



دانشکده مهندسی برق

گروه مهندسی برق گرایش سیستم‌های دیجیتال

## گزارش کارآموزی

nRF52832 میکروکنترلر

نام و نام خانوادگی دانشجو: علی نقی لو

نام و نام خانوادگی استاد کارآموزی: دکتر حسین حسینی‌نژاد محبتی

محل کارآموزی: زیست پردازش نصیر (Zestel)

نشانی: تهران، سیدخندان، خیابان دبستان، کوچه آگاهی، مرکز رشد واحدهای فناور دانشگاه  
صنعتی خواجه نصیرالدین طوسی

تلفن: ۰۲۱۸۸۴۶۸۷۷۵

تاریخ انجام کارآموزی: تابستان ۱۴۰۳



مُنْزَلٌ



## تشکر و قدردانی

در آغاز این گزارش، مایلم از تمامی افرادی که در طول دوره کارآموزی من در شرکت زیستل همراه و راهنمای من بودند، صمیمانه تشکر و قدردانی نمایم.

ابتدا از دکتر حسینی‌نژاد، استاد کارآموزی و مدیر محترم شرکت زیستل که با فراهم‌کردن این فرصت ارزشمند، زمینه‌ساز یادگیری و تجربه‌های گران‌بهای من شدند، قدردانی می‌کنم.

از دکتر درمانی، استاد گران‌قدر دانشگاهی که بخش زیادی از مهارت‌ها و دانش سخت‌افزار و نرم‌افزار خود را مديون آموزش‌های دقیق و ارزشمند ایشان هستم، بسیار سپاسگزارم.

تشکر ویژه‌ای از دکتر کاتوزیان، مسئول فنی شرکت زیستل و سرپرست کارآموزی بنده، به دلیل راهنمایی‌های فنی و تخصصی ایشان که نقش مهمی در پیشرفت و توسعه توانمندی‌های من داشت، دارم.

از مهندس رضایی و جزایری که همگی از دانشجویان سال بالایی دانشگاه و نیروهای رسمی شرکت زیستل بودند و با صبوری و دانش خود به من و دیگر کارآموزان در کار با میکروکنترلر<sup>۱</sup> nRF52832 و سایر پروژه‌ها کمک‌های فراوانی کردند، نهایت تشکر را دارم.

حضور و حمایت این عزیزان، بی‌تردید یکی از عوامل مهم در موفقیت و بهره‌وری این دوره کارآموزی بود و برای من به عنوان یک تجربه ارزشمند و فراموش‌نشدنی باقی خواهد ماند.



برخی عضوهای شرکت

<sup>۱</sup> Microcontroller



## فهرست مطالب

عنوان	صفحه
فهرست جداول .....	۵
فهرست شکل‌ها .....	۱۰
چکیده.....	۱۰
فصل اول .....	۱۰
معرفی شرکت زیست پردازش نصیر.....	۱۰
۱-۱- مقدمه .....	۱
۲-۱- تاریخچه و تولیدهای زیستل .....	۱
۳-۱- محصولات و خدمات زیستل.....	۱
۴-۱- چگونگی ارتباط مسائل انسانی و مدیریتی زیستل.....	۱
۴-۵- افتخارات زیستل .....	۱
۵-۱- واپايش کیفیت در زیستل.....	۱
۵-۷- چشم انداز زیستل.....	۱
فصل دوم .....	۷
کارهای انجام شده در دوره کارآموزی .....	۷
۱-۲- مقدمه .....	۷
۲-۲- تراشه nRF52832 و شرکت Nordic Semiconductor .....	۷
۳-۲- مشاهده ویدیوهای آموزشی .....	۷
۴-۲- آمادگی اولیه و مرور زبان برنامه نویسی C .....	۱۲
۵-۲- فعالیت‌های عملی .....	۱۲
۶-۲- راهنمایی‌ها .....	۱۴
فصل سوم .....	۱۵

## ۱۵ ..... شرکت Nordic Semiconductor و تراشه nRF52832

۱۵ ..... معرفی شرکت Nordic Semiconductor	-۱-۳
۱۵ ..... تراشه nRF52832	-۲-۳
۱۸ ..... علت نام‌گذاری nRF52832	-۳-۳
۱۸ ..... کاربردها و ویژگی‌ها	-۴-۳
۱۸ ..... بورد محصولات نوار قلب و فشارسنج شرکت	-۵-۳
۱۹ ..... پروگرامر و دیباگر JLINK V9-STM	-۶-۳
۲۰ ..... Segger Embedded Studio	-۷-۳
۲۱ ..... نتیجه‌گیری	-۸-۳

## ۲۲ ..... فصل چهارم

۲۲ ..... راه اندازی RGB با GPIO	
۲۳ ..... کد	-۱-۴
۲۴ ..... توضیحات کد	-۲-۴
۲۵ ..... ماشین حالت	-۳-۴
۲۵ ..... نتیجه‌گیری	-۴-۴

## ۲۷ ..... فصل پنجم

۲۷ ..... راه اندازی RGB با GPIOTE	
۲۷ ..... کد	-۱-۵
۲۸ ..... توضیحات کد	-۲-۵
۲۹ ..... ماشین حالت	-۳-۵
۳۰ ..... نتیجه‌گیری	-۴-۵

## ۳۱ ..... فصل ششم

۳۱ ..... راه اندازی PWM و GPIOTE	
۳۱ ..... کد	-۱-۶

۳۲ .....	۲-۶ - توضیحات کد
۳۳ .....	۳-۶ - نتیجه‌گیری
۳۵.....	فصل هفتم
۳۵.....	ارسال متن به Debug Terminal
۳۵ .....	۱-۷ - کد
۳۵ .....	۲-۷ - توضیحات کد
۳۶ .....	۳-۷ - نتیجه‌گیری
۳۷.....	فصل هشتم
۳۷.....	کار با SSD 1306 مدل OLED
۳۷ .....	۱-۸ - کد
۳۸ .....	۲-۸ - توضیحات کد
۳۹ .....	۳-۸ - نتیجه‌گیری
۴۱.....	فصل نهم
۴۱.....	کار با Timer و پروتکل UART
۴۱ .....	۱-۹ - کد
۴۴ .....	۲-۹ - توضیحات کد
۴۷ .....	۳-۹ - نتیجه‌گیری
۴۹.....	فصل دهم
۴۹.....	پروتکل SPI
۴۹ .....	۱-۱۰ - کد
۵۰ .....	۲-۱۰ - توضیحات کد
۵۲ .....	۳-۱۰ - نتیجه‌گیری

۵۳.....	<b>فصل یازدهم</b>
۵۳.....	<b>کار با SAADC</b>
۵۳.....	۱-۱۱- کد
۵۴ .....	۲-۱۱- توضیحات کد
۵۶ .....	۳-۱۱- نتیجه گیری
۵۷.....	<b>فصل دوازدهم</b>
۵۷.....	<b>مبانی Bluetooth Low Energy</b>
۵۷ .....	۱-۱۲- مقدمه
۵۸ .....	۲-۱۲- نسخه های بلوتوث
۵۹ .....	۳-۱۲- پشته پروتکل BLE
۶۱ .....	Physical Layer -۴-۱۲
۶۲ .....	Link Layer -۵-۱۲
۶۳ .....	Host Controller Interface -۶-۱۲
۶۳ .....	Logical Link Control and Adaptation Protocol -۷-۱۲
۶۴ .....	Security Manager Protocol -۸-۱۲
۶۴ .....	Attribute Protocol -۹-۱۲
۶۴ .....	Generic Attribute Profile -۱۰-۱۲
۶۸ .....	Generic Access Profile -۱۱-۱۲
۷۲ .....	BLE Communication -۱۲-۱۲
۸۱ .....	Soft Device -۱۳-۱۲
۸۲ .....	۱۴-۱۲- شکل کلی کد زدن برای BLE
۸۵.....	<b>فصل سیزدهم</b>
۸۵.....	<b>بلوتوث با کنترل LED و نظارت بر دکمه فشاری</b>
۸۵ .....	۱-۱۳- کد
۹۲ .....	۲-۱۳- توضیحات کد

ک

۹۸ ..... نتیجه‌گیری ۳-۱۳

۱۰۱ ..... نتیجه‌گیری

۱۰۳ ..... مرجع‌ها



## فهرست جداول

عنوان	صفحه
۱ جدول (۱-۳) مقایسه کلی خانواده nRF	۱۷

---



## فهرست شکل‌ها

عنوان	صفحه
شکل (۱-۱) نماد شرکت.....	۱
شکل (۲-۱) الف) تپ اکسی‌متر ب) نوار قلب ج) تب سنج د) فشارسنج دیجیتال.....	۲
شکل (۳-۱) نرم افزار زیستا.....	۳
شکل (۴-۱) محیط دوستانه‌ی شرکت.....	۳
شکل (۵-۱) لوح تقدیر و تندیس رویداد ملی هوش مصنوعی و سلامت.....	۴
شکل (۶-۱) محصول نوآورانه‌ی ثبت نوار قلب .....	۴
شکل (۷-۱) بخش کنترل کیفیت شرکت.....	۵
شکل (۸-۱) کد QR.....	۶
شکل (۱-۲) ویدیوهای ضبط شده .....	۱۴
شکل (۱-۳) نماد شرکت Nordic Semiconductor .....	۱۵
شکل (۲-۳) تراشه nRF52832 .....	۱۵
شکل (۳-۳) وجه بالایی بورد .....	۱۸
شکل (۴-۳) وجه زیرین بورد .....	۱۹
شکل (۵-۳) J-LINK .....	۱۹
شکل (۶-۳) نمودار پایه‌ها .....	۲۰
شکل (۷-۳) نماد SES .....	۲۰
شکل (۱-۴) قرمز، سبز، و آبی .....	۲۴
شکل (۲-۴) نمودار حالت.....	۲۵
شکل (۳-۴) کد QR (۳-۴) .....	۲۶

شکل (۱-۵) قرمز، سبز، و آبی.....	۲۹
شکل (۲-۵) نمودار حالت.....	۲۹
شکل (۳-۵) کد QR.....	۳۰
شکل (۱-۶) روند افزایشی.....	۳۳
شکل (۲-۶) روند کاهشی.....	۳۳
شکل (۳-۶) روند نما.....	۳۳
شکل (۴-۶) کد QR.....	۳۴
شکل (۱-۷) دیباگ ترمینال داخلی SEGGER.....	۳۶
شکل (۲-۷) کد QR.....	۳۶
شکل (۱-۸) نمایش نماد دانشگاه و شرکت روی OLED.....	۳۸
شکل (۲-۸) نرم افزار LCD Vision.....	۳۹
شکل (۳-۸) کد QR.....	۳۹
شکل (۱-۹) دریافت تایم و نمایش در ترمینال.....	۴۵
شکل (۲-۹) ارسال قرمز.....	۴۵
شکل (۳-۹) ارسال سبز.....	۴۶
شکل (۴-۹) ارسال آبی.....	۴۶
شکل (۵-۹) ارسال ساعت.....	۴۶
شکل (۶-۹) Arduino IDE و PuTTY.....	۴۷
شکل (۷-۹) CP2102 مدل USB to TTL رابط.....	۴۷
شکل (۸-۹) کد QR.....	۴۸
شکل (۱۰-۱) خواندن رجیستر.....	۵۱
شکل (۱۰-۲) کد QR.....	۵۲

شکل (۱-۱۱) سیگنال ژنراتور و مشخصات تپ.....	۵۵
شکل (۲-۱۱) مشاهده خروجی .....	۵۵
شکل (۳-۱۱) کد QR .....	۵۶
شکل (۴-۱۲) نماد قدیمی BLE .....	۵۷
شکل (۵-۱۲) فلسفه نام‌گذاری .....	۵۸
شکل (۶-۱۲) سیر تکامل بلوتوث .....	۵۸
شکل (۷-۱۲) سرعت و برد ارتباط بی‌سیم .....	۵۹
شکل (۸-۱۲) پیکربندی بین نسخه‌های بلوتوث و انواع دستگاهها .....	۵۹
شکل (۹-۱۲) پشته پروتکل BLE .....	۶۰
شکل (۱۰-۱۲) کانال‌های فرکانسی BLE .....	۶۱
شکل (۱۱-۱۲) ماشین حالت لایه لینک و انتقال حالت.....	۶۳
شکل (۱۲-۹) سلسله مراتب داده GATT .....	۶۵
شکل (۱۳-۱۲) روش‌های تبادل داده.....	۶۶
شکل (۱۴-۱۲) M-IMU .....	۶۷
شکل (۱۵-۱۲) سلسله مراتب داده GATT برای مثال .....	۶۸
شکل (۱۶-۱۲) پارامترهای Broadcast .....	۷۳
شکل (۱۷-۱۲) مثالی از ارتباط BLE .....	۷۵
شکل (۱۸-۱۲) ساختار بسته BLE .....	۷۷
شکل (۱۹-۱۲) ساختار PDU یک بسته تبلیغاتی .....	۷۸
شکل (۲۰-۱۲) ساختار PDU یک بسته اتصال.....	۷۹
شکل (۲۱-۱۲) گام‌های تنظیم تبلیغات.....	۷۹
شکل (۲۲-۱۲) انواع داده قابل تبلیغ.....	۸۰

۸۱	شکل (۲۱-۱۲) اجزای Soft Device
۸۲	شکل (۲۲-۱۲) نحوه نام‌گذاری Soft Device
۸۳	شکل (۲۳-۱۲) روند کلی کد زدن برای BLE
۹۴	شکل (۱-۱۳) شروع برنامه
۹۴	شکل (۲-۱۳) دستگاه در حالت تبلیغات
۹۴	شکل (۳-۱۳) نحوه اتصال در nRF Connect
۹۵	شکل (۴-۱۳) دستگاه در حالت متصل
۹۵	شکل (۵-۱۳) برنامه‌ی nRF Connect
۹۵	شکل (۵-۱۳) پیام متصل شدن
۹۶	شکل (۶-۱۳) نرم‌افزارهای مختلف NORDIC
۹۷	شکل (۷-۱۳) واپایش LED آبی با کمک نرم‌افزار nRF Connect
۹۷	شکل (۸-۱۳) واپایش در nRF Connect
۹۸	شکل (۹-۱۳) واپایش در nRF Blinky
۹۸	شکل (۱۰-۱۳) پیام‌های Logger

## چکیده

در این گزارش، به بررسی تجربه‌های عملی و تحقیقاتی در طول دوره کارآموزی در شرکت زیست پردازش نصیر پرداخته شده است. تمرکز اصلی بر روی تراشه nRF52832 و کاربردهای آن در پروژه‌های مختلف از جمله راهاندازی LED‌ها با استفاده از پروتکل‌های GPIOTE و PWM، ارسال داده‌ها به ترمینال دیباگ با استفاده از SEGGER Logger داخلی، و پیاده‌سازی ارتباطات بی‌سیم با استفاده از بلوتوث کم‌صرف (BLE) بوده است. همچنین، به تحقیق در مورد معماری نرم‌افزاری و ساخت‌افزاری موردن استفاده در این پروژه‌ها و نحوه به کارگیری آن‌ها در توسعه سیستم‌های اینترنت اشیاء (IoT) پرداخته شده است. نتیجه‌گیری‌های حاصل از این دوره نشان‌دهنده اهمیت تجربه عملی در کنار مطالعات نظری در توسعه توانمندی‌های فنی می‌باشد.

**واژه‌های کلیدی:** کارآموزی، nRF52832، بلوتوث کم‌صرف، پروتکل، SEGGER Embedded Studio



## فصل اول

### معرفی شرکت زیست پردازش نصیر

#### ۱-۱- مقدمه

در دنیای امروز که فناوری به سرعت در حال پیشرفت است، صنعت سلامت نیز از این پیشرفت‌ها بی‌نصیب نمانده است. یکی از شرکت‌های دانش بنیان که در این زمینه فعالیت چشمگیری دارد، شرکت زیست پردازش نصیر (Zistel<sup>۱</sup>) است. زیستل با بهره‌گیری از دانش و فناوری روز دنیا، محصول‌هایی و خدمات‌هایی نوین در حوزه سلامت ارائه می‌دهد و به دنبال بهبود کیفیت زندگی افراد است.

#### ۲-۱- تاریخچه و تولیدهای زیستل

شرکت زیست پردازش نصیر، با نام تجاری زیستل که نماد آن را در شکل (۱-۱) مشاهده می‌کنید، در فروردین سال ۱۴۰۰ با شماره ثبت ۵۷۵۴۰۷ تأسیس شد. این شرکت با هدف ارتقای سطح سلامت جامعه و دسترسی آسان به خدمات سلامت، فعالیت خود را آغاز نمود. اعضای این شرکت متشکل از دانشجوها و عضوهای هیئت علمی دانشگاه صنعتی خواجه نصیرالدین طوسی می‌باشند.

زیستل با تمرکز بر تولید و عرضه تجهیزهای پزشکی هوشمند و قابل حمل، امکان پایش مداوم وضعیت سلامت افراد را فراهم می‌آورد. محصول‌ها زیستل به مردم کمک می‌کنند تا به راحتی و در هر زمان و مکانی، علامت‌های حیاتی خود را اندازه‌گیری کرده و از وضعیت سلامت خود آگاه شوند.



**زیستل**

شکل (۱-۱) نماد شرکت

<sup>۱</sup> Zistel

### ۳-۱- محصولات و خدمات زیستل

تجهیزات پزشکی خانگی: زیستل طیف گسترده‌ای از تجهیزهای پزشکی خانگی مانند فشارسنج‌های دیجیتال، تپ اکسی‌متر<sup>۱</sup>، تب‌سنج و دستگاه‌های نوار قلب<sup>۲</sup> را که در شکل (۲-۱) مشاهده می‌کنید، تولید و عرضه می‌کند. این تجهیزها با دقت بالا و کاربری آسان، امکان خودمراقبتی را برای افراد فراهم می‌آورند.



شکل (۲-۱) (الف) تپ اکسی‌متر (ب) نوار قلب (ج) تب‌سنج (د) فشارسنج دیجیتال

نرم‌افزارهای سلامت: زیستل علاوه بر سخت‌افزارهای پزشکی، نرم‌افزارهای موبایلی و وب اپلیکیشن‌هایی نظیر "زیستا" را نیز ارائه می‌دهد (شکل ۳-۱) که امکان جمع‌آوری، تحلیل و انتقال داده‌های سلامت کاربرها را از طریق تجهیزهای این شرکت فراهم می‌آورد.

<sup>1</sup> Pulse Oximeter

<sup>2</sup> ElectroCardioGram (ECG)



شکل (۳-۱) نرم افزار زیستا

#### ۴-۱- چگونگی ارتباط مسائل انسانی و مدیریتی زیستل

در زیستل، توجه ویژه‌ای به مسئله‌های انسانی و مدیریتی می‌شود. همان‌طور که در شکل (۴-۱) مشخص است، شرکت با ایجاد محیط کاری دوستانه و حمایت از نوآوری و خلاقیت، سعی در افزایش رضایت و بهره‌وری کارکنان دارد. ارتباط‌های مدیریتی در شرکت به صورت باز و شفاف است که این امر باعث تسهیل در جریان اطلاعات و هماهنگی بین تیم‌های مختلف می‌شود.



شکل (۴-۱) محیط دوستانه‌ی شرکت

## ۱-۵- افتخارات زیستل

شرکت زیستل، با تمرکز بر فناوری‌های پیشرفته در حوزه سلامت، موفق به کسب جایگاه برتر به عنوان دستیار هوشمند خودمراقبتی در رویداد ملی هوش مصنوعی و سلامت<sup>۱</sup> شد (شکل ۱-۵). این افتخار به دلیل ارائه محصول نوآورانه دستگاه نوار قلب خانگی که در شکل ۱-۶ قابل مشاهده است و با بهره‌گیری از هوش مصنوعی و قابلیت پایش از راه دور<sup>۲</sup>، به شناسایی و پایش بیماری‌های قلبی از جمله آریتمی قلب<sup>۳</sup> کمک می‌کند، به دست آمد.



شکل (۱-۵) لوح تقدیر و تندیس رویداد ملی هوش مصنوعی و سلامت



شکل (۱-۶) محصول نوآورانه ثبت نوار قلب

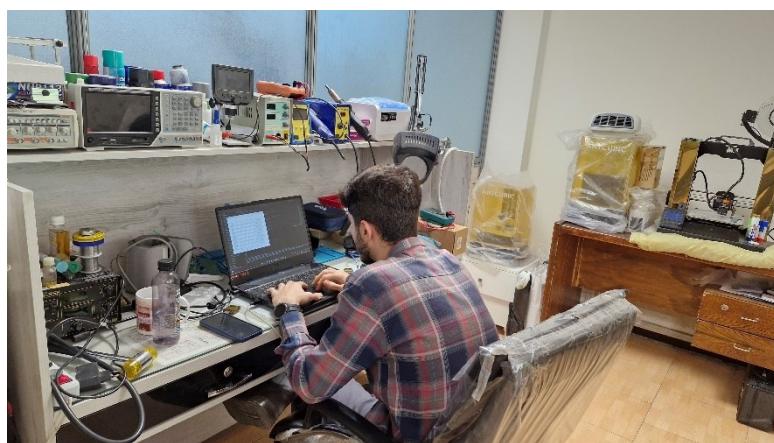
<sup>۱</sup> National event of AI and Health

<sup>۲</sup> Telemedicine

<sup>۳</sup> Cardiac arrhythmia

## ۶-۱- واپایش کیفیت در زیستل

بخش واپایش کیفیت<sup>۱</sup> به عنوان یکی از اجزای کلیدی شرکت، نقشی حیاتی در تصمین کیفیت محصولات و بهبود فرآیندهای تولید دارد. همان‌طور که در شکل ۷-۱ مشخص است، این بخش با نظارت مداوم و دقیق بر تمامی مراحل تولید، از ورود مواد اولیه تا محصول نهایی، اطمینان حاصل می‌کند که استانداردهای کیفی به طور کامل رعایت می‌شوند. عملکرد مؤثر واپایش کیفیت نه تنها باعث افزایش رضایت مشتریان می‌شود، بلکه از طریق کاهش تولید محصولات معیوب، هزینه‌های اضافی را نیز به حداقل می‌رساند.



شکل (۷-۱) بخش کنترل کیفیت شرکت

## ۷-۱- چشم‌انداز زیستل

زیستل با چشم‌اندازی روشن به آینده، به دنبال توسعه محصول‌ها و خدمات‌های خود و گسترش فعالیت‌های خود در حوزه سلامت است. این شرکت با همکاری با متخصص‌ها و پژوهشگران، تلاش می‌کند تا راهکارهای نوینی برای بهبود سلامت افراد ارائه دهد و به سهم خود به ارتقای سطح سلامت جامعه کمک کند.

در کل، زیست پردازش نصیر (زیستل) با ارائه محصولات و خدمات باکیفیت و نوآورانه، گامی مهم در جهت دیجیتالی شدن صنعت سلامت برداشته است. برای اطلاعات بیشتر، کد QR موجود در شکل ۸-۱ را اسکن کنید.

<sup>۱</sup> Quality Control (QC)



شکل (۸-۱) کد QR

## فصل دوم

### کارهای انجام شده در دوره کارآموزی

#### ۱-۲ - مقدمه

در دوره کارآموزی در شرکت زیستل، کارهای مختلفی در زمینه‌های مختلف انجام شد که به تفصیل در ادامه شرح داده می‌شود. این فعالیت‌ها شامل مشاهده ویدیوهای آموزشی، انجام پروژه‌های عملی و آشنایی با ابزارها و تجهیزات مرتبط با میکروکنترلر nRF52832 بود.

#### ۲-۲ - تراشه nRF52832 و شرکت Nordic Semiconductor

یکی از پرطرفدارترین تراشه‌های تولید شده توسط شرکت nRF52832، Nordic Semiconductor است. بررسی جزئیات کامل مربوط به این تراشه، از جمله ویژگی‌ها، قابلیت‌ها، پروگرمر<sup>۱</sup>، قیمت و همچنین اطلاعات جامع درباره این شرکت سازنده نروژی، در حوصله‌ی این فصل نیست و به طور مفصل در فصل بعدی یعنی فصل سوم ارائه خواهد شد.

#### ۳-۲ - مشاهده ویدیوهای آموزشی

در همان ابتدای کارآموزی، به مدت تقریبی ۱ الی ۲ هفته به مشاهده ویدیوهای آموزشی از کanal یوتیوبی<sup>۲</sup> Sumair's Embedded Engineering معرفی کیت<sup>۳</sup>‌های توسعه برای بلوتوث<sup>۴</sup>، نصب و فعال‌سازی Segger Embedded Studio<sup>۵</sup>، ساختار

<sup>1</sup> Programmer

<sup>2</sup> YouTube

<sup>3</sup> Kit

<sup>4</sup> Bluetooth

<sup>5</sup> Segger Embedded Studio (SES)

پوشه‌های SDK<sup>۱</sup>, پیکربندی<sup>۲</sup> پرونجا<sup>۳</sup> sdk\_config.h, اشکال‌زدایی<sup>۴</sup> از طریق j-link و ارتباط زنجیره<sup>۵</sup> UART<sup>۶</sup> بود. این دوره آموزشی مقدماتی برای آشنایی با محیط توسعه و سخت‌افزار nRF52832 طراحی شده بود.

فهرستی کامل و جزیی از ویدیوهای و قسمت‌های مختلف SDK در nRF52832 روى SEGGER برای دوره کارآموزی به شرح زیر است:

### بخش اول ویدیوها: آشنایی با محیط توسعه و سخت‌افزار

- معرفی کیت‌های توسعه برای بلوتوث Segger Embedded Studio
- نصب و فعال‌سازی Segger Embedded Studio
- ساختار پوشه‌های SDK 16.0
- توضیح Segger Embedded Studio به زبان ساده
- خروجی پایه (چشمک‌زدن<sup>۷</sup> LED<sup>۸</sup>)
- خواندن ورودی دیجیتال از دکمه‌ها
- پیکربندی<sup>۹</sup> sdk\_config.h با استفاده از ویزارد<sup>۱۰</sup> CMSIS
- اشکال‌زدایی از طریق j-link با استفاده از کتابخانه Segger RTT<sup>۱۱</sup>
- استفاده از پرونجا Boards.h برای عملیات ورودی/خروجی ساده
- ارتباط زنجیره UART
- ماژول nRF LOG<sup>۱۲</sup>

<sup>1</sup> Software Development Kit (SDK)

<sup>2</sup> Configuration

<sup>3</sup> File

<sup>4</sup> Debug

<sup>5</sup> Serial

<sup>6</sup> Universal Asynchronous Receiver-Transmitter (UART)

<sup>7</sup> Blink

<sup>8</sup> Light-Emitting Diode (LED)

<sup>9</sup> Wizard

<sup>10</sup> Cortex Microcontroller Software Interface Standard (CMSIS)

<sup>11</sup> Real-Time Transfer (RTT)

<sup>12</sup> Logger (LOG)

- وقفه‌های <sup>۱</sup>GPIOTE
- معرفی تایمرهای سخت‌افزاری <sup>۲</sup>
- معرفی <sup>۳</sup>PPI
- تایمرها بخش دوم: تایمرها با PPI و GPIOTE
- تایمرها بخش سوم: تایمرها به عنوان شمارنده <sup>۴</sup>
- تایمرهای کاربردی <sup>۵</sup>
- بازبینی <sup>۶</sup>BSP (بسته پشتیبانی برد)
- معرفی <sup>۷</sup>SAADC (مبدل آنالوگ به دیجیتال) - تک‌پایانه و تفاضلی <sup>۸</sup>
- حالت تک شات <sup>۹</sup> - SAADC - وقفه محور <sup>۱۰</sup>
- نمونه‌برداری پیوسته <sup>۱۱</sup> SAADC - رویدادمحور <sup>۱۲</sup>
- RTC - شمارنده زمان واقعی - تایمر کم‌صرف
- تایmer نگهبان WDT<sup>۱۳</sup>
- NVMC<sup>۱۴</sup> و حافظه فلاش داخلی
- UARTE<sup>۱۵</sup> غیر مسدود‌کننده رویدادمحور
- کتابخانه زنجیره UART - ارتباط USART
- مقدمه‌ای بر مدولاسیون پهنه‌ای تپ (PWM) و معماری آن

<sup>1</sup> General Purpose Input/Output Task

<sup>2</sup> Harware Timer

<sup>3</sup> Programmable Peripheral Interconnect (**PPI**)

<sup>4</sup> Counter

<sup>5</sup> Application Timer

<sup>6</sup> Board Support Package (**BSP**)

<sup>7</sup> Successive Approximation Analog-to-Digital Converter (**SAADC**)

<sup>8</sup> Single Ended & Differential

<sup>9</sup> Single Shot Mode

<sup>10</sup> Interrupt Driven

<sup>11</sup> Continuous Sampling

<sup>12</sup> Event Driven

<sup>13</sup> Watchdog Timer (WDT)

<sup>14</sup> Non-Volatile Memory Controller (NVMC)

<sup>15</sup> Universal Asynchronous Receiver-Transmitter with EasyDMA (**UARTE**)

- برنامه‌نویسی حالت مشترک PWM<sup>۱</sup>
- برنامه‌نویسی حالت گروهی PWM
- برنامه‌نویسی حالت انفرادی PWM
- برنامه‌نویسی حالت شکل موج PWM
- کتابخانه PWM (تولید PWM با استفاده از تایمراهی سختافزاری)
- کم‌صرف با استفاده از ساعت بسامد پایین<sup>۲</sup> PWM
- واپايش موتور DC
- واپايش موتور DC با استفاده از کتابخانه PWM و L298N
- واپايش موتور پلهای<sup>۳</sup>
- A4988 معرفی و سیم‌کشی
- A4988 واپايش موتور پلهای با PWM - بخش دوم - درایور
- مقدمه و مبانی TWI<sup>۴</sup> - I2C<sup>۵</sup>
- MPU6050 با حسگر شتاب‌سنج و ژیروسکوپ<sup>۶</sup> TWI - I2C
- مقدمه‌ای بر ارتباط SPI<sup>۷</sup>
- ارتباط SPI با حسگر LIS3DSH

#### بخش دوم ویدیوها: بلوتوث کم‌صرف<sup>۸</sup>

- مقدمه‌ای بر بلوتوث کم‌صرف

<sup>۱</sup> Pulse Width Modulation (PWM)

<sup>۲</sup> Low Frequency Clock (LFCLK)

<sup>۳</sup> Stepper

<sup>۴</sup> Two-Wire Interface (TWI)

<sup>۵</sup> Inter-Integrated Circuit (I2C)

<sup>۶</sup> Gyroscope

<sup>۷</sup> Serial Peripheral Interface (SPI)

<sup>۸</sup> Bluetooth Low Energy (BLE)

- مبانی نرمافزارهای Nordic
- نصب و آزمایش nRF Connect
- ایجاد یک پروژه پایه
- تنظیم BLE برای nRF Logger
- برنامهنویسی تایمرهای کاربردی
- برنامهنویسی BSP
- مبانی برنامهنویسی مدیریت توان
- فعالسازی BLE برای Softdevice
- مبانی برنامهنویسی<sup>۱</sup> GAP
- مبانی برنامهنویسی<sup>۲</sup> GATT
- مبانی برنامهنویسی<sup>۳</sup> Advertisement
- آغازینه‌سازی<sup>۴</sup> سرویس‌ها
- آغازینه‌سازی عامل<sup>۵</sup>‌های اتصال
- اپلیکیشن پایه BLE نهایی
- نصب Sniffer و WireShark بسته
- مبانی تبلیغات
- انواع نشانی<sup>۶</sup> دستگاه BLE
- نشانی ثابت تصادفی
- نشانی خصوصی غیرقابل حل

<sup>۱</sup> Generic Access Profile (GAP)<sup>۲</sup> Generic Attribute Profile (GATT)<sup>۳</sup> Initialization<sup>۴</sup> Parameter<sup>۵</sup> Address

- نشانی خصوصی قابل حل تصادفی
- تنظیم نام دستگاه
- تنظیم ظاهر دستگاه BLE
- تنظیم پرچم<sup>۱</sup>های دستگاه دربسته Advertisement
- سطح توان انتقال دربسته Advertisement
- درج UUID<sup>۲</sup> سرویس‌ها دربسته BLE Advertisement
- درج داده‌های سرویس دربسته Advertisement
- افزودن داده‌های فاصله اتصال در BLE Advertisement
- داده‌های خاص فروشنده در بسته‌های Advertisement
- معرفی پروفایل‌ها در BLE
- معرفی سرویس‌ها در BLE

#### ۴-۴- آمادگی اولیه و مرور زبان برنامه‌نویسی C

پس از دریافت وظایف اولیه و آگاهی از اینکه بخش عمده‌ای از کار بر روی میکروکنترلر nRF52832 شامل برنامه‌نویسی به زبان C است، یک روز کامل در خانه به مرور جزوات مبانی برنامه‌نویسی و استفاده از وبگاه‌های آموزشی نظریer GeeksforGeeks پرداخته شد تا آمادگی بیشتری برای انجام پروژه‌های عملی ایجاد شود.

