24. September 2008

MSP430 Education System v2.0

Benutzerhandbuch

Forschungs- uns Transferzentrum Leipzig e.V.
an der HTWK Leipzig
Wächterstraße 13
D-04107 Leipzig
Tel.: +49 (0)341-3076 1136
Fax: +49 (0)341-3076 1220
E-Mail: info@ftz-leipzig.de
Internet: http://www.msp430-buch.de

Dokument-Version 1.1

I. Inhaltsverzeichnis

II. Copyright

Im Buch verwendete Bezeichnungen für Erzeugnisse, die zugleich ein eingetragenes Warenzeichen darstellen, wurden nicht besonders gekennzeichnet. Alle Marken- und Warennamen (Zeichen) wurden ohne Gewährleistung der freien Verfügbarkeit genutzt. Das Fehlen der © Markierung ist demzufolge nicht gleichbedeutend mit der Tatsache, dass die Bezeichnung als freier Warenname gilt. Ebenso wenig kann anhand der verwendeten Bezeichnung auf eventuell vorliegende Patente oder einen Gebrauchsmusterschutz geschlossen werden.

Die Informationen in diesem Handbuch wurden sorgfältig übergeprüft und können als zutreffend angenommen werden. Dennoch sei ausdrücklich darauf hingewiesen, dass der Autor weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgeschäden übernimmt, die auf den Gebrauch oder den Inhalt dieser Ausarbeitung zurückzuführen sind. Die in diesem Handbuch enthaltenen Angaben können ohne vorherige Ankündigung geändert werden. Der Autor geht damit keinerlei Verpflichtungen ein.

Kontaktadressen:

FTZ Leipzig e.V. an der HTWK Leipzig
Carsten Kögler / Thomas Minner
Wächterstraße 13
D-04107 Leipzig
Tel.: +49 (0)341-3076 1136
Fax: +49 (0)341-3076 1220
E-Mail: msp430edu@ftz-leipzig.de
Internet: http://www.ftz-leipzig.de/

III. History

- 1.0 – 27.3.2008
  - Erste Version für Hardware 2.0
- 1.1 – 24.9.2007
  - neueste IAR-Version dokumentiert
  - Schaltplan aktualisiert
  - Design überarbeitet

IV.Abbildungsverzeichnis

V. Tabellenverzeichnis

VI. Beschreibung verwendeter Symbole

Alle verwendete Symbole in einer Übersicht, die zur Verwendung dieses Handbuches wichtig sind.

Mit diesem Symbol werden Textpassagen gekennzeichnet, die besondere Beachtung beim Anwender finden sollen. Es werden wichtige Einstellungen und Einschränkungen beschrieben, die bei Verwendung des Gesamtsystems wichtig sind.

1 Einführung

Dieses Mikrocontroller Experimentiersystem ist zu Studien-, Lehr-, Aus- und Weiterbildungszwecken entworfen worden. Es ist speziell an Personen gerichtet, die einen einfachen Einstieg in die Mikrocontrollerprogrammierung suchen.

Der verwendete 16-Bit RISC-Controller der Firma Texas Instruments gehört zu der weit verbreiteten MSP430-Familie. Die Nutzergemeinde um diesen Controller wird immer größer, da er ein sehr leistungsfähiger, stromsparender und vor allem günstiger Mikrocontroller ist. Mit dem Einsatz dieses Experimentiersystems sind Sie ein Teil dieser Gemeinde und werden die Vorzüge dieser Lösung schnell zu schätzen wissen.

Zur Programmierung können Sie die frei zur Verfügung stehende Kickstart Version des IAR C- und Assembler-Compilers „IAR Embedded Workbench“ [1] verwenden. Diese Version ist auf 4 kByte C-Code begrenzt, der Assemblerteil kann jedoch uneingeschränkt verwendet werden. Dies wird durch die gut bedienbare Entwicklungsumgebung, ein integriertes Programmierinterface, ein zusätzliches Terminalprogramm sowie einem umfangreichen Debugger wieder mehr als ausgeglichen. Durch die Codebegrenzung können jedoch nur kleinere Projekte realisiert werden. Die Software IAR Embedded Workbench kann von der Firma Texas Instruments [1] oder direkt bei IAR Systems [2] bezogen werden.

Als Alternative steht ihnen auch der frei verfügbare und kostenlose GNU C-Compiler (GCC) [3] zur Verfügung, sowie weitere sehr gute Compiler der Firmen Rowley [4] und Imagecraft [5]. Über den Einsatz des entsprechenden Compilers kann jeder individuell entscheiden. Alle Compiler sind prinzipiell einsetzbar und lauffähig.

Dieses Handbuch beschreibt den Aufbau der Hardware, der Software und unterstützt sie bei ersten Schritten der Programmierung mit Hilfe der IAR Kickstart Programmierumgebung. Zum besseren Verständnis wird an einer Beispielapplikation der komplette Ablauf beschrieben und erklärt.

Das vorliegende Handbuch soll ebenso Lehrmaterial wie auch Dokumentation darstellen und eine Hilfestellung beim Erlernen komplexer Abläufe sein. Alle Schritte werden in Kombination mit der IAR Embedded Workbench beschrieben und erklärt. Softwarebeschreibungen beziehen sich, in diesem Handbuch, immer auf den IAR Compiler.

2 Unterschiede zwischen Version 1.4 und 2.0

Die Überarbeitung des MSP430 Education Systems hat einige Änderungen mit sich gebracht. Bild 2-1 zeigt die ältere Version 1.4, die auch im Buch „Mikrocontrollertechnik“ von Prof. Matthias Sturm beschrieben ist.

Bild 2-1: MSP430 Education System Version 1.4

Die wichtigsten Unterschiede sind:

- Neuer erweiterter Mikrocontroller MSP430F2272 statt MSP430F1232. Dieser bietet die günstige Programmierung über USB (Spy-by-Wire (SBW) und eZ430), mehr I/Os und Speicher bei gleichzeitiger weitgehender Codekompatibilität.
- Infrarotschnittstelle mit Sender und Empfänger. Diese wurden an freie Ports angeschlossen.
- Vereinfachung der Stromversorgung. Wegfall aller Jumper.
- Versorgung des LCDs mit +5V
- Kompaktere Bauweise
- Wegfall des Anzeigeinstruments
- Wegfall des Lochrasterfeldes an der linken unteren Ecke. Als Ausweichmöglichkeit stehen die Stiftleisten am Mikrocontroller zur Verfügung.

Ein separates Dokument beschreibt die notwendigen Codeänderungen.

3 Features des MSP430 Education Systems 2.0

.

- **Controller**
  - MSP430F2272 (32kB + 256B Flash Memory / 1kB RAM)
  - alle Pins des MSP430 sind über zwei Stiftleisten zugänglich
- **Spannungsversorgung**
  - 3,3 V Spannungsversorgung mit verschiedenen Anschlussmöglichkeiten:
  - Getrennte Spannungsversorgung für analoge On-Chip-Peripherie
- **Taktversorgung**
  - Interner Oszillator mit typisch 12 kHz (VLO)
  - Interne FLL mit bis zu 16 MHz (DCO)
  - Externer Uhrenquarz mit 32.768 Hz
  - Externer HF-Quarz mit bis zu 16 MHz
- **Ein- und Ausgabe**
  - 8 LEDs an einem Port zur universellen Nutzung als Ausgabeelement (z.B. als Test- und Debugelemente)
  - 4 Taster zur universellen Nutzung als Eingabeelement
  - 1 LC-Display (zweizeilig x 16 Zeichen) mit 4-Bit-Ansteuerung, um die Debugmöglichkeit zu erhalten
  - 2 Potentiometer 100k linear
  - 1x Piezo-Lautsprecher mit Komplementär-Gegentakt-Endstufe
  - Infrarotschnittstelle zum Senden- und Empfangen von RC5 Codes
- **Programmierung**
  - Anschluss über Standard 14-Pin Steckverbinder
  - Programmierung über Spy-by-wire mit USB-Adapter
  - Programmierung über JTAG mit parallelem Adapter
- **weitere Schnittstellen**
  - serielle Schnittstelle RS-232, Verbindung zum PC über einfaches Verlängerungskabel
- **Sonstiges**
  - Reset-Taster
- **Abmaße**
  - Hauptplatine 93 x 80 mm
  - Abmaße über Alles: 100 x 82 mm

-

4 Hardwarebeschreibung

Das MSP430 Education System ist unter der Voraussetzung entworfen worden, mehrere Anwendungsgebiete in einem System zu vereinen. Der Einsatz als Education System ist ebenso möglich wie der Einsatz als Evaluation System zum Entwerfen eigener Applikationen. Durch Herausführen aller Pins des Controllers auf die Stiftleisten X7 und X8 können alle Funktionen des Controllers für eigene Applikationen genutzt werden. Es wurden bestimmte Standardkomponenten auf der Platine fest installiert, um eine Mindestausstattung zu gewährleisten und einfache Applikationen leicht entwickeln zu können.

Durch die Kompaktheit des eingesetzten Controllers und der wenigen zur Verfügung stehenden Ports, ist eine Mehrfachbelegung bestimmter Ports unvermeidbar.

Zur Bereitstellung einer seriellen Kommunikation wurde das UART0-Modul des MSP430F1232 verwendet und über einen RS-232 Treiber an eine Standard SUB-D9 Schnittstelle angeschlossen. Des Weiteren wurde ein Standard LC-Display (zweizeilig mit jeweils 16 Zeichen) installiert.

Im Folgenden werden alle wichtigen Elemente des MSP430 Education System beschrieben und erläutert. Einen Überblick aller Systeme gibt das Bild 4-1.

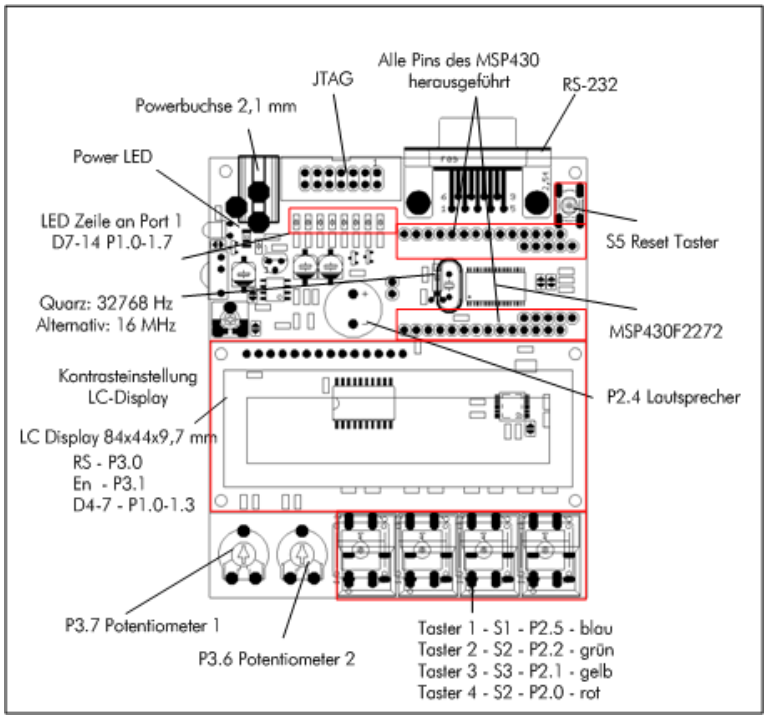


Bild 4-1: Hauptelemente des MSP430 Education Systems

Im Folgenden werden kurz alle wichtigen Informationen übersichtlich dargestellt, damit diese während der Programmierung zusammenhängend zur Verfügung stehen.

Tabelle 1: Belegungsplan des Mikrocontrollers

µC Pin	Name	Stiftleiste	Beschreibung und angeschlossene Peripherie
1	Test	X7.1	Anschluss des JTAG-Moduls, schaltet den Controller zum programmieren frei,
2	VCC	X7.2	Spannungsversorgung des Mikrocontrollers mit 3,3 V
3	P2.5	X7.3	S1 - Taster1 (blau)
4	GND	X7.4	Masseanschluss des Mikrocontrollers
5	XOUT	X7.5	Ausgang des 32768 Hz Quarzes
6	XIN	X7.6	Eingang des 32768 Hz Quarzes
7	/RST	X7.7	Resetanschluss des Resettasters S1
8	P2.0	X7.8	S4 – Taster4 (rot)
9	P2.1	X7.9	S3 – Taster3 (gelb)
10	P2.2	X7.10	S2 – Taster2 (grün)
11	P3.0	X7.11	RS-Leitung des LC-Displays
12	P3.1	X7.12	EN (Enable) Leitung des Displays
13	P3.2	X7.13	zur freien Verfügung, nicht belegt
14	P3.3	X7.14	zur freien Verfügung, nicht belegt
15	AVSS	X11.1	Masseanschluss des ADC-Moduls im Mikrocontroller
16	AVCC	X11.2	Masseanschluss des ADC-Moduls im Mikrocontroller
17	P4.0	X11.3	zur freien Verfügung, nicht belegt
18	P4.1	X11.4	zur freien Verfügung, nicht belegt
19	P4.2	X11.5	zur freien Verfügung, nicht belegt
20	P4.3	X12.1	zur freien Verfügung, nicht belegt
21	P4.4	X12.2	zur freien Verfügung, nicht belegt
22	P4.5	X12.3	zur freien Verfügung, nicht belegt
23	P4.6	X12.4	zur freien Verfügung, nicht belegt
24	P4.7	X12.5	zur freien Verfügung, nicht belegt
25	P3.4	X8.14	TX Signal zum RS-232 Anschluss
26	P3.5	X8.13	RX Signal vom RS-232 Anschluss
27	P3.6	X8.12	Potentiometer 2
28	P3.7	X8.11	Potentiometer 1
29	P2.3	X8.10	Infrarotempfänger TSOP1736
30	P2.4	X8.9	Ansteuerung des Lautsprechers SP1
31	P1.0	X8.8	LED 0 (D7) (low active), LCD Datenleitung D4
32	P1.1	X8.7	LED 1 (D8) (low active), LCD Datenleitung D5
33	P1.2	X8.6	LED 2 (D9) (low active), LCD Datenleitung D6
34	P1.3	X8.5	LED 3 (D10) (low active), LCD Datenleitung D7
35	P1.4	X8.4	LED 4 (D11) (low active), JTAG TCK (X3-7)
36	P1.5	X8.3	LED 5 (D12) (low active), JTAG TMS (X3-5)
37	P1.6	X8.2	LED 6 (D13) (low active), JTAG TDI (X3-3)
38	P1.7	X8.1	LED 7 (D14) (low active), JTAG TDO (X3-1)

Im Bild 4-2 wird nochmals die Lage der Stiftleisten X7, X8, X11 und X12 dargestellt, durch die alle Pins des MSP430F1232 Mikrocontrollers frei verfügbar sind, dargestellt.

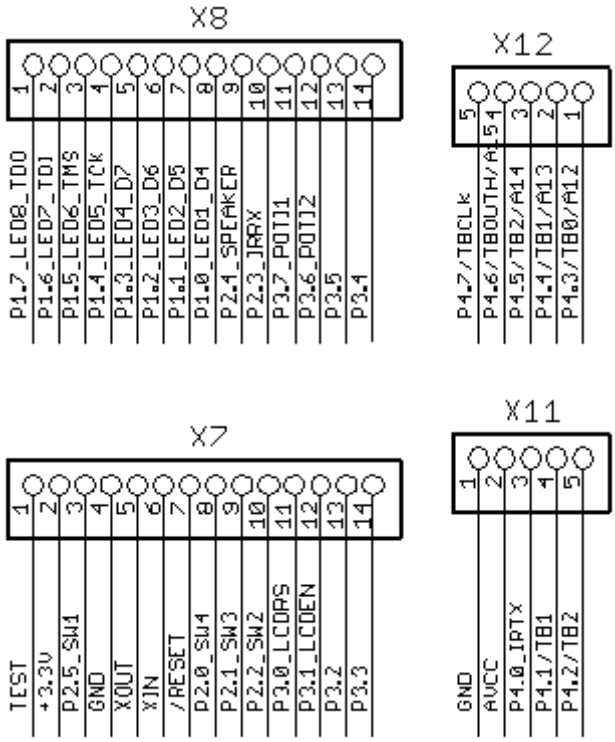


Bild 4-2: Belegung der Stiftleisten X7, X8, X11 und X12 mit Signalnamen

4.1 Der Mikrocontroller MSP430F2272

Der MSP430F2272 ist ein sehr leistungsfähiger 16-Bit Mikrocontroller und ein Nachfolgetyp des MSP430F1232. Er verbraucht sehr wenig Strom und ist zudem beim Kauf sehr günstig. Die wichtigsten Features dieses Controllers zeigt Tabelle 2 im Kurzüberblick. Weitere Features und die komplette Beschreibung können auf der CD oder in aktueller Form auf der Texas Instruments Homepage [6] gefunden werden.

