Der Cortex Microcontroller Software Interface Standard

- Ideen, Entwurfsziele, Bereiche der Standardisierung, Aufbau -

Ralf Kendl
Fachbereich MNI
Hauptseminar Sommersemester 2011
Software für technische Systeme
Institut für Technik und Informatik

Inhaltsverzeichnis

1 Vorwort

In der folgenden Ausarbeitung gebe ich eine Einführung in den Cortex Microcontroller Software Interface Standard. Das Auseinandersetzen mit diesem Thema erfolgte im Rahmen des Hauptseminars Software für technische Systeme am Institut für Technik und Informatik im Sommersemester 2011 an der Technischen Hochschule Mittelhessen. Der Text ist für Studenten der Informatik oder vergleichbarer Studiengänge mit fortgeschrittenem Kenntnisstand geschrieben. Es werden keine Grundlagen zu Mikrocontrollertechnik und Softwareentwicklung vermittelt, sondern als bekannt vorausgesetzt. Zum Einstieg in die Thematik beginne ich im Kapitel Embedded Software damit, die besondere Situation bei der Erstellung von Software für eingebettete Systeme zu erläutern. Danach folgt das Kapitel rund um den Cortex Microcontroller Standard, welchen ich nachfolgend mit CMSIS abkürzen werde. Da dieser Standard für eine spezielle Reihe von Mikroprozessoren geschrieben wurde, gehe ich im Kapitel Cortex Mikroprozessor auf diese Prozessoren gesondert ein. Ziel dieser Ausarbeitung ist es, ein Verständnis für die Ideen, den Aufbau und die Ziele von CMSIS zu entwickeln.

2 Embedded Software

2.1 Einleitung

Unter dem Begriff Embedded Software versteht man die Verbindung von Embedded Systems und Software Engineering, also die Erstellung von Software auf eingebetteten Systemen. Bei dieser Form der Softwareentwicklung existieren einige Unterschiede zum üblichen Entwicklungsprozess. Die zu erstellende Software wird in der Regel auf einem System entwickelt, welches nicht das Zielsystem darstellt. Der Grund dafür ist einfach, denn die Leistungsfähigkeit von eingebetteten Systemen ist eingeschränkter und daher wäre eine Entwicklung auf dem Zielsystem mit mehr Zeit und somit auch Kosten verbunden. Deshalb nutzt man die Technik des Cross-Compilings, um den geschriebenen Source Code vom Hostsystem auf dem Zielsystem lauffähig zu bekommen. Weiterhin ist die Portierung von Source Code von einem Zielsystem auf ein Anderes problematisch, da die Hardware Architektur (Register, Befehlssatz, usw.) unterschiedlich sein kann. In diesem Fall sind Anpassungen der Software an die neue Umgebung notwendig. Ein solcher Prozess ist durchaus nicht unüblich. Vorstellbare Szenarien sind zum Beispiel: Die Hardware des eingesetzten Zielsystems ist veraltet und wird vielleicht nicht mehr hergestellt. Weiterhin kann eine Erweiterung des Funktionsumfangs der Software dazu führen, dass die Leistungsfähigkeit der Hardware nicht mehr ausreicht.

Es existieren noch viele weitere Aspekte, die beim Erstellen von Embedded Software zu beachten sind. Für das Verständnis von CMSIS reichen die oben Vorgestellten jedoch aus. Weitere Betrachtungen entfallen deshalb an dieser Stelle.

2.2 Allgemeiner Workflow

Ich möchte nun einen Workflow aus Sicht eines Entwicklers für Embedded Software analysieren. In Abbildung 2.1 ist dieser Ablauf graphisch veranschaulicht.

Das allgemeine Vorgehen könnte man folgendermaßen beschreiben: Zuerst muss das Zielsystem und die Aufgabenstellung bekannt sein. Danach erfolgt die Einarbeitung in das Zielsystem. Eine übliche Tätigkeit ist hierbei das Studieren des User Guides. Nach erfolgtem Einarbeiten entwickelt man die gewünschte Software und transferiert sie auf das Zielsystem. Möchte man nun das erstellte Programm oder ein Modul daraus auf einem weiteren Zielsystem lauffähig machen, dann erfolgt wieder eine Einarbeitung und darauf die notwendigen Änderungen am Source Code. Im gleichen Schema würden weitere Portierungen erfolgen.