#### ۵-۵- فعالیت‌های عملی

پس از مشاهده ویدیوهای آموزشی و مرور مبانی برنامه‌نویسی زبان C، تحقیقاتی جامع پیرامون مشخصات فنی این میکروکنترلر، رابط اتصال آن به رایانه کیفی<sup>۳</sup>، بررسی تجاری، و قیمت‌های مرتبط

<sup>1</sup> Flag

<sup>2</sup> Universally Unique Identifier (UUID)

<sup>3</sup> Laptop

انجام شد که همان طور که قبلا هم اشاره شده بود، همگی به طور مفصل در فصل ۳ مورد بحث قرار گرفته است. در نهایت، فعالیتهای عملی زیر صورت گرفت:

- ✓ راهاندازی RGB<sup>۱</sup> با GPIO<sup>۲</sup>: برنامه‌ریزی و راهاندازی RGB به‌گونه‌ای که با فشردن دکمه، رنگ<sup>۳</sup> تغییر کند. جزئیات بیشتر در فصل ۴ تشریح خواهد شد.
- ✓ راهاندازی RGB با GPIOTE: استفاده از وقفه برای تغییر رنگ LED با فشردن دکمه. توضیحات کامل در فصل ۵ آمده است.
- ✓ راهاندازی RGB با PWM و GPIOTE: تنظیم شدت نور LED قرمز با استفاده از وقفه و کاهش یا افزایش تدریجی آن. این موضوع به طور جامع در فصل ۶ مورد بحث قرار خواهد گرفت.
- ✓ ارسال متن به Debug Terminal: بهره‌گیری از SEGGER Logger داخلی برای ارسال متن به پایانه<sup>۴</sup> دیباگ. توضیحات کامل‌تر در فصل ۷ ارائه خواهد شد.
- ✓ کار با OLED: نمایش اطلاعات بر روی نمایشگر<sup>۵</sup> OLED با استفاده از پروتکل‌های SPI و I2C. جزئیات بیشتر در فصل ۸ توضیح داده خواهد شد.
- ✓ نمایش عدد تایмер با استفاده از UART و واپاپیش LEDها: کار با تایمر و ارسال مقادیر آن از طریق UART. این مطلب در فصل ۹ به صورت مفصل بررسی خواهد شد.
- ✓ کار با مژول SPI: برقراری ارتباط با مژول SPI و تبادل داده‌ها. در فصل ۱۰ به تفصیل به این موضوع پرداخته خواهد شد.
- ✓ خواندن سطح ولتاژ باتری با ADC: بهره‌گیری از مبدل آنالوگ به دیجیتال برای اندازه‌گیری سطح ولتاژ باتری. این موضوع در فصل ۱۱ به طور کامل بررسی خواهد شد.
- ✓ بلوتوث با کنترل LED و نظارت بر دکمه فشاری: بهره‌گیری از پروتکل بلوتوث کم مصرف برای ارسال یا دریافت واپاپیش قسمت‌هایی از مدار، که پیش نیاز‌های این موضوع در فصل ۱۲ و خود پروژه عملی در فصل ۱۳ به صورت مفصل بررسی خواهند شد.

همگی این فعالیتها با استفاده از برد داخلی محصول آینده‌ی EBM1 یعنی نوار قلب و فشارسنج

<sup>1</sup> Red Green Blue (RGB)

<sup>2</sup> General Purpose Input/Output

<sup>3</sup> Light Emitting Diode (LED)

<sup>44</sup> Terminal

<sup>5</sup> Organic Light Emitting Diode (OLED)

شرکت زیستل انجام شد که شامل دکمه‌فشاری، الای‌دی RGB، نمایشگر OLED و بسیاری از تراشه‌ها و امکانات پیشرفته بود و در انتهای هر فعالیت یک ویدیو مفصل (شکل ۱-۲) با بت توضیحات تکمیلی گرفته شد و به صورت کد QR در انتهای هر فصل از گزارش قرار داده شد که عنوان آن‌ها در زیر آورده شده است:

"RGB via GPIO" ➤ جلسه اول در ۱۹ دقیقه ویدیو

"RGB(int)" ➤ جلسه دوم در ۱۵ دقیقه ویدیو

"PWM and GPIOTE" ➤ جلسه سوم در ۱۹ دقیقه ویدیو

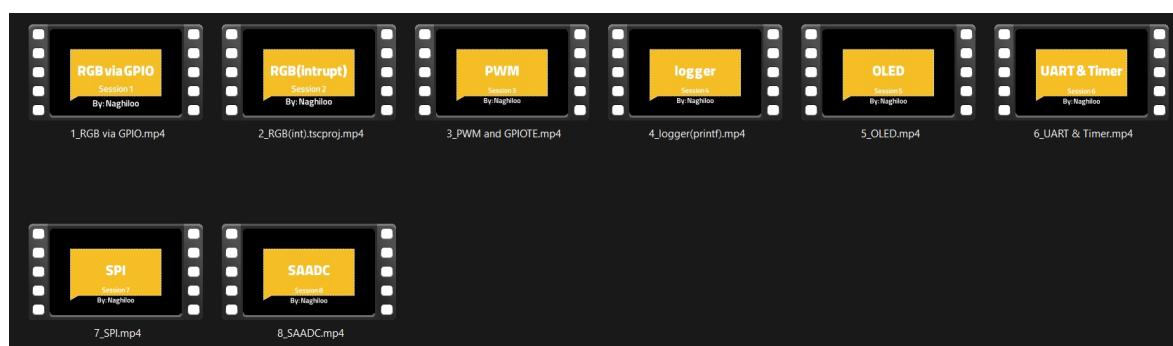
"logger.printf()" ➤ جلسه چهارم در ۱۲ دقیقه ویدیو

"OLED" ➤ جلسه پنجم در ۱۲ دقیقه ویدیو

"UART & Timer" ➤ جلسه ششم در ۳۵ دقیقه ویدیو

"SPI" ➤ جلسه هفتم در ۸ دقیقه ویدیو

"SAADC" ➤ جلسه هشتم در ۹ دقیقه ویدیو ➤



شکل (۱-۲) ویدیوهای ضبط شده

در طول این دوره، دانش و مهارت‌های عملی در زمینه برنامه‌نویسی و کار با میکروکنترلر nRF52832 بهبود یافت و تجربیات ارزشمندی کسب شد.

## ۶-۲- راهنمایی‌ها

در این دوره، از راهنمایی‌های ارزشمند آقایان مهندس سروش رضایی و یوسف جزایری بهره‌مند شدیم. آقایان سروش رضایی و یوسف جزایری در آشنایی اولیه با میکروکنترلر nRF52832 و انجام پروژه‌های عملی، کمک‌های شایانی ارائه دادند. همچنین از دکتر دانیال کاتوزیان که همواره ما کارآموزان را از نکات فنی، تجربی و... بهره‌مند ساختند، صمیمانه تشکر می‌کنیم.

## فصل سوم

### nRF52832 و تراشه Nordic Semiconductor

#### ۱-۳- معرفی شرکت Nordic Semiconductor

شرکت Nordic Semiconductor، تأسیس شده در سال ۱۹۸۳ در نروژ، یکی از پیشروان صنعت نیمه‌رساناهای است. این شرکت به دلیل تولید تراشه‌های بی‌سیم با مصرف انرژی پایین شهرت دارد (شکل ۱-۳). محصولات این شرکت در بسیاری از صنایع از جمله سلامت، خانه‌های هوشمند، پوشیدنی‌ها و اینترنت اشیاء (IoT) مورد استفاده قرار می‌گیرند. تمرکز اصلی Nordic بر تحقیق و توسعه مداوم، ارائه محصولاتی با کیفیت بالا و عملکرد برتر است.



شکل (۱-۳) نماد شرکت Nordic Semiconductor

#### ۲-۳- تراشه nRF52832

تراشه nRF52832 (شکل ۲-۳) یکی از محصولات برجسته Nordic Semiconductor است که به دلیل ویژگی‌ها و عملکرد برتر خود مورد توجه بسیاری از توسعه‌دهندگان قرار گرفته است.



شکل (۲-۳) تراشه nRF52832

این تراشه بر اساس معماری ARM Cortex-M4 طراحی شده، قیمت آن بین ۱۰۰,۰۰۰ تا ۳۰۰,۰۰۰ تومن متغیر است که داخل ایران موجود نیست و باید از چین وارد شود و دارای ویژگی‌های زیر است:

#### پردازنده مرکزی (CPU):

پردازنده M4 ARM با بسامد ۶۴ مگاهرتز و پشتیبانی از FPU (واحد ممیز شناور).

#### حافظه:

۱۲ کیلوبایت حافظه فلش. ۳۲/۶۴ کیلوبایت RAM

#### رادیو بی‌سیم:

پشتیبانی از Bluetooth Low Energy (BLE)، Bluetooth 4/5.0/5.4 و ANT+.

پشتیبانی از ۲.۴ گیگاهرتز پروتکل‌های بی‌سیم اختصاصی.

#### صرف انرژی کم:

دارای حالت‌های مختلف صرف انرژی کم برای کاربردهای باتری محور.

#### واحدهای محیطی (Peripheral Units):

ADC (مبدل آنالوگ به دیجیتال) بادقت ۱۲ بیتی.

PDM، TWI، I2S، PWM، I2C، SPI، UART

تایمرباکس و کانترهای متعدد (۵ تایمرباکس ۳۲ بیتی هر کدام دارای ۴ کانال).

RTC (Real-Time Clock) با کریستال خارجی ۳۲.۷۶۸ کیلوهرتزی و صرف انرژی کم و یک کانتر ۲۴ بیتی با قابلیت شمارش تا ۱۶,۷۷۷,۲۱۵ تیک.

WDT (Watchdog Timer)

PPI (Programmable Peripheral Interconnect) برای ارتباط بین واحدهای محیطی بدون نیاز به CPU.

امنیت:

پشتیبانی از رمزگاری AES (Advanced Encryption Standard)

دسترسی امن به حافظه و پشتیبانی از Secure Boot

پشتیبانی نرم افزاری:

نرم افزار SDK جامع Nordic برای توسعه و برنامه نویسی.

کتابخانه ها و نمونه کدهای مختلف برای توسعه سریع تر.

ویژگی های اضافی:

پشتیبانی از DFU (Device Firmware Update) برای به روز رسانی نرم افزار از راه دور.

یک مقایسه کلی بین هم خانواده های این تراشه در جدول ۱-۳ دیده می شود:

**جدول (۱-۳) مقایسه کلی خانواده nRF**

SoC	nRF5340	nRF52840	nRF52833	nRF52832	nRF52820	nRF52811	nRF52810	nRF52805
Bluetooth	5.4	5.4	5.4	5.4	5.4	5.4	5.4	5.4
Thread	Yes	Yes	Yes		Yes	Yes		
Matter	Yes	Yes						
Zigbee	Yes	Yes	Yes		Yes			
Bluetooth Mesh	Yes	Yes	Yes	Yes	Yes			
Flash	1 MB + 256 KB	1 MB	512 KB	512/256 KB	256 KB	192 KB	192 KB	192 KB
RAM	512 KB + 64 KB	256 KB	128 KB	64/32 KB	32 KB	24 KB	24 KB	24 KB
CPU	128 MHz Arm Cortex-M33 + 64 MHz Arm Cortex-M33	64 MHz Arm Cortex-M4 with FPU	64 MHz Arm Cortex-M4 with FPU	64 MHz Arm Cortex-M4 with FPU	64 MHz Arm Cortex-M4			
High-Speed SPI	Yes	Yes	Yes					
QSPI	Yes	Yes						
USB	Yes	Yes	Yes		Yes			
PWM, PDM, I2S	Yes	Yes	Yes	Yes		PWM, PDM	PWM, PDM	
ADC, Comp	Yes	Yes	Yes	Yes	COMP	ADC, COMP	ADC, COMP	ADC
Temp Range	-40 to 105 °C	-40 to 85 °C	-40 to 105 °C	-40 to 85 °C	-40 to 105 °C	-40 to 85 °C	-40 to 85 °C	-40 to 85 °C
Supply voltage range	1.7 to 5.5 V	1.7 to 5.5 V	1.7 to 5.5 V	1.7 to 3.6 V	1.7 to 5.5 V	1.7 to 3.6 V	1.7 to 3.6 V	1.7 to 3.6 V
Packages	QFN94	QFN73	QFN73	QFN48	QFN40	QFN48	QFN48	WLCSP28
	WLCSP95	WLCSP94	QFN40	WLCSP50		QFN32	QFN32	
				WLCSP76		WLCSP33	WLCSP33	
GPIOs	48	48	18-42	32	18	15-32	15-32	10

### ۳-۳- علت نام‌گذاری nRF52832

نام‌گذاری nRF52832 دارای معانی خاصی است که به ویژگی‌ها و مشخصات این تراشه اشاره دارد:

n: نشان‌دهنده "Nordic" به عنوان شرکت سازنده تراشه است.

RF: مخفف "Radio Frequency" به معنای بس‌امد رادیویی است که به توانایی تراشه در ارتباطات بی‌سیم اشاره دارد.

۵۲: سری ۵۲ تراشه‌های Nordic را مشخص می‌کند که شامل چندین مدل با ویژگی‌های مختلف است.

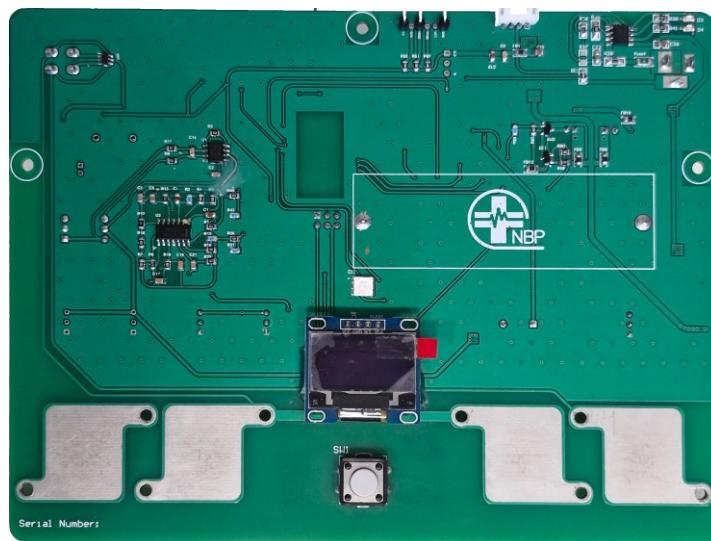
۸۳۲: بخش عددی که نشان‌دهنده مدل خاص این تراشه در سری ۵۲ است.

### ۴-۴- کاربردها و ویژگی‌ها

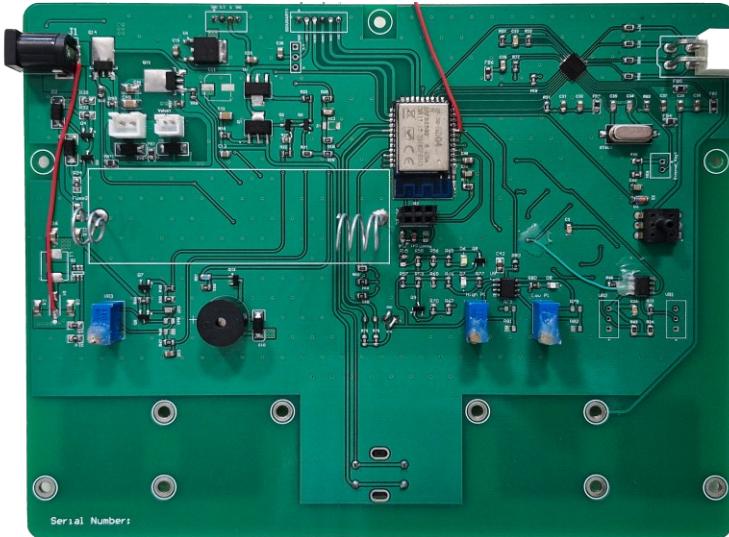
تراشه nRF52832 به دلیل قابلیت‌های متعدد و مصرف انرژی پایین، در کاربردهای گسترده‌ای از جمله تجهیزات پزشکی، پوشیدنی‌های هوشمند، دستگاه‌های هوشمند خانگی و دستگاه‌های اینترنت اشیا مورد استفاده قرار می‌گیرد. این تراشه با ارائه امکانات پیشرفته و کاربری آسان، توسعه‌دهنده‌گان را قادر می‌سازد تا محصولاتی با عملکرد بالا و مصرف انرژی بهینه تولید کنند.

### ۵-۵- بورد محصولات نوار قلب و فشارسنج شرکت

بورد آموزشی‌ای که برای دوره کارآموزی در نظر گرفته شد، بورد داخلی محصولات نوار قلب و فشارسنج بود که در شکل ۳-۳ و ۴-۳ قابل مشاهده است.



شکل (۳-۳) وجه بالایی بورد



شکل (۴-۳) وجه زیرین بورد

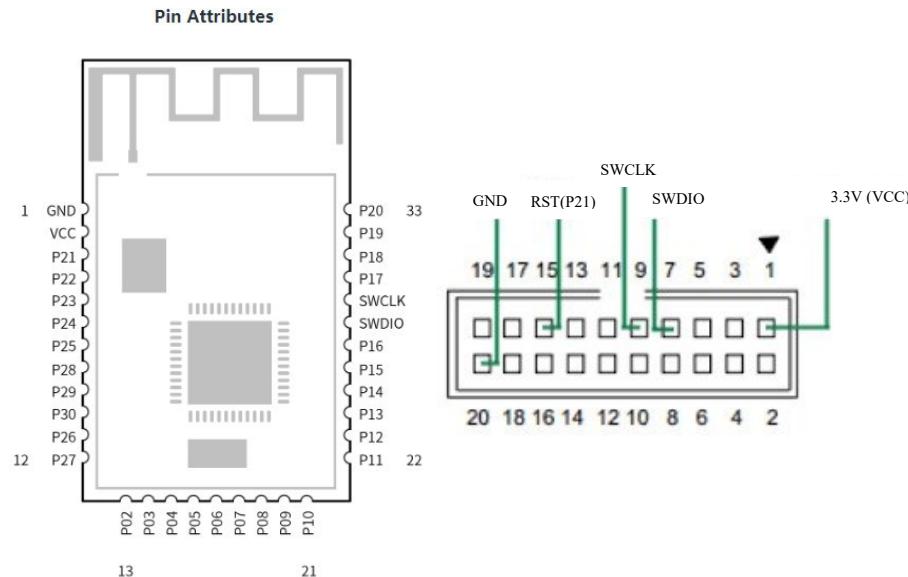
### ۶-۳- پروگرم و دیباگر JLINK V9-STM

(شکل ۳-۵)، ابزاری حرفه‌ای برای دیباگ و برنامه‌ریزی میکروکنترلرهای nRF52832 است که توسط شرکت SEGGER تولید می‌شود. این دستگاه با پشتیبانی از پروتکل‌های SWV، SWD، JTAG و SWI و سرعت بالای انتقال داده، ابزاری مناسب برای توسعه‌دهندگان دستگاه‌های نهفته است. قیمت JLINK V9-STM بسته به منطقه جغرافیایی و عرضه‌کننده متغیر است، اما به‌طور کلی در بازار ایران بین ۱,۰۰۰,۰۰۰ تا ۱,۳۰۰,۰۰۰ تومان متغیر است.



شکل (۵-۳)

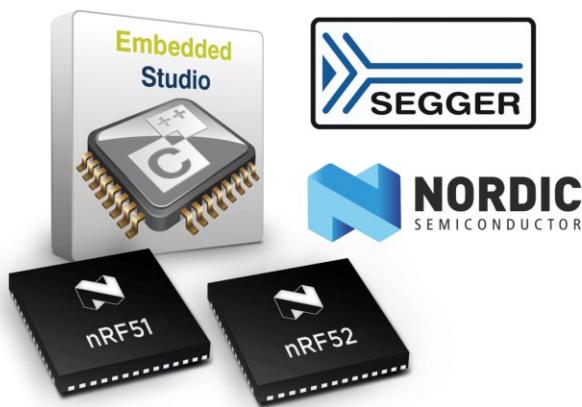
برای اتصال تراشه به J-LINK با حداقل تعداد پین، به صورتی که در شکل (۶-۳) آمده است، عمل شده است:



شکل (۶-۳) نمودار پایه‌ها

### Segger Embedded Studio - ۷-۳

Segger Embedded Studio یک محیط توسعه یکپارچه (IDE) قدرتمند و همه‌جانبه برای برنامه‌نویسی و توسعه نرم‌افزارهای تعبیه شده (Embedded Software) است. این IDE که توسط شرکت Segger طراحی و توسعه یافته است، به ویژه برای کار با میکروکنترلرها (MCUs) و ریزپردازها (MPUs) بر اساس معماری‌های مختلف از جمله ARM Cortex-M بهینه سازی شده است.



شکل (۷-۳) نماد SES

### ۸-۳- نتیجه‌گیری

تراشه nRF52832 به عنوان یکی از محبوب‌ترین تراشه‌های تولید شده توسط شرکت Nordic Semiconductor، با ویژگی‌های برتر و قابلیت‌های متعدد خود، به طور گسترده در پروژه‌های مختلف استفاده می‌شود. در این فصل، به بررسی جامع و تفصیلی تراشه nRF52832، دیباگر و پروگرم J-LINK و همچنین معرفی و بررسی شرکت سازنده آن پرداخته شد. در فصل‌های بعدی، به جزئیات بیشتری از عملکرد و کاربردهای این تراشه خواهیم پرداخت.



## فصل چهارم

### راهاندازی RGB با GPIO

۱-۴ کد

در این فصل، به پیاده‌سازی یک ماشین حالت ساده برای تغییر رنگ LED با استفاده از میکروکنترلر nRF52832 پرداخته می‌شود. در این پروژه، با هر بار فشردن دکمه و فقط با کمک GPIO (و نه inturpt) رنگ LED تغییر می‌کند. کد زیر این پیاده‌سازی را نشان می‌دهد:

```

1. #include <stdbool.h>
2. #include <stdint.h>
3. #include "nrf_gpio.h"
4.
5. #define redled 22
6. #define greenled 17 // blue and green are misplaced.
7. #define blueled 23
8. #define button 25
9. #define micro_power_ctrl 18
10.
11. int btnPressed = 0;
12. bool flag = true;
13. enum color_t {R ,G ,B} color = R;
14.
15. int main(void)
16. {
17.     nrf_gpio_cfg_output(redled);
18.     nrf_gpio_cfg_output(greenled);
19.     nrf_gpio_cfg_output(blueled);
20.     nrf_gpio_cfg_input(button, NRF_GPIO_PIN_PULLDOWN);
21.
22.     nrf_gpio_cfg_output(micro_power_ctrl);
23.     nrf_gpio_pin_set(micro_power_ctrl);
24.
25.     while(true)
26.     {
27.         btnPressed = nrf_gpio_pin_read(button);
28.         if(nrf_gpio_pin_read(button) == 1)
29.         {
30.             if(flag)
31.             {
32.                 switch (color)
33.                 {
34.                     case R:
35.                         nrf_gpio_pin_set(redled);
36.                         nrf_gpio_pin_clear(greenled);
37.                         nrf_gpio_pin_clear(blueled);
38.                         color = G;
39.                         break;

```

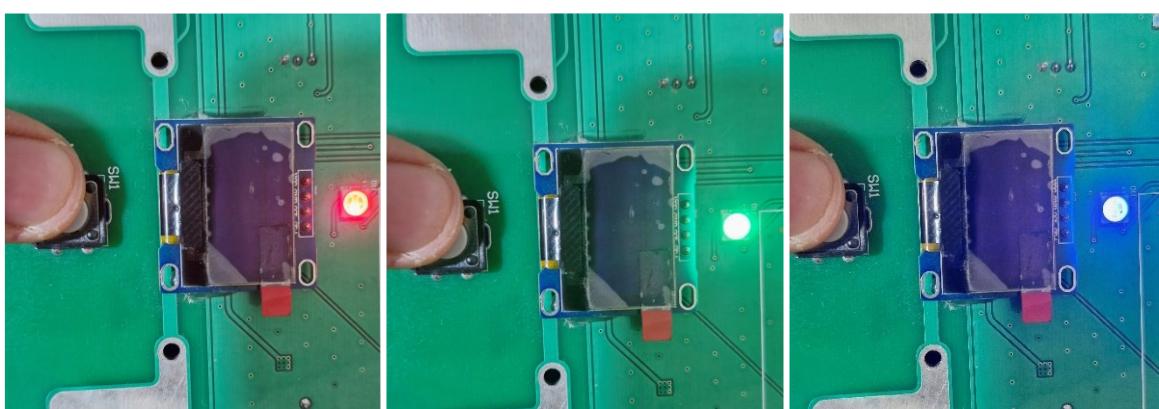
```

40.         case G:
41.             nrf_gpio_pin_clear(redled);
42.             nrf_gpio_pin_set(greenled);
43.             nrf_gpio_pin_clear(blueled);
44.             color = B;
45.             break;
46.         default:
47.             nrf_gpio_pin_clear(redled);
48.             nrf_gpio_pin_clear(greenled);
49.             nrf_gpio_pin_set(blueled);
50.             color = R;
51.             break;
52.         }
53.     flag = false;
54. }
55. else
56. {
57.     flag = true;
58. }
59. }
60. }
61. }
```

## ۲-۴- توضیحات کد

ابتدا، پین‌های مربوط به LED‌ها و دکمه به صورت ثابت تعریف می‌شوند. سپس، پین‌های خروجی برای LED‌ها و پین ورودی برای دکمه پیکربندی می‌شوند. پین واپايش برق میکروکنترلر نیز به عنوان خروجی پیکربندی شده و فعال می‌شود.

