Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

Projekt Bootloader

Dokumentation

Autoren:
Elias Medawar
Jonathan Stoppani
Simon Honegger
Yves Peissard

Supervisor:
Daniel Gachet

19. Juni 2010

Abstract

Dieses Projekt hat zum Ziel im Ramen des Kurses “Mikroprozessoren 2” der Hochschule für Technik und Archi- tektur einen Bootloader auf einem Mikrochip MC68332 zu implementieren. Dabei werden die behandelten Themen des Mikroprozessor Kurses Programmieren in C und Grundlagen Assembler angewendet. Nach Aufstarten des Mi- krokontrollers kann eine externe Maschine (Terminal) mittels serieller Schnittstelle angeschlossen werden, mit der anschliessend Befehle eingegeben werden können um den Bootloader zu konfigurieren. Der Benutzer soll schliesslich unter zwei verschiedenen Anwendungen auswählen können, die er mit dem Bootloader starten will.

Was ist ein Bootloader? Definition von Wikipedia: Ein Bootloader ist eine spezielle Software, die gewöhnlich durch die Firmware (z. B. dem BIOS bei IBM-kompatiblen PCs) eines Rechners von einem bootfähigen Medium geladen und anschließend ausgeführt wird. Der Bootloader lädt dann weitere Teile des Betriebssystems, gewöhnlich einen Kernel.

Inhaltsverzeichnis

Teil I

Architektur

1 Überblick

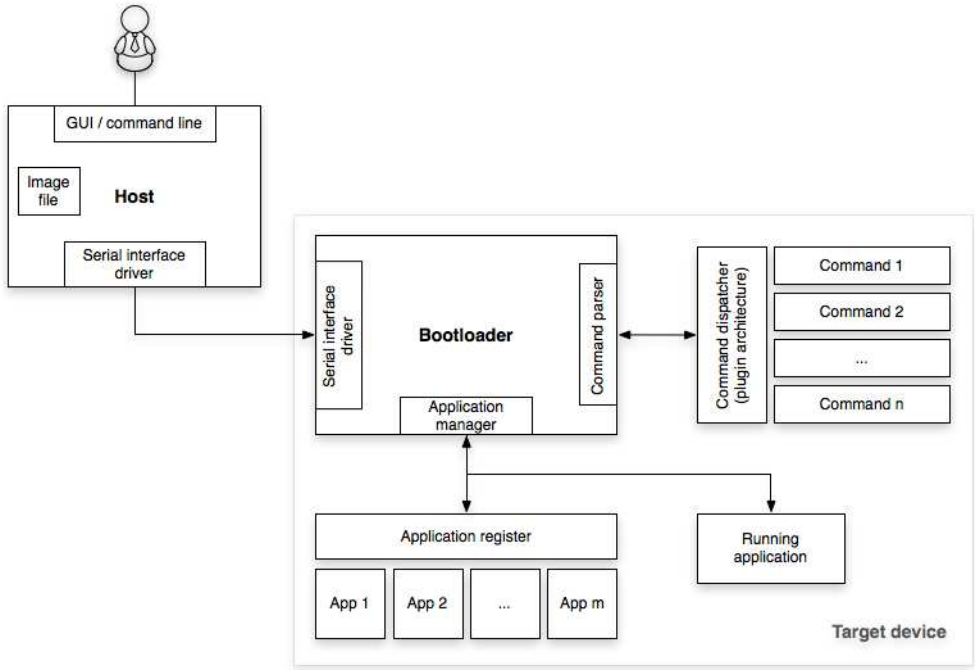


Abbildung 1: Übersicht der Architektur

In der obenstehenden Gesammtübersicht sind die einzelnen Komponenten und deren Verknüpfung dargestellt. Im folgenden Kapitel wird die Architektur der jeweiligen Komponenten näher beschrieben.