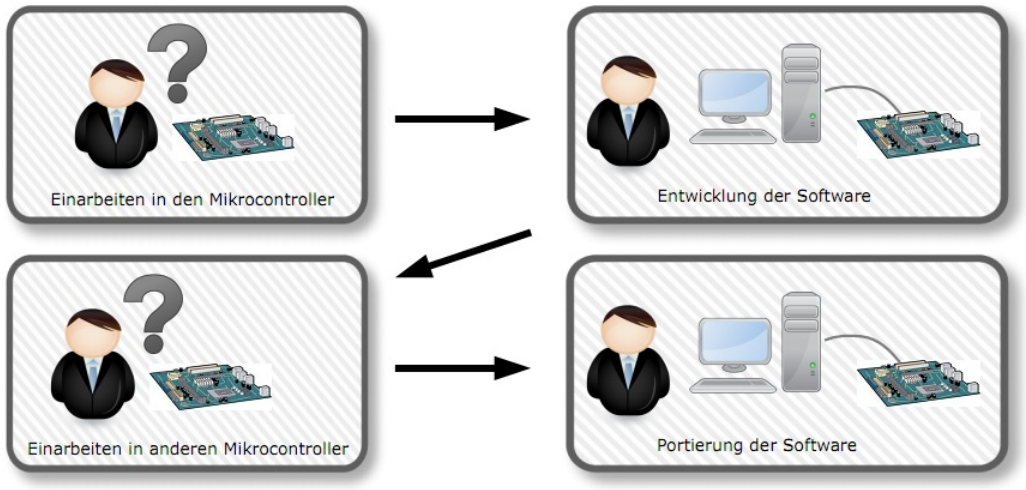


Abbildung 2.1: Workflow des Entwicklungsprozesses

2.2.1 Bewertung

Diese Vorgehensweise ist für viele Entwicklungsprozesse von Vorteil. Durch die intensive Auseinandersetzung mit der Hardware besitzt der Softwareentwickler ein hohes Maß an Fachwissen. Die Individualität der Hardware führt zur Individualität der Software und somit zu einer gezielten, lösungsorientierten Programmierung. Diese Faktoren führen im Ergebnis zu einer perfomanten Software. Die Qualität des Produkts und die Bereicherung an Wissen sind daher attraktive Gründe zum Einsatz dieses Modells in der Lehre. Den genannten Vorteilen stehen aber auch einige Nachteile gegenüber. Für einen Anfänger stellt sich ein hoher Anspruch, denn er wird mit sehr viel Material konfrontiert, welches er beherrschen muss. Betriebswirtschaftlich stellt sich die Frage der Kosten, denn der Zeitverbrauch durch Einarbeitungen in Hardware ist hoch. Betrachtet man die Grundsätze von gutem Softwaredesign, so stellt man schnell fest, dass die Wiederverwendbarkeit der Software nicht gegeben ist und darüber hinaus ein weiterer Kostenfaktor für erhöhten Implementierungsaufwand entsteht. Als ein weiterer Nachteil stellt sich die Frage nach der Konkurrenzfähigkeit beim Entwickeln in dieser Vorgehensweise, denn eine schnelle Bereitstellung der Software ist nur schwerlich realisierbar. Dies aber bleibt ein wichtiges Kriterium, da der Markt im Embedded Systems Bereich stark wächst und ein kurzer „Time to Market“ einen hohen Erfolgsfaktor darstellt.

3 Cortex Microcontroller Software Interface Standard

3.1 Einleitung

Mit der Einführung des CMSIS versucht man den im vorherigen Kapitel besprochenen Nachteilen beim Entwicklungsprozess zu begegnen. Die Idee dabei ist, den Softwareentwickler so weit wie möglich von der Hardware zu entkoppeln. Das realisiert man durch die Einbindung einer Hardwareabstraktionsschicht. Sie fungiert als Schnittstelle zwischen Hardware und Software. Konkret wird bei CMSIS eine Treiberbibliothek zur Verfügung gestellt, über die man mittels komfortablen Funktionsaufrufen die Hardware ansprechen kann. Somit entfällt für den Softwareentwickler die Notwendigkeit sich mit der Hardware auseinandersetzen zu müssen, da er jetzt nur einmal die Bibliothek studieren muss und mit diesem Wissen sein Programm für mehrere Plattformen schreiben kann. Diese Unterstützung wird von der Firma ARM gestellt und ist für die Modellreihe der Cortex M Mikroprozessoren geschrieben. Wie der Einsatz von CMSIS aussieht wird in diesem Kapitel genauer erläutert.

