دستورکار سری سوم آزمایشگاه ریزپردازنده و زبان اسمبلی

امیرحسین ادواری ۹۸۲٤۳۰۰٤ – زهرا حیدری ۹۸۲٤۳۰۲۰

بخش تحلیلی

-1

تکنولوژی CMSIS (cortex microcontroller software interface standard)

یک کتابخانه هست که به برنامه نویسان در programming یک پردازنده Cortex-M کمک میکند .

از زمان ساخت میکروکنترلر ها , با پیچیده تر شدن Microcontroller Unit peripherals یا همان واحدهای میکروکنترلرها , وجود errors و bugs در کد بسیار زیاد بوده .

بعد ها میکروکنترلر های 32 بیتی وارد بازار شد و configure کردن control registers بسیار دشوار شد . برای همین کمپانی Keil تکنولوژی CMSIS را اختراع کرد که یک Configuration Interface بسیار ساده بین هسته میکروکنترلر و واحدهای peripheral به وجود آورد . که این باعث افزایش سرعت configure کردن peripheral ها شد و باعث کاهش Errors و bugs ها شد زیرا استفاده کردن از کتابخانه ها و کدهای آماده باعث کاهش احتمال خطا می شود .

همچنین به ساده سازی کد کمک میکند و کد برای ما قابل فهم تر میشود . هزینه های development کاهش میدهد . برنامه نویسان میتوانند راحت تر و سریع تر با interface های نرم افزاری استاندارد سازی شده , نرم افزار بنویسند .

این کتابخانه همچنین portability و Re-usability را افزایش میدهد یعنی می توان نرم افزار را راحت تر تغییر یا به روزرسانی کرد .

کاربرد CMSIS-DSP : برای DEVELOP کردن یک سیستم Digital Signal Processing DSP یا پردازش سیگنال دیجیتال به صورت Real-Time است یعنی این پردازش باید در زمان مشخصی اتفاق بیفتد .

این کتابخانه دارای تعداد زیادی فانکشن DSP است که برای پردازنده های Cortex-M نوشته شده است .

این کتابخانه در صنعت کاربرد زیادی دارد و برای Matlab کدهای زبان C را Optimize کرده است .

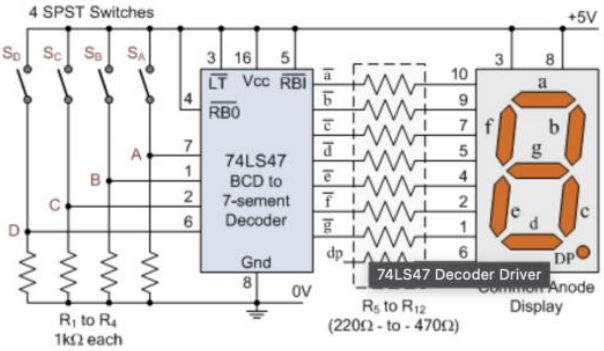
-2

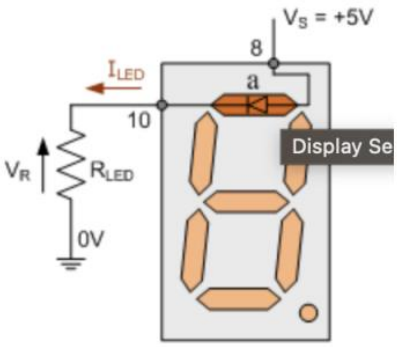
خیر کار صحیحی نمی باشد .

ساختار seven segment مجموعه ای از دیود های نوری است که به وسیله جریان الکتریکی روشن می شوند .

بنابراین بهتر است که برای محدود کردن جریان از مقاومت استفاده کنیم. بهترین روش هم اتصال این مقاومت ها به صورت سری به پایه ها است زیرا اکر این کار را نکنیم جریان maximum وارد LED می شود و برای مدت کوتاهی بسیار نورانی خواهد بود و بعد کاملا نابود می شود .

در شکل زیر نحوه ی صحیح اتصال seven segment به میکروکنترلر نشان داده شده است .





-3

ساختار LED یا light Emitting diode یک دیود نوری است که فوتون تولید می کند . جنس و متریال به کار رفته در LED , طول موجی که فوتون تولید می کند را تعیین می کند . متریال های مختلف , طول موج های مختلف را تولید می کنند که باعث تولید رنگ های مختلف می شوند .

