

موقاله العلیم

جزوه دوره میکروکنترلرهای STM32 سری ARM



درس: رضا فتاحی

مقدمه:

مخف ARM Advanced RISC Machines یک شرکت انگلیسی نیمه هادی و طراحی نرم افزار مستقر در کمبریج، انگلستان است که تجارت اصلی آن طراحی پردازنده های ARM (CPU) است. همچنین تراشه های دیگری را طراحی می کند، ابزارهای توسعه نرم افزار را تحت برندهای DS-5 و Keil و RealView فراهم می کند.

در حالی که CPU های ARM برای اولین بار در Acorn Archimedes کامپیوتر رومیزی ظاهر شدند، سیستم های امروزی عمدهاً شامل سیستم های تعییه شده، از جمله CPU های ARM هستند که تقریباً در تمام گوشی های هوشمند مدرن استفاده می شوند.

STM32

یک خانواده از مدارهای مجتمع میکروکنترلر ۳۲ بیتی است که توسط STMicroelectronics ساخته شده است. تراشه های STM32 در سری های مرتبط گروه بندی می شوند که بر اساس همان هسته پردازنده Cortex-, Cortex-M7، Cortex-M4، Cortex-M3، Cortex-M0+، Cortex-M0 هستند: 32 بیتی هستند. داخل، هر میکروکنترلر از هسته (های) ARM، حافظه فلاش، رم استاتیک، رابط اشکال زدایی و تجهیزات جانبی مختلف تشکیل شده است.

جدول زیر خانواده میکروکنترلهای STM32 را خلاصه می کند.

STM32 series	ARM CPU core(s)		STM32 series	ARM CPU core(s)
F0	Cortex-M0		WB, WL	M4 & M0+ (2 core)
C0, G0, L0	Cortex-M0+		F7	Cortex-M7
F1, F2, L1	Cortex-M3		H7	M7 (1 core), M7 & M4 (2 core)
F3, F4, G4, L4, L4+	Cortex-M4		H5, L5, U5, WBA	Cortex-M33

دسته بندی میکروکنترلر های ARM بر اساس کاربرد:

پردازنده‌های Cortex-A : مناسب طیف وسیعی از دستگاه‌هایی هستند که از یک سیستم عامل مانند لینوکس یا اندروید استفاده می‌کنند و در طیف گسترده‌ای از برنامه‌ها از گوشی‌های ارزان قیمت گرفته تا گوشی‌های هوشمند، رایانه‌های لوحی، ستتاپ باکس‌ها و استفاده می‌شوند.

پردازنده‌های Cortex-R (Real-time applications) : برنامه‌های بلادرنگ با کارایی بالا مانند کنترل کننده‌های دیسک سخت (یا کنترل کننده‌های درایو حالت جامد SSD)، تجهیزات شبکه و چاپگرهای در بخش سازمانی، دستگاه‌های مصرف‌کننده مانند پخش‌کننده‌های بلوری و پخش‌کننده‌های رسانه، و همچنین خودروسازی را هدف قرار می‌دهند.

پردازنده‌های Cortex-M (Microcontroller core) : مناسب ترین نوع این دسته از میکروها برای طراحی و ساخت دستگاه‌هایی هستند که از سیستم عامل‌های نهفته embedded استفاده می‌کنند همچنین برای کاربردهای کم‌هزینه و کم‌صرف بسیار مناسب هستند. این دوره و جزوی آموزشی با تمرکز بر روی این نوع پردازنده‌ها طراحی شده است.

نرم افزار:

پس از طراحی و ساخت سخت افزار نوبت به نوشتن نرم افزار برای راه اندازی مدار می‌رسد. چهار روش متداول برای نوشتن برنامه عبارتند از:

- ۱- Register
- ۲- توابع SPL (Standard peripheral libraries)
- ۳- توابع LL (Low Layer)
- ۴- توابع HAL (Hardware Abstraction Layer)

Register

در این روش هیچ تابعی وجود ندارد و لازم است برنامه نویس اطلاعات جامعی از ساختار داخلی میکرو و نحوه پیکربندی رجیسترها داشته باشد. در این روش با استفاده از امکانات موجود در CMSIS رجیسترها لازم را مقدار دهی کرده و برنامه مورد نظر را می‌نویسیم. CMSIS یک لایه نرم‌افزاری برای سخت‌افزار پردازنده‌های

کورتکس M می‌باشد که توسط بنیاد ARM و فارغ از شرکت سازنده میکروکنترلر تعریف شده است. این روش برای افراد مبتدی توصیه نمی‌گردد.

CMSIS: Cortex Microcontroller Software Interface Standard

(Standard peripheral libraries)SPL توابع

مجموعه‌ای از کتابخانه‌های ایجاد شده توسط شرکت ST است که قبل از معرفی توابع LL و HAL برای ساده تر کردن برنامه نویسی و عدم درگیر شدن برنامه نویسان با رجیستر های میکرو معرفی شد . با استفاده از توابع SPL و معرفی آن‌ها به کامپایلر بصورت دستی می‌توان قابلیت های مد نظر برنامه نویس را به برنامه اضافه نمود. برای هر بخش از میکرو کنترلر توابعی آماده برای کار با آن بخش از میکرو در نظر گرفته شده که به صورت جامع برای تمام میکروکنترلر های خانواده ST قابل استفاده است که تبدیل برنامه نوشته شده را برای سری های دیگر تسهیل می‌کند . با توجه به این که در حال حاضر استفاده از نرم افزار Cube توسعه شرکت ST پیشنهاد می‌گردد ، توابع SPL دیگر به روز رسانی نشده و کارآمدی لازم را ندارد.

(Low Layer)LL توابع

این توابع درایورهای لایه پایین خدمات سخت افزاری را بر اساس ویژگی‌های موجود لوازم جانبی STM32 ارائه می‌دهند. در این روش دیگر به صورت مستقیم با رجیسترها درگیر نخواهیم شد و فقط از توابعی به نام توابع LL استفاده خواهیم کرد این توابع با توجه به آرگومان‌های ورودی‌شان، رجیسترها را مقداردهی می‌کنند. با فراخوانی هر کدام از این توابع عملابخشی از یک واحد جانبی راهاندازی می‌شود. برای این‌که یک واحد جانبی به طور کامل راهاندازی شود باید با یک توالی خاص از چندین تابع LL استفاده کنیم.

(Hardware Abstraction Layer)HAL توابع

HAL بخشی از مجموعه ابزار STM32CubeIDE است که به کاربران این امکان را می‌دهد تا کدهای مختلف را برای دستگاه‌های جانبی STM32 به طور خودکار تولید کنند. HAL را می‌توان به عنوان یک کتابخانه بسیار انتزاعی در نظر گرفت که تقریباً بین پردازنده‌های STM32 فرآگیر شده است. این توابع از سطح رجیستر فاصله بیشتر و در بالاترین سطح ممکن برنامه نویسی قرار دارند و عملابه ندرت از رجیسترها استفاده می‌شود، اگرچه دسترسی به رجیسترها در این سطح بدون هیچ مشکلی ممکن است و با استفاده از ماکروهایی می‌توانیم به رجیسترها دسترسی داشته باشیم. وقتی که از یک تابع HAL استفاده می‌کنیم علاوه بر اینکه خودش رجیسترها را دسترسی داشته باشیم.

یک بخش را تنظیم می‌کند، در اکثر موارد بسیاری از کارهایی که در توابع LL باید خود کاربر انجام می‌داد را نیز انجام می‌دهد. توابع HAL می‌تواند ابزار بسیار مفیدی باشد اما نقاط ضعفی نیز دارد. از جمله افزایش مقدار کد به دلیل افزونگی و ضعف در عملکرد پرسرعت است.

مقایسه توابع HAL با توابع LL

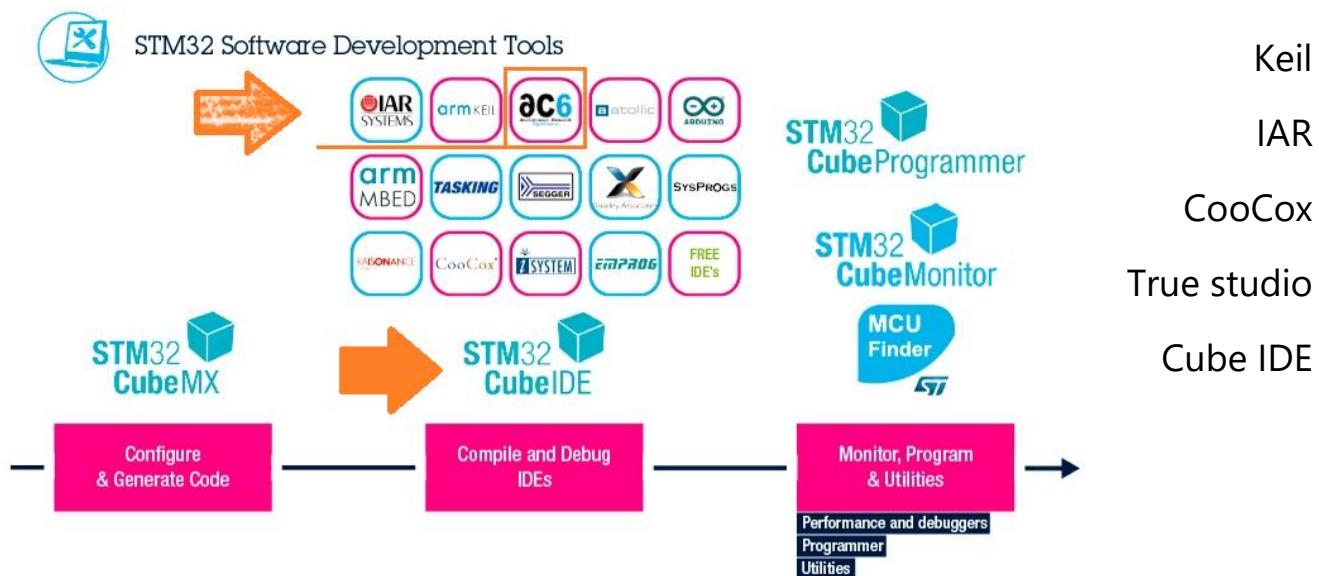
HAL API ها سطح بالا و ویژگی گرا را با سطح حمل بالا ارائه می‌دهد. آنها MCU و پیچیدگی محیطی را برای کاربر نهایی پنهان می‌کنند. LL API ها سطح پایینی را در سطح ثبات ارائه می‌دهد، با بهینه سازی بهتر اما قابلیت حمل کمتر.

: STM32CubeMX

یک ابزار جامع برای پیکربندی اجزای مختلف میکروکنترلرهای شرکت ST می‌باشد که توسط خود این شرکت توسعه داده شده و به صورت گرافیکی اجازه می‌دهد از طریق یک فرآیند گام به گام به پیکربندی بسیار آسان میکروکنترلرها و ریزپردازندههای STM32 و همچنین تولید کد اولیه C مربوطه برای هسته Cortex-M اقدام شود.

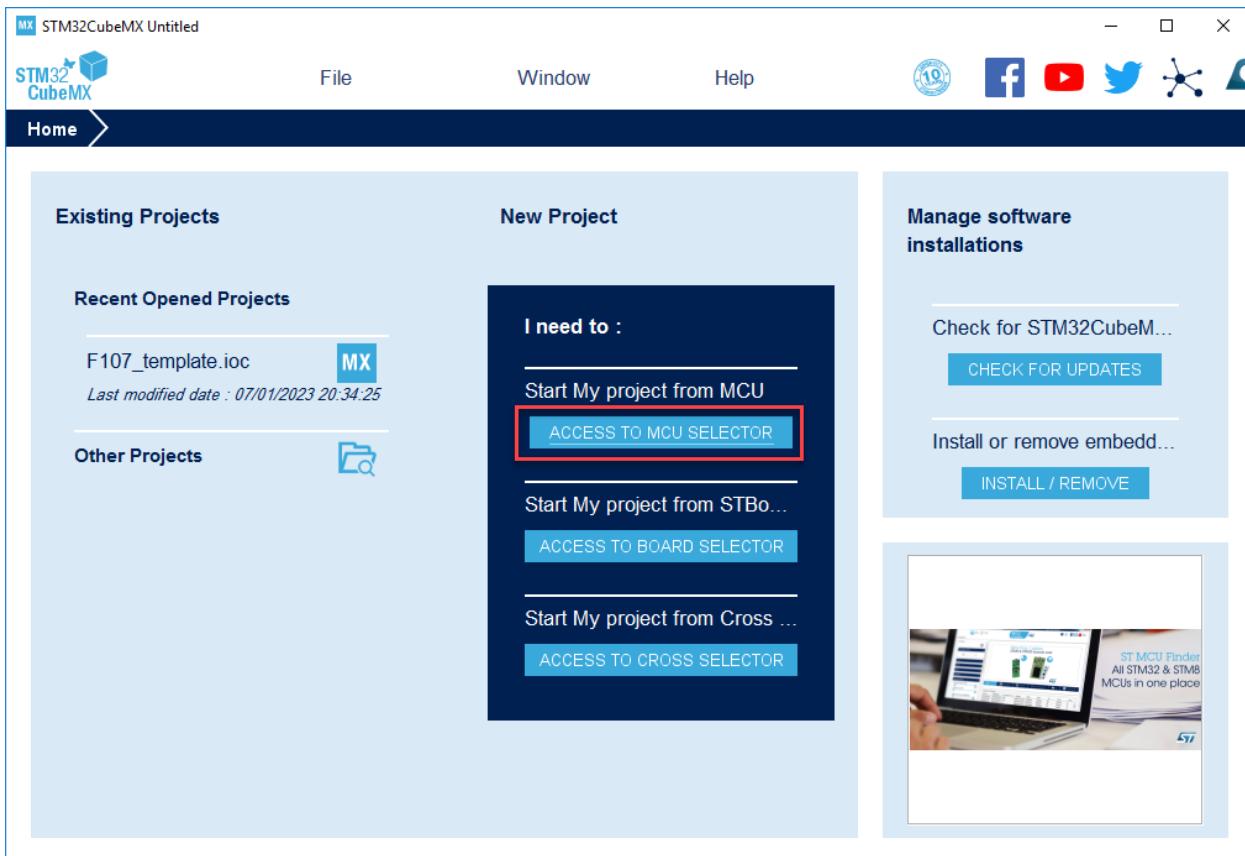
محیط توسعه نرم افزار برای میکروهای ST :

برای توسعه نرم افزار در میکروهای STM32 نیاز به یک محیط توسعه نرم افزار می‌باشد که از معروف ترین آن‌ها می‌توان به موارد زیر اشاره کرد.

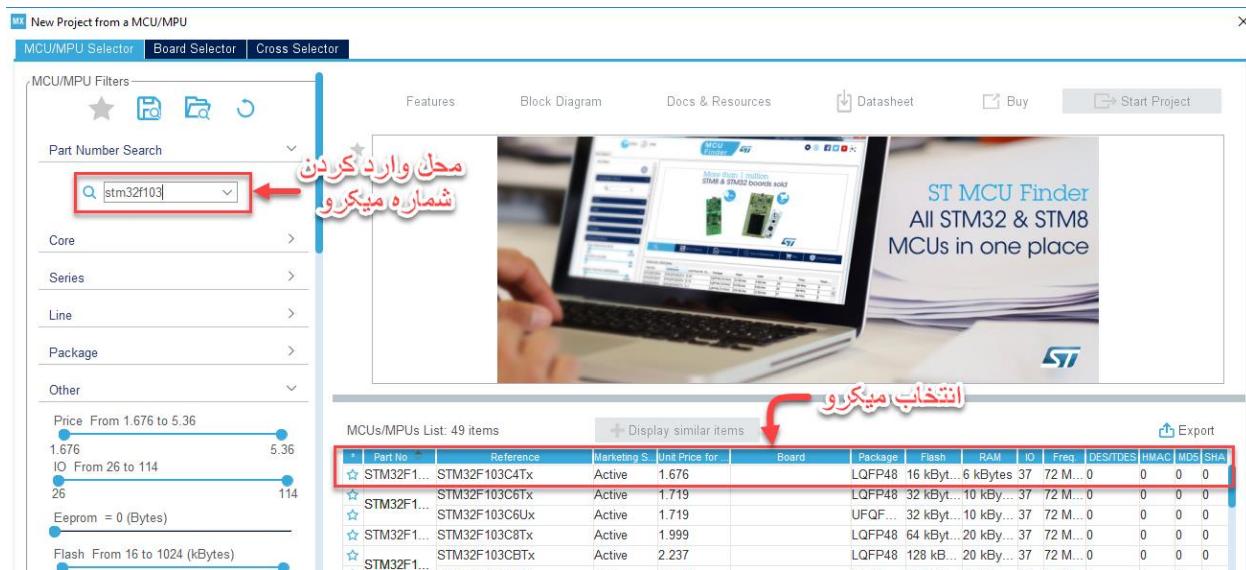


مراحل ایجاد پروژه در STM32CubeMX

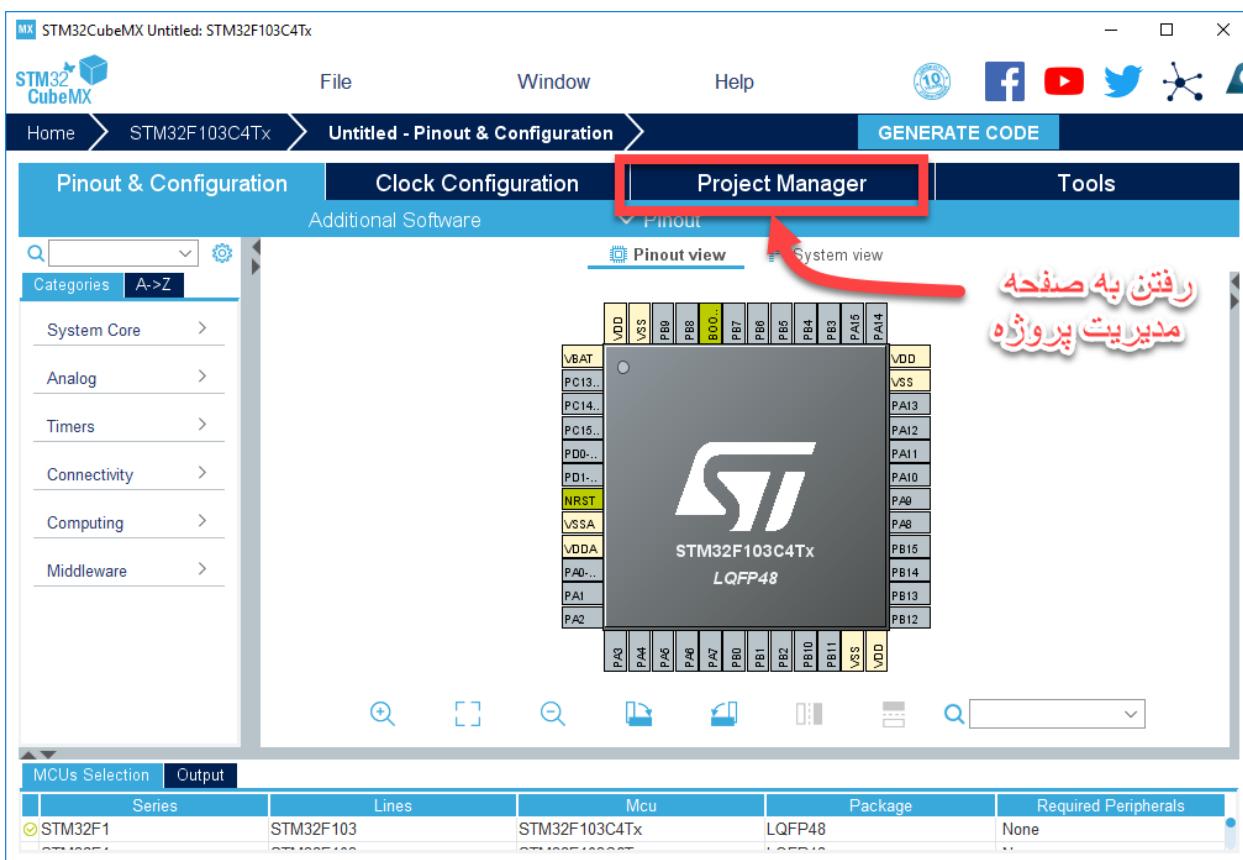
- ۱- پس از باز کردن برنامه Cube ، با کلیک بر روی گزینه Access to MCU SELECTOR وارد صفحه انتخاب نوع میکرو می شویم .



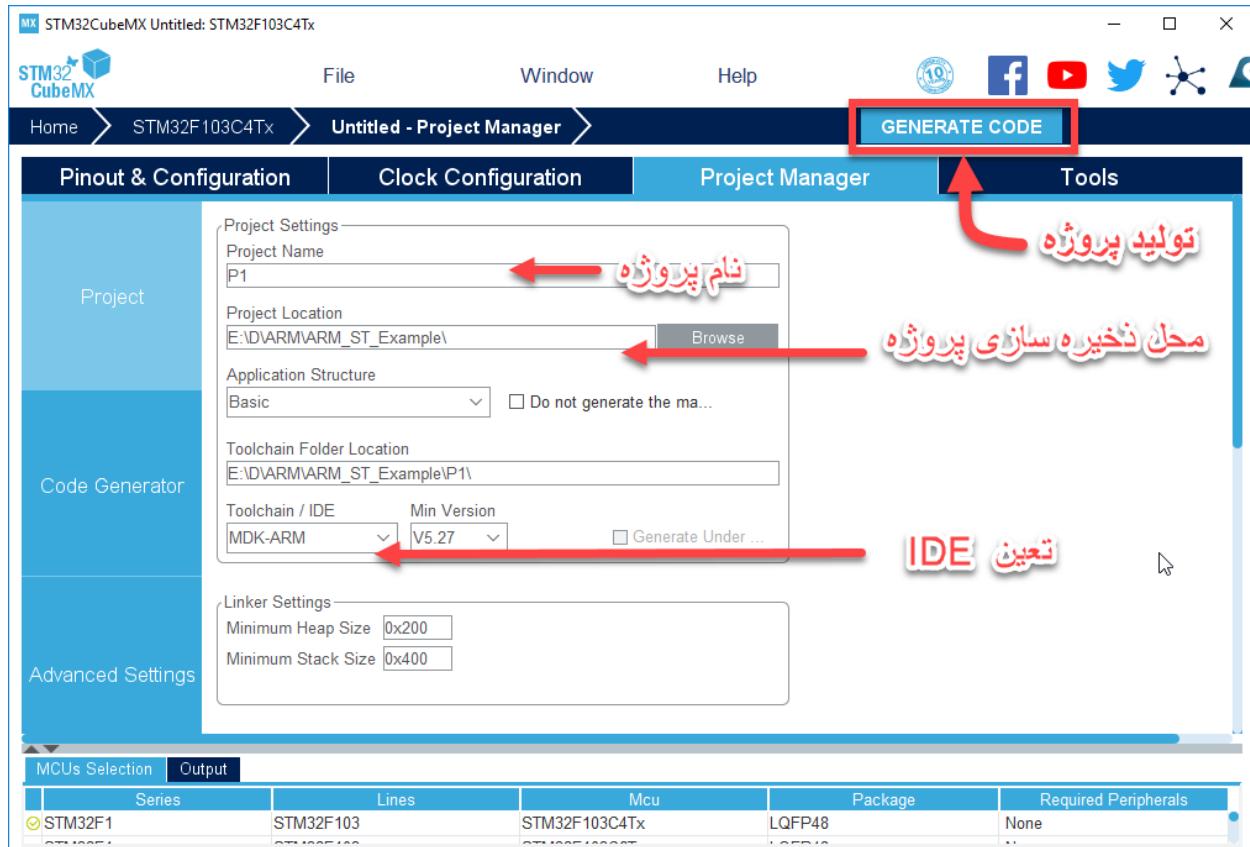
- ۲- در صفحه New Project from a MCU با تایپ شماره کامل یا بخشی از شماره میکرو ، لیستی از میکروها به نمایش در می آید، که از بین آنها میکرو مورد نظرتان را با کلیک انتخاب کنید. برای مثال فرض کنید می خواهیم با میکروی STM32F103C4 پروژه ای را شروع کنیم.



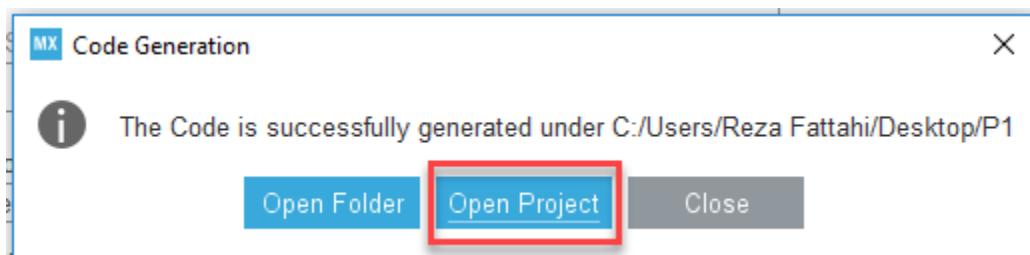
۳- در مرحله بعد صفحه‌ای باز می‌شود که می‌توانید با توجه به پروژه بخش‌های مختلف را انتخاب و پیکر بندي مناسب را برای میکرو انجام دهید. در آخر با کلیک بر روی Project Manager وارد صفحه مدیریت پروژه شوید.



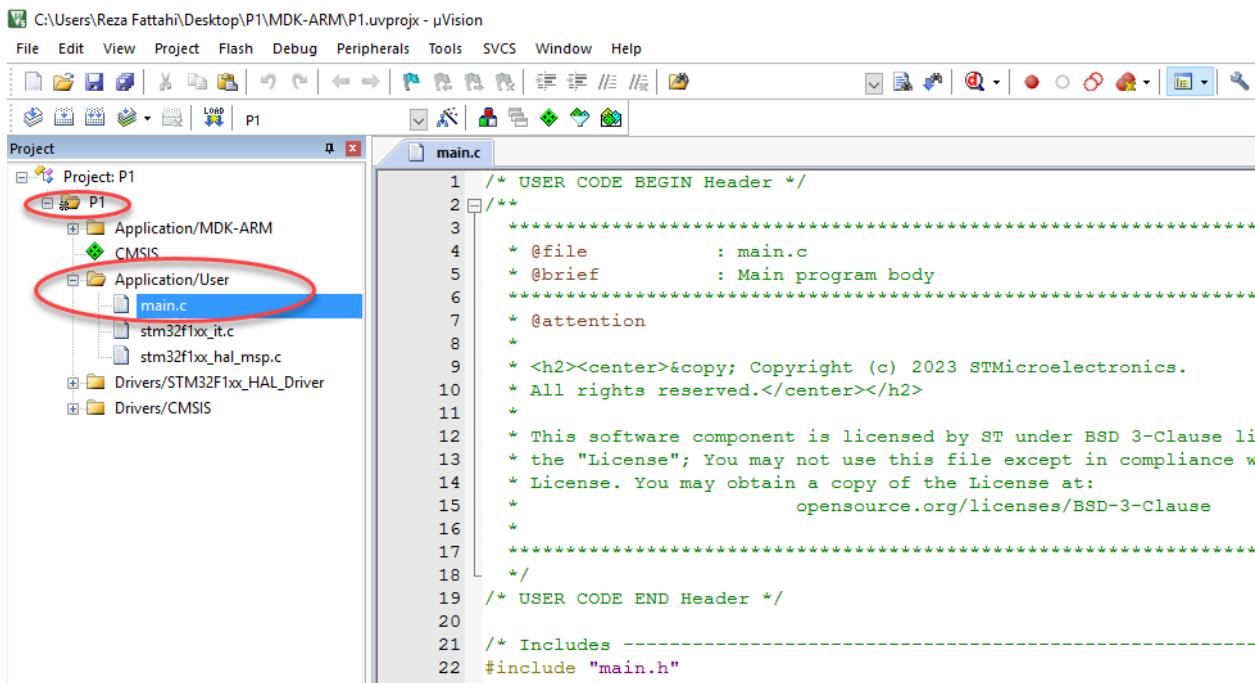
۴- در این صفحه می‌توانید نام پروژه، محل ذخیره سازی و نوع IDE که قرار است پروژه برای آن ساخته شود را مشخص نمایید. توجه داشته باشید ما برای IDE استفاده می‌کنیم (MDK-ARM) در آخر با کلیک بر روی GENERATE CODE کدهای لازم برای پروژه با توجه به IDE انتخاب شده آماده می‌شود.



۵- اگر در روند ساخت پروژه مشکلی پیش نیاید پنجره زیر باز خواهد شد که با انتخاب کدهای تولید شده در keil IDE آماده ویرایش و اضافه کردن کد کاربر خواهد شد.



۶- در پنجره Project و مسیر مشخص شده در تصویر زیر می‌توانید فایل main را باز و کدهای لازم را در جای مناسب اضافه کنید.



نوشتن برنامه :

هنگام نوشتن برنامه ، بهتر است هر بخش از کد را در محل مناسب که در فایل‌ها با comment مشخص شده بنویسید. این کار باعث می‌شود تا اگر در برنامه Cube تغییرات جدیدی ایجاد کنید این تغییرات به کدها اعمال شود و کدهای قبلی که نوشته‌اید نیز باقی بماند، در غیر این صورت کدهای قبلی شما پاک خواهد شد. در شکل‌های زیر محل قرار گرفتن کدها نمایش داده شده است.

```
22 #include "main.h"
23
24 /* Private includes -----
25 /* USER CODE BEGIN Includes */
26 هدرهای اضافه شده توسط کاربر
27
28 /* USER CODE END Includes */
29
30 /* Private typedef -----
31 /* USER CODE BEGIN PTD */
32 تعریف نام جدید برای انواع داده
33
34 /* USER CODE END PTD */
35
36 /* Private define -----
37 /* USER CODE BEGIN PD */
38 تعریف define
39
40 /* USER CODE END PD */
```

```

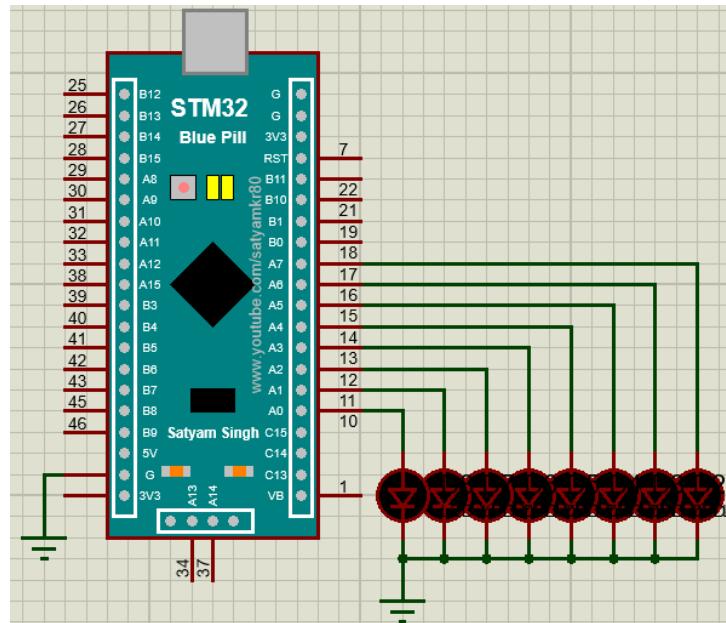
50  /* USER CODE BEGIN PV */
51
52  معرفی متغیر ها
53  /* USER CODE END PV */
54  /* Private function prototypes -----
55 void SystemClock_Config(void);
56 /* USER CODE BEGIN PFP */
57  معرفی و تعریف تابع های برنامه نویس
58
59  /* USER CODE END PFP */
60  /* Private user code -----
61 /* USER CODE BEGIN 0 */
62  کدهای مورد نظر کاربر
63
64  /* USER CODE END 0 */
65
66 /**
67  * @brief  The application entry point.
68  * @retval int
69 */
70 int main(void)

92  /* Initialize all configured peripherals */
93  /* USER CODE BEGIN 2 */
94  کدهای کاربر که باید یک بار قبل از کد اصلی اجرا شود
95
96  /* USER CODE END 2 */
97  /* Infinite loop */
98  /* USER CODE BEGIN WHILE */
99  while (1)
100 {
101     /* USER CODE END WHILE */
102
103     /* USER CODE BEGIN 3 */
104  کدهای اصلی برنامه
105 }
106 /* USER CODE END 3 */
107 }
108 }
```

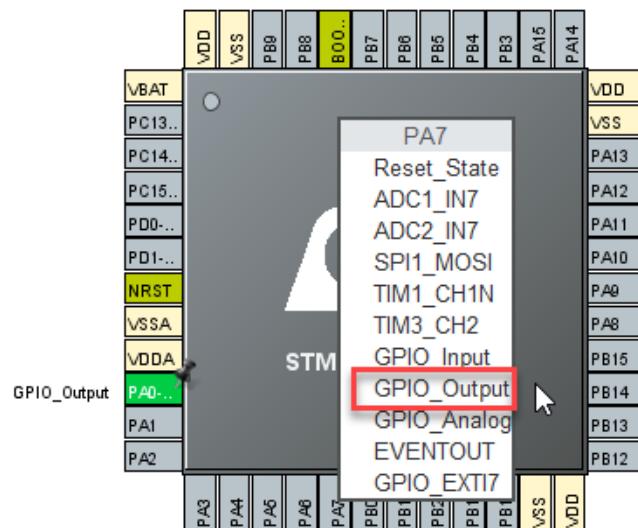
نکته : در مثال هایی که از برنامه پروتئوس استفاده شده لازم است برای stm32 Bluepill کتابخانه مربوط به این قطعه با نام BLUEPILL.lib و BLUEPILL.idx را به پروتئوس اضافه کنید. توجه داشته باشید

شبیه‌سازی‌های انجام شده برای سری STM32 در پروتئوس کامل نیست و لازم است که بعضی مثال‌ها حتماً بر روی برد واقعی اجرا شود.

مثال: هشت عدد LED به PORTA متصل کرده و یک چشمک زن بسازید.



ابتدا در برنامه Cube میکروی STM32F103C4 را انتخاب می‌کنیم. سپس روی پایه‌های PA0 تا PA7 کلیک کرده و از لیستی که وظیفه‌های آن پایه را نشان می‌دهد GPIO_Output را انتخاب می‌کیم.



در توابع HAL برای صفر و یک کردن پایه‌های خروجی از دوتابع زیر استفاده می‌شود.

HAL_GPIO_WritePin(GPIOA, GPIO_PIN_RESET); صفر کردن

HAL_GPIO_WritePin(GPIOA, GPIO_PIN_SET); یک کردن

همچنین برای ایجاد تاخیر می‌توان از تابع زیر استفاده کرد.

HAL_Delay(زمان تاخیر میلی ثانیه);

با توجه به سه تابع بالا، برنامه چشمک زن به شکل زیر نوشته می‌شود.

while (1)

{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

 HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);

 HAL_GPIO_WritePin(GPIOA,0x0f,GPIO_PIN_SET);

 HAL_Delay(300);

 HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);

 HAL_GPIO_WritePin(GPIOA,0xf0,GPIO_PIN_SET);

 HAL_Delay(300);

}

مثال: یک LED به PA.5 متصل است برنامه‌ای بنویسید که آن LED چشمک زن شود.

با استفاده از تابع HAL_GPIO_TogglePin و همچنین وجود تابع HAL_GPIO_WritePin می‌توان این

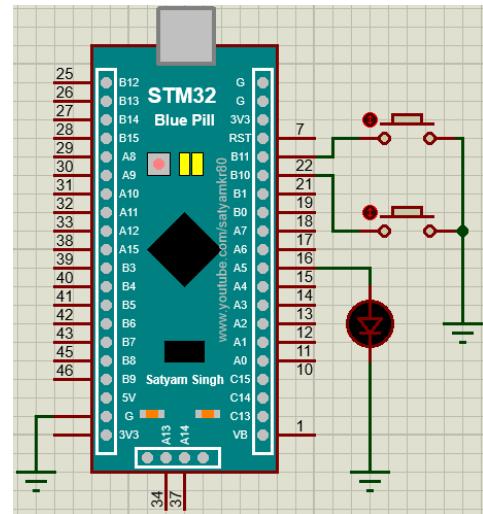
برنامه را به سه روش زیر نوشت.

```
while (1)
{
    HAL_GPIO_WritePin(GPIOA,1<<5,GPIO_PIN_RESET);
    HAL_Delay(300);
    HAL_GPIO_WritePin(GPIOA,1<<5,GPIO_PIN_SET);
    HAL_Delay(300);
}
```

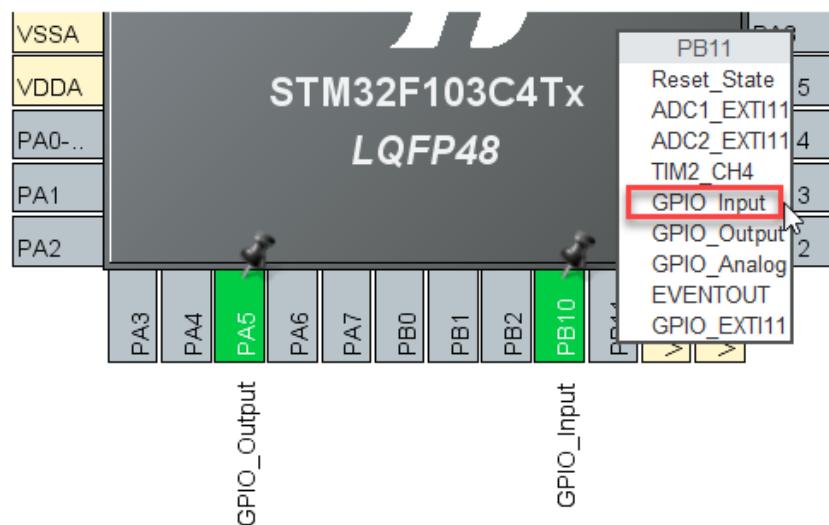
```
while (1)
{
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
    HAL_Delay(300);
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);
    HAL_Delay(300);
}
```

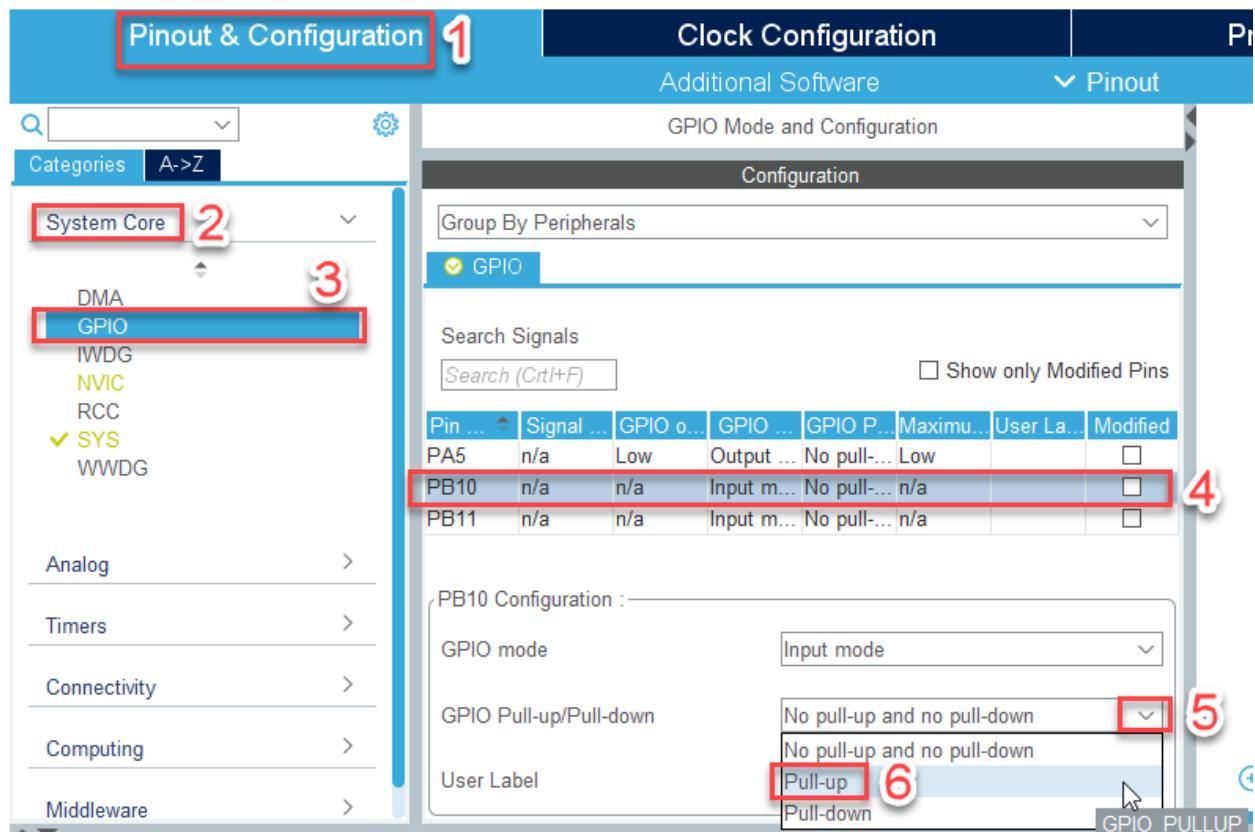
```
while (1)
{
    HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
    HAL_Delay(300);
}
```

مثال: مطابق شکل زیر یک LED به PA.5 و دو عدد کلید به PB.10 و PB.11 متصل است برنامه‌ای بنویسید که با زدن کلید 11 LED روشن و با زدن کلید 10 LED خاموش شود. در ضمن پایه‌های 10 و 11 PB از داخل میکرو PULL_UP شوند.



لازم است در برنامه Cube پایه‌های متصل به کلیدها بصورت ورودی پیکربندی شوند در ضمن از مسیری که در شکل زیر مشخص شده Pull_up داخلي این دو پایه فعال شود.

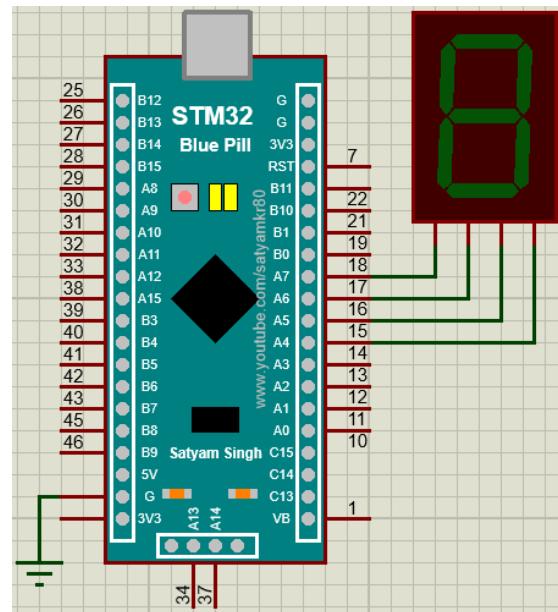




-تابع HAL برای خواندن یک پایه ورودی HAL_GPIO_ReadPin میباشد. در نتیجه برنامه این مثال به صورت زیر خواهد بود.

```
while (1)
{
    if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_11)==0
    {
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
        HAL_Delay(300);
    }
    if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_10)==0
    {
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);
        HAL_Delay(300);
    }
}
```

مثال: مطابق شکل زیر یک 7seg_BCD به PA.7 تا PA.4 متصل است برنامه‌ای بنویسید که ابتدا یک شمارنده بالا شمار ۰ تا ۱۵ و سپس یک شمارنده پایین شمار ۱۵ تا ۰ بر روی 7seg داشته باشیم.



برای اجرای این برنامه نیاز به یک متغیر داریم که آن را در محل مشخص شده قبل از main تعریف می‌کنیم.

```
/* Private variables ----- */
/* USER CODE BEGIN PV */

int i;

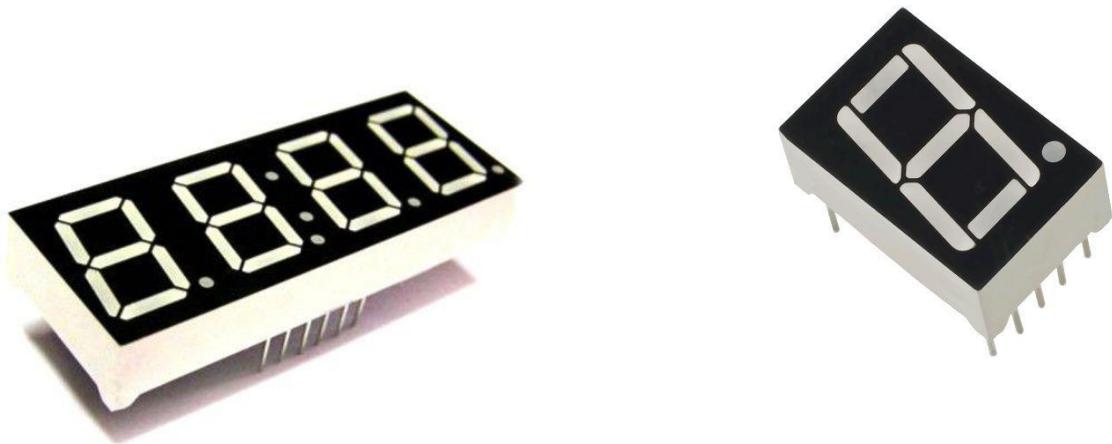
/* USER CODE END PV */

while (1)
{
    for(i=0;i<=15;i++)
    {
        HAL_GPIO_WritePin(GPIOA,0xf<<4,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GPIOA,i<<4,GPIO_PIN_SET);
        HAL_Delay(300);
    }
}
```

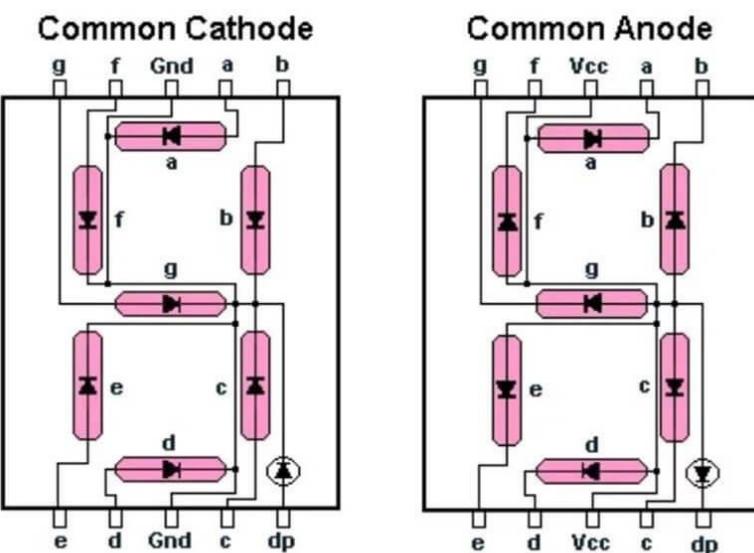
```
    }  
    for(i=15;i>=0;i--)  
    {  
        HAL_GPIO_WritePin(GPIOA,0xf<<4,GPIO_PIN_RESET);  
        HAL_GPIO_WritePin(GPIOA,i<<4,GPIO_PIN_SET);  
        HAL_Delay(300);  
    }  
}
```

نمایشگر هفت قسمتی : seven segment

این نمایشگرها در بسیاری از دستگاه‌ها برای نمایش مقادیر و بعضی از حروف و علائم مورد استفاده قرار می‌گیرند.

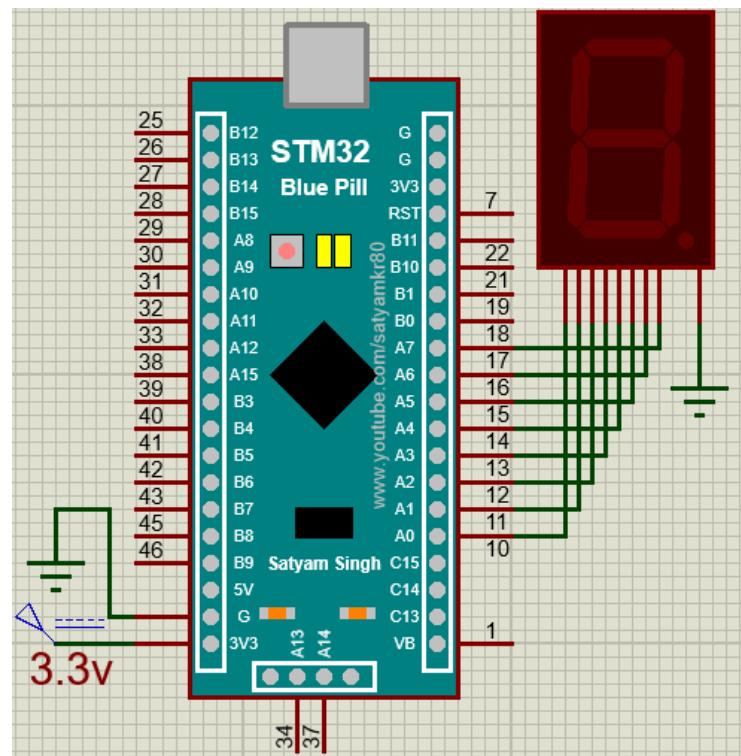


سون سگمنت‌ها در نوع کاتد مشترک common anode و آند مشترک common cathode در دسترس می‌باشند.



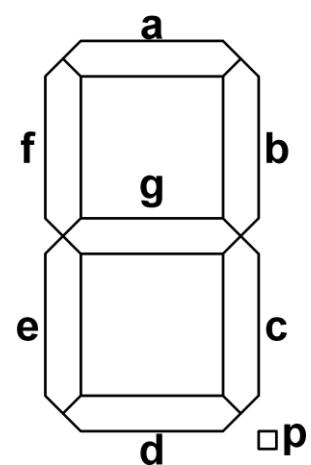
- در مدارها و مثال‌های این بخش مطالب برای سون سگمنت کاتد مشترک بیان شده است . بدیهی است برای سون سگمنت‌های آند مشترک کافی است منطق صفر و یک عوض شود.

