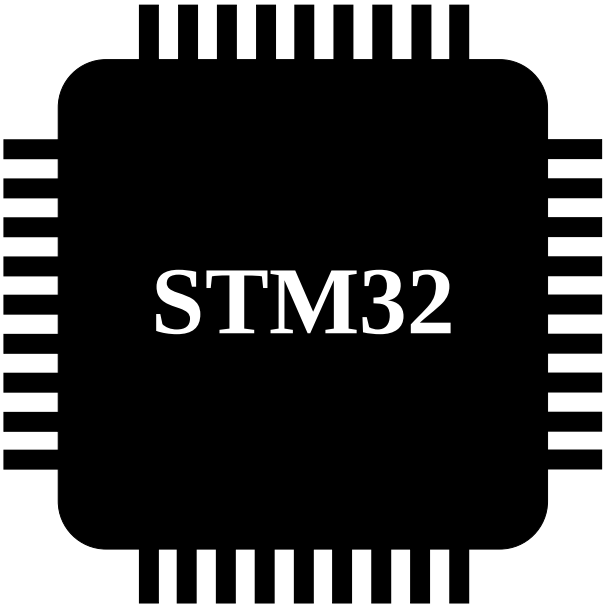
دانشگاه سجاد

دانشکده‌ی مهندسی برق و مهندسی پزشکی

دستورکار آزمایشگاه سیستم‌های دیجیتال ۲

تهیه و تنظیم: محمدمهدی دزیانی

زیر نظر: دکتر حمیدرضا رضایی ده‌سرخ



فهرست مطالب

مقدمه

با توجه به استفاده روزافزون از تراشه‌های مختلف میکروکنترلر در پروژه‌های گوناگون، یادگیری کار با میکروکنترلرها به یک نیاز اساسی برای هر دانشجوی رشته برق تبدیل شده است. این تراشه‌ها در سیستم‌های کنترل صنعتی، رباتیک، اینترنت اشیا (IoT) و بسیاری از کاربردهای دیگر نقشی حیاتی ایفا می‌کنند. در راستای اهمیت این موضوع، دروس متعددی در برنامه درسی دانشجویان کارشناسی گنجانده شده است تا به این هدف دست یابند و آن‌ها را برای ورود به حوزه‌های تخصصی مرتبط آماده سازند.

درس آزمایشگاه سیستم‌های دیجیتال ۲ یکی از دروس عملی در این راستا محسوب می‌شود. این درس نه‌تنها بر پیاده‌سازی مدارهای مبتنی بر میکروکنترلر تمرکز دارد، بلکه دانشجویان را با برنامه‌نویسی میکروکنترلرها و نحوه تعامل آن‌ها با سایر تجهیزات سخت‌افزاری آشنا می‌کند. تسلط بر این مفاهیم به دانشجویان این امکان را می‌دهد تا پروژه‌های عملی خود را به شکل کارآمدتر طراحی و اجرا کنند و در مسیر توسعه فناوری‌های نوین گام بردارند.

این دستورکار براساس میکروکنترلر STM32H7B0VBT6 از شرکت STMicroelectronics با استفاده از نرم‌افزار STM32CubeIDE در قالب ١٤ آزمایش طراحی شده است. در این دستورکار تلاش شده است تا دانشجویان با بخش‌های مختلف میکروکنترلر آشنا شده و توانایی راه‌اندازی هر بخش را کسب کنند. به منظور تسهیل در یادگیری، آزمایش‌ها به صورت ساده طرح‌ریزی شده‌اند تا دانشجو در پایان، دید کلی نسبت به بخش‌های مختلف میکروکنترلر پیدا کند. با توجه به محدودیت زمانی کلاس‌ها، انتظار می‌رود دانشجویان پیش از ورود به کلاس، آزمایش مربوطه را به طور کامل مطالعه کرده و با آمادگی کامل در کلاس حاضر شوند.

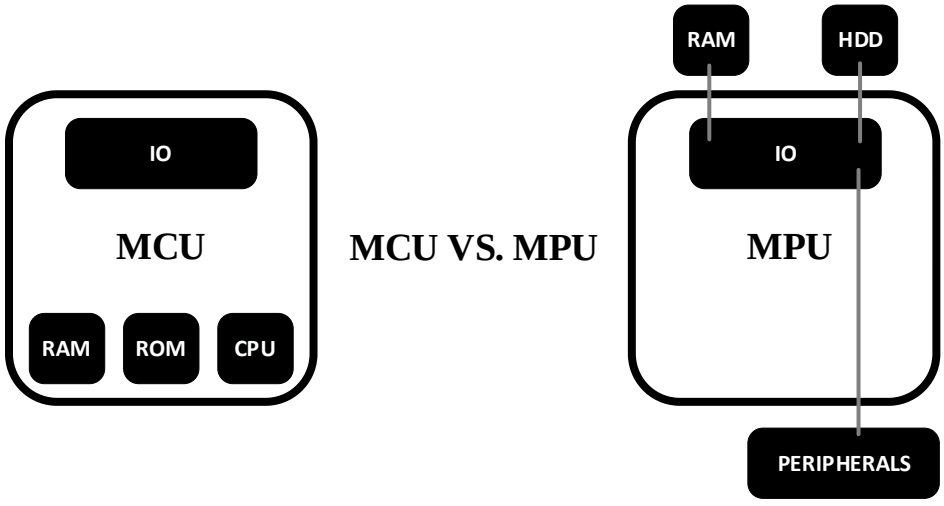
در پایان، بر خود لازم می‌دانم که از جناب آقای دکتر حمیدرضا رضایی ده‌سرخ که با راهنمایی‌های ارزشمند و حمایت‌های خود نقش بسزایی در پیشبرد این دستورکار داشتند، صمیمانه قدردانی کنم. همچنین از دانشجوی گرامی، جناب آقای مهدی حیدری، که با تلاش و همکاری مؤثر خود سهمی مهم در بهبود کیفیت این دستورکار ایفا کردند، نهایت سپاس را دارم.

محمدمهدی دزیانی

بهار ١٤٠٤

آشنایی با میکروکنترلر با هسته ARM و برد EWB-STM32H7B0

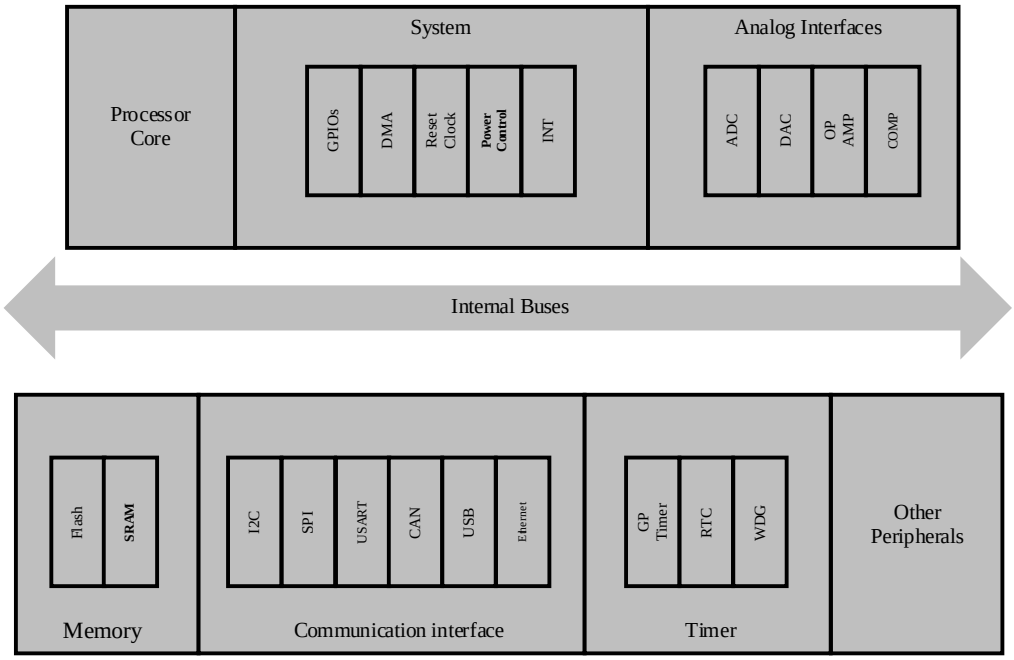
در دنیای امروز، میکروکنترلرها نقش بسیار مهمی در طیف وسیعی از پروژه‌های الکترونیکی ایفا می‌کنند. این تراشه‌های کوچک و قدرتمند شامل پردازنده، حافظه و تجهیزات جانبی مختلفی هستند که همگی بر روی یک تراشه قرار دارند. این ویژگی‌ها، میکروکنترلرها را به ابزاری ارزان‌تر و کوچکتر نسبت به میکروپروسسورها تبدیل کرده است. میکروکنترلرها و میکروپروسسورها هر دو جزء کلیدی الکترونیک سیستم‌های نهفته هستند، اما از نظر طراحی و کاربردهایشان تفاوت‌های مهمی دارند. میکروپروسسورها معمولاً به عنوان قلب یک سیستم کامپیوتری بزرگ‌تر شناخته می‌شوند و عمدتاً برای پردازش داده‌ها و اجرای برنامه‌های پیچیده طراحی شده‌اند. این پردازنده‌ها به صورت مستقل از حافظه و سایر اجزاء عمل می‌کنند و نیاز به اجزاء خارجی بیشتری برای تشکیل یک سیستم کامل دارند. از سوی دیگر، میکروکنترلرها به طور خاص برای کنترل سیستم‌های تعبیه‌شده طراحی شده‌اند و معمولاً شامل پردازنده، حافظه RAM و ROM، واحدهای ورودی/خروجی (I/O) و... در یک تراشه واحد هستند. این ترکیب باعث می‌شود میکروکنترلرها مناسب‌تر برای کاربردهایی مانند کنترل ابزار، حسگرها و دستگاه‌های خانگی باشند که نیاز به حجم کمتری از داده و مصرف انرژی پایین دارند. بنابراین، در حالی که میکروپروسسورها برای کارهای محاسباتی پیچیده‌تر و پردازش داده‌های گسترده‌تر مناسب هستند، میکروکنترلرها بیشتر برای وظایف خاص و کنترل سیستم‌های کوچک به کار می‌روند.



شکل(۱) : مقایسه میکروپروسسور و میکروکنترلر

میکروکنترلرها در انواع مختلفی از پروژه‌ها و دستگاه‌ها کاربرد دارند. یکی از مثال‌های رایج، سیستم‌های کنترل دما در ترموستات‌ها است که با اندازه‌گیری دما و تنظیم گرمایش یا سرمایش در ساختمان‌ها، راحتی کاربران را تأمین می‌کند. همچنین در وسایل خانگی هوشمند مانند یخچال‌ها و ماشین‌های لباسشویی، میکروکنترلرها برای بهینه‌سازی فرآیندها و کنترل عملکرد دستگاه‌ها استفاده می‌شوند. در عرصه تکنولوژی‌های پوشیدنی، این تراشه‌ها به حسگرهای ضربان قلب و فشار خون در ساعت‌های هوشمند کمک می‌کنند تا داده‌های مربوط

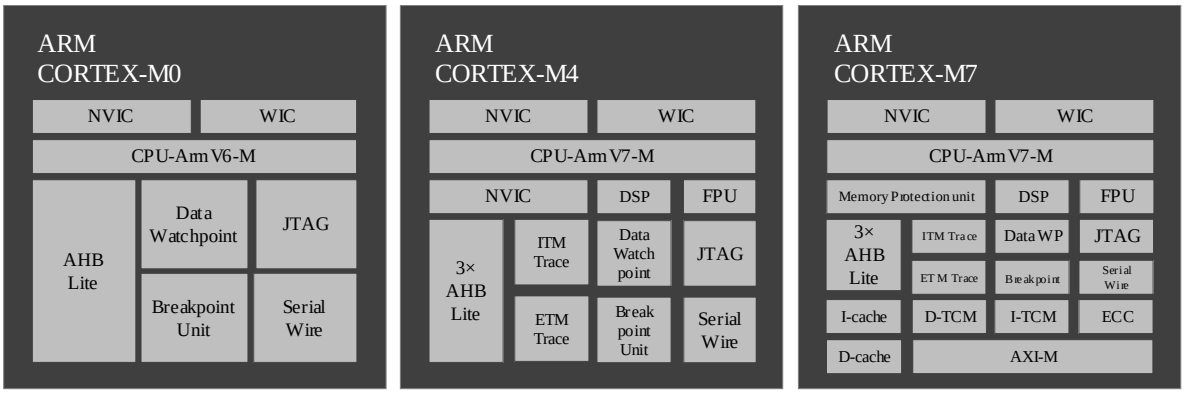
به سلامتی را جمع‌آوری و تحلیل کنند. به علاوه، در رباتیک، میکروکنترلرها به عنوان مغز سیستم عمل می‌کنند و مسئولیت کنترل حرکات، پردازش داده‌های سنسورها و تعامل با محیط را بر عهده دارند. این کاربردهای متنوع، نشان‌دهنده قدرت و انعطاف‌پذیری میکروکنترلرها در زندگی روزمره هستند.



شکل(۲): واحدهای جانبی موجود در میکروکنترلرها

یکی از معماری‌های پرکاربرد در طراحی پردازنده میکروکنترلرها، معماری ARM است. ARM یا Advanced RISC Machine به معنای دستگاه پیشرفته RISC است. عبارت RISC مخفف Reduced Instruction Set Computer است و به معنای رایانه‌ای با مجموعه‌ای از دستورالعمل‌های ساده‌شده به کار می‌رود. معماری RISC به شما امکان می‌دهد دستورات را به چندین دستورالعمل ساده و غیر پیچیده تقسیم کنید که برای تحقق اهداف کوچک طراحی شده‌اند. این معماری به دلیل کاهش تعداد دستورالعمل‌ها و استفاده کمتر از ترانزیستورها، کارایی بالایی دارد.

میکروکنترلرهایی که با معماری ARM ساخته شده‌اند، معمولا با نام Cortex-(A,R,M)X عرضه می‌شوند، A برای سیستم‌های کاربردی مانند تلفن همراه، تبلت و سیستم‌های نهفته، R برای سیستم‌های زمان واقعی یا بلادرنگ و M برای سیستم‌های میکروکنترلری است. خانواده Cortex-M شامل مدل‌های مختلفی مانند Cortex-M0، Cortex-M3، Cortex-M4، Cortex-M7 و ... است که هر کدام ویژگی‌ها و کاربردهای خاص خود را دارند. به عنوان مثال، Cortex-M4 با واحد پردازش اعشاری (FPU) برای کاربردهای پردازش سیگنال دیجیتال مناسب است، در حالی که Cortex-M0 برای کاربردهای ساده‌تر و کم‌مصرف طراحی شده است.



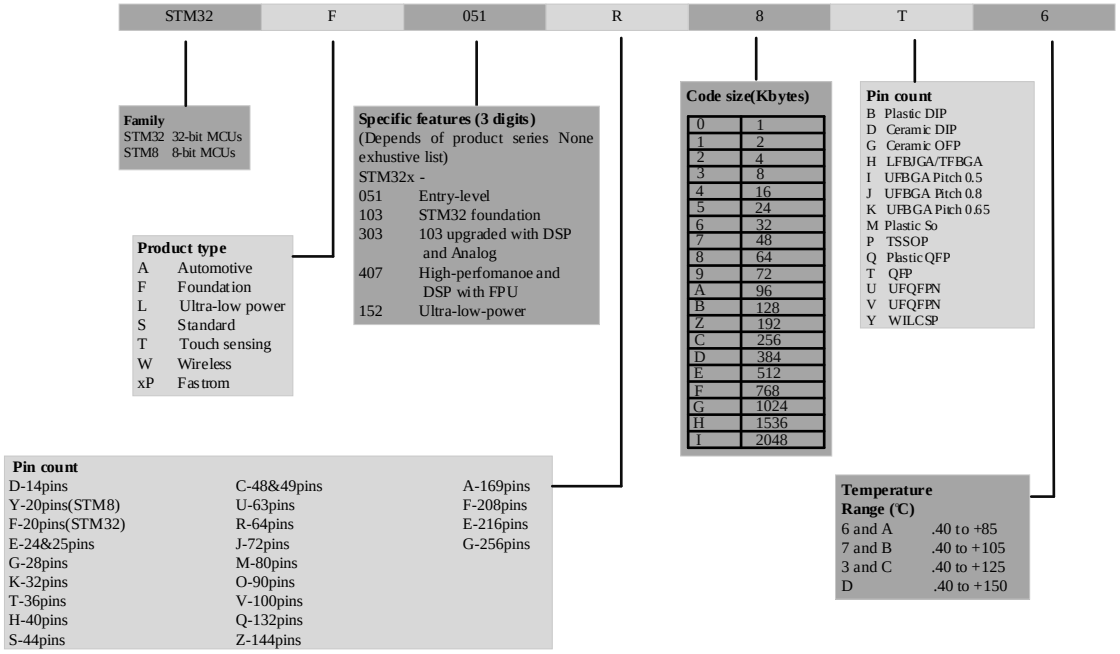
شکل(۳): مقایسه معماری‌های مختلف ARM

خانواده STM32 از شرکت STMicroelectronics، میکروکنترلرهای ۳۲ بیتی مبتنی بر پردازنده‌های ARM Cortex-M را شامل می‌شود. این میکروکنترلرها به دلیل امکانات گسترده و عملکرد بالا، انتخابی مناسب برای طیف وسیعی از پروژه‌ها هستند. با گذشت زمان، تجهیزات جانبی جدیدی برای پاسخ به نیازهای بازار به میکروکنترلرها اضافه شده‌اند. میکروکنترلرهای STM32 به چهار دسته اصلی تقسیم می‌شوند: کارایی بالا، میان‌رده، کم‌مصرف و بی‌سیم. هر یک از این دسته‌ها ویژگی‌های منحصر به فردی دارند که آن‌ها را برای کاربردهای خاص مناسب می‌سازد.

	Family	Core	Max Frequency	Flash
High Performance	STM32H7	Cortex-M7-Cortex-M4	480 MHz - 240 MHz	1 to 2 MB
	STM32F7	Cortex-M7	216 MHz	256 KB to 2 MB
	STM32F4	Cortex-M4	180 MHz	64 KB to 2 MB
	STM32F2	Cortex-M3	120 MHz	128 KB to 1 MB
Mainstream	STM32G4	Cortex-M4	170 MHz	32 to 512 KB
	STM32F3	Cortex-M4	72 MHz	16 to 512 KB
	STM32F1	Cortex-M3	72 MHz	16 KB to 1 MB
	STM32G0	Cortex-M0+	64 MHz	16 to 512 KB
	STM32F0	Cortex-M0	48 MHz	16 to 256 KB
Ultra-low-power	STM32U5	Cortex-M33	160 MHz	1024 to 2048 KB
	STM32L5	Cortex-M33	110 MHz	256 to 512 KB
	STM32L4+	Cortex-M4	120 MHz	512 KB to 2 MB
	STM32L4	Cortex-M4	80 MHz	64 KB to 1 MB
	STM32L1	Cortex-M3	32 MHz	32 to 512 KB
	STM32L0	Cortex-M0+	32 MHz	8 to 192 KB
Wireless	STM32WB	Cortex-M4-Cortex-M0+	64 MHz -32 MHz	256 KB to 1 MB
	STM32WL	Cortex-M4	48 MHz	64 KB to 256 KB

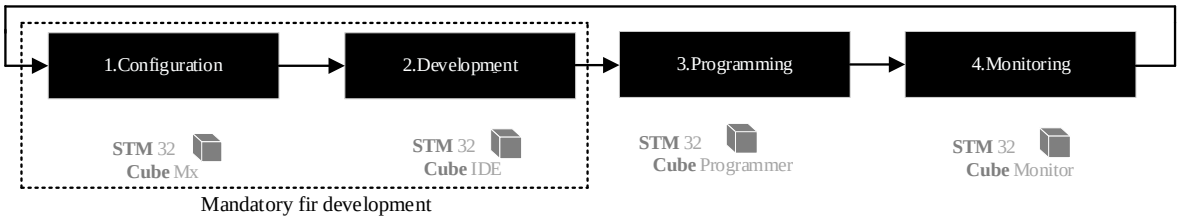
جدول(۱): مقایسه سرعت پردازش و حافظه سری‌های مختلف میکروکنترلر ARM

جدول فوق به انتخاب خانواده میکروکنترلر مناسب بسته به نیاز برنامه کمک می‌کند، اما برای انتخاب صحیح میکروکنترلر، باید ویژگی‌ها و اجزای محصول را دقیقا بررسی کرد. همچنین روشی برای نام‌گذاری میکروکنترلرهای شرکت ST وجود دارد که براساس آن می‌توان با برخی از ویژگی‌های هر میکروکنترلر آشنا شد. نام‌گذاری میکروکنترلرهای STM32 براساس روش زیر انجام می‌شود.



شکل(٤): تشخیص بعضی از امکانات میکروکنترلرهای STM32 با استفاده از شناسه قطعه

STMicroelectronics برخی از برنامه‌های نرم‌افزاری را ارائه می‌کند که به کاربران اجازه می‌دهد با استفاده از بردها و MCU برنامه‌نویسی کنند.



شکل(۵): نرم‌افزارهای مرتبط با میکروکنترلرهای STM32

نرم افزار STM32CubeIDE: این نرم‌افزار از یک محیط گرافیکی برای تنظیم امکانات جانبی میکروکنترلر، یک محیط برنامه‌نویسی و همینطور محیطی برای عیب‌یابی پروژه و انتقال کد کامپایل شده بر روی میکروکنترلر تشکیل شده است. از نکات مثبت این نرم افزار می‌توان به این موضوع اشاره نمود که در هر زمان در طول توسعه کاربر می‌تواند به تنظیمات اولیه و پیکربندی ابزارهای جانبی دسترسی داشته باشد.

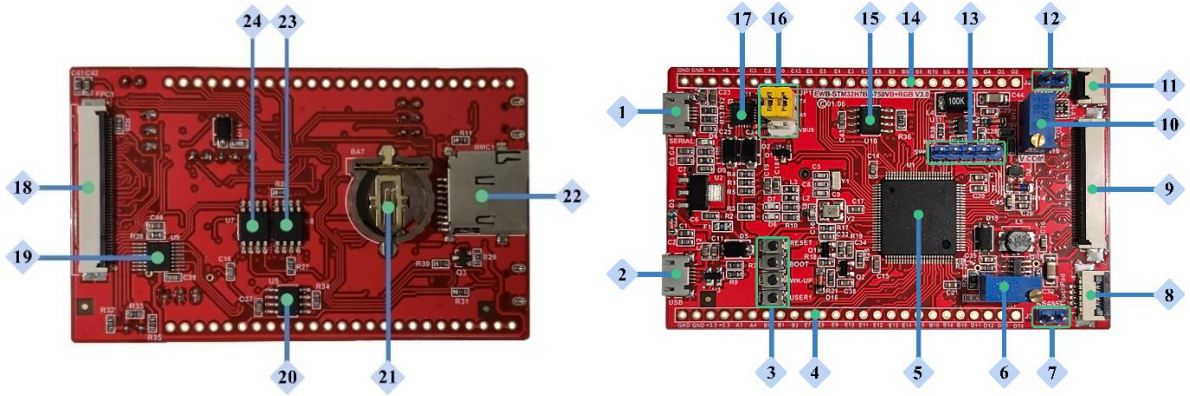
نرم افزار STM32CubeMX: این نرم‌افزار یک محیط گرافیکی ساده برای تنظیمات اولیه میکروکنترلر است. خروجی این نرم‌افزار به صورتی در نظر گرفته شده است که بر روی انواع کامپایلرها بدون نیاز به فایل‌های دیگر اجرا می‌شود. این نرم‌افزار امکان انتخاب میکروکنترلر یا برد میکروکنترلری، تنظیم پایه‌ها و کلاک، انتخاب و تنظیم امکانات جانبی، تخمین مصرف توان تراشه و انتخاب روش برنامه‌ریزی میکروکنترلر را فراهم می‌کند.

نرم افزار STM32CubeProgrammer: این نرم‌افزار محیطی برای خواندن، نوشتن و هم‌چنین دیباگ کردن حافظه داخلی از طریق پورت‌های JTAG و SWD است. همچنین اجازه دسترسی به حافظه داخلی STM32 مانند Flash، ROM و ... را ارائه می‌دهد.

نرم افزار STM32CubeMonitor: یک نرم‌افزار آنالایزر برای میکروکنترلرهای STM32 است. این برنامه از طریق پورت‌های JTAG و SWD با میکروکنترلر ارتباط می‌گیرد و امکان نظارت بر متغییرها مختلف به صورت بلادرنگ را می‌دهد.

* یک نرم‌افزار دیگر به نام ST-MCU-Finder وجود دارد که می‌توان براساس ویژگی‌های مورد نیاز، میکروکنترلر مورد نظر را پیدا کرد. همچنین تمامی نرم‌افزارهای ذکر شده در سایت ST به صورت رایگان در دسترس است. این ابزارها از سوی شرکت STMicroelectronics ارائه شده‌اند و از به‌روزرسانی‌های منظم و پشتیبانی رسمی برخوردارند.

معرفی برد EWB-STM32H7B0: برد EWB-STM32H7B0 یک برد ٤ لایه طراحی شرکت کویر الکترونیک است. این برد متشکل از یک میکروکنترلر STM32H7B0 است که از سری بالارده شرکت ST است. این برد شامل امکانات جانبی فراوانی برای انجام گستره متنوعی از پروژه‌های میکروکنترلری است. هم‌چنین تنوع درگاه‌های ارتباطی این امکان را فراهم می‌کند تا با دستگاه‌های الکترونیکی مختلف ارتباط برقرار کرده و تبادل اطلاعات داشته باشد.



شکل(٦): بُرد EWB-STM32H7B0 و اجزای مختلف آن

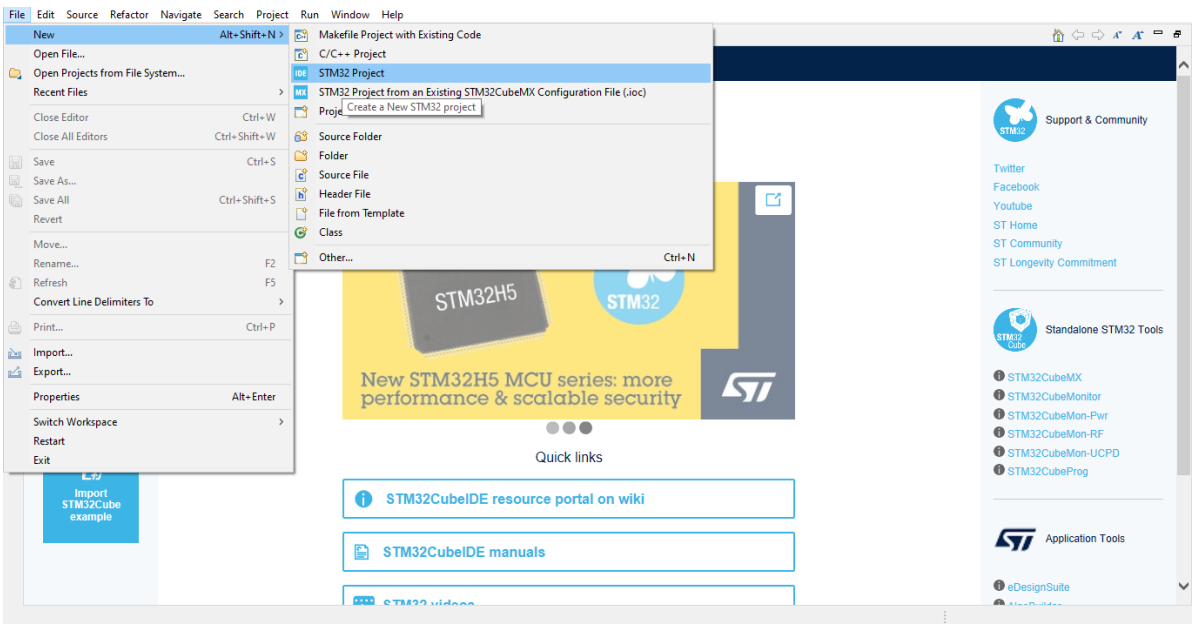
بخش‌های مختلف این برد معرفی شده است.

۱. USB Serial	۹. FPC40(RGB LCD)	۱۷. USB Serial (CP2104)
۲. USB OTG	۱۰. VR1 (V-COM)	۱۸. FPC50 (RGB LCD)
۳. RESET- BOOT-WKUP	۱۱. CPT TOUCH	۱۹. XPT2046
٤. J3(OUTPUT PORT)	۱۲. CAN OUT	۲۰. RS485 (SP3485)
٥. STM32H7B0VBT6	۱۳. SWD	۲۱. RTC BACKUP
٦. BLK (For Backlight)	١٤. J2(OUTPUT PORT)	۲۲. MICROSD
۷. RS485 OUT	١٥. CAN (SN65HDV230)	۲۳. QSPI(W25Q64)
۸. FPC4(RST TOUCH)	١٦. JP1/JP2(SERIAL/USB HOST)	٢٤. SPI(W25Q64)

نحوه ایجاد پروژه در نرم افزار STM32CubeIDE:

برای ایجاد یک پروژه جدید در نرم‌افزار CubeIDE مانند زیر عمل کنید:

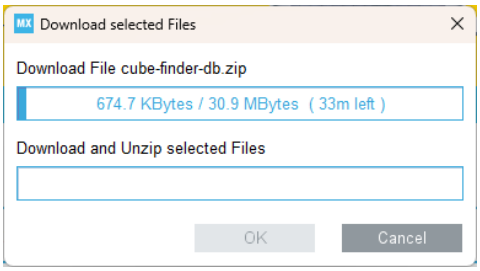
۱- پس از ایجاد یک مسیر اختصاصی و باز کردن نرم‌افزار، از سربرگ فایل گزینه New و سپس STM32 Project را انتخاب نمایید.



شکل(۷): ساخت پروژه جدید در نرم‌افزار CubeIDE

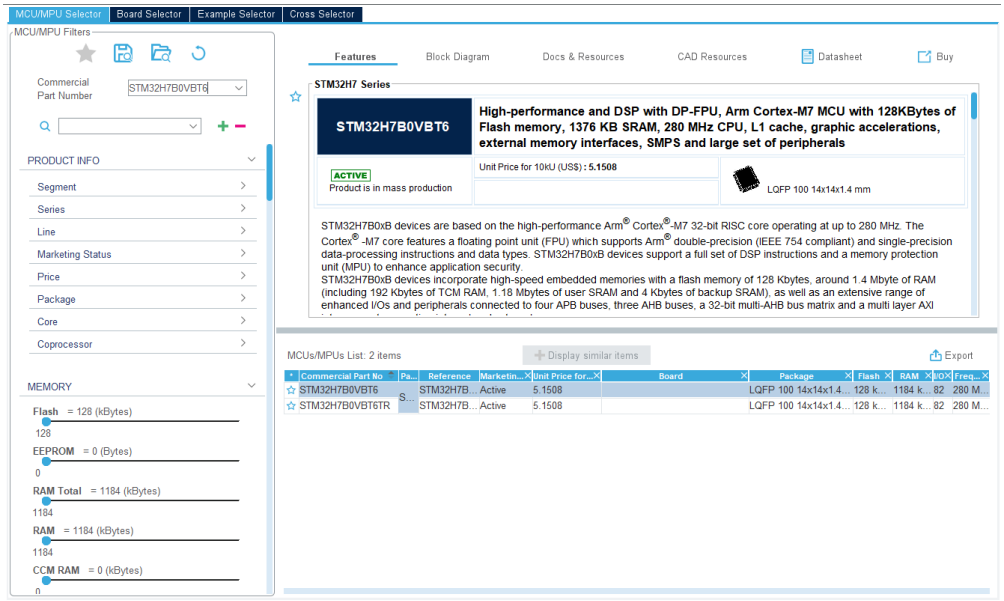
۲- در این مرحله اتصال اینترنت شما باید برقرار باشد تا نرم‌افزار ابزارهای جانبی و اطلاعات به روز را دریافت کند.

* در صورت بروز مشکل در این قسمت از ابزارهای رفع تحریم استفاده نمایید. در صورتی که به اینترنت متصل نبودید یا به دلیل شرایط تحریم نتوانستید این بخش را بروزرسانی کنید، می‌توانید با صرفه نظر از این مرحله به صورت آفلاین نیز میکروکنترلر مورد نظر را انتخاب کنید.



شکل(۸): تصویر بروزرسانی داده‌های نرم‌افزار CubeMX در ابتدای اجرا

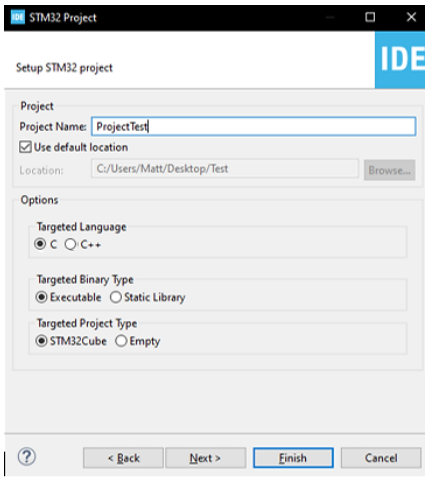
۳- از بخش Target selection در سربرگ MPU/MCU Selector در بخش Commerical Part Number نام کامل میکروکنترلر را وارد کنید و مدل دقیق را انتخاب کنید.



شکل(۹): پنجره مربوط به انتخاب میکروکنترلر در نرم‌افزار CubeIDE

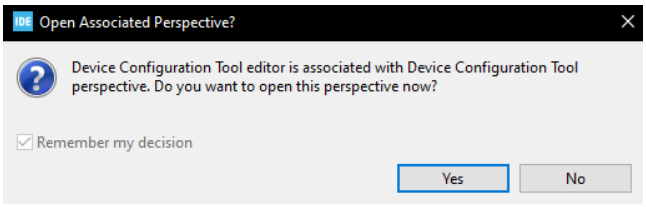
* در این دستورکار از میکروکنترلر STM32H7B0VBT6 استفاده می‌شود.

٤ – در بخش بعدی نام پروژه ، زبان برنامه نویسی، تنظیمات پروژه را طبق تصویر زیر انجام دهید.



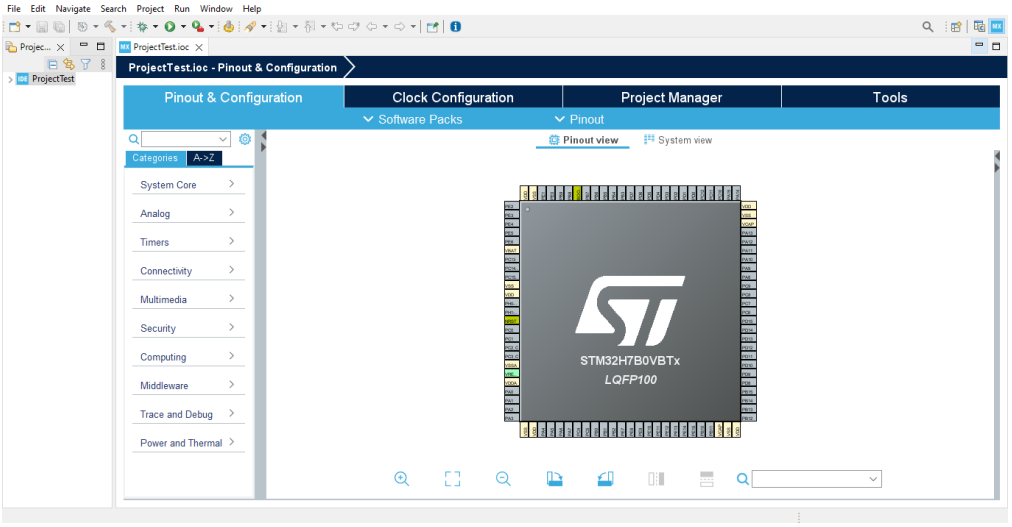
شکل(۱۰): پنجره انتخاب نام پروژه و محل جایگذاری آن در نرم‌افزار CubeIDE

۵- در پیام بعدی گزینه Yes را انتخاب کنید.

