



انجمن علمی دانشجویی  
مکملسی برق  
و کامپیوتر  
دانشگاه تربیت مدرس



## آموزش یادگیری

### میکروکنترلر ARM

### انتشار رایگان

## انتشار رایگان کتاب آموزشی میکروکنترلر ARM

با همت شاخه IEEE دانشگاه تربیت مدرس و انجمن علمی مهندسی  
برق و کامپیوتر دانشگاه تربیت مدرس

### مؤلفین:

مهندس امیر حسن آشنایی دانشجو دکتری کنترل امیر کبیر

مهندس پوریا اسپری دانشجو کارشناسی ارشد امیر کبیر

مهندس احسان حاجی کارشناسی ارشد رازی کرمانشاه

### سرفصل

#### فصل اول: معرفی ARM

- فصل هفتم: مدولاسیون عرض پالس (PMW)
- فصل هشتم: مبدل آنالوگ به دیجیتال (ADC)
- فصل نهم: پروتکل ارتباطی سریال Usart
- فصل دهم: مبدل دیجیتال به آنالوگ (DAC)
- فصل یازدهم: پروتکل (SPI)
- فصل دوم: تنظیمات اولیه و معرفی نرم افزار Cube
- فصل سوم: ساخت پروژه بدون استفاده از Cube
- فصل چهارم: درگاههای ورودی / خروجی (GPIO)
- فصل پنجم: وقفه خارجی External Interrupt
- فصل ششم: تایمر

@tmu\_ieee



ieee\_tmu



tmu.ieee.org.ir



## سخنی با مولف:

در این کتاب به صورت قدم به قدم از ابتدایی ترین حالت ممکن شروع به توضیح مباحث نموده‌ایم، که روند آموزش بدین گونه است که مفاهیم در قالب مثال‌ها و پروژه‌های متنوع بیان گردیده است، بطوریکه دانشجو می‌تواند پس از اتمام هر قسمت، بدون هیچ ابهامی آن را عملیاتی کند و از آن در پروژه‌های پیچیده‌تر به راحتی بهره ببرد. در فصول اول و دوم به معرفی پردازنده ARM و نحوه تنظیمات اولیه آن پرداخته شده و در فصل سوم نیز به مختصر توضیحی درباره ایجاد پروژه بدون استفاده از نرم‌افزار Cube صرفا جهت آشنایی با این سبک از برنامه نویسی پرداخته‌ایم. در باقی فصل‌ها نیز بطور کامل از نرم‌افزار Cube که برنامه نویسی را به مقدار بسیار زیادی راحت و قابل فهم می‌کند پرداخته‌ایم.

## فهرست مطالب

### فصل اول: معرفی ARM

۱-۱ مقدمه

۱-۲ حضور ARM در صنعت گوشی های هوشمند

۱-۳ انواع پردازنده های رایج ARM

۱-۴ انواع هسته های پردازنده های سری Cortex-M

۱-۵ برد های دیسکاوری STM32 (Discovery)

### فصل دوم: تنظیمات اولیه و معرفی نرم افزار

۲-۱ نرم افزار Keil uVision5

۲-۱-۱ امکانات و ویژگی های مجموعه ابزارهای Keil

۲-۱-۲ تنظیمات کامپایلر Keil uVision5 License پس از نصب

۲-۱-۳ اضافه و نصب کردن پک های نرم افزاری قطعات مورد استفاده در کامپایلر

۲-۲ اضافه و نصب کردن پک های نرم افزاری قطعات مورد استفاده در نرم افزار STM32CubeMX

### فصل سوم: ساخت پروژه بدون استفاده از Cube

۳-۱ ساخت پروژه بدون استفاده از cube

۳-۲ شروع برنامه نویسی

### فصل ۴: در گاه های ورودی / خروجی (GPIO) و محیط دیباگ

۴-۱ در گاه های ورودی / خروجی (GPIO)

۴-۱-۱ بررسی ساختار GPIO مطابق با شماتیک مداری آن

۴-۱-۲ جهت ادامه توضیحات نرم افزار STM32CubeMX را باز می کنیم

۴-۱-۳ نوشتن برنامه نویسی در Keil

۴-۱-۴ محیط دیباگ

## فصل ۵: وقفه خارجی External Interrupt

۱-۵ وقفه خارجی (External Interrupt)

## فصل ۶: TIMER

۱-۶ تایمر

## فصل ۷: مدولاسیون عرض پالس (Pulse Width Modulation) PWM

۱-۷ مدولاسیون عرض پالس

۲-۷ طراحی موج سینوسی

۳-۷ طراحی فاصله سنج

## فصل ۸: مبدل آنالوگ به دیجیتال (ADC) Analog to Digital Converter

۱-۸ مبدل آنالوگ به دیجیتال

۱-۱-۸ روش های استفاده از ADC

۱-۱-۸-۲ وقفه تایمیری

۱-۱-۸-۳ Pollforconversion

۱-۱-۸-۴ Intrupt

۱-۱-۸-۵ DMA (روشنهایی)

## فصل ۹: پروتکل ارتباطی سریال Usart

۱-۹ پروتکل ارتباطی سریال

## فصل دهم: مبدل دیجیتال به آنالوگ (DAC) Digital to Analog Converter

۱-۱۰ مبدل دیجیتال به آنالوگ

## فصل یازدهم: پروتکل SPI Serial peripheral interface

۱-۱۱ SPI پروتکل

**فصل اول**

**معرفی ARM**

امروزه با پیشرفت روز افرون تجهیزات و الکترونیکی شدن آن ها، بکارگیری سیستم های یکپارچه رونق زیادی یافته است. به طوری که در اکثر دستگاه های جدید از این سیستم ها استفاده می شود. به عنوان مثال گوشی های همراه و ... اکثراً دارای این تجهیزات الکترونیکی می باشند. با توجه به این موضوع اکثر شرکت ها و کارخانجات الکترونیکی به سمت این سیستم های الکترونیکی روی آورده اند. که این خود باعث ایجاد یک رقابت در بین تولیدکنندگان پردازنده های سرعت بالا شده است.

در این خلال نسل جدید پردازنده های ARM<sup>۱</sup> به بازار معرفی شدند ، که دارای سیستم پردازش ۳۲ بیتی با سرعت پردازش چند مگاهرتز تا چند صد مگاهرتز می باشند . سرعت بالا، قیمت ارزان و حجم کم این پردازنده ها باعث شد که اکثر تولیدکنندگان میکروکنترلرها و پروسسورها مانند ATMEL PHILIPS، ... آن را در لیست محصولات خود قرار دهند. حجم کم پردازنده های ARM باعث شده که اکثر فضای داخلی میکروکنترلرها برای تجهیزات جانبی مانند<sup>۲</sup> Serial, LAN<sup>۳</sup>, USB<sup>۴</sup>, ADC<sup>۵</sup>, DAC<sup>۶</sup> و ... بکار گرفته شود . پردازنده های ARM توسط کمپانی Acorn Computers بر اساس معماری مبتنی بر RISC در دهه ۸۰ میلادی توسعه یافتند. Acorn Computers یک کمپانی بریتانیایی است که در سال ۱۹۷۸ در کمبریج انگلستان آغاز به کار کرد و محصولاتی نیز به بازار فرستاد که از جمله‌ی آن می‌توان به کامپیوتر BBC Micro اشاره کرد. در سال ۱۹۹۸ میلادی این کمپانی بریتانیایی تفکیک شد و زیرمجموعه‌های آن، امروزه به صورت مستقل به فعالیت می‌پردازند. یکی از موفق‌ترین زیرمجموعه‌های آکورن، Advanced RISC Machines نام دارد که بیشتر با عنوان ARM شناخته می‌شود . در ابتدا به صورت مستقل فعالیت می‌کرد، اما کمپانی ژاپنی SoftBank چند سال پیش با پرداخت بیش از ۳۲ میلیارد دلار، این کمپانی بریتانیایی را تصاحب کرد.

<sup>1</sup> processor

<sup>2</sup> Advanced RISC Machines

<sup>3</sup> Local Area Network

<sup>4</sup> Universal Serial Bus

<sup>5</sup> Analog to Digital Converter

<sup>6</sup> Digital to Analog Converter

## ۱-۲ حضور ARM در صنعت گوشی‌های هوشمند

پس از آنکه گوشی‌های هوشمند موج جدیدی در دنیای فناوری ایجاد کردند و الگوی استفاده از گجت‌های دیجیتال را تغییر دادند، پردازنده‌ها نیز شاهد تغییر و تحولات و پاگرفتن معماری جدیدی بودند. با وجود تسلط Intel<sup>۱</sup> و AMD<sup>۲</sup> بر پردازنده‌های مورد استفاده در پی‌سی که مبتنی بر معماری x86 هستند، گوشی‌های هوشمند تو سط پردازنده‌های مبتنی بر معماری ARM قبصه شده‌اند. اهمیت معماری ARM و پردازنده‌های مبتنی بر این معماری به اندازه‌ای افزایش یافته است که مایکروسافت نیز به همکاری با Qualcomm<sup>۳</sup>، بزرگ‌ترین تولیدکننده‌ی تراشه‌های مبتنی بر معماری ARM برای گجت‌های موبایل پرداخته است.

یک کمپانی یا به بیان بهتر یک لبراتوار است که متخصصان آن به طراحی پردازنده مشغول هستند. البته عبارت اختصاری ARM دو مفهوم را شامل می‌شود. در مورد معماری و طراحی پردازنده‌های ARM، این عبارت مخفف Acorn RISC Machines<sup>۴</sup> است، حال آنکه اگر منظور کمپانی توسعه‌دهنده‌ی این معماری باشد، اختصار ARM کوتاه شده‌ی عبارت Advances RISC Machines است. به هیچ‌وجه دستی در تولید تراشه ندارد و تنها معماری ARM را طراحی می‌کند، حال آنکه کمپانی‌های نظری کوالکام، اپل و سامسونگ با دریافت گواهی استفاده از معماری آرم، تراشه‌های اختصاصی خود را مبتنی بر این معماری توسعه می‌دهند. البته این سه کمپانی شاخص‌ترین نام‌هایی هستند که گواهی معماری آرم را در پردازنده‌های خود استفاده می‌کنند و تعداد تولیدکنندگان پردازنده با استفاده از این معماری بالا است. اغلب گجت‌های الکترونیکی کوچک که از وجود باتری برای تأمین انرژی استفاده می‌کنند، از پردازنده‌های مبتنی بر آرم در واحد پردازشی خود بهره می‌برند. همان‌طور که اشاره کردیم؛ ARM پردازنده‌های خود را برا اساس دستورات RISC توسعه می‌دهد، اما RISC چیست؟ RISC مخفف عبارت Reduced Instruction Set Computing است. برخلاف پردازنده‌های ARM که از این معماری استفاده می‌کنند، پردازنده‌های اینتل و ای‌ام‌دی که قدرت پردازشی در لپ‌تاپ و پی‌سی شما را تأمین می‌کنند، از معماری CISC<sup>۵</sup> یا Complex Instruction Set Computing

<sup>1</sup> Gajet

<sup>2</sup> Integrated Electronics

<sup>3</sup> Advanced Micro Devices

<sup>4</sup> Reduced Instruction Set Computer

<sup>5</sup> Complex Instruction Set Computer

استفاده می‌کند. دو معماری RISC و CISC برای استفاده در کاربردهای متفاوت طراحی شده‌اند. یک پردازنده‌ی مبتنی بر معماری RISC برای این منظور طراحی شده است تا تعداد دستورات ارسالی به پردازنده از سوی برنامه در حال اجرا کاهش یابد. در واقع مجموعه‌ی دستورات مورد استفاده در معماری RISC بسیار پایین‌تر است. با توجه به اینکه تعداد دستورات ارسالی در معماری RISC کاهش پیدا کرده، فرکانس پردازشی بالا است و پردازنده می‌تواند در هر ثانیه دستورات بیشتری در مقایسه با CISC اجرا کند.

زمانی که مجموعه‌ی دستورات اجرایی توسط پردازنده کاهش پیدا کند، پیچیدگی پردازنده نیز کاهش می‌یابد و می‌توان مدار تراشه را به شکل ساده‌تری طراحی کرد. پردازنده‌های RISC دارای ترانزیستورهای کمتری هستند که همین موضوع منجر به کاهش انرژی مصرفی توسط پردازنده می‌شود. سادگی طراحی پردازنده در کنار کاهش تعداد ترانزیستورها نتیجه‌ای جز کاهش سایز تراشه ندارد. سایز تراشه به سطح مقطعی اطلاق می‌شود که روی ویفر سیلیکونی برای ساخت یک پردازنده تخصیص داده می‌شود. نتیجه کاهش سایز، امکان اضافه کردن کامپوننت‌های بیشتر روی پردازنده با اتصالات کمتر است، از این‌رو پردازنده‌های ARM کوچک‌تر هستند و انرژی کمتری مصرف می‌کنند.

پردازنده‌های سریع، کوچک و کم‌صرف بهترین گزینه برای استفاده در گوشی‌های هوشمند هستند. هرچند گوشی‌های هوشمند این روزها قدرت پردازشی بالایی دارند، اما در یک گوشی هوشمند هیچ‌گاه بار پردازشی از طریق چند صد تردد روی هسته‌های مختلف اعمال نمی‌شود. سیستم عامل و اپلیکیشن‌های توسعه‌یافته برای گوشی‌های هوشمند نیز به منظور کاهش دستورات ارسالی به پردازنده بهینه شده‌اند تا بهترین نتیجه در زمان به کارگیری تراشه‌های ARM حاصل شود.

البته تمام مواردی که در بالا به آن‌ها اشاره کردیم به این معنی نیست که پردازنده‌های ARM قدرت پردازشی محدودی دارند. معماری ARM امروزه امکان استفاده از طراحی ۳۲ و ۶۴ بیتی را در اختیار کاربران قرار می‌دهد. همچنین با یک به مجموعه‌ای از قابلیت‌های دیگر اشاره کرد که علاوه بر گوشی‌های هوشمند، پردازنده‌های ARM را برای استفاده در کاربردهای دیگر نظری ابرایانه‌ها مناسب می‌کند.

نسبت عملکرد به ازای هر وات از انرژی مصرفی در پردازنده‌های ARM بسیار قابل قبول است. در صورتی که نرم‌افزار توسعه‌یافته برای پردازنده‌های ARM بهینه باشد، این پردازنده‌ها می‌توانند عملکرد بهتری در مقایسه با معماری x86 ارائه کنند، از این‌رو اهمیت پردازنده‌های ARM در کاربردهایی نظیر به کارگیری آن‌ها در سرورها و ابرایانه‌ها بیش از پیش پرنگ می‌شود.

خروجی پردازشی مورد انتظار از ۲۴ هسته‌ی پردازشی ۸۶x را می‌توان از چند صد هسته‌ی پردازشی کم‌صرف و کوچک مبتنی بر معماری ARM دریافت کرد. هسته‌های ۸۶x قدرت پردازشی مورد نیاز را تنها از چند هسته‌ی پردازشی و چندین ترد به دست خواهند آورد، حال آنکه در معماری ARM، وظیفه پردازشی روی چندین هسته‌ی پردازشی کم‌صرف با ظرفیت پایین‌تر تقسیم می‌شود. هر چند تعداد هسته‌های ARM بیشتر است، اما این هسته‌ها در مقایسه با ۲۴ هسته‌ی ۸۶x نیاز به انرژی کمتری دارند. بدین منظور با استفاده از پتانسیل هسته‌های ARM می‌توان قدرت پردازشی را بدون نیاز به بالابردن انرژی مصرفی، افزایش داد و نرم‌افزاری را که باید اجرا شود، برای معماری ARM بهینه کرد.

### ۱-۳- انواع پردازنده‌های رایج ARM

**پردازنده ARM7TDMI** : رایج‌ترین پردازنده ۳۲‌بیتی با معماری RISC در جهان است. با توجه به اندازه کوچک، کارایی بالا و قیمت و مصرف توان پایین استفاده از آن در سیستم‌های توسعه یافته و قابل حمل بسیار رایج است.

پردازنده ARM7TDMI-S : این پردازنده نسخه قابل سنتز ARM7TDMI می‌باشد که معماری آن توسط Verilog و VHDL بیان شده است و می‌توان این هسته را در کنار سایر بلوک‌های سیستم روی یک تراشه پیاده سازی کرد.

**پردازنده ARM926EJ-S** : این پردازنده دارای پنج مرحله سیستم خط لوله است که یک ضرب کننده برای پردازش سیگنال DSP می‌باشد. این پردازنده همچنین دارای حافظه نهان دستورالعمل و داده و MMU هستند همچنین از پردازنده‌های محبوب ARM9 می‌باشند که از سیستم عامل‌های لینوکس، ویندوز، سیمبیان پشتیبانی می‌کنند.

**پردازنده Cortex-M** : نسل هفتم پردازنده‌های ARM می‌باشند و دارای توان عملیاتی بالا و مصرف توان پایین هستند (در این کتاب این سری از پردازنده‌ها مورد استفاده قرار گرفته است).

میکروکنترلرهای ARM توسط شرکت‌های مختلفی از جمله NXP، Atmel، STM، TI و شرکت‌های مختلف دیگر طراحی و ساخته می‌شود که از میان خانواده‌هایی که بر مبنای پردازنده ARM7 تولید می‌شوند می‌توان به میکروهای سری LPC2XXX محصول شرکت NXP، میکروهای سری AT91SAM7 شرکت ATMEL و میکروهای سری STR7 شرکت STM اشاره کرد. پردازنده‌های سری Cortex-M در نسل هفتم پردازنده معرفی شدند و معماری آن‌ها به منظور استفاده در کنترلرهای بلاذرنگ بهینه سازی شده است. پردازنده ARM

های مختلف این خانواده می توانند به عنوان جایگزین مناسبی جهت پردازنده های مختلف در گستره وسیعی از کاربرد ها مورد استفاده واقع شوند.

## ۱-۴ انواع هسته های پردازنده های سری Cortex-M

### • Cortex-M0 هسته

این هسته یکی از کم مصرف ترین پردازنده های ارائه شده توسط شرکت ARM است که در عین مصرف توان اندک، کارایی بسیار بالایی را ارائه می کند. قابلیت اجرای دستور های Thumb-2 این امکان را فراهم می آورد که در عین کاهش حجم برنامه کارایی قابل قبولی ارائه شود.

میکروکنترلر های سری Cortex-M هستند. این میکروکنترلر ها در کاربرد هایی از قبیل سنسور های بدون سیم و سیستم های ریموت که منبع تغذیه مورد استفاده به صورت باتری بوده و میزان توان مصرفی آن بسیار با اهمیت است، مورد استفاده قرار می گیرند. همچنین با توجه به قیمت بسیار اندک و معناری ۳۲ بیتی می تواند به عنوان جایگزینی قدرتمند جهت میکروکنترلر های ۸ و ۱۶ بیتی مورد استفاده قرار گیرد.

### • Cortex-M3 هسته

این هسته یکی از پر کاربرد ترین پردازنده های ارائه شده توسط شرکت ARM است که به طور گستردگی در معناری بسیاری از میکروکنترلر ها مورد استفاده قرار می گیرد و به دلیل مزایای متعدد در مقایسه با هسته ARM7 در معناری نسل جدید میکروکنترلر ها مورد استفاده واقع می شود.

از میان خانواده معروف میکروکنترلر ها که بر اساس معناری این هسته ارائه شده اند می توان به موارد زیر اشاره کرد:

– میکروکنترلر های خانواده LPC43xx محصول شرکت NXP

– میکروکنترلر های خانواده LPC18xx محصول شرکت NXP

– میکروکنترلر های خانواده LPC17xx محصول شرکت NXP

– میکروکنترلر های خانواده LPC13xx محصول شرکت NXP

میکروکنترلر های خانواده STM32 محصول شرکت ST

میکروکنترلر های خانواده SAM3U و SAM3S محصول شرکت Atmel

میکروکنترلر های خانواده Stellaris محصول شرکت Texas

با توجه به ویژگی های منحصر به فرد پردازنده Cortex-M3، علاوه بر میکروکنترلر ها، در معماری سایر تراشه های جدید امروزی مورد استفاده قرار می گیرد که در این میان می توان به موارد زیر اشاره کرد:

Actel Smart Fusion\_

Cypress PSOC 5\_

تراشه های خانواده Smart Fusion ترکیبی از<sup>۱</sup> FPGA و مدارات آنالوگ برنامه پذیر هستند که به هسته پردازنده Cortex-M3 متصل شده اند.

#### • Cortex-M4 هسته

این هسته به منظور پیاده سازی الگوریتم های پردازشی از قبیل فیلتر های دیجیتال،<sup>۲</sup> FFT و ضرب ماتریس که در کاربردهای سنتز و فشرده سازی صدا یا کنترل دور موتور کاربرد دارد مورد استفاده قرار می گیرد و می تواند به عنوان جایگزینی مناسب جهت تراشه هایی از قبیل پردازنده های خانواده TI C2000 و Microchip dsPIC با مصرف توان بسیار کم مورد استفاده واقع شود. این خانواده از تراشه ها تحت نام کلی DSC مورد بحث قرار می گیرند که در کاربردهای ترکیبی کنترل و پردازش سیگنال مورد استفاده قرار می گیرند. از مشخصات این هسته استفاده از واحد های سخت افزاری ویژه جهت اجرای دستور های MAC و SIMD در یک سیکل کاری می باشد که باعث تسريع در اجرای الگوریتم های پردازش سیگنال خواهد شد.

#### • Cortex-M7 هسته

این هسته از عملکرد بسیار مطلوبی برخوردار است به عبارتی بهترین میکرویی است که شرکت ST توانسته تا به حال روانه بازار کند. از بارزترین مشخصات این سری از میکروکنترلر های STM32 می توان به حافظه فلاش ۲ مگابایتی و ماکریم کلاک ۴۰۰ مگا هرتزی آن اشاره کرد که در سال ۲۰۱۷ در دسترس عموم قرار گرفته است.

<sup>۱</sup> Field Programmable Gate Array

<sup>۲</sup> Fast Fourier Transform

## ۵-۱ برد های دیسکاوری STM32 : (Discovery) STM32

شرکت ST میکروالکترونیک یک راه سریع و آسان برای مهندسانی که بخواهند پروژه های مختلف را اجرا و برای ارزیابی تراشه میکروکنترلر داشته باشند ، ساخته شده است. این برد بسیار از ران بوده و به راحتی در بازار در دسترس است. در واقع یک برد آموزشی برای آشنایی و سهولت در استفاده از میکروکنترلر STM32 است. هر برد دیسکاوری شامل یک پروگرم ST-LINK برای برنامه نویسی و اشکال زدایی (Debugging) از طریق یک پورت USB در خود می باشد. ولتاژ مورد استفاده برای هر برد ۵ ولت از طریق کابل USB می باشد و یا یک منبع تغذیه خارجی ۵ ولتی ارائه شده است که آنها را می توان به عنوان منابع توان خروجی ۳.۳ یا ۵ ولتی استفاده کرد (در حال حاضر باید کمتر از ۱۰۰ میلی آمپر باشد). تمام بردها دیسکاوری همچنین شامل یک تنظیم کننده ولتاژ، کلید ریست، کلید کاربر (روشن و خاموش کردن)، چندین LED، هدر SWD در بالای برد و ردیف های پین هدر در پایین برد وجود دارد.

از جمله امکانات این برد می توان به موارد زیر اشاره کرد:

– مجهز به پروگرامر و دیباگر st-link مونتاژ شده روی برد

– دو عدد چیپ ST MEMS

– شتاب سنج ، میکروفون دیجیتال و ژیروسکوپ

– یک مبدل دیجیتال به آنالوگ صدا همراه با تقویت کننده کلاس D که به کمک میکروفون دیجیتال می توانید اقدام به ثبت و پردازش صدا کنید. تغذیه برد نیز ۵ ولت است که در خروجی علاوه بر ۵ ولت ولتاژ ۳ ولت نیز به منظور توسعه و استفاده از برد در مدارات مختلف درنظر گرفته شده است.

– ال ای دی ، کلیدهای مورد نیاز

– پایه های ورودی و خروجی

## **فصل دوم**

### **تنظیمات اولیه و معرفی نرم افزار**

## ۱-۲ نرم افزار Keil uVision5

شرکت Keil یکی از تولیدکنندگان قدیمی و توسعه دهنده کان ابزارها و نرم افزارهای مربوط به صنایع الکترونیکی و به شکل خاص، تولید ابزارهای کامپایل، عیب یابی و توسعه برای انواع میکروکنترلرها می باشد. محصولات این شرکت به عنوان مازول یا کامپوننت در محیط یک پارچه برنامه نویسی μVision IDE نصب می شوند. در این محیط توسعه، امکانات کاملی از قبیل ویرایشگر کد، عیب یابی و اشکال زدایی برنامه، شبیه سازی کامل و ... فراهم آورده شده است. نرم افزار Keil MDK-ARM محصول اصلی این شرکت است که را شامل می شود و میکروکنترلرهای دیگری مثل C251، C166، C51 به عنوان مازول نصب شده و قابل استفاده خواهند بود.

## ۱-۱ امکانات و ویژگی های مجموعه ابزارهای Keil

- محیط توسعه یک پارچه برای انجام انواع عملیات کامپایل، شبیه سازی، اشکال زدایی و ...

- پشتیبانی از خانواده های پردازنده های Cortex-M، Cortex-R، ARM، 8051، C166، C251 و ...

- برنامه نویسی پیشرفته C و C++ در پردازنده های ARM

- امکان تجزیه و تحلیل پیشرفته

- دارای بانک کاملی از مثال ها و نمونه ها جهت آشنایی کاربر

- برخورداری از سیستم عامل اختصاصی RTX با قابلیت پردازش و رایانش بی درنگ یا Real-Time

- وجود کتابخانه ی گسترده ای از GUI ها جهت نوشتن نرم افزارهای گرافیکی

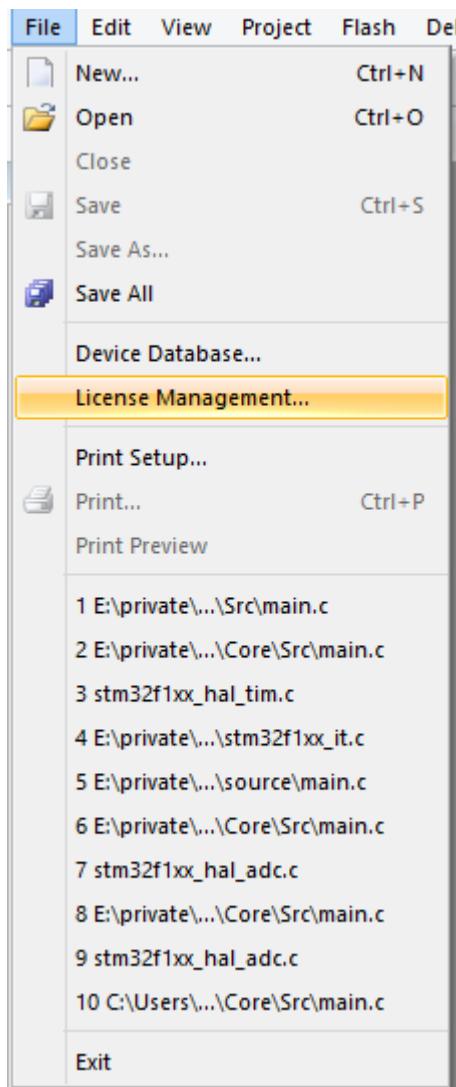
- وجود مجموعه ابزارها و کلاس های برنامه نویسی برای کار با شبکه های TCP/IP و نیز دستگاه های USB

- جهت دریافت کامپوننت ها و بسته های اضافی اینجا، دریافت Application Notes ها اینجا و مثال ها، سورس کدها، پروژه های نمونه و ... اینجا را بینید.

## ۲-۱-۲ تنظیمات License کامپایلر Keil uVision5 پس از نصب:

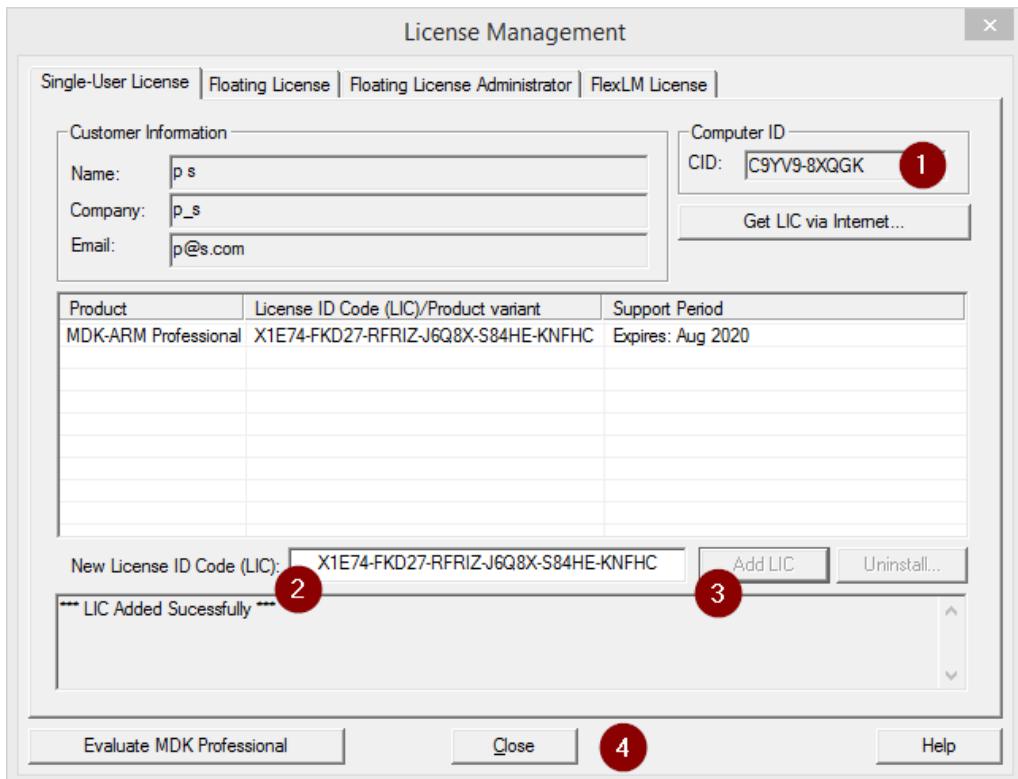
۱- برنامه را باز کنید.

(۱-۲) را انتخاب کنید License Management -۲



شکل ۱-۲

۳- عبارت مقابل CID را کپی کنید (شکل ۲-۲).



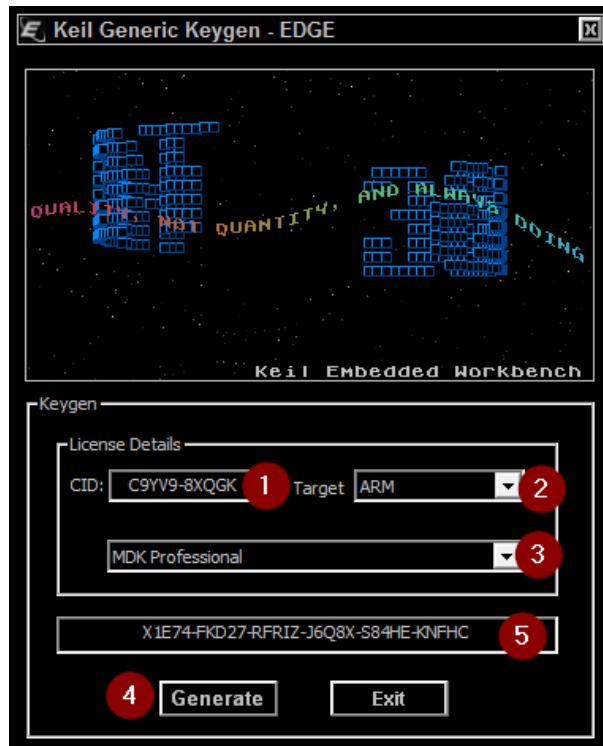
۲-۲ شکل

.(۳-۲) Keygen را باز کنید (شکل ۴)

Name	Date modified	Type	Size
Keygen 1_DownLoadLy.iR	8/28/2014 7:51 AM	Application	
Keygen 2_DownLoadLy.iR	8/28/2014 7:51 AM	Application	
mdk522_DownLoadLy.iR	12/17/2016 1:04 PM	Application	696,2

۳-۲ شکل

-۵ CID کپی شده در مرحله سوم را در شماره یک Paste کرده (شکل ۴) و گزینه ۲ و ۳ را نیز تنظیم نمایید، سپس Generate را انتخاب و کد تولید شده در گزینه ۵ را copy کنید (شکل ۴) و در گزینه ۲ (شکل ۲-۲) از شکل ۳ از Add LIC paste کنید سپس Close (گزینه ۲-۲) انتخاب و Close را بزنید.



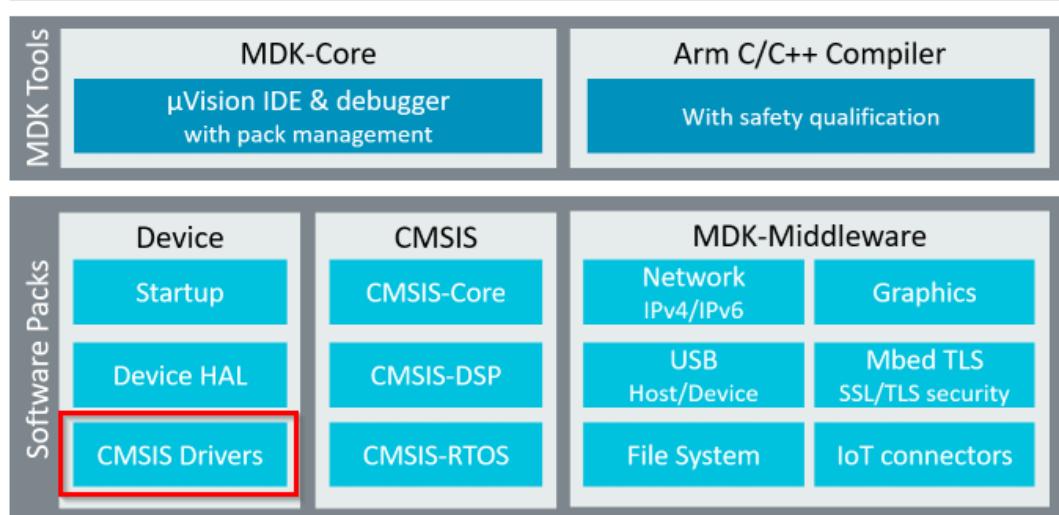
شکل ۴-۲

۱-۳-۲ اضافه و نصب کردن پک‌های نرم‌افزاری قطعات مورد استفاده در کامپایلر keil\_V5:

### الف- روش اول

۱- به آدرس www2.keil.com/MDK5 رفته و CMSIS Driver را انتخاب می‌کنیم(شکل ۵).

#### Product Components



(شکل ۵)

## ۶) DeviceList را انتخاب کرده (شکل ۶).

Device support for ARM Cortex-M based microcontrollers relies on CMSIS which is a common industry standard. It is straightforward and can be implemented by the chip vendor to be used by various toolchains. Currently, Atollic, IAR, and Keil use CMSIS-Pack technology for device support. A plug-in for Eclipse is available for custom toolchains.

CMSIS-Pack describes a delivery mechanism for software components, as well as device and board support. The typical content of a device family pack (DFP) is:

(شکل ۶)

۳- در اینجا لیست تمام شرکت‌ها بالا می‌آید لذا با توجه موضوع مورد بحث این کتاب گزینه ۱ را انتخاب کرده (شکل ۷) و سپس سری مورد نیاز را دانلود می‌کنیم (شکل ۸).

- Silicon Laboratories Devices 1855
- Sinowalth Devices 1
- SONiX Devices 60
- **STMicroelectronics** Devices 1424
- Texas Instruments Devices 350
- Toshiba Devices 232
- Unisoc Devices 1
- XMC Devices 2
- Zilog Devices 7

Products	Downloads	Support	Contact
Development Tools	Hardware & Collateral	MDK-Arm	Knowledgebase
Arm	ULINK Debug Adaptors	C51	Discussion Forum
C166	Evaluation Boards	C166	Product Manuals
C51	Product Brochures	C251	Application Notes
C251	Device Database	File downloads	Distributors
			Request a Quote
			Sales Contacts

(شکل ۷)

▼ STMicroelectronics

➤ STBlueNRG Series

➤ STBlueNRG-1 Series

➤ STBlueNRG-2 Series

➤ STM32F0 Series

➤ STM32F1 Series

➤ STM32F2 Series

➤ STM32F3 Series

➤ STM32F4 Series

➤ STM32F7 Series

➤ STM32G0 Series

➤ STM32G4 Series

➤ STM32H7 Series

➤ STM32L0 Series

➤ STM32L1 Series

➤ STM32L4 Series

➤ STM32L5 Series

➤ STM32MP1 Series

➤ STM32W1 Series

➤ STM32WB Series

Devices 1424

Cortex-M0 Devices 1

Cortex-M0 Devices 1

Cortex-M0 Devices 1

Cortex-M0 Devices 111

Cortex-M3 Devices 95

Cortex-M3 Devices 46

Cortex-M4 Devices 90

Cortex-M4 Devices 205

Cortex-M7 Devices 115

Cortex-M0+ Devices 81

Cortex-M4 Devices 118

Cortex-M4 Devices 72

Cortex-M0+ Devices 152

Cortex-M3 Devices 81

Cortex-M4 Devices 197

Cortex-M33 Devices 36

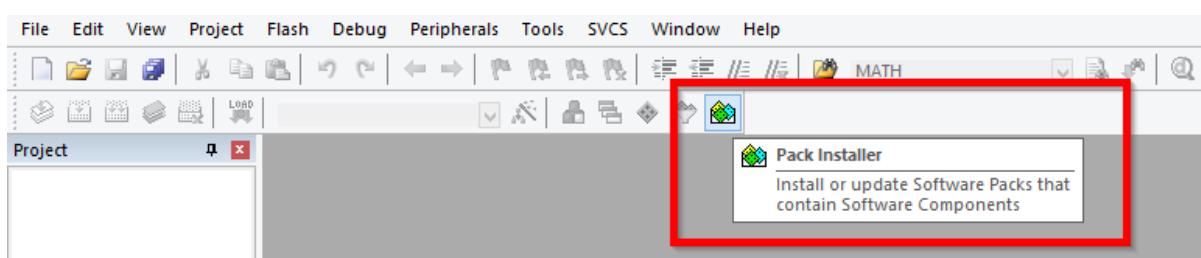
Cortex-M4 Devices 8

Cortex-M3 Devices 5

Cortex-M4 Devices 9

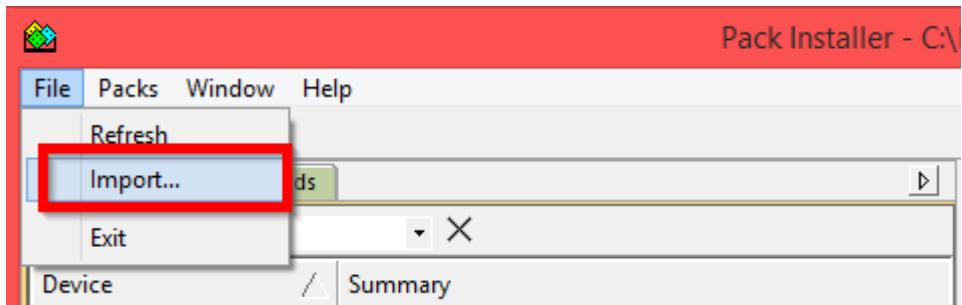
(شکل ۸)

۴- کامپایلر keil\_V5 را باز کرده و Pack Installer انتخاب می‌کنیم (گزینه ۱ از شکل ۹).



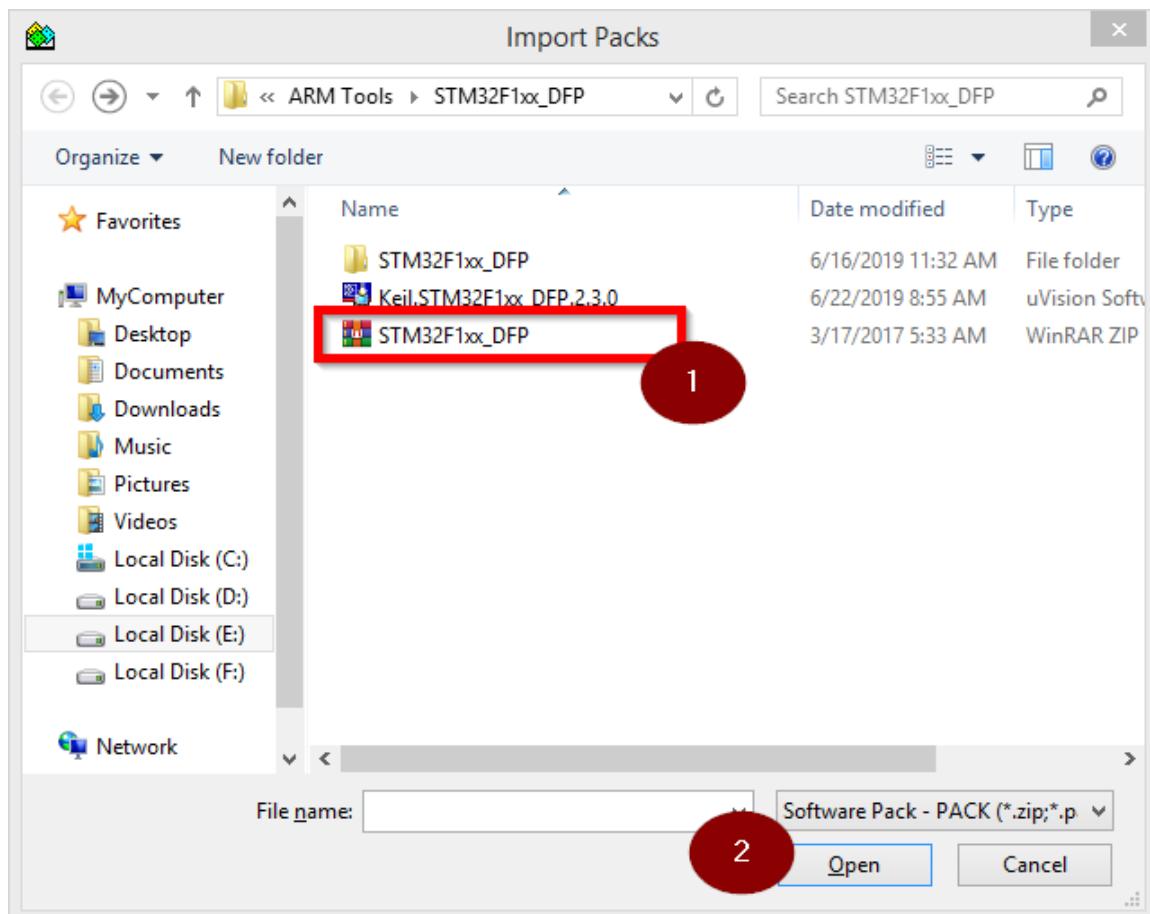
(شکل ۹)

۵- Import را انتخاب می کنیم (گزینه ۲ شکل ۱۰).



(شکل ۱۰)

۶- فایل دانلود شده را انتخاب نموده و گزینه open را می زنیم(شکل ۱۱).

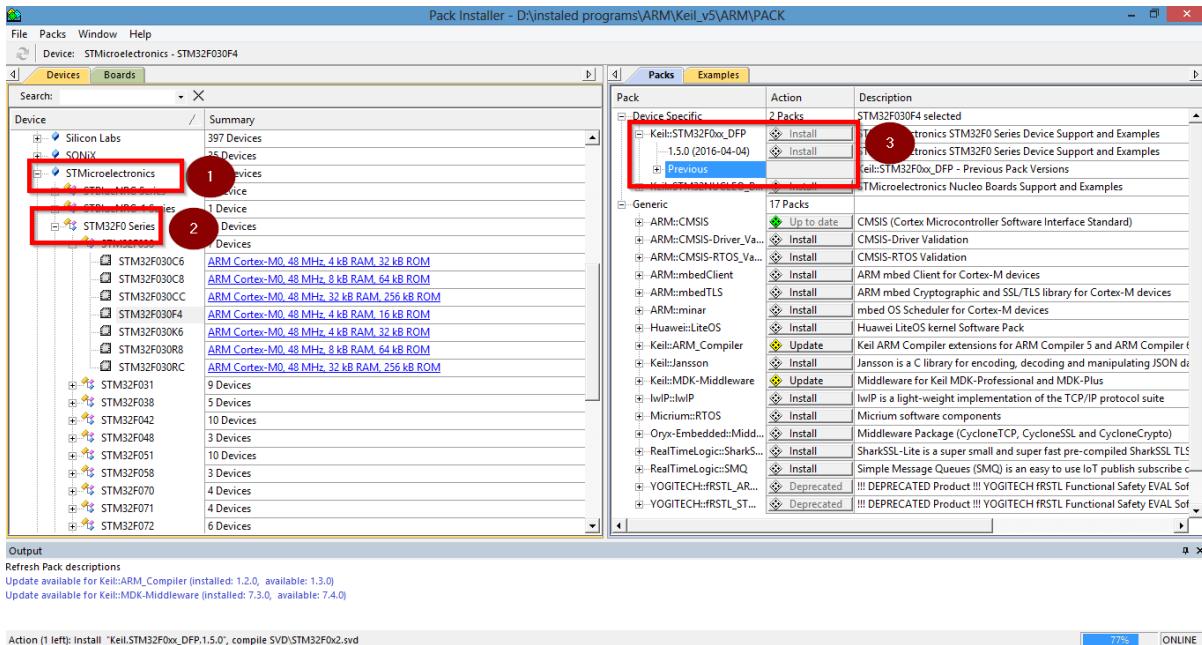


(شکل ۱۱)

## ب- روش دوم:

۱- در قسمت بعدی که درباره Cube توضیح داده می شود، پس از Generate کردن پروژه در آن نرم افزار به صورت خودکار باز می شود و در صورتی که کتابخانه میکرو از قبل اضافه و نصب شده باشد

مشکلی پیش نمی آید ولی در صورتی که کتابخانه میکرو از قبل اضافه نشده باشد Pack Installer باز می شود (شکل ۱۲)، ابتدا شرکت ST (گزینه ۱) سپس سری IC موردنظر (گزینه ۲) را انتخاب می کنیم و در نهایت در قسمت ۳ آن را نصب می کنیم.

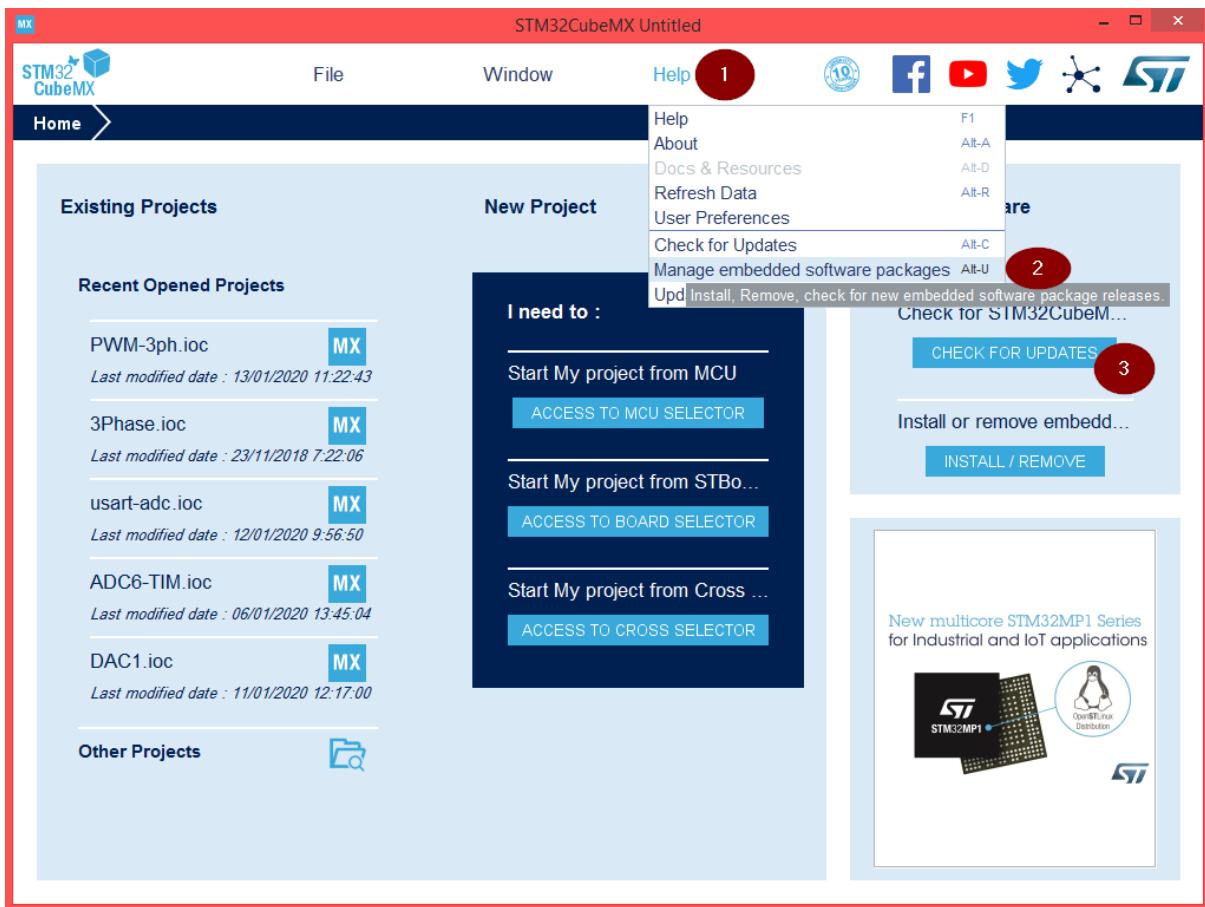


(۱۲) شکل

## ۲-۲ اضافه و نصب کردن پک‌های نرم‌افزاری قطعات مورد استفاده در

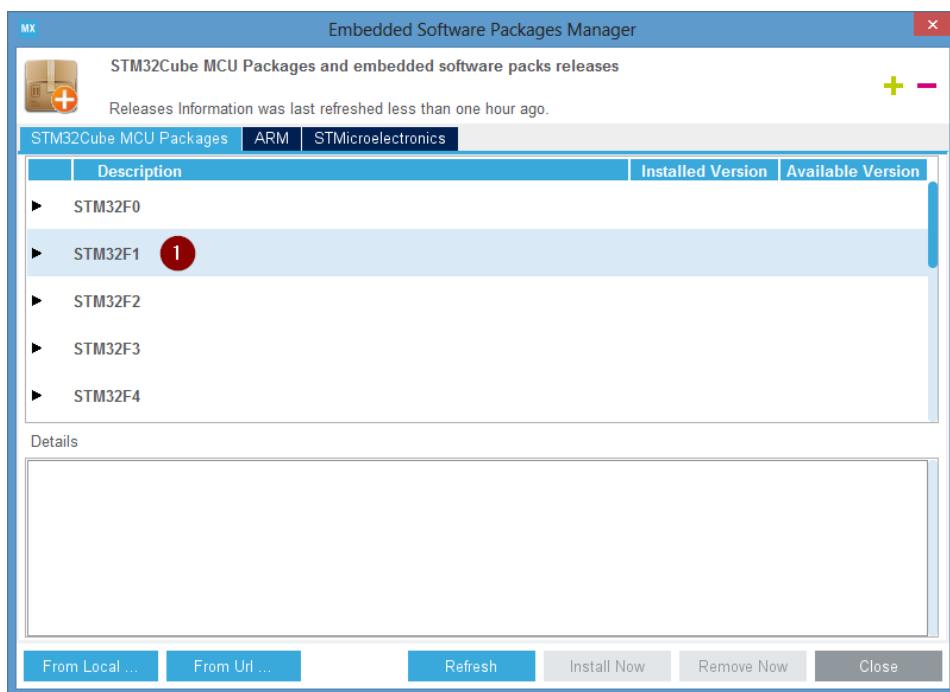
### نرم‌افزار: STM32CubeMX

- ۱ Help را انتخاب نموده سپس گزینه ۲ را انتخاب می کنیم (شکل ۱۳).