Tabelle 2: Die wichtigsten Features des MSP430F2272

- 32KB + 256B Flash Memory 1KB RAM
- Low Supply Voltage Range 1.8 V to 3.6 V
- Ultralow-Power Consumption
  - Active Mode: 270 µA at 1 MHz, 2.2 V
  - Standby Mode: 0.7 µA
  - Off Mode (RAM Retention): 0.1 µA
- Ultrafast Wake-Up From Standby Mode in Less Than 1 µs
- 16-Bit RISC Architecture, 62.5-ns Instruction Cycle Time
- Basic Clock Module Configurations:
  - Internal Frequencies up to 16 MHz With Four Calibrated Frequencies to ±1%

  - Internal Very-Low-Power Low-Frequency Oscillator
  - 32-kHz Crystal
  - High-Frequency Crystal up to 16 MHz
  - Resonator
  - External Digital Clock Source
  - External Resistor
- 16-Bit Timer_A With Three Capture/Compare Registers
- 16-Bit Timer_B With Three Capture/Compare Registers
- Universal Serial Communication Interface
  - Enhanced UART Supporting
- Auto-Baudrate Detection (LIN)
  - IrDA Encoder and Decoder
  - Synchronous SPI
  - I2C™
- 10-Bit, 200-ksps A/D Converter With Internal Reference, Sample-and-Hold, Autoscan, and Data Transfer Controller
- Brownout Detector
- Serial Onboard Programming, No External Programming Voltage Needed
- Programmable Code Protection by Security Fuse
- Bootstrap Loader
- On Chip Emulation Module
- Family Members Include:
- For Complete Module Descriptions, Refer to the MSP430x2xx Family User's Guide

Die geringe Stromaufnahme ist besonders wichtig für Anwendungen, die über längere Zeit mit einer Batterie arbeiten müssen. Die Stromaufnahme ist durch fünf Stromsparmodi beeinflussbar, in denen jeweils bestimmte interne Funktionseinheiten abgeschaltet werden. Die geringe Wake-Up-Zeit von 1 µs kommt vielen Anwendungen zu gute bei denen der Stromsparmodus aktiv genutzt wird und der Controller nur für eine kurze Zeit aus dem Schlafmodus aufwachen muss, um z.B. eine Messung durchzuführen. Das USART Modul ist besonders für die serielle Kommunikation wichtig. Durch dieses kann mit einem geringen Aufwand eine Kommunikationstrecke zwischen verschiedenen Systemen herstellt werden. Der in dieser Lösung eingesetzte MSP430F2272 ist weiterhin mit 32 kByte Flashspeicher und 1 kByte RAM ausgestattet und ermöglicht damit sehr leistungsfähige Applikationen.

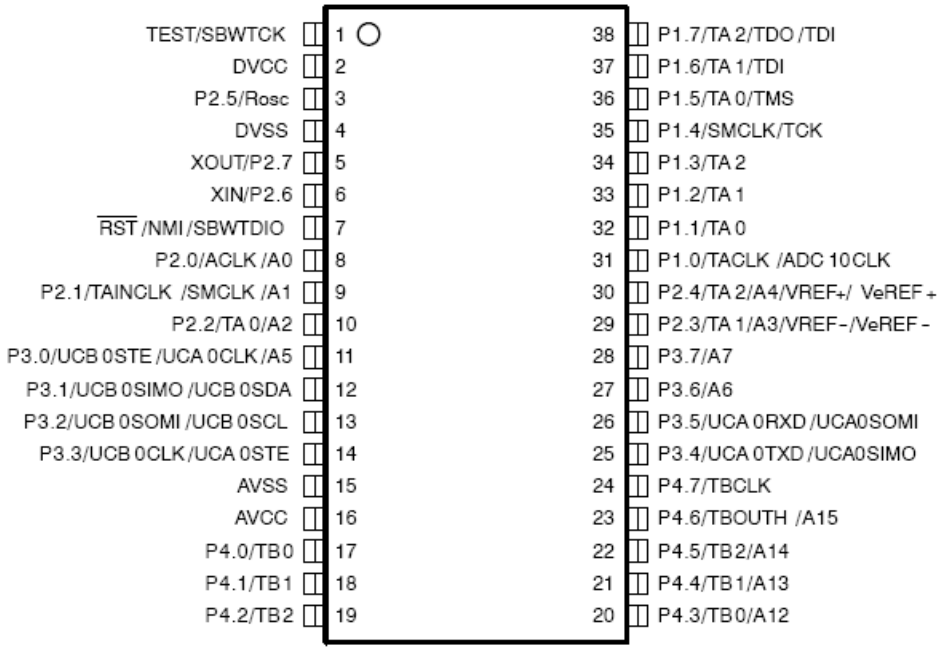
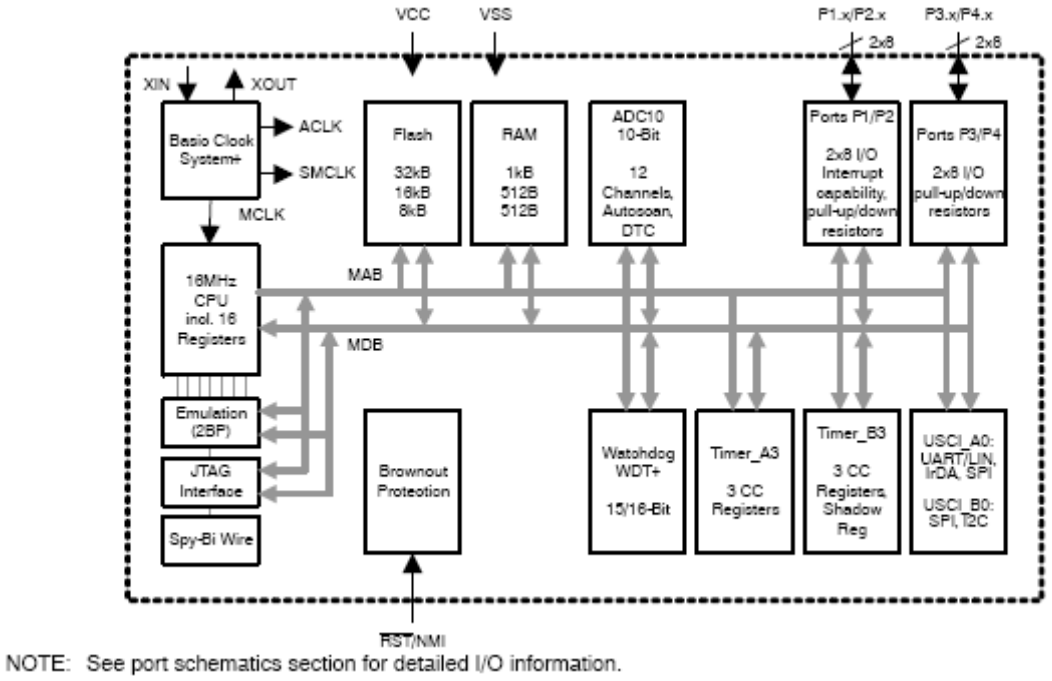


Bild 4-3: Pinbelegung des MSP430F2272

Im Bild 4-3 ist die komplette Pinbelegung des MSP430F2272 Mikrocontrollers dargestellt. An Hand dieses Bildes lassen sich die einzelnen Funktionseinheiten sowie die verwendeten Pins erkennen. In Bild 4-4 wird der interne Aufbau des MSP430F2272 mit Hilfe eines Blockschaltbildes dargestellt. Weitere Details können dem Datenblatt des MSP430F2272 [6] entnommen werden.



NOTE: See port schematics section for detailed I/O information.

Bild 4-4: Blockdiagramm des MSP430F2272

Der MSP430 verfügt intern über eine Reset-Logik, die den Einsatz eines externen Reset-Controllers unnötig macht. Die Brownout Funktion des Mikrocontrollers führt zum Abschalten des Controllers, wenn die Versorgungsspannung unter einen festgelegten Mindestwert fällt. Ansonsten könnte der Controller einen nichtdefinierten Zustand annehmen.
Der Mikrocontroller verfügt über 16 Interrupt-Vektoren, welche zum Teil maskierbar sind. Dies bedeutet, dass jeder Interrupt auf ein bestimmtes Ereignis eingestellt werden kann. Bei dem eingesetzten Mikrocontroller sind nur die beiden höchsten Prioritäten nicht maskierbar, somit können nur diese beiden nicht in ihrer Funktion beeinflusst werden. Die

Tabelle 3 zeigt alle möglichen Interrupts mit der jeweiligen Adresse nochmals in einer Übersicht. Weitere Funktionsweisen und Erklärungen können dem MSP430F2272 Datenblatt [6] und dem MSP430x2xx Family User's Guide [7] entnommen werden. In diesen Dokumenten werden auch die verwendeten Register näher beschrieben, um die Interruptfunktionen gezielt einsetzen zu können.

Tabelle 3: Übersicht aller Interrupt Vektor Adressen

Interruptquelle	Interruptflag	SYSTEM INTERRUPT	WORD ADDRESS	Priorität
Power-up, external reset, watchdog	WDTIFG (see Note 1) KEYS (see Note 1)	Reset	0FFFEh	31, highest
NMI, oscillator fault, flash memory access violation	NMIIFG (see Notes 1 and 4) OFIFG (see Notes 1 and 4) ACCVIFG (see Notes 1 and 4)	(nicht)-maskierbar, (nicht)-maskierbar, (nicht)-maskierbar	0FFFCh	30
Timer_B3			0FFFAh	29
Timer_B3			0FFF8h	28
			0FFF6h	27
Watchdog timer	WDTIFG	maskierbar	0FFF4h	26
Timer_A3	TACCR0 CCIFG (see Note 2)	maskierbar	0FFF2h	25
Timer_A3	TACCR1andTACCR2 CCIFGs. TAIFG (see Notes 1 and 2)	maskierbar	0FFF0h	24
UART Receive	URXIFG0	maskierbar	0FFEEh	23
UART Transmit	UTXIFG0	maskierbar	0FFECh	22
ADC10	ADC10IFG	maskierbar	0FFEAh	21
			0FFE8h	20
I/O Port P2 (eight flags)	P2IFG.0 to P2IFG.7 (see Notes 1 and 2)	maskierbar	0FFE6h	19
I/O Port P1 (eight flags)	P1IFG.0 to P1IFG.7 (see Notes 1 and 2)	maskierbar	0FFE4h	18
			0FFE2h.. 0FFC0h	17. lowest

4.2 Spannungsversorgung

Das MSP430 Education System arbeitet mit 2 verschiedenen Versorgungsspannungen. Der Mikrocontroller und die meisten Peripheriesysteme werden mit 3,3 V versorgt. Das Display und der Infrarotempfänger arbeiten mit +5 V.

Die Spannungsregler sind jeweils für eine Stromaufnahme von 100 mA ausgelegt. Ein größerer Strom kann zu Schäden an den Bauelementen führen.

Der Hohlstecker an X13 muss korrekt gepolt sein. Das Board ist nicht mit einem Brückengleichrichter ausgestettet. Die Masse muss auf dem Außenleiter angeschlossen sein. Ist der Hohlstecker falsch gepolt, sorgt eine Diode dafür, dass das Board nicht beschädigt wird.

Ein Netzteil wird an X1 über einen 2,1 mm Hohlstecker angeschlossen. Der Verpolungsschutz wird durch die Diode D9 realisiert. Es können Eingangsspannungen von 7-16 V angelegt werden. Die Spannungsregler IC4 und IC3 erzeugen 2 geregelten Versorgungsspannungen von 3,3 V und 5 V. Wurde ein Netzteil an die Platine angeschlossen, leuchtet die Diode D10. Bild 4-5 zeigt die wichtigsten Bauelemente.

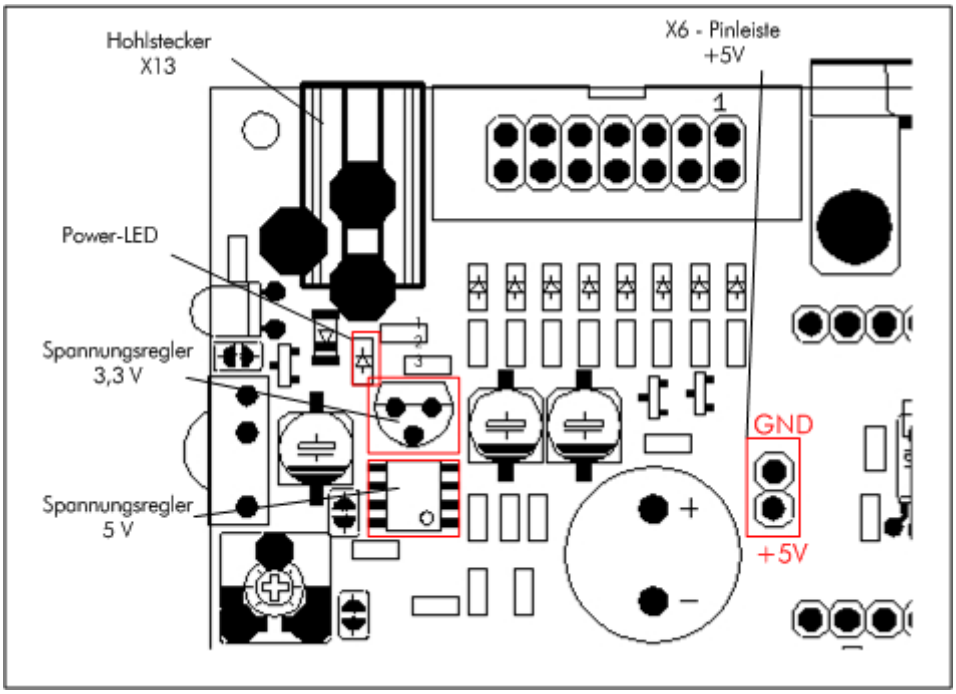


Bild 4-5: Spannungsversorgung MSP430 Education System

Die Version 2.0 hat keinen getrennten Anschluss für eine Batterie. In diesem Fall ist ein Adapter für die Hohlsteckerbuchse notwendig.

Die neue +5 V Versorgungsspannnung kann über die Stiftleiste X6 abgegriffen werden. Die Belegung ist in der obigen Abbildung ersichtlich.

4.3 JTAG-Schnittstelle

Die JTAG-Schnittstelle dient zum einen als Programmierschnittstelle, zum anderen als Debug-Schnittstelle. Durch diese kann ein Programm, direkt auf dem Controller Schritt für Schritt abgearbeitet werden. Hierdurch können Programmfehler direkt auf der Zielhardware gefunden und beseitigt werden, um die immer kürzer werdenden Entwicklungszyklen neuer Applikationen einhalten zu können.

4.3.1 4-Draht-Interface

Die Schnittstelle stellt vier Leitungen zur Verfügung, über die alle Daten transportiert werden. Die Programmiersoftware kann diese Daten in ihrem Debug-Fenster darstellen und auswerten. Die Belegung der Schnittstelle auf dem MSP430 Education System wird in Bild 4-6 dargestellt.

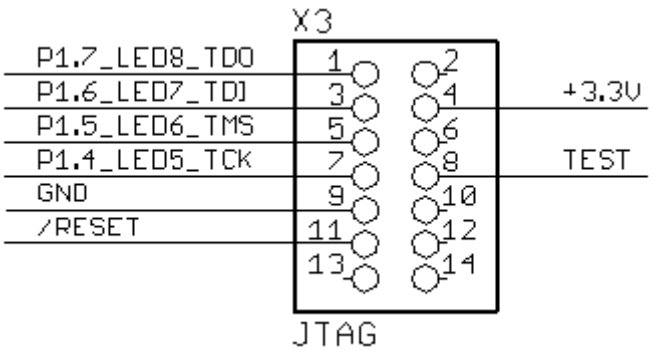


Bild 4-6: Belegung der JTAG-Schnittstelle

Nachfolgend werden alle Leitungen der JTAG-Schnittstelle kurz beschrieben. Die vier Signalleitungen haben folgende Funktion:

- **TCK**, Test Clock:

  An diesem Pin wird das Taktsignal für den TAP Controller angeschlossen. Der komplette JTAG-Testbetrieb läuft synchron mit diesem Takt. Lesevorgänge werden bei steigender Flanke von TCK vorgenommen, während Schreibvorgänge bei fallender Flanke ausgelöst werden. TCK ist normalerweise völlig unabhängig vom Arbeitstakt des Chips.

- **TDI**, Test Data Input:

  Dieser Pin ist der serielle Dateneingang. Sowohl Befehlsopcodes als auch Daten werden über diesen Pin seriell verarbeitet. TDI wird bei steigender Flanke von TCK übernommen.

- **TDO**, Test Data Output:

  Dieser Pin ist ein Ausgang, der Daten-Ausgang. Hier werden alle auszulesenden Daten seriell heraus geschoben. TDO wird immer bei fallender Flanke von TCK aktualisiert.

- **TMS**, Test Mode Select:

  Über diesen Eingang, wird der TAP-Controller gesteuert. Die Zustände des endlichen Automaten, der den TAP-Controller beschreibt, werden über diesen Pin (und TCK) gesteuert. TAP-Modi wie "Befehlsregister laden" oder "Datenregister ausschieben" werden über TMS eingestellt. TMS wird ebenfalls bei steigender Flanke von TCK übernommen.

Die **VCC_OUT** Leitung ermöglicht die Spannungsversorgung eines externen JTAG-Adapters. Mit Hilfe der **TEST** Leitung wird der angeschlossene Mikrocontroller in den Programmiermodus versetzt. Liegt diese Leitung auf HIGH-Pegel, kann der angeschlossene Mikrocontroller durch die JTAG-Schnittstelle programmiert werden.

4.3.2 2-Draht-Interface (Spy-by-Wire)

Der verwendete MSP430F2272 verfügt neben dem klassischen JTAG-Interface mit den Signalen TMS, TDI, TDO und TCK auch über eine neue serielle Version welche mit 2 Signalen auskommt. Dieser wird als Spy-by-Wire bezeichnet und wurde speziell für MSP430-Mikrocontroller von Texas Instruments entwickelt. Die Programiersignale werden über die TEST und RESET-Leitung an den Mikrocontroller angeschlossen. Da diese Leitungen ebenfalls über den 14-poligen JTAG-Steckverbinder zur Verfügung stehen, kann das Board mit beiden Interfaces programmiert werden. Für nähere Informationen zu den empfohlenen Programmieradaptern siehe Abschnitt 6.

Tabelle 4 Pinbelegung Spy-by-Wire

Pin am JTAG-Steckverbinder	4-Draht-Interface	Spy-by-Wire
11	RESET	SBWTCK – Taktleitung
8	TEST	SBWTDIO - Datenleitung

4.4 Serielle Schnittstelle RS-232

Die serielle Kommunikation wird über eine RS-232 Schnittstelle zur Verfügung gestellt. Hierbei kommt ein RS-232-Pegelwandler der Firma Maxim zum Einsatz. Dieser bereitet die Signale so auf, dass der Mikrocontroller mit einem PC oder anderem Gerät kommunizieren kann. Durch Auftrennen der Lötbrücken SJ2 und SJ3 kann der Pegelwandler vom Mikrocontroller getrennt werden. Dadurch werden die entsprechenden Controllerpins frei für eine alternative Beschaltung.

