به نام خدا

گزارش کار آزمایشگاه ریزپردازنده

آزمایش ۲

مدرس : مهندس بی طالبی

تارا برقیان

مهرشاد سعادتی نیا

نیم سال اول ۰۱-۱۴۰۰

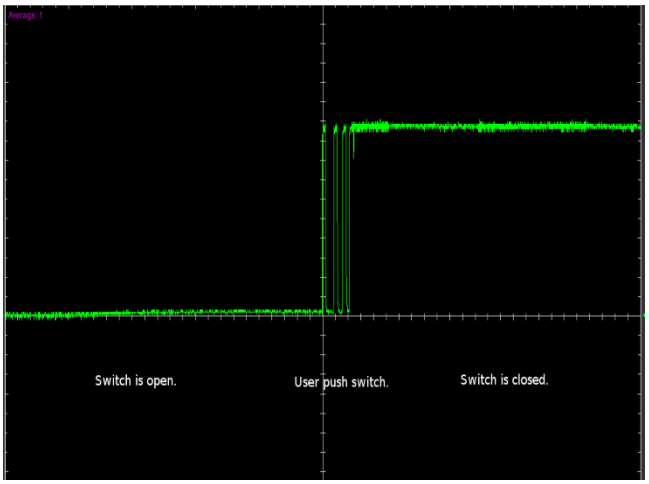
فهرست

مقدمه :

در این آزمایش هدف ، طراحی یک شمارنده نزولی و صعودی با قابلیت های استپ و خاموش شدن بود که با استفاده از کتابخانه ی STM324xx و ابزار keil و proteus انجام شد.

پاسخ سوالات تحلیلی:

۱. وقتی با یک کلید را فشار می دهیم تصور می کنیم که کلید به یکباره بسته یا باز می شود اما حقیقت اینست که با فشرده شدن کلید فقط در کسری از میکرو ثانیه بین دو فلز اتصال برقرار می شود و پس از آن دو فلز از هم جدا شده و مجدد با زمانی کمی طولانی تر از قبل به هم متصل می شوند، این اتفاق به دفعات تکرار می شود تا زمانی که کلید کاملاً بسته شود و همین باعث نوسانات زیادی بین ۰و ۱ می شود که در شکل موج زیر قابل مشاهده است. به این پدیده switch bouncing گفته می شود.



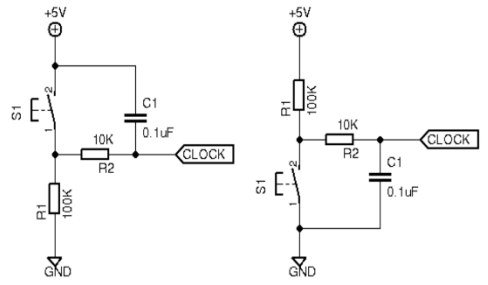
راه حل:

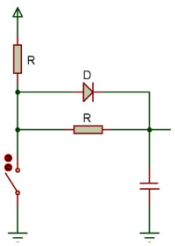
سخت افزاری: استفاده از خازن سرامیکی

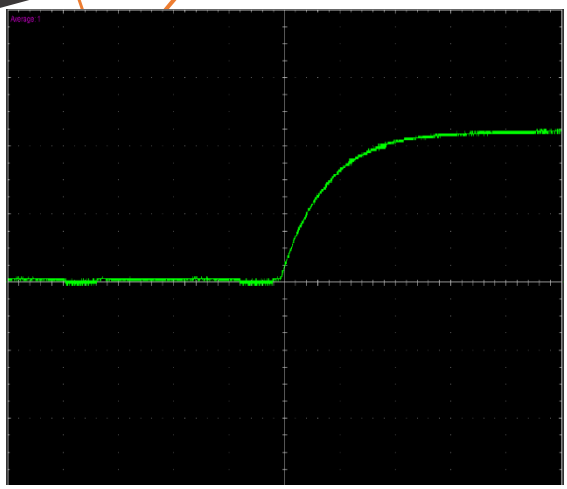
همانطور که در اسلاید های درس هم گفته شده بود یک راه حل برای این مشکل استفاده از خازن در مدار است که بصورت های زیر می توان آنرا بست و باعث می شود که نویز مدار گرفته شود و کلید بصورت یک موج نرم از صفر به یک تغییر وضعیت دهد.

