Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Jan Schlüter

Entwurf und Realisierung eines modular aufgebauten Experimentierboards für Mikrocontroller

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Jan Schlüter

Entwurf und Realisierung eines modular aufgebauten Experimentierboards für Mikrocontroller

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Jochen Schneider
Zweitgutachter : Prof. Dr.Ing. Karl-Ragmar Riemschneider

Abgegeben am 28. Februar 2013

Jan Schlüter

Thema der Bachelorthesis

Entwurf und Realisierung eines modular aufgebauten Experimentierboards für Mikrocontroller

Stichworte

Mikrocontroller, ARM Cortex-M3, Stellaris LM3S9D92, Experimentierboard, Registerebene, Platinendesign

Kurzzusammenfassung

Diese Arbeit umfasst den Entwurf und die Realisierung eines Experimentierboards für den Stellaris LM3S9D92 Mikrocontroller. Das Board wurde mit EAGLE entworfen und im Anschluss gefertigt. Um die grundlegende Peripherie des Mikrocontrollers zu erlernen, sind drei weitere Modulplatinen gefertigt worden. Testprogramme für das Board sind auf Registerebene des Mikrocontrollers geschrieben worden.

Jan Schlüter

Title of the paper

Design and implementation of a modular constructed experimentationboard for microcontrollers

Keywords

Microcontroller, ARM Cortex-M3, Stellaris LM3S9D92, Experimentationboard, Registerlevel, PCB design

Abstract

This report includes the design and the implemantation of an experimentationboard for the Stellaris LM3S9D92 microcontroller. The board has been designed with EAGLE and has been fabricated postdoctorated. For learning the fundamental peripherie of the microcontroller three additional module boards have been fabricated. The testsoftware for the board has been written on the register level of the microcontroller.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Einleitung

In den 1950er und 1960er Jahren wurden elektrotechnische Schaltungsaufgaben durch fest verdrahtete Analog- und Logikschaltungen realisiert. Im Laufe der Zeit wurden die Anforderungen an die Schaltungen immer komplexer und aufwendiger. Um die neuen Anforderungen zu erfüllen, mussten die Schaltungen flexibler und einfacher umsetzbar werden. Aus diesem Grund wurden dann programmierbare Logikbausteine, die sogenannten Field Programmable Gate Arrays, entwickelt. Diese Logikbausteine basaßen einen programmierbaren Befehlsspeicher, so dass jeder Logikbaustein speziell für seine Anwendung programmiert werden konnte.
Durch fortschreitende Entwicklung, Miniaturisierung und das Hinzufügen weiterer Funktionen zu den programmierbaren Logikbausteinen wurde 1971 der erste Mikrocontroller von der Firma Intel entwickelt. Dieser war auf die Verarbeitung von 4-Bit-Datenworten ausgelegt und wurde mit einem Takt von 0,740MHz betrieben. Durch kontinuierliche Forschung und Weiterentwicklung der Mikrocontroller gibt es diese heutzutage mit 8-, 16- und 32-Bit-Datenworten. Sie arbeiten oftmals mit einem Takt von 16 MHz. Außerdem haben die Mikrocontroller ein breites Spektrum an zusätzlichen Peripherien, wie Timer, Analog/Digital Wandler, serielle Schnittstellen und vielem mehr, erhalten.
Weitverbreitete Typen sind die Advanced RISC[1] Maschine (ARM) Mikrocontroller. Bei ARM handelt es sich um ein von der Firma Acorn entwickeltes 32-Bit-Chip-Design, das im Jahre 1983 entwickelt wurde. 1990 wurde die ARM-Sparte von Acorn in die Firma ARM Limited ausgelagert. Diese entwickelt seitdem das ARM-Chip-Design weiter. ARM Limited baut die Mikrocontroller jedoch nicht selbst, sondern vertreibt Lizenzen an diverse Halbleiterhersteller, die letztendlich die ARM-Mikrocontroller fertigen. Diese Arbeit beschäftigt sich damit, ein Experimentierboard für einen ARM-Mikrontroller, der seit 2005 gebauten ARMv7-M Familie zu entwerfen und zu realisieren.

[1] Reduced Instruction Set Computer: Computer mit reduziertem Befehlssatz

1.2 Aufgabenstellung

Im Zuge der ständigen Weiterentwicklung der Mikrocontroller will die Hochschule für Angewandte Wissenschaften (HAW) ihre Lehre vom Hitachi H8S Mikrocontroller auf einen ARM basierten Mikrocontroller umstellen. In den Hochschullaboren wird dazu das EKI-LM3S9B92 Evalutions Kit von Texas Instruments (TI) verwendet. Um den Einstieg für Studenten mit dem neuen Mikrocontroller zu erleichtern, soll ein Experimentierboard für einen ähnlichen oder gleichen ARM-Microkontroller entworfen und gefertigt werden. Anforderungen an das Board sind, dass es relativ einfach nachgebaut werden kann und dass die Möglichkeit besteht, die vorhandenen Mikrocontrollerperipherien, verpackt in einzelne Versuche, nach und nach zu konfigurieren und in Betrieb zu nehmen. Hat ein Student das Experimentierboard nachgebaut und alle Versuche durchlaufen, sollte er die grundlegende Funktion der Peripherien des ARM- Mikrocontroller verstanden haben und in der Lage sein, einen ARM Mikrocontroller selbstständig zu konfigurieren und für seine Zwecke zu nutzen.

2 Entwurf

2.1 Modulare Bauweise des Experimentierboards

2.1.1 Grundidee

Um der Anforderung nach einem möglichst einfachen Nachbau an das Experimentierboard gerechet zu werden, soll das Board modular aufgebaut werden. Modular bedeutet, dass das Board in unterteilten Einheiten gebaut werden soll. Baut ein Anwender das Experimentierboard selbst auf, müssen nicht gleich alle Module aufgebaut werden. Zum grundsätzlichen Betrieb werden nur die Grundmodule, sowie eine Spannungsversorgung und der Mikrocontroller benötigt. Um das Experimentierboard für jedermann zugänglich zu machen, soll das Board auch ein Programmermodul enthalten, dass ermöglicht, den Mikrocontroller per Universal Serial Bus (USB) an den PC anzuschließen und zu programmieren. Alle Anschlüsse des Mikrocontrollers sollen auf Buchsen geführt werden, damit das Board auch eigenständig als Mikrocontrollerboard mit Programmer genutzt werden kann. Will der Anwender nun mehr über die Peripherien des Mikrocontrollers lernen, sucht er sich das dazugehörige Experiment aus, baut das benötigte Hardwaremodul auf und kann das Experiment durchführen. Um bereits Gelerntes zu vertiefen, sind die einzelnen Versuche so aufgebaut, dass vorangegangene Versuche mit einbezogen werden können oder sogar sollen. Um den Mikrocontroller mit den einzelnen Modulen zu verbinden, sind einfache Steckbrücken vorgesehen, die es ermöglichen, die einzelnen Module nach dem Baukastenprinzip zusammenzuschalten.

2.1.2 Konzept

Zu Beginn der Konzepterstellung ist es wichtig zu überlegen, welche Mikrocontrollerperipherien ein Einsteiger kennenlernen sollte und wie man diese in einzelne Versuche verpacken kann. Nach den einzelnen Versuchen richten sich letztendlich auch die einzelnen Module, die auf dem Experimentierboard aufgebaut werden müssen. Außerdem sollte berücksichtigt werden, dass die Versuche im Schwierigkeitsgrad ansteigen. Das erste Modul sollte also auch das verständnismäßig leichteste sein.

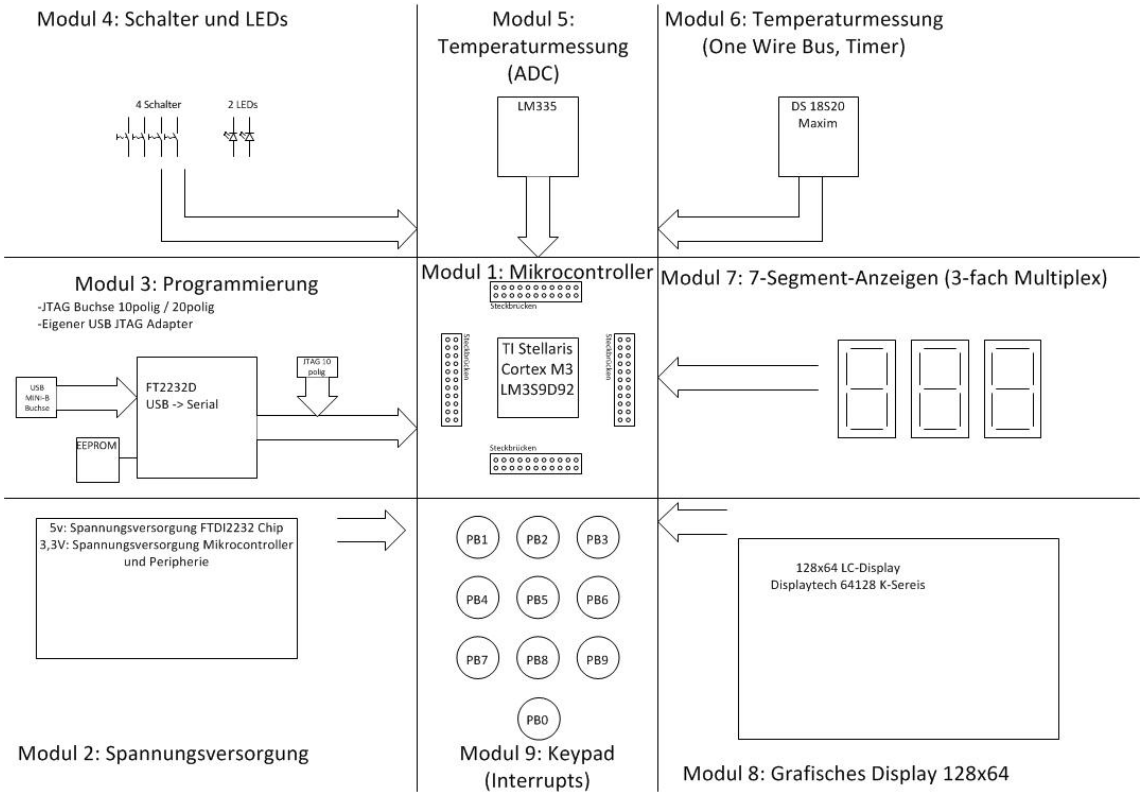


Abbildung 2.1: Konzeptionelle Darstellung des Experimentierboards für Mikrocontroller

Aus diesen Überlegungen ist das Konzept aus Abbildung 2.1 enstanden. Das Konzept zeigt, dass das Experimentierboard aus neun Hauptmodulen besteht.

- **Modul 1: Mikrocontroller**
  Der Mikrocontroller stellt das Herzstück des Experimentierboards dar. Um ihn herum muss das Board designed werden. Er stellt auch die Peripherie-Funiktionen zur Verfügung, die für alle weiteren Experimente benötigt werden.
- **Modul 2: Spannungsversorgung**
  Die Spannungsversorgung soll die beiden gebräuchlichsten Spannungen der Schaltungstechnik, Transistor-Transistor-Logik 5 V und Low-Voltage-Transistor-Transistor-Logik 3,3 V, bereitstellen. Somit ist sichergestellt, dass die meisten Integrated Circuit (IC) Bausteine versorgt werden können.
- **Modul 3: Programmierung**
  Das Modul 3 soll einen selbstgebauten USB->JTAG[2] Programmer beinhalten. Es soll

[2]Joint Test Action Group: IEEE1149.1 Standard, der das Programmieren, Debuggen und Testen von Mikrocontrollern oder FPGAs beschreibt

eine kostengünstige Alternative zu einem fertigen JTAG-Programmer aufzeigen. Die Möglichkeiten, den Controller mit einem externen JTAG-Programmer zu programmieren, soll erhalten bleiben. Aus diesem Grund ist eine 10-Pin-JTAG-Buchse vorgesehen.

- **Modul 4: Schalter, Taster und LED**
  In Modul 4 sollen einfache Ein- und Ausgangsfunktionen des Mikrocontrollers in Betrieb genommen werden. Zu diesem Zweck soll das Experimentierboard mit vier Schaltern und zwei LED‘s bestückt werden.

- **Modul 5: Temperaturmessung mittels Analog-Digital-Converter (ADC)**
  Modul 5 soll einen einfachen Temperatursensor (LM335) beschalten. Dieser liefert, entsprechend der Temperatur, eine analoge Ausgangsspannung, welche anschließend durch den Mikrocontroller analog-digital gewandelt werden soll. Der Digitalwert kann dann weiterverarbeitet werden und auf einem Ausgabemedium, als Teperaturwert, ausgegeben werden.

- **Modul 6: Temperaturmessung (One-Wire-Bus)**
  In Modul 6 soll ein Temperatursensor (DS1820) über einen One-Wire-Bus betrieben werden. Dies erfordert eine vorgegebene Kommunikationssequenz und deshalb ein genaues Timing, um den Temperatursensor auszulesen. Um die Kommunikation mit dem Temperatursensor zu ermöglichen, muss also der Timer des Mikrocontrollers genauer untersucht werden. Die Ausgabe der gemessenen Werte soll wieder durch die 7-Segment-Anzeigen oder das Liquid Crystal Display (LCD) erfolgen. Modul 5 und Modul 6 lassen sich sehr gut vergleichen, da beide Module eine Temperatur messen, dies aber auf eine unterschiedliche Art und Weise tun.

- **Modul 7: 7-Segment-Anzeige (3-fach-Multiplex)**
  In Modul 7 soll eine 7-Segment-Anzeige mit drei Stellen realisiert werden. Um Pins einzusparen, sollen die Anzeigen im Multiplex betrieben werden. Mittels der 7-Segment-Anzeige sollen gemessene Temperaturwerte oder Statusmeldungen des Mikrcontrollers angezeigt werden. Außerdem wird eine Codierung von Binary Coded Decimal[3] (BCD) auf die 7-Segment-Anzeige notwendig.

- **Modul 8: Grafisches Display 128x64 Pixel**
  In Modul 8 soll ein LCD an den Mikrocontroller angeschlossen werden. Es soll im Parallelbetrieb arbeiten. Auf dem Display können Statusmeldungen, Temperaturwerte, Buchstaben, Zahlen und Bilder angezeigt werden. Der Anwender lernt hierbei, Bibliotheken einzubinden und das Display in Betrieb zu nehmen. Fortgeschrittene Anwender können erlernen, wie sie selber eine passende Zeichenbibliothek für ein LCD anlegen.

[3] BCD: Binäre Codierung von Dezimalzahlen

- **Modul 9: Keypad (Interrupts)**
  Hier soll eine Art Codeschloss realisiert werden. Es sollen die Ziffern 0-9 zur Verfügung stehen. Jede Zahl wird durch einen entprellten Taster abgebildet. Erfolgt eine Eingabe, soll diese durch eine Interrupt Service Routine (ISR) verarbeitet werden. Ausgaben können wieder auf der 7-Segment-Anzeige oder dem LCD angezeigt werden.

Für fortgeschrittene Benutzer bietet der Mikrocontroller auch viele serielle Kommunikationsschnittstellen. Weil der Mikrocontroller USB und Ethernet Schnittstellen bereitstellt, sind auf dem Experimentierboard ebenfalls eine USB-Buchse und eine Ethernetbuchse vorgesehen. Da diese aber nicht zu den grundlegenden Peripherien eines Mikrocontrollers gehören, wurden keine besonderen Versuche für sie entwickelt. USB- und Ethernetbuchse sind ebenfalls modular und müssen nur dann aufgebaut werden, wenn sie auch tatsächlich benutzt werden sollen.

2.2 Auswahl des Mikrocontrollers

2.2.1 Beweggründe für den Stellaris LM3S9D92

Weil das Experimentierboard in erster Linie für die HAW Hamburg und ihre Studenten entworfen wird, soll der Mikrocontroller des Boards möglichst gleich zu denen der Hochschullabore sein. So brauchen die Studenten nicht zwei Mikrocontroller zu erlernen und können sich mit dem Experimentierboard direkt auf Praktika oder die Vorlesungen Computertechnik und Mikrocontrollertechnik vorbereiten. In den Laboren der HAW wird mit dem Evaluationskit Stellaris EKI-LM3S9B92 von TI gearbeitet. EKI steht hierbei für Evaluation Kit for IAR Systems Embedded Workbench. IAR Embedded Workbench ist die mitgelieferte Entwicklungsumgebung für das Evaluation Board. Der derzeitige Nachfolger für diese Mikrocontroller ist der TI ARM Cortex-M3 Stellaris LM3S9D92. Dieser Mikrocontroller ist pinkompatibel zum Stellaris LM3S9B92 und kann wie dieser eingesetzt werden. Ein weiterer Grund, der für einen TI Mikrocontroller spricht, ist die von TI angebotene Entwicklungsumgebung Code Composer Studio. Sie wird auch als Entwicklungsumgebung an der Hochschule eingesetzt. So haben interessierte Studenten die gewohnte Entwicklungsumgebung für das Experimentierboard und müssen sich nicht extra in neue Software einarbeiten.
Außerdem besitzt der Stellaris LM3S9D92 einen großen Umfang an Peripherien, die das Experimentierboard auch nach dem Einstieg in die Mikrocontrollerprogrammierung zu einem vollwertigen Entwicklungsboard für den LM3S9D92 werden lassen.

2.2.2 Aufbau des Stellaris LM3SD92

Der ARM Cortex-M3 Stellaris LM3S9D92 ist ein Mikrocontroller aus der ARMv7-M Familie. Hersteller des Mikrokontrollers ist die Firma Texas Instruments. Er wird in einem Low Quad Flat Package (LQFP) 100 gefertigt, wie Abbildung 2.2 zeigt.

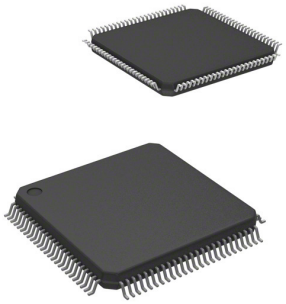


Abbildung 2.2: Stellaris LM3S9D92 Außenansicht

Der Stellaris LM3S9D92 hat ein energiesparendes Leistungsmanagement integriert. Wird er kurzzeitig nicht gebraucht, kann er in einen Sleep Modus gesetzt werden, in dem der Prozessor und das Speichersystem keinen Takt mehr erhalten. Die restlichen Peripherien haben jedoch noch den vollen Systemtakt. Wird der Mikrocontroller für längere Zeit nicht mehr benötigt, kann er in einen Deep Sleep Modus versetzt werden. In diesem Modus werden auch die restlichen Peripherien nur noch mit einem verminderten Takt von 30 kHz versorgt. Das Energiemanagement trägt dazu bei, dass die Leistungsaufnahme des Mikrocontrollers reduziert wird.
Grundsätzlich ist der LM3S9D92 in Harvard Architektur aufgebaut. Das bedeutet, dass der Befehlsspeicher physisch und logisch vom Datenspeicher getrennt ist. Die logische Trennung ergibt sich durch verschiedene Adressräume der Speicher. Die physische Trennung der Speicher erfolgt durch zwei verschiedene Speichermedien, sowie seperate Zugriffsbusse auf diese. Der Befehlsspeicher ist durch einen 512 kB Electrically Eraseable Programmeable Read Only Memory (EEPROM) realisiert worden. Der Datenspeicher liegt in einem 96 kB Static Random Access Memory (SRAM). Vorteil der Harvard Architektur ist, dass während eines Systemtaktes gleichzeitig auf beide Speicher zugegriffen werden kann. Dadurch kann der Mikrocontroller wesentlich schneller Befehle umsetzen. Ein anderer Pluspunkt der Harvard Architektur ist, dass der Programmcode im Befehlsspeicher vor versehentlichem Überschreiben durch Daten geschützt ist, da die Speicherbereiche voneinander getrennt sind. Abbildung 2.3 zeigt den schematischen Aufbau des Stellaris LM3S9D92 Mikrocontrollers. Die oben beschriebene Trennung der verschiedenen Speicher ist gut zu erkennen. Der SRAM Datenspeicher wird durch den Sytem Bus mit dem ARM Cortex-M3 Kern verbunden,

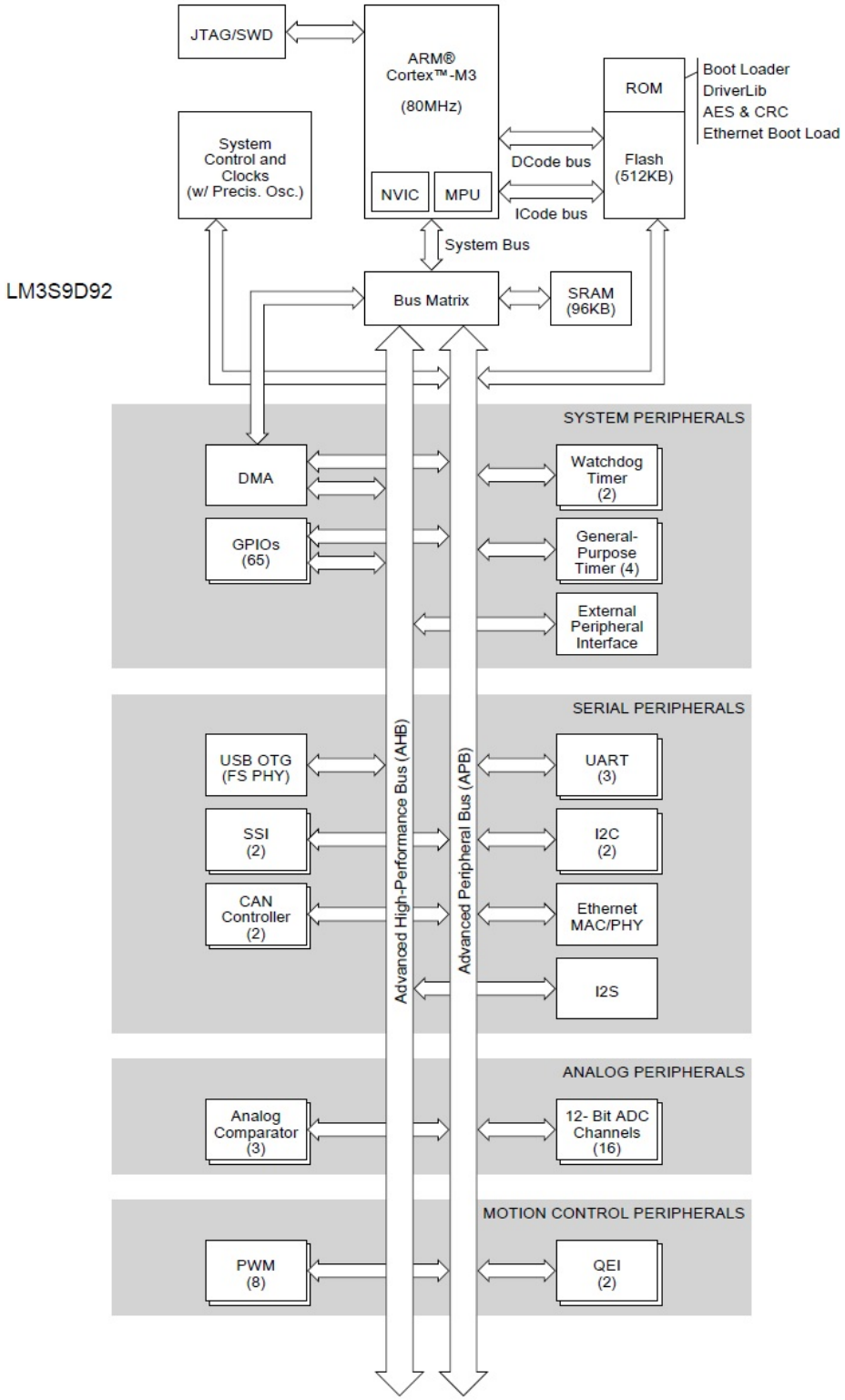


Abbildung 2.3: Blockschaltbild des Stellaris LM3S9D92 Mikrocontroller (Quelle [1])

während der Programmspeicher durch den DCode Bus und den ICode Bus mit dem Controllerkern vebunden ist. Neben den bereits beschriebenen Speichern ist noch ein ein weiterer Read Only Memory (ROM) Speicher zu erkennen. In diesem Speicher sind der Stellaris Bootloader sowie die Stellaris Peripherie Treiber Bibliothek gespeichert. Auf die Funktionalität von Stellaris wird im Abschnitt 4.3 genauer eingegangen. Der Mikrocontrollerkern besitzt eine 32-Bit-Architektur. Dies bringt mit sich, dass Bus und Registerbreiten ebenfalls auf eine 32-Bit-Wortbreite ausgelegt sind.
Direkt am Kern sitzt die Memory Protection Unit (MPU). Diese Einheit weist den zum Kern gehörigen Systemblöcken ihre eigenen Speicherbereiche zu und achet darauf, dass nicht fälschlicherweise auf sie zugegriffen wird.
Der nächste direkt mit dem Kern verbundene Systemblock ist der JTAG/ Single Wire Debug (SWD) Block. Über das JTAG Interface kann ein Programmcode in den Programmspeicher geladen werden. Außerdem können, während des laufenden Programms, Variablen und Werte aus dem Arbeitsspeicher des Mikrocontrollers ausgelesen werden (Debugging).
Der Hardwareblock System Control and Clocks beherbegt die Taktgeber für den Mikrocontroller.

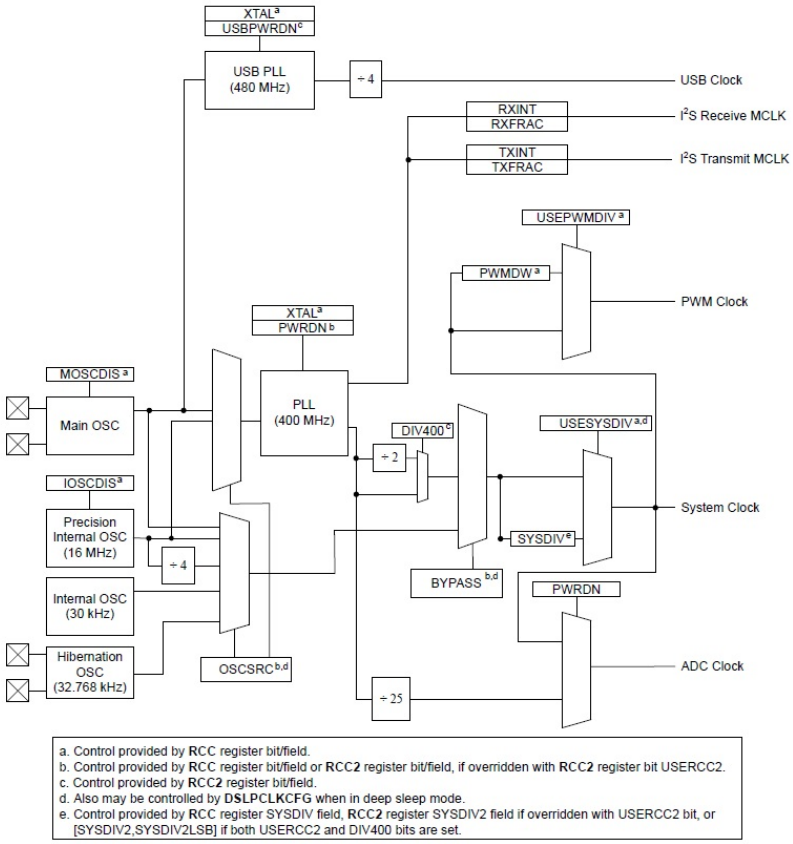


Abbildung 2.4: Taktbaum des Stellaris LM3S9D92 Mikrocontroller (Quelle [1])

Der direkte Takt für den Prozessorkern und die Peripheriefunktionen ist die System Clock. Abbildung 2.4 zeigt, wie der Hardwareblock die System Clock aus den angeschlossenen Oszillatoren generiert. Der LM3S9D92 wird standardmäßig durch einen internen 16MHz Präzessions-Oszillator mit dem Takt versorgt. Es besteht jedoch die Möglichkeit, einen externen Oszillator anzuschließen, um den Takt zu erzeugen. Mit dem Bypass Multiplexer kann gewählt werden, ob der Oszillatortakt auch gleich der Systemtakt ist. Soll der Mikrocontroller jedoch mit einem höheren Takt als 16 MHz betrieben werden, muss er vorher durch eine Phase Lock Loop (PLL), die als Frequenzvervielfacher arbeitet, auf 400 MHz angehoben werden. Die Funktion der PLL ist in der Literatur [4] ab Seite 1164 genauer beschrieben. Der 400 Mhz Takt kann anschließend halbiert oder direkt auf einen Divisionsblock geschaltet werden. Für den halbierten Takt von 200 MHz kann der Divisor frei von 3-64 gewählt werden. Somit liegt der Systemtakt zwichen 66,67 Mhz und 3,125 Mhz. Um die maximale Taktfrequenz des Mikrocontrollers zu erreichen, wird der 400 MHz Takt nicht halbiert sondern direkt auf den Divisionsblock gegeben. In diesem Fall kann der Divisor von 5-128 gewählt werden. Daraus resultieren mögliche Systemtakte von 80 Mhz bis 3,125 MHz. Die Takteinheit generiert weitere Taktdomänen für USB, Inter-Integrated Circuit (I^2C) , Puls Weiten Modulation (PWM) und Analog Digital Converter (ADC).
Der noch ausstehende Hardwareblock ist der Nested Vector Interrupt Controller (NVIC). Er verwaltet auftretende Interrupts, die während des laufenden Programms auftreten. Insgesamt hat der Stellaris LM3S9D92 53 priorisierbare und einen unpriorisierbaren Interrupt. Die genaue Funktion der Interrupts wird im Kapitel 4.4.2 besprochen.
Alle in diesem Abschnitt beschriebenen Hardwareblöcke sind die sogenannte Kernperipherien. Diese Komponenten braucht der ARM Cortex-M3 Kern zwingend, um zu funktionieren.

2.2.3 Verfügbare Mikrocontroller Peripheriefunktionen

In diesem Abschnitt wird die weiteren Mikrocontrollerperipherien betrachtet. Sie wird nicht zwingend zum Betrieb des Mikrocontrollerkerns benötigt. Ohne diese Peripherien hätte der Mikrocontroller jedoch auch keine Funktionen, mit denen der Nutzer arbeiten kann. Sie teilt sich in vier Hauptkategorien auf.

1. System Peripherien
    - Direct Memory Access, DMA
    - General-Purpose Input/Output, GPIOs (65 Stück)
    - Watchdog Timer (2 Stück)
    - General Purpose Timer (4 Stück)
    - External Peripheral Interface

2. Serielle Peripherien
    - Universal Serial Bus On The Go, USB OTG
    - Synchronous Serieal Interface, SSI
    - Controller Area Network Module, CAN (2 Stück)
    - Universal Asynchronous Transmitter/Receiver, UART (3 Stück)
    - Inter-Intergrated Circuit, I^2C(2 Stück)
    - Ethernet
    - Inter Interchip Sound, I^2S
3. Analoge Peripherien
    - Analog Comparator (3 Stück)
    - 12 Bit ADC Channels (16 Stück)
4. Flankengesteuerte Peripherien
    - Pulse With Modulator, PWM (8 Stück)
    - Quadrature Encoder Interface, QEI (2 Stück)

Die Peripheriefunktionen sind über zwei Busse, dem Advanced High-Performance Bus (AHB) und dem Advanced Peripheral Bus (APB), mit dem Kern verbunden und können so angesteuert werden. Im Folgenden wird nur auf die Peripheriefunktionen eingegangen, die auch in den Testprogrammen benutzt werden. Informationen zu weiteren Peripheriefunktionen sind dem Datenblatt des Stellaris LM3S9D92[1] zu entnehmen.

General-Purpose Input/Output, GPIOs

Der LM3S9D92 besitzt 65 programmierbare Ein- und Ausgangspins. Sie sind aufgeteilt auf die Ports A-J. Alle GPIOs sind 5 V tolerant außer Pin PB0 und PB1. Diese beiden Pins sind auf eine Eingangsspannung von 3,6 V limitiert. Es gibt die Möglichkeit jeden Pin als Analog- oder Digitalpin zu konfigurieren. Im Analogbetrieb können die Pins als Inputquelle für den ADC oder als Analogspannungsvergleicher dienen. Wird der Pin als Digitalpin programmiert, ist er pegelgesteuert. Das bedeutet, dass er nur noch zwei Zustände kennt. Zum einen die logische ’0’ (Low-Pegel) und zum anderen die logische ’1’ (High-Pegel). Der Low-Pegel entspricht einem Spannungslevel von 0 V und der High-Pegel einem Spannungslevel von 3,3 V. Digitale Eingangspins werden durch eine Schmitt-Trigger-Schaltung unterstützt. Das bedeutet, dass sie Spannungspegel von -0,3 V bis 1,2 V als Low-Pegel und Spannungspegel von 2,1 V bis 5 V als High-Pegel interpretieren. Die Eingänge können auch als Quelle für externe Inetrrupts dienen.

General-Purpose Timer

Der LM3S9D92 besitzt vier Timer. Sie können als One Shot Timer oder periodische Timer konfiguriert werden. Mit Hilfe der Timer können periodisch wiederkehrende Prozesse wie eine ADC-Umsetzung gestartet werden. Mit Hilfe der Timer lässt sich auch ein pulsweitenmoduliertes Signal erzeugen. Weitere Timer Modi sind Flanken zählen an Eingangspins oder die Zeitmessung zwischen beipielsweise zwei positiven Flanken an einem Eingangspin. Außerdem bietet der Timer eine Echtzeituhr, wenn eine externe Taktfrequenz von 32,768kHz bereitgestellt wird.

Eigentlich sind die Timer keine Uhren sondern nur Zähler, die in einem bestimmten Takt weiterzählen. Jeder Timer hat ein Zählregister. Entweder ist das Zählregister 32 Bit breit und kann somit bis 2^{32}-1=4294967296 gezählt werden, oder der Zähler hat ein 16 Bit breites Zählregister und kann deshalb bis 2^{16}-1=65535 zählen. Hat der Timer ein 16 Bit breites Zählregister, gibt es ein weiteres Register, das einen Vorteiler (Prescaler) für den Zähltakt beinhaltet. Beim LM3S9D92 besteht dieses Register aus 8 Bit und kann deshalb einen maximalen Vorteiler von 2^8=256 haben. Die Zeit, die der Mikrocontroller zum Inkrementieren der Zahl benötigt, hängt vom Takt ab. Bei einem Takt von 80 MHz braucht der Zähler 12,5 ns für ein Inkrement. Bei einem 32 Bit Zähler entspricht das einer maximalen Zählzeit von 53, 6 s. Für das 16 Bit Zählregister, mit einem Vorteiler von 256, ergeben sich 209,7 ms.

Analog Digital Converter, ADC

Der LM3S9D92 besitzt 16 12-Bit-ADC-Eingänge. Aufgabe eines ADC ist es, eine analoge Eingangsspannung in einen Digitalwert, der der Spannung entspricht, umzuwandeln. Standardmässig wird dem ADC eine Referenzspannung von 3 V gegeben. U_{ref} ist gleichzeitig die Maximalspannung U_{max}, die umgesetzt werden kann. Bei 12 Bit ADC Bausteinen hat man eine Auflösung von 2^{12}Bits=4096. Für U_{LSB} ergibt sich daher die Formel 2.1.

$$U_{LSB} = \frac{U_ref}{2^N} = \frac{3V}{2^{12}} = 0,732mV \tag{2.1}$$

Eine Umsetzung kann durch verschiedene Ereignisse gestartet werden: Interrupt, Software, Analog Vergleicher, PWM oder Eingangspin.

3 Umsetzung

3.1 Platinenlayoutsoftware EAGLE

Die Platinenlayoutsoftware EAGLE ermöglicht es, Schaltpläne für Platinen zu erstellen und diese in eine Platinenansicht zu überführen, auf der alle Bauteile vorhanden sind. Auf der Platinenansicht müssen dann die Leiterbahnen geroutet werden. Diese Aufgabe kann durch einen Autorouter übernommen oder von Hand ausgeführt werden. Viele elektronische Bauteile sind in der umfangreichen EAGLE-Bibliothek enthalten und können so einfach in den Schaltplan intergriert werden. Das fertige Platinenlayout kann direkt als Datei an den Fertigungsbetrieb geschickt werden, damit er die Platine fertigen kann. Sollen spezielle Bauteile verwendet werden, ist es möglich, diese in einem Bibliothekeditor zu erstellen und in einer eigenen Bibliothek zu speichern. Für diese Arbeit ist ebenfalls eine Bibliothek für Bauteile, die nicht in EAGLE enthalten sind, erstellt worden. Sie ist auf dem Datenträger zu dieser Thesis enthalten.
Um sämtliche Schaltpläne und Platinendesigns dieser Arbeit zu erstellen, wurde EAGLE in der Version 5.10.0 Light eingesetzt. Sie ist frei für den nichtkommerziellen Gebrauch als Download verfügbar. Die Light Version ist auf eine Platinengröße von 100 mm x 80 mm und zwei Lagen begrenzt. Für dieses Projekt erfüllt die Programmversion alle Anforderungen, da höchstens zweilagige Platinen im halben Euro Format (100mm x 80 mm) vorgesehen sind.