در LED هایی که تغییر رنگ می دهند در واقع چندین LED با متریال های مختلف وجود دارد که با عبور جریان از هر کدام از این بخش ها یک رنگ تولید می شود و با عبور جریان هم زمان از چندتای آنها رنگ های جدید ایجاد می شود .

تعویض LED در مدار خیلی راحت نیست چون هر LED احتیاج به یک ولتاژ معین برای روشن شدن دارد. درواقع Voltage Forward ولتاژی است که در زمان عبور جریان توسط LED استفاده میشود .این درجه یا Rating به رنگ LED بستگی دارد چون از متریالهای مختلف برای هرکدام استفاده شده است .

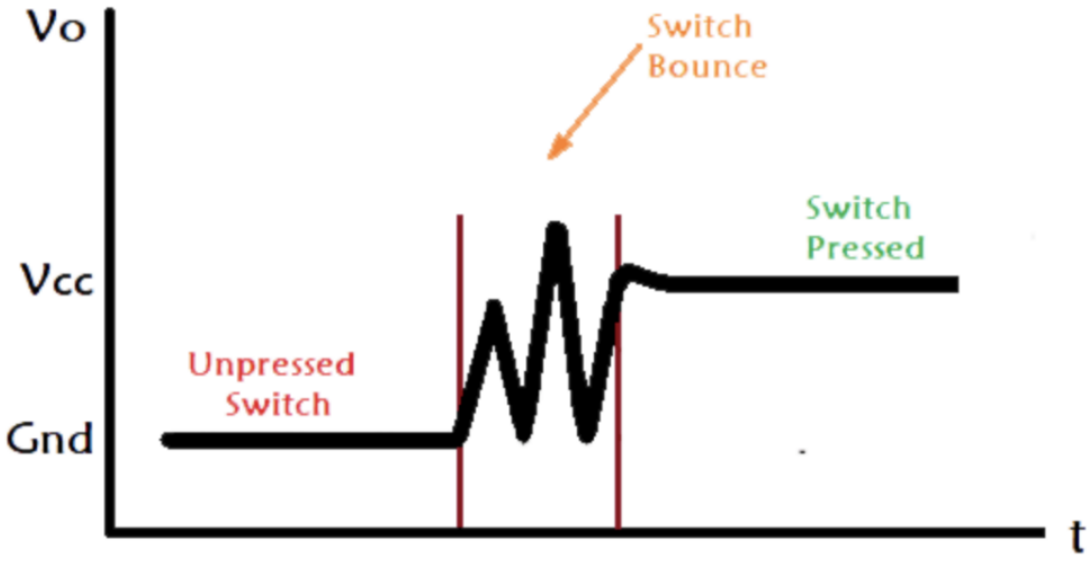
شکل زیر جدول Voltage Forward برای هر رنگ به همراه متریال و طول موج هرکدام است.

LED COLORS AND MATERIALS

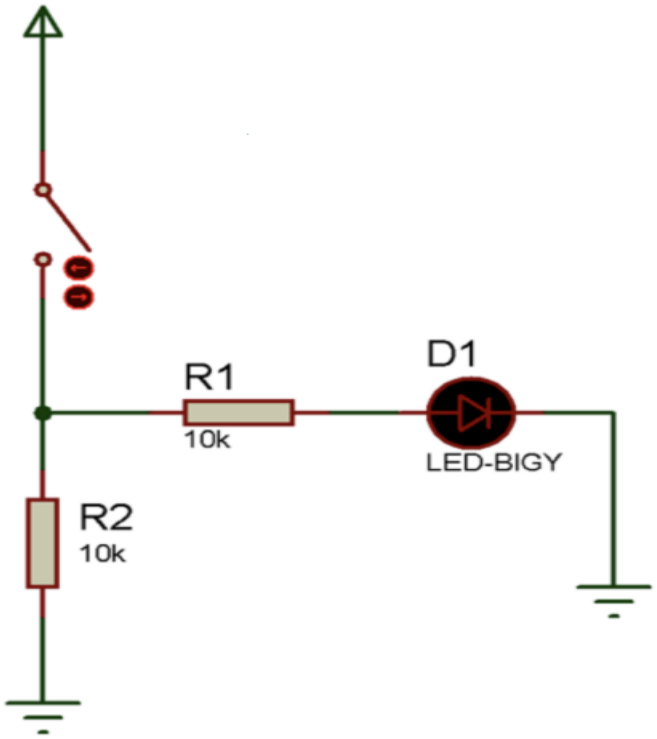
Color	Wavelength Range (nm)	Forward Voltage (V)	Material
Ultraviolet	< 400	3.1 - 4.4	Aluminium nitride (AlN) Aluminium gallium nitride (AlGaN) Aluminium gallium indium nitride (AlGaInN)
Violet	400 - 450	2.8 - 4.0	Indium gallium nitride (InGaN)
Blue	450 - 500	2.5 - 3.7	Indium gallium nitride (InGaN) Silicon carbide (SiC)
Green	500 - 570	1.9 - 4.0	Gallium phosphide (GaP) Aluminium gallium indium phosphide (AlGaInP) Aluminium gallium phosphide (AlGaP)
Yellow	570 - 590	2.1 - 2.2	Gallium arsenide phosphide (GaAsP) Aluminium gallium indium phosphide (AlGaInP) Gallium phosphide (GaP)
Orange / Amber	590 - 610	2.0 - 2.1	Gallium arsenide phosphide (GaAsP) Aluminium gallium indium phosphide (AlGaUInP) Gallium phosphide (GaP)
Red	610 - 760	1.6 - 2.0	Aluminium gallium arsenide (AlGaAs) Gallium arsenide phosphide (GaAsP) Aluminium gallium indium phosp Gallium phosphide (GaP)
Infrared	> 760	> 1.9	Gallium arsenide (GaAs) Aluminium gallium arsenide (AlGaAs)