مثال: مانند شکل زیر یک سون سگمنت کاتد مشترک را به پورت A متصل کرده و یک شمارنده ۰ تا ۹ بر روی آن می سازیم. (سون سگمنت کاتد مشترک را در پروتوپوس با نام 7SEG-MPX1_CC جستجو کنید).



ابتدا لازم است که مشخص کنیم برای روشن کردن هر عدد کدام قطعه ها باید روشن شوند. این کدها در جدول زیر نشان داده شده است.

عدد	p	g	f	e	d	c	b	a	كـ
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	1	1	1	1	0x6F



Number	0	1	2	3	4	5	6	7	8	9
Code	0x3F	0x06	0x5B	0x4F	0x66	0x6D	0x7D	0x07	0x7F	0x6F

```

/* USER CODE BEGIN PV */

int i;

unsigned char bts[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

/* USER CODE END PV */

while (1)

{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

for(i=0;i<=9;i++)

    {

        HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);

        HAL_GPIO_WritePin(GPIOA,bts[i],GPIO_PIN_SET);

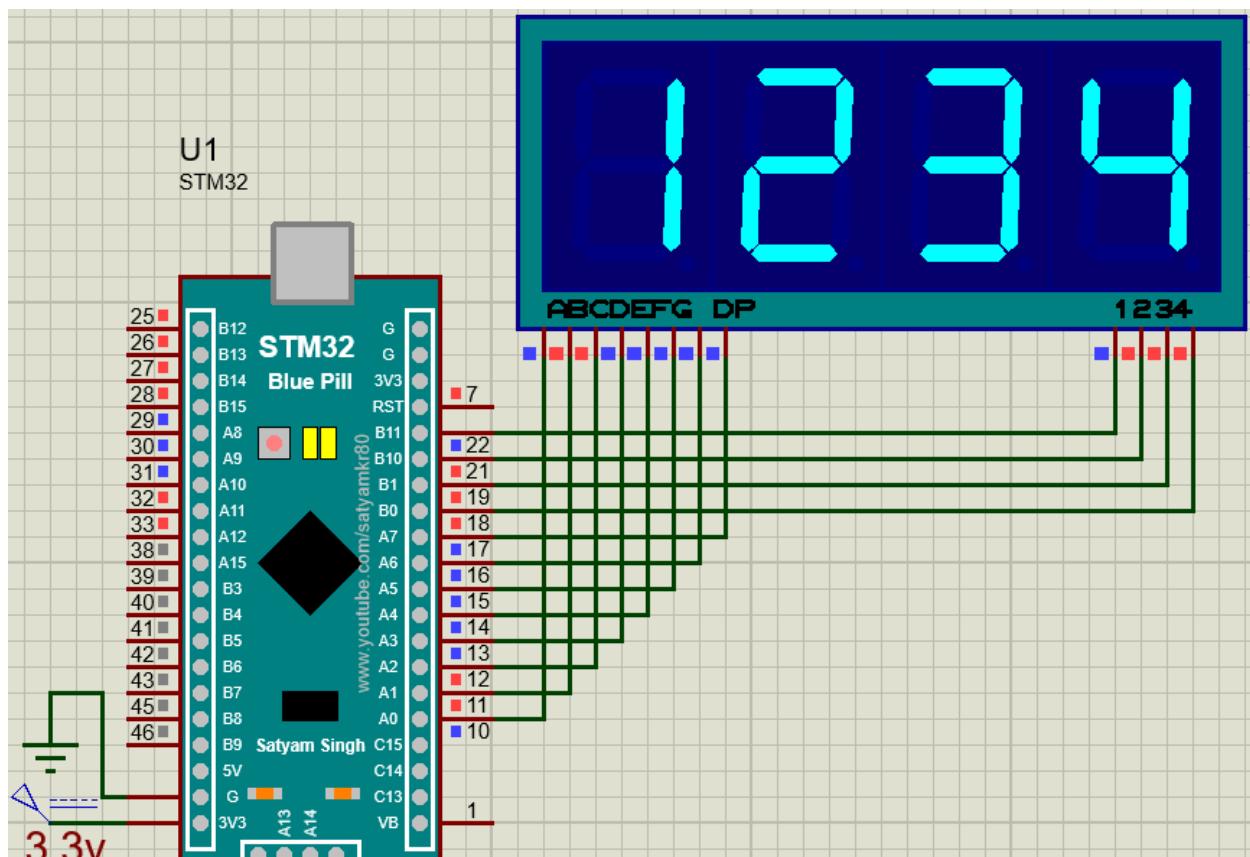
        HAL_Delay(500);

    }

}

```

نمایشگر (Display) : در اکثر مدارها برای نمایش دیتای خروجی نیاز به یک نمایشگر داریم که تعداد رقم های آن برای خروجی مدار ما مناسب باشد. فرض کنید در یک پروژه نیاز به چهار رقم در خروجی داریم؛ اگر قرار باشد که هر سون سگمنت را به یک پورت متصل کنیم، تعداد زیادی از پورت های میکرو اشغال می شود و ممکن است برای موارد دیگر با کمبود پورت مواجه شویم . راه حل این است که سون سگمنت را به شکل زیر متصل و آن ها را به روش مالتی پلکس روشن کنیم.



مثال : با توجه به شکل بالا برنامه‌ای بنویسید که روی نمایشگر عدد ثابت 1234 نمایش داده شود.

```
/* USER CODE BEGIN PV */
```

```
unsigned char bts[10]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
```

```
/* USER CODE END PV */
```

```
while (1)
```

```
{
```

```
/* USER CODE END WHILE */
```

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

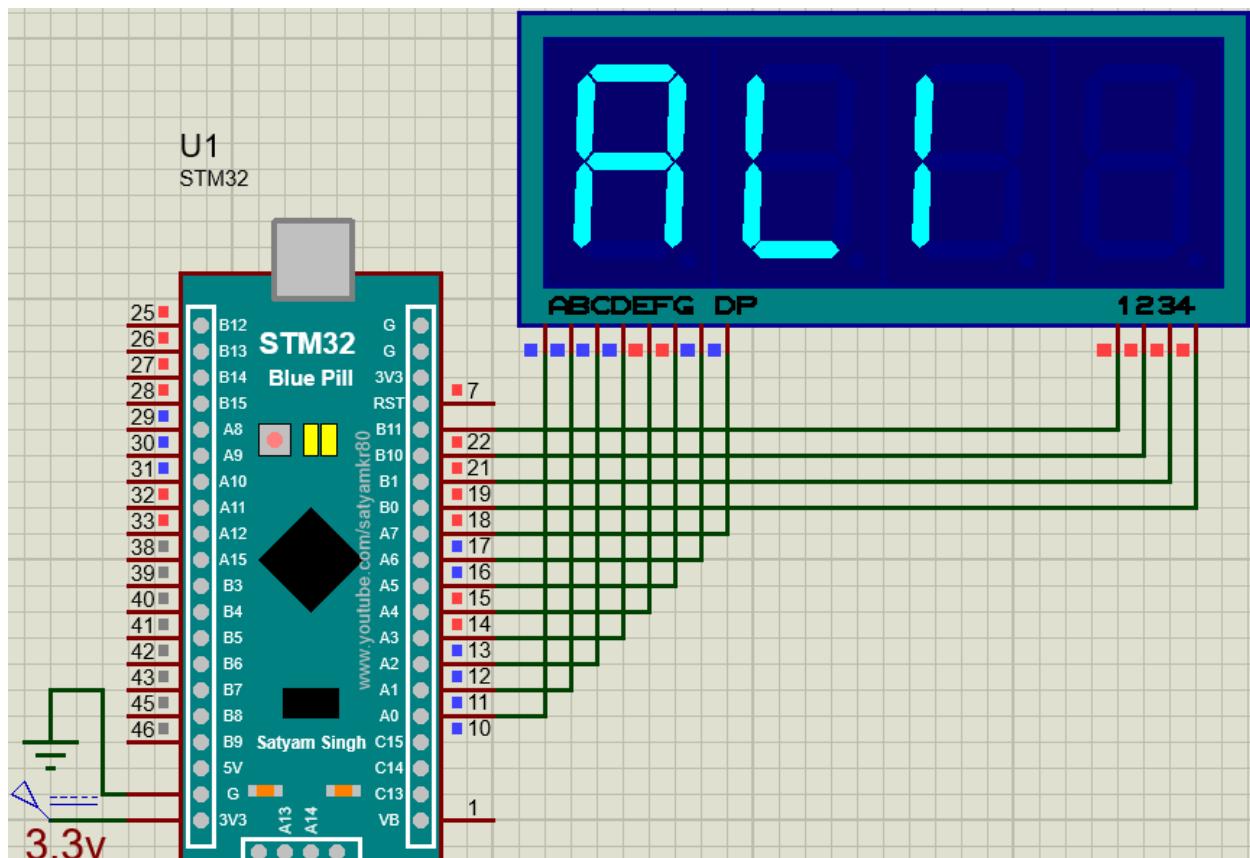
```
//-----Digit1-----
```

```
HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);
```

```
HAL_GPIO_WritePin(GPIOA,bts[1],GPIO_PIN_SET);
```

```
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_RESET);  
HAL_Delay(5);  
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_SET);  
//-----Digit2-----  
HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);  
HAL_GPIO_WritePin(GPIOA,bts[2],GPIO_PIN_SET);  
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_10,GPIO_PIN_RESET);  
HAL_Delay(5);  
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_10,GPIO_PIN_SET);  
//-----Digit3-----  
HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);  
HAL_GPIO_WritePin(GPIOA,bts[3],GPIO_PIN_SET);  
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_1,GPIO_PIN_RESET);  
HAL_Delay(5);  
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_1,GPIO_PIN_SET);  
//-----Digit4-----  
HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);  
HAL_GPIO_WritePin(GPIOA,bts[4],GPIO_PIN_SET);  
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0,GPIO_PIN_RESET);  
HAL_Delay(5);  
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0,GPIO_PIN_SET);  
}  
/* USER CODE END 3 */
```

مثال: کلمه ALI را بر روی سون سگمنت ۴ تایی نمایش دهید.



```
while (1)
```

```
{
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

//-----Digit1-----

    HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);

    HAL_GPIO_WritePin(GPIOA,0x77,GPIO_PIN_SET);

    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_RESET);

    HAL_Delay(5);

    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_SET);
```

//-----Digit2-----

```
    HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);  
    HAL_GPIO_WritePin(GPIOA,0x38,GPIO_PIN_SET);  
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_10,GPIO_PIN_RESET);  
    HAL_Delay(5);  
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_10,GPIO_PIN_SET);
```

//-----Digit3-----

```
    HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);  
    HAL_GPIO_WritePin(GPIOA,0x30,GPIO_PIN_SET);  
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_1,GPIO_PIN_RESET);  
    HAL_Delay(5);  
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_1,GPIO_PIN_SET);
```

//-----Digit4-----

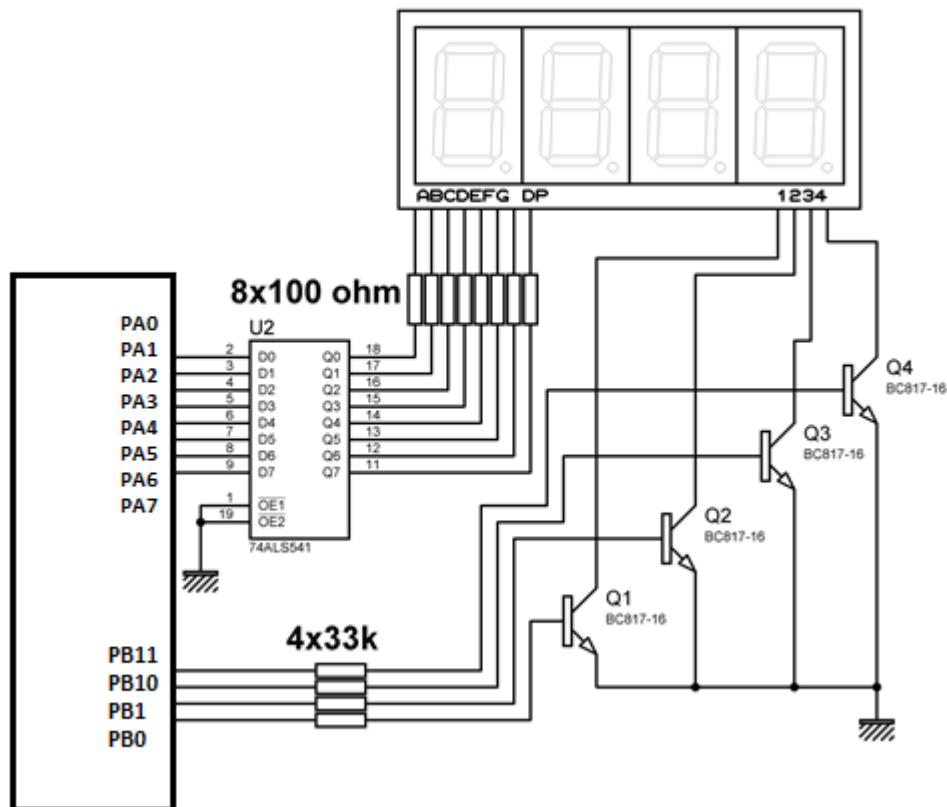
```
    HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);  
    HAL_GPIO_WritePin(GPIOA,0x00,GPIO_PIN_SET);  
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0,GPIO_PIN_RESET);  
    HAL_Delay(5);  
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0,GPIO_PIN_SET);
```

}

/* USER CODE END 3 */

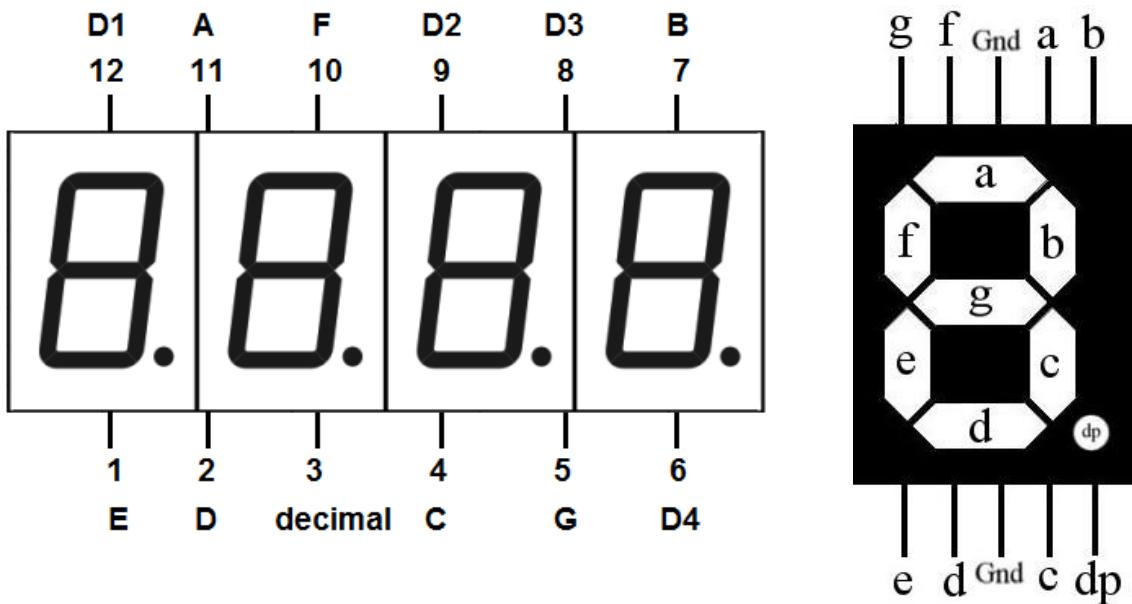
درایور نمایشگر :

با توجه به این که در مدار نمایشگر بالا ، کاتد هر رقم به یکی از بیت های PORTB متصل شده ، و جریان عبوری از کاتد در حالتی که تمام سگمنت ها و نقطه اعشار روشن باشد ، در حدود 80mA خواهد شد و با توجه به این که جریان قابل تحمل هر بیت از پورت حداکثر 5mA است پس در عمل نمی توان کاتد را مستقیم به پورت متصل کرد، و نیاز به یک درایور داریم مدار زیر که در بردا آزمایش نیز استفاده شده است می تواند یک نمونه درایور برای نمایشگرهای سون سگمنتی باشد.



لازم به ذکر است که مقدار مقاومت های 33k را می توان با توجه به نور سون سگمنت ها با مقادیر مختلف جایگزین کرد تا نور مناسب حاصل شود.

* شماره پایه ها و نحوه قرارگیری آن ها در قطعات واقعی به شکل زیر است :



تمرین: برنامه ای بنویسید که یک شمارنده ۰ تا ۹۹۹ بر روی نمایشگر داشته باشیم.

تمرین: برنامه ای بنویسید که به طور هم زمان دو شمارنده بالا شمار ۰ تا ۹۹ و شمارنده پایین شمار

۹۹ تا ۰ بر روی نمایشگر داشته باشیم .

تمرین: برنامه ای بنویسید که بر روی یک نمایشگر ۶ رقمی یک ساعت داشته باشیم . در ضمن اعداد

ساعت ، دقیقه و ثانیه را با روشن کردن نقطه اعشار یکان دقیقه و ساعت از هم جدا کنید.

تمرین: برنامه ای بنویسید که کلمه ALI روی نمایشگر حرکت کند(تابلوی روان) .

تمرین: در نرم افزار پروتئوس یک نمایشگر هشت رقمی را به میکرو متصل کنید و کلمه ALI را حرکت

دهیید . برنامه را طوری بنویسید که حرکت بصورت رفت و برگشت باشد.

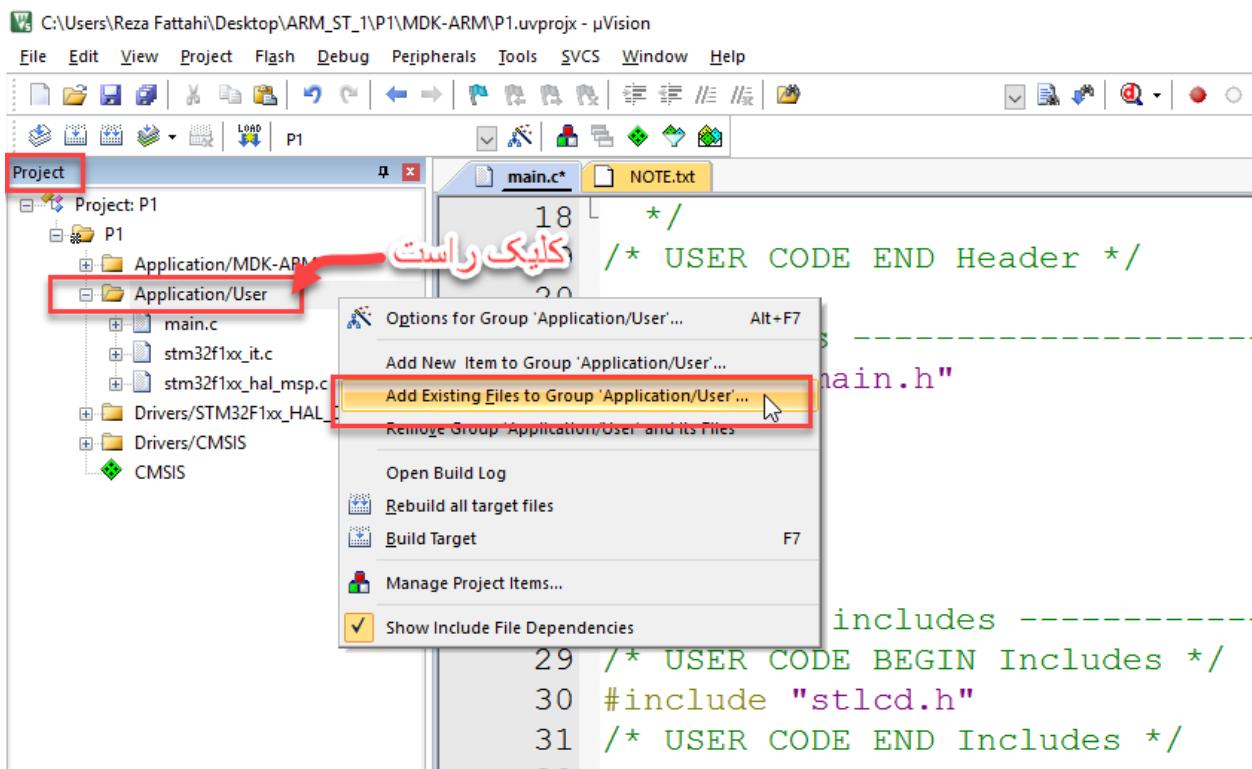
اضافه کردن هدر به برنامه:

فرض کنید برای راهاندازی LCD دو فایل با نام‌های `stlcd.c` و `stlcd.h` وجود داشته باشد ، در تابع `h`. معرفی توابع و در فایل `C`. بدنه توابع نوشته شده. برای اضافه کردن این دو فایل به برنامه پس از تولید پروژه فایل `h`. را در پوشه `Inc` و فایل `C`. را در پوشه `Src` کپی می‌کنیم. در برنامه ، فایل `h`. را با استفاده از پیش پردازنده زیر اضافه می‌کنیم.

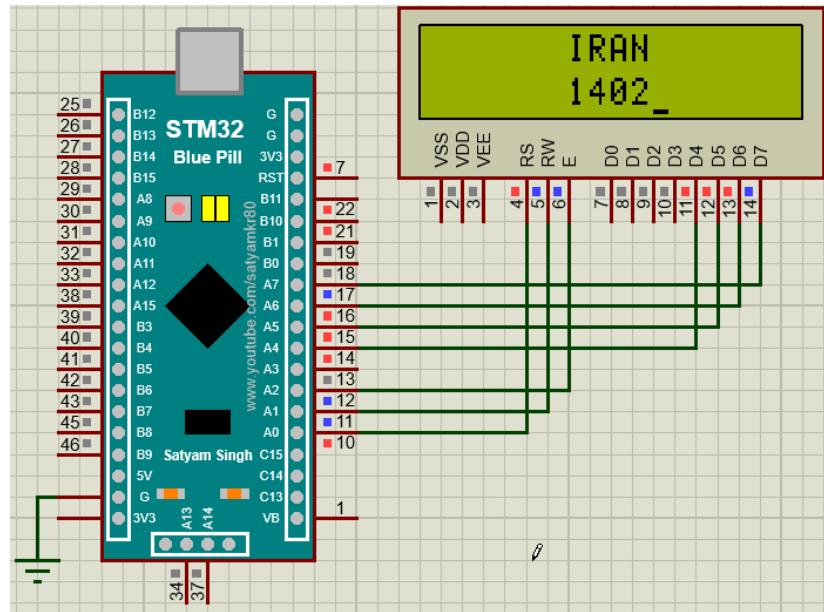
```
#include "stlcd.h"
```

نکته: در نسخه جدید `cube` دو پوشه `Inc` و `Src` در داخل پوشه `Core` قرار دارند.

و فایل `C`. را باید طبق شکل زیر به برنامه اضافه کنیم. در پنجره `Project` بر روی `Project/User` کلیک راست کرده ، سپس در منوی باز شده گزینه `Add Existing File to Group` را انتخاب می‌کنیم و با توجه به محلی که پروژه و فایل `C`. را ذخیره کرده‌ایم فایل به پروژه اضافه می‌شود.

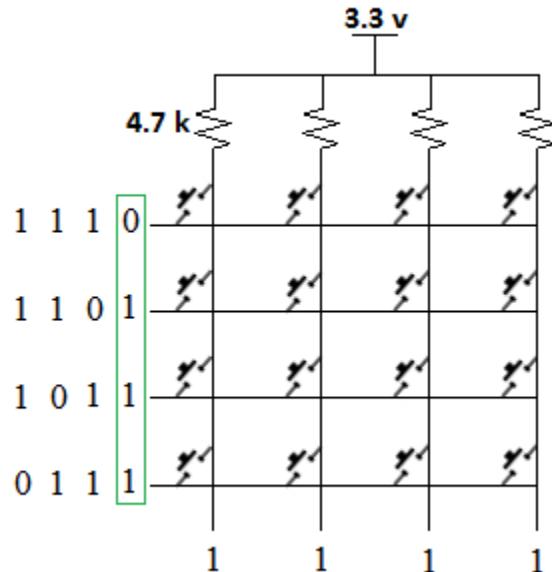


مثال : با توجه به شکل زیر برنامه‌ای بنویسید که روی خط اول lcd کلمه IRAN و روی خط دوم 1402 نوشته شود.



```
/* Private includes ----- */  
/* USER CODE BEGIN Includes */  
  
#include "stlcd.h"  
  
/* USER CODE END Includes */  
  
/* USER CODE BEGIN 2 */  
  
lcd_init();  
  
lcd_gotoxy(6,0);  
  
lcd_puts("IRAN");  
  
lcd_gotoxy(6,1);  
  
lcd_puts("1402");  
  
/* USER CODE END 2 */
```

صفحه کلید: یکی دیگر از وسایل ورودی صفحه کلید می باشد که در آن کلیدها به صورت ماتریسی چیده شده اند.

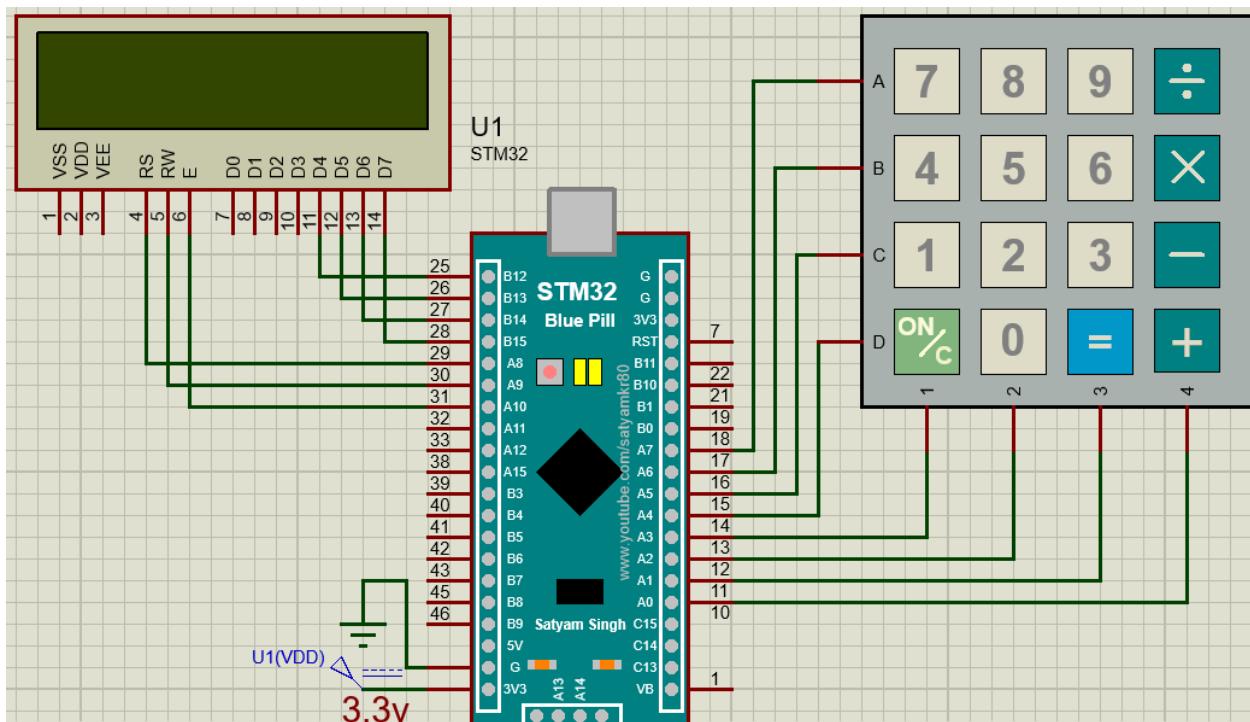


برای خواندن صفحه کلید به شکل زیر عمل شود:

ابتدا یک سطر را صفر و بقیه را یک می کنیم، سپس ستون ها را می خوانیم. اگر همه ی آن ها یک باشند، یعنی در آن سطر کلیدی زده نشده است. آن سطر را یک و سطر بعدی را صفر می کنیم و مجددا ستون هارا می خوانیم. این کار را برای همه ی سطراها انجام می دهیم. اگر سطروی را صفر کردیم و ستونی صفر شد با توجه به سطر و ستونی که صفر شده، کلید فعل مشخص می شود.

مثال : با توجه به مدار زیر برنامه‌ای بنویسید که با زدن هر کلید عدد متناظر با آن و برای کلیدهای غیر عددی از ۱۰ تا ۱۵ بر روی LCD نمایش داده شود .

÷	×	-	+	=	ON/C
10	11	12	13	14	15



در تنظیمات cube پایه‌های PA0 تا PA3 از میکرو که به سطرهای صفحه کلید متصل هستند را خروجی و پایه‌های مربوط به ستون‌ها یعنی PA3 تا PA0 را ورودی و از مسیر نشان داده شده در شکل زیر را، از داخل میکرو PULL_UP می‌کنیم.

Pin ...	Signal o.	GPIO ou...	GPIO m...	GPIO P...	Maximu...	User Label	Modified
PA0-W...	n/a	n/a	Input m...	Pull-up	n/a		<input checked="" type="checkbox"/>
PA1	n/a	n/a	Input m...	Pull-up	n/a		<input checked="" type="checkbox"/>
PA2	n/a	n/a	Input m...	Pull-up	n/a		<input checked="" type="checkbox"/>
PA3	n/a	n/a	Input m...	No pull...	n/a		<input type="checkbox"/>
PA4	n/a	Low	Output ...	No pull...	Low		<input type="checkbox"/>
PA5	n/a	Low	Output ...	No pull...	Low		<input type="checkbox"/>
PA6	n/a	Low	Output ...	No pull...	Low		<input type="checkbox"/>
PA7	n/a	Low	Output ...	No pull...	Low		<input type="checkbox"/>
PA8	n/a	Low	Output ...	No pull...	Low		<input type="checkbox"/>

- برای خوانایی و سهولت در برنامه نویسی با استفاده از پیش پردازنده `#define` برای صفر و یک شدن سطرها و همچنین خواندن ستون‌ها، نام‌های جدید تعریف کردہ‌ایم.

```

/* Private define -----*/
/* USER CODE BEGIN PD */

#define R1_0 HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_RESET)
#define R1_1 HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_SET)
#define R2_0 HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_RESET)
#define R2_1 HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_SET)
#define R3_0 HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET)
#define R3_1 HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET)
#define R4_0 HAL_GPIO_WritePin(GPIOA,GPIO_PIN_4,GPIO_PIN_RESET)
#define R4_1 HAL_GPIO_WritePin(GPIOA,GPIO_PIN_4,GPIO_PIN_SET)
#define C1 HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_3)
#define C2 HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_2)
#define C3 HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_1)
#define C4 HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0)

/* USER CODE END PD */

```

برای خواندن صفحه کلید یک تابع به نام `kb` تعریف شده است. هر گاه این تابع فراخوانی شود تمام کلیدها را جاروب کرده و اگر کلیدی فشرده شده باشد عدد مربوط به آن کلید و در غیر این صورت عدد ۱۶ را برمی‌گرداند.

```

/* Private variables -----*/

```

```

/* USER CODE BEGIN PV */

```

```

unsigned char k;

```

```
char t[16];

/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);

static void MX_GPIO_Init(void);

/* USER CODE BEGIN PFP */

unsigned char kb(void)

{

    unsigned char k=16;

    //-----ROW1-----

        R1_0;

        HAL_Delay(3);

        if(C1==0) {k=7;while(C1==0);}

        if(C2==0) {k=8;while(C2==0);}

        if(C3==0) {k=9;while(C3==0);}

        if(C4==0) {k=10;while(C4==0);}

        R1_1;

    //-----ROW2-----

        R2_0;

        HAL_Delay(3);

        if(C1==0) {k=4;while(C1==0);}

        if(C2==0) {k=5;while(C2==0);}

        if(C3==0) {k=6;while(C3==0);}
```

```
if(C4==0) {k=11;while(C4==0);}

R2_1;

//-----ROW3-----

R3_0;

HAL_Delay(3);

if(C1==0) {k=1;while(C1==0);}

if(C2==0) {k=2;while(C2==0);}

if(C3==0) {k=3;while(C3==0);}

if(C4==0) {k=12;while(C4==0);}

R3_1;

//-----ROW4-----

R4_0;

HAL_Delay(3);

if(C1==0) {k=15;while(C1==0);}

if(C2==0) {k=0;while(C2==0);}

if(C3==0) {k=14;while(C3==0);}

if(C4==0) {k=13;while(C4==0);}

R4_1;

return k;

}

/* USER CODE END PFP */

while (1)

{
```

```

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

k=kb();

if(k!=16)

{

    sprintf(t,"%d",k);

    lcd_puts(t);

}

}

/* USER CODE END 3 */

```

مثال : برنامه بالا را طوری تغیر دهید که با زدن کلیدهای غیر عددی علامت مربوط به آنها روی LCD شود همچنین با زدن کلید ON/C صفحه LCD پاک شود.

```

while (1)

{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

k=kb();

if(k<10)

{

    sprintf(t,"%d",k);

    lcd_puts(t);

}

if(k==10) lcd_putchar('/');

```

```

if(k==11) lcd_putchar('*');

if(k==12) lcd_putchar('-');

if(k==13) lcd_putchar('+');

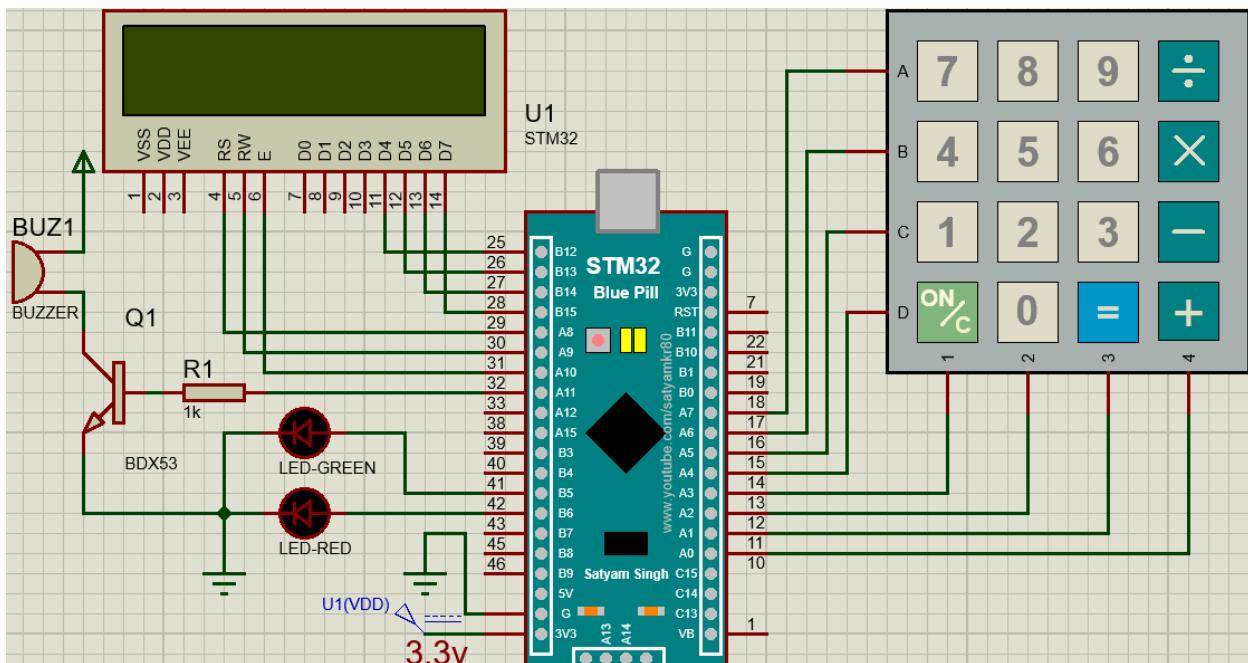
if(k==14) lcd_putchar('=');

if(k==15) lcd_clear();

}

```

با توجه به شکل زیر تمرین ها را انجام دهید.



تمرین: برنامه صفحه کلید را طوری تکمیل کنید که با زدن هر کلید علاوه بر نمایش عدد و علامت ، نیز بوق کوتاهی بزند.

ب) چند رقمی

الف) یک رقمی

.

تمرین: یک ماشین حساب بسازید. که کاربر یک رمز چهار رقمی را وارد کند اگر رمز صحیح بود led سبز روشن شود و buzzer یک بوق بزند، و اگر غلط بود led قرمز روشن شود و buzzer سه بار بوق بزند .

تمرين: تمرين قبل را طوري كامل کنيد که مدیر بتواند رمز دستگاه را نيز عوض کند . در ضمن هنگامی که کاربران رمز را وارد می کنند اعداد زده شده بصورت * روی lcd دیده شوند. همچنین اگر کاربر رمز ورودی را سه بار استبهار وارد کرد ، کیبورد قفل ، و هر دو led روشن شوند.

مبدل آنالوگ به ديجيتال : ADC



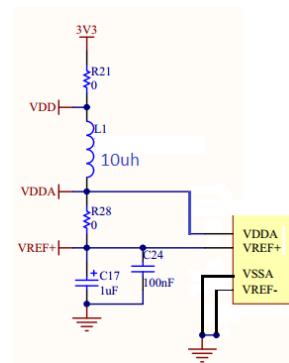
تمام کميتهای اطراف ما مانند دما، فشار، رطوبت، شدت نور و... مقادیری آنالوگ هستند، برای اندازه‌گیری و پردازش روی اين کميتهای لازم است که ابتدا آنها را به يك مقدار ديجيتال تبديل و سپس مورد پردازش قرار دهيم. مدار هايي که اين تبديل را انجام می دهند(مبدل آنالوگ به ديجيتال) ADC نام دارند.

در ميكروهاي STM بسته به نوع ميكرو يك يا دو واحد ADC با تعداد بيت خروجي 12 يا 16 بيت و حداکثرتا 16 کanal ورودی وجود دارد . دو کanal از کanalهای ورودی ، از داخل ميكرو يکی به سنسور دما و دیگری به يك ولتاژ مرجع متصل می باشد که می توان در موارد خاص از آنها استفاده کرد.

تغذيه و ولتاژ مرجع : ADC

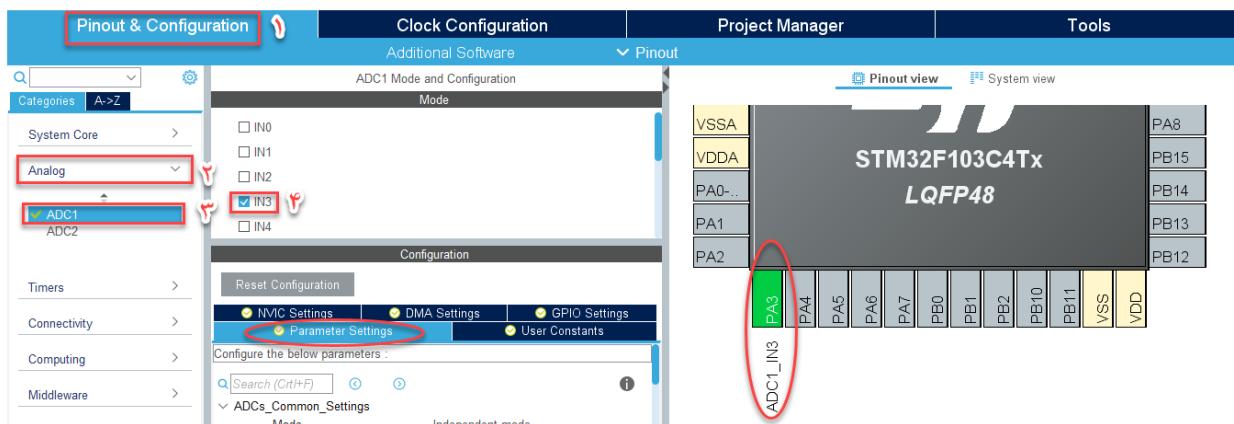
برای بهتر کردن عملکرد مبدل و همچنین مصون ماندن این قسمت ، از نويز تولیدي ميكرو ، معمولاً تغذيه بخش ADC از بقیه ميكرو جدا شده تا کاربر بتواند با متصل کردن يك تغذيه مناسب به پایه های VDDA و VSSA دقت بالاتری در خروجی ADC داشته باشد. البته کاربر می تواند بجای منبع تغذيه جدآگانه برای کل مدار از يك منبع استفاده کرده و روی پایه VDDA از يك فیلتر مانند شکل زير استفاده کند ، می توان مقاومت R28 را حذف و ولتاژ مرجع خارجي را به پایه VREF+ متصل نمود.

توجه شود که اگر ميكروي انتخاب شده فاقد پایه های VREF+ و VREF- و VDDA و VSSA همان پایه های ولتاژ مرجع می باشند.

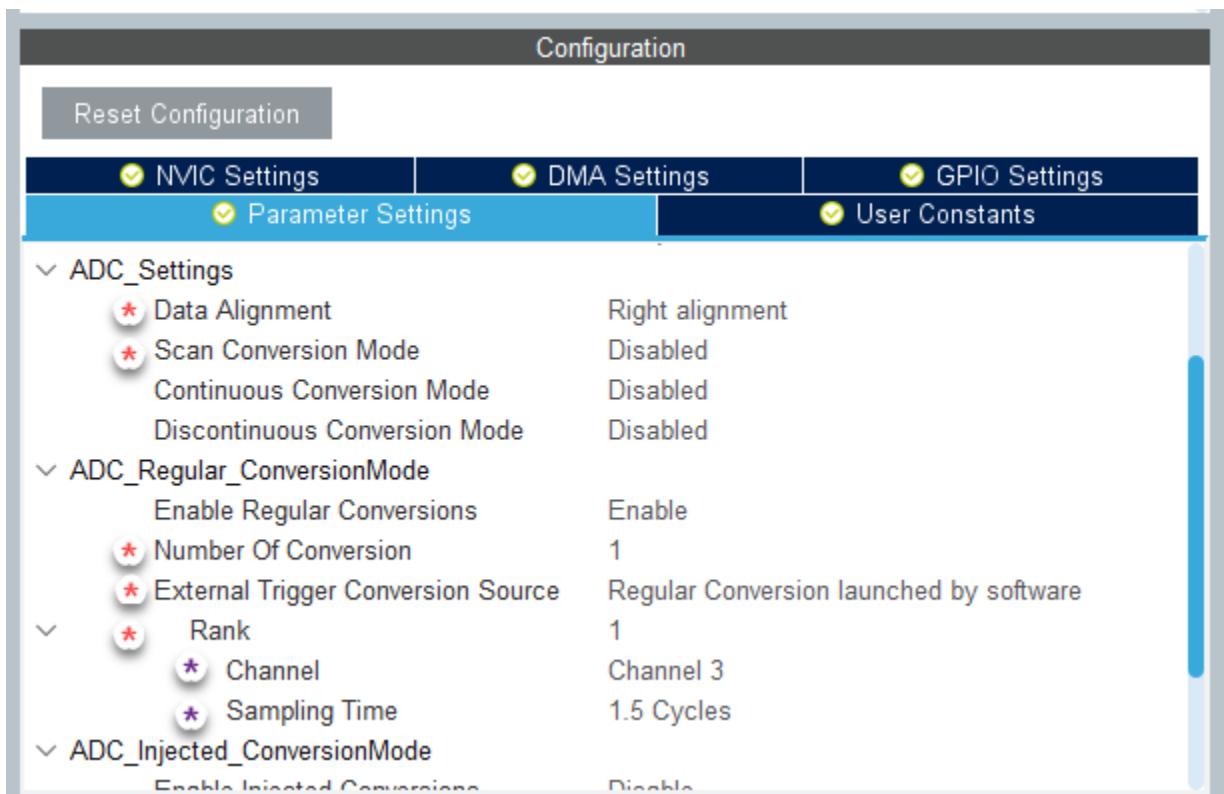


راه اندازی واحد ADC :

در نرم افزار cube و در صفحه Pinout از مسیر نشان داده شده در شکل زیر یکی از کانال های ADC را فعال و سپس در بخش parameter setting سایر مشخصات مدنظر را تعیین می کنیم.

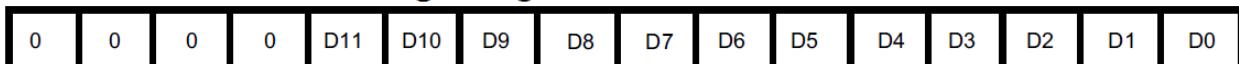


: پارامترهای بخش Parameter Setting



: در این قسمت می‌توانید مشخص کنید که دیتای خروجی از مبدل بصورت راست چین Right alignment یا چپ چین Left alignment در اختیار قرار گیرد . با توجه به این که دیتای خروجی در یک رجیستر 16 بیتی ذخیره می‌شود در دو حالت راست و چپ چین عدد خروجی ADC به صورت زیر خواهد بود .

Right alignment of 12-bit data



Left alignment of 12-bit data



: اگر تعداد کانال‌های ورودی به ADC فقط یک کانال باشد این قسمت غیر فعال Disable است. اگر بیش از یک کانال را فعال کنیم به حالت فعال Enable می‌شود.

: در این قسمت مشخص می‌کنیم چه تعداد از کانال‌های فعال برای تبدیل استفاده می‌شود.

: در این قسمت مشخص می‌کنیم که فرمان شروع تبدیل به صورت نرمافزاری باشد یا از طریق سخت‌افزار برای مثال رخدادی در یکی از تایمروها.

: به ازای هر کانال فعال یک Rank ایجاد می‌شود که می‌توان در داخل آن با تنظیم تعداد سیکل برای هر تبدیل ، تعداد نمونه برداری در ثانیه را با استفاده از روابط زیر مشخص نمود.

$$T_{conv} = \text{Sampling time} + 12.5 \text{ cycles}$$

با فرض این‌که در Rank مربوط به کانال بر روی Sampling Time=1.5 cycles قرار گرفته باشد تعداد سیکل‌های لازم برای تولید یک نمونه برابر است با: $T_{conv}=1.5+12.5=14 \text{ cycles}$

اگر فرکانس واحد ADC روی 14MHz تنظیم شده باشد ، زمان لازم برای تبدیل ورودی آنالوگ به دیجیتال برابر است با: $T_{conv}=14/14\text{Mhz}=1 \text{ us}$ در نتیجه این مبدل می‌تواند در یک ثانیه یک میلیون نمونه Msps: **Megasamples per second** (millions of samples per second)=1Msps برداری را انجام دهد.

نکته: تعداد پالس‌های لازم برای تولید یک نمونه و همچنین حداقل و حداکثر فرکانس واحد ADC باید از دیتا شیت آن میکرو استخراج گردد . مثال بالا برای میکروهای سری STM32F10xxx صدق می‌کند.