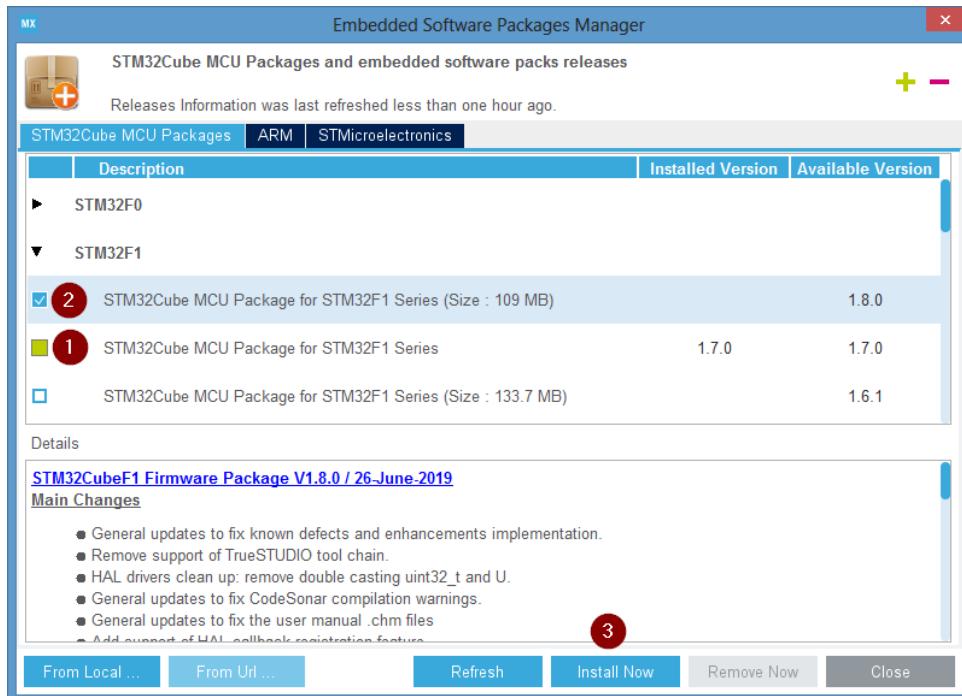
(۱۳) (شکل)

۲- در اینجا تمام سری های مورد نیاز موجود می باشد، بطور مثال STM32F1 را انتخاب کنید(شکل ۱۴).

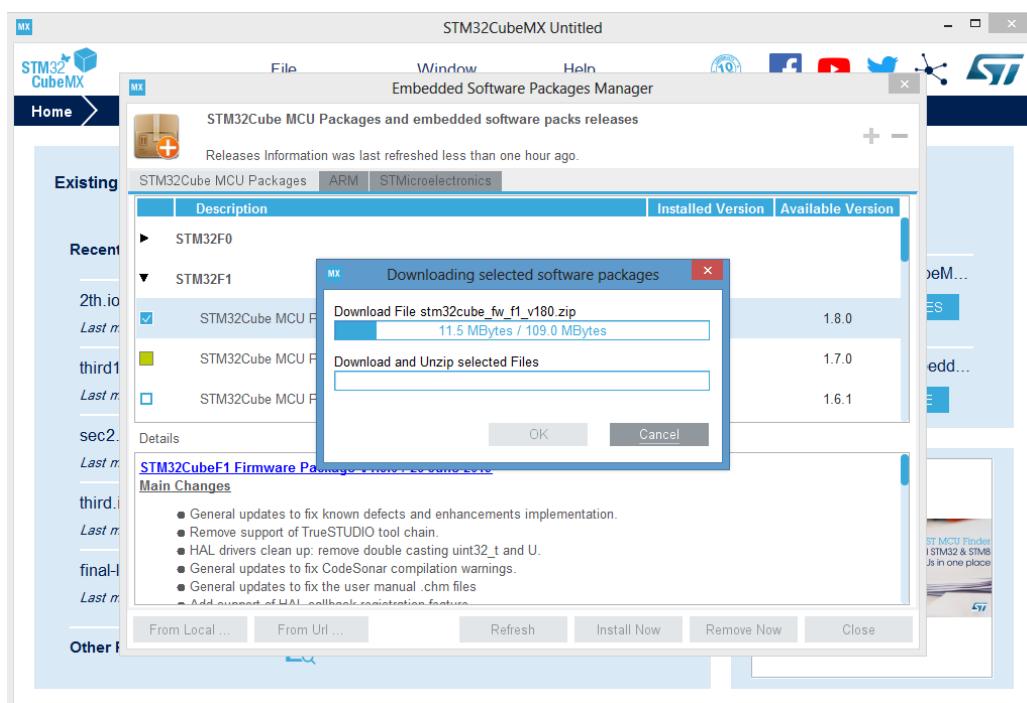


(شکل ۱۴)

-۳ تمامی آپدیت‌های سری STM32F1 دیده می‌شود، در گزینه ۱ مورد نصب شده مشاهده می‌شود و جهت آپدیت به آخرین نسخه گزینه ۲ را انتخاب نموده (شکل ۱۵) و سپس با زدن گزینه ۳ شروع به نصب می‌کند (شکل ۱۶).

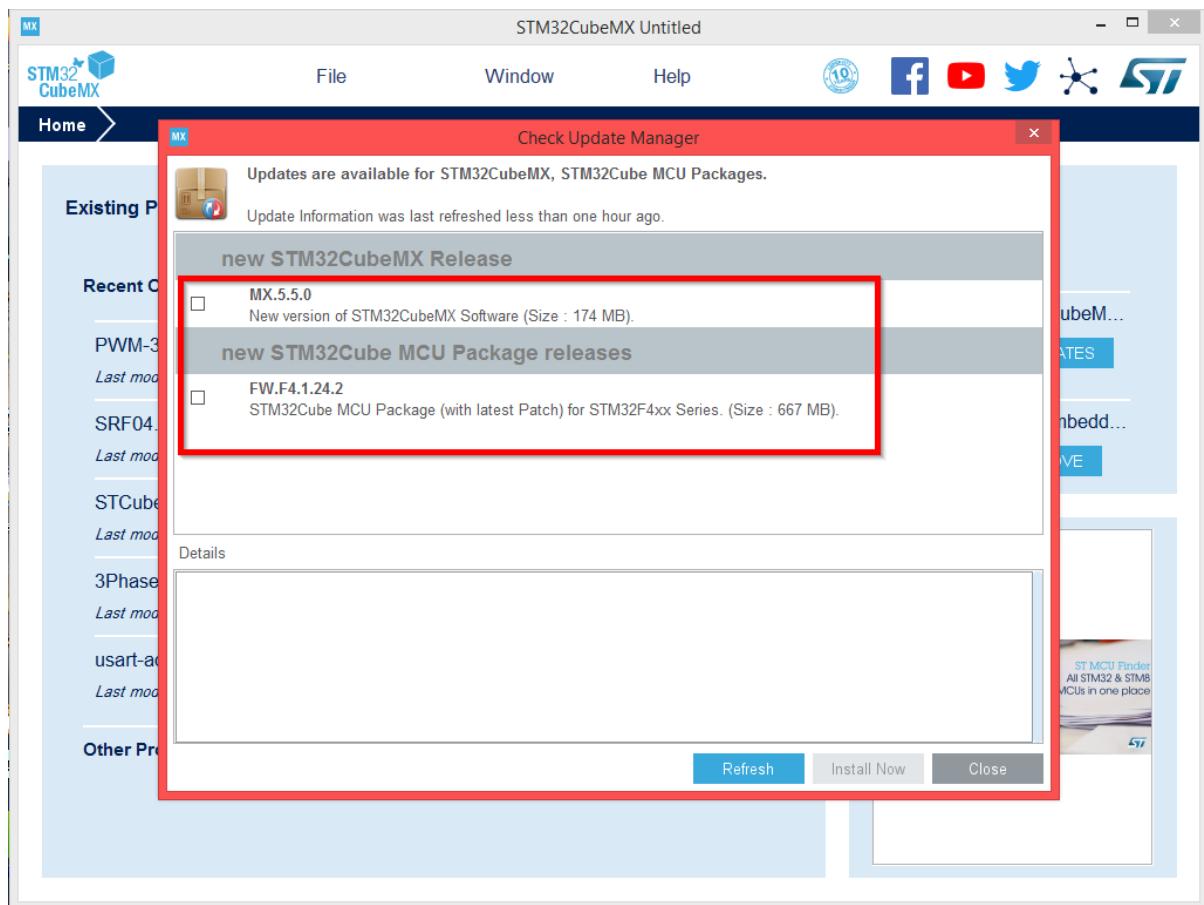


(شکل ۱۵)



(شکل ۱۶)

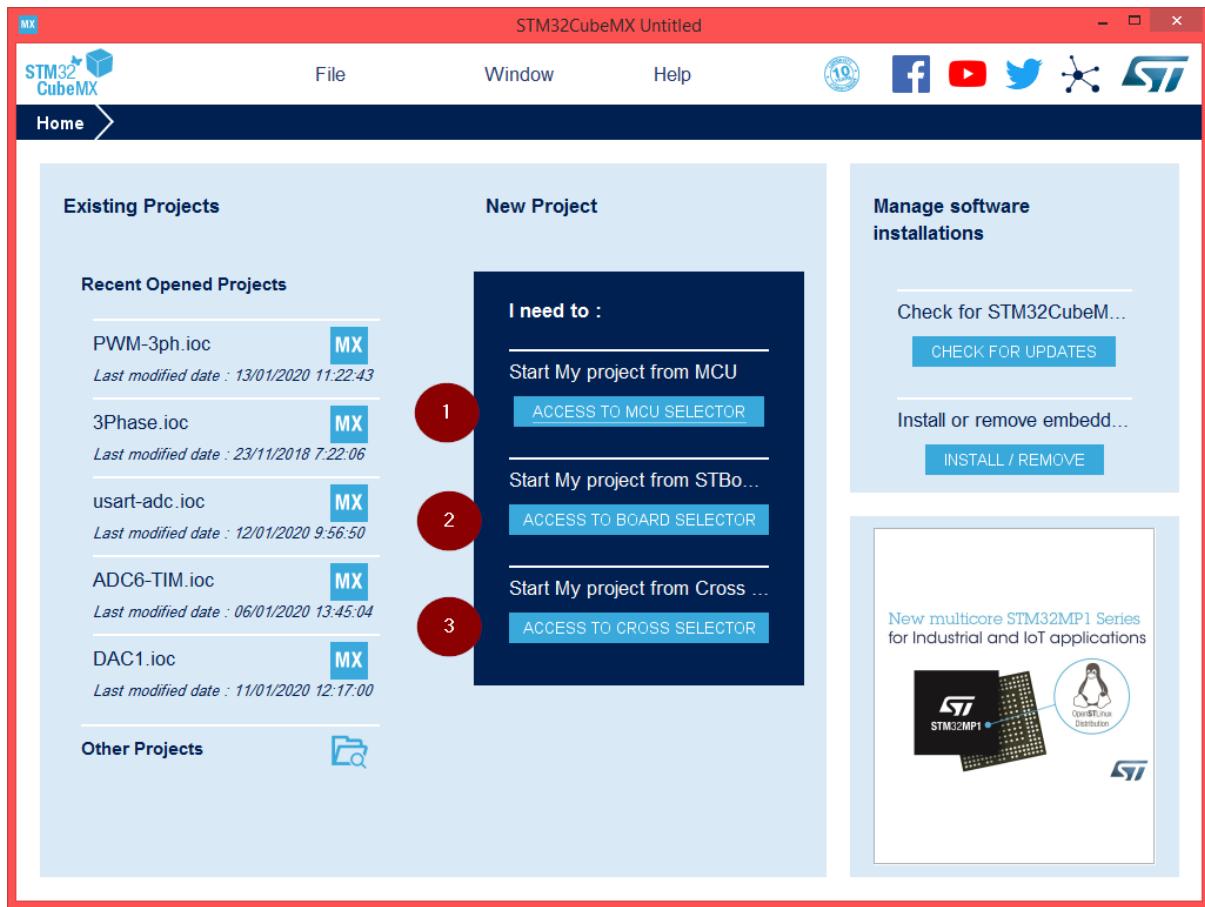
۴- همچنین با زدن گزینه ۳(شکل ۱۳) تمامی آپدیت‌ها اعم خود نرم‌افزار STM32CubeMX و سایر سری‌های نصب شده موجود می‌باشد(شکل ۱۷).



(شکل ۱۷)

#### ▪ تنظیمات اولیه در STM32CubeMX جهت ساخت پروژه:

۱- در صفحه نخست قسمت New Project دو روش برای ساخت پروژه وجود دارد یکی از طریق گزینه ۱(شکل ۱۸) که در این روش میکروکنترلر تنها در اختیار ACCESS TO MCU SELECTOR می‌باشد و دیگری گزینه ۲(شکل ۱۸) است که استفاده از ACCESS TO BOARD SELECTOR بردهای آماده میکروکنترلر می‌باشد و تفاوت آن با روش قبلی این می‌باشد که در این حالت برخی از پین‌ها که در برد مورد استفاده قرار گرفته مشخص شده و غیرقابل استفاده دیگری می‌باشند. در اینجا از روش اول که حالت جامع می‌باشد استفاده می‌کنیم.



(۱۸) شکل

۲- با انتخاب گزینه ۲ (شکل ۱۸) پنجره باز شده (شکل ۱۹) دارای قسمت های مختلف و متنوعی برای انتخاب میکروکنترلر می باشد. که عبارتند از:

الف- با استفاده از گزینه های ۱ تا ۴ با توجه به اطلاعاتی که در مورد Core, Series, Lines, Packages می توان میکروکنترلرهای مورد نیاز خود را در قسمت ۶ مشاهده و انتخاب نماییم (شکل ۱۹).

ب- در قسمت Other (گزینه ۵ شکل ۱۹) با فرض اینکه اطلاعاتی درمورد انواع میکروکنترلرها نداریم در ۶ پارامتر Price, Flash, EePROM, Ram, Freq, IO محدوده های مورد نظر خود را مشخص و میکروکنترلهای با این خصوصیات را در قسمت ۶ مشاهده و انتخاب می کنیم.

پ- در این روش شما میکروکنترلر مورد نظر را در اختیار دارید (بطور مثال STM32F103C8TX) و تنها با وارد نمودن نام آن در قسمت ۱ (شکل ۲۰) میکروکنترلر پیدا و در قسمت ۲ کلیه پارامترهای آن میکرو نمایش داده می شود. در قسمت ۳ با انتخاب مورد پیدا شده اطلاعات مختلفی از جمله Block, features

در تب‌های ۴ تا ۸ نشان داده می‌شود و با انتخاب Buy، DataSheet، Docs & Resource، Diagram

گزینه ۹ ساخت پروژه شروع می‌شود (شکل ۲۰).

The screenshot shows the STMicroelectronics website for the STM32MP1 Series. On the left, there are filters for Core (1), Series (2), Line (3), Package (4), and Other (5). The Other section is highlighted with a red box and contains filters for Price, IO, Eeprom, Flash, Ram, and Freq. A table below lists 1532 items, with the first item being the NUCLEO\_32F030B board (6). The main content area features the STM32MP1 logo and a Linux penguin icon representing the OpenSTLinux Distribution.

Par...	Reference	Board	Pack...	Flash	RAM	IO	Freq	GF...	CO...	D...	DE...	DES/TDES	FM...	HDF...	OT...	P...	D...	RF...	TA...	Trus...	VRE...
STM3...	STM32F030C6Tx	...	LQ...	32 ...	4 k...	39 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F030C8Tx	...	LQ...	64 ...	8 k...	39 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F030CCTx	...	LQ...	256 ...	32 ...	37 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F030F4Px	...	TS...	16 ...	4 k...	15 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F030K6Tx	...	LQ...	32 ...	4 k...	25 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F030R8Tx	NUCLEO_32F030B	LQ...	64 ...	8 k...	55 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F030RCTx	...	LQ...	256 ...	32 ...	51 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F031C4Tx	...	LQ...	16 ...	4 k...	39 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F031C6Tx	...	LQ...	32 ...	4 k...	39 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F031E6Yx	...	WL...	32 ...	4 k...	20 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F031F4Px	...	TS...	16 ...	4 k...	15 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
STM3...	STM32F031F6Px	...	TS...	32 ...	4 k...	15 48...	0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(شکل ۱۹)

MCU Selector Board Selector

MCU Filters

Part Number Search: stm32f103c8

**Core:** ARM Cortex-M3 (checked)

**Series:** STM32F1 (checked)

**Line:** STM32F103 (checked)

**Package:** LQFP48 (checked)

**Other:**

- Price = 2.056
- IO = 37
- Eeprom = 0 (Bytes)
- Flash = 64 (kBytes)
- Ram = 20 (kBytes)
- Freq. = 72 (MHz)

**STM32F103C8**

Mainstream Performance line, ARM Cortex-M3 MCU with 64 Kbytes Flash, 72 MHz CPU, motor control, USB and CAN

**ACTIVE** Active Product is in mass production

Unit Price for 10kU (US\$) : 2.056

LQFP48

The STM32F103x medium-density performance line family incorporates the high-performance ARM® Cortex®-M3 32-bit RISC core operating at a 72 MHz frequency, high-speed embedded memories (Flash memory up to 128 Kbytes and SRAM up to 20 Kbytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. All devices offer two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I2Cs and SPIs, three USARTs, an USB and a CAN.

The devices operate from a 2.0 to 3.6 V power supply. They are available in both -40 to +85 °C temperature range and the -40 to +105 °C extended temperature range. A comprehensive set of power-saving mode allows the design of low-power applications.

The STM32F103x medium-density performance line family includes devices in six different package types: from 36 pins to 100 pins. Depending on the device chosen, different sets of peripherals are included; the description below gives an overview of the complete range of peripherals proposed in this family.

These features make the STM32F103x medium-density performance line microcontroller family suitable for a wide range of applications such as motor drives, application control, medical and handheld equipment, PC and gaming peripherals, GPS platforms, industrial applications, PLCs, inverters, printers, scanners, alarm systems, video intercoms, and HVACs.

**Features**

- ARM®32-bit Cortex®-M3 CPU Core
  - 72 MHz maximum frequency, 1.25 DMIPS/MHz (Dhrystone 2.1) performance at 0 wait state memory access
  - Single-cycle multiplication and hardware division
- Memories
  - 64 or 128 Kbytes of Flash memory
  - 20 Kbytes of SRAM

MCUs List: 1 item

Part No.	Reference	Marketin	Unit Price for 10kU	Board	Package	Flash	RAM	IO	Freq	GFX Sc
STM32F103C8	STM32F103C8Tx	Active	2.056		LQFP48	64 kBytes	20 kBytes	37	72 MHz	0.0

(٢٠) شكل

### **فصل سوم**

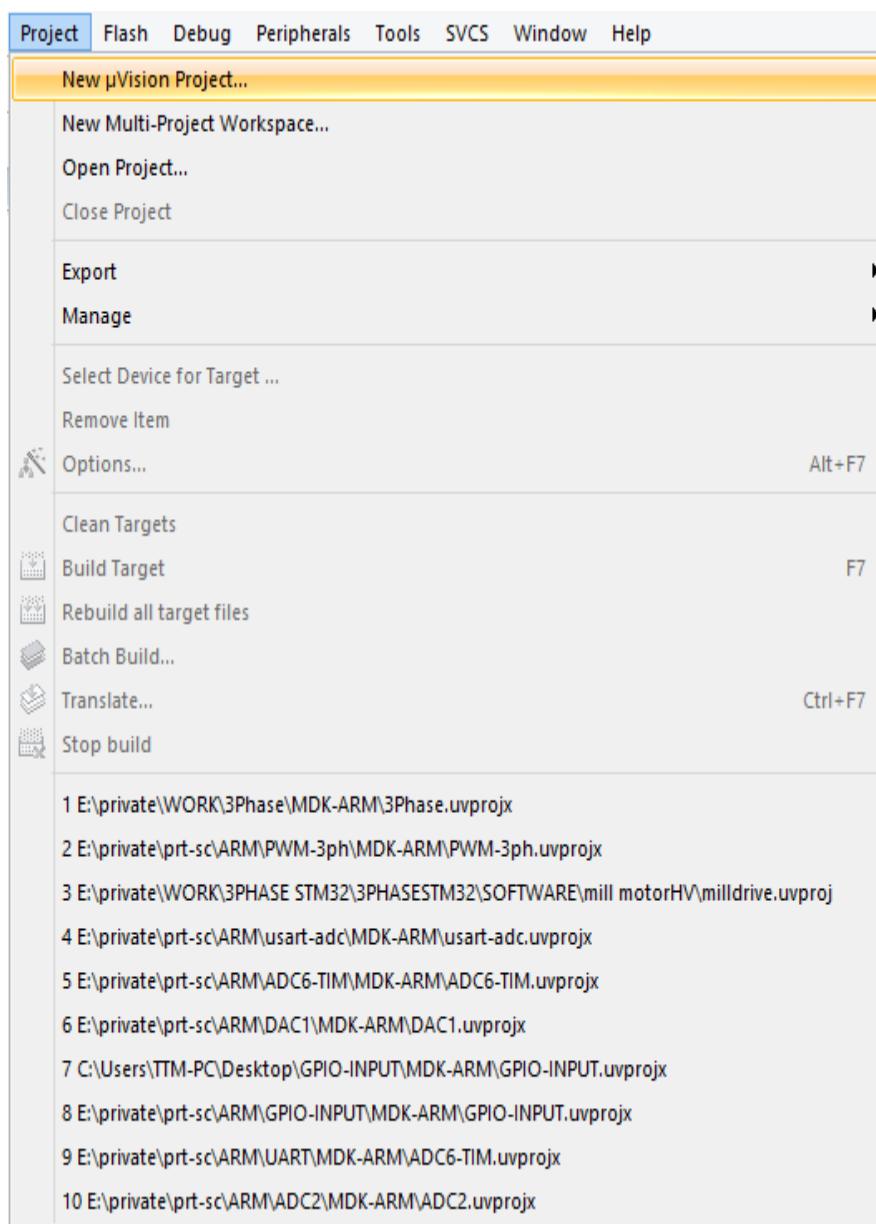
**ساخت پروژه بدون استفاده از Cube**

## ۳-۱ ساخت پروژه بدون استفاده از cube

در این فصل قصد داریم بطور خلاصه یک مثالی را بدون استفاده از نرم افزار Cube صرفاً جهت آشنایی انجام دهیم. چون هدف اصلی ساخت پروژه به واسطه Cube است که کار را بسیار ساده و قابل فهم و سریع می کند انجام می دهیم.

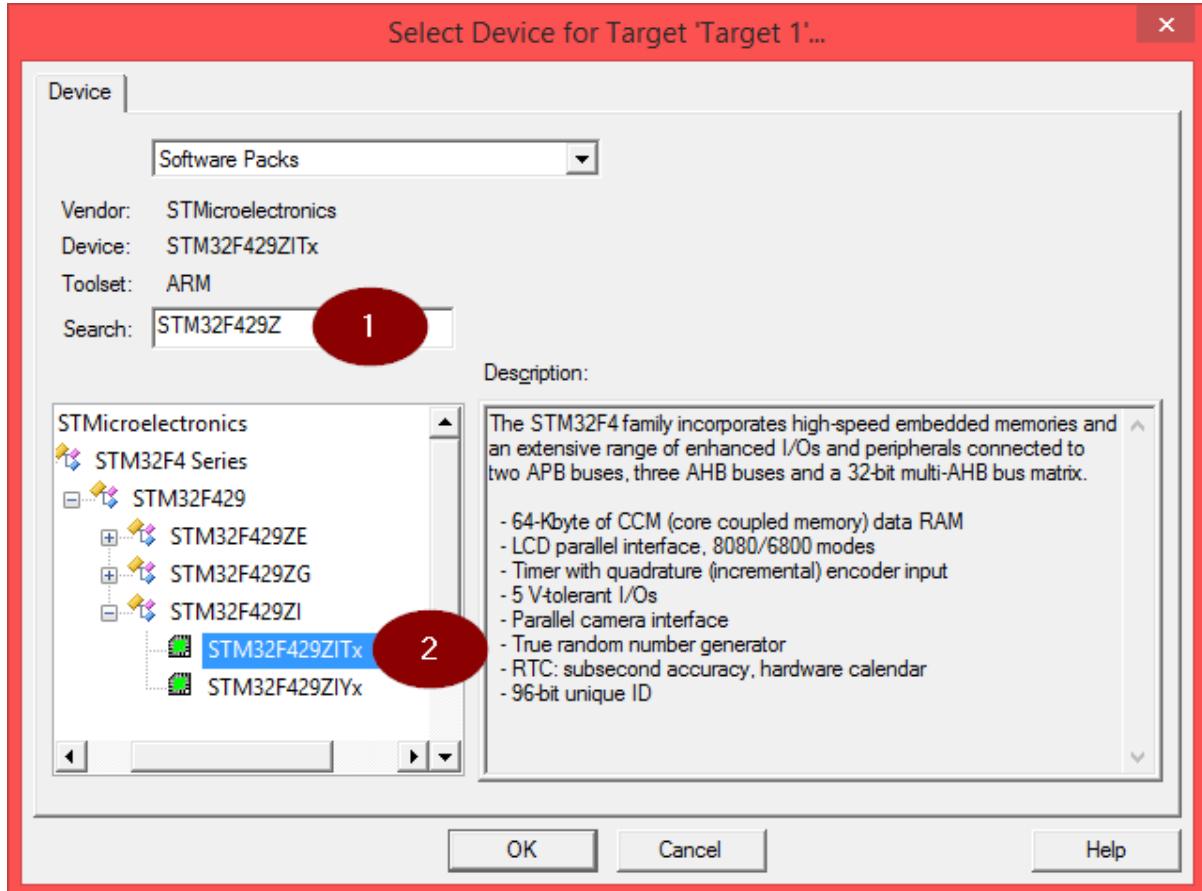
۱- ابتدا keil را باز می کنیم و از روی TaskBar گزینه New uVision Project را انتخاب می کنیم(شکل

. (۲۰



(شکل . (۲۰

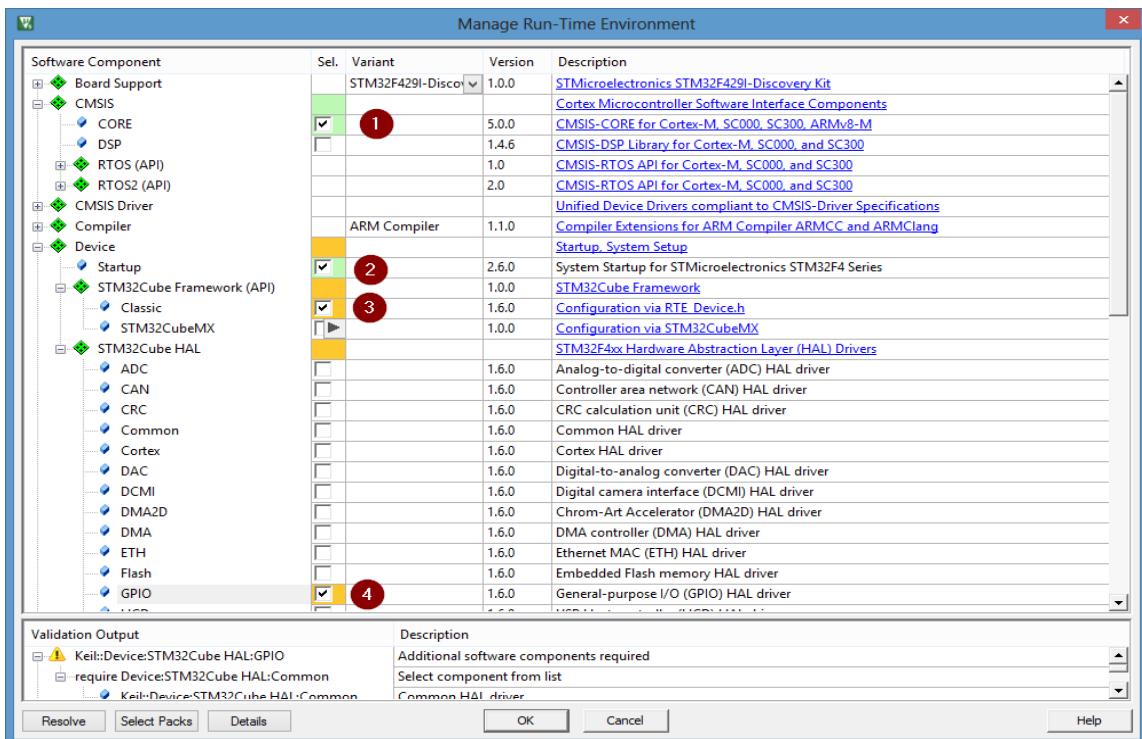
- ۲- در پنجره باز شده (شکل ۲۱) میکروکنترلر مورد نظر را به دو روش انتخاب می کنیم:
- الف- یا بطور مستقیم نام میکرو را وارد می کنید و آن را انتخاب می کنیم (شکل ۲۱ گزینه ۱).
  - ب- یا میکرو را از بین تقسیم‌بندی‌های قسمت ۲ انتخاب می کنیم (شکل ۲۱ گزینه ۲).



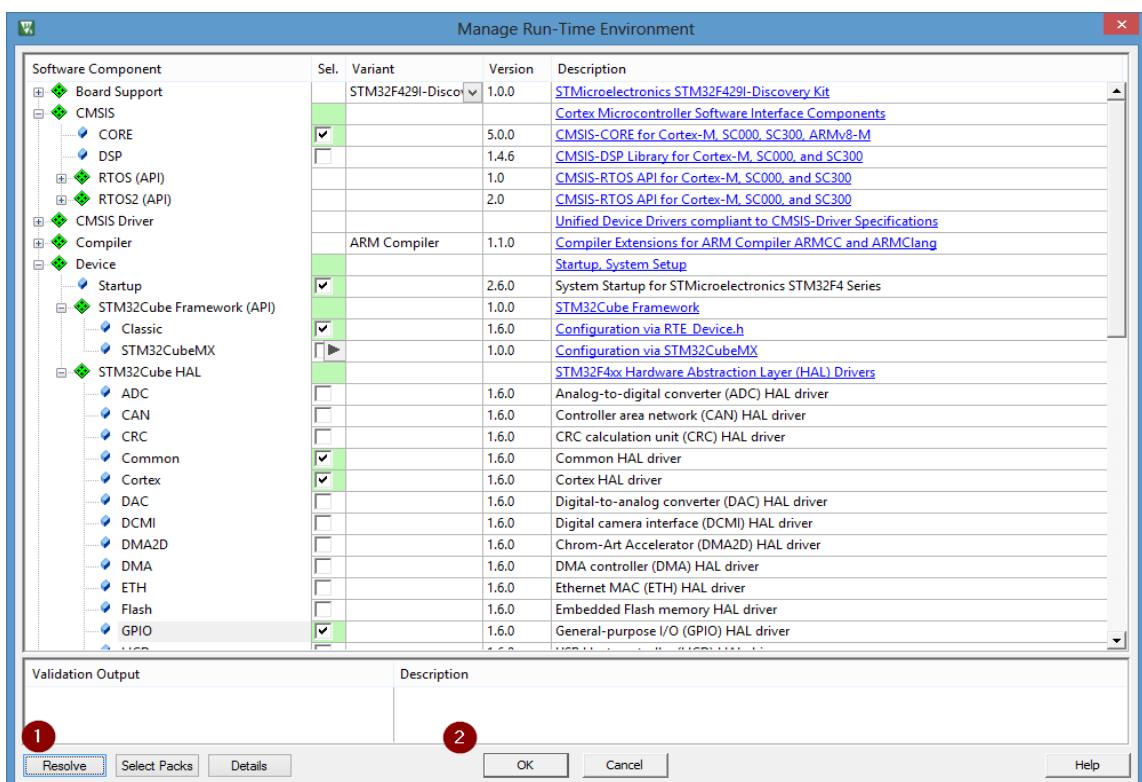
(شکل ۲۱)

- ۳- پنجره جدید (شکل ۲۲) باز شده Run Time Environment نام دارد که مشخص می کند می خواهد از کدام peripheral‌ها استفاده کنیم.
- ۴- با توجه به نیاز پروژه که روشن کردن<sup>۱</sup> LED می باشد موارد ۱ و ۲ و ۳ و ۴ (شکل ۲۲) را انتخاب می کنیم (کلیه پارامترها در فصول بعدی توضیح داده خواهد شد).
- ۵- همانطور که مشاهده می کنید برخی از سلوهای نارنجی رنگ می باشند که دلیل آن این است که نیازمند فعال شدن peripheral‌های دیگر می باشد. برای رفع این مشکل گزینه Resolve (گزینه ۱ شکل ۲۳) را انتخاب نموده که بصورت اتوماتیک مشکل را حل می کند و در نهایت OK را می زنیم.

<sup>۱</sup> Light Emitting Diode

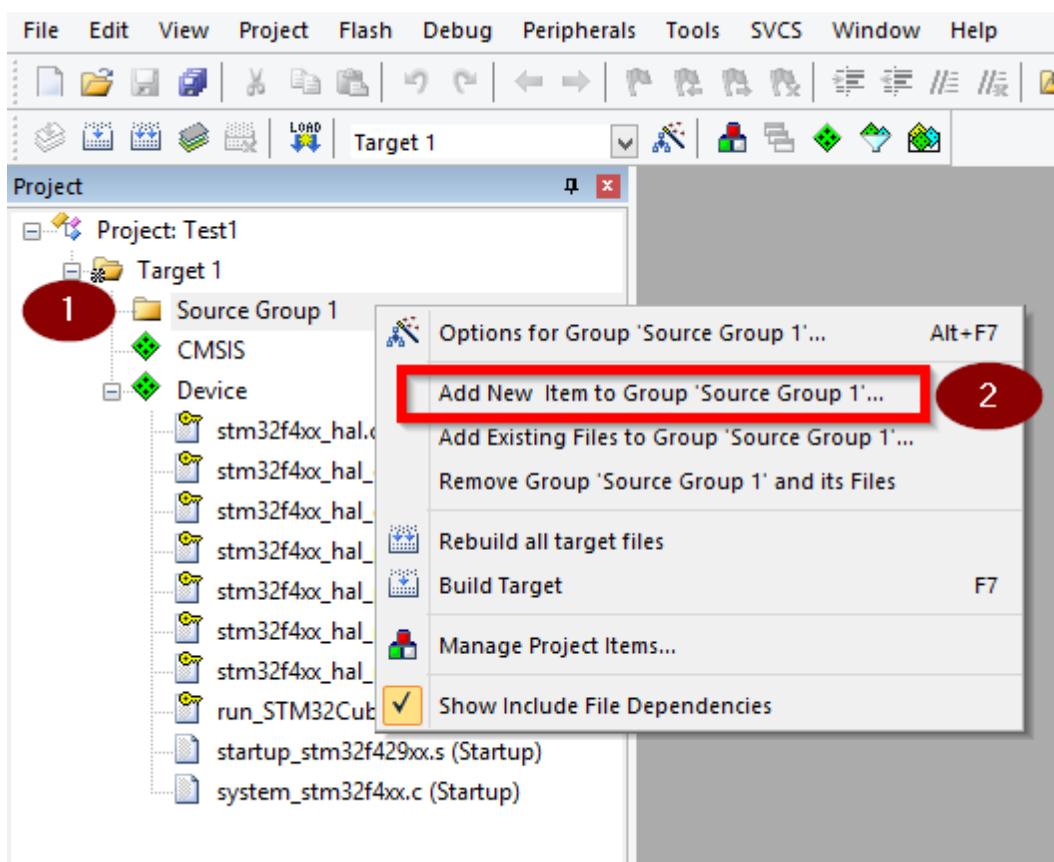


(٢٢) شكل



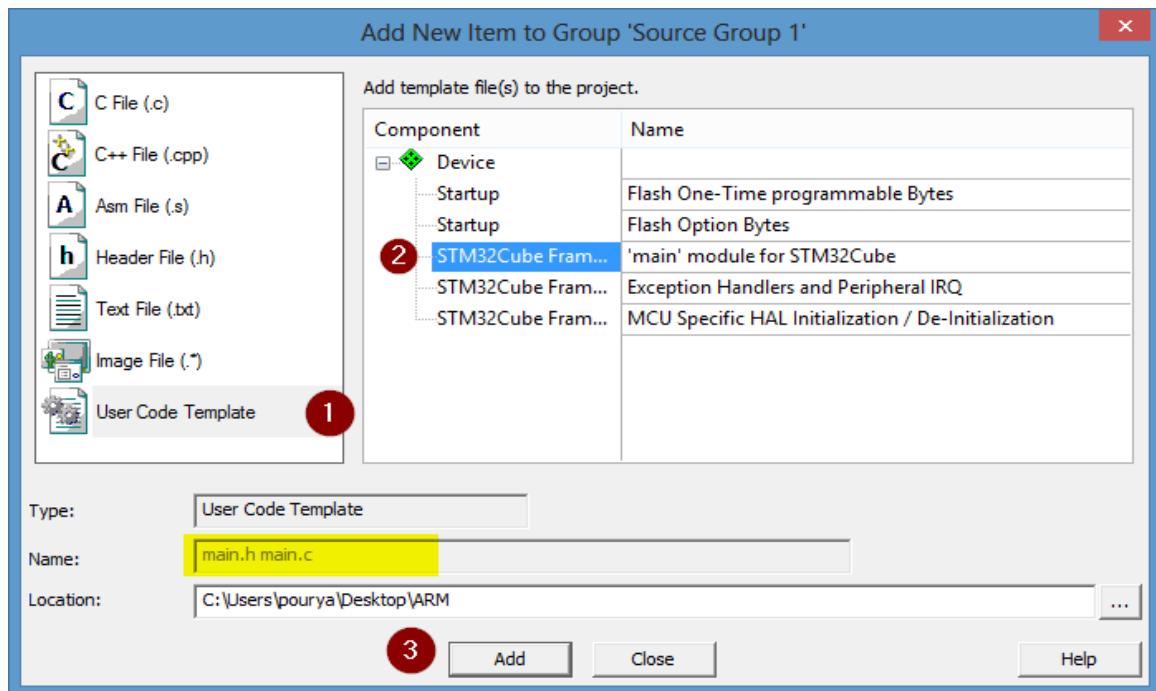
(٢٣) شكل

۶- حال پروژه ایجاد شده و کتابخانه‌های مدنظر ایجاد شده‌اند، مطابق با کلیه نرم افزارهای برنامه نویسی در اینجا نیز برنامه از ماثول Main ساخته می‌شود. در همین راستا برای اضافه کردن این ماثول بر روی کلیک راست کرده (گزینه ۱ شکل ۲۴) و Add New Item to Group Source Group می‌کنیم.



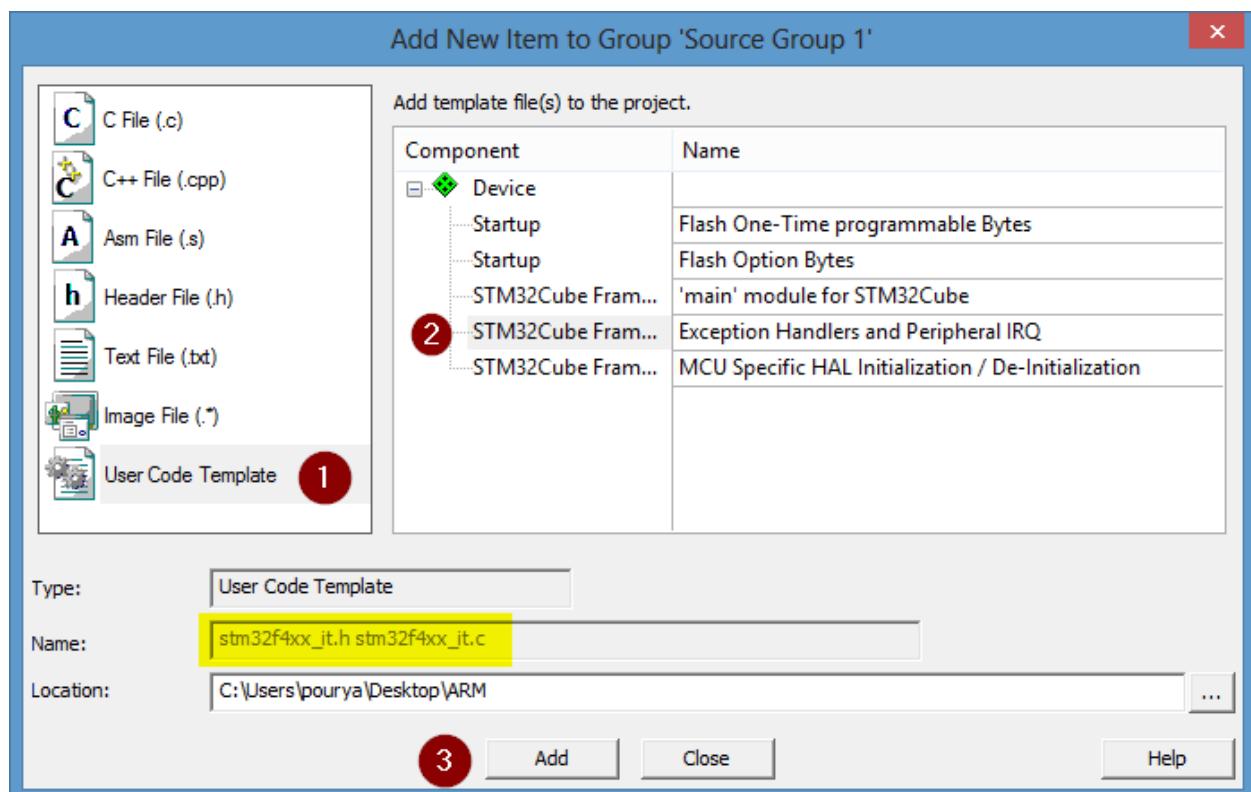
(شکل ۲۴)

۷- در پنجره جدید User Code Template (گزینه ۱) را باز سپس گزینه ۲ را انتخاب می‌کنیم تا ماثول با دو پسوند .c و .h اضافه شود(شکل ۲۵).



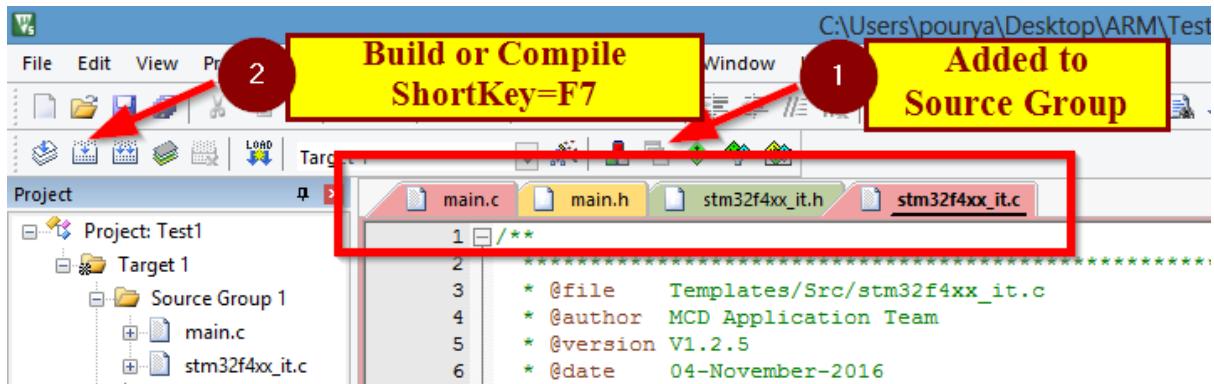
(۲۵) شکل

۸- مورد ضروری دیگر که باید به برنامه اضافه شود Exception Handlers and Peripheral IRQ است که تمامی دستورهای interrupt در آن می باشد و در صورت اضافه نکردن باعث هنگ کردن برنامه می شود (شکل ۲۶).

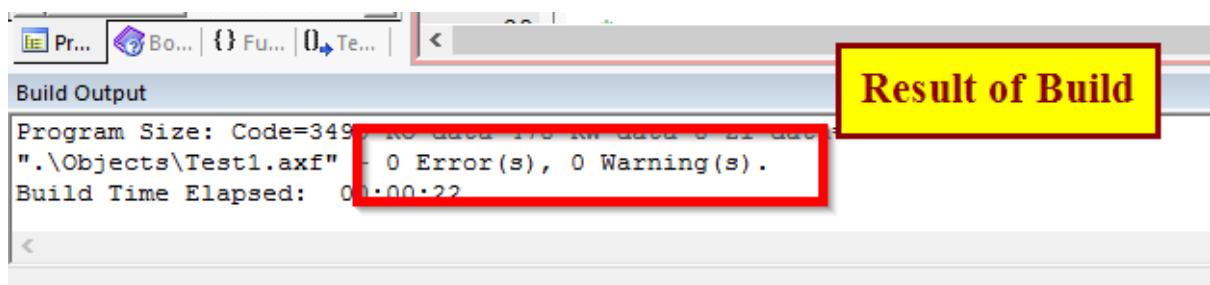


(شکل ۲۶)

-۹ همانطور که در قسمت ۱ (شکل ۲۷) مشاهده می شود موارد مذکور بندهای ۸ و ۹ اضافه شده است. حال با استفاده کلید میانبر F7 (گزینه ۲) برنامه را Build یا Compile می کنیم. نتیجه Build در شکل ۲۸ قابل مشاهده می باشد که دارای Error نمی باشد.



(شکل ۲۷)



(شکل ۲۸)

در این برنامه نیاز به استفاده از Input/Output General Purpose GPIO است که مخفف Peripheral GPIO می باشد و نقش ورودی و خروجی پایه های میکرو کنترلر را ایفا می کند (باتوجه به اینکه در این فصل هدف آشنایی با برنامه نویسی بدون استفاده از Cube می باشد بصورت مختصر توضیح داده شد ولی در فصل های آینده بطور مفصل مورد بحث قرار می گیرد).

## ۲-۳ شروع برنامه نویسی:

۱- در ابتدا کلاک های مربوط به GPIO را در داخل حلقه Main فعال می کنیم (شکل ۲۹).

```
_HAL_RCC_GPIOG_CLK_ENABLE();
```

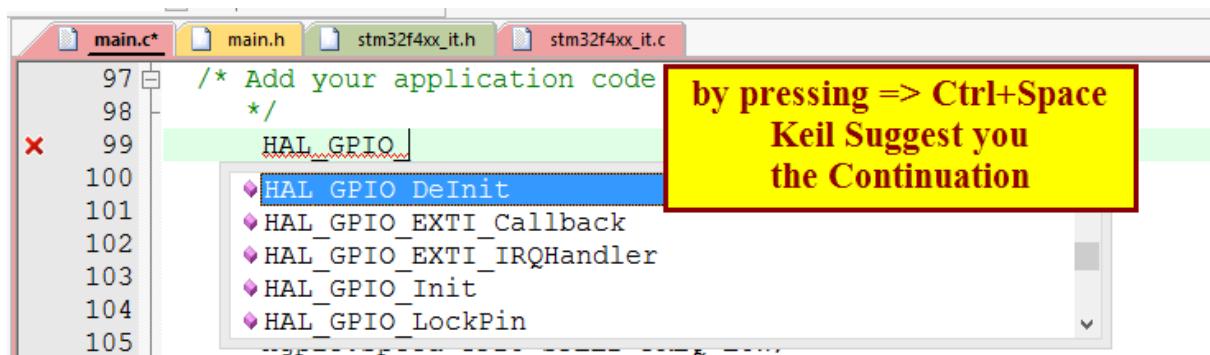
```

85 int main(void)
86 {
87 #ifdef RTE_CMSIS_RTOS // when
88     osKernelInitialize(); // init
89 #endif
90
91     HAL_Init();
92
93     /* Configure the system clock to 168 MHz */
94     SystemClock_Config();
95
96     __HAL_RCC_GPIOG_CLK_ENABLE(); // HAL_RCC_GPIOG_CLK_ENABLE();
97     /* Add your application code here */
98     /*
99
100

```

(۲۹) (شکل)

نکته: در Keil برای فهمیدن ادامهی جمله از کلید ترکیبی Ctrl + Space استفاده می شود. مانند شکل ۳۰ که با استفاده از آن پیشنهادهای Keil برای نوشت ادامه جمله کمک گرفته می شود ضمنا در این صورت می توان تنها با حروف کوچک جمله را نوشت و با استفاده از کمک های پیشنهادی، آن را بصورت اتوماتیک تصحیح می کند.



(۳۰) (شکل)

نکته: برای فهمیدن تعریف تابع بطور مثال فهمیدن ورودی، خروجی ها در keil می توان آن تابع را نوشت سپس با استفاده از کلید میانبر F12 یا راست کلیک و انتخاب Go To Definition به داخل تعریف تابع در کتابخانه های اضافه شده رفت (شکل ۳۱).

```

97  /* Add your application code here
98  */
99      HAL_GPIO_Init(GPIOG, &hgpi...
100
101
102 #ifdef RTE_CMSIS_RTOS
103     // create 'thread' functions
104     // example: tid name = osThre...
105
106     osKer
107 #endif
108
109     /* Infinite loop */
110     while (1)
111     {
112
113     }
114 }
115
116 /**
117 * @brief System Clock Configuration

```

(۳۱)

- در آغاز برای فراخوانی تابع GPIO در داخل حلقه Main()، HAL\_GPIO\_Init() را نوشته سپس با زدن کلید میانبر F12 یا راست کلیک و انتخاب Go To Definition به داخل تعریف تابع می‌رویم (شکل ۳۲) مشاهده می‌شود تابع خروجی ندارد (Void) و دو ورودی که یک Structure می‌باشد و شامل رجیسترها بی می‌باشد که در مراحل بعدی به تعریف آنها مراجعه می‌کنیم. ورودی اول نوع یا اسم ورودی و دومی یک متغیر سراسری است که نوع خاصی از داده می‌باشد.

```

186     *           the configuration information for the specified GPIO peripheral.
187     * @retval None
188     */
189 void HAL_GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_InitStruct)
190 {
191     uint32_t position;
192     uint32_t ioposition = 0x00U;

```

(۳۲)

- پس ابتدا یک متغیر سراسری از نوع GPIO\_InitTypeDef قبل از تابع Main به نام hgpio تعریف می‌کنیم (شکل ۳۳).

```
73
74     GPIO_InitTypeDef hgpio;
75
76     static void SystemClock_Config(void);
77     static void Error_Handler(void);
78
79     /* Private functions -----
80     */
81     * @brief Main program
82     * @param None
83     * @retval None
84     */
85     int main(void)
86 {
87 }
```

(شکل ۳۳)

- حال تابع HAL\_GPIO\_Init() را کامل می‌کنیم، برای ورودی اول پورت G را معرفی می‌کنیم و ورودی دوم را نیز متغیر سراسری که در بند ۳ تعریف کردیم استفاده می‌کنیم. نکته‌ای که باید به آن توجه کرد این است که باید در ورودی دوم آدرس Structure را وارد کنیم نه خود به همین دلیل باید ابتدای آن از علامت & استفاده کنیم (گزینه ۱ شکل ۳۴).

```
103     /* Configure the system clock to 168 MHz */
104     SystemClock_Config();
105
106     __HAL_RCC_GPIOG_CLK_ENABLE();
107     /* Add your application code here
108     */
109     2     hgpio.Mode=GPIO_MODE_OUTPUT_PP;
110     3     hgpio.Pull=GPIO_NOPULL;
111     4     hgpio.Speed=GPIO_SPEED_FREQ_LOW;
112     5     hgpio.Pin=GPIO_PIN_13 | GPIO_PIN_14;
113
114     1     HAL_GPIO_Init(GPIOG, &hgpio);
115
```

(شکل ۳۴)

۵- در گام بعدی به سراغ تعاریف GPIO\_TypeDef که در بند ۳ معرفی گردید می‌رویم تا عضوهای آن را معرفی و تنظیم کنیم، بنابراین بر روی آن کلیک کرده و F12 را می‌زنیم، در شکل ۳۵ مشاهده می‌شود که نوع و مقدار هر کدام از عضوهای گفته شده است.