4.5 Leuchtdioden

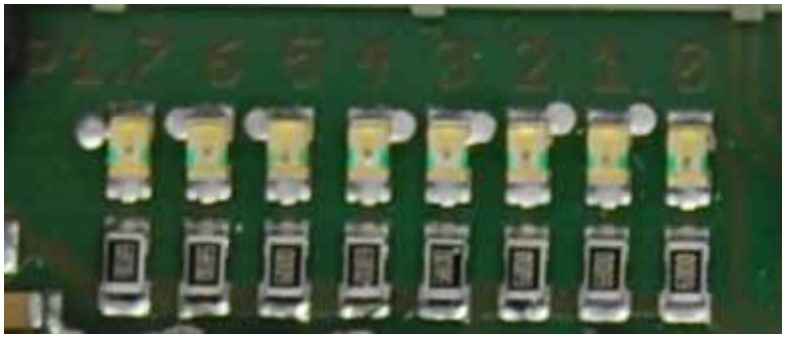


Bild 4-7: Installierte Leuchtdioden

Da Leuchtdioden die Grundlage jeder Mikrocontrollerausbildung darstellen, ist es wichtig, sie an einem zusammenhängenden Port anzuschließen. Dadurch gelingt es, die Verwendung und Funktion der zugehörigen Portregister zu trainieren und die Wichtigkeit dieser Einstellung zu demonstrieren. Diese Einstellungen können dann schneller auf alle anderen Peripheriesysteme adaptiert werden.
Die Leuchtdioden befinden sich an Port 1. Der Port wird ebenfalls für das JTAG-Interface und für die Ansteuerung des LCDs verwendet. Befindet sich der Mikrocontroller im Debug-Modus oder werden gerade Daten zum LCD gesendet, verändert das mit hoher Sicherheit den Status der LEDs.

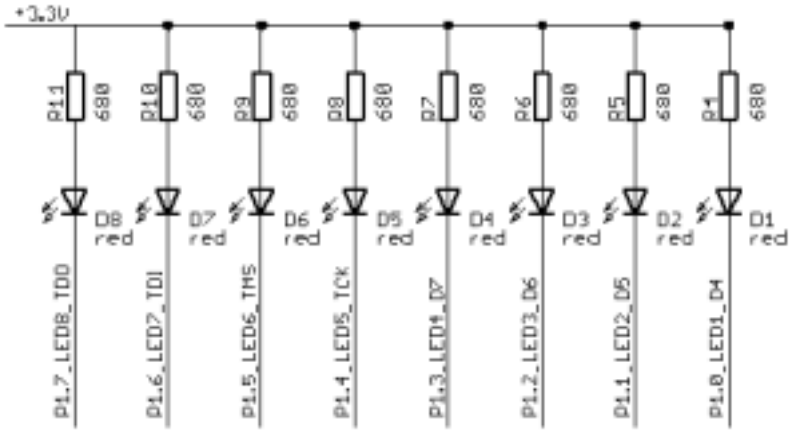


Bild 4-8: Beschaltung und Position der Leuchtdioden

Die Leuchtdioden wurden so an den Mikrocontroller angeschlossen, dass Sie bei Low-Pegel (low activ) am Mikrocontroller leuchten. Da viele Mikrocontroller in dieser Beschaltung mehr Strom treiben können, wurde diese Lösung auch hier eingesetzt, um eine gewisse Kompatibilität herstellen zu können.

Die Anschlussbelegung der Leuchtdioden zeigt das Bild 4-8. Tabelle 5 zeigt die Anschlussbelegung der Leuchtdioden am Mikrocontroller, der Stiftleiste X8 und den verwendeten Port.

Tabelle 5: Anschlussbelegung der Leuchtdioden

Port	Stiftleiste	µC Pin	zugehörige Leuchtdioden
P1.0	X8.8	31	LED 0 (D1) (low active),
P1.1	X8.7	32	LED 1 (D2) (low active),
P1.2	X8.6	33	LED 2 (D3) (low active),
P1.3	X8.5	34	LED 3 (D4) (low active),
P1.4	X8.4	35	LED 4 (D5) (low active)
P1.5	X8.3	36	LED 5 (D6) (low active)
P1.6	X8.2	37	LED 6 (D7) (low active)
P1.7	X8.1	38	LED 7 (D8) (low active)

4.6 Taster

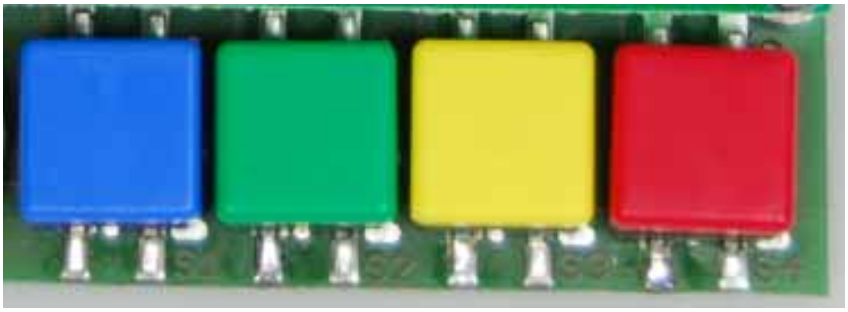


Bild 4-9: Installierte Taster

Da Taster wichtige Funktionen in vielen unterschiedlichen Systemen übernehmen, wurden auf dem MSP430 Education System vier Taster verwendet (Bild 4-9). Alle Taster wurden an einem interruptfähigen Port des MSP430 angeschlossen, um verschiedene Arten der Signalerkennung realisieren zu können. So kann ein Taster per Interruptfunktion, mit Hilfe des Timers oder einfach durch wiederholtes Abfragen (Polling) ausgelesen werden.

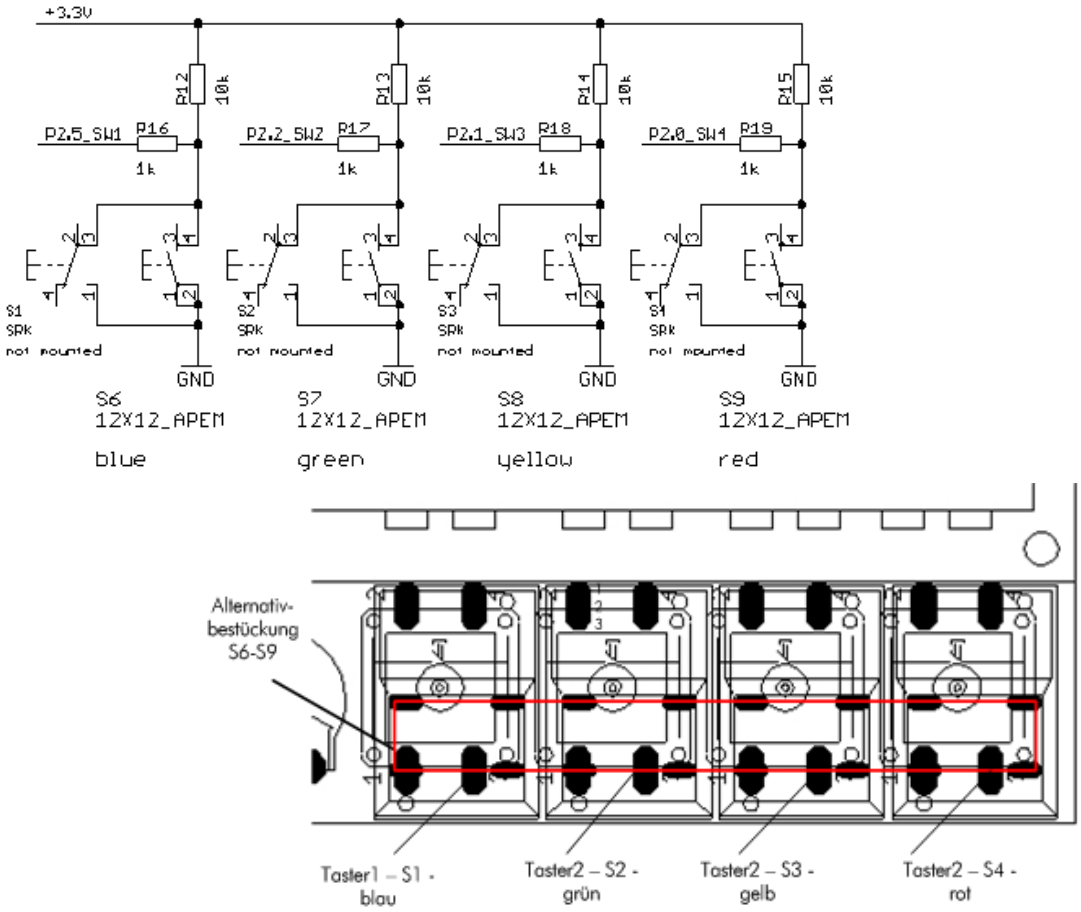


Bild 4-10: Beschaltung und Position der Taster

Wie in Bild 4-10 zu sehen ist, liegen die angeschlossen Ports des Mikrocontrollers (über Vorwiderstände R12-R15) direkt an der Versorgungsspannung (VCC) der Tastereingänge. Die Widerstände R16-R19 dienen ausschließlich der Strombegrenzung beim Drücken der Taster. Die Ports P2.0-2.2 und P2.5 müssen in dem zugehörigen Portregister als Eingang geschaltet werden, um eine Spannungsänderung und somit ein Schalten detektieren zu können. Da die Taster standardmäßig offen sind und die Ports als Eingänge geschaltet wurden, kommt es zu keinen Stromfluss in Richtung Mikrocontroller. Die Taster schalten nach GND, das bedeutet, nach betätigen eines Tasters, wird der angeschlossene Port von VCC nach Masse geschaltet. Wird der Taster betätigt, kommt es zum Stromfluss über den jeweiligen Vorwiderstand von VCC nach Masse. Da die gesamte Spannung über diesen Vorwiderstand abfällt, kann der Mikrocontroller nur noch einen Null-Pegel erkennen. Diese Veränderung von VCC auf GND kann dann über ein Programm, an dem jeweiligen Port, erkannt werden.

Da die Taster an den externen Interruptleitungen angeschlossen sind, ist auch eine interruptgesteuerte Entprellung möglich. Die Widerstände R16-R19 dienen der Absicherung der Mikrocontrollereingänge, um eine Überlastung dieser zu verhindern.

Die Tabelle 6 stellt die Anschlussbelegung noch einmal übersichtlich dar. Die Taster S6, S7, S8 und S9 stellen eine alternative Bestückungsvariante dar.

Tabelle 6: Anschlussbelegung der Taster

Port	Stiftleiste	µC Pin	Peripheriesystem
P2.0	X7.8	8	Taster S1
P2.1	X7.9	9	Taster S2
P2.2	X7.10	10	Taster S3
P2.5	X7.3	3	Taster S4

4.7 Infrarotschnittstelle

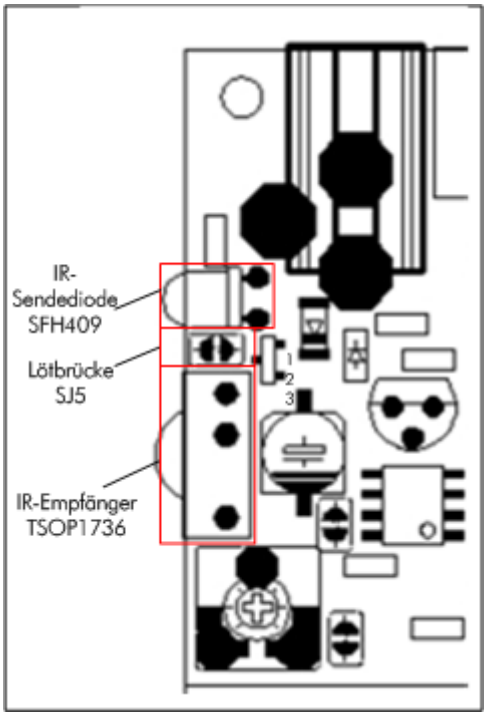


Bild 4-11: Lage der Infrarotkomponenten

Das MSP430 Education System ist mit Infrarotemfangs- und Sendeelementen ausgestattet. Der Empfänger sorgt selbstständig für eine Demodulation von RC5-Codes bei einer Frequenz von 36 kHz. Über die Lötbrücke SJ5 kann der Infrarotempfänger abgeschaltet werden. Tabelle 7 gibt einen Überblick über die Portbelegung.

Tabelle 7: Anschlussbelegung der Infrarotbauelemente

Port	Stiftleiste	µC Pin	Peripheriesystem
P2.3	X8.10	29	IR Emfangssignal
P4.0	X11.3	17	IR Sendesignal

4.8 LC-Display

Bild 4-12: Installiertes Dotmatrix Display

Das eingesetzte Display (Bild 4-12) ist ein Standard Dotmatrixdisplay mit zwei Zeilen und 16 Zeichen pro Zeile. Dieses Display kann einen fest eingestellten Zeichensatz und bestimmte selbst definierte Symbole darstellen, aber keine Grafiken. Zur Vermittlung der Funktionsweise eines solchen Displays reicht dies jedoch bei weitem aus. Bild 4-13 zeigt die Beschaltung des Displays auf dem MSP430 Education System.

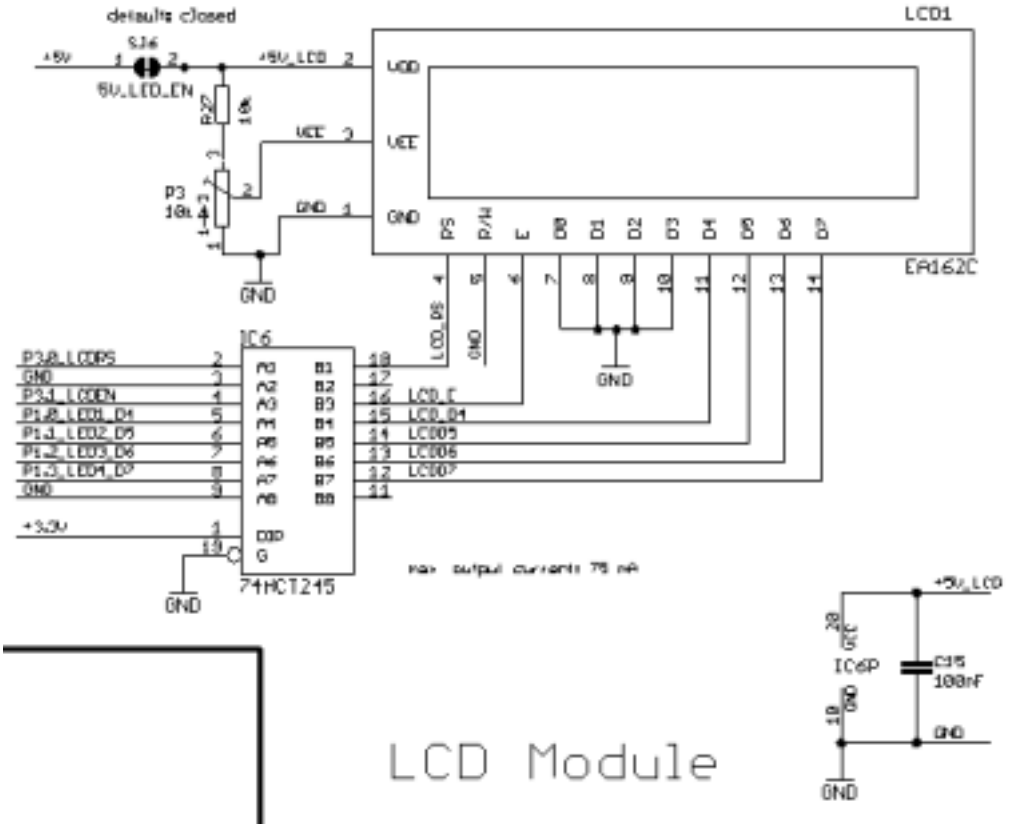


Bild 4-13: Beschaltung des LC-Displays

Es wurde der 4-Bit Modus gewählt, damit die Debugfunktionalität über die JTAG-Schnittstelle auch bei aktivem Display erhalten bleibt.

Das Display wird mit 5V betrieben. Der Oktalbuffer IC6 wird benötigt um die 3,3V-Spannungspegel vom Mikrocontroller umzusetzen. Auf Grund der Beschaltung ist es nicht möglich, Daten vom Display zu lesen. Die R/W-Leitung des Displays ist fest auf „schreiben" / Massepegel gelegt.

Über das Potentiometer R20 kann der Kontrast des Displays nachgeregelt und eingestellt werden.

Die nachfolgenden Tabellen beschreiben die wichtigsten Anschlüsse und Pinbelegungen des Displays.