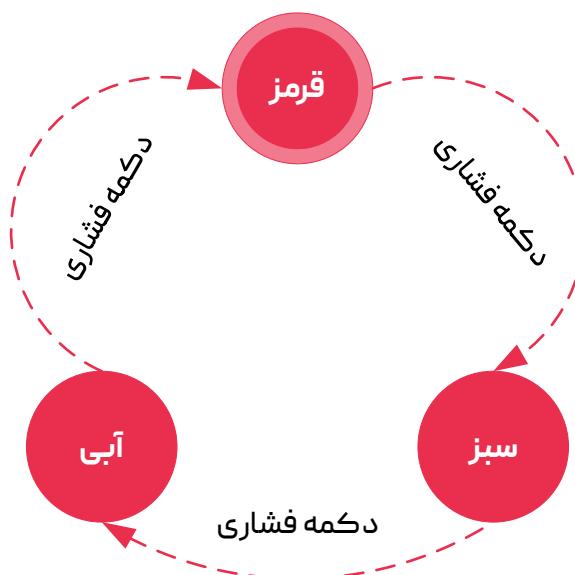
سپس، وضعیت دکمه به طور مداوم خوانده می‌شود. اگر دکمه فشرده شده باشد و flag برابر با true باشد، رنگ LED تغییر می‌کند (شکل ۱-۴). این تغییر رنگ با استفاده از یک ساختار switch انجام می‌شود که براساس رنگ فعلی، LED مناسب روشن و سایر LED‌ها خاموش می‌شوند. در پایان، flag به false تغییر داده می‌شود تا از تغییر مکرر رنگ در هنگام نگه داشتن دکمه جلوگیری شود. زمانی که دکمه رها می‌شود، flag به true تغییر داده می‌شود تا تغییر رنگ در فشردن بعدی دکمه ممکن شود.



شکل (۱-۴) قرمز، سبز، و آبی

### ۳-۴-۱- ماشین حالت<sup>۱</sup>

این کد از یک ماشین حالت ساده که نمودار حالت<sup>۲</sup> آن در شکل ۴-۴ مشخص هست، برای تغییر رنگ LED استفاده می‌کند. ماشین حالت یک تکنیک رایج در برنامه‌نویسی است که در آن دستگاه بین حالت‌های مختلف حرکت می‌کند. در اینجا، سه حالت مختلف برای رنگ‌های قرمز، سبز و آبی تعریف شده است. هر بار که دکمه فشرده می‌شود، ماشین حالت به حالت بعدی حرکت می‌کند و LED مربوط به آن رنگ روشن می‌شود.



شکل (۴-۴) نمودار حالت

### ۴-۴-۲- نتیجه‌گیری

این پروژه نمونه‌ای ساده از نحوه کار با GPIO و واپايش LEDها با استفاده از میکروکنترلر nRF52832 است. توضیحات کامل‌تر و نحوه اجرای این کد و نتایج آن در این فصل به تفصیل بررسی شده‌اند. این پیاده‌سازی، پایه‌ای برای درک بهتر کار با میکروکنترلرها و کاربردهای آن‌ها در پروژه‌های اینترنت اشیا<sup>۳</sup> فراهم می‌کند. برای توضیحات و اطلاعات بیشتر، می‌توانید به فیلم ضبط شده این فصل مراجعه کنید که کد QR آن در شکل ۴-۳ ارائه شده است.

<sup>1</sup> State Machine (SM)

<sup>2</sup> State Diagram

<sup>3</sup> Internet of Things (IoT)



شکل (۳-۴) کد QR

## فصل پنجم

### راه اندازی RGB با GPIO

#### ۱-۵ کد

در این قسمت، به پیاده‌سازی تغییر رنگ LED با استفاده از GPIO با استفاده از nRF52832 پرداخته شده است. در این پروژه، با فشردن دکمه، وقفه ایجاد شده و رنگ LED تغییر می‌کند. کد زیر این پیاده‌سازی را نشان می‌دهد:

```

1. #include <stdbool.h>
2. #include <stdint.h>
3. #include "nrf.h"
4. #include "nrf_drv_gpiote.h"
5. #include "app_error.h"
6. #include "boards.h"
7. #include "bsp.h"
8.
9. #define redled 22
10. #define greenled 17 // blue and green are misplaced.
11. #define blueled 23
12. #define button 25
13. #define micro_power_ctrl 18
14.
15. enum color_t {R ,G ,B} color = R;
16.
17. void in_pin_handler(nrf_drv_gpiote_pin_t pin, nrf_gpiote_polarity_t action)
18. {
19.     switch (color)
20.     {
21.         case R:
22.             nrf_gpio_pin_set(redled);
23.             nrf_gpio_pin_clear(greenled);
24.             nrf_gpio_pin_clear(blueled);
25.             color = G;
26.             break;
27.         case G:
28.             nrf_gpio_pin_clear(redled);
29.             nrf_gpio_pin_set(greenled);
30.             nrf_gpio_pin_clear(blueled);
31.             color = B;
32.             break;
33.         default:
34.             nrf_gpio_pin_clear(redled);
35.             nrf_gpio_pin_clear(greenled);
36.             nrf_gpio_pin_set(blueled);
37.             color = R;
38.             break;

```

```

39.    }
40. }
41.
42. static void gpio_init(void)
43. {
44.     ret_code_t err_code;
45.
46.     err_code = nrf_drv_gpiote_init();
47.     APP_ERROR_CHECK(err_code);
48.
49.     nrf_drv_gpiote_in_config_t in_config = GPIOTE_CONFIG_IN_SENSE_LOTOHI(true);
50.     in_config.pull = NRF_GPIO_PIN_PULLDOWN;
51.
52.     err_code = nrf_drv_gpiote_in_init(button, &in_config, in_pin_handler);
53.     APP_ERROR_CHECK(err_code);
54.
55.     nrf_drv_gpiote_in_event_enable(button, true);
56. }
57.
58. int main(void)
59. {
60.     nrf_gpio_cfg_output(redled);
61.     nrf_gpio_cfg_output(greenled);
62.     nrf_gpio_cfg_output(blueled);
63.     nrf_gpio_cfg_input(button, NRF_GPIO_PIN_PULLDOWN);
64.
65.     nrf_gpio_cfg_output(micro_power_ctrl);
66.     nrf_gpio_pin_set(micro_power_ctrl);
67.
68.     gpio_init();
69.
70.     while (true)
71.     {
72.         // Do nothing.
73.     }
74. }
```

## ۲-۵- توضیحات کد

در این کد، از GPIOTE برای ایجاد وقفه و تغییر رنگ LED استفاده شده است (شکل ۱-۵). به این صورت که با فشردن دکمه، وقفه‌ای ایجاد می‌شود و تابع in\_pin\_handler فراخوانی می‌شود که وظیفه تغییر رنگ LED را بر عهده دارد.

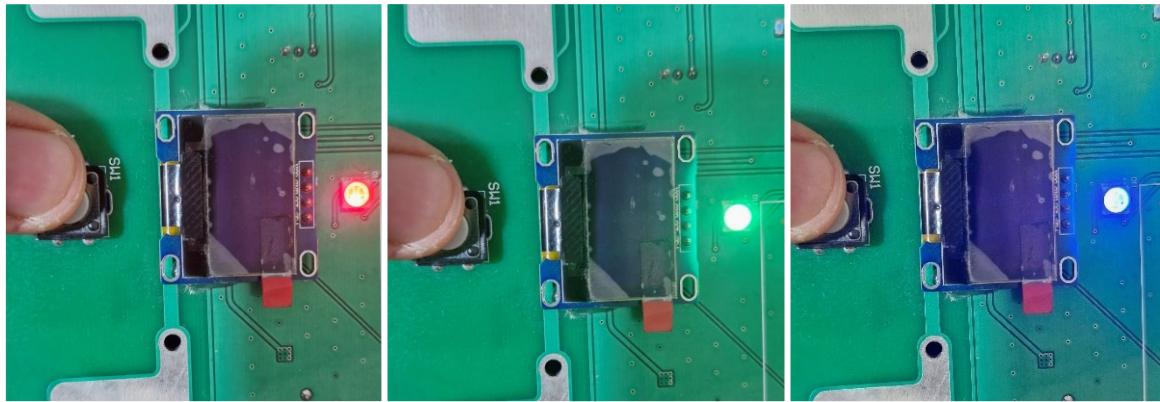
- ابتدا پین‌های مربوط به LEDها و دکمه به صورت ثابت تعریف شده‌اند. سپس، پین‌های خروجی برای LEDها و پین ورودی برای دکمه پیکربندی می‌شوند.

- در تابع gpio\_init، ابتدا GPIOTE مقداردهی اولیه می‌شود. سپس پیکربندی وقفه برای دکمه انجام شده و در نهایت، وقفه برای دکمه فعال می‌شود.

- در تابع in\_pin\_handler، بسته به رنگ فعلی، LED مناسب روشن و سایر LEDها خاموش می‌شوند. این تغییر رنگ با استفاده از یک ساختار switch انجام می‌شود.

- در تابع main، ابتدا پین‌های LED و دکمه پیکربندی می‌شوند و سپس تابع gpio\_init فراخوانی

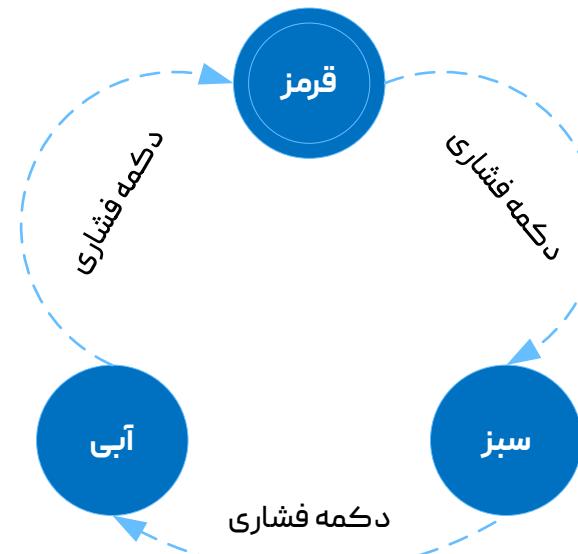
می شود تا GPIOTE و وقفه ها مقداردهی اولیه شوند.



شکل (۱-۵) قرمز، سبز، و آبی

### ۳-۵- ماشین حالت

در این کد نیز از یک ماشین حالت ساده متناظر با شکل ۲-۵، برای تغییر رنگ LED استفاده شده است. ماشین حالت به این صورت عمل می کند که هر بار دکمه فشرده می شود، دستگاه به حالت بعدی حرکت می کند و LED مربوط به آن رنگ روشن می شود.



شکل (۲-۵) نمودار حالت

#### ۴-۵ - نتیجه‌گیری

این پژوهه نمونه‌ای پیشرفته‌تر از نحوه کار با GPIO، GPIOTE و وقتهای در میکروکنترلر nRF52832 است. توضیحات کامل‌تر و نحوه اجرای این کد و نتایج آن در این فصل به تفصیل بررسی شده‌اند. این پیاده‌سازی، پلیهای برای درک بهتر کار با وقتهای و کاربردهای آن‌ها در پروژه‌های اینترنت اشیا فراهم می‌کند. برای توضیحات و اطلاعات بیشتر، می‌توانید به فیلم ضبط شده این فصل مراجعه کنید که کد آن در شکل ۳-۵ ارائه شده است.



شکل (۳-۵) کد QR

## فصل ششم

### راهاندازی RGB با PWM و GPIOTE

۱-۶- کد

در این بخش، به راهاندازی RGB با استفاده از PWM و GPIOTE برای تنظیم شدت نور LED قرمز پرداخته شده است. در این پروژه، با فشردن دکمه، وقفه‌ای ایجاد می‌شود و شدت نور LED قرمز به صورت تدریجی کاهش یا افزایش می‌یابد. کد زیر این پیاده‌سازی را نشان می‌دهد:

```

1. #include <stdbool.h>
2. #include <stdint.h>
3. #include "nrf.h"
4. #include "app_error.h"
5. #include "nrf_gpio.h"
6. #include "nrf_delay.h"
7. #include "app_pwm.h"
8. #include "nrf_drv_gpiote.h"
9.
10. #define button 25
11. #define redled 22
12. #define micro_power_ctrl 18
13.
14. int brightness = 0, d = -25;
15.
16. APP_PWM_INSTANCE(PWM1, 1); // Create the instance "PWM1" using TIMER1.
17.
18. void in_pin_handler(nrf_drv_gpiote_pin_t pin, nrf_gpiote_polarity_t action)
19. {
20.     if (brightness == 100 || brightness == 0) d = -d;
21.     brightness += d;
22. }
23.
24. static void gpio_init(void)
25. {
26.     ret_code_t err_code;
27.
28.     err_code = nrf_drv_gpiote_init();
29.     APP_ERROR_CHECK(err_code);
30.
31.     nrf_drv_gpiote_in_config_t in_config = GPIOE_CONFIG_IN_SENSE_LOTOHI(true);
32.     in_config.pull = NRF_GPIO_PIN_PULLDOWN;
33.
34.     err_code = nrf_drv_gpiote_in_init(button, &in_config, in_pin_handler);
35.     APP_ERROR_CHECK(err_code);
36.
37.     nrf_drv_gpiote_in_event_enable(button, true);
38. }
```

```

39.
40. int main(void)
41. {
42.     nrf_gpio_cfg_output(micro_power_ctrl);
43.     nrf_gpio_pin_set(micro_power_ctrl);
44.     nrf_gpio_cfg_input(button, NRF_GPIO_PIN_PULLDOWN);
45.
46.     ret_code_t err_code;
47.     gpio_init();
48.
49.     app_pwm_config_t pwm1_cfg = APP_PWM_DEFAULT_CONFIG_1CH(5000L, redled);
50.
51.     err_code = app_pwm_init(&PWM1, &pwm1_cfg, NULL);
52.     APP_ERROR_CHECK(err_code);
53.     app_pwm_enable(&PWM1);
54.
55.     while (true)
56.     {
57.         while (app_pwm_channel_duty_set(&PWM1, 0, (100 - brightness)) == NRF_ERROR_BUSY);
58.     }
59. }
60.

```

## ۲-۶- توضیحات کد

این کد از GPIOTE برای ایجاد وقفه و از PWM برای تنظیم شدت نور LED قرمز استفاده می‌کند. هدف این است که با فشردن دکمه، شدت نور LED به صورت تدریجی کاهش یا افزایش یابد (شکل ۱-۶ و شکل ۲-۶).

در ابتدای کد، پین‌های مربوط به LED قرمز و دکمه تعريف و پیکربندی شده‌اند.

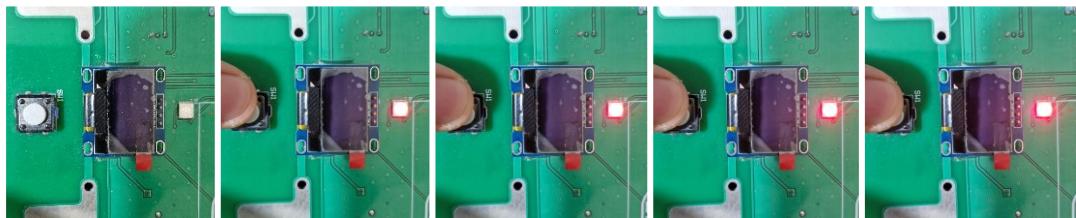
در این کد، از کتابخانه‌های PWM و GPIOTE استفاده شده است تا بتوان با فشردن دکمه، وقفه‌ای ایجاد کرده و شدت نور LED را تنظیم کرد.

- تابع in\_pin\_handler به عنوان هندرلر وقفه تعريف شده است. این تابع با هر بار فشردن دکمه فراخوانی می‌شود و مقدار brightness را تغییر می‌دهد. اگر مقدار brightness به ۰ یا ۱۰۰ برسد، جهت تغییرات (d) معکوس می‌شود.

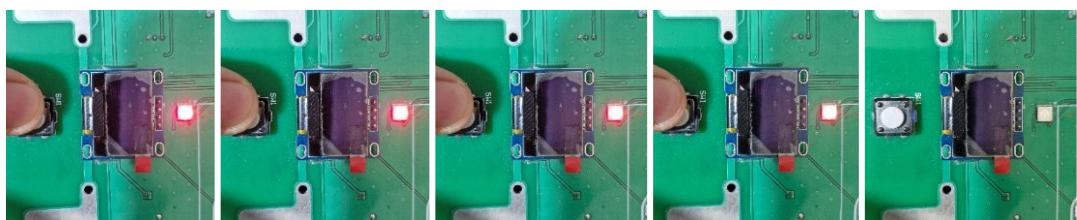
- در تابع gpio\_init، مقداردهی اولیه شده و پیکربندی وقفه برای دکمه انجام می‌شود.

- در تابع main، ابتدا پین‌های مربوط به LED و دکمه پیکربندی می‌شوند و سپس تابع gpio\_init فراخوانی می‌شود تا PWM و وقفه‌ها مقداردهی اولیه شوند. همچنین، مقداردهی اولیه شده و فعال می‌شود.

- در حلقه اصلی برنامه، مقدار brightness برای تنظیم شدت نور LED استفاده می‌شود. این مقدار به تابع app\_pwm\_channel\_duty\_set ارسال می‌شود تا PWM را تنظیم کند.

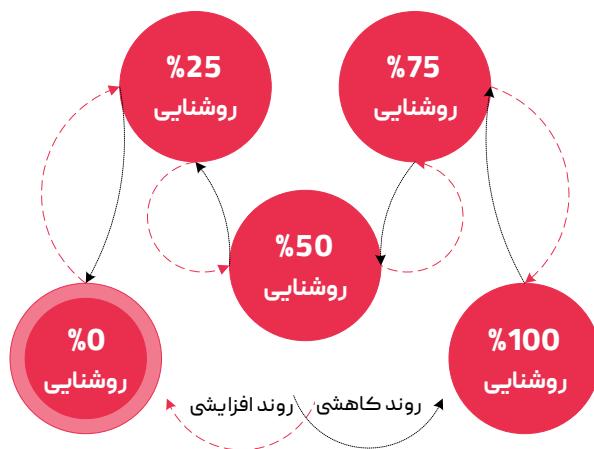


شکل (۱-۶) روند افزایشی



شکل (۲-۶) روند کاهشی

روند کد به طور خلاصه در شکل ۳-۶ به تصویر کشیده شده است:



شکل (۳-۶) روند نما

### ۶-۳- نتیجه‌گیری

این کد نمونه‌ای از نحوه استفاده از PWM و GPIOTE در میکروکنترلر nRF52832 برای تنظیم

شدت نور LED است. با هر بار فشردن دکمه، شدت نور LED قرمز به صورت تدریجی کاهش یا افزایش می‌یابد. این پیاده‌سازی، مفاهیم مهمی مانند وقفه‌ها و PWM را در کاربردهای عملی نشان می‌دهد و پایه‌ای برای پروژه‌های پیچیده‌تر فراهم می‌کند. برای توضیحات و اطلاعات بیشتر، می‌توانید به فیلم ضبط شده این فصل مراجعه کنید که کد QR آن در شکل ۴-۶ ارائه شده است.



شکل (۴-۶) کد QR

## فصل هفتم

### ارسال متن به Debug Terminal

۱-۷ - کد

در این بخش، نحوه ارسال متن به Debug Terminal با استفاده از امکانات داخلی SEGGER برای ارسال متن به پایانه دیباگ توضیح داده شده است. کد زیر نشان می‌دهد که چگونه می‌توان با استفاده از یک شمارنده، متن را به پایانه دیباگ ارسال کرد:

```

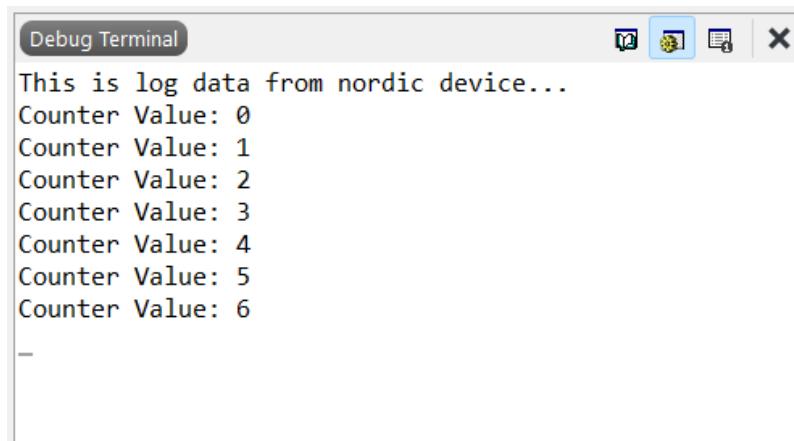
1. #include <stdbool.h>
2. #include <stdint.h>
3. #include "nrf.h"
4. #include "nordic_common.h"
5. #include "boards.h"
6. #include "nrf_delay.h"
7.
8. /**
9.  * @brief Function for application main entry.
10. */
11.
12. uint32_t count = 0;
13.
14. int main(void)
15. {
16.     nrf_gpio_cfg_output(22);
17.     printf("This is log data from nordic device...\n");
18.     while (true)
19.     {
20.         printf("Counter Value: %d\n", count);
21.         nrf_gpio_pin_set(22);
22.         nrf_delay_ms(500); // delay for 500 ms
23.         count++; // increment the counter by 1
24.     }
25. }
```

۲-۷ - توضیحات کد

پیکربندی پین خروجی: با استفاده از `nrf_gpio_cfg_output`، پین ۲۲ به عنوان خروجی پیکربندی شده است تا بتوان LED را واپايش کرد.

ارسال پیام اولیه: با استفاده از `printf("This is log data from nordic device...\n")`, یک پیام خوشامدگویی به پایانه دیباگ ارسال می‌شود (شکل ۷-۱).

حلقه بینهایت: در حلقه بینهایت، مقدار شمارنده به پایانه دیباگ ارسال می‌شود `(printf("Counter Value: %d\n", count))` روشن می‌شود LED Value: `%d\n", count)` و سپس تأخیری به مدت ۵۰۰ میلیثانیه اعمال می‌شود `(nrf_delay_ms(500))`. در نهایت، مقدار شمارنده افزایش می‌یابد! `++count`، نکته مهم این است که `printf` فقط داخل محیط دیباگ قابل استفاده است!



شکل (۱-۷) دیباگ ترمینال داخلی SEGGER

### ۳-۷- نتیجه‌گیری

این کد نمونه‌ای ساده از نحوه استفاده از توابع استاندارد C برای ارسال اطلاعات به پایانه دیباگ است. این روش می‌تواند به برنامه‌نویسان کمک کند تا فرآیندهای داخلی میکروکنترلر را در حین اجرا مشاهده و اشکال‌زدایی کنند. برای توضیحات و اطلاعات بیشتر، می‌توانید به فیلم ضبطشده این فصل مراجعه کنید که کد QR آن در شکل ۷-۲ ارائه شده است.



شکل (۲-۷) کد QR

## فصل هشتم

### کار با SSD 1306 مدل OLED

۱-۸- کد

در این بخش، نحوه نمایش اطلاعات بر روی نمایشگر OLED با استفاده از پروتکل‌های I2C یا TWI توضیح داده شده است. کد زیر نحوه ارتباط با نمایشگر OLED و نمایش اطلاعات را نشان می‌دهد:

```

1. #include <stdio.h>
2. #include <stdbool.h>
3. #include <stdint.h>
4. #include "boards.h"
5. #include "app_util_platform.h"
6. #include "app_error.h"
7. #include "nrf_drv_twi.h"
8. #include "nrf_delay.h"
9.
10. #include "ssd_lcd_By_Naghiloo.c"
11.
12. #define micro_power_ctrl 18
13.
14. /**
15. * @brief TWI events handler.
16. */
17. void twi_handler(nrf_drv_twi_evt_t const * p_event, void * p_context)
18. {
19.     switch (p_event->type)
20.     {
21.         case NRF_DRV_TWI_EVT_DONE:
22.             m_xfer_done = true;
23.             break;
24.         default:
25.             break;
26.     }
27. }
28.
29. /**
30. * @brief TWI initialization.
31. */
32. void twi_init (void)
33. {
34.     ret_code_t err_code;
35.
36.     const nrf_drv_twi_config_t twi_config = {
37.         .scl          = TWI_SCL_SSD,
38.         .sda          = TWI_SDA_SSD,
39.         .frequency   = NRF_DRV_TWI_FREQ_400K,
40.         .interrupt_priority = APP_IRQ_PRIORITY_HIGH,
41.         .clear_bus_init = false

```

```

42.     };
43.
44.     err_code = nrf_drv_twi_init(&m_twi, &twi_config, twi_handler, NULL);
45.     APP_ERROR_CHECK(err_code);
46.
47.     nrf_drv_twi_enable(&m_twi);
48. }
49.
50. /**
51. * @brief Function for main application entry.
52. */
53. int main(void)
54. {
55.     nrf_gpio_cfg_output(micro_power_ctrl);
56.     nrf_gpio_pin_set(micro_power_ctrl);
57.
58.     twi_init();
59.     LCD_init();
60.
61.     draw(0,0,KNTU);
62. }
63.
64. /** @} */
65.

```

## ۲-۸- توضیحات کد

پیکربندی پین خروجی: با استفاده از `nrf_gpio_cfg_output(micro_power_ctrl)`, پین مربوط به واپایش توان میکروکنترلر به عنوان خروجی پیکربندی شده و فعال می‌شود.  
`(nrf_gpio_pin_set(micro_power_ctrl))`

پیکربندی و راهاندازی TWI: تابع `twi_init` پیکربندی لازم برای ارتباط (I2C) TWI را انجام می‌دهد. این تابع شامل تنظیمات پایه‌های SCL و SDA و بسامد ارتباطی است. پس از انجام تنظیمات، TWI فعال می‌شود.

راهاندازی نمایشگر: تابع `LCD_init` برای راهاندازی اولیه نمایشگر OLED استفاده می‌شود. این تابع در پرونجا c `ssd_lcd_By_Naghiloo.c` تعریف شده است. با استفاده از تابع `draw(0,0,KNTU)`, اطلاعات روی نمایشگر OLED نمایش داده می‌شود که در شکل ۱-۸ قابل مشاهده است.



شکل (۱-۸) نمایش نماد دانشگاه و شرکت روی OLED

لازم به ذکر هست که پرونجا ssd\_lcd\_By\_Naghiloo.c توسط آقایان نقی لو (خودم) و محمدپور تکمیل شده و در آن تابع‌های خوبی نظیر draw, uart\_put\_str, uart\_get\_str، و کد هگز تصاویر زیادی ضمیمه شده است. برای مطالعه دقیق هر تابع و یا عکس، می‌توان وارد پرونجا شود و یا حتی خودتان کد تصاویر دلخواهتان را با نرم‌افزار بسیار کاربردی شکل (۲-۸) زیر یعنی LCD Vision، به آرایه‌ای از کد هگز تبدیل کنید و در کتابخانه قرار دهید!



شکل (۲-۸) نرم افزار LCD Vision

### ۳-۸- نتیجه‌گیری

این کد نمونه‌ای از نحوه استفاده از پروتکل‌های I2C برای ارتباط با نمایشگر OLED و نمایش اطلاعات است. این روش می‌تواند به برنامه‌نویسان کمک کند تا دستگاه‌های مختلف را با استفاده از پروتکل‌های ارتباطی استاندارد به میکروکنترلر متصل کنند و اطلاعات موردنیاز را نمایش دهند. برای توضیحات و اطلاعات بیشتر، می‌توانید به فیلم ضبط شده این فصل مراجعه کنید که کد QR آن در شکل ۳-۸ ارائه شده است.