2 Application Driver

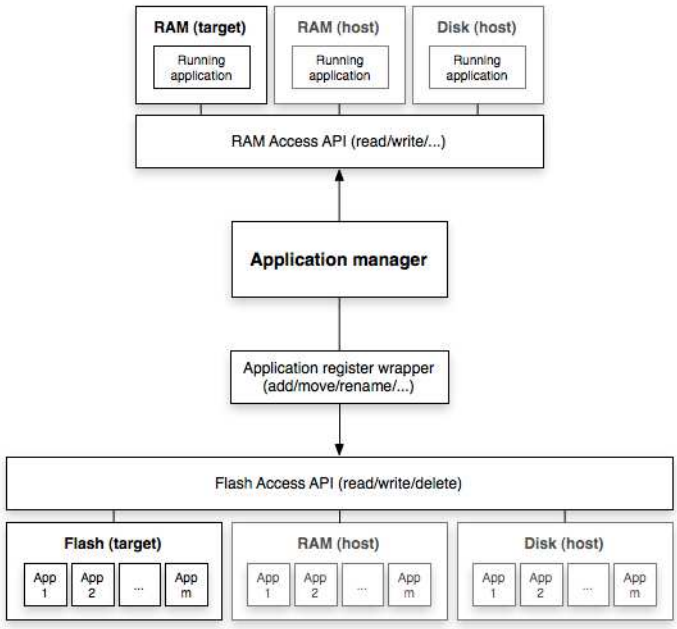


Abbildung 2: Übersicht des Application Driver

Die Aufgabe der Application Driver ist es, das Applikationsverzeichnis zu verwalten und den aktuellen Ihnalt des Flashspeichers in den Arbeitsspeicher zu laden.

2.1 Flash Access API

Mit der Flash Access API können entweder Flashspeicher, Arbeitsspeicher oder Festplatte benutzt werden. Die drei Hauptfunktionen der API sind lesen, schreiben und löschen. Der Zugriff auf den Flashspeicher ist nur mittels Mikro- kontroller MC68332 möglich (Zielplattform). Lesen und/oder Schreiben auf Arbeitsspeicher oder Festplatte sind für Testzwecke gedacht (Debugging).

2.2 Application Register Wrapper

Damit nicht direkt mit der Flash API gearbeitet werden muss, übernimmt der Application Register Wrapper die An- steuerung der Flash Access API. Dies erleichtert das Entwickeln des Bootloaders und ermöglicht den Code des App- lication Driver übersichtlich zu gestalten.

2.3 Serial Interface Driver

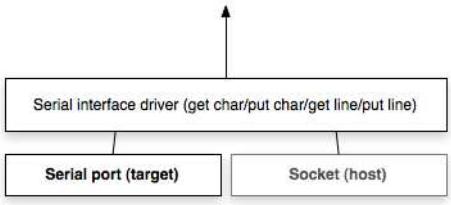


Abbildung 3: Schema des Serial Interface Driver

Der Serial Interface Driver ermöglicht die Kommunikation zwischen Zielplattform (MC68332) und Benutzer des Bootloader. Mittels eines Keyboards und einem Terminal werden Zeichen an den Mikrokontroller geschickt und das Resultat vom Kontroller anschliessend wieder an den Benutzer zurück geschickt. Funktionen wie *put_char*() oder *get_char*() müssen hier implementiert werden. Damit wir die Serielle Schnittstelle in unserer Testumgebung emulieren können werden wir ein Socket benutzen.

2.4 Command Parser

Der Command Parser analysiert die einkommenden Befehle des Benutzerkeyboards und delegiert das Aufrufen der entsprechenden Methoden (siehe Commands).

Teil II

Spezifikation

3 Anforderungsspezifikation

3.1 Funktional

3.1.1 Muss

Muss Ziele sind von der Projektspezifikation gegeben und müssen erfüllt werden.

1. Nach einschalten der Hardware wird der Bootloader automatisch gestartet
2. Der Bootloader muss eine Applikation aufnehmen und diese permanent speichern
3. Der Bootloader muss eine permanente Applikation starten
4. Der Benutzer muss eine existierende Applikation entfernen
5. Binärdaten des Formats S-Record können behandelt werden
6. Der Bootloader hat keine Auswirkung auf die Funktionsweise der auszuführenden Applikation

3.1.2 Kann

1. Der Bootloader zeigt eine Auswahl an zu verfügung stehenden Applikation an
2. Der Bootloader started nach einer gewissen Zeit eine vom Benutzer als default markierte Applikation
3. Eine Applikation kann auf das Gerät geladen werden ohne es permanent zu speichern; Dies auch wenn der Speicherplatz des permanenten Specihers belegt ist. Dieses kann ausgeführt werden; Bei Neustart des Geräts ist dieses nicht mehr vorhanden. Nach nicht permanenten laden einer Applikation; ist der vorherige Zustand nach dem neustarten wieder hergestellt
4. Nach permanentem speichern einer Applikation und späterer Löschung ist der Zustand für den Anwender un- verändert
5. Der Speicherplatz wird insofern optimal genutzt, dass keine unnötigen, nicht verwendbaren Lücken enstehen;
6. Es kann eine default Applikation gewählt werden, welche nach einem gewissen Timeout automatisch gestartet wird
7. Eine default Applikation wird nicht mehr als 5 mal hintereinander autmatisch gestartet, falls dessen verarbeitung fehlerhaft war

3.2 Nicht-Funktional

1. Der Code sowie Kommentar sind in englisch gehalten
2. Der Source code muss dokumentiert sein
3. Ein Nutzer mit shell Kentnissen kommt mit der Benutzerschnittstelle zurecht. D. h., das Grundprinzip des Ein- gebens von verschiedenen Kommandos und deren Parameter, sowie die Benutzung des "Hilfe" Befehls sollte dem Benutzer vertraut sein

4 Befehle

Im Folgenden sind alle geplanten Befehle aufgelistet:

`reset`

Macht ein *reset* der gesamten Konfiguration.

