آموزش طراحی بوت لودر برای میکروکنترلرها در سیستم های نهفته

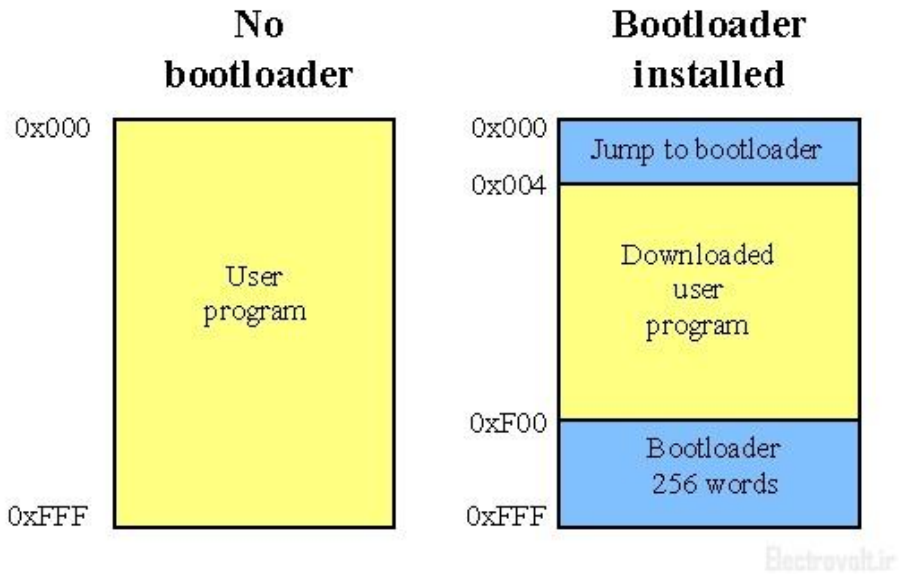
مقدمه

امروزه از میکروکنترلرها در گستره وسیعی از زندگی روزمره استفاده می شود. از میکروکنترلرهای کوچک 8 بیتی گرفته که در اسباب بازی ها ، اجاق گاز و ... استفاده می شود تا میکروکنترلرهای 32 بیتی قدرتمند که در گوشی ها و رسانه های تصویری کاربرد دارد. بدون میکروکنترلرها نه تنها زندگی ما کمتر هیجان انگیز است بلکه کنترل انسان روی اشیای محیط کمتر خواهد بود. بیلیون ها میکروکنترلر هر ساله در جهان به فروش می رسد. چه اتفاقی می افتد اگر این محصولات مبتنی بر میکروکنترلر زمانی که در میلیون ها وسیله فروخته شده کار می کنند ، نیاز به بهبود نرم افزاری داشته باشد ؟ آیا هر بار که مشکل نرم افزاری در آنها ایجاد می شود باید به کارخانه سازنده برای رفع مشکل بازگردانده شود ؟ آیا تلویزیون و سایر دستگاههای مبتنی بر میکروکنترلر میتوانند همواره آخرین نسخه نرم افزاری را داشته باشند ؟ پاسخ همگی این سوالات بستگی به وجود "بوت لودر" دارد.

تعریف و خصوصیات بوت لودر

بوت لودر (Boatloader) یک برنامه کاربردی (Application Software) است که هدف آن بروز رسانی نرم افزاری سیستم درون برنامه یا IAP(مخفف in application programming) بدون نیاز به سخت افزاری خاص (انواع پروگرامرها) می باشد.

بوت لودر می تواند در هنگام شروع ، یکپارچگی سخت افزار را بررسی نماید و قطعات سخت افزاری موجود در سیستم را چک کند. بوت لودر سایز ها و اندازه های مختلف و متفاوتی دارد و نیز پروتکل های متفاوتی از قبیل USART ، CAN، I2C، Ethernet، USBو ... را پشتیبانی می نماید. سیستم های دارای بوت لودر حداقل دو بخش نرم افزاری روی یک میکروکنترلر دارند و یک بخش نیز برای تصمیم گیری اجرای برنامه بین این دو بخش می باشد. در طی مراحل اولیه طراحی یک محصول رایج است که بوت لودر توسط تیم طراحی نادیده گرفته می شود. دلیل اصلی این امر این است که بوت لودر تا زمانی که بخواهد وارد بازار شود ، به بروز رسانی احتیاج ندارد. در حقیقت بوت لودر زمانی که یک محصول روانه بازار می شود به مهمترین بخش یک محصول تبدیل می شود. چرا که به سازنده محصول اجازه می دهد در آپدیت های بعدی از توانایی های محصول خود بهترین بهره برداری را داشته باشد. همچنین باعث می شود که شرکت بتواند باگ های نرم افزاری را که بعد از فروش کشف می شود رفع نماید. شکل زیر ساختار حافظه برنامه یک میکروکنترلر نمونه را با وجود بوت لودر و بدون آن نشان می دهد.



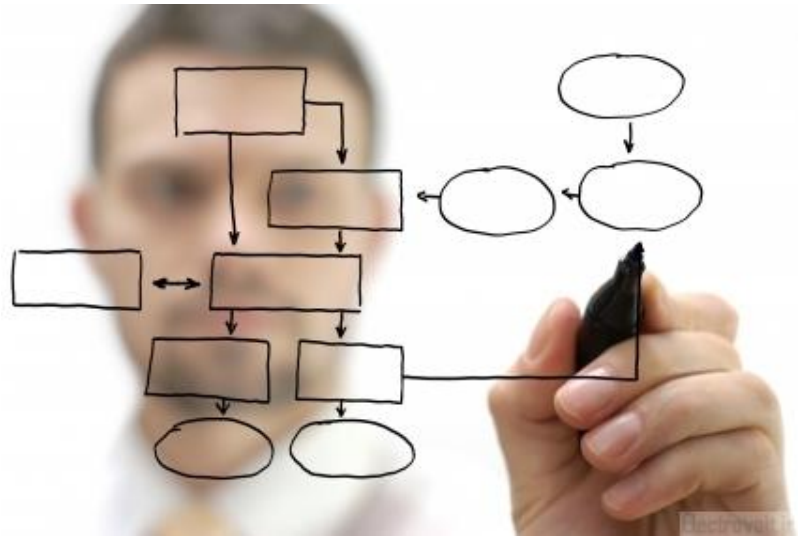
برای یک مهندس سیستم های نهفته (Embedded System Engineer) ، طراحی بوت لودر نیاز به درک عمیق از نحوه عملکرد پردازنده ، چگونگی استفاده از حافظه و کار روی پایین ترین سطح سیستم را دارد. توسعه نرم افزار بوت لودر هر چند که کاری چالش برانگیز است ، اما نکته مثبت آن استفاده از همان نرم افزار برای همه محصولات است. بنابراین با یک بار تهیه بوت لودر میتوان برای تمامی دستگاه ها و پروژه ها با تغییرات بسیار اندک از آن استفاده نمود.

هدف از ارائه این مقاله بیان اصول و راهکارهای لازم برای طراحی بوت لودر است به نحوی که خواننده بتواند بعد از مطالعه آن روش طراحی و پیاده سازی یک بوت لودر را درک نماید.

هر چند که بوت لودر برنامه کوچک و مفیدی است اما اغلب بدترین و تهدید آمیز ترین باگ ها را به سیستم تحمیل می کند. بنابراین برنامه نویس بوت لودر باید روی نقشه حافظه (Memory Map) ، بردارهای جابجایی (Vector Table) ، پارتیشن بندی حافظه Flash ، چگونگی تشخیص وجود برنامه جدید و روی پروتکل های ارتباطی سریال تسلط داشته باشد.

احتیاجات (Requirement) در برنامه نویسی

قبل از اینکه هر پروژه نرم افزاری جدیدی قبول شود ، همیشه توصیه می شود حداقل نیازهای پردازشی سیستم جمع آوری شود. این جمع آوری نیازها به توسعه دهنده (developper) نقشه راه برای طراحی محصول مورد نظر را می دهد. در مجمع کامپیوتر IEEE یک نیاز اینگونه تعریف می شود : "هر نیاز ویژگی است که باید در جهت رفع مشکل خاص باشد". بنابراین احتیاجات یک سیستم خواص کلیدی آن را بیان می کند و هیچ چیزی به اندازه لیست کامل احتیاجاتی که مشتری انتظار دارد ، برای شروع یک پروژه حیاتی نیست. اما این احتیاجات به مرور زمان شدیدا تغییر پذیر هستند به طوری که اغلب سراسر فرآیند طراحی سیستم را تغییر می دهد. چرا که مشتری نیاز به تغییر و ارتقای یک نیاز را می کند اما تمام کدنویسی باید بررسی شود و تغییرات خواسته شده اعمال شود. امروزه در فرآیندهای تغییر نرم افزاری ، دوره های ثابت زمانی تعریف می شود که در هر دوره شرکت سازنده ، سیستم را بروزرسانی کرده و تغییرات مورد نیاز مشتری را اعمال می نماید.



فاکتورهای اصلی برای طراحی و اجرای نیازهای یک سیستم عبارتند از:

- استخراج(Elicitation)
- تجزیه و تحلیل(Analysis)
- ویژگی ها(Specification)
- تایید اعتبار (Validation) و قابلیت ردیابی(Traceability)

بوت لودر در هر پروژه ، احتیاجات منحصر به فردی دارد که بر اساس نوع برنامه اپلیکشن (Application Program) مشخص می شود. اما ویژگی های یکسانی نیز وجود دارد که معمولا در همه بوت لودر ها وجود دارد. این ویژگی های عمومی به 7 گروه زیر دسته بندی می شوند.

