به نام خدا

گزارش کار آزمایشگاه ریزپردازنده

آزمایش 6

مدرس : مهندس بی‌طالبی

تارا برقیان

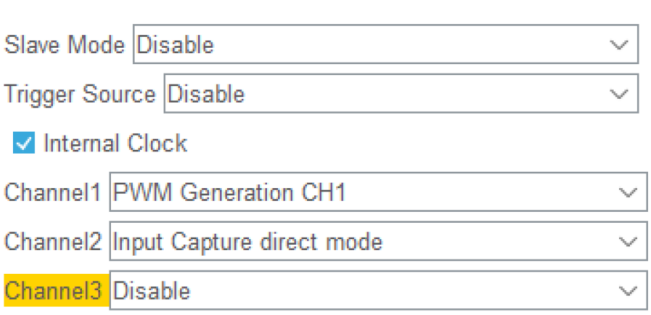
مهرشاد سعادتی نیا

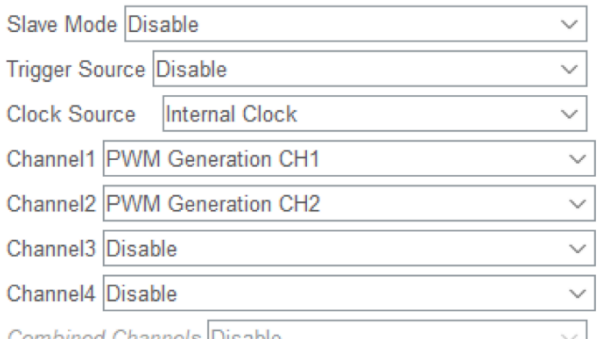
نیم سال اول 01-1400

استفاده از PWM برای LED های برد و باخت:

مشابه تمرین قبلی ابتدا فرایند فعالسازی PWM را در cubeMX انجام می دهیم:

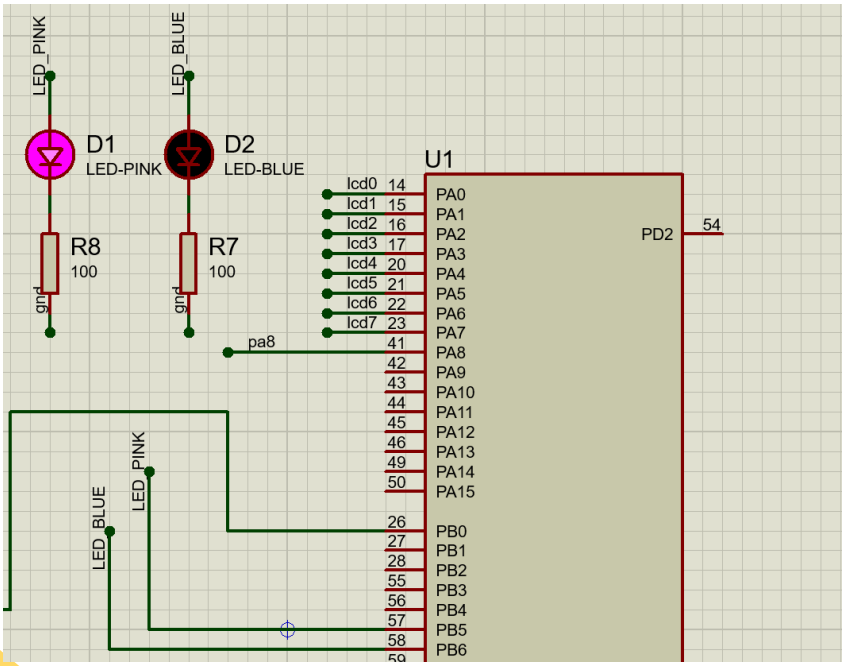
در ای تمرن برای دو LED قرمز و آبی به دو پین مجزای متصل به PWM احتیاج داریم که ما از کانال 1 تایمر 4 و کانال 2 تایمر 3 استفاده کردیم (کانال یک نتوانست LED را روشن کند)





در قسمت پالس هم duty cycle را تنظیم می کنیم و از mode 1 استفاده می کنیم.