3.2 Bereiche der Standardisierung

Um mit der Schnittstelle sinnvoll arbeiten zu können, sollte man wissen, was alles von dem Standard abgedeckt wird. Dadurch erhält man einen abstrakten Einblick in den Funktionsumfang der Bibliothek. Die Betrachtung der Standardisierung erfolgt auf Basis des Versionsstandes 2.0. Folgende Standards sind gegeben:

- Hardware Abstraction Layer: hier findet man alle Registerdefinitionen für Core, NVIC, MPU und diverse Funktionen zum Zugriff auf die Hardware
- Die Namen der System Exceptions
- Die Organisation von Header Files
- Die Systeminitialisierung
- Intrinsische Funktionen: alle prozessorspezifischen Befehle werden in C-Funktionen gekapselt
- Funktionen zur Kommunikation z.B. via Ethernet
- Zugriff auf die Systemfrequenz

3.3 Zugriff auf die Hardware

Durch das Einbinden einer Schnittstelle besteht also keine Notwendigkeit zum direkten Hardwarezugriff mehr. Das bedeutet, dass jegliche Software, sei es Anwendungssoftware, das Betriebssystem oder auch Middleware eine Zugriffsmöglichkeit über die CMSIS Schnittstelle bekommen muss. Ein schematisches Zugriffsmodell ist in Abbildung 3.1 zu sehen.

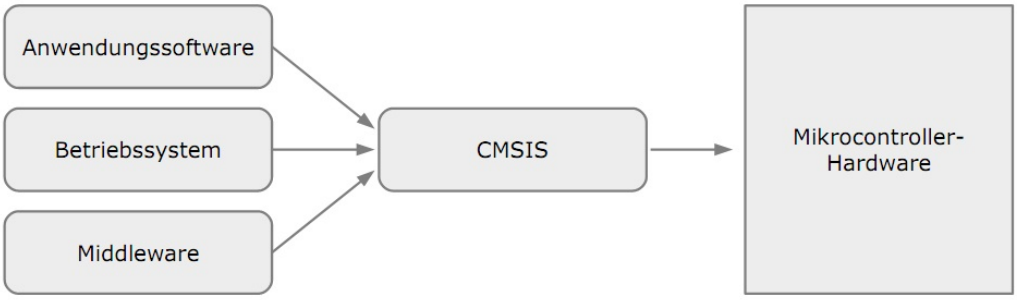


Abbildung 3.1: Zugriff auf Hardware mit CMSIS

3.4 Schichtenmodell

Eine genauere Betrachtung des Zugriffs erlaubt uns das Schichtenmodell in Abbildung 3.2. Es zeigt die unterschiedlichen angebotenen Access-Layer, die je nach Anwendungsfall von den jeweiligen Komponenten benutzt werden.

