Max-Planck-Institut
für
Radioastronomie

Diplomarbeit

Titel:

Entwicklung eines Prototypen einer universellen Mehrkanal DC-Bias-Versorgung mit Mikrocontroller für radioastronomische Empfangssysteme

2009/2010

- **Autor** | **Patrick Neuhalfen**
- **Referent** | **Prof. Dr. rer. nat. Bernd Klein**
- **Korreferent** | **Dipl. Ing. Math. Marc Lob**
- **Betreuer** | **Dipl. Ing. Frank Schäfer**

Titel der Diplomarbeit

Entwicklung eines Prototypen einer universellen Mehrkanal DC-Bias-Versorgung mit Mikrocontroller für radioastronomische Empfangssysteme

Angaben zur Person

Name	**:** Neuhalfen
Vorname	**:** Patrick
Geburtsdatum	**:** 07.05.1983
Fachhochschule	**:** Hochschule Bonn-Rhein-Sieg (Sankt Augustin**)**
Studiengang	**:** Elektrotechnik (Fachbereich 03)
Spezialisierung	**:** Automatisierungstechnik
Matrikel-Nummer	**:** 9005552
Anschrift	**:** Sandkaule 87, 53757 Sankt Augustin
Tel-Festnetz	**:** 02241/ 239445
Tel-Mobil	**:** 0175/ 3734338

Durchführung der Arbeit

Diese wissenschaftliche Arbeit wurde im Max-Planck-Institut für Radioastronomie angefertigt.

Anschrift des Institutes

Auf dem Hügel 69

53121 Bonn

Tel: 0228/ 525/ 0

Betreuung

Erstbetreuer	**:** Prof. Dr. rer. nat. Bernd Klein (FH)
Zweitbetreuer	**:** Dipl. Ing. Math. Marc Lob (FH)
Ansprechpartner	**:** Dipl. Ing. Frank Schäfer (MPI)

Abgabetermin

Datum: Sankt Augustin den ___.___.2010, **Unterschrift:**_____

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Diplomarbeit selbstständig und ohne andere als die angegebenen Hilfsmittel erstellt habe. Die Quellen der direkt oder indirekt übernommenen Textstellen sind ausführlich im Literaturverzeichnis dargelegt. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Bonn, den 28. Februar 2010

_____

Unterschrift

Danksagungen

Mein erster und aufrichtiger Dank gilt meinen Eltern Brigitte und Reimund Neuhalfen. Sie haben mich stets unterstützt und mir dieses Studium ermöglicht.

Ein ganz besonderer Dank geht an meine Freundin Kathrin Magerkurth, die immer für mich da war und stets hinter mir gestanden hat.

Meinen Betreuern Prof. Dr. rer. nat. Bernd Klein, Dipl. Ing. Math. Marc Lob, Dipl. Ing. Reinhard Keller und Dipl. Ing. Frank Schäfer bin ich für die Ermöglichung dieser Arbeit und der Beschäftigung in ihrem Institut ebenfalls sehr dankbar.

Ein besonderes Dankeschön geht an meine Kollegen Peter Lambertz, Dipl. Ing. Thomas Berenz und Dipl. Ing. Sener Türk, die mir jederzeit mit Rat und Tat zur Seite gestanden haben.

Allen Menschen, die sich die Zeit genommen haben, diese Arbeit zu lesen und zu korrigieren steht wohl der größte Dank zu. Ohne sie wäre das Endergebnis in dieser Form nicht möglich gewesen.

Inhaltsverzeichnis

Eidesstattliche Erklärung

Danksagungen

Abbildungsverzeichnis

Abkürzungsverzeichnis

1. Vorwort **Seite 1**

1.1) Hintergrund des Themas 1
1.2) Das Max-Planck-Institut 3
1.3) Effelsberg 5
1.4) Aufbau und Funktionsweise des Radioteleskops 6

2. Einleitung **13**

2.1) Erste Schritte 13
2.2) Was ist ein Mikrocontroller/ Prozessor? 13
2.3) Geschichte der Mikrocontroller 14
2.4) Anwendungsbereiche 17
2.5) Auswahl des Mikrocontrollers 18
2.6) Warum die Firma Mikrochip? 20
2.7) Das Entwicklungsboard 20

3. Ein Mikrocontroller im Detail **25**

3.1) Allgemeines 25
3.2) Der Aufbau 25
3.3) Der Kern 26
3.4) Der Programmspeicher 27
3.5) Der Datenspeicher 28
3.6) Ein-/Ausgabeschnittstellen (Input-/Output-Ports) 28
3.7) Zähler/Zeitgeber (Counter/Timer) 29
3.8) Analoge Signale 31
3.9) Interrupt-System 32
3.10) Komponenten zur Datenübertragung 34
3.11) Bausteine für die Betriebssicherheit 34
3.12) Software-Entwicklung 36

4. Die Entwicklungsumgebung **38**

4.1) Die Programmierung mit MPLAB 38
4.2) Anlegen eines Projektes 39
4.3) Die Arbeitsoberfläche 43
4.4) Das Menü *View* 51
4.5) Breakpoints 53
4.6) Der Simulator 54
4.7) Der In-Circuit-Debugger 57
4.8) Der Programmierer 65
4.9) Texteditor 66

5. Rund um die Programmierung **68**

5.1) Programmieren mit dem In-Circuit-Debugger 69
5.2) Das Programm macht sich auf den Weg 71
5.3) Die Programmiermöglichkeiten in MPLAB 73
5.4) Braucht man eine Header-Datei? 74
5.5) Was sind die Konfigurationsbits? 76
5.6) Weshalb sind die Datenblätter so wichtig? 77
5.7) Durch Kommentare zum Erfolg 78

6. Die Hardware **79**

6.1) Der PIC24FJ64GA002 79
6.2) Der Tiefschlafmodus 82
6.3) Die Spannungsversorgung 83
6.4) Der Master-Clear-Anschluss 84
6.5) Das I²C-Bus-System 84
6.6) Das Digital-Potenziometer 88
6.7) Der Bau des Testboards 91

7. Die Software **93**

7.1) Die Register 93
7.2) Der Quellcode 94

8. Fehler und Schwierigkeiten **106**

8.1) Hardware 106

8.2) Software 106

9. Zusammenfassung **109**

9.1) Ausblick 109

9.2) Schlusswort 109

10. Anlagen **110**

10.1) Literatur- Quellenverzeichnis 110

10.2) DVD 116

Abbildungsverzeichnis

Abbildung 1.1: Max-Planck-Logo und das Institut für Radioastronomie in Bonn 3
Abbildung 1.2: Bild des gesamten Himmels im Radiobereich 4
Abbildung 1.3: Radioteleskop in Effelsberg (1) 5
Abbildung 1.4: Radioteleskop in Effelsberg (2) 5
Abbildung 1.5: Funktionsweise des Radioteleskops in Effelsberg, Wellenstrahlung 6
Abbildung 1.6: Das Wellen-Spektrum 7
Abbildung 1.7: Der Primärfokus 9
Abbildung 1.8: Die Empfängerkabine 10
Abbildung 1.9: Innenansicht der Empfängerkabine 11
Abbildung 1.10: Der Sekundärfokus 11
Abbildung 1.11: Der Sekundärspiegel 12
Abbildung 2.1: Beispiele für Mikrocontroller(1) 13
Abbildung 2.2: Beispiele für Mikrocontroller(2) 14
Abbildung 2.3: Anwendungen von Mikrocontrollern 17
Abbildung 2.4: PIC24FJ64GA002-I/SS Mikrocontroller 20
Abbildung 2.5: Explorer 16 Entwicklungsboard 21
Abbildung 2.6: Fertige PIM-Sockel von Microchip; rechts eigener PIM-Sockel 21
Abbildung 2.7: Zuordnungsplan eigener PIM-Sockel 22
Abbildung 2.8: Eigener PIM-Sockel; Mikrocontroller und Adapterplatine für SMD1 23
Abbildung 2.9: Eigener PIM-Sockel; Mikrocontroller und Adapterplatine für SMD2 23
Abbildung 2.10: Eigener PIM-Sockel; Mikrocontroller und Adapterplatine für SMD3 23
Abbildung 2.11: Explorer 16 Entwicklungsboard mit eigenem Sockel 24
Abbildung 2.12: ICD3 Debugger von Microchip 25
Abbildung 3.1: Blockschaltplan eines Mikrocontrollers eingebettet in eine Anwendung 26
Abbildung 4.1: Project Wizard starten 39
Abbildung 4.2: Project Wizard Welcome 39
Abbildung 4.3: Auswahl des Mikrocontrollers 40
Abbildung 4.4: Auswahl der Toolsuite 41
Abbildung 4.5: Neues Projekt anlegen 41
Abbildung 4.6: Dateien hinzufügen 42
Abbildung 4.7: Zusammenfassung 43
Abbildung 4.8: Projektansicht 44
Abbildung 4.9: Neue Datei 44

Abbildung 4.10: Datei speichern 45
Abbildung 4.11: Dateien zum Projekt hinzufügen 46
Abbildung 4.12: Projektansicht 47
Abbildung 4.13: Auswahl des Tools zum Debuggen 48
Abbildung 4.14: Projekt übersetzen 48
Abbildung 4.15: Erfolgreich übersetzt 49
Abbildung 4.16: Buttons für die Simulation 49
Abbildung 4.17: Das Menü *View* 51
Abbildung 4.18: Watches 51
Abbildung 4.19: Disassembly Listing 52
Abbildung 4.20: Breakpoints 53
Abbildung 4.21: Simulator Settings 54
Abbildung 4.22: Asynchroner Stimulus 55
Abbildung 4.23: Zyklischer Stimulus 56
Abbildung 4.24: Hinzufügen von Signalen 57
Abbildung 4.25: Auswahl des Mikrocontrollers 59
Abbildung 4.26: Auswahl des ICD3 als Debuggtool 60
Abbildung 4.27: Informationen im Ausgabefenster 61
Abbildung 4.28: Status des ICD3 61
Abbildung 4.29: Spannungsversorgung 62
Abbildung 4.30: Programmiereinstellungen 63
Abbildung 4.31: Konfigurationsbits 64
Abbildung 4.32: Einstellungen des Debuggers 64
Abbildung 4.33: Einstellungen für die Programmierung 65
Abbildung 4.34: ICD3 als Programmiergerät 66
Abbildung 4.35: Einstellungen des Texteditors 67
Abbildung 5.1: Die Programmierschnittstelle (1) 69
Abbildung 5.2: Die Programmierschnittstelle (2) 70
Abbildung 5.3: Die Programmierschnittstelle (3) 70
Abbildung 5.4: Der Weg zur Header-Datei (1) 75
Abbildung 5.5: Der Weg zur Header-Datei (2) 75
Abbildung 5.6: Der Weg zur Header-Datei (3) 76
Abbildung 6.1: Blockschaltbild der Controller-Familie PIC24FJ64GB004 80
Abbildung 6.2: Pin-Diagramm des Mikrocontrollers 81
Abbildung 6.3: Festspannungsregelung für das Testboard 83

Abbildung 6.4: Anschluss des Master-Clear-Pins **84**
Abbildung 6.5: Anschluss des Master-Clear-Pins (2) **84**
Abbildung 6.6: Anschluss der Slaves an den Bus, Pull-Up-Widerstände **86**
Abbildung 6.7: I2C-Bus-Protokoll **87**
Abbildung 6.8: I²C Übertragungsprotokoll für die Ansteuerung eines EEProm **88**
Abbildung 6.9: Blockschaltbild des AD5252, Pin-Konfiguration **89**
Abbildung 6.10: Protokoll eines Programmiervorgangs **90**
Abbildung 6.11: Das fertige Testboard **91**
Abbildung 6.12: Die Verdrahtung des Testboards **92**
Abbildung 8.1: Device ID Revision **107**
Abbildung 8.2: Die Registeranzeige im Fenster *Watch* **108**

Abkürzungsverzeichnis:

ABS	Antiblockiersystems
ADU	Analog-Digital Umsetzer
ASC	Universal Synchronous
AVR	Advanced Virtual RISC
CAN	Controller Area Network
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
DC	Direct Current
DAU	Digital-Analog Umsetzer
ESP	Elektronisches Stabilitätsprogramm
EMV	Elektromagnetische Verträglichkeit
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
EVN	Europäisches VLBI-Netzwerk
GaAs	Galliumarsenid
GPS	Global Positioning System
I2C	Inter-Integrated Circuit
IBM	International Business Machines Corporation
IC	Integrated Circuit
ICD	In-Circuit-Debugger
ICSP	In-Circuit-Serial-Programming
JTAG	Joint Test Action Group
MIT	Massachusetts Institute of Technology
MMIC	Monolithic Microwave Integrated Circuits
MPIfR	Max-Planck-Institut für Radioastronomie
OST	Oscillator Start-Up-Timer
OTP	One Time Programmable
PC	Personal Computer
POR	Power-On-Reset
PWM	Pulsweitenmodulation
PWRT	Power-Up-Timer
RAM	Random Access Memory
RF	Radio Frequency

ROM	Read Only Memory
RTCC	Real-Time Clock-Calendar
SCI	Serial Communication Interface
SCL	Serial Clock Line
SDA	Serial Data Line
SFR	Spezial-Function Register
SMD	Surface-Mounted Device
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
USB	Universal Serial Bus
VLBI	Very Long Baseline Interferometry
PIC	Programmable Integrated Circuit
RISC	Reduced Instuction Set Computer
WDT	Watchdog Timer

1. Vorwort

Diese Diplomarbeit wurde zum Abschluss des Studiengangs Elektrotechnik der Fachrichtung Automatisierung angefertigt.
Die Ausarbeitung erfolgte im Max-Planck-Institut für Radioastronomie in Bonn.

Zu Beginn der Diplomarbeit kamen beim Diplomanden Patrick Neuhalfen folgende Gedanken auf:

Das Thema ist hochinteressant, keine Frage, aber…:

Mikrocontroller-Programmierung, geht's noch komplizierter?
Muss man ein Programmiergenie sein?
Wer betreut mich am besten bei diesem Projekt?
Wie eignet man sich das nötige Fachwissen an?
Gibt es gute aktuelle Literatur und/oder Internetseiten?
Wie gehe ich an dieses Projekt heran?
Und was wird noch alles auf mich zukommen?

In den folgenden Kapiteln werden Sie sehr schnell alle Antworten auf diese Fragen finden.
Eines ist ganz klar: zum Verständnis dieser wissenschaftlichen Arbeit ist technisches Grundverständnis und Vorwissen Voraussetzung.
Dem Diplomanden ist aber wichtig, dass nicht nur Experten verstehen, wovon er redet.
Es sollen auch Laien von Grund auf an das Thema Mikrocontroller herangeführt werden; andererseits sollen Experten nicht mit zu ausführlichen Grundlangen gelangweilt werden.

Die Arbeit soll jedem Leser einen ausführlichen Leitfaden zur Mikrocontroller-Thematik bieten und somit vielleicht sogar für ein eigenes Projekt motivieren.

1.1 Hintergrund des Themas

<u>Thema</u>: Entwicklung eines Prototypen einer universellen DC-Bias-Versorgung.

Radioastronomische Empfangssysteme benötigen eine Vielzahl von einzelnen Mikrowellen-Komponenten, wie z.B. Verstärker, Mischer, Multiplier, Schalter und Abschwächer, um die

jeweils gewünschten Funktionen des Empfängers zu realisieren. Diese Komponenten arbeiten üblicherweise bei Raumtemperatur und sind klassisch als komplette (kommerzielle) Komponenten mit Versorgung über eine einzelne Spannung (5 oder 15V) ausgeführt.
Mittlerweile sind von verschiedenen Firmen komplette Funktionsblöcke als integrierte Mikrowellen-Schaltungen auf GaAs Basis (MMICs) kommerziell erhältlich. Somit können die für ein Empfangssystem nötigen Funktionen vorteilhaft in kompakten RF-Modulen realisiert werden, dies erlaubt höhere Integration und weitgehende Vermeidung der fehleranfälligen koaxialen Verkabelung von Einzelkomponenten in semi-rigid Technik oder mechanisch komplexe Hohlleiteraufbauten. Die verwendeten MMICs enthalten nun aber keine Elektronik zur Versorgung mehr, sie benötigen meist mehrere DC-Spannungen, die beim Einschalten auch noch korrekt sequenziert werden müssen. Daher soll ein universelles DC-Versorgungsmodul entwickelt werden, welches am oder in unmittelbarer Nähe des RF-Moduls angebracht wird und die nötigen Versorgungsspannungen aus einer einzelnen Versorgungsspannung erzeugt und sequenziert. Dieses Modul muss folgende Anforderungen erfüllen:
Kompakter Aufbau, Unterbringung möglichst in einer extra Kammer des RF-Moduls.
Der Mikroprozessor startet bei Anlegen der Betriebsspannung ein Programm, welches die für das Modul nötigen Spannungen in der richtigen Reihenfolge anlegt und ein kontrolliertes Hochrampen erlaubt.
Prototyp: 4 Drain-Ausgänge bis 50mA, 4 Gate-Ausgänge bis 1mA
Um Störungen der RF-Schaltung, die über die Bias-Versorgung übertragene Digitalsignale entstehen, zu vermeiden, soll der Mikrocontroller nach Abarbeitung der Einschaltroutine so viele Funktionen wie nur möglich abschalten.
Die Spannungen müssen nach der Abschaltung weiterhin erhalten bleiben.
Die Sollwerte von Spannungen, Rampenzeiten oder Sequenzzeiten werden im nichtflüchtigen Speicher des Mikros abgelegt. Zur Justage des RF-Moduls nach dessen Aufbau müssen diese Werte programmierbar sein. Dazu kann der Mikrocontroller über DIP-Schalter o.ä. “aufgeweckt“ werden, er soll dann ein zweites Programm abarbeiten, welches die Programmierung dieser Werte im nichtflüchtigen Speicher erlaubt. Dazu wird ein externer Rechner z.B. über die serielle Schnittstelle des Mikros mit diesem verbunden, ein externes Programm auf diesem Rechner erlaubt dann das Einstellen der Spannungen über Werteingabe oder Regler am Bildschirm. Nach Abschluss der Justage werden die (neuen) Werte dann im nichtflüchtigen Speicher des Mikrocontrollers abgelegt und die Verbindung zu dem externen Rechner getrennt.

An dieser Stelle ist noch zu erwähnen, dass bei uns im Institut, aufgrund der EMV Problematik durch digitale Signale, noch keine Mikroprozessoren in Empfängern eingesetzt wurden. Deswegen handelt es sich hier um ein Forschungsprojekt, in dem es rauszufinden gilt, ob die Theorie des Projektes auch tatsächlich in der Praxis eingesetzt werden kann.

1.2 Das Max-Planck-Institut

Das Max-Planck-Institut für Radioastronomie gehört zu den 80 eigenständigen Forschungsinstituten der Max-Planck-Gesellschaft zur Förderung der Wissenschaften e.V.

MAX-PLANCK-GESELLSCHAFT

Abbildung 1.1: Max-Planck-Logo und das Institut für Radioastronomie in Bonn [3]

Den Geheimnissen des Universums auf der Spur

Das Max-Planck-Institut für Radioastronomie (kurz: MPIfR) in Bonn widmet sich der Erforschung astronomischer Objekte mittels Radiowellen und Infrarotwellen. Ergänzende Untersuchungen in den anderen Wellenlängenbereichen (optische, Röntgen- und Gamma-

Strahlung sowie Astro-Teilchen) werden ebenfalls vorgenommen. Die technologischen Entwicklungen im Institut umspannen den gesamten Beobachtungsbereich. Auch die theoretische Astrophysik ist ein weiteres Arbeitsgebiet.

Die Erforschung der Physik von Sternen, Galaxien und des Universums beinhaltet als Schwerpunkte:

- Die Sonne und andere Körper im Sonnensystem
- (Radio-)Sterne, Sternentstehungsgebiete, Supernova-Überreste und Pulsare
- Das Interstellare Medium der Milchstraße und externer Galaxien
- Interstellare Gase und Gasnebel
- Das Galaktische Zentrum und seine Umgebung
- Magnetfelder im Universum
- Radiogalaxien
- Quasare und andere aktive Galaxien
- Galaxien in den Frühphasen des Universums
- Kosmische Strahlung
- Hochenergieteilchenphysik
- Planetoide und Asteroide

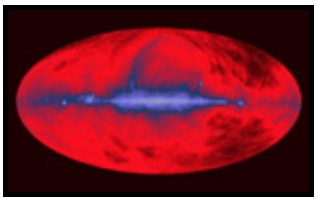


Abbildung 1.2: Bild des gesamten Himmels im Radiobereich [3]

In internationalen Kooperationen ist das Institut mit seinen drei Forschungsgruppen den Rätseln des Entstehens unseres Universums auf der Spur. Hierbei kommen die weltweit größten und wichtigsten Empfangsstationen für Radiosignale aus dem All zum Einsatz.
Das Max-Planck-Institut für Radioastronomie kooperiert in zahlreichen nationalen und internationalen Partnerschaften mit anderen führenden Forschungsinstituten und Organisationen auf dem Gebiet der Radioastronomie. In verschiedenen Großprojekten werden wissenschaftlich anspruchsvolle und hochinteressante Fragestellungen des Universums erforscht. Gemeinsam mit den Partnern werden neue Modelle für Radioteleskope konzipiert und realisiert. Derzeit sind insgesamt 183 Mitarbeiter am Institut für Radioastronomie tätig, darunter

61 Wissenschaftler und 30 Nachwuchswissenschaftler; hinzu kommen im Berichtsjahr 11 Drittmittelbeschäftigte und 8 Gastwissenschaftler. (http://www.mpifr-bonn.mpg.de/)

1.3 Radioteleskop Effelsberg

Das Radioteleskop Effelsberg stellt das Hauptbeobachtungsinstrument des Max-Planck-Instituts für Radioastronomie dar. Mit einem Paraboloid - Reflektor von 100 m Durchmesser und einer Höhe von ebenfalls 100 Metern gehört das Radioteleskop in Effelsberg zu den größten schwenkbaren Radioteleskopen der Welt. [3]

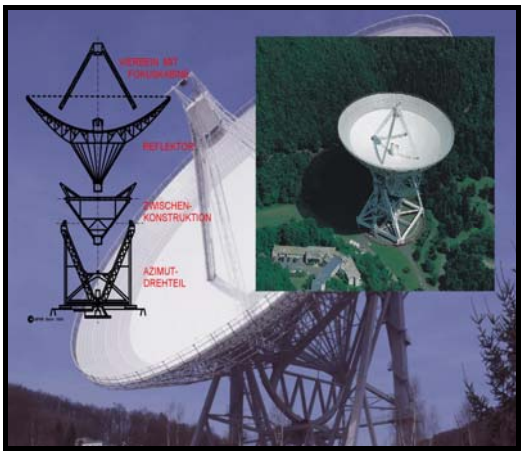


Abbildung 1.3: Radioteleskop in Effelsberg (1) [3]

Abbildung 1.4: Radioteleskop in Effelsberg (2) [3]

Seit der Inbetriebnahme im Jahre 1972 wurde kontinuierlich an der Verbesserung seiner Technologie gearbeitet (z.B. eine neue Oberfläche der Antennen-Schüssel, bessere Empfänger für hochqualitative Daten, extrem rauscharme Elektronik), so dass es auch heute noch als eins der weltweit modernsten Teleskope gilt. [3]

1.4 Aufbau und Funktionsweise des Radioteleskops

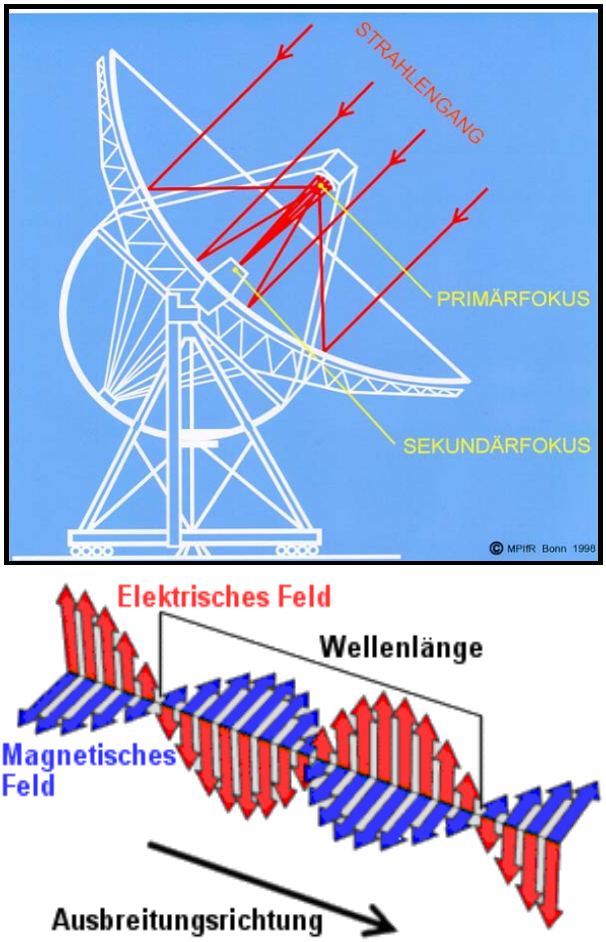


Abbildung 1.5: Funktionsweise des Radioteleskops in Effelsberg; Wellenstrahlung [3]

Je nach Frequenz (Schwingungen pro Sekunde) kann elektromagnetische Strahlung in unterschiedlichen Erscheinungsformen beobachtet werden. Das elektromagnetische Spektrum erstreckt sich von den niederfrequenten Radiowellen über das sichtbare Licht bis zur hochfrequenten Gammastrahlung. Dabei ist die Energie der Strahlung proportional zur Frequenz. *Abb. 1.5* zeigt eine Veranschaulichung der Elektromagnetischen Wellenstrahlung.

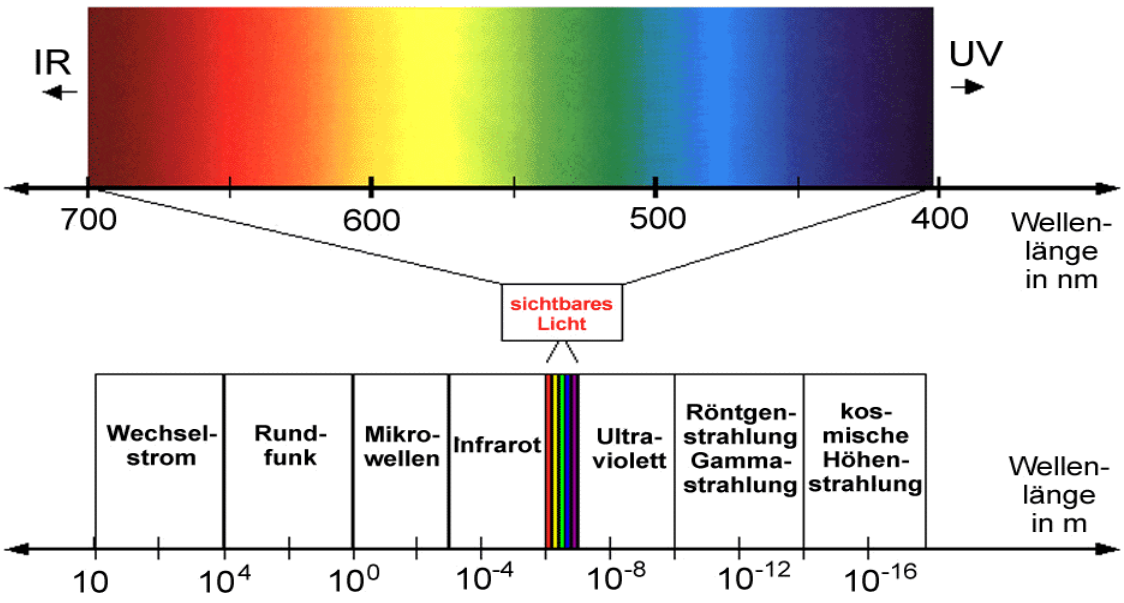


Abbildung 1.6: Das Wellen-Spektrum

Infarot (IR), Ultraviolett (UV)

Die Umrechnung der Wellenlänge in eine Frequenz *f* erfolgt mit der einfachen Formel **f = c/λ**, also Lichtgeschwindigkeit (im jeweiligen Medium) geteilt durch die Wellenlänge. Den Erkenntnissen der Quantenmechanik folgend, bestehen elektromagnetische Wellen aus einem Strom von Teilchen, den Photonen. Diese Betrachtung dient zur Erklärung verschiedener physikalischer Phänomene wie dem photoelektrischen Effekt. Photonen besitzen eine zur Frequenz proportionale Energie. (Müller, 2009, 16f.) [3]

E = h*f Die Konstante **h** ist dabei das plancksche Wirkungsquantum.

Die Energie kann in Joule (J) oder in Elektronenvolt (eV) angegeben werden. Dabei ist **c** die konstante Lichtgeschwindigkeit von 299.792,458 km pro Sekunde.

Zwischen Wellenlänge und Frequenz, sowie der Energie gelten die folgenden Beziehungen:

$\lambda = c / f$ $E = h \cdot c / \lambda$ $E = h \cdot f$

(Einstein´sche Regel)

Einheiten:

E: Energie

h: Planck´sches Wirkungsquantum

F: Frequenz

λ: Wellenlänge

c: Lichtgeschwindigkeit

Das physikalische Konzept der elektrischen und magnetischen Feldlinien, sowie der gegenseitigen Induktion elektrischer und magnetischer Felder wurde 1831 von dem englischen Chemophysiker Michael Faraday aufgestellt und experimentell bewiesen. 1873 entwickelte der schottische Physiker James Clerk Maxwell aus diesen Beobachtungen die nach ihm benannten Gleichungen. Die Maxwell'schen Gleichungen beschreiben Licht als oszillierendes elektromagnetisches Feld und erlauben die theoretische Berechnung von dessen Ausbreitungsgeschwindigkeit (Lichtgeschwindigkeit). Der deutsche Physiker Heinrich Hertz konnte schließlich 1888 die von Maxwell vorhergesagten Radiowellen künstlich erzeugen und nachweisen. Zu Ehren von Hertz, dessen Experimente die Grundlagen der drahtlosen Telegraphie und der Rundfunktechnik bildeten, erhielt die physikalische Einheit der Frequenz seinen Namen (1 Hertz = 1 Schwingung pro Sekunde).

Die mit Hilfe der Einstein´schen Regel berechnete Energie E ist die Energie eines Strahlenquants abhängig von der Frequenz bzw. Wellenlänge. Als Quantisierung bezeichnet man die Teilung einer Größe (z.B. Energie) in “sehr kleine Pakete“. Bezogen auf die elektromagnetische Wellenstrahlung und deren Quantisierung, nennt man diese “sehr kleinen Pakete“ Photonen.

Nach Planck ist jede Strahlung (auch die Lichtstrahlung) aus Energiequanten zusammengesetzt. Die Strahlungsenergie ist also stets ein ganzzahliges Vielfaches der Energie eines Strahlenquants.

Die Intensität der aus dem Weltraum empfangenen Strahlung ist, wie bereits beschrieben, sehr gering. Aus diesem Grund wird ein möglichst großer Parabolspiegel angestrebt, um die

Leistung des Eingangssignals zu vergrößern. Im Folgenden nun der physikalische Zusammenhang zwischen der Leistung **P**, der Intensität **I** (Leistung pro Fläche) und der Fläche **A**:

P = I · A mit **P**: Leistung, **I**: Intensität und **A**: Fläche

Aus dieser Gleichung geht hervor, dass die Fläche **A** proportional der Intensität **I** ist. Aus diesem Grund wird eine möglichst große Fläche des Reflektors angestrebt. Allerdings ist der Bau eines Radioteleskops mit einer Größe von ca. 100 Metern Durchmesser aus Statik- und Festigkeitsgründen nicht oder nur mit sehr hohem Kostenaufwand möglich. Man daher dazu über, Teleskope und Forschungszentren international zu vernetzen. Somit stellt man aus mehreren kleineren Teleskopen ein großes Teleskop her. Hierbei wird die Lichtleitertechnik, sowie Elektronik zur Kompensation der Zeitverschiebung und des daraus resultierenden Phasenversatzes eingesetzt. Diese neuartige Forschungsmethode wird als Radiointerferometrie mit sehr großen Basislängen bezeichnet (VLBI). Es existiert sowohl ein europäisches VLBI-Netzwerk (EVN), aber auch die Möglichkeit, das europäische VLBI-Netzwerk mit Teleskopen aus den USA zu koppeln. (Müller, 2009, S18) [3]

Empfangssystem

Das 100 m Radioteleskop Effelsberg hat die Möglichkeit zwei Brennpunkte zu benutzen, den Primär- und den Sekundärfokus. Die als ebene Wellen ankommenden Signale werden nun zunächst vom Primärspiegel in dessen Brennpunkt gebündelt. Hier befindet sich der Primärfokus.

Der Primärfokus

Abbildung 1.7: Der Primärfokus [3]

30 m vor dem Parabolspiegel befindet sich die Empfängerkabine des Primärfokus. Diese ist mit 4 Beinen am Primärspiegel befestigt, wovon eines begehbar ist. In der Empfängerkabine befinden sich mehrere Empfängerboxen, die jeweils einen Empfänger enthalten. Weiterhin findet man in der Empfängerkabine Einrichtungen vor, die Messungen direkt vor Ort möglich machen. In einer kreisrunden Öffnung im Boden der Empfängerkabine wird die Empfängerbox eingeschoben und im Brennpunkt des Primärspiegels fixiert. Da zur Messung verschiedener Wellenlängen auch verschiedene Empfänger benötigt werden, ist es möglich die Empfängerboxen zu wechseln. Neuerdings existieren auch Empfängerboxen, die mehrere Empfänger für unterschiedliche Wellenlängen enthalten. Somit können Ausfallzeiten bedingt durch Umbaumaßnahmen verringert werden. Allerdings können keine Messungen unterschiedlicher Wellenlängen gleichzeitig durchgeführt werden. (Müller, 2009, S19) [3]

Die Empfängerkabine

Abbildung 1.8: Die Empfängerkabine[3]

Wie bereits im Abschnitt “Primärfokus“ erläutert wurde, befinden sich neben den “Vor-Ort-Messsystemen“ mehrere Empfängerboxen in der Empfängerkabine. Diese Empfänger sind in sich geschlossene Systeme und funktionieren nach dem Trommelrevolver-Prinzip. Soll eine “Patrone“ gewechselt werden, muss die alte “Patrone“ dem “Lauf“ entnommen werden und anschließend die neue “Patrone“ dem “Lauf“ zugeführt werden, welche im vorliegenden Fall zusätzlich elektrisch mit dem Teleskop verbunden und eingestellt werden muss. Es sind allerdings keine weiteren Umbaumaßnahmen notwendig.

Abbildung 1.9: Innenansicht der Empfängerkabine [3]

Die Empfängerboxen enthalten neben der Elektronik den Kryostaten (auch Dewar genannt), ein oder mehrere Empfangshörner, welche die "Antennen" des Empfängers darstellen. Der Dewar oder Kryostat ist eine Kältemaschine, auf deren spezielle Funktion an dieser Stelle nicht weiter eingegangen wird. (Müller, 2009, S20) [3]

Der Sekundärfokus

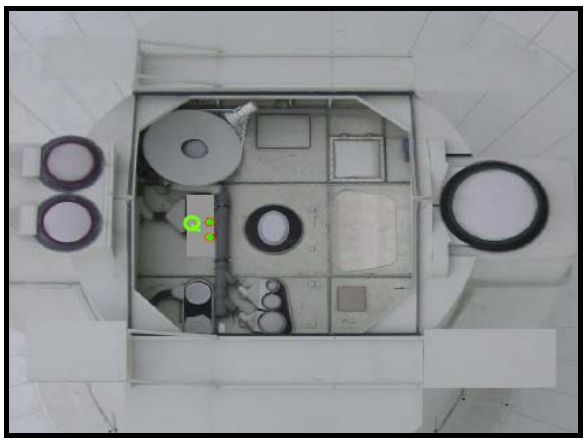


Abbildung 1.10: Der Sekundärfokus [3]

Durch einen weiteren konkaven Spiegel im Primärfokus des Hauptspiegels wird die elektromagnetische Wellenstrahlung umgelenkt in den sogenannten Sekundärfokus. Dieser befindet sich in der Mitte des Primärspiegels. Hier sind weitere Empfänger angeordnet, die, nachdem die Empfänger aus dem Primärfokus und die Öffnung im Sekundärspiegel geschlossen wurde, in Betrieb genommen werden können. Ein gleichzeitiges Betreiben von Primär- und Sekundärfokus ist nicht möglich. Die im Brennpunkt des Primärspiegels gebündelten Signale

werden hier durch den 6,5m großen, elliptischen Sekundärspiegel (Gregory-Spiegel) nochmals im Sekundärfokus gebündelt, wo weitere Empfängereinheiten in den Fokus gebracht werden können. Die neue vollvariable Oberfläche des Sekundärspiegels erlaubt eine noch genauere Bündelung der Signale und den Ausgleich von Unebenheiten im Primärspiegel, sowie Abweichungen bedingt durch Verformungen in Folge von Gewichtskräften.