های آبی و قرمز(صورتی) وصل می کنیم.LED هستند به PB6 و PB5سپس در داخل مدار پین های این کانال ها را که به



در داخل کد وقتی که بازیکن می بازد تابع loser_handler فراخوانی میشود و وقتی می برد winner_handler و کافیست در داخل این توابع PWM مربوط به هر LED را فعال کنیم ، مثلاً وقتی بازیکن می بازد می خواهیم LED صورتی را هر 500 میلی ثانیه روشن کنیم پس PWM کانال 2 تایمر 3 را که به آن مربوط است روشن می کنیم.

```
void loser_handler()
{
      lcd_clear();
      delay(10);
      lcd_puts("loser");
      delay(10);
      lcd_putchar(pressed);

  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);

}
```

در تصویر زیر هم مقدار Pulse که بیانگر duty cycle است را برای یکی از PWM ها مشاهده می کنیم:

```
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 499;
```

(499 برای پریود 999)

کیپد:

ابتدا برای راحتی کار منغیر های کلیدی که خود cube برای ما جنریت می کند را در آرایه هایی قرار دادیم تا دسترسی به آنها راحت تر باشد.

```
// Row Pins
const uint16_t ROW_GPIO_PIN[] =
{
  KP_A_Pin,
  KP_B_Pin,
  KP_C_Pin,
  KP_D_Pin
};

// Column Pins
const uint16_t COL_GPIO_PIN[] =
{
  KP_1_Pin,
  KP_2_Pin,
  KP_3_Pin,
  KP_4_Pin
};
```

Kp_a تا kp_d متصل به ردیف های کیپد و kp_1 تا kp_4

پین های متصل به ستون هستند.

پین های ستونی را در cube بعنوان اینتراپت خارجی تنظیم کردیم در نتیجه در کد به کمک تابع callback آن می توانیم آنها را تشخیص داده و اقدام مربوطه را انجام دهیم.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

  if(GPIO_Pin == KP_1_Pin)
  {
    col = 0;
    pressed = get_key_pressed(col);
  }

  if(GPIO_Pin == KP_2_Pin)
  {
    col = 1;
    pressed = get_key_pressed(col);
  }

  if(GPIO_Pin == KP_3_Pin)
  {
    col = 2;
    pressed = get_key_pressed(col);
  }

  if(GPIO_Pin == KP_4_Pin)
  {
    col = 3;
    pressed = get_key_pressed(col);
  }
}
```

به کمک این تابع کالبک می توانیم ستون را

تشخیص دهیم، سپس آنرا به تابع get_key_pressed پاس می دهیم تا ردیف را و در نتیجه کلید فشرده شده را تشخیص دهد.

الگوریتم تشخیص دادن کلید مشابه تمرین 3 است فقط این بار از توابه حال برای عملیات روی پین ها کمک گرفتیم.

بدنه ی تابع get_key_pressed در تصویر صفحه ی بعد قابل مشاهده است.

که با کمک اینتراپت ستون را تشخیص می دهیم و سپس با یک کردن row های مختلف تشخیص می دهیم که کدام کلید فشرده شده است. که منطق آن مشابه قبل است.

```
char get_key_pressed (uint8_t c)
{
  char key = 'N';

  for(uint8_t r = 0; r < 4; r++)
  {
    HAL_GPIO_WritePin(GPIOC, ROW_GPIO_PIN[r], GPIO_PIN_RESET);
    for(volatile int dad =0 ; dad<=delay_milis ; dad++ ){}
    HAL_GPIO_WritePin(GPIOC, ROW_GPIO_PIN[r], GPIO_PIN_SET);
    for(volatile int dad =0 ; dad<=delay_milis ; dad++ ){}

    if(HAL_GPIO_ReadPin(GPIOC, COL_GPIO_PIN[c]) == GPIO_PIN_SET)
    {
      for(volatile int dad =0 ; dad<=delay_milis ; dad++ ){}
      if(HAL_GPIO_ReadPin(GPIOC, COL_GPIO_PIN[c]) == GPIO_PIN_SET)
      {
        row=r;
        key = key_cal(r,c);
        if(key=='X')
          s=start;
        break;
      }
    }
  }


  while(HAL_GPIO_ReadPin(GPIOC, COL_GPIO_PIN[c]) == GPIO_PIN_RESET)
  for(volatile int dad =0 ; dad<=delay_milis ; dad++ ){}


  flg_check_input = true;
  flg_did_somthing = 0;
  return key;

