فاطمه سادات سیفی – 98243035

پارسا محمدپور – 98243050

سوالات تحلیلی:

۱) مزایای بهکارگیری کتابخانه های استاندارد CMSIS را شرح دهید. کاربرد CMSIS-DSP چیست؟

- CMSISیک استاندارد که توسط ARM ارائه شده و برنامه نویسی برای آنرا مقداری راحت تر و استاندارد تر میکند .رابط های نرمافزار CMSIS استفاده مجدد از نرم افزار را ساده می کند، و یادگیری را برای توسع هدهندگان میکروکنترلر سرعت میبخشد وباعث افزایش سرعت ساخت و اشکال زدایی پروژه و در نتیجه کاهش زمان عرضه به بازار برنامه های جدید میشود. CMSIS رابط هایی را برای پردازنده و ادوات جانبی، سیستم عامل های بلادرنگ (real time) و اجزای middleware فراهم می کند.CMSIS شامل یک مکانیسم تحویل برای دستگاه ها، بردها و نرم افزار است و ترکیب اجزای نرم افزار از چندین vendor را امکان پذیر میکند .و همچنین Open sourceاست.

-کتابخانه نرم افزاری CMSIS DSP مجموعه ای از توابع پردازش سیگنال رایج است که برای میکروکنترلرهای مبتنی بر پردازنده Cortex-M مورد استفاده قرار میگیرد .حتی اگر کد به طور خاص برای استفاده از مجموعه دستورالعمل DSP توسعه یافته پردازنده Cortex-M4 بهینه شده باشد، کتابخانه برای همه ی پردازنده های Cortex-M کامپایل میشود .همچنین توابع مجزایی برای کار بر روی اعداد صحیح 8 بیتی، اعداد صحیح 16 بیتی، اعداد صحیح 32 بیتی و مقادیر ممیز شناور 32 بیتی شامل میشود.

۲) آیا درایو مستقیم سون سگمنت (اتصال مستقیم آن به پایههای میکروکنترلر)کار صحیحی است؟ چنانچه پاسخ شما به این پرسش «مثبت» است دلایل خود را بیان کنید و در صورت پاسخ «منفی» راهکار جایگزین را شرح دهید.

درایو مستقیم سون سگمنت امکان پذیر است ولی برای جلوگیری از آسیب دیدن به دلیل جریان های غیر قابل پیش بینی توصیه نمیشود(مگر در شرایط خاص)و همچنین استفاده از آی سی های راه انداز مبدل BCDبه سون سگمنت کار را ساده تر میکند؛ تعداد خطوط برنامه نویسی را کاهش میدهد؛ تعداد پایه های مورد نیاز در میکرو را کاهش میدهد و نیاز به کد کردن به صورت دستی را رفع می نماید.
از طرف دیگر به لحاظ تجربی در نهایت اگر ولتاژ سون سگمنت زیر 5 ولت باشد به راحتی می توان صرفا با اتصال یک مقاومت مناسب بین پایه های میکرو و سون سگمنت و یا دیود های محافظت کننده که جریان های آسیب زننده را هدایت می کنند آنرا راه اندازی کرد.

Enhanced software reusability

- easier to reuse software code in different Cortex-M projects

Enhanced software compatibility

- by having a consistent software infrastructure

Easy to learn

- easy access to processor core features from the C language

Toolchain independent

- can be used with various compilation tools

Openness

- the source code for CMSIS core files can be downloaded from its public github repository

۳) چه تفاوتی در ساختار LED ها با رنگهای متفاوت وجود دارد؟ آیا میتوان در هر مداری آنها را جایگزین یکدیگر نمود شرح دهید.

LEDهای مختلف از مواد مختلف ساخته شده اند واین بر رفتار آنها تاثیر میگذارد.
روشنایی یک LED مستقیما به میزان جریان آن بستگی دارد .این به معنی دو چیز است .اولین مورد این است که LED های فوق العاده روشن باتری را با سرعت بیشتری تخلیه می کنند، زیرا برای روشنایی بیشتر از انرژی اضافی استفاده می شود. مورد دوم این است که شما می توانید با کنترل میزان جریان ، روشنایی یک LED را کنترل کنید.
اگر یک LED را مستقیم به منبع تغذیه متصل کنید، تا حدی توان الکتریکی را کنترل میکند و در صورت دریافت ولتاژ بالا، توان الکتریکی بالا میرود و در نتیجه LED میسوزد و از بین میرود.
طول موج های کوتاه تر نور از فوتون های انرژی بالاتر ساخته می شوند .رنگ قرمز دارای طول موج بیشتر / انرژی کمتر از زرد نسبت به سبز از آبی است که طول موج کوتاه تر / انرژی بالاتر است.
طول موج نور و در نتیجه رنگ به نوع ماده نیمه هادی استفاده شده برای ساخت دیود بستگی دارد. دلیل آن این است که ساختار باند انرژی نیمه هادی ها بین مواد متفاوت است، بنابراین فوتون ها با فرکانس های مختلف ساطع می شوند .جدول زیر نیمه هادی متداول برای ساخت ال ای دی بر اساس فرکانس آورده شده است:

	Color	Wavelength [nm]	Semiconductor material
	Infrared	$\lambda > 760$	Gallium arsenide (GaAs) Aluminium gallium arsenide (AlGaAs)
	Red	$610 < \lambda < 760$	Aluminium gallium arsenide (AlGaAs) Gallium arsenide phosphide (GaAsP) Aluminium gallium indium phosphide (AlGaInP) Gallium(III) phosphide (GaP)
	Orange	$590 < \lambda < 610$	Gallium arsenide phosphide (GaAsP) Aluminium gallium indium phosphide (AlGaInP) Gallium(III) phosphide (GaP)
	Yellow	$570 < \lambda < 590$	Gallium arsenide phosphide (GaAsP) Aluminium gallium indium phosphide (AlGaInP) Gallium(III) phosphide (GaP)
	Green	$500 < \lambda < 570$	**Traditional green:** Gallium(III) phosphide (GaP) Aluminium gallium indium phosphide (AlGaInP) Aluminium gallium phosphide (AlGaP) **Pure green:** Indium gallium nitride (InGaN) / Gallium(III) nitride (GaN)
	Blue	$450 < \lambda < 500$	Zinc selenide (ZnSe) Indium gallium nitride (InGaN) Silicon carbide (SiC) as substrate Silicon (Si) as substrate—under development
	Violet	$400 < \lambda < 450$	Indium gallium nitride (InGaN)
	Purple	multiple types	Dual blue/red LEDs, blue with red phosphor, or white with purple plastic
	Ultraviolet	$\lambda < 400$	Diamond (235 nm) Boron nitride (215 nm) Aluminium nitride (AlN) (210 nm) Aluminium gallium nitride (AlGaN) Aluminium gallium indium nitride (AlGaInN)—down to 210 nm
	Pink	multiple types	Blue with one or two phosphor layers: yellow with red, orange or pink phosphor added afterwards, or white with pink pigment or dye.
	White	Broad spectrum	Blue/UV diode with yellow phosphor

۴) مفهوم Switch Bouncing را توضیح دهید. راهکارهای نرم افزاری و سخت افزاری قابل به کارگیری برای حل این مشکل را مختصرا توضیح دهید.

وقتی دکمه فشاری یا سوئیچ ضامن یا میکرو سوئیچ را فشار می دهیم، دو قسمت فلزی با هم تماس پیدا می کنند تا منبع تغذیه را کوتاه کنند .اما آنها فوراً به هم وصل نمی شوند بلکه قطعات فلزی چندین بار قبل از برقراری اتصال پایدار واقعی متصل و جدا می شوند .هنگام رها کردن دکمه نیز همین اتفاق می افتد. این باعث میشود که راه اندازی کاذب یا راه اندازی چندگانه مانند دکمه چند بار فشار داده شود .درواقع switch bouncing رفتار غیر ایده آل هر سوئیچ است که چندین transition از یک ورودی واحد ایجاد می کند .هنگامی که ما با مدارهای برق سروکار داریم، switch bouncingمشکل عمده ای نیست، اما در زمانی که با مدارهای منطقی یا دیجیتال سروکار داریم، مشکلاتی ایجاد می کند.