الف: uint32\_t Pin: شماره پایه‌های مورد استفاده را مشخص می‌کند(گزینه ۱).

ب - Output Push Pull (GPIO\_MODE\_OUTPUT\_PP) دو مقدار uint32\_t Mode : و Output Open (GPIO\_MODE\_OUTPUT\_OD) Drain پایه قادر به تأمین جریان بار متصل به آن است، اما در حالت دوم، تنها یک ولتاژ بر روی پایه قرار گرفته ولی پایه قادر به تأمین جریان نیاز بار نیست(گزینه ۲).

پ : uint32\_t Pull : بر خلاف میکروکنترلرهای AVR که در پایه‌های خود تنها مقاومت Pull up دارند، در میکروکنترلرهای ARM هم مقاومت Pull up و هم Pull down جهت رفع نیاز بدون استفاده از مقاومت‌های خارجی تعییه شده است. به عبارتی مشخص کننده سطح منطقی پایه در حالت آزاد می‌باشد(گزینه ۳).

ت - uint32\_t Speed : مشخص کننده فرکانس تغییر حالت (سوئیچینگ) برای پایه مورد نظر است می‌باشد شامل سه گزینه Low ، Medium و High می‌باشد، که مقادیر فرکانسی متناظر با آنها بسته به نوع میکروکنترلر متفاوت است. هر چه بیشینه فرکانس سوئیچینگ را افزایش دهیم، مصرف توان نیز افزایش می‌یابد(گزینه ۴).

```

 66 1
 67     uint32_t Pin;           / 1 specifies the GPIO pins to be con-
 68                                         This parameter can be any value o-
 69
 70     uint32_t Mode;          / 2 specifies the operating mode for t-
 71                                         This parameter can be a value of |
 72
 73     uint32_t Pull;          / 3 specifies the Pull-up or Pull-Down
 74                                         This parameter can be a value of |
 75
 76     uint32_t Speed;         / 4 specifies the speed for the select-
 77                                         This parameter can be a value of |
 78
 79     uint32_t Alternate;     / 5 Peripheral to be connected to the
 80                                         This parameter can be a value of |
 81 }GPIO_InitTypeDef;
 82

```

<sup>1</sup> Switching

(شکل ۳۵)

- ۶ با توجه به توضیحات بند ۵ عضوهای آن را معرفی و تنظیم کنیم(شکل ۳۴ گزینه‌های ۲ تا ۵).
  - ۷ حال با برای استفاده از دستور HAL\_GPIO\_WritePin ، دستور WritePin را نوشته و مطابق معمول برای اطلاع از ورودی‌های آن بروی آن کلیک کرده و F12 را می‌زنیم(شکل ۳۶).
- ۱- اسم پورت  
۲- شماره پایه  
۳- حالت منطقی پایه

```

427 * @arg GPIO_PIN_SET: to set the port pin
428 * @retval None
429 */
430 void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
431 {
432     /* Check the parameters */
433     assert_param(IS_GPIO_PIN(GPIO_Pin));
434     assert_param(IS_GPIO_PIN_ACTION(PinState));

```

(شکل ۳۶)

- ۸ با توجه به بند ۷ HAL\_GPIO\_WritePin را در داخل حلقه While(1) کامل می‌کنیم و F7 را جهت کردن میزیم.

```

121 #endif
122
123 /* Infinite loop */
124 while (1)
125 {
126     HAL_GPIO_WritePin(GPIOG, GPIO_PIN_13, GPIO_PIN_SET);
127 }
128
129

```

(شکل ۳۷)

## فصل ۴

درگاههای ورودی / خروجی (GPIO)

و محیط دیباگ

## ۱-۴ درگاههای ورودی/خروجی (GPIO)

یک مدار مجتمع دیجیتال می‌باشد که از طریق پایه‌های خود با سایر اجزای مدار ارتباط برقرار می‌کند. به عبارتی در حالت ورودی با صفر و یک کردن پایه‌ها، انتقال اطلاعات به بیرون را انجام می‌دهد و در حالت خروجی با خواندن صفر و یک‌ها از محیط خارجی اطلاعات را دریافت می‌کند.

پایه‌ها ممکن است یک نقش ثابت را بر عهده می‌گیرند. بطور مثال پایه میکرو به عنوان پایه فرستنده (TX) یا گیرنده (RX) در پروتکل سریال (USART) تعریف می‌شود. در این حالت با توجه به قوانین USART، رفتار این پایه تغییر می‌کند. اما گاهی ممکن است نیاز باشد کاربر به صورت مستقیم پایه را یک یا صفر کند. یا در حالت ورودی مقدار اعمال شده بر روی آن را بخواند. در این صورت پایه باید به عنوان General Purpose (General Purpose Input/Output) GPIO شکل دلخواه و مستقیم شود. در واقع ما یک نقش عمومی (General) به پایه داده‌ایم که کاربر به آن را بخواند.

### ۴-۱ بردسی ساختار GPIO مطابق با شماتیک مداری آن

باتوجه به ساختاری مداری شکل ۱ به شرح ساختار GPIO می‌پردازیم:

الف- تنها چیزی که ما خارج از میکروکنترلر می‌بینیم پایه خروجی I/O Pin (گزینه ۱) می‌باشد و مابقی ساختار داخل میکروکنترلر است.

نکته: کلیه VDD های موجود در شکل ۵ و Vss معادل ۵V و VDD-FT (Five Tolerance) می‌باشد و معادل صفر ولت می‌باشد.

ب- دو دیود حفاظتی<sup>۱</sup> و<sup>۲</sup> را در شکل می‌بینیم، در صورتی که ولتاژ خروجی بیش از ۵ ولت شود دیود بالای جریان را هدایت می‌کند و مانع آسیب رساندن به پایه و میکرو می‌شود و اگر ولتاژ خروجی کمتر از صفر ولت (ولتاژ منفی) شود دیود پایینی جریان را هدایت می‌کند و مانع آسیب رساندن به پایه و میکرو می‌شود.

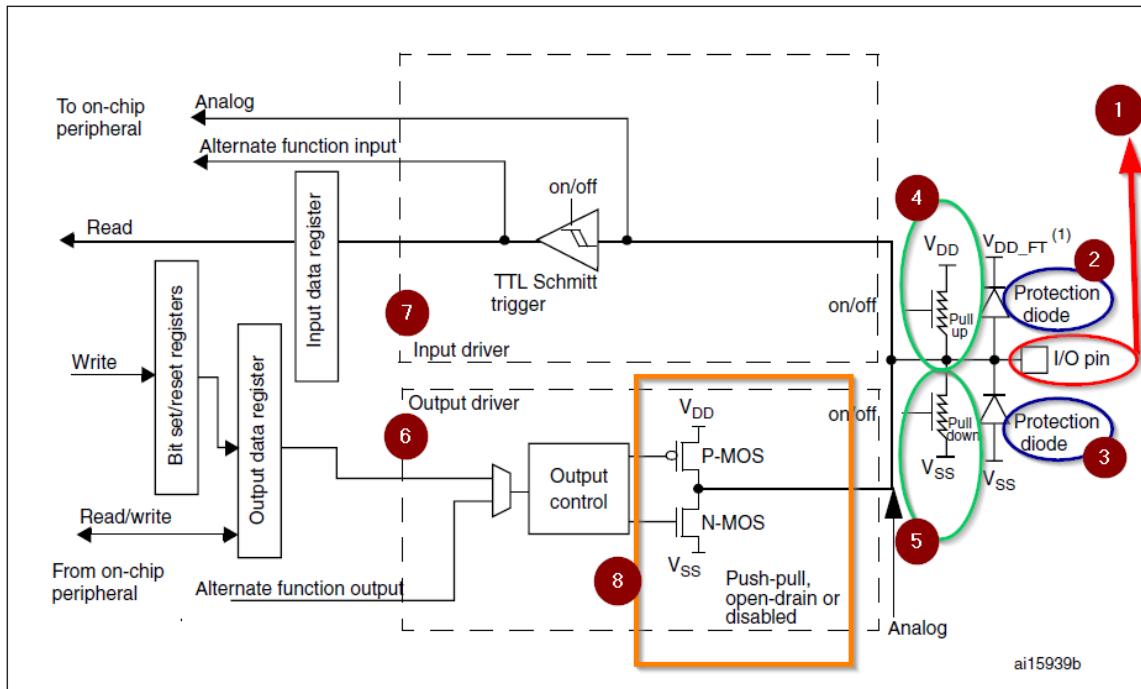
پ- دو مقاومت<sup>۳</sup> و<sup>۴</sup> را در شکل می‌بینیم، که کلیدهایی بر روی هر کدام از آن‌ها قرار دارد، بدین معنی که پایه میکرو وقتی به عنوان پین ورودی انتخاب می‌گردد، در حالتی که پایه آزاد می‌باشد مقدار آن با استفاده از مقاومت Pull-Up یک منطقی یا با استفاده از مقاومت Pull-Down صفر منطقی تنظیم گردد.

ت- یک طبقه خروجی (گزینه ۶) و یک طبقه ورودی (گزینه ۷) داریم، بطوریکه اگر پایه در مد خروجی باشد جریان از طبقه خروجی و اگر پایه در مد ورودی باشد جریان از طبقه ورودی عبور می‌کند.

<sup>1</sup> Transmit X

<sup>2</sup> Receive

و- در طبقه خروجی (گزینه ۶) دو ماسفت کانال N و P (گزینه ۸) مشاهده می شود که می توانند در دو حالت ایفای نقش کنند، یکی حالت Open Drain که پایه درین مدار باز می شود و جریانی را عبور نمی دهد و صرفا کنترل ولتاژ خروجی را انجام می دهد. حالت دیگر Push Pull می باشد که در مواردی مورد استفاده قرار می گیرد که خروجی ما به یک مصرف کننده مانند LED اتصال داشته باشد که نیاز به جریان کشی دارد.



(شکل ۱)

#### ۴-۱-۲ جهت ادامه توضیحات نرم افزار STM32CubeMX را باز می کنیم:

۱- مطابق توضیحات در فصل پیشین یک پروژه با انتخاب میکروکنترلر STM32F103C8TX ایجاد می کنیم و سپس در مراحل بعدی شروع به تنظیم Peripheral ها می کنیم.

۲- یک محیط گرافیکی می باشد که در گزینه ۱ (شکل ۲) میکرو را در قسمت PinOut مشاهده می کنیم و در قسمت Package (گزینه ۲) عدد ۴۸ نشان دهنده تعداد پایه های میکرو است که ۳۷ تای آن قابل استفاده می باشد.

۳- در ابتدای هر پروژه دو تنظیم مهم را باید انجام دهیم:  
نکته ۱: کلاک عامل تغذیه کننده peripheral ها می باشد، که مانند خون در بدن است و باید به تمامی peripheralها برسد.

نکته ۲: معمولاً در انجام پروژه‌ها از کلاک خارجی استفاده می‌کنیم و دلیل آن هم این است که در کلاک‌های داخلی با توجه به این که از مدار<sup>۱</sup> RLC استفاده جهت تولید کلاک استفاده می‌کنند و همانطور که می‌دانیم مدارات RLC نسبت به تغییرات دمایی عکس العمل نشان می‌دهند و در نهایت فرکانس خروجی تغییر می‌کنند به همین دلیل جهت تولید فرکانس با دقت بالا از کریستال خارجی استفاده می‌کنیم.

نکته ۳: اگر عدد روی کریستال ۸ مگاهرتز را در شکل ۳ مشاهده کنید تعدادی صفر پس ممیز قرار دارد (۸,۰۰۰)، که تعداد این صفرها نشان‌دهنده دقت کریستال می‌باشد که هر چه تعداد آن بیشتر، دقت کریستال نیز بیشتر می‌باشد.

الف- تنظیم کلاک میکروکنترلر که منبع فرکانس کاری میکرو کنترلر می‌باشد :

در تب Pinout & Configuration (گزینه ۴) در قسمت SystemCore (گزینه ۵) بر روی زیر شاخه آن RCC کلیک می‌کنیم و با سه حالت موافق می‌شویم که حالت اول (گزینه ۶) نوع کلاک از نوع خارجی و فرکانس بالا در محدوده مگاهرتز<sup>۳</sup> HSE (گزینه ۷) و حالت دوم باز هم کلاک از نوع خارجی ولی فرکانس پایین در محدوده کیلوهرتز<sup>۴</sup> LSE و در حالت سوم (گزینه ۸) کلاک از نوع داخلی می‌باشد که با وابسته به نوع میکرو معمولاً ۸ مگاهرتز می‌باشد. با توجه به اینکه می‌خواهیم در محدوده مگاهرتز کار کنیم HSE (گزینه ۷) را انتخاب می‌کنیم که دارای دو حالت یکی BYPASS (گزینه ۹) که برای کریستال‌های ۴ پایه حرфه‌ای با دقت خیلی بالا استفاده می‌شود و دیگری Crystal/Ceramic Resonator (گزینه ۱۰) کریستال ۲ پایه با دقت مناسب می‌باشد، که ما در کلیه پروژه‌ها Crystal/Ceramic Resonator را انتخاب می‌کنیم. مشاهده می‌شود که دو پایه میکرو (گزینه ۱۱ و ۱۲) به کریستال اختصاص داده شده است. حال برای تنظیمات کلاک به تب Clock Source (گزینه ۱۳) Configuration می‌رویم.

به این نمودار(شکل ۴) Clock Tree یا درخت کلاک می‌گوییم که نحوه توزیع کلاک را در peripheral‌های مختلف نشان می‌دهد (باتوجه به گستردگی درخت کلاک، جهت فهم بهتر فقط قسمت مورد نیاز این فصل توضیح داده می‌شود و در فصل‌های بعدی بر حسب نیاز بقیه پارامترها توضیح داده می‌شود) که در ابتدا باید منبع کلاک ورودی را مشخص می‌کنیم<sup>۵</sup> HSI. که یک کلاک داخلی با

<sup>1</sup> Resistor Inductor Capacitor

<sup>2</sup> Reset & Clock Control

<sup>3</sup> High Speed External

<sup>4</sup> Low Speed External

<sup>5</sup> High Speed Internal

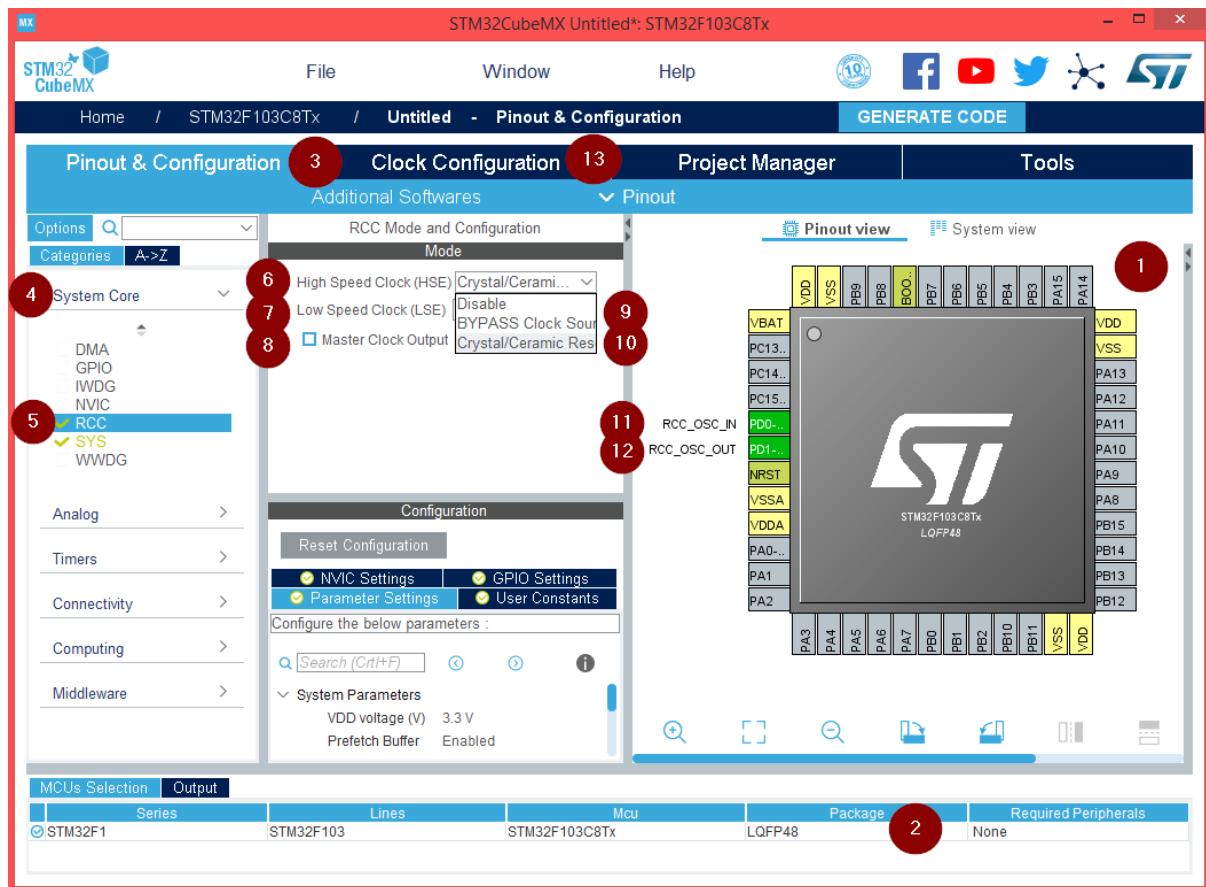
سرعت یا فرکانس بالا (گزینه ۱) از نوع RC که در اکثر میکروکنترلرهای موجود میباشد و مقدار آن غیرقابل تغییر است و دیگری Input frequency (گزینه ۲) که مربوط به همان Crystal/Ceramic Resonator میباشد که فعال کردیم (گزینه ۱۰ شکل ۲)، همچنین محدوده مقدار انتخاب کریستال در زیر آن نوشته شده است (4-16MHz) که مقدار پیشفرض آن ۸ میباشد. همانطور که مشاهده میکنید ماکریم فرکانسی کاری (HCLK) که میتوانیم در این میکرو داشته باشیم نهایتا 72MHz (گزینه ۳) است و ما هم میخواهیم از آن استفاده کنیم به همین دلیل داخل مربع ۷۲ را وارد میکیم و Enter میزنیم، و با پیغامی که در شماره ۴ نشان داده شده مواجه میشویم که مفهوم آن اینست که با توجه به تنظیمات موجود (منابع فرکانسی و اعداد وارد شده) نمیتوان این فرکانس را لحاظ کرد، آیا مایلید که تنظیمات اصلاح شود؟ OK را میزنیم تا بطور اتوماتیک تنظیمات اصلاح گردد.

نکته: مختصر توضیحی در مورد نحوه اصلاح تنظیمات(شکل ۵): منبع کلاک را خارجی انتخاب کرده (گزینه ۱) سپس با عبور از یک تقسیم کننده (گزینه ۲) وارد یک مالتی پلکسیر گردیده و خروجی آن که ۸ میباشد (گزینه ۳) وارد بلوک<sup>۱</sup> PLL (گزینه ۴) میشود و برای اینکه به فرکانس ۷۲ (گزینه ۶) برسد در ۹ (گزینه ۵) ضرب میگردد.

---

<sup>1</sup> High Clock Speed

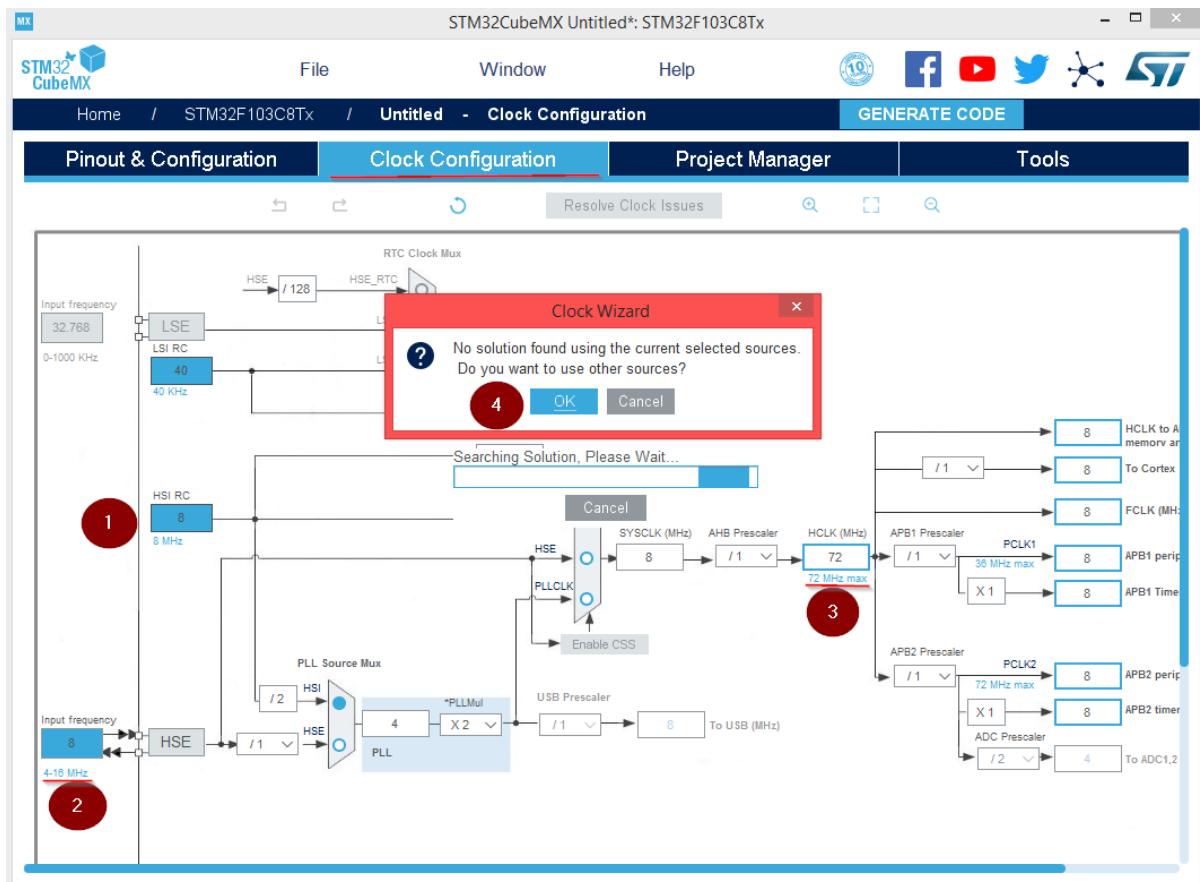
<sup>2</sup> Phase Lock Loop



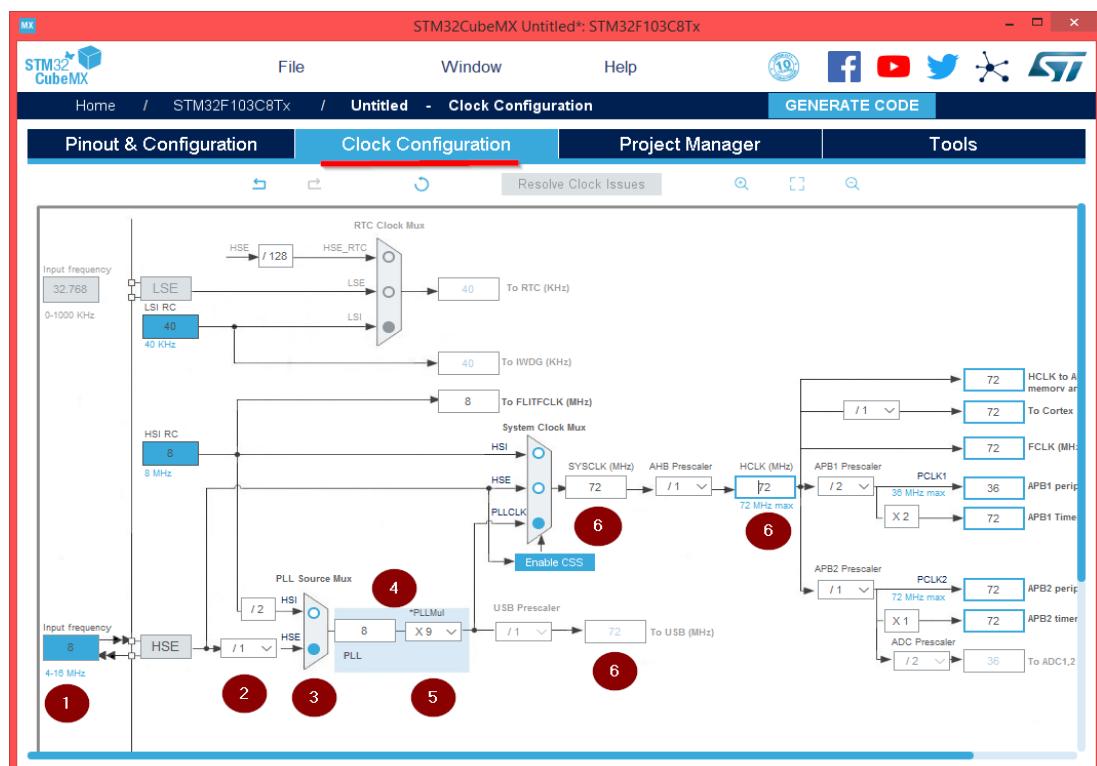
(شکل ۲)



(شکل ۳)

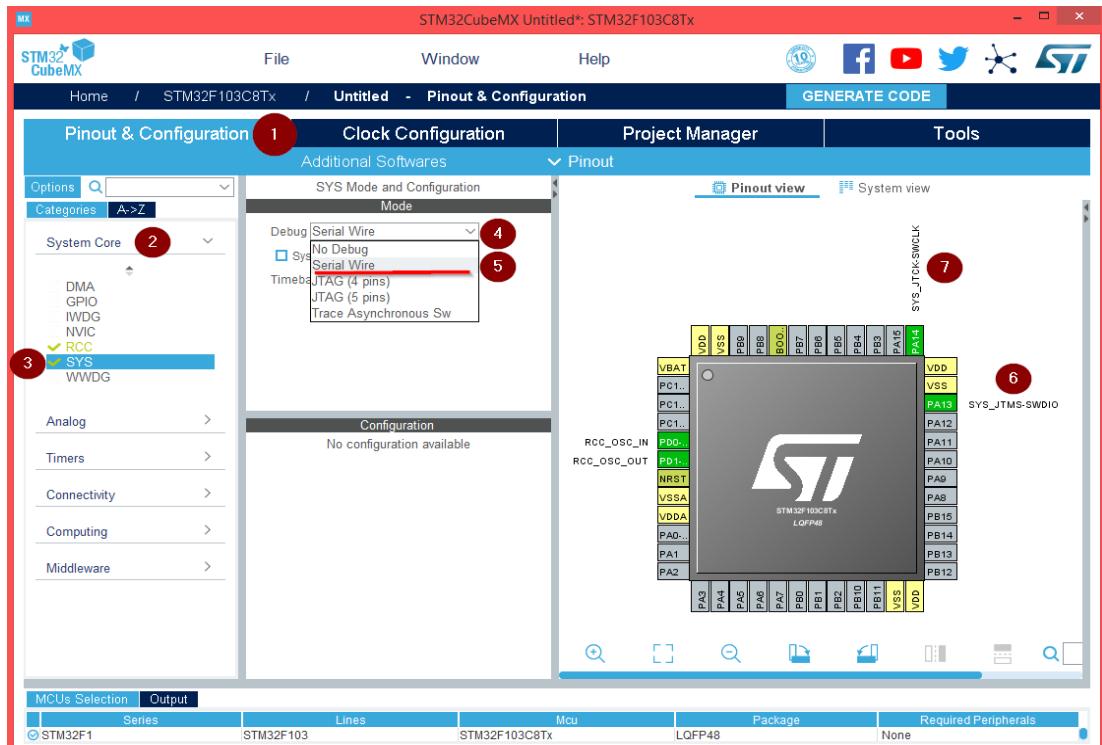


(٤) شكل



(٥) شكل

ب- تنظیمات مربوط به Pinout & Configuration (گزینه ۱) در قسمت SYS (شکل ۶): در تب Pinout & Configuration (گزینه ۲) بر روی زیر شاخه آن SYS (گزینه ۳) کلیک می‌کنیم، سپس Debug (گزینه ۴) را بر روی Serial Wire (گزینه ۵) تنظیم می‌کنیم. همانطور که مشاهده می‌کنیم دو پایه (گزینه ۶ و ۷) به آن اختصاص یافته است که از آن، جهت Load کردن برنامه با استفاده از Debuger (گزینه ۸) به انجام می‌شود.



(شکل ۶)

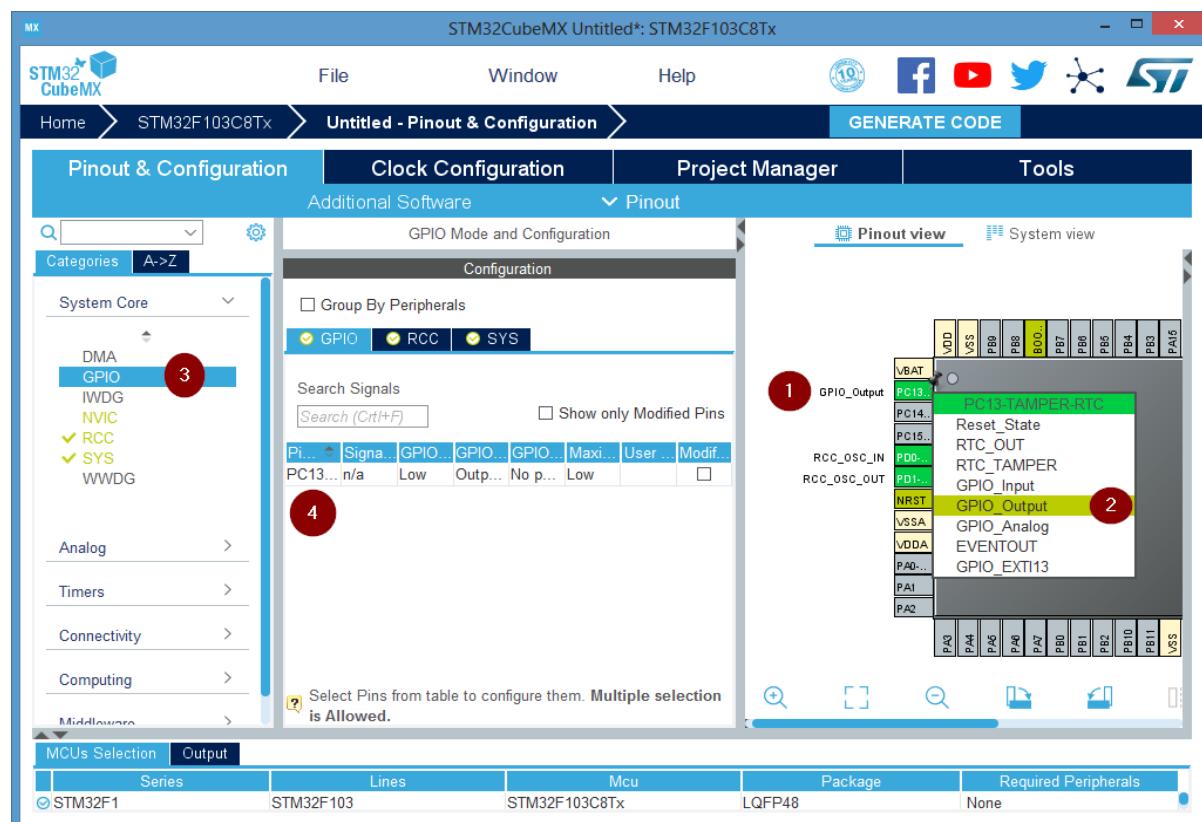
۴- حال بر روی پایه PC13 (گزینه ۱ شکل ۷) کلیک می‌کنیم و GPIO\_Output را به عنوان پورت خروجی انتخاب می‌کنیم. سپس در تب Pinout & Configuration (گزینه ۳) را انتخاب می‌کنیم که در صفحه باز شده بر روی PC13 (شکل ۴) کلیک می‌کنیم تا تنظیمات آن باز شود (شکل ۸) که: گزینه ۱ Output level: دارای دو مقدار High و Low می‌باشد که تعیین کننده سطح منطقی پایه خروجی (صفر یا یک منطقی) در زمان شروع به کار برنامه می‌باشد. در اینجا با توجه به اینکه LED مورد استفاده موردنی استفاده در بورد آماده STM32F103C8T6 پایه VDD یا یک منطقی وصل می‌باشد با صفر شدن پایه، LED روشن می‌شود و برعکس. حال با فرض اینکه می‌خواهیم LED در هنگام شروع به کار و قبل از سال دستور خاموش باشد لذا Output level آن را بر روی High تنظیم می‌کنیم.

گزینه ۲ - یکی حالت Open Drain Mode: که پایه درین مدار باز می‌شود و جریانی را عبور نمی‌دهد و صرفاً کنترل ولتاژ خروجی را انجام می‌دهد. حالت دیگر Push Pull می‌باشد که در مواردی مورد استفاده قرار می‌گیرد که خروجی ما به یک مصرف کننده مانند LED اتصال داشته باشد که نیاز به جریان کشی دارد. لذا آن را بروی Push Pul تنظیم می‌کنیم (مطابق توضیحات قسمت ۲-۴).

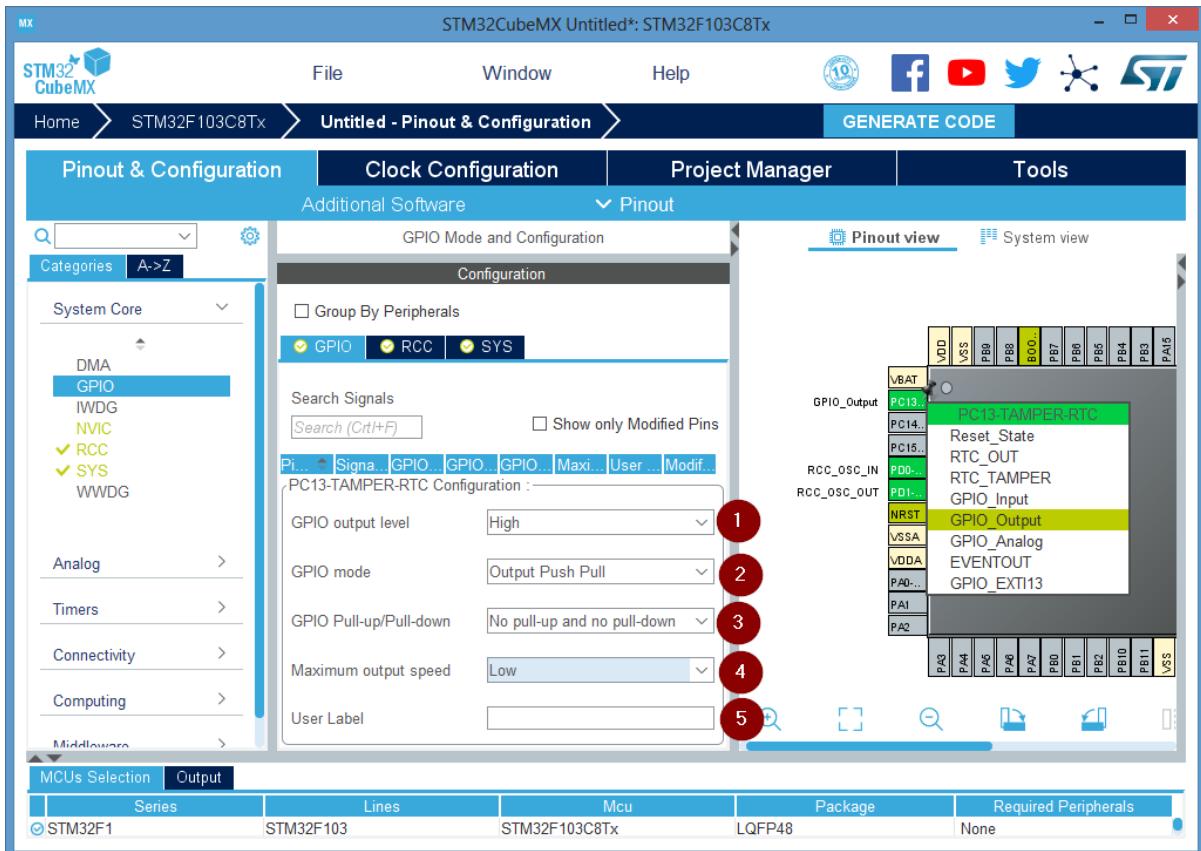
گزینه ۳ Pull-up/Pull-down: - پایه میکرو وقتی به عنوان پین ورودی انتخاب می‌گردد، در حالتی که پایه آزاد می‌باشد مقدار آن با استفاده از مقاومت Pull-Up یک منطقی یا با استفاده از مقاومت Pull-Down صفر منطقی تنظیم می‌گردد. ولی با توجه به اینکه ما این پایه را در حالت خروجی بررسی می‌کنیم مقدار این قسمت مهم نیست.

گزینه ۴ Maximum out-put speed: - بیشترین فرکانس تغییر حالت (سوئیچینگ) برای پایه مورد نظر است. که شامل سه گزینه Low و Medium و High می‌باشد، که مقادیر فرکانسی متناظر با آنها بسته به نوع میکروکنترلر متفاوت است. هرچه بیشینه فرکانس سوئیچینگ را افزایش دهیم، مصرف توان نیز افزایش می‌یابد. که در اینجا مقدار آن را Low قرار می‌دهیم.

گزینه ۵ User label: - تو سط این فیلد می‌توان به پایه، اسمی را نسبت داد تا در استفاده‌های بعدی کار کردن با این نام انجام گیرد.



(شکل ۷)



(شکل ۸)

۵- حال که تنظیمات به اتمام رسیده بر روی تب Project Manager (گزینه ۱ شکل ۹) کلید می‌کنیم. در قسمت Project Name (گزینه ۲) نام پروژه، در قسمت Project Location (گزینه ۳) پوشه مدنظر را انتخاب می‌کنیم و در قسمت Toolchain/IDE MDK-ARM (گزینه ۴) و در قسمت Min Version ، V5 را انتخاب می‌کنیم سپس بر روی Code Generator (گزینه ۶) کلیک می‌کنیم. در این بخش در مورد نحوه بکارگیری کتابخانه‌ها بحث می‌شود:

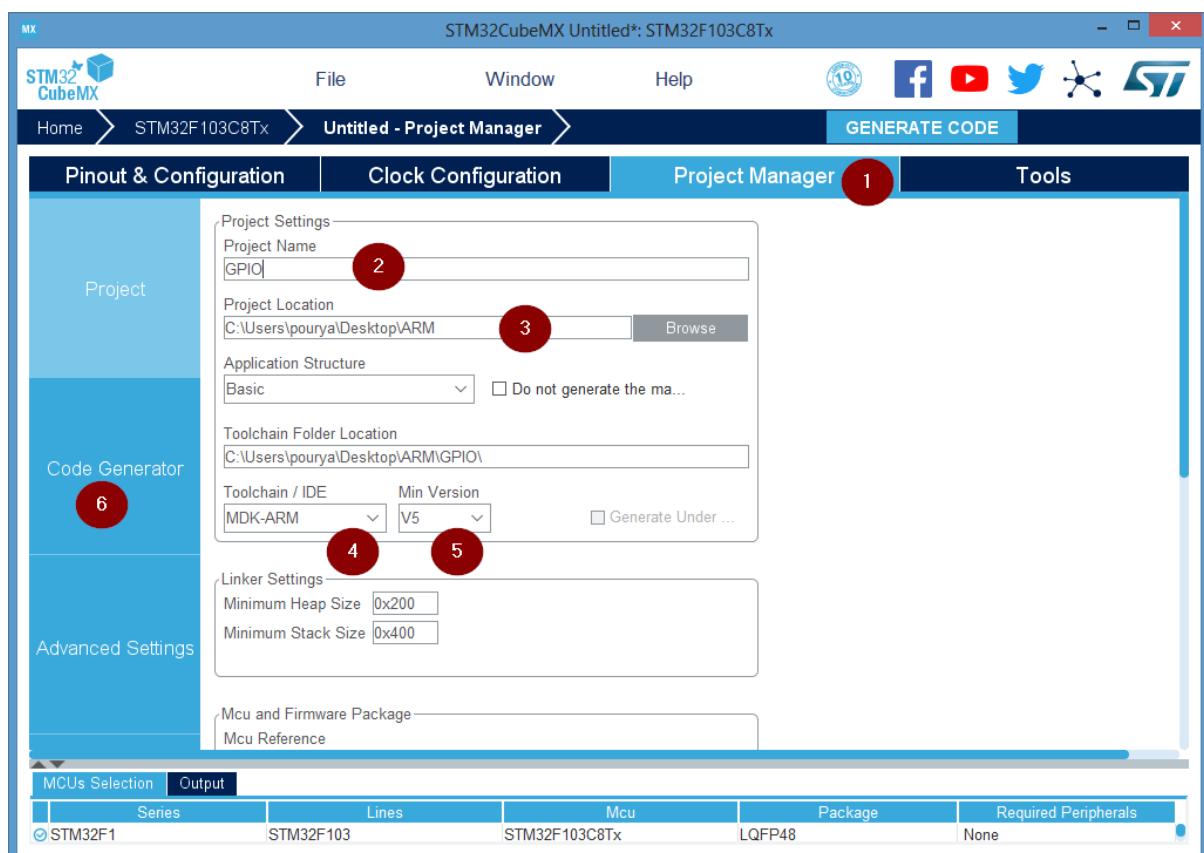
گزینه ۱- در این حالت تمام کتابخانه‌های میکروکنترلر اضافه می‌گردد که حجم فایل ایجادی را خیلی زیاد می‌کند.

گزینه ۲- در این حالت تنها کتابخانه‌های مورد نیاز اضافه می‌شود که حجم فایل ایجادی مقدار مناسب و معقولی می‌شود.

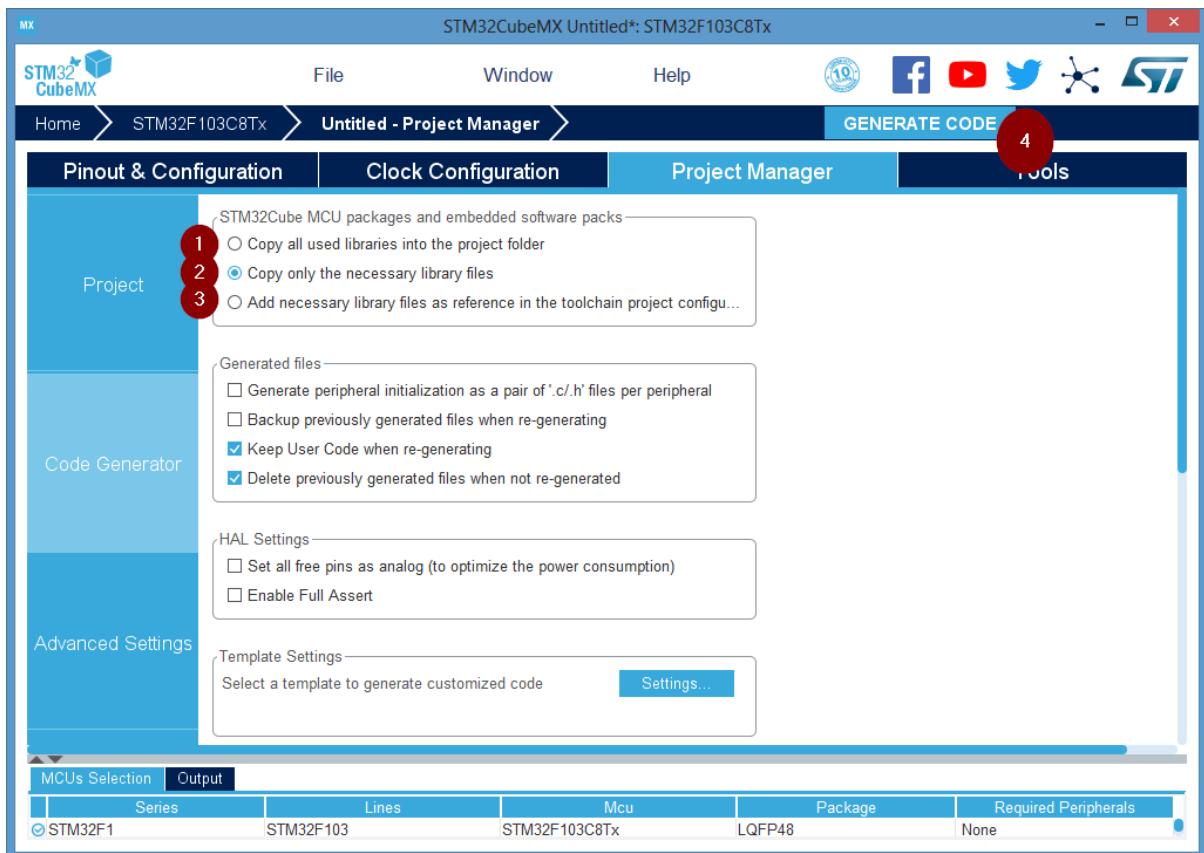
گزینه ۳- در این حالت از کتابخانه‌های موجود که در کامپیوتر ذخیره شده استفاده می‌شود و در صورت انتقال به کامپیوتر دیگر به دلیل تغییر آدرس قابلیت اجرا ندارد ولی حجم فایل ایجادی خیلی کم می‌باشد.

پس بهترین انتخاب حالت دوم می‌باشد.

۶- در پایان بر روی Generate Code (گزینه ۴) کلیک می‌کنیم تا پروژه ساخته شده و بطور اتوماتیک برنامه Keil باز می‌شود.

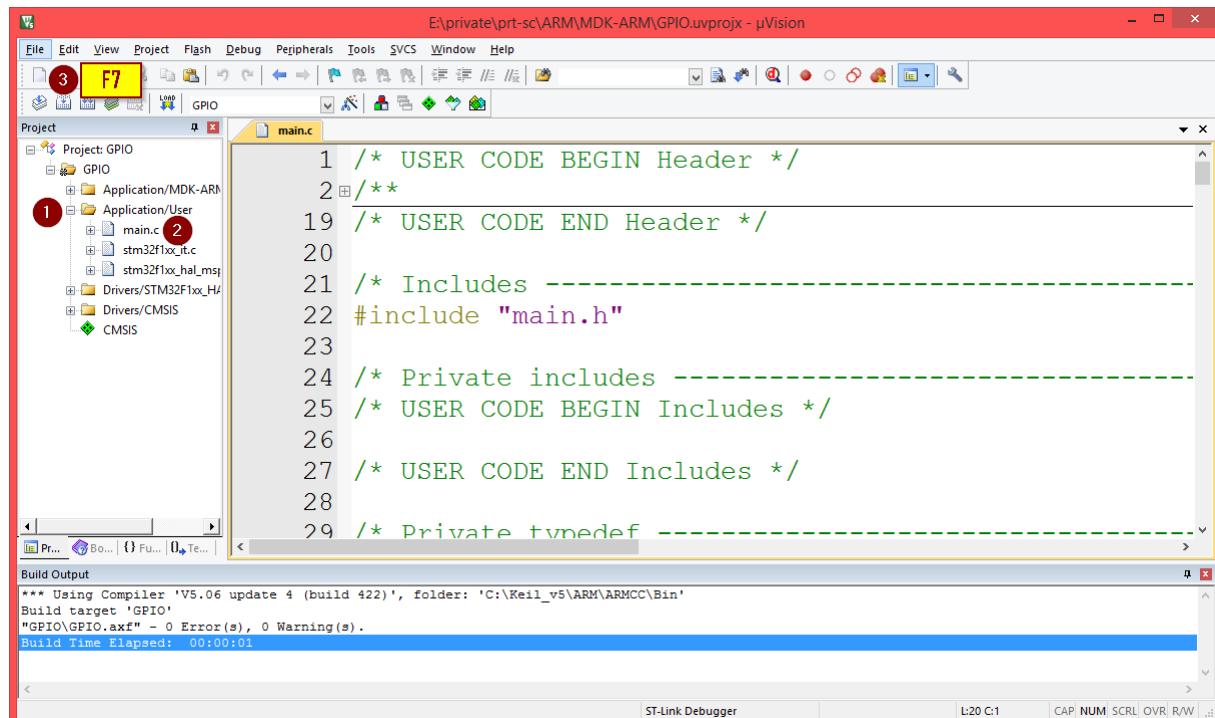


(۹) شکل



(شکل ۱۰)

۷- پس از باز شدن Keil از زیر شاخه Application/User فایل Main را باز می کنیم، مشاهده می کنیم که تمامی کتابخانه ها، ماثولها، پورت ها و تنظیمات آنها فراخوانی شده و دیگر مانند فصل سوم که بدون cub برنامه نوشتم نیاز به انجام این همه تنظیمات و تایپ که امکان اشتباه کردن هم بیشتر می شد وجود ندارد. قبل از انجام هر کاری ابتدا برنامه را Build یا کامپایل (F7) می کنیم (شکل ۱۱).



(شکل ۱۱)

#### ۴-۱-۳ نوشتن برنامه نویسی در Keil

نکته: تمام کدهایی که می‌نویسیم بهتر است بین :

```
/* USER CODE BEGIN */
/* USER CODE END */
```

با شد چون ممکن است برای اضافه کردن پارامتری دوباره به Cube برگردیم، دوباره پروژه را Generate کنیم، که در این صورت باعث پاک شدن کدهایی می‌شود که از این قانون پیروی نکرده است .

