Firmware-Updates – sicher und schnell:

Funktions- und Cyber-Sicherheit bei Firmware-Updates durch den Einsatz eines professionellen Bootloader-Systems

von Florian Seibold

In jedem komplexeren Elektronikgerät sind heute Mikrocontroller verbaut, die eine Firmware ausführen. Diese Software muss von Zeit zu Zeit aktualisiert werden. Firmware-Updates zielen in der Regel darauf ab, die Funktionalität zu erweitern oder die Sicherheit zu erhöhen. Wird eine dieser Anforderungen nicht erfüllt, sind Fehlfunktionen und Sicherheitslücken vorprogrammiert. Mit dem Bootloader-System semf steht dem Embedded Engineer ein professionelles Toolset zur Verfügung, mit welchem er seine Programmieraufgaben schneller, sicherer und komfortabler erledigen kann.

Ob Fehlerbeseitigung oder Funktionserweiterungen – Embedded Systeme müssen in immer kürzeren Zyklen upgedatet werden. Firmware-Aktualisierungen sind jedoch häufig kritische Arbeitsschritte und für den Entwickler meist mit großem Zeitaufwand verbunden.

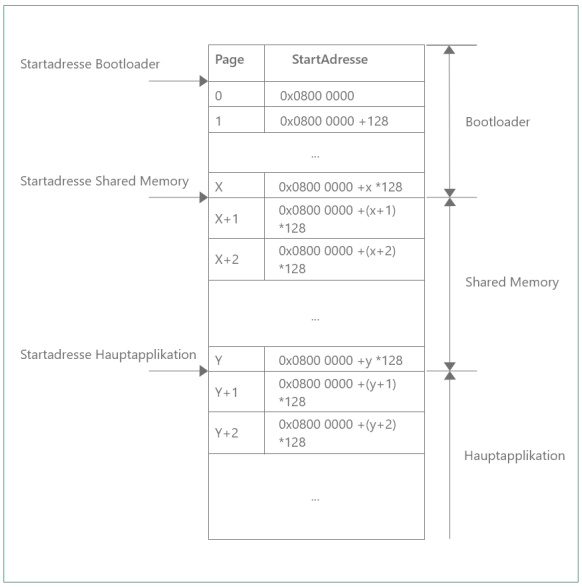
Viele Mikrocontroller-Hersteller liefern Pakete für den Firmware-Updateprozess mit. Diese sind in der Regel funktional. Sobald jedoch Anforderungen wie „Failsave bei Stromausfall während des Updateprozesses" oder „Schutz gegen Hackerangriffe" bestehen, muss die Software hohen Standards genügen und mit maximaler Professionalität programmiert sein.

Um dem Leser ein tieferes Verständnis für die Funktionsweise eines Mikrocontrollers und den damit verbundenen Risiken bei Firmware-Updates zu vermitteln, soll in diesem Beitrag auf Mikrocontroller-Grundlagen sowie auf die Themen Integritätscheck und Verschlüsselung eingegangen werden.

Aufteilung der Speicherbereiche eines Mikrocontrollers

Ein grundlegender Aspekt ist die Aufteilung der Speicherbereiche innerhalb von Flash-Speichern im Allgemeinen und der Mikrocontroller-internen Speicherbänken im Speziellen. Dabei ist besonders zu beachten, dass:

- Flash-Speicher in Sektoren (Sector) und Seiten (Pages) unterteilt sind.
- ein Sektor in der Regel mehrere Seiten beinhaltet.
- ein Sektor der kleinste löschbare Speicherbereich ist.
- beim Schreiben darauf geachtet werden muss, dass üblicherweise in einem Schreibbefehl nicht über die Grenzen einer Seite hinaus geschrieben werden kann. Es empfiehlt sich grundsätzlich dies in zwei Schreibbefehle zu trennen (einer pro Seite).
- je nach Flash-Technologie der Speicherbereich zwar Bytewise (8 Bit) gelesen, aber ggf. nur Half-Word (16 Bit) oder Wordwise (32 Bit) geschrieben werden kann. Dies kann zu sehr schwer nachvollziehbaren Fehlerbildern führen.



Im vorliegenden Beispiel wird davon ausgegangen, dass alle Seiten des internen Flashs des Mikrocontrollers 128 Bytes groß sind.

In der Regel liegt der Bootloader am Anfang des Speichers. In unserem Beispiel schließt daran ein Shared Memory an, in welchem Bootloader und Applikation relevante Informationen austauschen können, beispielsweise ob eine neue Firmware geflasht werden muss. An dritter Stelle liegt die Applikation, die in unserem Fall das Firmware-Update anstößt, das dann im Laufe des Prozesses überschrieben wird.

