Embedded Programming

Basiswissen und Anwendungsbeispiele der Infineon XC800-Familie

von
Prof. Dr.-Ing. Reiner Kriesten

Oldenbourg Verlag München

Prof. Dr.-Ing. Reiner Kriesten wurde 2009 an die Hochschule Karlsruhe – Technik und Wirtschaft berufen und vertritt dort Fachgebiete im Bereich der eingebetteten Systeme, Informatik und Software-Engineering in den Studiengängen Fahrzeugtechnologie und Mechatronik.

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
www.oldenbourg-verlag.de

Lektorat: Kathrin Mönch, Dr. Gerhard Pappert
Herstellung: Constanze Müller
Titelbild: thinkstockphotos.de (Abbildung: Irina Apetrei)
Einbandgestaltung: hauser lacour
Gesamtherstellung: Grafik & Druck GmbH, München

Dieses Papier ist alterungsbeständig nach DIN/ISO 9706.

ISBN 978-3-486-71284-1
eISBN 978-3-486-71985-7

Vorwort

Willkommen bei der eingebetteten Programmierung und der Inbetriebnahme von Mikrocontrollern (µCs). Es gibt mehrere Möglichkeiten, warum Sie dieses Buch aufschlagen. Entweder möchten Sie ein tieferes Verständnis in der Programmierung von µCs erlangen oder Sie wollen ein technisches System realisieren, dessen Logik in einen Mikrocontroller gegossen werden sollte. Auch ist es denkbar, dass Sie Student einer technischen Fachrichtung sind und eventuell einfach nur dieses verflixte Fach bestehen wollen.

Ebenso schwierig ist es zu erraten, welche Hintergründe Sie mitbringen. Die Programmierung von µCs bedeutet, einem technischen System Intelligenz zuzuführen – und diese wird mittlerweile in so ziemlich allen Bereichen benötigt, zum Beispiel der Industrietechnik, der Automobilindustrie, der Luftfahrt, … Entsprechend divers kann auch die Fachrichtung sein, aus welcher Sie originär stammen. Typischerweise wird diese Thematik von der Elektrotechnik oder der Informatik aus angegangen. Die Stärken und Potenziale beider Gruppen liegen auf der Hand. Während Elektroniker ein Hintergrundwissen zu dem schaltungstechnischen Aufbau eines eingebetteten Systems mitbringen sollten, gehen Informatiker diese Systeme eher vom Innern des Mikrocontrollers an und sind wohl in der Lage, die softwaretechnischen Feinheiten bei der Programmierung von µCs zu verstehen. Last but not least existieren weitere Gruppen wie Mechatroniker, Maschinenbauer, …, oder sogar der ein oder andere Mathematiker, der sich mit eingebetteten Systemen beschäftigt.

Somit stellen sich zwei prinzipielle Fragen für die Gestaltung dieses Buches. Welche Kenntnisse können vorausgesetzt werden? Und wie tief soll und muss auf theoretische Grundlagen und Hintergründe eingegangen werden?

Beide Fragen können klar beantwortet werden. Die Realisierung von Intelligenz in einem technischen System bedarf zumindest der Grundkenntnisse einer Programmiersprache und es wird hierbei in der Regel auf C zurückgegriffen. Deshalb sollte dem Leser wenigstens die Verwendung von Variablen und Kontrollstrukturen wie `if`, `while` nicht ganz fremd sein. Auf der anderen Seite ist es hilfreich, ein klein wenig Wissen im Bereich der Gleichstromtechnik mit sich zu bringen. Viel mehr als die berühmte Gleichung R=U/I sowie Kenntnisse über die grundlegende Funktionsweise eines Transistors sind aber nicht von Nöten.

Kommen wir zu der Frage, wie weit in diesem Buch auf Grundlagen und Basiswissen eingegangen wird. Ein Entwickler ist immer froh, wenn er in seinem Projekt zwei Ziele erreicht: sein System/Programm soll funktionieren und er versteht (oder glaubt zu verstehen), was er gemacht hat und warum alles läuft. Dieses Buch geht so vor, dass konkrete Anwendungen realisiert werden und die hierbei auftretenden Fragestellungen und Hintergründe erläutert werden. Die einzelnen Kapitel sind dabei so strukturiert, dass das Wissen sukzessive aufeinander aufbaut.

Aus diesen Überlegungen heraus grenzt sich das vorliegende Buch zu weiteren Büchern im Bereich *Mikrocomputertechnik* ab, die häufig grundlagenbezogener vorgehen. Zudem bedeutet die Programmierung eingebetteter µCs in einem gewissen Umfang auch ein Debugging,

und zwar ein Debugging auf Bitebene. Aus meiner Erfahrung heraus sind hierfür Kenntnisse über die Funktionsweise des Assemblers notwendig sowie das Wissen, die Lage von Programmcode und Daten im Flash und RAM zu bestimmen und zu manipulieren. Diesen Anforderungen wird im weiteren Verlauf ebenso Rechnung getragen und dem Entwickler die Möglichkeit gegeben, seine auftretenden Fehler selbständig aufzuspüren. Diverse Übungsaufgaben verschiedener Schwierigkeitsstufen sorgen dafür, dass das erlernte Wissen praktisch vertieft wird. Die Musterlösungen der Aufgaben befinden sich am Ende des Buches.

Wahrscheinlich werden Sie sich auch fragen, welche Kosten für die benötigte Hardware und Software anfallen werden und welche Programme denn überhaupt notwendig sind. Die gute Antwort zuerst: Sämtliche Softwareprogramme sind entweder Freeware oder aber als codegrößenbegrenzte Evaluierungsversion erhältlich. Softwareseitig ist an erster Stelle die Entwicklungsumgebung µVision von Keil zu nennen. Für die im Buch erstellten Programme ist die codegrößenbegrenzte Version (bis auf das Musikstück in Abschnitt 15.8) vollkommen ausreichend und unter [ARM11f] erhältlich. Weiterhin wird für den Betrieb des Evaluierungsboards der Device Access Server von Infineon benötigt, zu finden unter [INF11c]. Optional kann der Autokonfigurator DAVE in manchen Fällen gute Dienste bei der Initialisierung von Registern leisten. Auch dieser ist – inklusive seines Device Integration Packages (DIP) – über die Homepage von Infineon zu beziehen [INF11e]. Weitere Informationen zu den Programmen finden sich in Kapitel 3.

Von Seiten der Hardware existieren mehrere Optionen, diesem Buch zu folgen. Die einfachste Methode ist die Verwendung der Hardwaresimulation, welche in µVision integriert ist. Dabei wird der Mikrocontroller auf dem PC simuliert und seine relevanten Größen werden über grafische Fenster angezeigt. Bei diesem Vorgehen entstehen keinerlei Kosten. Sollen hingegen die Übungen auf realer Hardware verfolgt werden, so ist zuerst einmal ein Evaluierungsboard mit einem Controller der XC800-Familie (XC888, XC878, XC886) notwendig. Dieses findet sich unter [INF11b] mitsamt den notwendigen Kosten, die zum aktuellen Zeitpunkt im Bereich von 100 bis 150 Euro liegen. Da die Sensorik und Aktuatorik auf dem Evaluierungsboard für viele Aufgaben nicht ausreichend ist, sollte bei einem realen Betrieb entweder mit zusätzlichen Steckbrettern gearbeitet werden oder aber eine Zusatzplatine erstellt beziehungsweise bestellt werden. Die Verwendung von Steckbrettern stellt sicher die kostengünstigere Variante dar und mit diesen ist es in der Tat möglich, die Programme zu verfolgen. Neben der bereits auf dem Evaluierungsboard vorhandenen LED-Leiste und dem Potenziometer werden für die Aufgaben häufig 3 Taster sowie in einigen Anwendungen zwei 7-Segment-Anzeigen inklusive der zugehörigen Latches benötigt. Die Beschaffung der Taster, der 7-Segment-Anzeigen und der Latches sollte im Bereich weniger Euro liegen und ermöglicht es, dem Buch adäquat zu folgen. Die Verdrahtung der Bausteine kann gemäß den Schematics der Zusatzplatine erfolgen, siehe [KRI12] und Anhang 16.5.

Der Einsatz einer Zusatzplatine ist ebenfalls denkbar. Details zur Erstellung oder auch zur Bestellung der Platine finden sich unter [KRI12]. Für den Einsatz im Labor Mikrocomputertechnik der Hochschule Karlsruhe wurden ca. 20 Zusatzplatinen angefertigt. In dieser Stückzahl lagen die Kosten für die Fertigung und Bestückung bei ca. 200 Euro pro Platine.

An dieser Stelle sollen noch einige Anmerkungen zum Aufbau des Buches gegeben werden. Dem erfahrenen Programmierer wird sicherlich auffallen, dass so manche (sinnvolle) softwaretechnische Konstrukte wie die Bereitstellung von Abstraktionsschichten oder sogar die explizite Verwendung von Zeigern fehlen. Dies hat seinen guten Grund. Komplexere Programme benötigen aufgrund der Qualitäten der Wiederverwendbarkeit, Modularität oder

Effizienz eben solche Mechanismen. Jedoch werden kleinere Programme für den ungeübteren Programmierer durch diese Konstrukte unleserlicher und sie erschweren das Verständnis des µCs.

Auch die Verwendung einer hohen Anzahl von Fußnoten ist auffallend. Hierbei habe ich mich am Buch [SCH01] orientiert. An manchen Stellen des Buches stoßen wir gleichzeitig auf mehrere Fragestellungen, welche in divergente Richtungen gehen. Um den roten Faden nicht zu verlieren, werden zusätzliche Informationen in Fußnoten ausgegliedert.

Schließlich ist noch die Verwendung englischer Begriffe zu erwähnen. Im Bereich der eingebetteten Programmierung hat sich ein gewisser „Slang“ eingebürgert und bei vielen Begriffen ist es einfach nicht sinnvoll, diese einzudeutschen. Begriffe wie *Debugging*, *Steppen*, *Layout*, *Interrupt*, … oder auch *Timer*, *Counter* mit jeweils eigenständigen Bedeutungen sind als typische Vertreter anzuführen. In diesem Buch wird, soweit sinnvoll, von den englischen Begriffen Gebrauch gemacht. Verfechter der deutschen Sprache mögen mir dies verzeihen.

Danksagungen

Natürlich gelingt die Erstellung eines Buches lediglich durch die Mithilfe diverser Personen, die ich an dieser Stelle erwähnen möchte.

Zuerst sei den Dozenten Ralf Hanke und Christian Enders gedankt. Auf ihren langjährigen Einsatz in der Lehre und im Laborbetrieb sind einige Aufgabenstellungen und Lösungsansätze zurückzuführen, die auch in diesem Buch zu finden sind.

Die Erstellung der verwendeten Zusatzplatine bedurfte einer ganzen Reihe von Aktivitäten, angefangen von der Entwicklung und Optimierung elektronischer Schaltungen, dem korrekten Anschließen der Bauteile bis hin zu der Erstellung und Optimierung des Layouts sowie dem Fertigen der Probeplatinen. In all diesen Punkten standen mir Herr Beck, Herr Pluschke, Herr Stumpf und Herr Jäger beiseite und trugen ihren Teil zur Entwicklung der Platine bei.

Selbstverständlich kann dieses Buch auch nicht ohne die freundliche Unterstützung des Chip-Herstellers Infineon gelingen. Einige der in diesem Buch verwendeten Abbildungen sind aus Quellen von Infineon entnommen und helfen, den jeweiligen Sachverhalt plastisch darzustellen. Herr Kroh gebührt der Dank für die stete und schnelle Unterstützung bei der Beschaffung der Evaluierungsboards, den Aktivitäten zur Genehmigung der verwendeten Abbildungen und den gut gemeinten Wünschen, die Studenten mit der XC800-Familie „fleißig zu quälen“.