LED Color and

زمانی که ما یک دکمه را فشار میدهیم، دو قطعه فلزی با هم تماس پیدا میکنند تا جریان منتقل شود اما آنها در همان لحظه به صورت آنی متصل نمیشوند بلکه قطعات فلزی چندین بار Connect و Disconnect میشوند تا اینکه یک اتصال Stable ایجاد شود .همین اتفاق در زمان رها کردن دکمه هم اتفاق میفتد .این باعث Triggering Falseیا Triggering Multiple میشود یعنی انگاردکمه را چند بار فشار دادیم .



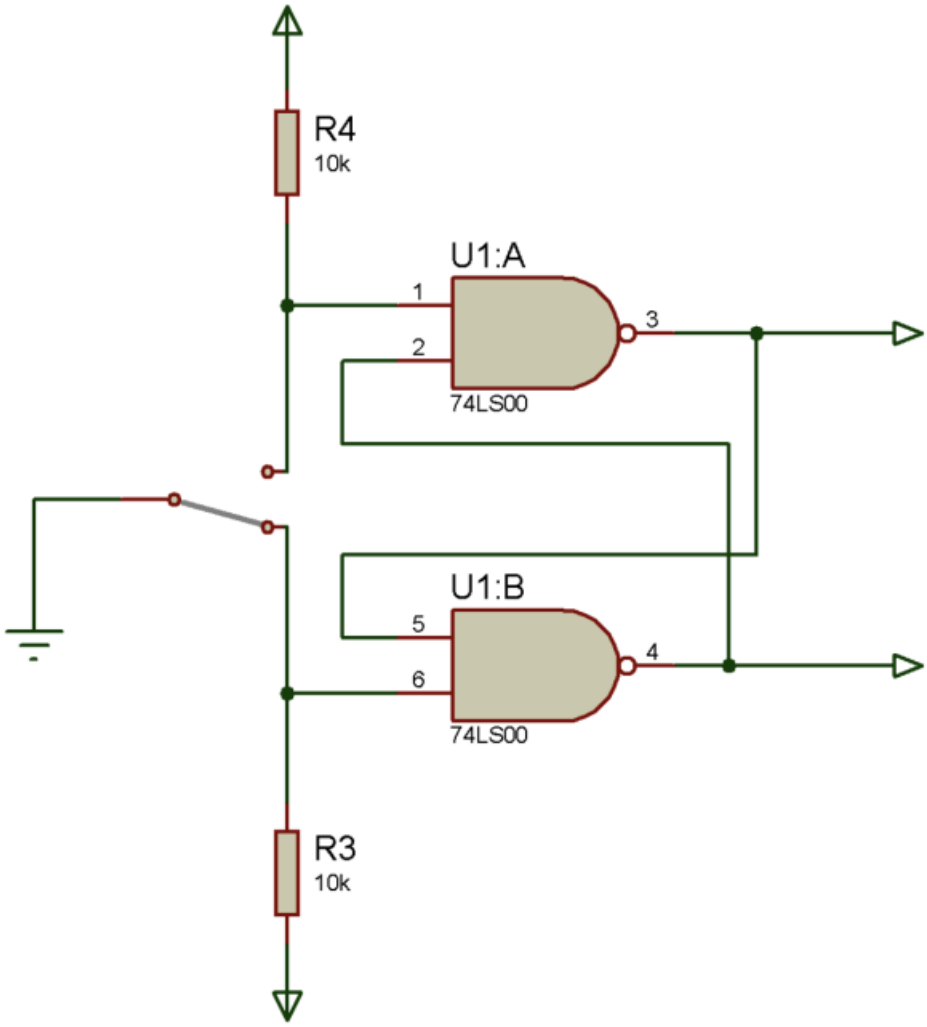
این اتفاق باعث تولید چند Transition برای یک ورودی میشود .برای مدارهای الکتریکی مشکل زیادی تولید نمیکند اما برای مدارهای منطقی و دیجیتال مشکلساز است .پس باید با Circuit Debouncing Switchحل شود .شکل روبهرو مدار عادی است که در آن Bounce Switch رخ میدهد.