}
```

اینپوت کپچر:

به کمک قابلیت اینپوت کپچر می توانیم فاصله ی بین دو کلیک متوالی را حسب کنیم و بازی را شروع کنیم که کد آن در زیر قابل مشاهده است.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(s!=start)
    return;

  //in chon ye bar bara hamishe bud dg tush o ziad neveshtm
  if(htim->Instance == TIM1)
  {
    input_capture = __HAL_TIM_GetCompare(&htim4, TIM_CHANNEL_1);
    int diff = (input_capture - prev) + 500*sec ;

    if (diff >= 3000 && diff <= 8000) {
      GPIOB->ODR |= (1L << 9); //for test
      sec = 180 ;
      s=level1;
      flg_did_somthing = 0;
    }
    else
      sec = 0 ;
  }
  prev = __HAL_TIM_GetCounter(&htim1);

}
```

با کمک متغیری مقدار کپچر شده ی قبلی را نگه میداریم و اگر اختلاف کپچر قبلی و فعلی بین 3000 و 8000 بازی را شروع می کنیم.

تنظیمات داخل cube مشابه تمرین قبلی است و از تایمر 1 کانال مربوطه آن بر روی PA8 قرار دارد و همچنین از تایمر 4 استفاده شده که کانال یک آن استفاده شده.

روند بازی:

برای بازی استیت های مختلف تعریف کرده ایم و به کمک یک enum بنام استیت آنها را مدیریت کردیم:

```
volatile enum STATE
  {
    start,
    level1,
    level2,
    level3,
    winner,
    loser
  };
```

هر کدام از این استیت ها تابع هندلر مربوط به خودش را دارد که جلوتر به شرح آن خواهیم پرداخت.

پیش از شروع بازی به کمک تابع زیر با تناوب یک ثانیه دو پیام گفته شده را نمایش می دهیم.

```
void start_handler()
{
  if(flg_start_sec)
  {
  flg_start_msg = !flg_start_msg;
  lcd_clear();
  if(flg_start_msg)
    lcd_puts(str1);
  else
    lcd_puts(str2);

  flg_start_sec = false;
  }
}
```

فلگی که در شرط چک شده نشان می دهد که در ابتدای شروع بازی هستسم و باید پیام های تناوبی را نشان دهد.

این فلگ در تابع TIM1_IRQHandler که به جای periodElapsedCallabck استفاده شده ست میشود.

همانطور که پیشتر در تابع IC_Callback دیدیم، اگر فاصله دو کلیک بیش از 3 ثانیه باشد استست بازی به level1 تغیر می کند و وارد بازی میشویم.

```
void level1_handler()
{
  mode = 6 ;

    lcd_gotoxy(14,1);
    int tmp = sec/2;
    lcd_put_int(tmp);
    lcd_gotoxy(0,0);

    /*if(flg_did_somthing >= mode && !flg_good_choice)
    {
      s = loser ;
      pressed = 'v';
      return;
    } */

  if(sec%mode == 0){

    flg_did_somthing = 0;
    flg_good_choice = false;

    random = make_random%10;
    lcd_clear();
    lcd_put_int(random);
  }

    check_user_input();


}
```

در اینجا ما از یک متغیر کمکی بنام sec کمک می گیریم که هر بار اینتراپت 500 میلی ثانیه ای تایمر یک پرتاب می شود این متغیر داخل IRQ_Handler ++ میشود و اینجا کافیست یک متغیر بنام modeتعریف کنیم که مقادیر آن در مراحل اول دوم و سوم به ترتیب 6 و 4و 2 می باشد و دلیل انتخاب آنها هم اینست که مثلا در مرحله یک که می خواهیم 3 ثانیه زمان داشته باشیم 6 * 0.5 برابر 3 میشود و 0.5 ثانیه هم دوره تناوب تایمر ماست.

از باقیمانده ی sec بر mode می فهمیم که آیا زمان کاربر برای وارد کردن تمام شده یا خیر.

اکر وارد این شرط شویم فلگ مربوطه (good_choice) را فالس می کنیم که بعدا از آن استفاده می کنیم.

بعد از این حلقه هم ورودی کاربر چک می شود تا ببنیم آیا بازنده است یا بازی ادامه پیدا می کنید.

