LARUS

Bildquelle : wallsdesk.com

BootLoader

Technische Referenz

und

Benutzerhandbuch [1] [2]

1 Zur Nomenklatur : Handbuch Version h.m.n.x korrespondiert mit Software-Version h.m.n und ist die x.te Text-Iteration passend zu dieser Software-Version.

2 Die Ornithologen mögen mir verzeihen. Das ist keine Möwe, sondern ein Albatros. Aber es ist ein schönes Bild. Außerdem : Einige der Kontributoren zu diesem Projekt haben schon früher zusammen an einem Variometer gearbeitet, das hieß „Albatros“.

1 Problemstellung

Die Bedienung des LARUS - erfolgt über ein „FrontEnd“, ein Gerät im Cockpit. Hier eine Abbildung.

Das Gerät ist baugleich mit dem Anzeigegerät (AD57) des Air Avionics Air Traffic Display (ATD1). Durch Kooperation mit Air Avionics wurde es möglich, diese Geräte zu einem annehmbaren Preis zu erwerben. Sie wurden ihrer eigenen Firmware entkleidet und erhielten (wie hier unschwer zu erahnen ist) eine andere Firmware, um sie als Vario zu nutzen.

Wesentlich : Die Front dieser Geräte besitzt einen Slot für uSD-Karten.

Erfahrungsgemäß muss die Vario-Firmware im Lauf der Zeit auf einen jeweils weiter entwickelten Stand gebracht werden[3], sie muss einen „Update“ erfahren..

Stand der Technik ist, der eigentlichen Applikation, hier dem Vario, gleich eine Unterstützungsfunktion mitzugeben, mit deren Hilfe diese „Updates“ erfolgen können. In vielen Systemen, z.B. FLARM, wird das sehr einfach gelöst : Beim Startvorgang muss eine uSD-Karte im Slot stecken oder USB-Stick im Gerät, die die neue SW beinhalten. Wenn das System startet, kann es feststellen, ob die eigene, schon geladene SW-Version älter, gleich alt, oder sogar jünger ist als die SW auf der Karte/dem Stick. Wenn die SW auf der Karte jünger ist, wird ein Ladevorgang angestoßen, der die neue Software in das Gerät lädt (Jargon : „flasht“).

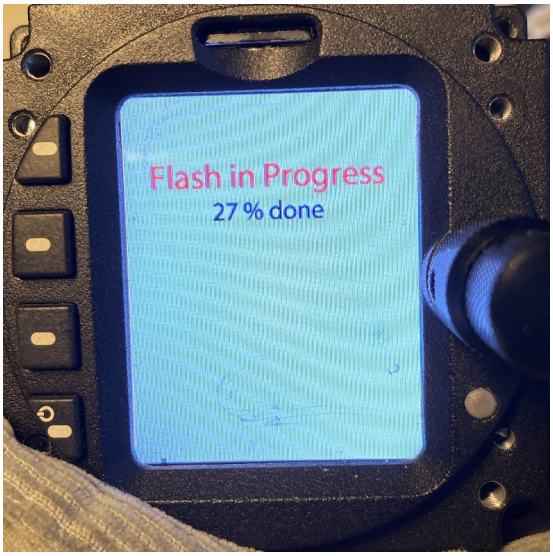
Auf diese Weise kann auch ein Benutzer im Feld, ohne Rückgriff auf die Ersteller der SW, den Update einfach durchführen : Er lädt sich die neueste Version aus dem Netz auf seinen Datenträger, steckt ihn vor dem nächsten Systemstart ins Gerät, schaltet ein – und fertig ist der Update.

Diese Methode hat jedoch einen gravierenden Nachteil :

Es ist (ohne Tricks, die zu nutzen dem eben zitierten Benutzer im Feld nicht möglich ist) schlicht nicht möglich, von einer jüngeren SW-Version zurück zu springen auf eine ältere. Was gebraucht wird, ist ein interaktiver BootLoader, der es ermöglicht auf einer Benutzeroberfläche zu entscheiden, welche SW-Version geladen werden soll.

3 Das gibt natürlich gleichermaßen für die anderen Systembestandteile (Main Sensor, Audio Utility Board, etc.).

Der hier beschriebene BootLoader leistet genau das. Der Benutzer bekommt die Wahl zwischen SW-Versionen. Die gelbe Cursor-Zeile wird mit dem äußeren kleinen Drehknopf auf die SW der gewünschten Version verschoben, mit dem doppelten SEL-Tastendruck wird diese SW ausgewählt und „geflasht“.



Nach Abschluss dieser Operation erscheint die Vario-Applikation

Soweit die erste Einführung. Der Rest des Handbuchs macht sie, Leserin oder Leser, detailliert mit dem bekannt, was hinter den Kulissen ablaufen muss, damit dieser Vorgang möglich wird.

2 Speicher-Organisation im AD57-Controller

Das AD57 wird getragen von einem Rechnerkern („Controller“) des Typs STM32F409VTG, hier kurz „F4“ genannt. Der Flash-Speicher des F4 ist in Segmenten organisiert :

```
//
//  Sector description table for STM32F4xx
//
ROM sector_descr_t Sector_Table[12] =
  { //first addr    last addr  sec_size  remaining_flash_size                  secno
    { 0x08000000,  0x08003FFF, 16*1024,  7*128*1024 + 64*1024 + 4*16*1024 },   //  0
    { 0x08004000,  0x08007FFF, 16*1024,  7*128*1024 + 64*1024 + 3*16*1024 },   //  1
    { 0x08008000,  0x0800BFFF, 16*1024,  7*128*1024 + 64*1024 + 2*16*1024 },   //  2
    { 0x0800C000,  0x0800FFFF, 16*1024,  7*128*1024 + 64*1024 + 16*1024   },   //  3
    { 0x08010000,  0x0801FFFF, 64*1024,  7*128*1024 + 64*1024             },   //  4
    { 0x08020000,  0x0803FFFF, 128*1024, 7*128*1024                       },   //  5
    { 0x08040000,  0x0805FFFF, 128*1024, 6*128*1024                       },   //  6
    { 0x08060000,  0x0807FFFF, 128*1024, 5*128*1024                       },   //  7
    { 0x08080000,  0x0809FFFF, 128*1024, 4*128*1024                       },   //  8
    { 0x080A0000,  0x080BFFFF, 128*1024, 3*128*1024                       },   //  9
    { 0x080C0000,  0x080DFFFF, 128*1024, 2*128*1024                       },   //  10
    { 0x080E0000,  0x080FFFFF, 128*1024, 128*1024                         }    //  11
  };
```

Der untere Teil des Adressraums (0x08000000 – 0x0807FFFF) ist in viele kleine Segmente geteilt, im oberen Adressraum (0x08080000 – 0x080FFFFF) ist die Segmentstruktur gröber.

Im unteren Adressraum residiert der BootLoader, im oberen Adressraum die Applikation.

Der Zweck des BootLoaders ist, wie beschrieben, den HEX-File einer auf der lokalen µSD-Karte gespeicherten Applikation in den oberen Adressraum zu schreiben (zu flashen) und dann diese Applikation zu starten[4].

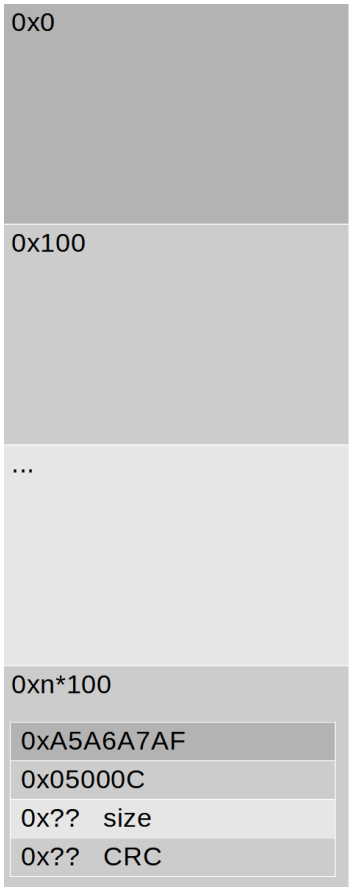
4 Das vorliegende Papier beschreibt den BootLoader „as-is“. Es ist geplant, den BootLoader so zu erweitern, dass auch Satelliten-Systeme (Sensor, Audio) geflasht werden können. Diese Flash-Vorgange verlaufen nach einem anderen Schema als der hier beschriebene Flash-Vorgang im AD57 selbst. !!! ToDo !!!

3 Bau des HEX-Files für den AD57

Das AD57 verfügt über ein EEProm, das in diesem Kontext benutzt wird, um die Ausführung der BootLoading-Operation zu steuern.

Wenn eine Applikation erfolgreich geflasht worden ist und dann gestartet wird, wird sie SOFORT einige Daten in das EEProm schreiben : *Die Signatur der Anwendung*. Die folgende Zeichnung soll erklären, was das ist und wie das geht.

```
__attribute__((__section__(".signature"))) uint32_t ThisApplicationsSignature[4] =
    {
     0xA5A6A7AF,
     VERSION_TXT6,
     0xF1F1F1F1,
     0xE2E2E2E2
    };
```



Durch geeignete Befehle im Linker-File der Applikation wird der aus dem IDE[5]-Build hervorgegangene native BIN-File (z.B. vario.bin) erweitert (verlängert) um die Datenstruktur der Signatur. Mehr noch : Die Signatur wird so in den BIN-File eingebaut, dass sie auf eine 256-byte-Blockgrenze zu liegen kommt. Das wird in der nebenstehenden Skizze des Layouts der Files <vario.bin> deutlich.

Zum Aufbau der Signatur :

1. Der erste Wert wird „magic number“ genannt. Diese Zahl dient als Erkennungsmuster, was ein paar Zeilen weiter unten klar wird.
2. Der zweite Wert enthält die Version und Build-Number des vorliegenden HEX-Files (z.B. 0x0005000C). Dieser Wert wird für die Generierung des neuen Namens für die HEX-File-Ausgabe gebraucht.
3. Der dritte Wert ist die Länge des Hex-Files in Bytes.
4. Der vierte Wert ist der CRC-Wert der Applikation.

Der dritte und der vierte Wert sind zum Zeitpunkt des Build in der IDE noch nicht bekannt. Sie müssen im Nachgang ermittelt werden und in den BIN-File eingesetzt werden. Diese Aufgabe hat das Werkzeug „AfterBurner“, hier im Folgenden grob beschrieben.

Der „Afterburner“ liest den BIN-File sequentiell und findet dabei seinen Synchronisationspunkt in der „magic number“ im ersten 32-bit-Wort eines (des letzten) 256-byte-Blocks. Dabei wird die Länge des Files bis dorthin berechnet, die 32 Byte der Datenstruktur kommen noch hinzu und dieser Wert wird dann ins vorletzte 32-bit-Wort der Datenstruktur geschrieben.

Parallel errechnet der „AfterBurner“ den CRC, allerdings AUSSCHLIESSLICH des letzten 32-bit-Wortes, und setzt den CRC dann ins letzte Wort ein.

Zuletzt wird der gesamte File als HEX-File mit einem neu konstruierten Namen ausgegeben.

Dieser Hex-File muss jetzt noch seinen Weg auf die µSD-Karte finden, die in den AD57 gesteckt wird.

5 Der Autor dieses Papiers nutzt den STM32FXCube als IDE und als Vehikel seiner Wahl, um die Applikation zu bauen. Statt dessen kann hier auch jedes andere Werkzeug mit vergleichbarem Funktionsumfang genutzt werden.

Der ganze Vorgang ist eigentlich übersichtlich. Die Problemwürze liegt darin, die einzelnen Schritte prozedural so hintereinander zu schalten, dass ohne Irrtümer immer ein ladbarer HEX-File entsteht.

Der „AfterBurner“ existiert in zwei Versionen : ein EXE für die WIN-10-Umgebung und ein EXE für die UNIX-Umgebung.

In der WIN-10-Umgebung wurde das EXE mithilfe von MS Visual Studio erstellt.
In der UNIX Umgebung konnte das EXE mit der IDE selbst gebaut werden.

Die Batch-Files zum Umschalten der Loader-Files und für den Aufruf des „Afterburners“ gibt es bisher nur in der WIN-10-Umgebung.

In der IDE (STM32FXCube aka Eclipse) des Autors stehen 3 verschiedene Linker-Files zur Verfügung, um den VARIO-bin-File zu erzeugen :

- als nicht relokatierten EXE, der auf 0x08 000 000 startet (normaler Build)[6]
- als relokatierten EXE, der auf 0x08 080 000 startet (Build für den BootLoader)
- als relokatierten EXE mit „Trampoline“
  (Trampoline bei 0x08 000 000, die Applikation bei 0x08 080 000) macht nur zum Testen Sinn

Mit Hilfe von Batch-Files, die im Verzeichnis <VersionManagament> residieren und als <external tools> ansprechbar sind, wird der Standard-Linker-File <STM32F407VGTX_FLASH.ld> mit einem der drei anderen Linkerfiles

- <STM32F407VGTX_FLASH_not_relocated>
- <STM32F407VGTX_FLASH_relocated_no_trampoline.ld>
- <STM32F407VGTX_FLASH_relocated_with_trampoline.ld>

überschrieben.

Nach dem Build muss der BIN-File noch von dem oben schon erwähnten „AfterBurner“ bearbeitet werden. Dabei entsteht der ladbare HEX-File.

6 Der BootLoader selbst als Applikation residiert IMMER auf Adresse 0x08000000.

4 Der AD57-Flash-Vorgang

Betrachtet man den eingeschwungenen Zustand (d.h. wenn zuvor schon einmal eine Applikation erfolgreich geflasht und gestartet worden war), dann startet ein Cold-Reset den BootLoader im unteren Adressraum, der dann, wenn er im EEProm Daten vorfinden, die zur Applikation im oberen Adressraum passen (was das heißt, wird weiter unten erklärt), sofort die Applikation im oberen Adressraum startet. Das ist der normal gebräuchliche **automatische Startvorgang** beim Einschalten des Varios.

Dieser automatische Startvorgang wird unterbrochen, wenn der BootLoader auf eine der folgenden Bedingungen trifft :

- Das System ist jungfräulich, im oberen Adressraum existierte noch nie eine Applikation, und das EEProm ist deshalb auch noch nicht mit „richtigen“ Daten bestückt.
- Das EEProm trägt Daten, die nicht zur Applikation passen. Das ist theoretisch möglich, praktisch sehr unwahrscheinlich, aber als Fehlersituation abgefangen.
- Der Anwender hält im Moment des Resets die oberste Taste am AD57 gedrückt.

Wenn der Bootloader auf diese Weise im **interaktiven Modus gestartet** wird, erscheint zunächst eine LandingPage mit folgendem Inhalt :

Wenn der Anwender auf ESC drückt, werden die Signatur im EEProm und die Signatur der bereits geladenen Applikation gegeneinander verprobt und – bei positivem Ergebnis der Verprobung – wird die Applikation im oberen Adressraum gestartet, analog zum **automatischen Startvorgang**.

Wenn der Anwender den dicken Knopf nach rechts dreht, erscheint eine Menue-Seite, auf der die zu diesem Zeitpunkt auf der µSD-Karte enthaltenen HEX-Files (hier VARIO*.hex) zu sehen sind[7].

7 Nach der ersten Raste erscheinen die HEX-Files für das Vario, nach der zweiten Raste erscheinen die HEX_Files für das Audio, nach der dritten Raste erscheinen die HEX-Files für den Sensor. Wie an anderer Stelle schon erwähnt, soll der BootLoader so erweitert werden, dass damit auch die Satelliten (Audio, Sensor) neu geflasht werden können. TODO

Der Benutzer bekommt hier die Wahl zwischen SW-Versionen. Das Einspeichern der Firmware wird im Fachjargon „flashen“ genannt (siehe im Bild : 2 * select to flash). Die gelbe Cursor-Zeile wird mit dem äußeren kleinen Drehknopf auf die SW der gewünschten Version verschoben, mit dem doppelten SEL-Tastendruck wird diese SW ausgewählt und „geflasht“.

Nach einem kurzen Moment des bangen Wartens (in dieser Zeit erfolgt der Erase des Flash-Speichers) erscheint eine Seite, die über den Flash-Fortschritt informiert. Das Flashen des Varios dauert ca. 40 Sekunden.

Nach Abschluss dieser Operation erscheint die Seite *<Vario-1>* der Vario-Applikation .

Wenn die EEProm-Daten das nicht zulassen (== Unterbrechung des automatischen Startvorgangs), erscheint die LandingPage erneut.

Dieser Fall kann eintreten, wenn der Nutzer zuerst einen neuen HEX-File zum Flashen auswählen wollte, es sich dann aber anders überlegt, auf die LandingPage zurückschaltet und die alte Version des Varios doch starten will.

Vorsicht:

Es können nur 8 HEX-Files angezeigt werden. Wenn mehr HEX-Files einer Sorte auf der Karte vorhanden sind, fallen die überzähligen ohne weitere Meldung/Warnung unter den Tisch.

Die HEX-Files erscheinen in der Auswahlmaske in ihrer physikalischen Reihenfolge, in der sie auf der µSD stehen. Das ist nicht notwendigerweise die lexikalische Reihenfolge.

Nun zu den Dingen, die hinter den Kulissen gesichert sein müssen, damit diese Vorgänge laufen können :

Beim Flashen wird der HEX-File blockweise gelesen und – eben geflasht. Um sicherzustellen, dass die Quelldaten auch identisch im Flash angekommen sind, wird jeder geflashte Block aus dem Flash zurückgelesen und mit dem Quell-Datenblock verglichen.

Sobald der Flash-Vorgang abgeschlossen ist, erfolgt noch ein CRC-Test und ein Vergleich der Dateigrößen aus der Quelle und den geflashten Daten.

Beim CRC-Test wird der während des Flashens errechnete CRC-Wert mit dem CRC-Wert verglichen, den der HEX-File in sich mitgebracht hat (erzeugt vom „AfterBurner“).

Werden diese Bedingungen nicht erfüllt, wird der Flash-Vorgang abgebrochen. TODO

Wird nach erfolgreichem Flashen die Applikation gestartet, dann kopiert die Applikation ihre Signatur-Daten aus ihrem eigenen geflashten Code in den ersten Block des EEProm-Speichers, damit beim nächsten Systemstart die Applikation automatisch gestartet werden kann.

Damit wird auch klar, unter welchen Bedingungen es dem BootLoader gestattet ist, beim automatischen Start direkt zur Applikation weiter zu springen : In der kleinen Pause nach dem Reset erfolgt eine Prüfung, ob diese Werte zur Applikation passen.

5 Bau des HEX-Files für einen Satelliten (Audio, Sensor)

Noch leer

6 Der Satelliten-Flash-Vorgang (Audio, Sensor)

Noch leer

7 Blame-Board von LARUS

LARUS konnte nur entstehen durch die Zusammenarbeit und die Beiträge folgender Personen (in alphanumerischer Reihenfolge)

Betz, Max, Segelflieger, Software- und Hardware-Entwickler, ehemaliger Student von Prof. Dr. Klaus Schäfer an der FH Darmstadt

- Hardware- und Software-Entwicklung des Sensor-Boards

- Guru des GitHub-Auftritts

Foerderer, Marc, Segelflieger, CEO und Partner bei Air Avionics

- Unterstützung durch Material-Überlassung

Langer, Stefan, deutscher Spitzenpilot, baut und vertreibt das OpenVario System

- Alpha-Tester, Unterstützung durch Test und Feedback

Leutenegger, Stefan, Schweizer Spitzenpilot, Professor an der TU München, Schwerpunkt-Thema „Steuerung autonomer Systeme“, viele Jahre Erfahrung um Bau von Drohnensteuerungen. hat in 2011 die Windschätzung im Butterfly Air Glide S geschrieben

- Software-Entwicklung Windschätzung

Maier, Felix, Segelflieger, Maschinenbauingenieur (FH), QS-Ingenieur bei Continental

- 3-D-Druck

Rupp, Horst, Segelflieger, Dipl.-Inform., Nestor des Projekts, ehemals Software-Entwickler, dann IT-Manager und IT-Berater, viele Jahre Erfahrung im Bau von Variometern, hat auf die alten Tage Programmieren wieder neu lernen müssen

- Software-Entwicklung FrontEnd und BootLoader

- Hardware- und Software-Entwicklung Prototype Audio

Schäfer, Klaus, Segelflieger, Professor an der FH Darmstadt, Schwerpunkt-Thema “Microcontroller“, viele Jahre Erfahrung im Bau von Drohnensteuerungen und Variometern

- Software-Entwicklung Sensor, AHRS und Windschätzung

- Guru der Entwicklungsplattform STM32CUBEIDE

Simon, Winfried, Segelflieger, Dipl.-Ing. für E-Technik, Software-Entwickler, viele Jahre Erfahrung im Bau von Variometern

- Wichtigster Sparringspartner und Ideenreiniger im Team

- Vorverarbeitung und Darstellung von Flugzeugpolaren für die Nutzung im FrontEnd

Wahlig, Uwe, deutscher Spitzenpilot, Software-Entwickler und SAP-Berater

- Alpha-Tester, Unterstützung durch Test und Feedback