Operationssystem für den SIRIUS Softcore Processor

Florian Zowislok

Hochschule Offenburg, Badstraße 24, 77652 Offenburg

07821/205363, florian.zowislok@fh-offenburg.de

Zusammenfassung

Den Hauptbestandteil des Operationssystems stellt der Zugriff auf SD-Karten mit dem Dateisystem FAT16 von Microsoft dar. Für die Bedienung wurde ein Kommandozeileninterpreter implementiert. Als Ein- und Ausgabegerät dient ein PC mit einem speziellen Terminalprogramm, welcher über USB mit dem Emulationsboard des SIRIUS Softcores verbunden ist. Das System wird über die Eingabe von Befehlen am Terminal gesteuert.

Der SIRIUS Softcore kann nur vom Flash des Emulationsboards booten. Da das Betriebssystem selbst jedoch auf der SD-Karte gespeichert werden soll, ist ein Basis-Betriebssystem erforderlich, welches im Flash abgelegt ist. Das Basis-Betriebssystem lädt gleich nach dem Start das eigentliche Betriebssystem von der SD-Karte. Falls jedoch keine SD-Karte gesteckt ist, ermöglicht das Basis-Betriebssystem mit einem Kommandozeileninterpreter einige Grundfunktionen.

1. Grundlagen

1.1. SIRIUS Prozessor

Der SIRIUS Softcore ist ein an der Hochschule Offenburg entwickelter RISC Prozessor, welcher für die Programmierung in C optimiert wurde und sowohl im 16-Bit als auch im 32-Bit Modus betrieben werden kann [1]. Für den 32-Bit Modus werden dazu immer zwei 16-Bit Register zusammengeschaltet. Die aktuell mögliche Taktfrequenz liegt bei 48 MHz.

1.2. Emulationsboard

Auch das Emulationsboard *(Abbildung 1.1)* wurde vollständig an der Hochschule Offenburg entwickelt. Das Cyclone III FPGA von ALTERA bildet das Kernstück des Boards, in welchem der SIRIUS Prozessor emuliert wird. Im FPGA stehen zusätzlich 64 KB RAM und verschiedene Peripherien wie SPI, RS232 oder USB für den SIRIUS Prozessor zur Verfügung. Da die die 64 KB RAM mit 16 Bit adressiert werden können, wird der Prozessor im 16-Bit Modus betrieben. Des Weiteren befinden sich auf dem Board unter anderem LEDs, DIP-Schalter, ein Piezo-Lautsprecher, ein USB-Anschluss, ein RS232-Anschluß, ein Reset-Knopf, ein Flash-IC zum Booten des Systems sowie ein SD-Karten-Anschluss.

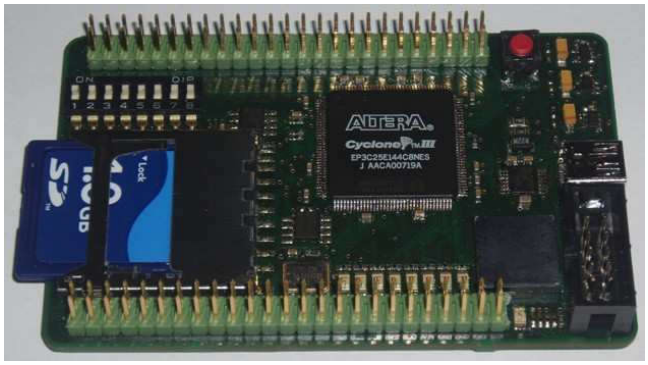

Abbildung 1.1: Emulationsboard mit SD-Karte

1.3. SIRIUS IDE

Die Entwicklungsumgebung SIRIUS IDE basiert auf der früheren FHOP-Umgebung und wurde an den SIRIUS Prozessor angepasst. Mit der IDE kann in der Programmiersprache C oder in Assembler programmiert werden. Der Compiler der SIRIUS IDE übersetzt den C-Code zuerst in Assembler-Code und anschließend mit dem CRASH Assembler ins Intel-Hex-File Format. Des Weiteren verfügt die IDE über einen Simulator, mit dem der fertig übersetzte Code getestet werden kann. Da im Simulator die SD-Karte nicht nachgebildet ist, können damit Programme, die auf die SD-Karte zugreifen, nicht getestet werden. *Abbildung 1.2* zeigt die SIRIUS IDE mit einem Testprogramm.