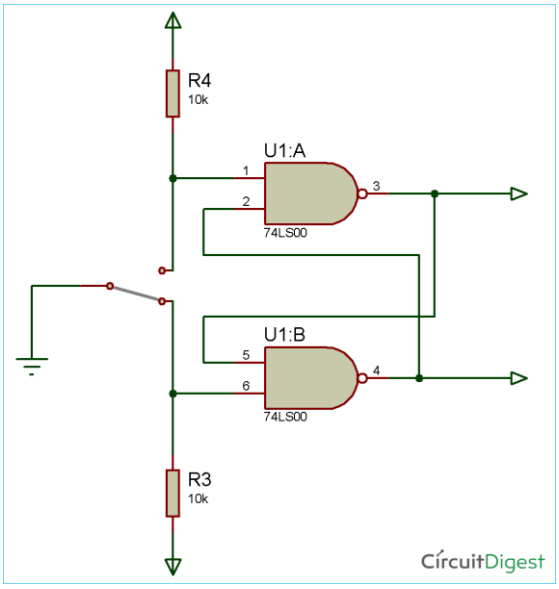
۱- **Hardware Debouncing**:

در تکنیک debouncing سخت افزاری از فلیپ فلاپ S-R برای جلوگیری از پرش سوئیچ مدار استفاده می کنیم. این بهترین روش انحرافی در بین همه است.

اجزای مورد نیاز:

- Nand Gate IC 74HC00
- Toggle Switch
- .Resistor (10k - 2nos)
- Capacitor (0.1uf)
- LED
- Breadboar

همانطور که در عکس مدار مشاهده می کنید، هر زمان که کلید به سمت A سوئیچ می شود، منطق خروجی "HIGH" می شود. و از یک oscilloscope برای تشخیص پرش استفاده شده است. و در نتیجه به جای پرش، با یک منحنی جزئی تغییر می کند. مقاومت های مورد استفاده در مدار، مقاومت های pull-up هستند.
هر زمان که جهش ایجاد میشود، فلیپ فلاپ خروجی را حفظ م یکند زیرا «0» از گیت خروجی NAND برمیگردد.



۲- **R-C Debouncing** :

در این روش مدار از یک شبکه RC برای محافظت در برابر پرش سوئیچ استفاده می کند. خازن در مدار، تغییرات فوری در سیگنال سوئیچینگ را فیلتر می کند. در نتیجه هنگامی که سوئیچ در حالت باز است ولتاژ خازن صفر باقی می ماند.

۳- **Switch Debouncing IC** :

آی سی هایی در بازار برای debouncing سوییچ وجود دارد. برخی از آ یس یهای بازگردان MAX6816، MC14490 و LS118 هستند.

۴- **Software debounce** :

هنگام کار با میکروکنترلرها، م یتوانیم با switch bounce به روشی متفاوت برخورد کنیم که هم در فضای سخ تافزار و هم در هزینه صرفهجویی میکند. برخی از برنامه نویسان اهمیت زیادی به سوئیچ های برگشتی نمی دهند و فقط یک تاخیر ۵۰ میلی ثانیه پس از اولین پرش اضافه می کنند. این باعث می شود میکروکنترلر ۵۰ میلی ثانیه صبر کند تا پرش متوقف شود و سپس به برنامه ادامه دهد. این در واقع یک روش خوب نیست، زیرا میکروکنترلر را با تاخیر در انتظار نگه می دارد.

راه دیگر نرم افزاری استفاده از interrupt برای کنترل switch bounce است. توجه داشته باشید که وقفه ممکن است هم در لبه بالارونده و هم در لبه پایین رونده فعال شود.

سوال کدی:

فرض صورت سوال:

ما این سوال را اینطوری در نظر گرفتیم که با دوبار فشار دادن کلید اول، LED سبز و رقم سمت راست نمایشگر سون سگمنت فال می شود و از عدد ۹ شروع به کم شدن می کند. با سه بار فشار دادن کلید دوم، نمایشگر سون سگمنت در هر حالتی که هست متوقف می شود و در این هنگام هرچقدر هم کلید اول فشار داده شود، هیچ اتفاقی نخواهد افتاد. سپس اگر برای بار چهارم کلید دوم فشار داده شود، نمایشگر سون سگمنت در همان حالتی که قبل از متوقف شدن بود، کار را ادامه میدهد.

توضیحات:

در این سوال از ما خواسته شده است که با استفاده از یک میکروکنترولر stm32f401re و با led و 7-segment کار کنیم.

برای این کار ابتدا باید کد سوال خواسته شده را در برنامه keil با زبان c و با استفاده از cmsis-core و کتابخانه stm32f4xx.h بنویسیم و سپس آن را با استفاده از ابزار proteus شبیه سازی کنیم. اما ابتدا قبل از اینکه به سراغ توضیح فایل کدی برویم، اجزا و نحوه وصل کردن این اجزا به بکدیگر را بیان میکنیم.

اجزا:

- یک میکرو کنترولر stm32f401re
- دو عدد LED یکی قرمز و دیگری سبز
- منبع و زمین
- یک 7-segment دو تایی که ورودی های آن active-low هستند
- دو عدد کلید
- دو عدد مقاومت(مورد استفاده قبل از LED ها و برای کاهش شدت جریان عبوری و جلوگیری از سوختن آنها)

هرکدام از این اجزا (به جز زمین و منبع) به یکی از پورت های این 7-segment متصل هستند. نحوه اتصال آنها و همچنین نوع هر پورت را به صورت مختصر در جدول زیر آوردیم:

نوع پورت	اجزا متصل	Port
خروجی	ورودی a سون سگمنت	PA0
خروجی	ورودی b سون سگمنت	PA1
خروجی	ورودی c سون سگمنت	PA2
خروجی	ورودی d سون سگمنت	PA3
خروجی	ورودی e سون سگمنت	PA4

خروجی	ورودی f سون سگمنت	PA5
خروجی	ورودی g سون سگمنت	PA6
خروجی	ورودی فعال کننده رقم سمت چپ سون سگمنت	PA7
خروجی	ورودی فعال کننده رقم سمت راست سون سگمنت	PA8
خروجی	ورودی LED سبز	PA9
خروجی	ورودی LED قرمز	PA10
خروجی	ورودی فعال کننده چراغ کوچک سمت راست پایین هر رقم	PA11
ورودی	خروجی کلید اول	PB0
ورودی	خروجی کلید دوم	PB1

پس برای انجام این کار دو مرحله داریم:

۱- قسمت کد c و نرم افزار keil :

برای اینکار ابتدا می آییم و یک فایل c در source group اضافه می کنیم. در این فایل که کد ما در آن قرار دارد، کتابخانه stm32f4xx.h را include می کنیم. سپس برای سهولت کار و اینکه یادمان بماند که هر کدام از ورودی ها و خروجی های دستگاه های دیگر، کدام پورت ها را در نظر گرفته بودیم، یک سری define انجام می دهیم تا در ادامه به جای استفاده از اعداد، از این اسم ها استفاده کنیم که راحت تر و خوانا تر باشد. هرکدام از این define ها، در اصل عددی که در هر پورت به خروجی مخصوص به آن اختصاص داده شده است، را مشخص می کند(مثلا اگر یک خروجی به PB0 متصل است عدد مربوط به آن، صفر است). پس با توجه به جدول بالا، یثبهدث ها را در کد به صورت زیر انجام دادیم:

```
#include<stm32f4xx.h>

// GPIO A(OUTPUTS):
//7-SEGMENT
#define a 0 //7-SEGMENT a
#define b 1 //7-SEGMENT b
#define c 2 //7-SEGMENT c
#define d 3 //7-SEGMENT d
#define e 4 //7-SEGMENT e
#define f 5 //7-SEGMENT f
#define g 6 //7-SEGMENT g
#define SSD1 7 //THE LEFT DIGIT
#define SSD2 8 //THE RIGHT DIGIT
#define dp 11 //7-SEGMENT dp

//LED
#define LED_G 9 // GREEN LED
#define LED_R 10 // RED LED

//GPIO B BUTTONS (INPUTS):
#define SW1 0
#define SW2 1
```