۸- حال می‌خواهیم توابعی را که می‌توانیم برای GPIO درنظر بگیریم بیان کنیم :

**الف()** : این تابع، مقداری را بر روی پین (که از قبل بصورت خروجی تعریف شده است) قرار می‌دهد. این تابع دارای سه آرگومان بوده که اولی، نام GPIO مورد نظر (نام پورت پایه مورد نظر – مثلاً GPIOA یا GPIOB یا ...)، آرگومان دوم، شماره پایه مورد نظر ( بصورت X\_GPIOC یا ...)، آرگومان سوم، وضعیت ۰ یا ۱ بودن خروجی است. دقت شود که برای که X شماره پایه است) و نهایتاً آرگومان سوم، وضعیت ۰ یا ۱ استفاده کرد و هم از توابع معادل آنها که بترتیب GPIO\_PIN\_RESET و GPIO\_PIN\_SET است. بعنوان مثال، هر یک از دو دستور زیر بطور یکسان، پایه C13 را روشن (۱)

می‌کنند:

```
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, 1);
```

```
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
```

ب) HAL\_GPIO\_ReadPin() : این تابع، مقدار دیجیتالی موجود بر روی پایه را می‌خواند. تابع دارای دو آرگومان بوده که به ترتیب نام پورت و شماره پین هستند. بعنوان مثال، دستور زیر نحوه قرائت یک ورودی از پایه A1 است:

```
HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1);
```

پ) HAL\_GPIO\_TogglePin(): این تابع، مقدار موجود بر روی پایه را معکوس می‌کند. واضح است که تنها دو آرگومان برای این تابع مورد نیاز است که یکی نام GPIO مورد نظر و دیگری، شماره پایه است. بعنوان نمونه، تابع زیر پایه B3 را معکوس می‌کند:

```
HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1);
```

نکته: در دو تابع WritePin و TogglePin در صورتیکه بخواهیم بیش از یک پایه (از یک پورت) را write و يا toggle کنیم، می‌توان بجای چندبار استفاده از توابع مذکور، به کمک اپراتور | (یا منطقی) تمامی پایه‌های مورد نظر را در یک خط دستور بصورت مثال زیر نوشت:

```
HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_1 | GPIO_PIN_2 |  
GPIO_PIN3);  
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1 | GPIO_PIN_2 |  
GPIO_PIN3, GPIO_PIN_SET);
```

۹- حال در داخل حلقه While(1)، بین USER CODE BEGIN و USER CODE END کد دستوری برای چشمک زدن LED مینویسیم (شکل ۱۲). برای این کار میتوانیم از دستور Toggle (شکل ۱۱) و یا Write (شکل ۱۲) استفاده کنیم، دستور Delay نیز مقدار زمان روشن و خاموش بودن LED را مشخص می‌کند.

The screenshot shows the µVision IDE interface with the project 'GPIO' open. The main window displays the 'main.c' source code. The code is as follows:

```

95  /* USER CODE BEGIN WHILE */
96  while (1)
97  {
98      /* USER CODE END WHILE */
99
100     /* USER CODE BEGIN 3 */
101
102     HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
103     HAL_Delay(500);
104
105 }
106 /* USER CODE END 3 */
107
108 /**
109 * @brief System Clock Configuration
110 * @retval None

```

The line `HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);` is highlighted with a blue box. The build output window at the bottom shows the compilation process.

(١١) شکل

The screenshot shows the µVision IDE interface with the project 'GPIO' open. The main window displays the 'main.c' source code. The code is as follows:

```

95  /* USER CODE BEGIN WHILE */
96  while (1)
97  {
98      /* USER CODE END WHILE */
99
100     /* USER CODE BEGIN 3 */
101
102     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
103     HAL_Delay(500);
104     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
105     HAL_Delay(500);
106
107 }
108 /* USER CODE END 3 */
109
110 /**
111 */

```

The lines `HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);` and `HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);` are highlighted with a blue box. The build output window at the bottom shows the compilation process.

(١٢) شکل

۱۰- حال می خواهیم نور LED را کنترل کنیم:

در این صورت ابتدا مقدار Delay را کاهش می‌دهیم تا تغییرات محسوس شوند (بطور مثال کل زمان روشن و خاموش شدن را ۲۰ در نظر میگیریم). هر قدر مدت زمان خاموش بودن LED را بیشتر کنیم، نور LED کم‌رنگ‌تر می‌شود و بلعکس. علت آن نیز این می‌باشد که سرعت روشن و خاموش شدن آنقدر زیاد است که متوجه نمی‌شویم و آن را به صورت پیوسته درک می‌کنیم (کد ۱).

نکته: با همین روش می‌توان یک موتور DC را کنترل کرد.

بافرض اینکه ۵ms روشن (SET) و ۱۵ms خاموش (RESET) باشد شدت روشنایی LED ۲۵٪ مقدار نامی آن می‌باشد.

$$\frac{5}{5 + 15} \times 100 = \%25$$

```
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
HAL_Delay(5);
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
HAL_Delay(15);
```

کد ۱

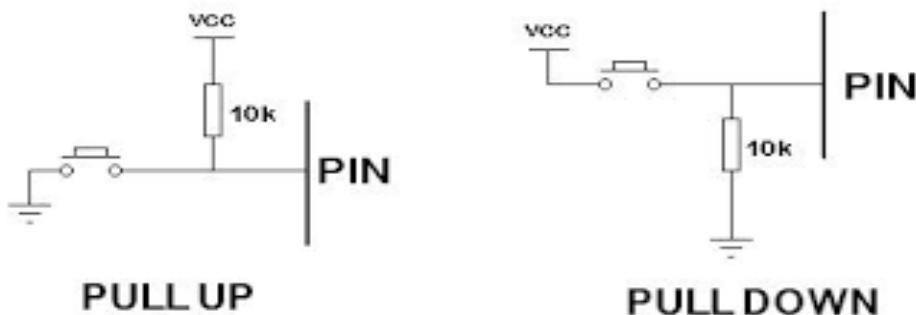
حال می‌خواهیم کدی را بنویسیم که با هر بار فشردن کلید، یک واحد به آن اضافه شود :

۱- Cube را باز می‌کنیم، تنظیمات قبلی را نگه می‌داریم و تنها PortA0 را با کلیک بر روی آن به عنوان GPIO\_Input انتخاب می‌کنیم.

۲- حال برای GPIO در حالت ورودی نیز با گزینه‌های زیر مواجهیم:

GPIO mode: که حالت ورودی را تنظیم می‌کند و (بر خلاف حالت خروجی) تنها شامل یک گزینه Input mode است (گزینه ۳).

میکروکنترلر AVR ST بر خلاف AVR که تنها از مقاومت‌های داخلی Pull-up/Pull-down برهه می‌برند، دارای هر دو مقاومت داخلی Pull-up و Pull-down می‌باشد، که توسط این منوی کشویی می‌توان نوع آنرا انتخاب کرد. می‌دانیم زمانیکه سمت دیگر کلید متصل به پایه ورودی به VCC متصل باشد، از مقاومت (داخلی یا خارجی) Pull-down و زمانیکه به GND متصل باشد، از مقاومت Pull-up استفاده می‌کنیم (شکل ۱۳)، تا از شناور (float) شدن پایه در حالت قطع کلید و نویزی شدن آن جلوگیری شود (گزینه ۴).

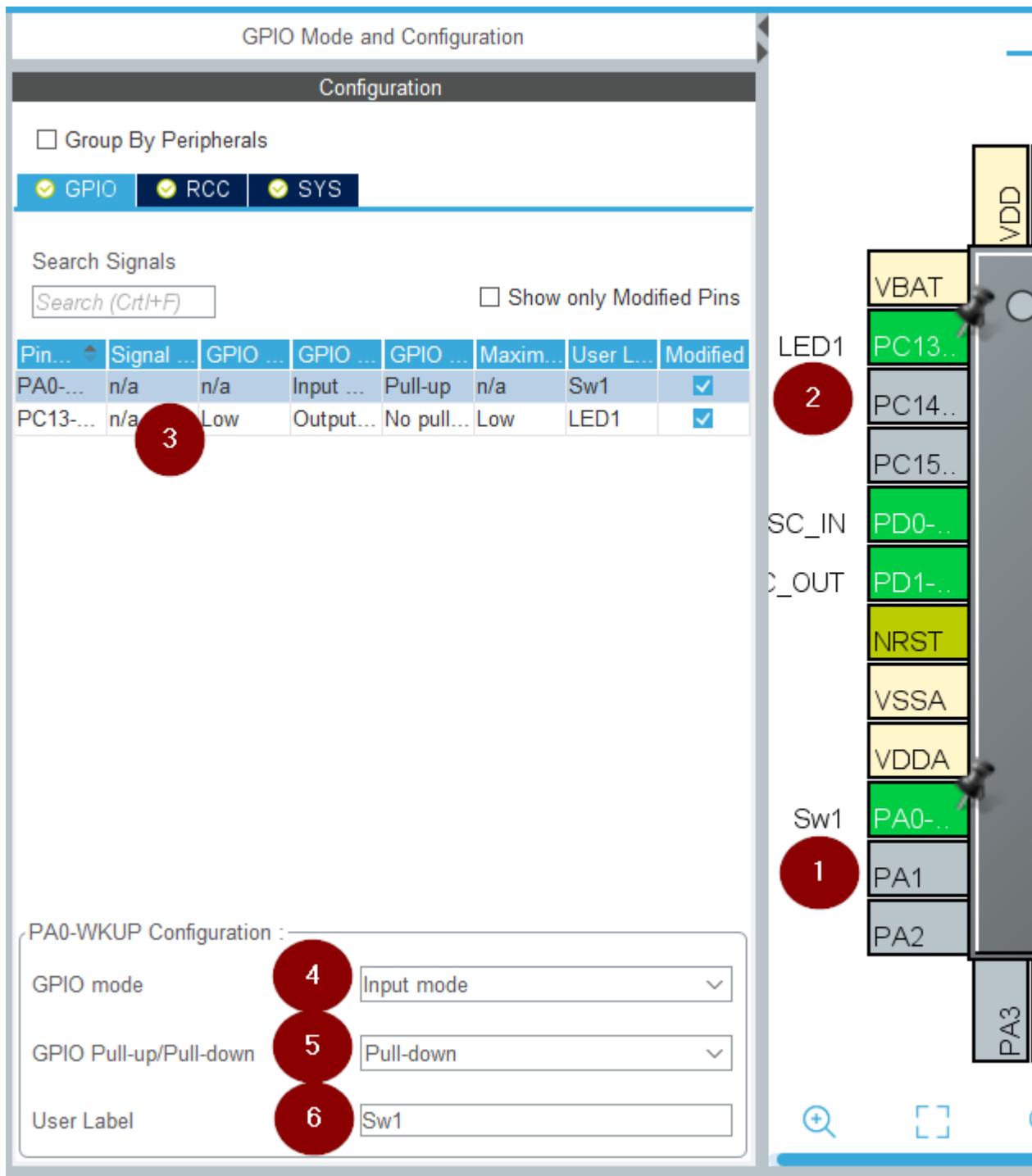


(شکل ۱۳)

User Label : جهت افزایش خوانایی برنامه و همچنین سهولت مدیریت پایه‌ها، می‌توان به آنها برچسب‌هایی را مناسب نمود. برای این منظور، کافیست در محیط STM32Cube MX بر روی پایه مورد نظر در شماتیک میکروکنترلر کلیک راست کرده و گزینه Configuration گزینه Enter User Label را انتخاب نمود یا در تب گزینه ۶ آن را وارد کنیم(شکل ۵). سپس نام دلخواه (با توجه به قوانین نام‌گذاری متغیرها) را به آن اختصاص داد. من بعد در محیط کدنویسی Keil هر زمانیکه بخواهیم نام پورت مربوط به پایه مورد نظر (A، B، C و...) را وارد کنیم، از عبارت Label\_GPIO\_Port استفاده می‌کنیم، که در آن، همان نام تخصیص یافته به پایه مورد نظر است. همچنین زمانیکه بخواهیم شماره پین مورد نظر را وارد کنیم، از عبارت Label\_Pin بجای شماره پایه استفاده می‌کنیم. برای درک بهتر موضوع به مثال زیر توجه کنید. دو دستور کاملاً معادل هستند:

```
HAL_GPIO_WritePin (GPIOG , GPIO_PIN_13, GPIO_PIN_SET);
HAL_GPIO_ReadPin (GPIOA , GPIO_PIN_0);
```

```
HAL_GPIO_WritePin (Label_GPIO_Port , Label_Pin ,
GPIO_PIN_SET);
HAL_GPIO_ReadPin (Label_GPIO_Port , Label_pin);
```



(شکل ۱۴)

- ۳- پروژه را Generate می‌کنیم تا وارد محیط برنامه نویسی Keil شویم.
- ۴- حال باید یک شرط در While(1) بنویسیم که هر بار کلید زده شد یک واحد به شمارنده اضافه شود (کد ۱) بنابراین ابتدا یک Counter به عنوان متغیر در ابتدای برنامه قبل از Main تعریف می‌کنیم.

```
uint32_t MyCounter;
```

```

while (1)
{
    if (HAL_GPIO_ReadPin(Sw1_GPIO_Port, Sw1_Pin==GPIO_PIN_SET) )
    {
        MyCounter++;
    }
}

```

#### ۴-۱-۴ محیط دیباگ

در نرم افزار Keil محیطی به نام Debug تعییه شده است که می توان پارامترهای مختلف میکرو را در حین اجرا آن مشاهده نمود.

به شرح مختصری از این قابلیت Keil می پردازیم:

گزینه ۱- باز زدن این دکمه به محیط دیباگ می رویم (Ctrl+ F5)

گزینه ۲- میکرو را Reset میکند.

گزینه ۳- برنامه Load شده را Run می کند. (F5)

گزینه ۴- برنامه در حال اجرا را متوقف می کند.

گزینه ۵- حالت های مختلف اجرایی برنامه می باشند. بطور مثال این قابلیت را ایجاد می کند که برنامه خط به خط با دستور ما حرکت کند یا اینکه از حلقه ای خارج شود.

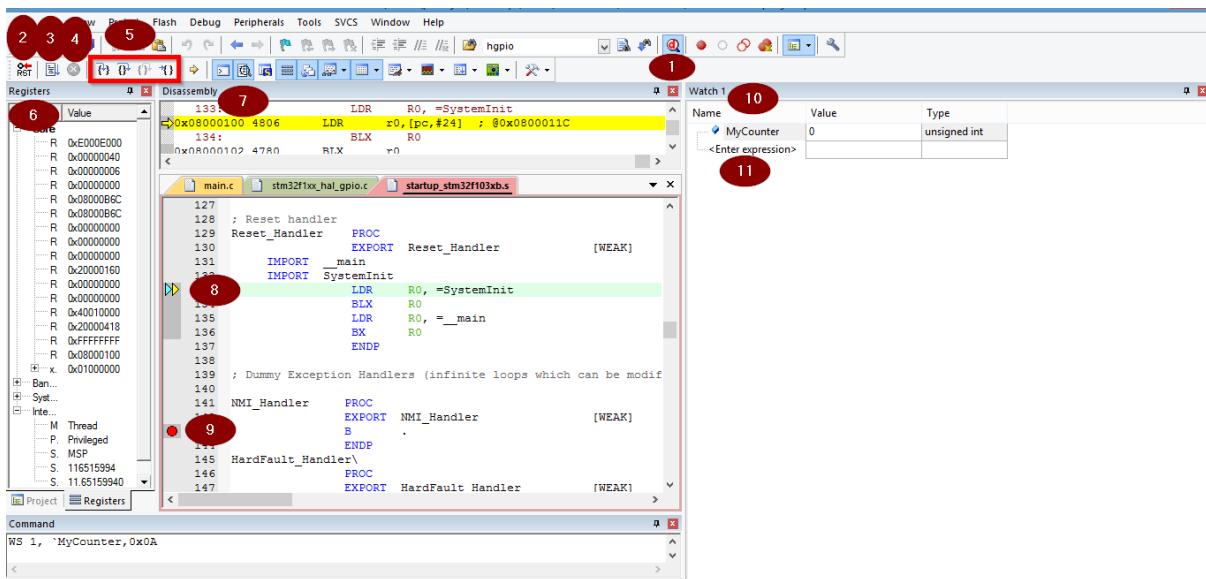
گزینه ۶ Registers -های میکرو را به صورت کد Hex نمایش می دهد.

گزینه ۷- کد برنامه را به زبان اسمبلي نشان می دهد. و فلش زرد رنگ نشان دهنده خطی است که برنامه در آن قرار دارد.

گزینه ۸- خطی است که برنامه در آن قرار دارد.

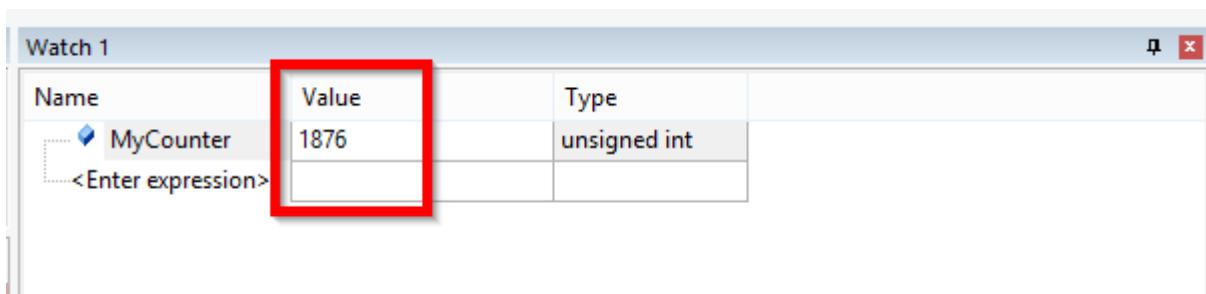
گزینه ۹- آن نقطه قمز را Break Point می گویند. با گذاشتن آن در هر قسمت از برنامه، برنامه در همان خط می ایستد.

گزینه ۱۰ Watch مهم ترین قسمت برنامه می باشد که در این پنجره با وارد نمودن متغیرها، رجیسترها و سایر پارامترهایی (گزینه ۱۱) که مقدار دارند می توان مقدار آن را بصورت RealTime مشاهده نمود.



(شکل ۱۵)

همانطور که مشاهده می‌کنید (شکل ۱۶) با یکبار فشار دادن کلید عدد آن به مقدار قابل توجهی زیاد می‌شود، دلیل؟ همانطور که میدانیم سرعت کاری میکرو 72MHz می‌باشد که نشان می‌دهد خطوط کد ما با چه سرعتی در حال اجراست، با توجه به اینکه کدهای داخل While(1) پیوسته در حال اجراست، مدت زمانی که طول می‌کشد ما دکمه را فشار دهیم و آن را آزاد کنیم، آن خط که Read می‌کند چندین هزار بار بلکه بیشتر تکرار می‌شود.



(شکل ۱۶)

راه حل:

اولین و راحتترین راه این است که یک مقدار Delay به داخل if اضافه کنیم که مشخص کننده مدت زمانی است که طول می‌کشد ما دکمه را فشار دهیم و آن را آزاد کنیم که دارای مشکلاتی از جمله:  
الف- تاخیر انتخاب شده یک مقدار نسبی است و به سرعت دکمه زدن کاربر، بستگی دارد.

ب- ممکن است کاربر انگشت خود را به هر دلیلی دیر بلنده کند که در این صورت باز هم مشکل قبلی پابرجاست.

پ- ممکن است در قسمتی از برنامه باشیم که مثلا میکروکنترلر داخل حلقه for یا While قرار دارد و کاربر دکمه را فشار می دهد که در این صورت میکروکنترلر متوجه این اتفاق نمی شود.

ت- بیخود میکروکنترلر را در جایی متوقف کرده ایم در حالیکه ممکن است در جایی دیگر نیاز باشد میکرو کارهای دیگری انجام دهد، به عبارتی میکروکنترلر با این سرعت و کاری بالا را جهت یک کار ساده متوقف کرده ایم.

بنابراین این روش مطلوب و قابل اعتمادی نیست.

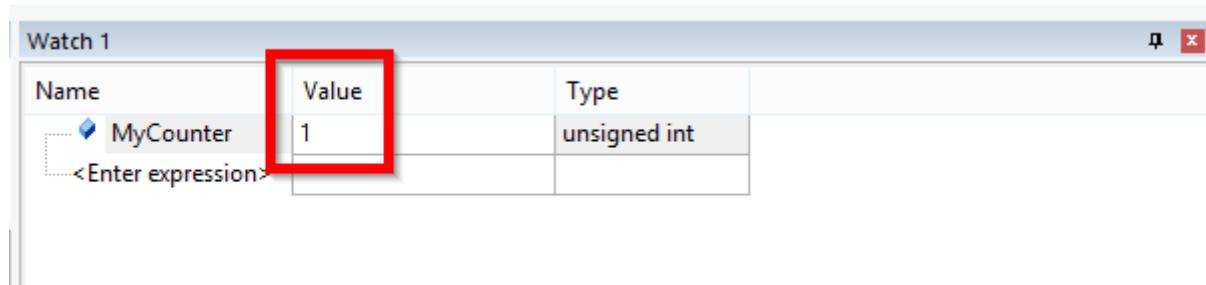
```
while (1)
{
    if (HAL_GPIO_ReadPin(Sw1_GPIO_Port, Sw1_Pin==GPIO_PIN_SET))
    {
        MyCounter++;
        HAL_Delay(100);
    }
}
```

برای حل مشکل اول و دوم باید راه حلی پیدا کنیم که بفهمیم کاربر انگشت خودرا از روی کلید برداشته است یا خیر.

برای این کار باید داخل if یک حلقه While با این شرط بنویسیم که، تاوقتی که انگشت کاربر روی کلید است هیچ کاری انجام ندهد

```
while (1)
{
    if (HAL_GPIO_ReadPin(Sw1_GPIO_Port, Sw1_Pin==GPIO_PIN_SET))
    {
        while(HAL_GPIO_ReadPin(Sw1_GPIO_Port, Sw1_Pin==GPIO_PIN_SET))
    }
}
```

```
    {
    }
    MyCounter++;
}
}
```



برای حل مشکل ب و ت به فصل بعدی مراجعه می کنیم.

# **فصل ۵**

## **وقفه خارجی**

### **External Interrupt**

## ۱-۵ وقفه خارجی(External Interrupt)

یکی دیگر از Peripheral interrupt های میکرو کنترلر External interrupt است و وظیفه آن چک کردن پایه میکرو کنترل به صورتی است که هر اتفاقی برای میکرو کنترلر یافتد به ما اطلاع دهد، البته نوع آن اتفاق را ما مشخص می کنیم. در میکرو کنترلرهای ST تعداد ۱۶ خط وقفه خارجی (Interrupt line) موجود است. معماری خط وقفه در میکرو کنترلرهای ARM ، به اینصورت است که تمامی پایه های IO قابلیت تبدیل شدن به وقفه خارجی را دارند. البته این بطور همزمان (در یک پروژه) نیست. به عبارت دیگر، تمامی پایه های شماره ۰ با هم یک خط وقفه را تشکیل می دهند (خط وقفه شماره ۰ – که شامل تمامی پورت ها می شود مثلا ... A0,B0,C0,D0...) به همین ترتیب، خطوط وقفه شماره ۱، ۲، ۳ و ... الی ۱۵ تعریف شده است. در هر زمان می توان تمامی ۱۶ خط وقفه را بکار گرفت، اما در هر خط، فقط یکی از پایه ها می تواند استفاده شود. از این رو، بعنوان مثال، پایه های A0 و B0 نمی توانند بطور همزمان برای وقفه خارجی منظور شوند (چرا که هردو از خط وقفه صفر هستند)، اما پایه های A3 و B2 و B4 می توانند بطور همزمان برای وقفه خارجی بکار روند.

حال Cube پروژه پیشین را باز می کنیم و اینبار کلید ورودی را در حالت اینترپتی فعال می کنیم.  
۱- بر روی A0 کلیک کرده و آن را در حالت EXTI0 (اینترپت خارجی) فعال می کنیم (گزینه ۱). سپس به تنظیمات تب Configuration در قسمت GPIO رفته و پایه PA0 (گزینه ۲) را که در حالت اینترپ خارجی تنظیم شده انتخاب می کنیم و تنظیمات زیر را انجام می دهیم (شکل ۱):

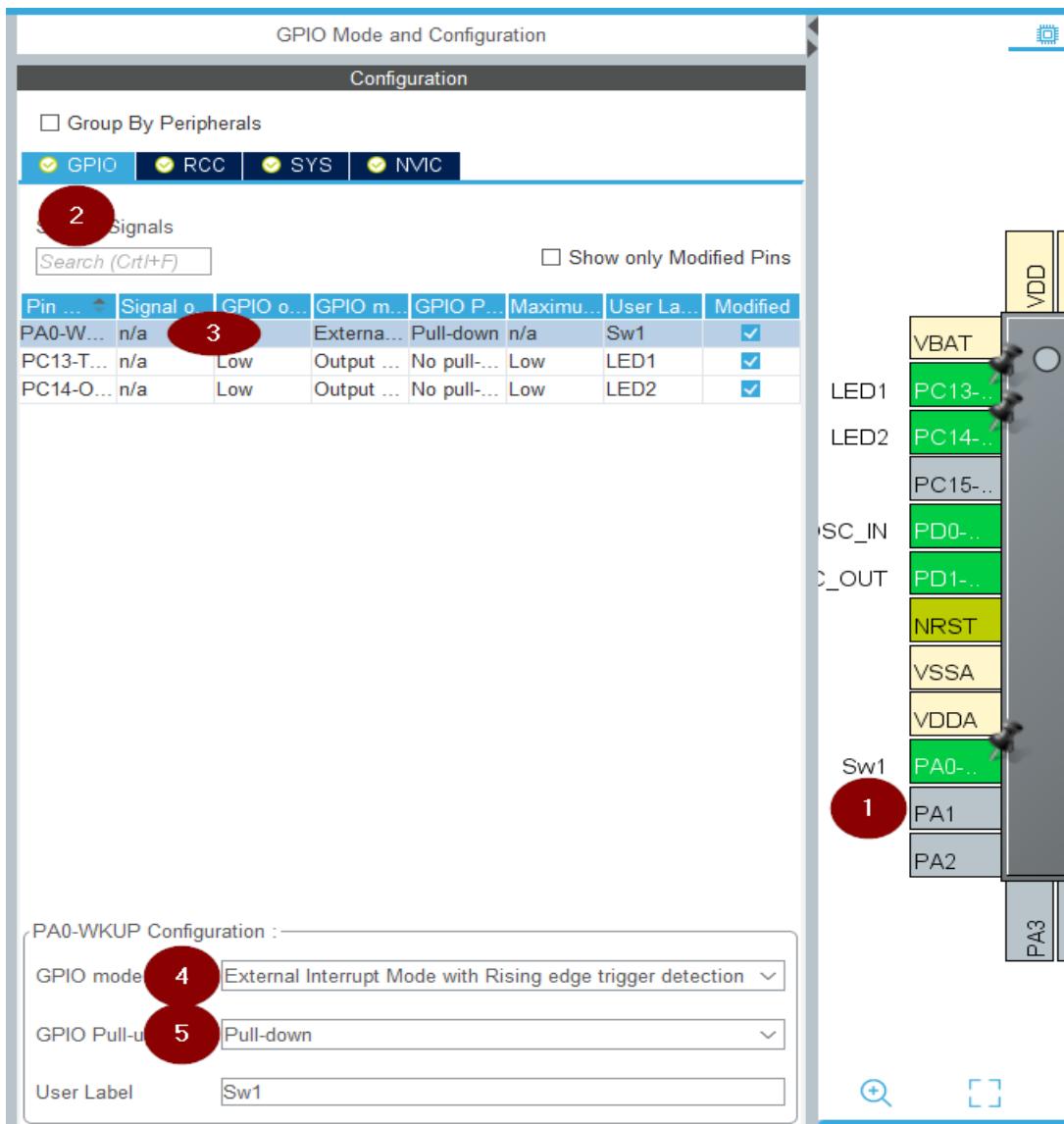
الف GPIO mode: در این قسمت می توان نوع حساسیت وقفه خارجی را تعیین نمود. گزینه هایی که در این منوی کشویی قابل انتخاب هستند، شامل دو دسته کلی External Event و External Interrupt می باشند. با دسته دوم فعلاً کاری نداریم. دسته اول نوع تحریک (trigger) وقفه های خارجی را مشخص می کند که خود به سه حالت تقسیم می شود که شامل (گزینه ۴):

Rising edge: حساس به لبه بالارونده می باشد بطوریکه پایه در صورت احساس لبه بالارونده به ما اطلاع می دهد (تغییر وضعیت یک منطقی به صفر منطقی).

Falling edge: حساس به لبه پایین رونده می باشد بطوریکه پایه در صورت احساس لبه پایین رونده به ما اطلاع می دهد (تغییر وضعیت صفر منطقی به یک منطقی).

Rising & Falling edge: حساس به هردو لبه پایین رونده و بالارونده می باشد بطوریکه پایه چه در صورت احساس لبه بالارونده و چه لبه پایین رونده به ما اطلاع می دهد (تغییر وضعیت یک منطقی به صفر منطقی و برعکس).

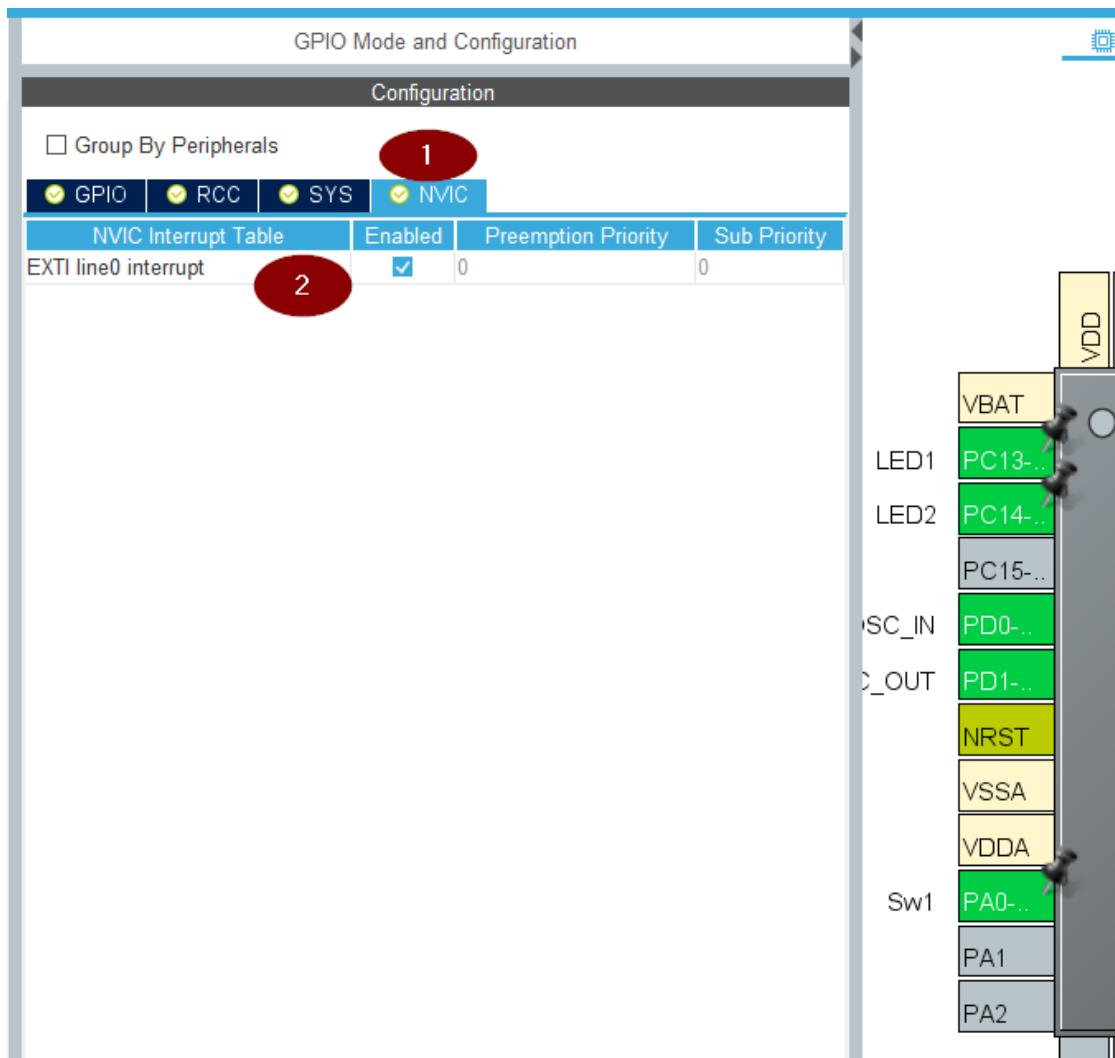
ب - همانند تنظیمات ورودی معمولی می باشد(گزینه ۵).



(شکل ۲)

۲ - سپس با مراجعه به سربرگ Configuration و انتخاب تب<sup>۱</sup> NVIC اینتراپت این پایه را نیز فعال می نماییم(شکل ۲).

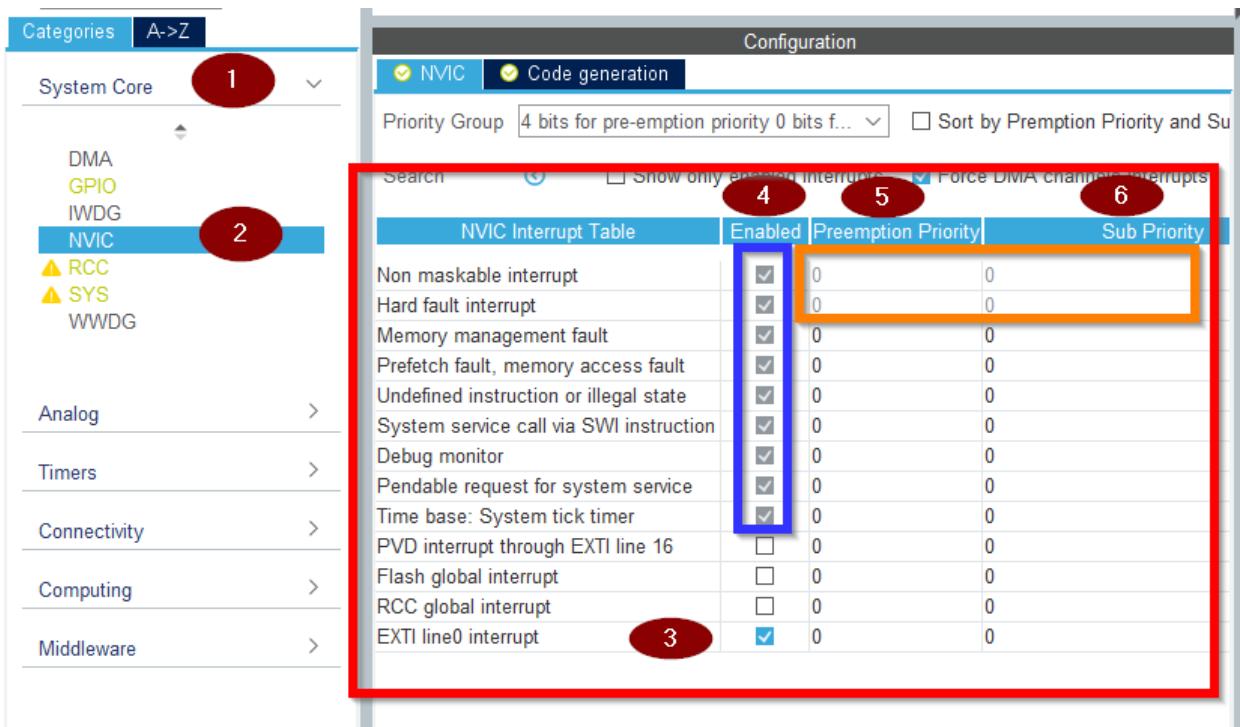
<sup>۱</sup> Nested Vector Interrupt Controller



(شکل ۲)

۳- روش دیگری نیز برای فعال سازی وقفه‌ها وجود دارد و آن مراجعه قسمت System Cores (گزینه ۱) و انتخاب NVIC (گزینه ۲) که در این صورت با تمامی وقفه‌های میکرو مواجه می‌شویم. همانطور که مشاهده می‌کنید آخرین وقفه همان وقفه‌ای می‌باشد که به روش قبلی فعال نمودیم. در قسمت ۴ مشاهده می‌شود که وقفه‌ایی وجود دارند از قبل فعال شده‌اند و امکان غیر فعال سازی آن وجود ندارد که در این میان، Non maskable interrupt (که مربوط به صحت عملکرد کلاک سیستم است) دارای بالاترین اولویت بوده و به محض احساس خطا یا از کار افتادگی کریستال مولد کلاک، فرخوانی می‌شود. مورد دیگر، تعیین اولویت نسبی وقفه‌هاست (گزینه ۵)، که میتوان آن را از ۰ تا ۱۵ انتخاب نمود که صفر یا شترین اولویت را دارد بطور مثال تعیین کننده این است که اگر دو وقفه همزمان روی دهد عملیات کدام وقفه اول انجام شود. حال سوال اینجاست که اگر بخواهیم بیش از ۱۶ وقفه را که

تقدم و تاخر آن‌ها برای ما مهم است را اولویت‌بندی کنیم چکار کنیم؟ در قسمت گزینه ۶ قسمتی با عنوان Sub Priority وجود دارد که در صورت برابری Priority به زیر اولویت‌ها رجوع می‌کند.



(شکل ۳)

۴- در نهایت پروژه را Generate می‌کنیم تا Keil ساخته شود.

۵- حال باید routin اینترپت را بنویسیم که پس از هر بار اینترپت به ما اطلاع دهد.

الف- ابتدا به طور موقتی (پس از تامین خواسته آن را پاک می‌کنیم) دستور مربوط به Rootin اینترپت را مینویسیم و سپس با زدن دکمه F12 به تعریف تابع می‌رویم (شکل ۴).

ب- تابع اینترپت را کپی می‌کنیم (شکل ۵)

پ- آن را پس از تابع Main و While(1) past، Weak را پاک می‌کنیم (شکل ۶).

نکته: تابع Weak نسخه ضعیف یک تابع می‌باشد که در صورت استفاده مجدد از آن weak را پاک می‌کنیم، که در اینصورت تابع قبلی غیر فعال می‌شود.

```
main.c* stm32f1xx_hal_gpio.c
89  /* Initialize all configured peripherals */
90  MX_GPIO_Init();
91  /* USER CODE BEGIN 2 */
92  HAL_GPIO_EXTI_Callback
93
94  /* USER CODE END 2 */
95  /* Infinite loop */
96  /* USER CODE BEGIN WHILE */
97  while (1)
98 {
```

(٤) شكل

```
main.c* stm32f1xx_hal_gpio.c
560 /*retval None
61 */
62 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
63 {
64  /* Prevent unused argument(s) compilation warning */
65  UNUSED(GPIO_Pin);
66  /* NOTE: This function Should not be modified, when the callback is needed,
67   the HAL_GPIO_EXTI_Callback could be implemented in the user file
68 */
69 }
```

(٥) شكل

```
main.c* stm32f1xx_hal_gpio.c
95  /* Infinite loop */
96  /* USER CODE BEGIN WHILE */
97  while (1)
98  {
99
100  /* USER CODE END WHILE */
101
102  /* USER CODE BEGIN 3 */
103  }
104
105 }
106 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
107 {
108
109 }
110 /* USER CODE END 3 */
111 */
112 /*
```

(٦) شكل

۶- حال در این مرحله شمارنده را داخل تابع ایترپت می‌نویسیم که با هر بار فشار دادن دکمه وارد تابع ایترپت می‌شود و یکی به MyCounter اضافه می‌کند. البته باید یک شرط برای آن در نظر بگیریم که اگر ایترپت مربوط به خط صفر بود این کار را انجام دهد.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin==GPIO_PIN_0)
    {
        MyCounter++;
    }
}
```

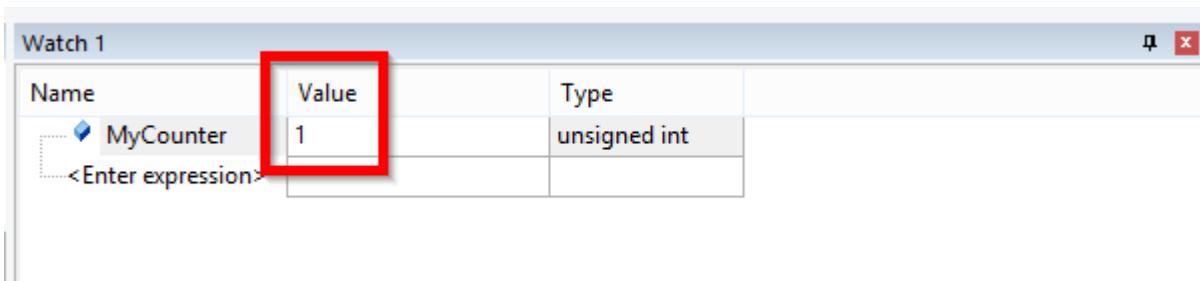
کد ۱

با توجه به اینکه Label گذاری کرده بودیم می‌توان شرط را به صورت زیر نوشت:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin==Sw1_Pin)
    {
        MyCounter++;
    }
}
```

کد ۲

با استفاده از ایترپت تمام ۴ مشکل مطرح شده در فصل گذشته حل می‌شود. و همانطور که در محیط دیباگ (شکل ۷) مشاهده می‌کنید با هر بار زدن کلید به مقدار MyCounter یک واحد اضافه می‌شود.



(شکل ۷)

فصل ٦

**TIMER**

## ۱-۶ تایمر

در بحث تایمر اولین و راحت ترین کار این است که زمان را چطور اندازه گیری کنیم؟

تنها چیزی که برای سنکرون کردن میکرو با محیط خارج داریم، کلاک است-بطور مثال فرکانس میکرو 72MHz STM32F103C8T6 است یعنی در هر ثانیه ۷۲ میلیون بار کلاک می زند پس ما میتوانیم با شمردن این لبها و با توجه به داشتن فرکانس و تبدیل آن به زمان مقدار آن را به دست آوریم، بطور مثال اگر تعداد لبهای بالارونده 36M بود می فهمیم ۰.۵Sec گذشته است.

\* ساختار تایمرهای یک میکرو یکسان می باشد ولی برخی تایمرها دارای ویژگی ها و آپشن های بیشتری می باشند، پیشرفتی ترین نوع تایمر را advanced می نامند.

\* قلب یک تایمر، شمارنده آن یا همان رجیستری که قرار است لبه های بالارونده را بشمارد (CNT) که مقادیر شمرده شده را نیز در خودش ذخیره می کند. که بیشترین مقدار آن وابسته به Bit تایмер است و چون اکثر تایمرها ۱۶ بیتی می باشند تا  $65535 = 1 - 2^{16}$  می توانند بشمارد.

\* عنصر محدود کننده CNT، ریجستر<sup>۱</sup> ARR می باشد بطور مثال اگر ARR را ۲۰۰۰ بگذاریم، CNT تا ۲۰۰۰ می توانند بشمارد و پس از آن CNT از صفر دوباره شروع به شماردن می کند.

حال چالشی که با آن روبرو می شویم این است که: (بافرض اینکه از میکرو STM32F103C8T6 استفاده می کنیم) فرکانس کاری میکرو 72MHz است یعنی ۷۲۰۰۰۰۰ کلاک می زند ۷۲ میلیون لبه دارد در حالیکه CNT نهایتاً تا ۶۵۵۳۵ را می توانند بشمارد و سرریز می شود بطور مثال اگر بخواهد یک ثانیه را بشمارد  $72000000 - 65535 = 1934464$  کلاک کم می آورد.

راه حل: ۱- سازنده های میکرو با پیش بینی این مشکل با استفاده از PSC، مقدار فرکانس کاری میکرو کنترلر F<sub>M</sub> را تقسیم می کند. سپس فرکانسی که در اختیار ما قرار می گیرد F<sub>CK-CNT</sub> می باشد.

$$F_{CK-CNT} = \frac{F_M}{PSC + 1}$$

یک دور شمارش CNT تا آخرین مقدارش که ARR می باشد چقدر طول میکشد؟ به عبارتی چه مدت زمان طول می کشد تایمر سرریز (Update Or Overflow) شود؟

<sup>1</sup> Auto Reload Register

<sup>2</sup> perscaler

$$T_{CK-CNT} = \frac{1}{F_{CK-CNT}}$$

$$T_{ovf} = ARR \times \frac{1}{F_{CK-CNT}}$$

$T_{CK-CNT}$  : مدت زمانی که هر پالس طول می کشد (دوره تناوب).

$T_{ovf}$  : مدت زمانی که طول می کشد تایمر سرریز (Update) شود.

مثال  $PSC$  و  $ARR$  طوری تنظیم شود که هر یک ثانیه سر ریز شود، یعنی  $T_{ovf}=1s$  شود:

یعنی باید  $Fck=ARR$  باشد.

$ARR=10000$  را یک عدد دلخواه بین ۳۵-۶۵۵۳۵ انتخاب می کنیم - فرضًا

مثال: حال مقدار ۰.۵s چطور بدست می آید؟

در این صورت  $PSC$  را ثابت نگه می داریم و  $ARR$  را نصف می کنیم ( $ARR=5000$ ) به عبارتی هرچه مقدار  $ARR$  باشد  $F_{CK-CNT}$  باید نصف آن باشد.

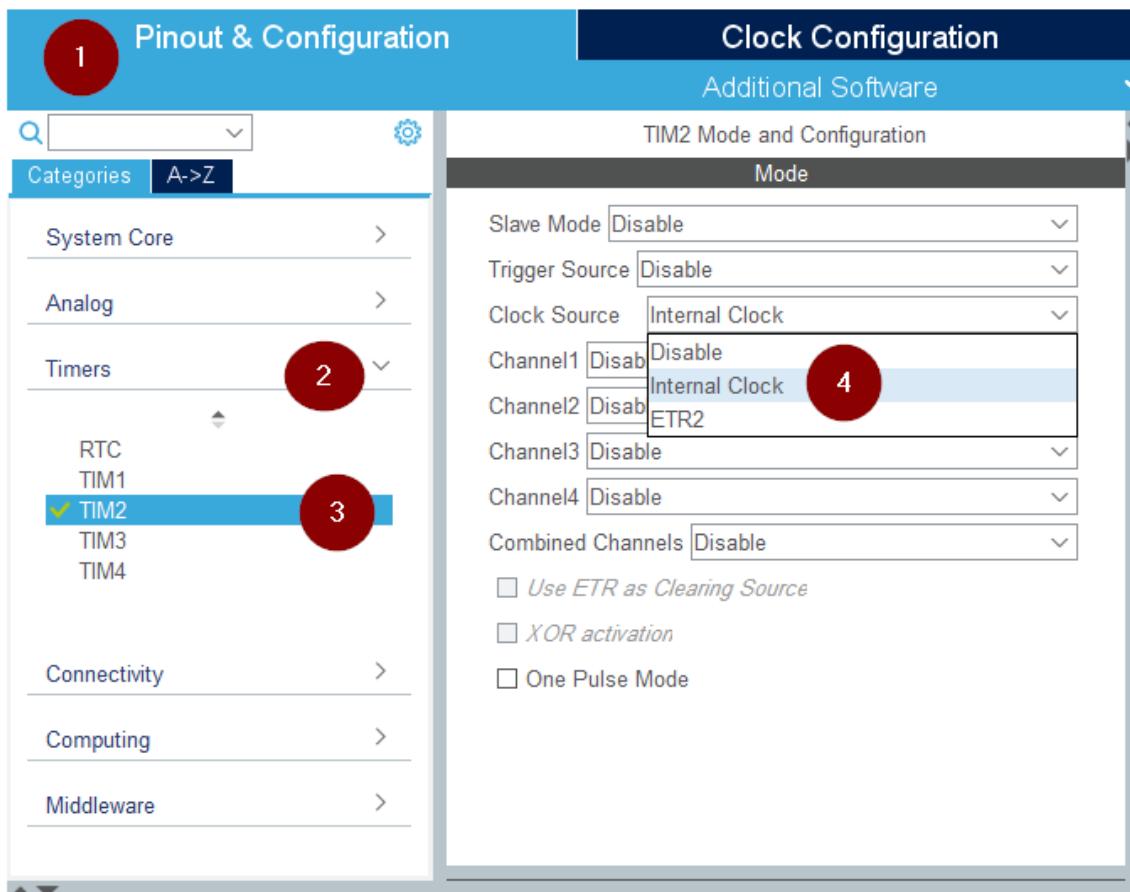
\*مدت زمانی که هر کلاک ( $CNT$ ) طول می کشد را *Tictimer* گویند. گاهی اوقات از آن به عنوان یک شمارنده ساده در مد TimeBase استفاده می کنیم.

$$Tictimer = \frac{1}{ARR}$$

می خواهیم برنامه‌ای بنویسیم که یک واحد زمانی را درست کنیم (در اصطلاح

-۱ Cube را باز کرده و تنظیمات معمول و ابتدایی کار را انجام داده، حال مطابق شکل ۱ یک تایмер را انتخاب می کنیم (با توجه به اینکه برنامه ساده‌ای می باشد یک تایмер بجز Timer1 (تایمر Advanced) را انتخاب میکنیم)، سپس منبع کلاک (گزینه ۴) را داخلی ( $F_M = 72MHz$ ) انتخاب می کنیم.

نکته: برای تنظیم کانفیگ تایمر در حالت Time Base پالس کلاک خود را از کریستال داخلی دریافت کند. همچنین اگر بخواهیم از کریستال خارجی برای این منظور استفاده کنیم، باید این منوی کشویی را بر روی گزینه ETR2 تنظیم کنیم.



-۲- با فعال کردن تایمر تب Configuration باز می شود (شکل ۲):

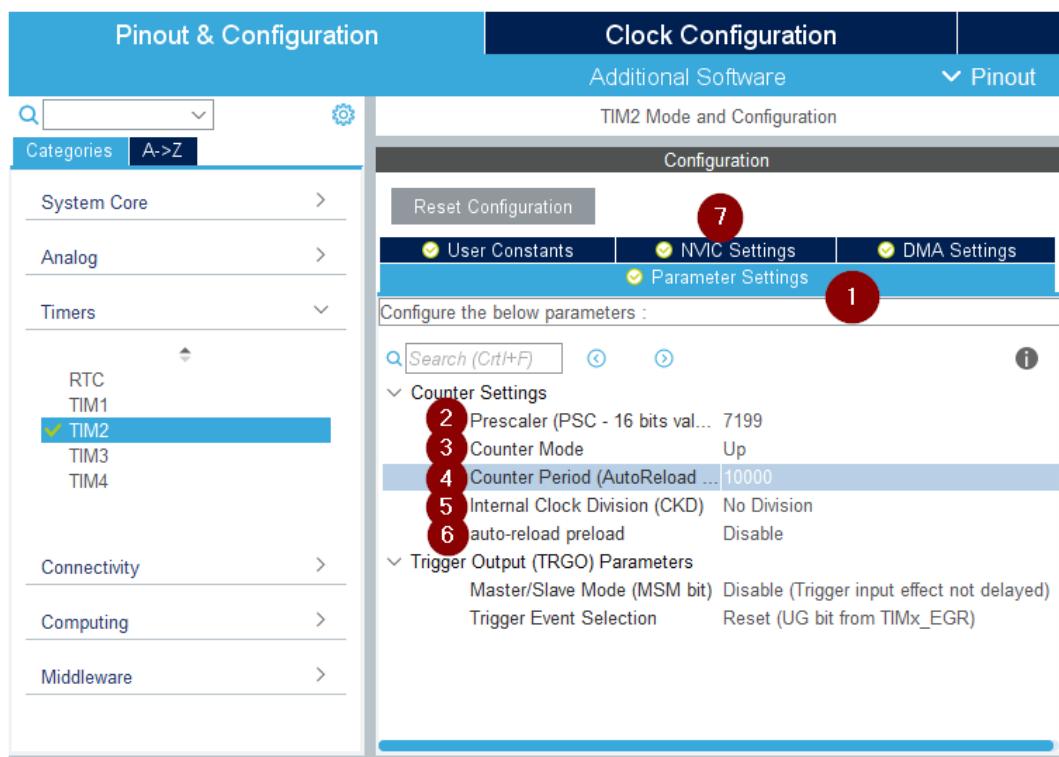
گزینه ۲ - Prescaler: یک رجیستر ۱۶ بیتی است که مقدار مقسم فرکانسی را تعیین می کند و می تواند هر عدد صحیحی بین ۰ تا ۶۵۵۳۵ باشد. دقت شود که مقداری که در این فیلد درج می کنیم باید یک واحد از مقدار مقداری که محاسبه نموده اید کمتر باشد، چرا که میکروکنترلر، خود یک واحد به آن اضافه کرده و سپس فرکانس کلاک را بر آن تقسیم می کند. پیرو مقدار آن را ۷۱۹۹ انتخاب می کنیم.