Tabelle 8: Anschlussbelegung des LC-Displays

Port	Stiftleiste	µC Pin	Peripheriesystem
P3.0	X7.11	11	Ansteuerung der RS-Leitung des LC-Displays
P3.1	X7.12	12	Ansteuerung der E (Enable) Leitung des Displays
P1.0	X8.8	21	LCD Datenleitung D4
P1.1	X8.7	22	LCD Datenleitung D5
P1.2	X8.6	23	LCD Datenleitung D6
P1.3	X8.5	24	LCD Datenleitung D7

Tabelle 9: Signal- und Belegungsplan des Displays

Pin	Signal	Pegel	Beschreibung
1	GND	L	negative Spannungsversorgung 0 V
2	VDD	H	positive Spannungsversorgung +5 V
3	VEE	-	Kontrasteinstellung des Displays
4	RS	H / L	Register Select
5	R/W	L	Read H / Write L, fest auf L-Pegel eingestellt
6	E	H	Enable
7	D0	H / L	Datenleitung 0
8	D1	H / L	Datenleitung 1
9	D2	H / L	Datenleitung 2
10	D3	H / L	Datenleitung 3
11	D4	H / L	Datenleitung 4
12	D5	H / L	Datenleitung 5
13	D6	H / L	Datenleitung 6
14	D7	H / L	Datenleitung 7
15	LED +	-	Hintergrundbeleuchtung Plus-Leitung
16	LED -	-	Hintergrundbeleuchtung Minus-Leitung

Um das Display verwenden zu können, werden noch weitere wichtige Informationen, wie die grundsätzlichen Eigenschaften des Displays, der verwendete Befehlsatz sowie die richtigen Initialisierungsparameter, benötigt. Ohne die Initialisierungsparameter kann das Display nicht verwendet werden. Alle benötigten Features werden auf den folgenden Seiten kurz beschrieben und können im zugehörigen Datenblatt [17] nochmals in ausführlicher Form nachgelesen werden. Die wichtigsten Eigenschaften des Displaycontrollers (HD44780) im Überblick:

- 4-Bit oder 8-Bit MPU-lnterface (hier 4-Bit verwendet)
- Integriertes Display RAM für 80 Zeichen
- Zeichengenerator ROM

  5x 7: 160 Zeichen

  5x10: 32 Zeichen
- Display Daten und Zeichengenerator RAM können von der MPU gelesen werden
- Umfangreicher Befehlssatz: Display löschen, Cursor home, Display ON/OFF, Cursor ON/OFF, Zeichen Blinkfunktion,Cursor shift, Anzeigen shift
- Interner Power On Reset (POR)

Der HD44780 verfügt über einen 8-Bit-Datenbus sowie über die Steuersignale R/W (Read/Write) das in dieser Lösung fest auf Schreiben liegt, RS (Register Select) und E (Enable). Es ist sowohl ein 8-Bit als auch ein 4-Bit Betrieb des Controllers möglich. In dieser Lösung wurde sich auf den 4-Bit Modus beschränkt, um die Debugfunktionalität zu erhalten. Die zu verwendende Datenbusbreite, kann lediglich während der Initialisierung festgelegt werden. Bei Verwendung des 4-Bit Betriebs müssen die im Folgenden beschriebenen Kommandos in zwei aufeinander folgenden Schritten an den HS44780 gesendet werden. Zuerst der High Nibble, dann der LOW Nibble. Beide Nibble werden über die Datenleitungen DB4-DB7 übertragen. Dadurch werden die Datenleitungen DB0-DB3 nicht mehr beachtet.

Mit dem Signal RS wird dem Displaycontroller mitgeteilt, ob Anweisungen, Instruktionscodes (RS=0) oder Daten (RS=1) übertragen werden. Bei der fallenden Flanke des Enable Signals (E) übernimmt der Display-Controller die Daten.

Im Folgenden wird der Ablauf der Hardware-Initialisierung beschrieben:

Nach Anlegen der Versorgungsspannung an das Display, sollte der Mikrocontroller ca. 15 ms warten bevor er mit der Initialisierung des HD44780 beginnt. Während der Power On Phase der Initialisierung werden die folgenden Schritte durchgeführt:

1. Display löschen

2. Funktion: DL= 0: 4-Bit Datenbusbreite
N = 1: 2-Zeilen Display
F = 0: 5x7 Zeichensatz

3. Display ON/OFF

Funktion: D = 0: Display aus
C = 0: Cursor aus
B = 0: Blinkfunktion aus

4. Entry mode set

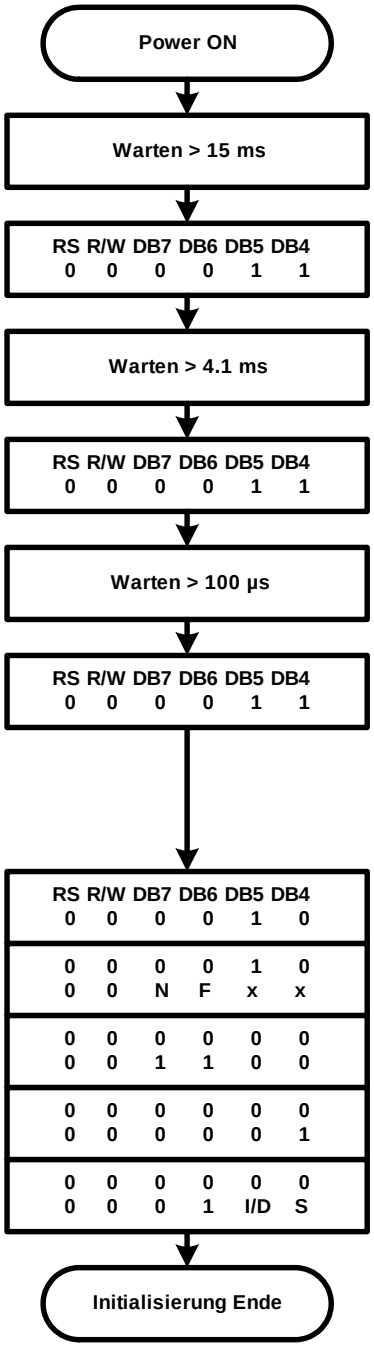
I/D = 1: Inkrementiert die DD-RAM Adresse nach dem Lesen/Schreiben

eines Zeichens

S = 0: Kein schieben des Displays

5. Schreiben des DD-RAM

Als erster Schritt erfolgt die Festlegung der Datenbusbreite. Während dieser Phase der Initialisierung werden nur die oberen Datenbits beachtet. Nachfolgend ein Ablaufdiagramm der 4-Bit-Initialisierungsroutine.



Wartezyklus

Busy Flag kann an dieser Stelle noch nicht abgefragt werden, da der 8-Bit Modus hier noch aktiv.

Wartezyklus

Busy Flag kann an dieser Stelle noch nicht abgefragt werden, da der 8-Bit Modus hier noch aktiv.

Wartezyklus

Busy Flag kann an dieser Stelle noch nicht abgefragt werden, da der 8-Bit Modus hier noch aktiv.

Ab dieser Position kann das Busy-Flag abgefragt werden. In diesen beiden Schritten wird der 4-Bit-Modus gesetzt. Einstellen der verwendeten Zeilen und des Zeichensatzes.
Zu beachten ist die Zeilenanzahl und der Zeichensatz können ab hier nicht mehr verändert werden.

Display auf An setzen

Display löschen

Entry-mode setzen

Sollte keine Anzeige auf dem Display erscheinen, bitte folgende Fragen beachten

1. Ist die Kontrastspannung richtig eingestellt?
2. Wurde die Initialisierung richtig durchgeführt?
3. Wurden die Pausenzeiten zwischen den einzelnen Schritten eingehalten?

Tabelle 10: Befehlssatz des Displaycontrollers HD44780

Instruktion	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Home	0	0	0	0	0	0	0	0	1	x
Entry mode set	0	0	0	0	0	0	0	1	I/D	S
Display ON/OFF	0	0	0	0	0	0	1	D	C	B
Cursor und Display shift	0	0	0	0	0	1	S/C	R/L	x	x
Funktion Set	0	0	0	0	1	DL	N	F	x	x
CG RAM Adresse setzen	0	0	0	1	CG-RAM Adresse					
DD RAM Adresse setzen	0	0	1	DD-RAM Adresse						
Busy Flag und Adresse lesen	0	1	BF	Adresszähler für CG-RAM und DD-RAM						
Daten in CG- oder DD-RAM schreiben	1	0	Zu schreibende Daten							
Daten aus CG- oder DD-RAM lesen	1	1	Gelesene Daten							

Beschreibung der eingesetzten Befehle:

Die Funktion der in Tabelle 10 zu sehenden Befehle, werden nachfolgend erläutert.

***Return* Home**

Setzt den Adresszähler des DD-RAM auf Adresse 0. Der Inhalt des DD-RAM bleibt unverändert. Der Cursor wird an die erste Position der ersten Zeile gesetzt.

Entry mode set

I./D: 1 = Adresspointer inkrementieren

0 = Adresspointer dekrementieren

S: 1 = Displayinhalt schieben,

0 = Displayinhalt nicht schieben.

Schreiben in DD-RAM verschiebt den Displayinhalt, DD-RAM lesen verschiebt nicht. Der Cursor bleibt an derselben Stelle stehen. Lesen oder schreiben in das CG_RAM hat ebenfalls keinen Einfluss auf den Displayinhalt.

Display ON/OFF

D: 1 = Display an 0 = Display aus

C: 1 = Cursor unsichtbar 0 =Cursor sichtbar

B: 1 = Zeichen unter dem Cursor blinkt 0 = Blinken aus

Cursor und Display shift (Tabelle 11)

Tabelle 11: Cursor- und Displayeinstellungen für die Bits S/C und R/L

S/C	R/L	Funktion
0	0	Bewegt den Cursor um eine Stelle nach links ohne das DD-RAM zu verändern
0	1	Bewegt den Cursor um eine Stelle nach rechts ohne das DD-RAM zu verändern
1	0	Verschiebt Displayinhalt und Cursor um eine Stelle nach links ohne das DD-RAM zu verändern
1	1	Verschiebt Displayinhalt und Cursor um eine Stelle nach rechts ohne das DD-RAM zu verändern

Funktion Set

Die Datenbusbreite, Anzahl Zeilen und der Zeichensatz können nur währen der Initialisierungsphase des Controllers gesetzt werden.

DL: 1 = 8-Bit Interface 0 = 4-Bit Interface

N: 1 = 2 Zeilen Display 0 = 1 Zeilen Display

F: 1 = 5x10 Zeichenfont 0 = 5x7 Zeichenfont

Busy Flag und Adresse lesen

BF: 1 = Controller arbeitet gerade eine interne Operation ab

0 = Controller akzeptiert neue Instruktionen

Das Busy Flag sollte vor jeder Schreiboperation ausgelesen werden, um sicherzustellen, dass der Controller bereit ist.

Daten in CG- oder DD- RAM schreiben

Es werden 8-Bit Daten zum CG- oder DD-RAM geschrieben. Das Ziel des Transfers hängt davon ab ob der Befehl „Set CG-RAM Adresse“ setzen oder „Set DD-RAM Adresse“ verwendet wurde.

Adresse inkrementiert (I/D = 1) bzw. dekrementiert (I/D = 0) geschieht automatisch.

Daten aus CG- oder DD- RAM lesen

Es werden 8-Bit Daten vom CG- oder DD-RAM gelesen. Das Ziel des Transfers hängt davon ab, ob der Befehl „Set CG-RAM Adresse“ setzen oder „Set DD-RAM Adresse“ verwendet wurde.
Adresse inkrementiert (I/D = 1) bzw. dekrementiert (I/D = 0) geschieht automatisch.
Das Bild 4-14 zeigt den Standardzeichensatz in einer Übersicht.

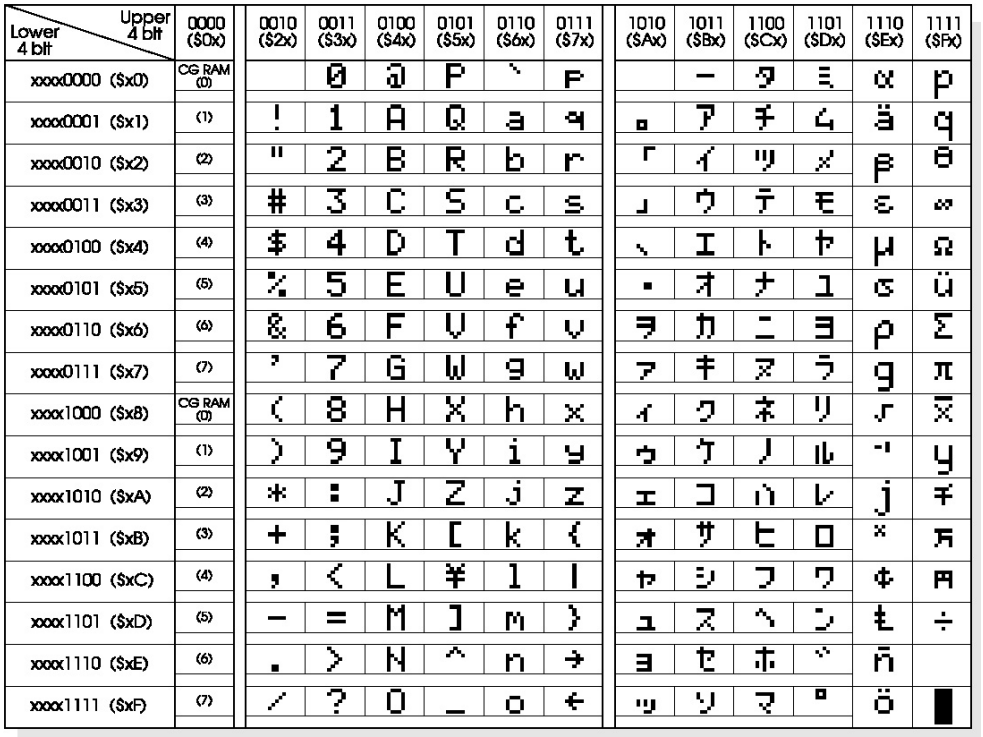


Bild 4-14: Standardzeichensatz des Displays

Weitere Einstellungen und Informationen zu diesem Display können aus dem zugehörigen Datenblatt [17] entnommen werden. In diesem Datenblatt wird ebenfalls die genaue Vorgehensweise beim Entwerfen und Darstellen eigener Zeichen beschrieben.

4.9 Lautsprecher

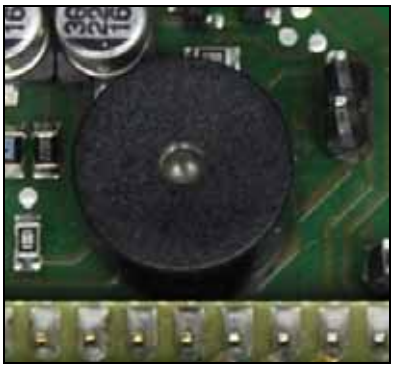


Bild 4-15: Eingesetzter Lautsprecher

Der Lautsprecher in Bild 4-15 wird mit einer einfachen KomplementärGegentakt-B-Endstufe angesprochen. Dies ist die einfachste Verstärkerschaltung zum Ansprechen eines Lautsprechers. Der Lautsprecher dient ebenfalls zum Ausgeben von Puls-Weiten-Modulierten Signalen, hier jedoch in Form von Tönen. Töne können natürlich auch ohne Puls-Weiten-Modulation erzeugt und an diesem Lautsprecher ausgeben werden. Hierzu wird einfach ein Signal mit fest eingestellten Pulsweiten und fester Frequenz an Port P2.4 ausgegeben.

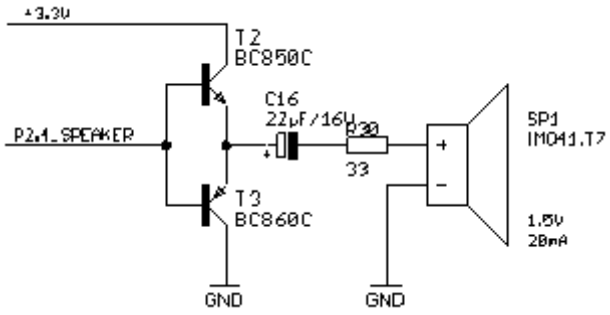


Bild 4-16: Beschaltung des Lautsprechers

Tabelle 12: Anschlussbelegung des Lautsprechers

Port	Stiftleiste	µC Pin	Peripheriesystem
P2.4	X8_8.9	20	Ansteuerung des Lautsprechers LAU1

4.10 Potentiometer

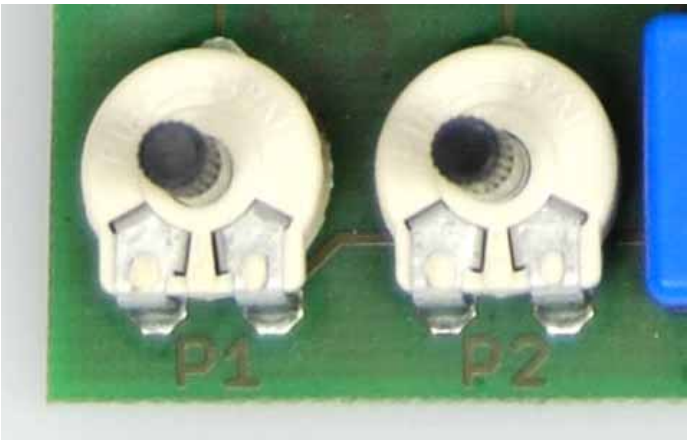
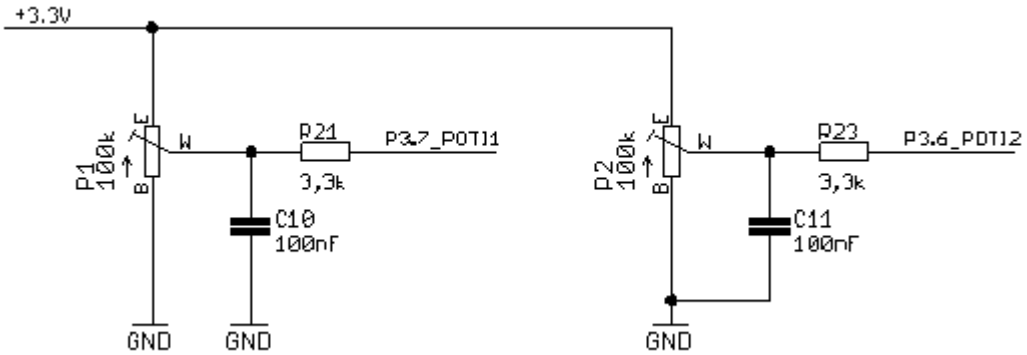


Bild 4-17: Installierte Potentiometer

Die beiden Potentiometer P1 und P2, wie in Bild 4-17 gezeigt, sind linear einstellbare Widerstände mit einem Widerstand von je 100 kOhm. Mit diesen Potentiometern soll die Verwendung des Analog-Digitalwandlers vom MSP430 erlernt werden. In einer Applikation können diese Potentiometer bestimmte Parameter dynamisch ändern. Das Verändern von Tönen ist genauso möglich, wie das Anzeige des AD-Wandler-Ergebnisses auf den LEDs.