لازم به ذکر است که این موضوع به معنی polling نیست و در واقع داریم مطابقت وروردی کاربر را با ورودی اصلی بررسی می کنیم.

```
void check_user_input()
{
  if(flg_check_input)
  {

    flg_did_somthing = 0;
    flg_did_somthing = false;
    int temp = pressed - '0';

    if(random != temp)
    {
      s = loser;
      flg_check_input = false;
      flg_winner = false ;
    }
    else
    {
      flg_check_input = false;
      flg_good_choice = true ;
    }

  }


}
```

در صورتی که استیت ما به loser تغییر پیدا کند وارد همان تابع loser_handler میشویم که در ابتدای گزارش ذکر کردم و LED به کمک PWM چشمک می زند.

مراحل بعدی هم مشابه مرحله ی اول است اما متغیر mode آن تفاوت دارد.

برای بررسی فشرده شدن سه ثانیه دکمه ی دوم به کمک polling عیناً از روش تمرین دوم استفاده کردیم که توضیحات آن در آن تمرین به طور مفصل داده شده بود و صرفاً از کد آن استفاده خواهیم کرد.

```
bool checkBTN2 ( void )
{
  if(GPIOB->IDR & MASK(0))
  {
    return true;
  }
  return false;
}

bool check_time_BTN2 (void)
{
  bool flg = true;
  volatile uint32_t cn = 0;
  volatile uint32_t _3_sec = 30*delay_milis ; // 3 sec = 6 * 0.5 sec :))

  for( cn=0; cn<= _3_sec ; cn++ )
  {
    if( !(checkBTN2()) )
    {
      flg = false;
      break;
    }
  }
```

در داخل بدنه تابع main هم بررسی می کنیم که آیا این دکمه فشرده شده بوده است یا نه و در اینصورت وارد یک حلقه بی پایان می شویم و برنامه تا ریست نشود دوباره کار نخواهد کرد.

```
if(check_time_BTN2())
{
  lcd_clear();
  lcd_puts("STOP");
  while(true)
  {
  }
}
```

تابع زیر هم در توضیحات جا مانده بود که برای نگاشت کردن عدد سطر و ستون به ورودی عددی کیپد مورد استفاده قرار می گیرد.

```
char key_cal(uint8_t r, uint8_t c){

  if(r == 0){
    switch(c) {
      case 0: return '7';
      case 1: return '8';
      case 2: return '9';
      case 3: return '/';
    }
  }
  else if(r == 1){
    switch(c) {
      case 0: return '4';
      case 1: return '5';
      case 2: return '6';
      case 3: return 'X';
    }
  }
  else if(r == 2){
    switch(c) {
      case 0: return '1';
      case 1: return '2';
      case 2: return '3';
      case 3: return '-';
    }
  }
  else{  // r == 3
    switch(c) {
      case 0: return 'C';
      case 1: return '0';
      case 2: return '=';
      case 3: return '+';
    }
  }
```

در انتها لازم میدانیم تعدادی از فلگ هایی که در برنامه استفاده شده و در طول گزارش به آنها اشاره نکردیم را مختصراً توضیح دهیم.

good_choice برای تشخیص انتخاب عدد درست توسط کابر استفاده شده

did_somethingبرای آنست که اگر کاربر در وقت مشخص عددی وارد نکرده بود ببازد.

بقیه موارد هم یا در طول گزارش توضیح داده شده و یا عملکرد آن از اسم و مقدارش پیداست.

```
volatile int input_capture = 0;
volatile int prev = 0;

volatile char* str1 = "Press + Start" ;
volatile char* str2 = "Trace & Place" ;

volatile bool flg_start_msg = false;
volatile bool flg_start_sec = false;
volatile bool flg_check_input = false;
volatile bool flg_good_choice = true;
volatile bool flg_winner = true;
volatile int flg_did_somthing = 0;
volatile int sec = 0 ;

volatile bool period = false;

volatile int mode = 250000;
volatile int random = 0;
volatile int make_random = 1234;

volatile int  row = 10;   //to keep pressed pin of port b
volatile int  col = 10;  //to keep pressed pin of port c
volatile int  delay_milis = 9000;
volatile char pressed = '#';
```