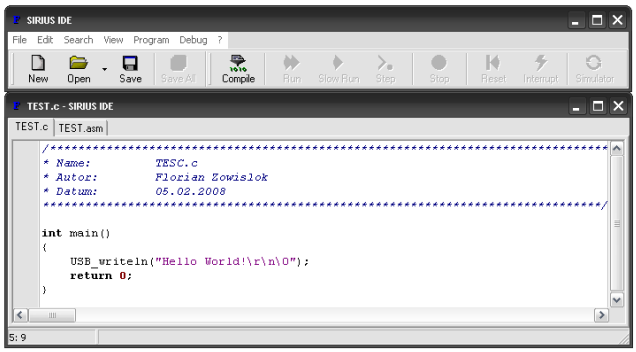

Abbildung 1.2 SIRIUS DIE

1.4. Dateisystem

Als Dateisystem wird FAT16 von Microsoft verwendet [2]. Zur Vereinfachung werden jedoch keine langen Dateinamen unterstützt. Es sind also nur acht Zeichen für einen Datei- bzw. Verzeichnisname möglich, wobei nur große Buchstaben, Ziffern und einige Sonderzeichen zulässig sind.

Beim Dateisystem FAT16 werden immer 512 Bytes zu einem Sektor zusammengefasst. Diese werden so eingeteilt, dass auf dem Datenträger verschiedene Bereiche zur Verfügung stehen, wie in *Abbildung 1.3* dargestellt. Im Datenbereich werden jeweils 2^n Sektoren zu einem Cluster zusammengefasst, wobei n so gewählt wird, dass jedes Cluster noch mit 16 Bit adressiert werden kann. Die Cluster werden fortlaufend durchnummeriert, wobei das erste Cluster mit der Nummer 2 (0x0002) beginnt. Die größte Clusternummer ist 65526 (0xFFF6), womit maximal 65525 Cluster möglich sind.

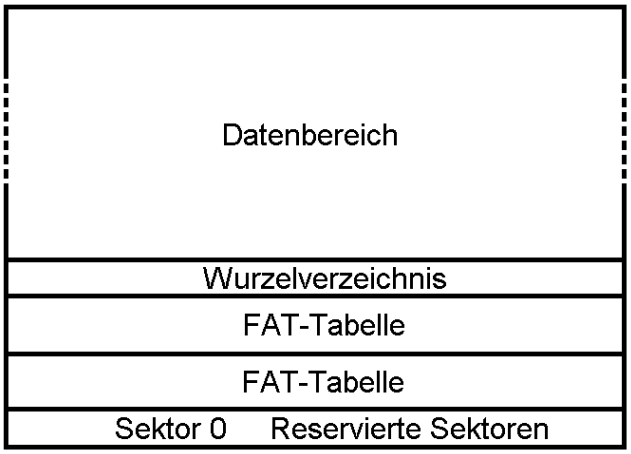

Abbildung 1.3 Datenträger mit FAT16

Im ersten Sektor des Datenträgers, dem Sektor 0, sind alle Daten hinterlegt, die benötigt werden, um auf das Dateisystem zuzugreifen. Anhand dieser Daten kann beispielsweise berechnet werden, an welcher Adresse ein bestimmter Bereich beginnt. Das Wurzelverzeichnis hat eine feste Größe, weshalb die Anzahl von Verzeichniseinträgen im Wurzelverzeichnis begrenzt ist. Jeder Datei wird ein 32-Byte Verzeichniseintrag (*Abbildung 1.4)* zugeordnet, wobei Unterverzeichnisse wie Dateien behandelt werden.