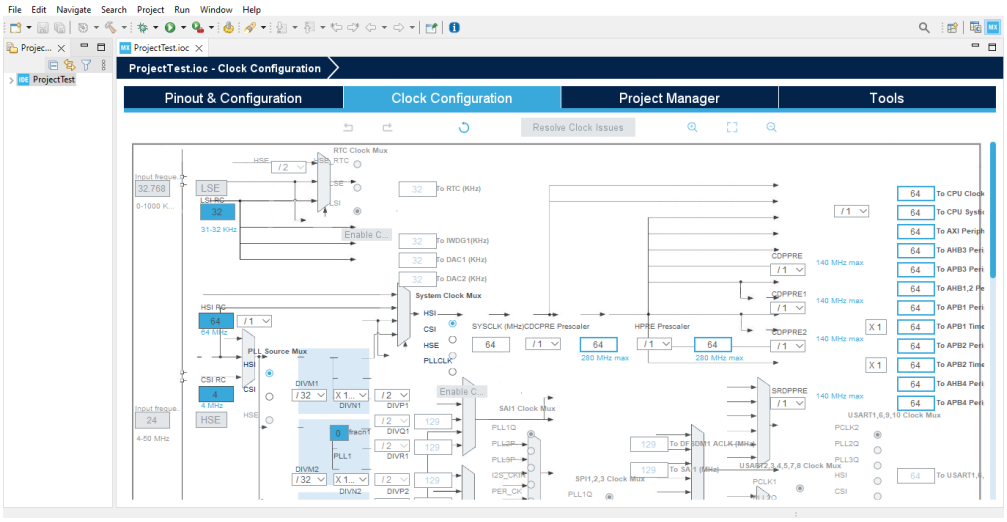


شکل(۱۱): پنجره انتخاب نمای Perspective در نرم‌افزار CubeIDE

۶- در این بخش می‌توان تنظیمات مختلف میکروکنترلر را انجام داد که طی پروژه‌های پیش‌رو توضیح داده خواهند شد.

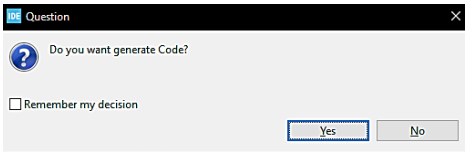


شکل(۱۲): پنجره تنظیمات ابتدایی پروژه قسمت امکانات جانبی

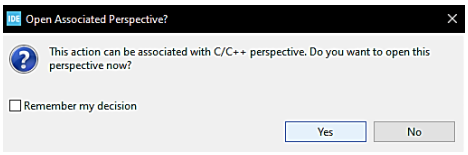


شکل(۱۳): پنجره تنظیمات ابتدایی پروژه قسمت کلاک و تقسیمات فرکانس

۷ – پس از انجام تنظیمات، کلیدهای میانبر Ctrl + S را بزنید و سپس گزینه Yes را انتخاب کنید.

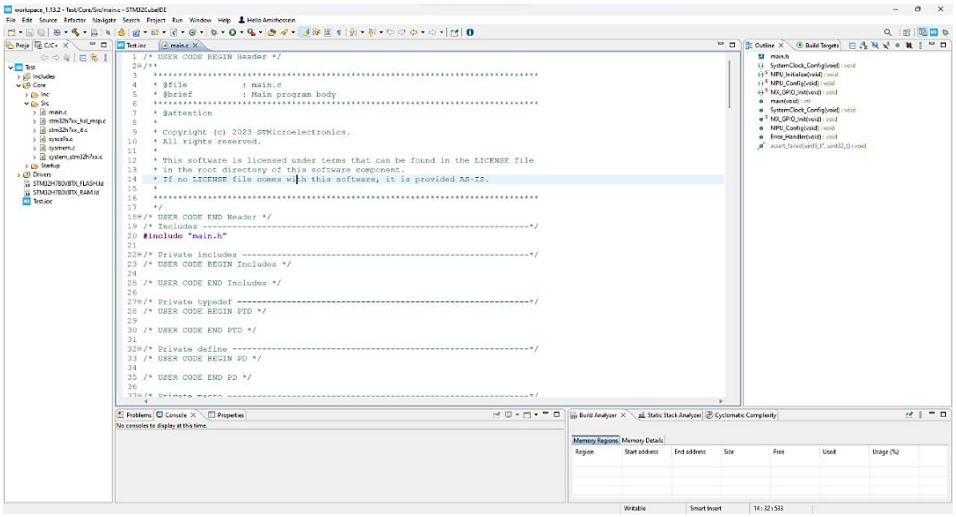


شکل(۱٤): پنجره تاییدیه ذخیره تنظیمات و تولید کدهای ابتدایی



شکل(۱۵): پنجره انتخاب نمای perspective هنگام ورود به فضای کدنویسی

۹- شمای نهایی محیط اجرا شده توسط کتابخانه HAL و ابزار CubeMX مانند شکل زیر است.



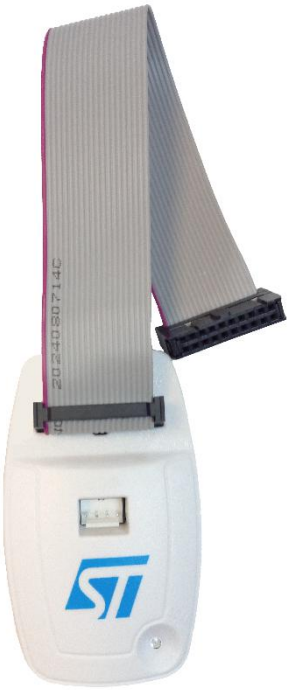
شکل(۱٦): فضای برنامه‌نویسی نرم‌افزار Cube IDE

* در این دستور کار برای کدنویسی از کتابخانه‌های HAL استفاده شده است. HAL که مخفف Hardware Abstraction Layer است، یک لایه واسطه بین کاربر و سخت‌افزار محسوب می‌شود که امکان تعامل با سخت‌افزار را بدون نیاز به دسترسی مستقیم به رجیسترها فراهم می‌کند. این کتابخانه با ارائه توابع سطح بالا، برنامه‌نویسی را ساده‌تر کرده و انتقال‌پذیری کد را افزایش می‌دهد، به‌طوری‌که برنامه‌های نوشته‌شده با HAL می‌توانند بدون تغییرات زیاد روی انواع مختلف میکروکنترلرهای STM32 اجرا شوند.

HAL شامل مجموعه‌ای از توابع استاندارد برای کنترل بخش‌های مختلف میکروکنترلر است که با نام‌گذاری مشخص، مانند HAL_GPIO_WritePin() برای تنظیم پایه‌های ورودی/خروجی، HAL_ADC_Start() برای شروع تبدیل آنالوگ به دیجیتال و HAL_UART_Transmit() برای ارسال داده از طریق UART ارائه شده‌اند. استفاده از این توابع به برنامه‌نویس کمک می‌کند تا بدون نیاز به درگیری با جزئیات سخت‌افزاری، به اجرای دستورات موردنظر بپردازد.

معرفی پروگرامر ST-Link V2:

پروگرامر و دیباگر ST-Link V2 برای میکروکنترلرهای سری STM8 و STM32 است که از پروتکل‌های رابط تک سیم (SWIM) و اشکال زدایی سری JTAG / سریال (SWD) برای برقراری ارتباط با میکروکنترلر STM8 و STM32 استفاده می‌کند. این پروگرامر دارای ایزولاسیون دیجیتال بین کامپیوتر و برد هدف نیز است. برای پروگرام کردن، پایه‌های RST، SWDIO، SWCLK، GND و VCC را به پایه‌های متناظر تنظیم شده در میکروکنترلر متصل می‌شود.



VTref	1	2	NC
Not used	3	4	GND
Not used	5	6	GND
SWDIO	7	8	GND
SWCLK	9	10	GND
Not used	11	12	GND
SWO	13	14	GND*
RESET	15	16	GND*
Not used	17	18	GND*
5V-Supply	19	20	GND*

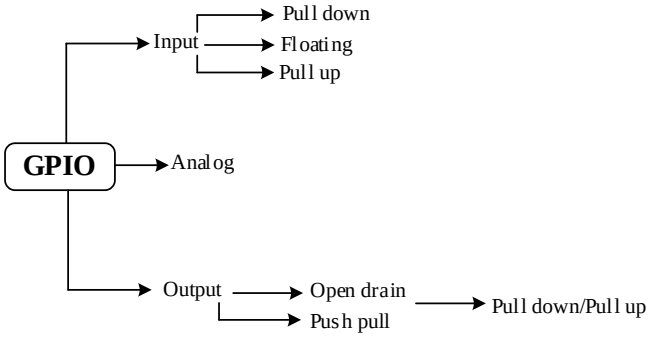
شکل(۱۷): مشخصات پایه‌های پورت موجود بر روی پروگرامر ST Link V2

آزمایش ۱: راه‌اندازی برد توسعه کویر و واحد GPIO

هدف آزمایش: در این آزمایش دانشجو با ایجاد پروژه در نرم‌افزار STM32CubeIDE، پروگرام کردن میکروکنترلر و همچنین دسترسی به پایه‌های میکرو آشنا می‌شود.

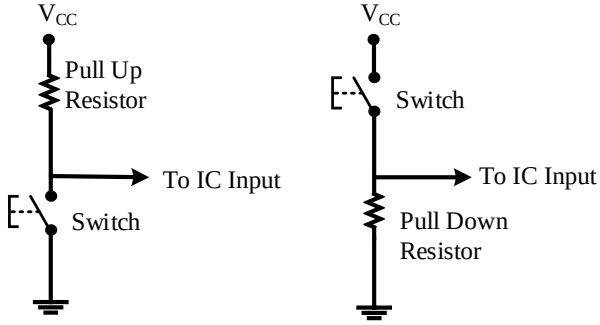
شرح آزمایش:

۱-مقدمه: GPIO یا General Purpose Input Output در STM32 برای توصیف پایه‌های ورودی/خروجی مورد استفاده قرار می‌گیرد. در حالی که بعضی از پین‌ها دارای هدف اختصاصی هستند، عملکرد یک پین GPIO قابل تنظیم است و می‌تواند توسط نرم‌افزار کنترل شود. هر بیت از پورت‌های ورودی/خروجی عمومی می‌تواند به‌طور مستقل توسط نرم‌افزار در حالت‌های ورودی، خروجی یا به صورت آنالوگ پیکربندی شود. در حالت پیش فرض تمام GPIOها درReset State قرار دارند تا مصرف انرژی کاهش پیدا کند. Reset State حالتی است که در آن پایه‌های میکروکنترلر در حالت High Impedance قرار می‌گیرند. هر پین را مانند تصویر زیر می‌توان بر روی حالت‌های مختلف تنظیم کرد.



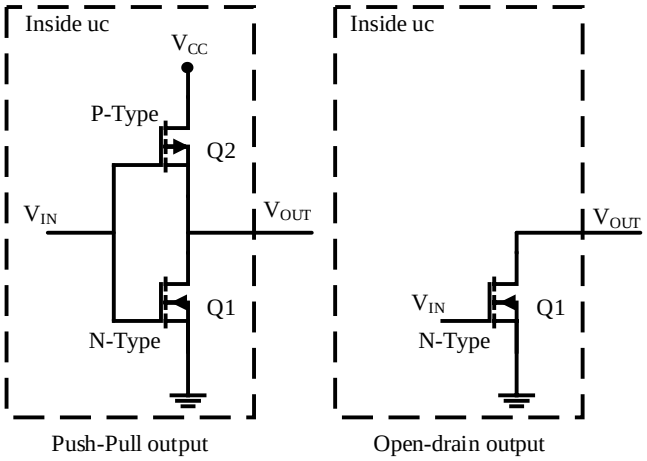
شکل(۱): حالت‌های قابل تنظیم بر روی پورت‌های میکروکنترلر STM32

در حالت ورودی پین می‌تواند با توجه به نوع عملکرد مورد نظر به صورت Pull down، Floating و یا Pull up قرار بگیرد. حالت Floating، حالتی است که پایه به صورت پیش فرض به VCC یا GND متصل نیست.



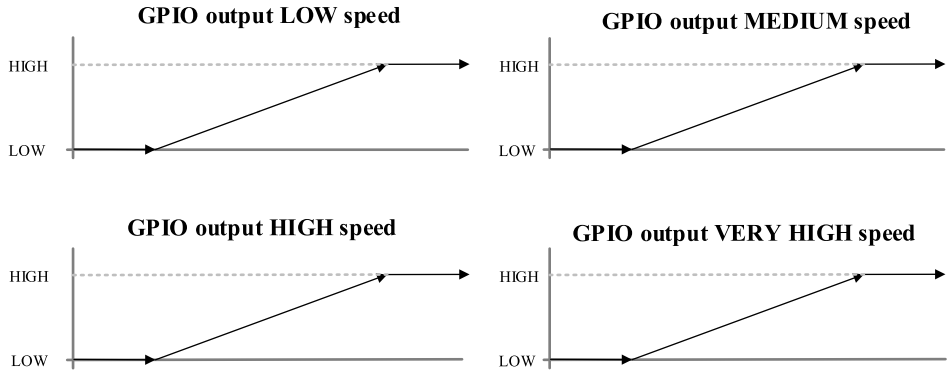
شکل(۲): شماتیک مربوط به نحوه Pull up و Pull down

درحالت خروجی پین می‌تواند با توجه به نوع عملکرد مورد نظر به صورت Open drain و یا Push Pull قرار بگیرد. در حالت Push-pull پین میکروکنترلر به صورت خروجی دارای جریان متغیر است. این حالت برای کاربردهایی که نیاز به تغییر سریع ولتاژ پین وجود دارد، مناسب است. در حالت open-drain، پین میکروکنترلر تنها می‌تواند به GND وصل یا آزاد شود. این حالت برای کاربردهایی که نیاز به مدارهایی با سطح ولتاژ متفاوت (برای اتصال به خروجی‌های باز در مدارهای چندگانه) استفاده می‌شود.



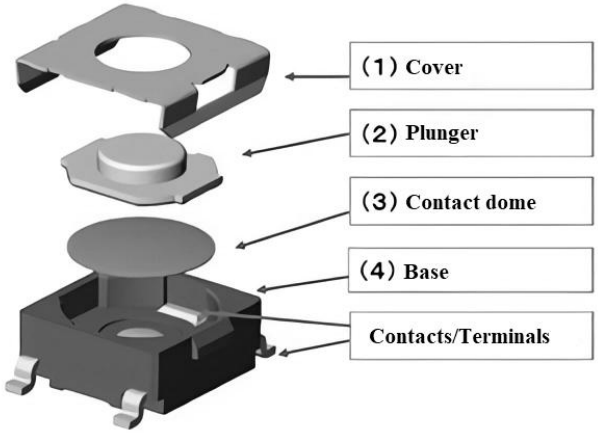
شکل(۳): تفاوت دو حالت Pushpull و Opendrain

می‌توان زمان لبه‌های بالارونده و پایین‌رونده در تغییر حالت از High به LOW و یا بالعکس پین میکرو را با توجه به عملکرد تغییر داد. به عبارت دیگر سرعت پایه را تنظیم کرد که به صورت LOW، High و Very High قابل تنظیم است. سرعت بالاتر GPIO باعث افزایش نویز EMI از STM32 و افزایش مصرف میکروکنترلر می‌شود. لازم است که سرعت GPIO را با سرعت مورد نیاز تطبیق داده شود. به عنوان مثال، استفاده از SPI در ٤٥ مگاهرتز به تنظیم سرعت بسیار بالا نیاز دارد.



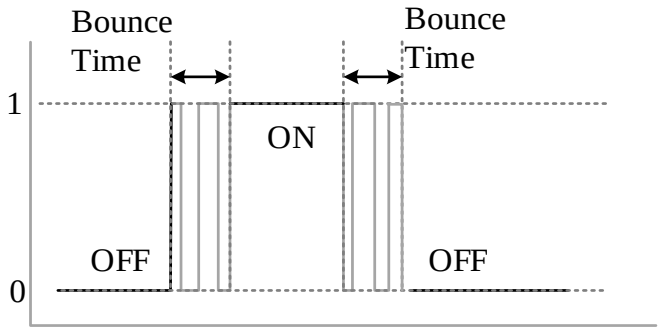
شکل(٤): مثالی از تفاوت تنظیم سرعت‌های مختلف بر روی پایه‌ها

کلید فشاری یا Push Button پس از تحریک یا فشار دادن، کنتاکت را وصل می‌کند و در حالت عادی باز بوده و با فشار دادن به صورت لحظه‌ای بسته می‌شود. Push Button یک حالت فنری دارد که با برداشتن فشار دست از روی آن کنتاکت مجددا باز می‌شود.



شکل(۵): نمای ساختمان یک کلید فشاری

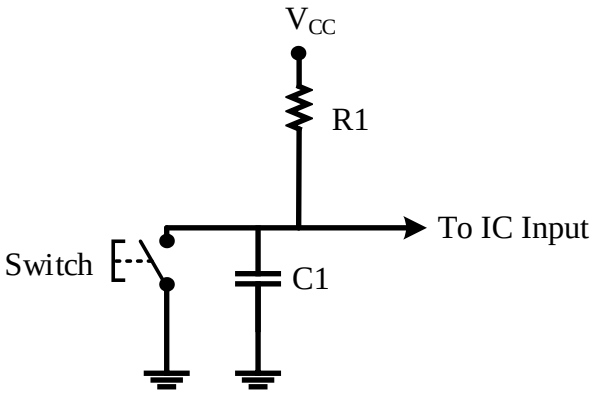
در حالت ایده‌آل انتظار می‌رود با قطع و وصل این عنصر، عملیات ۱ و ۰ منطقی بلافاصله و بدون اختلال انجام شود اما در واقعیت این اتفاق رخ نمی‌دهد. در واقع هنگام وصل یا قطع عناصر مذکور دو قطعه فلزی به یکدیگر متصل یا از هم جدا می‌شوند. در حالت وصل کلید هنگام برخورد دو قطعه فلزی به یکدیگر، به دلیل ضربه ممکن است چندین مرتبه قطعات از هم جدا و مجدد وصل شوند. این اتفاق تنها در عرض چند میکروثانیه اتفاق می‌افتد. به عبارتی کلید بین حالت قطع و وصل بودن در حال رفت و برگشت (Bouncing) است. در نهایت با پایان یافتن این نوسان، کلید بطور کامل وصل می‌شود. در حالت قطع کلید نیز این اتفاق با شدت کمتری رخ می‌دهد.



شکل(٦): تغییرات ولتاژ کلید در هنگام قطع و وصل شدن

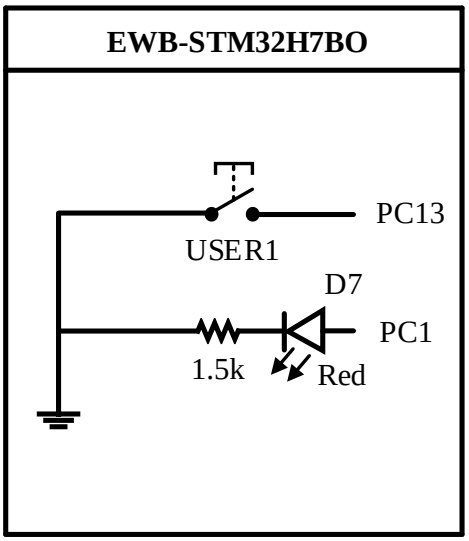
معمولا پردازنده‌ها با سرعت بسیار زیادی کار می‌کنند، بنابراین سخت‌افزار ممکن است تصور کند کاربر چندین مرحله متوالی کلید را قطع و وصل کرده است. این نکته نیز حائز اهمیت است که هر کلید مشخصات مختص به خود را داراست و تعداد دفعات قطع و وصل در بانسینگ و مدت زمان آن متفاوت خواهد بود. حتی اگر دو کلید مشابه را باهم مقایسه کنید احتمال اینکه بانسینگ آن‌ها باهم متفاوت باشد بسیار زیاد است. روش‌های مورد استفاده برای حذف بانسینگ، اصطلاحا دیبانسینگ نامیده می‌شود که بطور کلی می‌توان آن‌ها را به دو دسته روش‌های سخت‌افزاری و نرم‌افزاری تقسیم بندی کرد.

یک روش مرسوم و مناسب استفاده از یک مقاومت و خازن است به طوری که مقاومت pull-up شده و خازن نیز به زمین متصل باشد. هم‌چنین برای برطرف کردن این مشکل به صورت نرم‌افزاری می‌توان با استفاده از یک حلقه بی‌نهایت این مشکل را برطرف کرد که تا زمانی که حالت کلید تغییر نکند از حلقه خارج نشود.



شکل(۷): روش استفاده از کلید فشاری در مدارات

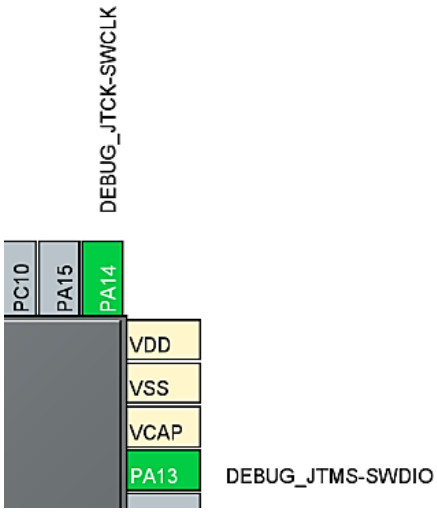
۲-شماتیک آزمایش: در این آزمایش از کلید فشاری با عنوان USER1 و از LED با عنوان D7 که بر روی برد EWB-STM32H7B0 قرار دارند، استفاده شده است. بنابراین نیازی به پیاده‌سازی شماتیک نیست.



شکل(۸): شماتیک داخلی کلید و LED در برد ارائه‌شده

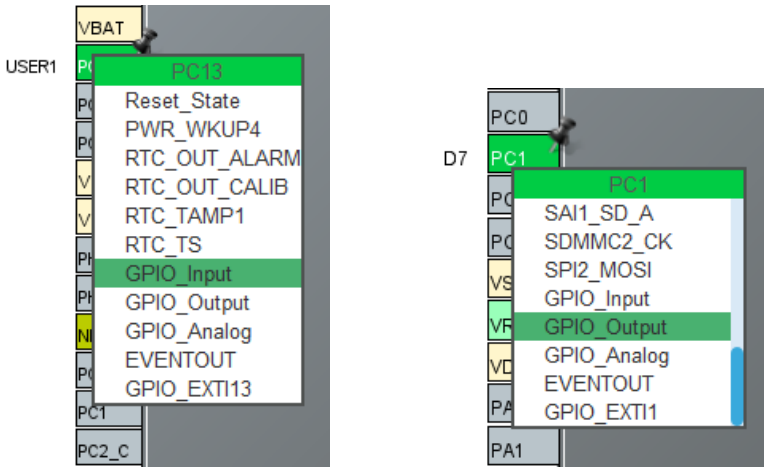
۳- تنظیمات نرم‌افزار STM32CubeIDE: مطابق مطالب گفته شده در مقدمه پروژه را ایجاد کنید و باقی تنظیمات را طبق مراحل ذیل پیش ببرید.

در هر آزمایش ابتدا از سربرگ Trace and debug گزینه Debug را انتخاب کنید و سپس در بخش Mode زیر گزینه Debug مورد Serial Wire را انتخاب نمایید. دقت کنید که حتما پایه‌های PA14 و PA13 باید مطابق تصویر سبز رنگ شوند. این کار جهت انتخاب شیوه پروگرام کردن میکرو انجام می‌شود.



شکل(۹): پین‌های پروگرام میکروکنترلر STM32H7B0VBT6

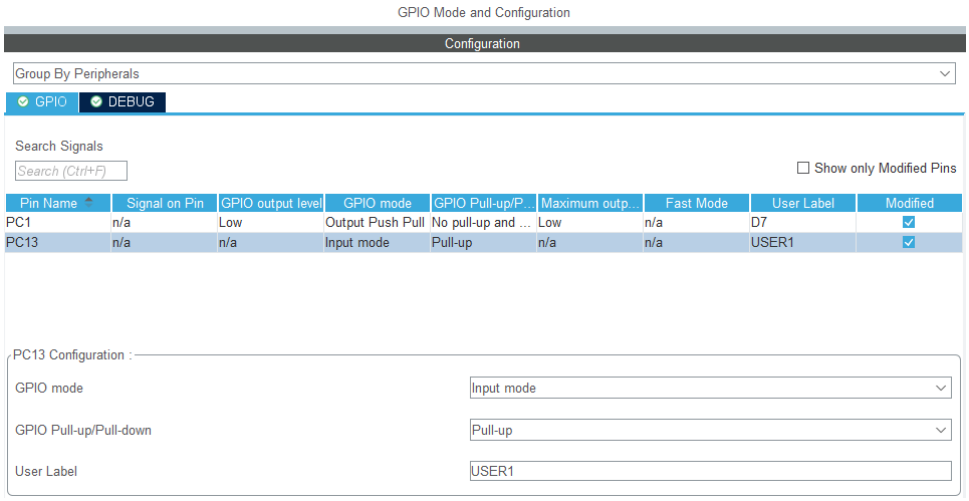
در ادامه پایه‌های ارتباطی با کلید و LED را تنظیم کنید. یک پایه ورودی برای کلید و یک پایه خروجی برای LED در نظر گرفته شده است. در ابتدا بر روی پایه مدنظر کلیک کنید و سپس از گزینه‌های ارائه شده GPIO_Output را برای پایه خروجی PC1، که به D7 بر روی برد توسعه متصل است و GPIO_Input را برای پایه ورودی PC13، که به کلید USER1 در برد توسعه متصل است، انتخاب کنید. سپس برای آدرس‌دهی مناسب‌تر به پایه‌ها طبق تصاویر زیر با راست کلیک کردن روی پایه گزینه Enter User Lable را انتخاب کنید. سپس نام‌های D7 و USER1 را به عنوان Label برای پایه‌های مورد نظر انتخاب کنید.



(ب) (الف)

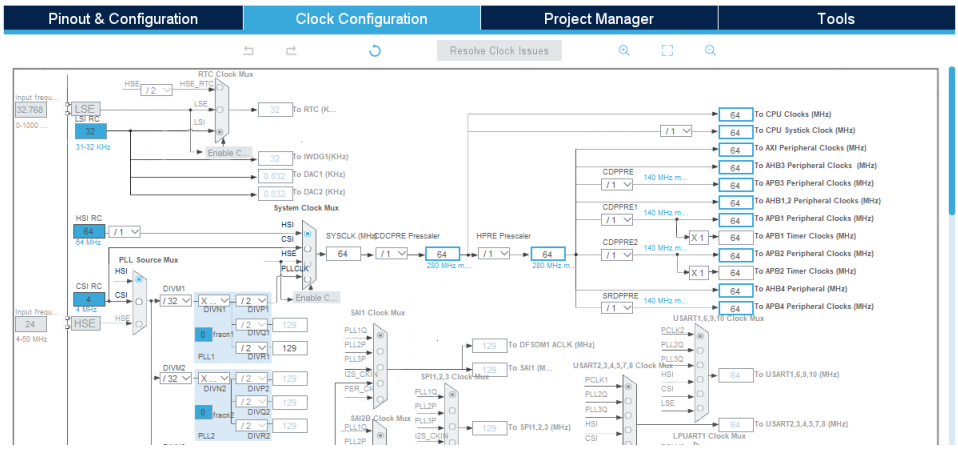
شکل(۱۰): (الف): پایه PC1 (LED) در مُد خروجی (ب): پایه PC13 (USER1) در مُد ورودی

در انتها به سربرگ System Core مراجعه کنید و زیرگزینه GPIO را انتخاب نمایید. در بین لیست پایه‌های ارائه شده در جدول Configuration پایه ورودی PC13 را انتخاب کنید و از گزینه‌های ارائه شده در جدول باید گزینه GPIO Pull-up/Pul-down را بیابید و مقدار آن را بر روی Pull-up تنظیم نمایید.



شکل(۱۱): تنظیمات واحد GPIO جهت تنظیم کلید و LED در نرم‌افزارCubeMx

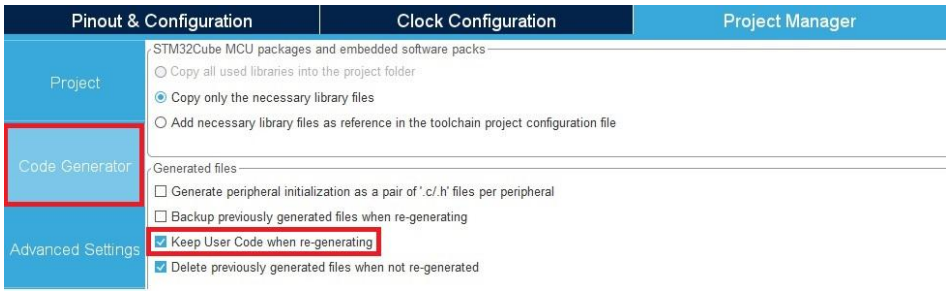
در این آزمایش تغییری در کلاک سیستم و سربرگ Clock Configuration ندهید.



شکل(۱۲): تنظیمات کلاک میکروکنترلر در نرم‌افزارCubeMx

بعد از ایجاد پروژه و تنظیم پایه‌ها و کلاک سیستم کلیدهای ترکیبی Ctrl + S را بزنید تا کدهای مورد نیاز متناسب با تنظیمات اعمال شده توسط کتابخانه HAL ایجاد شوند.

* حتما در بخش نشان داده شده، جهت حفظ کدهای نوشته شده، تغییرات را اعمال کنید.



شکل(۱۳): فعال‌سازی گزینه حفظ کد کاربر بعد از تولید فایل‌ها توسط نرم‌افزار

٤- معرفی توابع مورد استفاده: در این بخش به معرفی ٤ تابع HAL مربوط به GPIO پرداخته می‌شود:

1. HAL_GPIO_WritePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)

از این تابع برای مقدار دهی یا نوشتن در GPIO استفاده می‌شود. آرگومان اول GPIOx نام پورت مورد نظر برای مقداردهی است. آرگومان دوم شماره پین مربوط به GPIO هست و آرگومان سوم، PinState وضعیت خروجی پین را مشخص می‌کند که می‌تواند مقدار GPIO_PIN_RESET (0) یا مقدار GPIO_PIN_SET (1) را داشته باشد.

2. HAL_GPIO_ReadPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)

از این تابع برای خواندن مقدار GPIO استفاده می‌شود و می‌تواند مقادیر SET (1) یا RESET (0) را برگرداند. مانند تابع قبل آرگومان اول GPIOx نام پورت مورد نظر برای خواندن و آرگومان دوم شماره پین مربوط به GPIO است.

3. HAL_GPIO_TogglePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)

این تابع فقط مقدار پین مشخص شده را معکوس می‌کند.

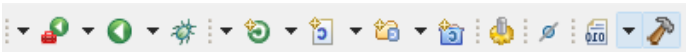
4. HAL_Delay(uint32_t Delay)

این تابع برای ایجاد تاخیر در برنامه استفاده می‌شود. عدد داخل پرانتز تاخیری در مقیاس میلی‌ثانیه ایجاد می‌کند.

٥- برنامه‌نویسی آزمایش: از قسمت Project Explorer->Core->Src به فایل main.c رفته و برنامه‌نویسی را آغاز کنید. فایل main.c شامل بخش‌های مختلفی است که در روند آزمایش‌ها، معرفی می‌شوند. در این برنامه به بخش while(1) رفته و کدهای زیر را در آن قسمت وارد کنید.

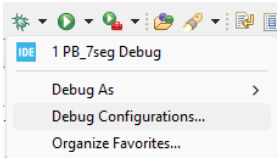
```
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
      //Check USER1 button state
      if(HAL_GPIO_ReadPin(USER1_GPIO_Port, USER1_Pin) == 0){
            //Turn on D7 pin
            HAL_GPIO_WritePin(D7_GPIO_Port, D7_Pin, GPIO_PIN_SET);
            // Delay for 1 second
            HAL_Delay(1000);
            //Turn off D7 pin
            HAL_GPIO_WritePin(D7_GPIO_Port, D7_Pin, GPIO_PIN_RESET);
      }
}
/* USER CODE END 3 */
```

در برنامه فوق در صورت فشرده شدن کلید USER1، LED را به مدت ۱۰۰۰ میلی‌ثانیه روشن و سپس LED خاموش می‌شود. برای Build کردن برنامه از قسمت سربرگ بر روی گزینه نشان داده شده کلیک نمایید و در صورت وجود خطا در برنامه آن را برطرف نمایید.



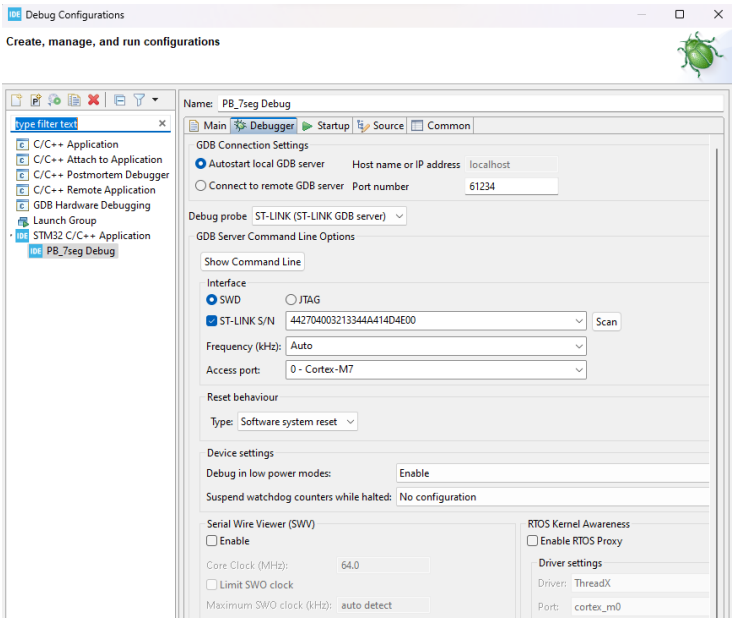
شکل(۱٤): نوار ابزار نرم‌افزارCubeMx

لازم به ذکر است که از  برای Build کردن پروژه، از  برای Debug و از  برای اجرای برنامه یا Run استفاده می‌شود. برای پروگرام کردن برنامه نیاز است که ابتدا تنظیمات لازم انجام شود. در سربرگ بالایی بر روی گزینه Debug و زیر گزینه Debug Configurations را انتخاب کنید.



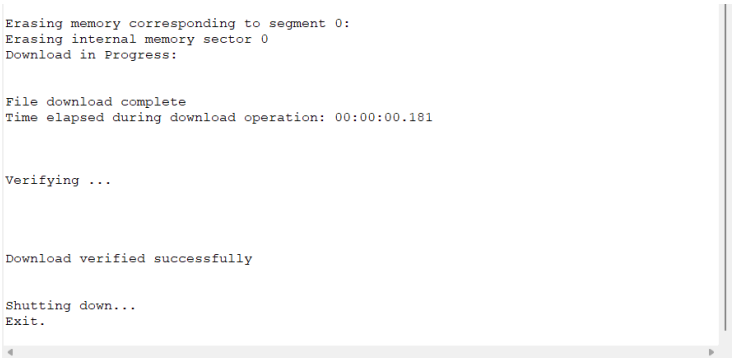
شکل(۱۵): موقعیت گزینهDebug Configuration در نرم‌افزار

پروگرامر را به کامپیوتر متصل نمایید. در پنجره باز شده سربرگ Debugger را انتخاب کنید و گزینه Debug probe را مطابق با تصویر تنظیم نمایید. در بخش گزینه ST-LINK S/N گزینه scan را بزنید. سپس از بین گزینه‌های ارائه شده سریال مرتبط با پروگرامر خود را انتخاب کنید.



شکل(۱٦): سربرگ مربوط به دستگاه Debugger در نرم‌افزار

تغییرات را ذخیره کنید و پنجره را ببندید. حال بر روی گزینه  کلیک کنید تا میکرو پروگرام شود. اگر تمام مراحل بالا را به درستی طی کرده باشید در پنجره کنسول باید پیام زیر را مشاهده کنید. در این قسمت اگر برنامه خطایی داشته باشد در سربرگ Problem می‌توانید مشاهده کنید و آن را در کد برنامه رفع کنید.



شکل(۱۷): سربرگ Problem در نرم‌افزار

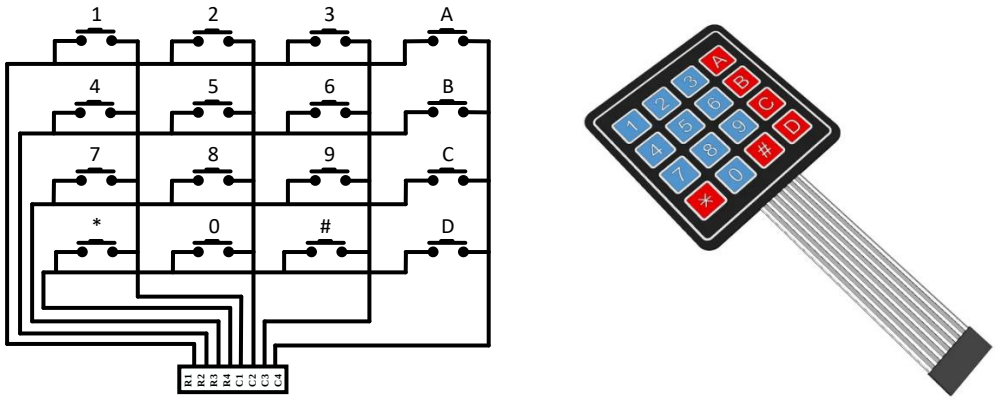
٦- تمرین: با استفاده از دستور TogglePin برنامه‌ای بنویسید که با هر بار فشردن کلید USER1 وضعیت LED معکوس شود.