3.2 Übersicht Experimentierboard

Das fertige Experimentierboard besteht aus vier einzelnen Modulplatinen, wie Abbildung 3.1 zeigt. Wären alle Module auf einer Platine umgesetzt worden, wäre die Platinengröße stark gestiegen. Dies hätte höhere Fertigungskosten zur Folge gehabt. Außerdem wäre ein Teil der Modularität verlorengegangen. Herzstück des Experimentierboards ist die Hauptplatine des Boards (Abbildung 3.1, Nr.1). Hierbei handelt es sich um eine doppelseitige Leiterplatte, deren Layout im Rahmen dieser Arbeit entwickelt wurde und anschließend von der Firma Beta Layout gefertigt wurde. Auf der Hauptplatine sind alle wichtigen Module enthalten um den

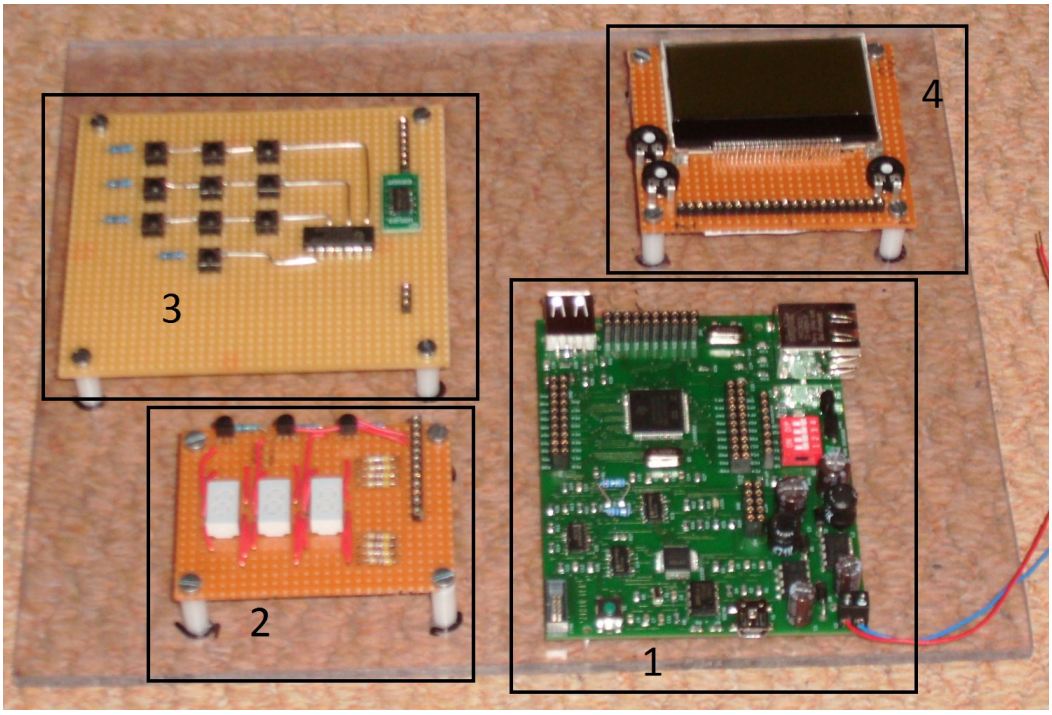


Abbildung 3.1: Gesamtansicht Experimentierboard

Mikrocontroller zu betreiben. Hierbei handelt es sich um die Module Mikrocontroller, Spannungsversorgung, Programmierung, Schalter/LEDs, Temperaturmessung (ADC) und Temperaturmessung (One-Wire-Bus). Außerdem beherbergt die Hauptplatine eine USB- und eine Ethernetbuchse. Mit allen diesen Modulen bestückt ist das halbe Europlatinenmaß, welches maximal für eine Platine zur Verfügung stand, voll ausgenutzt.
Die Module 7-Segment-Multiplex (Abbildung 3.1, Nr.2), Tastermatrix (Abbildung 3.1, Nr.3) und LC-Display (Abbildung 3.1, Nr.4) sind auf weitere externe Platinen ausgelagert worden. Im Prototyp sind die Platinen auf Lochrasterplatinen aufgebaut worden. Für jede dieser Modulplatinen ist ein fertiges Platinenlayout in EAGLE erstellt worden. Sie enthalten die Boarddateien zur Fertigung des Boards, sowie die Bestückungspläne für die Platinen und sind im Anhang A aufgeführt.

3.3 Hauptplatine

3.3.1 Übersicht

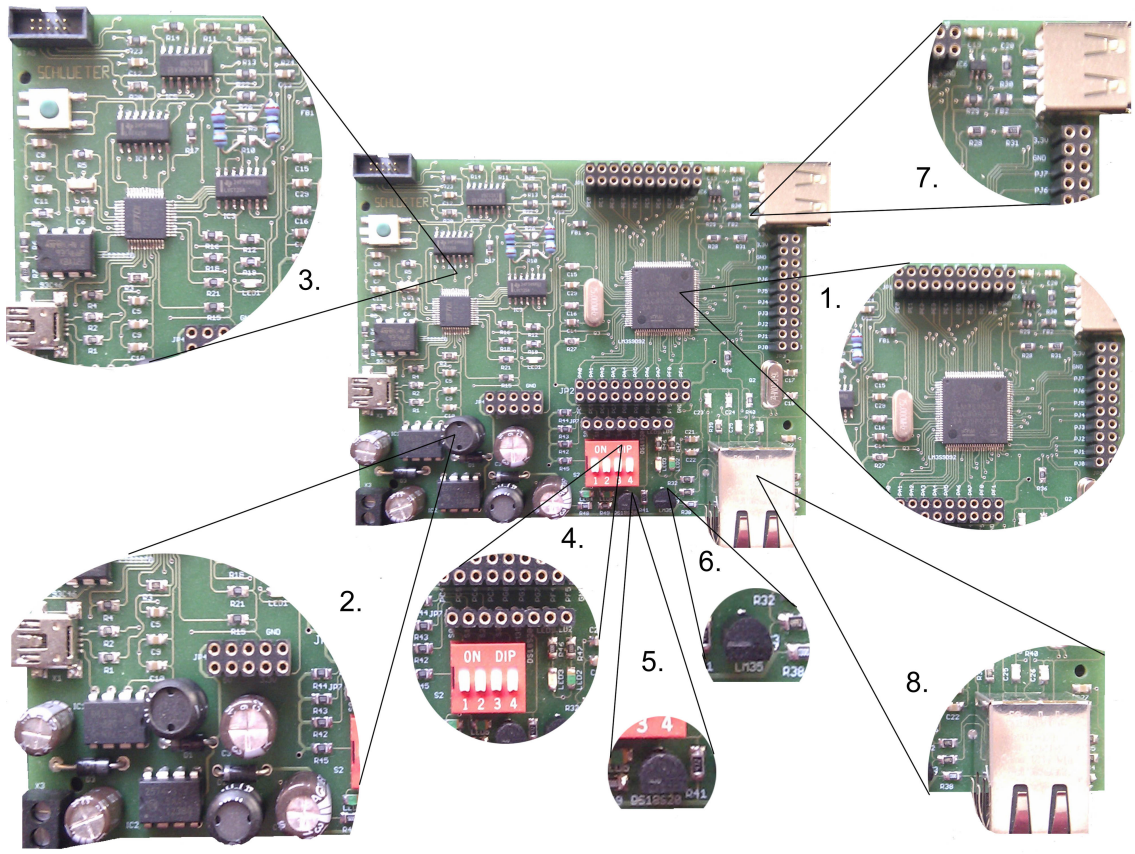


Abbildung 3.2: Draufsicht Hauptplatine, einzelne Module gezoomt

Die Hauptplatine ist eine doppelseitig gefertigte Platine im halben Euro Format (80 mm x 100 mm). Auf ihr wurden die Module 1-6 aus dem Kapitel 2.1.2 umgesetzt. Sie ist also bestückt mit dem Stellaris LM3S9D92 Mikrocontroller, einer Spannungsversorgung für 3,3 V und 5 V, einem USB zu JTAG Programmer für den Mikrocontroller, Schaltern, LEDs und den beiden Temperatursensoren LM35 und DS1820. Um die höchste Flexibilität für den Einsatz des Experimentierboards und des Mikrocontrollers zu haben, ist zunächst jeder freie Pin des Mikrocontrollers auf eine Buchsenleiste geführt worden. Von dort aus können die Pins entweder mit den anderen Modulen oder mit eigenen Schaltungen verbunden werden. Außerdem ist eine USB- und eine Ethernet Buchse auf das Experimentierboard gekommen. Da der Mikrocontroller diese weitverbreiteten und durchaus interessanten Funktionen bietet, sollen diese auch genutzt und von dem Experimentierboard vorgehalten werden. Auf der Hauptplatine wurden die einzelnen Module räumlich so gut wie möglich zusammengehalten, um

das Layout der Platine möglichst einfach nachvollziehbar zu machen. Die gasamte Hauptplatine ist von Hand in EAGLE geroutet worden. Für die Leiterbahnstärken, Bohrungen und Leiterbahnabstände wurden die Vorgaben [5] des Platinenfertigers Beta Layout eingehalten. Leiterbahnen für Datensignale haben eine Dicke von 0,2 mm, Leiterbahnen für die Spannungsversorgung haben, aufgrund der höheren Ströme, eine Dicke von 0,5 mm. Die Werte hierfür wurden ebenfalls aus der Beta Layout Spezifikation [5] unter dem Punkt Strombelastbarkeit von Cu-Bahnen auf Basismaterial ermittelt. Die Platine hat eine Kupferschichtstärke von 35 μm. Bei einer Leiterbahndicke von 0,5 mm ergibt sich bei einer 1 A Strombelastung eine Leiterbahnerwärmung von 10°C. Da die einzelnen Schaltregler zusammen maximal 1 A Stromstärke bereitstellen, wurde diese Leiterbahndicke gewählt.
Beim Routing der Platine wurden die Pins 8 und 11 des IC 3 (SN74LVC125A) miteinander vertauscht. Behoben ist der Fehler auf der Platine durch das Einlöten zweier bedrahteten 27 Ω Metallschichtwiderständen, welche die Pads der eigentlichen Widerstände R9 und R10 vertauschen. Auf der EAGLE-Board-Datei und im Schaltbild (Anhang **??**) ist der Fehler bereits behoben. In Abbildung 3.2 sind die einzelnen Module der Hauptplatine herausgezoomt und nummeriert. Zum Betrieb des Mikrocontrollers auf dem Experimentierboard sind lediglich das Modul 1 (Mikrocontroller), das Modul 2 (Spannungsversorgung) und das Modul 3 (Programmer) notwendig. Alle anderen Module werden nur für spezielle Anwendungen und Versuche benötigt. Nachfolgend sollen die einzelnen Module der Hauptplatine genauer erläutert und auf deren Einzelheiten eingegangen werden.

3.3.2 Mikrocontroller (Abb. 3.2, Nr.1)

Wie bereits in Kapitel 2.2 besprochen, kommt auf der Hauptplatine ein TI ARM Cortex-M3 Stellaris LM3S9D92 Mikrocontroller zum Einsatz. Verwendet wird das LQFP 100 Package. Als Spannungsversorgung werden 3,3 V benötigt. Um das Experimentierboard möglichst offen für jegliche Anwendungen zu halten, sind alle nicht fest benötigten Pins des Mikrocontrollers auf Buchsenleisten geführt. Von dort aus können sie, je nach Anwendung, durch Steckbrücken abgegriffen werden.

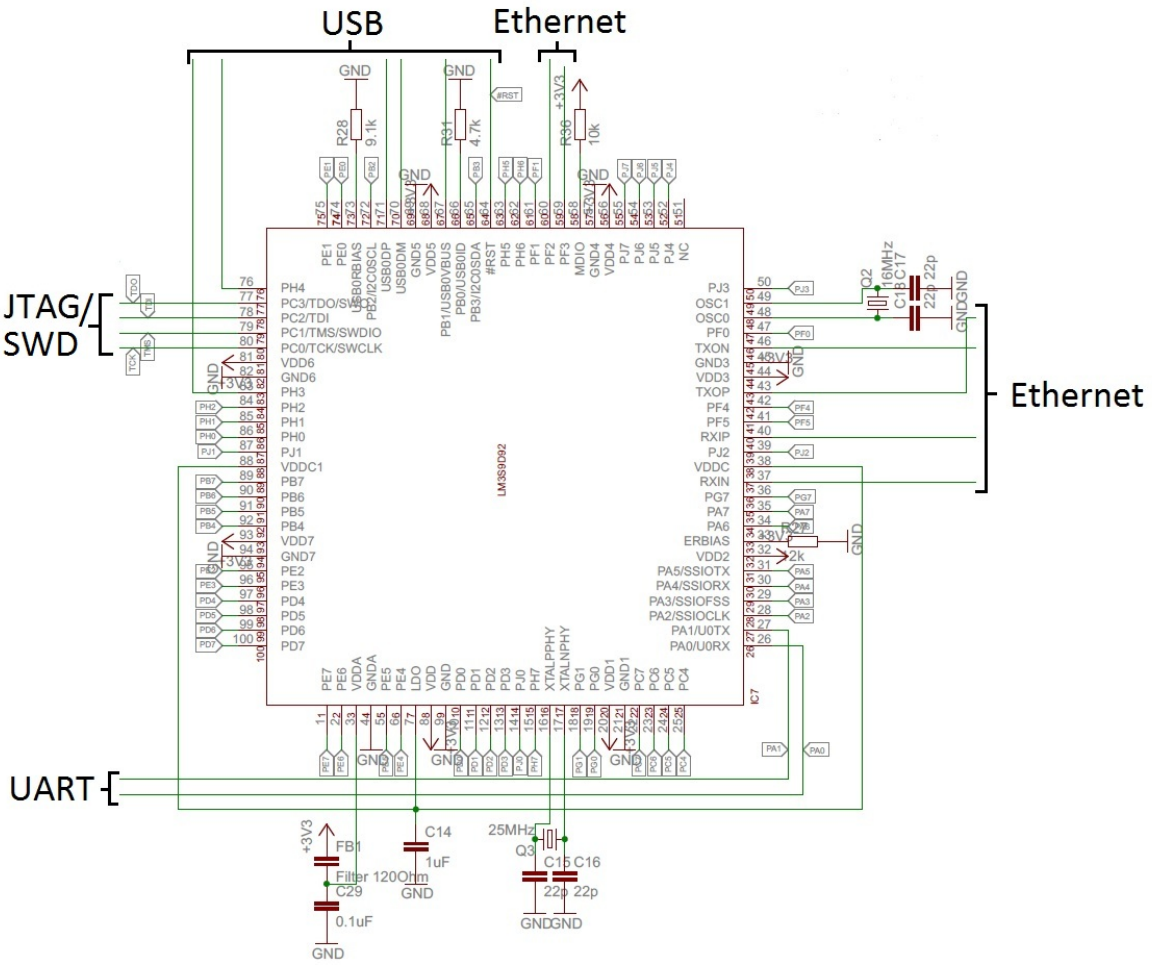


Abbildung 3.3: Stromlaufplan Mikrocontroller Stellaris LM3S9D92

Der Stromlaufplan des Mikrocontrollers (Abbildung 3.3) zeigt, dass nur wenige Pins des Mikrocontrollers fest mit weiteren Modulen auf dem Experimentierboard verbunden sind. Feste Verbindung vom Mikrocontroller gibt es nur zum Programmer(JTAG, UART), zur USB-Buchse und zur Ethernet-Buchse. Außerdem sind die Pins zur Spannungsversorgung fest auf dem Board geroutet. Jegliche Pins die auf dem Stromlaufplan mit einem Flag versehen sind, werden direkt vom Mikrocontroller auf eine Buchsenleiste geführt.
An Pin 16 XTALPPHY und Pin 17 XTALNPHY wird ein 25 MHz Oszillator für den Ethernettakt geschaltet. An Pin 48 OSC0 und Pin 49 OSC1 wird ein 16 MHz Oszillator als externer Takt geschaltet.
Pin 7 LDO liefert eine 1,3 V Versorgungsspannung für die meisten Logikfunktionen, den Prozessorkern und die meisten Peripherien. Sie muss an Pin 38 VDDC und Pin 39 VDDC geschaltet werden und durch einen mindestens 1 μF Kondensator gegen Ground entstört wer-

den. Pin 3 VDDA ist der Eingang für die Versorgungsspannung, der analog Funktionen des Mikrcontrollers, wie ADC oder analog Komparatoren. Er ist von der normalen Versorgungsspannung getrennt, um Rauschen auf den analogen Mikrcontrollerperipherien zu vermeiden. Um hochfrequente Störungen zu unterbinden, ist der analogen Versorgungsspannung noch ein Filter vorgeschaltet.

3.3.3 Spannungsversorgung (Abb. 3.2, Nr.2)

Die Hauptplatine benötigt zwei unterschiedliche Spannungslevel, um die verschiedenen IC-Bausteine auf der Platine zu versorgen. Zum einen werden 5 V für den Programmer und die USB-Buchse benötigt, zum anderen wird der Mikrocontroller mit 3,3 V versorgt. Um die beiden Spannungslevel zu realisieren, ist die Platine mit zwei Schaltreglern bestückt. Die 5 V werden von dem Spannungsregler LM2574N-5,0 erzeugt, die 3,3 V von dem Schaltregler LM2574N-3,3. Beide Schaltregler stammen von der Firma National Semiconductor. Sie regeln eine Eingangsspannung von 7 V - 40 V auf die vom IC bestimmte Ausgangsspannung und können dabei garantiert einen Strom von 0,5 A je Schaltregler treiben. Die äußere Beschaltung der Schaltregler ist nach Vorschlag des Datenblattes [6] Seite 1, Typical Application ausgeführt worden. Um die weiteren Modulplatinen oder weitere beliebige Bauteile zu versorgen, sind beide Spannungen und das Ground-Potential auf eine Buchsenleiste geführt. Sie können von dort aus per Steckbrücken abgegriffen werden. Weitere Informationen zur Funktion eines Schaltreglers sind der Literatur (Anhang [4] Seite 943) zu entnehmen.

3.3.4 Programmer (Abb. 3.2, Nr.3)

Der Programmer besteht aus den vier Komponenten: USB Mini B Buchse, EEPROM, FT2232D Chip und dem Multiplexer. Der Hauptchip des Programmers, hergestellt von der Firma Future Technology Devices International Ltd., ist der FT2232D Chip. Der FT2232D Chip ist ein Interface zwischen den seriellen Daten der Debugschnittstellen des Mikrocontrollers und USB. Der Chip beherbergt zwei USB zu seriell Schnittstellen (FT232B und FT245B). Beide Schnittstellen können separat konfiguriert und, unabhängig voneinander, als Port A und Port B eingesetzt werden. Die Konfiguration der Ports kann mit dem Tool FT Prog per USB in den extern angeschlossenen EEPROM geschrieben werden. Bei jedem Neustart des FT2232D Chips lädt dieser seine Konfigurationen nun aus dem EEPROM. Für den Programmer ist Port A als 245 FIFO für die Programmerschnittstellen JTAG und SWD definiert. Port A kann über den D2XX Direct Treiber mit dem PC kommunizieren. Um mit der JTAG Schnittstelle des Mikrocontrollers kommunizieren zu können, muss Port A des FT2232D Chips als Multi-Protocol Synchronous Serial Engine[4](MPSSE) konfiguriert werden. Diese

[4]Flexibles Protokoll zum Anbinden von JTAG oder I^2C an USB

Konfiguration wird durch den, von TI modifizierten, D2XX Treiber übernommen. Details zur Treiberprogrammierung sind der Literatur[7] zu entnehmen. Technisch wird die Konfiguration durch die Übertragung von Hex-Werten als Steuerbefehl an den FT2232D Chip realisiert. Da der Mikrocontroller sowohl über JTAG als auch über SWD gedebuggt werden kann, werden die beiden Signale gemultiplext. Die Steuersignale für den Multiplexer sind noch freie Port A Ausgänge des FT2232D Chips. Sie werden ebenfalls vom Code Composer Studio, je nach Wahl des Debuggmodus gesteuert. Der Multiplexer besteht aus zwei vierfach Bus-Buffern mit negativem Steuereingang und einem vierfach Bus-Buffer mit postivem Steuereingang. Die Bus-Buffer[5] schalten entweder das SWD oder Das JTAG Signal auf den Port A des FT2232D Chips durch.
Der Multiplexer ist vom In-Circuit Debug Interface (ICDI) Board von TI übernommen worden [8]. Der Grund hierfür ist, dass der Programmer Steuersignale für den JTAG/SWD Multiplexer von der Enwicklungsumgebung Code Composer Studio von TI erhält. Wäre der Aufbau des Multiplexers geändert worden, wäre die Programmierung und das Debuggen mit dem Code Composer Studio nicht möglich.
Port B des FT2232D Chips ist eine Schnittstelle mit dem UART des Mikrocontrollers. Port B ist als RS 232 UART Port im EEPROM konfiguriert. Durch einen Virtual COM Port Treiber kann per Terminal-Programm eine direkte serielle Verbindung vom PC zum Mikrocontroller hergestellt werden.
Das Experimentierboard ist so gestaltet, dass der Mikrocontroller auf dem Board, aber auch durch einen externen JTAG Programmer, programmiert und gedebuggt werden kann. Zum Anschluß dient ein 2x5 1,27 mm Wannenstecker. Ebensogut kann der Programmer auf dem Experimentierboard als Programmer für einen anderen Mikrocontroller verwendet werden, indem das Programmersignal über den Wannenstecker abgegriffen wird. Die Pinbelegung des JTAG Wannensteckers ist im Anhang A.1.5 zu finden.
Es ist unbedingt zu vermeiden, dass gleichzeitig mit dem internen Programmer und einem externen Programmer programmiert wird, da dies Beschädigungen der Hardware hervorrufen würde.

[5] Bus-Buffer: Logikschaltung die ausser den High- und Low-Pegeln noch einen dritten Pegel High Impedance bietet

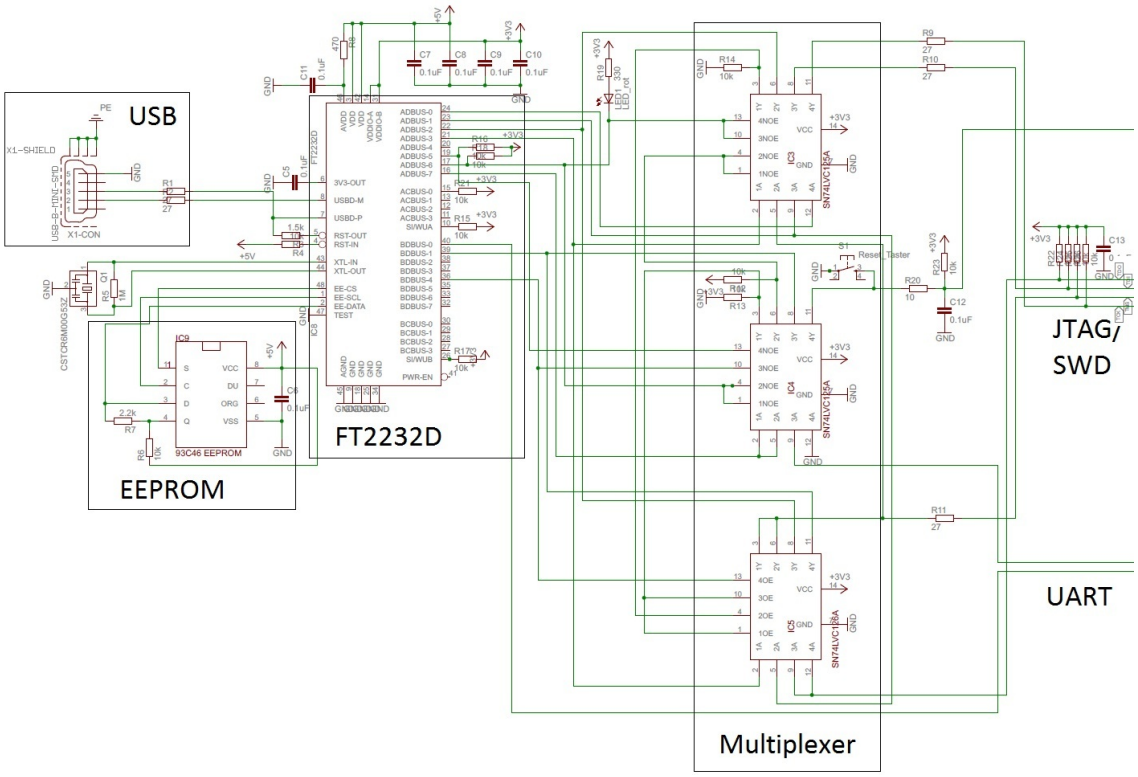


Abbildung 3.4: Stromlaufplan für den internen Programmer des Experimentierboard

Abbildung 3.4 zeigt noch einmal alle wichtigen Komponenten des Programmers. Die Schaltungen für die Spannungsversorgung, den EEPROM und den Resonator sind dem Datenblatt des FT2232D Chips zu entnehmen [9].

3.3.5 Schalter und LEDs (Abb. 3.2, Nr.4)

Das Modul Schalter und LEDs ist das erste Modul, dass kein Hauptmodul mehr ist. Es dient dazu, Ein- und Ausgänge des Mikrocontrollers kennenzulernen. Denkbar hierfür sind einfache Programme, in denen ein Schalter eine LED ein- und ausschalten kann. Die Schalter sind so verschaltet dass, wenn sie geöffnet sind, eine Spannung von 3,3 V (High-Pegel) anliegt. In geschlossenem Zustand liegen 0 V (Low-Pegel) über dem Schalter an. Die Ströme werden durch 10 kΩ Vorwiderstände begrenzt.
Die LEDs können durch Ausgänge des Mikrokotrollers ein- und ausgegeschaltet werden. Eine LED zieht, begrenzt durch den Vorwiderstand von 330 Ω, einen Strom von 7,9 mA. Um das Experimentierboard modular zu halten, sind die Schalter und die LED Ein- bzw. Ausgänge auf eine Buchsenleiste gelegt. Sollen sie verwendet werden, müssen sie über

Steckbrücken mit den, im Programm festgelegten, Ein- und Ausgängen des Mikrocontrollers verbunden werden.

3.3.6 Temperaturmessung, LM35 (Abb. 3.2, Nr.5)

Der LM35 ist ein Temperatursensor im TO-92 Gehäuse.

Abbildung 3.5: Bodenansicht LM35 Temperatursensor im TO-92 Gehäuse (Quelle [2])

Am +VS Pin muss eine Versorgungsspannung zwischen 4 V und 20 V angelegt werden. Bei einer Temperatur von 25°C garantiert der LM35 eine Genauigkeit von 0,5°C. Der gesamte Temperaturmessbereich erstreckt sich von -55°C bis +150°C. Ohne weitere Beschaltung ist jedoch nur eine Temperaturmessung von 2°C bis 150°C möglich. Um auch den negativen Temperaturbereich messen zu können, ist es nötig, die Beschaltung des LM35 laut Datenblatt ([2] Schaltbild 7, Seite 8) zu erweitern. In dieser Arbeit wird der Sensor in einer Schaltung für einen Temperaturbereich von 2°C bis 150°C betrieben. Der Sensor wandelt die Temperatur in eine lineare Ausgangsspannung um. Die Ausgangsspannung wird im Weiteren als Messspannung bezeichnet. Diese steht dem Nutzer zur Weiterverarbeitung bereit. Bei 0°C ist die Messspannung 0 V. Pro Grad Celsius steigt die Messspannung um 10 mV. Um diese Spannung auszuwerten, bedarf es einen ADC im Mikrocontroller. Dieser wandelt die analoge Spannung in einen 12 Bit breiten Digitalwert um. Dieser Digitalwert bestimmt letztendlich auch die Auflösung des Sensors.
Der verwendete Mikrocontroller stellt dem ADC eine interne Referenzspannung von 3 V zur Verfügung. Bei 12 Bit Auflösung des ADC stellt sich eine Bitbreite von 0,732 mV ein.

$$U_{LSB} = \frac{U_{ref}}{2^{nBit}} = \frac{3V}{2^{12}} = 0,000732V \tag{3.1}$$

Die Messspannung beträgt 10 mV/1°C, daraus errechnet sich eine Auflösung der Messung von 0,0732°C.

$$U_{Grad} = 10mV \tag{3.2}$$

$$^{\circ}C_{min} = \frac{1^{\circ}C \times U_{LSB}}{U_{Grad}} = \frac{1^{\circ}C \times 0,000732V}{0,01V} = 0,0732^{\circ}C \tag{3.3}$$

Die Auflösung der Messung wird jedoch durch die Stellenzahl der Ausgabe beschränkt.

3.3.7 Temperaturmessung, DS1820 (Abb. 3.2, Nr.6)

Der Temperatursensor DS1820 von der Firma Maxim ist wie der LM35 in einem TO-92 untergebracht. Er misst Temperaturen in einem Bereich von -55°C bis +125°C mit einer Genauigkeit von 0,5°C. Der DS1820 ist ein Temperatursensor der zur Familie der One-Wire-Bus-Komponenten zählt. One-Wire ist ein bidirektionaler, serieller Bus. Der Mikrocontroller ist der Bus-Master, wärend der DS1820 der Slave ist. Beim One-Wire-Bus kann die Spannungsversorgung des DS1820 über den Bus geschehen. Allerdings braucht jeder Busteilnehmer noch eine Rückleitung. Das One-Wire aus dem Namen One-Wire-Bus bezieht sich nur auf die Daten-/Spannungsleitung. Auf dem Experimentierboard hat der DS1820 eine separate Spannungsversorgung. Die Beschaltung hierfür ist im Datenblatt [3] zu entnehmen und in Abbildung 3.6 dargestellt.

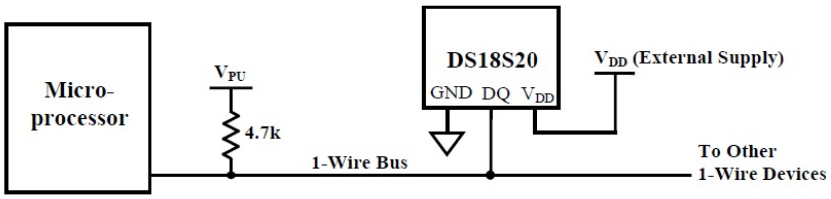


Abbildung 3.6: Schaltung DS1820 für externe Spannungsversorgung (Quelle [3])

Der DS1820 misst nur dann eine Temperatur, wenn er den Befehl vom Busmaster dazu erhält. Die gemessene Temperatur wird dann direkt digital in ein 16 Bit breites Temperaturregister geschrieben, welches anschließend vom Busmaster ausgelesen und bewertet werden kann. Bit 15-8 sind nur Vorzeichenbits und beinhalten lediglich die Information, ob die gemessene Temperatur positiv oder negativ gewesen ist. Der eigentliche Temperaturmesswert wird dann in den Bits 7-0 des Temperaturregisters gespeichert. Auf das Timing der Ansteuerung, sowie die Steuerbefehle für den DS1820, wird genauer im Kapitel 4.4.5 eingegangen.

3.3.8 USB (Abb. 3.2, Nr.7)

Da der Stellaris LM3S9D92 Mikrocontroller ein USB Interface bietet, ist dieses auf dem Experimentierboard berücksichtigt. Das Experimentierboard ist mit einer USB A Buchse ausgestattet. Hiermit ist es möglich, ein USB-Device an das Experimentierboard zu schließen. Der Mikrocontroller agiert als USB-Host. Um eine zu hohe Stromabnahme durch das Device zu verhindern, wurde der USB Spannung ein TPS 2051B IC von der Firma TI vorgeschaltet. Dieses IC ist ein strombegrenzender Leistungsschalter. Er limitiert den Maximalstrom für das USB-Device laut USB 2.0 Standard auf maximal 500 mA. Der Mikrocontroller bietet die Möglichkeiten, USB 2.0 Busverbindungen mit full-speed (12 Mega Bit per second (Mbps)) oder low-speed (1,5 Mbps) herzustellen. Auf diesem Experimentierboard ist der Mikrocontroller nur als USB-Host vorgesehen. Da USB nicht zu den Grundlagen für diesen Mikrocontroller zählt, ist kein spezielles Beispielprogramm hierzu erstellt worden. Einen einfachen Einstieg in das Thema USB kann der Benutzer durch Texas Instruments Stellaris Beispielprogramme erhalten.

3.3.9 Ethernet (Abb. 3.2, Nr.8)

Der Stellaris LM3S9D92 bietet ebenfalls eine Ethernet Schnittstelle. Um das Experimentierboard möglichst variabel und auch für fortgeschrittene Anwender attraktiv zu halten, wurde die Ethernetschnittstelle auf eine Ethernetbuchse geführt, um auch diesen Teil der Mikrocontrollerperipherie für den Anwender nutzbar zu machen. Der Mikrocontroller beherrscht sowohl den 10BASE-T[5] als auch den 100BASE-TX[6] Ethernet Standard. Auf dem Board ist eine Magjack SI-60024-F Ethernetbuchse verbaut. Diese Buchse hat den Vorteil, dass sie bereits einen internen 1:1 Übertrager bereitstellt. Dieser Übertrager trennt das Potential des Experimentierboards von dem Potential der angeschlossenen Hardware. Wie auch USB zählt Ethernet nicht zu den Einsteigerkenntnissen und wird dem Anwender deshalb nur auf dem Board bereitgestellt aber nicht weiter beschrieben oder in Beispielprogrammen erläutert.

[5] 10BASAE-T: Ethernet mit einer Datenrate von 10 Mbit/s
[6] 100BASAE-Tx: Ethernet mit einer Datenrate von 100 Mbit/s

3.4 Modulboards

3.4.1 7-Segment-Multiplex-Platine

Die 7-Segment-Anzeigen-Multiplex-Platine soll eine dreistellige Zahl darstellen können. Sie soll als Ausgabe für die Temperaturmessungen des LM35 und des DS1820 dienen. Der Sinn des Multiplexing wird klar, wenn man sich den Anschlussplan einer einzigen 7-Segment-Anzeige betrachtet.

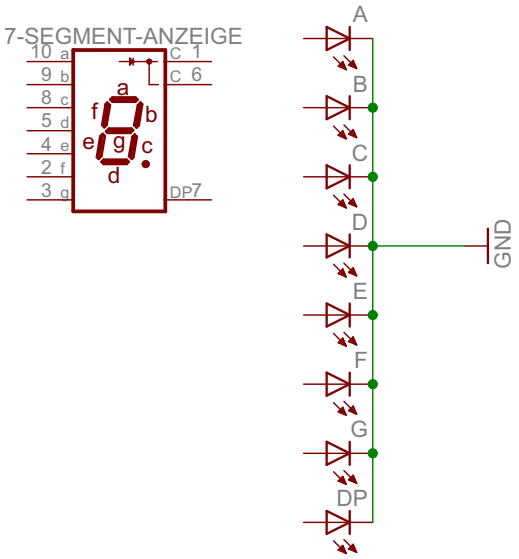


Abbildung 3.7: Aufbau 7-Segment-Anzeige

Abbildung 3.7 zeigt, dass jedes Fragment der 7-Segment-Anzeige durch eine LED beleuchtet wird. Da eine Dezimalzahl aus 7 Segmenten besteht, heißt die Anzeige 7-Segment-Anzeige. Ein achtes Segment kommt für das Komma hinzu. Der verwendete Typ 7-Segment-Anzeigen hat eine gemeinsame Kathode für die einzelnen Segment LEDs. Schließt man also eine 7-Segment-Anzeige normal an einen Mikrocontroller an, werden acht Ausgangspins benötigt, um die Anzeige zu betreiben. Sollen nun drei 7-Segment-Anzeigen angeschlossen werden, würde das schon 24 Ausgangspins des Mikrocontrollers verbrauchen. Da jedoch immer möglichst wenige Pins des Mikrocontrollers verbraucht werden sollen, wird hier die Methode des Multiplex angewendet. Genauer gesagt handelt es sich hierbei um ein Zeitmultiplex. Da das

menschliche Auge relativ träge ist, merkt es nicht, wenn die einzelnen Anzeigen nicht dauerhaft leuchten. Über die gemeinsame Kathode der Anzeigen lässt sich immer die gesamte Anzeige ausschalten. Prinzipiell können sich also mehrere 7-Segment-Anzeigen parallel geschaltet die Datenleitungen teilen. Ist das Datenregister passend beschrieben, wird die betreffende Anzeige einfach über ihre gemeinsame Kathode eingeschaltet. Dieses Prinzip zeigt das Schaltbild in Abbildung 3.8.

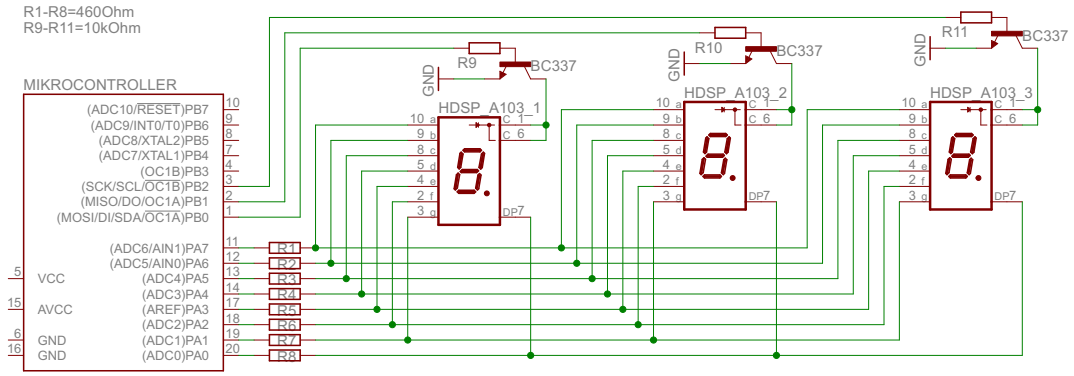


Abbildung 3.8: Schaltplan 7-Segment-Multiplex

Für die Anzahl der benötigten Mikrocontrollerpins ergibt sich folgende Gleichung.

$$Pins = n_{Anzeigen} + 8 \tag{3.4}$$

Für den Mikrocontroller bedeutet das Multiplexing, dass er in festen Zeitabständen immer wieder das Datenregister der 7-Segment-Anzeigen neu laden muss und die Anzeigen jeweils eine Stelle weiterschalten muss. Die Frage die Auftritt ist, wie lang darf das Zeitintervall sein, indem eine Anzeige leuchtet, damit die Anzeige noch wie eine einzige Zahl erscheint und nicht den Eindruck erweckt, dass sie flackert. Geht man hierbei von einem alten Fernseher mit 50 Hz Technik aus, ergibt sich, dass dieser alle 20 ms ein neues Bild geliefert hat. Für die 7-Segment-Anzeigen bedeutet das, dass ein gesamter Zyklus für alle drei Anzeigen 20 ms dauern darf.

$$t_{on} = \frac{1}{n_{Anzeigen} \times 50Hz} = 6,66ms \tag{3.5}$$

Da die Anzeigen nun nicht mehr dauerhaft leuchten, kann für den Betrachter der Eindruck entstehen, dass die Anzeige sehr dunkel ist. Aufgrund der kurzen Einschaltzeiten der LEDs besteht nun die Möglichkeit der Stromüberhöhung, um diesem Effekt entgegenzuwirken.