`help`

Zeigt eine Liste aller möglichen Befehle an.

`help command-name`

Zeigt die Hilfe für `command-name` an.

`download [-b app-name [-d]]`

Lädt die folgenden Daten in den Arbeitsspeicher als eine SRecord Datei.

-b app-id	Führt den Befehl `burn app-id` aus.
-d	Setzt die Applikation als default Startanwendung

`burn app-name [-d]`

Speichert das sich im Arbeitsspeicher befindende SRecordFile in den Flashspeicher unter dem Namen `app-name`.

-d	Setzt die zu brennende Applikation als default Startanwendung

---

`run [-r|app-id]`

Führt die default, oder die mit dem `app-id` identifizierte Applikation aus.

-r	Führt das sich im Arbeitsspeicher befindende SRecordFile aus.

`list`

Führt eine Liste der sich im Flashspeicher befindenden Applikationen auf.

`set [--default=app-id] [--timeout=seconds] [...]`

Setzt die Werte der gegebenen Konfigurationsoptionen.

-d app-id	Siehe `--default-app`
--default=app-id	Setzt die default Applikation
-t seconds	Siehe `--default-app`
--timeout=seconds	Setzt die Wartezeit nach der die default Applikation gestartet werden soll
-e num	Siehe `--default-app`
--errcount=num	Anzahl Neustarts im Falle eines fehlerhafen Starts einer Applikation. Sobald errorcount erreicht ist, wird das Timeout deaktiviert und den Bootloader wird in Terminal-Mode umschalten.

`bootconfig`

Führt die aktuelle Konfiguration (app-name, timeout, errorcount) des Bootloaders auf.

`erase app-id`

Entfernt die mit dem `app-id` identifizierte Applikation aus der Applikationsliste des Flashspeichers.

Teil III

Implementation

5 Implementation auf Computer

5.1 Arbeitsvorgang

5.1.1 Entwicklung der Komponenten

Nach Aufteilung der verschiedenen Arbeiten, haben sich die einzelnen Projektmitglieder unabhängig voneinander den entsprechenden Komponenten gewidmet. Dies geschah, indem jeder sein eigenes Eclipseprojekt eröffnete und anschliessend die bereits vorgefertigten *.h* Files importierte um den entsprechenden C Code dazu zu schreiben.

5.1.2 Tests

Zu jeder implementierter Komponente ist ein Test geschrieben worden, der die verschiedenen Funktionalitäten der Komponente testet. Da wir den Umgang mit C-Unit Tests im Mikroprozessorkurs erst nach Beginn unserer Testphase erlernt haben, verzichteten wir auf einen Gebrauch von C-Unit. Getestet wurde, ob die jeweiligen Komponenten die korrekten Errorcodes -1, 0 oder 1 zurückgeben. Dazu wurde im Test die Komponente dementsprechend aufgerufen und überprüft, ob nach Aufruf der korrekte Errorcode zurückgegeben wird.

5.1.3 Zusammenführen des Codes

Sobald die einzelnen Komponenten fertiggestellt und mit den dafür vorgesehenen Tests getestet worden waren, haben wir den gesamten Code in ein Eclipseprojekt importiert. Da wir einzelne Komponenten nicht oder nur Teilweise implementieren konnten (siehe *Aufgetauchte Probleme*), haben wir nicht das gesamte Projekt mit Eclipse testen können.

5.2 Aufgetauchte Probleme

Im Verlaufe der Umsetzung haben wir bemerkt, dass unser Architekturkonzept zu generisch und zu Aufwändig geplant war, um alle vorgesehenen Komponenten in der vorgegebenen Zeit zu Ende zu führen. Die Umsetzung der verschiedenen Komponenten (Application Manager, Command Parser, Application Register us.w.) war viel Zeitaufwändiger als wir anfangs eingeplant haben. Der Grund für den grossen Zeitaufwand war, dass uns die Erfahrung mit der Programmiersprache C und der dazugehörigen Strategie der Makefiles fehlte. Ausserdem haben wir festgestellt, dass während der Umsetzung der Komponenten noch einzelne Änderungen in diversen *.h* Files vorgenommen werden mussten, was wiederum mit einem zeitlichen Mehraufwand verbunden war. Tabelle 1 zeigt eine Übersicht der implementierten Komponenten.

5.3 Übersicht der implementierten Komponenten

	Implementiert	**Getestet**
ApplicationManager		
ConfigurationRegistry		
SerialDriver		
SocketDriver		
CommandDispatcher		
CommandRegistry		
PersistentStorage		
VolatileStorage		
HDDStorage		
RAMStorage		
FlashStorage		
RAMVolatileStorage		
HDDVolatileStorage		
Main		

Tabelle 1: Übersicht der Implementation auf Computer

5.4 Implementierung auf MC68K

5.5 Arbeitsvorgang

5.5.1 Migrieren auf Zielplattform

Nach Entwickeln des Codes auf unseren persönlichen Laptops, haben wir den Code auf die Zielplattform MC68K geladen. Dazu haben wir den MC68K an eine Serielle Schnittstelle angeschlossen und mittels dem Programm CodeWarrior den Code auf die Zielplattform geladen.

5.5.2 Syntaxanpassung

Damit der Code auf der Zielplattform ausgeführt werden konnte, mussten wir die Syntax anpassen, damit sie dem ANSI C Standard entspricht. Im Folgenden sind die wichtigsten Änderungen aufgelistet:

- Variabeln mussten am Anfang der Funktion deklariert werden
- Die dezimalen Argumente der *printf()* Funktion mussten nach *long* gecastet werden
- Deklarationen wie $intresponse = 0;$ mussten in $response = 0;$ umgewandelt werden.
- $\#include < stdlib.h >$ musste überall wo davon Gebrauch gemacht wird eingefügt werden
- Funktion *strdup*() der Standardbibliothek *string.h* wurde vom MC68K nicht anerkannt.

5.6 Aufgetauchte Probleme

Zum einen waren die Syntaxanpassungen damit alle Warnungen von CodeWarrior eliminiert waren, relativ zeitaufwändig, zum andern haben wir ca. 4 Stunden verloren, da wir Anfangs mit einer Zielplattform gearbeitet haben, die einen Defekt aufwies. Als wir die Zielplattform gewechselt haben, konnten wir den Code zum ersten mal kompilieren. Da wir nicht alle Komponenten in der vorgegebenen Zeit implementieren konnten, haben wir nur einen Teil des Projektes auf der Zielplattform testen können.

5.7 Übersicht der implementierten Komponenten

	Implementiert	**Getestet**
ApplicationManager		
ConfigurationRegistry		
SerialDriver		
SocketDriver		
CommandDispatcher		
CommandRegistry		
PersistentStorage		
VolatileStorage		
HDDStorage		
RAMStorage		
FlashStorage		
RAMVolatileStorage		
HDDVolatileStorage		
Main		

Tabelle 2: Übersicht der Implementation auf MC68K

Teil IV

Schlussfolgerung

5.8 Team

Das produktive Arbeiten im Team entscheidet in jedem Projekt über Erfolg oder Misserfolg des Projektes. Das Projekt muss zuerst vom gesamten Team analysiert und verstanden werden bevor überhaupt etwas anderes gemacht werden kann. Alle Projektmitglieder müssen mit den getroffenen Entscheidungen einverstanden sein, damit zum nächsten Schritt im Projekt vorangegangen werden kann. Die Zusammenarbeit im Team und die Aufteilung der diversen Arbeiten war für uns eine bereichernde Herausforderung.

5.9 Erfahrung

Vor allem im Bereich der Informatik ist es ein sehr wichtiger Punkt, Erfahrung mit der Entwicklung von Softwareprodukten zu sammeln. Dieses Projekt gab uns die Möglichkeit, Erfahrung im Bereich der Projektkonzipierung, der Entwicklung mit der Programmiersprache C und Erfahrung im Umgang mit Mikrokontrollern zu sammeln.

5.10 Wir als künftige Ingenieure

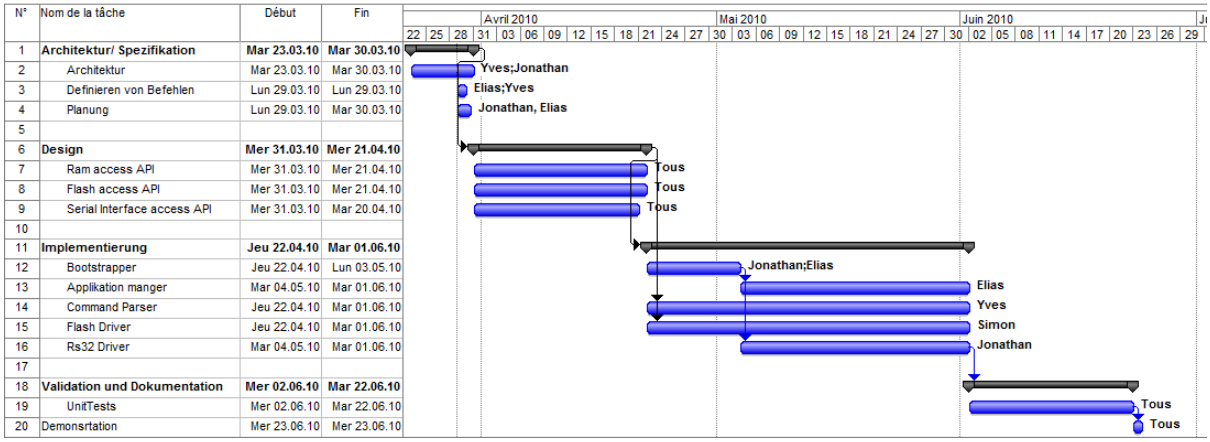
In unserer Zukunft als Ingenieure werden wir oft auf Informatikprobleme stossen, mit denen wir uns nicht auskennen. Das Bootloaderprojekt hat uns die Möglichkeit gegeben, ein unbekanntes Problem (Umsetzung eines Bootloaders) zu analysieren und zu lösen. Wir haben festgestellt, dass das Gesamte Problem in kleinere Teilprobleme im Team aufgeteilt werden kann. Dabei muss jedoch geachtet werden, dass die Kommunikation zwischen den Entwicklern der einzelnen Teilprobleme stets aufrecht erhalten bleibt. Dieses Projekt war für uns eine Wichtige Erfahrung, die wir in unserer Zukunft als Informatikingenieure gebrauchen werden.

Teil V

Anlagen

6 Planung

Abbildung 4 zeigt die Grossplanung für das Bootloader Projekt. Diese Planung wurde während der Entwicklung kontinuierlich aktualisiert.



N°	Nom de la tâche	Début	Fin
1	**Architektur/ Spezifikation**	**Mar 23.03.10**	**Mar 30.03.10**
2	Architektur	Mar 23.03.10	Mar 30.03.10
3	Definieren von Befehlen	Lun 29.03.10	Lun 29.03.10
4	Planung	Lun 29.03.10	Mar 30.03.10
5			
6	**Design**	**Mer 31.03.10**	**Mer 21.04.10**
7	Ram access API	Mer 31.03.10	Mer 21.04.10
8	Flash access API	Mer 31.03.10	Mer 21.04.10
9	Serial Interface access API	Mer 31.03.10	Mar 20.04.10
10			
11	**Implementierung**	**Jeu 22.04.10**	**Mar 01.06.10**
12	Bootstrapper	Jeu 22.04.10	Lun 03.05.10
13	Applikation manger	Mar 04.05.10	Mar 01.06.10
14	Command Parser	Jeu 22.04.10	Mar 01.06.10
15	Flash Driver	Jeu 22.04.10	Mar 01.06.10
16	Rs32 Driver	Mar 04.05.10	Mar 01.06.10
17			
18	**Validation und Dokumentation**	**Mer 02.06.10**	**Mar 22.06.10**
19	UnitTests	Mer 02.06.10	Mar 22.06.10
20	Demonsrtation	Mer 23.06.10	Mer 23.06.10

Abbildung 4: Diagramm der Planung

7 Entwicklungsprozess

Das ganze Projekt wird auf github (`http://github.com`) verwaltet.

- Git Repository auf github: `http://github.com/garetjax/Bootloader`

8 Ressourcen

8.1 Interne Ressourcen

- HTML Build der Dokumentation: `http://garetjax.github.com/Bootloader`
- HTML Build der Spezifikation: `http://garetjax.github.com/Bootloader/api`
- PDF Build der Dokumentation: `http://garetjax.github.com/Bootloader/static/documentation.pdf`
- PDF Build der Spezifikation: `http://garetjax.github.com/Bootloader/static/api.pdf`

8.2 Externe ressourcen

- Git: `http://git-scm.com`
- Sphinx: `http://sphinx.pocoo.org`
- Doxygen: `http://www.stack.nl/~dimitri/doxygen`