سپس برای اینکه در این فرآیند به ست کردن بعضی از بیت ها نیازمندیم، پس یک سری MASK هایی تولید می کنیم که این کار را برای ما حات تر کنند.
MASK اول که SET1 نام دارد، همانطور که از نام آن مشخص است، یک ورودی x می گیرد و عدد1 عه unsigned long را x بیت به سمت راست شیفت می دهد. پس در اصل اگر آن را را عدد یک عددی OR کنیم، بیت x حاصل را 1 می کند.
MASK دوم که SET0 نام دارد، همانطور که از نام آن مشخص است، یک ورودی x می گیرد و سپس عدد 1 عه unsigned long را x واحد به سمت راست شیفت می دهد. و سپس نقیض آن عدد را محاسبه می کند. پس اگر دقت کنیم حاصل یک عدد است که جر بیت x آن تمام بیت های آن 1 است. پس اگر از این عدد در AND کردن استفاده کنیم، می تواند یک بیت خاص(بیت x را) برایمان صفر کند.
کد های زیر برای همین منظور است:

```
//MASKS:
#define SET1(x) (1ul << (x))
#define SET0(x) (~(1ul << (x)))
```

حال در کد ورودی و خروجی توابعی را که از آنها استفاده کردیم را به همراه اسمشان مشخص میکنیم.(در ادامه هر یک را توضیح خواهیم داد):

```
// FUNCTIONS
void Set7Segment(void);
int Delay(volatile int time);
void ChangeLEDAndDigit(void);
void Handle(void);
```

برای اینکه ساز و کار مورد نظر را پیاده سازی کنیم، احتیاج به تعریف تعدادی متغییر داریم که بتوانیم با استفاده از آنها حالت های مختلف را از هم جدا کنیم و در هر حالت خروجی مطلوب را ست کنیم. همچنین بعضا نیاز داریم که بتوانیم درون بعصی از این توابع مقدار های یکی دو تا از این متغییر ها را عوض کنیم یا دوباره ست کنیم. پس برای این منظور، یک سری متغییر استاتیک تعریف کردیم که در همه تابع ها در دسترس باشند و همچنین بتوانیم آنها را در هرکدام از این توابع نیز تغییر بدهیم. در ادامه کارکرد هر کدام از این متغییر ها را خواهیم گفت:

- Sw1_number :

این متغییر برای این است که بتوانیم تعداد کلیک کردن های روی دکمه شماره اول را داشته باشیم.(همانطور که در صورت سوال ذکر شده است، اگر کلید اول را دوبار فشار دهیم، یک سری از تغیرات ایجاد می شود و لی اگر یک بار فشار دهیم، اتفاق نمیفتد.) ما برای اینکه بتوانیم بفهمیم که دفعه چندم است که این کلیک کردن انجام شده است به یک متغییر برای ذخیره کردن تعداد کلیک های انجام شده داریم.

- Sw_number2:

دقیقا همانکارکرد متغییر بالا را دارد، اما برای کلید دوم.(تعداد دفعاتی که کلید دوم فشار داده شده است را نگه می دارد)

- State:

این متغییر در اصل استیتی را که در هر لحظه در آن هستیم مشخص می کند. این استیت ها در اصل اعداد صحیح هستند.(اعداد صحیح از بازه 2- تا 2) استیت شماره یک، (که همان استیتی که در لحظه شروع در آن هستیم نیز می باشد) حالتی است که در آن LED عه سبز روشن است و شمارنده سمت راست فعال است. استیت شماره ۲ استیتی است که در آن LED قرمز و همینظور شمارنده سمت چپ فعال است. استیت شماره صفر یعنی اینکه فعلا در حالتی هستیم که عدد روی سون سگمنت نباید هیچ تغییری بکندو باید ثابت بماند. اما استیت های منفی تنها زمانی بوجود می آیند که ما کلید دوم را ۳ بار فشار دهیم. در این صورت ما برای اینکه هم روند شمارش را متوقف کنیم و هم اینکه این استیتی که تا آن لحظه در آن بودیم و بعد از چهارمین بار کلیک شدن دکمه دوم باید به آن برگردیم، پس نیاز است که ما این استیت را ذخیره داشته باشیم. برای همین منظور ما state را برابر با منفی همان استیتی که در آن بود قرار دادیم. در این حالت با قرارداد اینکه استیت منفی یعنی حالتی که در آن روند شمارش متوقف است، می توانیم هم باعث توقف شمارنده شویم و هم استیت قبلی را(که عدد خود state می شود قرینع استیت قبلی) در آن نگه داریم. سپس هرگاه هم بخواهیم که از حالت توقف خارج شویم، میدانیم که قرینه استیت قبلیمان، را دراختیار داریم.

- Counter:

این متغییر برای این قرار دارد که عددی که نمایشگر 7-segment نشان می دهد را مشخص کند و با استفاده از تابع Set7Segment برای پین های سون سگمنت، پین های مورد نظر را با توجه به این عدد ست میکنیم.

تیکه کد زیر برای همین مورد است:

```
/*
 ****************************** NUMBERS *********************************
 */
// NUMBERS FOR FINDING HOW MANY TIMES ONE HAS BEEN PUSHED
static int sw1_number = 0, sw2_number = 0;
static int state = 1; // 1 => RIGHT , 2 => LEFT , 0 => STOP , ALL NEGATIVE NUMBERS => STOP
static int counter = 9; // SHOWS NUMBER OF THE 7-SEGMENT
```

حال ابتدا به سراغ تابع main میرویم و آن را ابتدا توضیح میدهیم و سپس در هر قسمت توضیح تابع های مورد نظر را می آوریم.

در این تابع، ابتدا باید بیایم و initialization های اولیه را انجام دهیم (شامل تنظیمات پورت ها که هرکدام به عنوان ورودی یا خروجی استفاده می شوند و یا No pull-up, pull-down هستند یا ...)برای هر پوورت و سپس مقدار های اولیه را مقدار دهی کنیم.(برای سون سگمنت و LED ها و ...)
پس برای اینکار به صورت زیر پیش می رویم:
ابتدا برای اینکه GPIO های A و B قابل استفاده شوند، باید کلاک آنها را روشن کنیم. برای اینکار ابتدا باید در رجیستر RCC (Reset and Clock Control) پین مربوط به پورت های A و B را که از آنها در این پروژه استفاده کرده ام، فعال کنیم. یک راه این بود که با استفاده از MASK ها و پین مرتبط با هر کدام از این GPIO ها آنها را فعال کنیم، اما یک راه راحت تر که کتابخانه stm32f4xx.h دارد این است که از مقادیر پیش فرض خودش که مربوط به ست کردن این پین هاست استفاده کنیم.(همین روش دوم را انجام دادیم.) هم برای GPIO عه A و هم B اینکار را انجام دادیم. کد های زیر برای همین منظور است:

```
RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOAEN; // TURNING ON THE CLOCKS FOR GPIOA
RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOBEN; // TURNING ON THE CLOCKS FOR GPIOB
```

سپس میدانیم که با توجه به نحوه استفاده کردن از آنها و توضیح کوتاه داده شده در جدول ابتدای کار، هر کدام از این پین های GPIOA و GPIOB خروجی هستند یا ورودی.(همه پین های خروجی را در این سوال بر روی GPIOA قرار دادیم و تمام پین های ورودی را بر روی GPIOB قرار دادیم.) حال با توجه به اینکه باید مشخص کنیم که وظیفه هر کدام از این پین های GPIOA و GPIOB چیست و همچنین با توجه به توضیحات موجود در رفرنس ها، میدانیم که برای اینکه یک پینی قابلیت خواندن داشته باشد، باید mode آن مشخص باشد و برای اینکه یک پین بتواند یک مقداری را بخواند، باید mode آن (mode مربوط به همان پین) مقدار "01" قرار داده شود. و همچنین می دانیم که اگر بخواهیم مطلبی را با استفاده از یکی از پین ها بنویسیم، باید mode آن پین، به مقدار "00" ست شود.
پس با توجه به توضیحات فوق و همچنین اینکه در جدول ابتدا آمده است که کدام پین ها را به کدام منظور(خواندن یا نوشتن) نیاز داریم، میایم و برای پورت های A و B آن مقدار]ا را ست میکنیم. برای پورت A ، برای اینکه می خواهیم از پین 0 تا 11 آن قابلیت نوشتن داشته باشند، پس باید برای هرکدامشان، رجیستر مرتبط با خودشان را مقداردهی کنیم. برای اینکار می توانستین از MASK ها استفاده کنیم، ولی خب خیلی طولانی تر میشد، به همین دلیل به طور مستقیم مقدار دهی را انجام دادیم. برای اینکار با توجه به اینکه از پین 0 تا پین 11 باید قابلیت نوشتن را داشته باشند، باید برای هرکدامشان مقدار "01" را ست کنیم که خب چون پشت سر هم هستند(در رجیستری که باید این مقدار ها را ست کنیم) پس صرفا باید اینها را به هم بچسبانیم و یک عدد هگز تولید می شود و سپس آن را به رجیستر مورد نظر اساین می کنیم. پس 12 تا "01" پشت سر هم باید بیایند که پس حاصل می شود:
010101010101010101010101 => 0101 0101 0101 0101 0101 0101 => 0x555555

سپس باید مشابه همین کار را برای پورت A انجام دهمی با این تفاوت که در این پورت فقط دو پین ابتدایی را مقداردهی کنیم آن هم مقداردهی مناسب برای خواندن ورودی. پس با توجه به اینکه صرفا تعداد دو تا پین است، پس با استفاده از MASK ها مقداردهی را انجام می دهیم. بر ای این کار این رجیستر مورد نظر را با عددی که بیت های 0 تا سومش صفر است، NAD می کنیم، تا در خروجی چهار بیت ذکر شده همگی مطابق انتظار، مقدار 0 را داشته باشند. تیکه کد زیر برای همین دو قسمت گفته شده است:

```
// GPIO A IS OUTPUT SO WE SET MODE REGISTER "01" FOR IT(WRITING MODE). AND BECAUSE FROM 0 TO 11 (12 NUMBER) ARE USED WE
// ARE USING 12 "01". => 0101 0101 0101 0101 0101 0101 => 5 5 5 5 5 5
GPIOA -> MODER = 0x555555;
// GPIO B IS INPUT SO WE SET MODE REGISTER "00"  FOR IT(READING MODE) FOR BOTH INPUTS.
GPIOB -> MODER &= SET0(0) & SET0(1) & SET0(2) & SET0(3);
```

حال به خاطر اینکه در شماتیک مدار، از pull-down های خود پین های ورودی استفاده کردیم، باید بیایم و بیت هایی که مشخص می کنند ورودی های ما از pull-down های خود میکروکنترولر استفاده می کنند را ست کنیم، تا مدار درست کار کند.برای اینکار باید مقدار رجیستر مشخص کننده حالت ورودی (همین حالت -push down بودن یا ...) برای حالتی که pull-down باشد مقداردهی کنیم. با توجه به اینکه در رفرنس منوآل گفته شده که برای pull-down بودن، باید بیت ها را به شکل "10" مقدار دهی کرد و با توجه به اینکه ما دو پین ورودی داریم که پشت سر هم هستند، پس از MASK ها استفاده میکنیم. برای اینکار بیت هایی که انتظار داریم یک باشند را (بیت شماره صفر و بیت شماره دو) را با MASK ای که صفر میکرد، AND می کنیم. و برای آن بیت های هم که باید یک شوند(بیت شماره یک و بیت شماره سه) هم از MASK سازنده ک استفاده می کنیم و آن را با رجیستر مورد نظر OR میکنیم. کد زیر برای همین بخش است:

```
// SETTING B0 AND B1 PULL DOWN "10". AND WE HAVE TO FO IT FOR OUR BOTH SW
GPIOB ->PUPDR |= SET1(1); // SW1
GPIOB ->PUPDR &= SET0(0); // SW1
GPIOB ->PUPDR |= SET1(3); // SW2
GPIOB ->PUPDR &= SET0(2); // SW2
```

حال باید کلاک خود سیستم کتنفیگوریشن را روشن کنیم. برای این کار می توانستیم از MASK ها استفاده کنیم و یا با استفاده از فیچر های کتابخونه stm32f4xx.h استفاده کنیم، که راه دوم را انجام دادیم. تیکه کد زیر برای همین بخش است:

```
//TURNING ON SYSTEM CONFIGURATION CLOCK.
RCC -> APB2ENR |= RCC_APB2ENR_SYSCFGEN;
```

حال مقداردهی های اولیه برای حالت آغازین را انجام میدهیم. برای اینکار ابتدا تابع Set7Segment را فراخوانی میکنیم.(در قسمت بعد، در رابطه با این تابع توضیح می دهیم.) در این تابع، با توجه به متغییر counter همانطور که قبلا ذکر شد، پین های سون سگمنت، مقداردهی می شوند.(آن پین هایی که برای نمایش عدد بر روی سونسگمنت تغییر می کنند را در این تابع با توجه به مقدار متغییر counter مقداردهی میکنیم.) همچنین با توجه به اینکه در شروع LED قرمز باید خاموش باشد و LED سبز باید روشن باشد، پس باید پین های متناظر با این دو را هم مقداردهی کنیم. برای مقداردهی کردن خروجی، باید بر جای مناسب رجیستر GPIOA(چون خروجی در پورت A است برای همین این GPIO را قرار دادیم) که بستگی به پین مورد نظر دارد.(اینکه کدام بیت این رجیستر باید مقدار موردنظر را بگیرد) مقداردهی میشود. برای اینکه پینی را روشن کنیم باید مقدار آن پین را یک کنیم و برای اینکه یک پینن را خاموش کنیم باید مقدار آن را صفر کنیم. با توجه به define های صورت گرفته در ابتدای برنامه، میتوانیم برای اینکه مقدار پین LED سبز را روشن کنیم، (با استفاده از MASK ها) از LED_G استفاده کنیم که با توجه به عددی که در ابتدای برنامه برای آن گذاشته شد(شماره پین آن) بیت مورد نظر را می توان یک کرد. و مشابه همین کار را می توان برای LED قرمز انجام دادریا، فقط این LED باید خاموش شود و مقدار آن یک شود.همچنین باید نمایشگر، از بین دو عدد موجود، عدد سمت راستی را در ابتدا نشان دهد، که برای همین منظور به طور مشابه با LED ها، پین های متناظر با آنها را ست می کنیم. یک پین دیگر را هم در اینجا مقدار دهی میکنیم و آن را خاموش میکنیم(البته چون این پین و دو پینی که نمایانگر عدد روشن شده در نمایشگر هستند، متعلق به سون سگمنت هستند و این سون سگمنت هم active-low است، پس مقدار آنها نقیض می شود، اگر بخواهیم یکی روشن باشد باید مقدار 0 را بگیرد و اگر بخواهیم خاموش باشد باید مقدار 1 را بگیرد. این پین dp هم استفاده اش در زمانی است که نمایشگر ثابت می ماند، در آن لحظه، این پین روشن خواهد شد.) تیکه کد زیر مربوط به همین مورد است:

```
//SETTING THE DEFAULT VALUE OF THE 7-SEGMENT
Set7Segment();
GPIOA -> ODR |= SET1(LED_G); // TURNING ON THE GREEN LED
GPIOA -> ODR &= SET0(LED_R); // TURNING OFF THE RED LED

GPIOA -> ODR |= SET1(SSD2); // TURNING ON THE RIGHT DIGIT OF THE 7-SEGMENT
GPIOA -> ODR &= SET0(SSD1); // TURNING OFF THE LEFT BIT OF THE 7-SEGMENT
GPIOA -> ODR |= SET1(dp); // TURNING OFF THE LITTLE LIGHT IN THE 7-SEGMENT(dp)
```