Die Verwendung der Entwicklungsumgebung µVision von KEIL (ARM Germany GmbH) ermöglicht mit seinen vielfältigen Analysemöglichkeiten die reibungslose Inbetriebnahme der Controller und trägt wesentlich zum erfolgreichen Erlernen der Thematik bei. Auch bei ARM erfuhr ich bei meinen Anfragen prompte und freundliche Unterstützung.

Neben den gerade erwähnten Hilfestellungen ist ein kompetentes Review bei der Erstellung eines Buches unerlässlich. Tom Henrich und Christian Enders haben sich in meinem Fall freundlicherweise dieser Aufgabe angenommen.

Einen Hauptteil der Last für dieses Buch trägt jedoch – wie könnte es anders sein – die Familie. So endete der ein oder andere Abend vor dem PC anstelle von einem schönen Glas Rotwein. Danke hierfür an Carolin für die Geduld. Auch meiner Rasselbande Jannik, Moritz und nun Marie sei hier gedankt – sobald sie in der Lage sind, diese Zeilen zu lesen.

Inhaltsverzeichnis

1 Einführung

1.1 Über den Einsatz und die Inbetriebnahme von µCs

In vielen Bereichen wie der Fahrzeugindustrie, der Automatisierungstechnik oder der Medizintechnik gewinnen elektronische Systeme – bestehend aus der Wirkungskette Sensorik, Mikrocomputer (µC), Aktuatorik – seit längerer Zeit an Bedeutung. Laut [MMC09] werden zwei Drittel der Innovationen im Automobil durch die Elektronik ermöglicht oder von ihr maßgeblich beeinflusst, wobei Elektronik als Gesamteinheit von Hardware (HW) und Software (SW) verstanden wird. Über die Programmierung der µCs bestimmt sich hierbei maßgeblich die gesteigerte Intelligenz des elektronischen oder mechatronischen Systems.

Rechnereinheiten, welche in derartigen Systemen eingesetzt sind, werden als *Eingebettete Systeme* oder auch *Embedded Systems* bezeichnet [WIK11b]. Sie unterscheiden sich von „gewöhnlichen“ PC-Rechnern (Personal Computer) in der Regel durch eine geringere Leistungsfähigkeit, einem niedrigeren Verbrauch an Energie sowie der Tatsache, dass sie diverse Aufgaben in technischen Umfeldern wahrnehmen müssen. Hieraus resultieren auch die unterschiedlichen Schnittstellen zur „Außenwelt“[1].

Dieser Trend der vielseitigen Anwendungen spiegelt sich auch im Aufbau der (eingebetteten) Mikrocomputer wider. Anforderungen sind vorhanden, mit Hilfe eines Chips – oder einem Set von ähnlichen Derivaten – eine Unmenge von Applikationen abdecken zu können und den Einsatz unter veränderten Randbedingungen zu gewährleisten. Auf elektrischer Ebene bedeutet dies, dass Chips für verschiedene Umgebungstemperaturen ausgelegt werden, mit mehreren Spannungspotenzialen arbeiten oder aufgrund von Kostenfaktoren mit einer unterschiedlichen Anzahl von Eingangs- und Ausgangspins hergestellt werden.

Interessant ist ein Vergleich der Produktion eingebetteter Mikrocontroller und der Controller für den Markt der Personal Computer (PC). Während 1990 ca. eine Milliarde eingebetteter µCs hergestellt wurden, liegt die jährliche Produktion in 2010 bereits bei über 10 Milliarden Stück mit steigendem Verlauf [MAT10]. Bei PC Prozessoren werden hingegen weitaus geringere Steigerungsraten und Stückzahlen erreicht. Letztere liegen in 2010 bei ungefähr 300 Millionen [FAZ11].

Der breit gefächerte Einsatz eines µCs zeigt sich insbesondere in seinen Konfigurationsmöglichkeiten. So existieren heutzutage unzählige Register, die das Verhalten des Rechners beeinflussen. Die Inbetriebnahme eines µCs setzt folglich in einem ersten Schritt voraus, sich mit seinen Möglichkeiten auseinanderzusetzen und den Einfluss der Register auf sein Verhalten zu verinnerlichen.

Die Konfiguration der Register erfolgt über Software und stellt in der Regel den Initialisierungsteil eines Programms dar. Hierbei ist es sekundär, welche Art der Programmiersprache

[1] In einem eingebetteten System ist bereits die Ausgabe auf einem Bildschirm oder die Eingabe über eine Tastatur nicht standardmäßig vorgesehen.

verwendet wird. Exakter formuliert wird unter der Konfiguration eine Belegung der Register mit bestimmten Werten verstanden und es ist essentiell, den *Sinn* der Register zu verstehen. Das eigentliche Setzen der Werte ist hingegen trivial und kann über einfache Anweisungen realisiert werden wie `P3_DIR=0xFF` in C oder `mov P3_DIR,#0FFh` in Assembler. Erst im weiteren Verlauf der Programmierung „höherwertiger“ Logik ergibt sich ein deutlicher Unterschied in der Verwendung der Programmiersprachen. So wird in dem Buch deutlich, dass höherwertige Konstrukte leichter in einer dafür vorgesehenen Sprache (C) realisiert werden können als in einer maschinennahen Beschreibung (Assembler).

Neben der zu erstellenden Software wird ein Mikrocomputer in der Regel an eine reale Hardware angeschlossen und ist in diese Umgebung integriert. Die hierfür notwendigen Fähigkeiten zur Auslegung elektronischer Schaltungen, zur Erstellung von Schematics, … sind deutlich anders geartet als die Fähigkeiten, den Rechner softwareseitig in Betrieb zu nehmen. Aus diesem Grund stellen die Hersteller der µCs Evaluierungsboards bereit, bei welchen bereits eine Verdrahtung zu bestimmter Sensorik und Aktuatorik vorhanden ist (Taster, Potentiometer, LED, …).

Schließlich ist zu erwähnen, dass in vielen Unternehmen die Hardware-Entwicklung und die Software-Entwicklung in verschiedene Entwicklungsbereiche getrennt sind und eine Spezialisierung in beiden Richtungen zu erkennen ist.

1.2 Auswahl des Mikrocontrollers

Die in diesem Buch zugrunde gelegte Familie der *Infineon XC800* Mikrocomputer ist in dem Kontext zu sehen, eine große Anzahl von Applikationen abbilden zu können. Sie basiert auf dem wohl am weitest verbreiteten Controller, dem 8051, und stellt den aktuellen Stand der Technik als Erweiterung dieses Controllers dar. Die XC800-Familie *basiert* somit auf dem 8051-Controller, jedoch sind diverse zusätzliche Funktionalitäten in die Prozessorarchitektur integriert.

Erwähnt sei, dass Derivate des 8051 von anderen Herstellern ähnlich in Betrieb genommen werden können. Selbstverständlich werden hierbei unterschiedliche Register mit anderen Bedeutungen existieren, die eine oder andere Peripherie-Einheit wird unterschiedlich gestaltet sein, … Jedoch ist das grundlegende Verständnis und die Vorgehensweise zur Inbetriebnahme vergleichbar. Kurzum: wer einen µC der XC800-Familie programmieren kann, der sollte in der Lage sein, auch andere µCs selbständig in Betrieb zu nehmen – und dies stellt ein Hauptziel dieses Buches dar.

Warum fiel die Wahl auf die XC800-Familie, oder genauer das Derivat *XC888*, um den Umgang mit eingebetteten Controllern zu erlernen? Hierfür existieren mehrere Gründe:

- Die 8051-Prozessorfamilie stellt wohl die bekannteste Controllerfamilie dar und es existiert somit ein reichhaltiges Set an Literatur und Möglichkeiten der Internetrecherche. Viele Anwendungen finden sich beispielsweise in [INF11d].
- Unter Verwendung der Entwicklungsumgebung (*IDE*, Integrated Development Environment) von Keil [ARM11f] kann ein erstelltes Programm auf dem PC simuliert werden. Selbst für die Verifikation und das Debugging von Programmen ist somit keine Hardware notwendig, die Programme werden auf PC-Ebene simuliert. Allerdings sei betont, dass diese Simulation auf einer künstlichen Hardware nicht immer dieselben Resul-

tate liefert wie in der Praxis. Ein einfaches Beispiel ist das Drücken eines Tasters. Während reale Taster *prellen*, ist dies über einen Simulationswerkzeug nur schwerlich abzubilden.

- Bei dem gewählten Mikrocomputer handelt es sich um eine klassische 8-Bit Architektur [WIK11]. Diese besitzt im Gegensatz zu 16-Bit, 32-Bit oder 64-Bit-Systemen eine Reihe an „Herausforderungen“. So reicht beispielsweise der mögliche Adressraum ohne weitere Maßnahmen nicht aus, um sämtliche geforderten Funktionen abzubilden oder exakter, um sämtliche Register mit einer eindeutigen Adresse zu belegen. Solche Herausforderungen zwingen den Entwickler, auf wichtige Randbedingungen wie Speicherrestriktionen oder eine begrenzte Bitbreite ein verstärktes Augenmerk zu legen. Er lernt damit die Eigenarten von Controllern kennen und insbesondere den Unterschied zur Programmierung gewöhnlicher PCs.
- Die vorgestellte XC800-Familie besitzt ein breites Einsatzgebiet und deckt insbesondere aktuelle Trends im Automotive Umfeld ab, zum Beispiel *LIN*-Kommunikation (Local Interconnect Network), Ansteuerung von Brushless-Motoren, erweiterte Konzepte zur Hall-Auswertung, …
- Mit dem Konfigurationswerkzeug *DAVE* (Digital Assistant Virtual Engineer) gelingt es in effizienter Weise, eine Grundkonfiguration für den µC zu erstellen. Auf diesem Gerüst aufbauend kann fortan direkt „in der Applikation“ entwickelt werden. Ein großer Teil der Initialisierung kann folglich über Konfigurationswerkzeuge abgedeckt werden[2].

In dem Buch wird die eingebettete Programmierung der Mikrocomputer anhand von konkreten Beispielen der XC800-Familie von Infineon vorgestellt. Das dabei verwendete Derivat XC888 ist insbesondere den Derivaten XC878, XC886 sehr ähnlich, so dass diese für den weiteren Verlauf ebenfalls geeignet sind.

1.3 Zielgruppe und Aufbau des Buches

In der Literatur existieren divergente Ansätze, dem Leser das Thema Mikrocomputertechnik näher zu bringen. Dabei sind die Zielsetzungen verschiedenartig. So kann einerseits der Fokus auf die Grundlagen von Mikrocomputern gelegt werden. Dies beinhaltet die Vorstellung der Speicherbausteine, des Assembler-Befehlssatzes, Informationen zum Aufbau der *ALU* (Arithmetic Logic Unit) und der Befehlsabarbeitung etc.

Auf der anderen Seite steht die Inbetriebnahme – oder besser die konkrete Programmierung – von Mikrocomputern. Hierbei ist das Verständnis über die Grundlagen und den inneren Aufbau eines µCs zwar förderlich, aber bei Weitem nicht ausreichend. Zudem werden Programmbeispiele im Bereich der Grundlagen häufig in Assembler verfasst (über Assembler ist es möglich, konkret die Benutzung der ALU und des proprietären Befehlssatzes eines µCs deutlich zu machen, so dass dies eine logische Fortführung des Konzepts ist). Jedoch spielt seit sicher mehr als einem Jahrzehnt die Sprache C die Hauptrolle bei eingebetteten Controllern. Es stellt sich daher die Frage, auf welche Art und Weise die Programmierung von Mikrocomputern vermittelt werden soll: über die Anwendung von C einerseits oder über die Darlegung von Grundlagen andererseits.

---

[2] Dennoch wird in diesem Buch nur oberflächlich auf DAVE eingegangen, da das grundlegende Verständnis der Register in einem ersten Schritt erlernt werden muss.