Abbildung 1.11: Der Sekundärspiegel [3]

Die geplanten Module zur Bias-Versorgung, für die in der Forschungsarbeit einen Prototyp entwickelt wurde, befinden sich in einem Empfangssystem im Sekundärfokus. Wie in *Abb.1.10* zu sehen ist, sind im Sekundärfokus mehrere Empfangssysteme fest verbaut. Anders als im Primärfokus müssen diese somit nicht gewechselt werden.
Insgesamt sind zurzeit 7 Empfangssysteme im Sekundärfokus verbaut, von denen 3 Systeme parallel betrieben werden können. Somit ist dieser in der Lage, Messungen
in mehreren Wellenlängenbereichen durchzuführen. Die Empfänger können Wellenlängen von 1,3 mm bis 21 cm aufnehmen. Die dreistöckige und fast 10 Meter hohe Fokuskabine ist gleichzeitig der Zugang zum Hauptspiegel. (Müller, 2009, S22) [3]

Max-Planck-Institut
für Radioastronomie

Für weitere Informationen besuchen Sie **www.mpifr-bonn.mpg.de**

2. Einleitung

2.1 Erste Schritte

In den ersten Wochen seiner Arbeit war der Diplomand auf der Suche nach guter Literatur. Er suchte ein modernes Buch, das die Vorgehensweise an ein Mikrocontroller-Projekt beschreibt. Es sollte alle nötigen Grundlagen einfach vermitteln, von der Auswahl des Prozessors über der Programmierung in *C* bis hin zum Einsatz des Controllers in einer realen Umgebung. Leider musste der Diplomand feststellen, dass nichts nach seiner Vorstellung zu finden war. Es sind viele alte Bücher vorhanden, die die Programmierung von 4- oder 8-Bit-Controllern in Assembler vermitteln wollen. Außerdem gibt es Bücher, die überwiegend hochkomplizierte Anwendungen und Fachausdrücke enthalten.
Im Internet gibt es Foren, die bei spezifischen Problemstellungen helfen können, allerdings sind ein fortgeschrittenes Basiswissen und Praxiserfahrung vorausgesetzt.

Im Umfang einer Diplomarbeit kann diese Lücke möglicherweise nicht geschlossen werden.

Doch genau dort möchte der Diplomand ansetzen. Sein Ziel ist es, auch dem, der sich neu in die Thematik Mikrocontroller einarbeiten möchte, einen einfachen Einstieg zu bieten.
Das Wissen, welches sich der Diplomand von Grund auf angeeignet hat, wird er so einfach wie möglich versuchen weiterzugeben.

2.2 Was ist ein Mikrocontroller/Prozessor?

So können Mikrocontroller aussehen:

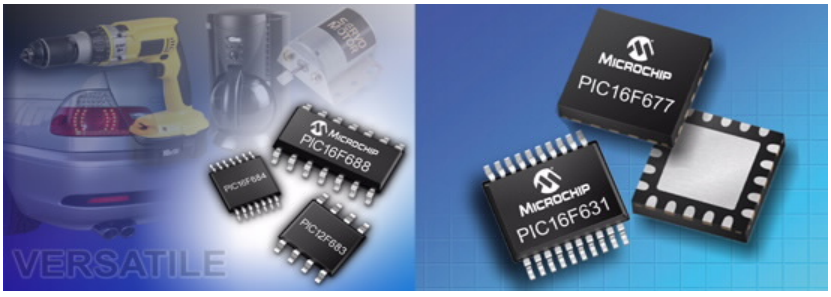


Abbildung 2.1: Beispiele für Mikrocontroller (1) [1]

Abbildung 2.2: Beispiele für Mikrocontroller (2) [1]

Sie sind je nach Familie und Bauform unterschiedlich groß. Es gibt Mikrocontroller von einigen Zentimetern und es gibt welche, die nur einige Millimeter groß sind.
Auf den ersten Blick sind nur Plastik und Metallbeinchen zu erkennen, der erste Eindruck täuscht aber gewaltig.

2.3 Die Geschichte der Mikrocontroller/Prozessoren

Am Anfang stand die Frage, ob die geschichtliche Entwicklung der Mikroprozessortechnik in der Arbeit aufgegriffen werden soll. Die Entscheidung fiel aus folgendem Grund positiv aus:
Es ist einfacher, sich ein besseres Gesamtbild über die heutige Technik der Mikrocontroller zu machen. Außerdem wird kurz der heutige Stand der Technik und dessen mögliche Zukunft erläutert. Da man dieses Wissen nicht zwingend benötigt wird, um sich in die Materie der Mikrocontroller einzuarbeiten, können Eilige diesen Teil auch überspringen und bei *Abschnitt 2.4* wieder einsteigen.

Im folgenden Abschnitt werden nur einige markante Punkte aus der Geschichte der Mikroprozessoren/Controller aufgegriffen. Für ausführlichere Informationen zum Thema Geschichte und Entwicklung der Mikroprozessoren, wird das Buch "*Mikroprozessortechnik*" von Klaus Wüst empfohlen.

Die Geschichte der Computer beginnt lange vor der Geschichte der Mikroprozessoren, denn die ersten Rechensysteme waren mechanisch. Sie wurden unter anderem von Blaise Pascal oder Wilhelm Leibniz im 16. Jahrhundert erbaut. Sie funktionierten mit einem Rädertriebwerk und konnten addieren und subtrahieren. All diese Maschinen litten aber an Fehleranfälligkeit der Mechanik und blieben Einzelstücke.
Besser waren die Rechner mit elektromagnetischen Relais, wie die 1941 von Konrad Zuse fertiggestellte Z3. Die Z3 lief immerhin mit einer Taktfrequenz von 5 Herz. Sie war frei pro-

grammierbar und verarbeitete sogar Gleitkommazahlen. Das Programm wurde über einen Lochstreifen eingegeben. Die Mark I von Howard Aiken wurde 1944 fertig. Sie arbeitete ebenfalls mit Relais und konnte Festkommazahlen verarbeiten.

Die nächste Generation von Rechnern wurde mit Elektronenröhren betrieben. Berühmt wurde COLOSSUS, ein während des zweiten Weltkrieges in

Großbritannien entwickelter Röhrenrechner. Röhrenrechner waren Ungetüme. ENIAC (USA, 1946) besaß z.B. 18000 Elektronenröhren, 1500 Relais und eine Masse von 30 Tonnen. Alle Rechner wurden bis zu diesem Zeitpunkt mit Programmen gespeist, die entweder von Lochstreifen eingelesen oder sogar durch elektrische Verbindungen mit Steckbrücken und Kabeln dargestellt wurden. Das war natürlich sehr umständlich. Einer der damaligen Projektmitarbeiter war John von Neumann. Er entwickelte ein neues Konzept, in welchem das Programm wie die Daten einfach im Arbeitsspeicher liegen sollte. Damit war die "*von-Neumann-Maschine*" geschaffen, heute die dominierende Architektur. Im Jahr 1953 stellte eine kleine amerikanische Büromaschinen-Firma namens IBM ihren ersten Elektronenröhrencomputer (IBM701) vor.

Die Erfindung des Transistors durch Bardeen, Brattain und Shockley 1948 (Nobelpreis 1956) in den Bell Laboratorien revolutionierte die Computerwelt in kurzer Zeit. Transistoren können sich in binärer Logik gegenseitig schalten wie Röhren, sind aber viel kleiner und billiger. Schon 1950 wurde am Massachusetts Institute of Technology (M.I.T.) der erste Transistoren Computer TX-0 gebaut. Die Nachfolger hatten bereits ein modernes Buskonzept und Zykluszeiten von 5µs und weniger. Sie waren um Längen günstiger als Röhrenrechner und verdrängten diese vom Markt.

Der nächste Technologie-Sprung wurde durch die Erfindung der integrierten Schaltkreise (ICs) durch Robert Noyce im Jahr 1958 ausgelöst. Es war nun möglich, Dutzende von Transistoren und damit ganze Logikschaltungen auf einem Chip unterzubringen. Eine zentrale Prozessoreinheit (CPU) passte allerdings noch nicht auf einen Chip, sondern musste aus vielen integrierten Schaltungen aufgebaut werden. Trotzdem wurden die Computerschaltungen kompakter, schneller, zuverlässiger und kostengünstiger. Noyce gründete mit Gordon Moore und Arthur Rock eine kleine Firma, die sich mit ICs beschäftigte und den Namen Intel erhielt. Im Rahmen eines Entwicklungsauftrages (1969) schlug einer der Intel-Ingenieure vor, einen der Chips als frei programmierbare CPU zu bauen. Man verwirklichte dieses Konzept und brauchte statt der vorgesehenen 12 Chips nur noch 4 Chips. Einer davon war die CPU die die Bezeichnung 4004 erhielt. Sie hatte eine 4-Bit ALU (Arithmetisch Logische Einheit), 4-Bit Register, 4 Datenleitungen und konnte mit ihren 12 Adressleitungen einen Arbeitsspeicher von 4 Kbyte ansprechen. Man musste in den IC 2300 Transistoren

einbauen, um alle Funktionen zu realisieren. Der 4004 wird heute als der erste Mikroprozessor betrachtet. Die Weiterentwicklung dieses Konzepts erfolgte rasant. 1978 bracht Intel den ersten 16-Bit-Mikroprozessor 8086 auf den Markt. Aus so einem solchen Prozessor als Kern entwickelten sich die heute sogenannten PCs (Personal Computer).
Die Computer- und speziell die Mikroprozessortechnik ist ein Gebiet, das sich bekanntermaßen recht schnell entwickelt. An dieser Stelle daher einige Zahlen, nennen die dies verdeutlichen:

Die Komplexität: Der erste Mikroprozessor (i4004) hatte ca. 2300 Transistoren, heute werden Prozessoren mit mehr als 1 Milliarde Transistoren gefertigt.

Der Integrationsgrad: Die Anzahl von Elementschaltungen (z.B. Gatter) ist von weniger als 100 auf mehr als 1 Million pro Chip angestiegen.

Der Arbeitstakt: Die ersten Mikroprozessoren wurden mit Taktfrequenzen unterhalb von 1 MHz betrieben, aktuelle PC-Prozessoren laufen mit über 4 GHz

Die Verarbeitungsbreite: Der i4004 arbeitete mit 4 Bit Verarbeitungsbreite, heute sind 64 Bit Verarbeitungsbreite üblich.

Der Fortschritt in der Mikroprozessortechnik wirkte sich nicht nur auf die Spitzenprodukte aus, sondern ermöglichte auch riesige Fortschritte bei computergesteuerten Geräten, den *Embedded Systems*. Ohne die heute verfügbaren preiswerten und leistungsfähigen Mikrocontroller wären Handys, MP3-Player, kompakte GPS-Empfänger usw. überhaupt nicht denkbar. So rasant diese Entwicklung auch ist, verläuft sie doch seit vielen Jahren sehr gleichmäßig. Gordon Moore, einer der Intel-Gründer, stellte 1994 fest, dass sich die Anzahl der Transistoren in den integrierten Schaltungen regelmäßig in 18 Monaten verdoppelt. Dieser Zusammenhang, das *Mooresche Gesetz*, lässt sich an Speicherbausteinen und Prozessoren beobachten. Offenbar war man bisher immer in der Lage, die technologischen Probleme zu meistern, die mit der weiteren Verkleinerung einhergehen. Wenn allerdings die Bauelemente in den Integrierten Schaltungen nur noch die Größe von einigen Atomen haben, werden quantenphysikalische Effekte ihr Verhalten bestimmen und eine ganz neue Technologie wird entstehen. Natürlich ist es noch nicht so weit. Wenn man sich allerdings vor Augen führt, was vor einigen Jahren noch nahezu wie unvorstellbar war und heute als ganz selbst-

verständlich gilt, ist absehbar, in welche Richtung sich die Technik mit neuen Meilensteinen weiterentwickelt. (Wüst, 2009, S1f.) [18]

2.4 Anwendungsbereiche

Mikrocontroller sind heute aus der Technik nicht mehr wegzudenken.

Abbildung 2.3: Anwendungen von Mikrocontrollern

Man findet sie heute in nahezu jedem elektronischen Gerät. Sie werden in Thermometern zur Anzeige der Temperatur eingesetzt und steuern in Kaffeeautomaten die richtige Zusammensetzung des Kaffees. Mikrocontroller öffnen und schließen automatische Garagentore und unterstützen den Autofahrer in gefährlichen Situationen durch c und ESP. In der heutigen Welt der Computer finden sich hauptsächlich zwei Anwendungsbereiche, die Personal Computer (PC) und die Mikrocontroller zur Steuerung von Geräten. Diese werden auch als *Embedded Systems* bezeichnet, die in die Anwendung eingebettet sind und als Bestandteil des zu steuernden Gerätes angesehen werden, zum Beispiel in einer Waschmaschine oder einem Blue-Ray-Player. Entsprechend dem hohen Marktanteil der Mikrocontroller gibt es viele Hersteller, die “Familien“ entwickeln und vertreiben. Sie reichen von einfachen 4-Bit Controllern z. B. für Fahrradcomputer bis hin zu 32-Bit-Bausteinen, z.B. für moderne Mobilfunkgeräte. Eine Familie umfasst mehrere Bausteine mit gleichem Befehls- und Registersatz, die sich jedoch in der Ausführung der Peripherieeinheiten und in der Speichergröße voneinander unterscheiden. (Schmitt, 2008, S11) [17]

Ein Mikrocontroller verfügt über eine unterschiedliche Anzahl von Ein- und Ausgängen sowie über spezielle Hardwarebausteine im Inneren, mit denen verschiedene Aufgaben vereinfacht werden. So verfügt nahezu jeder Mikrocontroller über einen Timer, mit dem man Zeiten bestimmen und Signale definierter Länge generieren kann. Ebenso haben sehr viele Bausteine einen integrierten Analog-Digital-Wandler, der es ermöglicht, analoge Signale wie

z.B. Temperatur oder Batteriespannung zu messen. Die Größe des Mikrocontrollers hängt in erster Linie von der Anzahl der Ein- und Ausgänge ab. Ein Mikrocontroller, der nur die Betriebsspannung überwachen soll und beim Unterschreiten eines bestimmten Schwellenwertes ein Signal ausgeben soll, wird mit weniger Ein- und Ausgängen (Pins) auskommen als ein Controller, der die Zimmertemperatur mehrerer Zimmer regeln soll und die Steuerung über ein Farbdisplay mit Touchscreen visualisiert.

2.5 Die Auswahl des Mikrocontrollers

Damit der Prototyp unserer universellen Mehrkanal DC-Bias-Versorgung am Ende überhaupt seine spezifischen Aufgaben erfüllen kann, muss zuerst der richtige Mikrocontroller ausgewählt werden.
Somit bestand ein der ersten Aufgaben der Projektes darin, eine Liste der Kriterien zusammenzustellen, die der Mikrocontroller erfüllen sollte. Anhand der Projektausschreibung ergaben sich folgende Anforderungen.

1.) Kompaktheit
2.) Deep-Sleep
3.) RS232
4.) I2C
5.) SPI
6.) Lieferbarkeit
7.) JTAG

Der Mikrocontroller sollte möglichst klein sein, da eines der Hauptkriterien die Größe des DC-Versorgungsmoduls ist. Dieses sollte so klein wie möglich sein, wenn möglich sogar in einer Extra-Kammer des zu versorgenden RF-Moduls passen. Somit sollte die Bauform auf jeden Fall *SMD* sein. SMD (*surface-mounted device*) bedeutet “oberflächenmontierbares Bauelement“. Während die Anschlussdrähte konventioneller Bauelemente, wie sie auch bis zum heutigen Tage durchaus noch Verwendung finden, durch Bestückungslöcher geführt werden und auf der Rückseite der Leiterplatte (oder über Innenlagen) verlötet werden müssen (Durchkontaktierung), entfällt dies bei SMD-Bauelementen. Dadurch werden sehr dichte Bestückungen und vor allem eine beidseitige Bestückung der Leiterplatte möglich, was die elektrischen Eigenschaften der Schaltungen speziell bei höheren Frequenzen positiv beeinflusst und den Platzbedarf der Bauelemente verringert. Dadurch können die Geräte kleiner und zugleich wesentlich preiswerter hergestellt werden.

Ein weiteres wichtiges Kriterium ist, dass der Mikrocontroller in einen sogenannter "*Tiefschlafmodus*" (Deep-Sleep) versetzt werden kann. In diesem Modus soll der Mikrocontroller seinen eigenen Takt abschalten, damit keine digitalen Störungen auf das empfindliche RF-Modul übertragen werden. Des Weiteren sollte er bestimmte Schnittstellen besitzen, z.B. I2C, SPI und RS232, auf die später in der Arbeit noch ausführlicher eingegangen wird. Wichtig ist auch JTAG (Joint Test Action Group).

JTAG ist ein Verfahren zum Testen und Debuggen von elektronischer Hardware direkt in der Schaltung. Auch bekannt unter dem Namen *Boundary Scan Test*. Da eine langfristige und gute Lieferbarkeit bei Mikrocontrollern nicht selbstverständlich ist, muss man sich im Vorfeld gut über den Stand der Dinge informieren.

Bei Mikrocontrollern gibt es unter anderem zwei große Herstellerfirmen. Zu einem Microchip (USA) und zum andern Amtel (USA). Beide bieten mehrere Controller-Familien an. Die Unterschiede zwischen Microchip (PIC) und Amtel (AVR) sind geringfügig, beide bieten leistungsstarke Controller mit verschiedenen Spezifikationen und Bauformen. Die Entwicklungsumgebung kann auf der entsprechenden Herstellerseite kostenlos herunterladen (microchip.com, amtel.com) werden. Es gibt auch eine Vielzahl an Development- Boards, auf denen man den Mikrocontroller in einer realen Hardwareumgebung testen kann. Laut erfahrenen Benutzern von Mikrocontrollern ist die Wahl der Hersteller Geschmacks- und Gewöhnungssache (z.B. Support und Design der Homepage).

Nach reichlicher Recherche wurden leistungsstarke Controller für das Projekt gefunden. Die Entscheidung fiel auf PIC24FJ64GA002-I/SS. Die wichtigsten Spezifikationen werde ich im *Abschnitt 6.1* ausführlich erläutert, da erst nach Lesen der ersten Kapitel ein gutes Grundverständnis für die Fachausdrücke und Funktionen gewonnen werden kann.

Bei den SMD-Controllern von Microchip gibt es verschiedene Bauformen. Ausgewählt wurde die Bauform SS. Mit den Abmessungen 28 Pins 10,2 x 7,8 x 2 mm (Länge, Breite ,Höhe). Zu Testzwecken wurden die Mikrocontroller auch zusätzlich im einfachen Dip-Gehäuse bestellt. (Kosten ca. 3 Euro, Lieferzeit ca. 14 Tage.)

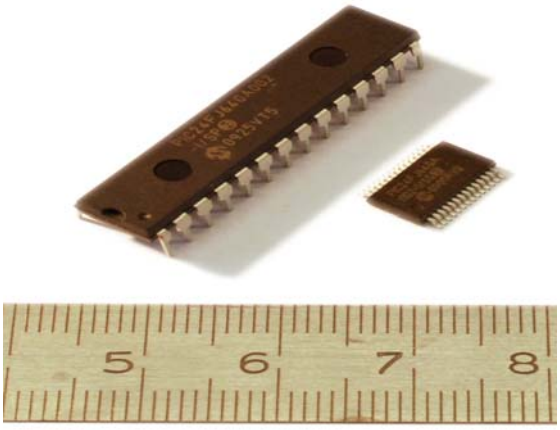


Abbildung 2.4: PIC24FJ64GA002-I/SS Mikrocontroller

PIC bedeutet hier, dass der Mikrocontroller von der Firma Microship ist. Die Abkürzung PIC steht schlicht für *Programmable Integrated Circuit.*

2.6 Warum die Firma Microchip?

Die Tatsache, dass die Firma Microchip eine komplette kostenlose Entwicklungsumgebung zur Verfügung stell, spielte bei der Auswahl des Produktes eine zentrale Rolle. Ein weiterer Pluspunkt ist, dass die Controller bei bekannten Lieferanten für elektronische Bauteile gekauft werden können, was ihnen zu großer Beliebtheit verhilft. Dadurch wiederum sind auch einige Beispiele und Diskussionen im Internet zu finden.

2.7 Das Entwicklungsboard

Da ein Mikrocontroller ohne Softwareprogramm nicht arbeiten kann, müssen nun die Programmierung und ein Test des Mikrocontrollers erfolgen. Hierbei ist ein spezielles Entwicklungsboard hilfreich. Im Internet finden sich viele Bastelvorschläge für Programmiergeräte, mit denen PICs programmiert werden können. Entscheidet man sich aber für ein Tool von Mikrochip, hat es den Vorteil, dass das Programmiergerät direkt von MPLAB aus verwendet werden kann und keine zusätzliche Software für das Programmieren erforderlich ist. Außerdem werden Fehler beim Eigenbau vermieden. Nach eingehender Recherche wurde das *Explorer 16 Development Board* bestellt. (Kosten ca. 100 Euro, Lieferzeit 10 Tage.)

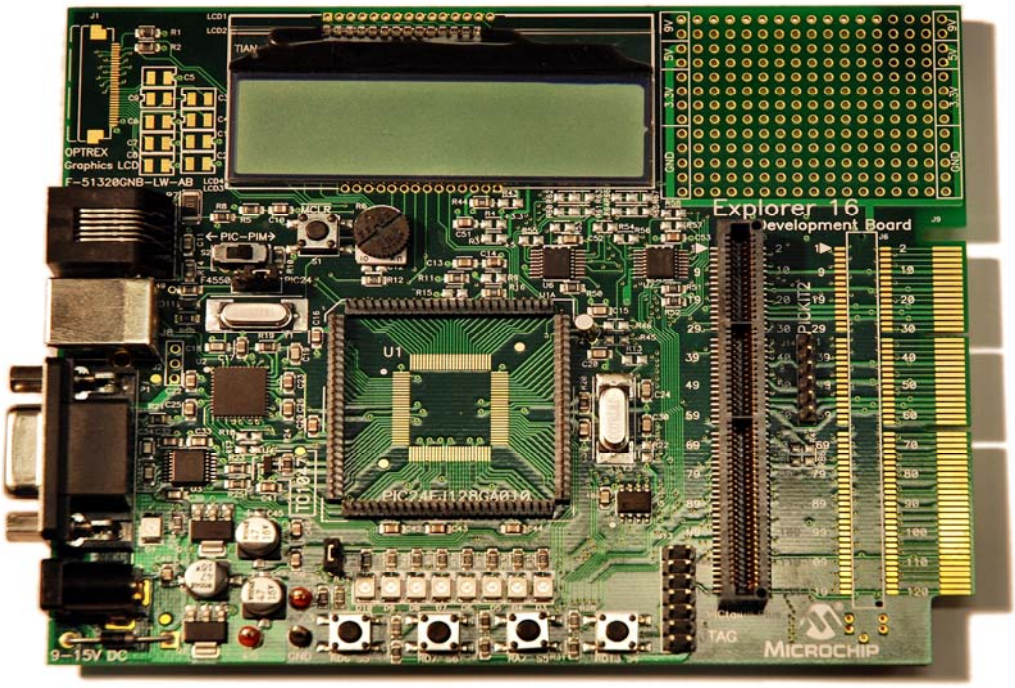


Abbildung 2.5: Explorer 16 Entwicklungsboard

Bei der Bestellung des *Explorer 16 Boards* wurden zwei Testsockel mit einem Test-Controller (100 Pins) mitgeliefert.

Abbildung 2.6: Fertige PIM-Sockel von Microchip; rechts eigener PIM-Sockel

Leider kann hier der Mikrocontroller nicht ausgesucht werden, sondern man bekommt einen Standartcontroller für diese Bestellung. Dies stellt ein Problem dar, da eine Einarbeitung mit dem Controller erfolgen soll, der später auch verwendet wird. Nach zahlreichen Tagen Recherche im Internet konnte ein Sockel mit dem passenden Mikrocontroller PIC24FJ64GA002 gefunden werden. Lieferzeit betrug allerdings 3-6 Monate. Dies war für

ein zeitlich begrenztes Projekt dieser Art unzumutbar. Die Alternativlösung bestand darin diesen Sockel selbst zu entwerfen. Dazu mussten alle Funktionen der Pins des mitgelieferten Mikrocontrollers (100 Pin) mit den Funktionen des Controllers (28 Pin) abgeglichen werden. Dies erwies sich als recht schwierig, denn ein einzelner Pin hat 3 bis 5 Funktionen, die dem Board zugeordnet werden mussten. Danach musste ein Platinen-Layout erstellt und in Auftrag gegeben werden. Auf der kommenden Abbildung wird der Zuordnungsplan des Sockels dargestellt. Eine detailliertere Abbildung ist in den Anlagen zu finden. (Lieferzeit der Platine ca. 2 Wochen)

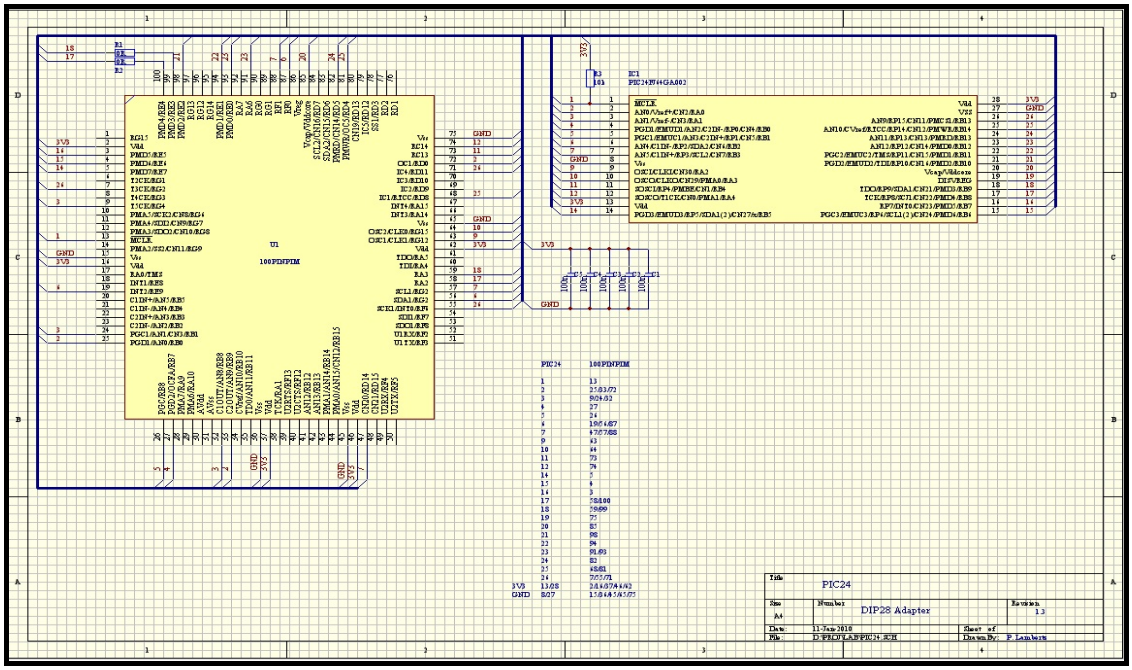


Abbildung 2.7: Zuordnungsplan eigener PIM-Sockel

Im Folgenden ist die entworfene Platine mit bereits aufgelötetem Sockel für den Controller abgebildet. Zusätzlich wurde eine Adapterplatine anfertigt, die den Aufsatz eines SMD-Gehäuses ermöglicht. Damit ist es nun möglich dem Mikrocontroller in beiden Bauformen zu programmieren und zu Tests in eine Anwendung einzusetzen.

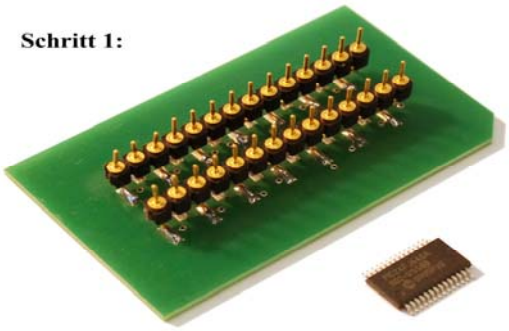


Abbildung 2.8: Eigener PIM-Sockel; Mikrocontroller und Adapterplatine für SMD (1)

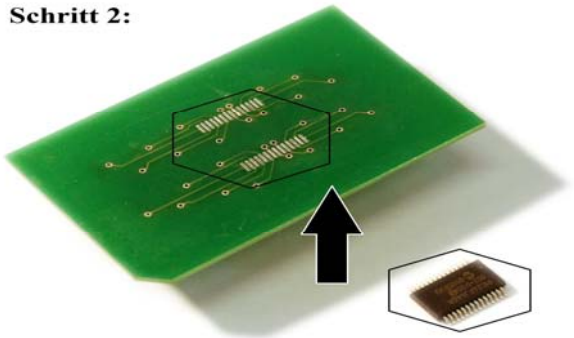


Abbildung 2.9: Eigener PIM-Sockel; Mikrocontroller und Adapterplatine für SMD (2)

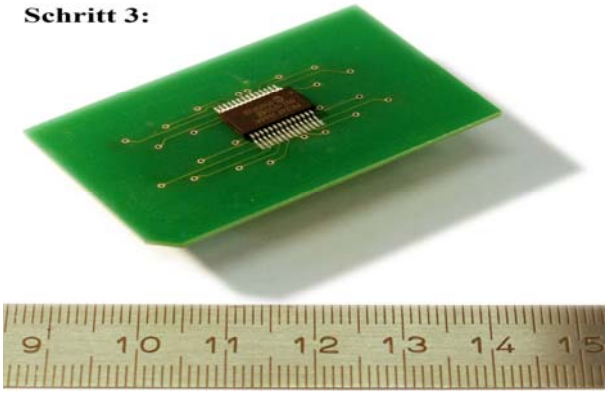


Abbildung 2.10: Eigener PIM-Sockel; Mikrocontroller und Adapterplatine für SMD (3)

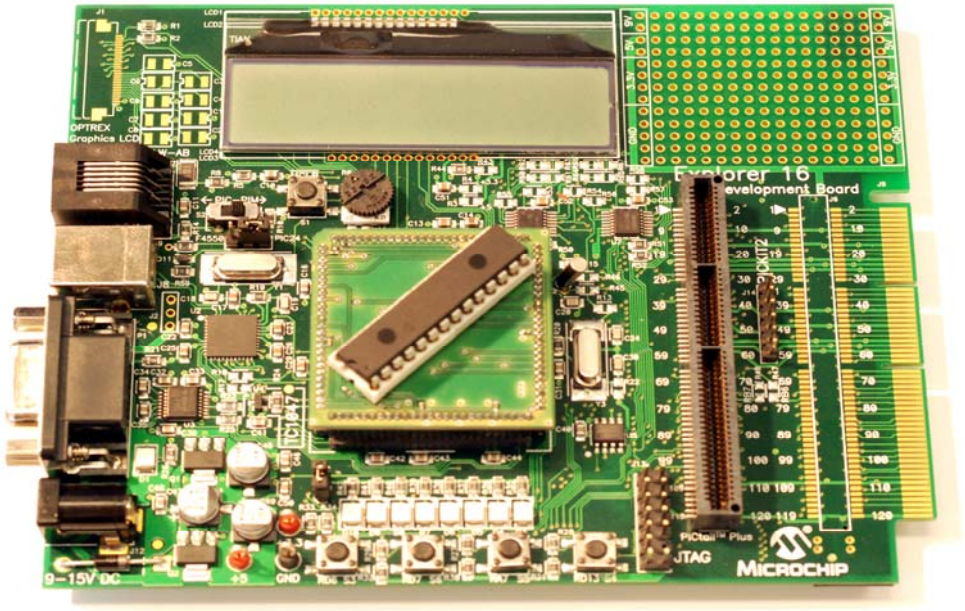


Abbildung 2.11: Explorer 16 Entwicklungsboard mit eigenem Sockel

Außerdem wird empfohlen sich einen In-Circuit-Debugger anschaffen. Die Entscheidung fiel auf den MPLAB ccc 3 (ebenfalls von Microchip). Mit diesem Modul kann der Mikrocontroller in der realen Hardware-Umgebung testen. Die Details zur Programmierschnittstelle werden in *Kapitel 5* näher erläutert.

Abbildung 2.12: ICD3 Debugger von Mikrochip

3. Ein Mikroprozessor/Controller im Detail

3.1 Allgemeines

Bei einem Mikrocontroller (Abk. MC oder µC) sind die CPU, Speicher, Peripheriekomponenten und Interrupt-System auf einem Chip integriert. Ein Mikrocontroller kann also mit sehr wenigen externen Bausteinen betrieben werden, man nennt sie daher auch *Single-Chip-Computer* oder *Einchip*-Computer. Im Gegensatz zum Mikroprozessor steht beim Mikrocontroller nicht nur die hohe Verarbeitungsleistung im Vordergrund, sondern auch eine hohe funktionelle Integration. Je mehr Funktionen schon auf dem Mikrocontroller-Chip sind, desto weniger Zusatzbausteine werden benötigt. Dies hat viele Vorteile für die Entwicklung vollständiger Systeme. Der Schaltungsentwurf wird einfacher und das vollständige System kompakter. Die Verlustleistung ist geringer. Durch die geringere Anzahl von Leitungen, Sockeln und Steckern verringert sich auch das Risiko von mechanischen Verbindungsstörungen. Außerdem sind die Kosten für die Fertigung und das Testen der Schaltung geringer. (Wüst, 2009, S238) [18]

3.2 Der Aufbau

Ein Mikrocontroller enthält einen Kern (Core), der mit den Bestandteilen Rechenwerk, Steuerwerk, Registersatz und Busschnittstelle ungefähr einer üblichen CPU entspricht. Dazu kommen mehr oder weniger viele Peripheriekomponenten, die der Controller für seine Steuerungs- und Kommunikationsaufgaben im eingebetteten System braucht, z.B.:

- Verschiedene Arten von Speicher
- Kommunikationsschnittstellen (UART, I²C,SPI,CAN usw.)
- Ein konfigurierbares Interrupt-System mit externen Interrupts
- Eine Oszillatorschaltung
- Ein- und Ausgabeports (IO-Ports)
- Zähler-/Zeitgeber-Bausteine
- Analog-Digital-Wandler
- Digital-Analog-Wandler
- Echtzeituhr (RTC)
- Ansteuerung von LCD- und LED-Anzeigeelementen

Mikrocontroller, die den gleichen Kern haben und sich in den Peripheriekomponenten unterscheiden, nennt man *Derivate*. Von manchen Mikrocontrollern existieren sehr viele Derivate. (Wüst, 2009, S239) [18]

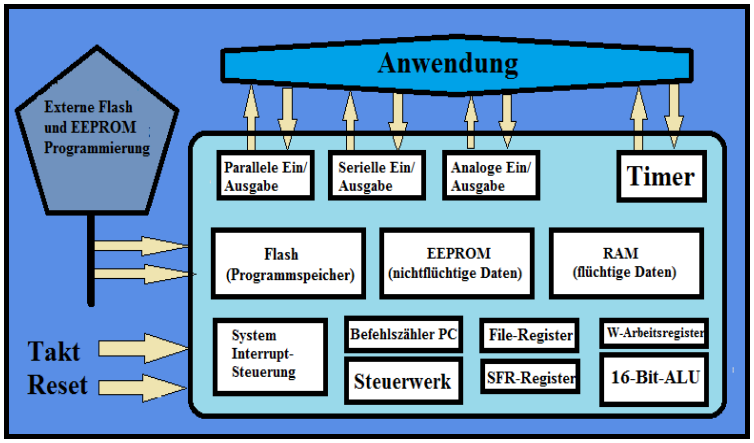


Abbildung 3.1: Blockschaltplan eines Mikrocontrollers eingebettet in eine Anwendung

3.3 Der Kern

Die Kerne der Mikrocontroller entsprechen dem Mikroprozessor eines Rechners. Sie bestimmen daher die Datenverarbeitungsbreite, den Befehlssatz, den Registersatz, die Adressierungsarten und die Größe des adressierbaren Speichers. Unterschieden wird zwischen der RISC- und CISC-Architektur.

CISC steht für Complex Instruction Set Computer. Übersetzt ist das ein Prozessor mit komplexem Befehlssatz. Der CISC-Prozessor zeichnet sich durch einen großen Befehlsumfang und komplexe Adressierungsmöglichkeiten aus. Sie benötigen in der Regel vier bis zehn Takte bis ein Befehl ausgeführt ist. Der Mikrocode wird der Reihe nach ausgeführt. Alle anderen Anweisungen müssen solange auf die Anweisung warten. Charakteristisch für die CISC-Architektur ist eine große Anzahl, meist einige Hunderte, von zum Teil sehr komplexen Maschinenbefehlen unterschiedlicher Länge. Die Ausführung wird durch ein Mikroprogrammwerk gesteuert. Ein Beispiel ist der Pentium-Prozessor.

RISC steht für Reduced Instuction Set Computer, was so viel heißt wie *Computer mit eingeschränktem Befehlssatz.* Dies bedeutet, dass jeder einzelne Befehl nur relativ einfache Operationen ausführt. Durch diese Beschränkung ist die Ausführung der Einzelbefehle schneller möglich als für die komplexen Befehle der CISC-Prozessoren. Ein Beispiel ist hier der PIC-Mikroprozessor. Heutige RISC-Prozessoren überschreiten aber auch die Grenzen der engen Definition und enthalten auch komplexere Befehle.

Welche Kerne sind nun in der Mikrocontrollerwelt sinnvoll und verbreitet? Da in vielen Anwendungen keine hohe Rechenleistung gebraucht wird, haben die meisten Mikrocontroller noch einen 8-Bit-Kern, sogar 4-Bit-Kerne werden noch häufig eingesetzt. Da diese Arbeit ein Forschungsprojekt ist und kein Produkt, was kostenoptimiert hergestellt werden soll, spielen minimal höhere Kosten für einen leistungsfähigeren Prozessor (16Bit) kostet keine Rolle. In der privaten Wirtschaft muss das Preis-Leistungs-Verhältnis für eine Anwendung berücksichtigt werden. Es macht z.B. keinen Sinn einen teuren 32-Bit-Controller nur für ein Anzeigesignal einer leeren Batteriespannung einzusetzen. Bei der Arbeitsfrequenz gibt man sich häufig mit bescheidenen Taktfrequenzen zufrieden, z.B. 12 MHz oder weniger. Die Anzahl der rechenintensiven Controlleranwendungen wächst aber, z.B. bei der Steuerung von KFZ-Motoren. Der Trend geht daher zu schnelleren 16- und 32-Bit-Kernen. Wie auch bei Mikroprozessoren findet man bei Mikrocontrollern CISC- und RISC-Kerne. Die Kerne der Mikrocontroller werden auf ihre spezielle Aufgabe angepasst. So werden beispielsweise Befehle oder Register ergänzt, um die spezielle ON-Chip-Peripherie anzusprechen. Um eine gute Speicherausnutzung zu erreichen, haben viele Controller Bitbefehle, mit denen einzelne Bits direkt gespeichert und manipuliert werden können. (Wüst, 2009, S239f.) [18]

3.4 Programmspeicher

Ein Mikrocontroller, der ohne externen Programmspeicher in einem Embedded System arbeiten soll, muss sein übersetztes Anwendungsprogramm in einem On-chip-Programmspeicher haben. Dieser Speicher muss auch ohne Versorgungsspannung seine Daten halten, kann also kein DRAM oder SRAM sein. Die Größe der Speicher reicht von wenigen Byte bis zu mehreren Mbyte. Folgende Speicher werden eingesetzt:

EPROM und EEPROM: Der Mikrocontroller kann beim Kunden programmiert werden; das Programm bleibt veränderbar; geeignet für Entwicklung und Test der Programme.

OTP-ROM (One Time Programmable): Der Mikrocontroller kann einmal beim Kunden programmiert werden; geeignet für Kleinserien. Man erkennt die OTP-Typen durch ein *C* im Namen.

Flash-EEPROM: Der Mikrocontroller verwaltet den Flash-Speicher selbst und kann über Kommandosequenzen programmiert werden, auch dann wenn er sich schon im fertigen System befindet.

Masken-ROM: Programmcode wird bei der Herstellung des Mikrocontrollers eingearbeitet (Maske) und ist nicht mehr änderbar; geeignet für Großserien.

Nichtflüchtiges RAM (non-volatile RAM, NV-RAM): Der Inhalt der Speicherzellen wird vor dem Ausschalten in EEPROM-Zellen übertragen. (Wüst, 2009, S240) [18]

3.5 Datenspeicher

Um das System einfach zu halten, wird als ON-Chip-Datenspeicher meist SRAM verwendet, das ohne Refresh auskommt. Für Daten, die ohne Spannungsversorgung erhalten bleiben sollen, besitzen manche Controller einen EEPROM oder NVRAM-Bereich. Die Datenbereiche eines µC werden nach Funktion unterschieden:

Allgemeiner Datenbereich: Zwischenspeicherung von Programmdaten

Spezial Funktion Register: Zur Konfiguration und Ansteuerung der Peripheriebereiche des Mikrocontrollers

Registerbänke: Für Zeiger und evtl. auch arithmetische/logische Operanden

Stack: Für kurzzeitige Zwischenspeicherung von Daten, in der Größe begrenzt

Bitadressierbarer Bereich: Ein Bereich, in dem ohne Verwendung von Masken auf einzelne Bits zugegriffen werden kann

Die Größe der On-Chip-Datenspeicher ist sehr unterschiedlich, von wenigen Bytes bis hin zu einigen kByte, in der Regel aber deutlich kleiner als der Programmspeicher. Bei der Adressierung von Daten muss zwischen internen und externen Speicher unterschieden werden, die internen und externen Adressbereiche können sogar überlappen. Register und Ram sind nicht unbedingt streng getrennt, bei manchen Mikrocontrollern ist das On-Chip-RAM mit den Registern zu einem großen Register-File zusammengefasst. (Wüst, 2009, S241) [18]

3.6 Ein-/Ausgabeschnittstellen (Input/Output-Ports)

Input/Output-Ports, kurz I/O-Ports, sind für Mikrocontroller besonders wichtig, sie erlauben den Austausch *digitaler Signale* mit dem umgebenden System. Ein Beispiel: Ein Mikrocontroller, der eine Infrarot-Fernbedienung steuert, kann über einen digitalen Eingang feststellen,

ob gerade eine Taste gedrückt wird. Über einen digitalen Ausgang kann er die Sendediode ein- und ausschalten. Über I/O-Ports werden also binäre Signale verarbeitet, die in TTL-Pegeln dargestellt werden. Alle Mikrocontroller verfügen daher über eigene I/O-Ports, diese sind meistens in Gruppen zu 8 Bit organisiert. Typischerweise gehört zu jedem Port ein Datenregister und ein Richtungsregister, das die Richtung des Datenaustausches (Ein- oder Ausgabe) festlegt. Die Portausgänge sind in ihrer Schaltungstechnik einfach gehalten: Die Ausgangsstufen sind Open-Collector-, Open-Drain- oder Gegentakt-Endstufen. Ports werden auch benutzt, um externe Bausteine anzusteuern, in diesem Fall findet man auch Tristate-Ausgänge. Mikrocontroller–Ports sind oft rücklesbar, d.h. im Ausgabebetrieb kann der anliegende Wert vom Controllerkern wieder eingelesen werden. Dies erspart die zusätzliche Abspeicherung des Portzustandes in RAM-Zellen.
Um einen vielseitigen Mikrocontroller zu erhalten, der aber nicht allzu viele Anschlussstifte hat, werden über die Portanschlüsse oft alternativ andere Funktionen abgewickelt, z.B. Analogeingang, Zählereingang oder serielle Datenübertragung. Über Konfigurationsregister kann dann per Software die Arbeitsweise dieser Anschlussstifte festgelegt werden. (Wüst, 2009, S241f.) [18]

3.7 Zähler/Zeitgeber (Counter/Timer)

Zähler/Zeitgeber-Bausteine sind typische und sehr wichtige Peripheriegruppen, die fast jeder Mikrocontroller besitzt. Schon bei einer einfachen Impulszählung wird der Nutzen dieser Baugruppe offensichtlich. Ohne den Zähler-/Zeitgeberbaustein müsste ein Mikrocontroller den betreffenden Eingang in einer Programmschleife ständig abfragen (*pollen*) und bei jeder zweiten Flanke den Wert einer Speichervariablen inkrementieren. Der Zählerbaustein befreit die CPU von dieser zeitraubenden Aufgabe. Das Kernstück des Zählers-/Zeitgeberbausteins ist ein Zähler, der durch eingehende Impulse inkrementiert oder dekrementiert wird. Im *Zählerbetrieb* (Counter) kommen diese Impulse über einen Anschlussstift von außen in den Mikrocontroller und werden einfach gezählt. Im *Zeitgeberbetrieb* (Timer) werden die Impulse durch das Herunterteilen des internen Oszillatortaktes gewonnen. Da der Oszillator bekannt ist, sind damit exakte Zeitmessungen möglich. Die Umschaltung erfolgt durch einen Multiplexer. Eine Torsteuerung bewirkt, dass nur Impulse auf den Zähler gelangen, wenn der Eingang durch den Toreingang freigegeben ist (*Gated Timer*). Damit kann z.B. festgestellt werden, über welchen Anteil einer bestimmten Zeitspanne ein Signal *HIGH* war. Die Umschaltung zwischen Zähler und Zeitgeberfunktion, der Toreingang und der Teilerfaktor werden über ein Konfigurationsregister (*z.B. Special-Function-Register*) programmiert. Die Zähler können aufwärts oder abwärts laufen. Jeder eintreffende Impuls erhöht bzw. erniedrigt das

Zählerregister um eins. Der Maximalwert für einen Zähler mit N Bit liegt bei 2^N-1, das nächste Inkrement führt zum Zählerüberlauf. Beim Überlauf kann der Zählerbaustein einen Interrupt auslösen oder einen Pegelwechsel an einem Anschlussstift bewirken. Nach dem Überlauf muss die Zählung nicht unbedingt bei null beginnen, der Interrupt-Handler kann auch einen anderen Wert ins Zählerregister laden (*Software-Reload*). Manche Zähler verfügen über einen *Auto-Reload-Betrieb*, auch *Hardware-Reload* genannt. Dabei wird beim Überlauf ein Wert aus dem Autoreload-Register ins Zählregister übertragen. Der Zähler beginnt danach also nicht beim Startwert Null, sondern beim Reload-Wert und entsprechend schneller kommt es zum nächsten Zählerüberlauf. Über den Reload-Wert lässt sich im Zeitgeberbetrieb bequem die Zykluszeit des Zählerüberlaufs einstellen. Die Zeit von einem Zählerüberlauf zum nächsten ist bei einem Zählregister mit N Bit und einer Zählerinkrementzeit T_I.

$T = (2^N - \text{Startwert}) \cdot T_I$

Die Verwertung des Inhalts eines Zählerregisters kann auf zwei Arten erfolgen: Durch *Capture* und durch *Compare***.**

Capture-Modus: Durch ein *Capture* (Auffangen) wird der momentane Inhalt des Zählerregisters in ein Capture-Register übertragen, ohne den Zähler anzuhalten. Das Capture-Register kann von der CPU gelesen werden. Ein *Capture* kann durch einen Programmbefehl oder ein externes Ereignis, wie z.B. eine Pegeländerung an einem digitalen Eingang, ausgelöst werden.

Compare-Modus: Hier wird der Inhalt des Zählerregisters nach jeder Änderung mit dem Wert in einem Vergleichsregister verglichen. Wenn beide Werte identisch sind, tritt ein Compare-Ereignis ein. Dieses löst, je nach Betriebsart, z.B. einen Pegelwechsel an einem Ausgabestift oder einen Interrupt aus.

Typische Verwendungen eines Zählers sind:

- Zählung von Impulsen
- Messen von Zeiten
- Erzeugung von Impulsen
- Erzeugung von plusweiten-modulierten Signalen

Wegen der großen Praxisrelevanz dieser Anwendungen haben parktisch alle Mikrocontroller eine mehr oder weniger leistungsfähige Zähler-/Zeitgeber-Einheit. (Wüst, 2009, S242f.) [18]

3.8 Analoge Signale

In *Embedded Systems* ist es oft erforderlich, mit analogen Signalen zu arbeiten. Solche Signale kennen nicht nur die beiden Bereiche *HIGH* und *LOW*, sondern können kontinuierlich alle Spannungswerte zwischen 0 Vc und einer maximalen Spannung annehmen. Analoge Signale werden beispielsweise von Messfühlern geliefert und repräsentieren dann gemessene physikalische Größen. Ein Beispiel dafür wäre das Signal eines Temperatursensors. Solche Signale können von Mikrocontrollern mit einem *Analogeingang* verarbeitet werden. In anderen Fällen erzeugt der Mikrocontroller mit einem *Analogausgang* analoge Signale, die dann zur Peripherie übertragen werden, beispielweise zu einem spannungsgesteuerten externen Oszillator. Analoge Signale können nicht über I/O-Ports ein- und ausgegeben werden, diese verarbeiten digitale Spannungspegel. Es gibt zwei Möglichkeiten analoge Signale einzulesen: *Analog-Digital-Umsetzer* und *analoge Komparatoren*. Für die Ausgabe von analogen Signalen gibt es die Möglichkeit, einen *Digital-Analog-Umsetzer* zu benutzen oder mit einem PWM-Verfahren zu arbeiten.

Analog-Digital-Umsetzer: Analog-Digital-Umsetzer, abgekürzt ADU, (engl. Analog-/Digital-Converter, ADC) sind Bausteine, die einen analogen Eingang haben und N digitale Ausgangsleitungen. Ein am Eingang anliegendes analoges Signal wird in eine ganze Zahl umgerechnet und in binärer Darstellung an den N digitalen Ausgangsleitungen ausgegeben.
Die Analog-Digital-Umsetzung hat dann eine Auflösung von N Bit. Das analoge Eingangssignal U_e muss im Bereich von U_{min} bis U_{max} liegen, die erzeugte digitale Zahl Z liegt für einen Umsetzer mit N Bit im Bereich 0 bis 2^N-1. Ein 8-Bit-Umsetzter erzeugt beispielweise Zahlen im Bereich 0 bis 255. Die Umsetzung ist eine lineare Abbildung nach der Formel:
$Z = ((U_e-U_{min}) / (U_{max}-U_{min})) \cdot (2^N-1)$
Die Höhe der Quantisierungsstufen ULSB hängt von der Auflösung des Analog-Digital-Wandlers ab: $U_{LSB} = (U_{max}-U_{min}) / (2^N-1)$

Analoge Komparatoren: Analoge Komparatoren sind einfacher als Analog-Digital-Umsetzer. Sie haben eine digitale Ausgangsleitung und bestimmen nur, ob die anliegende Signalspannung größer oder kleiner als eine Vergleichsspannung ist. Kommt diese Vergleichsspannung von einem programmierbaren Digital-/Analog-Umsetzer, so ist mit entsprechenden Algorithmen auch die Bestimmung des unbekannten Signals möglich.

Digital-Analog-Umsetzer: Digital-Analog-Umsetzer, abgekürzt DAU, (engl. Digital/Analog-Converter, DAC) arbeiten genau umgekehrt zu ADUs. Sie erhalten an ihren digitalen Ein-

gangsleitungen eine Ganzzahl in binären Darstellungen und erzeugen am Ausgang die entsprechende analoge Spannung. Diese Spannung ergibt sich gemäß:

$U_a = (Z / 2^N - 1) \cdot (U_{max} - U_{min}) + U_{min}$

Es sind nur wenige Mikrocontroller mit ON-Chip-DAU ausgerüstet, es bleiben aber die Alternativen eines externen DAU oder PWM-Verfahrens. (Wüst, 2009, S247f.) [18]

3.9 Interrupt-System

Ein Mikrocontroller verfügt bereits auf seinem Chip verschiedene vielseitige Peripherie-Komponenten. Über die I/O-Ports können weitere, externe Komponenten angeschlossen werden. Der Mikrocontroller muss also eine Vielzahl von Subsystemen ansteuern und bedienen. Ohne das Interrupt-Konzept müsste die CPU in vielen Abfrageschleifen die Zustandsflags der Peripheriekomponenten abfragen, wertvolle Zeit ginge verloren. Viel effizienter ist es, wenn z.B. der Analog-Digital-Umsetzer durch einen Interrupt signalisiert, dass die Wandlung beendet ist und ein Ergebnis vorliegt. Durch den Interrupt-Betrieb wird die Effizienz enorm gesteigert und wegen der vielfältigen Peripherie spricht hier noch mehr für das Interrupt-Konzept als bei den Mikroprozessoren. Ein zusätzlicher Aspekt ist die oft geforderte *Echtzeitfähigkeit*, eine garantierte und sichere Reaktion des Mikrocontrollers auf ein Ereignis innerhalb einer definierten maximalen Reaktionszeit. Meldet beispielsweise der Endschalter eines motorbetriebenen Rolltores, dass das Tor nun (fast) seine Endposition erreicht hat, muss der Mikrocontroller in der Schaltung innerhalb sehr kurzer Zeit den Motor ausschalten. Wegen der genannten Gründe verfügen fast alle Mikrocontroller über die Möglichkeit der Interrupt-Verarbeitung. Das Interrupt-System erlaubt es, auf ein Peripherie-Ereignis sofort zu reagieren, ohne die entsprechenden Komponenten ständig abzufragen. Nach der Interrupt-Anforderung wird in der Regel der laufende Befehl noch beendet und danach ein Interrupt-Handler aktiviert, der die Serviceanforderungen der Interrupt-Quelle bearbeitet. Die maximale Reaktionszeit lässt sich also überblicken und bei bekannter Taktfrequenz ausrechnen.

Hier ein paar Beispiele für typische Interrupt-Quellen und die auslösenden Ereignisse: Beim Zähler könnten die Ereignisse *Compare-Ergebnis* oder *Zählerüberlauf* auftreten. Bei einer seriellen Schnittstelle *Zeichen empfangen*, *Empfangspuffer voll*, *Sendepuffer leer* und *Übertragungsfehler*. Ein ADU könnte ein Interrupt auslösen, wenn eine *Umsetzung* beendet ist. Oder ein I/O-Pin (externes Signal) kann bei einer *steigenden-* oder *fallenden Flanke* ein Interrupt auslösen. Bei der CPU können die Ereignisse *Divisionsfehler*, *unbekannter Opcode* oder *Daten-Ausrichtungsfehler* auftreten. Wegen der vielen möglichen Quellen muss der Controller also mit vielen und evtl. auch gleichzeitigen Interrupt-Anforderungen umgehen. Wie bei den Prozessoren muss also die Maskierung, Priorisierung und Vektorisierung der

Interrupt-Quellen bewerkstelligt werden. Man führt hier aber keinen externen Interrupt-Controller ein, sondern integriert diesen auf dem Chip. Damit entfällt die Übertragung der Interrupt-Quellen-Nr. via Datenbus. Die meisten modernen Mikrocontroller haben ein ausgefeiltes und flexibel konfigurierbares Interrupt-System mit den folgenden möglichen Eigenschaften:

- Jede Interrupt-Quelle kann einzeln aktiviert und deaktiviert (maskiert) werden
- Für kritische Phasen können zentral alle Interrupts deaktiviert werden
- Für jede Interrupt-Quelle kann ein eigener Interrupt-Handler geschrieben werden, Größe und Position können durch das Programm festgelegt werden.
- Für jede Interrupt-Quelle kann eine Priorität festgelegt werden, um auch geschachtelte Interrupts zu ermöglichen
- Das Interrupt-System kann auch im laufenden Betrieb konfiguriert werden.

Das Abschalten aller Interrupts ist z.B. während der Konfigurierung des Interrupt-Systems angebracht. Wie erfolgt der Aufruf des Interrupt-Handlers? Es gibt auch hier mehrere Möglichkeiten:

Sprungtabelle: Jede vorhandene Interrupt-Quelle wird zu einer bestimmten, festen Adresse im Code verzweigt. Dort kann dann ein ganz kurzer Interrupt-Handler oder ein Sprungbefehl zu einem längeren Interrupt-Handler stehen.

Vektorentabelle: In der Interrupt-Vektorentabelle wird für jeden Interrupt die Einsprung-Adresse eines Interrupt-Handlers hinterlegt. Bei Auslösung eines Interrupts wird dorthin verzweigt.

Die Interrupt-Behandlungsroutine muss vor dem Rücksprung in das unterbrochene Programm alle Register und Flags wiederherstellen. Die meisten Mikrocontroller unterstützen den Programmierer und speichern bei der Auslösung des Interrupts nicht nur die Rücksprung-Adresse, sondern auch die Flags auf dem Stack. Die Registerinhalte müssen dann mit Transportbefehlen in dem allgemeinen Speicher oder den Stack gesichert werden. Manche Mikrocontroller bieten die Möglichkeit zwischen mehreren Register-Speicherbänken umzuschalten. Der Interrupt-Handler kann dann z.B. zu Beginn der Interrupt-Behandlung auf eine andere Registerbank schalten und am Ende wieder auf die ursprüngliche Bank zurückschalten. Das Sichern von Registern ist somit überflüssig. Mit dieser zeitsparenden Technik sind moderne Controller vielen Mikroprozessoren weit voraus. (Wüst, 2009, S249f.) [18]

3.10 Komponenten zur Datenübertragung

Die Möglichkeit des Datenaustausches mit anderen Bausteinen oder Systemen ist für Mikrocontroller sehr nützlich. Beispiele sind der Datenaustausch mit PCs, mit anderen Mikrocontrollern, mit intelligenten Displays, Kartenlesegeräten oder Speichern mit Kommunikationsinterface. Der Datenaustausch wird fast immer seriell aufgebaut, um mit wenigen Leitungen auszukommen. Fast alle Mikrocontroller sind daher mit irgendeiner Art von serieller Schnittstelle ausgestattet. Es sind asynchrone und synchrone Schnittstellen verbreitet, manche Mikrocontroller haben auch mehrere Schnittstellen. Die Schnittstellen tragen Bezeichnungen wie *Serial Communication Interface* (SCI) oder *Universal Synchronous* (ASC). Die wichtigsten Schnittstellen sind der SPI-Bus (*Serial Peripheral Interface*) der Fa. Motorola, der I²C-Bus (*Inter Integrated Circuit Bus*) der Fa. Philips und der CAN-Bus (*Controller Area Network*) der Fa. Bosch. Ebenso gibt es ein asynchrones serielles Interface sowie den USB-Bus (*Universal Serial Bus*). Um eine spätere Kommunikation mit dem PC zu gewährleisten, wird oft die RS/232-Schnittstelle verwendet. Manche Mikrocontroller besitzen sogar eine IrDA-Schnittstelle, die eine Datenübertragung mittels Infrarotlicht ermöglicht.
Warum sich der Diplomand in diesem Projekt für den I²C-Bus entschieden hat und wie dieser genau funktioniert, wird in Kapitel 6 ausführlich schildert. Auf die Funktionen der anderen Schnittstellen wird an dieser Stelle nicht weiter eingegangen. (Wüst, 2009, S250) [18]

3.11 Bausteine für die Betriebssicherheit

Da Mikrocontroller auch in Steuerungen eingesetzt werden, deren Fehlfunktion gravierende Folgen haben kann, sind sie zum Teil mit speziellen Bausteinen zur Verbesserung der Betriebssicherheit ausgerüstet.

Watchdog-Timer:

Ein Mikrocontroller könnte durch einen versteckten Programmfehler oder durch umgebungsbedingte Veränderungen vom Speicher- oder Registerinhalten, z.B. Störsignale, in eine Endlosschleife geraten. Damit fällt er praktisch aus und die Steuerung ist blockiert. Diese gefährliche Situation soll ein *Watchdog-Timer* (WDT) vermeiden. Ein *Watchdog-Timer* ist ein freilaufender Zähler, der bei Überlauf einen Reset des Mikrocontrollers auslöst. Im normalen Programmablauf muss daher der WDT regelmäßig durch das Programm zurückgesetzt werden, um den WDT-Reset zu vermeiden. Dies kann man z.B. in der Hauptprogrammschleife machen, die ständig durchlaufen wird. Gerät das Programm des Controllers dann durch eine Störung ungewollt in eine Endlosschleife, so findet das Zurücksetzten des Watch-

dog-Timers nicht mehr statt (es sei denn, der Rücksetzbefehl des WDT liegt innerhalb dieser Endlosschleife) und dieser löst nach einer gewissen Zeit ein Reset aus. Nun wird das System neu hochgefahren und initialisiert und kann wieder korrekt arbeiten.
Kritisch ist auch der Ausfall des Ozillatortaktes. Ein *Oscillator-Watchdog* hat eine eigene Oszillatorschaltung. Er vergleicht ständig den Oszillatortakt mit einer eigenen Oszillatorfrequenz und reagiert bei gravierenden Abweichungen, indem er auf den eigenen Oszillatortakt umschaltet und außerdem ein Reset auslöst. (Wüst, 2009, S253) [18]

Power-Up-Timer:

Durch den Power-UP-Timer wird der Mikrocontroller nach dem Anlegen der Versorgungsspannung für die Dauer von 72 ms im Rest-Status gehalten. Der Prozessor beginnt demnach nicht sofort mit der Abarbeitung des Programms, sondern wartet noch eine kurze Zeit. Das ist sinnvoll, wenn die Betriebsspannung an den Chip angelegt wird. Da an der Versorgungsspannung des Gerätes oft recht große Kondensatoren anliegen, die erst vollständig geladen werden müssen, dauert es eine gewisse Zeit bis die volle Betriebsspannung erreicht wird. Aus diesem Grund schaltet man den *Power-Up-Timer* ein. Dadurch wird sichergestellt, dass die richtige Referenzspannung für eine Analog-Digital-Wandlung anliegt. Dauert der Anstieg der Versorgungsspannung auf die volle Betriebsspannung 1 ms, würden bei einem 4-MHz-Takt bereits ca. 1.000 Befehle bearbeitet werden. Daher sollte man bei Problemen am Anfang des Programms zuerst prüfen, ob stabile Zustände am Mikrocontroller vorliegen und der Mikrocontroller nicht für eine kurze Zeit außerhalb der Spezifikationen betrieben wird. Will man den *Power-Up-Timer* aktivieren, geht dies im Programmcode durch die Konstante "_PWRTE_ON." Über "_PWRTE_OFF" wird der Timer ausgeschaltet.

Brown-Out-Protection:

Unter *Brown-Out* versteht man einen kurzzeitigen Einbruch der Versorgungsspannung unter den erlaubten Minimalwert, der aber nicht ausreicht, um ein *Power-on-Reset* auszulösen. Das ist ein kritischer Zustand, da sich z.B. der Inhalt von Speicherzellen verändert haben könnte. Es ist daher sicherer, nach einem *Brown-Out* ein Reset auszulösen, um den Mikrocontroller neu zu initialisieren. Genau dies tut eine *Brown-Out-Protection*. Die Schaltschwelle ist entweder fest vorgegeben oder programmierbar. (Wüst, 2009, S253) [18]

Überprüfung der Versorgungsspannung:

Speziell für batteriebetriebene Geräte ist ein Baustein zur Überprüfung der Versorgungsspannung (*Low-Voltage-Detect, LVD*) gedacht. Dieser soll ein Absinken der Versorgungsspannung schon dann feststellen, wenn die Spannung noch im zulässigen Bereich ist. Das Absinken kann dann durch den noch einwandfrei funktionierenden Mikrocontroller angezeigt werden, so dass noch Zeit bleibt, z.B. den Akku zu wechseln. (Wüst, 2009, S253f.) [18]

Code Protect:

Nach Entwicklung eines Gerätes, kann man die Software gegen unbefugtes Auslesen schützen. Dafür wird das Bit *Code Protection* im Konfigurationsregister auf 0 gesetzt. Jetzt kann der im Chip gespeicherte Programmcode nicht mehr ausgelesen werden und es wird verhindert, dass eine Kopie des Programms erstellt werden kann. Sollen Änderungen am Code vorgenommen werden, muss der Originalcode bearbeitet und erneut in den Speicher geladen werden. Der Leseschutz wird im Quellcode durch "_CP_ALL" aktiviert und über "_CP_OFF" deaktiviert. In diesem Projekt geht es aber nicht darum, in der privaten Wirtschaft Gewinn zu erzielen, sondern die Technologie kommender Empfängers zu verbessern. In diesem Fall muss der Code nicht gegen ein Auslesen geschützt sein. In anderen Bereichen hingegen, wie z.B. der Entwicklung von Entertainment-Produkten, ist dies ein durchaus sehr wichtiges Merkmal.

3.12 Software-Entwicklung

Wenn ein Programm für einen PC entwickelt wird, ist es selbstverständlich, dass auch die Programmentwicklung auf einem PC stattfindet. Der PC hat Bildschirm, Tastatur, Maus, Laufwerke usw. und wird zum Editieren, Testen, Debuggen und Speichern des Programms benutzt. Entwicklungssystem und Zielsystem sind also identisch; der erzeugte Maschinencode kann direkt auf dem Entwicklungsrechner ausgeführt werden. Bei einer Mikrocontroller-Entwicklung sieht die Situation anders aus. Der Mikrocontroller kann nicht als Entwicklungssystem benutzt werden, weil er in der Regel weder Bildschirm noch Tastatur hat. Außerdem wird ein Mikrocontroller nicht größer als nötig ausgewählt, auf dem fertigen System ist daher kein Raum für platzraubende Entwicklungswerkzeuge. Benötigt wird deshalb ein separater Entwicklungsrechner, z.B. ein PC. Das Zielsystem ist aber hinsichtlich Prozessor, Befehlssatz und Peripherie völlig verschieden zum Entwicklungssystem. Der Maschinencode wird durch einen Übersetzter auf dem Entwicklungsboard erzeugt, kann aber nicht mehr auf dem Entwicklungssystem ausgeführt werden. Betrieben wird dadurch eine *cross-platform*-Entwicklung. Aus diesem Grund werden für Mikrocontroller-Entwicklungen spezielle

Werkzeuge benötigt und die meisten Hersteller bieten auch eine ganze Palette davon an. Besonders wichtig sind dabei die Möglichkeiten zur Fehlersuche. Kaum ein Programm läuft gleich fehlerfrei, meistens ist eine längere Phase des Testens und Fehlersuchens nötig.

Die Softwareentwicklung für Mikrocontroller und *Embedded Systems* erfolgte früher überwiegend in Assembler. Assemblersprache bietet den Vorteil absoluter Transparenz und maximaler Kontrolle über das Programm. Ein gut geschriebener Assemblercode ist meist auch kürzer und effizienter als der durch einen Hochsprachen-Compiler erzeugte. Ein Assembler und Linker wird meistens vom Hersteller des Mikrocontrollers mitgeliefert. Bei größeren Programmen wird die Programmierung in Assembler mühsam und unübersichtlich. Fehler sind jetzt schwerer zu finden. Für mathematische Operationen, wie beispielweise das Wurzelziehen, braucht man außerdem Bibliotheken, wenn man nicht gerade ein sehr erfahrener Programmierer ist. Deshalb kommen heute meistens Hochsprachen-Compiler zum Einsatz. Die meistgenutzte Programmiersprache ist *C*, es wird aber auch Basic, Java, Pascal und zunehmend *C++* genutzt. Eine häufige Vorgehensweise ist, das Programm in *C* zu schreiben und anschließend einzelne zeitkritische Abschnitte durch Assembler zu ersetzten. Nach der Übersetzung fügt der Linker die Codeabschnitte zusammen. Compiler sind beim Hersteller des Mikrocontrollers oder bei Drittanbietern erhältlich. (Wüst, 2009, S254f.) [18]

4. Die Entwicklungsumgebung

4.1 Die Programmierung mit MPLAB

Bei der Entwicklungsumgebung MPLAB von Micochip handelt es sich um ein professionelles Tool, das nichts missen lässt, was für die Programmierung der Mikrocontroller benötig wird.
Die aktuelle Software kann auf der Internetseite von Microchip (www.mircochip.com) kostenfrei heruntergeladen und genutzt werden. Mit der Entwicklungsumgebung ist die Programmierung der Controller von Microchip in Assembler möglich. Es können auch verschiedene C-Compiler integriert werden, für die Vollversionen sind allerdings vom entsprechenden Hersteller kostenpflichtige Lizenzen zu erwerben. Der Diplomand benutzt eine Free-Version eines C-Compilers, da die Vollversion des Compilers seines Erachtens nur höher optimierend ist. Für dieses Projekt stehen bereits ausreichend Hardware-Ressourcen zur Verfügung, so dass die Free-Version vollkommen ausreichend.

Um den Mikrocontroller zu programmieren, steht ein Testeditor mit Syntax-Highlighting (farbliche Unterscheidung von Befehlen, Zahlen und Kommentaren) zur Verfügung. Es können über den Texteditor auch Breakpoints (Haltepunkte) definiert und der Code schrittweise abgearbeitet werden. Um sich den aktuellen Inhalt der Register ansehen zu können, stehen unterschiedliche Anzeigearten zur Auswahl. Ein besonderes Merkmal ist der integrierte Simulator. Hiermit kann die Software für einen Mikrocontroller schreiben werden, ohne einen solchen angeschlossen zu haben. Der PC simuliert so die Funktionsweise des Controllers.
Die Signale sind über die Zeit anzeigbar, die Signalzustände der Eingangspins lassen sich ändern und eine Stoppuhr kann verwendet werden, um die Dauer einer Schleife zu bestimmen. Um eine funktionierende Schaltung mit einem Mikrocontroller aufzubauen zu können, müssen die Hardware und die Software ordnungsgemäß funktionieren. Durch den Simulator besteht eine sehr gute Möglichkeit, die Software von der Hardware zu trennen und so lediglich die Software zu untersuchen. Mögliche Fehler in der Beschaltung des Mikrocontrollers spielen dann keine Rolle und die Software wird erst nach erfolgreicher Simulation in den Mikrocontroller geladen. Danach besteht die Möglichkeit, den Mikrocontroller über den In-Circuit-Debugger in echter Hardwareumgebung zu testen und die Fehler zu beseitigen. So können Schritt für Schritt die Fehler gesucht und berichtigt werden. (Hofmann, 2009, S56) [12]

Die Installation wird einfach über den Aufruf der Datei *Install_MPLAB_vXXX.exe* gestartet.

4.2 Anlegen eines Projektes

Nach der erfolgreichen Installation von MPLAB kann die Entwicklungsumgebung gestartet werden. Um ein Programm für einen Mikrocontroller zu erstellen, muss zuerst ein Projekt angelegt werden, in dem der verwendete Mikrocontroller und der Projektname festgelegt werden. Zur Anlage eines Projektes wird der Projekt-Wizard über die Schaltfläche *Project – Project Wizard* gestartet.

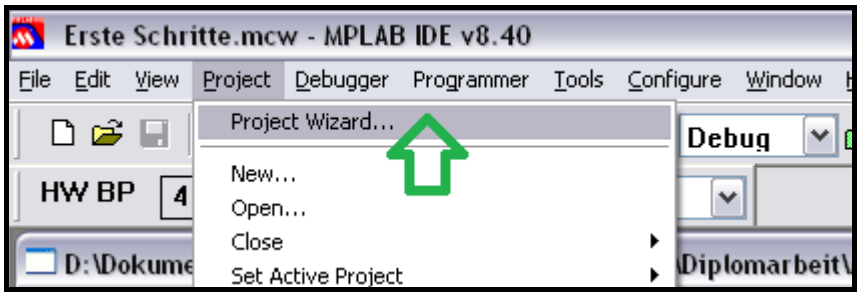


Abbildung 4.1: Project Wizard starten

Die anstehende Begrüßung übergeht man mit *Weiter*.

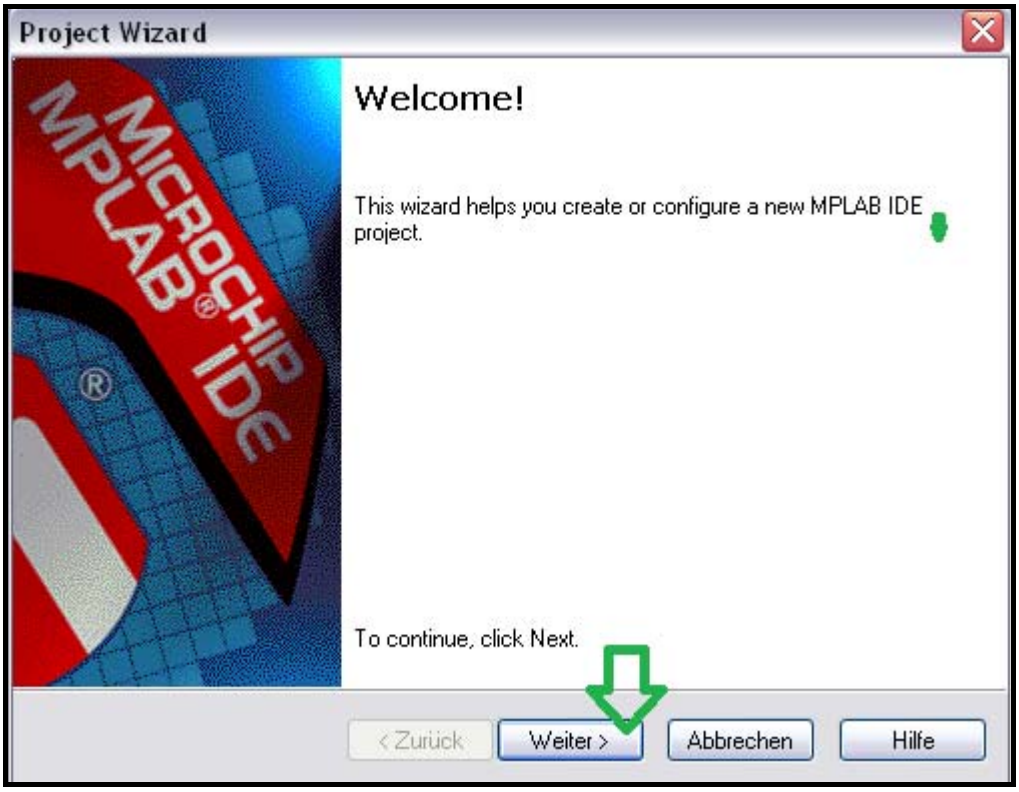


Abbildung 4.2: Project Wizard Welcome

Danach öffnet sich ein Fenster für die Auswahl des verwendeten Mikrocontrollers. Ausgewählt wurde der PIC24FJ64GA002.

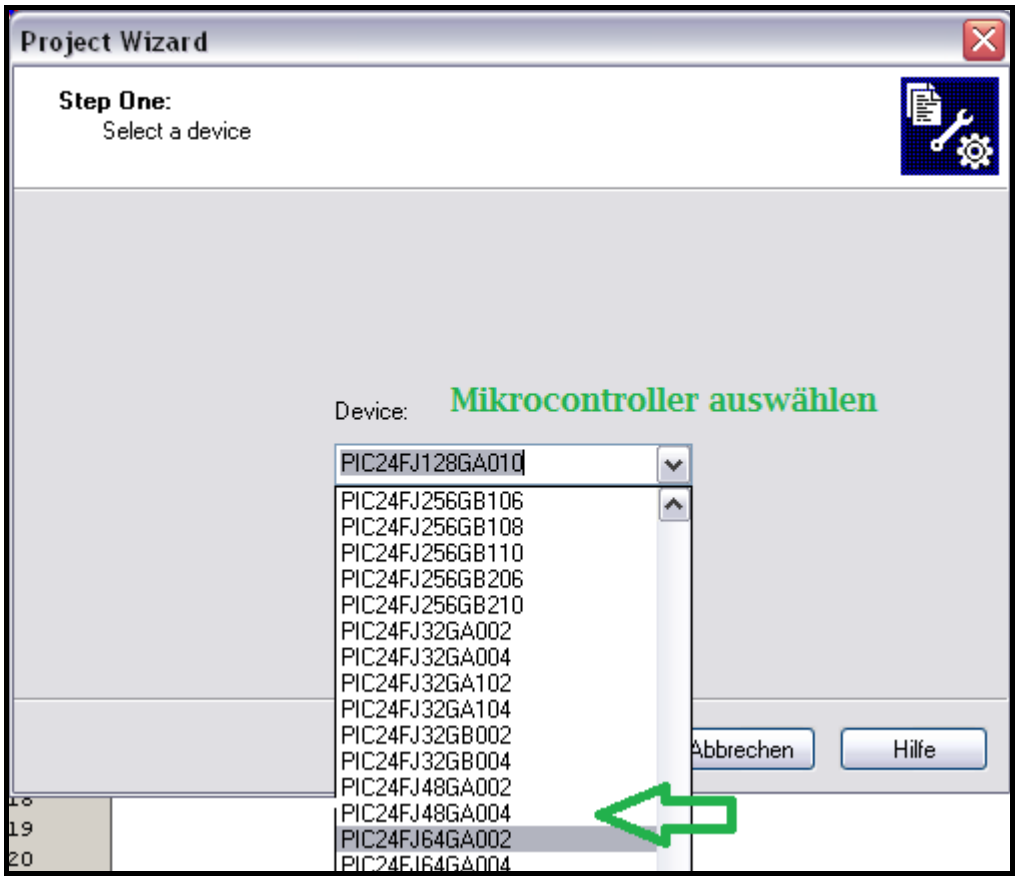


Abbildung 4.3: Auswahl des Mikrocontrollers

Nachdem der Mikrocontroller gewählt wurde, muss festgelegt werden mit welchen Tools das geschriebene Programm übersetzt werden soll.

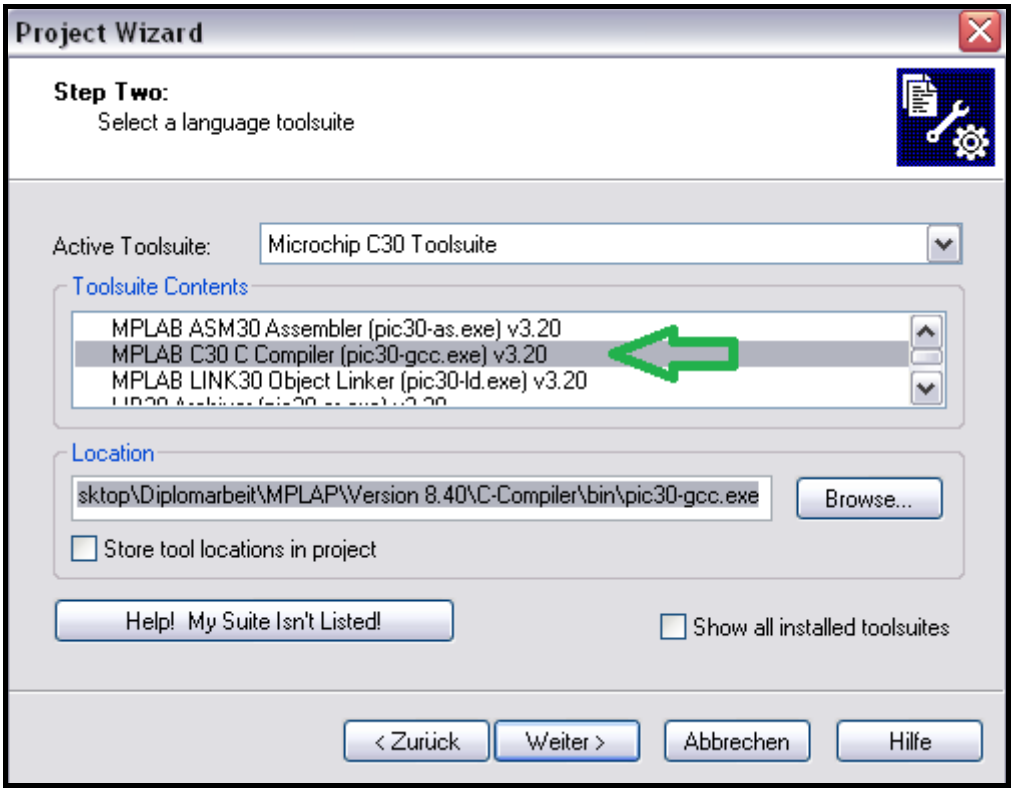


Abbildung 4.4: Auswahl der Toolsuite

Im nächsten Fenster besteht die Möglichkeit, dem Projekt einen Namen zu geben und festzulegen, in welchem Verzeichnis es gespeichert werden soll. Über den Button *Browse* wird mit einem Explorer der entsprechende Ordner ausgewählt und die Projektdatei benannt.

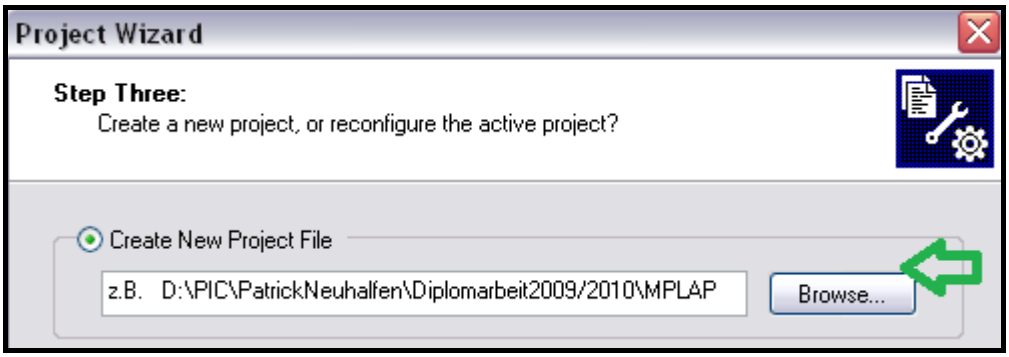


Abbildung 4.5: Neues Projekt anlegen

Sind bereits Dateien aus anderen Projekten vorhanden, können sie mithilfe des folgenden Fensters dem Projekt hinzufügt werden. Dazu sollten die Dateien vorher in das Projektverzeichnis kopieren werden, damit die Projektdateien nicht über die ganze Festplatte verstreut gespeichert werden. Die Dateien können auch zu einem späteren Zeitpunkt in das Projektim-

portiert werden, so dass das Fenster auch ohne Angaben mit dem Button *Weiter* übergangen werden kann.

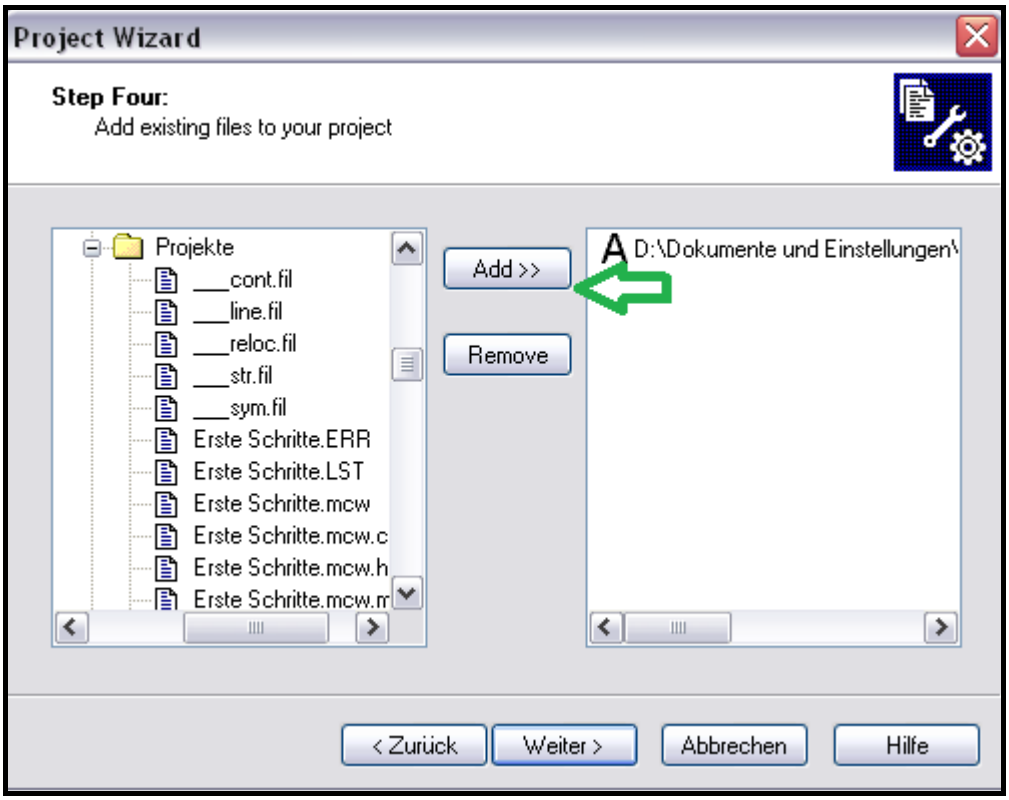


Abbildung 4.6: Dateien hinzufügen

Es wurden nun alle Einstellungen vorgenommen, die erforderlich sind, um ein neues Projekt anzulegen. Es erscheint nochmals eine Übersicht, in der die Einstellungen angezeigt werden. Sind nach einer kurzen Kontrolle die vorgenommenen Angaben richtig, wird das Projekt durch einen Klick auf den Button *Fertigstellen* angelegt. (Hofmann, 2009, S60) [12]

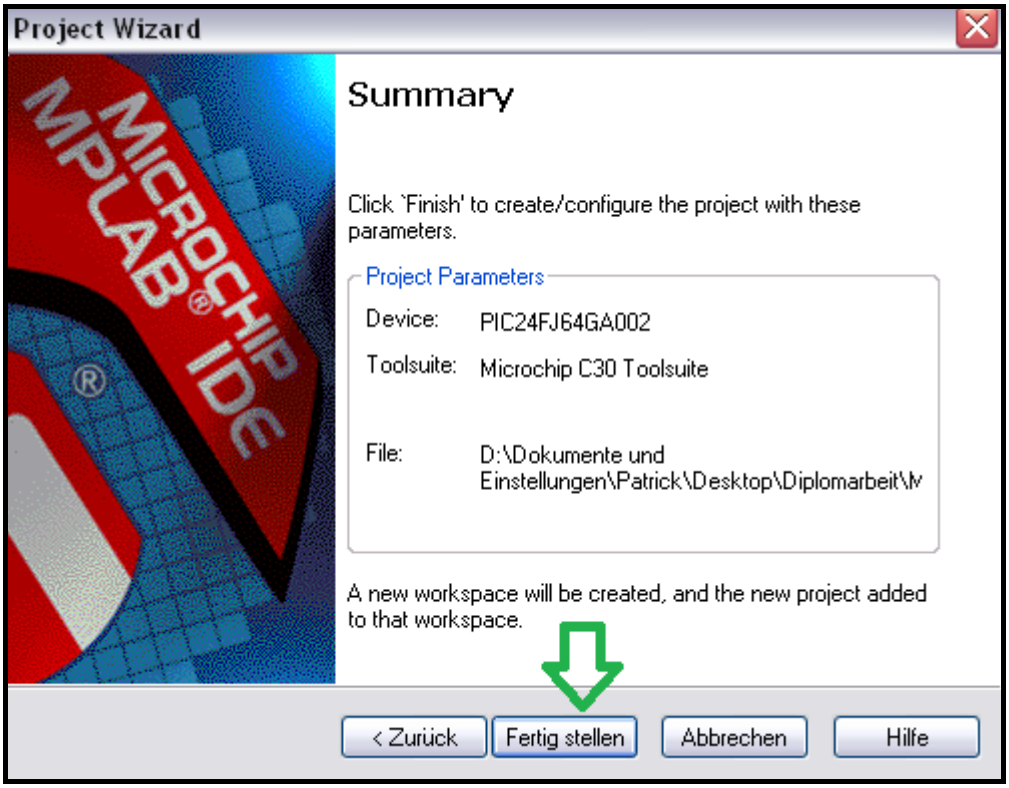


Abbildung 4.7: Zusammenfassung

4.3 Die Arbeitsoberfläche

Die Arbeitsoberfläche ist noch relativ leer. Daher sollten zuerst zwei hilfreiche Fenster sichtbar geschaltet werden. Zum einen ist dies das Projektfenster. Dort sind alle Dateien sichtbar, die zu dem Projekt gehören. Die Dateien können durch einen Doppelklick zum Bearbeiten geöffnet werden. Das Fenster wird sichtbar, wenn im Menüpunkt *View* der Haken vor *Projekt* setzt wird. Ein weiteres wichtiges Fenster ist das *Output*-Fenster. Darin werden alle Meldungen, Fehler und Warnungen angezeigt, die während der Arbeit mit MPLAB entstehen. Dies ist wichtig, um zu erfahren, ob eine korrekte Verbindung zur Hardware besteht und ob der Übersetzungsvorgang erfolgreich abgeschlossen wurde. Mit einem Haken vor *Output* wird auch dieses Fenster aktiviert. Die Fenster können nun in ihrer Größe angepasst und auf der Arbeitsfläche angeordnet werden. Beim Beenden von MPLAB wird diese Anordnung gespeichert, so dass man nach dem erneuten Aufruf des Projekts stets mit der letzten Ansicht starten kann. (Hofmann, 2009, S61) [12]

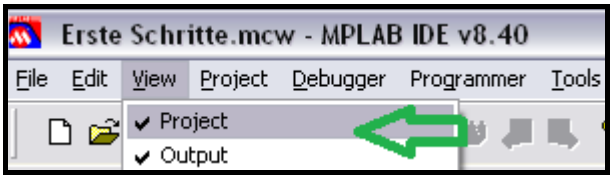


Abbildung 4.8: Projektansicht

Da bei der Generierung des Projekts keine Dateien in das Projekt eingebunden wurden, ist es nun an der Zeit eine neue C-Datei zu generieren. Dazu wählt man aus dem Menü *File* den Punkt *New* aus. Es wird ein Fenster mit dem Namen *Untitled* geöffnet. Das Sternchen hinter dem Namen zeigt an, dass Änderungen in diesem File vorgenommen wurden. In diesem Text-File kann der C-Code eingeben und bearbeitet werden.

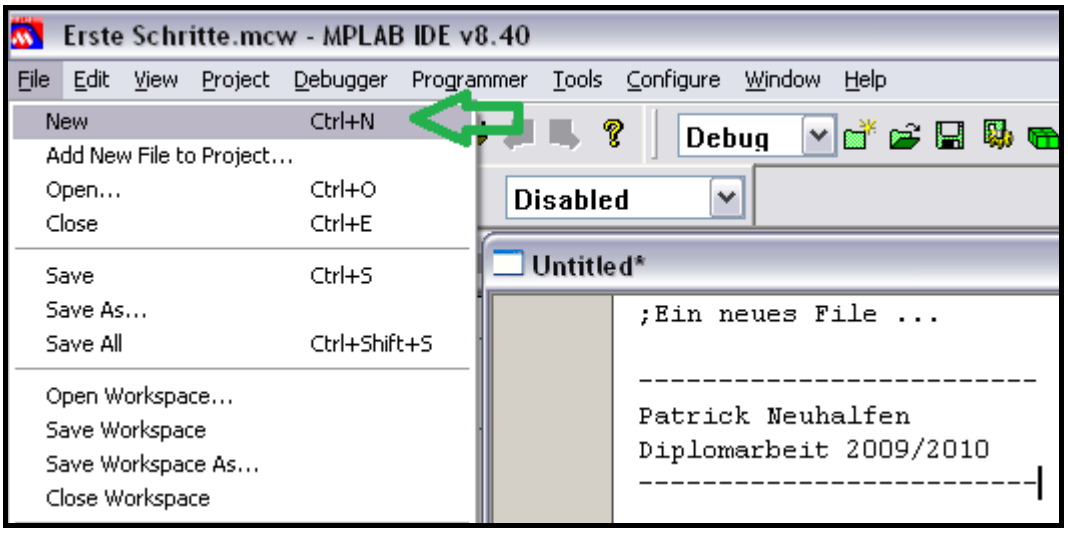


Abbildung 4.9: Neue Datei

Um die Textdatei nach der Bearbeitung zu speichern, wird der Punkt *Save As* aus dem Menü File ausgewählt. Danach wird der gewünschte Pfad ausgewählt und die Datei benannt. Wichtig ist, dass hinter dem Dateinamen die Endung angegeben wird, damit der Compiler später erkennen kann, um welchen Typ es sich handelt. Als Endung für die Dateien, die einen C-Code enthalten, wird die Endung *.c* genutzt. (beim Assemblercode *.asm*)
Für Dateien, in denen Konstanten oder Makros definiert werden, wird die Endung *.inc* (Include) genutzt. Nachdem die Datei unter dem Namen mit Endung abgespeichert wurde, funktioniert auch das Syntax-Highlighting. Für Kommentare, Zahlen und Schlüsselwörter (z.B. while) werden unterschiedliche Farben verwendet. (Hofmann, 2009, S61) [12]

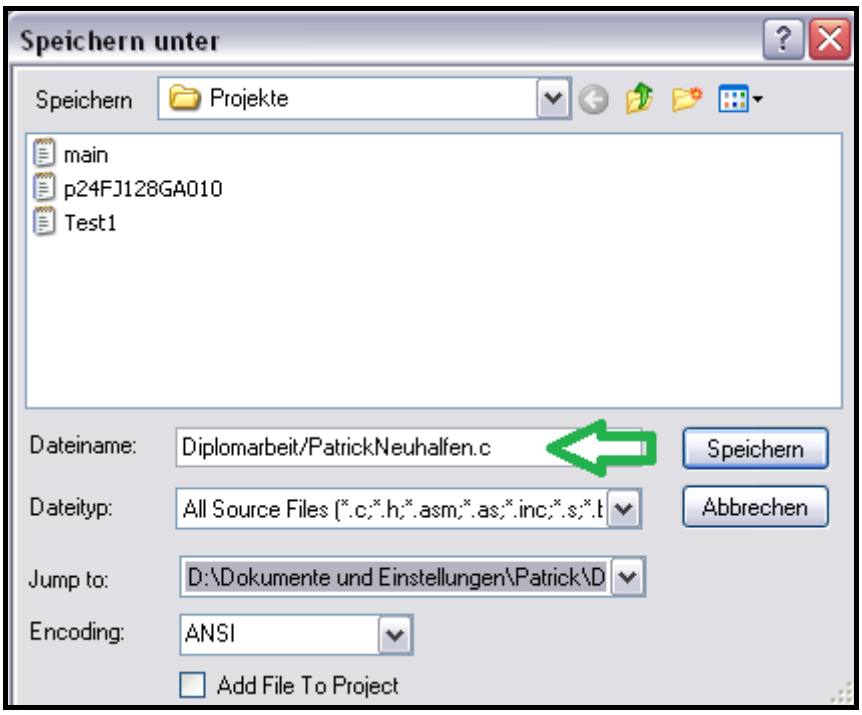


Abbildung 4.10: Datei Speichern

Die Datei mit dem C-Code ist nun gespeichert und liegt auf der Festplatte.
Allerdings ist sie noch nicht im Projekt vorhanden und muss daher noch in das Projekt importiert werden. Dazu wird im Menü *Project* den Punkt *Add Files to Project* gewählt und die gewünschte Datei selektiert. Auf dem gleichen Weg können auch andere bereits erstellte Dateien dem Projekt hinzufügt werden. Die Dateien sollten sich vor dem Hinzufügen zum Projekt im Projektordner befinden, damit alle bearbeiteten Dateien in einem gemeinsamen Verzeichnis stehen. (Hofmann, 2009, S62) [12]

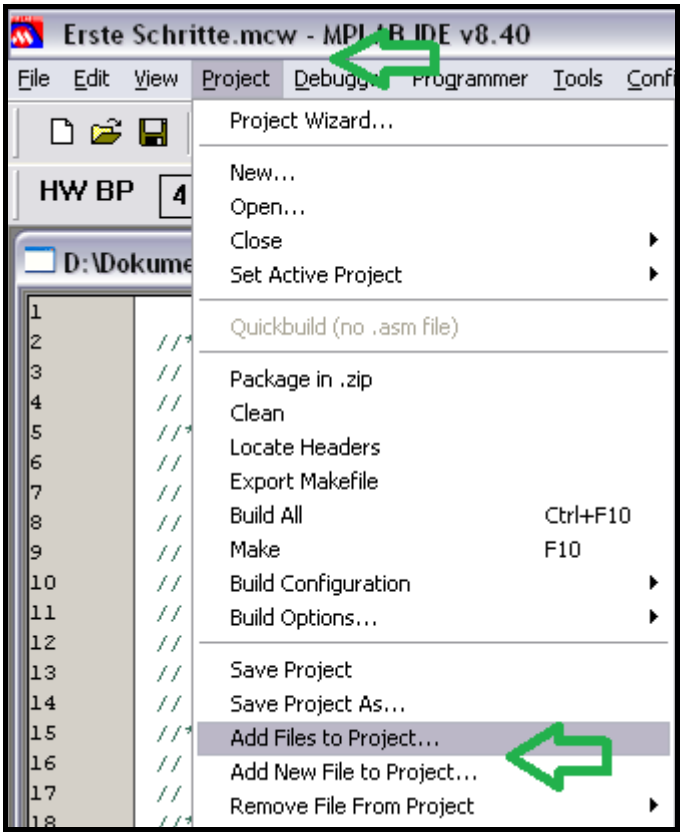


Abbildung 4.11: Dateien zum Projekt hinzufügen

Nachdem die C-Datei erfolgreich in das Projekt eingefügt wurde, taucht sie auch im Projektfenster auf. Hinter dem Projektnamen steht ein Sternchen, welches darauf hindeutet, dass die aktuellen Änderungen noch nicht gespeichert wurden.

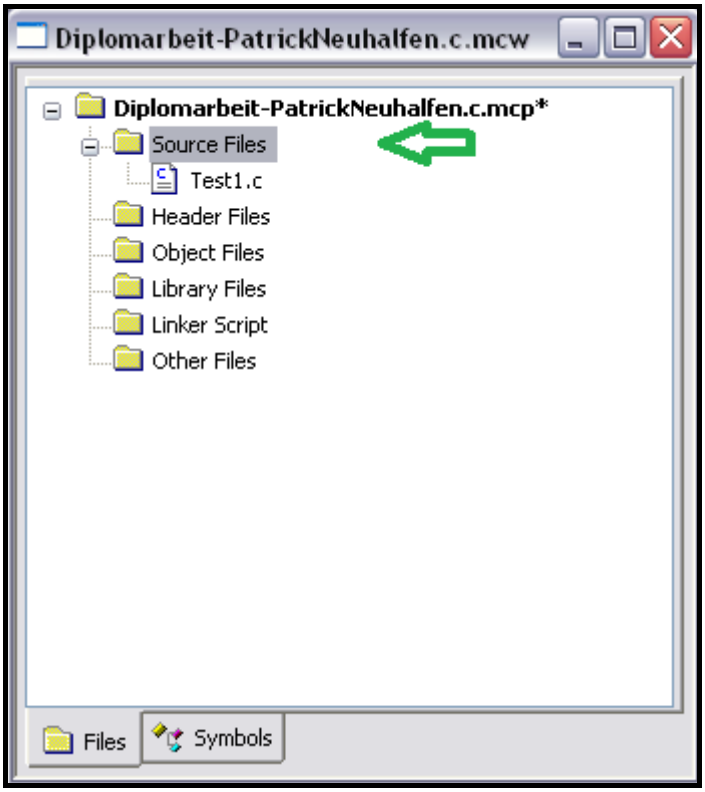


Abbildung 4.12: Projektansicht

Wurde der Code eingegeben und die Datei abgespeichert, ist es an der Zeit das Programm zu testen und eventuelle Fehler zu beseitigen. MPLAB verfügt über einen Simulator, mit dem der PIC simuliert werden kann. Es ist daher nicht erforderlich, einen echten Mikrocontroller an den Rechner anzuschließen. Um den Simulator für die Fehlersuche (Debugger) zu verwenden, wird im Menü *Debugger – Select Tool* der Punkt *MPLAB SIM* ausgewählt.
Es wird ein Haken vor dem aktiven Debugger angezeigt. Soll das Programm in der realen Hardwareumgebung getestet werden, sprich im Mikrocontroller, wird der MPLAB ICD 3 Debugger gewählt. Der Einsatz beider Debugg-Methoden überzeugte den Diplomanden.

Läuft ein Test des Programms vor dem Laden in den PIC in der Umwicklungsumgebung mit dem Simulator fehlerfrei, kann das Programm in den Mikrocontroller geladen und in der realen Umgebung getestet werden. Falls nun ein Fehler auftritt, liegt dies wahrscheinlich an der Hardware.

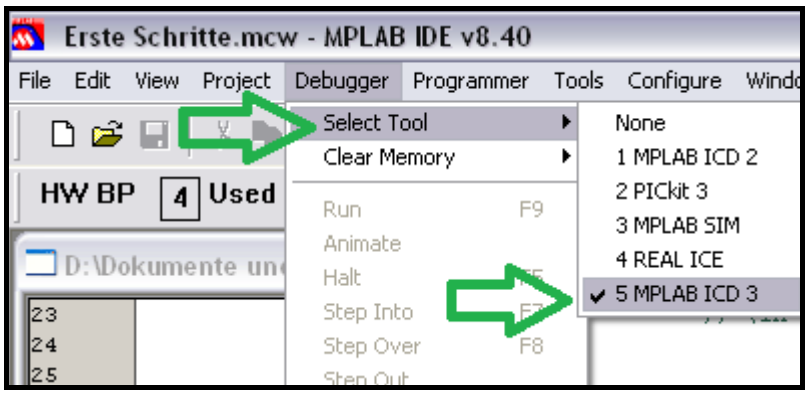


Abbildung 4.13: Auswahl des Tools zum Debuggen

Bevor das Programm ausführt werden kann, muss es übersetzt werden. Um ein lauffähiges Programm zu generieren, wird im Menü *Project* der Punkt *Build All* ausgewählt. Dies funktioniert ebenfalls, wenn ein Button zur Ausführung des Programms geklickt wird (*Run, Step Into*). Wurden seit der letzten Ausführung Änderungen durchgeführt, wird der Programmcode automatisch neu übersetzt.

Wurde alles fehlerfrei programmiert und erfolgreich übersetzt, erscheint im Output-Fenster die Meldung *BUILD SUCCEEDED*. Falls ein Fehler aufgetreten ist, wird er durch einen Doppelklick auf die Fehlermeldung lokalisiert und behoben.

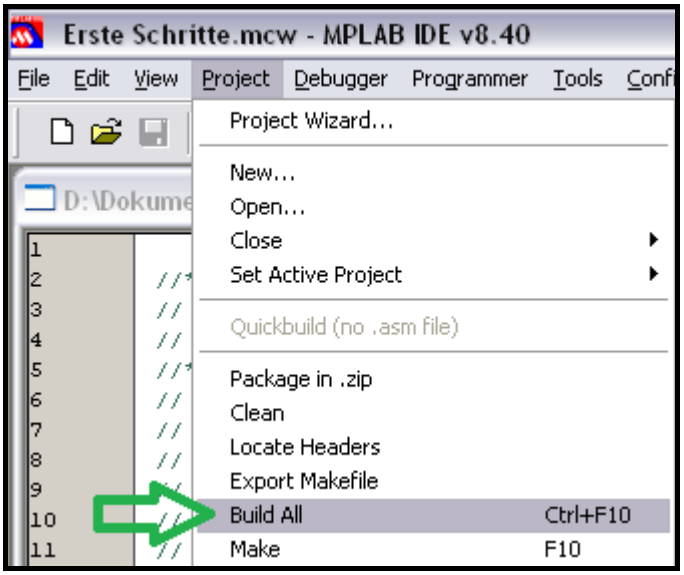


Abbildung 4.14: Projekt übersetzen

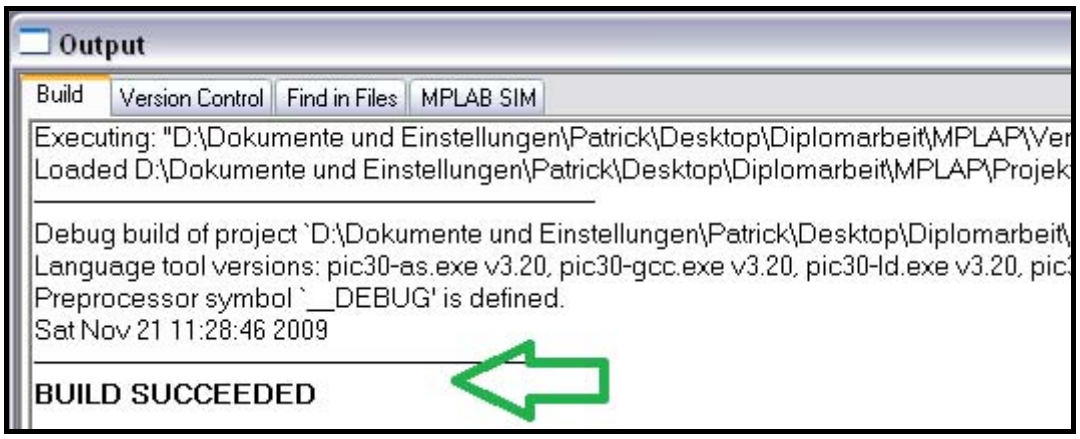


Abbildung 4.15: Erfolgreich übersetzt

Nun ist das Programm lauffähig und kann im Detail analysiert werden.

Nach der Auswahl erscheinen auch zusätzliche Icons in der Symbolleiste. Mit diesen Icons kann der Debugger gesteuert und die Fehler im Programm schrittweise beseitigt werden. (Hofmann, 2009, S62) [12]

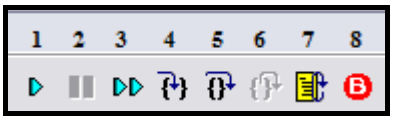


Abbildung 4.16: Buttons für die Simulation

Mithilfe der Icons kann das Programm schrittweise aufgerufen und so die Auswirkung jedes einzelnen Befehls geprüft werden. Die folgende Beschreibung verdeutlicht die Wirkungsweise der Icons:

Run (**1**): Hiermit kann die Ausführung gestartet werden. Die Befehle werden nacheinander abgearbeitet, bis das Programm angehalten wird. Die Ausführung kann durch den Button *Halt* oder einen Breakpoint unterbrochen werden.

Halt (**2**): Bewirkt eine Unterbrechung des laufenden Programms

Animate (**3**): Der Programmablauf wird simuliert und unterscheidet sich nicht von einer schrittweisen Abarbeitung. Aber auf eine einfache Art und Weise wird hiermit erkennbar, an welcher Stelle sich das Programm gerade befindet, und bei Bedarf kann die Ausführung gestoppt werden.

Step Into (**4**): Bei jedem Klick auf dieses Icon wird ein Befehl ausgeführt. Taucht im Programmablauf ein Unterprogrammaufruf auf, wird auch dieses schrittweise ausgeführt.

Step Over (**5**): Analog dem *Step Into* wird jeder Befehl einzeln ausgeführt, allerdings wird ein aufgerufenes Unterprogramm in einem Schritt ausgeführt und es findet kein Sprung für eine schrittweise Bearbeitung in das Unterprogramm statt.

Step Out (**6**): Wurde durch eine schrittweise Ausführung ein Unterprogramm gestartet, können mit diesem Befehl alle Kommandos im Unterprogramm ausgeführt werden und ein Rücksprung ins Hauptprogramm wird möglich.

Reset (**7**): Das Nutzen dieses Icons bewirkt einen Prozessor-Reset und das Programm beginnt von vorn.

Breakpoints (**8**): Hier erscheint eine Übersicht der gesetzten Breakpoints. Diese können ersetzt oder gelöscht werden.
(Hofmann, 2009, S64) [12]

4.4 Das Menü *View*

Zur Analyse des Programms, stehen verschiedene Hilfsmittel zur Verfügung. Diese können über das Menü *View* und durch Auswahl des entsprechenden Punktes genutzt werden.

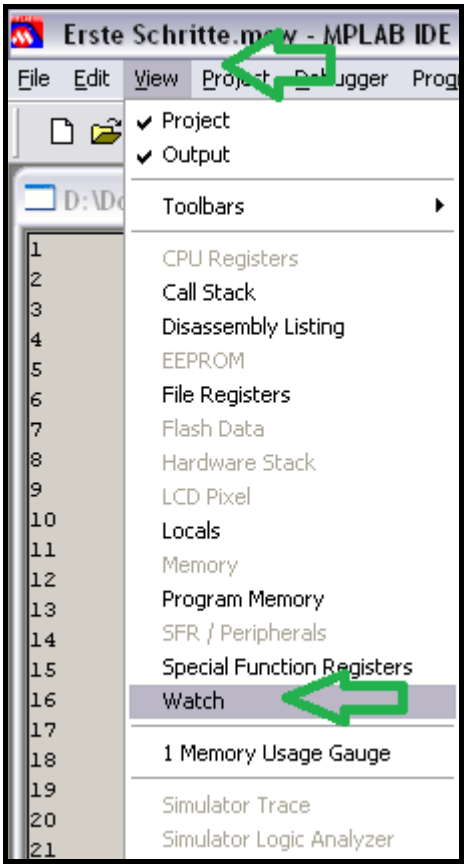


Abbildung 4.17: Das Menü *View*

Watches

Eine wichtige Programmkomponente ist das Fenster *Watch*. Hier erfolgt eine Anzeige des Inhaltes aller Register, die während der Laufzeit verändert werden können. Ändert sich während der Programmausführung der Inhalt eines Registers, wird der Wert in rot dargestellt.