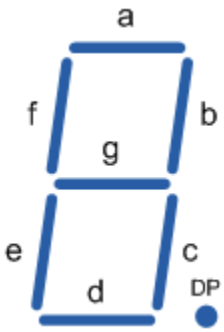
حال در این قسمت، حلقه بی نهایتمان را داریم که ساز و کار اصلی در آن انجام می شود. چون در این حلقه، کارهایی که باید صورت میگرفت خیلی زیاد بود، برای همین برای آن یک تابع جدا در نظر گرفتیم و تمام کار ها را در آن تابع انجام دادیم. پس در این حلقه بی نهایت فقط این تابع را فراخوانی کردیم. اسم این تابع، Handle می باشد.(در ادامه در رابطه با این تابع توضیح می دهیم.) تیکه کد زیر برای همین قسمت است:

```
while(1){
  Handle();
}
```

حال به سراغ توابع تعریف شده و استفاده شده در این پرژه می رویم و هرکدام را توضیح میدهیم:

- تابع Set7Segment:

این تابع همانطور که کارکرد آن در قسمت های قبل به شکل مختصر توضیح داده شد برای این است که با توجه به عدد جدید متغییر counter ، طوری پین های متصل به نمایشگر سونسگمنت را مقدار دهی کند که عددی که نمایش داده خواهد شد، همان عدد counter باشد. برای این کار هر سری که این تابع فراخوانی می شود، ۱۰ حالت مختلف بوجود میاید که ناشی از حالت های مختلف ورودی می باشد. در این تابع تنها پین هایی مقدار دهی میشوند که در تعیین عدد نشان داده شده بر روی نمایشگر تاثیر داشته باشند.(تنها پورت هایی که به a,b,c,d,e,f و g وصلهستند را مقدار دهی میکنیم)
همانطور که از تمرین های قبل می دانیم، در یک سون سگمنت مطابق با شکل زیر، هرکدام از ورودی ها متناظر با یمی از خط های موجود در نمایشگر است که با توجه به آن ورودی آن خط روشن و یا خاموش می شود. شکل زیر بیانگر این خط ها و حرف متناظر با آنها است:



پس با توجه به این عکس، برای هر کدام از اعداد صفر تا ۹ (کل اعداد ممکن برای ورودی) می توانیم پیدا کنیم که کدام خط ها باید روشن و کدام خط ها باید خاموش باشند.(در نظر میگیریم که سون سگمنت ما، active-low است، بنابر این در ورودی های آن، صفر به معنای روشن و یک به معنای خاموش است.)

	a	b	c	d	e	f	g
0	0 روشن	0 روشن	0 روشن	0 روشن	0 روشن	0 روشن	1 خاموش
1	1 خاموش	0 روشن	0 روشن	1 خاموش	1 خاموش	1 خاموش	1 خاموش
2	0 روشن	0 روشن	1 خاموش	0 روشن	0 روشن	1 خاموش	0 روشن
3	0 روشن	0 روشن	0 روشن	0 روشن	1 خاموش	1 خاموش	0 روشن
4	1 خاموش	0 روشن	0 روشن	1 خاموش	1 خاموش	0 روشن	0 روشن
5	0 روشن	1 خاموش	0 روشن	0 روشن	1 خاموش	0 روشن	0 روشن
6	0 روشن	1 خاموش	0 روشن	0 روشن	0 روشن	0 روشن	0 روشن
7	0 روشن	0 روشن	0 روشن	1 خاموش	1 خاموش	1 خاموش	1 خاموش
8	0 روشن	0 روشن	0 روشن	0 روشن	0 روشن	0 روشن	0 روشن
9	0 روشن	0 روشن	0 روشن	0 روشن	1 خاموش	0 روشن	0 روشن

حال با توجه به جدول، یک switch بزرگ ساختیم که برای هر حالت ورودی به طور جداگانه مقدار ها را تنظیم میکند. سپس برای هر ورودی با توجه به مقادیر موجود در جدول، پین متناظر با آن را صفر یا یک کردیم. روش یک سا صفر کردن هر پین هخم همانند قبل است و تغییری ندارد.(برای اینکار در رجیستری که می خواستیم خروجی را در آن قرار دهیم، بیتی که متناظر با پین مورد نظرمان بود (که البته با استفاده ازdefineهای ابتدا دیگر نیازی به نوشتن خود عدد آنها نبود و از همان define ها استفاده میشد) و با استفاده از MASK ها، بیت متناظر را در صورت نیاز یک یا صفر کردیم.) تیکه کد مربوط به این تابع به صورت زیر است:

```
void Set7Segment(void){ // HERE WE SET THE 7-SEGMENT FOR THE COUNTER.(OUR 7-SEGMENT IS ACTIVE-LOW)
  switch(counter){
    case 0: // ONLY G IS OFF
      GPIOA -> ODR |= SET1(g); // OFF
      GPIOA -> ODR &= SET0(a) & SET0(b) & SET0(c) & SET0(d) & SET0(e) & SET0(f); // ON
      break;
    case 1: // ONLY B AND C ARE ON
      GPIOA -> ODR |= SET1(a) | SET1(d) | SET1(e) | SET1(f) | SET1(g); // OFF
      GPIOA -> ODR &= SET0(b) & SET0(c); // ON
      break;
    case 2: // ONLY C AND F ARE OFF
      GPIOA -> ODR |= SET1(c) | SET1(f); // OFF
      GPIOA -> ODR &= SET0(a) & SET0(b) & SET0(d) & SET0(e) & SET0(g); // ON
      break;
    case 3: // ONLY E AND F ARE OFF
      GPIOA -> ODR |= SET1(e) | SET1(f); // OFF
      GPIOA -> ODR &= SET0(a) & SET0(b) & SET0(c) & SET0(d) & SET0(g); // ON
      break;
    case 4: // ONLY A, D AND E ARE OFF
      GPIOA -> ODR |= SET1(a) | SET1(d) | SET1(e); // OFF
      GPIOA -> ODR &= SET0(b) & SET0(c) & SET0(f) & SET0(g); // ON
      break;
    case 5: // ONLY B AND E ARE OFF
      GPIOA -> ODR |= SET1(b) | SET1(e); // OFF
      GPIOA -> ODR &= SET0(a) & SET0(c) & SET0(d) & SET0(f) & SET0(g); // ON
      break;
    case 6: // ONLY B IS OFF
      GPIOA -> ODR |= SET1(b); // OFF
      GPIOA -> ODR &= SET0(a) & SET0(c) & SET0(d) & SET0(e) & SET0(f) & SET0(g); // ON
      break;
    case 7: // ONLY A, B AND C ARE ON.
      GPIOA -> ODR |= SET1(d) | SET1(e) | SET1(f) | SET1(g); // OFF
      GPIOA -> ODR &= SET0(a) & SET0(b) & SET0(c); // ON
      break;
    case 8: // ALL OF THEM ARE ON
      GPIOA -> ODR &= SET0(a) & SET0(b) & SET0(c) & SET0(d) & SET0(e) & SET0(f) & SET0(g); // ON
      break;
    case 9: // ONLY E IS OFF
      GPIOA -> ODR |= SET1(e); // OFF
      GPIOA -> ODR &= SET0(a) & SET0(b) & SET0(c) & SET0(d) & SET0(f) & SET0(g); // ON
      break;
  }
}
```

- تابع Delay:

این تابع در اصل باید برای ساخت delay استفاده میشد. اما چون در این تمرین اجازه استفاده از interrupt ها را نداریم، مجمور شدیم از این تابع برای چک کردن تغییر ورودی ها (کلیک شدن دکمه ها) هم استفاده کنیم. چون این تابع در اصل برای ساخت دیلی بود و همانطور که گفته شده بود، برای ساخت دیلی، دو حلقه تودرتو ساختیم که هرکدامشان به اندازه ورودی تابع طی میشوند.(این کار را انجام دادیم و ورودی تابع را به شکل volatile int گذاشتیم تا باعث ایجاد دیلی در مدار شود، در غیر اینصورت، هیچ دیلی ای ایجاد نمیشد.) سپس در هر سری پیمایش حلقه دردونی، ما یک سری شرط را چک میکنیم که این شرط ها معادل با این هستند که آیا کلیک شده است یا نه. برای اینکار آن پین خروجی هر کداک از کلید های ورودی با را استفاده از MASK ها بیت متناظر با آن پین را چک میکنیم که یک شده است یا خیر؟ اگر یک شد حاصل جواب، یعنی کلیک شده است و به داخل بدنه if می رویم. در بدنه if ما در اصل یک while ای را میبینیم که شرط آن در اصل همان شرط if مان بود. و می گوییم تا وقتی که این شرط برقرار بود منتظر بماند. اما دلیل این کار این است که وقتی کلیک انجام میشود، مدت زمانی که این پین ورودی یک میماند، بیش از یک بار اجرا شدن بدنه حلقه توپی است، بنابراین، سری بعد هم، در اصل با همان کلیک قبلی وارد همین if می شود(منظور از همان کلیک قبلی، یک بودن پین مربوط به ورودی کلیدی که کلیک شده است می باشد) و بنابراین، به ازای یک کلیک، بیش از یک بار ما داخل if

می رفتیم و چون اینجا برایمان تعداد کلیک ها مهم است، پس این حلقه را گذاشتیم تا از این اتفاق جلوگیری کند.
این تابع سپس در بدنه if با توجه به اینکه این کلیک چندم کلید اول بود، یا به کار خود ادامه می دهد و به سراغ if بعدی میرود، یا عدد 1 را به عنوان خروجی برمیگرداند.
این تابع ۴ نوع خروجی مختلف دارد:
0: این عدد برای مواقعی است که در حین اجرای حلقه، هیچ اتفاقی که باعث تغییر عملکرد شود، اتفاق نیفتاده است، و ادامه کار باید به صورت طبیعی انجام شود.
1: این عددبرای این است که وقتی کلید اول دوبار فشار داده شد، این عدد را برمیگردانیم و در جایی که این تابع فراخوانی شده است، اقدامات لازم را برای این حالت(در این حالت باید از ابتدا با شماره ۹ شروع به پایین آمددن کند و LED سبز روشن شود.) در همان جایی که این تابع صدا زده شده بود، انجام شود.(این تابع فقط در تابع Handle صدا زده شده است که قبلا توضیح کمی در رابطه ب آن دادیم.)
2:این عدد برای وقتی است که کلید دوم، ۳ بار فشار داده شده است، در این حالت، باید اجرای این تابع متوقف شود و کارهای متناسب با این اتفاق(متوقف شدن تایمر) انجام شود.(این تابع در تابع Handle صدا شده است، پس اقدامات مناسب در همان تابع Handle انجام میشود.)
3: این عدد برای آن است که یعنی باید از حالت متوقف بودن خارج شویم و دوباره کار را ادامه دهیم(ادامه همان نحوه شمارش قبلی).

حال که برای اینکه کلید دوم را هم چک کنیم که آیا فشار داده شده است یا نه، مشابه با حالتی که برای کلید اول چک میکردیم، چک میکنیم و دوباره همان while را قرار میدهیم. سپس دو تا if دیگر نیز قرار میدهیم که چک میکنند آیا تعداد دفعات کلیک کردن، به تعدادی رسیده است که نیازمند اقدامی باشد یا خیر. تیکه کد زیر مربوط به همین تابه می باشد:

```
 ********************************** FUNCTION FOR MAKING A DELAY ************************************
 */
// RETURN VALUE 0 => NO CLICK WHICH LEAD TO ANY CHANGE
// RETURN VALUE 1 => SW1 HAS CLICKED TWICE, SO WE GO TO GREEN STATE(RIGHT OR STATE = 2)
// RETURN VALUE 2 => SW2 HAS CLICKED 3 TIMES, SO WE GO TO STOP STATE (STATE = 0)
// RETURN VALUE 3 => SW2 HAS CLICKED 4 TIMES, SO WE BACK TO OUR PREVIOUS STATE.
int Delay(volatile int time){
  //MAKING TWO NESTED LOOP TO STAYING HERE FOR A SECOND AND IN THIS TIME WE ALSO HAVE TO
  // TAKE CARE OF CLICKS
  for(int i = 0; i < time; i++){
    for (int j = 0; j < time; j++){
      if(GPIOB -> IDR & SET1(SW1) && state > 0){ // IF WE HAVE CLICKED ON THE SW1
        sw1_number++; // WE INCREASE NUMBER OF CLICKING ON SW1
        //WHEN WE CLICK, THE INPUT STAY ONE FOR A LONGER TIME THAN FINISHING THIS BODY, SO AGAIN IT WILL
        //ACT LIKE WE CLICKED. FOR PREVENTING THAT, WE SET A BUSY-WAITING (EMPTY LOOP) TO STAYE HERE
        //AS LONG AS THE INPUT HAS CHANGED TO ZERO.(WHEN WE HAVE FINISHED CLICKING)
        while(GPIOB -> IDR & SET1(SW1))
          ;
        if(sw1_number == 2) // IF WE ARE CLICKING ON THE SW1 FOR THE 2nd TIME, WE RETURN 1, OTHERWISE WE CONTINUE THE PROCESS
          return 1;
      }
      if(GPIOB -> IDR & SET1(SW2)){ // IF WE HAVE CLICKED ON THE SW2
        sw2_number++;// WE INCREASE NUMBER OF CLICKING ON SW2
        //WHEN WE CLICK, THE INPUT STAY ONE FOR A LONGER TIME THAN FINISHING THIS BODY, SO AGAIN IT WILL
        //ACT LIKE WE CLICKED. FOR PREVENTING THAT, WE SET A BUSY-WAITING (EMPTY LOOP) TO STAYE HERE
        //AS LONG AS THE INPUT HAS CHANGED TO ZERO.(WHEN WE HAVE FINISHED CLICKING)
        while(GPIOB -> IDR & SET1(SW2))
          ;
        if(sw2_number == 3) // IF WE ARE CLICKING ON THE SW2 FOR THE 3rd TIME, WE RETURN 2, OTHERWISE WE CONTINUE THE PROCESS
          return 2;
        if(sw2_number == 4)// IF WE ARE CLICKING ON THE SW2 FOR THE 4th TIME, WE RETURN 3, OTHERWISE WE CONTINUE THE PROCESS
          return 3;
      }
    }
  }
  // IF WE DIDN'T PRESS ANYTHING OR ATLEAST IT WASN'T ENOUGH
  return 0;
}
```

- تابع ChangeLEDAndDigit:

این تابع را صرفا وقتی فراخوانی میکنیم که به حالتی رسیده باشیم که شمارنده یک سمت(شمارنده سمت راست یا چپ یا به عبارتی شمارنده LED سبز و یا LED قرمز) به صفر رسیده باشد و هیچ کلیک کردن خاصی هم اتفاق نیفتاده است و صرفا باید جای LED های روشن و سمتی که نمایشگر عدد را نشان می دهد عوض شود. همانطور که در این تابع مشخص است، اول یک if قرار داده ایم تا بفهیمم که در چه حالتی هستیم. برای اینکار از متغییر state استفاده می کنیم. به طوری که همانطور که از قبل قرارداد کرده ایم، اگر1 = state باشد یعنی الان در حالتی هستیم که LED سبز روشن بوده و الان باید LED قرمز روشن شود. اگر در این حالت باشیم، state را برابر 2 می گذاریم و LED سبز و پین روشن کننده نمایشگر راستی را خاموش میکنیم و پین مربوط به نمایشگر سمت چپ و LED قرمز را روشن میکنیم. اگر هم شرط if برقرار نبود، در بدنه else، دقیقا برعکس اینکار را انجام میدهیم و آنهایی را که در حالت قبل روشن کرده بودیم، خاموش و آن هایی را که خاموش کرده بودیم، روشن میکردیم.