Dieses Buch geht hierbei pragmatisch vor. Assemblerprogramme und weitere Grundlagen sind insoweit Gegenstand des Buches, wie es für das Verständnis notwendig ist. Beispielsweise ist es in einem C-Programm nicht offensichtlich, an welcher Speicherstelle bestimmte Programm- und Datensegmente platziert sind, in Assembler jedoch sehr wohl[3]. Aus diesem Grund wird derartiges Wissen zu Beginn über Assembler dem Leser vermittelt, jedoch im weiteren Verlauf vorwiegend auf die Programmierung in C eingegangen. Abschnitte, welche sich „über den notwendigen Bedarf hinaus“ mit Assembler beschäftigen, sind optional und mit einem * gekennzeichnet.

Die Zielgruppe dieses Buches ist auf Studenten und Ingenieure fokussiert, welche ein Basiswissen in C aufweisen. Aus diesem Grund steht in den vorhandenen Beispielen die *Lesbarkeit* der Programme im Vordergrund und nicht etwa die softwaretechnisch „beste“ Lösung. Es wird absichtlich auf die extensive Verwendung von *Getter-* und *Setter-Funktionen* oder *Makros* verzichtet. Diese finden in Realität zwar breite Verwendung und stellen einen sinnvollen Abstraktionsmechanismus für Module bereit. Jedoch führt der Einsatz solcher Mechanismen bei der Zielgruppe häufig zu Irritationen im Gegensatz zu einer „Straight-Forward“-Programmierung. Auch wird auf die Auslagerung von Programmteilen in verschiedene Module und deren Integration über Header-Dateien verzichtet.

Über das Konfigurationswerkzeug DAVE [INF11e] kann eine Basiskonfiguration verschiedener Peripherie-Einheiten über eine grafische Bedienoberfläche erfolgen. Nachteilig wirkt sich jedoch aus, dass durch bloße grafische Konfiguration das Verständnis des µCs nicht gefördert wird und dem Anwender nicht unbedingt bewusst wird, inwieweit die getroffenen Einstellungen sinnvoll sind[4]. Auch ist der von DAVE erstellte Code für große Projekte angelegt und daher sehr stark modularisiert und abstrahiert, so dass die Lesbarkeit des generierten Codes beeinträchtigt ist. Aus diesem Grund wird das Konfigurationswerkzeug lediglich kurz vorgestellt, aber nicht weiter verwendet.

Das Niveau des Buches entspricht einem Kurs Mikrocomputertechnik oder eingebettete Programmierung an technischen Hochschulen und genau für diesen Zweck wurde das Buch erstellt.

Erwähnt sei weiter, dass durch die Existenz eines Simulators die Anwendungen auch ohne Hardware verifiziert werden können. Für den Einsatz und den Bezug realer Hardware in Form von Evaluierungsboards und Zusatzplatinen stehen weitere Informationen bereit und können über den Autor [KRI12] oder [INF11b] in elektronischer Form bezogen werden (Schematics, Layout, Beschreibung elektronischer Bauteile, …).

---

3 In C-Programmen übernimmt dies in der Regel das Linker-File.

4 Hierfür muss die Kenntnis vorhanden sein, was die Einstellungen in einzelnen Registern bewirken.

2 Rechnerarchitekturen

2.1 Kenngrößen von Mikrocontrollern

Aktuelle Mikrocontroller weisen verschiedenartige Performanz und unterschiedliche Schnittstellen auf. Für eine Klassifikation bedarf es in einem ersten Schritt der Kenntnis, welche Parameter eines µCs relevant sind:

- *Taktfrequenz*: Unter der Taktfrequenz wird die Geschwindigkeit des Rechnerkerns verstanden. Ein Mikroprozessor besitzt eine – intern oder extern vorgegebene – Taktquelle und aus dieser wird die Zeit abgeleitet, mit welcher der Kern seine Befehle ausführt. Es muss somit unterschieden werden zwischen der Frequenz des Rechnerkerns und der Frequenz der vorgegebenen Taktquelle, in der Regel einem Oszillator oder einer *PLL* (Phasenregelschleife, Phase-Locked Loop). Bei dem in diesem Buch verwendeten XC888-Rechner ist der Takt des Kerns, die *CPU Clock* C_{CLK} (Central Processing Unit), standardmäßig auf 24 MHz gesetzt. Weiter ist zu beachten, dass der Takt des Kerns nicht identisch sein muss zum Takt der internen Peripherie-Einheiten, siehe Abschnitt 2.2. Der hierfür geltende Takt wird im weiteren Verlauf als *Peripheral Clock* P_{CLK} bezeichnet.
- *Flash-ROM*: Das Flash-ROM (Read-Only Memory) stellt den internen Programmspeicher eines Rechners dar. Die Größe des Flashs bestimmt, welche Menge an Programmcode auf den Rechner geladen werden kann[5].
- *RAM*: Das RAM (Random Access Memory) stellt den Datenspeicher in einem Rechner dar. Grob formuliert bestimmt dieser die Menge möglicher Variablen eines Programms[6].
- *EEPROM*: Das EEPROM (Electrically Erasable Programmable Read-Only Memory) ist ein Datenspeicher, welcher bei Entzug der Versorgungsspannung seine Daten beibehält. Deshalb wird er typischerweise zur Speicherung von Konfigurationsparametern eingesetzt. In Automobilen bedeutet dies beispielsweise die Aussage, ob ein Fahrzeug Rechts- oder Linkslenker ist, ob Xenon-Licht verbaut ist, die Speicherung von Fehlerzuständen, … Sowohl der Flash-ROM als auch der EEPROM verlieren ihre Daten bei Spannungsverlust nicht, sind also *nicht volatil*. Jedoch unterscheidet sich die Art der Speicherverwendung in der Anzahl möglicher Schreibzugriffe, der Art und Weise des lesenden Zugriffs, … Insofern ist die Trennung von nicht volatilem Datenspeicher einerseits und Programmspeicher andererseits in unterschiedlichen physikalischen Medien gerechtfertigt.
- *Anzahl der Pineingänge und Pinausgänge*: Über die Pins führt ein Mikrocomputer die Interaktion zum umgebenden System aus. Eine höhere Pinanzahl gibt folglich die Möglichkeit, mehr Sensorik und Aktuatorik anzuschließen.

---

5 Leider ist der Begriff *Read-Only Memory* bei einem Flash-Speichermedium nicht mehr zeitgemäß. Ein Flash-ROM kann diverse Male überschrieben werden.

6 Auch der Begriff *Random Access Memory* ist nicht ganz passend, denn ein Hauptmerkmal des RAM ist der Verlust der Daten bei Entzug der Versorgungsspannung.

- *Peripherie-Einheiten*: Neben dem eigentlichen Kern besitzt ein Mikroprozessor weitere interne Peripherie, siehe Abbildung 4 und Abbildung 5. Ein typisches Beispiel einer Peripherie-Einheit stellen Timer dar. Diese ermöglichen es dem Entwickler, definierte Zeitintervalle zu erstellen, siehe Kapitel 9.
- *Wortbreite eines Rechners*: Die Wortbreite stellt die Grundverarbeitungsdatengröße bei einem Rechner dar [WIK11c]. Dies bedeutet, dass sowohl die CPU-Architektur als auch häufig Adressleitungen mit einer entsprechenden Anzahl von Bits ausgelegt sind. Klassische Wortbreiten bei Mikrocontrollern sind 8-Bit, 16-Bit oder auch 32-Bit.

Wie Abbildung 1 zeigt, werden eingebettete Mikrocontroller zuerst einmal anhand ihrer Wortbreite klassifiziert. Die Relevanz dieser Kenngröße verdeutlicht das folgende Beispiel.

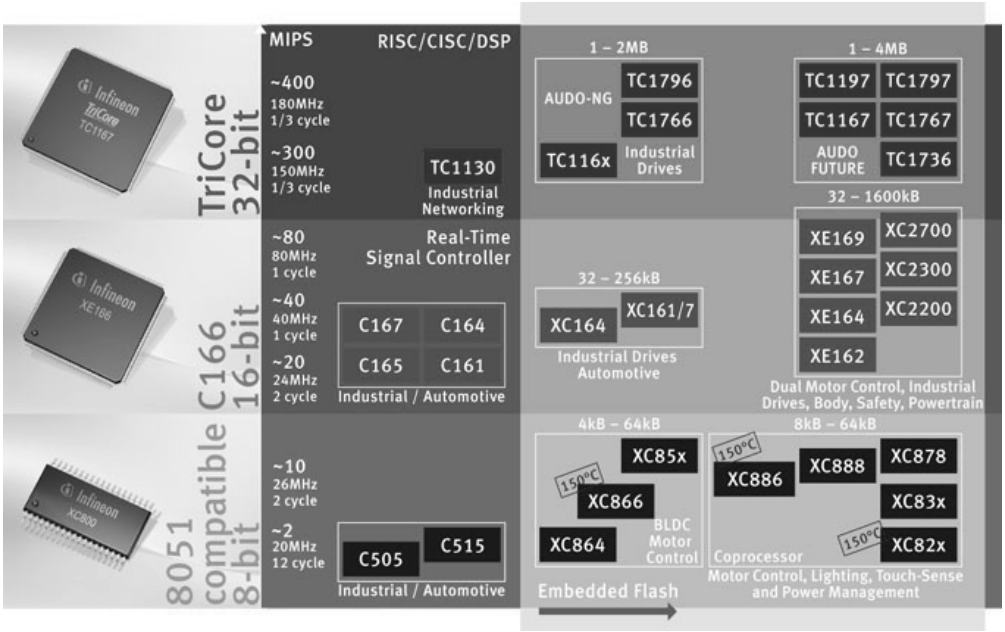

Abbildung 1: Überblick der Mikroprozessor-Familien von Infineon [INF11].

Beispiel: Ein 32-Bit Prozessor kann in seiner Recheneinheit zwei 32-Bit breite Variablen `a_32`, `b_32` direkt einlesen und verarbeiten, also beispielsweise addieren und das Ergebnis zurückspeichern. Hierbei muss der Rechner keine besonderen Maßnahmen ergreifen. Abbildung 2 zeigt diese gewöhnliche Addition von zwei 32-Bit Variablen in einer 32-Bit Architektur, `ergebnis_32=a_32+b_32`. Beginnend mit dem niederwertigen Bit werden die einzelnen Bits der Variablen `a_32`, `b_32` miteinander addiert sowie ein eventuelles Überlaufbit der vorigen Bitstelle. Der Ergebnisspeicher, genannt *Akkumulator*, ist ebenfalls 32-Bit breit und kann das Gesamtergebnis problemlos aufnehmen (allerdings ohne einen möglichen Überlauf im höchstwertigen Bit, hierfür ist in der Regel ein separates Überlaufbit vorhanden)[7].

[7] Der Akkumulator kann selbst als Zwischenspeicher agieren und somit dieselbe physikalische Einheit wie der Ergebnisspeicher sein.

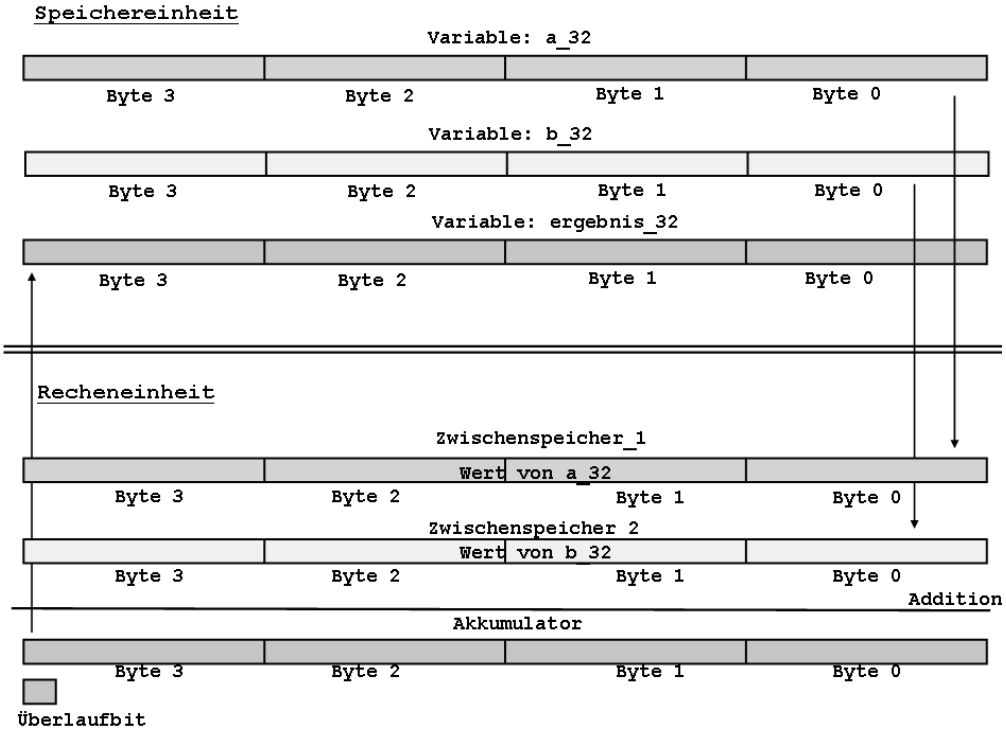

Abbildung 2: Addition von zwei 32-Bit Variablen in einer 32-Bit Architektur.