Hierfür gilt nachfolgende Formel.

$$I_{max} = n_{Anzeigen} \times I_{forward} \tag{3.6}$$

$I_{forward}$ ist aus dem Datenblatt der 7-Segmentanzeige zu entnehmen. Maximal sollte die Stromangabe für I_{peak} aus dem Datenblatt jedoch nicht überschritten werden. Kommt es bei der Stromüberhöhung zu Fehlern im Timing oder werden die Anzeigen aus irgendwelchen Gründen überhaupt nicht weitergeschaltet, hat dies eine Zerstörung der betroffenen 7-Segment-Anzeige zur Folge. Für diese Platine ist eine Stromüberhöhung nicht nötig gewesen, da die Anzeige auch so ausreichend hell erscheint. Der Strom pro Datenleitung ist auf dieser Platine $I_{forward}$=3,47 mA

$$I_{forward} = \frac{U - U_{forward}}{R} = \frac{3,3V - 1,7V}{460\Omega} = 3,47mA \tag{3.7}$$

Praktisch wäre eine Stromüberhöhung der einzelnen Segmente durch das Verändern ihres Vorwiderstandes möglich.
Auf der fertigen Platine, dargestellt in Abbildung 3.9, ist zu sehen, dass einfache Transistoren benutzt werden, um die Kathoden der einzelnen Anzeigen zu schalten.

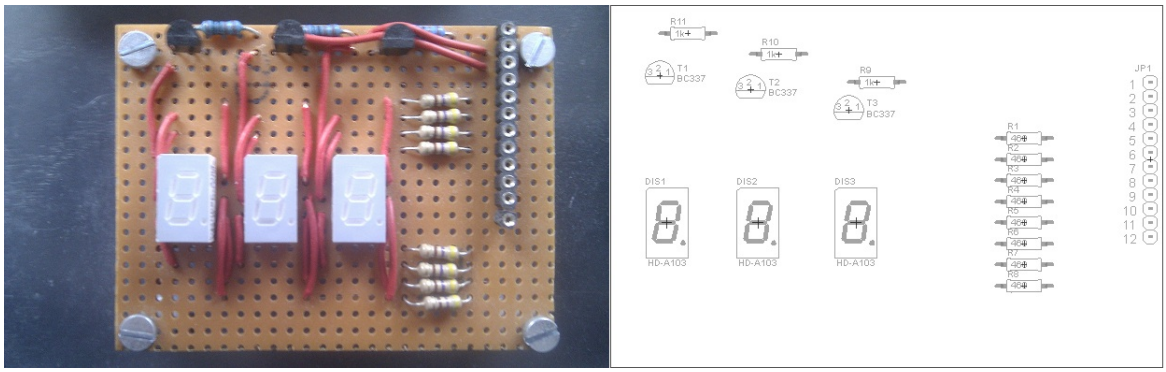


Abbildung 3.9: a.Draufsicht 7-Segment-Multiplex-Platine b.Bestückungsplan

Die Platine im Prototyp ist auf einer 69 mm x 51 mm Lochrasterplatine mit dem Rastermaß 2,54 mm gefertigt. Für Fertigungen in höherer Stückzahl steht eine fertige EAGLE-Board-Datei bereit. (Siehe Anhang A.2.1). Die Pinbelegung der Modulplatine steht im Anhang A.2.5.

3.4.2 LCD Platine

Die LCD Platine beherbergt das Modul 8 LCD. Das Display ist vom Typ Displaytech 64128k. Es hat eine Auflösung von 128x64 Pixeln und eine verstellbare Hintergrundfarbe. Drei Hin-

tergrund LEDs in den Farben Rot, Blau und Grün lassen sich durch vorgeschaltete Potentiometer in ihrer Leuchtkraft steuern und können somit jede beliebige Farbe erzeugen. Zur Kommunikation mit dem Mikrocontroller hat das Display einen Treiber Integrated Circuit (Treiber IC), der die Verbindung nach aussen herstellt, die eingehenden Informationen verarbeitet und dann das Display ansteuert. Hierbei handelt es sich um einen ST7665R von dem Hersteller Sitronix.
Das Treiber IC ST7565R kann über zwei Wege mit dem Mikrocontroller kommunizieren, zum einen über eine serielle Verbindung mit dem Namen 4-Line Serial Peripheral Interface (SPI) oder im 8 Bit parallel Modus. Auf der Modulplatine wurde die 8 Bit parallele Kommunikation mit 6800 Interface gewählt, da sie mikrocontrollerseitig nur Ein- und Ausgänge braucht und nicht den SPI. Dies ist für den Einsteiger einfacher zu programmieren und zu überblicken.
Das Treiber IC beinhaltet einen 65x132 Bit Random Access Memory (RAM). Das RAM steht in direkter Verbindung mit den Pixeln des Displays. Wird ein Bit im RAM geändert, so ändert sich auch das Pixel auf dem Display. Um nicht jedes einzelne Bit ansteuern zu müssen, wurde das RAM in eine Matrixstruktur von acht Bytes senkrecht unterteilt. Nutzt der Entwickler diese Form der Adressierung, kann beim Datentransfer immer byteweise gearbeitet werden. Soll nun ein Byte vom Mikrocontroller an das Display gesendet werden, muss im Vorfeld per Schreibbefehl die Startadresse für das Byte gesendet werden. Weiß der Displaytreiber an welche Stelle das Byte gespeichert werden soll, kann per Schreibdatenbefehl das tatsächliche Datenbyte in das RAM geschrieben werden.
Grundlegende Displayeinstellung, wie der Kontrast oder ein gesamtes Löschen des Displays, werden dem Displaytreiber mit dem Schreibbefehl-Kommando übermittelt. Auf die genaue Konfiguration und die Kommunikation mit dem Displaycontroller wird im Kapitel 4.4.6 eingegangen.

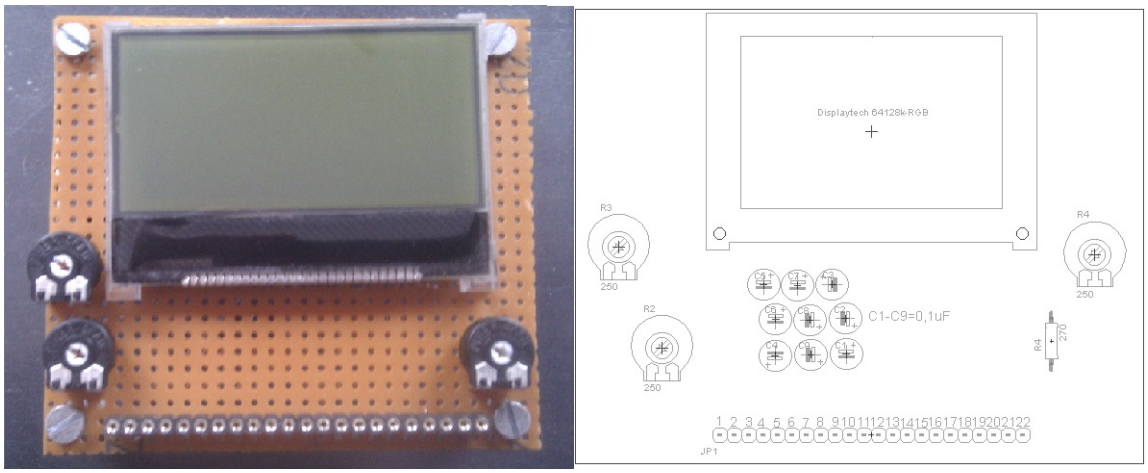


Abbildung 3.10: a. Draufsicht LCD Platine b. Bestückungsplan

Abbildung 3.10 zeigt die fertige LCD Platine. Sie ist auf einer Lochrasterplatine von 67 mm x 75 mm aufgebaut und hat ein Rastermaß von 2,54 mm. Da das LCD ein Rastermaß für seine Anschlusspins hat, muss die Lochrasterplatine in diesem Bereich nachgebohrt werden, um auch hier ein Rastermaß von 1,27 mm zu haben. Die Pinbelegung steht im Anhang A.3.4. Für Fertigungen in größerer Stückzahl steht auch für die LCD Platine eine fertige EAGLE-Board-Datei (Anhang A.3.1) zur Verfügung, um diese Platine direkt in Auftrag geben zu können.

3.4.3 Tastermatrix Platine

Die Tastermatrix Platine entspricht einer Art Zahlencodeschloß. Jede Ziffer von 0-9 wird durch einen Taster S0-S9 dargestellt. Wird der entsprechende Taster betätigt, soll die dazugehörige Ziffer vom Mikrocontroller detektiert werden. Die Auswertung der Tasterbetätigung soll vom Mikrocontroller durch eine ISR geschehen. Die Form der Matrixschaltung wird gewählt, um Ein- und Ausgänge am Mikrocontroller zu sparen und diese für andere Aufgaben offen zu halten.

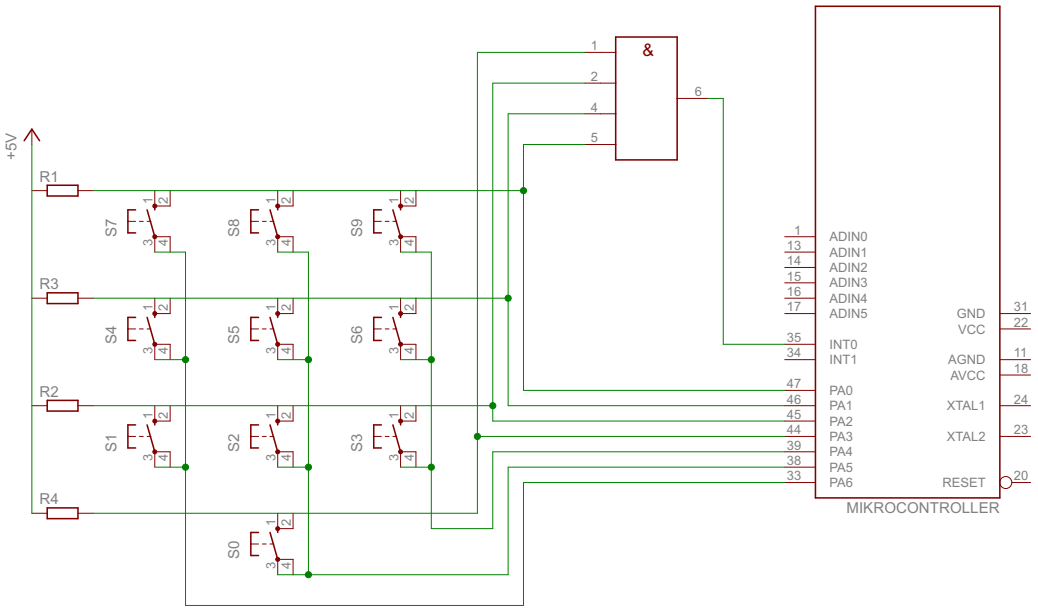


Abbildung 3.11: Schaltschema Tastermatrix

Der schematische Schaltplan der Tastermatrix in Abbildung 3.11 zeigt, dass jede Spalte der Matrix auf einen Ausgang des Mikrocontrollers geschaltet wird. Jede Zeile des Eingangs wird auf einen Eingang des Mikrocontrollers gelegt. Außerdem werden die Zeilen noch auf ein UND-Logikgatter geschaltet, um den Ausgang des Gatters als Auslösequelle für den Interrupt nutzen zu können.
Im Ruhezustand der Schaltung haben alle Spaltenausgänge des Mikrocontrollers Low-Pegel. Wird nun beispielsweise Taster S0 betätigt, wird die Zeile an Eingang PA3 auf den Low-Pegel gezogen. Außerdem schaltet der Ausgang des UND-Gatters ebenfalls auf Low-Pegel. Diese fallende Flanke wird vom Mikrocontroller erkannt und löst den Interrupt zur Auswertung des Ereignisses aus.
Wird die Schaltung wie in Abbildung 3.11 in Betrieb genommen, wird es zu Fehlern in der Auswertung des Mikrocontrollers kommen. Der Grund hierfür liegt in den Eigenschaften der Taster. Sie haben laut Datenblatt [10] eine Prellzeit von maximal 5 ms. Prellen bedeutet, dass der Taster nicht sauber schließt und während der Prellzeit, in schneller Folge, schaltet. Dieses Verhalten produziert ungewollt Flanken, die den Interrupt zur Auswertung des Tastendrucks ständig auslösen würden.
Messungen haben gezeigt, dass die verwendeten neuen Micro-Taster kaum prellen. Messungen an einem gebrauchten Taster weisen jedoch ein deutliches Prellverhalten auf. Aus diesem Grund wird die Platine mit einer Entprellschaltung, dargestellt in Abbildung 3.12, ausgestattet.

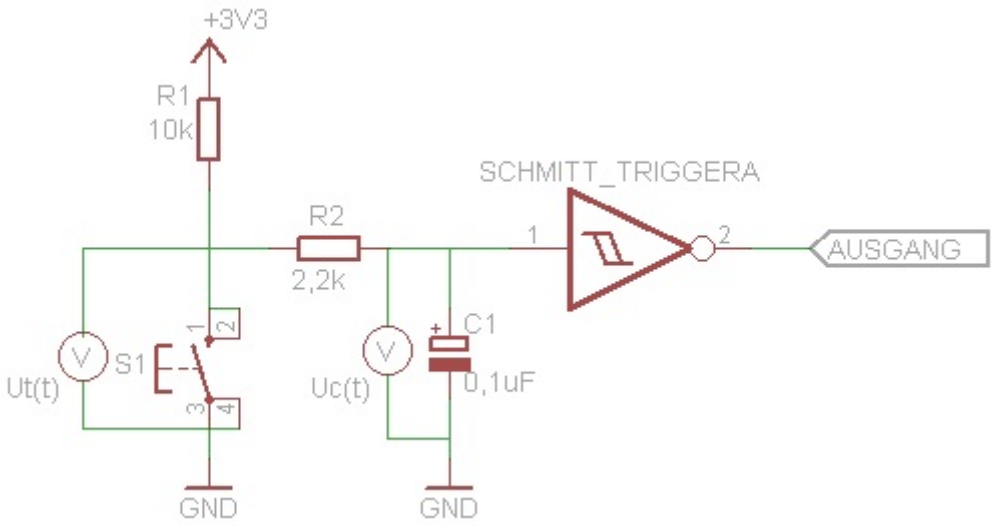


Abbildung 3.12: Schaltbild Taster entprellen

Im ersten Schritt wird das Springen des Pegels über dem Taster durch das parallelgeschaltete RC-Glied unterbunden. Im Zweiten Schritt werden die Pegel für low und high durch einen Schmitt Trigger in Bereiche aufgeteilt. Die Dimensionierung des RC-Gliedes hängt von der

Prellzeit des Tasters ab. Die genaue Funktion des Schmitt Triggers ist in Abbildung 3.13 dargestellt.

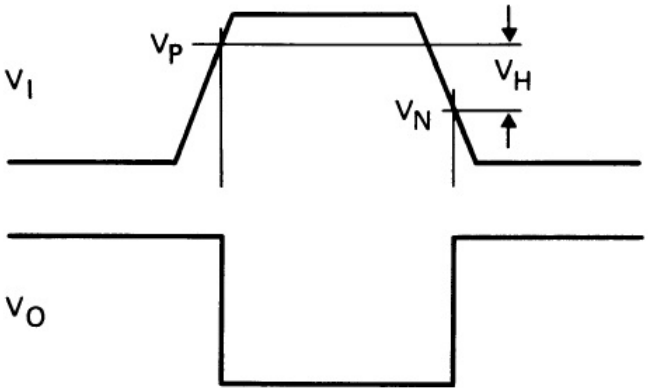


Abbildung 3.13: Funktion Schmitt Trigger

Aufgabe des Schmitt Triggers ist es, bei einem gewissen Pegel dauerhaft zu schalten bzw. abzuschalten. V_P gibt die Schaltschwelle für den High-Pegel an. V_N ist die Schaltschwelle für den Low-Pegel. Messungen am Schmitt Trigger haben gezeigt, dass V_P = 1,8 V und V_N = 1,3 V ist. Das dem Schmitt Trigger vorgeschaltete RC-Glied sorgt nun dafür, dass es keine Spannungssprünge am Eingang des Schmitt Triggers mehr gibt, welche ihn dazu veranlassen, für einen Schaltvorgang mehrere Flanken zu generieren. In dieser Schaltung ist ein invertierender Schmitt Trigger verbaut. Die nachfolgende Messung in Abbildung 3.14 zeigt die Spannungsverhältnisse innerhalb der Entprellschaltung. Die Messpunkte für u_t(t) und u_c(t) sind Abbildung 3.12 zu entnehmen.

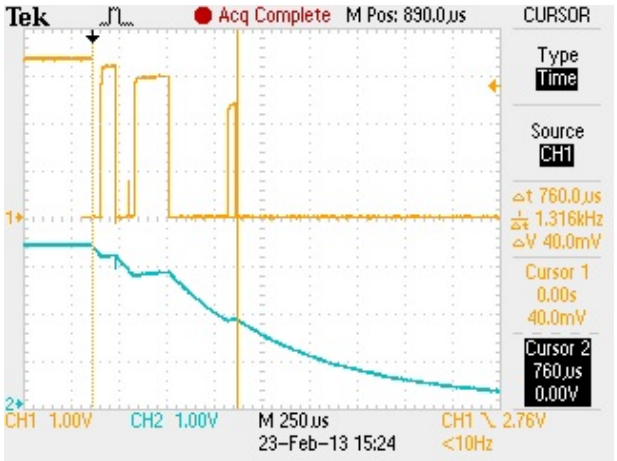


Abbildung 3.14: Gemessene Spannungen innerhalb der Entprellschaltung

Die gelbe Spannung aus Channel 1 ist $u_t(t)$, die Spannung direkt über dem Taster. Bei dieser Messung wurde eine Prellzeit von 760 μs ermittelt. Die blaue Spannung, gemessen auf Channel 2, zeigt die Spannung $u_c(t)$ über dem Kondensator und somit am Eingang des Schmitt Triggers. Es ist unbedingt zu beachten, dass das RC-Glied in dieser Messung noch nicht an die tatsächliche Prellzeit angepasst ist. Wichtig für $u_c(t)$ ist es, dass sie nur einmal die Schwellspannung V_N des Schmitt Triggers unterschreitet, um nur einen Interrupt zu generieren.
Anhand dieser Messung ist das RC-Glied dimensioniert worden. Die Spannung $u_c(t)$ lässt sich durch folgende Formel beschreiben.

$$u_c(t) = U \times (1 - e^{-\frac{t}{RC}}) \tag{3.8}$$

Der Grenzfall ist, wenn der Taster bei 760 μs erstmals prellt.
$mit: \tau = R \times C, U_c = 1,3V, U = 3,3V$

$$\tau = \frac{-t}{ln(1 - \frac{U_c}{U}} = \frac{-760\mu s}{ln(1 - \frac{1,3V}{3,3V}} = 9,8ms \tag{3.9}$$

Mit diesem Wert für τ wäre die gemessene Prellzeit sicher entprellt. Um Bauteile mit erhältlichen Werten verwenden zu können, ist τ=10 ms gewählt worden. Hierfür ergibt sich R=100kΩ und C=0,1 μF. Da immer nur ein Taster zur Zeit betätigt wird, reicht es aus eine Entprellschaltung pro Zeile vorzusehen.
Durch die Entprellschaltung, insbesondere durch das RC-Glied, tritt nun eine Verzögerungen für das Timing beim Auswerten im Mikrocontroller auf. Auf das Timing wird im Abschnitt 4.4.7 genau eingegangen.

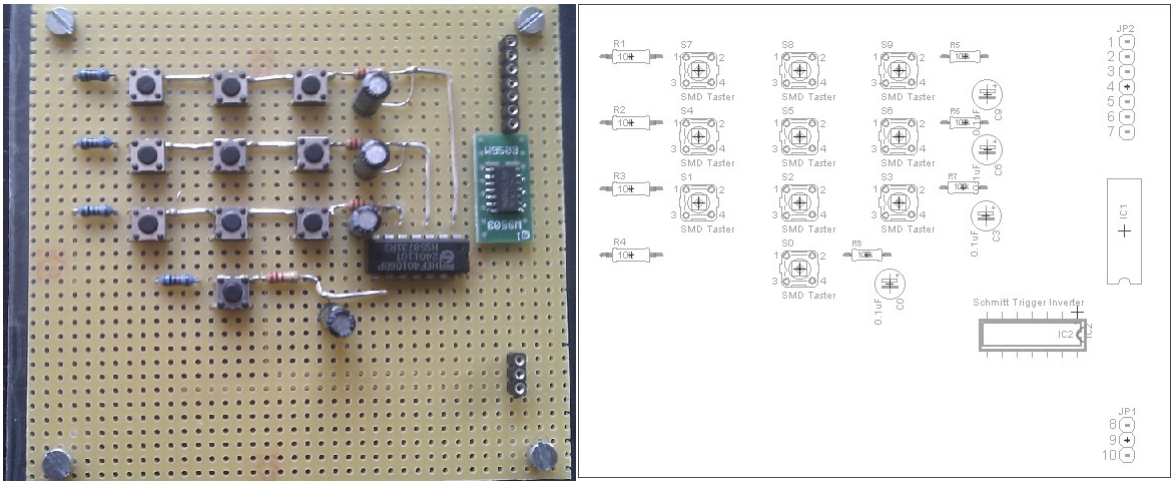


Abbildung 3.15: a. Draufsicht Tastermatrixplatine b. Bestückungsplan

Abbildung 3.15 zeigt die fertig erstellte Tastermatrixplatine und deren Bestückungsplan. Sie ist im Prototyp auf einer 90 mm x 100 mm Lochrasterplatine mit dem Rastermaß 2,54 mm aufgebaut. Im Rahmen der Arbeit ist auch eine fertige EAGLE-Board-Datei zur Erstellung einer Platine erstellt worden (Anhang A.4.1).

4 Inbetriebnahme

4.1 Übersicht

In diesem Kapitel soll das Experimentierboard in Betrieb genommen und seine Funktionen getestet werden. Bevor Programme auf den Mikrocontroller geladen und gestartet werden können, müssen zuvor die enstprechende Entwicklungsumgebung und ein Konfigurationsprogramm für den EEPROM des FT2232D Chips auf dem betreffenden Rechner installiert und konfiguriert werden. Diese Schritte werden im Unterkapitel 4.2 Software genau beschrieben.
Ist die nötige Software installiert, können erste Programme auf den Mikrocontroller aufgespielt werden. Diese Programme sind auf Registerebene und nicht mit den sonst, für diesen Mikrcontroller üblichen, Stellaris Funktionen geschrieben. Auf die Unterschiede, sowie die Vor- und Nachteile, dieser beiden Programmierebenen wird im Unterpunkt 4.3 Programmierebene eingegangen.
Die Programme aus Unterpunkt 4.4 Testprogramme sind nach Schwierigkeitsgrad gestaffelt. Sie sollen dem Anwender nach und nach die verschiedenen Peripherien des Mikrocontrollers nahebringen. Die Testprogramme zeigen viele für den Einsteiger wichtige Möglichkeiten, die Peripherien des Mikrocontrollers zu nutzen. Die einzelnen Funktionen des Mikrocontrollers sind in mit dem Experimentierboard durchführbare Versuche verpackt und lassen den Anwender immer tiefer in die Funktionen des LM3S9D92 einblicken.

4.2 Software

4.2.1 FT Prog 2.6.8

Um den Programmer auf dem Experimentierboard nutzen zu können, muss eine Basiskonifguration in das externe EEPROM des FT2232D Chips geschrieben werden. Der EEPROM lässt sich mit dem Programm FT Prog 2.6.8 beschreiben. Es wird zum freien Download auf der Homepage des Chipherstelllers FTDI Chip angeboten. Das Programm bietet die Möglichkeit, ein XML-Konfigurationstemplate (Anhang C.1) über eine grafische Oberfläche zu erstellen. Dieses Template enthält alle Parameter die der FT2232D Chip braucht, um seine

Ports zu konfigurieren.
Im ersten Schritt wird der verwendete FTDI Chip benannt. Ist der richtige Chip gewählt, zeigt das Programm die Einstellmöglichkeiten für den jeweiligen Chip an. Weiterhin können seine ID und die USB-Parameter eingestellt werden. Im letzten Schritt werden die Hardwareeinstellungen für die Ports gesetzt. Für das Experimentierboard ist Port A als 245 FIFO mit dem D2XX Direct Treiber konfiguriert. In dieser Konfiguration ist es möglich, dass der FT2232D Chip den MPPSE Modus zur Umsetzung von JTAG auf USB realisiert. Port B ist als RS 232 UART mit dem Virtual COM Port Treiber konfiguriert. Die Konfiguration von Port B dient als Schnittstelle zwischen UART und USB. Der Treiber stellt einen virtuellen COM Port am PC bereit. So kann per Terminalprogramm mit dem Mikrocontroller kommuniziert werden.
Das fertige Template kann nun auf den EEPROM des FT2232D Chips geladen werden. Eine detaillierte Schritt für Schritt Anleitung zum Programmieren des EEPROMS und zur Auswahl der Parameter sowie des Chips enthält die Bedienungsanleitung [11]
Ist das EEPROM des FT2232D Chips fertig programmiert, kann der Chip per USB mit dem Rechner vebunden werden. Damit die konfigurierten Funktionen des Chips vom Rechner unterstützt werden, müssen noch der D2XX Direct Treiber und der Virtual Com Port Treiber auf dem PC installiert werden. Beide Treiber wurden von TI an die Stellaris Evaluationsboards angepasst und sind auf der Homepage von Texas Instruments vefügbar. Für diese Arbeit sind die Stellaris FTDI Treiber in der Version 2.06.00 verwendet worden. Sind die Treiber installiert, muss das Experimentierboard folgendermaßen im Gerätemanager des PC erkannt werden.

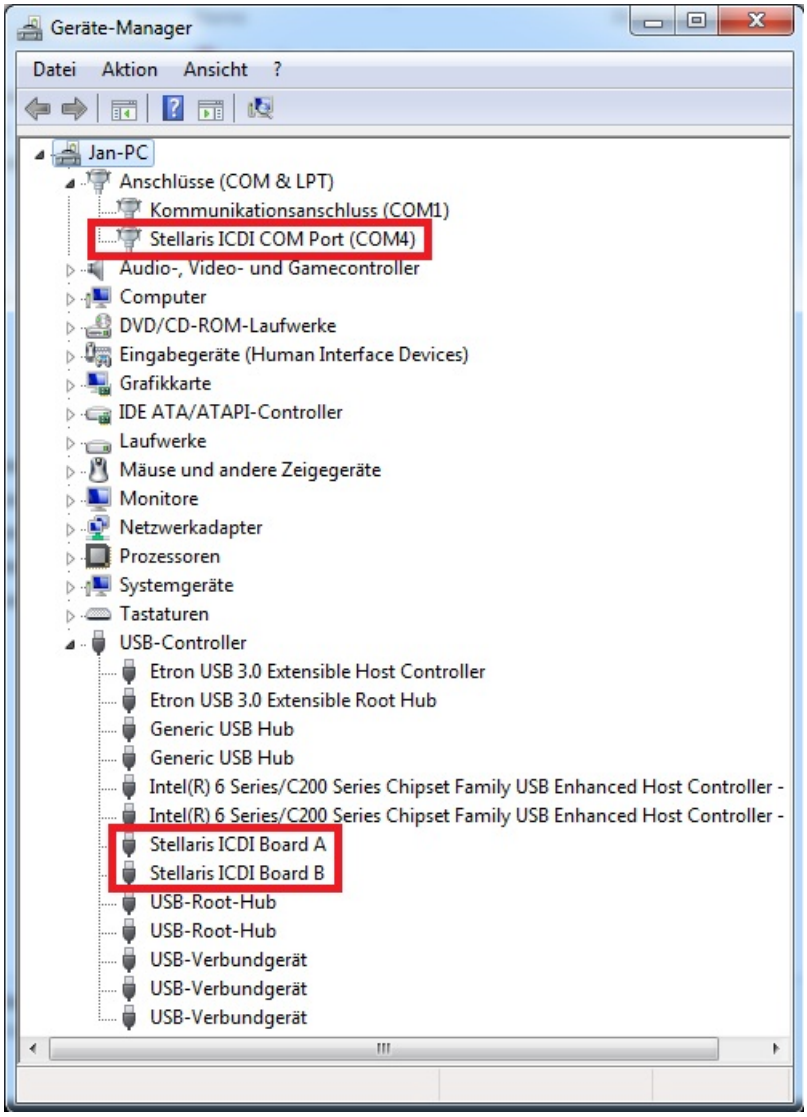


Abbildung 4.1: Gerätemanager mit angeschlossenem Experimentierboard

Abbildung 4.1 zeigt den Gerätemanager mit angeschlossenem Experimentierboard. Die rot markierte Hardware in den Kästen ist das Experimentierboard. Die Nummer des virtuellen COM Ports kann von Rechner zu Rechner unterschiedlich sein. Wird das Board nicht, wie auf der Abbildung, vom Rechner erkannt, wird der Programmer und die UART Verbindung über den FT2232D Chip zum PC nicht hergestellt.

4.2.2 Code Composer Studio v5

Das Code Composer Studio v5 ist die verwendete Entwicklungsumgebung für den Stellaris LM3S9D92 Mikrocontroller. Das Code Composer Studio v5 basiert auf der aktuellen Eclipse Version. Für diese Arbeit wurde das Code Composer Studio in der Version 5.2.1.00018 verwendet. Studenten können es frei, nach Anmeldung auf der TI Homepage, herunterladen. Innerhalb der Enwticklungsumgebung können für jedes Testprogramm einzelne Projekte angelegt werden.
Für das Experimentierboard müssen CCS-Projekte angelegt werden. Als Device ist ein ARM Mikrocontroller zu wählen. Der Mikrocontrollertyp ist LM3S9D92. Als Connection ist der Programmer Stellaris IN-Circuit Debug Interface zu wählen. Abbuildung 4.2 zeigt die Eingabemaske für ein neues CCS-Projekt wie beschrieben.

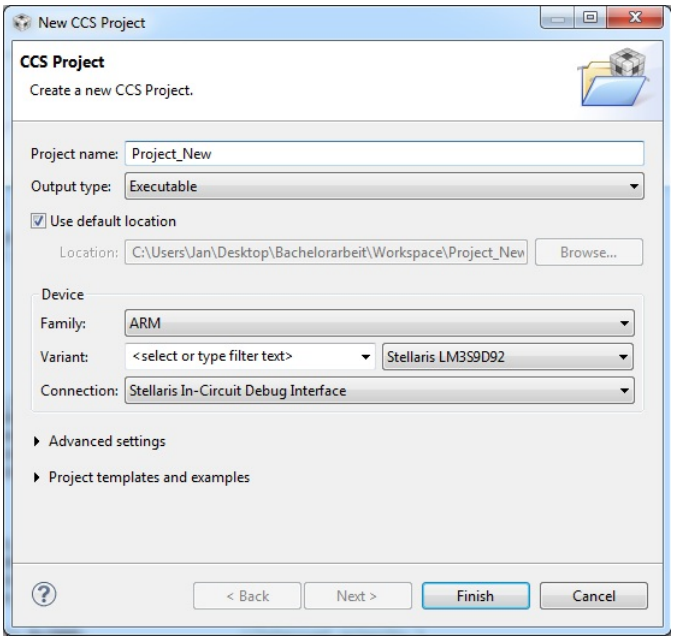


Abbildung 4.2: Eingabemaske CCS-Projekt im Code Composer Studio

Weiterhin verwaltet das Code Composer Studio v5 in das Projekt eingebundene Header- und C-Dateien. Soll ein Projekt in das Flash-ROM des Mikrocontrollers geladen werden, wird der Quellcode vorher vom Code Composer Studio v5 gelinkt und compiliert. Im Debugmodus lässt sich der Inhalt der einzelnen Register und verwendeten Variablen des Programms aus dem RAM auslesen. Der Quellcode kann entweder ganz, bis zu einem festgesetzten Haltepunkt oder nur schrittweise ausgeführt werden.
Damit die für diese Arbeiten entwickelten Testprogramme lauffähig sind, ist es notwendig

zusätzlich zum Code Composer Studio auch eine Version der Stellaris Ware von Texas Instruments in den Installationsordner des Code Composer Studios zu Installieren. Die Stellaris Ware enthält vorgefertigte C-Dateien für die Benutzung von Interrupts, auf die in den Tesprogrammen zurückgegriffen wird. Ist die Stellaris Ware installiert muss sie noch in das Code Composer Studio eingebunden werden. Der Download und das Einbinden der Stellaris Ware kann aus dem TI Resource Explorer unter dem Menüpunkt “examples“ erfolgen. Geöffnet wird der TI Resource Explorer unter dem Reiter Help -> Welcome to CSS im Code Composer Studio.

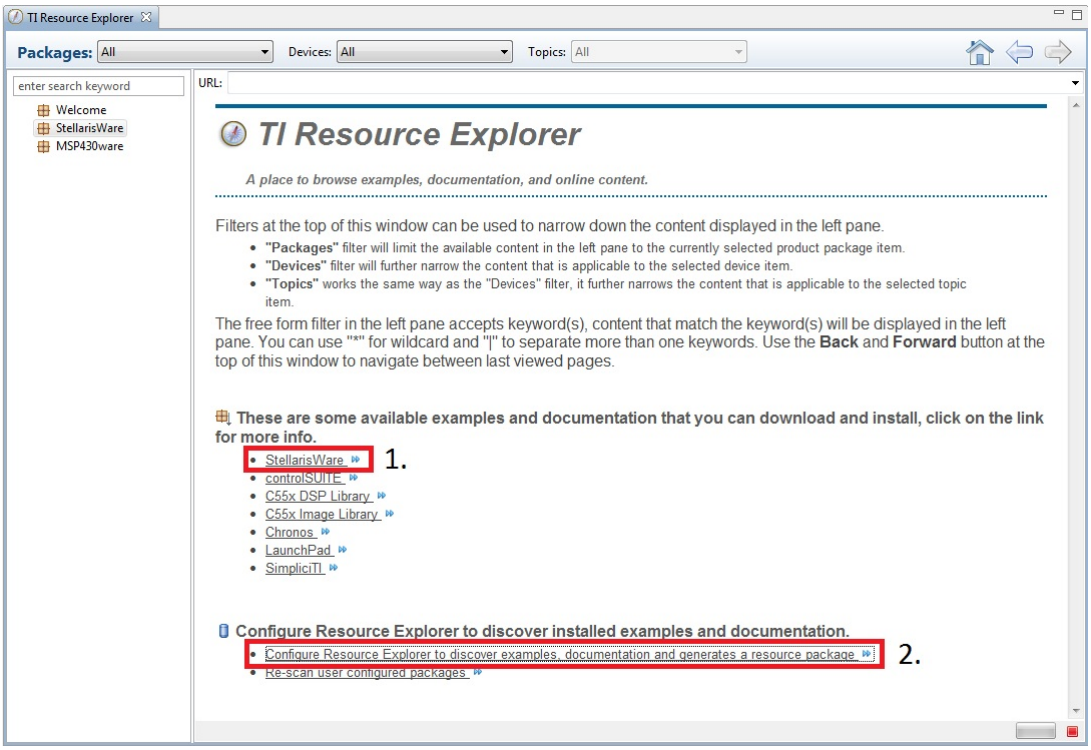


Abbildung 4.3: TI Resource Explorer Menüpunkt “examples“

In Abbildung 4.3 sind die zu benutzenden Links zu sehen. Link 1 aktiviert den Download der aktuellen Stellaris Ware. Ist diese in den Installationsordner des Code Composer Studios installiert, lässt sie sich durch Link 2 in die Entwicklungsumgebung einbinden.
Um die Funktionen der seriellen UART Verbindung zum Mikrocontroller zu nutzen, empfiehlt es sich ein Terminalprogramm in das Code Composer Studio zu installieren. Die Terminalverbindung kann im Edit- und im Debugmodus geöffnet und benutzt werden. Eine detaillierte Anleitung zur Installation des Terminalprogramms ist im Internet unter: "How to install the terminal plugin in CCSv5"[12] zu finden.

4.2.3 LM Flash Programmer

Der LM Flash Programmer ist ein Tool zum direkten Programmieren des Flash-Speichers des Stellaris LM3S9D92 Mikrocontrollers. Es wird als Download von TI zur Verfügung gestellt.
Mit diesem Programm lässt sich direkt auf den Flash-Speicher des Mikrocontrollers zugreifen. Funktionen des Programms sind: Programme als Binärdateien in den Programmspeicher schreiben, den Programmspeicher leeren oder den Programmspeicher zurückzusetzen. In diesem Projekt wird der LM Flash Programmer dazu verwendet, die Debug Ports wieder zu entsperren. Ein Nebeneffekt ist, dass dei diesem Vorgang ebenfalls der Programmspeicher gelöscht wird. Notwendig ist dieser Vorgang, wenn der Mikrocontroller durch fehlerhafte Programmierung, zum Beispiel der Clock Register, gesperrt ist und zurückgesetzt werden muss.

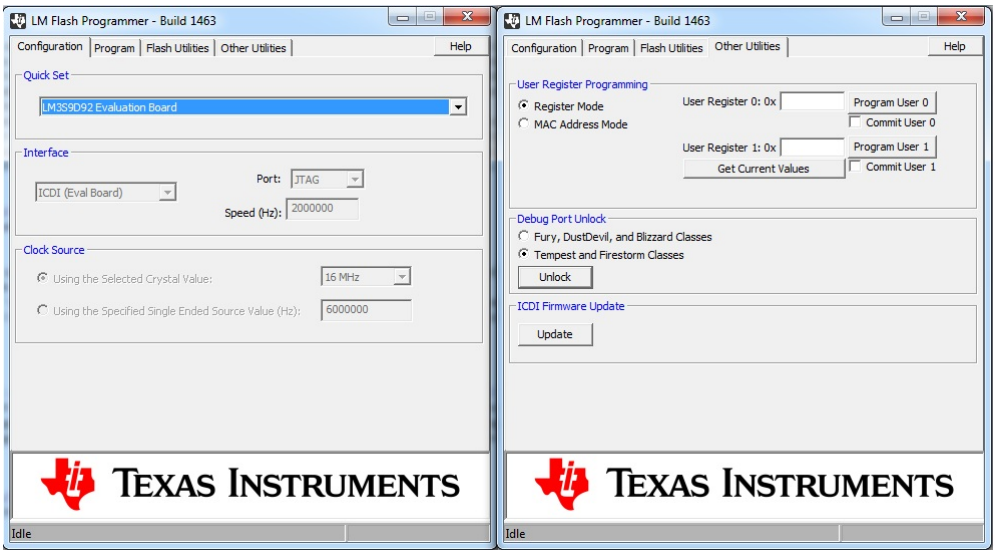


Abbildung 4.4: Konfiguration des LM Flash Programmer zum Entriegeln der Debugports

Abbildung 4.4 zeigt die notwendigen Konfigurationen des Programms, um die Debug-Ports des Mikrocontrollers zu entsperren. Mit dem Reiter “Configuration“ lässt sich die Verbindung zum Mikrocontroller konfigurieren. Unter LM3S9D92 Evaluations Board sind die Parameter für das Experimentierboard vorgespeichert. Unter dem Reiter “Other Utilities“ ist der Menüpunkt Debug Port Unlock zu finden. Der LM3S9D92 Mikrocontroller gehört zur Firestorm Klasse von TI ARM Cortex M3 Mikrocontrollern, deshalb ist der Button bei Tempest and Firestorm Classes zu setzen. Mit dem Unlock Button wird nun der Flash-Speicher des Mikrocontrollers zurückgesetzt und lässt sich wieder mit dem Code Composer Studio programmieren und debuggen.