1- قابلیت سوئیچ یا انتخاب حالت عملکرد سیستم بین برنامه اپلیکشن و برنامه بوت لودر

راهکار اول : فشردن دکمه ای که به منظور بوت روی سیستم قرار دارد برای مثلا 10 ثانیه

راهکار دوم : اختصاص یک بایت در حافظه EEPROM و تغییر مقدار آن از خارج سیستم به منظور بوت

راهکار سوم : ورود به بوت در صورت دریافت یک مقدار خاص از خارج سیستم به عنوان دستور ورود به بوت لودر

2- قابلیت ارتباط با حداقل یکی از اینترفیس های ارتباطی نظیر USB ، CAN، I2C، UARTو...

راهکار اول : انتقال فایل بوت به صورت سریال با پورت UART از یک وسیله خارجی به سیستم

راهکار دوم : بارگزاری فایل بوت درون یک SDcard و قرار دادن آن درون سیستم

راهکار سوم : اتصال پورت USB مخصوص بوت سیستم به یک سیستم دیگر و ریختن فایل بوت

3- قابلیت ذخیره و تجزیه و تحلیل فایل های اجرایی نظیر hex ، S-Recordو...

راهکار : بررسی عمقی روی فرمت فایل هگز

4- توانایی خواندن ، نوشتن و پاک کردن بخشی از حافظهFlash

راهکار : مطالعه دیتاشیت و هدر فایل های مربوطه و یافتن توابع مورد نیاز و قوانین حاکم بر آن

5- توانایی خواندن ، نوشتن و پاک کردن بخشی از حافظهEEPROM

راهکار : مطالعه دیتاشیت و هدر فایل های مربوطه و یافتن توابع مورد نیاز و قوانین حاکم بر آن

6- توانایی چک کردن صحت برنامه کاربردی و اطمینان از اشتباه نبودن برنامه(Checksum)

راهکار : توانایی محاسبه چک سام از روی برنامه و چک کردن آن قبل از ریختن نهایی برنامه اپلیکشن روی حافظه فلش

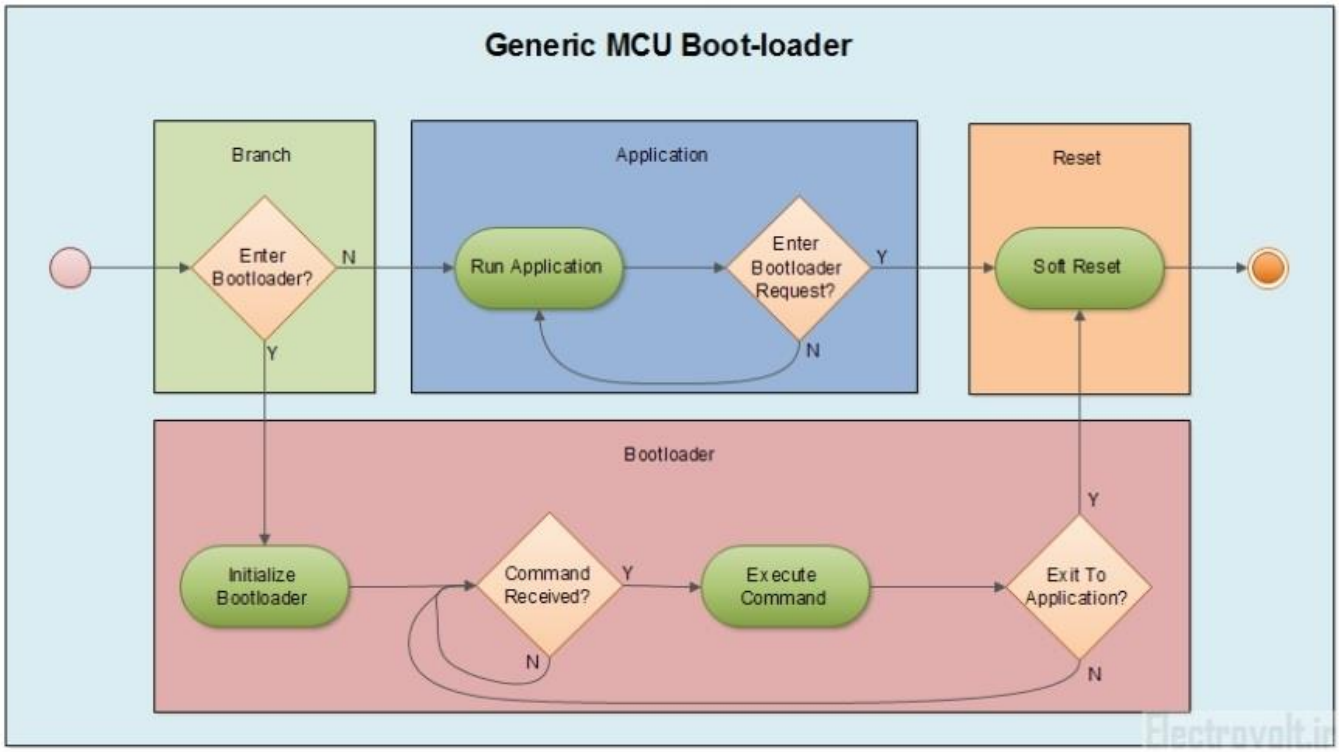
7- محافظت از برنامه بوت لودر و برنامه کاربردی

راهکار : قرار دادن بوت لودر در بخش محافظت شده حافظه فلش و فعال کردن قفل های امنیتی

هر یک ازین دسته بندی های فوق خود میتواند به دسته بندی های کوچکتری برای تولید احتیاجات بوت لودر برای هر محصولی استفاده شود.

سیستم بوت لودر

برنامه بوت لودر در ابعاد و اندازه های مختلف و متفاوتی ارائه می شود اما در حالت کلی عملکرد یک سیستم با بوت لودر نسبتا استاندارد است. در شکل زیر سه بخش اصلی که در هر برنامه بوت لودر وجود دارد را مشاهده می کنید. بخش تصمیم گیری (branching code) ، بخش برنامه کاربردی (Application code) و بخش برنامه بوت لودر(boatloader code) . بخش ریست نرم افزاری (Software Reset) نیز برای هردو کد بوت لودر و اپلیکشن جهت راه اندازی مجدد سیستم استفاده می شود.



برای اکثر سیستم ها ما همواره در حال اجرای بخش اپلیکشن هستیم. در ابتدای شروع برنامه برای اجرا شدن برنامه بوت لودر و یا برنامه اپلیکشن در بخش branch تصمیم گیری می شود. برای این بخش از یک دستور شرطی ساده نظیر 0 یا 1 بودن یکی از پایه های GPIO برای رفتن به بوت لودر میتوان استفاده کرد. هر چند که در سیستم های تولید شده توسط کارخانه های تولید کننده جدید اغلب از یک راه حل بوت گذاری قوی تر استفاده می گردد. در یک سیستم قوی تر ، بوت لودر ابتدا اجرا می شود و برخی از توابع سیستمی پایه را برای بررسی یکپارچکی سیستم قبل از تصمیم گیری شرطی اجرا می کند. به همین دلیل است که در بسیاری از موارد کد تصمیم گیری در بوت لودر گنجانده می شود. کد برنامه کاربردی تنها بعد از اینکه تصمیم گیری انجام شد و هیچ درخواستی برای اجرای بوت لودر وجود نداشت ، اجرا می شود. حتی در حالی که برنامه کاربردی در حال بارگزاری است و وظایف خود را انجام می دهد ، درخواست مبنی بر مراجعه به بوت لودر نیز باید گنجانده شود. زمانی که برنامه چنین درخواستی را دریافت نماید ، برنامه اپلیکشن ، پاکسازی و راه اندازی مجدد می شود. توسط تایمر سگ نگهبان (whatchdog timer) نیز میتوان سیستم را به صورت نرم افزاری ریست کرد.

کد بوت لودر وظیفه اجرای تمامی توابع بوت لودر را دارد. زمانی که برنامه بوت لودر اجرا می شود ، برنامه شروع به راه اندازی واحدهای زیر مجموعه می کند که به منظور اجرای تمامی توابع بوت لودر مورد نیاز است. این زیر مجموعه ها اغلب واحد هایی نظیر کلاک سیستم (system clock) ، وقفه (interrupt) ، بردارهای حافظه (memory vector) و پروتکل های ارتباطی مورد نیاز می باشند که به بوت لودر اجازه می دهند با محیط اطراف خود در ارتباط باشد و دستورات مورد نیاز برای بوت را از روی حافظه فلش اجرا نماید.

رفتار بوت لودر

برنامه بوت لودر از نظر برنامه نویسی تفاوت چندانی با یک برنامه اپلیکشن ندارد و در حقیقت یک برنامه اپلیکشن استاندارد به نام بوت لودر است. آنچه که برنامه بوت لودر را ویژه می سازد ، توانایی به اشتراک گذاشتن حافظه فلش ، حذف و نوشتن بخش هایی از آن است. مشابه هر برنامه اپلیکشن

دیگری ، یکی از اولویت های اول در برنامه بوت لودر ، راه اندازی صحیح سیستم و واحدهای جانبی مورد نیاز است. به منظور کاهش حجم برنامه بوت لودر ، از نوشتن کدهای اضافی و راه اندازی واحدهای جانبی بدون استفاده صرف نظر می گردد.