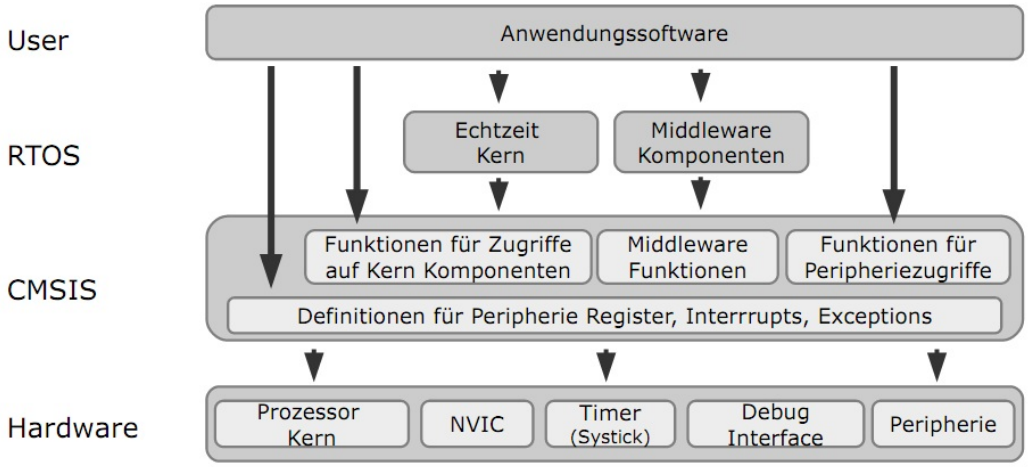


Abbildung 3.2: Schichtenmodell

3.5 Workflow mit CMSIS

Die Veränderungen des Arbeitsablaufs zum im Kapitel 1 vorgestellten Workflow sind signifikant. Der Softwareentwickler muss sich zuerst mit der Schnittstelle und somit hauptsächlich mit der Bibliothek von CMSIS auseinandersetzen. Dies ist leichter für ihn, weil er aus seiner Qualifikation heraus in der Regel ein umfassenderes Verständnis von Software besitzt als von Hardware. Hinzu kommt, dass sich viele Funktionen aus der Bezeichnung erschließen und damit den Lernprozess unterstützen. Dies hilft gerade Anfängern sich besser einzuarbeiten. Danach kann das Entwickeln der Software unter Einbeziehung der Schnittstelle erfolgen. Eine Portierung des Programms auf eine neue Plattform würde nur das neue Einbinden einer gerätespezifischen Datei bedeuten, was konkret die Veränderung von einer Zeile des Codes bedeutet. In Abbildung 3.3 wird der Workflow graphisch veranschaulicht.

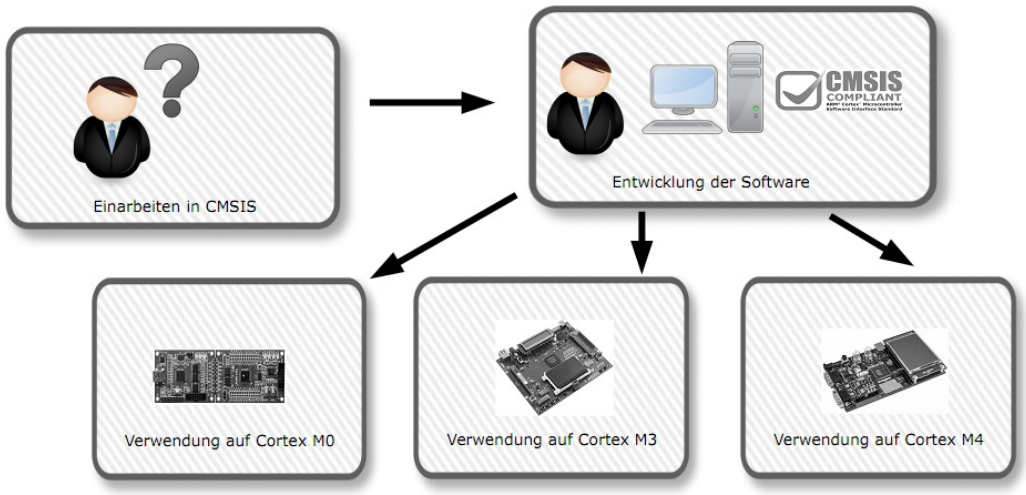


Abbildung 3.3: Entwicklungsprozess mit CMSIS

3.5.1 Bewertung

Das Potential von CMSIS zur Verbesserung des Entwicklungsprozesses ist enorm hoch. Die Einsteigerfreundlichkeit ist durch die Konzentration auf die Schnittstelle deutlich gestiegen. Eine Wiederverwendung von Code wird jetzt möglich und somit lassen sich Portierungen schneller und kostengünstiger realisieren. Durch das Wegfallen von Einarbeitungzeit für weitere Mikrocontrollerarchitekturen sinkt ebenfalls der Kostenfaktor und die Effizienz steigt. Diese Punkte summieren sich außerdem zu einem beschleunigten „Time to Market“, der wiederum die Konkurrenzfähigkeit in dem Marktsegment steigert. Jedoch sollten vor dem Einsatz von CMSIS auch die Nachteile bedacht werden. Man nimmt eine Fixierung auf die Produkte der Cortex M Reihe in Kauf. Dies schränkt die Wahl der möglichen Controller erheblich ein. Weiterhin erkauft man sich viele Vorteile durch die Abgabe von Wissen an die Bibliothek. In wie weit man das vertreten kann,

muss sicherlich von Fall zu Fall unterschiedlich bewertet werden. Auch die Beschränkung der Programmiersprache sollte beachtet werden, denn die Verfügbarkeit reduziert sich auf die Sprache C. Ebenfalls gilt es die Wahl des Compilers zu überprüfen, denn es wird nur eine geringe Zahl an Compilern unterstützt.

3.6 Bibliothek

Die Bibliothek besitzt einen normierten Aufbau, der sich in einen herstellerunabhängigen und einen herstellerabhängigen Teil untergliedert. Jeder Hersteller stellt eine zentrale Header Datei zur Verfügung, die in das Software Projekt eingebunden wird. Von dieser Datei gehen dann alle weiteren Verknüpfungen in Richtung der Bibliothek aus. In Abbildung 3.4 wird der Aufbau der Bibliothek dargestellt.

