



دانشکده مهندسی برق

دستور کار آزمایشگاه میکروپروسسور

فهرست مطالب

آزمایش ۱: کار با پورت‌های ورودی - خروجی عمومی (GPIO)

۶	
۷	۱-۱- درسنامه
۷	۱-۱-۱- پیکربندی ورودی
۸	۱-۱-۲- پیکربندی خروجی
۹	۱-۲- آزمایش
۱۲	۱-۲-۱- ال‌ای‌دی چشمک‌زن
۱۲	۱-۲-۲- روشن و خاموش کردن ال‌ای‌دی به کمک دو Pushbutton
۱۲	۱-۳- تنظیمات نرم‌افزاری و کدنویسی
۱۲	۱-۴- پرسش‌ها و تمرین‌های برنامه نویسی

آزمایش ۲: راه‌اندازی نمایشگر هفت‌قطعه‌ای (7-Segment)

۱۴	
۱۵	۲-۱- درسنامه
۱۶	۲-۱-۱- روش مالتی پلکس
۱۷	۲-۲- آزمایش
۱۷	۲-۲-۱- شمارنده‌ی تک رقمی ۰ تا ۹
۱۷	۲-۲-۲- شمارنده‌ی تک رقمی ۰ تا ۹۹
۱۷	۲-۳- تنظیمات نرم‌افزاری و کدنویسی
۱۸	۲-۴- پرسش‌ها و تمرین‌های برنامه نویسی

آزمایش ۳: راه‌اندازی صفحه‌کلید ماتریسی (Keypad)

۱۹	
۲۰	۳-۱- درسنامه
۲۱	۳-۲- آزمایش
۲۱	۳-۲-۱- راه‌اندازی صفحه‌کلید ماتریسی
۲۱	۳-۳- تنظیمات نرم‌افزاری و کدنویسی
۲۱	۳-۴- پرسش‌ها و تمرین‌های برنامه نویسی

آزمایش ۴: وقفه‌ی خارجی در میکروکنترلرهای STM32

۲۴	
۲۵	۴-۱- درسنامه
۲۵	۴-۲- آزمایش
۲۵	۴-۲-۱- تغییر وضعیت LED به کمک وقفه‌ی خارجی

۲۵ ۴-۳- تنظیمات نرم‌افزاری و کدنویسی

۲۸ ۴-۴- پرسش‌ها و تمرین‌های برنامه نویسی

۲۹ **آزمایش ۵: راه‌اندازی نمایشگر کاراکتری**

۳۰ ۵-۱- درسنامه

۳۲ ۵-۲- آزمایش

۳۲ ۵-۲-۱- درایورنویسی نمایشگر کاراکتری مد ۴ بیتی

۳۳ ۵-۲-۲- استفاده از نمایشگر

۳۳ ۵-۳- پرسش‌ها و تمرین‌های برنامه نویسی

۳۴ **آزمایش ۶: کار با واحد زمان‌سنج حقیقی (RTC)**

۳۵ ۶-۱- درسنامه

۳۶ ۶-۲- آزمایش

۳۶ ۶-۲-۱- ساعت دیجیتال

۳۶ ۶-۳- تنظیمات نرم‌افزاری و کدنویسی

۳۸ ۶-۴- پرسش‌ها و تمرین‌های برنامه نویسی

۳۹ **آزمایش ۷: کار با واحد زمان‌سنج‌ها و شمارنده‌ها (Timer/Counter) - بخش اول**

۴۰ ۷-۱- درسنامه

۴۰ ۷-۱-۱- زمان‌سنج چیست و چگونه ساخته می‌شود؟

۴۰ ۷-۱-۲- ساخت زمان‌سنج دیجیتال

۴۱ ۷-۱-۳- زمان‌سنج‌ها در میکروکنترلرهای STM32

۴۳ ۷-۱-۴- کاربرد زمان‌سنج‌ها در میکروکنترلرهای STM32

۴۵ ۷-۱-۵- مد Input Capture

۴۷ ۷-۲- آزمایش

۴۷ ۷-۲-۱- Basic Timer

۴۷ ۷-۲-۲- Input Capture

۴۹ ۷-۳- تنظیمات نرم‌افزاری و کدنویسی

۴۹ ۷-۳-۱- Basic Timer

۵۲ ۷-۳-۲- Input Capture

۵۵ ۷-۴- سؤال‌ها و تمرین‌های برنامه‌نویسی

۵۶ **آزمایش ۸: کار با واحد زمان‌سنج‌ها و شمارنده‌ها (Timer/Counter) - بخش دوم**

۵۷ ۸-۱- درسنامه

۵۷	۸-۱-۱- مد مقایسه خروجی چیست؟
۵۷	۸-۱-۲- مدولاسیون عرض پالس (PWM)
۵۹	۸-۲- آزمایش
۵۹	۸-۲-۱- مد مقایسه خروجی
۵۹	۸-۲-۲- مدولاسیون عرض پالس
۶۰	۸-۳- تنظیمات نرم‌افزاری و کدنویسی
۶۱	۸-۳-۱- آزمایش زمان سنج در مد مقایسه خروجی
۶۳	۸-۳-۲- آزمایش زمان سنج در مد مدولاسیون عرض پالس
۶۴	۸-۴- پرسش‌ها و تمرین‌های برنامه نویسی

آزمایش ۹: کار با واحد مبدل آنالوگ به دیجیتال (ADC)

۶۵	۹-۱- درسنامه
۶۷	۹-۲- آزمایش
۶۷	۹-۳- تنظیمات نرم‌افزاری و برنامه نویسی
۷۱	۹-۴- پرسش‌ها و تمرین‌های برنامه نویسی

آزمایش ۱۰: پروتکل ارتباطی SPI

۷۳	۱۰-۱- درسنامه
۷۳	۱۰-۱-۱- انتقال داده در پروتکل SPI
۷۴	۱۰-۱-۲- مزایا و معایب پروتکل SPI
۷۵	۱۰-۱-۳- سایر نکات پروتکل SPI
۷۵	۱۰-۱-۴- پارامترهای مهم پروتکل SPI
۷۶	۱۰-۱-۵- مفاهیم CPOL و CPHA در پروتکل SPI
۷۷	۱۰-۲- آزمایش
۷۷	۱۰-۲-۱- برقراری ارتباط بین دو میکروکنترلر با پروتکل SPI
۷۷	۱۰-۳- تنظیمات نرم‌افزاری و کدنویسی
۷۷	۱۰-۴- پرسش‌ها و تمرین‌های برنامه نویسی

آزمایش ۱۱: پروتکل ارتباطی UART

۷۹	۱۱-۱- درسنامه
۷۹	۱۱-۱-۱- تفاوت ارتباط سریال با موازی
۸۰	۱۱-۱-۲- ارتباط سریال آسنکرون (غیرهمگام)

۸۱	۱۱-۱-۳- قوانین ارتباط سریال
۸۳	۱۱-۱-۴- سخت افزار ارتباط سریال
۸۴	۱۱-۲- آزمایش
۸۴	۱۱-۲-۱- برقراری ارتباط بین دو میکروکنترلر
۸۵	۱۱-۲-۲- برقراری ارتباط بین میکروکنترلر و کامپیوتر
۸۶	۱۱-۳- تنظیمات نرم افزاری و برنامه نویسی
۸۸	۱۱-۴- پرسش ها و تمرین های برنامه نویسی

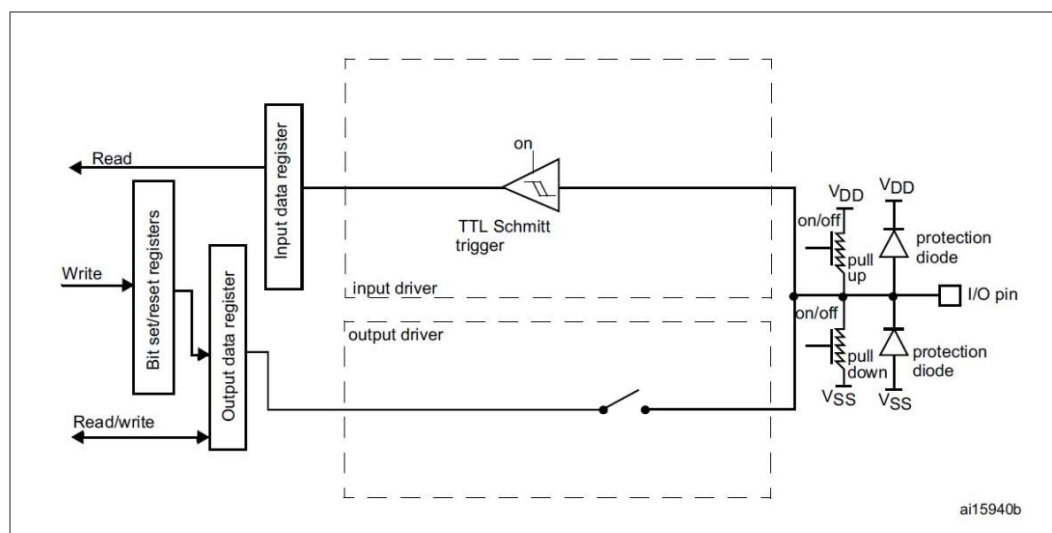
۹۰ **آزمایش ۱۲: پروتکل ارتباطی I²C**

۹۱	۱۲-۱- درسنامه
۹۲	۱۲-۱-۱- عملکرد I ² C
۹۴	۱۲-۲- آزمایش
۹۴	۱۲-۲-۱- درایور نمایشگر کاراکتری با ماژول I ² C
۹۴	۱۲-۳- تنظیمات نرم افزاری و برنامه نویسی
۹۴	۱۲-۴- پرسش ها و تمرین های برنامه نویسی

آزمایش ۱:

کار با پورت‌های ورودی – خروجی عمومی

(GPIO)



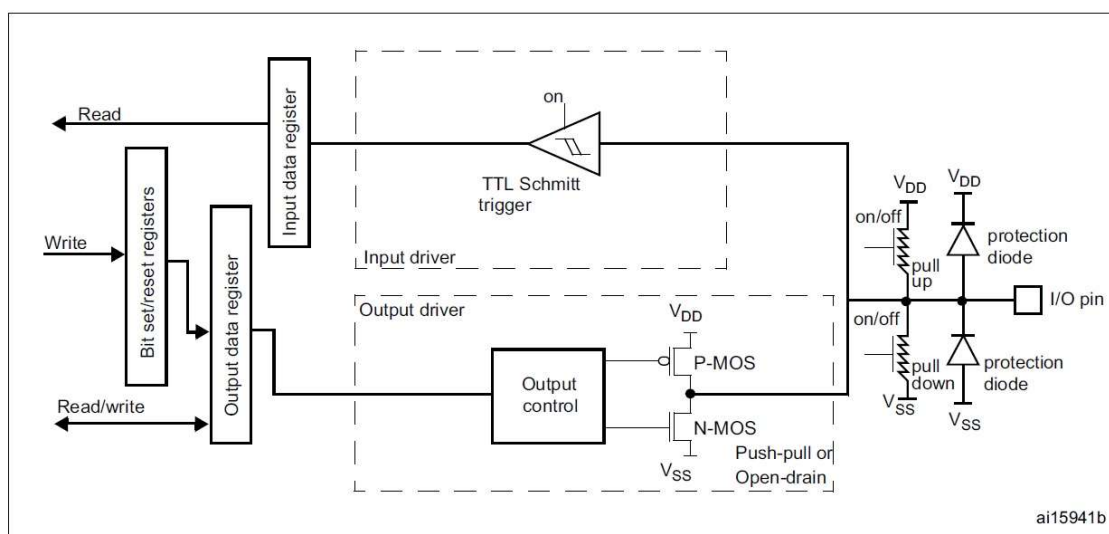
شکل ۱-۱ پیکربندی ورودی

۲-۱-۱- پیکربندی خروجی

زمانی که پین I/O به عنوان خروجی برنامه‌نویسی می‌شود:

- بافر خروجی فعال می‌شود.
- مد OPEN-DRAIN: با نوشتن صفر در رجیستر خروجی NMOS فعال شده و با نوشتن یک در رجیستر خروجی، خروجی امپدانس بالا HI-Z می‌شود. (ترانزیستور PMOS هیچگاه فعال نمی‌شود).
- مد PUSH-PULL: با نوشتن صفر در رجیستر خروجی NMOS فعال شده و با نوشتن یک ترانزیستور PMOS فعال می‌شود.
- اشمیت تریگر ورودی فعال می‌شود.
- مقاومت‌های PULL UP / PULL DOWN داخلی بسته به مقدار رجیستر GPIOx_PUPDR می‌توانند فعال شوند.
- دیتای حاضر روی پین با هر کلاک AHB1 (دقت شود این باس بسته به هر میکروکنترلر می‌تواند متفاوت باشد) نمونه برداری می‌شود.

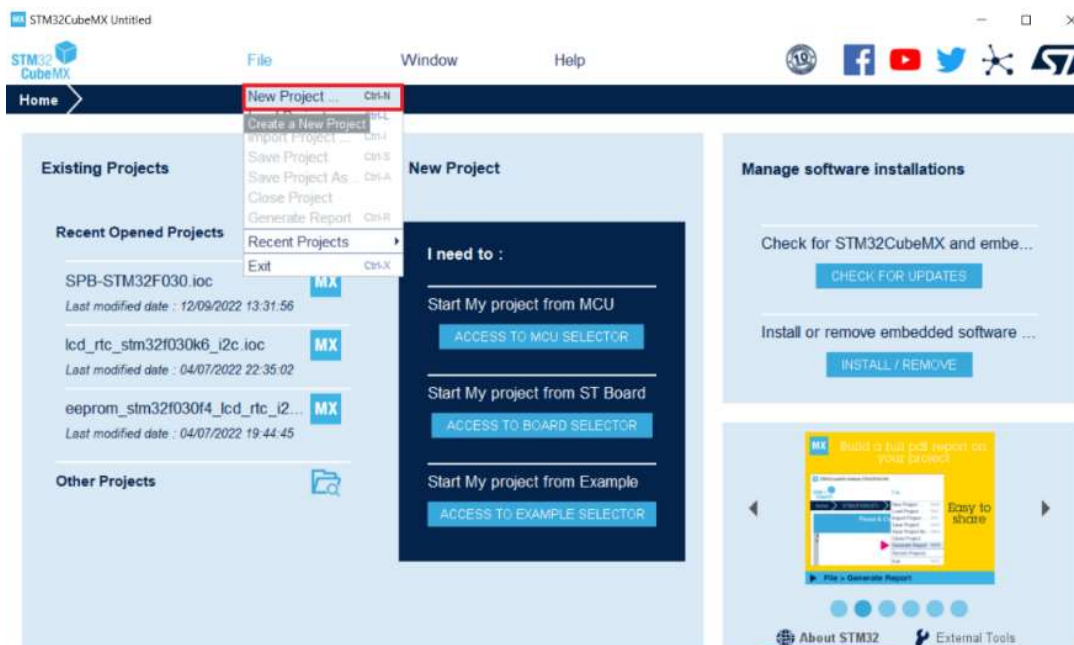
- با مراجعه به رجیستر دیتای ورودی می‌توان این مقدار را خواند.
- با مراجعه به رجیستر دیتای خروجی می‌توان این مقدار را خواند.



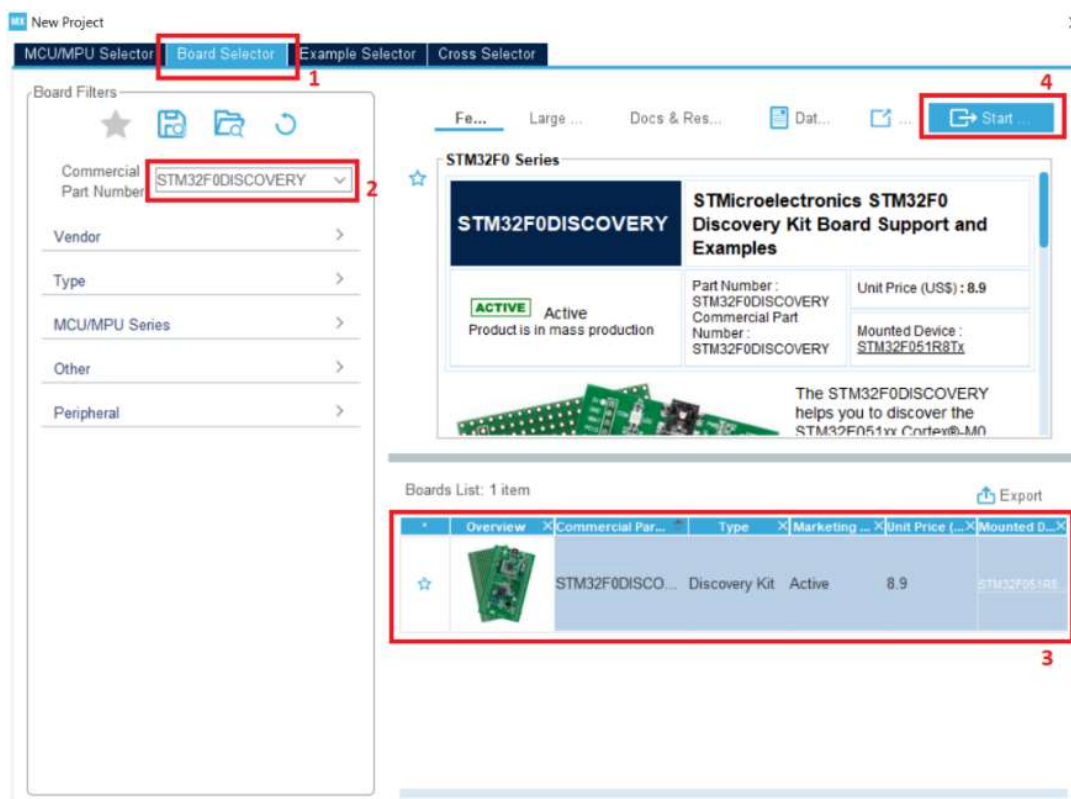
شکل ۱-۲ پیکربندی خروجی

۲-۱- آزمایش

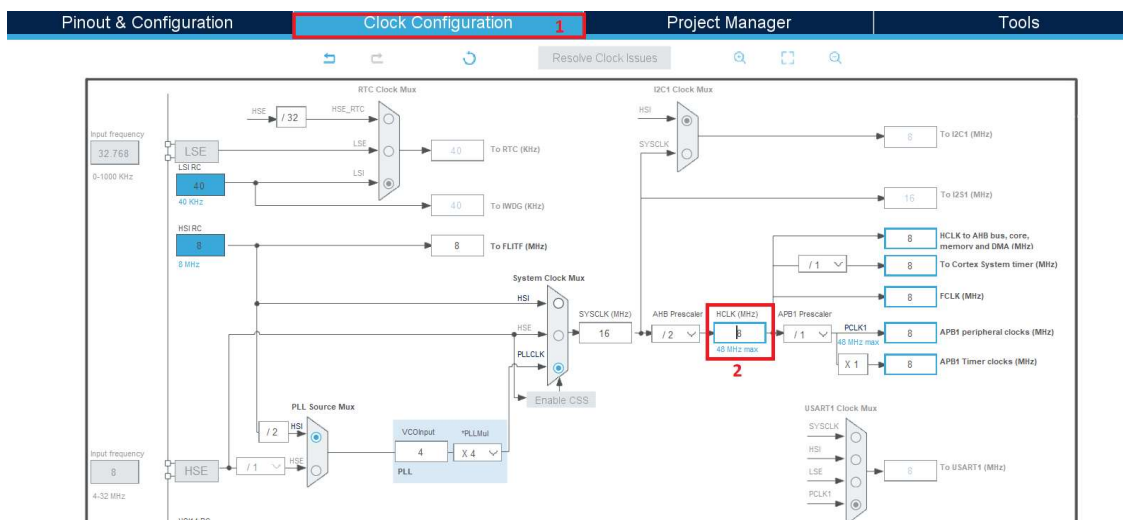
برای تنظیمات اولیه ابتدا یک پروژه جدید برای هر کدام از آزمایش‌ها ایجاد می‌کنیم:



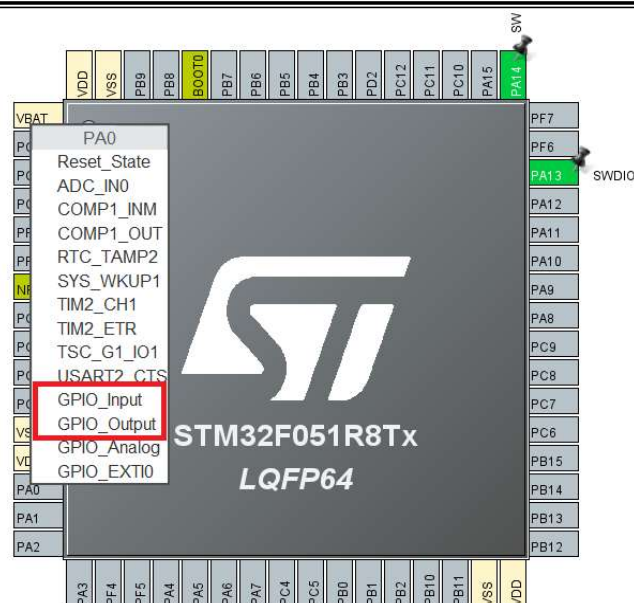
برد یا میکروکنترلر مورد نظر را انتخاب می‌کنیم:



کلاک میکروکنترلر را برابر با مقدار 8MHz قرار می‌دهیم:



روی پایه‌ی مورد نظر کلیک کرده و بر حسب نیاز خروجی یا ورودی را انتخاب می‌کنیم.



برای خواند ورودی می‌توان از تابع زیر استفاده نمود:

```
/* USER CODE BEGIN 2 */
HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0);
/* USER CODE END 2 */
```

برای نوشتن خروجی یا Toggle (تغییر وضعیت) نیز می‌توان از توابع زیر استفاده نمود:

```
/* USER CODE BEGIN 2 */
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_SET);
HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8);
/* USER CODE END 2 */
```

برای ایجاد تاخیر به منظور حذف بانس کلید نیز می‌توان از تابع زیر استفاده کرد:

```
/* USER CODE BEGIN 2 */
HAL_Delay(100);
/* USER CODE END 2 */
```

۱-۲-۱- ال ای دی چشمک زن

برنامه‌ای بنویسید که در آن یک ال ای دی هر ۵۰۰ میلی ثانیه خاموش و روشن شود.

۱-۲-۲- روشن و خاموش کردن ال ای دی به کمک دو Pushbutton

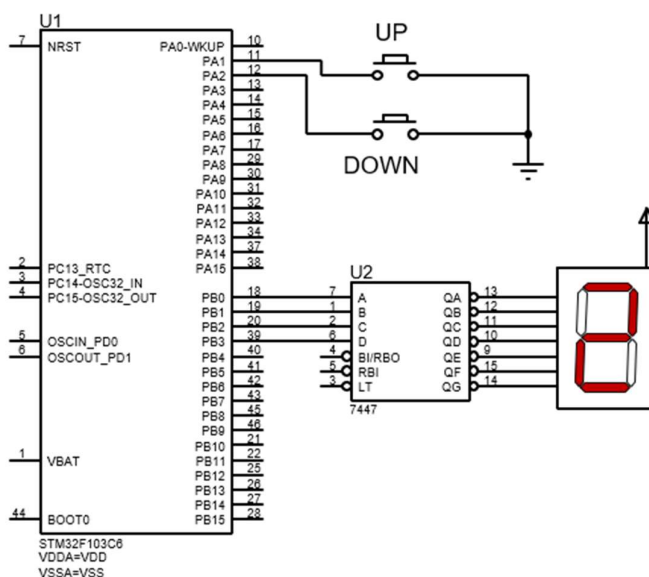
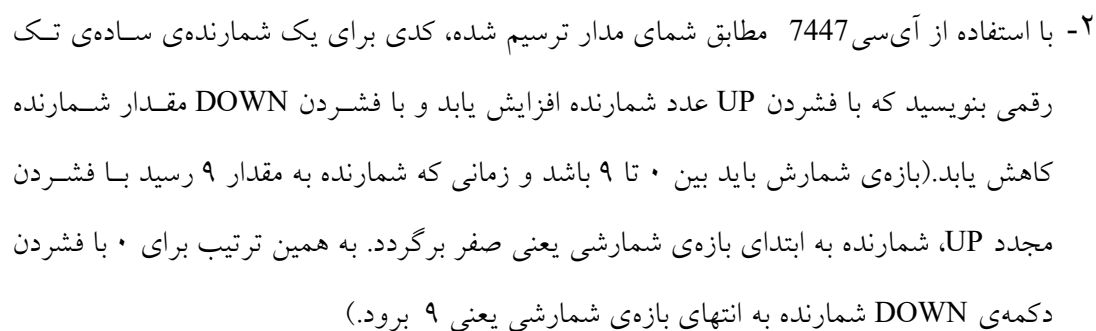
با دو عدد Pushbutton برنامه‌ای بنویسید که با فشردن یک کلید، ال ای دی روشن و با فشردن دیگری خاموش شود.