ضریب تفکیک: resolution

پارامتری که مشخص می‌کند حساسیت ADC چقدر است . یعنی به ازای چه مقدار تغییر در ولتاژ ورودی عدد خروجی یک واحد تغییر می‌کند و از رابطه زیر قابل محاسبه است. در این رابطه Vref ولتاژ مرجع و n تعداد بیت خروجی مبدل می‌باشد .

$$\text{ضریب تفکیک} = V_{ref}/(2^n - 1)$$

عدد خروجی مبدل را می‌توان از رابطه زیر بدست آورد.

$$\text{ضریب تفکیک} = \text{عدد خروجی}/V_{in}$$

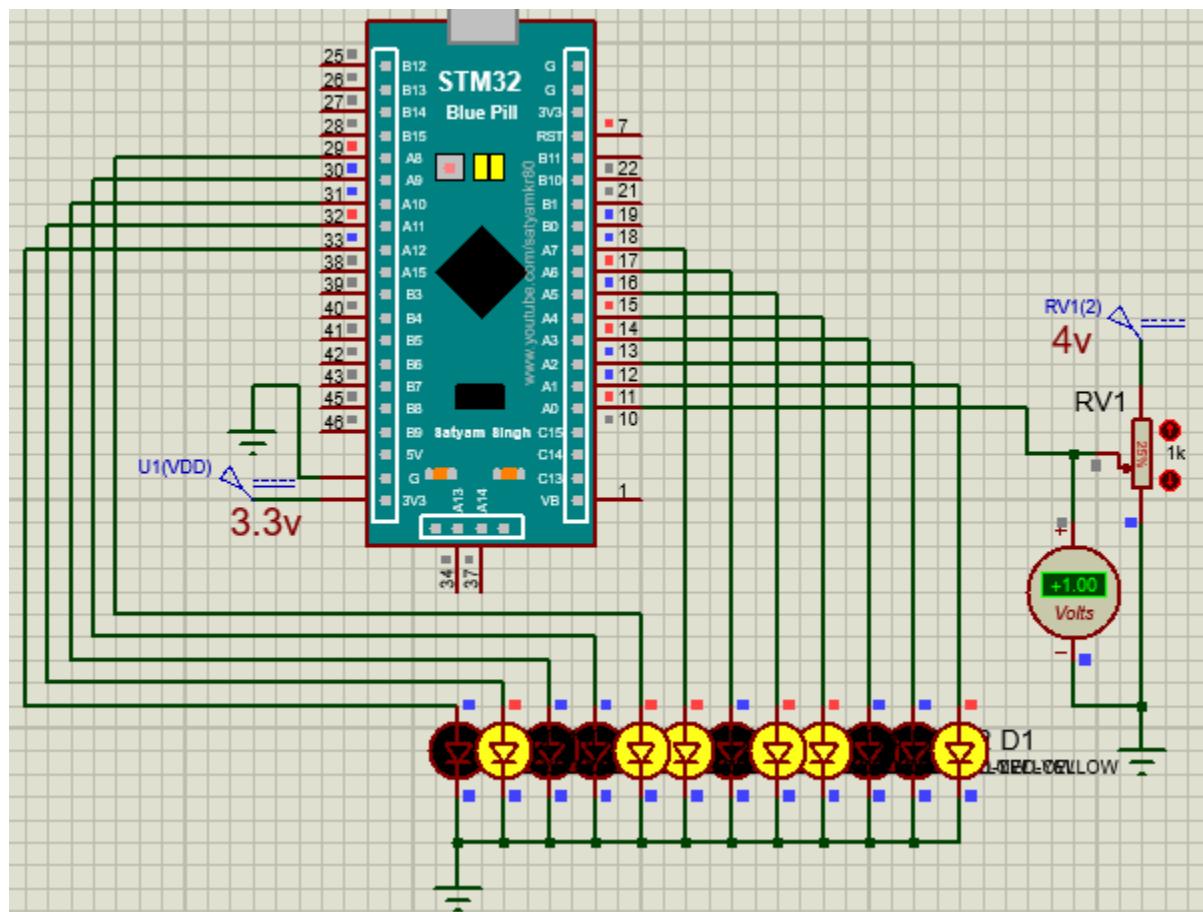
مثال: اگر ولتاژ مرجع 3.3v و تعداد بیت خروجی مبدل 12 بیت و ولتاژ ورودی 1v باشد مطلوب است

۱) ضریب تفکیک؟ ۲) عدد خروجی؟

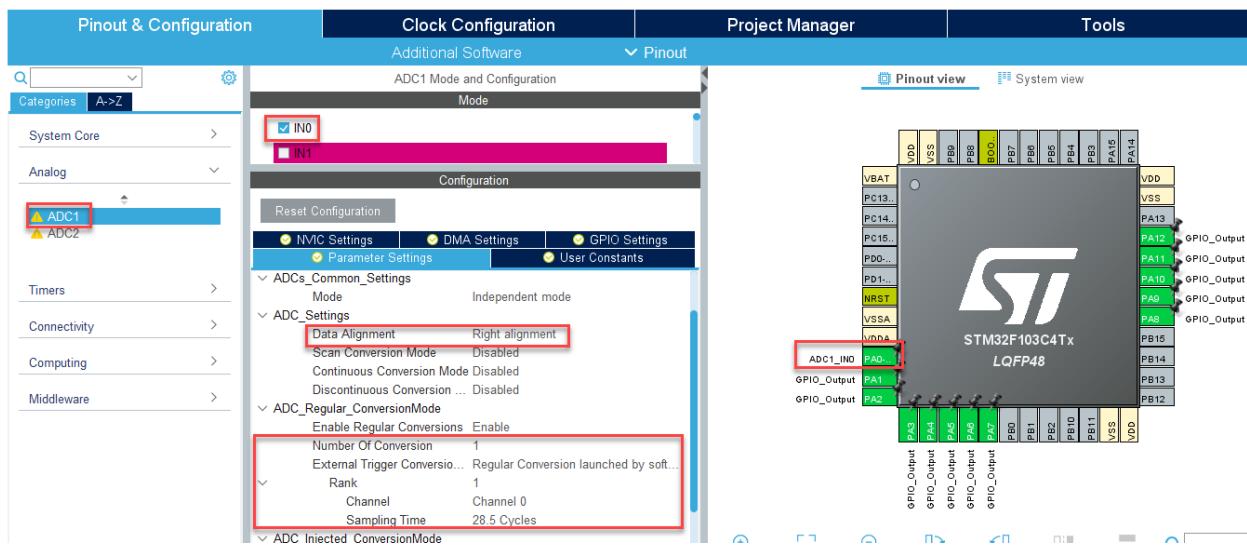
$$\text{ضریب تفکیک} = 3.3/(2^{12}-1) = 3300\text{mv}/(4095) = 0.8058\text{mv}$$

$$\text{عدد خروجي} = 1000 \text{mv} / 0.8058 \text{mv} = 1241$$

مثال: مانند شکل زیر ۱۲ عدد LED به پایه های PA1 تا PA12 و یک پتانسیومتر به کanal صفر ADC میکرو متصل است. در برنامه cube تنظیمات لازم را انجام داده و سپس در keil برنامه‌ای بنویسید که عدد خروجی مبدل بر روی LED ها بصورت باپنری دیده شود.



تنظیمات Cube به شکل زیر انجام می‌شود.



```
/* USER CODE BEGIN 2 */
```

```
hadc1.Init.NbrOfConversion=1;
```

```
HAL_ADC_Init(&hadc1);
```

```
/* USER CODE END 2 */
```

```
/* Infinite loop */
```

```
/* USER CODE BEGIN WHILE */
```

```
while (1)
```

```
{
```

```
/* USER CODE END WHILE */
```

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

```
HAL_ADC_Start(&hadc1);
```

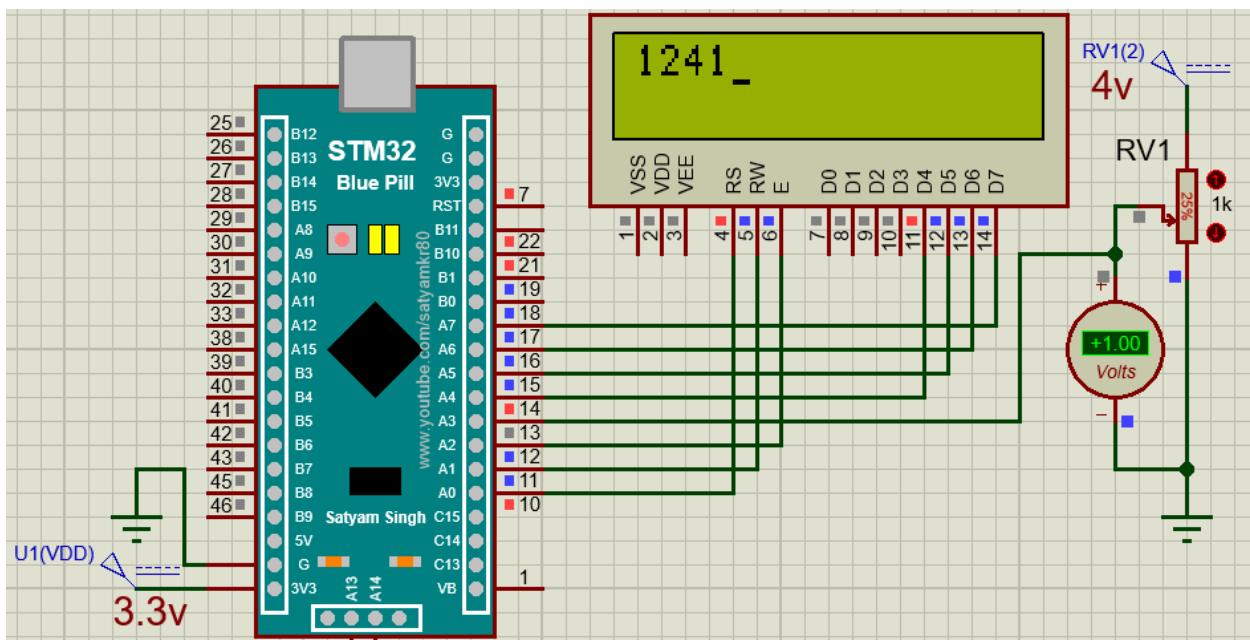
```
if(HAL_ADC_PollForConversion(&hadc1,100)==HAL_OK)
```

```
{
```

```
a=HAL_ADC_GetValue(&hadc1);
```

```
    HAL_GPIO_WritePin(GPIOA,0xffff<<1,GPIO_PIN_RESET);  
    HAL_GPIO_WritePin(GPIOA,a<<1,GPIO_PIN_SET);  
}  
  
HAL_ADC_Stop(&hadc1);  
  
HAL_Delay(300);  
  
}  
  
/* USER CODE END 3 */
```

مثال: مانند شکل زیر یک LCD و یک پتانسیومتر به میکرو متصل کنید.(پتانسیومتر به کanal ۳ متصل است). در برنامه cube تنظیمات لازم را انجام داده و سپس در keil برنامه‌ای بنویسید که عدد خروجی مبدل بر روی LCD دیده شود.



تنظیمات Cube به شکل زیر انجام می‌شود.

The screenshot shows the STM32CubeMX software interface with the following tabs:

- Pinout & Configuration**: Shows the STM32F103C4Tx LQFP48 pinout with the ADC1 pin highlighted in red.
- Clock Configuration**: Shows the clock settings for ADC1.
- Project Manager**: Shows the project structure.
- Tools**: Shows various development tools.

The **ADC1 Mode and Configuration** section is expanded, showing the following configuration:

- Mode**: IN2, IN3 (selected)
- Configuration**:
 - Reset Configuration**: NVIC Settings, DMA Settings, Parameter Settings, User Constants
 - ADC_Settings**:
 - Data Alignment: Right alignment
 - Scan Conversion Mode: Disabled
 - Continuous Conversion Mode: Disabled
 - Discontinuous Conversion: Disabled
 - ADC_Regular_ConversionMode**:
 - Enable Regular Conversions: Enable
 - Number Of Conversion: 1
 - External Trigger Conversion...: Regular Conversion launched by soft.
 - Rank: 1
 - Channel: Channel 3
 - Sampling Time: 28.5 Cycles

```
/* USER CODE BEGIN 2 */

lcd_init();

hadc1.Init.NbrOfConversion=1;

HAL_ADC_Init(&hadc1);

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)

{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

    HAL_ADC_Start(&hadc1);

    if(HAL_ADC_PollForConversion(&hadc1,100)==HAL_OK)

    {

        a=HAL_ADC_GetValue(&hadc1);

        sprintf(t,"%04d",a);

        lcd_gotoxy(0,0);

        lcd_puts(t);

    }

    HAL_ADC_Stop(&hadc1);

    HAL_Delay(100);

}

/* USER CODE END 3 */
```

مثال : با اضافه کردن کدهای مناسب ، مثال قبل را یک ولتمتر تبدیل کنید.

پاسخ: با توجه به این که عدد خروجی مبدل ، به ازای ولتاژ ورودی $1V$ برابر با 1241 می باشد کافی است عدد خروجی مبدل را به 1241 تقسیم کنیم. در نتیجه کدهای زیر را اضافه می کنیم.

```
/* USER CODE BEGIN PV */  
unsigned int a;  
char t[16];  
float v;  
/* USER CODE END PV */  
/* USER CODE BEGIN 2 */  
lcd_init();  
hadc1.Init.NbrOfConversion=1;  
HAL_ADC_Init(&hadc1);  
/* USER CODE END 2 */  
/* Infinite loop */  
/* USER CODE BEGIN WHILE */  
while (1)  
{  
/* USER CODE END WHILE */  
/* USER CODE BEGIN 3 */  
HAL_ADC_Start(&hadc1);  
if(HAL_ADC_PollForConversion(&hadc1,100)==HAL_OK)  
{  
a=HAL_ADC_GetValue(&hadc1);
```

```

v=a/1241;

lcd_putchar(65);

sprintf(t,"V=%4.2f",v);

lcd_gotoxy(0,0);

lcd_puts(t);

}

HAL_ADC_Stop(&hadc1);

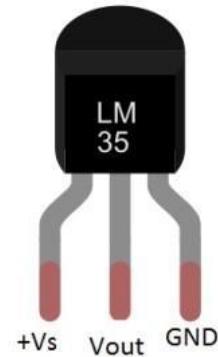
HAL_Delay(100);

}

/* USER CODE END 3 */

```

سنسور دما **LM35** : این سنسور آنالوگ ، با حساسیت $10\text{mV}/\text{C}^{\circ}$ را می‌توان به طور مستقیم به یکی از کانال‌های ADC متصل نمود .



مثال: سنسور LM35 و یک LCD را به میکرو متصل کرده و یک دماسنج بسازید.

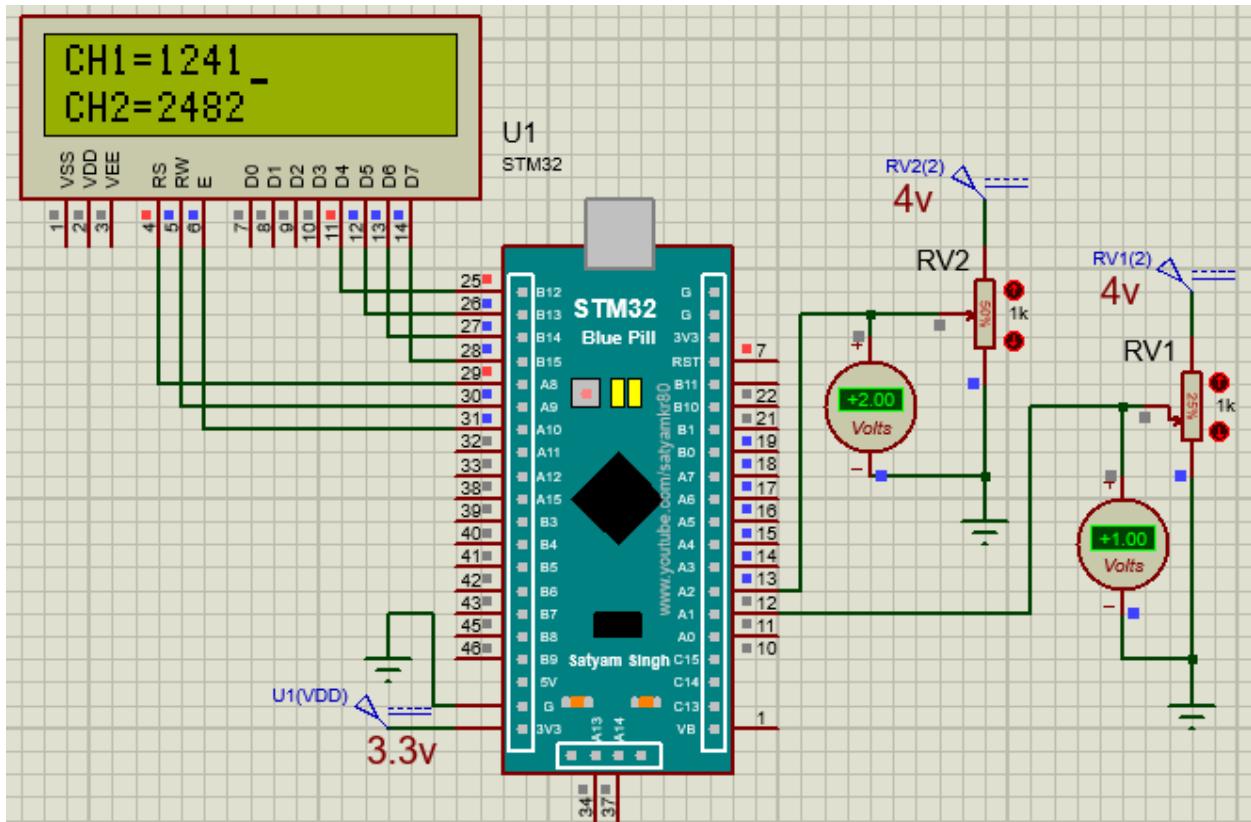
پاسخ: فرض می کنیم دما 10°C باشد. در نتیجه ولتاژ خروجی سنسور 10mV خواهد بود . پس عدد خروجی مبدل برابر است با :

$$\text{ضریب تفکیک}=3.3/(2^{12}-1)=3300\text{mV}/(4095)=0.8058\text{mV}$$

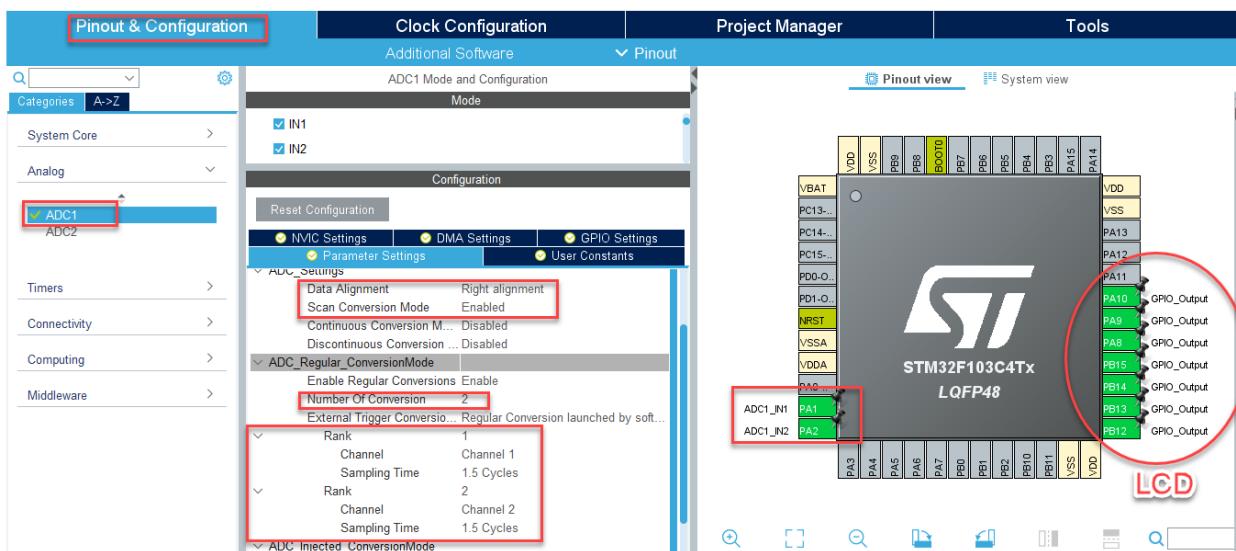
$$\text{عدد خروجی}=10\text{mV}/0.8058\text{mV}=12.41$$

پس کافی است عدد خوانده شده را بر عدد 12.41 تقسیم و سپس نمایش دهیم.

مثال: مطابق شکل زیر دو عدد پتانسیومتر و یک LCD را به میکرو متصل کرده و عدد خروجی کanal ADC1_IN2 را روی خط اول و عدد خروجی کanal ADC1_IN1 را روی خط دوم LCD نمایش دهید.



پاسخ: در Cube باید مانند شکل زیر دو کanal را ، راه اندازی و پیکر بندی کنیم.



و برنامه به صورت زیر است.

```

while (1)

{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

        hadc1.Init.NbrOfConversion=1;

        HAL_ADC_Init(&hadc1);

        HAL_ADC_Start(&hadc1);

        if(HAL_ADC_PollForConversion(&hadc1,100)==HAL_OK)

        {

            a=HAL_ADC_GetValue(&hadc1);

            sprintf(t,"CH1=%04d",a);

            lcd_gotoxy(0,0);

            lcd_puts(t);
}

```

```
    }

    HAL_ADC_Stop(&hadc1);

    HAL_Delay(100);

hadc1.Init.NbrOfConversion=2;

HAL_ADC_Init(&hadc1);

    HAL_ADC_Start(&hadc1);

    if(HAL_ADC_PollForConversion(&hadc1,100)==HAL_OK)

    {

        a=HAL_ADC_GetValue(&hadc1);

        sprintf(t,"CH2=%04d",a);

        lcd_gotoxy(0,1);

        lcd_puts(t);

    }

    HAL_ADC_Stop(&hadc1);

    HAL_Delay(100);

}

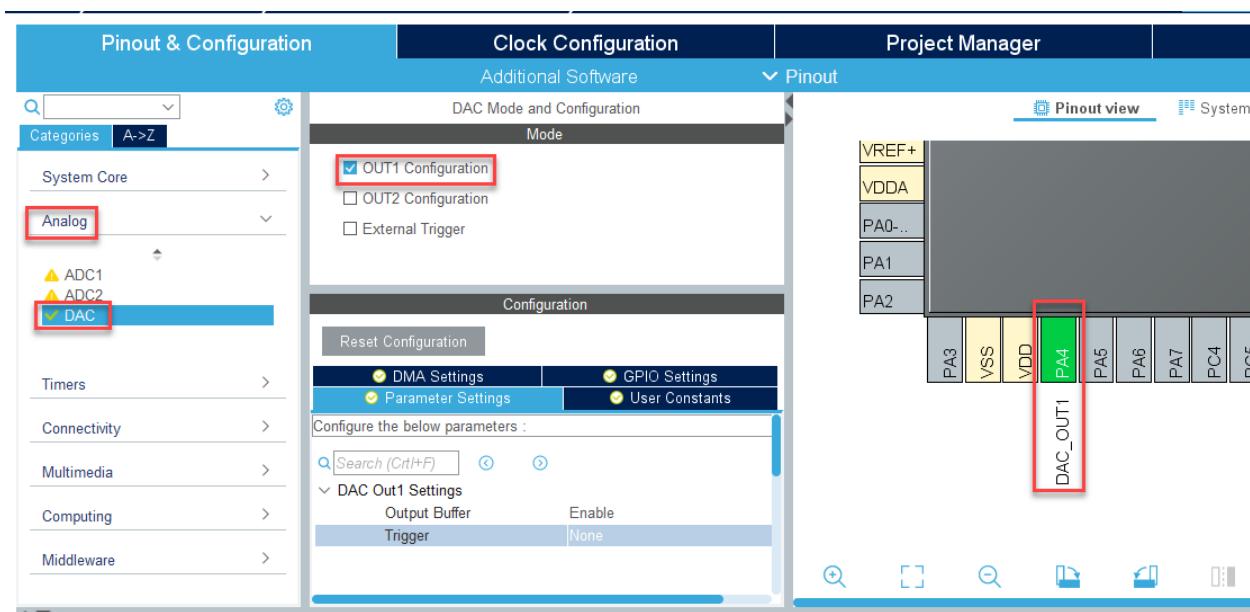
/* USER CODE END 3 */
```

مبدل دیجیتال به آنالوگ : DAC

مبدل های دیجیتال به آنالوگ یا DAC بخش اصلی پروژه هایی هستند که در آنها می خواهیم دیتای دیجیتال را به شکلی که دستگاه های آنالوگ بتوانند با آنها ارتباط برقرار کنند، تبدیل کنیم. به زبان ساده DAC ها گام ضروری در تبدیل صفر و یک های کامپیوتر به ویدیو، صدا و سایر داده های حسی است که می توانید درک کنید.

در اکثر مدل های این میکرو یک واحد DAC وجود دارد.

برای راه اندازی این واحد با توجه به شکل زیر تنظیم های لازم را در cube انجام می دهیم .



پس از فعال کردن این واحد یکی از پایه های میکرو به عنوان خروجی DAC پیکربندی می شود . ولتاژ روی این پایه از رابطه زیر محاسبه می شود.

$$V_{out} = \frac{V_{ref}}{4095} * var$$

در این رابطه var عدد ورودی به DAC است، که می تواند بین 0 تا 4095 باشد. در نتیجه ولتاژ خروجی نیز می تواند بین 0 تا Vref تغییر کند.

در برنامه keil از توابع زیر جهت راه اندازی و مقدار دهی به DAC استفاده می شود.

```
HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
```

```
HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,var);
```

در تابع `SetValue` متغیر `var` به عنوان مقدار ورودی به DAC وارد خواهد شد.

مثال: برنامه‌ای بنویسید که در خروجی DAC یک موج دندان‌اره‌ای داشته باشیم.



```
/* USER CODE BEGIN 2 */
```

```
HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
```

```
/* USER CODE END 2 */
```

```
/* Infinite loop */
```

```
/* USER CODE BEGIN WHILE */
```

```
while (1)
```

```
{
```

```
    for(i=0;i<4096;i++)
```

```
{
```

```
        HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,i);
```

```
        for(j=0;j<1000;j++); تاخیر
```

```
}
```

```
}
```

```
/* USER CODE END 3 */
```

مثال: برنامه‌ای بنویسید که در خروجی DAC یک موج مثلثی داشته باشیم.



```
/* USER CODE BEGIN 2 */
```

```
HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
```

```
/* USER CODE END 2 */
```

```
/* Infinite loop */
```

```
/* USER CODE BEGIN WHILE */
```

```
while (1)
```

```
{
```

```
    for(i=0;i<4096;i++)
```

```
{
```

```
        HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,i);
```

```
        for(j=0;j<1000;j++); تاخیر
```

```
}
```

```
    for(i=4095;i>=0;i--)
```

```
{
```

```
        HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,i);
```

```
        for(j=0;j<1000;j++); تاخیر
```

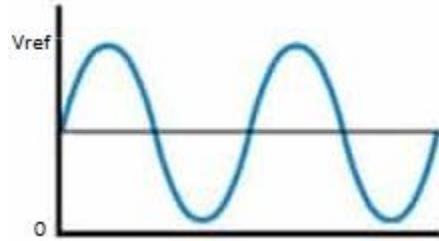
```

    }
}

/* USER CODE END 3 */

```

مثال: برنامه‌ای بنویسید که در خروجی DAC یک موج سینوسی داشته باشیم.



با توجه به روابط مثلثاتی، نامعادله‌های زیر را خواهیم داشت.

$$-1 \leq \sin(x) \leq 1$$

$$-1+1 \leq \sin(x)+1 \leq 1+1$$

$$0 \leq \sin(x)+1 \leq 2$$

$$0 * 2000 \leq 2000 * [\sin(x)+1] \leq 2 * 2000$$

$$0 \leq 2000 * \sin(x) + 2000 \leq 4000$$

توجه داشته باشید که تابع سینوس در کتابخانه `math.h` قرار دارد . و همچنین زوایا بر حسب رادیان محاسبه خواهند شد یعنی زاویه 0 تا 360 درجه باید در بازه 0 تا 2π در تابع سینوس قرار بگیرد.

با توجه به مطالب بالا ، برنامه زیر در خروجی DAC یک موج سینوسی ایجاد می‌کند.

```

/* Private includes -----*/
/* USER CODE BEGIN Includes */

#include "math.h"

/* USER CODE END Includes */
/* USER CODE BEGIN PV */

float x;

```

```

int y;

/* USER CODE END PV */

/* USER CODE BEGIN 2 */

HAL_DAC_Start(&hdac,DAC_CHANNEL_1);

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)

{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

    for(x=0;x<6.28;x=x+0.1)

    {

        y=2000+2000*sin(x);

        HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,y);

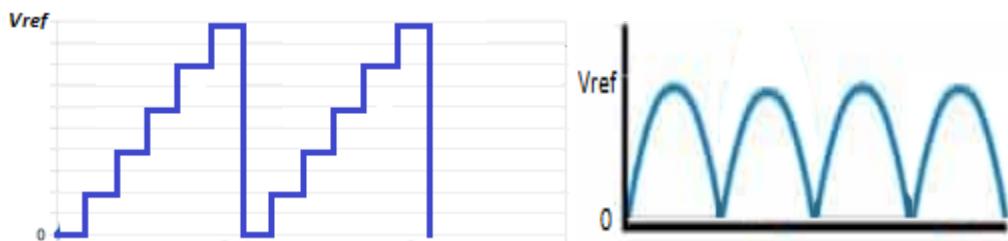
    }

}

/* USER CODE END 3 */

```

تمرین: شکل موج‌های زیر را در خروجی ADC ایجاد کنید.



وقفه : Interrupt

در این میکرو ۱۶ وقفه خارجی وجود دارد که در سه گروه دسته بندی می‌شوند.

stm32f1xx_it.c INT0 تا INT4 در دسته اول قرار می‌گیرند. با فعال شدن هر یک از این وقفه‌ها در فایل
یک زیر برنامه برای هر یک از وقفه‌ها ایجاد می‌شود که می‌توان روتین سرویس مربوط به آن وقفه را درون آن
نوشت. برای مثال اگر INT1 و INT2 را فعال کنیم، دو زیر برنامه به صورت زیر در فایل c
ایجاد می‌شود.

```
void EXTI1_IRQHandler(void)
{
    /* USER CODE BEGIN EXTI1_IRQn 0 */
    /* USER CODE END EXTI1_IRQn 0 */

    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
    /* USER CODE BEGIN EXTI1_IRQn 1 */

    کد کاربر برای روتین سرویس وقفه یک

    /* USER CODE END EXTI1_IRQn 1 */
}

/**
 * @brief This function handles EXTI line2 interrupt.
 */

```

```
void EXTI2_IRQHandler(void)
{
    /* USER CODE BEGIN EXTI2_IRQn 0 */
    /* USER CODE END EXTI2_IRQn 0 */

    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_2);
}
```

```
/* USER CODE BEGIN EXTI2_IRQHandler 1 */
```

کد کاربر برای روتین سرویس وقفه دو

```
/* USER CODE END EXTI2_IRQHandler 1 */
```

```
}
```

INT5 تا INT9 در دسته دوم قرار می‌گیرند. برای تمام این پنج وقفه فقط یک زیر برنامه به صورت زیر در فایل stm32f1xx_it.c ایجاد می‌شود.

```
void EXTI9_5_IRQHandler(void)
```

```
{
```

```
/* USER CODE BEGIN EXTI9_5_IRQHandler 0 */
```

```
/* USER CODE END EXTI9_5_IRQHandler 0 */
```

```
HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_6);
```

```
/* USER CODE BEGIN EXTI9_5_IRQHandler 1 */
```

```
/* USER CODE END EXTI9_5_IRQHandler 1 */
```

```
}
```

INT10 تا INT15 در دسته سوم قرار می‌گیرند. برای تمام این شش وقفه فقط یک زیر برنامه به صورت زیر در فایل stm32f1xx_it.c ایجاد می‌شود.

```
void EXTI15_10_IRQHandler(void)
```

```
{
```

```
/* USER CODE BEGIN EXTI15_10_IRQHandler 0 */
```

```
/* USER CODE END EXTI15_10_IRQHandler 0 */
```

```
HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_11);
```

```
HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_12);
```

```
/* USER CODE BEGIN EXTI15_10_IRQHandler 1 */
```

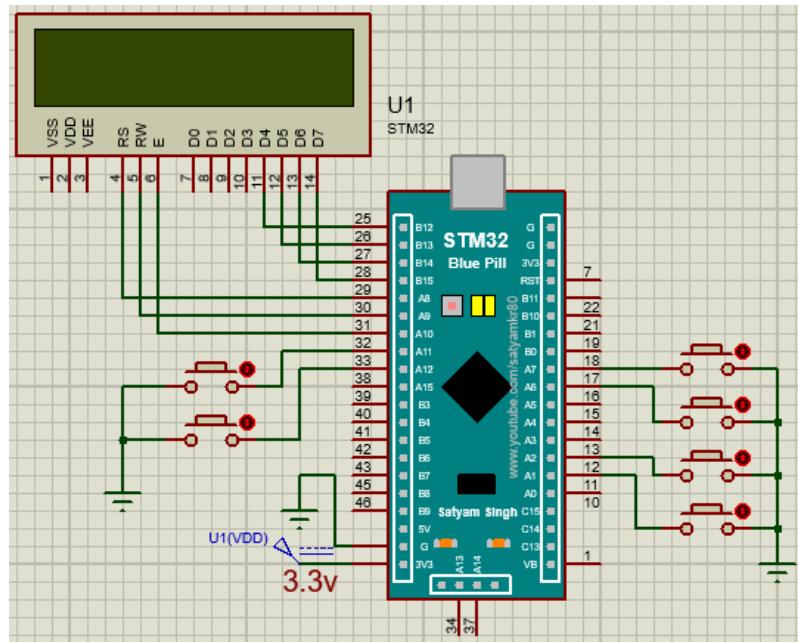
```
/* USER CODE END EXTI15_10_IRQHandler 1 */
}
```

حال سوال این است که چگونه تشخیص دهیم کدام وقفه از این دسته فعال شده است. در توابع HAL تابعی به نام Callback برای پشتیبانی از وقفه‌های خارجی و داخلی که توسط واحدهای جانبی تولید می‌شود وجود دارد. بدنه این تابع را می‌توانید از داخل فایل stm32f1xx_hal_gpio.c کپی و به فایل main.c اضافه کنید. سپس با استفاده از دستور شرطی if مشخص می‌کنیم که کدام وقفه فعال شده و روتین سرویس مورد نظر را اضافه می‌کنیم.

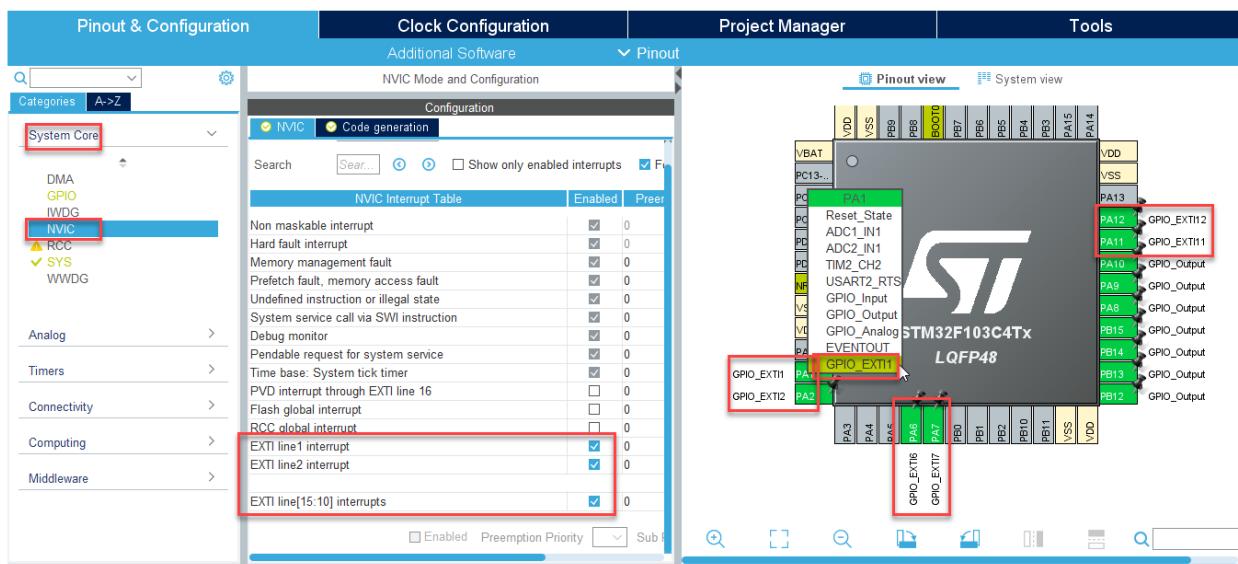
```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(GPIO_Pin);
    /* NOTE: This function Should not be modified, when the callback is needed,
       the HAL_GPIO_EXTI_Callback could be implemented in the user file
    */
    if(GPIO_Pin==GPIO_PIN_10)
    {
        روتین سرویس
    }
}
```

نکته: در داخل روتین سرویس از تابع HAL_DELAY استفاده نکنید. زیرا باعث اختلال در عملکرد برنامه شده و دیگر از روتین سرویس خارج نمی‌شود.

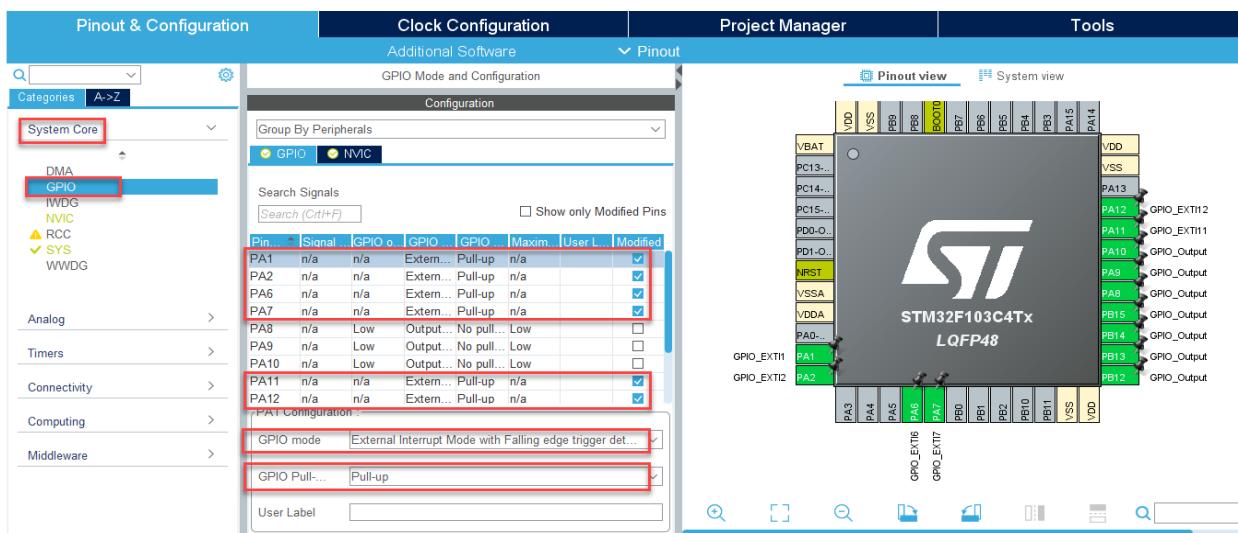
مثال: مانند شکل زیر، شش عدد کلید به وقفه‌های INT1,2,6,7,11,12 و یک LCD به میکرو متصل می-کنیم. می‌خواهیم برنامه‌ای بنویسیم که روی خط اول LCD یک شمارنده بالا شمار 0 تا 15 در حال اجرا باشد (برنامه اصلی) و با زدن هر کلید وقفه‌ای ایجاد شده و با توجه به شماره وقفه تولید شده، عبارت INT1 تا INT12 بر روی خط دوم LCD نمایش داده شود.



پاسخ : در نرم افزار cube طبق شکل زیر روی پایه‌های مورد نظر کلیک کرده ، و وظیفه پایه را بر روی وقفه خارجی GPIO_EXTIn قرار می‌دهیم. سپس از بخش NVIC گزینه System Core را انتخاب و در پنجره NVIC وقفه‌های لیست شده را فعال می‌کنیم.



همچنین طبق شکل زیر از بخش System Core گزینه GPIO را انتخاب و در پنجره GPIO هر یک از پایه‌ها را که وقفه‌ای روی آن فعال شده انتخاب و سپس در قسمت GPIO mode مشخص می‌کنیم که اجرای روتین سرویس وقفه در چه حالتی از سیگنال وقفه صورت پذیرد. در بخش GPIO-Pull نیز می‌توان مشخص کرد که پایه میکرو از داخل Pull_Down یا Pull_Up و یا در حالت عادی خود باشد. در این مثال پایه‌ها از داخل Pull_up و پاسخ وقفه، روی لبه پایین رونده قرار داده شده است.



روش مناسب برای اجرای اجرای روتین سرویس وقفه‌ها استفاده از تابع CALL_BACK می‌باشد. برای این مثال نیز از همین تابع استفاده می‌کنیم.

برنامه اصلی (شمارنده ۰ تا ۱۵)

```
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    for(i=0;i<=15;i++)
    {
        sprintf(t,"%02d",i);
        lcd_gotoxy(7,0);
        lcd_puts(t);
        HAL_Delay(500);
    }

}
```

روتین‌های سرویس که در قسمت **USER CODE BEGIN 4** نوشته شده است.

```
/* 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
```

*/

```
if(GPIO_Pin==GPIO_PIN_1)
{
    lcd_gotoxy(0,1);
    lcd_puts("INT1 ");
}

if(GPIO_Pin==GPIO_PIN_2)
{
    lcd_gotoxy(0,1);
    lcd_puts("INT2 ");
}

if(GPIO_Pin==GPIO_PIN_6)
{
    lcd_gotoxy(0,1);
    lcd_puts("INT6 ");
}

if(GPIO_Pin==GPIO_PIN_7)
{
    lcd_gotoxy(0,1);
    lcd_puts("INT7 ");
}

if(GPIO_Pin==GPIO_PIN_11)
{
    lcd_gotoxy(0,1);
```

```

lcd_puts("INT11");

}

if(GPIO_Pin==GPIO_PIN_12)
{

    lcd_gotoxy(0,1);
    lcd_puts("INT12");

}

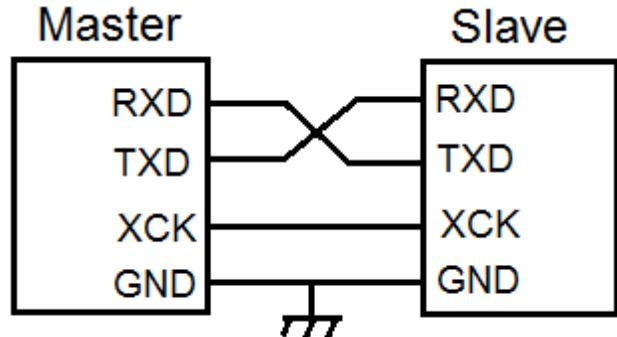
}

/* USER CODE END 4 */

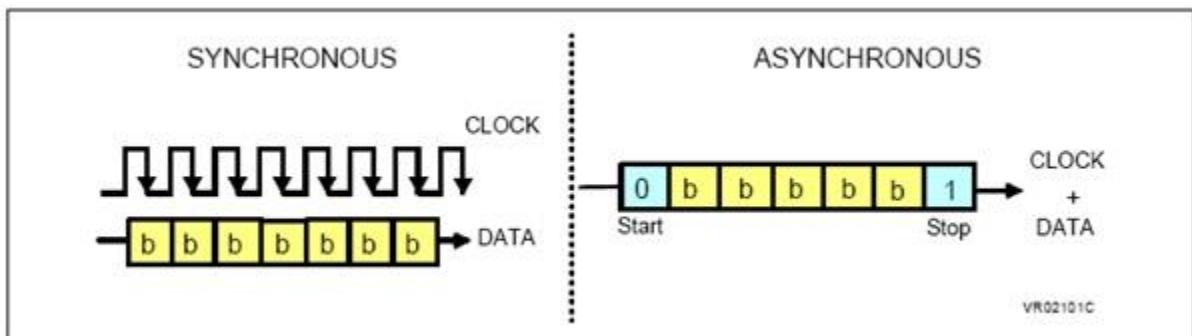
```

The Universal Synchronous and Asynchronous serial Receiver and Transmitter :USART

در این نوع ارتباط سیم بندی زیر را داریم .



در حالت سنکرون دیتا روی لبه بالا یا پایین رونده پالس ساعت خوانده می شود.

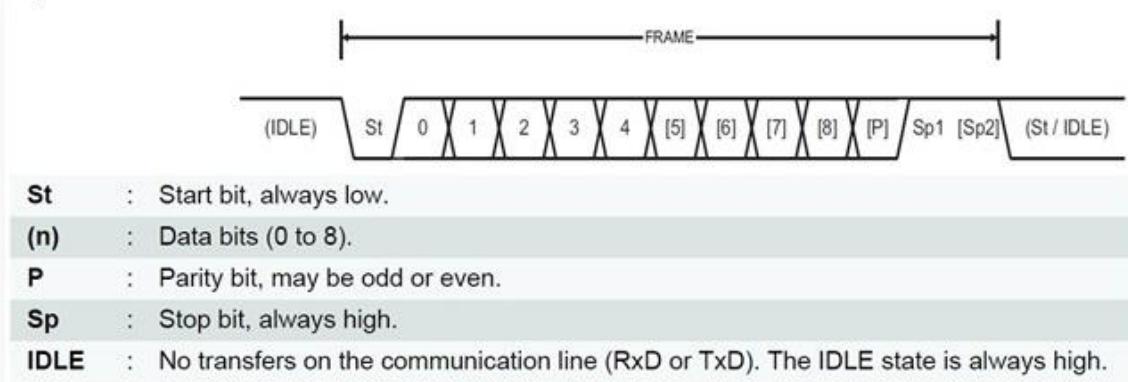


قاب دیتا : Frame Format

هنگام ارسال دیتا در حالت Asynchronous علاوه بر دیتا ، بیت های کنترلی نیز همراه با آن ارسال می شود به مجموعه بیت های کنترلی و دیتا فریم Frame گفته می شود. این بیت های کنترلی عبارتند از:

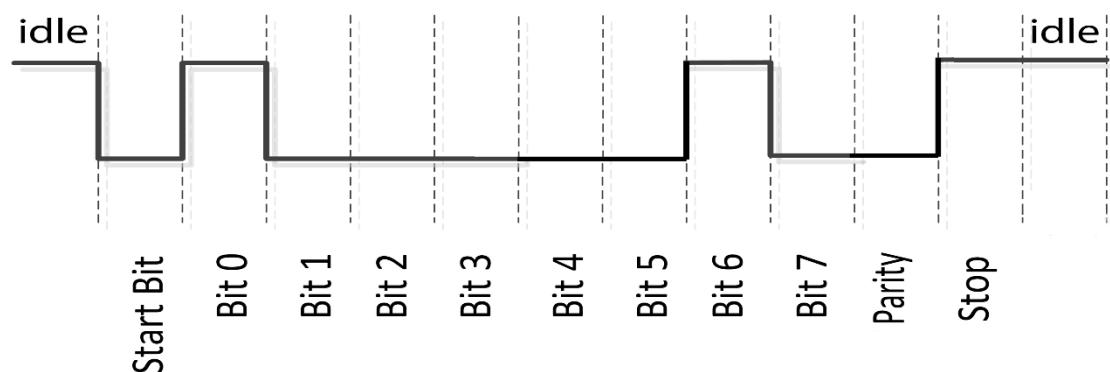
بیت شروع Start bit : این بیت که همواره صفر است به گیرنده شروع ارسال دیتا را خبر می دهد.
بیت پایان Stop bit : این بیت که همواره یک است به گیرنده اتمام ارسال دیتا را خبر می دهد. تعداد این بیت می تواند یک بیت ، دو بیت و در بعضی سیستم ها یک و نیم بیت باشد.
بیت توازن Parity bit : این بیت که برای آشکارسازی خط استفاده می شود می تواند توازن زوج یا فرد باشد.

Figure 23-5. Frame formats.



مثال: فرض کنید می خواهیم کد اسکی حرف A را با توازن زوج و یک بیت توقف ارسال کنیم قاب دیتای آن را رسم کنید . (A= 65 = 01000001)

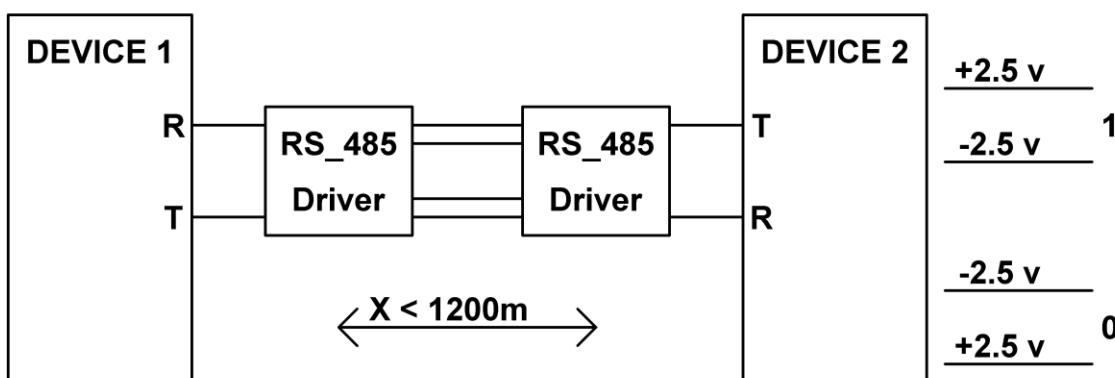
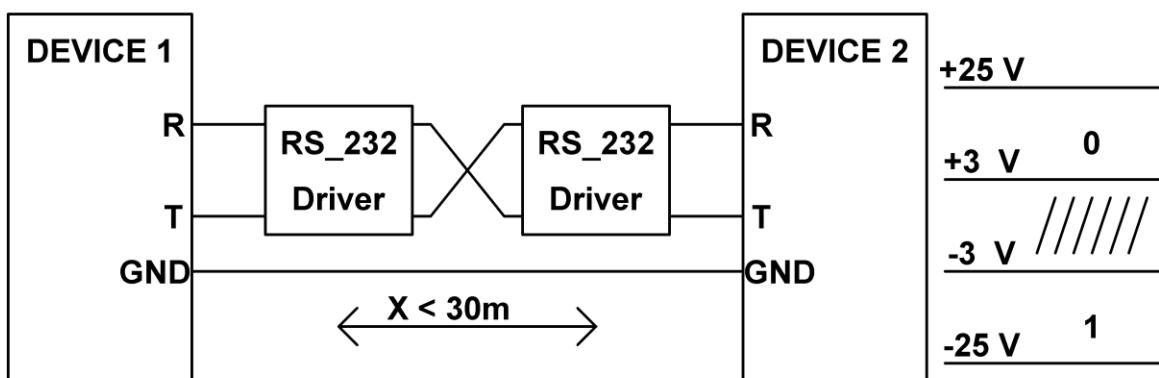
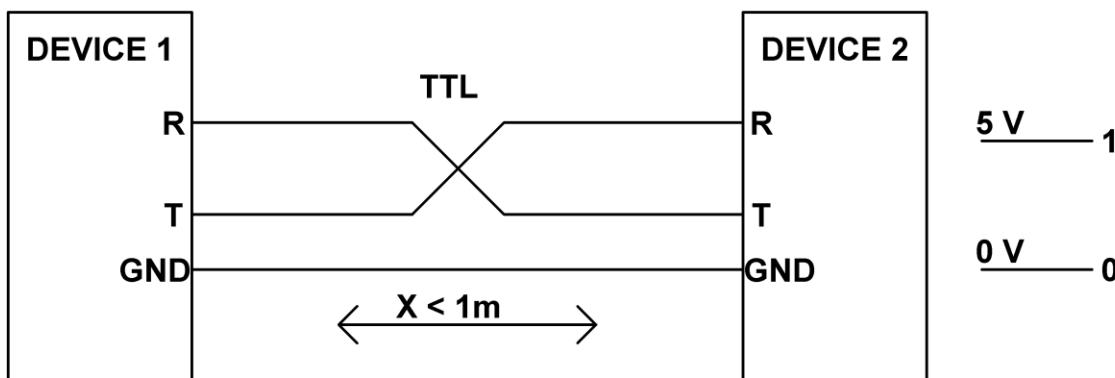
0x41, 801 (8 Data bits, Odd Parity, 1 Stop)



روش های ارتباط سخت افزاری بین دو دستگاه در : UART

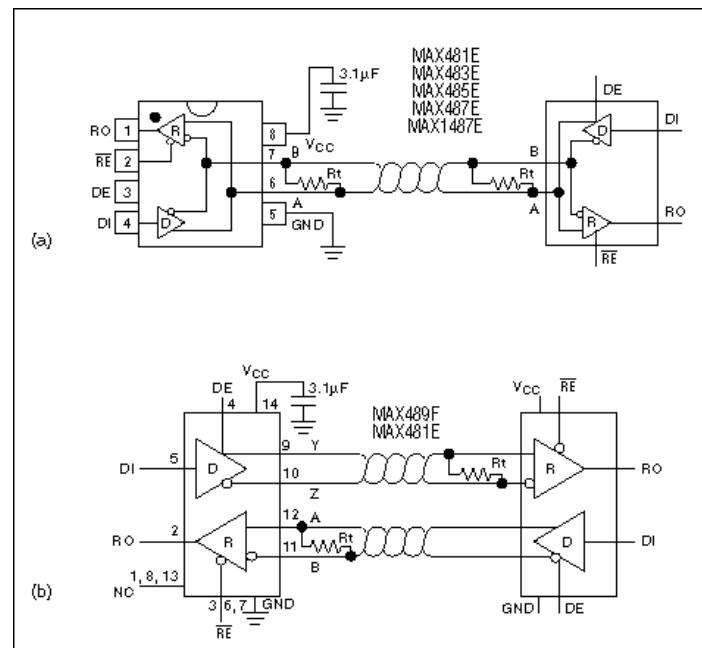
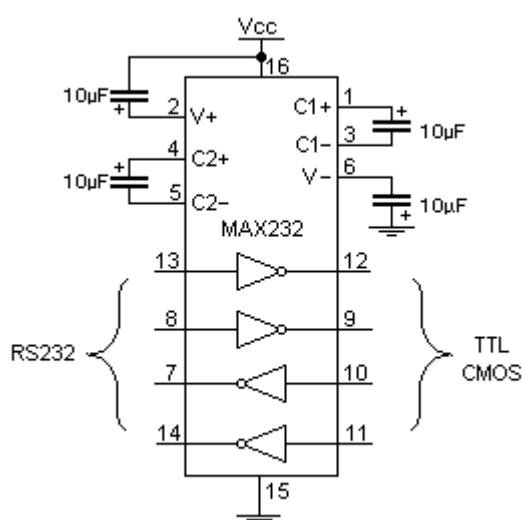
سه روش **TTL** ، **RS485** و **RS232** پر کاربرد تر می باشند.