4.3 Programmierebene

Um den Stellaris LM3S9D92 zu programmieren, hat der Benutzer grundsätzlich die Wahl zwischen zwei Ebenen der Programmierung. Einerseits gibt es die Registerebene, andererseits die Ebene der Stellaris Funktionen. Für jede Einstellung und Konfiguration am Mikrocontroller gibt es Register in denen bitweise Speicherinhalte geändert werden, um diese Einstellungen an der Hardware zu verändern. Da der LM3S9D92 ein 32-Bit-System ist, haben alle Register eine Breite von 32 Bit. Jedes einzelne Register hat seine feste Adresse im Programmspeicher damit ein eindeutiger Zugriff auf das Register erfolgen kann. Um den Umgang mit den Registern zu erleichtern, hat TI Makros für den Zugriff auf jedes Register angelegt.

```
#define GPIO_PORTA_DIR_R   (*((volatile unsigned long *)0x40004400))
```

Obiges Listing zeigt exemplarisch das Datenrichtungsregister von Port A. In der inneren Klammer wird zuerst ein Zeiger auf die Adresse des Datenregisters gerichtet. Eine direkte Zeigerzuweisung der Zahl 0x40004400 ist wegen inkompatibler Datentypen nicht möglich, deshalb wird der Cast-Operator angewandt. Damit das Makro keinen Optimierungen vom Compiler zum Opfer fällt, wird es durch das Attribut “volatile“ ergänzt. In der äußeren Klammer wird mit dem Dereferenzierungsoperator der direkte Zugriff auf den Inhalt der Adresse ermöglicht. Um beispielsweise einen Pin als digitalen Ausgang zu konfigurieren, sind zwei Registerzugriffe durchzuführen.

```
GPIO_PORTA_DEN_R |= 0x01;          //enable digital I/O pin PA0
GPIO_PORTA_DIR_R |= 0x01;          //PA0 output LED
```

Der erste Zugriff konfiguriert Port A Pin 0 als digitalen Pin. Der zweite Befehl setzt fest, dass Port A Pin 1 ein Ausgangspin ist.
Wird auf Registerebene programmiert, muss sich der Anwender im Vorfeld über die zur Peripherie gehörigen Register informieren, um beim Programmieren alle gewollten Konfigurationen für die Peripherie durchführen zu können.
Die Stellaris Funktionen sind eine Zusammenfassung mehrerer Registerzugriffe in einer Funktion. Sie sollen es dem Anwender erleichtern, die Peripherien zu konfigurieren. Die Stellaris Funktionen werden kostenlos von Texas Instruments angeboten und sind für alle Stellaris Mikrocontrollertypen gültig. Beim Stellaris LM3S9D92 sind sie auch im internen ROM des Mikrocontrollers gespeichert, damit keine extra Bibliotheken eingebunden werden müssen.
Im folgenden soll nocheinmal der Pin 0 von Port A als digitaler Ausgang konfiguriert werden. Diesmal mithilfe der Stellaris Funktionen. Um den Pin zu programmieren, muss der Anwender lediglich zwei definierte Konstanten und die passende Stellaris Funktion zu wissen. Port

A entspricht der Konstanten GPIO _PORTA_BASE, hinter der die Startadresse des Ports liegt:

```
#define GPIO_PORTA_BASE    0x40004000  // GPIO Port A
```

Pin 0 wird in folgender Konstanten gespeichert:

```
#define GPIO_PIN_0      0x00000001  // GPIO pin 0
```

Diese beiden Konstanten werden in zwei unterschiedlichen Headerfiles (hw_gpio.h und gpio.h) definiert. Sie dienen nun der Funktion zum Konfigurieren des Ausgangs als Übergabeparameter.

Listing 4.1: Stellaris Funktion GPIO Output

```
void
GPIOPinTypeGPIOOutput(unsigned long ulPort, unsigned char ucPins)
{
  //
  // Check the arguments.
  //
  ASSERT(GPIOBaseValid(ulPort));

  //
  // Set the pad(s) for standard push-pull operation.
  //
  GPIOPadConfigSet(ulPort, ucPins, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD);

  //
  // Make the pin(s) be outputs.
  //
  GPIODirModeSet(ulPort, ucPins, GPIO_DIR_MODE_OUT);
}
```

Listing 4.1 zeigt, dass die Portadresse und die gewünschte Pinnummer an die Funktion übergeben werden. Im ersten Schritt wird mit der ASSERT() Funktion geprüft, ob die Adresse des Ports gültig ist. Im Anschluss wird durch zwei weitere Stellaris Funktionen der Pin konfiguriert und als Ausgang geschaltet.

```
#define HWREG(x)  (*((volatile unsigned long *)(x)))
```

In den folgenden Funktionen wird das Makro HWREG(x) benutzt. Es wurde im Header hw_types.h definiert und ist einfach nur ein Pointer auf den Speicherinhalt der ihm übergebenen Adresse.

Listing 4.2: Stellaris Funktion Pad Configuration

```
void
GPIOPadConfigSet(unsigned long ulPort, unsigned char ucPins,
                 unsigned long ulStrength, unsigned long ulPinType)
```

```
{
  //
  // Check the arguments.
  //
  ASSERT(GPIOBaseValid(ulPort));
  ASSERT((ulStrength == GPIO_STRENGTH_2MA) ||
         (ulStrength == GPIO_STRENGTH_4MA) ||
         (ulStrength == GPIO_STRENGTH_8MA) ||
         (ulStrength == GPIO_STRENGTH_8MA_SC));
  ASSERT((ulPinType == GPIO_PIN_TYPE_STD) ||
         (ulPinType == GPIO_PIN_TYPE_STD_WPU) ||
         (ulPinType == GPIO_PIN_TYPE_STD_WPD) ||
         (ulPinType == GPIO_PIN_TYPE_OD) ||
         (ulPinType == GPIO_PIN_TYPE_OD_WPU) ||
         (ulPinType == GPIO_PIN_TYPE_OD_WPD) ||
         (ulPinType == GPIO_PIN_TYPE_ANALOG));

  //
  // Set the output drive strength.
  //
  HWREG(ulPort + GPIO_O_DR2R) = ((ulStrength & 1) ?
                                 (HWREG(ulPort + GPIO_O_DR2R) | ucPins) :
                                 (HWREG(ulPort + GPIO_O_DR2R) & ~(ucPins)));
  HWREG(ulPort + GPIO_O_DR4R) = ((ulStrength & 2) ?
                                 (HWREG(ulPort + GPIO_O_DR4R) | ucPins) :
                                 (HWREG(ulPort + GPIO_O_DR4R) & ~(ucPins)));
  HWREG(ulPort + GPIO_O_DR8R) = ((ulStrength & 4) ?
                                 (HWREG(ulPort + GPIO_O_DR8R) | ucPins) :
                                 (HWREG(ulPort + GPIO_O_DR8R) & ~(ucPins)));
  HWREG(ulPort + GPIO_O_SLR) = ((ulStrength & 8) ?
                                (HWREG(ulPort + GPIO_O_SLR) | ucPins) :
                                (HWREG(ulPort + GPIO_O_SLR) & ~(ucPins)));

  //
  // Set the pin type.
  //
  HWREG(ulPort + GPIO_O_ODR) = ((ulPinType & 1) ?
                                (HWREG(ulPort + GPIO_O_ODR) | ucPins) :
                                (HWREG(ulPort + GPIO_O_ODR) & ~(ucPins)));
  HWREG(ulPort + GPIO_O_PUR) = ((ulPinType & 2) ?
                                (HWREG(ulPort + GPIO_O_PUR) | ucPins) :
                                (HWREG(ulPort + GPIO_O_PUR) & ~(ucPins)));
  HWREG(ulPort + GPIO_O_PDR) = ((ulPinType & 4) ?
                                (HWREG(ulPort + GPIO_O_PDR) | ucPins) :
                                (HWREG(ulPort + GPIO_O_PDR) & ~(ucPins)));
  HWREG(ulPort + GPIO_O_DEN) = ((ulPinType & 8) ?
                                (HWREG(ulPort + GPIO_O_DEN) | ucPins) :
                                (HWREG(ulPort + GPIO_O_DEN) & ~(ucPins)));
```

Listing 4.2 zeigt die Funktion zur Konfiguration des Pins. Übergabeparameter sind hier die Startadresse des Ports, die Pinnummer und zwei weitere Konstanten.

```
#define GPIO_STRENGTH_2MA    0x00000001  // 2mA drive strength
#define GPIO_PIN_TYPE_STD    0x00000008  // Push-pull
```

Beide Konstanten werden in dem Header gpio.h definiert. Die übrigen Konstanten, die in der Funktion auftauchen, sind ebenfalls in dem Header gpio.h definiert worden und geben an,

um welchen Wert die Startadresse des Ports erhöht werden muss, um an das gewünschte Register zu gelangen. Zusammengefasst stellt diese Funktion eine Stromtreiberfähigkeit von 2mA und einen digitalen Ausgang für diesen Pin ein.
Die zweite Funktion die aus GPIOPinTypeGPIOOutput(unsigned long ulPort, unsigned char ucPins) aufgerufen wird, setzt die Richtung, also Eingang oder Ausgang oder eine alternative Funktion, des Pins fest. Übergabeparameter sind wieder die Startadresse des Ports sowie die Pinnummer. Außerdem wird der Modus des Pins übergeben. Dieser wird im Headerfile gpio.h definiert. In diesem Fall:

```
#define GPIO_DIR_MODE_OUT    0x00000001  // Pin is a GPIO output
```

Listing 4.3: Stellaris Funktion Pin Input/Output

```
void
GPIODirModeSet(unsigned long ulPort, unsigned char ucPins,
               unsigned long ulPinIO)
{
  //
  // Check the arguments.
  //
  ASSERT(GPIOBaseValid(ulPort));
  ASSERT((ulPinIO == GPIO_DIR_MODE_IN) || (ulPinIO == GPIO_DIR_MODE_OUT) ||
         (ulPinIO == GPIO_DIR_MODE_HW));
  //
  // Set the pin direction and mode.
  //
  HWREG(ulPort + GPIO_O_DIR) = ((ulPinIO & 1) ?
                                (HWREG(ulPort + GPIO_O_DIR) | ucPins) :
                                (HWREG(ulPort + GPIO_O_DIR) & ~(ucPins)));
  HWREG(ulPort + GPIO_O_AFSEL) = ((ulPinIO & 2) ?
                                  (HWREG(ulPort + GPIO_O_AFSEL) | ucPins) :
                                  (HWREG(ulPort + GPIO_O_AFSEL) &
                                  ~(ucPins)));
}
```

Listing 4.3 zeigt, dass zuerst geprüft wird, ob die Übergabeparameter korrekt waren. Im Anschluss wird das Datenrichtungsregister des Pins auf Ausgang gesetzt.
Zusammengefasst ergibt die Stellaris Variante zur Konifguration des Pins 0 von Port A, dass drei Funktionsaufrufe benötigt werden. Weiterhin sind Konstanten in drei unterschiedlichen Headerfiles definiert, die der Nutzer nachvollziehen muss, um den Vorgang zu verstehen. Auf Registerebene ist die Pinkonfiguration nach zwei Zeilen abgeschlossen. Für einen Neueinsteiger in die Programmierung des Stellaris LM3S9D92, der verstehen will, welche Register bei der Programmierung genau angesprochen werden müssen, um einen Pin zu konfigurieren, ist die Registerebene auf jeden Fall wesentlich transparenter und einfacher nachzuvollziehen. Aus diesem Grund werden Beispiele in dieser Arbeit immer auf Registerebene geschrieben.
Vorteil der Stellaris Funktionen ist, dass sie für mehrere Controllertypen gleich sind. Das

Programm ist also kompatibel zu anderen TI Mikrocontrollern die Stellaris Funktionen unterstützen.

4.4 Testprogramme

4.4.1 LED Ein/Aus

Das erste Programm hat die Aufagbe, eine LED mit einem Schalter ein- und auszuschalten. Hierfür müssen ein Pin als Eingang und ein Pin als Ausgang konfiguriert werden. Der Benutzer soll so einen leichten Einstig in die Programmierung des Stellaris LM3S9D92 erhalten und an die Konfiguration von Pins zu Ein- und Ausgängen geführt werden.

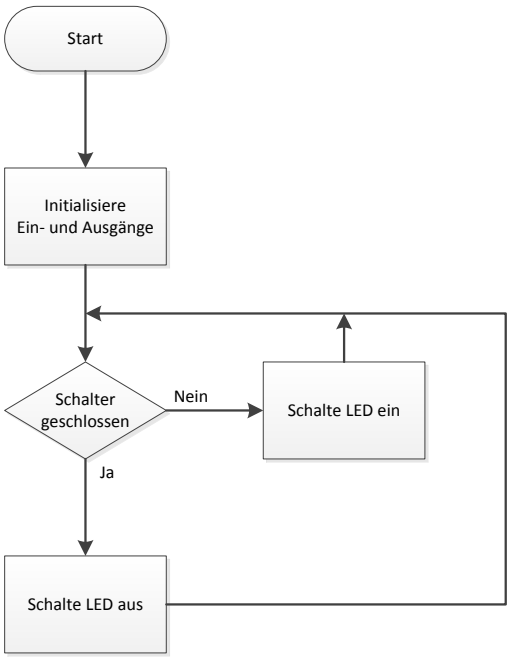


Abbildung 4.5: Flussdiagramm für das Testprogramm LED Ein/Aus

Das Flussdiagramm in Abbildung 4.5 zeigt, dass in dem Programm immer wieder der Eingang mit dem Schalter abgefragt wird. Hat der Eingang einen High-Pegel, wird die LED am Ausgang eingeschaltet. Hat der Eingang Low-Pegel, wird die LED wieder ausgeschaltet. Als Eingang dient Port B, Pin 4. Als Ausgang ist Port D, Pin 0 konfiguriert. Wichtig bei dem Stellaris LM3S9D92 ist, dass die Ports als Standardeinstellung aus Energiespargründen nicht

getaktet sind. Um überhaupt eine Funktion an den Ports zu erhalten, muss mit dem Run Mode Clock Gating Control Register 2 (RCGC2) der Takt an die Ports durchgeschaltet werden. Eine Übersicht der wichtigsten Register zum Konfigurieren der GPIOs enthält Tabelle 4.1. Die genaue Beschreibung der einzelnen Register ist dem Datenblatt [1] des Stellaris LM3S9D92 zu entnehmen.

Register	Beschreibung	Seite Datenblatt [1]
SYSCTL_RCGC2_R	Clock enable für die Ports A-J	283-285
GPIO_PORTn_DEN_R	Setzt Digitalmodus für die Pins	441
GPIO_PORTn_DIR_R	Datenrichtungsregister	421
GPIO_PORTn_DRxR_R	Stromtreiberfähigkeit des Ausgangs	432-434
GPIO_PORTn_DATA_R	Datenregister	420
GPIO_PORTn_AFSEL_R	Setzt alternative Funktionen des Ports	430
GPIO_PORTn_AMSEL_R	Setzt analog Modus für Pins	446

Tabelle 4.1: Übersicht der wichtigsten Register für GPIOs

Der gesamte Quellcode für das Programm ist im Anhang B.1 zu finden.

4.4.2 Interrupt

Das Testprogramm Interrupt soll die grundsätzliche Handhabung von Interrupts beim LM3S9D92 beleuchten. Hierzu wird ein Interrupt über eine externe Interruptquelle ausgelöst. In der Interrupt Service Routine (ISR) wird dann eine LED ein- und ausgeschaltet. Als Interruptquelle dient ein entprellter Taster am Pin PB2. Ein Interrupt ist, wie die Übersetzung des Namens vermuten lässt, eine Unterbrechung des laufenden Programms. Der NVIC kann bis zu 53 priorisierbare sowie einen nichtpriorisierbaren Interrupt (NMI) verwalten. Jeder Interrupt kann eine Priorität zwischen 0 und 7 erhalten. 0 stellt dabei die höchste und 7 die niedrigste Priorität dar. Die Interrupts werden durch Ereignisse aus den Peripheriefunktionen, wie beispielsweise fertige Zähler, fertige ADC-Umsetzungen, oder externe Ereignisse, wie eine steigende Flanke an einem bestimmeten Eingangsin, ausgelöst. Jedem Ereignis ist ein fester Interrupt zugeordnet. An der Adresse des Interrupts steht dann die Interrupt Service Routine, die den Programmcode für den spezifischen Interrupt enthält. Tritt ein Interrupt auf, arbeitet das NVIC folgende Arbeitsschritte ab.

1. Der NVIC vergleicht die Priorität des auftretenden Interrupts mit der des laufenden Programms bzw. des laufenden Interrupts.
    - Bei höherer Priorität wird der neue Interrupt akzeptiert und weiter ausgeführt.

- Bei gleicher oder niedrigerer Priorität erhält der Interrupt den Status Pending und er muss darauf warten, dass der laufende Interrupt mit höherer oder gleicher Priorität fertig ist.

2. Der laufende Maschinenbefehl wird beendet.
3. Programmzähler und Satusregister des laufenden Programms werden in den Stack kopiert.(Aktuelles Programm wird gesichert)
4. Die Priorität des Interrupts als Priorität für das laufende Programm übernehemen.
5. Programmzähler auf die Startadresse der Interrupt Service Routine (Speicherbereich in dem der Programmcode steht, der bei diesem Interrupt ausgeführt werden soll) setzen.
6. Programm aus der Interrupt Service Routine ausführen.
7. Alten Programmzähler und Statusregister aus dem Stack zurück kopieren.
8. Altes Programm weiterlaufen lassen.

Um eine ISR zu benutzen, muss der Mikrocontroller wissen, welche Startadresse die ISR hat. Um die Adressen für die insgesamt 53 möglichen Interrupts des LM3S9D92 festzulegen, liefert TI die Datei startup_css.c (Anhang B.3). In ihr wird jedem Interrupt eine Standard ISR mit Startadresse zugeordnet. Will man nun für einen bestimmten Interrupt eine eigene ISR verwenden, so muss diese zu Anfang von startup_css.c deklariert werden und später in der Vektortabelle für die Interrupts anstelle der Standard ISR für die besagte Interruptquelle eingetragen werden. Ein Beispiel für eine solche Datei ist im Anhang B.3 zu finden. Die wichtigsten Register zur Konfiguration von Interrupts mit externer Quelle sind in Tabelle 4.2 aufgelistet.

Register	Beschreibung	Seite Datenblatt [1]
GPIO_PORTn_IS_R	Stellt Pegel- oder Flankensensivität ein	442
GPIO_PORTn_IBE_R	Sensitiv auf beide Flanken	423
GPIO_PORTn_IEV_R	Sensitiv auf positive oder negative Flanke	424
GPIO_PORTn_IM_R	Interrupt wird an NVIC gesendet	425
GPIO_PORTn_ICR_R	Interrupt Status Flag	429
NVIC_ENn_R	Aktiviert Interrupt	128-129
NVIC_DIS_R	Deaktiviert Interrupt	130-131
NVIC_PRIn_R	Priorität des Interrupts	139

Tabelle 4.2: Übersicht der wichtigsten Register für Interrupts mit externer Quelle

4.4.3 7-Segment-Anzeigen-Multiplex

Beim 7-Segment-Multiplex werden drei 7-Segment-Anzeigen im Zeitmultiplex betrieben. Die Anzeigen sollen in späteren Versuchen der Messwertausgabe der Temperatursensoren dienen. Dieses Programm verwendet zusätzlich zu den bekannten Ein- und Ausgängen einen Timer sowie einen Timerinterrupt. Im Hauptprogramm werden lediglich die Ausgänge und der Timer initialisiert. Der Timer ist so konfiguriert, dass er periodisch arbeitet. Das bedeutet er zählt bis zu einem festgesetzten Wert und startet danach neu. Immer wenn der Grenzwert erreicht ist, löst der Timer einen Interrupt aus, der die Steuerung der 7-Segment-Anzeigen übernimmt. Die Register, um einen Timer samt Timerinterrupt in diesem Programm zu konfigurieren, sind in Tabelle 4.3 aufgezählt.

Register	Beschreibung	Seite Datenblatt [1]
SYSCTL_RCGC1_R	Clock Timer	271-274
TIMERn_CTL_R	Timer: start, stop	560-561
TIMERn_CFG_R	16 oder 32 Bit Zählregister	554
TIMERn_TxPR_R	Taktteiler bei 16 Bit Zählregiser	576-577
TIMERn_TxMR_R	Timer Modus Konfiguration	555-558
TIMERn_TxILR_R	Timer Vergleichsregister	572-573
TIMERn_IMR_R	Timer Interrupt aktivieren	562-563
TIMERn_ICR_R	Timerinterrupt Status Flag	570-571

Tabelle 4.3: Übersicht der benutzten Timerregister

Die verwendete Timerkonfiguration nutzt nur einen bestimmten Betriebsmodus des Timers. Die Timer des LM3S9D92 können noch auf viele weitere Modi konfiguriert werden. Alle möglichen Konfigurationen des Timers sind im Datenblatt [1] beschrieben. Im Flussdiagramm der ISR (Abbildung 4.6) wird dargestellt, wie der Multiplex softwareseitig erfolgt. Zuerst wird der Timer Interrupt Satus zurückgesetzt damit der Timerinterrupt wieder neu auftreten kann. Dann wird geprüft in welchem Durchlauf sich die ISR befindet. Insgesamt gibt es drei Durchläufe. Je nach Durchlauf wird nun die erste, die zweite oder die dritte 7-Segment-Anzeige eingeschaltet. Nachdem die zum Durchlauf gehörige Anzeige eingeschaltet ist, werden die Datenleitungen mit dem entsprechenden Wert beschaltet und die 7-Segment-Anzeige zeigt die entsprechende Zahl an.
Die Beschaltung der Ausgänge für die Datenleitungen ist in eine Extrafunktion ausgelagert worden, um die Übersichtlichkeit des Programms zu wahren. In ihr werden entsprechend der Beschaltung die richtigen Ausgänge gesetzt. Übergabeparameter der Funktion ist die Zahl, die ausgegeben werden soll. Der Quellcode des gesamten Programms ist im Anhang B.4 zu finden.

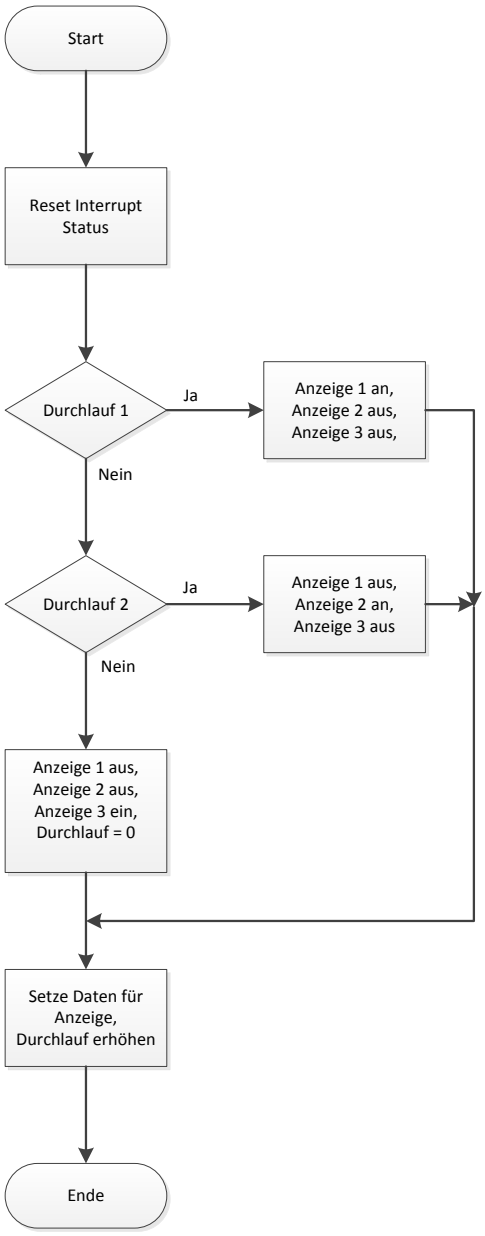


Abbildung 4.6: Flussdiagramm für Interrupt Service Routine des Timerinterrupts

4.4.4 Temperaturmessung Sensor LM35

Die Temperaturmessung mit dem Temperatursensor LM35 ist eine Erweiterung des Programms 7-Segment-Anzeigen-Multiplex. Die vom Sensor gemessene Temperatur wird im Mikrocontroller analog-digital gewandelt und weiterverarbeitet. Das 7-Segment-Anzeigen-Multiplex-Programm muss nur noch um einen ADC erweitert werden. Der LM3S9D92 bietet 16 ADC-Eingänge mit einer Auflösung von 12 Bit. In diesem Programm ist der ADC auf Port E, Pin 7 programmiert.
Um die Konfiguration des verwendeten ADC zu verstehen, ist es notwendig, den gesamten Aufbau der ADC-Einheit des Mikrocontrollers zu betrachten. Der ADC besteht aus 2 ADC-Modulen ADC 0 und ADC 1. Jeder der beiden ADCs hat 4 sogenannte Sequencer zur Verfügung. Jeder Sequencer kann frei programmiert werden und beliebige Samples der insgesamt 16 ADC-Eingänge machen. Zu jedem Sequencer gehört ein First In First Out (FIFO) Speicher, in dem die Ergebnisse der Umsetzungen gespeichert werden. Sequencer 0 macht in einer Sequenz acht Samples. Sequencer 1 und 2 machen in einer Sequenz jeweils vier Samples. Sequencer 3 macht nur ein Sample pro Sequenz. Welcher Eingang für welches Sample benutzt wird, kann im Sequencer-Konfigurations-Register frei eingestellt werden. In diesem Programm wurde Sequencer 3 von ADC 0 benutzt, da pro Sequenz nur ein Sample von Eingangspin PE 7 benötigt wird.
Die Tabelle 4.4 zeigt noch einmal die benutzten ADC-Register für dieses Programm.

Register	Beschreibung	Seite Datenblatt [1]
SYSCTL_RCGC0_R	Clock ADC	263-265
ADCn_ACTSS_R	Sequencer: Start, Stop	632
ADCn_EMUX_R	Sequenz Trigger	642-646
ADCn_CTL_R	Referenzspannung und Auflösung	657
ADCn_SSMUX3_R	Wählt Eingang für Umsetzung	676
ADCn_SSCTL3_R	Sequencer 3: Interrupts, Sequenzende	677
ADCn_PSSI_R	Start Sequenz aus Software	652-653
ADCn_SSFIFO3_R	Ergebnisse der Umsetzung	663

Tabelle 4.4: Übersicht der benutzten ADC-Register

Das Flussdiagramm in Abbildung 4.7 zeigt die Abläufe, die in die ISR des 7-Segment-Anzeigen-Multiplex Programm aufgenommen worden sind, um die Messspannung des Sensors auszuwerten und auf der 7-Segment-Anzeige auszugeben. Damit die Temperatur nicht mit jeder Messung springt und die Anzeige ruhig betrachtet werden kann, werden 100 Messwerte analog-digital gewandelt und vor der Ausgabe gemittelt. Die einzelnen Stellen der gemessenen Temperatur werden durch Modulodivision ermittelt und auf Variablen gespei-

chert, die dann auf den 7-Segment-Anzeigen ausgegeben werden. Der gesamte Quellcode für dieses Testprogramm ist im Anhang B.5 einzusehen.

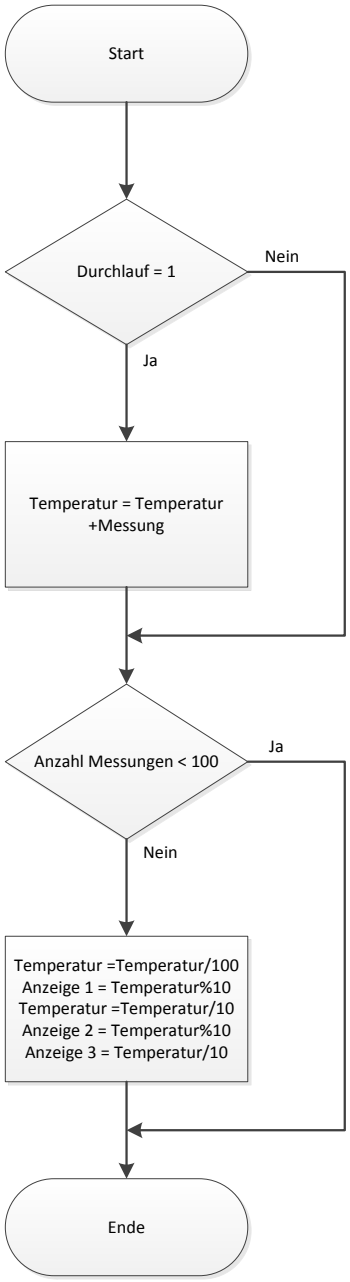


Abbildung 4.7: Flussdiagramm für die Erweiterungen der Timerinterrupts ISR aus 4.6

4.4.5 Temperaturmessung Sensor DS1820

Bei dem Temperatursensor DS1820 von der Firma Maxim Integrated handelt es sich um einen komplett digitalen Temperatursensor. Die Ausgabe der gemessenen Temperatur erfolgt wie beim LM35 Temperatursensor wieder über die 7-Segment-Anzeigen. Demzufolge ist auch dieses Testprogramm eine Erweiterung des Testprogramms 7-Segment-Anzeigen-Multiplex. Die Messung der Temperatur erfolgt wie beim LM35 Sensor wieder in der Timer ISR (Abbildung 4.6 Testprogramm 7-Segment-Anzeigen-Multiplex).
Allerdings unterscheidet sich der Sensor DS1820 gänzlich vom Sensor LM35. Beim Temperatursensor DS1820 handelt es sich um eine One-Wire-Buskomponente. Theoretisch könnte ein gesamtes Bussystem mit mehreren Temperatursensoren oder anderen Busteilnehmern wie A/D-Wandler oder Schlüsselschalter betrieben werden. Beim One-Wire-Bus handelt es sich um ein Single Master, Multi Slave Bussystem. Jede One-Wire-Komponente hat weltweit eine einzigartige Identifikationsnummer (ID). Alle Teilnehmer eines One-Wire-Bussystems sind über eine einzige Datenleitung DQ miteinander verbunden. Will der Master mit einem Slave kommunizieren, kann der Slave über die ID ausgewählt werden. In diesem Programm ist der Mikrocontroller der Busmaster, der DS1820 ist der Slave. Um die Temperatur zu messen und aus dem Sensor auszulesen ergibt sich der prinzipielle Ablauf zur Durchführung einer Messung, dargestellt in Abbildung 4.8.

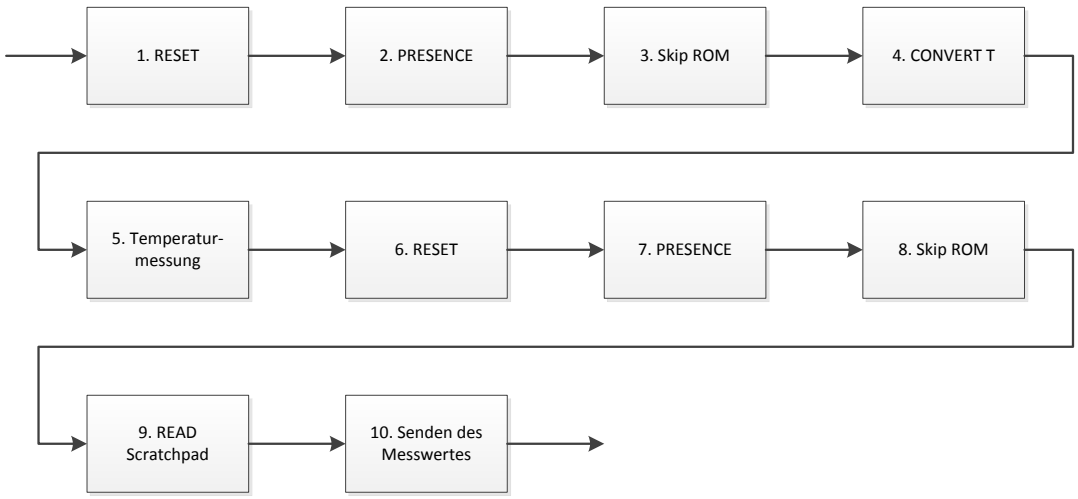


Abbildung 4.8: Ablaufsequenz Temperaturmessung DS1820

Jeder Befehl an den Temperatursensor beginnt mit einer Initialisierungssequenz. Hierzu schickt der Busmaster ein RESET-Pulse (Bus wird für mindestens 480μs auf Low-Pegel gesetzt). Der Slave antwortet mit einem PRESENCE-Pulse (Der Slave setzt den Bus für

60-240μs auf Low-Pegel). Das Timing für die Initialisierungssequenz ist in Abbildung 4.9 zu sehen.

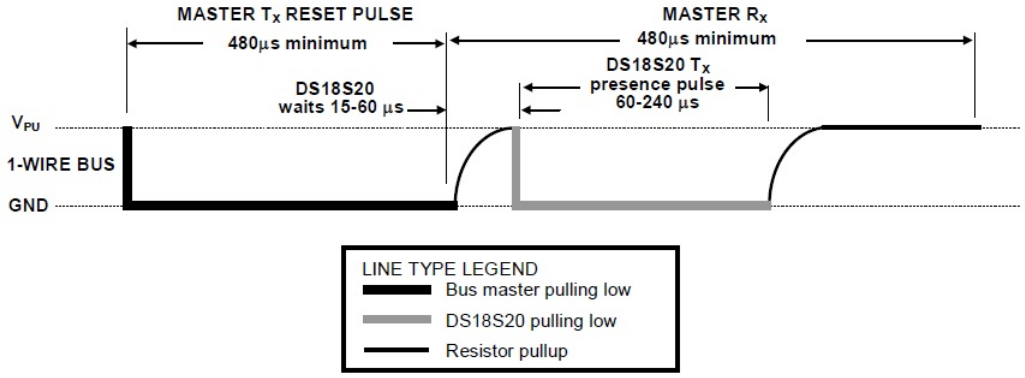


Abbildung 4.9: Timing Initialisierungssequenz DS1820 (Quelle [3])

Im Anschluss sendet der Master ein ROM-Command an den Sensor. Insgesamt gibt es fünf 8 Bit breite ROM-Commands zur Adressierung der Slave-Teilnehmer. Da in diesem Programm nur ein Busteilnehmer vorhanden ist, wird das Skip ROM Kommando geschickt. Es bewirkt, dass alle Slaves am Bus gleichzeitig angesprochen werden. Nach der Adressierung folgt ein Function-Command vom Master. Das Function-Command ist wiederum 8 Bit breit und heißt CONVERT T. Es veranlasst die Temperaturmessung im Sensor. Die Messung im Sensor dauert maximal 750 ms. Nach dieser Wartezeit leitet der Busmaster eine erneute Initialisierungssequenz ein und adressiert anschließend den Temperatursensor. Die gemessenen Daten werden mit dem Function-Command Read Scratchpad ausgelesen. Insgesamt sendet der Sensor 9 Bytes an den Master. Der Messwert der Temperatur verbirgt sich dabei in Byte 0. Das Vorzeichen der Temperatur wird in Byte 1 übermittelt. In diesem Programm wird nur Byte 0 mit dem Temperaturwert ausgelesen und ausgewertet. Anschließend wird die Temperatur auf den 7-Segment-Anzeigen ausgegeben.
Die jeweils 8 Bit breiten ROM- oder Function-Commands werden vom LSB zum MSB Bit für Bit gesendet. Dies wird durch sogenannte Master-Write-Time-Slots gemacht. Das genaue Timing der Write-Time-Slots zeigt Abbbildung 4.10. Will der Master eine 0 senden, zieht er den Bus 60-120 μs auf Low-Pegel. Will er eine 1 senden, zieht er den Bus für etwa 6 μs auf Low-Pegel. Bei beiden Write-Slots macht der Sensor nach 15-60 μs ein Sample und erkennt somit ob eine 1 oder eine 0 gesendet wurde.

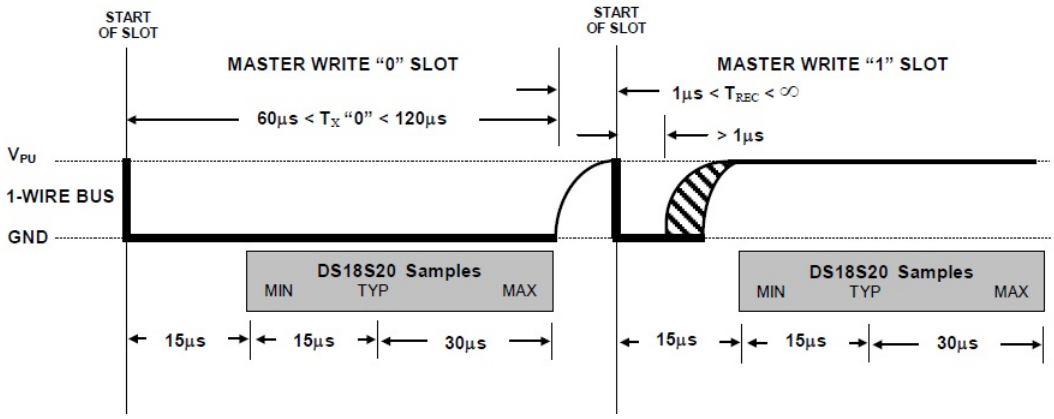


Abbildung 4.10: Timing Write Time Slots DS1820 (Quelle [3])

Um Daten zu empfangen, muss der Master Read-Time-Slots initialisieren. Hierzu zieht er den Bus für 1 μs auf Low-Pegel. Sendet der Sensor eine 0, hält er den Bus weiterhin auf dem Low-Pegel. Wird eine 1 gesendet, setzt der Sensor den Bus auf high. 15μs nach Einleitung des Read-Time-Slots muss der Master ein Sample ziehen und somit die gesendete 1 oder 0 detektieren. Ein gesamter Read-Time- Slot dauert 60 μs. Das Timing für den Read-Time-Slot wird in Abbildung 4.11 dargestellt.