با اضافه کردن یک دیود سرعت حذف پرشها و نویزها افزایش می یابد. دیود مطابق تصویر نشان داده شده به مدار اضافه می گردد:

حال اگر بخواهیم شکل موج را در این وضعیت بررسی کنیم به وضوح مشخص است که مشکل switch bouncing حل شده است.







راه حل:

نرم افزاری: استفاده از interrupt

بعضی از برنامه نویسان برای حل این مشکل از delayاستفاده می کنند، اما ایراد این روش اینست که در صورت تکرار زمان میکروکنترلر را هدر می دهد که خوب نیست.

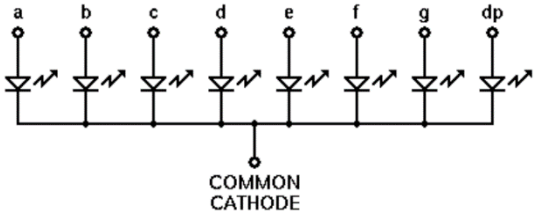
راه حل بهتر نرم افزاری برای این مشکل interrupt است تا از بسته شدن کلید مطمئن شویم و بعد به ادامه ی کار بپردازیم.

۲. دلیل استفاده از مقاومت محدود کننده اینست که جریانی که به LED وارد می شود را محدود کند تا از سوختن آن جلوگیری شود. مقدار مقاومت مورد نیاز برای یک LED از رابطه زیر حاصل می شود.

$$R = \frac{V - V_{LED}}{I}$$

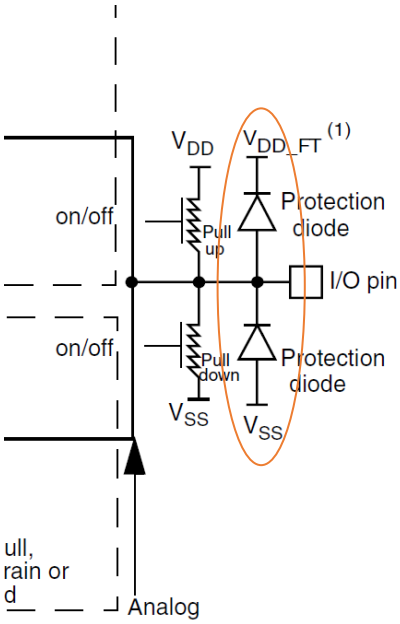
اگر افت ولتاژ LED برابر با ولتاژ منبع در یک مدار باشد، بدیهیست که نیازی به مقاومت محدود کننده نخواهیم داشت.

۳. کاتد مشترک به این معنی است که کاتد تمام LED های ۷ سگمنت مشترک است و به یک پین وصل است ولی هر آند LED پین مجزای خود را دارد و وقتی جریان از به هر یک از این پین های آند وارد میشود به سمت کاتد مشترک میرود و آن LED را روشن می کند.
آند مشترک دقیقا برعکس کاتد مشترک می باشد و آند ها مشترکند.
شکل زیر تصویر یک ۷ سگمنت کاتد مشترک است.



۴. همانطور که در رفرنس STM32F4 نوشته شده است بین پین ها و بدنه ی چیپ مقاومت های pull up و pull down قرار می دهیم و قبل از آنها دیود هایی برای محافظت میگذاریم تا از ریزپردازنده در برابر جریان کشی ها غیر قابل پیش بینی محافظت کنیم.

دیود های محافظت را به شکل روبه رو می بندیم:



این دیود ها تخلیه بار الکتریکی یا جریان کشی های ناخواسته را از ریزپردازنده دور می کند و به سمت مقاومت ها هدایت می کند.