توجه داشته باشید فاصله های مشخص شده بین فرستنده و گیرنده حالت عمومی دارد و با توجه به نویز محیط ، جنس کابل و سرعت انتقال دیتا **Baud Rate** تغییر می کند.



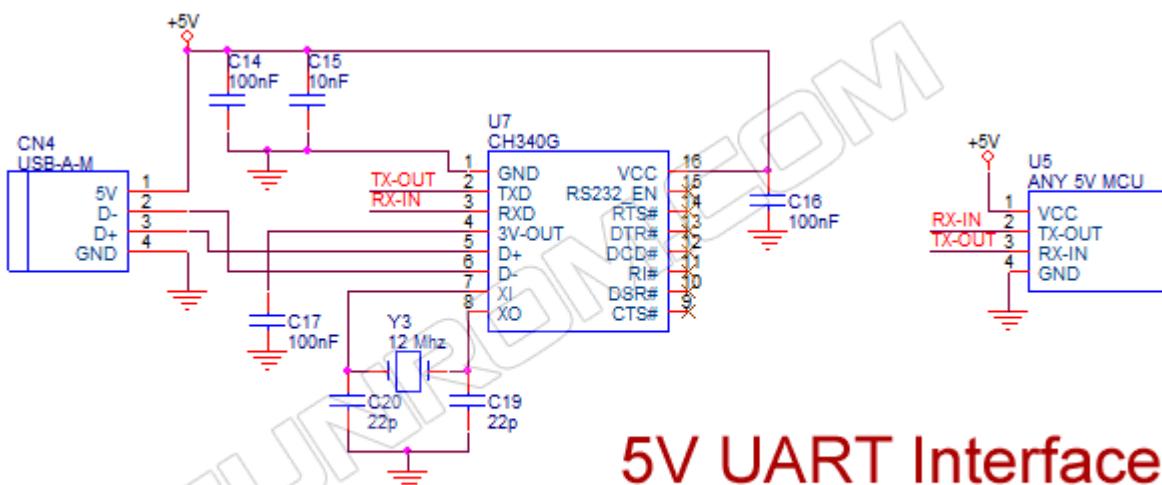
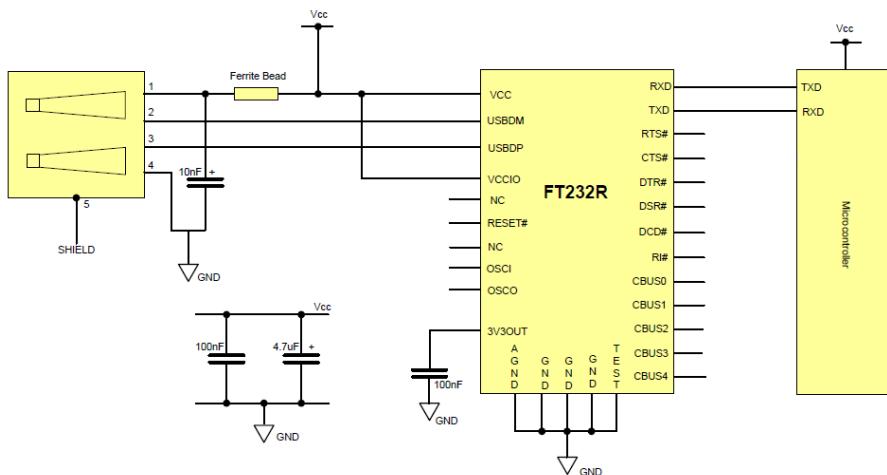
Data rate (bps)	Distance (m)
2400	60
4800	30
9600	15
19200	7.6
38400	3.7
56000	2.6

برای درایور RS232 می توانید از چیپ MAX232 و برای درایور RS485 از چیپ MAX487 یا 75179 یا 75176 استفاده کنید.

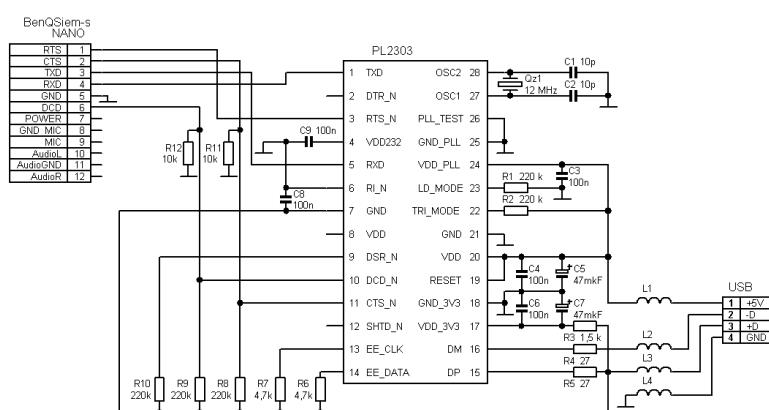


: USB به UART مبدل

برای این منظور می توانید از چیپ های FT232 , PL2303 , CH340G استفاده کنید.



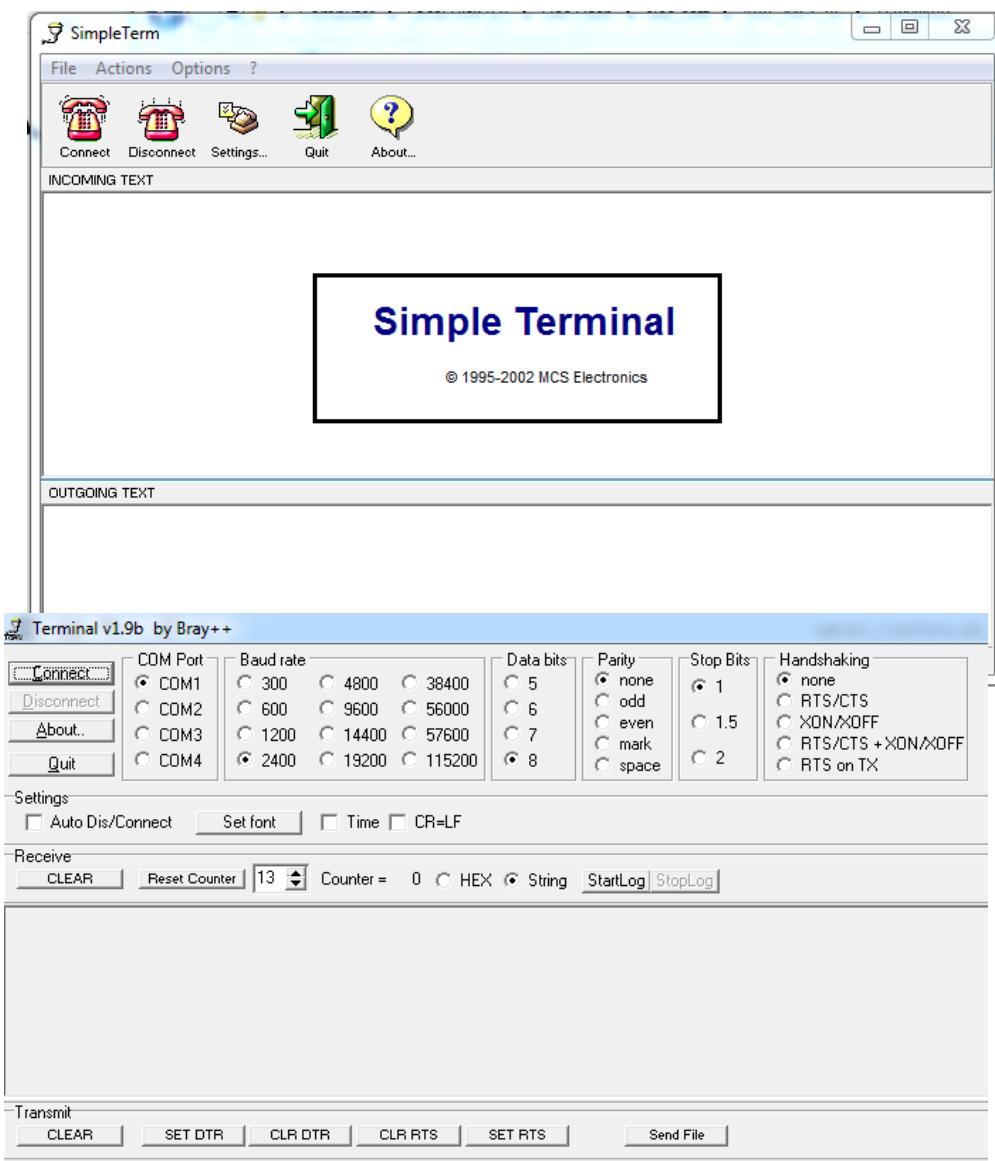
Sunrom Technologies	http://www.sunrom.com
Title CH340G TO 5V MCU INTERFACING	Rev 1
Code 4511	
Date: Friday, April 17, 2015	Sheet 1 of 1



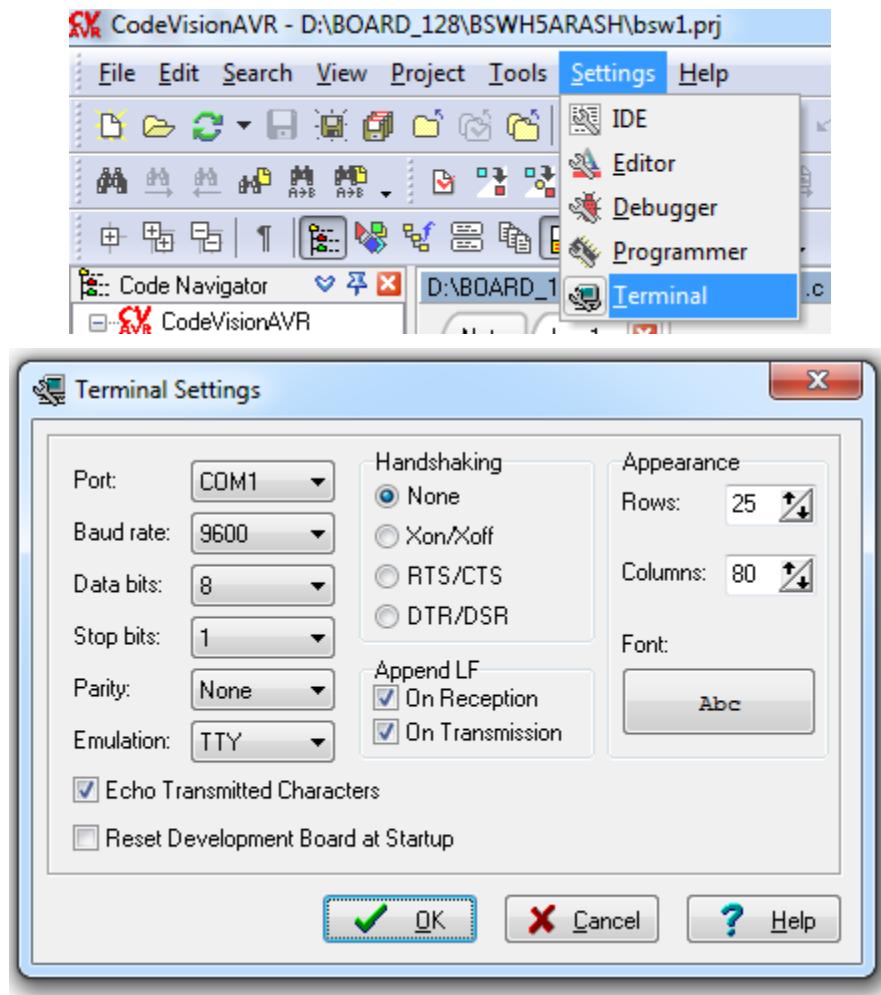
نرم افزارهای ترمینال Terminal

اگر بخواهیم میکرو را با استفاده از **UART** به کامپیوتر وصل کنیم لازم است که در کامپیوتر نرم افزاری داشته باشیم تا ارتباط ما را با پورت سریال برقرار کند به این نرم افزارها **Terminal** گفته می شود . در فایل های مربوط به درس دونرم افزار **simpleterm** و **CodeVision** در اختیار شما قرار گرفته ، در نرم افزار **CodeVision** نیز یک ترمینال وجود دارد.

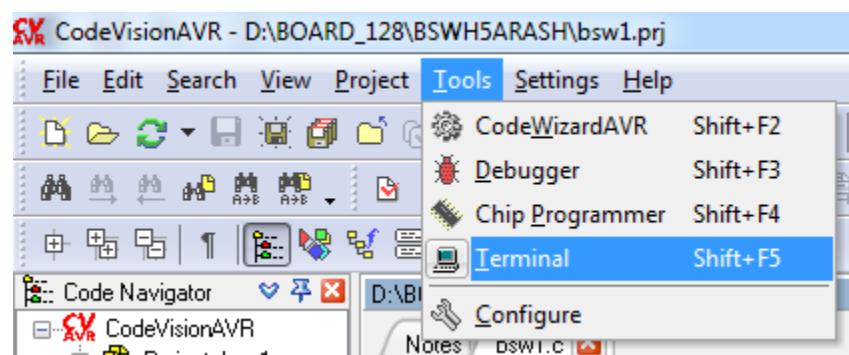
- توجه داشته باشید که لازم است قبل از استفاده از ترمینال باید آن را تنظیم کنید.



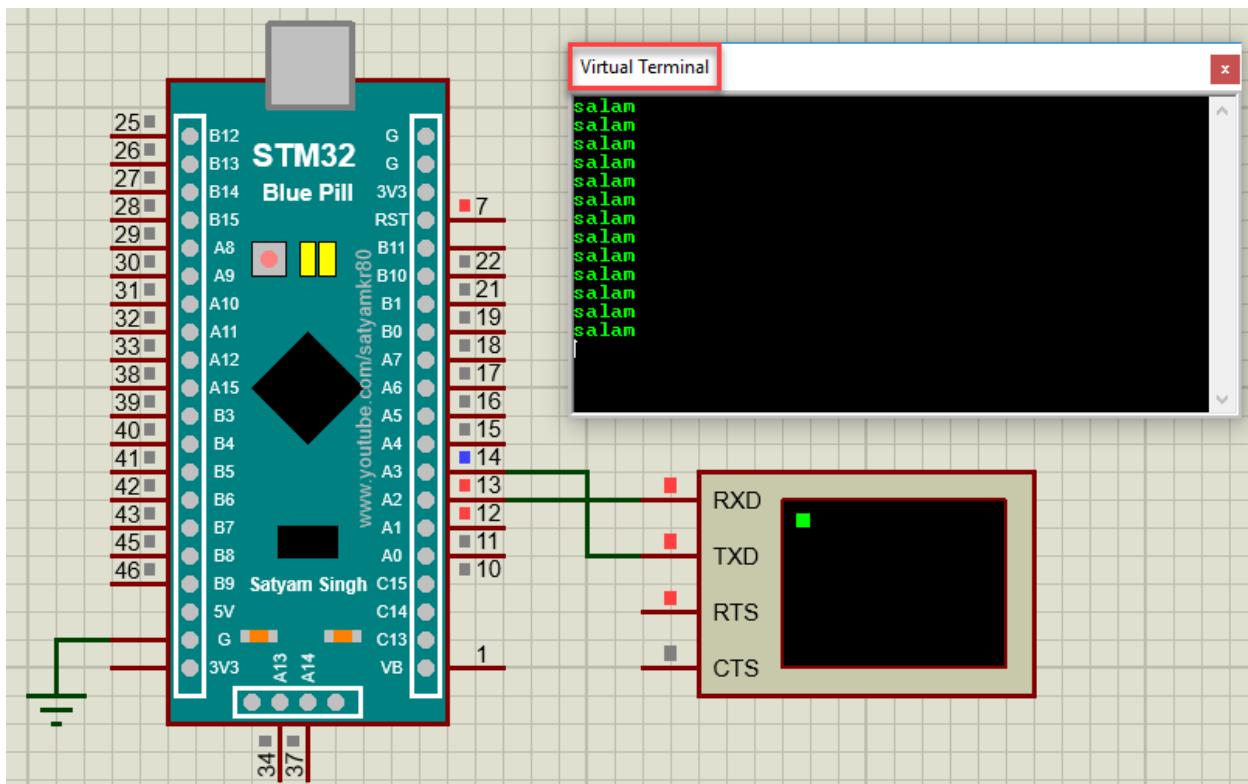
اگر می خواهید از ترمینال موجود در **CodeVision** استفاده کنید ابتدا از مسیر زیر تنظیم کنید.



برای استفاده از **Terminal** نیز از مسیر زیر اقدام کنید.



مثال: برنامه ای بنویسید که میکرو هر دو ثانیه یک بار کلمه **SALAM** را به کامپیوتر ارسال نماید.



در برنامه پروتئوس می‌توانید به جای کامپیوتر از ترمینال مجازی استفاده کنید.

ابتدا در برنامه cube مطابق با شکل زیر تنظیم‌های لازم را برای یکی از واحدهای سریال انجام می‌دهیم.

The screenshot shows the STM32CubeMX software interface with the following configuration details:

- Pinout & Configuration**: Shows the STM32F103C4Tx LQFP48 pinout with USART2_TX (PA2) and USART2_RX (PA3) highlighted.
- Clock Configuration**: Shows the clock configuration for USART2.
- Project Manager**: Shows the project manager interface.
- Tool**: Shows the tool interface.
- USART2 Mode and Configuration** tab:
 - Mode**: Asynchronous
 - Configuration** tab:
 - Reset Configuration**
 - NVIC Settings**, **DMA Settings**, **GPIO Settings** (radio buttons)
 - Parameter Settings** (radio button selected)
 - Search (Ctrl+F)**
 - Basic Parameters**:
 - Baud Rate: 9600 Bits/s
 - Word Length: 8 Bits (including Parity)
 - Parity: None
 - Stop Bits: 1
 - Advanced Parameters**:
 - Data Direction: Receive and Transmit
 - Over Sampling: 16 Samples

در keil برای ارسال دیتا ، از طریق UART می توان از تابع زیر استفاده کرد.

```
HAL_UART_Transmit(&huartx,(unsigned char *)Data,TimeOut);
```

استراکت &huartx : حاوی تنظیمات پورت سریال می باشد، که در آن x می تواند عدد ۱ و ۲ و ۳... باشد.

متغیر رشته ای: یک متغیر رشته ای ، حاوی دیتایی که می خواهیم ارسال کنیم. برای مثال:

```
char d[ ]="SALAM\n\r";
```

تعداد کاراکتر: مشخص می کند چند کاراکتر از رشته مورد نظر ارسال شود. برای رشته بالا باید عدد ۷ قرار دهیم. (\n و \r باعث می شود رشته ها در گیرنده به خط بعدی منتقل شود)

TimeOut : مشخص می کند تابع حداکثر تا چه زمانی برای ارسال دیتا سعی کند و بعد از آن از تابع خارج شود.

با توجه به توضیحات بالا برنامه به شکل زیر خواهد بود.

```
/* USER CODE BEGIN PV */
```

```
char d[]="salam\n\r";
```

```
/* USER CODE END PV */
```

```
/* USER CODE BEGIN WHILE */
```

```
while (1)
```

```
{
```

```
/* USER CODE END WHILE */
```

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

```
    HAL_UART_Transmit(&huart2,(unsigned char *)d,7,100);
```

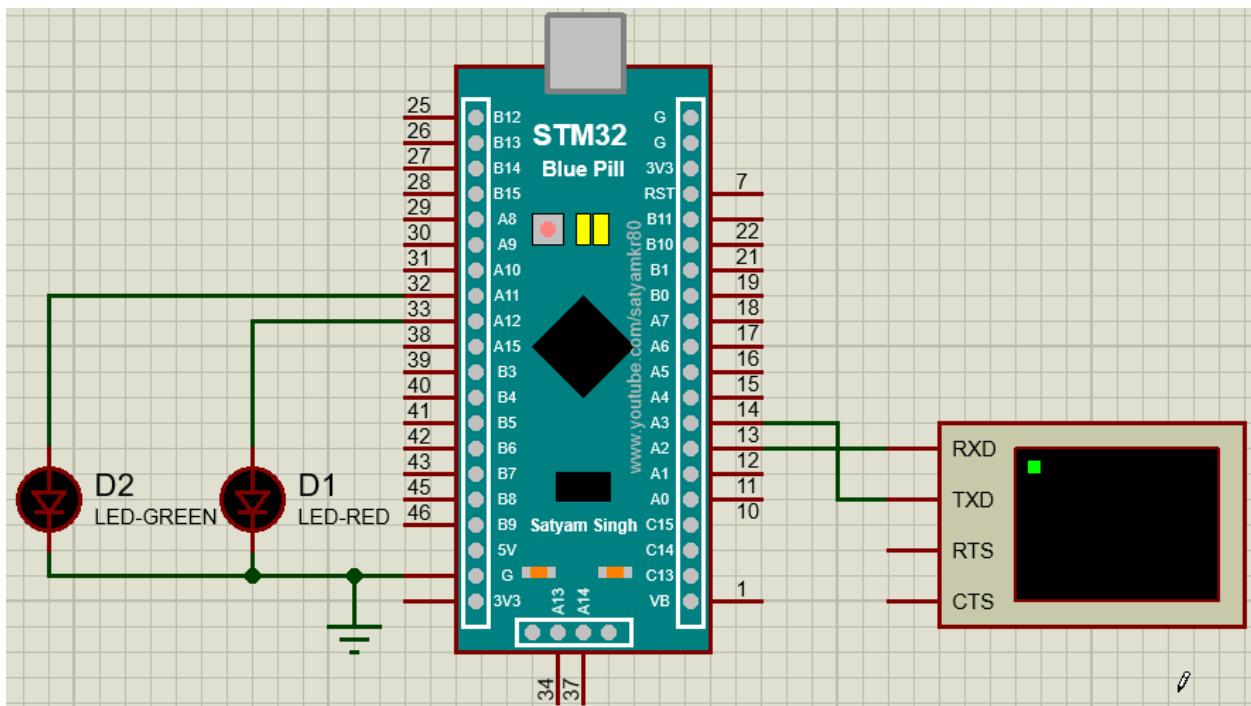
```
    HAL_Delay(500);
```

```
}
```

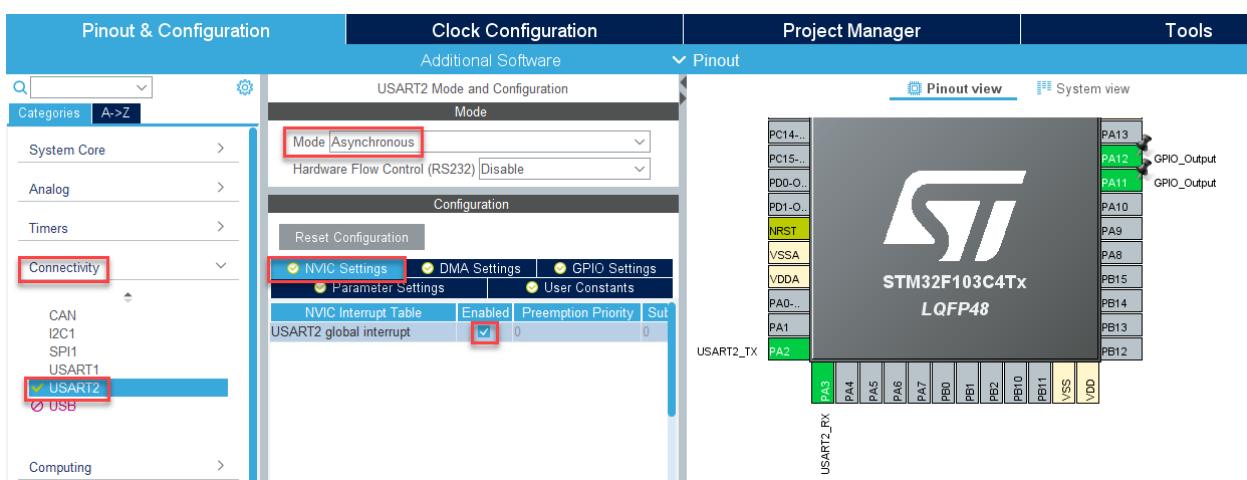
```
/* USER CODE END 3 */
```

مثال: برنامه ای بنویسید که کاراکتری از پورت سریال دریافت کند. اگر کاراکتر دریافتی r بود LED قرمز ، اگر g

بود LED سبز روشن شود و اگر کاراکتر O بود هر دو LED خاموش شوند.



در برنامه cube علاوه بر تنظیم‌های مثال قبل لازم است که وقفه بخش UART را نیز از مسیر نشان داده شده در تصویر زیر فعال کنیم.



تابع خواندن دیتای ورودی از پورت سریال بر اساس وقفه واحد UART به صورت زیر می‌باشد.

تعداد کارکتر ورودی ، منغیر رشته‌ای (HAL_UART_Receive_IT(&huart2,(unsigned char *);

عدد مشخص شده در آرگومان "تعداد کاراکتر ورودی" در متغیر `huart2.RxXferCount` ذخیره می‌شود و با ورود هر کاراکتر به پورت سریال یک واحد از آن کم می‌شود در نتیجه هرگاه این متغیر صفر شود یعنی به تعداد مشخص شده دیتا دریافت شده است، پس می‌توان با چک کردن این متغیر از تکمیل فرآیند ورود دیتا مطلع شد و برنامه مورد نظر را اجرا کرد.

```

/* USER CODE BEGIN PV */

char h[20];

/* USER CODE END PV */

/* USER CODE BEGIN 2 */

HAL_UART_Receive_IT(&huart2,(unsigned char *)h,1);

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)

{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

    if(huart2.RxXferCount==0)

    {

        if(h[0]=='r')

        {

            HAL_GPIO_WritePin(GPIOA,GPIO_PIN_12,GPIO_PIN_SET);

            HAL_GPIO_WritePin(GPIOA,GPIO_PIN_11,GPIO_PIN_RESET);

    }
}

```

```

        }

        if(h[0]=='g')

        {

HAL_GPIO_WritePin(GPIOA,GPIO_PIN_11,GPIO_PIN_SET);

HAL_GPIO_WritePin(GPIOA,GPIO_PIN_12,GPIO_PIN_RESET);

        }

        if(h[0]=='o')

        {

HAL_GPIO_WritePin(GPIOA,GPIO_PIN_11,GPIO_PIN_RESET);

HAL_GPIO_WritePin(GPIOA,GPIO_PIN_12,GPIO_PIN_RESET);

        }

        HAL_UART_Receive_IT(&huart2,(unsigned char *)h,1);

    }

/* USER CODE END 3 */

```

روش دیگر برای دریافت دیتا از طریق وقفه استفاده از **CallBack** بخش **UART** میباشد.

```

/* USER CODE BEGIN 2 */

HAL_UART_Receive_IT(&huart2,(unsigned char *)h,1);

/* USER CODE END 2 */

```

```
/* Infinite loop */  
/* USER CODE BEGIN WHILE */  
while (1)  
{  
/* USER CODE END WHILE */  
  
/* USER CODE BEGIN 3 */  
}  
/* USER CODE END 3 */
```

در انتهای فایل **main.c** می‌باشد.

```
/* USER CODE BEGIN 4 */  
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)  
{  
if(h[0]=='r')  
{  
  
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_12,GPIO_PIN_SET);  
  
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_11,GPIO_PIN_RESET);  
}  
if(h[0]=='g')  
{
```

```

HAL_GPIO_WritePin(GPIOA,GPIO_PIN_11,GPIO_PIN_SET);

HAL_GPIO_WritePin(GPIOA,GPIO_PIN_12,GPIO_PIN_RESET);

}

if(h[0]=='o')

{

HAL_GPIO_WritePin(GPIOA,GPIO_PIN_11,GPIO_PIN_RESET);

HAL_GPIO_WritePin(GPIOA,GPIO_PIN_12,GPIO_PIN_RESET);

}

HAL_UART_Receive_IT(&huart2,(unsigned char *)h,1);

}

/* USER CODE END 4 */

```

:Clock

می دانیم که پردازنده ها از تعداد زیادی مدارهای ترتیبی و ترکیبی ساخته شده اند. برای هماهنگی بین قسمت های مختلف نیاز به یک پالس ساعت داریم که همه قسمت ها را با هم هماهنگ کند. برای تامین این پالس در میکروهای ST سه روش وجود دارد.

الف) اسیلاتور داخلی (HSI)

ب) اسیلاتور خارجی bypass clock source

ج) اسیلاتور داخلی با کریستال خارجی high-speed external (HSE)clock

حلقه قفل شده فاز PLL :

PLL یک سیستم کنترلی الکترونیکی است که یک موج سینوسی یا مربعی با فرکانس و فاز معین و پایدار، متناسب با ورودی یا مرجع می‌سازد.

PLL یک موتور تولید پالس ساعت در میکرو است که برای تولید پالس ساعت با فرکانس بسیار بالاتر از HSI داخلی یا ساعت خارجی استفاده می‌شود. اگر میکرو را با فرکانس کم HSE یا HSI راه اندازی کنید، اکثر تجهیزات جانبی مانند USB و Ethernet PHY نمی‌توانند کار کنند. بنابراین، در این موارد می‌توانید از PLL کمک بگیرید.

گذرگاه AHB و APB :

فرکانس خروجی PLL برای برخی از واحدهای جانبی مناسب ولی برای برخی زیاد است در نتیجه تقسیمی از فرکانس خروجی PLL برای وسایل جانبی ارسال می‌شود. در نتیجه در داخل میکرو دونوع BUS داریم

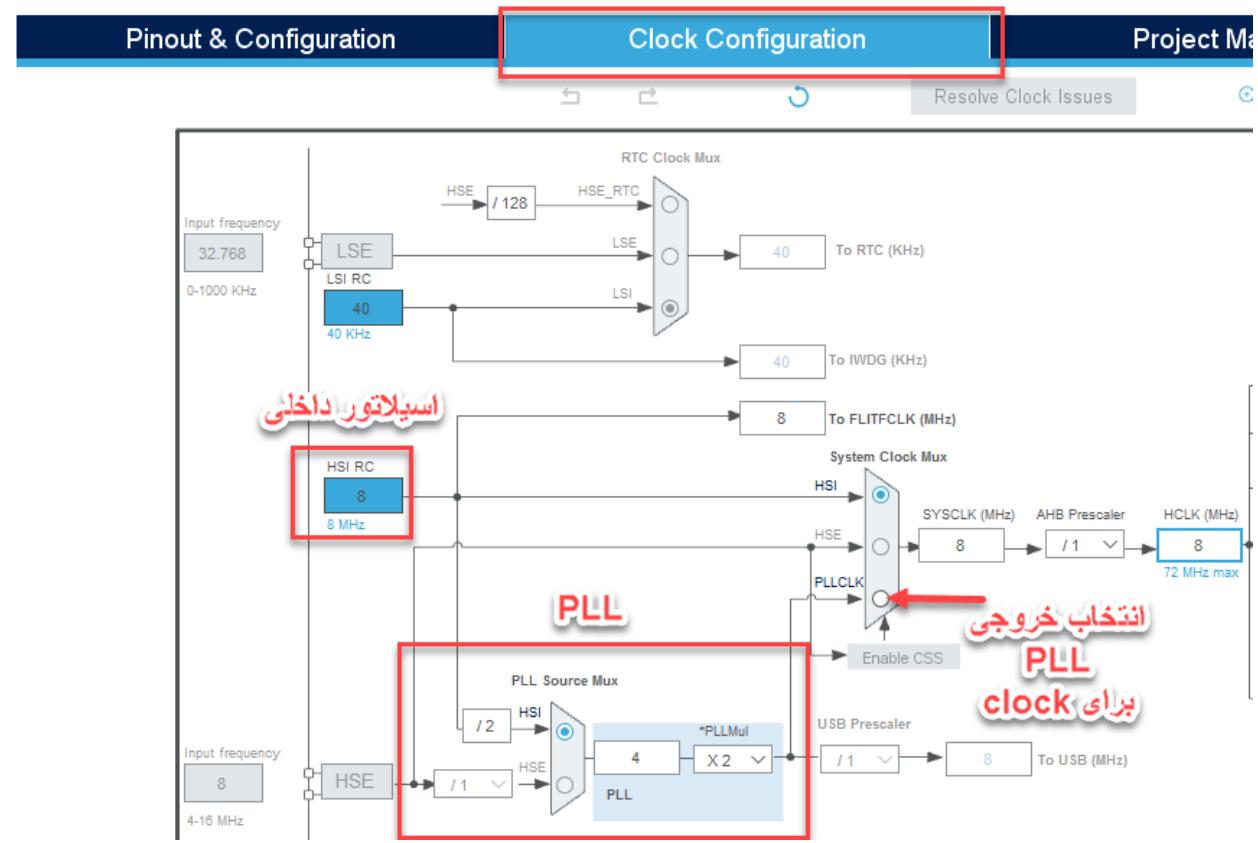
Advanced High-performance Bus **AHB(۱)**

ARM Peripheral Bus APB (۲)

فرکانس کار واحد های روی بس AHB بالا و فرکانس کار واحد های روی بس APB پائین است و برای هریک فرکانس مخصوص آن تولید می‌شود.

: Clock

در حالت پیش فرض ، میکرو با اسیلاتور RC داخلی راه اندازی می‌شود. برای مشاهده و پیکربندی های لازم وارد صفحه Clock Configuration می‌شویم. می‌توان با تغییر در پارامترهای بخش PLL فرکانس کار مورد نظر را برای میکرو ایجاد کرد . و با انتخاب گزینه PLLCLK آن را به میکرو اعمال نمود.



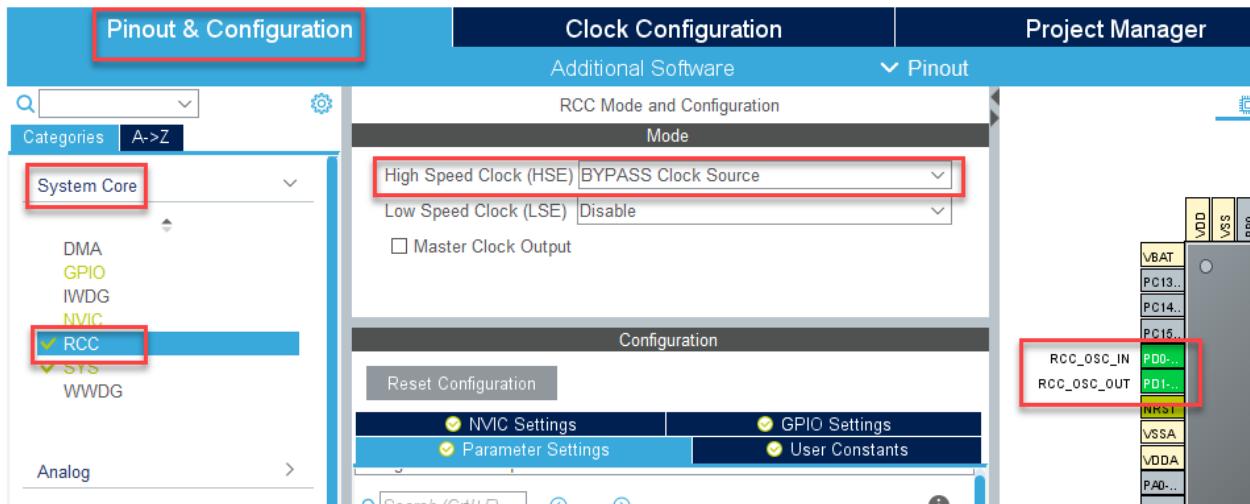
برای استفاده از اسیلاتور خارجی bypass و یا اسیلاتور داخلی با کریستال خارجی ، باید در نرم افزار cube ، از مسیر نشان داده شده در شکل زیر یکی از آنها را انتخاب کنیم.

The screenshot shows the RCC Mode and Configuration settings in the STM32CubeMX software:

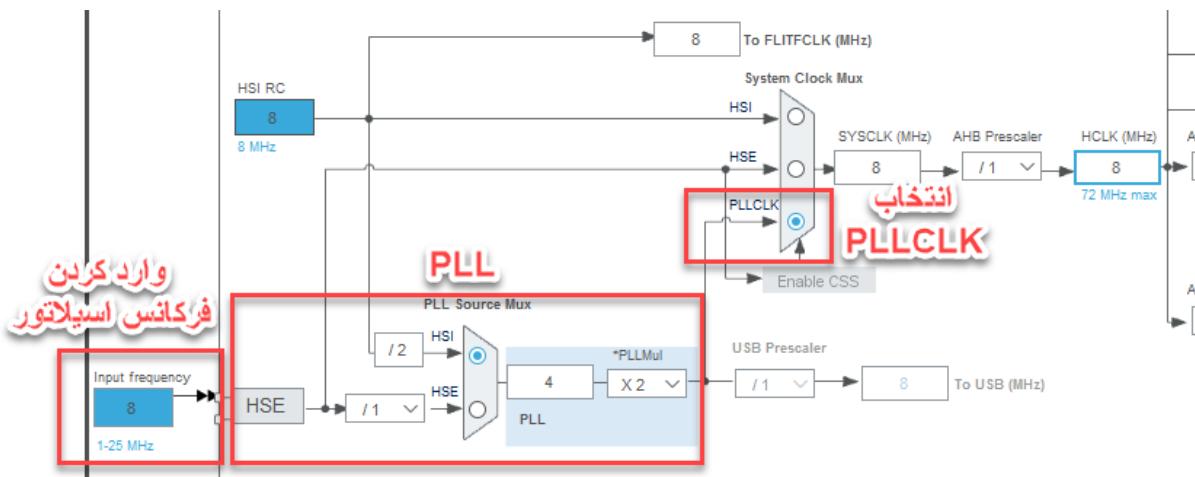
- Mode:**
 - High Speed Clock (HSE) is set to **Disable**.
 - Low Speed Clock (LSE) is set to **Disable**.
 - BYPASS Clock Source** is selected from the dropdown menu.
 - Master Clock Output is checked.
- Configuration:**
 - Reset Configuration button.
 - Parameter Settings, User Constants, and NVIC Settings tabs are visible at the bottom.

On the left sidebar, the categories are listed under **System Core** and **RCC**.

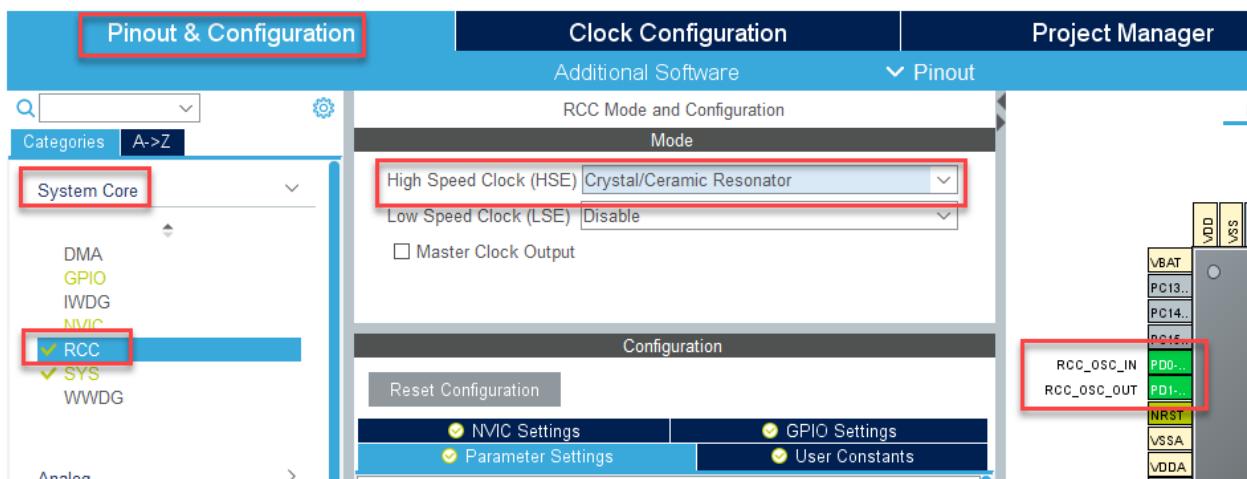
همانطور که در شکل زیر نشان داده شده است در صورت انتخاب **bypass** ، دو پایه از میکرو جهت اتصال اسیلاتور پیکربندی می شود. توجه داشته باشید که اسیلاتور خارجی باید به پایه **RCC_OSC_IN** متصل شود.



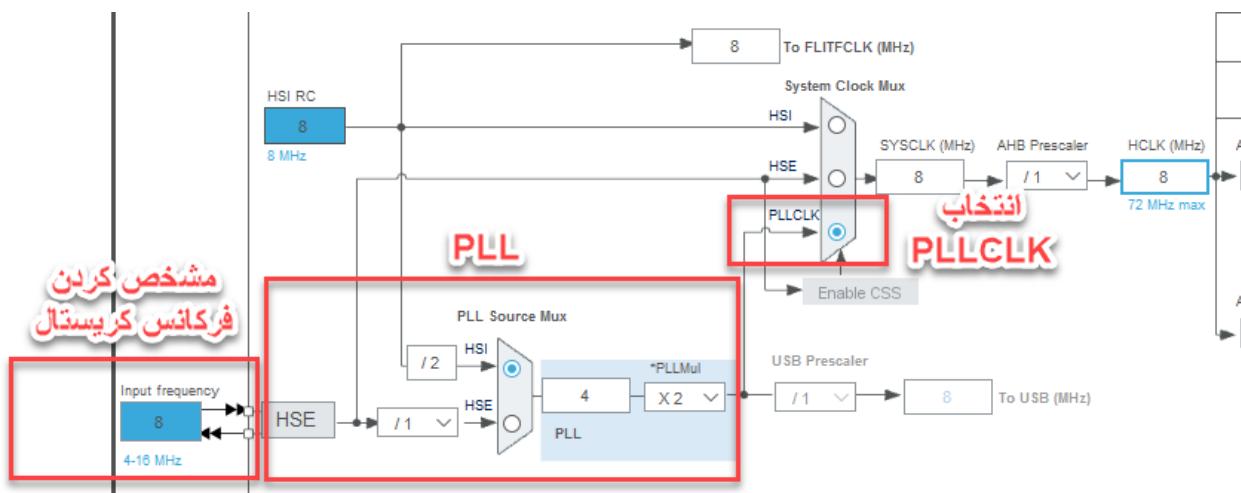
همچنین در صفحه **Clock Configuration** می توان بقیه تنظیمات را انجام داد.



برای استفاده از اسیلاتور داخلی با کریستال خارجی ، تنظیمات مانند شکل زیر انجام می شود.

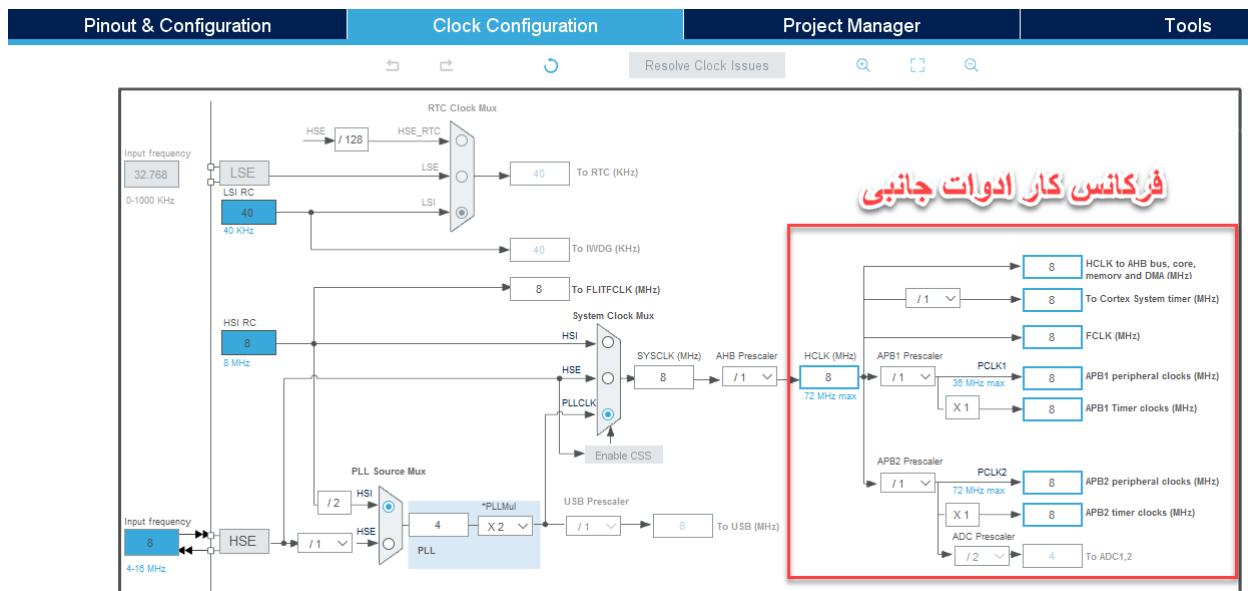


همچنین در صفحه Clock Configuration می‌توان بقیه تنظیمات را انجام داد.



فرکانس کار ادوات جانبی:

همانند شکل زیر در صفحه Clock Configuration می‌توان با تغیر در مقدار پارامترهای ضرب و تقسیم مسیر فرکانس گذرگاه APB1 و APB2، فرکانس ورودی به دستگاههای جانبی را تنظیم نمود.



برای اطلاع از این که هر یک از ادوات جانبی بر روی کدام گذرگاه قرار دارد باید در دیتا شیت میکرو رجیسترهای RCC_APB2ENR و RCC_APB1ENR را جستجو کنیم.

APB1 peripheral clock enable register (RCC_APB1ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	DAC EN	PWR EN	BKP EN	Res.	CAN EN	Res.	USB EN	I2C2 EN	I2C1 EN	UART5E N	UART4 EN	USART 3EN	USART 2EN	Res.	
Res.	rw	rw	rw	Res.	rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWD GEN	Reserved				TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN		
rw	rw	Res.	rw	Res.				rw	rw	rw	rw	rw	rw		

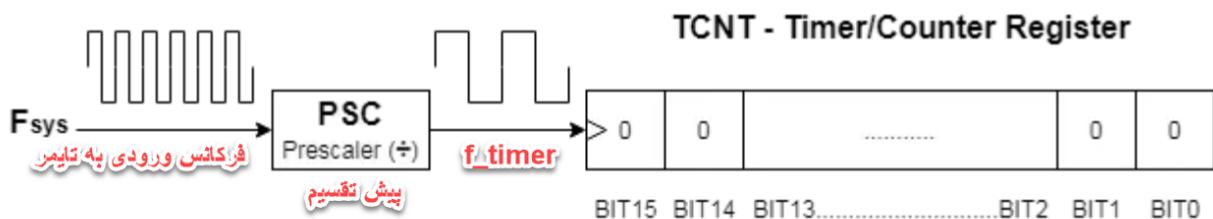
APB2 peripheral clock enable register (RCC_APB2ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USAR T1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN	
rw		rw	rw	rw	rw	rw		rw	rw	rw	rw	rw		rw	

تایمر : Timer

در بسیاری از پروژه‌ها ، لازم است که عملی در زمان یا بازه‌های زمانی مشخصی اجرا شود . در نتیجه نیاز به بخشی داریم که بتوان آن را طوری پیکربندی کرد تا در زمان‌های مورد نیاز با ایجاد اینترپرایت اجرای برنامه را به روتین سرویس هدایت و برنامه مورد نظر را اجرا نماید . در میکروهای ST تعدادی تایمر/کانتر وجود دارد.