گزینه ۳ - Counter Mode: این پارامتر، نحوه (جهت) شمارش تایمر را مشخص می کند و شامل سه حالت Center Aligned و Up و Down است. واضح است که در حالت Up مقدار رجیستر CNT از صفر شروع به شمارش کرده تا به ARR برسد و ریست (صفر) شود. در حالت Down عکس این عمل روی می دهد. و نهایتاً در حالت Center Aligned از صفر شروع به شمارش کرده و پس از رسیدن به ARR، ریست نشده، بلکه شروع به شمارش معکوس تا صفر می کند. در اکثر پروژه های شامل تایمر، از حالت شمارش Up استفاده می شود. در حالت Time Base مدد آن مهم نمی باشد.

گزینه ۴- Counter Period (ARR): این پارامتر، همان ARR است که یک رجیستر ۱۶ بیتی است و می‌تواند هر عدد صحیحی بین ۰ تا ۶۵۵۳۵ را بخود بگیرد. دقت شود که مقداری که در این فیلد درج می‌شود باید یک واحد از مقدار مورد نظر کوچک‌تر باشد (زیرا شمارش در CNT از صفر شروع می‌شود)، پیرو مثال مقدار آن را ۱۰۰۰۰ انتخاب می‌کنیم (در اصل باید ۹۹۹۹ انتخاب شود ولی با توجه به بزرگ بودن مقدار آن ناچیز است و تاثیر گذار نیست در نتیجه برای فهم بهتر موضوع آن را رند می‌کنیم).

گزینه ۵- Internal Clock Division (CKD): این پارامتر برای حالتی است که از کریستال خارجی برای تأمین پالس کلک استفاده شود.

گزینه ۶- Auto-reload Preload: این پارامتر به معنای بارگذاری یا عدم بارگذاری مجدد رجیستر CNT است. در صورت Disable بودن، تایمر تنها یک بار شمارش خواهد کرد و با رسیدن به مقدار نهایی، دیگر بارگذاری مجدد نخواهد شد. در صورتیکه آنرا بر روی Enable تنظیم کنیم، شمارش و سرریز تایمر بصورت نامتناهی تکرار می‌گردد.

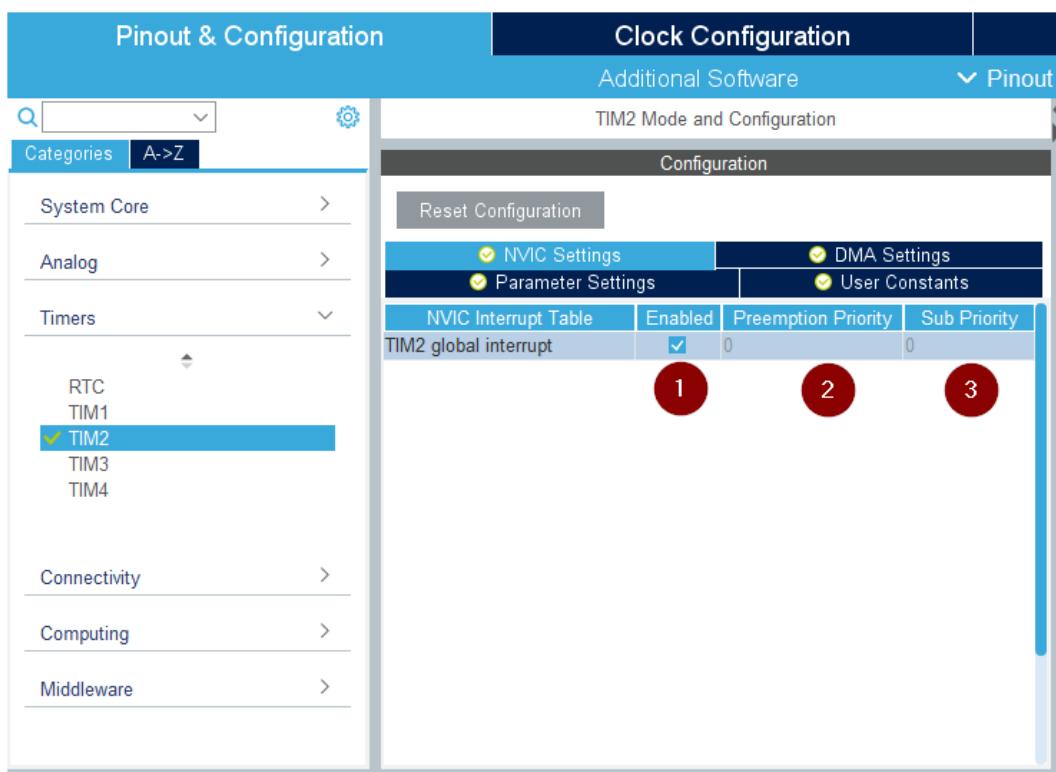


(شکل ۲):

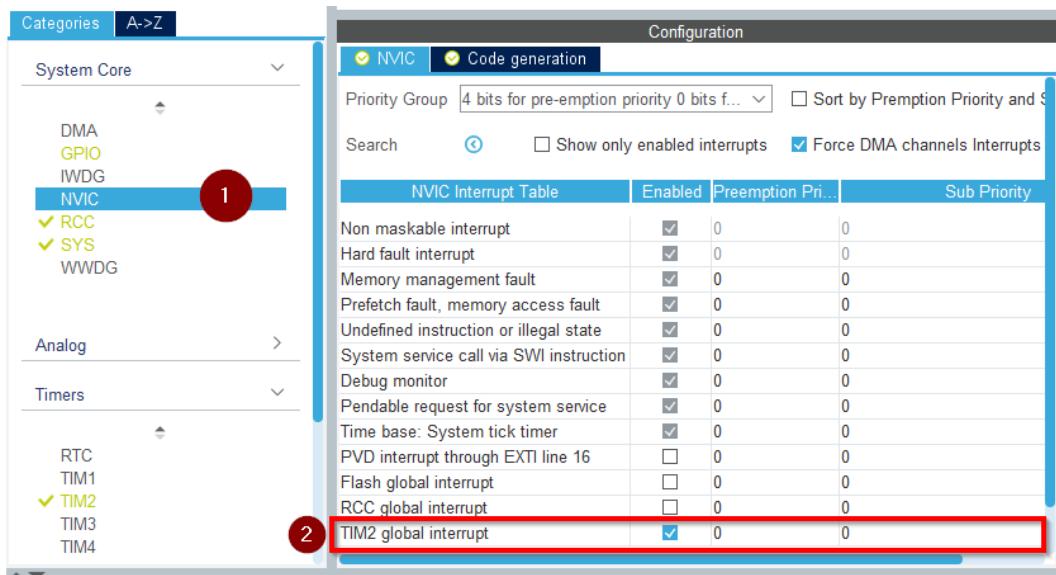
حال به تب NVIC Setting (گزینه ۷) میرویم.

۳- در این تب (شکل ۳) می‌توان هر یک از وقفه‌های قابل استفاده در تایمر را فعال نموده (گزینه ۱) و اولویت‌بندی آن را نیز تنظیم نمود (گزینه ۲ و ۳). شایان ذکر است که این اولویت‌بندی، تنها جهت مدیریت ترتیب سرویس‌دهی به وقفه‌ها در صورت وقوع همزمان وقفه‌های مربوط به تایمر بکار می‌رود. در اینجا با توجه به اینکه می‌خواهیم هر بار تایمر سرریز یا به اصطلاح Update شد به ما اطلاع دهد تاکه Enable مقابله با توجه به اینکه می‌خواهیم هر بار تایمر سرریز یا به اصطلاح Update شد به ما اطلاع دهد تاکه یک وقفه داریم اولویت‌بندی وقفه‌ها اهمیتی ندارد و همان مقدار پیش‌فرض را می‌گذاریم (گزینه ۲ و ۳).

۴- همچنین می‌توانستیم اینترپات مربوط به تایمر را از تب NVIC (گزینه ۱) موجود در System Core فعال کنیم (شکل ۴).

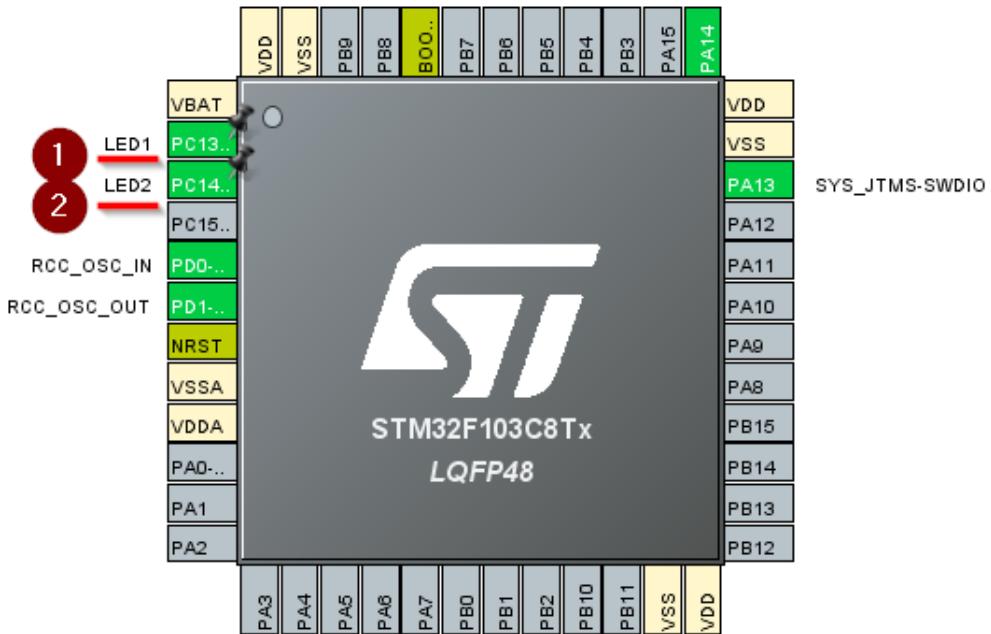


(شکل ۳)



(۴) شکل

-۵ - LED1,2 را در پایه های ۱۳ و ۱۴ پورت C فعال می کنیم(شکل ۵).



(۵) شکل

-۶ - در نهایت پروژه را Generate می کنیم.

-۷ - در گزینه ۱ شکل ۶ مشاهده می کنیم که Timer2 قبل از (1) while فراخوانی شده است. اگر بر روی آن کلیک کنیم سپس F12 را بزنیم تمامی تعاریف و پارامترهایی(شکل ۷) را که برای آن در Cube تعريف

کرده بودیم در انتهای تابع Main م شاهده می‌کنیم. در صورت نیاز می‌توانیم بدون مراجعه مجدد به پارامترهای آن را مستقیماً از همینجا تغییر داد.

```

88     /* USER CODE END SysInit */
89
90     /* Initialize all configured peripherals */
91     MX_GPIO_Init();
92     MX_TIM2_Init(); 1
93     /* USER CODE BEGIN 2 */
94
95     /* USER CODE END 2 */
96
97     /* Infinite loop */
98     /* USER CODE BEGIN WHILE */
99     while (1)
100    {
101        /* USER CODE END WHILE */
102
103        /* USER CODE BEGIN 3 */
104    }
105    /* USER CODE END 3 */

```

(شکل ۶)

```

150 static void MX_TIM2_Init(void)
151 {
152
153     /* USER CODE BEGIN TIM2_Init_0 */
154     /* USER CODE END TIM2_Init_0 */
155
156     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
157     TIM_MasterConfigTypeDef sMasterConfig = {0};
158
159     /* USER CODE BEGIN TIM2_Init_1 */
160
161     /* USER CODE END TIM2_Init_1 */
162     htim2.Instance = TIM2;
163     htim2.Init.Prescaler = 7199;
164     htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
165     htim2.Init.Period = 10000;
166     htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
167     htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
168     if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
169     {
170         Error_Handler();
171     }
172     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
173     if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
174     {
175         Error_Handler();
176     }

```

(شکل ۷)

-۸- یکبار نیاز است که تایمر را فعال کنیم به همین دلیل آن را داخل Main و قبل از (1) اجام

می‌دهیم (شکل ۸):

گزینه ۱- مد کاری تایمر را انتخاب می‌کنیم.

گزینه ۲- دستور Start تایمر فعال می‌شود.

گزینه ۳- با توجه به اینکه اینترپت تایمر را فعال کرده ایم در اینجا IT را نیز اضافه می‌کنیم.

گزینه ۴- با زدن دکمه F12 به تعریف تابع میرویم، مشاهده می‌کنیم که تابع یک ورودی دارد و آن هم آدرس و شماره تایمر می‌باشد. با توجه به اینکه ورودی \* دارد نیاز است که قبل از ورودی (Pointer) & بگذاریم تا به آدرس ورودی مورد نظر برود، از این به بعد با Handler (یک Structure) است که تمام تنظیمات یک Peripheral را در خود نگه می‌دارد) کار می‌کنیم.

```
main.c* stm32f1xx_hal_tim.c
87
88     /* USER CODE END SysInit */
89
90     /* Initialize all configured peripher
91     MX_GPIO_Init();
92     MX_TIM2_Init();
93     /* USER CODE BEGIN 2 */
94
95
96     HAL_TIM_Base_Start_IT(&htim2);
97     1     2     3     4
98
99     /* USER CODE END 2 */
100
101    /* Infinite loop */
102    /* USER CODE BEGIN WHILE */
103    while (1)
104 {
```

(شکل ۸)

```

432     * @brief Starts the TIM Base generation in interrupt mode.
433     * @param htim TIM Base handle
434     * @retval HAL status
435     */
436 HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)
437 {
438     uint32_t tmpsmcr;
439
440     /* Check the parameters */
441     assert_param(IS_TIM_INSTANCE(htim->Instance));
442
443     /* Enable the TIM Update interrupt */
444     __HAL_TIM_ENABLE_IT(htim, TIM_IT_UPDATE);

```

(۹) شکل

- ۹- حال باید routin ایترپت را بنویسیم که پس از هر بار Update به ما اطلاع دهد.
- الف- ابتدا به طور موقتی (پس از تامین خواسته آن را پاک می کنیم) دستور مربوط به Update تایмер را مینویسیم و سپس با زدن دکمه F12 به تعریف تابع میرویم (شکل ۱۰).
- ب- تابع Update را کپی کرد می کنیم (شکل ۱۱).
- پ- آن را پس از تابع Main، past کرده و کلمه Weak را پاک می کنیم (شکل ۱۲).

```

95
96     HAL_TIM_Base_Start_IT(&htim2);
97
98     HAL_TIM_PeriodElapsedCallback
99
100    /* USER CODE END 2 */
101
102    /* Infinite loop */

```

(۱۰) شکل

```

4830 */
4831 weak void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
4832 {
4833     /* Prevent unused argument(s) compilation warning */
4834     UNUSED(htim);
4835
4836     /* NOTE : This function should not be modified, when the callback
4837             the HAL_TIM_PeriodElapsedCallback could be implemented
4838     */
4839 }

```

(۱۱) شکل

```

101  /* Infinite loop */
102  /* USER CODE BEGIN WHILE */
103  while (1)
104  {
105      /* USER CODE END WHILE */
106
107      /* USER CODE BEGIN 3 */
108  }
109
110 }
111 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
112 {
113
114
115
116 }
117 /* USER CODE END 3 */
118 /* */

```

(شکل ۱۲)

۱- حال برای اینکه بفهمیم چه زمانی تایمر آپدیت و وارد اینتراپت می‌شود، از LED استفاده می‌کنیم، بطوریکه هر زمان وارد اینتراپت آپدیت شد، LED تغییر وضعیت بدهد(به عبارتی Toggle شود. (اما نکته مهم اینجاست که باید مشخص کنیم اگر اینتراپت آپدیت مربوط به تایمر ۲ بود این اتفاق بیوفند چون ممکن است چندین تایمر فعال باشد. بنابراین اگر Handeler ای که به ما ارجاع داده شد برابر با Handeler تایمر ۲ (اگر تایمر ۲ اینتراپت زد) بود، LED را Toggle کن (در اینجا هر یک ثانیه یکبار این اتفاق می‌افتد) یا به عبارتی هر خواسته دیگری را داشتیم اعمال کن.

```

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim==&htim2)
    {
        HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
    }
}

```

۱۱- برای اینکه هردو LED ، شوند می‌توان دستور را دوبار نوشت یا اینکه چون هردو LED از یک پورت می‌باشند از دستور ( | ) استفاده کرد و کد را به صورت زیر نوشت.

```
HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin | LED2_Pin);
```

۱۲- در کد نوشته شده با توجه به اینکه Level هر دو LED در Cube گرفته شد، با وارد شدن به اینتراتپ با هم خاموش و روشن می‌شوند. برای اینکه معکوس هم شوند ۲ راه وجود دارد:

الف Level - یکی از LED ها را High و دیگری را Low تنظیم می‌کنیم.

ب- در داخل Main و بعد از Start کردن تایمر یکی از LED ها را Toggle می‌کنیم.

```
HAL_GPIO_TogglePin(LED2_GPIO_Port, LED1_Pin);
```

۱۳- برای اینکه مشاهده کنیم تایمر ما می‌شمرد وارد Debug می‌شویم. بر روی TIM2 کلیک راست کرده و آن را به watch1 ، Add می‌کنیم (با باز کردن آن تمام رجیسترهای آن نشان داده می‌شود) همچنین می‌توانیم هر کدام از ریجسترها مانند CTN که شمارنده می‌باشد را جداگانه وارد می‌کنیم و تغیرات آن را مشاهده می‌کنیم. همانطور که در شکل ۱۳ می‌بینیم، مقدار ARR و PSC ثابت ولی مقدار CNT پیوسته در حال تغییر می‌باشد.

نکته: وقتی مقدار پارامترها در حال تغییر باشند در لحظه تغییر سلول آنها به رنگ سبز درمی‌آید و پارامترهایی که ثابت هستند سفید رنگ باقی می‌مانند.

Name	Value	Type
((TIM_TypeDef *...))	0x40000000	pointer
CR1	1	unsigned int
CR2	0	unsigned int
SMCR	0	unsigned int
DIER	1	unsigned int
SR	30	unsigned int
EGR	0	unsigned int
CCMR1	0	unsigned int
CCMR2	0	unsigned int
CCER	0	unsigned int
CNT	5099	unsigned int
PSC	7199	unsigned int
ARR	10000	unsigned int
RCR	0	unsigned int
CCR1	0	unsigned int
CCR2	0	unsigned int
CCR3	0	unsigned int
CCR4	0	unsigned int
BDTR	0	unsigned int
DCR	0	unsigned int
DMAR	1	unsigned int
OR	0	unsigned int
<Enter expression>		

(شکل ۱۳)

۱۴- حال برنامه را طوری تغییر دهید که در هر  $10ms$  Update شود.

۱۵- با تغییر ARR این کار را انجام می‌دهیم. با توجه به اینکه  $F_{clk\_cnt} = 10000$  می‌باشد بنابراین هر کلاک

۱۰۰ زمان می‌برد پس برای اینکه در هر  $10 ms$  شود نیاز است  $100 CNT$  بار کلک

بزند و سریز شود بنابراین باید  $ARR = 100$  شود.

$$Tic = \frac{1}{10000} = 100 \mu s$$

$$10 ms = ARR \times 100 \mu s \stackrel{so}{\Rightarrow} ARR = 100$$

۱۶- حال می خواهیم زمانی که CNT ۳۰ کلاک زد متوجه شویم و کاری را انجام دهیم، این در حالی است که هر Update ۱۰ ms تایمر می شود و به ما اینترپت می دهد.

راه حل: در این صورت دیگر از حالت اینترپت استفاده نمی کنیم و مقدار CNT را خوانده و از آن استفاده می کنیم.

۱۷- پس از اعمال تغییرات مرحله ۱۵ برنامه ای می نویسیم که تا ۳ ms LED1 SET و پس از آن (۷ ms باقیمانده) RESET شود(شکل ۱۲).

۱۸- بنا بر این در داخل حلقه While(1) از دستور if استفاده می کنیم، دستور داخل if بدین گونه می باشد که می گوییم اگر CNT کوچکتر از  $\frac{3ms}{100us} = 30$  (بود) SET ، LED1 و در غیر این صورت RESET شود(شکل ۱۲).

۱۹- به دو روش می توان دستور داخل if را می توان نوشت:  
نکته: چون یک peripheral Structure می باشد: برای دسترسی به عضوهای آن از عملگر میخ ( $->$ ) استفاده می کنیم.

الف- ریجستری: اسم و شماره peripheral (TIM2) را نوشه سپس با عملگر میخ ( $->$ ) به عضوهای آن (در اینجا CNT) دسترسی پیدا می کنیم و می خواهیم مقدار آن کمتر از ۳۰ باشد.

```
while (1)
{
    if (TIM2->CNT < 30)
    {

        HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET) ;
    }
    else
    {

        HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET) ;
    }
}
```

ب- معادل: Hall در این روش معادل Counter یک تایمر را با استفاده از دستور Hall دریافت کنیم. برای افرایش خوانایی برنامه این روش پیشنهاد می شود.

```
if (__HAL_TIM_GET_COUNTER(&htim2) < 30)
```

نکته: به علت اینکه سرعت روشن و خاموش شدن خیلی زیاد است متوجه نمی‌شویم و آن را به صورت پیوسته در ک می‌کنیم، به عبارتی  $30^{\circ}$  درصد شدت روشنایی LED مشاهده می‌شود.

$$\frac{30}{30 + 70} \times 100 = \%30$$

۲۰- اینکه ما پیوسته مقدار CNT را در حلقه While(1) چک می‌کنیم روش مناسبی نمی‌باشد چون میکرو را درگیر یک کار کم اهمیت می‌کنیم، باید روشی پیدا کنیم که تنها زمانی که CNT به مقدار مدنظر ما رسید، صفر یا یک باشد به عبارتی طوری تنظیم شود که درصدی از ARR یک و درصدی نیز صفر شود. که آن روش PWM یا مدولاسیون عرض پالس می‌باشد که در فصل بعدی به آن می‌پردازیم.

## ٧ فصل

مدوّلاسيون عرض پالس

PWM

(Pulse Width Modulation)

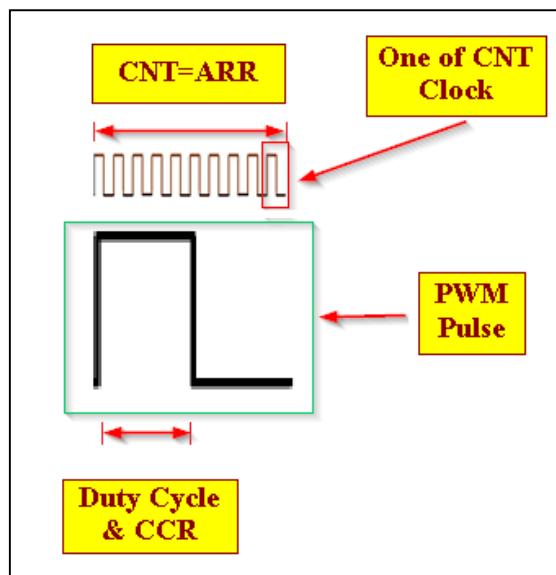
## ۱-۷ مدولاسیون عرض پالس

مد کاری دیگر تایمر، تولید سیگنال PWM است. PWM به معنای مدولاسیون عرض پالس می‌باشد به طوری که مشخص می‌کند عرض پالس یا Duty Cycle پالس تولیدی چقدر باشد و مقدار آن عرض پالس با تعیین می‌شود CCR<sup>۲</sup> عددی است کوچکتر، مساوی CNT (شکل ۱۳).

$$0 \leq CCR \leq CNT$$

نکته: در مد کاری PWM مقدار CNT برابر ARR می‌باشد (شکل ۱۳).

$$CNT = ARR$$



(شکل ۱۳)

$$T_{PWM} = \frac{1}{F_{PWM}}$$

$$F_{CK-CNT} = \frac{F_M}{PSC + 1}$$

$$T_{CK-CNT} = \frac{1}{F_{CK-CNT}}$$

$$CNT \times T_{CK-CNT} = T_{PWM}$$

<sup>۱</sup> Pulse Width Modulation

<sup>۲</sup> Capture Compare Register

$$ARR = CNT$$

نکات و روش استفاده از روابط:

رابطه ۱- مقدار فرکانسی می باشد که می خواهیم در خروجی بر روی یک پایه تولید کنیم و با دانستن آن مقدار  $T_{PWM}$  بدست می آید.

رابطه ۲- مقدار دلخواهی است و با توجه به آن مقدار  $F_{CK-CNT}$  بدست می آید.

نکته: هرچه مقدار  $PSC$  کمتر باشد دقیق بازه تغییرات عرض پالس یا همان  $CCR$  بیشتر می شود اما تا جایی می توان آن را کم کرد که مقدار بدست آمده  $CNT$  یا  $ARR$  از محدوده 65535 بیشتر نشود.

رابطه ۳- دوره تناوب  $T_{CK-CNT}$  می باشد که از معکوس فرکانس  $F_{CK-CNT}$  بدست می آید.

رابطه ۴- در نهایت، با دانستن دوره تناوب  $T_{CK-CNT}$  و  $T_{PWM}$  در می یابیم چه تعداد کلاک  $CNT$  نیاز است.

رابطه ۵- می دانیم در مد کاری  $PWM$  مقدار  $CNT$  برابر  $ARR$  می باشد.

راه حل کوتاه برای بدست آوردن  $ARR$ :

ابتدا انتخاب دلخواه  $PSC$  (با توجه به نکته گفته شده) سپس تعیین فرکانس  $PWM$  موردنیاز و در نهایت بدست آوردن مقدار  $ARR$ .

$$F_{CK-CNT} = \frac{F_M}{PSC + 1}$$

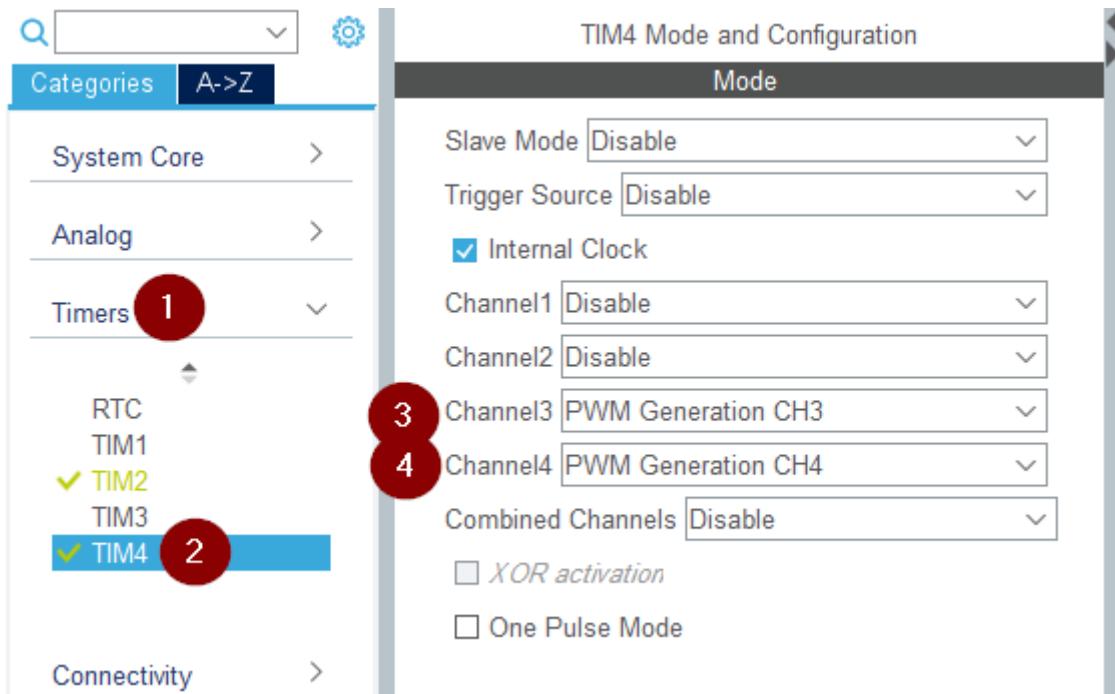
$$ARR = CNT = \frac{F_{CK-CNT}}{F_{PWM}}$$

در غالب میکروکنترلهای ST، هر تایمر دارای ۴ کانال تولید سیگنال PWM است

یکی از ویژگی های مناسب  $PWM$  در  $ARM$  این است که می توان  $Not$  همان سیگنال را در پایه ای دیگر دریافت نمود (یعنی زمانی که سطح منطقی یک سیگنال یک باشد دیگری صفر می باشد و بلعکس) که در مدارهای  $H$ -Bridge که معمولا در ساخت درایوهای موتوری و سایر کاربردها استفاده می شود یک امتیاز مهم محسوب می شود.

حال یک سیگنال  $PWM$  با فرکانس ۱۰۰ Hz طراحی می کنیم، بهمین منظور  $Cube$  و تنظیمات را انجام می دهیم:

-۱ را انتخاب کرده و تیک *Internal Clock* را می‌زنیم و سپس کانال های ۳ و ۴ را در حالت *PWM Generation* قرار می‌دهیم (شکل ۱۴).



(شکل ۱۴)

پس از انتخاب کانال‌ها تب *Configuration* باز می‌شود (شکل ۱۵):

گرینه ۶- با توجه به اینکه می‌خواهیم فرکانس 100Hz را اعمال کنیم. طبق روابط بدست آمده داریم:

$$F_{CK-CNT} = \frac{F_M}{PSC + 1} = \frac{72M}{7199 + 1} = 10000$$

$$ARR = CNT = \frac{F_{CK-CNT}}{F_{PWM}} = \frac{10000}{100} = 100$$

گرینه ۵- این پارامتر، نحوه (جهت) شمارش تایمر را مشخص می‌کند و شامل سه حالت Up، Down و Center Aligned است. همانطور که گفته شد در حالت Up مقدار رجیستر CNT از صفر شروع به شمارش کرده تا به ARR برسد و ریست (صفر) شود. در حالت Down عکس این عمل روی می‌دهد. و نهایتاً در حالت Center Aligned از صفر شروع به شمارش کرده و پس از رسیدن به ARR، ریست نشده، بلکه شروع به شمارش معکوس تا صفر می‌کند. در اینجا همان UP را انتخاب می‌کنیم.

گرینه ۷- این پارامتر برای حالتی است که از کریستال خارجی برای تأمین پالس کلاک استفاده شود.

گزینه ۸- پارامتر Auto-reload Preload به معنای بارگذاری یا عدم بارگذاری مجدد رجیستر CNT است. در صورت Disable بودن، تایمر تنها یک بار شمارش خواهد کرد و با رسیدن به مقدار نهایی، دیگر بارگذاری مجدد نخواهد شد. در صورتیکه آنرا بر روی Enable تنظیم کنیم، شمارش و سرریز تایمر بصورت نامتناهی تکرار می‌گردد.

گزینه ۹- با انتخاب ۱ PWM mode تا زمانیکه مقدار CNT به CCR نرسیده است، خروجی در حالت ۱ منطقی است و پس از آن برابر با ۰ خواهد شد. در حالیکه در حالت ۲ PWM mode عکس این موضوع روی می‌دهد.

نکته: در ۱ PWM mode هرچه مقدار CCR بیشتر شود عرض پالس (Duty Cycle) یا به عبارتی مدت زمان ۱ بودن آن بیشتر می‌شود ولی در ۱ PWM mode عکس این موضوع روی می‌دهد.

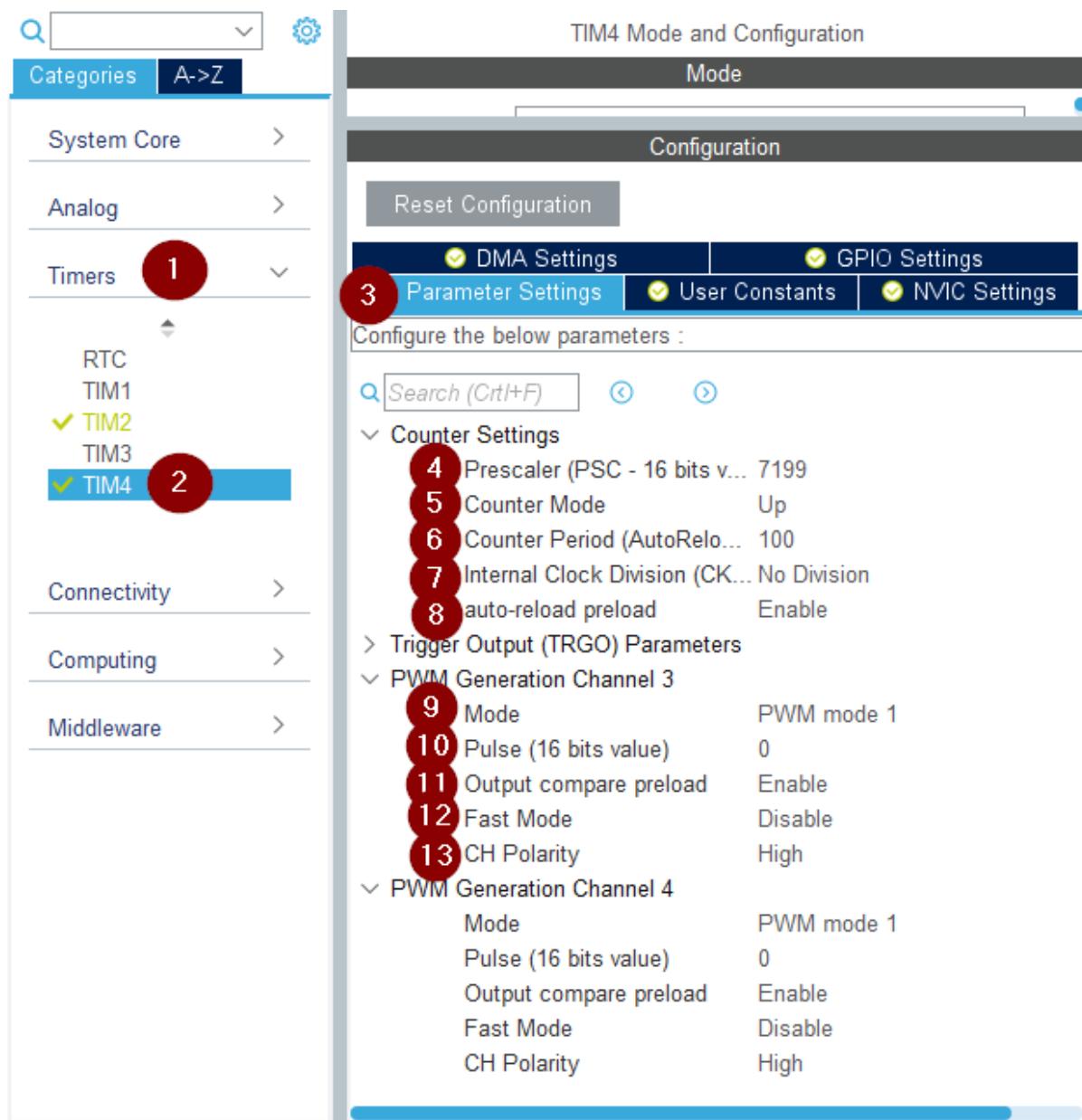
گزینه ۱۰- این پارامتر ۱۶ بیتی مقدار CCR را مشخص می‌کند و اصولاً مقدار اولیه آن را ۰ تنظیم می‌کنیم تا عرض پالس در ابتدا صفر بوده و پس از مقدارهای در برنامه، تنظیم شود.

نکته: توجه شود که مقداردهی به CCR باید با توجه به ARR انجام شود (یعنی بزرگ‌تر از آن مقداردهی نشود).

## گزینه ۱۱-

گزینه ۱۲- در صورت فعال کردن این گزینه، درست در لحظه برابر شدن مقادیر CCR با CNT خروجی تغییر سطح می‌دهد. اما در صورت غیرفعال بودن، چند سیکل کاری (کلاک) بعد از برابری این دو رجیستر، خروجی تغییر سطح می‌دهد.

گزینه ۱۳- در صورت استفاده از PWM-NOT مورد استفاده قرار می‌گیرد.



(شکل ۱۵)

۲- یکبار نیاز است که تایمر را فعال کنیم به همین دلیل آن را داخل Main و قبل از (1) اجام

می‌دهیم (شکل ۸):

الف- مد کاری تایمر (PWM) را انتخاب می‌کنیم.

ب- با دستور Start تایمر فعال می‌شود.

پ- با زدن دکمه F12 به تعریف تابع میرویم، مشاهده می‌کنیم که تابع دو ورودی دارد که اولی تایمر(با توجه به اینکه ورودی \* دارد نیاز است که قبل از ورودی (Pointer) & بگذاریم تا به آدرس ورودی

موردنظر برود) و دومی شماره کانال تایمر می‌باشد. در اینجا باید کانال ۳ و ۴ تنظیم گردد.

```
HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim,  
uint32_t Channel)
```

```
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);  
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);
```

۳- پس از فعالسازی باید عرض پالس (CCR) مورد نیاز را اعمال کنیم که به دو صورت انجام می‌شود و همچنین میدانیم باید

( $0 \leq CCR \leq CNT$ ) و در اینجا با توجه به اینکه مقدار CNT برابر ۱۰۰ می‌باشد داریم:

الف- رجیستری: اسم و شماره peripheral (TIM4) را نوشته سپس با عملگر میخ ( $\leftarrow\rightleftharpoons$ ) به عضوهای آن (در اینجا CCR) دسترسی پیدا می‌کنیم.

```
TIM4->CCR3=10;  
TIM4->CCR4=100;
```

ب- معادل: Hall در این روش مقدار CCR را از طریق COMPARE با استفاده از دستور Hall دریافت کنیم که برای افزایش خوانایی برنامه این روش پیشنهاد می‌شود. همچنین با زدن F12 می‌توان از ورودی‌های آن که شامل نام و شماره تایмер، کanal فعال شده و مقدار عرض پالس می‌باشد مطلع شد.

```
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, 10);  
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 100);
```

```
#define __HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__,  
__COMPARE__)\
```

حال می‌خواهیم برنامه‌ای بنویسیم که هر 10ms LED1 افزایش و LED2 کاهش یابد. ابتدا یک متغیر در اول برنامه، قبل از Main تعریف می‌کنیم (یا داخل پرانتز حلقه for).

```
uint16_t i;
```

در داخل حلقه while(1) یک حلقه for می‌نویسیم و مقدار آن را از ۰ تا ARR در هر مرحله یکی یکی افزایش می‌دهیم. در داخل حلقه for نیز مقدار CCR (COMPARE) را برای یک کanal بصورت افزایشی (i) و کanal دیگر کاهشی اعمال می‌کنیم و برای ۱۰ms یک Delay اعمال می‌کنیم.

```
while (1)  
{
```

```

for (int i=0 ; i<=100 ; i++)
{
    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, i);

    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 100-i);
    HAL_Delay(10);
}
}

```

حال می خواهیم روشی استفاده کنیم که دستور Delay را حذف کنیم (بطور کلی تمام تلاش ما اینست که تا جایی که امکان دارد از دستور Delay کمتر استفاده کنیم تا بیهوده برنامه را در جایی متوقف نکنیم). که دو روش وجود دارد:

### الف\_ استفاده از تایمر دیگر:

۱- با استفاده از روش های قبلی یک تایمر را در حالت Time Base فعال نموده که در هر ۱۰ms (سرریز) شود و به ما اینترپت بدهد. بدین منظور آن را با فرض  $PSC=71$  بست می آوریم:

$$F_{CK-CNT} = \frac{F_M}{PSC + 1} = \frac{72M}{71 + 1} = 1M$$

$$T_{ovf} = ARR \times \frac{1}{F_{CK-CNT}} \Rightarrow 10ms = ARR \times 1us \Rightarrow ARR = 10000$$

۲- پس از یافتن تابع اینترپت آپدیت مطابق روش گفته شده در مباحث پیشین، دستور مدنظر را داخل آن می نویسیم.

```

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim==&htim2)
    {
        for(int i=0 ; i<=100 ; i++)
        {

            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, i);

            __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 100-i);
        }
    }
}

```

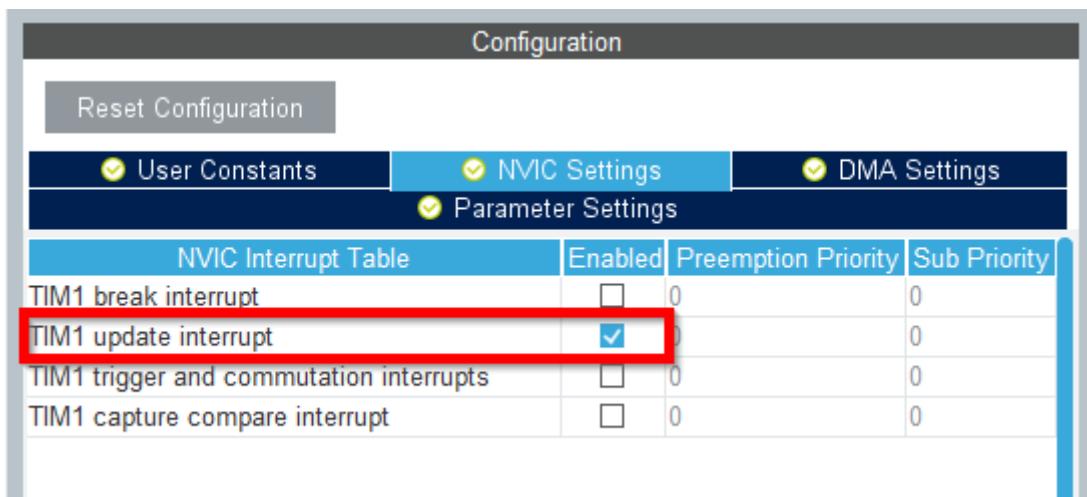
```

        }
    }
}

```

الف\_ استفاده از همان تایمر که وظیفه تولید موج PWM را دارد:

۱- بدین منظور تنها کافیست به تب NVIC برویم و اینترافت آپدیت آن را فعال کنیم(شکل ۱۶).



شکل ۱۶

۲- تابع اینترافت نیز همان تابع اینترافت TimeBase می باشد.

```

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDefDef
*htim)
{
    if(htim==&htim4)
    {
        for(int i=0 ; i<=100 ; i++)
        {

            HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, i);

            HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 100-i);
        }
    }
}

```

یکی از ویژگی های مهم تایمر در مد کاری PWM تولید موج PWM در یک کانال و تولید NOT همان موج PWM در همان کانال و در دو پین مختلف می باشد، که استفاده های زیادی در صنعت از جمله ساخت درایو موتورهای القایی سنکرون و آسنکرون، سوییچ رلوکتانسی و همچین مدارهای مبدل ولتاژ دارد.

نحوه تنظیمات Cube مانند قبل است، با چند تنظیم اضافه:

۱- اولاً همه‌ی تایمرا این قابلیت را ندارند(مثلا در این میکرو فقط تایمر ۱ این قابلیت را دارد) بنابراین با باز PWM Generation CH2 CHN2 ، channel2 و PWM Generation CH1 CHN1 ، channel1 نمودن

را انتخاب می‌کنیم.

۲- دوماً نکته خیلی مهم که اهمیت این حالت میکرو را دوچندان می‌کند قابلیت اعمال Dead Time برای PWM و PWM-N است. که معمولاً به صورت مداری این کار صورت می‌گرفته یا با تنظیمات پیچیده که در زهایت هم آن مقدار دقیق اعمال نمی‌شود(مانند میکروهای ATMega ) ولی در این میکروکنترلر با کیفیت و دقت مناسب پیاده‌سازی می‌شود.

معمولًا در مداراتی که از این مد کاری استفاده می‌شود مانند H-brige که در شکل ۱۷ برای کنترل دور موتور DC نشان داده شده است، همانطور که مشاهده می‌کنید ماسفت‌ها دو به دو به صورت ضربدری روشن و خاموش می‌شوند بطوریکه فرمان PWM و PWM-N از یک کانال به ترتیب به ترانزیستورهای S1 و S2 و فرمان PWM-N و PWM به ترتیب به ترانزیستورهای S3 و S4 از کانال دیگر اعمال می‌شوند. در یک لحظه فرمان S1 و S4 صفر(به پایین روند)، S2 و S3 یک (به بالاروند) می‌شود، که این اتفاق باعث می‌شود در دفعات پیاپی به موتور شوک وارد شود بنابراین برای رفع این مشکل ابتدا باید S1 و S4 خاموش شوند و سپس بعد از مقداری تاخیر S2 و S3 روشن شوند (در زمان تاخیر هر چهار ترانزیستور خاموش می‌باشند). که به آن مقدار تاخیر Dead Time گویند(شکل ۱۸).

نکته: در اینجا ترانزیستورها نقش کلید را دارند به طوریکه با فرمان یک روشن و با فرمان صفر خاموش می‌شوند.

میزان این تاخیر که Dead Time نامیده می‌شود به صورت زیر محاسبه می‌شود:

$$DT = DTG[7:0] \times t_{DTS} \quad \text{With } (t_{dtg} = t_{DTS})$$

$$DT = (64 + DTG[5:0]) \times t_{dtg} \quad \text{With } (t_{dtg} = 2 \times t_{DTS})$$

$$DT = (32 + DTG[4:0]) \times t_{dtg} \quad \text{With } (t_{dtg} = 8 \times t_{DTS})$$

$$DT = (32 + DTG[4:0]) \times t_{dtg} \quad \text{With } (t_{dtg} = 16 \times t_{DTS})$$

مقدار  $t_{DTS}$  همواره ثابت و برابر  $\frac{1}{2} \mu s$  می باشد.

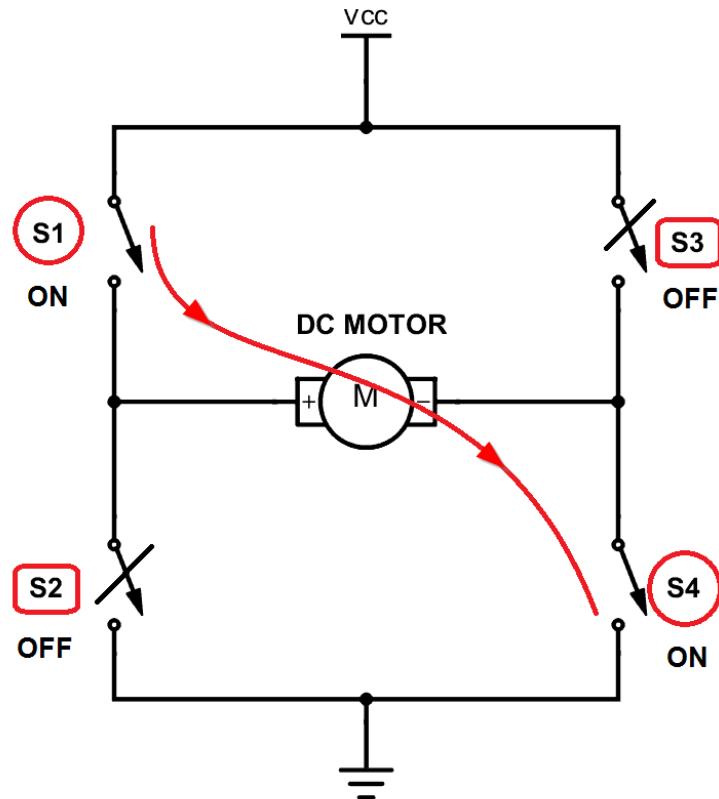
مقدار عددی است که در DTG[A:0] (Dead Time Generation) مورد نیاز Cube در قرار می دهیم. مقدار صفر تا A نیز تعداد بیتی است که مجاز به انتخاب DeadTime هستیم. بطور مثال اگر A=7 باشد عددی که بدست می آید باید کوچکتر مساوی  $2^{7+1} = 256$  باشد در غیر این صورت به فرمول بعدی می رویم.

مقدار DT (Dead Time) بر حسب زمانی است که می خواهیم بر موج PWM اعمال کنیم. معمولاً مقدار DT برابر  $2\mu s$  می باشد. ولی برای آشنایی بیشتر با فرمول های فوق فرض می کنیم می خواهیم برابر  $200\mu s$  باشد. داریم:

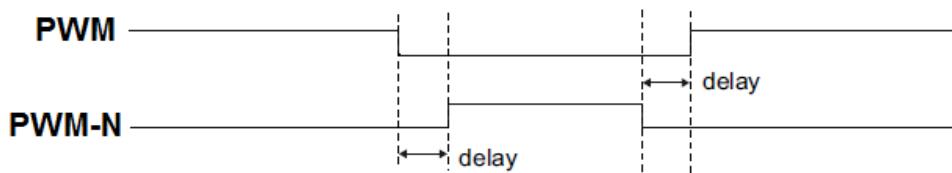
$$200\mu s = DTG[7:0] \times \frac{1}{2} \mu s \Rightarrow DTG[7:0] = 400 \quad NO! \quad With \quad (t_{dtg} = t_{DTS})$$

$$200\mu s = (64 + DTG[5:0]) \times 1\mu s \Rightarrow DTG[5:0] = 136 \quad NO! \quad With \quad (t_{dtg} = 2 \times t_{DTS})$$

$$DT = (32 + DTG[4:0]) \times t_{dtg} \Rightarrow DTG[4:0] = 18 \quad YES \checkmark \quad With \quad (t_{dtg} = 8 \times t_{DTS})$$



شکل ۱۷



شکل ۱۸

- ۳- حال پروژه را Generate می کنیم تا Keil باز شود.
- ۴- ابتدا تایмер را در حالت تایمری و مد PWM و PWM-N شروع می کنیم و سپس عرض پالس آن (CCR) را با دستور COMPARE مشخص می کنیم (مثلا ۵۰ درصد).