راهکار سخت افزاری :

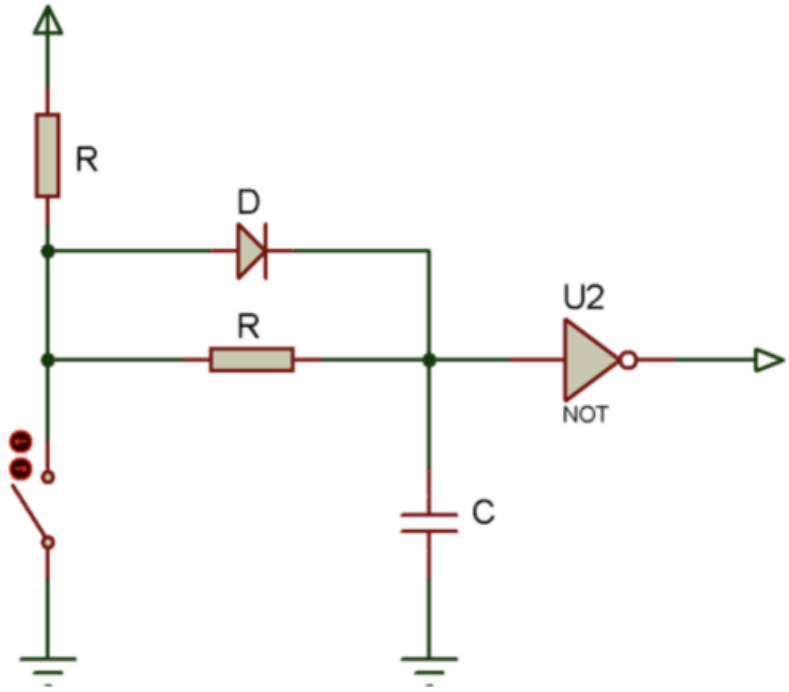
1 -اضافه کردن Flip Flop یا Hardware Debouncing :

در این راهکار از یک Flop-Flip-SR استفاده میکنیم که از دو گیت NAND تشکیل شده .مطابق شکل در صورتی که Toggle به سمت A برود، خروجی High یا یک میشود. وقتی Switch بین روشن و خاموش حرکت میکند، Flop Flipخروجی را حفظ میکند چون ۰ از خروجی گیتهای NAND میآید.



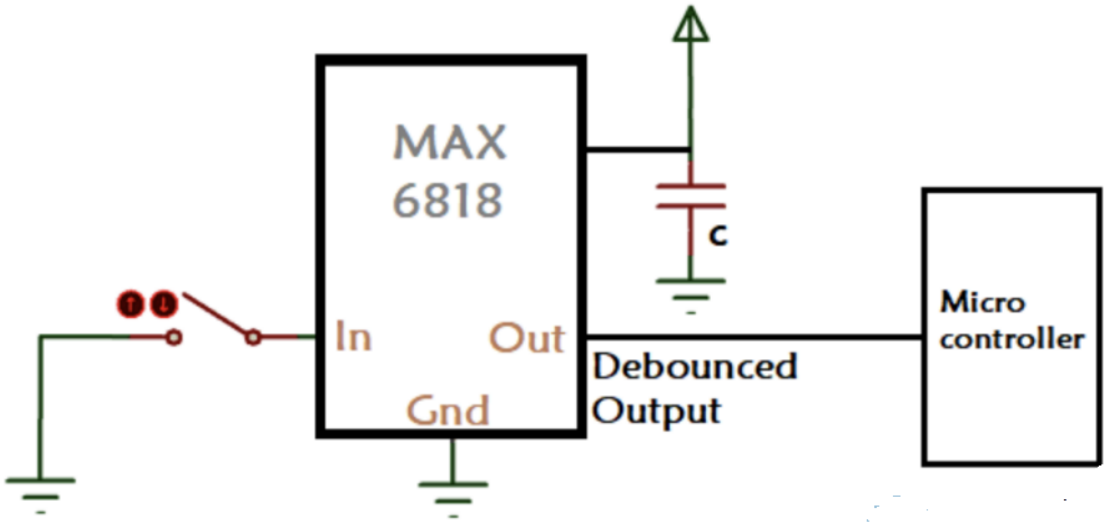
2- اضافه کردن RC یا RC Debouncing :