آزمایش ۲: راه‌اندازی صفحه کلید و سون‌سگمنت

هدف آزمایش: در این آزمایش دانشجو با نحوه راه‌اندازی صفحه کلید ٤×٤ و سون‌سگمنت آشنا می‌شود.

شرح آزمایش:

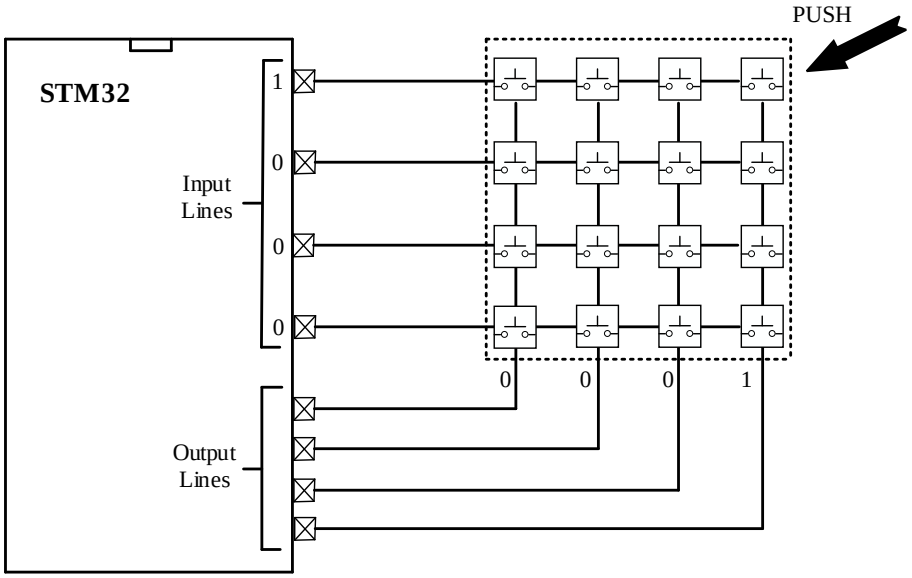
۱-مقدمه: صفحه کلیدهای ماتریسی متشکل از تعدادی کلید فشاری هستند که بصورت یک الگوی ماتریس به یکدیگر متصل شده‌اند و باعث اشغال پایه کمتری از میکروکنترلر می‌شود. کیپد ٤ در ٤ معمولاً شامل ١٦ کلید است که در چهار ردیف و چهار ستون چیده شده‌اند. این کیپد به کاربران این امکان را می‌دهد که با فشار دادن کلیدهای مختلف، شماره‌ها یا فرمان‌های خاصی را وارد کنند. استفاده از این نوع کیپد در دستگاه‌های مختلف مانند ماشین‌های حساب، پنل‌های دزدگیر و برخی از دستگاه‌های الکترونیکی دیگر رایج است. طراحی منظم آن باعث می‌شود که دسترسی به کلیدها آسان‌تر و سریع‌تر باشد.



(ب) (الف)

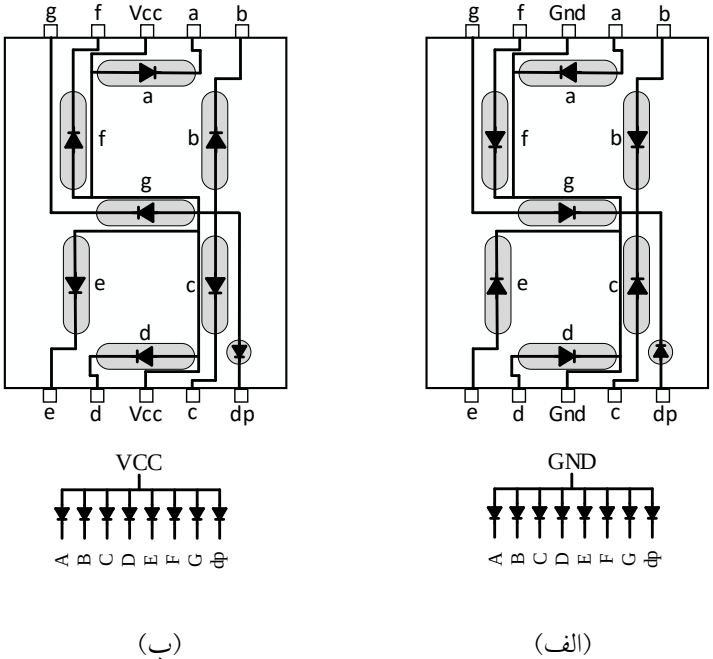
شکل(۱): (الف): کیپد ٤ در ٤ (ب): راهنمای اتصالات داخلی کیپد

طبق تصویر فوق به هشت پایه از میکروکنترلر نیاز است تا بتوان ١٦ کلید را کنترل کرد. برای این منظور ٤ پایه را به عنوان خروجی صفحه کلید (ردیف‌ها) و ٤ پایه را به عنوان ورودی صفحه کلید (ستون‌ها) در نظر گرفته می‌شود. برای اسکن یک کیپد ٤ در ٤ با استفاده از میکروکنترلر، شما باید ردیف‌ها و ستون‌های کیپد را به درستی متصل کنید. ابتدا میکروکنترلر یکی از ردیف‌ها را به حالت LOW (یا HIGH) قرار می‌دهد، در حالی که سایر ردیف‌ها را در حالت HIGH (یا LOW) نگه می‌دارد، سپس وضعیت هر ستون را بررسی می‌کند. اگر یک کلید فشرده شده باشد، سیگنالی از آن ستون به میکروکنترلر خواهد رسید و با ترکیب شماره ردیف و ستون می‌توان تشخیص داد کدام کلید فشرده شده است. این عملیات برای هر ردیف به صورت تکراری انجام می‌شود تا تمامی کلیدها اسکن شوند. این روش ساده و مؤثر به میکروکنترلر اجازه می‌دهد تا ورودی‌های کاربر را به سرعت پردازش کند.



شکل(۲): راهنمای اتصالات کیپد به میکروکنترلر

سون سگمنت‌ها قطعاتی هستند که در آنها از ۸ عدد LED استفاده شده است که ۷ عدد از این LEDها برای نمایش عدد و حروف هستند و یک LED برای نمایش ممیز است. LED از پایه‌های آند و کاتد تشکیل شده است لذا سون سگمنت نیز به دو دسته آند مشترک و کاتد مشترک تقسیم می‌شود.



(ب) (الف)

شکل(۳): (الف) سون‌سگمنت کاتدمشترک (ب) سون‌سگمنت آندمشترک

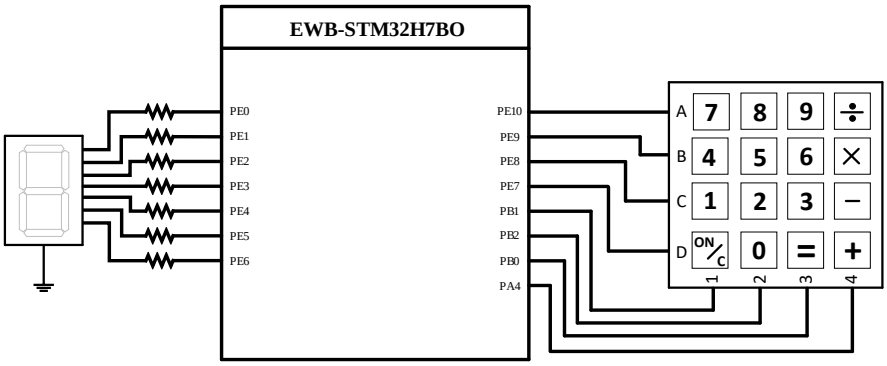
در سون سگمنت آندمشترک سر مثبت یا آند تمامی LEDها از داخل به پایه مشترک متصل شده‌اند که باید به V_{CC} وصل شوند و برای روشن کردن هر پایه باید آن را به زمین متصل کنیم. همچنین برای سون سگمنت کاتد مشترک پایه مشترک به زمین وصل می‌شود و برای روشن کردن هر پایه باید به آن ولتاژ V_{CC} اعمال

کنیم. در جدول زیر اعدادی که می‌توان روی یک سون سگمنت آندمشترک و کاتدمشترک نمایش داد آورده شده است.

Numbers	Common Cathode		Common Anode	
	(DP)GFEDCBA	HEX Code	(DP)GFEDCBA	HEX Code
0	00111111	0x3F	11000000	0xC0
1	00000110	0x06	11111001	0xF9
2	01011011	0x5B	10100100	0xA4
3	01001111	0x4F	10010000	0xB0
4	01100110	0x66	10011001	0x99
5	01101101	0x6D	10010010	0x92
6	01111101	0x7D	10000010	0x82
7	00000111	0x07	11111000	0xF8
8	01111111	0x7F	10000000	0x80
9	01101111	0x6F	10010000	0x90

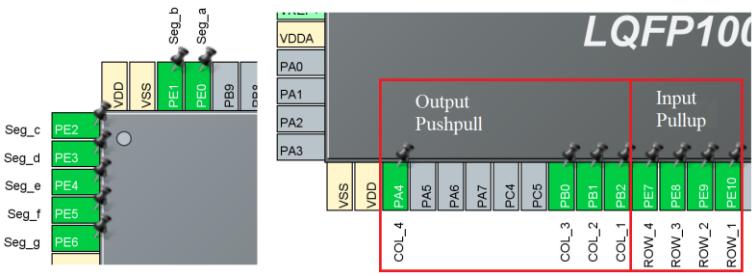
جدول(۱): مقادیر مورد نیاز جهت نمایش اعداد روی سون‌سگمنت

۲- شماتیک آزمایش: همانطور که در شماتیک پایه‌های اختصاص یافته برای سون‌سگمنت و صفحه کلید مشخص شده است. در این آزمایش با فشردن هر کلید عدد متناظر با آن روی سون سگمنت نمایش داده شود.



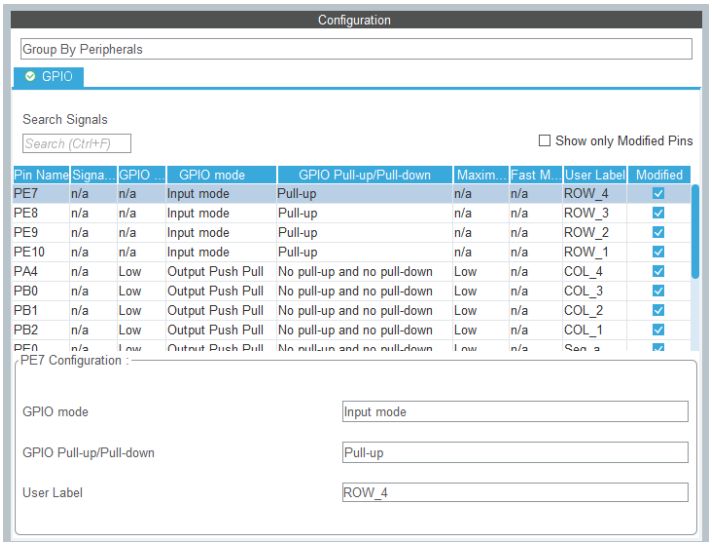
شکل(٤): شماتیک اتصال پایه‌های سون‌سگمنت و کیپد به برد

۳- تنظیمات نرم‌افزار STM32CubeIDE: پروژه را ایجاد کنید و نحوه پروگرام را بر روی serial wire تنظیم کنید. پایه های PE7 الی PE10 را ورودی و پایه های PB0 الی PB2 و پایه PA4 را خروجی تنظیم کنید. ۷ پایه خروجی نیز برای سون سگمنت در نظر بگیرید، برای این منظور از پایه PE0 تا PE6 را به ترتیب خروجی تنظیم کنید. دقت کنید که Label گذاری‌ها مانند شکل زیر انجام شود.



شکل(۵): تعریف پین‌های سون‌سگمنت و کیپد در نرم‌افزار

در بخش GPIO پایه های ورودی PE7 الی PE10 را بر روی Pull-up تنظیم کنید.



شکل(٦): تنظیمات واحد GPIO برای سون‌سگمنت و کیپد

بدون تغیر در سربرگ Clock Configuration کلید ترکیبی Ctrl + S را بزنید تا کدهای مورد نیاز متناسب با تنظیمات اعمال شده با استفاده از کتابخانه HAL ایجاد شوند.

٤- معرفی توابع مورد استفاده: در این آزمایش از توابع معرفی شده در آزمایش قبل استفاده شده است به جز Output data register یا رجیستر ODR که برای مشخص کردن وضعیت پینی که در حالت خروجی تنظیم شده استفاده می‌شود.

٥- برنامه‌نویسی آزمایش: در زیر یک متغیر از نوع Char تعریف شده تا خروجی تابع keypad_scan در آن قرار گیرد.

```
/* USER CODE BEGIN PV */
char key_pressed = 0;
/* USER CODE END PV */
```

در ادامه برنامه، تابع keypad_scan نوشته شده است. همانطور که از نام آن مشخص است وظیفه اسکن کردن صفحه کلید را برعهده دارد. در ابتدا در متغییر keys به معرفی متغیرهای صفحه کلید پرداخته شده است. در ادامه با توجه به الگوی گفته شده شروع به اسکن ستون‌های صفحه کلید کرده و سپس سطر مربوطه را تشخیص می‌دهد.

```
/* USER CODE BEGIN 0 */
//Scans the keypad and returns pressed key
char keypad_scan(void)
{//Keypad button mapping in a 4x4 matrix
  char keys[4][4] = {{'1', '2', '3', 'A'},
                     {'4', '5', '6', 'B'},
                     {'7', '8', '9', 'C'},
                     {'*', '0', '#', 'D'}};
```

```
  for(int i = 0; i < 4; i++)
    {
      // Set current column as output and low
      switch(i)
      {
        case 0:
             HAL_GPIO_WritePin(COL_1_GPIO_Port, COL_1_Pin, GPIO_PIN_RESET);
             HAL_GPIO_WritePin(COL_2_GPIO_Port, COL_2_Pin, GPIO_PIN_SET);
             HAL_GPIO_WritePin(COL_3_GPIO_Port, COL_3_Pin, GPIO_PIN_SET);
             HAL_GPIO_WritePin(COL_4_GPIO_Port, COL_4_Pin, GPIO_PIN_SET);
          break;
        case 1:
             HAL_GPIO_WritePin(COL_1_GPIO_Port, COL_1_Pin, GPIO_PIN_SET);
             HAL_GPIO_WritePin(COL_2_GPIO_Port, COL_2_Pin, GPIO_PIN_RESET);
             HAL_GPIO_WritePin(COL_3_GPIO_Port, COL_3_Pin, GPIO_PIN_SET);
             HAL_GPIO_WritePin(COL_4_GPIO_Port, COL_4_Pin, GPIO_PIN_SET);
          break;
        case 2:
             HAL_GPIO_WritePin(COL_1_GPIO_Port, COL_1_Pin, GPIO_PIN_SET);
             HAL_GPIO_WritePin(COL_2_GPIO_Port, COL_2_Pin, GPIO_PIN_SET);
             HAL_GPIO_WritePin(COL_3_GPIO_Port, COL_3_Pin, GPIO_PIN_RESET);
             HAL_GPIO_WritePin(COL_4_GPIO_Port, COL_4_Pin, GPIO_PIN_SET);
          break;
        case 3:
             HAL_GPIO_WritePin(COL_1_GPIO_Port, COL_1_Pin, GPIO_PIN_SET);
             HAL_GPIO_WritePin(COL_2_GPIO_Port, COL_2_Pin, GPIO_PIN_SET);
             HAL_GPIO_WritePin(COL_3_GPIO_Port, COL_3_Pin, GPIO_PIN_SET);
             HAL_GPIO_WritePin(COL_4_GPIO_Port, COL_4_Pin, GPIO_PIN_RESET);
          break;
    }
    // Read current rows
    if(HAL_GPIO_ReadPin(ROW_1_GPIO_Port, ROW_1_Pin) == 0)
      return keys[0][i];
    if(HAL_GPIO_ReadPin(ROW_2_GPIO_Port, ROW_2_Pin) == 0)
      return keys[1][i];
    if(HAL_GPIO_ReadPin(ROW_3_GPIO_Port, ROW_3_Pin) == 0)
      return keys[2][i];
    if(HAL_GPIO_ReadPin(ROW_4_GPIO_Port, ROW_4_Pin) == 0)
      return keys[3][i];
    }
    return 0; // No key pressed
    }
/* USER CODE END 0 */
```

در حلقه While اصلی با هر بار اسکن صفحه کلید در صورتی که که صفحه کلید فشرده شده باشد، خروجی متناظر با استفاده از دستور ODR در خروجی پورت E قرار می‌گیرد و در سون‌سگمنت نمایش داده می‌شود.

```
        /* USER CODE BEGIN 3 */
        key_pressed = keypad_scan();//Scan the keypad for a pressed key
        if(key_pressed != 0)//If a key is pressed
        {
            switch(key_pressed)//Determine which key was pressed
            {
                case '0':
                    GPIOE->ODR = 0x3F; // Display 0 on 7-segment (0b0111111)
                    break;
                case '1':
                    GPIOE->ODR = 0x06; // Display 1 on 7-segment (0b0000110)
```

```
                break;
            case '2':
                GPIOE->ODR = 0x5B; // Display 2 on 7-segment (0b1011011)
                break;
            case '3':
                GPIOE->ODR = 0x4F; // Display 3 on 7-segment (0b1001111)
                break;
            case '4':
                GPIOE->ODR = 0x66; // Display 4 on 7-segment (0b1100110)
                break;
            case '5':
                GPIOE->ODR = 0x6D; // Display 5 on 7-segment (0b1101101)
                break;
            case '6':
                GPIOE->ODR = 0x7D; // Display 6 on 7-segment (0b1111101)
                break;
            case '7':
                GPIOE->ODR = 0x07; // Display 7 on 7-segment (0b0000111)
                break;
            case '8':
                GPIOE->ODR = 0x7F; // Display 8 on 7-segment (0b1111111)
                break;
            case '9':
                GPIOE->ODR = 0x6F; // Display 9 on 7-segment (0b1001111)
                break;
            default:
                GPIOE->ODR = 0x00; // Reset display if an invalid key
        }
    }
```

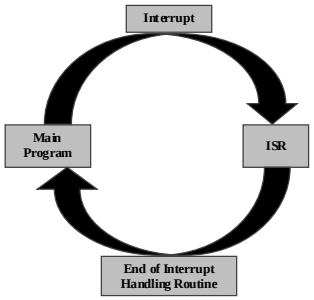
٦- **تمرین:** با تغییر در برنامه بالا چند حرف قابل نمایش را بر روی سون‌سگمنت نمایش دهید و همچنین یک کلید برای پاک کردن سون‌سگمنت در نظر بگیرید.

آزمایش ۳: راه‌اندازی وقفه‌ها در میکروکنترلر

هدف آزمایش: در این آزمایش دانشجو با مفهوم وقفه و نحوه فعال کردن آن آشنا می‌شود.

شرح آزمایش:

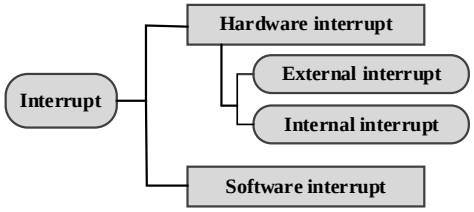
۱-مقدمه: بررسی وقفه‌ها (Interrupts) و نحوه عملکرد آن‌ها اهمیت زیادی دارد زیرا یکی از ویژگی‌های حیاتی میکروکنترلرها هستند که به آن‌ها اجازه می‌دهد تا به رویدادهای فوری واکنش نشان دهند و کارایی سیستم را افزایش دهند. وقفه به صورت یک سیگنال خاص تعریف می‌شود که به پردازنده اعلام می‌کند که نیاز به توجه فوری به یک رویداد خاص وجود دارد. در این حالت، جریان عادی برنامه متوقف می‌شود و پردازنده به سرویس روال وقفه (ISR) می‌رود.



شکل(۱): چرخه وقوع وقفه در میکروکنترلرهایSTM32

به دلایل مختلفی می‌توان از وقفه استفاده کرد. مهم‌ترین علت استفاده از وقفه واکنش به رویدادهای فوری مانند فشار دادن دکمه، دریافت داده از سنسور یا پایان کار یک تایمر است. همچنین مدیریت منابع جهت جلوگیری از اتلاف وقت در حال انتظار برای وقوع یک رویداد، از دیگر دلایل استفاده از وقفه است. دلیل دیگر افزایش کارایی به این منظور که پردازنده می‌تواند به کارهای دیگر بپردازد و تنها در صورت بروز رویداد خاص، به ISR مربوطه برود.

وقفه‌ها به طور کلی به دو نوع سخت‌افزاری و نرم‌افزاری تقسیم می‌شوند. وقفه‌های سخت‌افزاری خود به دو نوع وقفه داخلی و وقفه خارجی بخش‌بندی می‌شود.

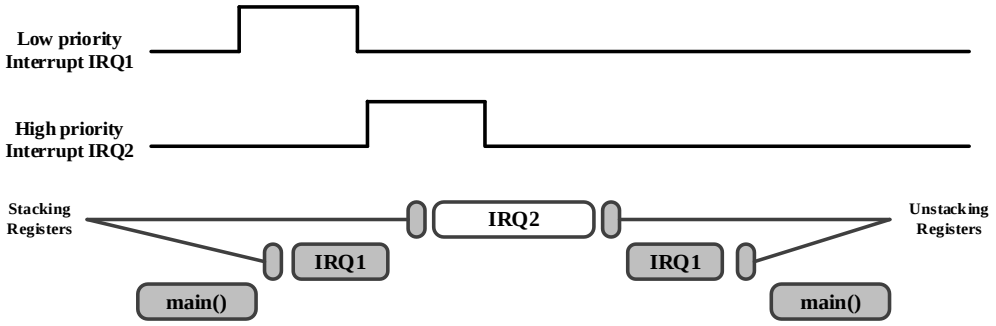


شکل(۲): انواع وقفه در میکروکنترلرهای STM32

وقفه‌های داخلی که ناشی از رویدادهایی که در خود میکروکنترلر رخ می‌دهد، مانند پایان یک تایمر یا خطای سخت‌افزاری است. نوع دوم، وقفه‌های خارجی که ناشی از رویدادهای خارجی مانند فشردن یک دکمه یا

دریافت سیگنال از حسگرها است. نوع سوم، وقفه‌های نرم‌افزاری که این نوع وقفه‌ها به‌صورت برنامه‌ریزی شده توسط نرم‌افزار ایجاد می‌شوند. به‌عنوان مثال، ممکن است برای درخواست منابع خاص یا عملیات خاصی استفاده شوند.

برای مدیریت وقفه‌ها و تعیین اولویت آن‌ها در هنگام وقوع از یک کنترل‌کننده وقفه NVIC یا Nested Vectored Interrupt Controller استفاده می‌شود. بر اساس NVIC هنگام رخ دادن یک وقفه با اولویت بالا، پردازنده وقفه با اولویت پایین را که حتی زودتر رخ داده باشد را در حالت انتظار نگاه می‌دارد و ابتدا وقفه با اولویت بالاتر را اجرا می‌کند و سپس وقفه با اولویت پایینتر را اجرا می‌کند.



شکل(۳): اجرای وقفه با اولویت بالاتر میان وقفه دیگر

مراحل عملکرد وقفه‌ها به‌طور کلی شامل موارد زیر است:

۱- تولید وقفه: یک سیگنال وقفه تولید می‌شود.

۲- تشخیص وقفه: میکروکنترلر با توجه به NVIC تشخیص می‌دهد کدامیک از وقفه‌ها فعال شده است.

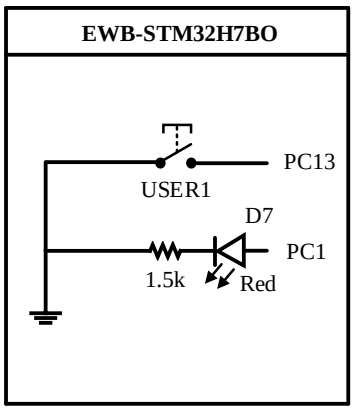
۳- انجام وظایف: وقتی که یک وقفه رخ می‌دهد، پردازنده به‌طور موقت برنامه اصلی را متوقف می‌کند و به ISR مراجعه می‌کند.

٤- مدیریت وضعیت: در صورتی که پردازنده در حال اجرای کدی باشد که غیرقابل وقفه است، ممکن است وقفه‌ها نادیده گرفته شوند تا بعداً اجرا شوند.

۵- بازگشت به برنامه اصلی: پس از اجرای ISR، میکروکنترلر به برنامه اصلی باز می‌گردد تا اجرای دستورالعمل‌های باقی‌مانده را ادامه دهد.

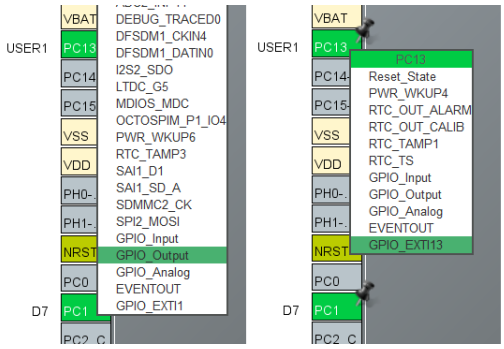
در ساختار میکروکنترلر، عملیات ورود و خروج از وقفه به‌صورت سخت‌افزاری مدیریت می‌شود. پردازنده به‌طور خودکار هنگام وقوع وقفه، اجرای برنامه اصلی را متوقف کرده و به روال سرویس‌دهی وقفه (ISR) منتقل می‌شود. در این فرآیند، برخی از رجیسترهای ضروری در استَک (stack – قسمتی از حافظه موقت RAM است که برای ذخیره‌سازی موقت مقادیر هنگام اجرای توابع و وقفه‌ها استفاده می‌شود) ذخیره شده و پس از اجرای ISR، بازیابی می‌شوند. اگرچه این مکانیزم سخت‌افزاری باعث افزایش سرعت پاسخگویی به وقفه‌ها می‌شود، اما همچنان هزینه پردازشی دارد و در سیستم‌های بلادرنگ باید بهینه مدیریت شود.

۲-شماتیک آزمایش: در این آزمایش نیز مانند آزمایش شماره یک از قطعات کلید USER1 متصل به پایه PC13 وD7 LED متصل به پایه PC1 استفاده می‌شود.



شکل(٤): شماتیک داخلی کلید و LED برد

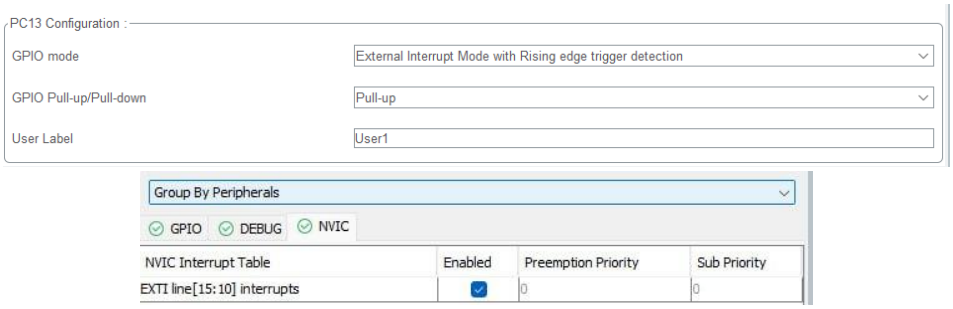
۳- تنظیمات نرم‌افزار STM32CubeIDE: طبق روال پیشین پروژه را ایجاد کنید. تنظیمات کلاک را تغییر ندهید و حالت دیباگ را بر روی Serial Wire قرار دهید. پایه PC1 را که به LED با عنوان D7 بر روی برد توسعه متصل است را به عنوان خروجی تنظیم کنید و سپس پایه PC13 را که به کلید USER1 در برد توسعه متصل است به عنوان پایه وقفه خارجی (GPIO_EXTI13) تنظیم کنید.



(ب) (الف)

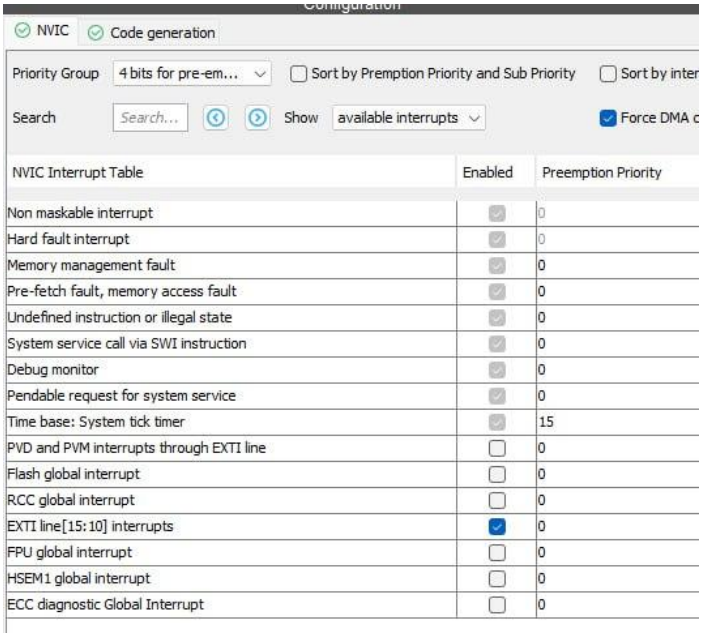
شکل(٥): (الف) تنظیم پایه PC13 به عنوان EXTI13 (ب) پایه PC1 (LED) در مُد خروجی

سپس از سربرگ GPIO پایه PC13 را بصورت زیر تنظیم کنید. پس از آن از سربرگ NVIC مد وقفه را برای این پایه فعال کنید.



شکل(٦): فعال‌سازی وقفه مربوط به EXTI13 با تریگر Rising در نرم‌افزار

در سربرگ NVIC بخش Configuration می‌توانید اولویت وقفه را تغییر دهید.



شکل(۷): جدول وقفه‌های NVIC در نرم‌افزار

پس از ذخیره تغییرات محیط کدنویسی اجرا می‌شود.

٤- معرفی توابع مورد استفاده: در این آزمایش از یک تابع برای فراخوانی وقفه استفاده شده است.

HAL_GPIO_EXTI_IRQHandler (uint16_t GPIO_Pin)

این تابع برای مدیریت درخواست وقفه EXTI استفاده می‌شود. زمانی که یک وقفه خارجی رخ می‌دهد، این تابع توسط HAL فراخوانی شده و پردازش اولیه وقفه را انجام می‌دهد. در نهایت، این تابع HAL_GPIO_EXTI_Callback را صدا می‌زند تا اقدامات لازم برای وقفه اجرا شود.

HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)

این تابع یک تابع Callback است که هنگام وقوع وقفه خارجی (EXTI) فراخوانی می‌شود. در این تابع می‌توان عملیات مربوط به پردازش وقفه را تعریف کرد. به عنوان مثال، بررسی می‌شود که کدام پین وقفه ایجاد کرده و متناسب با آن واکنش نشان داده شود.

٥- برنامه‌نویسی آزمایش: در فایل Main.c در ابتدای برنامه متغییر گلوبال Interrupt را برای ذخیره نوع عملکرد برنامه تعریف کنید.

```
/* USER CODE BEGIN PV */
// Interrupt flag to indicate an external event
uint8_t Interrupt = 0;
/* USER CODE END PV */
```

در main.c بین خطوط مجاز برای تعریف تابع، می‌توانید تابع روال وقفه را تعریف کنید و درون آن مشخص کنید که اگر وقفه رخ داد متغیر حالت وقفه که در ابتدای برنامه تعریف شده را تغییر مقدار بدهد.

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    // Check if the interrupt is from USER1 button
    if (GPIO_Pin == USER1_Pin)
        // Set the interrupt flag
        Interrupt = 1;
}
/* USER CODE END 4 */
```

در نهایت در حلقه while الگوریتم زیر را پیاده‌سازی کنید که در حالت پیش فرض LED با سرعت بالا چشمک بزند و هنگام وقوع وقفه ۵ بار با سرعت پایین تر چشمک بزند و سپس به حالت قبل بازگردد.