Ohne einen Mikrocontroller wäre dies natürlich auch möglich, doch kann hier das Zusammenspiel mehrerer ganz unterschiedlicher Komponenten in einer Applikation erlernt werden. Durch Umprogrammierung können in dieser Lösung sehr einfach die Potentiometer vertauscht oder mit anderen Parametern belegt werden. Bei einem festen Aufbau, wäre dies nicht ohne erheblichen Aufwand möglich.



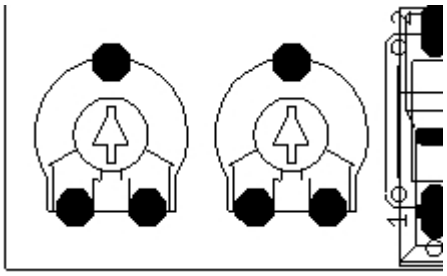


Bild 4-18: Beschaltung der Potentiometer

Die Widerstände R21 und R23 dienen als Schutz des Mikrocontrollers gegen Kurzschluss. Das Bild 4-18 und die Tabelle 13 beschreiben die wichtigsten Anschlüsse und Positionen dieser zwei Potentiometer.

Tabelle 13: Anschlussbelegung der Potentiometer

Port	Stiftleiste	µC Pin	Peripheriesystem
P3.6	X8.12	17	Potentiometer 1
P3.7	X8.11	18	Potentiometer 1

4.11 Reset-Beschaltung des MSP430

Der MSP430F1232 verfügt über einen internen Reset-Controller, der den Einsatz eines externen Reset-Controllers unnötig macht und somit Kosten bei der Entwicklung einer eigenen Applikation spart.

Durch Druck auf den Taster S5 wird im Controller ein Reset ausgelöst. Vom JTAG-Adapter kommt ebenfalls ein Reset-Signal, das zum Zurücksetzen genutzt werden kann. Bild 4-19 zeigt die Beschaltung und Lage des Reset-Tasters.

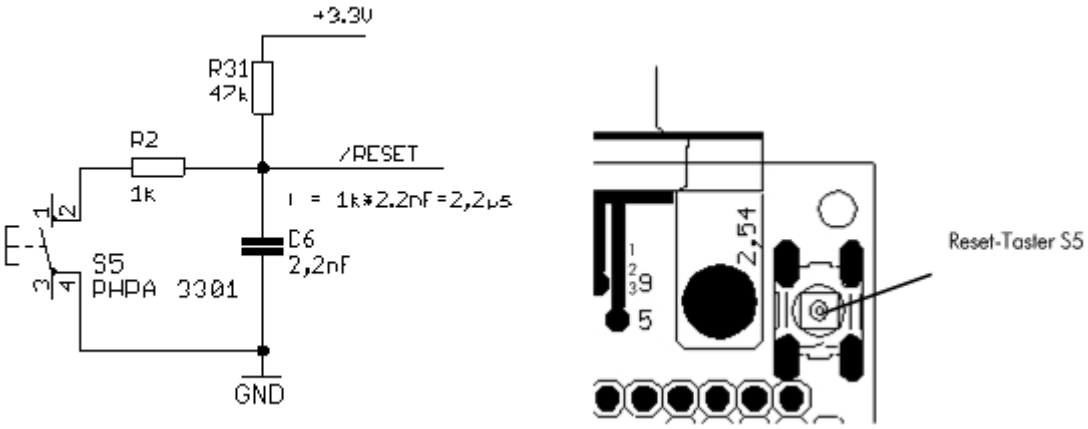


Bild 4-19: Resetbeschaltung des MSP430

4.12 Lötbrücken des MSP430 Education Systems

Das MSP430 Education System erlaubt, durch Aktivieren bzw. Deaktivieren von Lötbrücken verschiedene Funktionalitäten zu beeinflussen. Die Einstellmöglichkeiten der einzelnen Lötbrücken zeigt Tabelle 14 im Überblick.

Name	Beschreibung	Default
SJ1	Beschaltung von Pin1 am RS-232 Levelconverter. Wird bei Bestückung von alternativem IC benötigt	geschlossen
SJ2	Verbindung zwischen Mikrocontroller und RS-232-Levelconverter (TX)	geschlossen
SJ3	Verbindung zwischen Mikrocontroller und RS-232-Levelconverter (RX)	geschlossen
SJ4	Aktivierung der 5V Spannungsversorgung	geschlossen
SJ5	Aktivierung des Infrarot-Empfängers	geschlossen
SJ6	Aktivierung des Infrarot-Senders	geschlossen

Tabelle 14: Funktionsübersicht aller Lötbrücken

Die Lage der Lötbrücken zeigt das Bild 4-20.

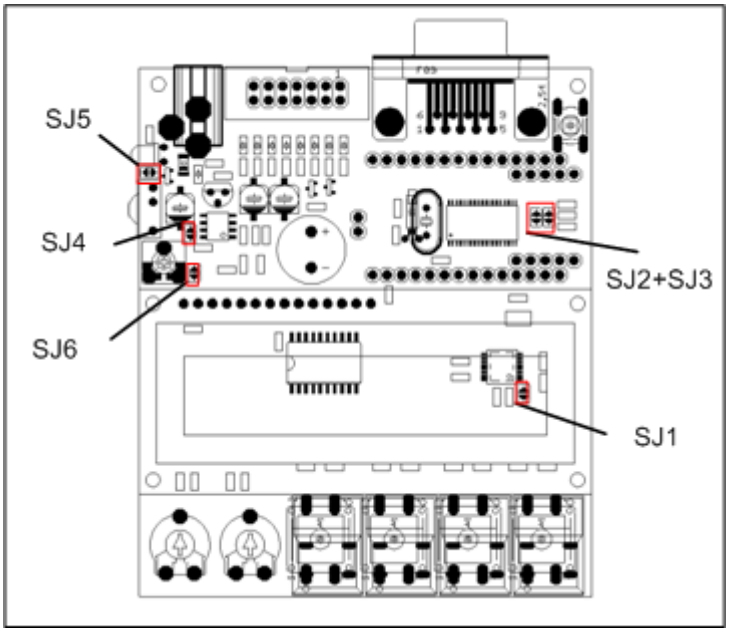


Bild 4-20: Lage der Lötbrücken Top-Ansicht

5 Softwarebeschreibung

In diesem Kapitel wird die Entwicklungsumgebung IAR Embedded Workbench von IAR Systems [2] kurz beschrieben und erklärt. Die Software kann als Kickstart Version von der Texas Instruments Homepage [1] oder der IAR Systems Homepage [2] kostenfrei herunter geladen werden.

Es wurde sich für diese Entwicklungsumgebung entschieden, da sie Assembler- und C-Programmierung in einer Umgebung vereint, einen umfangreichen Debugger enthält, kostenfrei genutzt werden kann und von Texas Instruments direkt unterstützt wird (JTAG-Funktionalität von TI selbst implementiert).

Da es sich bei der IAR Embedded Workbench grundsätzlich um einen kommerziellen Compiler handelt, muss der Anwender einige Abstriche bei der Kickstart Version in Kauf nehmen. Einzig der Assemblerteil kann in seiner vollen Funktionalität verwendet werden. Der C-Compiler ist jedoch auf 4 kByte C-Code begrenzt.

Die Entwicklungsumgebung enthält zusätzlich ein sehr umfangreiches Handbuch und eine guten Hilfe-Funktion, die bei den meisten Fragen weiterhelfen kann.

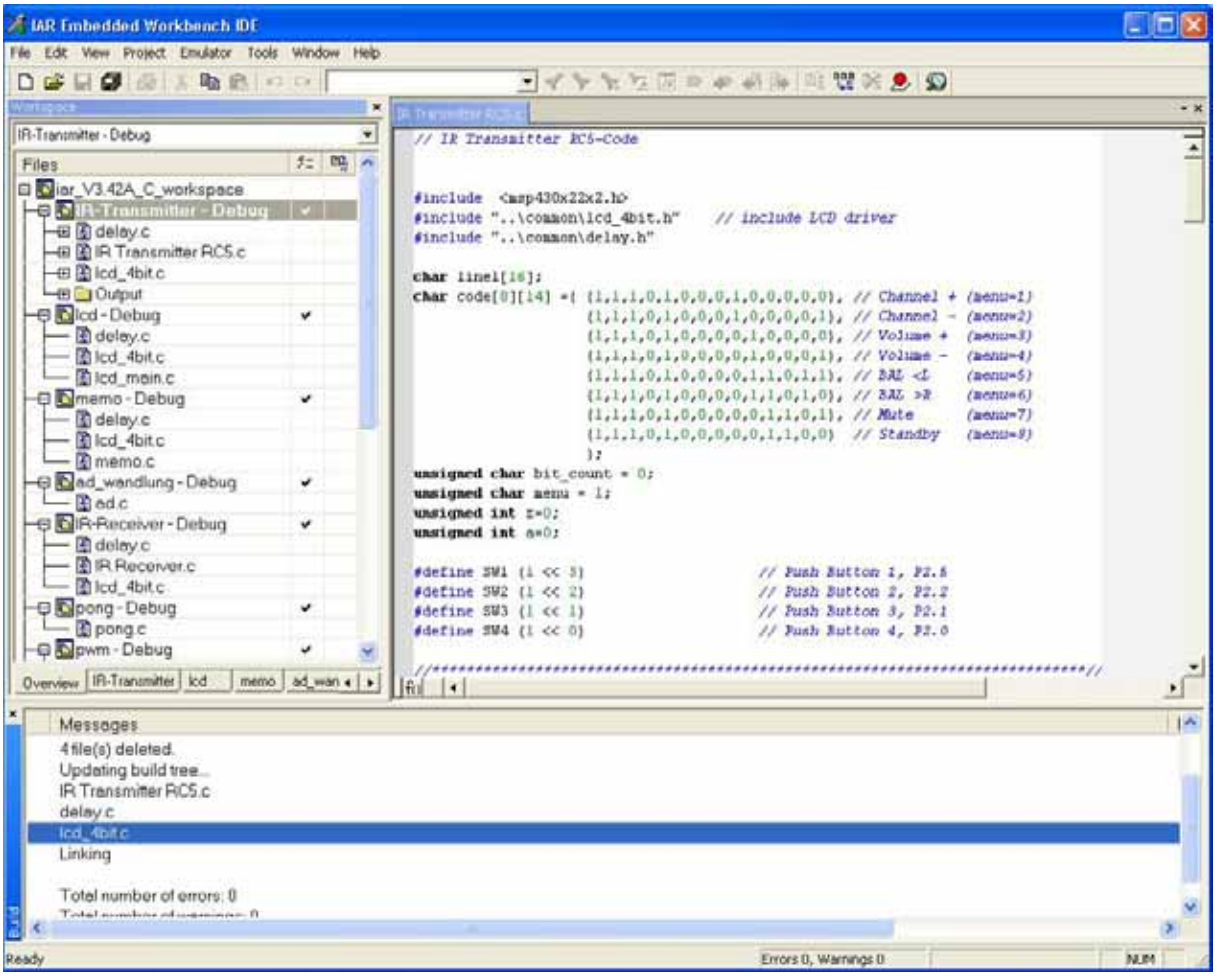
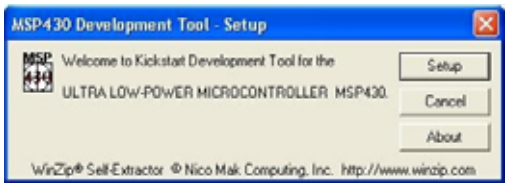


Bild 5-1: Entwicklungsumgebung IAR Embedded Workbench

5.1 Installation der Software

Nachfolgend wird an Hand der Tabelle 15 die Installation der Software Schritt für Schritt erklärt. Sollte es dennoch zu Problemen kommen, können Sie die beiliegende Dokumentation zu Hilfe nehmen oder den Support der Firmen Texas Instruments und IAR Systems nutzen.

Tabelle 15: Installationsschritte in einer Übersicht

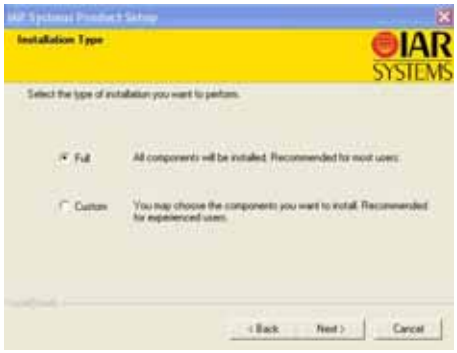


1. Nach Doppelklick auf das Installationsfile erscheint folgendes Fenster. In diesem Fenster klickt man auf den **Setup**-Button.

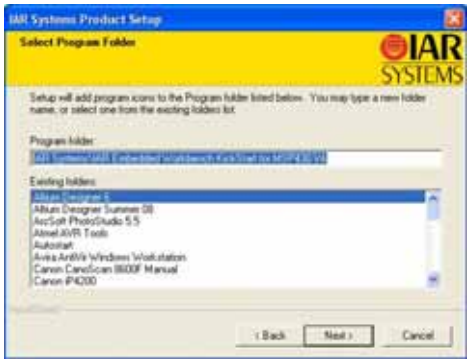
2. Es öffnet sich folgendes Fenster, in dem man einzelne Informationen zum Produkt nach lesen kann. Man klickt auf **Next**.

3. Es folgt ein Fenster in dem die Lizenz-Vereinbarungen aufgeführt sind. Man sollte diese Informationen genau lesen, denn diese Lizenzvereinbarungen muss man durch klicken auf den **Accept**-Button bestätigen, um zum nächsten Fenster zu gelangen.

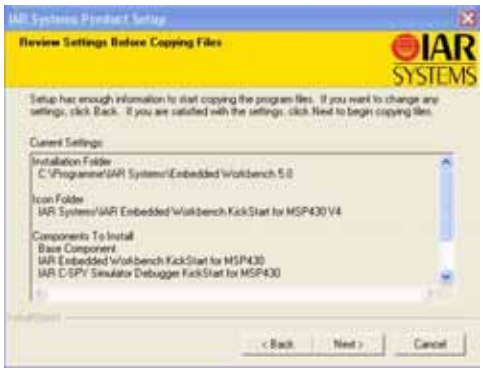
4. In diesem Fenster wählt man den Ort der Installation. Klickt man auf *Browse* kann man den Ort der Installation selber bestimmen. Da sich jedoch alle weiteren Beschreibungen auf den Standardordner beziehen sollte dieser auch eingestellt bleiben. Man klickt auf **Next**.



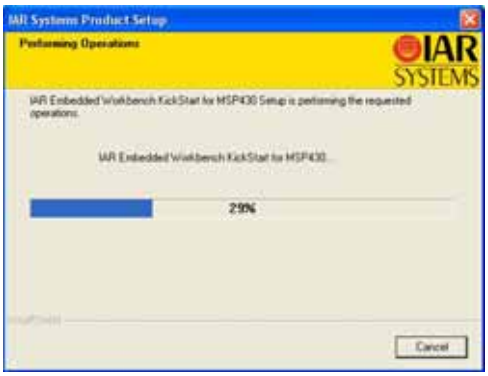
5. Durch klicken auf **Custom** kann man die Installationsparameter verändern. Es wird empfohlen das volle Packet zu installieren.



6. In diesem Fenster kann man den Eintrag in dem Start-Programme-Ordner verändern. Man klickt auf **Next** um zum nächsten Fenster zu gelangen.



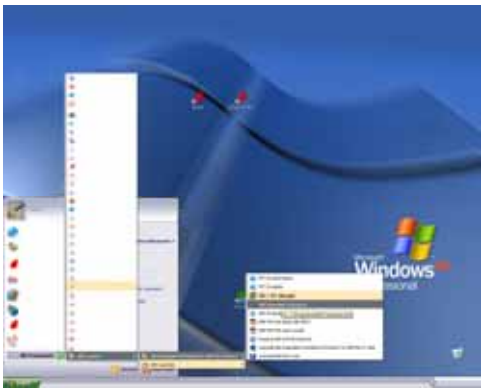
7. In diesem Fenster werden alle gewählten Features nochmals dargestellt. Durch klicken auf **Next** startet man die Installation.



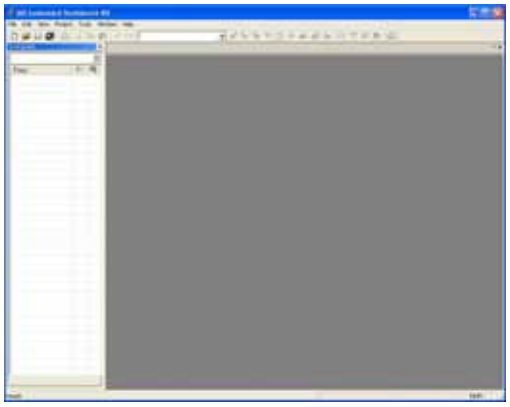
8. Installationsfortschritt. Durch drücken von **Cancel** kann man die Installation abbrechen.

9. Nach erfolgreicher Installation erscheint folgendes Fenster. Die Häkchen lässt man gesetzt, um die **Readme**-Datei anzuzeigen und weitere Tools zu installieren. **Finish** beendet die Installation des Hauptprogramms.

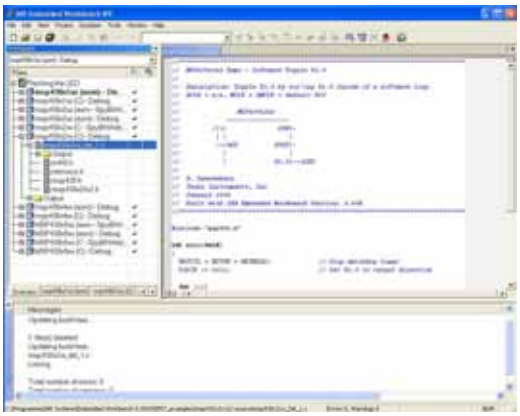
10. Nach dem betätigen des **Finish**-Button erscheint die **Readme**-Datei im HTML Browser.