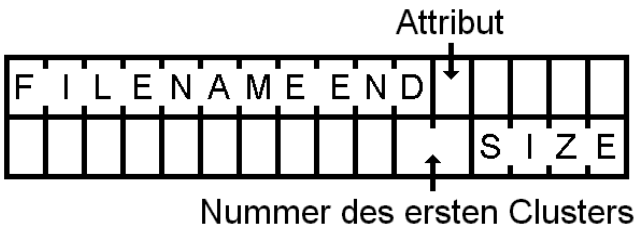

Abbildung 1.4: 32-Byte Verzeichniseintrag

In den ersten acht Bytes wird der Datei- bzw. der Verzeichnisname gespeichert, die Dateiendung in den folgenden drei Bytes, wobei diese bei einem Verzeichnis alle den Wert 0x20 enthalten, was einem Leerzeichen entspricht. Im Attribut-Byte stellen die einzelnen Bits verschiedene Eigenschaften dar, wie z.B. „versteckt“ oder „schreibgeschützt“. Besonders wichtig dabei ist das Bit, welches anzeigt, ob es sich um eine Datei oder um ein Verzeichnis handelt. In dem zwei Bytes großen Feld „Nummer des ersten Clusters“ wird angegeben, in welchem Datencluster die Datei bzw. das Verzeichnis beginnt. Die letzen vier Bytes enthalten die Größe der Datei in Bytes. Bei einem Verzeichnis enthält dieses Feld den Wert „0“. In den übrigen 14 Bytes werden verschiedene Zeitstempel gespeichert, auf die nicht näher eingegangen wird.

Die FAT-Tabelle wird meist doppelt auf dem Datenträger angelegt. Laut Spezifikation wären jedoch auch nur eine oder mehr als zwei möglich. Die FAT-Tabelle bildet alle Daten-Cluster ab, wobei jedem Cluster ein 16-Bit Wert zugeordnet wird:

0x0000:	Cluster ist frei
0x0002 bis 0xFFF6:	Clusternummer des nächsten Clusters
0xFFF7:	Cluster ist defekt
0xFFFF:	Letztes Cluster

Abbildung 1.5 zeigt, wie die Clusterverwaltung auf dem Datenträger funktioniert. In den Daten-Clustern befinden sich die tatsächlichen Dateien, wobei der Verzeichniseintrag einer Datei nur die Nummer des ersten Clusters enthält. Datei 1 beginnt mit Cluster A0. Der 16-Bit Wert der FAT-Tabelle gibt an, dass es das letzte Cluster der Datei ist. Datei 2 beginnt in Cluster A2. Der zugehörige Eintrag in der FAT-Tabelle gibt an, dass die Datei im Cluster A3 fortgesetzt wird, wobei der entsprechende Eintrag in der FAT-Tabelle wiederum die Nummer des nächsten Clustern angibt. Dieser Vorgang wird bis zum letzten Cluster wiederholt, welchem in der FAT-Tabelle der Wert 0xFFFF zugeordnet ist. Die Einträge der FAT-Tabelle ergeben somit eine einfach verkettete Liste von Clustern. Für die freien Cluster steht in der FAT-Tabelle der Wert 0x0000.

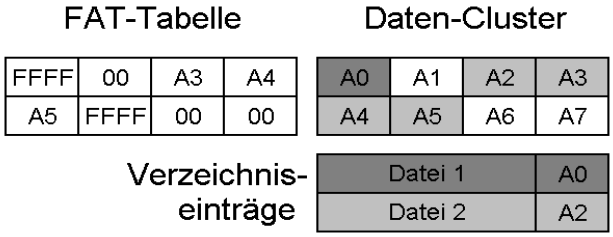

Abbildung 1.5: Clusterverwaltung

2. Betriebssystem

2.1. Kommandozeile

Das Betriebssystem besteht im Wesentlichen aus einem Kommandozeileninterpreter. Die Ein- und Ausgabe erfolgt über das Terminalprogramm USB-COM, welches auf einem Windows PC ausgeführt werden kann, der per USB mit dem Emulationsboard verbunden ist. Mit dem Bedienterminal können Zeichenketten in beide Richtungen übertragen werden. Ankommende Zeichenketten werden im oberen Textfeld ausgegeben, im unteren Textfeld können Befehle eingegeben werden, die erst mit Betätigen der Enter-Taste gesendet werden. Durch Betätigung der Schaltfläche „Clear Text“ wird das obere Textfeld geleert. Mit der Schaltfläche „Load HEX File“ kann eine komplette Intel-Hex-Datei vom PC zeilenweise an das Emulationsboard übertragen werden. *Abbildung 2.1* zeigt das USB-COM bei angeschlossenem Emulationsboard, nachdem das Betriebssystem gestartet wurde.