Nutzen des semf Bootloaders beim Speichermanagement

Im Rahmen des Speichermanagements bietet das Bootloader-System **semf** einige nützliche Vorteile. So abstrahiert **semf** die verwendeten Speichertechnologien und bietet Lösungen für konkrete Problemstellungen durch:

- Basisklassen für Flash- und EEPROM-Speicher
- Implementierungen für interne Flashbereiche von Mikrocontrollern sowie externe Flashs via I2C, bzw. SPI angebundene Speicherbausteine
- Offene Architektur für Anpassungen in Spezialfällen
- Gute Lesbarkeit des Sourcecodes durch intuitive API

Einsatz von Linker-Skripten für korrekte Adressierung

Der Linker ist dafür zuständig die jeweiligen Binärdaten an die korrekte Stelle im Speicher zu schreiben. Im Rahmen eines Firmware-Updates muss in der Regel für Bootloader und Applikation ein eigenständiges Projekt definiert und an die jeweils korrekte Stelle im Flash adressiert werden. In den jeweiligen Linkerskripten kann die Startadresse definiert werden, an die die Firmware geschrieben wird.

Wichtig dabei zu beachten ist, dass:

- die Startadresse des Bootloaders mit der des Reference-Manuals übereinstimmen sollte bzw. der Standardeinstellung von neu angelegten Projekten entspricht.
- die Startadresse der Applikation dieselbe sein muss, zu der auch vom Bootloader aus gesprungen wird.

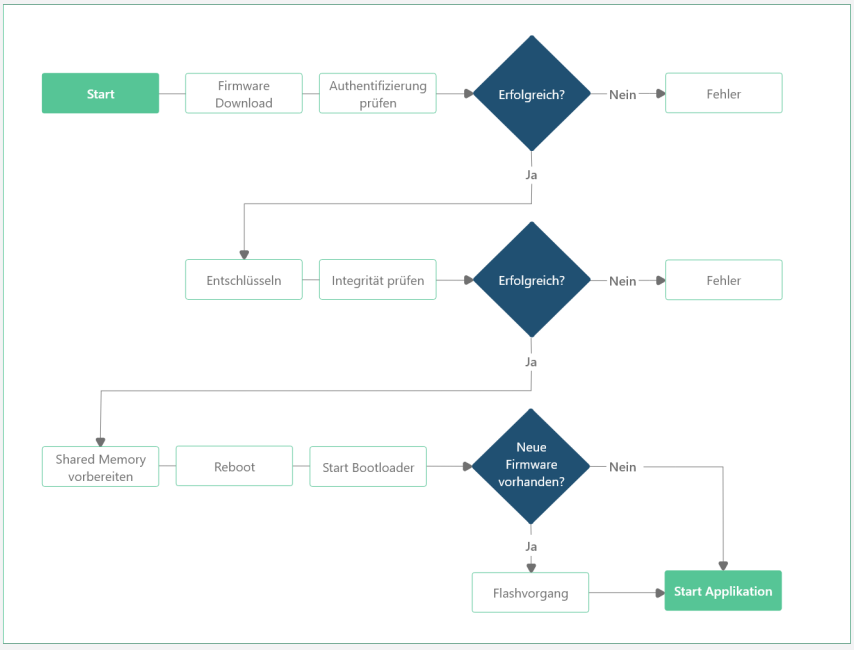
In drei Schritten zum sicheren Firmware-Update:

1. Schritt:
Firmware Download

2. Schritt:
Firmware Check

3. Schritt:
Boot-Prozess

Ablauf Bootloader Prozess



1. Schritt: Firmware Download

Um ein Firmware-Update durchführen zu können, muss das neue Binary zuerst auf das Gerät bzw. einen externen Speicher geladen werden. Diese Aufgabe kann mit einem File-Download mittels klassischen Kommunikationsschnittstellen wie CAN, USB oder Ethernet durchgeführt werden.

2. Schritt: Firmware Check

Nachdem das Binary komplett ist, werden Größe, Authentifizierung und Integrität geprüft. Sofern das Firmware-File über einen Header verfügt, in welchem beispielsweise die Version hinterlegt ist, können auch Downgrades oder andere Funktionen gestartet oder blockiert werden. Nach Abschluss der Prüfungen werden alle notwendigen Flags im Shared Memory gesetzt. Damit weiß der Bootloader, dass eine neue Firmwareversion zur Verfügung steht und wo sie im Speicher abgelegt ist. Abschließend wird der Mikrocontroller neu gestartet und landet damit im Bootloader.

```
// Address where the shared data is stored in the internal flash memory.
constexpr uint32_t kSharedMemoryAddress 0x08004400
// Address where the firmware file is stored in the external flash memory.
constexpr uint32_t kExternalFlashFirmwareFirstSector 12

// Classes depend on the used microcontroller and external flash hardware
// For reset
semf::Power power;
// Where the shared menory is located.
semf::Flash internalFlash;
// Where the new firmware file is stored.
semf::Flash externalFlash;

// Decryption
uint8_t aesData[16];
semf::AesCbcMbed aesCbc(aesData, sizeof(aesData));
// Hash computation
uint8_t sha256Data[32];
semf::HashSha256Mbed sha256(sha256Data, sizeof(sha256Data));
// Verification with signature
uint8_t pkcs1Data[128];
semf::SignaturePkcs1Mbed pkcs1(pkcs1Data, sizeof(pkcs1Data));
semf::BootloaderSharedMemoryFlash sharedMemory(internalFlash, kSharedMemoryAddress);
uint8_t firmwareupdaterData[1024];
semf::FirmwareUpdater firmwareupdater(externalFlash, aesCbc, sha256, pkcs1,
  sharedMemory, firmwareupdaterData, sizeof(firmwareupdaterData));

/**
 * Slot for firmware is ok and firmare updater is finished.
 */
void onFirmwareReady()
{
  // do system reset
  powerModes.reset();
}

// Start
firmwareupdater.start(externalFlash.address(kExternalFlashFirmwareFirstSector));
```