تیکه کد زیر مربوط به همین تابع می باشد:

```
/*
 ************************************ FUNCTION FOR SETTING LED AND CHOOSONG THE ON AND OFF DIGIT ************************
 */
void ChangeLEDAndDigit(void){
  if(state == 1){// RIGHT DIGIT WAS ON AND GREEN LED WAS ON
        state = 2; // CHANGING THE STATUS
        GPIOA -> ODR |= SET1(SSD1); // TURNING ON THE LEFT DIGIT
        GPIOA -> ODR &= SET0(SSD2); // TURNING OFF THE RIGHT DIGIT
        GPIOA -> ODR |= SET1(LED_R); // TURNING ON THE RED LED
        GPIOA -> ODR &= SET0(LED_G); // TURNING OFF THE RED
      }else{// LEFT DIGIT WAS ON AND RED LED WAS ON
        state = 1; // CHANGING STATUS
        GPIOA -> ODR |= SET1(SSD2); // TURNING ON THE RIGHT DIGIT
        GPIOA -> ODR &= SET0(SSD1); // TURNING OFF THE LEFT DIGIT
        GPIOA -> ODR |= SET1(LED_G); // TURNING ON THE GREEN LED
        GPIOA -> ODR &= SET0(LED_R); // TURNING OFF THE RED LED
      }
}
```

- تابع Handle:

این تابع همانطور که قبلا اشاره شد برای جلوگیری از شلوغ شدن main ایجاد شده است. در هر سری که اجرای این تابع تمام شود، دوباره خود همین تابع در آن حلقه بی نهایت گفته شده در قسمت main دوباره صدا زده میشود.

این تابع با توجه به اینکه عدد مربوط به نمایشگر و اطلاعات مورد نیاز برای نمایش آن از قبل(قبل از حلقه بی نهایت قسمت main) مقداردهی شده بودند، پس صرفا دیلی را به اندازه ای که تقریبا یک ثانیه باشد، ایجاد میکند.(برای اینکار از همان تابع دیلی ای که تعریف کردیم استفاده می کند.) همانطور که گفته شد، این تابع delay یک خروجی دارد و ما بعد از صدا کردن تابع delay در این تابع، مقدار آن را در یک متغییر local به اسم delay میریزیم. سپس با یک سری شرط مقدار این متغییر را چک میکنیم و در هخر مورد در صورت نیاز تغییراتی را اعمال می کنیم. حالت های زیر ممکن است بوجود بیاید:

- در حالتی که مقدار این متغییر یک باشد و همچنین بدانیم که state مان در حالت توقف نیست(کوچکتر مساوی صفر نیست) این شرط را اجرا می کنیم. در این شرط مقدار counter را یکی کم میکنیم و و سپس چک میکنیم که آیا مقدار حال حاضر counter منفی یک شده است یا نه، واگر منفی یک شده بود، آن را ۹ میکنیم و یک بار تابع ChangeLEDAndDigit را صدا میکنیم تا جای نمایشگر روشن و خاموش و همچنین LED روشن و خاموش عوض شود. سپس چه این شرط برقرار شد چه نشد، (شرطی که میدید آیا counter منفی یک شده است یا نه) تابع Set7Segment را صدا میکنیم تا عدد counter روی نمایشگر نشان داده شود.

- در حالتی که delay یک باشد و state مان در حالت توقف نباشد (یعنی state مان بزرگتر از صفر باشد)، در این حالت باید هر کدام از LED ها که روشن بود را خاموش میکردیم و LED سبز را روشن میکردیم و counter را دوباره عدد ۹ قرار میدادیم و همچنین state را هم یک میکردیم. پس در همین شرط، آمدیم و کار های گفته شده را کردیم و LED قرمز و پین روشن کننده قسمت چپ نمایشگر را خاموش و LED سبز و پین مربوط به قسمت راستی نمایشگر را روشن کردیم و state را هم برابر با یک گذاشتیم. سپس در انتهای این شرط، تابع Set7Segment را صدا کردیم تا عدد counter را بر روی نمایشگر قرار دهد.

- در حالتی که delay برابر با ۲ باشد، در این حالت، باید فرآیند کم شدن عدد روی نمایشگر را متوقف کنیم. برای اینکار state را منفی کردیم، یعنی نوشتیم state = - state که همانطور که قبلا اشاره شد، این کار هم باعث می شود که استیتی که در آن قرار داشتیم حفظ شودو هم اینکه با توجه به اینکه state های موچکتر از صفر را در حالت توقف در نظر گرفتیم، پس در حالت توقف نیز میمانیم. و همچنین چراغ کوچک سمت راست پایین (پین dp) نمایشگر را هم روشن کردیم تا معلوم باشد.

- در حالتی که delay برابر با 3 باشد، در این حالت یعنی دیگر باید از حالت توقف خارج شویم و به اجرای عادی برنامه از همان قسمتی که در حال اجرا بودیم بازگردیم. برای اینکار ابتدا چراغ کوچک سمت راست پایین (پین dp) را هم خاموش کردیم. سپس برای اینکه از state توقف خارج شویم، متغییر state را قرینه کردیم تا مقدار آن همان مقداری که قبل از توقف بود، برگردد. همچنین در اینجا تعداد دفعاتی که کلید شماره دوم فشار داده شده است را هم صفر میکنیم تا برای سری بعدی که صدا می شود، درست عمل کند. و در آخر نیز counter را آپدیت میکنیم و از مقدار آن یکی کم می کنیم.(چون به اندازه کافی بر روی عدد کنونی توقف کرده ایم) و سپس تابع Set7Segment را فراخوانی میکنیم تا مقدار روی نمایشگر را با مقدار counter ست کند.(همچنین چک میکنیم که مقدار counter منفی نشده باشد و اگر منفی شده باشد، اقدامات لازم را انجام می دهیم(تابع ChangeLEDAndDigit را صدا میکنیم و مقدار counter را هم ۹ میگذاریم.))