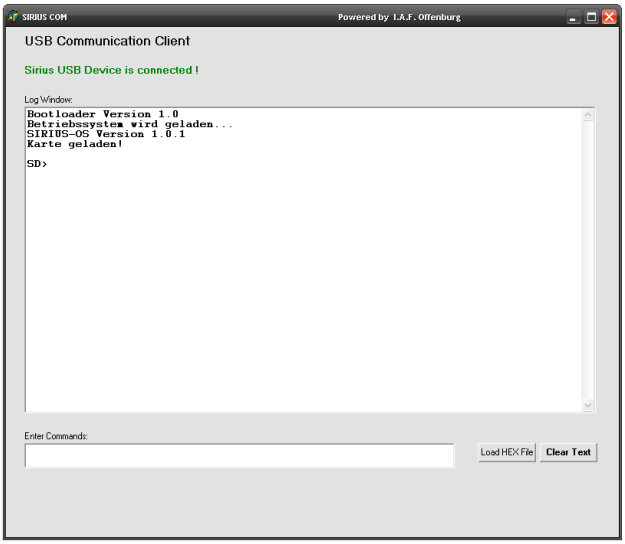

Abbildung 2.1: USB_COM

Beim Start wird zunächst die aktuelle Betriebssystemversion auf dem Terminal ausgegeben. Anschließend wird die SD-Karte initialisiert und die Dateisystemparameter geladen. Die Eingabeaufforderung erfolgt nun in einer Endlosschleife. Darin wird zuerst der aktuelle Pfad ausgegeben und dann auf eine Eingabe im Terminal gewartet. Nach der Eingabe der Befehlszeile wird nacheinander geprüft, ob die SD-Karte gesteckt ist, initialisiert wurde, und das Dateisystem FAT16 enthält. Sobald eine Bedingung nicht zutrifft, wird die Eingabe verworfen und die Endlosschleife von vorne durchlaufen. Zusätzlich wird eine Fehlermeldung ausgegeben bzw. die Karte neu initialisiert. Werden jedoch alle drei Bedingungen erfüllt, wird die Befehlszeile interpretiert.

2.2. Interpretation der Befehlszeile

Bevor die Befehlszeile interpretiert wird, werden alle Kleinbuchstaben in Großbuchstaben umgewandelt. Danach wird geprüft, ob die Eingabe mit einem vordefinierten Befehl übereinstimmt und die entsprechende Funktion aufgerufen. Falls die Eingabe mit keinem Befehl übereinstimmt, wird überprüft, ob sich auf der SD-Karte im aktuellen oder im Betriebssystem-Verzeichnis eine Hex-Datei mit entsprechendem Dateinamen befindet und gegebenenfalls ausgeführt. Beim Aufruf der jeweiligen Funktion wird ein Zeiger auf die Befehlszeile übergeben, der mit dem Wert „*X*“ um die Länge des Befehls korrigiert werden muss, damit er auf die Parameter des Befehls zeigt, wie in *Abbildung 2.2* dargestellt.

Abbildung 2.2 Zeiger auf Parameter

Folgende Befehle können in die Befehlszeile eingegeben werden:

DIR:	Zeigt den Inhalt des aktuellen Verzeichnisses
CD:	Wechselt das Verzeichnis
MD:	Erstellt ein Verzeichnis
RD:	Löscht ein Verzeichnis
DEL:	Löscht eine Datei
LOAD:	Lädt eine Datei vom Terminal auf die SD-Karte
RENAME:	Benennt eine Datei um
UPDATE:	Schreibt ein Programm von der SD-Karte in den Flash
LED OFF:	Schaltet die LED Anzeige des Timers aus
LED ON:	Schaltet die LED Anzeige des Timers ein

BELIEBIGE EINGABE:

Bei einer beliebigen Eingabe wird auf der SD Karte ein Hex-Programm mit genau diesem eingegebenen Dateinamen gesucht und ausgeführt. Dabei kann es sich um Anwendungen (*Kapitel 2.4*) oder um auf die SD Karte ausgelagerte Programm-Module handeln (*Kapitel 2.5*). Der Anwender merkt dabei keinen Unterschied gegenüber den direkt implementierten Befehlen.