در ساده‌ترین حالت می‌توان مدار تایمر/کانتر را به شکل زیر نمایش داد.

**پیش تقسیم (Prescaler)**

با توجه به این‌که تایمر انتخابی ما روی گذرگاه APB1 یا APB2 باشد می‌توان از فرکانس ورودی به تایمر مطلع شد. این پالس ورودی را می‌توان قبل از رسیدن به تایمر بر عددی که در رجیستر PSC قرار می‌گیرد تقسیم نمود. این رجیستر 16 بیتی می‌باشد در نتیجه عدد داخل آن می‌تواند بین 0 تا 65535 باشد. و فرکانس ورودی به تایمر f_{timer} را می‌توان از رابطه زیر محاسبه نمود.

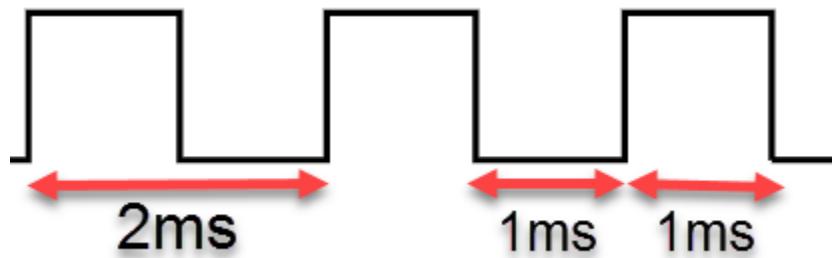
$$f_{\text{timer}} = \frac{f_{\text{sys}}}{(PSC + 1)}$$

: Counter Period

توسط این پارامتر می‌توان مشخص کرد که عدد داخل تایمر تا چه عددی افزایش و سپس صفر شود.

مثال: با این فرض که فرکانس ورودی به تایمر 8MHz باشد ، یکی از تایمرها را طوری تنظیم کنید که بتوان روی یکی از پایه‌های میکرو پالسی با فرکانس 500hz داشته باشیم . یعنی یک میلی ثانیه 1 و یک میلی ثانیه 0 باشد.

$$T = \frac{1}{f} = \frac{1}{500} = 2ms$$



در حالت تایмер، فرکانس یا زمان هر بار اجرای وقفه را می‌توان از روابط زیر بدست آورد.

$$f = \frac{f_{\text{timer}}}{(PSC + 1)(CP + 1)} \quad T = \frac{(PSC + 1)(CP + 1)}{f_{\text{timer}}}$$

با توجه به این‌که می‌خواهیم هر $1ms$ وقفه تایмер اجرا شود مقادیر زیر بدست می‌آید.

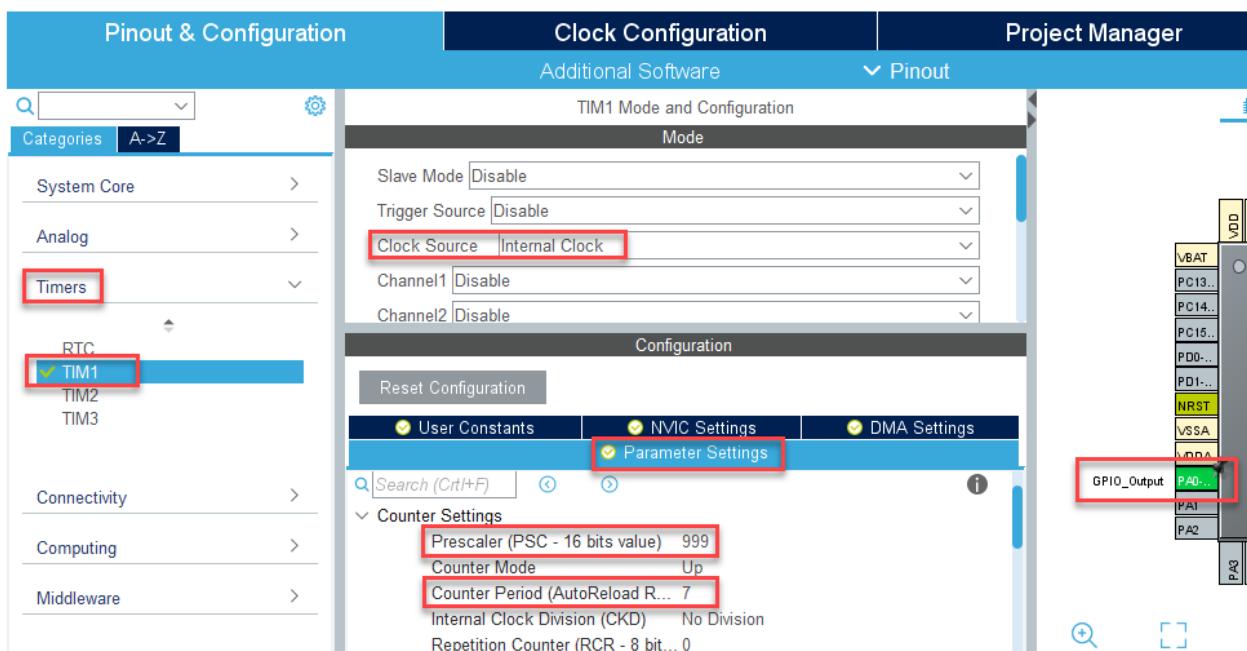
$$(PSC + 1)(CP + 1) = T * f_{\text{timer}}$$

$$(PSC + 1)(CP + 1) = 1ms * 8mhz$$

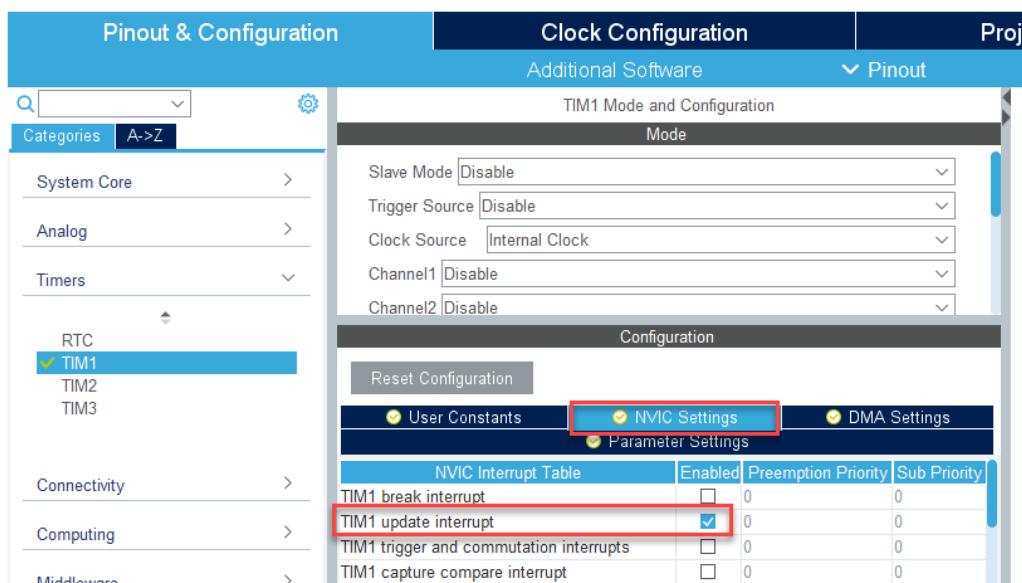
$$(PSC + 1)(CP + 1) = 8000$$

از آنجایی که یک معادله و دو مجھول داریم تعداد جواب‌های متعددی برای معادله بدست می‌آید لذا یکی از مجھولات را مفروض و دیگری را بدست می‌آوریم. برای مثال $PSC=999$ و $CP=7$ بدست می‌آید. البته بهتر است خروجی را با $PSC=7$ و $CP=999$ نیز بدست آورده با هم مقایسه کنید.

تنظیمات لازم در cube مانند شکل زیر است. همچنین پایه PA0 برای پالس خروجی در نظر گرفته شده است.



همچنین لازم است وقفه تایمر را نیز مانند شکل زیر فعال کنیم.



در فایل main باید توسط تابع زیر تایمر را راهاندازی کنیم.

```
/* USER CODE BEGIN 2 */

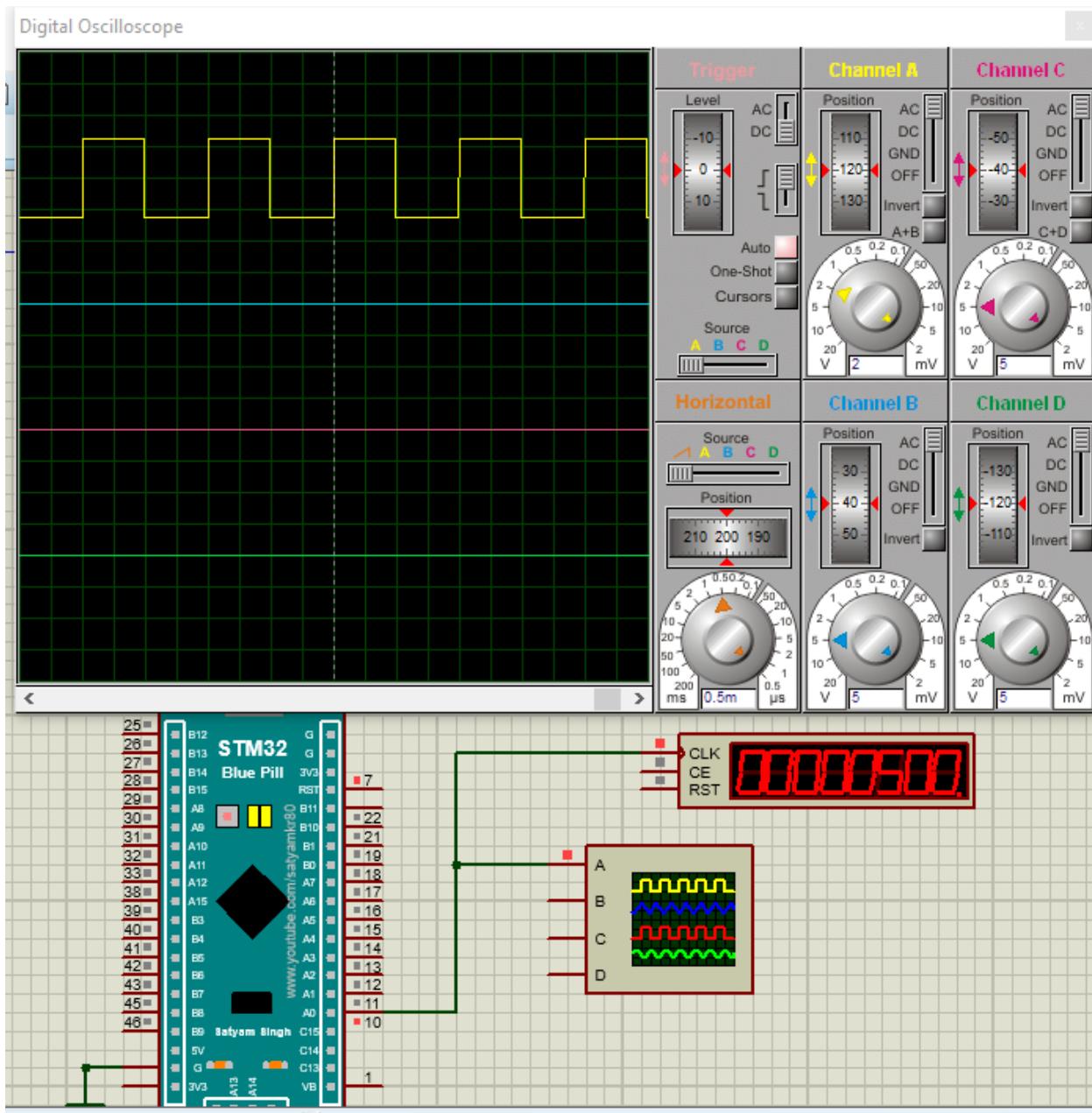
HAL_TIM_Base_Start_IT(&htim1);

/* USER CODE END 2 */
```

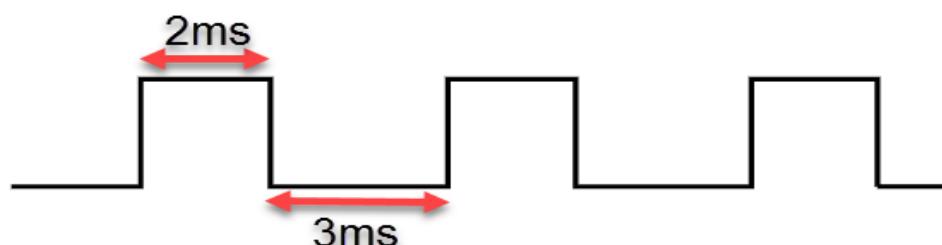
و در فایل `stm32f1xx_it.c` و در قسمت روتین سرویس تایمر برنامه مورد نظر خود را بنویسیم.

```
void TIM1_UP_IRQHandler(void)
{
    /* USER CODE BEGIN TIM1_UP_IRQn 0 */
    /* USER CODE END TIM1_UP_IRQn 0 */

    HAL_TIM_IRQHandler(&htim1);
    /* USER CODE BEGIN TIM1_UP_IRQn 1 */
    HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_0); برنامه کاربر
    /* USER CODE END TIM1_UP_IRQn 1 */
}
```



مثال: با توجه به مثال بالا با استفاده از تایمر، پالسی با مشخصات شکل زیر روی پایه PA0 ایجاد کنید.



در فایل `stm32f1xx_it.c` دو متغیر `p` و `n` را تعریف کرده ، و در تابع مربوط به وقفه تایمر ، روتنین سرویس را می نویسیم.

```
/* USER CODE BEGIN PV */

int p=0,n=0;

/* USER CODE END PV */

void TIM1_UP_IRQHandler(void)

{
    /* USER CODE BEGIN TIM1_UP_IRQn 0 */

    /* USER CODE END TIM1_UP_IRQn 0 */

    HAL_TIM_IRQHandler(&htim1);

    /* USER CODE BEGIN TIM1_UP_IRQn 1 */

    n++;

    if(p==0&&n==3)
    {
        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_0);

        p=1;

        n=0;
    }

    if(p==1&&n==2)
    {
        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_0);

        p=0;

        n=0;
    }
}
```

```

    }

/* USER CODE END TIM1_UP_IRQHandler */

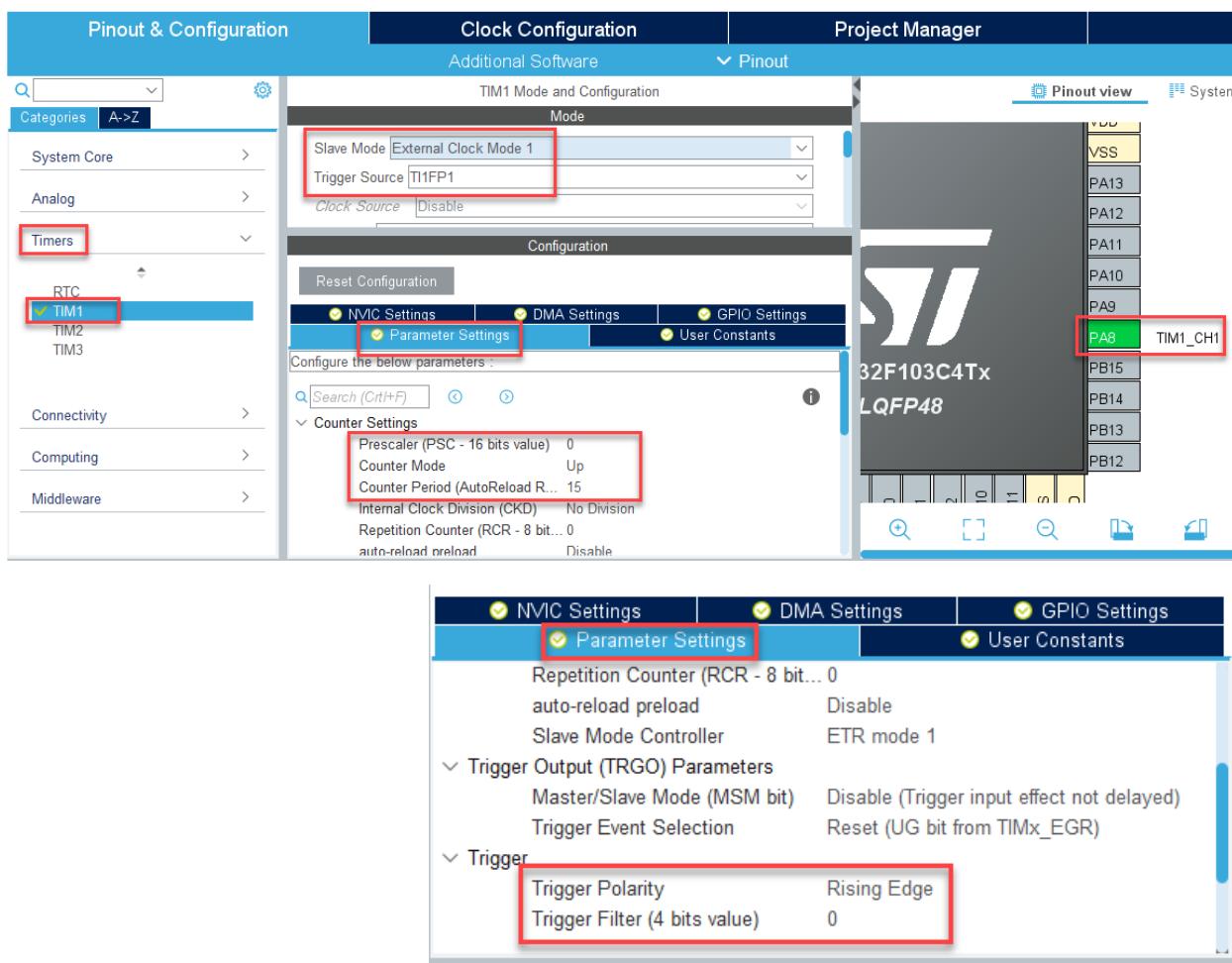
}

```

تمرین: یک LCD به میکرو متصل کرده و با استفاده از تایمر وقفه‌های یک ثانیه تولید و توسط آن یک ساعت همراه با آلارم بسازید.

: Counter

در مواردی که بخواهیم پالس تولید شده از مدار دیگری را شمارش کنیم، لازم است که تایمر را در حالت **counter** قرار داده و پالس مورد نظر را به عنوان ورودی به مدار تایمر/کنتر وارد کنیم. تنظیم‌های لازم در **cube** مانند شکل‌های زیر می‌باشد.



Slave Mode : در این قسمت با انتخاب گزینه External Clock Mode مشخص می‌کنیم که پالس ورودی به تایمر/کانتر از خارج میکرو وارد می‌شود.

Trigger Source : در این قسمت با انتخاب گزینه TI1FP1 مشخص می‌کنیم که پالس ورودی از طریق یکی از پایه‌های میکرو (PA8) به تایمر/کانتر وارد شود.

PSC: در این بخش ، عدد پیش تقسیم را مشخص می‌کنیم ، قرار دادن صفر در این قسمت یعنی فرکانس پالس ورودی به عدد یک تقسیم شود. به عبارت دیگر بدون تغیر در اختیار تایمر/کانتر قرار بگیرد.

Counter Mode : در این قسمت با انتخاب گزینه UP مشخص می‌کنیم که شمارنده صعودی باشد. گزینه‌های دیگری نیز مانند Down برای انتخاب وجود دارد.

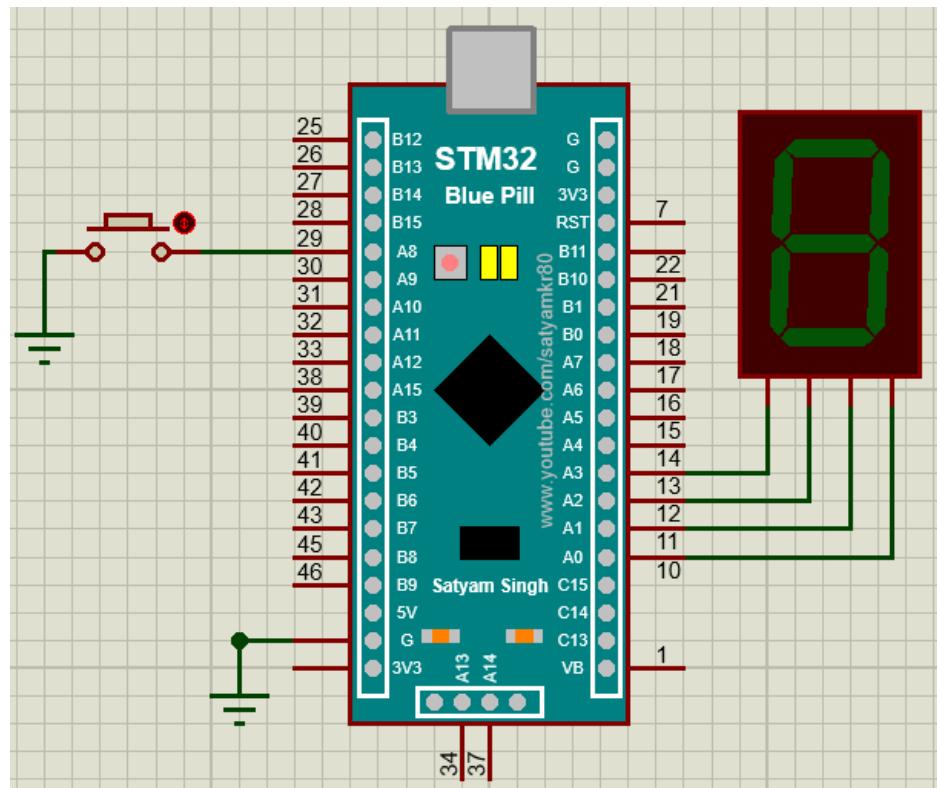
Counter Period : در این قسمت مشخص می‌کنم که شمارنده تا چه عددی افزایش یابد توجه داشته باشید که شمارنده روی عدد وارد شده در این قسمت صفر می‌شود. برای مثال اگر مقدار وارد شده ۱۵ باشد شمارنده تا عدد ۱۴ افزایش و با ورود پالس بعدی صفر می‌شود.

Trigger Polarity : در این قسمت مشخص می‌کنم که شمارنده در کدام لبه پالس ورودی عمل کند . گزینه‌های قابل انتخاب لبه بالارونده ، پایین رونده و هر دو لبه می‌باشد.

Rising Edge _ Falling Edge _ Both Edge

Trigger Filter : می‌توان برای جلوگیری از ورود نویزهای ناخواسته ، روی پالس ورودی فیلتری قرار داد ، قدرت این فیلتر را می‌توان با عددی بین ۰ تا ۱۵ مشخص کرد.

مثال: با توجه به شکل زیر تنظیم‌های لازم را در cube انجام داده ، سپس برنامه‌ای بنویسید که تعداد دفعاتی که کلید زده می‌شود توسط کانتر شمارش و روی 7segbcd نمایش داده شود. در ضمن کانتر به لبه پایین رونده پالس حساس باشد و روی عدد ۱۵ صفر شود .



علاوه بر تنظیم‌های بالا، پایه‌های PA0 تا PA3 را برای سون سگمنت خروجی می‌کنیم همچنین پایه PA8 را از داخل میکرو Pull Up می‌کنیم.

Pin	Signal	Type	GPIO	Max I	User	Modif.
PA8	TIM1...	n/a	Input	...	Pull-up	n/a

تابع راه اندازی تایمر/کانتر `HAL_TIM_Base_Start(&htim1);`

تابع مقدار دهی اولیه به تایمر/کانتر `(مقدار اولیه, __HAL_TIM_SET_COUNTER(&htim1, 0);)`

تابع خواندن عدد داخل تایمر/کانتر `= __HAL_TIM_GET_COUNTER(&htim1);`

با استفاده از توابع فوق برنامه مثال مطرح شده به صورت زیر خواهد بود.

```
/* USER CODE BEGIN PV */

int p;

/* USER CODE END PV */

/* USER CODE BEGIN 2 */

HAL_TIM_Base_Start(&htim1);

__HAL_TIM_SET_COUNTER(&htim1, 0);

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)

{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

    p = __HAL_TIM_GET_COUNTER(&htim1);

    HAL_GPIO_WritePin(GPIOA, 0xf, GPIO_PIN_RESET);

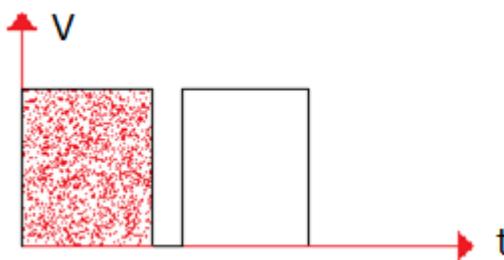
    HAL_GPIO_WritePin(GPIOA, p, GPIO_PIN_SET);

    HAL_Delay(100);

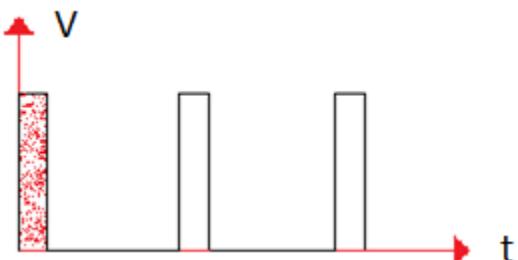
}

/* USER CODE END 3 */
```

یکی از راه‌های کنترل بعضی از دستگاه‌ها، کنترل انرژی است که به آن‌ها رسید. توسط تغییر در عرض پالس، می‌توانیم انرژی و در نتیجه، عملکرد آن دستگاه را کنترل کنیم. برای مثال اگر دو موج هم دامنه و هم فرکانس زیر را به دو موتور مشابه بدهیم:



سرعت بیشتر به دلیل انرژی بیشتر



سرعت کمتر به دلیل انرژی کمتر



مثال: با فرض این که فرکانس پالس ورودی به تایمر 8MHz باشد یک موج PWM با فرکانس 1kHz روی یکی از کانال‌های خروجی ایجاد کنید.

فرکانس موج PWM را می‌توان از رابطه زیر بدست آورد.

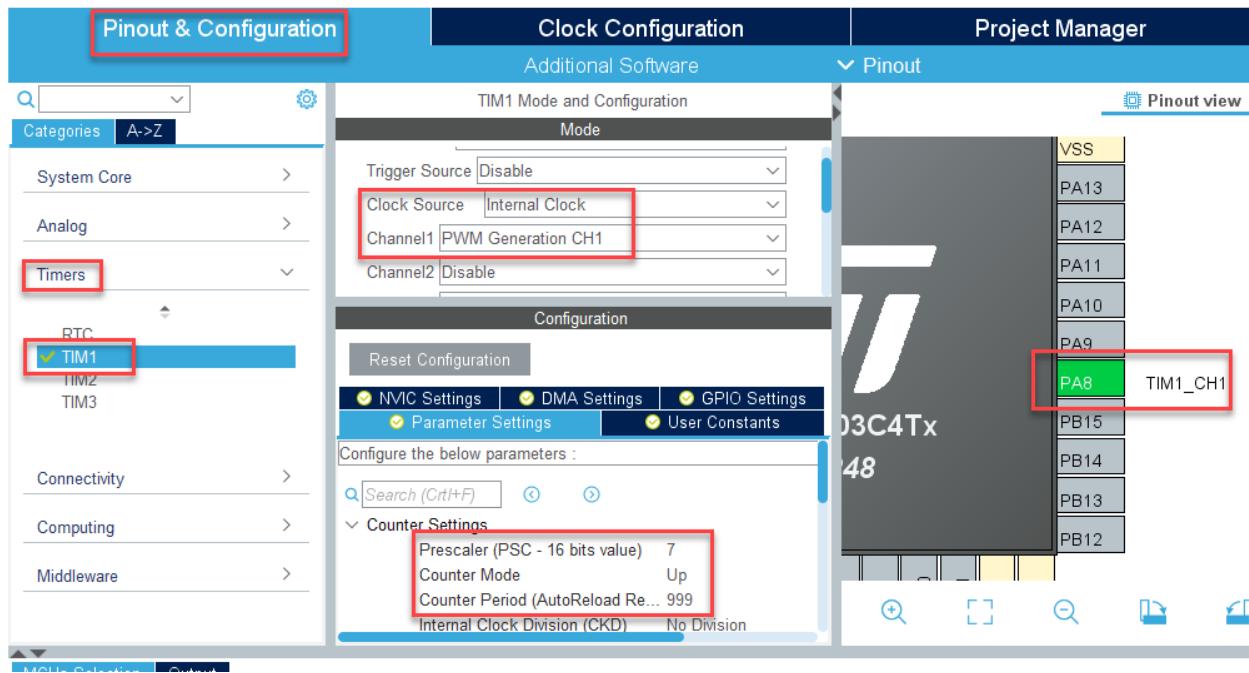
$$f_{pwm} = \frac{f_timer}{(PSC + 1)(CP + 1)}$$

با جاگذاری مقادیر خواهیم داشت:

$$(PSC + 1)(CP + 1) = 8000$$

جواب‌های متعددی برای PSC و CP بدست می‌آید، از آنجایی که تغییر در عرض پالس با تغییر در مقدار CP صورت می‌گیرد بهتر است مقدار PSC کوچک و CP بزرگتر انتخاب شود با این کار بازه بزرگتری برای تغییر خواهید داشت و تغییر عرض پالس با نرمی و دقیق‌تری انجام می‌شود. برای مثال یک انتخاب می‌تواند CP=999 و PSC=7 باشد.

برای راه اندازی PWM در cube يکی از تایمیرها را که کانال خروجی آن PWM داشته باشد را انتخاب و سپس تنظیم‌های شکل زیر را انجام می‌دهیم.



راه اندازی واحد PWM

مقدار دهی به CP (مقدار) _HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1);

با توجه به دو تابع بالا برنامه این مثال به صورت زیر است.

```
/* USER CODE BEGIN 2 */

HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_1);

__HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1,300);

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */
```

```
while (1)
```

```
{
```

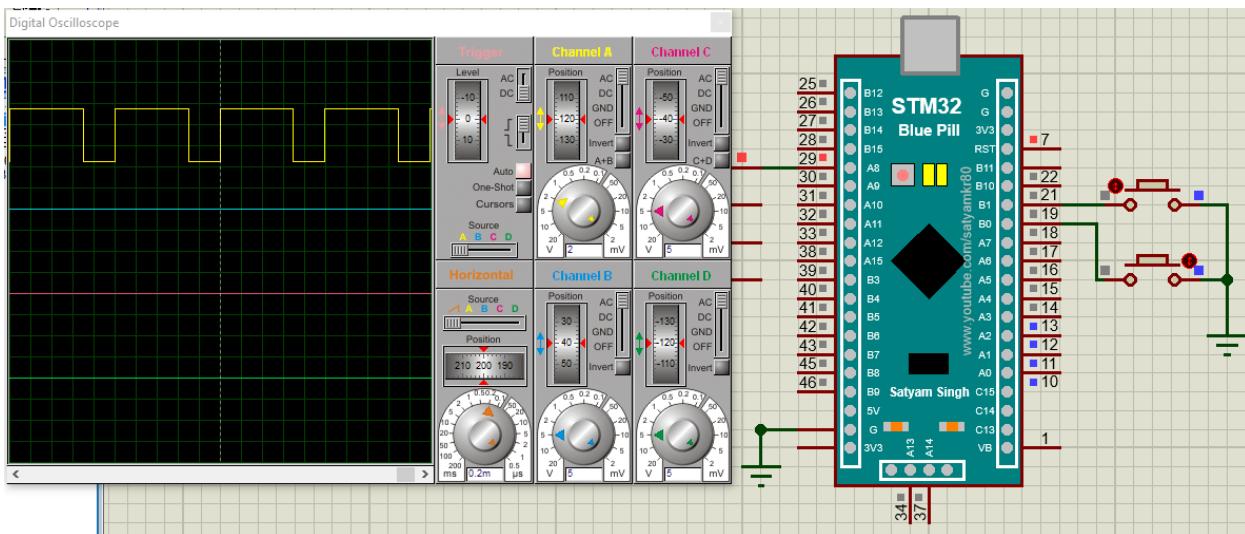
```
/* USER CODE END WHILE */
```

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

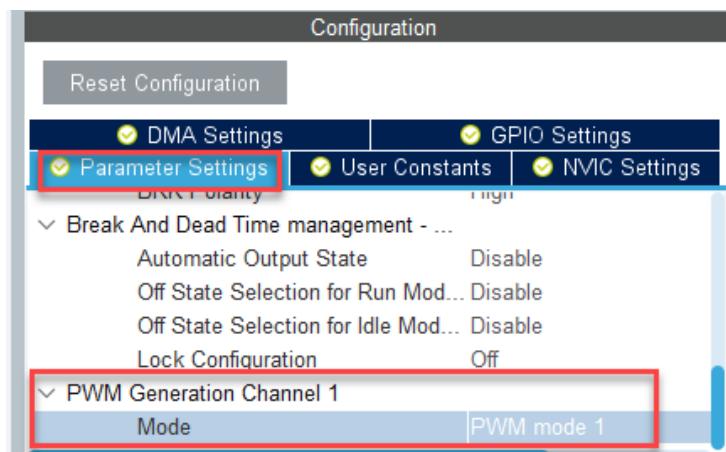
```
}
```

```
/* USER CODE END 3 */
```

و خروجی به شکل زیر است.



توجه داشته باشید که اگر در تنظیم‌ها PWM MODE 2 را انتخاب کنید خروجی NOT تصویر بالا خواهد شد.

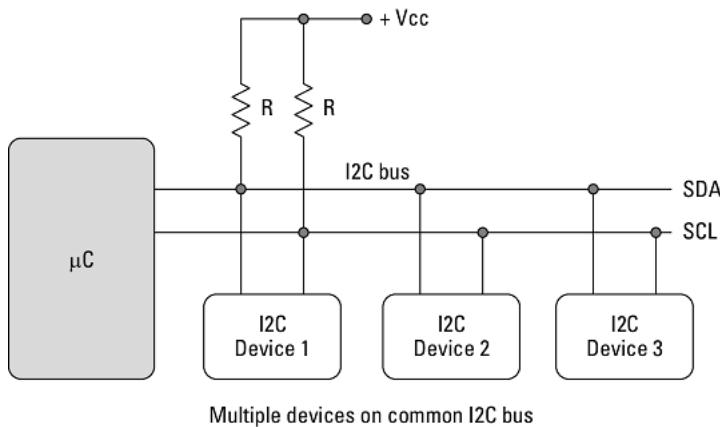


تمرین: برای مدار مثال قبل ، پس از انجام تنظیم‌های لازم در cube برنامه‌ای بنویسید که کاربر بتواند ، با استفاده از دو کلید متصل به PB0 و PB1 عرض پالس خروجی را تغیر دهد. (پایه‌های متصل به کلیدها را از داخل میکرو Pull Up کنید).

ارتباط سریال دو سیمه: I2C یا TWI

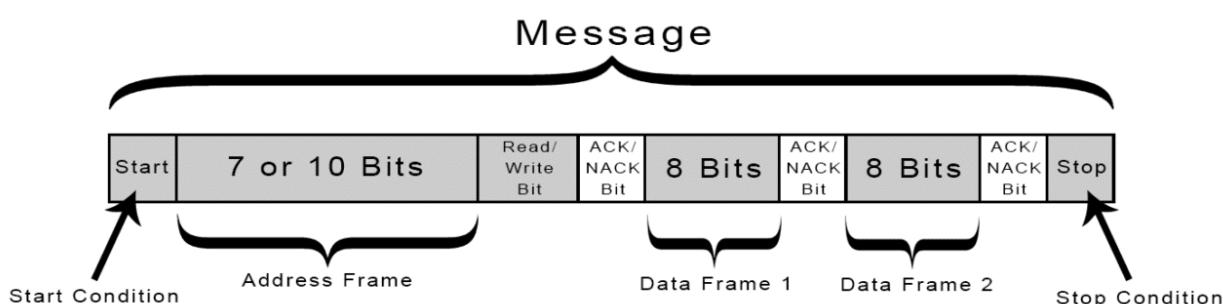
یک ارتباط سنکرون دو سیمه می‌باشد که با یک سیم SDA دیتا و با سیم دیگر SCL پالس همزمانی بین دستگاه‌های متصل منتقل می‌شود . توجه داشته باشید که سیم زمین نیز بین دستگاه‌ها متصل است.

The two I2C wires are called serial clock (SCL) and serial data (SDA).



در این نوع ارتباط هر یک از دستگاه‌ها (slave) دارای یک آدرس منحصر به فرد می‌باشند. هرگاه نیاز باشد از طریق میکرو (master) سیگنال‌های لازم همراه با آدرس قطعه ارسال و دیتا یا دستور جابجا می‌شود.

در این پروتکل ارتباطی ، داده‌ها در پیام‌ها منتقل می‌شوند. پیام‌ها به فریم‌هایی از داده‌ها تقسیم می‌شوند. هر پیام دارای یک قاب آدرس ، که حاوی آدرس باینری Slave و یک یا چند فریم داده حاوی داده‌های در حال انتقال است می‌باشد . این پیام همچنین شامل شرایط شروع و توقف، بیت‌های خواندن/نوشتن و بیت‌های ACK/NACK بین هر فریم داده است.



بیت صفرم آدرس ، می‌تواند صفر یا یک باشد. اگر بخواهیم از slave دیتایی را بخوانیم این بیت ۱ و اگر بخواهیم دیتایی را در slave بنویسیم این بیت صفر خواهد بود. در نتیجه هر slave دارای دو آدرس است ، یکی برای خواندن و یکی برای نوشتن. برای مثال آدرس‌های خواندن و نوشتمن در سه قطعه که با پروتکل I2C کار می‌کنند در زیر آمده است.

شماره قطعه	آدرس نوشتمن	آدرس خواندن
DS1307	0xd0	0xd1
LM75	0x90 (A0,A1,A2=0)	0x91 (A0,A1,A2=0)
AT24C32	0xa0 (A0,A1,A2=0)	0xa1 (A0,A1,A2=0)

سرعت انتقال دیتا: در اولین نسخه از این پروتکل سرعت انتقال دیتا 100 kbps تعیین شده بود ، در سال های بعد نسخه‌هایی با سرعت‌های بالاتر ، از این پروتکل معرفی شده است .

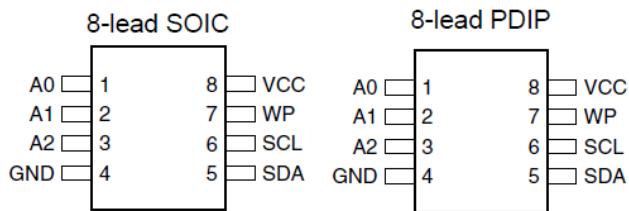
Maximum Speed	Standard mode= 100 kbps
	Fast mode= 400 kbps
	High speed mode= 3.4 Mbps
	Ultra fast mode= 5 Mbps

: AT24C64 حافظه

این IC یک حافظه EEPROM با ارتباط سریال I2C می‌باشد . ظرفیت این حافظه 64kbit می‌باشد و هر خانه حافظه یک بایت یعنی 8bit است در نتیجه ظرفیت این حافظه 8kbyte می‌باشد. پیکربندی و کاربرد پایه‌های این قطعه در شکل زیر نشان داده شده است.

Pin Configurations

Pin Name	Function
A0 - A2	Address Inputs
SDA	Serial Data
SCL	Serial Clock Input
WP	Write Protect



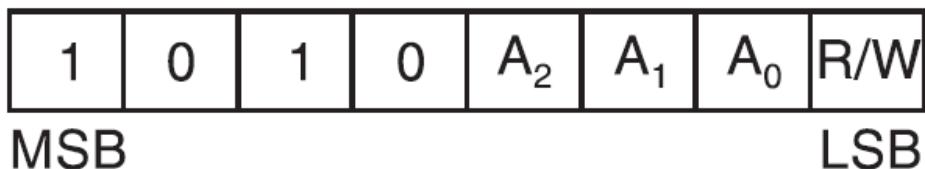
A0-A2 : اگر لازم باشد در مداری چند قطعه از این IC را استفاده کنیم ترکیب های مختلف این پایه ها باعث خواهد شد تا بتوان برای هر IC آدرس متفاوتی ایجاد کرده و تا ۸ قطعه را به کار ببریم.

I2C : پایه های ارتباط SDA-SCL

WP : یک کردن این پایه ، حافظه را در برابر نوشتہ شدن محافظت کرده و فقط قابل خواندن خواهد بود.

آدرس قطعه : Device Address

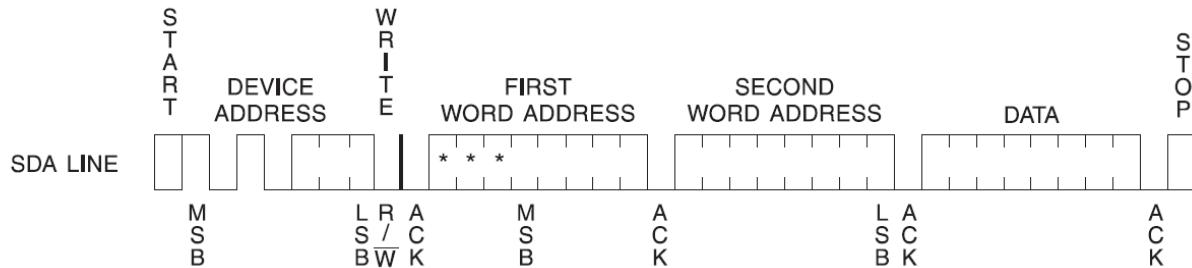
همانطور که گفته شد هر قطعه آدرس منحصر به فردی دارد که به آن آدرس قطعه گفته می شود و آدرس این قطعه به شکل زیر است:



با توجه به ترکیب های مختلف آدرس های 0xa0 تا 0xaf برای این قطعه قابل تعیین است. یادآوری می شود که هر قطعه دارای دو آدرس متفاوت برای نوشت و خواندن می باشد . ۰ بودن بیت صفرمین بايت ، آدرس نوشت و ۱ بودن این بیت ، آدرس خواندن را مشخص می کند ، برای مثال اگر هر سه پایه A2-A1-A0 را صفر کنیم ، آدرس 0xa0 برای نوشت و آدرس 0xa1 برای خواندن حافظه تعیین خواهد شد.

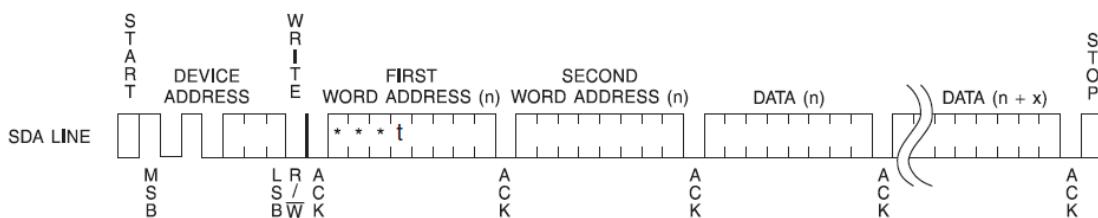
عملیات نوشت نیک بایت : برای نوشت نیک بایت در حافظه ، باید مطابق با شکل زیر عملیات نشان داده شده به ترتیب انجام شود.

Byte Write



و برای نوشتن تعدادی بایت پشت سر هم باید عملیات به شکل زیر انجام شود.

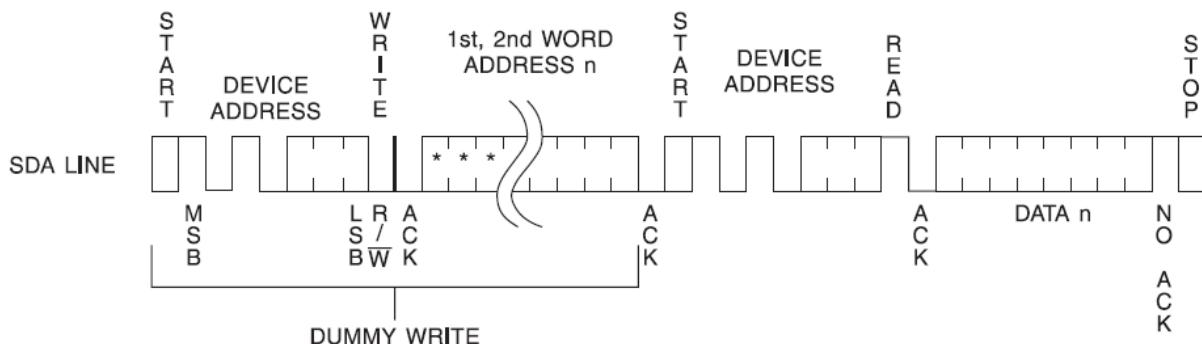
Page Write



- Note:
1. * = DON'T CARE bits
 2. t = DON'T CARE bit for AT24C32C
 3. acknowledge

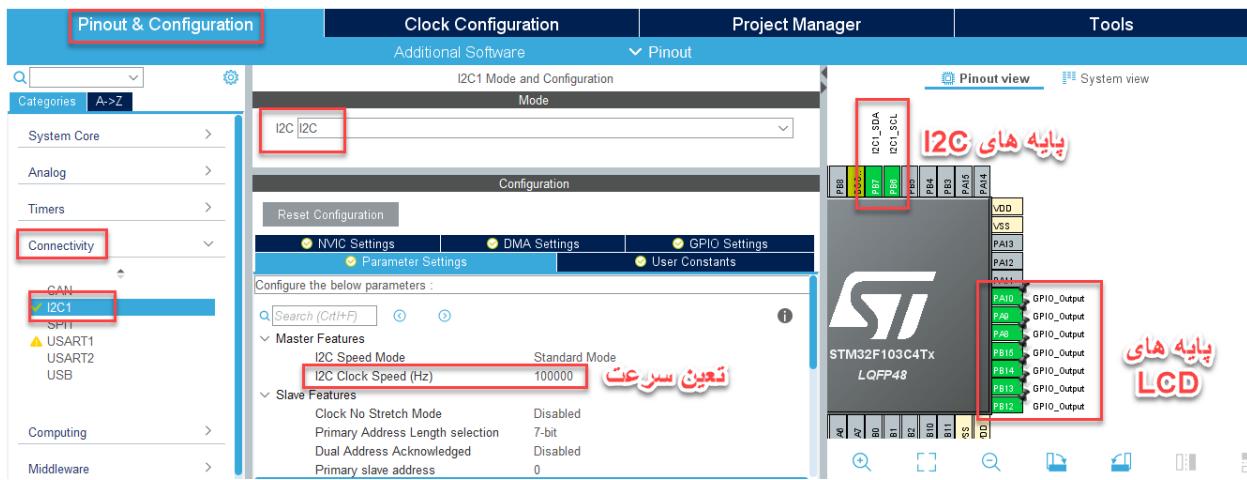
و برای خواندن یک بایت باید عملیات به شکل زیر انجام شود.

Random Read

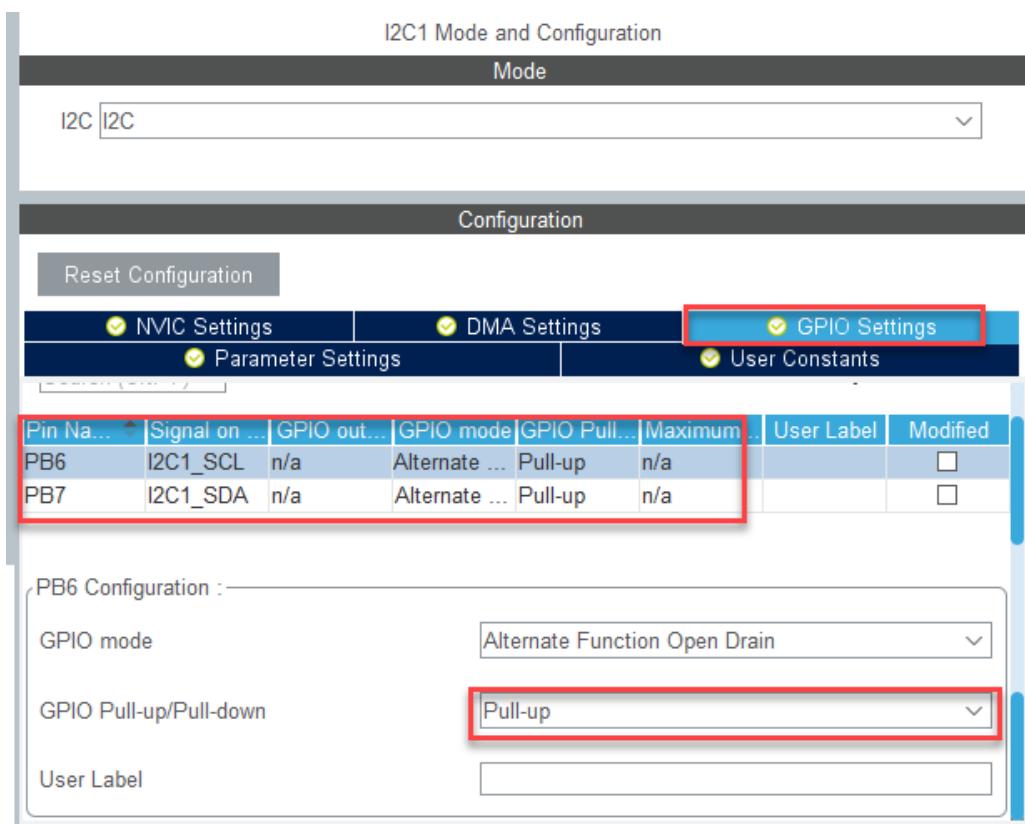


مثال : یک حافظه AT24C64 و یک LCD را به میکرو متصل کرده و برای مدار برنامه‌ای بنویسید که یک بار کلمه IRAN را در چهار خانه حافظه نوشته و سپس در یک حلقه ، محتوای حافظه را خوانده و به مدت ۲ ثانیه روی LCD نمایش دهید سپس پاک کرده و بعد از یک ثانیه دوباره این روند تکرار شود.

