

دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

**Over the Air Programming**

مهدی صادقی

استاد راهنما: دکتر مجید نبی

تیر 97

فهرست مطالب

[فصل اول 6](#_Toc517732336)

[مقدمه 6](#_Toc517732337)

[فصل دوم 7](#_Toc517732338)

[اینترنت اشیاء 7](#_Toc517732339)

[سنسورها 8](#_Toc517732340)

[شبکه‌ی سنسورها 10](#_Toc517732341)

[کاربردهای WSN 12](#_Toc517732342)

[مسیریابی در شبکه‌ی سنسور 13](#_Toc517732343)

[فصل سوم 15](#_Toc517732344)

[سیستمهای تعبیه شده 15](#_Toc517732345)

[AVR 15](#_Toc517732346)

[ARM 17](#_Toc517732347)

[فصل چهارم 18](#_Toc517732348)

[برنامه ریزی از راه دور 18](#_Toc517732349)

[فواید استفاده از OTA 20](#_Toc517732350)

[فصل پنجم 21](#_Toc517732351)

[پیاده سازی 21](#_Toc517732352)

[روش اول 21](#_Toc517732353)

[جزئیات برنامه پایتون فرستنده 22](#_Toc517732354)

[جزئیات کتابخانه flashlib 24](#_Toc517732355)

[جزئیات گیرنده 33](#_Toc517732356)

[روش دوم 33](#_Toc517732357)

[جزئیات کتابخانهی ArduinoOTA 34](#_Toc517732358)

[مراجع 35](#_Toc517732359)

# فصل اول

# مقدمه

با پیشرفت تکنولوژی در زمینه ساخت تراشه‌ها به ما امکان ساخت سنسورهایی کوچک از نظر سایز، کم مصرف و در عین حال دقیق را داده است. این سنسورها در عمل می‌توانند یک کمیت را انداز‌‌ه‌گیری کرده و پس از پردازش آن‌ها را به وسیله‌ی یک خط ارتباطی که اکثرا ارتباطی رادیویی است به یک مرکز اصلی برای پردازش بیشتر بفرستند. با کوچک‌تر و ارزان‌تر شدن این سنسورها امکان استفاده از تعداد زیادی از آن‌ها در یک سیستم واحد برای کاربردی خاص را می‌دهد که این امکان در سال‌های اخیر موضوع تحقیقات بسیاری بوده است که قصد ساخت شبکه‌ای از سنسورها برای دریافت و پردازش اطلاعات در محیط‌های بزرگ را دارند. اما نکته ای که وجود دارد این است که در بسیاری از این شبکه ها، خود سنسورها هم می­توانند داده­ها را پردازش کنند و قابل برنامه ریزی هستند. در این شرایط و در تعداد بالا برای آپدیت نرم­افزاری نمتوان همه­ی این سنسور ها را از محل جمع آوری و تک تک برنامه ریزی کرد. به همین دلیل نیاز به یک روش برای برنامه ریزی سنسورها و میکروکنترلر ها از راه دور احساس میشود که در این گزارش سعی در ارائه­ی روش هایی برای برطرف کردن این نیاز میشود.

در ابتدا به اینترنت اشیاء و شبکه­های سنسوری میپردازیم و پس از آن برخی از سیستم­های تعبیه شده و معماری­های میکروکنترلر ها بررسی میشوند. در پایان به نحوه­ی برنامه ریزی از راه دور (OTA) این میکروکنترلرها در شبکه سنسوری و جزئیات پیاده سازی انجام گرفته پرداخته خواهد شد.

# فصل دوم

# اینترنت اشیاء[[1]](#footnote-1)

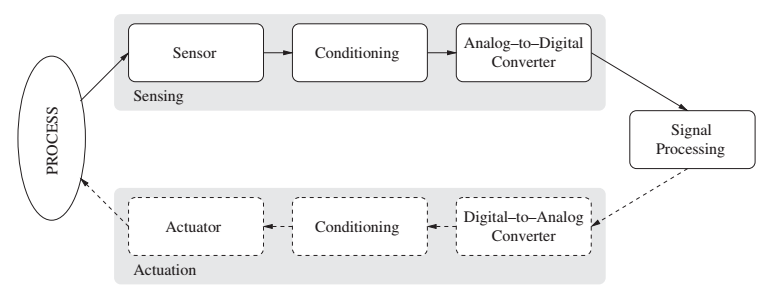
هدف از این فصل آشنایی اولیه با اینترنت اشیاء و مفاهیم مرتبط با آن است.

اینترنت اشیاء اصطلاحی است که برای توصیف دنیایی که در آن اشیاء قادر خواهند بود با اتصال به اینترنت یا به کمک ابزارهای ارتباطی، با سایر اشیاء تعامل داشته باشند و اطلاعات خود را با هم و یا با انسان ها به اشتراک بگذارند و کلاس جدیدی از قابلیت­ها، برنامه های کاربردی و سرویس­ها را ارائه دهند. دنیایی که درآن تمامی اشیا و دستگاههای نامتجانس قابلیت آدرس­دهی و در نتیجه قابلیت کنترل­پذیری دارند. این ارتباط می­تواند ارتباط بین گوشی­های تلفن هوشمند باشد و یا ارتباط بین چراغ­های راهنمایی و رانندگی با یکدیگر. اینترنت اشیا، نوآوری آینده در زمینه تکنولوژی­های بیسیم محسوب میشود و در بسیاری از زمینه­ها و حوزه­ها دارای کاربرد است.

یکی دیگر از دلاید اهمیت یافتن این تکنولوژی رشد تعداد وسایل – مانند سنسور یا تگ­های RFID – در اطراف ماست که می­توانند با یکدیگر از طریق یک سیستم آدرس دهی یکتا ارتباط برقرار کنند. در واقع کلمه­ی اشیاء در عبارت اینترنت اشیاء می‌توانند رجوع شود به طیف گسترده‌ای از دستگاه‌ها مانند ایمپلنت نظارت بر قلب، فرستنده زیست تراشه در حیوانات مزرعه، حلزون‌های الکتریکی در آب‌های ساحلی، خودروها با سنسورها، دستگاه‌های تجزیه و تحلیل [دی ان ای](https://fa.wikipedia.org/wiki/%D8%AF%DB%8C_%D8%A7%D9%86_%D8%A7%DB%8C) برای نظارت بر محیط زیست، مواد غذایی یا دستگاه‌های عملیات میدانی که به آتش نشانان در عملیات جستجو و نجات کمک می‌کند. دانشمندان حقوقی «اشیاء» را به عنوان یک «نا ترکیبی از سخت‌افزار، نرم‌افزار، اطلاعات و خدمات» قلمداد می‌کنند [1].

در ادامه به تعریف برخی مفاهیم مرتبط با این حوزه که در بحث برنامه­ریزی از راه دور (OTA) اهمیت می­یابند می­پردازیم.