Hingegen gestaltet sich die Addition von zwei 32-Bit Variablen `a_32`, `b_32` in einer 16-Bit Architektur schwieriger, siehe Abbildung 3. Da in der CPU lediglich 16-Bit Werte addiert werden können besteht ein Lösungsansatz darin, zuerst die „niederwertigen Hälften“ der Variablen miteinander zu addieren und das Ergebnis (ohne Überlauf) stellt die niederwertige Hälfte des Gesamtergebnisses dar. Die höherwertige Hälfte des Gesamtergebnisses ergibt sich anschließend durch die Addition des Überlaufbits und der höherwertigen Hälften der Variablen `a_32`, `b_32`. Die vom Rechner ausgeführte Abarbeitungssequenz stellt sich im Pseudocode wir folgt dar, wobei `low(var)` die niederwertigen 2 Byte einer 4-Byte Variablen `var` darstellt:

```
low(ergebnis) = low(a_32)+low(b_32);
high(ergebnis)= high(a_32)+high(b_32)+Ueberlaufbit_low;
```

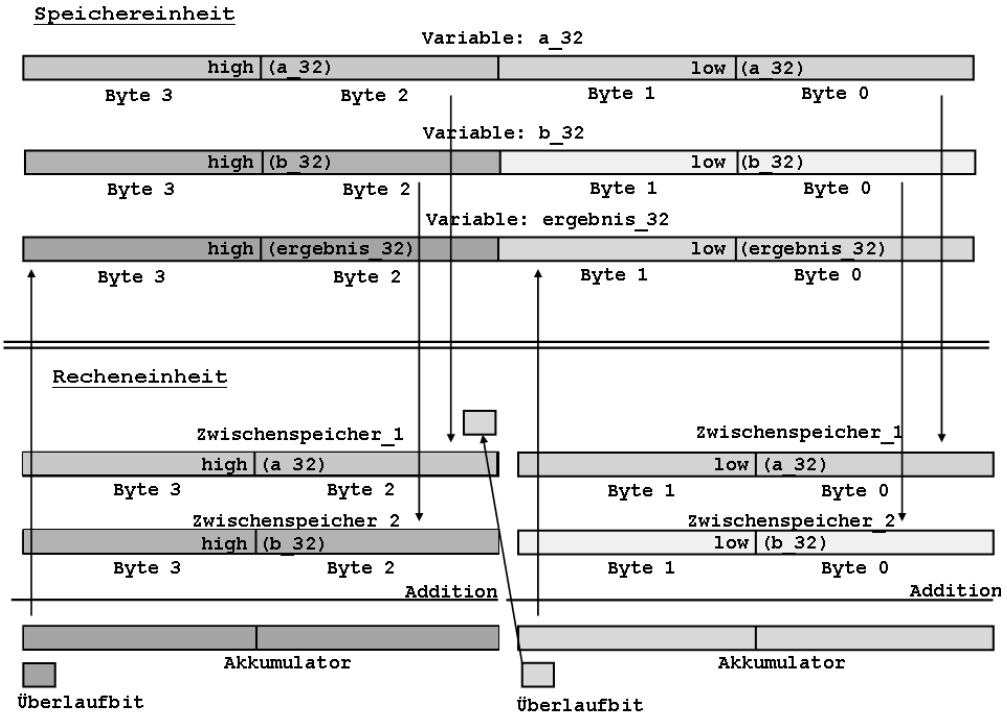

Abbildung 3: Addition von zwei 32-Bit Variablen in einer 16-Bit Architektur.

Fazit: Bereits eine Addition von zwei Variablen kann in Abhängigkeit der gewählten Wortbreite wesentliche Unterschiede in der Ausführungsdauer nach sich ziehen.

2.2 Die Funktionseinheiten der XC800-Familie

Die Mehrzahl der aktuellen 8-Bit Prozessoren besitzt denselben Ursprung, den 8051-Volkscontroller [WAL07]. Verschiedenartige Ausprägungen existieren auf dem Markt und sind zugeschnitten auf die jeweiligen Applikationen. Die im Folgenden verwendete Controller-Familie XC800 genügt einer Vielzahl von Anforderungen der Industrietechnik und der Fahrzeugindustrie, siehe [INF11]. Beispiele hierfür sind Busanschlüsse wie CAN [LAW11], LIN [LIN11], aber auch verbesserte Möglichkeiten der Auswertung von *PWM*-Signalen (Pulsweitenmodulation) oder Hallsensoren. Diesen Anforderungen wird in der Regel Rechnung getragen, indem entsprechende interne Peripherie-Einheiten in den µC integriert werden.

Abbildung 4 und Abbildung 5 stellen die Rechnerarchitekturen des aktuellen XC888-Derivats und eines älteren C515C-Derivats der Vorgängerfamilie gegenüber. Es ist deutlich zu erkennen, dass eine Vielzahl von Peripherie-Einheiten in den aktuellen µCs hinzugekommen ist. Nichtsdestotrotz besitzen beide Chips ähnliche Basisfunktionalitäten wie *Timer*, *Capture/Compare*-Einheiten und einen verwandten Rechnerkern.

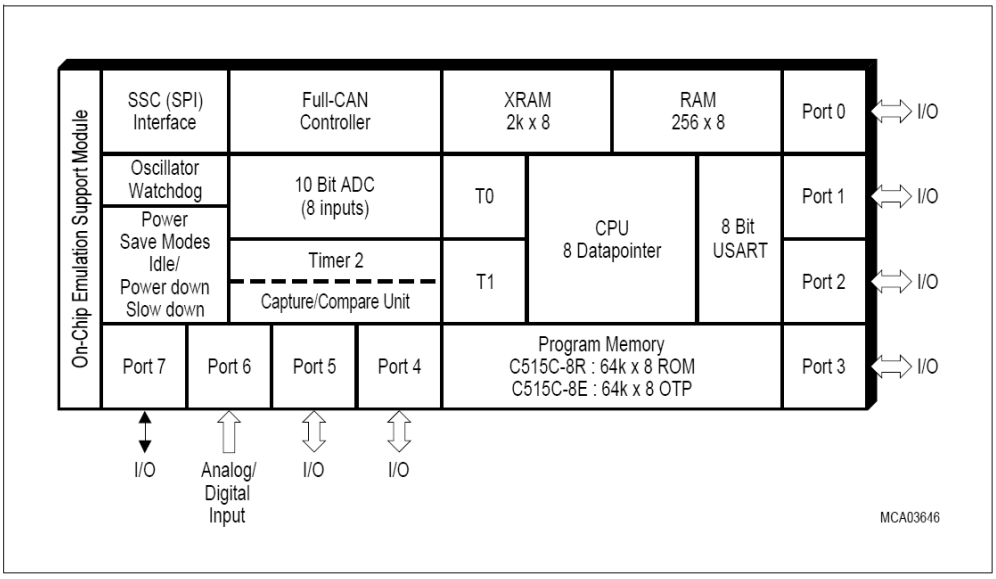

Abbildung 4: Funktionseinheiten des älteren C515C-Derivats [INF00].

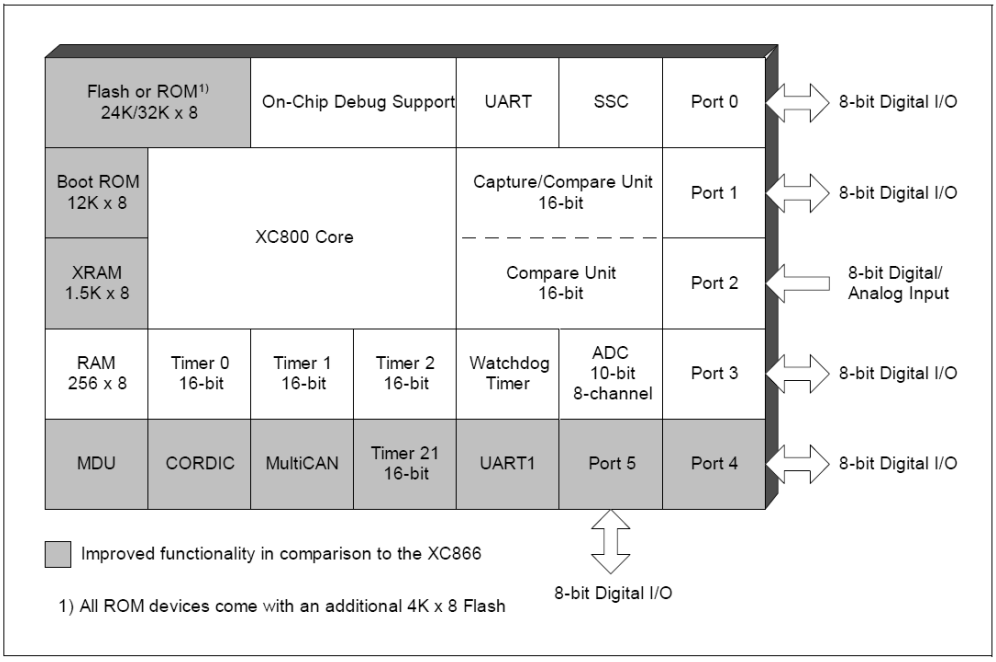

Abbildung 5: Funktionseinheiten des Infineon XC888-Derivats [INF10].

2.3 Abgrenzung der Begrifflichkeiten

Häufig werden die Begriffe *Mikroprozessor*, *Mikrocontroller*, *Chip* und *Mikrocomputer* synonym zueinander verwendet. Jedoch sind bei genauerer Betrachtung Unterschiede vorhanden, welche für das Verständnis relevant sind.

Der Begriff des Mikroprozessors stellt im eigentlichen Sinne das Rechenwerk oder die CPU dar, also das „nackte Gehirn“. In Abbildung 5 wird dieses unter dem Namen *XC800 Core* geführt. Verschiedene Peripherie-Einheiten ergänzen den Prozessor zu einem Mikrocontroller, welcher in ein Gehäuse – einem Chip – integriert ist. Eine Trennung von Peripherie-Einheiten und Prozessor wird getroffen, da die Peripherie nach der Konfiguration häufig ihre Aufgaben erledigen kann, ohne den Prozessor zu belasten.

Der Begriff des Mikrocomputers wird hingegen verwendet, wenn von der gesamten Platine die Rede ist. Diese beinhaltet neben dem Mikrocontroller weitere externe Bausteine und Treiber wie DA-Wandler (Digital-Analog), Strom- und Spannungsverstärker, Transceiver, Spannungsregulatoren, … Abbildung 6 zeigt die Platine des XC800-Evaluierungsboard, wobei oben mittig der Mikrocontroller sehr schön zu sehen ist.

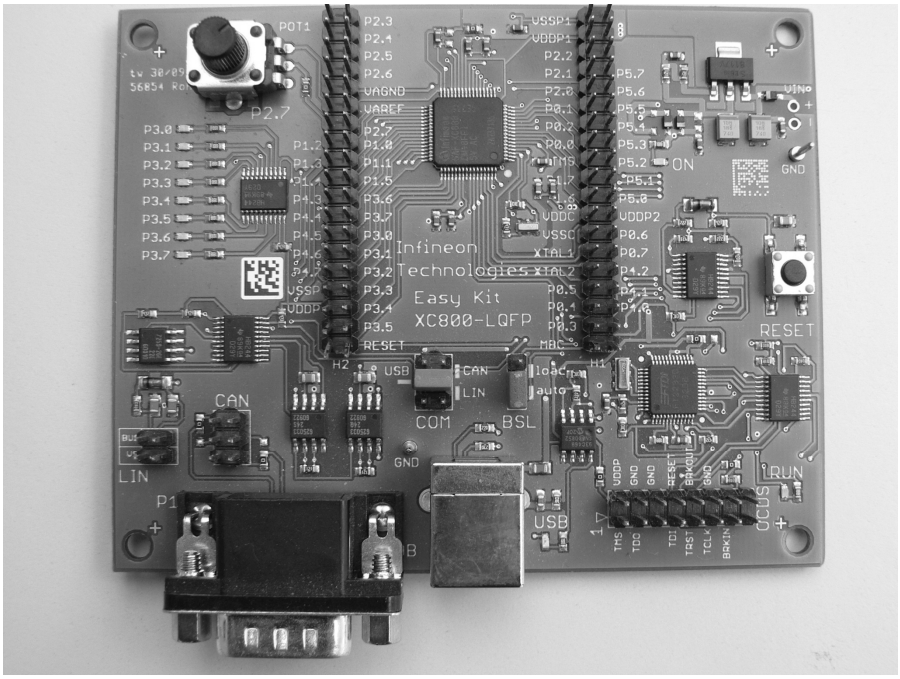

Abbildung 6: Mikrocomputer mit XC888-Mikroprozessor.

3 Inbetriebnahme der HW und der SW

3.1 Überblick über die notwendigen Komponenten

Bei der Inbetriebnahme von Mikrocontrollern spielen einige Komponenten zusammen. So ist es einerseits notwendig, ein Programm zu besitzen, mit welchem der Code verfasst werden kann. Anschließend ist dafür zu sorgen, dass das Programm in eine für den Controller verständliche Sprache übersetzt wird und schließlich auf den Rechner geladen und ausgeführt wird.

Andererseits sollte die notwendige Hardware zur Verfügung stehen. Der µC muss dabei in der Lage sein, mit dem PC zu kommunizieren, auf welchem der übersetzte Code platziert ist. Zusätzlich sollte der Controller an Sensorik und Aktuatorik angeschlossen sein, damit eine Beeinflussung dieser „Außenwelt“ über das Programm möglich ist.

Und schließlich ist es für Entwicklungszwecke sinnvoll, wenn alternativ zu der realen Hardware eine Umgebung existiert, welche diese Hardware simuliert. Auf diese Weise ist es möglich, bei einer Fehlersuche die Ursache in Bezug auf Hardware oder Software einzugrenzen. Zudem wird über einen solchen Simulator sichergestellt, dass Programme auch ohne Existenz von Hardware entwickelt und (zu einem großen Teil) verifiziert werden können.

In diesem Kapitel werden die Anforderungen und das Setup beschrieben, um solch ein lauffähiges Gesamtsystem zu erlangen.

3.2 Die integrierte Entwicklungsumgebung

Folgende zentrale Frage soll in diesem Abschnitt geklärt werden: *Welche Anforderungen muss ein Programm besitzen, das mit einem Mikrocontroller interagiert*? Hierfür kann eine ganze Reihe von signifikanten Punkten aufgezählt werden:

- Es soll ein Editor existieren, durch den der Mikrocontroller „bequem“ programmiert werden kann. Unter bequemer Programmierung kann das *Syntax-Highlighting* verstanden werden, dass *Auto-Complete* von Variablen- und Funktionsnamen, …
- Die Programmierung des Mikrocontrollers soll mit Hilfe verschiedener Programmiersprachen erfolgen können.
- Es muss ein Übersetzer existieren, welcher das Programm in eine für den µC verständliche Form transferiert. Dieser Übersetzer ist für die Programmiersprache C ein Set bestehend aus den Komponenten *Präprozessor*, *Compiler*, *Linker* und *(Flash-)Loader*[8].
- Es muss die Möglichkeit bestehen, dass übersetzte Programm auf den Mikrocomputer zu *flashen*. Das Programm muss somit über eine physikalische Verbindung auf den Mikroprozessor übertragen werden, so dass es von diesem auch ausgeführt werden kann.

[8] Es ist erkennbar, dass das gesuchte Programm in Realität ein ganzes Set an zusammenhängenden Programmen ist.

- Es soll die Möglichkeit bestehen, ein Programm zu *debuggen*. Dabei wird der Programmablauf auf dem µC nicht etwa vom Mikrocontroller selbst gesteuert. Vielmehr besteht für den Entwickler die Möglichkeit, die Abarbeitung der Befehlssequenzen auf dem µC über seinen PC zu steuern. Zudem soll es möglich sein, einzelne Variablen während des Ablaufs zu betrachten und zu modifizieren.
- Häufig ist die Hardware in einem Projekt nicht sofort verfügbar und ein erstellter Code wartet somit eine gewisse Zeit auf seinen Funktionstest. Insofern ist es wichtig, einen Simulatorbetrieb für einen µC zu besitzen. Ein erstelltes Programm wird in diesem Fall nicht auf die Hardware geladen und dort ausgeführt, sondern es existiert eine virtuelle Simulation der Hardware. Die Codeausführung wird auf dem PC simuliert und alle relevanten Variablen, Eingänge und Ausgänge können über grafische Bedienelemente betrachtet und modifiziert werden. Abbildung 15 stellt einen Screenshot eines solchen Simulatorbetriebs dar.
- Es soll extensive Literatur zu dem Programm vorhanden sein.
- Es soll möglich sein, zumindest Software kleiner Codegröße kostenlos zu entwickeln.

Ein Tool, welches sämtliche beziehungsweise die meisten dieser Anforderungen erfüllt, wird als (integrierte) Entwicklungsumgebung oder *IDE* (Integrated Development Environment) bezeichnet.

3.3 Arbeiten mit der Entwicklungsumgebung

3.3.1 Download und Installation der IDE

Die Wahl der IDE fällt in diesem Buch auf *µVision* von *KEIL*. Diese Umgebung erfüllt alle der im letzten Abschnitt erwähnten Punkte und stellt eine sehr bekannte Entwicklungsumgebung im Bereich der 8051-Controllerfamilie dar. Die Installation der IDE ist in der Regel problemlos möglich durch das Befolgen der Anweisungen während des Setups. Spezielle Einstellungen sind nicht vorzunehmen. Der Download für eine codegrößenbegrenzte Lizenz erfolgt über die Homepage von Keil [ARM11f].

Die im weiteren Verlauf entwickelten Programme sind unter der Version *µVision4* von Keil auf einem Windows-XP Betriebssystem entstanden[9]. Probleme in Verbindung mit anderen Betriebssystemen sind nicht bekannt.

3.3.2 Anlegen eines Projekts

Nach dem Öffnen von µVision muss neues *Projekt* ausgewählt werden. In diesem Projekt sind sämtliche Informationen beinhaltet, die ein Programm zur Ausführung benötigt. Dies sind die notwendigen Quelldateien, die Festlegung des Zielprozessors, die Einstellung für den Betrieb auf realer Hardware beziehungsweise in der Simulationsumgebung, …

Das Anlegen des Projekts erfolgt durch Auswahl von *Project* → *New µVision Project*. Nach der Festlegung des Projektnamens und des Speicherorts ist der gewünschte Prozessortyp zu

[9] Genauer gesagt wurde die Version µVision 4.10 verwendet.

selektieren, siehe Abbildung 7. In unserem Fall fällt die Auswahl auf den *XC888CM-8FF* von Infineon[10].

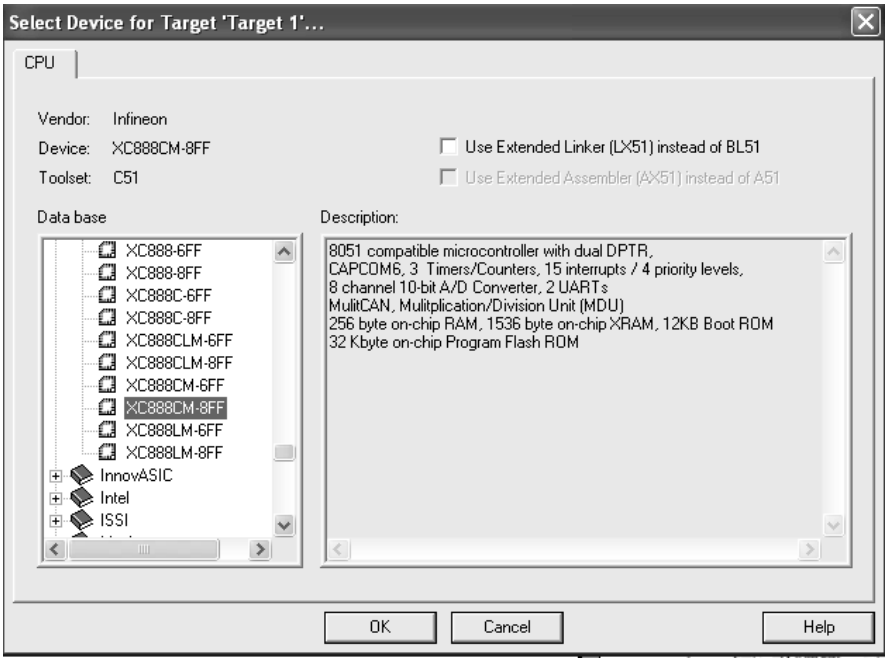

Abbildung 7: Auswahl des XC888-Derivats beim Anlegen des Projekts.

Anschließend wird gemäß Abbildung 8 gefragt, ob der Infineon XC800 Startup-Code in das Projekt integriert werden soll. Dieser Code initialisiert bestimmte Speicherbereiche mit dem Wert 0 und nimmt (wenige) weitere Aktionen vor. Beachtenswert ist, dass der zur Verfügung gestellte Startup-Code in Assembler verfasst ist, da dies besonders effizient realisierbar ist. Selbstverständlich kann hierauf aufbauend der eigene Code in C verfasst werden[11]. Für die folgenden C-Programme wird der Startup-Code jeweils hinzugefügt[12]. Im Gegensatz hierzu werden in Assembler Programme entwickelt, welche die Verwendung des Startup-Codes überflüssig machen[13].

10 Je nach Variante des Evaluierungsboards können Sie ein unterschiedliches Derivat besitzen. Die exakte Bezeichnung finden Sie klein auf dem µC aufgedruckt. In der Regel sind die Derivate des XC888, XC886, XC878 für die im Buch behandelten Themen als identisch anzusehen.

11 Am Ende des Startup-Codes erfolgt ein Sprung an die Sprungmarke ***main***, so dass wie in C üblich mit der ***main***-Routine „begonnen“ werden kann. Details zu Sprungmarken in Assembler werden im weiteren Verlauf des Buches erörtert.

12 Im Simulatorbetrieb lässt sich ein C-Programm ohne Startup-Code ausführen. Auf realer Hardware ist dies jedoch nicht der Fall.

13 Genauer gesagt stören sich die erstellten Assemblerprogramme und der Startup-Code gegenseitig, da sie auf denselben Speicherbereich zugreifen.

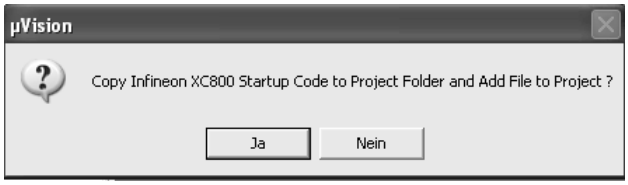

Abbildung 8: Die Integration des XC800 Startup-Codes ist in C notwendig.