در این راهکار یک خازن تغییرات دکمه را فیلتر میکند پس وقتی Switch باز است، ولتاژ دو سر خازن صفر است و خازن شارژ میشود. با بستن کلید، خازن Dischargeمیشود و خراجی یک میشود .زمانی که Bounceاتفاق میافتد، خازن ولتاژ را در Vin نگه میدارد تا به Vccیا زمین برسد.



3- استفاده از IC های مخصوص یا Switch Debouncing :

در بازار IC هایی وجود دارند جه به منظور کنترل Bouncing Switch طراحی شده اند.



راهکار نرم افزاری :

در زمان برنامه نویسی باید یک Delay به کد اضافه کنیم که باعث میشود کنترلر برای مدتی استپ کند . اما این راهکار خیلی خوب نیست چون باعث استپ شدن برنامه میشود و زمان پردازش زیاد میشود .پس میتوان به جای Delay از Interrupt استفاده کنیم

منابع:

https://www.keil.com https://developer.arm.com
https://armsoftware.github.io https://microcontrollerslab.com
https://circuitdigest.com https://content.ccontrols.net/blog/cmsis
https://developer.arm.com/tools-and-software/embedded /cmsis

گزارش بخش عملی

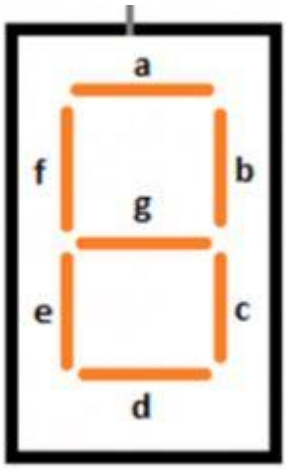
برای پیاده‌سازی عملکرد تصریح شده در صورت پروژه، از دو دکمه، یک ریزپردازنده، یک سون‌سگمنت دوتایی و دو led سبز و قرمز و نیز چند مقاومت استفاده شده است.

بدین منظور دو پین ورودی (برای دکمه‌ها) و نه پین خروجی (هفت پین برای سگمنت و دو پین برای ledها) نیاز داریم. همه آن‌ها را در GPIOA قرار می‌دهیم. پیکربندی به صورت زیر است (پین‌های قرمزرنگ خروجی و پین‌های آبی رنگ ورودی‌اند):

PORT	PA10	PA9	PA8	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
USAGE	G	F	E	D	C	B	A	الای‌دی سبزرنگ	الای‌دی قرمزرنگ	کلیددوم	کلیداول

مطابق جدول بالا پین صفر و یک را برای ورودی و پین‌های دو تا ده را برای خروجی‌هایمان درنظر میگیریم.

مطابق شکل زیر هر کدام از حروف A تا G مربوط به یک پاره خط کوچک در سون‌سگمنت است. مشخصا برای نمایش اعداد مختلف بایستی ترکیب خاصی از این سیگنال‌ها فعال شوند.



جزییات کد

- در ابتدای برنامه، مشابه جدولی که پیش‌تر آورده شد، پین‌های ادوات مختلف شرح‌داده شده را تعریف می‌کنیم:

main.c

```
1  #include <stm32f4xx.h>
2  #include <stdbool.h>
3  #define B1 (0)      // input
4  #define B2 (1)      // input
5  #define RED (2)     // output
6  #define GREEN (3)   // output
7  #define A (4)       // output
8  #define B (5)       // output
9  #define C (6)       // output
10 #define D (7)       // output
11 #define E (8)       // output
12 #define F (9)       // output
13 #define G (10)      // output
14 #define MASK(x) (1UL << (x))
```

یک ماکرو برای MASK کردن نیز تعریف می‌کنیم که عملکرد آن واضح است.