## سنسورها[[2]](#footnote-2)

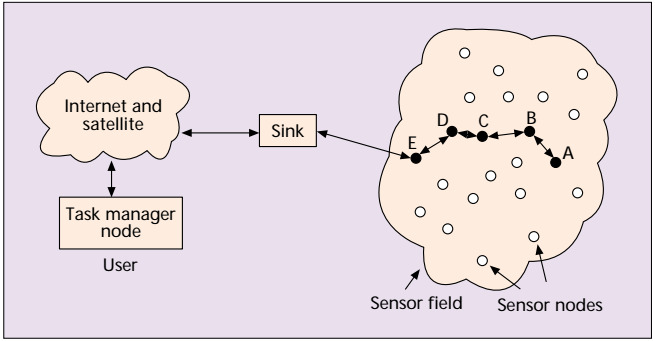
سنسور به ابزاری گفته می‌شود که بتواند اطلاعاتی از یک شیء فیزیکی و یا یک اتفاق پیرامون خود به دست آورد و آن را به سیگنال‌هایی تبدیل کند که قابل استفاده و تحلیل باشند. در واقع اگر مبدل[[3]](#footnote-3) را دستگاهی فرض کنیم که انرژی را از حالتی به حالت دیگر تبدیل می‌کند، سنسورها نوعی مبدل محسوب می‌شوند که انرژی در دنیای فیزیکی را به سیگنال‌های الکتریکی تبدیل می‌کنند که برای سیستم‌های کامپیوتری و کنترلرها قابل فهم باشد. در شکل (1،1)

شکل (2.‏0‏0.1) مراحل کار سنسور [1]

مراحل کار یک سنسور نشان داده شده. در مرحله‌ی اول اطلاعات یک پدیده‌ی فیزیکی توسط سنسور گرفته شده و تبدیل به سیگنال‌های الکتریکی می‌شود، اما در اکثر مواقع این سیگنال‌ها مستقیما قابل استفاده نیستند. به همین دلیل سیگنال الکتریکی تولید شده وارد مرحله Conditioning می شود. در این مرحله کارهای مختلفی روی سیگنال صورت می‌گیرد تا آن را برای استفاده مناسب کند. برای مثال می‌توان سیگنال را تقویت و یا تضعیف کرد تا در محدوده قابل قبول قرار گیرد. در مرحله‌ی آخر سیگنال نهایی تولید شده را به وصیله مبدل آنالوگ به دیجیتال به فرمت قابل فهم برای کامپیوترهای دیجیتالی تبدیل می‌کنیم. برخی از سیستم‌ها در کنار سنسور یک محرک[[4]](#footnote-4) نیز قرار می‌دهند که به آن امکان می‌دهد یک پدیده را مستقیما تغییر دهد. برای مثال برای کنترل دمای خانه کنار یک سنسور دما محرکی قرار می‌دهند که در صورت افزایش دما وسایل سرمایشی خانه را روشن کند [1]. برخی سنسورها هم فراتر از یک سنسور عادی عمل می‌کنند. به این صورت که می‌توانند داده‌هایی که به دست آمده را خود تحلیل کنند و بر مبنای آن تصمیم بگیرند که این نوع سنسورها را سنسور هوشمند[[5]](#footnote-5) می‌نامند [2]. در واقع سنسورهای هوشمند نیازی به یک کنترلر بیرونی برای تصمیم گیری ندارند.

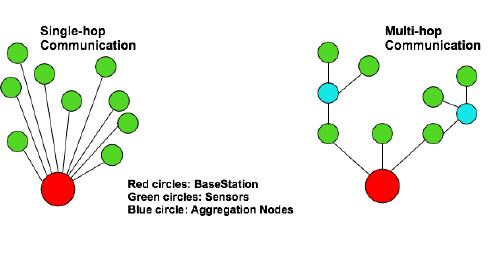
## شبکه‌ی سنسورها[[6]](#footnote-6)

شبکه‌ی سنسورها از تعداد بسیار زیادی سنسور تشکیل شده‌اند که هر کدام از آن‌ها امکان دریافت اطلاعاتی از محیط را دارند و می‌توانند روی این اطلاعات پردازش انجام داده و یا آن‌ها را از طریق یک زیرساخت شبکه‌ای (که معمولا بی‌سیم است) به یک مرکز اصلی منتقل کنند. در این شبکه‌ها هر کدام از این سنسور‌ها را یک گره[[7]](#footnote-7) می‌نامند. در صورتی که شبکه برای انتقال اطلاعات از یکی از انواع شبکه‌های بی‌سیم استفاده کند، به آن شبکه‌ی سنسورهای بی‌سیم[[8]](#footnote-8) می‌گویند.

در یک WSN گره‌ها به صورت شکل (1،2) در محیط پخش می‌شوند. هر کدام از گره‌ها می‌توانند اطلاعات را دریافت و به مرکز اصلی[[9]](#footnote-9) برسانند. گره‌ها از طریق پروتکل‌های تک‌ پله‌ای یا چند پله‌ای (بسته به کاربردی که برای شبکه در نظر گرفته شده) داده‌های خود را به مقصد می‌رسانند. در این نوع شبکه‌ها غالبا لازم نیست تمام گره ها با دقت جایگزاری شده باشند و می‌توان به صورت تصادفی آن‌ها را در محیط پخش کرد. البته برخی پروتکل‌های مسیریابی نیز وجود دارند که شبکه را ملزم به جایگزاری دقیق گره‌ها می‌کنند.

شکل (2،2) ساختار شبکه‌ی سنسورها [13]

تقریبا در تمام کاربردهای WSN تمام گره‌ها داده‌های خود را به یک مرکز اصلی ارسال می‌کنند و در مواردی مرکز اصلی می‌تواند درخواست‌هایی را برای گره‌ها ارسال کند. همچنین گره‌ها از نظر منابع انرژی، پردازشی و پهنای باند محدودند و به همین دلیل باید روی منابعی که در اختیار دارند مدیریت کنند. از طرفی اکثر گره‌ها ساکن هستند بجز چند گره خاص. در واقع در سایر انواع شبکه گره‌ها می‌توانند حرکت کنند که باعث تغییر توپولوژی می‌گردد اما در WSN این گونه نیست.

 در زیر ویژگی‌هایی که در اکثر شبکه‌های سنسوری بی‌سیم مشترک است به صورت خلاصه آورده شده.

شکل (2،3) تفاوت شبکه‌های چند پله‌ای و تک‌ پله‌ای

* **استفاده از باتری به عنوان منبع انرژی:** گره‌ها انرژی خود را از باتری می‌گیرند، به همین دلیل مصرف انرژی یکی از اصلی‌ترین مسائلی است که در هنگان طراحی پروتکل‌ها و سنسور باید در نظر گرفت به طوری که سعی می‌شود پروتکل‌ها به گونه‌ای طراحی شوند که کمترین میزان انرژی را مصرف کنند.
* **قدرت پردازشی در گره‌ها:** در اکثر شبکه‌های سنسور، گره‌ها امکان پردازش اطلاعاتی که به دست می‌آورند را دارند، اما به دلیل محدودیت انرژی قدرت پردازشی آن‌ها کم است.
* **ارتباط چند پله‌ای[[10]](#footnote-10) و یا تک پله‌ای[[11]](#footnote-11):** اکثر شبکه‌های سنسوری به یکی از دو دسته بندی تک پله‌ای و یا چند پله‌ای تقسیم می‌شوند. در حالت تک پله‌ای بین هر گره و مرکز اصلی گره‌ی دیگری وجود ندارد. در واقع همه‌ی گره‌ها مستقیما به مرکز اصلی متصلند، در حالی که در حالت چند پله‌ای ممکن است داده قبل از رسیدن به مرکز اصلی از بین چند گره رد شده باشد. در شکل (1،3) شبکه‌های تک پله‌ای و چند پله‌ای آورده شده است.
* **پویایی[[12]](#footnote-12):** گره‌ها در یک شبکه‌ی سنسور باید پویا باشند. به این معنی که بتوانند در صورتی که یک یا چند گره دیگر از کار افتادند و یا اینکه گره‌های جدیدی به شبکه اضافه شد، بدون این که به عملکرد کلی شبکه صدمه بزنند در شرایط جدید به فعالیت خود ادامه دهند [3].

## کاربردهای WSN

در ادامه به صورت خلاصه چند نمونه از کاربردهای شبکه‌ی سنسورهای بی‌سیم در دنیای واقعی آورده شده.

* **جمع آوری داده از محیط های بزرگ**

برای بدست جمع آوری و تجمیع داده‌های به دست آمده از یک محیط می‌توان از WSN استفاده کرد. به این صورت که هر گره در بازه‌های زمانی خاص داده‌هایی را که جمع‌آوری کرده است را به مرکز اصلی ارسال می‌کند. مرکز اصلی می‌تواند خود داده‌ها را تحلیل کند و یا این‌ که آن‌ها را به جایی دیگر (برای مثال سرورهای متصل به اینترنت) ارسال کند [4].

* **موارد نظامی**

گره‌های WSN می‌توانند به راحتی در میدان نبرد پیاده‌سازی شوند. اطلاعاتی مانند موضع نیروهای دوست و دشمن، وجود یا عدم وجود تشعشات اتمی و یا شیمیایی و یا حتی مانیتور کردن تجهیزات می‌تواند از طریق WSN به دست آید [4]. کاربرد‌های نظامی امنیت شبکه اهمیتی دو چندان پیدا می‌کند.

* **کنترل ترافیک**

سنسورهایی که می‌توانند میزان عبور و مرور وسایل نقلیه را تشخیص دهند می‌توانند در سطح شهر پخش شده و یک شبکه‌ی سنسور را تشکیل دهند. از داده‌هایی که از این شبکه به دست می‌آید می‌توان در مسیریابی در نقشه‌های الکترونیکی استفاده کرد، به طوری که از مسیرهایی که ترافیک زیادی دارند در مسیریابی تا حد ممکن استفاده نشود. از آنجا که این سنسورها در شهر قرار دارند و ثابت هم هستند، می‌توان آن‌ها را بر بستر شبکه‌های سیمی[[13]](#footnote-13) پیاده سازی کرد [4].

## مسیریابی در شبکه‌ی سنسور

مسیریابی در WSNها به دلیل شرایط خاصی که بر این شبکه‌ها حاکم است چالش‌های خاص خود را دارد. در این شبکه‌ها به دلیل تعداد زیاد گره، نمیتوان یک سیستم آدرس‌دهی جامع ایجاد کرد که به هر کدام از گره‌ها یک آدرس جداگانه بدهد. به همین دلیل پروتکل IP که در اینترنت کاربرد فراوان دارد، در WSN به خوبی عمل نمی‌کند. به دلیل همین تفاوت‌ها پروتکل‌های مسیریابی زیادی ایجاد شده‌اند که علاوه بر استفاده از الگوریتم‌های مسیریابی عادی و ثابت شده در سایر انواع شبکه، از الگوریتم‌هایی که مخصوص شبکه‌های سنسور طراحی شده و با شرایط آن سازگار است نیز استفاده می‌کنند. در واقع بسته به کاربردی که دارید می‌توانید بهترین پروتکل برای خود را انتخاب کنید. در این بخش در ابتدا دو روش دسته‌بندی برای این پروتکل‌ها را می‌گوییم و پس از آن به معرفی برخی از الگوریتم‌های استفاده شده در این پروتکل‌ها می‌پردازیم.

به صورت کلی پروتکل‌های مسیریابی در WSN را می‌توان به چند روش دسته‌بندی کرد. روش اول دسته‌بندی بر اساس ساختار شبکه است که در آن هر پروتکل در یکی از دسته‌های هموار[[14]](#footnote-14) و یا سلسله‌مراتبی[[15]](#footnote-15) قرار می‌گیرد. روش دوم دسته‌بندی، دسته‌بندی بر اساس نحوه‌ی کار پروتکل است که در آن دسته‌های پرسش محور[[16]](#footnote-16)، چند مسیره[[17]](#footnote-17)، مذاکره محور[[18]](#footnote-18) و ارتباط محور[[19]](#footnote-19) تقسیم می‌شوند. در ادامه هر کدام از این دسته‌بندی‌ها توضیح داده شده است. روش سومی نیز برای دسته‌بندی پروتکل‌ها وجود دارد که آن‌ها را به سه دسته‌ی Proactive، Reactive و Hybrid تقسیم می‌کند. در پروتکل‌های Proactive تمام مسیرها قبل از این که واقعا نیاز شوند محاسبه می‌گردند و در حافظه تمام گره‌ها قرار می‌گیرند و پس از آن گره‌ها در صورت نیاز به یک مسیر کافیست در جدول حافظه خود مسیر مورد نظرشان را پیدا کنند. اما در پروتکل‌های Reactive مسیرها تنها در زمانی که واقعا به آن‌ها نیاز شود محاسبه می‌شوند.

پروتکل‌های Hybrid هم بسته به شرایط شبکه از ترکیبی از دو روش قبل استفاده می‌کنند. برای مثال مسیر گره‌های نزدیک خود را به صورت Proactive، و مسیر گره‌های دورتر را از روش Reactive به دست می‌آورند [3].

روش Proactive بسیار سریع است. در واقع گره‌ها برای پیدا کردن مسیر مورد نظرشان به سرعت آن را در حافظه خود پیدا می‌کنند که این باعث کاهش مصرف انرژی در امر مسیریابی میشود. اما اغلب این روش برای شبکه‌های کوچک‌تر مناسب است، زیرا در این روش یک گره باید تمام مسیرها را در حافظه‌ی خود ذخیره کند که این امر در شبکه‌های بزرگ هزینه‌بر است. از طرفی در صورتی که گره‌ها حرکت کنند، تمام جداول مسیریابی در تمام گره‌ها باید بروزرسانی شوند که این امر مصرف انرژی بالایی برای یک WSN دارد. در نتیجه روش Reactive برای شبکه‌های بزرگ و یا شبکه‌هایی که گره‌ها در آن زیاد حرکت می‌کنند روش بهتری است، اما پیدا کردن هر باره‌ی مسیر کاری است که انرژی زیادی مصرف می‌کند [3].

# فصل سوم

# سیستم­های تعبیه شده[[20]](#footnote-20)

سیستم­های تعبیه شده کامپیوترهایی با وظایف خاص هستند که در داخل یک ابزار یا سیستم مکانیکی یا الکترونیکی بزرگتر قرار دارند و در بسیاری از موارد خاصیت Real Time دارند. پخش کننده موسیقی، ساعت دیجیتال، کنترل کننده خودکار هواپیما و کنترلر چراغ راهنمایی و رانندگی از نمونه­های سیستم­های تعبیه شده هستند.

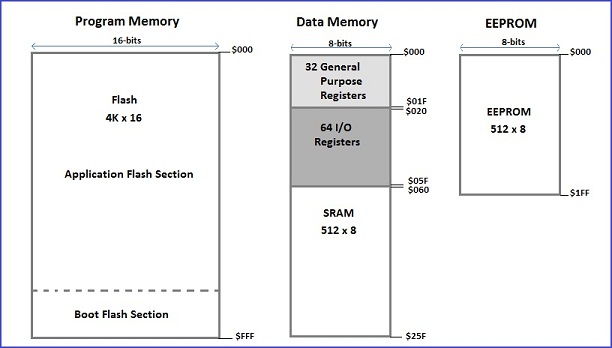
امروزه اکثر سیستم­های تعبیه شده از میکروکنترلرها استفاده می­کنند اما میکروپروسسورهای معمول و DSPها هم در برخی سیستم­های پیچیده رایج هستند. در این پروژه ما روی برنامه ریزی میکروکنترلرها تمرکز کرده­ایم.

امروزه دو معماری در اکثر میکروکنترلرهای استفاده شده در سیستم های تعبیه شده کوچک تا متوسط استفاده میشوند: AVR و ARM. در ادامه به صورت خلاصه به معرفی این دو معماری می­پردازیم و در این معرفی­ها تاکید اصلی بر نحوه قرارگیری برنامه در آن­ها قرار دارد.

## AVR

AVR خانواده ای از میکروکنترلرهاست که توسط شرکت ATMEL طراحی و ساخته می­شوند. این خانواده به دلیل سادگی نسبی و قیمت ارزان در سیستم­های ساده، پروژه­های دانشجویی و سیستم­هایی که قیمت تمام شده­ی پایین نیاز دارند استفاده­ی زیادی دارد.

این مدل میکروکنترلرها سه نوع حافظه دارند: حافظه­ی برنامه، حافظه­ی داده و حافظه­ی EEPROM



شکل(3،1) انواع حافظه در AVR

دستورات برنامه در یک حافظه­ی Flash قرار دارد و هر کدام بین 16 تا 32 بیت طول دارند. یک ویژگی جالب این حافظه صفحه بندی شده بودن آن است. به این معنا که خواندن و نوشتن روی آن تنها در صفحه با سایز مشخص انجام میگیرد. این سایز برای ATMega32 برابر 128 بایت است. یعنی اگر بخواهیم یک بایت را تغییر بدهیم باید کل 128 بایتی که آن بایت در آن قرار دارد را از نو در حافظه بنویسیم.

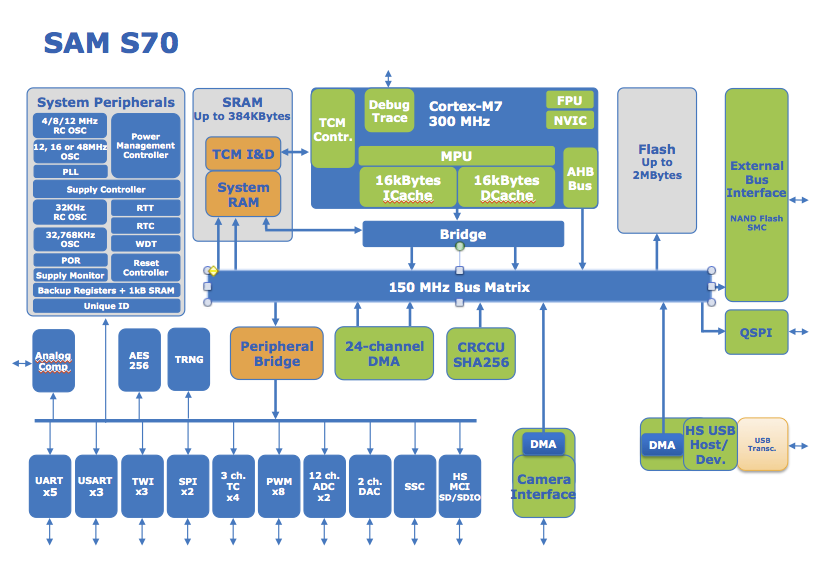
در انتهای حافظه­ی برنامه[[21]](#footnote-21) بخش خاصی قرار دارد که برنامه­ی بوت­لودر در آن قرار میگیرد. بوت­لودر قطعه کدی است که درست بعد از ریست شدن میکروکنترلر اجرا میشود. تنها فرق بین بوت لودر با کد معمولی، دسترسی آن به حافظه­ی برنامه است. به این معنا که تنها بوت­لودر امکان نوشتن در حافظه­ی برنامه را دارد. این ویژگی اهمیت زیادی برای OTA دارد. در فصل بعد از این ویژگی برای پیاده­سازی یکی از انواع OTA استفاده خواهیم کرد.

## ARM

ARM نوعی از معماری پردازنده های کامپیوتری است که منطبق بر طراحی RISC CPU و توسط کمپانی بریتانیایی ARM Holding طراحی شده است. معماری ARM که دستورالعمل های 32 بیتی را پردازش می کند از دهه 1980 تا به امروز در حال توسعه است. ARM مخفف Advanced RISC Machine می باشد. از آنجایی که این معماری بر اساس طراحی RISC بنا شده، هسته اصلی CPU نیاز به 35 هزار ترانزیستور دارد این در حالی است که پردازنده های معمولی رایج x86 که بر اساس CISC طراحی شده اند حداقل نیاز به میلیون ها ترانزیستور دارند. این موضوع مهمترین دلیل مصرف بسیار پایین انرژی در پردازنده های مبتنی بر ARM است که باعث استفاده گسترده آنها در ابزارهای پرتابل مانند تلفن هوشمند یا تبلت شده نیز همین است.

امروزه میکروکنترلرهای بسیاری با استفاده از پردازنده­های ARM ساخته شده­اند که میتوان به میکروکنترلرهای سری SAM از شرکت ATMEL اشاره کرد.

بر خلاف معماری ARM که تمام حافظه­ی داده ها را از حافظه­ی برنامه جدا کرده بود، در این معماری هر دو در یک حافظه قرار دارند و مهم تر این که برنامه اجرا شده در user space امکان نوشتن در آن را دارد. به همین دلیل OTA در این معماری بدون نیاز به ریست کردن قابل انجام است و در واقع بسیاری از سیستم عامل­ها از این قابلیت به صورت پیش فرض برخوردارند.



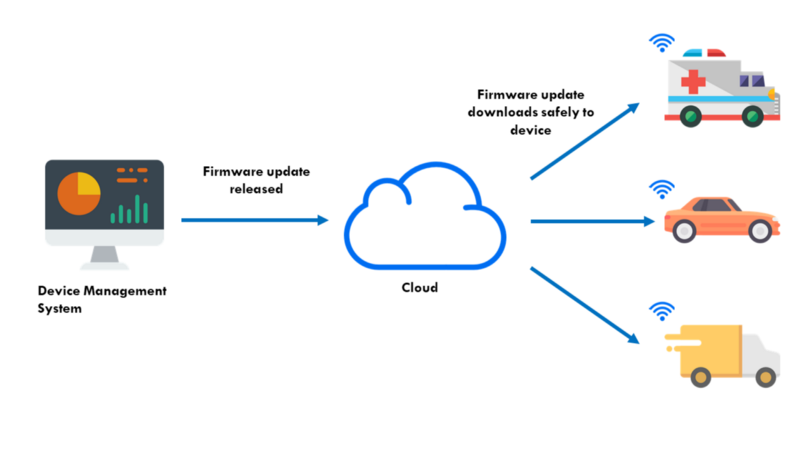
# فصل چهارم

# برنامه ریزی از راه دور

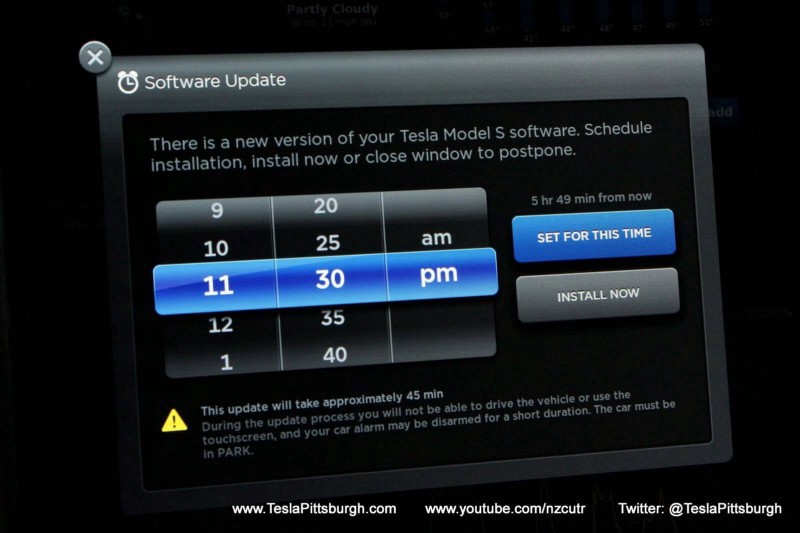
همانطور که در بخش های قبل توضیح داده شد شبکه­ی سنسورها شامل تعداد زیادی سیستم تعبیه شده هستند که در یک محیط گسترده شده­اند. در بساری از موارد خود نود ها بجز جمع آوری اطلاعات برخی پردازش ها را هم می­توانند روی این داده ها انجام دهند. برای مثال در یک سیستم جمع آوری دما ممکن است نودها دما را با دمایی که قبلا ارسال کرده اند مقایسه کنند و تنها اختلاف دما را ارسال کنند. این مقایسه نوعی پردازش اطلاعات است.

حال در نظر بگیرید که در چنین سیستمی نیاز بود نحوه­ی پردازش اطلاعات تغییر کند. برای مثال در سیستم قبل بجای ارسال اختلاف، میانگین دما در بازه های 10 دقیقه ای ارسال شود. این تغییر نیازمند تغییر در کد برنامه در تمامی نودهاست. در حالی که در بسیاری از سیستم­ها میتوان تمام نودها را جمع آوری کرد و با اتصال به رایانه برنامه را بروزرسانی کرد، اما در برخی موارد این کار امکان پذیر نیست و یا صرفه اقتصادی و زمانی ندارد. تصور کنید هزاران نود را در یک جنگل برای ثبت رطوبت پخش کرده­اید و حال میخواهید آن­ها را جمع آوری کنید تا برنامه­ی جدیدی روی آن­ها قرار دهید.

این مشکل تنها در شبکه­ی سنسورها وجود ندارد. تصور کنید یک سیستم تعبیه شده در قله یک کوه قرار گرفته تا وضعیت هوا در آن نقطه را اندازه گیری کند و به مرکز ارسال کند. با این که در اینجا مانند شبکه­ی سنسورها با تعداد زیادی نود روبرو نیستیم، اما به دلیل سختی دسترسی به سیستم همچنان برای آپدیت نرم افزار سیستم با روش سنتی با مشکل مواجهیم. به دلیل همین مشکلات است که سیستم­هایی برای برنامه ریزی از راه دور (OTA) پیشنهاد شده است.



برنامه ریزی از راه دور به روش­های مختلف بروزرسانی نرم­افزار و یا تغییر تنظیمات در سیستم بدون دسترسی مستقیم به آن گفته میشود. تا قبل از OTA شما برای برنامه­ریزی دستگاه مجبور بودید، دستگاه را به صورت فیزیکی در اختیار بگیرید، به کامپیوتر متصل کنید، برنامه­ریزی کنید، دستگاه را دوباره سر هم کنید و به جای اول برگردانید. اما این روش برای شرکت ها غیر قابل توسعه ( از نظر تعداد) و سنگین است، با این حال هنوز استفاده میشود. برای مثال شرکت کرایسلر در سال2015 یک پتچ نرم افزاری را به وسیله پست کردن فلش USB برای مشتریان منتشر کرد. این کار مشتریان را در خطر قرار میداد زیرا این فلش ها میتوانست در بین راه تغییر پیدا کند و بدافزار درون آن­ها قرار گیرد. در مقابل در سال 2016 شرکت تسلا یک آپدیت OTA منتشر کرد که به خودروهای این شرکت اجازه میداد به صورت خودکار پارک کنند. این آپدیت تسلا بدون دخالت مشتریان نصب میشد و در عمل کاربران این خودروها صبح در حالی از خواب بلند شدند که خودروهایشان قابلیت­های جدیدی پیدا کرده بود.



با این که انجام OTA از طریق سیم هم امکان­پذیر است اما در اکثر موارد، این عمل از طریق سیستم­های بیسیم صورت میگیرد مانند Wifi، Bluetooth و یا 3G. امروزه اکثر سیستم عامل­های معروف گوشی­های هوشمند از این قابلیت به صورت پیشفرض پشتیبانی میکنند و کاربر میتواند به صورت اتوماتیک و یا دستی[[22]](#footnote-22) این آپدیت ها را دریافت کند.

## فواید استفاده از OTA

* باگ­ها و رفتار سیستم به صورت مرتب می­تواند بهبود یابد
* شرکت­ها امکان تست ویژگی­های جدید را قبل از عرضه به بازار می­یابند. به این صورت که آپدیت قبل از عرضه عمومی برای تعداد محدودی از دستگاه­ها منتشر شود و پس از اطمینان از سلامت آن را برای سایر مشتریان منتشر کرد.
* باعث کاهش هزینه­های شرکت میشود.
* استقاده کنندگان از دستگاه میتوانند بدون نگرانی بابت آپدیت ها و نحوه­ی دریافت آنان از دستگاه استفاده کنند.
* امکان توسعه کمی دستگاه ها را به شدت بالا میبرد. به این معنا که با افزایش تعداد دستگاه ها مشکلی برای آپدیت­های نرم­افزاری آن­ها پیش نمی­آید.

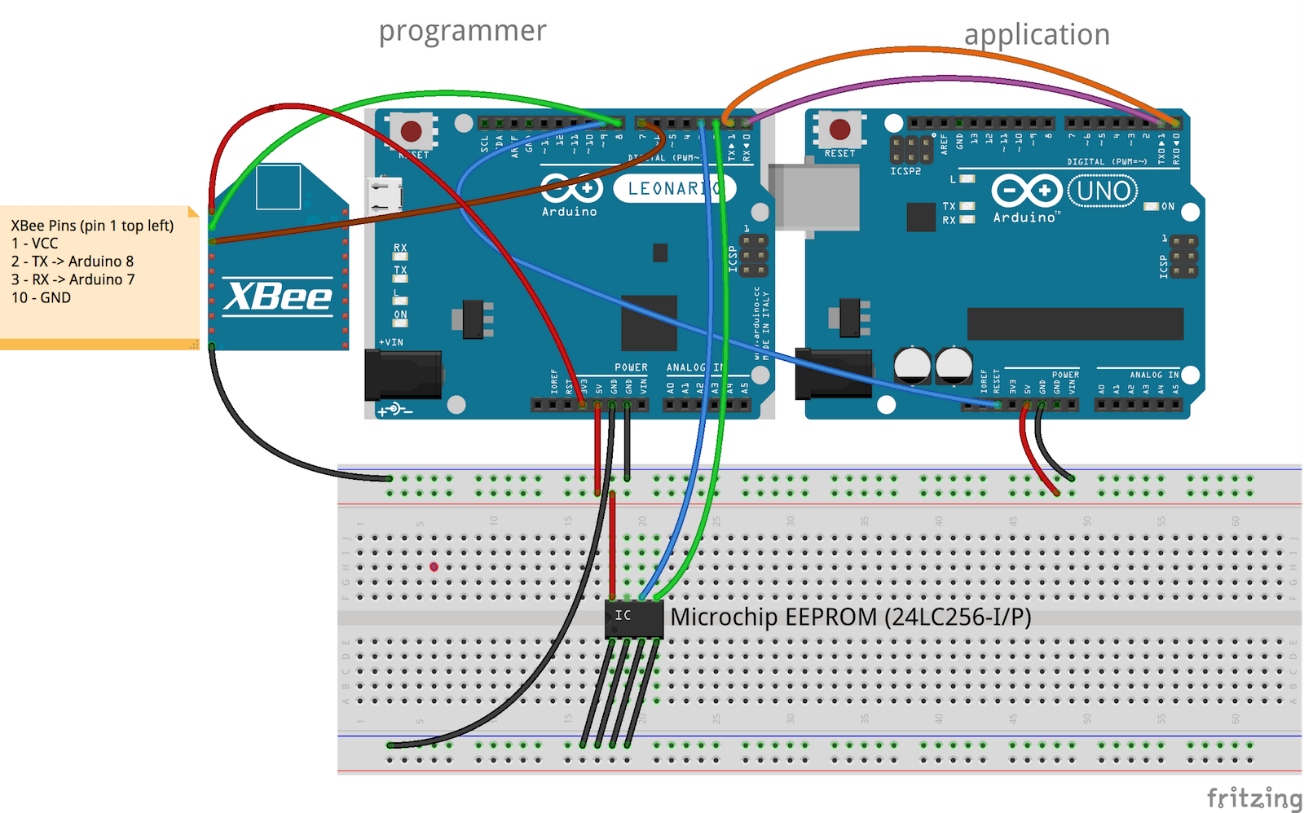
# فصل پنجم

# پیاده سازی

در این پروژه ما روی پیاده سازی OTA به صورت مستقیم روی دو میکروکنترلر مختلف و با دو روش مختلف تمرکز کرده­ایم. در روش اول قصد داریم یک OTA را بر یک آردوینو UNO که خود یک میکروکنترلر AVR است پیاده سازی کنیم و در روش دوم میکروکنترلر یک ESP8266 به عنوان میکروکنترلر هدف انتخاب شده است.

## روش اول

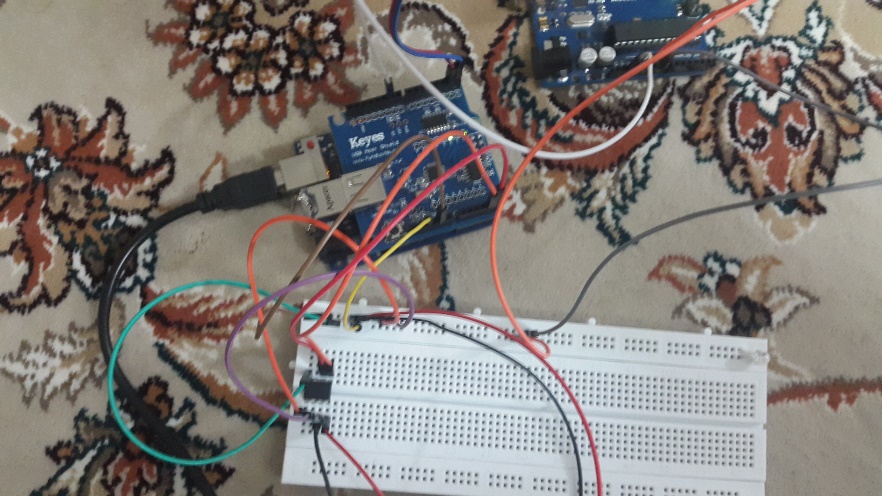
در این روش هدف ایجاد OTA در Arduino است. برد آن به صورت زیر است (با این تفاوت که بجای ماژول XBee از ماژول ESP8266 جهت ارتباط استفاده شده است).



این پروژه شامل بخش­های زیر است:

* آردوینو برنامه ریز: این برد وظیفه­ی پروگرام کردن آردوینو دیگر را دارد.
* آردینو هدف: هدف نهایی پروژه تغییر کد این آردوینو است
* رام: یک ماژول EEPROM 256 کیلوبایتی برای ذخیره برنامه قبل از Flash
* ماژول وایفای: در این پروژه از ماژول ESP8266 جهت اتصال به وایفای استفاده میشود. این ماژول از جمله معروف ترین ماژول ها برای کار با وایفای است و از طریق پورت سریال میتوان با آن ارتباط برقرار کرد.
* برنامه­ی فرستنده: یک برنامه پایتون که فایل hex ورودی را گرفته و پس از حذف آدرس­ها از خطوط برنامه، آن را برای آردوینو برنامه ریز ارسال میکند.

در اینجا یکی از آردوینوها فایل برنامه را از طریق وایفای دانلود کرده، در EEPROM ذخیره میکند و تنها زمانی که کل برنامه به صورت صحیح دریافت شد، شروع به Flash کردن برنامه روی آردوینو دیگر میکند. روش­های دیگری هم برای انجام OTA در میکروکنترلرهای AVR وجود دارد که تمام آن­ها کمبودهایی دارند. در این روش هر بسته از داده چک میشود و ack برای فرستنده ارسال میشود.



### جزئیات برنامه پایتون فرستنده

این برنامه در ابتدا آدرس و پورت گیرنده و همچنین آدرس فایل hex که قصد ارسالش را داریم را گرفته و پس از آن به گیرنده متصل میشود. ارسال از سه مرحله تشکیل شده است. در مرحله­ی اول متن “flashme” به عنوان یک MAGICSTRING به گیرنده ارسال میشود تا گیرنده متوجه شروع فرایند ارسال شود. پس از آن سایز فایل برنامه برای گیرنده ارسال میشود و در نهایت فایل برنامه ارسال میشود. در مرحله آخر پس از ارسال هر خط دستوری، برنامه منتظر یک رشته “OK” از گیرنده میماند. پس از پایان این مراحل گیرنده شروع به flash میکند و در صورتی که flash موفقیت آمیز بود یک “OK” دیگر برمیگرداند.

فایل hex به صورت خام ارسال نمیشود و تنها بخش دستورات ارسال میشود. ساختار فایل hex به صورت کلی شامل چندین خط رکورد است. هر رکورد شامل بخش­های زیر است:

* یک کاراکتر : برای مشخص شدن شروع رکورد
* 2 بایت برای مشخص کردن تعداد بایت داده
* 4 بایت مشخص کننده آدرسی که داده باید در آن قرار گیرد
* 2 بایت مشخص کننده نوع رکورد (در برنامه های معمولی تنها 00 و 01 مشاهده میشود و سایر انواع وجود ندارد)
* بخش داده که سایز آن بسته به تعداد بایت مشخص شده متفاوت است. در کامپایلر avr-gcc این سایز به صورت ثابت 16 در نظر گرفته میشود.
* 2 بایت Checksum

در هنگام ارسال کد برای سادگی کار در بخش گیرنده، تنها بخش داده ها جدا شده و برای گیرنده ارسال میشود.



### جزئیات کتابخانه flashlib

جهت استفاده دوباره از نتایج این پروژه در پروژه­های بعدی، بخش­های اصلی کد گیرنده در یک کتابخانه جمع آوری شده است و در بخش گیرنده (جزئیات آن بعد از این بخش توضیح داده خواهد شد) استفاده شده.

این کتابخانه شامل دو کلاس است:Processor و Flasher

کلاس Processor وظیفه دارد داده هایی که به تابع process داده میشود در EEPROM ذخیره کند. به این صورت که در ابتدا آدرس ذخیره را صفر کرده و پس از آن با هر بار ورود داده، آدرس را به اندازه سایز ورودی اضافه میکند.

void Processor::reset()

{

currentEepromAddress = EEPROM\_OFFSET\_ADDRESS;

maxEepromAddress = EEPROM\_OFFSET\_ADDRESS;

}

uint8\_t Processor::process(const uint8\_t \*buf, size\_t sz)

{

if (eeprom->write(currentEepromAddress, buf, sz) != 0)

{

return -1;

}

currentEepromAddress += sz;

if (currentEepromAddress > maxEepromAddress)

{

maxEepromAddress = currentEepromAddress;

}

return OK;

}

جهت ارتباط با EEPROM از یک کتابخانه آماده استفاده شده که به وسیله­ی پروتکل I2C عمل میکند.

کلاس Flasher وظیفه دارد کدهای ذخیره شده در EEPROM را روی آردوینو هدف از طریق پورت سریال برنامه­ریزی کند. همانطور که در بخش AVR گفته شد، امکان نوشتن روی حافظه برنامه در AVR هنگامی که در ناحیه کد user هستیم وجود ندارد و حتما باید در ناحیه بوت­لودر باشیم. به همین خاطر کلاس Flasher در ابتدای فعالیتش، پین reset آردوینو هدف را یک بار صفر و یک میکند تا آردونو هدف ریست شده و وارد بوت لودر خود شود.

void Flasher::bounce()

{

pinMode(resetPin, OUTPUT);

digitalWrite(resetPin, LOW);

delay(200);

digitalWrite(resetPin, HIGH);

delay(300);

pinMode(resetPin, INPUT);

}

بوت لودر استفاده شده در آردوینوها Optiboot نام دارد که امکان برنامه ریزی از طریق پورت سریال را فراهم میکند(این امکان به صورت پیش فرض در AVR وجود ندارد(. Optiboot پس از شروع به کار چند میلی ثانیه منتظر میماند تا ببیند پکت برنامه از پورت سریال می­آید یا نه. در صورتی که پکت وجود داشت شروع به نوشتن در حافظه برنامه AVR میکند، در غیر این صورت کد User را اجرا میکند و از بوت لودر خارج میشود.

کلاس Flasher پس از ریست کردن آردوینو هدف شروع به ارسال دستورات به Optiboot از طریق سریال میکند. در ابتدا دستور STK\_GET\_PARAMETER و بایت 0x81به آردوینو هدف ارسال میشود و در صورتی که بوت لودر آماده باشد، یک STK\_SYNC و STK\_OK بر میگرداند. به همین ترتیب دستورات زیر نیز به Optiboot ارسال میشوند تا آن را آماده دریافت برنامه کنند:

STK\_GET\_PARAMETER, 0x81

STK\_GET\_PARAMETER, 0x82

STK\_GET\_PARAMETER, 0x83

STK\_READ\_SIGN, NULL

پس گذشتن از این مراحل، دستور STK\_ENTER\_PROGMODE ارسال میشود و شروع به ارسال برنامه میکنیم.

برای ارسال برنامه، کدها باید در دسته­های 128 بایتی ارسال شوند و پس از آن که ارسال کردیم برای اطمینان از درست بودن ارسال یک دستور STK\_READ\_PAGE ارسال میکنیم تا بوت لودر هدف همان کدها را برای ما ارسال کند و ما چک کنیم که با چیزی که ارسال کردیم برابر است یا نه.

int Flasher::flash(uint16\_t startAddress, int size({

uint8\_t result = OK;

bounce();

if ((result = flashInit()) != OK){

return result;

}

if ((result=sendToOptiboot(STK\_ENTER\_PROGMODE, buffer, 0, 0))==-1){

return result;

}

uint16\_t currentAddress = startAddress;

uint16\_t endAddress = size + EEPROM\_OFFSET\_ADDRESS;

while (currentAddress < endAddress){

int len = 0;

if (endAddress - currentAddress < PROG\_PAGE\_SIZE){

len = endAddress - currentAddress;

}

else{

len = PROG\_PAGE\_SIZE;

}

if ((result = eeprom->read(currentAddress, buffer + 3, len)) != 0){

return -1;

}

addr[0] = ((currentAddress - startAddress + PROGRAM\_START\_ADDRESS)) & 0xff;

addr[1] = (((currentAddress - startAddress + PROGRAM\_START\_ADDRESS)) >> 8) & 0xff;

if ((result = sendPageToOptiboot(addr, buffer, len)) == -1){

return result;

}

currentAddress += len;

}

if ((result = sendToOptiboot(STK\_LEAVE\_PROGMODE, buffer, 0, 0)) == -1){

return result;

}

return OK;

{

int Flasher::flashInit(){

clearRead();

int dataLen = 0;

cmd\_buffer[0] = 0x81;

if ((dataLen = sendToOptiboot(STK\_GET\_PARAMETER, cmd\_buffer, 1, 1)) == -1){

return dataLen;

}

cmd\_buffer[0] = 0x82;

if ((dataLen = sendToOptiboot(STK\_GET\_PARAMETER, cmd\_buffer, 1, 1)) == -1){

return dataLen;

}

// this not a valid command. optiboot will send back 0x3 for anything it doesn't understand

cmd\_buffer[0] = 0x83;

if ((dataLen = sendToOptiboot(STK\_GET\_PARAMETER, cmd\_buffer, 1, 1)) == -1){

return dataLen;

}

else if (readBuffer[0] != 0x3){

return -1;

}

dataLen = sendToOptiboot(STK\_READ\_SIGN, NULL, 0, 3);

if (dataLen != 3)

{

return -1;

}

else if (readBuffer[0] == 0x1E && readBuffer[1] == 0x94 && readBuffer[2] == 0x6)

{

//atmega168

}

else if (readBuffer[0] == 0x1E && readBuffer[1] == 0x95 && readBuffer[2] == 0x0f)

{

//atmega328p

}

else if (readBuffer[0] == 0x1E && readBuffer[1] == 0x95 && readBuffer[2] == 0x14)

{

//atmega328

}

else

{

return -1;

}

return OK;

}

int Flasher::readOptibootReply(uint8\_t len, int timeout)

{

long start = millis();

int pos = 0;

while (millis() - start < timeout)

{

if (serial->available() > 0)

{

readBuffer[pos] = serial->read();

pos++;

if (pos == len)

{

// we've read expected len

break;

}

}

}

if (millis() - start >= timeout)

{

return -2;

}

// consume any extra

clearRead();

if (pos == len)

{

return pos;

}

else

{

return -1;

}

}

int Flasher::sendToOptiboot(uint8\_t command, uint8\_t \*arr, uint8\_t len, uint8\_t responseLength)

{

serial->write(command);

if (arr != NULL && len > 0)

{

for (int i = 0; i < len; i++)

{

serial->write(arr[i]);

}

}

serial->write(CRC\_EOP);

// make it synchronous

serial->flush();

// add 2 bytes since we always expect to get back STK\_INSYNC + STK\_OK

int replyLen = readOptibootReply(responseLength + 2, OPTIBOOT\_READ\_TIMEOUT);

if (replyLen < 2)

{

return -1;

}

if (readBuffer[0] != STK\_INSYNC)

{

return -1;

}

if (readBuffer[replyLen - 1] != STK\_OK)

{

return -1;

}

uint8\_t dataReply = replyLen - 2;

for (int i = 0; i < dataReply; i++)

{

readBuffer[i] = readBuffer[i + 1];

}

// zero the ok

readBuffer[replyLen - 1] = 0;

// return the data portion of the length

return dataReply;

}

int Flasher::sendPageToOptiboot(uint8\_t \*addr, uint8\_t \*buf, uint8\_t dataLen)

{

if (sendToOptiboot(STK\_LOAD\_ADDRESS, addr, 2, 0) == -1)

{

return -1;

}

buffer[0] = 0;

buffer[1] = dataLen;

buffer[2] = 0x46;

//remaining buffer is data

// add 3 to dataLen

if (sendToOptiboot(STK\_PROG\_PAGE, buffer, dataLen + 3, 0) == -1)

{

return -1;

}

uint8\_t replyLen = sendToOptiboot(STK\_READ\_PAGE, buffer, 3, dataLen);

if (replyLen == -1)

{

return -1;

}

if (replyLen != dataLen)

{

return -1;

}

bool verified = true;

// verify each byte written matches what is returned by bootloader

for (int i = 0; i < replyLen; i++)

{

if (readBuffer[i] != buffer[i + 3])

{

verified = false;

break;

}

}

if (!verified)

{

return -1;

}

return OK;

}

### جزئیات گیرنده

در بخش گیرنده ابتدا ماژول ESP8266 را راه اندازی میکنیم و یک Access Point میسازیم تا کسی که میخواهد آپدیت را انجام دهد به آن متصل شود. پس از آن منتظر میمانیم تا یک کلاینت به پورت TCP متصل شده و کلمه­ی “flashme” را ارسال کند. پس از گرفتن این کلمه، سایز برنامه را گرفته و بایت به بایت برنامه را به کلاس Processor از کتابخانه flashlib میدهیم تا در EEPROM ذخیره شود. پس از پایان این مرحله تابع flash از کلاس Flasher را فراخوانی میکنیم تا شروع به نوشتن برنامه در آردوینو هدف کنیم. در پایان هر بخش یک رشته “OK” به کلاینت فرستنده ارسال میشود تا از صحت آپدیتش مطمئن شود.

