

ATSHA و امنیت

فاز یک: پیاده­سازی یک پروتکل احراز هویت با چالش متغیر

بهار 96

چکیده

لازم است توجه داشته باشیم که در هنگام خریداری چیپ ATSHA، حتما چیپ از نوع دارای ارتباط I2C باشد، در غیر این صورت نرم­افزار پروژه فعلی قادر به برقراری ارتباط با چیپ ATSHA نخواهد بود.

فهرست مطالب

[1- تعریف پروژه 1](#_Toc486943532)

[1-1- تعریف پروژه (نام پروژه) 2](#_Toc486943533)

[1-1-1- جای دادن پروتکل احراز هویت در برنامه 2](#_Toc486943534)

[2- مفاهیم نظری 3](#_Toc486943535)

[2-1- مقدمه 4](#_Toc486943536)

[2-1-1- استفاده از ATSHA به صورت یک جعبه سیاه 4](#_Toc486943537)

[2-2- حافظههای داخلی ATSHA 4](#_Toc486943538)

[2-2-1- ناحیه داده (DZ) 6](#_Toc486943539)

[2-2-2- ناحیه پیکربندی (CZ) 6](#_Toc486943540)

[2-2-3- جمعبندی چگونگی پیکربندی نواحی CZ و DZ 11](#_Toc486943541)

[2-3- ارتباط I2C با چیپ ATSHA 12](#_Toc486943542)

[2-3-1- مفهوم CRC Checksum دادهها در چیپ ATSHA 13](#_Toc486943543)

[2-3-2- رمزکردن دادهها هنگام داد و ستد با ATSHA 13](#_Toc486943544)

[2-4- قالب استاندارد استفاده از ATSHA در یک چرخه کاری 13](#_Toc486943545)

[2-4-1- دریافت/ ارسال داده به ATSHA در چندین استفاده از پروتکل I2C 15](#_Toc486943546)

[2-4-2- آدرس پیش فرض ATSHA 16](#_Toc486943547)

[2-4-3- تایمر watch-dog چیپ ATSHA 16](#_Toc486943548)

[2-5- قالب ارسال فرمان / دریافت داده در چیپ ATSHA204A 16](#_Toc486943549)

[2-5-1- بافر ورودی/ خروجی داده در ATSHA 16](#_Toc486943550)

[2-5-2- قالب ارسال فرمان به ATSHA 17](#_Toc486943551)

[2-5-3- قالب دریافت داده از ATSHA 20](#_Toc486943552)

[2-5-4- مجموعه قوانین خواندن و نوشتن داده در بافر 21](#_Toc486943553)

[2-6- فرمانهای ارسالی به ATSHA در این پیادهسازی و پاسخ آنها 22](#_Toc486943554)

[2-6-1- بیدار کردن ATSHA 22](#_Toc486943555)

[2-6-2- فرستادن چیپ به حالت خواب/ به حالت بیکار 23](#_Toc486943556)

[2-6-3- ریست کردن شمارنده بافر دادهها 23](#_Toc486943557)

[2-6-4- خواندن داده از ATSHA (فرمان Read) 24](#_Toc486943558)

[2-6-5- نوشتن داده در ATSHA (فرمان Write) 25](#_Toc486943559)

[2-6-6- قفل کردن CZ و DZ با استفاده از فرمان Lock 27](#_Toc486943560)

[2-7- سازگاری پیادهسازی ارتباط I2C در گزارش «پروتکل ارتباطی I2C» با ATSHA 29](#_Toc486943561)

[2-7-1- سنکرونسازی ارتباط I2C در صورت از دست رفتن ارتباط 29](#_Toc486943562)

[3- الگوریتمها و پروتکلها 31](#_Toc486943563)

[3-1- مقدمه 32](#_Toc486943564)

[3-2- الگوریتم پیشنهادی برای احراز هویت 32](#_Toc486943565)

[3-2-1- انتخاب رشته XOR و الگوریتمهای بروزرسانی 34](#_Toc486943566)

[3-2-2- نقاط ضعف و قوت الگوریتم پیشنهادی 35](#_Toc486943567)

[3-3- پیادهسازی الگوریتمها 36](#_Toc486943568)

[3-3-1- نحوه استفاده از پروتکل احراز هویت در برنامه اصلی موجود در پردازنده 38](#_Toc486943569)

[3-4- سطح دوم پروتکل 47](#_Toc486943570)

[3-4-1- وضعیت راهاندازی سطح دوم 49](#_Toc486943571)

[3-4-2- وضعیت بیدار کردن ATSHA 50](#_Toc486943572)

[3-5- وضعیت بررسی بیدار شدن ATSHA 50](#_Toc486943573)

[3-6- وضعیت ارسال فرمان به ATSHA 51](#_Toc486943574)

[3-7- دیاگرام وضعیت دریافت پاسخ از ATSHA 52](#_Toc486943575)

[3-8- وضعیتهای فرستادن چیپ به حالت خواب و بیکار 53](#_Toc486943576)

[3-9- وضعیت خاتمه پروتکل سطح دو 54](#_Toc486943577)

[3-10- وضعیت نامعین، ریکاوری از وضعیت نامعین و وضعیت ریست کردن بافر ATSHA 55](#_Toc486943578)

[3-10-1- وضعیت Timeout سطح دوم 55](#_Toc486943579)

[3-11- ماژول سطح دوم پروتکل پروگرام/ احراز هویت ATSHA 56](#_Toc486943580)

[3-11-1- تابع پیادهسازی سطح دوم پروتکل (ATSHA\_PL2()) 58](#_Toc486943581)

[3-11-2- تابع تعیین عملیات بعدی سطح دوم (PL2\_Assign\_Operation()) 59](#_Toc486943582)

[4- پیادهسازی پروگرامر ATSHA (نرمافزار) 62](#_Toc486943583)

[4-1- مقدمه 63](#_Toc486943584)

[4-2- الگوریتم پروگرام 63](#_Toc486943585)

[4-2-1- دیاگرام وضعیت سطح یکم پروتکل پروگرام 64](#_Toc486943586)

[4-2-2- وضعیت نامعین در سطح اول پروتکل 68](#_Toc486943587)

[4-3- پیادهسازی نرمافزاری پروگرامر ATSHA 69](#_Toc486943588)

[4-3-1- گذار وضعیتها در پروگرامر ATSHA (تابع Change\_State()) 70](#_Toc486943589)

[4-4- پیادهسازی نرمافزار الگوریتم 72](#_Toc486943590)

[4-4-1- ماژول Main 72](#_Toc486943591)

[4-4-2- ماژول سطح یک پروتکل پروگرام ATSHA 72](#_Toc486943592)

[4-4-3- ماژول تنظیم فرمانهای ارسالی به ATSHA 74](#_Toc486943593)

[4-4-4- تابع کدکردن دادهها (Encode\_Input\_Data()) 76](#_Toc486943594)

[5- پیادهسازی الگوریتم احراز هویت (نرمافزار) 77](#_Toc486943595)

[5-1- مقدمه 78](#_Toc486943596)

[5-2- الگوریتم احراز هویت 78](#_Toc486943597)

[5-2-1- دیاگرام وضعیت سطح یکم پروتکل احراز هویت 79](#_Toc486943598)

[5-2-2- وضعیت نامعین 84](#_Toc486943599)

[5-3- پیادهسازی نرمافزاری الگوریتم احراز هویت (ماژولهای کد) 84](#_Toc486943600)

[5-3-1- ماژول سطح اول پروتکل احراز هویت ATSHA 85](#_Toc486943601)

[5-3-2- ماژول تنظیم فرمانهای ارسالی به ATSHA 91](#_Toc486943602)

[6- سختافزار 93](#_Toc486943603)

[6-1- مقدمه 94](#_Toc486943604)

[6-2- پروگرامر ATSHA 94](#_Toc486943605)

[مراجع 95](#_Toc486943606)

[پيوست – الف – کارهای آینده 96](#_Toc486943607)

[الف-1- تولید CRC برای یک بردار داده دلخواه 97](#_Toc486943608)

[الف-1-1 نحوه تولید CRC توسط Matlab 97](#_Toc486943609)

[پيوست - ب 98](#_Toc486943610)

[ب-1- مقدمه 99](#_Toc486943611)

فهرست جدول‌ها

فهرست شکل‌ها



# تعریف پروژه

## تعریف پروژه (نام پروژه)

مفهوم احراز هویت (Authentication) که در ادامه مورد استفاده می­شود به معنای اثبات وجود چیپ ATSHA مورد نظر بر روی بورد و همینطور دست نخورده و اصلی بودن برنامه مورد استفاده در پردازنده آن است.

در این پیاده­سازی نباید از استفاده از آدرس 0xFF برای ATSHA استفاده شود.

در هر استیت حلقه سنکرون باید دست کم دو تیک بمانیم.

### استفاده از چیپ ATSHA برای ایجاد قفل سخت­افزاری

اول باید از پروگرامر استفاده شود تا اطلاعات مورد نیاز برای احراز هویت

پس از تعریف متغیری از نوع command\_response\_struct (که در اینجا با نام command\_response   
نام­گذاری می­شود)، اولین بایت آرایه command که به صورت command\_response.command[0] تعریف   
می­شود در اولین مقداردهی (و قبل از اولین فراخوانی پروتکل احراز هویت) باید برابر ثابت NO\_COMMAND قرار داده شود که به معنای عدم وجود فرمان در بافر command است. این امر در پیاده­سازی اهمیت خاصی دارد و به سطح دوم اجازه نمی­دهد که در اولین استفاده از پروتکل و زمانی که هیچ فرمانی در سطح یک تعریف شده شروع به ارسال فرمان نامعینی به سمت ATSHA بکند. در اینجا لازم است یادآوری شود که برای انجام عملیات فوق لازم است هدر ATSHA\_Command\_Assignment.h نیز در ابتدای ماژول تعریف شود، چون ثابت NO\_COMMAND در این هدر تعریف می­شود.

## چگونگی انتخاب آدرس Slave در I2C برای ATSHA

## استفاده از پروگرامر ATSHA

### جای دادن پروتکل احراز هویت در برنامه

از طرف دیگر در هنگام بروزرسانی داده­ها پس از احراز هویت، الگوریتم باید تا انتها پیش برود، در غیر این صورت اگر داده­ها به صورت کامل بروزرسانی نشوند ممکن است در چرخه­های بعدی احراز هویت الگوریتم دچار مشکل شود. به همین دلیل هرگاه پرچم خروجی تابع پروتکل برابر PL1 UPDATE ATSHA بشود، نباید اجازه خروج از پروتکل را صادر کرد.

**تذکر مهم:** بعضی از وضعیت­های الگوریتم احراز هویت به نسبت به زمان زیادی برای پردازش احتیاج دارند. به این دلیل در صورت استفاده از پروتکل احراز هویت در حلقه آسنکرون این مطلب باید در نظر گرفته شود تا زمان­بندی برنامه اصلی دچار مشکل نشود.

برای استفاده از برنامه فوق تعریف سه متغیر مطابق جدول زیر ضروری می­باشد:

شکل ‏1‑1: متغیرهای مورد نیاز برای استفاده از توابع الگوریتم احراز هویت در برنامه دلخواه

|  |  |  |
| --- | --- | --- |
| نام متغیر | نوع متغیر | توضیحات |
| PL1\_output\_flag | PL1\_flags | متغیری از نوع enum، نشان دهنده  پرچم­های خروجی سطح یک پروتکل |
| PL2\_output\_flag | PL2\_flags | متغیری از نوع enum، نشان دهنده  پرچم­های خروجی سطح دو پروتکل |
| \*pcommand\_response | command\_response\_struct | پوینتر به ساختاری از نوع command\_response\_struct.  این پوینتر بدون توجه به روش پیاده­سازی باید تعریف شود. |

تنظیم ثابت ATSHA\_WD\_DURATION در هدر ATSHA\_PL2.h.

تنظیم زمان Timeout سطح دوم



# مفاهیم نظری

## مقدمه

در این فصل کلیات چیپ ATSHA شامل بخش­بندی­ حافظه­های آن، نحوه برقراری ارتباط با چیپ و استفاده از آن، و در نهایت چند فرمان مورد استفاده در این فاز پروژه مورد بررسی قرار می­گیرد.

### استفاده از ATSHA به صورت یک جعبه سیاه

بهتر است چیپ ATSHA که در یک بورد پیاده­سازی می­شود فاقد هرگونه برچسب نام قطعه باشد. به این ترتیب یک کاربر مجاز باید زمان زیادی را صرف این مطلب بکند که تشخیص دهد چیپ مجهولی که در بورد قرار گرفته چگونه چیپی است و احیاناً پی ببرد که این چیپ به عنوان یک نوع قفل سخت­افزاری در بورد مورد استفاده قرار می­گیرد. تمامی این موارد هزینه شکستن قفل سخت­افزاری بورد را بالا خواهد برد و به طور بالقوه باعث ایمن شدن سخت­افزار و نرم­افزار آن می­شود.

## حافظه­های داخلی ATSHA

قبل از شروع این بخش لازم است با چند مفهوم در ارتباط با بخش­بندی ATSHA آشنا شد:

* **تعریف اسلات (slot)**: منظور از اسلات در ATSHA معمولاً 32 بایت پشت سرهم از حافظه است.
* **تعریف بسته (block):** مفهوم بسته در ATSHA و در این گزارش معادل اسلات است و در ادامه این گزارش تنها از مفهوم اسلات استفاده خواهد شد.
* **تعریف کلمه (word):** منظور از کلمه (هنگامی که به حافظه اطلاق شود)، 4 بایت پشت سر هم داده است.

برای درک چگونگی کارکرد و استفاده بهینه از ATSHA لازم است با فضای ذخیره اطلاعات در آن آشنا شد. شکل ‏2‑1 دسته­بندی حافظه­های ATSHA را نشان می­دهد. همانگونه که مشاهده می­شود در بالاترین سطح می­توان حافظه­های ATSHA را به دو دسته EEPROM و SRAM تقسیم کرد. همانگونه که می­دانیم حافظه­های ایستا (static) مثل SRAM تنها برای ذخیره موقتی اطلاعات استفاده می­شوند. این حافظه­ در ATSHA نیز عملکرد یکسانی دارد و پس از بازگرداندن ATSHA به مد Sleep (بخش ‏2-4-) اطلاعات موجود در آن پاک می­شود (البته در مد idle اطلاعات آن دست نخورده باقی می­ماند).

حافظه EEPROM در ATSHA مطابق شکل ‏2‑1 خود به سه بخش مجزا تقسیم می­شود:

* ناحیه داده (DZ)[[1]](#footnote-1): از این ناحیه برای ذخیره کلید­ها، مدل بوردی که ATSHA بر روی آن سوار شده و یا سایر اطلاعاتی استفاده می­شود که مرتبط با میزبان است (بخش ‏2-2-1-).
* ناحیه پیکربندی (CZ)[[2]](#footnote-2): این ناحیه حاوی اطلاعات عمومی ATSHA (مثل شماره سریال، نوع پورت ارتباطی و ...) است. همچنین سطوح دسترسی DZ نیز در این ناحیه تعریف می­شود   
  (بخش ‏2-2-2-).
* ناحیه با یکبار امکان پیکربندی (OTP)[[3]](#footnote-3): از این ناحیه­ برای ذخیره اطلاعاتی استفاده می­شود که تنها یکبار قصد پروگرام آن بر روی ATSHA را داریم (اصطلاحاً داده­های فیوز مانند). قبل از قفل کردن این ناحیه، می­توان آزادانه هر گونه اطلاعاتی را بروی آن نوشت، اما پس از قفل کردن ناحیه امکان دسترسی به آن وجود ندارد. در این پروژه از ناحیه فوق استفاده نمی­شود.

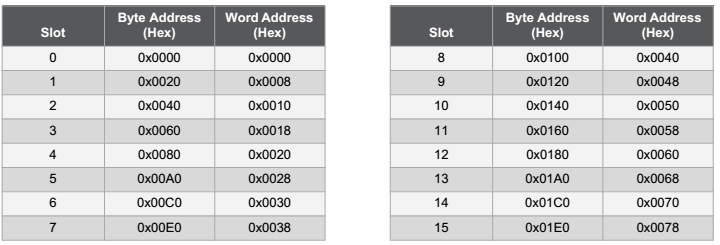
در ادامه در مورد جزئیات هر یک از نواحی بحث خواهد شد. چگونگی خواندن و نوشتن داده در این نواحی (به غیر از OTP) در بخش­های فرمان خواندن و نوشتن خواهد آمد.



شکل ‏2‑1: بخش­بندی حافظه­های داخلی ATSHA

### ناحیه داده (DZ)

مهمترین کاربرد این ناحیه، ذخیره امن داده­ها است. DZ طول 512 بایتی دارد که مطابق شکل ‏2‑2 به 16 اسلات با طول مساوی 32 بایت (8 کلمه) تقسیم­بندی می­شود. منظور از آدرس بایت (Byte Address) در شکل ‏2‑2، آدرس اولین بایت هر اسلات می­باشد. بنابراین مثلاً آدرس اولین بایت اسلات شماره 3 برابر 0x60 خواهد بود، زیرا قبل از اولین بایت این اسلات، بایت داده وجود دارد. همچنین اگر مبنای شمارش را کلمه در نظر بگیریم، آنگاه اولین بایت اسلات شماره 3 آدرس کلمه 0x18 خواهد داشت که معادل 24-امین کلمه است.

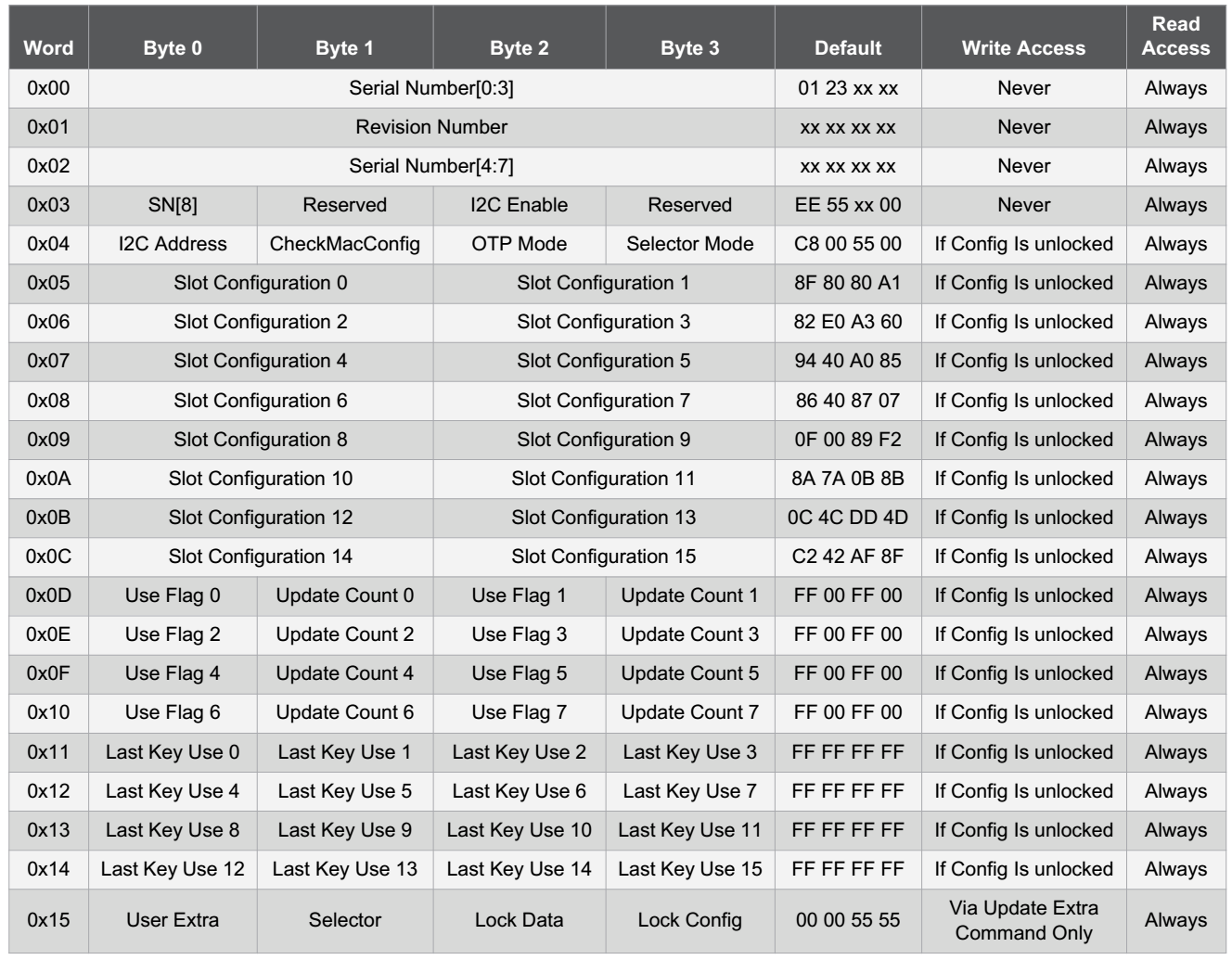


شکل ‏2‑2: پیکربندی DZ و چگونگی آدرس­دهی بخش­های مختلف آن

اولین قدمی که برای نوشتن و یا خواندن داده در این ناحیه لازم است، خارج کردن آن از قفل می­باشد. خارج کردن این ناحیه از قفل با استفاده از بیت­های قفل DZ در CZ (بایت­های دوم و سوم کلمه 0x15 از CZ، به بخش   
‏2-2-2- رجوع شود) انجام می­شود. توجه داریم امکان تعریف سطح دسترسی برای هر تک اسلات این ناحیه نیز وجود دارد. پس از خارج کردن این ناحیه از قفل، می­توان با استفاده از دستور نوشتن (بخش ‏2-6-4-) در این ناحیه داده نوشت. امکان رمزدار کردن داده نوشته شده در این ناحیه نیز وجود دارد. برای اینکار لازم است بیت تعریف سطح پیکربندی متناظر در CZ متناظر با این slot به مقدار مناسبی تنظیم شود که توضیح در این مورد در بخش بعدی و جدول ‏2‑3 آمده است.

### ناحیه پیکربندی (CZ)

چینش CZ در شکل ‏2‑3 آمده است. همانگونه که مشاهده می­شود این حافظه دارای 22 کلمه است که بعضی از کلمات تنها قابلیت خواندن دارند، اما سایر کلمات را می­توان در شرایط مناسب هم خواند و هم نوشت. مشابه CZ، خواندن و نوشتن در این ناحیه از طریق دستورهای خواندن و نوشتن انجام می­شود. در ادامه در مورد کلمات مختلف صحبت خواهد شد.



شکل ‏2‑3: چینش CZ و آدرس­ هر بایت آن در کنار قابل خواندن و نوشتن بودن هر بایت

* **کلمات 0x00 تا 0x04:** این کلمات حاوی اطلاعات کلی در مورد پیکربندی چیپ هستند. کلمات 0x00 و 0x02 حاوی شماره سریال ATSHA می­باشند که در این پیاده­سازی از آنها استفاده می­شود. بایت صفرم از کلمه پنجم، حاوی آدرس I2C این چیپ ATSHA می­باشد. برای برقراری هرگونه ارتباط با چیپ، لازم است چیپ با این آدرس در ارتباط I2C مورد خطاب بگیرد. یادآوری می­شود که بایت CheckMacConfig در خواندن و نوشتن داده به صورت رمز شده اهمیت دارد که در این گزارش از آن صرف نظر می­شود. از توضیح در مورد OTP Mode نیز به دلیل عدم استفاده از آن در این گزارش صرف نظر می­شود.
* **کلمات 0x0D تا 0x14:** در این پیاده­سازی کاربردی ندارند و از بررسی آنها صرف نظر می­شود.
* **کلمه 0x15:** 
  + **بایت اول:** محتویات این داده را می­توان پس از قفل کردن CZ نیز تغییر داد. در این پروژه از این بایت استفاده­ای نمی­شود.
  + **بایت دوم:** در این پیاده­سازی از این بایت استفاده نمی­شود.
  + **بایت سوم:** از این بایت برای قفل کردن DZ استفاده می­شود. نکته­ای که باید به آن توجه داشت این است که **تا زمانی که CZ قفل شود، امکان خواندن و یا نوشتن داده در DZ و OTP وجود ندارد.** بنابراین در هنگام پروگرام ATSHA، باید ابتدا CZ را به نحو مناسب پیکربندی و قفل نمود. پس از اینکار و با توجه به جدول ‏2‑1، نوشتن مقدار 0x55 با استفاده از دستور Lock (بخش ‏2-6-6-) در بایت LockData به این معنا خواهد بود که این ناحیه باز است و می­توان در آن داده نوشت، اما خواندن داده از آن مجاز نمی­باشد. پس از قفل کردن ناحیه نیز با توجه به سطح دسترسی تعریف شده برای هر اسلات، می­توان از آن داده خواند یا در آن نوشت. **بنابراین برای استفاده از داده­های نوشته شده در DZ پس از پروگرام داده در آن، باید LockData پس از خاتمه تنظیمات چیپ به مقداری غیر از 0x55 تنظیم شود تا بتوان داده­های موجود در DZ را پس از قفل شدن به صورت آزادانه خواند.**یادآوری می­شود پس از قفل نمودن DZ امکان باز کردن مجدد آن وجود ندارد.

**یادآوری:** قفل کردن OTP نیز با استفاده از همین بایت و با توجه به دیگر پارامترها انجام می­شود که در این گزارش از بررسی این مورد صرف نظر می­شود.

جدول ‏2‑1: مقادیر بایت LockData

|  |  |  |  |
| --- | --- | --- | --- |
| مقدار ذخیره شده | خواندن | نوشتن | توضیح |
| 0x55 | غیرمجاز | مجاز | DZ باز است. در صورت باز بودن قفل DZ، تنها  می­توان داده­های آن را تنظیم نمود.. سطوح دسترسی تعریف شده در CZ اعمال نخواهد شد. |
| هرمقداری غیر از 0x55 | مجاز | مجاز | DZ قفل است. پس از قفل کردن این ناحیه، بسته به سطح دسترسی تعریف شده در کلمات 0x05 تا 0x0C، می­توان در هر اسلات DZ داده نوشت یا از آن خواند. خارج کردن DZ از قفل ممکن نمی­باشد. سطوح دسترسی تعریف شده در CZ (در کلمات 0x05 تا 0x0C) هنگام خواندن و نوشتن در نظر گرفته خواهد شد. |

* + **بایت چهارم:** از بایت چهارم برای قفل کردن خود ناحیه CZ استفاده می­شود. همانگونه که اشاره شد پس از پیکربندی CZ باید این ناحیه قفل شود، در غیر این صورت امکان استفاده از اطلاعات آن وجود نخواهد داشت. در صورتی که این بایت مقداری غیر از 0x55 داشته باشد، خواندن و نوشتن داده در آن آزاد است. برای قفل کردن این ناحیه لازم است مقدار 0x55 با استفاده از دستور Lock (بخش   
    ‏2-6-6-) درون آن نوشته شود. پس از اینکار تنها می­توان داده­های موجود در CZ را خواند و امکان نوشتن داده در ان وجود ندارد، به همین دلیل پس از قفل نمودن DZ امکان باز کردن مجدد آن وجود ندارد. جدول ‏2‑2 این اطلاعات را خلاصه می­کند.

جدول ‏2‑2: مقادیر بایت Lock Config

|  |  |  |  |
| --- | --- | --- | --- |
| مقدار ذخیره شده | خواندن | نوشتن | توضیح |
| 0x55 | مجاز | مجاز | CZ باز است. خواندن و نوشتن تمامی داده­های موجود در CZ (غیر از مقادیری تنها خواندنی) مجاز است. |
| هر مقداری غیر از 0x55 | مجاز | غیر مجاز | CZ قفل است. امکان اصلاح داده­های موجود در CZ وجود نخواهد داشت. خارج کردن CZ از قفل ممکن نمی­باشد. |

* **کلمات 0x05 تا 0x0C:** این کلمات سطح دسترسی به هر یک از 16 اسلات DZ را تعریف می­کنند. همانگونه که در شکل ‏2‑3 می­توان مشاهده کرد، به هر یک از 16 ناحیه، یک دو بایتی اختصاص داده شده است که با استفاده از آن می­توان سطح دسترسی و همچنین کاربرد آن اسلات را تعریف نمود. مفهوم هر یک از این بایت­ها در جدول ‏2‑7 آمده است. توجه داریم هنگامی که DZ باز است، هیچکدام از محدودیت­هایی که توسط این بایت­ها تعریف می­شود به DZ اعمال نخواهد شد.تنها پس از قفل شدن DZ است که این سطوح دسترسی تعریف شده معنا پیدا خواهند کرد.

منظور از سطح دسترسی که در بالا به آن اشاره شد، چگونگی امکان دسترسی به محتویات یک اسلات است. سطوح دسترسی چندین حالت مختلف دارد:

* در یک سطح، رمزدار بودن یک اسلات تعیین می­شود. رمزدار بودن مشخص می­کند که
  + می­توان به صورت نامحدود در یک اسلات داده نوشت و از آن خواند (در این پیاده­سازی از این حالت استفاده می­شود)،
  + نوشتن داده در اسلات هیچگاه مجاز نیست،
  + خواندن و نوشتن داده در اسلات تنها پس از احراز هویت نویسنده و تنها به صورت رمز شده مجاز است.
* در سطح بعدی تعداد دفعاتی که می­توان از محتویات یک اسلات استفاده کرد مشخص می­شود.

جدول ‏2‑3: مقادیر بیت­های هر بایت SlotConfig

|  |  |  |
| --- | --- | --- |
| شماره بیت | نام | کاربرد |
| 0 – 3 | ReadKey | اسلاتی را نشان می­دهد که برای خواندن داده­ها به صورت رمز شده از آن استفاده می­شود. در این پیاده­سازی مقدار انتخاب شده برای این چهار بیت اهمیتی ندارد و در نتیجه مقدار آن برابر 0x00 انتخاب می­شود. |
| 4 | CheckOnly | صفر = می­توان از این اسلات برای تمامی فرمان­های موجود در چیپ استفاده نمود (در این پیاده­سازی از این مد استفاده می­شود).  یک = از این اسلات تنها می­توان برای CheckMac و Gendig استفاده می­شود. |
| 5 | SingleUse | صفر = هیچ حدی برای تعداد دفعات استفاده از این اسلات وجود ندارد (در این پیاده­سازی از این مد استفاده می­شود).  یک = تنها به تعدادی که توسط بایت Useflag متناظر با این اسلات تعریف می­شود می­توان از این اسلات استفاده نمود. |
| 6 | EncryptRead | صفر = خواندن داده بدون رمز کردن آن مجاز است (در این  پیاده­سازی از این مد استفاده می­شود).  یک = نیاز است که اسلات مخفی باشد و تنها خواندن رمز شده مجاز است. |
| 7 | IsSecret | این بیت مشخص می­کند که آیا نوشتن داده در این اسلات به صورت رمز شده است یا خیر.  صفر = این اسلات رمز شده نیست و خواندن داده­ها به صورت رمز نشده از آن مجاز است (در این پیاده­سازی از این مد استفاده  می­شود).  یک = این اسلات رمز شده است. تبادل داده با این اسلات تنها به صورت رمز شده مجاز است. |
| 8 – 11 | WriteKey | اسلاتی از کلید که از آن برای تایید نوشتار به صورت رمزدار شده استفاده می­شود. |
| 12 - 15 | WriteConfig | این سه بیت با توجه به فرمانی که از DZ استفاده می­کند (فرمان Derivekey و یا Write) پیکربندی می­شوند. در صورت استفاده از Write، این بایت­ها با توجه به جدول ‏2‑8 (جدول بعدی) پیکربندی می­شوند. در صورت استفاده از این فرمان (مشابه پیاده­سازی فعلی) لازم است بیت دوازدهم برابر صفر قرار داده شود.  در صورت استفاده از فرمان Derivekey باید از جدول 2-4 دیتاشیت (مرجع (1)) استفاده شود. |

جدول ‏2‑4: مقادیر مجاز بیت­های WriteConfig در مورد استفاده از فرمان Write برای این اسلات

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| بیت 13 | بیت 14 | بیت 15 | نام مد | کاربرد |
| 0 | 0 | 0 | Always  (خواندن و نوشتن بدون محدودیت) | می­توان داده­های رمز نشده را در هر زمان در این اسلات نوشت. خواندن داده به صورت 4 یا 32 بایتی از این اسلات مجاز است.  بدلیل دسترسی بدون قید توصیه شده که از این مد برای ذخیره کلید استفاده نشود.  (از این مد در پیاده­سازی این پروژه استفاده می­شود) |
| 1 | 0 | X | Never  (مجاز نبودن نوشتن) | نوشتن داده بر روی این اسلات با دستور نوشتن مجاز نخواهد بود و نمی­توان داده نوشته بر روی ان را خواند.  می­توان از اسلات­هایی که در این مد پیکربندی می­شوند برای ذخیره کلیدهای (ثابت) استفاده کرد که دسترسی به آنها از خارج از برنامه به هیچ عنوان ممکن نیست. نوشتن کلید در این حالت باید قبل از قفل کردن DZ و CZ انجام شود. |
| X | 0 | 1 | Never  (مجاز نبودن نوشتن) | مشابه مد بالایی است. |
| X | 1 | X | Encrypt  (خواندن و نوشتن پس از احراز هویت و تنها به صورت رمز شده) | برای نوشتن داده در این اسلات، نیاز است یک MAC به صورت صحیح محاسبه شود. داده­ها تنها به صورت رمز شده می­توانند در این اسلات نوشته شوند (چگونگی محاسبه این MAC در بخش 8-5-18-1 دیتاشیت مرجع (1) آمده است). |

در پیاده­سازی فعلی که رمزدار کردن داده­ها مد نظر نیست، محتویات بیت­های IsSecret و EncryptRead به صفر تنظیم می­شوند. به این ترتیب دسترسی به محتوای این دو اسلات به صورت آزادانه صورت می­پذیرد و این امکان وجود دارد که داده­های موجود در آن دو را در هر زمان به روز رسانی کرد. اما توجه داریم امنیت این روش بسیار پایین می­باشد.