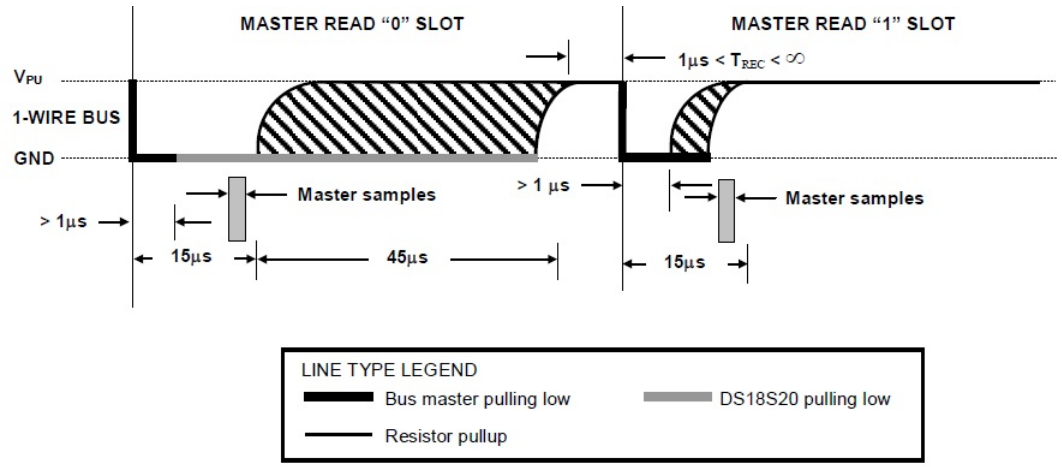


Abbildung 4.11: Timing Read Time Slots DS1820 (Quelle [3])

Um das Timing im Programm zu realisieren, wurde ein One-Shot Timer initialisiert, der eine Durchlaufzeit von 12,5ns bis 819μs hat. Um diesen Timer zu realisieren, wurde die System Clock auf 80MHz erhöht. Hierfür muss der Taktbaum Abbildung 2.4 des Mikrocontrollers neu konfiguriert werden.

Register	Beschreibung	Seite Datenblatt [1]
SYSCTL_RCC0_R	Konfigurtion System Clock	220-222
SYSCTL_RCC2_R	Konfigurtion System Clock	228-230
SYSCTL_MISC_R	PLL Mask Interrupt Status	216-217

Tabelle 4.5: Übersicht der benutzten SYSCTL-Register

Tabelle 4.5 zeigt die dafür benötigten Register. Die genaue Bedeutung der einzelnen Bits im Register ist dem Datenblatt [1] des LM3S9D92 zu entnehmen. In einer “timerwait(int time(μs))“ Funktion kann nun die Wartezeit in μs übergeben werden und entsprechend in das Timer Intervall Load Register geladen werden. So können die Wartezeiten aus den vorgegebenen Timings des Sensors einfach umgesetzt werden.
Der gesamte Quellcode des Programms ist im Anhang B.6.

4.4.6 Liquid Crystal Display

In diesem Testprogramm wird ein LC-Display vom Typ Displaytech 128x64k angesteuert. Innerhalb des Displays ist ein Displaycontroller ST7565 verbaut. Vor dem Start des Displays muss der Displaycontroller initialisiert werden. In diesem Programm ist der Displaycontroller auf 6800 Parallel Mode eingerichtet. Dies geschieht durch die entsprechende Beschaltung der Pins P/S=high und C86=high (Anhang A.3.5). Damit ergibt sich für die Steuereingänge die dargestellte Funktion in Tabelle 4.6.

E (/RD)	A0	W/R (/WR)	Funktion
1	1	1	Read Display Data
1	1	0	Write Display Data
1	0	1	Status Read
1	0	0	Write Control Data (Command Mode)
0	X	X	Disable Communication

Tabelle 4.6: Wahrheitstabelle Steuereingänge Displaycontroller ST7565

Der Displaycontroller hat 2 Betriebsmodi, zum einen den Command Mode (A0=0), in dem er Kommandos für seine Konfiguration oder die Adressierung der nachfolgenden Daten entgegenimmt, zum anderen den Data Mode (A0=1), in dem der Displaycontroller Daten in das Display-Data-RAM schreibt oder ausliest.
Die weitere Displayinitialisierung, wie Anordnug des Adressraumes, Kontrast und Spannungsversorgung, erfolgt durch das Senden von Steuerbefehlen an den Displaycontroller. Dazu muss der Displaycontroller in den Command Mode versetzt werden. Im Command

Mode können 32 Bit breite Steuerbefehle an den Displaycontroller gesendet werden, um das Display zu konfigurieren oder um die Adressierung für die nächsten Nutzdaten im Data Mode vorzunehmen.

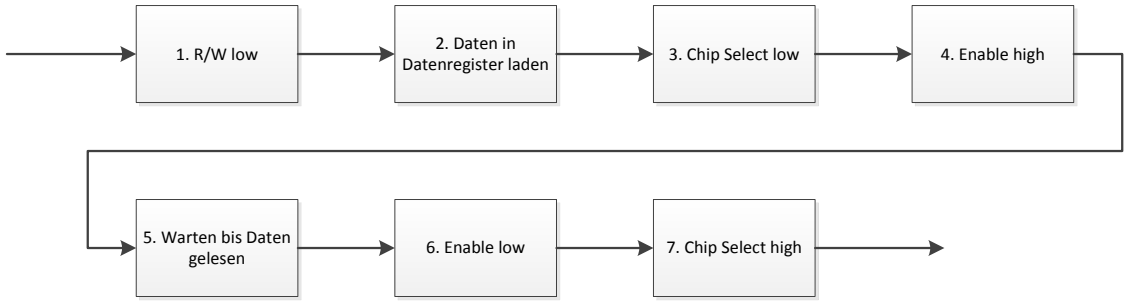


Abbildung 4.12: Ablaufsequenz Datensendung an Displaycontroller ST7565

Abbildung 4.12 zeigt den Ablauf, der erfolgt, wenn Daten an den Displaycontroller übermittelt werden. Am Anfang wird der Displaycontroller in den Schreibmodus versetzt. Anschließend werden die zu sendenden Daten in das Datenregister des Mikrocontrollers geladen. Dann wird der Controller über Chip Select low und Enable high dazu gebracht, die anliegenden Datenbits zu lesen. Anschließend wird die Übertragung dadurch beendet, dass Chip Select auf high und Enable auf Low-Pegel gesetzt werden. Um Pixel auf dem Display zu ändern, müssen Daten an das sogenannte Display RAM gesendet werden. Das Display RAM ist ein 128x65 Bit Speicher. Er bildet das Display in digitaler Form ab. Ändert man ein Bit im Display RAM, wird automatisch das entsprechende Pixel auf dem Display mit geändert. Die Adressaufteilung des Display RAM ist im Anhang B.7.5 zu finden. Der Display RAM ist in 128 Spalten und 65 Zeilen aufgeteilt. Die Zeilen wiederum sind nochmal in acht 8 Bit breite Seiten unterteilt. Die Daten werden in diesem Programm immer byteweise gesendet. Bevor ein Datenbyte gesendet wird, muss dem Displaycontroller im Command Mode die Seite und die Spalte des zu schreibenden Bytes gesendet werden. Erst danach folgt das Datenbyte, dass dann direkt an die vorher übermittelte Speicheradresse geschrieben wird.
Für die Bilder, Zahlen oder Zeichen, die das Display abbilden soll, sind spezielle Headerdateien gechrieben worden, in denen die einzelnen Bitfolgen abgespeichert sind. Insgesamt gibt es drei Headerdateien. Die Datei font.h (Anhang B.7.2) enthält den gesamten ASCII-Zeichensatz. Jedes Zeichen ist ein Byte hoch und 5 Bit breit und passt somit immer genau auf eine Seite des Display RAM. Dieser Header ist eine leicht modifizierte Variante der Datei glcdfont.h und stammt von der Hompage [13]. Die Headerdatei numbers.h (Anhang B.7.3) enthält die Ziffern von 0-9 über die gesamte Displayhöhe. Eine Nummer ist dabei 32 Bit breit. In der Headerdatei pic.h (Anhang B.7.4) ist ein Bild gespeichert, dass das ganze Display also

128x64 Pixel ausfüllt.
Der gesamte Quellcode für das Testprogramm LC Dislpay ist im Anhang B.7.1 zu finden.

4.4.7 Tastermatrix

Im Programm Tastermatrix geht es darum, eine Tastermatrix mit 10 Tastern auf die Ziffern 0-9 zu dekodieren.Die Matrix besteht aus 3 Spalten und 4 Zeilen. Jede Zeile ist auf einen Eingang gelegt, jede Spalte auf einen Ausgang (Siehe Abbildung 3.11). Durch eine logische UND-Verknüpfung der einzelnen Zeilen wird ein Interrupt ausgelöst, sobald einer der 10 Taster betätigt wurde. Innerhalb der ISR wird die Auswertung des Tasters ausgeführt. Der Programmablaufplan in Abbildung 4.13 zeigt, dass nach Auftreten eines Interrupts sofort die rechte und die mittlere Spalte auf High-Pegel gesetzt werden. Im Anschluss wird geprüft, ob an den Zeileneingängen ein Wert anliegt. Liegt noch ein Wert an, wurde ein Taster aus der linken Spalte betätigt und kann dekodiert werden. Liegt kein Signal an den Eingängen an, wird weitergeprüft, ob der Taster aus der mittleren Spalte stammt. Hierzu werden dann die linke und die rechte Spalte auf High-Pegel gesetzt. Ist nun ein Signal an den Eingängen vorhanden, kann ein Taster der mittleren Spalte zugewiesen werden. Ist bis jetzt kein Taster detektiert worden, wird die linke Spalte geprüft. Hierzu werden die rechte und die mittlere Spalte auf High-Pegel gesetzt. Nun kann der entsprechende Taster detektiert werden. Ergibt die gesamte Prüfung kein Ergebnis, liegt eine Fehldekodierung vor. Die detektierte Zahl wird anschließend in ein 4 Felder breites Array geschrieben.

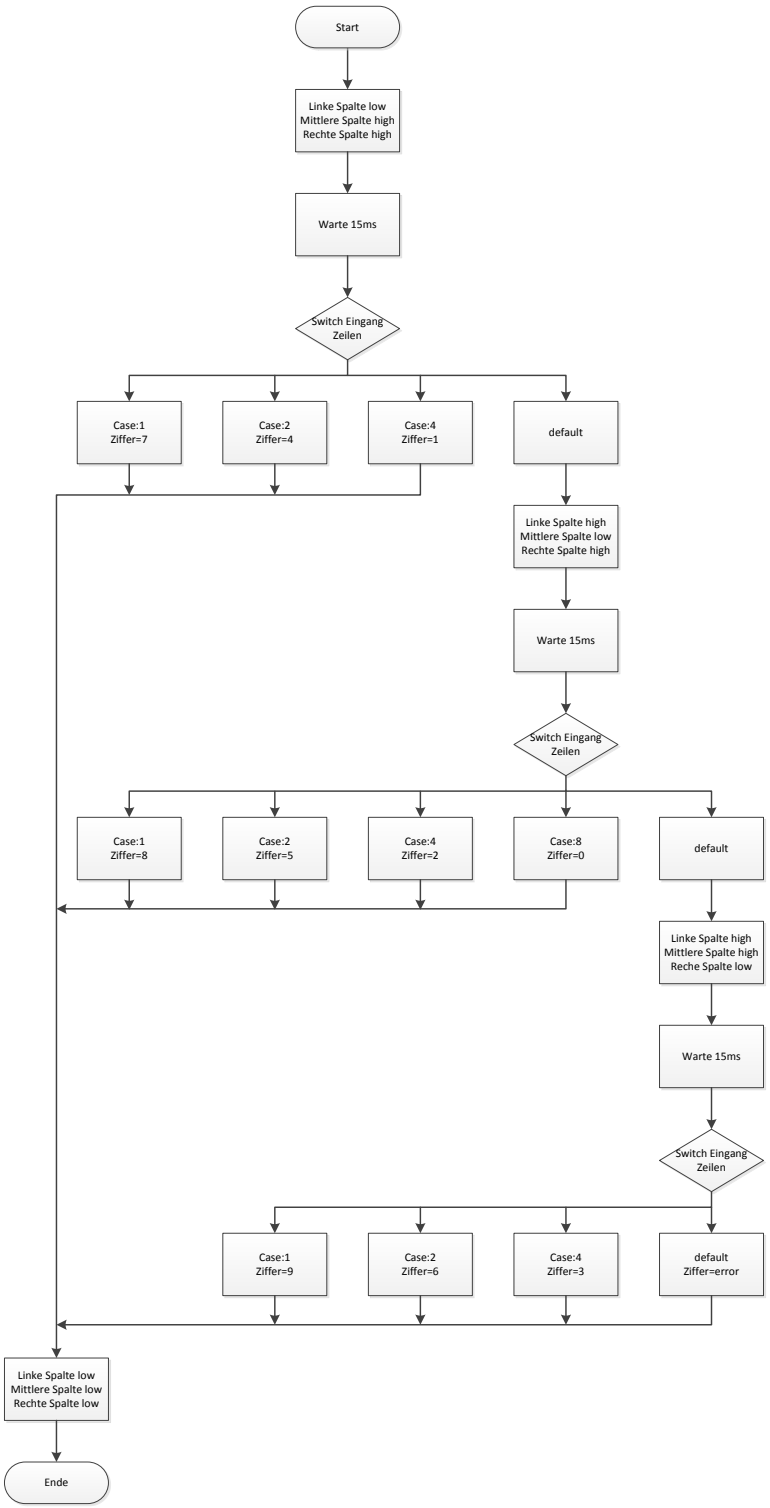


Abbildung 4.13: Flussdiagramm Zifferdetektierung

Durch die Entprellschaltung ist darauf zu achten, dass die einzelnen Spalten nicht sofort nach Umschalten der Ausgänge geprüft werden können. Grund hierfür ist das RC-Glied. Die Spannung kann nicht mehr einfach springen sondern muss der Formel 3.8 folgen. Damit die Auswertung sicher erfolgen kann, ist eine Wartezeit von 15 ms zwischen Umschalten der Spalte und Auswertung der Spalte im Programm eingeführt worden. Die nachfolgenden Messungen von $u_c(t)$ verdeutlichen, wie sich die Spannung über dem Kondensator verhält.

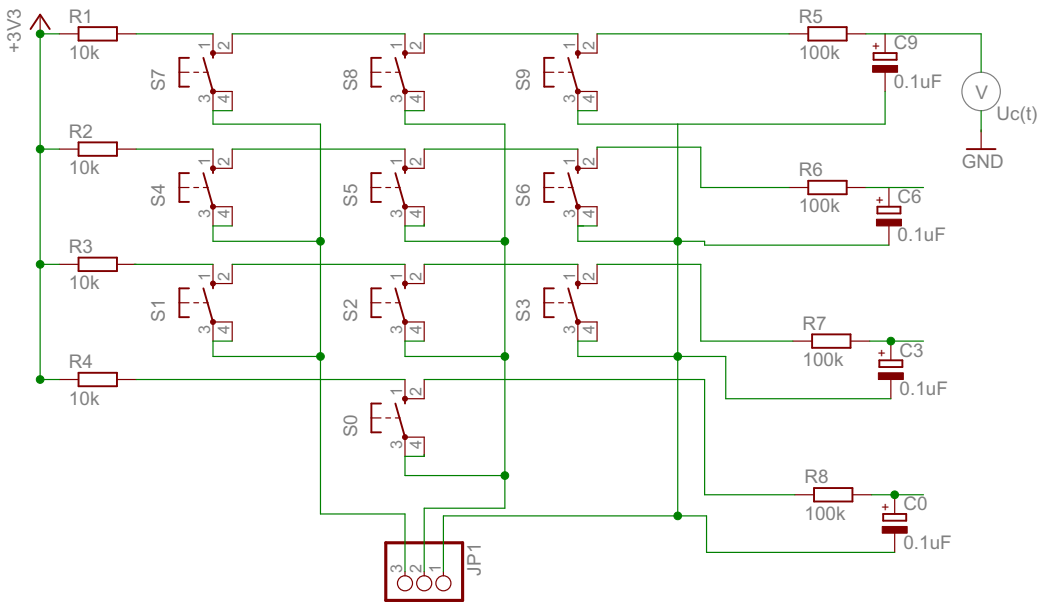


Abbildung 4.14: Messschaltung der Spannung $u_c(t)$ für verschieden Spalten

Abbildung 4.14 zeigt die Messschaltung für $u_c(t)$. Die Spannung wird direkt über dem Kondensator C9 und vor dem Schmitt Trigger gemessen.

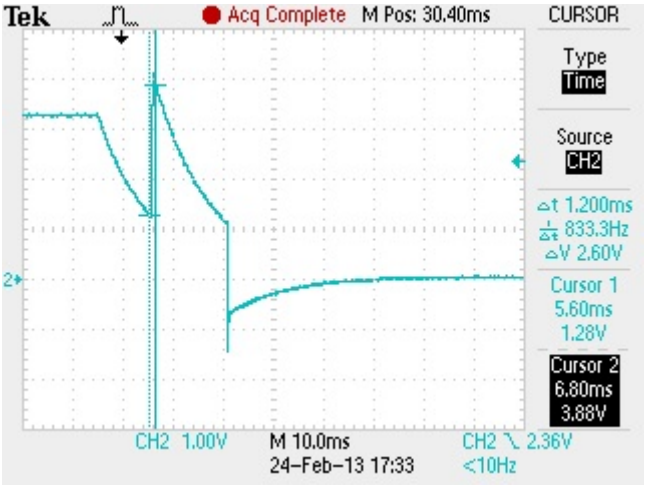


Abbildung 4.15: Messung 1: $u_c(t)$ bei Auswertung der linken Spalte

Messung 1 zeigt den Verlauf der Spannung $u_c(t)$ beim Betätigen eines Tasters in der linken Spalte. Nach Betätigung fällt $u_c(t)$ erst unter die Schwellspannung V_N=1,3 V des Schmitt Triggers, dann wird der Interrupt ausgelöst. Die Spannung springt sofort auf 4,2 V. Der Sprung resultiert aus der Restspannung $u_c(t)$ und der zugeschalteten Spannung von 3,3 V der rechten Spalte. Die Cursor messen die Zeit, die zwischen Auslösen des Interrupts und dem Setzen der Spalte auf den Low-Pegel vergeht. Hier wird die linke Spalte ausgewertet. Die Spalte wird direkt nach Auslösen des Interrupts auf den Low-Pegel gezogen. Das bedeutet, die Auswertung tritt 15 ms nach Auftreten des Interrupts auf. Diese Wartezeit muss so gewählt werden, dass $u_c(t)$ unter 1,3 V, also V_N des Schmitt Triggers, fällt. Würde nicht gewartet werden, käme es zu einer Fehlauswertung, da noch alle Eingänge am Mikrocontroller Low-Pegel hätten. Direkt nach der Auswertung werden alle Spalten auf Low-Pegel gesetzt. Das hat zur Folge, dass $u_c(t)$ nochmal schlagartig um 3,3 V fällt. Danach entlädt sich der Kondensator und der Vorgang der Auswertung ist abgeschlossen.

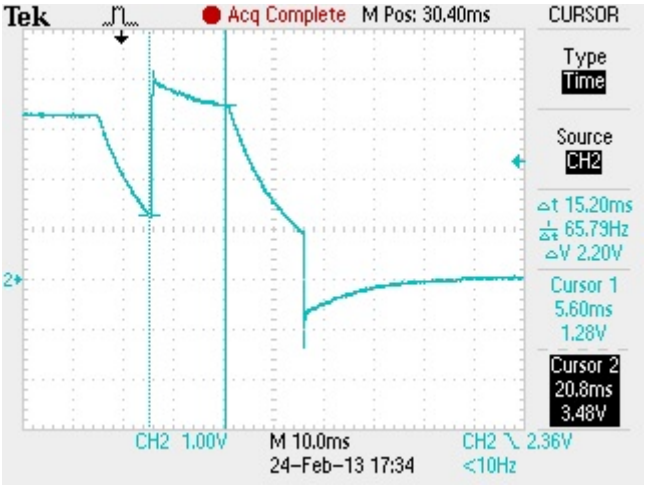


Abbildung 4.16: Messung 2: $u_c(t)$ bei Auswertung der mittleren Spalte

Messung 2 zeigt eine Auswertung der mittleren Spalte. Direkt nach dem Interrupt entlädt sich der Kondensator bis auf die angelegte Spannung von 3,3 V. Es ist zu erkennen, dass nach 15 ms die mittlere Spalte auf den Low-Pegel gezogen wird und sich der Kondensator weiter auf 0 V entladen kann. Die Auswertung der Spalte erfolgt dann nach 30 ms, um eine Fehldetektion zu vermeiden. Direkt nach der Auswertung ist ebenfalls der Spannungssprung von 3,3 V zu beobachten.

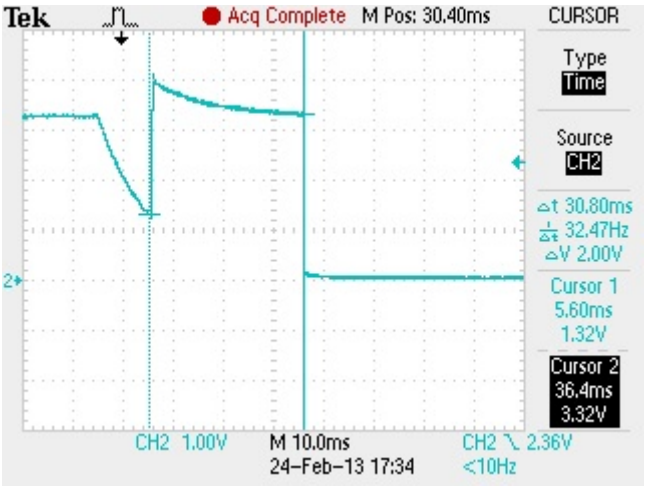


Abbildung 4.17: Messung 3: $u_c(t)$ bei Auswertung der rechten Spalte

Messung 3 zeigt die Auswertung der rechten Spalte. Erst nach 30 ms wird der Pegel auf low gezogen. Da beim Schalten auf den Low-Pegel direkt 3,3 V abgeschaltet werden, fällt

die Spannung direkt auf fast 0 V. Die Auswertung erfolgt dann 45 ms nach Auftreten des Interrupts.
Die Ausgabe der detektierten Ziffern erfolgt auf dem LCD. Jede Ziffer wird nacheinander ausgegeben. Dies geschieht durch das Programm: Testprogramm Liquid Crystal Display und dem Header numbers.h (Anhang B.7.3). Sind vier Ziffern eingegeben, werden diese mit einem, im Programm gespeicherten Code, verglichen. Stimmen die eingeggebenen Ziffern mit dem Code überein, wird das Display weiß und ist zur nächsten Eingabe bereit. Ist der Code falsch, wird das Display schwarz und es kann erneut ein Code eingegeben werden. Die Reaktion auf den Code erfolgt entweder durch die Displaycontrollerfunktion lcd_clear() oder die Funktion lcd_invers(). Der gesamte Quellcode des Programms ist im Anhang B.8.1 zu finden. Die Displayfunktionen, die im Testprogramm Liquid Crystal Display erstellt wurden, sind in eine separate C-Datei ausgelagert worden, werden aber in diesem Programm wieder benutzt.

5 Schluss

5.1 Zusammenfassung

Die Aufgabe der Arbeit war es, ein Experimentierboard für ARM-Mikrocontroller zu planen und zu realisieren. Um dieses Ziel zu erreichen, ist zuerst der Stellaris LM3S9D92 ausgewählt worden. Dieser Mikrocontroller ist gewählt worden weil er der direkte Nachfolger zu den, in den Laboren der HAW verwendeten, Mikrocontrollern Stellaris LM3S9B92 ist.
Konzeptionell wurde die Arbeit folgendermaßen aufgebaut: Zuerst wurde eine Hauptplatine für den Mikrocontroller, den Programmer und die Spannungsversorgung entwickelt und gefertigt. Im Anschluss wurden weitere Platinen und Komponenten entwickelt, um die grundlegenden Peripherien des Mikrocontrollers in interessanten Versuchen zu benutzen und aufeinanderaufbauend zu verstehen. Aus diesem Konzept sind noch die Modulplatinen 7-Segment-Multiplex-Platine, LCD Platine und die Tastermatrixplatine enstanden. Die Modulplatinen wurden als Prototyp auf Lochrasterplatine aufgebaut. Alle Platinen besitzen fertige EAGLE-Board-Dateien und können sofort gefertigt werden.
Ein wichtiges Kriterium für das Experimentierboard war, dass es modular aufgebaut sein sollte. Dieser Punkt wurde in der Arbeit sehr sorgfältig verfolgt und umgesetzt. Es ist möglich, dass nicht das ganze Experimentierboard nachgebaut werden muss, damit der Anwender einen funktionierenden ARM-Mikrocontroller zur Verfügung hat. Jedes einzelne Modulboard wird nur für einen bestimmten Versuch verwendet und kann bei Nichtgebrauch einfach ignoriert werden. Auf der Hauptplatine ist das Layout darauf ausgerichtet, dass Module, die zusammengehören auch physikalisch nah beieinander positioniert sind. Dadurch ist es möglich, dass auf der Hauptplatine für den Mikrocontrollerbetrieb unwichtige Module einfach weggelassen werden können. Um den Mikrocontroller nicht fest zu verbauen, wurden seine Ausgänge auf Steckerbuchsen gelegt und können so für jede beliebige Anwendung genutzt werden. Um die einzelnen Peripheriefunktionen und den Mikrocontroller kennenzulernen und in den einzelnen Experimenten zu testen, wurden verschiedene Testprogramme mit verschiedenen Schwierigkeitsgraden programmiert. Die Testprogramme wurden alle auf Registerebene des Mikrocontrollers geschrieben. So ist es für den Anwender möglich, direkt zu sehen, welcher Änderungen es an den Registern für die bestimmte Peripherie bedarf. Alle Programme können mit dem Experimentierboard als Experiment nachgebaut und in Betrieb genommen werden. Die Testprogramme benutzen die für Einsteiger wichtigsten

Peripherien des Mikrocontrollers. Dazu zählen Ein- und Ausgänge, Interrupts, ADC, Systemtakt, Timer und das Einrichten von Bibliotheken. Hat der Anwender alle Experimente durchgearbeitet, sollte er die grundsätzlichen Hardwaremöglichkeiten des Mikrocontrollers kennen und anwenden können.

5.2 Ausblick

Für eine Weiterentwicklung des Experimentierboards wären die Ausarbeitungen von Versuchen für USB und Ethernet denkbar. Diese beiden Hardwarekomponenten sind schon auf dem Experimentierboard vorgesehen und zur Benutzung bereit. Außerdem wäre es denkbar, Versuche für die serielle Kommunikationsperipherie wie SSI, CAN-Bus, UART oder I^2C zu entwickeln. Diese Versuche würden dem Anwender dann weiterführende Einblicke in speziellere Peripherien des Stellaris LM3S9D92 bieten. Ein anderer Ansatzpunkt für die Weiterentwicklung des Experimentierboards wäre es, gänzlich alleinstehende Treiber für den FTDI-Chip zu schreiben und somit nicht mehr auf die TI Treiber angewiesen zu sein. Dieser Schritt sollte relativ einfach durchführbar sein, da FTDI bereits Aplication Notes zum Umschreiben ihrer Treiber ausgegeben hat.

5.3 Fazit

Zusammenfassend ist zu sagen, dass das Experimentierboard sehr gut für Anfänger auf dem Gebiet der Texas Instruments ARM Mikrocontroller geeignet ist. Durch die einzelnen Experimente und die freie Bauform des Experimentierboards ist eine günstige und weit umfassende Lernumgebung für den Stellaris LM3S9D92 entstanden. Durch den Hardwareaufbau der einzelnen Platinen lernen Interessierte nicht ausschließlich den Umgang mit dem Mikrocontroller sondern auch das Erstellen von Platinen und einfachen elektrischen Schaltungen.

Literaturverzeichnis

[1] Texas Instruments: *Stellaris LM3S9D92 Microcontroller Data Sheet*. Version: Januar 2012. Austin, Texas, . – http://www.ti.com/lit/ds/symlink/lm3s9d92.pdf abgerufen am 12.10.2012

[2] Texas Instruments: *LM35 Precision Centigrade Temperature Sensors*. Version: November 2000. Austin, Texas, . – http://www.ti.com/lit/ds/symlink/lm35.pdf abgerufen am 20.02.2013

[3] Maxim Intergrated: *DS18S20 High-Precision 1-Wire Digital Thermometer*. Version: August 2010. San Jose, . – http://datasheets.maximintegrated.com/en/ds/DS18S20.pdf abgerufen am 20.02.2013

[4] TIETZE, Ulrich ; SCHENK, Christoph: *Halbleiter-Schaltungstechnik*. Springer, 2002. – ISBN 3–540–42849–6

[5] Beta Layout GmbH: *Homepage: Vorgaben PCB-Pool*. Aarbergen, . – http://www.pcb-specification.com/de abgerufen am 18.02.2013

[6] Texas Instruments: *LM2574/LM2574HV Simple Switcher 0,5A Step-Down Voltage Regulator*. Version: November 2004. Austin, Texas, . – http://www.ti.com/lit/ds/symlink/lm2574.pdf abgerufen am 18.02.2013

[7] Future Technology Devices International Ltd.: *D2XX_Programmer‘s_Guide*. Version: 1.3. Glasgow, UK, . – http://www.ftdichip.com/Support/Documents/ProgramGuides/D2XX_Programmer%27s_Guide%28FT_000071%29.pdf abgerufen am 22.02.2013

[8] Texas Instruments: *Stellaris LM3S9D92 Evaluation Kit User‘s Manual*. Version: July 2012. Austin, Texas, 2011. – http://www.ti.com/lit/ug/spmu174/spmu174.pdf abgerufen am 02.10.2012

[9] Future Technology Devices International Ltd.: *FT2232D Dual USB to Serial UART/FIFO IC Datasheet*. Version: 2.05. – http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232D.pdf abgerufen am 19.02.2013

[10] Omron: *Tactile Switch B3F*. – http://www.omron.com/ecb/products/pdf/en-b3f.pdf abgerufen am 24.02.2013

[11] Future Technology Devices International Ltd.: *Application Note AN_124 User Guide For FTDI FT_Prog Utility*. Version: 1.4. Glasgow, UK, . – http://www.ftdichip.com/Support/Documents/AppNotes/AN_124_User_Guide_For_FT_PROG.pdf abgerufen am 21.02.2013

[12] Texas Instruments: *Homepage: How to install the terminal plugin in CCSv5*. – http://processors.wiki.ti.com/index.php/How_to_install_the_terminal_plugin_in_CCSv5 abgerufen am 05.02.2013

[13] Ladyada: *Homepage Ladyada ST7565 LCDs*. – http://www.ladyada.net/learn/lcd/st7565.html abgerufen am 03.11.2012

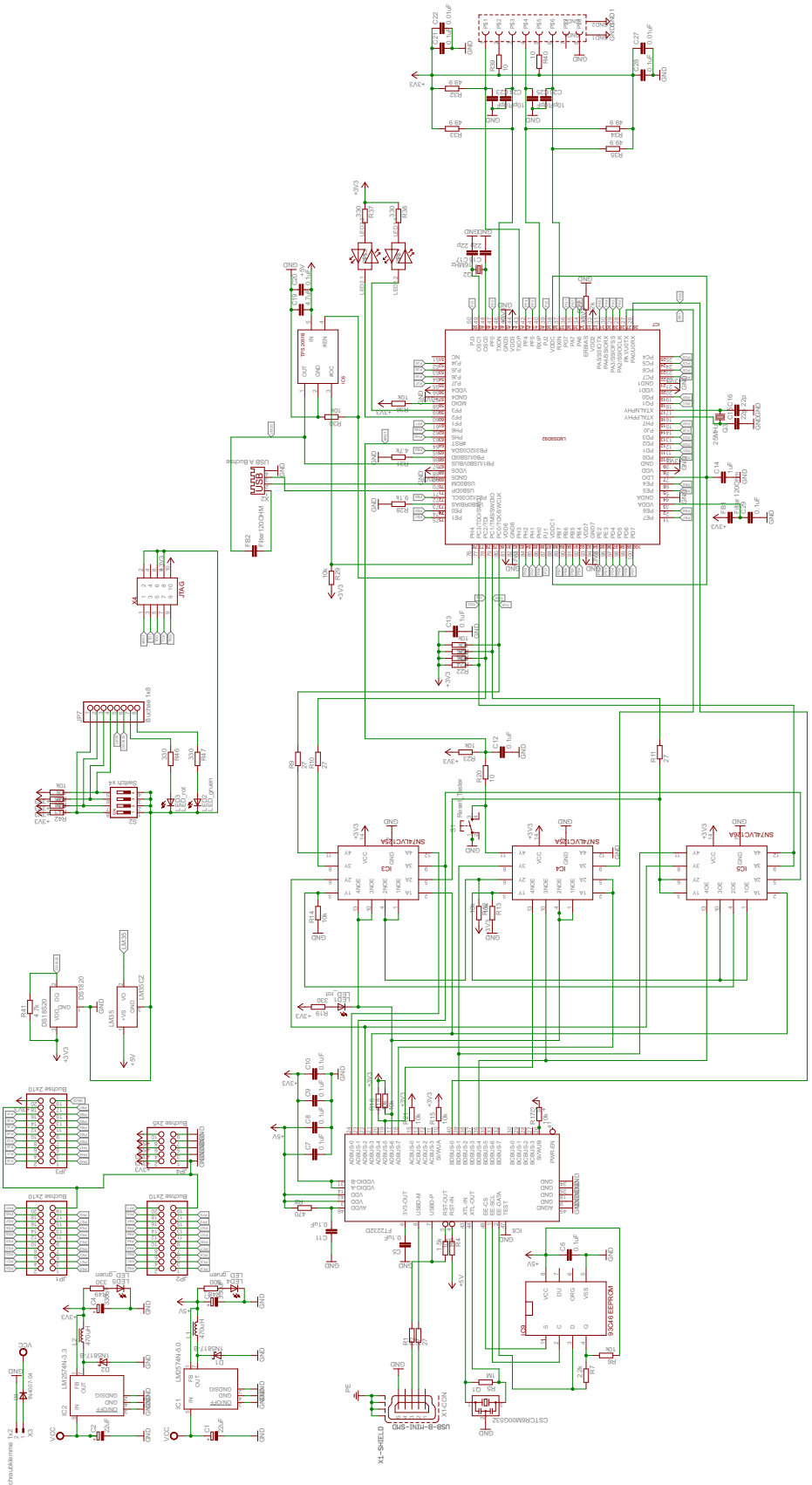
[14] Sitronx: *ST7565R 65x132 Dot Matrix LCD Controller/Driver*. Version: 1.5. – http://www.lcd-module.de/eng/pdf/zubehoer/st7565r.pdf abgerufen am 08.02.2013

Anhang

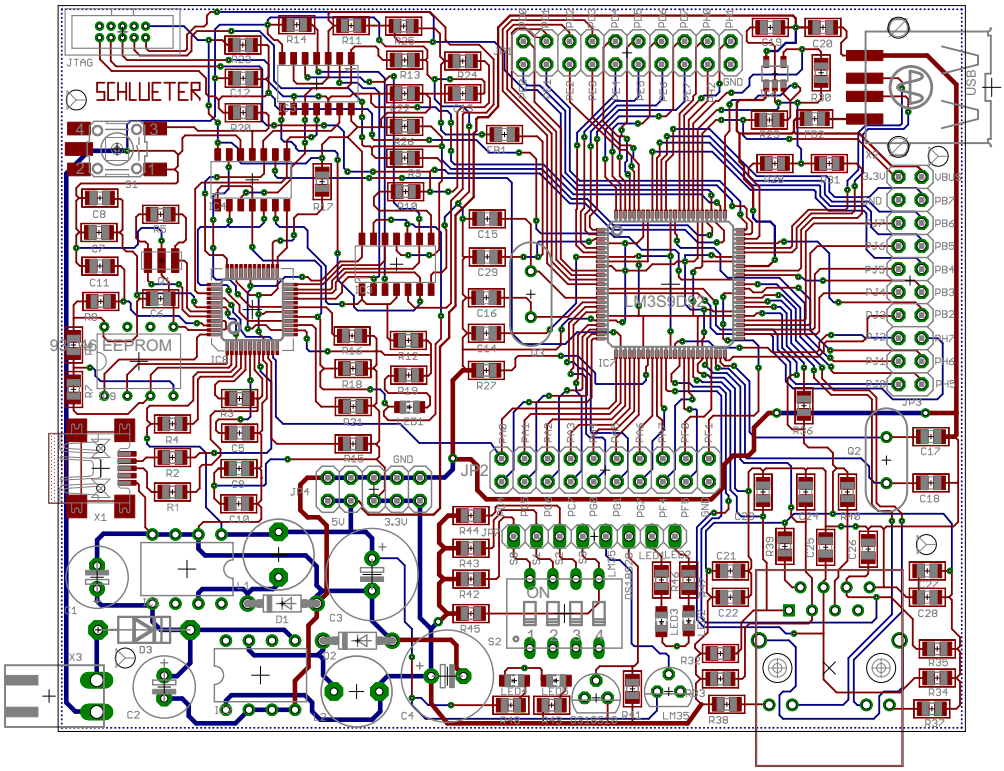
A Schaltpläne

A.1 Hauptplatine

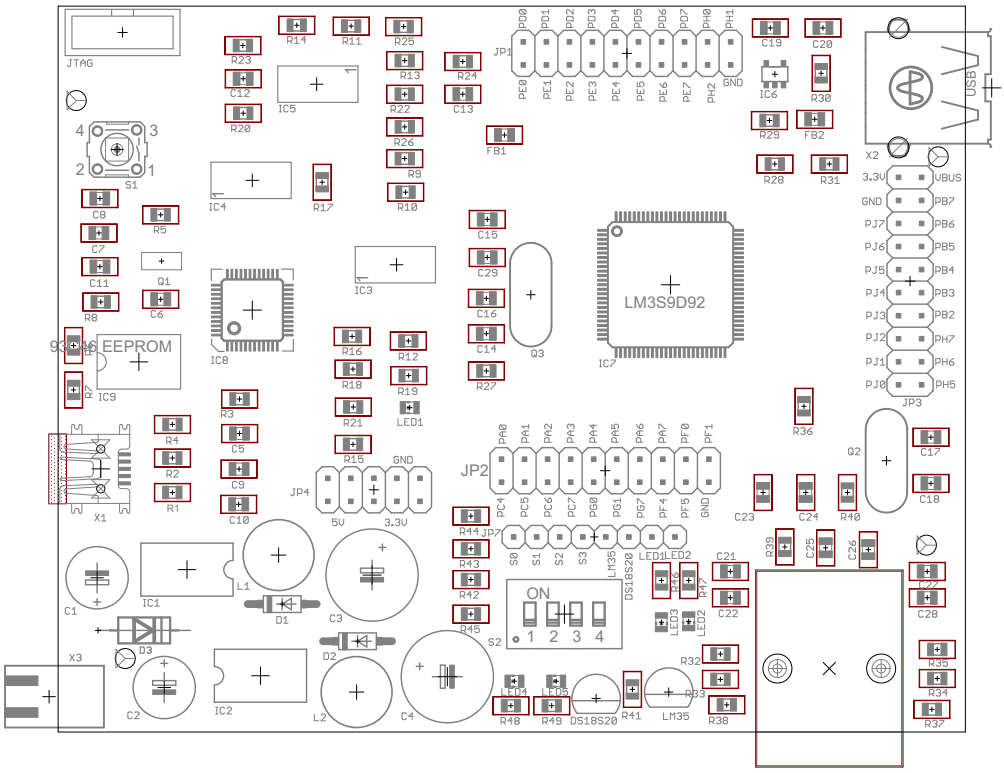
A.1.1 Schaltplan Hauptplatine



A.1.2 Layout Hauptplatine



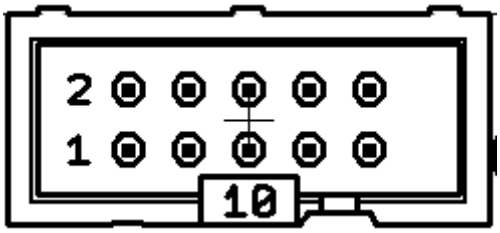
A.1.3 Bestückungsplan Hauptplatine



A.1.4 Bauteilliste Hauptplatine

Menge	Wert	Device	Bauteile
2	0.01uF	C-EUC0805	C22, C27
13	0.1uF	C-EUC0805	C5, C6, C7, C8, C9, C10, C11, C12, C13, C20, C21, C28, C29
1	1.5k	R-EU_R0805	R3
1	1M	R-EU_R0805	R5
1	1N4007-04	1N4007-04	D3
2	1N5817-B	1N5817-B	D1, D2
1	1uF	C-EUC0805	C14
1	2.2k	R-EU_R0805	R7
2	4.7k	R-EU_R0805	R31, R41
1	4.7uF	C-EUC0805	C19
1	9.1k	R-EU_R0805	R28
3	10	R-EU_R0805	R20, R39, R40
22	10k	R-EU_R0805	R4, R6, R12, R13, R14, R15, R16, R17, R18, R21, R22, R23, R24, R25, R26, R29, R30, R36, R42, R43, R44, R45
4	10pF	C-EUC0805	C23, C24, C25, C26
1	12k	R-EU_R0805	R27
1	16MHz	CRYSTALHC49US	Q2
4	22p	C-EUC0805	C15, C16, C17, C18
2	22uF	CPOL-EUE2.5-7	C1, C2
1	25MHz	CRYSTALHC49US	Q3
5	27	R-EU_R0805	R1, R2, R9, R10, R11
4	49.9	R-EU_R0805	R32, R33, R34, R35
1	93C46 EEPROM	ST_93C46_DIP08	IC9
6	330	R-EU_R0805	R19, R37, R38, R46, R47, R49
2	330uF	CPOL-EUE3.5-10	C3, C4
1	470	R-EU_R0805	R8
2	470uH	L-RL8XX	L1, L2
1	500	R-EU_R0805	R48
1	Buchse 1x8	PINHD-1X8	JP7
1	Buchse 2x5	PINHD-2X5	JP4
3	Buchse 2x10	PINHD-2X10	JP1, JP2, JP3
1	CSTCR6M00G53Z	CSTCR6M00G53Z	Q1
1	DS1820	DS1820	DS18S20
1	FT2232D	FT2232D	IC8
1	Filter 120Ohm	C-EUC0805	FB1
1	Filter120OHM	C-EUC0805	FB2
1	JTAG	BKL_2X5_1.27	X4
3	LED_gruen	LEDCHIP-LED0805	LED2, LED4, LED5
2	LED_rot	LEDCHIP-LED0805	LED1, LED3
1	LM3S9D92	LM3S9D92	IC7
1	LM35CZ	LM35CZ	LM35
1	LM2574N-3.3	LM2574N-3.3	IC2
1	LM2574N-5.0	LM2574N-5.0	IC1
1	MAGJACK	MAGJACK	X5
1	Reset_Taster	LSG	S1
2	SN74LVC125A	TI_SN74LVC125A_SIOC14	IC3, IC4
1	SN74LVC126A	TI_SN74LVC126A_SIOC14	IC5
1	Schraubklemme 1x2	90-2	X3
1	Switch x4	DIP04YL	S2
1	TPS 2051B	TPS2051B	IC6
1	USB A Buchse	USB-A003	X2
1	USB-B-MINI-SMD	USB-B-MINI-SMD	X1

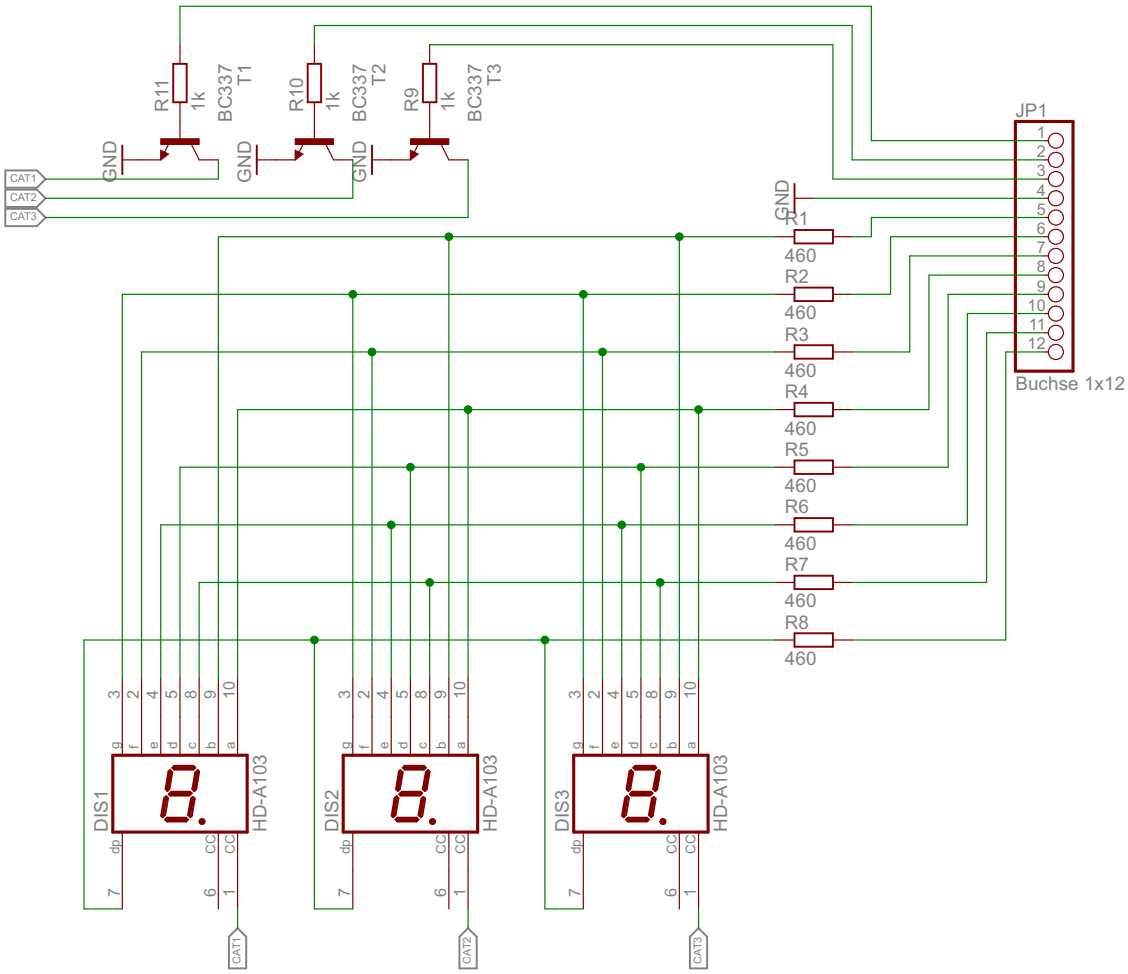
A.1.5 Pinbelegung JTAG Buchsenstecker



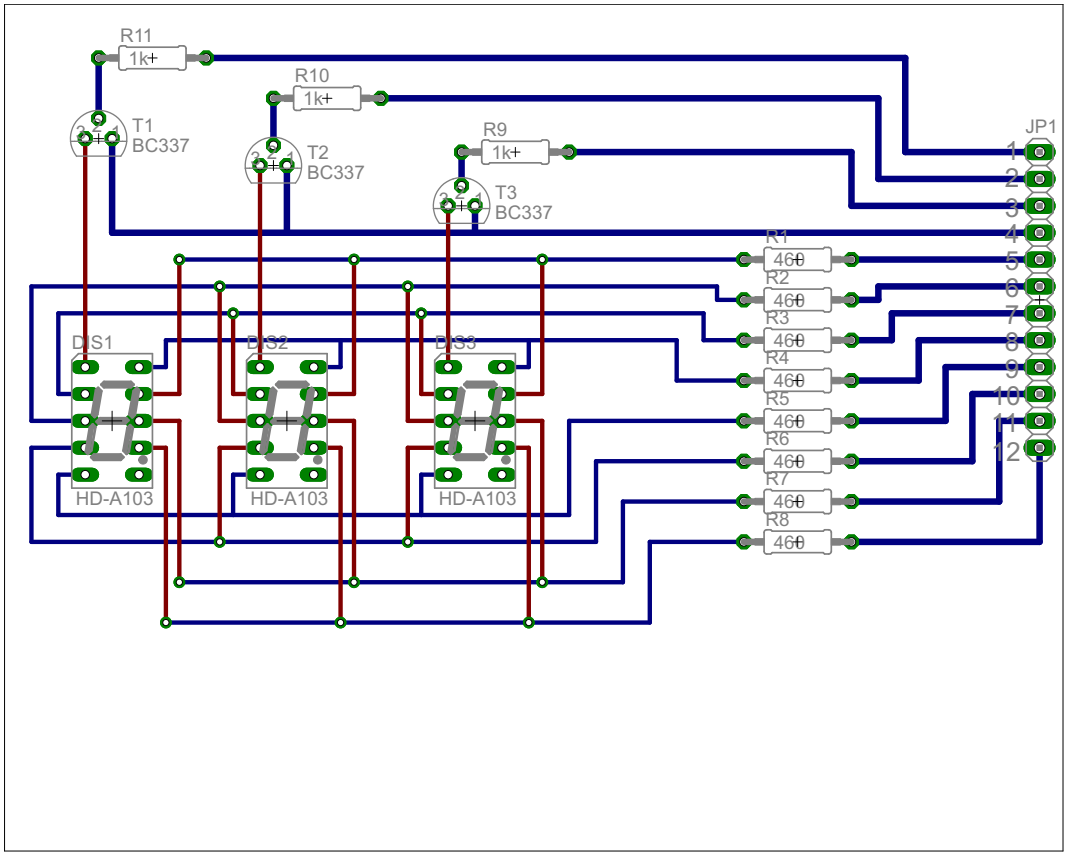
Pin	Wert
1	3,3V
2	TMS
3	GND
4	TCK
5	GND
6	TDO
7	N.C.
8	TDO
9	GND
10	#RST

A.2 7-Segment-Multiplex

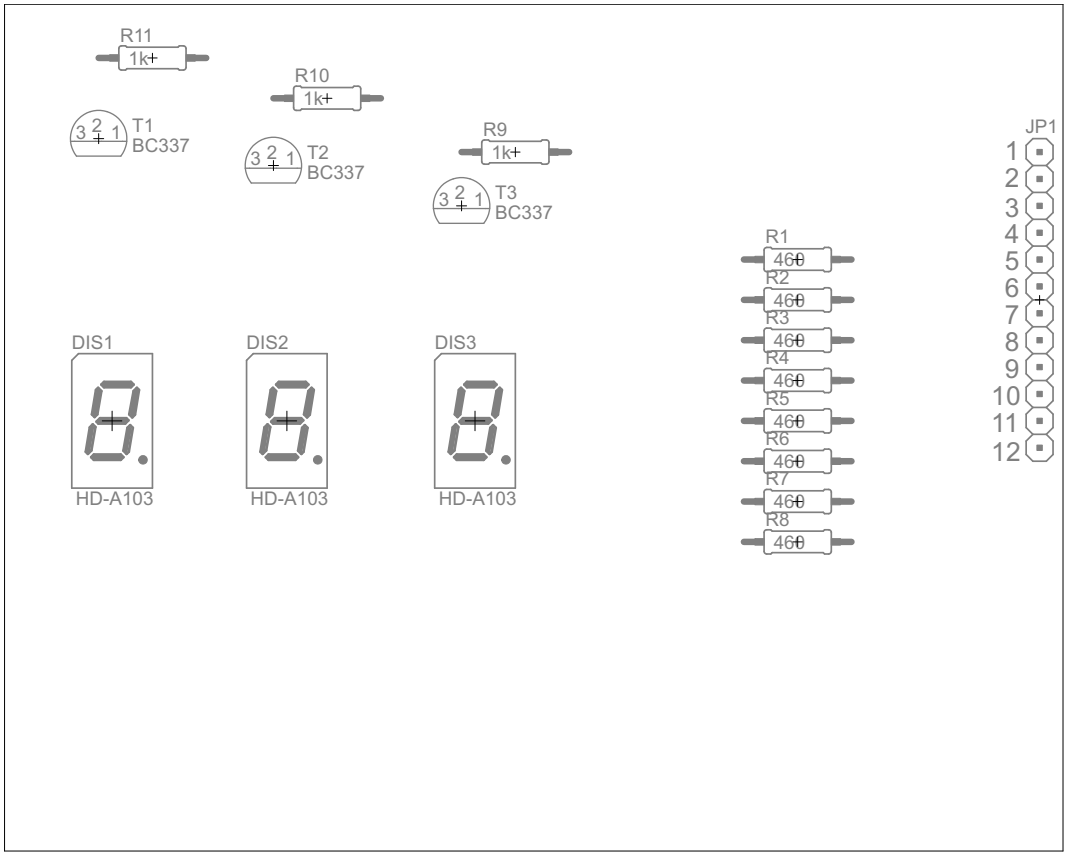
A.2.1 Schaltplan 7-Segment-Multiplex Platine



A.2.2 Layout 7-Segment-Multiplex Platine



A.2.3 Bestückungsplan 7-Segment-Multiplex Platine



A.2.4 Bauteilliste 7-Segment-Multiplex Platine

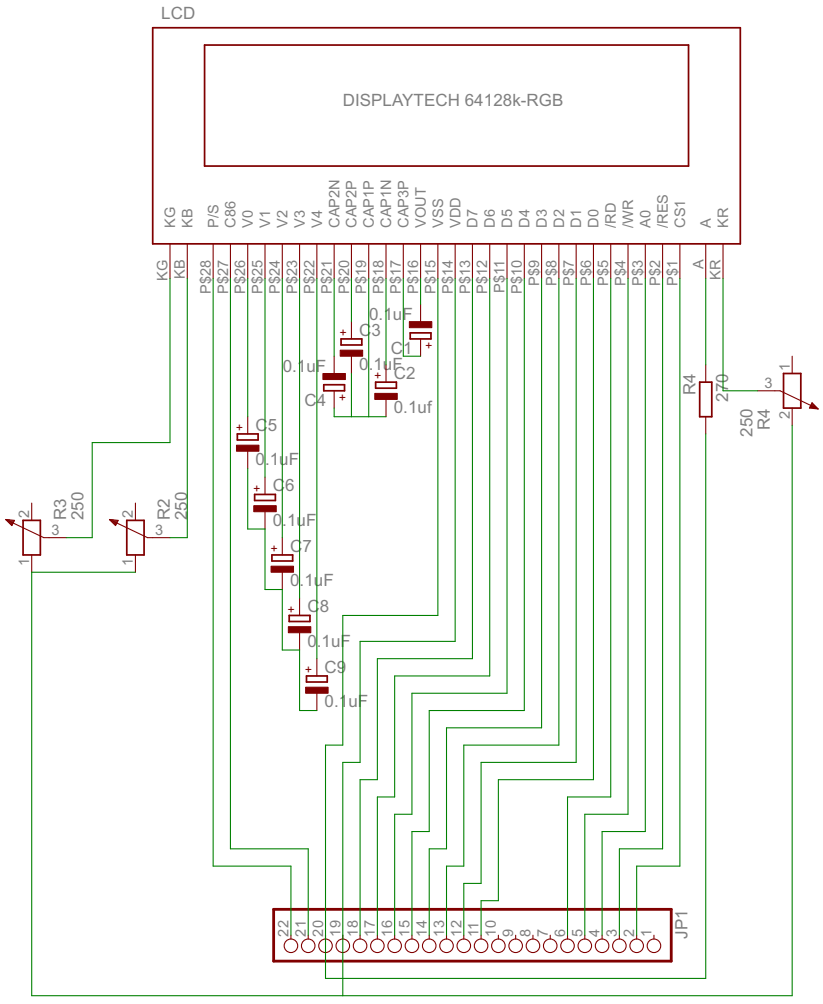
Menge	Wert	Device	Bauteile
3	1k	R-EU_0207/10	R9, R10, R11
8	460	R-EU_0207/10	R1, R2, R3, R4, R5, R6, R7, R8
3	BC337	BC337	T1, T2, T3
1	Buchse 1x12	PINHD-1X12	JP1
3	HD-A103	HD-A103	DIS1, DIS2, DIS3

A.2.5 Pinliste 7-Segment-Multiplex Platine

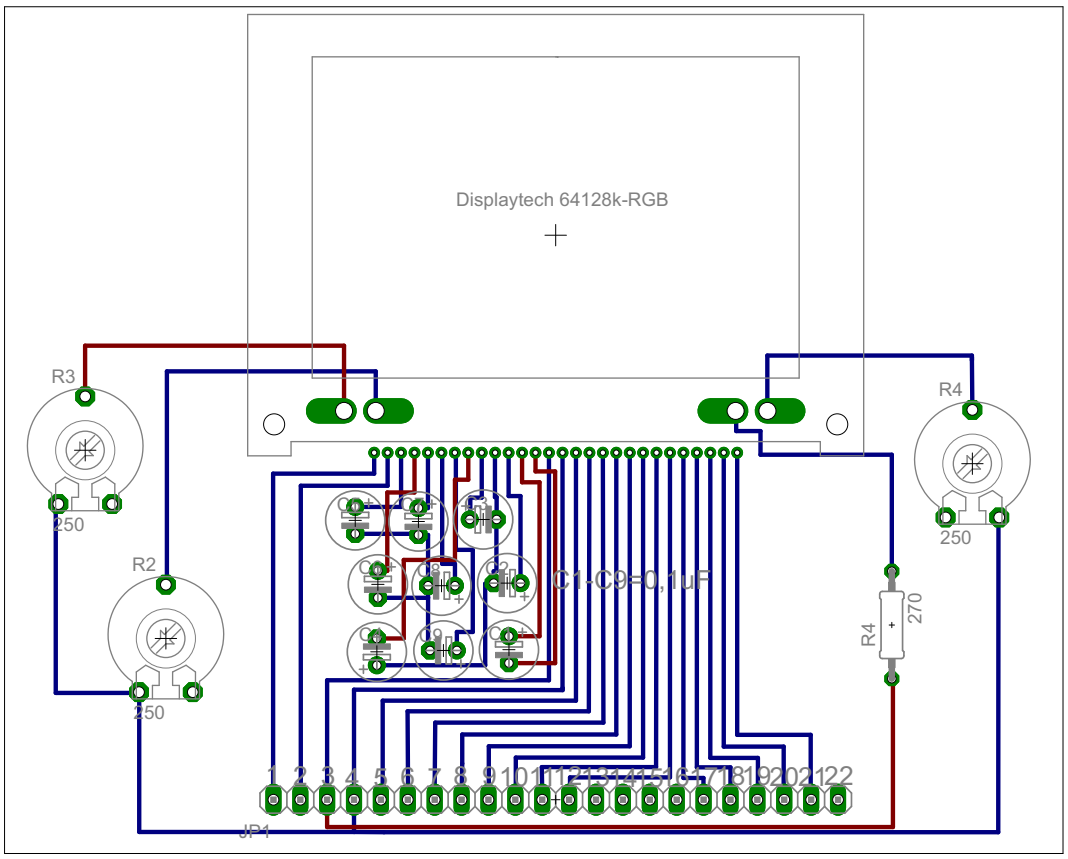
Pin JP1	Funktion	Anschluß für Testprogramm
1	Segment 1	PJ1
2	Segment 2	PJ2
3	Segment 3	PJ3
4	GND	GND
5	b	PD6
6	g	PD0
7	f	PD1
8	a	PD7
9	d	PD3
10	e	PD2
11	c	PD5
12	DP	PD4

A.3 LCD Platine

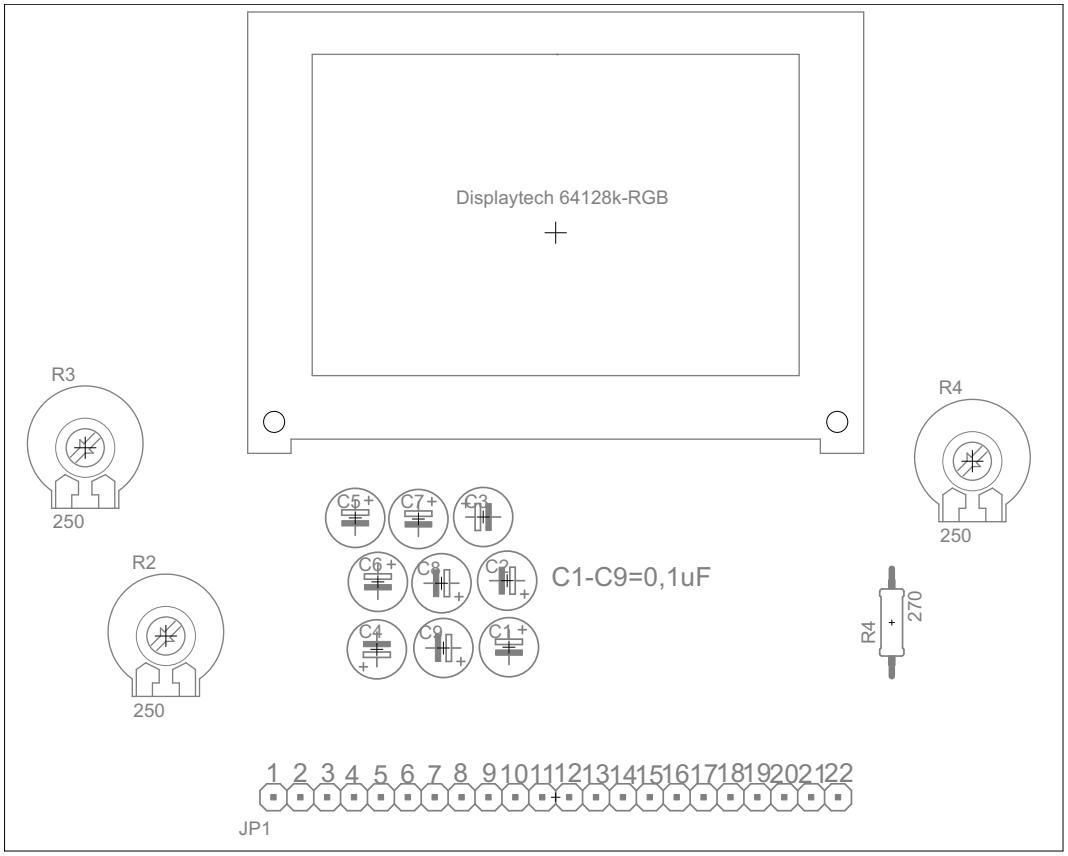
A.3.1 Schaltplan LCD Platine



A.3.2 Layout LCD Platine



A.3.3 Bestückungsplan LCD Platine



A.3.4 Bauteilliste LCD Platine

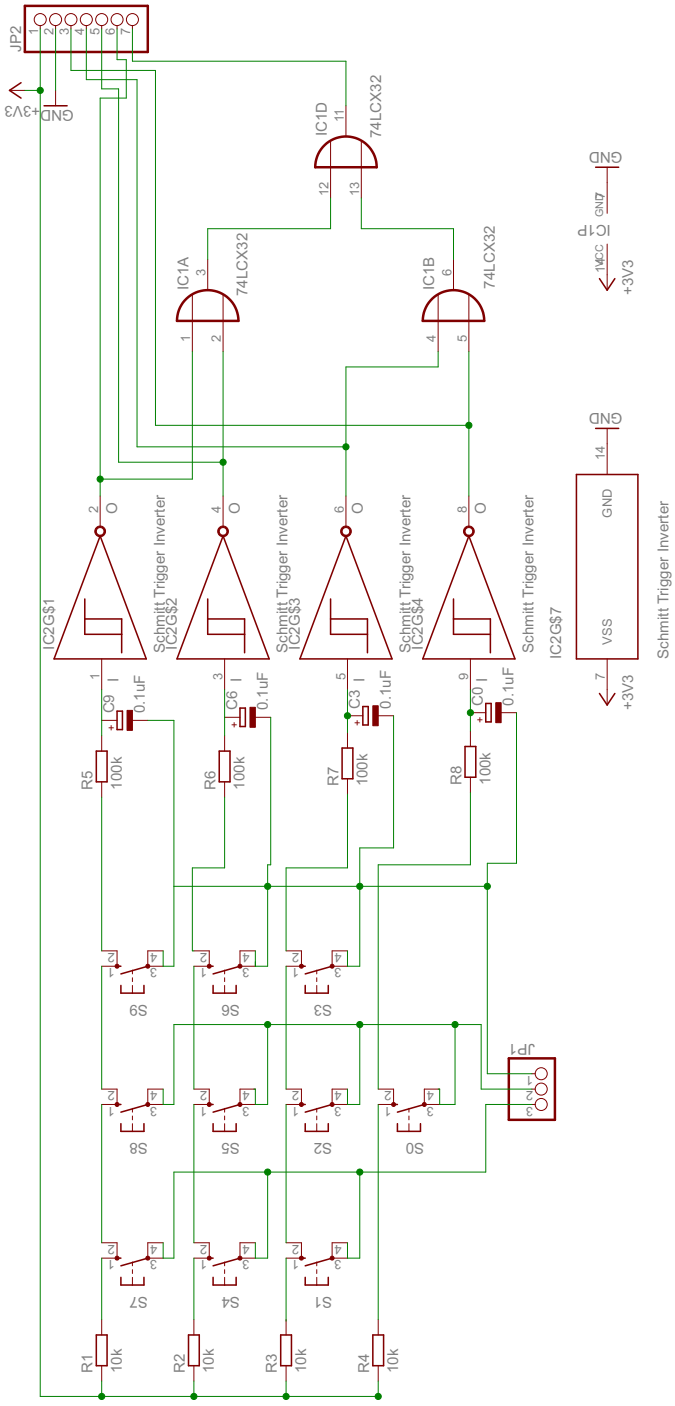
Menge	Wert	Device	Bauteile
8	0.1uF	CPOL-EUE2.5-6	C1, C3, C4, C5, C6, C7, C8, C9
1	0.1uf	CPOL-EUE2.5-6	C2
1	128x64 Pixel	DISPLAYTECH64128K_RGB	LCD
3	250	POTENTIOMETER_PT-10	R2, R3, R4
1	Buchse 1x22	PINHD-1X22	JP1
1	R4	R-EU_0207/10	270

A.3.5 Pinliste LCD Platine

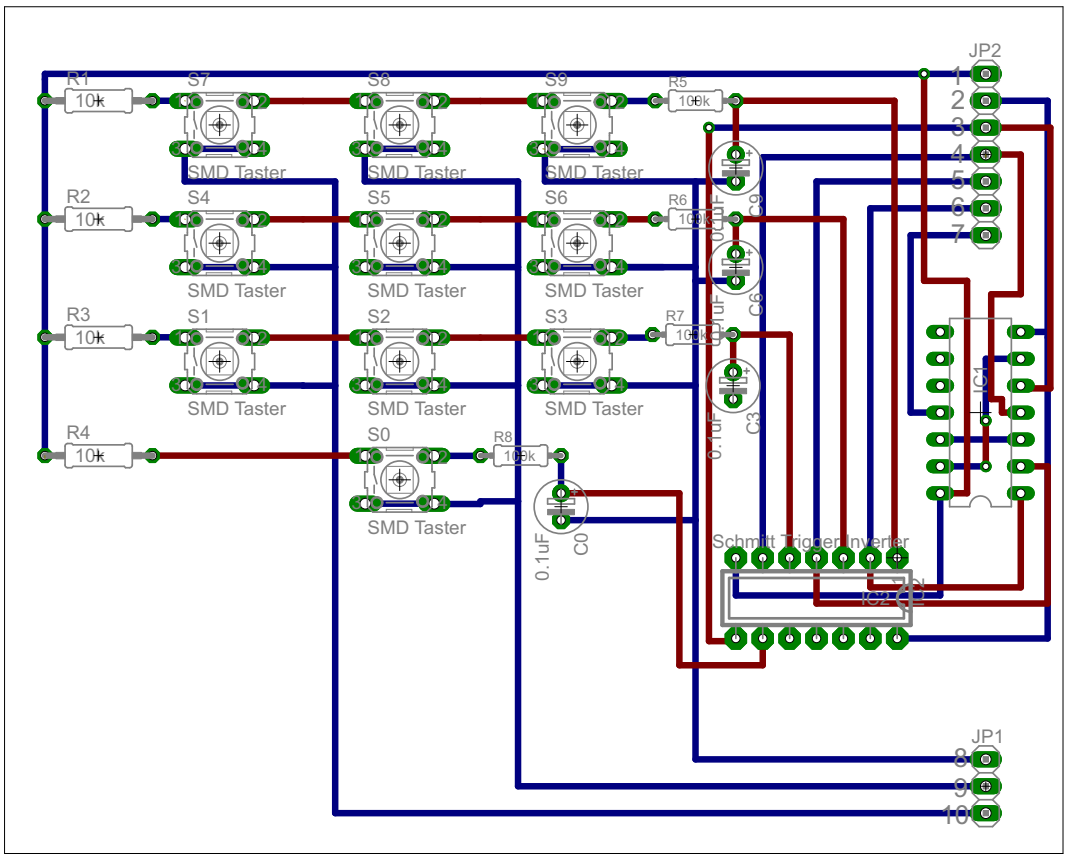
Pin JP1	Funktion	Anschluß für Testprogramm
1	P/S	PH6
2	C86 2	PH5
3	Vss	GND
4	Vdd	3,3V
5	D7	PD7
6	D6	PD6
7	D5	PD5
8	D4	PD4
9	D3	PD3
10	D2	PD2
11	D1	PD1
12	D0	PD0
13	N.C.	N.C
14	N.C. 2	N.C.
15	N.C. 3	N.C.
16	N.C.	N.C.
17	/RD	PH1
18	/WR	PH0
19	A0	PH2
20	/RES	PG1
21	CS1	PH7
22	N.C.	N.C.

A.4 Tastermatrix Platine

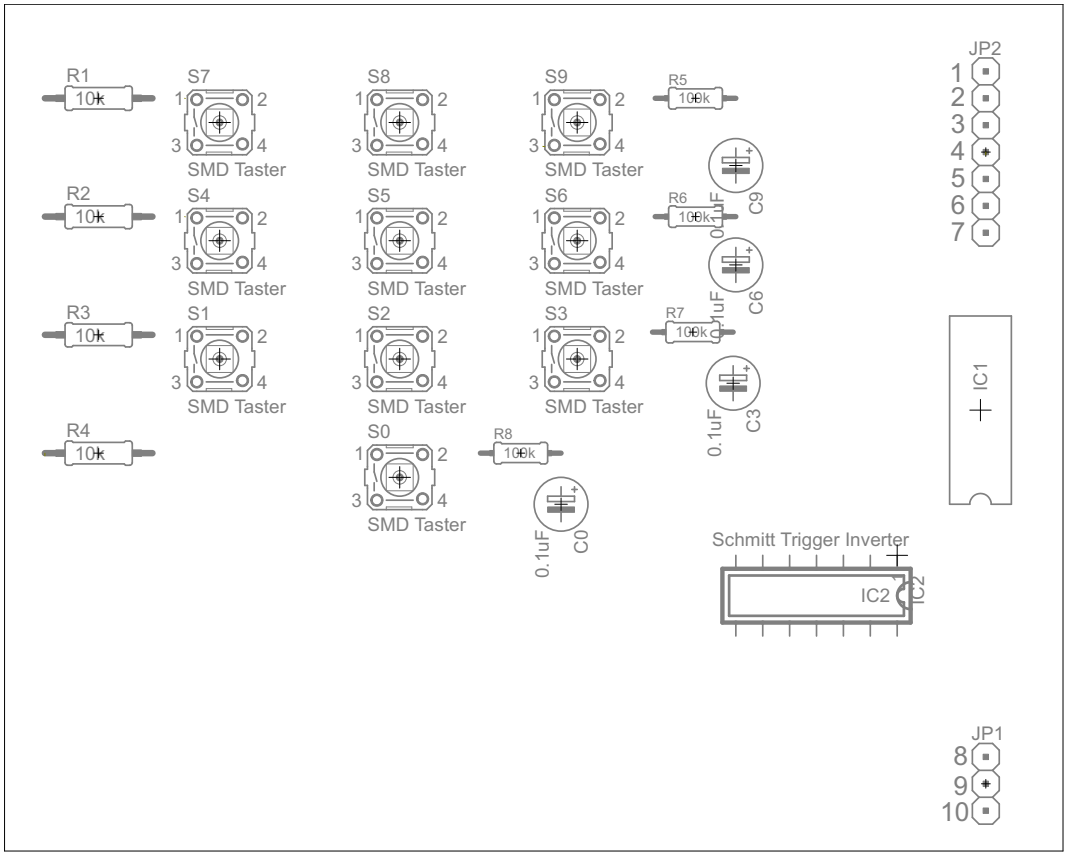
A.4.1 Schaltplan Tastermatrix Platine



A.4.2 Layout Tastermatrix Platine



A.4.3 Bestückungsplan Tastermatrix Platine



A.4.4 Bauteilliste Tastermatrix Platine

Menge	Wert	Device	Bauteile
4	0.1uF	CPOL-EUE2.5-5	C0, C3, C6, C9
4	100k	R-EU_0204/7	R5, R6, R7, R8
4	10k	R-EU_0207/10	R1, R2, R3, R4
1	74HC32N	74HC32N	IC1
1	Buchse 1x3	PINHD-1X3	JP1
1	Buchse 1x7	PINHD-1X7	JP2
10	SMD Taster	10-XX	S0, S1, S2, S3, S4, S5, S6, S7, S8, S9
1	Schmitt Trigger Inverter	HEF40106_MODUL	IC2

A.4.5 Pinliste Tastermatrix Platine

Pin JP1	Funktion	Anschluß für Testprogramm
1	9,6,3	PE5
2	8,5,2,0 2	PE6
3	7,4,1	PE7

Pin JP2	Funktion	Anschluß für Testprogramm
1	3,3V	3,3V
2	GND	GND
3	1-3	PE2
4	4-6	PE1
5	7-9	PE0
6	0	PE3
7	Interrupt	PB2

B Quellcode

B.1 Testprogramm LED Ein/Aus

```
/*****************************************************************************
//BA_Experimentierboard_Versuch1
//Jan Schlüter
//17.10.2012
//LED Ein- und Ausschalten mit einem Schalter
//GPIOs
*****************************************************************************/

#include "lm3s9d92.h"


int main(void) {


  SYSCTL_RCGC2_R = 0x0000000A;        //clock Port B and Port D
  GPIO_PORTB_DEN_R |= 0x10;           //digital I/O pin PB4
  GPIO_PORTD_DEN_R |= 0x01;           //digital I/O pin PD0

  GPIO_PORTB_DIR_R &= ~0x10;          //PB4 input  (switch)
  GPIO_PORTD_DIR_R |= 0x01;           //PD0 output (LED)


  while(1){
    //Polling  PB4 input, if PB4=0 -> LED off
    if(((GPIO_PORTB_DATA_R & 0x10) == 0)){
      GPIO_PORTD_DATA_R &= ~0x01;
    }
    else{
      GPIO_PORTD_DATA_R |= 0x01;
    }
  }
}
```