نام دیگر این دیود ها clamping diode می باشد، چون ولتاژ ورودی را در بازه ی مشخصی جمع (clamp) می کند.

توضیح کد

در ابتدا، برای سادگی کار تمامی اعداد ۰ تا ۹ و حروف n و f را که به صورت هگز نیاز بود، نام گذاری کرده و در آرایه ی hex_codes قرار دادیم تا موقع شمارش، بتوان راحت تر به ان ها دسترسی داشت و پین ها را ست کرد.

```
//array containing all the hex codes of 7 seg
uint32_t hex_codes[] = {N0, N1, N2, N3, N4, N5, N6, N7, N8, N9};
```

تابع delay همانطور که در صورت پروژه امده بود، یک حلقه است که ۲۵۰۰۰ بار میچرخد و تاخیر ایجاد میکند.
استفاده از volatile جهت جلوگیری از حذف این حلقه توسط کامپایلر لازم بود. این حلقه تاخیری معادل نیم ثانیه ایجاد میکند.

```
static void delay( )
{
    volatile uint32_t c = 0;
    for( c=0; c<=DELAY_COUNT; c++ ){

        }
}
```

در تابع init مقدار دهی های اولیه انجام شده است. در ابتدا clock برای پورت ها روشن شده چون به طور دیفالت در حالت sleepهستند و حتما باید روشن شوند.

Mode برای پین های مورد نظر تنظیم شده است. از ۷ پین ابتدای A,B برای ست کردن 7segment ها استفاده شده است. و از پین ۹ پورت B برای یک led که برای تست است. (در ادامه توضیح داده خواهد شد.) همچنین از پین های پورت C برای ورودی گرفتن استفاده کردیم.

در اخر ست کردیم که در اغاز On روی 7seg ها نشان داده شود.

```
static void init( void )
{
    //Enable the AHB clock GPIO ports A,B,C
  SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOAEN);
  SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOBEN);
  SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOCEN);

    //set all Port A0..3,B0..3 as output - C0 input
  GPIOA->MODER = 0X55555555;
  GPIOB->MODER = 0X55555555;
  GPIOC->MODER = 0x05;

    //Show On at first
    GPIOA->ODR &= 0XFFFF0000;
    GPIOB->ODR &= 0XFFFF0000;
    GPIOA->ODR |= N0;
    GPIOB->ODR |= letter_N;

}
```

توابع checkBTN1 و checkBTN2 یک بولین برمیگردانند و چک میکنند که ایا ورودی مورد نظر فشرده شده یا خیر.

```
bool checkBTN1 ( void )
{
    if(GPIOC->IDR & MASK(0))
    {
        return true;
    }
    return false;
}
```

توابع check_time_BTN1 و check_time_BTN2 برای بررسی مدت زمانی است که دکمه فشرده میشود. اگر دکمه ۳ ثانیه فشرده شود یک led ابی رنگ روشن میشود ولی اگر زودتر از ۳ ثانیه دکمه رها شود، داخل شرط میرود و مقدار فلگ را false میکند.

Led ابی رنگ کمک میکند تا هنگام تست مدار متوجه شویم چه زمانی ۳ ثانیه شده و دکمه را دقیقا چه زمانی رها کنیم. (برای تست با کمک روش سرکشی)

```
bool check_time_BTN1 (void)
{
    bool flg = true;
    volatile uint32_t cn = 0;
    volatile uint32_t _3_sec = 6*DELAY_COUNT ; // 3 sec = 6 * 0.5 sec :))

  for( cn=0; cn<= _3_sec ; cn++ )
    {
        if( !(checkBTN1()) )
        {
            flg = false;
            break;
        }
    }
    if(flg)
    {
        GPIOB->BSRR = MASK(9);
        delay();
    }
    GPIOB->BSRR = MASK(25);
    return flg;
}
```

تابع wait_until_sth_pressed در ابتدا مقدار اخر شمارش شده را روی 7seg ست میکند و منتظر میماند تا کاربر دوباره یک دکمه را فشار دهد. هر بار که شمارنده cn به انتها میرسد صفر میشود لذا این حلقه تمام نخواهد شد مگر اینکه کاربر یک دکمه را فشار دهد.

```
//return true if btn1 pressed , false if btn 2
bool wait_until_sth_pressed (void)
{

    GPIOB->BSRR = 0xFFFF0000;
    GPIOA->BSRR = 0xFFFF0000;
    GPIOB->BSRR = hex_codes[j];
    GPIOA->BSRR = hex_codes[i];

    delay();

    volatile int cn = 0;
    for( cn=0; cn<=DELAY_COUNT; cn++ )
    {

        if( GPIOC->IDR & MASK(0) )
            return true;
        if( GPIOC->IDR & MASK(1) )
            return false;
        if(cn == DELAY_COUNT )
            cn=0;
    }
    return false; //never happen! just to make sure it will be compile.
}
```

توابع btn1_handeler و btn2_handeler رفتاری مشابه دارند اولی صعودی و دومی نزولی می شمارد.

شمارنده های i,j نیز گلوبال تعریف شده تا همواره در طول برنامه تغییر نکند. در ابتدای هر حلقه ابتدا دهگان ست میشود و سپس در حلقه ی داخلی یکان شمارش خواهد شد. از انجایی که خواسته شده بود بعد از ۹۹ ، مجدد به صفر برگردیم و برعکس، در اخرین خط کدها j,i دوباره تنظیم میشوند تا مشکلی در نمایش پیش نیاید.

بعد از هر تغییر یکان هم بررسی میشود ایا کلید دیگری فشار داده شده است یا خیر. اگر هم هر کلید فشار داده شده بود با کمک توابعی که در قبل توضیح دادیم بررسی میشود چه مدت فشار داده شده اند.

```
//do incremental counting
for(; i < 10; i++)
{
    GPIOA->BSRR = 0xFFFF0000;
    GPIOA->BSRR = hex_codes[i];
    for(; j < 10; j++)
    {
    GPIOB->BSRR = 0xFFFF0000;
    GPIOB->BSRR = hex_codes[j];

    // if btn 2 pressed
    if (checkBTN2())
    {
        bool chck = check_time_BTN2();
        //if pressed 3sec
        if(chck)
            {
                GPIOA->BSRR = 0xFFFF0000;
                GPIOA->BSRR = hex_codes[0];
                GPIOB->BSRR = 0xFFFF0000;
                GPIOB->BSRR = letter_F;
                while(1);
            }
        else
                btn2_handeler();
    }
```

```
        //if pressed 3sec
        if(check_time_BTN1())
            {
                bool flg = wait_until_sth_pressed();
                if(flg)
                    btn1_handeler();
                else
                    btn2_handeler();
            }

    delay();
}
    j=0;
    if( i == 9)
    {
        i=-1;
    }
}
```

این قسمت از کد یک حلقه بی نهایت دارد که هرگز تمام نخواهد شد مگر آنکه سیستم باز نشانی شود. همچنین 7seg ها OF را نشان میدهند.