11. Nach dem Neustart des Computers wählt man aus dem **Programme-Ordner** des Starmenüs den Punkt **IAR Systems**, dort den Unterpunkt IAR **Embedded Workbench for...** und danach den Unterpunkt **IAR Embedded Workbench**. Durch klicken auf diesen Unterpunkt öffnet sich die Programmierumgebung.



12. Dies ist das Bild was sich nach dem öffnen bietet. Man sieht die Standardoberfläche mit den Menüleisten am oberen Rand.



13. In diesem Bild sieht man die Programmierumgebung in Verwendung. Wie man zu diesem Bild kommt, zeigt das nächste Kapitel.

Damit ist die Installation der Software an dieser Stelle abgeschlossen.

Wurde die Installation wie beschrieben durchgeführt, befinden sich nun mehrere wichtige Dateien und User Guides im Programmordner der IAR Embedded Workbench. Eine Liste der wichtigsten User Guides zeigt Tabelle 16. Diese Dateien befinden sich, die oben beschriebene Installation vorausgesetzt, in folgendem Ordern des Laufwerks C: C:\Programme\IAR Systems\ew23\430\doc\. In diesem Ordner befinden sich viele weitere Dateien, die während der Programmierung weiter helfen können.

Tabelle 16: Wichtige User Guides und Handbücher

Dateiname	Dokumentname
a430.html	Release notes for MSP430 IAR Assembler V3.42A
clib.pdf	IAR C LIBRARY FUNCTIONS
EW430_AssemblerReference.pdf	MSP430 IAR Assembler
EW430_CompilerReference.pdf	MSP430 C/C++ Compiler
EW430_UserGuide.pdf	MSP430 IAR Embedded Workbench IDE User Guide
icc430.htm	Release notes for the MSP430 IAR C/C++ Compiler
MSP-FET430 Errata.pdf	MSP430 Flash Emulation Tool User Guide
MSP-FET430 Users Guide.pdf	MSP-FET430 FLASH Emulation Tool (FET) Users Guide
readme.htm	Release notes for IAR Embedded Workbench for MSP430

Nun kann ein Beispielprojekt aus dem IAR Programmordner gewählt werden. Die Schritte die hierfür nötig sind, zeigt das nächste Kapitel.

5.2 Hauptbestandteile der IAR Embedded Workbench

Um ein Beispielprojekt laden, kompilieren und übertragen zu können, ist es wichtig, die einzelnen Symbole der Programmierumgebung, zu kennen. Deshalb werden in diesem Abschnitt einzelne Bestandteile der Software kurz erklärt, damit sich der Anwender, nach erfolgreicher Installation, orientieren kann.

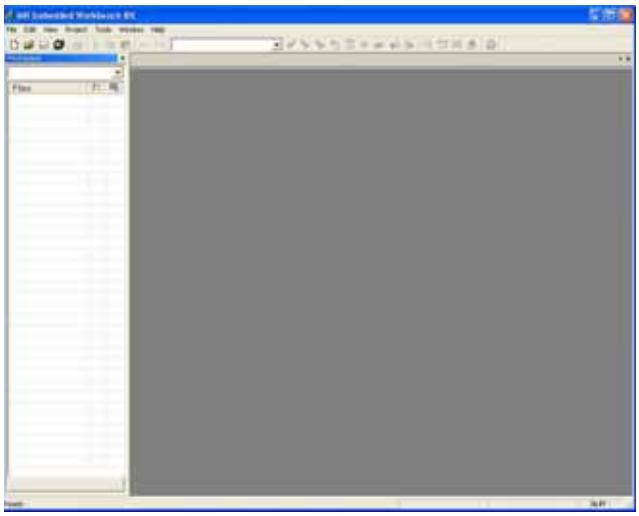


Bild 5-2: Bildschirm nach Start der Software

Nach dem Start der Software durch klicken auf das IAR Embedded Workbench Symbols im Startmenü zeigt sich dem Benutzer folgender Bildschirm wie in Bild 5-2 zu sehen.

Die Programmieroberfläche besteht aus 3 großen Fenstern wie in das Bild 5-3 zeigt. Diese drei Fenster bilden die Programmierzentrale für jeden Anwender. Das **Workspace-Fenster** stellte alle **Projekte** und auch zu dem Projekt zugehörende Dateien, die in das **Workspace** eingebunden sind übersichtlich dar. Und zeigt zu dem nach einer Compilierung ebenfalls alle eingebundenen Headerfiles. Diese Headerfiles sind, Standardbibliotheken, die nach Einbinden in das Hauptprogramm ihre Vereinbahrungen zur Verfügung stellen.

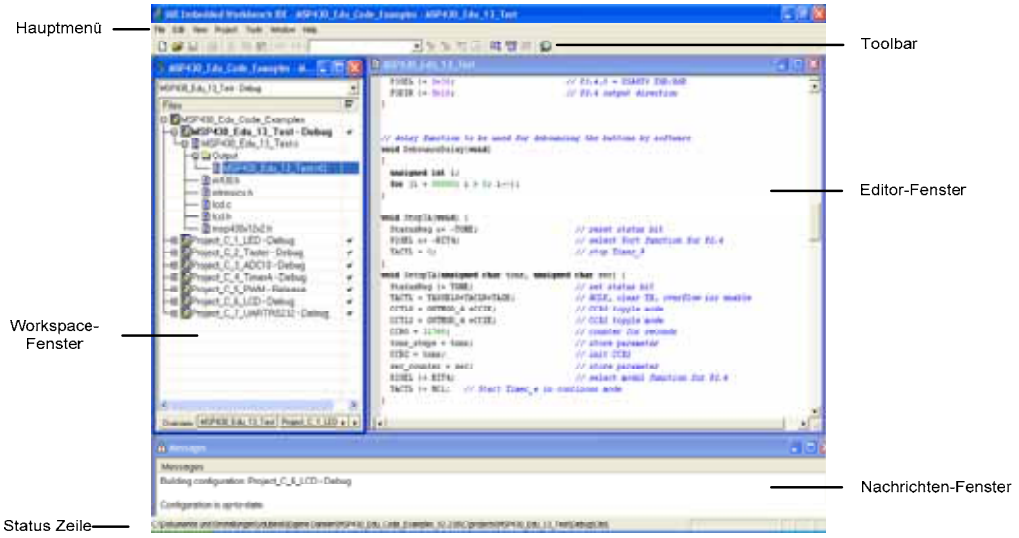


Bild 5-3: Hauptbestandteile der Programmierumgebung

Das **Editor-Fenster** ist das eigentliche Programmier-Fenster. In diesem Fenster werden alle Programme geschrieben und entworfen. Dieses Fenster stellt auch die Ausgangsbasis für eigene Programme dar.

Das **Nachrichten-Fenster** stellt die Schnittstelle vom Programm zum Anwender dar. Alle relevanten Daten die das Programm dem Anwender zurückgibt, werden in diesem Fenster dargestellt. Es berichtet zum einen über Fehler, Fehlerart und Ort des Fehlers und zum anderen über Suchergebnisse und sonstige wichtige Meldungen. Nach dem compilieren eines Programms, werden alle wichtigen Informationen in diesem Fenster angezeigt.

Weitere Einstellungen werden in der Hauptmenü-Zeile und den entsprechenden Untermenüs zur Verfügung gestellt. Die genaue Funktionsweise jedes Unterpunktes lässt sich in den zugehörigen Handbüchern der Programmierumgebung IAR Embedded Workbench for MSP430 [11] und [15] nachlesen.

Das Hauptmenü stellt den Zugang zu mehreren Grundfunktionen dar, in denen projektspezifische Einstellungen getätigt werden können. Nicht minder wichtig sind die Buttons der Toolbars. Sie nehmen dem Programmierer sehr viel Arbeit ab und gestalten das Programmieren sehr übersichtlich. Wichtige Symbole und Buttons der Programmierumgebung zeigt das Bild 5-4. Alle Funktionen der Buttons sind ebenfalls im zugehörigen Handbuch nachzulesen.

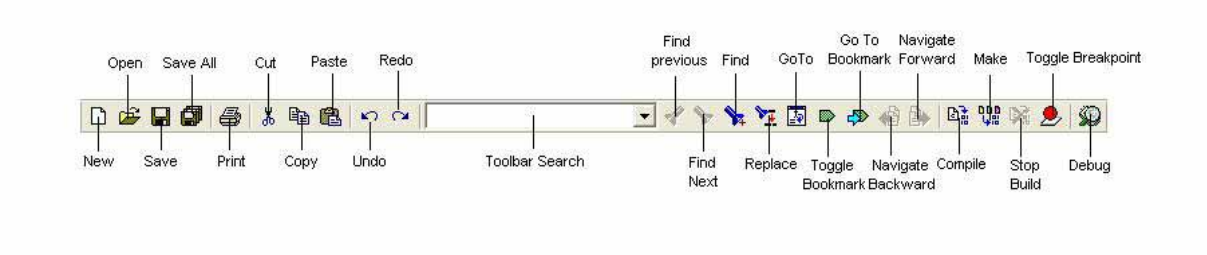


Bild 5-4: Funktionen der Toolbar Buttons

Das zweite wichtige Teil dieser Programmierumgebung ist der Debugger. Mit Hilfe des Debuggers kann der erzeugte Programmcode auf den Mikrocontroller geladen und dort auch ausgeführt werden. Es stellt somit Download- und Debug-Software in einer Umgebung dar. Denn Programmcode kann der Nutzer im Debugger schrittweise durchlaufen und somit debuggen. In Bild 5-5 werden die wichtigsten Bestandteile des Debuggers übersichtlich dargestellt. Diese Fenster helfen, um den geschriebenen Programmcode von Fehlern zu bereinigen. Nachfolgend werden alle Fenster kurz beschrieben um die entsprechende Funktionsweise verstehen zu können. Eine ausführliche Beschreibung findet sich in den zugehörigen Handbüchern [9] und [10].

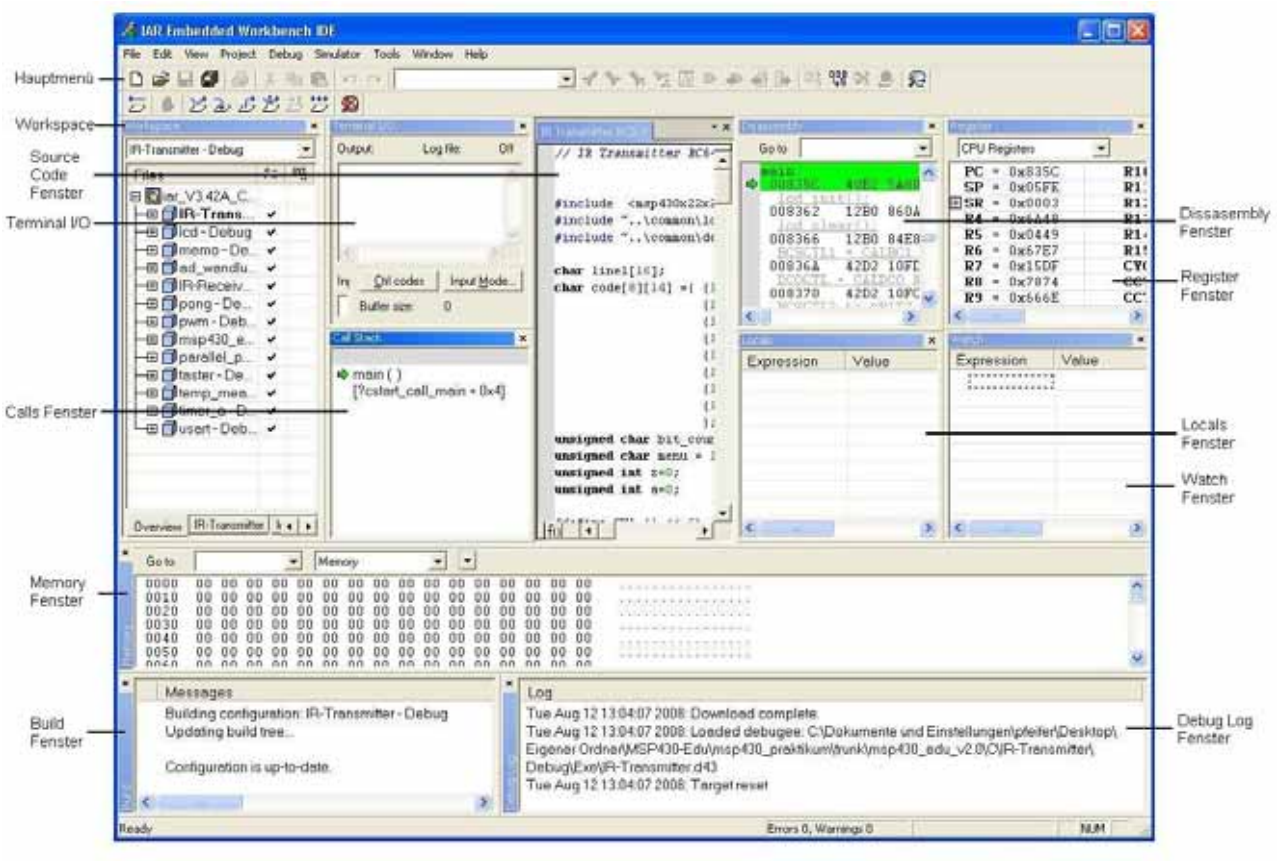


Bild 5-5: Hauptbestandteile des Debuggers

Das **Source Code Fenster und das Disassembly Fenster** stellt den zu debuggenden C- oder Assemblercode übersichtlich dar. Durch das Betätigen des Memory Buttons im Desassembly Fenster, kann zwischen der Memory-, RAM-, FLASH-, SFR- und INFO-Ansicht gewechselt werden. Durch Rechtsklicken in diesem Fenster öffnet sich ein Menü, in dem sehr hilfreiche Optionen zum Debuggen des Sourcecodes gefunden werden können.

Im **Calls-Fenster** wird der Speicherort der Stacksicherung, eines in C geschriebenen Programms dargestellt. Diese Angabe hat folgendes Format: *module\function(values)*.

Das **Locals-Fenster** zeigt automatisch alle lokalen Variablen und alle dazugehörigen Werte übersichtlich an. Mit diesem Fenster läst sich das vorhanden sein bestimmter Variablen und ihrer aktuellen Werte überprüfen.

Im **Watch-Fenster** kann der Anwender sich bestimmte C-Variablenwerte anzeigen lassen, die einfach zugefügt werden können. Dieses Fenster zeigt alle aufgenommenen Variablen mit den jeweils aktuellen Werten dieser Variablen an und ist sehr nützlich bei der Suche von Fehlern im geschriebenen Programmcode. Der Anwender legt sich lediglich eine Liste der interessanten Variablen an und hat immer die Kontrolle über deren aktuelle Werte.

Das **Memory-Fenster** gibt die Speicherbelegung des intern verwendeten Speichers wieder. Es zeigt die Adresse und die Belegung des Speicherblockes sowie die darin gespeicherten Information. Der

Speicher kann in 8,16 oder 32-Bit Organisation anzeigt werden. Durch Doppelklick auf einen Speicherblock öffnet sich ein Fenster in dem jede Speicherzelle vom Anwender gezielt verändert werden kann. Dieses Fenster bietet eine weitere Möglichkeit, den geschrieben Programmcode auf dem Mikrocontroller zu beeinflussen.

Im **Register-Fenster** werden alle controllerspezifischen Registerinhalte kontinuierlich angezeigt und können auch dort direkt beeinflusst werden. Dies bedeutet, dass während der Debugphase Registerinhalte auf dem Controller verändern werden können und somit ein direkter Einfluss auf das laufende Programm besteht. Der Registerwert läst sich in dem zugehörigen Kästchen verändern. Diese Änderung wird nach dem Wechsel zu einem anderen Register übernommen.

Das **Terminal I/O Fenster** ermöglicht die Eingabe der Daten in das Source-Programm, und lässt den Ausgang vom Source-Programm anzeigen.

Dieser Überblick sollte jedem Anwender die wichtigsten Fenster näher gebracht haben, um den Debugger gezielt einsetzen zu können. Weitere Informationen können ebenfalls aus den zugehörigen Handbüchern oder der Hilfedatei entnommen werden. Der Debugger besitzt weiterhin eine nützliche Toolbar. Mit Hilfe dieser Toolbar ist es möglich, schnell die wichtigsten Optionen und Funktionen des Debuggers zu erreichen und einzelne Einstellungen für den Debugvorgang zu treffen. Die Toolbar stellt die zentralen Debug- und Abarbeitungsfunktionen zur Verfügung. Der Mikrocontroller kann durch einen einfachen Mausklick gestartet oder genauso wieder gestoppt werden. Das Setzen und Entfernen von Breakpoints gelingt mit einem Mausklick. Durch einen weiteren Mausklick kann dieser Breakpoint ebenso einfach erreicht werden. Das Bild 5-6 gibt einen gesamten Überblick der zur Verfügung stehenden Buttons und Symbole und weiterhin eine Kurzbeschreibung zu jedem Punkt.

Wie in diesem Bild zu sehen, gibt es natürlich noch viele weitere Symbole und Buttons, die jedoch in dem zugehörigen Handbuch sehr gut erklärt und beschrieben werden.