B.2 Testprogramm Interrupt

```
/*****************************************************************************
//BA_Experimentierboard_Versuch2
//Jan Schlüter
//18.02.2013
//Interrupt LED Ein– und Ausschalten
//GPIOs, Interrupt
*****************************************************************************/


#include "lm3s9d92.h"


//Interrupt Service Routine
void myHandler(void)
{
  GPIO_PORTG_DATA_R = GPIO_PORTG_DATA_R^0x01; //Switch LED XOR
  GPIO_PORTB_ICR_R |= 0x04;                   //Interrupt status PB2 cleared
}

void main(void) {



    //GPIOs
    SYSCTL_RCGC2_R = 0x00000042;      //Unlock and clock Port B and Port G
    GPIO_PORTB_DEN_R |= 0x04;         //Enable digital I/O pin PB2
    GPIO_PORTG_DEN_R |= 0x02;         //Enable digital I/O pin PG1
    GPIO_PORTG_DIR_R |= 0x02;         //PG1 output LED pin
    GPIO_PORTB_DIR_R &= ~0x04;        //PB2 input, Interrupt source


    //Interrupt
    GPIO_PORTB_IS_R &= ~0x04;         //Sensible on edges
    GPIO_PORTB_IBE_R &= ~0x04;        //Interrupt event controlled by IEV register
    GPIO_PORTB_IEV_R |= 0x04;         //Sensible on rising edge
    GPIO_PORTB_IM_R |= 0x04;          //Interrupt unmasked send to interruptcontroller

    NVIC_PRI0_R |= 0x2000;            //Interrupt priority 1
    NVIC_EN0_R |= 0x02;               //Enables interrupt Port B


    while(1){
    }

}
```

B.3 Datei startup_ccs.c

```
/*
 * startup_ccs.c
 *
 *  Created on: 19.10.2012
 *      Author: Texas Instruments
 */


//*****************************************************************************
//
// startup_ccs.c - Startup code for use with TI's Code Composer Studio.
//
// Copyright (c) 2009-2012 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 9107 of the EK-LM3S9D92 Firmware Package.
//
//*****************************************************************************

//*****************************************************************************
//
// Forward declaration of the default fault handlers.
//
//*****************************************************************************
void ResetISR(void);
static void NmiSR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);

//*****************************************************************************
//
// External declaration for the reset handler that is to be called when the
// processor is started
//
//*****************************************************************************
extern void _c_int00(void);

//*****************************************************************************
//
// Linker variable that marks the top of the stack.
//
//*****************************************************************************
extern unsigned long __STACK_TOP;
```

```
//*****************************************************************************
//
// External declarations for the interrupt handlers used by the application.
//
//*****************************************************************************
extern void myHandler(void);


//*****************************************************************************
//
// The vector table.  Note that the proper constructs must be placed on this to
// ensure that it ends up at physical address 0x0000.0000 or at the start of
// the program if located at a start address other than 0.
//
//*****************************************************************************
#pragma DATA_SECTION(g_pfnVectors, ".intvecs")
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((unsigned long)&__STACK_TOP),
                                            // The initial stack pointer
    ResetISR,                               // The reset handler
    NmiSR,                                  // The NMI handler
    FaultISR,                               // The hard fault handler
    IntDefaultHandler,                      // The MPU fault handler
    IntDefaultHandler,                      // The bus fault handler
    IntDefaultHandler,                      // The usage fault handler
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    IntDefaultHandler,                      // SVCall handler
    IntDefaultHandler,                      // Debug monitor handler
    0,                                      // Reserved
    IntDefaultHandler,                      // The PendSV handler
    IntDefaultHandler,                      // The SysTick handler
    IntDefaultHandler,                      // GPIO Port A
    myHandler,                              // GPIO Port B
    IntDefaultHandler,                      // GPIO Port C
    IntDefaultHandler,                      // GPIO Port D
    IntDefaultHandler,                      // GPIO Port E
    IntDefaultHandler,                      // UART0 Rx and Tx
    IntDefaultHandler,                      // UART1 Rx and Tx
    IntDefaultHandler,                      // SSI0 Rx and Tx
    IntDefaultHandler,                      // I2C0 Master and Slave
    IntDefaultHandler,                      // PWM Fault
    IntDefaultHandler,                      // PWM Generator 0
    IntDefaultHandler,                      // PWM Generator 1
    IntDefaultHandler,                      // PWM Generator 2
    IntDefaultHandler,                      // Quadrature Encoder 0
    IntDefaultHandler,                      // ADC Sequence 0
    IntDefaultHandler,                      // ADC Sequence 1
    IntDefaultHandler,                      // ADC Sequence 2
    IntDefaultHandler,                      // ADC Sequence 3
    IntDefaultHandler,                      // Watchdog timer
    IntDefaultHandler,                      // Timer 0 subtimer A
    IntDefaultHandler,                      // Timer 0 subtimer B
    IntDefaultHandler,                      // Timer 1 subtimer A
    IntDefaultHandler,                      // Timer 1 subtimer B
    IntDefaultHandler,                      // Timer 2 subtimer A
    IntDefaultHandler,                      // Timer 2 subtimer B
```

```
    IntDefaultHandler,                      // Analog Comparator 0
    IntDefaultHandler,                      // Analog Comparator 1
    IntDefaultHandler,                      // Analog Comparator 2
    IntDefaultHandler,                      // System Control (PLL, OSC, BO)
    IntDefaultHandler,                      // FLASH Control
    IntDefaultHandler,                      // GPIO Port F
    IntDefaultHandler,                      // GPIO Port G
    IntDefaultHandler,                      // GPIO Port H
    IntDefaultHandler,                      // UART2 Rx and Tx
    IntDefaultHandler,                      // SSI1 Rx and Tx
    IntDefaultHandler,                      // Timer 3 subtimer A
    IntDefaultHandler,                      // Timer 3 subtimer B
    IntDefaultHandler,                      // I2C1 Master and Slave
    IntDefaultHandler,                      // Quadrature Encoder 1
    IntDefaultHandler,                      // CAN0
    IntDefaultHandler,                      // CAN1
    IntDefaultHandler,                      // CAN2
    IntDefaultHandler,                      // Ethernet
    IntDefaultHandler,                      // Hibernate
    IntDefaultHandler,                      // USB0
    IntDefaultHandler,                      // PWM Generator 3
    IntDefaultHandler,                      // uDMA Software Transfer
    IntDefaultHandler,                      // uDMA Error
    IntDefaultHandler,                      // ADC1 Sequence 0
    IntDefaultHandler,                      // ADC1 Sequence 1
    IntDefaultHandler,                      // ADC1 Sequence 2
    IntDefaultHandler,                      // ADC1 Sequence 3
    IntDefaultHandler,                      // I2S0
    IntDefaultHandler,                      // External Bus Interface 0
    IntDefaultHandler                       // GPIO Port J
};

//*****************************************************************************
//
// This is the code that gets called when the processor first starts execution
// following a reset event.  Only the absolutely necessary set is performed,
// after which the application supplied entry() routine is called.  Any fancy
// actions (such as making decisions based on the reset cause register, and
// resetting the bits in that register) are left solely in the hands of the
// application.
//
//*****************************************************************************
void
ResetISR(void)
{
    //
    // Jump to the CCS C initialization routine.
    //
    __asm("    .global _c_int00\n"
          "    b.w     _c_int00");
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a NMI.  This
// simply enters an infinite loop, preserving the system state for examination
// by a debugger.
//
//*****************************************************************************
static void
```

```
NmiSR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a fault
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
static void
FaultISR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives an unexpected
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
static void
IntDefaultHandler(void)
{
    //
    // Go into an infinite loop.
    //
    while(1)
    {
    }
}
```

B.4 Testprogramm 7-Segment-Multiplex

```
/*****************************************************************************
//BA_Experimentierboard_Versuch3
//Jan Schlüter
//23.10.2012
//Multiplex von drei 7-Segment-Anzeigen
//GPIOs, TImer, Interrupt
*****************************************************************************/

#include "lm3s9d92.h"


//Function prototypes
void setSegment(int x);

//Interrupt Service Routine
void timerHandler (void){

  static int i=0;

  TIMER1_ICR_R |= 0x01;          //Reset interrupt status

  if (i==1){
    GPIO_PORTJ_DATA_R = 0x02;    //PJ1 on, PJ2 off, PJ3 off
    setSegment(8);
  }
  else if (i==2){
    GPIO_PORTJ_DATA_R = 0x04;    //PJ1 off, PJ2 on, PJ3 off
    setSegment(5);
  }
  else{
    GPIO_PORTJ_DATA_R = 0x08;    //PJ1 off, PJ1 off, PJ3 on
    setSegment(1);
    i=0;
  }

  i++;
}


void main(void) {

  //Ein- und Ausgänge
  SYSCTL_RCGC2_R = 0x00000108;     //clock Port J and Port D

  GPIO_PORTJ_DEN_R |= 0x0E;        //Digital I/O pin PJ1-3
  GPIO_PORTD_DEN_R |= 0xFF;        //Digital I/0 pin PD0-7

  GPIO_PORTJ_DIR_R |= 0x1E;        //PJ1-4 output
  GPIO_PORTD_DIR_R |= 0xFF;        //PD0-7 output

  GPIO_PORTD_DR8R_R |= 0xFF;       //8mA current driver
  GPIO_PORTJ_DR8R_R |= 0x0E;       //8mA current driver

  GPIO_PORTD_DATA_R &= ~0xFF;      //PD off
  GPIO_PORTJ_DATA_R = 0x08;        //PJ3 on, PJ2 off, PJ1 off

  setSegment(4);
```

```
  //Timer 7-Segment

  SYSCTL_RCGC1_R |= 0x00020000;    //Timer 1 activate
  TIMER1_CTL_R &= ~0x0101;         //Timer 1 A B stop
  TIMER1_CFG_R = 0x00000000;       //Timer 32 bit mode
  TIMER1_TAMR_R  |= 0x02;          //Periodic timer mode
  TIMER1_TAILR_R = 0x0000FFFF;     //max. countervalue
  TIMER1_IMR_R|= 0x01;             //Timer 1 overflow interrupt activate
  NVIC_PRI5_R |= 0x2000;           //Interrupt priority 1
  NVIC_EN0_R |= 0x00200000;        //activates interrupt Timer 1A
  TIMER1_CTL_R |= 0x0101;          //Timer 1 A B stop start

  while (1){

  }
}


void setSegment(int x){

  GPIO_PORTD_DATA_R &= ~0xFF;              //PD off

  switch(x){
  case 0: GPIO_PORTD_DATA_R = 0xEE;break;  //PD 0
  case 1: GPIO_PORTD_DATA_R = 0x60;break;  //PD 1
  case 2: GPIO_PORTD_DATA_R = 0xCD;break;  //PD 2
  case 3: GPIO_PORTD_DATA_R = 0xE9;break;  //PD 3
  case 4: GPIO_PORTD_DATA_R = 0x63;break;  //PD 4
  case 5: GPIO_PORTD_DATA_R = 0xAB;break;  //PD 5
  case 6: GPIO_PORTD_DATA_R = 0xAF;break;  //PD 6
  case 7: GPIO_PORTD_DATA_R = 0xE0;break;  //PD 7
  case 8: GPIO_PORTD_DATA_R = 0xEF;break;  //PD 8
  case 9: GPIO_PORTD_DATA_R = 0xEB;break;  //PD 9
  default: GPIO_PORTD_DATA_R = 0x02;       //PD -
  }
}
```

B.5 Testprogramm Temperaturmessung mit dem Sensor LM35

```
/*****************************************************************************
//BA_Experimentierboard_Versuch4
//Jan Schlüter
//02.12.2012
//Temperature Messung mit Sensor LM35
//ADC, Timer, GPIOs, Interrupt
*****************************************************************************/

#include "lm3s9d92.h"

//global variables
int TEMP;


//Function prototypes
void setSegment(int x);
void setSegmentDP(int x);

//Interrupt Service Routine
void timerHandler (void){

  static int i=0,j=0;
  static int s1,s2,s3;



  TIMER1_ICR_R |= 0x01;                    //reset interrupt status

  if (i==1){
    GPIO_PORTJ_DATA_R = 0x02;              //PJ1 on, PJ2 off, PJ3 off
    setSegment(s3);
  }
  else if (i==2){
    GPIO_PORTJ_DATA_R = 0x04;              //PJ1 off, PJ2 on, PJ3 off
    setSegmentDP(s2);
  }
  else{
    GPIO_PORTJ_DATA_R = 0x08;              //PJ1 off, PJ2 off, PJ3 on
    setSegment(s1);
  }

  if (i==1){
    ADC0_PSSI_R |= 0x08;                   //Start sample sequence ADC 0
    TEMP = TEMP+((ADC0_SSFIFO3_R*7)/10);   //Read out ADC sample and calculate temperature
    j++;

  }

  if (i==3){
    i=0;
  }


  if(j==100){                              //calculate temperature average of 100 samples
    TEMP = TEMP/j;
```

```
      s1=TEMP%10;
      TEMP=TEMP/10;
      s2=TEMP%10;
      TEMP=TEMP/10;
      s3=TEMP;
      j=0;
    }

    i++;

}


void main(void) {

  SYSCTL_RCGC0_R = 0x00010000;   //Unlock clock for ADC0
  SYSCTL_RCGC2_R = 0x00000118;   //Unlock and clock Port J and PORT D and PORT E

  GPIO_PORTJ_DEN_R |= 0x0E;      //Enable digital I/O pin PJ 1-3
  GPIO_PORTD_DEN_R |= 0xFF;      //Enable digital I/0 pin PD 0-7

  GPIO_PORTJ_DIR_R |= 0x0E;      //PJ 1-3 output
  GPIO_PORTD_DIR_R |= 0xFF;      //PD 0-7 output

  GPIO_PORTD_DR8R_R |= 0xFF;     //8mA current drive
  GPIO_PORTJ_DR8R_R |= 0x0E;

  GPIO_PORTD_DATA_R &= ~0xFF;    //PD off
  GPIO_PORTJ_DATA_R = 0x08;      //PJ1 off, PJ2 off, PJ3 on

  setSegment(2);

  //ADC PE7 input

  GPIO_PORTE_DIR_R &= ~0x80;     //PE7 input
  GPIO_PORTE_DEN_R &= ~0x80;     //PE7 disable digital mode

  GPIO_PORTE_AMSEL_R |= 0x80;    //PE7 enable analog function
  GPIO_PORTE_AFSEL_R |= 0x80;    //PE7 alternative function

  ADC0_ACTSS_R &= ~0x0F;         //Disable sequencer ADC0
  ADC0_EMUX_R = 0x00;            //sample sequence 3 triggered by software
  ADC0_CTL_R = 0x10;             //12 bit resolution Vref=3v intern
  ADC0_SSMUX3_R = 0x00;          //1st sample select
  ADC0_SSCTL3_R = 0x00;          //1st sample => end of sequence
  ADC0_ACTSS_R |= 0x08;          //Enable sequencer 3 ADC0



  //Timer
  SYSCTL_RCGC1_R |= 0x00020000; //Timer 1 unlock
  TIMER1_CTL_R &= ~0x0101;      //Timer 1 A B stop
  TIMER1_CFG_R = 0x00000000;    //Timer 32 bit mode
  TIMER1_TAMR_R  |= 0x02;       //Periodic mode selected
  TIMER1_TAILR_R = 0x0000FFFF; //Stop value of counter
  TIMER1_IMR_R|= 0x01;          //Timer 1 overflow interrupt enable
  NVIC_PRI5_R |= 0x2000;        //Interrupt priority 1
  NVIC_EN0_R |= 0x00200000;     //Enables interrupt Timer 1A
  TIMER1_CTL_R |= 0x0101;       //Timer 1 A B start
```

```
  while (1){

  }
}

//numbers
void setSegment(int x){

  GPIO_PORTD_DATA_R &= ~0xFF;                    //PD off

  switch(x){
    case 0: GPIO_PORTD_DATA_R = 0xEE;break;     //PD 0
    case 1: GPIO_PORTD_DATA_R = 0x60;break;     //PD 1
    case 2: GPIO_PORTD_DATA_R = 0xCD;break;     //PD 2
    case 3: GPIO_PORTD_DATA_R = 0xE9;break;     //PD 3
    case 4: GPIO_PORTD_DATA_R = 0x63;break;     //PD 4
    case 5: GPIO_PORTD_DATA_R = 0xAB;break;     //PD 5
    case 6: GPIO_PORTD_DATA_R = 0xAF;break;     //PD 6
    case 7: GPIO_PORTD_DATA_R = 0xE0;break;     //PD 7
    case 8: GPIO_PORTD_DATA_R = 0xEF;break;     //PD 8
    case 9: GPIO_PORTD_DATA_R = 0xEB;break;     //PD 9
    default: GPIO_PORTD_DATA_R = 0x02;          //PD -

  }


}

//numbers with dotpoint
void setSegmentDP(int x){

  GPIO_PORTD_DATA_R &= ~0xFF;                    //PD off

  switch(x){
    case 0: GPIO_PORTD_DATA_R = 0xFE;break;     //PD 0
    case 1: GPIO_PORTD_DATA_R = 0x70;break;     //PD 1
    case 2: GPIO_PORTD_DATA_R = 0xDD;break;     //PD 2
    case 3: GPIO_PORTD_DATA_R = 0xF9;break;     //PD 3
    case 4: GPIO_PORTD_DATA_R = 0x73;break;     //PD 4
    case 5: GPIO_PORTD_DATA_R = 0xBB;break;     //PD 5
    case 6: GPIO_PORTD_DATA_R = 0xBF;break;     //PD 6
    case 7: GPIO_PORTD_DATA_R = 0xF0;break;     //PD 7
    case 8: GPIO_PORTD_DATA_R = 0xFF;break;     //PD 8
    case 9: GPIO_PORTD_DATA_R = 0xFB;break;     //PD 9
    default: GPIO_PORTD_DATA_R = 0x12;          //PD -

    }
}
```

B.6 Testprogramm Temperaturmessung Sensor DS1820

```
/*****************************************************************************
//BA_Experimentierboard_Versuch5
//Jan Schlüter
//03.11.2012
//Temperaturmessung mit dem Sensor DS1820
//Timer 1, Timer 2, SysClock 80MHz, Interrupt
*****************************************************************************/

#include "lm3s9d92.h"

int TEMP;
int T;


//Function prototypes
void setSegment(int x);
void setSegmentDP(int x);
int reset_presence(void);        //Reset sequence waiting for presence
void writeBit(int bit);          //Send bit
int readBit(void);               //Read bit
void writeByte(int data);        //Send byte
int readByte(void);              //Read byte
void measureTemp(void);          //Start measurement
int readTemp(void);              //Temperature reading from DS1820
void timerwait(int time);        //Wait, transfer value time in us

//Interrupt Service Routine
void timerHandler (void){

  static int i=0,j=0;
  static int s1,s2,s3;

  TIMER1_ICR_R |= 0x01;          //Reset interrupt status

  if (i==0){
    GPIO_PORTJ_DATA_R = 0x02;    //PJ1 on, PJ2 off, PJ3 off
    setSegment(s3);
    i++;
  }
  else if (i==1){
    GPIO_PORTJ_DATA_R = 0x04;    //PJ1 off, PJ2 on, PJ3 off
    setSegmentDP(s2);
    i++;
  }
  else{
    GPIO_PORTJ_DATA_R = 0x08;    //PJ1 off, PJ2 off, PJ3 on
    setSegment(s1);
    i=0;
  }

  j++;

  if(j==100){
    measureTemp();               //Start temperature measurement

  }
```

```
  if(j==200){
    TEMP=readTemp();              //Read out DS1820
    s1=TEMP%10;
    TEMP=TEMP/10;
    s2=TEMP%10;
    TEMP=TEMP/10;
    s3=TEMP;
    j=0;
  }

}


void main(void) {

  //set SysClock up to 80MHz
  SYSCTL_RCC_R |= 0x00000800;       //Bypass on
  SYSCTL_RCC_R &= ~0x00400000;      //Disable sysdiv
  SYSCTL_RCC_R |= 0x00002000;       //Disable PLL
  SYSCTL_MISC_R |= 0x40;            //Clear PLL raw interrupt
  SYSCTL_RCC_R &= ~0x000007C0;      //Reset xtal
  SYSCTL_RCC_R |= 0x00000540;       //OSC 16MHz
  SYSCTL_RCC2_R |= 0x00000800;      //Bypass 2 on
  SYSCTL_RCC2_R |= 0x80000000;      //Use RCC2
  SYSCTL_RCC2_R |= 0x40000000;      //Div 400 off
  SYSCTL_RCC2_R &= ~0x1FC00000;     //Clear sysdiv2
  SYSCTL_RCC2_R |= 0x01000000;      //Sysdiv /5 => 80MHz
  SYSCTL_RCC2_R &= ~0x00000070;     //Clear OSC Source
  SYSCTL_RCC2_R |= 0x00000010;      //OSC Source precision internal oscillator
  SYSCTL_RCC2_R &= ~0x00002000;     //Enable PLL
  SYSCTL_RCC_R |= 0x00400000;       //USESYSDIV enable

  while((SYSCTL_RIS_R & 0x40)==0){  //Wait for PLL Lock
    }

  SYSCTL_RCC2_R &= ~0x00000800;     //Bypass off

  //GPIOs
  SYSCTL_RCGC2_R = 0x00000108;      //Unlock and clock Port J and PORT D

  GPIO_PORTJ_DEN_R |= 0x1E;         //Enable digital I/O pin PJ 1-4
  GPIO_PORTD_DEN_R |= 0xFF;         //Enable digital I/0 pin PD 0-7

  GPIO_PORTJ_DIR_R |= 0x1E;         //PJ1-4 output
  GPIO_PORTD_DIR_R |= 0xFF;         //PD0-7 output

  GPIO_PORTD_DR8R_R |= 0xFF;        //8mA current drive
  GPIO_PORTJ_DR8R_R |= 0x0E;

  GPIO_PORTD_DATA_R &= ~0xFF;       //PD off
  GPIO_PORTJ_DATA_R = 0x08;         //PJ1 off, PJ2 off, PJ3 on
  GPIO_PORTJ_DATA_R |= 0x10;        //PJ4 high

  //Timer 1
  SYSCTL_RCGC1_R |= 0x00020000;     //Timer 1 unlock
  TIMER1_CTL_R &= ~0x0101;          //Timer 1 A B stop
  TIMER1_CFG_R = 0x00000000;        //Timer 32 bit mode
```

```
  TIMER1_TAMR_R   |= 0x02;            //Periodic mode selected
  TIMER1_TAILR_R = 0x0004FFFF;       //Max. value of counter
  TIMER1_IMR_R|= 0x01;               //Timer 1 overflow Interrupt enable
  NVIC_PRI5_R |= 0x2000;             //Interrupt priority 1
  NVIC_EN0_R |= 0x00200000;          //Enables interrupt Timer 1A
  TIMER1_CTL_R |= 0x0101;            //Timer 1 A B stop start

  //Timer 2
  SYSCTL_RCGC1_R |= 0x00040000;      //Timer 2 unlock
  TIMER2_CTL_R &= ~0x0101;           //Timer 2 A B stop
  TIMER2_CFG_R = 0x00000000;         //Timer 2 32 bit mode
  TIMER2_TAMR_R   |= 0x01;           //One shot mode mode selected


  while (1){

  }
}


void setSegment(int x){

  GPIO_PORTD_DATA_R &= ~0xFF;                  //PD off

    switch(x){
      case 0: GPIO_PORTD_DATA_R = 0xEE;break;  //PD 0
      case 1: GPIO_PORTD_DATA_R = 0x60;break;  //PD 1
      case 2: GPIO_PORTD_DATA_R = 0xCD;break;  //PD 2
      case 3: GPIO_PORTD_DATA_R = 0xE9;break;  //PD 3
      case 4: GPIO_PORTD_DATA_R = 0x63;break;  //PD 4
      case 5: GPIO_PORTD_DATA_R = 0xAB;break;  //PD 5
      case 6: GPIO_PORTD_DATA_R = 0xAF;break;  //PD 6
      case 7: GPIO_PORTD_DATA_R = 0xE0;break;  //PD 7
      case 8: GPIO_PORTD_DATA_R = 0xEF;break;  //PD 8
      case 9: GPIO_PORTD_DATA_R = 0xEB;break;  //PD 9
      default: GPIO_PORTD_DATA_R = 0x02;       //PD -
  }
}

void setSegmentDP(int x){

  GPIO_PORTD_DATA_R &= ~0xFF;                 //PD off

  switch(x){
    case 0: GPIO_PORTD_DATA_R = 0xFE;break;   //PD 0
    case 1: GPIO_PORTD_DATA_R = 0x70;break;   //PD 1
    case 2: GPIO_PORTD_DATA_R = 0xDD;break;   //PD 2
    case 3: GPIO_PORTD_DATA_R = 0xF9;break;   //PD 3
    case 4: GPIO_PORTD_DATA_R = 0x73;break;   //PD 4
    case 5: GPIO_PORTD_DATA_R = 0xBB;break;   //PD 5
    case 6: GPIO_PORTD_DATA_R = 0xBF;break;   //PD 6
    case 7: GPIO_PORTD_DATA_R = 0xF0;break;   //PD 7
    case 8: GPIO_PORTD_DATA_R = 0xFF;break;   //PD 8
    case 9: GPIO_PORTD_DATA_R = 0xFB;break;   //PD 9
    default: GPIO_PORTD_DATA_R = 0x12;        //PD -
  }
}

int readTemp(void){
  int byte0,byte1,result,lock=1;
```

```
  lock=reset_presence();
  if(lock==0){
    writeByte(0xCC);                //Send skip ROM command
    writeByte(0xBE);                //Send read Scratchpad command
    byte0=readByte();
    byte1=readByte();

  }
  T=byte0;
  result = byte0;
  return ((10*result)/2);           //Calculate temperature out of register value
}


void measureTemp(void){
  int lock=1;

  lock=reset_presence();
  if (lock==0){
    writeByte(0xCC);                //Send skip ROM command
    writeByte(0x44);                //Send Convert T command
  }

}

void timerwait(int time){

  int timerload;

  timerload = time*82;              //Timer max value in tics
  TIMER2_TAILR_R = timerload;       //Timer stop value
  TIMER2_CTL_R |= 0x0101;           //Timer 2 A B start
  while((TIMER2_RIS_R & 0x01)==0){  //Wait for timer reach stop value
  }
  TIMER2_ICR_R |= 0x01;             //Reset timer raw interrupt
  TIMER2_CTL_R &= ~0x0101;          //Timer 2 A B stop

}


int reset_presence(void){
  int result;
  GPIO_PORTJ_DIR_R |= 0x10;         //Bus output
  GPIO_PORTJ_DATA_R &= ~0x10;       //Bus low
  timerwait(480);                   //Wait 480us
  GPIO_PORTJ_DATA_R |= 0x10;        //Bus high
  timerwait(70);                    //Wait 70us
  GPIO_PORTJ_DIR_R &= ~0x10;        //Bus input
  result = (GPIO_PORTJ_DATA_R & 0x10)>>4;     //Result = 0->presence, 1-> no presence
  timerwait(400);                   //Wait 410us
  GPIO_PORTJ_DIR_R |= 0x10;         //Bus output
  return result;
}

void writeBit(int bit){
  GPIO_PORTJ_DIR_R |= 0x01;         //Bus output

  if(bit==1){                       //Write '1'
    GPIO_PORTJ_DATA_R &= ~0x10;     //Bus low
```

```
    timerwait(6);                    //Wait 6us
    GPIO_PORTJ_DATA_R |= 0x10;       //Bus high
    timerwait(64);                   //Wait 64us
  }
  else{                              //Write '0'
    GPIO_PORTJ_DATA_R &= ~0x10;      //Bus low
    timerwait(60);                   //wait 60us
    GPIO_PORTJ_DATA_R |= 0x10;       //Bus high
    timerwait(10);                   //wait 10us
  }
}


int readBit(void){
  int result;
  GPIO_PORTJ_DIR_R |= 0x10;          //Bus output
  GPIO_PORTJ_DATA_R &= ~0x10;        //Bus low
  timerwait(6);                      //Wait 6us
  GPIO_PORTJ_DATA_R |= 0x10;         //Bus high
  timerwait(9);                      //Wait 9us
  GPIO_PORTJ_DIR_R &= ~0x10;         //Bus input
  result = (GPIO_PORTJ_DATA_R & 0x10)>>4;     //Read bit
  timerwait(55);                     //Wait 55us
  return result;
}

void writeByte(int data){            //Write one bit to Bus
  int loop;

  for(loop=0;loop<8;loop++){
    writeBit(data & 0x01);           //Send LSB
    data >>= 1;                      //Shift data

  }
}

int readByte(void){                 //Read one byte from Bus
  int loop, result;

  for(loop=0;loop<8;loop++){
    result >>= 1;                   //Shift result
    if (readBit()==1){
      result |= 0x80;
    }
  }
  return result;
}
```

B.7 LC Display

B.7.1 Testprogramm LC Display

```
  /*****************************************************************************
//BA_Experimentierboard_Versuch6
//Jan Schlüter
//17.10.2012
//Ansteuerung eines 128x64 LCD
//GPIOs, Librarys
*****************************************************************************/

#include "lm3s9d92.h"
#include "font.h"
#include "pic.h"
#include "numbers.h"

//Function prototypes
void init_disp(void);
void send_data(int data);
void send_command(int data);
void send_pixel_byte(int data, int page, int column);
void lcd_invers(int x);
void lcd_clear(void);
void send_font(int fig, int page, int column);
void lcd_pic();
void send_num(int num, int page, int column);
void wait(int x);

void main(void) {


    SYSCTL_RCGC2_R = 0x000000C8;   //Unlock and clock Port H and PORT D and PORT G

    GPIO_PORTH_DEN_R |= 0xE7;      //Enable digital I/O pin PH0-2 and PH5-7
    GPIO_PORTD_DEN_R |= 0xFF;      //Enable digital I/0 pin PD0-7
    GPIO_PORTG_DEN_R |= 0x02;      //Enable digital I/O in PG1

    GPIO_PORTH_DIR_R |= 0xE7;      //PH0-2 and PH5-7 output
    GPIO_PORTD_DIR_R |= 0xFF;      //PD0-7 output
    GPIO_PORTG_DIR_R |= 0x02;      //PG1 output

    GPIO_PORTG_DATA_R |=0x02;      //RES high
    GPIO_PORTH_DATA_R = 0x00;      //Port H low
    GPIO_PORTH_DATA_R |= 0x80;     //CS high

    init_disp();
    lcd_clear();
    //send_pixel_byte(0xFF,0,0);
    //send_pixel_byte(0xFF,7,127);
    //send_font('a',2,10);
    lcd_pic();
    //send_num(1,2,5);
    //send_num(3,2,31);
    //send_num(9,2,62);
    //send_num(7,2,94);
    while(1){
```

```
    }

}

void send_data(int data){
  GPIO_PORTH_DATA_R &= ~0x01;       //R/W Low
  GPIO_PORTD_DATA_R = data;         //Load data
  GPIO_PORTH_DATA_R &= ~0x80;       //Chip Select low
  GPIO_PORTH_DATA_R |= 0x02;        //E high
  wait(3);                  //Wait
  GPIO_PORTH_DATA_R &= ~0x02;       //E low
  GPIO_PORTH_DATA_R |= 0x80;        //Chip select high
}

void send_command(int data){
  GPIO_PORTH_DATA_R &= ~0x04;       //A0 Low command mode
  send_data(data);                  //Send data
  GPIO_PORTH_DATA_R |= 0x04;        //A0 high data mode
}

void send_pixel_byte(int data, int page, int column){
  send_command(0xB0+page);          //Set page
  send_command(0x10+((column&0xF0)>>4));  //MSB column adress
  send_command(0x00+(column&0x0F));       //LSB column adress
  GPIO_PORTH_DATA_R |= 0x04;        //A0 high data mode
  send_data(data);                  //Send data
  GPIO_PORTH_DATA_R &= ~0x04;       //A0 Low command mode
}

void init_disp(void){
  wait(60000);              //Wait
  GPIO_PORTH_DATA_R &= ~0x80;       //Chip select low
  GPIO_PORTH_DATA_R |= 0x40;        //P/S high parallel mode
  GPIO_PORTH_DATA_R |= 0x20;        //C86 high 6800 mode
  GPIO_PORTG_DATA_R &= ~0x02;       //RES low
  wait(10000);
  GPIO_PORTG_DATA_R |= 0x02;        //RES high
  wait(10000);
  send_command(0xE2);               //Internal reset
  send_command(0xA0);               //ADC mode normal
  send_command(0xC0);               //Output mode normal
  send_command(0xA2);               //LCD bias 1/9 ST7565R
  send_command(0x2F);               //Power supply mode
  send_command(0x27);               //Resistor ratio (brightness steilheit verstärkung)
  send_command(0xF8);               //Booster ratio
  send_command(0x00);               //Booster ratio 4x (Beschaltung Kondensatoren)
  send_command(0x81);               //Set output voltage
  send_command(0x00);               //Set output voltage (brightness start 0=8.4V ---> 63=5.1V)
  //send_command(0xE0);             //Read modify write
  send_command(0xAF);               //Display on
}

//Writes LCD pixel 0
void lcd_clear(void){
  int i,j;
  for( i=0;i<8;i++){
    for(j=0;j<128;j++){
      send_pixel_byte(0x00,i,j);
    }
```

```
  }
}

//Write picture from header file to LCD ROM
void lcd_pic(){
  int i,j,k=0;
  for( i=0;i<8;i++){
    for(j=0;j<128;j++){
      send_pixel_byte(pic4[k],i,j);
      k++;
    }
  }
}

//Send ASCII-Sign from header file to LCD ROM
void send_font(int fig, int page, int column){
  int i;
  for(i=0;i<5;i++){
    send_pixel_byte(font[(fig*5)+i],page,(column+i));
  }


}

//Send number from header file to LCD ROM
void send_num(int num, int page, int column){
  int i,j,k=0;

  for(i=page;i<(page+4);i++){
    for(j=column;j<(column+26);j++){
      send_pixel_byte(number[(num*104)+k],i,j);
      k++;
    }
  }
}

//Set hole display black or white
//Transfer parameter: x=1 black, x=0 white background
void lcd_invers(int x){
  send_command(0xA4+x);
}

void wait(int x){
  int i;
  for(i=0;i<x;i++){

  }
}
```