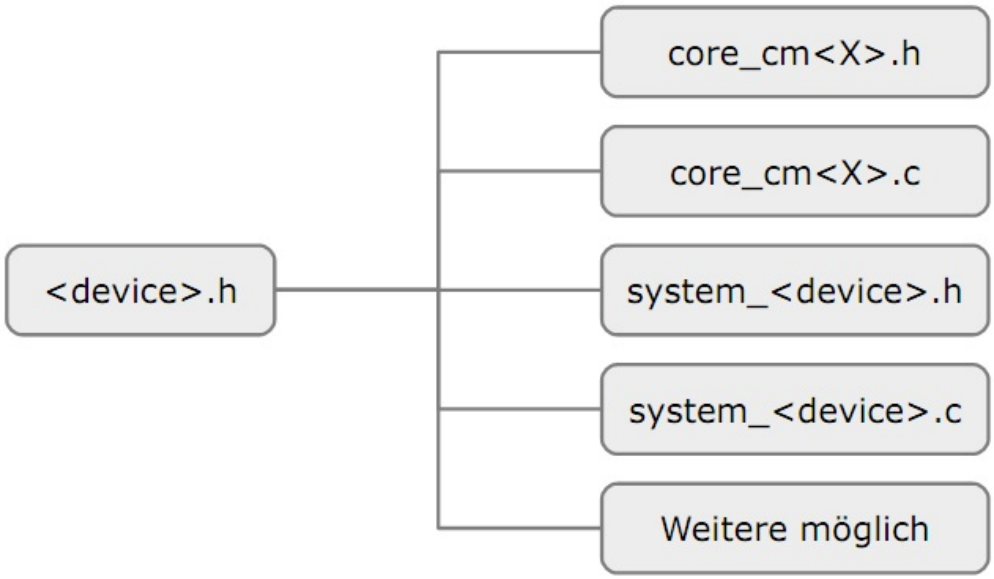


Abbildung 3.4: Struktur der Bibliothek

3.6.1 Herstellerunabhängige Dateien

Für jeden Cortex M Prozessor bietet die Bibliothek eine core_cm Header und C-Source Datei an. Die Header Datei beinhaltet Register Definitionen und Zugangsfunktionen für die Kern Komponenten wie zum Beispiel NVIC und Systick. Weiterhin findet man dort die Deklarationen der intrinsischen Funktionen. Zusätzlich enthält der Header beim Cortex M3 und M4 eine Funktion zur Ausgabe von Debug Nachrichten mittels dem Instrumentation Trace Module. Die C-Source Datei enthält die Implementierungen der intrinsischen Funktionen, die nicht durch Deklarationen in der Header Datei abgebildet werden können.

3.6.2 Herstellerabhängige Dateien

In der system_device Header Datei werden alle spezifischen Interrupt Nummern und Register definiert. In der entsprechenden C-Source Datei findet man die Funktion SystemInit, die eine vollständige Systeminitialisierung des Mikrocontrollers bewirkt. Die zentrale Header Datei device.h beherbergt dann alle Charakteristiken der entsprechenden Controller Hardware, wie die Interrupt und Exception Nummern, die Konfguration des Cortex Prozessors und zusätzliche Hilfsfunktionen für die Peripheriegeräte. In dieser Datei können auch Verweise zu weiteren, herstellerspezifischen Bibliotheken stehen, falls diese benötigt werden.

3.7 Paket

Die Bibliothek von CMSIS ist frei im Internet erhältlich und um effektiv damit umgehen zu können, sollte man sich im Paket orientieren können. Nach Entpacken findet man die Verzeichnisübersicht, wie sie in Abbildung 3.5 zu sehen ist.