- سپس متغیرهای مورد استفاده در برنامه را تعریف می‌کنیم:

```
16 volatile int8_t counter = 9;
17 volatile bool is_red_active = 1;
18 volatile uint8_t b1_clicks = 0;
19 volatile uint8_t b2_clicks = 0;
20
```

مطابق تصویر یک عدد هشت بیتی به‌عنوان شمارشگر تایمر، یک متغیر منطقی برای مشخص کردن LED فعال، و دو عدد هشت بیتی نیز برای ذخیره کردن تعداد دفعات فشردن دکمه‌ها تعریف می‌کنیم. چون تایمر از نه شروع می‌شود مقدار اولیه آن نه می‌شود. از آنجا که ابتدا با LED قرمز کارکرد مدار آغاز می‌شود، به متغیر بولین یک می‌دهیم. تعداد دفعات فشردن دکمه‌ها نیز در ابتدای برنامه واضحا صفر است.

- یک تابع برای چک کردن فشردن دکمه‌ها ایجاد می‌کنیم (تابع **check_clicks**):

```
21 void check_clikcs(void){
22     if(GPIOA->IDR & MASK(B1) ){
23         b1_clicks++;
24 
25       //Stay here until the user releases the button:
26         while(GPIOA->IDR & MASK(B1)){}
27     }
28     if(GPIOA->IDR & MASK(B2) ) {
29         b2_clicks++;
30 
31         //Stay here until the user releases the button:
32         while(GPIOA->IDR & MASK(B2)){}
33 
34     }
35 }
```

اگر در GPIOA بیت مربوط به دکمه اول یک شود، متغیر مربوطه را increment می‌کنیم و مادامیکه کاربر دکمه را رها نکرده، در یک لوپ قرار می‌گیریم. مشابه همین روال برای دکمه دوم برقرار است.

- تابعی نیز برای ایجاد تاخیر تعریف کرده‌ایم (تابع delay) :

```
36 void delay(uint8_t halfsec){
37   for(int i = 0; i < halfsec; i++){
38     for(int j = 0; j < 400000; j++) {
39 
40       // check push-buttons :
41       check_clikcs();
42     }
43   }
44 }
```

در این تابع یک حلقه طولانی داریم که در آن صرفا فشردن دکمه‌ها چک می‌شود. وروی این تابع تعداد نیم‌ثانیه‌های تاخیر مورد نیاز است.

- درنهایت یک تابع برای تبدیل اعداد صفر تا نه به یک رشته ۱۶ بیتی می‌نویسیم(تابع **segment**):

این تابع یک عدد به‌عنوان ورودی می‌گیرد و یک رشته ۱۶بیتی برمی‌گرداند به‌طوریکه در آن بیت-های متناظر با کاراکترهایی که برای نشان دادن آن عدد روی سون‌سگمنت لازم است یک شده و سایر سیگنال‌ها صفر شده باشند.

```
// Convert integer to its correspondig 7-segment  format
uint16_t segment(uint16_t value) {
  switch(value){
    case 0:
      return MASK(A) | MASK(B) | MASK(C) | MASK(D) | MASK(E) | MASK(F);
    case 1:
      return MASK(B) | MASK(C);
    case 2:
      return MASK(A) | MASK(B) | MASK(D) | MASK(E) | MASK(G);
    case 3:
      return MASK(A) | MASK(B) | MASK(C) | MASK(D) | MASK(G);
    case 4:
      return MASK(B) | MASK(C) | MASK(F) | MASK(G);
    case 5:
      return MASK(A) | MASK(C) | MASK(D) | MASK(F) | MASK(G);
    case 6:
      return MASK(A) | MASK(C) | MASK(D) | MASK(E) | MASK(F) | MASK(G);
    case 8:
      return MASK(A) | MASK(B) | MASK(C) | MASK(D) | MASK(E) | MASK(F) | MASK(G);
    case 7:
      return MASK(A) | MASK(B) | MASK(C);
    case 9:
      return MASK(A) | MASK(B) | MASK(C) | MASK(D) | MASK(F) | MASK(G);
    default:
      return MASK(A) | MASK(B) | MASK(C) | MASK(D) | MASK(E) | MASK(F);
  }
}
```

به‌طور مثال برای نشان دادن عدد هفت، خروجی متناظر با A، B و C بایستی یک شده و سایر بیت‌ها صفر شوند (اینکه هرکدام از این حروف خروجی کدام پین GPIOA هستند در ابتدای این بخش ذیل یک جدول آورده شده‌است)

- تابع **main**:

مطابق شکل زیر ابتدا GPIOA را فعال می‌کنیم (میدانیم به‌طور پیشفرض برای ایجاد بهینگی در توان مصرفی کلاک GPIOها قطع است لذا آن‌را فعال می‌کنیم)

```
int main(void){

  // Enable GPIOA:
  RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
```

برای ست کردنMODER (که تعیین می‌کند هر پین از چه‌نوعی می‌باشد (ورودی یا خروجی)) می‌دانیم که هر دوبیت متوالی در این رجیستر مربوط به مدکاری یک پین GPIO می‌باشد (۰۰ برای ورودی و ۱۰ برای خروجی) از این‌رو مقدار MODER بایستی به صورت زیر باشد(صرفا ۲۲ بیت پایین برایمان اهمیت دارد زیرا عملا یازده پین مورد استفاده است، سایر بیت‌ها را یک میکنیم):

TYPE	OUT		OUT		OUT		OUT		OUT		OUT		OUT		OUT		IN		IN	
Binary	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0
Hex	D		5				5				5				5				0			

طبق جدول بالا و توضیحات ذکر شده MODER را به شکل زیر مقداردهی می‌کنیم:

```
79  // Set Input & Outputs as decared at the beginning  (00 for input and 10 for output)
80  GPIOA->MODER = 0xFFD55550;
```

سپس تعیین می‌کنیم که pull-down برای پین‌های ورودی فعال شوند (10 برای pull-down می‌باشد از طرفی دو پین اول و دوم GPIOA ورودی می‌باشد) :

```
82  // Pull-down for first and second input (B1 & B2)
83  GPIOA->PUPDR &= ~MASK(0);
84  GPIOA->PUPDR |= MASK(1);
85  GPIOA->PUPDR &= ~MASK(2);
86  GPIOA->PUPDR |= MASK(3);
```

در نهایت به لوپ اصلی برنامه می‌رسیم:

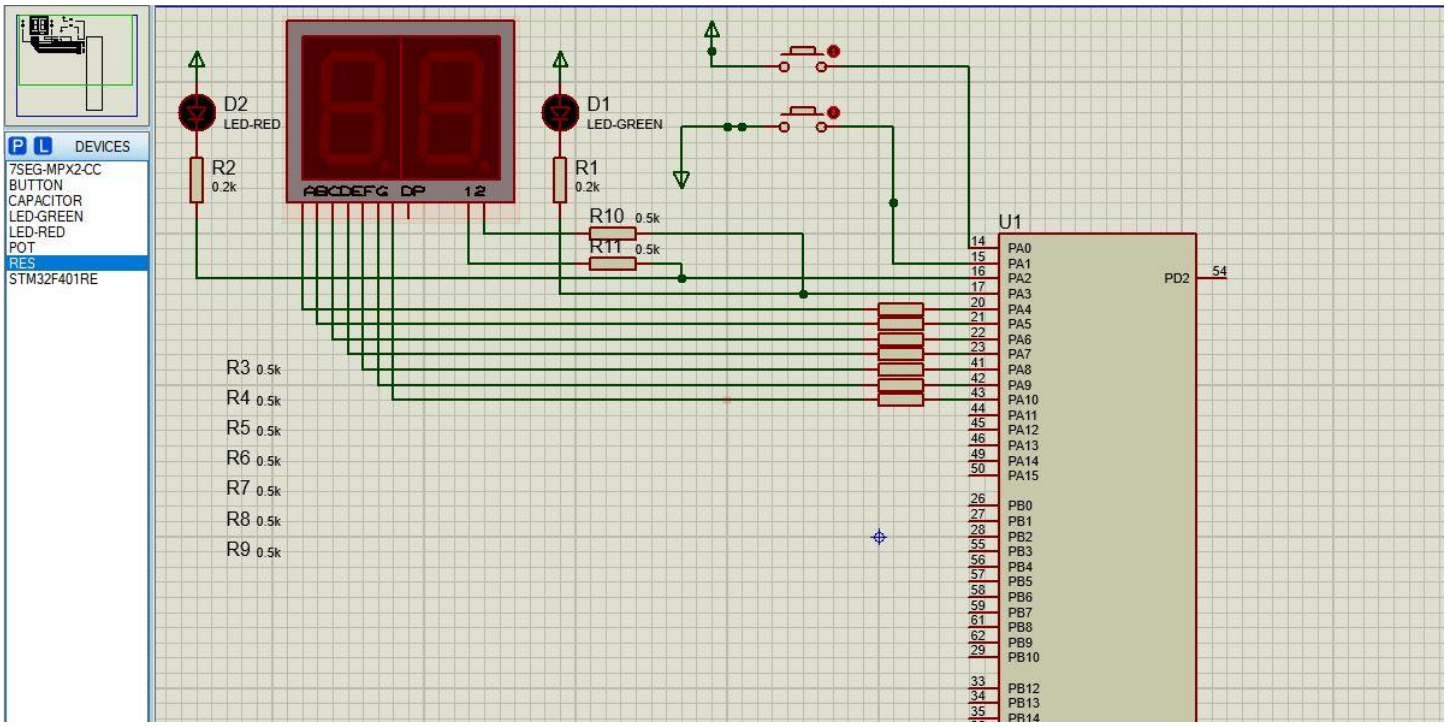
```
1     while(1){
2
3           // Check wether user has pushed buttons:
4           check_clicks();
5
6           // If b1 has been pushed two times, change the timer and led side.
7           if( b1_clicks == 2){
8                 b1_clicks = 0;
9                 b2_clicks = 0;
10                counter = 9;
11                is_red_active = !is_red_active;
12          }
13
14          // If b2 has been pushed for 4th time, count !
15          if( b2_clicks == 4){
16                b2_clicks = b1_clicks =  0;
17          }
18
19 //if b2 has been pushed 3 times, stop the timer else set proper outputs & wait 1s
20          if( b2_clicks != 3){
21
22 // if active LED is red, activate red and deactivate green and show counter on
23 //segments:
24                if( is_red_active ) {
25                      GPIOA->ODR &= 0x0;
26                      GPIOA->ODR |= MASK(GREEN) | segment(counter);
27                }
28 // if active LED is green, activate green and deactivate red and show counter on
29 //segments:
30                else {
31                      GPIOA->ODR &= 0x0;
32                      GPIOA->ODR |= MASK(RED) | segment(counter);
33                }
34                //decrement counter :
35                counter--;
36 // if counter reached -1, reset counter, change led and 7segment's side :
37                if(counter == -1) {
38                      counter = 9;
39                      is_red_active = !is_red_active;
40                }
41                delay(2);
42          }
43    }
```