دو مدل رفتاری از برنامه بوت لودر وجود دارد. در مدل اول بارگزاری و قرار گیری کد روی حافظه از ابتدا تا انتها به صورت کاملا اتوماتیک و درون سیستم انجام می گردد. مانند بوت لودرهای SdCard که در آن برنامه جدید روی حافظه SD ریخته شده و جایگزین برنامه قبلی می گردد.

در مدل دوم سیستم دارای مدیریت اتوماتیک نیست. در عوض بوت لودر در حالت Idle قرار می گیرد و منتظر دریافت دستورات از منبع خارجی می شود. منبع خارجی معمولا یک نرم افزار کامپیوتری است که بوت لودر را به حالت بوت میبرد و عمل بارگزاری و جایگزینی را انجام میدهد. دلیل اصلی استفاده از مدل دوم در سیستمهای بدون حافظه خارجی یا Sd Card است.

هر بوت لودر از نظر نوع و تعداد دستوراتی که پشتیبانی می کند ، یکتا است. بوت لودرهای پیچیده دستورات بیشتری از بوت لودرهای ساده دارند. به ازای اضافه شدن هر دستور جدید ، استفاده از حافظه فلش میکرو به عنوان بوت لودر افزایش و حجم آن بخش از فلش که برای ذخیره برنامه کاربر می باشد ، کاهش می یابد. هر بوت لودر حداقل باید از سه دستور زیر پشتیبانی کند:

1. پاک کردن : Flash پاک کردن برنامه اپلیکشن قدیمی از روی حافظه فلش
2. نوشتن : Flash نوشتن برنامه جدید اپلیکشن روی حافظه فلش
3. خروج یا ریست : خروج از برنامه بوت لودر و اجرای برنامه اپلیکشن

با این سه دستور همه توابع نوشتن فایل در Flash انجام می شود. اما دستورات دیگری نیز هستند که برای بهبود قابلیت های بوت لودر استفاده می شوند. این دستورات عبارتند از:

1. افزایش امنیت دسترسی به حافظه فلش با قرار دادن قفل روی آن
2. پاک کردن ، نوشتن و خواندن از حافظهEEPROM
3. بررسی صحت برنامه اپلیکشن جدید قبل از ریخته شدن روی حافظه فلش(Check Sum)
4. بررسی صحت برنامه اپلیکشن ریخته شده روی حافظه فلش(Verify)

قفل کردن حافظه ، باعث افزایش امنیت و کاهش خواندن/نوشتن تصادفی روی حافظه می گردد. با افزودن Check Sum به برنامه اپلیکشن و ارسال نمودن آن به میکرو در هنگام انجام پروسه بوت ، میتوان از صحت برنامه اپلیکشن اطمینان حاصل کرد و از ریخته شدن برنامه غلط روی حافظه فلش جلوگیری نمود. استفاده از حافظه EEPROM میتواند در هنگام پیکربندی سیستم در حالت های خاص به کار گرفته شود.

مراحل عمومی پروسه بوت لودینگ یک وسیله به صورت زیر است:

1. بازکردن فایل شامل محتویات حافظه فلش
2. شروع بوت لودر
3. پاک کردن برنامه اپلیکشن قبلی از حافظه فلش
4. ارسال باینری برنامه اپلیکشن جدید به بوت لودر
5. تولید چک سام و تطبیق آن
6. ریختن برنامه اپلیکشن جدید روی حافظه فلش
7. خروج از بوت لودر و ریست شدن سیستم

رفتار اپلیکشن

رفتار برنامه اپلیکشن در بیشتر قسمت ها به جز یک قسمت به رفتار برنامه بوت لودر شبیه نمی باشد. در حین اجرای برنامه اپلیکشن نیاز به دریافت دستور برای رفتن به برنامه بوت لودر را داریم و این یعنی برنامه اپلیکشن دو قابلیت زیر از قابلیت های برنامه بوت لودر را دارد:

1. تخصیص بخشی از حافظه و سخت افزار به منظور تشخیص ورود به حالت بوت
2. ریست کردن سیستم برای تصمیم گیری(Branching)

بهترین مکان برای ذخیره مقداری که توسط برنامه اپلیکشن به منظور تصمیم گیری میان اجرای برنامه بوت لودر یا برنامه اپلیکشن می باشد ، حافظه EEPROM است. این مقدار ذخیره شده درحافظه EEPROM توسط برنامه تصمیم گیری (Branching Code) مورد استفاده قرار می گیرد. حافظه EEPROM بین هر دو برنامه بوت لودر و اپلیکشن مشترک می باشد و از این رو هم در هنگام دریافت درخواست بوت از جانب کاربر توسط برنامه اپلیکشن مقداری در آن ریخته می شود تا در هنگام تصمیم گیری مورد استفاده قرار گیرد و هم در پایان پروسه بوت توسط برنامه بوت لودر به حالت اولیه خود بر می گردد.

ریست نرم افزاری میکروکنترلر در دو زمان مورد نیاز است. یکی در هنگام آمدن درخواست ورود به بوت و دیگری در هنگام پایان پروسه بوت. این ریست شدن معمولا به دو روش انجام می گیرد. در روش اول با استفاده از تایمر سگ نگهبان (whatchdog timer) برنامه داخل یک حلقه نامتناهی نظیر while(1) شده و بعد از مدتی ریست می شود. در روش دوم مقداری خاص درون رجیستر تایمر سگ نگهبان تنظیم شده و بعد از مدتی ریست می شود. در روش دوم تاخیر به حداقل می رسد.

نکته : یک راه راحت تر در میکروکنترلرهای Cortex M3 با استفاده از تابع NVIC_SystemReset است که در هدر فایل core_cm3.h قرار دارد و می توان با آن سیستم را به صورت نرم افزاری ریست نمود.

تصمیم گیری در شروع

هنگامی که سیستم شروع به کار می کند ، حداقل دو نرم افزار مختلف قابلیت اجرا توسط CPU را دارد ؛ بوت لودر ، اپلیکشن و احتمالا فایل پشتیبان (backup)از برنامه اپلیکشن. بنابراین ضروری است یک کد تصمیم گیری در هنگام شروع گنجانده شود که فرآیند تصمیم گیری را مدیریت کند.

روش های مختلفی برای این تصمیم گیری وجود دارد. ساده ترین روشی که اغلب مورد استفاده قرار میگیرد ، بررسی یکی از پایه های GPIO برای گرفتن تصمیم است. یعنی اگر سیگنال GPIO منطق 1 شود ، اپلیکشن اجرا شود و اگر 0 شود بوت لودر اجرا شود. این روش بسیار ساده بوده و از یک کلید Posh Button جهت رفتن به بوت استفاده می کند. کد تصمیم گیری اغلب به منظور سرعت بالای آن و پرش به برنامه اپلیکشن (jamp to application)به زبان اسمبلی نوشته می شود.

اولین برنامه نمونه ای که برای این قسمت آورده شده است برگرفته از AN10866 از شرکت NXP می باشد که در کامپایلر Keil و برای میکروکنترلر LPC1788 نوشته شده است:

```
1  #include <lpc177x_8x.h>
2  #define USER_FLASH_START 10000
3
4  __asm void boot_jump( uint32_t address ){
5      LDR SP, [R0]    ;Load new stack pointer address
6      LDR PC, [R0, #4] ;Load new program counter address
7  }
8
9  void execute_user_code(void)
10 {
11   /* Change the Vector Table to the USER_FLASH_START
12   in case the user application uses interrupts */
13   SCB->VTOR = USER_FLASH_START & 0x1FFFFF80;
14   boot_jump(USER_FLASH_START);
15 }
16
17 int main (void){
18
19   //Branching Code
20
21   LPC_GPIO0->DIR &= ~(1<<9);
22
23   if( (LPC_GPIO0->PIN & (1<<9)) == 0 )
24       execute_user_code();
25
26 while(1);
27 }
28
29
```


همانطور که در مثال فوق مشاهده می کنید ، برنامه اپلیکشن از آدرس 0x10000 حافظه فلش آغاز شده است. در تابع Main اولین کار تصمیم گیری است. در این مثال اگر پایه 9 از GPIO0 به زمین متصل شده باشد ، تابع execute_user_code اجرا شده و برنامه به بخش اپلیکشن پرش (jump) می کند؛ در غیر این صورت برنامه داخل حلقه نامتناهی گیر می افتد. در این برنامه به علت اینکه هنوز بوت لودر را ننوشته ایم حلقه while(1) قرار دادیم اما برنامه بوت لودر دقیقا جایگزین while(1) شده یا درون این حلقه نوشته می شود.

توابع execute_user_code و boot_jump عینا از AN10866 شرکت NXP آورده شده است. این Application Note را میتوانید از لینک زیر دریافت کنید:

دانلود مستقیم AN10866 شرکت NXP

یکی از مشکلات برنامه قبلی ، استفاده از یک پایه GPIO برای تصمیم گیری است و این کار راه حل مطمئنی نمی باشد. چرا که میتواند به صورت تصادفی) با فشار اشتباه توسط کاربر یا بوجود آمدن Noise یا (Bounce برنامه بوت شود. ضمنا برنامه قبلی نمیتواند تشخیص دهد آیا فایل شامل برنامه اپلیکشن جدید وجود دارد یا خیر تا در هنگامی که فایلی آماده نیست وارد بوت نشویم. بهترین راه حل استفاده از یک متغیر است که در حافظه EEPROMتعریف شده باشد. در این حالت بررسی 0 یا 1 بودن پایه GPIO حذف شده و با فرض اینکه یک کاراکتر Ascii با نام Mode درون EEPROMذخیره شده باشد ، برنامه فوق به صورت زیر تغییر می کند:

```
if(Mode != 'B') { ... }
```

در این مثال کاراکتر B نشان دهنده بوت شدن است که هر مقدار ثابت دیگری نیز میتواند باشد.