```
/* USER CODE BEGIN 3 */
   // Check if an interrupt has occurred
   if (Interrupt)
   {
        // Blink LED 5 times with 500ms delay
        for (int Cycle = 0; Cycle < 10; Cycle++)
        {
            // Toggle the D7 pin state
            HAL_GPIO_TogglePin(D7_GPIO_Port, D7_Pin);
            // Wait for 500 milliseconds
            HAL_Delay(500);
        }
        // Reset the interrupt flag after execution
        Interrupt = 0;
    }
    else
    {
        // Toggle the D7 pin state
        HAL_GPIO_TogglePin(D7_GPIO_Port, D7_Pin);
        // Wait for 100 milliseconds
        HAL_Delay(100);
    }
```

٦- تمرین: برنامه را طوری تغییر دهید که که کلید دارای سه وضعیت با قابلیت تکرار باشد. وضعیت اول چشمک زدن LED با بازه ۲۵۰میلی‌ثانیه، وضعیت دوم روشن شدن LED به طور دائم و وضعیت سوم خاموش شدن کامل LED است.

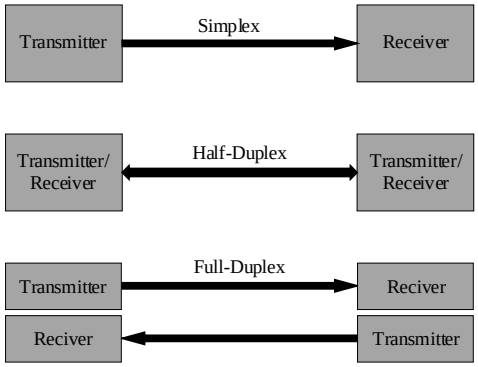
آزمایش ۴: راه‌اندازی رابط سریال USART

هدف آزمایش: در این آزمایش دانشجو با مفهوم ارتباط سریال و نحوه راه‌اندازی ارتباط USART آشنا می‌شود.

شرح آزمایش:

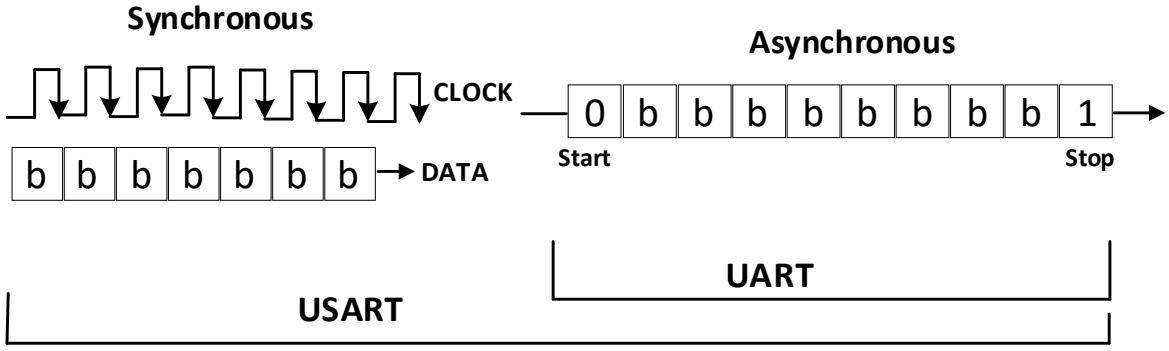
۱-مقدمه: پروتکل سریال USART (Universal Synchronous/Asynchronous Receiver/Transmitter) یکی از مهم‌ترین و پرکاربردترین روش‌ها برای انتقال داده‌ها بین دستگاه‌های الکترونیکی به شمار می‌آید. در دنیای امروزی، ارتباطات بین تجهیزات الکترونیکی نقش بسیار حیاتی دارد و نیاز به استانداردهایی کارآمد و قابل اعتماد برای این منظور احساس می‌شود. کاربردهای USART شامل ارتباط با سنسورها، تبادل داده‌ها بین میکروکنترلرها و ارسال دستورات به ماژول‌های مختلف مانند GPS یا GSM است. استفاده از USART در این سیستم‌ها به دلیل سادگی پیاده‌سازی، صرفه‌جویی در منابع و انعطاف‌پذیری بالا، باعث محبوبیت این پروتکل در بین طراحان و توسعه‌دهندگان شده است.

پروتکل USART قادر به کار در دو حالت همزمان- سنکرون (Synchronous) و غیر همزمان- آسنکرون (Asynchronous) است که هر یک ویژگی‌ها و مزایای خاص خود را دارند. قابلیت پشتیبانی از این دو حالت نه‌تنها دامنه کاربرد این پروتکل را گسترش می‌دهد، بلکه امکان تنظیمات دقیق‌تر و بهینه‌تر بر اساس نیازهای خاص هر پروژه را نیز فراهم می‌آورد. در حالت سنکرون(USART)، داده‌ها همراه با سیگنال کلاک ارسال می‌شوند. این نوع ارتباط معمولاً سریع‌تر بوده و برای انتقال داده در فواصل کوتاه که نیاز به تأخیر کم دارد، مناسب است. در حالت آسنکرون(UART)، ارسال داده بدون سیگنال کلاک انجام می‌شود. داده‌ها به‌صورت بسته‌های خاصی ارسال شده و با برخی بیت‌های کنترلی، مانند بیت‌های استاپ و پریتی، مشخص می‌شوند. USART یک ارتباط سریال دو طرفه و full-duplex است، به این معنی که خط دریافت و ارسال اطلاعات از هم جدا هستند و می‌توان در صورت نیاز هر دو عمل را باهم انجام داد.



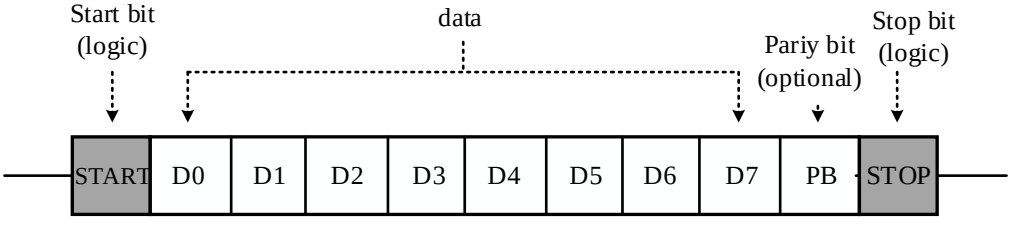
شکل(۱): انواع حالت های ارتباط USART

در ارتباط آسنکرون که سیگنال کلاک ارسال نمی‌شود، هر دو سمت ارتباط باید از یک نرخ ارسال داده مشخص و برابر استفاده کنند. به این نرخ، Baud Rate گفته می‌شود. بادریت (Baud Rate) به معنای تعداد سیگنال‌های انتقال یافته در واحد زمان است و به طور خاص، در ارتباطات سریال، نشان‌دهنده تعداد بیت‌هایی است که در هر ثانیه منتقل می‌شوند. به عبارتی ساده‌تر، بادریت سرعت انتقال داده‌ها را تعیین می‌کند. برای مثال، اگر بادریت یک ارتباط ۹٦۰۰ باشد، به این معناست که می‌توانید ۹٦۰۰ بیت داده را در هر ثانیه ارسال کنید. این مقدار می‌تواند بسته به نیازهای سیستم یا پروتکل ارتباطی تنظیم شود و معمولاً در مقادیر متداولی مانند ۱۲۰۰، ۲٤۰۰، ٤۸۰۰، ۹٦۰۰، ۱۱۵۲۰۰ و ... وجود دارد. البته در حالت آسنکرون ممکن است مشکلاتی همچون هم‌زمان نبودن ساعت‌ها یا خطاهای انتقال به وجود آید که نیاز به استفاده از روش‌هایی مانند CRC یا تایید دریافت را ضروری می‌سازد.



شکل(۲): تفاوت ارتباط سنکرون و آسنکرون USART

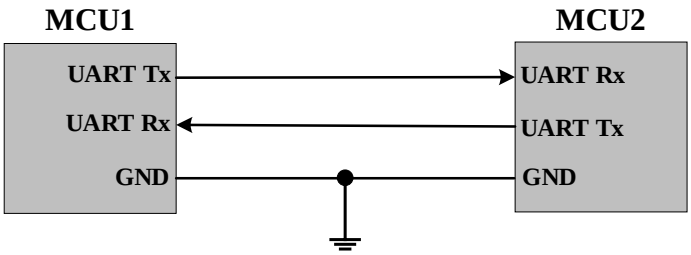
بسته داده در پروتکل USART شامل چند بخش اصلی است: ابتدا یک بیت Start وجود دارد که نشان‌دهنده‌ی شروع انتقال داده‌ها است و به هماهنگی فرستنده و گیرنده کمک می‌کند. پس از آن، بیت‌های داده قرار می‌گیرند که شامل اطلاعات واقعی هستند و معمولاً بین ۵ تا ۹ بیت طول دارند. همچنین، یک بیت پریتی (بیت توازن یا بیت مقایسه) برای تشخیص خطا در داده‌ها وجود دارد که می‌تواند به روش‌های مختلفی به صورت اختیاری تنظیم شود. در نهایت، بیت‌های Stop، پایان انتقال را نشان می‌دهند و به فرستنده و گیرنده اجازه می‌دهند تا آماده‌ی دریافت داده‌های بعدی شوند. این ساختار باعث می‌شود که ارتباطات سریال با دقت و اطمینان انجام شوند.



شکل(۳): قالب ارسال داده در USART

پیکربندی USART شامل تنظیمات اولیه‌ای است که برای برقراری ارتباط مؤثر باید مورد توجه قرار گیرد. اولین مورد، Baud Rate است که نمایانگر سرعت انتقال داده‌ها به بیت بر ثانیه است. پس از آن، تعداد بیت‌های داده معمولاً ۸ یا ۹ بیت انتخاب می‌شود، که بستگی به نیازهای خاص ارتباط دارد. همچنین، نوع پریتی باید مشخص شود که می‌تواند none، even یا odd باشد تا از بروز خطا در انتقال داده‌ها جلوگیری شود. در نهایت، تعداد بیت‌های Stop که عموماً یک یا دو است، باید تعیین گردد تا مشخص شود که انتهای هر داده چگونه شناسایی شود. تنظیم این پارامترها به برقراری ارتباطی روان و بدون خطا کمک می‌کند.

در هر ارتباط USART دو طرفه یا full-duplex، به حداقل دو پایه یا سیم (به‌جز سیم کلاک در حالت سنکرون) شامل یک سیم برای دریافت اطلاعات (RX) و یک سیم برای ارسال اطلاعات (TX) نیاز است. در حالت عادی USART اطلاعات به‌صورت سریال، توسط این دو پین ارسال و دریافت می‌شوند. درصورتی که ارتباط به صورت سنکرون باشد، یک پایه یا سیم CLK نیز نیاز است.

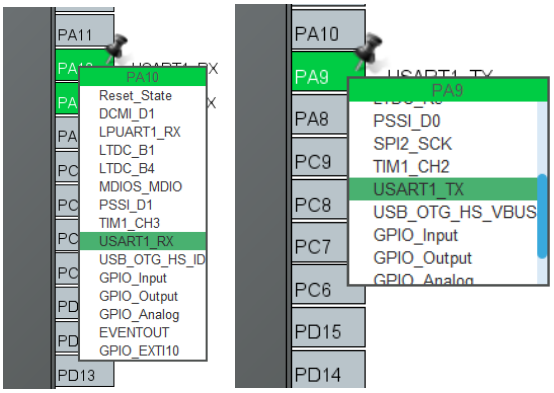


شکل(٤): نحوه اتصال ارتباط UART بین دو دستگاه

مدیریت وقفه‌ها در USART ویژگی کلیدی است که امکان دریافت داده‌ها را بدون نیاز به چک کردن مداوم وضعیت پورت USART فراهم می‌کند. با این حال، عیب‌یابی در این زمینه از اهمیت بالایی برخوردار است و شناسایی و رفع عیوبی نظیر هم‌زمانی نامناسب Baud Rate، اتصالات نامناسب سخت‌افزاری و اختلالات در خط ارتباطی بسیار ضروری است. به طور کلی، پروتکل USART به‌دلیل سادگی و عملکرد قابل اعتماد، در بسیاری از پروژه‌های الکترونیکی و برنامه‌نویسی کاربرد دارد. استفاده‌ی مناسب از USART و تنظیمات صحیح می‌تواند ارتباط موثری بین میکروکنترلرها و دیگر دستگاه‌ها برقرار کند.

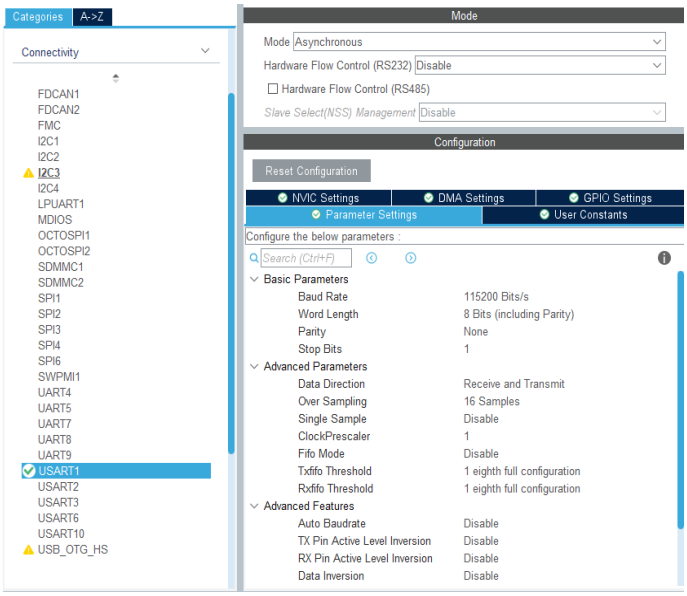
۲-شماتیک آزمایش: در این آزمایش با استفاده از یک کابل شارژ اندروید Micro USB خروجی سریال برد را مستقیما به کامپیوتر متصل کنید (مورد ۱ در معرفی برد EWB-STM32H7B0). لازم به ذکر است که برای این آزمایش نصب درایور CP210x بر روی سیستم مورد استفاده الزامی است.

۳- تنظیمات نرم‌افزار STM32CubeIDE: مانند آزمایش‌های پیشین فایل پروژه را ایجاد کنید. کانفیگ منابع کلاک را بصورت پیش فرض قرار دهید و پروگرامر را به صورت SWD فعال کنید سپس پایه‌های PA9 و PA10 را به عنوان USART1_TX و USART1_RX تنظیم کنید.



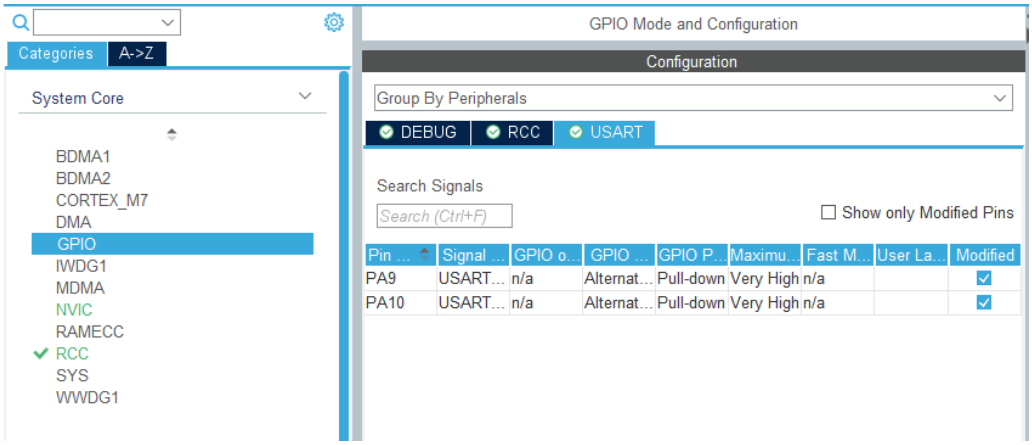
شکل(٥): تنظیم پین‌های PA10 و PA9 به عنوان USART1 RX و USART1 TX

از سربرگ Connectivity زیر شاخه USART1 را انتخاب کنید و تنظیمات زیر را اعمال کنید.



شکل(٦): تنظیمات ارتباط USART در نرم‌افزار CubeMx

از سربرگ System Core زیرشاخه GPIO را انتخاب کنید و تنظیمات زیر را برای پین‌های USART اعمال کنید.



شکل(٧): تنظیم پایه‌های ارتباط UART در واحد GPIO

در انتها تنظیمات را generate کنید.

٤- معرفی توابع مورد استفاده: در این آزمایش لازم است که سه تابع توضیح داده شود.

sprintf (char * str, const char * format, ...)

این تابع در زبان C برای قالب‌بندی رشته‌ها و ترکیب متن با مقادیر متغیرها، اعداد و داده‌های دیگر استفاده می‌شود. آرگومان اول آدرس یک رشته (بافر مقصد) است که نتیجه در آن ذخیره می‌شود. آرگومان دوم رشته قالب است که تعیین می‌کند داده‌های بعدی چگونه در متن قرار گیرند.

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

از این تابع برای فرستادن اطلاعات از طریق UART استفاده می‌شود. آرگومان اول به ساختار UART مورد استفاده اشاره می‌کند. آرگومان دوم بافر داده‌ای است که اطلاعات آن ارسال می‌شود. آرگومان سوم تعداد بایت داده و آرگومان چهارم حداکثر مدت زمان مجاز برای ارسال داده را مشخص می‌کند.

HAL_UART_Receive (UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)

از این تابع برای دریافت اطلاعات از طریق UART استفاده می‌شود. آرگومان اول به ساختار UART مورد استفاده اشاره می‌کند. آرگومان دوم بافر داده‌ای است که اطلاعات در آن قرار می‌گیرد. آرگومان سوم تعداد بایت داده و آرگومان چهارم حداکثر مدت زمان مجاز برای ارسال داده را مشخص می‌کند.

٥- برنامه‌نویسی آزمایش: کتابخانه stdio.h را اضافه کنید.

```
/* USER CODE BEGIN Includes */
// Include standard input/output library
#include <stdio.h>
/* USER CODE END Includes */
```

متغییر آرایه پیام و شمارنده را تعریف کنید.

```
/* USER CODE BEGIN 1 */
// Define buffer for message and variable X
uint8_t MSG[35] = {'\0'};
uint8_t X = 0;
/* USER CODE END 1 */
```

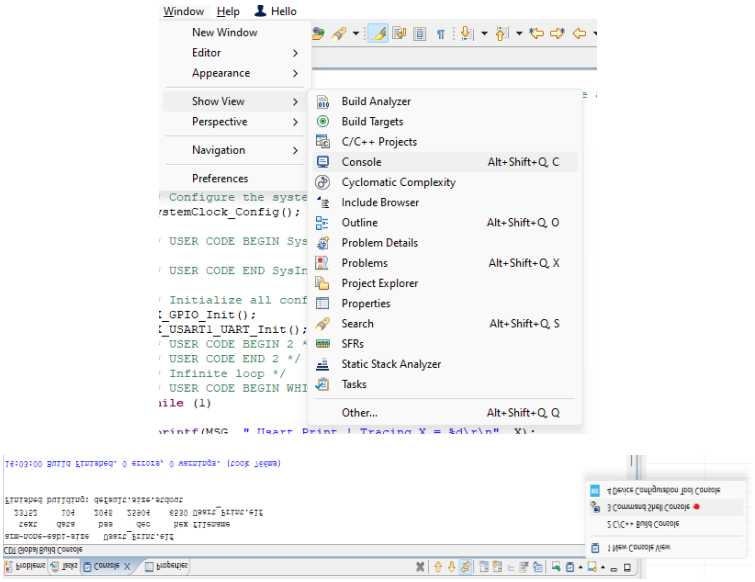
درون حلقه بینهایت خطوط زیر را بنویسید.

```
  while (1)
  {
      /* USER CODE END WHILE */
      // Format the message with current value of X
      sprintf(MSG, " Usart Print ! Tracing X = %d\r\n", X);
      // Transmit the formatted message via UART
      HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
      // Wait for 500 ms
```

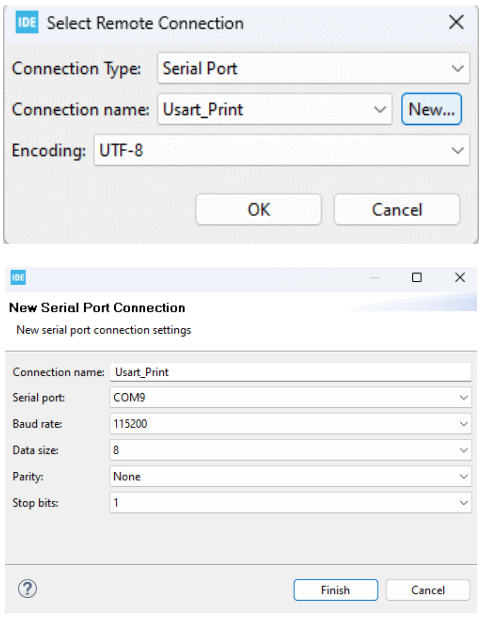
```
        HAL_Delay(500);
        // Increment X for next transmission
        X++;
        /* USER CODE BEGIN 3 */
    }
```

در داخل حلقه ابتدا مقدار شمارنده حلقه در متن جایگذاری شده و سپس کل متن توسط تابع sprintf در متغییر MSG بارگذاری می‌شود. در خط بعدی محتویات آرایه MSG توسط USART ارسال می‌شود. برای دریافت داده ها در سمت کامپیوتر در نرم افزار CubeIDE مراحل زیر را طی کنید:

1. Window > Show View > Console
2. Menu bar > NEW icon > Command Shell console > Connection type: Serial port > Set Baud Rate & Connection Name > Encoding: UTF-8



شکل(۸): نحوه دسترسی به Console و Command Shel Console

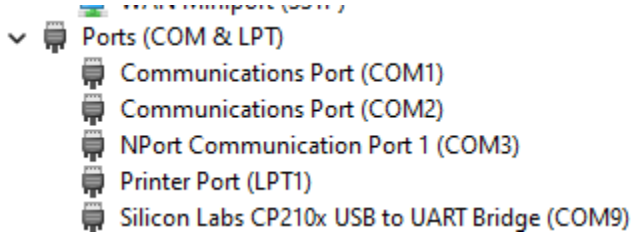


شکل(۹): تنظیمات مربوط به پورت سریال

و گزینه Finish را انتخاب کنید. برای اطلاع از شماره پورت سریال فعال مراحل زیر را طی کنید:

۱- از My computer به قسمت Device Manager بروید.

۲- زیرشاخه Ports خط آخر که با silicon Labs شروع شده است را بیابید و شماره پورت COM را یادداشت کنید.



شکل(۱۰): بررسی پورت COM متصل‌شده جهت ارتباط با برد

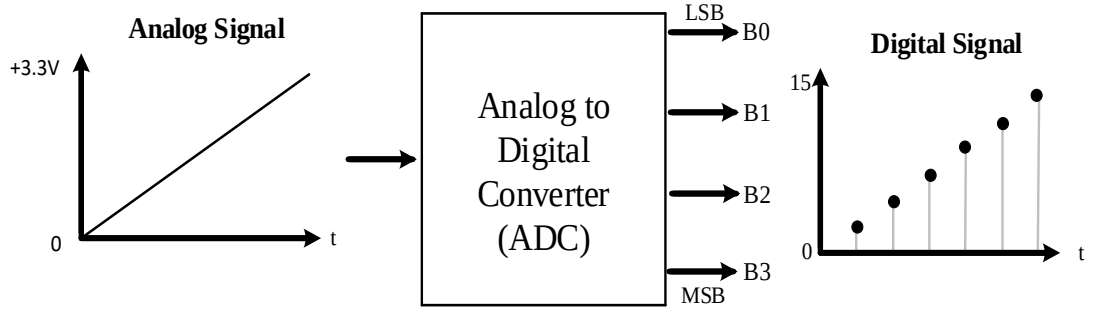
٦- تمرین: برنامه‌ای بنویسید که با هر بار فشردن کلید USER1 یک پیام از طریق USART ارسال شود. در این برنامه از وقفه استفاده کنید.

آزمایش ۵: راه‌اندازی واحد ADC

هدف آزمایش: در این آزمایش دانشجو با مفاهیم ADC و نحوه راه اندازی واحد آن آشنا می‌شود.

شرح آزمایش:

۱-مقدمه: در حالت کلی دنیای خارج از میکروکنترلر ماهیت آنالوگ یا پیوسته دارد. مبدل‌های آنالوگ به دیجیتال (ADC) و دیجیتال به آنالوگ (DAC) امکان ارتباط میکروکنترلر با دنیای خارج از میکروکنترلر را فراهم می‌کنند. واحد ADC (مبدل آنالوگ به دیجیتال) در میکروکنترلرهای STM32 به عنوان یک بخش کلیدی برای تبدیل سیگنال‌های آنالوگ به مقادیر دیجیتال است و با قابلیت‌های متنوعی که دارد به میکروکنترلر این امکان را می‌دهد تا با دنیای خارج از خود ارتباط برقرار کند.



شکل(۱): مبدل آنالوگ به دیجیتال با چهاربیت دقت

در اینجا به برخی از ویژگی‌ها و عملکردهای ADC در این میکروکنترلرها اشاره می‌شود:

۱-رزولوشن: میکروکنترلرهای STM32 عموماً دارای ADC با رزولوشن ۱۲ بیت هستند. این بدین معناست که خروجی دیجیتال ٤٠٩٦ ($۲^{۱۲}$) سطح مختلف را می‌تواند نمایان کند. این رزولوشن به شما اجازه می‌دهد تا اختلافات کوچک در سیگنال‌های آنالوگ را با دقت بیشتری اندازه‌گیری کنید.

۲- کانال‌های ورودی: ADC در STM32 معمولاً شامل ۱۸ کانال است، که ١٦ کانال خارجی و ۲ کانال داخلی برای نظارت بر ولتاژهای داخلی هستند. این ویژگی به شما این امکان را می‌دهد تا چندین منبع سیگنال را به‌طور همزمان یا به‌صورت متناوب نمونه‌برداری کنید.

۳- روش کار: واحد ADC در STM32 از روش تقریب متوالی (SAR) برای تبدیل سیگنال آنالوگ به دیجیتال استفاده می‌کند. در این روش، ADC ولتاژ ورودی را با ولتاژ مرجع مقایسه کرده و به تدریج سطح ولتاژ را به یک عدد باینری تبدیل می‌کند. در این روش مقدار دیجیتالی تولید شده، متناسب با ولتاژ مرجع اعمال شده به

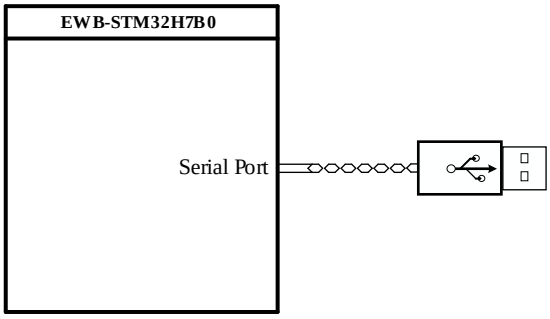
بلوک ADC است. رابطه مقدار دیجیتال خروجی متناسب با ولتاژ آنالوگ ورودی را می‌توان با رابطه زیر نشان داد.

$$\frac{V_{in}}{V_{ref}} = \frac{Digital\ Value}{2^n - 1}$$

٤- سرعت نمونه‌برداری: ADCهای STM32 می‌توانند با سرعت‌های نمونه‌برداری بسیار بالا (تا چند مگاهرتز) کار کنند. این ویژگی برای کاربردهای زمان واقعی و اندازه‌گیری سیگنال‌های متغیر بسیار مهم است.

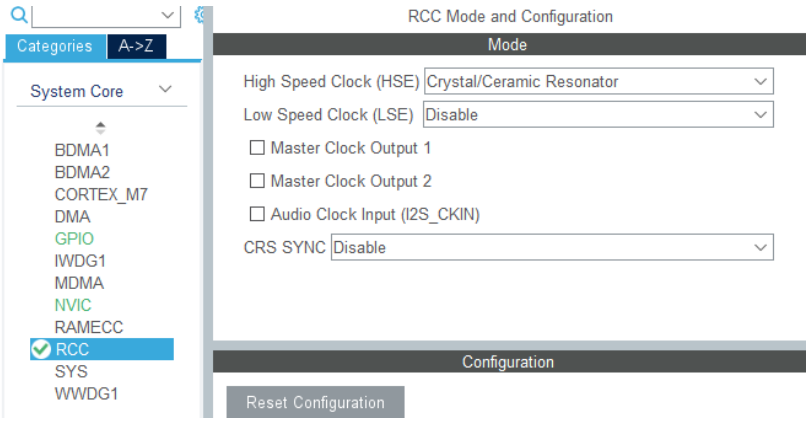
با این ویژگی‌ها، واحد ADC در STM32 به یکی از ابزارهای مهم برای پروژه‌های الکترونیکی و کنترل صنعتی تبدیل شده است.

۲-شماتیک آزمایش: در این آزمایش از یک پتانسیومتر برای ساخت یک ولتاژ متغیر استفاده شده است. این ولتاژ به پایه وردی PB0 اعمال می‌شود. سپس داده‌های دریافتی توسط ارتباط سریال به کامپیوتر ارسال می‌شود.



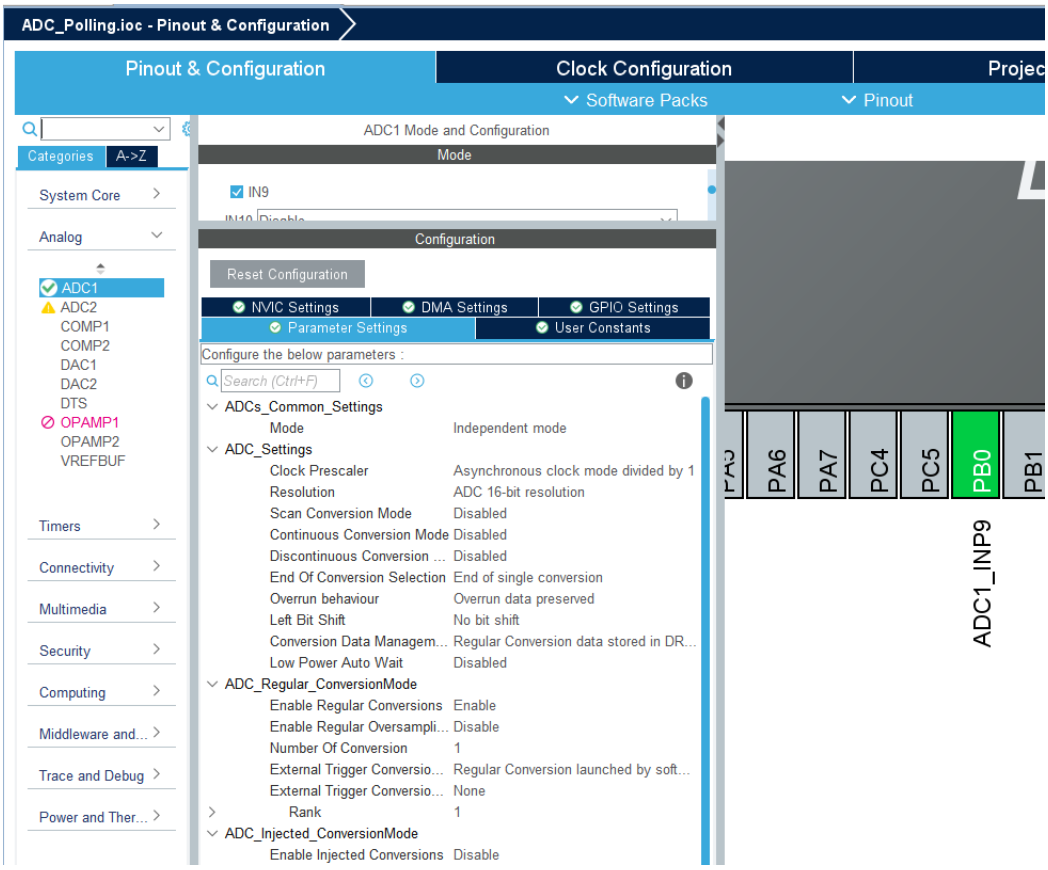
شکل(۲): شماتیک استفاده شده

۳- تنظیمات نرم‌افزار STM32CubeIDE: پروژه را مانند قبل ایجاد کنید و همچنین واحد UART را برای ارسال داده فعال کنید. تنظیمات کلاک را نیز مطابق تصویر زیر تنظیم کنید تا از کریستال خارجی به عنوان منبع کلاک استفاده شود.



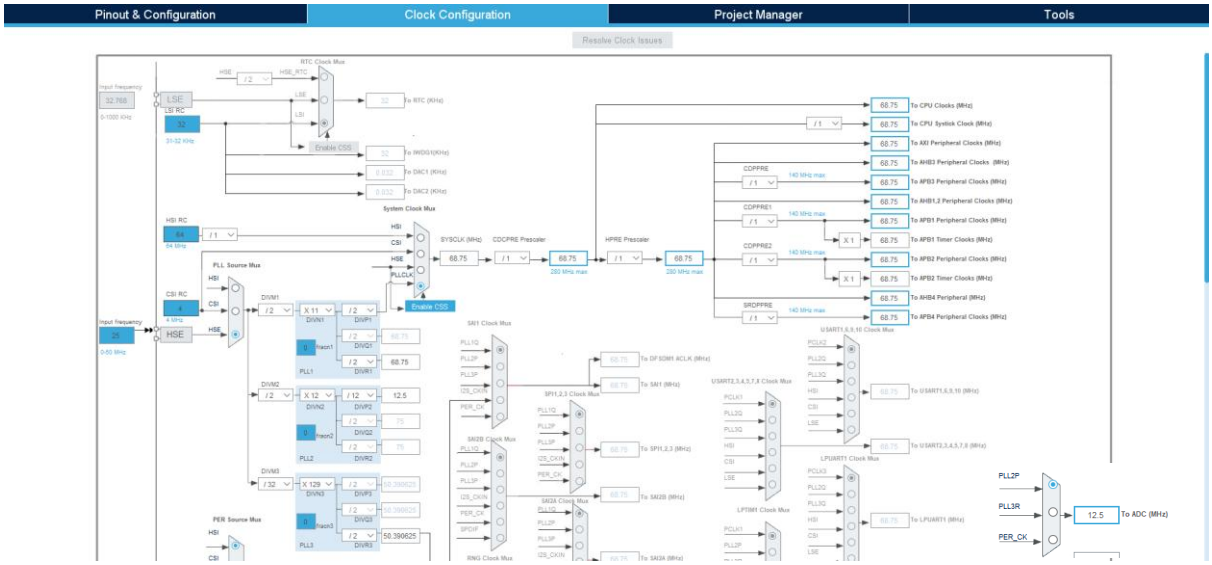
شکل(۳): تنظیم واحد RCC بر روی کلاک خارجی

پس از آن واحد ADC1 را از منوی سمت چپ در زیرشاخه Analog پیدا کرده و تنظیمات زیر را اعمال کنید.



شکل(٤): تنظیمات واحد ADC در نرم‌افزار

تنظیمات کلاک سیستم را مطابق شکل زیر تنظیم کنید.



شکل(۵): تنظیمات کلاک میکروکنترلر در نرم‌افزارCubeMx

پس از اعمال تنظیمات تغییرات را ذخیره و کد را ایجاد کنید.

٤- معرفی توابع مورد استفاده: در این آزمایش ٤ تابع استفاده شده که به معرفی آن‌ها پراخته می‌شود.

HAL_ADCEx_Calibration_Start (ADC_HandleTypeDef *hadc, uint32_t SingleDiff)

این تابع یک کالیبراسون خودکار است. آرگومان اول اشاره به کانال مورد نظر ADC دارد. آرگومان دوم اشاره به نوع کالیبره دارد که می‌توان کانال به صورت ورودی تک سر و یا تفاضلی باشد. در ADC، کانال‌های ورودی می‌توانند به دو صورت تک سر (Single-ended) و تفاضلی (Differential) پیکربندی شوند. در کانال تک سر که اولین نوع پیکربندی است که تنها یک سیگنال آنالوگ نسبت به مرجع زمین (GND) اندازه‌گیری می‌شود. این روش برای اندازه‌گیری سیگنال‌های آنالوگ مثبت مناسب است و از آنجایی که پیکربندی ساده‌تری دارد، در بسیاری از برنامه‌های عمومی مورد استفاده قرار می‌گیرد. در کانال تفاضلی ADC به اندازه‌گیری اختلاف ولتاژ بین دو ورودی آنالوگ می‌پردازد. در این پیکربندی، ولتاژ ورودی‌ها نسبت به یکدیگر اندازه‌گیری می‌شود. این روش برای اندازه‌گیری سیگنال‌هایی که ممکن است تحت تأثیر نویز الکتریکی قرار گیرند، بسیار مناسب است و می‌تواند دقت اندازه‌گیری را بهبود بخشد. از این رو، معمولاً برای سنجش سیگنال‌های کوچک، خصوصاً در محیط‌های پر سر و صدا، از این نوع پیکربندی استفاده می‌شود. انتخاب بین این دو نوع پیکربندی بستگی به نوع سیگنال و شرایط محیطی دارد و می‌تواند تأثیر زیادی بر بهبود دقت اندازه‌گیری‌ها داشته باشد.

HAL_ADC_Start (ADC_HandleTypeDef *hadc)

این تابع برای فعال کردن ADC به کار می‌رود. آرگومان این تابع به ADC مورد نظر برای فعال کردن اشاره دارد. لازم به ذکر است که با تابع HAL_ADC_Stop می‌توان ADC را غیر فعال کرد.

HAL_ADC_PollForConversion (ADC_HandleTypeDef *hadc, uint32_t Timeout)

با اجرای این تابع در صورتی که واحد ADC هنوز عمل تبدیل دیجیتال به آنالوگ را به طور کامل انجام نداده باشد، میکروکنترلر منتظر خواهد ماند تا این فرآیند کامل شود. آرگومان اول این تابع به کانال مورد نظر اشاره دارد و آرگومان دوم حداکثر زمان انتظار را تعیین می‌کند.

HAL_ADC_GetValue (ADC_HandleTypeDef *hadc)

این تابع برای دریافت دیتا تبدیل شده استفاده می‌شود. آرگومان این تابع به ADC مورد نظر برای دریافت دیتا اشاره دارد.

٥- برنامه‌نویسی آزمایش: متغیرهای مقدار ADC و آرایه پیام UART را تعریف کنید.

```
/* USER CODE BEGIN 1 */
      // Variable to store ADC result
      uint16_t ADC_RES = 0;
```

```
      // Array to store message as a string
      uint8_t MSG[35] = {'\0'};
/* USER CODE END 1 */
```

ADC را برای دقت بیشتر کالیبره کنید.

```
/* USER CODE BEGIN 2 */
  // Calibrate The ADC On Power-Up For Better Accuracy
    HAL_ADCEx_Calibration_Start(&hadc1, ADC_CALIB_OFFSET, ADC_SINGLE_ENDED);
/* USER CODE END 2 */
```

در مرحله بعدی تبدیل ADC را شروع کنید و با نرخ به روز سازی ۱ ثانیه آن را تنظیم کنید، در نهایت مقدار را در متغیر AD_RES وارد کنید و بر روی UART ارسال کنید.

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

    /* USER CODE BEGIN 3 */
        //Start ADC Conversion
        HAL_ADC_Start(&hadc1);
        // Poll ADC Peripheral & Timeout = 10mSec
        HAL_ADC_PollForConversion(&hadc1, 10);
        //Read ADC Conversion Result
        ADC_RES = HAL_ADC_GetValue(&hadc1);
        // Delay for 1 millisecond to ensure stability
        HAL_Delay(1);
        //Send ADC Value on USART Port
        sprintf(MSG, "ADC Value = %d\r\n", ADC_RES);
        HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
          // Delay for 500 milliseconds before next reading
        HAL_Delay(1000);
  }
  /* USER CODE END 3 */
```

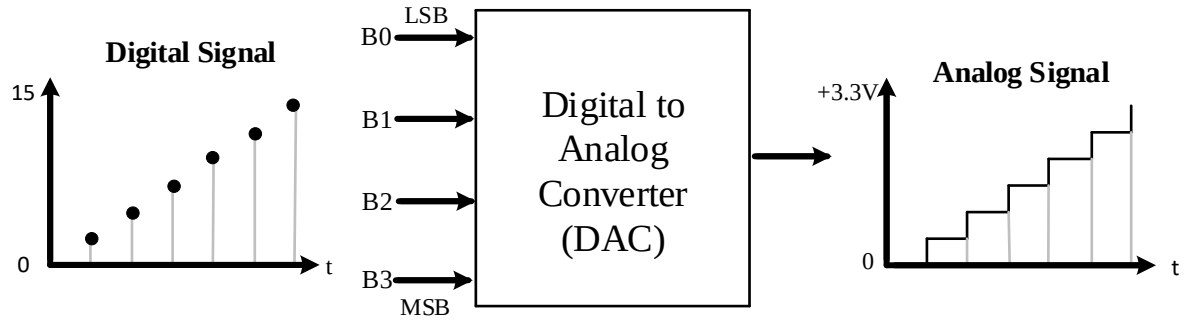
٦- تمرین: برنامه را طوری تغییر دهید که ولتاژ متغییر ایجاد شده توسط پتانسیومتر بر روی پورت سریال ارسال شود.

آزمایش ۶: راه‌اندازی واحد DAC

هدف آزمایش: در این آزمایش دانشجو با واحد DAC و نحوه راه‌اندازی آن آشنا می‌شود.