In einem weiteren Schritt wird eine Datei erstellt, die den eigentlichen Programmcode beinhaltet. Über *File → New* erscheint eine neue Datei im rechten Anzeigefenster, welche zuerst einmal mit der richtigen Endung abgespeichert werden muss. Im Fall eines C-Programms ist dies die Endung *.c*, im Falle von Assembler *.asm*.

Anschließend muss dem Projekt mitgeteilt werden, dass die Datei dem Projekt zugehörig sein soll. Dies erfolgt über einen rechten Mausklick auf *Source Group 1* im Projektfenster links und anschließend über die Auswahl *Add Files to Source Group 1*. Nach erfolgreicher Einbindung in das Projekt wird diese Datei im Projektfenster angezeigt, siehe Abbildung 9.

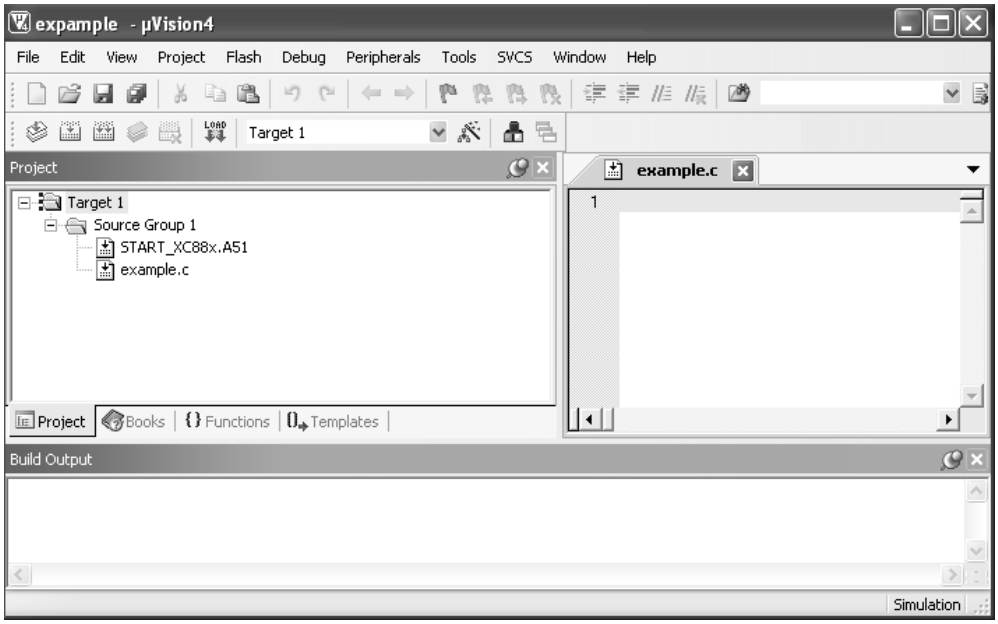

Abbildung 9: Die Datei *example.c* ist in das Projekt eingebunden.

3.3.3 Build und Ausführung eines C-Programms im Simulatorbetrieb

Sowohl in Assembler als auch in C ist es notwendig, ein erstelltes Programm in eine Binärform zu verwandeln. Dies heißt, der Text der Datei muss in eine für den Prozessor lesbare Form übersetzt werden. Dieser Vorgang wird allgemein als *Build* eines Programms bezeichnet. In Assembler wird der hierzu notwendige Übersetzer ebenfalls Assembler genannt. In C ist diesem Assembler ein Compiler vorgeschaltet, welcher den C-Code in einem ersten Schritt in Assemblercode transferiert.

Das Anstoßen des Builds erfolgt über *Project* → *Build* beziehungsweise *Project* → *Build All Target Files* oder über die entsprechenden Symbole in der Programmleiste. Im unten platzierten *Build Output*-Fenster kann verifiziert werden, ob bei der Übersetzug des Programms Fehler aufgetreten sind.

Das einfachste Programm in C stellt eine leere `main`-Routine dar und in der Tat zeigt Abbildung 10, dass die Übersetzung dieses Programms problemlos gelingt. Somit kann an dieser Stelle nun näher auf konkrete Beispiele eingegangen werden. So soll in dem folgenden Programm der Eingabe-/Ausgabeport P3 *getoggelt* werden[14]. Port P3 ist hierbei auf dem Evaluierungsboard mit der LED-Leiste verdrahtet, siehe Abschnitt 3.5.

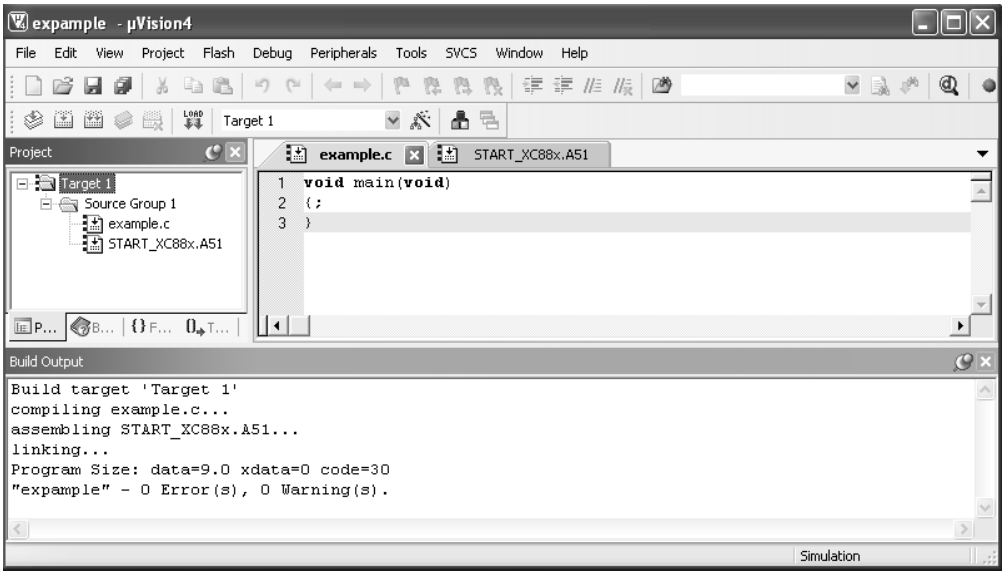

Abbildung 10: Erfolgreicher Build einer leeren ***main***-Routine.

In einem ersten Schritt wird Port P3 als Ausgang konfiguriert mit Hilfe des Registers `P3_DIR`. In einer weiteren Endlosschleife kann dieser Port anschließend seine Werte togglen lassen über das Eingabe-/Ausgaberegister `P3_DATA`. Der zugehörige C-Code lautet:

```
/*************************************************************
                    Programmbeschreibung
 * Autor: Reiner Kriesten
 * Datei: example.c
 * Beschreibung: Togglen von Port P3
*************************************************************/
#include <XC888CLM.H>
void main(void)
{
      P3_DIR=0xFF; // P3 ist Ausgang
```

14 *Togglen* bezeichnet das zyklische Setzen und Löschen einzelner oder mehrerer Bits.

```
        while(1)
        {
                P3_DATA = ~ P3_DATA; //Togglen P3_DATA
        }
}
```

Eine nähere Betrachtung verdient das `#include`-Statement. Die in der `main`-Routine verwendeten Register `P3_DATA` und `P3_DIR` besitzen als Richtungsregister beziehungsweise Eingabe-/Ausgaberegister spezielle Funktionalitäten und werden deshalb auch als *SFR* (Special Function Register) bezeichnet. Um diese Funktionalitäten zu gewährleisten, müssen die Register an festen Adressen residieren[15]. Es ist aber der IDE zu Beginn eines Projekts nicht bekannt, welcher Name `P3_DATA`, `P3_DIR` denn welche Adresse darstellt. Die Zuordnung von Name zu Adresse ist folglich bereitzustellen und dies erfolgt in der Datei *XC888CLM.H*, welche in gewohnter Manier inkludiert wird. Abbildung 11 zeigt die erfolgreiche Erstellung des Beispielcodes.

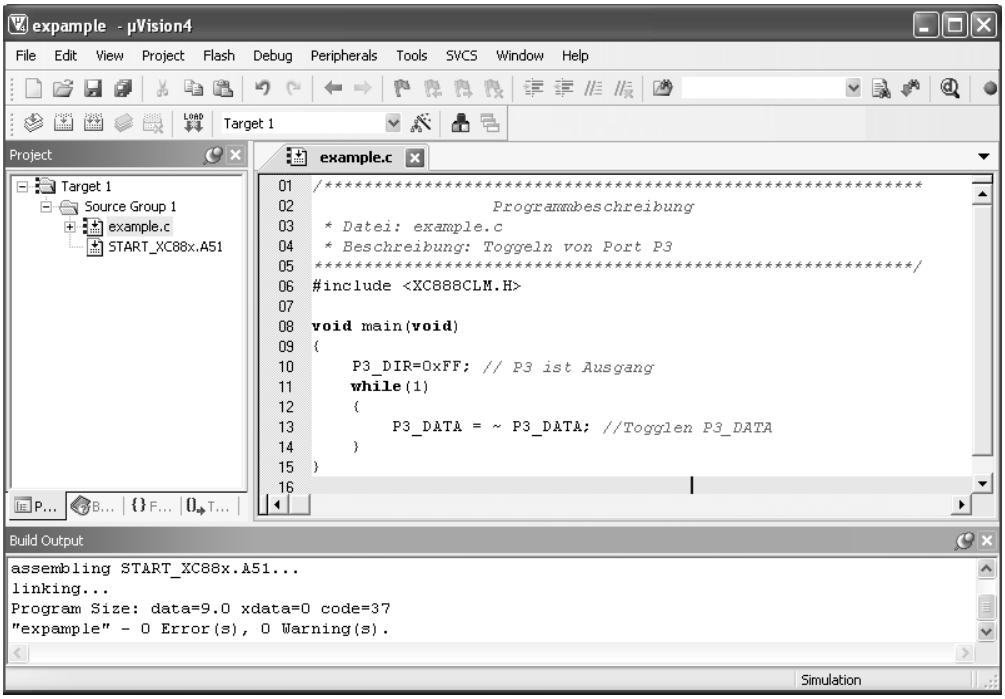

Abbildung 11: Erfolgreiche Kompilierung des Programms zum Togglen von Port P3.

Nach dem Build des Programms ist es möglich, dieses im Simulatorbetrieb zu verifizieren. Im Reiter *Debug*, welcher per rechten Mausklick auf *Target 1* → *Options for Target „Target 1“* im Projektfenster links erreicht werden kann, wird zwischen Simulatorbetrieb und Betrieb auf realer Zielhardware unterschieden, siehe Abbildung 12. Hierbei bietet es sich an, das

[15] Bei „gewöhnlichen“ Variablen existiert hingegen eine gewisse Freiheit in der Platzierung auf dem Datenspeicher.

Häkchen *Limit Speed to Real-Time* zu setzen. Auf diese Weise wird das Programm auf dem Simulator mit derselben Taktrate ausgeführt, wie es auf der realen Zielhardware der Fall ist. Beim *Steppen* durch das Programm sind die realen Zeiten des Zielprozessors unten rechts im Hauptfenster zu sehen und somit können Timings wie die einer Blinkfrequenz verifiziert werden.