B.7.2 Headerdatei font.h

```
int font[] = {
0x00, 0x00, 0x00, 0x00, 0x00, // Ascii 0
0x7C, 0xDA, 0xF2, 0xDA, 0x7C, //ASC(01)
0x7C, 0xD6, 0xF2, 0xD6, 0x7C, //ASC(02)
0x38, 0x7C, 0x3E, 0x7C, 0x38,
0x18, 0x3C, 0x7E, 0x3C, 0x18,
0x38, 0xEA, 0xBE, 0xEA, 0x38,
0x38, 0x7A, 0xFE, 0x7A, 0x38,
0x0, 0x18, 0x3C, 0x18, 0x0,
0xFF, 0xE7, 0xC3, 0xE7, 0xFF,
0x0, 0x18, 0x24, 0x18, 0x0,
0xFF, 0xE7, 0xDB, 0xE7, 0xFF,
0xC, 0x12, 0x5C, 0x60, 0x70,
0x64, 0x94, 0x9E, 0x94, 0x64,
0x2, 0xFE, 0xA0, 0xA0, 0xE0,
0x2, 0xFE, 0xA0, 0xA4, 0xFC,
0x5A, 0x3C, 0xE7, 0x3C, 0x5A,
0xFE, 0x7C, 0x38, 0x38, 0x10,
0x10, 0x38, 0x38, 0x7C, 0xFE,
0x28, 0x44, 0xFE, 0x44, 0x28,
0xFA, 0xFA, 0x0, 0xFA, 0xFA,
0x60, 0x90, 0xFE, 0x80, 0xFE,
0x0, 0x66, 0x91, 0xA9, 0x56,
0x6, 0x6, 0x6, 0x6, 0x6,
0x29, 0x45, 0xFF, 0x45, 0x29,
0x10, 0x20, 0x7E, 0x20, 0x10,
0x8, 0x4, 0x7E, 0x4, 0x8,
0x10, 0x10, 0x54, 0x38, 0x10,
0x10, 0x38, 0x54, 0x10, 0x10,
0x78, 0x8, 0x8, 0x8, 0x8,
0x30, 0x78, 0x30, 0x78, 0x30,
0xC, 0x1C, 0x7C, 0x1C, 0xC,
0x60, 0x70, 0x7C, 0x70, 0x60,
0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0xFA, 0x0, 0x0,
0x0, 0xE0, 0x0, 0xE0, 0x0,
0x28, 0xFE, 0x28, 0xFE, 0x28,
0x24, 0x54, 0xFE, 0x54, 0x48,
0xC4, 0xC8, 0x10, 0x26, 0x46,
0x6C, 0x92, 0x6A, 0x4, 0xA,
0x0, 0x10, 0xE0, 0xC0, 0x0,
0x0, 0x38, 0x44, 0x82, 0x0,
0x0, 0x82, 0x44, 0x38, 0x0,
0x54, 0x38, 0xFE, 0x38, 0x54,
0x10, 0x10, 0x7C, 0x10, 0x10,
0x0, 0x1, 0xE, 0xC, 0x0,
0x10, 0x10, 0x10, 0x10, 0x10,
0x0, 0x0, 0x6, 0x6, 0x0,
0x4, 0x8, 0x10, 0x20, 0x40,
0x7C, 0x8A, 0x92, 0xA2, 0x7C,
0x0, 0x42, 0xFE, 0x2, 0x0,
0x4E, 0x92, 0x92, 0x92, 0x62,
0x84, 0x82, 0x92, 0xB2, 0xCC,
0x18, 0x28, 0x48, 0xFE, 0x8,
0xE4, 0xA2, 0xA2, 0xA2, 0x9C,
0x3C, 0x52, 0x92, 0x92, 0x8C,
0x82, 0x84, 0x88, 0x90, 0xE0,
0x6C, 0x92, 0x92, 0x92, 0x6C,
```

```
0x62, 0x92, 0x92, 0x94, 0x78,
0x0, 0x0, 0x28, 0x0, 0x0,
0x0, 0x2, 0x2C, 0x0, 0x0,
0x0, 0x10, 0x28, 0x44, 0x82,
0x28, 0x28, 0x28, 0x28, 0x28,
0x0, 0x82, 0x44, 0x28, 0x10,
0x40, 0x80, 0x9A, 0x90, 0x60,
0x7C, 0x82, 0xBA, 0x9A, 0x72,
0x3E, 0x48, 0x88, 0x48, 0x3E,
0xFE, 0x92, 0x92, 0x92, 0x6C,
0x7C, 0x82, 0x82, 0x82, 0x44,
0xFE, 0x82, 0x82, 0x82, 0x7C,
0xFE, 0x92, 0x92, 0x92, 0x82,
0xFE, 0x90, 0x90, 0x90, 0x80,
0x7C, 0x82, 0x82, 0x8A, 0xCE,
0xFE, 0x10, 0x10, 0x10, 0xFE,
0x0, 0x82, 0xFE, 0x82, 0x0,
0x4, 0x2, 0x82, 0xFC, 0x80,
0xFE, 0x10, 0x28, 0x44, 0x82,
0xFE, 0x2, 0x2, 0x2, 0x2,
0xFE, 0x40, 0x38, 0x40, 0xFE,
0xFE, 0x20, 0x10, 0x8, 0xFE,
0x7C, 0x82, 0x82, 0x82, 0x7C,
0xFE, 0x90, 0x90, 0x90, 0x60,
0x7C, 0x82, 0x8A, 0x84, 0x7A,
0xFE, 0x90, 0x98, 0x94, 0x62,
0x64, 0x92, 0x92, 0x92, 0x4C,
0xC0, 0x80, 0xFE, 0x80, 0xC0,
0xFC, 0x2, 0x2, 0x2, 0xFC,
0xF8, 0x4, 0x2, 0x4, 0xF8,
0xFC, 0x2, 0x1C, 0x2, 0xFC,
0xC6, 0x28, 0x10, 0x28, 0xC6,
0xC0, 0x20, 0x1E, 0x20, 0xC0,
0x86, 0x9A, 0x92, 0xB2, 0xC2,
0x0, 0xFE, 0x82, 0x82, 0x82,
0x40, 0x20, 0x10, 0x8, 0x4,
0x0, 0x82, 0x82, 0x82, 0xFE,
0x20, 0x40, 0x80, 0x40, 0x20,
0x2, 0x2, 0x2, 0x2, 0x2,
0x0, 0xC0, 0xE0, 0x10, 0x0,
0x4, 0x2A, 0x2A, 0x1E, 0x2,
0xFE, 0x14, 0x22, 0x22, 0x1C,
0x1C, 0x22, 0x22, 0x22, 0x14,
0x1C, 0x22, 0x22, 0x14, 0xFE,
0x1C, 0x2A, 0x2A, 0x2A, 0x18,
0x0, 0x10, 0x7E, 0x90, 0x40,
0x18, 0x25, 0x25, 0x39, 0x1E,
0xFE, 0x10, 0x20, 0x20, 0x1E,
0x0, 0x22, 0xBE, 0x2, 0x0,
0x4, 0x2, 0x2, 0xBC, 0x0,
0xFE, 0x8, 0x14, 0x22, 0x0,
0x0, 0x82, 0xFE, 0x2, 0x0,
0x3E, 0x20, 0x1E, 0x20, 0x1E,
0x3E, 0x10, 0x20, 0x20, 0x1E,
0x1C, 0x22, 0x22, 0x22, 0x1C,
0x3F, 0x18, 0x24, 0x24, 0x18,
0x18, 0x24, 0x24, 0x18, 0x3F,
0x3E, 0x10, 0x20, 0x20, 0x10,
0x12, 0x2A, 0x2A, 0x2A, 0x24,
0x20, 0x20, 0xFC, 0x22, 0x24,
```

```
0x3C, 0x2, 0x2, 0x4, 0x3E,
0x38, 0x4, 0x2, 0x4, 0x38,
0x3C, 0x2, 0xC, 0x2, 0x3C,
0x22, 0x14, 0x8, 0x14, 0x22,
0x32, 0x9, 0x9, 0x9, 0x3E,
0x22, 0x26, 0x2A, 0x32, 0x22,
0x0, 0x10, 0x6C, 0x82, 0x0,
0x0, 0x0, 0xEE, 0x0, 0x0,
0x0, 0x82, 0x6C, 0x10, 0x0,
0x40, 0x80, 0x40, 0x20, 0x40,
0x3C, 0x64, 0xC4, 0x64, 0x3C,
0x78, 0x85, 0x85, 0x86, 0x48,
0x5C, 0x2, 0x2, 0x4, 0x5E,
0x1C, 0x2A, 0x2A, 0xAA, 0x9A,
0x84, 0xAA, 0xAA, 0x9E, 0x82,
0x84, 0x2A, 0x2A, 0x1E, 0x82,
0x84, 0xAA, 0x2A, 0x1E, 0x2,
0x4, 0x2A, 0xAA, 0x9E, 0x2,
0x30, 0x78, 0x4A, 0x4E, 0x48,
0x9C, 0xAA, 0xAA, 0xAA, 0x9A,
0x9C, 0x2A, 0x2A, 0x2A, 0x9A,
0x9C, 0xAA, 0x2A, 0x2A, 0x1A,
0x0, 0x0, 0xA2, 0x3E, 0x82,
0x0, 0x40, 0xA2, 0xBE, 0x42,
0x0, 0x80, 0xA2, 0x3E, 0x2,
0xF, 0x94, 0x24, 0x94, 0xF,
0xF, 0x14, 0xA4, 0x14, 0xF,
0x3E, 0x2A, 0xAA, 0xA2, 0x0,
0x4, 0x2A, 0x2A, 0x3E, 0x2A,
0x3E, 0x50, 0x90, 0xFE, 0x92,
0x4C, 0x92, 0x92, 0x92, 0x4C,
0x4C, 0x12, 0x12, 0x12, 0x4C,
0x4C, 0x52, 0x12, 0x12, 0xC,
0x5C, 0x82, 0x82, 0x84, 0x5E,
0x5C, 0x42, 0x2, 0x4, 0x1E,
0x0, 0xB9, 0x5, 0x5, 0xBE,
0x9C, 0x22, 0x22, 0x22, 0x9C,
0xBC, 0x2, 0x2, 0x2, 0xBC,
0x3C, 0x24, 0xFF, 0x24, 0x24,
0x12, 0x7E, 0x92, 0xC2, 0x66,
0xD4, 0xF4, 0x3F, 0xF4, 0xD4,
0xFF, 0x90, 0x94, 0x6F, 0x4,
0x3, 0x11, 0x7E, 0x90, 0xC0,
0x4, 0x2A, 0x2A, 0x9E, 0x82,
0x0, 0x0, 0x22, 0xBE, 0x82,
0xC, 0x12, 0x12, 0x52, 0x4C,
0x1C, 0x2, 0x2, 0x44, 0x5E,
0x0, 0x5E, 0x50, 0x50, 0x4E,
0xBE, 0xB0, 0x98, 0x8C, 0xBE,
0x64, 0x94, 0x94, 0xF4, 0x14,
0x64, 0x94, 0x94, 0x94, 0x64,
0xC, 0x12, 0xB2, 0x2, 0x4,
0x1C, 0x10, 0x10, 0x10, 0x10,
0x10, 0x10, 0x10, 0x10, 0x1C,
0xF4, 0x8, 0x13, 0x35, 0x5D,
0xF4, 0x8, 0x14, 0x2C, 0x5F,
0x0, 0x0, 0xDE, 0x0, 0x0,
0x10, 0x28, 0x54, 0x28, 0x44,
0x44, 0x28, 0x54, 0x28, 0x10,
0x55, 0x0, 0xAA, 0x0, 0x55,
```

```
0x55, 0xAA, 0x55, 0xAA, 0x55,
0xAA, 0x55, 0xAA, 0x55, 0xAA,
0x0, 0x0, 0x0, 0xFF, 0x0,
0x8, 0x8, 0x8, 0xFF, 0x0,
0x28, 0x28, 0x28, 0xFF, 0x0,
0x8, 0x8, 0xFF, 0x0, 0xFF,
0x8, 0x8, 0xF, 0x8, 0xF,
0x28, 0x28, 0x28, 0x3F, 0x0,
0x28, 0x28, 0xEF, 0x0, 0xFF,
0x0, 0x0, 0xFF, 0x0, 0xFF,
0x28, 0x28, 0x2F, 0x20, 0x3F,
0x28, 0x28, 0xE8, 0x8, 0xF8,
0x8, 0x8, 0xF8, 0x8, 0xF8,
0x28, 0x28, 0x28, 0xF8, 0x0,
0x8, 0x8, 0x8, 0xF, 0x0,
0x0, 0x0, 0x0, 0xF8, 0x8,
0x8, 0x8, 0x8, 0xF8, 0x8,
0x8, 0x8, 0x8, 0xF, 0x8,
0x0, 0x0, 0x0, 0xFF, 0x8,
0x8, 0x8, 0x8, 0x8, 0x8,
0x8, 0x8, 0x8, 0xFF, 0x8,
0x0, 0x0, 0x0, 0xFF, 0x28,
0x0, 0x0, 0xFF, 0x0, 0xFF,
0x0, 0x0, 0xF8, 0x8, 0xE8,
0x0, 0x0, 0x3F, 0x20, 0x2F,
0x28, 0x28, 0xE8, 0x8, 0xE8,
0x28, 0x28, 0x2F, 0x20, 0x2F,
0x0, 0x0, 0xFF, 0x0, 0xEF,
0x28, 0x28, 0x28, 0x28, 0x28,
0x28, 0x28, 0xEF, 0x0, 0xEF,
0x28, 0x28, 0x28, 0xE8, 0x28,
0x8, 0x8, 0xF8, 0x8, 0xF8,
0x28, 0x28, 0x28, 0x2F, 0x28,
0x8, 0x8, 0xF, 0x8, 0xF,
0x0, 0x0, 0xF8, 0x8, 0xF8,
0x0, 0x0, 0x0, 0xF8, 0x28,
0x0, 0x0, 0x0, 0x3F, 0x28,
0x0, 0x0, 0xF, 0x8, 0xF,
0x8, 0x8, 0xFF, 0x8, 0xFF,
0x28, 0x28, 0x28, 0xFF, 0x28,
0x8, 0x8, 0x8, 0xF8, 0x0,
0x0, 0x0, 0x0, 0xF, 0x8,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xF, 0xF, 0xF, 0xF, 0xF,
0xFF, 0xFF, 0xFF, 0x0, 0x0,
0x0, 0x0, 0x0, 0xFF, 0xFF,
0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
0x1C, 0x22, 0x22, 0x1C, 0x22,
0x3E, 0x54, 0x54, 0x7C, 0x28,
0x7E, 0x40, 0x40, 0x60, 0x60,
0x40, 0x7E, 0x40, 0x7E, 0x40,
0xC6, 0xAA, 0x92, 0x82, 0xC6,
0x1C, 0x22, 0x22, 0x3C, 0x20,
0x2, 0x7E, 0x4, 0x78, 0x4,
0x60, 0x40, 0x7E, 0x40, 0x40,
0x99, 0xA5, 0xE7, 0xA5, 0x99,
0x38, 0x54, 0x92, 0x54, 0x38,
0x32, 0x4E, 0x80, 0x4E, 0x32,
0xC, 0x52, 0xB2, 0xB2, 0xC,
0xC, 0x12, 0x1E, 0x12, 0xC,
```

```
0x3D, 0x46, 0x5A, 0x62, 0xBC,
0x7C, 0x92, 0x92, 0x92, 0x0,
0x7E, 0x80, 0x80, 0x80, 0x7E,
0x54, 0x54, 0x54, 0x54, 0x54,
0x22, 0x22, 0xFA, 0x22, 0x22,
0x2, 0x8A, 0x52, 0x22, 0x2,
0x2, 0x22, 0x52, 0x8A, 0x2,
0x0, 0x0, 0xFF, 0x80, 0xC0,
0x7, 0x1, 0xFF, 0x0, 0x0,
0x10, 0x10, 0xD6, 0xD6, 0x10,
0x6C, 0x48, 0x6C, 0x24, 0x6C,
0x60, 0xF0, 0x90, 0xF0, 0x60,
0x0, 0x0, 0x18, 0x18, 0x0,
0x0, 0x0, 0x8, 0x8, 0x0,
0xC, 0x2, 0xFF, 0x80, 0x80,
0x0, 0xF8, 0x80, 0x80, 0x78,
0x0, 0x98, 0xB8, 0xE8, 0x48,
0x0, 0x3C, 0x3C, 0x3C, 0x3C,};
```

B.7.3 Headerdatei numbers.h

```
//numbers lib
int number[] = {
    0x00,0x80,0xE0,0xF0,0xF8,0xF8,0xFC,0xFC,0xFE,0xFE,0xFE,0x7E,0x7E,0xFE,0xFE,0xFC,0xFC,0xFC,0xF8,0
        xF8,0xF0,0xE0,0x80,0x00,0x00,0x00,   //0

    0xFC,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x00,0x00,0x00,0x01,0x03,0x7F,0xFF,0xFF,0
        xFF,0xFF,0xFF,0xFF,0xFE,0xE0,0x00,

    0x07,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0xE0,0xFF,0xFF,0
        xFF,0xFF,0xFF,0xFF,0xFF,0x3F,0x00,

    0x00,0x00,0x01,0x07,0x0F,0x1F,0x3F,0x3F,0x7F,0x7F,0x7F,0x7E,0x7E,0x7E,0x7E,0x7F,0x7F,0x7F,0x3F,0
        x3F,0x1F,0x0F,0x07,0x01,0x00,0x00,

    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0
        x00,0x00,0x00,0x00,0x00,0x00,0x00,   //1

    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x03,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x00,0x00,0x00,0x00,0x00,0x00,0x00,

    0x00,0x00,0x00,0x00,0x00,0x60,0xE0,0xC0,0xC0,0x80,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        xE8,0x00,0x00,0x00,0x00,0x00,0x00,

    0x00,0x00,0x00,0x00,0x00,0x00,0x1F,0x1F,0x3F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0
        xFF,0x00,0x00,0x00,0x00,0x00,0x00,

    0x00,0x3F,0x7E,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0x7E,0x7E,0x7E,0x7E,0x7E,0
        x7E,0x7E,0x7E,0x7E,0x7F,0x40,0x00,   //2

    0x00,0x00,0x00,0x00,0x01,0x03,0x07,0x0F,0x1F,0x3F,0x7F,0xFF,0xFF,0xFF,0xFF,0xFE,0xFC,0xF8,0xF0,0
        xE0,0xC0,0x00,0x00,0x00,0x00,0x00,

    0x00,0x00,0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x07,0x8F,0xFF,0xFF,0xFF,0xFF,0
        xFF,0xFF,0xFF,0xFE,0xFC,0xF0,0x80,

    0x00,0x00,0x00,0x0F,0x3F,0x3F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0
        x3F,0x3F,0x3F,0x1F,0x0F,0x07,0x00,

    0x00,0xFC,0xFE,0xFE,0xFE,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0xFE,0xFE,0xFE,0xFE,0xFE,0xFC,0
        xFC,0xF8,0xF8,0xF0,0xC0,0x00,0x00,   //3

    0x00,0x01,0x00,0x00,0x00,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xF1,0xFF,0xFF,0xFF,0xFF,0
        xFF,0x7F,0x7F,0x3F,0x1F,0x02,0x00,

    0x00,0x00,0x00,0x00,0x00,0x07,0x0F,0x0F,0x07,0x07,0x07,0x0F,0x0F,0x0F,0x1F,0xFF,0xFF,0xFF,0xFF,0
        xFE,0xFE,0xFC,0xFC,0xF8,0xE0,0x00,

    0x00,0x00,0x00,0x07,0x3E,0x3E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0
        x3F,0x3F,0x3F,0x1F,0x0F,0x03,0x00,

    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1E,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0
        xFE,0xFE,0x00,0x00,0x00,0x00,0x00,   //4

    0x00,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0x7E,0x7E,0x7E,0x7E,0x7E,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0
        xFF,0xFF,0xFE,0xFE,0xFE,0xFE,0x62,

    0x00,0x00,0x01,0x07,0x0F,0x3F,0x7F,0xFF,0xFE,0xFC,0xF8,0xE0,0xC0,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0
        xFF,0xFF,0x00,0x00,0x00,0x00,0x00,
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x03,0x0F,0x1F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0
    x7F,0x7F,0x7F,0x00,0x00,0x00,0x00,

0x00,0xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0xFE,0xFE,0xFE,0xFE,0xFC,0xFC,0xFC,0xFC,0xFC,0xF8,0xF8,0xF8,0
    xF0,0xE0,0xC0,0x80,0x00,0x00,0x00,   //5

0x00,0x01,0x81,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0xC1,0xC3,0xFF,0xFF,0xFF,0xFF,0xFF,0
    xFF,0xFF,0xFF,0xFF,0x1C,0x00,0x00,

0x00,0x00,0x1F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x0F,0x0F,0x0F,0x0F,0x0F,0x0F,0x07,0x07,0
    x07,0x03,0x01,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0
    x7E,0x7E,0x7E,0x7E,0x00,0x00,0x00,

0x00,0xC0,0xF0,0xF8,0xF8,0xFC,0xFC,0xFC,0xFE,0xFE,0xFE,0x7E,0x7E,0x7E,0xFE,0xFE,0xFE,0xFC,0xFC,0
    xF8,0xF8,0xF0,0xE0,0xC0,0x00,0x00,   //6

0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC1,0x00,0x40,0xC0,0xC0,0xC0,0xE1,0xFF,0xFF,0xFF,0
    xFF,0xFF,0xFF,0xFF,0xFF,0x7E,0x00,

0x00,0x03,0x0F,0x3F,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFE,0xF8,0xE1,0xC3,0x0F,0x0F,0x0F,0x0F,0x0F,0
    x0F,0x0F,0x07,0x03,0x01,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x01,0x07,0x0F,0x1F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x7E,0x7C,0x70,0x60,0
    x40,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x06,0x1E,0x7E,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFC,0xE0,0x80,0x00,0x00,0x00,0
    x00,0x00,0x00,0x00,0x00,0x00,0x00,   //7

0x00,0x00,0x00,0x00,0x00,0x01,0x07,0x1F,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFE,0xF8,0xE0,0
    x80,0x00,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x07,0x1F,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
    xFF,0xFE,0xF8,0xE0,0x00,0x00,0x00,

0x00,0x7F,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0
    x7F,0x7F,0x7F,0x7F,0x7F,0x7F,0x00,0x00,

0xC0,0xF0,0xF8,0xF8,0xFC,0xFC,0xFE,0xFE,0xFE,0xFE,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0xFE,0xFE,0xFE,0
    xFC,0xFC,0xF8,0xF0,0xE0,0xC0,0x00,   //8

0x07,0x1F,0x3F,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xF0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xF1,0xFF,0xFF,0
    xFF,0xFF,0xFF,0x7F,0x3F,0x1F,0x00,

0x00,0xC0,0xF8,0xFC,0xFE,0xFE,0xFF,0xFF,0xFF,0x1F,0x0F,0x07,0x07,0x07,0x07,0x0F,0x1F,0xFF,0xFF,0
    xFF,0xFE,0xFC,0xFC,0xF0,0x80,0x00,

0x00,0x00,0x07,0x0F,0x1F,0x3F,0x3F,0x7F,0x7F,0x7F,0x7E,0x7E,0x7E,0x7E,0x7E,0x7E,0x7F,0x7F,0x7F,0
    x3F,0x3F,0x1F,0x1F,0x07,0x00,0x00,

0x00,0x00,0x00,0x00,0x00,0x02,0x06,0x0E,0x1E,0x7E,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFC,0xF0,0xE0,0
    x80,0x00,0x00,0x00,0x00,0x00,0x00,   //9

0x00,0x80,0xC0,0xE0,0xF0,0xF0,0xF0,0xF8,0xF8,0xF8,0xF0,0xC3,0x07,0x1F,0x7F,0xFF,0xFF,0xFF,0xFF,0
    xFF,0xFF,0xFC,0xF0,0xC0,0x00,0x00,

0x08,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC7,0x03,0x01,0x01,0x00,0x00,0x81,0xFF,0xFF,0xFF,0
    xFF,0xFF,0xFF,0xFF,0xFF,0xFE,0x00,
```

```
    0x00,0x00,0x03,0x07,0x0F,0x1F,0x3F,0x3F,0x3F,0x7F,0x7F,0x7F,0x7E,0x7E,0x7F,0x7F,0x7F,0x7F,0x3F,0
        x3F,0x1F,0x1F,0x0F,0x07,0x00,0x00,

};
```

B.7.4 Headerdatei pic.h

```
//the walking dead
int pic4 [] ={
    0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x7F,0
        x3F,0x3F,0x3F,0x3F,0x3F,0x7F,0x7F,0x7F,0x7F,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0
        x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x7F,0xFF,0xFF,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0
        xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x7F,0xFF,0
        x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x3F,0x7F,0x7F,0x7F,0x7F,0x7F,0xFF,0xFF,0xFF,0
        xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x00,

    0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0
        x00,0x00,0x00,0x00,0x00,0x00,0xC0,0xE0,0xC0,0xC0,0x80,0x80,0x00,0x01,0x01,0x03,0x07,0x07,0
        x0F,0x1F,0x3F,0x7F,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xF8,0xF8,0xF8,0
        xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xFF,0xFF,0xFC,0xC0,0x00,0x00,0x00,0x00,0
        x01,0x3F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x0F,0x00,0x00,0x00,0x00,0x00,0xC0,0xFF,0xFF,0
        x00,0x00,0x00,0x00,0x00,0x00,0xE0,0xE0,0xE0,0xC0,0xC0,0xC0,0x80,0x80,0x00,0x00,0x00,0x01,0
        x03,0x03,0x0F,0x0F,0x3F,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x00,

    0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0
        x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFC,0xF8,0xF0,0x00,0
        x00,0x00,0x00,0x00,0x00,0x03,0x0F,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x3F,0x3F,0x3F,0
        x3F,0x3F,0x3F,0x3F,0x3F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFC,0xC0,0x00,0x00,0
        x00,0x00,0x03,0x07,0x07,0x07,0x03,0x03,0x00,0x00,0x00,0x00,0x00,0x80,0xFC,0xFF,0xFF,0x7F,0
        x00,0x00,0x00,0x00,0x00,0x00,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFE,0xFC,0xF8,0
        xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x07,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x00,

    0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0
        x00,0x00,0x00,0x00,0x00,0x00,0xEF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x3F,0x0B,0
        x00,0x00,0x00,0x00,0x00,0x80,0xF8,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xF0,0xF0,0xF0,0
        xF0,0xF0,0xF0,0xF0,0xF0,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFC,0xC0,0
        x00,0x00,0x00,0x00,0x07,0x7F,0x0F,0x00,0x00,0x00,0x00,0x00,0xF8,0xFF,0xFF,0xFF,0xFF,0x72,0
        x00,0x00,0x00,0x00,0x00,0x00,0x0F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x3F,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xFC,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x00,

    0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0
        x00,0x00,0x00,0x00,0x00,0x00,0x0B,0x1F,0x0F,0x0F,0x0F,0x07,0x07,0x03,0x01,0x80,0x80,0xC0,0
        xE0,0xE0,0xF0,0xF8,0xFE,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1F,0x1F,0x1F,0
        x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0xBF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        xFE,0xF0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xE0,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xF2,0
        x00,0x00,0x00,0x00,0x00,0x00,0x0E,0x1F,0x0F,0x0F,0x0F,0x0F,0x0F,0x07,0x07,0x03,0x01,0x00,0
        x00,0x80,0xC0,0xE0,0xF0,0xF8,0xFC,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x00,

    0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFC,0
        xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xFC,0xFC,0xFC,0xFE,0xFE,0xFE,0xFE,0xFF,0xFF,0xFF,0xFF,0
        xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFC,0xF8,0xF8,0xF8,0xF8,0xF8,0xFC,0xFC,0xFC,0
        xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0xFD,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        xFF,0xFF,0xFF,0xFC,0xFC,0xFC,0xFC,0xFC,0xFC,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFD,0
        xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xFC,0xFC,0xFC,0xFC,0xFE,0xFE,0xFE,0xFF,0
        xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x00,
```

```
    0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x01,0
        x00,0x00,0x01,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x8F,0x9F,0x9F,0x00,0x00,0x0F,0xFF,0
        x01,0x00,0x00,0x98,0x98,0x98,0xF8,0xF9,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x7F,0x07,0x01,0
        x80,0x01,0x0F,0x0F,0x01,0xC0,0x80,0x03,0x1E,0xF0,0xC0,0x07,0x0F,0x4F,0x03,0x80,0xF0,0xFF,0
        xFF,0x00,0x00,0xF8,0xF8,0xF8,0xF8,0xF8,0xFF,0x01,0x00,0x00,0x8F,0x07,0x63,0xF0,0xFC,0xFF,0
        xFF,0xF7,0x00,0x00,0xFF,0xDD,0x00,0x00,0x0F,0x83,0xE0,0xF0,0x00,0x00,0xFF,0x87,0x01,0x20,0
        xF8,0xF9,0xE9,0xC0,0xC0,0xC0,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x00,

    0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFB,0xF9,0xF9,0xF9,0xF9,0
        xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xF8,0xFF,0xF8,0xF8,0xF8,0xFF,0xFF,0xFF,0xF8,0xF8,0xF8,0xFF,0
        xF8,0xF8,0xF8,0xF9,0xF9,0xF9,0xF9,0xF9,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFB,0xF8,0xF8,0xFE,0
        xFF,0xFF,0xFC,0xFC,0xFC,0xFF,0xFF,0xFC,0xF8,0xF8,0xFF,0xFE,0xF8,0xF8,0xFC,0xFF,0xFF,0xFF,0
        xFF,0xF8,0xF8,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xF8,0xF8,0xF8,0xFF,0xFE,0xFC,0xF8,0xF9,0xFB,0
        xFF,0xFC,0xF8,0xF8,0xFF,0xFB,0xF8,0xF8,0xFE,0xFF,0xFF,0xFF,0xF8,0xF8,0xFF,0xFF,0xFE,0xFC,0
        xF8,0xF8,0xF8,0xF8,0xFD,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
        x00,

};
```

B.7.5 Schema Aufteilung Display RAM LCD (Quelle [14])

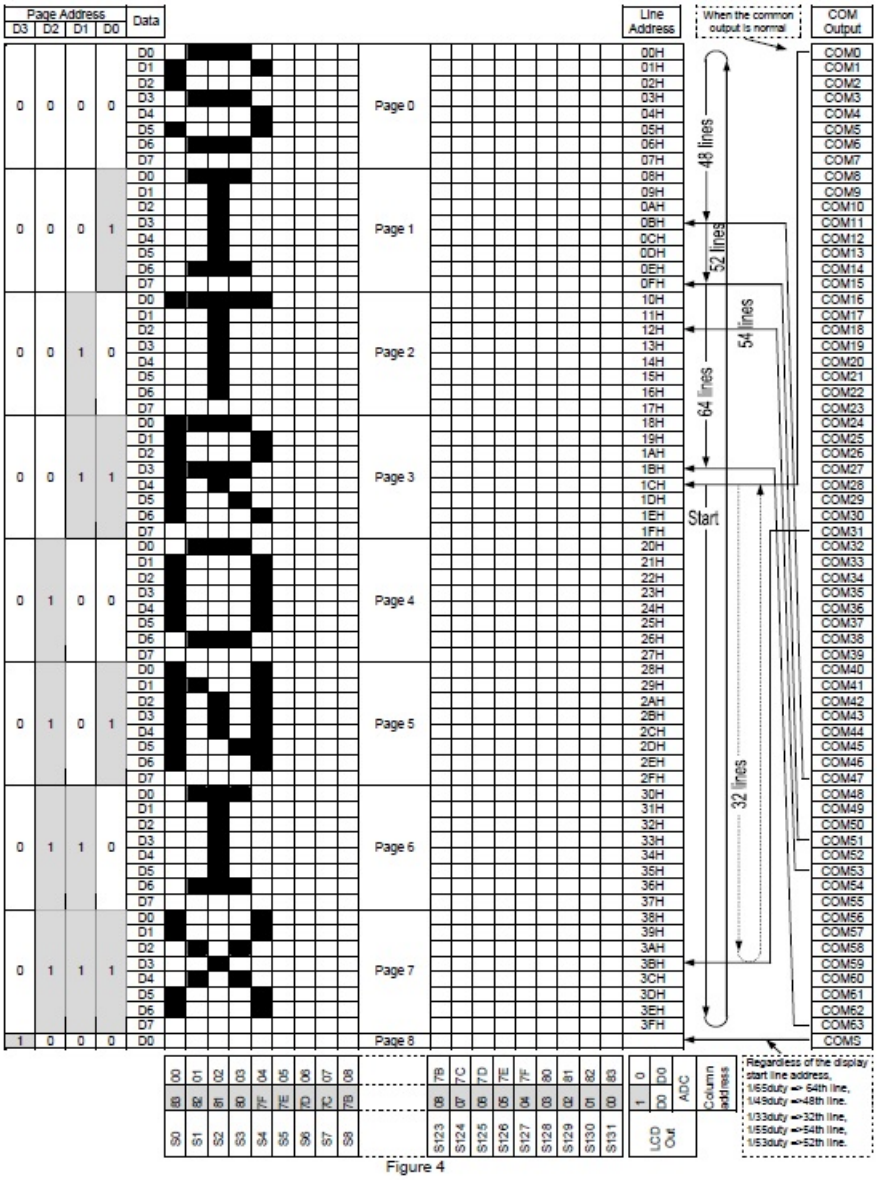


Figure 4

B.8 Tastermatrix

B.8.1 Testprogramm Tastermatrix

```
/*****************************************************************************
//BA_Experimentierboard_Versuch5
//Jan Schlüter
//27.10.2012
//Auswertung einer Tastermatrix und Ausgabe auf LCD
//GPIOs, Interrupt, Library, SysClock 80MHz
*****************************************************************************/



#include "lm3s9d92.h"
#include "disp.h"


//Waiting Time  before decoding in us
#define WAIT 15000


//Global variables
int code[4];                                    //Code array
int codee [4] = {0.0,0,0};                      //Reference code

//Function prototype
void timerwait(int time);


//Interrupt Service Routine
void myHandler(void)
{
  static int count = 0;
  int i,j=0;


  GPIO_PORTE_DATA_R |= 0x60;                    //Test left column
  timerwait(WAIT);                              //WAIT
  switch(GPIO_PORTE_DATA_R & 0x0F){             //Test inputs
    case 0x1: code[count]=7;break;
    case 0x2: code[count]=4;break;
    case 0x4: code[count]=1;break;
    default:GPIO_PORTE_DATA_R &= ~0x60;
        GPIO_PORTE_DATA_R |= 0xA0;              //Test middle column
        timerwait(WAIT);                        //WAIT
        switch(GPIO_PORTE_DATA_R & 0x0F){       //Test inputs
          case 0x1: code[count]=8;break;
          case 0x2: code[count]=5;break;
          case 0x4: code[count]=2;break;
          case 0x8: code[count]=0;break;
          default:GPIO_PORTE_DATA_R &= ~0xA0;
              GPIO_PORTE_DATA_R |= 0xC0;        //Test left column
              timerwait(WAIT);                  //WAIT 0,625ms
              switch(GPIO_PORTE_DATA_R & 0x0F){ //Test inputs
                case 0x1: code[count]=9;break;
                case 0x2: code[count]=6;break;
                case 0x4: code[count]=3;break;
```

```
                default:code[count]=11;
              }
        }

  }

  GPIO_PORTE_DATA_R &= ~0xE0;        //Set back PE5-7

  count++;                           //Increment count
  if(count==1){
    lcd_invers(0);                   //clear LCD
    lcd_clear();
    send_num(code[count-1],2,5);     //Send number to LCD ROM
  }
  else if(count==2){
    send_num(code[count-1],2,36);    //Send number to LCD ROM
  }
  else if(count==3){
    send_num(code[count-1],2,67);    //Send number to LCD ROM
  }
  else if(count==4){
    send_num(code[count-1],2,99);    //Send number to LCD ROM
    timerwait(50000);                //Wait 51 ms
    for(i=0;i<4;i++){                //compare reference Code with input code
      if(code[i]==codee[i]){
        j++;
      }
    }
    if(j==4){                        //Code right: LCD white
      lcd_clear();
    }
    else{                            //Code wrong LCD black
      lcd_invers(1);
    }

    count = 0;                       //Set back count
    j=0;
  }

  timerwait(200000);                 //Wait 0,2 s
  GPIO_PORTB_ICR_R |= 0x04;          //Interrupt PB2 cleared

}

void main(void) {


    //set SysClock up to 80MHz
    SYSCTL_RCC_R |= 0x00000800;        //Bypass on
    SYSCTL_RCC_R &= ~0x00400000;       //Disable sysdiv
    SYSCTL_RCC_R |= 0x00002000;        //Disable PLL
    SYSCTL_MISC_R |= 0x40;             //Clear PLL raw interrupt
    SYSCTL_RCC_R &= ~0x000007C0;       //Reset xtal
    SYSCTL_RCC_R |= 0x00000540;        //OSC 16MHz
    SYSCTL_RCC2_R |= 0x00000800;       //Bypass 2 on
    SYSCTL_RCC2_R |= 0x80000000;       //Use RCC2
    SYSCTL_RCC2_R |= 0x40000000;       //Div 400 off
    SYSCTL_RCC2_R &= ~0x1FC00000;      //Clear sysdiv2
    SYSCTL_RCC2_R |= 0x01000000;       //Sysdiv /5 => 80MHz
    SYSCTL_RCC2_R &= ~0x00000070;      //Clear OSC Source
```

```
    SYSCTL_RCC2_R |= 0x00000010;        //OSC Source precision internal oscillator
    SYSCTL_RCC2_R &= ~0x00002000;       //Enable PLL
    SYSCTL_RCC_R |= 0x00400000;         //USESYSDIV enable

    while((SYSCTL_RIS_R & 0x40)==0){    //Wait for PLL Lock
      }
    SYSCTL_RCC2_R &= ~0x00000800;     //Bypass off

    //GPIOs
    SYSCTL_RCGC2_R = 0x000000DA;      //Unlock and clock Port B and Port D and Port E

    GPIO_PORTB_DEN_R |= 0x04;         //Enable digital I/O pin PB2
    GPIO_PORTE_DEN_R |= 0xFF;         //Enable digital I/0 pin PE

    GPIO_PORTB_DIR_R &= ~0x04;        //PB2 input
    GPIO_PORTE_DIR_R |= 0xF0;         //PE4 output
    GPIO_PORTE_DIR_R &= ~0x0F;        //PE0-3 input

    GPIO_PORTE_DATA_R &= ~0xF0;       //Switch output PE4 off & outputs PE5-7 low

    //Interrupt
    GPIO_PORTB_IS_R &= ~0x04;         //Sensible on edges
    GPIO_PORTB_IBE_R &= ~0x04;        //Interrupt event controlled by IEV register
    GPIO_PORTB_IEV_R |= 0x04;         //Sensible on rising edge
    GPIO_PORTB_IM_R |= 0x04;          //Interrupt unmasked send to interruptcontroller

    NVIC_PRI0_R |= 0x2000;            //Interrupt priority 1
    NVIC_EN0_R |= 0x02;               //Enables interrupt Port B

    //GPIOs
    GPIO_PORTH_DEN_R |= 0xE7;         //Enable digital I/O pin PH0-2 and PH5-7
    GPIO_PORTD_DEN_R |= 0xFF;         //Enable digital I/0 pin PD0-7
    GPIO_PORTG_DEN_R |= 0x02;         //Enable digital I/O pin PG1

    GPIO_PORTH_DIR_R |= 0xE7;         //PH0-2 and PH5-7 output
    GPIO_PORTD_DIR_R |= 0xFF;         //PD0-7 output
    GPIO_PORTG_DIR_R |= 0x02;         //PG1 output


    init_disp();
    lcd_clear();

    //Timer 2
    SYSCTL_RCGC1_R |= 0x00040000;     //Timer 2 unlock
    TIMER2_CTL_R &= ~0x0101;          //Timer 2 A B stop
    TIMER2_CFG_R = 0x00000000;        //Timer 2 32 bit mode
    TIMER2_TAMR_R  |= 0x01;           //One shot mode mode selected




    while(1){

    }


}

void timerwait(int time){
```

```
  int timerload;

  timerload = time*82;                //Timer max value in tics
  TIMER2_TAILR_R = timerload;         //Timer stop value
  TIMER2_CTL_R |= 0x0101;             //Timer 2 A B start
  while((TIMER2_RIS_R & 0x01)==0){    //Wait for timer reach stop value
  }
  TIMER2_ICR_R |= 0x01;               //Reset timer raw interrupt
  TIMER2_CTL_R &= ~0x0101;            //Timer 2 A B stop

}
```

C Templates

C.1 Template Konfiguration EEPROM

```
<?xml version="1.0" encoding="utf-16"?>
<FT_EEPROM>
  <Chip_Details>
    <Type>FT2232D</Type>
  </Chip_Details>
  <USB_Device_Descriptor>
    <VID_PID>1</VID_PID>
    <idVendor>0403</idVendor>
    <idProduct>BCDA</idProduct>
    <bcdUSB>USB 2.0</bcdUSB>
  </USB_Device_Descriptor>
  <USB_Config_Descriptor>
    <bmAttributes>
      <RemoteWakeupEnabled>false</RemoteWakeupEnabled>
      <SelfPowered>true</SelfPowered>
      <BusPowered>false</BusPowered>
    </bmAttributes>
    <IOpullDown>false</IOpullDown>
    <MaxPower>0</MaxPower>
  </USB_Config_Descriptor>
  <USB_String_Descriptors>
    <Manufacturer>TXI</Manufacturer>
    <Product_Description>Experimentierboard</Product_Description>
    <SerialNumber_Enabled>true</SerialNumber_Enabled>
    <SerialNumber />
    <SerialNumberPrefix>0B</SerialNumberPrefix>
    <SerialNumber_AutoGenerate>true</SerialNumber_AutoGenerate>
  </USB_String_Descriptors>
  <Hardware_Specific>
    <Port_A>
      <Hardware>
        <UART>false</UART>
        <_245FIFO>true</_245FIFO>
        <CPUFIFO>false</CPUFIFO>
        <OPTO>false</OPTO>
        <HighIO>false</HighIO>
      </Hardware>
      <Driver>
        <VCP>false</VCP>
        <D2XX>true</D2XX>
      </Driver>
    </Port_A>
    <Port_B>
```

```
        <Hardware>
          <UART>true </UART>
          <_245FIFO>false </_245FIFO>
          <CPUFIFO>false </CPUFIFO>
          <OPTO>false </OPTO>
          <HighIO>false </HighIO>
        </Hardware>
        <Driver>
          <VCP>true </VCP>
          <D2XX>false </D2XX>
        </Driver>
      </Port_B>
    </Hardware_Specific>
</FT_EEPROM>
```

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 28. Februar 2013	
Ort, Datum	Unterschrift