شرح آزمایش:

۱-مقدمه: واحد DAC (مبدل دیجیتال به آنالوگ) در میکروکنترلرهای STM32 برای تبدیل مقادیر دیجیتال به سیگنال‌های آنالوگ استفاده می‌شود. این واحد به شما این امکان را می‌دهد که از میکروکنترلر برای تولید سیگنال‌های آنالوگ دقیق بهره‌برداری کنید. واحد مبدل دیجیتال به آنالوگ یک عدد در ورودی دریافت می‌کند و در خروجی سطح ولتاژی متناسب با عدد دریافتی در ورودی ایجاد می‌کند. عمل DAC را می‌توان عکس عمل ADC بیان کرد.



شکل(۱): مبدل دیجیتال به آنالوگ با ٤بیت دقت

واحد DAC در میکروکنترلرهای STM32 به عنوان ابزاری مفید و انعطاف‌پذیر برای تولید سیگنال‌های آنالوگ در کاربردهایی مانند: ۱- تولید سیگنال‌های صوتی ۲- کنترل موتورها ۳- تولید ولتاژ مرجع ٤- سیستم‌های مخابراتی و ... استفاده می‌شود.

در ادامه به برخی از ویژگی‌ها و عملکردهای DAC در این میکروکنترلرها اشاره می‌شود:

۱- رزولوشن: DAC در STM32 معمولاً دارای ۱۲ بیت رزولوشن است. این بدین معناست که می‌تواند ٤۰۹٦ ($۲^{۱۲}$) سطح مختلف ولتاژ را تولید کند. هرچه رزولوشن بالاتر باشد، دقت سیگنال آنالوگ بهتر خواهد بود. برای بدست آوردن حداقل ولتاژ خروجی تولید شده توسط DAC، باید Digital Value را در فرمول زیر برابر با عدد ۱ قرار بدهید.

$$Vout = \frac{Vref}{2^n - 1} \times Digitla\ Value$$

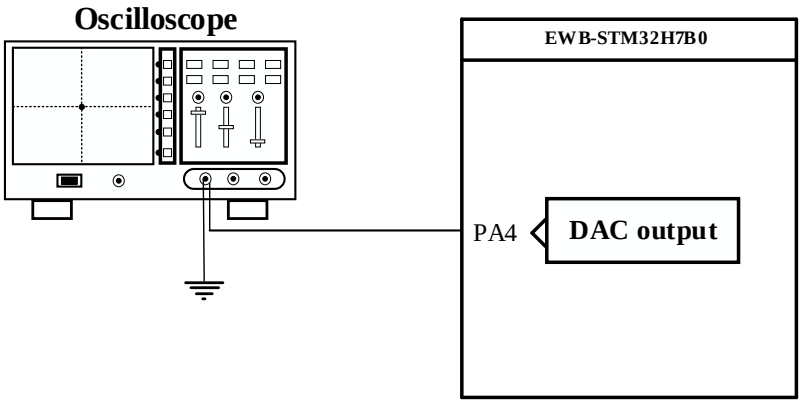
۲- کانال‌های خروجی: مدل‌های مختلف میکروکنترلر STM32 معمولاً دارای یک یا چند کانال DAC هستند. به عنوان مثال، مدل‌های با چند کانال می‌توانند به شما این امکان را بدهند که چندین سیگنال آنالوگ را به‌طور همزمان تولید کنید.

۳- روش‌های کار: در میکروکنترلرهای STM32، DAC داخلی از شبکه مقاومتی R-2R ladder برای تبدیل مقدار دیجیتال به ولتاژ آنالوگ استفاده می‌کند. این روش شامل یک آرایه مقاومت‌های R و 2R است که با تقسیم ولتاژهای ورودی، خروجی آنالوگ ایجاد می‌کند. از آنجا که این مدار سخت‌افزاری است، خروجی تولیدشده پیوسته و دقیق‌تر از تکنیک‌های شبه‌آنالوگ است.

٤- سرعت خروجی: DAC در STM32 می‌تواند با سرعت‌های مختلف عمل کند، که معمولاً به مقادیر دیجیتال ورودی و تنظیمات آن بستگی دارد. این قابلیت به‌خصوص در کاربردهای بلادرنگ و تولید سیگنال‌های صوتی مهم است.

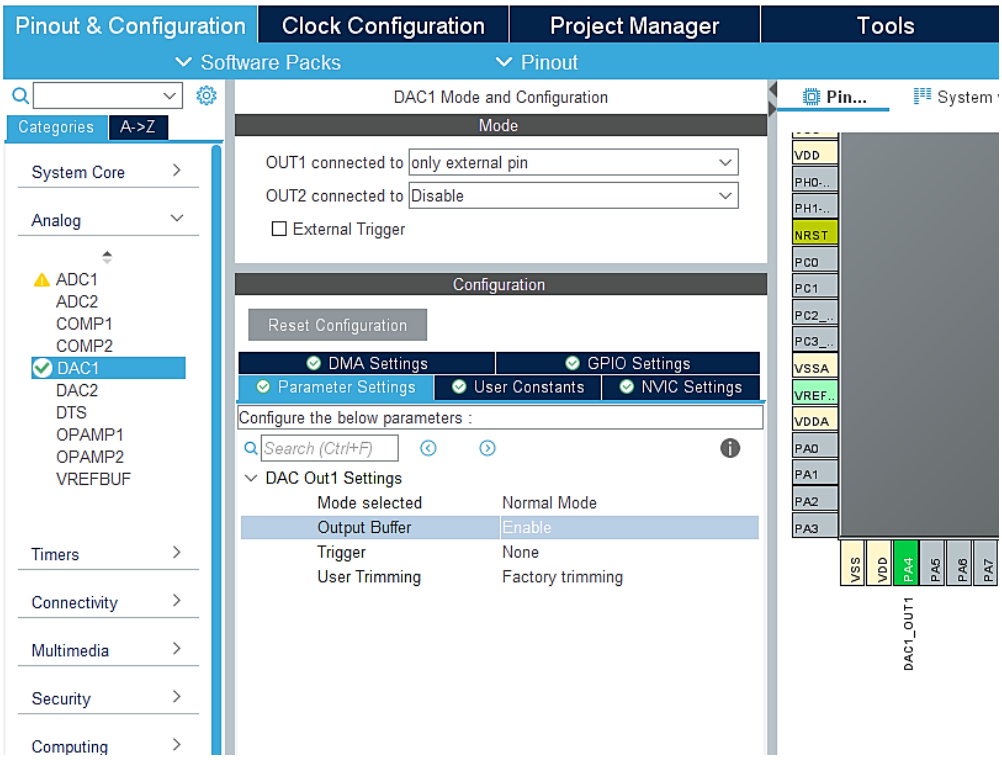
۵- تنظیم ولتاژ مرجع: DAC معمولاً امکان تنظیم ولتاژ مرجع را دارد، که این به شما این امکان را می‌دهد تا دامنه خروجی سیگنال آنالوگ را تنظیم کنید. ولتاژ مرجع در میکروکنترلرهای STM32 توسط تغذیه بخش مبدل آنالوگ تعیین می‌شود. این تغذیه با نام‌های VDDA و GNDA شناخته می‌شود و قابلیت اعمال ۱.۸ تا ۳.۳ولت به آن وجود دارد. در بعضی دیگر پایه‌ای به نام VREF نیز وجود دارد که امکان کاربردهایی نظیر تنظیم ولتاژ مرجع را به کاربر می‌دهد.

۲-شماتیک آزمایش: در این آزمایش شکل موجی با ٤ سطح ولتاژ مختلف بر روی پایه خروجی DAC تولید می‌شود. کافی است پایه خروجی PA4 را به اسیلوسکوپ متصل کنید تا شکل موج ایجاد شده را مشاهده کنید.



شکل(۲): شماتیک اتصال خروجی DAC به اسیلوسکوپ

۳- تنظیمات نرم‌افزار STM32CubeIDE: پروژه را مانند قبل ایجاد کنید و تغییری در کلاک سیستم ایجاد نکنید. از سربرگ Analog گزینه DAC1 را بیابید و در بخش گشوده شده تنظیمات زیر را اعمال کنید. راه‌اندازی بخش DAC تنظیمات زیادی ندارد و پس از اعمال تنظیمات مشخص شده آن را ذخیره و کد را ایجاد کنید.



شکل(۳): تنظیمات واحد DAC در نرم‌افزار

٤- معرفی توابع مورد استفاده:

HAL_DAC_Start (DAC_HandleTypeDef *hdac, uint32_t Channel)

تابع HAL_DAC_Start برای فعال‌سازی DAC در میکروکنترلرهای STM32 استفاده می‌شود. آرگومان اول به ساختار DAC_HandleTypeDef اشاره دارد که شامل تنظیمات پیکربندی DAC است و آرگومان دوم کانال انتخابی (مانند کانال ۱ یا ۲) برای شروع به کار DAC را مشخص می‌کند. پس از فراخوانی این تابع، DAC آماده ارسال سیگنال‌های آنالوگ از کانال انتخابی می‌شود.

HAL_DAC_SetValue (DAC_HandleTypeDef *hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)

از این تابع جهت تنظیم مقدار مشخص شده برای کانال DAC تنظیم می‌شود. آرگومان اول مانند تابع قبل به یک ساختار DAC_HandleTypeDef اشاره دارد. آرگومان دوم کانال DAC و آرگومان سوم مقدار داده را مشخص می‌کند. آرگومان چهارم دامنه داده را مشخص می‌کند. پارامتر چهارم فرمت داده وارد شده به DAC است که مطابق گزینه‌های زیر مشخص می‌شود.

DAC_ALIGN_8B_R: ترازبندی به راست برای داده ۸ بیتی یعنی اگر عدد ۸ بیتی باشد، این عدد در سمت راست رجیستر قرار می‌گیرد و سمت چپ رجیستر (بیت‌های باقی‌مانده) با صفر پر می‌شوند. (برای مثال (0000000011010101

DAC_ALIGN_12B_L: ترازبندی به چپ برای داده ۱۲ بیتی یعنی اگر عدد ۱۲ بیتی باشد، این عدد در سمت چپ رجیستر قرار می‌گیرد و سمت راست رجیستر (بیت‌های کم‌ارزش‌تر) با صفر پر می‌شوند. (مثال 1101010101010000)

DAC_ALIGN_12B_R: ترازبندی به راست برای داده ۱۲ بیتی یعنی در این حالت، همان داده ۱۲ بیتی به سمت راست رجیستر انتقال می‌یابد و سمت چپ رجیستر با صفر پر می‌شود. (مثال 0000110101010101)

۵- برنامه‌نویسی آزمایش: ثابت‌های زیر را برای استفاده در کد در بخش مرتبط با آن تعریف کنید.

```
/* USER CODE BEGIN PD */
// Select DAC1 for usage
#define DACx              DAC1
// Select Channel 1 for DAC
#define DACx_CHANNEL      DAC_CHANNEL_1
/* USER CODE END PD */
```

ساختار مرتبط با DAC را فراخوانی کنید.

```
/* USER CODE BEGIN PV */
// DAC configuration structure
DAC_HandleTypeDef         DacHandle;
// DAC channel configuration structure
static DAC_ChannelConfTypeDef    sConfig;
/* USER CODE END PV */
```

واحد DAC استفاده شده را به کتابخانه HAL معرفی کنید. DAC را راه‌اندازی کنید و وضعیت سلامت راه اندازی را توسط یک if چک کنید.

```
/* USER CODE BEGIN 2 */
DacHandle.Instance = DACx;
//
if (HAL_DAC_Start(&DacHandle,DACx_CHANNEL) != HAL_OK){
      //start Error
      Error_Handler();
}
  /* USER CODE END 2 */
```

در نهایت چهار سطح ولتاژ مختلف به DAC وارد کنید و بین هر مقداردهی ۵۰ میلی ثانیه تاخیر قراردهید.

```
while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
      /*##-1- Set DAC Channel1 DHR register#####################################*/
        if (HAL_DAC_SetValue(&DacHandle, DACx_CHANNEL, DAC_ALIGN_8B_R, 0x00) !=
HAL_OK)
        {
          /* Setting value Error */
          Error_Handler();
        }

        HAL_Delay(1);
```

```
        /*##-2- Set DAC Channel1 DHR register######################################*/
          if (HAL_DAC_SetValue(&DacHandle, DACx_CHANNEL, DAC_ALIGN_8B_R, 0x64) !=
HAL_OK)
          {
            /* Setting value Error */
            Error_Handler();
          }

          HAL_Delay(1);
         /*##-3- Set DAC Channel1 DHR register######################################*/
          if (HAL_DAC_SetValue(&DacHandle, DACx_CHANNEL, DAC_ALIGN_8B_R, 0x96) !=
HAL_OK)
          {
            /* Setting value Error */
            Error_Handler();
          }
          HAL_Delay(1);

          /*##-4- Set DAC Channel1 DHR register######################################*/
          if (HAL_DAC_SetValue(&DacHandle, DACx_CHANNEL, DAC_ALIGN_8B_R, 0xFF) !=
HAL_OK)
          {
            /* Setting value Error */
            Error_Handler();
          }
          HAL_Delay(1);
  }
```

شکل موج خروجی را توسط اسیلوسکوپ مشاهده کنید و حتما دقت کنید زمین اسیلوسکوپ به زمین برد متصل باشد.

۶- تمرین: با استفاده از واحد DAC یک شکل موج سینوسی ایجاد کنید.

آزمایش ۷: راه‌اندازی واحد Timer

هدف آزمایش: در این آزمایش دانشجو با نحوه راه‌اندازی واحد تایمر و استفاده از آن آشنا می‌شود.

شرح آزمایش:

۱-مقدمه: تایمرها در میکروکنترلرهای STM32 یکی از اجزای کلیدی و کاربردی هستند که به طور کلی برای کنترل زمان و انجام عملیات شمارش استفاده می‌شوند. این تایمرها قابلیت‌های گسترده‌ای دارند و به کاربر اجازه می‌دهد تا برای اهداف مختلفی مانند زمان‌بندی، واکنش به رویدادها و کنترل دستگاه‌های دیگر برنامه‌ریزی کند.

تایمرها در چند حالت مختلف کار می‌کنند که عبارت‌اند از:

۱- حالت تایمر: در این حالت، تایمر برای شمارش زمان بر اساس یک منبع کلاک داخلی عمل می‌کند. این حالت به طور معمول برای ایجاد تأخیر یا زمان‌بندی وظایف استفاده می‌شود.
۲- حالت شمارنده خارجی: تایمر می‌تواند به‌عنوان یک شمارنده برای شمارش لبه‌های بالارونده یا پایین‌رونده سیگنال‌های خارجی عمل کند. این حالت برای اندازه‌گیری فرکانس و یا شمارش پالس‌ها بسیار مفید است.
۳- حالت PWM: این حالت برای تولید سیگنال‌های PWM با فرکانس و چرخه وظیفه قابل تنظیم استفاده می‌شود. این قابلیت برای کنترل موتورها، چراغ‌های LED و دیگر دستگاه‌های الکترونیکی کاربرد دارد.
۴- حالت Input Capture: این حالت برای ثبت زمان وقوع یک رخداد خارجی هنگامی که یک لبه (بالارونده یا پایین‌رونده) در یک پین ورودی رخ می‌دهد، استفاده می‌شود. این حالت می‌تواند برای اندازه‌گیری زمان بین وقایع یا فرکانس سیگنال‌های ورودی کاربردی باشد.

تایمرها معمولاً از یک یا چند رجیستر برای ذخیره و مدیریت اطلاعات استفاده می‌کنند:

- رجیستر TCNT (Timer Counter Register): شمارنده اصلی که مقدار فعلی شمارش را نگه‌داری می‌کند.
- رجیستر ARR (Auto-Reload Register): حداکثر مقدار شمارش را مشخص می‌کند. بعد از رسیدن به این مقدار، شمارنده دوباره به صفر باز می‌گرداند.
- رجیستر PSC (PRESCALER): این رجیستر مقسمی برای شمارش تایمر است و می‌توان با افزایش و کاهش آن سرعت شمارش تایمر را تغییر داد.

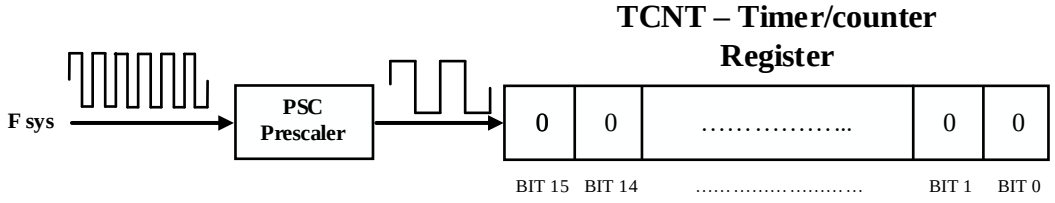
در ادامه به چند ویژگی و قابلیت تایمر اشاره شده است:

۱- سیگنال‌های وقفه: تایمرها قادرند سیگنال‌های وقفه ارسال کنند که پردازنده را قادر می‌سازد تا سریعاً به شرایط خاصی، مانند اتمام شمارش یا رسیدن شمارنده به مقدار مشخصی، واکنش نشان دهد.

۲- تنظیمات برنامه‌پذیر: میکروکنترلرهای STM32 دارای تنظیمات قابل برنامه‌ریزی متعددی برای تایمرها هستند، این قابلیت به مهندسان امکان می‌دهد تایمرها را با دقت بالا و متناسب با نیازهای خاص خود پیکربندی کنند.

۳- عملکرد مستقل: تایمرها معمولاً مستقل از پردازنده مرکزی عمل می‌کنند، به این معنی که می‌توانند بدون دخالت پردازشگر، وظایف زمانی را اجرا کنند.

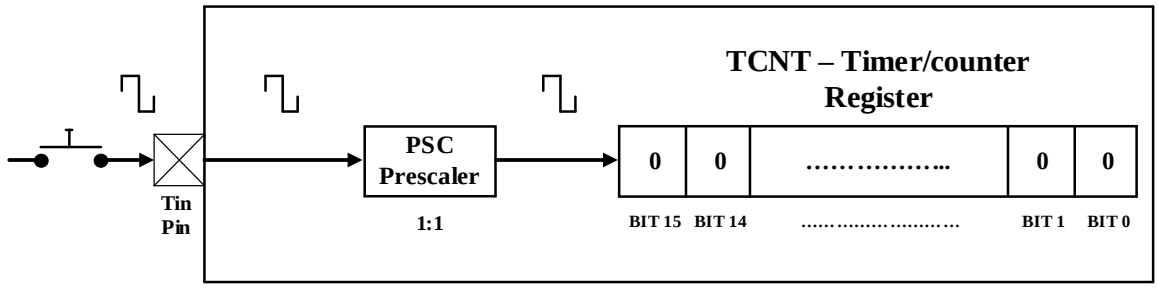
به عنوان مثال، یک تایمر ۱۶ بیتی قادر است از ۰ تا ۶۵۵۳۵ شمارش کند. این شمارش با هر سیکل کلاک به میزان یک واحد افزایش می‌یابد. نکته مهم این است که منبع کلاک تایمر به‌طور مستقیم از فرکانس اصلی سیستم تأمین نمی‌شود، بلکه فرکانس آن از طریق یک تقسیم‌کننده قابل تنظیم به دست می‌آید.



شکل(۱): دیاگرام تایمر بیسیک STM32

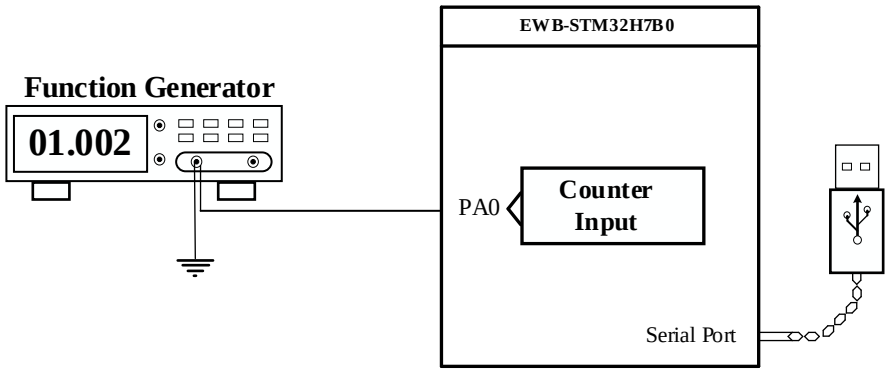
در حالت تایمر، رجیستر TCNT با هر سیکل کلاک بر اساس فرکانس Fsys/PSC به اندازه یک واحد افزایش می‌یابد. به عنوان مثال، اگر فرکانس سیستم 80MHZ (Fsys) و تقسیم‌کننده ۱۰۲۴ باشد، رجیستر TCNT هر ۱۲.۸ میکروثانیه یک واحد افزایش می‌یابد. در نتیجه، اگر بخواهید از ۰ تا ۶۵۵۳۵ شمارش کنید، این فرآیند تقریباً ۰.۸۳۹ ثانیه به طول می‌انجامد و در پایان شمارش یک سیگنال وقفه ایجاد می‌شود. اگر بخواهید به ازای مقادیر متفاوتی سیگنال وقفه دریافت کنید، قادر خواهید بود با استفاده از قابلیت سخت‌افزاری به نام رجیستر Preload، شمارش را از یک مقدار اولیه به جای صفر شروع کنید. این امکان به شما اجازه می‌دهد تا هر فاصله زمانی دلخواهی را به راحتی ایجاد کنید.

ماژول تایمر می‌تواند در حالت شمارنده (Counter) نیز عمل کند، به‌طور خاص زمانی که منبع سیگنال به‌طور دقیق مشخص نیست. به عنوان نمونه، اگر یک سیگنال خارجی مانند فشردن یک کلید ورودی اعمال شود، تایمر می‌تواند با هر لبه بالارونده یا پایین‌رونده سیگنال شمارش کند. با توجه به تصویر زیر، سیگنال از یک منبع خارجی به ورودی کلاک تایمر اعمال شده و از طریق یک تقسیم‌کننده عبور می‌کند. از این طریق می‌توانید تعداد دفعات فشردن کلید را با نظارت بر مقدار رجیستر TCNT تعیین کنید.



شکل(۲): دیاگرام تایمر STM32 در حالت شمارنده با منبع خارجی

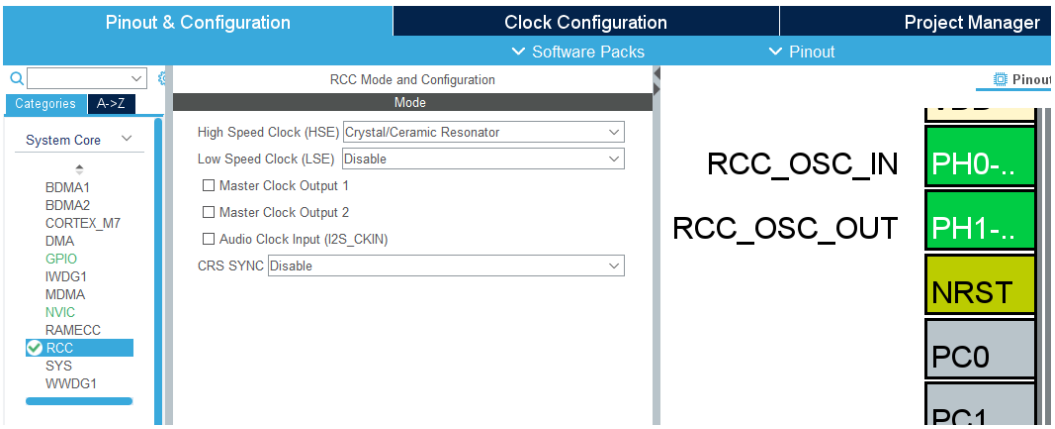
۲-شماتیک آزمایش: در این آزمایش فقط لازم است که پایه PA0 را به عنوان ورودی پالس در نظر بگیرید و از طریق پورت USART مقدار فرکانس اندازه‌گیری شده را مشاهده کنید.



شکل(۳): شماتیک اتصال دستگاه مولد سیگنال به برد

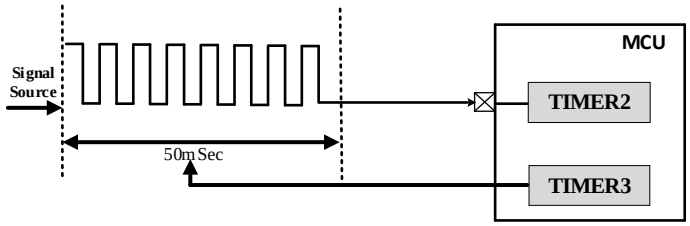
* دامنه ولتاژ اعمالی به پایه ورودی PA0 حداکثر ۳.۳ ولت است و نباید شامل ولتاژهای منفی شود لذا در اتصالات این آزمایش دقت شود.

۳- تنظیمات نرم‌افزار STM32CubeIDE: مانند قبل پروژه را ایجاد کنید. واحد USART را نیز فعال کنید. سپس تنظیمات بخش کلاک سیستم را طبق تصویر زیر بر روی منبع کلاک خارجی پر سرعت یا HSE و از نوع کریستال/رزوناتور سرامیکی تنظیم کنید.



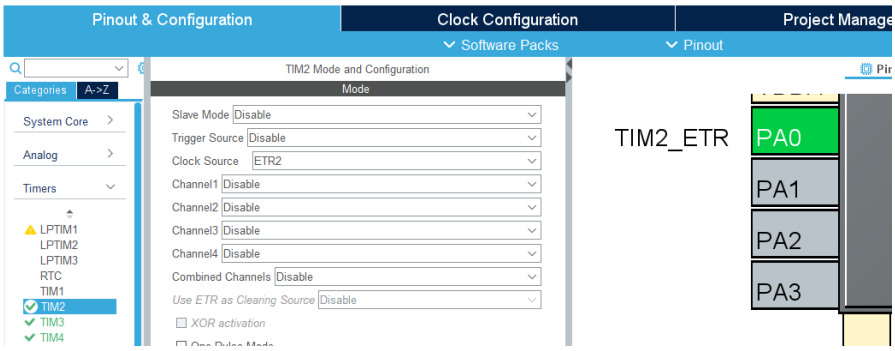
شکل(٤): تنظیم واحد RCC بر روی کلاک خارجی

در این آزمایش، به دو تایمر نیاز است که تنظیمات آن‌ها طبق تصویر انجام می‌شود. فرکانس به این صورت اندازه‌گیری می‌شود: از تایمر TIM2 در حالت شمارنده و از تایمر TIM3 در حالت تایمر ساده استفاده شده است.هنگامی که TIM3 سرریز می‌شود، مقدار شمارنده‌ی TIM2 خوانده می‌شود. این مقدار در ۲۰ ضرب می‌شود تا فرکانس سیگنال ورودی محاسبه گردد.



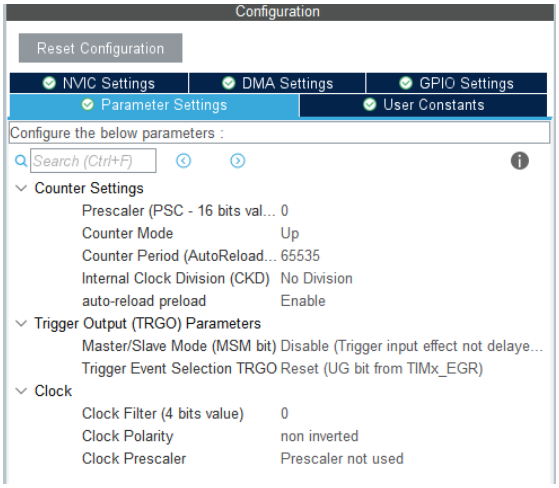
شکل(۵): دیاگرام عملکردی آزمایش فرکانس‌متر

ابتدا تایمر TIM2 را فعال کنید، بدین صورت که منبع کلاک را بر روی ETR2 که معادل پایه PA0 است قرار دهید.



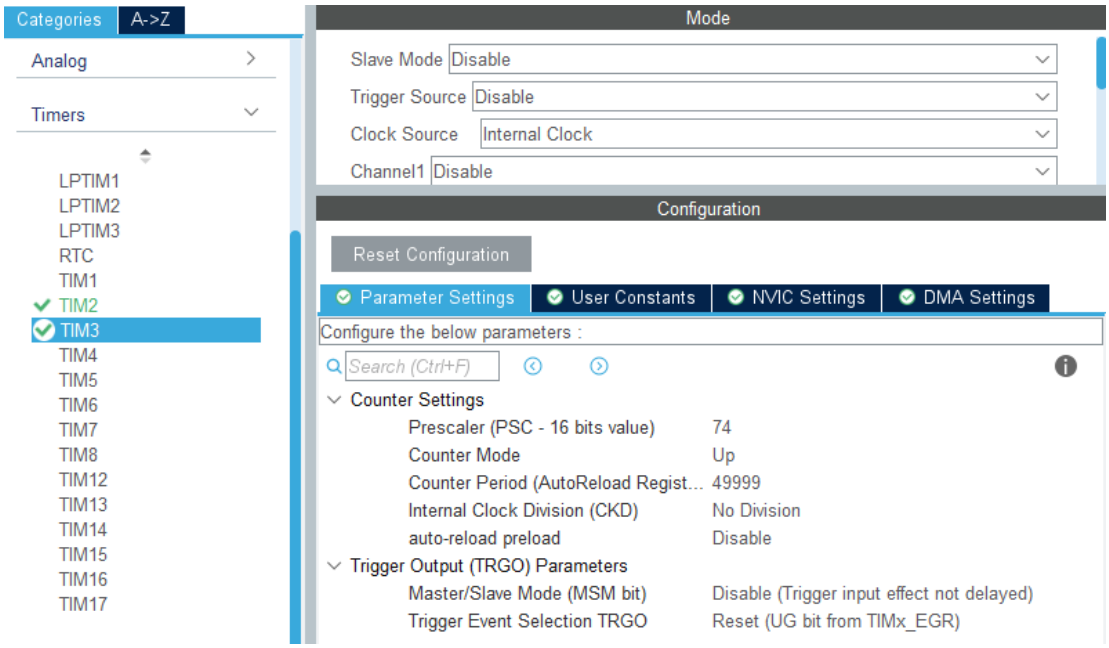
شکل(٦): تنظیمات واحد TIM2 در حالت شمارنده با منبع خارجی

از سربرگ Configuration وارد بخش Parameter Settings شوید و تنظیمات را دقیقا مطابق تصویر انجام دهید. مد شمارشی در این تایمر را بالا رونده قرار دهید و مد خود ریست شونده شمارشگر تایمر را نیز فعال کنید.



شکل(۷): تنظیمات واحد TIM2 در حالت شمارنده با منبع خارجی

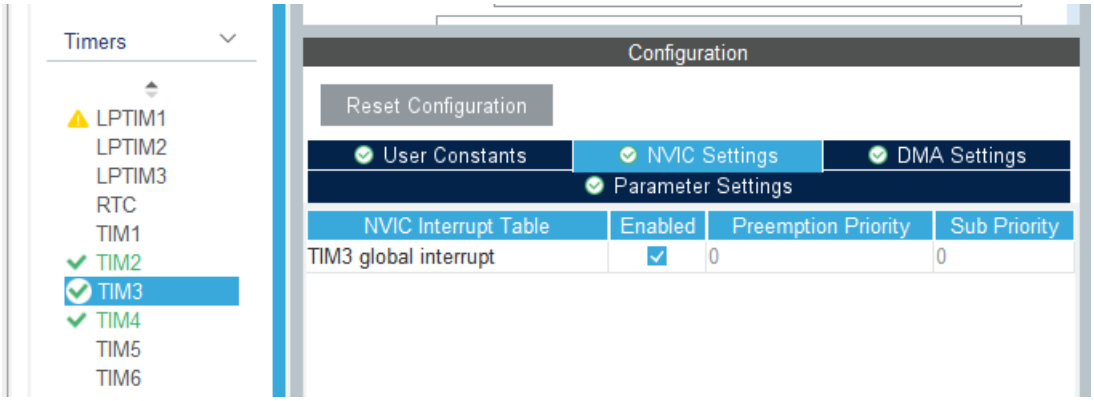
برای تایمر TIM3 منبع کلاک را بر روی کلاک داخلی قرار دهید و مقسم را بر روی عدد ۷٤ تنظیم کنید، سپس مد شمارنده تایمر را بر روی حالت بالارونده قرار دهید. مقدار Counter Period را نیز ٤۹۹۹۹ در نظر بگیرید.



شکل(۸): تنظیمات واحد TIM3 در حالت شمارنده

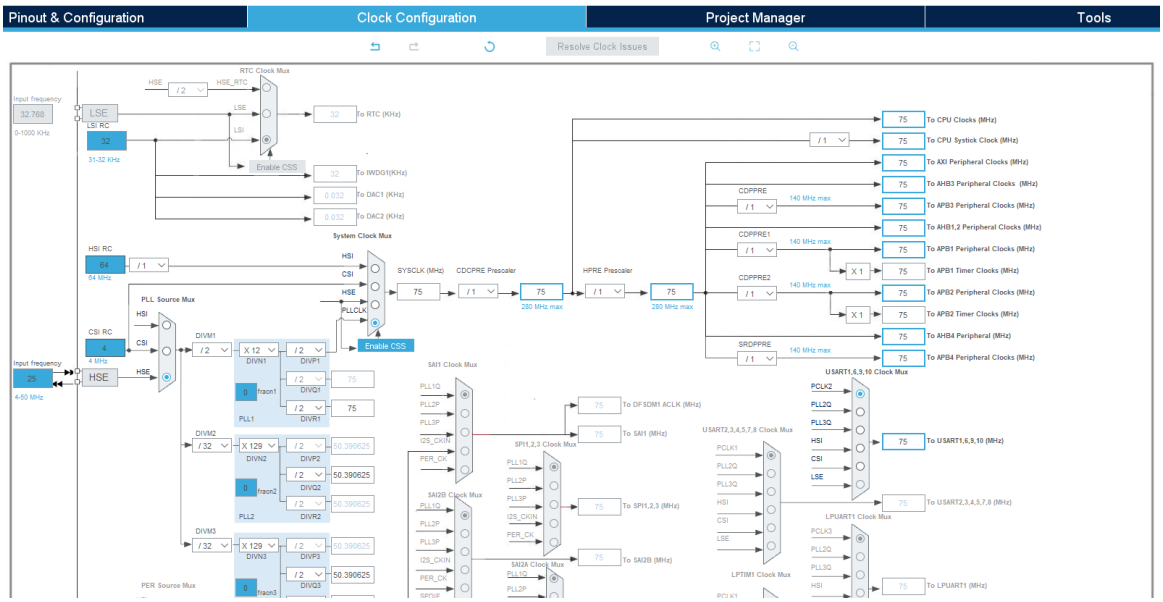
$$T_{OUT} = \frac{PSC \times Preload}{F_{CLK}}$$

فرکانس سیستم ۷۵مگاهرتز و مقدار مقسم (PSC) ۷٤ است و زمان مدنظر T_{out} برابر با ۰.۰۵ثانیه است بنابرین با جای گذاری در رابطه بالا عدد ٤۹۹۹۹ برای Counter Period بدست می‌آید. برای این تایمر از بخش NVIC Settings وقفه مرتبط با آن را فعال کنید.



شکل(۹): فعال‌سازی وقفه مربوط به TIM3

در سربرگ Clock Configuration تنظیمات مربوط به فرکانس سیستم را تعیین کنید.



شکل(۱۰): تنظیمات کلاک میکروکنترلر در نرم‌افزارCubeMx

در این بخش فوق مالتی پلکسرهای PLL Source و System Clock را به ترتیب بر روی HSE و PLLCLK قرار دهید. تنظیمات را ذخیره کنید و به بخش کد نویسی آزمایش وارد شوید.

٤- معرفی توابع مورد استفاده: در این آزمایش به توابع زیر نیاز است.

HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim)

این تابع در کتابخانه‌های HAL به صورت weak تعریف شده است، به این معنا که امکان بازتعریف آن وجود دارد. این تابع زمانی که تایمر سرریز می‌شود، به صورت خودکار فراخوانی می‌شود. آرگومان ورودی این تابع به ساختار TIM_HandleTypeDef اشاره دارد که حاوی اطلاعات پیکربندی مربوط به ماژول تایمر است. همچنین، برای شروع و توقف فعالیت تایمر، می‌توان از توابع HAL_TIM_Base_Start و HAL_TIM_Base_Stop استفاده کرد.