2.3. Basis-Betriebssystem

Beim Einschalten des Emulationsboards kann zunächst nur der Flash gelesen werden. Da sich das eigentliche Betriebssystem jedoch auf der SD-Karte befindet, wird zunächst ein Basis-Betriebssystem aus dem Flash geladen, welches anschließend das eigentliche Betriebssystem von der SD-Karte lädt. Dieser Bootvorgang ist in *Kapitel 3* ausführlich beschrieben. Falls es jedoch nicht möglich ist das Betriebssystem von der SD-Karte zu laden, weil z.B. eine Datei oder auch die komplette SD-Karte nicht vorhanden sind, soll mit dem Basis-Betriebssystem eine gewisse Grundfunktionalität mittels Eingabeaufforderung möglich sein.

Grundlage des Basisbetriebssystems ist ein vorläufiger Entwicklungsstand des eigentlichen Betriebssystems, welcher entsprechend angepasst wurde. Einzelne Funktionen wurden komplett entfernt und stattdessen einige auf die SD-Karte ausgelagerte Programme direkt implementiert. Das Basisbetriebssystem enthält jedoch keine neuen Funktionen und wird deshalb nicht detaillierter beschrieben.

Das Basis-Betriebssystem verfügt ebenso wie das eigentliche Betriebssystem über eine Befehlszeile, die auch genauso funktioniert. Der Zugriff auf die SD-Karte ist jedoch nicht möglich. Stattdessen sind die Programm-Module aus *Kapitel 2.5* direkt implementiert und werden auch genauso aufgerufen. Ebenso ist das Ein- und Ausschalten der LED-Anzeige möglich. Folgende Befehle werden unterstützt:

HELP LHX RUN MEM EDT PTR PTW (*Kapitel 2.5*)

LED OFF LED ON (*Kapitel 2.2*)

2.4. Anwendungen

Mit der SIRIUS-IDE kann ein Benutzer eigene Anwendungen programmieren. Dazu wird bei der Kompilierung im Assembler-Code die Datei „BIOS.asm“ eingebunden. Diese Datei enthält jedoch nicht das BIOS, sondern Referenzen auf bestimmte BIOS-Funktionen bzw. globale Daten, da das eigentliche BIOS schon im Betriebssystem enthalten ist. Durch das Einbinden der Referenzen auf BIOS-Funktionen kann der Anwender die Funktionen verwenden, die schon im BIOS implementiert sind. Außerdem werden somit Funktionen gekapselt, auf die der Anwender keinen Zugriff haben soll. Zusätzlich ist angegeben, bei welcher Speicheradresse der Programmcode beginnen soll. Bei Anwendungen hat diese Startadresse den Wert 0x8000. Bei der Kompilierung ins Intel-Hex-Format wird der Programmcode festen Speicheradressen zugeordnet, beginnend mit der Startadresse. Es stehen maximal 28 KB RAM für die Anwendung zur Verfügung. Somit hat die höchste zulässige Adresse den Wert 0xEFFF.

Es gibt zwei Möglichkeiten eine Anwendung auszuführen:

1. Die Anwendungen werden mit dem Befehl „LHX“ über die USB-Schnittstelle vom PC in den RAM geladen. Mit dem Befehl „RUN 8000“ wird das Programm anschließend gestartet. Die Variante ist auch möglich, wenn nur das Basis-Betriebssystem geladen ist, in dem auf die SD-Karte nicht zugegriffen werden kann.
2. Die Anwendung ist auf der SD-Karte gespeichert und wird über den Dateinamen aufgerufen, wobei die Endung „HEX“ vorausgesetzt ist. Der Vorteil bei dieser Variante ist, dass über einen globalen String Parameter übergeben werden können. Falls sich die Anwendung nicht auf der SD-Karte befindet, kann sie vorher über die USB-Schnittstelle vom PC geladen und auf der SD-Karte gespeichert werden.