## روش دوم

در روش دوم پیاده­سازی OTA بجای استفاده از دو میکروکنترلر (یکی برای برنامه­ریزی و دیگری به عنوان هدف) تنها از یک میکروکنترلر استفاده میکنیم و آن میکرو خود، خود را پروگرام میکند. به این منظور از میکروکنترلر خود ماژول ESP8266 استفاده میکنیم که یک میکروکنترلر مبتنی بر معماری Xtensa است. همچنین برای پیاده سازی OTA از کتابخانه­ی ArduinoOTA که توسط خود شرکت توسعه دهنده­ی ESP ساخته شده استفاده خواهیم کرد. همچنین در این روش آپدیت از طریق یک صفحه­ی وب که به وسیله­ی ESP سرو میشود انجام میگیرد.

در ابتدا یک بار باید باید Firmware اولیه که از OTA پشتیبانی میکند را از طریق پورت سریال روی ESP برنامه ریزی میکنیم. برای این کار میتوان مستقیما از محیط توسعه­ی Arduino استفاده کرد به این شکل که در ابتدا پیکیج های مربوط به شناخت برد ESP را در بخش Board Manager نصب میکنیم و در بخش Sketch > Board نوع برد را به Generic ESP8266 Module تغییر میدهیم. با این کار امکان برنامه ریزی ESP به IDE آردوینو اضافه میشود.

حال پایه های ESP را طبق جدول زیر به Arduino متصل میکنیم:

|  |  |
| --- | --- |
| ESP8266 | Arduino |
| RX | RX |
| TX | TX |
| CH\_PD | 3.3v |
| VCC | 3.3v |
| GPIO\_0 | GND |
| GND | GND |

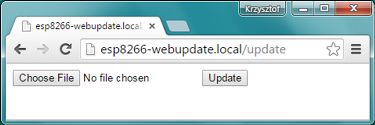
پس از آن کد ارائه شده همراه این گزارش را روی برد آپلود میکنیم. نکته­ی مهم اینجاست که قبل از شروع آپلود پین ریست را لحظه­ای به صفر و سپس به VCC متصل کنیم تا ESP وارد مود Programming بشود. پس از پایان آپلود کد روی ESP میتوان آن را از کامپیوتر جدا کرده و از آن به بعد از طریق OTA به بروزرسانی برنامه پرداخت.

برای آپلود برنامه جدید میتوان به وایفایی که ESP به آن متصل شده، متصل شد و به آدرس زیر در مرورگر رفت:

http://<<esp\_ip\_address>>/update

پس از آن از طریق فرمی که در آنجا نشان داده شده، میتوان فایل bin برنامه جدید را بارگزاری کرد.

نکته­ی مهمی که باید در نظر گرفت آن است که اگر برنامه­ای که به صورت OTA روی ESP ریخته میشود، قابلیت آپدیت OTA نداشته باشد، دیگر امکان بارگزاری از راه دور را از دست خواهیم داد و باید دوباره ماژول را به کامپیوتر متصل کرد و برنامه­ی OTA را دوباره به صورت عادی آپلود کرد.



### جزئیات کتابخانه­ی ArduinoOTA

این کتابخانه به منظور ارائه­ی قابلیت OTA در ماژول­­های ESP ساخته شده است که در آن برنامه­ی جدید گرفته شده را در حافظه بین برنامه­ی فعلی و بخش SPIFFS (که در این پروژه صفر قرار داده شده) قرار میدهد و پس از تکمیل دریافت برنامه، آن را در ROM برنامه مینویسد. پس از ریست شدن ماژول، برنامه جدید از ROM در حافظه­ی RAM قرار میگیرد و فرایند OTA تکمیل میگردد.

# مراجع

|  |  |
| --- | --- |
| [1] | W. Dargie and C. Poellabauer, Fundementals of Wireless Sensor Networks Theory and Practice, Wiley, 2010. |
| [2] | I. R. Bassey and E. P. Obot, "A Review of Wireless Sensor Networks: Applications, Challenges and Prospects in Biomedicine," *ARPN Journal of Engineering and Applied Sciences,* vol. 11, pp. 1830-1839, 2016. |
| [3] | J. N. Al-Karaki and A. E. Kamal, "Routing Techniques in Wireless Sensor Networks: A Survey," *IEEE Wireless Commun,* vol. 6, pp. 6-28, 2004. |
| [4] | K. Sohrabi, D. Minoli and T. ZNATI, Wireless Sensor Networks Technology, Protocols, and Applications, Wiley, 2007. |
| [5] | A. Kumar and S. Pahuja, "A Comparative Study of Flooding Protocol and Gossiping Protocol in WSN," *International Journal of Computer Technology & Applications,* vol. 5, pp. 797-800, 2014. |
| [6] | M. Chandna, "Comparative Analysis of Flooding and Gossiping in Wireless Sensor Networks Using SIR," *International Journal of Computer Science and Information Technologies (IJCSIT),* vol. 6, pp. 4020-4023, 2015. |
| [7] | J. Kulik, W. Hinzelman and H. Balakrishnan, "Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks," *Wireless Networks,* vol. 8, pp. 169-185, 2002. |
| [8] | C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication paradigm for Sensor Networks," in *ACM MobiCom*, Boston, 2000. |
| [9] | W. R. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," in *International Conference on System Sciences*, Hawaii, 2000. |
| [10] | M. Logambal and V. Thiagarasu, "APPLICATIONS OF WIRELESS SENSOR NETWORKS: AN OVERVIEW," *IJESRT,* 2017. |
| [11] | W. R. Heinzelman, J. Kulik and H. Balakrishnan, "Adaptive Protocole for Information Dissemination in Wireless Sensor Network," in *5th ACM/IEEE Mobicom Conference (MobiCom ’99)*, Seattle, WA, 1999. |
| [12] | C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann and F. Silva, "Directed Diffusion for Wireless Sensor Networking," *IEEE/ACM Transactions on Networking,* vol. 11, pp. 2-16, 2003. |
| [13] | D. Braginsky and D. Estrin, "Rumor Routing Algorithm For Sensor Networks," in *International Conference on Distributed Computing Systems (ICDCS’01)*, 2001. |
| [14] | I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communications Magazine,* vol. 8, pp. 104-112, 2002. |

1. Internet of Things [↑](#footnote-ref-1)
2. Sensors [↑](#footnote-ref-2)
3. Transducer [↑](#footnote-ref-3)
4. Actuator [↑](#footnote-ref-4)
5. Smart Sensor [↑](#footnote-ref-5)
6. Sensor Network [↑](#footnote-ref-6)
7. Node [↑](#footnote-ref-7)
8. Wireless Sensor Network (WSN) [↑](#footnote-ref-8)
9. Base Station [↑](#footnote-ref-9)
10. Multi-hop [↑](#footnote-ref-10)
11. Single-hop [↑](#footnote-ref-11)
12. Dynamic [↑](#footnote-ref-12)
13. Wired [↑](#footnote-ref-13)
14. Flat [↑](#footnote-ref-14)
15. Hierarchical [↑](#footnote-ref-15)
16. Query-based [↑](#footnote-ref-16)
17. Multipath-based [↑](#footnote-ref-17)
18. Negotiation-based [↑](#footnote-ref-18)
19. Coherent-based [↑](#footnote-ref-19)
20. Embedded Systems [↑](#footnote-ref-20)
21. Program Memory [↑](#footnote-ref-21)
22. Manual [↑](#footnote-ref-22)