شکل (۳-۸) کد QR



## فصل نهم

### کار با Timer و پروتکل UART

۱-۹ - کد

این کد وظیفه واپایش LEDهای RGB، نمایش عدد تایمر و ارسال مقادیر آن از طریق UART و به کمک رابط USB to TTL مدل CP2102 را بر عهده دارد. همچنین با استفاده از ورودی‌های کیبورد، LEDهای مختلف را واپایش کرده و اطلاعات مربوط به رنگ و عدد تایمر را روی نمایشگر OLED نمایش می‌دهد.

```

1. #include <stdbool.h>
2. #include <stdint.h>
3. #include <stdio.h>
4. #include "string.h"
5. #include "nrf_gpio.h"
6. #include "app_uart.h"
7. #include "app_util_platform.h"
8. #include "app_error.h"
9. #include "nrf_delay.h"
10. #include "nrf.h"
11. #include "nrf_drv_twi.h"
12. #include "app_timer.h"
13. #include "nrf_drv_clock.h"
14. #include "nrf_uart.h"
15. #include "ssd_lcd_By_Naghiloo.c"
16.
17. #define redled 22
18. #define greenled 17 // blue and green are misplaced.
19. #define blueled 23
20. #define micro_power_ctrl 18
21. #define empty 0
22.
23. #define UART_TX_BUF_SIZE 8 /*< UART TX buffer size. */
24. #define UART_RX_BUF_SIZE 8 /*< UART RX buffer size. */
25.
26. char in_text [100] = {};
27. char out_text [100] = {};
28.
29. int sec = 0;
30. int min = 0;
31.
32. #define INTERVAL APP_TIMER TICKS(1000)
33.
34. APP_TIMER_DEF(m_app_timer_id);
35.
36. static void lfcclk_config(void)
37. {

```

```

38.     ret_code_t err_code = nrf_drv_clock_init();
39.     APP_ERROR_CHECK(err_code);
40.     nrf_drv_clock_lfcclk_request(NULL);
41. }
42.
43. bool flag = true;
44.
45. static void app_timer_handler(void * p_context)
46. {
47.     char temp[100];
48.     sec++;
49.     if(sec == 60)
50.     {
51.         sec = 0;
52.         min++;
53.     }
54.
55.     if (strcmp(&in_text,"clock") == 0)
56.     {
57.         if(flag)
58.         {
59.             flag = false;
60.             LCD_Clear();
61.             nrf_gpio_pin_clear(redled);
62.             nrf_gpio_pin_clear(blueled);
63.             nrf_gpio_pin_clear(greenled);
64.         }
65.
66.         draw(38,3,digit(min/10));
67.         draw(51,3,digit(min%10));
68.         draw(64,3,colon);
69.         draw(77,3,digit(sec/10));
70.         draw(90,3,digit(sec%10));
71.
72.         nrf_gpio_pin_toggle(redled);
73.         nrf_gpio_pin_toggle(blueled);
74.         nrf_gpio_pin_toggle(greenled);
75.     }
76.
77.     sprintf(&temp, "%d:%d\n",min,sec);
78.     uart_put_str(temp);
79. }
80.
81. static void timers_init(void)
82. {
83.     ret_code_t err_code;
84.
85.     err_code = app_timer_init();
86.     APP_ERROR_CHECK(err_code);
87.
88.     err_code = app_timer_create(&m_app_timer_id, APP_TIMER_MODE_REPEATED,
app_timer_handler);
89.     APP_ERROR_CHECK(err_code);
90. }
91.
92. void uart_error_handle(app_uart_evt_t * p_event)
93. {
94.     if (p_event->evt_type == APP_UART_COMMUNICATION_ERROR)
95.     {
96.         APP_ERROR_HANDLER(p_event->data.error_communication);
97.     }
98.     else if (p_event->evt_type == APP_UART_FIFO_ERROR)
99.     {
100.         APP_ERROR_HANDLER(p_event->data.error_code);
101.     }
102. }
103.
104. void twi_handler(nrf_drv_twi_evt_t const * p_event, void * p_context)

```

```

105. {
106.     switch (p_event->type)
107.     {
108.         case NRF_DRV_TWI_EVT_DONE:
109.             m_xfer_done = true;
110.             break;
111.         default:
112.             break;
113.     }
114. }
115.
116. void twi_init (void)
117. {
118.     ret_code_t err_code;
119.
120.     const nrf_drv_twi_config_t twi_config = {
121.         .scl      = TWI_SCL_SSD,
122.         .sda      = TWI_SDA_SSD,
123.         .frequency = NRF_DRV_TWI_FREQ_400K,
124.         .interrupt_priority = APP_IRQ_PRIORITY_HIGH,
125.         .clear_bus_init = false
126.     };
127.
128.     err_code = nrf_drv_twi_init(&m_twi, &twi_config, twi_handler, NULL);
129.     APP_ERROR_CHECK(err_code);
130.
131.     nrf_drv_twi_enable(&m_twi);
132. }
133.
134. int main(void)
135. {
136.     twi_init();
137.     LCD_init();
138.
139.     nrf_gpio_cfg_output(redled);
140.     nrf_gpio_cfg_output(greenled);
141.     nrf_gpio_cfg_output(blueled);
142.     nrf_gpio_cfg_output(micro_power_ctrl);
143.     nrf_gpio_pin_set(micro_power_ctrl);
144.
145.     uint32_t err_code;
146.
147.     const app_uart_comm_params_t comm_params =
148.     {
149.         7,
150.         11,
151.         empty,
152.         empty,
153.         APP_UART_FLOW_CONTROL_DISABLED,
154.         false,
155.         NRF_UART_BAUDRATE_115200
156.     };
157.
158.     APP_UART_FIFO_INIT(&comm_params,
159.                         UART_RX_BUF_SIZE,
160.                         UART_TX_BUF_SIZE,
161.                         uart_error_handle,
162.                         APP_IRQ_PRIORITY_LOWEST,
163.                         err_code);
164.
165.     APP_ERROR_CHECK(err_code);
166.
167.     lfclk_config();
168.     timers_init();
169.     err_code = app_timer_start(m_app_timer_id, INTERVAL, NULL);
170.
171.     while (true)
172.     {

```

```

173.     uart_get_str(&in_text,true,false);
174.
175.     if (strcmp(&in_text,"red") == 0)
176.     {
177.         nrf_gpio_pin_set(redled);
178.         nrf_gpio_pin_clear(blueled);
179.         nrf_gpio_pin_clear(greenled);
180.         draw(0,0,redimage);
181.     }
182.
183.     if (strcmp(&in_text,"blue") == 0)
184.     {
185.         nrf_gpio_pin_clear(redled);
186.         nrf_gpio_pin_set(blueled);
187.         nrf_gpio_pin_clear(greenled);
188.         draw(0,0,blueimage);
189.     }
190.
191.     if (strcmp(&in_text,"green") == 0)
192.     {
193.         nrf_gpio_pin_clear(redled);
194.         nrf_gpio_pin_clear(blueled);
195.         nrf_gpio_pin_set(greenled);
196.         draw(0,0,greenimage);
197.     }
198.
199.     if(strcmp(&in_text,"clock") != 0)
200.         flag = true;
201.    }
202. }

```

## ۲-۹- توضیحات کد

این کد، یک رابط کاربری تعاملی برای واپايش اجزای مختلف دستگاه ایجاد می‌کند. با استفاده از این کد می‌توان LED‌های RGB را واپايش کرد، زمان را بر روی نمایشگر OLED نمایش داد و اطلاعات مربوط به دستگاه را از طریق UART ارسال نمود.

### تعريفها و متغیرها:

شامل تعريف پین‌های LED‌های قرمز، سبز و آبی، دکمه و واپايش توان میکروکنترلر است.

تعريف بافرهای ورودی و خروجی برای ذخیره‌سازی متن ورودی و خروجی از طریق UART.

تعريف و مقداردهی اولیه برای متغیرهای تایمر، شامل ثانیه و دقیقه.

### راهاندازی تایمر:

استفاده از توابع app\_timer\_create و app\_timer\_init برای راهاندازی تایمر نرمافزاری.

تعريف تابع app\_timer\_handler که در هر تیک تایمر اجرا می‌شود و وظیفه شمارش ثانیه‌ها و دقیقه‌ها و بهروزرسانی نمایشگر OLED را بر عهده دارد.

:پیکربندی UART

استفاده از app\_uart\_init برای تنظیم پارامترهای ارتباط UART از جمله سرعت انتقال داده و پین‌های مربوط به ارتباط.

:پیکربندی TWI (I2C)

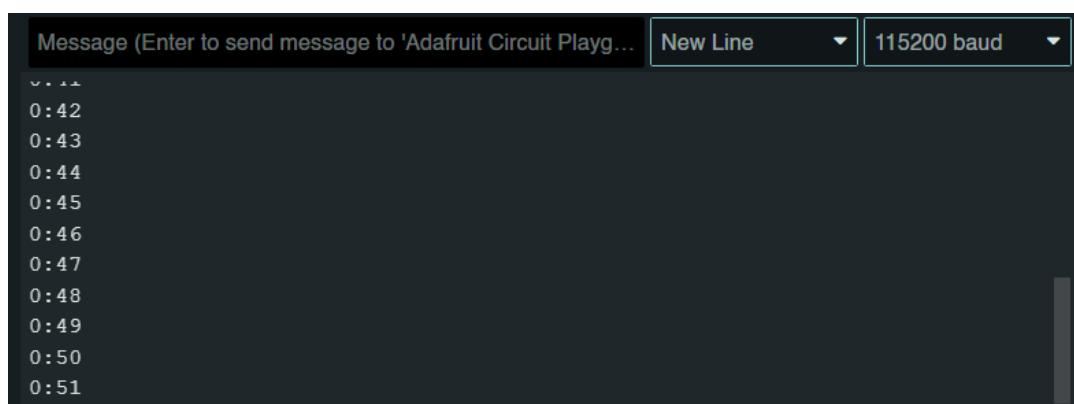
استفاده از nrf\_drv\_twi\_init برای پیکربندی و راهندازی ارتباط TWI برای ارتباط با نمایشگر OLED.

:تابع main

راهاندازی اولیه پین‌های GPIO برای LED‌ها و توان میکروکنترلر.

راهاندازی و شروع تایمر نرم‌افزاری.

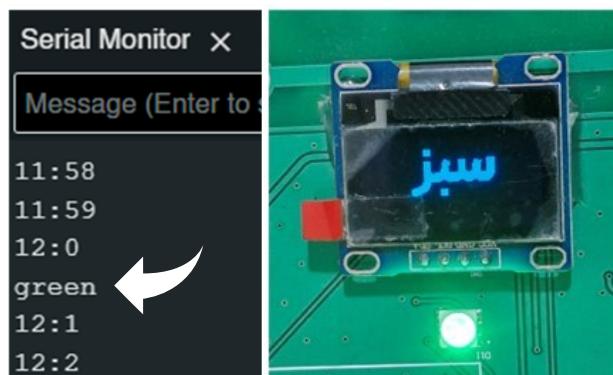
دریافت ورودی از طریق UART و تنظیم وضعیت LED‌ها بر اساس ورودی. اگر ورودی "red" باشد، LED مربوطه روشن می‌شود و تصویر مربوط به رنگ روی OLED نمایش داده می‌شود. اگر ورودی "green" یا "blue" باشد، تایمر نمایش داده می‌شود و هر سه LED روشن می‌شوند تا رنگ سفید را ایجاد کنند که این موارد در شکل‌های ۱-۹ تا ۵-۹ قابل مشاهده هست.



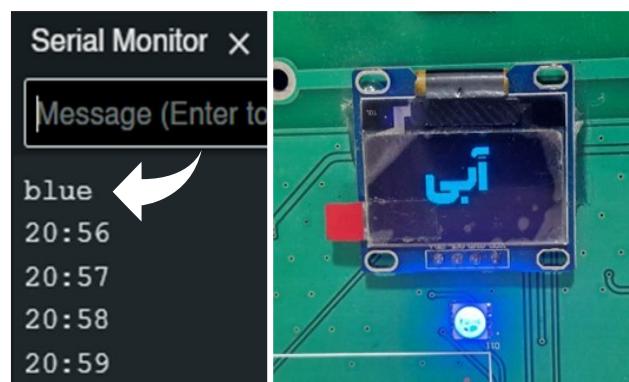
شکل (۱-۹) دریافت تایم و نمایش در ترمینال



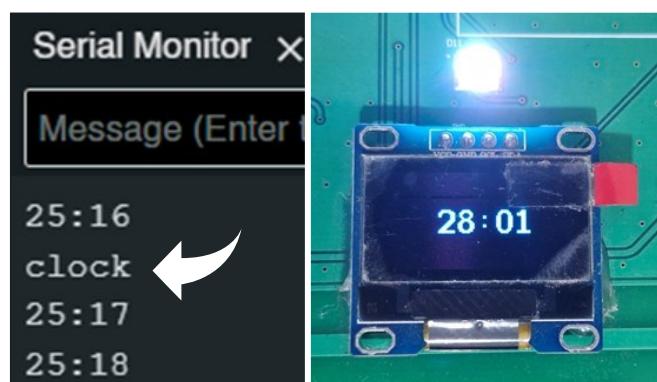
شکل (۲-۹) ارسال قرمز



شکل (۳-۹) ارسال سبز

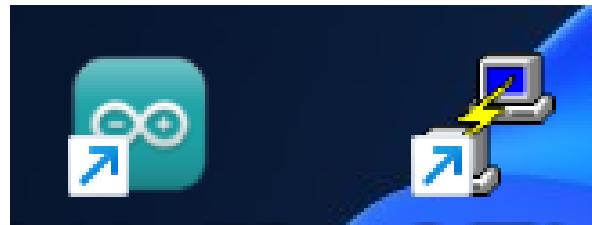


شکل (۴-۹) ارسال آبی



شکل (۵-۹) ارسال ساعت

لازم به ذکر است که تنها پایانه دارای کرکتر \n، پایانه موجود در Arduino IDE (شکل ۶-۹) هست و باید روی new line تنظیم شود و نباید سراغ پایانه های مشابه مثل PuTTY (شکل ۶-۹) و Hercules و... رفت. ضمناً درایور رابط CP2102 (شکل ۷-۹) هم، قبل اتصال به لپ تاپ باید نصب شود.



شکل (۶-۹) Arduino IDE و PuTTY



شکل (۷-۹) رابط CP2102 مدل USB to TTL

### ۳-۹- نتیجه‌گیری

این کد نشان‌دهنده توانایی میکروکنترلر در واپايش و هماهنگی میان چندین دستگاه جانبی از جمله LEDها، تایمر نرمافزاری، UART و نمایشگر OLED است. با استفاده از این کد، می‌توان به راحتی وضعیت LEDها را بر اساس ورودی‌های کیبورد تغییر داد و اطلاعات تایمر را به صورت همزمان از طریق UART و نمایشگر OLED مشاهده کرد. این قابلیت‌ها برای توسعه برنامه‌های پیچیده‌تر و کاربردهای متنوع در پروژه‌های اینترنت اشیا (IoT) بسیار مفید و کاربردی هستند.

برای توضیحات و اطلاعات بیشتر، می‌توانید به فیلم ضبط شده این فصل مراجعه کنید که کد QR آن در شکل ۸-۹ ارائه شده است.



شكل (۸-۹) کد QR

## فصل دهم

### SPI پروتکل

۱-۱۰ - کد

در این فصل، به نحوه استفاده از پروتکل SPI در میکروکنترلر NRF پرداخته می‌شود. با استفاده از این پروتکل، می‌توان به تبادل داده با دستگاه‌های جانبی مختلف مانند حسگرهای نمایشگرها و حافظه‌ها پرداخت. در این کد، مثالی از ارتباط SPI برای نوشتند و خواندن از یک رجیستر نمایش داده شده است.

```

1. #include <stdbool.h>
2. #include <stdint.h>
3. #include <stdio.h>
4. #include "app_error.h"
5. #include "nrf_delay.h"
6. #include "nrf_gpio.h"
7. #include "nrf_drv_spi.h"
8.
9. #define ledred 22
10.
11. #define SPI_BUFSIZE 8
12. uint8_t spi_tx_buf[SPI_BUFSIZE];
13. uint8_t spi_rx_buf[SPI_BUFSIZE];
14.
15. #define SPI_INSTANCE 0
16.
17. volatile uint8_t SPIReadLength, SPIWriteLength;
18.
19. static volatile bool spi_xfer_done;
20. static const nrf_drv_spi_t spi = NRF_DRV_SPI_INSTANCE(SPI_INSTANCE);
21.
22. void spi_event_handler(nrf_drv_spi_evt_t const * p_event, void * p_context)
23. {
24.     spi_xfer_done = true;
25. }
26.
27. void spi_init(void)
28. {
29.     nrf_drv_spi_config_t spi_config = NRF_DRV_SPI_DEFAULT_CONFIG;
30.
31.     spi_config.ss_pin    = 6;
32.     spi_config.miso_pin = 2;
33.     spi_config.mosi_pin = 3;
34.     spi_config.sck_pin  = 4;
35.     spi_config.frequency = NRF_DRV_SPI_FREQ_8M;
36.     spi_config.mode = NRF_DRV_SPI_MODE_0;
37.     spi_config.bit_order = NRF_DRV_SPI_BIT_ORDER_MSB_FIRST;
38.

```

```

39.     APP_ERROR_CHECK(nrf_drv_spi_init(&spi, &spi_config, spi_event_handler, NULL));
40. }
41.
42. static void adsWreg (uint8_t reg, uint8_t val)
43. {
44.     SPIWriteLength = 2;
45.     SPIReadLength = 0;
46.
47.     spi_tx_buf[0] = reg & ~(1 << 7);
48.     spi_tx_buf[1] = val;
49.
50.     spi_xfer_done = false;
51.
52.     APP_ERROR_CHECK(nrf_drv_spi_transfer(&spi, spi_tx_buf, SPIWriteLength, spi_rx_buf,
53.                                         SPIReadLength));
54.     while(!spi_xfer_done);
55. }
56.
57. int adsRreg(int reg)
58. {
59.     spi_tx_buf[0] = reg | (1 << 7);
60.
61.     spi_xfer_done = false;
62.
63.     APP_ERROR_CHECK(nrf_drv_spi_transfer(&spi, spi_tx_buf, 2, spi_rx_buf, 2));
64.
65.     while(!spi_xfer_done){};
66.
67.     return spi_rx_buf[1];
68. }
69.
70. int main(void)
71. {
72.     spi_init();
73.
74.     nrf_gpio_cfg_output(ledred);
75.
76.     while (true)
77.     {
78.         nrf_gpio_pin_toggle(ledred);
79.         adsWreg(0x21, 0x7);
80.         int val = adsRreg(0x21);
81.         printf("value: %d\r\n", val);
82.         nrf_delay_ms(500);
83.     }
84. }

```

## ۲-۱۰- توضیحات کد

این کد شامل بخش‌های زیر است:

تعریف‌ها و متغیرها:

شامل تعریف پین‌های LED قرمز و بافرهای انتقال و دریافت SPI.

تعریف متغیرهای طول خواندن و نوشتمن SPI.

راهاندازی SPI:

استفاده از `nrf_drv_spi_init` برای پیکربندی و راهاندازی ارتباط SPI. تنظیم پین‌های `sck_pin`, `mosi_pin`, `miso_pin` و `ss_pin` تنظیم بسامد، حالت و ترتیب بیت‌ها برای ارتباط SPI.

تابع نوشتن در رجیستر:

تابع `adsWreg` برای نوشتن یک مقدار در رجیستر مشخصی از دستگاه متصل به SPI استفاده می‌شود.

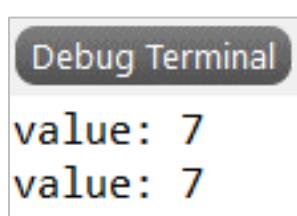
تنظیم طول نوشتن و خواندن، تنظیم پرچم انتقال و فراخوانی تابع انتقال SPI.

تابع خواندن از رجیستر:

تابع `adsRreg` برای خواندن یک مقدار از رجیستر مشخصی استفاده می‌شود. تنظیم پرچم انتقال و فراخوانی تابع انتقال SPI.

تابع اصلی:

راهاندازی اولیه پین LED و SPI در حلقه اصلی، LED قرمز روشن و خاموش می‌شود، مقدار مشخصی در یک رجیستر نوشته می‌شود و سپس مقدار خوانده شده از آن رجیستر چاپ می‌شود (شکل ۱۰-۱).



شکل (۱۰-۱) خواندن رجیستر

### ۳-۱۰- نتیجه‌گیری

این کد نشان‌دهنده توانایی میکروکنترلر در برقراری ارتباط با دستگاه‌های جانبی با استفاده از پروتکل SPI است. با استفاده از این کد، می‌توان به راحتی مقادیر را در رجیسترهاي دستگاه متصل به SPI نوشت و از آن‌ها خواند. این قابلیت‌ها برای توسعه برنامه‌های پیچیده‌تر و کاربردهای متنوع در پروژه‌های اینترنت اشیا (IoT) و دستگاه‌های تعبیه‌شده بسیار مفید و کاربردی هستند. برای توضیحات و اطلاعات بیشتر، می‌توانید به فیلم ضبط شده این فصل مراجعه کنید که کد QR آن در شکل ۲-۱۰ ارائه شده است.



شکل (۲-۱۰) کد QR

## فصل یازدهم

### کار با SAADC

۱-۱۱- کد

در این فصل، نحوه استفاده از SAADC (Successive Approximation Analog to Digital Converter) در میکروکنترلر NRF برای تبدیل سیگنال‌های آنالوگ به داده‌های دیجیتال را بررسی می‌کنیم. در این مثال، از SAADC برای خواندن مقدار ولتاژ از یک ورودی و تبدیل آن به مقدار دیجیتال استفاده شده است.

```

1. #include <stdbool.h>
2. #include <stdint.h>
3. #include <stdio.h>
4. #include <string.h>
5. #include "nrf.h"
6. #include "nrf_drv_saadc.h"
7. #include "boards.h"
8. #include "app_error.h"
9. #include "nrf_delay.h"
10.
11. #define micro_power_ctrl 18
12. #define RESOLUTION 10
13.
14. void saadc_init(void)
15. {
16.     ret_code_t err_code;
17.     nrf_saadc_channel_config_t channel_config =
18.         NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN4);
19.
20.     err_code = nrf_drv_saadc_init(NULL, NULL);
21.     APP_ERROR_CHECK(err_code);
22.
23.     err_code = nrf_drv_saadc_channel_init(0, &channel_config);
24.     APP_ERROR_CHECK(err_code);
25. }
26.
27. float volts;
28.
29. int main(void)
30. {
31.     nrf_gpio_cfg_output(micro_power_ctrl);
32.     nrf_gpio_pin_set(micro_power_ctrl);
33.
34.     saadc_init();
35.     nrf_saadc_resolution_set(RESOLUTION);
36.

```

```

37.     nrf_saadc_value_t adc_val;
38.
39.     while (true)
40.     {
41.         nrf_drv_saadc_sample_convert(0, &adc_val);
42.         volts = adc_val * ((1.0 / 6.0) / (0.6 * 1024.0)) * 3.29;
43.         nrf_delay_ms(500);
44.     }
45. }
```

## ۲-۱۱- توضیحات کد

تعریف‌ها:

micro\_power\_ctrl به پین شماره ۱۸ اختصاص داده شده است.

RESOLUTION مقدار ۱۰ بیت برای رزولوشن ADC تعریف شده است.

:saadc\_init تابع

این تابع برای تنظیم و پیکربندی SAADC استفاده می‌شود.

کانال ورودی NRF\_SAADC\_INPUT\_AIN4 به SAADC تنظیم شده است.

SAADC و کانال ورودی آن با استفاده از توابع nrf\_drv\_saadc\_init و nrf\_drv\_saadc\_init آن را انتخاب می‌کنند. این توابع مقداردهی اولیه می‌شوند.

:volts و متغیر nrf\_drv\_saadc\_sample\_convert تابع

این تابع دو ورودی دریافت می‌کند که اولی کانال و دیگری متغیری است که مقدار اندازه‌گیری شده در آن ذخیره می‌شود و سپس با فرمول (۱-۱۱) در متغیر volts، برای نمایش دادن، ذخیره می‌شود:

$$\text{volts} = [V_P - V_N] * \text{Gain} / (\text{Ref} * 2^{\text{Resolution}}) \quad (11-1)$$

البته به علت اینکه ولتاژ رفرنس ما دقیقاً مشخص نبود، باید ضریب دیگری به صورت تجربی در فرمول ضرب می‌شد.

تابع main

پین micro\_power\_ctrl به خروجی تنظیم شده و مقداردهی اولیه می‌شود.

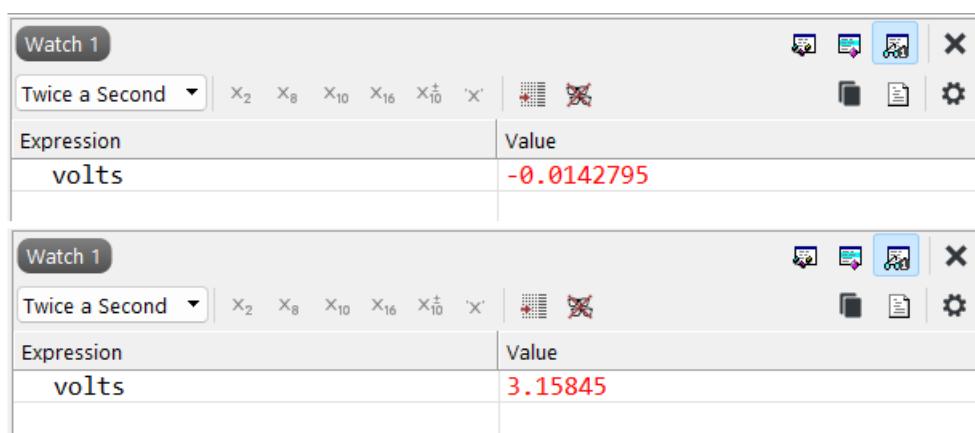
مقداردهی اولیه می‌شود و رزولوشن آن تنظیم می‌شود.

در یک حلقه بینهایت، مقدار ولتاژ از ورودی SAADC خوانده شده و به مقدار دیجیتال تبدیل می‌شود. سپس مقدار دیجیتال به ولتاژ معادل تبدیل شده و در متغیر volts ذخیره می‌شود. این عملیات هر ۵۰۰ میلی ثانیه تکرار می‌شود و در نهایت به کمک سیگنال ژنراتور دو کاناله نمانام MEGATEK که در شکل ۱-۱۱ مشاهده می‌شود، به پایه‌ی آنalog میکروکنترلر، سیگنال تپی با مشخصات مذکور در همان شکل ۱-۱۱ داده شد.



شکل (۱-۱۱) سیگنال ژنراتور و مشخصات تپ

در نهایت عدد خروجی، یعنی مقدار نمونه برداری شده در ۱۰ بیت، بسیار نزدیک به مقادیر اصلی تپ بود که در شکل ۲-۱۱ قابل مشاهده است.



شکل (۲-۱۱) مشاهده خروجی

### ۳-۱۱- نتیجه‌گیری

در این فصل، آموخته شد که چگونه از SAADC برای اندازه‌گیری ولتاژ آنالوگ و تبدیل آن به مقدار دیجیتال استفاده کنید. SAADC یک ابزار قدرتمند برای خولندن مقادیر آنالوگ در میکروکنترلرهای nRF است و با تنظیمات مناسب، می‌توان دقت و رزولوشن<sup>۱</sup> موردنیاز را به دست آورد. این مبدل در کاربردهای مختلف مانند اندازه‌گیری حسگرهای آنالوگ، خواندن ولتاژ باتری و سایر سیگنال‌های آنالوگ بسیار مفید است. برای توضیحات و اطلاعات بیشتر، می‌توانید به فیلم ضبطشده این فصل مراجعه کنید که کد QR آن در شکل ۳-۱۱ ارائه شده است.



شکل (۳-۱۱) کد QR

<sup>۱</sup> Resolution

## فصل دوازدهم

### Bluetooth Low Energy مبانی

#### ۱-۱۲ - مقدمه

بلوتوث کم مصرف (BLE) یا Bluetooth Smart (شکل ۱-۱۲) که به عنوان Bluetooth Low Energy نیز شناخته می‌شود، توسط گروه علاقه‌مند ویژه بلوتوث<sup>۱</sup> توسعه داده شده است و به عنوان یک فناوری نوآورانه هدف دارد تا جایگزین بسیاری از فناوری‌های بی‌سیم استاندارد موجود در بازار شود. این فناوری به دلیل عملکرد خوب و گستردگی استفاده، به گزینه‌ای ایده‌آل برای کاربردهای متنوعی تبدیل شده است. BLE امروزه در تمامی رایانه‌ها، تبلت‌ها و تلفن‌های هوشمند موجود است و در حوزه‌هایی نظیر سلامت الکترونیک، اتومبیل‌سازی، ارتباطات صوتی، خانه‌های هوشمند، واپیش بازی‌ها، دستگاه‌های امنیتی و اینترنت اشیا (IoT) کاربردهای فراوانی دارد. در این فصل موارد زیادی به [۱]، [۲]، و [۳] ارجاع داده شده است که شکل‌ها از [۳] می‌باشند.