٥- برنامه‌نویسی آزمایش: متغیرهای زیر را تعریف کنید. متغییر gu32_CounterTicks برای پایش شمارنده تایمر و gu32_Freq برای ذخیره فرکانس ورودی استفاده می‌شود.

```
/* USER CODE BEGIN PV */
// Variable to store the counter ticks
uint32_t gu32_CounterTicks = 0;
// Variable to store the frequency value
uint32_t gu32_Freq = 0;
/* USER CODE END PV */
```

همچنین برای رشته پیام نیز متغیر زیر را تعریف کنید.

```
/* USER CODE BEGIN 1 */
      // Array to store the message to be transmitted
      uint8_t MSG[35] = {'\0'};
/* USER CODE END 1 */
```

در ابتدا تایمرهای مورد استفاده را فعال کنید.

```
/* USER CODE BEGIN 2 */
  // Start the TIM2 base timer (non-interrupt mode)
  HAL_TIM_Base_Start(&htim2);
  // Start the TIM3 base timer with interrupt enabled
  HAL_TIM_Base_Start_IT(&htim3);
/* USER CODE END 2 */
```

تابع مرتبط با نمونه برداری از سیگنال ورودی را مطابق شکل زیر تنظیم کنید.

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
{
	// Check if the interrupt is triggered by TIM3
	if (htim -> Instance == TIM3)
	{
		// Store the current counter value from TIM2
		gu32_CounterTicks = TIM2 -> CNT;
		// Calculate frequency by multiplying counter ticks
		gu32_Freq = gu32_CounterTicks * 20;
		// Reset the TIM3 counter
		TIM3 -> CNT = 0;
		// Reset the TIM2 counter
		TIM2 -> CNT = 0;
	}
}
/* USER CODE END 4 */
```

و در حلقه اصلی برنامه ارسال داده را بنویسید:

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

      /* USER CODE BEGIN 3 */
        // Formatting the frequency value into the message
        sprintf(MSG, "Freq Value = %d\r\n", gu32_Freq);
       // Sending the formatted message via UART
        HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
       // Delay of 500 ms
        HAL_Delay(500);
    }
```

۶- تمرین: برنامه ای بنویسید که فاصله زمانی بین دو بار فشردن کلید را اندازه‌گیری نماید و آن را از طریق USART ارسال کند.

آزمایش ۸: تولید موج PWM

هدف آزمایش: در این آزمایش، دانشجو با نحوه‌ی تولید پالس مربعی به وسیله‌ی واحد تایمر آشنا می‌شود.

شرح آزمایش:

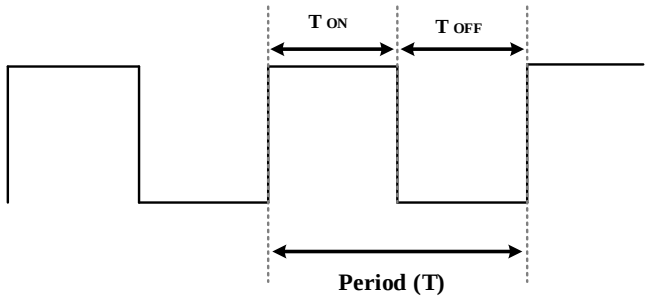
۱-مقدمه: مدولاسیون پهنای پالس، یا PWM (Pulse Width Modulation)، روشی مؤثر برای تولید سیگنال‌های دیجیتال است که کاربردهای گسترده‌ای در کنترل تجهیزات الکترونیکی دارد. به‌ویژه در میکروکنترلرهایی مانند STM32، PWM به عنوان یکی از روش‌های کلیدی برای تنظیم روشنایی LEDها، کنترل دور موتور و سایر وظایف کنترلی استفاده می‌شود.

PWM از تغییرات نسبت زمان‌های فعال و غیرفعال برای تولید سیگنال‌های دیجیتال استفاده می‌کند. این پارامترها شامل موارد زیر هستند:

زمان فعال (T_{on}) : مدت زمانی که سیگنال در حالت HIGH است.

زمان غیرفعال (T_{off}) : مدت زمانی که سیگنال در حالت LOW است.

دوره تناوب (T) : مجموع T_{on} و T_{off} که نشان‌دهنده‌ی فاصله‌ی بین دو پالس متوالی است.



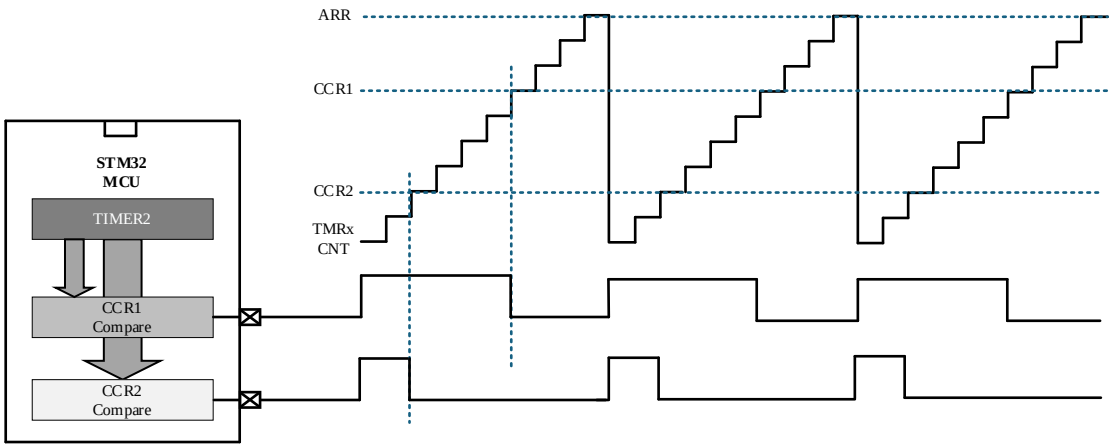
شکل(۱): دوره تناوب در یک سیگنال PWM

میزان روشنایی یا مقدار DC سیگنال تولید شده به نسبت زمان فعال به دوره کلی است (که چرخه وظیفه یا Duty Cycle نامیده می‌شود)، بستگی دارد. این چرخه وظیفه به صورت درصد بیان می‌شود و نشان‌دهنده‌ی این است که چه میزان از یک دوره‌ی کلی، سیگنال در وضعیت HIGH قرار دارد.

$$Duty\ Cycle[\%] = \frac{T_{ON}}{T_{ON} + T_{OFF}} \times 100$$

دقت چرخه وظیفه یک PWM با واحد bit مشخص می‌شود. برای مثال، مقدار D.C یک PWM با دقت ۳ بیت می‌تواند بین ۸ حالت مختلف (۰٪، ۱۲.۵٪، ۲۵٪، ۳۷.۵٪، ۵۰٪، ۶۲.۵٪، ۷۵٪، ۸۷.۵٪ و ۱۰۰٪) تغییر کند. در مقابل، یک PWM با دقت ۸ بیت قادر است ۲۵۶ حالت مختلف را اجرا کند که امکان تغییرات بسیار دقیق‌تری را فراهم می‌آورد.

در آزمایش فرکانس‌متر بیان شد که یکی از حالت‌های کاری تایمرها، حالت PWM است. در این حالت کاری، تایمر توسط یک منبع کلاک داخلی تحریک می‌شود و بصورت بالارونده تا مقدار رجیستر ریست شونده شروع به شمارش می‌کند. کانال خروجی نیز در همین حین بر روی ۱ منطقی تنظیم می‌شود و به همین صورت باقی می‌ماند تا شمارنده به مقدار رجیستر CCRx برسد، سپس کانال خروجی به ۰ منطقی تغییر می‌کند و مجدد در همین حالت ۰ منطقی باقی می‌ماند تا شمارنده به مقدار رجیستر ریست شونده (ARRx) دست یابد. این روند تا پایان کار تایمر تکرار می‌شود.



شکل(۲): دیاگرام زمان‌سنجی TIMER و همگام‌سازی موج‌های خروجی با CCR1 و CCR2

در این حالت شما یک پالس PWM را ساخته‌اید که فرکانس آن توسط منبع کلاک داخلی، رجیستر ARRx و مقسم کلاک (PSC) تعیین می‌شود:

$$F_{PWM} = \frac{F_{CLK}}{(ARR + 1) + (PSC + 1)}$$

چرخه وظیفه نیز توسط رجیستر CCRx و ARRx تعیین می‌شود:

$$Duty\ Cycle[\%] = \frac{PSC}{ARR} \times 100$$

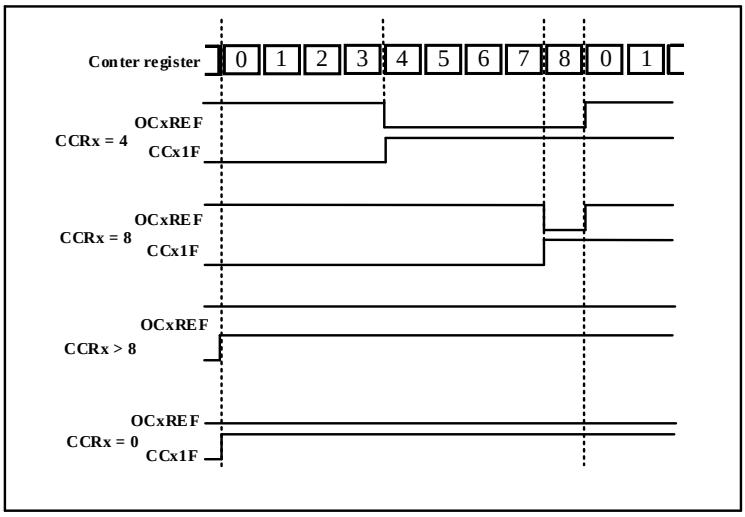
جدول زیر فرکانس و دقت PWM را در فرکانس 72MHZ محاسبه نموده است.

STM32 16_bit timer	PWM resolution	PWM frequency
72 MHz	16 bit	~1.1 kHz
72 MHz	14 bit	~4.4 kHz
72 MHz	12 bit	~17.5 kHz
72 MHz	10 bit	~70 kHz
72 MHz	8 bit	~281 kHz
72 MHz	6 bit	~1.125 kHz
72 MHz	4 bit	~4.5 kHz

جدول(۱): جدول مقایسه فرکانس PWM و دقت PWM برای تایمر ۱۶ بیتی STM32

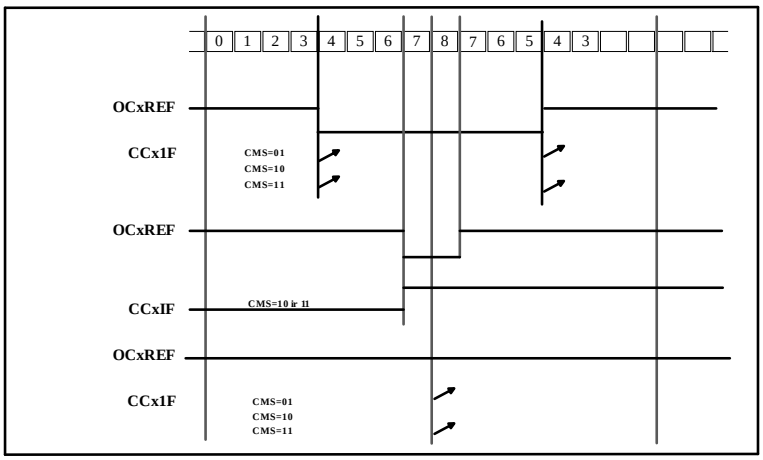
روش‌های کاری PWM به دو نوع اصلی تقسیم می‌شود که هر کدام ویژگی‌ها و کاربردهای خاص خود را دارند:

۱- PWM تراز شونده با لبه (Edge-Aligned PWM): در این حالت، سیگنالOCxREF، سیگنال مرجع PWM، تا زمانی که مقدار شمارنده تایمر (TIMx_CNT) کمتر از مقدار مقایسه (TIMx_CCRx) باشد، در حالت HIGH باقی می‌ماند. به محض رسیدن شمارنده به مقدارCCRx ، سیگنال به LOW تبدیل می‌شود. اگر مقدار CCRx بیشتر از مقدار ریست شونده (TIMx_ARR) باشد، OCxREF همواره HIGH خواهد بود و در صورت صفر بودن CCRx ، OCxREF نیز به LOW تغییر می‌کند.



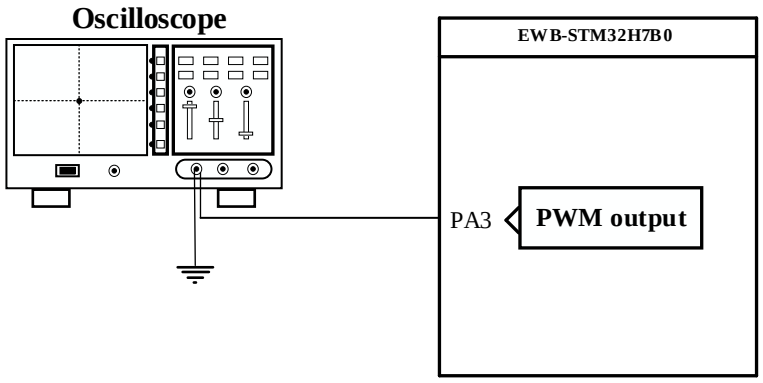
شکل(۳): شکل موج PWM در حالت (Edge-Aligned PWM)

۲- PWM ترازشونده با مرکز (Center-aligned PWM): در این روش، پرچم مقایسه‌ای می‌تواند در هر دو حالت شمارش رو به بالا یا پایین فعال شود، که این بستگی به نوع پیکربندی دارد. این نوع PWM عموماً در کاربردهایی که نیاز به کنترل دقیق‌تری دارند، مورد استفاده قرار می‌گیرد.



شکل(٤): شکل موج PWM در حالت (Center-aligned PWM)

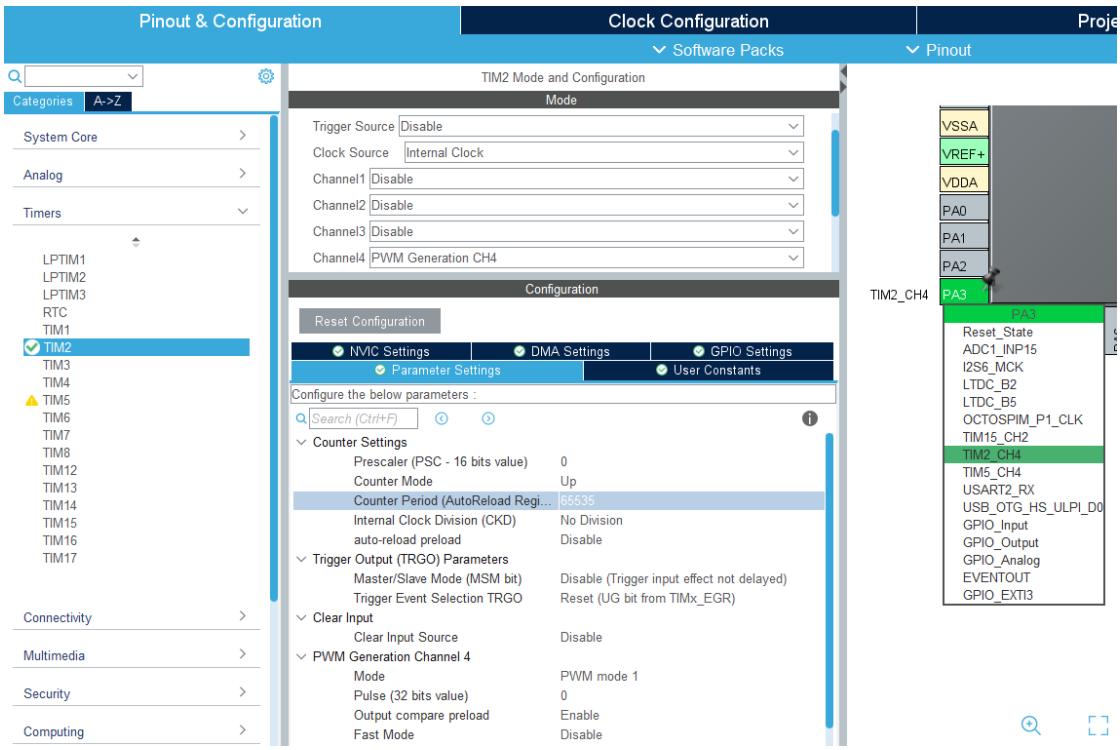
۲-شماتیک آزمایش: در این آزمایش تنها کافی است که پایه PA3 را به اسیلوسکوپ متصل نمایید.



شکل(۵): شماتیک اتصال خروجی PWM تایمر به اسیلوسکوپ

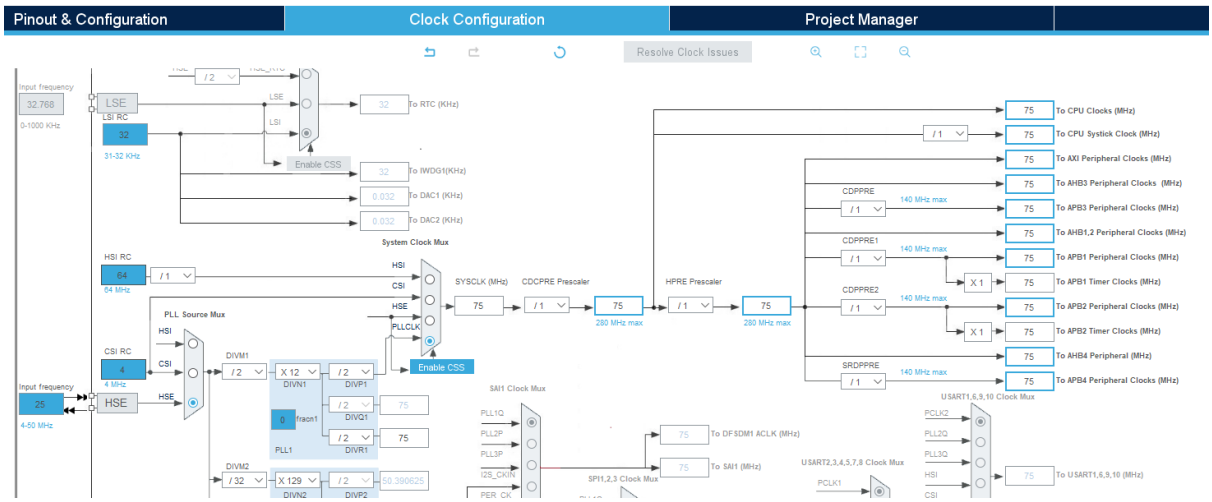
۳- تنظیمات نرم‌افزار STM32CubeIDE: پروژه را طبق روش مرسوم گذشته ایجاد کنید. بخش دیباگ را بر روی Serial wire تنظیم کنید. همچنین کلاک سیستم را بر روی HSE و مد کریستال خارجی تنظیم کنید.

تایمر ۲ را فعال کنید و منبع تحریک آن را بر روی کلاک ورودی تنظیم کنید. پین PA3 را بر روی TIM2_CH4 تنظیم کنید. سپس کانال ٤ تایمر ۲ را برای تولید PWM تنظیم کنید. دوره شمارنده را بر روی ٦٥٥٣٥ قرار داده، رجیستر خودریست شونده را فعال کنید و پین PA3 را بر روی TIM2_CH4 قرار دهید. مشاهده می‌کنید که پین PA3 سبز رنگ می‌شود.



شکل(٦): تنظیمات کانال چهار TIM2 با خروجی PWM

در انتها تنظیمات کلاک را مانند تصویر زیر تنظیم کنید.



شکل(۷): تنظیمات کلاک میکروکنترلر در نرم‌افزارCubeMx

٤- معرفی توابع مورد استفاده: در این آزمایش از رجیستر CCR استفاده شده است. CCR یا Capture Compare Register رجیستری است که مقدار مقایسه‌شونده در آن قرار می‌گیرد و با گذر مقدار شمارنده از مقدار CCR، خروجی تغییر وضعیت می‌دهد و پس از رسیدن به مقدار ARR یا Auto Reload Register به حالت اولیه باز می‌گردد.

٥- برنامه‌نویسی آزمایش: پس از ایجاد کتابخانه HAL ، متغییر نگهدارنده مقدار مقایسه را تعریف کنید.

```
/* USER CODE BEGIN 1 */
      // Duty cycle for Channel 4 is initialized to 0
      uint16_t CH4_DC = 0;
/* USER CODE END 1 */
```

حال پیش از حلقه بی‌نهایت، PWM را برای تایمر ۲ و کانال ٤ فعال کنید و داخل حلقه بی‌نهایت از دو حلقه متناهی استفاده کنید تا دو مد افزایشی و کاهشی برای PWM ایجاد شود.

```
  // Start PWM signal generation on TIM2, Channel 4
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
    while (1)
    {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      //increase the duty cycle from 0 to 65400
        while (CH4_DC < 65400)
        {
              // Set the current duty cycle value for Channel 4
              TIM2->CCR4 = CH4_DC;
              // Increment the duty cycle by 100 units
              CH4_DC += 100;
              // Add a small delay of 1ms to control the speed of the change
              HAL_Delay(1);
        }
        //decrease the duty cycle from 65400 to 100
```

```
        while (CH4_DC > 100)
        {
                // Set the current duty cycle value for Channel 4
                TIM2->CCR4 = CH4_DC;
                // Decrease the duty cycle by 100 units
                CH4_DC -= 100;
                // Add a small delay of 1ms to control the speed of the change
                HAL_Delay(1);
        }
}
```

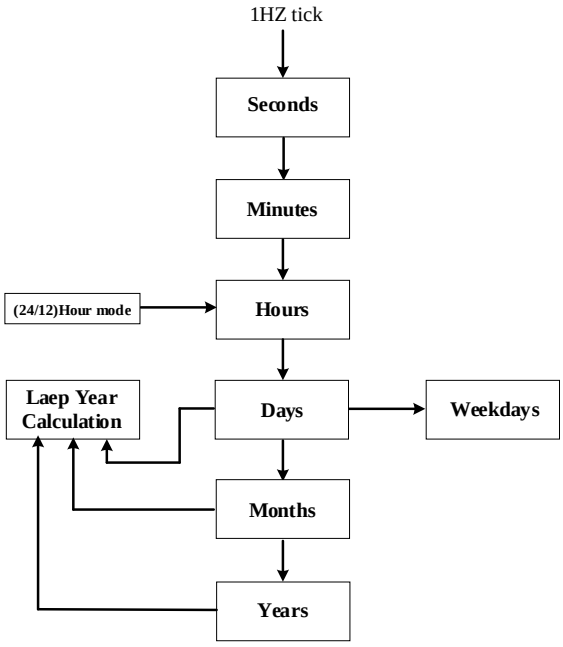
٦- تمرین: با استفاده از یک پتانسیومتر یک پالس مربعی با D.C قابل تنظیم ایجاد کنید.

آزمایش ۹: راه‌اندازی واحد RTC

هدف آزمایش: در این آزمایش دانشجو با نحوه راه‌اندازی واحد RTC در میکروکنترلر آشنا می‌شود.

شرح آزمایش:

۱-مقدمه: RTC (Real-Time Clock) در میکروکنترلرهای STM32، وظیفه حفظ ساعت و تاریخ را در سیستم‌هایی که نیاز به نگهداری زمان دارند، بر عهده دارد. این ماژول به‌ویژه در کاربردهای مختلف مانند ساعت‌های دیجیتال، سیستم‌های هشدار و زمان‌بندی رویدادها استفاده می‌شود. RTCها با مصرف توان بسیار کم طراحی شده‌اند زیرا وقتی تغذیه اصلی سیستم قطع است معمولاً به‌کار خود ادامه می‌دهند. مصرف کم این امکان را برای آن‌ها فراهم می‌کند تا زمان فعلی را نسبت به مرجع زمانی که معمولاً توسط ریزپردازنده مستقیماً تنظیم می‌شود، حفظ کنند. یک واحد RTC ممکن است شامل امکاناتی مانند حافظه پشتیبان، تایمر واچ‌داگ جهت نظارت بر میکروپروسسور و تایمرهای پایین‌شمار برای ایجاد رویدادهای Real-time باشد. برخی از RTCها همچنین دارای خروجی‌های وقفه‌ای در سطح ثانیه یا دقیقه هستند که به پردازنده امکان پردازش دقیق‌تر رویدادهای زمانی را می‌دهد.



شکل(۱): نحوه شمارش تاریخ و ساعت در واحد RTC

در ادامه به بیان ویژگی‌ها و قابلیت‌های RTC در میکروکنترلرهای STM32 پرداخته می‌شود:

۱- ساختار و ویژگی‌ها: RTC معمولاً شامل یک حافظه غیر فرار (Backup SRAM) است که امکان ذخیره تنظیمات و داده‌ها را در زمان قطع برق فراهم می‌کند. این ماژول می‌تواند تحت ولتاژ پایین (معمولاً زیر ۱.۳ ولت) کار کند و برای استمرار عمل به باتری پشتیبان نیاز دارد. (مانند باتری سکه‌ای CR2032)

۲- ثبت زمان و تاریخ: RTC می‌تواند زمان را به صورت ساعت (HH)، دقیقه (MM) و ثانیه (SS) و همچنین تاریخ را به‌صورت روز(DD)، ماه (MM) و سال (YYYY) ثبت کند. مهم‌تر از آن، RTC توانایی مدیریت تقویم میلادی و تشخیص سال‌های کبیسه را دارد. RTCها همچنین قابلیت تنظیم زمان و تاریخ را دارند و می‌توان آن‌ها را به راحتی از طریق نرم‌افزار راه‌اندازی کرد.

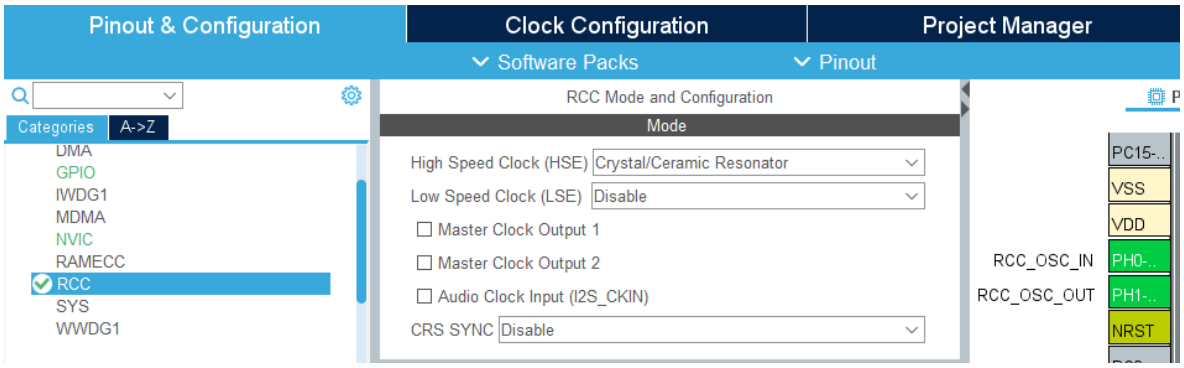
۳- بیدار شدن از حالت sleep: RTC می‌تواند زمان‌های مشخصی را به عنوان زنگ (alarm) تنظیم کرده و سیستم را از حالت خواب بیدار کند. این ویژگی برای کاربردهایی که نیاز به مصرف انرژی پایین دارند، بسیار کارآمد است.

٤- کنترل سرعت نوسان: RTC می‌تواند به گونه‌ای تنظیم شود که نوسان ساعت (با نوسان‌ساز32.768 kHz) به بهترین شکل انجام شود و قادر به اصلاح نوسانات زمانی باشد. RTC توانایی نگهداری زمان با دقت بالا را دارد و بر حسب نیاز می‌توان آن را تنظیم کرد.

۵- مدیریت Interrupt: RTC قادر به تولید وقفه (interrupt) در زمان‌های مشخص است. این امکان به برنامه‌نویس این اختیار را می‌دهد که عملیات خاصی را در زمان‌های از پیش تعیین‌شده اجرا کند، که برای مدیریت کارهای زمان‌محور بسیار مفید است.

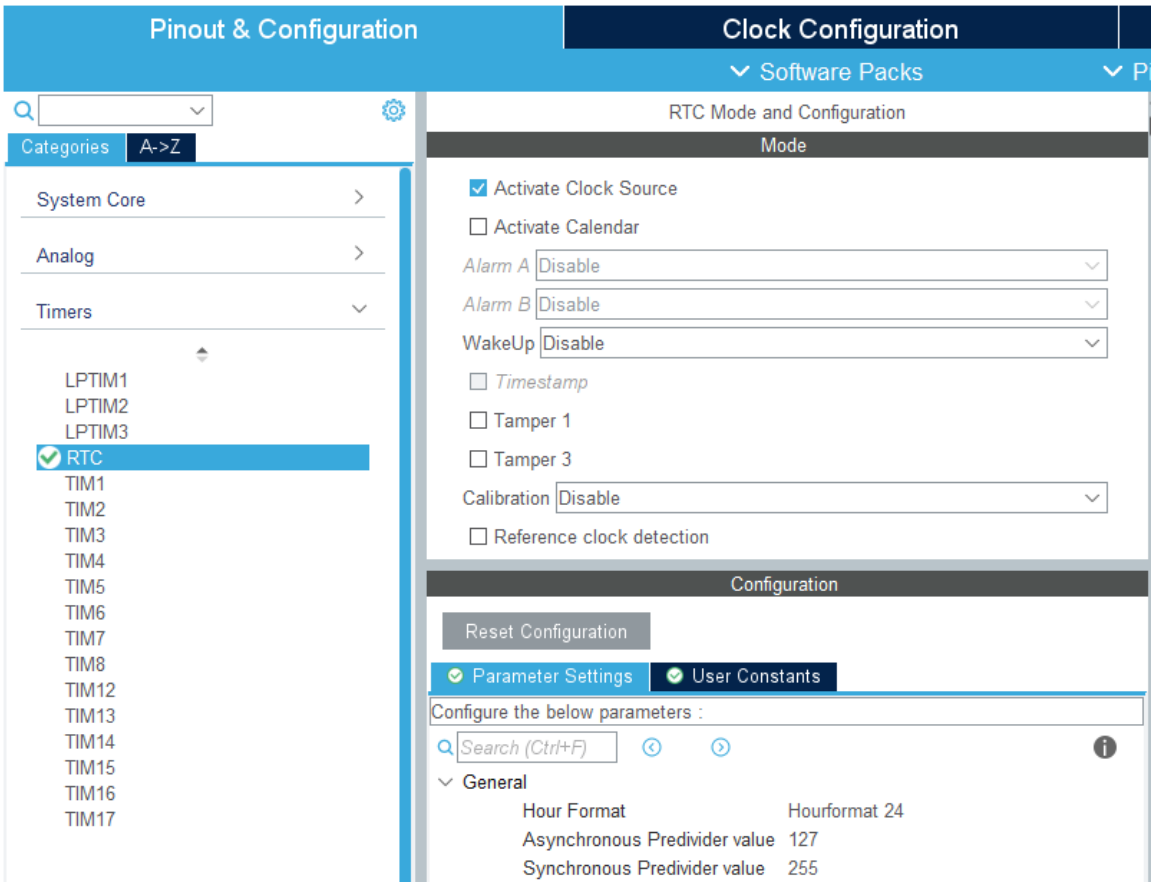
۲-شماتیک آزمایش: در این آزمایش کافی است پورت خروجی سریال را با کابل مربوطه به کامپیوتر متصل کنید.

۳- تنظیمات نرم‌افزار STM32CubeIDE: برای تنظیمات نرم‌افزار STM32CubeIDE، پروژه را مطابق با روش‌های مرسوم ایجاد کنید. بخش دیباگ را بر روی Serial Wire تنظیم کنید. سپس کلاک سیستم HSE را بر روی حالت کریستال خارجی تنظیم کنید و پورت USART را نیز مشابه آزمایش چهارم برای میکروکنترلر فعال نمایید.



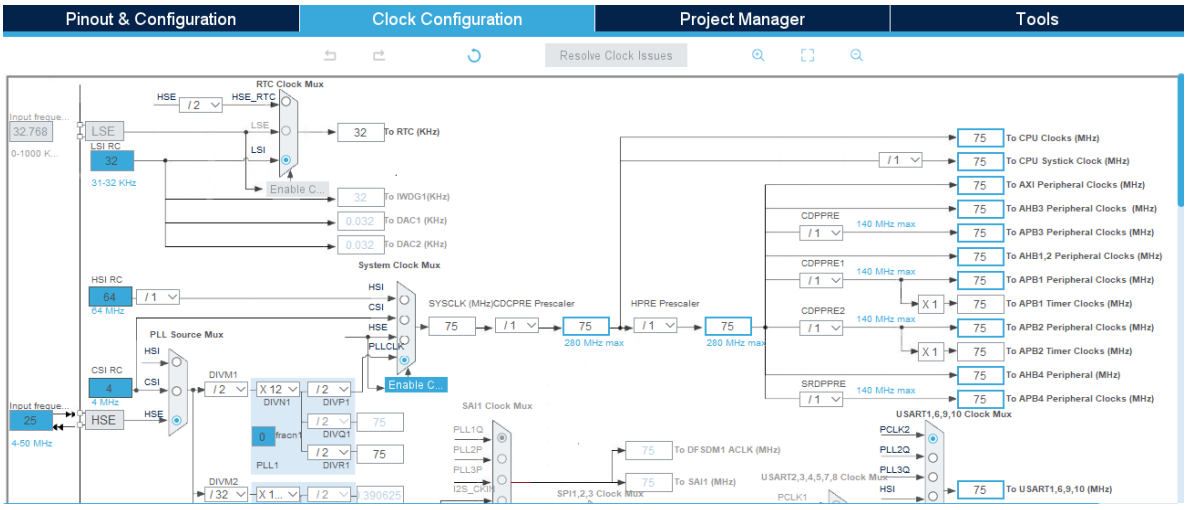
شکل (۲): فعالسازی کریستال خارجی برای کلاک HSE و LSE

در نهایت، از سربرگ تایمرها، زیرشاخه RTC را انتخاب کرده و تنظیمات مورد نظر را اعمال کنید.



شکل(۳): تنظیمات واحد RTC در نرم‌افزار

در نهایت از بخش Clock Configuration بخش مرتبط با ورودی کلاک RTC را مشابه تصویر زیر تنظیم کنید، دقت کنید که کلاک مربوط به RTC بر روی LSI فعال شده باشد.



شکل(٤): تنظیمات کلاک واحد RTC

تنظیمات ایجاد شده را ذخیره کنید.

٤- معرفی توابع مورد استفاده: در این آزمایش از دو تابع زیر استفاده شده است.

HAL_RTC_GetTime (RTC_HandleTypeDef *hrtc, RTC_TimeTypeDef *sTime, uint32_t Format)

این تابع برای دریافت زمان حاضر مورد استفاده قرار می‌گیرد. آرگومان اول اشاره به یک ساختار RTC_HandleTypeDef دارد که حاوی اطلاعات پیکربندی برای RTC است. آرگومان دوم اشاره به ساختار زمان و آرگومان سوم اشاره به فرمت پارامترهای وارد شده دارد که می‌تواند باینری یا BCD باشد.

HAL_RTC_GetDate (RTC_HandleTypeDef *hrtc, RTC_DateTypeDef *sDate, uint32_t Format)

این تابع برای دریافت تاریخ حاضر مورد استفاده قرار می‌گیرد. آرگومان اول مانند تابع قبل اشاره به یک ساختار RTC_HandleTypeDef دارد که حاوی اطلاعات پیکربندی برای RTC است. آرگومان دوم اشاره به ساختار تاریخ و آرگومان سوم نیز اشاره به فرمت پارامترهای وارد شده دارد که می‌تواند باینری یا BCD باشد.

٥- برنامه‌نویسی آزمایش: کتابخانه stdio.h را اضافه کنید.

```
/* USER CODE BEGIN Includes */
// Include standard input-output header
#include <stdio.h>
/* USER CODE END Includes */
```

دو نوع متغییر زمان و تاریخ را فراخوانی کنید.

```
/* USER CODE BEGIN PTD */
// Define RTC time structure
RTC_TimeTypeDef Time_Struct;
// Define RTC date structure
RTC_DateTypeDef Date_Struct;
/* USER CODE END PTD */
```

آرایه MSG را برای ذخیره دیتای ارسالی از طریق USART تعریف کنید.

```
/* USER CODE BEGIN 1 */
// Initialize message array
uint8_t MSG[35] = {'\0'};
/* USER CODE END 1 */
```

قبل از حلقه بینهایت عنوان پروژه را از طریق USART ارسال کنید.

```
  /* USER CODE BEGIN WHILE */
  // Format the initial message
  sprintf(MSG,"STM32H750 RTC experiment \r\n");
  // Transmit the message over UART
  HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
  // Clear the message array
  for( uint8_t counter = 0 ; counter < sizeof(MSG) ; counter++ )
    MSG[counter]=0;
  // Delay for 500 ms
  HAL_Delay(500);
  // Infinite loop
```

در حلقه While نیز کد زیر را بنویسید.