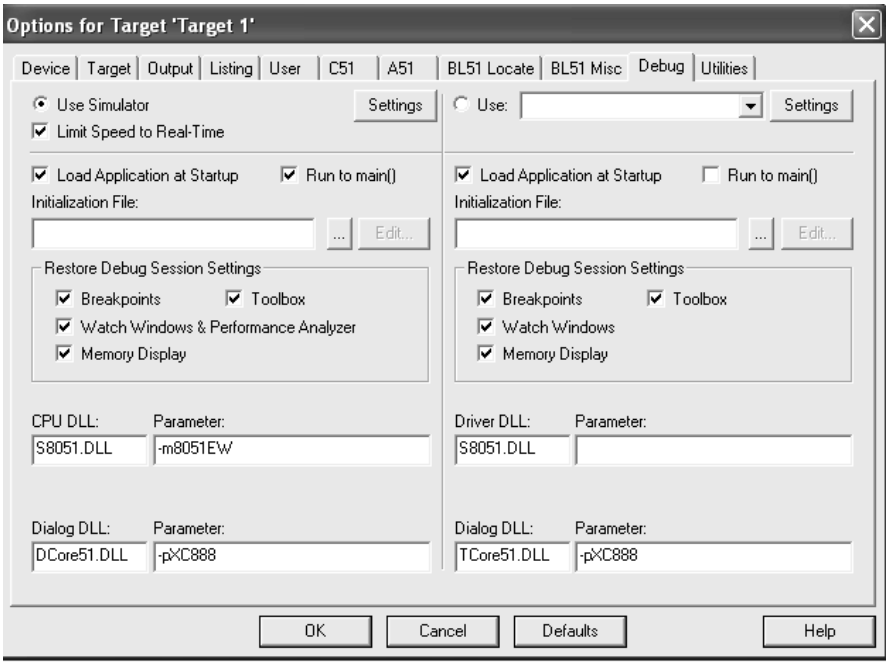

Abbildung 12: Konfiguration des Simulatorbetriebs.

Die Ausführung des Programms erfolgt in einer Debug-Session über *Debug* → *Start/Stop Debug Session* oder Drücken des entsprechenden Symbols. Beachtenswert ist hierbei, dass das Programm nicht instantan ausgeführt wird. Vielmehr wird das Programm „scharfgeschaltet", das heißt es wartet *vor* dem ersten auszuführenden Befehl auf weitere Aktionen[16]. Das „Warten" an einer bestimmten Anweisung wird über den gelben Pfeil *vor* der jeweiligen Anweisung dargestellt, siehe Abbildung 13.

Die Ausführung des Codes kann jetzt in unterschiedlicher Art und Weise erfolgen:

- Das Programm kann „am Stück" ausgeführt werden. Durch *Debug* → *Run* wird dieser Mode gewählt und es ist am Bildschirm nicht nachverfolgbar, an welcher Stelle des Codes sich die Abarbeitung befindet. Nichtsdestotrotz kann über den Run-Mode das Verhalten des Programms in seiner Gesamtheit nachempfunden werden. Gestoppt werden kann der Mode über den Befehl *Debug* → *Stop*.
- Unter dem Reiter *Peripherals* können Peripherie-Einheiten des µCs in einem grafischen Fenster angezeigt werden und diese werden dynamisch während des Programmablaufs

16 Das Warten erfolgt, indem der Befehlszähler auf den entsprechenden Programmcode gesetzt wird und die *Debug-Unit* des µCs die Ausführung dieses Befehls kontrolliert.

aktualisiert. In dem gegebenen Beispiel sollte Port P3 unter *Peripherals* → *I/O-Ports* aktiviert werden. Abbildung 14 zeigt einen Screenshot zu dem Zeitpunkt, zu dem das Programm den Port gerade aktiv geschaltet hat.

- Soll sukzessive Anweisung für Anweisung analysiert werden, so ist anstelle des Run-Mode der Betrieb im *Step-Mode* respektive *Step-Over-Mode* zu wählen. Bei dem Betrieb im Step-Mode wird jeder Befehl einzeln ausgeführt und anschließend das Programm angehalten. Somit besitzt der Entwickler die Möglichkeit, interessante Werte und Variablen zu verifizieren. Während der Step-Mode auch in Unterfunktionen verzweigt, wird beim Step-Over-Mode ein Funktionsaufruf *en block* ausgeführt.

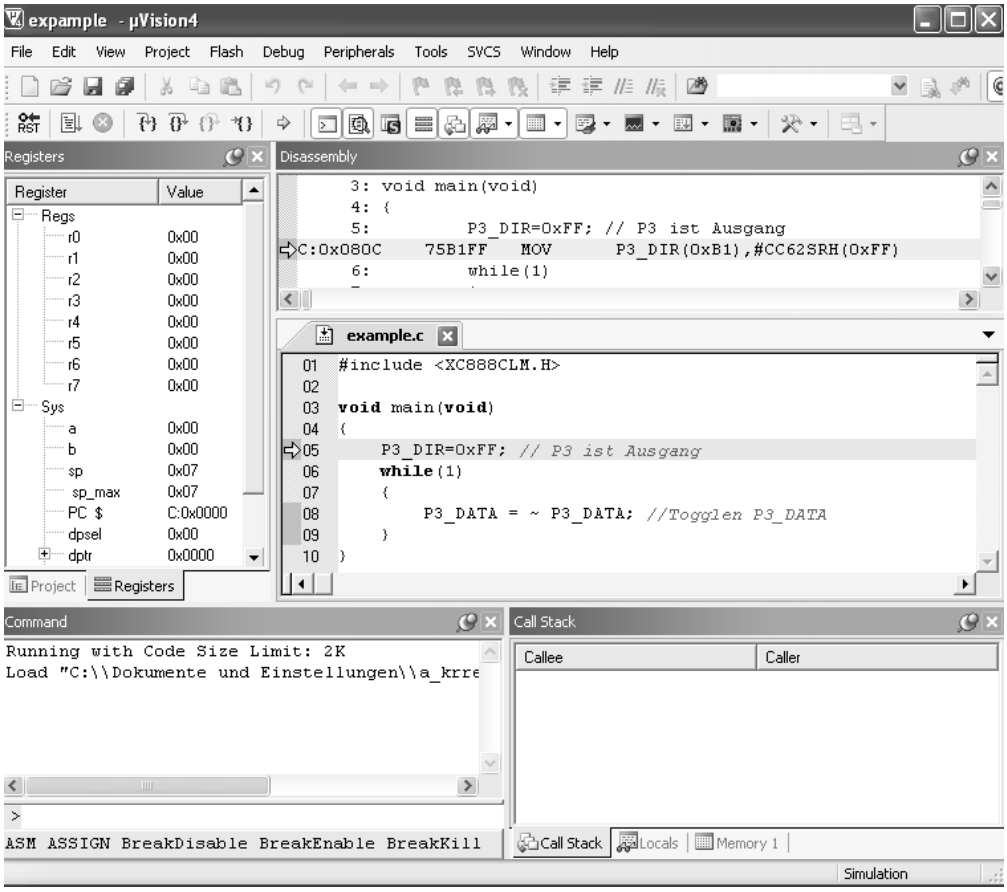

Abbildung 13: Step-Modus im Debug-Betrieb (oben: Assembler-Code, unten: C-Code).

Erwähnenswert ist weiterhin, dass sämtliche Register- und Variablenwerte des µCs während einer Debug-Session angezeigt werden können, also deutlich mehr Informationen als in den Anzeigefenstern der Peripherals dargestellt. Über *View* → *Watch Windows* → *Watch* wird ein Fenster geöffnet, in dem es möglich ist, sämtliche Register und Variablen manuell einzugeben und anzuzeigen, siehe Abbildung 15. Über das *Memory-Window* kann hingegen die Bytebelegung im RAM und ROM abgelesen werden. Beispielsweise zeigt die Eingabe *c:0x00* die Inhalte des Flash-ROM ab Adresse 0x00 an (*c:* steht für Code-Segment), während die

Eingabe von *d:0x80* das RAM ab Adresse 0x80 darstellt (*d:* steht für direkt adressierbares Daten-Segment)[17].

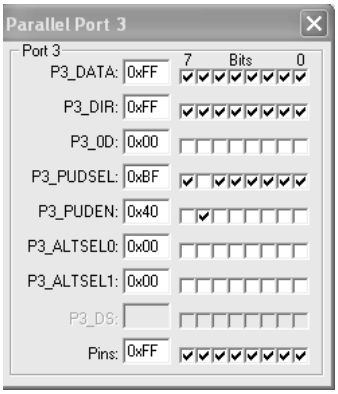

Abbildung 14: Geschalteter Port P3 im Simulations-Mode.

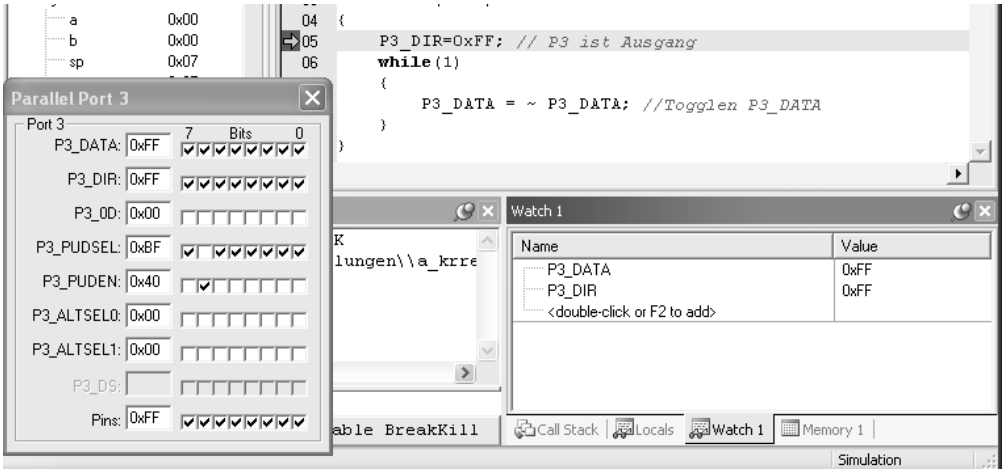

Abbildung 15: Anzeigen der Register über das Watch-Window.

3.3.4 Build und Ausführung eines Assembler-Programms*

Die Erstellung und das Übersetzen eines Assembler-Programms erfolgt analog zur Vorgehensweise in C. In einem ersten Schritt wird eine Assembler-Datei wie *example_ass.asm* mit Endung *.asm* oder *.a51* erstellt und in *dasselbe* Projekt eingebunden[18]. Im Projektfenster ist jetzt anhand des Pfeils in dem Textblatt-Symbol der Dateien *START_XC88x.A51*, *example.c*, *example_ass.asm* zu erkennen, dass sowohl die bisherigen Dateien als auch die neue *.asm*-Datei in das Build inkludiert sind. Dies ist jedoch nicht gewünscht, denn *anstelle* der bisheri-

17 Es ist sogar eine Modifikation der Werte über das Watch-Window und das Memory-Window möglich.

18 Alternativ kann ein neues Projekt angelegt werden und die Assembler-Datei dort integriert werden.

gen Dateien soll die neu erstellte Assembler-Datei exklusiv verwendet werden. Aus diesem Grund wird über *rechte Maustaste* → *example.c* → *Options for File ‚example.c'* → *Include in Target Build* das Flag gelöscht, das für die Integration der Datei in den Build-Prozess verantwortlich ist. Analog wird diese Aktion in einem weiteren Schritt für die Datei *START_XC88x.A51* durchgeführt. Abbildung 16 zeigt das Projekt in einer Konfiguration, in welcher lediglich die Assembler-Datei *example_ass.asm* in das Build einbezogen wird.

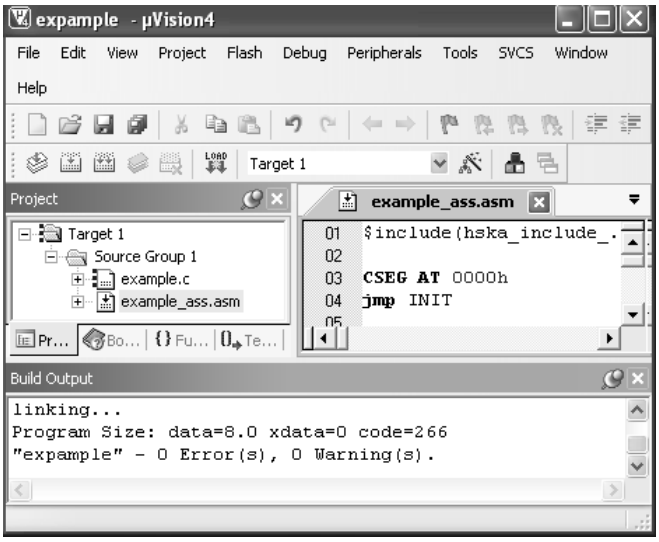

Abbildung 16: Exklusive Verwendung von *example_ass.asm* im Build.