امروزه درون اکثر میکروکنترلرها حافظه EEPROM داخلی نیز وجود دارد. البته میتوان از حافظه EEPROM خارجی و اتصال آن توسط رابط های سریال نظیر SPI و I2C برای میکروهایی که این نوع حافظه را ندارند ، استفاده کرد که در صورت استفاده از حافظه خارجی با کمی تاخیر (delay) نسبت به حافظه داخلی در خواندن/نوشتن اطلاعات مواجه هستیم. در مواقعی که هیچ کدام در دسترس نباشند میتوان متغیر را درون حافظه فلش در آدرس خاصی ذخیره کرد. استفاده از حافظه فلش نیز منطقی است ، چرا که حافظه فلش در همه میکروکنترلرها وجود دارد و سرعت بسیار بالایی هم دارد.

نکته : برای تعریف متغیر در محل خاصی از حافظه در کامپایلر Keil از کلمه کلیدی attribute به صورت زیر استفاده می شود:

```
char Mode __attribute__((at(0x10000))) = 'B';
```

در این مثال کاراکتری با نام Mode در آدرس 0 10000xبا مقدار اولیه 'B' ذخیره شده است.

گفتیم که کد تصمیم گیری باید وجود برنامه بوت لودر را بررسی کند و در صورتی که فایل برنامه جدید وجود نداشت) در حالت بوت با (SdCard و یا هیچ دیتایی به صورت سریال به میکروکنترلر وارد نشد (در حالت بوت با ارتباط سریال) عملیات بوت را متوقف نماید. علاوه بر این زمانی که دیتای برنامه جدید وارد می شود به علت حساسیت بالای این دیتا ، باید عدد checksum که ناشی از دریافت توسط میکروکنترلر می باشد و عدد checksumکه بعد از دریافت مجددا تولید می شود با هم تطابق کامل داشته باشد. در صورت تطابق ، دیتای وارد شده عملیات بوت لودینگ تکمیل می شود و در غیر این صورت عملیات مردود شده و سیستم ریست می گردد. این دو موضوع را نیز میتوان با اضافه کردن دو متغیر به برنامه قبلی تکمیل کرد.

به مثال زیر توجه کنید:

```
if((Checksum_Complete == TRUE) && (StartUpTmr == EXPIRED))
{
  if((*ResetVector != 0xFFFF)&&(Mode != 'B')&&(Boot_ToolPresent != TRUE)&&(Checksum_Valid
!= FALSE))
  {
    App_LoadImage();
  }
  else
  {
    Boot_LoadImage();
  }
}
```

این مثال راه حل قوی برای کد تصمیم گیری ارائه می کند StartUpTmr .یک متغیر اختیاری است که حلقه if را کنترل می کند. همچنین زمانی Checksum_Completeبرابر TRUE می گردد که برنامه اپلیکشن جدید وجود داشته و checksum برای آن تولید شده باشد. بعد از این کد هر یک از شرط های زیر را نیز چک می کند:

1- آیا بردار ریست (محل شروع برنامه اپلیکشن) وجود دارد ؟

2- آیا در حالت بوت هستیم ؟

3- آیا ابزارهای ارتباطی آماده برای کار هستند ؟

4- آیا چک سام تولید شده معتبر است ؟

در صورتی که همه این شرط ها برقرار بود عملیات Boot اجرا می گردد و در غیر این صورت به برنامه اپلیکشن قدیمی خواهیم رفت و هیچ تغییری در برنامه قبلی حاصل نمی گردد.

توجه : آدرس بردار ریست (ResetVector) میتواند بین عدد 0x0000 و حداکثر مقدار ذخیره سازی حافظه Flash باشد بنابراین در اینجا فرض کردیم همواره مخالف آدرس 0xFFFF است.

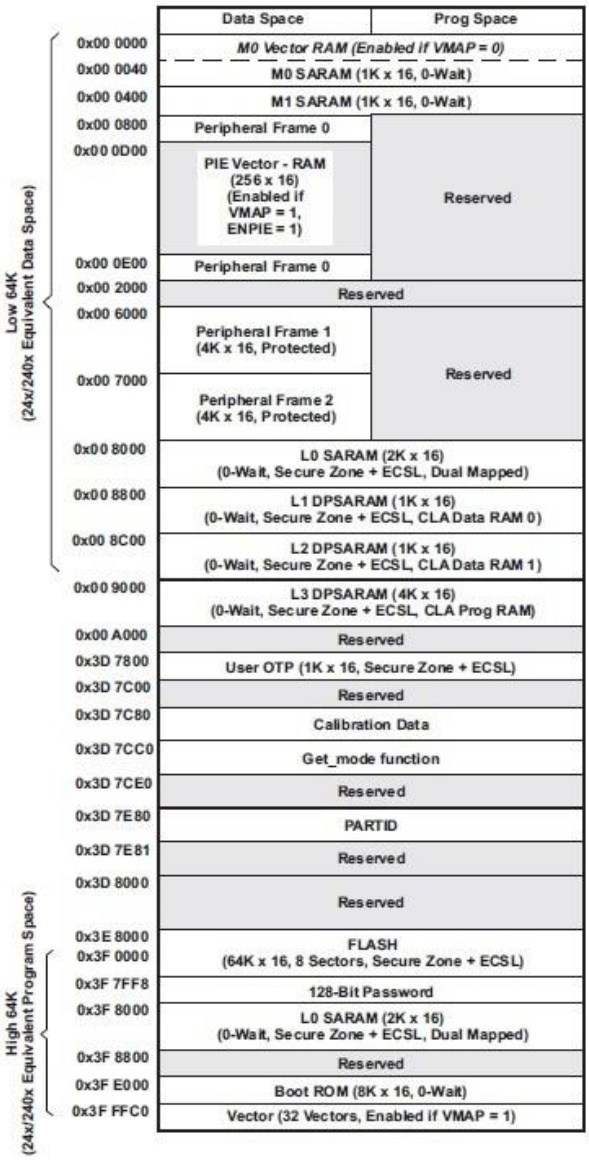
پارتیشن بندی حافظه

هر میکروکنترلر دارای یکی از انواع حافظه های دائمی (Non-volatile Memory) است که برای ذخیره برنامه اجرایی منیکروکنترلر به کار می رود. در اکثر مواقع از حافظه های سریع Flash به عنوان حافظه دائمی در میکروکنترلرها استفاده می گردد. حافظه Flash خود به بخش های کوچکتری تقسیم بندی می گردد. به کوچکترین بخش حافظه فلش Page گفته می شود. از به هم پیوستن Page ها ، سکتور (Sector) بوجود می آید. از به هم پیوستن سکتور ها نیز بلاک (Block) تشکیل می شود. در هر میکروکنترلر دسترسی به هر یک از این بخش های مختلف حافظه فلش میتوانند متفاوت باشد. اکثر میکروکنترلرها فقط اجازه نوشتن در یک بایت از حافظه فلش در هر سیکل کلاک را می دهند ؛ برخی دیگر نیز ممکن است اجازه نوشتن 8 تا 256 بایت را نیز در یک سیکل کلاک بدهند. اما مسئله پاک کردن (Erase) حافظه فلش متفاوت است. در اکثر میکروکنترلرها پاک کردن حافظه فلش اغلب بر اساس سکتور می باشد و پاک کردن بخش های کوچکتر از آن امکان پذیر نمی باشد. هر سکتور اغلب شامل 4 کیلوبایت می باشد.

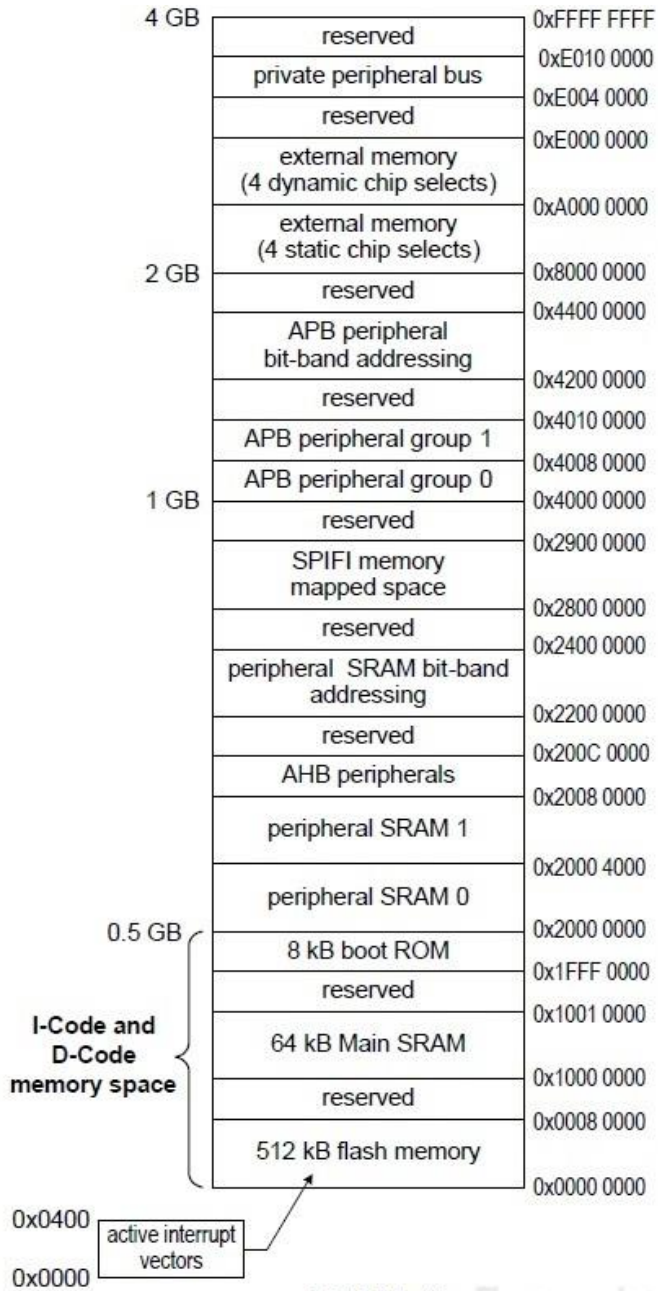
طراح بهتر است قبل از عمیق شدن روی طراحی بوت لودر ، با مراجعه به بخش حافظه میکروکنترلر در دیتاشیت یا راهنمای کاربر (User Manual) نسبت به پارتیشن بندی حافظه فلش میکروکنترلر اطلاعات لازم را بدست آورده و روی نحوه عملکرد و ساماندهی حافظه فلش در میکروکنترلر مسلط شود. چرا که یکی از مهمترین بخش های بوت لودر ریختن برنامه و پاک کردن آن از روی حافظه فلش می باشد.