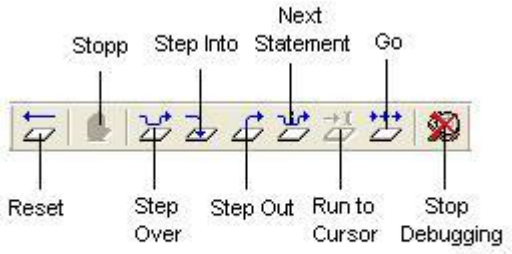


Bild 5-6: Funktionen der Toolbars des Debuggers

Die Programmierumgebung IAR Embedded Workbench stellt eine leistungsfähige Programmierumgebung mit sehr gutem Debugger dar. Die einzelnen Funktionen der Buttons und Symbole zu kennen vereinfacht die Programmierung ebenso stark, wie das beherrschen der Programmiersprachen selbst. Damit sind die Grundlagen für das weitere Arbeiten mit dieser Programmierumgebung gelegt.

Alle Bestandteile und Funktionen werden in den installierten Handbüchern und Zusatzdateien, wie in Tabelle 16 aufgelistet, genau beschrieben. Das Verwenden dieser Dateien ist ebenso wichtig, wie das Verwenden dieses Handbuchs, da nur durch Kombination aller Bestandteile (Hardware und Software) ein gut funktionierender Entwicklungsaufbau realisiert werden kann.

5.3 Erste Schritte

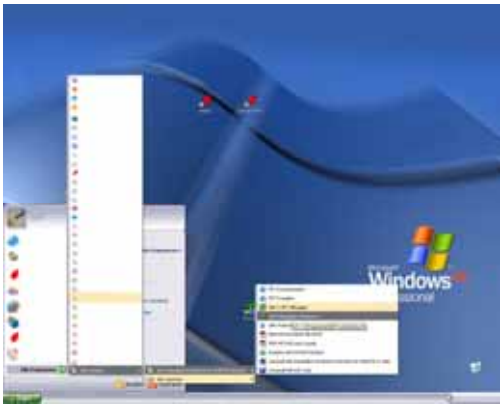
Nachdem in den vorherigen Kapiteln die Hard- und Software genau beschrieben wurde, steht in diesem Kapitel die Anwendung beider Bereiche im Vordergrund. Nachfolgend werden der richtige Aufbau und die richtige Verwendung des MSP430 Education System genau beschrieben. Die Kenntnisse der vorherigen Kapitel werden hier vorausgesetzt und benötigt.
Es werden der richtige Funktionsaufbau, das Laden eines bestehenden Beispielprogramms, das Erstellen eines eigenen Programms und der Programmdownload auf den Mikrocontroller Schritt für Schritt beschrieben.

5.4 Laden eines Beispielprojektes

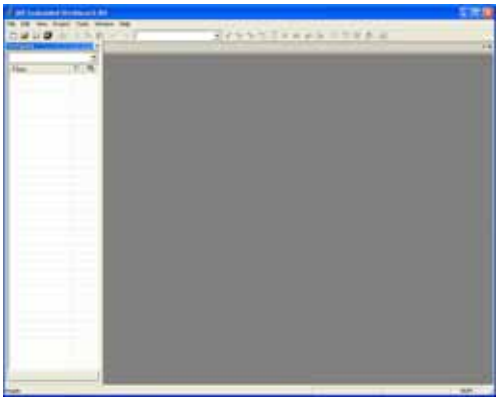
Um die Softwareinstallation nun zu testen, kann ein Beispielprojekt aus dem IAR Programmordner geladen, bearbeitet ausgeführt und werden. Eine Fülle von Beispielprogrammen wird während der Installation in den IAR Programmordner installiert. Auf der mitgelierten CD zum MSP Education System stehen weiter Beispielprogramme in C und Assembler zur Verfügung mit denen die einzelnen Funktionen der einzelnen Komponenten getestet werden können. Das Finden und Öffnen dieser Dateien innerhalb der IAR Embedded Workbench wird in der nachfolgenden

Tabelle 17 detailliert beschrieben und erläutert. Werden die einzelnen Schritte befolgt, ist der Anwender in der Lage sein erstes Projekt zu öffnen und zu erstellen. Dies ist der erste wichtige Schritt in Richtung eigene Applikationen.

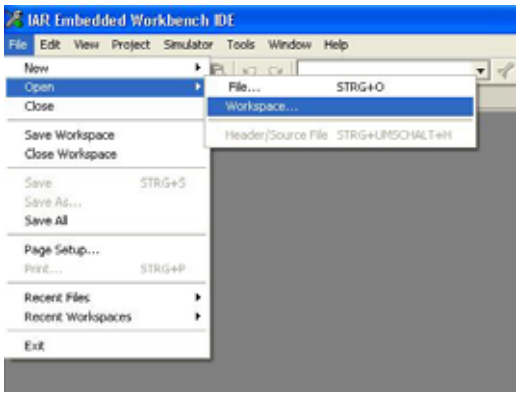
Tabelle 17: Laden eines vorhanden Beispielprojektes



1. Man startet die **IAR Embedded Workbench** aus dem Startmenü heraus.

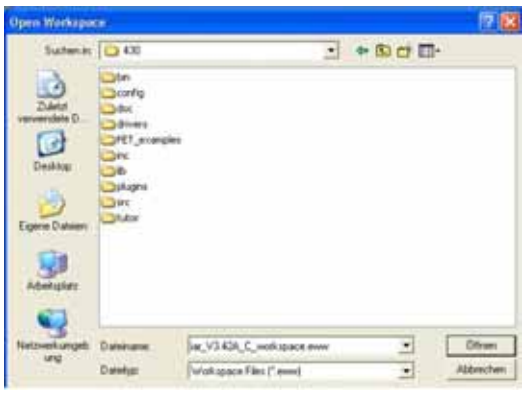


2. Nach dem Start der Software, zeigt sich dem Anwender dieses Bild.

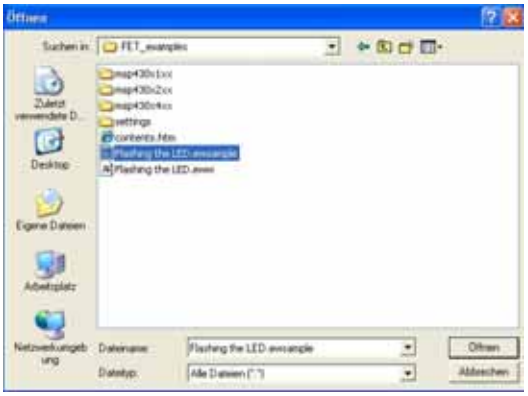


3. Im Menü wählt man **File** um dann im Untermenü **Open** einen neuen **Workspace** zu öffnen.

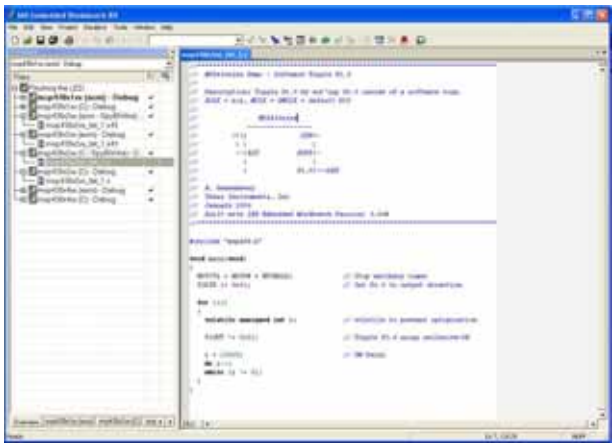
4. Es erscheint folgendes Fenster. Man wählt den **IAR Systems** Ordner im Windows Programme Ordner. Voraussetzung ist ebenfalls, dass die Installation wie in Kapitel 4.1 beschrieben durchgeführt wurde. Man klickt doppelt auf den **IAR Systems** Programmordner.



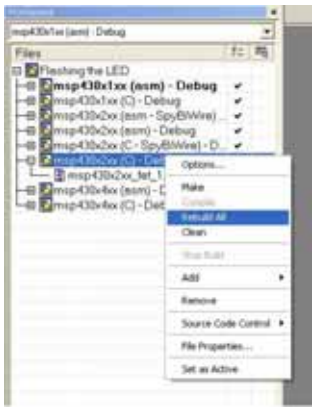
5. Nach einem Doppelklick auf den **ew23** Ordner und einen weiterer Doppelklick auf den **430** Ordner zeigt sich dem Anwender folgendes Fenster. Man wählt den Ordner mit dem Namen **FET_examples**.



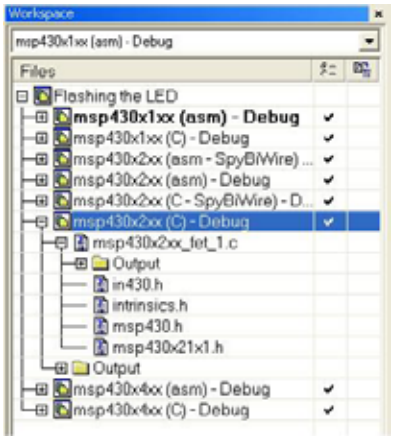
6. In diesem Ordner findet man den Workspace unter dem Namen „**fet_projects.eww**“. Dieses File beinhaltet alle Beispielprogramme, die in C und in Assembler geschrieben sind.



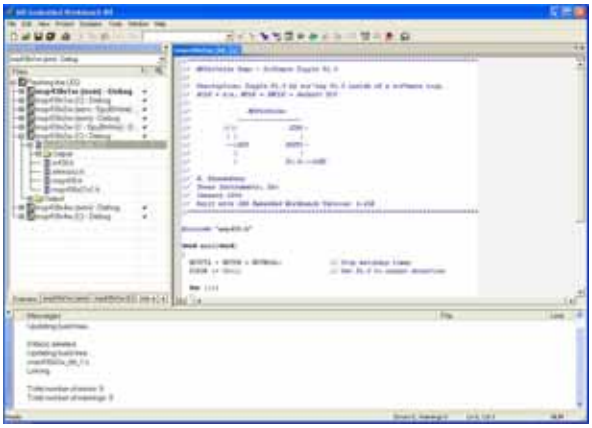
7. In diesem Workspace findet man mehrere Projekte mit dem Namen **msp430x**. Diese Projekte sind für verschiedene MSP430 Typen vorgesehen. Da hier ein MSP430F2272 zum Einsatz kommt, wählt man ein Projekt unter dem Namen **msp430x2xx.** Diese Beispielprogramme können beim Entwurf einer eigenen Applikation sehr hilfreich sein, um bestimmte Abläufe zu demonstrieren.



8. Für die ersten Schritte wird ein C - Programm verwendet und deshalb wählt man das Projekt unter dem Namen **msp430x2xx_fet_1.c**. Danach drückt man mit der rechten Maustaste auf diesen File und auf dem angezeigten Menü wählt man **Rebuild all.**



9. Bei dem Compilieren des Projekts werden die Bibliotheken in das Projekt einbezogen und in dem Fenster angezeigt

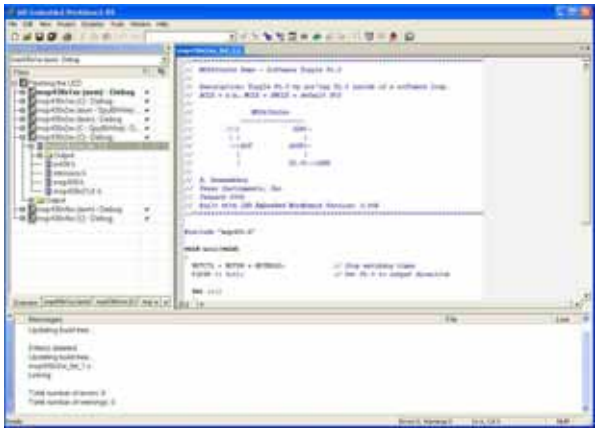


10. Durch einen Doppelklick auf die File **msp430x2xx_fet_1.c**, öffnet sich das Programmierfenster mit dem C-Code. Nach erfolgreicher Durchführung dieser Schritte zeigt sich dem Anwender dieses Bild.

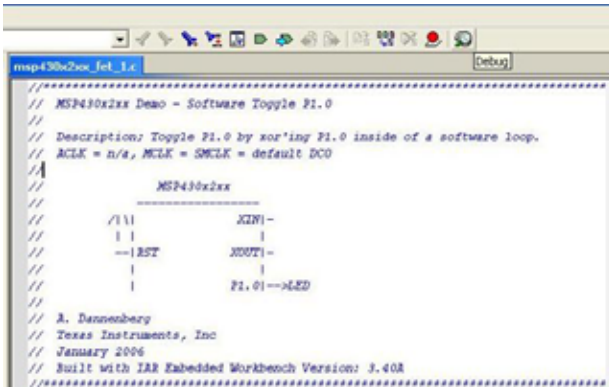
Der Anwender sollte sich auch die Beispielprogramme auf der mitgelieferten CD zum MSP Education System ansehen. Diese Programme sind keine fertigen Applikationen, sondern Beispielprogramme zur Demonstration der Peripheriemodule. Somit kann der Anwender auf einfache Art und Weise eigene Fehler finden oder vermeiden. Nach dem das Beispielprojekt geladen wurde, kann es nun auf das angeschlossene MSP430 Education System geladen werden. Welche Schritte hierfür nötig sind, zeigt die

Tabelle 18. Es werden alle wichtigen Schritte übersichtlich beschrieben und erklärt.

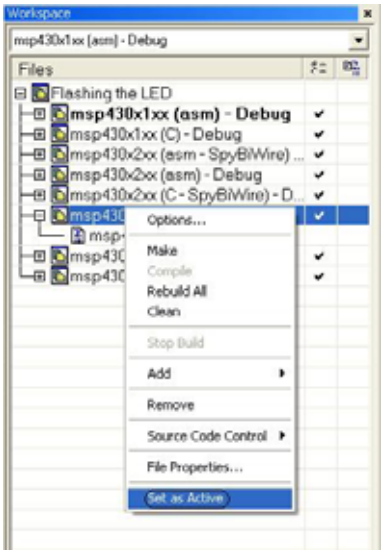
Tabelle 18: Programmieren des Mikrocontrollers



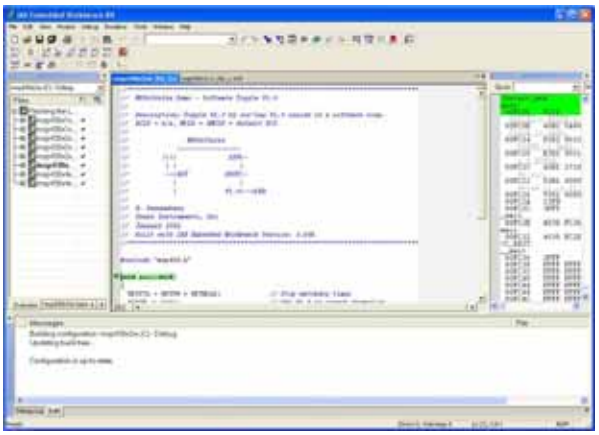
1. Nach dem Laden des Beispielprogramms zeigt sich dem Anwender folgender bekannter Bildschirm. Hat man eine Verbindung zum MSP430 Education System aufgebaut, wie in Kapitel 5.1 beschrieben, kann man nun auf den **Debug**-Button der Toolbar klicken und der Debugger wird geladen.



2. Der Debugger versucht den angeschlossenen Mikrocontroller mit dem erstellten Programm zu programmieren.



3. Bevor man nun debuggen kann muss man darauf achten das das richtige Projekt aktiv ist (Projektname **fett** geschrieben). Dazu führt man einen Rechtsklick auf den Projektnamen aus und wählt den Punkt **Set as Active** heraus. Nun kann man den Debugger über den **Debug-Button** laden.



4. Ist das Programmieren erfolgreich verlaufen, so zeigt sich dem Anwender folgender Bildschirm. Dies sind die Debug-Fenster, in denen das geladene Programm gestartet oder Zeile für Zeile abgearbeitet werden kann. Um das Programm zu starten, drückt man den ***GO**-Button* und der Controller arbeitet das Programm selbständig ab.

Nach Abarbeitung dieser beiden Tabellen kann ein Projekt geladen, kompiliert und auf den Mikrocontroller geladen werden. Nun sind die Voraussetzungen für die Erstellung eines ersten eigenen Projects und die richtige Nutzung der Software geschaffen.