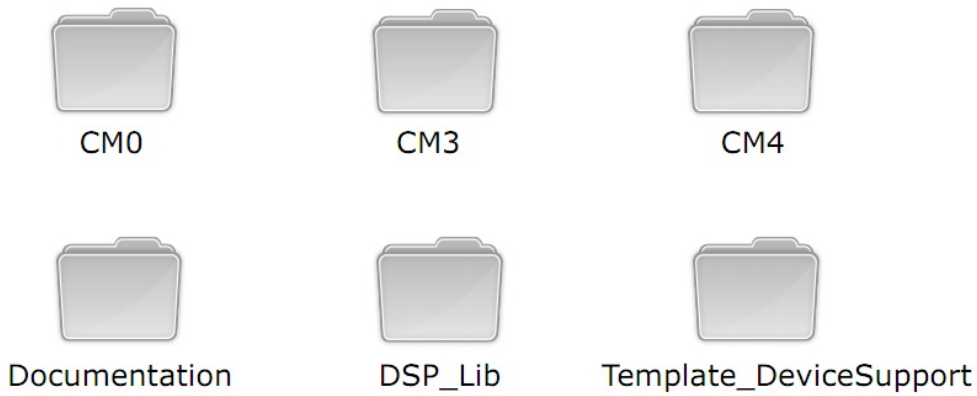


Abbildung 3.5: Das CMSIS Paket

Wie man erkennen kann, gibt es für jeden unterstützten Cortex Prozessor einen eigenen Ordner. Hier finden sich die entsprechenden herstellerunabhängigen Dateien mit ihrer individuellen Konfiguration, wie sie im vorherigen Abschnitt besprochen wurden. Weiterhin befindet sich dort im Unterverzeichnis DeviceSupport von der Firma ARM gefertigte Schablonen, die den Herstellern der Mikrocontroller verdeutlichen sollen, wie die Mindestanforderungen bei der herstellerspezifischen Implementierung aussehen.

Im Ordner Documentation befindet sich die SIMD *SingleInstructionMultipleData* Befehlsübersicht. Dieser Befehlssatz erhöht die Leistungsfähigkeit des Cortex bei vielen Instruktionen. Des Weiteren findet man im Ordner Documentation Informationen über die verwendeten Kodierrichtlinien und wie man diese überprüfen kann. Auf diesen Punkt werde ich in einem späteren Abschnitt nochmals eingehen. Die Dokumentation liefert darüber hinaus noch eine Beschreibung aller Dateien, eine Funktionsübersicht, zahlreiche Beispielprogramme, Hilfestellung zu den Debug Funktionalitäten und eine Erklärung zur

System View Description. Diese beinhaltet Informationen über die Peripherie in einer XML Struktur.

Der Ordner DSP_Lib umfasst eine Sammlung von 61 Funktionen für die digitale Signalverarbeitung. Hier finden sich zum Beispiel Hoch- und Tiefpassfilter, Funktionen zur Motorsteuerung, komplexe Arithmetik oder Transformationen. Zur leichteren Benutzung gibt es viele Beispiele. Die DSP Funktionalitäten sind momentan nur für den Cortex M4 und den Cortex M3 verfügbar. Beim Cortex M3 muss dazu aber der nicht vorhandene Hardware Teil mit Software emuliert werden. Eine Unterstützung für den Cortex M0 wird aber in Aussicht gestellt und sollte in den nächsten Versionen von CMSIS zu finden sein.

3.8 Beispiel

Ein kurzes Code Beispiel in Abbildung 3.6 soll die einfache Benutzung der CMSIS Funktionen verdeutlichen.

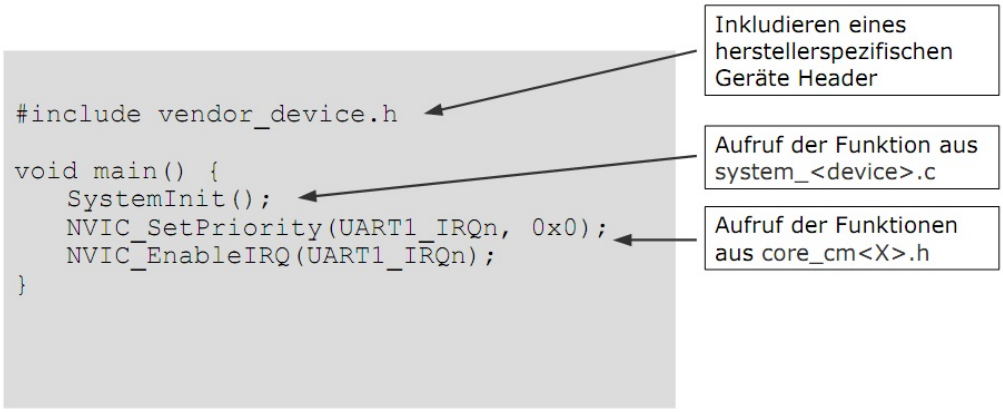


Abbildung 3.6: Beispielcode

Es ist leicht zu erkennen, dass nach dem Aufruf des Main Blocks die Funktion SystemInit eine vollständige Initialisierung des Systems vornimmt und der Controller nun einsatzbereit ist. Danach wird die Priorität eines Interrupts eingestellt und im Anschluss daran wird dieser aktiviert. Die sinnvolle und sprechende Namensbelegung der Funktionen sorgen für ein rasches Verständnis der Materie und somit für einen transparenten Code.