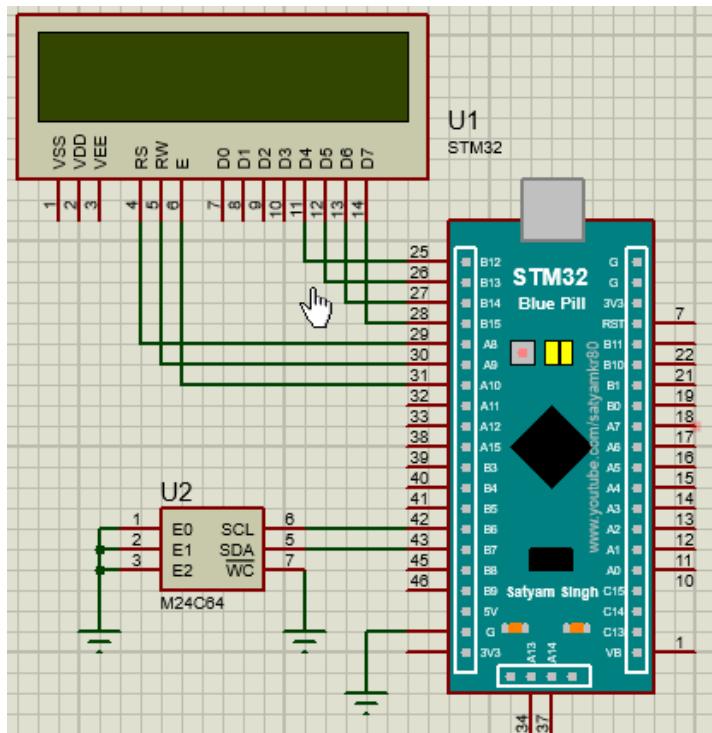
در نرم افزار cube مانند شکل زیر تنظیم های لازم را انجام می دهیم.



همان طور که در شکل مشخص است پایه های PB6 و PB7 به عنوان SCL و SDA پیکربندی شده اند. لازم است که این دو پایه، از خارج میکرو، یا داخل میکرو Pull Up شوند برای pull up داخلی مانند شکل زیر عمل کنند.



با توجه به تنظیمات cube سخت‌افزار مدار به شکل زیر است.



دو تابع پرکاربرد در ارتباط I2C به شرح زیر است :

```
HAL_I2C_Master_Transmit(&hi2c1,名 آرایه,آدرس قطعه برای نوشتن,time out);
```

```
HAL_I2C_Master_Receive(&hi2c1,名 آرایه,آدرس قطعه برای خواندن,time out) ;
```

توجه داشته باشید ، دیتایی که می خواهیم در قطعه نوشته شود را ، در یک آرایه قرار می دهیم و تعداد آن را در تابع بعد از نام آرایه مشخص می کنیم. اگر دیتای مورد نظر در یک متغیر غیر آرایه ذخیره شده باشد باید به جای نام آرایه ، آدرس متغیر (&var) را بنویسیم و تعداد بایت را 1 قرار دهیم. در هنگام خواندن دیتا نیز همین موارد قابل اجرا می باشد.

با توجه به توضیح بالا برنامه به شکل زیر است:

```
/* Private includes ----- */  
/* USER CODE BEGIN Includes */  
#include "stlcd.h"  
/* USER CODE END Includes */
```

```
/* USER CODE BEGIN PV */

unsigned char d[20],s;

/* USER CODE END PV */

/* USER CODE BEGIN 2 */

lcd_init();

lcd_clear();

HAL_Delay(20);

d[0]=0x00;

d[1]=0x00;

d[2]='I';

HAL_I2C_Master_Transmit(&hi2c1,0xa2,d,3,5);

HAL_Delay(20);

d[0]=0x00;

d[1]=0x01;

d[2]='R';

HAL_I2C_Master_Transmit(&hi2c1,0xa2,d,3,5);

HAL_Delay(20);

d[0]=0x00;

d[1]=0x02;

d[2]='A';

HAL_I2C_Master_Transmit(&hi2c1,0xa2,d,3,5);

HAL_Delay(20);

d[0]=0x00;
```

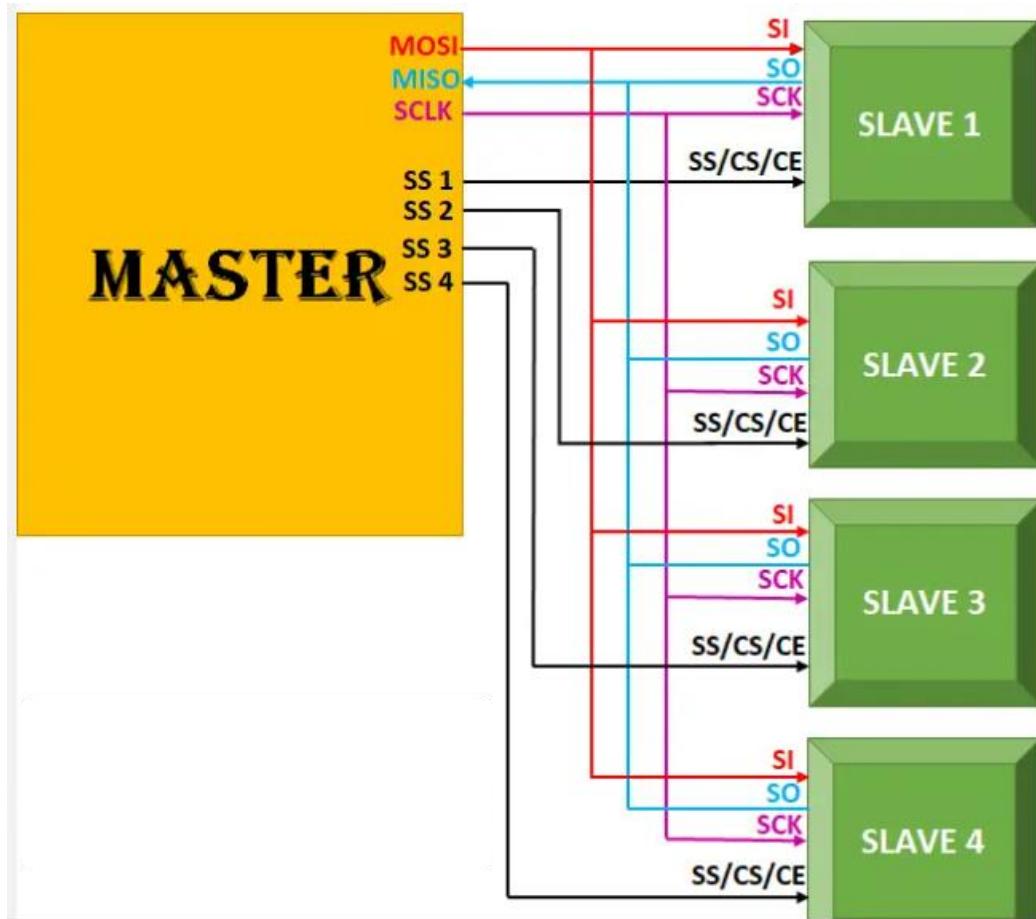
```
d[1]=0x03;  
d[2]='N';  
HAL_I2C_Master_Transmit(&hi2c1,0xa2,d,3,5);  
HAL_Delay(20);  
/* USER CODE END 2 */  
/* Infinite loop */  
/* USER CODE BEGIN WHILE */  
while (1)  
{  
/* USER CODE END WHILE */  
/* USER CODE BEGIN 3 */  
d[0]=0x00;  
d[1]=0x00;  
HAL_I2C_Master_Transmit(&hi2c1,0xa2,d,2,1);  
HAL_I2C_Master_Receive(&hi2c1,0xa3,&s,1,1);  
lcd_putchar(s);  
d[0]=0x00;  
d[1]=0x01;  
HAL_I2C_Master_Transmit(&hi2c1,0xa2,d,2,1);  
HAL_I2C_Master_Receive(&hi2c1,0xa3,&s,1,1);  
lcd_putchar(s);  
d[0]=0x00;  
d[1]=0x02;
```

```
HAL_I2C_Master_Transmit(&hi2c1,0xa2,d,2,1);
HAL_I2C_Master_Receive(&hi2c1,0xa3,&s,1,1);
lcd_putchar(s);
d[0]=0x00;
d[1]=0x03;
HAL_I2C_Master_Transmit(&hi2c1,0xa2,d,2,1);
HAL_I2C_Master_Receive(&hi2c1,0xa3,&s,1,1);
lcd_putchar(s);
HAL_Delay(2000);
lcd_clear();
HAL_Delay(1000);
}

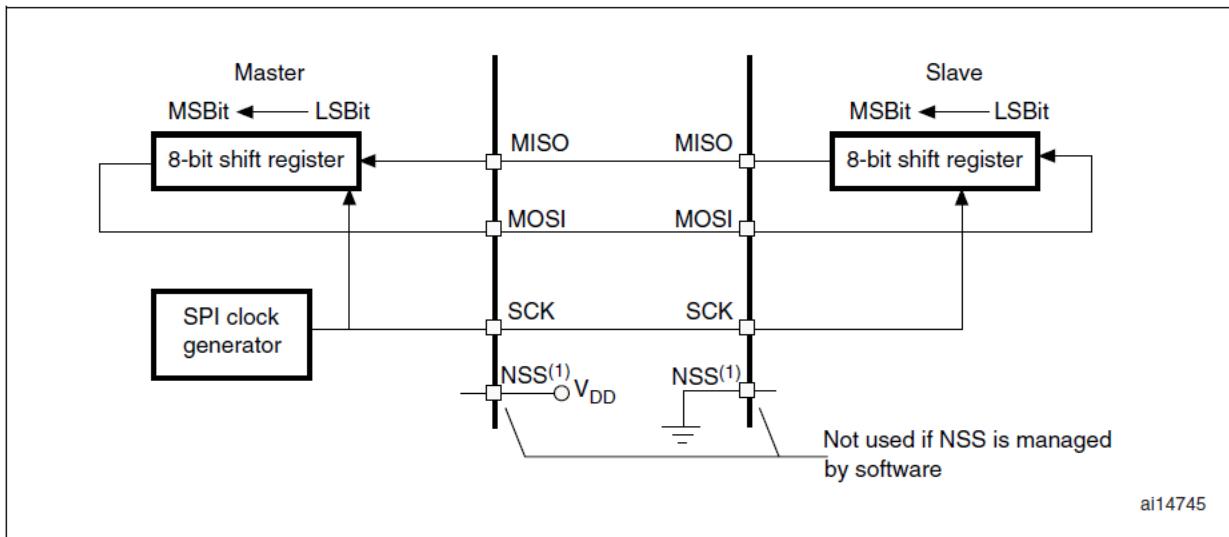
/* USER CODE END 3 */
```

ارتباط Serial Peripheral Interface : SPI

این ارتباط نیز از نوع سریال و سنکرون می‌باشد. در این پروتکل ارتباطی مانند شکل زیر یک Master و حداقل یک یا چند Slave داریم.



سیم بندی این ارتباط به صورت شکل زیر می‌باشد.



وظیفه هر یک از خطوط در این ارتباط به شرح زیر است :

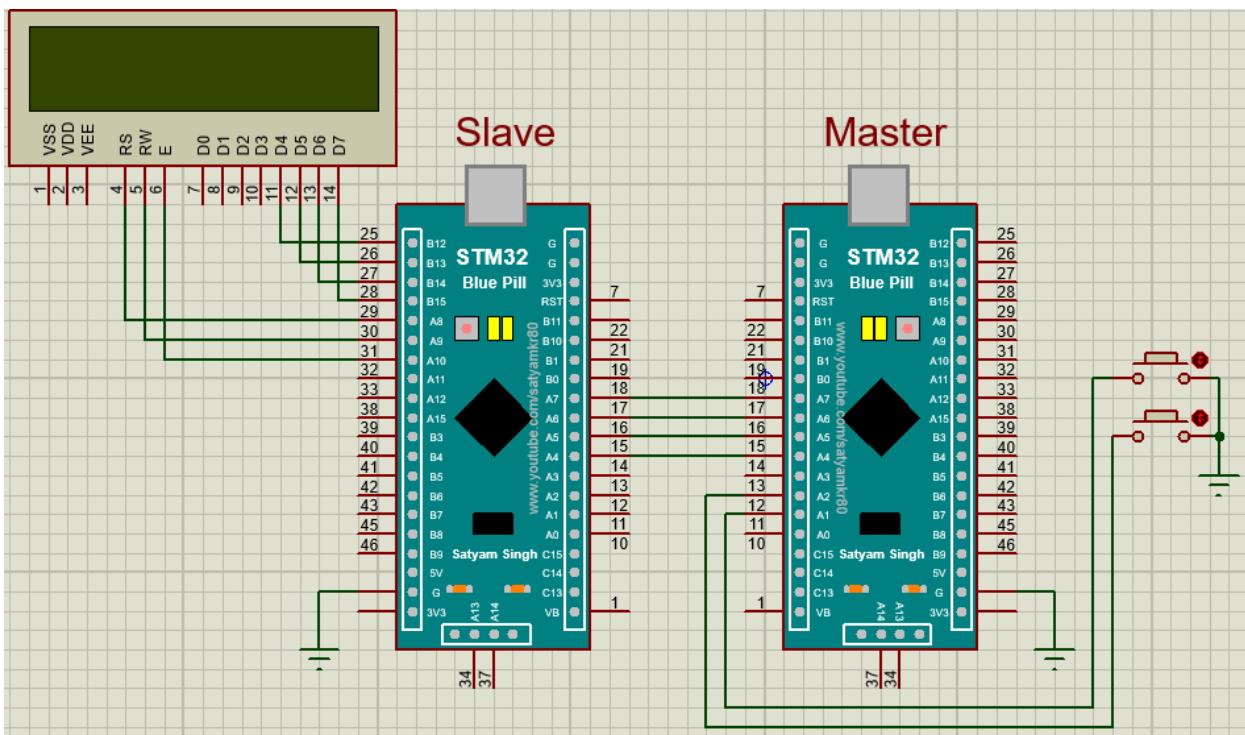
دیتای خارج شده از slave (Master In / Slave Out data) MISO توسط این پایه به Master وارد می شود.

دیتای خارج شده از Master (Master Out / Slave In data) MOSI توسط این پایه به slave وارد می شود.

در این ارتباط ، به ازای هر پالس یک بیت دیتا بین Master و Slave جابه جا می شود . وظیفه تولید این پالس بر عهده Master می باشد ، پالس تولیدی هم در Master استفاده می شود و هم از طریق پایه SCK به slave منتقل می شود تا به طور همزمان یک بیت جابه جا شود.

برای برقراری ارتباط ابتدا در Master NSS (Slave select) پایه ای را که به Slave متصل است صفر می کنیم با این کار slave را انتخاب یا به عبارت دیگر فعال نموده و سپس اقدام به انتقال دیتا می کنیم .

مثال: مانند شکل زیر دو میکرو را توسط SPI به یکدیگر متصل کنید. روی Master دو عدد کلید و روی Slave یک LCD قرار دهید. برنامه‌ای بنویسید که در Master با زدن کلیدها ، عدد داخل یک متغیر، افزایش یا کاهش یابد سپس از طریق LCD به slave ارسال و بر روی LCD نمایش داده شود.



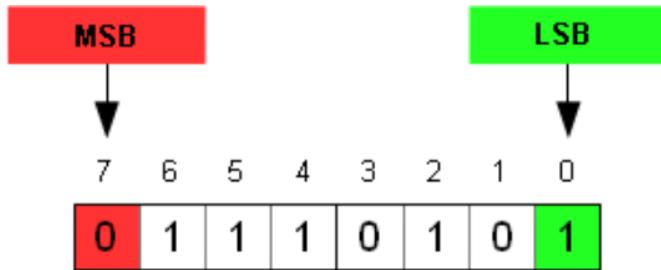
تنظیم‌های Cube برای Master مطابق شکل زیر انجام می‌شود.

The screenshot shows the STM32CubeMX software interface with the 'Pinout & Configuration' tab selected. The SPI1 mode is set to 'Full-Duplex Master'. The 'Parameter Settings' section shows Data Size as 8 Bits and Prescaler (for Baud Rate) as 16. A callout highlights the connection between PA1 and PA2 labeled 'اتصال به کلید با pull up داخلی' (Connection to internal pull-up on the key). The pinout view shows the STM32F103C4T6 LQFP48 package with PA1 and PA2 highlighted.

مشخص می‌کند که میکرو در حالت Master و یک ارتباط دو طرف کامل قرار بگیرد.

مشخص می‌کند در هر انتقال چند بیت دیتا جابه‌جا شود. 8 بیت یا 16 بیت.

. مشخص می‌کند انتقال دیتا از بیت کم ارزش‌تر LSB شروع شود یا پر ارزش‌تر MSB . First Bit



: پیش تقسیم فرکانس ورودی به SPI را مشخص می‌کند که نتیجه آن فرکانس SCK ، یا به عبارت دیگر سرعت انتقال دیتا است، که در بخش بعدی Baud Rate نمایش داده می‌شود.

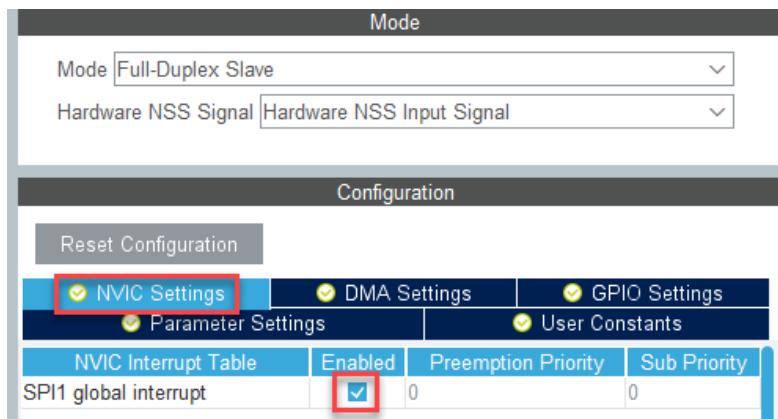
: مشخص می‌کند هنگام بی‌کاری SPI پایه SCK در منطق صفر LOW و یا یک High Clock Polarity باشد.

: مشخص می‌کند انتقال دیتا در اولین لبه (1 Edge) یا دومین لبه (2 Edge) پالس SCK منتقل شود.

پایه‌های SPI : با فعال کردن SPI سه پایه برای MOSI-MISO-SCK تعیین می‌شود. یک پایه را نیز خروجی می‌کنیم که به پایه NSS در Slave متصل خواهد شد. هنگام نوشتن برنامه قبل از ارسال دیتا این پایه را صفر می‌کنیم و بعد از دستور ارسال ، آن را غیر فعال یعنی یک می‌کنیم.

تنظیم‌های Cube : برای Slave مطابق شکل زیر انجام می‌شود.

در قسمت Hardware NSS Input گزینه NSS گزینه Full Duplex Slave Mode در بخش گزینه Master تنظیم می کنیم. همچنین مانند شکل زیر وقفه Signal را انتخاب می کنیم. بقیه موارد را مانند SPI را فعال می کنیم.



توابع ارسال و دریافت توسط SPI :

تابع ارسال : HAL_SPI_Transmit(&hspi1, تعداد دیتای ارسالی, نام متغیر رشته‌ای یا آدرس متغیر, Timeout);
تابع دریافت بر پایه وقفه : HAL_SPI_Receive_IT(&hspi1, تعداد دیتای ارسالی, نام متغیر رشته‌ای یا آدرس متغیر, Timeout);

متغیر hspi1.RxXferCount : تعداد دیتای مشخص شده در توابع بالا در این متغیر ذخیره می‌شود و با دریافت هر دیتا یک واحد از آن کم می‌شود در نتیجه هرگاه این متغیر صفر شود یعنی دیتا بطور کامل دریافت شده است. توجه داشته باشید لازم است برای دریافت بعدی دوباره این متغیر با فراخوانی تابع دریافت ، مجدد مقدار دهی شود.

با استفاده از توابع و متغیر گفته شده ، برنامه Master و Slave به صورت زیر خواهد شد.

برنامه : Master

```
/* USER CODE BEGIN PV */

signed char i=0;

/* USER CODE END PV */

/* USER CODE BEGIN WHILE */
```

```
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_1)==0)
    {
        i++;
        if(i>9) i=0;
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_4,GPIO_PIN_RESET);
        HAL_SPI_Transmit(&hspi1,&i,1,500);
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_4,GPIO_PIN_SET);
        HAL_Delay(200);
    }

    if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_2)==0)
    {
        i--;
        if(i<0) i=9;
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_4,GPIO_PIN_RESET);
        HAL_SPI_Transmit(&hspi1,&i,1,100);
        HAL_GPIO_WritePin(GPIOA,GPIO_PIN_4,GPIO_PIN_SET);
        HAL_Delay(200);
    }

    /* USER CODE END 3 */
}
```

برنامه : Slave

```
/* USER CODE BEGIN Includes */

#include "stlcd.h"

#include "stdio.h"

/* USER CODE END Includes */

/* USER CODE BEGIN PV */

unsigned char d[10];

char s[16];

/* USER CODE END PV */

/* USER CODE BEGIN 2 */

lcd_init();

HAL_SPI_Receive_IT(&hspi1,d,1);

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)

{

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

if(hspi1.RxXferCount==0)

{

//HAL_GPIO_WritePin(GPIOB,0xf<<12,GPIO_PIN_RESET);
```

```

//HAL_GPIO_WritePin(GPIOB,d[0]<<12,GPIO_PIN_SET);

sprintf(s,"%02d",d[0]);

lcd_gotoxy(7,0);

lcd_puts(s);

HAL_SPI_Receive_IT(&hspi1,d,1);

}

}

/* USER CODE END 3 */

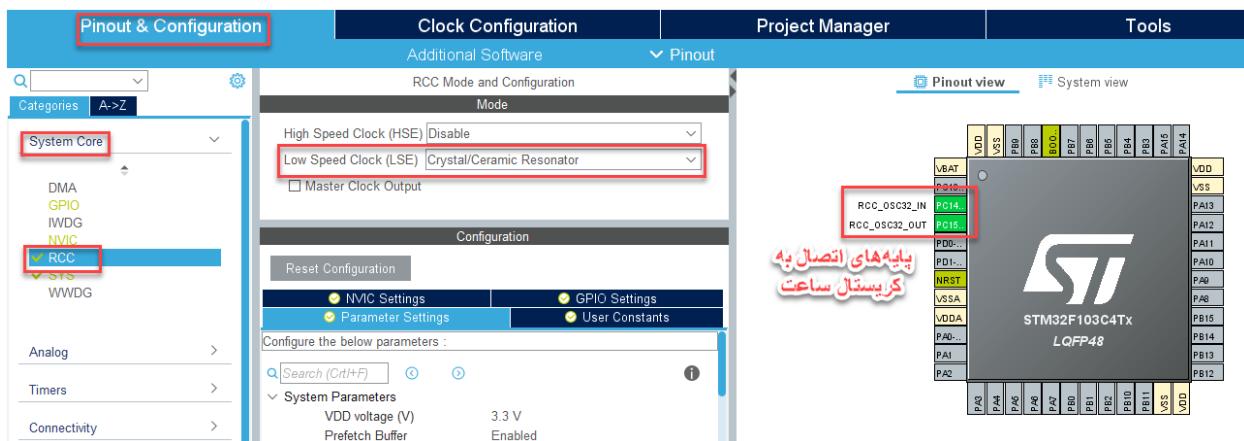
```

ساعت و تقویم با RTC : (Real Time Counter)

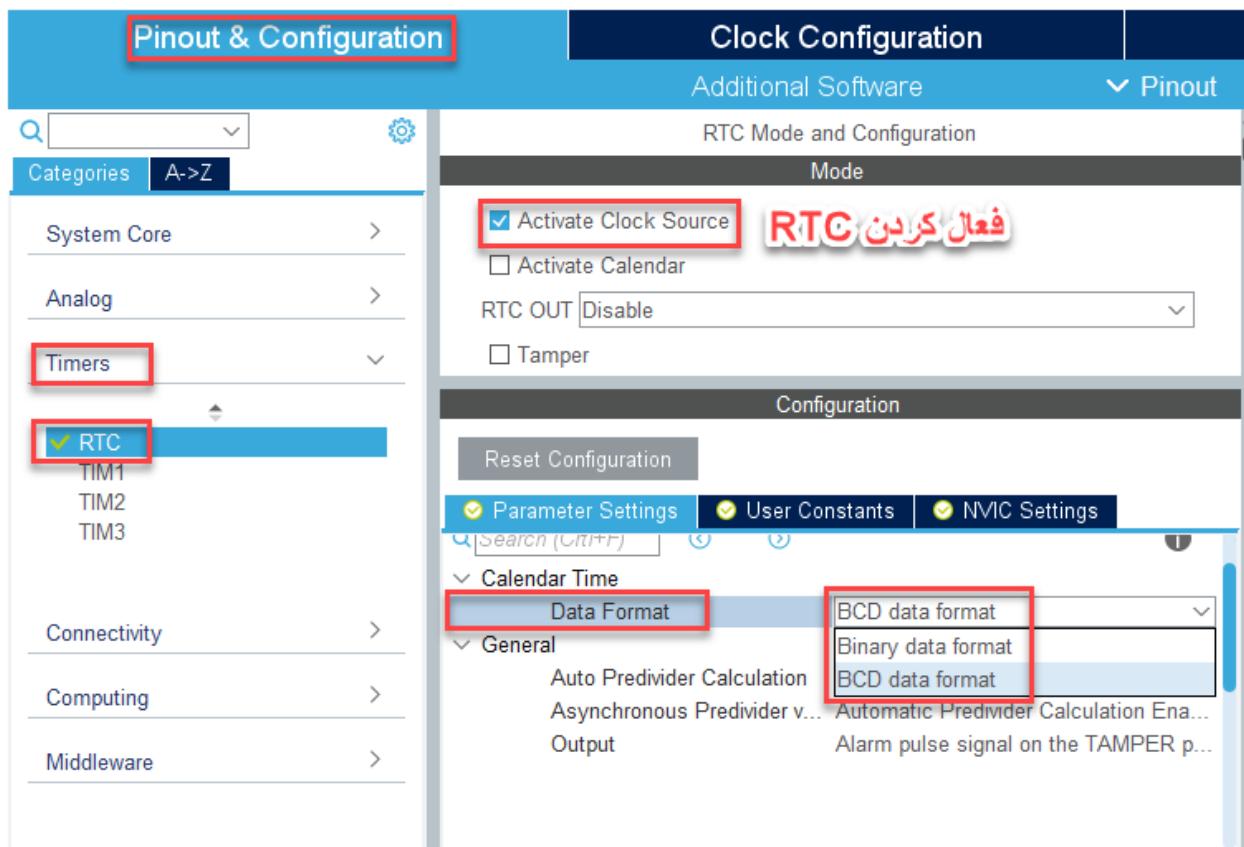
برای زمان سنجی واقعی لازم است که بتوانیم پالس یک ثانیه و به تبع آن دقیقه و ساعت را تولید کنیم. در این میکرو با راه اندازی واحد RTC می‌توان از یک ساعت و تقویم موجود بهره مند شد.

: cube تنظیم‌های

برای راه اندازی این واحد ابتدا لازم است مانند شکل زیر از مسیر مشخص شده، اسیلاتور Low Speed (LSE) را فعال کنیم با این کار دو پایه از میکرو جهت اتصال به کریستال با فرکانس 32768hz پیکربندی می‌شود.

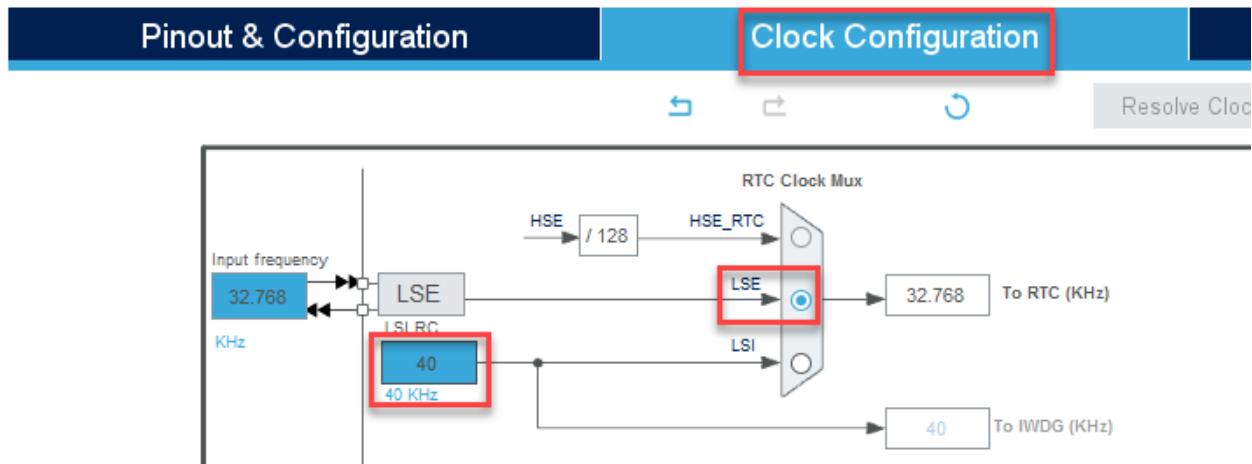


در مرحله بعد باید مانند شکل زیر RTC را فعال کنیم.

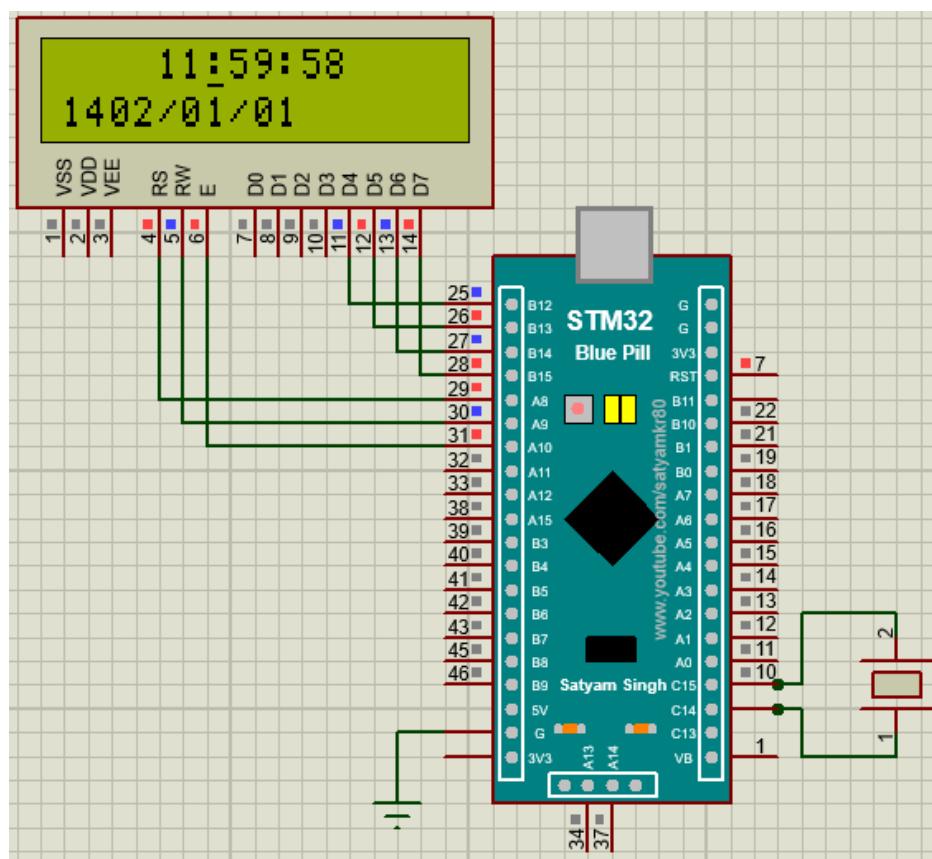


در بخش Data Format می‌توان مشخص کرد که اعداد مربوط به ساعت و تاریخ در قالب اعداد باینری در اختیار قرار گیرند یا BCD.

در مرحله بعد مانند شکل زیر به صفحه Clock Configuration وارد و از طریق فرکانس ورودی به RTC را به خروجی اسیلانتور 32768hz متصل می‌کنیم. توجه داشته باشید که می‌توان برای راهاندازی RTC از فرکانس داخلی 40khz وجود نیز استفاده کرد.



مثال: برای مدار شکل زیر با راه اندازی RTC روی خط اول LCD ساعت و روی خط دوم تاریخ را نمایش دهید.(توجه داشته باشید، در پروتئوس برای این برد ، امکان اتصال کریستال وجود ندارد و به طور پیش فرض در شبیه سازی تعریف شده است . (در شکل زیر کریستال نمایشی است .)



توابع کار با RTC :

ابتدا لازم است دو متغیر از نوع `date` و `time` تعریف شود.

RTC_TimeTypeDef نام متغیر ;

RTC_DateTypeDef نام متغیر ;

متغیرهای تعریف شده از نوع استراکت هستند و اجزای آن‌ها عبارتند از:

```
time.Hours=11;
```

time.Minutes=59;

```
time.Seconds=56;
```

```
date.Year=2;
date.Month=1;
date.Date=10;
date.WeekDay=7;
```

می‌توان متغیرهای زمان و تاریخ را مقدار دهی ، و توسط تابع زیر به RTC اعمال کرد.

```
HAL_RTC_SetTime(&hrtc,&نام متغیر,RTC_FORMAT_BIN);
```

```
HAL_RTC_SetDate(&hrtc,&نام متغیر,RTC_FORMAT_BIN);
```

دقت شود که در تابع بالا علامت & آدرس متغیر را به تابع ارسال می‌کند.

و برای خواندن زمان و تاریخ از تابع زیر استفاده می‌شود .

```
HAL_RTC_GetTime(&hrtc,&نام متغیر,RTC_FORMAT_BIN);
```

```
HAL_RTC_GetDate(&hrtc,&نام متغیر,RTC_FORMAT_BIN);
```

با توجه به تابع بالا برنامه مدار به صورت زیر خواهد شد.

```
/* USER CODE BEGIN Includes */
#include "stlcd.h"
#include "stdio.h"

/* USER CODE END Includes */
/* USER CODE BEGIN PV */

RTC_TimeTypeDef time;
RTC_DateTypeDef date;
char d[16];

/* USER CODE END PV */
/* USER CODE BEGIN 2 */
```

```
lcd_init();

time.Hours=11;
time.Minutes=59;
time.Seconds=58;

HAL_RTC_SetTime(&hrtc,&time,RTC_FORMAT_BIN);

date.Year=2;
date.Month=1;
date.Date=1;
date.WeekDay=7;

HAL_RTC_SetDate(&hrtc,&date,RTC_FORMAT_BIN);

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)

{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

        HAL_RTC_GetTime(&hrtc,&time,RTC_FORMAT_BIN);

        sprintf(d,"%02d:%02d:%02d",time.Hours,time.Minutes,time.Seconds);

        lcd_gotoxy(4,0);

        lcd_puts(d);

        HAL_RTC_GetDate(&hrtc,&date,RTC_FORMAT_BIN);

        sprintf(d,"14%02d/%02d/%02d",date.Year,date.Month,date.Date);
}
```

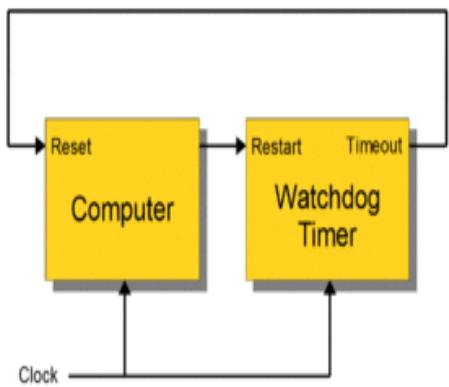
```

lcd_gotoxy(0,1);
lcd_puts(d);
}

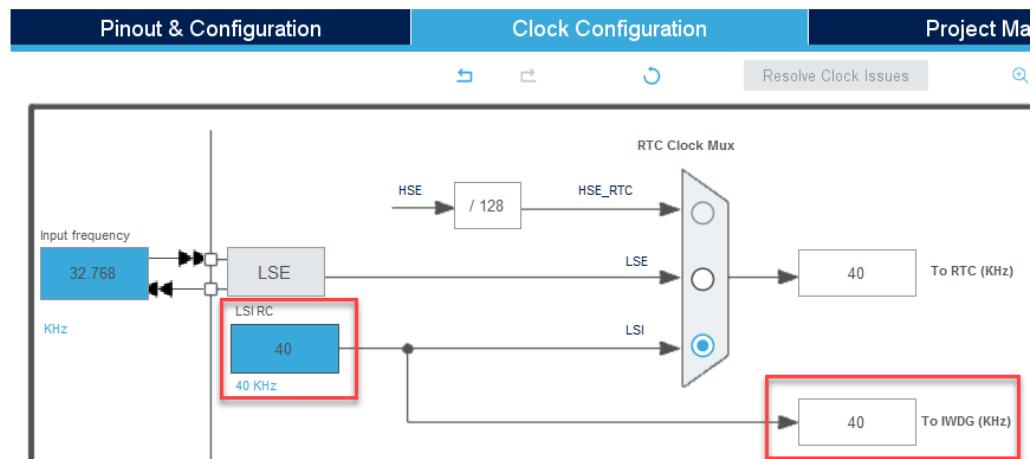
/* USER CODE END 3 */

```

هنگ هرگاه سیستمی هنگ کند برای راه اندازی مجدد باید آن را ریست کنیم. یکی از عوامل هنگ کردن ، مشکل در نرم افزار سیستم می باشد . یکی از واحدهای پیش‌بینی شده در این میکرو (IWDG) independent watchdog می باشد. این بخش یک تایمر قابل تنظیم است ، در برنامه زمان مورد نظر برای آن تنظیم می شود ، اگر به هر علتی میکرو هنگ کند و نتواند WD را ریست کند بعد از زمان تنظیم شده WD میکرو را ریست و سیستم را از هنگ خارج می کند.



پالس ورودی به تایمر WD ، از یک اسیلاتور داخلی 40kHz تامین می شود.



: IWDG پارامترهای تنظیم

IWDG Mode and Configuration
<input checked="" type="checkbox"/> Activated

Configuration
Reset Configuration
<input checked="" type="checkbox"/> Parameter Settings <input checked="" type="checkbox"/> User Constants
Configure the below parameters :
Clocking
IWDG counter clock prescaler : 32
IWDG down-counter reload value : 4000

از مسیر نشان داده شده در شکل بالا وارد صفحه IWDG شده و با زدن تیک آن را فعال می‌کنیم. در بخش **reload** پیش تقسیم Prescaler می‌توان از بین اعداد مشخص شده عددی را انتخاب کرد. در بخش **value** می‌توان عددی بین صفر تا 4095 را وارد کرد. با توجه به اعداد وارد شده در دو بخش قبلی زمان تایмер از رابطه زیر محاسبه می‌شود. عدد بدست آمده بر حسب ثانیه است.

$$t = \text{reload value} * \frac{PSC}{40000}$$

برای مثال، با پیش تقسیم 32 و عدد ورودی 4000 زمان محاسبه شده $t=3.2$ ثانیه بدست می‌آید. یعنی برنامه نویس باید طوری عمل کند که قبل از این زمان، WD را با استفاده از تابعی که در زیر آمده مقدار دهی مجدد (Refresh) نماید. همچنانی اگر سیستم به هر علتی هنگ کند بعد از 3.2s ریست خواهد شد.

`HAL_IWDG_Refresh(&hiwdg);`

برای راه اندازی نیز از تابع `__HAL_IWDG_START(&hiwdg);` استفاده می‌شود.

مثال: iwdg را، راهاندازی و یک LCD را به میکرو متصل کنید. سپس برنامه‌ای بنویسید که وسط خط اول یک شمارنده بالا شمار 0 تا 15 نمایش داده شود. برنامه را یک بار بدون استفاده از تابع Refresh و یک بار با Refresh اجرا و نتایج را با هم مقایسه کنید.

```
/* Private includes ----- */
/* USER CODE BEGIN Includes */

#include "stlcd.h"
#include "stdio.h"

/* USER CODE END Includes */
/* USER CODE BEGIN PV */

int i=0;
char d[16];

/* USER CODE END PV */
/* USER CODE BEGIN 2 */
```

```
lcd_init();
__HAL_IWDG_START(&hiwdg);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */

    i++;
    if(i>15) i=0;
    sprintf(d,"%02d",i);
    lcd_gotoxy(7,0);
    lcd_puts(d);
    HAL_Delay(500);
    HAL_IWDG_Refresh(&hiwdg);
}

/* USER CODE END 3 */
```

برای شبیه سازی هنگ کردن سیستم ، می توان خط زیر را به برنامه اضافه و نتیجه اجرا را مشاهده نمود.

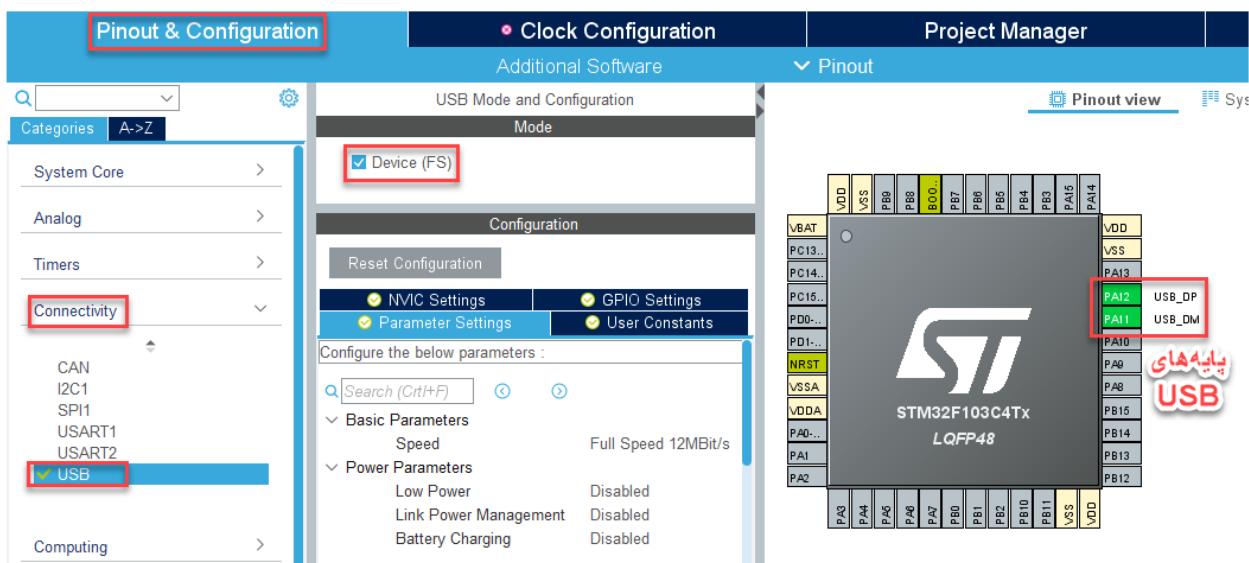
```
if(i==5) while(1);
```

ارتباط سریال : USB

یکی از فرآگیر و پرکاربردترین درگاه‌های ارتباط سریال، پورت USB می‌باشد. در اکثر میکروکنترلرهای STM32، پورت USB به صورت پیش فرض وجود داشته و می‌توان با تبدیل آن به پورت سریال مجازی، اطلاعات را به صورت سریال و بدون نیاز به آیسی مبدل بین کامپیوتر و میکرو انتقال داد.

برای راهاندازی این پورت بر روی STM32F103 مراحل زیر را طی می‌کنیم.

ابتدا مطابق شکل زیر از مسیر مشخص شده پورت USB را فعال می‌کنیم.

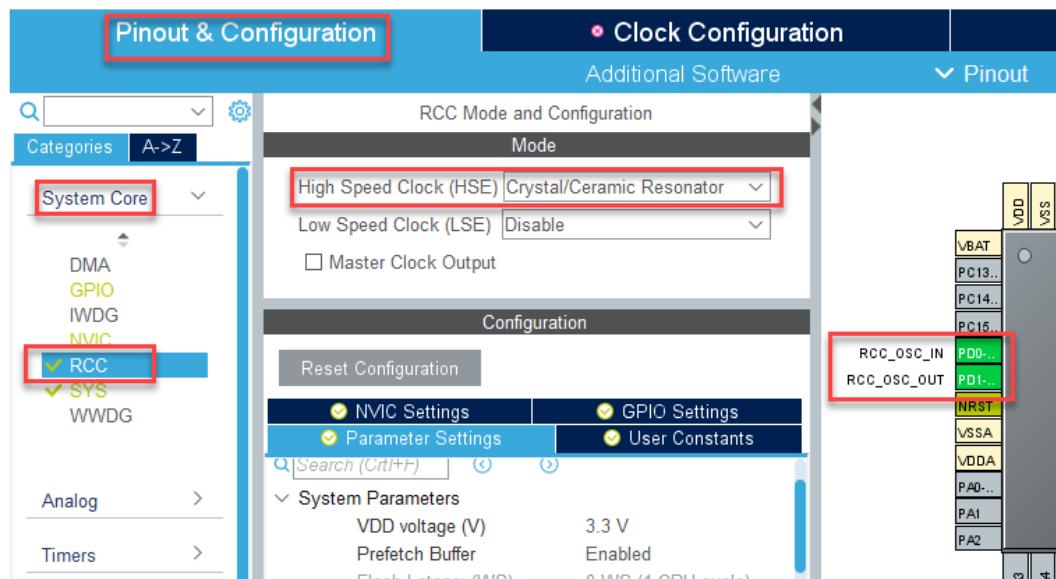


برای کنترل پورت USB دو روش نرمافزاری FS و سختافزاری HS وجود دارد.

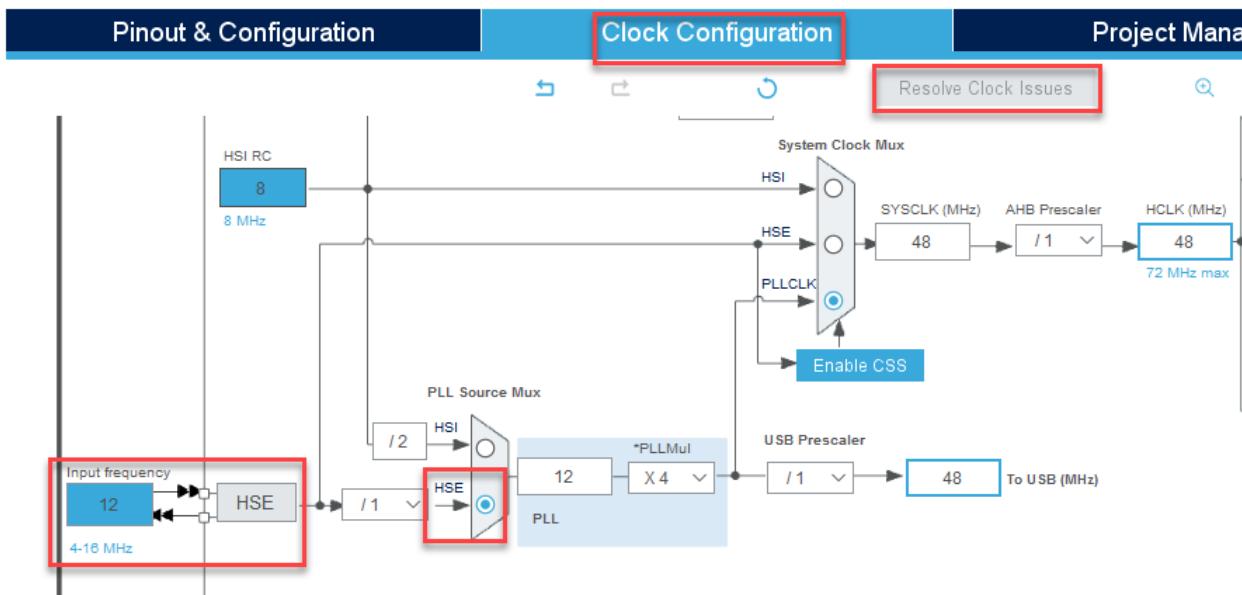
(FS: Fast Speed نرم افزاری کنترل می‌شود (در میکروهای سری F1 و F3)

و HS: High Speed در این حالت به یک لایه فیزیکی خارج از میکرو نیاز است . (در میکروهای سری F4 به بالا وجود دارد)

جهت عملکرد صحیح واحد USB لازم است فرکانس کار این بخش بر روی 48mhz قرار گیرد . به همین منظور ابتدا منبع Clock میکرو را روی کریستال خارجی قرار می‌دهیم.