3. Schritt: Boot-Prozess

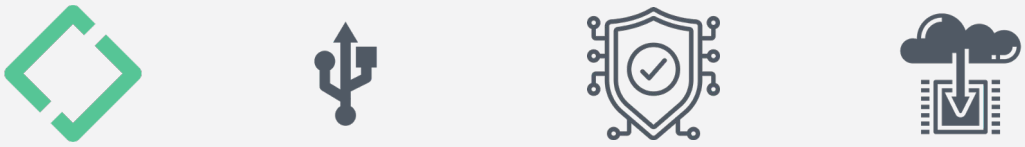
Der Bootloader ist einfach aufgebaut. Er prüft im Shared Memory, ob eine neue Firmware verfügbar ist und startet dann den Prozess des Überschreibens der alten Datei. Sobald dieser Prozess abgeschlossen ist, wird die Applikation neu gestartet.

```
constexpr uint32_t kSharedMemoryAddress 0x08004800
constexpr uint32_t kApplicationAddress 0x08005400

uint8_t data[128];
semf::BootloaderSharedMemoryFlash sharedMemory(kSharedMemoryAddress);

// Classes depend on the used microcontroller and external flash hardware
semf::Flash externalFlash;
semf::Flash internalFlash;
semf::Bootloader bootloader(externalFlash, internalFlash,
  sharedMemory, kApplicationAddress, data, sizeof(data));

bootloader.start();
// Blocks until bootloader is finished
while(!bootloader.isready());
bootloader.startApplication();
```

Nutzen des **semf** Bootloaders beim Booten

Im Rahmen des Boot-Prozesses bietet der **semf** Bootloader die folgenden Vorteile:

- verschlüsselte Firmware-Updates mit Zertifikatsschutz
- Flashen von beliebig vielen Speicherbausteinen während des Updates
- Fail-Save bei z.B. Spannungsausfall



SICHERE PROZESSE

Schutzmaßnahmen für einen sicheren Updateprozess

Bei der Softwarekonzeption ist der Bereich „Firmware-Update“ in der Regel der anspruchsvollste Teil, denn es soll sichergestellt sein, dass:

- regelmäßige Updates möglich sind, idealerweise fail-save
- der Hersteller vor Hackerangriffen geschützt ist
- sämtliche Nutzerdaten sicher sind und geschützt bleiben

AUTHENTIFIZIERUNG

Durch die Verwendung von Zertifikaten nach z.B. PKCS1 Standard in Verbindung mit einem Private Key, den nur der Hersteller selbst verwendet, kann weitestgehend sichergestellt werden, dass Firmware von Dritten nicht aufgespielt werden kann.

INTEGRITÄT

Um sicherzustellen, dass die neu zu flashende Firmware funktionssicher ist, sollte ein Integritätscheck durchgeführt werden. Übliche Verfahren sind eine einfache CRC oder ein SHA-2 Hash. Die CRC oder der Hash wird der Firmware z.B. im Header mitgegeben. Während des Firmwarechecks wird dieser geprüft und nur bei Korrektheit wird der Prozess fortgeführt.

VERSCHLÜSSELUNG

Wird zusätzlich noch das Binary der Firmware verschlüsselt, bevor es ausgeliefert wird, ist größtmögliche Sicherheit gewährleistet, weil so der Key für die Authentifizierung nicht aus der Binärdatei ausgelesen werden kann.

Fazit

Die Umsetzung eines sichern und fehlerfreien Firmware-Updateprozesses ist eine anspruchsvolle Aufgabe und verlangt tiefes Detailwissen über die Funktionsweise von Mikrocontroller und Speichertechnologien, der Kryptographie sowie der verwendbaren Toolchain. Bei der Projektkonzeption sollte besonderes Augenmerk auf Anforderungen nach Authentifizierung, Integrität und Verschlüsselung gelegt werden.

Das **semf** Bootloader-System wurde für den professionellen Einsatz entwickelt und bietet Einsteigern eine stabile Basis und Profis die notwendige Flexibilität und Offenheit für spezielle Anforderungen und Konfigurationen. Der **semf** Entwickler-Support hilft beim Onboarding und steht jederzeit mit Rat und Tat zur Seite. Mehr Informationen zu Firmware-Update-Verfahren und zum **semf** Bootloader auf www.semf.io.

KONTAKT
go@semf.io
+49 (0) 7807 / 890 80 30

semf

www.semf.io

25.01.2022