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

    /* USER CODE BEGIN 3 */
    // Get current RTC time
    HAL_RTC_GetTime(&hrtc, &Time_Struct, RTC_FORMAT_BIN);
    // Get current RTC date
    HAL_RTC_GetDate(&hrtc, &Date_Struct, RTC_FORMAT_BIN);
    // Format date into the message
    sprintf(MSG,"%d-%d-
%d",Date_Struct.Year+2023,Date_Struct.Month,Date_Struct.Date);
    // Transmit the date over UART
    HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
    // Clear the message array
    for( uint8_t counter = 0 ; counter < sizeof(MSG) ; counter++ )
      MSG[counter]=0;
    // Format time into the message
sprintf(MSG,"%d:%d:%d\r\n",Time_Struct.Hours,Time_Struct.Minutes,Time_Struct.Secon
ds);
    // Transmit the time over UART
    HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
    // Clear the message array
    for( uint8_t counter = 0 ; counter < sizeof(MSG) ; counter++ )
      MSG[counter]=0;

    // Delay for 1 second
    HAL_Delay(1000);
  }
    /* USER CODE END 3 */
```

داخل حلقه ابتدا تاریخ و زمان دریافت شده و سپس توسط USART ارسال می‌شود.

٦- تمرین: با ایجاد تغییراتی در برنامه ساعت و تاریخ روز را تنظیم کنید و سپس برنامه شروع به کار کند.

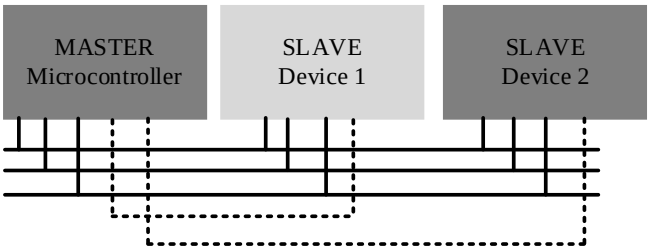
آزمایش ۱۰: راه‌اندازی ارتباط SPI

هدف آزمایش: در این آزمایش دانشجو با نحوه راه‌اندازی SPI و کار با حافظه فلش آشنا می‌شود.

شرح آزمایش:

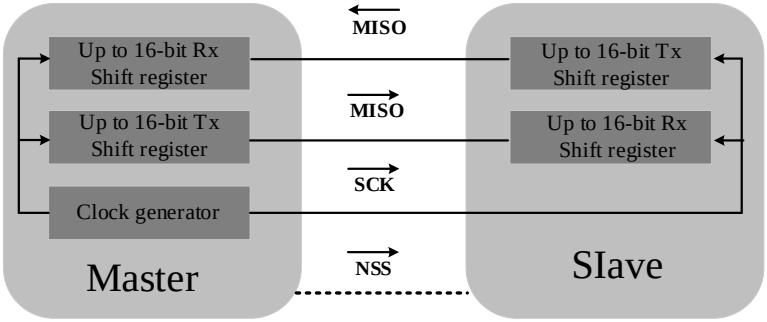
۱-مقدمه: رابط جانبی سریال SPI یکی از پروتکل‌های ارتباطی پرکاربرد در سیستم‌های میکروکنترلی است که برای تبادل داده بین میکروکنترلرها، میکروپروسسورها و دستگاه‌های جانبی مانند حافظه‌های فلش، سنسورها و نمایشگرهای OLED و TFT مورد استفاده قرار می‌گیرد. این پروتکل که در دهه ۱۹۸۰ توسط شرکت موتورولا توسعه یافت، امکان ارتباط دوطرفه سریع و کارآمد را فراهم می‌کند. به دلیل سرعت بالا و انعطاف‌پذیری، SPI به‌ویژه در پروژه‌هایی که نیاز به انتقال داده سریع بین میکروکنترلر و حافظه یا سنسورهای پیشرفته دارند، گزینه‌ای مناسب محسوب می‌شود.

در رابط SPI معمولا دو دستگاه حضور دارند که به درگاه SPI متصل هستند که یکی از دستگاه‌ها باید Master باشد و باقی Slave نام می‌گیرند، تفاوت این دو گروه در این است که Master با تولید پالس کلاک و ارسال یک فریم از داده ارتباط را شروع می‌کند و در همین زمان داده‌های سریال خارج شده از Master وارد Slave می‌شوند و این اساس اعمال خواندن و یا نوشتن توسط SPI است.



شکل(۱): قابلیت اتصال ارتباط SPI به چند دستگاه

در تصویر زیر نحوه اتصال دستگاه‌های Master و Slave را مشاهده می‌شود.



شکل(۲): نحوه اتصال دستگاه‌های Master و Slave در ارتباط SPI

درSPI ، چهار پایه اصلی وجود دارد:

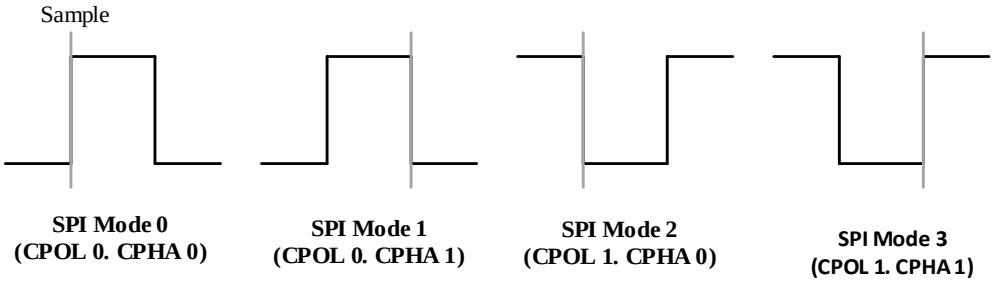
۱- MOSI (Master Output Slave Input) : این پایه به عنوان خروجی از Master و ورودی به Slave عمل می‌کند و داده‌ها را از Master به Slave ارسال می‌کند.

۲- MISO (Master Input Slave Output): این پایه ورودی به Master و خروجی از Slave است و برای دریافت داده‌ها از Slave به کار می‌رود.

۳- SCLK: این پایه سیگنال کلاک سریال است که Master وظیفه تأمین آن را بر عهده دارد و برای زمان‌بندی انتقال داده‌ها استفاده می‌شود.

٤- SS (Slave Select): این پایه برای انتخاب و کنترل Slave مورد نظر در مواردی که چندین Slave به Master متصل هستند، استفاده می‌شود.

دستگاه Master در ارتباط SPI موظف است تا سیگنال Clock را برای آماده سازی اولیه و تداوم روند انتقال اطلاعات ایجاد کند. بنابراین Master امکان مشخص کردن سرعت انتقال داده را توسط کنترل فرکانس سیگنال کلاک خط SCK را دارد که این مورد قابل برنامه ریزی است. کلاک SPI دو پارامتر دیگر نیز برای کنترل دارد که فاز کلاک (CPHA) و قطب کلاک (CPOL) نام دارند. فاز کلاک CPHA مشخص می‌کند که انتقال داده در هر بایت در کدام فاز انجام بپذیرد که دو حالت ابتدای پالس و انتهای پالس را می‌توان تنظیم کرد، قطب کلاک (CPOL) نیز مشخص میکند که حالت ماندگار پالس به چه صورت باشد که بین دو حالت 1 و 0 قابل انتخاب است. با مشاهده تصویر زیر میتوانید بیشتر توضیحات فوق را درک کنید.



شکل(۳): حالت‌های تشخیص داده بر اساس سیگنال کلاک

با تنظیم این پارامتر می‌توانیم به ٤ مُد کاری متفاوت دست پیدا کنیم.

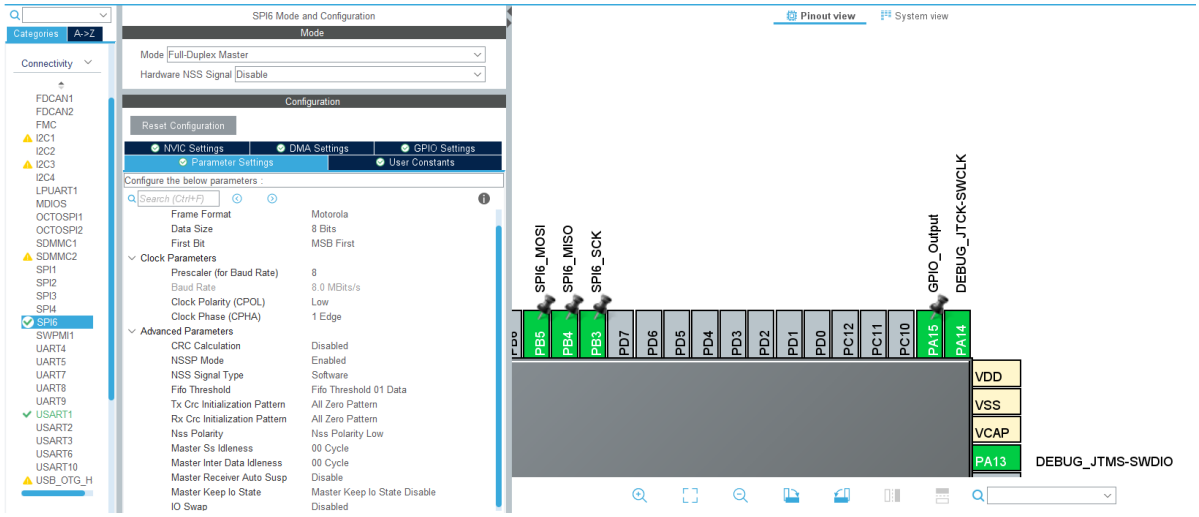
Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

جدول(۱): جدول تنظیم مُدهای تشخیص داده

۲-شماتیک آزمایش: در این آزمایش، هدف نمایش اطلاعات حافظه فلش تعبیه‌شده روی برد توسعه است. برای این کار، کافی است پورت خروجی سریال را با کابل مربوطه به کامپیوتر متصل کنید.

۳- تنظیمات نرم‌افزار STM32CubeIDE: تنظیمات کلاک سیستم و دیباگ را طبق پروژه‌های گذشته به صورت پیش فرض انجام دهید. USART را نیز مانند آزمایش‌های پیشین فعال کنید.

در بخش Connectivity گزینه SPI6 را انتخاب نمایید. تنظیمات زیر را اعمال کنید. همچنین پایه PA15 را طبق تصویر فعال و بر روی حالت کاری خروجی تنظیم کنید تا برای کاربرد کنترل Slave استفاده کنید. باقی پایه های PB5، PB4 و PB3 را برای کاربری SPI6 استقاده کنید.



شکل(٤): تنظیمات مربوط به ارتباط SPI در نرم‌افزار

تنظیمات را ذخیره کنید و کد را ایجاد کنید.

٤- معرفی توابع مورد استفاده: در این آزمایش از تابع HAL_SPI_TransmitReceive جهت برقراری ارتباط SPI استفاده شده است.

HAL_SPI_TransmitReceive(SPI_HandleTypeDef * hspi, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)

این تابع برای ارسال و دریافت داده‌ها از طریق SPI در حالت بلاکینگ (در این حالت، اجرای برنامه تا زمان تکمیل انتقال داده‌ها متوقف می‌شود) مورد استفاده قرار می‌گیرد. آرگومان اول اشاره به یک ساختار SPI_HandleTypeDef دارد که حاوی اطلاعات پیکربندی برای ماژول SPI است. آرگومان دوم اشاره به بافر داده‌های ارسالی و آرگومان سوم اشاره به بافر داده‌های دریافتی دارد. آرگومان چهارم مقدار حجم داده‌ها را مشخص می‌کند و آرگومان پنجم تعیین‌کننده مدت زمان مجاز برای اجرای عملیات است. این تابع مقدار HAL_StatusTypeDef را بازمی‌گرداند که نشان‌دهنده وضعیت اجرای عملیات است.

۵- برنامه‌نویسی آزمایش: در ابتدای برنامه متغیرهای زیر را تعریف کنید.

```
/* USER CODE BEGIN PV */
// Declare an array for storing messages
uint8_t MSG[35] = {'\0'};
// Declare variables to hold Chip ID and Unique ID
uint32_t Chip_ID;
uint32_t Chip_Unique_ID;
/* USER CODE END PV */
```

آرایه MSG برای انتقال داده بر بستر USART استفاده می‌شود. متغییرهای CHIP_ID و Chip_Unique_ID بابت ذخیره شناسه های حافظه فلش استفاده می‌شوند. شناسه Chip_ID اطلاعات کلی شامل حجم و نوع حافظه را نشان می‌دهد برای تمام بردها باید مقدار یکسانی دریافت شود. شناسه Chip_Unique_ID یک داده منحصر به هر حافظه فلش است که باید برای هر برد مقدار متفاوتی داشته باشد. نحوه دریافت این دو شناسه را می‌توانید در برگه اطلاعاتی حافظه فلش W25Q64JV شرکت WINBOND مطالعه کنید.

داخل تابع Main، توابع مرتبط با خواندن شناسه نوشته شده است.

```
/* USER CODE BEGIN 1 */
	// Function to transmit and receive data via SPI interface
	uint8_t flash_spi (uint8_t Data)
	{
	    uint8_t ret;
	    // Transmit data and receive response via SPI6
	    HAL_SPI_TransmitReceive(&hspi6, &Data, &ret, 1 , 100);
	    return ret;
	}

	// Function to get the JEDEC ID of the flash memory
	uint32_t flash_JEDEC_ID(void)
	{
	    uint32_t Chip_ID_L = 0 , Memory_type = 0 , Chip_Capacity = 0 , ID = 0;
	 //Start communication with the flash memory by pulling the chip select low
	    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, RESET);
	    flash_spi(0x9F);  // Send JEDEC ID command (0x9F)
	    Chip_ID_L = flash_spi(0xA5);  // Receive Chip ID Low byte
	    Memory_type = flash_spi(0xA5);  // Receive Memory Type byte
	    Chip_Capacity = flash_spi(0xA5);  // Receive Chip Capacity byte
	    // End communication by pulling chip select high
	    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
	    // Combine the received bytes to form the full JEDEC ID
	    ID = Chip_ID_L | Memory_type << 8 | Chip_Capacity << 16;
	    return ID;  // Return the full JEDEC ID
	}
	// Function to retrieve the unique ID from the flash memory
	uint32_t flash_UniqueID(void)
	{
	    uint64_t UniqueID = 0;
	    // Start communication with the flash memory
	    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, RESET);
	    // Send commands to retrieve the unique ID
	    for(int i = 0; i < 5; i++)
	        flash_spi(0x4B);  // Send Read Unique ID command
	    // Collect unique ID bytes by shifting and combining them
	    for(int i = 0; i <= 64; i += 8)
```

```
            UniqueID = UniqueID | flash_spi(0xA5) << i; // Accumulate unique ID
        // End communication by pulling chip select high
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, SET);
        return UniqueID;  // Return the unique ID
    }
/* USER CODE END 1 */
```

تابع Flash_SPI تابع اصلی برای برقراری ارتباط با حافظه فلش است. نحوه کارکرد آن بدین صورت است که Data را در ورودی دریافت می‌کند و برای حافظه فلش ارسال می‌کند و داده دریافتی از حافظه فلش را نیز در خروجی توسط متغییر ret باز می‌گرداند. ورودی و خروجی این تابع متغییرهای ۸ بیتی هستند.

برای خواندن شناسه‌های حافظه فلش باید ابتدا دستورهایی را به صورت کد هگز طبق روند مخصوصی به آن ارسال شود که به ترتیب 0x9F و 0x4B را برای شناسه Chip_ID و Chip_Unique_ID ارسال می‌شود. سپس خواندن اطلاعات را از چیپ حافظه انجام داده و درون متغییرهای محلی در تابع ذخیره و در نهایت مقادیر در خروجی تابع برگردانده می‌شود. توجه کنید برای خواندن از چیپ باید کد 0xA5 را برای چیپ ارسال کنید.

تمامی مراتب موارد فوق و باقی امکانات این حافظه را می‌توان در برگه اطلاعاتی آن مشاهده کرد و برای کاربردهای بیشتر استفاده کنید. در نهایت در حلقه بینهایت بصورت پیوسته مقادیر شناسه‌ها خوانده و توسط USART به سمت کامپیوتر ارسال می‌شود.

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

    /* USER CODE BEGIN 3 */
    // Retrieve JEDEC ID and Unique ID
    Chip_ID = flash_JEDEC_ID();
    Chip_Unique_ID = flash_UniqueID();
    HAL_Delay(500);  // Wait for 500 ms
    // Format and send the chip ID over UART
    sprintf(MSG, "\nHello! Chip_ID = %d\r\n", Chip_ID);
    HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
    // Clear the message buffer
    for( uint8_t counter = 0; counter < sizeof(MSG); counter++)
        MSG[counter] = 0;

    HAL_Delay(500);  // Wait for 500 ms
    // Format and send the unique ID over UART
    sprintf(MSG, "Hello! Chip_Unique_ID = %d\r\n", Chip_Unique_ID);
    HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 100);
    // Clear the message buffer
    for( uint8_t counter = 0; counter < sizeof(MSG); counter++)
        MSG[counter] = 0;

    HAL_Delay(500);  // Wait for 500 ms
}
```

۶- تمرین: با مراجعه به دیتاشیت حافظه فلش W25Q64JV برنامه‌ای بنویسید که دیتا دلخواهی را بر روی فلش نوشته و سپس آن را بخوانید.

آزمایش ۱۱: راه‌اندازی ارتباط I2C

هدف آزمایش: در این آزمایش دانشجو با مفاهیم ارتباط I2C و نحوه راه‌اندازی آن آشنا می‌شود.

شرح آزمایش:

۱-مقدمه: I2C، که مخفف Inter-Integrated Circuit است، یک پروتکل ارتباطی سریال است که برای اتصال دستگاه‌های دیجیتال طراحی شده است. این پروتکل در اوایل دهه ۱۹۸۰ توسط شرکت NXP (که پیش از آن به نام Philips شناخته می‌شد) توسعه یافت. I2C به عنوان یک پروتکل ارتباطی دو سیمه (Two Interface Wire) نیز شناخته می‌شود و به طور گسترده در میکروکنترلرها و IC های مختلف به کار می‌رود. هدف اصلی این پروتکل تسهیل و ساده‌سازی ارتباط بین دستگاه‌های الکترونیکی در مدارهای مجتمع و سیستم‌های embedded است.

ویژگی‌های مهم ارتباط I2C:

۱- پیکربندی دوسیمه: I2C به دو خط اصلی نیاز دارد: خط داده (SDA) و خط کلاک (SCL). این مساله باعث می‌شود که I2C یک گزینه اقتصادی و کم‌هزینه برای ارتباط بین دستگاه‌ها باشد.

۲- مدهای ارتباطی: I2C یک پروتکل چند Master و چند Slave است. به این معنا که چندین دستگاه می‌توانند به عنوان Master (کنترل‌کننده) و Slave (دستگاه‌های پاسخ‌دهنده) در شبکه حضور داشته باشند. این قابلیت به سیستم اجازه می‌دهد تا به راحتی به دستگاه‌های متعدد متصل شود.

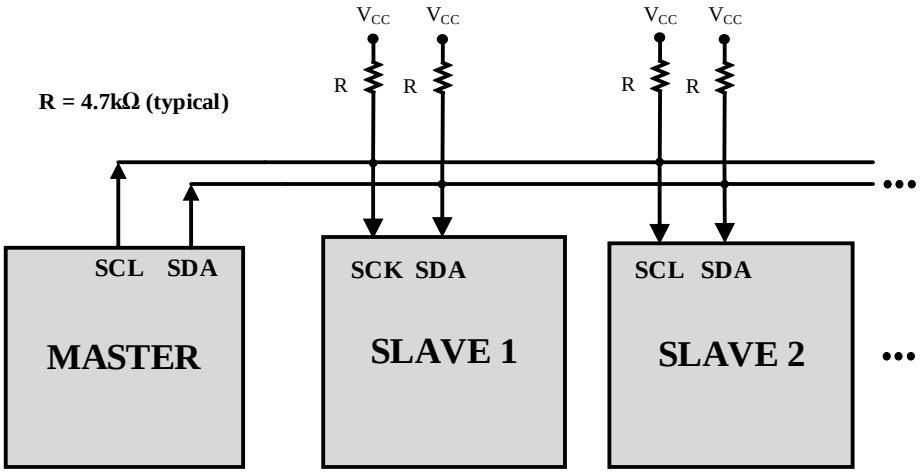
۳- سرعت‌های مختلف: پروتکل I2C در ابتدا با سرعت ۱۰۰ کیلوبیت بر ثانیه آغاز به کار کرد. اما به‌طور مداوم ارتقاء یافته و از آن زمان پنج حالت سرعتی جدید به آن اضافه شده است:

- مد استاندارد (SM): حداکثر سرعت ۱۰۰ کیلوبیت بر ثانیه
- مد سریع (FM): حداکثر سرعت ٤۰۰ کیلوبیت بر ثانیه
- مد سریع پلاس (+FM): حداکثر سرعت ۱ مگابیت بر ثانیه
- مد خیلی سریع (HS): حداکثر سرعت ۳.٤ مگابیت بر ثانیه
- مد بسیار سریع (UF): حداکثر سرعت ۵ مگابیت بر ثانیه

٤- معماری: یکی از ویژگی‌های منحصر به فرد I2C ساختار Open-Drain آن است. این ویژگی به دستگاه‌ها اجازه می‌دهد تا به طور همزمان بر روی خط داده ارسال و دریافت کنند، بدون اینکه تداخل ایجاد شود. هر دستگاه می‌تواند با ایجاد یک سیگنال پایین (۰) بر روی خط، کنترل آن را بر عهده بگیرد.

۵- سلامت داده (Data Integrity): پروتکل I2C از سیستم تأیید دو طرفه (ACK/NACK) استفاده می‌کند تا اطمینان حاصل کند که داده‌ها به درستی منتقل شده‌اند. پس از هر انتقال داده، دستگاه دریافت‌کننده باید یک بیت تأیید (ACK) ارسال کند تا تأیید کند که داده به درستی دریافت شده است.

ارتباط I2C از طریق دو سیم SDA و SCL برقرار می‌شود که به ترانزیستورهایی متصل هستند که به صورت Open-Drain عمل می‌کنند. برای اینکه این خطوط بتوانند به حالت بالا (۱) برگردند، مقاومت‌های کششی (Pull-up resistors) به منبع ولتاژ متصل می‌شوند. در شرایط عادی، این مقاومت‌ها خط را به مقدار Vcc می‌برند و زمانی که یک دستگاه خروجی را فعال می‌کند، آن خط را به زمین (GND) متصل می‌کند و سیگنال را به حالت پایین (۰) تغییر می‌دهد.

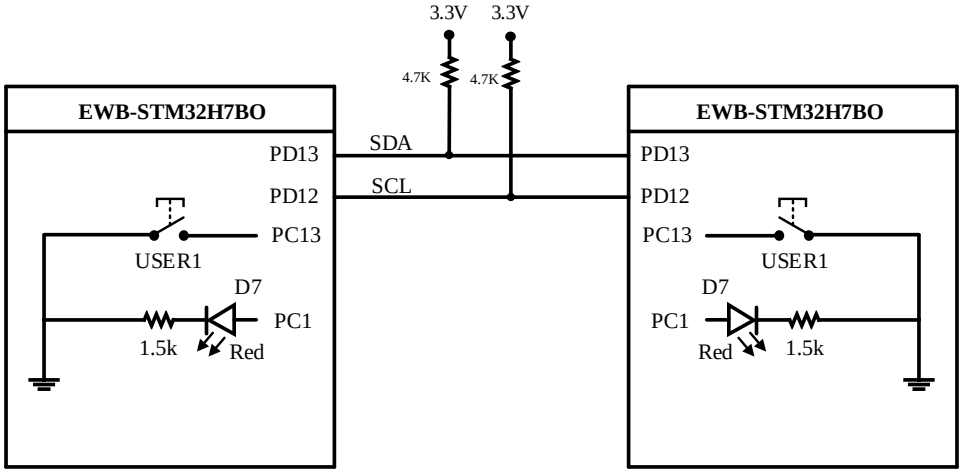


شکل(۱): نحوه اتصال Master و چند Slave با یک خط ارتباطی I2C

روند ارتباط در I2C با یک وضعیت شروع (START) آغاز می‌شود. پس از آن، آدرس دستگاه Slave و بیت‌های R/W برای تعیین نوع انتقال (خواندن یا نوشتن) ارسال می‌شوند. در ادامه، یک بیت تأیید از سمت دستگاه Slave ارسال می‌شود. اگر Slave موجود باشد و آماده به کار باشد، داده‌ها منتقل می‌شوند و در نهایت، یک وضعیت پایان (STOP) ارتباط را خاتمه می‌دهد. سپس اگر Slave وجود داشته باشد و سالم باشد، یک پیام تایید مثبت (بصورت ارسال یک بیت ACK) به سمت Master ارسال می‌کند و در غیر این صورت پیام تایید را منفی درنظر می‌گیرد (NACK). بعد از آن یک بایت از داده به همراه یک پیام تایید از Slave ارسال می‌شود و در آخر Master می‌تواند با اعمال وضعیت پایان (تغییر وضعیت SDA هنگامی که SCL وضعیت ۱ دارد) به ارتباط خاتمه دهد. ترتیب این روند را بصورت خلاصه میتوان به این صورت بیان کرد :

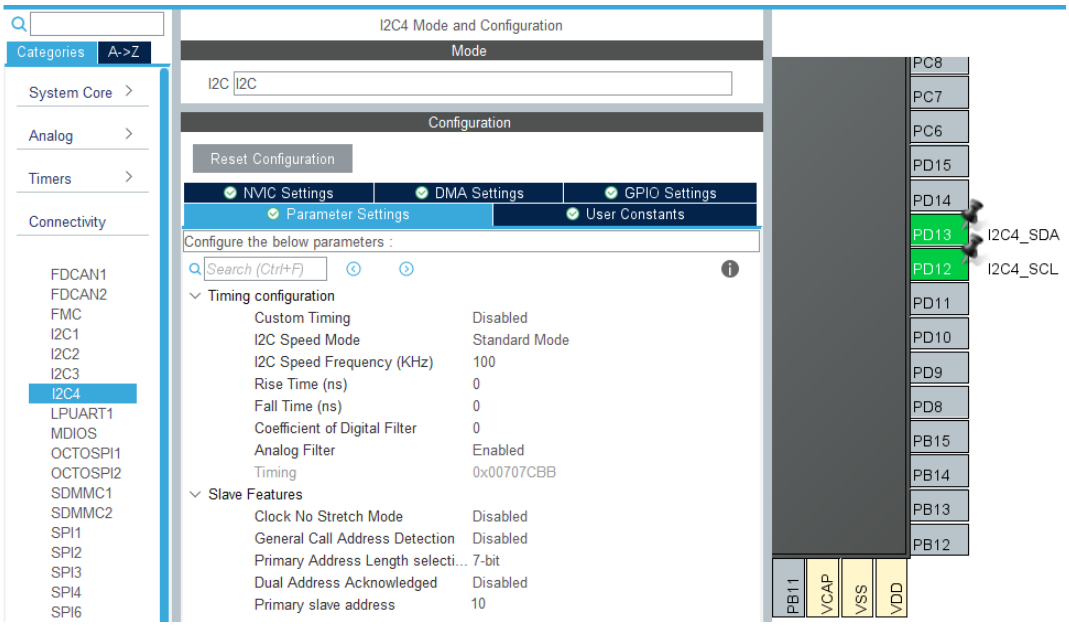
۱. وضعیت شروع (S)
۲. آدرس +R/W
۳. تاییدیه مثبت یا منفی (Ack | NAck)
٤. وضعیت شروع مجدد (Sr)
۵. وضعیت پایان (P)

۲-شماتیک آزمایش: در این آزمایش از دو برد دیسکاوری استفاده می‌شود که از طریق پروتکل I2C به یکدیگر متصل شده‌اند. هدف این آزمایش، کنترل روشن و خاموش شدن LED یک برد، توسط کلیدی است که به پایه PC13 برد دیگر متصل شده است. اتصال دو برد به یکدیگر باید با استفاده از پروتکل I2C انجام شود. همچنین لازم است دو پین ارتباطی I2C با استفاده از دو عدد مقاومت ٤.٧کیلواُهم به صورت Pull-up بسته شوند. جزئیات این اتصالات در شماتیک زیر نشان داده شده است.



شکل(۳): شماتیک مربوط به آزمایش I2C

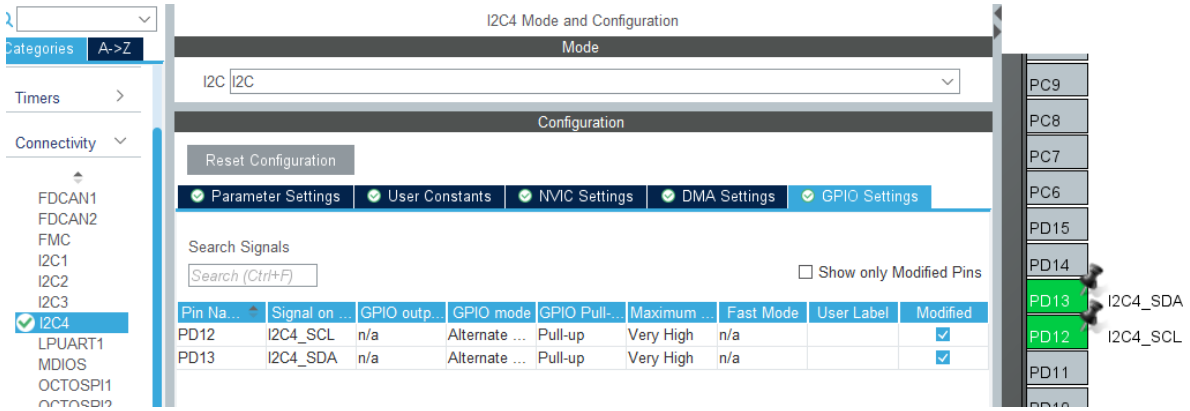
۳- تنظیمات نرم‌افزار STM32CubeIDE: پروژه را طبق روش مرسوم گذشته ایجاد کنید. بخش دیباگ را بر روی Serial wire تنظیم کنید و کلاک سیستم را بر روی HSE و مد کریستال خارجی تنظیم کنید. از سربرگ Connectivity زیر شاخه I2C4 را انتخاب کنید. سپس تنظیمات I2C و پایه‌های PD13 و PA12 را مطابق تصویر زیر به عنوان I2C4_SDA و I2C4_SCL اعمال کنید.



شکل(٤): تنظیمات واحد I2C4 در نرم‌افزار

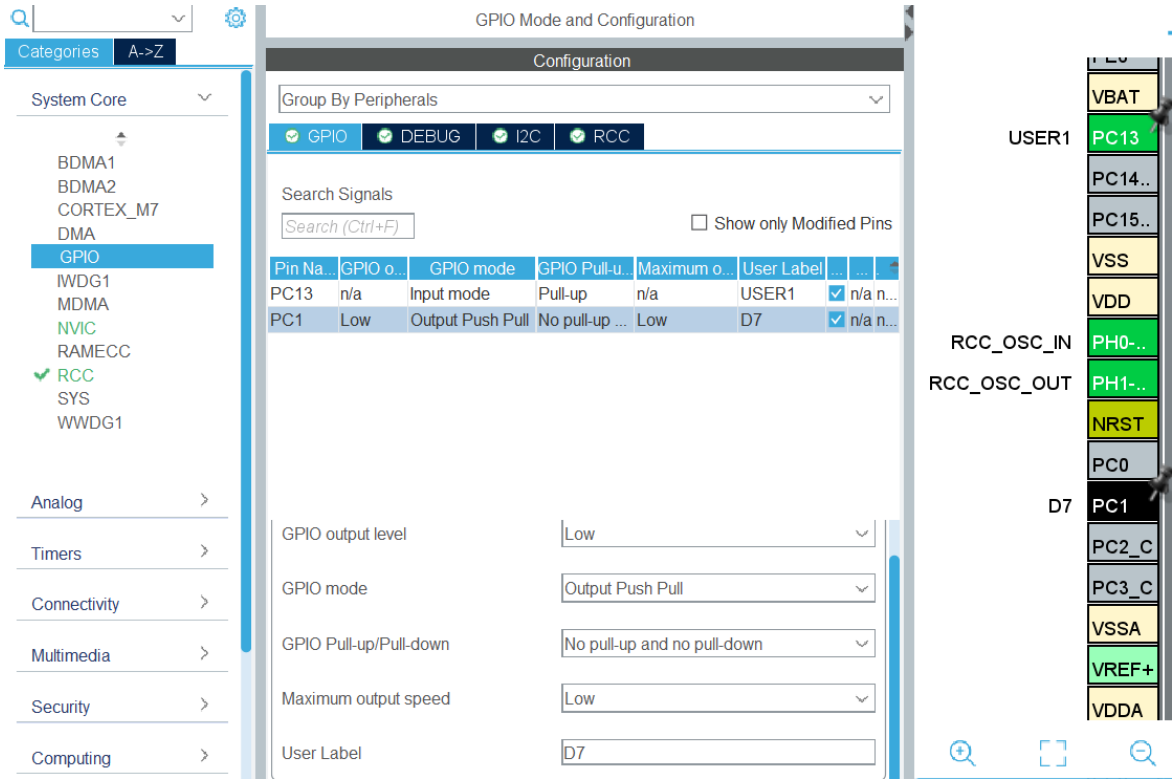
در حالت کلی برای اتصال چند Master بر روی یک خط باس نیاز است تا Slave address را نیز برای هر دستگاه متفاوت تنظیم کنیم اما در این آزمایش دو دستگاه بیشتر بر روی یک باس قرار ندارند بنابرین نیازی به تنظیم Slave address های متفاوت نیست و آدرس ۱۰ برای هر دو دستگاه اتخاذ می‌گردد.

مطابق تصویر زیر تنظیمات پین های مربوط به واحد I2C را نیز اعمال کنید.



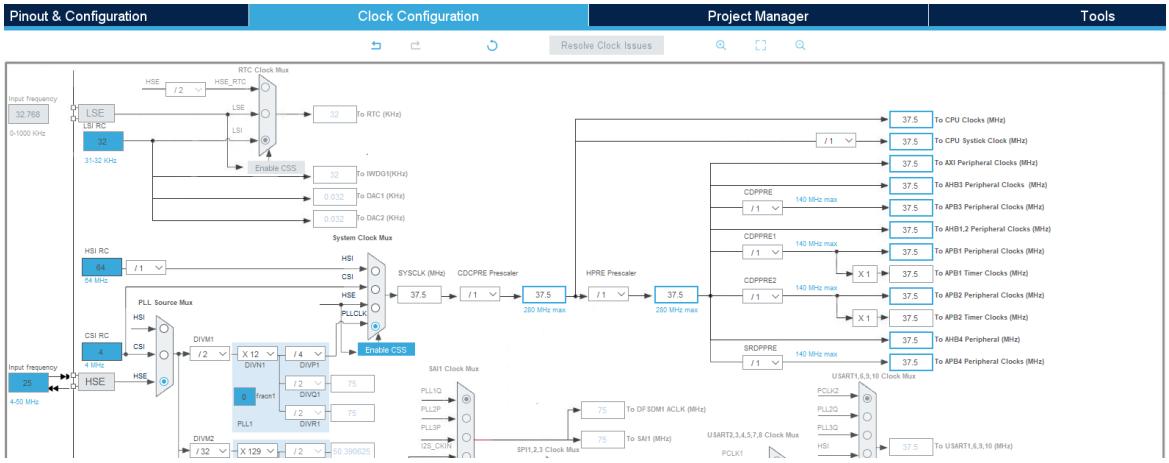
شکل(۵): تنظیمات پین‌های ارتباط I2C در واحد GPIO

یکی از مزیت‌های این پروتکل Open Drain بودن آن است و سبب می‌شود تا در صورت تداخل دیتا به دستگاه‌ها آسیبی وارد نشود، اما از سوی دیگر وجود مقاومت‌های Pull Up را ضروری می‌کند. در ادامه مطابق تصاویر زیر پین‌های مربوط به LED D7 و کلید فشاری USER1 را در واحد GPIO تنظیم کنید.



شکل(٦): تنظیمات پین‌های ارتباط I2C در واحد GPIO

در انتها تنظیمات کلاک میکروکنترلر را مانند تصویر زیر اعمال کنید.



شکل(۷): تنظیمات کلاک میکروکنترلر در نرم‌افزارCubeMx

پس از اعمال تنظیمات با فشردن Ctrl+S برنامه را Generate نمایید.

٤- معرفی توابع مورد استفاده: در این بخش به معرفی دو تابع پرداخته می‌شود.

HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout)

این تابع جهت ارسال داده توسط این پروتکل استفاده می‌شود. ورودی‌های آن علاوه بر آدرس استراکچر مربوط به واحد I2C مورد استفاده، آدرس دستگاه Slave، رشته داده‌ها، اندازه رشته و حداکثر مدت زمان مجاز برای ارسال داده است.

HAL_I2C_Slave_Receive(I2C_HandleTypeDef *hi2c, uint8_t *pData, uint16_t Size,uint32_t Timeout)

این تابع جهت دریافت داده‌ها در مد Slave و بدون استفاده از اینتراپت استفاده می‌شود. آرگومان اول آدرس استراکت مربوط به واحد I2C مورد استفاده است، آرگومان دوم رشته‌ای برای ذخیره داده دریافتی و آرگومان سوم طول این رشته است. آرگومان چهارم نیز حداکثر مدت زمان مجاز برای دریافت داده است.