اما هدف اصلی از توجه به حافظه فلش و تسلط روی ساماندهی آن ، تعیین کردن مناسب ترین بخش برای پیاده سازی بوت لودر می باشد. برنامه اپلیکشن (Application Program) و برنامه بوت لودر (Boatloader Program) دو بخشی هستند که باید در حافظه فلش جاسازی شده و به طور کامل از هم مجزا باشند. به عبارت دیگر برنامه اپلیکشن نه تنها نباید بتواند دستکاری روی برنامه بوت لودر انجام دهد بلکه هرگز نباید از وجود برنامه بوت لودر اطلاع داشته باشد.

شرکت های سازنده میکروکنترلر ها در بخش نقشه حافظه (Memory Map) از دیتاشیت ، به شیوه ای متفاوت به بررسی حافظه در میکروکنترلر می پردازند. در نقشه های حافظه به هر یک از بایت های حافظه (چه حافظه موقت و چه حافظه دائمی) یک آدرس منحصر به فرد نسبت می دهند. در شکل های زیر دو نمونه از نقشه حافظه مربوط به میکروکنترلرهای LPC1788 از شرکت NXP و TMS320F28035 از شرکت Texas Instrument را مشاهده می کنید.



TMS320F28035



LPC177x/8x

همانطور که مشاهده می کنید ، میکروکنترلر LPC1788 دارای 512 کیلوبایت حافظه فلش برای برنامه اپلیکشن می باشد که از آدرس 0 تا 80000 را به خود اختصاص داده است که 1 کیلوبایت اول آن مربوط به جدول بردارهای وقفه می باشد. همچنین در این میکروکنترلر 8 کیلوبایت حافظه فلش از آدرس 1FFF0000 تا آدرس 200000000 برای برنامه بوت لودر اختصاص داده شده است. در میکروکنترلر TMS320F28035 نیز 64 کیلوبایت حافظه فلش برای برنامه اپلیکشن و 8 کیلوبایت حافظه فلش برای برنامه بوت لودر اختصاص داده شده است. با دقت در نقشه های حافظه این دو میکرو و مقایسه آن ها دو موضوع اصلی مشخص می شود. اول آن که در میکروکنترلر LPC1788 هر سکتور معادل 4 کیلوبایت است اما در TMS320F28035 هر سکتور معادل 8 کیلوبایت است که در پاک کردن حافظه فلش اهمیت دارد. دوم آن که بخش اختصاص داده شده به برنامه بوت لودر در میکروکنترلر LPC1788 بعد از حافظه فلش مربوط به برنامه اپلیکشن است اما در میکروکنترلر TMS320F28035 این بخش قبل از برنامه اپلیکشن است.

تذکر: در صورت بیشتر شدن برنامه بوت لودر از میزان اختصاص یافته آن در قسمت تعبیه شده ، میتوان برنامه بوت لودر را به بخش بزرگتر حافظه فلش ، منتقل کرد. به طور کلی با داشتن دید عمیق و در نظر گرفتن همه جانبه میتوان هر بخشی از حافظه فلش را به بوت لودر اختصاص داد.

به طور کلی در هنگام انتخاب محل قرارگیری بوت لودر فاکتورهای زیر را در نظر داشته باشید:

1- حجم برنامه بوت لودر

2- محل قرارگیری جدول بردارها (Vector Table)

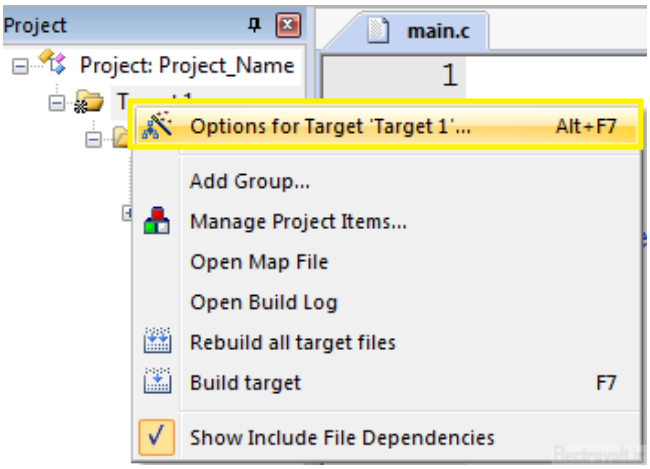
3- شناسایی بخش های حفاظتی و امن حافظه فلش (همه حافظه فلش ممکن است محافظت شده نباشد)

توجه کنید که سایز برنامه اپلیکشن همیشه در اولویت قرار دارد و برنامه بوت لودر تا جای ممکن باید کوچکتر شود تا فضای بیشتری برای برنامه اپلیکشن محیا باشد. معمولا با ارائه آپدیت ، برنامه اپلیکشن حجم بیشتری را به خود اختصاص می دهد لذا حافظه فلش باید طوری پارتیشن بندی شود که همه فضاهای خالی حافظه در صورت وجود در انتهای بخش اپلیکشن قرار گیرد.

نکته دیگری که باید مورد توجه قرار گیرد این است که برنامه اپلیکشن باید مضربی از سایز سکتور باشد. چرا که کوچکترین سایزی که میتوان حافظه فلش را پاک کرد یک سکتور است و برای اینکه برنامه اپلیکشن قبلی را به طور کلی پاک کنیم نیاز است که دقیقا مضربی از سایز سکتور (معمولا 4 KB) باشد. همچنین باید توجه داشت که شروع برنامه اپلیکشن از یکی از مضارب سکتور باشد. برای مثال وقتی که سایز هر سکتور 4KB است ، شروع برنامه اپلیکشن میتواند از 0 ، 4KB ، 8KB و ... باشد. برای برخی از میکروکنترلرها نظیر سری S12X شرکت NXP میتواند مضربی از 256 بایت باشد. چرا که کمترین مقداری از حافظه فلش که میتواند پاک شود ، 256 بایت است.

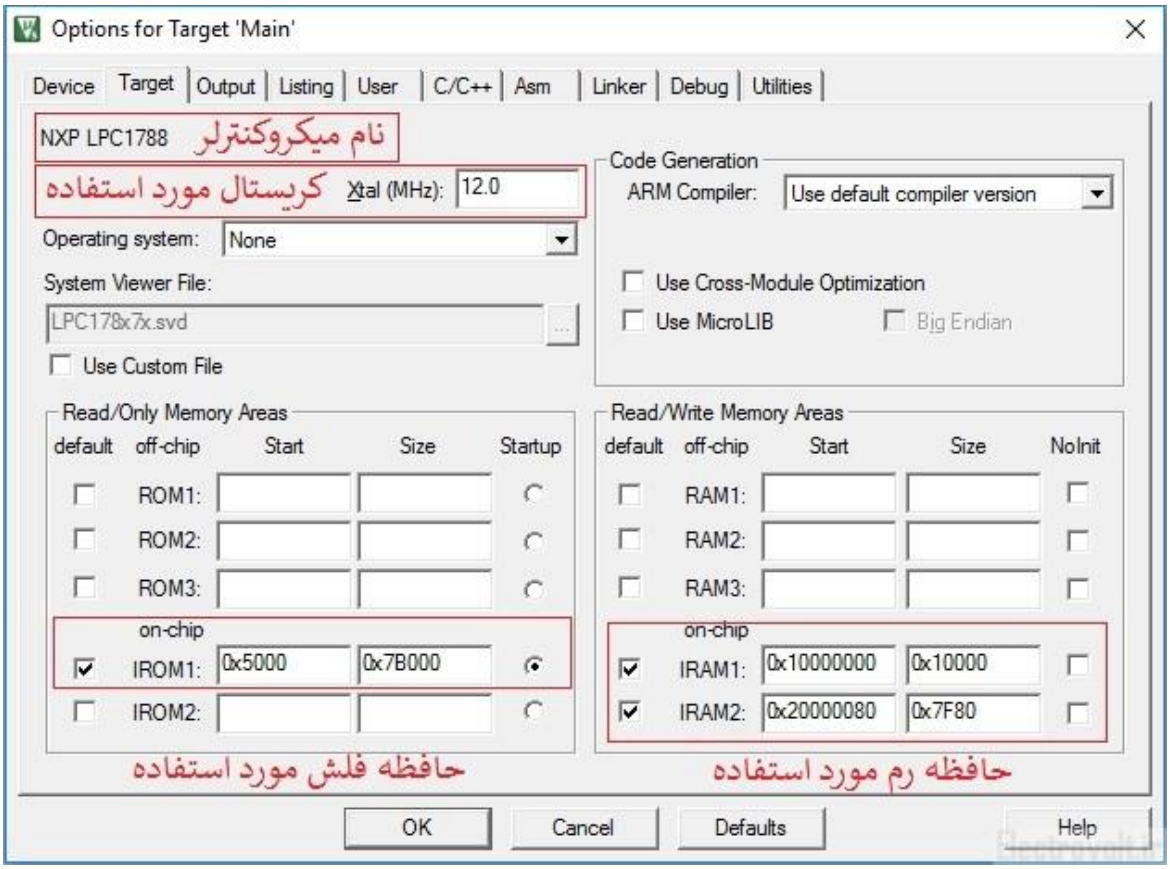
در برخی از میکروکنترلرها نظیر LPC178x و LPC177x ممکن است فضایی مخصوص برای بوت لودر اختصاص داده شود. قرار گرفتن بوت لودر در این فضای ویژه به علت اقدامات امنیتی و قفل های قوی تر توصیه می شود.