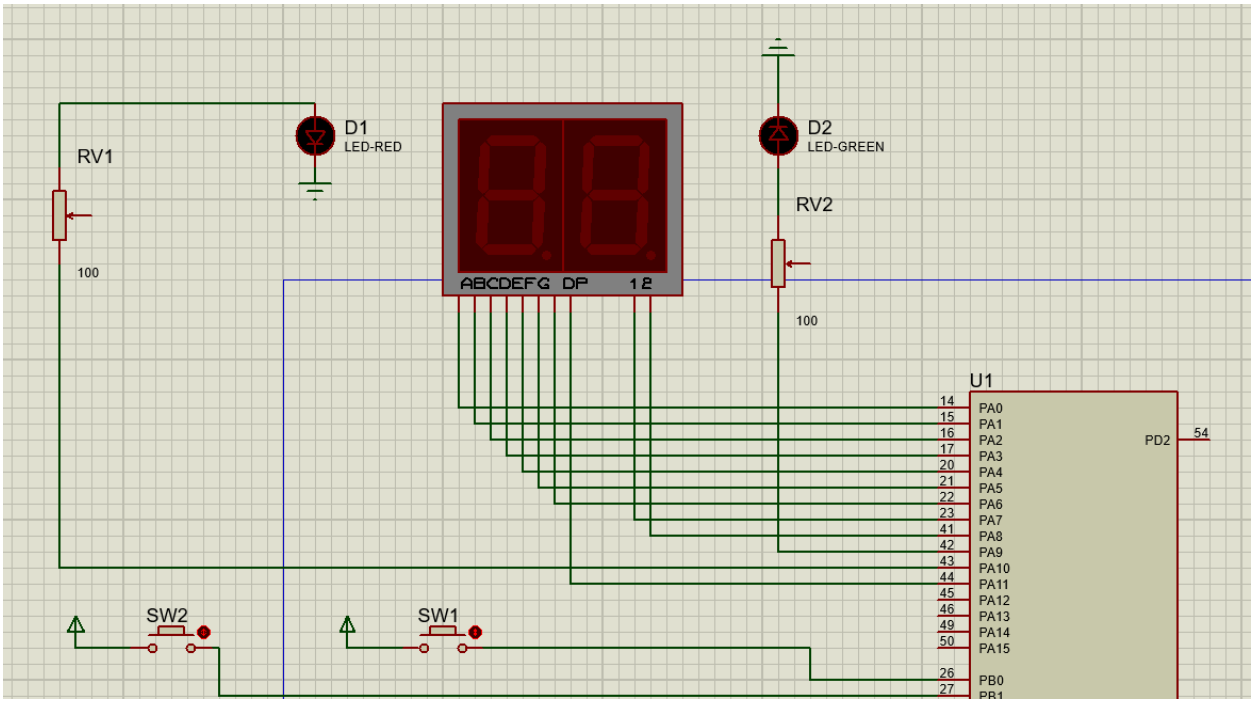
در این سوال ما اینطور در نظر گرفتیم که اگر کلید دوم، ۳ بار کلیک شود، در آن صورت تا قبل از کلیک چهارم کلید دو، درگر اگر کلید شماره ک هم دوبار کلیک شود، هیچ تغییر ایجاد نمیشود.
تیکه کد مربوط به این تابع، به صورت زیر است:

```
/*
 ********************************** FUNCTION FOR HANDLING THE MAIN LOOP ************************************************
 */
void Handle(void){
  int delay = Delay(850);


  if (delay == 0 && state > 0){ // COUNTINUE OUR USUAL WORK
    counter--; // WE HAVE WAITED ENOUGH ON THE PREVOISE COUNTER, SO WE UPDTE THE COUNTER
    if(counter == -1){ // WE CHECK IF THE NEW COUNTER IS NOT OUT OF BANDS(IS NOT LESS THAN 0)
      counter = 9; // IF COUNTER IS NEGATIVE, THEN WE REINITIALIZ IT
      ChangeLEDAndDigit(); // NOW WE CHANGE THE LED COLOR AND THE DIGIT WHICH WAS ACTIVE ON 7-SEGMENT
    }
    Set7Segment(); // WE UPDATE THE 7-SEGMENT WITH THE NEW COUNTER
  }
  if(delay == 1 && state > 0){ // GOING TO THE BEGGINIG OF STATE 1
    sw1_number = 0; // WE RESET NUMBER OF CLICKS ON SW1
    state = 1; // HERE WE CHANGE OUR STATE AND GO TO STATE 1 (STATE = 1)
    GPIOA -> ODR |= SET1(SSD2); // TURNING ON THE RIGHT DIGIT
    GPIOA -> ODR &= SET0(SSD1); // TURNING OFF THE LEFT DIGIT
    GPIOA -> ODR |= SET1(LED_G); // TURNING ON THE GREEN LED
    GPIOA -> ODR &= SET0(LED_R); // TURNING OFF THE RED
    counter = 9; // HERE WE REINITIALIZE THE COUNTER AND
    Set7Segment(); // HERE WE SET THE 7-SEGMENT FOR THE NEW COUNTER
  }
  if(delay == 2){ // CLICKING FOR THE 3rd TIME
    state = -state; // IT WILL SHOW THAT WE SHOULD STOP AND ALSO IN THIS WAY (STATE = -STATE) WE CAN SAVE OUR PREVIOSE STATE
    GPIOA -> ODR &= SET0(dp); // TURNING ON THE LITTLE LIGHT (dp) ON THE 7-SEGMENT
  }
  if(delay == 3){ // CLICKING FOR THE 4th TIME
    sw2_number = 0; // WE RESET NUMBER OF CLICKS ON SW2
    state = -state; // WE ALSO RESTORE THE PREVIOUSE STATE
    GPIOA -> ODR |= SET1(dp); // TURNING OFF THE LITTLE LIGHT (dp) ON THE 7-SEGMENT
    counter--; // WE HAVE WAITED ENOUGH ON THIS NUMBER, SO WE UPDATE THE counter AND
    if(counter == -1){ // WE CHECK IF THE NEW COUNTER IS NOT OUT OF BANDS(IS NOT LESS THAN 0)
      counter = 9; // IF COUNTER IS NEGATIVE, THEN WE REINITIALIZ IT
      ChangeLEDAndDigit(); // NOW WE CHANGE THE LED COLOR AND THE DIGIT WHICH WAS ACTIVE ON 7-SEGMENT
    }
    Set7Segment(); // WE SET THE 7-SEGMENT FOR THE NEW COUNTER
  }
}
```

۲- قسمت شبیه سازی در proteus :

در این قسمت صرفا عکس هایی از شماتیک مدار و چند نکته را قرار میدهیم.



- در اینجا، قبل از LED ها یک مقاومت قرار داده ایم زیرا میخواهیم با این حرکت از ورود جریان با شدت بلا به LED که ممکن است باعث سوختن آن شود، جلوگیری شود.

- برای اینکه بتوانیم برنامه ای که زدیم را بر روی میکروکنترولر، بر روی آن ریات کلیک کردیم و سپس به قسمت edit properties رفتیم و در آنجا در قسمت program file آدرس فایل hex ساخته شده به وسیله keil را قرار می دهیم.

- تمامی موارد مورد نیاز ازجمله LED ها ، سون سگمنت، کلید ها، مقاومت ها، نحوه اتصال پین ها و همچنین پین های مورد استفاده و ... در عکس بالا قرار دارد.

- نحوه اتصال پین ها به بخش های سون سگمنت، مبنای قراردادن آن define های داخل فایل کد c می باشد.

- با تنظیم مقاومت ها، شدت روشنایی LED ها کم و زیاد میشد، برای همین آنها را (مقاومتشان را) کم کردیم تا روشن شدن آنها واضح تر باشد.

- به جز راه حل گفته شده در قسمت تابع Delay برای جلوگیری از چندین بار خوانده شدن یک بار کلیک شدن کلید، میشو قبل از ورود آنها به میکروکنترولر هم خازن گذاشت، اما این راه را با چند نوع خازن موجود امتحان کردیم ولی جواب نداد.(احتمالا در تنظیماتش باید تغییرات خاصی لحاظ میکردیم.)

- در این شبیه سازی در اصل برای پایه های سون سگمنت هم باید یک مقاومت قرار میدادیم.(مثل برای LED ها تا زا اینکه جریان به شدت از آن عبور کند و باعث سوختن و یا آسیب به آن نشود) اما به دلیل آنکه شکل خیلی شلوغ و نامرتب میشد، از گذاشتن آنها صرف نظر کردیم.

منابع:

سوالات تحلیلی:

- https://www.keil.com
- https://developer.arm.com
- https://armsoftware.github.io
- https://www.eca.ir
- https://microcontrollerslab.com
- https://circuitdigest.com
- https://www.allaboutcircuits.com
- کلاس حل تمرین و کلاس حل تمرین و اسلاید های استاد

سوال کدی:

- https://www.keil.com/pack/doc/cmsis/DSP/html/index.html
- https://www.st.com/en/embedded-software/stsw-stm32065.html
- https://stm32f4-discovery.net/2014/04/stm32f429-discovery-gpio-tutorial-with-onboard-leds-and-button/
- https://www.aparat.com/v/6ESNT?playlist=26086398
- https://www.keil.com/dd/docs/arm/st/stm32f4xx/stm32f4xx.h
- https://stm32f4-discovery.net/2015/03/library-54-general-library-for-stm32f4xx-devices/
- https://stackoverflow.com/questions/9541768/error-no-previous-prototype-for-function-why-am-i-getting-this-error
- https://forum.allaboutcircuits.com/threads/real-time-simulation-in-proteus.86978/
- https://www.edaboard.com/threads/proteus-real-time-simulation.200427/
- https://www.edaboard.com/threads/proteus-error-excess-cpu-load-simulation-cannot-run-in-real-time.261791/
- https://microcontrollerslab.com/7-segment-display-interfacing-with-pic-microcontroller/
- https://www.aparat.com/v/ywiSD
- **کلاس درس استاد و حل تمرین**
- **رفرنس منوآل**
- **دیتاشیت**