2.5. Programm-Module

Um den Speicherbedarf des Betriebssystems im RAM möglichst gering zu halten, werden einzelne Programm-Module auf die SD-Karte ausgelagert. Damit diese Module unabhängig vom aktuellen Pfad verwendet werden können, werden sie als Intel-Hex-Datei im Betriebssystemverzeichnis „OS“ gespeichert. Der Aufruf erfolgt wie bei Anwendungen über den jeweiligen Dateinamen, wobei die Endung .HEX vorausgesetzt wird. Grundsätzlich werden die Programm-Module genauso wie Anwendungen erstellt. Damit es jedoch keinen Speicherkonflikt mit den Anwendungen gibt, wird ein anderer Speicherbereich verwendet. Die Startadresse hat hier den Wert 0x7000, wobei die einzelnen Module nicht mehr als 4 KB Speicher benötigen dürfen, da bei Adresse 0x8000 der Speicherbereich der Anwendungen beginnt. Es gibt insgesamt sieben ausgelagerte Programm-Module:

HELP.HEX:	Ausgabe unterstützter Befehle mit Erläuterung
LHX.HEX:	Laden einen Hex-Datei vom PC in den RAM
RUN.HEX:	Ausführen von Programmcode an einer speziellen Adresse
MEM.HEX:	Ausgabe des RAM-Speichers bei einer bestimmten Adresse
EDT.HEX:	Ändern des RAM-Speichers bei einer bestimmten Adresse
PTR.HEX:	Direktes Lesen eines Ports
PTW.HEX:	Direktes Schreiben eines Ports

2.6. Schreiben auf die SD-Karte

Im Betriebssystem sind bereits Routinen zum Schreiben von Dateien auf die SD-Karte implementiert, die beim Laden eines Programms auf die SD-Karte mit dem Befehl „LOAD“ verwendet werden. Diese Routinen können auch von Anwendungsprogrammen verwendet werden. Dabei wird zuerst eine bereits vorhandene oder neu erstellte Datei geöffnet. Anschließend kann die Datei immer blockweise zwischen einem und 512 Bytes verlängert werden. Die bereits vorhandenen Bytes können dabei nicht überschrieben werden. Die neuen Bytes werden jeweils am Ende der Datei angehängt. Nach dem Schreibvorgang muss die Datei wieder geschlossen werden.

3. Bootvorgang

3.1. Laden des Basisbetriebssystems

Beim Einschalten des Emulationsboard muss das System zunächst gebootet werden. Dazu befindet sich im FPGA ein kleiner ROM mit welchem die ersten 2 KB des Flashs in den untersten Bereich des RAMs kopiert werden (Adresse 0x0000 bis 0x07FF). Darin ist das komplette BIOS enthalten. Anschließend wird ein Sprung an Adresse 0x0780 im RAM durchgeführt. Der hier folgende Code lädt das restliche Programm aus dem Flash, das Basis-Betriebssystem. Dies ist in *Abbildung 3.1* dargestellt. Für das Basis-Betriebssystem ist eine maximale Größe von 28 KB vorgeschrieben. Mit einem Sprung an Adresse 0x0800 wird es gestartet.

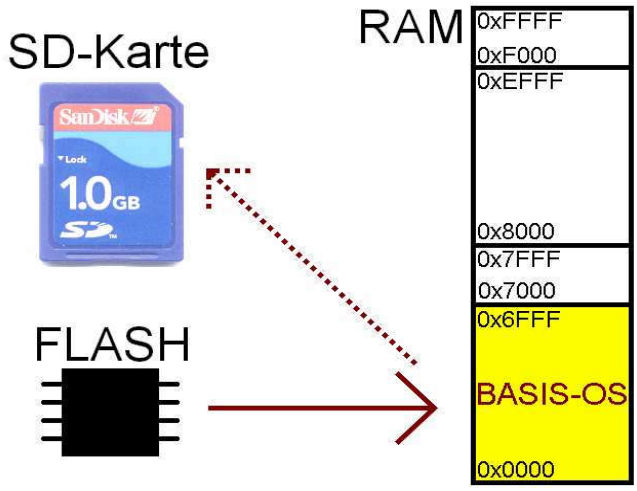

Abbildung 3.1: Laden des Basisbetriebssystems

3.2. Laden des Betriebssystems

Das Basis-Betriebssystem versucht zuerst auf die SD-Karte zuzugreifen (*Abbildung 3.1*). Wenn dies nicht möglich ist, wird der Vorgang abgebrochen und dem Anwender steht nur das Basis-Betriebssystem zur Verfügung. Gelingt der Zugriff jedoch, werden zwei Hex-Dateien aus dem Verzeichnis „OS“ in den RAM geladen, zuerst das Betriebssystem „SD.HEX“ und anschließend die Datei „LOADER.HEX“. *Abbildung 3.2* zeigt diesen Vorgang mit der entsprechenden Speichereinteilung.