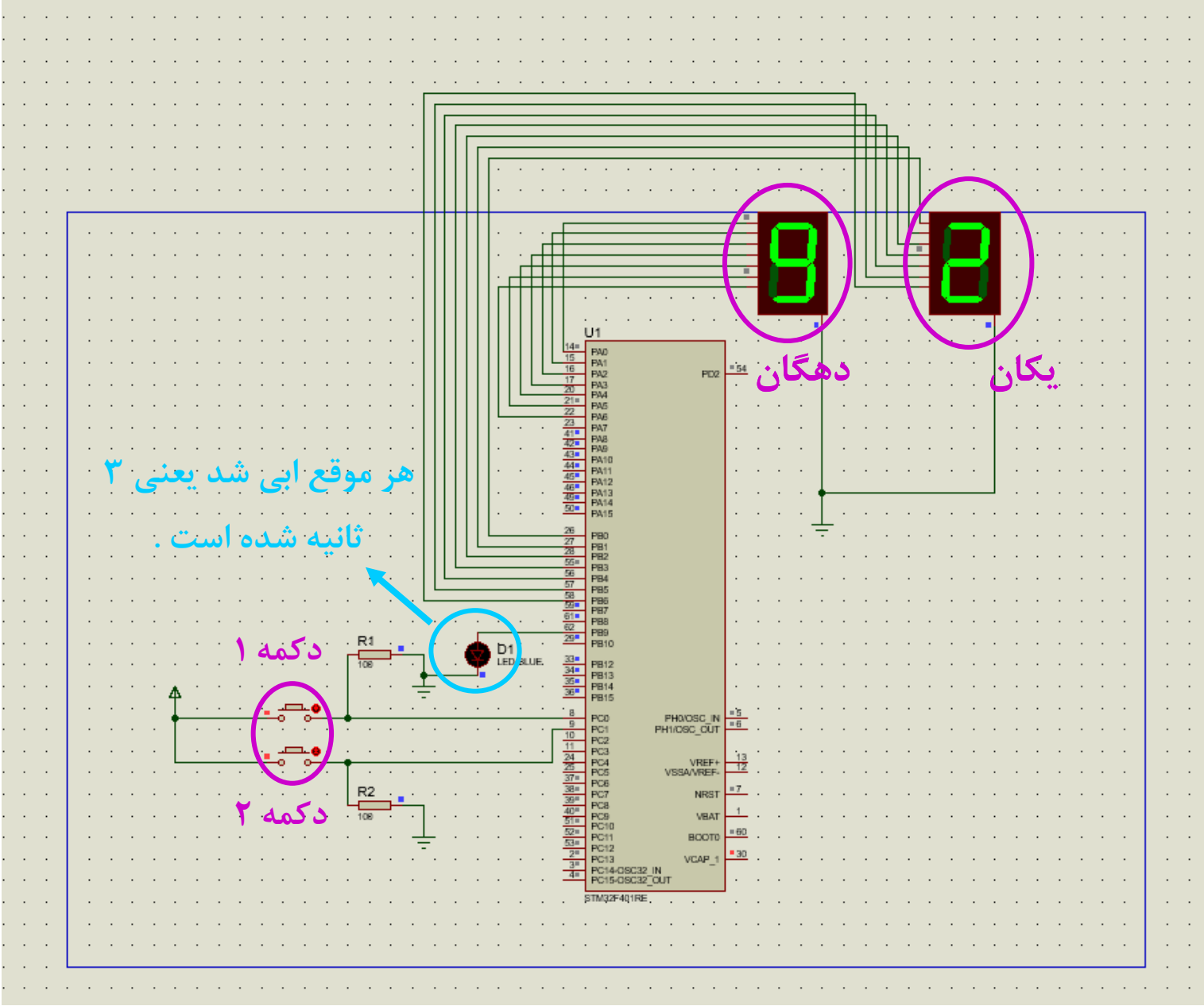
```
    //if pressed 3sec
    if(check_time_BTN2())
        {
            GPIOA->BSRR = 0xFFFF0000;
            GPIOA->BSRR = hex_codes[0];
            GPIOB->BSRR = 0xFFFF0000;
            GPIOB->BSRR = letter_F;
            while(1);
        }
```

تابع start_app در شروع برنامه، مشخص میکند ابتدا کدام کلید فشرده شده است و کار را ادامه میدهد.

```
void start_app(void)
{
    if(checkBTN1())
        btn1_handeler();
    if(checkBTN2())
        btn2_handeler();

}
```

توضیح مدار


U1
دهگان
یکان
هر موقع ابی شد یعنی ۳
ثانیه شده است .
دکمه ۱
دکمه ۲
R1
R2
D1
LED-BLUE
STM32F401RE