بعد از مطالعه دقیق دیتاشیت و انتخاب مکان های مورد نظر خود ، برای برنامه اپلیکشن و بوت لودر ، به منظور پیاده سازی آن باید به تنظیمات لینک کننده (Linker) توجه داشت. در نرم افزار Keil این تنظیمات از منوی Project و گزینه Options قابل دسترسی است. شکل زیر نحوه دسترسی به تنظیمات در نرم افزار Keil را نشان می دهد.



تنظیمات مورد نیاز ما در سربرگ Target از پنجره Options می باشد. در این بخش محدوده عملیاتی حافظه RAM و ROM قابل تغییر است. با مراجعه به دیتاشیت آدرس شروع و اندازه RAM را در بخش IRAM1 وارد می کنیم. در برخی از میکروکنترلرها ممکن است دو RAM با سایزهای

متفاوت وجود داشته باشد. برای این میکروکنترلرها آدرس شروع و اندازه RAM دوم را در بخش IRAM2 وارد می کنیم. در بخش IROM1 نیز بسته به نوع برنامه (اپلیکشن یا بوت لودر بودن) و همچنین آدرس و حجم انتخابی برای بوت لودر یا اپلیکشن ، این بخش را نیز تکمیل می کنیم. شکل زیر این پنجره را برای میکروکنترلر LPC1788 نمونه نشان می دهد.



همانطور که در شکل مشاهده می کنید ، این میکروکنترلر دو بخش RAM مجزا از هم دارد برای اینکه از هر دو RAM استفاده کنیم آدرس شروع و اندازه آنها را در IRAM1 و IRAM2 وارد می کنیم. همچنین آدرس شروع برنامه از 0x5000 می باشد در نتیجه میتوان تشخیص داد که این مثال برای برنامه اپلیکشن بوده و برنامه بوت لودر از آدرس 0 تا 0x5000 (یعنی 20KB) بوده است. سایز حافظه Flash در این میکروکنترلر 0x80000 می باشد که 20 کیلوبایت اول آن مربوط به بوت لودر است در نتیجه مقدار 0x7b000 از حافظه فلش در دسترس اپلیکشن است.

تذکر : برنامه بوت لودر در میکروکنترلرهای LPC مضربی از 4 کیلو بایت (0x1000 هگز) باید باشد.

بردارهای وقفه و ریست

مفهوم بردار ریست مقداری گیج کننده به نظر می رسد که فقط در زمان توسعه برنامه بوت لودر مطرح می شود. در حالت عادی برنامه نویس نیازی به دانستن محل ذخیره این بردار ندارد و کامپایلر همه چیز را به صورت اتوماتیک کنترل می کند. بردار ریست (Reset Vector) در جایی از حافظه قرار دارد و به اولین دستورالعمل برنامه که باید اجرا شود ، اشاره می کند. زمانی که میکروکنترلر روشن یا ریست می شود ابتدا به این بردار مراجعه

کرده و به آدرسی که این بردار به آن اشاره می کند رفته و اولین دستور را از آن آدرس اجرا می کند. برای زمانی که بوت لودر وجود دارد این بردار باید روی آدرس شروع برنامه بوت لودر یا آدرس شروع برنامه تصمیم گیری اشاره کند.

بردار ریست می تواند هر نقطه ای از حافظه فلش که برنامه نویس بخواهد قرار گیرد اما بهتر است محل ذخیره این بردار را از حالت پیش فرض تغییر ندهیم. یکی از راههای بدست آوردن آدرس این بردار جستجو در فایل هگز تولید شده توسط کامپایلر می باشد. این بردار رکورد خاصی از فایل هگز می باشد که میتوان از سایز رکورد ها آن را تشخیص داد و بعد از تشخیص میتوان مقدار آن را روی آدرس شروع برنامه بوت لودر مورد نظر قرار داد.
استفاده از این روش توصیه نمی گردد چرا که امکان خطا ممکن است وجود داشته باشد و همچنین امکان کنترل روی مقدار بردار وجود ندارد. راه بهتر این است که مکان بردار ریست را از دیتاشیت شناسایی کرده و آن را در برنامه بوت لودر قرار دهیم. در این روش طراح کنترل کاملی روی بردار ریست داشته و نیازی به خواندن و ترجمه کردن زبان hex نیست.

بردار ریست همانند دیگر ثابت ها در ابتدای برنامه تعریف می شود با این تفاوت که دقیقا باید در آدرسی که برای آن اختصاص یافته است ، ذخیره گردد. در میکروکنترلر LPC1788 این:

```
U32 APP_RESET __attribute__((at(0x00000004))) = 0x5000;
```

در این مثال یک متغیر با نام APP_RESET در خانه 4 حافظه فلش تعریف شده است که مقدار آن محل شروع برنامه اپلیکشن می باشد.

نکته : در میکروکنترلرهای CortexM3 بردار ریست در محل0x00000004 از حافظه قرار دارد.

تعریف بردار ریست تنها موضوع چالش برانگیز در طراحی بوت لودر نیست. چرا که بردار ریست جزئی از مجموعه بردارهایی است که در یک جدول قرار می گیرند و به آن ها جدول بردار (Vector Table) گفته می شود. این جدول شامل مقدار شروع اشاره گر پشته (initial Stack Pointer Value)، بردار ریست (Reset Vector) ، بردارهای استثنا (exception vectors) ، بردارهای وقفه (interrupt vector) و تعدادی بردارهای مورد نیاز دیگر هستند. شکل زیر جدول بردارها را برای میکروکنترلر LPC1788 نشان می دهد.

Exception number	IRQ number	Offset	Vector
127	111	0x1FC	IRQ111
.		.	.
.		.	.
.		.	.
18		0x004C	IRQ2
17	2	0x0048	IRQ1
16	1	0x0044	IRQ0
15	0	0x0040	Systick
14	-1	0x003C	PendSV
13	-2	0x0038	Reserved
12			Reserved for debug
11			SVCall
10	-5	0x002C	Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value.

Fig 185. Vector table

برنامه بوت لودر شبیه به هر برنامه اپلیکشن دیگری نیاز به Vector Table دارد. به عبارت دیگر برنامه اپلیکشن باید Vector Table مربوط به خود و برنامه بوت لودر نیز باید Vector Table مربوط به خود را داشته باشد. و این به معنی آن است که یک جدول بردار اضافی نیاز داریم که باید در جایی ذخیره گردد. در برخی از میکروکنترلرهایی که برای بوت لودر حافظه اختصاصی قرار داده اند این موضوع حل شده و نیاز به هیچ کار اضافی نمی باشد. اما برای زمانی که در بخش اصلی حافظه فلش برنامه بوت لودر را قرار می دهیم یا میکروکنترلر ما هیچ حافظه اختصاصی برای بوت لودر ندارد ، نیاز است که جدول بردارهای جدیدی را تنظیم و به پردازنده معرفی نماییم.

معرفی فرمت اینتل هگز

هر کامپایلر از فرمت خاصی برای انتقال برنامه به میکروکنترلر استفاده می کند که شرکت های سازنده آن ها را ایجاد و توسعه داده اند. از میان فرمت های رایج مورد استفاده میتوان به فرمت S-Record شرکت Motorola و فرمت Hex شرکت Intel اشاره کرد. برنامه نوشته شده در کامپایلر قبل از ریخته شدن روی میکروکنترلر به این فرمت ها تبدیل شده و سپس توسط پروگرامر مورد استفاده قرار می گیرد. ما در اینجا فرمت Hex را به علت معروف بودن و ساده تر بودن انتخاب می کنیم.

در طراحی بوت لودر نیاز به درک فرمت ذخیره سازی فایل هگز داریم چرا که اولا برنامه میکروکنترلر ترکیبی از برنامه بوت لودر و اپلیکشن مجزا از هم است که قبل از پروگرام باید با هم ترکیب گردد و ثانیا فایل اپلیکشن جدیدی که باید جایگزین برنامه اپلیکشن قبلی گردد دارای فرمت هگز است.

در فایل های با فرمت هگز (Intel Hex format) ، اطلاعات باینری مربوط به میکروکنترلر در خطوط متنی مبنای 16 و به فرم کدهای ASCII قرار گرفته اند. هر خط شامل یک رکورد مجزا می باشد که به آن Hex Record نیز می گویند. در زیر نمونه یک برنامه میکروکنترلر تبدیل شده به فرمت اینتل هگز را مشاهده می کنید:

```
:10010000214601360121470136007EFE09D2190140
:100110002146017E17C20001FF5F16002148011928
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

همانطور که مشاهده می کنید هر رکورد در یک خط قرار دارد که با کولن (colon) شروع شده و در پایان با Enter (یعنی line feed + carriage return) به خط بعدی می رود. همچنین در پایان همه فایل های هگز عبارت 00000001FF وجود دارد.

به طور کلی هر رکورد از 6 بخش زیر تشکیل می شود:

1- کد شروع (Start Code) : یک کاراکتر اسکی کولن (:)

2- تعداد بایت های دیتا (Byte count) : دو کاراکتر اسکی بعدی ، تعداد بایت های دیتای اصلی موجود در رکورد را نشان می دهد. حداقل این دو کاراکتر 00 و حداکثر آن FF می باشد. این بخش در اغلب برنامه ها 10 یا 20 در مبنای هگز (یعنی 16 یا 32 در مبنای 10) می باشد.

3- آدرس (Address) : چهار کاراکتر بعدی آدرس 16 بیتی مربوط به آدرس ذخیره دیتای اصلی موجود در رکورد در حافظه را مشخص می کند. با این 16 بیت حداکثر 64 کیلوبایت را میتوان آدرس دهی کرد. در صورتی که مقدار برنامه بیشتر از 64 کیلوبایت باشد یک رکورد با آدرس 16 بیتی به نام پایه آدرس (Address Base) اضافه می گردد که 16 بیت با ارزش آدرس ذخیره سازی را مشخص می کند.

4- نوع رکورد (Record type) : دو کاراکتر بعدی نوع رکورد را مشخص می کند که بین 00 تا 05 می تواند باشد. در زیر انواع رکورد های ممکن آورده شده است.

```
00 - data record
01 - end-of-file record
02 - extended segment address record
04 - extended linear address record
05 - start linear address record (MDK-ARM only)
```

5- دیتا (Data) : در این بخش 2 nکاراکتر در فرمت هگز قرار می گیرد که معرف n بایت دیتا می باشد. در صورت عدم وجود دیتا از قرار دادن این بخش صرفنظر می گردد.

6- چک سام (Checksum) : دو کاراکتر بعدی مربوط به چک سام می باشد که برابر است با مجموع دو به دو (بایت به بایت) همه کاراکتر ها و در نهایت مکمل دو (two's complement) نتیجه آن می باشد.

برای مثال اگر رکورد را به صورت زیر

```
:0300300002337A1E
```

در نظر بگیریم داریم:

```
03 + 00 + 30 + 00 + 02 + 33 + 7A = E2
```

که با معکوس کردن تک تک بیت های E2 مکمل دو آن به صورت 1E بدست می آید.

اگر بار دیگر به مثال اولی که برای فایل هگز آورده شده بود مراجعه کنیم ، بخش های مختلف آن به صورت تفکیک شده زیر در می آید.

Example.hex

```
:100100002146013601214701360 07EFE09D2190140
:100110002146017E17C20001FF5F16002148011928
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

Start code | Byte count | Address | Record type | Data | Checksum

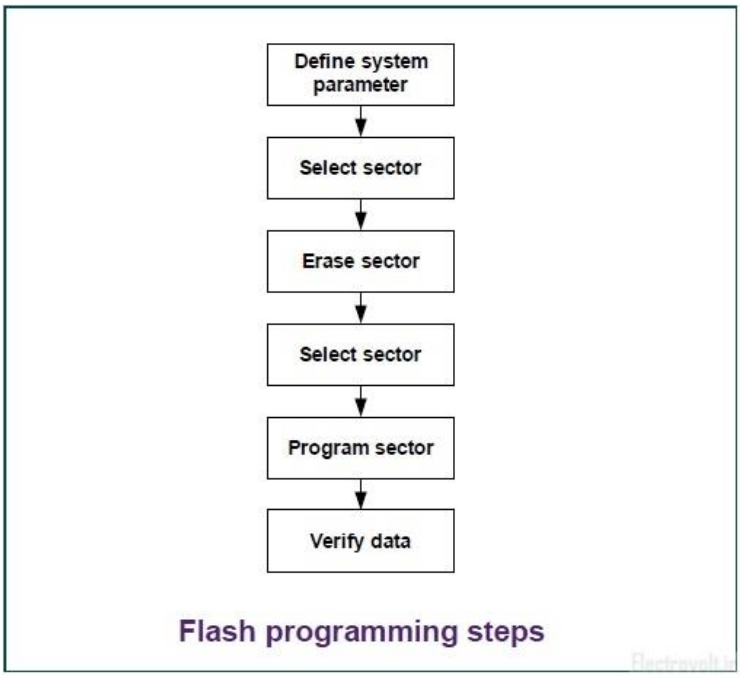
نحوه استفاده از IAP

عبارت iap مخفف in application programming به معنای برنامه ریزی درون اپلیکشن می باشد و در هنگام نوشتن برنامه برای بوت لودر در میکروکنترلرها از آن استفاده می شود. هدر فایل lpc177x_8x_iap.h نیز بخشی از CMSIS بوده و برای برنامه ریزی میکروکنترلرهای Cortex M3 نوشته شده است. بر اساس AN10835 در میکروکنترلرهای ساخت شرکت NXP به هنگام خروج از کارخانه 8 کیلوبایت برنامه بوت لودر به ابتدای آن اضافه می شود. این برنامه که به Secondary USB Boatloader معروف است به طور کامل در AN10866 آورده شده است. در نتیجه از آدرس 0x0000 تا آدرس 0x2000 به این بوت لودر اختصاص یافته است. وجود این بوت لودر باعث می شود تا بتوان توسط متصل کردن USB به یک سیستم خارجی نظیر کامپیوتر برنامه اپلیکشن میکروکنترلر را از آدرس 0x2000 به بعد وارد کرده و بوت لودر آن را با برنامه اپلیکشن قبلی جایگزین می کند.

دستورات موجود در iap بر اساس سکتور ها عمل می کند. یعنی برای مثال به هنگام پاک کردن کل سکتور پاک می شود. در شکل زیر لیستی از این دستورات و کد آنها را در مبنای 10 مشاهده می کنید :

IAP Command	Command Code
Prepare sector(s) for write operation	50_{10}
Copy RAM to flash	51_{10}
Erase sector(s)	52_{10}
Blank check sector(s)	53_{10}
Read Part ID	54_{10}
Read Boot code version	55_{10}
Compare	56_{10}
Reinvoke ISP	57_{10}

شکل زیر نیز گام های مورد نیاز برای استفاده از توابع موجود در iap.h به منظور برنامه ریزی حافظه فلش را نشان می دهد.



در گام اول ضمن اضافه کردن iap.h و iap.c به پروژه ، برخی از ثوابت باید در ابتدای برنامه تعریف شود که آن ها را در شکل زیر مشاهده می کنید:

```
#define IAP_CLK Fcclk

#define IAP_LOCATION 0x7FFFFFF1
#define iap_entry(a, b)      ((void (*)())(IAP_LOCATION))(a, b)

unsigned long command[5] = {0,0,0,0,0};
unsigned long result[3]= {0,0,0};
```

Define system parameter

در گام دوم با استفاده از تابع PrepareSector سکتورهای مورد نظر را برای پاک شدن آماده می کنیم. الگوی این تابع به صورت زیر است که در آن شماره سکتور ابتدایی و انتهایی به عنوان ورودی های تابع هستند.

```
PrepareSector(uint32_t start_sec, uint32_t end_sec);
```

در گام سوم با استفاده از تابع EraseSector سکتورهای آماده شده در گام دوم را پاک می کنیم. الگوی این تابع به صورت زیر است که در آن شماره سکتور ابتدایی و انتهایی به عنوان ورودی های تابع هستند.

```
EraseSector(uint32_t start_sec, uint32_t end_sec);
```

در گام چهارم مجددا با استفاده از تابع PrepareSector سکتورهای مورد نظر را برای برنامه ریزی شدن آماده می کنیم.

در گام پنجم با استفاده از تابع CopyRAM2Flash میتوان دیتای موجود روی حافظه SRAM را به منظور پروگرام شدن به حافظه Flash انتقال داد. الگوی این تابع به صورت زیر است که در آن source آدرس شروع بافر مبدا در حافظه SRAM و dest آدرس شروع بافر مقصد در حافظه فلش و size سایز آرایه ها می باشد. توجه کنید که تعداد بایت هایی که در هر بار میتوان روی حافظه فلش ریخت باید مضربی از 256 باشد.

```
CopyRAM2Flash(uint8_t * dest, uint8_t* source, IAP_WRITE_SIZE size);
```

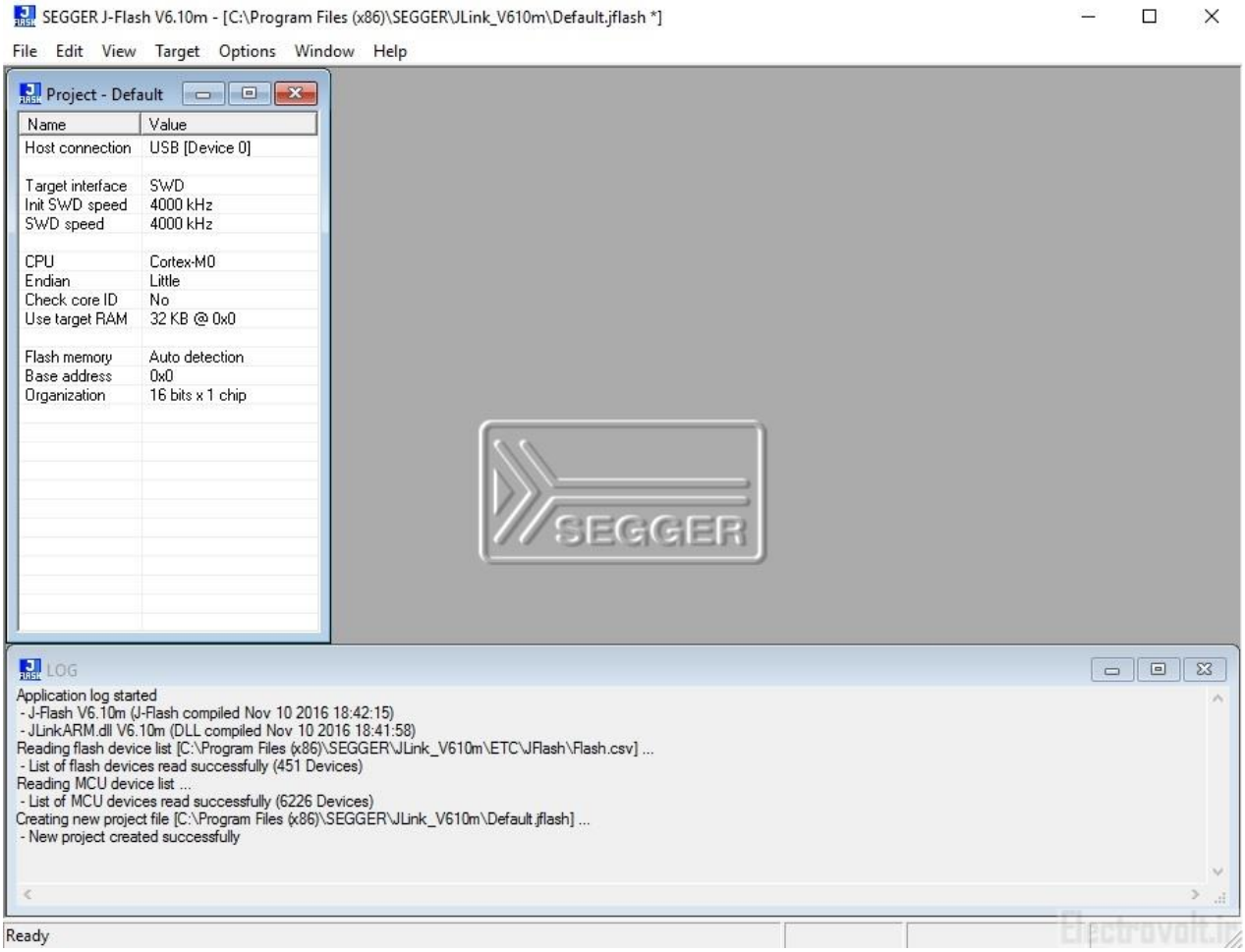
در گام ششم با استفاده از تابع Compare میتوان دو بافر را با یکدیگر مقایسه کرد. الگوی این تابع به صورت زیر است که در آن addr1 و addr2 آدرس شروع بافرهای مقایسه شونده در حافظه های SRAM/Flash هستند و size هم سایز مقایسه شدن آن ها است.

```
Compare(uint8_t *addr1, uint8_t *addr2, uint32_t size);
```

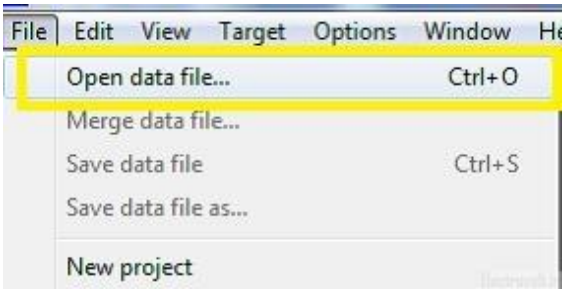
ترکیب دو برنامه بوت لودر و اپلیکشن

بعد از طراحی برنامه های بوت لودر و اپلیکشن به صورت جداگانه ، دو فایل هگز جداگانه برای هر یک بوجود می آید. برای ریختن برنامه نهایی روی میکروکنترلر با استفاده از پروگرامر نظیر Jlink ابتدا این دو فایل را با هم ترکیب می کنیم و بعد از تبدیل شدن به یک فایل هگز آن را پروگرام می

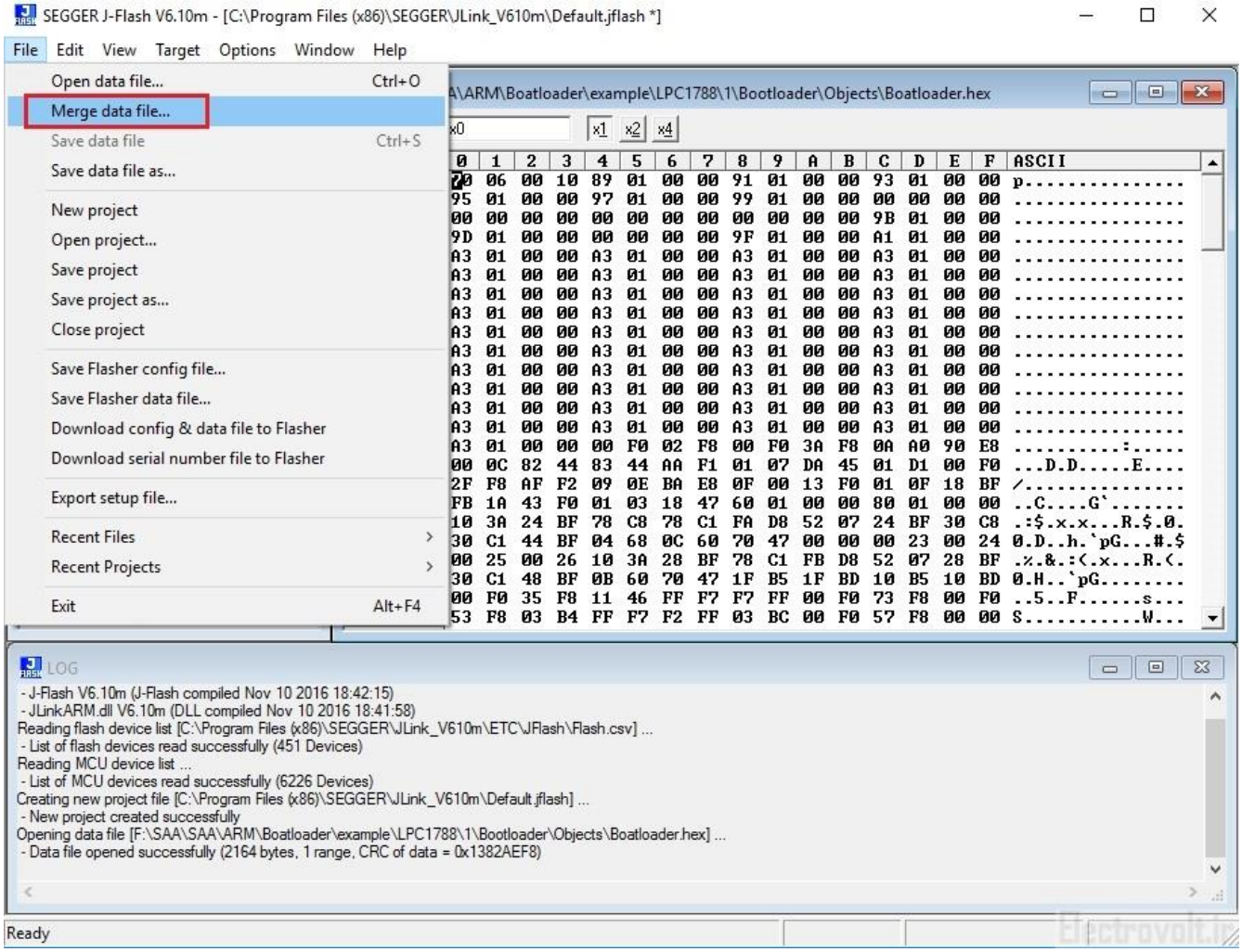
کنیم. برای ترکیب (Combine) کردن دو فایل هگز و تولید یک فایل هگز راه های مختلفی وجود دارد. بهترین راه استفاده از نرم افزار اصلی Jlink یعنی نرم افزار J-Flash می باشد. بعد از دانلود و نصب آن از طریق سایت segger.com این نرم افزار را باز می کنیم. به هنگام باز شدن پنجره زیر را مشاهده می کنید:



از منوی File گزینه Open data file را انتخاب کرده و یکی از فایل های هگز را باز می کنیم.



سپس مجددا از منوی File گزینه Merge data file را انتخاب کرده و فایل هگز دومی را انتخاب می کنیم.



به محض انتخاب فایل دوم ترکیب شده آن ها را مشاهده می کنید. در نهایت فایل ترکیب شده را از طریق منوی File و گزینه Save data file as ذخیره می کنیم.

دانلود سورس برنامه Keil مربوط به این مقاله شامل:

- سورس فایل برنامه Boatloader نوشته شده درKeil V5.15
- سورس فایل یک برنامه Application نمونه نوشته شده درKeil V5.15

لینک دانلود سورس برنامه فوق از فروشگاه الکترو ولت