Der Assembler-Code zum Togglen des Ports P3 kann wie folgt gestaltet werden:

```
;*************************************************************
;                          Programmbeschreibung
;* Datei: example_ass.asm
;* Beschreibung: Togglen von Port P3
;*************************************************************
$include(hska_include_.inc)

CSEG AT 0000h
jmp INIT

org 100h
INIT:
mov P3_DIR, #0ffh
MAIN:
mov a, P3_DATA
cpl a
mov P3_DATA, a
jmp MAIN
END
```

Typisch ist, dass das Assembler-Programm aus deutlich mehr Codezeilen besteht wie der vorherige C-Code. Dies ist der Fall, da die Assembler-Sprache hardwarenäher als die Verwendung von C ist und somit ein C-Befehl häufig aus mehreren Assembler-Befehlen zusammengesetzt wird. Zudem gilt, dass im vorhandenen Assembler-Programm mehr Information vorhanden ist als im vergleichbaren C-Code, namentlich die Lage des Codes im Flash-ROM[19].

Auch in einem Assembler-Programm muss eine Zuordnung von Registernamen zu Adressen stattfinden. Während hierfür die Header-Datei *XC888CLM.H* im C-Code verantwortlich ist, wird an dieser Stelle die Datei ***hska_include_.inc*** inkludiert[20], siehe Kapitel 16.1. Auf weitere Erklärungen des Assembler-Codes wird an dieser Stelle verzichtet und auf Abschnitt 4.3 verwiesen.

Zu guter Letzt bleibt zu erwähnen, dass im Debug-Betrieb über das *Disassembly-Fenster* die Nähe von Assembler zur Hardware zu erkennen ist, siehe Abbildung 17. Ein Befehl wie ***jmp INIT*** stellt im Programmcode eine Bitfolge dar, im gegebenen Fall die Bytes 21 00 an den Adressen 0x00 und 0x01 des Flash-ROM. Hierin ist sowohl der eigentliche Sprungbefehl `AJMP` codiert als auch die Sprungadresse 100h. Weitere Details über die Zusammensetzung des Befehls sind unter [INF06] zu finden.

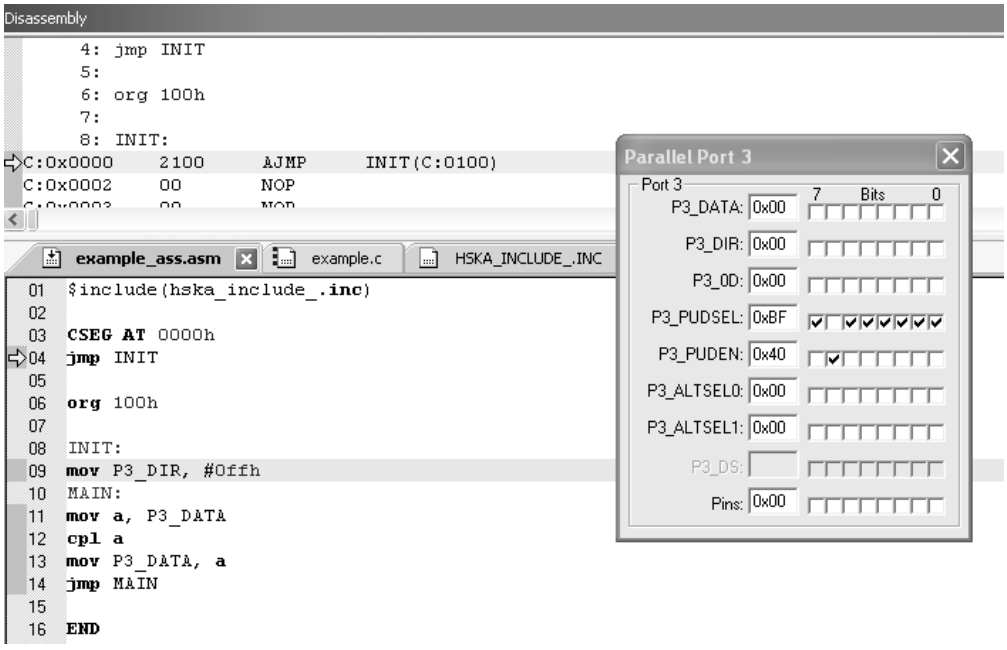

Abbildung 17: Das Disassembly-Fenster zeigt die Nähe zur HW auf.

19 Die Lage des Codes im Speicher wird anhand der Control Statements ***CSEG*** und ***org*** bestimmt. Diese Information existiert selbstverständlich auch in einem C-Build, jedoch nicht direkt in der Programmdatei.

20 Damit die IDE die Datei findet, sollte sie in den Ordner <ROOT>\C51\ASM kopiert werden, also bei einer Standard-Installation in C:\Keil4\C51\ASM.

3.4 Informationen zu der verwendeten Hardware

Die Konfiguration der letzten Abschnitte ermöglicht sowohl die Erstellung eines Programms als auch dessen Ausführung und Verifikation im Simulatorbetrieb. Grundsätzlich verhält sich ein Code in der Simulation und auf realer Hardware identisch – jedoch ist es in einigen Fällen nicht möglich, sich auf die Simulation zu verlassen. Als Beispiel wurde bereits die Inbetriebnahme von Tastern erwähnt, welche während des Drückvorgangs prellen. Auch die korrekte Ansteuerung einer 7-Segment-Anzeige ist lediglich schwer in der Simulation nachzuprüfen, denn das Leuchtbild parallel angeordneter LED der Simulation unterscheidet sich signifikant von der Anordnung der LED auf der realen Anzeige. Diese Gründe sowie die Faszination, die „echte“ Umwelt zu beeinflussen, stellen die Motivation zur Verwendung realer Hardware dar.

Im Fall der Beschaffung von Hardware sollte auf die folgenden Punkte geachtet werden, um die weiter vorgestellten Programme sinnvoll mitverfolgen zu können:

- Der gewählte Zielprozessor sollte von der XC800-Familie von Infineon sein.
- In den vorgestellten Programmen wird ein Set an Sensorik und Aktuatorik verwendet wie verschiedenartig angeschlossene Taster, Potentiometer, LED, 7-Segment-Anzeigen, Servomotoren, …

Die in diesem Buch verwendete Hardware besteht aus einem Evaluierungsboard sowie einer Zusatzplatine, auf welcher sich diverse Sensor- und Aktuator-Bausteine befinden. Je näher eine beschaffte Hardware an der beschriebenen Konfiguration liegt, desto einfacher können die vorgestellten Programme nachverfolgt werden.

Das verwendete Evaluierungsboard *KIT_XC888_SK* ist über Infineon beziehbar[21]. Ein Auszug des Schaltplans findet sich in Anhang 16.4, weitere Informationen liegen dem Starter-Kit bei. Erwähnt sei noch einmal, dass ähnliche Evaluierungsboards der XC800-Familie ebenfalls geeignet sind. Zusätzliche Information zu diesen und weiteren Starter-Kits finden sich unter [INF11b].

Als Sensorik und Aktuatorik ist auf dem verwendeten Evaluierungsboard bereits eine LED-Leiste vorhanden, so dass alle 8 Pins von Port P3 fest mit einer LED verbunden sind, siehe Abbildung 18. Des Weiteren existiert ein Potentiometer, welches mit dem Eingangspin P2.7 fest verdrahtet ist. Diese vorhandene Sensorik und Aktuatorik wird für die folgenden Beispiele nicht ausreichend sein und aus diesem Grund wurde an der Hochschule Karlsruhe eine Zusatzplatine mit weiteren Komponenten entwickelt. Der Schaltplan der Zusatzplatine findet sich in Anhang 0 sowie weitere Informationen in elektronischer Form in [KRI12].

[21] Sehr preisgünstige Mikrocontroller der XC800-Familie sind als USB-Stick erhältlich [INF11f]. Diese besitzen jedoch eine begrenzte Anzahl von herausgeführten Pins und sind somit nur bedingt für die Verfolgung des Buches geeignet.

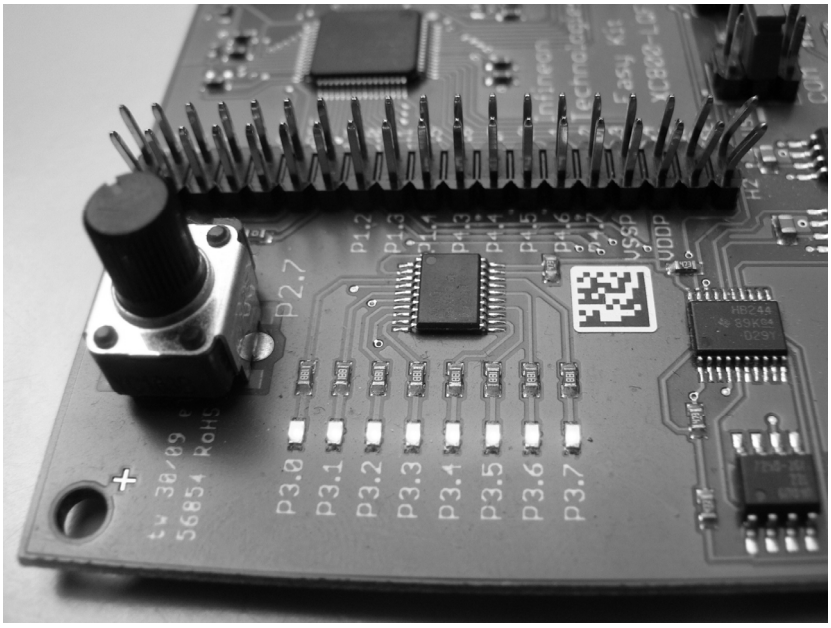

Abbildung 18: Aktive LED und Potenziometer des Evaluierungsboard *KIT_XC888_SK* .

3.5 Inbetriebnahme des Evaluierungsboards

Für die Kommunikation der Entwicklungsumgebung mit dem Evaluierungsboard ist ein weiteres Programm verantwortlich, der *DAS* (Device Access Server) von Infineon. Dieser Device Access Server stellt die Schnittstelle zwischen realer Hardware und IDE dar und sorgt somit dafür, dass der Code auf den Mikrocontroller heruntergeladen wird, die Debugging-Schnittstelle korrekt angesteuert wird, … Beachtenswert ist hierbei, dass der DAS von der IDE aus betrieben werden kann und somit für den Anwender (fast) unsichtbar erscheint. Der Download des DAS findet sich unter [INF11c] sowie typischerweise auf der beiliegenden CD eines Starter-Kits. Besondere Einstellungen sind während der Installationsroutine nicht vorzunehmen.

Zur Inbetriebnahme des DAS wird in einem ersten Schritt das Ziel *Infineon DAS Client for XC800* im Reiter *Debug* gemäß Abbildung 20 ausgewählt[22]. Weiter sind für den Device Access Server die Einstellungen des verwendeten Evaluierungsboards zu konfigurieren. Dies erfolgt durch die Auswahl der Box *Settings*. Wird das verwendete Evaluierungsboard ohne Adapter mitgeliefert, so ist der entsprechende DAS-Server als *JTAG over USB Chip* zu konfigurieren, bei Mitlieferung eines Adapters als *JTAG over USB Box*. Zudem ist darauf zu achten, dass als *USCALE Device* der richtige µC ausgewählt wird, in unserem Fall der XC888. Abbildung 21 fasst diese Einstellungen grafisch zusammen.

[22] Der Reiter *Debug* findet sich unter *Flash* → *Configure Flash Tools* oder aber *rechte Maus auf Target 1* → *Options for Target 'Target 1'*.