3.9 MISRA-C 2004

MISRA-C ist ein Programmierstandard für die Sprache C, der von der Motor Industry Software Reliability Association erschaffen wurde. Er wurde anfangs für den Einsatz in der Automobilindustrie entwickelt. Heute erfreut er sich großer Popularität und

stellt einen weit verbreiteten Standard für alle kritischen Systeme dar. Mit über 100 Programmierregeln zur Vermeidung von Laufzeitfehlern, struktuellen Schwächen und Sicherstellung der Gültigkeit von Ausdrücken ist der MISRA Standard eine mächtige Entwicklungsrichtlinie. Die CMSIS Bibliothek nutzt diesen Standard konsequent und das Einhalten dieser Richtlinien bei der Anpasssung der herstellerspezifischen Dateien kann mit dem Software Tool Lint geprüft werden. MISRA-C gilt als ein wichtiges Qualitätsmerkmal bei der Beurteilung des CMSIS Codes.

4 Cortex Mikroprozessor

4.1 Einleitung

Da man mit CMSIS nur eine spezielle Reihe von Mikroprozessoren benutzen kann, darf ein Kapitel zur grundsätzlichen Erläuterung dieser Chips nicht fehlen. Ich halte es nicht für angebracht hier eine elektrotechnische Abhandlung über die genauen Spezifikationen zu schreiben, sondern möchte nur eine kleine Einführung geben, die den Leser danach ermächtigen soll, Cortex M Prozessoren einzustufen und die jeweiligen Unterschiede in der Modellreihe differenzieren zu können. In der Folge kann eine einfache Entscheidung bei der Wahl des Prozessors anhand des Anwendungsgebietes getroffen werden.

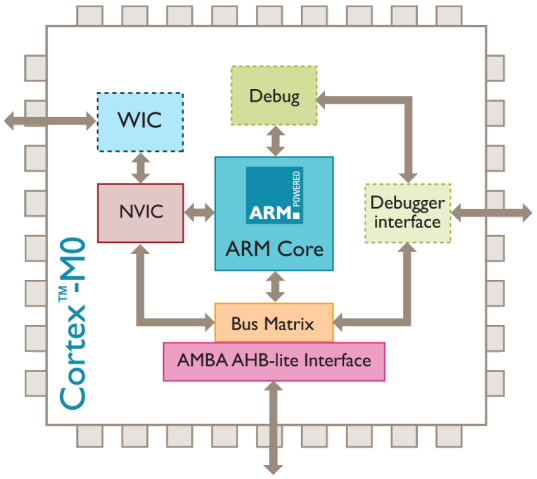
4.2 Allgemeines

Der Hersteller der Cortex Prozessoren ist die Firma ARM Ltd. mit Sitz in England. ARM ist ein fabrikloses Unternehmen, das heißt es entwickelt das Design von Prozessoren und verkauft die Intellectual Property über den Lizenzweg an Herstellerfirmen. Die Cortex Prozessorenreihe ist eine 32 Bit RISC-Architektur und differenziert sich in drei Produktkategorien:

- A : Application, für High-End Systeme mit Betriebsystem
- R : Realtime, für Echtzeitsysteme
- M : Microcontroller, für energieoptimierte Systeme

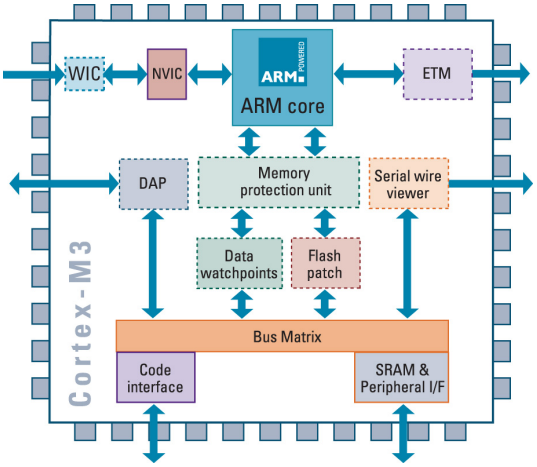
Für CMSIS ist nur die Produktreihe M von Belang, die Modelle aus A und R sind nicht zur Bibliothek kompatibel. Der Cortex M1 wird hier ebenfalls nicht weiter diskutiert, da er nur eine FPGA optimierte Version des Cortex M0 ist.