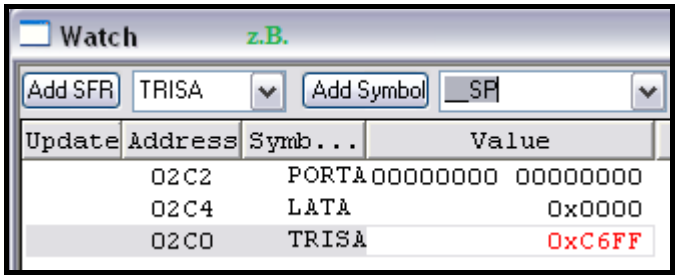


Abbildung 4.18: Watches

Die Register sind in zwei Gruppen eingeteilt:

Die erste Gruppe sind die *Special Funktion Register (SFR)*, die zweiten sind die selbst definierten Register mit ihren Symbolnamen. Zu den *Special Funktion Registern* zählen unter anderem auch das W-Register oder die Port-Register (z.B. PORTA). Um ein neues Register beobachten zu können, wird über das Drop-down-Menü zuerst das entsprechende Register ausgewählt und durch einen Klick auf den Button *Add SFR* oder *Add Symbol* der Liste hinzugefügt. Es ist auch möglich, die Adresse des Registers direkt in die Liste einzugeben. Dazu wird die unterste Zeile ausgewählt und in der Spalte *Address* die Adresse in hexadezimaler Form (ohne 0x) angegeben. Die Reihenfolge der Register kann per *Drag´n Drop* mit der Maus verändert werden. Durch Klicken auf die unteren Reiter *Watch 1 bis 4* ist es möglich, verschiedene Zusammenstellungen der Variablen zu generieren. (Hofmann, 2009, S66f.) [12]

Disassembly Listing

Der Disassembler, der den generierten Maschinencode wieder in einen lesbaren Code zurückwandelt, bietet die Möglichkeit eines Einblicks darüber, wie der programmierte Code übersetzt werden kann und welche Daten in den Mikrocontroller geladen werden. Bedauerlicherweise gehen bei diesem Schritt auch alle definierten Registernamen verloren und es wird lediglich noch die Adresse dargestellt. Im *Disassembly Listing* werden in der linken Spalte die Adresse des Programmcodes und daneben den Maschinencode sichtbar. In der dritten Spalte findet man den zurückübersetzten Code. Falls verfügbar, findet sich auf der rechten Seite der Programmcode, wie er im Original programmiert wurde. Der grüne Pfeil zeigt den aktuellen Stand des Programmzählers an. Bei Unsicherheiten in der Feststellung des Programmstandes bietet das *Disassembly Listing* eine Referenz, da es den Code auch im Mikrocontroller zeigt. (Hofmann, 2009, S67) [12]

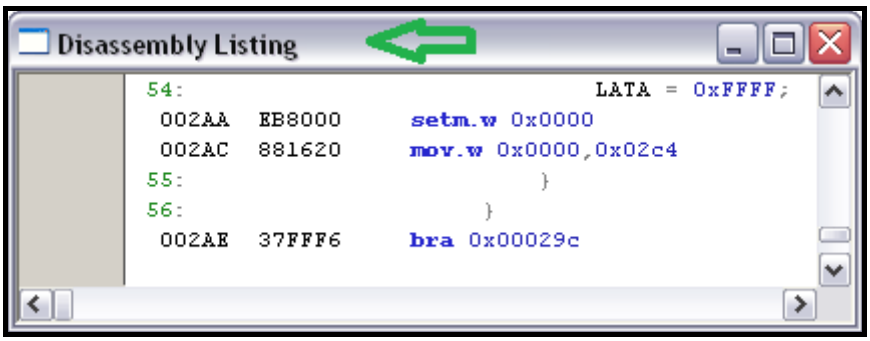


Abbildung 4.19: Disassembly Listing

4.5 Breakpoints

Bei der Programmierung kommt es immer wieder vor, dass man sich bestimmte Codestellen genau ansehen möchte. Um nun nicht das komplette Programm schrittweise mit *Step Into* oder *Step Over* abarbeiten zu müssen, kann ein Breakpoint vor den interessanten Teil des Codes gesetzt werden. Durch die Nutzung des Programms mit *Run* führt der Debugger nun alle Befehle zügig aus und stoppt an der gewählten Stelle. Es gibt verschiedene Möglichkeiten einen Breakpoint zu setzten. Entweder durch Nutzen des Icons *Breakpoints* und Angabe der Position oder durch Doppelklick auf den Befehl in der gewünschten Zeile. Der aktive Breakpoint wird durch einen roten Kreis mit einem weißen *B* gekennzeichnet. Das Programm stoppt an dieser Stelle und die aktuellen Registerwerte werden im Fenster *Watch* angezeigt. Danach kann der Code schrittweise abgearbeitet werden und die Veränderungen der Register können angesehen werden. Im Simulator können beliebig viele Breakpoints gesetzt werden, während im echten Mikrocontroller aufgrund der beschränkten Ressourcen meist nur ein Breakpoint möglich ist. Falls mehrere Breakpoints in einem Programm gewünscht sind, muss der Breakpoint nach dem Erreichen gelöscht und an der nächsten Stelle aktiviert werden. (Hofmann, 2009, S68) [12]

```
25
26  z.B.
27      //============================
28      //  HAUPTPROGRAMM
29      //============================
30
31
32      int main(void)                          // Hier beginnt
33          {
34
35
36  B       TRISA = 0x0000 ;
37          LATA = 0xFFFF ;
38
39          while(1)                            // Dies ist die
40                                              // Dies ist ein
41                                              // In diesem Be
42                                              // Eine solche
43                                              // Ohne diese S
44
45
46
47              {
48  B               if (PORTAbits.RA3 == 1)
49                  {
50                      LATA = 0x0000;
51                  }
```

Abbildung 4.20: Breakpoints

4.6 Simulator

Mit dem zuvor beschriebenen Merkmal ist es möglich, die internen Register anzusehen und deren Veränderung zu interpretieren. Da ein Mikrocontroller in der Regel auch über Eingänge verfügt, müssen diese auch simuliert werden können. Um eine Simulation zu ermöglichen, wird aus dem *Debugger* der Punkt *Stimulus* und *New Workbook* aufgerufen. Es öffnet sich ein neues Fenster mit verschiedenen Reitern (Tabs). (Hofmann, 2009, S69) [12]

Grundeinstellungen:

Bevor der Simulator genutzt wird, sollten noch einige Grundeinstellungen vorgenommen werden. Das Fenster für die Vorgaben wird über das Menü *Debugger* und den Punkt *Settings* geöffnet. Die wichtigsten Einstellungen befinden sich im Tab *Osc/Trace*. Hier muss die gewünschte Taktfrequenz des Mikrocontrollers eingestellt werden, damit die Signale zum richtigen Zeitpunkt angelegt werden können. Bei allen anderen Tabs können die Standardeinstellungen beibehalten werden. (Hofmann, 2009, S69) [12]

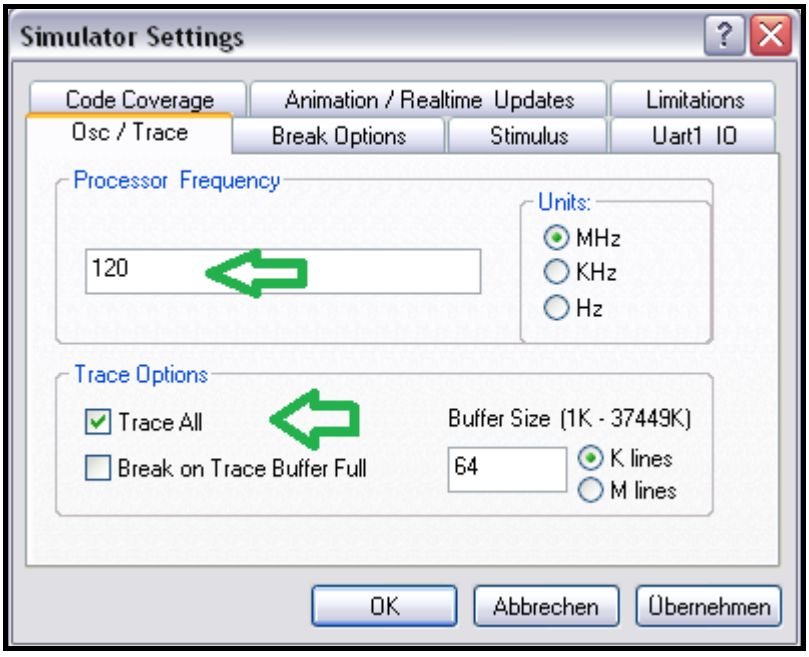


Abbildung 4.21: Simulator Settings

Asynchroner Stimulus

Im Reiter *Asynch* im Fenster *Stimulus* besteht die Möglichkeit, asynchron zum Programmablauf einen Eingangspin auf einen zuvor festgelegten Wert zu setzen. Asynchron bedeutet, dass der Zustand des Pins zu einem beliebigen Zeitpunkt geändert werden kann. Nach der Definition der auszuführenden Aktion und der Auswahl des gewünschten Pins erscheint

unter der Spalte *Fire* ein Button mit einem ">". Wird dieser während der Programmlaufzeit betätigt, wird die Aktion, die in dieser Zeile steht, ausgeführt. In der ersten Spalte *Pin/SFR* wird der entsprechende Eingangspin ausgewählt. Anschließend erscheinen in der Spalte *Action* fünf verschiedene Aktionen.

Toggle: Mit dieser Aktion wird der Signalzustand des entsprechenden Pins umgekehrt. Wenn aktuell ein High-Pegel anliegt, wird nach dem Klicken auf den Button ein Low-Pegel angelegt und umgekehrt.

Set High: Der Eingang wird auf einen High-Pegel gelegt, ungeachtet dessen welcher Pegel vorher anlag.

Set Low: Der Eingang wird auf einen Low-Pegel gelegt.

Pulse High: Der Eingang wird für eine eingestellte Dauer (*Spalte Width und Units*) auf einen High-Pegel gelegt. Danach wird an den Eingang automatisch wieder ein Low-Pegel gelegt.

Pulse Low: Es wird ein Low-Puls für die eingestellte Dauer an dem Pin generiert.

In der Spalte *Comments/Message* sind Kommentare einzutragen, um die Aktionen verständlicher zu gestalten. Die Aktionen können im Einzelschrittbetrieb oder während der Nutzung des Programms ausgeführt werden. (Hofmann, 2009, S70) [12]

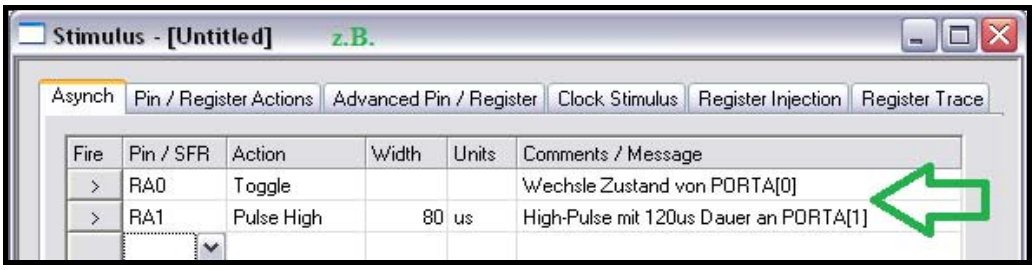


Abbildung 4.22: Asynchroner Stimulus

Zyklischer Stimulus

Um die Eingangspins automatisch zu stimulieren, wird der Tab *Pin/Register Actions* ausgewählt. Hier kann definiert werden, zu welchem Zeitpunkt an dem Eingang ein High- oder Low-Pegel angelegt werden soll. In der Spalte *Time* wird der Zeitpunkt angegeben, zu dem der Signalwechsel erfolgen soll und in den Spalten rechts neben der Spalte *Time* wird der

Pegel festgelegt, der zu diesem Zeitpunkt angelegt werden soll. Das Einfügen von Signalen in die Tabelle erfolgt in der obersten Zeile über *Click here to Add Signals*.

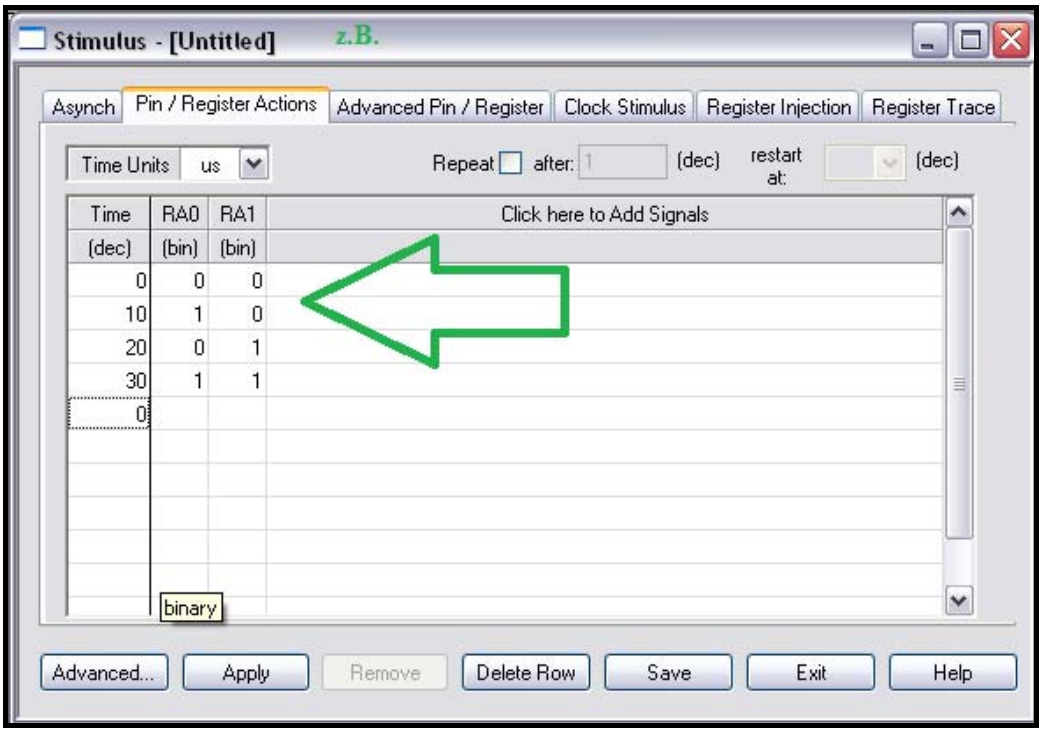


Abbildung 4.23: Zyklischer Stimulus

Es öffnet sich ein weiteres Fenster, in dem die gewünschten Signale auszuwählen sind. In der linken Liste (*Available Signals*) stehen die verwendbaren Signale, die mit dem Button *Add =>* in die Liste der ausgewählten Signale (*Selected Signals*), übernommen werden können. Mit dem Button *Remove <=* werden die Signale wieder aus der Liste entfernt. Die Reihenfolge der Signale kann über die Buttons *Move Up* und *Move Down* verändert werden. Der Button *Move up* verschiebt das selektierte Signal um eine Position nach oben. Mit *Move Down* wird es um eine Position nach unten verschoben. (Hofmann, 2009, S70) [12]

Sonstige Stimulus Tabs

Die Tabs *Advanced Pin/Register, Clock Stimulus, Register Injection* und *Register Trace* sind für das Verständnis der Grundlagen nicht von großer Bedeutung und werden daher an dieser Stelle nicht näher erklärt. Detaillierte Informationen können der Dokumentation von MPLAP entnommen werden.

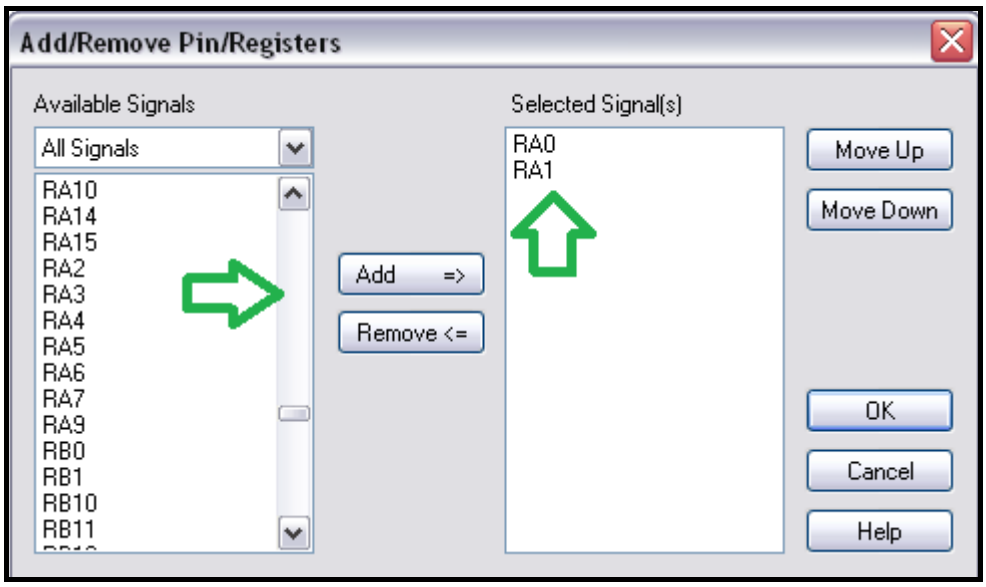


Abbildung 4.24: Hinzufügen von Signalen

Im Tab *Advanced Pin/Register* können Triggerereignisse in Abhängigkeit von vorgegebenen Bedingungen definiert werden.

Der Tab *Clock Stimulus* kann verwendet werden, um Taktsignale an einen Pin anzulegen.

Der Reiter *Register Injection* bietet die Möglichkeit, Werte in ein Register zu laden. Die Werte müssen zuvor in einem Text-File definiert werden.

Um die Änderung der Register zu verfolgen, kann mit Hilfe des Tabs *Register Trace* ein Dateiname angeben werden, in dem die Daten gespeichert werden sollen.

4.7 In-Circuit-Debugger IC3

Da nicht nur eine Simulation eines Mikrocontrollers gewünscht ist, sondern dieser auch in einer Schaltung verwendet werden soll, muss das geschriebene Programm in den Speicher des Mikrocontrollers zu laden werden. Hier gibt es nun eine Vielzahl von Möglichkeiten, mit denen der Flashspeicher beschrieben werden kann. Der Controller kann mit Programmiergeräten der verschiedensten Hersteller programmiert werden. Viele Geräte ermöglichen lediglich das reine Programmieren. Wenn die Software mithilfe des Simulators ausreichend getestet werden kann, ist diese einfache und kostengünstige Variante oft ausreichend. Werden Programme allerdings umfangreicher und soll mit realen Signalen ein Fehler gesucht werden, werden die Grenzen schnell erreicht. Es gibt aber auch die verschiedensten Geräte, mit denen der Mikrocontroller debugged werden kann, während er in einer Schaltung eingebaut ist. Um die Fähigkeiten des Mikrocontrollers voll ausnutzen zu können, wird ein leistungsfähiges Programmier- und Debuggerät benötigt.

Der Diplomand hat sich aus Kompatibilitätsgründen für den In-Circuit-Debugger 3 (ICD3) von der Firma Microchip entschieden. Der ICD3 ermöglicht es, das Programm in den Mikrocontroller zu laden und dann Codezeile für Codezeile ablaufen zu lassen.
Durch den kontrollierten Programmablauf ist es nun möglich, die Registerzustände in Echtzeit zu betrachten und nachzuvollziehen. Ohne dieses Merkmal wäre eine Fehlersuche nahezu unmöglich.
Es gibt im Internet auch zahlreiche Bastelvorschläge für Programmiergeräte, mit denen PICs programmiert werden können. Ein Tool von Microchip hat den Vorteil, dass das Programmiergerät direkt von MPLAB aus verwendet werden kann und keine zusätzliche Software für das Programmieren erforderlich ist. Über das Menü *Configure* und *Select Device* ist eine Übersicht der verwendbaren Programmiergeräte der verschiedenen Mikrocontroller erhältlich. Ein grüner Punkt bedeutet, dass der Mikrocontroller voll unterstützt wird, ein gelber Punkt, dass es nicht komplett getestet wurde und die Ausbauphase noch anhält. Geräte mit einem roten Punkt funktionieren nicht mit dem ausgewählten Mikrocontroller. (Hofmann, 2009, S73f.) [12]

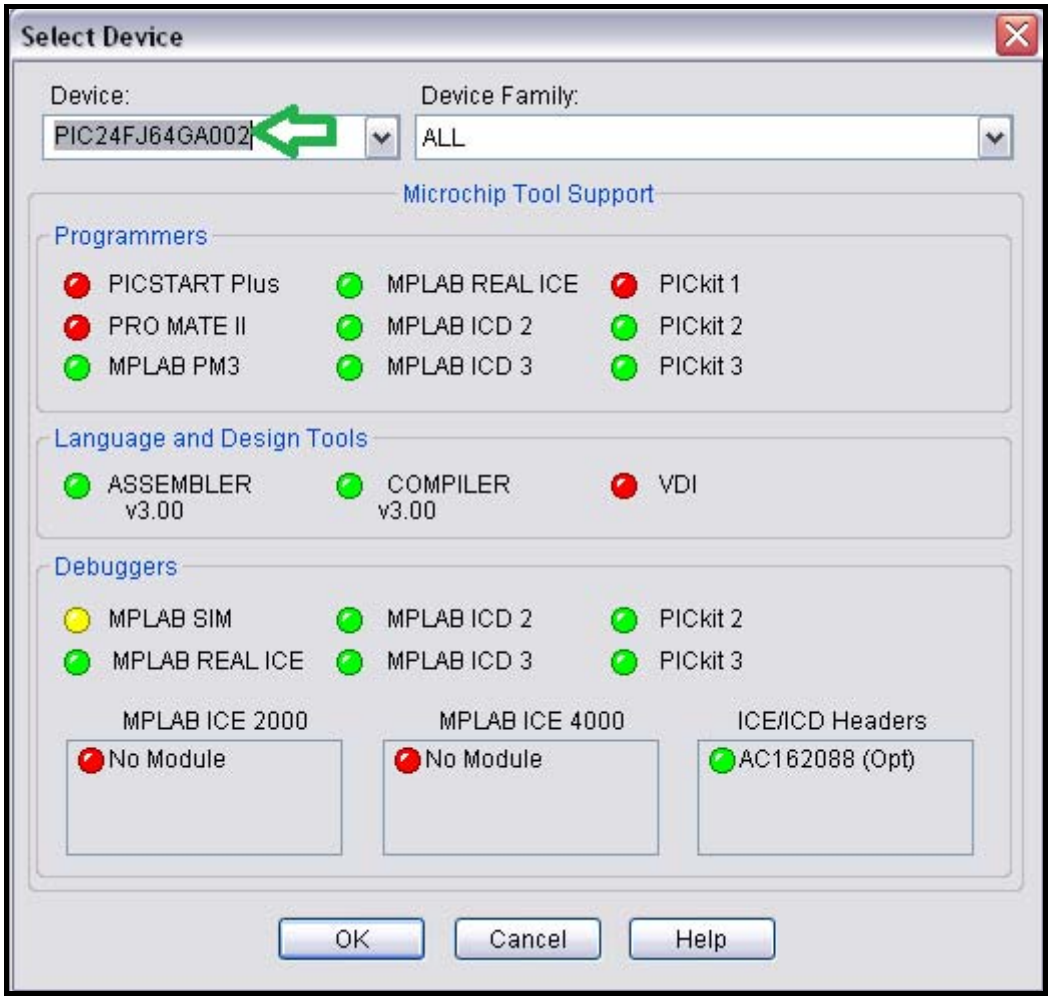


Abbildung 4.25: Auswahl des Mikrocontrollers

Im Folgenden wird die Programmierung und Analyse eines Mikrocontrollers mit dem In-Circuit-Debugger erklärt. Dieser hat eine große Verbreitung gefunden und wird von zahlreichen Entwicklern verwendet.

Nach der Installation kann man der Debugger über das Menü *Debugger*, *Select Tool* durch einen Klick auf *MPLAB ICD 3* ausgewählt werden. (Hofmann, 2009, S74) [12]

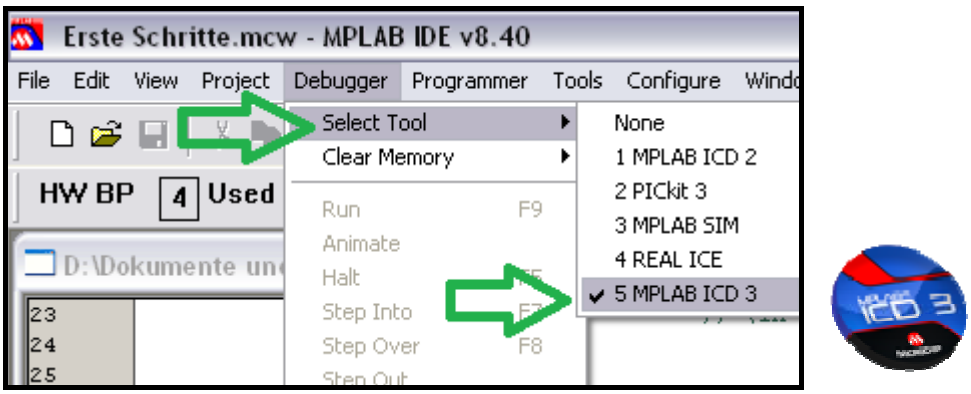


Abbildung 4.26: Auswahl des ICD3 als Debugtool

Bei Auswahl des Menüpunktes während kein Mikrocontroller angeschlossen ist, erscheint die Warnung *Invalid Target Device Id*. Bevor eine Verbindung zwischen ICD3 und dem Mikrocontroller stattfindet, müssen die richtigen Einstellungen vorgenommen werden, damit Probleme vermieden werden. Um die Einstellungen vorzunehmen, sollte zuerst der MPLAB ICD3 Setup Wizard ausführt werden. Dieser nimmt die ersten Grundeinstellungen vor und ist unter dem Menü *Debugger* zu finden. Hier wird der USB-Port als Kommunikationsschnittstelle eingestellt. Im nächsten Schritt wird die Spannungsversorgung des Mikrocontrollers ausgewählt. Soll der Mikrocontroller in einer Schaltung mit einer geringen Stromaufnahme getestet werden, kann wird die Einstellung *Power target from the MPLAB ICD3* verwendet werden. Es sollte sichergestellt sein, dass keine externe Spannung am Mikrocontroller anliegt. In den meisten Fällen findet die Einstellung *Target has own power supply* Anwendung, da so die komplette Hardwareschaltung mit Spannungsversorgung getestet werden kann. Anschließend wird ein Haken bei dem Punkt *MPLAB DIE atomatically connects tot he MPLAB ICD3* und *MPLAB ICD3 automatically downloads the required operating system* gesetzt. Diese beiden Punkte erleichtern die Arbeit mit dem Debugger, da sich die Entwicklungsumgebung MPLAB automatisch mit dem Debugger verbindet und die benötigte Software hinein lädt. Zum Abschluss erscheint eine Übersicht und die Einstellungen können mit dem Button *Fertigstellen* übernehmen werden. (Hofmann, 2009, S74f.) [12]

Jetzt können die Schaltung und der Mikrocontroller mit Spannung versorgt werden und über den Menüpunkt *Connect* im Menü *Debugger* eine Verbindung aufgebaut werden. Wurden alle Verbindungen richtig hergestellt und die Einstellungen vorgenommen, zeigt die Meldung im Output-Fenster an, dass die Verbindung erfolgreich hergestellt wurde.

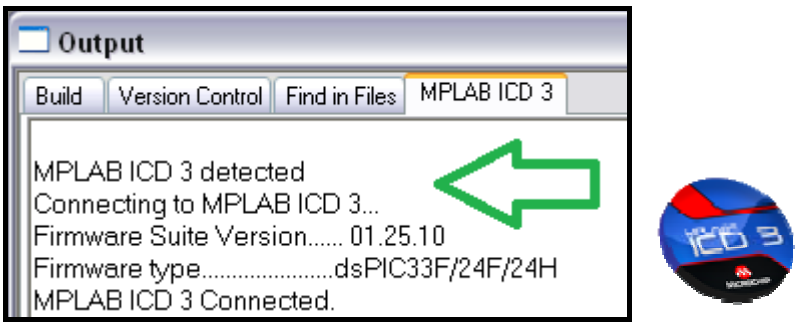


Abbildung 4.27: Informationen im Ausgabefenster

Sollte es dennoch zu Problemen bei der Verbindung kommen, können die Einstellungen über das Menü *Debugger* und den Punkt *Settings*, die mit dem Wizard gemacht wurden, nochmals aufgerufen werden. Im Tab *Status* erscheint eine Übersicht, ob der ICD-Debugger verbunden ist und ob der Selbsttest bestanden wurde.

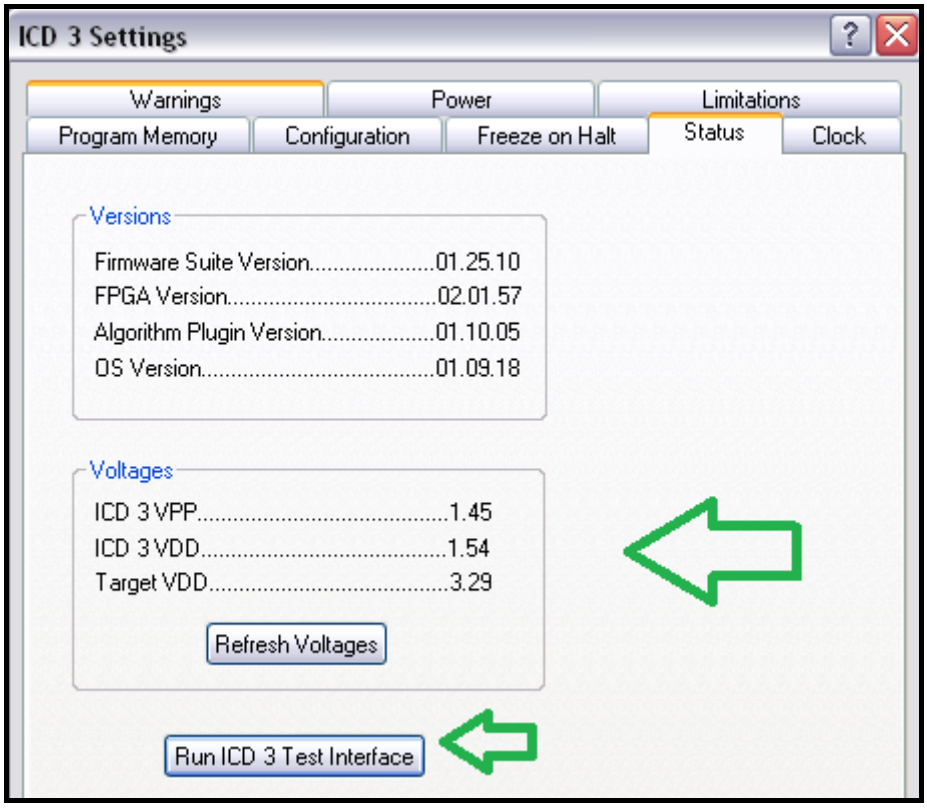


Abbildung 4.28: Status des ICD3

Über den Tab *Power* kann überprüft werden, ob die angelegten Spannungen im richtigen Bereich sind. Bei einem Mikrocontroller mit einer Versorgungsspannung von 5V sollte die angezeigte Spannung Target VDD zwischen 4,5 V und 13,5 V liegen. Die Programmierspannung Target VPP sollte zwischen 9,0 V und 13,5 V liegen. Sie wird vom Programmier-

gerät zur Verfügung gestellt. Über den Button *Update* die Anzeige der Spannungswerte erneuert oder nochmals gemessen werden.

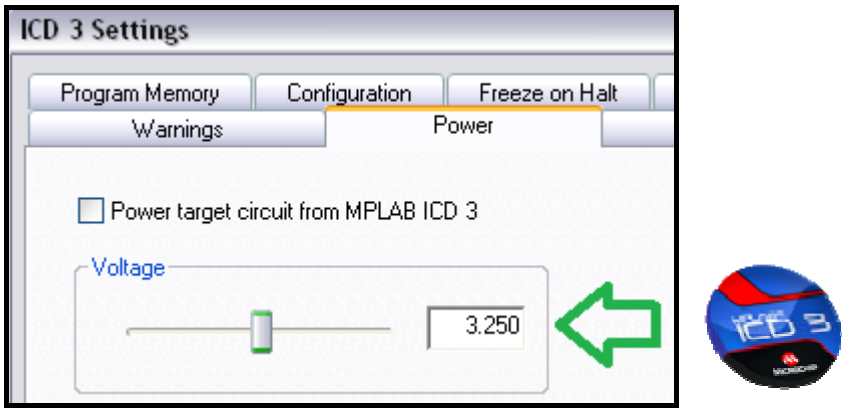


Abbildung 4.29: Spannungsversorgung

Ein weiterer interessanter Tab ist *Program Memory*. Hier erfolgt die Einstellung des Speicherbereiches. Die Selektion des Buttons *Allow ICD3 to select memories and ranges* erlaubt dem ICD3 die Einstellungen selbst vorzunehmen. Das führt dazu, dass lediglich der erforderlich Platz im Flash-Speicher beschrieben wird. (Hofmann, 2009, S77) [12]

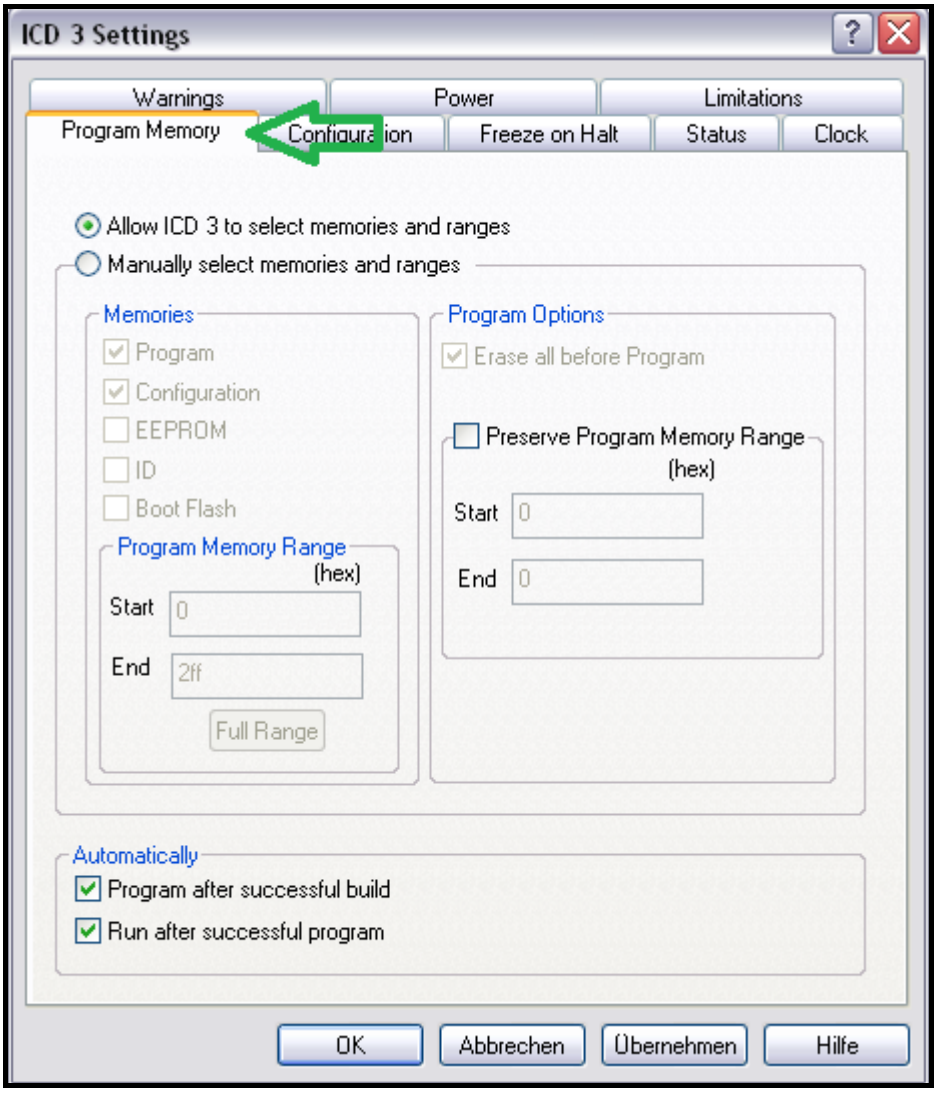
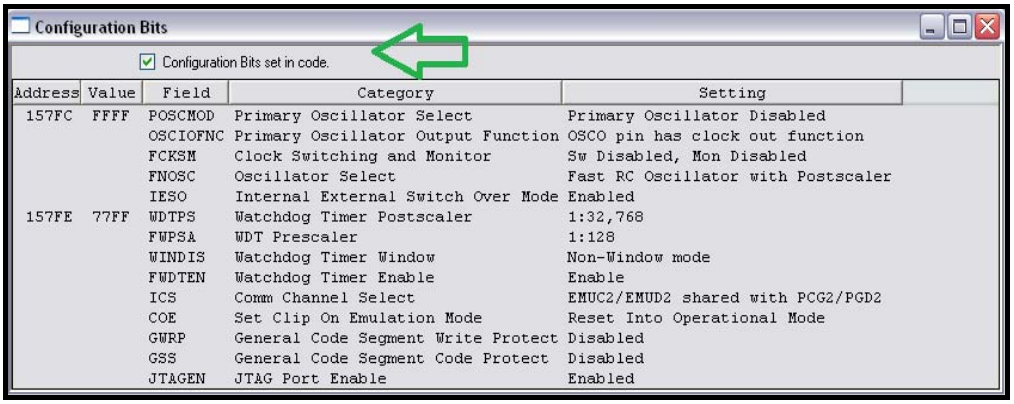


Abbildung 4.30: Programmiereinstellungen

Zum Abschluss können die Konfigurationsbits überprüft werden. Eine Übersicht der aktuellen Werte erhält man über das Menü *Configure* mit dem Punkt *Configuration Bits*. Falls der Chip sich einmal nicht so verhält wie erwartet, sollte zuerst diese Einstellungen überprüft und entsprechend der Anforderungen angepasst werden.

Bei Setzung der Konfigurationsbits im Quellcode, wird der entsprechende Haken in der obersten Zeile des Fensters gesetzt. Weitere Informationen zu den Konfigurationsbits folgen in *Kapitel 5.5*. (Hofmann, 2009, S77) [12]



Address	Value	Field	Category	Setting
157FC	FFFF	POSCMOD	Primary Oscillator Select	Primary Oscillator Disabled
		OSCIOFNC	Primary Oscillator Output Function	OSCO pin has clock out function
		FCKSM	Clock Switching and Monitor	Sw Disabled, Mon Disabled
		FNOSC	Oscillator Select	Fast RC Oscillator with Postscaler
		IESO	Internal External Switch Over Mode	Enabled
157FE	77FF	WDTPS	Watchdog Timer Postscaler	1:32,768
		FWPSA	WDT Prescaler	1:128
		WINDIS	Watchdog Timer Window	Non-Window mode
		FWDTEN	Watchdog Timer Enable	Enable
		ICS	Comm Channel Select	EMUC2/EMUD2 shared with PCG2/PGD2
		COE	Set Clip On Emulation Mode	Reset Into Operational Mode
		GWRP	General Code Segment Write Protect	Disabled
		GSS	General Code Segment Code Protect	Disabled
		JTAGEN	JTAG Port Enable	Enabled

Abbildung 4.31: Konfigurationsbits

Weitere Konfigurationen bezüglich des Debuggens und Programmierens können über den Menüpunkt *Configure* und *Settings* vorgenommen werden. Im Tab *Debugger* sollte der Haken vor *automatically save files before running* gesetzt werden. Dadurch wird sichergestellt, dass die Änderungen vor dem Start des Programms auf der Festplatte gespeichert werden.

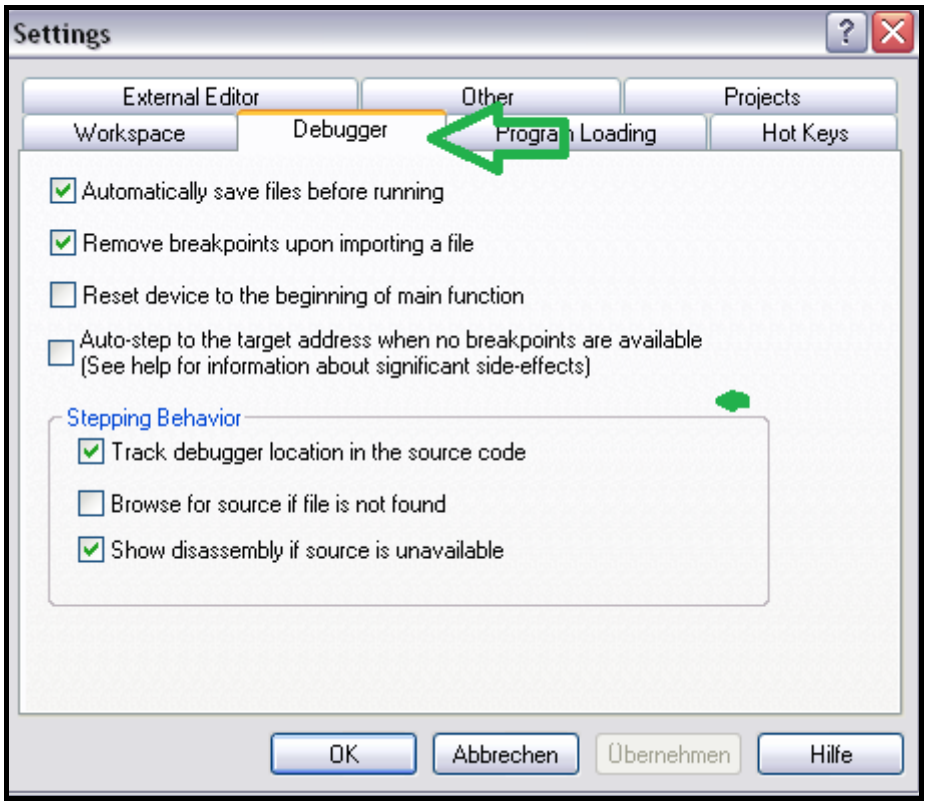


Abbildung 4.32: Einstellungen des Debuggers

Im Reiter *Program Loading* kann außerdem festgelegt werden, welche Speicher vor dem Programmieren des Chips gelöscht werden sollen. Hier sollte ein Haken vor *Clear program*

memory upon loading a program gesetzt werden, um sicherzugehen, dass sich keine alten Programmteile mehr im Flash-Speicher befinden. (Hofmann, 2009, S77) [12]

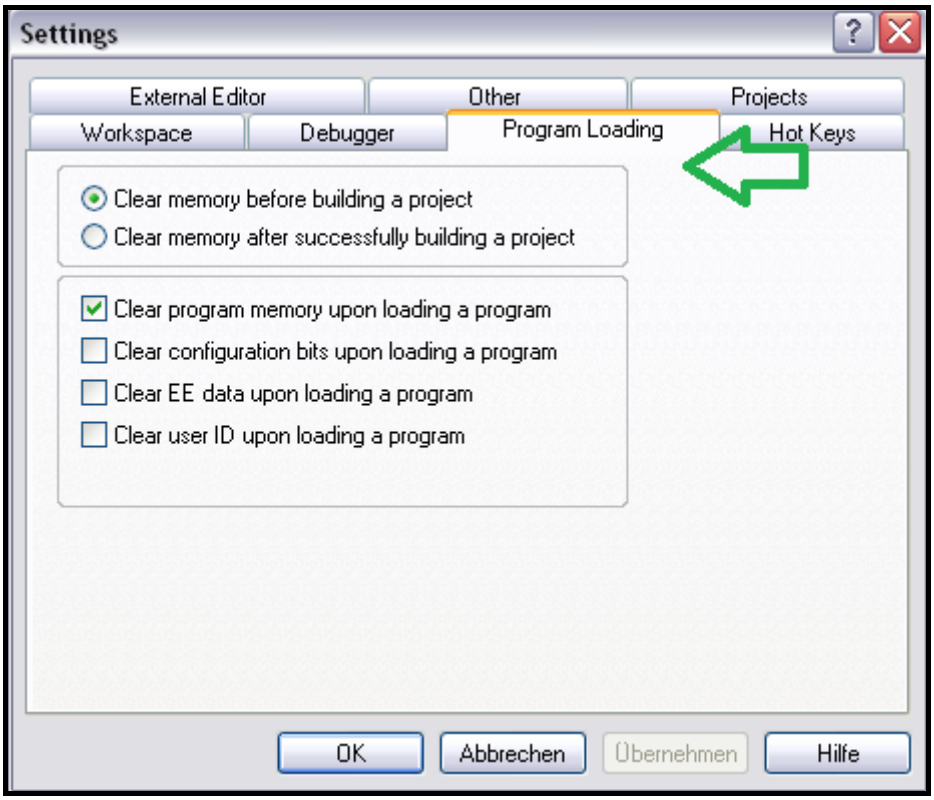


Abbildung 4.33: Einstellungen für die Programmierung

4.8 Programmieren

Wurden nun alle Fehler beseitigt, muss für die Programmierung in den Flash-Speicher der ICD3 in den Programmiermodus versetzt werden. Dazu wird über das Menü *Programmer -> Select Programmer* das Gerät *MPLAB ICD3* ausgewählt. Der ICD3 kann nicht gleichzeitig als Debugger und Programmiergerät funktionieren und muss daher umgeschaltet werden. Soll der Mikrocontroller in der Schaltung eingesetzt werden, die verkauft werden soll, liegt das höchste Interesse darin, dass der programmierte Code nicht von Konkurrenten ausgelesen werden kann. Müssen die Konfigurationsbits vor dem Programmieren entsprechend gesetzt werden. Um den programmierten Code vor dem Auslesen zu schützen, wird das Konfigurationsbit *Code Protect* auf *ON* gesetzt.

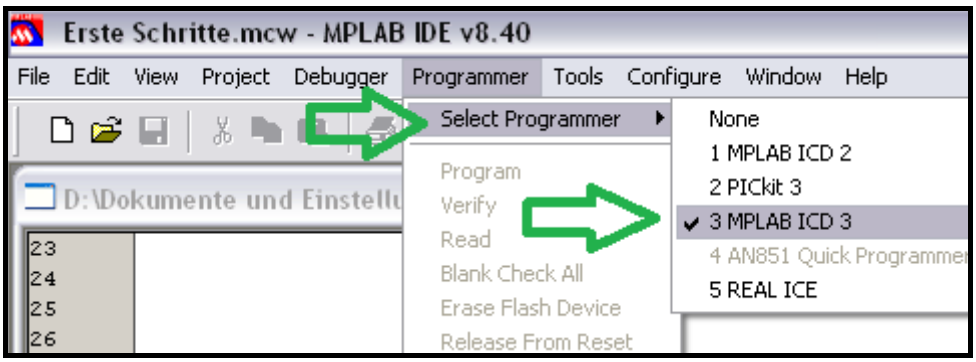


Abbildung 4.34: ICD3 als Programmiergerät

Nachdem der ICD3 als Programmiergerät ausgewählt wurde, wird eine Verbindung zwischen MPLAB und den ICD3 hergestellt, der angeschlossene Mikrocontroller gesucht und ein Selbsttest durchgeführt. Ob die Verbindung einwandfrei hergestellt wurde, kann man anhand der Meldungen im Output-Fenster verfolgen. Wurde kein Fehler bei dem Verbindungsaufbau gemeldet, kann der Chip über den Menüpunkt *Programmer -> Program* programmiert werden. Die Meldungen geben Auskunft über die Aktion, die gerade ausgeführt wird. Zuerst werden die Konfigurationsbits geprüft, der Speicher gelöscht und das neue Programm in den Flash-Speicher geschrieben. Wurde das Programm in den Speicher geschrieben, wird nochmals geprüft, ob alle Daten auch richtig geschrieben wurden (Verifing). Zum Schluss werden noch die Konfigurationsbits in den Mikrocontroller geschrieben und geprüft. Nach der Meldung *Programming Succeeded* ist der Mikrocontroller einsatzbereit und kann in der endgültigen Schaltung eingesetzt werden. (Hofmann, 2009, S80) [12]

4.9 Texteditor

Der Diplomand nutzt den in MPLAB integrierten Texteditor. Sollten beim Öffnen einer Datei die Befehle nicht gleichmäßig untereinander stehen, wurde vermutlich der Wert für einen Tabulatorschritt falsch eingestellt. Um eine übersichtliche Darstellung zu erhalten, sollte über das Menü *Edit* der Punkt *Properties* aufgerufen werden und im Tab *'C' File Types* der Wert für Tab *Size* auf 3 gesetzt werden. In diesem Tab kann auch das *Highlight Matched* aktiviert werden sowie die Zeilennummern ein- und ausgeschaltet werden. Ebenfalls kann hier das Setzten eines Breakpoints per Doppelklick auf die gewünschte Zeile aktiviert werden. (Hofmann, 2009, S81) [12]

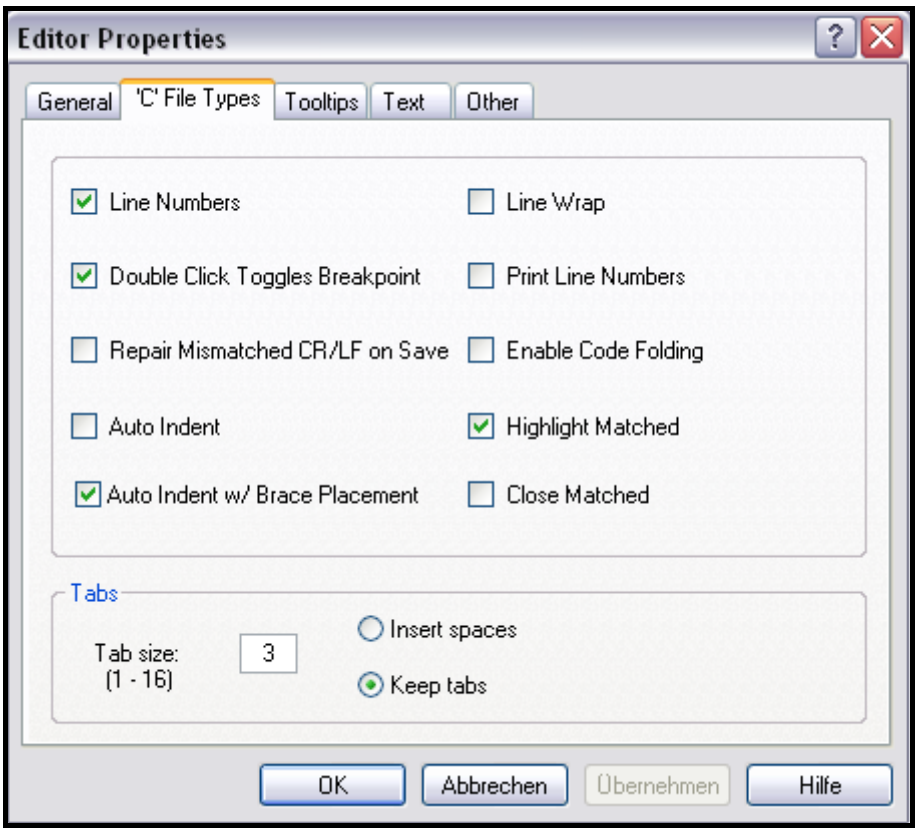


Abbildung 4.35: Einstellungen des Texteditors

5. Rund um die Programmierung

Nach erfolgter aufwändiger Einrichtung der Hardware und Software kann nun mit der Programmierung begonnen werden.
Der erste Test gilt dem Hardware-PIM-Sockel. So lässt sich herausfinden, ob der Controller über den entworfenen Sockel programmierbar ist. Hierzu plante der Diplomad eine Generierung eines Ausgangsignals, welches sich zwischen 0 V und 3 V umschalten lässt. Dieses Signal könnte zum Beispiel ein LED leuchten lassen.
Wie kann die Programmierung gestartet werden?
Was ist wichtig und was muss berücksichtigt werden?
Durch den selbstgebauten PIM-Sockel besteht nun die Möglichkeit, den Mikrocontroller über das *Explorer-16-Testboard* zu programmieren.
Da der Mikrocontroller im fertigen RF-Modul als SMD-Bauform fest verlötet sein wird, muss dennoch eine Möglichkeit bestehen den Mikrocontroller zu programmieren. Deshalb wird in diesem Kapitel erklärt, wie der Mikrocontroller mit einem Programmiergerät verbunden werden kann und wie die endgültige Schaltung beschaffen sein muss, damit das geschriebene Programm in der fertigen Hardware funktioniert.

Um das Programm in den Mikrocontroller zu laden, werden 3-4 Pins zur Verfügung gestellt. Diese können bei Bedarf auch nach dem Programmieren für andere Zwecke als Eingang oder Ausgang verwendet werden. Es handelt sich dabei um folgende Pins:

PGC = Programming Clock = Takt für die Programmierdaten
PGD = Programming Data = Daten für die Programmierung
VPP = Progamming Voltage = hohe Programmierspannung ca. 11 bis 13 V
PGM = Low-Voltage Programming Enable = Freigabe-Pin, damit der Mikrocontroller ohne hohe Programmierspannung programmiert werden kann.

Neben diesen Pins werden noch weitere für die Versorgungsspannung (VDD und VSS) benötigt. Falls der Mikrocontroller nicht in einer Schaltung, an der eine Betriebsspannung anliegt, angeschlossen ist, muss die Versorgungsspannung für die Programmierung von dem Programmiergerät zur Verfügung gestellt werden.

5.1 Programmieren mit dem In-Circuit-Debugger

Die grundlegenden Eigenschaften eines In-Circuit-Debugger wurden bereits in *Abschnitt 4.7* ausführlich erläutert. Aber wie muss die Verbindung zwischen PC, ICD3 und dem Testboard aufgebaut sein? Und was muss beachtet werden, um eine reibungslose Kommunikation zu gewährleisten?
Daher soll im folgenden Abschnitt erklärt werden, wie der Programmcode von der Entwicklungsumgebung MPLAB (PC) in den Mikrocontroller (Testboard) gelangt.

Der ICD3 muss zur einen Seite via USB-Schnittstelle mit dem PC verbunden werden.

Abbildung 5.1: Die Programmierschnittstelle (1)

Auf der anderen Seite muss eine Verbindung zwischen dem ICD3 und seiner Programmierschnittstelle mit der Hardwareschaltung bestehen.
Wie dies im Detail aussieht, zeigen die *Abb. 5.1, 5.2 und 5.3*.

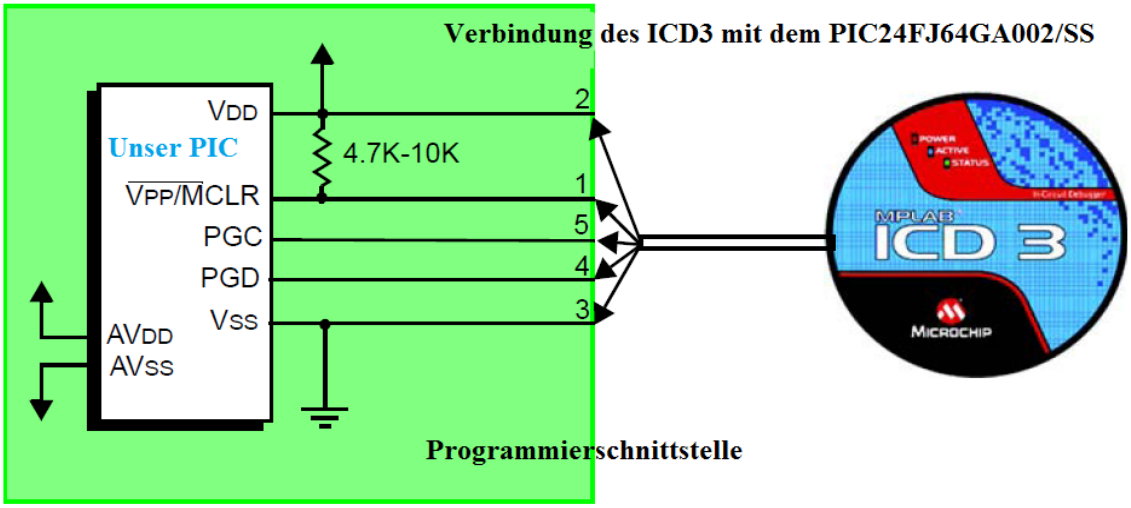


Abbildung 5.2: Die Programmierschnittstelle (2)

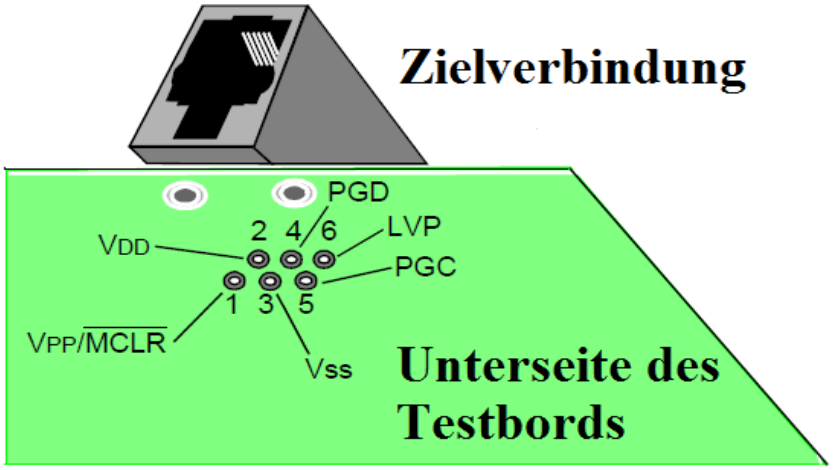


Abbildung 5.3: Die Programmierschnittstelle (3)

Um Fehler beim Programmieren und Debuggen zu vermeiden, sollten nach Möglichkeit die Pins PGC und PGD ausschließlich für Programmierzwecke verwendet werden. Ist es aus irgendwelchen Gründen trotzdem erforderlich, diese Pins für andere Zwecke zu benutzen, sollten nach Möglichkeit weniger wichtige Merkmale (z.B. Warnung, wenn die Batteriespannung einen bestimmten Wert unterschreitet) oder einfache passive Schaltungen (z.B. Taster) an diese Pins angeschlossen werden. Im schlimmsten Fall kann der Mikrocontroller in der Schaltung nicht *debugged* werden. Der Pin PGM ist zwar für die Programmierung nicht erforderlich, es sollten aber auch hier einige Punkte beachtet werden. Über diesen Pin wird dem Mikrocontroller mitgeteilt, dass das Programmieren mit geringer Spannung, also nicht mit 11 bis 13 V an VPP, erfolgt. Um den Baustein mit lediglich 5 V zu programmieren, muss zuerst das Konfigurationsbit *Enable Low Voltage Programming* gesetzt werden. Liegt

dann ein High-Pegel am Pin PGM an, kann der PIC auch ohne hohe Programmierspannung beschrieben werden. Dies kann sinnvoll sein, wenn der Programmcode im fertigen Gerät aktualisiert werden soll. Zur Vermeidung von Fehlern bei der Programmierung, sollte dieser Pin nur als Ausgang verwendet werden, der auf einen Eingang eines externen Bauteils geht. Zusätzlich kann ein Pull-Down-Widerstand von ca. 10 kOhm angeschlossen werden, damit der Pin erst auf High-Pegel gezogen wird, wenn dies auch gewollt ist. Wird der Ausgang eines externen Schwellwertschalters an diesen Pin angeschlossen, ist nicht sicher sein, welcher Pegel zum Programmierzeitpunkt an diesem Pin anliegt. Über den Eingang *MCLR/VPP* wird die hohe Programmierspannung an den Mikrocontroller angelegt. Ist diese Spannung größer als 9 V, wird der internen Hardware mitgeteilt, dass der Flash-Speicher programmiert werden soll. Nach dem Programmieren wird dieser Pin als Reset-Pin verwendet. Wird der Pin also während des Betriebs auf Low-Pegel gelegt, wird ein Reset ausgeführt und das Programm beginnt von vorne. Um einen unbeabsichtigten Reset zu vermeiden, muss dieser Pin immer auf High-Pegel liegen. Der Pin darf allerdings nicht direkt mit der Betriebsspannung VDD verbunden werden, da sonst auch die hohe Programmierspannung an dem Spannungsversorgungspin anliegen würde und den Baustein zerstören könnte. VPP muss daher über einen Widerstand mit VDD verbunden werden. Der Widerstand hat in der Regel einen Wert von 10 kOhm. Über einen Taster oder Jumper nach Masse kann so ein kontrollierter Reset ausgeführt werden. (Hofmann, 2009, S82f.) [12]

5.2 Das Programm macht sich auf den Weg

In diesem Abschnitt wird die Auswahl der Programmiersprache begründet. Außerdem wird erklärt, wie der entwickelte Quellcode funktionsfähig in den Mikrocontroller gelangt.

Die theoretischen Grundlagen der Hochsprache C gehen weit über den Rahmen dieser Dokumentation hinaus. Daher erfolgt an dieser Stelle für alle Interessierten ein Verweis auf das Literaturverzeichnis im Anhang [6, 14, 15].

Außerdem möchte sich der Diplomand auf einen Leitfanden für den Einstieg in die Materie der Mikrocontroller fokussieren.

Zur Frage, wie die Daten generiert werden und dann in den Mikrocontroller übertragen werden könne, lässt sich Folgendes sagen: Leider versteht der Mikrocontroller keine Programmiersprachen wie Basic, C oder Assembler. Der Mikrocontroller kann lediglich mit Nullen und Einsen beschrieben werden, welche anschließend im Flash-Programmspeicher abgelegt werden. Da der Flash-Speicher ein nichtflüchtiger Speicher ist, der die Daten nach dem Ausschalten behält, kann der Programmcode hier gespeichert werden. Eine Speicherung der

Daten im RAM-Speicher, würde dazu führen, dass das Gerät nach dem Ausschalten nicht mehr funktioniert.
Um nun Daten zu generieren, die der Mikrocontroller versteht, sind mehrere Schritte notwendig. Prinzipiell könnte der Programmcode schlicht in binärer Form geschrieben und mit einem Programmiergerät in den Controller geladen werden. Dies wäre allerdings nahezu unmöglich zu verstehen und nachzuvollziehen. Daher werden verschiedene Sprachen für die Programmierung genutzt. Am Gebräuchlichsten für die Mikrocontrollerprogrammierung ist C oder Assembler. Beide Sprachen sind hardwarenah und können daher direkt mit den physikalisch vorhandenen Registern kommunizieren. Assembler hat sich in der Zeit beliebt gemacht, in der Rechenleistung und Speichplatz relativ knapp waren. Ein in Assembler erstellter Code ist fast immer schneller und benötigt weniger Speicherplatz, ist aber auch unübersichtlicher. In einigen Bereichen wird die Assemblersprache auch heute noch eingesetzt, z.B. bei der Programmierung für Grafikkartentreiber, bei welcher zeitkritische Stellen von enormer Bedeutung sind. Aber auch für sehr kleine Mikrocontroller, die aus wirtschaftlichen Gründen in der Rechenleistung und Speicherplatz sehr stark eingeschränkt sind.
Da in den letzten Jahren die Rechenleistung und der Speicherplatz rasant gestiegen sind, werden in diesem Bereich kaum noch Grenzen erreicht. Deswegen hat sich für Mikrocontroller die Hochsprache C als Standard etabliert. Gerade bei größeren Projekten sorgt C für erheblich mehr Übersicht und vermeidet somit viele Fehler. Da die Anforderungen an das System dieser Arbeit nicht geschwindigkeitsorientiert sind und der Speicherplatz des genutzten Controllers mehr als ausreichend ist, stand von Beginn an fest, dass die Programmiersprache C für das Projekt genutzt wird.

Hier ein kleines Beispiel eines Zählers der von 9 nach 0 herunter zählt:

C-Code:	**Assembler:**	**Maschinencode:**
for (i=9; i>0;i--);	movlw 0x09	000C 3009
	movwf 0x20	000D 00A0
	zaehler	000E 0BA0
	decfsz 0x20, 1	000F 280E
	goto zaehler	

Wie am Beispiel zu erkennen ist, wäre es unverständlich, direkt im Maschinencode zu programmieren. Daher wird ein Programm genutzt, das den C- oder Assemblercode in die Ma-

schinensprache übersetzt. Dieses Tool wird *Compiler* genannt. Ein *Compiler* arbeitet den Code schrittweise ab und interpretiert die vom Entwickler programmierten Befehle. Dies erfolgt häufig in mehreren Durchgängen. Nach dem Kompilieren müssen noch einzelne Module mit einem weiteren Tool, dem Linker, verbunden werden. Wenn alles richtig funktioniert und kein Fehler gemacht wurde, erscheint nun der Maschinencode, der prinzipiell nur auf dem vorgesehenen Controller lauffähig ist. Während des Übersetzungsvorgangs wird eine Datei mit der Endung *.hex* generiert, die beschreibt, an welcher Stelle im Speicher die einzelnen Befehle stehen sollen. Dieses File kann nun mit einem Programmiergerät in den Mikrocontroller geladen werden. (Hofmann, 2009, S15) [12]

5.3 Die Programmiermöglichkeiten in MPLAB

Bei der Programmierung werden verschiedene Schritte durchlaufen. Um ein neues Programm in den Flash-Speicher des Mikrocontrollers zu schreiben, muss dieser zuvor gelöscht sein. Nach dem Beschreiben soll sichergestellt werden, dass die Daten richtig im Programmspeicher stehen. Daher werden die geschriebenen Daten nochmals ausgelesen und mit denen auf dem PC verglichen.

Selbstverständlich können die Daten durch einzelne Schritte in den Mikrocontroller geschrieben und geprüft werden. In der Regel läuft die Programmierung jedoch automatisiert ab und die nötigen Schritte werden nacheinander ausgeführt. Damit eine Datenübertragung vom PC in den PIC möglich wird, muss zuerst der ICD3 von MPLAB initialisiert und eine Verbindung hergestellt werden. Nach dem Herstellen der Verbindung wird von MPLAB geprüft, ob der richtige Mikrocontroller angeschlossen ist. Wurde bei den Einstellungen ein falscher Typ ausgewählt, z.B. PIC24FJ128GB004 statt PIC24FJ64GA002, wird an dieser Stelle eine Warnung ausgegeben. In den meisten Fällen ist dies kein Störfaktor und der PIC kann trotzdem programmiert werden. In Einzelfällen kann es jedoch zu Problemen bei der Programmausführung kommen. Bei MPLAB werden die nötigen Befehle für das Programmieren im Menü *Programmer* aufgeführt.

Mit den Befehlen können die folgenden Aktionen ausgeführt werden:

Erase Part: Mit diesem Befehl wird der interne Flash-Speicher des Mikrocontrollers gelöscht. Im Speicher erscheinen nach dem Löschvorgang nur Einsen.

Blank Check: Zur Überprüfung der erfolgreichen Löschung wird ein sogenannter *Blank Check* durchgeführt. Dabei wird der Inhalt des Flash-Speichers gelesen und getestet, ob an jeder Speicherstelle eine Eins steht. Ist dies nicht der Fall, befindet sich noch ein altes Pro-

gramm im Chip. Bei einer gestörten Übertragung kann es vorkommen, dass nicht alle Speicherzellen ordnungsgemäß gelöscht werden. Dann sollte die Verbindung überprüft und der Vorgang wiederholt werden.

Program: Nachdem keine Daten mehr im Programmspeicher vorhanden sind, kann das Programm mit dem Befehl *Program* in den Flash-Speicher geladen werden. Das Programm muss hierzu vorher geladen und übersetzt werden. Die Konfigurationsbits werden erst zum Schluss in den PIC programmiert. Danach wird automatisch die Überprüfung des Programmcodes und der Konfigurationsbits gestartet.

Read: Mit dem Befehl *Read* wird der gesamte Speicherinhalt gelesen, sofern der Chip nicht mit dem Konfigurationsbit *Code Protect* gegen ein Auslesen geschützt ist. Sollte der Mikrocontroller unbeschrieben sein, wird aus jeder Programmspeicherzelle der Wert 0x3FFF ausgelesen. Befindet sich ein Programm im Speicher, erscheinen der binär codierte *Ausführungscode* und die zurückgewandelten Assemblerbefehle (Disassembly). Die Kommentare und Variablennamen, die im Quellcode angegeben wurden, werden nicht genannt. Zum Aufrufen des Programmcodes muss eventuell das Fenster über das Menü *View* und den Punkt *Program Memory* sichtbar geschaltet werden.

Verify: Damit nicht jede Zeile einzeln mit dem programmierten Code verglichen werden muss, wird der Befehl *Verify* ausgewählt, welcher MPLAB die Daten automatisch vergleichen lässt.
Nach erfolgreicher Beschriftung des Programmspeichers kann die Betriebsspannung ohne einen Datenverlust entfernt und wieder angelegt werden.
(Hofmann, 2009, S84f.) [12]

5.4 Braucht man eine Header-Datei?

Zu Beginn der Programmierung muss die richtige Header-Datei eingefügt
werden. Bedauerlicherweise ist die Datei gut versteckt. Daher folgt nun ein kurzer Leitfaden zum Einfügen:
Zuerst wird im Hauptfenster von MPLAB die Datei bzw. das Fenster (Diplomarbeit-PatrickNeuhalfen.c.mcw) angezeigt.

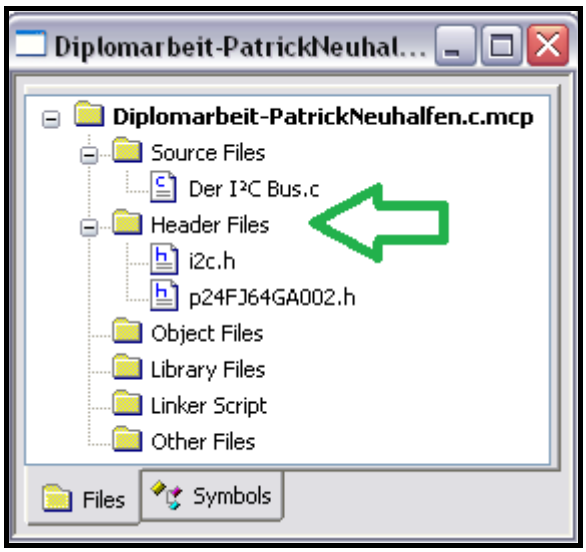


Abbildung 5.4: Der Weg zur Header-Datei (1)

Durch Rechtsklick auf *Header Files* öffnet sich ein kleines Fenster, in dem die Option *Add Files* ausgewählt wird.

Daraufhin erscheint ein Fenster mit dem Ordner MPLAP.

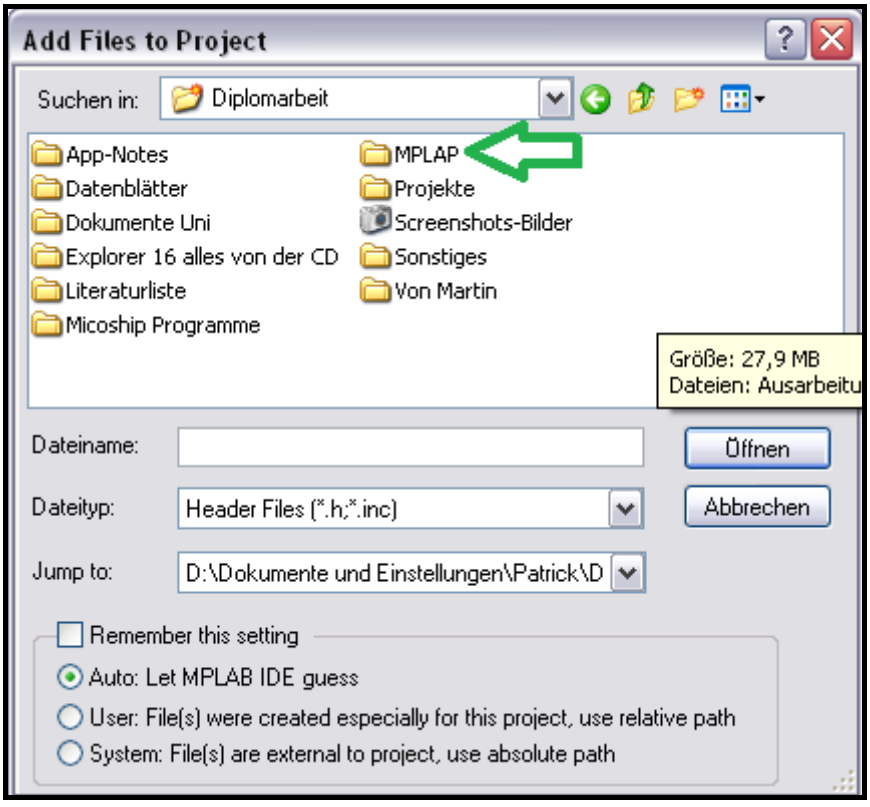


Abbildung 5.5: Der Weg zur Header-Datei (2)

Dann muss dem folgenden Ordner-Pfad gefolgt werden:

MPLAB/ Version x.xx (hier: 8.40)/ C-Compiler/ support/ PIC24F/ h/

Nun kann die richtige Header-Datei ausgewählt und eingefügt werden.

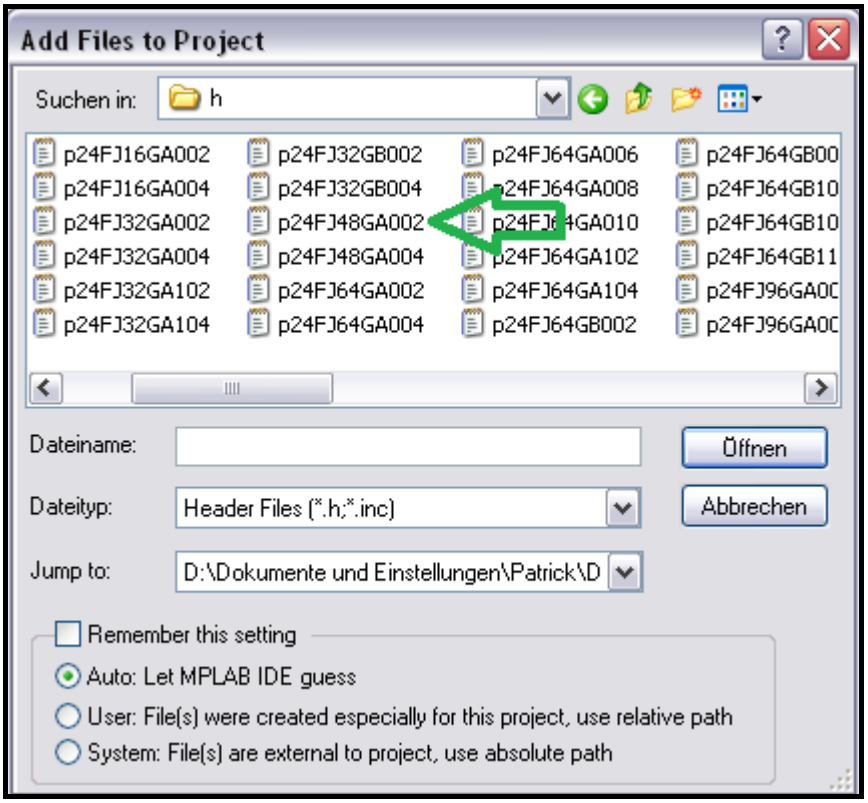


Abbildung 5.6: Der Weg zur Header-Datei (3)

Den Inhalt der Header-Dateien bilden üblicherweise die Vereinbarung von Makros und symbolischen Konstanten, die Beschreibung der Aufrufschnittstelle von externen Funktionen und die Deklaration von nutzbaren Datentypen.

5.5 Was sind die Konfigurationsbits?

Der Zugriff auf das Register, in welchem die Konfigurationsbits stehen, kann nur während des Programmierens erfolgen. Nach Programmierung des Chips können diese Bits nicht mehr verändert werden. Um eine Änderung an diesen Bits vornehmen zu können, muss der gesamte Chip gelöscht und anschließend mit geänderten Einstellungen neu programmiert werden. Dies ist sinnvoll, da sonst nachträglich das Bit für den Ausleseschutz zurückgesetzt werden könnte, um so die Daten aus dem Speicher auslesen zu können.

Die Konfigurationsbits können auf zwei Arten gesetzt werden: im Code selbst oder über das Menü *Configure* im Untermenü *Configurations Bits*. Es empfiehlt sich, die Bits direkt im Code zu setzen, damit die Aktivierung des Ausleseschutzes nicht versäumt wird. Gleichzeitig werden hierdurch die entsprechenden Einstellungen für eine Serienproduktion sichtbar. Das Register, in dem die Konfiguratinsbits gesetzt werden, hat eine Breite von 14 Bit. Um die Konfigurationsbits im Code zu setzten, wird das Schlüsselwort *__CONFIG* benutzt. Dieses Schlüsselwort sorgt für eine Auflistung der Konfigurationsbits durch die Verknüpfung & (innerhalb einer Zeile). (Hofmann, 2009, S85f.) [12]

Benutzte Einstellung im Projekt:

```
_CONFIG1( JTAGEN_ON & GCP_OFF & GWRP_OFF & COE_OFF & FWDTEN_ON
          & ICS_PGx2)

_CONFIG2( FCKSM_CSDCMD & OSCIOFNC_ON & POSCMOD_OFF & FNOSC_PRI )
```

5.6 Weshalb sind die Datenblätter so wichtig?

Ein Kernpunkt der Programmierung von Mikrocontrollern besteht in der Arbeit mit Datenblättern. Jeder Mikrocontroller besitzt ein eigenes Datenblatt oder ein Familien-Datenblatt. In einem Familien-Datenblatt sind alle Controller vom Kern auf gleich und unterscheiden sich lediglich in ihrer Größe (Anzahl der Pins), dem Speicherplatz und den Peripheriekomponenten. Daher sind sie auch in ihrer Handhabung identisch.
Ohne die korrekten Datenblätter ist eine erfolgreiche Programmierung nicht möglich.
In den meisten Fällen besitzt ein Mikrocontroller ein Haupt-Datenblatt, in welchem alle Funktionen recht ausführlich erläutert werden.
Für einen allgemeinen Überblick ist dies ausreichend. Für die nähere Betrachtung eines bestimmten Punktes wird allerdings ein gesondertes, ausführlicheres Datenblatt benötigt.