**توجه:** در هنگام خواندن و یا نوشتن داده در ATSHA، باید دقت شود در صورتی از خواندن/ نوشتن با طول چهار بایت استفاده شود که تنظیمات حافظه اجازه انجام این کار را بدهد. در غیر این صورت خواندن و نوشتن چهاربایتی ممکن نیست. معمولاً خواندن با اندازه 32 بایت در تمامی حالت­ها مجاز است.

**توجه:** در پیاده­سازی­های آینده که ممکن است از فرمان Derivekey نیز استفاده شود. در این صورت باید مقادیر این بایت­ها به صورت مقتضی و بگونه­ای تنظیم شود که بتوان از هر دو فرمان Derivekey و فرمان نوشتن به صورت صحیح استفاده کرد. این مقادیر در بخش 2-1-2-14 دیتاشیت مرجع (1) آمده­اند.

### جمع­بندی چگونگی پیکربندی نواحی CZ و DZ

از آنجا که الگوریتم احراز هویت فعلی شماره سریال ATSHA را با یک عدد XOR کرده و سپس حاصل آن را با یک عدد سوم ذخیره شده در خود ATSHA مقایسه می­کند، لازم است دو داده در DZ موجود در ATSHA پیکربندی شوند (یکی مقداری که شماره سریال با آن XOR می­شود و دیگری حاصل XOR). از آنجا که ترجیح   
می­دهیم این دو مقدار ذخیره شده در DZ در هر بار احراز هویت به روزرسانی نیز بشوند و در شرایط فعلی نیز خواندن و نوشتن به صورت رمزدار انجام نمی­شود، اسلات­های مورد استفاده نیز قابلیت خواندن و نوشتن آزادانه دارند. آنچه که برای پیکربندی نواحی داده لازم است، پیکربندی و قفل کردن CZ در ابتدا و سپس پیکربندی DZ است. اگر فرض کنیم قرار است از اسلات­های صفرم و یکم برای ذخیره این دو داده استفاده شود قدم­های زیر برای پیکربندی CZ و DZ در هنگام پروگرام IC انجام می­شود:

* بایت صفرم و اول SlotConfig 0 در CZ به 0x00 تنظیم می­شوند. عملیات مشابهی برای SlotConfig 1 نیز انجام می­شود.
* بایت LockData و LockConfig با استفاده از دستور Lock به مقدار 0x00 تنظیم می­شود.
* (برای پیاده­سازی الگوریتم احراز هویت) مقدار XOR و نتیجه XOR در اسلات­های صفرم و اول DZ ذخیره می­شوند.

به این ترتیب DZ و CZ هر دو به صورت مناسب پیکربندی می­شوند و می­توان داده­های موجود در دو اسلات را به صورت دلخواه بروز کرد.

## ارتباط I2C با چیپ ATSHA

اصولاً چیپ­های ATSHA در دو قالب در بازار عرضه می­شوند: بعضی از چیپ­ها دارای پروتکل ارتباطی I2C و بعضی دیگر دارای پروتکل ارتباطی تک سیمه (SWI) هستند. اگر یک چیپ دارای پروتکل ارتباطی I2C باشد، آنگاه در بایت I2C ENABLE در CZ (بایت دوم کلمه 0x03)، بیت صفرم برابر صفر است (به جدول 2-2 مرجع (1) رجوع شود). همانگونه که در این جدول نیز اشاره شده است، بایت I2C ENABLE یک بایت تنها خواندنی می­باشد، به این ترتیب تغییر پروتکل ارتباطی چیپ ATSHA **غیر ممکن** است.

**یادآوری:** لازم به ذکر است پروتکل ارتباطی تنها یک وسیله برقراری میان ATSHA و میزبان است. نوع   
داده­هایی که میان ATSHA و میزبان جابجا می­شود ارتباطی به پروتکل ارتباطی ندارد و بنابراین در صورتی که بتوان پروتکل ارتباطی را (با خریداری چیپ مناسب و همینطور بازنویسی نرم­افزاری پروتکل ارتباطی در سمت میزبان) تغییر داد، می­توان از ارتباط SWI به جای I2C استفاده نمود. ضمناً یادآوری می­شود که سطح بندی پروتکل­های احراز هویت توسط ATSHA و پروگرام ATSHA (بخش ‏3-3-) اجازه می­دهد که به سادگی پروتکل ارتباطی را از I2C (پیاده­سازی فعلی) به SWI تغییر داد.

آدرسی که چیپ ATSHA با آن در ارتباط I2C مورد خطاب قرار می­گیرد در بایت I2C Address از CZ (بایت اول کلمه 0x04) قرار می­گیرد. هفت بیت یک تا هفت این بایت (منهای بایت صفرم) نشان دهنده آدرس افزاره در ارتباط I2C می­باشند و بیت صفرم نیز رزرو شده است. به صورت پیش فرض، مقدار قرار گرفته در این بایت برابر 0xC8 است (طبق جدول 2-2 در مرجع (1))، بنابراین در اولین باری که قرار است چیپ ATSHA پروگرام شود لازم است با این آدرس مورد خطاب قرار بگیرد. اما باید مقدار این آدرس را به یک مقدار دلخواه تغییر داد تا در صورتی که چندین چیپ ATSHA در ارتباط I2C حضور دارند، بتوان این چیپ را با آدرس مورد نظر مورد خطاب قرار داد.

توجه داریم چه در هنگام ارسال فرمان به ATSHA و چه در هنگام دریافت داده از آن:

* سمت میزبان (معمولاً میکروکنترلر) در ارتباط Master است و پالس ساعت و سیگنال آغاز ارتباط را فراهم می­کند. بنابراین میزبان در هنگام ارسال داده در مد Master Transmit و در هنگام دریافت داده در مد Master Receive فعالیت می­کند.
* چیپ ATSHA در ارتباط Slave است و قادر به فراهم­سازی پالس ساعت I2C نیست. بنابراین در هنگام دریافت فرمان از میزبان در مد Slave Receive و در هنگام ارسال نتایج به آن در مد Slave Transmit فعالیت می­کند.

**یادآوری:** جدول زیر نمایش اختصاری مدهای ارتباطی در I2C را طبق گزارش «*پروتکل ارتباطی I2C*» نمایش می­دهد.

جدول ‏2‑5: مدهای ارتباطی I2C

|  |  |  |
| --- | --- | --- |
| نام وضعیت | مخفف | توضیح |
| I2C\_MT\_MODE | MT | ارتباط برای فعالیت در مد MT تنظیم می­شود |
| I2C\_MR\_MODE | MR | ارتباط برای فعالیت در مد MR تنظیم می­شود |
| I2C\_SR\_MODE | SR | ارتباط برای فعالیت در مد SR تنظیم می­شود |
| I2C\_ST\_MODE | ST | ارتباط برای فعالیت در مد ST تنظیم می­شود |

### مفهوم CRC Checksum داده­ها در چیپ ATSHA

روش CRC Checksum یک روش بررسی یکپارچگی داده است که با استفاده از آن می­توان بررسی نمود که آیا داده­ای که توسط ارتباط I2C دریافت شده صحیح می­باشد یا خیر. به صورت خلاصه این روش به این صورت عمل می­کند که باقی­مانده حاصل تقسیم داده­های دریافتی در ارتباط I2C به یک مقدار از قبل مشخص شده محاسبه   
می­شود و در کنار داده­های اصلی به سمت گیرنده ارسال می­شود. گیرنده نیز با استفاده از داده­های دریافتی خود مجدداً باقی­مانده تقسیم را محاسبه می­کند و آن را با باقی­مانده دریافتی مقایسه می­کند. در صورتی که حاصل این دو مقدار برابر باشد، نتیجه می­شود که داده­ها بدون مشکل بر روی خط جابجا شده­اند. در چیپ ATSHA، مقدار مقسوم علیه این تقسیم برابر 0x8005 انتخاب می­شود. برای بررسی جزئیات بیشتر این الگوریتم به گزارش   
«*پروتکل ارتباطی I2C*» رجوع شود.

### رمزکردن داده­ها هنگام داد و ستد با ATSHA

یکی از قابلیت­های ATSHA، امکان رمز کردن داده­ها هنگام ارسال آن به ATSHA است. این امکان باعث می­شود که یک شنونده غیر مجاز نتواند به محتوای واقعی داده­ها دسترسی پیدا کند. رمز کردن داده­ها به این صورت است که در سمت میزبان داده با یک مقدار مشخص XOR می­شود که دقیقاً همین مقدار در TempKey (که در حافظه SRAM چیپ ATSHA تعریف می­شود) نیز وجود دارد. در سمت ATSHA، چیپ خود این قابلیت را دارد که در صورت انتخاب فرمان Read صحیح، داده­ها را مجدداً با مقدار ذخیره شده در TempKey به صورت منطقی XOR کند تا داده اصلی بدست آید. در نهایت داده اصلی در ATSHA ذخیره می­شود.

در این پیاده­سازی، داده­های ارسالی به ATSHA رمز نمی­شوند، اما رمز کردن داده­ها در پیاده­سازی­های آینده باید مد نظر قرار بگیرد. توضیحات در مورد چگونگی رمز کردن داده­ها در بخش 8-5-18-1 دیتاشیت ATSHA (مرجع (1)) آمده است.

## قالب استاندارد استفاده از ATSHA در یک چرخه کاری

**یادآوری:** منظور از یک فریم ارتباط I2C در ادامه این گزارش، یک سیگنال آغاز ارتباط I2C، بایت آدرس، بایت(های) داده و در نهایت سیگنال توقف است.

منظور از یک چرخه کاری در چیپ ATSHA، بیدار کردن چیپ، ارسال و دریافت (بعضاً چندین باره) داده (فرمان) به آن و در نهایت بازگرداندن آن به حالت بیکار است. به این ترتیب یک احراز هویت می­تواند شامل چندین چرخه­کاری باشد. چیپ ATSHA بگونه­ای طراحی شده است که می­توان در پایان کار با آن در یک چرخه کاری، آن را به حالت بیکار (idle) و یا خواب (sleep) بازگرداند. بنابراین هر چرخه کاری برقراری ارتباط با چیپ ATSHA بدون توجه به عملیاتی که قرار است در آن چرخه انجام شود (خواندن داده از چیپ، نوشتن داده در آن، انجام یک عملیات مشخص) مشابه شکل ‏2‑4 شامل چهار مرحله است:



شکل ‏2‑4: چرخه­کاری برقراری ارتباط با ATSHA تا بازگرداندن آن به حالت خواب/ بیکار

1. **بیدار کردن چیپ ATSHA:** برای بیدار کردن چیپ از حالت خواب/ بیکار کافیست تنها یک بایت 0x00 توسط ارتباط I2C به گونه­ای به چیپ ارسال شود که ATSHA برای مدت زمانی بیش از *منطق* صفر را بر روی خط مشاهده کند. مقدار *تقریباً* برابر می­باشد و بنابراین فرکانس کاری ارتباط I2C نیز باید به صورت متناسب تنظیم شود (بخش ‏2-6-1-). پس از بیدار کردن چیپ، ATSHA در قالب بسته­های استاندارد چهاربایتی یک بایت 0x11 به بیرون ارسال می­کند تا نشان بدهد آماده اجرای عملیات است. (به بخش ‏2-6-1- رجوع شود).
2. **ارسال فرمان به ATSHA:** ارسال فرمان به چیپ توسط فریم ارتباط I2C انجام می­شود. برای اینکار مشابه شکل ‏2‑5، ابتدا یک سیگنال آغاز ارتباط I2C به چیپ ارسال می­شود، سپس شماره چیپ به آن ارسال می­شود، پس از آن بایت آدرس کلمه (Word Address Byte) که حاوی یکی از چهار فرمان کلی است که می­توان به چیپ داد (دو فرمان خوابیدن و مد بیکار، دو فرمان محل نوشتن داده در بافر که در ادامه در مورد آن در بخش ‏2-5-2- بحث خواهد شد) به چیپ ارسال می­شود و در کنار آن در صورت لزوم چندین بیت اضافی دیگر حاوی داده ارسال می­شوند، و در نهایت نیز سیگنال توقف ارتباط I2C به چیپ ارسال می­شود. پس از اتمام این چرخه­کاری ارتباط I2C، چیپ ATSHA شروع به انجام عملیات خواسته شده از آن می­کند و تا زمانی که مشغول به انجام عملیات باشد، آدرس خود را NACK خواهد کرد. پس از خاتمه ارسال فرمان به ATSHA، میزبان منتظر می­ماند تا چیپ عملیات مربوطه را انجام دهد. نرم زمانی مورد نیاز برای اجرای یک فرمان با معین می­شود. شرح کامل الگوی ارسال فرمان به ATSHA در بخش ‏2-5-2- آمده است.

**یادآوری:** نیازی به ارسال مجدد سیگنال آغاز ارتباط I2C پس از اولین بیدار کردن چیپ ATSHA نیست. در واقع به محض بیدار کردن چیپ می­توان اولین فرمان را به آن ارسال نمود و چیپ تا زمان WD (بخش ‏2-4-3-) خود بیدار خواهد ماند.



شکل ‏2‑5: ارسال یک فرمان به ATSHA

1. **دریافت پاسخ از ATSHA:** پس از توقف به اندازه ، می­توان نتیجه فرمان اجرا شده در چیپ را در خروجی ارتباط I2C چیپ دریافت نمود. دریافت نتیجه فرمان با تعویض مد ارتباطی I2C در سمت میزبان به دریافت در حالت Master (I2C\_MR\_MODE) ممکن خواهد شد. نتیجه فرمان دست کم یک بایت و بیشینه 32 بایت است. علاوه بر این تمامی فرمان­ها شامل یک بایت در ابتدا (بایت count یا تعداد بایت ارسالی در خروجی چیپ) و دو بایت اضافی در انتها (بایت­های CRC) می­باشند. بنابراین طول داده خروجی ATSHA در پاسخ به یک فرمان دست کم 4 بایت و بیشینه 35 بایت خواهد شد. شرح کامل الگوی دریافت داده از ATSHA در بخش ‏2-5-3- آمده است.
2. **بازگرداندن چیپ به حالت خواب/ بیکار:** آخرین مرحله در یک چرخه­کاری، بازگرداندن چیپ به حالت بیکار یا خواب می­باشد. این قسمت مشابه ارسال یک فرمان عادی به ATSHA می­باشد و جزئیات آن در بخش ‏2-6-2-آمده است.

علاوه بر موارد کلی که در مورد چیپ ATSHA در بالا به آن اشاره شد، چند نکته دیگر نیز در مورد استفاده از چیپ ATSHA اهمیت دارد که در زیر بخش­های پیش رو در مورد آن بحث خواهد شد.

### دریافت/ ارسال داده به ATSHA در چندین استفاده از پروتکل I2C

یکی از خواص جالب چیپ، قابلیت ارسال و دریافت داده به آن در چندین تکرار پروتکل I2C است. این خاصیت اجازه می­دهد که اگر قرار است N بایت به چیپ ارسال شود/ از آن خوانده شود، برای هر بایت ابتدا یکبار ارتباط I2C برقرار شود و بایت ارسال شود/ دریافت شود، و پس از آن ارتباط I2C متوقف شود؛ سپس برای بایت بعدی نیز همین چرخه تکرار شود. توجه داریم عدد N باید هم در مد ارسال و هم در مد دریافت برای ATSHA مشخص شود تا هیچگونه ابهامی پس از ارسال سیگنال توقف ارتباط برای آن ایجاد نشود. در این پیاده­سازی پروتکل، هر خواندن و نوشتن به صورت بایت به بایت صورت می­پذیرد. دلیل دیگری که ترجیح می­دهیم خواندن و نوشتن را به صورت بایت به بایت انجام دهیم این است که در صورت دریافت NACK از سمت ATSHA، تنها بایت فعلی مجدداً به سمت آن ارسال می­شود و نیازی به ریست کردن بافر و ارسال مجدد تمامی داده­ها نخواهد بود.

### آدرس پیش فرض ATSHA

به صورت پیش­فرض، آدرس ATSHA برابر 0xC8 است و این آدرس در بایت صفرم از کلمه 0x04 ناحیه پیکربندی CZ قرار دارد. در صورتی که تغییر این آدرس پیش فرض مطلوب باشد، لازم است محتوای این بایت عوض بشود که برای انجام آن لازم است از دستور نوشتن (Write) بر روی این ناحیه استفاده شود. در این پیاده­سازی اما این آدرس دستخوش تغییر نخواهد شد.

### تایمر watch-dog چیپ ATSHA

به منظور افزایش امنیت چیپ ATSHA و عدم امکان دسترسی به اطلاعات آن بیش از یک زمان معین، چیپ دارای یک سازوکار ریست Watch-Dog (WD) است. در صورتی که مجموع زمان کار با چیپ ATSHA از زمان بیشتر شود، چیپ به صورت خودکار وارد مد خواب خود می­شود و تمامی وضعیت­ها و اطلاعات فرّار داخل چیپ ریست می­شوند. در بهترین حالت باید پیاده­سازی بگونه­ای باشد که تمامی فرمان­ها قبل از *انجام شوند، اما راه­حل دیگر استفاده از حالت* Idle است. در حالت Idle، داده­های موجود در SRAM یعنی   
RNG seed و TempKey نگاه داشته می­شوند. طبق جدول 7-2 از مرجع (1)، کمترین زمان برای *برابر 700 میلی­ثانیه است، بنابراین لازم است تمامی­ عملیات­های احراز هویت در زمانی به حد قابل قبول کمتر از این مقدار صورت بپذیرند.*

با توجه به اهمیت نگاه­داری زمان WD در هنگام پیاده­سازی، استفاده از الگوهای استاندارد نگاه­داری زمان مانند آنچه در گزارش «*استانداردسازی نوشتار کد*» در شرکت صانیک آمده است اهمیت دارد. در واقع لازم است زمان اجرای هر وضعیت ذخیره شود و دقت شود که مجموع این زمان­ها از *تجاوز نکند. توجه داریم این مورد به خصوص زمانی اهمیت دارد که عملیات احراز هویت (در نسخه­های آینده) بسیار پیچیده می­شود.*

***توجه:*** همانگونه که در ادامه اشاره خواهد شد (بخش ...)، برای جلوگیری از پیچیده­ شدن بیش از حد کد   
نرم­افزار، در پیاده­سازی پروتکل احراز هویت و همینطور پروگرام ATSHA (و به طور خاص در سطح دوم آن. به بخش ‏3-3- رجوع شود) در هر چرخه­ کاری چیپ تنها یک فرمان به چیپ ارسال می­شود و نتیجه آن از ATSHA دریافت می­شود. این عمل باعث می­شود که مطمئن باشیم که هیچگاه انجام یک عملیات از زمان WD چیپ بیشتر نخواهد شد.

## قالب ارسال فرمان / دریافت داده در چیپ ATSHA204A

مهمترین بحثی که در ارتباط با ATSHA مطرح است، قالب ارسال فرمان به چیپ و همینطور چگونگی خواندن نتایج از آن است. در این بخش نشان داده می­شود که یک فرمان در یک چرخه­کاری در چه قالبی به ATSHA ارسال می­شود، و همینطور داده­های دریافتی از آن چه قالبی دارند. قبل از توضیح این قالب­ها، لازم است در مورد بافر ورودی/ خروجی داده در ATSHA204A بحث شود.

### بافر ورودی/ خروجی داده در ATSHA

تمامی داده­های ورودی/ خروجی به چیپ در یک بافر نوشته می­شوند که باعث می­شود دسترسی به داده­ها حتی پس از اجرای یک فرمان ممکن باشد. شماتیک انتزاعی این بافر در شکل ‏2‑6 نشان داده شده است.



شکل ‏2‑6: تجسم بافر ورودی/ خروجی داده در ATSHA204A

همانگونه که در این شکل مشاهده می­شود، طول بافر داده برابر 88 بایت است. می­توان فرض کرد بافر یک شمارنده نیز دارد (که در دیتاشیت IC با نام Buffer Counter از آن یاد می­شود) و مشخص می­کند داده قرار است در کدام بایت بافر نوشته شود یا خوانده شود. در یک تصور انتزاعی، هنگام ارسال یک فرمان از سمت میزبان (میکروکنترلر) به ATSHA، اولین بایت در نزدیکترین مکان به سمت ATSHA قرار می­گیرد و داده­ها نیز به همین ترتیب در سمت ATSHA خوانده می­شوند. برعکس در هنگام ارسال پاسخ از سمت ATSHA به سمت میزبان، اولین بایت در نزدیک­ترین مکان به سمت میزبان قرار می­گیرد. بنابراین بافر به صورت FIFO است. از طرف دیگر پس اجرای یک عملیات در سمت ATSHA، داده­های ارسالی به سمت میزبان در این بافر باقی می­مانند. برای خواندن مجدد این داده­ها پس از اتمام یک خواندن، کافیست ابتدا ارتباط I2C با ATSHA برقرار شود، سپس فرمان RESET (بخش ‏2-6-3-) به ATSHA ارسال شود و پس از آن ارتباط I2C قطع شود. با انجام اینکار شمارنده بافر به ابتدای آن بازخواهد گشت. پس از آن با برقراری مجدد ارتباط I2C، داده­ها با شروع از اولین بایت مجدداً به سمت میزبان ارسال می­شوند. البته با انجام همین عملیات نیز می­توان نوشتن را نیز ابتدای بافر آغاز کرد. توجه داریم اگر طول یک فرمان ارسالی به ATSHA بیش از 88 بایتِ طول بافر باشد، چیپ پس از ارسال هر بایت پس از بایت 88، یک NACK باز می­گرداند و داده­ای در بافر نوشته نخواهد شد. این مسئله باید در هنگام ارسال فرمان به چیپ رعایت شود. همچنین نمی­توان بیش از 88 بایت داده از چیپ ATSHA دریافت کرد، و تلاش برای خواندن تعداد بایت بیشتر داده:

1. باعث نمی­شود چیپ به ابتدای بافر بازگشته و خواندن را مجدداً از آنجا آغاز نماید.
2. با هربار ارسال فرمان خواندن، چیپ عدد 0xFF را باز می­گرداند.

نکته­ای که باید در اینجا متذکر شد این است که پس از هر بار خواندن داده از بافر، در فرمان نوشتن بعدی در ATSHA، داده­ها مجدداً از ابتدای بافر نوشته می­شوند. بنابراین پس از اتمام یک چرخه نیازی به ارسال فرمان RESET و بازگشت به ابتدای بافر نیست. مجموعه قوانینی که باید در هنگام نوشتن/ خواندن داده در بافر رعایت شود در بخش ‏2-5-4- آمده است.

### قالب ارسال فرمان به ATSHA

قالب ارسال فرمان به سمت ATSHA در یک چرخه­کاری در شکل ‏2‑7 آمده است.



شکل ‏2‑7: قالب استاندارد ارسال اطلاعات به ATSHA

همانطور که مشاهده می­شود پس از برقراری ارتباط I2C در مد I2C MT Mode، اولین بایتی که به چیپ ارسال می­شود فرمان و یا آدرس کلمه است. شرح آدرس کلمه­ها در جدول ‏2‑6 آمده است. همانگونه که مشاهده   
می­شود، آدرس کلمه به صورت کلی سه مطلب را تعیین می­کند:

1. **فرمان­های Sleep و idle**: فرمان حالت خواب/ بیکار به ATSHA ارسال می­شود،
2. **فرمان Reset:** بافر داده ریست بشود،
3. **فرمان Command:** (قرار است) فرمان به چیپ ارسال شود.

در مورد فرمان Reset در بخش قبلی بحث شد و اشاره شد در صورتی که هدف خواندن مجدد داده از ابتدای بافر باشد، یک فرمان Reset به چیپ ارسال می­شود. خواندن مجدد داده زمانی مفید است که چیپ اعلان کند که دستور مورد نظر با موفقیت اجرا شده است، اما پس از بررسی CRC داده دریافتی در خروجی مشخص شود داده اشتباه دریافت شده است. برای بازگرداندن چیپ به مدهای خواب و بیکار نیز مشابه Reset تنها کافیست آدرس کلمه متناظر به چیپ ارسال شده و سپس ارتباط قطع شود. چگونگی استفاده از سه کلمه آدرس فوق در بخش­های   
‏2-6-2 و ‏2-6-3 آمده است.

به صورت کلی برای اجرای یک فرمان جدید، کلمه آدرس Commnad به چیپ ارسال می­شود. بنابراین هرگاه قرار است ATSHA یک عملیات به غیر از سه عملیات اشاره شده در بالا را انجام بدهد لازم است Command به آن ارسال شود. پس از ارسال Command می­توان Opcodeهای مختلف متناظر با فرمان­های مورد نظر را به ATSHA فرستاد. شرح این فرمان­ها در بخش 8-5-2 دیتاشیت مرجع (1) آمده است.

جدول ‏2‑6: آدرس کلمات تعریف شده برای ATSHA204A

|  |  |  |
| --- | --- | --- |
| آدرس کلمه | مقدار | توصیف |
| Reset | 0x00 | شمارنده آدرس بافر ورودی خروجی ریست می­شود. خواندن و نوشتن بعدی از ابتدای بافر I/O آغاز می­شود. |
| Sleep  (مد خواب) | 0x01 | ATSHA وارد مد خواب می­شود. مصرف توان در این مد پایین است، تا زمان ارسال فرمان بیدار شدن چیپ پاسخی به ورودی­ها نمی­دهد. تمامی  وضعیت­های فرّار (یعنی داده­های SRAM) ریست می­شوند. |
| Idle mode  (مد بیکار) | 0x02 | ATSHA وارد وضعیت بیکار می­شود. تا زمان ارسال فرمان بیدار شدن چیپ پاسخی به ورودی­ها نمی­دهد. محتوای TemKey و RNG نگه داشته می­شود. |
| Command  (مد فرمان) | 0x03 | مد عادی کارکرد ATSHA است. بایت­هایی که در ادامه خواهند یک فرمان به ATSHA هستند و به ترتیب در بافر نوشته خواهند شد. |
| ----------- | 0x04 – 0xFF | رزرو شده است. |

شرح بیت­های بعدی تا قبل از Checksum در جدول ‏2‑7 آمده است. توجه داریم محاسبه دقیق بایت Count (که چگونگی آن در این جدول آمده است) از اهمیت خاصی برخوردار است، زیرا همانگونه که اشاره شد در صورتی که مقدار این بایت با طول بایت­های فرمان ارسالی برابر نباشد، ATSHA بایت­های بیشتر را NACK خواهد کرد. بیت­ها از بیت سوم (Opcode) تا بیت دو تا مانده آخر (DataN) در واقع قالب استاندارد ارسال فرمان به ATSHA هستند. قالب استاندارد فرمان­ها در بخش 8 دیتاشیت IC (مرجع (1)) آمده است. ضمن اینکه فرمان­هایی که قرار است در این پیاده­سازی به سمت ATSHA فرستاده شوند در بخش ‏2-6- مورد بررسی قرار می­گیرند. توجه داریم در صورت ارسال فرمان بیدار شدن، خوابیدن و یا حالت بیکار نیازی به ارسال این بایت­ها نیست (به بخش­های ‏2-6-1- و ‏2-6-2- رجوع شود).

جدول ‏2‑7: بایت­های فرمان ارسالی به ATSHA

|  |  |
| --- | --- |
| اسم بایت | کارکرد |
| Count (تعداد بایت) | مشخص می­کند که به غیر از کلمه آدرس چند بایت قرار است به ATSHA ارسال شود. تعداد بایت از رابطه زیر بدست می­آید:  تعداد بایت = یک بایت برای Count + تعداد بایت اشغال شده از بایت Opcode تا DataN + دو بایت CRC check. |
| Opcode | عملیاتی که قرار است در چرخه فعلی انجام شود |
| Param1 | اولین پارامتر فرمان. |
| Param2  (Param 2-1 و Param 2-2) | دومین پارامتر فرمان. شامل دو بایت است. |
| Data | داده­ای که لازم است در کنار بعضی فرمان­ها ارسال شود (مثل داده­ای که نیاز است در ATSHA نوشته شود). |

آخرین دو بایت ارسالی به ATSHA، دو بایت­ Checksum هستند. این دو بایت کمک می­کنند که ATSHA بتواند صحت داده دریافتی بر روی خط ارتباطی را بررسی کند. در صورتی که این بررسی وجود یک مشکل را اعلان کند، یک بایت خطا در یک قالب چهار بایتی به سمت میزبان بازگردانده خواهد شد (به بخش بعدی رجوع شود). CRC از بایت Count به بعد محاسبه می­شود.

**یادآوری:** همانگونه که در بخش قبل اشاره شد، بیشینه طول دستورهایی که به ATSHA فرستاده می­شود باید کمتر و یا مساوی 88 بایت باشد. با بررسی تمامی فرمان­های ممکنی که می­توان به ATSHA فرستاد (که تمامی فرمان­ها در بخش 8 دیتاشیت IC موجود هستند) مشخص می­شود که طول تمامی فرمان­ها کمتر و یا مساوی این مقدار است. بنابراین در صورت ارسال تنها یک فرمان به ATSHA در یک چرخه­کاری، مطمئن خواهیم بود که طول داده­های نوشته شده در بافر ATSHA از حد مجاز تجاوز نمی­کند. تنها در صورتی که قرار باشد چند فرمان به صورت همزمان به چیپ فرستاده شود باید طول بافر باید به عنوان یک عامل موثر در نظر گرفته شود.

### قالب دریافت داده از ATSHA

پس از ارسال یک فرمان به ATSHA، نوبت به دریافت پاسخ از چیپ می­رسد. دریافت پاسخ و به عبارت دیگر خواندن داده از بافر ورودی/ خروجی چیپ با تغییر مدکاری I2C به I2C MR Mode ممکن می­شود[[4]](#footnote-4). شکل ‏2‑8 نشان دهنده قالب داده­های دریافتی از ATSHA در هر بار خواندن داده از آن است.



شکل ‏2‑8: قالب استاندارد دریافت اطلاعات از ATSHA

همانگونه که در شکل مشاهده می­شود اولین بایت دریافتی در یک بسته داده از ATSHA همیشه طول آن بسته داده است. به صورت کلی مقدار این بایت به صورت زیر معین می­شود:

|  |  |
| --- | --- |
| (‏2‑1) | یک بایت برای Count + یک بایت برای Status/Error + N بایت داده + 2 بایت Checksum Count = |

توجه داریم چهار بایت Count، Status/Error و Checksum همیشه در هر خواندن داده از چیپ وجود دارند. بنابراین . به عبارت دیگر، خروجی همیشه دست کم برابر چهار است و بیشینه برابر طول بافر ورودی/ خروجی است.

بایت دوم دریافتی از چیپ که با نام خطا/ وضعیت (Error/status) شناخته می­شود نشان دهنده وضعیت فعلی چیپ است. جدول ‏2‑8 مقادیر ممکن برای بایت وضعیت/ خطا را مشخص می­کنند. همانگونه که محتوای جدول نیز نشان می­دهد، این بیت مشخص می­کند که آیا عملیات متناظر با فرمان فعلی به درستی انجام پذیرفته است یا خیر، و همچنین اینکه فرمان دریافتی توسط چیپ در مرحله قبل به دلیل مشکلات در ارتباط I2C به صورت صحیحی دریافت شده است یا خیر (توسط بررسی CRC). توجه داریم تنها در صورت دریافت 0x00، دریافت بیت­های   
باقی­مانده معنادار است و در غیر از آن باید مجدداً فرمان قبلی به ATSHA ارسال شود.

جدول ‏2‑8: مقادیر ممکن بایت وضعیت/ خطای ATSHA

|  |  |  |
| --- | --- | --- |
| وضعیت | مقدار | توصیف |
| اجرای موفق عملیات | 0x00 | عملیات خواسته شده از ATSHA به صورت موفق انجام شده است. |
| مشکل در Checkmac | 0x01 | فرمان Check به صورت صحیحی به چیپ فرستاده شد، اما پاسخ دریافتی از ATSHA با مقدار مورد نظر تفاوت دارد[[5]](#footnote-5). |
| خطای تجزیه و تحلیل | 0x03 | ATSHA به درستی فرمان را دریافت کرده، اما طول آن، مقدار فرمان، یا بعضی پارامترها دارای مشکل بوده­اند. لازم است فرمان مجدداً ارسال شود. |
| خطای اجرا | 0x0F | ATSHA به درستی فرمان را دریافت کرده، اما در شرایط فعلی چیپ قادر به اجرای آن نیست. نیاز است تغییری در وضعیت چیپ و یا مقدار فرمان انجام شود. |
| بیدار شدن | 0x11 | چیپ بیدار شده است. |
| خطا در مخابره داده­ها (با محاسبه CRC) | 0xFF | فرمان به دلیل مشکلات مخابراتی به صورت صحیحی دریافت نشده است و مجدداً باید ارسال شود. |
| ---------- | سایر مقادیر | سایر مقادیر غیر مجاز هستند. |

آخرین دو بایت دریافتی در قالب بسته داده، بایت­های Checksum هستند. با استفاده از این بایت­ها می­توان مشخص کرد که آیا بسته داده دریافتی از ATSHA یکپارچه می­باشد و یا خیر، که این کار با محاسبه Checksum از بقیه داده­ها در سمت میزبان و مقایسه آن با این دو بایت انجام می­شود. در صورتی که مقایسه نشان دهنده وجود یک خطا باشد، با استفاده از فرمان Reset از ATSHA درخواست می­شود که دوباره داده­ها را ارسال کند، یا اینکه می­توان تمامی چرخه ارسال فرمان و دریافت پاسخ را مجدداً تکرار نمود. یادآوری می­شود که CRC در این حالت از تمامی­ بایت­های دریافتی محاسبه خواهد شد.