4.3 Cortex M0



Der M0 ist der kleinste Prozessor der Modellreihe und bietet auf einer sehr kleinen Chipfläche mit unter 12.000 Gattern eine beachtliche Leistung. Möglich macht dies der neue Befehlssatz Thumb2 von ARM, der beim M0 aber nur zum Teil implementiert wurde. Das macht Maschinencode vom M0 vollständig aufwärtskompatibel zu den größeren Modellen. Ein weiteres Merkmal des M0 stellt der geringe Stromverbrauch von 85 Mikrowatt pro MHz Taktfrequenz dar. Mit diesen Eigenschaften platziert sich dieser Prozessor im Low-Budget Segment und soll dort altgediente 8- und 16 Bit Modelle verdrängen. Anwendungsszenarien für den M0 können Beleuchtungs- und Motorsteuerungen sowie Messgeräte sein.

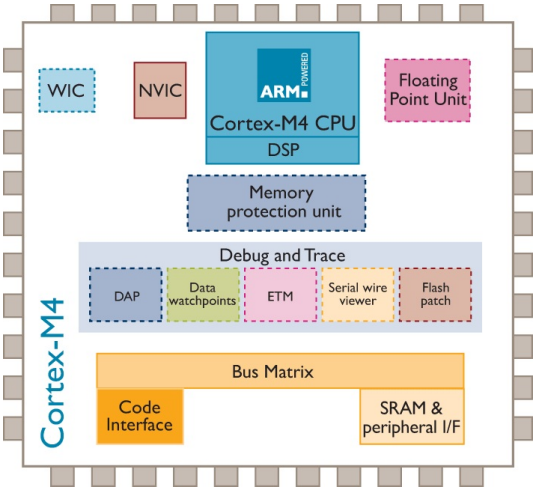
4.4 Cortex M3



Der M3 ist für anspruchsvollere Anwendungsgebiete ausgelegt. Er besitzt den vollen Thumb2 Befehlssatz und darüber hinaus noch erweiterte Debug und Trace Möglichkeiten. Zum Betrieb eines OS wurde ein Speicherschutz eingerichtet. Als eine weitere Veränderung zeigt sich die Verwendung einer Harvard Architektur anstatt der von Neumann Architektur, welche noch beim M0 Anwendung findet. Die Harvard Architektur verfügt über zwei getrennte Bussysteme zum Laden von Daten und Befehlen und kann somit in einem Zyklus das verarbeiten, was man in der von Neumann Architektur mit zwei Buszyklen erledigt. Anwendungsgebiete des M3 finden sich in der Medizintechnik, der Industrieautomatisierung und im Bereich der Haustechnik.

4.5 Cortex M4

Der M4 ist der momentan performanteste Mikroprozessor der M Reihe. Er bietet neben allen Funktionen des M3 zusätzliche neue Funktionen für die digitale Signalverarbeitung in Form eines um DSP-Befehle erweiterten Thumb2 Befehlssatzes, eine zusätzliche MAC-Einheit, die Multiply-Accumulate Operationen in einem Zyklus verarbeiten kann, sowie optional eine Gleitkomma-Einheit. Dies macht ihn für die Anwendungsgebiete Unterhaltungselektronik, Medizin- und Energietechnik interessant.



5 Quellenverzeichnis

- Yiu, Joseph: The Definitive Guide to the ARM Cortex-M3 Second Edition: Elsevier Inc, 2010
- Wüst, Klaus: Mikroprozessortechnik Grundlagen, Architekturen, Schaltungstechnik und Betrieb von Mikroprozessoren und Mikrocontrollern: Vieweg + Teubner, 2010
- http://www.arm.com
- http://www.electronic-data.com/een/2008_2/12397.asp
- http://www.onarm.com/cmsis/
- http://www.phaedsys.org/standards/misra/misraguidelines.html
- http://www.polyscope.ch/dlCenter/ps/2009_20/PS20_S20_22.pdf
- http://www.silica.com/fileadmin/06_Press/Literature_Center/MarktundTechnik_Sonderheft.pdf
- http://www.youtube.com/watch?v=ttXW58-drYw
- http://www.youtube.com/watch?v=JXE3GzNDM2c