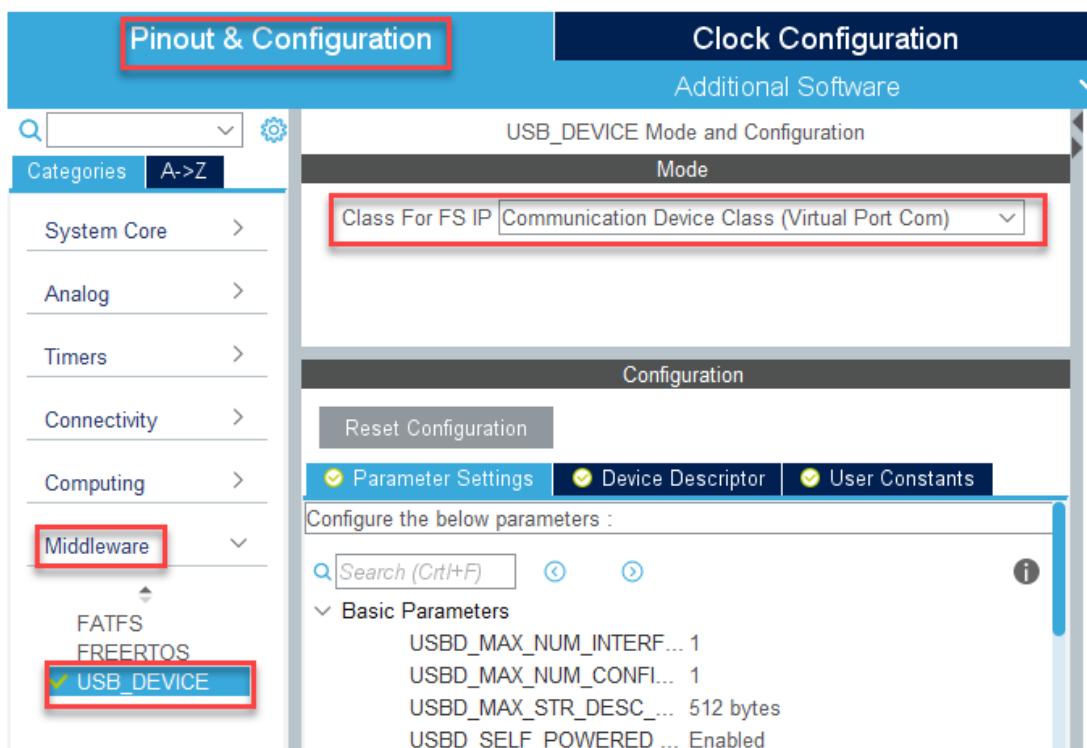


و در صفحه Clock Configuration تنظیم‌های زیر را انجام می‌دهیم.



توجه داشته باشید که ساده‌تر است با کلیک بر روی گزینه Resolve Clock Issues کار تنظیم فرکانس را بر عهده نرم‌افزار قرار دهید.

همچنین باید تنظیم‌های زیر نیز انجام شود.



ممکن است لازم باشد در صفحه Project Manager مقدار حافظه اختصاص یافته به Stack و Heap را افزایش دهیم . برای مثال در برد F407 (Heap=2000 Stack=4000) قرار گرفت.

برنامه: می خواهیم برنامه ای بنویسیم که هر ۲ ثانیه کلمه IRAN از طریق usb ارسال گردد.

ابتدا هدرفایل زیر را اضافه می کنیم. می توانید فایل usbd_cdc_if.c را باز و نام هدر را کپی و به فایل main.c اضافه کنید.

```
/* Includes ----- */
```

```
#include "main.h"
#include "usb_device.h"
```

سپس دیتای مورد نظر را که در مثال ما کلمه IRAN همراه با کarakترهای لازم جهت زیر هم چاپ شدن است در داخل یک آرایه قرار می دهیم .

```
/* USER CODE BEGIN PV */
```

```
uint8_t d[20]="IRAN \n\r";
/* USER CODE END PV */
```

تابع ارسال دیتا توسط **usb** به صورت زیر است.

```
CDC_Transmit_FS(تعداد بایت ارسالی , نام آرایه);
```

با توجه به موارد بالا ، حلقه اصلی برنامه به صورت زیر است.

```
while (1)
```

```
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
    CDC_Transmit_FS(d,7);
    HAL_Delay(1000);
}
```

دریافت: دریافت دیتا در فایل **usbd_cdc_if.c** و در داخل تابع زیر صورت می‌گیرد.

```
static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len)
```

```
{
/* USER CODE BEGIN 6 */
    USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);
    USBD_CDC_ReceivePacket(&hUsbDeviceFS);
    if(Buf[0]=='کاراکتر مورد نظر کاربر')
{
    برنامه کاربر
}
```

```

return (USBD_OK);

/* USER CODE END 6 */

}

```

هرگاه دیتایی از طریق پورت **usb** وارد می‌شود، در خانه صفرم آرایه **Buf** قرار می‌گیرد. کاربر می‌تواند با بررسی دیتای موجود در این خانه دستورهای مورد نظر خود را اجرا نماید.

برای مثال فرض کنید ۸ عدد LED به PA متصل است ، می‌خواهیم برنامه‌ای بنویسیم که اگر کاربر کاراکتر "a" را به میکرو ارسال کرد چهار LED اول و اگر حرف ارسالی "b" بود چهار LED دوم روشن شود.

```

static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len)

{
/* USER CODE BEGIN 6 */

USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);

USBD_CDC_ReceivePacket(&hUsbDeviceFS);

if(Buf[0]=='a')

{
    HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA,0x0f,GPIO_PIN_SET);

}

if(Buf[0]=='b')

{
    HAL_GPIO_WritePin(GPIOA,0xff,GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA,0xf0,GPIO_PIN_SET);

}

```

```

return (USBD_OK);

/* USER CODE END 6 */

}

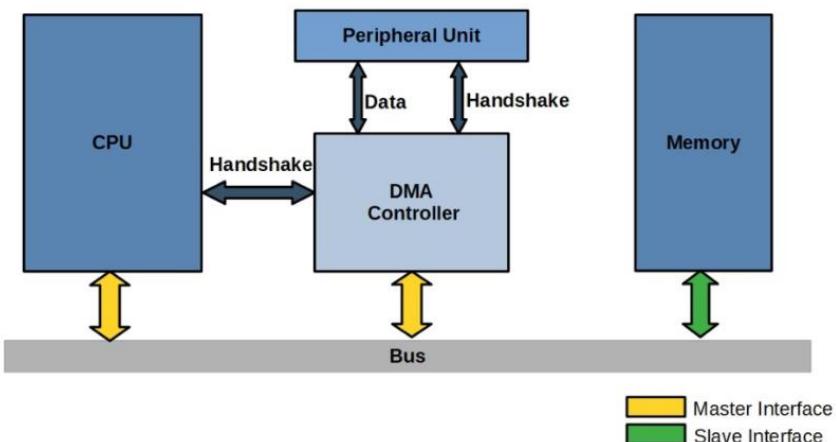
```

نکته: توجه داشته باشید ، برای این که کامپیوتر پورت usb میکرو را شناسایی کند نیاز به درایور دارد که باید آن را در اینترنت جستجو و سپس بر روی کامپیوتر نصب نمایید.

فایل STM32 virtual com PORT Driver en.stsw-stm32\02.zip را جستجو کنید.

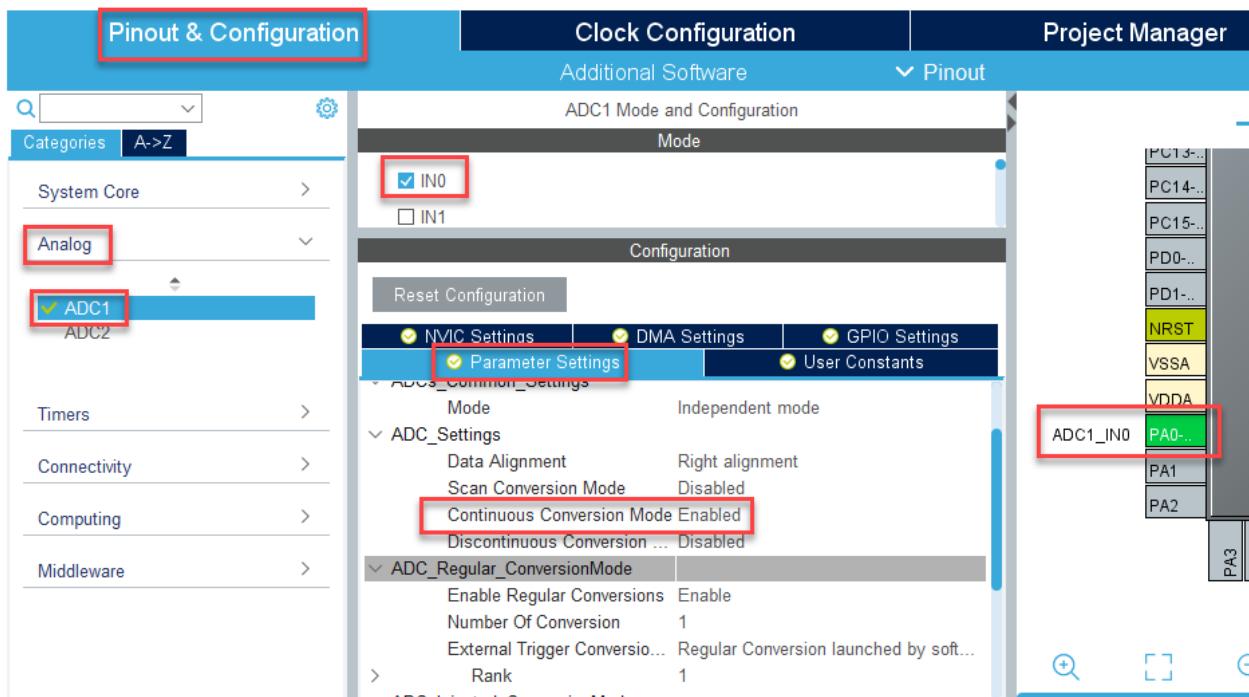
دسترسی مستقیم به حافظه (DMA)

دسترسی مستقیم به حافظه (DMA) روشی است که به دستگاه ورودی/خروجی (I/O) اجازه می‌دهد تا داده‌ها را مستقیماً به یا از حافظه اصلی ارسال یا دریافت کند و با دور زدن CPU، عملیات حافظه را سرعت بخشد. دسترسی مستقیم به حافظه (DMA) به منظور انتقال داده‌ها با سرعت بالا بین دستگاه‌های جانبی و حافظه و همچنین حافظه به حافظه استفاده می‌شود. داده‌ها را می‌توان به سرعت توسط DMA بدون هیچ گونه عملکرد CPU منتقل کرد. این کار منابع CPU را برای سایر عملیات آزاد نگه می‌دارد.



مثال: می‌خواهیم یکی از کانال‌های ADC را فعال و دیتای خروجی مبدل را با استفاده از DMA بخوانیم.

ابتدا در cube تنظیم‌های معمول ADC را انجام می‌دهیم. سپس مانند شکل زیر continuous conversion را فعال می‌کنیم.

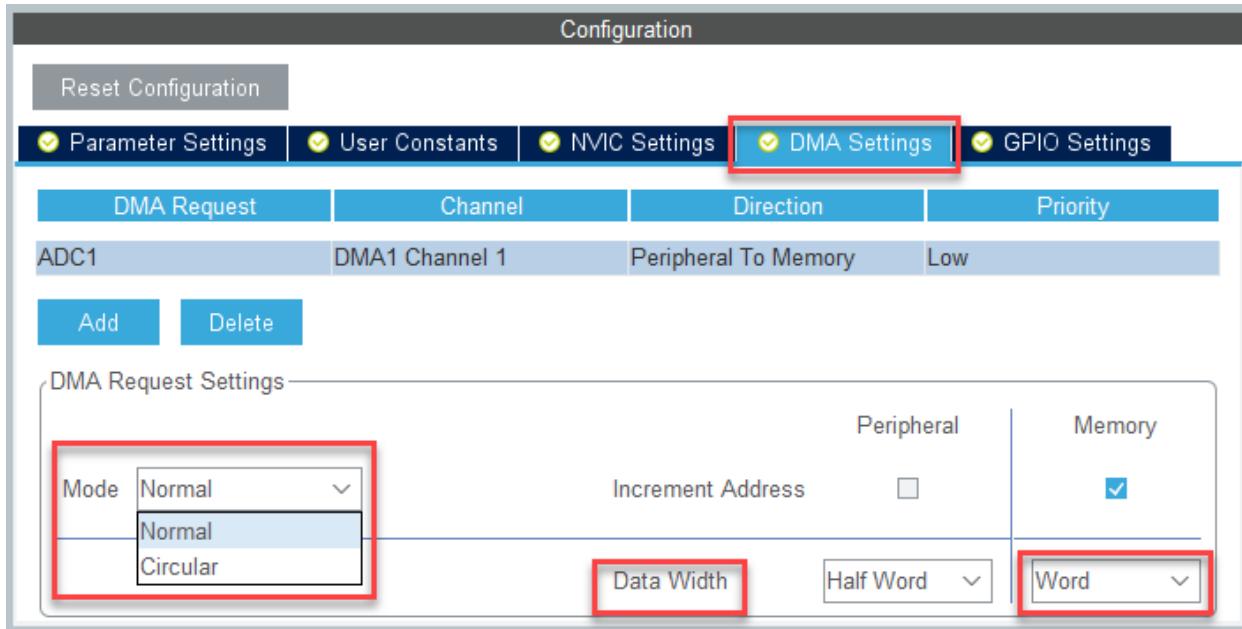


در مرحله بعد برگه DMA بخش ADC را باز و با کلیک بر روی گزینه ADC1 را انتخاب می‌کنیم.



با اضافه کردن DMA به ADC لازم است با توجه به شکل زیر در قسمت Mode مشخص کنیم که خواندن ADC به صورت Normal باشد یا Circular . اگر قرار است پس از خواندن دیتا پردازشی روی آنها صورت بگیرد و سپس دیتا جدید خوانده شود ، بهتر است Normal را انتخاب کنید در این صورت لازم است پس از اتمام هر مرحله DMA را دوباره start کنید.

در بخش Data With می‌توانید مشخص کنید دیتا درون حافظه ، ۱۶ بیتی (Half Word) یا ۳۲ بیتی (Word) ذخیره شود.



در keil برای خواندن دیتا ، آرایه‌ای با تعداد خانه‌هایی که می‌خواهیم به آن تعداد دیتا بخوانیم یا بیشتر تعريف می‌کنیم. حال کافی است تابع زیر را اجرا کنیم.

`HAL_ADC_Start_DMA(&hadc1,(unsigned int *) ;` (تعداد دیتا ، نام آرایه)

هرگاه به تعداد مشخص شده در تابع ، دیتا خوانده و در آرایه قرار گرفت وقفه‌ای رخ خواهد داد که می‌توانیم با استفاده از تابع `call back` زیر برای پردازش دیتا اقدام کنیم.

`void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)`

{

برنامه کاربر

`HAL_ADC_Start_DMA(&hadc1,(unsigned int *) ;` (تعداد دیتا ، نام آرایه)

}

اگر DMA را در حالت Normal قرار داده باشیم لازم است پس از برنامه کاربر دوباره `start` کنیم.

مثال: می خواهیم کانالی از ADC را ، توسط DMA ۲۰ بار بخوانیم و میانگین آنها را روی LCD نمایش دهیم.

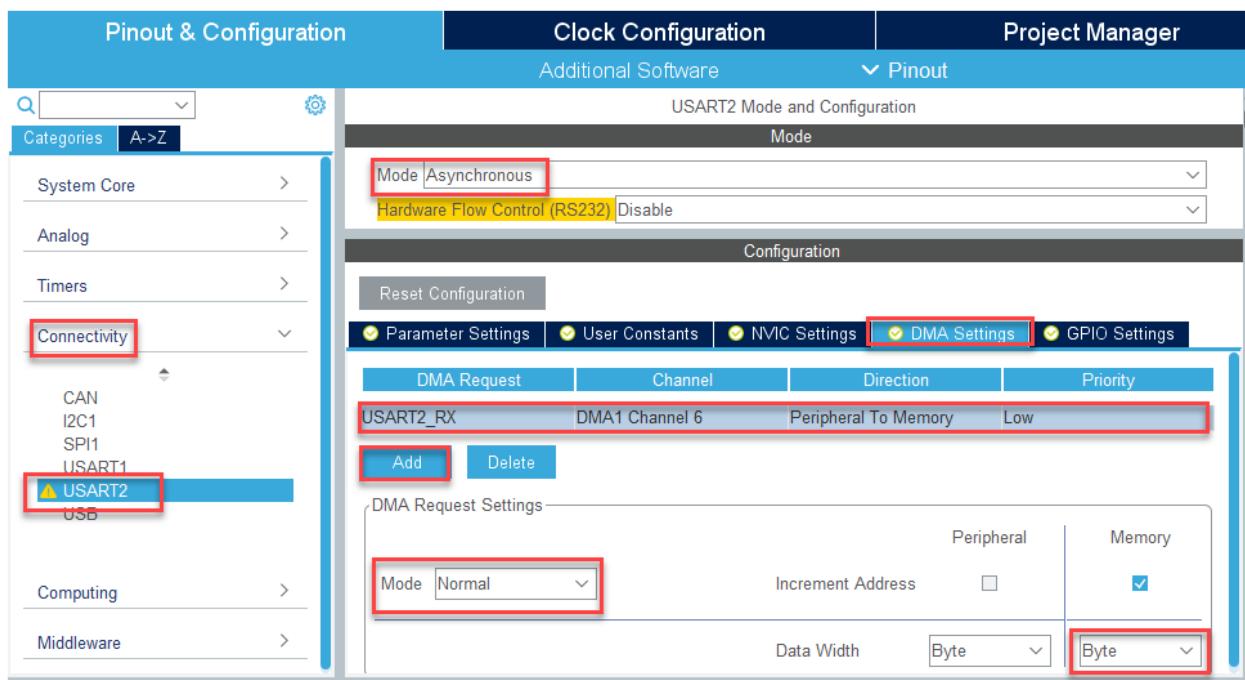
```
/* USER CODE BEGIN Includes */
#include "stlcd.h"
#include "stdio.h"
/* USER CODE END Includes */
/* USER CODE BEGIN PV */
char t[16],i;
int ad[20],sum,m;
float v;
/* USER CODE END PV */
/* USER CODE BEGIN 2 */
lcd_init();
HAL_ADC_Start_DMA(&hadc1,(unsigned int *)ad,20);
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

```
/* USER CODE BEGIN 4 */

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    sum=0;
    for(i=0;i<20;i++)
    {
        sum=ad[i]+sum;
    }
    m=sum/20.0;
    v=(3.3/4095.0)*m;
    sprintf(t,"%4.2f",v);
    lcd_gotoxy(0,1);
    lcd_puts(t);
    HAL_ADC_Start_DMA(&hadc1,(unsigned int *)ad,20);
}

/* USER CODE END 4 */
```

مثال: می خواهیم UART2 را ، راه اندازی و توسط DMA بخوانیم. هرگاه تعداد کاراکترهای ورودی به ۵ بایت رسید رشته ورودی را روی LCD نمایش دهید.



```

/* USER CODE BEGIN Includes */

#include "stlcd.h"

#include "stdio.h"

/* USER CODE END Includes */

/* USER CODE BEGIN PV */

char d[20];

/* USER CODE END PV */

/* USER CODE BEGIN 2 */

lcd_init();

HAL_UART_Receive_DMA(&huart2,(unsigned char *)d,5);

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */

```

```
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

}

/* USER CODE END 3 */

/* USER CODE BEGIN 4 */

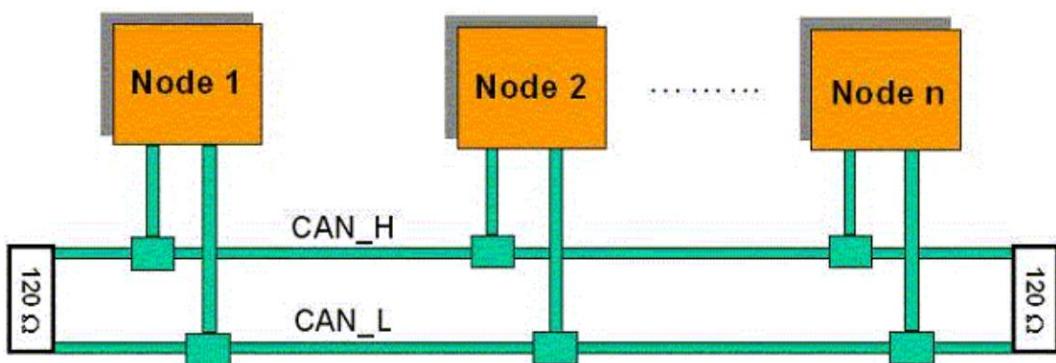
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    /* Prevent unused argument(s) compilation warning */
    // UNUSED(huart);

    /* NOTE: This function should not be modified, when the callback is needed,
       the HAL_UART_RxCpltCallback could be implemented in the user file
    */
    if(huart->Instance==USART2)
    {
        lcd_gotoxy(0,0);
        lcd_puts(d);
        HAL_UART_Receive_DMA(&huart2,(unsigned char *)d,5);
    }
}

/* USER CODE END 4 */
```

ارتباط CAN (Controller Area Network)

این پروتکل ابتدا برای ارتباط بین قسمت‌های مختلف خودرو پایه‌ریزی شد. این پروتکل در سال ۱۹۸۰ توسط شرکت Bosch با هدف کاهش سیم‌کشی‌های انبوه موجود در خودروها، ابداع شد. و پس از موفقیت در صنعت خودرو، در دیگر صنایع نیز مورد استفاده قرار گرفت. پروتکل CAN علاوه بر تأمین ایمنی در سطح شبکه، نیاز به سخت‌افزار پیچیده‌ای برقراری ارتباط ندارد و از دو رشته سیم به هم تابیده و خطوط دیفرانسیلی استفاده می‌کند. با توجه به امنیت بالا، از پروتکل CAN در هوایپیماها نیز استفاده شده است و سنسورهای سیستم ناوبری هوایپیما برای برقراری ارتباط با کامپیوتر مرکزی از این پروتکل استفاده می‌کنند.



بعضی از ویژگی‌های این پروتکل:

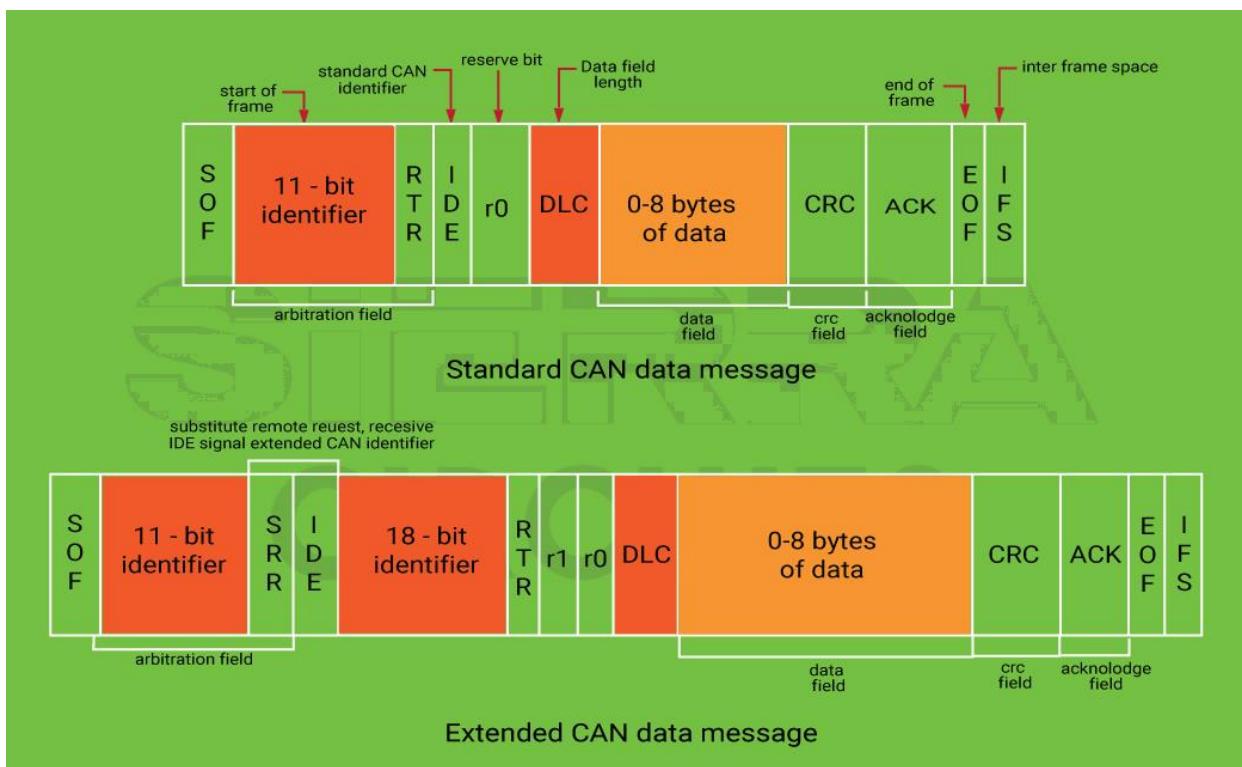
- اگر در شبکه CAN یکی از Node‌ها کرش یا هنگ کند، به صورت اتوماتیک توسط لایه‌ی فیزیکی از شبکه خارج می‌شود و شبکه به کار خود ادامه می‌دهد.
- عدم نیاز به عناصر خارجی زیاد و سادگی مدار راهانداز؛ تنها به یک واسطه لایه‌ی فیزیکی نیاز دارد که معمولاً یک آی سی ۸ پایه است.
- پروتکل CAN با توجه به لایه‌ی فیزیکی مورد استفاده (خطوط داده دیفرانسیلی) قادر است در محیط‌هایی با نویز زیاد مثل محیط‌های صنعتی کار کنند.
- پروتکل CAN از لایه‌بندی مدل استاندارد OSI استفاده می‌کند.
- تأمین امنیت داده‌های ارسالی با استفاده از CRC در لایه‌ی سخت‌افزاری
- اولویت‌بندی پیام‌های ارسالی به نحوی که داده‌هایی با اولویت بالاتر زودتر ارسال می‌شوند.
- شبکه‌ی CAN یک شبکه Real Time است؛ یعنی ارسال و دریافت داده‌ها را در بازه‌های زمانی مشخص، گارانتی می‌کند.
- فریم‌های ارسالی در CAN به شکل زیر می‌باشد.

دو نوع استاندارد در این پروتکل با نام‌های زیر وجود دارد:

استاندارد CAN 2.0 A

استاندارد CAN 2.0 B

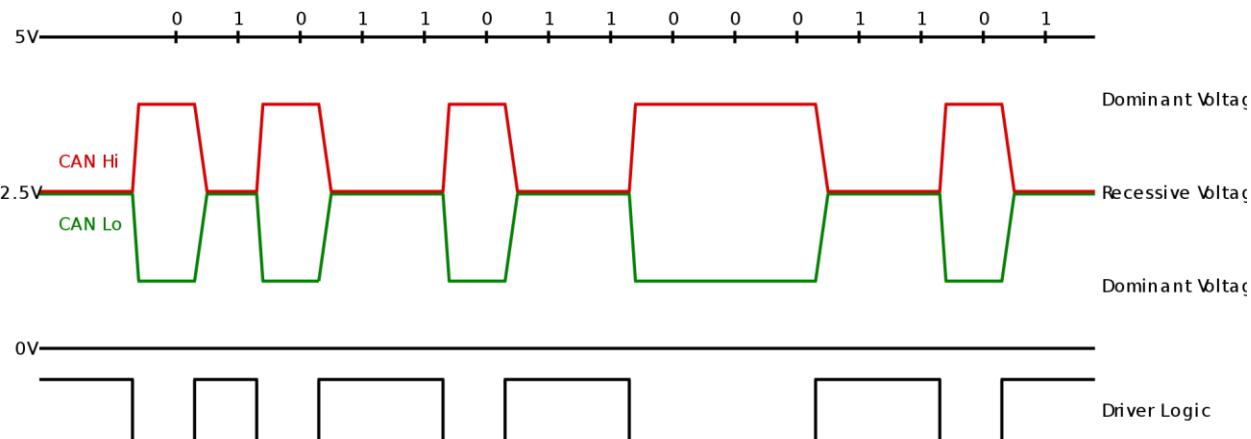
تفاوت این دو استاندارد در تعداد بیت‌های آدرس‌دهی (identifier) است؛ به صورتی که در استاندارد CAN 2.0A تعداد بیت‌های مجاز برای آدرس ۱۱ بیت هستند و در استاندارد CAN 2.0B تعداد بیت مجاز به ۲۹ بیت گسترش یافته‌اند.



همان‌طور که در تصویر فوق مشاهده می‌کنید، هر بسته‌ی داده‌ای دارای یک آدرس منحصر به‌فرد است که بسته به نوع استاندارد CAN مورد استفاده، می‌تواند ۱۱ یا ۲۹ بیت طول داشته باشد. تعداد داده‌های موجود در هر فریم توسط DLC مشخص می‌شود که در بیشترین حالت می‌تواند ۸ بایت داده باشد. بخش بعد CRC است که توسط سخت‌افزار تولید و بررسی می‌شود. در صورت صحت آدرس و CRC، گیرنده با ارسال ACK فرستنده را از دریافت اطلاعات آگاه می‌کند.

در تصویر زیر ، بس دیفرانسیلی مورداستفاده در CAN طبق استاندارد ISO 11898-2 رسم شده است . مقدار ولتاژ مرجع 2.5 ولت می باشد. تعیین سرعت مورداستفاده یکی از پارامترهای مهم در شبکه CAN است. با توجه به طول بس، باید از سرعت مناسب استفاده کرد. در بیشترین حالت، طبق استاندارد طول بس می تواند تا ۵۰۰ متر باشد و بیشترین سرعت قابل پشتیبانی در ISO 11898-2، یک مگابیت بر ثانیه است. اما محدودیت هایی وجود دارد که باید آنها را رعایت کرد؛ مثلاً نمی توان انتظار داشت در خطوط دیتا با طول ۲۰۰ متر بتوان داده ها را با سرعت ۱ مگابیت بر ثانیه منتقل کرد. برای سهولت ما چهار مقدار از تناسب طول خطوط داده و بیت ریت را آمده کرده ایم که به شرح زیر هستند:

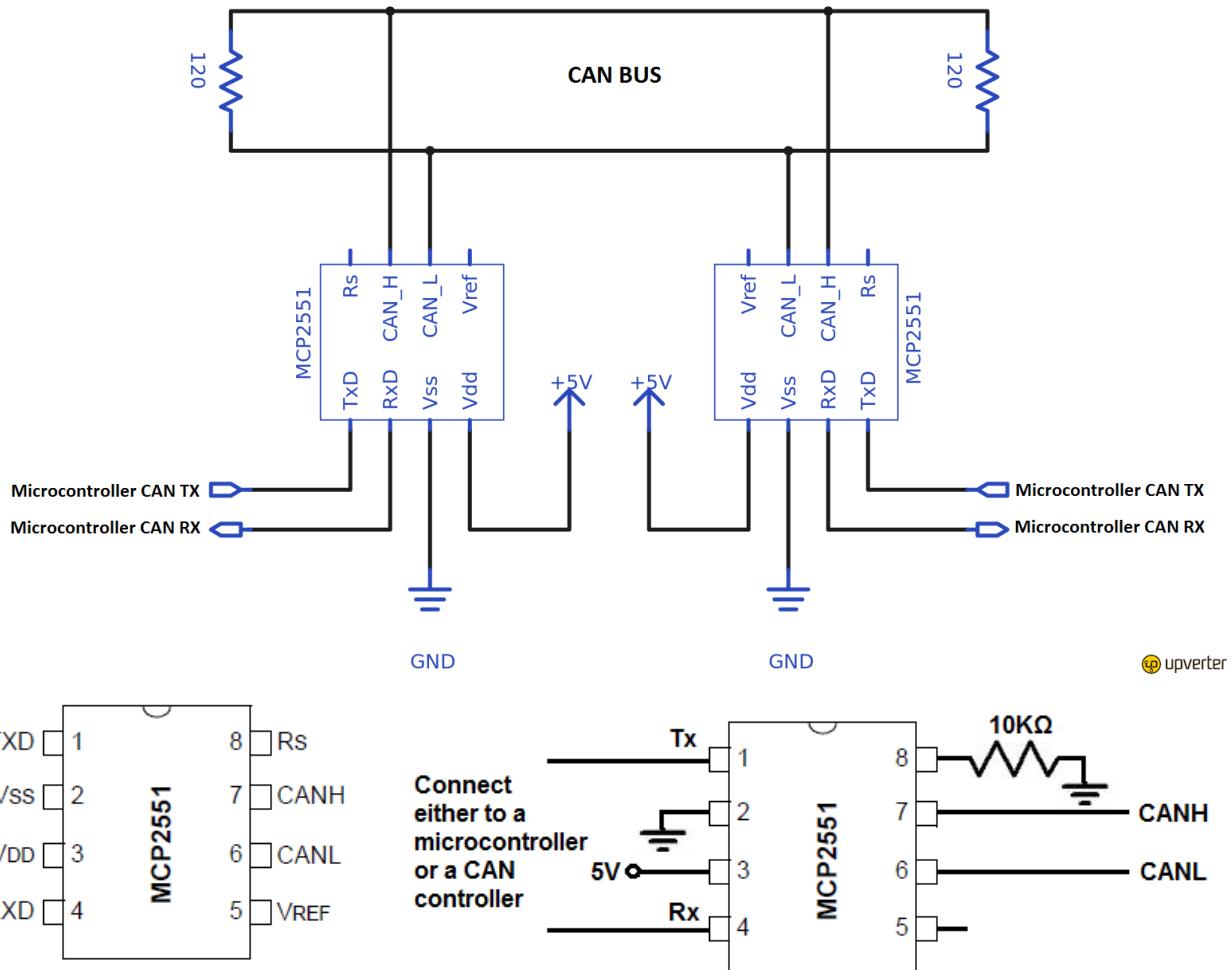
۱. سرعت ۱ مگابیت بر ثانیه برای خطوط داده با حداکثر طول ۴۰ متر
۲. سرعت ۵۰۰ کیلوبیت بر ثانیه برای خطوط داده با حداکثر طول ۱۰۰ متر
۳. سرعت ۲۵۰ کیلوبیت بر ثانیه برای خطوط داده با حداکثر طول ۲۰۰ متر
۴. سرعت ۱۲۵ کیلوبیت بر ثانیه برای خطوط داده با حداکثر طول ۵۰۰ متر



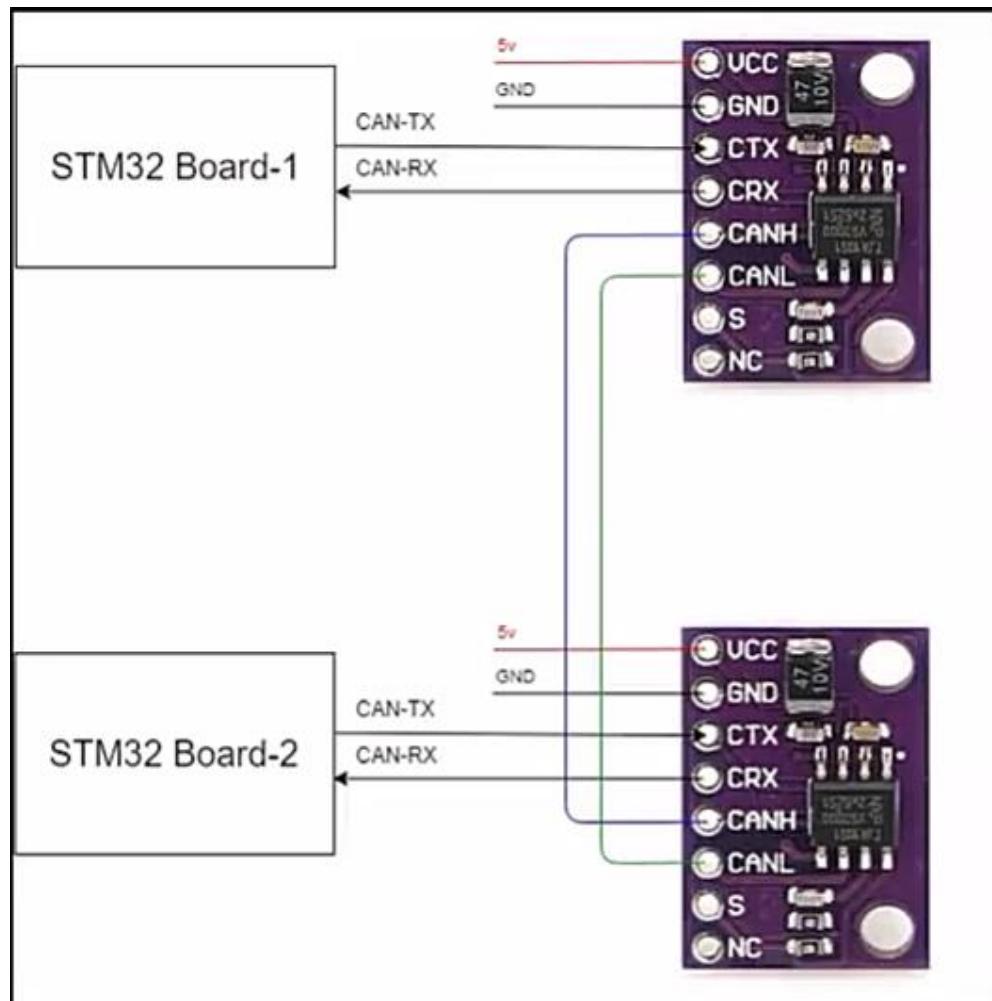
سخت افزار لایه فیزیکی :

معمولًا به دلیل آسیب پذیری لایه فیزیکی با توجه به نوع عملکرد آن، این قسمت را در چیپ میکروکنترلر قرار نمی دهند تا در صورت بروز حادثه در بس، که موجب سوختن این آی سی واسط می شود، به میکروکنترلر یا کنترلر CAN صدمه وارد نشود. آی سی های مختلف توسط شرکت های مختلف برای واسط شبکه های CAN ساخته شده اند ولی در دسترس ترین و مناسب ترین آن ها، آی سی MCP2551 ساخت شرکت میکرو چیپ است. در نگاه اول، افرادی که قصد راه اندازی بس CAN توسط میکروکنترلرهای ARM را دارند، به دلیل وجود تغذیه های ۵ ولت این آی سی، از آن استفاده نمی کنند. معادل این آی سی با تغذیه های ۳.۳ ولت، هم کمیاب است و هم قیمتی تا ۴ برابر این آی سی دارد. در صورتی که به راحتی و بدون هیچ

مدار اضافه‌ای می‌توان این آی سی را به میکروکنترلر ARM متصل کرد (در صورتی که میکروکنترلر از ۳.۳ استفاده کند). فقط دقت داشته باشید که پایه‌ی ۸ را حتماً به زمین متصل کنید. اگر بتوانید یک خازن ۱۰۰ نانو فاراد هم بر روی پایه‌ی ۵ و زمین قرار دهید که بهتر خواهد بود. پایه‌ی ۵ در واقع همان ولتاژ رفرنس ۲.۵ ولت است که برای تشخیص سطح منطقی باس به کار می‌رود.

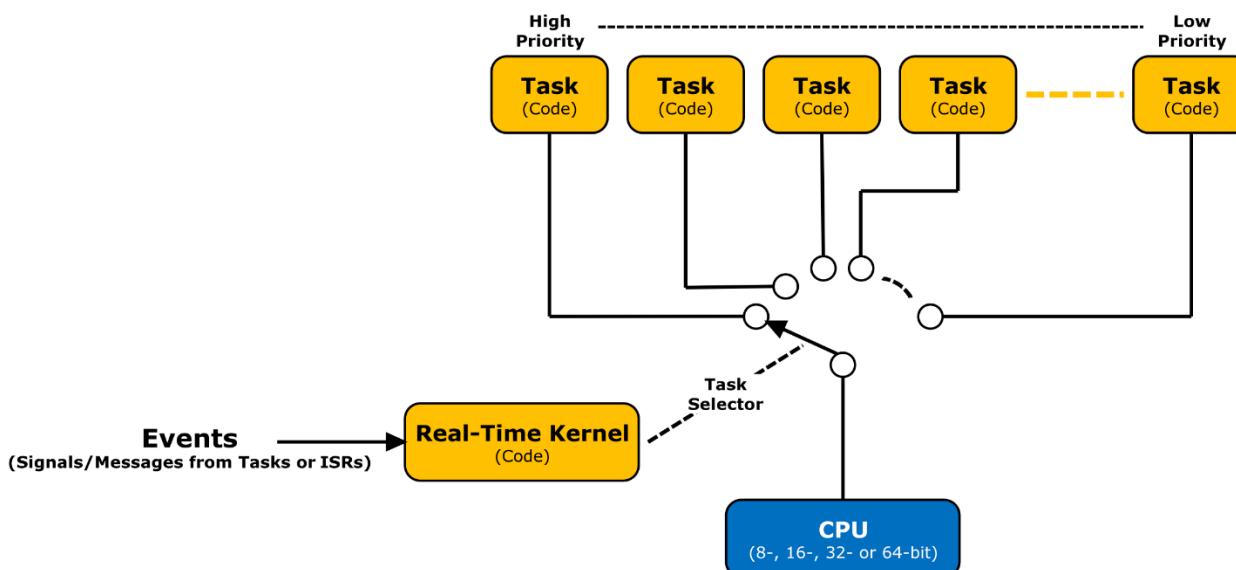


در شکل زیر نحوه اتصال دو میکرو با استفاده از مازول CAN نشان داده شده است.

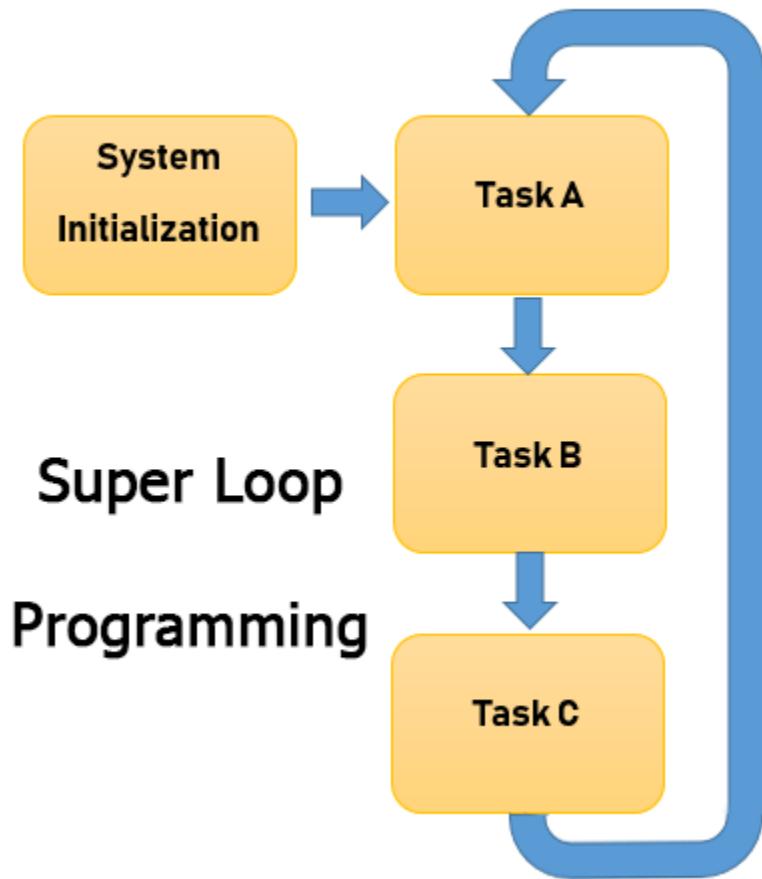


سیستم عامل های بلادرنگ RTOS

به نظر می رسد اکثر سیستم عامل ها به چندین برنامه اجزاهی همزمان را می دهند. به این کار چند وظیفه ای می گویند. در واقع، هر هسته پردازنده تنها می تواند یک رشته اجرا را در هر نقطه از زمان اجرا کند. بخشی از سیستم عامل به نام زمانبند، مسئول تصمیم گیری در مورد زمان اجرای برنامه است و با جابجایی سریع بین هر برنامه، توهمندی همزمان را فراهم می کند.



در برنامه های متداول که برای میکروکنترلرهای نوشته می شود یک حلقه اصلی super loop داریم که تمام خواسته های برنامه نویس در آن قرار گرفته است و به ترتیب اجرا می شوند.



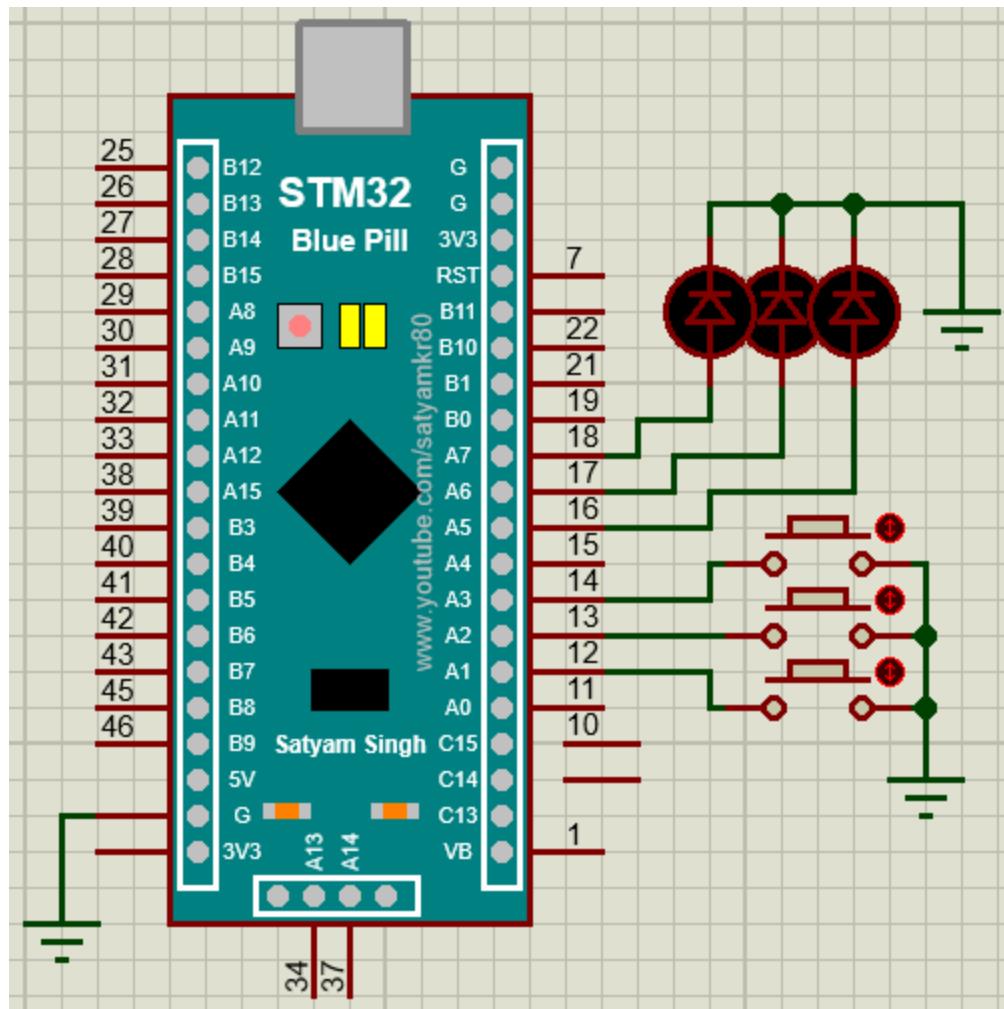
سیستم عامل بلادرنگ (RTOS) یک سیستم عامل با دو ویژگی کلیدی است: پیش بینی پذیری و جبر. در یک RTOS، وظایف مکرر در یک محدوده زمانی محدود انجام می شود، در حالی که در یک سیستم عامل همه منظوره، لزوماً اینطور نیست.

یک کلاس از RTOS است که به اندازه کافی کوچک برای اجرا روی یک میکروکنترلر طراحی شده است - اگرچه استفاده از آن به برنامه های میکروکنترلر محدود نمی شود.

میکروکنترلر یک پردازنده کوچک و با منابع محدود است که روی یک تراشه، خود پردازنده، حافظه فقط خواندنی (Flash یا ROM) را برای نگه داشتن برنامه برای اجرا و حافظه دسترسی تصادفی (RAM) مورد نیاز برنامه ها را در خود جای داده است. اجرا می کند. معمولاً برنامه مستقیماً از حافظه فقط خواندنی اجرا می شود.

میکروکنترلرها در برنامه‌های عمیق تعبیه شده استفاده می‌شوند که معمولاً یک کار بسیار خاص و اختصاصی برای انجام دارند. محدودیت‌های اندازه و ماهیت برنامه نهایی اختصاصی، به ندرت استفاده از یک پیاده‌سازی کامل RTOS را تضمین می‌کند - یا در واقع استفاده از یک پیاده‌سازی کامل RTOS را ممکن می‌سازد. بنابراین FreeRTOS عملکرد اصلی زمان‌بندی زمان واقعی، ارتباطات بین وظایف، زمان‌بندی و همگام‌سازی اولیه را فراهم می‌کند. این بدان معنی است که با دقت بیشتری به عنوان یک هسته واقعی یا اجرایی زمان واقعی توصیف می‌شود. عملکردهای اضافی، مانند رابط کنسول فرمان، یا پسته‌های شبکه، می‌توانند با اجزای الحقیقی گنجانده شوند.

برای درک بهتر عملکرد میکرو در حالت معمولی فرض کنید سه عدد کلید و سه عدد LED را به میکرو متصل کنیم. حال برنامه‌ای می‌نویسیم که با نگه داشتن هر کلید، LED مناسب به آن کلید شروع به چشمک زدن کند.