### مجموعه قوانین خواندن و نوشتن داده در بافر

به عنوان جمع­بندی، مجموعه قوانینی که در ادامه خواهند آمد باید در هنگام ارسال فرمان به چیپ/ خواندن داده از آن رعایت شوند:

1. طول داده­های دریافتی از چیپ دست کم 4 بایت و بیشینه 35 بایت است و قالب آنها مشابه شکل ‏2‑8 می­باشد. بنابراین هیچگاه نباید بیش از 32 بایت داده از یکی از نواحی داده خوانده شود.
2. تنها در سه موقعیت می­توان داده از بافر خواند:

* پس از بیدار شدن، داده 0x11 در قالب 4 بایتی به بیرون ارسال می­شود.
* اگر در زمان اجرای یک دستور مشکلی در کارکرد چیپ به وجود بیاید، چیپ یک بایت خطا در قالب چهاربیتی باز می­گرداند.
* پس از اتمام اجرای یک عملیات، 1 تا 32 بایت را می­توان در قالب 4 تا 35 بیتی قرائت کرد.

1. داده­های ورودی/ خروجی همیشه به ترتیب در اولین بایت بافر قرار می­گیرند. بنابراین خواندن آنها در سمت میزبان نیز از اولین بایت خواهد بود (نیازی به هیچگونه عملیات اضافی برای بازگرداندن شمارنده به ابتدای بافر نیست).
2. امکان قرائت مجدد داده­های موجود در بافر با استفاده از فرمان Reset وجود دارد.
3. در صورتی که فرمان ارسالی به بافر بیش از 88 بایت باشد، بایت­های اضافی دریافت نخواهند شد، زیرا بافر قابلیت بازگشت به ابتدا را ندارد. بنابراین طول فرمان همیشه باید کمتر مساوی 88 بایت باشد. در مورد خواندن داده از ATSHA اما از آنجا که طول داده دریافتی همیشه کمتر از 35 بایت است مشکلی وجود ندارد.
4. اولین نوشتن داده بعد از خواندن داده از ATSHA از ابتدای بافر است، بنابراین قبل از نوشتن مجدد نیازی به بازگرداندن بافر به ابتدا نیست.

## فرمان­های ارسالی به ATSHA در این پیاده­سازی و پاسخ آنها

در این پیاده­سازی تنها از چند فرمان پایه چیپ استفاده می­شود که شرح آنها در ادامه خواهند آمد.

**یادآوری:** یکی از راههای افزایش سرعت اجرای برنامه­ها، انجام بیشینه محاسبات ممکن به صورت آفلاین می­باشد. بر این اساس ترجیح می­دهیم در جایی که تمامی بایت­های فرمانی که قرار است به ATSHA فرستاده شوند مشخص است، بایت­های CRC را از قبل محاسبه کرده و در حافظه ذخیره کنیم. پیوست الف چگونگی محاسبه CRC را برای یک چند جمله­ای دلخواه با استفاده از Matlab نمایش می­دهد.

### بیدار کردن ATSHA

برای بیدار کردن چیپ ATSHA باید ارتباط I2C در مد MT تنظیم شده و سپس یک بایت 0x00 به صورتی بر روی خط ارسال شود که طول یک بیت صفر آن از *بیشتر باشد. این سیگنال با نام* WAKE TOKEN *شناخته می­شود.* همانگونه که در دیتاشیت IC و جدول 7-2 (مرجع (1)) آمده است، مقدار *باید دست کم برابر باشد. از طرف دیگر ارتباط فرکانس کاری* I2C *(یا ) با فرکانس کاری پردازنده () به صورت زیر است:*

|  |  |
| --- | --- |
| (‏2‑2) |  |

که و دو رجیستر تنظیم سرعت ارتباط I2C می­باشند (به گزارش «*پروتکل ارتباطی I2C*» رجوع شود). بنابراین اگر با حاشیه اطمینان، *و به عبارتی نصف مقدار مطلوب باشد، آنگاه:*

|  |  |
| --- | --- |
| (‏2‑3) |  |

اگر بیشینه فرکانس کاری میکروکنترلر AVR را برابر 8MHz در نظر بگیریم[[6]](#footnote-6)، آنگاه خواهیم داشت:

|  |  |
| --- | --- |
| (‏2‑4) |  |

برای ارسال WAKE TOKEN به سمت IC، بعد از ارسال سیگنال آغاز ارتباط I2C آدرس 0x0F بر روی خط ارسال می­شود که دارای چهار بیت صفر است. همانگونه که در گزارش «*پروتکل ارتباطی I2C*» نیز شرح داده شده است، میان بایت­های آدرس و بایت داده تفاوتی در ارتباط وجود ندارد، بنابراین ارسال این آدرس توسط میزبان در سمت ATSHA و در زمانی که IC در حالت خواب/ بیکار است نه به عنوان آدرس بلکه به عنوان WAKE TOKEN تلقی شده و در نتیجه IC آماده به کار خواهد شد.

پس از ارسال WAKE TOKEN به سمت ATSHA و پس از مدتی به اندازه *،* چیپ در قالب استاندارد چهار بایتی و مشابه جدول ‏2‑9 عدد 0x11 را به بیرون می­فرستد تا اعلان کند آماده کار می­باشد. در قبل از این مدت اما چیپ آدرس خود در ارتباط I2C را NACK خواهد کرد.

جدول ‏2‑9: پاسخ دریافتی از چیپ پس از بیدار کردن آن

|  |  |  |  |
| --- | --- | --- | --- |
| Checksum2 | Checksum1 | Status/Error | Count |
| 0x55 | 0x14 | 0x11 | 0x04 |

### فرستادن چیپ به حالت خواب/ به حالت بیکار

برای فرستادن چیپ به حالت خواب، میزبان I2C در مد MT فرمان 0x01 را به سمت آن ارسال می­کند و در نهایت ارتباط را متوقف می­کند. در این حالت و پس از طی زمان مقتضی، چیپ به حالت خواب فرو می­رود. برای فرستادن چیپ به حالت بیکار نیز تنها کافیست به جای عدد 0x01، عدد 0x02 به چیپ ارسال شود. این فرمان­ها به ترتیب در جدول ‏2‑10 و جدول ‏2‑11 نشان داده شده است.

جدول ‏2‑10: فرمان فرستادن چیپ به حالت خواب

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Checksum2 | Checksum1 | Param2 | | Param1  (انتخاب ناحیه) | Opcode | Count | آدرس کلمه |
| بایت یک (Param21) | بایت صفر  (Param20) |
| ---- | | ---- | | ---- | ---- | ---- | 0x01 |

جدول ‏2‑11: فرمان فرستادن چیپ به حالت بیکار

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Checksum2 | Checksum1 | Param2 | | Param1  (انتخاب ناحیه) | Opcode | Count | آدرس کلمه |
| بایت یک (Param21) | بایت صفر  (Param20) |
| ---- | | ---- | | ---- | ---- | ---- | 0x02 |

### ریست کردن شمارنده بافر داده­ها

در صورتی که به هر دلیلی نیاز باشد که شمارنده بافر داده­ها ریست شود، کافیست کلمه آدرس 0x00 توسط یک بایت به ATSHA ارسال شود. این فرمان در جدول ‏2‑12 نشان داده شده است. پس از پایان اینکار شمارنده بافر ریست شده است و می­توان داده را از ابتدای بافر خواند و یا از ابتدای آن نوشت.

جدول ‏2‑12: فرمان ریست کردن شمارنده بافر داده­ها

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Checksum2 | Checksum1 | Param2 | | Param1  (انتخاب ناحیه) | Opcode | Count | آدرس کلمه |
| بایت یک (Param21) | بایت صفر  (Param20) |
| ---- | | ---- | | ---- | ---- | ---- | 0x02 |

### خواندن داده از ATSHA (فرمان Read)

برای خواندن داده از یکی از سه ناحیه داده ATSHA، لازم است فرمان خواندن که مطابق جدول ‏2‑13 تعریف می­شود به ATSHA ارسال شود. همانگونه که مشاهده می­شود مقادیر Count و Opcode در این دستور ثابت و به ترتیب برابر 0x07 و 0x02 انتخاب می­شوند. پس از تخصیص مقدار به این دو بایت لازم است مقادیر Param1 و Param2 انتخاب شود که در ادامه در مورد چگونگی انجام آن بحث خواهد شد. خروجی این فرمان بسته به طول خواندن مورد نظر (4 بایت یا 32 بایت)، 8 و یا 36 بایت می­باشد.

جدول ‏2‑13: فرمان خواندن داده از یکی از نواحی ATSHA

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Checksum2 | Checksum1 | Param2 | | Param1  (انتخاب ناحیه) | Opcode | Count | آدرس  کلمه |
| بایت یک (Param21) | بایت صفر  (Param20) |
| CRC از بایت Count به بعد محاسبه می­شود. | | برمبنای شماره کلمه­ای که قرار است خوانده شود | | طبق جدول ‏2‑14 و جدول ‏2‑15 | 0x02 | 0x07 | 0x03 |

* **نحوه انتخاب Param1:**

جدول ‏2‑14 و جدول ‏2‑15 نحوه انتخاب Param1 را جهت ارسال فرمان مشخص می­کنند. همانگونه که در این جدول نشان داده شده است، این پارامتر ناحیه مورد نظر و همینطور اندازه خواندن (4 یا 32 بایت) را تعیین   
می­کند. اصولاً برای دریافت داده از ATSHA، لازم است قفل بودن ناحیه و همینطور امکان خواندن داده از آن مورد بررسی قرار بگیرد. اما از آنجا که در این پروژه تنها قرار است از DZ داده خوانده شود و خواندن از این حافظه بدون محدودیت و همینطور رمزنگاری است، نیازی به لحاظ کردن سایر ملاحظات نیست. اما در پروژه­های بعدی لازم است کلیه ملاحظات خواندن از سایر نواحی در نظر گرفته شود.

جدول ‏2‑14: شرح بیت­های Param1 فرمان خواندن

|  |  |
| --- | --- |
| شماره بیت | کاربرد |
| بیت­های صفر و یک | انتخاب از میان یکی از سه ناحیه داده (جدول ‏2‑15) |
| بیت­های 2 - 6 | باید برابر صفر باشد |
| بیت 7 | اگر یک باشد، 32 بایت داده خوانده خواهد شد، در غیر این صورت 4 بایت خوانده می­شود. خواندن داده از OTP تنها به صورت چهاربایتی است |

جدول ‏2‑15: شرح بیت­ صفر و یک Param1 فرمان خواندن   
(با استفاده از اطلاعات ارائه شده در بخش­ 8-5-3 دیتاشیت ATSHA در مرجع (1))

|  |  |  |
| --- | --- | --- |
| نام ناحیه | مقدار | تبصره­های خواندن داده از ATSHA (به بخش ‏2-2- رجوع شود) |
| CZ | 0 | بدون توجه به قفل بودن ناحیه امکان خواندن داده از آن وجود دارد. |
| OTP | 1 | در این گزارش مورد بررسی قرار نمی­گیرد. |
| Data | 2 | اگر CZ قفل نشده است امکان خواندن وجود ندارد (خطا بازگردانده می­شود)  اگر CZ قفل باشد به پیکربندی که برای هر اسلات در CZ انجام می­شود بستگی دارد. |

* **نحوه انتخاب Param2:**

|  |  |
| --- | --- |
| Param2 | |
| بایت یک (Param21) | بایت صفر  (Param20) |
| همیشه برابر 0x00 انتخاب می­شود | متناظر با شماره اولین کلمه (با شروع از 0x00) |

شکل پارامتر دوم فرمان (که همانگونه که می­دانیم یک دو بایتی است) بسته به اندازه خواندن (چهار یا 32 بایتی) آدرس کلمه/ اسلاتی را مشخص می­کند که قصد داریم از آن داده بخوانیم (به بخش 8-5-4 دیتاشیت مرجع (1) رجوع شود). توجه داریم بایت یکم این دو بایتی باید همیشه برابر صفر (0x00) انتخاب شود. قانون انتخاب بایت صفرم نیز بسیار ساده می­باشد. در واقع این هشت­بایتی نشان دهنده شماره کلمه­ای از یک ناحیه است که قصد داریم از آن داده بخوانیم. بنابراین به عنوان مثال اگر قصد داریم 13-امین کلمه از CZ را بخوانیم، باید مقدار این بایت را برابر 0x0C قرار بدهیم (و نه 0x0D چون آدرس اولین کلمه CZ برابر 0x00 است!) و الی آخر. در مورد DZ نیز اتفاق کاملا مشابهی می­افتد. به عنوان مثال اگر قرار است اسلات سوم ناحیه در یک قالب 32 بایتی خوانده شود، از آنجا که باید نوشتن را از کلمه 32 آغاز کنیم، مقدار این بایت برابر 0x20 خواهد شد. همچنین اگر قرار است کلمه پنجم از اسلات دوازدهم خوانده شود، این آدرس برابر 0x65 خواهد شد.

* **پاسخ به فرمان:**

پس اجرای فرمان، ATSHA در قالب استاندارد اشاره شده در بخش ‏2-5-3- یک بسته داده از بافر دریافت خواهد کرد که به صورت نمایش داده شده در جدول ‏2‑16 خواهد بود.

جدول ‏2‑16: پاسخ دریافتی از ATSHA بعد از فرستادن فرمان خواندن

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Checksum2 | Checksum1 | Data | Status/Error | Count |
| به داده دریافتی بستگی دارد | | بسته به طول خواندن 4 یا 32 بایت | 0x00 | 0x08 اگر چهار بایت خوانده شود/ 0x24 اگر 32 بایت خوانده شود |

### نوشتن داده در ATSHA (فرمان Write)

برای نوشتن داده در یکی از سه ناحیه داده ATSHA، لازم است فرمان خواندن که مطابق جدول ‏2‑17 تعریف می­شود به ATSHA ارسال شود. همانگونه که مشاهده می­شود مقدار Opcode در این دستور 0x12 انتخاب می­شود، اما مقدار Count به طول بقیه فرمان بستگی دارد و در این پیاده­سازی (به دلیل نوشتن­های 32 بایتی) مقدار آن برابر 39 (0x27) انتخاب می­شود. پس از تخصیص مقدار به این دو بایت لازم است مقادیر Param1 و Param2 انتخاب شود که در ادامه در مورد چگونگی انجام آن بحث خواهد شد. پس از آن مجموعه بایت Data1 مشخص   
می­کند که چه داده­هایی قرار است در ناحیه مورد نظر نوشته شوند. طول این مجموعه یا 4 بایت و یا 32 بایت خواهد بود. ضمن اینکه در صورتی که قرار است داده به صورت رمزشده در ATSHA نوشته شود لازم است این بایت­ها در سمت میزبان رمز شده باشند. همچنین در صورتی که قرار است داده به صورت رمز شده نوشته شود، Data2 یک مقدار 32 بایتی خواهد داشت که چگونگی ساخت آن در بخش 8-5-18-1 دیتاشیت IC (مرجع (1)) آمده است.

جدول ‏2‑17: فرمان نوشتن داده در یکی از نواحی ATSHA

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Checksum2 | Checksum1 | Data2 | | | Data1 | | | Param2 | | Param1  (انتخاب ناحیه) | Opcode | Count | آدرس  کلمه |
| Data2N | … | Data20 | Data1N | … | Data10 | Param21  بایت یک | Param20  بایت صفر |
| CRC از بایت Count به بعد محاسبه می­شود. | | (صفر یا 32 بایت)  MAC برای تایید آدرس و داده[[7]](#footnote-8) | | | (4 یا 32 بایت)  داده­ای که قرار است در یک ناحیه نوشته شود[[8]](#footnote-9) | | | برمبنای شماره  کلمه­ای که قرار است داده در آن نوشته شود | | طبق جدول جدول ‏2‑18 و جدول ‏2‑19 | 0x12 | با توجه به جدول ‏2‑7 | 0x03 |

* **نحوه انتخاب Param1:**

جدول ‏2‑18 و جدول ‏2‑19 نحوه انتخاب Param1 را جهت ارسال فرمان مشخص می­کنند. همانگونه که در این جدول نشان داده شده است، این پارامتر ناحیه مورد نظر و همینطور اندازه نوشتن (4 یا 32 بایت) را تعیین   
می­کند. اصولاً برای نوشتن داده در ATSHA، لازم است قفل بودن ناحیه و همینطور امکان نوشتن داده بر روی آن مورد بررسی قرار بگیرد. در این پروژه نوشتن داده بر روی DZ بدون محدودیت صورت می­پذیرد، اما در پروژه­های بعدی لازم است کلیه ملاحظات نوشتن داده در نظر گرفته شود.

جدول ‏2‑18: شرح بیت­های Param1 فرمان نوشتن

|  |  |
| --- | --- |
| شماره بیت | کاربرد |
| بیت­های صفر و یک | انتخاب از میان یکی از سه ناحیه داده (جدول ‏2‑15) |
| بیت­های 2 - 5 | باید برابر صفر باشد |
| بیت 6 | یک = داده­ای که قرار است نوشته شود باید رمز شده باشد  صفر = اگر DZ / OTP قفل شده­اند، باید صفر باشد. |
| بیت 7 | اگر یک باشد، 32 بایت داده نوشته خواهد شد، در غیر این صورت 4 بایت نوشته می­شود. |

جدول ‏2‑19: شرح بیت­ صفر و یک Param1 فرمان نوشتن   
(با استفاده از اطلاعات ارائه شده در بخش­ 8-5-3 دیتاشیت ATSHA در مرجع (1))

|  |  |  |
| --- | --- | --- |
| نام ناحیه | مقدار | تبصره­های نوشتن داده بر روی ATSHA (به بخش ‏2-2- رجوع شود) |
| CZ | 0 | تنها در زمان قفل نبودن CZ امکان نوشتن داده وجود دارد.  پس از قفل شدن ناحیه امکان نوشتن داده وجود ندارد.  نوشتن داده به صورت رمز شده معنا ندارد و مجاز نیست. |
| OTP | 1 | در این گزارش مورد بررسی قرار نمی­گیرد. |
| Data | 2 | اگر CZ قفل نشده است امکان نوشتن وجود ندارد (خطا بازگردانده می­شود)  اگر CZ قفل باشد و DZ قفل نباشد می­توان در تمامی اسلات­های DZ داده نوشت.  اگر CZ قفل باشد و DZ نیز قفل باشد، خواندن و نوشتن داده به SlotConfiguration بستگی دارد (به جدول ‏2‑4 رجوع شود). |

* **نحوه انتخاب Param2:**

نحوه انتخاب Param2 برای نوشتن داده در ATSHA مشابه نحوه انتخاب این پارامتر برای خواندن داده است و بنابراین از توضیح آن صرف نظر می­شود.

* **پاسخ به فرمان:**

پس اجرای فرمان، ATSHA در قالب استاندارد اشاره شده در بخش ‏2-5-3- یک بسته داده از بافر دریافت خواهد کرد که به صورت نمایش داده شده در جدول ‏2‑20 خواهد بود.

جدول ‏2‑20: پاسخ دریافتی از ATSHA بعد از فرستادن فرمان نوشتن

|  |  |  |  |
| --- | --- | --- | --- |
| Checksum2 | Checksum1 | Status/Error | Count |
| به داده دریافتی بستگی دارد | | 0x00 | 0x04 |

### قفل کردن CZ و DZ با استفاده از فرمان Lock

برای قفل کردن نواحی DZ، OTP و CZ از فرمان قفل کردن (Lock) استفاده می­کند. این فرمان بایت­های LockData و LockConfig در آخرین کلمه CZ را به صفر تنظیم می­کند و به این ترتیب باعث قفل شدن این دو ناحیه می­شود. توجه داریم اگر این نواحی از قبل قفل شده باشند، آنگاه ATSHA یک خطا باز خواهد گرداند. پس از تنظیم این دو بایت، نوبت به انتخاب Param1 و Param2 می­رسد. قبل از توضیح در مورد نحوه انتخاب این دو پارامتر لازم است یادآوری شود که از آنجا که پس از قفل کردن یک ناحیه داده امکان باز کردن مجدد آن وجود ندارد، ATSHA سازوکاری را در نظر گرفته است که به وسیله آن می­توان صحت داده­ای که در هنگام پروگرام چیپ ATSHA در یک ناحیه داده نوشته شده است را بررسی کرد؛ به این ترتیب می­توان اطمینان حاصل کرد که داده نوشته شده در چیپ مقدار صحیحی دارد. این سازوکار عبارتست: از محاسبه CRC یک ناحیه. در واقع اگر بیت هفتم Param1 در این برابر صفر قرار داده شود، آنگاه لازم است CRC داده­هایی که در داخل ATSHA پروگرام شده است در Param2 این فرمان فراهم شده و به سمت ATSHA ارسال شود. پس از ارسال این CRC، ATSHA خود CRC داده­های موجود در ناحیه مورد نظر را محاسبه کرده و با مقدار ارسالی مقایسه می­کند. در صورتی که این دو مقدار برابر باشند، IC ناحیه مورد نظر را قفل خواهد کرد، در غیر این صورت یک خطا باز خواهد گرداند. توجه داریم در صورتی که بیت هفتم برابر یک انتخاب شود، آنگاه Param2 مقدار 0x0000 را اختیار می­کند[[9]](#footnote-10). پس اجرای فرمان، ATSHA در قالب استاندارد اشاره شده در بخش ‏2-5-3- یک بسته داده از بافر دریافت خواهد کرد که به صورت نمایش داده شده در جدول ‏2‑20 خواهد بود.

**توجه:**  در این پیاده­سازی از بررسی CRC در هنگام فقل کردن نواحی داده صرف نظر می­شود، اما لازم است در پیاده­سازی­های آینده محاسبه CRC مورد توجه قرار بگیرد.

جدول ‏2‑21: قفل کردن نواحی داده ATSHA

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Checksum2 | Checksum1 | Param2 | | Param1  (انتخاب ناحیه) | Opcode | Count | آدرس  کلمه |
| Param21  بایت یک | Param20  بایت صفر |
| CRC از بایت Count به بعد محاسبه می­شود. | | دو بایت CRC که با توجه به بیت 7 از Param1 و جدول ‏2‑19 انتخاب می­شوند. | | جدول ‏2‑18 | 0x17 | 0x07 | 0x03 |

جدول ‏2‑22: شرح بیت­های Param1 فرمان قفل کردن

|  |  |
| --- | --- |
| شماره بیت | کاربرد |
| بیت­ صفر | صفر برای CZ، یک برای DZ و OTP. |
| بیت­های 1 - 6 | باید برابر صفر باشد. |
| بیت 7 | اگر یک باشد، آنگاه ناحیه بدون بررسی CRC قفل می­شود. در غیر این صورت باید CRC نیز به صورت توضیح داده شده در جدول ‏2‑19 فراهم شود. |

جدول ‏2‑23: شرح بایت­های Param2 فرمان قفل کردن

|  |  |  |
| --- | --- | --- |
| مقدار بیت هفت از Param1 | نام ناحیه | چگونگی محاسبه |
| اگر یک باشد | ------- | در هر دو بایت مقدار 0x00 قرار داده می­شود. |
| اگر صفر باشد | CZ | CRC با شروع از کلمه صفرم برای تمامی 88 بایت داده محاسبه می­شود. |
| DZ و OTP | داده­های DZ و سپس OTP به ترتیب در کنار یکدیگر قرار می­گیرند (DZ در ابتدا) و سپس CRC محاسبه می­شود[[10]](#footnote-11). |

جدول ‏2‑24: پاسخ دریافتی از ATSHA بعد از فرستادن فرمان قفل کردن

|  |  |  |  |
| --- | --- | --- | --- |
| Checksum2 | Checksum1 | Status/Error | Count |
| به داده دریافتی بستگی دارد | | 0x00 | 0x04 |

## سازگاری پیاده­سازی ارتباط I2C در گزارش «*پروتکل ارتباطی I2C*» با ATSHA

بنابراین به صورت کلی پیاده­سازی باید بگونه­ای باشد که در صورتی که ATSHA دچار هر یک از مشکلات زیر شود، بتوان آخرین عملیات درخواستی از آن را مجدداً و بدون ایجاد مشکل در پیوستگی الگوریتم انجام داد (یعنی انجام مجدد عملیات باعث نشود داده­ای از دست برود و یا از یک قدم الگوریتم احراز هویت به اشتباه عبور شود).

1. ATSHA به بیدار شدن خود پاسخ ندهد:
2. داده ارسالی به ATSHA دارای مشکل باشد: در صورتی که فرمان ارسالی به ATSHA دارای مشکل باشد، لازم است مجدداً تمامی فرمان به چیپ ارسال شود. در این حالت چون یک پاسخ از سمت ATSHA دریافت شده و در نتیجه بافر داده به ابتدای آن بازگشته است، تنها کافیست فرمان قبلی مجدداً به سمت ATSHA ارسال شود. در صورتی که پس از N بار ارسال مجدد فرمان همچنان خطایی توسط Status/Error دریافت شود، پروتکل ریست شده و سپس الگوریتم احراز هویت مجدداً از ابتدا آغاز می­شود (به شرط اینکه این آغاز مجدد هیچگونه مشکلی را در نسخه­های آینده پیاده­سازی ایجاد نکند).
3. داده دریافتی از ATSHA دارای مشکل باشد: همانگونه که در بخش ... اشاره شد، در این حالت تنها کافیست یک فرمان Reset به سمت چیپ ارسال شود و داده­ها مجدداً خوانده شوند.
4. چرخه ارسال داده به ATSHA به دلیل مشکل در ارتباط I2C در میان کار متوقف شود:
5. چرخه دریافت داده از ATSHA به دلیل مشکل در ارتباط I2C در میان کار متوقف شود:
6. ارتباط I2C دچار یک وضعیت نامعین شود: توجه داریم در این حالت ارتباط I2C همچنان برقرار است و ارسال و دریافت داده­ها به انجام می­پذیرد، اما (معمولاً به دلیل وجود نویز در پردازنده) ارتباط در یک وضعیت نامشخص قرار گرفته است. در این حالت (با ایجاد یک تغییر در پروتکل I2C)، ابتدا یک پرچم به بیرون می­فرستیم که نشان­دهنده وجود اشکال در ارتباط است. سپس با متوقف و سپس شروع کردن مجدد ارتباط یک فرمان Reset به چیپ ATSHA می­فرستیم. حال مطمئن خواهیم شده که به ابتدای بافر داده بازگشته­ایم، و در نتیجه مجدداً می­توان ارسال فرمان. خواندن داده از چیپ را آغاز نمود.
7. ATSHA به ارتباط I2C پاسخ ندهد: در این حالت وضعیت پروتکل I2C بعد از طی یک بیشینه زمان معین از حالت مشغول به حالت آماده تغییر نمی­کند. این مفهوم که آن را از دست رفتن سنکرون­سازی ارتباط I2C میان دو چیپ می­نامیم دارای اهمیت خاصی است و به صورت جداگانه در یک زیربخش در ادامه مورد بررسی قرار خواهد گرفت.
8. مدت زمان انجام یک عملیات پس از چندبار تکرار آن به دلیل انجام نشدن عملیات در چرخه­های قبلی از مقدار WD فراتر برود:

### سنکرون­سازی ارتباط I2C در صورت از دست رفتن ارتباط

یکی از مشکلاتی که ممکن است در هنگام کار با ATSHA ایجاد شود، از دست رفتن ارتباط I2C با چیپ به هر دلیلی (شامل نویز و ...) است. دو مسئله در اینجا مطرح است:

1. چگونه می­توان تشخیص داد ارتباط I2C از دست رفته است؟
2. در صورت تشخیص از دست رفتن ارتباط چه عملی باید انجام شود؟

پاسخ به این دو سوال در ادامه خواهد آمد.

#### تشخیص از دست رفتن ارتباط I2C

همانگونه که در بخش

#### عملیاتی که پس از تشخیص از دست رفتن ارتباط I2C انجام می­پذیرد

توجه داریم باید زمان کافی برای ریست کردن هم موجود باشد. اگر نباشد باید اول چیپ را به حالت Idle بفرستیم بعد. ولی جالبه ارتباط را از دست دادیم کلاً! چه کنیم اینجا!

چند بار که ریست بشه، کلاً ارتباط با چیپ در کماست (به احتمال زیاد چیپ برداشته شده است). بنابراین یک حمله تشخیص داده می­شود و یک عملیات انجام می­شود.



# الگوریتم­ها و پروتکل­ها

## مقدمه

برای استفاده از ATSHA در راستای احراز هویت نیازمند تعریف یک الگوریتم احراز هویت   
(Authentication Algorithm) هستیم. در این فصل در مورد جنبه­های نظری این الگوریتم بحث و نقاط قوت و ضعف آن بحث می­شود. از آنجا که برای پیاده­سازی هرگونه الگوریتم احراز هویت به کمک ATSHA لازم است ابتدا اطلاعاتی نیز در چیپ ATSHA (و قبل از اولین استفاده) پروگرام شود، در این فصل در مورد نحوه پیاده­سازی یک الگوریتم پروگرام که منجر به پیاده­سازی پروگرامر ATSHA می­شود نیز بحث خواهد شد.

مهمترین جنبه­ای که در مورد پیاده­سازی الگوریتم­ها باید در نظر گرفته شود، نحوه پیاده­سازی آن به صورت نرم­افزاری و هماهنگ­سازی ارتباط I2C، چرخه­کاری و کلیت الگوریتم­های پروگرام و احراز هویت است. در این فصل نشان داده می­شود که چگونه الگوریتم­های احراز هویت و پروگرام ATSHA به سطح مختلف شکسته می­شود و نقش هر سطح در کنار توضیح کامل سطح دوم و همینطور نحوه پیاده­سازی آن در این فصل شرح داده می­شود.

## الگوریتم پیشنهادی برای احراز هویت

الگوریتم احراز هویت پیشنهادی در فاز یک این پروژه را می­توان یک الگوریتم با چالش متغیر در نظر گرفت (زیرا محتوای سوالی که از مشتری پرسیده می­شود در تکرارهای مختلف احراز هویت تغییر می­کند، ) که نامتقارن است (زیرا هیچگونه کلیدی در سمت میزبان ذخیره نمی­شود). به صورت کلی الگوریتم فوق به این صورت است که یک رشته (که در این فاز آن را رشته XOR می­نامیم) در ATSHA ذخیره شده و سپس حاصل XOR این رشته با شماره سریال در یک قسمت دیگر ATSHA ذخیره می­شود. در هر بار فراخوانی الگوریتم احراز هویت، میزبان شماره سریال و رشته XOR را از ATSHA می­خواند و حاصل XOR این دو رشته را با مقدار نوشته شده در ATSHA مقایسه می­کند. در صورتی که حاصل این دو مقدار برابر باشد نتیجه می­شود که میزبان به درستی احراز هویت شده است و چیپ به کار خود ادامه خواهد داد. پس از هر احراز هویت، رشته XOR با استفاده از یک الگوریتم خاص بروزرسانی می­شود و همینطور حاصل XOR نیز پس از محاسبه در سمت میزبان مجدداً در ATSHA ذخیره خواهد شد. برای پیچیده­تر کردن امر احراز هویت:

1. رشته شماره سریال پس از انجام بعضی عملیات (جابجایی 4 بیت LSB و MSB هر بایت و NOT کردن حاصل نهایی) رمز شده و سپس XOR خواهد شد.
2. ترتیب خواندن رشته XOR و حاصل XOR در کد تصادفی است.
3. یک رشته انحرافی نیز در ATSHA ذخیره می­شود که میزبان در هر تلاش برای احراز هویت آن را از ATSHA می­خواند، اما به محتویات آن توجهی نخواهد کرد و آن را نیز پس از اتمام احراز هویت به روز رسانی خواهد کرد. پیش و پس از فرستادن رشته XOR، چندین رشته انحرافی مختلف به سمت ATSHA ارسال می­شود.

برای پیاده­سازی الگوریتم فوق لازم است ابتدا ATSHA پروگرام شود که الگوریتم پروگرام ATSHA در شکل ‏3‑1 آمده است. همانگونه که مشاهده می­شود برای پروگرام ATSHA لازم است دو رشته داده در حافظه امن آن (در اینجا DZ) ذخیره شوند که یکی رشته XOR و دیگری حاصل XOR است. ضمن اینکه رشته انحرافی نیز در DZ ذخیره خواهد شد. پیاده­سازی همراه با جزئیات الگوریتم پروگرام ATSHA در فصل چهارم آمده است.

الگوریتم احراز هویت توسط یک ATSHA که از قبل پروگرام شده است در شکل ‏3‑2 آمده است. همانگونه که مشاهده می­شود اولین قدم در راه احراز هویت، خواندن رشته انحرافی از سمت ATSHA است. میزبان پس از دریافت این رشته آن را کنار می­گذارد. سپس نوبت به دریافت شماره سریال ATSHA، رشته XOR و همینطور حاصل XOR از ATSHA می­رسد که همانگونه که اشاره شد برای پیچیده­تر کردن الگوریتم، خواندن دو رشته­ آخر با ترتیب تصادفی انجام می­شود. در پایان خواندن این سه رشته داده، سمت میزبان با انجام عملیات مناسب حاصل XOR را محاسبه کرده و با مقدار دریافتی از ATSHA مقایسه می­کند. در صورتی که این دو مقدار برابر باشند احراز هویت موفق تلقی شده و سپس رشته XOR و همینطور حاصل XOR با الگوریتمی که در ادامه خواهد آمد بروزرسانی می­شوند و مجدداً در ATSHA ذخیره خواهند شد که ترتیب ارسال رشته نیز به صورت تصادفی است و در میان آنها چندین رشته انحرافی فرستاده می­شود تا الگوریتم را پیچیده­تر کند. در صورت عدم موفقیت احراز هویت اما یک پرچم عدم موفقیت احراز هویت به بیرون بازگردانده خواهد شد. جزئیات دقیق­تر پیاده­­سازی الگوریتم احراز هویت در فصل پنجم آمده است.



شکل ‏3‑1: الگوریتم پروگرام ATSHA



شکل ‏3‑2: الگوریتم احراز هویت توسط ATSHA (پس از پروگرام آن)

### انتخاب رشته XOR و الگوریتم­های بروزرسانی

همانگونه که در ادامه اشاره خواهد شد برای ساده­سازی پیاده­سازی الگوریتم، به جای خواندن تنها شماره سریال ATSHA ترجیح بر خواندن یک رشته 32 بیتی از CZ است که حاوی شماره سریال   
و بعضی اطلاعات دیگر نیز می­باشد. این امر باعث می­شود جابجایی اطلاعات با ATSHA راحت­تر شود و پروتکل درگیر خواندن چند باره بایت­های مختلف شماره سریال از قسمت­های مختلف CZ نیز نشود. در ادامه تعاریف زیر را فرض می­کنیم (منظور از رمز کردن یک رشته در ادامه، جابجا کردن چهار بیت MSB و LSB هر بایت رشته و NOT کردن حاصل است):

1. X: رشته XOR که شماره سریال رمز شده با آن XOR می­شود.
2. Xc: رشته XOR رمز شده.
3. S: رشته شماره سریال.
4. Sc: رشته شماره سریال رمز شده.
5. R: رشته حاصل XOR شماره سریال و رشته XOR
6. Rc: رشته رمز شده حاصل XOR شماره سریال و رشته XOR

از میان رشته­های فوق، همیشه رشته Xc در سمت ATSHA ذخیره می­شود، رشته S در داخل ATSHA وجود دارد و همینطور رشته Rc نیز در ATSHA ذخیره می­شود. سایر رشته­ها نیز در سمت میزبان ATSHA مورد استفاده قرار خواهند گرفت.

در پروگرامر ATSHA، اولین رشته X در مبنای 16 به صورت زیر انتخاب شده است:

|  |  |
| --- | --- |
| (‏3‑1) | X= |

بنابراین اولین رشته Xc که در ATSHA ذخیره می­شود از رمز کردن رشته فوق حاصل خواهد شد. سپس رشته R نیز با استفاده از رشته S که از ATSHA خوانده می­شود ساخته شده و سپس رشته Rc در ATSHA ذخیره خواهد شد. در هر بار استفاده از الگوریتم احراز هویت، رشته­های Xc، S و Rc از ATSHA خوانده شده و در سمت میزبان تبدیل به رشته­های X، Sc و R می­شوند و احراز هویت با استفاده از این رشته­ها صورت خواهد پذیرفت.

در هر بار فراخوانی الگوریتم احراز هویت، به صورت مشابه رشته­های Xc، S و Rc از ATSHA خوانده شده و با رمزگشایی تبدیل به رشته­های X، Sc و R می­شوند. سپس الگوریتم احراز هویت انجام شده و هر سه این رشته­ها با یک الگوریتم مشخص بروزرسانی می­شوند. در نهایت رشته­های بروز شده Xc، S و Rc که در سمت میزبان تولید می­شوند به سمت ATSHA فرستاده خواهند شد. برای بروزرسانی رشته X، ابتدا 4 بیت LSB و MSB هر بایت جابجا شده و حاصل NOT می­شود، سپس هر بایت رشته با دومین بایت رشته XOR می­شود. به این ترتیب الگوریتم بروزرسانی رشته نیز تا حدی تصادفی خواهد بود.

برای تولید اولین رشته انحرافی که در پروگرامر تولید می­شود از رشته X به این صورت استفاده می­شود که رشته هشت بایتی ابتدای رشته X، یک واحد شیفت یافته به راست، یک واحد شیفت یافته به چپ و دو واحد شیفت یافته به چپ محاسبه شده و در چیپ ذخیره می­شود. پس از هربار احراز هویت نیز رشته انحرافی به این صورت ساخته   
می­شود که رشته شماره سریال ATSHA با یکی از بایت­های رشته X، XOR شده و رشته جدید ساخته می­شود.

### نقاط ضعف و قوت الگوریتم پیشنهادی

مهمترین نقاط قوت الگوریتم ارائه شده عبارتند از اینکه می­توان الگوریتم را به سادگی در پروگرامر و همینطور در هنگام احراز هویت پیاده نمود. ضمن اینکه احتمال رخ داد وضعیت­ ناموفق در هنگام پروگرام و همینطور احراز هویت کم است و همینطور بازیابی از یک وضعیت نامعین با توجه به توضیحاتی که در ادامه خواهد آمد بسیار ساده و کارآمد می­باشد. یکی از مهمترین مزایای الگوریتم پیشنهادی، غیرمتقارن بودن آن است. غیر متقارن بودن به این معناست که هیچ داده با ارزشی که بتوان با استفاده از آن الگوریتم احراز هویت را شکست در سمت میزبان ذخیره نمی­شود و این یک مزیت برای پیاده­سازی است.

یکی از مهمترین مشکلات الگوریتم­های­ احراز هویت امکان دستیابی به الگوی چالش-پاسخ و استفاده از آن برای شکستن الگوریتم است. به عنوان مثال در الگوریتم پیشنهادی اگر هیچگونه عامل تصادفی بودنی در ترتیب خواندن رشته­ها وجود نداشته باشد، اگر یک شنونده غیرمجاز به جای چیپ ATSHA یک چیپ میکروکنترلر قرار دهد که رشته ارسالی به ATSHA را ضبط کرده و در زمان احراز هویت مجدداً آن رشته را به سمت میزبان بفرستد، میزبان عدم وجود ATSHA را تشخیص نخواهد داد و در نتیجه الگوریتم قابل شکستن است. به همین دلیل در پیاده­سازی الگوریتم ترتیب خواندن رشته­ها از ATSHA تصادفی است. ضمن اینکه در زمان نوشتن داده در ATSHA، رشته XOR در میان چندین رشته انحرافی که تعداد آنها تصادفی است ارسال می­شود و به این ترتیب شنونده غیرمجاز قادر نخواهد بود زمان ارسال این رشته مهم را تشخیص دهد. به این ترتیب الگوریتم پیشنهادی در فاز یک تا حد قابل قبولی در برابر نفوذ مقاوم است.

مهمترین عیب الگوریتم فوق این است که داده­ها به صورت رمز شده بر روی خط ارتباطی ارسال نمی­شوند، بنابراین یک شنونده (دانا و دارای اشراف به امور!) می­تواند با گوش دادن به خط و پس از تلاش­های به قدر کافی زیاد به چگونگی عملیات احراز هویت پی ببرد و با پروگرام مجدد میزبان عملیات احراز هویت را دچار اخلال کند. بنابراین پیچیدگی کم الگوریتم و عدم استفاده از چکیده برنامه مهمترین معایب الگوریتم فوق هستند.

## پیاده­سازی الگوریتم­ها

در این بخش در مورد ساختار کلی نحوه پیاده­سازی الگوریتم پروگرام و احراز هویت بحث می­شود. اساس پیاده­سازی که در این بخش صورت گرفته است علاوه بر ساده و خوانا بودن پیاده­سازی، قابلیت بسط دادن آن به   
پیاده­سازی­های آینده است.

یکی از مهمترین مفروضات پیاده­سازی فوق، نامحدود بودن زمان احراز هویت توسط الگوریتم است[[11]](#footnote-12). از طرف دیگر پیاده­سازی الگوریتم به صورت همروند (concurrent) با سایر قسمت­های برنامه است که این بدان معناست که در هر فراخوانی الگوریتم، تنها یک عملیات با طول زمانی بسیار محدود صورت می­پذیرد و با حفظ وضعیت پروگرام/ احراز هویت، می­توان به سایر قسمت­های برنامه میزبان پرداخت. بنابراین تنها محدودیتی که از لحاظ زمانی در   
پیاده­سازی ایجاد می­شود WD چیپ است. برای غلبه بر مشکل WD ترجیح می­دهیم که پس از اتمام انجام هر تک عملیاتی که در قالب یک چرخه­کاری ATSHA انجام می­پذیرد، چیپ را مجدداً به حالت خواب/ بیکار ببریم (و سعی می­کنیم در پیاده­سازی­های آینده نیز این امر را در دستور کار قرار دهیم. باید توجه داشت که این امر مشکلی در پیاده­سازی الگوریتم ایجاد نکند). اما در صورتی که نتوان الگوریتم­های احراز هویت آینده (و یا بعضی از دستورهای آنها) را با این روش پیاده­سازی کرد، لازم است راه­حل­های جایگزین در نظر گرفته شوند (به عنوان مثال چند دستور در فاصله 700 میلی­ثانیه­ای قبل از WD به چیپ فرستاده شوند).

شکل ‏3‑3 ایده کلی پیاده­سازی الگوریتم پروگرام ATSHA (در پروگرامر آن) و همینطور الگوریتم احراز هویت (در میزبان) را نمایش می­دهد. برمبنای این شکل، پروتکل پروگرام و احراز هویت به سه سطح:

* + **سطح اول**: الگوریتم پروگرام ATSHA / الگوریتم احراز هویت توسط ATSHA
  + **سطح دوم:** الگوریتم چرخه­کاری ATSHA
  + **سطح سوم:** ارتباط I2C با ATSHA

شکسته می­شود. در سطح اول الگوریتم، قدم­های کلی الگوریتم پروگرام/ احراز هویت طی می­شود. به این مفهوم سطح اول مجردترین سطح پروتکل را تشکیل می­دهد و با استفاده از آن می­توان پیشرفت یک الگوریتم را مشاهده نمود. سطح دوم الگوریتم نماینده یک چرخه­کاری ATSHA است که در آن هر تک فرمانی که سطح اول تعیین کند به چیپ ارسال می­شود و سپس پاسخ آن دریافت می­شود. سطح سوم پروتکل شامل ارتباط I2C با چیپ ATSHA و جزئیات مرتبط با آن می­باشد که پیاده­سازی آن منطبق با گزارش «*پیاده­سازی پروتکل I2C*» است. توجه داریم هر سطح پروتکل یک پرچم را به بیرون از خود باز می­گرداند تا وضعیت فعلی خود را به سطح بالاتر اعلان کند. بنابراین پایین­ترین سطح پرچم وضعیت ارتباط I2C را به بیرون باز می­گرداند، سطح میانی یک پرچم که نشان دهنده مشغول بودن، خاتمه یا عدم موفقیت چرخه­کاری است را به سطح یک برمی­گرداند، و در نهایت سطح یک در کنار مشغول بودن، موفقیت یا عدم موفقیت احراز هویت/ پروگرام (و البته یک پرچم دیگر!) را به میزبان (در اینجا میکروکنترلر) و یا پروگرامر اعلان می­کند.



شکل ‏3‑3: سطح­بندی پیاده­سازی الگوریتم پروگرام ATSHA/ الگوریتم احراز هویت در سمت میزبان

مهمترین علتی که باعث شده است الگوریتم­ها را در سه سطح پیاده­سازی کنیم، مستقل شدن هر یک از سطوح و در اصلاح modular شدن پیاده­­سازی است. با ترتیبی که در بالا در نظر گرفته شده است (و همانگونه که در ادامه نیز نشان داده خواهد شد)، سطح یک الگوریتم تنها وظیفه دنبال کردن قدم­های پروگرام ATSHA و یا الگوریتم احراز هویت را برعهده دارد (یعنی چه داده­ای باید در ATSHA نوشته شود و یا از آن خوانده شود و ...) و تنها ارتباط آن با سطح دوم دریافت یک پرچم از آن و جابجا کردن یک فرمان و پاسخ آن است. سطح دوم، یک چرخه­کاری را دنبال می­کند و تنها ارتباط آن با سطح قبلی دریافت فرمان در قالب یک رشته و فرستادن پاسخ به آن در قالب یک رشته دیگر است. پایین­ترین سطح ارتباط نیز تنها با سطح دوم در ارتباط است و تنها ارتباط آن با سطح قبلی فرستادن پرچم مشغول بودن ATSHA و چند مورد جزئی دیگر است. بنابراین مشاهده می­شود که جداکردن این سه سطح باعث مدیریت ساده­تر الگوریتم­های احراز هویت می­شود که این خود باعث modular شدن عملیات احراز هویت/ پروگرام ATSHA خواهد شد.

نکته­ای دیگری که در پیاده­سازی فوق وجود دارد این است که از آنجا که یک فرمان مشخص از خارج از سطح دوم پروتکل به آن داده می­شود، هرگونه عملیات ریست کردن بافر و ... تاثیری در اجرای فرمان و خروجی آن نخواهد داشت (به عبارت دیگر فرمان هیچگاه گم نخواهد شد). به عنوان مثال فرض کنیم قرار است یک داده با یک عدد XOR شود و سپس در ATSHA نوشته شود. چون حاصل XOR این داده به سطح دوم داده می­شود، تا هر زمان که این داده بر روی بافر نوشته نشده می­توان بافر را ریست کرد، فرمان را از ابتدا نوشت و غیره. ضمن اینکه عدم موفقیت در ارسال فرمان تنها زمانی ایجاد خواهد شد که از WD عبور کنیم و تا این زمان ارسال/ دریافت داده به هر نحو ممکن ادامه خواهد یافت. بنابراین پیاده­سازی فوق پیاده­سازی مناسبی است. این امر به خصوص از آنجا مطلوب است که در صورتی که یک فرمان تکراری به چیپ ارسال شود (به عنوان مثال به دلیل اینکه وضعیت فعلی در نرم­افزار گم شده است)، هیچ مشکلی در کلیت الگوریتم ایجاد نخواهد شد.

با توجه به اینکه سطح دوم پروتکل حالت کلی­تری دارد و از آن در سطح اول (در پروگرامر و همینطور در پروتکل احراز هویت و در پیاده­سازی­های آینده) نیز استفاده می­شود، در ادامه ابتدا در مورد سطح دوم پروتکل بحث خواهد شد و سپس در مورد سطح اول در پروگرامر و در الگوریتم احراز هویت به صورت جداگانه بحث می­شود. ضمن اینکه برای رعایت پیوستگی مطالب ارائه شده در این گزارش، پیاده­سازی هر سطح پروتکل در قالب ماژول برنامه آن در ادامه هر بخش خواهد آمد.

### نحوه استفاده از پروتکل احراز هویت در برنامه اصلی موجود در پردازنده

یکی از مسائلی که لازم است در این فصل به آن پرداخته شود، نحوه قرار­گیری پروتکل احراز هویت در برنامه اصلی پیاده­سازی شده در پردازنده است. سطح یک پروتکل که توسط تابع ATSHA\_Authentication\_PL1()   
پیاده­سازی می­شود به عنوان خروجی چهار پرچم برمی­گرداند که عبارتند از (بخش ...):

* + پروتکل مشغول است (PL1\_BUSY)
  + پروتکل خاتمه یافته است (PL1\_FINISHED)
  + عدم موفقیت پروتکل احراز هویت (PL1\_FAILURE)
  + بروزرسانی داده­های ATSHA (PL1\_UPDATA\_ATSHA) برای احراز هویت­های بعدی

نکته­ای که در استفاده از ATSHA اهمیت دارد این است که در زمانی پروتکل مشغول بروزرسانی داده­های ATSHA است، امکان رها کردن پروتکل وجود ندارد (به بخش ... رجوع شود)، اما در سایر وضعیت­ها می­توان پروتکل را رها نموده و در زمان مورد نیاز مجدداً به آن مراجعت نمود. سطح دوم پروتکل نیز که توسط تابع ATHSA\_PL2()   
پیاده­سازی می­شود نیز سه پرچم به عنوان خروجی بر می­گرداند که عبارتند از:

* + پروتکل مشغول است (PL2\_BUSY)
  + پروتکل خاتمه یافته است (PL2\_FINISHED)
  + عدم موفقیت پروتکل احراز هویت (PL2\_FAILURE)

ضمن اینکه وضعیت­ سطح سه پروتکل تنها در داخل سطح دو مورد بررسی قرار می­گیرد و امکان دسترسی به آن از خارج سطح دوم وجود ندارد.

توجه داریم سطح دو و سه الگوریتم به دلیل ماهیت ذاتی آنها که پروتکل­های ارتباطی هستند باید در حلقه آسنکرون پیاده­سازی شوند. ضمن اینکه سطح دو نیز نیاز دارد که مرتباً وضعیت ATSHA را بررسی کند و فرمان ارسالی بعدی به آن را معین کند (بخش 3-4) و به همین دلیل نیاز است که در حلقه آسنکرون پیاده­سازی شود. بر خلاف این دو سطح، سطح یک تنها حامل وضعیت­هایی است که کلیات نحوه تعامل با ATSHA را مشخص می­کنند و بعضاً می­توانند حاوی پردازش­های سنگین باشند، بنابراین به لحاظ ماهیتی سطح یک پروتکل به زمان بیشتری برای گذار وضعیت (در بعضی وضعیت­ها) احتیاج دارد و بنابراین بهتر است در حلقه آسنکرون پیاده شود (اگر چه می­توان آن را با در نظر گرفتن بعضی موارد در حلقه آسنکرون پیاده کرد). با در نظر گرفتن این موارد، دو روش برای استفاده از پروتکل احراز هویت در برنامه اصلی پیشنهاد می­شود:

**روش اول: جدا کردن سطح یک و دو (سطح یک در حلقه سنکرون و سطح دو در حلقه آسنکرون)**

در این روش که به عنوان پیشنهاد اصلی برای پیاده­سازی در نظر گرفته می­شود، قسمتی از پروتکل احراز هویت در حلقه سنکرون و قسمتی دیگر در حلقه آسنکرون پیاده­سازی می­شود. نحوه پیاده­سازی پروتکل در این روش به این صورت است که تابع سطح دوم پروتکل مطابق فلوچارت شکل ... و با استفاده از حلقه­ if ارائه شده در حلقه آسنکرون پیاده­سازی می­شود. ضمناً سطح یک پروتکل نیز طبق دیاگرام وضعیت شکل ... در داخل حلقه سنکرون پیاده­سازی می­شود و فراخوانی آن تنها در صورت ارجاع به وضعیت احراز هویت است. این پیاده­سازی علی­رغم پیچیده­تر بودن نسبت به روش دوم (که روش دوم در ادامه خواهد آمد)، چند مزیت مهم دارد که عبارتند از:

1. با جدا کردن سطح یک و دو در پیاده­سازی، امکان حمله و تغییرات در پروتکل احراز هویت بسیار   
   سخت­ می­شود، چون عملاً برنامه احراز هویت در قسمت­های مختلف حافظه فلش پخش می­شود.
2. در صورتی که تمامی پروتکل در حلقه آسنکرون پیاده شود (مشابه روش دوم) ممکن است بعضی وضعیت­هایی که در سطح یک پروتکل تعریف می­شوند و زمان زیادی را اشغال کنند باعث ایجاد مشکل در زمان­بندی برنامه شوند.
3. زمان احراز هویت توسط پروتکل در برنامه قابل برنامه­ریزی است و به بیان دیگر می­توان وضعیت احراز هویت را به عنوان یکی از وضعیت­های برنامه اصلی پروتکل معرفی نمود.

توجه داریم سطح یک پروتکل اصولاً حاوی محاسبات احراز هویت است. به عنوان مثال عملیات بر روی رشته­هایی که از ATSHA خوانده می­شوند در این سطح صورت می­پذیرد، بنابراین قرار دادن آن در حلقه سنکرون تضمین می­کند که تمامی عملیات در یک تیک زمانی برنامه انجام خواهد شد (البته به شرط اینکه زمان مطلق هر یک از وضعیت­ها از مقدار تیک زمانی برنامه به حد کافی کمتر باشد)، ضمناً عملیات ارسال و دریافت به صورت عادی در حلقه آسنکرون ادامه خواهد یافت.

یکی از معایب جدا کردن دو سطح، فاصله افتادن بین ارسال فرمان­ها و طولانی شدن زمان احراز هویت است. در واقع هر گذار وضعیت در سطح یک در این پیاده­سازی در انتهای تیک زمانی رخ می­دهد، و بنابراین اگر ارسال یک فرمان در ابتدای تیک زمانی به پایان برسد ارسال فرمان بعدی تا پایان این تیک زمانی به تاخیر خواهد افتاد[[12]](#footnote-13).

شکل ‏3‑4 نحوه جای دادن توابع متناظر با دو سطح پروتکل در حلقه اصلی را نمایش می­دهد. به دلیل سازوکار درونی پیاده­سازی سطح دوم، این پروتکل تنها زمانی فعال می­شود که یک فرمان جدید از سمت سطح یک در بافر فرمان قرار داده شود (به فصل پنجم و بخش ‏3-4- رجوع شود). بنابراین پس از خاتمه ارسال فرمان فعلی، سطح دوم بیکار خواهد ماند و منتظر سطح یک خواهد ماند تا یک فرمان جدید از طرف سطح یک ارسال شود. به همین دلیل می­توان آن را بدون هیچ مشکلی در حلقه آسنکرون قرار داد.

برای قرار دادن سطح یک در حلقه سنکرون و با فرض اینکه یک وضعیت با نام احراز هویت (AUTHENTICATION STATE) در کد تعریف شده است، پیاده­سازی را براساس دیاگرام وضعیت ارائه شده در شکل ‏3‑5 انجام می­دهیم. در وضعیت AUTHENTICATION STATE، ابتدا تابع پیاده­سازی سطح یک پروتکل (یعنی تابع ATSHA\_Authentication\_PL1()) فراخوانی می­شود تا عملیات مطلوب در سطح یک انجام شود و در ضمن پرچم خروجی پروتکل تعیین گردد. با استفاده از این پرچم، گذار وضعیت بعدی معلوم می­شود. اگر پرچم خروجی پروتکل عدم موفقیت باشد، ابتدا یک شمارنده که تعداد دفعات عدم موفقیت احراز هویت را نگه می­دارد افزایش می­یابد. سپس مقدار این شمارنده با یک آستانه (Threshold) مقایسه می­شود و در صورتی گذر از این آستانه، پروتکل وارد وضعیت BREACH STATE می­شود که در ادامه در مورد آن صحبت خواهد شد. در غیر این صورت، اگر نیاز باشد که به یکی از وضعیت­های عادی برنامه main باز گردیم، می­توان این عمل را انجام داده و در غیر این صورت مجدداً عمل احراز هویت انجام می­شود. اگر وضعیت سطح یک BUSY اعلان شود، به این معناست که پروتکل همچنان مشغول احراز هویت می­باشد. در این حالت در صورتی که لازم است به یکی از وضعیت­های عادی main بازگردیم، می­توان این کار را انجام داد، در غیر این صورت وضعیت بعدی بازهم برابر AUTHENTICATION STATE قرار داده می­شود. اگر پرچم خروجی برابر UPDATA ATSHA باشد، به این معناست که پروتکل در حال بروزرسانی داده­های موجود در ATSHA می­باشد. در این حالت و بر طبق بحث ارائه شده در فصل پنجم (بخش ...) مجاز نیستیم که ارتباط با ATSHA را قطع کنیم چون الگوریتم احراز هویت دچار مشکل خواهد شد، بنابراین وضعیت برابر AUTHENTICATION STATE قرار داده می­شود تا عملیات بروزرسانی منقطع نشود. در نهایت اگر پرچم سطح یک برابر FINISHED باشد به این معناست که پروتکل با موفقیت احراز هویت را انجام داده است، بنابراین ابتدا شمارنده تعداد احراز هویت­های ناموفق برابر صفر قرار داده می­شود و وضعیت بعدی نیز به یکی از وضعیت­های تابع main تغییر خواهد کرد.

وضعیت BREACH STATE که در بالا در مورد آن بحث شد وضعیتی است که در صورتی رخ می­دهد که تعداد احراز هویت­های ناموفق از یک آستانه مشخص فراتر برود. در این وضعیت ابتدا مجدداً مقدار شمارنده تعداد احراز هویت­های ناموفق با آستانه تعریف شده مقایسه می­شود و در صورتی که مقدار از این آستانه بیشتر باشد، هر عملیاتی که لازم است توسط توابع مورد نیاز انجام شود تا دسترسی به نرم­افزار را غیر ممکن کند انجام می­شود. در ضمن از آنجا که گذار وضعیت در ادامه برنامه معنایی ندارد (چون قرار است دسترسی به حلقه اصلی مسدود شود) وضعیت بعدی برای برنامه در نظر گرفته نمی­شود. در غیر این صورت پروتکل به وضعیت احراز هویت گذار خواهد کرد تا عملیات احراز هویت مجدداً آغاز شود. علت بررسی شمارنده عدم موفقیت این مطلب می­باشد که ممکن است وضعیت تابع main (به دلیل نویز و یا هر مورد دیگری) اشتباهاً برابر BREACH STATE قرار بگیرد، و بررسی شرط فوق موجب می­شود که در صورت رخداد این اشتباه بتوان از این وضعیت خارج شد و برنامه را در یک وضعیت مناسب قرار داد. جدول ‏3‑1 حاوی یک پیاده­سازی پیشنهادی برای مطالب اشاره شده در زبان C می­باشد که می­توان آن را در تابعی که وضعیت برنامه را حلقه سنکرون بررسی می­کند (معمولاً تابع Change\_State()) قرار داد.



شکل ‏3‑4: فلوچارت برنامه main سمت میزبان پس از اضافه کردن وضعیت­ احراز هویت



شکل ‏3‑5: دیاگرام وضعیت برنامه main سمت میزبان پس از اضافه کردن وضعیت­ احراز هویت   
(AUTHENTICATION STATE) و وضعیت تشخیص نفوذ در سیستم (BREACH STATE)

جدول ‏3‑1: کد C پیاده­سازی سطح اول پروتکل در حلقه سنکرون در روش اول با استفاده از وضعیت (فرضی)  
 AUTHENTICATION STATE تعریف شده در برنامه اصلی

|  |
| --- |
| switch (current\_state) // current\_state variable denotes the states of the main program.  {  case (AUTHENTICATION\_STATE): // The state in which authentication function is called.  PL1\_output\_flag = ATSHA\_Authentication\_PL1(); // Call the authentication protocol level one function.  if (PL1\_output\_flag == PL1\_FAILURE) // If PL1 has returned a failure flag  {  authentication\_failure\_counter++;  if (authentication\_failure\_counter > AUTHENTICATION\_FAILURE\_THRESHOLD) // If authentication failure  // exceeds a threshold  {  next\_state = BREACH\_STATE; // There has been a breach, so stop the normal function of the program.  }  else if () // Else if we need to continue with the normal program  {  next\_state = ...; // Proceed to normal program states.  }  else // We are allowed to continue the authentication protocol.  {  next\_state = AUTHENTICATION\_STATE;  }  }  else if (PL1\_output\_flag == PL1\_BUSY)  {  if () // If we need to continue with the normal program.  {  next\_state = ...; // Proceed to normal program states.  }  else // We are allowed to continue the authentication protocol.  {  next\_state = AUTHENTICATION\_STATE;  }  }  else if (PL1\_output\_flag == PL1\_UPDATE\_ATSHA)  {  next\_state = AUTHENTICATION\_STATE;  }  else if (PL1\_output\_flag == PL1\_FINISHED)  {  authentication\_failure\_counter = 0;  next\_state = ...; // Proceed to normal program states.  }  break;    case (BREACH\_STATE): // The state in which the necessary tasks are performed to stop divulging the information due to   an attack .  if (authentication\_failure\_counter > AUTHENTICATION\_FAILURE\_THRESHOLD) // Check to see if failure  // counter is actually above  // the threshold so that we  // are not in this state by accident.  {  ... // Do the necessary tasks to stop the normal function of the software.  next\_state = BREACH\_STATE;  }  else // Else we need to exit this state, because the program is here due to a mistake.  {  next\_state = AUTHENTICATION\_STATE;  }      } |

**روش دوم: فراخوانی تمامی سطوح پروتکل به صورت همزمان (در بوت و یا حلقه آسنکرون)**

در روش دوم پیاده­سازی، تمامی سطوح پروتکل به صورت همزمان (در اصل تو در تو) فراخوانی می­شوند. مهمترین مزیتی که پیاده­سازی فوق می­دهد امکان استفاده از الگوریتم احراز هویت در بوت سیستم است. البته توجه داریم که برای پیاده­سازی در بوت، لازم است تابع درون یک حلقه while فراخوانی بشود که در ادامه در مورد آن بحث خواهد شد.

برای پیاده­سازی الگوریتم در بوت، می­توان از فلوچارت شکل ‏3‑12 استفاده نمود. همانگونه که مشاهده می­شود در این پیاده­سازی با استفاده از یک حلقه while بررسی می­شود که آیا پرچم خروجی تابع ATSHA\_Authentication\_PL1() به PL1\_FINISHED تغییر یافته است یا خیر و در صورتی که پاسخ منفی باشد توسط یک حلقه if بررسی می­شود که آیا پرچم یکی از دو حالت مشغول یا بروزرسانی ATSHA است. در صورتی که پاسخ به این سوال مثبت باشد، تابع پروتکل فراخوانی شده و به آن اجازه می­دهد که پروتکل احراز هویت را ادامه دهد ولی اگر پرچم PL1\_FAILURE باشد، کل اجرای برنامه متوقف می­شود. در نهایت اگر هیچکدام از موارد فوق صحیح نباشد، به این معناست که پرچم پروتکل خاتمه یا PL1\_FINISHED است که در این صورت با بازگشت به ابتدای حلقه while، برنامه بوت از چرخه احراز هویت خارج خواهد شد. جدول ‏3‑2 یک نمونه کد C پیاده­سازی الگوریتم در بوت برنامه را نمایش می­دهد.



شکل ‏3‑6: فلوچارت نحوه پیاده­سازی الگوریتم احراز هویت در بوت برنامه

جدول ‏3‑2: کد C پیاده­سازی پروتکل احراز هویت در بوت برنامه

|  |
| --- |
| while (PL1\_output\_flag ~= PL1\_FINISHED)  {  if (PL1\_output\_flag == PL1\_BUSY || PL1\_output\_flag == PL1\_UPDATE\_ATSHA) // If PL1 has returned a failure flag  {  PL1\_output\_flag = ATSHA\_Authentication\_PL1(); // Call the authentication protocol.  }  else if (PL1\_output\_flag == PL1\_FAILURE)  {  ... // Do the necessary tasks to stop the normal function of the software.  }  } |

روش دیگر پیاده­سازی، استفاده از پروتکل در تابع آسنکرون می­باشد. همانگونه که اشاره شد مهمترین عیب­ انجام اینکار، طولانی بودن نسبی عملیات بعضی وضعیت­های سطح یک است که ممکن است باعث بروز بعضی مشکلات در زمان­بندی برنامه بشود. فلوچارت نحوه پیاده­سازی پروتکل احراز هویت در شکل ‏3‑7 آمده است. همانگونه که مشاهده می­شود در ابتدای فراخوانی الگوریتم احراز هویت در حلقه آسنکرون، وضعیت یک پرچم بررسی می­شود که تعیین می­کند آیا باید احراز هویت انجام شود یا خیر و یا اینکه پرچم خروجی PL1 بروزرسانی ATSHA می­باشد و یا خیر و در این دو حالت وارد چرخه احراز هویت می­شویم. دلیل بررسی شرط برابر بودن پرچم خروجی PL1 با بروزرسانی داده­های ATSHA همانگونه که قبلاً اشاره شد این مطلب است که در حین بروزرسانی داده­ها مجاز به رها کردن پروتکل نیستیم. در مورد پرچم احراز هویت نیز این پرچم پرچمی است که می­تواند در برنامه حلقه سنکرون و یا هر قسمت مناسب دیگر برنامه تنظیم شود و تنظیم شدن آن به این معناست که باید احراز هویت انجام شود. در همین نقطه لازم است ذکر شود که یک عیب دیگر پیاده­سازی الگوریتم در حلقه آسنکرون می­تواند بررسی همین دو شرط باشد؛ ممکن است پردازنده نیاز به بروزرسانی به پایه­های ارتباط I2C داشته باشد، اما به دلیل نیاز به بروزرسانی داده توسط ATSHA برنامه متوجه نشود که نمی­توان این پایه­ها را رها کرد (چون برنامه با تنظیم پرچم احراز هویت فرض می­کند که دیگر احراز هویتی انجام نخواهد شد. توجه داریم این مشکل در دو پیاده­سازی قبلی وجود ندارد). راه حل این مشکل، بررسی پرچم PL1 در زمان تنظیم پرچم احراز هویت است و بنابراین اگر این پرچم برابر PL1\_UPDATE\_ATSHA باشد، این پرچم باید همچنان به احراز هویت تنظیم شود.