- در ابتدای کد فشردن دکمه‌ها را توسط تابع check_clicks() که پیش‌تر شرح داده شد، بررسی می-کنیم(خط چهارم تکه کد)

- سپس بررسی می‌کنیم که اگر تعداد فشردن دکمه اول دوبار شده‌است، تایمرها و شمارنده‌های دفعات فشردن دکمه را ریست کرده و LED فعال را تغییر می‌دهیم (خطوط هفتم تا سیزدهم)
- سپس اگر کلیددوم برای بارچهارم فشرده‌شده باشد، تایمر بایستی کارکند و شمارنده‌های دکمه‌ها بایستی ریست شوند. (خطوط ۱۵ تا ۱۷)
- درنهایت اگر تعداد دفعات فشردن کلید دوم برابر سه (که به معنی توقف تایمر است) نباشد، بررسی می‌کنیم که کدام تایمر در حال حاضر فعال است، سپس متناسب با آن سیگنال مربوط به تایمر مربوطه را LOW می‌کنیم و جهت مخالف آنرا HIGH می‌کنیم و سگمنت‌هایی که بایستی روشن شوند را نیز از طریق تابع segment مشخص می‌کنیم و یک می‌کنیم. (در سون‌سگمنتی که استفاده کردیم سیگنال‌های تعیین‌کننده روشن بودن تایمر(منظور پایه‌های 12 روی ic مربوطه می‌باشد) ACTIVE-LOW هستند) سپس بایستی تایمرفعال فعلی decrement شده و در صورت صفر شدن، ریست شده و طرف مخالف فعال شود. درنهایت حدود یک ثانیه تاخیر ایجاد می‌کنیم (تابع مربوطه شرح داده شد) (خطوط ۲۰ تا آخر)

پیاده‌سازی روی پروتئوس

مدار را مطابق تصویر زیر پیکربندی می‌کنیم :



پین‌های مشخص شده در قسمت قبلی را به علاوه یک مقاومت به پایه‌های متناسب متصل می‌کنیم.(توجه شود که دکمه‌هارا به Vdd متصل کردیم و پین‌های PA0 و PA1 را روی pull-down تنظیم کردیم) و مدار را تست می‌کنیم. مشاهده می‌شود که مدار به درستی کار می‌کند.

تصاویری از اجرا