۱-۳- تنظیمات نرم افزاری و کدنویسی

برای ایجاد پروژه جدید، از نرم افزار CubeIDE وارد قسمت File شده، گزینه‌ی New و سپس STM32 Project را انتخاب کنید. پس از انتخاب میکروکنترلر مورد نظر فایل .ioc به شما نمایش داده می شود که تمام تنظیمات اولیه به کمک همین فایل انجام می شود. دقت کنید که پس از انجام تنظیمات Ctrl+s را زده تا پروژه ذخیره شود. در نهایت شروع به کدنویسی داخل فایل main.c کنید.

۱-۴- پرسش ها و تمرین های برنامه نویسی

۱- برای شمای مدار زیر کدی بنویسید که با تغییر وضعیت هر کلید از دیپ سویچ از OFF به ON و بالعکس LED متناظر آن روشن و خاموش شود.

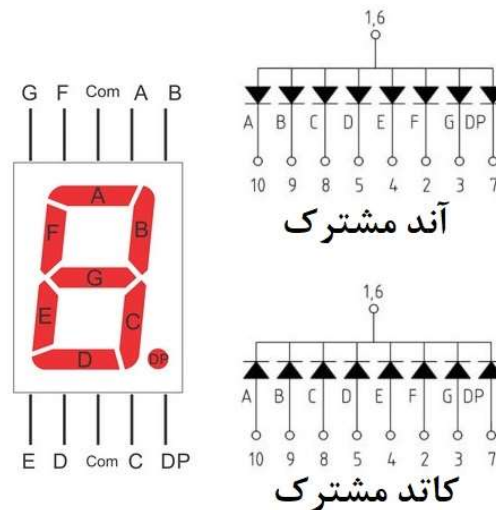


آزمایش ۲:

راه اندازی نمایشگر هفت قطعه‌ای (7-Segment)

۲-۱- درسنامه

برای نمایش اعداد و برخی از حروف انگلیسی در سامانه‌های نهفته، می‌توان از نمایشگرهای هفت قسمتی (7Segment) استفاده نمود. سون سگمنت‌های یکی از پرکاربردترین نمایشگرها در صنعت هستند. یک سون سگمنت شامل هفت عدد LED نمایش ارقام و حروف و همچنین یک عدد LED برای نمایش نقطه ممیز اعداد اعشاری است. سون سگمنت‌ها در دو نوع کاتدمشترک و آندمشترک ساخته می‌شوند. در سون سگمنت‌های آند مشترک، تمام پایه‌های آند دیودها را به هم وصل می‌کنند و در نوع کاتد مشترک پایه‌های کاتد دیودها به هم متصل هستند.



شکل ۲-۱ نمایشگر هفت قطعه‌ای

مثال: برای نمایش عدد 7 روی یک سون سگمنت کاتد مشترک، پایه‌ی COM باید به زمین متصل بوده و LED های a, b و c مقدار High یا 3.3v داشته باشند و سایر LED ها خاموش باشند.

مثال: برنامه‌ای بنویسید که یک سون سگمنت کاتد مشترک را راه‌اندازی کرده و در فاصله‌ی زمانی تقریبی 500 ms اعداد صفر تا نه را شمارش کند.

آند مشترک			کاتد مشترک		
Number	g f e d c b a	Hex code	Number	g f e d c b a	Hex Code
0	1000000	C0	0	0111111	3F
1	1111001	F9	1	0000110	06
2	0100100	A4	2	1011011	5B
3	0110000	B0	3	1001111	4F
4	0011001	99	4	1100110	66
5	0010010	92	5	1101101	6D
6	0000010	82	6	1111101	7D
7	1111000	F8	7	0000111	07
8	0000000	80	8	1111111	7F
9	0010000	90	9	1001111	4F

۱-۱-۲- روش مالتی پلکس

عیب اصلی سون سگمنت‌ها در این است که با افزایش تعداد ارقام و استفاده از چندین سون سگمنت تعدادی زیادی از پین‌های میکروکنترلر اشغال می‌شود. برای رفع این مشکل پایه‌های a, b, c, ..., f, DP همه‌ی سون سگمنت‌ها را به یکدیگر متصل نموده و پایه‌های مشترک آن‌ها را کنترل می‌کنیم. به طور مثال اگر چهار عدد سون سگمنت کاتد مشترک در اختیار داشته باشیم و بخواهیم ال‌ای دی a را در سون سگمنت دوم روشن کنیم، باید پایه‌های کاتد سون سگمنت‌های یک، سه و چهار غیرفعال باشند (در اینجا به دلیل کاتد مشترک بودن High) و پایه‌ی کاتد سون سگمنت دوم فعال باشد. به این ترتیب ال‌ای دی دلخواه روی سون سگمنت دلخواه روشن می‌شود.

مثال: می‌خواهیم عدد ۷۴۵۸ را در یک سون سگمنت چهارتایی نشان دهیم.

مطابق مراحل زیر پیش می‌رویم:

سون سگمنت اول را روشن می‌کنیم.

عدد ۷ را روی پایه‌های a تا g بار گذاری می‌کنیم.

یک تاخیر حدوداً ۲۰ میلی ثانیه‌ای ایجاد می‌کنیم.

سون سگمنت اول را خاموش و سون سگمنت دوم را روشن می‌کنیم.

عدد ۴ را روی پایه‌های a تا g بار گذاری می‌کنیم.
 یک تاخیر حدوداً ۲۰ میلی ثانیه‌ای ایجاد می‌کنیم.
 سون سگمنت دوم را خاموش و سون سگمنت سوم را روشن می‌کنیم.
 عدد ۵ را روی پایه‌های a تا g بار گذاری می‌کنیم.
 یک تاخیر حدوداً ۲۰ میلی ثانیه‌ای ایجاد می‌کنیم.
 سون سگمنت سوم را خاموش و سون سگمنت چهارم را روشن می‌کنیم.
 عدد ۸ را روی پایه‌های a تا g بار گذاری می‌کنیم.
 یک تاخیر حدوداً ۲۰ میلی ثانیه‌ای ایجاد می‌کنیم.
 و این حلقه باید دائماً تکرار شود.

۲-۲- آزمایش

۲-۲-۱- شمارنده‌ی تک رقمی ۰ تا ۹

برنامه‌ای بنویسید که اعداد ۰ تا ۹ را به فاصله‌ی ۵۰۰ میلی ثانیه روی سون سگمنت شمارش کند.

۲-۲-۲- شمارنده‌ی تک رقمی ۰ تا ۹۹

با استفاده از یک سون سگمنت دو رقمی و روش مالتی پلکسینگ برنامه‌ای بنویسید که اعداد ۰ تا ۹۹ را به فاصله‌ی ۵۰۰ میلی ثانیه شمارش کند.

۲-۳- تنظیمات نرم افزاری و کدنویسی

همانند آزمایش جلسه‌ی قبل یک پروژه بسازید و برای روشن نمودن سون سگمنت از تابع

HAL_GPIO_WritePin استفاده کنید.

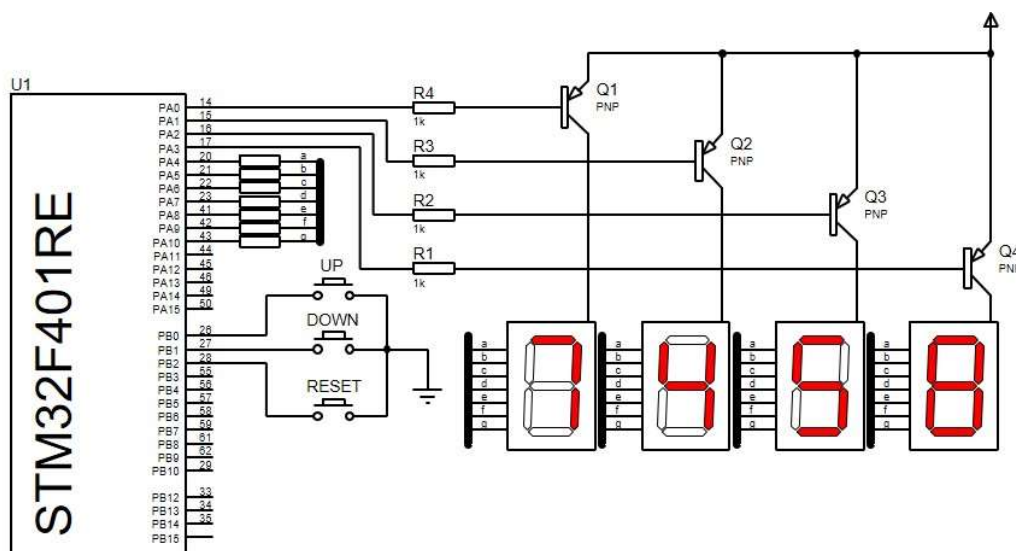
۴-۲- پرسش‌ها و تمرین‌های برنامه نویسی

۱- با استفاده از سه عدد Push Button برای کلیدهای افزایش، کاهش و ریست و همچنین یک

7segment چهارتایی، با استفاده از روش مالتی پلکسینگ شمارنده‌ای چهار رقمی از ۰ تا ۹۹۹۹

بسازید.

- با فشردن کلید افزایش مقدار شمارنده یک عدد افزایش می‌یابد.
- با فشردن کلید کاهش مقدار شمارنده یک عدد کاهش می‌یابد.
- با فشردن کلید ریست شمارنده صفر می‌شود.

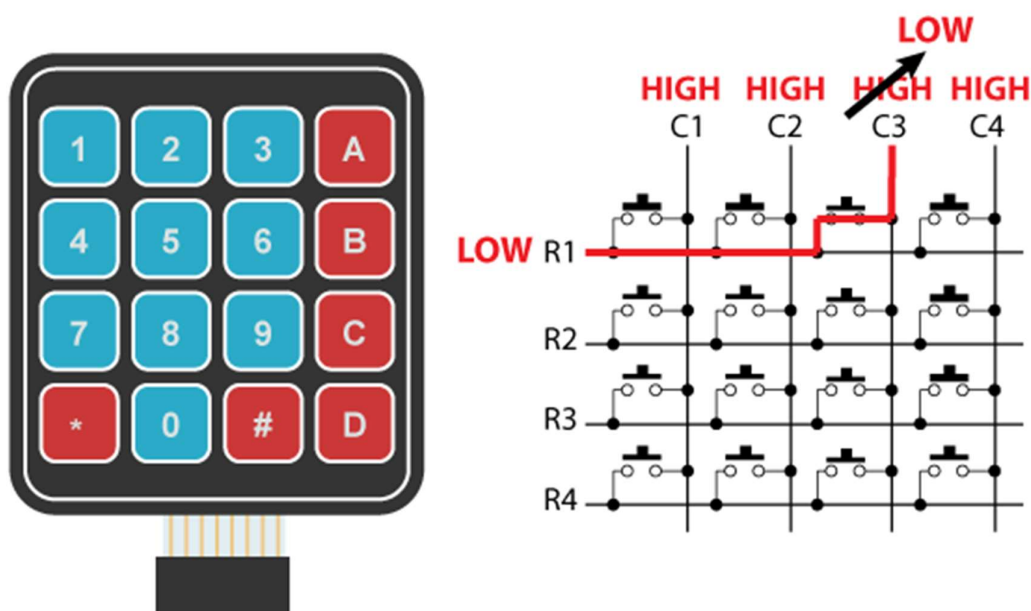


آزمایش ۳:

راه اندازی صفحه کلید ماتریسی (Keypad)

۳-۱- درسنامه

صفحه کلید ماتریسی یکی از پرکاربردترین قطعات مورد استفاده در مدارهای مبتنی بر میکروکنترلر است. به کمک این قطعه می توان اطلاعات را از دنیای بیرون به میکروکنترلر منتقل نمود. صفحه کلید ماتریسی متشکل از تعدادی دکمه فشاری (Push Button) است که در قالب سطرها و ستون ها کنار هم قرار گرفته اند. مزیت استفاده از صفحه کلید ماتریسی به جای استفاده از دکمه های فشاری به صورت تکی، کاهش پین های اشغال شده از میکروکنترلر است.



با اتصال سطرها و ستون ها به میکروکنترلر، برای خواندن کلید فشرده شده باید از میان سطرها و ستون ها یکی را ورودی و دیگری را خروجی تعریف کنیم. فرض شود سطرها ورودی تعریف و Pull-Up شده اند. همچنین تمام ستون ها خروجی و با مقدار اولیه ی High می باشند. میکروکنترلر در یک حلقه باید تک تک ستون ها را از حالت High به Low ببرد و ورودی را بخواند. در صورتی که ورودی تغییر مقدار دهد، آنگاه میکروکنترلر نتیجه می گیرد که کلید متناظر با آن سطر و ستون فشرده شده است.

مثال: در شکل بالا زمانی که کاربر کلید شماره ی ۳ را بفشارد و میکرو در حلقه ی ورودی کلید باشد، ستون سه و سطر یک به هم متصل شده اند. زمانی که میکرو مقدار ستون ۳ را به Low تغییر می دهد و ورودی را

می خواند متوجه می شود که سطر ۱ نیز Low شده است و نتیجه می گیرد که کلید سطر ۱ و ستون ۳ فشرده شده است (یعنی کلید 3).

۳-۲- آزمایش

۳-۲-۱- راه اندازی صفحه کلید ماتریسی

برنامه ای برای صفحه کلید ماتریسی 4×3 بنویسید که کلید فشرده شده از Keypad روی سون سگمنت نمایش داده شود.

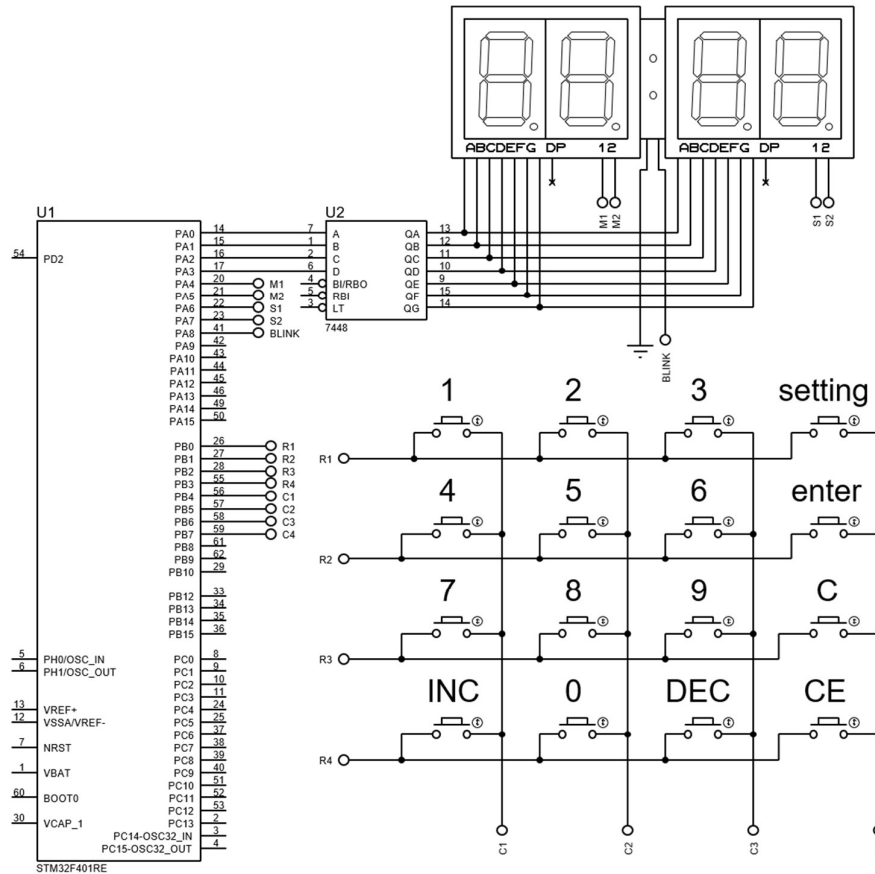
۳-۳- تنظیمات نرم افزاری و کدنویسی

در این آزمایش نیاز است تا از درسنامه کمک گرفته و درایوری برای صفحه کلید ماتریسی بنویسید. دقت شود که می بایست دو فایل با پسوند c و h ایجاد کرده و پس از تکمیل درایور این فایل ها را به پوشه ی پروژه ی خود بیافزایید تا بتوانید از توابعی که نوشته اید داخل فایل main.c استفاده کنید.

۳-۴- پرسش ها و تمرین های برنامه نویسی

- ۱- برای شماتیک شکل زیر برنامه ای بنویسید با این مشخصات:
 - با فشردن کلید setting وارد برنامه شده و برنامه آماده ی دریافت دو عدد دو رقمی برای ثانیه و دقیقه باشد. (محدودیتی برای این دو عدد وجود ندارد).
 - ابتدا کاربر عدد دقیقه را وارد کند با فشردن کلید enter عدد دقیقه ذخیره شده و کاربر عدد ثانیه را وارد می کند.
 - بعد از وارد کردن عدد ثانیه و فشردن کلید enter این عدد نیز ذخیره و از setting خارج شده و

هر دو عدد ثانیه و دقیقه به روش ماتری پلکس نمایش داده می شوند و هم چنین led دو تایی بین آن ها به صورت چشمک زن خاموش و روشن می شود. (نیازی نیست تا اعداد ثانیه و دقیقه با گذشت زمان تغییر کنند و تنها لازم است تا ال ای دی میان آن ها با زمان تقریبی بین ۲۰۰ تا ۶۰۰ میلی ثانیه خاموش و روشن شود.)



در setting دکمه ها باید بدین گونه عمل کنند:

- ۱- دکمه ی setting: تنها در ابتدای ورود به تنظیمات کار کند و در سایر موارد عملکردی نداشته باشد.
- ۲- دکمه ی enter: پس از وارد کردن هر رقم با فشردن این دکمه آن رقم ذخیره شده و برنامه آماده دریافت رقم بعدی شود.
- ۳- کلید C: کل ارقام وارد شده را پاک کند و برنامه منتظر بماند تا کاربر دوباره ارقام را از اول وارد کند.
- ۴- کلید CE: آخرین رقمی که کاربر وارد کرده را پاک کند و منتظر دریافت یک عدد جدید باشد. مثلن اگر کاربر در حال وارد کردن رقم دهگان ثانیه شمار است با فشردن کلید CE باید به رقم یکان دقیقه شمار برود و آن را از اول وارد کند.

۵- کلیدهای DEC و INC رقم وارده را می توانند قبل از فشردن کلید enter کاهش یا افزایش دهند. مثلاً فرض کنید در حال ورود عدد یکان ثانیه شمار هستیم و کلید ۸ را فشرده ایم اما هنوز enter را نزده ایم با فشردن کلید INC عدد به ۹ تغییر پیدا می کند و با فشردن کلید DEC عدد ۷ می شود.

آزمایش ۴:

وقفه‌ی خارجی در میکروکنترلرهای STM32

۴-۱- درسنامه

وقفه یک سیگنال به ریزپردازنده است که به توجه و پاسخ سریع پروسسور نیاز دارد. هنگامی که یک وقفه رخ می‌دهد، پردازنده عملیات جاری خود را متوقف می‌کند تا به درخواست وقفه رسیدگی کند. مثال: خود را به عنوان میزبان (پروسسور) در نظر بگیرید که منتظر مهمان (دستورالعمل) هستید. در صورتی که زنگ در وجود نداشت برای این که بدانید مهمان آمده است یا خیر، مدام باید از پنجره سرکشی (Polling) کنید تا در صورت وجود مهمان در را برایش باز کنید. اما در صورتی که زنگ در یا آیفون به درستی کار کند، لازم نیست که میزبان دائماً وجود مهمان را چک کند به محض این که مهمان زنگ در را بزند (Interrupt)، میزبان از وجود وی مطلع شده و در را برایش باز می‌کند.

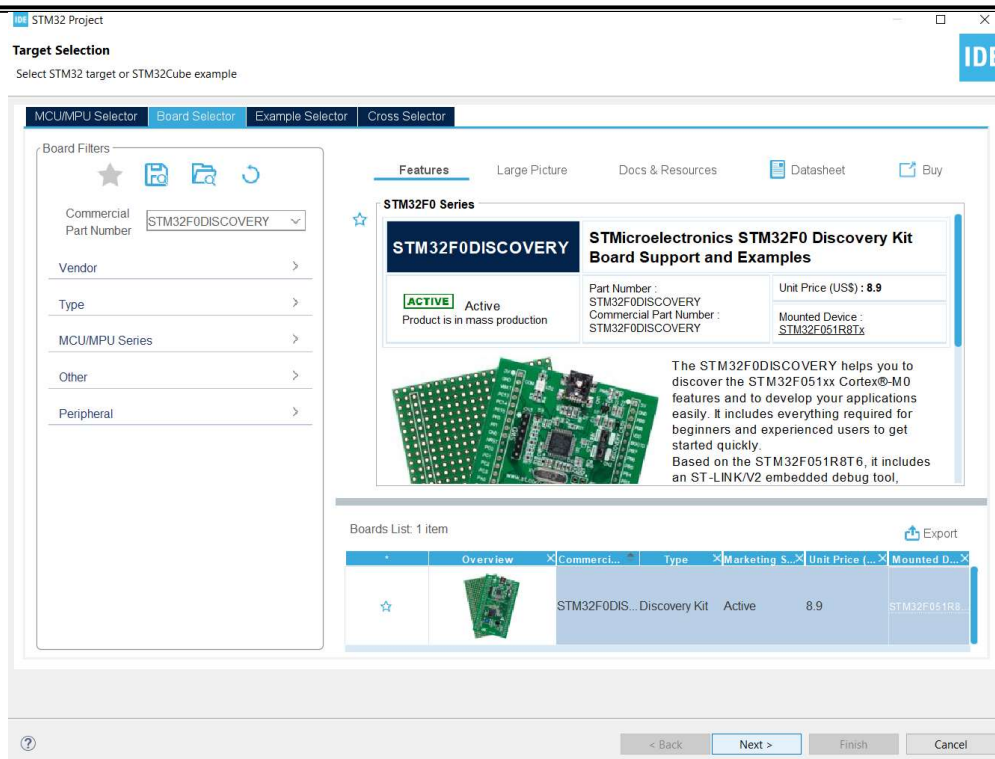
۴-۲- آزمایش

۴-۲-۱- تغییر وضعیت LED به کمک وقفه‌ی خارجی

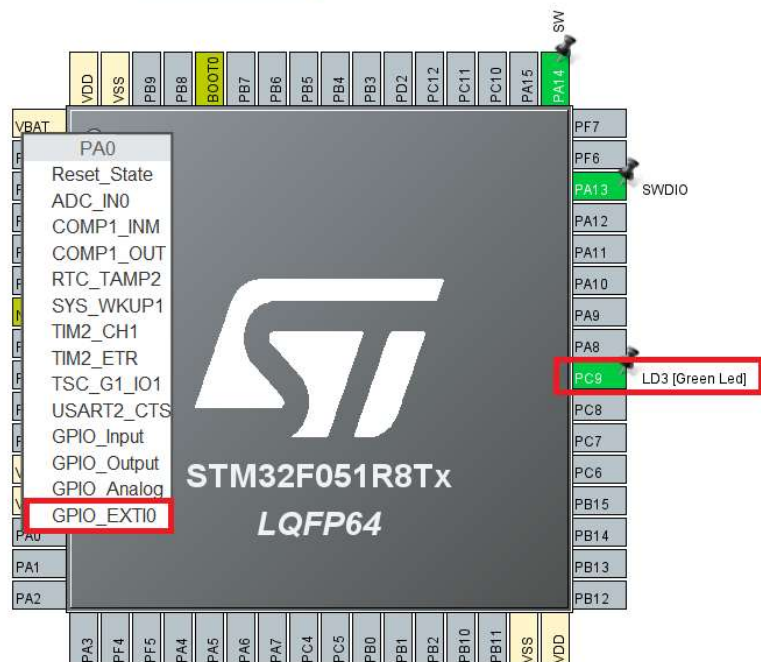
با استفاده از وقفه‌های خارجی یک پین دلخواه از میکروکنترلر را به نحوی پیکره بندی کنید که وقفه‌ی خارجی برای آن فعال شود. سپس یک Push-button به این پایه متصل کنید که با هر بار فشردن کلید یک LED از وضعیت روشن به خاموش و بالعکس تغییر وضعیت دهد. دقت شود از وقفه استفاده کنید نه (Polling).

۴-۳- تنظیمات نرم‌افزاری و کدنویسی

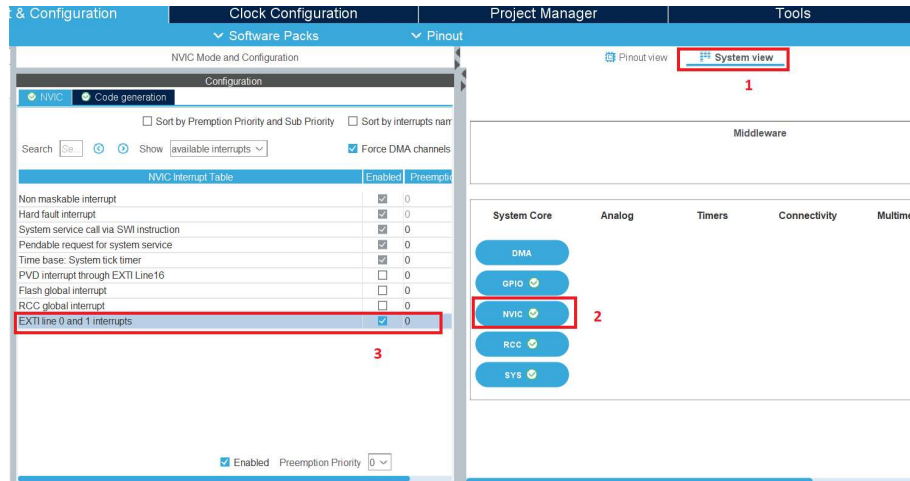
- ابتدا وارد نرم افزار CubeMX شده از قسمت File-New Project میکروکنترلر یا برد موردنظر خود را انتخاب کنید و یک پروژه بسازید. (STM32F0DISCOVERY)



- روی یک پین برای پوش باتن کلیک کنید و GPIO_EXTI را انتخاب کنید. روی پین دیگری کلیک کرده و برای GPIO_OUTPUT LED را انتخاب کنید.



- از بخش NVIC system view را باز کنید و تیک EXTI_LINE_X_INTERRUPT را بزنید.
(متناسب با شماره پایه برای پوش باتن).
- از بخش system view، قسمت GPIO تنظیمات مناسب را برای پایه‌ها انجام دهید. (حساس به لبه ی بالارونده یا پایین رونده مقاومت پول آپ یا پول داون و ... برای پوش باتن)



- پروژه ایجاد شده توسط Cube MX را به Cube IDE وارد کنید.
- در قسمت کتابخانه ی HAL فایل GPIO را باز نموده و تابع Callback را به main.c اضافه کنید. (weak_ حذف شود)

```

/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(GPIO_Pin);

    /* NOTE: This function should not be modified, when the callback is
    needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the
    user file
    */
}
/* USER CODE END 4 */

```

- داخل تابع Callback مقدار ولتاژ led را با هر بار رخ دادن وقفه تغییر دهید.

۴-۴- پرسش‌ها و تمرین‌های برنامه نویسی

- ۱- با استفاده از مفهوم وقفه‌ی خارجی دو عدد Pushbutton را به نحوی راه‌اندازی کنید که با فشردن یکی led روشن و با فشردن دیگری led خاموش شود.

آزمایش ۵:

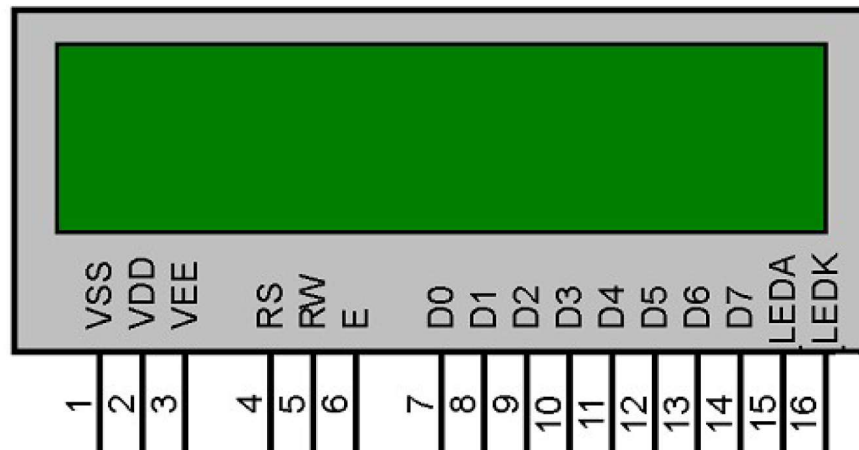
راه اندازی نمایشگر کاراکتری

۱-۵- درسنامه

نمایشگرهای کاراکتری همان گونه که از نامشان پیداست برای نمایش کاراکترهای اسکی مناسب هستند. این نمایشگرها می توانند چندین سطر و ستون داشته باشند و دارای نور پس زمینه سبز یا آبی هستند. ترکیبات زیادی مانند 8×1 ، 8×2 ، 10×2 و 16×1 وجود دارد. نمایشگر استفاده شده در این آزمایشگاه دارای دو سطر و شانزده ستون می باشد. تراشه ی کنترلر این نمایشگر HD44780 نام دارد که با مراجعه به دیتاشیت این قطعه میتوان به نحوه راه اندازی نمایشگر کاراکتری پی برد.



تمامی نمایشگرهای کاراکتری فوق دارای ۱۶ پین هستند و رویکرد برنامه نویسی نیز برای آنها یکسان است. در ادامه توضیحات پینهای نمایشگر 16×2 آمده است:



شماره پایه	نام پایه	نوع عملکرد	تفسیر عملکرد	اتصال
Pin 1	Ground	Source Pin	پایه ی زمین نمایشگر	Connected to the ground of the MCU/ Power source
Pin 2	VCC	Source Pin	پایه ی ولتاژ تغذیه	Connected to the supply pin of Power source
Pin 3	V0/VEE	Control Pin	تنظیم کنتراست	Connected to a variable POT that can source 0-5V
Pin 4	Register Select	Control Pin	انتخاب بین دستور و داده	Connected to a MCU pin and gets either 0 or 1. 0 -> Command Mode 1-> Data Mode
Pin 5	Read/Write	Control Pin	انتخاب بین عملیات نوشتن یا خواندن	Connected to a MCU pin and gets either 0 or 1. 0 -> Write Operation 1-> Read Operation
Pin 6	Enable	Control Pin	برای انجام خواندن و یا نوشتن سطح ولتاژ این پایه باید بالا باشد	Connected to MCU and always held high.
Pin 7-14	Data Bits (0-7)	Data/Command Pin	مجموعه ای از پایه ها برای ارسال دستور یا داده	<u>In 4-Wire Mode</u> Only 4 pins (0-3) is connected to MCU <u>In 8-Wire Mode</u> All 8 pins(0-7) are connected to MCU
Pin 15	LED Positive	LED Pin	پایه مثبت نور پس زمینه	Connected to +5V
Pin 16	LED Negative	LED Pin	پایه منفی نور پس زمینه	Connected to ground

Hex Code Command to LCD Instruction Register

0F	LCD ON, cursor ON
01	Clear display screen
02	Return home
04	Decrement cursor (shift cursor to left)
06	Increment cursor (shift cursor to right)
05	Shift display right
07	Shift display left
0E	Display ON, cursor blinking
80	Force cursor to beginning of first line
C0	Force cursor to beginning of second line
38	2 lines and 5x7 matrix
83	Cursor line 1 position 3
3C	Activate second line
08	Display OFF, cursor OFF
0C	Display ON, cursor OFF
C1	Jump to second line, position 1
C2	Jump to second line, position 2

```

void lcd_cmd(uint8_t cmd)
{
    RS_LOW;
    EN_LOW;

    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, (cmd>>0)&0x01);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_3, (cmd>>1)&0x01);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_4, (cmd>>2)&0x01);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_5, (cmd>>3)&0x01);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_6, (cmd>>4)&0x01);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, (cmd>>5)&0x01);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, (cmd>>6)&0x01);
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, (cmd>>7)&0x01);

    HAL_Delay(5);
    EN_HIGH;
    HAL_Delay(1);
    EN_LOW;
    HAL_Delay(5);
}

```

کد تابع دستور مشاهده می شود. دقت شود که تابع داده نیز مانند کد تابع دستور است با این تفاوت که سطح ولتاژ پایه ی RS در تابع داده باید High و در تابع دستور باید Low باشد.

۲-۵- آزمایش

۱-۲-۵- درایورنویسی نمایشگر کاراکتری مد ۴ بیتی

با استفاده از برنامه نویسی رجیستری درایوری برای نمایشگر کاراکتری در مد ۴ بیت بنویسید به نحوی که: دارای توابع init، clear، send_data، send_command، put_string و gotoxy باشد.

- تابع init برای مقدار دهی اولیه باشد.
- تابع clear برای پاک کردن صفحه ی نمایش است.
- توابع send_data و send_command به ترتیب دستور و داده را به نمایشگر ارسال می کنند.
- تابع gotoxy نشانگر نمایشگر را به موقعیت (x,y) می برد تا بتوان در آن نقطه کاراکتری چاپ نمود.

- تابع `put_string` باید سه ورودی داشته باشد یک ورودی رشته ای برای چاپ باشد و دو ورودی دیگر موقعیت سطر و ستون در نمایشگر تا به نقطه مورد نظر برود و یک رشته را چاپ کند.

۲-۲-۵- استفاده از نمایشگر

با استفاده از درایوری که برای نمایشگر گرافیکی در اختیار دارید (مد ۸ بیت یا ۴ بیت تفاوتی نمی کند) و دو عدد پوش باتن برنامه ای بنویسید به نحوی که:

- ابتدا رشته ی `welcome` را در ابتدای سطر اول نمایش دهد.
- دو ثانیه صبر و سپس صفحه نمایش گر پاک شود.
- متغیری با نام `count` داخل برنامه ایجاد کنید با مقدار اولیه صفر.
- مقدار `count` روی نمایشگر به نمایش در آید. (مقدار این متغیر بین ۱۲۰ تا تغییر کند)
- با فشردن یکی از کلیدها این مقدار افزایش و با فشردن کلید دیگر این مقدار کاهش یابد و روی نمایشگر مقدار جدید نمایش داده شود.
- برنامه را به نحوی بنویسید که با توجه به تغییر `count` از یک تا سه رقم ، جای این عدد همیشه ثابت باشد به چپ و راست جابه جا نشود.

سوال: با توجه به مقدار حداقل و حداکثر متغیر `count` از چه نوع داده ای باید بهره برد و `Format specifier` متناسب آن چیست؟

۳-۵- پرسش ها و تمرین های برنامه نویسی

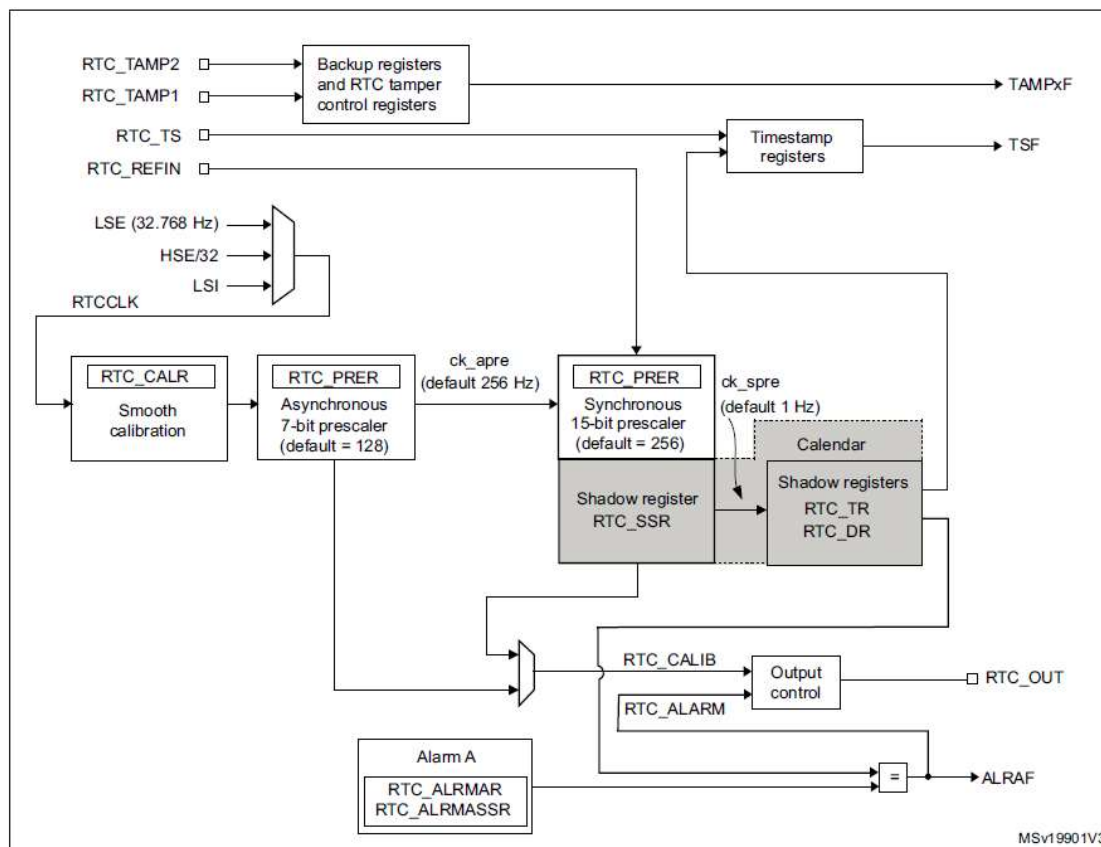
- ۱- نام یا نام خانوادگی خود را به زبان فارسی روی نمایشگر کاراکتری ۱۶۲ نمایش دهید. (در صورتی که هم نام و هم نام خانوادگی شما بیش از ۸ کاراکتر است می توانید کلمه ی دیگری که بین ۴ تا ۸ کاراکتر داشته باشد انتخاب کنید

آزمایش ۶:

کار با واحد زمان سنج حقیقی (RTC)

۱-۶- درسنامه

زمان پارامتر مهمی است که در اکثر سامانه‌های میکروکنترلری از اهمیت ویژه‌ای برخوردار است. در میکروکنترلرهای خانواده st نیز برای نگهداری و شمارش زمان واقعی واحدی به نام RTC تعبیه شده است. این بخش تراشه‌ای مجهز به شمارنده‌ی ۳۲ بیتی است که می‌تواند از کریستال خارجی به خصوصی که برای این منظور تعبیه شده است به عنوان منبع کلاک خارجی بهره‌برد. از آن جایی که بایستی زمان شمارش شده وقتی که تغذیه اصلی تراشه قطع می‌شود به کار خود ادامه دهد، این بخش به باتری پشتیبانی می‌تواند مجهز شود که برای این منظور طراحی شده است. کلاک بخش RTC ابتدا توسط Prescaler تقسیم می‌شود. معمولاً مقدار این پیش تقسیم‌کننده طوری تنظیم می‌شود که پالس یک هرتز به واحد RTC برسد.



شکل ۶-۱ بلوک دیاگرام RTC در STM32F051xxxx

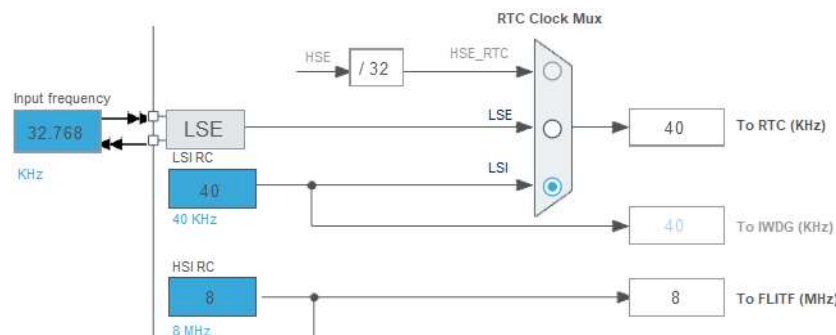
۶-۲- آزمایش

۶-۲-۱- ساعت دیجیتال

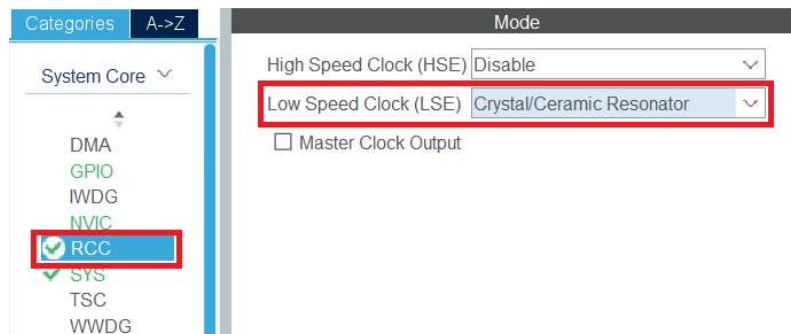
با کمک نمایشگر کاراکتری و بخش RTC میکروکنترلر یک ساعت دیجیتال بسازید که ساعت، دقیقه و ثانیه را در سطر اول نمایشگر و آلارم را در سطر دوم آن نمایش دهد. با تنظیم آلارم A و برابر شدن زمان با این آلارم یک ال ای دی روشن شود.

۶-۳- تنظیمات نرم افزاری و کدنویسی

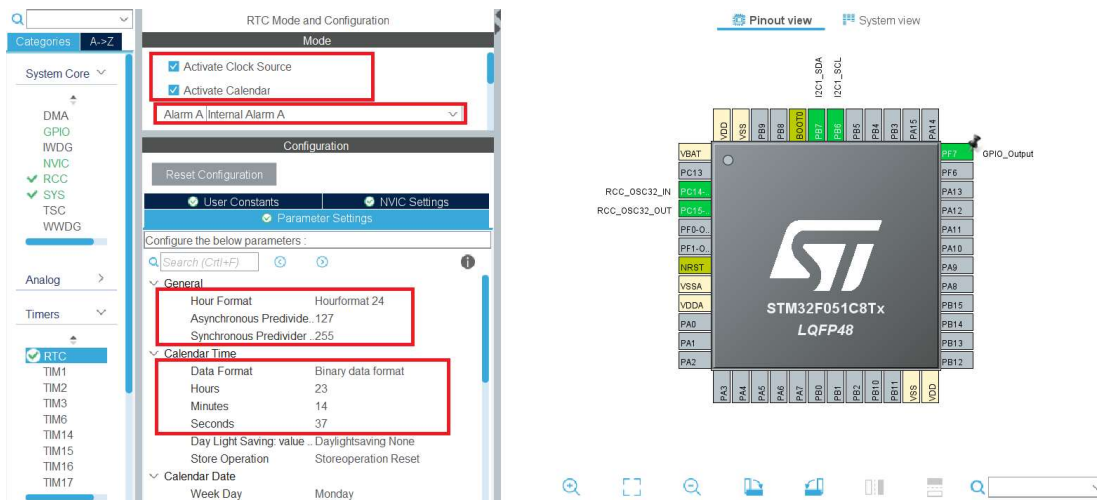
بردی که در اختیار داریم (STM32F0DISCOVERY) فاقد کریستال 32.768KHz است پس ما باید از منبع کلاک داخلی میکرو که مقدار 40KHz را دارد استفاده کنیم:



در صورتی که امکان تغذیه RTC از کریستال خارجی فراهم بود منبع کلاک را LSE انتخاب می کنیم اما ابتدا باید از بخش RCC دسترسی به این منبع را ممکن کنیم:



در ادامه:



Alarm A

Hours 23
Minutes 14
Seconds 45
Sub Seconds 0

Alarm Mask Date Week . Enable
Alarm Mask Hours Disable
Alarm Mask Minutes Disable
Alarm Mask Seconds Disable
Alarm Sub Second Mask All Alarm SS fields are masked.
Alarm Date Week Day ... Date
Alarm Date 1

وقفه‌ی آلارم را فعال می‌کنیم:



برای خواند زمان، تاریخ و آلارم به سه استراکچر نیاز داریم:

```
/* USER CODE BEGIN PV */

RTC_TimeTypeDef gTime = {0};
RTC_DateTypeDef gDate = {0};
RTC_AlarmTypeDef gAlarm = {0};

/* USER CODE END PV */
```

داخل حلقه زمان و تاریخ را می‌خوانیم:

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    HAL_RTC_GetTime(&hrtc, &gTime, RTC_FORMAT_BIN);
    HAL_RTC_GetDate(&hrtc, &gDate, RTC_FORMAT_BIN);
    HAL_RTC_GetAlarm(&hrtc, &gAlarm, RTC_ALARM_A, RTC_FORMAT_BIN);
    sprintf(buffer_1, "Time:
%02u:%02u:%02u", gTime.Hours, gTime.Minutes, gTime.Seconds);
    lcd_send_string(buffer_1, 0, 0);
    sprintf(buffer_1, "Alarm:
%02u:%02u:%02u", gAlarm.AlarmTime.Hours, gAlarm.AlarmTime.Minutes, gAlarm
.AlarmTime.Seconds);
    lcd_send_string(buffer_1, 0, 1);
    HAL_Delay(200);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
```

فراموش نشود که تابع Callback مربوط به وقفه‌ی آلارم را به main.c اضافه کنید:

```
/* USER CODE BEGIN 4 */
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(hrtc);

    /* NOTE : This function should not be modified, when the callback is
needed,
the HAL_RTC_AlarmAEventCallback could be implemented in
the user file
*/
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_7, GPIO_PIN_SET);
}
/* USER CODE END 4 */
```

۴-۶- پرسش‌ها و تمرین‌های برنامه نویسی

۱- با افزودن کپید به این آزمایش، زمان و آلارم باید توسط کاربر قابل تنظیم باشد.

آزمایش ۷:

کار با واحد زمان سنج ها و شمارنده ها

(Timer/Counter) – بخش اول

۷-۱- در سنامه

به جرئت می توان گفت که بحث Timer در میکروکنترلرها و مشخصاً در میکروکنترلرهای STM32، گسترده ترین مبحث است. به علاوه Timer از آن مواردی است که تقریباً در تمامی پروژه ها به وفور استفاده می شود و اهمیت بسیار مهمی دارد.

۷-۱-۱- زمان سنج چیست و چگونه ساخته می شود؟

به عنوان یک تعریف ساده و کلی می گوئیم که Timer ابزاری است که زمان را برای ما می سنجد، و سنجش زمان می تواند با استفاده از سیستم ها و روش های مختلفی ساخته شود. مثلاً زمان می تواند با استفاده از یک سیستم مکانیکی، پنوماتیکی و یا الکترونیکی ساخته شود. اما روش ساختی که در این آزمایش مدنظر ما است، روش الکترونیکی و مشخصاً الکترونیک دیجیتال است. ساعت مچی نمونه ی خوبی از یک سیستم الکترونیکی-مکانیکی که زمان را برای ما می سنجد و معمولاً مبنای عملکرد آن بر اساس یک کریستال کوآرتز با فرکانس ۳۲.۷۶۸ کیلوهرتز است.

در میکروکنترلر هم معمولاً با استفاده از کریستال یا یک روش مشابه دیگر، یک سیگنال کلاک تولید می شود و ما با استفاده از همین سیگنال کلاک، زمان را می سنجیم. در این آزمایش با اینکه سیگنال کلاک چگونه ساخته می شود و سایر مقدمات آن کاری نداریم و فرض می کنیم که یک سیگنال کلاک با فرکانس مشخص داریم و می خواهیم با استفاده از ادوات دیجیتالی که در اختیار داریم، یک Timer بسازیم.

۷-۱-۲- ساخت زمان سنج دیجیتال

قبل از اینکه نحوه ی ساخت Timer دیجیتال را توضیح بدهیم باید با یک مفهوم دیگر به اسم Counter یا شمارنده و ارتباط آن با Timer آشنا بشویم. اصول ساخت Timer بر اساس یک شمارنده است. یعنی ما

برای سنجش زمان، یک شمارنده داریم که عمل شمارش را انجام می‌دهد و برای محاسبه‌ی زمان، باید عدد شمارش شده توسط شمارنده را در مدت زمان هر شمارش ضرب کنیم.

فرض کنید یک شمارنده ۴ بیتی داریم که کلاک این شمارنده از یک منبع کلاک با فرکانس یک کیلوهرتز تأمین می‌شود. این شمارنده ۴ بیتی می‌تواند از عدد ۰ تا ۱۵ را شمارش کند و با توجه به فرکانس، میزان هر شمارش یک میلی ثانیه است. پس با استفاده از یک شمارنده توانستیم تایمری بسازیم که می‌تواند زمان را با پایه زمانی یک میلی ثانیه بسنجد و برای زمان‌های دیگر هم می‌توانیم عدد ۱ تا ۱۶ (با احتساب عدد ۰ به عنوان اولین عدد شمارش) را در یک میلی ثانیه ضرب کنیم.

تا اینجا با Timer و روش ساخت آن با استفاده از کانتر و همچنین با نحوه‌ی عملکردش آشنا شدیم. در ادامه می‌خواهیم Timer در میکروکنترلرهای STM32 را بررسی کنیم.

۳-۱-۷- زمان‌سنج‌ها در میکروکنترلرهای STM32

تعداد Timer در میکروکنترلرهای STM32 در مجموع ۱۴ عدد است که این ۱۴ عدد به انواع زیر تقسیم می‌شوند:

- Advanced-control timers
- General-purpose timers
- Basic timers

تایمرهای ۱ و ۸ از نوع Advanced-control، تایمرهای ۲ تا ۵ و ۹ تا ۱۴ از نوع General-purpose و در نهایت تایمرهای ۶ و ۷ از نوع Basic هستند. تفاوت عمده‌ی Timer در میکروکنترلرهای STM32 در امکاناتی که دارند، می‌باشد. به عنوان مثال در تایمرهای Advanced-control، مدهایی برای اینترفیس انکودر و سنسور هال وجود دارد که در سایر تایمرها موجود نیست یا در تایمرهای General-purpose، قابلیت PWM و Input capture وجود دارد که این امکانات در تایمرهای Basic موجود نیست.

قابل ذکر است که با توجه به میکروکنترلر، نوع و تعداد تایمرهای موجود در میکروکنترلر می‌تواند متفاوت باشد. مثلاً در میکروکنترلر STM32F103C8T6، تایمر نوع Basic وجود ندارد. همان‌طور که قبلاً هم گفتیم Timer در میکروکنترلرهای STM32 بسیار گسترده است و دارای قابلیت‌ها و امکانات بسیار زیادی

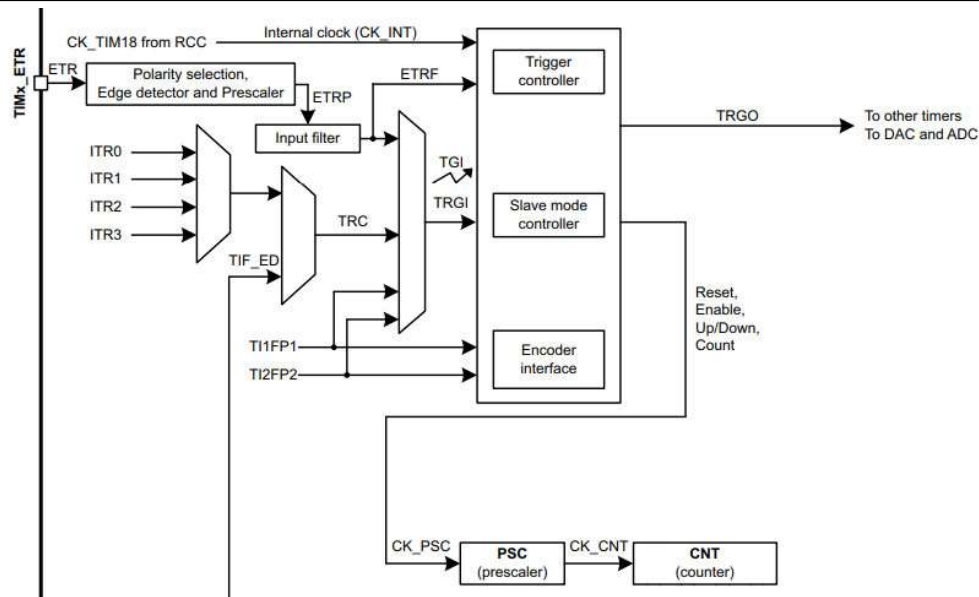
می‌باشد، از همین جهت ما در این آزمایش و آزمایش بعدی، مهم‌ترین قابلیت‌های تایمرها و مواردی که کاربردی‌تر هستند را مورد بررسی قرار خواهیم داد.

Timer در میکروکنترلرهای STM32 دارای یک Counter یا شمارنده ۱۶ بیتی است (در سری‌های متفاوت این عدد تا ۳۲ بیت هم می‌رسد) که این شمارنده می‌تواند به صورت بالا شمار، پایین شمار و بالا-پایین شمار، شمارش کند. همان‌طور که می‌دانید نکته مهم این است که این شمارنده ۱۶ بیتی با چه فرکانسی می‌تواند شمارش کند. حداکثر فرکانس واحد همه‌ی تایمرها در میکروکنترلر مدنظر ما، 48MHz است که از طریق باس‌های مربوطه تأمین می‌شود.

حال ما می‌توانیم این فرکانس 48MHz را مستقیماً به شمارنده ۱۶ بیتی بدهیم و یا اینکه این فرکانس را با استفاده از Prescaler به فرکانس‌های کوچک‌تری تبدیل کرده و سپس آن را به شمارنده بدهیم. تایمر در میکروکنترلرهای STM32 دارای یک Prescaler با طول ۱۶ بیت است که فرکانس ورودی واحد تایمر را به عددی بین ۱ تا ۶۵۵۳۶ تقسیم می‌کند.

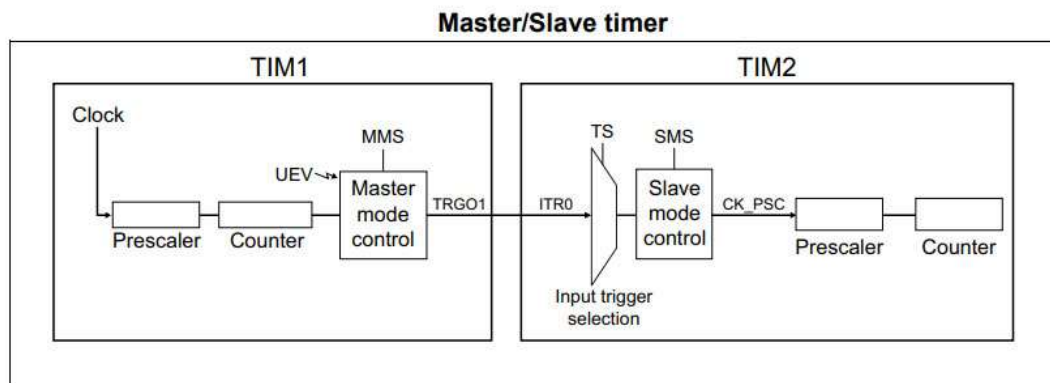
پس ما علاوه بر اینکه با استفاده از Prescaler ها و ضرب‌کننده‌های فرکانسی که قبل از واحد تایمر قرار دارند، می‌توانیم فرکانس ورودی واحد تایمر را تعیین کنیم، با استفاده از Prescaler که در خود واحد تایمر قرار دارد هم این انعطاف را داریم که فرکانس را تا حد بسیار زیادی، و تقریباً به هر عددی که بخواهیم تغییر بدهیم. توجه کنید که هم Prescaler و هم Counter هر دو ۱۶ بیتی هستند و Prescaler دقیقاً قبل از Counter قرار داده شده است و کلاک متصل به Counter، دقیقاً همان کلاکی است که از خروجی Prescaler گرفته می‌شود. کلاک ورودی به واحد Timer در میکروکنترلرهای STM32 می‌تواند از روش‌های مختلفی مانند کلاک داخلی میکروکنترلر، پین‌های خارجی و از طریق یک تایمر دیگر تأمین بشود.

تصویر زیر منابع مختلف کلاک تایمر را نشان می‌دهد:



شکل ۷-۱-۷ Timer در میکروکنترلرهای STM32

همان‌طور که در تصویر بالا مشاهده می‌کنید، کلاک از هر منبعی که تأمین بشود، قبل از رفتن به شمارنده، ابتدا توسط Prescaler تقسیم فرکانسی می‌شود و در نهایت به شمارنده متصل خواهد شد. قبلاً گفتیم که کلاک تایمر می‌تواند از طریق یک تایمر دیگر هم تأمین بشود. در این حالت یک تایمر به‌عنوان Master و تایمر دیگر به‌عنوان Slave در نظر گرفته می‌شود.



شکل ۷-۱-۲ تایمر اول به عنوان Prescaler تایمر دوم عمل می‌کند.

۷-۱-۴- کاربرد زمان‌سنج‌ها در میکروکنترلرهای STM32

تایمیری که در میکروکنترلرهای STM32 قرار داده شده است کاربردهای بسیار زیادی دارد، اما کاربردهایی

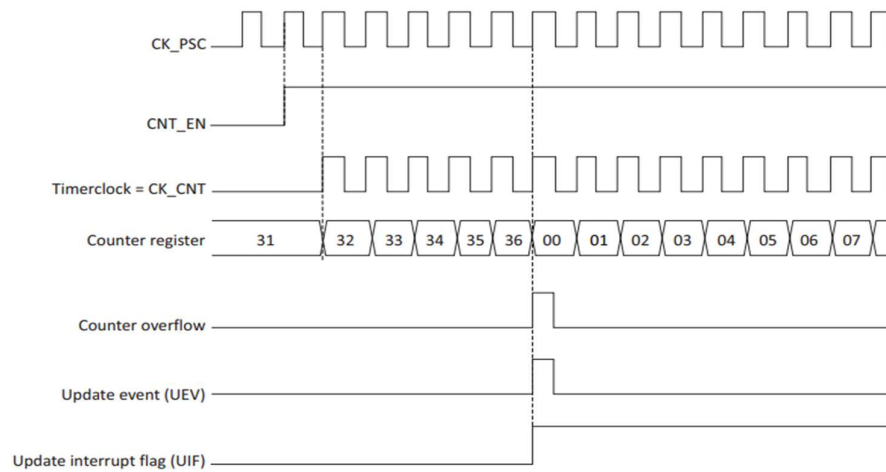
که مهم هستند و ما می‌خواهیم به آن‌ها پردازیم شامل پایه زمانی، PWM و Input capture است. همچنین هرکدام از این تایمرها دارای ۴ کانال مختلف هستند که می‌توان از این کانال‌ها برای تولید PWM یا خواندن سیگنال ورودی (Input capture) و ... استفاده کرد. ما در ادامه می‌خواهیم با استفاده از تایمر، قابلیت پایه زمانی را راه‌اندازی و مدت‌زمان ۱ ثانیه را اندازه‌گیری کنیم.

تایمر را در حالت بالا شمار راه‌اندازی کرده و پس‌ازاینکه اعداد شمارش‌شده توسط شمارنده معادل ۱ ثانیه شد، وقفه واحد تایمر را فعال کرده تا متوجه سپری شدن زمان ۱ ثانیه بشویم و متناسب با آن عملیات موردنظر را انجام بدهیم. اما چگونه متوجه بشویم که اعداد شمارش‌شده توسط شمارنده، معادل ۱ ثانیه است؟

خب همان‌طور که قبلاً گفتیم با استفاده از منبع کلاک و Prescaler، کلاک ورودی Counter واحد تایمر را مشخص می‌کنیم. به‌عنوان مثال منبع کلاک را از نوع داخلی انتخاب کرده و مقدار آن را بر روی 8MHz و Prescaler را بر روی ۷۹۹۹ (به این دلیل ۸۰۰۰ قرار ندادیم چون که شمارش از ۰ شروع می‌شود) تنظیم می‌کنیم. خب با این تنظیمات منبع کلاک بر عدد Prescaler تقسیم‌شده و درنهایت به ورودی کلاک Counter متصل می‌شود. یعنی 8MHz بر عدد ۸۰۰۰ تقسیم‌شده و درنهایت فرکانس 1KHz به ورودی کلاک Counter متصل می‌شود.

با این تنظیمات با اضافه شدن هر عدد به شمارنده، مدت‌زمان 1ms سپری می‌شود. حال باید شمارنده تا عدد ۱۰۰۰ بشمارد تا ما بتوانیم مدت‌زمان ۱ ثانیه را به‌دست بیاوریم. اما چگونه؟ یک رجیستر به اسم ARR یا Auto-reload وجود دارد و زمانی که مقدار شمارنده به مقدار این رجیستر رسید، تایمر یک وقفه به ما می‌دهد و ما متوجه سپری شدن زمانی که مدنظرمان بود می‌شویم و عملیات دلخواه را در روتین وقفه انجام می‌دهیم.

ابتدا به تصویر زیر توجه کنید:

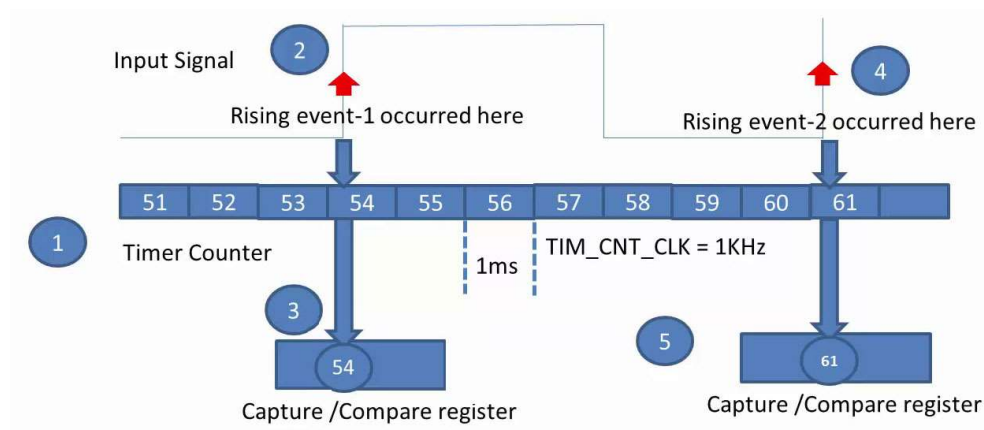


شکل ۷-۳ مقدار رجیستر **Auto-reload** عدد ۳۶ است و زمانی که مقدار شمارنده به عدد ۳۶ برسد شمارنده ریست و یک وقفه **Update interrupt (flag)** هم ایجاد خواهد شد.

همچنین توجه کنید که شمارنده یک کلاک پس از فعال شدن کلاک ورودی شمارنده (CNT_EN) شروع به شمارش می‌کند. این یعنی اگر قرار باشد تا ۱۰۰۰ بشماریم باید مقدار **Auto-reload** را عدد ۹۹۹ تنظیم کنیم چرا که عدد ۰ هم با توضیحی که اکنون دادیم شمارش خواهد شد.

۵-۱-۷- مد Input Capture

معنی لغوی عبارت **Input capture** معادل با ضبط یا ثبت ورودی است. آن چیزی هم که ما می‌خواهیم به آن پردازیم تقریباً مرتبط با همین معنی لغوی است. برای توضیحات مربوط به **Input Capture** به تصویر زیر توجه کنید:



شکل ۷-۴ مد **input capture**

در شکل بالا ما یک سیگنال ورودی (Input Signal) داریم که می‌خواهیم با استفاده از قابلیت Input capture میکروکنترلر فرکانس این سیگنال ورودی را اندازه بگیریم. برای اندازه‌گیری سیگنال ورودی ابتدا باید شمارنده‌ی واحد تایمر را با فرکانس مشخص، به صورت بالا شمار (یا پایین شمار) راه‌اندازی کنیم (مرحله ۱ در تصویر).

سپس سیگنالی که می‌خواهیم فرکانس آن را اندازه‌گیری کنیم را به یکی از کانال‌های تایمر که بر روی پین میکروکنترلر قرار دارد، متصل می‌کنیم و وقفه‌ی این کانال را فعال و حساس به لبه‌ی بالارونده (یا پایین‌رونده) قرار می‌دهیم. در این صورت در هر لبه‌ی بالارونده، یک وقفه رخ می‌دهد (مرحله ۲ در تصویر). پس از اینکه وقفه‌ی مربوط به لبه‌ی بالارونده رخ داد، مقدار شمارنده در لحظه وقوع وقفه، در رجیستر Capture ذخیره می‌شود (مرحله ۳ در تصویر).

شمارنده همچنان به شمارش خود ادامه خواهد داد. در حالی که شمارنده به شمارش خود ادامه می‌دهد، بر روی دومین لبه‌ی بالارونده سیگنال یک وقفه‌ی دیگر رخ می‌دهد (مرحله ۴ در تصویر). پس از اینکه وقفه‌ی مربوط به دومین لبه‌ی بالارونده رخ داد، مقدار شمارنده در لحظه وقوع وقفه، دوباره در رجیستر Capture ذخیره می‌شود (مرحله ۵ در تصویر). اکنون ما یک سری اطلاعات داریم که باید با استفاده از این اطلاعات، مقدار فرکانس سیگنال ورودی را محاسبه کنیم. خب ما اطلاعات مقدار شمارنده در لحظات وقوع وقفه‌ها که در رجیستر Capture ثبت شده است و همچنین فرکانس کلاک شمارنده را در اختیار داریم. حال اگر ما مقدار فرکانس کلاک شمارنده، یعنی 1KHz را بر اختلاف دو مقدار ۶۱ و ۵۴ تقسیم کنیم، فرکانس سیگنال ورودی را محاسبه کرده‌ایم. برای محاسبه فرکانس در زمان‌های مختلف، مراحل بالا را به صورت مستمر تکرار می‌کنیم.

کلیت محاسبه فرکانس سیگنال ورودی، مستقل از اینکه از چه نوع ابزاری میکروکنترلر، FPGA یا هر ابزار دیگری برای این محاسبه استفاده می‌کنیم، روشی است که در بالا ذکر کردیم.

۷-۲- آزمایش

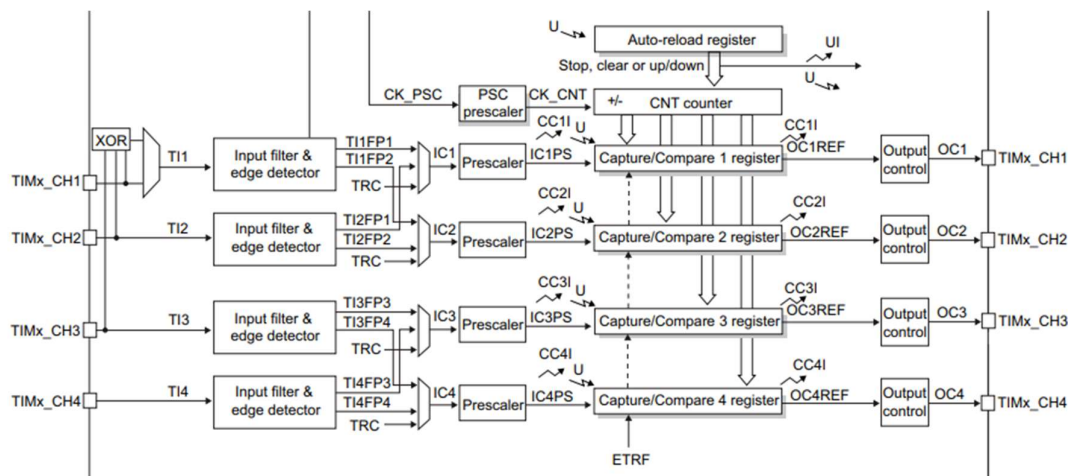
۷-۲-۱- Basic Timer

برنامه‌ای بنویسید که یک پایه میکروکنترلر را هر یک ثانیه Toggle کند. (Basic Timer)

۷-۲-۲- Input Capture

به کمک فانکشن ژنراتور یک موج مربعی به یکی از پایه‌های میکروکنترلر (حدود ۴۰ کیلوهرتز) اعمال کنید. سپس به کمک تایمرها در مد input Capture این فرکانس را اندازه گیری کنید.

برای توضیح حالت Input capture در میکروکنترلرهای STM32 ابتدا به تصویر زیر دقت کنید:



شکل ۷-۵ ساختار زمان‌سنج در میکروکنترلرهای stm32

برای اینکه از حالت Input capture در میکروکنترلرهای STM32 استفاده کنیم و فرکانس یک سیگنال ورودی را اندازه بگیریم، ابتدا باید این سیگنال را به یکی از کانال‌های ورودی واحد تایمر متصل کنیم. همان‌طور که از تصویر بالا مشاهده می‌کنید، در مسیر این سیگنال بلوک‌های فیلتر دیجیتال، تشخیص لبه و Prescaler قرار دارد. برای سادگی کار، ما از فیلتر دیجیتال و Prescaler استفاده نخواهیم کرد اما تشخیص لبه یا همان Edge detector را در حالت لبه‌ی بالارونده قرار می‌دهیم تا با هر لبه‌ی بالارونده یک وقفه رخ بدهد.

با اعمال تنظیمات بالا، در هر لبه‌ی بالارونده‌ی سیگنال ورودی یک وقفه رخ خواهد داد و محتوای رجیستر شمارنده به رجیستر Capture منتقل خواهد شد. نکته‌ای که بسیار مهم است و باید به آن توجه ویژه کرد، نقش فرکانس کلاک شمارنده و همچنین مقدار Auto-reload register است. درواقع با توجه به محدوده‌ی فرکانس سیگنال ورودی، باید فرکانس کلاک شمارنده و مقدار Auto-reload register را به نحوی انتخاب کنیم که از صحت فرکانس محاسبه‌شده مطمئن باشیم.

همان‌طور که گفتیم برای محاسبه‌ی فرکانس سیگنال ورودی، نیاز است که مقدار شمارنده در دو لبه‌ی بالارونده‌ی متوالی را داشته باشیم. از جهتی شمارنده‌ی تایمر در میکروکنترلرهای STM32 با توجه به تنظیماتی که ما اعمال کردیم از ۰ تا مقدار Auto-reload register شروع به شمارش می‌کند که بیشترین مقدار این رجیستر با توجه به ۱۶ بیتی بودن آن برابر با ۶۵۵۳۵ است. پس از اینکه شمارنده به مقدار Auto-reload register رسید یک سرریز رخ می‌دهد و دوباره از ۰ شروع به شمارش می‌کند. با توجه به بازه ۰ تا ۶۵۵۳۵ رجیستر Auto-reload، در زمان شمارش ممکن است چندین حالت مختلف رخ بدهد که در ادامه به تفصیل هر حالت را بررسی خواهیم کرد.

حالت اول:

اگر دو وقفه‌ی مربوط به دو لبه‌ی بالارونده‌ی متوالی، زمانی رخ بدهد که هنوز هیچ سرریزی اتفاق نیفتاده باشد، هیچ مشکلی وجود ندارد و ما می‌توانیم به‌درستی و بدون هیچ خطایی مقدار فرکانس سیگنال ورودی را محاسبه کنیم. مثلاً فرض کنید وقفه‌ی اول زمانی رخ بدهد که مقدار شمارنده ۵۰۰ و وقفه‌ی دوم زمانی رخ بدهد که مقدار شمارنده ۱۰۰۰ است.

حالت دوم:

اگر وقفه‌ی اول قبل از سرریز، و وقفه‌ی دوم بعد از سرریز رخ بدهد. در این حالت هم می‌توان با یک تکنیک ساده که در ادامه، هنگام نوشتن کد خواهیم گفت، به درستی و بدون هیچ خطایی اختلاف بین دو مقدار و به دنبال آن مقدار فرکانس سیگنال ورودی را محاسبه کنیم. مثلاً فرض کنید وقفه‌ی اول زمانی رخ بدهد که مقدار شمارنده ۴۰۰۰ و وقفه‌ی دوم زمانی رخ بدهد که شمارنده سرریز کرده است و به مقدار ۲۰۰۰ رسیده است.

البته در این حالت اگر وقفه‌ی دوم بعد از سرریز رخ بدهد و همچنین مقدار شمارنده در لحظه‌ی وقوع این

وقفه، از مقدار شمارنده در لحظه‌ی وقوع وقفه‌ی اول بیشتر باشد، می‌توان گفت که این حالت یک حالت جدید است. اما ما این حالت را یک زیر حالت از حالت دوم در نظر گرفتیم.

تنها فرقی که بین این دو زیر حالت وجود دارد، این است که در زیر حالت اول، عدد پس از سرریز از عدد قبل سرریز کوچک‌تر، اما در زیر حالت دوم، عدد پس از سرریز از عدد قبل سرریز بزرگ‌تر است. همین مثال بالا را در نظر بگیرید که شمارنده ابتدا ۴۰۰۰۰ بود و پس از سرریز به مقدار ۲۰۰۰ رسید. در زیر حالت دوم فرض کنید به جای ۲۰۰۰ به عدد ۵۰۰۰۰ برسد.

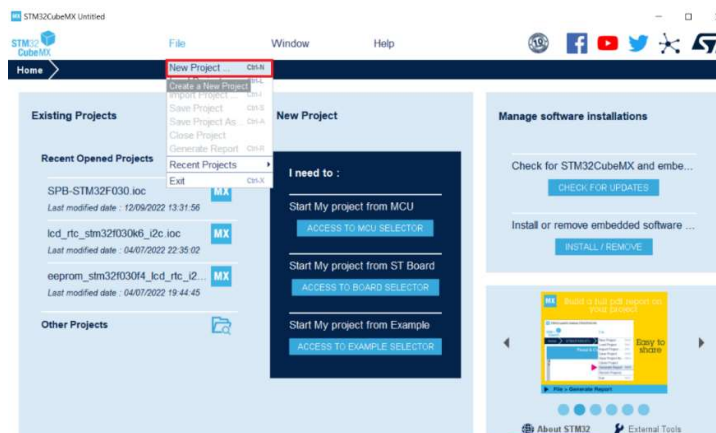
حالت سوم:

حالتی که برای ما مشکل ایجاد می‌کند و باعث می‌شود که نتوانیم اختلاف بین دو مقدار را به درستی محاسبه کنیم، حالتی است که بیش از یک بار سرریز رخ بدهد. مثلاً فرض کنید وقفه‌ی اول زمانی رخ بدهد که مقدار شمارنده ۶۰۰۰ و وقفه‌ی دوم زمانی رخ بدهد که شمارنده دو بار یا بیشتر سرریز کرده است و شمارنده به مقدار ۳۰۰۰ رسیده است. البته در این حالت هم با یک راه‌کار بسیار ساده می‌توانید فرکانس سیگنال ورودی را به درستی محاسبه کنید.

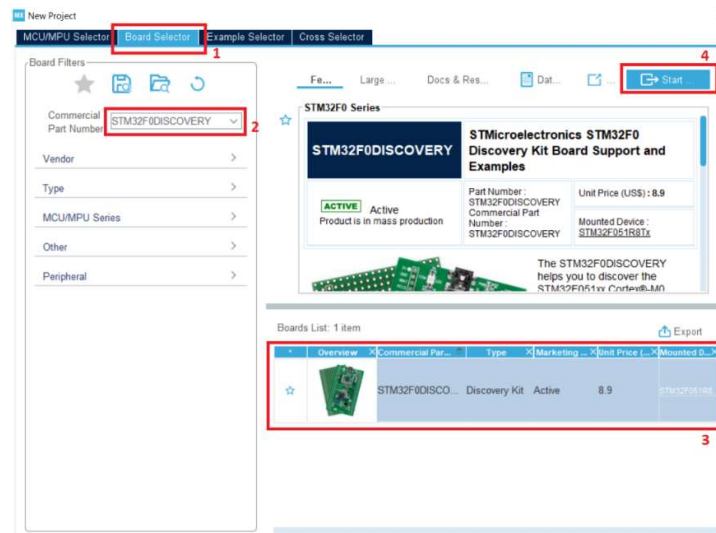
۷-۳- تنظیمات نرم‌افزاری و کدنویسی

۷-۳-۱- Basic Timer

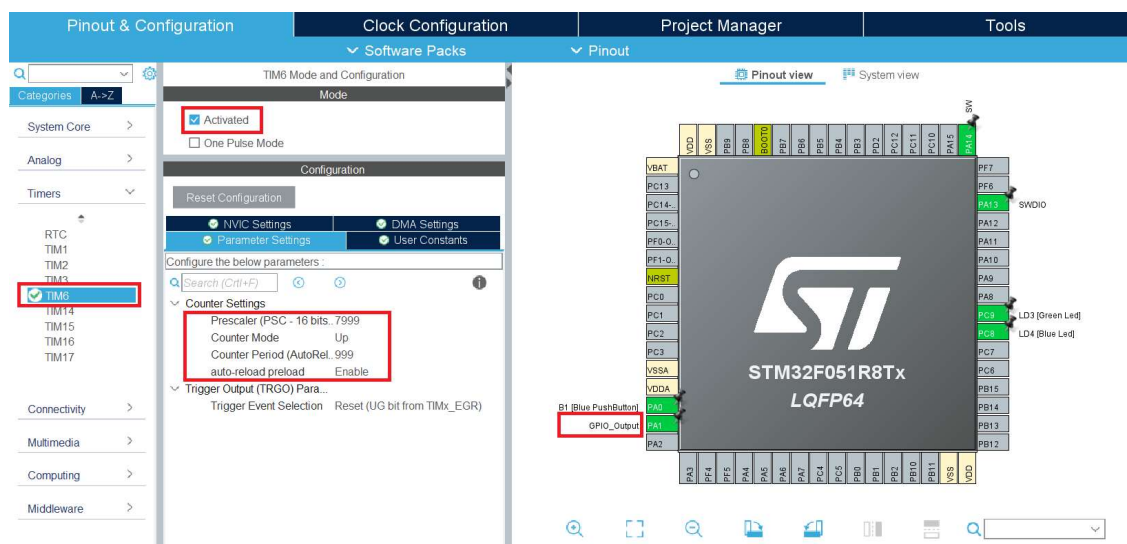
برای تنظیمات اولیه ابتدا یک پروژه جدید ایجاد می‌کنیم:



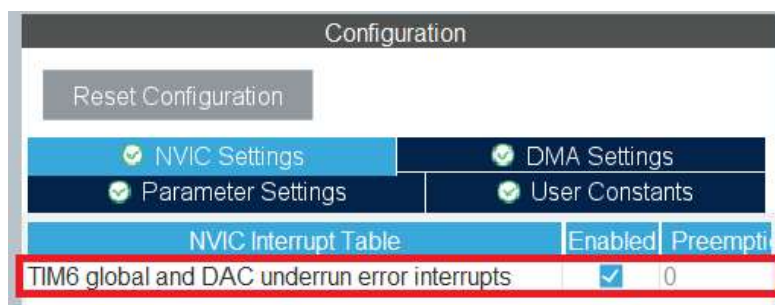
برد یا میکروکنترلر مورد نظر را انتخاب می‌کنیم:



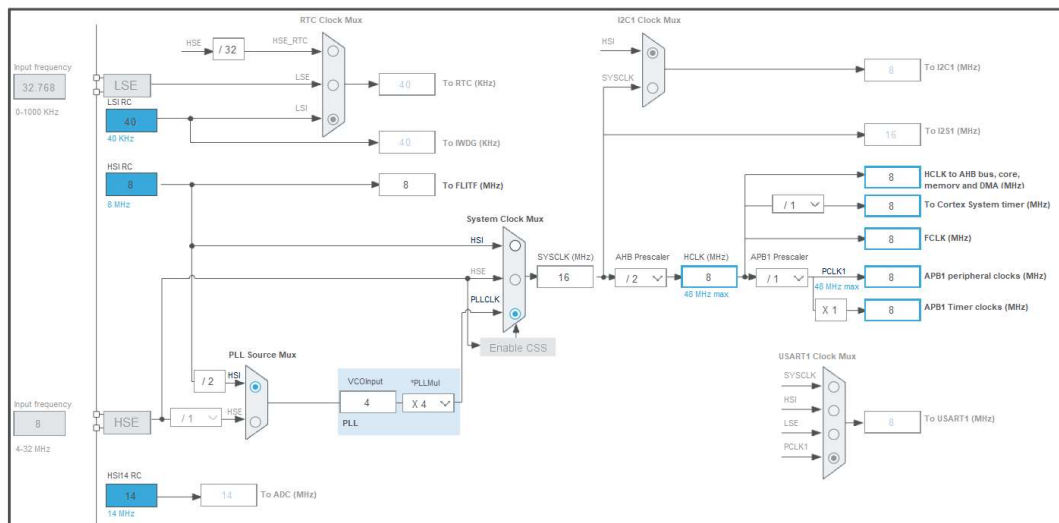
ابتدا مقادیر مناسب را مانند تصویر زیر برای TIM6 تنظیم می‌کنیم، یک پین از میکروکنترلر را هم بر روی حالت خروجی قرار می‌دهیم تا هر ۱ ثانیه یک بار در روتین وقفه آن را Toggle کنیم:



همچنین در قسمت NVIC Setting وقفه مربوطه را نیز فعال می‌کنیم:



تنظیمات Clock Configuration را هم به نحوی تنظیم می‌کنیم که کلاک ورودی تایمر 8MHz باشد:



پس از انجام تنظیمات بالا به نرم افزار CubeIDE می رویم تا کد این برنامه را بنویسیم. تابع callBack مناسب را در فایل main.c کپی می کنیم:

```

/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(htim);

    /* NOTE : This function should not be modified, when the callback is
    needed,
               the HAL_TIM_PeriodElapsedCallback could be implemented in
    the user file
    */
}
/* USER CODE END 4 */

```

فراغش نشود که ابتدا باید تایمر شروع به شمارش کند یعنی:

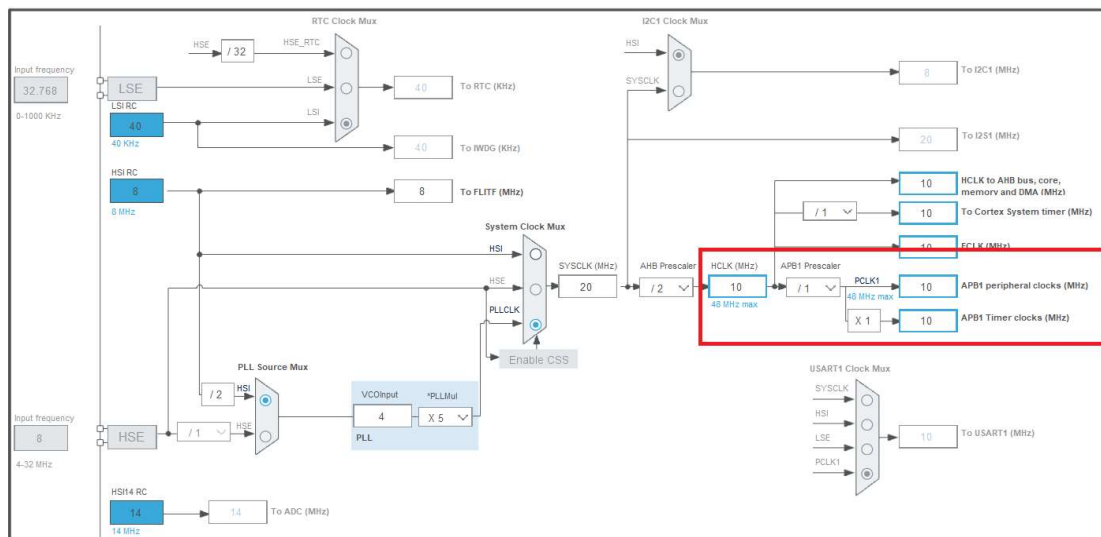
```
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim6);
/* USER CODE END 2 */
```

با برابر شدن مقدار رجیستر CNT و ARR این وقفه رخ می‌دهد. تنها کافی است با هر بار وقوع وقفه وضعیت پایه تغییر کند. خروجی مطابق شکل زیر می‌شود.



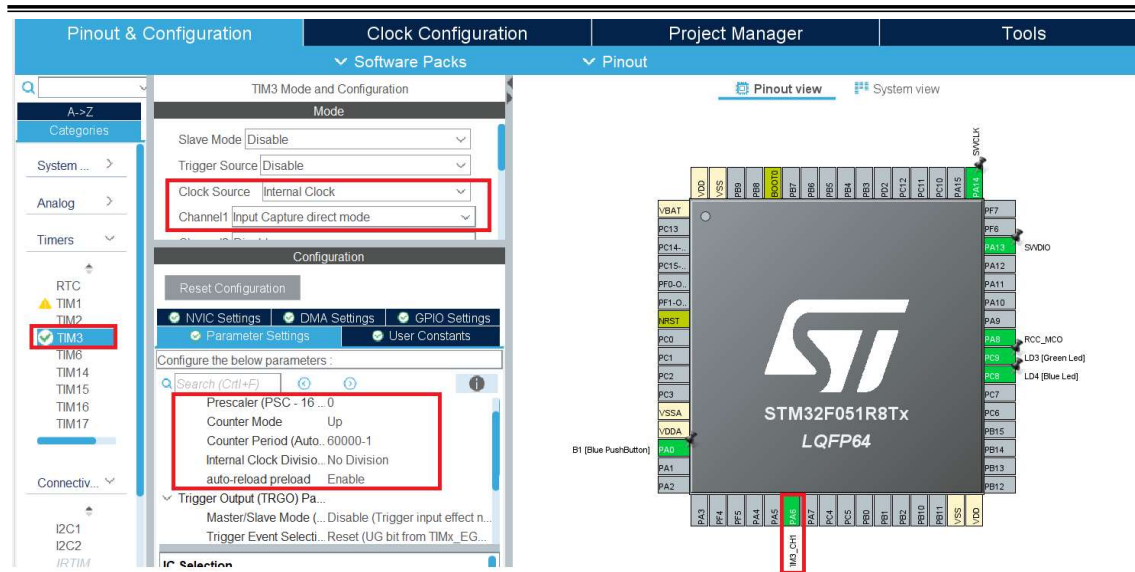
۲-۳-۷- Input Capture

بعد از ایجاد یک پروژه‌ی جدید برای فرکانس متر، فرکانس کاری تایمرها را برابر 10MHz تنظیم می‌کنیم:

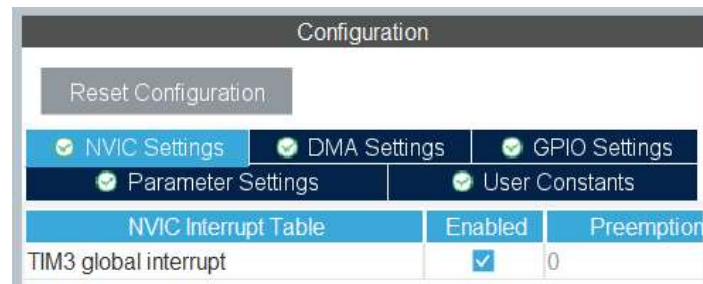


با فعال‌سازی کانال یک از تایمر سه در مد Input Capture سایر پارامترهای تایمر نیز بدین ترتیب

مقداردهی می‌شوند:



وقفه را برای تایمر سه فعال می‌کنیم:



توابع Callback برای سرریز تایمر و ثبت ورودی را در فایل main.c کپی می‌کنیم:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    UNUSED(htim);
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    UNUSED(htim);
}
```

با فعال سازی زمان‌سنج:

```
/* USER CODE BEGIN 2 */

HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);

/* USER CODE END 2 */
```

در صورت وقوع وقفه‌ی سرریز باید یک مقدار متغیر ovc افزوده شود تا مقدار دقیق فرکانس قابل

محاسبه باشد. در صورت وقوع وقفه Input Captre برای بار اول value_1 و برای بار دوم value_2 ذخیره شده و در نهایت فرکانس محاسبه می‌شود.

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(htim);

    /* NOTE : This function should not be modified, when the callback is
    needed,
               the HAL_TIM_PeriodElapsedCallback could be implemented in
    the user file
    */
    ovc_value++;
}

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(htim);

    /* NOTE : This function should not be modified, when the callback is
    needed,
               the HAL_TIM_IC_CaptureCallback could be implemented in the
    user file
    */

    if(!start_flag)
    {
        value_1 = HAL_TIM_ReadCapturedValue(&htim3, TIM_CHANNEL_1);
        ovc_value = 0;
        start_flag = 1;
    }else if(start_flag == 1)
    {
        value_2 = HAL_TIM_ReadCapturedValue(&htim3, TIM_CHANNEL_1);
        freq = (float)10000 / ((value_2 + (ovc_value * 60000)) -
value_1); //KHz
        start_flag = 0;
    }
}

/* USER CODE END 4 */
```

۴-۷- سؤال ها و تمرین های برنامه نویسی

- ۱- برنامه ای بنویسد که یک عدد از کیپد دریافت کند و متناسب با آن عدد موج مربعی با فرکانس دلخواه روی یکی از پایه های میکروکنترلر ایجاد کند.
- ۲- موج مربعی با Duty Cycle و فرکانس دلخواه به یکی از پایه های میکرو متصل کنید. حال با کمک Input Capture زمان سنج میکرو این پارامترهای را اندازه گیری کرده و نمایش دهید.

آزمایش ۸:

کار با واحد زمان سنج ها و شمارنده ها

(Timer/Counter) – بخش دوم

۸-۱- در سنامه

در آزمایش قبل به بررسی زمان‌سنج‌های STM32 پرداختیم و دو مد واحد زمانی و ثبت ورودی را مورد آزمایش قرار دادیم. در این آزمایش نیز دو مد دیگر که Output Compare و Pulse Width Modulation هستند، بررسی می‌کنیم.

۸-۱-۱- مد مقایسه خروجی چیست؟

در این حالت، تایمر باید تا حداکثر سطح (تنظیم شده توسط ARR) شمارش کند، موارد زیر رخ می‌دهد:

- پین خروجی مربوطه را به یک مقدار قابل برنامه ریزی که توسط حالت مقایسه خروجی تعریف شده است، تغییر می‌دهد.

- یک پرچم در ثبت وضعیت وقفه تنظیم می‌کند.
 - اگر ماسک وقفه مربوطه تنظیم شده باشد، یک وقفه ایجاد می‌کند.
 - اگر بیت فعال‌سازی مربوطه تنظیم شده باشد یک درخواست DMA ارسال می‌کند.
- در این آزمایش ما وضعیت یک پین را Toggle می‌کنیم.

۸-۱-۲- مدولاسیون عرض پالس (PWM)

می‌دانیم که سیگنال‌ها دو نوع هستند، آنالوگ و دیجیتال. سیگنال‌های آنالوگ دارای ولتاژهایی مانند ۱ ولت، ۳ ولت و ... هستند اما سیگنال‌های دیجیتال فقط مقادیر ۰ و ۱ را می‌پذیرند. یعنی ولتاژ آنها یا ۰ یا ۱ است. از طرفی خروجی سنسورها سیگنال‌های آنالوگ است و چون میکروکنترلرها فقط زبان دیجیتال را می‌فهمند، پس این سیگنال‌ها ابتدا به کمک مبدل‌های ADC به معادل دیجیتال خود تبدیل شده و بعد وارد مرحله‌ی پردازش با میکروکنترلر می‌شوند. اما بعد از اینکه پردازش شدند، از آنجایی که قرار است دوباره به

دیوایس‌هایی بروند که آنها نیز با مقادیر آنالوگ کار می‌کنند و کنترل می‌شوند، پس مجدداً باید از فرمت دیجیتال به آنالوگ برگردانده شوند. برای این تبدیل از روش‌هایی مانند PWM یا مبدل‌های دیجیتال به آنالوگ (DAC) استفاده می‌کنیم.

PWM روشی برای کنترل دیوایس‌های آنالوگ با استفاده از مقادیر دیجیتال است. مثلاً کنترل شدت نور یک LED و یا کنترل سرعت یک فن DC.

این دیوایس‌ها با سیگنال‌های آنالوگ کار می‌کنند اما PWM سیگنال‌های آنالوگ خالص تولید نمی‌کند. در حقیقت پالس‌هایی با عرض بسیار کم تولید می‌کند که در مجموع شبیه به یک سیگنال آنالوگ به نظر می‌رسند. این پالس‌های کوتاه براساس duty cycle ساخته می‌شوند.

PWM در Duty cycle

منظور از duty cycle، نسبت زمانی که در آن سیگنال در وضعیت High قرار دارد، به کل زمان یک سیکل است. با این تعریف، واضح است که اگر یک سیگنال همواره یک باشد، duty cycle آن ۱۰۰ درصد و اگر همواره صفر باشد، duty cycle آن صفر درصد است.

25% Duty Cycle



50% Duty Cycle



75% Duty Cycle



شکل ۸-۱ مفهوم Duty Cycle در مدولاسیون عرض پالس

۸-۲- آزمایش

۸-۲-۱- مد مقایسه خروجی

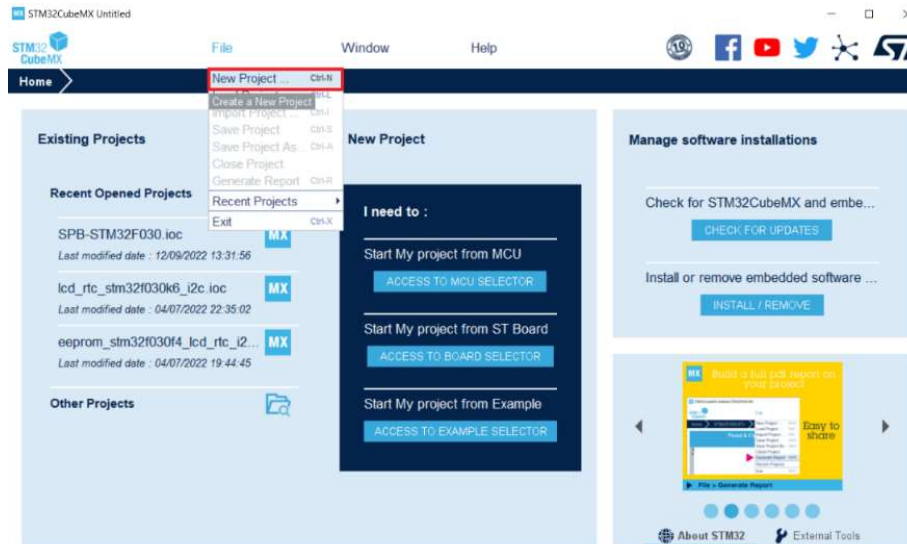
با اتصال یک Pushbutton به میکروکنترلر، با هر بار فشردن کلید فرکانس موج مربعی ۱۰ کیلوهرتز افزایش یابد (تا رسیدن به فرکانس ۱۰۰ کیلوهرتز) با رسیدن به حداکثر فرکانس و فشردن کلید این بار فرکانس با پله‌های ۱۰ کیلوهرتز کاهش باید تا سرانجام به فرکانس ۱۰ کیلوهرتز برسد. به کمک زمان‌سنج میکروکنترلر در مد مقایسه خروجی موج مربعی در بازه‌ی فرکانسی (10KHz تا 100KHz) ایجاد کنید.

۸-۲-۲- مدولاسیون عرض پالس

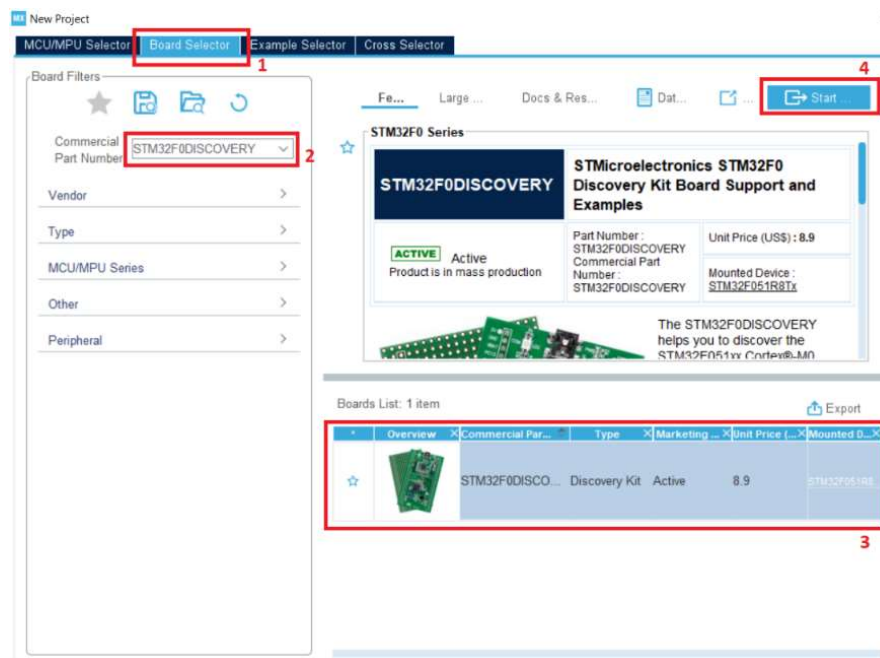
به کمک مدولاسیون عرض پالس و تغییر Duty Cycle، یک دیود نوری را به نحوی راه‌اندازی کنید که نور آن رفته رفته افزایش یافته و با رسیدن به مقدار حداکثر ۹۰٪ دوباره نور آن کاهش یابد تا به سطح ۱۰٪ برسد و این چرخه ادامه یابد.

۳-۸- تنظیمات نرم‌افزاری و کدنویسی

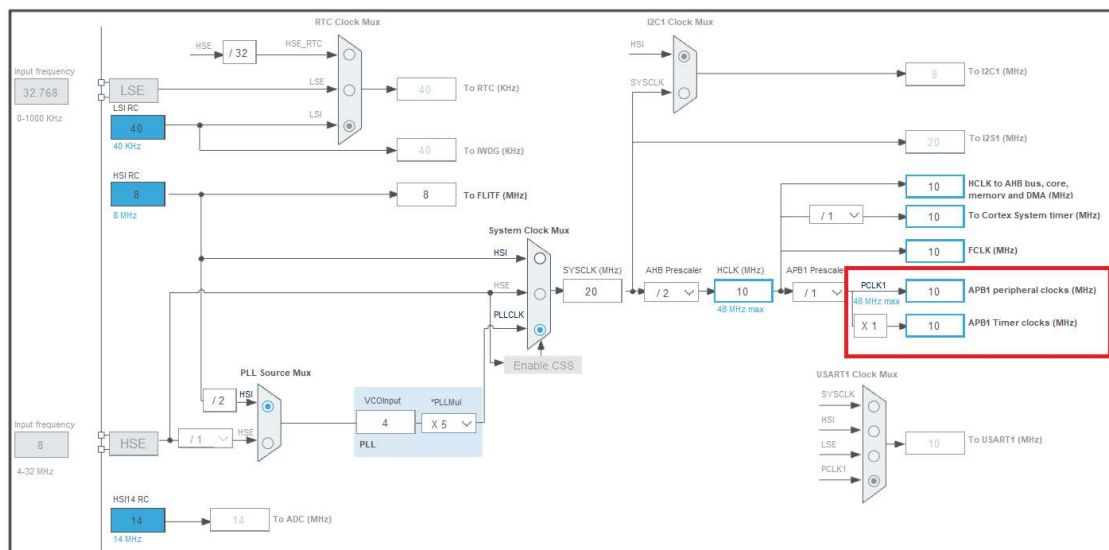
برای تنظیمات اولیه ابتدا یک پروژه جدید برای هر کدام از آزمایش‌ها ایجاد می‌کنیم:



برد یا میکروکنترلر مورد نظر را انتخاب می‌کنیم:



ابتدا فرکانس کاری زمان سنج را روی 10MHz تنظیم می کنیم:



تنظیمات زمان سنج مطابق شکل زیر است:

كد:

```

/* USER CODE BEGIN PV */
uint16_t freq_lut[10] = {500,250,167,125,100,83,71,62,55,50};
uint8_t up_down_flag = 1; //increase frequency
uint8_t counter = 0;
/* USER CODE END PV */

/* USER CODE BEGIN WHILE */

while (1)
{
    if(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0))
    {
        HAL_Delay(20);
        while(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0));
        HAL_TIM_OC_Stop(&htim3, TIM_CHANNEL_1);
        if(counter>=9)
        {
            up_down_flag = 0;
        }else if(counter<=0)
        {
            up_down_flag = 1;
        }

        counter = up_down_flag ? (counter+1) : (counter-1);
        __HAL_TIM_SET_AUTORELOAD(&htim3,freq_lut[counter]-1);
        HAL_TIM_OC_Start(&htim3, TIM_CHANNEL_1);

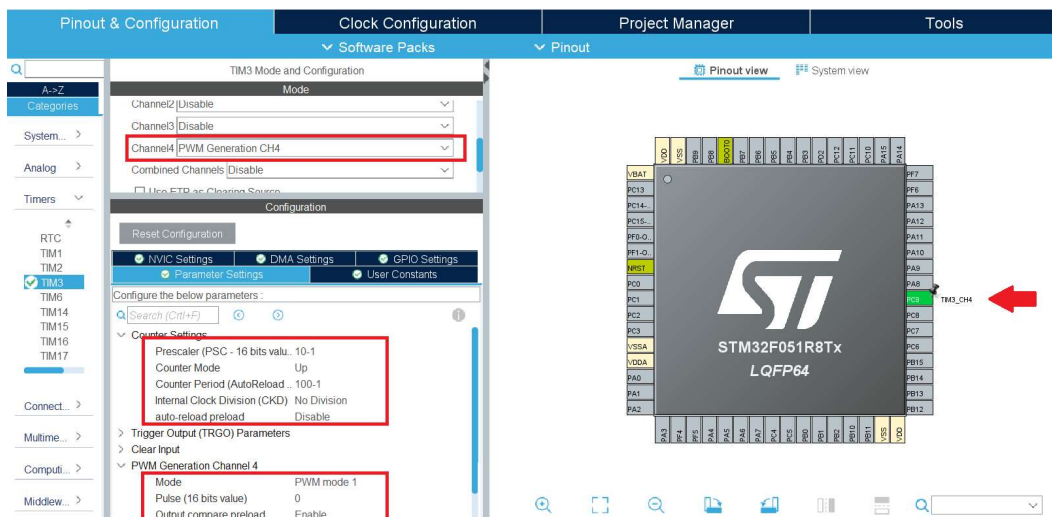
    }
}
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */

```

۲-۳-۸- آزمایش زمان‌سنج در مد مدولاسیون عرض پالس

تنظیمات زمان‌سنج:



کد:

```

/* Infinite loop */
/* USER CODE BEGIN WHILE */
uint8_t pwm_base = 6;
uint8_t up_down_flag = 1;
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);

while (1)
{
    __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_4,pwm_base);
    if(pwm_base>=90)
    {
        up_down_flag = 0;
    }else if(pwm_base<=5)
    {
        up_down_flag = 1;
    }
    pwm_base = up_down_flag ? (pwm_base+2) : (pwm_base-6);
    HAL_Delay(50);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */

```

۴-۸- پرسش‌ها و تمرین‌های برنامه نویسی

- ۱- با سخت‌افزار مناسب و به کمک PWM زمان‌سنج‌ها برنامه‌ای بنویسید که بتوان دور موتور یک موتور DC را کنترل نمود.
- ۲- به کمک مد مقایسه خروجی سه موج مربعی با اختلاف فاز ۴۵، ۹۰ و ۱۸۰ درجه نسبت به موج مربعی مرجع بسازید.

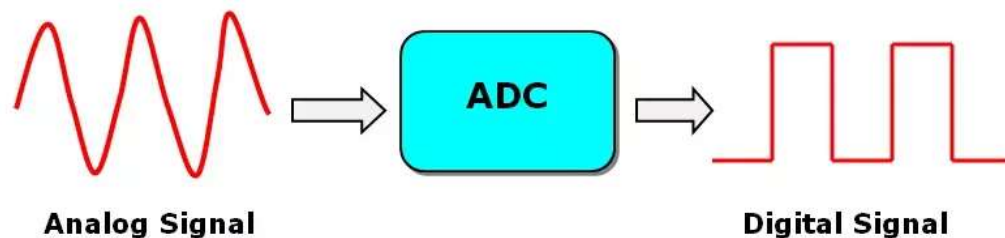
آزمایش ۹:

کار با واحد مبدل آنالوگ به دیجیتال (ADC)

۹-۱- درسنامه

دنیایی که در آن زندگی می‌کنیم از انواع کمیت‌های آنالوگ تشکیل شده است. برای اندازه‌گیری این کمیت‌ها با استفاده از میکروکنترلر‌ها بایستی ابتدا این کمیت‌ها را به منطق دیجیتال (اعداد باینری) تبدیل کنیم. در اکثر میکروکنترلر‌ها بخشی تحت عنوان مبدل آنالوگ به دیجیتال (ADC) وجود دارد. که امکان اندازه‌گیری کمیت‌های آنالوگ را در اختیار ما قرار می‌دهد.

کلمه ADC مخفف Analog to Digital Converter به معنای مبدل آنالوگ به دیجیتال است. در واقع ADC مقادیر فیزیکی یک پدیده در دنیای واقعی را به یک زبان دیجیتالی تبدیل می‌کند که در سیستم‌های کنترل، محاسبات داده، انتقال داده و پردازش اطلاعات استفاده می‌شود. معمولاً از ADC برای تبدیل مقادیری مثلاً ولتاژ و جریان به سیگنال دیجیتال استفاده می‌شود. در میکروکنترلرهای STM32 بسته به نوع میکروکنترلر مورد استفاده، چند کانال مبدل آنالوگ به دیجیتال ۱۲ یا ۱۶ بیتی در اختیار شما قرار دارد.



شکل ۹-۱ تبدیل سیگنال آنالوگ به دیجیتال توسط ADC

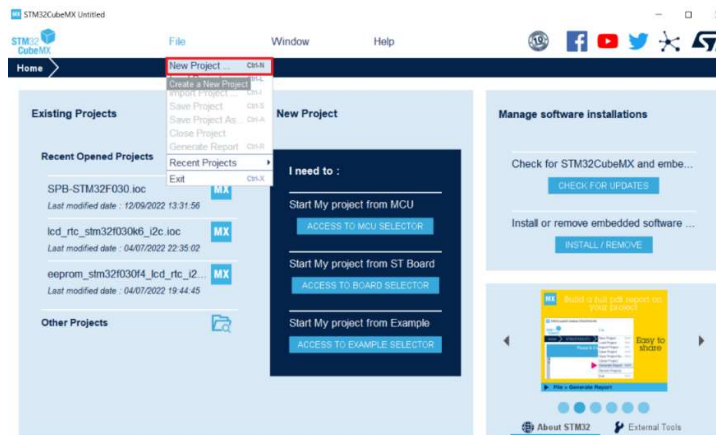
مبدل آنالوگ به دیجیتال تعبیه شده در میکروکنترلرهای STM32 از روش SAR یا تقریب متوالی استفاده می‌کند، که توسط آن تبدیل در چند مرحله انجام می‌شود. تعداد مراحل تبدیل برابر با تعداد بیت‌های مبدل ADC است. هر مرحله توسط کلاک ADC هدایت می‌شود. طراحی داخلی ADC بر اساس تکنیک خازن تغییر یافته است. ولتاژ مرجع در برخی پکیج‌های میکروکنترلر STM32 به پایه VDDA متصل است.

۹-۲- آزمایش

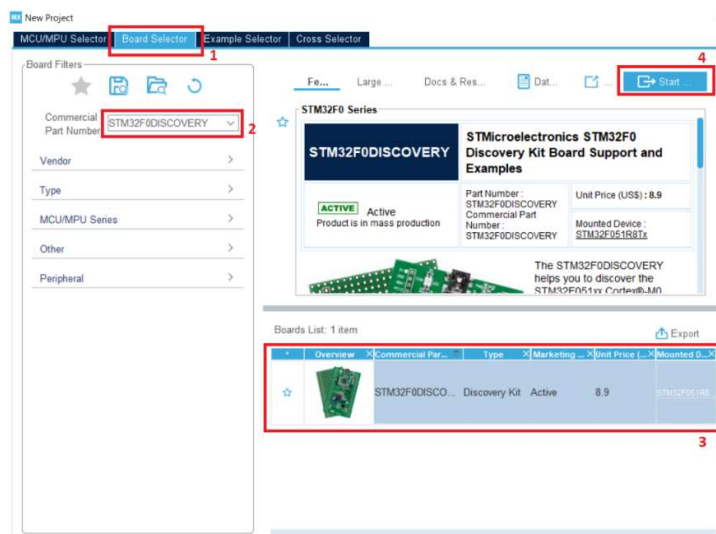
در این آزمایش قصد داریم تا مقدار ولتاژ ورودی داده شده به پایه ADC میکروکنترلر STM32 را اندازه‌گیری نموده و در نرم‌افزار STM STUDIO مقدار آن را مشاهده کنیم. برای این کار بایستی یک پتانسیومتر در ورودی ADC میکروکنترلر قرار دهیم تا تغییرات ولتاژ ورودی را اندازه‌گیری کنیم.

۹-۳- تنظیمات نرم‌افزاری و برنامه نویسی

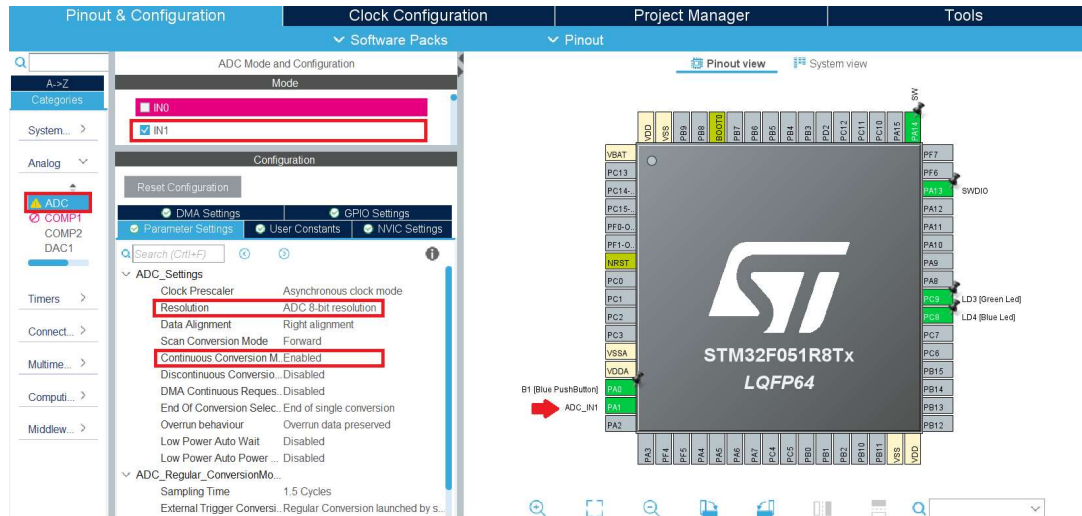
برای تنظیمات اولیه ابتدا یک پروژه جدید ایجاد می‌کنیم:



برد یا میکروکنترلر مورد نظر را انتخاب می‌کنیم:



سپس در بخش Parameter Setting ابتدا بایستی مد کاری ADC را انتخاب کنید. در قدم بعدی بایستی برای این که ADC به طور پیوسته کار کند، گزینه Continuous Conversion Mode را فعال کنید. پس از آن مقدار Resolution را برابر هشت بیت قرار دهید.



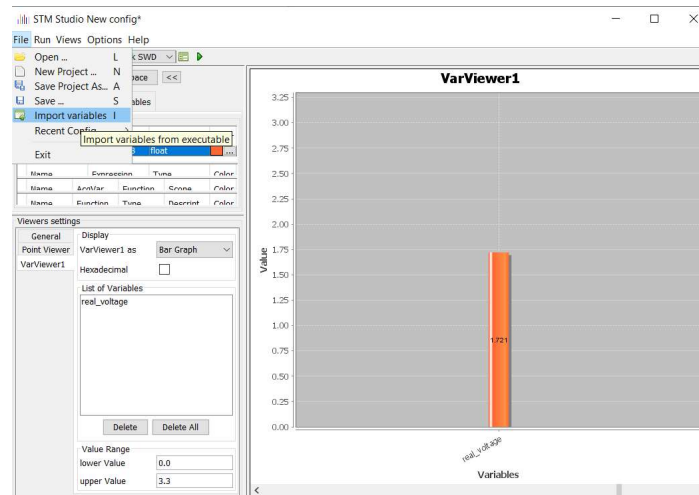
متغیرهای عمومی این گونه تعریف می شوند:

```
/* USER CODE BEGIN PV */
uint32_t adc_value;
float volatile real_voltage;
/* USER CODE END PV */
```

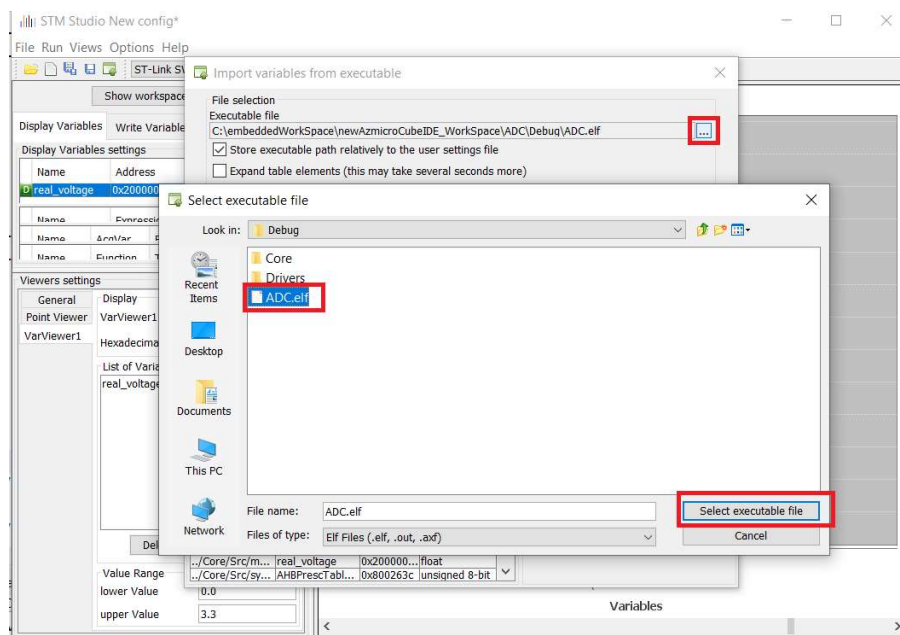
کد داخل حلقه‌ی بی‌نهایت برای خواندن ولتاژ پتانسیومتر:

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
HAL_ADC_Start(&hadc);
while (1)
{
    if (HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY) != HAL_OK)
    {
        Error_Handler();
    }else
    {
        adc_value = HAL_ADC_GetValue(&hadc);
        real_voltage = (float)adc_value/255*3.3;
    }
}
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

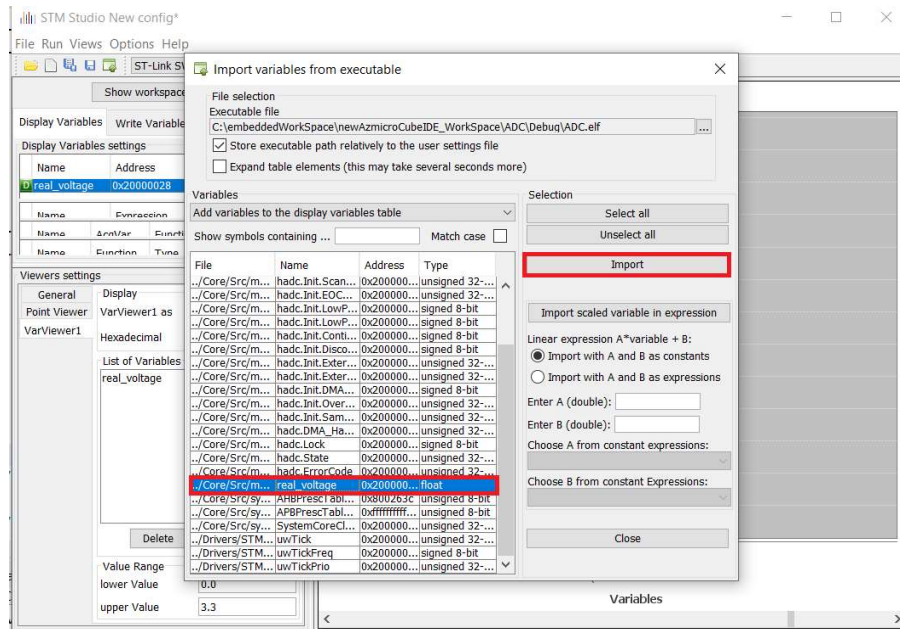
با تعریف متغیر به صورت عمومی می‌توان مقدار آن را در حین اجرای برنامه، در نرم‌افزار STM-STUDIO مشاهده نمود. به این منظور ابتدا از قسمت File گزینه‌ی Import Variables را انتخاب می‌کنیم.



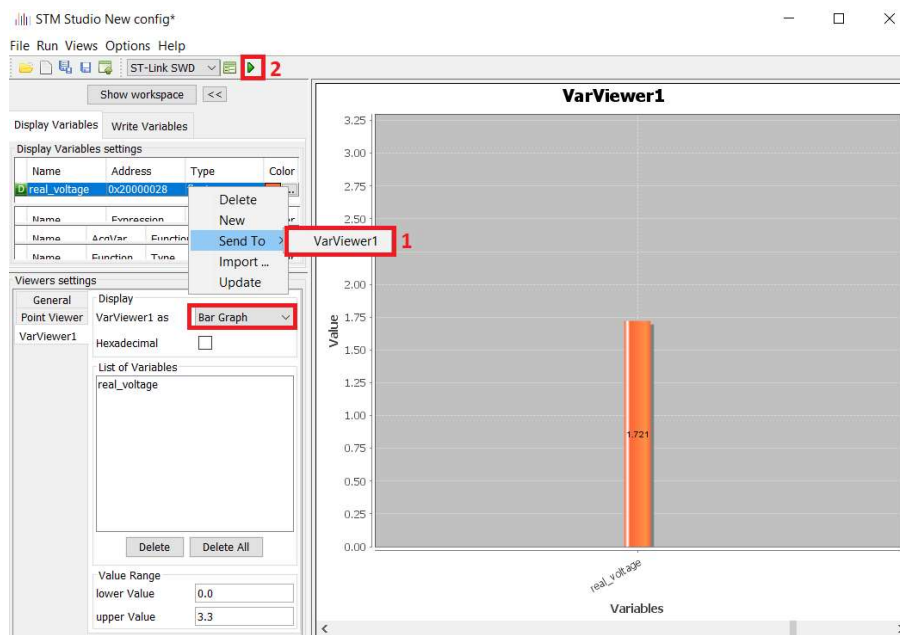
در نهایت فایل elf را از دایرکتوری پروژه انتخاب نموده و متغیر مورد نظر را اضافه می‌کنیم.



متغیر مورد نظر را import می‌نماییم.



حال با کلیک روی متغیر و انتخاب `varView1` و سپس کلیک بر `Run` (فلش سبز رنگ) مقدار متغیر مشاهده می‌شود.



۴-۹- پرسش‌ها و تمرین‌های برنامه نویسی

- ۱- سخت‌افزار مناسبی طراحی کنید که به کمک واحد ADC میکروکنترلر و برنامه نویسی آن، بتوان مقدار یک خازن را خواند.
- ۲- با مراجعه به Reference Manual میکروکنترلر STM32F051R8 متوجه می‌شویم که یک سنسور برای اندازه‌گیری دمای سطح تراشه در این میکروکنترلر تعبیه شده است. به کمک این سنسور و واحد ADC دمای سطح تراشه را اندازه‌گیری کنید.

آزمایش ۱۰:

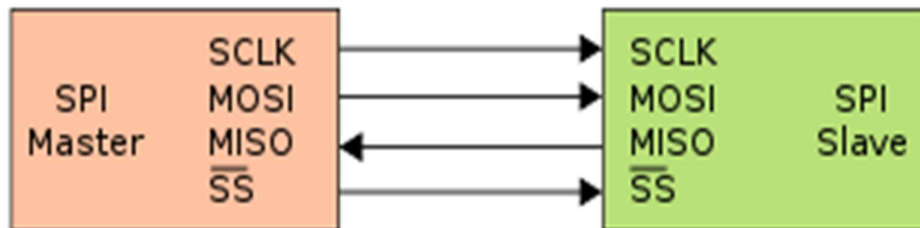
پروتکل ارتباطی SPI

۱-۱۰-۱- درسنامه

ارتباط قطعات الکترونیکی دقیقا مانند ارتباط بین انسان هاست. هر دو طرف باید به یک زبان صحبت کنند. در الکترونیک به این زبان ها پروتکل ارتباطی می گویند.

عبارت SPI کوتاه شده ی Serial Peripheral Interface به معنای واسط ارتباط سریال است.

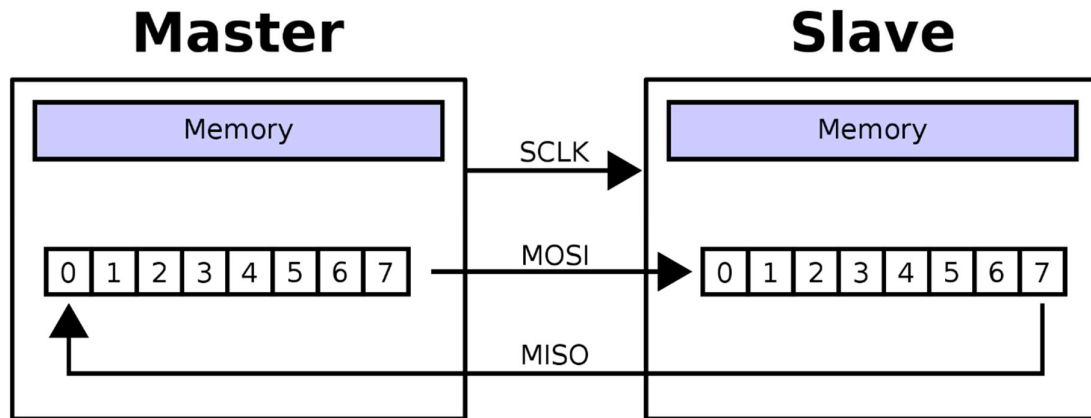
از پروتکل SPI برای برقرار ارتباط بین سنسورها، ماژول ها و میکروکنترلرهای مختلف استفاده می شود. قطعاتی که از SPI پشتیبانی می کنند تنها از ۴ پایه برای انتقال اطلاعات استفاده می نمایند. یکی با نام MOSI و دیگری با نام MISO که علاوه بر این دو پایه در SPI دو پایه دیگر نیز در انتقال اطلاعات نقش دارند. یکی از این پایه ها برای همزمان کردن انتقال داده ها بین دو طرف استفاده می شود که SCK نام دارد و پایه دیگر که SS نام دارد برای اعلام آغاز و پایان انتقال داده کاربرد دارد.



شکل ۱۰-۱ شمای پروتکل SPI

۱-۱۰-۱-۱ انتقال داده در پروتکل SPI

پروتکل SPI شامل دو shift register است که یکی در Master و دیگری در Slave قرار دارد. هم چنین یک مولد پالس ساعت در Master قرار دارد که پالس ساعت را برای shift register ها فراهم می کند.



شکل ۱۰-۲ انتقال داده بین دو شیفت رجیستر در پروتکل SPI

مولد پالس ساعت در Master پالس ساعت را برای هر دو ثبات جابه جایی که در Master و Slave قرار دارند فراهم می کند و بر روی پایه SCK قرار می دهد. رجیسترهای جابجایی در SPI هشت یا شانزده بیت طول دارند. بنابراین بعد از هشت یا شانزده پالس ساعت اطلاعات داخل دو ثبات جابجایی با هم دیگر تعویض می شوند.

۱۰-۱-۲- مزایا و معایب پروتکل SPI

با توجه به مزایا و معایب این پروتکل و مقایسه با پروتکل های دیگر می توان انتخاب درست را در پروژه های خود داشت:

مزایا:

- بیت شروع و پایانی وجود ندارد، لذا داده ها پیوسته منتقل خواهند شد.
- سیستم پیچیده آدرس دادن slave مانند I2C وجود ندارد.
- سرعت بالاتر انتقال اطلاعات نسبت به I2C
- با توجه به دو خط جدا برای MISO و MOSI امکان دریافت و ارسال داده به صورت هم زمان وجود دارد.

معایب:

- استفاده از چهار سیم (در صورتی که I2C و UART از دو سیم استفاده می کنند)
- هیچ علامتی مبنی بر دریافت صحیح داده وجود ندارد (I2C این امکان را دارد)
- هیچ گونه روش تشخیص خطا ندارد مانند پیریتی در UART

۳-۱-۱۰- سایر نکات پروتکل SPI

- حداکثر فاصله‌ای که داده می‌تواند تحت پروتکل SPI طی کند حدود ۳ متر است.
- حداکثر سرعت پروتکل SPI برابر است با نصف کلاک سیستم.
- پروتکل SPI به صورت پیش فرض دوطرفه (FULL-DUPLEX) است.
- در ارتباط به صورت نیمه دوطرفه (HALF-DUPLEX) پایه‌ی MOSI از MASTER به پایه‌ی MISO از SLAVE متصل می‌شود.
- در ارتباط یک طرفه MASTER یا فقط دریافت می‌کند یا فقط ارسال.

۴-۱-۱۰- پارامترهای مهم پروتکل SPI

پارامترهای تعیین کننده در این پروتکل عبارتند از:

- MODE: MASTER یا SLAVE
- BUS CONFIG: FULL-DUPLEX یا HALF-DUPLEX یا SIMPLEX
- DFF: ۸ بیتی یا ۱۶ بیتی
- CPHA: نمونه برداری در لبه‌ی اول یا دوم کلاک
- CPOL: ولتاژ کلاک در حالت بیکاری (HIGH OR LOW)
- SSM: انتخاب SLAVE به صورت نرم‌افزاری یا سخت افزاری
- CLOCK PRESCALER: تقسیم کننده‌ی کلاک

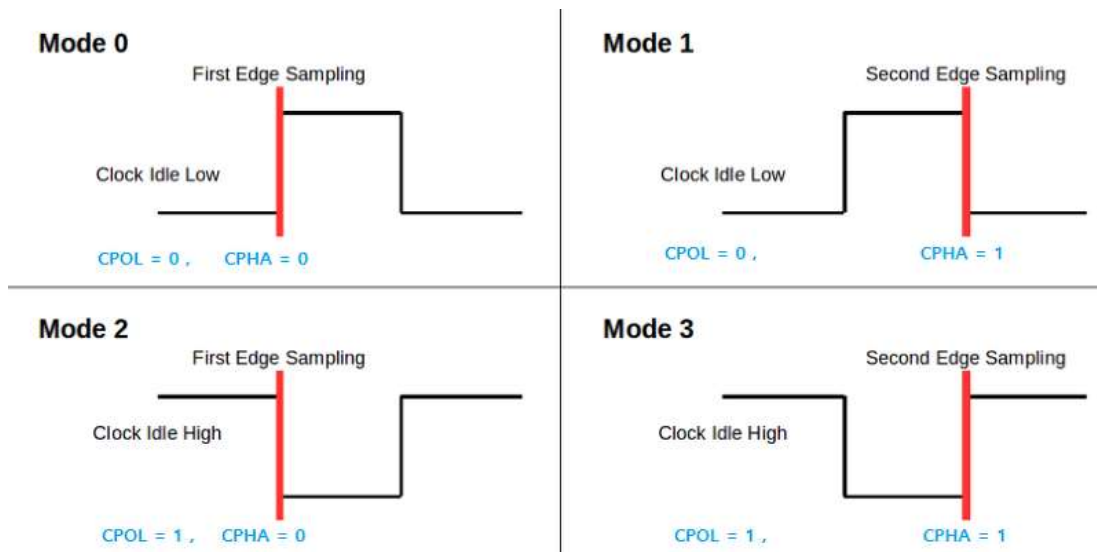
۵-۱-۱۰- مفاهیم CPOL و CPHA در پروتکل SPI

Master علاوه بر تنظیم کردن فرکانس پالس ساعت باید قطبیت (Polarity) کلاک و فاز آن را نیز نسبت به داده‌ها مشخص کند.

در $CPOL=0$ سطح مبنای اولیه ی کلاک low است.

در $CPOL=1$ برعکس حالت بالا مقدار پایه‌ای پالس ساعت High است.

زمانی که $CPHA=0$ است، به این معناست که داده‌ها روی اولین لبه ی پالس ساعت و $CPHA=1$ به این معناست که داده‌ها روی لبه ی دوم پاس ساعت بدون توجه به اینکه پالس ساعت در حال بالا رفتن است یا پایین رفتن، خوانده می‌شوند.



شکل ۱۰-۳ مفهوم CPOL و CPHA در پروتکل SPI

۲-۱۰- آزمایش

۱-۲-۱۰- برقراری ارتباط بین دو میکروکنترلر با پروتکل SPI

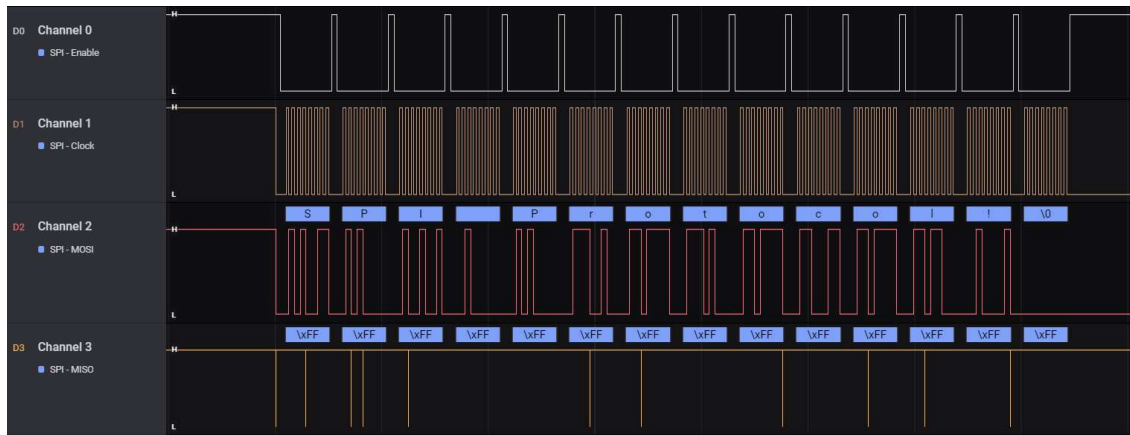
یک میکروکنترلر به عنوان Master و یک میکروکنترلر به عنوان Slave انتخاب کنید. به میکروکنترلر Master صفحه کلید ماتریسی و به Slave یک نمایشگر کاراکتری متصل کنید Master باید کلید وارد شده را بخواند و برای Slave ارسال کند. Slave نیز کلید خوانده شده را روی نمایشگر نشان دهد.

۳-۱۰- تنظیمات نرم افزاری و کدنویسی

از تابع زیر می توان هم برای دریافت و هم برای ارسال اطلاعات به کمک پروتکل SPI بهره برد:

```
HAL_SPI_TransmitReceive(hspi, pTxData, pRxData, Size, Timeout);
```

خروجی برنامه به شکل زیر است:



۴-۱۰- پرسش ها و تمرین های برنامه نویسی

۱- با استفاده از نرم افزار Proteus دما را از سنسور TC72 به کمک پروتکل SPI بخوانید و روی

نمایشگر نشان دهید.

آزمایش ۱۱:

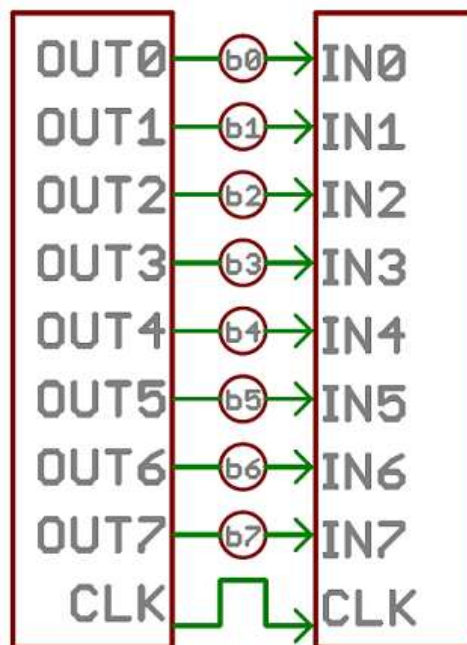
پروتکل ارتباطی UART

۱-۱-۱- درسنامه

ارتباط سریال چیست؟ قبل از شروع به توصیف ارتباط سریال بهتر است بدانیم که؛ که سیستم های نهفته و مدارات الکترونیک دیجیتال پیشرفته (میکروکنترلرها، پردازنده ها و...) نیازمند ارتباط با یکدیگر هستند. برای اینکه این مدارها اطلاعات خود را مبادله کنند، باید یک پروتکل ارتباطی مشترک داشته باشند. صدها پروتکل ارتباطی برای دستیابی به این تبادل اطلاعات تعریف شده اند و به طور کلی هرکدام از آنها می توانند به دودسته تقسیم شوند: موازی یا سریال.

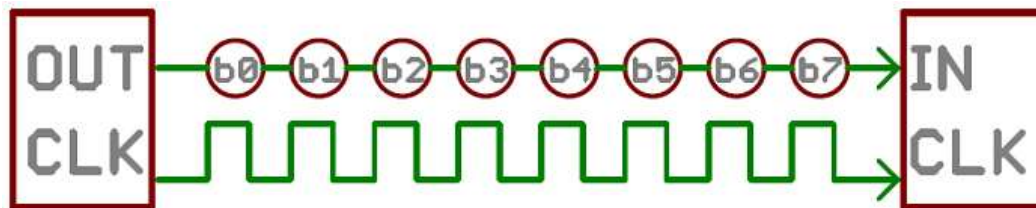
۱-۱-۱- تفاوت ارتباط سریال با موازی

در ارتباطات موازی چندین بیت را همزمان انتقال می دهند. آنها معمولاً به باس هایی (خطوط ارتباطی) بین هشت یا شانزده خط یا خطوط بیشتر، برای انتقال داده نیاز دارند. داده ها در سطوح صفر و یک منطقی منتقل می شوند.



شکل ۱-۱۱ باس داده ۸ بیتی که توسط یک کلاک کنترل می شود، در هر پالس ساعت یک بایت را انتقال می دهد و در آن ۹ سیم استفاده می شود.

در ارتباطات سریال در هر لحظه (کلاک) یک بیت از داده‌ها انتقال داده می‌شود. UART به معنای Universal Asynchronous Receiver-Transmitter که یک پروتکل ارتباط سریال است و از طریق آن داده‌ها بین دو میکروکنترلر و یا یک میکروکنترلر و کامپیوتر جا به جا می‌شوند. انتقال داده به صورت سریال و از طریق دو سیم انجام می‌شود که یکی برای دریافت داده (RX) و دیگری برای ارسال داده (TX) هستند. برای تمایز داده‌ها از یکدیگر در این پروتکل بیت‌های start bit و stop bit استفاده می‌شود.



شکل ۱۱-۲ مثالی از یک ارتباط سریال که نشان می‌دهد برای انتقال یک بیت در هر پالس ساعت فقط ۲ سیم مورد نیاز است.

ارتباط موازی قطعاً مزیت‌های خودش را دارد سریع، ساده و قابل اجرا است. اما خطوط ورودی/خروجی (I/O) بسیار بیشتری نیاز دارد.

۲-۱-۱- ارتباط سریال آسنکرون (غیرهمگام)

در طول زمان، ده‌ها پروتکل ارتباط سریال برای پاسخگویی به نیازهای خاص سامانه‌های نهفته طراحی شده‌اند. USB (باس سریال جهانی) و اترنت، دو نمونه از پروتکل‌های ارتباطی سریال محاسباتی معروف هستند. پروتکل‌های ارتباطی سریال دیگری که رایج‌اند عبارت‌اند از SPI و I2C. هر یک از این ارتباطات سریال را می‌توان به یکی از این دو گروه اختصاص داد: آسنکرون یا سنکرون.

در ارتباط آسنکرون، داده‌ها بدون نیاز به سیگنال کلاک خارجی منتقل می‌شوند. این روش انتقال برای کاهش تعداد سیم‌ها و پین‌های I/O مورد نیاز مناسب است، اما این یعنی انتقال و دریافت مطمئن‌تر داده‌ها پیچیدگی بیشتری دارد. معمولاً وقتی از ارتباط سریال صحبت می‌کنیم درواقع منظورمان پروتکل سریال آسنکرون است.

۳-۱-۱۱- قوانین ارتباط سریال

پروتکل سریال آسنکرون تعدادی قوانین داخلی دارد (مکانیزم‌هایی که به انتقال داده‌ها با سرعت مطمئنه و بدون خطا کمک می‌کند). ساز و کاری که برای حذف سیگنال کلاک خارجی مورد استفاده قرار می‌گیرند عبارت‌اند از:

- بیت‌های داده
- بیت‌های همگام‌سازی
- بیت توازن
- نرخ داده (باودریت).

از طریق انواع این مکانیزم‌های سیگنالینگ، شما متوجه خواهید شد که برای ارسال اطلاعات به صورت سریال راه‌های مختلفی وجود دارد. این پروتکل قابل تنظیم بوده و نکته مهم این است که مطمئن شویم هر دو دستگاهی که در باس سریال قرار گرفته‌اند از پروتکل‌های کاملاً مشابه استفاده می‌کنند.

نرخ داده در ارتباط سریال

نرخ داده (Baud Rate) مشخص می‌کند که اطلاعات با چه سرعتی بر روی خط سریال ارسال می‌شوند، که معمولاً با واحد بیت در ثانیه (bps) بیان می‌شود. اگر نرخ داده را تعبیر کنید، متوجه می‌شوید که چقدر طول می‌کشد تا یک بیت را منتقل کنید.

نرخ داده می‌تواند در حدود هر مقداری باشد. فقط لازم است که هر دو دستگاه با سرعت یکسان عمل کنند. یکی از رایج‌ترین نرخ‌های باند، به‌ویژه برای موارد ساده که سرعت خیلی مهم نیست، ۹۶۰۰ بیت در ثانیه است. دیگر نرخ داده‌های استاندارد ۱۲۰۰، ۲۴۰۰، ۴۸۰۰، ۱۹۲۰۰، ۳۸۴۰۰، ۵۷۶۰۰ و ۱۱۵۲۰۰ هستند.

هرچه نرخ داده بالاتر رود، داده‌ها سریع‌تر ارسال / دریافت می‌شوند، اما محدودیت‌هایی برای سرعت انتقال داده‌ها وجود دارد. شما معمولاً سرعت بیش از ۱۱۵۲۰۰ را نمی‌بینید (این مقدار برای اغلب میکروکنترلرها سرعت بالایی است. به علاوه اینکه شما شاهد خطاهایی در پایان دریافت خواهید بود، زیرا کلاک‌ها و دوره‌های نمونه‌گیری نمی‌توانند با این نرخ داده به کارشان ادامه دهند).

قالب بندی داده ها در ارتباط سریال

هر بلوک (معمولاً یک بایت) از داده در به صورت یک بسته از بیت ها منتقل می شود. قالب ها با اضافه کردن بیت همگام سازی و بیت های توازن به داده های اصلی ایجاد می شوند.



شکل ۱۱-۳ فریم سریال. برخی از نمادها دارای اندازه بیت های قابل تنظیم هستند.

بیاید به جزئیات هر یک از این قسمت های این فریم پردازیم:

دیتا

محتوای هر بسته سریال، شامل اطلاعاتی است که انتقال می دهد. به این بلوک داده تکه (chunk) میگوییم، زیرا اندازه آن مشخص نیست. مقدار داده ها در هر بسته می تواند در هر مقداری بین ۵ تا ۹ بیت تنظیم شود. مطمئناً، اندازه استاندارد داده، همان بایت ۸ بیتی شماست، اما اندازه های دیگر کارایی های دیگری را دارند. یک محتوای ۷ بیتی می تواند کارآمدتر از ۸ باشد، مخصوصاً اگر شما فقط کاراکترهای اسکی ۷ بیتی را انتقال می دهید.

بعد از توافق در مورد طول کاراکتر، هر دو دستگاه سریال باید در مورد ترتیب ارسال داده های خود به توافق برسند. آیا داده ها به ترتیب از باارزش ترین بیت (msb) به کم ارزش ترین بیت فرستاده می شوند یا برعکس؟ اگر غیر از این باشد، می توانید فرض کنید که داده ها به ترتیب از کم ارزش ترین بیت (lsb) منتقل می شوند.

بیت همگام سازی

بیت های همگام سازی دو یا سه بیت مخصوص اند که توسط هر تکه داده انتقال داده می شوند. آن ها بیت (های) شروع و بیت (های) توقف هستند. همان طور که از نامشان برمی آید، این بیت ها شروع و پایان بسته را تعیین می کنند. همیشه یک بیت شروع وجود دارد، اما تعداد بیت های توقف به یک یا دو بیت قابل تغییر است. (هرچند معمولاً یک بیت است)

بیت شروع همیشه با یک خط داده بیکار (idle) که از ۱ به ۰ تغییر می کند نشان داده می شود، در حالی که بیت (های) توقف با نگه داشتن خط در ۱ به حالت بیکار (idle) بازمی گردند.

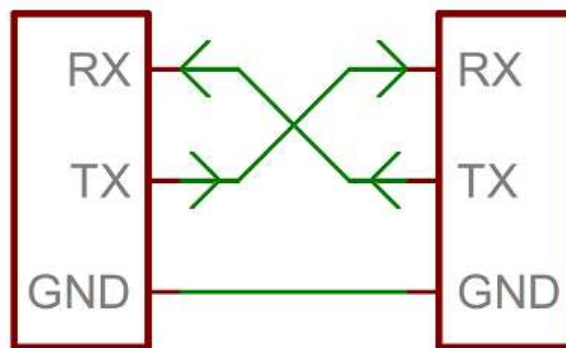
بیت پرییتی (parity bit)

Parity یک شکل بسیار ساده و سطح پایین از کنترل خطا است که در دو نوع وجود دارد: زوج یا فرد. برای تولید بیت توازن (parity bit)، تمام ۵-۹ بیت از بایت داده اضافه می‌شوند و درستی مجموع این بیت‌ها تصمیم می‌گیرد که آیا بیت تنظیم شده است یا نه. به عنوان مثال، اگر پرییتی روی زوج تنظیم شده باشد و به یک بایت داده مانند b01011101^۰ اضافه شده باشد که دارای تعداد فردی از ۱ است (۵ بار)، بیت پرییتی باید ۱ تنظیم شود. برعکس، اگر حالت پرییتی بر روی فرد تنظیم شود، بیت توازن ۰ است. پرییتی اختیاری است و به طور گسترده‌ای مورد استفاده قرار نمی‌گیرد. بیت پرییتی می‌تواند در انتقال رسانه‌های با نویز زیاد مفید باشد، اما انتقال داده‌ها را کمی کند می‌کند و برای تشخیص خطا هم به فرستنده و هم به گیرنده نیاز دارد (معمولاً داده‌های دریافتی ناموفق باید دوباره ارسال شوند).

۴-۱-۱۱- سخت افزار ارتباط سریال

یک باس سریال فقط دو سیم دارد (یکی برای ارسال اطلاعات و دیگری برای دریافت). به همین ترتیب، دستگاه‌های سریال باید دارای دو پین سریال باشند:

- گیرنده RX
- فرستنده TX



شکل ۴-۱۱- توجه کنید که برچسب‌های RX و TX متناسب با خود دستگاه‌ها باشند. بنابراین RX از یک دستگاه باید به TX یک دستگاه دیگر متصل شود. منطقی است که فرستنده باید با گیرنده صحبت کند نه فرستنده دیگری.

یک رابط سریال قرار گرفته بین دو دستگاهی که هردوی آن‌ها داده‌ها را ارسال و دریافت می‌کنند، می‌تواند

به صورت کاملاً دوطرفه یا نیمه دوطرفه باشد.

- کاملاً دوطرفه یعنی هر دو دستگاه می توانند به طور همزمان ارسال و دریافت انجام دهند. (FULL-DUPLEX)
- ارتباط نیمه دوطرفه به این معنی است که دستگاه های سریال باید نوبتی ارسال و دریافت کنند. (HALF-DUPLEX)
- برخی از باس های سریال ممکن است فقط یک اتصال بین دستگاه فرستنده و گیرنده باشند. به عنوان مثال، LCD های فعال شده سریال (یک نوع ماژول نمایشگر با ارتباط سریال) فقط دریافت می کنند و هیچ اطلاعاتی برای بازگرداندن به دستگاه کنترلی ندارند. این چیزی است که به عنوان ارتباطات سریال ساده شناخته می شود. همه چیزی که نیاز دارید تنها یک سیم از فرستنده دستگاه master، به خط RX شنونده است. (SIMPLEX)

۲-۱۱- آزمایش

۱-۲-۱۱- برقراری ارتباط بین دو میکروکنترلر

برای این که داده ای را بین دو میکروکنترلر از طریق ارتباط سریال آسنکرون جا به جا کنیم لازم است که مطابق شکل ۱۱-۴، TX یک میکروکنترلر به RX میکروکنترلر دیگر متصل شود چون که داده ارسالی یکی درواقع داده دریافتی دیگری است و همچنین برای هم مبنا کردن دو میکروکنترلر زمین های آنها را به یکدیگر متصل کنیم و در نهایت هم باید Baud Rate دو میکروکنترلر یکسان باشد.

برنامه ای بنویسید که رشته ی ذخیره شده در یک میکروکنترلر را به کمک پروتکل UART برای میکروکنترلر دیگری ارسال کند. در نهایت میکروکنترلر دریافت کننده، رشته ی دریافتی را روی نمایشگر کاراکتری نشان دهد.

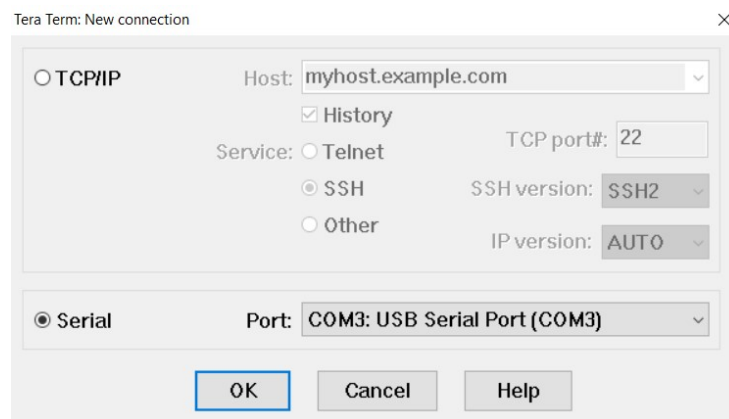
۲-۲-۱۱- برقراری ارتباط بین میکروکنترلر و کامپیوتر

برای اتصال میکروکنترلر و کامپیوتر لازم است که از یک مازول واسط USB-to-Serial استفاده کنیم که در مدل‌های مختلفی وجود دارد. پس از اتصال آن به کامپیوتر قسمت Device Manager کامپیوترتان را بررسی کنید که این مازول را شناخته باشد و اگر شناخته بود به صورت آنلاین یا با دانلود و نصب درایور از لینک مربوطه آن مازول را به کامپیوترتان بشناسانید تا به آن یک پورت اختصاص دهد که داده‌ها از طریق همین پورت بین کامپیوتر و میکروکنترلر جا به جا شوند.



شکل ۱۱-۵ مازول مبدل USB به سریال TTL تراشه CH340G

پس از نصب درایور بر روی کامپیوتر برای نمایش داده‌های ارسالی می‌توانید از نرم افزار متن باز TeraTerm استفاده کنید.

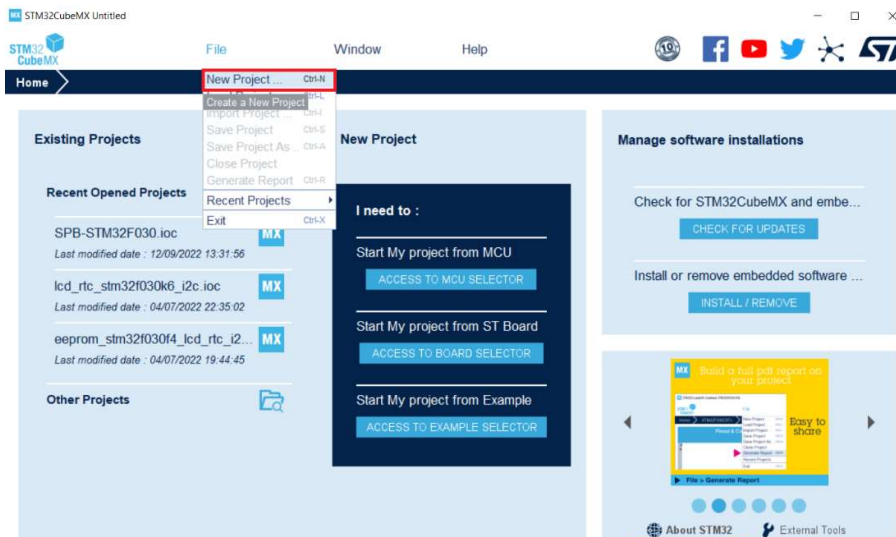


شکل ۱۱-۶ پنجره تنظیم اولیه در نرم افزار TeraTerm

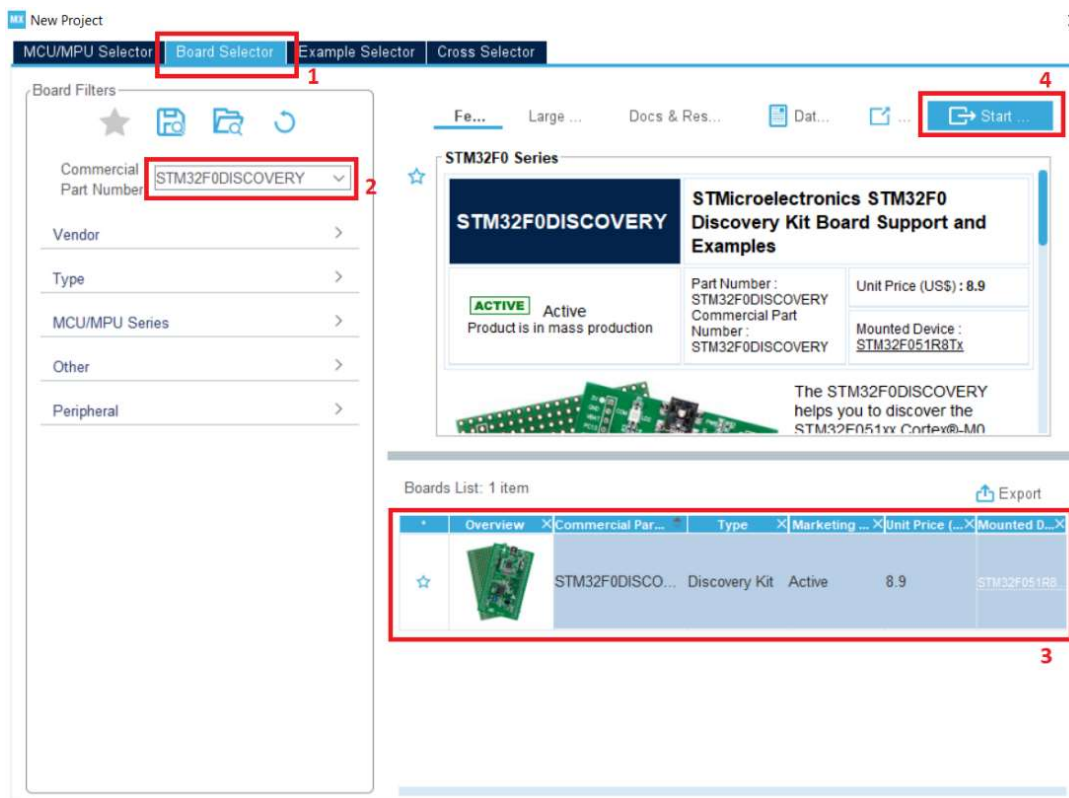
برنامه‌ای بنویسید که دو عدد و یک عملگر از رایانه به میکرو ارسال شده و پس از پردازش در میکروکنترلر حاصل به کمک UART به رایانه بازگردانده شده و در ترمینال مجازی نمایش داده شود.

۱۱-۳- تنظیمات نرم افزاری و برنامه نویسی

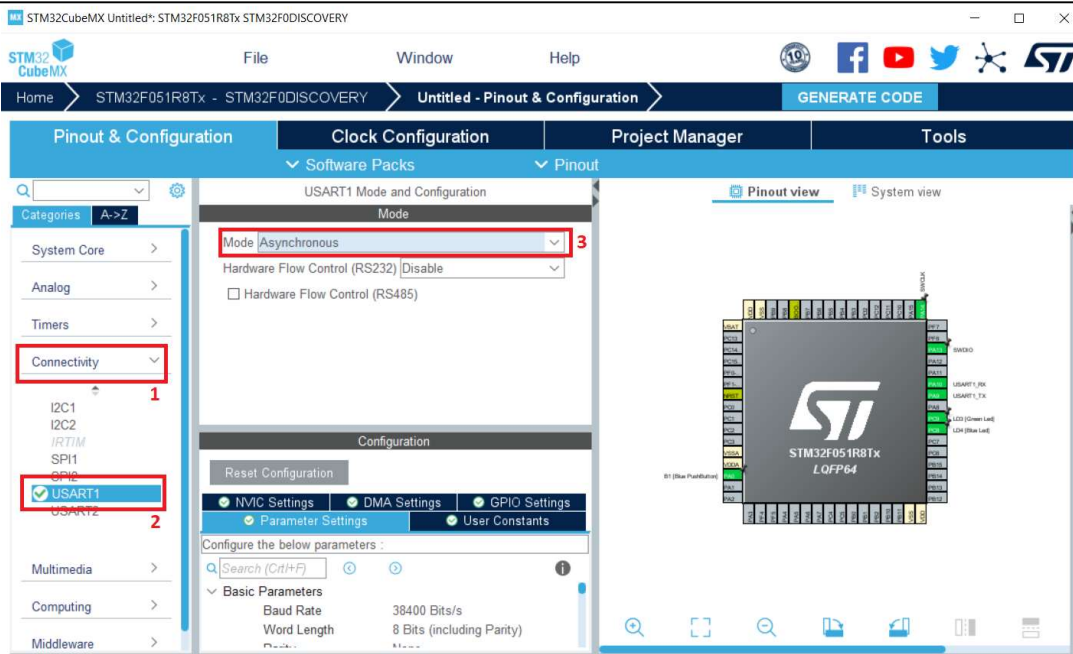
برای تنظیمات UART ابتدا یک پروژه جدید ایجاد می کنیم:



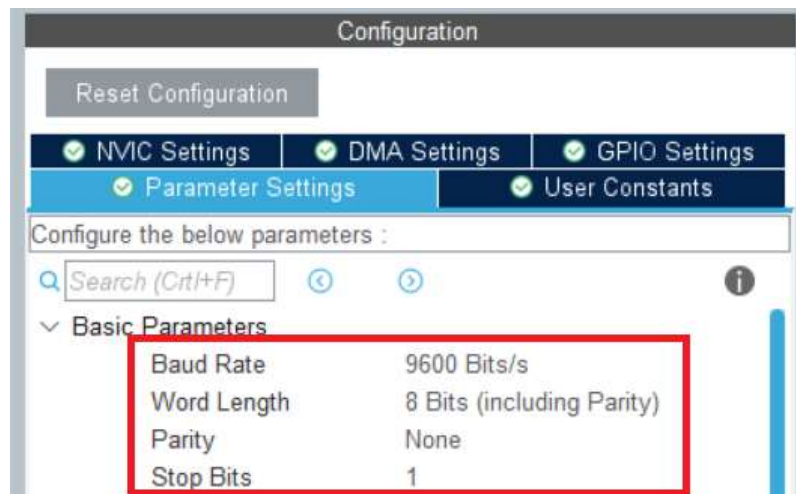
برد یا میکروکنترلر مورد نظر را انتخاب می کنیم:



سپس USART1 را در مد آسنکرون فعال می کنیم:



برای تنظیمات UART لازم است که Baud Rate و دیگر پارامترهایی که در درسنامه مطرح شد را به درستی مشخص کنیم:



پس از generate کردن کد در نرم افزار Stm32CubeMX و Import پروژه در نرم افزار CubeIDE شروع به کدنویسی می‌کنیم. برای ارسال اطلاعات:

```
/* USER CODE BEGIN 2 */

uint8_t data_tx[100] = "This is UART Test!\r\n" ;
HAL_UART_Transmit(&huart1,data_tx,strlen(data_tx),HAL_MAX_DELAY);

/* USER CODE END 2 */
```

و برای دریافت اطلاعات:

```
/* USER CODE BEGIN 2 */

uint8_t data_rx[100];
HAL_UART_Receive(&huart1, data_rx, NUMBER_OF_BYTES, HAL_MAX_DELAY);

/* USER CODE END 2 */
```

در نرم افزار TeraTerm از قسمت Setup – Serial Port ... تنظیمات را مطابق شکل انجام می دهیم.

۴-۱۱- پرسش ها و تمرین های برنامه نویسی

۱- برنامه ای بنویسید که در آن وقفه ی UART1 برای دریافت و ارسال داده فعال شود. در این برنامه

هر کاراکتری که دریافت می گردد عیناً مشابه آن اکو می شود.

۲- برنامه ای بنویسید که یک دیتای ۱۶ بیتی از را از پورت A بخواند (با ۱۶ عدد DIP Switch بتوان دیتا را تغییر داد) و به خروجی سریال بفرستد. نرخ ارسال را ۹۶۰۰ در نظر بگیرید و خروجی روی نمایشگر رایانه نشان داده شود.

۳- برنامه ای بنویسید که بتوان از طریق نرم افزار MATLAB دور موتور یک موتور DC را به کمک پروتکل UART کنترل نمود.

آزمایش ۱۲:

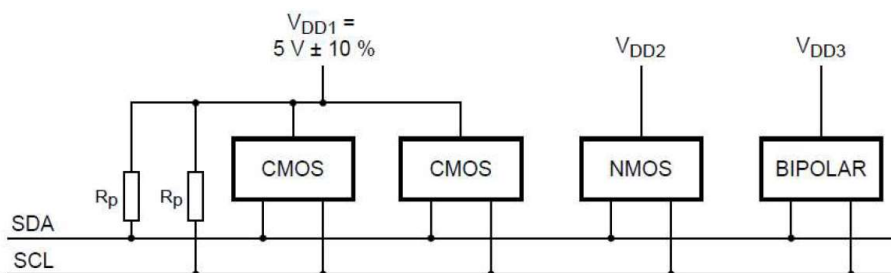
پروتکل ارتباطی I²C

۱-۱۲- درسنامه

پروتکل I^2C یک پروتکل ارتباطی سریال است که داده ها بیت به بیت در طول یک (خط SDA) منتقل می شوند. I^2C بهترین امکانات پروتکل های SPI و UART را ترکیب می کند. با I^2C ، می توانید چندین Slave را به یک Master متصل نمود (مانند SPI) و می توان چندین Master داشت که یک یا چندین Slave را کنترل می کنند. به عنوان مثال زمانی که لازم است تا بیش از یک میکروکنترلر اطلاعات را در کارت حافظه ثبت کند یا متن را در یک LCD نمایش دهند، پروتکل I^2C واقعاً مفید است. I^2C فقط از دو سیم برای انتقال داده ها بین دستگاه ها استفاده می کند:

SDA (داده های سریال) - خطی برای ارسال و دریافت داده توسط master و slave

SCL (کلاک سریال) - خطی که حامل سیگنال کلاک است.



V_{DD3} , V_{DD2} به دستگاه متصل بستگی دارند به عنوان مثال می توانند برابر ۱۲ ولت باشند.

یک ویژگی I^2C این است که هر دستگاه روی باس باید به صورت Open-Drain به سیگنال کلاک (به اختصار SCL) و سیگنال داده (به اختصار SDA) متصل شود.

در اینجا دو مفهوم مهم از این پیکربندی باس Open-Drain شرح داده می شود:

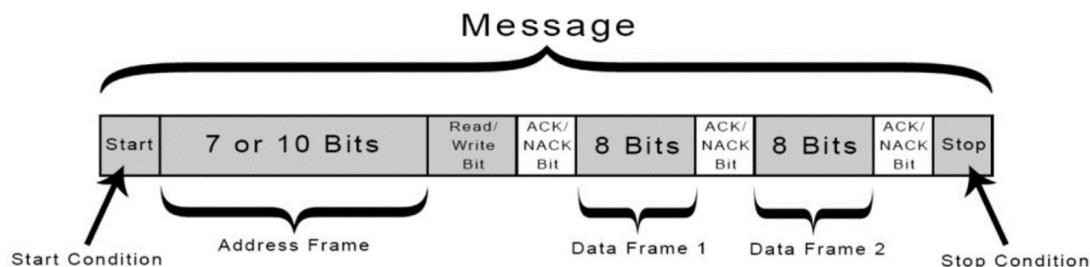
هر دستگاهی در باس می تواند با خیال راحت سیگنال ها را به سطح منطقی صفر برساند، حتی اگر دستگاه دیگری سعی در بالا بردن سطح سیگنال ها داشته باشد. این اساس ویژگی «همگام سازی کلاک» یا «کشش کلاک» I^2C است. Master کلاک سریال را تولید می کند اما در صورت لزوم یک دستگاه Slave می تواند SCL را پایین نگه دارد و در نتیجه فرکانس کلاک را کاهش دهد.

دستگاه هایی با ولتاژهای تغذیه متفاوت می توانند در همان باس وجود داشته باشند، تا زمانی که دستگاه

های ولتاژ پایینتر در اثر ولتاژ بالاتر آسیب نبینند. به عنوان مثال یک دستگاه ۳.۳ ولت می تواند با یک دستگاه ۵ ولت ارتباط برقرار کند.

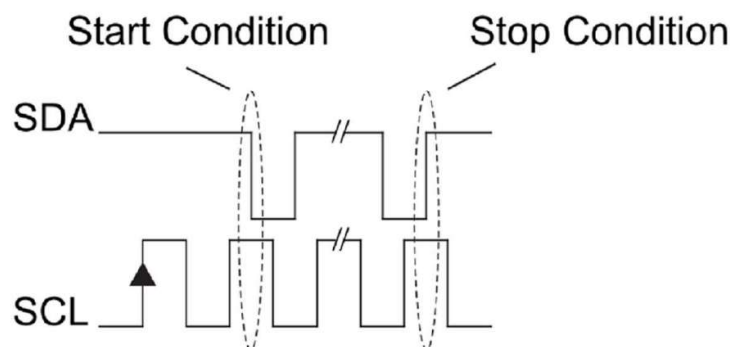
۱-۱-۱۲- عملکرد I²C

با I²C، داده ها در قالب پیام منتقل می شوند. پیامها به فریم هایی از داده ها تقسیم می شوند. هر پیام دارای یک بخش آدرس است که حاوی آدرس باینری Slave است و یک یا چند فریم داده حاوی داده های در حال انتقال است. این پیام همچنین شامل شرایط شروع و توقف بیتی خواندن و نوشتن و بیت های ACK/NACK بین هر فریم داده است:



شرط شروع: زمانی که سطح ولتاژ خط SCL برابر یک منطقی است خط SDA از سطح ولتاژ بالا به سطح ولتاژ پایین تغییر وضعیت می دهد.

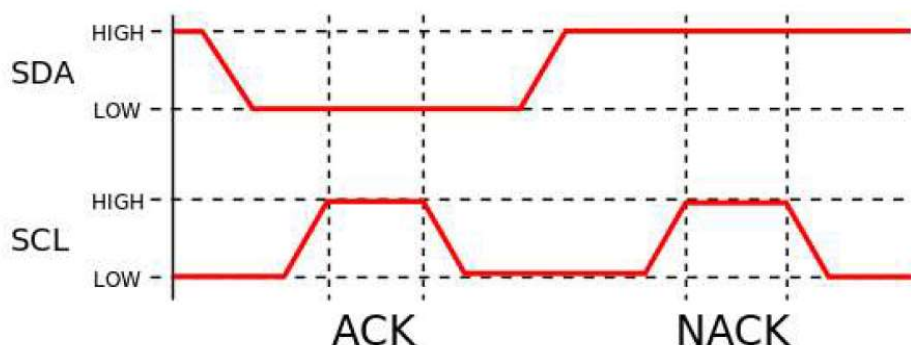
شرط توقف: زمانی که سطح ولتاژ خط SCL برابر یک منطقی است خط SDA از سطح ولتاژ پایین به سطح ولتاژ بالا تغییر وضعیت می دهد.



فریم آدرس: یک دنباله ۷ یا ۱۰ بیتی منحصر به فرد برای هر Slave، زمانی که master می خواهد با او صحبت، کند آدرس Slave را مشخص می کند.

بیت Read/Write: یک بیت واحد که مشخص می‌کند آیا Master داده‌ها را به Slave ارسال می‌کند (Write- سطح ولتاژ پایین) یا درخواست داده از آن دارد (Read- سطح ولتاژ بالا).

بیت ACK/NACK: هر فریم در یک پیام با یک بیت تایید یا عدم تایید دنبال می‌شود. اگر یک فریم آدرس یا فریم داده با موفقیت دریافت شد یک بیت ACK از دستگاه دریافت‌کننده به فرستنده بازگردانده می‌شود (صفر منطقی) اگر پیام به درستی دریافت نشده باشد یک بیت NACK ارسال می‌شود (یک منطقی).

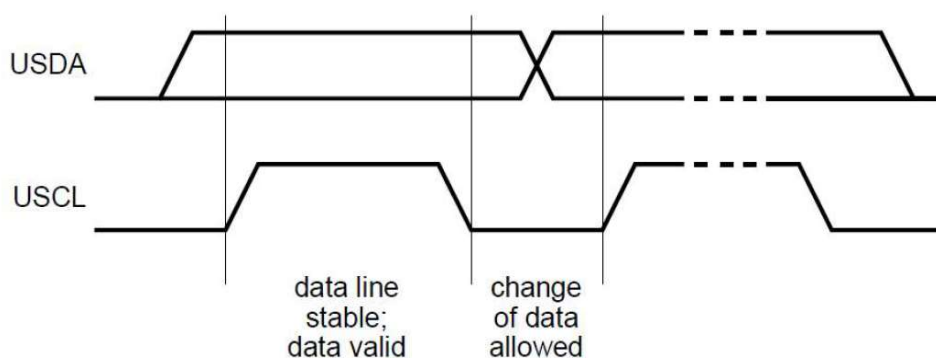


در جدول زیر چهار مد ممکن برای I²C را مشاهده می‌کنید:

Mode	Data Rate	Notes
Standard Mode	Up to 100Kbps	Supported by stm32F4
Fast Mode	Up to 400Kbps	Supported by stm32F4
Fast Mode+	Up to 1Mbps	Supported by some stm32F4
High Speed Mode	Up to 3.4 Mbps	Not Supported by stm32F4

نکته: دستگاه‌های که در مد Standard کار کنند قادر نیستند در باس با مد Fast ارتباط برقرار کنند اما دستگاه‌هایی که در مد Fast کار می‌کنند قادر هستند در یک باس با مد Standard به درستی عمل کنند.

نکته: دیتا در خط SDA تنها زمانی می‌تواند تغییر کند که خط SCL صفر منطقی (LOW) باشد.



۱۲-۲- آزمایش

۱۲-۲-۱- درایور نمایشگر کاراکتری با ماژول I²C

درایوری که برای نمایشگر کاراکتری نوشته بودید را به نحوی بازنویسی کنید که دیتا و دستور با استفاده از ماژول مبدل I²C به نمایشگر ارسال شود.

۱۲-۳- تنظیمات نرم‌افزاری و برنامه نویسی

به دیتا شیت ماژول مبدل ال‌سی‌دی به I²C مراجعه کنید.

۱۲-۴- پرسش‌ها و تمرین‌های برنامه نویسی

۱- به کمک سنسور SHT10 دما و رطوبت محیط را اندازه‌گیری کرده و روی نمایشگر نشان دهید.