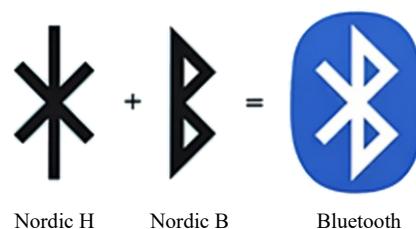
شکل (۱-۱۲) نماد قدیمی BLE

این فصل به بررسی پروتکل Bluetooth Low Energy و ساختار آن می‌پردازد. BLE به عنوان یک فناوری کم مصرف، امکان استفاده در دستگاه‌های کوچک با باتری‌های کم حجم را فراهم می‌کند و همین ویژگی آن را به یکی از فناوری‌های انقلابی در بازار ارتباطات بی‌سیم تبدیل کرده است. نام "بلوتوث" از

---

<sup>۱</sup> Bluetooth Special Interest Group (SIG)

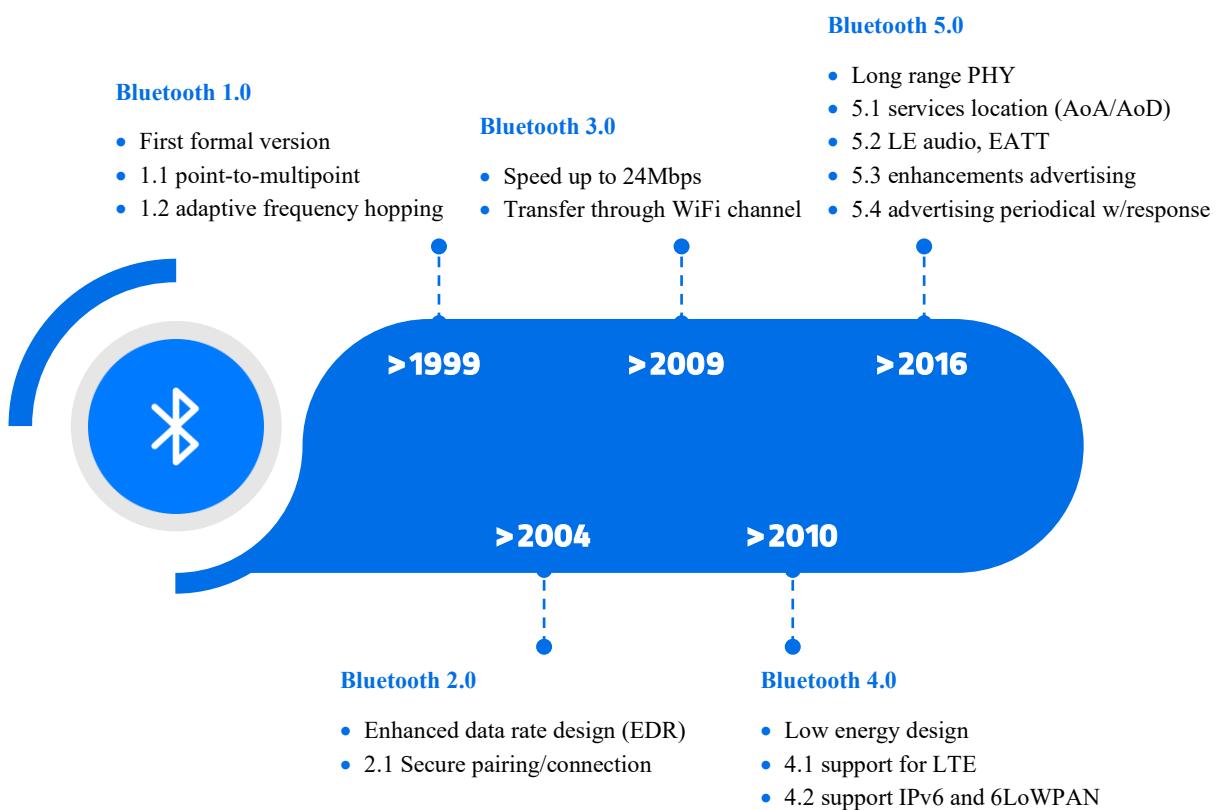
هارالد بلوتوث، پادشاه دانمارکی الهام گرفته شده است (شکل ۲-۱۲). این نام به این دلیل انتخاب شد که نشان‌دهنده توانایی این فناوری در اتصال و یکپارچه‌سازی دستگاه‌های مختلف است، درست همانطور که هارالد بلوتوث پادشاهی‌های مختلف را متحد کرد.



شکل (۲-۱۲) فلسفه نام‌گذاری

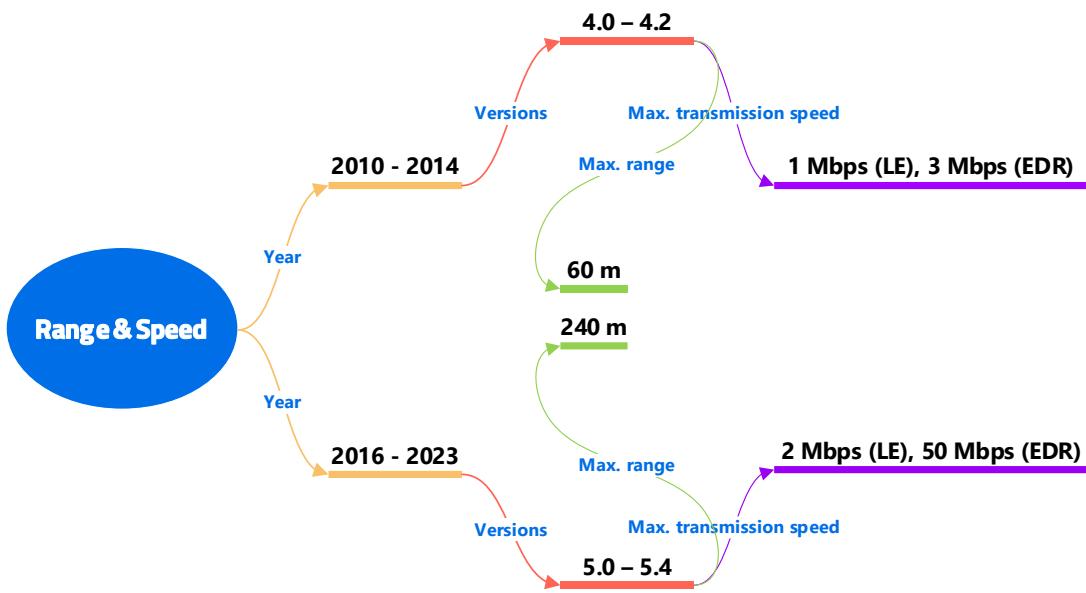
## ۲-۱۲- نسخه‌های بلوتوث

در شکل ۳-۱۲ خلاصه‌ی زیر، ویژگی‌های کلیدی نسخه‌های مختلف بلوتوث مرور می‌شود.



شکل (۳-۱۲) سیر تکامل بلوتوث

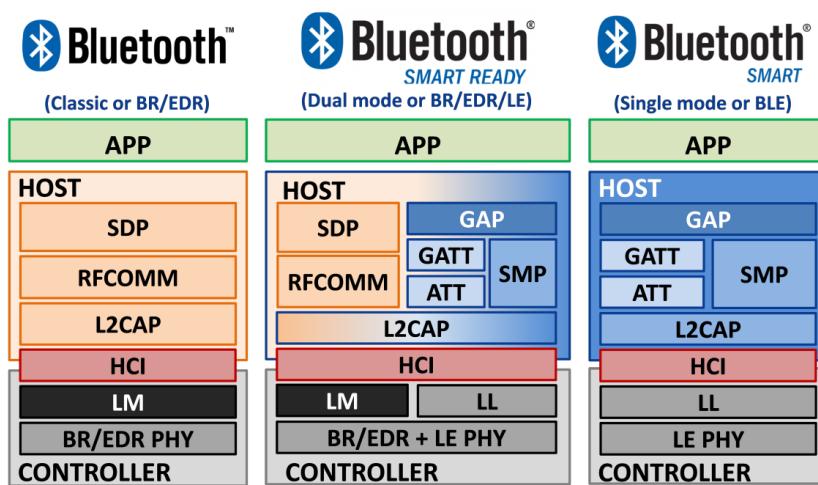
شکل ۱۲-۴ به طور گرافیکی سرعت و برد ارتباط بی‌سیم BLE را به تصویر می‌کشد.



شکل (۴-۱۲) سرعت و برد ارتباط بی‌سیم

### ۳-۱۲- پشته پروتکل BLE

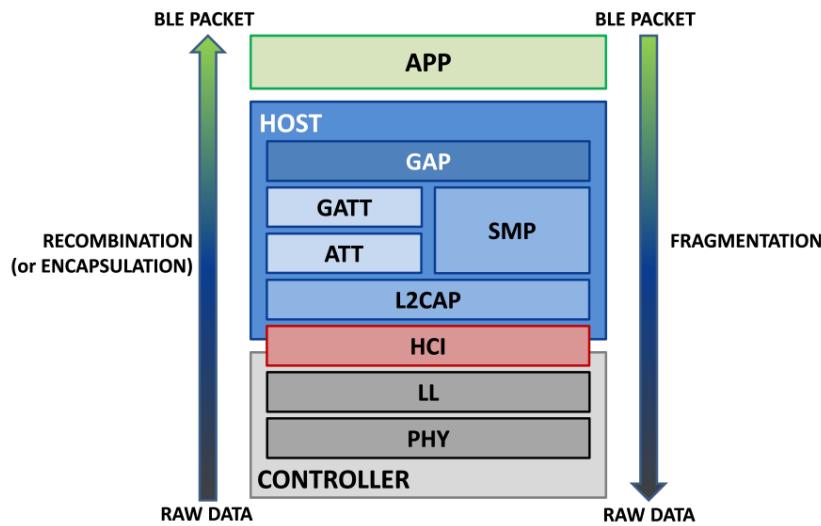
پروتکل بلوتوث کم‌صرف (BLE) از یک ساختار لایه‌ای تشکیل شده است که شامل سه بلوک اصلی است. در شکل ۱۲-۵ ساختار کلی پروتکل هر سه نوع بلوتوث<sup>۱</sup>, BR/EDR/LE, BR/EDR/LE, و LE نشان داده شده است.



شکل (۱۲-۵) پیکربندی بین نسخه‌های بلوتوث و انواع دستگاهها [۱]

<sup>۱</sup> Basic Rate/Enhanced Data Rate (BR/EDR)

پشته پروتکل برای BLE به شرح زیر (شکل ۶-۱۲) است:



شکل (۶-۱۲) [۱] پشته پروتکل BLE

- Application (App) بالاترین لایه پشته است و رابط مستقیم با کاربر را تشکیل می‌دهد. این لایه پروفایل‌هایی را تعریف می‌کند که به برنامه‌های مختلف اجزا می‌دهد تا از قابلیت‌های مشترک استفاده کنند و با هم تعامل داشته باشند. این پروفایل‌های کاربردی توسط گروه ویژه بلوتوث (Bluetooth SIG) مشخص شده‌اند و تعامل پذیری بین دستگاه‌های سازندگان مختلف را تشویق می‌کنند. مشخصات بلوتوث همچنین تعریف پروفایل‌های اختصاصی فروشندۀ را برای موارد استفاده‌ای که توسط پروفایل‌های تعریف شده توسط SIG پوشش داده نشده‌اند، امکان‌پذیر می‌سازد.

- Host شامل لایه‌های زیر است:
  - پروفایل دسترسی عمومی (GAP)
  - پروفایل ویژگی عمومی (GATT)
  - پروتکل واپایش رابط منطقی و تطبیق (L2CAP)
  - پروتکل ویژگی (ATT)
  - پروتکل مدیر امنیتی (SMP)
  - رابط واپایش میزبان (HCI)، سمت میزبان

- واپایش کننده در لایه‌های زیر ساختار یافته است:

- رابط واپایش میزبان (HCI)، سمت واپایش کننده

- لایه رابط (LL)

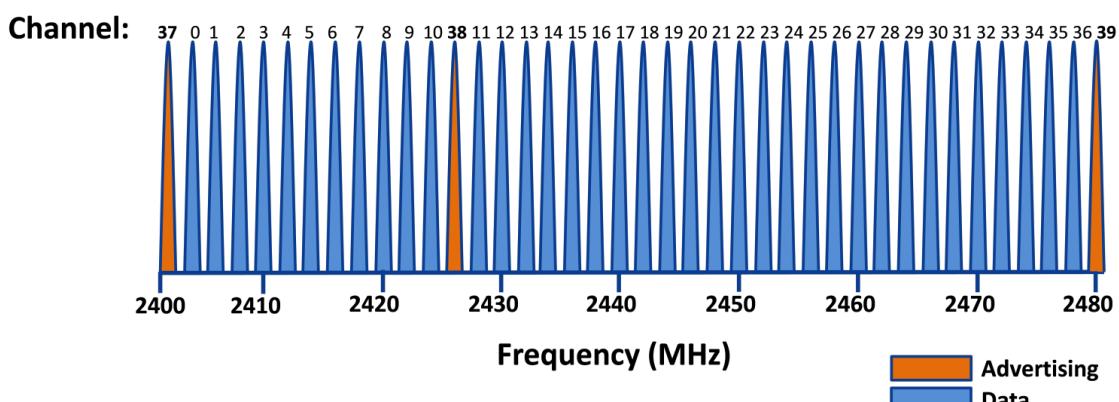
- لایه فیزیکی (PHY)

در بخش‌های بعدی، تمام لایه‌های پشته پروتکل به طور مفصل شرح داده خواهند شد.

## Physical Layer - ۴-۱۲

فنّاوری BLE برای کار در باند صنعتی، علمی و پزشکی (ISM<sup>۱</sup>) در محدوده ۲.۴ تا ۲.۵ گیگاهرتز، مشابه با Wi-Fi و BR/EDR طراحی شده است. به طور خاص، باند رادیویی BLE از ۲.۴۰۰۰ گیگاهرتز تا ۲.۴۸۳۵ گیگاهرتز گسترش می‌یابد و به ۴۰ کanal تقسیم می‌شود، همانطور که در شکل ۷-۱۲ نشان داده شده است. بسامد مرکزی این کanalها  $k = 2 + 2^{40} \times k$  مگاهرتز است، که در آن  $k = 0, 1, \dots, 39$ . سه تا از این کanalها (۳۷، ۳۸ و ۳۹) برای تبلیغات رزرو شده‌اند، در حالی که ۳۷ کanal دیگر برای تبادل بسته‌های داده در ارتباطات استفاده می‌شوند (در بخش Connections بیشتر بخوانید). بسته‌های تبلیغاتی (در بخش Broadcasting بیشتر بخوانید).

برای جلوگیری از تداخل و محوشدن با سایر ارتباطات بی‌سیم در همان باند رادیویی، BLE از جهش فرکانسی تطبیقی<sup>۲</sup> استفاده می‌کند که یک راهبرد است که به صورت شبه‌تصادفی کanal ارتباطی مورداستفاده توسط دو مشخصه اصلی رابط را تعریف می‌کند.



شکل (۷-۱۲) کanal‌های فرکانسی [۱] BLE

<sup>۱</sup> Industrial, Scientific and Medical (ISM)

<sup>۲</sup> Adaptive Frequency Hopping (AFH)

کارکرد لایه فیزیکی (PHY) در سرعت ۱ مگابیت بر ثانیه (LE 1M PHY) اجباری است؛ در این حالت، هر بیت ارسال شده با یک نماد واحد مطابقت دارد (انتقال بدون کدگذاری). با استفاده از کدگذاری تصحیح خطأ، می‌توان بیت‌های بیشتری را فقط با یک نماد مرتبط کرد، که به معنای نرخ بیت ۵۰۰ کیلوبیت بر ثانیه و ۱۲۵ کیلوبیت بر ثانیه است، زمانی که طرح کدگذاری به ترتیب از دو یا هشت نماد در هر بیت استفاده می‌کند. یک نرخ داده رادیویی اختیاری که PHY از آن پشتیبانی می‌کند ۲ مگابیت بر ثانیه (LE 2M PHY) است، اما در این حالت، فقط با داده‌های بدون کدگذاری کار می‌کند.

PHY BLE همچنین محدودیت‌هایی را برای توان انتقال رادیویی تعریف می‌کند که بین حداقل ۱۰۰ میلی‌وات (۲۰-دسی‌بل میلی‌وات) و حداکثر ۱۰۰ میلی‌وات (۱۰+دسی‌بل میلی‌وات) است. به منظور بهینه‌سازی مصرف انرژی، همراه با کاهش تداخل‌ها و افزایش برد سیگنال، می‌توان واپايش توان خروجی دستگاه را به صورت محلی تغییر داد. توان انتقال ویژگی اصلی مفیدی برای مدل‌سازی حداکثر برد انتقال BLE است.

## Link Layer - ۵-۱۲

LL بخشی از پشته است که مستقیماً با PHY ارتباط برقرار می‌کند؛ در واقع، از ترکیبی از یک بخش سخت‌افزاری (HW<sup>۱</sup>) و یک بخش نرم‌افزاری (SW<sup>۲</sup>) تشکیل شده است. LL نوع ارتباطاتی را که می‌توان بین دستگاه‌های BLE از طریق مدیریت وضعیت رابط رادیو (link state of the radio) ایجاد کرد، مطابق شکل ۸-۱۲ تعریف می‌کند. LL همچنین نقش‌های مختلفی را که یک دستگاه می‌تواند ایفا کند، تعریف می‌کند، یعنی مستر<sup>۳</sup>، اسلیو<sup>۴</sup>، تبلیغ‌کننده<sup>۵</sup> و اسکنر<sup>۶</sup>.

معمولًاً LL توسط فروشنده‌گان سیلیکون در سخت‌افزار پیاده‌سازی می‌شود تا از بارگذاری بیش از حد واحد پردازش مرکزی (CPU<sup>۷</sup>) که مسئول مدیریت تمام لایه‌های نرم‌افزاری پشته است، جلوگیری کند. قابلیت‌های آن به راحتی خودکار می‌شوند، اما از نظر محاسباتی گران هستند.

<sup>۱</sup> Hardware (HW)

<sup>۲</sup> Software (SW)

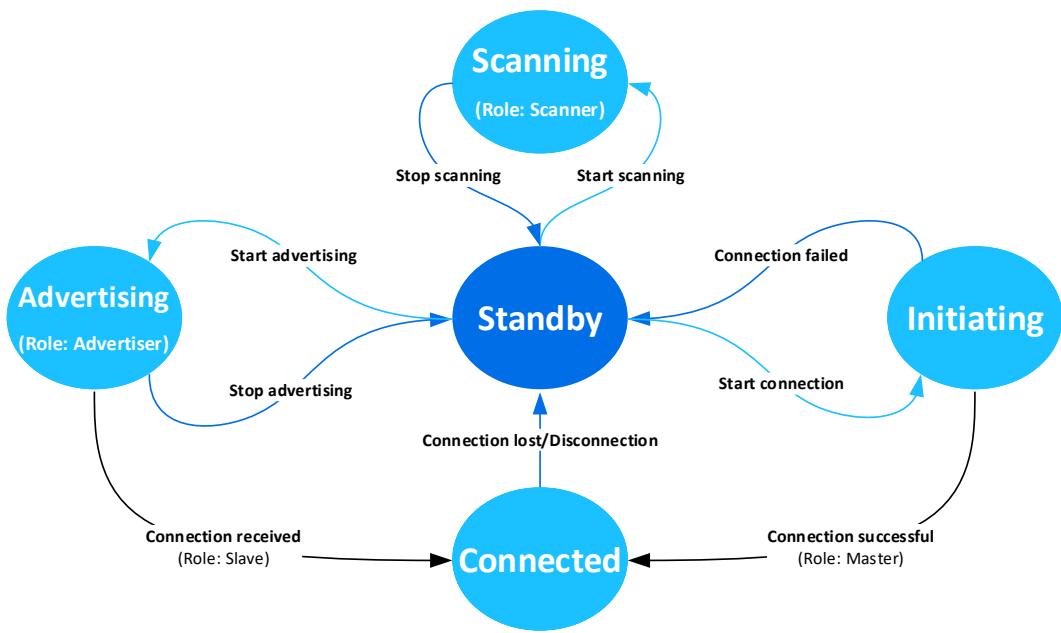
<sup>۳</sup> Master (Central)

<sup>۴</sup> Slave (Peripheral)

<sup>۵</sup> Advertiser (Broadcaster)

<sup>۶</sup> Scanner (Observer)

<sup>۷</sup> Central Process Unit (CPU)



شکل (۸-۱۲) ماشین حالت لایه لینک و انتقال حالت

## Host Controller Interface - ۶-۱۲

HCI یک پروتکل استاندارد است که مسئولیت ارتباط بین واپاپیش کننده که پایین‌ترین بخش پروتکل است، و میزبان، یعنی هسته پشتی پروتکل BLE که مدیریت ارتباط بین سخت‌افزار و برنامه کاربردی کاربر را بر عهده دارد، را بر عهده دارد؛ بنابراین، نقش آن تعریف مجموعه‌ای از دستورات و رویدادها برای ترجمه داده‌های خام به بسته‌های داده برای ارسال آن‌ها از طریق درگاه زنجیره به لایه میزبان و بالعکس است. این کار به دلیل مدولار بودن پروتکل ضروری بوده است و به همین دلیل، واپاپیش کننده، میزبان و برنامه کاربردی را در یک بسته واحد قرار نمی‌دهد.

## Logical Link Control and Adaptation Protocol - ۷-۱۲

L2CAP پروتکلی مشترک با BR/EDR است که به عنوان یک چندگانه‌ساز عمل می‌کند؛ این پروتکل داده‌ها را از لایه‌های پایین‌تر (LL برای BLE و LM برای BR/EDR) دریافت کرده و آن‌ها را مطابق با لایه‌های بالاتر (ATT و SMP برای BR/EDR و RFCOMM برای BLE) در قالب بسته استاندارد

بسته‌بندی می‌کند، و بالعکس؛ این فرآیندها به ترتیب بازسازی (کپسوله‌سازی)<sup>۱</sup> و قطعه‌سازی<sup>۲</sup> نامیده می‌شوند، همان‌طور که در شکل ۸-۱۲ نشان داده شده است.

## Security Manager Protocol - ۸-۱۲

روش‌های ارتباط امن را تعریف و ارائه می‌کند که در حوصله‌ی بحث نیست و از آن گذر می‌شود.

## Attribute Protocol - ۹-۱۲

پروتکل ویژگی (ATT) نقش‌های "Client" و "Server" را تعریف می‌کند. Client دستگاهی است که درخواست دریافت داده از Server را دارد، و Server داده‌ها را برای Client می‌فرستد. این نقش‌ها معمولاً با مستر و اسلیو در لایه Link Layer مطابقت دارند، اما یک دستگاه می‌تواند همزمان هم Client و Server باشد، بدون توجه به اینکه مستر یا اسلیو است.

داده‌ها را در قالب "ویژگی‌ها" سازماندهی می‌کند. هر ویژگی شامل یک هندل<sup>۳</sup> (شناسه داخلی)، یک شناسه منحصر به فرد جهانی<sup>۴</sup>، مجموعه‌ای از مجوزها، و یک مقدار<sup>۵</sup> است. پروتکل ATT در پروتکل GATT قرار دارد که از نقش‌های تعریف شده در ATT برای ایجاد ارتباطات استفاده می‌کند.

## Generic Attribute Profile - ۱۰-۱۲

ATT دو نقش در یک اتصال تعریف می‌کند، Client و Server که با آن‌هایی که در پروتکل GATT توصیف شده‌اند مطابقت دارد. گروه ویژه بلوتوث (Bluetooth SIG) برخی از خدمات و مشخصه‌های استاندارد را نشان داده شده توسط یک قالب نشانی ۱۶ بیتی تعریف می‌کند، اما قدرت BLE این است که به سازندگان اجازه می‌دهد تا خدمات خود را با استفاده از یک 128 UUID بیتی برای تطبیق این فناوری با برنامه‌های جدید و اصلی تعریف کنند.

در طول ایجاد اتصال، سرور خدمات و مشخصه‌های خود را در معرض کلاینت قرار می‌دهد تا نحوه ساختاردهی اتصال را تعریف کند. ساختار منطقی پروفایل سرور GATT که شامل چندین سرویس و

<sup>1</sup> Recombination (or Encapsulation)

<sup>2</sup> Fragmentation

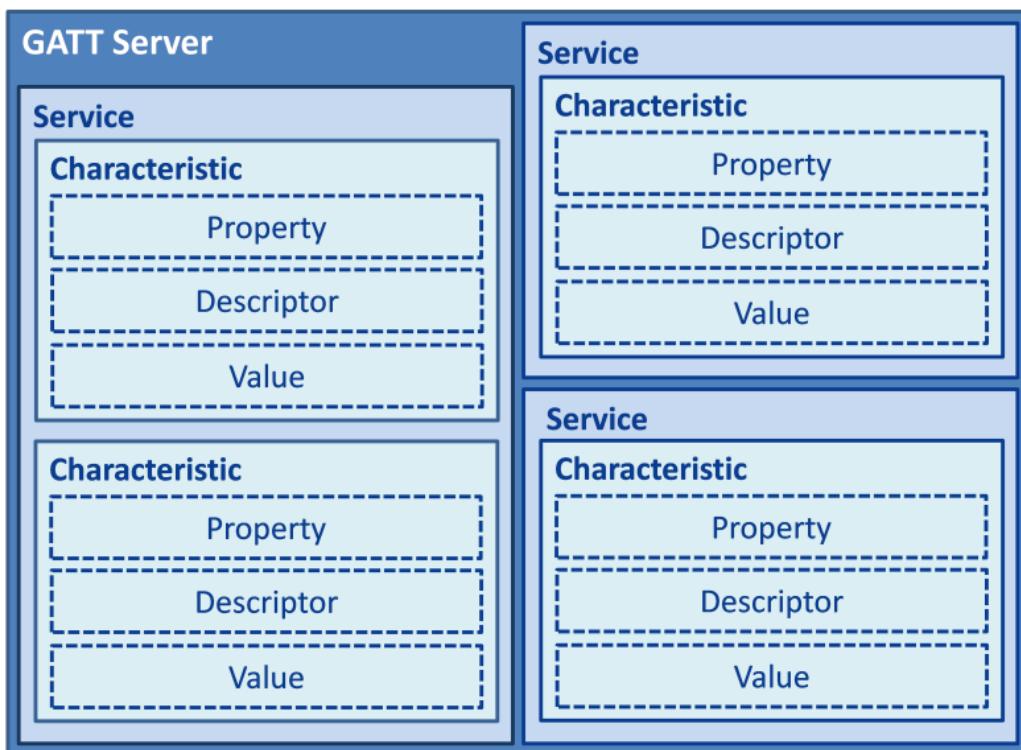
<sup>3</sup> Handle

<sup>4</sup> Universally Unique Identifier (UUID)

<sup>5</sup> Value

مشخصه است، در شکل ۹-۱۲ نشان داده شده است.

قبل از برقراری اتصال، سرور GATT، خدمات<sup>۱</sup> و مشخصه‌های<sup>۲</sup> خود را نمایش می‌دهد. همان‌طور که در این شکل نشان داده شده است، سرویس‌ها و مشخصه‌ها برای تشکیل یک ساختار داده منطقی تعریف شده‌اند. علاوه بر این، هر مشخصه ویژگی‌های<sup>۳</sup> خود، یک توصیف‌گر<sup>۴</sup> که تعریف می‌کند چه کاری انجام می‌دهد، و مقدار داده<sup>۵</sup> را نشان می‌دهد.



شکل (۹-۱۲) سلسه مراتب داده [۱] GATT

یک سرویس اساساً یک ظرف است که از نظر مفهومی ویژگی‌های مرتبط را گروه‌بندی می‌کند، در حالی که مشخصه‌ها ویژگی‌هایی هستند که در یک سرویس گنجانده شده‌اند و هر کدام برای برقراری ارتباط یک نوع داده خاص استفاده می‌شوند. این پارادایم مبتنی بر سرویس یک انتزاع بیشتر در بالای معماری کلاینت-سرور است که در آن سرویس‌ها رفتار تعریف شده‌ای دارند که همیشه همان نوع پاسخ را می‌دهند.

<sup>1</sup> Services

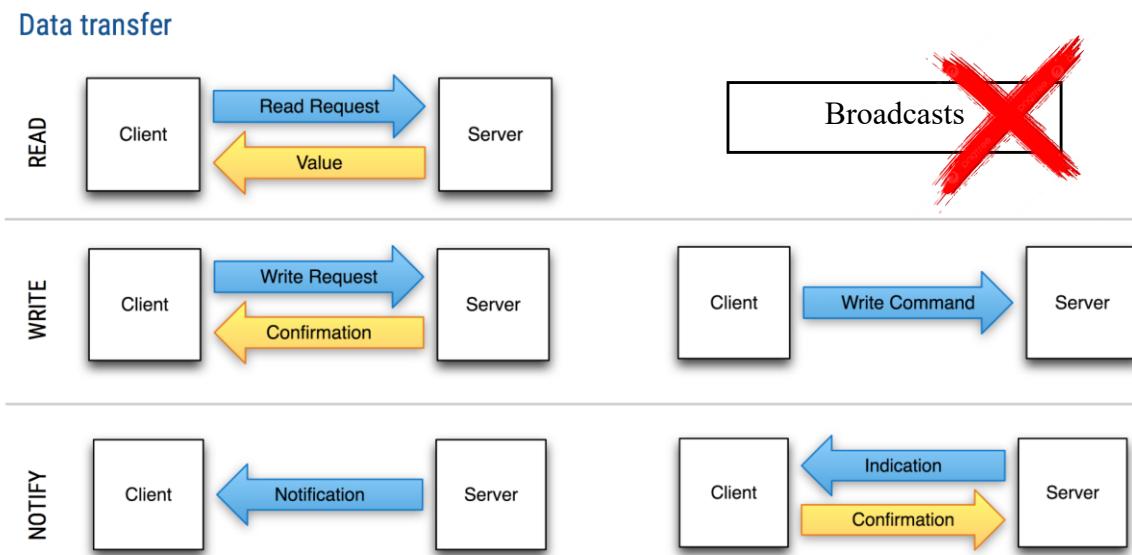
<sup>2</sup> Characteristics

<sup>3</sup> Property

<sup>4</sup> Descriptor

<sup>5</sup> Value

مشخصه‌ها شامل مقدار داده، یک توصیف‌گر هستند که اطلاعات اضافی درباره مشخصه، مقدار آن و برخی ویژگی‌ها را می‌دهد. این ویژگی‌ها به کلاینت نشان می‌دهند که چه عملیاتی را می‌توان روی مشخصه انجام داد؛ همان‌طور که در شکل ۱۰-۱۲ مشخص است، ویژگی‌های پرکاربرد (روش‌های انتقال داده بین Client و Server) عبارت‌اند از:



شکل (۱۰-۱۲) روش‌های تبادل داده

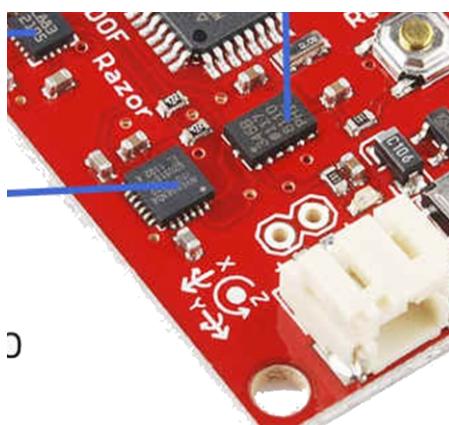
#### ❖ از طریق Broadcasts (GAP):

در این روش، داده‌ها به صورت عمومی و بدون نیاز به ایجاد اتصال ارسال می‌شوند. این روش برای کاربردهایی مناسب است که داده‌ها باید به چندین دستگاه به طور همزمان ارسال شوند، مانند اطلاعات تبلیغاتی یا پیام‌های عمومی. اگرچه از این روش، نباید چندان به عنوان روش انتقال داده یاد کرد!

برای روشن تر کردن ساختار سلسله مراتبی پروفایل سرور GATT، می توانیم از مثال زیر استفاده کنیم. فرض کنید نیاز داریم داده ها را از یک واحد اندازه گیری اینرسی مغناطیسی (M-IMU<sup>۱</sup>) که شامل سه حسگر مختلف (شتاب سنج، ژیروسکوپ و مغناطیس سنج) است (شکل ۱۱-۱۲) و نقش مستر را دارد، به یک دستگاه موبایل مانند گوشی هوشمند یا تبلت که نقش اسليو را دارد، ارسال کنیم. همچنین فرض کنیم که برد دارای باتری داخلی برای قابلیت حمل است و یک سنسور دما دارد. در واقع، دانستن این داده ها مهم است زیرا خروجی M-IMU تحت تأثیر تغییرات دما، مثلًا به دلیل گرم شدن بیش از حد باتری، قرار دارد. در این مثال، ساخت پروفایل با دو سرویس مختلف می تواند مفید باشد، همانطور که در شکل ۱۲-۱۲ نشان داده شده است.

**سرویس M-IMU:** این سرویس همه داده های مربوط به M-IMU را مدیریت می کند و به چهار مشخصه مختلف تقسیم می شود. سه مشخصه قابل اطلاع رسانی وظیفه ارسال داده از حسگر به دستگاه مرکزی را بر عهده دارند؛ یکی برای داده های شتاب سنج<sup>۲</sup>، یکی برای ژیروسکوپ<sup>۳</sup> و دیگری برای مغناطیس سنج<sup>۴</sup>. آخرین مشخصه قابل نوشتن است تا گره<sup>۵</sup> مرکزی بتواند برخی ویژگی های M-IMU را تغییر دهد، مانند بسامد نمونه برداری یا تعداد حسگر های در حال انتقال.

**سرویس وضعیت باتری و دما:** این سرویس دارای دو مشخصه قابل اطلاع رسانی است که برای ارسال داده های مربوط به شارژ باقی مانده باتری و دما استفاده می شود. برای حفظ انرژی، می توان داده ها را با نرخ پایین تری نسبت به سرویس قبلی ارسال کرد. این یکی دیگر از دلایل اهمیت مدیریت صحیح ساختار منطقی GATT است؛ به این ترتیب، می توان انتقال داده را بسته به کاربرد خاص جدا کرد.



شکل ۱۱-۱۲ (M-IMU)

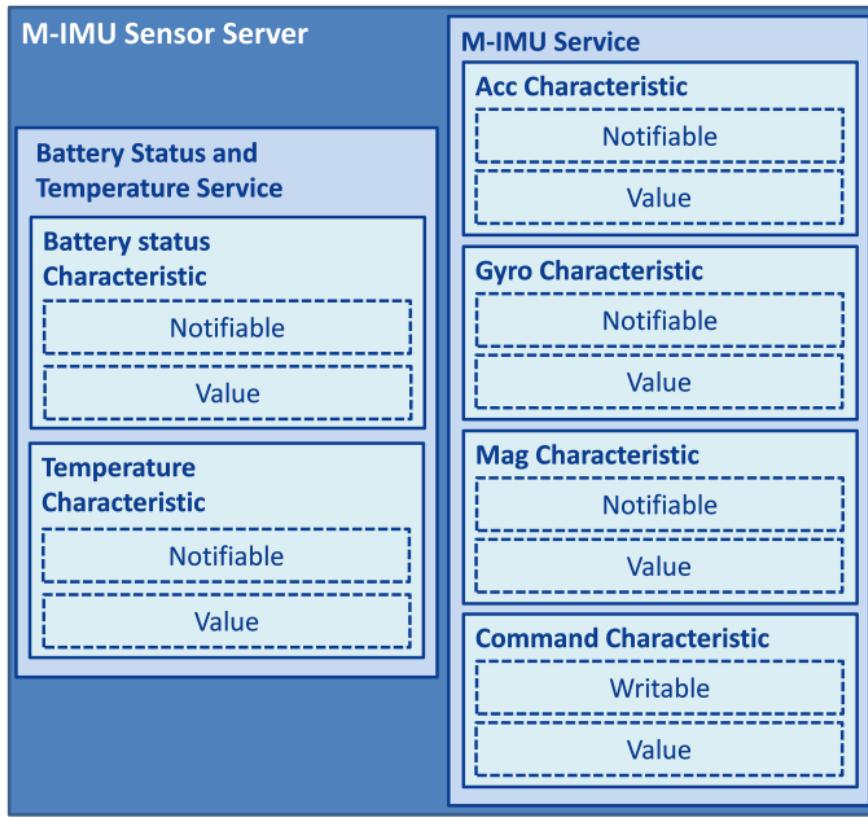
<sup>۱</sup> Magneto-Inertial Measurement Unit (M-IMU)

<sup>۲</sup> Accelerometer

<sup>۳</sup> Gyroscope

<sup>۴</sup> Magnetometer

<sup>۵</sup> Node



شکل (۱۲-۱۲) سلسله مراتب داده GATT برای مثال [۱]

## Generic Access Profile - ۱۱-۱۲

در پشتی GAP در بالاترین سطح هسته آن قرار دارد، این پروفایل نقش‌های دستگاه، حالتهای رویه‌ها را مشخص می‌کند و علاوه بر این، برقراری اتصال و امنیت را مدیریت می‌کند. این لایه مستقیماً با لایه کاربردی و در نتیجه با کاربر ارتباط برقرار می‌کند که می‌تواند تمام عامل‌های موردنیاز شبکه را تعریف کند. علاوه بر این، پیوند بین کاربر و تمام پروتکل‌های پشتی را فراهم می‌کند؛ در واقع، تمام پروتکل‌های پایین‌تر را پیاده‌سازی و واپايش می‌کند.

❖ کجاها مشارکت دارد؟

- لایه فیزیکی
- لایه لینک
- رویدادهای پخش

- انواع اسکن
- روش کنترل لایه لینک

### ❖ نقش‌های مختلف GAP چیست؟

در بلوتوث، دستگاه‌ها می‌توانند چهار نقش اساسی داشته باشند:

- پخش‌کننده (Broadcaster): این دستگاه فقط داده‌ها را تبلیغ می‌کند و نمی‌تواند با دستگاه‌های دیگر جفت شود یا به آن‌ها متصل شود. به عبارت دیگر، فقط اطلاعاتی را ارسال می‌کند و قادر به دریافت یا ایجاد ارتباط پایدار نیست.
- مشاهده‌گر (Observer): این دستگاه داده‌های تبلیغ شده را اسکن می‌کند و می‌تواند آن‌ها را دریافت کند، اما قادر به ایجاد اتصال نیست. به عنوان مثال، دستگاه‌هایی که به دنبال دستگاه‌های بلوتوث اطراف خود می‌گردند، در این نقش قرار می‌گیرند.
- مرکزی (Central): این دستگاه نقش اصلی را در یک اتصال بلوتوث ایفا می‌کند. دستگاه مرکزی داده‌های تبلیغ شده را اسکن کرده و درخواست اتصال به دستگاه‌های دیگر (پریفرال‌ها) را آغاز می‌کند. یک دستگاه مرکزی می‌تواند به چندین دستگاه پریفرال متصل شود.
- پریفرال (Peripheral): این دستگاه نقش فرعی را دارد و معمولاً به دستگاه مرکزی متصل می‌شود. دستگاه پریفرال داده‌های خود را تبلیغ می‌کند و می‌تواند به درخواست اتصال دستگاه مرکزی پاسخ دهد.

### ❖ سرویس‌های GAP

- Device name: نامی که به دستگاه بلوتوث اختصاص داده می‌شود و برای شناسایی آن توسط سایر دستگاه‌ها استفاده می‌شود.
- Appearance Characteristics: اطلاعاتی در مورد نوع دستگاه (مثلاً ساعت، هدفون، یا گوشی همراه) و ویژگی‌های ظاهری آن (مثلاً رنگ، اندازه) را ارائه می‌دهد.

○ **PPCP<sup>1</sup>**: به مجموعه پارامترهایی گفته می‌شود که نحوه برقراری و حفظ ارتباط بین یک دستگاه مرکزی (Master) و یک دستگاه فرعی (Slave) در بلوتوث را تعیین می‌کند. این پارامترها به دستگاه فرعی اجازه می‌دهند تا کنترل بیشتری بر روی فرکانس ارتباط و زمان‌بندی تبادل داده‌ها داشته باشد.

○ **Central device address resolution**: فرایندی که به دستگاه‌های بلوتوث اجازه می‌دهد تا آدرس دستگاه‌های مرکزی را حل کرده و به صورت منحصر به فرد شناسایی کند.

○ **Resolvable private address**: یک آدرس بلوتوث موقت و تصادفی است که برای حفظ حریم خصوصی استفاده می‌شود و می‌تواند به آدرس عمومی دستگاه مرتبط شود.

❖ **حالتهای امنیتی GAP** به روش‌های مختلفی که دستگاه‌های بلوتوث می‌توانند برای برقراری ارتباط امن استفاده کنند، اشاره دارد. این حالت‌ها به دو دسته کلی تقسیم می‌شوند: **Safe Mode 2** و **Mode 1**.

○ **Safe Mode 1**

- **بدون امنیت<sup>2</sup>**: هیچگونه رمزگاری یا احراز هویتی انجام نمی‌شود. این حالت ساده‌ترین حالت است اما از نظر امنیتی بسیار ضعیف است و نباید در محیط‌های حساس استفاده شود.

- **جفت‌سازی بدون احراز هویت با رمزگاری<sup>3</sup>**: دستگاه‌ها بدون نیاز به احراز هویت می‌توانند به یکدیگر متصل شوند، اما داده‌های مبادله شده رمزگاری می‌شوند. این حالت امنیت نسبت به حالت قبلی بالاتر است.

<sup>1</sup> Peripheral Preferred Connection Parameters (PPCP)

<sup>2</sup> No Security

<sup>3</sup> Unauthenticated pairing with encryption

▪ جفت‌سازی با احراز هویت با رمزنگاری<sup>۱</sup>: قبل از برقراری اتصال، دستگاه‌ها باید هویت یکدیگر را تأیید کنند و سپس داده‌ها با رمزنگاری محافظت می‌شوند. این حالت امنیت بالاتری نسبت به دو حالت قبلی دارد.

▪ جفت‌سازی با رمزنگاری LE Secure Connection<sup>۲</sup>: این حالت از امن‌ترین حالت‌های جفت‌سازی در بلوتوث است و از الگوریتم‌های رمزنگاری قوی برای محافظت از داده‌ها استفاده می‌کند.

#### Safe Mode 2 ○

▪ جفت‌سازی بدون احراز هویت با امضای داده<sup>۳</sup>: مشابه حالت دوم در Safe Mode 1 است، با این تفاوت که از امضای دیجیتال برای تأیید اصالت داده‌ها استفاده می‌شود.

▪ جفت‌سازی با احراز هویت با امضای داده<sup>۴</sup>: مشابه حالت سوم در Safe Mode 1 است، با این تفاوت که از امضای دیجیتال برای تأیید اصالت داده‌ها استفاده می‌شود.

درخواست به روزرسانی پارامترهای اتصال به معنای درخواستی است که توسط یک دستگاه (ممکن‌باشد) دستگاه فرعی یا Slave (به دستگاه اصلی Host یا Master) ارسال می‌شود تا پارامترهای کنونی ارتباط بین این دو دستگاه تغییر کند. این تغییرات می‌توانند برای بهبود کارایی سیستم، تطبیق با شرایط جدید یا افزایش طول عمر باتری انجام شوند.

در ابتدا اطلاعات به سرعت مبادله می‌شود: این بدان معنی است که در ابتدای برقراری اتصال، سرعت تبادل اطلاعات بین دستگاه‌ها بالا است تا اطلاعات ضروری به سرعت رد و بدل شود.

دستگاه فرعی می‌تواند درخواست تغییر پارامترهای اتصال را از طریق درخواست به روزرسانی

<sup>1</sup> Authenticated pairing with encryption

<sup>2</sup> With encryption LE Secure Connection pairing

<sup>3</sup> Unauthenticated pairing with data signature

<sup>4</sup> Authenticated pairing with data signature

پارامترهای اتصال بدهد: این به دستگاه فرعی اجازه می‌دهد تا در صورت نیاز، پارامترهایی مانند سرعت انتقال داده، فاصله زمانی بین دو انتقال داده و سایر پارامترهای مرتبط را تغییر دهد.

این کار باعث بهبود کارایی سیستم می‌شود: با تغییر پارامترهای اتصال، می‌توان بهینه سازی‌هایی در مصرف انرژی، کاهش تداخل و افزایش سرعت انتقال داده‌ها انجام داد که در نهایت به بهبود عملکرد کلی سیستم منجر می‌شود.

درخواست به روزرسانی پارامترهای اتصال به دستگاه اصلی یا میزبان بستگی دارد. این درخواست همیشه با یک محدوده ارائه می‌شود تا امکان بیشتری برای تغییر پارامترهای اتصال وجود داشته باشد: این بدان معنی است که دستگاه فرعی نمی‌تواند هر تغییری را در پارامترها درخواست کند و محدود به محدوده‌ای است که توسط دستگاه اصلی تعیین شده است. این محدوده به منظور جلوگیری از ایجاد اختلال در سیستم و حفظ پایداری آن تعریف می‌شود.

## BLE Communication - ۱۲-۱۲

BLE از طریق دو حالت اصلی انجام می‌شود: پخش (*Broadcasting*) و اتصالات (*Connections*).

### ❖ پخش (*Broadcasting*) ❖

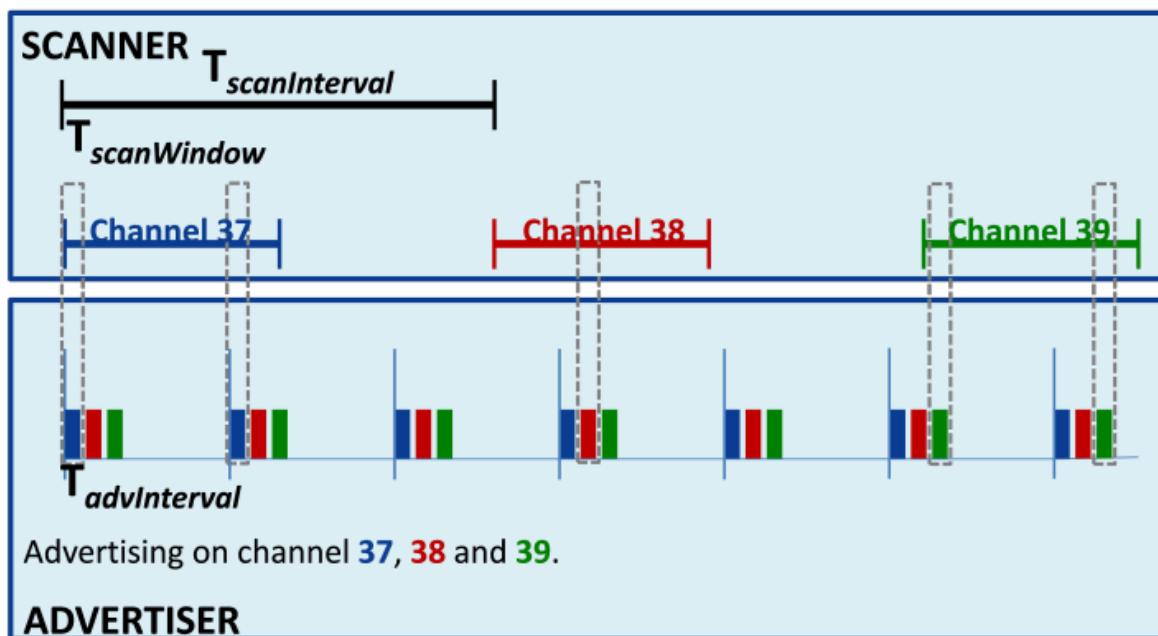
پخش بدون اتصال: در این حالت، یک دستگاه (پخش‌کننده) بسته‌های تبلیغاتی را به دستگاه‌های دیگر در محدوده خود ارسال می‌کند. این روش برای انتقال سریع داده به چندین دستگاه مناسب است، اما برای داده‌های حساس مناسب نیست، زیرا هیچ مکانیزم امنیتی یا حفظ حریم خصوصی در آن وجود ندارد.

#### ▪ نقش‌ها در پخش:

*Broadcaster (Advertiser)* ✓: بسته‌های تبلیغاتی را به طور دوره‌ای ارسال می‌کند.  
*Observer (Scanner)* ✓: به طور دوره‌ای به دنبال دریافت این بسته‌ها از پخش‌کننده‌ها می‌گردد.

#### ▪ عامل‌های مهم (شکل ۱۲-۱۳):

- ✓ فاصله تبلیغاتی (*Advertising Interval*): نرخ ارسال بسته‌های تبلیغاتی توسط پخش‌کننده.
- ✓ فاصله اسکن (*Scan Interval*): نرخ فعال شدن رادیو اسکنر.
- ✓ پنجره اسکن (*Scan Window*): مدت زمان فعال بودن رادیو<sup>۱</sup> برای اسکن در هر فاصله اسکن.



شکل (۱۲-۱۳) پارامترهای [۱] Broadcast

- به طور معمول، یک بسته تبلیغاتی شامل موارد زیر است:
- نام دستگاه<sup>۲</sup>: نامی که به دستگاه داده شده است و برای شناسایی آن استفاده می‌شود.
- پرچم‌ها<sup>۳</sup>: مجموعه‌ای از بیت‌ها هستند که اطلاعات اضافی در مورد دستگاه را ارائه می‌دهند. مثلًاً اینکه: فلگ‌ها می‌توانند دو دسته‌بندی پشتیبانی را نمایندگی کنند.
- قابلیت کشف دستگاه \*
- پشتیبانی از دستگاه‌های قدیمی (Legacy) \*

دستگاه تبلیغاتی می‌تواند نشان دهد که آیا برای مدت زمان محدودی تبلیغ خواهد

<sup>1</sup> Advertising Duration

<sup>2</sup> Device name

<sup>3</sup> Flags

کرد یا خیر.

- همچنین می‌تواند نشان دهد که آیا از BR/EDR پشتیبانی می‌کند یا فقط LE.
- ظاهر<sup>۱</sup>: اطلاعاتی در مورد نوع دستگاه، مانند ساعت، هدفون یا سنسور، ارائه می‌دهد.
  - سرویس اولیه<sup>۲</sup>: یک شناسه منحصر به فرد است که سرویس اصلی ارائه شده توسط دستگاه را مشخص می‌کند. سرویس‌ها مجموعه‌ای از ویژگی‌ها هستند که دستگاه می‌تواند آن‌ها را ارائه دهد، مانند سنسور دما، کنترل نور یا پخش موسیقی. دو نوع ۱۲۸ بیتی و ۱۶ بیتی دارد که ۱۶ بیتی صرفاً خلاصه است و این کار بهره‌وری انتقال را افزایش می‌دهد زیرا نیازی به ارسال ۱۶ بایت در هر بار نیست.

#### » حالت‌های کاری تبلیغات:

- BLE\_ADV\_MODE\_IDLE: این حالت به معنای حالت بیکار یا غیرفعال است. در این حالت دستگاه هیچ بسته تبلیغاتی ارسال نمی‌کند.
- BLE\_ADV\_MODE\_DIRECTED\_HIGH\_DUTY: در این حالت، دستگاه به صورت مستقیم و با فرکانس بالا به یک دستگاه خاص بسته تبلیغاتی ارسال می‌کند. این حالت معمولاً برای ایجاد ارتباط اولیه با یک دستگاه خاص استفاده می‌شود.
- BLE\_ADV\_MODE\_DIRECTED: این حالت نیز برای ارسال مستقیم بسته‌های تبلیغاتی به یک دستگاه خاص است، اما با فرکانس کمتری نسبت به حالت قبلی.
- BLE\_ADV\_MODE\_FAST: در این حالت، دستگاه به صورت مداوم و با سرعت بالا بسته‌های تبلیغاتی را ارسال می‌کند تا به سرعت توسط دستگاه‌های دیگر شناسایی شود.
- BLE\_ADV\_MODE\_SLOW: در این حالت، دستگاه به صورت دوره‌ای و با سرعت کم بسته‌های تبلیغاتی را ارسال می‌کند تا مصرف انرژی را کاهش دهد.

#### ❖ اتصالات (Connections):

اتصالات دائمی: ارتباط بین دو دستگاه که در آن بسته‌ها به صورت دائمی و دوره‌ای بین آنها مبادله می‌شود. این ارتباط خصوصی است و می‌تواند با تنظیمات امنیتی محافظت شود.

#### ▪ نقش‌ها در اتصالات:

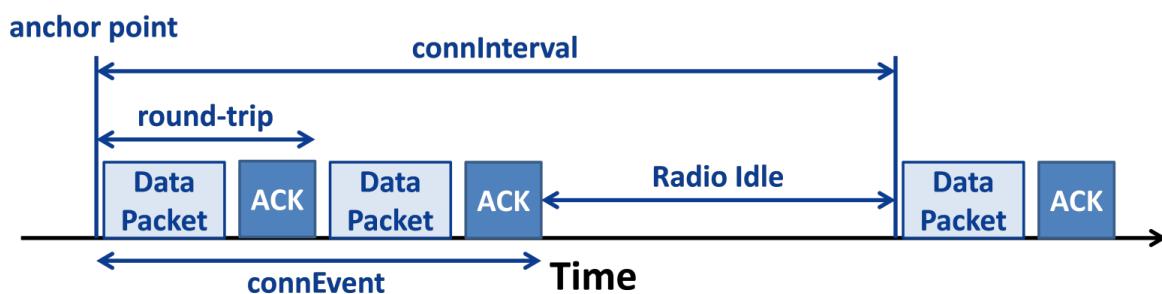
<sup>1</sup> Appearance

<sup>2</sup> Primary service UUID

اسکن می‌کند تا بسته‌های تبلیغاتی قابل اتصال را پیدا کند و ارتباط را آغاز می‌کند.

بسته‌های تبلیغاتی قابل اتصال ارسال می‌کند و درخواست‌های اتصال را از مستر می‌پذیرد.

عامل‌های دیگری نیز وجود دارند که برای توصیف اتصال بلوتوث مفید هستند (شکل ۱۴-۱۲) و می‌توان از آن‌ها برای تنظیم خود اتصال نیز استفاده کرد:



شکل (۱۴-۱۲) [۱] مثالی از ارتباط BLE

❖ فاصله اتصال (*connInterval*) زمان بین شروع دو رویداد اتصال متوالی است، به عبارت دیگر، مجموع رویداد اتصال و زمان بیکاری رادیو است. فاصله اتصال باید مضرب ۱.۲۵ میلی ثانیه در محدوده ۷.۵ میلی ثانیه تا ۴ ثانیه باشد.

○ *Minimum Connection Interval*: حداقل فاصله زمانی بین دو اتصال موفق. به عبارت دیگر، کوتاه‌ترین زمانی که دستگاه‌ها می‌توانند برای تبادل داده‌ها منتظر بمانند. این پارامتر بر روی پهنای باند و پاسخگویی سیستم تأثیر می‌گذارد. بین ۷.۵ میلی ثانیه تا ۴ ثانیه قابل تنظیم است.

○ *Maximum Connection Interval*: حداکثر فاصله زمانی بین دو اتصال موفق. این پارامتر به دستگاه‌ها اجازه می‌دهد تا در صورت نیاز، ارتباط را کنترل کند و در مصرف انرژی صرفه‌جویی نمایند. بین ۷.۵ میلی ثانیه تا ۴ ثانیه قابل تنظیم است.

❖ زمان نظارت بر اتصال (*connSupervisionTimeout*) حداکثر زمانی است که می‌تواند بدون دریافت دو بسته معتبر جریان یابد، قبل از اینکه اتصال قطع شود. زمان نظارت بر اتصال باید مضرب ۱۰ در محدوده ۱۰۰ میلی ثانیه تا ۳۲ ثانیه باشد. به طور خلاصه این عامل مدت زمانی

را مشخص می‌کند که دستگاه مرکزی منتظر پاسخ دستگاه فرعی می‌ماند. اگر در این مدت زمانی، دستگاه فرعی پاسخ ندهد، اتصال قطع می‌شود.

❖ تأخیر اسلیو اتصال (*connSlaveLatency*) تعداد رویدادهای اتصال است که می‌توان بدون خطر قطع اتصال از آن‌ها صرف‌نظر کرد. مقدار تأخیر اسلیو اتصال نباید باعث زمان نظارت بر اتصال شود و باید یک عدد صحیح در محدوده صفر تا  $(1 - \text{connSupervisionTimeout}/(\text{connInterval} \times 2))$  باشد. علاوه بر این، تأخیر اسلیو اتصال نباید کمتر از ۵۰۰ باشد و هنگامی که روی صفر تنظیم شود، دستگاه اسلیو باید در هر گوش دهد، بدون اینکه اتصال قطع شود. *anchor point*

#### ❖ ساختار بسته (*BLE Packet*)

همانطور که بالاتر توضیح داده شد، BLE دو نوع ارتباط (یعنی پخش و اتصال) را امکان‌پذیر می‌کند که به معنای دو نوع مختلف بسته است. این دو نوع بسته ساختاری مشترک دارند که در شکل ۱۲-۱۵ نشان داده شده است. این ساختار به چهار زیربخش اصلی تقسیم شده و به شرح زیر است:

- مقدمه (PRE<sup>۱</sup>): طول این بخش به نرخ داده رادیویی بستگی دارد و به ترتیب برای ارتباطات با استاندارد LE 1M PHY یا LE 2M PHY که در بخش ۴-۱۲ توضیح داده شد، یک یا دو بایت است. مقدمه توالی بسیار ساده‌ای از بیت‌هایی از آن برای تنظیم کنترل خودکار بهره و تعیین فرکانس متناسب با نرخ داده رادیویی استفاده می‌کند.
- آدرس دسترسی (AA<sup>۲</sup>): این بخش شامل چهار بایت بعدی است و ارتباط برقرار شده روی یک لینک فیزیکی را شناسایی می‌کند. از این آدرس برای حذف بسته‌هایی استفاده می‌شود که برای گیرنده‌های دیگری ارسال شده‌اند.
- واحد داده پروتکل (PDU<sup>۳</sup>): محدوده PDU بین دو تا ۲۵۷ بایت است و طول آن کاملاً به نوع ارتباط مورد استفاده بستگی دارد که جزئیات بیشتر آن در ادامه توضیح داده خواهد شد.
- کد تشخیص خطأ (CRC<sup>۴</sup>): این بخش شامل سه بایت است که خطاهای احتمالی را فقط در PDU بررسی می‌کند. خطاهای ممکن است در حین انتقال بسته ایجاد شده باشند.

<sup>1</sup> Preamble (PRE)

<sup>2</sup> Access Address (AA)

<sup>3</sup> Protocol Data Unit (PDU)

<sup>4</sup> Cyclic Redundancy Check (CRC)

یک تحلیل دقیق از تکنیک‌های تصحیح خطأ، با تمرکز خاص بر روی CRC در BLE، در [2] ارائه شده است.



شکل (۱۵-۱۲) ساختار بسته BLE [1]

کانال پخش PDU (که در شکل ۱۶-۱۲ نشان داده شده است) دارای یک هدر ۱۶ بیتی و یک بار مفید با اندازه متغیر از صفر تا ۲۵۵ بایت است. هدر شامل چهار بیت است که نوع PDU را نشان می‌دهد، زیرا چندین نوع بسته پخش وجود دارد.

(ADV\_IND): این کد نشان‌دهنده یک بسته تبلیغاتی است که برای اطلاع‌رسانی عمومی درباره وجود یک دستگاه استفاده می‌شود. هر دستگاهی که در حالت اسکن باشد می‌تواند این بسته‌ها را دریافت کند.

(ADV\_DIRECT\_IND): این کد نیز یک بسته تبلیغاتی است اما با این تفاوت که به یک دستگاه خاص ارسال می‌شود. این نوع تبلیغات برای ایجاد ارتباط مستقیم بین دو دستگاه استفاده می‌شود.

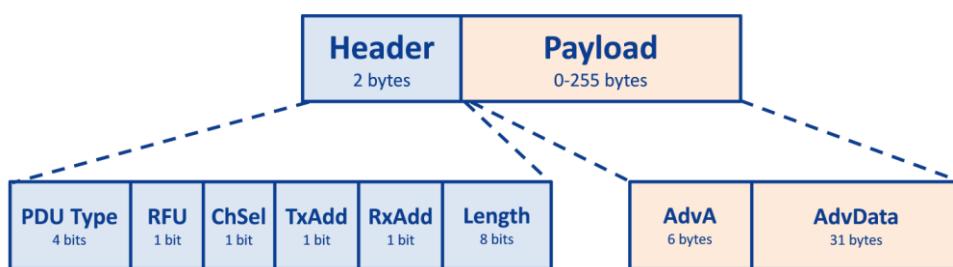
(ADV\_NONCONN\_IND): این کد برای تبلیغاتی استفاده می‌شود که قصد ایجاد اتصال پایدار ندارند. مثلاً برای انتقال اطلاعات کوتاه‌مدت یا ارسال نوتیفیکیشن‌ها.

(SCAN\_REQ): این کد نشان‌دهنده یک درخواست اسکن است. یک دستگاه با ارسال این درخواست از دستگاه‌های دیگر می‌خواهد که اطلاعات تبلیغاتی خود را ارسال کنند.

(SCAN\_RSP): این کد پاسخ به درخواست اسکن است. دستگاهی که درخواست اسکن دریافت کرده، با ارسال این بسته اطلاعات تبلیغاتی خود را در اختیار دستگاه درخواست‌کننده قرار می‌دهد.

(CONNECT\_REQ): این کد برای درخواست ایجاد یک اتصال بلوتوثی استفاده می‌شود. (ADV\_SCAN\_IND): این کد ترکیبی از تبلیغ و اسکن است. دستگاه همزمان با تبلیغ خود، درخواست اسکن نیز ارسال می‌کند تا دستگاه‌های دیگر بتوانند اطلاعات آن را دریافت کرده و در صورت نیاز به آن متصل شوند.

بیت بعدی برای استفاده در آینده (RFU<sup>1</sup>) رزرو شده است، در حالی که سه بیت بعدی به ترتیب برای انتخاب کanal (ChSel<sup>2</sup>)، آدرس فرستنده (TxAdd<sup>3</sup>) و آدرس گیرنده (RxAdd<sup>4</sup>) اختصاص داده شده‌اند که بسته به نوع PDU استفاده شده، بهره‌برداری می‌شوند. در نهایت، آخرین هشت بیت در هدر طول بار مفید را نشان می‌دهد. به جز برای برخی وظایف خاص، بسته داده پخش شده توسط تبلیغ‌کننده ارسال می‌شود و بار مفید شامل شش بایت از آدرس دستگاه تبلیغ‌کننده (AdvA<sup>5</sup>) است که حاوی آدرس دستگاه عمومی یا تصادفی تبلیغ‌کننده است) بسته به اینکه TxAdd برابر صفر یا یک باشد (و حداقل ۳۱ بایت داده تبلیغاتی (AdvData<sup>6</sup>) که همان بار مفید واقعی است. در صورتی که این بار مفید برای برنامه خاصی کافی نباشد، اسکنر می‌تواند درخواست یک بسته دیگر با حداقل ۳۱ بایت را بدهد؛ این فرآیند به عنوان اسکن فعال شناخته می‌شود و اگر استفاده شود، در نوع PDU اعلام می‌شود. توصیف و تحلیل دقیقی از اسکن غیرفعال و فعال و عملکرد آنها در [۳] پیشنهاد شده است.



شکل (۱۶-۱۲) ساختار PDU یک بسته تبلیغاتی [۱]

از سوی دیگر، بسته اتصال (شکل ۱۷-۱۲) از یک هدر دو بایتی تشکیل شده است که حاوی چندین پارامتر است: شناسه لینک منطقی (LLID) (۲ بیت)، شماره توالی بعدی مورد انتظار (NESN<sup>8</sup>) (۱ بیت)، شماره توالی (SN<sup>9</sup>) (۱ بیت)، داده بیشتر (MD<sup>10</sup>) (۱ بیت)، RFU (۳ بیت) و طول محموله (۸ بیت). به طور خاص، مقدار LLID نشان می‌دهد که آیا PDU حاوی داده‌ها یا پیام‌های کنترلی است. حداقل محموله موثر برای بسته‌های داده‌ای (ConnData) 20 بایت است که توسط مشخصات BLE

<sup>1</sup> Reserved for Future Use (RFU)

<sup>2</sup> Channel Selection (ChSel)

<sup>3</sup> Transmitter Address (TxAdd)

<sup>4</sup> Receiver Address (RxAdd)

<sup>5</sup> Advertiser's device Address (AdvA)

<sup>6</sup> Advertising Data (AdvData)

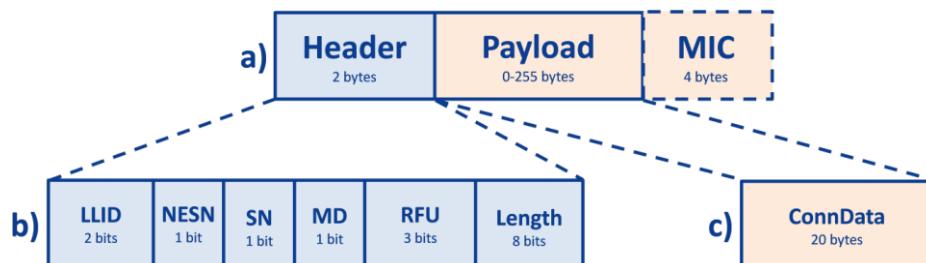
<sup>7</sup> Logical Link Identifier (LLID)

<sup>8</sup> Next Expected Sequence Number (NESN)

<sup>9</sup> Sequence Number (SN)

<sup>10</sup> More Data (MD)

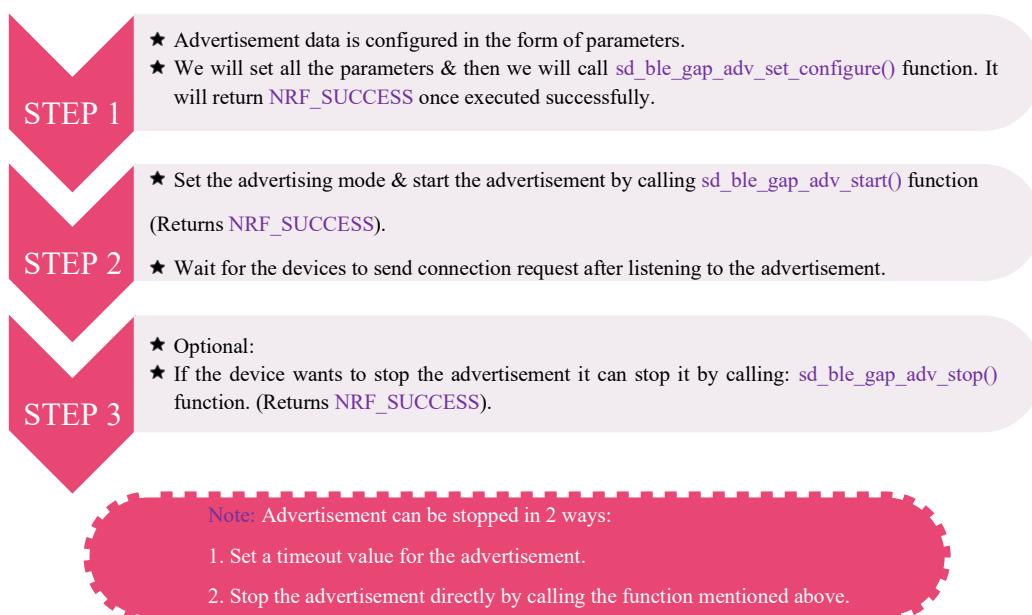
تعیین شده است. محدودیت ۲۰ بایت وابسته به LL نیست، بلکه یک مشخصه لایه GATT است، جایی که مشخص شده است حداکثر واحد انتقال (MTU) برای یک بسته ارسال شده بین یک دستگاه اصلی و یک دستگاه فرعی باید کمتر از ۲۳ بایت باشد. سپس، با حذف هدر دو بایتی، محموله موثر می‌تواند حداکثر ۲۰ بایت باشد. علاوه بر این، یک مقدار اختیاری بررسی یکپارچگی پیام (MIC<sup>۱</sup>) با چهار بایت وجود دارد که برای تأیید اعتبار PDU داده‌ها در یک اتصال رمزگذاری شده LL استفاده می‌شود.



شکل (۱۷-۱۲) ساختار PDU یک بسته اتصال [۱]

در ادامه به چند ریزه‌کاری مهم در کد زدن پرداخته می‌شود.

نمایه ۱۸-۱۲ گام‌های تنظیم تبلیغات به شرح شکل ۱۸-۱۲ است:



شکل (۱۸-۱۲) گام‌های تنظیم تبلیغات

نمایه ۱۹-۱۲ انواع داده‌هایی که می‌تواند در بسته تبلیغات باشد به شرح شکل ۱۹-۱۲ است:

<sup>۱</sup> Message Integrity Check (MIC)

```

typedef struct
{
    ble_advdata_name_type_t      name_type;
    uint8_t                      short_name_len;
    bool                         include_appearance;
    uint8_t                      flags;
    int8_t *                     p_tx_power_level;
    ble_advdata_uuid_list_t      uuids_more_available;
    ble_advdata_uuid_list_t      uuids_complete;
    ble_advdata_uuid_list_t      uuids_solicited;
    ble_advdata_conn_int_t *     p_slave_conn_int;
    ble_advdata_manuf_data_t *   p_manuf_specific_data;
    ble_advdata_service_data_t * p_service_data_array;
    uint8_t                      service_data_count;
    bool                         include_ble_device_addr;
    ble_advdata_le_role_t        le_role;
    ble_advdata_tk_value_t *     p_tk_value;
    uint8_t *                     p_sec_mgr_oob_flags;
    ble_gap_lesc_oob_data_t *   p_lesc_data;
} ble_advdata_t;

```

شکل (۲۰-۱۲) انواع داده قابل تبلیغ

#### ❖ انواع آدرس:

آدرس تصادفی: آدرس تصادفی یک مقدار ۶ بایتی تصادفی است که به عنوان آدرس دستگاه به دستگاه BLE تخصیص داده می‌شود. دو نوع آدرس تصادفی وجود دارد:

آدرس تصادفی ثابت (Random Static Address)

آدرس تصادفی خصوصی (Random Private Address)

آدرس تصادفی ثابت: این یک آدرس تصادفی است که کاربر می‌تواند مستقیماً تخصیص دهد. این آدرس تغییر نخواهد کرد و در طول چرخه قدرت دستگاه ثابت باقی می‌ماند.

آدرس تصادفی خصوصی: این یک آدرس تصادفی است که ممکن است چندین بار در طول چرخه قدرت دستگاه تغییر کند. این آدرس به دو نوع تقسیم می‌شود:

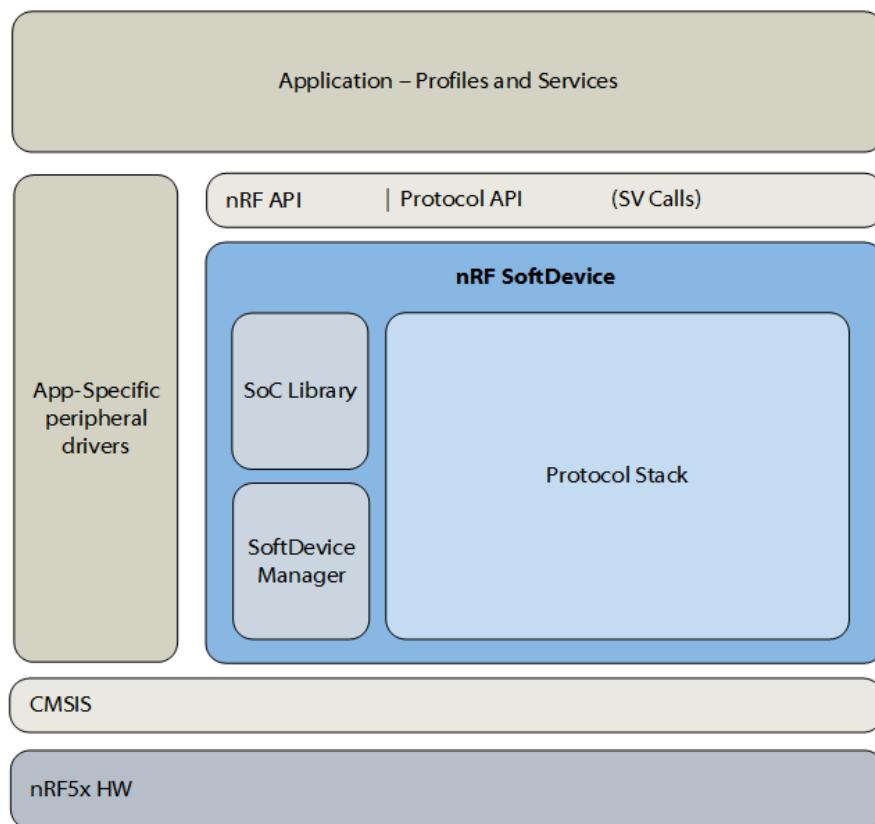
آدرس خصوصی قابل حل (Resolvable Private Address): دستگاهها با استفاده از یک کلید مشترک با یکدیگر ارتباط برقرار می‌کنند، این کلید برای حل آدرس خصوصی دستگاه استفاده می‌شود.

آدرس خصوصی غیرقابل حل (Unresolvable Private Address): آدرس دستگاه به مرور زمان به روزرسانی می‌شود که این امر امنیت دستگاه را افزایش می‌دهد. مشخصات اصلی BLE پیشنهاد می‌کند که این زمان ۱۵ دقیقه باشد.

## Soft Device -۱۳-۱۲

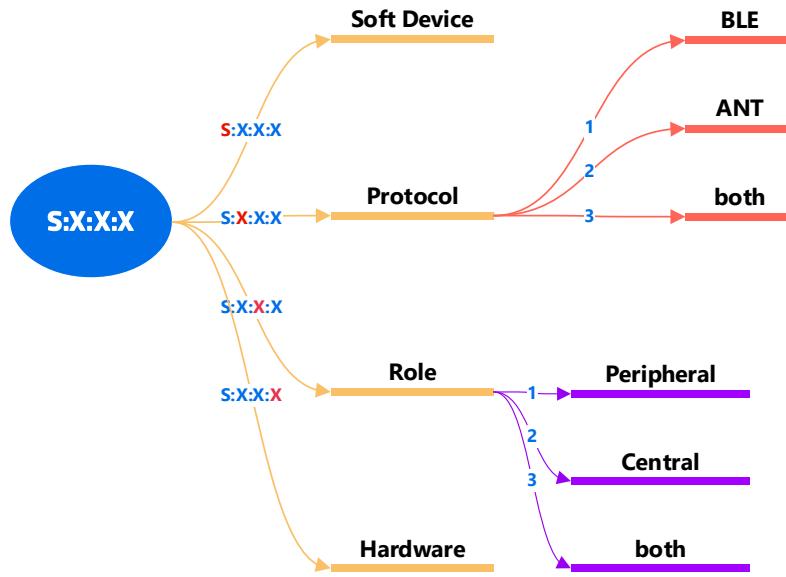
در واقع یک برنامه نرمافزاری از پیش کامپایل شده است که توسط شرکت Nordic برای دستگاه‌های بلوتوث کم‌صرف توسعه داده شده است. این نرمافزار تمام زیرساخت‌های لازم برای ارتباط بلوتوث را فراهم می‌کند و به توسعه‌دهندگان اجازه می‌دهد تا روی نوشتن کدهای کاربردی خود تمرکز کنند و نیازی به پیاده‌سازی جزئیات پروتکل بلوتوث نداشته باشند.

مشابه آنچه در شکل ۲۱-۱۲ مشاهده می‌شود، به عبارت ساده‌تر، Soft Device یک لایه نرمافزاری است که بین سخت‌افزار دستگاه و کدهای کاربردی شما قرار می‌گیرد و تمام کارهای مربوط به ارتباط بلوتوث را مدیریت می‌کند.



شکل (۲۱-۱۲) اجزای Soft Device

هر نرمافزار دستگاه یک کد نام‌گذاری خاص دارد که شامل اطلاعاتی در مورد قابلیت‌های آن است. به طور کلی، این کد به شکل S:X:X:X است که شکل ۲۲-۱۲ نشان داده شده است:



شکل (۲۲-۱۲) نحوه نام‌گذاری Soft Device

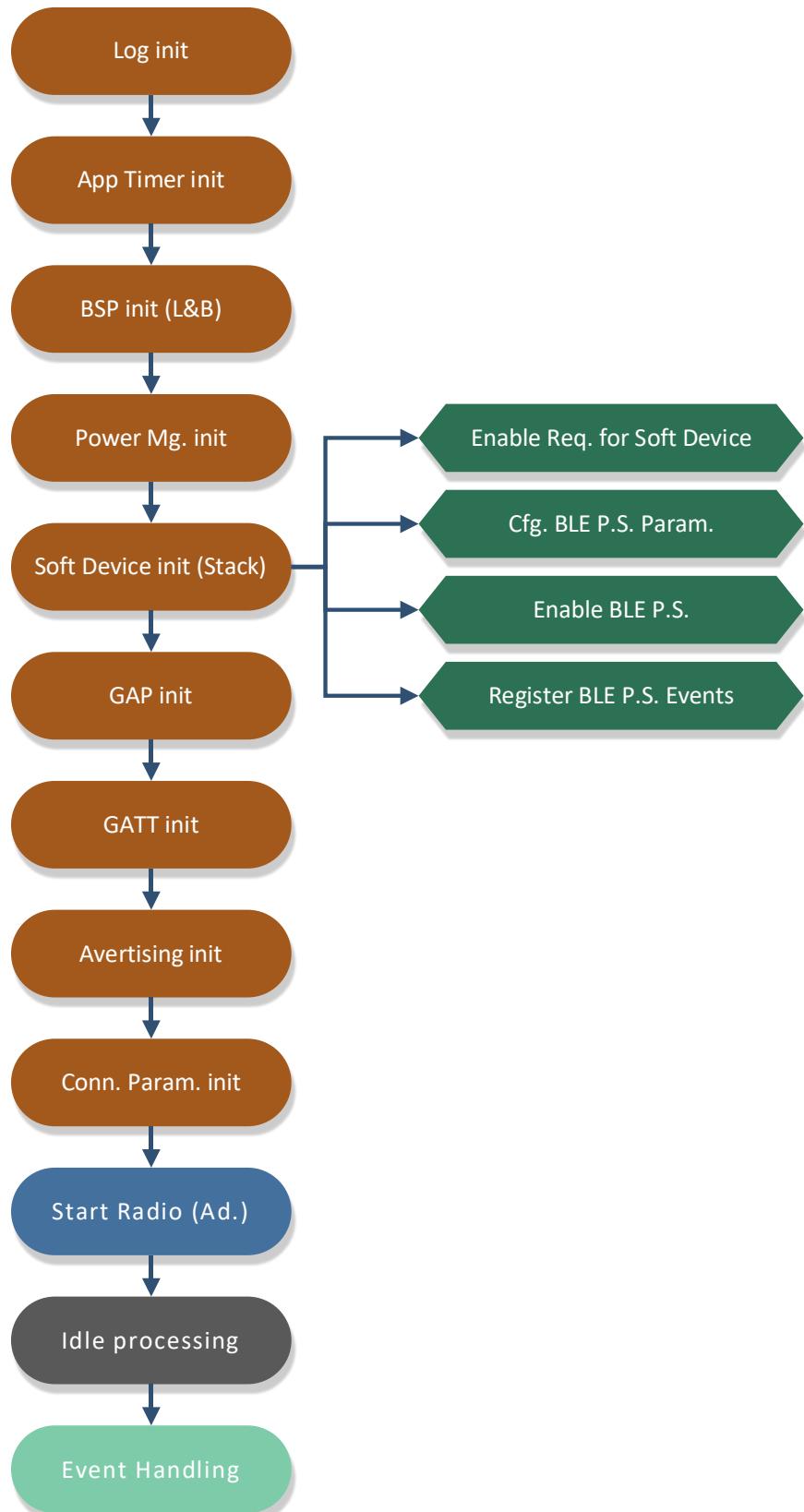
مثال:

S112: نرمافزاری است که فقط از بلوتوث کم‌صرف پشتیبانی می‌کند، نقش Peripheral دارد و برای سخت‌افزار nRF52832 طراحی شده است.

S132: نرمافزاری است که از بلوتوث کم‌صرف پشتیبانی می‌کند، می‌تواند هم نقش Central و هم Peripheral داشته باشد و برای سخت‌افزار nRF52832 طراحی شده است.

#### ۱۴-۱۲- شکل کلی کد زدن برای BLE

برای کد زدن در پروژه‌های نیازمند BLE، به‌طور کلی باید از روندnamی شکل ۲۳-۱۲ پیروی کرد:



شکل (۱۲-۲۳) روند کلی کد زدن برای BLE



## فصل سیزدهم

### بلوتوث با کنترل LED و نظارت بر دکمه فشاری

۱-۱۳ - کد

در این پروژه، نخستین گام در پیاده‌سازی یک سیستم ساده بلوتوث با استفاده از nRF52832 برداشته شد. هدف اصلی، نمایش وضعیت اتصال بلوتوث و LED‌ها با استفاده از رنگ‌های مختلف و نیز مشاهده وضعیت LED و دکمه فشاری در تلفن‌های همراه بود. در ادامه، خلاصه‌ای از عملکرد سیستم و کد مربوط به آن ارائه می‌شود.

```

1. #include <stdint.h>
2. #include <string.h>
3. #include "nordic_common.h"
4. #include "nrf.h"
5. #include "nrf_drv_gpiote.h"
6. #include "app_error.h"
7. #include "ble.h"
8. #include "ble_err.h"
9. #include "ble_hci.h"
10. #include "ble_srv_common.h"
11. #include "ble_advdata.h"
12. #include "ble_conn_params.h"
13. #include "nrf_sdh.h"
14. #include "nrf_sdh_ble.h"
15. #include "boards.h"
16. #include "app_timer.h"
17. #include "app_button.h"
18. #include "ble_lbs.h"
19. #include "nrf_ble_gatt.h"
20. #include "nrf_ble_qwr.h"
21. #include "nrf_pwr_mgmt.h"
22.
23. #include "nrf_log.h"
24. #include "nrf_log_ctrl.h"
25. #include "nrf_log_default_backends.h"
26.
27.
28. #define ADVERTISING_LED           22          // Red           /**< Is on when device is
advertising. */
29. #define CONNECTED_LED            17          // Green         /**< Is on when device has
connected. */
30. #define LEDBUTTON_LED             23          // Blue          /**< LED to be toggled with
the help of the LED Button Service. */
31. #define LEDBUTTON_BUTTON          25          //              /**< Button that will trigger
the notification event with the LED Button Service */
32. #define micro_power_ctrl          18          //
33.
34. #define DEVICE_NAME               "Naghiloo_intern"   /**< Name of device. Will
be included in the advertising data. */
35.
36. #define APP_BLE_OBSERVER_PRIO      3           /**< Application's BLE
observer priority. You shouldn't need to modify this value. */

```

```

37. #define APP_BLE_CONN_CFG_TAG           1                                /**< A tag identifying the
SoftDevice BLE configuration. */
38.
39. #define APP_ADV_INTERVAL               64                               /**< The advertising interval
(in units of 0.625 ms; this value corresponds to 40 ms). */
40. #define APP_ADV_DURATION              BLE_GAP_ADV_TIMEOUT_GENERAL_UNLIMITED   /**< The advertising time-out
(in units of seconds). When set to 0, we will never time out. */
41.
42.
43. #define MIN_CONN_INTERVAL             MSEC_TO_UNITS(100, UNIT_1_25_MS)    /**< Minimum acceptable
connection interval (0.5 seconds). */
44. #define MAX_CONN_INTERVAL             MSEC_TO_UNITS(200, UNIT_1_25_MS)    /**< Maximum acceptable
connection interval (1 second). */
45. #define SLAVE_LATENCY                0                                 /**< Slave latency. */
46. #define CONN_SUP_TIMEOUT              MSEC_TO_UNITS(4000, UNIT_10_MS)    /**< Connection supervisory
time-out (4 seconds). */
47.
48. #define FIRST_CONN_PARAMS_UPDATE_DELAY APP_TIMER TICKS(20000)          /**< Time from initiating
event (connect or start of notification) to first time sd_ble_gap_conn_param_update is called (15 seconds). */
49. #define NEXT_CONN_PARAMS_UPDATE_DELAY APP_TIMER TICKS(5000)            /**< Time between each call
to sd_ble_gap_conn_param_update after the first call (5 seconds). */
50. #define MAX_CONN_PARAMS_UPDATE_COUNT 3                                /**< Number of attempts
before giving up the connection parameter negotiation. */
51.
52. #define BUTTON_DETECTION_DELAY        APP_TIMER TICKS(50)                /**< Delay from a GPIO
event until a button is reported as pushed (in number of timer ticks). */
53.
54. #define DEAD_BEEF                   0xDEADBEEF                         /**< Value used as error code
on stack dump, can be used to identify stack location on stack unwind. */
55.
56.
57. BLE_LBS_DEF(m_lbs);                                              /**< LED Button Service
instance. */
58. NRF_BLE_GATT_DEF(m_gatt);                                         /**< GATT module instance. */
59. NRF_BLE_QWR_DEF(m_qwr);                                         /**< Context for the Queued
Write module.*/
60.
61. static uint16_t m_conn_handle = BLE_CONN_HANDLE_INVALID;          /**< Handle of the current
connection. */
62.
63. static uint8_t m_adv_handle = BLE_GAP_ADV_SET_HANDLE_NOT_SET;      /**< Advertising handle used
to identify an advertising set. */
64. static uint8_t m_enc_advdata[BLE_GAP_ADV_SET_DATA_SIZE_MAX];       /**< Buffer for storing an
encoded advertising set. */
65. static uint8_t m_enc_scan_response_data[BLE_GAP_ADV_SET_DATA_SIZE_MAX];  /**< Buffer for storing an
encoded scan data. */
66.
67. /**@brief Struct that contains pointers to the encoded advertising data. */
68. static ble_gap_adv_data_t m_adv_data =
69. {
70.     .adv_data =
71.     {
72.         .p_data = m_enc_advdata,
73.         .len    = BLE_GAP_ADV_SET_DATA_SIZE_MAX
74.     },
75.     .scan_rsp_data =
76.     {
77.         .p_data = m_enc_scan_response_data,
78.         .len    = BLE_GAP_ADV_SET_DATA_SIZE_MAX
79.     }
80. };
81. };
82.
83. enum {press=0 , release=1} state = release;    /**<The button action (press/release). Used in
"button_event_handler" */
84.
85. /**@brief Function for assert macro callback.
86. *
87. * @details This function will be called in case of an assert in the SoftDevice.
88. *
89. * @warning This handler is an example only and does not fit a final product. You need to analyze
90. *         how your product is supposed to react in case of Assert.
91. * @warning On assert from the SoftDevice, the system can only recover on reset.
92. *
93. * @param[in] line_num    Line number of the failing ASSERT call.
94. * @param[in] p_file_name File name of the failing ASSERT call.
95. */
96. void assert_nrf_callback(uint16_t line_num, const uint8_t * p_file_name)
97. {
98.     app_error_handler(DEAD_BEEF, line_num, p_file_name);
99. }
100.
101.
```

```

102. /**@brief Function for handling Queued Write Module errors.
103. *
104. * @details A pointer to this function will be passed to each service which may need to inform the
105. *         application about an error.
106. *
107. * @param[in] nrf_error Error code containing information about what went wrong.
108. */
109. static void nrf_qwr_error_handler(uint32_t nrf_error)
110. {
111.     APP_ERROR_HANDLER(nrf_error);
112. }
113.
114.
115. /**@brief Function for handling write events to the LED characteristic.
116. *
117. * @param[in] p_lbs Instance of LED Button Service to which the write applies.
118. * @param[in] led_state Written/desired state of the LED.
119. */
120. static void led_write_handler(uint16_t conn_handle, ble_lbs_t * p_lbs, uint8_t led_state)
121. {
122.     if (led_state)
123.     {
124.         nrf_gpio_pin_set(LEDBUTTON_LED);
125.         NRF_LOG_INFO("Received LED ON!");
126.     }
127.     else
128.     {
129.         nrf_gpio_pin_clear(LEDBUTTON_LED);
130.         NRF_LOG_INFO("Received LED OFF!");
131.     }
132. }
133.
134.
135. /**@brief Function for handling the Connection Parameters Module.
136. *
137. * @details This function will be called for all events in the Connection Parameters Module that
138. *         are passed to the application.
139. *
140. * @note All this function does is to disconnect. This could have been done by simply
141. *       setting the disconnect_on_fail config parameter, but instead we use the event
142. *       handler mechanism to demonstrate its use.
143. *
144. * @param[in] p_evt Event received from the Connection Parameters Module.
145. */
146. static void on_conn_params_evt(ble_conn_params_evt_t * p_evt)
147. {
148.     ret_code_t err_code;
149.
150.     if (p_evt->evt_type == BLE_CONN_PARAMS_EVT_FAILED)
151.     {
152.         err_code = sd_ble_gap_disconnect(m_conn_handle, BLE_HCI_CONN_INTERVAL_UNACCEPTABLE);
153.         APP_ERROR_CHECK(err_code);
154.     }
155. }
156.
157.
158. /**@brief Function for handling a Connection Parameters error.
159. *
160. * @param[in] nrf_error Error code containing information about what went wrong.
161. */
162. static void conn_params_error_handler(uint32_t nrf_error)
163. {
164.     APP_ERROR_HANDLER(nrf_error);
165. }
166.
167.
168. /**@brief Function for initializing the Connection Parameters module.
169. */
170. static void conn_params_init(void)
171. {
172.     ret_code_t err_code;
173.     ble_conn_params_init_t cp_init;
174.
175.     memset(&cp_init, 0, sizeof(cp_init));
176.
177.     cp_init.p_conn_params          = NULL;
178.     cp_init.first_conn_params_update_delay = FIRST_CONN_PARAMS_UPDATE_DELAY;
179.     cp_init.next_conn_params_update_delay = NEXT_CONN_PARAMS_UPDATE_DELAY;
180.     cp_init.max_conn_params_update_count = MAX_CONN_PARAMS_UPDATE_COUNT;
181.     cp_init.start_on_notify_cccd_handle = BLE_GATT_HANDLE_INVALID;
182.     cp_init.disconnect_on_fail        = false;
183.     cp_init.evt_handler             = on_conn_params_evt;
184.     cp_init.error_handler          = conn_params_error_handler;