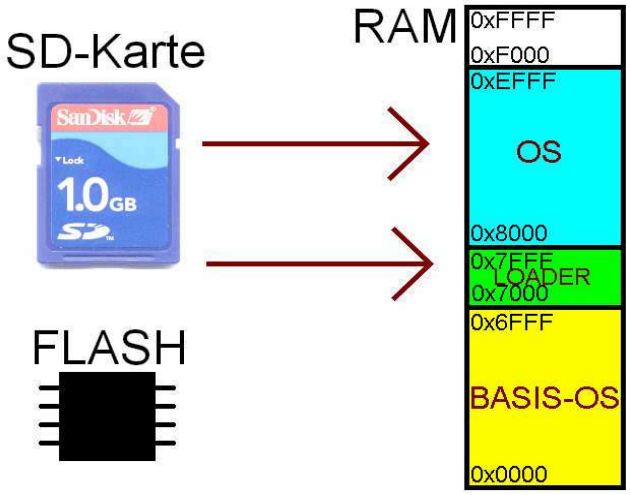

Abbildung 3.2: Laden von der SD-Karte

3.3. Funktion des Loaders

Nach dem Laden in den RAM wird der Loader ausgeführt. Der Loader ist ein eigenständiges Programm und greift nicht auf die BIOS-Routinen im unteren Speicherbereich zu. Er überschreibt das Basis-Betriebssystem mit dem Betriebssystem, das von der SD-Karte geladen wurde, wie in *Abbildung 3.3* dargestellt. Anschließend wird ein Sprung an Adresse 0x0780 durchgeführt, womit die Hardware neu initialisiert und das geladene Betriebssystem gestartet wird.

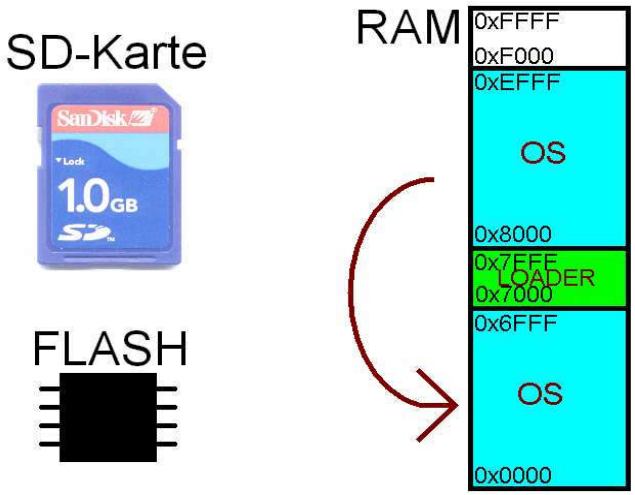

Abbildung 3.3: Überschreiben des Basis-OS

4. Ausblick

Bei dem erstellten Betriebssystem bieten sich viele Erweiterungsmöglichkeiten, die in zukünftigen Projekten implementiert werden könnten. Im Folgenden sind einige Bespiele genannt:

- CRC-Prüfung bei der Kommunikation mit der SD-Karte
- Fehlerbehandlung
- Kopieren und verschieben von Dateien und Verzeichnissen
- Ausführen von Batch-Dateien für automatische Abläufe
- Ausgabe von verschiedenen Dateitypen auf dem Terminal
- Erstellen eines Schedulers, mit dem mehrere Prozesse gleichzeitig ausgeführt werden können (Multitasking)

5. Quellen

[1] Dirk Jansen, Nidal Fawaz, Daniel Bau, Marc Durrenberger:

"A Small High Performance Microprocessor Core SIRIUS for Embedded Low Power Designs, Demonstrated in a Medial Mass Application of an Electronic Pill (ePille®)"

Embedded Systems Design Topic, Techniques and Trend, p. 363-372, June 2007, California, USA

ISBN 978-0-387-72257-3

[2] Microsoft FAT32 File System Specification Version 1.03

http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx