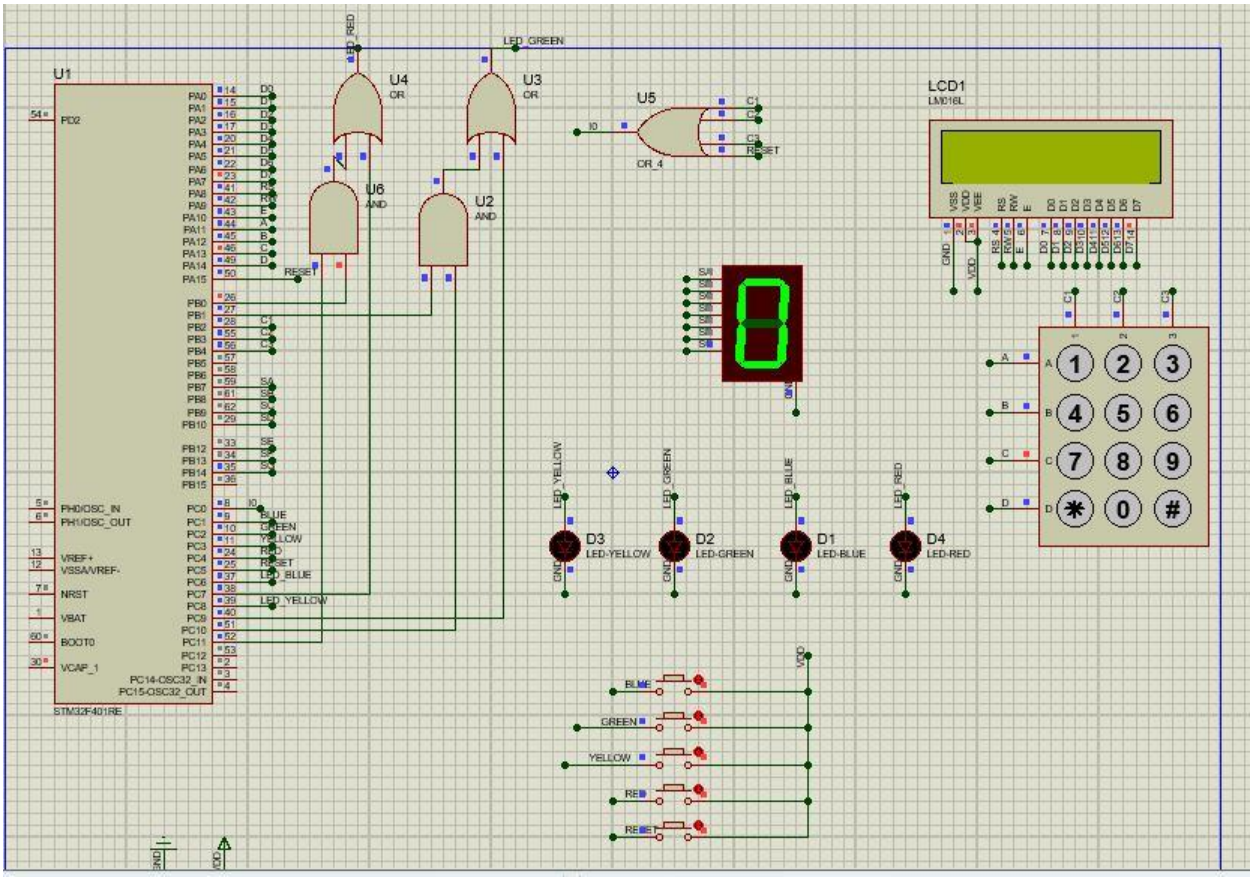
این تابع صرفا همه LED هارا خاموش میکند.

```
char clickedLED(){
```

این تابع با سرکشی بررسی میکند که کاربر کلید کدام رنگ را (حین بازی) فشرده است(یا اینکه هیچ دکمه‌ای نفشرده است).

در تابع main پس از فعالسازی ها و اولویت‌بندی های مربوطه صرفا روال مربوط به 1 گذاشتن روی سطرهای کیپد اجرا میشود که پیشتر شرح داده شد و نیز بررسی می‌شود که آیا بایستی بازی اجرا شود یا خیر.

پروتئوس



نکته : برای اجرای صحیح RESET در هنگام شروع کار ابتدا یکبار آنرا به طور لحظه ای بفشارید، پس از آن سیستم به درستی کار میکند.