```

```

185.     err_code = ble_conn_params_init(&cp_init);
186.     APP_ERROR_CHECK(err_code);
187. }
188.
189.
190.
191. /**@brief Function for initializing services that will be used by the application.
192. */
193. static void services_init(void)
194. {
195.     ret_code_t          err_code;
196.     ble_lbs_init_t      init      = {0};
197.     nrf_ble_qwr_init_t qwr_init = {0};
198.
199.     // Initialize Queued Write Module.
200.     qwr_init.error_handler = nrf_qwr_error_handler;
201.
202.     err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
203.     APP_ERROR_CHECK(err_code);
204.
205.     // Initialize LBS.
206.     init.led_write_handler = led_write_handler;
207.
208.     err_code = ble_lbs_init(&m_lbs, &init);
209.     APP_ERROR_CHECK(err_code);
210. }
211.
212.
213. /**@brief Function for starting advertising.
214. */
215. static void advertising_start(void)
216. {
217.     ret_code_t          err_code;
218.
219.     err_code = sd_ble_gap_adv_start(m_adv_handle, APP_BLE_CONN_CFG_TAG);
220.     APP_ERROR_CHECK(err_code);
221.
222.     nrf_gpio_pin_set(ADVERTISING_LED);
223. }
224.
225.
226. /**@brief Function for initializing the Advertising functionality.
227. *
228. * @details Encodes the required advertising data and passes it to the stack.
229. *          Also builds a structure to be passed to the stack when starting advertising.
230. */
231. static void advertising_init(void)
232. {
233.     ret_code_t          err_code;
234.     ble_adpdata_t       advdata;
235.     ble_adpdata_t       srdata;
236.
237.     ble_uuid_t          adv_uuids[] = {{LBS_UUID_SERVICE, m_lbs.uuid_type}};
238.
239.     // Build and set advertising data.
240.     memset(&advdata, 0, sizeof(advdata));
241.
242.     advdata.name_type      = BLE_ADVDATA_FULL_NAME;
243.     advdata.include_appearance = true;
244.     advdata.flags           = BLE_GAP_ADV_FLAGS_LE_ONLY_GENERAL_DISC_MODE;
245.
246.
247.     memset(&srdata, 0, sizeof(srdata));
248.     srdata.uuids_complete.uuid_cnt = sizeof(adv_uuids) / sizeof(adv_uuids[0]);
249.     srdata.uuids_complete.p_uuids = adv_uuids;
250.
251.     err_code = ble_advdata_encode(&advdata, m_adv_data.adv_data.p_data, &m_adv_data.adv_data.len);
252.     APP_ERROR_CHECK(err_code);
253.
254.     err_code = ble_advdata_encode(&srdata, m_adv_data.scan_rsp_data.p_data, &m_adv_data.scan_rsp_data.len);
255.     APP_ERROR_CHECK(err_code);
256.
257.     ble_gap_adv_params_t adv_params;
258.
259.     // Set advertising parameters.
260.     memset(&adv_params, 0, sizeof(adv_params));
261.
262.     adv_params.primary_phy      = BLE_GAP_PHY_1MBPS;
263.     adv_params.duration        = APP_ADV_DURATION;
264.     adv_params.properties.type = BLE_GAP_ADV_TYPE_CONNECTABLE_SCANNABLE_UNDIRECTED;
265.     adv_params.p_peer_addr      = NULL;
266.     adv_params.filter_policy    = BLE_GAP_ADV_FP_ANY;
267.     adv_params.interval         = APP_ADV_INTERVAL;

```

```

268.
269.     err_code = sd_ble_gap_adv_set_configure(&m_adv_handle, &m_adv_data, &adv_params);
270.     APP_ERROR_CHECK(err_code);
271. }
272.
273.
274. /**@brief Function for initializing the GATT module.
275. */
276. static void gatt_init(void)
277. {
278.     ret_code_t err_code = nrf_ble_gatt_init(&m_gatt, NULL);
279.     APP_ERROR_CHECK(err_code);
280. }
281.
282.
283. /**@brief Function for the GAP initialization.
284. */
285. * @details This function sets up all the necessary GAP (Generic Access Profile) parameters of the
286. *         device including the device name, appearance, and the preferred connection parameters.
287. */
288. static void gap_params_init(void)
289. {
290.     ret_code_t err_code;
291.     ble_gap_conn_params_t gap_conn_params;
292.     ble_gap_conn_sec_mode_t sec_mode;
293.
294.     BLE_GAP_CONN_SEC_MODE_SET_OPEN(&sec_mode);
295.
296.     err_code = sd_ble_gap_device_name_set(&sec_mode,
297.                                         (const uint8_t *)DEVICE_NAME,
298.                                         strlen(DEVICE_NAME));
299.     APP_ERROR_CHECK(err_code);
300.
301.     memset(&gap_conn_params, 0, sizeof(gap_conn_params));
302.
303.     gap_conn_params.min_conn_interval = MIN_CONN_INTERVAL;
304.     gap_conn_params.max_conn_interval = MAX_CONN_INTERVAL;
305.     gap_conn_params.slave_latency = SLAVE_LATENCY;
306.     gap_conn_params.conn_sup_timeout = CONN_SUP_TIMEOUT;
307.
308.     err_code = sd_ble_gap_ppcp_set(&gap_conn_params);
309.     APP_ERROR_CHECK(err_code);
310. }
311.
312.
313. /**@brief Function for handling BLE events.
314. */
315. * @param[in] p_ble_evt Bluetooth stack event.
316. * @param[in] p_context Unused.
317. */
318. static void ble_evt_handler(ble_evt_t const * p_ble_evt, void * p_context)
319. {
320.     ret_code_t err_code;
321.
322.     switch (p_ble_evt->header.evt_id)
323.     {
324.         case BLE_GAP_EVT_CONNECTED:
325.             NRF_LOG_INFO("Connected");
326.             nrf_gpio_pin_set(CONNECTED_LED);
327.             nrf_gpio_pin_clear(ADVERTISING_LED);
328.             m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
329.             err_code = nrf_ble_qwr_conn_handle_assign(&m_qwr, m_conn_handle);
330.             APP_ERROR_CHECK(err_code);
331.             //err_code = app_button_enable();
332.             nrf_drv_gpiote_in_event_enable(LED_BUTTON_BUTTON, true);
333.             //APP_ERROR_CHECK(err_code);
334.             break;
335.
336.         case BLE_GAP_EVT_DISCONNECTED:
337.             NRF_LOG_INFO("Disconnected");
338.             nrf_gpio_pin_clear(CONNECTED_LED);
339.             m_conn_handle = BLE_CONN_HANDLE_INVALID;
340.             //err_code = app_button_disable();
341.             nrf_drv_gpiote_in_event_disable(LED_BUTTON_BUTTON);
342.             //APP_ERROR_CHECK(err_code);
343.             advertising_start();
344.             break;
345.
346.         case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
347.             // Pairing not supported
348.             err_code = sd_ble_gap_sec_params_reply(m_conn_handle,
349.                                         BLE_GAP_SEC_STATUS_PAIRING_NOT_SUPP,
350.                                         NULL,

```

```

351.             APP_ERROR_CHECK(err_code);
352.         break;
353.
354.
355.     case BLE_GAP_EVT_PHY_UPDATE_REQUEST:
356.     {
357.         NRF_LOG_DEBUG("PHY update request.");
358.         ble_gap_phys_t const phys =
359.         {
360.             .rx_phys = BLE_GAP_PHY_AUTO,
361.             .tx_phys = BLE_GAP_PHY_AUTO,
362.         };
363.         err_code = sd_ble_gap_phy_update(p_ble_evt->evt.gap_evt.conn_handle, &phys);
364.         APP_ERROR_CHECK(err_code);
365.     } break;
366.
367.     case BLE_GATTS_EVT_SYS_ATTR_MISSING:
368.         // No system attributes have been stored.
369.         err_code = sd_ble_gatts_sys_attr_set(m_conn_handle, NULL, 0, 0);
370.         APP_ERROR_CHECK(err_code);
371.     break;
372.
373.     case BLE_GATTC_EVT_TIMEOUT:
374.         // Disconnect on GATT Client timeout event.
375.         NRF_LOG_DEBUG("GATT Client Timeout.");
376.         err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gattc_evt.conn_handle,
377.                                         BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
378.         APP_ERROR_CHECK(err_code);
379.     break;
380.
381.     case BLE_GATTS_EVT_TIMEOUT:
382.         // Disconnect on GATT Server timeout event.
383.         NRF_LOG_DEBUG("GATT Server Timeout.");
384.         err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gatts_evt.conn_handle,
385.                                         BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
386.         APP_ERROR_CHECK(err_code);
387.     break;
388.
389.     default:
390.         // No implementation needed.
391.     break;
392. }
393. }
394.
395.
396. /**@brief Function for initializing the BLE stack.
397. *
398. * @details Initializes the SoftDevice and the BLE event interrupt.
399. */
400. static void ble_stack_init(void)
401. {
402.     ret_code_t err_code;
403.
404.     err_code = nrf_sdh_enable_request();
405.     APP_ERROR_CHECK(err_code);
406.
407.     // Configure the BLE stack using the default settings.
408.     // Fetch the start address of the application RAM.
409.     uint32_t ram_start = 0;
410.     err_code = nrf_sdh_ble_default_cfg_set(APP_BLE_CONN_CFG_TAG, &ram_start);
411.     APP_ERROR_CHECK(err_code);
412.
413.     // Enable BLE stack.
414.     err_code = nrf_sdh_ble_enable(&ram_start);
415.     APP_ERROR_CHECK(err_code);
416.
417.     // Register a handler for BLE events.
418.     NRF_SDH_BLE_OBSERVER(m_ble_observer, APP_BLE_OBSERVER_PRIO, ble_evt_handler, NULL);
419. }
420.
421.
422. /**@brief Function for handling the idle state (main loop).
423. *
424. * @details If there is no pending log operation, then sleep until next the next event occurs.
425. */
426. static void idle_state_handle(void)
427. {
428.     if (NRF_LOG_PROCESS() == false)
429.     {
430.         nrf_pwr_mgmt_run();
431.     }
432. }
433.

```

```

434.
435. /**@brief Function for initializing power management.
436. */
437. static void power_management_init(void)
438. {
439.     ret_code_t err_code;
440.     err_code = nrf_pwr_mgmt_init();
441.     APP_ERROR_CHECK(err_code);
442. }
443.
444.
445. /**@brief Function for handling events from the button handler module.
446. */
447. * @param[in] pin_no      The pin that the event applies to.
448. * @param[in] button_action The button action (press/release). Unused for Ali.
449. */
450. static void button_event_handler(nrf_drv_gpiote_pin_t pin_no, nrf_gpiote_polarity_t button_action)
451. {
452.     ret_code_t err_code;
453.
454.     switch (pin_no)
455.     {
456.         case LEDBUTTON_BUTTON:
457.             NRF_LOG_INFO("%s", (state == release) ? "pressed" : "released");
458.             err_code = ble_lbs_on_button_change(m_conn_handle, &m_lbs, state);
459.             if (err_code != NRF_SUCCESS &&
460.                 err_code != BLE_ERROR_INVALID_CONN_HANDLE &&
461.                 err_code != NRF_ERROR_INVALID_STATE &&
462.                 err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
463.             {
464.                 APP_ERROR_CHECK(err_code);
465.             }
466.             break;
467.
468.         default:
469.             APP_ERROR_HANDLER(pin_no);
470.             break;
471.     }
472.
473.     state=1-state;
474. }
475.
476.
477. /**@brief Function for initializing the button handler module.
478. */
479. static void buttons_init(void)
480. {
481.     ret_code_t err_code;
482.
483.     nrf_gpio_cfg_input(LEDBUTTON_BUTTON, NRF_GPIO_PIN_PULLDOWN);
484.     err_code = nrf_drv_gpiote_init();
485.     APP_ERROR_CHECK(err_code);
486.     nrf_drv_gpiote_in_config_t in_config = GPIOTE_CONFIG_IN_SENSE_TOGGLE(true);
487.     in_config.pull = NRF_GPIO_PIN_PULLDOWN;
488.     err_code = nrf_drv_gpiote_in_init(LEDBUTTON_BUTTON, &in_config, button_event_handler);
489.     APP_ERROR_CHECK(err_code);
490.
491. /**@brief Function for the LEDs initialization.
492. */
493. * @details Initializes all LEDs used by the application.
494. */
495. static void leds_init(void)
496. {
497.     nrf_gpio_cfg_output(ADVERTISING_LED);
498.     nrf_gpio_cfg_output(CONNECTED_LED);
499.     nrf_gpio_cfg_output(LEDBUTTON_LED);
500. }
501.
502.
503. /**@brief Function for the Timer initialization.
504. */
505. * @details Initializes the timer module.
506. */
507. static void timers_init(void)
508. {
509.     // Initialize timer module, making it use the scheduler
510.     ret_code_t err_code = app_timer_init();
511.     APP_ERROR_CHECK(err_code);
512. }
513.
514.
515. static void log_init(void)

```

```

516. {
517.     ret_code_t err_code = NRF_LOG_INIT(NULL);
518.     APP_ERROR_CHECK(err_code);
519.
520.     NRF_LOG_DEFAULT_BACKENDS_INIT();
521. }
522.
523.
524. /**@brief Function for application main entry.
525. */
526. int main(void)
527. {
528.     nrf_gpio_cfg_output(micro_power_ctrl1);
529.     nrf_gpio_pin_set(micro_power_ctrl1);
530.
531.     // Initialize.
532.     log_init();
533.     leds_init();
534.     timers_init();
535.     buttons_init();
536.     power_management_init();
537.     ble_stack_init();
538.     gap_params_init();
539.     gatt_init();
540.     services_init();
541.     advertising_init();
542.     conn_params_init();
543.
544.     // Start execution.
545.     NRF_LOG_INFO("Blinky example started.");
546.     advertising_start();
547.
548.     // Enter main loop.
549.     for (;;)
550.     {
551.         idle_state_handle();
552.     }
553. }
554.
555.
556. /**
557. * @}
558. */
559.

```

## ۱۳-۲- توضیحات کد

LED‌های مختلف برای نمایش وضعیت‌ها:

LED قرمز (Advertising\_LED): زمانی که دستگاه در حال تبلیغ (Advertising) است روشن می‌شود.

LED سبز (CONNECTED\_LED): زمانی که دستگاه به یک دستگاه دیگر متصل می‌شود روشن می‌شود.

LED آبی (LEDBUTTON\_LED): این LED با استفاده از اپلیکیشن بلوتوث کنترل می‌شود.

دکمه فشاری:

از دکمه فشاری (LEDBUTTON\_BUTTON) برای ارسال یک اعلان به اپلیکیشن بلوتوث در هنگام

فشار دادن استفاده می‌شود. این اعلان می‌تواند برای کنترل LED آبی نیز استفاده شود.

**عملکرد بلوتوث:**

دستگاه با نام Naghiloo\_intern تبلیغ می‌شود.

پارامترهای اتصال (Connection Parameters) با استفاده از مقادیر پیش‌فرض پیکربندی شده‌اند.

دستگاه پس از اتصال به صورت خودکار LED‌ها را تنظیم می‌کند و قابلیت مدیریت پارامترهای اتصال و GATT را دارد.

**ساختار کلی کد:**

: main(void) این تابع نقطه ورود اصلی برنامه است. تمامی مازول‌ها و توابع مورد نیاز در اینجا مقداردهی اولیه می‌شوند و تبلیغات بلوتوث شروع می‌شود.

: led\_write\_handler این تابع برای مدیریت دستورات LED از طریق سرویس بلوتوث (ble\_lbs) استفاده می‌شود. زمانی که یک دستور برای روشن یا خاموش کردن LED از اپلیکیشن بلوتوث دریافت می‌شود، این تابع فراخوانی می‌شود.

: ble\_evt\_handler این تابع تمامی رویدادهای BLE را مدیریت می‌کند، از جمله اتصال و قطع اتصال، و اقدامات مناسب را انجام می‌دهد.

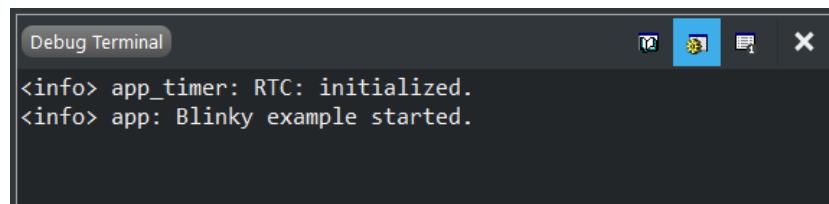
: advertising\_start و advertising\_init این توابع مسئولیت پیکربندی و شروع تبلیغات بلوتوث را دارند.

: buttons\_init این تابع دکمه‌های فشاری را مقداردهی اولیه می‌کند تا در هنگام فشار دادن دکمه، بتوان رویداد مناسب را به BLE ارسال کرد.

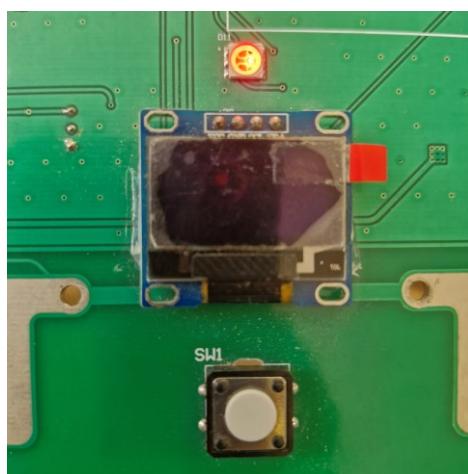
: log\_init و timers\_init این توابع تایمربهای و سیستم لاغ‌گیری را مقداردهی اولیه می‌کنند.

نحوه عملکرد:

دستگاه در حالت تبلیغ بلوتوث شروع به کار می‌کند (شکل ۱-۱۳) و مطابق شکل ۲-۱۳، LED قرمز روشن می‌شود.

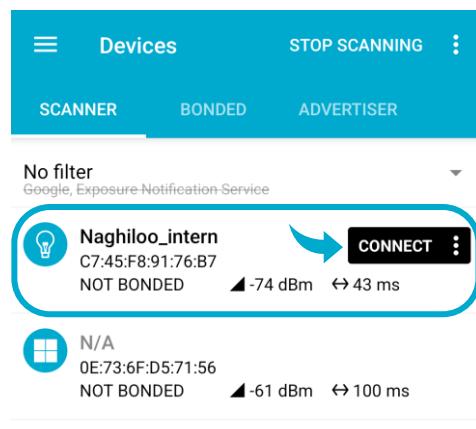


شکل (۱-۱۳) شروع برنامه

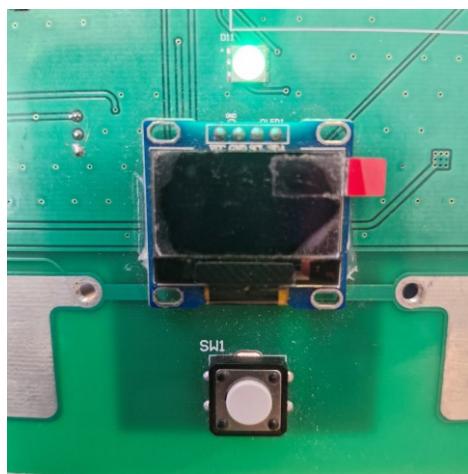


شکل (۲-۱۳) دستگاه در حالت تبلیغات

زمانی که یک دستگاه به nRF52 متصل شود (شکل ۱-۱۳)، مطابق شکل ۴-۱۳، LED سبز روشن می‌شود و LED قرمز خاموش می‌شود.



شکل (۳-۱۳) نحوه اتصال در



شکل (۴-۱۳) دستگاه در حالت متصل

در این فرآیند از برنامه خود شرکت سازنده‌ی تراشه یعنی nRF Connect (شکل ۵-۱۳) استفاده شد.



شکل (۵-۱۳) برنامه‌ی nRF Connect

بعد از اتصال (شکل ۵-۱۳)، از طریق اپلیکیشن، LED آبی می‌تواند روشن یا خاموش شود. البته از آن جایی که در تصویر ۶-۱۳ مشخص هست، شرکت NORDIC نرمافزارهای متنوعی را توسعه داده است و برای این کار می‌توان به دو شیوه عمل کرد، به کمک نرمافزار nRF Connect و یا به کمک نرم افزار

nRF Blinky

```
Debug Terminal
<info> app_timer: RTC: initialized.
<info> app: Blinky example started.
<info> app: Connected
```

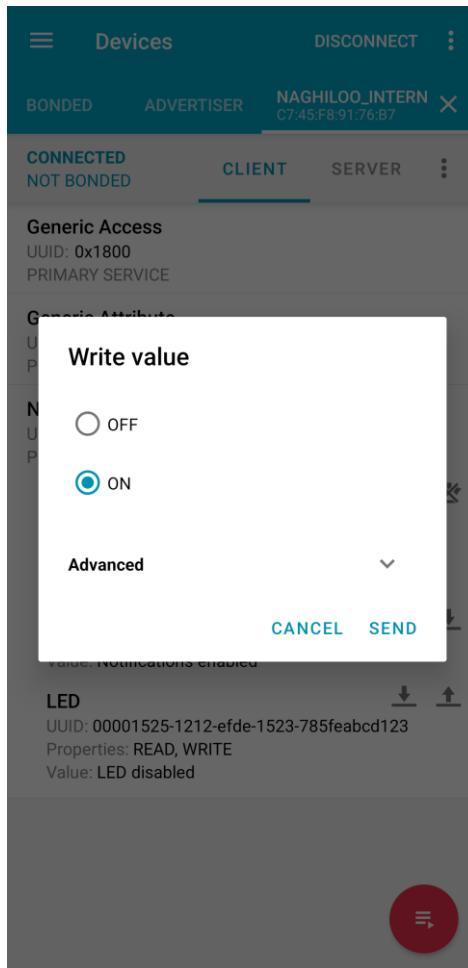
شکل (۵-۱۳) پیام متصل شدن



شكل (۱۳-۶) نرم افزارهای مختلف NORDIC

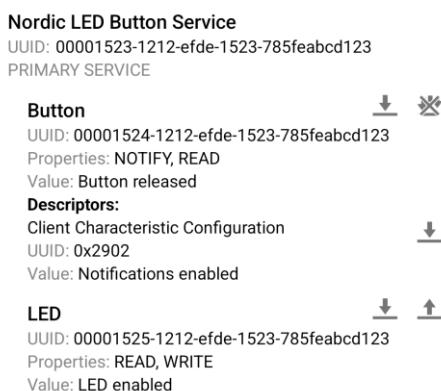
محیط نرم افزار nRF Connect مختص هدف خاصی نیست و برای هر نوع کاری قابلیت استفاده دارد؛ اما نرم افزار nRF Blinky مختص این پروژه هست و حتی محیطی گرافیکی تر و کاربرپسندتری دارد که در ادامه به ترتیب در شکل های ۷-۱۳ و ۹-۱۳ هر دو برنامه به نمایش گذاشته شده است.

نکته مهمی که باید ذکر شود این است که حتماً برای دریافت وضعیت دکمه‌ی فشاری، باید حتماً در Notification nRF Connect را فعال کرد در غیر این صورت وضعیت آن در نرم افزار تغییر نخواهد کرد.



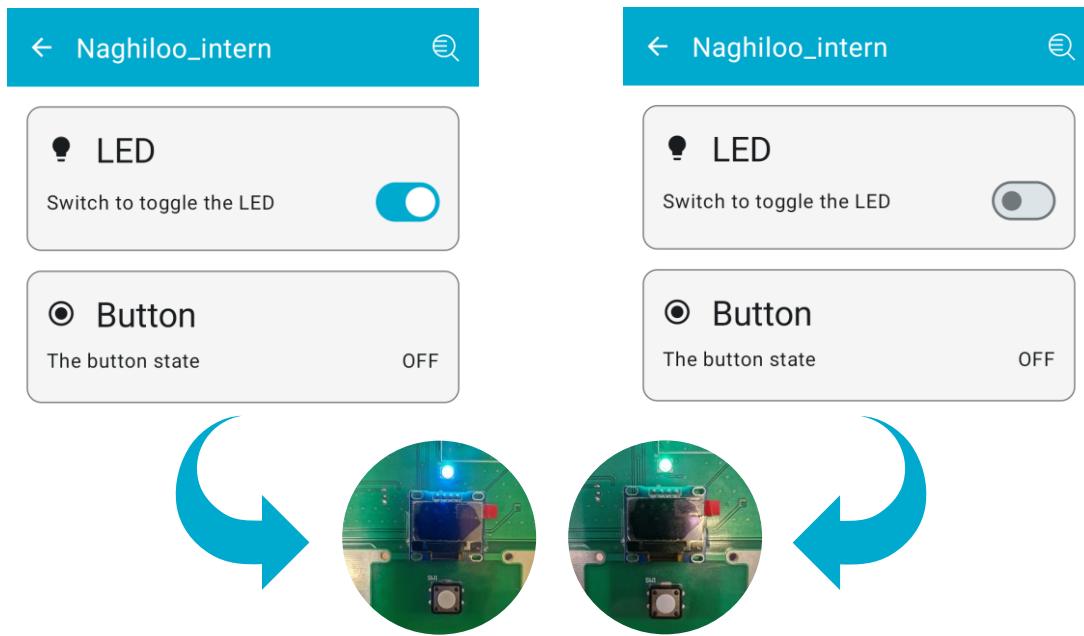
شکل (۷-۱۳) واپایش LED آبی با کمک نرمافزار nRF Connect

همان‌طور که در شکل ۸-۱۳ مشخص هست، هربار که خودمان LED آبی را خاموش یا روشن می‌کنیم، وضعیت آن، به صورت LED enable یا LED disable جلوی LED enable Value نوشته می‌شود که این قابلیت‌ها ناشی از آن است که Properties ما، هم از نوع READ و هم از نوع WRITE هست. در کنار واپایش LED آبی، با فشار دادن دکمه فشاری، اعلان‌های مربوطه را به اپلیکیشن ارسال می‌شود.



شکل (۸-۱۳) واپایش در nRF Connect

همان‌طور که بالاتر گفته شد، استثنائاً در این پروژه یعنی LBS<sup>۱</sup> می‌توان از نرمافزار مختص آن یعنی استفاده کرد که در شکل ۹-۱۳ مشخص هست.



شکل (۹-۱۳) واپایش در nRF Blinky

در انتها، تمام مراحل کار به صورت log در پایانه دیباگر داخلی (به صورت خلاصه RTT Logger) در شکل ۱۰-۱۳ مشاهده می‌شود.

A screenshot of a computer terminal window titled 'Debug Terminal'. The window has a dark background and contains white text representing log messages. The messages are as follows:

```

<info> app_timer: RTC: initialized.
<info> app: Blinky example started.
<info> app: Connected
<info> app: Received LED ON!
<info> app: pressed
<info> app: released
<info> app: Received LED OFF!
<info> app: Disconnected

```

شکل (۱۰-۱۳) پیام‌های Logger

### ۱۳-۳- نتیجه‌گیری

این پروژه یک پایه خوب برای توسعه برنامه‌های پیچیده‌تر با استفاده از بلوتوث است، مانند واپایش

<sup>۱</sup> LED Button Service (LBS)

بیشتر دستگاهها، مدیریت بهتر اتصال‌ها و استفاده از ویژگی‌های پیشرفته‌تر GATT.



## نتیجه‌گیری

در پایان این دوره کارآموزی، توانایی‌های علمی و عملیاتی نگارنده در زمینه برنامه‌نویسی و کار با تراشه‌های بی‌سیم به طور چشم‌گیری افزایش یافته است. تجربه‌های عملی به دست آمده از کار با nRF52832 و استفاده از ابزارهایی مانند SEGGER Embedded Studio و پروتکل‌های ارتباطی بلوتوث، پایه‌ای قوی برای توسعه پروژه‌های پیچیده‌تر در آینده فراهم آورده است. همچنین، آشنایی با چالش‌های واقعی در پیاده‌سازی سیستم‌های بی‌سیم IoT، درک عمیق‌تری از نحوه طراحی و اجرای این سیستم‌ها را به همراه داشته است. این تجربه به عنوان یک مرحله مهم در مسیر حرفه‌ای کارآموز باقی خواهد ماند.



## مراجع

- [1] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino and D. Formica, "Performance Evaluation of Bluetooth Low Energy: A Systematic Review," *Sensors*, vol. 17, no. 12, p. 2898, 2017.
- [2] E. Tsimbalo, X. Fafoutis and R. J. Piechocki, "CRC Error Correction in IoT Applications," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 1, pp. 361-369, 2017.
- [3] A. F. Harris, V. Khanna, G. Tuncay, R. Want and R. Kravets, "Bluetooth Low Energy in Dense IoT," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 30-36, 2016.