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);
```

```

HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_2);

__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 5000);
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 5000);

```

## ۲-۷ طراحی موج سینوسی

حال می خواهیم پروژه ای طراحی کنیم که خروجی میکرو موج سینوسی تولید کند:  
با فرض اینکه می خواهیم موج سینوسی تولید شده مانند برق شهر فرکانس ۵۰Hz و پریود ۲۰ms باشد پروژه را آغاز می کنیم:

- همانطور که در شکل ۱۹ می بینید یک موج سینوسی با پریود Tsin داریم که باید برای ساخت موج سینوسی از کلاک های دیجیتال با عرض پالس های متغیر استفاده می کنیم. که تعداد این کلاک ها با توجه به نیاز و بصورت دلخواه انتخاب می شوند.
- حال اگر تعداد کلاک را ۳۰۰ انتخاب کنیم پریود هر کلاک برابر است با :

$$T_{CLOCK} = \frac{20ms}{300} = \frac{1}{15}ms \cong 66us$$

- سپس موج PWM با توجه به پریود بدست آمده طراحی می کنیم:  
نکته: کمترین مقدار را برای PSC انتخاب می کنیم تا دامنه تغییرات ARR زیاد و در نتیجه دقت بالاتر شود.

$$F_{CK-CNT} = \frac{F_M}{PSC + 1} = \frac{72M}{0 + 1} = 72M$$

$$T_{ovf} = ARR \times \frac{1}{F_{CK-CNT}} \Rightarrow \frac{1}{15}ms = ARR \times \frac{1}{72}us \Rightarrow ARR = 4800$$

- موج PWM باید به صورت سینوسی تغییر کند بطوریکه:

بررسی پارامترهای داخل کمان سینوس:

$$\sin(2\pi ft + phi)$$

: معادل فرکانس برق شهر ۵۰ می باشد.

: برای شیفت فاز یا به عبارتی ایجاد اختلاف فاز استفاده می شود که آن را صفر در نظر می گیریم.

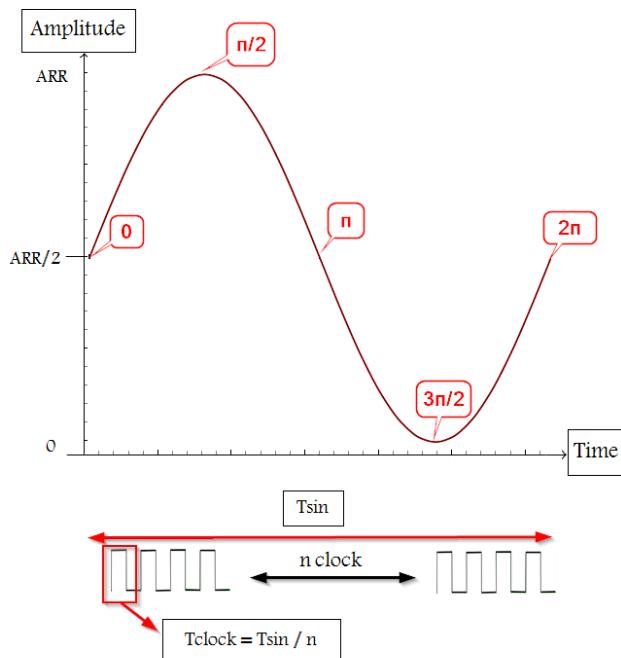
$t$  با توجه به این که پریود هر کلک  $\frac{1}{15} ms$  طول می کشد در هر مرحله  $20ms$  از زمان طی شده است. بطور مثال پس از  $300$  مرحله  $20ms$  زمان طی شده است، که یک پریود کامل موج سینوسی محسوب می شود.

نکته: فارغ از اینکه مقدار داخل کمان سینوس چه باشد در نهایت مقدار سینوس عددی بین صفر و یک است ولی CCR باید عددی بین  $0$  تا  $ARR$  باشد بنابراین سینوس باید ضریبی از  $ARR$  داشته باشد.

حال مطابق نمودار شکل ۱۹ رابطه CCR را بدست می آوریم:

$$CCR = \frac{ARR}{2} \times \sin(2\pi ft + phi) + \frac{ARR}{2}$$

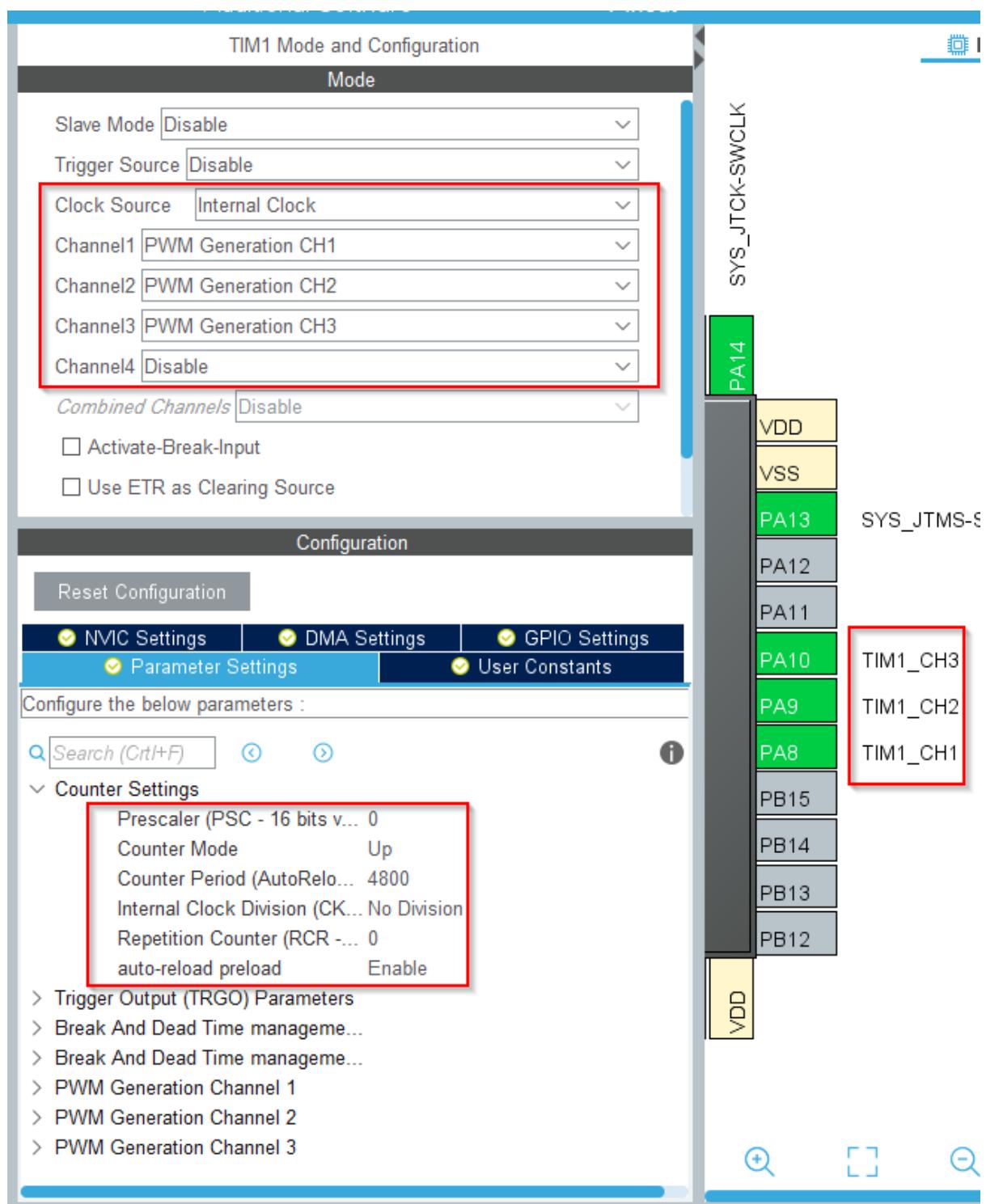
$$CCR = 2400 \times \sin(2 \times \pi \times 50 \times t + phi) + 2400$$



شکل ۱۹

-۵ Cube را باز کرده تنظیمات اولیه را انجام می دهیم.

۶- کانال‌های ۱ و ۲ و ۳ از تایمیر یک را در حالت PWM قرار میدهیم و پارامترهای بدست آمده در بندهای قبلی را در آن اعمال می‌کنیم ( $PSC = 0$ ,  $ARR = 4800$ ) (شکل ۲۰)، در اینجا به یک اینترپت آپدیت یا سرریز نیاز داریم، بطوریکه که هر  $\frac{1}{15} ms$  به ما اطلاع دهد که مقدار CCR را تغییر دهیم بنابراین اینترپت آپدیت را نیز فعال می‌کنیم (شکل ۲۱).



(شکل ۲۰)

NVIC Interrupt Table	Enabl...	Preemption Prio...	Sub Prior.
TIM1 break interrupt	<input type="checkbox"/>	0	0
TIM1 update interrupt	<input checked="" type="checkbox"/>	0	0
TIM1 trigger and commutation inter...	<input type="checkbox"/>	0	0
TIM1 capture compare interrupt	<input type="checkbox"/>	0	0

(شکل ۲۱)

۷- حال برنامه را Generate می کنیم تا Keil باز گردد.

نکته: برخی از پارامترها در کتابخانه HAL تعریف شده‌اند که تابع نیستند و تنها define شده‌اند، که به آنها Macro می‌گوییم. برای فعال سازی آن‌ها نیز باید از دو Under Line در ابتدای فعال سازی آنها استفاده نمود. در اینجا اینترپت آپدیت مربوط به PWM بدین گونه است، بنابراین برای اطلاع از تعداد و نوع ورودی‌های این درایور اینترپت F12 را نوشته و HAL\_TIM\_ENABLE\_IT را می‌زنیم تا به داخل تعریف آن که در کتابخانه وجود دارد برویم، همانطور که مشاهده می‌کنید ۲ ورودی دارد، ورودی اول Hall Handler آن و دومی نوع اینترپت می‌باشد.

```
#define __HAL_TIM_ENABLE_IT( __HANDLE , __INTERRUPT )  
((__HANDLE__)->Instance->DIER |= (__INTERRUPT__))
```

```
__HAL_TIM_ENABLE_IT(&htim1, TIM_IT_UPDATE);
```

نکته: با توجه به اینکه می‌خواهیم از توابع ریاضی مانند سینوس استفاده کنیم. باید تابع آن را به برنامه اضافه کنیم

(قبل از تابع Main).

```
#include "math.h"
```

۸- ابتدا مدهای کاری که میخواهیم با آنها کار کنیم را در داخل Main استارت میزنیم.

```
    HAL_TIM_ENABLE_IT(&htim1, TIM_IT_UPDATE);  
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);  
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);  
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
```

۹- حال تابع آپدیت اینترپت را مطابق گذشته پیدا نموده و پس از تابع Main، میکنیم و کد مربوطه را با توجه به موارد توضیح داده شده داخل آن مینویسیم، همچنین متغیرهای مورد استفاده را در ابتدای برنامه تعریف میکنیم.

```
uint16_t MyCounter;  
float t;
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)  
{  
    if(htim==&htim1)  
    {  
        uint16_t PulseWidth;  
  
        MyCounter++;  
        if(MyCounter==301)  
        {  
            MyCounter=0;  
        }  
  
        t= (float)MyCounter * (1/15000.0);  
  
        PulseWidth=2400*sin(2*pi*50*t)+2400;  
  
        HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, PulseWidth);  
    }  
}
```

حال اگر بخواهیم فرمان یک موتور القایی سه فاز را بنویسیم، سه کanal از تایمر ۱ را در حالت PWM با همان تنظیمات قبلی انتخاب نموده و پس از Generate کردن پروژه از Cube، در Keil نیز آنها را فعال میکنیم (کد). با توجه به اینکه در موتور القایی هر فاز با فاز بعدی ۱۲۰ درجه اختلاف فاز دارد، داخل کمان سینوس از شیفت فاز استفاده میکنیم (کد).

```

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim==&htim1)
    {
        uint16_t PulseWidth1 , PulseWidth2 ,
PulseWidth3;

        MyCounter++;
        if (MyCounter==301)
        {
            MyCounter=0;
        }

        t= (float)MyCounter * (1/15000.0);

        PulseWidth1=2400*sin( 2*pi*50*t - 2*pi/3 )+2400;
        PulseWidth2=2400*sin( 2*pi*50*t)+2400;
        PulseWidth3=2400*sin( 2*pi*50*t + 2*pi/3 )+2400;

        HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1,PulseWidth1);

        HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_2,PulseWidth2);

        HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_3,PulseWidth3);

    }
}

```

البته در ساخت درایو و اینورترها باید پارامترهای متعددی از جمله حلقه‌های کنترلی، ساختار مداری در قسمتهای فرمان و قدرت و همچنین پارامترهای حفاظتی در قسمت‌های مختلف مدار باید مورد مطالعه و بررسی قرار گیرند تا به یک محصول قابل اطمینان و صنعتی جهت عرضه به بازار صورت بگیرد.

### ۳-۷ طراحی فاصله سنج

می‌خواهیم با استفاده از ماژول SRF04 یک فاصله سنج طراحی کنیم.

ماژول SRF04 یک فاصله سنج صوتی می‌باشد که با ارسال سیگنال صوتی و بازگشت آن در اثر برخورد به مانع قادر به اندازه گیری فاصله می‌باشد. همانطور که در شکل ۱ می‌بینید این ماژول دارای ۴ پایه به شرح ذیل می‌باشد:

VCC- تغذیه ماژول می‌باشد.

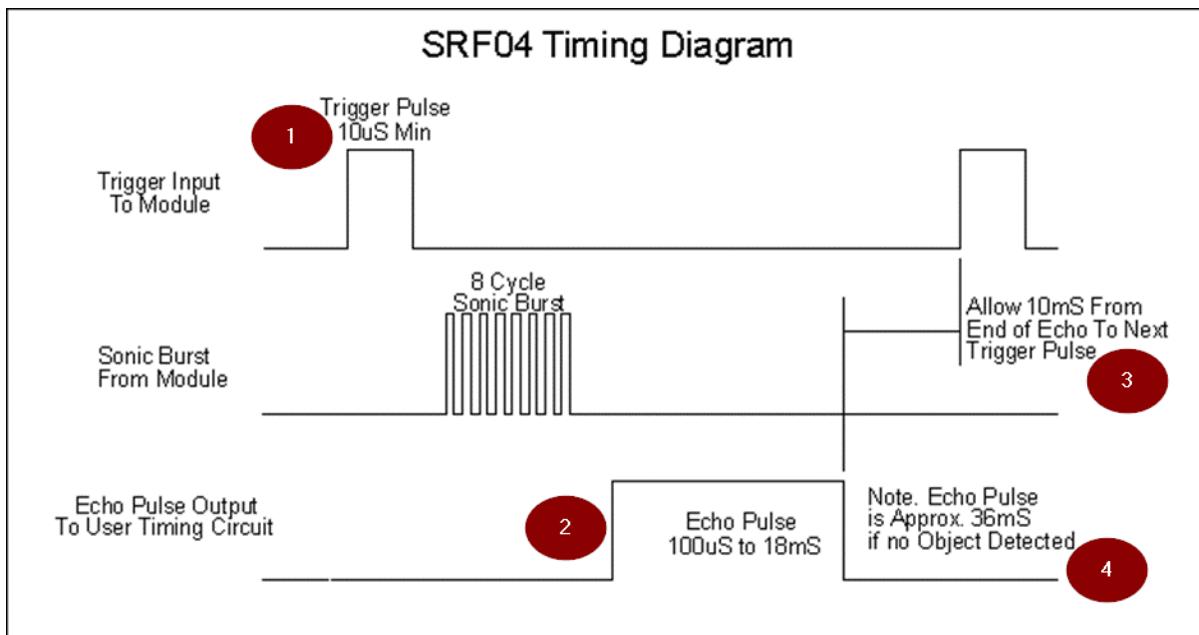
GND- زمین ماژول می‌باشد.

Trig\_ پایه تحریک ماژول می‌باشد. همانطور که در شکل ۲ مشاهده می‌کنیم برای فعال سازی این ماژول باید به مدت 10us (گزینه ۱) این پایه را یک کنیم.

Echo- سیگنال صوتی ارسال شده را در هنگام بازگشت دریافت می‌کند. همانطور که در شکل ۲ مشاهده می‌کنیم مدت زمان مجاز دریافت سیگنال بین 100us تا 18ms (گزینه ۲) می‌باشد.



(شکل ۱)



(شکل ۲)

نکته ۱: حداقل و اکثر فاصله قابل اندازه گیری این مژوول بین 3cm تا 300cm می باشد.

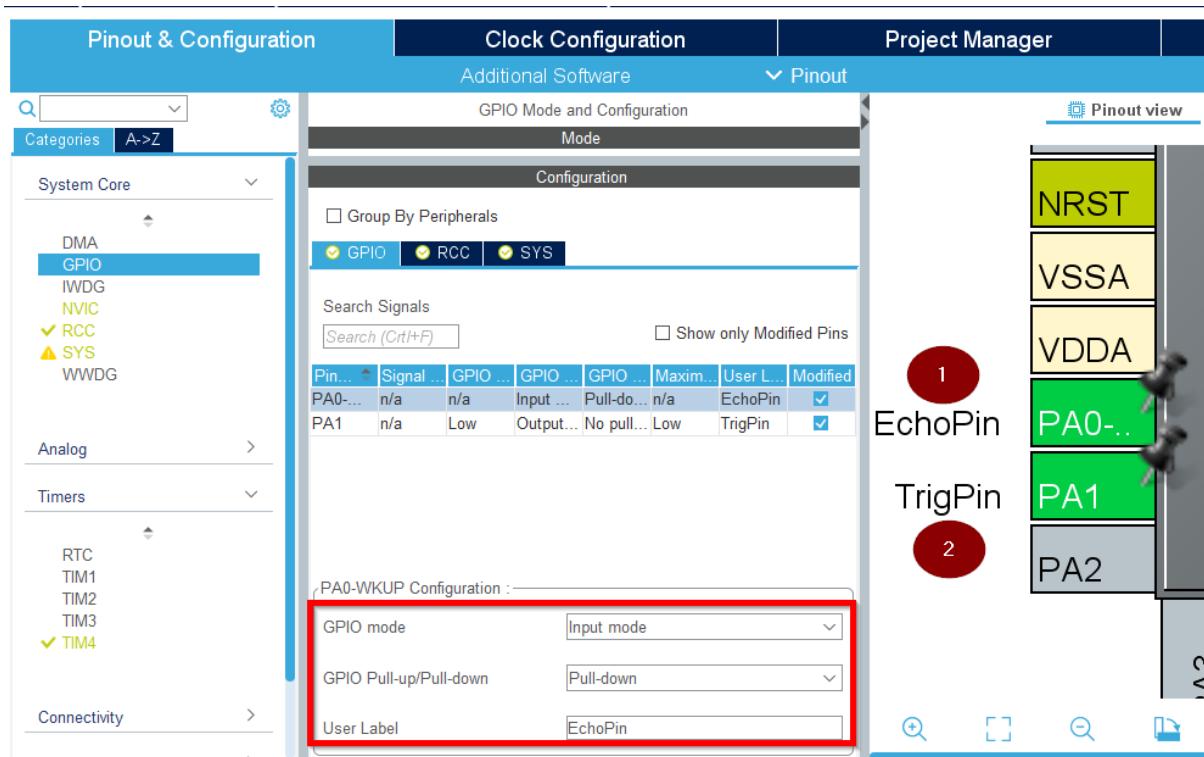
نکته ۲: همانطور که در شکل ۲ مشاهده می کنیم فاصله زمانی هر بار ارسال صوت توسط مژوول حداقل 10ms می باشد.

نکته ۳: همانطور که در شکل ۲ مشاهده می کنیم مدت زمانی که پایه Echo صبر می کند تا سیگنال صوتی را دریافت نماید 36ms (گزینه ۴) می باشد. در صورتی که مانعی در مقابل مژوول وجود نداشته باشد سیگنال بازگشتن نیز وجود ندارد لذا پایه Echo سیگنالی را حس نمی کند.

- حال Cube را باز می کنیم تا مقدمات نوشتن برنامه و تنظیمات اولیه را فراهم سازیم.

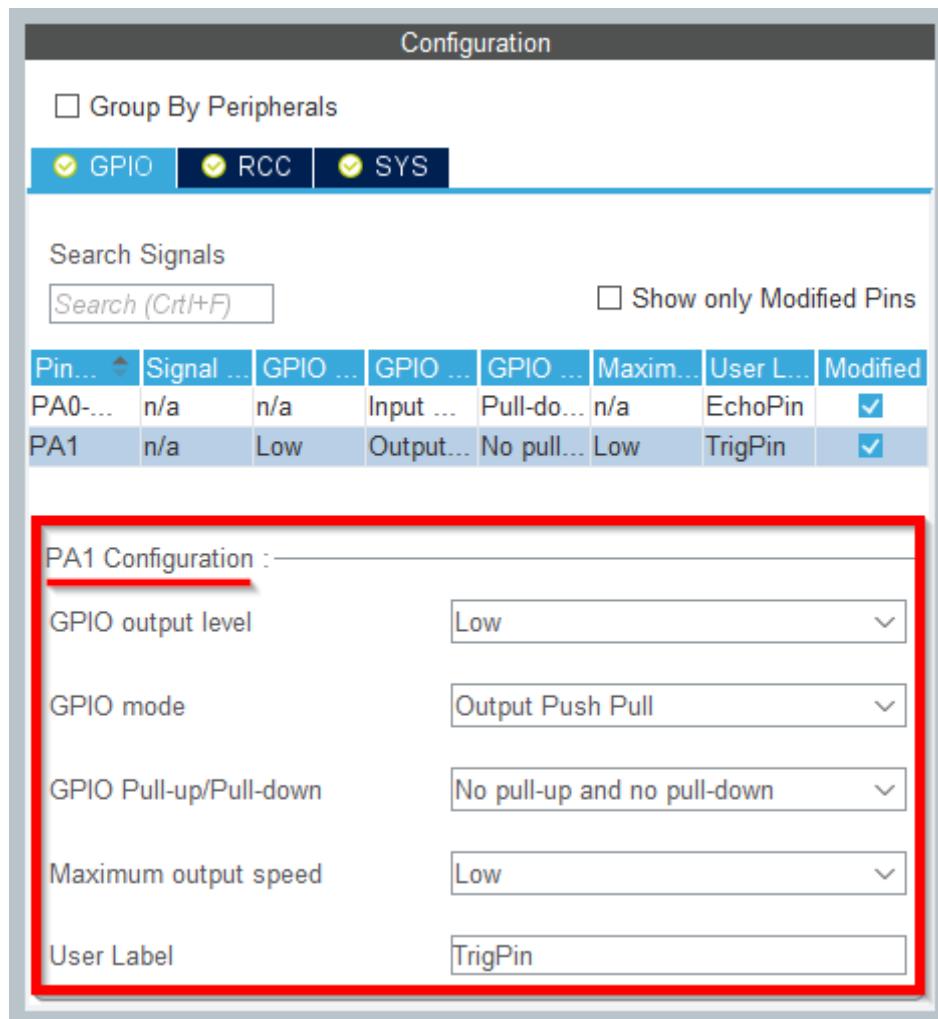
- ابتدا تنظیمات اولیه را طبق روال انجام می دهیم.

- پایه Echo را به عنوان پین ورودی تنظیم و آن را Lable گذاری می نماییم (گزینه ۱). همچنین تنظیمات مورد نیاز را مطابق شکل ۳ انجام می دهیم.



(۳) شکل

-۴ پایه Trig را به عنوان پین خروجی تنظیم و آن را گذاری می‌نماییم (گزینه ۲ شکل ۳). همچنین تنظیمات مورد نیاز را مطابق شکل ۴ انجام می‌دهیم.

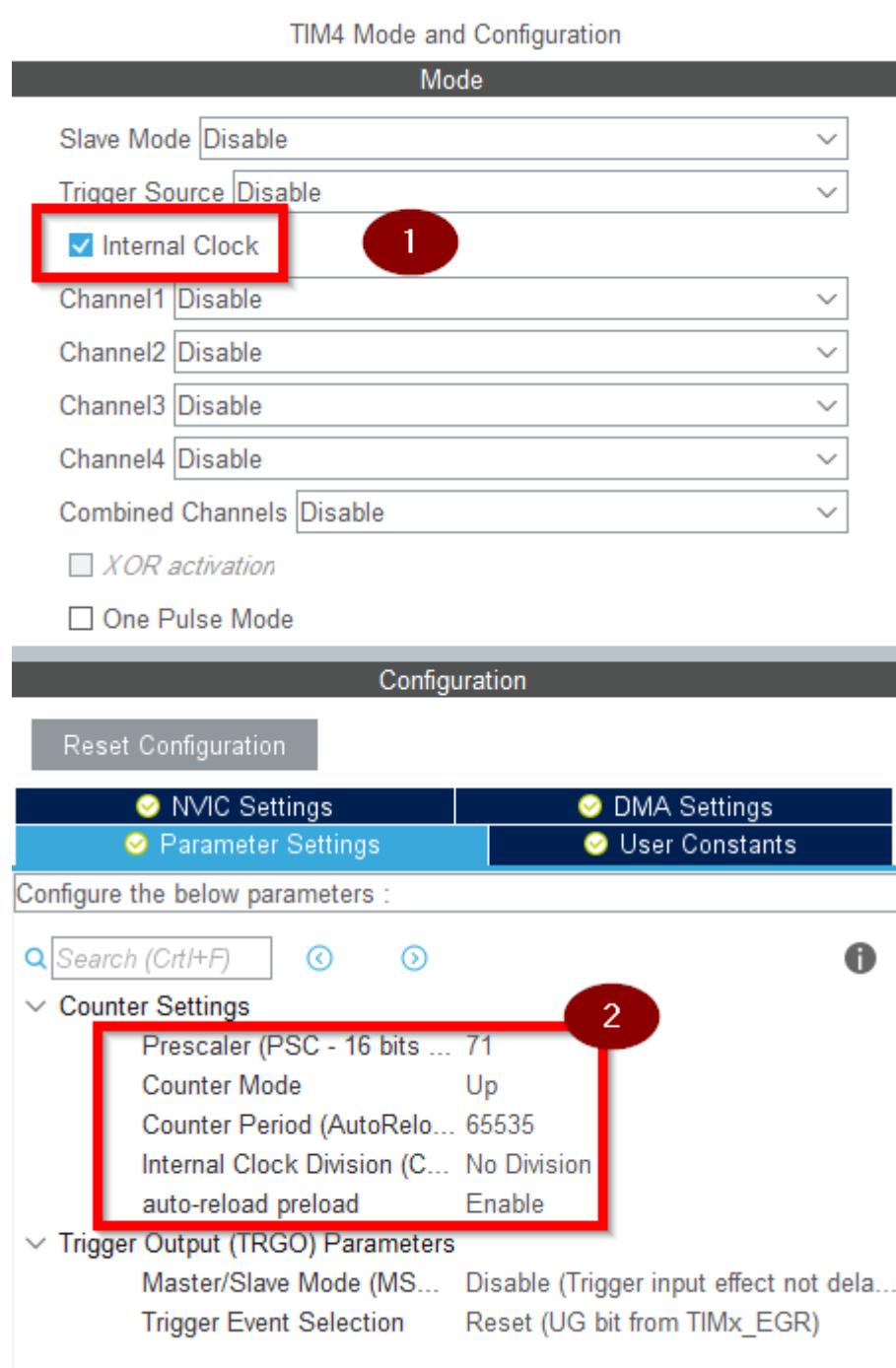


(۴) شکل

- برای اندازه گیری زمان در قسمت های مختلف برنامه نیاز به فعال سازی تایمر در مد Time Base داریم. کمترین زمان مورد نیاز 10us می باشد به همین دلیل PSC را ۷۱ انتخاب می کنیم تا زمان سرریز شدن برای ما اهمیتی ندارد بنابراین بیشترین مقدار را برای ARR اعمال می کنیم که برابر ۶۵۵۳۵ می باشد چون صرفاً شمردن برای ما اهمیت دارد. بنابراین هر کلاک CNT برابر 1us می شود.

$$F_{CK-CNT} = \frac{F_M}{PSC + 1} = \frac{72}{71 + 1} = 1MHz$$

$$T_{CK-CNT} = \frac{1}{F_{CK-CNT}} = 1us$$



(شکل ۵)

۶- حال برنامه را با توجه به نکات گفته شده به شرح ذیل می‌نویسیم:

```
float Time;
float Distance;
```

۱۵

```
1- while (1)
2- {
```

```
3-
4-
    HAL_GPIO_WritePin(TrigPin_GPIO_Port, TrigPin_Pin, GPIO_PIN_SET);
5-    __HAL_TIM_SET_COUNTER(&htim4, 0);
6-    HAL_TIM_Base_Start(&htim4);
7-    while(__HAL_TIM_GET_COUNTER(&htim4)<20)
8-    {
9-
10-    }
11-    HAL_TIM_Base_Stop(&htim4);
12-
13-    HAL_GPIO_WritePin(TrigPin_GPIO_Port, TrigPin_Pin, GPIO_PIN_RESET);
14-
15-    __HAL_TIM_SET_COUNTER(&htim4, 0);
16-    HAL_TIM_Base_Start(&htim4);
17-
18-
    while(HAL_GPIO_ReadPin(EchoPin_GPIO_Port, EchoPin_Pin==GPIO_PIN_RESET))
19-    {
20-        if(__HAL_TIM_GET_COUNTER(&htim4)==36000)
21-        {
22-            HAL_TIM_Base_Stop(&htim4);
23-            break;
24-        }
25-
26-    }
27-    __HAL_TIM_SET_COUNTER(&htim4, 0);
28-    HAL_TIM_Base_Start(&htim4);
29-
30-    while(HAL_GPIO_ReadPin(EchoPin_GPIO_Port, EchoPin_Pin==GPIO_PIN_SET))
31-    {
32-        if(__HAL_TIM_GET_COUNTER(&htim4)==36000)
33-        {
34-            HAL_TIM_Base_Stop(&htim4);
35-            break;
36-        }
37-        HAL_TIM_Base_Stop(&htim4);
38-
39-
```

```

40-      Time= __HAL_TIM_GET_COUNTER(&htim4) * 0.000001 *
0.5;
41-      Distance=3400 * Time * 100;
42-
43-
44-      }
45-  }

```

کد ۲

#### ۷- نکات تکمیلی پیرامون کد نوشته شده:

الف- در ابتدای برنامه (خط ۴) پایه Trig را روشن می کنیم، همانطور که در توضیحات این مأژول گفته شد باید پایه Trig به مدت 10us یک شود، یعنی معادل ۱۰ تا CNT ولی برای اطمینان خاطر آن را ۲۰ می گذاریم، برای این کار شمارنده CNT را صفر می کنیم (خط ۵) و سپس تایمر را Start می کنیم (خط ۶)، سپس یک حلقه While می نویسیم بطوریکه تا زمانی CNT کوچکتر از ۲۰ است هیچ اتفاقی نیوفتد، بنابراین وقتی از حلقه While خارج می شود 20us گذشته است (خط ۷ تا ۱۰). در پایان این فرایند دوباره تایمر را Stop می کنیم (خط ۱۱) و پایه Trig را صفر می کنیم (خط ۱۳).

ب- حال باید منتظر دریافت سیگنال صوتی بازگشتی و ذخیره مدت زمان این رفت و برگشت سیگنال صوت باشیم، بدین منظور یک حلقه While (خط ۱۸) می نویسیم بطوریکه تا زمانی که سیگنالی دریافت نکرده مقدار پایه Echo صفر است و هیچ اتفاقی نمی افتد، نکته حائز اهمیت این است که ممکن است هیچ وقت این پایه یک نشود به عبارتی مانع در جلوی مأژول نباشد که در این صورت باید میکرو در حلقه while متوقف شده و هیچ کار مفید دیگری انجام ندهد، به همین دلیل یک TimeOut برای آن در نظر می گیریم که در صورتی بعد از یک زمان معین پایه Echo یک نشد میکرو از حلقه Whlie خارج شود، در اینجا مقدار TimeOut را 36000us (طبق نکته ۲، مدت زمانی که پایه Echo صبر می کند تا سیگنال صوتی را دریافت نماید 36ms می باشد) انتخاب می کنیم (به همین دلیل قبل از While در خط ۱۵ و ۱۶ تایمر را Start کردیم). در صورتی که TimeOut رخ ندهد (یعنی مانع وجود داشته و پایه Echo یک شده) با یک شدن پایه Echo از حلقه While خارج می شود لذا پس از خارج شدن از حلقه While بلا فاصله دوباره تایمر را Start (خط ۲۷ و ۲۸) می کنیم. تا چه زمانی تایمر زمان را اندازه گیری کند؟ تا زمانی که کل سیگنال بازگشتی دریافت شود به عبارتی تا زمانی که پایه Echo یک باشد (خط ۳۰). پس از صفر شدن پایه Echo، میکرو از حلقه While خارج می شود و تایمر را Stop می کنیم سپس مقدار CNT تایمر را در یک متغیر مانند T ذخیره

می کنیم و با توجه به اینکه بر حسب میکرو ثانیه می باشد آن را در  $10^{-6}$  ضرب می کنیم، همچنین همانطور که می دانیم زمان بدست آمده زمان رفت و برگشت می باشد بنابراین مقدار آن را نصف می کنیم (خط ۴۰).

پ- با توجه به اینکه سرعت صوت  $340 \text{ m/s}$  می باشد برای بدست آوردن فاصله، آن را در مدت زمان طی شده (T) ضرب می کنیم و برای تبدیل متر به سانتی متر آن را در صد ضرب می کنیم (خط ۴۱).

## ٨ فصل

مبدل آنالوگ به دیجیتال

Analog to Digital Converter  
(ADC)

## ۱-۸ مبدل آنالوگ به دیجیتال

ADC<sup>۱</sup> مخفف و به معنی تبدیل سیگنال آنالوگ به دیجیتال در داخل میکرو کنترلر می باشد. بطور مثال یک سیگنال آنالوگ روی پایه میکرو داریم که ولتاژ آن بین صفر تا  $3/3$  ولت است و می خواهیم آن را به یک عدد دیجیتال تبدیل کنیم (این سیگنال می تواند سیگنال منشور نور، صدا، دما و یا هر چیز دیگری باشد). اگر بخواهیم آن را با pin\_read بخوانیم چون تنها دو ورودی صفر و یک را می خواند (resetی set) به عبارتی از یک محدوده به پایین را صفر و از یک محدوده به بالا را یک می خواند، پس نمی تواند مقادیر مختلف را بفهمد و مقایسه کند. در اینصورت از واحد ADC استفاده می کنیم که هر مقداری بین  $0$  تا  $3/3$  ولت را در داخل میکرو به یک عدد دیجیتال تبدیل می کند.

12 بیتی بودن ADC به این معناست که ورودی  $0$  ولت با عدد  $0$  و ورودی برابر با VREF+ با عدد  $4095$  در رجیستر خروجی ADC (که<sup>۲</sup> DR نام دارد) متناظر می شود. هر ولتاژ آنالوگی در این بین بصورت خطی به یک عدد صحیح متناظر می گردد. شایان ذکر است که در صورت اعمال ولتاژ منفی به پایه ADC، مقدار  $0$  در رجیستر خروجی ظاهر می گردد. دقت شود که حتی الامکان از اعمال ولتاژ منفی به ADC اجتناب کنید، چرا که احتمال ایجاد خطأ در میکرو کنترلر را افزایش می دهد.

از آنجائیکه تغذیه واحد ADC با توجه به دقت بالای آن باید ریپل بسیار کمی داشته باشد، از تغذیه سایر بخش های میکرو کنترلر جدا بوده و توسط دو پایه مجزای VDDA و VSSA تأمین می شود. البته VSSA به VSS که زمین میکرو کنترلر است متصل می شود. شایان ذکر است که همواره در ورودی پایه VDDA از یک فیلتر سلفی - خازنی برای کاهش نویز استفاده می گردد.

می دانیم بازه ولتاژ آنالوگ قابل قرائت توسط میکرو کنترلر، از  $0$  ولت تا  $3,3$  ولت است. علت حضور پایه VREF+ در میکرو کنترلر آن است که دقت تفکیک مقادیر آنالوگ، در صورتیکه بازه تغییرات سیگنال ورودی کمتر از بازه فوق باشد، افزایش یابد. بعنوان مثال، بیشترین خروجی IC سنسور دما (LM35) در حالات عادی برابر  $1$  ولت (متناظر با  $100$  درجه سانتی گراد) است. در اینصورت، مقدار VREF+ را برابر روی  $1$  ولت تنظیم می کنیم تا دقت تفکیک داده های آنالوگ ورودی افزایش یابد (در اینجا  $3/3$  برابر شود). در برخی از انواع میکرو کنترلرهای پایه دیگری با نام VREF- موجود است که بطور مشابه می توان با استفاده از آن، کوچک ترین ولتاژ آنالوگ ورودی را بجای صفر ولت حالت پیش فرض تنظیم نمود.

بحث های ذیل با توجه به میکرو کنترل STM32F103C8TX می باشد.

<sup>1</sup> Analog to Digital Converter

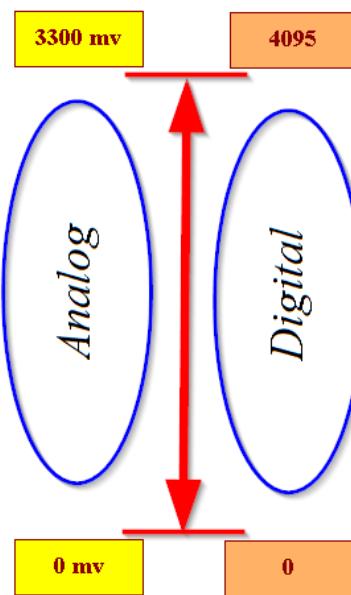
<sup>2</sup> Data Register

فرضاً روى پايه ميكرو يك ولتاثر ۰ تا ۳۳۰۰ ميلى ولت داريم و مى خواهيم مقدار آن را بخوانيم.  
 اگر مقدار پايه صفر ولت باشد آن را صفر و اگر ۳۳۰۰ ميلى ولت باشد آن را ۴۰۹۵ ( $2^{12} - 1 = 4096$ ) مى خواند که عددی ديجيتال است.

همانطور که مشاهده مى كنيد دقت اندازه گيري ولتاثر ۸۰۵uv مى باشد که دقت خيلي بالايي محسوب مى شود.  

$$\frac{3300mv}{4095} = 805 uv$$

Sampling و Hold مدت زمانی طول مى كشد که مقدار آن از ۱.۵ تا ۱۸ سيكل (هرچه بيشتر شود کيفيت بيشتر و دقت كمتر مى شود) و ۱۲.۵ سيكل تبديل است. با فرض ۱.۵ سيكل بودن sampling، اين فرایند ۱۴ سيكل يا کلاک زمان مى برد و با توجه به اين که فرکانس کاري ADC آن ۱۴MHz است به عبارتی فرکانس نمونه برداری ۱۶سيكل مى باشد فرکانس هر کلاک  $\frac{14MHz}{14clk} = 1MHz$  است. حال در cube بيش ترين مقداری که با فرکانس ۷۲MHz مى شود برای آن تعين ۱.  $\frac{14}{12MHz}$  است لذا فرکانس نمونه برداری برابر  $\frac{12MHz}{14clk} = 857KHz$  مى شود و هر تبديل در ۱۶us انجام مى شود.



نکته: مهم‌ترین مشخصه که  $ADC$  ها از هم متمایز می‌کند تعداد بیت (دقت تبدیل) و سرعت تبدیل نکته:  $STM32F103C8TX$  :توانایی خواندن ورودی از ۱۰ سنسور را دارد.

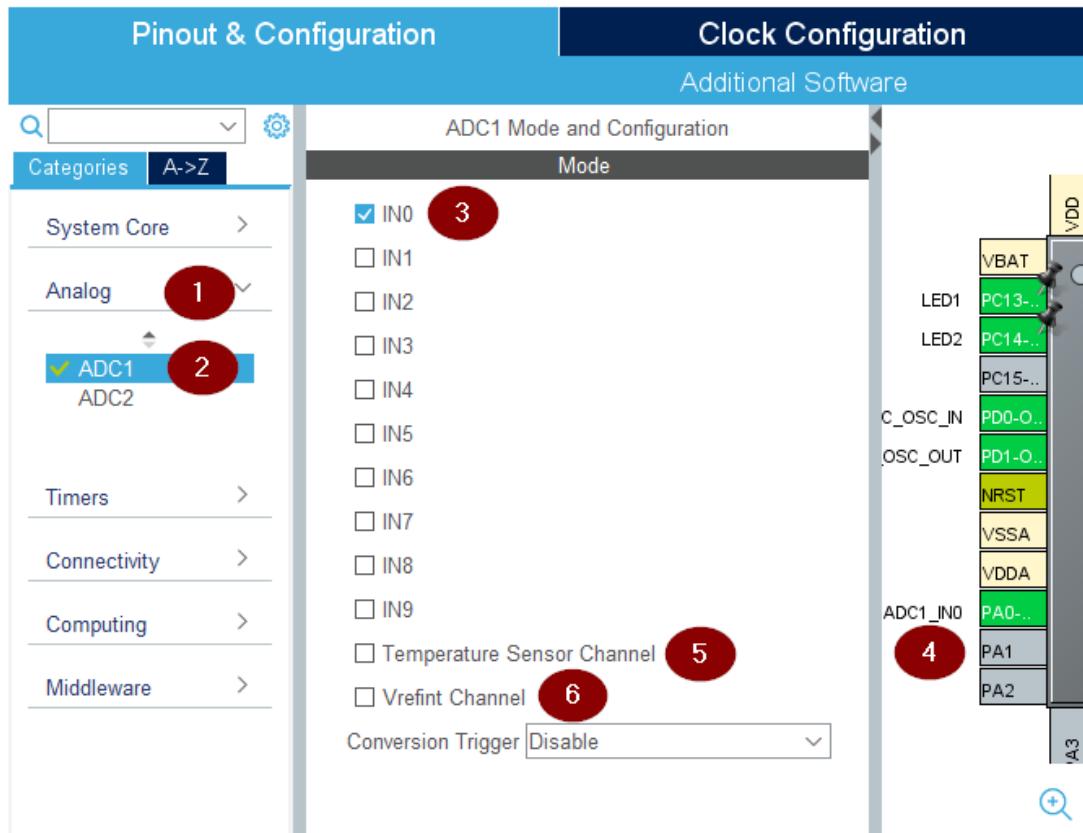
دقت تبدیل آنالوگ به دیجیتال در اکثر میکروکنترلهای ARM (از جمله STM32 ) برابر با ۱۲ بیت است. بیشترین سرعت تبدیل نیز در سری STM32F1 با معماری Coretex M3 برابر با  $1MSample/sec$  می‌باشد، البته در صورت موازی‌سازی واحدهای ADC ، سرعت نمونه‌برداری می‌تواند تا چندین برابر افزایش یابد .(این در حالیست که در میکروکنترلهای AVR این مقدار برابر  $15kSample/sec$  است. سرعت بالای تبدیل در میکروکنترلهای ARM بیشتر در کاربردهای پردازش صوت اهمیت می‌یابد. شایان ذکر است که در میکروکنترلهای سری STM32F4 و STM32F1 بترتیب ۲ و ۳ کانال مستقل ADC وجود دارد. هر یک از این کانال‌ها قابلیت دریافت تعدادی ورودی (بعنوان نمونه ۱۰ عدد در سری STM32F103 ) را داشته که در هر لحظه، یکی از آنها بوسیله یک مالتی‌پلکسor به واحد ADC متصل می‌شود.

- نکته بسیار مهم: فرکانس سیگنالی که قرار است از آن نمونه‌برداری شود باید حد اکثر نصف فرکانس نمونه برداری باشد.

$$f_{signal} \leq \frac{F_{sampling}}{2}$$

مثال: خواندن مقدار سیگنال آنالوگ از پایه ADC :

- ۱ Cube را باز میکنیم و تنظیمات اولیه را طبق روال انجام می‌دهیم (شکل ۱).
- ۲ از قسمت Analog (گزینه ۱)، ترجیحها ADC1 (گزینه ۲) را باز می‌کنیم، و کانال‌هایی را که می‌خواهیم سنسور آنالوگ را به آن وصل کنیم انتخاب می‌کنیم. بطور مثال با انتخاب IN0 (گزینه ۳)، پورت A0 به ADC0\_IN0 (گزینه ۴) تبدیل می‌شود.
- ۳ Temperature sensor Chanel (گزینه ۵): یک سنسور دمایی داخل خود میکروکنترلر که با دقت مناسبی دمای داخلی میکروکنترلر را اندازه‌گیری می‌کند.
- ۴ VRefint Chanel (گزینه ۶): ولتاژ آنالوگ داخلی میکروکنترلر که صرفاً جهت انجام تست‌ها منظور شده است (جهت تست صحبت قرائت داده).



(شکل ۱)

## ۵- تنظیمات قسمت Configuration (شکل ۲):

گزینه ۱- Mode: این گزینه، حالت عملکرد واحدهای ADC را تعیین می‌کند. در میکروکنترلرهایی که تنها یک واحد ADC دارند، تنها انتخاب Independent mode است. اما در میکروکنترلرهایی با بیش از یک واحد ADC، می‌توان چندین مبدل آنالوگ به دیجیتال را بصورت موازی با هم راهاندازی نمود و سرعت را با توجه تعداد واحدها دو تا سه برابر نمود. بطور مثال اگر ADC2 و کanal IN0 را انتخاب کنیم سرعت برابر می‌شود.

گزینه ۲- Data Alignment: با توجه به اینکه رجیستر خروجی ADC دارای ۱۶ بیت بوده، اما دقت تبدیل برابر ۱۲ بیت است، ۴ بیت بدون استفاده باقی می‌ماند. این گزینه بیانگر جهت چینش نتیجه تبدیل در رجیستر خروجی ADC است. با انتخاب Right aligned نتیجه تبدیل از سمت راست در رجیستر قرار گرفته و ۴ بیت سمت چپ خالی می‌ماند. انتخاب Left aligned نتیجه عکس در بر دارد. در غالب کاربردها از Right aligned استفاده می‌شود.

گزینه ۳- Scan Conversion Mode: در صورتیکه قصد قرائت بیش از یک کanal از ADC را داشته باشیم، باید بین کanal های ورودی مالتیپلکسر سوئیچ کنیم. برای این منظور باید این گزینه را فعال کنیم تا

میکروکنترلر دائمًا تمامی کانال‌ها را اسکن کرده و پس از هر تبدیل، کانال ورودی را تغییر دهد. اما در صورتیکه تنها از یک کانال ADC استفاده می‌کنید، این گزینه را بر روی Disabled قرار دهید. وقت شود تا زمانیکه مقدار فیلد Scan Conversion Mode برابر با ۱ باشد، فیلد Number of Conversions نمی‌تواند فعال (Enabled) شود.

گزینه ۴- Continuous Conversion Mode: اگر قصد دارید خواندن ورودی و تبدیل آنالوگ مکرراً صورت گیرد، این گزینه را فعال (Enable) کنید. در غیر اینصورت (اگر تنها یک مرتبه قصد خواندن ورودی را دارید)، آنرا بر روی Disabled تنظیم کنید. در این حالت، باید پیش از هر تبدیل، یک مرتبه start ADC را نمود.

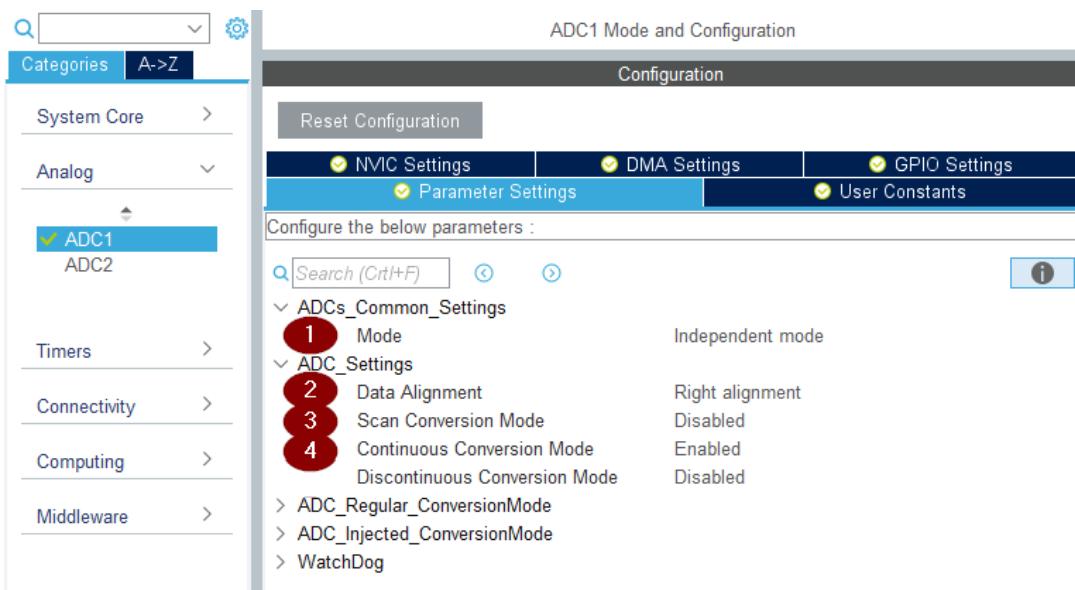
گزینه ۵- Enable Regular Conversions: برای فعال سازی تبدیل آنالوگ به دیجیتال باید این گزینه را بر روی Enabled تنظیم کنیم. Regular در مقابل حالت Injected قرار می‌گیرد.  
گزینه ۶- Number of Conversions: تعداد کانال‌های ورودی آنالوگ را که قصد خواندن را داریم در این قسمت وارد می‌کنیم. شایان ذکر است، تا زمانیکه مقدار این فیلد برابر با ۱ باشد، فیلد Scan Conversion Mode نمی‌تواند فعال (Enabled) شود.

گزینه ۷- External Trigger Conversion Source: به کمک این گزینه می‌توان منبع خارجی فعال ساز تبدیل آنالوگ به دیجیتال را مشخص نمود. در صورت انتخاب گزینه Regular Conversion Launched by Software، فعال سازی واحد ADC توسط دستور مربوطه (برنامه‌نویسی) و نه از طریق منبع تحریک خارجی انجام می‌گیرد. اما با انتخاب گزینه Timer x Trigger Out event پس از هر بار سرریز تایمر مربوطه، دستور تبدیل برای واحد ADC صادر می‌شود.

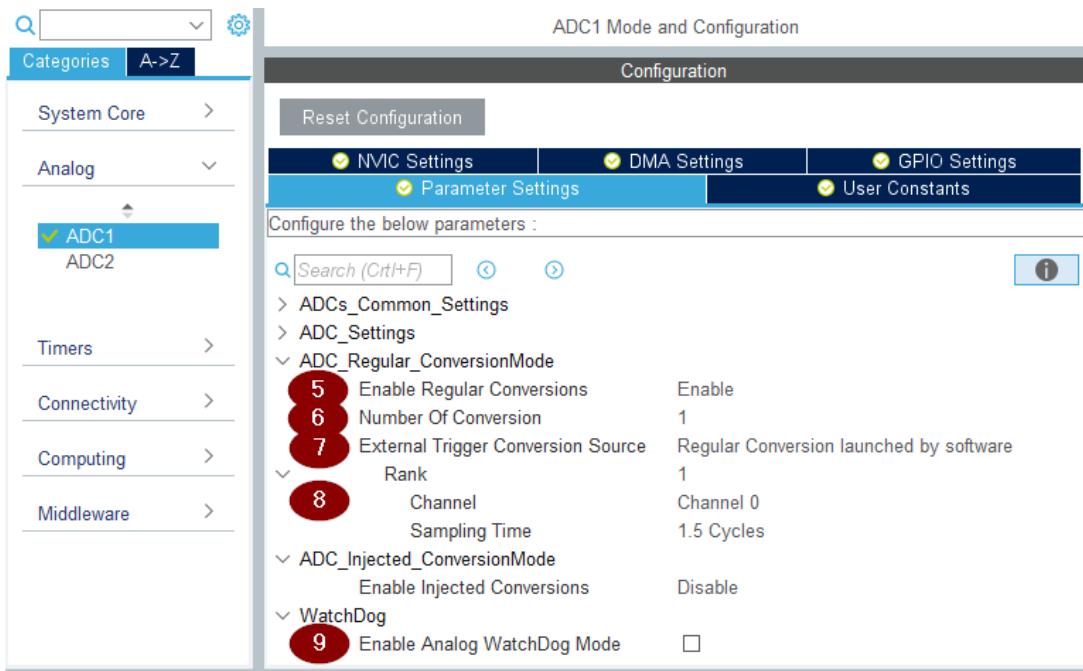
گزینه ۸- Rank: به تعداد کانال‌هایی که در قسمت Pinout فعال کرده‌اید، Rank در قسمت ADC1 Configuration ظاهر می‌شود. Rank ترتیب خواندن (اسکن) کانال‌ها را مشخص می‌کند. دو پارامتر در هر Rank قابل تنظیم است؛ یکی Channel که شماره کانال انتخابی است، و دیگری Sampling Time که مدت زمانیست که میکروکنترلر از داده نمونه‌برداری می‌کند. هرچه این مدت زمان بزرگ‌تر انتخاب شود، پایداری داده آنالوگی که قصد خواندن آنرا داریم، افزایش می‌یابد. اما قابلیت خواندن داده‌های فرکانس بالا را از دست می‌دهیم. این پارامتر در غالب کاربری‌ها برابر  $1/5$  سیکل است.

گزینه ۹- Analog Watchdog Mode: این حالت، در اصل یک مقایسه‌گر است. با فعال سازی این گزینه، در صورتیکه مقدار خوانده شده از ADC خارج از بازه‌ای مابین حد تحتانی (Low Threshold)

و حد فوقانی (High Threshold) باشد، میکروکنترلر بصورت خودکار، وقفه‌ای را فعال می‌کند که می‌توان از آن استفاده نمود. در ذیل تنظیمات مربوطه، Watchdog Mode تعیین کننده نوع (Regular یا Injected) و تعداد کانال‌هایی است که می‌خواهیم Watchdog بر روی آنها نظارت داشته باشد. البته تنها دو انتخاب داریم؛ یکی فقط نظارت بر یک کانال (Single regular channel) و دیگری تمامی کانال‌های ADC داریم. گزینه Analog watchdog Channel (All regular channels) دقت شود که مقادیر High Threshold و Low threshold باید بصورت دیجیتالی وارد شوند؛ یعنی مقدار معادل ولتاژ آنالوگ مورد نظر در بازه 0 تا 4095 در نهایت، گزینه Interrupt Mode جهت فعال‌سازی وقفه مربوط به Watchdog است، که در صورت Enable شدن، با اجرا شدن وقفه (خروج از محدوده معین شده)، پردازنده به وقفه سرویس‌دهی می‌کند.



(شکل ۲)



(شکل ۳)

-۶ Generate Project را می‌زنیم تا فایل Keil ساخته شود.

-۷ واحد ADC تنظیم شده و آماده کار می‌باشد ولی مانند تایمر تا زمانی که آن را Start نکنیم تبدیل آن شروع نمی‌شود. با توجه به اینکه در مد Continuous تنظیم شده یکبار که Start شود تا انتها تبدیل را انجام می‌دهد و اطلاعات را در Regular Data Register میریزد ولی در مد Discontinuous باید هر بار که نیاز باشد مجدداً فعال شود. همچنین طبق روال با زدن F12 به تعریف تابع می‌رویم تا تعداد ورودی و نوع آنها مطلع شویم.

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

```
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc)
```

-۸ حال باید یک متغیر تعریف کنیم تا نتایج تبدیل را در آن بریزیم، با توجه به ۱۶ بیتی بودن نتایج تبدیل (البته نتایج تبدیل ۱۲ بیتی است ولی تعداد خانه‌های Data Register ۱۶ بیتی می‌باشد) یک متغیر ۱۶ بیتی تعریف می‌کنیم. ولی به دلایلی که بعداً آشنا می‌شویم آن را ۳۲ بیتی انتخاب می‌کنیم.