Der in diesem Projekt verwendete Mikrocontroller (PIC24FJ64GA002/SS) hat ein Familien-Datenblatt (PIC24FJ128GB004 Family-Data-Sheet) von 342 Seiten. Der I²C-Bus nimmt in diesem Datenblatt 8 Seiten ein. Zur erfolgreichen Programmierung des Busses bedarf es allerdings eines weitaus größeren Umfanges. Glücklicherweise existiert ein Unter-Datenblatt mit 56 Seiten zum Thema I²C. Zusätzlich werden Datenblätter der zu steuernden Komponenten benötigt, im Falle des Dual I²2 Digital-Potenziometers umfasst es 28 Seiten.

Hieran wird deutlich, dass für die Arbeit mit Mikrocontrollern ein großer Umfang an Unterlagen anfällt. Alle Datenblätter sind in englischer Sprache verfasst. Sie können als pdf-Datei auf der Homepage des jeweiligen Herstellers heruntergeladen werden.

5.7 Durch Kommentare zum Erfolg

Um erfolgreich programmieren zu können, sollte der eigene Quellcode ausführlich kommentiert werden. Dies steigert die Übersichtlichkeit des Programms und vermeidet Unsicherheiten bezüglich des eigenen Gedankengangs.
Zudem erleichtern Kommentare die Fehlersuche. Besonders für größere Projekte oder bei einem Mitarbeiterwechsel sind sie unabdingbar.
Daher ist es sinnvoll, die einzelnen Code-Zeilen von Beginn an zu kommentieren.
Für jede Sprache ist festgelegt, an welcher Stelle ein Kommentar beginnt und wo er endet. Man unterscheidet im Allgemeinen zwischen Block- und Zeilenkommentaren. Zeilenkommentare enden automatisch am Zeilenende. Blockkommentare hingegen können sich über mehrere Zeilen erstrecken und enden erst nach der Einleitung beim Endzeichen.

Mögliche Kommentare in *C*:

```
Code   /* Ein Kommentar,
          der auch Zeilenumbrüche
          enthalten darf. */ Code

Code   // Ein Kommentar, der bis zum Zeilenende geht
       // Soll er weitergehen, muss er erneut als Kommentar
       // gekennzeichnet werden.
```

Wird ein Quelltext verarbeitet (kompiliert, interpretiert, geparst etc.), werden Kommentare von der verarbeitenden Software ignoriert und haben daher keinen Einfluss auf das Ergebnis.

6. Die Hardware

Eine Hauptaufgabe des Mikrocontrollers liegt bei diesem Projekt in der Steuerung bzw. im Hochrampen von Spannungen. Daher muss eine Lösung für die Spannungserhaltung nach Abschalten des Mikrocontrollers gefunden werden. Hierbei stellen sich folgende Fragen:

1. Wie kann der Mikrocontroller Spannungen langsam an einen bestimmten Sollwert heranfahren?
2. Wie können diese Sollwerte nach Abschaltung des Controllers erhalten werden?

Da das fertige RF-Modul möglichst klein gehalten werden soll, muss die angestrebte Lösung platzsparend und effizient sein.
Nach langen Überlegungen entstand die Grundidee, digitale Potenziometer über einen I²C-Bus anzusteuern. Es existieren digitale Potenziometer, die ein integriertes EEMEM-Register besitzen, wodurch die Sollwerte abgespeichert werden können.
Die nächste Herausforderung bestand in der praktischen Umsetzung.
Dieses Kapitel befasst sich mit dem Bau des Test-Moduls und erläutert die wichtigsten Komponenten. Außerdem wird die Auswahl des Bus-Systems und dessen Funktion ausführlich dargelegt.

6.1 Der Mikrocontroller im Detail

Die wichtigste Komponente ist selbstverständlich der ausgewählte Mikrocontroller. Der *PIC24FJ64GA002-I/SS* gehört zur Controller-Familie *PIC24FJ64GB004 FAMILY* der Firma Microchip.
Die beiden folgenden Abbildungen zeigen den inneren und äußeren Aufbau des Mikrocontrollers.

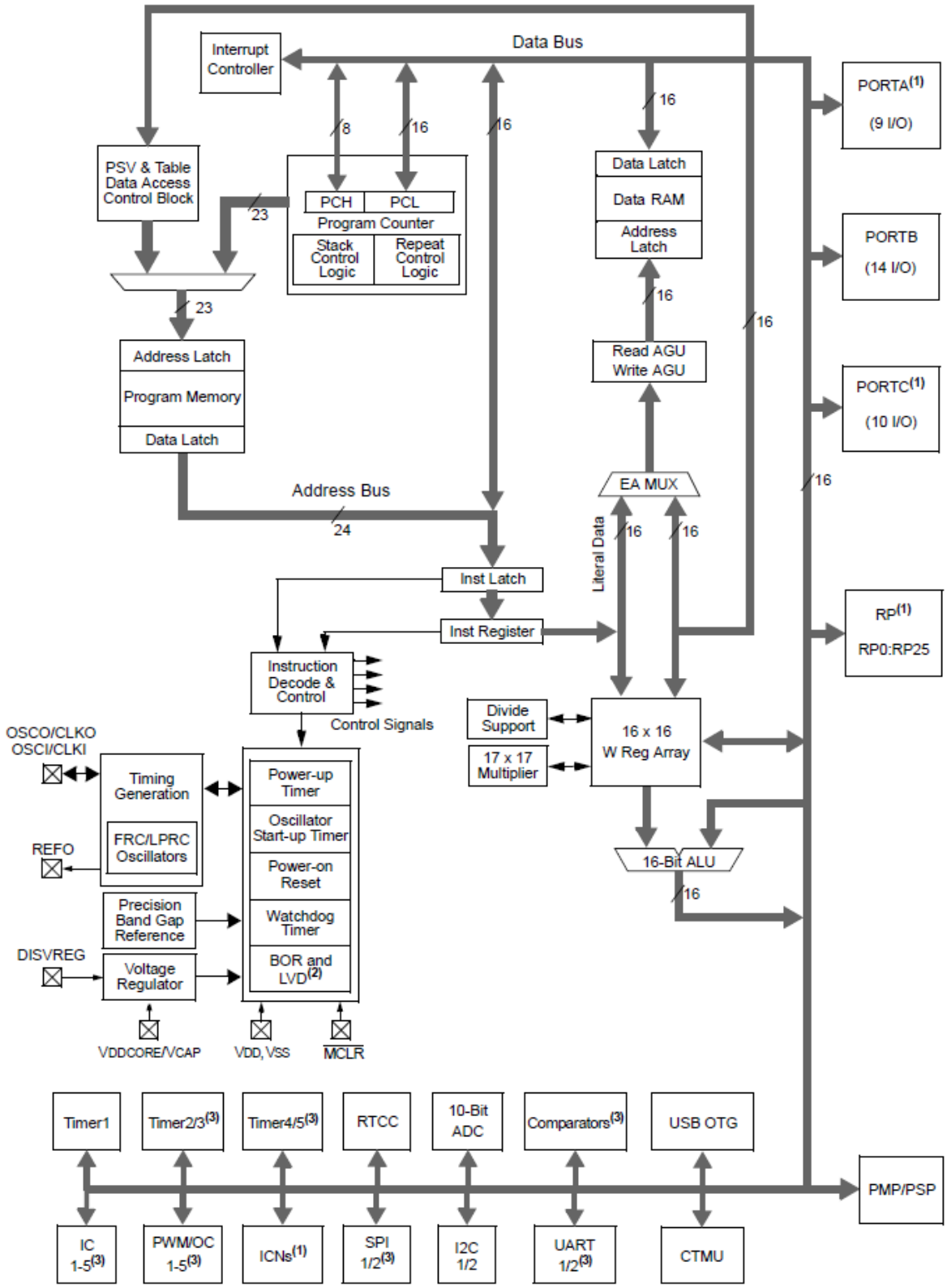


Abbildung 6.1: Blockschaltbild der Controller-Familie PIC24FJ64GB004 [1]

Info: 28-Pin SPDIP/SOIC, die grau hinterlegten Pin sind 5,5 V tolerant.

Auf *Abb. 6.2* ist die Kompaktheit des Mikrocontrollers gut zu erkennen. Ein Pin kann bis zu elf verschiedene Funktionen erfüllen.

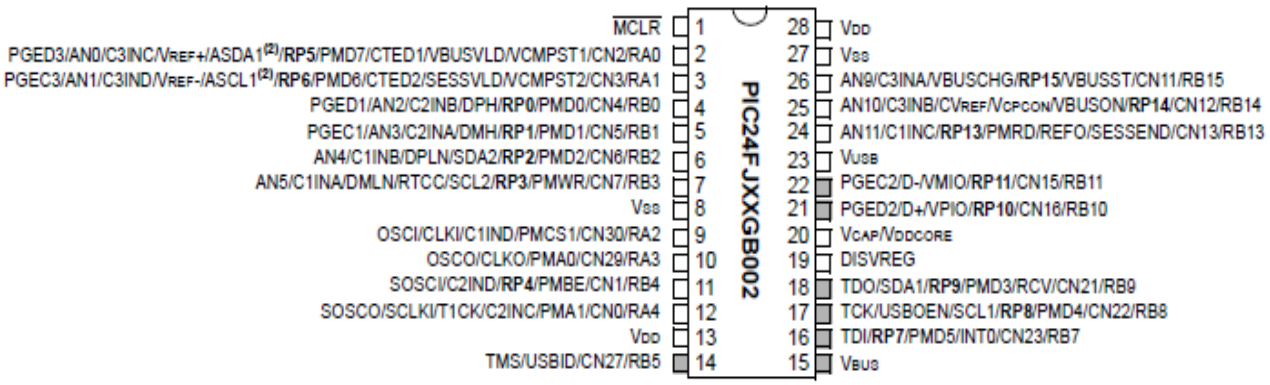


Abbildung 6.2: Pin-Diagramm des Mikrocontrollers [1]

Die wichtigsten Spezifikationen:

Pins: 28

MIPS: bis zu 16 Operationen a 32 MHz.

Betriebsspannung: von 2,0 V bis 3,6 V

Flash-Programmspeicher: 64 k bis zu 10.000 Mal beschreibbar/lesbar

(Datenerhaltung mindestens 20 Jahre)

SRam: 8k

Power-Management-Modes: Sleep, Deep-Sleep, Idle, Doze und Alternate Clock

Power-On-Reset (POR)

Power-Up-Timer (PWRT)

Oscillator Start-Up-Timer (OST)

Watchdog Timer (WDT)

In-Circuit-Serial-Programming (ICSP)

In-Circuit-Debug (ICD)

JTAG (Boundary Scan and Programming Support)

10-Bit-Analog-Digital-Wandler (10 Kanäle)

Hardware Real-Time Clock-Calendar (RTCC)

Fünf 16-Bit-Timer

Zwei UART-Module mit Unterstützung für RS-485, RS-232 und LIN 1.2

Zwei SPI-Module

Zwei I2C-Module

6.2 Der Tiefschlafmodus

Der *PIC24FJ64GA002/SS* unterstützt verschiedene *Power-Saving-Features*.

Viele moderne Controller haben bereits derartige Modi, beispielsweise *Sleep-Mode*, *Idle-Mode* oder *Doze-Mode*. Den *Deep-Sleep-Mode* unterstützen jedoch die Wenigsten.

Deshalb stellte dieser Modus ein entscheidendes Kriterium bei der Auswahl des Mikrocontrollers für dieses Projekt dar.

Im sogenannten *Deep-Sleep-Mode* (Tiefschlafmodus) wird der Mikrocontroller in einen Zustand versetzt, in welchem der Stromverbrauch nahe Null liegt. Somit ist eine Trennung des Controllers von der Spannungsversorgung (VDD) durch externe Schalter nicht notwendig.

Um Leckströme zu vermeiden, wird der Kern des Mikrocontrollers im Tiefschlafmodus von der Spannungsversorgung getrennt. Daher sind die meisten Peripheriegeräte und Funktionen während des Tiefschlafmodus nicht verfügbar. Einige wenige jedoch hängen direkt an der VDD-Versorgungsschiene des Mikrocontrollers und sind daher auch während des *Deep-Sleep-Mode* aktiv. Der Mikrocontroller kann in den Tiefschlafmodus versetzt werden, indem man das Bit DSEN(<15>) des DSCON-Registers setzt. Durch diesen Modus wird das Register DSWAKE gelöscht. RTC (*Real-Time-Clock*) und RTCC (*Real-Time-Clock-Calender*) können, je nach Einstellung, ohne Unterbrechung weiterlaufen.

Um die Gefahr eines versehentlich ausgelösten Tiefschlafmodus zu verringern, muss die Ausführung (Command-Befehl) des *Deep-Sleep-Mode* in den folgenden drei Befehlszyklen geschehen. Nach dem fünften Befehlszyklus wird das Bit DSEN (<15>) wieder zurückgesetzt. Erfolgt eine Ausführung des Modus, während das Bit DSEN nicht gesetzt ist, fällt der Controller nur in den normalen Schlafmodus (*Sleep-Mode*).

Der Mikrocontroller kann durch ein MCLR (*Master-Clear-Event*), POR (*Power-On-Reset*), RTCC (*Real-Time-Clock-Calender*), INT0 (*I/O-Interrupt-On-Change*) oder DSWDT (*Deep-Sleep-Watch-Dog-Timer*) aus dem Tiefschlafmodus geweckt werden. Nach dem Erwachen löst der Controller automatisch einen POR (*Power-On-Reset*) aus. Durch die Abfrage des Bit DPSLP(<10>) des Registers RCON kann dies festgestellt werden. Wenn ein POR durch Erwachen des Controllers ausgelöst wurde, wird das Bit gesetzt. Es werden alle Register, das RCON, DSCON und DSGPRx ausgenommen, in den Ausgangszustand zurückgesetzt. Durch den Reset werden die Daten der meisten SFR (*Spezial-Function-Register*), die sich in RAM befinden, gelöscht. Für diesen Fall existieren zwei separate Datenspeicher-Register (DSGPR0 und DSGPR1). Durch das Auslesen dieser Register und die Rücksetzung des Bits RELEASE(<0>) des Registers DSCON, können alle Informationen, die zur Verwaltung des Prozessors nötig sind, wiederhergestellt werden.

An dieser Stelle ist es wichtig, dass das Bit DPSLP(<10>) des Registers RCON manuell per Software zurückgesetzt werden muss. Die Quelle des *Wake-Up-Events* kann durch Auslesen des Registers DSWAKE ermittelt werden. Dieses Register wird beim Start des *Deep-Sleep-Mode* automatisch zurückgesetzt. Das Register sollte daher möglichst nach dem Erwachen aus dem Tiefschlafmodus ausgelesen werden.
Eine genaue Übersicht der Register ist in einem spezifischen Datenblattteil des Mikrocontrollers aufgelistet.

6.3 Die Spannungsversorgung

Für die Gewährleistung einer stabilen Spannungsversorgung des Testboards, werden der Festspannungsregler und zwei Kondensatoren eingesetzt.

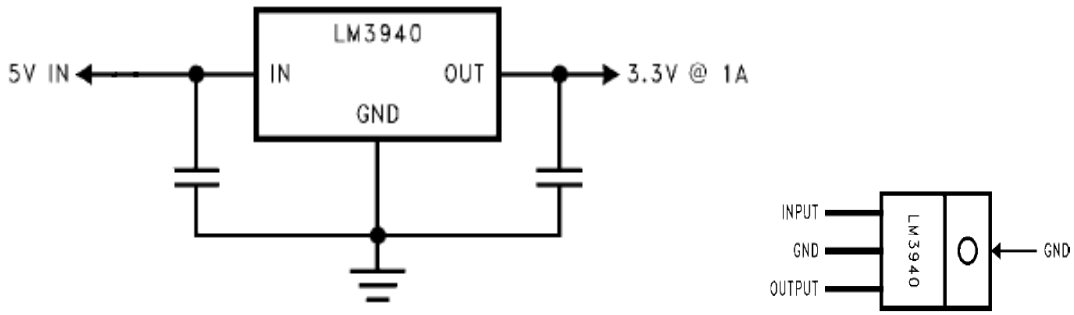


Abbildung 6.3: Festspannungsregelung des Testboards [1]

C1 (eingangsseitig) = 100 nF
C2 (ausgangsseitig) = 10 nF

Zur zusätzlichen Stabilität werden zwei Elektrolytkondensatoren parallel geschaltet.

C3 (eingangsseitig) = 100 µF
C4 (ausgangsseitig) = 10 µF

Dieser Aufbau liefert bei ca. 5 V eine exakte, stabile Spannung von 3,3 V.

6.4 Master-Clear-Anschluss

Dieses Anschlussverfahren stammt aus dem Datenblatt des Controllers.

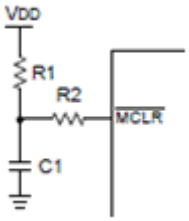


Abbildung 6.4: Anschluss des Master-Clear-Pins [1]

mit:

C1 = 1 µF, 20 V Keramik

R1 = 10 kOhm

R2 = 100-470 Ohm

Bei Schwierigkeiten bei der Programmierung des Controllers, kann C1 per *Jumper* für die Dauer der Programmierung isoliert werden.

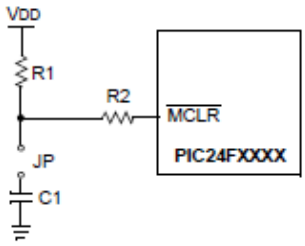


Abbildung 6.5: Anschluss des Master-Clear-Pins (2) [1]

Durch die Möglichkeit, einen Jumper direkt auf Masse zu legen, könnte ein manueller Reset ausgeführt werden.

6.5 Das I²C-Bus-System

Es existieren auch andere Bus-Systeme (z.B. den SPI-Bus), die eine solche Aufgabe ebenfalls realisieren könnten. Da der I²C-Bus kein 4-Draht-Bus ist, sondern nur 2 Leitungen benötigt, wurde er für dieses Projekt ausgewählt.

I²C steht für *Inter-Integrated Circuit.*, in älteren Büchern ist häufig die Bezeichnung TWI, *Two-Wire-Interface (Zweidrahtverbindung)* zu finden. Das bedeutet, dass der Bus zur Kommunikation zwischen verschiedenen Bauteilen *(ICs)* verwendet wird. Diese erfolgt über nur

2 Leitungen, die Taktleitung *(SCL)* und die Datenleitung *(SDA)*. Diese Tatsache war ein entscheidendes Kriterium bei der Auswahl des Bus-Systems. Der SPI-Bus, der die Steuerung des Potenziometers ebenfalls hätte übernehmen können, benötigt mindestens 4 Leitungen. Die Schnittstelle wurde von Philips Semiconductors (heute *NXP Semiconductors*) entwickelt. Das Bussystem ist mittlerweile weit verbreitet und wird von zahlreichen ICs verschiedener Hersteller unterstützt. Der genutzte Mikrocontroller verfügt ebenfalls über eine solche Schnittstelle. Die Daten werden als 8-Bit-Werte mit einer Geschwindigkeit von bis zu 3,4 Mbit übertragen. Die Standardgeschwindigkeit liegt bei 100 kbit, was für das Steuerregister der vorhandenen Bauteile völlig ausreichend ist. Für dieses Projekt ist die Zuverlässigkeit des Bussystems wesentlich wichtiger als die Übertragungsgeschwindigkeit. Die Möglichkeiten der I²C-Schnittstelle sind dennoch vielseitig und umfangreich. (Hofmann, 2009, S189)

Funktionsweise der I²C-Schnittstelle

Jedes IC, das an den I²C-Bus angeschlossen wird, hat eine eindeutige Adresse *(Device Address)*. In der Regel ist der Mikrocontroller der Bus-Master, der die Kommunikation regelt. Der Master spricht die angeschlossenen ICs *(Slave=Sklaven)* über diese Adresse an. Nach der Startsequenz überträgt der Master die *Slave-Adresse* des anzusprechenden Geräts und teilt mit, ob Daten gesendet oder aus dem internen Speicher gelesen werden sollen. Dies erfolgt über das letzte Bit der Adresse. Sollen Daten vom *Slave* gelesen werden, wird das letzte Bit auf *High-Pegel* gesetzt. Sollen Daten an den *Slave* übertragen werden, wird das Bit auf *Low-Pegel* gezogen. Erkennt ein *Slave* anhand der Adresse, dass er angesprochen wurde, bestätigt er den korrekten Empfang, indem er das nächste Bit (*Acknowledge*) auf Null setzt. Danach wird in den meisten Fällen die interne Adresse des anzusprechenden Registers im IC übertragen. Im Anschluss daran werden die Daten, die in das Register geschrieben oder aus dem Register ausgelesen werden sollen, übertragen. Es besteht außerdem die Möglichkeit, mehrere Master an einen Bus zu hängen, was für dieses Projekt jedoch nicht notwendig ist. Damit die Leitungen im Leerlaufmodus einen *High-Pegel* aufweisen, müssen zwei Pull-Up-Widerstände angehangen werden. Werden mehrere ICs angeschlossen oder sind die Busleitungen länger, steigt die Kapazität und die Widerstände müssen kleiner sein, damit die Kapazitäten ausreichend schnell umgeladen werden können. Dieser Effekt ist bei höheren Datenübertragungsraten kritischer als bei Standardgeschwindigkeit. Für die Standardgeschwindigkeit von 100 kBit kann der Widerstand zwischen 3 und 10 kOhm liegen. Für höhere Übertragungsraten sollte ein Widerstand zwischen 1 und 5 kOhm gewählt werden. Da zum

Ende des Projekts mehrere Teilnehmer an den Bus gehangen werden, wurde ein typischer Wert von Rp = 4,7 kOhm gewählt.

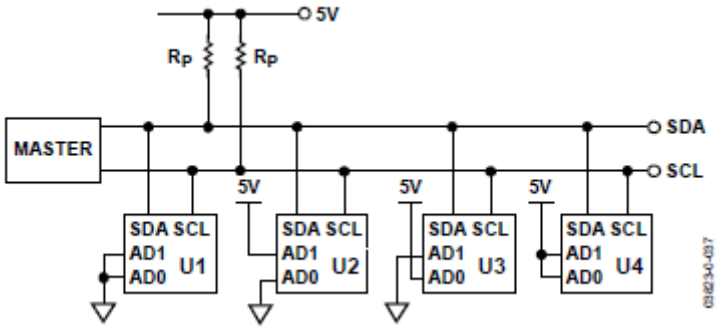


Abbildung 6.6: Anschluss der *Slaves* an den Bus; Pull-Up-Widerstände [1]

Damit die angeschlossenen ICs erkennen, wann die Übertragung beginnt oder endet, gibt es spezielle Start- und Stoppsequenzen. Zum Starten einer Übertragung werden zuerst die Datenleitung *(SDA)* und dann die Taktleitung *(SCL)* auf *Low-Pegel* gezogen. Nach der Startsequenz gibt der Master den Takt der Übertragung vor. Zum Beenden einer Übertragung wird die Taktleitung *(SCL)* auf *High-Pegel* gezogen. Anschließend wird die Datenleitung *(SDA)* von *Low-Pegel* auf *High-Pegel* nachgeschaltet. Danach ist der Bus wieder hochohmig und kann von anderen Busmastern wiederverwendet werden. Um eine sichere Datenübertragung gewährleisten zu können, ist ein Datenwechsel nur dann erlaubt, wenn die Taktleitung *(SCL)* auf *Low-Pegel* liegt. Die Daten müssen vor der steigenden und nach der fallenden Taktflanke stabil anliegen. (Hofmann, 2009, S189fc)

Das Bus-Protokoll

Der Datentransfer kann nur gestartet werden, wenn der Bus nicht beschäftigt ist.

Bei der Datenübertragung darf der Pegel der Leitung nicht geändert werden, während sich die Taktleitung auf *High-Pegel* befindet, da Änderungen sonst als *Start-* bzw. *Stoppbedingungen* interpretiert werden.

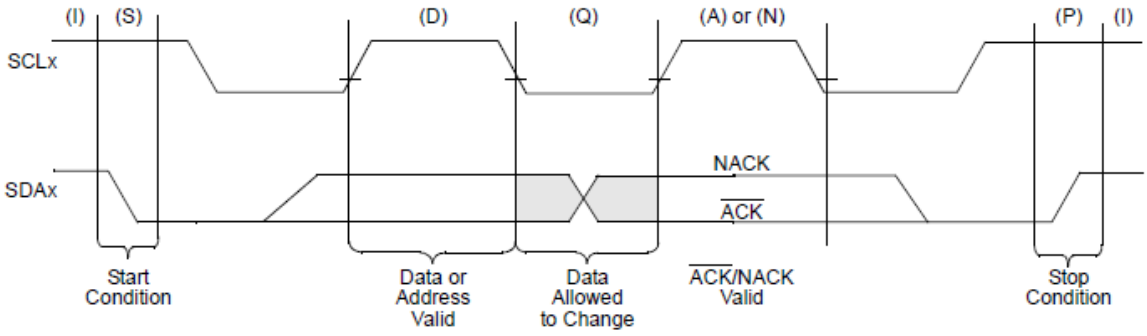


Abbildung 6.7: I2C Bus Protokoll [1]

Legende:

(I): Leerlaufmodus: Die Takt- *(SCL)* und Datenleitungen *(SDA)* befinden sich auf *High*, dies ist vor der Start- und nach der Stoppsequenz der Fall.

(S): Startsequenz: Während sich der Bus im Leerlaufmodus befindet, wird Datenleitung *(SDA)* von *High* auf *Low* gezogen, die Taktleitung *(SCL)* liegt dabei auf *High*.

Alle Datenübertragungen beginnen mit der Startsequenz.

(P): Stoppsequenz: Die Datenleitung *(SDA)* wird von *Low* zurück auf *High* gezogen, dabei muss die Taktleitung *(SCL)* ebenfalls auf *High* liegen.

(D): Das Datenmaterial: Nach der Startsequenz kann mit der Datenübertragung begonnen werden. Jedes Mal, wenn die Taktleitung *(SCL)* auf *High* liegt, wird ein Bit übertragen.

(Q): Zustandswechselbereich: Jedes Mal, wenn sich die Taktleitung *(SCL)* auf *Low* befindet, kann die Datenleitung *(SDA)* ihren Zustand ändern.

(A)(N): Bestätigungsbits: Jede Datenbyte-Übertragung muss vom *Slave (Acknowledge)* oder vom Master *(Not-Acknowledge)* anerkannt werden. Der *Slave* zieht also die Datenleitung *(SDA)* auf *Low* bzw. der Master auf *High*.

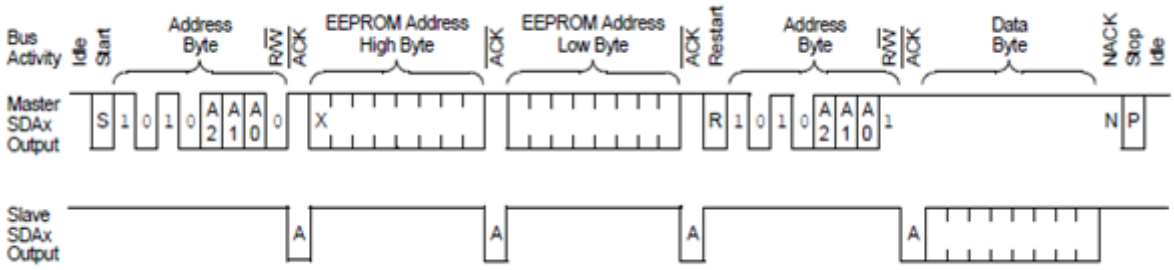


Abbildung 6.8: I²C-Übertragungsprotokoll für die Ansteuerung eines EEProm [1]

Jede Übertragung beginnt mit der Start- und endet mit der Stoppsequenz. Das erste gesendete Byte muss immer die *Slave-Adresse* (7 Bit) plus ein Lese-*(1)*Schreib-Bit *(0)*sein. Es muss sichergestellt werden, dass beim ersten Byte immer der Schreibbefehl *(0)* steht, da der Master dem *Slave* zuerst mitteilen muss, was gelesen werden soll. Der Slave bestätigt mit einem Bit *(Acknowledge)* jedes ankommende Byte.

(R) Wiederholter Start: Um die Busrichtung umzukehren ohne die Übertragung zu beenden, muss ein wiederholter Start ausgeführt werden. Dieses Byte enthält dieselbe *Slave-Adresse.* Nun wird am Ende allerdings der Lesebefehl *(1)* gesetzt. Nun kann der *Slave* die Daten übertragen während der Master weiterhin den Takt generiert und dabei die Datenleitung *(SDA)* auf *High* belässt. Nach jedem Byte bestätigt der Master den Empfang mit dem Bestätigungsbit *(Not-Acknowledge)*, er setzt die Datenleitung *(SDA)* also auf *High*. Danach folgt die Stoppsequenz. Der Master stoppt die Übertragung und der Bus springt wieder in den Leerlaufmodus.

6.6 Das Digital-Potenziometer

Digitale bzw. elektronische Potenziometer bestehen aus $n - 1$ hintereinander geschalteten einzelnen Widerständen (z.B. 100) sowie aus n Feldeffekttransistoren bestehenden elektronischen Schaltern. Diese Anordnung ist mit einer digitalen Steuerschaltung zu einem integrierten Schaltkreis zusammengefasst. Solch digitale Potenziometer werden sowohl als Trimmpotenziometer (sie behalten ihren eingestellten Wert lebenslang) oder zur Einstellung über Taster, Inkrementalgeber oder Mikrocontroller verwendet. Sie haben dementsprechend einen flüchtigen und/oder einen nichtflüchtigen Speicher für die *Schleiferstellung*.

Für dieses Projekt fiel die Entscheidung auf das *Dual 256-Position I²C Nonvoltile Memory Digital Potenziometer AD5252.* Dieses Potenziometer ist I²C-ready und besitzt eine 8-Bit-Auflösung, d.h. 2^8 entsprechen 256 Positionen. In dieser Arbeit wird lediglich das Grund-

prinzip des Potenziometers erläutert. Weitere Spezifikationen und Merkmale können auf der beigefügten DVD im Anhang eingesehen werden

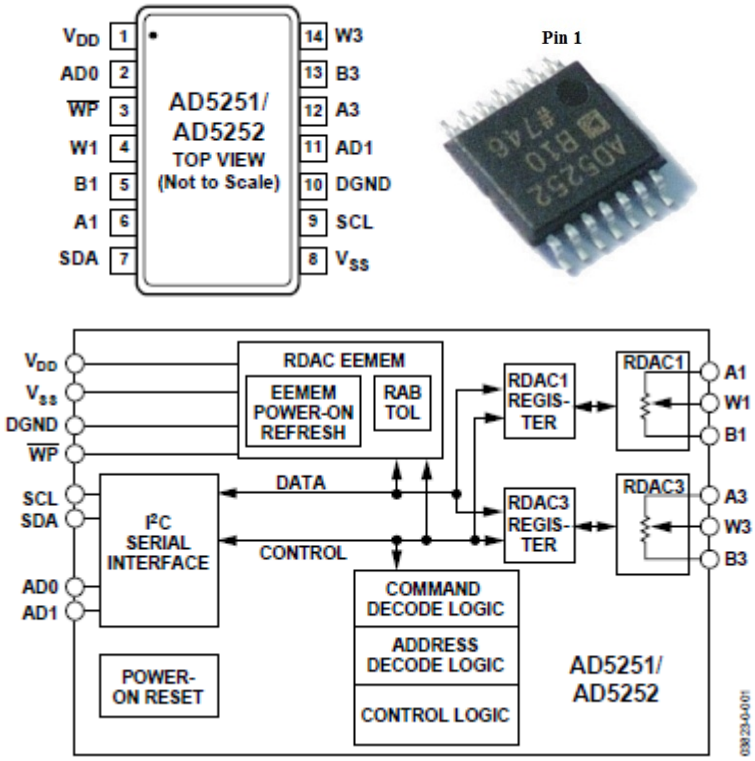


Abbildung 6.9: Blockschaltbild des AD5252, Pin-Konfiguration [1]

Pin **1**: Positive Spannungsversorgung

Pin **2**: Eines der zwei möglichen Adresspins zur Ansteuerung der *Slaves*

Pin **3**: Schreibschutz, Aktiv *Low*

Pin **4**: Wiper-Terminal von RDAC1

Pin **5**: Terminal B von RDAC1

Pin **6**: Terminal A von RDAC1

Pin **7**: Serielle Dateneingabe bzw. -ausgabe

Pin **8**: Negative Spannungsversorgung

Pin **9**: Serielle Taktleitung

Pin **10**: Digitale Masse

Pin **11**: Eines der zwei möglichen Adresspins zur Ansteuerung der *Slaves*

Pin **12**: Terminal A von RDAC3

Pin **13**: Terminal B von RDAC3

Pin **14**: Wiper-Terminal von RDAC3

In der folgenden Abbildung ist eine Programmiersequenz zum Schreiben von Datenbytes ins RDAC oder EEMEM zu sehen.

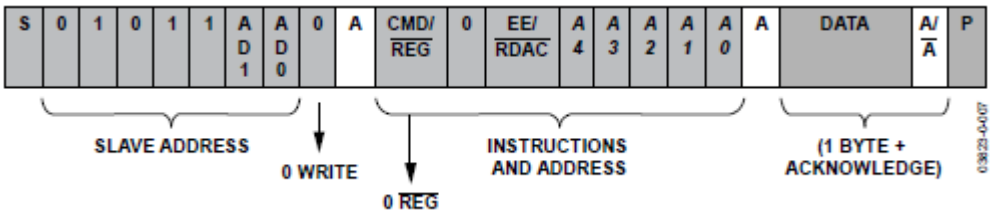


Abbildung 6.10: Protokoll eines Programmiervorgangs [1]

Die *Slave-Adresse* ist bis auf die drei letzten Bits festgelegt. AD1 und AD0 können zur Veränderung der Adresse seitens der Hardware auf *High* oder *Low* gelegt werden. Dies ermöglicht den Anschluss mehrerer *Slaves* an den Bus. Das letzte Bit muss beim Schreibvorgang *0* lauten. Zum Auslesen des Potenziometers muss es auf *1* gesetzt werden.
Das CMD- bzw. REG-Bit erlaubt bei einer logischen *1* die Aktivierung des Bit-Befehls und bei einer logischen *0* den Registerzugriff.
Das EE- bzw. RDAC-Bit erlaubt bei einer logischen *1* das Schreiben in das EEMEM- Register und bei einer logischen *0* das Schreiben in das RDAC- Register.
Die Bits A0 bis A4 sind reserviert, ebenso *00000* und *00010*. Die Kombination *00001* steuert das RDAC1, die Kombination *00011* steuert das RDAC3. Alle übrigen Kombinationen dienen dem Speichern und Auslesen des EEMEM.

6.7 Der Bau des Testboards

Das Testboard wurde auf eine Lochraster-Europakarte (100x160 mm) aufgebaut.

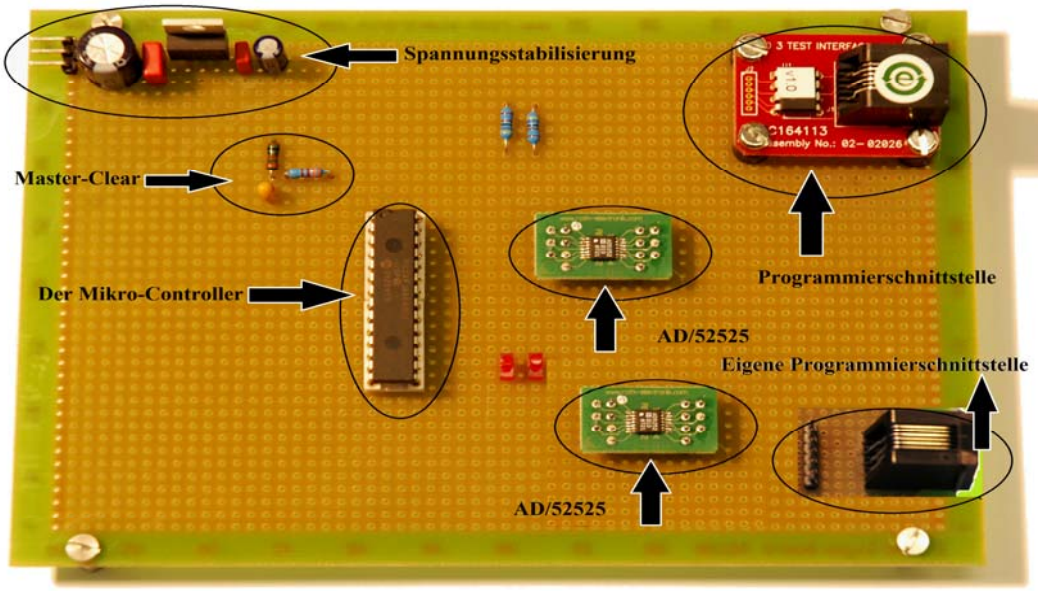


Abbildung 6.11: Das fertige Testboard

Oben links befindet sich die Spannungsversorgung. Der Anschluss sowie das Kabel wurden mit drei Stiften realisiert, die äußeren Masse und in der Mitte 5 V. Das Kabel kann somit nicht in falscher Richtung angeschlossen werden.