٥- برنامه‌نویسی آزمایش: در این ارتباط به دلیل یکسان‌بودن برنامه‌ها هر دو برد دارای یک Slave Address هستند و به همین منظور یک آدرس مشترک برای آن‌ها تعریف می‌کنیم.

```
/* USER CODE BEGIN PD */
#define COMMON_ADDRESS 10
/* USER CODE END PD */
```

برای ارسال و دریافت نیز به دو بافر نیازمندیم اما به این دلیل که هدف تنها روشن و خاموش نمودن یک LED است تنها یک بایت برای این بافرها کافی است.

```
/* USER CODE BEGIN PV */
uint8_t tx_data[1] = "1";
uint8_t rx_data[1] = {0};
/* USER CODE END PV */
```

کد زیر را در while موجود در تابع main بنویسید.

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        //Check if USER1 button is pressed
        if (!HAL_GPIO_ReadPin(USER1_GPIO_Port, USER1_Pin)) {
            //Send data via I2C in Master mode
           if (HAL_I2C_Master_Transmit(&hi2c4, COMMON_ADDRESS << 1, tx_data, 1,
HAL_MAX_DELAY) == HAL_OK){
              HAL_Delay(10); //Debounce delay
              while (!HAL_GPIO_ReadPin(USER1_GPIO_Port, USER1_Pin)) {}//Wait for
button release
              HAL_Delay(10); //Additional debounce delay
           }
        }
        //Receive data via I2C in Slave mode
        if (HAL_I2C_Slave_Receive(&hi2c4, rx_data, 1, 10) == HAL_OK) {
           if (rx_data[0] == '1') {
              HAL_GPIO_TogglePin(D7_GPIO_Port, D7_Pin); //Toggle LED if received
data is '1'
              rx_data[0] = 0; //Clear received data
           }
              HAL_Delay(100); //Small delay for next operation
        }
            /* USER CODE END WHILE */
```

در این برنامه داده به روش polling در حال ارسال و دریافت است، هنگامی که کلید فشرده شود میکروکنترلر در مد Master محتوای بافر tx را ارسال می‌کند. در همین حین برنامه روبرو در مد Slave، به طور مداوم ورود دیتا را بررسی می‌کند و زمانی که دیتا وارد شود آن را در بافر rx ذخیره می‌کند.

سپس در صورتی که مقدار دریافت شده '1' باشد وضعیت LED را Toggle می‌کند.

۶- تمرین: LED D8 را فعال نمایید و از آن برای نمایش وضعیت ارتباط، استفاده کنید.

*LED D8 به پایه PA2 برد متصل است.

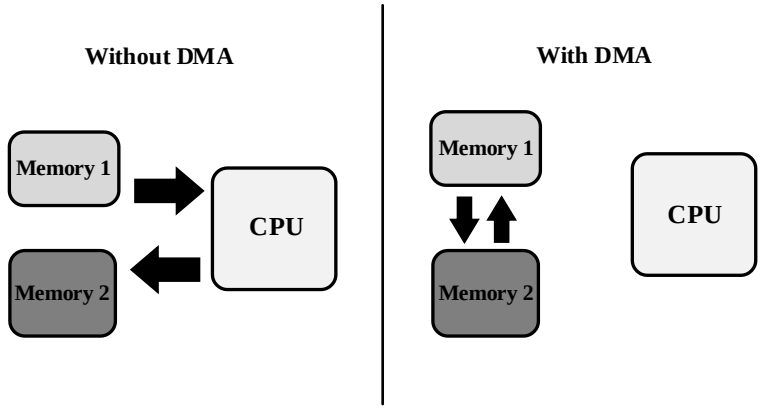
آزمایش ۱۲: آشنایی با DMA

هدف آزمایش: در این آزمایش دانشجو با مفهوم DMA و نحوه راه‌اندازی آن آشنا می‌شود.

شرح آزمایش:

۱-مقدمه: امروزه سرعت انتقال اطلاعات به عوامل کلیدی در طراحی سیستم‌های embedded تبدیل شده است. میکروکنترلرها به عنوان قلب این سیستم‌ها، نیاز به رویکردهایی دارند که بتوانند بهینه‌سازی قابل توجهی در عملکرد و مصرف منابع ایجاد کنند. یکی از تکنیک‌های بسیار مؤثر در این زمینه، DMA یا Direct Memory Access است.

تکنولوژی DMA به میکروکنترلرها این امکان را می‌دهد که انتقال داده‌ها بین واحدهای جانبی (مثل حسگرها، ارتباطات سریال و دیگر دستگاه‌ها) و حافظه بدون دخالت پردازنده انجام شود. این فرآیند به معنی دسترسی مستقیم و بی‌واسطه به حافظه است و به توسعه‌دهندگان این امکان را می‌دهد که بار پردازشی CPU را کاهش دهند و سیستم را بهبود بخشند. تصویر زیر به طور واضح وظیفه DMA را به تصویر می‌کشد که برای فهم بهتر می‌توان به جای Memory 1 و Memory 2 ابزارهای جانبی مانند Spi، Usart، ADC، DAC و... قرار داد.



شکل(۱): تفاوت تبادل دیتا در دوحالت استفاده از DMA و بدون آن

اهمیت و مزایای استفاده از DMA :

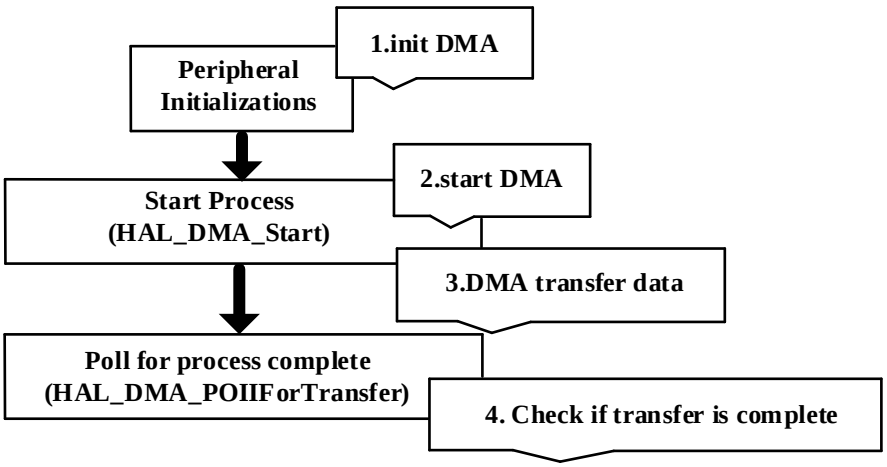
۱- کاهش بار پردازشی: با انتقال داده‌ها به طور خودکار بدون نیاز به پردازشCPU ، بار پردازشی پردازنده به طور چشمگیری کاهش می‌یابد. این موضوع به پردازنده این امکان را می‌دهد که بر روی دیگر وظایف متمرکز شود و به این ترتیب، عملکرد کلی سیستم بهبود پیدا می‌کند.

۲- افزایش سرعت انتقال داده‌ها DMA: می‌تواند داده‌ها را با سرعت بیشتری نسبت به پردازنده انتقال دهد. این امر به ویژه در پروژه‌هایی که نیاز به پردازش‌های سریع و مداوم داده دارند، اهمیت دارد.

۳- مدیریت چندوظیفه‌ای: با استفاده از DMA، می‌توان به طور همزمان چندین عملیات را انجام داد. به عنوان مثال، در زمانیکه داده‌ها از یک حسگر دریافت می‌شود، پردازنده می‌تواند همزمان به پردازش داده‌های دیگر بپردازد.

در میکروکنترلرهای STM32، واحد DMA به دو بخش اصلی تقسیم می‌شود که شامل DMA1 با ۷ کانال و DMA2 با 5 کانال است. هر کانال می‌تواند به صورت مستقل عمل کند و عملیات انتقال داده را بین امکانات جانبی و حافظه به راحتی مدیریت کند. به این ترتیب، امکان انجام چندین انتقال به‌طور همزمان وجود دارد و این کارایی کلی سیستم را به شدت افزایش می‌دهد.

روند DMA بصورت کلی به شکل زیر انجام میپذیرد:



شکل(۲): روند استفاده از DMA با استفاده از توابع HAL

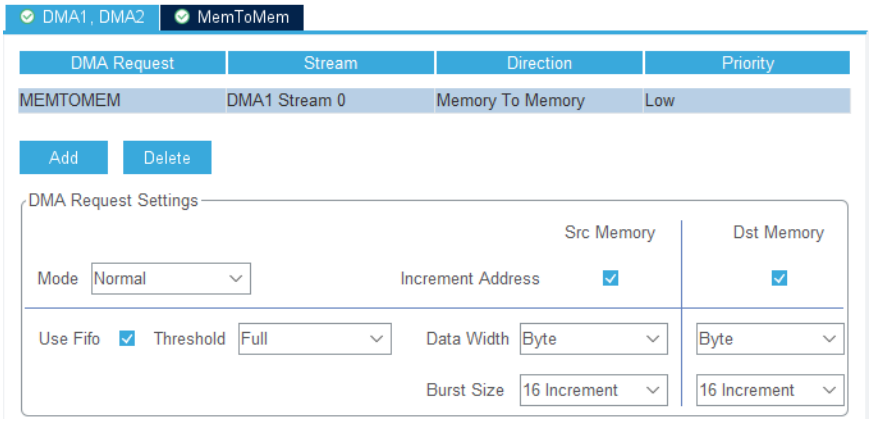
ابتدا تنظیمات مربوطه روی DMA انجام شده و پس از مقداردهی اولیه، این واحد فعال می‌شود. در مرحله بعدی، انتقال داده آغاز شده و پس از تکمیل انتقال، اطمینان حاصل می‌شود که داده به‌درستی منتقل شده است. یکی از چالش‌های مهم در مدیریت DMA، بررسی وقوع خطا در فرآیند انتقال و نیاز به هماهنگی در زمان‌بندی است. در سیستم‌های بلادرنگ (Real-Time)، طراحی درست این بخش اهمیت بالایی دارد و موجب بهینه‌سازی عملکرد کلی پردازنده خواهد شد.

در این آزمایش قصد داریم با استفاده از DMA و تابع memcpy، زمان انتقال یک آرایه به یک آرایه دیگر را اندازه‌گیری کنیم و آن را بر روی پورت سریال نمایش دهیم.

۲-شماتیک آزمایش: در این آزمایش کافی است پورت خروجی سریال را با کابل مربوطه به کامپیوتر متصل کنید و با فشردن کلید RESET بر روی برد نتایج را مشاهده نمایید.

۳- تنظیمات نرم‌افزار STM32CubeIDE: تنظیمات اولیه کلاک سیستم را تغییر ندهید و تنظیمات ارتباط USART را مطابق آزمایش های گذشته اعمال کنید سپس باقی تنظیمات را مطابق مراحل آزمایش پیش ببرید.

از سربرگ DMA در منوی System Core گزینه Add را کلیک کنید و از منوی بازشده گزینه MEMTOMEM را انتخاب نمایید. با انتخاب این گزینه تنظیمات مربوطه باز می‌شود، سپس آن را مطابق تصویر زیر اصلاح کنید.



شکل(۳): تنظیمات واحد DMA در نرم‌افزار

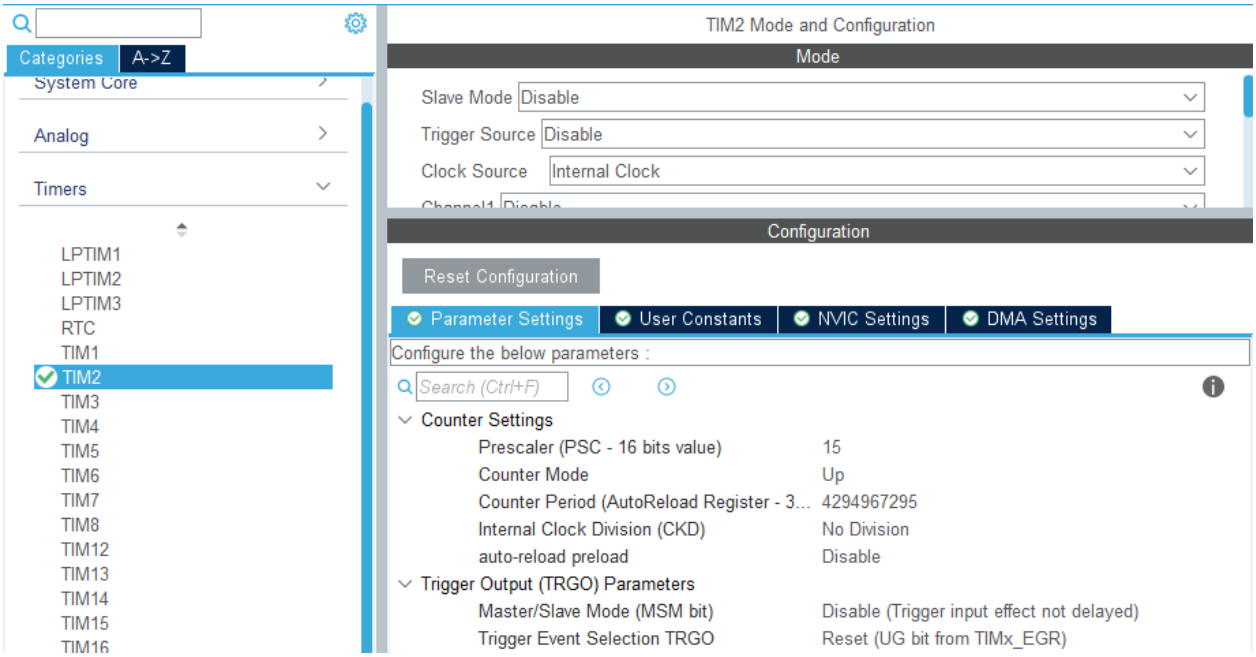
در گزینه های قابل تنظیم این واحد موارد تازه‌ای مشاهده می‌شود که در ادامه توضیح داده خواهد شد:

- Increment Address: در صورت فعال بودن این گزینه، آدرس مبدأ و مقصد پس از هر انتقال، مقداردهی شده و به میزان مشخصی افزایش می‌یابد. همچنین، مقدار Burst Size تعیین می‌کند که چه تعداد داده در هر انتقال اضافه شود. این گزینه در جابه‌جایی مقادیر آرایه‌ها بسیار مفید است.
- Data Width: این گزینه مشخص می‌کند که اندازه (عرض) داده‌ها در هر انتقال چقدر باشد. مقدار آن می‌تواند بایت (Byte)، نیم‌کلمه (Half Word) یا کلمه (Word) باشد.
- FIFO (First In, First Out): این واحد، بافری داخلی برای انتقال داده‌ها فراهم می‌کند. استفاده از FIFO باعث افزایش کارایی انتقال داده می‌شود، زیرا ابتدا داده‌ها ذخیره شده و سپس خوانده می‌شوند.
- Mode: این گزینه در انتقال هایی کاربرد دارد که نیاز داریم عمل انتقال به طور پیوسته توسط این واحد صورت گیرد.

برای اندازه‌گیری زمان انتقال داده‌ها به طور دقیق، نیاز است تا از تایمرها نیز استفاده شود. به همین دلیل تایمر دو طوری تنظیم می‌شود تا هر بار افزوده شدن به مقدار شمارشگر آن، بیانگر ۲۵۰ نانوثانیه باشد.

محاسبات این بخش وابسته به مقدار PSC است. فرکانس شمارش تایمر برابر با تقسیم فرکانس ورودی واحد تایمر بر مقدار تنظیم‌شده در رجیستر مقسم آن است. مقدار این رجیستر با توجه به حداقل زمان موردنیاز برای شمارش (۲۵۰ نانوثانیه) و فرکانس ورودی تایمر (٦٤ مگاهرتز) برابر با ١٦ محاسبه می‌شود. با این حال، عددی که باید در رجیستر تنظیم شود، با توجه به فرمول محاسبه تایمر، یک واحد کمتر از مقدار به‌دست‌آمده خواهد بود.

تنظیمات تایمر دو را مطابق تصویر زیر وارد کنید.



شکل(٤): تنظیم واحد TIM2 جهت برآورد زمان انتقال اطلاعات

برنامه را Generate کنید و وارد محیط برنامه نویسی شوید.

٤- معرفی توابع مورد استفاده: در این برنامه تنها یک تابع مربوط به واحد DMA بوده است که در ادامه شرح داده خواهد شد.

```
HAL_DMA_Start(DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t DstAddress,
uint32_t DataLength)
```

تابع HAL_DMA_Start یکی از توابع کتابخانه HAL برای راه‌اندازی و شروع عملیات انتقال داده توسط واحد DMA در میکروکنترلرهای STM32 است. این تابع نقش تنظیم رجیسترهای لازم برای شروع عملیات DMA را ایفا می‌کند و با مشخص کردن آدرس مبدا، مقصد و تعداد داده‌هایی که باید منتقل شوند، انتقال را آغاز می‌کند. آرگومان های این تابع به شرح زیر است:

- hdma: اشاره‌گر به ساختار کنترل DMA که شامل تنظیمات کانال و پیکربندی‌های اولیه است.
- SrcAddress: آدرس مبدأ داده‌ها (RAM، Flash یا رجیستر پیرامونی).
- DstAddress: آدرس مقصد داده‌ها (RAM، Flash یا رجیستر).
- DataLength: تعداد داده‌هایی که باید منتقل شوند (بر حسب واحد داده: بایت، نیم‌کلمه، یا کلمه).

در برنامه نیز رجیستری برای تشخیص وضعیت انتقال واحد DMA استفاده شده است که طبق H7 Reference Manual به صورت زیر تشریح می‌شود:

16.5.1 DMA low interrupt sataus register (DMA_LISR)

Address offset:0x00

Reset value:0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	TCIF3	HTIF3	TEIF3	DMEIF3	Res	FEIF3	TCIF2	HTIF2	TEIF2	DMEIF2	Res	FEIF2
				r	r	r	r		r	r	r	r	r		r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res	Res	Res	TCIF1	HTIF1	TEIF1	DMEIF1	Res	FEIF0	TCIF0	HTIF0	TEIF0	DMEIF0	Res	FEIF0
				r	r	r	r		r	r	r	r	r		r

Bits 31:28,15:12 Reserved, must be kept at reset value

Bits 27 ,21 ,11 ,5 TCIF[3:0] :stream x transfer complete interrupt flag (x = 3 to 0)
This bit is set by hardware. It is cleaed by software writing 1 to the corresponding bit in the DMA_LIFCR register
0:no transfer complote event on stream x
1:a transfer complote event occurred stream x

شکل(۵): رجیستر LISR یا پرچم‌های واحد DMA

طبق توضیحات مربوط به بیت TCIF، این بیت زمانی یک می‌شود که عملیات انتقال توسط واحد DMA به پایان رسیده باشد.

۵- برنامه‌نویسی آزمایش:

ابتدا کتابخانه string.h را به دلیل استفاده از تابع memset به برنامه اضافه کنید.

```
/* USER CODE BEGIN Includes */
#include "string.h"
/* USER CODE END Includes */
```

متغیرهای مورد استفاده در این آزمایش را نیز پیش از تابع main تعریف می‌کنیم:

```
/* USER CODE BEGIN PV */
uint8_t array_1[1024];
uint8_t array_2[1024];
uint8_t MSG[50];
uint32_t time,error_flag = 0;
/* USER CODE END PV */
```

در برنامه دو آرایه ۱ کیلوبایتی برای عملیات انتقال در نظر گرفته شده است و سپس یک آرایه برای ارسال داده‌ها توسط واحد UART تعریف شده است. متغیر time برای ذخیره مقدار شمارش‌شده و متغیر error_flag به جهت تشخیص وجود خطا پس از انتقال، تعریف شده است.

خطوط زیر را نیز پیش از حلقه while موجود در تابع main اضافه نمایید.

```
  /* USER CODE BEGIN 2 */
  // Initialize array_1 with the value 25
  memset(array_1, 25, 1024);
  // Reset the timer
  TIM2->CNT = 0;
  // Start the timer
  TIM2->CR1 |= 0x01;
  // Start data transfer using DMA
```

```
  HAL_DMA_Start(&hdma_memtomem_dma1_stream0, array_1,array_2, 1024);
  // Wait for DMA transfer to complete
  while(!(DMA1->LISR & (1 << 5))){}
  // Stop the timer
  TIM2->CR1 &= ~0x01;
  // Save the execution time of the DMA operation
  time = TIM2->CNT;

  // Verify the copied data
  for(int i = 0; i < 1024; i++)
      if(array_2[i] != array_1[i]) { error_flag = 1; }
  // If the transfer was successful, display the execution time
  if(!error_flag) {
      sprintf(MSG, "Took time with DMA = %fs\r\n", time * 0.00000025);}
  // Otherwise, display an error message
  else {
      sprintf(MSG, "The copy operation was not successful (DMA)\r\n",time *
0.00000025);
      error_flag = 0;
  }
  // Send the message via UART
  HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 1000);

  // Initialize array_1 with the value 63
  memset(array_1, 63, 1024);
  // Reset the timer
  TIM2->CNT = 0;
  // Start the timer
  TIM2->CR1 |= 0x01;
  // Copy data using memcpy
  memcpy(array_2, array_1, 1024);
  // Stop the timer
  TIM2->CR1 &= ~0x01;
  // Save the execution time of the memcpy operation
  time = TIM2->CNT;

  // Verify the copied data
  for(int i = 0; i < 1024; i++)
      if(array_2[i] != array_1[i]) { error_flag = 1; }
  // If the transfer was successful, display the execution time
  if(!error_flag) {
      sprintf(MSG, "Took time with memcpy = %fs\r\n", time * 0.00000025);
  }
  // Otherwise, display an error message
  else {
      sprintf(MSG, "The copy operation was not successful (memcpy)\r\n",time *
0.00000025);
      error_flag = 0;}
  // Send the message via UART
  HAL_UART_Transmit(&huart1, MSG, sizeof(MSG), 1000);
  /* USER CODE END 2 */
```

در این برنامه ابتدا به آرایه array_1 مقدار داده می‌شود، سپس پیش از شروع انتقال، تایمر دو نیز به صورت رجیستری راه‌اندازی می‌شود. پس از انتقال، تایمر غیرفعال می‌شود و با خواندن رجیستر CNT و ضرب آن در مقدار ۲۵۰ نانوثانیه مدت زمان انتقال به دست می‌آید. سپس با استفاده از یک حلقه صحت انتقال بررسی می‌شود و در صورت درستی، مقدار به دست آمده در خروجی چاپ خواهد شد. برنامه یکبار با استفاده از

DMA این عمل را انجام می‌دهد و بار دیگر با استفاده از تابع memcpy که تابع بهینه‌ای است که برای این امر درنظر گرفته شده است.

٦- تمرین: برنامه را طوری تغییر دهید که عملیات انتقال بدون استفاده از توابع صورت گیرد، سپس مدت زمان اندازه‌گیری شده را در خروجی چاپ نمایید.

آزمایش ۱۳: پروژه ۱– طراحی و ساخت دستگاه مولد موج

۱– مقدمه: دستگاه‌های مولد موج (Waveform Generators) به عنوان ابزارهای اساسی در حوزه الکترونیک و مهندسی برق، نقش مهمی در تولید و مدیریت سیگنال‌های الکتریکی ایفا می‌کنند. این تجهیزات قادرند انواع مختلف سیگنال‌ها را با اشکال موج متفاوتی مانند سینوسی، مربعی و مثلثی تولید کنند. استفاده از مولدهای موج به مهندسان و پژوهشگران این امکان را می‌دهد که به‌طور دقیق‌تر عملکرد مدارها را بررسی کرده و شرایط مختلف آزمایشی را شبیه‌سازی نمایند. این دستگاه‌ها به‌ویژه در آزمایشگاه‌ها و پروژه‌های تحقیقاتی کاربرد فراوانی دارند. آن‌ها می‌توانند به‌عنوان منبع سیگنال‌های ورودی در تست و بهینه‌سازی مدارها، تحلیل رفتار سیستم‌های الکترونیکی و ارزیابی پاسخ‌های مختلف سیستم‌ها به سیگنال‌های متنوع به کار روند. علاوه بر این، مولدهای موج ابزاری مهم در امر آموزش هستند که به دانشجویان اجازه می‌دهند با مفاهیم پایه‌ای الکترونیک، مانند فرکانس، دامنه و تحلیل شکل موج، آشنا شوند. این آموزش‌ها نه‌تنها درک نظری آن‌ها را عمیق‌تر می‌کنند، بلکه مهارت‌های عملی و تجربی لازم برای طراحی مدارها و تحلیل سیگنال‌ها را نیز تقویت می‌کنند.

۲– شرح پروژه: در این پروژه، دانشجویان به طراحی و پیاده‌سازی یک مولد موج ساده خواهند پرداخت. هدف این پروژه، تسهیل یادگیری و تقویت مفاهیم کلیدی الکترونیک از طریق برنامه‌نویسی و طراحی مدار با استفاده از میکروکنترلر برای ایجاد دستگاه‌های مولد موج است.

۳– جزئیات پروژه: در این پروژه، دانشجویان باید موارد زیر را طراحی و پیاده‌سازی کنند:

۳–۱– تولید شکل موج‌های مختلف:

- موج مربعی: این نوع موج دارای تغییرات ناگهانی در ولتاژ است و در بسیاری از کاربردها مانند کلیدزنی و کنترل موتور یا شدت نور به کار می‌رود.
- موج سینوسی: موجی پیوسته و بدون تغییر ناگهانی است که برای شبیه‌سازی سیگنال‌های صوتی و RF استفاده می‌شود.
- موج مثلثی: این موج به‌دلیل ویژگی‌های خاصش، می‌تواند در تحلیل‌های فرکانسی و درون‌مداری مورد استفاده قرار گیرد.

۳–۲– تغییر نوع موج:

- مدار طراحی شده باید قابلیت تغییر بین انواع موج‌ها (مربعی، سینوسی و مثلثی) با استفاده از کلید تعبیه شده را داشته باشد.

۳-۳- تنظیم دامنه و فرکانس:

- دانشجویان باید بتوانند قابلیت تنظیم دامنه (ولتاژ) موج تولید شده را بین ۰ تا ۳.۳ ولت را ایجاد کنند.
- دانشجویان باید بتوانند قابلیت تنظیم فرکانس بین ۱ هرتز تا ۱ کیلوهرتز را ایجاد کنند.

۳-٤- نمایش اطلاعات:

- دانشجویان باید بتوانند قابلیت نمایش اطلاعات شامل نوع موج، دامنه و فرکانس بر روی یک صفحه نمایش LCD (نوع LCD اهمیتی ندارد) را ایجاد کنند.

۳-۵- اجزای لازم:

- دانشجویان می‌توانند با بردهای موجود در آزمایشگاه سیستم‌های دیجیتال ۲ و مدارهای جانبی آزمایش‌های قبل پروژه را پیاده‌سازی کنید. تنها قطعه مورد نیاز LCD مورد نظر است که انتخاب آن به دانشجو بستگی دارد.
- دانشجویان می‌توانند برای بهبود عملکرد خود از نرم‌افزارهای شبیه‌ساز نیز استفاده کنند.

* در انتهای پروژه تهیه گزارش و ارائه آن الزامی است. در هنگام جمع‌آوری نتایج، به‌جز اطلاعات دقیق درباره عملکرد مدار، چالش‌ها و مسائلی که در طی انجام پروژه با آن‌ها مواجه شده‌اید را نیز مستند کنید. این اطلاعات می‌تواند برای تحلیل نهایی و ارائه مفید باشد.

* طراحی مدارهای مورد نیاز و برنامه کاملا به عهده دانشجو است و بنا به صلاحدید مدرس می‌تواند بخشی از نمره نهایی آزمایشگاه باشد.

آزمایش ۱۴: پروژه ۲– طراحی و ساخت دستگاه ضبط صوت دیجیتال

۱– مقدمه: دستگاه‌های ضبط صوت به عنوان ابزار اصلی در ثبت و ذخیره‌سازی صدا، تاریخچه‌ای طولانی و پرکاربرد دارند. این دستگاه‌ها از زمان اولین ضبط‌های آنالوگ تغییرات زیادی کرده و به تکامل دیجیتال رسیده‌اند. انواع مختلفی از دستگاه‌های ضبط صوت وجود دارد که هر کدام ویژگی‌ها و کاربردهای خاص خود را دارند.

دستگاه‌های آنالوگ: این دسته از دستگاه‌ها با استفاده از نوارهای مغناطیسی یا دیسک‌های وینیل کار می‌کنند و سیگنال‌های صوتی را به صورت فیزیکی ثبت می‌کنند. کیفیت صدا و امکان ویرایش محدودتر از دستگاه‌های دیجیتال است، اما هنوز هم برخی از کاربران به این روش‌ها علاقه‌مندند.

دستگاه‌های دیجیتال: در فناوری دیجیتال دستگاه‌ها با تبدیل سیگنال صوتی به داده‌های دیجیتال، امکان ضبط و ویرایش صدا با کیفیت بسیار بالا و بدون افت کیفیت را فراهم می‌کنند. از مهم‌ترین مزایای دستگاه‌های دیجیتال می‌توان به ذخیره‌سازی آسان، قابلیت ویرایش بدون افت کیفیت و دسترسی سریع به محتوای ضبط شده اشاره کرد.

دستگاه‌های ضبط صوت در حوزه‌های مختلفی از جمله رسانه، موسیقی، آموزش، پزشکی و امنیت کاربرد دارند؛ از یادداشت‌برداری صوتی در آموزش گرفته تا خلق آثار هنری در موسیقی، که نقشی کلیدی در یادگیری و تولید محتوا ایفا می‌کنند.

۲– شرح پروژه: این پروژه به طراحی و ساخت یک دستگاه ضبط صوت دیجیتال متمرکز است که با استفاده از ماژول KY-35 به عنوان میکروفون، آمپلی فایر PAM8403 و بلندگوی ۸ اهم ۰.۵ وات، قابلیت ضبط و پخش صدا را فراهم کند. کاربران قادر خواهند بود با استفاده از دو کلید مجزا، به ضبط و پخش صدا بپردازند. علاوه بر این، سیستم دارای نشانگرهای LED برای نمایش وضعیت ضبط و پخش خواهد بود. این دستگاه از امکان ضبط صدای ۲۰ تا ۳۰ ثانیه‌ای برخوردار است و در صورت فشردن مجدد کلید ضبط، قابلیت جایگزینی صوت قبلی را دارد. هدف اصلی این پروژه ایجاد بستری برای یادگیری عملی و کسب مهارت‌های تکنیکی در زمینه الکترونیک و برنامه‌نویسی میکروکنترلرها، به همراه پردازش سیگنال‌های صوتی است، به گونه‌ای که دانشجویان بتوانند ایده‌های خود را به واقعیت تبدیل کرده و درک عمیق‌تری از کاربردهای میکروکنترلرها در پروژه‌های عملی پیدا کنند.

۳– جزئیات پروژه: برای پیاده‌سازی دستگاه ضبط صوت دیجیتال، موارد زیر با دقت در نظر گرفته شده است:

۱– ورودی صدا: الف)ماژول KY-35 که به عنوان ورودی صدا عمل می‌کند و قادر به دریافت و تقویت امواج صوتی از محیط است.

۲- آمپلی فایرPAM8403: این آمپلی فایر با توان کم، توانایی تقویت سیگنال‌های صوتی دریافت شده از میکروکنترلر را دارد و به بلندگو اجازه می‌دهد تا صدا را به صورت واضح و با کیفیت پخش کند.

۳- بلندگو: بلندگوی ۸ اهم ۰.۵ وات به عنوان خروجی نهایی دستگاه عمل کرده و صدای ضبط شده را پخش می‌کند.

٤- کنترل ضبط و پخش: با استفاده از دو کلید فشاری، کاربر می‌تواند فرآیند ضبط و پخش را کنترل کند. فشردن کلید ضبط آغاز فرآیند ضبط و فشردن کلید پخش آغاز فرآیند پخش صدا است.

۵- نشانگرهای LED: دو LED برای نمایش وضعیت ضبط (نور قرمز) و پخش (نور آبی) در نظر گرفته شده است که وضعیت فعلی دستگاه را به کاربر نشان می‌دهد.

٦- عملکرد جایگزینی: در صورتی که کاربر مجدداً کلید ضبط را فشار دهد، صدای جدید ضبط شده، جایگزین صدای قبلی می‌شود.

۷- مدت زمان ضبط: دستگاه قادر به ضبط صدا به مدت حداقل ۲۰ ثانیه و حداکثر ۳۰ ثانیه باشد که این قابلیت اجازه می‌دهد کاربر تمام جزئیات مهم را در طول ضبط دریافت کند.

* در انتهای پروژه تهیه گزارش و ارائه آن الزامی است. در هنگام جمع‌آوری نتایج، به‌جز اطلاعات دقیق درباره عملکرد مدار، چالش‌ها و مسائلی که در طی انجام پروژه با آن‌ها مواجه شده‌اید را نیز مستند کنید. این اطلاعات می‌تواند برای تحلیل نهایی و ارائه مفید باشد.

* طراحی مدارهای مورد نیاز و برنامه کاملا به عهده دانشجو است و بنا به صلاحدید مدرس می‌تواند بخشی از نمره آزمایشگاه باشد.

منابع

[1] STMicroelectronics, "STM32H7B0xB Datasheet," Available: https://www.st.com/resource/en/datasheet/stm32h7b0ib.pdf.

[2] IBM, "Microcontroller vs Microprocessor: What's the Difference?", Available: https://www.ibm.com/think/topics/microcontroller-vs-microprocessor.

[3] WatElectronics, "What is ARM Processor - ARM Architecture and Applications," Available: https://www.watelectronics.com/arm-processor-architecture-working/.

[4] Kavir Electronic, "EWB-STM32H7B0 Board," Available: https://kavirelectronic.ir/eshop/my-productions/1201808-ewb-stm32h7b0-board.html.

[5] STMicroelectronics, "Getting Started with STM32 - STM32 Step by Step," Available: https://wiki.st.com/stm32mcu/wiki/STM32StepByStep:Getting_started_with_STM32_:_STM32_step_by_step.

[6] STMicroelectronics, "Getting Started with EXTI in STM32," Available: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_EXTI

[7] STMicroelectronics, "Getting Started with UART in STM32," Available: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_UART.

[8] STMicroelectronics, "Getting Started with DAC in STM32," Available: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_DAC.

[9] STMicroelectronics, "Getting Started with ADC in STM32," Available: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_ADC.

[10] EmbeTronicX, "STM32F407 Timer Tutorial Using STM32CubeIDE," Available: https://embetronicx.com/tutorials/microcontrollers/stm32/stm32f407-timer-tutorial-using-stm32cubeide/.

[11] Embedded There, "STM32 Timer Tutorial Using Interrupt," Available: https://embeddedthere.com/stm32-timer-tutorial-using-interrupt/.

[12] Microcontrollers Lab, "STM32 Nucleo PWM Timers Using STM32CubeIDE," Available: https://microcontrollerslab.com/stm32-nucleo-pwm-timers-stm32cubeide/.

[13] ControllersTech, "Internal RTC in STM32," Available: https://controllerstech.com/internal-rtc-in-stm32/.

[14] SteppeSchool, "STM32 SPI Programming - Polling MPU9250," Available: https://www.steppeschool.com/pages/blog/stm32-spi-programming-polling-mpu9250.

[15] Winbond, "W25Q64JV Datasheet," Available: https://www.alldatasheet.com/datasheet-pdf/pdf/1243794/WINBOND/W25Q64JV.html.

[16] STMicroelectronics, "Getting Started with DMA in STM32," Available: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_DMA.

[17] UPV DISCA, "STM32F439xx User Manual," Available: https://www.disca.upv.es/aperles/arm_cortex_m3/llibre/st/STM32F439xx_User_Manual/index.html.

پیوست۱: لیست قطعات مورد نیاز

* لیست قطعات مورد نیاز برای آزمایش ۱ الی ۱۲

نام قطعه	تعداد
برد برد	یک عدد
برد EWB-STM32H7B0	یک عدد
سیم جامپر	--
صفحه کلید ٤*٤	یک عدد
سون سگمنت	یک عدد
مقاومت ۲۲۰اهم	هفت عدد
پتانسیومتر ۱۰کیلواهم	یک عدد
کابل شارژر اندروید Type-B	یک عدد

* لیست قطعات مورد نیاز برای آزمایش ۱۳- پروژه ۱

نام قطعه	تعداد
برد برد	یک عدد
برد EWB-STM32H7B0	یک عدد
16*2 Character LCD	یک عدد
پتانسیومتر ۱۰کیلواهم	یک عدد

* لیست قطعات مورد نیاز برای آزمایش ١٤- پروژه ۲

نام قطعه	تعداد
برد برد	یک عدد
برد EWB-STM32H7B0	یک عدد
KY-35 module / AUX cable	یک عدد
بلندگوی ۸ اهم ۰.۵وات	یک عدد
PAM8403 module	یک عدد
کلید فشاری	دو عدد