پس از ­گذار از شرط اول، نوبت به بررسی وضعیت­های مختلف پرچم سطح یک می­رسد. اگر پرچم خروجی پروتکل برابر PL1\_FAILURE باشد، به این معناست که پروتکل دچار عدم موفقیت شده است و در نتیجه باید اعمال مناسبی برای جلوگیری از دسترسی به برنامه انجام شود. همچنین اگر پرچم برابر PL1\_UPDATE\_ATSHA و یا PL1\_BUSY باشد، باید پروتکل فراخوانی شده و اجازه گذار وضعیت به آن داده شود. در نهایت اگر پرچم برابر PL1\_FINISHED باشد به این معناست که پروتکل خاتمه یافته است و در نتیجه عملیات احراز هویت خاتمه یافته است. جدول ‏3‑3 یک نمونه کد C پیاده­سازی الگوریتم در حلقه آسنکرون برنامه را نمایش می­دهد.



شکل ‏3‑7: فلوچارت نحوه پیاده­سازی الگوریتم احراز هویت در حلقه آسنکرون

جدول ‏3‑3: کد C پیاده­سازی پروتکل احراز هویت در حلقه آسنکرون برنامه

|  |
| --- |
| while (PL1\_output\_flag ~= PL1\_FINISHED)  {  if (PL1\_output\_flag == PL1\_BUSY || PL1\_output\_flag == PL1\_UPDATE\_ATSHA) // If PL1 has returned a failure flag  {  PL1\_output\_flag = ATSHA\_Authentication\_PL1(); // Call the authentication protocol.  }  else if (PL1\_output\_flag == PL1\_FAILURE)  {  ... // Do the necessary tasks to stop the normal function of the software.  }  } |

## سطح دوم پروتکل

همانطور که در قبل نیز اشاره شد، هدف از سطح دوم پروتکل، پیاده­سازی یک چرخه کاری ATSHA   
می­باشد. در این بخش در مورد نحوه­ پیاده­سازی سطح دوم پروتکل بحث می­شود. سعی بر این است که این   
پیاده­سازی به نحوی باشد که بتوان در پیاده­سازی­های آینده و بدون ایجاد هیچگونه تغییرات چه در پروگرامر ATSHA و چه در پروتکل احراز هویت از آن استفاده نمود.

دیاگرام وضعیت ساده ارائه شده در شکل ‏3‑8 که در واقع بسط داده شده دیاگرام وضعیت ارائه شده در شکل ‏2‑1 است، نحوه پیاده­سازی پروتکل را نمایش می­دهد. برای نیل به یک پیاده­سازی ساده و خوانا که حجم برنامه­ نوشته شده را تا حد مطلوبی کاهش می­دهد و در عین حال امکان بازیابی برنامه از وضعیت­های خطا را ممکن بسازد، پیاده­سازی سطح دو بر اساس دیاگرام وضعیت ارائه شده در این شکل انجام می­شود. همانگونه که مشاهده می­شود چرخه­کاری به این صورت عمل می­کند که پس از دریافت یک فرمان از سطح یک، چیپ را از حالت خواب بیدار   
می­کند و پاسخ WAKE TOKEN را طبق بخش ‏2-6-1- دریافت می­کند. سپس فرمان مورد نظر به چیپ ارسال می­شود و پس از آن نوبت به دریافت پاسخ می­رسد و در انتهای چرخه کاری نیز بسته به نیاز، چیپ به یکی از دو حالت خواب/ بیکار فرو خواهد رفت. چند نکته در مورد پیاده­سازی فوق از چرخه­کاری اهمیت دارد:

1. دوره زمانی ارسال و دریافت در یک چرخه کاری با پیاده­سازی فوق هیچگاه از 700 میلی ثانیه WD چیپ بیشتر نخواهد شد (چون مدت زمان اجرای هیچ فرمانی بیش از چند ده میلی­ثانیه نیست).
2. در طول یک چرخه کاری تنها می­توان **یک فرمان** را به چیپ ارسال نمود. **بنابراین در پیاده­سازی­های آینده باید در نظر گرفته شود که در سطح دوم تنها می­توان یک فرمان را به ATSHA فرستاد و پاسخ آن را دریافت کرد.**
3. انتخاب از میان فرستادن چیپ به حالت خواب/ بیکار کمک می­کند تا برای الگوریتم­های احراز هویت پیچیده­تر چیپ بتواند بعضی اطلاعات را طبق توضیحات ارائه شده در بخش ‏2-6-2- در خود نگه دارد.
4. پیاده­سازی سطح دو به صورتی انجام می­پذیرد که در صورتی که انجام یک عملیات از مقدار 700   
   میلی­ثانیه فراتر برود، پروتکل یک پرچم عدم موفقیت پروتکل را به بیرون بازگرداند. در پیاده­سازی فعلی (و به طور مشخص در سطح یک) دریافت این پرچم معادل یک عدم موفقیت در احراز هویت/ پروگرام ATSHA تلقی می­شود.



شکل ‏3‑8: دیاگرام وضعیت پروتکل سطح دو در بالاترین سطح

وضعیت­هایی که برای پیاده­سازی دیاگرام وضعیت سطح دو مورد استفاده قرار می­گیرند در جدول ‏3‑4   
آمده­اند. توجه داریم مقادیر عددی تخصیص یافته به این وضعیت­ها در نرم­افزار کد کاملاً اختیاری هستند و منطق مقدار­دهی به آنها به این صورت بوده است که اولین وضعیت ممکن (PL2 INITIATION) کوچکترین عدد را دریافت کرده است و الی آخر. در ادامه در مورد گذار وضعیت­ها در پروتکل سطح دوم بحث خواهد شد.

**یادآوری:­** همانگونه که در بخش قبلی نیز بحث شد، تابع پیاده­سازی سطح دوم سه پرچم خروجی دارد که عبارتند از PL2\_BUSY، PL2\_FINISHED و PL2\_FAILURE است که مورد اول نشان می­دهد که پروتکل مشغول است، مورد نشان می­دهد که پروتکل خاتمه یافته است و مورد سوم نشان می­دهد که یک عدم موفقیت به دلیل عدم موفقیت در ارسال فرمان قبل از زمان WD چیپ ATSHA رخ داده است. در ادامه بیشتر در مورد این سه پرچم بحث خواهد شد.

جدول ‏3‑4: وضعیت­های سطح دوم پروتکل و نماد آنها در کد

| وضعیت | نماد در کد |
| --- | --- |
| راه­اندازی سطح دوم | PL2 INITIATION |
| بیدار کردن ATSHA | WAKE ATSHA |
| بررسی بیدار شدن ATSHA | CHECK ATSHA AWAKE |
| ریست کردن بافر ATSHA | RESET ATSHA BUFFER |
| ارسال فرمان به ATSHA | SEND COMMAND |
| دریافت پاسخ از ATSHA (خواندن بافر داده) | READ RESPONSE |
| فرستادن ATSHA به حالت خواب | SET ATSHA SLEEP |
| فرستادن ATSHA به حالت بیکار | SET ATSHA IDLE |
| پایان دادن به عملیات سطح دوم | PL2 FINALIZATION |
| بازیابی پروتکل سطح 2 از یک وضعیت نامعین | PL2 RECOVERY STATUS |
| کذر زمان WD failsafe چیپ | PL2 TIMEOUT |

### وضعیت راه­اندازی سطح دوم

وضعیت راه­اندازی سطح دوم یا PL2 INITIALIZATION اولین وضعیت پس از راه­اندازی پروتکل است. دیاگرام وضعیت این حالت در شکل ‏3‑9 کشیده شده است. بر طبق این دیاگرام در این وضعیت ابتدا بررسی می­شود که آیا در ورودی تابع این پروتکل فرمان جدیدی قرار داده شده است و یا خیر. اگر پاسخ مثبت باشد، ابتدا پرچم خروجی پروتکل به PL2\_BUSY تغییر می­کند تا نشان بدهد پروتکل مشغول به کار شده است و سپس شمارنده تعداد بایت­های ارسالی از این فرمان به سمت ATSHA صفر خواهد شد (در ادامه در مورد این شمارنده بیشتر بحث خواهد شد). در نهایت نیز گذار وضعیت به وضعیت WAKE ATSHA انجام می­شود. توجه داریم اگر فرمان قبلی و فعلی یکسان باشند پروتکل از وضعیت PL2 INITIALIZATION گذار نخواهد کرد که این به معنای آن است اگر پروتکل آماده به کار باشد ولی فرمان ارسالی به آن تفاوتی نکرده باشد، پروتکل مشغول به کار نخواهد شد. در مورد چرایی این پیاده­سازی لازم است یادآوری شود که همانگونه که در روش اول پیاده­سازی پروتکل احراز هویت اشاره شد لازم است پروتکل سطح دوم به صورتی در حلقه آسنکرون پیاده­سازی شود که پس از پایان ارسال یک فرمان، فرمان تکراری مجدداً به سمت ATSHA ارسال نشود. بررسی شرط فوق باعث می­شود که تنها در صورتی که ارسال یک فرمان خاتمه یافته است بتوان مجدداً یک فرمان به سمت ATSHA فرستاد.

**تذکر:** در پیاده­سازی پروتکل نام فرمان ارسالی به ATSHA (که پارامتر command\_name تعریف شده در ساختار command\_response است) برای مقایسه برابر بودن دو فرمان ارسالی به آن استفاده می­شود. بنابراین اگر در پیاده­سازی­های آینده نیاز باشد که یک فرمان یکسان چندین بار به سطح دوم ارسال شود   
تنها کافیست نام­های مختلفی به این فرمان نسبت داده شود.

لازم به یادآوری است که در صورت گذار به وضعیت WAKE ATSHA، متغیر زمان­سنج سطح دوم که از آن برای نگه­داشتن زمان WD سطح دوم استفاده می­شود صفر خواهد شد تا زمان­گیری WD با ارسال WAKE TOKEN همزمان شود.



شکل ‏3‑9: دیاگرام وضعیت راه­اندازی سطح دوم پروتکل

### وضعیت بیدار کردن ATSHA

وضعیت بیدار کردن ATSHA دومین وضعیت پس از راه­اندازی پروتکل است. دیاگرام وضعیت این حالت در شکل ‏3‑10 کشیده شده است. در هر زمانی که این وضعیت فراخوانی بشود، یک WAKE TOKEN با استفاده از ارتباط I2C به صورتی که در بخش ‏2-6-1- توضیح داده شد به سمت ATSHA ارسال می­شود. پس از پایان ارسال این سیگنال نیز طبق بحثی که در همان بخش شد پروتکل منتظر می­ماند تا I2C وارد وضعیت NACK شدن آدرس Slave در مد MT (وضعیت I2C\_MT\_ADDRESS\_NACK) شود و سپس وارد وضعیت  
CHECK ATSHA AWAKE می­شود. علت اینکه پروتکل منتظر وضعیت NACK کردن آدرس خود می­شود این است که به سطح سوم (پروتکل I2C) اجازه دهد که گذار وضعیت­های مورد نیاز برای فرستادن سیگنال WAKE TOKEN را بدهد. این امر تنها زمانی به صورت کامل رخ داده است که سطح سوم وارد وضعیت NACK شدن آدرس خود بشود.



شکل ‏3‑10: دیاگرام وضعیت بیدار کردن ATSHA

## وضعیت بررسی بیدار شدن ATSHA

هدف از وضعیت CHECK ATSHA AWAKE، بررسی این مطلب است که آیا چیپ پس از دریافت WAKE TOKEN بیدار شده است و یا خیر. دیاگرام وضعیت این حالت در شکل ‏3‑11 آمده است. پس از گذار به وضعیت CHECK ATSHA AWAKE، پروتکل شروع به آدرس­دهی ATSHA در حالت MR می­کند تا به بافر داده آن دسترسی پیاده کرده و پاسخ به WAKE TOKEN را بخواند. در صورتی که سطح دوم موفق نشود که در طی یک تیک زمانی اینکار را انجام دهد (یعنی به بافر دسترسی پیدا کند که در این بدان معناست که چیپ بیدار نشده و در نتیجه آدرس خود را NACK می­کند، یعنی وضعیت I2C\_MR\_ADDRESS\_NACK)، مجدداً به وضعیت WAKE ATSHA باز می­گردد تا یک WAKE TOKEN جدید به سمت ATSHA ارسال شود. در غیر این صورت پاسخ به WAKE TOKEN دریافت شده و در نتیجه باید صحت این پاسخ بررسی و مطابقت آن با جدول ‏2‑9 بررسی شود. اگر پاسخ دریافتی از چیپ صحیح باشد، آنگاه چیپ به وضعیت SEND COMMAND گذار می­کند، در غیر این صورت چیپ مجدداً به وضعیت WAKE ATSHA باز می­گردد.

توجه داریم در یک حالت بسیار نادر ممکن است ارسال WAKE TOKEN در نزدیکی پایان یک تیک زمانی برنامه انجام شود و پس از آن نیز از آنجا که I2C تا زمان بیدار شدن چیپ از سوی آن NACK دریافت خواهد کرد و تیک زمانی نیز در این لحظه پایان پیدا می­کند، چیپ مجدداً به وضعیت WAKE ATSHA بازگردد. در این حالت مجدداً WAKE TOKEN به سمت ATSHA ارسال می­شود، در حالی که ممکن است چیپ نیز به دلیل   
WAKE TOKEN قبلی بیدار شود. نکته­ای که باید در نظر داشت این است که ارسال WAKE TOKEN دوم باعث ایجاد مشکل نخواهد شد، چون چیپ آن را در نظر نخواهد گرفت. بنابراین ممکن است چرخه فوق زمان بیدار شدن چیپ را به اندازه یک تیک زمانی افزایش بدهد، اما اخلالی در روند آن ایجاد نخواهد کرد.



شکل ‏3‑11: دیاگرام وضعیت بررسی بیدار شدن ATSHA

## وضعیت ارسال فرمان به ATSHA

هدف از وضعیت SEND COMMAND، ارسال فرمان مورد نظر به ATSHA در یک چرخه­کاری است. دیاگرام وضعیت این حالت در شکل ‏3‑12 آمده است. در این وضعیت با تغییر مد کاری ATSHA به MT، فرمان مورد نظر در بافر ATSHA نوشته می­شود. در هر بار فراخوانی وضعیت SEND COMMAND، وضعیت ارتباط I2C بررسی می­شود. در صورتی که این ارتباط مشغول باشد در تکرار بعدی مجدداً به همین وضعیت گذار خواهیم کرد تا عملیات ارسال فرمان تکمیل شود. پس از پایان ارسال بایت فعلی و زمانی که وضعیت I2C به I2C\_PROTOCOL\_FINISHED تغییر پیدا کند، اندیس ارسال یک واحد افزایش پیدا می­کند. با پایان یافتن این چرخه و زمانی که تمامی بایت­های این فرمان ارسال شوند[[13]](#footnote-14)، سطح دوم به وضعیت READ RESPONSE گذار خواهد نمود.



شکل ‏3‑12: دیاگرام وضعیت ارسال فرمان به ATSHA

## دیاگرام وضعیت دریافت پاسخ از ATSHA

هدف از وضعیت READ RESPONSE، خواندن بافر ATSHA و به عبارت دیگر دریافت پاسخ از آن است. دیاگرام وضعیت این حالت در شکل ‏3‑13 نمایش داده شده است. همانگونه که مشاهده می­شود مشابه حالت ارسال فرمان، دریافت پاسخ از ATHSA توسط I2C و تا زمان دریافت تمامی بایت­های پاسخ ادامه پیدا می­کند. در اینجا لازم است منظور از تمامی بایت­های پاسخ معین شود. توجه داریم همانگونه که طول یک فرمان ارسالی به ATSHA معین است، طول پاسخ به یک فرمان نیز به طور پیش فرض معین است (به عنوان مثال می­دانیم یک خواندن 32 بایتی از ATSHA طول 36 بایتی دارد). اما در صورتی که ATSHA قادر نباشد فرمان را به صورت صحیح انجام دهد، در قالب چهار بایتی یک خطا به بیرون ارسال می­کند. بنابراین لزوماً با دانستن طول پاسخ به یک فرمان ارسالی   
نمی­توان تضمین کرد که طول پاسخ دریافتی از ATSHA برابر همان مقدار مورد نظر است. به همین دلیل در وضعیت دریافت پاسخ از ATSHA، اولین بایت دریافتی (که همان count است)، تعدادی بایتی که باید در این وضعیت دریافت شود را مشخص می­کند. پس از اتمام دریافت، وضعیت پاسخ دریافتی بررسی می­شود و در صورتی که بایت خطا/ وضعیت آن مقداری غیر از مقدار مورد نظر داشته باشد و یا CRC دریافتی با مقدار مورد نظر تطبیق نداشته باشد، مجدداً به وضعیت SEND COMMAND باز می­گردیم تا با ارسال مجدد فرمان پاسخ آن دریافت شود.

توجه داریم مشابه پیاده­سازی که در ارتباط I2C شده است، اگر CRC دریافتی صحیح نباشد نمی­توان با NACK کردن از ATSHA خواست که کل بافر را مجدداً بفرستد. در واقع همانطور که اشاره شد برای انجام اینکار لازم است یک فرمان RESET به سمت ATSHA ارسال شود تا کل بافر مجدداً ارسال شود. به همین دلیل   
پیاده­سازی بررسی CRC خارج از تابع I2C و پس از پایان دریافت تمامی داده­ها است. در صورتی که CRC دریافتی با مقدار محاسبه شده از داده­ها برابر نباشد، ابتدا یک فرمان Reset به سمت ATSHA ارسال می­شود و سپس تمامی بافر از ابتدا خوانده می­شود.

اگر پس از دریافت فرمان تعیین شود که هم CRC آن صحیح است و هم بایت وضعیت/ خطای آن مقدار مطلوب را دارد، بسته به درخواستی که از سطح اول پروتکل آمده است، پروتکل وارد وضعیت­های  
SET ATSHA IDLE و یا SET ATSHA SLEEP می­شویم.



شکل ‏3‑13: دیاگرام وضعیت دریافت پاسخ از ATSHA

## وضعیت­های فرستادن چیپ به حالت خواب و بیکار

هدف از وضعیت­های SET ATSHA SLEEP و SET ATSHA IDLE، فرستادن چیپ به وضعیت­های خواب و بیکار است. دیاگرام وضعیت این حالت­ها در شکل ‏3‑14 آمده است. همانگونه که مشاهده می­شود بسته به درخواست سطح بالاتر پروتکل، به یکی از دو وضعیت فوق وارد می­شویم که بر طبق آن فرمان­های مربوطه مطابق بخش ‏2-6-2- به چیپ ارسال می­شوند. پس از پایان ارسال فرمان­ها نیز ابتدا پرچم خاتمه پروتکل به بیرون ارسال می­شود و سپس پروتکل وارد وضعیت PL2 FINALIZATION می­شود.



شکل ‏3‑14: دیاگرام وضعیت فرستادن ATSHA به حالت­های خواب و بیکار

## وضعیت خاتمه پروتکل سطح دو

پس از فرستادن پروتکل به یکی از حالت­های خواب و یا بیکار، پروتکل وارد وضعیت نامعین می­شود. دیاگرام وضعیت این حالت در شکل ‏3‑16 آمده است. همانگونه که مشاهده می­شود با گذار به این وضعیت ابتدا نام فرمانی که در فراخوانی فعلی پروتکل به سمت ATSHA فرستاده شده است برای مقایسه با فراخوانی بعدی ذخیره می­شود، ضمن اینکه پرچم خروجی پروتکل نیز به PL2\_FINISHED تغییر می­کند تا به سطح یک نشان داده شود که ارسال فرمان در سطح دو خاتمه یافته است. در نهایت وضعیت پروتکل به PL2\_INITIALIZATION تغییر می­کند تا پروتکل برای استفاده­های بعدی آماده به کار شود. عمل دیگری که در این وضعیت انجام می­شود، صفر کردن   
زمان­سنج سطح دوم است که با صفر کردن آن، مقدار زمان­سنج برای کاربردهای بعدی از صفر شروع خواهد شد.



شکل ‏3‑15: دیاگرام وضعیت فرستادن ATSHA به حالت­های خواب و بیکار

## وضعیت نامعین، ریکاوری از وضعیت نامعین و وضعیت ریست کردن بافر ATSHA

در صورتی که وضعیت فعلی سطح دوم در نرم­افزار گم شود، پروتکل وارد وضعیت نامعین می­شود. دیاگرام وضعیت این حالت در شکل ‏3‑16 آمده است. همانگونه که مشاهده می­شود پس از ورود به وضعیت نامعین، پروتکل ابتدا وارد یک وضعیت میانی به نام PL2 RECOVRY می­شود. هدف از این وضعیت این است که به سطح سوم اجازه داده شود که تبادل فعلی داده توسط ارتباط I2C خاتمه پیدا کند تا با آزاد شدن I2C بتوان عملیات بعدی را انجام شد. پس از آزاد شدن I2C، چیپ وارد وضعیت RESET ATSHA BUFFER می­شود که در آن مطابق با بخش   
‏2-6-3-، یک فرمان ریست شمارنده بافر داده­ها به چیپ ارسال می­شود. پس از خاتمه ارسال این فرمان، چیپ مجدداً آماده دریافت فرمان خواهد بود و به همین دلیل به وضعیت SEND COMMAND گذار می­کنیم.

توجه داریم با ارسال فرمان ریست به چیپ، بدون توجه به اینکه ممکن است چیپ در حال ارسال فرمان و یا دریافت پاسخ بوده است، شمارنده بافر به ابتدای آن باز می­گردد. بنابراین گذار به وضعیت SEND COMMAND باعث می­شود که چیپ مجدداً آماده دریافت فرمان شود و در نتیجه در کلیت اجرای سطح اول پروتکل مشکلی پیش نخواهد آمد.



شکل ‏3‑16: دیاگرام وضعیت ریست کردن بافر ATSHA

### وضعیت Timeout سطح دوم

در صورتی که ارسال و دریافت یک فرمان به ATSHA تا قبل از عمل کردن WD آن خاتمه پیدا نکند، پروتکل وارد وضعیت نامعین می­شود. توجه داریم بررسی عبور از زمان WD در هر بار گذار وضعیت (و در تابعی با نام PL2\_Assign\_Operation() که در بخش ‏3-11-2- تعریف شده است) انجام می­پذیرد. دیاگرام وضعیت این حالت در شکل ‏3‑17 آمده است. همانگونه که مشاهده می­شود با ورود به این وضعیت ابتدا مقدار زمان­سنج سطح دوم برابر صفر قرار داده می­شود، سپس پرچم خروجی به PL2\_FAILURE تنظیم می­شود و در نهایت وضعیت پروتکل برابر PL2\_INITIALIZATION قرار داده می­شود تا در زمان استفاده بعدی از پروتکل بتوان از ابتدا عملیات را آغاز کرد. نکته­ای که در مورد این وضعیت اهمیت دارد آن است که اگر سطح دوم در دفعات قبلی کارکرد یک پرچم عدم موفقیت بازگردانده باشد، باید به آن اجازه داده شود که حتی فرمان تکراری را نیز ارسال کند. در توضیح علت این امر، همانگونه که در مورد سطح اول الگوریتم احراز هویت نیز اشاره خواهد شد (بخش ... دیاگرام وضعیت)، در صورتی که سطح یک موفق به اتمام یک عملیات به دلیل timeout در سطح دوم نشود، بعضاً مجبور است همان فرمانی که ارسال آن موفقیت آمیز نبوده را مجدداً به سمت ATSHA بفرستد. حال از آنجا که در وضعیت PL2\_INITIALIZATION بررسی می­شود که فرمان قبلی و فعلی برابر بوده و یا خیر و تنها در صورت عدم برابری اجازه ارسال فرمان داده می­شود، در وضعیت PL2 TIMEOUT، اسم فرمان برابر NO\_COMMAND (فرمان تهی) قرار داده می­شود تا در استفاده بعدی از پروتکل بتوان مجدداً با آن یک فرمان ارسال کرد و به این ترتیب پروتکل در فراخوانی بعد و با حضور پرچم عدم موفقیت دچار مشکل نخواهد شد.



شکل ‏3‑17: دیاگرام وضعیت ریست کردن بافر ATSHA

## ماژول سطح دوم پروتکل پروگرام/ احراز هویت ATSHA

این ماژول و هدر متناظر آن که حاوی تمامی توابع پیاده­سازی­ مرتبط با سطح دوم پروتکل می­باشد به صورت ATSHA\_PL2.c نام­گذاری شده است. توجه داریم این ماژول میان پروتکل احراز هویت و پروتکل پروگرام یکسان است، ضمن اینکه در پیاده­سازی فازهای بعدی پروژه نیز (با کمترین و یا حتی هیچگونه تغییرات) می­توان از این پیاده­سازی برای سطح دوم استفاده کرد. فهرست توابعی که در این ماژول تعریف شده­اند در جدول ‏3‑5 و فهرست ثوابت تعریف شده آن در جدول ‏3‑6 آمده است. در اینجا لازم است در مورد بعضی ثوابت تعریف شده در این هدر توضیحات بیشتری ارائه شود:

* **ATSHA\_WD\_DURATION:** این ثابت نشان دهنده زمان WD چیپ ATSHA است و نشان   
  می­دهد این زمان معادل چند تیک زمانی برنامه است. به این ترتیب این ثابت باید بر همین اساس (یعنی تعداد تیک­های برنامه حلقه اصلی) تنظیم شود.
* **MAX\_COMMAND\_LENGTH:** نشان دهنده بیشینه طول فرمان ارسالی به چیپ به واحد بایت است. در پیاده­سازی فاز اول این مقدار برابر 39 خواهد بود.
* **MAX\_RESPONSE\_LENGTH:** نشان دهنده بیشینه طول پاسخ دریافتی از ATSHA است. در پیاده­سازی فاز اول این مقدار برابر 36 خواهد بود.
* **پارامترهای نرخ ارسال سیگنال WAKE TOKEN شامل WAKE\_TOKEN\_I2C\_BITRATE و WAKE\_TOKEN\_I2C\_FREQUENCY\_PRESCALAR:** این دو پارامتر نشان دهنده نرخ بیت و همینطور مقسم فرکانسی برای ارسال WAKE TOKEN به سمت ATSHA هستند که در مورد آنها در بخش ‏2-6-1- توضیح داده شد. یادآوری می­شود که اگر قرار باشد فرکانس کاری چیپ از 8MHz تجاوز کند لازم است مقدار این پارامتر به صورت مناسب اصلاح شود.

در ادامه در مورد کارکرد هر یک از توابع این ماژول بحث خواهد شد.

جدول ‏3‑5: فهرست توابع تعریف شده در ماژول سطح دو پروتکل

|  |  |
| --- | --- |
| تابع | کارکرد |
| ATSHA\_PL2 | تابع پیاده­سازی چرخه­کاری ATSHA |
| PL2\_Assign\_Operation | تعیین وضعیت پروتکل با توجه به پرچم ارتباط I2C، پاسخ دریافتی از ATSHA در صورت وجود و همینطور وضعیت فعلی پروتکل |

جدول ‏3‑6: فهرست ثوابت/ enumها/ ساختارها/ متغیرهای خارجی تعریف شده در هدر سطح دو پروتکل

| نام | تعریف |
| --- | --- |
| ثوابت | |
| ATSHA\_WD\_DURATION | مد زمان WD چیپ ATSHA (به واحد تیک زمانی برنامه) |
| MAX\_COMMAND\_LENGTH | بیشینه طول فرمان ارسالی به ATSHA |
| MAX\_RESPONSE\_LENGTH | بیشینه طول پاسخ دریافتی از ATSHA |
| PL2\_INITIALIZATION تا PL2\_TIMEOUT | وضعیت­های سطح دوم پروتکل |
| WAKE\_TOKEN\_I2C\_BITRATE | نرخ بیت WAKE TOKEN در ارتباط I2C |
| WAKE\_TOKEN\_I2C\_ FREQUENCY\_PRESCALAR | مقسم فرکانسی WAKE TOKEN در ارتباط I2C |
| WAKE\_TOKEN\_COMMAND | فرمان WAKE TOKEN |
| ATSHA\_AWAKE\_RESPONSE | پاسخ ATSHA به فرمان بیدار شدن |
| ATSHA\_AWAKE\_RESPONSE\_LENGTH | طول پاسخ به فرمان بیدار شدن |
| RESET\_BUFFER\_WORD\_ADDRESS | کلمه آدرس ریست کردن بافر داده ATSHA |
| RESET\_RESPONSE\_LENGTH | طول فرمان ریست |
| ATSHA\_SLEEP\_WORD\_ADDRESS | کلمه آدرس بردن ATSHA به حالت خواب |
| ATSHA\_IDLE\_WORD\_ADDRESS | کلمه آدرس بردن ATSHA به حالت بیکار |
| enumها | |
| PL2\_flag | پرچم­های خروجی سطح دوم شامل:  PL2\_READY: آماده بودن سطح دوم برای انجام عملیات  PL2\_BUSY: مشغول بودن سطح دو  PL2\_FINISHED: سطح دو خاتمه یافته است  PL2\_FAILURE: عدم موفقیت در سطح دو |
| ساختارها | |
| command\_response\_struct | ساختار فرمان. پاسخ شامل موارد زیر:  command[]: آرایه­ای از کاراکترها با طول MAX\_COMMAND\_LENGTH حاوی فرمان ارسالی به ATSHA در این چرخه  command\_name: نام فرمان ارسالی به ATSHA  response[]: آرایه­ای از کاراکترها با طول MAX\_RESPONSE\_LENGTH حاوی پاسخ دریافتی از ATSHA در این چرخه  response\_status\_error: مقدار بایت وضعیت/خطا در پاسخ به فرمان ارسالی  response\_length: طول پاسخ دریافتی از ATSHA  sleep\_idle: متغیر تعیین کننده بردن چیپ به حالت خواب/ بیکار پس از اجرای فرمان |

### تابع پیاده­سازی سطح دوم پروتکل (ATSHA\_PL2())

|  |  |
| --- | --- |
| **کارکرد:** | دریافت فرمان­ مورد نظر از سطح بالاتر پروتکل و اجرای یک چرخه­کاری |
| **ورودی­ها:** | * اشاره­گر به متغیر از نوع command\_response\_srtuct (\*pcommand\_response) * زمان فعلی سیستم (شمارنده تیک زمانی برنامه) |
| **خروجی­:** | enum از نوع PL2\_flags، نشان دهنده وضعیت فعلی سطح دوم |

**شرح تفصیلی:** هدف از این تابع پیاده­سازی یک چرخه­کاری ATSHA می­باشد که شرح چگونگی کارکرد این چرخه­کاری در دیاگرام­های وضعیت­ بخش­ ‏3-4- آمده است. پیاده­سازی این سطح با استفاده از یک حلقه switch صورت می­پذیرد که در آن بسته به عملیات فعلی که در متغیر current\_operation ذخیره می­شود، پروتکل وارد یکی از وضعیت­های مورد اشاره در بخش­ ‏3-4- می­شود. توجه داریم current\_operation یک متغیر ایستا (static) است و بنابراین عملیات در هر بار فراخوانی تابع حفظ خواهد شد. در هر یک از وضعیت­ها و به جز چند استثناء، پروتکل با استفاده از ارتباط I2C یک یا چند بایت را به سمت ATSHA ارسال می­کند. متغیری که در کد تعداد بایت­های ارسالی/ دریافتی را حفظ می­کند transception\_counter نام دارد که یک متغیر ایستا است و بسته به وضعیت فعلی پروتکل، تعداد بایت­های ارسالی/ دریافتی به ATSHA را در خود حفظ می­کند. این متغیر در وضعیت   
PL2\_INITIALIZATION و همینطور پس از خاتمه ارسال/ دریافت داده به چیپ صفر خواهد شد. از طرف دیگر متغیر محلی I2C\_status که پارامتر خروجی پروتکل I2C است (به گزارش «*پروتکل ارتباطی I2C*» رجوع شود) وضعیت فعلی I2C را در خود نگه­داری می­کند و هرگونه گذار وضعیت در پروتکل تنها زمانی صورت می­پذیرد که I2C خاتمه پیدا کند. توجه داریم (به استثنای وضعیت WAKE ATSHA) برای تعیین گذار وضعیت­ها تابع PL2\_Assign\_Operation() فراخوانی می­شود که هدف از آن تعیین وضعیت این سطح با توجه به وضعیت ارسال/ دریافت داده به ATSHA است. متغیر ایستای دیگری که در کد تعریف شده است previous\_command نام دارد که این متغیر نام آخرین فرمانی را که توسط پروتکل ارسال شده است در خود نگه می­دارد. همانگونه که در   
بخش ‏3-4-1- اشاره شد مقایسه فرمان فعلی و قبلی ارسالی به سطح دوم برای راه­اندازی آن اهمیت دارد که این متغیر برای این کاربرد مورد استفاده قرار می­گیرد.

متغیر دیگری که در پیاده­سازی تابع از آن استفاده شده است، ATSHA\_timer نام دارد که یک ایستا است و هدف از آن نگه داشتن زمان برنامه است. این متغییر با هر بار ایجاد تغییر در زمان سیستم، یک واحد افزایش پیاده می­کند و بنابراین می­توان از آن برای نگاه داشتن زمان WD استفاده نمود.

با در نظر گرفتن توضیحات کلی فوق، موارد زیر در پیاده­سازی سایر وضعیت­ها در نظر گرفته شده است:

* **WAKE ATSHA**: در پیاده­سازی این وضعیت و مطابق توضیحی که در مورد دیاگرام وضعیت آن داده شد، بررسی می­شود که آیا I2C وارد وضعیت NACK کردن آدرس شده است و در صورتی که این مطلب صحیح باشد، پروتکل عملیات CHECK\_ATSHA\_AWAKE را انجام خواهد داد.
* **CHECK ATSHA AWAKE:** مطابق مطلبی که در مورد دیاگرام وضعیت این حالت توضیح داده شد، در صورتی که در طول یک تیک زمانی وضعیت پروتکل از NACK کردن آدرس تغییر نکند، مجدداً وارد وضعیت WAKE ATSHA می­شویم که این امر توسط یک حلقه if بررسی می­شود. از طرف دیگر توجه داریم اندیس داده ورودی به پروتکل I2C توسط transception\_counter تعیین می­شود که علت آن با توجه به بحث­هایی که تا این نقطه شد واضح است (و همین موضوع در مورد سایر وضعیت­ها نیز رعایت می­شود).
* **READ RESPONSE:** پس از کامل شدن دریافت اولین بایت (که همان بایت count است) که این امر توسط یک حلقه if تعیین می­شود، طول پاسخ دریافتی برابر این بایت قرار داده می­شود.
* **SET ATSHA IDLE و SET ATSHA SLEEP:** پس از گذار به این وضعیت­ها، تایمر ATSHA صفر می­شود و سپس پرچم PL2\_FINISHED به بیرون ارسال می­شود تا به سطح بالاتر نشان داده شود پروتکل خاتمه یافته است.
* **PL2\_TIMEOUT:** در این وضعیت پرچم PL2\_FAILURE به بیرون ارسال می­شود تا نشان دهد عملیات سطح دوم ناموفق بوده است.

### تابع تعیین عملیات بعدی سطح دوم (PL2\_Assign\_Operation())

|  |  |
| --- | --- |
| **کارکرد:** | تعیین عملیات بعدی که باید در چرخه­کاری انجام بشود بر اساس وضعیت فعلی پروتکل |
| **ورودی­ها:** | * پرچم وضعیت فعلی I2C (I2C\_status) * اشاره­گر به متغیر از نوع command\_response\_structure (\*pcommand\_response) * طول داده­ای که قرار است با ATSHA مبادله بشود (transaction\_length) * عملیات فعلی پروتکل (operation\_current) * عملیات بعدی پروتکل (operation next) * عملیات بازگشتی (recursion\_operation) |
| **خروجی­:** | تابع فاقد خروجی می­باشد. |

**شرح تفصیلی:** هدف از این تابع، بررسی و تعیین گذار وضعیت پروتکل می­باشد، به این ترتیب این تابع در زمان اجرای تقریباً تمامی وضعیت­های سطح دوم فراخوانی می­شود. اولین مطلبی که در این تابع بررسی می­شود، عبور از زمان WD چیپ ATSHA است و در صورتی که پاسخ صحیح است، پروتکل عملیات متناظر با وضعیت PL2 TIMEOUT را اجرا خواهد کرد. در غیر این صورت بررسی می­شود که آیا عملیات I2C خاتمه یافته است یا خیر و در صورت منفی بودن پاسخ، تابع اجازه می­دهد که عملیات وضعیت فعلی ادامه پیدا کند. در صورتی که I2C خاتمه یافته باشد نیز بررسی می­شود که آیا تبادل داده با ATSHA کامل شده است و یا خیر که در صورتی منفی بودن پاسخ، وضعیت فعلی پروتکل ادامه خواهد یافت. توجه داریم تبادل داده به ATSHA یک مفهوم کلی است و شامل ارسال داده به آن و دریافت داده می­شود. طول تبادل داده با ATSHA توسط متغیر transception\_length که یکی از ورودی­های این تابع است معین می­شود. در صورتی که دریافت پاسخ کامل شده باشد، سه وضعیت پیش خواهد آمد:

* در صورتی که پروتکل در حال انجام عملیات ارسال داده به ATSHA بوده (وضعیت­های RESET\_ATSHA\_BUFFER و SEND\_COMMAND) طبق گذارهای وضعیت ارائه شده در دیاگرام وضعیت­های سطح دوم به وضعیت بعدی گذار می­کنیم.
* در صورتی که ATSHA مشغول عملیات بررسی بیدار بودن ATSHA هست و در صورتی که پاسخ دریافتی از ATSHA پاسخ به یک WAKE TOKEN باشد، به وضعیت بعدی (یعنی SEND\_COMMAND) گذار خواهیم کرد.
* در صورتی که دو مورد فوق صحیح نباشد، به این معناست که یا ارسال فرمان به ATSHA خاتمه یافته یا یک پاسخ از ATSHA دریافت شده است. فاقد از این که کدام یک از این دو مورد صحیح باشد، در این نقطه CRC داده­های موجود در آرایه response[] محاسبه می­شود (که البته این CRC محاسبه شده در حالت ارسال فرمان به چیپ کاربردی ندارد). حال:
  + اگر در وضعیت قبلی یک فرمان (یعنی ریست کردن بافر و یا هر فرمان دلخواه دیگر) به چیپ ارسال شده است، به عملیات بعدی گذار می­کنیم.
  + در غیر این صورت اگر در وضعیت قبلی یک WAKE TOKEN ارسال شده و پاسخ صحیح برای آن دریافت شده است، بازهم به عملیات بعدی گذار خواهیم کرد.
  + اگر دو مورد قبلی صحیح نباشد به این معناست که یک پاسخ از چیپ دریافت کرده­ایم. در این حالت بررسی می­شود که آیا CRC دریافتی صحیح است یا خیر و ضمناً آیا بایت وضعیت/خطای دریافتی برابر مقدار مورد نظر در متغیر command\_response است یا خیر و در صورتی که پاسخ هر دو مورد صحیح باشد به وضعیت بعدی گذار خواهیم کرد. در غیر این صورت پاسخ دریافتی برابر مقدار مورد نظر نیست و به وضعیتی که توسط recursion\_operation تعیین می­شود گذار خواهیم کرد.

برای بررسی CRC، ابتدا آخرین دو بایت دریافتی (که همان بایت­های CRC هستند) در آرایه سراسری received\_CRC\_bytes[] (به گزارش «*پروتکل ارتباطی I2C*» رجوع شود) ذخیره می­شوند. توجه داریم انجام عملیات فوق در مدهای ارسال هیچ مشکلی ایجاد نخواهد کرد، چون تمامی آرایه­ها و طول­ها از قبل تعریف شده­اند.



# پیاده­سازی پروگرامر ATSHA (نرم­افزار)

## مقدمه

در این فصل در مورد پیاده­سازی نرم­افزاری پروگرامر ATSHA بحث خواهد شد. پروگرامر ATSHA باید داده­های مورد نیاز در الگوریتم احراز هویت را در نواحی داده موجود در چیپ ATSHA (یعنی DZ و CZ) پروگرام کند. توجه داریم که از آنجا که با پیچیده­تر شدن الگوریتم­های احراز هویت وظیفه پروگرامر تغییر نخواهد کرد،   
پیاده­سازی که در ادامه برای پروگرامر خواهد آمد را می­توان با تغییرات کوچک (در سطح یک الگوریتم آن) در فازهای بعدی پروژه نیز مورد استفاده قرار داد. پروگرامر ATSHA با توجه به مطالبی که در فصل پنجم خواهد آمد با استفاده از یک میکروکنترلر ATmega8 پیاده­سازی می­شود.

## الگوریتم پروگرام

با در نظر گرفتن الگوریتم پیشنهادی برای احراز هویت، الگوریتمی که برای پروگرام ATSHA در سطح یک مورد استفاده قرار می­گیرد در شکل ‏4‑1 آمده است. الگوریتم ارائه شده در این شکل در واقع همان الگوریتم ارائه شده در شکل ‏3‑1 است که با در نظر گرفتن نحوه پیاده­سازی در ATSHA (ترتیب خواندن و نوشتن داده­ها و همینطور قفل کردن نواحی داده) اصلاح شده است. در پیاده­سازی این الگوریتم موارد اشاره شده در بخش ‏2-2- در مورد نحوه نوشتار داده در حافظه­های ATSHA مد نظر قرار گرفته است.



شکل ‏4‑1: الگوریتم پروگرام ATSHA در سطح اول پروتکل

با توجه به شکل، اولین قدمی که در سمت پروگرامر انجام می­شود تنظیم مقادیر Slot Configuration موجود در CZ برای اسلات­های صفر تا سه DZ است. علت اینکه این بیت­ها برای چهار ناحیه تنظیم می­شوند این مطلب است که از آنجا که نوشتن داده در ATSHA تنها در قالب 4 یا 32 بایتی ممکن است، مقادیر دو   
Slot Configuration به صورت همزمان با یکدیگر پیکربندی می­شود تا نوشتن داده در قالب چهار بایتی صورت بپذیرد (اگر چه از اسلات سوم DZ که متناظر با Slot Configuration 3 استفاده نمی­شود). پس از تنظیم این مقادیر نوبت به قفل کردن CZ می­رسد تا دسترسی به سایر نواحی داده ATSHA ممکن شود. در ادامه ابتدا رشته XOR (که با اسلات حاوی شماره سریال XOR می­شود) در اسلات صفر DZ ذخیره می­شود. پس از این مرحله اسلات شماره صفر CZ (که حاوی شماره سریال ATSHA هست) از آن خوانده می­شود و با XOR کردن رشته XOR با مقادیر خوانده شده از این اسلات، حاصل در اسلات شماره یک DZ ذخیره می­شود. یادآوری می­شود علت خواندن اسلات شماره صفر CZ به جای خواندن تنها شماره سریال ATSHA این است که اولاً خواندن 4 بایتی شماره سریال از قسمت­های مختلف CZ پیاده­سازی الگوریتم را پیچیده می­کند، و ضمناً نوشتن اطلاعات در ATSHA تنها در قالب 4 یا 32 بایتی ممکن است. به همین دلیل ترجیح بر خواندن تمامی 32 بایت اسلات صفر ATSHA به جای خواندن تنها بایت­های شماره سریال است. در آخرین مرحله نیز یک رشته انحرافی در اسلات شماره دو DZ ذخیره می­شود و در نهایت DZ قفل می­شود تا عملیات پروگرام ATSHA به پایان برسد. با در نظر گرفتن موارد فوق، چگونگی   
پیاده­سازی این الگوریتم با استفاده از وضعیت­های نرم­افزاری مناسب در بخش بعدی شرح داده خواهد شد.

### دیاگرام وضعیت سطح یکم پروتکل پروگرام

دیاگرام وضعیت پیاده­سازی سطح اول پروتکل پروگرامر با توجه به الگوریتم ارائه شده شکل ‏4‑1 در شکل ‏4‑2 نمایش داده شده است. قبل از توضیح در مورد کلیات این دیاگرام وضعیت لازم است اشاره شود که پیاده­سازی این الگوریتم دارای دو متغیر وضعیت یکی برای وضعیت فعلی (current\_PL1\_state) و دیگری برای وضعیت بعدی (next\_PL1\_state) است. هنگامی که قرار است یک فرمان در یک وضعیت مشخص به چیپ ارسال شود، وضعیت فعلی به PL2 BUSY CYCLE تغییر می­کند تا نشان دهند پروتکل مشغول ارسال اطلاعات به سمت ATSHA است. ضمن اینکه متغیر وضعیت بعدی نیز با توجه به فرمانی که بعد از فرمان فعلی قرار است به ATSHA ارسال شود تنظیم می­شود. به عنوان مثال اگر قرار است در وضعیت فعلی داده در اسلات صفرم DZ و سپس در اسلات یکم نوشته شود، وضعیت فعلی به PL2 BUSY CYCLE و وضعیت بعدی به WRITE DZ 107 تغییر خواهد کرد. با در نظر گرفتن این مورد، در ادامه گذار وضعیت در این پروتکل توضیح داده می­شود.

**قرارداد نام­گذاری وضعیت­ها:** قراردادی که برای نام­گذاری وضعیت­های خواندن داده از یک ناحیه داده در نظر گرفته می­شود این است که به وضعیت یک عدد سه رقمی نسبت داده می­شود که صدگان معرف شماره اسلات، ده­گان معرف اولین کلمه اسلات و یکان معرف آخرین کلمه­ای از اسلات است که خوانده می­شود. به عنوان مثال READ DZ 107 به این معناست که 8 کلمه (32 بایت) از اسلات شماره یک DZ (کلمات صفر تا هفت) خوانده می­شود و یا READ CZ 025 به این معناست که چهار کلمه از اسلات صفرم CZ (کلمات 2 تا 5) خوانده می­شود[[14]](#footnote-15). در مورد نوشتن در نواحی داده نیز تنها کلمه WRITE به جای READ مورد استفاده قرار می­گیرد. اسامی وضعیت­های مختلف سطح یک با توجه به قرارداد فوق در جدول ‏4‑1 آمده است.

جدول ‏4‑1: وضعیت­های سطح اول در پروگرامر و نماد آنها در کد

| وضعیت | نماد در کد |
| --- | --- |
| راه­اندازی الگوریتم پروگرام | PROGRAMMING INITIATION |
| بررسی پیشرفت پروگرام ATSHA | ASSESS PROGRAMMING PROGRESS |
| مشغول بودن سطح دوم پروتکل | PL2 BUSY CYCLE |
| موفق بودن پروگرام | PROGRAMMING SUCCESSFUL |
| عدم موفقیت در سطح دوم پروتکل | LEVEL TWO FAILURE |
| تنظیم SlotConfiguration0 و SlotConfiguration1 | WRITE CZ SLOT CONFIG01 |
| تنظیم SlotConfiguration2 و SlotConfiguration3 | WRITE CZ SLOT CONFIG23 |
| نوشتن داده در تمامی اسلات صفرم DZ | WRITE DZ 007 |
| نوشتن داده در تمامی اسلات یکم DZ | WRITE DZ 107 |
| نوشتن داده در تمامی اسلات دوم DZ | WRITE DZ 207 |
| خواندن اسلات صفرم CZ (حاوی شماره سریال) | READ CZ 007 |
| خواندن بایت­های قفل در CZ | READ LOCKCONFIGS |
| قفل کردن DZ توسط فرمان قفل کردن | LOCK DZ |
| قفل کردن CZ توسط فرمان قفل کردن | LOCK CZ |

توجه داریم در این سطح پروتکل هرگاه قرار باشد فرمانی به ATSHA فرستاده شود، طول پاسخ به فرمان و در کنار آن پاسخ مورد نظر نیز (که در بایت وضعیت/ خطای ATSHA معین می­شود) به سطح دوم داده می­شود. ضمن اینکه برای سطح دوم تعیین می­شود که چیپ باید پس از ارسال فرمان به حالت خواب/ بیکار برود (که این موارد توسط تابع Assign\_Response\_Params() که در بخش ‏4-4-2-3- تعریف شده است به سطح دو داده خواهند شد).

با توجه به دیاگرام وضعیت، اولین قدم پس از فراخوانی این الگوریتم وضعیت   
PROGRAMMING INITIATION که به معنای آغاز پروگرام است. با ورود به این وضعیت، پروتکل به وضعیت   
READ LOCKCONFIGS مراجعت می­کند. هدف از این وضعیت، خواندن بیت­های قفل نواحی داده از CZ است. برای توضیح اینکه چرا تعیین وضعیت این قفل­ها اهمیت دارد باید اشاره کرد که این قفل­ها معین می­کنند که پروگرام ATSHA تا کجا پیش رفته است. بنابراین اگر CZ قفل شده باشد به این معناست که پروگرام این ناحیه خاتمه یافته است و دیگر نمی­توان (نیازی نیست) در آن اطلاعات نوشت، و همینطور قفل بودن DZ مفهوم مشابهی دارد. با اتمام کار پروتکل در وضعیت READ LOCK CONFIGS، پروتکل وارد وضعیت   
ASSESS PROGAMMING PROGRESS می­شود که هدف آن بررسی بایت­های قفل نواحی است. با توجه به قفل بودن نواحی، پروتکل وارد یکی از سه وضعیت WRITE CZ SLOTCONFIG01 (ادامه الگوریتم از پروگرام CZ)، WRITE DZ 007 (ادامه الگوریتم از پروگرام DZ) یا PROGRAMMING SUCCESSFUL می­شود که مورد آخر به این معناست که پروگرام ATSHA به پایان رسیده است[[15]](#footnote-16).

با فرض اینکه پروتکل کار خود را از پیکربندی CZ ادامه می­دهد، پیکربندی این ناحیه شامل نوشتن 0x00 در SlotConfigurationها و سپس قفل کردن CZ است. مورد اول به ترتیب توسط وضعیت­های   
WRITE CZ SLOTCONFIG01 و SLOTCONFIG23 و مورد آخر توسط وضعیت LOCK CZ انجام می­شود. این سه قدم اولین وضعیت­هایی هستند که پروتکل پروگرام از آنها عبور می­کند. با اتمام پیکربندی CZ نوبت به پیکربندی DZ خواهد رسید.

برای پیکربندی DZ با گذار به وضعیت WRITE DZ 007، ابتدا رشته XOR که در رابطه (‏3‑1) آمده است در اسلات صفرم DZ نوشته می­شود. در مرحله بعدی با گذار به وضعیت READ CZ 007، اسلات صفرم CZ که حاوی شماره سریال چیپ ATSHA نیز می­باشد از CZ خوانده شده و با محاسبه حاصل XOR آن با رشته ذخیره شده در اسلات صفر DZ، مقدار آن در وضعیت WRITE DZ 107 در اسلات یکم نوشته می­شود[[16]](#footnote-17). در آخرین مرحله نیز با گذار به وضعیت WRITE DZ 207، رشته انحرافی در دومین اسلات DZ نوشته می­شود. پس از اتمام نوشتار داده­ها در اسلات­های مورد نظر DZ، نوبت به قفل کردن این ناحیه می­رسد که اینکار در وضعیت LOCK DZ صورت می­پذیرد. پس از قفل کردن DZ، پروگرام ATSHA با موفقیت انجام شده است و پروتکل سطح اول خاتمه می­یابد و وضعیت به PROGRAMMING SUCCESSFUL تغییر می­کند که در پی آن پرچم خاتمه موفقیت آمیز پروتکل به بیرون ارسال می­شود.

توجه داریم که هر زمان سطح دوم پروتکل یک پرچم عدم موفقیت به سطح یک ارسال کند، وضعیت این سطح به LEVEL TWO FAILURE تغییر خواهد کرد که در آن یک پرچم عدم موفقیت به بیرون از سطح یک ارسال می­شود و پروتکل برای استفاده بعدی به وضعیت PROGRAMMING INITIATION باز خواهد گشت.



شکل ‏4‑2: دیاگرام وضعیت تابع سطح اول الگوریتم پروگرام ATSHA

### وضعیت نامعین در سطح اول پروتکل

منظور از وضعیت نامعین در سطح اول پروتکل حالتی است که در آن مقدار متغیر current\_PL1\_state (وضعیت فعلی سطح یک) خارج از مقادیر متناظر با وضعیت­های استاندارد این سطح باشد. معیار تصمیم­گیری در مورد بازگردانی پروتکل به یک وضعیت مفهوم، قفل شدن نواحی داده ATSHA است که بررسی آن همانگونه که در بخش قبل اشاره شد توسط وضعیت READ LOCKCONFIGS صورت می­گیرد. با استفاده از این وضعیت و مشابه قبل معین می­شود که پروگرام کدام ناحیه با قفل کردن آن به پایان رسیده است، و با تعیین این مطلب می­توان پروگرام ATSHA را از ناحیه بعدی ادامه داد.

شکل ‏4‑3 نحوه گذار از یک وضعیت نامعین به وضعیت معین را نشان می­دهد. پس از ورود به یک وضعیت نامعین ابتدا پرچم خروجی سطح دوم بررسی می­شود و در صورتی که مقدار آن عدم موفقیت باشد پروتکل به PROGRAMMING FAILURE گذار می­کند، اگر مقدار آن مشغول و یا یک مقدار نامعین باشد به وضعیت   
PL2 BUSY CYCLE گذار می­کند (تا زمانی که پرچم خاتمه سطح دو به بیرون ارسال شود) و اگر مقدار آن نشان دهنده خاتمه سطح دوم باشد به وضعیت خواندن بایت­های قفل گذار می­کند. پس از خواندن بایت­های قفل، تعیین وضعیت بعدی مشابه بحثی است که در بخش قبل شد و با تعیین اینکه کدام ناحیه­ها قفل شده و یا نشده­اند، پروتکل به یکی از سه وضعیت WRITE CZ SLOTCONFIG01 (ادامه از پروگرام CZ)، WRITE DZ 007 (ادامه از پروگرام DZ) و یا PROGRAMMING SUCCESSFUL گذار می­کند.

**توجه:** روشی که برای خارج کردن سطح اول پروتکل در پروگرامر از وضعیت نامعین پیشنهاد شده را   
می­توان در پیاده­سازی­های آینده نیز بدون هیچگونه تغییر مورد استفاده قرار داد. در واقع وظیفه پروگرامر تنها نوشتن داده در یکی از سه ناحیه داده ATSHA است و بنابراین قفل کردن نواحی داده پس از نوشتار داده ضرورت دارد. به این ترتیب روش فوق یک روش مطمئن برای خارج کردن پروتکل از وضعیت نامعین است.



شکل ‏4‑3: الگوریتم پروگرام ATSHA در سطح اول پروتکل

## پیاده­سازی نرم­افزار الگوریتم

علاوه بر ماژول سطح دو و ماژول سطح سه (پروتکل I2C) و متعلقات آن، نرم­افزار شامل سه ماژول دیگر نیز می­باشد که فهرست تمامی ماژول­های مورد استفاده در جدول ‏4‑2 آمده است. توجه داریم ماژول ATSHA\_Programmer\_PL1.c حاوی پروتکل پروگرام ATSHA (سطح یک پروتکل) است که تابع ATSHA\_Programmer\_PL1() که از توابع همین ماژول است برای پروگرام ATSHA فراخوانی می­شود. در ادامه در مورد هر یک از این ماژول­ها بحث خواهد شد.

جدول ‏4‑2: ماژول­های پیاده­سازی شده در پروگرامر ATSHA

|  |  |
| --- | --- |
| ماژول | کارکرد |
| Main | ماژول اصلی برنامه پروگرامر |
| ATSHA\_Programmer\_PL1 | سطح یک پروتکل پروگرام ATSHA (حاوی الگوریتم پروگرام ATSHA) |
| ATSHA\_Command\_Assignment | تنظیم فرمان برای ارسال به چیپ |

### ماژول Main

این ماژول که با نام Main.cpp نام­گذاری شده است، شامل تابع Main() و کلیه توابع زیر مجموعه آن است که در واقع وظیفه فراخوانی سطح یک پروتکل پروگرام را دارد. ماژول Main دارای یک هدر به همین نام می­باشد که تمامی متغیرها و تعاریفی که در ماژول Main مورد استفاده قرار می­گیرند در آن آمده است. برای تعریف بعضی از متغیرهای تعریف شده در هدر این ماژول، از هدر ماژول­های بعدی نیز استفاده شده است که توضیحات در مورد آنها در ادامه خواهد آمد. فهرست توابعی که در این ماژول تعریف می­شوند در جدول ‏4‑3 آمده است. ثوابتی که در این هدر این ماژول تعریف می­شوند نیز در جدول ‏4‑4 آمده­اند. در ادامه در مورد هر یک از این توابع بحث خواهد شد.

جدول ‏4‑3: فهرست توابع تعریف شده در ماژول Main(به غیر از تابع main())

|  |  |
| --- | --- |
| تابع | کارکرد |
| Initial\_Setup | تنظیمات اولیه رجیسترها و متغیرها |
| Parameter\_Initial\_Setup | تخصیص دادن یک مقدار اولیه به متغیرها درون تابع Initial\_Setup() |
| timer\_update | به روزرسانی زمان فعلی برنامه |
| Change\_State | تابع تعیین وضعیت حلقه اصلی بر اساس وضعیت فعلی |

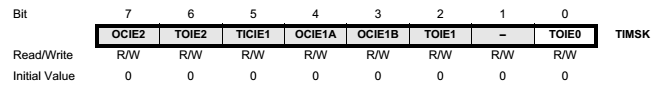
جدول ‏4‑4: ثوابت، متغیرها و متغیرهای خارجی (external) تعریف شده در هدر ماژول Main

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| ثوابت | |
| GREEN\_LED\_PORT\_DIRECTION و GREEN\_LED\_PORT\_WRITE | اسامی اختصاص یافته به DDRC و PORTC (پورت دارای LED سبز) |
| RED\_LED\_PORT\_DIRECTION و RED\_LED\_PORT\_WRITE | اسامی اختصاص یافته به DDRD و PORTD (پورت دارای LED سبز) |
| EEPROM\_ERRONEOUS\_STATE\_ADD | آدرس ذخیره وضعیت دارای خطا در EEPROM |
| START\_UP\_DELAY | تاخیر بالا آمدن پروگرامر |
| enumها | |
| Generic\_State | READY: حلقه آسنکرون آماده اجرا است  BUSY: حلقه آسنکرون مشغول است  WAIT: حلقه آسنکرون خاتمه یافته است |
| Total\_State | PROGRAM\_ATSHA: وضعیت پروگرام ATSHA  PROGRAMMING\_FINISHED: پروگرام خاتمه یافته است  PROGRAMMING\_FAILURE: عدم موفقیت در پروگرام |
| متغیرها | |
| current\_state | وضعیت فعلی تابع main() |
| status\_register | متغیر حاوی وضعیت­های حلقه آسنکرون |
| start\_up\_flag | پرچم نشان دهنده عبور از زمان بالا آمدن بورد |
| PL1\_output\_flags | پرچم خروجی سطح یک پروگرام |
| PL2\_output\_flags | پرچم خروجی سطح دو پروگرام |
| state\_timer | متغیر حاوی زمان هر وضعیت حلقه آسنکرون |
| command\_response | متغیری از نوع command\_response\_struct (جدول ‏3‑5) |

#### تابع main()

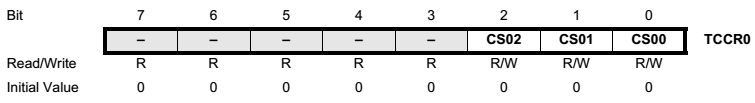
فلوچارت نحوه پیاده­سازی تابع main() پروگرامر در شکل ‏4‑6 آمده است. همانگونه که مشاهده می­شود پیاده­سازی تابع main از همان الگوریتم استاندارد پیاده­سازی نرم­افزارها (که شامل دو رجیستر tick\_timer و status\_register هست) تبعیت می­کند. برای نگه داشتن زمان تیک برنامه، در این پیاده­سازی از تایمر/کانتر صفر استفاده می­شود. تایمر کانتر صفر شامل سه رجیستر است:

* TIMSK: این رجیستر در شکل ‏4‑4 آمده است. با مساوی یک قرار دادن بیت صفرم این رجیستر، وقفه تایمر/کانتر صفر فعال خواهد شد.



شکل ‏4‑4: رجیستر TIMSK

* TCCR0: این رجیستر که در شکل ‏4‑5 آمده است در واقع مقسم فرکانسی این تایمر/کانتر است



شکل ‏4‑5: رجیستر TCCR0

* TCNT0: رجیستر شمارنده این تایمر/کانتر است که هشت بیتی می­باشد.

با در نظر گرفتن تعاریف فوق، همینطور انتخاب فرکانس کلاک میکرو برابر 4MHz و TCCR0 = 0x04 که تقسیم فرکانسی 256 را نتیجه می­دهد، زمان تیک برنامه از رابطه زیر محاسبه خواهد شد:

|  |  |
| --- | --- |
| (‏4‑1) | Tick time = |

بنابراین تیک زمانی برنامه تقریباً برابر 16 میلی­ثانیه خواهد شد. با فعال شدن وقفه این تایمر نیز مقدار متغیر status\_register به صورت استاندارد تنظیم خواهد شد.

برای پیاده­سازی الگوریتم پروگرام ATSHA از روش اول که در بخش ‏3-3-1- در مورد آن بحث شد (جداسازی سطح اول و دوم) استفاده می­شود. البته توجه داریم الگوریتمی که در بخش یاد شده در مورد آن بحث شد برای پیاده­سازی الگوریتم **احراز هویت** مورد استفاده قرار می­گیرد، اما استفاده از آن برای پروگرام نیز مانعی ندارد. با در نظر گرفتن این روش پیاده­سازی، سطح دوم پروتکل در حلقه آسنکرون و سطح یکم نیز در حلقه سنکرون (و به عبارت دقیق­تر در تابع Change\_State()) پیاده­سازی می­شود.



شکل ‏4‑6: فلوچارت تابع main()

### گذار وضعیت­ها در پروگرامر ATSHA (تابع Change\_State())

مهمترین عملیاتی که در تابع Change\_State() پروگرامر انجام می­شود، گذار وضعیت و فراخوانی سطح یک پروتکل پروگرام بر مبنای وضعیت فعلی پروگرامر است. دیاگرام وضعیت تابع main() در شکل ‏4‑7 نمایش داده شده است. پس از آغاز به کار پروگرامر، اولین وضعیتی که پروتکل وارد آن می­شود PROGRAM ATSHA است. در این وضعیت تابع سطح یک فراخوانی می­شود و پروگرام ATSHA آغاز می­شود. با فراخوانی این تابع، پرچم خروجی آن به PL1\_BUSY تغییر خواهد کرد، و تا زمانی که این پرچم مشغول باشد تابع Change\_State() وضعیت را برابر PROGRAM ATSHA تنظیم خواهد کرد. حال اگر این پرچم برابر PL1\_FINISHED بشود به این معناست که پروتکل خاتمه یافته است، و پروتکل وارد وضعیت PROGRAMMING FINISHED خواهد شد که در آن LED سبز رنگ پروگرامر به مدت 10 ثانیه روشن خواهد شد. پس از گذشت این ده ثانیه (که معادل تقریباً 670 تیک زمانی است) پروتکل مجدداً وارد وضعیت PROGRAM ATSHA می­شود. این گذار مجدد به پروگرامر اجازه می­دهد که پروگرام یک چیپ ATSHA دیگر را از سر بگیرد، و بنابراین نیازی به روشن و خاموش کردن مجدد پروگرامر برای انجام یک پروگرام جدید نخواهد بود. در نهایت در صورتی پرچم خروجی سطح یک برابر PL1\_FAILURE باشد آنگاه پروگرام ATSHA (به دلایل مختلف) دچار مشکل شده است. بنابراین پروتکل به وضعیت   
PROGRAMMING FAILURE گذار نموده که در آن LED قرمز پروگرامر به مدت 10 ثانیه روشن خواهد ماند. با خاتمه این زمان نیز پروگرامر مجدداً به وضعیت PROGRAM ATSHA گذار خواهد نمود که مشابه وضعیت PROGRAMMING FINISHED به آن اجازه می­­دهد یک چیپ را از ابتدا پروگرام کند.

توجه داریم در صورتی که پروتکل پروگرام درگیر یک وضعیت نامعین بشود، وضعیت بعدی پروتکل برابر PROGRAM ATSHA قرار داده خواهد شد و در ضمن هر دو LED پروگرامر خاموش خواهند شد. به این ترتیب در صورت ورود پروگرامر به یک وضعیت نامعین، با گذار مجدد به وضعیت PROGRAM ATSHA پروگرامر به کمک تابع سطح یک مجدداً وضعیت پروگرام را بررسی خواهد کرد و با تعیین وضعیت چیپ، به صورت خودکار وضعیت بعدی خود را تنظیم خواهد نمود.



شکل ‏4‑7: دیاگرام وضعیت تابع main()

### ماژول سطح یک پروتکل پروگرام ATSHA

این ماژول که با نام ATSHA\_Programmer\_PL1.c نام­گذاری شده است حاوی مجموعه توابع مورد نیاز برای پیاده­سازی سطح یک پروتکل پروگرام ATSHA از جمله تابع ATSHA\_Programming\_Protocol() می­باشد که برای پروگرام ATSHA در تابع main() فراخوانی می­شود. فهرست توابعی که در این ماژول تعریف می­شوند در جدول ‏4‑3 آمده است. مهمترین ثوابتی که در این هدر این ماژول تعریف می­شوند نیز در جدول ‏4‑4 آمده­اند. در ادامه در مورد کارکرد هر یک از این توابع بحث خواهد شد.

جدول ‏4‑3: فهرست توابع تعریف شده در ماژول سطح یک پروتکل پروگرام ATSHA

|  |  |
| --- | --- |
| تابع | کارکرد |
| Assign\_PL1\_States | تخصیص وضعیت در سطح یک |
| Change\_PL1\_State | تغییر وضعیت در سطح یک |
| Retrieve\_Status | بازیابی پروتکل از یک وضعیت نامعین |
| Assign\_Response\_Params | تخصیص پارامترهای پاسخ (به یک فرمان مشخص) |

جدول ‏4‑4: فهرست ثوابت/ enumها/ ساختارها/ متغیرهای خارجی تعریف شده در هدر ماژول سطح یک پروتکل پروگرام ATSHA

| ثابت / متغیر خارجی | تعریف |
| --- | --- |
| ثوابت | |
| ATSHA\_SLAVE\_ADDRESS | آدرس ATSHA در ارتباط I2C |
| PROGRAMMING\_INITIATION تا LOCK\_DZ | وضعیت­های سطح یک پروتکل |
| WCZ055\_SUCCESSFUL\_RESPONSE | مقدار بیت وضعیت/ خطای پاسخ در صورتی که فرمان به صورت موفقیت آمیز اجرا شده باشد |
| WCZ00\_RESPONSE\_LENGTH | طول پاسخ دریافتی متناظر با پاسخ ارسالی به ATSHA |
| enumها | |
| PL1\_flags | پرچم­های خروجی سطح یک پروتکل:  PL1\_BUSY: سطح یک مشغول است.  PL1\_FINISHED: سطح یک خاتمه یافته است.  PL1\_FAILURE: پروتکل موفق به پروگرام ATSHA نشده است. |
| structها | |
| PL1\_data\_struct | ساختار حاوی وضعیت­های پروتکل:  current\_PL\_state: وضعیت فعلی سطح یک  next\_PL\_state: وضعیت بعدی سطح یک |

#### تابع پیاده­سازی سطح یک پروتکل پروگرام ATSHA (ATSHA\_Programming\_Protocol())

|  |  |
| --- | --- |
| **کارکرد:** | پروگرام ATSHA |
| **ورودی­ها:** | * زمان فعلی سیستم (current\_system\_time) * پوینتر به متغیر از نوع command\_response\_struct (\*command\_response) |
| **خروجی­:** | پرچم خروجی پروتکل از نوع PL1\_flags |

**شرح تفصیلی:** این تابع وظیفه تنظیم گذار وضعیت در بالاترین سطح پروتکل پروگرام را بر عهده دارد که همانگونه که اشاره شد وضعیت فعلی و بعدی به ترتیب در دو متغیر current\_PL1\_state و next\_PL1\_state نگهداری می­شوند که نحوه تعریف آنها سراسری (global) است تا وضعیت پروتکل محفوظ بماند. پیاده­سازی حلقه اصلی تابع با استفاده از یک حلقه switch است که بر روی current\_PL1\_state نوشته می­شود. عموماً در هر وضعیت بسته به عملیاتی که قرار است انجام شود ابتدا یکی از توابع ماژول ارسال فرمان (بخش ‏4-4-3-) فراخوانی می­شود. تمامی فرمان­ها در آرایه command[] متغیر command\_response نگه­داری می­شوند که نوع آن از نوع ساختار command\_response\_struct می­باشد که شرح آن در جدول ‏3‑6 آمده است. توجه داریم این تابع پوینتر به متغیری از نوع فوق را دریافت می­کند (اصل متغیر در تابع main() تعریف شده است) تا بتوان فرمان ارسالی را به سطح دوم ارسال کرد. پس از تعیین فرمان باید پاسخ موفقیت آمیز به فرمان نیز تعیین شود که این عمل توسط تابع Assign\_Response\_Params() (بخش ‏4-4-2-3-) انجام خواهد شد. در پایان انجام عملیات هر وضعیت نیز وضعیت فعلی و بعدی تخصیص داده می­شوند. توجه داریم خروجی تابع نیز پرچم وضعیت پروتکل است که مقدار آن اشاره به سه حالت مشغول بودن پروتکل، خاتمه پروتکل و یا عدم موفقیت در آن را دارد.

در مورد پیاده­سازی وضعیت WRITE\_DZ\_107 یادآوری می­شود که پس از خواندن اسلات شماره سریال از ATSHA، مقدار آن در آرایه response در ساختار command\_response در دسترس خواهد بود که می­توان با استفاده از آن حاصل XOR را محاسبه نموده و مجدداً در ATSHA نوشت.

#### تابع تعیین وضعیت سطح یک (Assign\_PL1\_State())

|  |  |
| --- | --- |
| **کارکرد:** | تخصیص وضعیت­های فعلی و بعدی در پروتکل |
| **ورودی­ها:** | * وضعیت فعلی (current\_state) * وضعیت بعدی (next\_state) * پرچم خروجی سطح یک (output\_flag) |
| **خروجی­:** | تابع فاقد خروجی می­باشد. |

**شرح تفصیلی:** از تابع فوق برای تعیین متغیرهای وضعیت فعلی (current\_PL1\_state) و وضعیت بعدی (next\_PL1\_state) استفاده می­شود. مهمترین تفاوت تابع فوق با تابع Change\_PL1\_State() این است که این تابع وظیفه تعیین مسیر پروتکل را دارد (و مقدار دلخواهی را به وضعیت فعلی و بعدی تخصیص می­دهد) در حالی که تابع دیگر تنها مسئول گذار وضعیت است و مقدار وضعیت بعدی را به وضعیت فعلی تخصیص می­دهد. کارکرد این تابع به این صورت که ورودی­های خود را به سه متغیر از ساختار PL1\_data یعنی current\_PL1\_state، next\_PL1\_state و PL1\_output\_flag تخصیص می­دهد.

#### تابع تغییر وضعیت سطح یک (Change\_PL1\_State())

|  |  |
| --- | --- |
| **کارکرد:** | تغییر وضعیت فعلی سطح یک پروتکل برمبنای پرچم خروجی سطح دو |
| **ورودی­ها:** | پرچم خروجی سطح دو (PL2\_output\_flag) |
| **خروجی­:** | تابع فاقد خروجی می­باشد. |

**شرح تفصیلی**: از این تابع برای تعیین وضعیت بعدی پروتکل در وضعیت PL2\_BUSY\_CYCLE استفاده می­شود (با تابع تخصیص وضعیت تعریف شده در بخش ‏4-3-3-2- مقایسه شود). با ورود به این تابع پرچم خروجی سطح دوم بررسی می­شود. اگر این پرچم PL2\_BUSY باشد به این معناست که پروتکل همچنان مشغول ارسال فرمان است و بنابراین گذار وضعیتی انجام نخواهد شد. همچنین اگر پرچم برابر PL2\_READY باشد نیز نتیجه   
می­شود که ارسال فرمان فعلی به چیپ آغاز نشده و در نتیجه وضعیت پروتکل تغییر نخواهد کرد. از طرف دیگر اگر پرچم خروجی پروتکل برابر PL2\_FINISHED باشد، به این معناست که عملیات فعلی پروتکل خاتمه یافته است و در نتیجه پروتکل به وضعیت بعدی گذار خواهد نمود. در نهایت نیز اگر پرچم عدم موفقیت سطح دوم یعنی PL2\_FAILURE\_FLAG به بیرون بازگردانده شود به این معناست که یک عدم موفقیت روی داده است و در نتیجه پروتکل به وضعیت LEVEL\_TWO\_FAILURE گذار خواهد نمود.

#### تابع تعیین پاسخ موفقیت­آمیز به فرمان ارسالی (Assign\_Response\_Params())

|  |  |
| --- | --- |
| **کارکرد:** | تخصیص پارامترهای متناظر با پاسخ به فرمان ارسالی در متغیر command\_response |
| **ورودی­ها:** | اشاره­گر به متغیر از نوع command\_response\_structure (\*command\_response)  طول (length)  پاسخ موفقیت آمیز (successful\_response)  متغیر تعیین حالت خواب یا بیکار (setSHA\_sleep\_idle) |
| **خروجی­:** | تابع فاقد خروجی می­باشد. |

**شرح تفصیلی**: تابع فوق وظیفه تخصیص پارامترهای پاسخ به فرمان ارسالی به ATSHA را برعهده دارد. هر ورودی این تابع نشان دهنده یک ویژگی پاسخی است که قرار است از ATSHA دریافت شود که با دریافت آنها در ورودی تابع، هر کدام به متغیر متناظر در ساختار command\_response نسبت داده خواهند شد.

### ماژول تنظیم فرمان­های ارسالی به ATSHA

این ماژول و هدر متناظر آن برای تنظیم بافر command[] در ساختار command\_struct که در سطح دوم پروتکل مورد استفاده قرار می­گیرد تعریف شده و به صورت ATSHA\_Assign\_Command.c نام­گذاری شده است. توجه داریم این ماژول میان پروتکل احراز هویت و پروتکل پروگرام یکسان است، ضمن اینکه در پیاده­سازی فازهای بعدی پروژه نیز (با کمترین و یا حتی هیچگونه تغییرات) می­توان از این ماژول استفاده کرد. فهرست توابعی که در این ماژول تعریف شده­اند در جدول ‏3‑5 و فهرست ثوابت تعریف شده آن در جدول ‏3‑6 آمده است.

در اینجا لازم است در مورد قرارداده­های نام­گذاری در این ماژول بحث شود:

* **نام گذاری توابع ماژول:** اسم­گذاری تمامی توابع به صورت «Buffer\_نام فرمانAssign\_» است. اگر فرمان ارسالی شامل خواندن و یا نوشتن داده در یکی از نواحی داده باشد قانون زیر نیز برای *نام فرمان* رعایت می­شود:
  + در صورتی که قرار بر خواندن داده از یک ناحیه باشد، حرف R در کنار نام ناحیه خواهد آمد. ضمن اینکه سه­تایی (آخرین کلمه، اولین کلمه، شماره slot) نیز در کنار این مجموعه قرار خواهند گرفت. بنابراین خواندن تمامی داده­های اسلات اول از DZ به صورت RDZ107 و همینطور خواندن کلمه پنجم از اسلات صفرم CZ با RCZ055 نمایش داده خواهد شد.
  + در صورتی که قرار بر نوشتن داده باشد، به جای حرف R از حرف W استفاده خواهد شد.
* **نام گذاری ثوابت هدر:** برای نام­گذاری ثوابت مورد استفاده در هدر این ماژول، از قائده   
  «پارامتر فرمان\_نام فرمان» استفاده می­شود. بنابراین به عنوان مثال بایت Param20 فرمان نوشتن در اسلات دوم DZ به صورت WDZ207\_PARAM20 خواهد بود.

جدول ‏4‑5: فهرست توابع تعریف شده در ماژول تنظیم فرمان­های ارسالی به ATSHA

|  |  |
| --- | --- |
| تابع | کارکرد |
| Assign\_WCZ055\_Buffer | تنظیم بافر ارسال فرمان به نوشتن در CZ، اسلات صفر، کلمه پنج |
| Assign\_WCZ066\_Buffer | تنظیم بافر ارسال فرمان به نوشتن در CZ، اسلات صفر، کلمه شش |
| Assign\_LOCKCZ\_Buffer | تنظیم بافر ارسال فرمان به فرمان قفل کردن CZ |
| Assign\_RCZ007\_Buffer | تنظیم بافر ارسال فرمان به خواندن CZ، اسلات صفر، کلمه صفر تا کلمه هفت |
| Assign\_WDZ007\_Buffer | تنظیم بافر ارسال فرمان به نوشتن در DZ، اسلات صفر، کلمه صفر تا کلمه هفت |
| Assign\_WDZ107\_Buffer | تنظیم بافر ارسال فرمان به نوشتن در DZ، اسلات یک، کلمه صفر تا کلمه هفت |
| Assign\_WDZ207\_Buffer | تنظیم بافر ارسال فرمان به نوشتن در DZ، اسلات دو، کلمه صفر تا کلمه هفت |
| Assign\_LOCKDZ\_Buffer | تنظیم بافر ارسال فرمان به فرمان قفل کردن DZ |
| Assign\_RCZ255\_Buffer | تنظیم بافر ارسال فرمان به خواندن CZ، اسلات دو، کلمه پنج |
| Encode\_Input\_Data | کد کردن شماره سریال ATSHA |

جدول ‏4‑6: فهرست ثوابت/ ساختارها/ متغیرهای خارجی تعریف شده در هدر تنظیم فرمان­های ارسالی به ATSHA

| نام | تعریف |
| --- | --- |
| ثوابت | |
| WCZ055\_WORD\_ADDRESS تا WCZ055\_DATA13 | Word Address تا Data13 فرمان نوشتن بر روی اسلات صفر، کلمه پنج CZ |
| WCZ066\_WORD\_ADDRESS تا WCZ066\_ DATA13 | Word Address تا Data13 فرمان نوشتن بر روی اسلات صفر، کلمه شش CZ |
| LOCKCZ\_WORD\_ADDRESS تا LOCKCZ\_PARAM21 | Word Address تا Param21 فرمان قفل کردن CZ |
| RCZ007\_WORD\_ADDRESS تا RCZ007\_DATA13 | Word Address تا Data13 فرمان خواندن از اسلات صفر CZ |
| WDZ007\_WORD\_ADDRESS تا WDZ007\_DATA131 | Word Address تا Data131 فرمان نوشتن بر روی اسلات صفر DZ |
| WDZ107\_WORD\_ADDRESS تا WDZ107\_PARAM21 | Word Address تا Param21 فرمان نوشتن بر روی اسلات یک DZ |
| WDZ207\_WORD\_ADDRESS تا WDZ207\_DATA13 | Word Address تا Data13 فرمان نوشتن بر روی اسلات دو DZ |
| LOCKDZ\_WORD\_ADDRESS تا LOCKDZ\_PARAM21 | Word Address تا Param21 فرمان قفل کردن DZ |
| RCZ225\_WORD\_ADDRESS تا RCZ225\_PARAM21 | Word Address تا Param21 فرمان خواندن از اسلات دو CZ |

توجه داریم تقریباً تمامی توابع موجود در این ماژول یک عمل را انجام می­دهند که عبارت است از دریافت اشاره­گر به متغیری از نوع ساختار command\_response، تنظیم بایت­های آرایه command موجود در این ساختار و در نهایت محاسبه CRC این فرمان و قرار دادن آن در دو بایت انتهایی فرمان. تنها استثنا در این مورد یکی تابع Assign\_WDZ007\_Buffer() است که در این تابع رشته XOR رابطه (‏3‑1) قبل از نوشته شدن توسط تابع Encode\_Input\_Data() کدگذاری می­شود تا داده­ها به صورت صحیح (یعنی کد نشده تا در الگوریتم احراز هویت مشکلی ایجاد نشود) در DZ نوشته شوند. استثنای دیگر در توابع فوق نیز تابع Assign\_WDZ107\_Buffer() است. در این تابع که پس از وضعیت READ\_CZ\_007 فراخوانی می­شود (که در آن شماره سریال ATSHA خوانده   
می­شود) شماره سریال خوانده شده ابتدا توسط تابع Encode\_Input\_Data() کد می­شود و سپس با رشته XOR، XOR می­شود تا حاصل XOR ساخته شود. سپس حاصل این عملیات در بافر command[] نوشته شده و به سمت ATSHA ارسال می­شود.

در ادامه تنها در مورد تابع Encode\_Input\_Data() بحث خواهد شد.

### تابع کدکردن داده­ها (Encode\_Input\_Data())

|  |  |
| --- | --- |
| **کارکرد:** | تابعی برای کد کردن (تعویض جای 4 بیت LSB و MSB) یک رشته داده. |
| **ورودی­ها:** | * اشاره­گر به آرایه داده­ها (\*pdata) |
| **خروجی­:** | تابع فاقد خروجی می­باشد. |

**شرح تفصیلی:** تابع فوق که کارکردی بسیار ساده دارد برای رمز کردن داده­های دریافتی از اسلات صفرم CZ (شامل شماره سریال) جهت ذخیره کردن مجدد آن در چیپ استفاده می­شود. در این تابع 4 بیت پایین و بالای هر بایت دریافتی با یکدیگر جابجا شده و در همان بایت متناظر ذخیره می­شود. اینکار برای تمامی 32 بایت اسلات صفرم CZ انجام خواهد شد



# پیاده­سازی الگوریتم احراز هویت (نرم­افزار)

## مقدمه

پس از پروگرام چیپ ATSHA با استفاده از نرم­افزار ارائه شده در فصل قبل و همینطور سخت­افزار ارائه شده در فصل بعدی، احراز هویت یک میزبان را می­توان به کمک ATSHA انجام داد. در این فصل در مورد پیاده­سازی نرم­افزاری الگوریتم احراز هویت بحث می­شود. بنابراین در ابتدا در مورد دیاگرام وضعیت نرم­افزار و گذارهای وضعیت در آن بحث می­شود و در ادامه نیز در مورد ماژول­های پیاده­سازی نرم­افزار بحث خواهد شد. یادآوری می­شود که نحوه قرار دادن تابع پروتکل احراز هویت در برنامه اصلی پیاده­سازی شده در پردازنده اهمیت خاصی دارد که بحث در این مورد در بخش ‏3-3-1- آمده است.

## الگوریتم احراز هویت

الگوریتمی که برای احراز هویت ATSHA در سطح یک مورد استفاده قرار می­گیرد در شکل ‏5‑1 آمده است. الگوریتم ارائه شده در این شکل در واقع همان الگوریتم ارائه شده در شکل ‏3‑2 است که با در نظر گرفتن نحوه پیاده­سازی در ATSHA اصلاح شده است. در پیاده­سازی این الگوریتم موارد اشاره شده در بخش ‏2-2- در مورد نحوه نوشتار داده در حافظه­های ATSHA مد نظر قرار گرفته است.



شکل ‏5‑1: الگوریتم پروگرام ATSHA در سطح اول پروتکل

با توجه به شکل، الگوریتم احراز هویت شامل دو بخش کلی است:

1. **احراز هویت:** شامل مجموعه عملیات برای تایید صحت نرم­افزار مورد استفاده است. این مجموعه شامل خواندن رشته شماره سریال، رشته XOR و رشته حاصل XOR در کنار رشته انحرافی است. بر طبق بحث ارائه شده در فصل سوم، ترتیب خواندن رشته انحرافی و رشته XOR به صورت تصادفی است تا پیچیدگی الگوریتم افزایش پیدا کند.
2. **بروزرسانی:** پس از احراز هویت به کمک داده­های فعلی موجود در ATSHA و برای تغییر دادن چالش در استفاده بعدی از پروتکل احراز هویت، داده­های موجود در ATSHA بروزرسانی می­شوند. بروزرسانی ATSHA بر خلاف پروگرام آن تنها شامل نوشتن داده بر روی DZ است و کاملاً مشابه وضعیت­های ارائه شده در فصل پنجم (یعنی نوشتن داده در اسلات­های اول تا سوم DZ که به ترتیب شامل رشته XOR، حاصل XOR و رشته انحرافی هستند) می­باشد.

### دیاگرام وضعیت سطح یکم پروتکل احراز هویت

دیاگرام وضعیت پیاده­سازی سطح یک پروتکل احراز هویت با توجه به الگوریتم ارائه شده در شکل ‏3‑2 در شکل ‏5‑2 نمایش داده شده است. قبل از توضیح در مورد کلیات الگوریتم، لازم است اشاره شود که در پیاده­سازی الگوریتم احراز هویت نیز دو متغیر وضعیت، یکی برای وضعیت فعلی (current\_PL1\_state) و دیگری برای وضعیت بعدی (next\_PL1\_state) استفاده می­شود و به صورت مشابه هنگامی که قرار است یک فرمان در یک وضعیت مشخص به چیپ ارسال شود، وضعیت فعلی به PL2 BUSY CYCLE تغییر می­کند تا نشان دهند پروتکل مشغول ارسال اطلاعات به سمت ATSHA است. ضمن اینکه متغیر وضعیت بعدی نیز با توجه به فرمانی که بعد از فرمان فعلی قرار است به ATSHA ارسال شود تنظیم می­شود. ضمن اینکه یادآوری می­شود که تمامی قراردادهای نام­گذاری وضعیت­ها و توابع در این فصل مشابه قوانین ارائه شده در مورد پروگرامر (بخش ...) است. جدول ‏5‑1 فهرست تمامی وضعیت­های سطح یک احراز هویت را نمایش می­دهد.

جدول ‏5‑1: وضعیت­های سطح اول در الگوریتم احراز هویت و نماد آنها در کد

| وضعیت | نماد در کد |
| --- | --- |
| راه­اندازی الگوریتم احراز هویت | AUTHENTICATION INITIATION |
| مشغول بودن سطح دوم پروتکل | PL2 BUSY CYCLE |
| موفق بودن احراز هویت | AUTHENTICATION SUCCESSFUL |
| عدم موفقیت سطح دوم پروتکل | LEVEL TWO FAILURE |
| عدم موفقیت احراز هویت | AUTHENTICATION FAILURE |
| احراز هویت میزبان با داده­های دریافتی از ATSHA | HOST AUTHENTICATION |
| نوشتن داده در تمامی اسلات صفرم DZ | WRITE DZ 007 |
| نوشتن داده در تمامی اسلات یکم DZ | WRITE DZ 107 |
| نوشتن داده در تمامی اسلات دوم DZ | WRITE DZ 207 |
| خواندن اسلات صفرم CZ (حاوی شماره سریال) | READ CZ 007 |
| خواندن اسلات صفرم DZ | READ DZ 007 |
| خواندن اسلات یکم DZ | READ DZ 107 |
| خواندن اسلات صفرم DZ | READ DZ 207 |
| وضعیت کمکی خواندن اسلات یکم DZ | READ DZ 107a |
| وضعیت کمکی خواندن اسلات صفرم DZ | READ DZ 207a |

همانگونه که در شکل مشاهده می­شود پس از اولین ورود به الگوریتم، پروتکل وارد وضعیت AUTHENTICATION INITIATION می­شود که در آن پرچم خروجی پروتکل به مشغول و وضعیت پروتکل به READ CZ 007 تغییر می­کند. در وضعیت READ CZ 007، پروتکل ابتدا فرمان خواندن اسلات حاوی شماره سریال از CZ را در سطح دوم قرار می­دهد، و تعیین می­کند که سطح دو پس از انجام فرمان چیپ را به حالت خواب ببرد (که فرستادن چیپ به حالت خواب در تمامی فرمان­های ارسالی پیش رو نیز صادق است، بنابراین از ذکر مجدد آن خودداری می­شود). با ارسال این فرمان وضعیت فعلی پروتکل (current\_PL1\_state) برابر PL2\_BUSY\_CYCLE تنظیم می­شود که به معنای آن است که سطح دوم پروتکل مشغول به کار شده است و تا زمانی که این سطح خاتمه پیدا نکند، پروتکل به وضعیت بعدی گذار نخواهد کرد. قبل از گذار به وضعیت PL2\_BUSY\_CYCLE و برای تعیین وضعیت بعدی پروتکل (next\_PL1\_state) بایت صفرم و یکم رشته XOR با یکدیگر مقایسه می­شوند. توجه داریم در هر فراخوانی پروتکل احراز هویت، این دو بایت مقادیر متفاوتی را اختیار   
می­کنند و بنابراین مقایسه آنها باعث می­شود یک روش تصادفی برای گذار وضعیت­ها ایجاد شود. در صورتی که بایت صفرم بزرگتر از بایت یکم باشد، گذار وضعیت از مسیر اصلی و در غیر این صورت از مسیر کمکی انجام می­شود:

* **مسیر اصلی (main path):** ابتدا رشته انحرافی خوانده می­شود (وضعیت READ DZ 207)، سپس رشته XOR خوانده خواهد شد (وضعیت READ DZ 007) و سپس پروتکل رشته حاصل XOR را خواهد خواند (وضعیت READ DZ 107).
* **مسیر کمکی (auxiliary path):** ابتدا رشته XOR خوانده خواهد شد (وضعیت READ DZ 007a)، سپس رشته انحرافی خوانده می­شود (وضعیت READ DZ 207a) و سپس پروتکل رشته حاصل XOR را خواهد خواند (وضعیت READ DZ 107).

با انجام عملیات فوق، تصادفی شدن خواندن رشته­ها که در مورد علت آن در بخش **Error! Reference source not found.** بحث شد برآورده   
می­شود.

پس از عبور از مسیر اصلی یا کمکی و با در اختیار داشتن تمامی اطلاعات مورد نیاز، پروتکل وارد وضعیت HOST AUTHENTICATION می­شود که هدف از آن بررسی صحت داده­های دریافتی است. به این منظور در این وضعیت رشته XOR با رشته شماره سریال، XOR شده و حاصل با مقدار خوانده شده از ATSHA مقایسه می­شود. در صورتی که این دو مقدار برابر نباشند، نتیجه می­شود که هویت میزبان قابل تایید نیست و در نتیجه پروتکل پرچم عدم موفقیت احراز هویت را به بیرون باز می­گرداند. در غیر این صورت اما نتیجه می­شود که احراز هویت صحیح بوده، و از این نقطه به بعد پروتکل وارد وضعیت­های مرتبط با بروزرسانی داده­ها در ATSHA می­شود. قبل از ورود به این وضعیت­ها و در وضعیت احراز هویت میزبان، پروتکل رشته XOR را بر طبق الگوریتم ارائه شده در فصل سوم بروزرسانی می­کند و سپس حاصل XOR با این رشته بروز شده را نیز محاسبه خواهد کرد. یادآوری می­شود تا این نقطه پروتکل مشغول احراز هویت بوده است و پرچم خروجی آن برابر PL1\_BUSY خواهد بود.

با ورود به وضعیت­های بروزرسانی داده، پروتکل با استفاده از فرمان نوشتن، داده­های موجود در ATSHA را برای استفاده­های بعدی از الگوریتم احراز هویت بر طبق روش­های اشاره شده در فصل سوم بروزرسانی خواهد کرد. در این وضعیت­ها پرچم خروجی پروتکل برابر PL1\_UPDATE\_ATSHA خواهد شد. نکته­ای که باید در مورد این   
وضعیت­ها متذکر شد (و این امر در بخش ‏3-3-1- نحوه استفاده از پروتکل اشاره شد) این است که پس از ورود به این مرحله از پروتکل، حتماً باید اجازه داده شود که بروزرسانی تمامی داده­ها تا انتها پیش برود. این بدان معناست که به هیچ عنوان نباید پروتکل پس از ورود به این وضعیت­ها رها شود، در غیر این صورت ممکن است در استفاده­های بعدی به دلیل عدم همخوانی داده­ها، احراز هویت دچار مشکل شود. وضعیت­های بروزرسانی داده عبارتند از   
WRITE DZ 107، WRITE DZ 007 و WRITE DZ 207 که در آنها به ترتیب حاصل XOR، رشته XOR و رشته انحرافی در DZ نوشته خواهد شد. برای بروزرسانی داده­ها ابتدا حاصل XOR در ATSHA نوشته می­شود، سپس یک عدد تصادفی (سه بیت LSB اولین بایت رشته XOR) به یک شمارنده نسبت داده خواهد شد و تا زمانی که این شمارنده صفر نشود، پروتکل در قالب وضعیت WRITE DZ 207 اقدام به ارسال رشته انحرافی به سمت ATSHA خواهد کرد. با اتمام این شمارنده، وضعیت پروتکل به WRITE DZ 007 گذار می­کند و در این وضعیت رشته XOR به سمت ATSHA ارسال می­شود. با اتمام ارسال این رشته، مجدداً یک عدد تصادفی (سه بایت LSB دومین بایت رشته XOR) به یک شمارنده نسبت داده خواهند شد و مجدداً تا صفر شدن این شمارنده، پروتکل رشته­های انحرافی را در قالب وضعیت WRITE DZ 207 به چیپ ارسال خواهد کرد. به این ترتیب رشته XOR به صورت مطلوبی میان سایر رشته­ها مخفی خواهد شد.

با اتمام چرخه فوق، پروتکل وارد وضعیت AUTHENTICATION SUCCESSFUL خواهد شد که در آن پرچم موفقیت پروتکل به بیرون ارسال شده، و برای کارکردهای بعدی الگوریتم پروتکل وارد وضعیت AUTHENTICATION INITIATION خواهد شد.

توجه داریم طبق دیاگرام ارائه شده در این شکل و مشابه پروتکل پروگرام، در صورتی که سطح دوم یک پرچم عدم موفقیت به بیرون باز گرداند، پروتکل وارد وضعیت LEVEL TWO FAILURE خواهد شد که در آن پرچم عدم موفقیت پروتکل احراز هویت به بیرون بازگردانده خواهد شد. پس از انجام اینکار و برای اینکه پروتکل در استفاده بعدی دچار ابهام نشود و ضمناٌ الگوریتم احراز هویت از جایی که دچار مشکل بوده ادامه پیدا کند، وضعیت فعلی پروتکل برابر PL2 BUSY STATUS قرار داده می­شود (البته توجه داریم به دلیل ارسال پرچم عدم موفقیت به بیرون، پروتکل بلافاصله وارد این وضعیت نمی­شود). به این ترتیب در فراخوانی بعدی پروتکل، پروتکل شروع به ارسال آخرین فرمانی که اجرای آن ناموفق بوده خواهد کرد و پس از اتمام ارسال این فرمان، پروتکل به وضعیت بعدی (که توسط متغیر next\_PL1\_state از قبل از بروز عدم موفقیت در سطح دوم مشخص شده و تغییر هم نکرده است گذار خواهد نمود. در نهایت توجه داریم در وضعیت AUTHENTICATION SUCCESSFUL که آخرین وضعیت پروتکل در صورت یک احراز هویت موفقیت آمیز است، بررسی می­­شود که آیا پروتکل هیچگاه وارد وضعیت   
LEVEL TWO FAILURE شده است یا خیر. در صورتی که پاسخ مثبت باشد، احراز هویت مجدداً یکبار دیگر از ابتدا انجام خواهد شد تا مطمئن باشیم احراز هویت به صورت صحیح صورت پذیرفته است. اما در صورتی که مشخص شود پروتکل هیچگونه عدم موفقیت در سطح دوم را تجربه نکرده است، پروتکل پرچم خاتمه احراز هویت را به بیرون ارسال خواهد کرد.



شکل ‏5‑2: دیاگرام وضعیت سطح یک پروتکل احراز هویت

### وضعیت نامعین

شکل ‏5‑3 نحوه برخورد با وضعیت نامعین در سطح یک پروتکل احراز هویت را نمایش می­دهد. همانگونه که مشاهده می­شود با ورود به وضعیت نامعین، وضعیت فعلی پروتکل به PL2 BUSY CYCLE تغییر می­کند. گذار به این وضعیت به پروتکل اجازه می­دهد که اگر در حال ارسال یک فرمان به سمت ATSHA در سطح دو بوده، بتواند ارسال فرمان را خاتمه دهد. مشکل اینکار البته این است که اگر پروتکل در حال ارسال فرمان نبوده با ورود به این وضعیت مجدداً شروع به تکرار ارسال فرمان قبلی می­کند، اما ارسال مجدد یک فرمان (دست کم در فاز یک پروژه) مشکلی ایجاد نخواهد کرد. در حین تنظیم وضعیت فعلی به PL2 BUSY CYCLE، وضعیت بعدی یا next\_PL1\_state نیز تنظیم می­شود تا پس از اتمام ارسال فرمان فعلی، پروتکل وارد یک وضعیت معین شود. برای انتخاب وضعیت بعدی معیار فرمانی است که در وضعیت فعلی به ATSHA ارسال می­شود:

* اگر فرمان فعلی ارسالی از نوع خواندن داده از ATSHA بوده، به این معناست که پروتکل در حال احراز هویت بوده است. بنابراین پروتکل وارد اولین وضعیت احراز هویت یعنی READ CZ 007 می­شود.
* اگر فرمان ارسالی قبلی از نوع نوشتن بوده، آنگاه پروتکل در حال بروز کردن داده­های ATSHA بوده است. بنابراین پروتکل وارد اولین وضعیت بروزرسانی داده یعنی WRITE DZ 107 می­شود.

به این ترتیب می­توان پروتکل را از یک وضعیت نامعین خارج نمود.



شکل ‏5‑3: وضعیت نامعین در سطح یک پروتکل احراز هویت

## پیاده­سازی نرم­افزاری الگوریتم احراز هویت (ماژول­های کد)

علاوه بر ماژول سطح دو و ماژول سطح سه (پروتکل I2C) و متعلقات آن، الگوریتم احراز هویت شامل دو ماژول دیگر نیز می­باشد که فهرست تمامی ماژول­های مورد استفاده در جدول ‏5‑2 آمده است. توجه داریم ماژول ATSHA\_Authentication\_PL1.c حاوی پروتکل احراز هویت ATSHA (سطح یک پروتکل) است که تابع ATSHA\_Authentication\_PL1() که از توابع همین ماژول است برای پروگرام ATSHA فراخوانی می­شود. در ادامه در مورد هر یک از این ماژول­ها بحث خواهد شد.

جدول ‏5‑2: ماژول­های پیاده­سازی شده در الگوریتم احراز هویت توسط ATSHA

|  |  |
| --- | --- |
| ماژول | کارکرد |
| ATSHA\_Authentication\_PL1 | سطح یک پروتکل احراز هویت توسط ATSHA |
| ATSHA\_Command\_Assignment | تنظیم فرمان برای ارسال به چیپ |

### ماژول سطح اول پروتکل احراز هویت ATSHA

این ماژول و هدر متناظر آن که حاوی تمامی توابع پیاده­سازی­ مرتبط با سطح اول پروتکل احراز می­باشد به صورت ATSHA\_Authentication\_PL1.c نام­گذاری شده است. فهرست توابعی که در این ماژول تعریف شده­اند در جدول ‏5‑3 و فهرست ثوابت تعریف شده آن در جدول ‏5‑4 آمده است.در ادامه در مورد کارکرد هر یک از توابع این ماژول بحث خواهد شد.