```
Uint32_t ADCResult = 0;
```

9- دستور گرفتن نتایج تبدیل را با استفاده از کتابخانه Hal نوشته و آن را داخل متغیر (ADCRsult) که تعريف کرده‌ایم میریزیم(داخل حلقه(1)).

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

10- برنامه را Load می‌کنیم، همانطور که در Debug مشاهده می‌کنید با توجه به اینکه پایه ADC را به تغذیه 3.3V وصل کرده‌ایم باز هم ۴۰۹۵ مشاهده نمی‌شود که علت آن این است که مقدار پایه دقیقاً ۳.۳V نیست لذا مسئله مهمی نیست ولی مشکلی که باید به آن توجه کرد این است که مقداری نویز وجود دارد که مقدار خوانده شده همواره در یک بازه تقریباً ۱۰ تایی بالا و پایین می‌رود که البته بسته به شرایط محیطی این مقدار ممکن است خیلی بیشتر شود(شکل ۳ و ۴).

Watch 1		
Name	Value	Type
ADCRsult	4040	unsigned s...
<Enter expression...		

(شکل ۴)

Watch 1		
Name	Value	Type
ADCRsult	4031	unsigned s...
<Enter expression...		

(شکل ۵)

11- با توجه به اینکه این نویز روندی تکرار شونده دارد به همین دلیل میانگین آن صفر است لذا از آن میانگین می‌گیریم تا بازه تغییرات را به حداقل برسانیم(در برنامه‌های بعدی آموزش داده می‌شود).

## ۱-۱-۸ روش های استفاده از ADC

باتوجه به اینکه کد بند ۹ در While نوشته شده با سرعت خیلی بالایی (72MHz) خوانده می شود، در حالی که سرعت ADC 857 KHz (هر 1.16us یک تبدیل صورت می پذیرد) می باشد. یعنی یک مقدار چندین بار خوانده می شود لذا باید کاری کنیم که در هر بار خواندن متوجه شویم که داریم مقدار جدید را می خوانیم و داده قبلی نیست. راه حل؟

## ۱-۲-۱ وقفه تایمری

می توان تایمری تنظیم کرد که هر 1.2us آپدیت شود و اینتراپت بدهد و هر بار که اینتراپت زده شد مقدار تبدیل را بخوانیم که در اینصورت مطمئن هستیم مقدار جدید را می خوانیم. اما مشکل این روش این می باشد میکرو را به شدت درگیر می کند چون هر 1.2us فرآخوانی می شود و باید عملیاتی را انجام دهد، به این دلیل روش مناسبی نمی باشد.

## ۳-۱-۸ Pollforconversion

در این روش، زمانی که Flag مربوط به<sup>۱</sup> EOC یک شود یعنی تبدیل صورت گرفته است به همین دلیل دستور (۱) را می نویسیم. با زدن F12 به تعریف تابع می رویم. مشاهده می کنیم (کد ۲) ورودی اول Handeler تابع می باشد که در اینجا ADC است و دومی Timeout می باشد. تازمانی که تبدیل صورت نگیرد میکرو از این کد عبور نمی کند و مشکل اینجاست که اگر به هر دلیلی تبدیل صورت نگیرد میکرو قفل می شود، که برای رفع این مشکل از زمان TimeOut استفاده گردیده که در اینجا بطور مثال 20ms فرصت داده شده که ADC مقدار تبدیل شده را تحويل دهد، در غیر اینصورت از آن عبور می کند (البته همانطور که می دانید هر تبدیل در 1.16us انجام می شود و 20ms بیشتر جنبه امنیتی دارد که در صورت رخ دادن مشکل در پایه ADC ، کار متوقف نشود ) پس باید خروجی تابع یعنی HAL\_StatusTypeDef (کد ۲) را چک کنیم تا بدانیم خروجی به دلیل صحیح بودن شرایط (HAL\_OK) میکرو از کد ۱ عبور کرده یا اینکه مشکل داشته (HAL\_ERROR) و با اتمام زمان TimeOut از آن عبور کرده است، بدین منظور بر روی آن (کد ۲)، F12 را می زنیم که خروجی های آن به شرح ذیل می باشد (کد ۳). یک متغیر به نام Readstatus (کد ۴) تعریف می کنیم و خروجی تابع را در آن می ریزیم. حال کد نهایی بدین گونه می شود (کد ۵).

<sup>۱</sup> End of Conversion

```
HAL_ADC_PollForConversion(&hadc1, 20);
```

1

```
HAL_StatusTypeDef  
HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t  
Timeout)
```

2

```
typedef enum  
{  
    HAL_OK          = 0x00U,  
    HAL_ERROR       = 0x01U,  
    HAL_BUSY        = 0x02U,  
    HAL_TIMEOUT     = 0x03U  
} HAL_StatusTypeDef;
```

3

```
HAL_StatusTypeDef Readstatus;
```

4

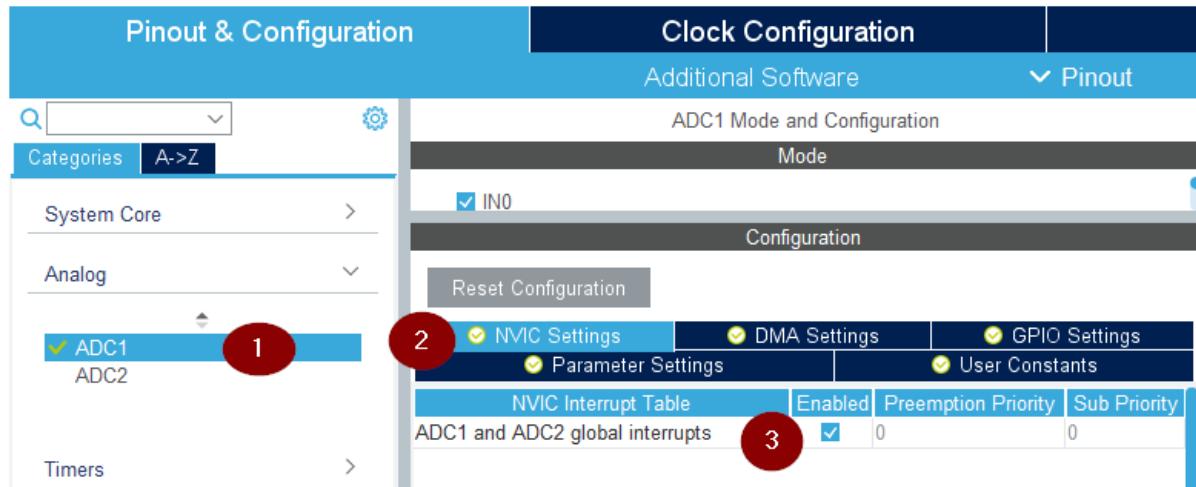
```
while (1)  
{  
    Readstatus=HAL_ADC_PollForConversion (&hadc1, 20);  
  
    if (Readstatus==HAL_OK)  
    {  
        ADCRslt = HAL_ADC_GetValue (&hadc1);  
    }  
}
```

5

## Intrupt ۴-۱-۱-۸

ابتدا از داخل تپ NVIC ، اینتراپت ADC را روشن می کنیم(شکل ۴) و دوباره پروژه قبلی را Generate ADC می کنیم و همچنین اینبار ADC را با حالت اینتراپتی فعال می کنیم(کد ۱). طبق روال برای پیدا کردن rootin اینتراپت بطور موقتی (پس از تامین خواسته آن را پاک می کنیم) دستور مربوط به ADC ، CallBack را مینویسیم(کد ۲) و سپس با زدن دکمه F12 ، rootin اینتراپت آورده می شود، لذا آن را کپی کرده و بعد از past می کنیم(کد ۳). در این روش دیگر نیاز نیست پیوسته مقدار ADC را Read کنیم. بلکه در صورت

تبديل شدن مقدار آن اطلاع داده می‌شود و فقط کافی است مقدار آن را داخل ADCResult ذخیره کنیم (کد ۴). البته باید یک دستور شرطی داخل اینترپت مشخص کنیم که تبدیل حاصل شده از کدام می‌باشد.



(شکل ۴)

```
HAL_ADC_Start_IT(&hadc1);
```

1

```
HAL_ADC_ConvCpltCallback
```

2

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
}
```

3

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if(hadc->Instance == ADC1)
    {
        ADCResult = HAL_ADC_GetValue(&hadc1);
    }
}
```

4

۸-۱-۱-۵ (روشنهایی DMA)

در ۹۹,۹ درصد موضع از روش<sup>۱</sup> DMA استفاده می‌کنیم.

طراحان میکروکنترلر یک مدار واسط به نام DMA مخفف (Direct Memory Access) را طراحی کرده‌اند که وظیفه‌ی آن تنها انتقال اطلاعات از peripheral Register به هر SRam باشد (SRam) و یا برعکس و همچنین از داخل SRam به SRam ولی نمی‌تواند اطلاعات را پردازش کند و همه این اتفاقات بدون دخالت CPU صورت می‌پزیرد که خیلی قابلیت مفیدی محسوب می‌شود.

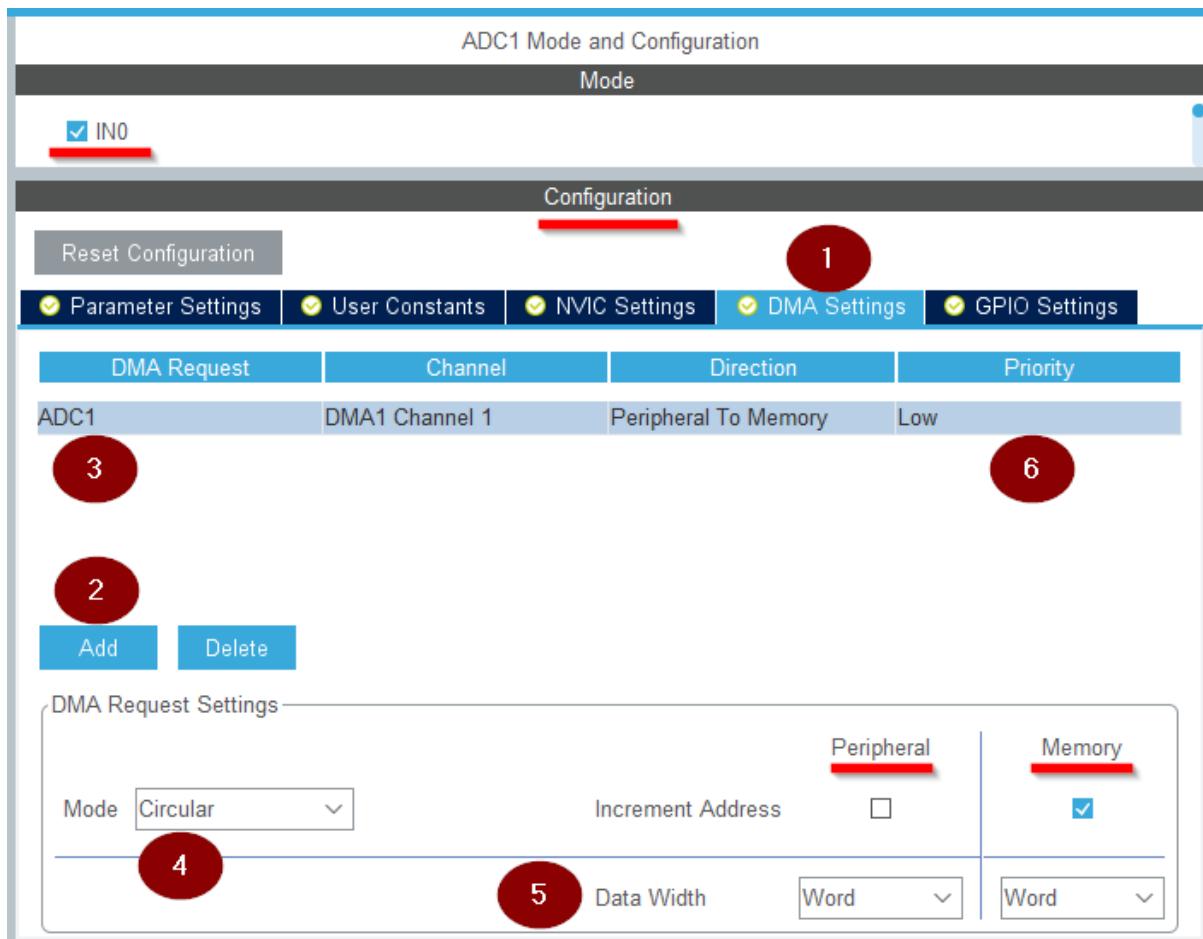
حال در قسمت ADCResult می‌خواهیم مقادیر تبدیل یافته را از Data Register به DMA توسط انتقال دهیم. بدین منظور Cube را باز کرده:

۱- تنظیمات اولیه را طبق روال انجام می‌دهیم و مانند حالت قبلی ADC1 ، IN0 را انتخاب کرده و تنظیمات Parameter Setting را انجام می‌دهیم.

۲- در تب Configuration ، DMA Setting (گزینه ۱) را انتخاب کرده و Add (گزینه ۲) را می‌زنیم و در ستون DMA Request (گزینه ۳) را انتخاب می‌کنیم. و در قسمت Mode (گزینه ۴) را انتخاب می‌کنیم. در قسمت Data Width پنهانی داده‌هایی (تعداد بیت داده‌ها) که قرار است از peripheral انتخاب شوند. اطلاعاتی که در ADC برداشته و به Memory انتقال یابند منتقل مشخص شوند. انتخاب می‌کنیم و در می‌شوند ۱۶ بیتی هستند ولی برای راحتی کار در ادامه همیشه آن را ۳۲ بیتی (Word) انتخاب می‌کنیم و در نتیجه Memory نیز باید ۳۲ بیتی انتخاب شود(گزینه ۵). نیز در صورت استفاده از چند ADC جهت تعیین اولویت بین آنها مشخص می‌شود (گزینه ۶) (شکل ۵).

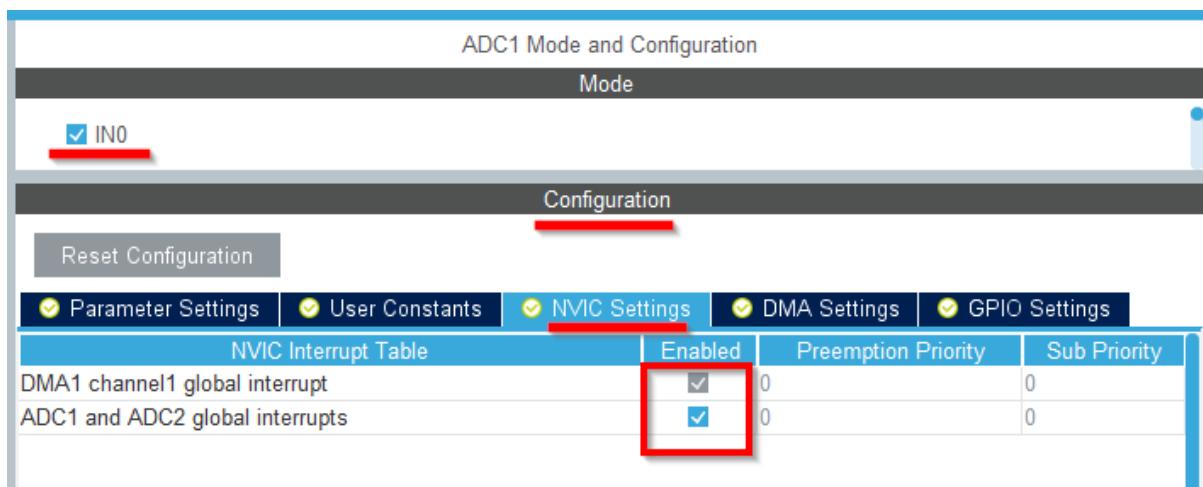
---

<sup>۱</sup> Direct Memory Access



(شکل ۵)

۳- در تب NVIC نیز Global Interrupts را فعال سازی می کنیم (اینترافت DMA اتوماتیک فعال می شود و امکان غیرفعال سازی آن وجود ندارد) (شکل ۶).



(شکل ۶)

۴- در پایان پروژه را Generate می کنیم تا فایل Keil ساخته شود.

۵- در این مرحله ADC را در حالت DMA ، START می کنیم(کد۱) و با زدن F12 به تعریف تابع (کد۲) آن

می رویم و از تعداد و نوع ورودی های آن مطلع می شویم. مشاهده می کنیم دارای سه ورودی می باشد که:

الف- ورودی اول Handeler آن می باشد(کد۱).

ب- ورودی دوم محلی که قرار است داده ها به آنجا منتقل شوند را از ما می خواهد. در موارد قبلی داده ها به

متغیر ADCResult منتقل می شد که یک خانه بود ولی در اینجا می خواهیم با توجه به دقت مورد نیاز،

چندین داده انتقال دهیم (جهت میانگین گیری برای از بین بردن نویز) بنابراین به چندین خانه نیازمندیم که

در اینجا باید آرایه (کد۳) درست نمود. این ورودی یک Pointer است ولی با توجه به اینکه مدیریت

کردن Pointer سخت است می توان آدرس درایه اول یک آرایه را برای آن معرفی کرد(کد۱).

نکته: تعداد درایه های آرایه که مشخص می شوند باید از تعداد داده هایی که قرار است منتقل شوند بیشتر

باشد(مثلاً دو برابر).

پ- ورودی سوم تعداد داده هایی که قرار است منتقل شوند نوشته می شوند(کد۱).

```
HAL_ADC_Start_DMA (&hadc1, ADCRsult, 50);
```

1

```
HAL_StatusTypeDef HAL_ADC_Start_DMA(ADC_HandleTypeDef* hadc,  
          uint32_t* pData, uint32_t Length)
```

2

```
uint32_t ADCRsult[100];
```

3

نکته: پس از اینکه ۱۰۰ داده در آرایه ریخته شد و تمام آرایه ها پر شدند، داده ۱۰۱ کجا ریخته شود؟ با

توجه به اینکه در Cube مدل Circulate را انتخاب کردیم(شکل ۵ گزینه ۴) دوباره می چرخد و از اول پر

می کند.

نکته ۲: حال سوال بعدی این است که در زمان چرخش مجدد برای از دست نرفتن داده چکار کرد؟ باید از

طريق اينtrapت ADC متوجه شويم که ۵۰ داده منتقل گردیده است.

۶- طبق روال بطور موقتی Rootin ADC اينtrapت را که مربوط به اتمام تبدیل می باشد را می نويسیم(پس از

تامین خواسته آن را پاک می کنیم) سپس F12 را می زنیم(کد۴) و دستور مربوط به ConvCpltCallback به

(کد۵)، ADC را کپی کرده و بعد از Main ، past می کنیم(کد۵).

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
}
```

نکته: همانطور که در کد ۵ مشاهده می‌کنید اسمی از DMA در این اینترپت وجود ندارد، که دلیل آن کوپل شدن DMA و ADC می‌باشد.

تعريف متغير محلی: اگر یک متغير را در داخل یک تابع تعريف کنیم آن متغير در جاهای دیگر برنامه کاربردی ندارد و تعريف نشده محسوب می‌شود.

۷- ابتدا یک متغير محلی (uint32\_t sum;) داخل تابع اینترپت جهت جمع کردن تمامی درایه‌های آرایه ADCResult و سپس یک متغير سراسری از نوع Float (چون مقدار میانگین می‌تواند عدد اعشاری باشد) جهت میانگین گرفتن از مقدار جمع شده در sum با نام Sensor1 تعريف می‌کنیم. حال از حلقه for میانگین گیری استفاده می‌کنیم و در هر مرحله مقدار آن را در Sensor1 ذخیره می‌کنیم (کد ۷).

توضیح تکمیلی ADC 50: نمونه گیری انجام می‌دهد که طول می‌کشد، این  $50 \times 1.16\mu s = 58\mu s$  نمونه به ترتیب در داخل درایه‌های آرایه ADCResult قرار می‌گیرند. هر بار که تبدیل این ۵۰ نمونه به این طریق انجام شد اینترپت زده می‌شود و در داخل اینترپت از این ۵۰ نمونه جهت از بین بردن نویز و کاهش نوسان تا حد زیاد، میانگین گرفته می‌شود و مقدار آن ذخیره می‌شود.

نکته: زمانی که ADCResult برابر ۵۰ تقسیم می‌شود، چون ۵۰ عددی int می‌باشد حاصل تقسیم نیز int می‌شود. بنابراین برای رفع این مشکل دو راه حل وجود دارد:

الف- بصورت  $50,0$  نوشته شود.

ب- بصورت 50(Float) نوشته شود.

```
float Sensor1;
```

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    uint32_t sum;
```

```
for (int i=0 ; i<50 ; i++)
{
    sum+=ADCRsult [ i ] ;
}
Sensor1=sum/50.0;
```

7

-۸ حال به Debug می رویم، ۵۰ مقدار ADCResult دارای نوسان زیادی بودند (گزینه ۲) ولی پس از میانگین گیری نوسان آن تا حد مطلوبی کاهش می یابد(گزینه ۱)(شکل ۷). در شکل ۸ نیز مشاهده می کنید با توجه به اینکه آرایه ۱۰۰ تایی انتخاب کردیم ولی ۵۰ نمونه از ADC می گیریم درایه های آرایه ADCRsult از ۰ تا ۴۹ مقدار دارند ولی از ۵۰ تا ۱۰۰ صفر می باشند.

Watch 1

Name	Value	Type
sensor1	4036.89...	float
ADCResult	0x20000...	unsigned...
[0]	4036	unsigned...
[1]	4035	unsigned...
[2]	4040	unsigned...
[3]	4041	unsigned...
[4]	4041	unsigned...
[5]	4040	unsigned...
[6]	4038	unsigned...
[7]	4037	unsigned...
[8]	4036	unsigned...
[9]	4044	unsigned...
[10]	4040	unsigned...
[11]	4040	unsigned...
[12]	4036	unsigned...
[13]	4036	unsigned...
[14]	4036	unsigned...
[15]	4035	unsigned...
[16]	4036	unsigned...
[17]	4039	unsigned...
[18]	4036	unsigned...
[19]	4037	unsigned...
[20]	4031	unsigned...
[21]	4037	unsigned...
[22]	4040	unsigned...
[23]	4042	unsigned...
[24]	4040	unsigned...

1

2

Call Stack + Locals Watch 1 Memory 1

(شكل ٧)

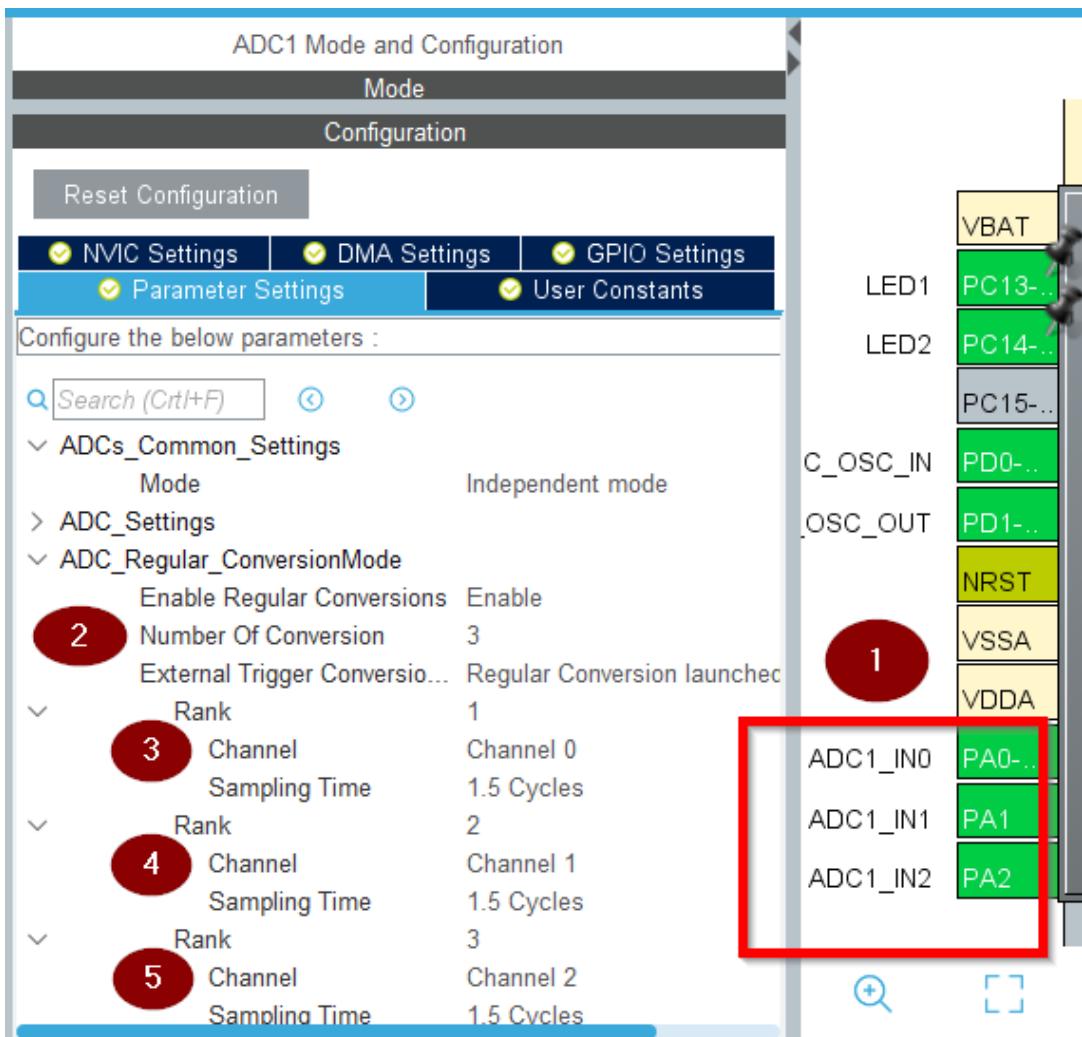
Watch 1

Name	Value	Type
◆ [46]	4036	unsigned...
◆ [47]	4037	unsigned...
◆ [48]	4037	unsigned...
◆ [49]	4036	unsigned...
◆ [50]	0	unsigned...
◆ [51]	0	unsigned...
◆ [52]	0	unsigned...
◆ [53]	0	unsigned...
◆ [54]	0	unsigned...
◆ [55]	0	unsigned...
◆ [56]	0	unsigned...
◆ [57]	0	unsigned int
◆ [58]	0	unsigned...
◆ [59]	0	unsigned...
◆ [60]	0	unsigned...
◆ [61]	0	unsigned...
◆ [62]	0	unsigned...
◆ [63]	0	unsigned...
◆ [64]	0	unsigned...
◆ [65]	0	unsigned...
◆ [66]	0	unsigned...
◆ [67]	0	unsigned...
◆ [68]	0	unsigned...
◆ [69]	0	unsigned...
◆ [70]	0	unsigned...
◆ [71]	0	unsigned...
◆ [72]	0	unsigned...

Call Stack + Locals | Watch 1 | Memory 1

(شکل ۸)

۹- دوباره به Cube برمیگردیم و این بار ۳ پایه IN0,IN1,IN2 (گزینه ۱) را به عنوان ADC انتخاب می‌کنیم و تنها تغییری که انجام می‌دهیم تعداد تبدیل‌ها (Number Of Conversion) را ۳ می‌کنیم (گزینه ۲) و به ترتیبی که می‌خواهیم به Rank‌ها شماره کانال‌ها (گزینه ۳ و ۴ و ۵) را اختصاص می‌دهیم (شکل ۷).



.(شکل ۷)

۱۰- درنهایت پروژه را دوباره Generate می کنیم.

۱۱- در این حالت تعداد درایه های آرایه ما ۳ برابر می شوند (کد ۹ و ۱۰) و تعداد متغیرها نیز ۳ برابر (کد ۱۱ و ۱۲).

HAL\_ADC\_Start\_DMA (&hadc1, ADCRsult, 50\*3);

8

uint32\_t ADCRsult[100\*3];

9

```
float Sensor1;
float Sensor2;
float Sensor3;
```

10

uint32\_t sum1; uint32\_t sum2; uint32\_t sum3;

11

۱۲-جدول (شکل۸) نشان دهنده نحوه چیدمان مقادیر خوانده شده از کانال های ADC و در مقابل شماره درایه های آرایه که مقادیر ADC در آن ریخته می شود، مشاهده می کنید که شماره هر کانال (مانند IN0 ) ۳ درایه یکبار تکرار می شود بنابراین در حلقه for نیز مقدار متغیر آن باید به صورت  $i=+3$  در بیايد.

<u>ADCResult</u>	<u>Chanel</u>
ADCResult[0]	IN0
ADCResult[1]	IN1
ADCResult[2]	IN2
ADCResult[3]	IN0
ADCResult[4]	IN1
ADCResult[5]	IN2
ADCResult[6]	IN0
ADCResult[7]	IN1
ADCResult[8]	IN2
.	.
.	.
.	.
.	.
.	.
.	.
.	.
ADCResult[147]	IN0
ADCResult[148]	IN1
ADCResult[149]	IN2

(شکل۸)

۱۳-حال با در نظر گرفتن تمام موارد اشاره شده اینترابت را می نویسیم.

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    uint32_t sum1; uint32_t sum2; uint32_t sum3;

    for(int i=0 ; i<50*3 ; i+=3)
    {
        sum1+=ADCRsult[i];
        sum2+=ADCRsult[i+1];
        sum3+=ADCRsult[i+2];
    }
}
```

```

{
    Sensor1=sum1/50.0;
    Sensor2=sum2/50.0;
    Sensor3=sum3/50.0;
}

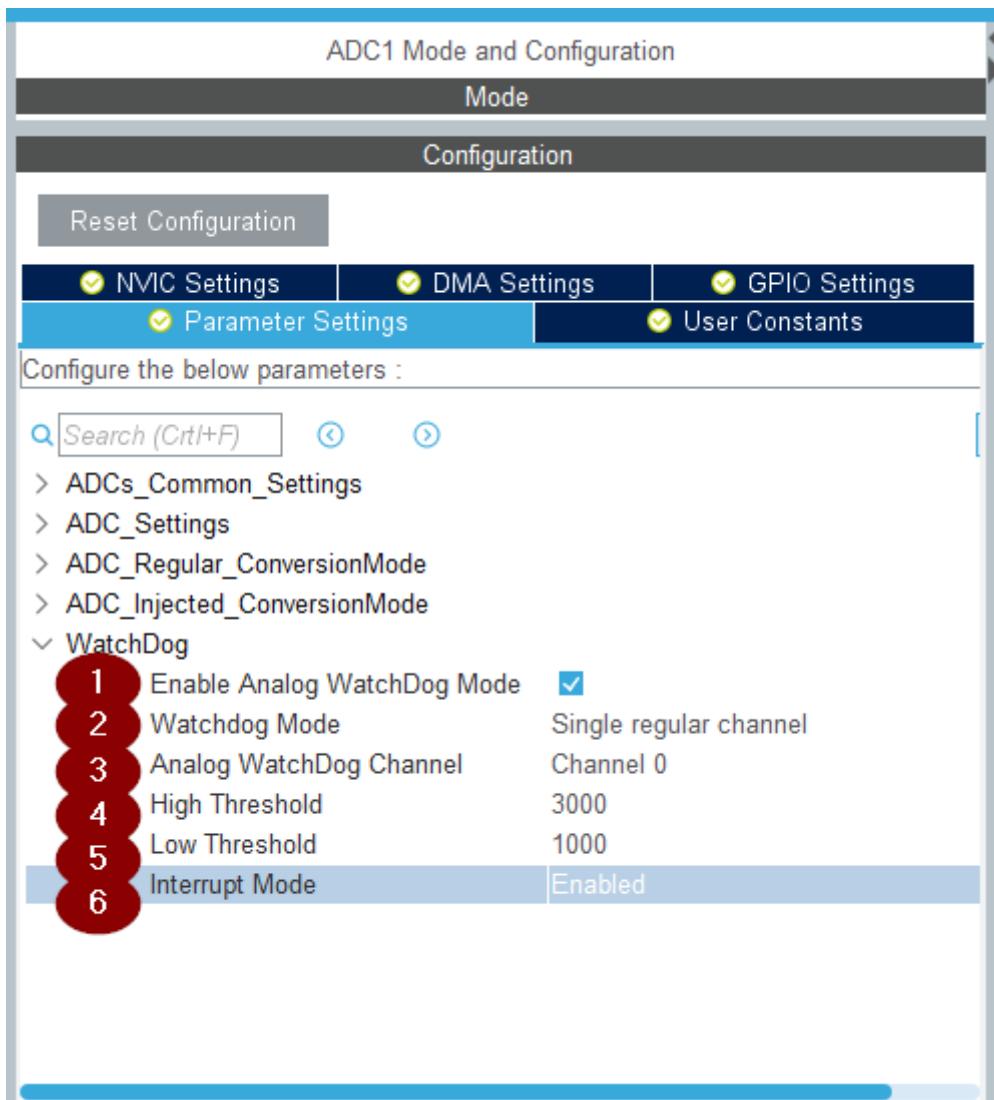
```

12

مثال: حال می خواهیم برنامه ای بنویسیم که در صورت گذشتن از آستانه مشخص اینترپت بدهد(مثلا ۱۰۰۰ و ۳۰۰۰).

برای این کار از یک قابلیت میکروکنترلر به نام Watchdog استفاده می کنیم.

(شکل ۹): این حالت، در اصل یک مقایسه گر است. با فعال سازی Enable Analog Watchdog Mode - ۱۴ این گزینه، در صورتیکه مقدار خوانده شده از ADC خارج از بازه ای مابین حد تحتانی (Low Threshold) و حد فوقانی (High Threshold) باشد، میکروکنترلر بصورت خودکار، وقفه ای را فعال می کند که می توان از آن استفاده نمود. در ذیل تنظیمات مربوطه، Watchdog Mode تعیین کننده نوع (Regular) یا Injected و تعداد کانال هایی است که می خواهیم Watchdog بر روی آنها نظارت داشته باشد. البته تنها دو انتخاب داریم؛ یکی فقط نظارت بر یک کانال (Single regular channel) و دیگری تمامی کانال های Analog watchdog Channel گزینه ADC (All regular channels). می کند. وقت شود که مقادیر Low threshold و High threshold باشد بصورت دیجیتالی وارد شوند؛ یعنی مقدار معادل ولتاژ آنالوگ مورد نظر در بازه ۰ تا ۴۰۹۵ در نهایت، گزینه Interrupt Mode جهت فعال سازی وقفه مربوط به Watchdog است، که در صورت Enable شدن، با اجرا شدن وقفه (خروج از محدوده معین شده)، پردازنده به وقفه سرویس دهی می کند. حال مطابق شکل ۹ تنظیمات را انجام می دهیم.



(شکل ۹)

۱۵- درنهایت پروژه را دوباره Generate می کنیم.

۱۶- طبق روال بطور موقتی Rootin اینترپت Watchdog را می نویسیم (پس از تامین خواسته آن را پاک می کنیم) سپس F12 را می زنیم (کد ۱۳) و دستور مربوط به LevelOutOfWindowCallback (کد ۱۴)، را کپی کرده و بعد از Main ، past می کنیم (کد ۱۵) و با توجه به نیاز داخل تابع LevelOutOfWindowCallback را پر می کنیم.

HAL\_ADC\_LevelOutOfWindowCallback

13

```
void HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef* hadc)
{
```

```

    ADCResult = HAL_ADC_GetValue(&hadc1);
}

```

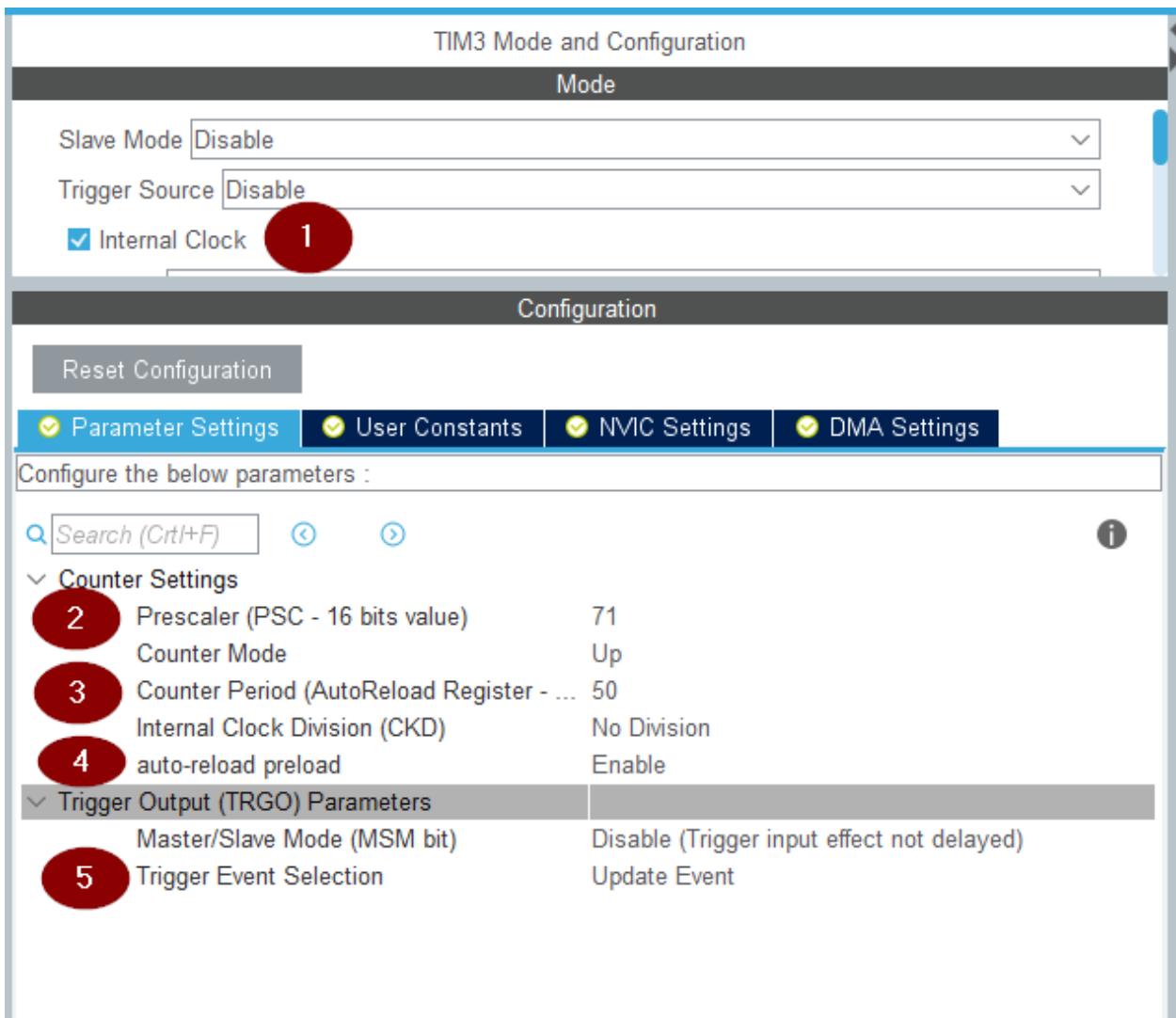
14

حال می خواهیم برنامه را طوری بنویسیم که هر 50us یک نمونه آماده به ما بدهد.

- ابتدا تایمر<sup>۳</sup> را در حالت Time Base (گزینه ۱) و با زمان آپدیت یا سرریز 50us فعال می کنیم. حال می خواهیم تایمر هر 50us ، ADC را Trig کند به همین دلیل Trig Event را روی حالت Update تنظیم کنیم تا هر وقت آپدیت یا سرریز شد Trig بدهد(گزینه ۵). در اینجا دیگر نیازی به فعال کردن ایترابت تایمر نداریم چون نمی خواهیم در هر آپدیت کار دیگری انجام دهیم، تنها Trig کردن است که به طور اتوماتیک انجام می شود(شکل ۱۰).

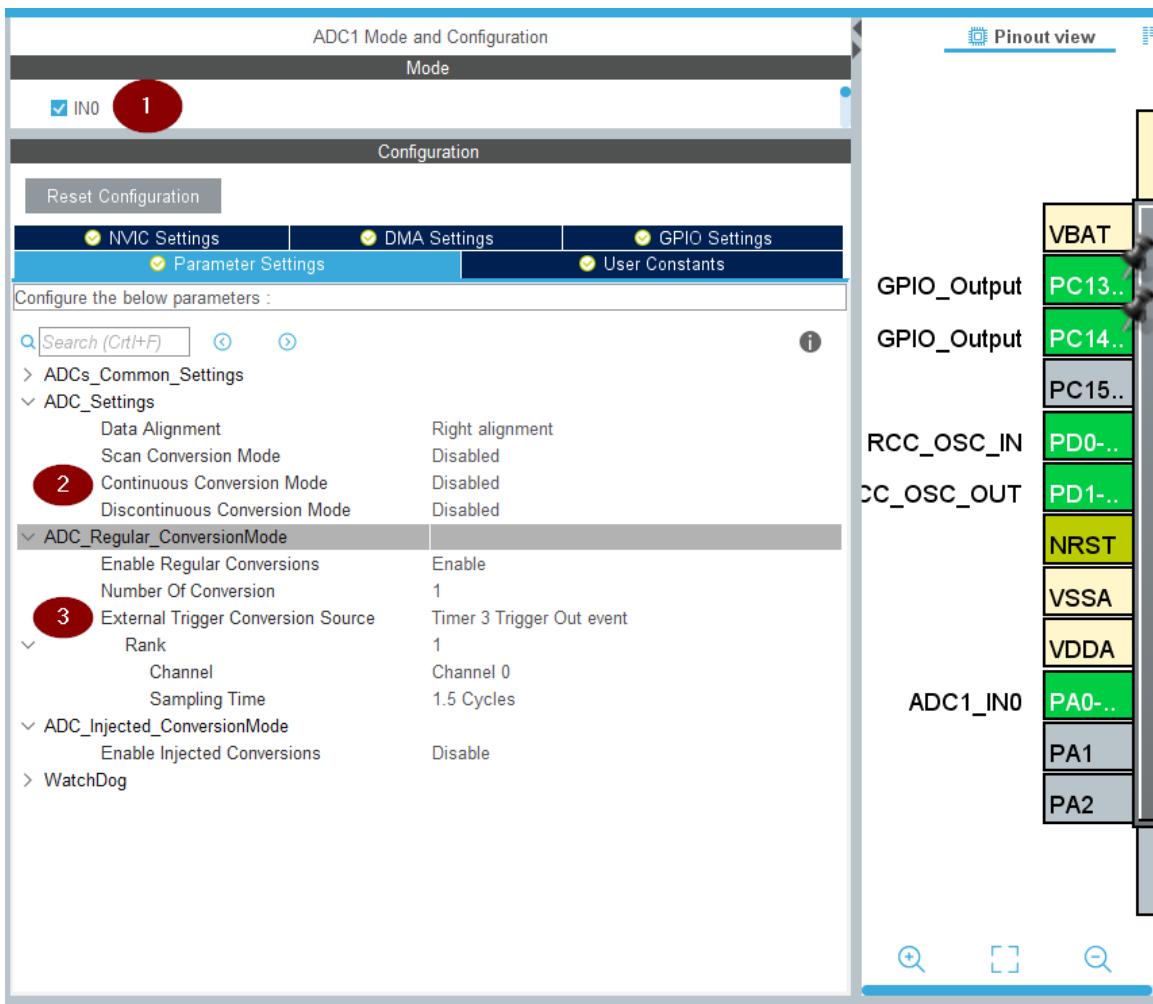
$$F_{CK-CNT} = \frac{F_M}{PSC + 1} = \frac{72M}{71 + 1} = 1M$$

$$T_{ovf} = ARR \times \frac{1}{F_{CK-CNT}} \Rightarrow 50\mu s = ARR \times \frac{1}{1M} \Rightarrow ARR = 50$$



(شکل ۱۰)

-۲ کanal ADC in0 را فعال می کنیم (گزینه ۱) و تنظیمات همان قبلی می باشد با این تفاوت که را غیرفعال می کنیم (گزینه ۲)، چون می خواهیم هر زمانی که تایمر گرد نمونه گیری انجام شود. حال باید در قسمت External Trig Conversion Trig مشخص کنیم چه چیزی قرار است ADC را Trig کند بنابراین آن را بر روی Trig event مربوط به تایمر ۳ (گزینه ۳) تنظیم می کنیم (شکل ۱۱).



(شکل ۱۱)

۳- پروژه را Generate می کنیم تا Keil باز شود.

۴- در این حالت دو تغییر انجام می شود:

الف- در این حالت تایمر بصورت اتوماتیک فقط هر  $50\text{us}$  ADC را جهت نمونه گیری Trig می کند.

ب- در اینجا دیگر نمی توان  $50$  نمونه گرفت و سپس میانگین گرفت، چون همانطور که در مثال قبل

مشاهده شد زمان لازم برای  $50$  نمونه معادل  $1.16\text{us} \times 50 = 58\text{us}$  است، بنابراین تعداد نمونه

$$30 \times 1.16\text{us} = 34.8\text{us}$$

```
uint32_t ADCValue[90];
```

```
HAL_ADC_Start_DMA(&hadc1, ADCValue, 30);
```

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
```

```

{
    uint32_t sum=0;
    for(int i=0 ; i<30 ; i++)
    {
        sum+=ADCValue[i];
    }
    Sensor=sum/30.0;
}

```

سوال: در اینجا سیگنال با چه فرکانسی قابل نمونه برداری است؟

باتوجه به نکته مهمی که در ابتدای فصل گفته شد (فرکانس سیگنالی که قرار است از آن نمونه برداری شود باید حداقل نصف فرکانس نمونه برداری باشد) باتوجه به اینکه فرکانس نمونه برداری ما  $(F_{sampling} = \frac{1}{50\mu s})$  است بنابراین سیگنال هایی با فرکانس کمتر از  $f_{signal} \leq \frac{F_{sampling}}{2} = 10KHz$  قابلیت نمونه برداری را دارند.

## **۹ فصل**

### **پروتکل ارتباطی سریال**

**Usart**

## ۱-۹ پروتکل ارتباطی سریال

غالب پروتکل‌های ارتباطی میکروکنترلرهای ARM از نوع سریال (بیت به بیت) است. دو ارتباط معروف Recieve<sup>۱</sup> و UART نیز از این نوع بوده و در آن هر دستگاه (گیرنده – فرستنده) دارای دو پایه عموماً با نام RX (Receive) و TX (Transmit) با نامد است. در ارتباط UART تنها به همین دو سیم برای انتقال داده نیازمندیم. یک دستگاه به TX دیگر دستگاه متصل می‌گردد و بالعکس که بصورت آسنکرون این ارتباط صورت می‌پذیرد. شایان ذکر است که زمین (GND) دو دستگاه نیز باید مشترک باشد. البته در ارتباط USART به یک سیم سوم کلاک (CLK) نیز جهت سنکرون کردن دو دستگاه نیازمندیم. ارتباطات USART/UART از نوع دوطرفه (Full-Duplex) است، یعنی هر دستگاه می‌تواند بطور همزمان داده را از سال کرده و دریافت کند (زیرا دو خط ارتباطی داریم). در میکروکنترلرهای ARM بسته به نوع، چندین واحد USART وجود دارد. مهم‌ترین تنظیمات در ارتباط USART تنظیم سرعت می‌باشد که به آن Baudrate می‌باشد که سرعت انتقال اطلاعات را بر حسب Bit/s مشخص می‌کند که می‌تواند تا ۴۰۵۶ بت بالا برسد که مقدار بسیار بالایی می‌باشد. تعداد یتهایی که در هر بار ارسال می‌شوند ۸ یا ۹ بت می‌باشد، که معمولاً ما به صورت ۸ بیت ارسال می‌کنیم.

-۱ مربوط به آخرین مثال ADC را باز می‌کنیم. و تنظیمات جدید را به آن اضافه می‌کنیم (شکل ۱۶):  
-۲ در قسمت Connectivity ، USART1 (گزینه ۱) و مد کاری را Asynchronous (گزینه ۲) را انتخاب می‌کنیم. همانطور که در شکل مشاهده می‌کنید پورت‌های RX و TX در قسمت PinOut نشان داده شده است. گزینه ۳ RS232 می‌باشد که یک پروتکل ارتباطی کاملاً شبیه USART است با این تفاوت که سطوح ولتاژها مقداری بالاتر می‌باشند و همچنین نویز و خطای کمتر و در فواصل دورتر قابل استفاده می‌باشد که این ویژگی‌ها باعث شده بیشتر در صنعت مورد استفاده قرار گیرند (گزینه ۳).

-۳ پارامترهای شاخص در ارتباط با تنظیمات USART/UART در Cube عبارت اند از:  
الف- Baud rate : سرعت تبادل داده (نرخ انتقال بیت‌ها) را بر حسب bits/sec (bps) مشخص می‌کند و لزوماً باید برای هر دو دستگاه یکسان تنظیم شود. در بیشتر دستگاه‌ها و ماژول‌های مجهر به UART مقادیر از پیش تعیین شده‌ای برای سرعت انتقال قابل انتخاب است (که مضارب ۱۲۰۰ bps هستند) و معروف‌ترین آنها ۹۶۰۰ bps است. اما یکی از مزایای میکروکنترلرهای ARM قابلیت انتخاب مقدار صحیح دلخواه برای baud rate است، که اصطلاحاً به آن Fractional Baud Rate می‌باشد.

<sup>۱</sup> Universal Synchronous Asynchronous Receiver Transmitter

می‌گویند. بافرض اینکه میخواهیم مقدار ولتاژ ADC را ارسال کنیم می‌دانیم که اطلاعات ADC 12 بیتی (۴۰۹۵) هستند ولی دراینجا به صورت ۸ بیتی (۲۵۵) ارسال می‌شود، بنابراین باعث می‌شود دقت ارسال داده‌ها را به ۸ بیت کاهش دهیم. در ADC هر 50us یک تبدیل صورت می‌پذیرد، یعنی در هر ثانیه ۲۰۰۰۰ هزار داده خواهیم داشت، با توجه به اینکه هر داده ۸ بیت می‌باشد، بنابراین باید ۱۶۰۰۰۰ بیت در هر ثانیه ارسال گردد. ولی با درنظر مقدار parity و سایر موارد مقدار آن را برای اطمینان چند برابر می‌کنیم مثلاً ۵۰۰۰۰۰ (گزینه ۴).

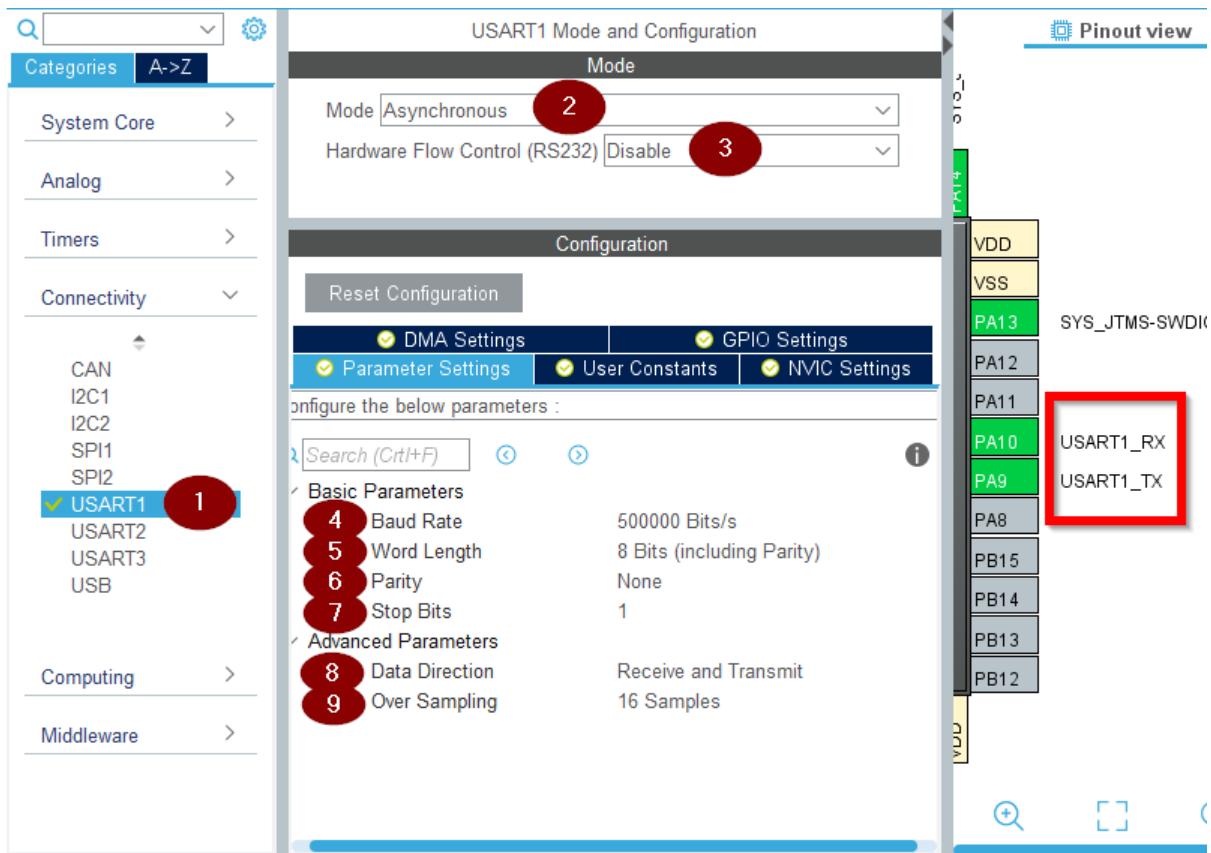
ب - Word Length : طول داده را تعیین می‌کند که می‌تواند ۸ بیتی یا ۹ بیتی باشد (گزینه ۵).

پ - Parity : عامل مشخص کننده وجود یا عدم وجود خطأ در داده‌های دریافتی در گیرنده است، که می‌تواند غیرفعال (none) زوج (even) و یا فرد (odd) باشد. دقت شود که به کمک parity تنها قادر به تشخیص خطأ بوده و نمی‌توانیم آنرا اصلاح کنیم (گزینه ۶).

ت - Stop bit : تعداد بیت‌های پایانی هر بسته (frame) اطلاعات را مشخص می‌کند که می‌تواند ۱ یا ۲ بیت باشد و همواره برابر با ۱ منطقی است. در غالب کاربردها، از ۱ بیت پایانی استفاده می‌کنیم (گزینه ۷).

ث - Data Direction : جهت ارسال داده را مشخص می‌کند، که بسته به کاربرد می‌تواند در حالات Receive and Transmit Only یا Receive and Transmit Over Sampling تنظیم گردد (گزینه ۸).

ح - Over Sampling : مقدار آنرا بر روی ۱۶ Samples تنظیم می‌کنیم (گزینه ۹).



شکل ۱۶

-۴ پروژه را Generate می‌کنیم تا Keil باز شود.

دراینجایک مدل USB serial به نام PL2303 برای انتقال دیتا بین میکروکنترلر و PC نیاز است، ابتدا یک مثال ساده را انجام می‌دهیم . می‌خواهیم یک رشته اطلاعات را هر 500ms ارسال کنیم مانند (Salam!)

-۵ با توجه به اینکه می‌خواهیم دائمًا این فرایند انجام شود آن را در While(1) می‌نویسیم. حال دستور ارسال را نوشته (کد ۱) سپس طبق روال F12 را می‌زنیم تا به تعریف تابع (کد ۲) برویم و از نوع ورودی‌های آن مطلع شویم.

HAL\_UART\_Transmit

کد ۱

```
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef
*huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

کد ۲

نکته: هر کاراکتر یک عدد ۸ بیتی است. بطور مثال حرف S یک کاراکتر پس ۸ بیت می‌باشد.

نکته: char : معادل int ، ۳۲ بیتی است.

نکته: برای تبدیل یک کاراکتر که معادل int\_8 می‌باشد به uint8 آن را cast می‌کنیم به عبارتی قبل از پارامتری که می‌خواهیم آن را cast کنیم (int8\_t\*) را اضافه می‌کنیم.

۶- همانطور که در کد ۲ رامشاهده می‌کنیم: ورودی اول Handeler تابع و دومی Pointer است یعنی آدرس داده‌ای که قرار است آن را ارسال کنیم یا خود رشته داده‌ای که قصد ارسال آن را داریم و سومی تعداد بایت‌هایی است که قرار است با یک دستور ارسال شوند و سومی Timeout می‌باشد، در صورتیکه به هر دلیلی داده ارسال نشد پس از گذشت Timeout لحظه شده از آن خط عبور می‌کند.

```
while (1)
{
    HAL_UART_Transmit(&huart1, (uint8_t*)"Salam!\n\r", 8, 100);
    HAL_Delay(500);
}
```

کد ۳

۷- برای نمایش اطلاعات ارسال شده به PC نرم‌افزارهای زیادی (با عنوان Terminal) از جمله Matlab و Naminic Hyper Terminal و Hercules است که ابتدا از Hyper Terminal استفاده می‌نماییم.

نکته: یکی از مشکلات Hercules این است که حداقل تا ۲۵۶۰۰۰ Bps را ساپورت می‌کند.

نکته: مفهوم \n\r به معنای رفتن به خط بعدی می‌باشد.

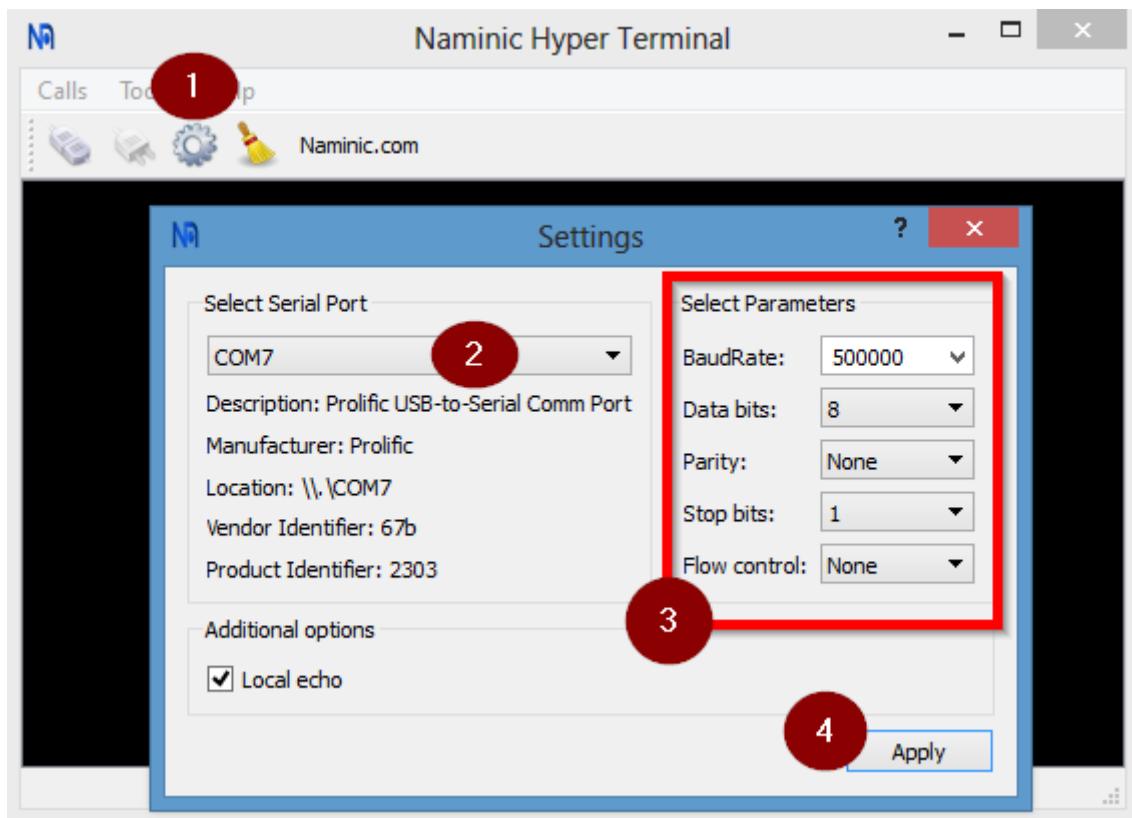
۸- در ابتدا تنظیمات پورت سریال (گزینه ۱) را در Hyper Terminal انجام می‌دهیم (شکل ۱۷).

الف- با اتصال USB به PC، برنامه پورت را شناسایی می‌کند (گزینه ۲). مژول در قسمت Device Management قابل رویت می‌باشد.

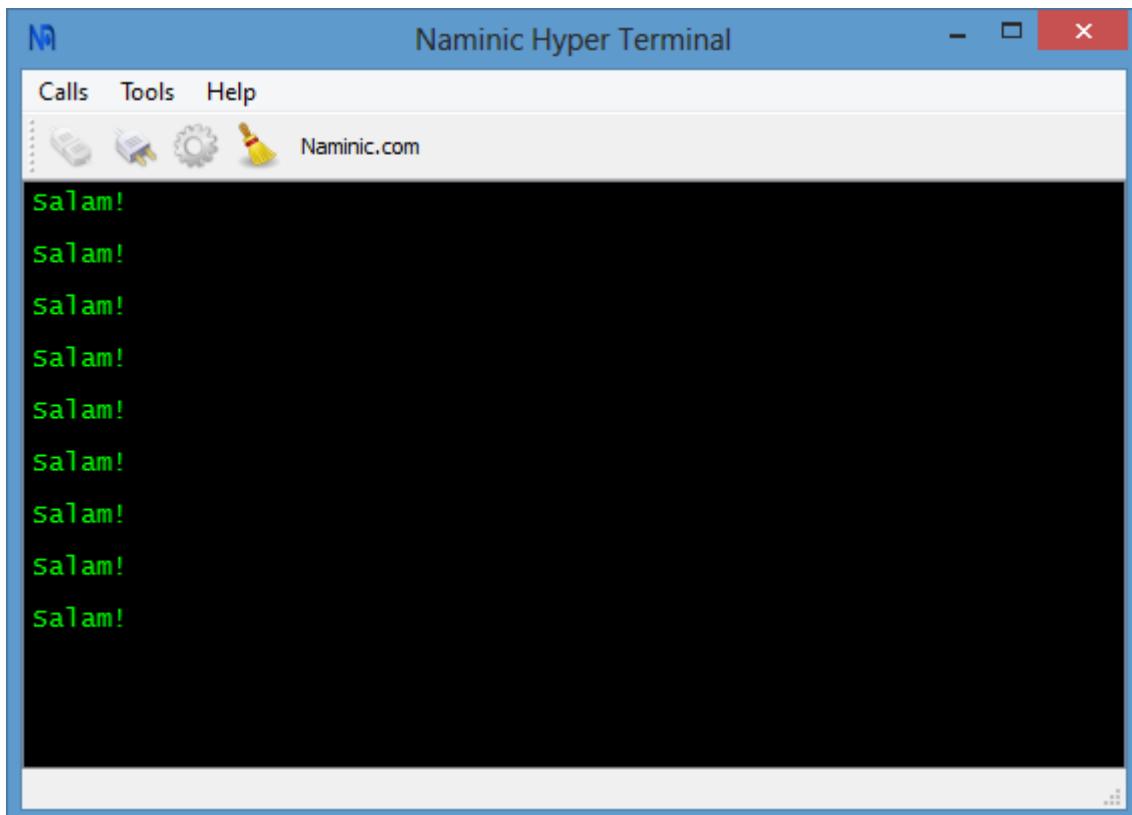
ب- مقدار پارامترهای خواسته شده در گزینه ۳ مطابق هر آنچه در Cube تنظیم نمودیم، در اینجا نیز وارد می‌کنیم (گزینه ۳).

۹- در پایان برنامه را اجرا می‌کنیم (شکل ۱۸).

۱۰- مشاهده می کنیم پس از هر ۵۰۰ms پیام ارسال می شود(شکل ۱۸).



شکل ۱۷



## ۱۸ شکل

حال می خواهیم مقادیر Sensor در آخرین مثال ADC که با Trig تایمر همراه بود را نمایش می دهیم.

- ۱- در اینجا ورودی دوم را همانگونه که از پیش گفته به جای Pointer از آرایه استفاده می کنیم، بنابراین با توجه به اینکه سایز هر کدام از داده های ارسالی باید ۸ بیتی باشد و همچنین با اطلاع از این قضیه داده ها ۸ بیتی تنظیم گردیده لذا در تعریف آرایه نیز سایز هر آرایه را ۸ بیت تعریف می کنیم )  
با توجه به اینکه در هر ارسال یک مقدار از Sensor ارسال می گردد مقدار یک برای آرایهها uint8\_t).  
کافی می باشد اما محض اطمینان آن را ۱۰ می گذاریم (کد ۴).

```
uint8_t Tdata[10];
```

کد ۴

- ۲- تغییر دیگری که نیاز است انجام دهیم مربوط به مقدار سنسور است که باید بر ۱۶ تقسیم گردد چون اطلاعات اولیه براساس ۱۲ بیت بوده ولی در اینجا مجبوریم اطلاعات را جهت ارسال ۸ بیتی کنیم.  
ونکته دیگر اینکه در هر ارسال [0] Tdata را ارسال می کنیم (کد ۵).

$$\frac{2^{12}}{2^8} = 2^4 = 16$$

```
Tdata[0]=Sensor/16;
```

کد ۵

- ۳- در نهایت دستور ارسال اطلاعات را در انتهای تابع ConvCpltCallback مینویسیم (کد ۶).

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    uint32_t sum=0;
    for(int i=0 ; i<30 ; i++)
    {
        sum+=ADCValue[i];
    }
    Sensor=sum/30.0;

    Tdata[0]=Sensor/16;
    HAL_UART_Transmit(&huart1,Tdata,1,10);
}
```

کد ۶

- ۴- در اینجا پایه PWM با عرض پالس ۴۰ درصد به ورودی سنسور وصل می‌کنیم.
- ۵- برای دیدن داده‌ها از نرم‌افزار مطلب استفاده می‌کنیم.
- ۶- دستور ایجاد پورت سریال بدین صورت می‌باشد: ورودی اول معرف پورت سریال ماست و دومی مشخص کننده سرعت Bufer و سومی BudRte می‌باشد که تعداد نمایش‌های ورودی را مشخص می‌کند (بطور مثال با توجه به اینکه هر 20us یک ورودی دریافت می‌کنیم، اگر بخواهیم داده‌های ۶۰ ثانیه را مشاهده کنیم باید مقدار آن را ۱۲۰۰۰۰۰ قرار دهیم) مابقی اطلاعاتی که در Cube تنظیم کردیم بطور پیش‌فرض همان اطلاعات در اینجا تنظیم شده است.

```
s =
serial('COM6','baudrate',500000,'InputBufferSize',1000000)
';
کد ۷
```

۷- با کد ۷

```
fopen(s)
```

پورت را باز می‌کنیم و شروع به دریافت داده می‌کند (با fclose(s) نیز بسته می‌شود).

۸- با نوشتن s و زدن Enter اطلاعات دریافت شده اطلاعات نشان داده می‌شود (شکل\_).

با استفاده از کد ۸ داده‌ها خوانده می‌شود و داخل یک متغیر ریخته می‌شود، همچنین ورودی دوم آن مشخص کننده تعداد ورودی‌هایی می‌باشد که می‌خواهیم ببینیم.

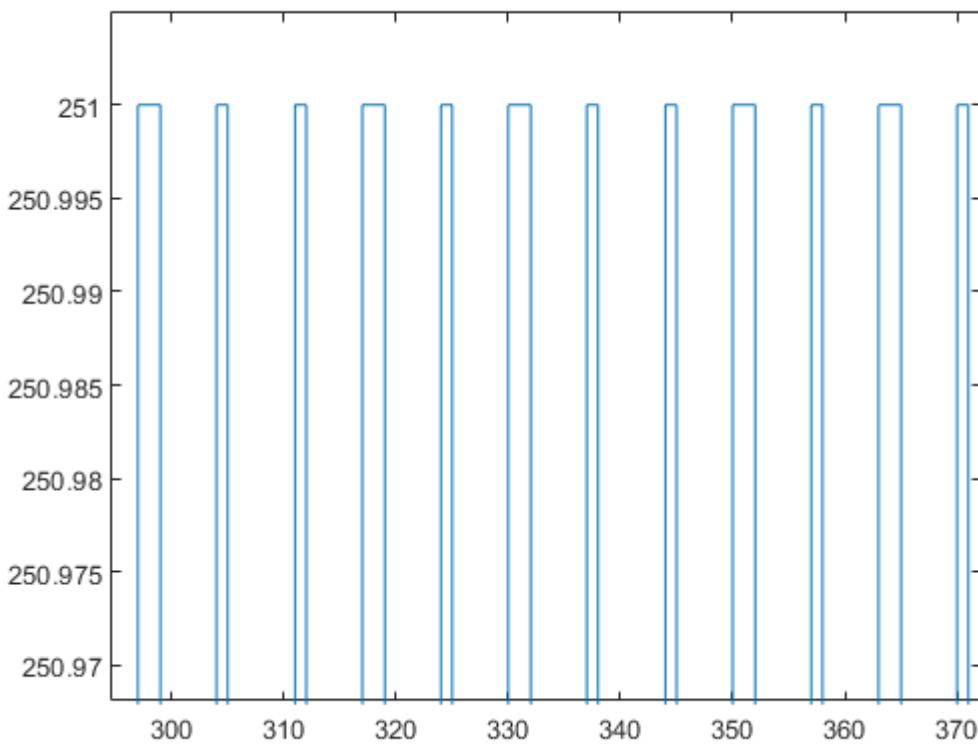
```
Data=fread(s,10000);
```

کد ۸

برای نمایش داده‌ها نیز از کد ۹ استفاده می‌شود (شکل ۱۹):

```
Plot(Data)
```

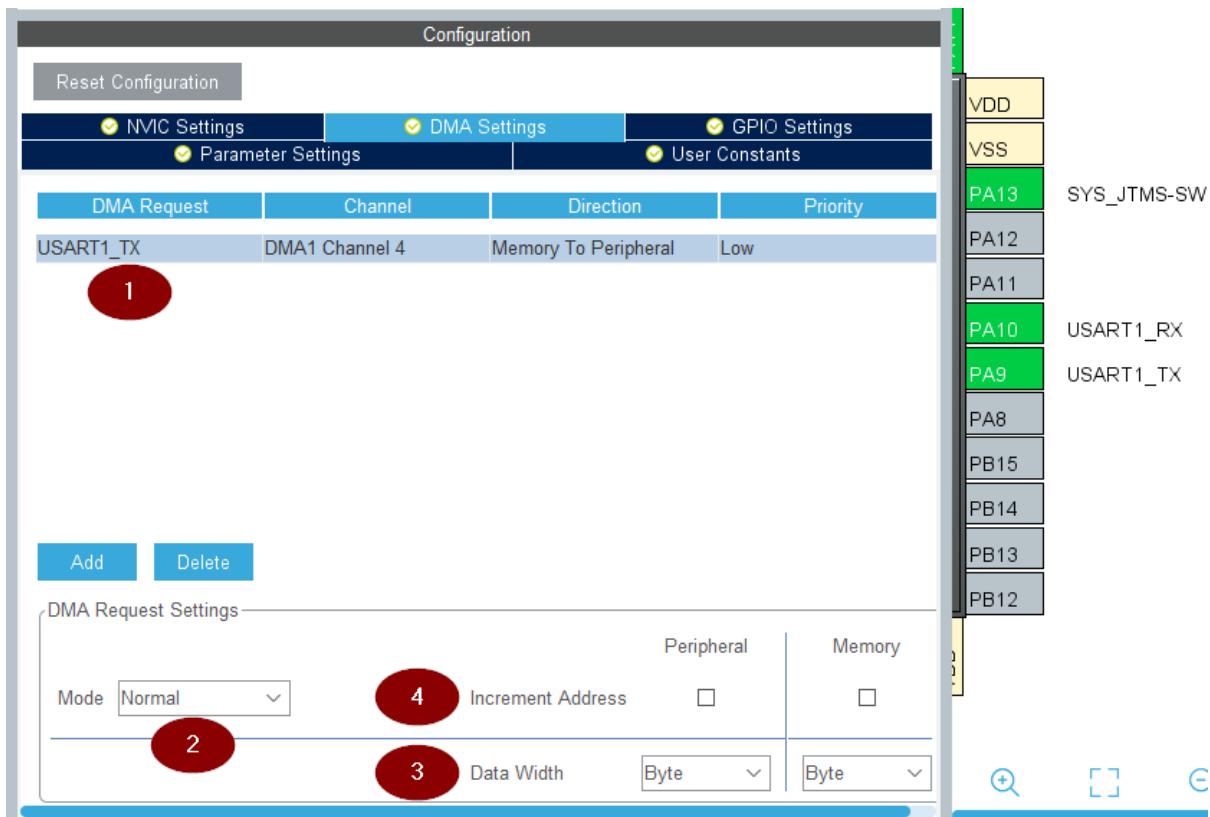
کد ۹



(شکل ۲۰)

حال می خواهیم اطلاعات را از طریق DMA دریافت کنیم. بطوریکه ما یک DMA می دهیم و اطلاعات سنسور را برمی دارد و داخل USART DMA می ریزد.

- ۱- تنظیمات قبلی Cube را حفظ و مقداری تغییرات را روی آن اعمال می کنیم.
- ۲- در اینجا باید DMA مربوط به حالت ارسالی یعنی TX را فعال کنیم (شکل ۲۱ گزینه ۱) و مدار کاری آن را Normal انتخاب نماییم (گزینه ۲) چون اگر Circular انتخاب کنیم یک داده چندین بار ارسال می نماید و با توجه به ۸ بیتی بودن داده های ارسالی در USART ، DataWidth را Byte انتخاب می کنیم. همچنین هیچ کدام موارد Increment Address را بر خلاف تنظیمات DMA مربوط به ADC تیک نمی زنیم چون می خواهیم تنها داده ذخیره شده ( $Tdata[0]$ ) را ارسال کنیم و اولاً تنها این خانه مقدار دارد و دوماً اجازه دسترسی به باقی درایه ها وجود ندارد.



(شکل ۲۱)

۳- پروژه را Generate میکنیم تا Keil باز شود.

۴- در حالت قبلی وقتی از کد ۱۰ جهت ارسال اطلاعات استفاده می شد تا زمانی که اطلاعات ارسال نمی شد میکروکنترلر از آن خط عبور نمی کرد و به همین دلیل از TimeOut استفاده می کردیم تا در صورت بروز مشکل بصورت خودکار پس اتمام زمان TimeOut به خط بعدی برود. ولی در اینجا به دلیل استفاده از DMA ، میکرو داده ها جهت ارسال یا دریافت در اختیار DMA می گذارد و خودش بقیه کارها را انجام می دهد.(کد)

```
HAL_UART_Transmit(&huart1, Tdata, 1, 10);
```

کد ۱۰

```
HAL_UART_Transmit(&huart1, Tdata, 1);
```

کد ۱۱

۵- در نهایت با اعمال این تغییر برنامه بصورت کد(۱۴) می شود همچنین متغیرها و فراخونی Priepheral نیز بدین صورت می باشد(کد ۱۲ و ۱۳).

```
uint32_t ADCValue[90];  
uint8_t Tdata[10];  
float Sensor;
```

١٢٥

```
HAL_ADC_Start_DMA(&hadc1, ADCValue, 30);  
HAL_TIM_Base_Start(&htim3);
```

١٣٥

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)  
{  
    uint32_t sum=0;  
    for(int i=0 ; i<30 ; i++)  
    {  
        sum+=ADCValue[i];  
    }  
    Sensor=sum/30.0;  
  
    Tdata[0]=Sensor/16;  
    HAL_UART_Transmit_DMA(&huart1, Tdata, 1);  
}
```

١٤٥

فصل دهم

مبدل دیجیتال به آنالوگ

Digital to Analog Converter

(DAC)

## ۱-۱ مبدل دیجیتال به آنالوگ

میکروکنترلرهای ARM دارای چند واحد تبدیل دیجیتال به آنالوگ (DAC) بوده و دقت تبدیل آنها در اکثر میکروکنترلرهای ARM (از جمله STM32 ) برابر با ۱۲ بیت است. به عبارت دیگر، مقادیر صحیح مابین ۰ و ۴۰۹۵ را به مقادیر آنالوگ بین ۰ تا VREF+ تبدیل کرده و بر روی پایه های معینی قرار می دهد. از عمدۀ کاربردهای DAC می توان به تولید صوت آنالوگ اشاره کرد.

واحد DAC در میکروکنترلرهای ARM دارای قابلیت تولید موج های مثلثی (Trianglex) و همچنین نویز سفید (LFSRx) با دامنه قابل تنظیم هستند. عموماً از نویز سفید به منظور شناسایی (تابع تبدیل) سیستم های خطی استفاده می شود.

برای کار با DAC کافیست در برنامه، مقدار مورد نظر برای تبدیل را در رجیستر DHRx (Data Hold Register) وارد کنیم. پس از تبدیل (که ۱ تا ۳ سیکل کلک به طول می انجامد)، ولتاژ آنالوگ در رجیستر DORx (Data Output Register) ذخیره می گردد. این ولتاژ پس از عبور از واحد مبدل DAC بر روی پایه (با قابلیت جریان دهنده) قابل دسترسی است.

در اینجا یک اسکوپ طراحی می کنیم بطوریکه یک موج PWM و موج مثلثی که تو سط DAC می سازیم را در متلب نمایش دهیم.

با توجه به اینکه همه میکروکنترلها داری PORT ، DAC نیستند از میکروکنترل STM32F107VCTX استفاده می کنیم .

۱- تنظیمات اولیه Cube را انجام می دهیم سپس موارد زیر را انتخاب می کنیم:

الف ADC1\_IN8 - ، را به همراه ایترپت و DMA فعال می کنیم. مانند حالت قبلی از Trig تایمر ۳ استفاده می کنیم بطوریکه هر 50us یک تبدیل را انجام دهد.(PSC=1,ARR=1800)

ب TIM4 - در حالت PWM Generation در Chanel3 با فرکانس 100Hz و در این صورت PSC=7199 و ARR=100 .

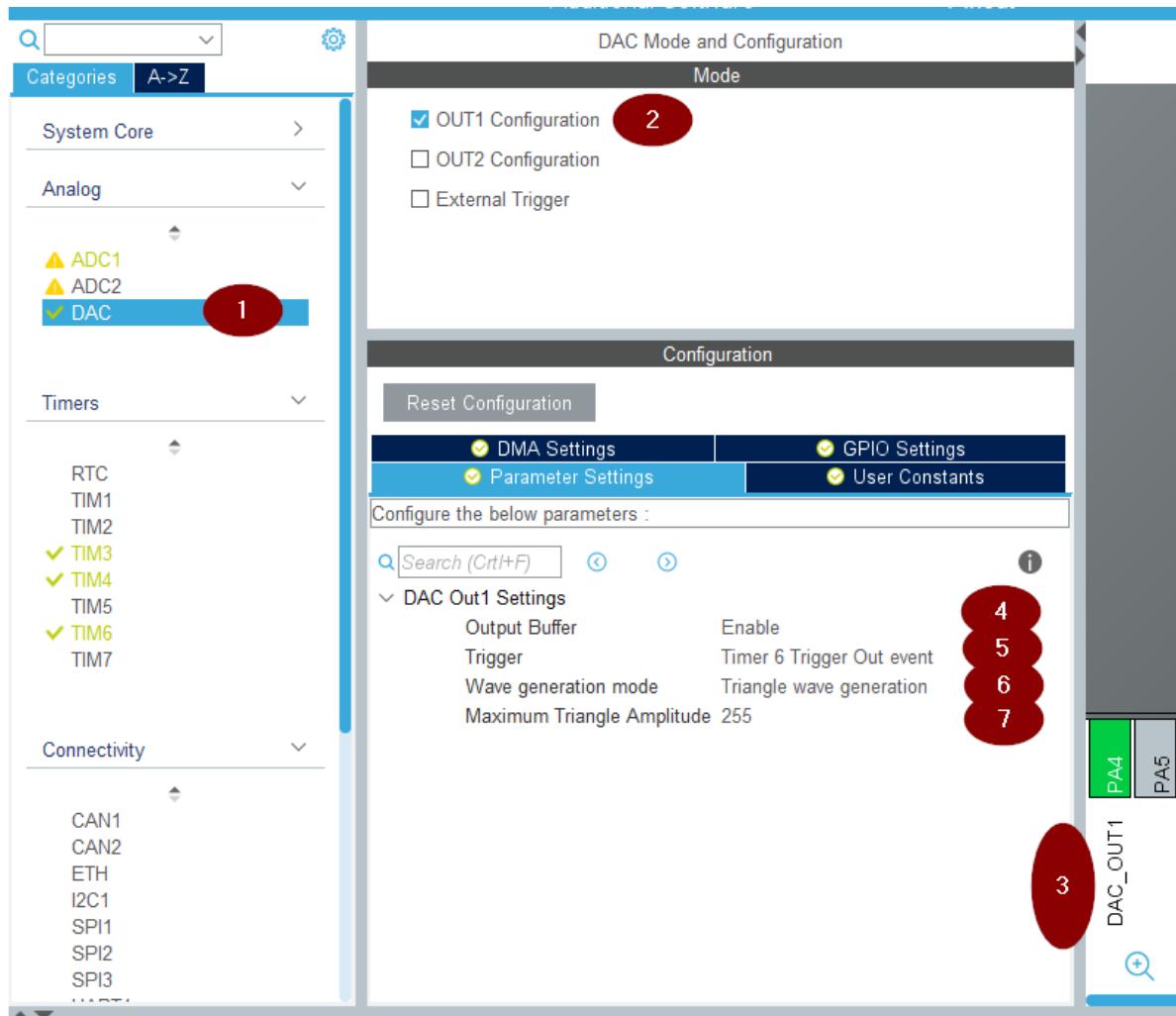
پ USART2\_RX و USART2\_TX را فعال می کنیم، مهمترین خاصیت اسکوپ، فرکانس یا به عبارتی سرعت آن است. در اینجا مطابق با توضیحات پیشین Budrate را ۱۰۰۰۰۰ می گذاریم. و اینترپت آن را نیز فعال می کنیم.

ت- برای تنظیم DAC داریم (شکل ۱۸) :

گزینه ۴ Output Buffer: بافر خروجی DAC است که در صورت فعال سازی آن، قابلیت جریان دهی به خروجی مهیا خواهد شد. از آنجاییکه در غالب کاربردهای آنالوگ، نیاز به جریان داریم، همواره این گزینه را بر روی Enabled تنظیم می کنیم.

گزینه ۵ Trigger: بطور کلی دو حالت برای تحریک مبدل دیجیتال به آنالوگ وجود دارد. یک حالت به کمک گُدنویسی (Software trigger) است. در این حالت، کاربر در زمان دلخواه، داده دیجیتال را در رجیستر DHRx وارد می کند تا تبدیل انجام گردد. اما حالت دیگر، استفاده از یک منبع تحریک خارجی (External Trigger) است. این منابع، تایمراها هستند که با انتخاب هریک از گزینه های Timer x Trigger Out event (Trigger) می توان تعیین نمود که کدام تایمر در دوره های متناوب (با فاصله زمانی قابل تنظیم) واحد مبدل دیجیتال به آنالوگ را تحریک کند. در صورت انتخاب تایمر بعنوان منبع تحریک خارجی، باید در تنظیمات سربرگ Trigger Configuration) تایمر مربوطه و تحت مجموعه Trigger Output (TRGO) Parameters را بر روی Event Selection تنظیم نمود. در این شرایط، ابتدا داده دیجیتال مورد نظر را در رجیستر DHRx وارد می کنیم و میکروکنترلر بطور خودکار با هر بار سرریز شدن تایمر مربوطه، یک پالس تحریک برای DAC صادر می کند تا داده را به آنالوگ تبدیل کرده و به پایه خروجی منتقل کند. با فعال سازی Trigger ، تنظیمات دیگری در زیر آن پدیدار می گردد که شامل Wave Generation Mode و Trig می باشد. مورد اول بیانگر نوع موج تولیدی توسط واحد DAC است و مورد دوم تعیین Maximum Amplitude است. کننده دامنه موج تولیدی (تا ۴۰۹۵) است.

بنابراین در اینجا ما تایمر ۶ را بطوریکه هر (PSC=71, ARR=10) 10us آپدیت شود و اینترپت بدهد برای Trig کردن DAC انتخاب می کنیم. و همچنین مقدار Amplitude را ۲۵۵ لحظه می کنیم.



شکل ۱۸

- پروژه Generate می کنیم تا وارد Keil شویم.
- در ابتدا peripheral های مورد نیاز را فعال می کنیم و همچنین متغیرهای لازم را تعریف می نماییم:

نکته: تعداد داده هایی که تبدیل می کنیم باید متناسب دوره تناوب 50us مربوط به Trig تایمر باشد. بطور مثال اگر ۱۰۰ داده را انتخاب کنیم، (همانطور که از پیش محاسبه نمودیم هر تبدیل 1.16us زمان میبرد) مدت زمان 1.16us طول می کشد در حالیکه دوره تناوبی که برای آپدیت تایمر ۳ انتخاب نمودیم 50us بوده است. بنابراین ۲۰ داده را انتخاب می کنیم.

```
uint32_t ADCValue[20];
float Sensor;
uint8_t TData[1];
```

```
HAL_TIM_Base_Start(&htim3);
```

```

HAL_ADC_Start_DMA(&hadc1, ADCValue, 20);

HAL_DAC_Start(&hdac, DAC_CHANNEL_1);

HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_1, 50);

```

۴- حال برنامه را بصورت زیر مطابق روش‌های پیشین می‌نویسیم.

```

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    uint32_t sum=0;

    if (hadc->Instance == ADC1)
    {

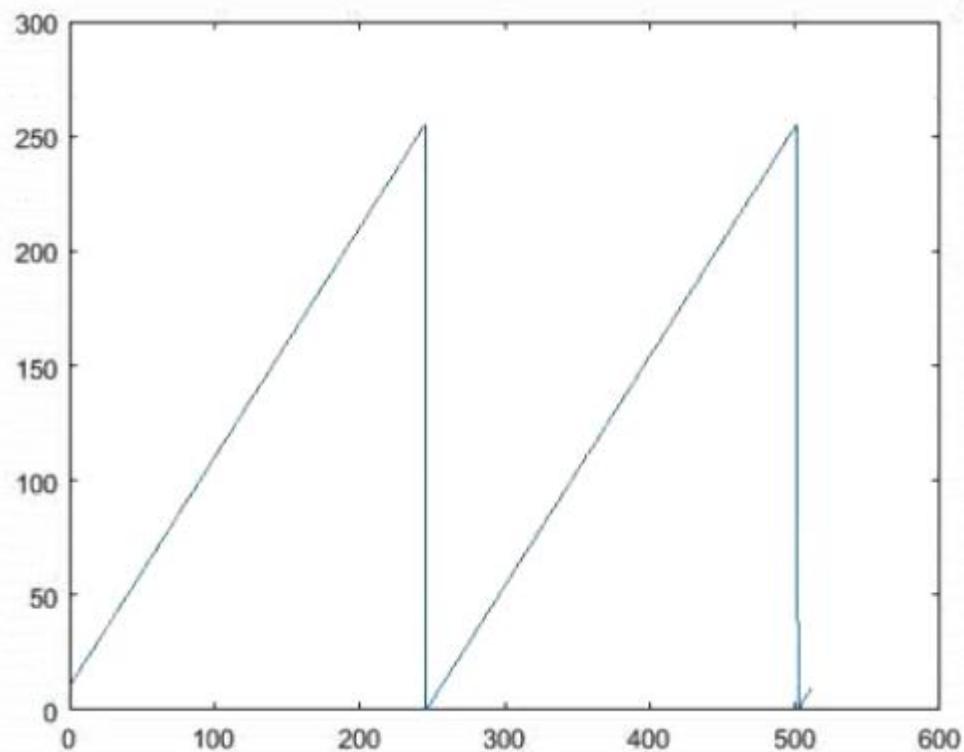
        for(int i=0 ; i<20 ; i++)
        {
            sum+=ADCValue[i];
        }

        Sensor=sum/20.0;

        TData[0]=Sensor/16.0;
        HAL_UART_Transmit(&huart2, TData, 1, 100);
    }
}

```

۵- حال موج مثلثی را به پایه ADC وصل می‌کنیم و نمودار آن را با استفاده دستورات ذکر شده در متلب به ترتیب در شکل‌های ۱۹ مشاهده می‌کنیم.



(١٩) شكل

# فصل یازدهم

## SPI پروتکل

### Serial peripheral interface

#### ۱-۱۱ SPI پروتکل

پروتکل SPI مانند Usart دیتا را به صورت سریال منتقل می کند. SPI. یکی از پرسرعت ترین رابط های ارتباطی می باشد. در SPI مانند Usrt دو دستگاه فرستنده و گیرنده وجود دارد ولی بخلاف usart که گیرنده و فرستنده بودن آن محدود نبود یعنی هر کدام از دستگاه ها می توانستند همزمان هم فرستنده باشند هم گیرنده در SPI مقداری محدودیت وجود بطوریکه هر کدام از دستگاه ها نمی توانند بطور همزمان هم فرستنده باشند هم گیرنده.

در SPI همیشه یک Master داریم که مشخص کننده آن است که چه زمانی داده ارسال شود به عبارتی دستور ارسال اطلاعات را می دهد و یک یا چند Slave داریم که اطلاعات را دریافت می کند و می توانند به Master ارسال کنند.

بطور کلی SPI چهار سیم برای ارتباط دارد. چون SPI سنکرون می باشد باید یک عامل سنکرون کنند بین دو دستگاه وجود داشته باشد. که در حالت USR این عامل می توانست کلاک یا برد (Budrate) باشد. این سنکرون سازی در SPI به عهده سیم کلاک می باشد که وظیفه تأمین کلاک آن همیشه به عهده master می باشد (به همین دلیل slave نمی تواند مستقل اطلاعاتی را ارسال کند).

دو سیم دیگر در ارتباط SPI وجود دارد که مشخص کننده جهت ارسال اطلاعات می باشند-put (slave out) (slave out master input) یعنی اطلاعات از Master وارد و از Slave خارج می شود که به صورت برعکس (put-master output) (output-master output) گردد که جهت جریان انتقال اطلاعات برعکس حالت قبلی می باشد. در حالت USAR این سیم ها (Tx-Rx) به صورت ضربدری وصل می شوند ولی در اینجا به صورت مستقیم وصل می شوند.

سیم چهارم ChipSelect نام دارد که اجازه دریافت یا عدم دریافت اطلاعات را به Slave می دهد. در Master و Slave یک DataRegister وجود دارد که می تواند ۸ یا ۱۶ بیت باشد.

در صورت استفاده از یک Master و یک Slave نیازی به استفاده از ChipSelect نیست و کافیست ChipSelect را صفر کنیم که هر اطلاعاتی فرستادیم دریافت شود.

## مزایای ارتباط سریال: SPI:

سریعتر از سریال آسنکرون است.

سخت افزار دریافت می تواند یک شیفت رجیستر ساده باشد.

از چندین slave پشتیبانی می کند.

## معایب ارتباط SPI:

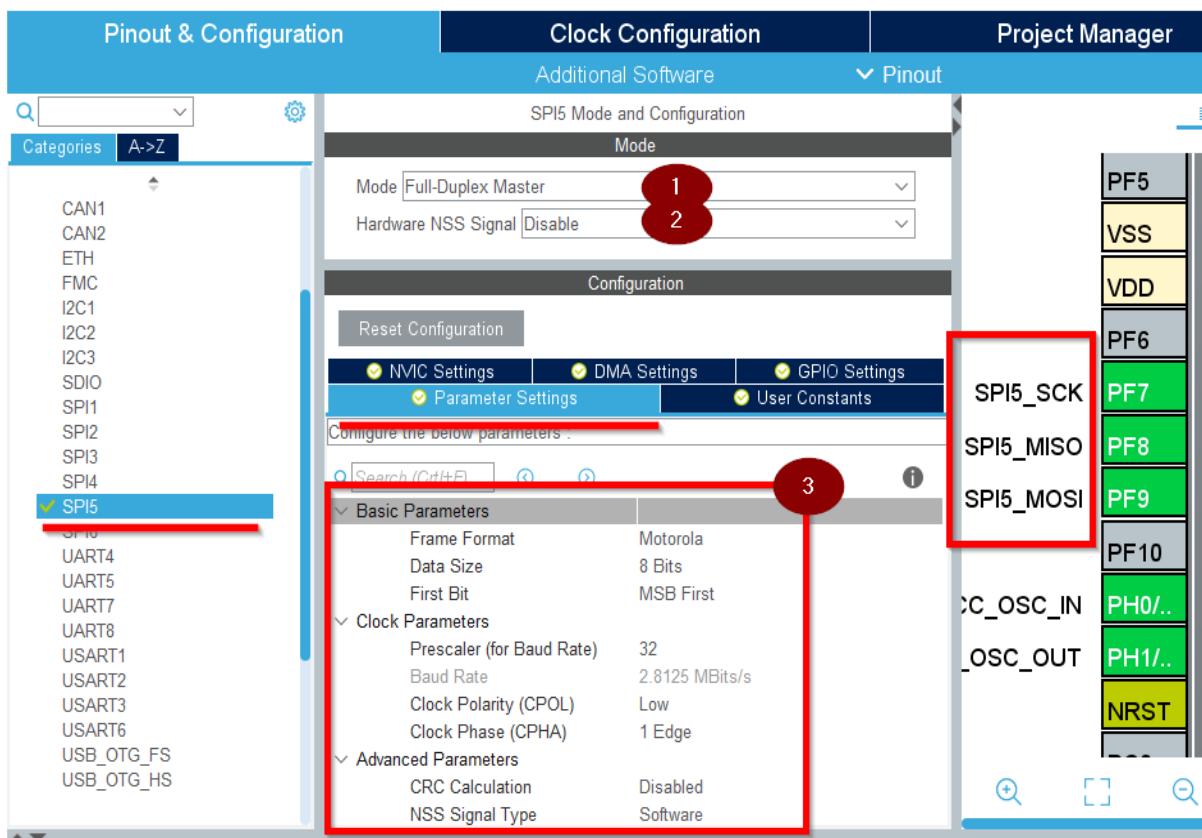
- به خطوط سیگنال بیشتری (سیم ها) نسبت به سایر روش های ارتباطی نیاز دارد.

- ارتباطات باید از قبل به طور دقیق معرفی شوند (شما نمی توانید مقادیر تصادفی داده را هر زمان که بخواهید ارسال کنید).

Master باید تمام ارتباطات را کنترل کند Slave ها نمی توانند به طور مستقیم با یکدیگر ارتباط برقرار کنند.

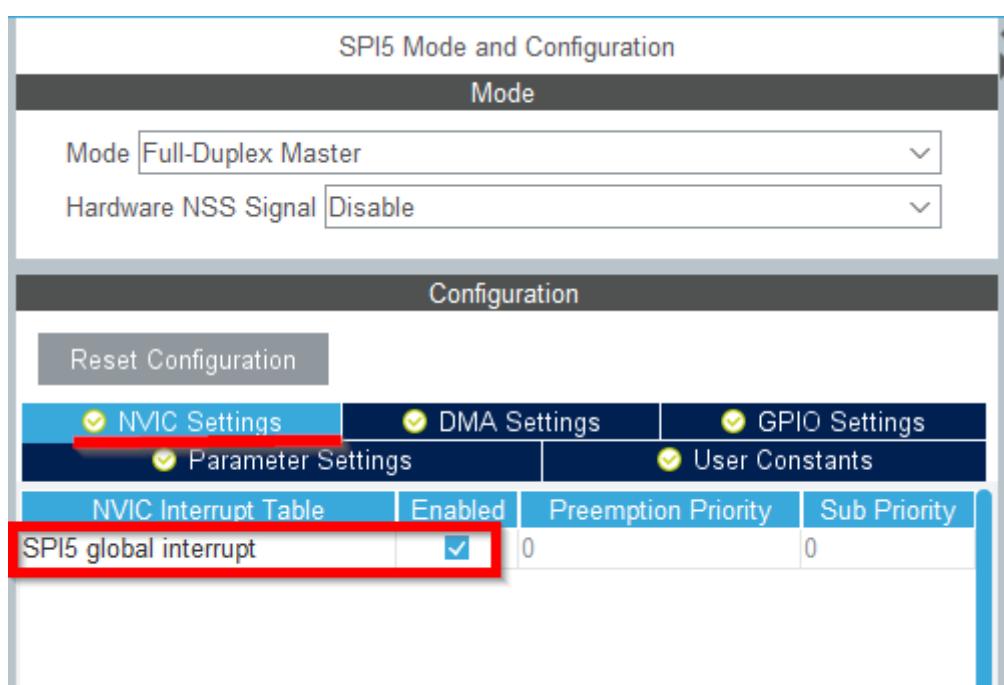
در صورت اتصال دو دستگاه با ارتباط SPI به هم دیگر که میکروکنترلر ما نقش Master را دارد تنظیمات را به صورت زیر انجام می دهیم.

- ۱ Cube را باز می کنیم و تنظیمات اولیه را انجام می دهیم.
- ۲ چون می خواهیم میکروکنترلر ما نقش Master را داشته باشد مد کاری را Full-Duplex Master انتخاب می کنیم (شکل ۱ گزینه ۱).
- ۳ با توجه به اینکه از یک Master و یک Slave استفاده می کنیم نیاز به فعال کردن ChipSelect نیست (شکل ۱ گزینه ۲) در صورت Master بودن ChipSelect باید به صورت Output و در صورت Slave بودن باید به صورت Input انتخاب گردد. در قسمت Parameter Setting نیز پارامترها مطابق شکل که یک معیار استاندارد در اغلب دستگاه ها می باشد تنظیم می گردد البته در صورت نیاز به سرعت بیشتر می توان با کاهش PSC بر سرعت BaudRate افزود.



(شکل ۱)

۴- همچنین ایتراپت SPI را نیز فعال می‌کنیم تا در زمان ارسال و در یافتن اطلاعات به ما اطلاع دهد(شکل ۲).



## (شکل ۲)

۵- پروژه را Generate میکنیم تا Keil باز شود.

۶- دستور های مورد نیاز در Keil برای:

الف- ارسال: شامل ۴ ورودی می باشد، ورودی اول Handeler ماست که در اینجا `&hspi` می باشد و دومی یک پوینتر جهت معرفی داده مورد نظر که بهتر است به صورت آرایه تعریف گردد و سومی تعداد درایه‌ای است که می خواهیم ارسال کنیم و چهارمی نیز TimeOut می باشد(کد ۱). بطور مثال(کد ۲).

```
HAL_StatusTypeDef HAL_SPI_Transmit(SPI_HandleTypeDefDef  
*hspi, uint8_t *pData, uint16_t Size, uint32_t Timeout)  
(کد ۱)
```

```
HAL_SPI_Transmit(&hspi5, TData, 1, 100);  
(کد ۲)
```

ب- دریافت: مانند قبلی (کد ۳) بطور مثال (کد ۴).

```
HAL_StatusTypeDef HAL_SPI_Transmit(SPI_HandleTypeDefDef  
*hspi, uint8_t *pData, uint16_t Size, uint32_t Timeout)  
(کد ۳)
```

```
HAL_SPI_Transmit(&hspi5, RData, 6, 100);  
(کد ۴)
```

پ- ارسال و دریافت اطلاعات: مانند قبلی با این تفاوت که هم آرایه ارسالی هم آرایه دریافتی به همراه هم اضافه می شوند که در اینجا تعداد ورودی ها پنج تا می شود(کد ۵) بطور مثال(کد ۶).

```
HAL_StatusTypeDef HAL_SPI_TransmitReceive(SPI_HandleTypeDefDef  
*hspi, uint8_t *pTxData, uint8_t *pRxData, uint16_t  
Size, uint32_t Timeout)  
(کد ۵)
```

```
HAL_SPI_TransmitReceive(&hspi5, TData, RData, 6, 100);  
(کد ۶)
```

۷- برای ایترپت نیز مانند گذشته تعریف تابع (کد ۷ و ۸) آن را نوشته و سپس به تعریف آن رفته و آن را پس از تابع `Main` می کنیم که در این صورت با دریافت (کد ۹) و ارسال(کد ۱۰) داده ایترپت داده می شود که با توجه به نیاز دستواراتی را داخل تابع ایترپت می نویسیم.

HAL\_SPI\_TxCpltCallback

(٧٥)

HAL\_SPI\_RxCpltCallback

(٨٦)

```
void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi)
{
    if(hspi==&hspi5)
    {
        }
}
```

(٩٥)

```
void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi)
{
    if(hspi==&hspi5)
    {
        }
}
```

(١٠٥)