```
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_1)==0)
    {
        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
        HAL_Delay(100);
    }

    while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_2)==0)
    {
        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_6);
        HAL_Delay(100);
    }

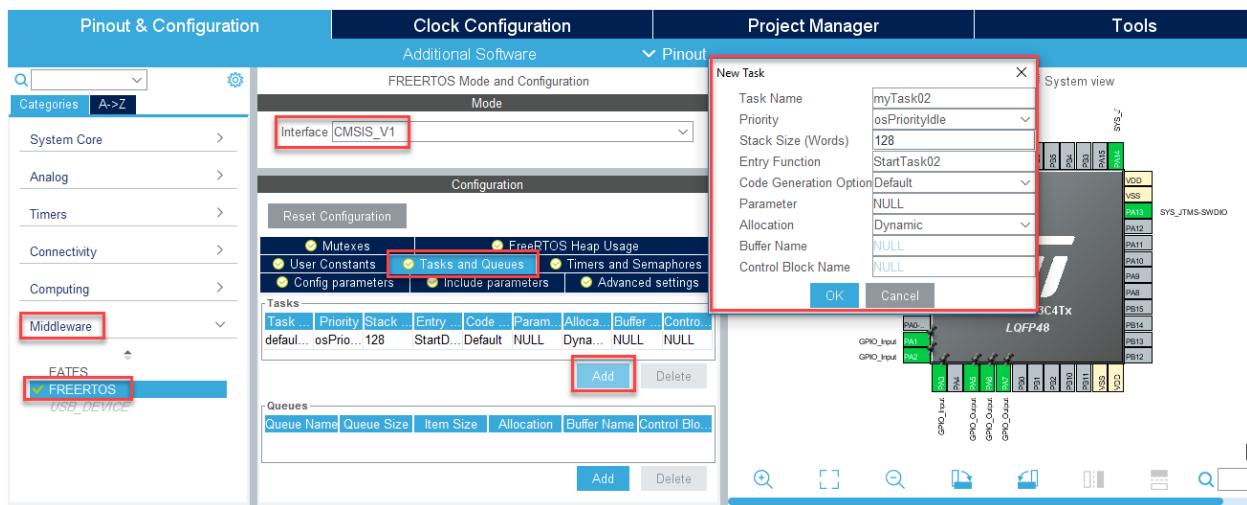
    while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_3)==0)
    {
        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_7);
        HAL_Delay(100);
    }

}

/* USER CODE END 3 */
```

همان طور که در برنامه بالا دیده می شود اگر کاربر کلیدی را فعال کند برنامه درون حلقه آن کلید قرار گرفته و تا وقتی کلید غیر فعال نشود از حلقه آن کلید خارج نخواهد شد . بنابراین اگر کلیدی فعال باشد فشردن کلیدهای دیگر نادیده گرفته خواهد شد . حال اگر بخواهیم این مشکل را حل کنیم یکی از روش ها استفاده از RTOS می باشد. تا با تعریف چند Task خواندن هر کلید را در یک Task قرار داده و این مشکل را حل کنیم.

در شکل زیر تنظیم‌های لازم در Cube برای راهاندازی RTOS نشان داده شده است.



با فعال کردن این بخش یک Task به صورت پیش فرض تعریف شده است برای اضافه کردن Task های دیگر با کلیک بر روی گزینه ADD پنجره New Task باز می شود که می توان علاوه بر نام ، موارد دیگری را نیز برای آن تعیین کرد. برای مثال ما ، دو Task جدید نیاز است که جملا سه Task خواهیم داشت.

پس از تولید کد، در فایل main مانند کدهای زیر برای هر Task یک زیر برنامه باز می‌شود که کاربر می‌تواند برنامه مورد نظر خود را در داخل آن بنویسد.

نکته: توجه داشته باشید که درون Task ها برای ایجاد تاخیر از تابع HAL_Delay استفاده نکنید برای تاخیر از تابع osDelay(ms) استفاده شود.

```
/* USER CODE END Header_StartDefaultTask */
```

```
void StartDefaultTask(void const * argument)
```

{

```
/* USER CODE BEGIN 5 */

/* Infinite loop */

for(;;)

{

    osDelay(1);

}

/* USER CODE END 5 */

}

/* USER CODE BEGIN Header_StartTask02 */

/**

 * @brief Function implementing the myTask02 thread.

 * @param argument: Not used

 * @retval None

 */

/* USER CODE END Header_StartTask02 */

void StartTask02(void const * argument)

{

    /* USER CODE BEGIN StartTask02 */

    /* Infinite loop */

    for(;;)

    {

        osDelay(1);

    }

}
```

```
}

/* USER CODE END StartTask02 */

}

/* USER CODE BEGIN Header_StartTask03 */

/**
 * @brief Function implementing the myTask03 thread.
 * @param argument: Not used
 * @retval None
 */

/* USER CODE END Header_StartTask03 */

void StartTask03(void const * argument)

{
    /* USER CODE BEGIN StartTask03 */

    /* Infinite loop */
    for(;;)
    {
        osDelay(1);
    }

    /* USER CODE END StartTask03 */
}
```

به این ترتیب برنامه مثال مربوط به کلیدها به صورت زیر خواهد بود.

```
/* USER CODE END Header_StartDefaultTask */

void StartDefaultTask(void const * argument)

{
    /* USER CODE BEGIN 5 */

    /* Infinite loop */

    for(;;)
    {
        while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_1)==0)

        {
            HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);

            osDelay(100);
        }
    }

    /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_StartTask02 */

/**
 * @brief Function implementing the myTask02 thread.
 *
 * @param argument: Not used
 *
 * @retval None
 */

```

```
/* USER CODE END Header_StartTask02 */

void StartTask02(void const * argument)

{

/* USER CODE BEGIN StartTask02 */

/* Infinite loop */

for(;;)

{

    while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_2)==0)

    {

        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_6);

        osDelay(100);

    }

}

/* USER CODE END StartTask02 */

}

/* USER CODE BEGIN Header_StartTask03 */

/**



* @brief Function implementing the myTask03 thread.

* @param argument: Not used

* @retval None

*/



/* USER CODE END Header_StartTask03 */

void StartTask03(void const * argument)
```

```
{  
/* USER CODE BEGIN StartTask03 */  
/* Infinite loop */  
for(;;)  
{  
    while(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_1)==3)  
    {  
        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_7);  
        osDelay(100);  
    }  
}  
/* USER CODE END StartTask03 */  
}
```

راهاندازی LCD گرافیکی :

برای این بخش از LCD گرافیکی سیاه و سفید با کنترلر KS108 و با ابعاد 128×64 استفاده خواهیم کرد.



این lcd دارای ۲۰ پایه به شرح زیر می باشد .

PIN	1	2	3	4	5	6	7	8	9	10
SIGNAL	Vss	VDD	V0	D/I	R/W	E	DB0	DB1	DB2	DB3
PIN	11	12	13	14	15	16	17	18	19	20
SIGNAL	DB4	DB5	DB6	DB7	CS1	CS2	RES	VEE	A	K

پایه ۱ : Vss اتصال به GND برای تغذیه LCD

پایه ۲ : VDD اتصال به ۵ ولت برای تغذیه LCD

پایه ۳ : V0 متصل به سر وسط یک پتانسیومتر برای تنظیم کنتراست

پایه ۴ : اگر این پایه را صفر کنیم یعنی می‌خواهیم به lcd دستور بدھیم یک یعنی دیتا

پایه ۵ : اگر این پایه را صفر کنیم یعنی می‌خواهیم در lcd بنویسیم یک یعنی می‌خوانیم

پایه ۶ : برای ورود دستور یا دیتا باید روی این پایه یک لبه پایین رونده ایجاد کنیم برای این منظور ابتدا این پایه را یک و سپس صفر می‌کنیم.

پایه ۷ تا ۱۴ : D0_D7 دستور یا دیتا برای ورود روی این پایه‌ها قرار می‌گیرد

پایه ۱۵ : با یک شدن این پایه قطعه 64×64 سمت چپ LCD فعال می‌شود.

پایه ۱۶ : با یک شدن این پایه قطعه 64×64 سمت راست LCD فعال می‌شود.

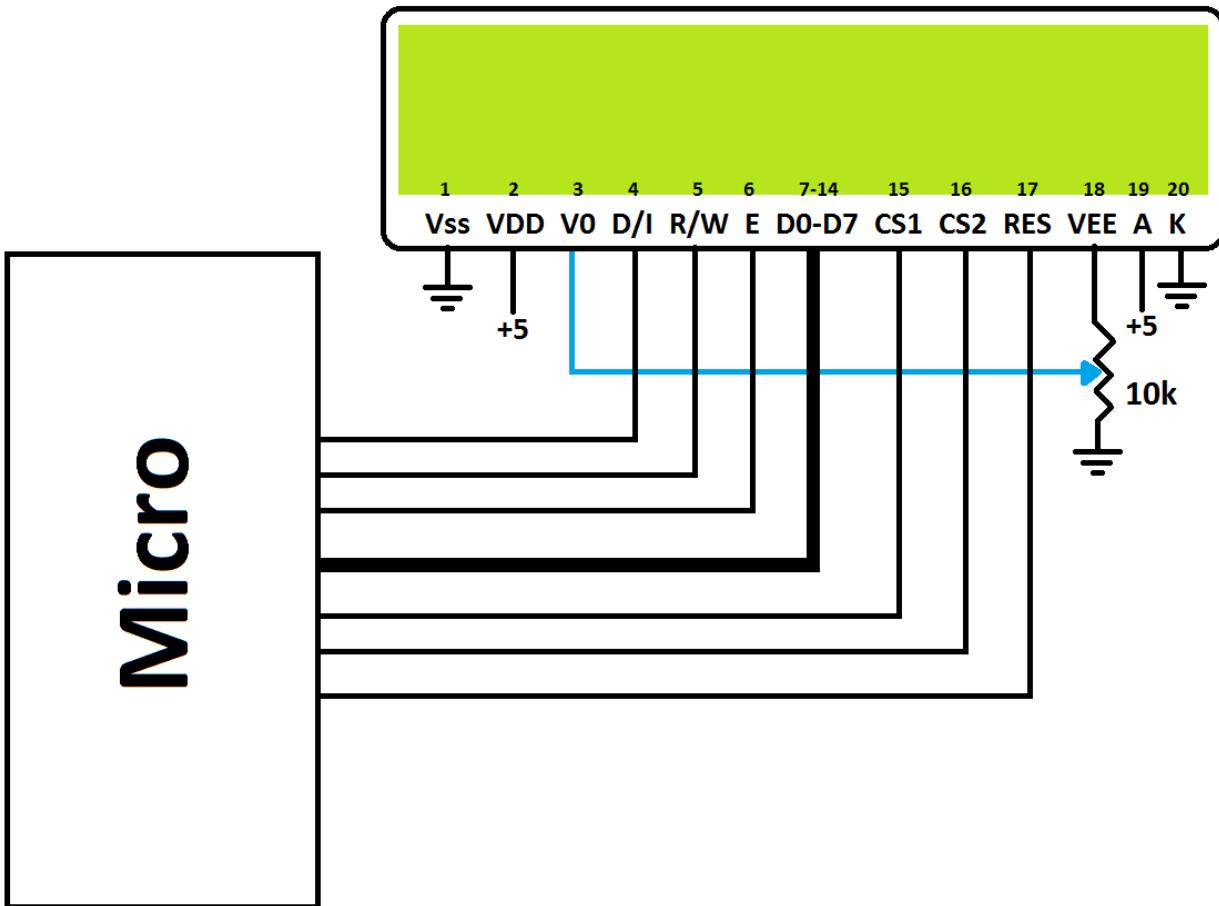
پایه ۱۷ : RES با صفر و یک کردن این پایه رجیسترهاي داخلی روی مقادیر پیش فرض قرار می‌گیرند.

پایه ۱۸ : VEE روی این پایه ولتاژ ۱۵- ولت برای اتصال به پتانسیومتر کنتراست وجود دارد.

پایه ۱۹ : A پایه آند LED مربوط به نور پشت صفحه Back Light که به ۵+ ولت متصل می‌شود.

پایه ۲۰ : K پایه کاتد LED مربوط به نور پشت صفحه Back Light که به GND متصل می‌شود.

در شکل زیر نحوه اتصال LCD به میکرو نشان داده شده است.



با توجه به برگه‌های دیتا شیت، رسم و کنترل تصویر بر روی LCD با سه رجیستر زیر امکان پذیر است.

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Read Display Date	1	1	Read data						Reads data (DB[7:0]) from display data RAM to the data bus.		
Write Display Date	1	0	Write data						Writes data (DB[7:0]) into the DDRAM. After writing instruction, Y address is incremented by 1 automatically		
Status Read	0	1	Busy	0	ON/OFF	Re-set	0	0	0	0	Reads the internal status BUSY 0: Ready 1: In operation ON/OFF 0: Display ON 1: Display OFF RESET 0: Normal 1: Reset
Set Address (Y address)	0	0	0	1	Y address (0~63)						Sets the Y address at the column address counter
Set Display Start Line	0	0	1	1	Display start line (0~63)						Indicates the Display Data RAM displayed at the top of the screen.
Set Address (X address)	0	0	1	0	1	1	1	Page (0~7)			Sets the X address at the X address register.
Display On/off	0	0	0	0	1	1	1	1	1	0/1	Controls the display ON or OFF. The internal status and the DDRAM data is not affected. 0: OFF, 1: ON

رجیستر X

در این LCD ، به هر ۸ سطر یک Page گفته می شود. رجیستر X مشخص می کند کدام Page فعال باشد. عدد داخل این رجیستر می تواند از 0xbf برای Page0 تا 0xb8 برای Page7 باشد.

رجیستر Y

این رجیستر مشخص می کند کدام ستون فعال باشد. با توجه به این که هر بخش LCD دارای ۶۴ ستون می باشد عدد داخل این رجیستر می تواند از 0x40 برای ستون صفر تا 0x7f برای ستون ۶۳ باشد. توجه داشته باشید هر گاه دیتایی در یک ستون قرار می گیرد عدد داخل رجیستر ۷ به طور خودکار یک واحد اضافه می شود.

رجیستر Z : Display Start Line Register

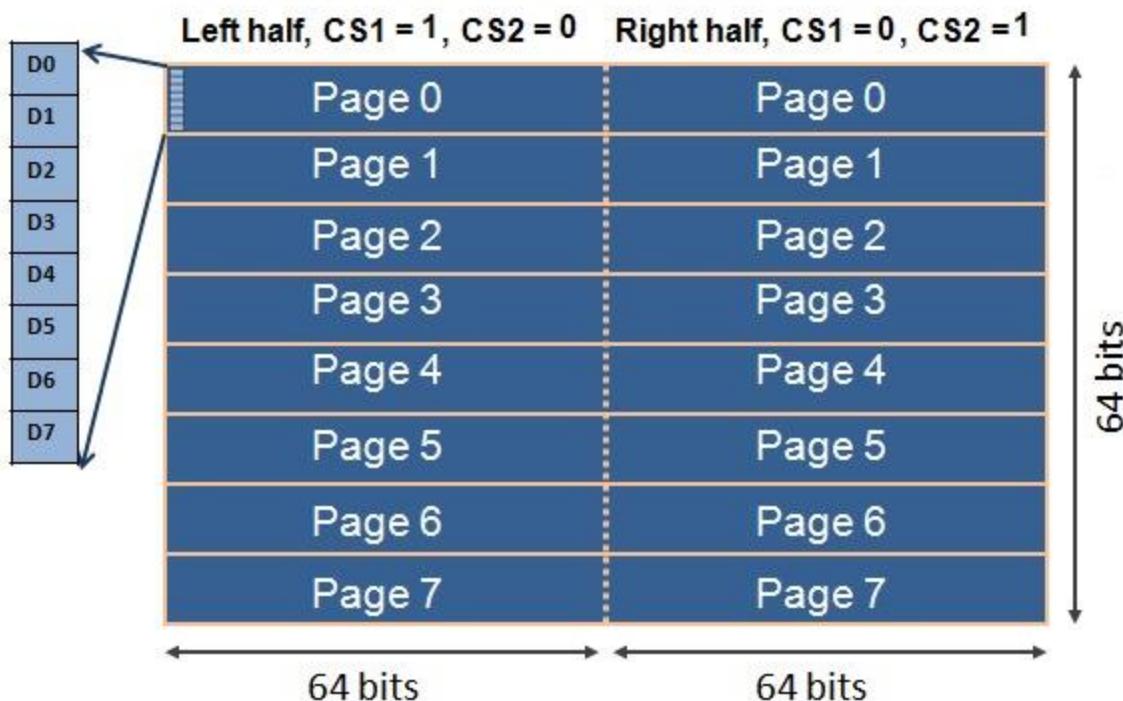
این رجیستر مشخص می‌کند Page0 از کدام سطر شروع شود. با توجه به این‌که ۶۴ سطر داریم ، عدد داخل این رجیستر می‌تواند از ۰xC0 برای سطر صفر تا ۰xff برای سطر ۶۳ باشد. توجه داشته باشید که اگر پس از رسم تصویر ، عدد داخل این رجیستر را تغییر دهید تصویر روی LCD به سمت بالا یا پایین جا به جا می‌شود.

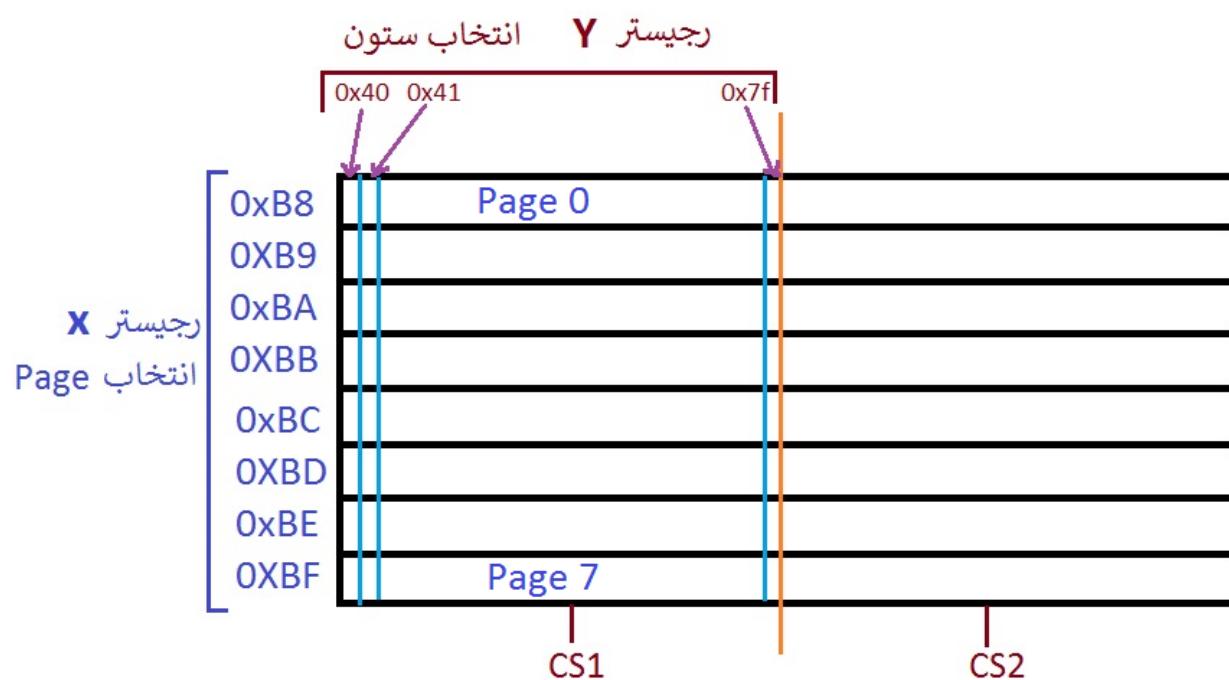
روشن و خاموش کردن نمایش: Display On/Off

با ارسال کد دستوری ۰x3E نمایش خاموش و با ارسال ۰x3F نمایش روشن می‌شود .

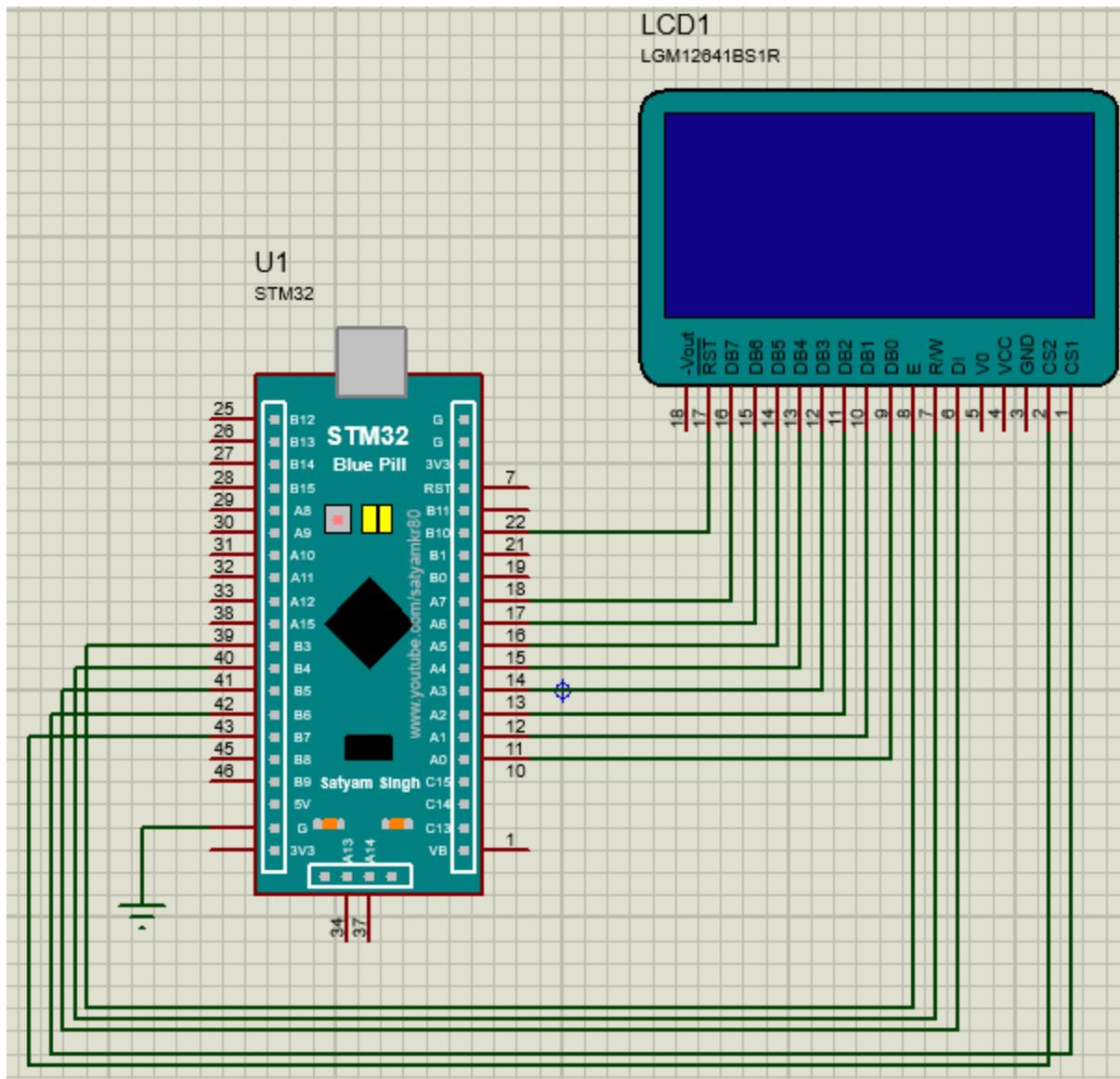
نکته: توجه داشته باشید قبل از ارسال دستور روشن شدن ، هر دو پایه CS1,2 فعال باشد.

در شکل‌های زیر نحوه تقسیم بندی ، آدرس‌دهی و چیدمان دیتا روی صفحه نمایش نشان داده شده است.





مثال : مطابق شکل زیر LCD گرافیکی را به میکرو متصل کرده و تصویر یک مربع با ابعاد 8×8 پیکسل را بر روی آن نمایش دهید.



در تنظیم cube کافی است تمام پایه‌های متصل به LCD را در حالت خروجی قرار دهیم.

برنامه این مثال به صورت زیر است. توجه داشته باشید که برای صفر و یک کردن پایه‌های کنترلی عباراتی بصورت ماکرو تعریف شده است. همچنین تابع `comm` برای اجرای ارسال دستور و تابع `datw` برای ارسال دیتا به LCD تعریف شده‌اند.

```
/* Private define ----- */  
/* USER CODE BEGIN PD */  
  
#define RST_1 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_10,GPIO_PIN_SET)  
#define RST_0 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_10,GPIO_PIN_RESET)  
  
  
#define EN_1 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_3,GPIO_PIN_SET)  
#define EN_0 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_3,GPIO_PIN_RESET)  
  
  
#define RW_1 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_4,GPIO_PIN_SET)  
#define RW_0 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_4,GPIO_PIN_RESET)  
  
  
#define DI_1 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5,GPIO_PIN_SET)  
#define DI_0 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5,GPIO_PIN_RESET)  
  
  
#define CS1_1 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6,GPIO_PIN_SET)  
#define CS1_0 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_6,GPIO_PIN_RESET)  
  
  
#define CS2_1 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_SET)  
#define CS2_0 HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_RESET)
```

```
/* USER CODE END PD */  
/* USER CODE BEGIN PFP */  
  
void comm(char c)  
{  
    HAL_GPIO_WritePin(GPIOA,0xFF,GPIO_PIN_RESET);  
    HAL_GPIO_WritePin(GPIOA,c,GPIO_PIN_SET);  
    DI_0;  
    RW_0;  
    EN_1;  
    EN_0;  
    HAL_Delay(2);  
}  
  
  
void datw(char c)  
{  
    HAL_GPIO_WritePin(GPIOA,0xFF,GPIO_PIN_RESET);  
    HAL_GPIO_WritePin(GPIOA,c,GPIO_PIN_SET);  
    DI_1;  
    RW_0;  
    EN_1;  
    EN_0;  
    for(k=0;k<100;k++);  
    //HAL_Delay(2);
```

{

void glcd_init(void)

{

RST_0;

HAL_Delay(1);

RST_1;

CS1_1;

CS2_1;

comm(0x3f);

CS1_0;

CS2_0;

}

/* USER CODE END PFP */

/* USER CODE BEGIN 2 */

glcd_init();

//-----suqare-----

CS1_1;

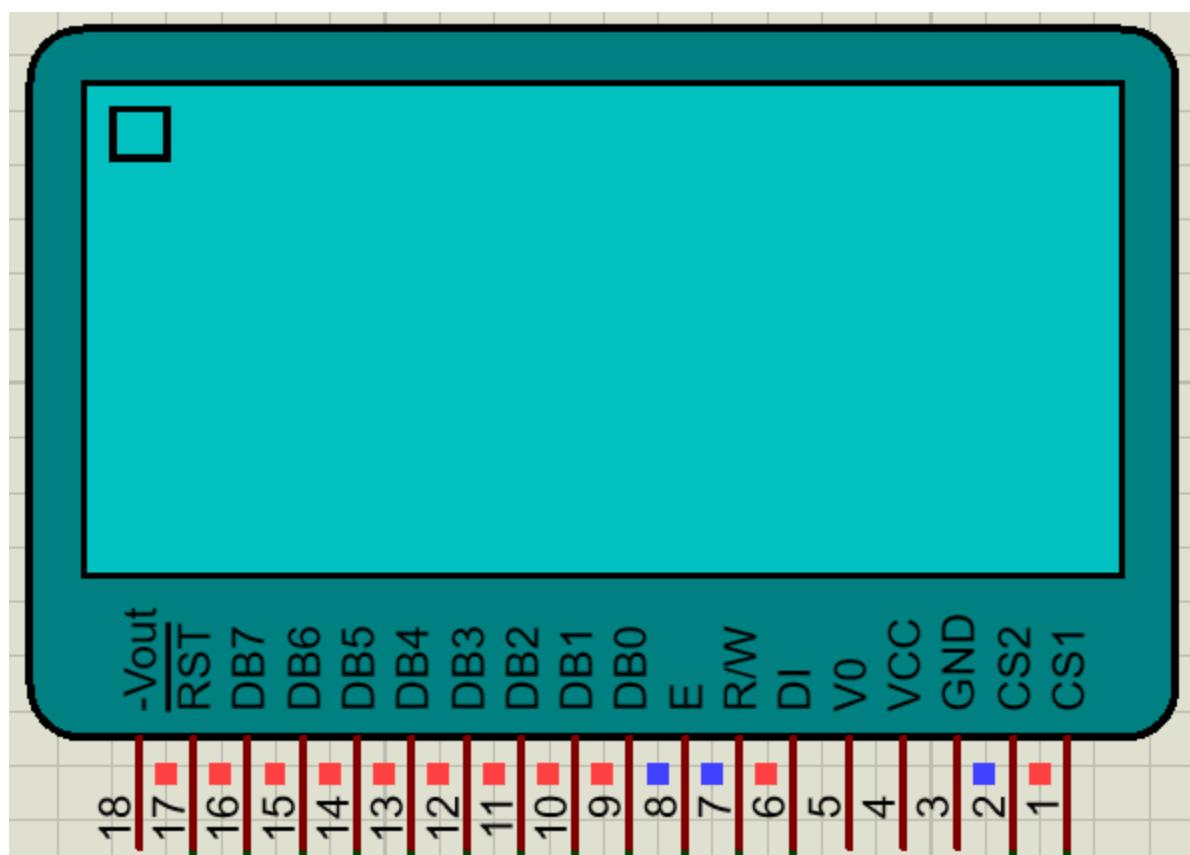
datw(0xFF);

datw(0x81);

datw(0x81);

datw(0x81);

```
datw(0x81);  
datw(0x81);  
datw(0x81);  
datw(0xff);  
/* USER CODE BEGIN 2 */
```



مثال : یک لوزی که هر ضلع آن ۸ پیکسل باشد را طوری رسم کنید که در وسط LCD قرار گیرد نصف آن در CS1 و نیم دیگر در CS2 .

```
/* USER CODE BEGIN 2 */  
//----- Diamond -----  
  
CS1_1;  
  
CS2_0;  
  
comm(0xBB);  
  
comm(0x78);  
  
datw(0x80);  
  
datw(0x40);  
  
datw(0x20);  
  
datw(0x10);  
  
datw(0x08);  
  
datw(0x04);  
  
datw(0x02);  
  
datw(0x01);  
  
  
comm(0xBC);  
  
comm(0x78);  
  
datw(0x01);  
  
datw(0x02);  
  
datw(0x04);  
  
datw(0x08);
```

```
datw(0x10);
```

```
datw(0x20);
```

```
datw(0x40);
```

```
datw(0x80);
```

```
CS1_0;
```

```
CS2_1;
```

```
comm(0xBB);
```

```
comm(0x40);
```

```
datw(0x01);
```

```
datw(0x02);
```

```
datw(0x04);
```

```
datw(0x08);
```

```
datw(0x10);
```

```
datw(0x20);
```

```
datw(0x40);
```

```
datw(0x80);
```

```
comm(0xBC);
```

```
comm(0x40);
```

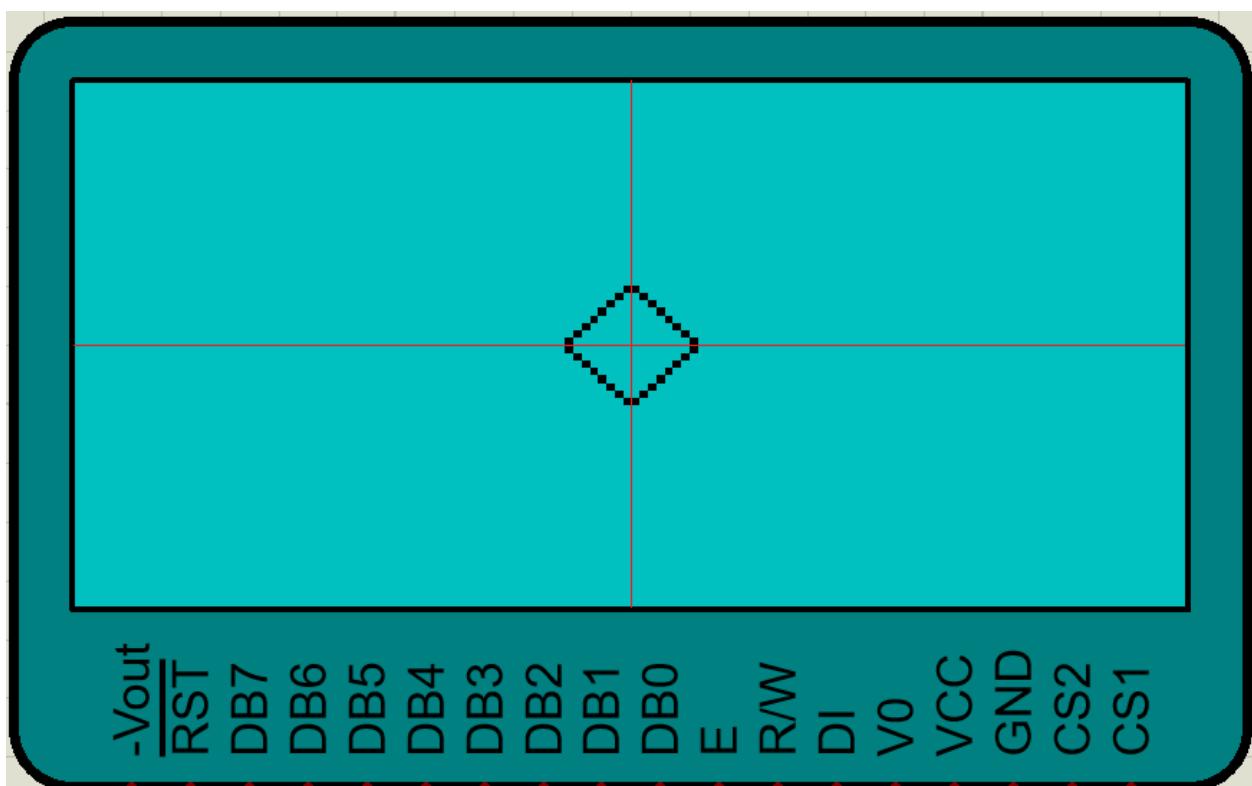
```
datw(0x80);
```

```
datw(0x40);
```

```
datw(0x20);
```

```
datw(0x10);  
datw(0x08);  
datw(0x04);  
datw(0x02);  
datw(0x01);  
/* USER CODE BEGIN 2 */
```

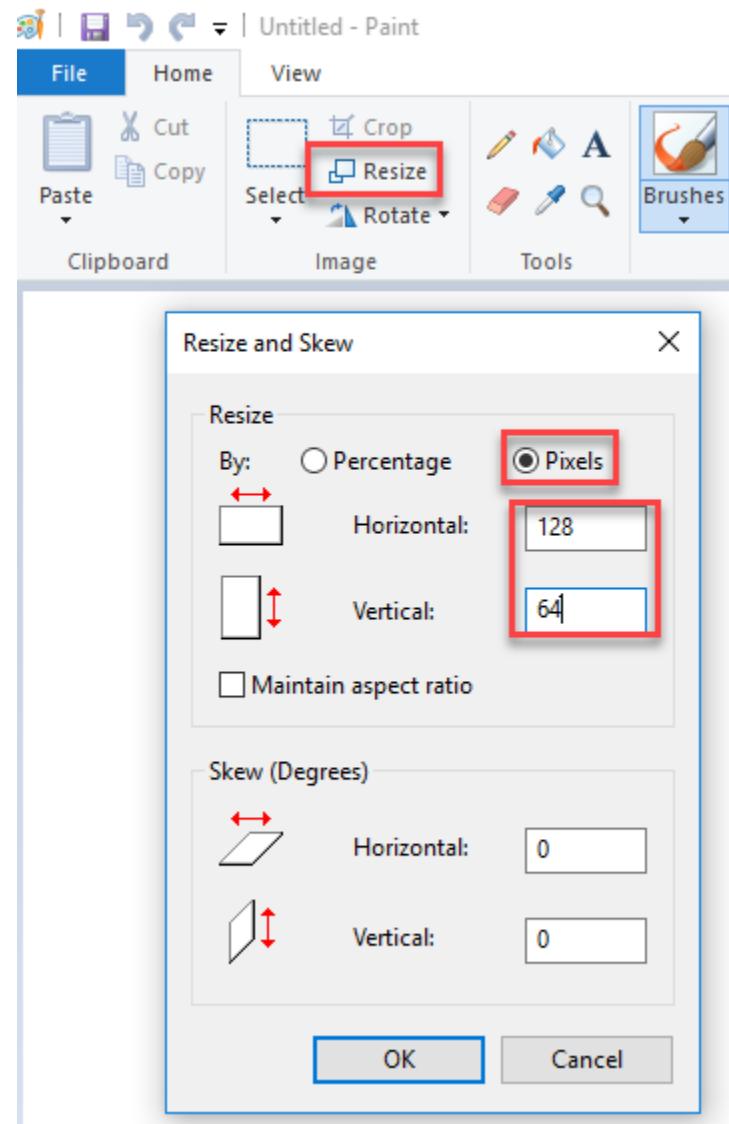
توجه: خطوط قرمز در شکل زیر جهت راهنمایی رسم شده است.



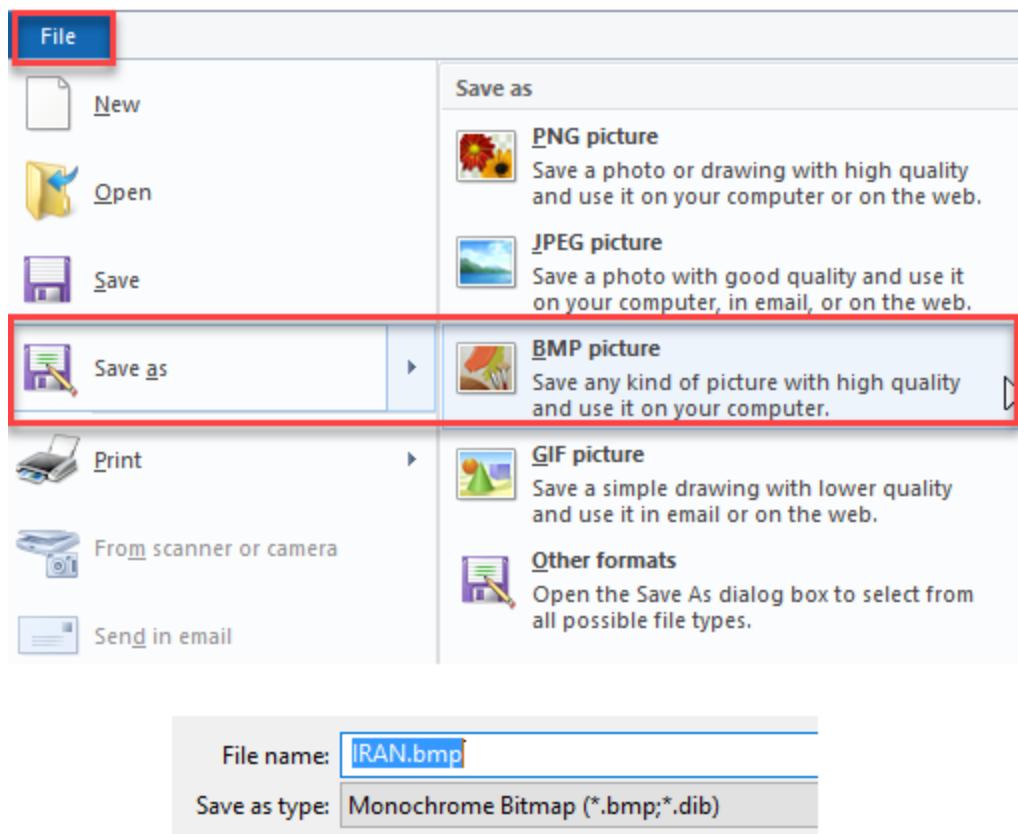
نمایش یک تصویر کامل بر روی LCD :

برای این منظور لازم است با طی مراحل زیر ابتدا تصویر به دیتا تبدیل شود.

- ۱- یک برنامه نقاشی مانند Paint را اجرا کنید.
- ۲- با توجه به تصویر زیر اندازه صفحه نقاشی را روی $128*64$ پیکسل تنظیم کنید.



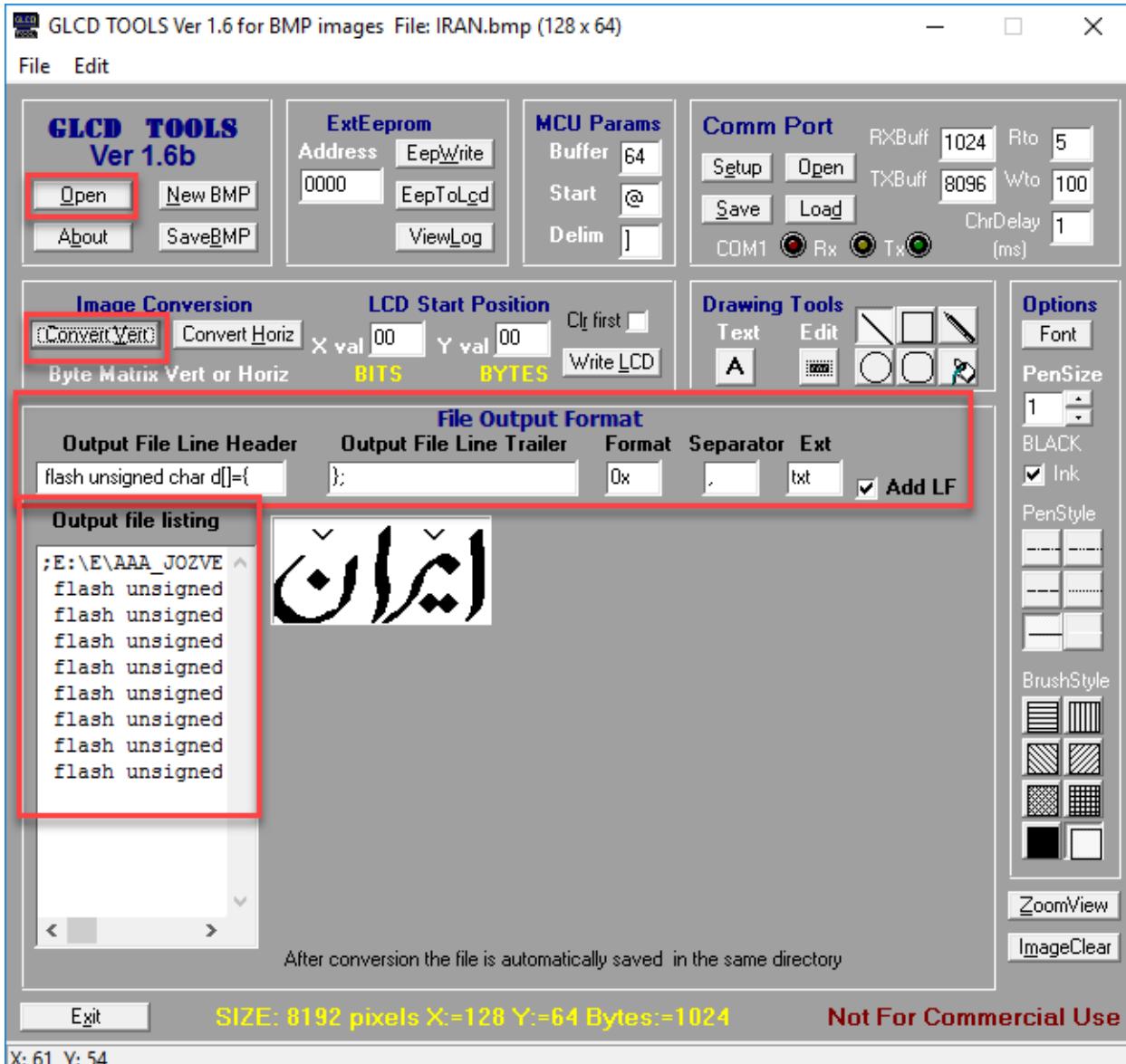
۳- فایل را با نام دلخواه و با فرمت BMP تک رنگ ذخیره کنید.



۴- متن یا نقاشی مورد نظر را در محدوده ۶۴×۱۲۸ پیکسل رسم کنید.



۵- برای تبدیل تصویر به دیتا نرم‌افزارهای متنوعی وجود دارد. برای نمونه ما از نرم‌افزار GLCDtool16b.exe استفاده می‌کنیم.



۶- با استفاده از گزینه Open فایل تصویر را در برنامه بارگذاری می‌کنیم.

۷- در قسمت File Output Format قالب دیتای خروجی را مشخص می‌کنیم. در بخش Header مشخص می‌کنیم که در ابتدای دیتا چه عباراتی قرار بگیرد. در Trailer مشخص می‌کنیم در انتهای هر خط دیتا چه عبارت یا علامتی قرار بگیرد. در بخش Format مشخص می‌کنیم قبل از هر کد دیتا چه علامتی قرار بگیرد. و Separator مشخص می‌کند برای جدا سازی کدها از چه علامتی استفاده شود.

۸- برای تبدیل تصویر به کد باید از بخش Image Conversion یکی از دو گزینه Convert Vert یا Convert Horiz را انتخاب کنیم . اگر دیتا روی LCD بصورت عمودی قرار می‌گیرد، گزینه Vertical و اگر افقی قرار می‌گیرد گزینه Horizontal را کلیک کنید.

۹- دیتای تولید شده را از بخش Output File Listing کپی و به برنامه خود منتقل کنید.

در برنامه زیر با توجه به دیتای تولید شده ، در هشت مرحله دیتای موجود در هر یک از آرایه‌ها را خوانده و در Page مربوط به آن دیتا قرار می‌دهیم. توجه داشته باشید در هر آرایه ۱۲۸ بایت دیتا وجود دارد که ۶۴ بایت آن را در CS1 و ۶۴ بایت بعدی در CS2 قرار می‌گیرد. آخرین حلقه for با متغیر Z باعث می‌شود تصویر به سمت بالا حرکت کند.

```
/* USER CODE BEGIN PV */

unsigned char i,z;

const unsigned char
d0[]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40,0xC0,0x80,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x80,0xC0,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x80,0xC0,0xE0,0xF0,0xF0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x40,0xC0,0x80,
0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xC0,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xC0,0x00,
0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

const unsigned char
d1[]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x03,0x06,0x0C,0x0C,
0x06,0x03,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x
03,0x06,0x0C,0x0C,0x06,0x03,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

```
00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0xFC,0xFE,0xFF,0xFF,0xFF,0xFF,0xF  
F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
```

const unsigned char

const unsigned char

const unsigned char


```
CS1_1;  
CS2_0;  
comm(0xB8);  
comm(0x40);  
for(i=0;i<64;i++) datw(d0[i]);  
CS1_0;  
CS2_1;  
comm(0xB8);  
comm(0x40);  
for(i=64;i<128;i++) datw(d0[i]);  
//-----PAGE1-----  
CS1_1;  
CS2_0;  
comm(0xB9);  
comm(0x40);  
for(i=0;i<64;i++) datw(d1[i]);  
CS1_0;  
CS2_1;  
comm(0xB9);  
comm(0x40);  
for(i=64;i<128;i++) datw(d1[i]);  
//-----PAGE2-----  
CS1_1;
```

```
CS2_0;  
comm(0xBA);  
comm(0x40);  
for(i=0;i<64;i++) datw(d2[i]);  
CS1_0;  
CS2_1;  
comm(0xBA);  
comm(0x40);  
for(i=64;i<128;i++) datw(d2[i]);  
//-----PAGE3-----  
CS1_1;  
CS2_0;  
comm(0xBB);  
comm(0x40);  
for(i=0;i<64;i++) datw(d3[i]);  
CS1_0;  
CS2_1;  
comm(0xBB);  
comm(0x40);  
for(i=64;i<128;i++) datw(d3[i]);  
//-----PAGE4-----  
CS1_1;  
CS2_0;
```

```
comm(0xBC);
comm(0x40);
for(i=0;i<64;i++) datw(d4[i]);
CS1_0;
CS2_1;
comm(0xBC);
comm(0x40);
for(i=64;i<128;i++) datw(d4[i]);
//-----PAGE5-----
CS1_1;
CS2_0;
comm(0xBD);
comm(0x40);
for(i=0;i<64;i++) datw(d5[i]);
CS1_0;
CS2_1;
comm(0xBD);
comm(0x40);
for(i=64;i<128;i++) datw(d5[i]);
//-----PAGE6-----
CS1_1;
CS2_0;
comm(0xBE);
```

```
comm(0x40);

for(i=0;i<64;i++) datw(d6[i]);

CS1_0;

CS2_1;

comm(0xBE);

comm(0x40);

for(i=64;i<128;i++) datw(d6[i]);

//-----PAGE7-----

CS1_1;

CS2_0;

comm(0xBF);

comm(0x40);

for(i=0;i<64;i++) datw(d7[i]);

CS1_0;

CS2_1;

comm(0xBF);

comm(0x40);

for(i=64;i<128;i++) datw(d7[i]);

/* USER CODE END 2 */

/* Infinite loop */

/* USER CODE BEGIN WHILE */

while (1)

{
```

```
/* USER CODE END WHILE */
```

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

```
for(z=0xc0;z<0xff;z++)  
{  
    comm(z);  
    HAL_Delay(100);  
}
```

```
}
```

```
/* USER CODE END 3 */
```



پروگرام کردن میکرو:

در میکروهای ST چهار روش برای پروگرام کردن میکرو پیش بینی شده است .

- ۱ - استفاده از ST Link
- ۲ - استفاده از Boot Loader از طریق UART
- ۳ - استفاده از پایه‌های USB میکرو
- ۴ - استفاده از قابلیت DFU برای پروگرام کردن از طریق USB
- ۵ - استفاده از JTAG