جدول ‏5‑3: فهرست توابع تعریف شده در ماژول سطح یک پروتکل

|  |  |
| --- | --- |
| تابع | کارکرد |
| ATSHA\_Authentication\_PL1 | تابع سطح اول پروتکل پروگرام ATSHA |
| Assign\_PL1\_States | تابع تخصیص وضعیت­های فعلی و بعدی پروتکل در سطح یک |
| Change\_PL1\_State | تابع انجام گذار وضعیت در سطح یک (در وضعیت PL2\_BUSY\_CYCLE) |
| Retrieve\_Status | تابع بازیابی پروتکل از وضعیت نامعین |
| Assign\_Response\_Params | تخصیص پارامترهای پاسخ به فرمان ارسالی در متغیر command\_response |
| Array\_Manipulation | انجام عملیات­های مختلف بر روی رشته­های داده 32 بایتی |

جدول ‏5‑4: فهرست ثوابت/ enumها/ ساختارها/ متغیرهای خارجی تعریف شده در هدر سطح یک پروتکل

| نام | تعریف |
| --- | --- |
| ثوابت | |
| ATSHA\_SLAVE\_ADDRESS | آدرس slave است که ATSHA در مدهای slave ارتباط I2C به آن پاسخ  می­گوید.  **توجه:** در صورتی که نیاز باشد این آدرس تغییر کند باید در پروگرامر ATSHA و در CZ آن آدرس مطلوب را قرار داد. |
| AUTHENTICATION\_INITIATION تا READ\_DZ\_207a | وضعیت­های سطح اول پروتکل احراز هویت |
| RCZ007\_SUCCESSFUL\_RESPONSE تا WDZ207\_RESPONSE\_LENGTH | محتوای بایت خطا/وضعیت در بلوک دریافتی پاسخ به یک فرمان متناظر با پاسخ موفقیت آمیز برای فرمان­های مختلف ارسالی به ATSHA و همینطور طول مورد انتظار پاسخ به یک فرمان مشخص |
| SUCCESSFUL\_AUTHENTICATION و FAILED\_AUTHENTICATION | پرچم­های خروجی تابع احراز هویت (تابع Authenticity\_Verification()) |
| MAIN\_AUTHENTICATION\_PATH و AUXILIARY\_AUTHENTICATION\_PATH | پرچم­های نشان­دهنده مسیر اصلی و کمکی احراز هویت |
| NO\_PL2\_FAILURE\_FLAG و PL2\_FAILURE\_FLAG | پرچم­های نشان­دهنده عدم موفقیت در سطح دوم و موفق بودن سطح دوم |
| XOR\_STRING\_NOT\_WRITTEN و XOR\_STRING\_WRITTEN | پرچم نشان­دهنده نوشته شدن رشته XOR پس از ورود به وضعیت متناظر با آن |
| AUTHENTICITY\_VERIFICATION  SWAP\_CONTENT  ENCODE\_DECODE\_DATA | فرمان­هایی که تابع Array\_Manipulation() قادر به اجرای آنهاست (به ترتیب): احراز هویت – جابجایی محتوای دو رشته – رمزگذاری و رمزگشایی از یک رشته |
| WC\_DATA1\_OFFSET | شماره اولین بایتی در رشته فرمان ارسالی به ATSHA |
| enumها | |
| PL1\_flag | پرچم­های خروجی سطح اول شامل:  PL1\_BUSY: مشغول بودن سطح اول  PL1\_UPDATE\_ATSHA: بروزرسانی ATSHA پس از خاتمه احراز هویت  PL1\_FINISHED: سطح یک خاتمه یافته است  PL1\_FAILURE: عدم موفقیت سطح یک |
| structها | |
| PL1\_data\_struct | Current\_PL1\_state: وضعیت فعلی سطح یک  Next\_PL1\_state: وضعیت بعدی سطح یک  diversion\_write\_counter: شمارنده تعداد دفعاتی که رشته انحرافی در ATSHA نوشته می­شود.  encoded\_CZ007data[]: آرایه نگهدارنده رشته شماره سریال کد شده  XORstring[]: آرایه نگهدارنده رشته XOR |
| PL1\_internal\_flags\_struct | PL1\_output\_flag: پرچم خروجی سطح یک  authentication\_flag: پرچم بررسی صحت احراز هویت (پرچم خروجی تابع Array\_Manipulation())  write\_DZ\_007\_flag: پرچم نشان دهنده اینکه آیا رشته XOR نوشته شده است یا خیر.  PL2\_failure\_flag: پرچم نشان دهنده رخداد عدم موفقیت سطک دوم در سطح یک  authentication\_path: پرچم نشان دهنده مسیر انتخاب شده برای احراز هویت (مسیر اصلی یا کمکی) |

#### تابع پیاده­سازی سطح یک پروتکل احراز هویت (ATSHA\_Authentication\_PL1())

|  |  |
| --- | --- |
| **کارکرد:** | پیاده­سازی سطح یک الگوریتم احراز هویت (تابع اصلی فراخوانی الگوریتم احراز هویت) |
| **ورودی­ها:** | * اشاره­گر به متغیر از نوع command\_response\_srtuct (\*pcommand\_response) * زمان فعلی سیستم (شمارنده تیک زمانی برنامه) |
| **خروجی­:** | enum از نوع PL1\_flags، نشان دهنده وضعیت فعلی سطح یک |

**شرح تفصیلی:** این تابع حاوی بالاترین سطح الگوریتم احراز هویت می­باشد و پیاده­سازی آن برمبنای دیاگرام وضعیت ارائه شده در شکل ‏5‑2 است. این تابع از یک متغیر از نوع PL1\_data\_struct (جدول ‏5‑4) و یک متغیر از نوع PL1\_internal\_flags\_struct استفاده می­کند که پارامترهای تعریف شده دومین ساختار به صورت زیر مورد استفاده قرار می­گیرند:

* متغیر write\_dz\_007\_flag: پرچمی که مشخص می­کند آیا رشته XOR بروز شده در زمان بروزرسانی داده­ها در ATSHA نوشته شده است یا خیر.
* متغیر authentication\_path: متغیری که مسیری که برای احراز هویت طی می­شود (مسیر اصلی و یا کمکی) را در خود نگه می­دارد.
* متغیر PL2\_failure\_flag: پرچمی که نشان می­دهد یک عدم موفقیت در سطح دوم رخ داده است.

تابع با استفاده از متغیرهای فوق در کنار متغیرهای وضعیت فعلی سطح یک (current\_PL1\_state) و همینطور وضعیت بعدی سطح یک (next\_PL1\_state)، احراز هویت و همینطور بروزرسانی داده­های ATSHA را پیش می­برد. تابع توسط یک حلقه switch پیاده­سازی می­شود که بر روی متغیر وضعیت فعلی (current\_PL1\_state) بسته شده است. توجه داریم هرگاه یک چیپ فرمان مشخص را در یک وضعیت به سمت ATSHA می­فرستد و پس از اتمام ارسال این فرمان، بافر مربوط به پاسخ دریافتی از ATSHA در ساختار command\_response حاوی پاسخ دریافتی به فرمان خواهد بود. بنابراین استخراج پاسخ به یک فرمان در وضعیت کنونی پروتکل در وضعیت بعدی آن قابل مشاهده خواهد بود. به عنوان مثال وضعیت پس از فرمان READ DZ 007 (پس از عبور از وضعیت PL2 BUSY CYCLE)، وضعیت READ DZ 107 است. بنابراین پاسخ دریافتی به فرمان خواندن رشته XOR که در وضعیت READ DZ 007 ارسال شده است در وضعیت READ DZ 107 در بافر پاسخ (response) ساختار command\_response موجود خواهد بود.

با در نظر گرفتن موارد فوق، برای درک بهتر چگونگی کارکرد این تابع در ادامه در مورد نحوه پیاده­سازی بعضی از وضعیت­ها بحث خواهد:

* **وضعیت READ\_CZ\_007:** پس از قرار دادن فرمان خواندن CZ در بافر فرمان (command) متغیر command\_response توسط تابع Assign\_ReadCommand\_Buffer()، نوبت به تعیین مسیر اصلی و یا کمکی گذار وضعیت احراز هویت مطابق با دیاگرام وضعیت شکل ‏5‑2 می­رسد. برای تعیین مسیر، با استفاده از یک حلقه if تعیین می­شود که آیا بایت صفرم آرایه XORstring[] از بایت یکم آن بزرگتر است یا خیر و در صورتی که پاسخ مثبت باشد، مسیر اصلی احراز هویت انتخاب می­شود (یعنی وضعیت بعدی READ DZ 207 خواهد بود)، و در غیر این صورت مسیر بعدی برابر READ DZ 007a خواهد بود که مسیر کمکی است.
* **وضعیت READ\_DZ\_207**: ورود این وضعیت به این معناست که مسیر اصلی احراز هویت انتخاب شده است، در این وضعیت ابتدا مقدار شماره سریال که در وضعیت قبلی (READ DZ 207) خوانده شده و اکنون در بافر response ساختار command\_response موجود است در آرایه encoded\_CZ007data[] ذخیره می­شود. اما قبل از اینکار محتوای بافر response به صورتی که در بخش ‏3-2-1- اشاره شد رمزگشایی می­شود تا بتوان الگوریتم احراز هویت را طبق بخش ‏3-2-1- انجام داد.
* **وضعیت READ DZ 007:** از آنجا که در وضعیت قبلی رشته انحرافی خوانده شده است، در این مرحله نیازی به ذخیره­ کردن هیچگونه داده­ای نیست. فرمان خواندن رشته XOR در این وضعیت به سمت ATSHA فرستاده خواهد شد.
* **وضعیت READ DZ 007a:** در صورتی که گذار وضعیت از مسیر کمکی انتخاب شود، پروتکل پس از وضعیت READ\_CZ\_007 ابتدا وارد این وضعیت می­شود. بنابراین محتوای بافر response در این نقطه حاوی شماره سریال خواهد بود که پس از رمز شدن در آرایه encoded\_CZ007data[] ریخته خواهد شد.
* **وضعیت READ DZ 207a:** با ورود به این وضعیت پس از وضعیت READ DZ 007a، محتوای رشته XOR در بافر response متغیر command\_response وجود دارد که می­توان ان را برداشت نمود.
* **وضعیت READ DZ 107:** پس از طی مسیر اصلی و یا کمکی، پروتکل به وضعیت فوق خواهد رسید. توجه داریم اگر پروتکل از مسیر اصلی به این وضعیت رسیده باشد، آنگاه رشته XOR در وضعیت قبلی قرائت شده است، بنابراین در ابتدای این وضعیت این مسئله متغیر authentication\_path بررسی   
  می­شود و در صورتی که پاسخ صحیح باشد، رشته موجود در بافر response پس از رمزگشایی در آرایه XORstring[] ریخته خواهد شد.
* **وضعیت HOST AUTHENTICATION:** با ورود به این وضعیت ابتدا حاصل XOR که در وضعیت قبلی خوانده شده است رمزگشایی می­شود. سپس با فراخوانی تابع Array\_Manipulation() احراز هویت با استفاده از رشته XOR، حاصل XOR و شماره سریال انجام خواهد شد. در صورتی که احراز هویت موفقیت آمیز باشد، ابتدا رشته XOR طبق الگوریتم ارائه شده در بخش ‏3-2-1- بروزرسانی   
  می­شود و سپس حاصل XOR با کمک تابع Array\_Manipulation() محاسبه می­شود؛ پس از انجام عملیات فوق نیز گذار به وضعیت­های بروزرسانی ATSHA انجام خواهد شد. توجه داریم که برای   
  صرفه­جویی در حافظه مورد استفاده در برنامه، رشته حاصل XOR در یک آرایه جداگانه ذخیره نخواهد شد، بلکه مستقیماً در بافر command متغیر command\_response ریخته خواهد شد. در صورتی که احراز هویت موفقیت آمیز نباشد نیز پروتکل وارد وضعیت AUTHENTICATION\_FAILURE   
  می­شود.
* **وضعیت WRITE DZ 107:** قبل از نوشتن حاصل XOR بروز شده در ATSHA، ابتدا رشته آن توسط تابع Array\_Manipulation() بروز می­شود. سپس با تخصیص پارامترهای مناسب، ارسال فرمان نوشتن این رشته در ATSHA تنظیم خواهد شد. همچنین متغیر diversion\_write\_counter که نشان دهنده تعداد دفعاتی است که یک رشته انحرافی قبل از ارسال رشته XOR در ATSHA نوشته خواهد شد برابر سه بیت LSB بایت صفر قرار خواهد گرفت.
* **WRITE DZ 207:** حلقه if که در این پیاده­سازی این وضعیت مورد بررسی قرار گرفته است بررسی می­کند که آیا تعداد دفعات نوشتن رشته انحرافی خاتمه یافته است یا خیر. در صورتی که پاسخ صحیح باشد، بررسی می­شود که آیا رشته XOR (در وضعیت بعدی) در ATSHA نوشته شده است و یا خیر. در صورتی که پاسخ منفی باشد پروتکل به وضعیت WRITE DZ 007 گذار خواهد کرد و در غیر این صورت پروتکل به وضعیت AUTHENTICATION\_\_SUCCESSFUL گذار خواهد نمود.
* **وضعیت PL2 BUSY CYCLE:** همانگونه که در پیاده­سازی نرم­افزاری نیز مشاهده می­شود تابع ATSHA\_PL2() در این وضعیت نیز فراخوانی می­شود. علت این امر در واقع ممکن ساختن پیاده­سازی پروتکل از طریق روش دوم است که در آن لازم است سطح دوم پروتکل از داخل سطح یک و در این وضعیت فراخوانی شود (بخش ‏3-3-1-). توجه داریم فراخوانی این تابع در این وضعیت مشکلی در پیاده­سازی روش اول ایجاد نخواهد کرد. تابع دیگری که در این وضعیت فراخوانی می­شود (یعنی Change\_PL1\_State()) برای گذار وضعیت در سطح یک مورد استفاده قرار می­گیرد که در بخش   
  ‏5-3-1-2- در مورد آن بحث خواهد شد.

#### تابع تغییر وضعیت در سطح یک پروتکل (Change\_PL1\_State())

|  |  |
| --- | --- |
| **کارکرد:** | تغییر وضعیت سطح یک پس از ورود به وضعیت PL2\_BUSY\_CYCLE |
| **ورودی­ها:** | * پرچم خروجی سطح دوم پروتکل (PL2\_output\_flag) |
| **خروجی­:** | تابع فاقد خروجی می­باشد. |

**شرح تفصیلی**: از این تابع برای تعیین وضعیت بعدی پروتکل در وضعیت PL2\_BUSY\_CYCLE استفاده می­شود (با تابع تخصیص وضعیت تعریف شده در بخش ‏5-3-1-5- مقایسه شود). با ورود به این تابع پرچم خروجی سطح دوم بررسی می­شود. اگر این پرچم PL2\_BUSY باشد به این معناست که پروتکل همچنان مشغول ارسال فرمان است و بنابراین گذار وضعیتی انجام نخواهد شد. همچنین اگر پرچم برابر PL2\_READY باشد نیز نتیجه می­شود که ارسال فرمان فعلی به چیپ آغاز نشده و در نتیجه وضعیت پروتکل تغییر نخواهد کرد. از طرف دیگر اگر پرچم خروجی پروتکل برابر PL2\_FINISHED باشد، به این معناست که عملیات فعلی پروتکل خاتمه یافته است و در نتیجه پروتکل به وضعیت بعدی گذار خواهد نمود. در نهایت نیز اگر پرچم عدم موفقیت سطح دوم یعنی PL2\_FAILURE\_FLAG به بیرون بازگردانده شود به این معناست که یک عدم موفقیت روی داده است و در نتیجه پروتکل به وضعیت LEVEL\_TWO\_FAILURE گذار خواهد نمود.

#### تابع تعیین پارامترهای پاسخ یک فرمان (Assign\_Response\_Params())

|  |  |
| --- | --- |
| **کارکرد:** | تخصیص پارامترهای پاسخ به یک فرمان مشخص |
| **ورودی­ها:** | * اشاره­گر به متغیر از نوع command\_response\_srtuct (\*pcommand\_response) * طول پاسخ (length) * پاسخ موفق (مقدار بایت وضعیت/خطای پاسخ) (successful\_response) * متغییر تعیین فرستادن چیپ به حالت خواب/ بیکار (setSHA\_sleep\_idle) |
| **خروجی­:** | تابع فاقد خروجی می­باشد. |

**شرح تفصیلی:** این تابع در هر وضعیتی که قرار است یک فرمان به سمت ATSHA فرستاده شود فراخوانی می­شود و سه پارامتر را در ساختار command\_response مقدار­دهی می­کند:

* response\_status\_error: پارامتری که تعیین می­کند مقدار بایت خطا/ وضعیت در پاسخ موفقیت آمیز به فرمان فعلی چه خواهد بود.
* response\_length: پارامتری که تعیین می­کند طول پاسخ به فرمان فعلی چقدر باید باشد.
* sleep\_idle: پارامتری که تعیین می­کند چیپ پس از اجرای فرمان باید به وضعیت خواب و یا بیکار برود.

#### بازیابی وضعیت (Retrieve\_Status())

|  |  |
| --- | --- |
| **کارکرد:** | تخصیص پارامترهای پاسخ به یک فرمان مشخص |
| **ورودی­ها:** | * اشاره­گر به متغیر از نوع command\_response\_srtuct (\*pcommand\_response) |
| **خروجی­:** | تابع فاقد خروجی می­باشد. |

**شرح تفصیلی:** هدف از این تابع، تعیین وضعیت الگوریتم در صورت گم شدن وضعیت در پروتکل و بر مبنای دیاگرام وضعیت ارائه شده در شکل ‏5‑3 است. با ورود به این وضعیت، ابتدا مقدار وضعیت فعلی الگوریتم برابر PL2\_BUSY\_CYCLE قرار داده می­شود. این عمل باعث می­شود که در بازگشت مجدد به تابع سطح یک، ارسال فرمانی که در شرایط فعلی در بافر مانده است خاتمه پیدا کند. برای تعیین وضعیت بعدی که پروتکل به آن گذار خواهد کرد نیز با استفاده از یک حلقه if محتوای بایت سوم بافر command که همان opcode فرمان ارسالی قبلی به چیپ است بررسی خواهد شد. در صورتی که فرمان ارسالی به چیپ از نوع خواندن بوده، وضعیت بعدی برابر   
READ CZ 007 خواهد شد و در صورتی که فرمان برابر نوشتن باشد، وضعیت بعدی برابر WRITE DZ 107 خواهد شد.

#### تابع تخصیص وضعیت در سطح یک (Assign\_PL1\_States())

|  |  |
| --- | --- |
| **کارکرد:** | تخصیص وضعیت­های فعلی و بعدی در پروتکل |
| **ورودی­ها:** | * وضعیت فعلی (current\_state) * وضعیت بعدی (next\_state) * پرچم خروجی سطح یک (output\_flag) |
| **خروجی­:** | تابع فاقد خروجی می­باشد. |

**شرح تفصیلی:** از تابع فوق برای تعیین متغیرهای وضعیت فعلی (current\_PL1\_state) و وضعیت بعدی (next\_PL1\_state) استفاده می­شود. مهمترین تفاوت تابع فوق با تابع Change\_PL1\_State() این است که این تابع وظیفه تعیین مسیر پروتکل را دارد (و مقدار دلخواهی را به وضعیت فعلی و بعدی تخصیص می­دهد) در حالی که تابع دیگر تنها مسئول گذار وضعیت است و مقدار وضعیت بعدی را به وضعیت فعلی تخصیص می­دهد. کارکرد این تابع به این صورت که ورودی­های خود را به سه متغیر از ساختار PL1\_data یعنی current\_PL1\_state، next\_PL1\_state و PL1\_output\_flag تخصیص می­دهد.

#### تابع انجام عملیات بر روی رشته­های داده (Array\_Manipulation())

|  |  |
| --- | --- |
| **کارکرد:** | احراز هویت از طریق رشته­های داده دریافتی از ATSHA |
| **ورودی­ها:** | * پوینتر به آرایه اول (\*pvec1) * پوینتر به آرایه دوم (\*pvec2) * پوینتر به آرایه سوم (\*pvec3) * فرمان مورد نظر (input\_command) * بایت XOR که یک رشته با آن XOR می­شود (XOR\_byte) |
| **خروجی­:** | پرچم نشان دهنده حاصل احراز هویت   * SUCCESSFUL\_AUTHENTICATION: احراز هویت موفق بوده است * FAILED\_AUTHENTICATION: احراز هویت ناموفق بوده است |

**­شرح تفصیلی:** تابع فوق برای انجام سه عملیات مختلف بر روی یک، دو یا سه رشته داده مورد استفاده قرار می­گیرد:

* **مد فرمان AUTHENTICITY\_VERIFICATION:** XOR کردن دو رشته با یکدیگر (\*pvec1 و \*pvec2) و مقایسه حاصل آن با یک رشته سوم (\*pvec3) جهت احراز هویت و بازگرداندن یک پرچم موفقیت در احراز هویت در صورت برابر بودن تمامی عناصر رشته.
* **مد فرمان SWAP\_CONTENT:** جابجا کردن محتوای دو رشته داده با همدیگر (\*pvec1 و \*pvec2)
* **ENCODE\_DECODE\_DATA:** رمزگذاری و رمزگشایی یک رشته با الگوریتم ارائه شده در بخش ‏3-2-1-.

با ارسال هر یک از سه مد فوق به تابع توسط متغیر input\_command، تابع عملیات متناظر با آن را با استفاده از پوینترهایی انجام می­دهد که به انجام آن عملیات تخصیص یافته­اند. یادآوری می­شود که اگر یک پوینتر در یک عملیات مشخص کاربرد ندارد، مقدار آن در ورودی تابع برابر صفر قرار داده می­شود. در مورد مد AUTHENTICITY\_VERIFICATION نیز توجه داریم تابع علاوه بر مقایسه حاصل XOR دو رشته با رشته سوم (یعنی \*pvec3)، حاصل XOR را نیز در همین آرایه ذخیره خواهد کرد. اینکار باعث حذف نیاز به بعضی دیگر توابع خواهد شد.

### ماژول تنظیم فرمان­های ارسالی به ATSHA

این ماژول و هدر متناظر آن برای تنظیم بافر command[] در ساختار command\_struct که در سطح دوم پروتکل مورد استفاده قرار می­گیرد تعریف شده و به صورت ATSHA\_Assign\_Command.c نام­گذاری شده است. فهرست توابعی که در این ماژول تعریف شده­اند در جدول ‏5‑5 و فهرست ثوابت تعریف شده آن در جدول ‏5‑6 آمده است.

جدول ‏5‑5: فهرست توابع تعریف شده در ماژول تنظیم فرمان­های ارسالی به ATSHA

|  |  |
| --- | --- |
| تابع | کارکرد |
| Assign\_ReadCommand\_Buffer | تنظیم بافر فرمان به یک فرمان خواندن از ATSHA |
| Assign\_WriteCommand\_Buffer | تنظیم بافر فرمان به یک فرمان نوشتن در ATSHA |

جدول ‏5‑6: فهرست ثوابت/ ساختارها/ متغیرهای خارجی تعریف شده در هدر تنظیم فرمان­های ارسالی به ATSHA

| نام | تعریف |
| --- | --- |
| ثوابت | |
| NO\_COMMAND | یک تعریف برای فرمان تهی (یعنی بافر command فاقد فرمان است) |
| RCZ007\_COMMAND تا RCZ007\_PARAM21 | نام فرمان تا Param21 فرمان خواندن اسلات صفر CZ |
| RDZ007\_COMMAND تا RDZ007\_PARAM21 | نام فرمان تا Param21 فرمان خواندن اسلات صفر DZ |
| RDZ107\_COMMAND تا RDZ107\_PARAM21 | نام فرمان تا Param21 فرمان خواندن اسلات یک DZ |
| RDZ207\_COMMAND تا RDZ207\_PARAM21 | نام فرمان تا Param21 فرمان خواندن اسلات دو DZ |
| WDZ007\_COMMAND تا WDZ007\_PARAM21 | نام فرمان تا Param21 فرمان نوشتن در اسلات صفر DZ |
| WDZ107\_COMMAND تا WDZ107\_PARAM21 | نام فرمان تا Param21 فرمان نوشتن در اسلات یک DZ |
| WDZ207\_COMMAND تا WDZ207\_PARAM21 | نام فرمان تا Param21 فرمان نوشتن در اسلات دو DZ |
| RW\_WORD\_ADDRESS | Word Address مورد استفاده برای فرمان خواندن و نوشتن |
| READ\_OPCODE و WRITE\_OPCODE | کد عملیاتی فرمان خواندن و نوشتن داده در ATSHA |

توجه داریم کارکرد هر دو تابع معرفی شده در این ماژول ساده و مشابه توابع بخش ‏4-4-3- می­باشد. تابع Assign\_WriteCommand\_Buffer() برای تعریف فرمان نوشتن در ماژول مورد استفاده قرار می­گیرد. ضمن اینکه این تابع بایت­های داده فرمان را نیز به عنوان ورودی دریافت می­کند. از طرف دیگر تابع Assign\_ReadCommand\_Buffer() نیز برای تعریف یک فرمان خواندن در ATSHA استفاده می­شود.



# سخت­افزار

## مقدمه

در انتها هم یکدور چک می­کند الگوریتم را و در صورتی که صحیح باشد، یک LED را روشن می­کند.

الان خط I2C چه ولتاژی باید داشته باشد؟ فعلاً لااقل به نظر 2.7v؟

*آقا مقاومت­های پول آپ در I2C را در پروگرامر بگذاریم یا در بورد هست؟ یا هر دو ولی مثلا سوئیچ داشته باشه؟ بعد اینکه پول آپ روی خط ATSHA که در بورد هست مشکلی ایجاد میکنه یا خیر؟*

*حتماً جک ATmega به جای خود آن نصب شود تا بتوان پروگرامر را گذاشت و برداشت؟*

*تغذیه را خود پروگرامر بدهد؟*

## پروگرامر ATSHA

# مراجع



# پيوست – الف – کارهای آینده

## الف-1- تولید CRC برای یک بردار داده دلخواه

برای تولید CRC برای یک رشته باینری دلخواه می­توان از برنامه متلب CRC\_Generator.m که در بسته متلب این پروژه آمده استفاده کرد. چگونگی پیاده­سازی این الگوریتم در ادامه خواهد آمد.

### الف-1-1 نحوه تولید CRC توسط Matlab

برای تولید CRC اعمال زیر به ترتیب انجام می­شود:

1. یک مولد تولید کد CRC با استفاده از دستور comm.CRCGenerator() تولید می­شود. این تابع متعلق به جعبه ابزار مخابراتی متلب است که مدهای کاری مختلفی دارد. در این پیاده­سازی، این تابع یک رشته باینری (متناظر با چند جمله­ای CRC) را به عنوان ورودی دریافت می­کند و به عنوان خروجی مولد CRC را باز می­گرداند.
2. با استفاده از تابع step() (که برای تولید پاسخ پله یک سیستم استفاده می­شود) رشته داده­ای که کد CRC به آن چسبانده شده تولید می­گردد. این تابع شی مولد CRC و همینطور رشته باینری متناظر با پیام را به عنوان ورودی دریافت می­کند.

کد CRC\_Generator.m با استفاده از دو قدم فوق CRC را تولید می­کند. به این منظور ابتدا رشته متناظر با نمایش مبنای 16 پیام در متغیر hex\_codeword\_string نوشته می­شود و ضمناً چند جمله­ای مولد نیز در hex\_CRC\_polynomial نوشته می­شود. با تبدیل این دو مقدار به مقادیر باینری توسط تابع hexToBinaryVector()، مقادیر باینری به ورودی­های تابع مولد CRC و همینطور تابع پاسخ پله داده می­شوند. در نهایت پس از تولید پاسخ، تنها بایت­های CRC در خروجی انتخاب شده و این مقادیر توسط پیام در مبنای 16 نمایش داده خواهند شد.



# پيوست - ب

## ب-1- مقدمه

1. Data Zone [↑](#footnote-ref-1)
2. Configuration Zone [↑](#footnote-ref-2)
3. One-Time Programmable Zone [↑](#footnote-ref-3)
4. توجه داریم هنگامی که مد کاری ارتباط I2C سمت میزبان به MR تغییر می­کند، یک فرمان خواندن را به سمت ATSHA می­فرستد (با ارسال یک در بیت هشتم آدرس Slave، به گزارش «*پروتکل ارتباطی I2C*» رجوع شود). به همین دلیل تنها تغییر مدکاری ارتباط I2C کافیست تا بتوان داده­های بافر ATSHA را قرائت نمود. [↑](#footnote-ref-4)
5. این پاسخ در پروژه فعلی کاربردی ندارد و فقط برای کامل بودن اطلاعات آورده شده است. [↑](#footnote-ref-5)
6. مقدار 8MHz در صورت استفاده از میکروکنترلرهای ATmega معتبر است و بنابراین نیازی به دستکاری مقادیر این رجیسترها در هنگام کار با این چیپ­ها نیست. اما در صورت استفاده از سایر میکروکنترلرهایی که فرکانس کاری بالاتری دارند، لازم است مقادیر رجیسترها به صورت مناسبی اصلاح شوند. [↑](#footnote-ref-6)
7. Data2 تنها در صورتی مقداردهی می­شود که قرار باشد داده به صورت رمز شده نوشته شود. در غیر این صورت (مشابه این پیاده­سازی) مقداردهی نمی­شود. [↑](#footnote-ref-8)
8. ممکن است این داده برای DZ رمز شده باشد. داده­ای که قرار است در CZ نوشته شود هیچگاه رمز شده نیست. [↑](#footnote-ref-9)
9. همانگونه که در دیتاشیت مرجع (1) نیز اشاره شده است، بهتر است نواحی داده پس از بررسی CRC قفل شوند تا احتمال وجود خطا در آنها کمینه شود. به همین دلیل در دیتاشیت نیز توصیه می­شود که حتماً CRC قبل از قفل کردن ناحیه بررسی شود. [↑](#footnote-ref-10)
10. از آنجا که داده­هایی که قرار است در سه ناحیه داده نوشته شوند از قبل معلوم هستند، یک راه حل مناسب برای مقایسه CRC بدون افزایش بار محاسباتی، محاسبه CRC داده­هایی که قرار است در سه ناحیه ریخته شوند در خارج از برنامه است. این مطلب بایستی در پیاده­سازی­های آینده پروگرامر ATSHA مد نظر قرار بگیرد. [↑](#footnote-ref-11)
11. این مطلب نیازمند جستجوی دقیق­تر و موشکافی است. باید بررسی شود که آیا نامحدود انتخاب کردن زمان احراز هویت و پیاده­سازی همروند آن با سایر قسمت­های برنامه باعث ایجاد مشکلات و راههای نفوذ در الگوریتم خواهد شد یا خیر. [↑](#footnote-ref-12)
12. اگر به دیاگرام وضعیت شکل ‏5‑1 که متعلق به پروتکل احراز هویت است رجوع شود مشاهده می­شود که در سطح یک پس از تنظیم فرمان، پروتکل همیشه وارد وضعیت PL2 BUSY CYCLE می­شود که در آن وضعیت سطح دوم بررسی می­شود. به این ترتیب در بین هر بار فراخوانی سطح یک، همیشه در یک تیک زمانی سطح دوم بیکار خواهد ماند، چون لازم است سطح یک از وضعیت PL2 BUSY CYCLE به یک وضعیت دیگر گذار کرده و بافر فرمان را تنظیم کند. به این ترتیب مشکلی که در بالا اشاره شد یکی از موارد غیر قابل رفع در پیاده­سازی الگوریتم به صورت فوق است. [↑](#footnote-ref-13)
13. توجه داریم طول فرمان ارسالی همیشه در بایت دوم فرمان وجود دارد! [↑](#footnote-ref-14)
14. توجه داریم که آدرس­دهی CZ نیز بر اساس شماره اسلات و همینطور شماره کلمه است، اگر چه در تقسیم­بندی ارائه شده در بخش ‏2-2-2-، CZ تنها با واحد کلمه تعریف شده است. بنابراین به عنوان مثال CZ025 معادل بایت­های دوم شماره سریال تا بایت یکم revision در CZ است. یادآوری می­شود که دستور خواندن (و به طور خاص پارامتر دوم آن) ایجاب می­کند که آدرس­دهی به ترتیب فوق انجام شود. [↑](#footnote-ref-15)
15. در توضیح اینکه چرا اولین وضعیت بررسی قفل نواحی است ذکر این نکته نیز ضروری می­باشد که اگر پروگرام در نوبت قبلی استفاده از پروگرامر به هر دلیلی ناموفق باشد (به عنوان مثال قطع و وصل شدن پروگرامر در حین کار، قطع شدن جریان برق و غیره)، شروع از این وضعیت تعیین می­کند که پروگرام در دفعه قبلی تا چه مرحله­ای پیش رفته است. عدم انجام اینکار می­تواند باعث ایجاد مشکل در زمان پروگرام نواحی بشود که از قبل قفل   
    شده­اند. [↑](#footnote-ref-16)
16. یادآوری می­شود که همانگونه که اشاره شد، منظور از حاصل XOR در واقع حاصل عملیاتی است که بر روی شماره سریال و همینطور رشته XOR برای پروتکل احراز هویت انجام می­شود. [↑](#footnote-ref-17)