Bei den meisten Europakarten sind die beiden ersten Lochreihen rechts und links miteinander verbunden. Um einen Kurzschluss zu vermeiden, sollte die Stiftleiste zur Spannungsversorgung an der zweiten Lochreihe angesetzt werden. Daneben ist die stabile Spannungsversorgung mit dem LM3940 zu sehen. Die Master-Clear-Beschaltung wurde analog *Abb. 6.1* aufgebaut. Oben rechts wurde ein kleines Testinterface für den Debugger angebracht.

Darunter ist die Schnittstelle sichtbar, über welche sich der Mikrocontroller programmieren und debuggen lässt.

Der Controller (PIC24FJ64GA002/SS) und die beiden Dual-Potentiometer (AD/5252) wurden zu Testzwecken gesockelt, um bei einem technischen Defekt einen Austausch zu erleichtern. Die blauen Widerstände mittig des Bildes stellen die beiden Pull-Up-Widerstände für den I²C-Bus dar. Um die Unterlage beim Löten zu schützen, wurden an allen vier Ecken Schraubbeine angebracht.

Die Verdrahtung von Lochrasterplatinen wird nach einiger Zeit recht unübersichtlich. Um im späteren Projektverlauf eine Übersichtlichkeit zu gewährleisten, wurden unterschiedliche Farben verwendet.

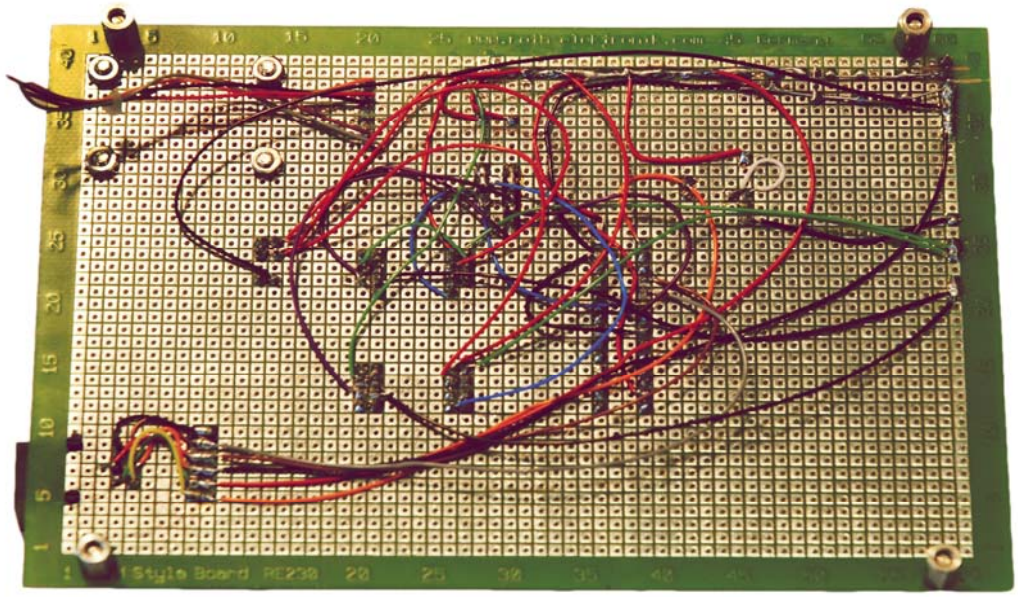


Abbildung 6.12: Die Verdrahtung des Testboards

Legende:

Rot	-	VDD
Schwarz	-	VSS
Grau	-	Programmierpin (PGC)
Braun	-	Programmierpin (PGD)
Blau	-	I²C-Bus (SDA)
Lila	-	I²C-Bus (SCL)
Grün	-	Adressleitungen (*Slaves*)
Orange	-	Master-Clear-Beschaltung
Weiß	-	Brücke

7. Die Software

Dieses Kapitel erklärt die Software, die zur Kommunikation über das I²C-Bussystem dient. Das folgende Programm soll dem Mikrocontroller die Möglichkeit geben, die digitalen Potenziometer so zu steuern, dass ein langsames Anfahren der Sollwerte gewährleistet wird. Nach Abschaltung der Versorgungsspannung soll die Programmierung des EEMEM-Registers die Stellung der Schleifer erhalten. Im Anschluss wird der Mikrocontroller in den *Sleep-Mode* versetzt.

7.1 Die Register

Register dienen der Steuerung des Controllers. Ein moderner Mikrocontroller besitzt Hunderte davon. Die Register des PIC24FJ64GA002/SS haben eine Breite von 16 Bit. Jedes Bit hat dabei eine individuelle Funktion. Im Allgemeinen gibt es vier Kategorien. In den Konfigurationsregistern können diverse Einstellungen vorgenommen werden. Kontrollregister hingegen steuern die Prozesse gewünschter Funktionen. Statusregister geben Auskunft über aktuelle Ereignisse. Andere Register, wie z.B. Übertragungs- oder Pufferregister, werden zum Zwischenspeichern von Datenbytes verwendet.
Im nächsten Abschnitt erfolgt eine Auflistung der wichtigsten Register für das Testprogramm:

Interrupt Register (IFS3): Enthält das Interrupt-Flag-Bit für ein Masterevent des I-2C-Moduls (2).

Control Register (I2CxCON): Dieses Register erlaubt die Kontrolle über die I²C-Modul-Operationen.

Status Register (I2CxSTAT): Dieses Register beinhaltet die Anzeige der Status-Flaggen bzw. den Zustand des Moduls während der Operationen.

Address Mask Register (I2CxMSK): Dieses Register bestimmt, welche Bit-Positionen, die *Mehrere-Adress-Unterstützung* erlauben, im (IC2xADD) ignoriert werden können.

Receive Buffer Register (I2CxRCV): Das Pufferregister, von welchem die Daten-Bytes gelesen werden können. Dieses Register ist ausschließlich zum Lesen geeignet.

Transmit Register (IC2xTRN): Das Übertragungsregister, in welches die Bytes während der Übertragungs-Operation geschrieben werden. Dieses Register ist zum Lesen und Schreiben geeignet.

Address Register (I2CxADD): Dieses Register enthält die *Slave-Adresse*.

Baud Rate Generator Reload Register (I2CxBRG): In diesem Register kann die Übertragungsgeschwindigkeit eingestellt werden.

Control Register (DSCON): Ermöglicht die Kontrolle über den Tiefschlafmodus.

Status Register (DSWAKE): Informationsregister des *Deep-Sleep-Mode*.

Datenspeicher-Register (DSGPR0 und DSGPR1): Notfallregister für SFR, POR nach DS.

Eine ausführliche Registerbeschreibung ist im Anhang beigefügt.

7.2 Der Quellcode

```
/ ********************************************************************
*        Beschreibung:          PIC FJ64 GA002 /SS   - Programmcode          *
**********************************************************************
*        Dateiname:             Der I²C Bus.h                                *
*        Datum:                 17.02.2010                                   *
*        Version:               v. 1.4 BETA                                  *
*        Autor:                 Patrick Neuhalfen                            *
*        e-mail:                p.neuhalfen(at)mpifr-bonn.mpg.de             *
**********************************************************************
*        Bemerkungen:           Tests: I²C-Bussystem und  Rampen der Spannungen  *
**********************************************************************/

#include "p24fj64GA002.h"
// Hier wird die Header-Datei eingebunden. Die Registernamen, welche im
// späteren Verlauf genutzt werden, werden definiert.

#include "i2c.h"
```

```
// Bekanntmachung der I²C-Register

//=============================================================
//        Einstellungen der Konfigurationsbits
//=============================================================

_CONFIG1( JTAGEN_ON & GCP_OFF & GWRP_OFF & COE_OFF & FWDTEN_ON
          & ICS_PGx2)

_CONFIG2( FCKSM_CSDCMD & OSCIOFNC_ON & POSCMOD_OFF & FNOSC_PRI )

//=============================================================
//        Deklaration der globalen Variablen
//=============================================================

int count  =  0;                  // Globale Variable
int flagISR  =  0;                // Globale Variable


int test  =  0;
// Diese Variable dient der Auslesung des Puffer-Empfangsregisters. Wenn auf das
// Pufferregister zugegriffen wird, wird dieses ausgelesen und zurückgesetzt. Dadurch wird
// verhindert, dass ein Puffer-Überlauf auftritt und sich das Programm aufhängt.

int quelle  =  0;
// Dient dem Auslesen des Wake-Up-Events nach dem DS.

long a  =  0;
// 8-Byte-Zwischenvariable für die Zeitfunktion

long timer  =  0x00003FFF;
// 8-Byte-Variable für die Zeit, 0x0000FFFF = = 65535 Befehle, bei einem Prozessortakt von
// ca. 100 kHz entspricht dies etwa einer halbe Sekunde pro Erhöhung

int sollwert  =  0x55;
// Eingabe des Sollwertes in Hex (00-FF)     z.B. 0x99 = = 6 kOhm
```

```
int istwert;
// Variable für den Ist-Wert

void hochrampen (void);
//Funktion zum Anfahren der Spannungen

void deepsleep (void);
// Funktion des Deep-Sleep-Mode

void zeit (void);
// Zeitfunktion zur Überbrückung der Zeit; für eine genauere Zeit sollte ein Timer verwendet
// werden

// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
//          Hauptprogramm
// = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

int main (void)
{
// Hier beginnt das eigentliche Programm. Jedes C-Programm startet mit den Anweisungen
// in der Funktion main.

// PORTKONFIGURATION für einen Pegelwechsel am digitalen Ausgang RB3
// (für Testzwecke sehr gut zu benutzen), kann durch die Kommentarfunktion deaktiviert
// werden.

        TRISB  =  0x0000;
// Setze den Port B auf Ausgang

// Durch dem Befehl registerXbits.bitnameX können die einzelnen Bits der Register
// angesprochen werden.

        PORTBbits.RB3  =  1;
// Setze PIN RB3 auf 1
```

```
        AD1PCFG = 0x9FFF;
// Dieser Befehl deaktiviert alle analogen Eingänge. Der Befehl 0x0018 würde nur die
// für den I²C-Bus benötigten analogen Ausgänge (AN4 & AN5) ausschalten.

        COCON = 0x0000;
// Comperator Control Register deaktivieren (default)

// REGISTERKONFIGURATION für die Interrupts

        quelle = DSWAKE;
// Auslesen der Quelle für den POR

        IPC12bits.MI2C2P = 7;
// Hier wird die Interrupt-Priorität des Masters vom I2C2 Modul festgelegt [0-7].
// höchste Priorität wurde gewählt.

        IFS3bits.MI2C2IF = 0;
// Hier muss das Interrupt-Status-Flag-Bit zurückgesetzt werden.

        IEC3bits.MI2C2IE = 1;
// Hier wird der Interrupt des Masters (I²C-Modul) aktiviert.

// REGISTERKONFIGURATION für die I²C-Schnittstelle

        I2C2CONbits.A10M = 0;
// Register für 7-Bit-Adresse
        I2C2CONbits.I2CEN = 1;
// I²C-Module werden aktiviert.

        I2C2ADD = 0x002C;
// Salve-Adresse AD/5252 [1]
// Hex 2C, Binär 0101100 (AD1 & AD2 auf Masse)
// Slave-Adresse AD/5252 [2]
// Hex 2E,  Binär 0101110 (AD1 auf VDD & AD2 auf Masse)
```

```
        I2C2BRG = 0x009D;
// Taktgeschwindigkeit wird auf 100 kHz eingestellt.

        Nop ( );
// No-Operation (ein Befehlszyklus nichts machen)
// Wichtig beim Debugger-Überlauf (gute Stelle für Breakpoint)
        Nop ( );
        Nop ( );
        Nop ( );

        I2C2CONbits.SEN = 1;
// Mit dem Befehl wird die Startsequenz der I²C-Bussystems eingeleitet.

        while (1)
        {
// Dies ist die Hauptschleife (main-loop), eine Programmschleife, welche wiederkehrende
// Befehle enthält. Ist sie leer, durchläuft der Controller die Schleife immer wieder, ohne dass
// etwas passiert (nur Strom wird verbraucht). Eine solche Schleife ist notwendig, da es auf
// dem Controller kein Betriebssystem gibt, das nach Beendigung des Programmes die
// Kontrolle übernehmen könnte.

                If (flagISR == 1)
                {
// Abfrage, ob die ISR durchlaufen wurde

                        switch (count)
                        {
// Welchen Status hat die Variable count?

                                case 0:
// Fall 0 kann hier nicht eintreffen, da die Startbedingung einen Interrupt voraussetzt und sich
// durch die ISR die globale Variable um 1 erhöht.

                                        Nop ( );
```

```
                                    I2C2CONbits.SEN = 1;
// Startbedingung

                                    break;
// Rücksprung aus der Anweisung

                                case 1:
                                    Nop ( );
                                    Nop ( );
                                    I2C2TRN = 0x58;
// Slave-Adresse in Hex + R/W-Bit (7 Bit + 1 Bit)
// Errechnet aus Datenblatt (AD/5252) und den Ausschlüssen AD1 & AD2
// Write-Mode 0x58 (AD/5252/1) & 0x5C (AD/5252/2)
// Read-Mode 0x59 (AD/5252/1) & 0x5D (AD/5252/2)
// Nur Adresse 0x2C (AD/5252/1) & 0x2E (AD/5252/2)

                                case 2:
                                    Nop ( );
                                    Nop ( );
                                    test = I2C2RCV;
// Puffer-Empfangsregister auslesen.

                                    I2C2TRN = 0x01;
// Ansteuerung des 1. Registers des AD/5252 (Dual)
// 0x01 (RDAC) & 0x21 (EEMEM)
// Ansteuerung des 2. Registers des AD/5252 (Dual)
// 0x03 (RDAC) & 0x23 (EEMEM)

                                    Break;

                                case 3:
                                    Nop ( );
                                    Nop ( );
                                    test = I2C2RCV;
// Puffer-Empfangsregister auslesen.
```

```
                    hochrampen ( );
// Aufruf der Funktion zum Anfahren der Sollwertspannung
                    break;

                case 4:
                    Nop ( );
                    Nop ( );
                    test = I2C2RCV;
// Puffer-Empfangsregister auslesen.

                    I2C2CONbits.PEN = 1;
// Ausführung der Bus-Stoppbedingung.
// Bus hiernach wieder im I-Mod.

                    break;

                case 5:
                    Nop ( );
                    Nop ( );
                    test = I2C2RCV;
                    deepsleep ( );
// Aufruf der DS-Funktion

                    break;

                case 6:
                    Nop ( );
                    Nop ( );
                    count = 0;
// Die globale Variable count wird zurückgesetzt.

                    break;

                default:
```

```
                                    Nop ( );
                                    Nop ( );
                                    break;
// Für einen nicht definierten Zustand wird keine Aktion ausgeführt

                            }
                            flagISR  =  0;
// Der globalen Variablen flagISR wird der Wert 0 zugewiesen.

                    }
            }
return 0;
}
// Rücksprung aus der Schleife
// Ende der Hauptschleife

// Es folgt die IR

/ ********************************************************************
 *   Funktion:   void__attribute__( (interrupt, no_auto_psv) )  _MI2C2Interrupt (void)   *
 ********************************************************************
*   Eingang:            : /                                                    *
*   Ausgang:            : /                                                     *
*********************************************************************
*   Überblick:          Inkrementiert die globale Variable count, setzt die globale      *
*                       Variable flagISF auf 1 und setzt das Interrupt Flagbit zurück.   *
*   Bemerkung:          Die Funktion wird durch den Interrupt des I²C-Moduls (2)         *
*                       aufgerufen.                                                      *
*********************************************************************/

void __attribute__ ( (interrupt, no_auto_psv) )  _MI2C2Interrupt (void)
            {
//Interrupt-Service-Routine

                    Nop ( );
```

```
                Nop ( );
                Nop ( );
// Breakpoints

                Count  + +;
// Die globalen Variable count wird inkrementiert.

                flagISR  =  1;
// Der globalen Variable flagISR wird der Wert 1 zugewiesen.

                IFS3bits.MI2C2IF  =  0;
// Das Interrupt-Status-Flag wird zurückgesetzt.
        return;
        }

/ ********************************************************************
 *   Funktion:          void hochrampen (void)                        *
 ********************************************************************
*   Eingang:           : /                                           *
*   Ausgang:           : /                                           *
********************************************************************
*   Überblick:         Fährt Ist-Wert langsam an den Soll-Wert heran.  *
*   Bemerkung:         Aufgerufen durch den 4. Case-Fall der switch-Anweisung  *
********************************************************************/

void hochrampen (void)
        {
// Funktion zum Anfahren der Bias-Versorgung

                If (sollwert  !=  0)
                {
// Soll-Wert ungleich Null?

                        while (istwert  !=  sollwert)
                                {
```

```
//Überprüfung
                                        Nop ( );
                                        Nop ( );
                                        test = I2C2RCV;
// Puffer-Empfangsregister auslesen.

                                        istwert ++;
// Ist-Wert wird inkrementiert

                                        I2C2TRN = istwert;
// Ist-Wert wird über den Bus zum AD/5252 übertragen.

                                        while ( I2C2STATbits.TRSTAT == 1)
                                                {
                                                Nop ( );
                                                }
// Warten bis die Übertragung des Datenbytes abgeschlossen ist.

        Nop ( );
        Nop ( ):
        Nop ( );
        zeit ( );
// Aufruf der Zeitfunktion

        Nop ( );
        Nop ( ):
        Nop ( );

                                }
                        istwert = sollwert;
// Soll-Wert wird Ist-Wert zugewiesen

                }
                                else
                                I2C2TRN = 0x00;
```

```
                    test = I2C2RCV;
// Die Spannung bzw. der Widerstand wird auf Null gesetzt (unterste Schleiferstellung)

                    Istwert = sollwert;
// Soll-Wert wird Ist-Wert zugewiesen

                    while (I2C2STATbits.TRSTAT == 1)
                        {
                            Nop ( );
                        }
// Warten, bis die Übertragung des Datenbytes abgeschlossen ist.

        return;
        }

/ ********************************************************************
 *   Funktion:          void zeit (void)                               *
 ********************************************************************
*   Eingang:           : /                                            *
*   Ausgang:           : /                                            *
*********************************************************************
*   Überblick:         Zeitschleife                                   *
*   Bemerkung:         Die Dauer der Zeitschleife wird durch die globale Variable  *
*                      long timer festgelegt                          *
*********************************************************************/

void zeit (void)
        {
// Funktion zur Generierung einer Zeitschleife

            for (a = 0; a <= timer; a++)
// For-Schleife, globale Variable a wird solange erhöht bis der Wert der globalen Variable
// timer überschritten wird.

                {
```

```
                    Nop ( );
                }
// Während die Schleife läuft, wird keine Funktion ausgeführt.
        return;
        }

/ ********************************************************************
 *   Funktion:          void deepsleep (void)                          *
 ********************************************************************
*    Eingang:           : /                                            *
*    Ausgang:           : /                                            *
*********************************************************************
*    Überblick:         PIC wird in den DS-Mode versetzt               *
*    Bemerkung:         PIC kann durch verschiedene Ereignisse wieder geweckt werden, *
*                       POR wird ausgeführt.                           *
*********************************************************************/

void zeit (void)
        {
// Funktion DS

                Nop ( );
                Nop ( ):
                Nop ( );
                DSCONbits.DSEN  =  1;
// Mikrocontroller möchte sich in den Tiefschlafmodus begeben.

                Nop ( );
                mPWRMGNT_ GotoSleepMode ( );
// Bestätigung innerhalb von drei Zyklen durch Command-Befehl.

                Nop ( );
        return;
        }
```

8. Fehler und Schwierigkeiten

In diesem Kapitel werden die schwerwiegendsten Fehler und Probleme beschrieben, die bei der Arbeit mit Mikrocontrollern auftreten können und auch bei der Bearbeitung dieses Projekts gelöst werden mussten. Es gilt zu beachten, dass Hardware und Software zu gleichen Teilen am Auftreten eines Fehlers beteiligt sein können.

8.1 Hardware

Durch den Aufbau des Testboards auf einer Lochrasterplatine können unregelmäßig kalte Lötstellen oder ungewollte Kurzschlüsse entstehen. Daher sollten alle Verbindungen des fertigen Boards mit dem Messgerät überprüft werden.

Zu beachten ist, dass die linke und rechte Lochreihe auf der Europakarte komplett verbunden ist. Um einen späteren Austausch der Bauteile zu erleichtern, sollten möglichst Sockel eingesetzt werden.

Ein Ersatzbaustein sollte für eventuelle Fehler stets zur Verfügung stehen.

Der erste eingesetzte Mikrocontroller während dieses Projekts hatte einen kleinen Defekt. Aber bei den ersten Programmierversuchen wird der Fehler nicht in der Hardware vermutet. Deshalb folgten zahlreiche Tage der Programmänderung und des mehrfachen Durchmessens der Schaltung. Erst nach dem Austausch des Mikrocontrollers ergaben sich sinnvolle Ergebnisse.

Einige digitale Potenziometer haben einen *Write Protect Pin,* der bei auf *aktiv-Low* voreingestellt ist. An diesen Pin muss somit ein *High-Pegel* angelegt werden, damit die Kommunikation problemlos funktioniert.

Die exakte Studie sämtlicher Pinbelegungen ist daher sehr wichtig und kann eine zeitintensive Fehlersuche ersparen.

Der *In-Curcuit-Debugger* ist ebenfalls nicht hundertprozentig verlässlich. Wenn ein Programm nicht richtig funktioniert oder MPLAB eine Fehlermeldung ausgibt, hilft es häufig, den Debugger noch einmal neu anzuschließen, inklusive des USB-Kabels am PC.

In den meisten Fällen sollten die Fehler in der Software gesucht werden.

Bei Nichtbeachtung der oben genannten Punkte besteht die Gefahr einer höchst zeitintensiven Fehlersuche.

8.2 Software

Viele Fehler können durch aufmerksames Lesen bereits im Vorfeld vermieden werden.

Alle Datenblätter und Anleitungen sind in englischer Sprache verfasst.

Ein einziges ungenau übersetztes Wort kann bereits eine langwierige Fehlersuche nach sich ziehen.

Es sollte beachtet werden, dass ein Mikrocontroller mehrere identische Module besitzen kann, z.B. zwei I²C-Module. Dies führt zu doppelten zugehörigen Registern. Im Datenblatt wird jedoch nur ein einzelnes Register aufgeführt, z.B. I2CXCON (Kontrollregister-I²C-Bus). Das Wissen, dass *X* im Registernamen für das entsprechende Modul steht, muss also vorhanden sein. Soll das Register des zweiten I²C-Moduls angesprochen werden, muss I2C2CON aufgerufen werden.

Alle *Application Notes,* die auf der Homepage des Herstellers zu finden sind, sind bedauerlicherweise komplett auf den Assembler-Code ausgelegt und ein Befehl, mit welchem die einzelnen Bits des Registers angesprochen werden können, ist nicht aufgeführt.

Das Aktivieren der I²C-Ports stellte sich als besonders schwierig heraus. Wenn ein Pin die Funktion eines analogen Eingangs besitzt, wird diese Eigenschaft mit Priorität behandelt. Die analogen Eingänge müssen daher deaktiviert werden, bevor das I²C-Modul reibungslos aktiviert werden kann. Diese Information ist aus dem Datenblatt nicht ersichtlich. Lediglich die notwenige Aktivierung der I²C-Module durch den Registerbefehl I2C1CONbits.I2CEN = 1 wird erwähnt. Das Statusregister des I²C-Moduls zeigt jedoch auch dann eine erfolgreiche Aktivierung an, wenn die notwendige Deaktivierung zuvor nicht erfolgt ist und das Programm daher nicht funktionieren kann.

Funktioniert das Programm auch nach Beheben dieses Fehlers nicht, kann unter Umständen das *Silicon-Errata-Datasheet* Abhilfe schaffen.

Auf dem Datenblatt sind alle bekannten Chip-Fehler der jeweiligen Produktionsserie aufgelistet. Um diese Information nutzen zu können, muss die *Device-ID* des Mikrocontrollers bekannt sein. Diese befindet sich Output-Fenster von MPLAB.

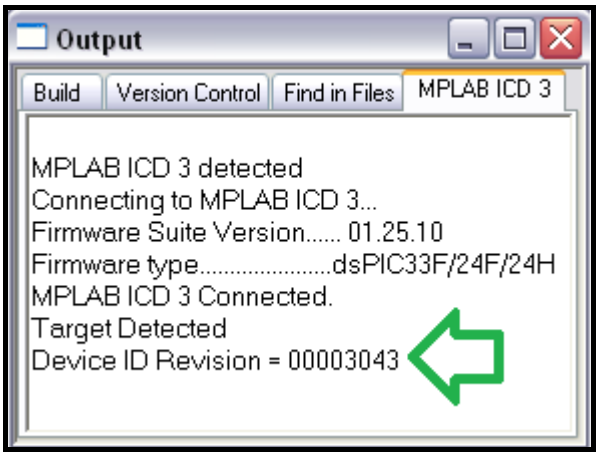


Abbildung 8.1: Device ID Revision

Anhand der *Device-ID* kann die Produktionsnummer des Mikrocontrollers in der Tabelle *SILICO DEVREV VALUES* bestimmt werden.

Für dieses Projekt entsprach sie B5.

Im Falle des B5 ist das erste I²C-Modul gemäß *SILICON DEVREV VALUES* mit einigen Fehlern behaftet. Hier hilft nur das Umschreiben aller Register auf ein Ersatzmodul.

Eine weitere Tücke im Ablauf ist, dass sich das Puffer-Empfangsregister nach einer Datenbyte-Übertragung nicht selbstständig zurücksetzen kann. Das Register muss manuell ausgelesen werden, damit es für das folgende Datenbyte wieder zur Verfügung steht.

Eine allgemeine Hilfestellung bietet die Aufzählung aller Register im Fenster *Watch,* siehe *Abb. 8.2*.

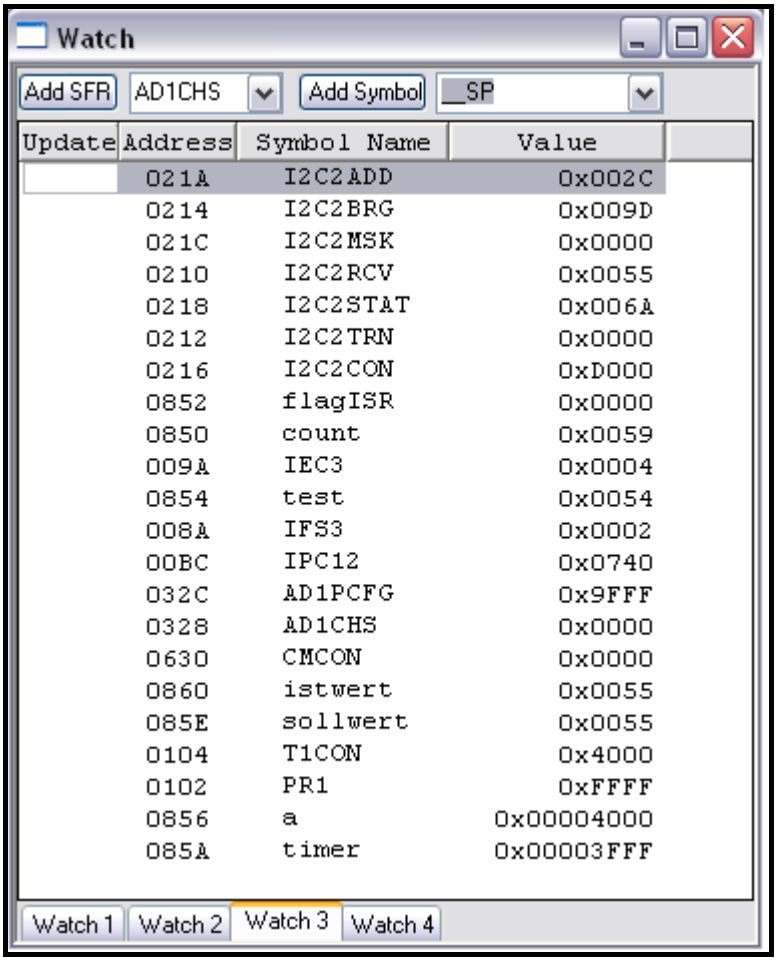


Abbildung 8.2: Die Registeranzeige im Fenster *Watch*

Hier können viele Fehler beim Debuggen des Mikrocontrollers aufgedeckt werden.

Ausführliche Kommentare im Quellcode erleichtern eine spätere Fehlersuche ebenfalls.

9. Zusammenfassung

In diesem letzten Kapitel wird ein Ausblick auf die kommenden Wochen und Monate des Diplomanden und seines Projekts gegeben. Zusätzlich soll ein kurzes Fazit der vergangenen Projektarbeit gezogen werden.

9.1 Ausblick

Die Fertigstellung dieses Projektes ist das höchste Ziel des Diplomanden.

Die Lochrasterkarte war für die ersten Funktionen völlig ausreichend. Durch die aufwendige Verdrahtung und zahlreiche Lötstellen, können jedoch auch weiterhin Probleme und Wackelkontakte auftreten.

Die Funktion des Testboards soll noch erweitert werden. Deshalb wird zurzeit an einer gefertigten Karte gearbeitet, die ausschließlich aus SMD-Bausteinen besteht. Durch Steckverbindungen soll diese Karte auch um mehrere Digital-Potenziometer erweiterbar sein. Diese können dann zu höherer Auflösung kaskadiert werden. Desweiteren soll die Karte über eine RS232-Schnittstelle verfügen.

Auf der neuen Karte werden im Anschluss der I²C-Bus und der Tiefschlafmodus ausführlich getestet. Der Quellcode wird um neue Funktionen erweitert und optimiert. Außerdem soll der Mikrocontroller noch auf plötzlich auftretende Fehler reagieren, was im Programmcode berücksichtigt werden muss.

Der Abschluss des Projektes soll in der Möglichkeit bestehen, den Mikrocontroller aus dem Tiefschlafmodus zu wecken und die Soll-Werte der Spannungen zu ändern.

Wenn dies erfolgreich getestet wurde, kann das kompakte RF-Modul gefertigt werden.

9.2 Schlusswort

Wie in *Kapitel 8* beschrieben, können zahlreiche Probleme auftreten, ehe die ersten Funktionen realisiert werden können. Hier gilt das Sprichwort *Learning by Doing*.

Die Ausarbeitung dieser wissenschaftlichen Arbeit hat dem Diplomanden einige Erfolgserlebnisse beschert, besonders zum jetzigen Zeitpunkt, da zahlreiche Fehler bereits beseitigt werden konnten. Daher wird das funktionstüchtige RF-Modul aller Wahrscheinlichkeit in absehbarer Zeit realisiert werden können.

Diese Arbeit soll sowohl einen anschaulichen Leitfaden zur Programmierung eines Mikrocontrollers bieten als auch ein Nachschlagewerk für fortgeschrittene Programmierer sein.

10. Anlagen

Das Literaturverzeichnis wurde komplett mit **Citavi Pro 2.5.2.0** erstellt.

10.1 Literatur- Quellenverzeichnis

Microchip
http://www.microchip.com/
Datenblätter, Spezifikationen, Handbücher und Anleitungen. **[1]**

NXP
http://www.nxp.com/
Spezifikationen I²C-Bus. **[2]**

Max-Planck-Institut für Radioastronomie
http://www.mpifr-bonn.mpg.de/ und Intranet
Infos übers Institut, Radioteleskop Effelsberg und Bilder. **[3]**

http://de.wikipedia.org/
Info und Diagramm Wellenspektrum **[4]**

RS-Components GmbH
http://de.rs-online.com
Lieferant der Bauteile, Mikrocontroller, Explorer 16 und Debugger.

Mikrocontroller
http://www.mikrocontroller.net/
Forum

Dip-Arbeit Martin Müller
Thema: Konstruktion eines Kryostaten

Altenburg, Jens (2000): **Mikrocontroller-Programmierung**. Assembler und C-Programmierung mit der ST7-Mikrocontrollerfamilie ; mit einer CD-ROM. München: Hanser.

Cover:

[5]

Erlenkötter, Helmut (2009): **C. Programmieren von Anfang an**. 16. Aufl. Reinbek bei Hamburg: Rowohlt-Taschenbuch-Verl. (Grundkurs Computerpraxis, 60074).

Cover:

[6]

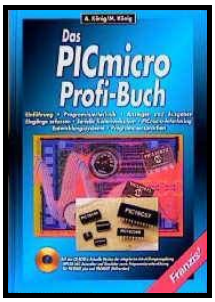
Kohtz, Dieter (2000): **Messen, Steuern und Regeln mit PIC-Mikrocontrollern**. Schaltungen und Programme für Praxis und Hobby ; mit 18 Tabellen ; [PIC-16C5X-Familie, PIC 16C71, PIC 16C84, Entwicklungssysteme, Einführung in die Programmierung, Anwendungen]. 2., verb. Aufl. Poing: Franzis.

Cover:

[7]

König, Anne (1999): **Das PICmicro-Profi-Buch**. Einführung, Programmiertechnik, Anzeigen und Ausgeben, Eingänge erfassen, serielle Kommunikation, PICmicro-Interfacing, Entwicklungssysteme, Programmiersprachen; mit 65 Tabellen. König, Manfred (Hg.). Poing: Franzis.

Cover:



[8]

König, Anne (2001): **Das PICmicro-Hochsprachenbuch**. Der Umgang mit Hochsprachen in der Entwicklungsumgebung MPLAB von Microchip. König, Manfred (Hg.). Poing: Franzis.

Cover:

[9]

König, Anne (2007): **Das große PIC-Micro Handbuch**. 2., überarb. Aufl. König, Manfred (Hg.). Poing: Franzis (Franzis PC & Elektronik).

Cover:

[10]

Lehmann, Stefan; Harth, Wolfram (2007): **PIC-Microcontroller-Programmierung**. [einfaches Programmieren mit BASIC ; zahlreiche Anwendungsbeispiele: von der blinkenden LED bis zum Schrittmotor ; über das Buch erhältlich: vorprogrammierter Übungsbaustein und Testplatine]. 2., überarb. und erw. Aufl. Heidelberg: mitp.

Cover:

[11]

Michael Hofmann; Hofmann, Michael (2009): **Mikrocontroller für Einsteiger**. Schaltungen entwerfen und Software programmieren; [auf CD-ROM: Beispielprogramme, Layoutdaten, Schaltpläne, Datenblätter]. Poing: Franzis (Franzis PC & Elektronik).

Cover:

[12]

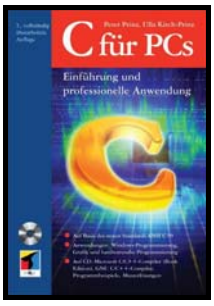
Mumm, Thorsten (2006): **PICs für Einsteiger**. Tipps und Tricks rund um das PICkit 1 Flash Starter Kit. Poing: Franzis (Elektronik).

Cover:

[13]

Prinz, Peter; Kirch-Prinz, Ulla (2002): **C für PCs**. [Einführung und professionelle Anwendung ; auf der Basis des neuen Standards ANSI C 99 ; Anwendungen: Windows-Programmierung, Grafik und hardwarenahe Programmierung ; auf CD: Microsoft C/C++-Compiler (Book Edition), GNU C/C++-Compiler, Programmbeispiele, Musterlösungen]. 3., vollst. überarb. Aufl. Bonn: mitp-Verl.

Cover:



[14]

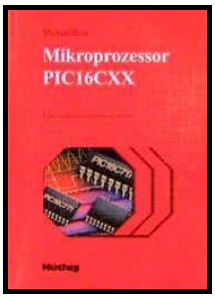
cPrinz, Peter; Kirch-Prinz, Ulla (2004): **C - kurz & gut**. Dt. Orig.-Ausg., 1. Aufl., 1. korr. Nachdr. Beijing: O'Reilly (O'Reillys Taschenbibliothek).

Cover:

[15]

Rose, Michael (2000): **Mikroprozessor PIC16CXX**. Architektur und Applikation. 2., völlig neubearb. u. erw. Aufl. Heidelberg: Hüthig.

Cover:



[16]

Schmitt, Günter (2008): **PIC-Microcontroller**. Programmierung in Assembler und C - Schaltungen und Anwendungsbeispiele für die Familien PIC18, PIC16, PIC12, PIC10. München: Oldenbourg

Cover:

[17]

Wüst, Klaus (2009): **Mikroprozessortechnik**. Grundlagen, Architekturen, Schaltungstechnik und Betrieb von Mikroprozessoren und Mikrocontrollern ; mit 44 Tabellen. 3., aktualisierte und erw. Aufl. Wiesbaden: Vieweg + Teubner (Studium).

Cover:

[18]

10.2 DVD

Inhalt:

Diplomarbeit (pdf-Datei)

Datenblätter

Anleitungen

Projektdateien (MPLAP)

Pläne, Bilder und Screenshots

Vorträge der Diplomanden-Runde