Die IAR Embedded Workbench gibt bei auftretenden Fehler eine Standardfehlermeldung (Bild 5-7) aus. Ist ein Fehler aufgetreten, empfiehlt sich folgende Fehleranalyse:

1. **Überprüfung der Verbindung zwischen Education System und JTAG-Adapter.**
2. **Überprüfung der Verbindung zwischen JTAG-Adapter und Computer.**

3. **Neustart des Mikrocontrollers über den Reset-Taster S1.**
4. **Neustart der Software.**
5. **Überprüfen der Einstellungen für den C-Spy Debugger im Projekt.**
6. **Überprüfung des MSP430 Education System auf Beschädigung.**

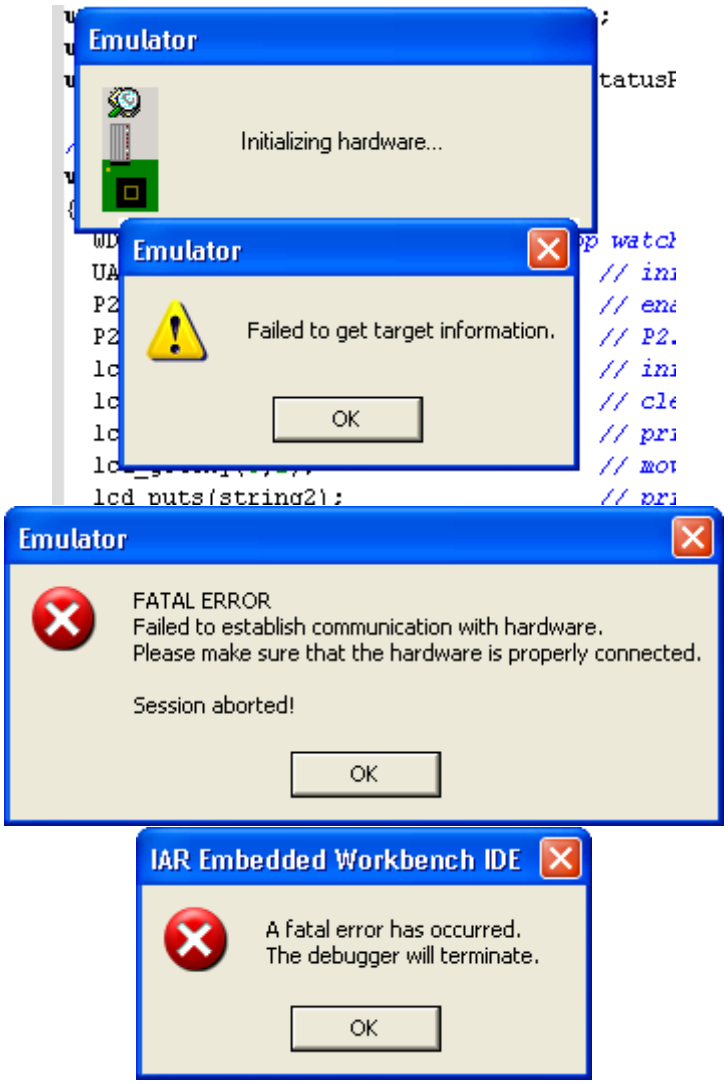


Bild 5-7: Fehlermeldung bei nicht erfolgreichem Download

5.5 Erstes eigenes Programm (basierend auf IAR EW V4.11)

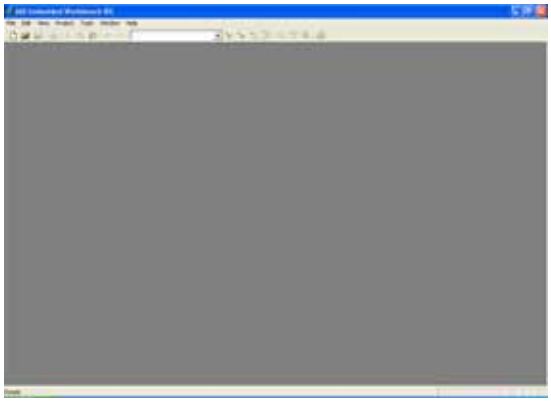
Nach Abarbeitung der Kapitel 5.3 und **Fehler! Verweisquelle konnte nicht gefunden werden.** sind die Grundlagen für das Erstellen eines eigenen Projektes vermittelt. Um nun ein eigenes Projekt zu erstellen, sind wichtige Abläufe und Einstellungen zu beachten. Die genaue Vorgehensweise beim Erstellen und Konfigurieren eines eigenen Projektes zeigt die

Tabelle 19. In ihr werden auch wichtige Projekteinstellungen an Hand von Bildern beschrieben. Hierdurch gelingt es sehr schnell, die Eigenheiten der Programmierumgebung zu erkennen und zu nutzen. Nach Abarbeitung dieses Abschnitts ist der Anwender in der Lage, diese Softwareumgebung gezielt für eigene Projekte und Applikationen einzusetzen.

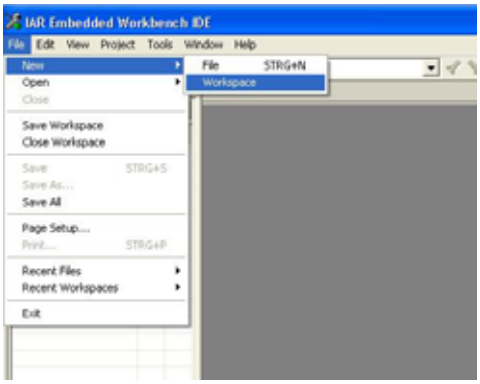
Alle Einstellungen und Einstellmöglichkeiten können hier natürlich nicht beschrieben werden. Dies wird ausführlich in den zugehörigen Handbüchern der einzelnen Programmteile beschrieben. Eine Übersicht aller Handbücher zeigt die Tabelle 16.

Zum Erstellen eines eigenen Projektes werden Grundkenntnisse in einer Programmiersprache vorausgesetzt. In diesem Beispiel wird sich auf die Programmiersprache Assembler beschränkt, da diese Maschinensprache sehr hardwarenah ist und dadurch das grundsätzliche Verständnis für die meisten Abläufe gut vermittelt werden kann. Durch Einsatz der Hochsprache C kann die Programmierung wesentlich beschleunigt werden, da diese wesentlich einfacher und schneller erlernbar ist als Assembler.

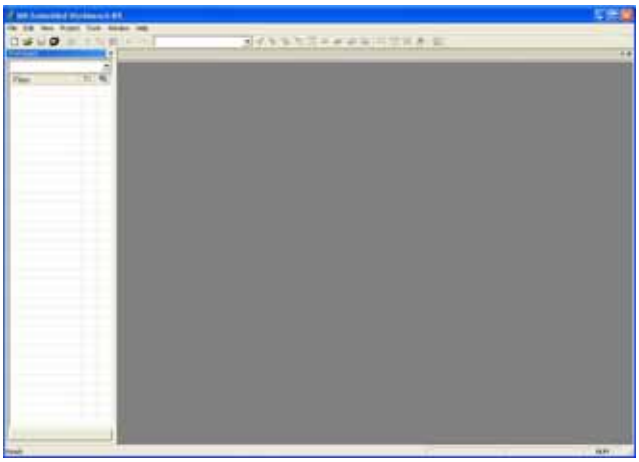
Tabelle 19: Das erste eigene Projekt



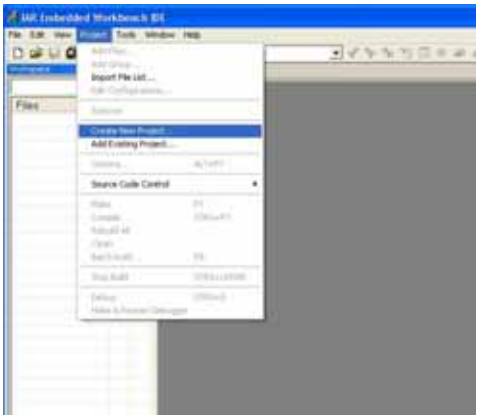
1. Man startet wie gewohnt die IAR Embedded Workbench. Nach dem Laden sieht man dieses Fenster.



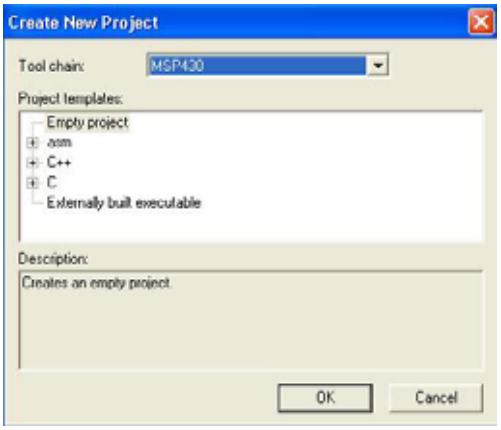
2. Um einen neuen Workspace anzulegen klickt man auf **File** im Menü und im Untermenü auf **New** um einen neuen **Workspace** zu erstellen.



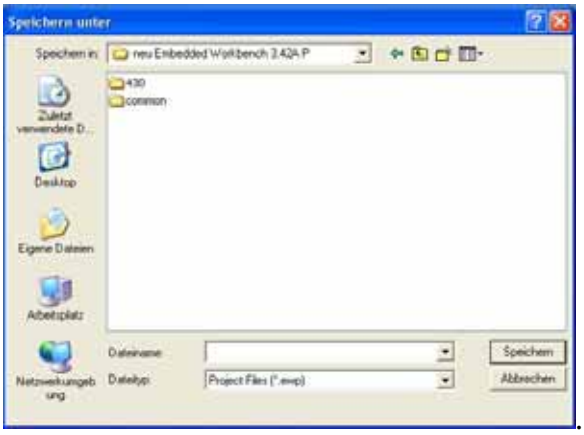
3. Dem Anwender zeigt sich nun dieses Bild. Das Workspace-Fenster ist geöffnet. und zeigt somit das erfolgreiche Anlegen einer neuen Arbeitsfläche.



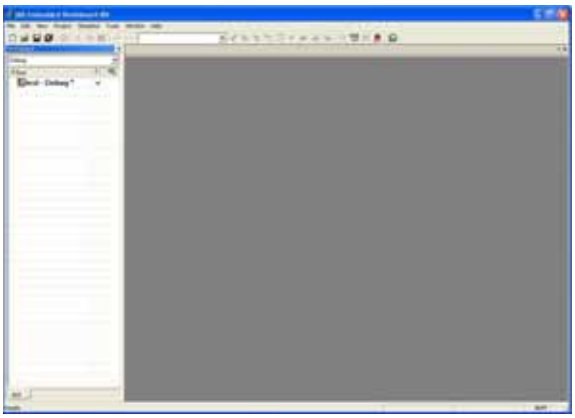
4. Jetzt kann man ein neues Projekt erstellen. Man wählt hier den Eintrag **Create New Project**. da ein neues Projekt angelegt werden soll.



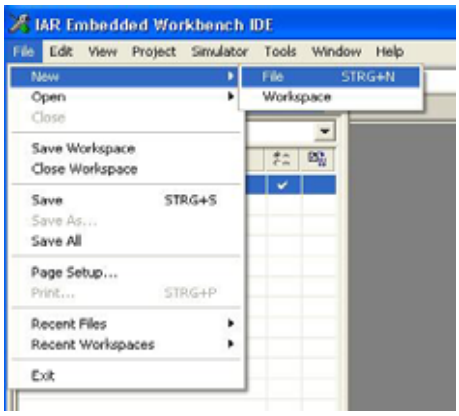
5. Dem Anwender zeigt sich nun dieses Bild. Man kann zwischen verschiedenen Project templates wählen. Für dieses erste eigene Project wählt man nun ein **Empty Project**.



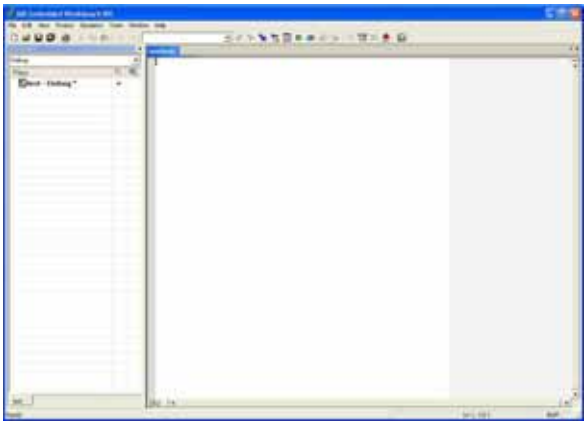
6. Es öffnet sich ein neues Fenster. In diesem Fenster wählt man einen Ordner in dem das neue Projekt gespeichert werden soll sowie den Namen des Projektes. Ist beides festgelegt, speichert man.



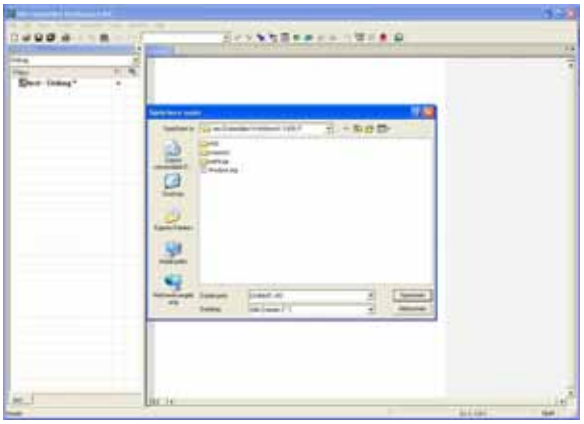
7. Dem Anwender zeigt sich nun dieses Bild. Das Projekt-Fenster ist geöffnet und zeigt somit das erfolgreiche Anlegen eines neuen Projektes. Um nun ein Programm schreiben zu können muss man noch ein neues Sourcefile anlegen.



8. Um ein neues Source-File anzulegen, wählt man erneut unter **File** den Unterpunkt **New** und dann **File**.

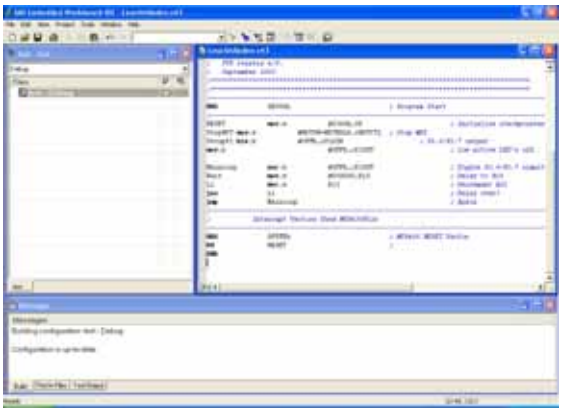


9. Es öffnet sich ein neues Fenster mit dem Namen **Untitle1**. Um das neue Sourcefile zu speichern, drückt man nun auf den **Speichern** Button.

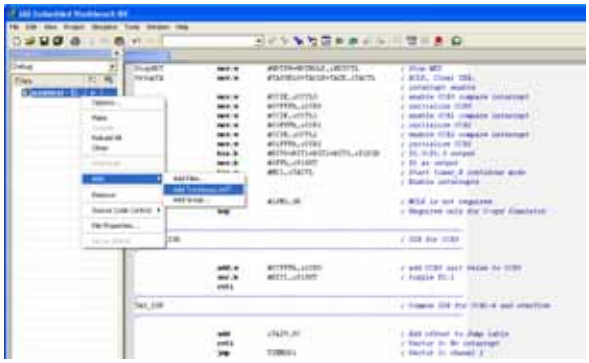


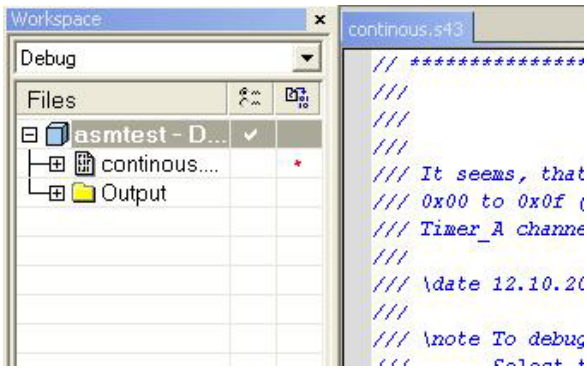
10. Da hier ein Assemblerprogramm entwickelt werden soll, wählt man als Namenserweiterung **„.s43“**. Dies ist die Bezeichnung für eine IAR EW Assemblerfile. Die Datei nennt man nun **xxx.s43** wobei xxx für einen selbst gewählten Namen steht. Danach drückt man auf **Speichern**.

11. Jetzt kann man mit der Programmierung beginnen. In dem Sourcefile kann man jetzt sein eigenes Assemblerprogramm speichern. Für diejenigen, die noch keine so guten Kenntnisse in dieser Programmiersprache haben, ist im Anschluss an diese Tabelle der Sourcecode eines Beispielprogramms abgedruckt.

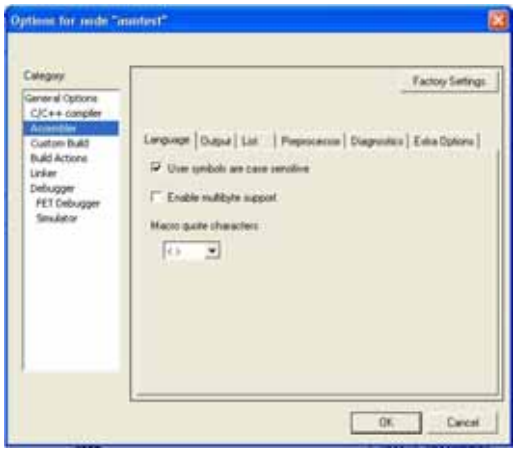


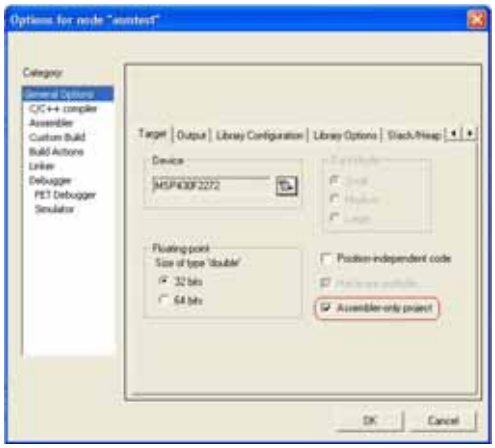
12. Dieser Sourcecode zählt binär von 0x00 zu 0x0f und in Abhängigkeit davon leuchten die LED's an P1.0 bis P1.3. Um das Projekt compilieren und ausführen zu können, müssen sehr viele Grundeinstellungen vorgenommen werden. Alle Einstellungen werden in den folgenden Fenstern genau beschrieben.





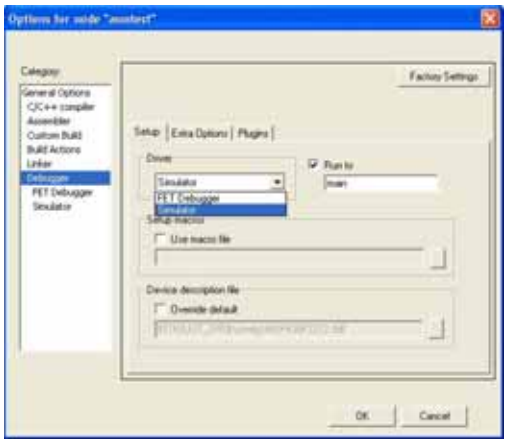
13. Die Quelldatei muss nun in das Projekt eingefügt werden. Dazu macht man einen Rechtsklick auf das Projekt und wählt im Kontextmenü **Add** und wählt zwischen **Add Files ..** und **Add xxx.s43** . Danach ist das Sourcefile in das Projekt eingebunden.

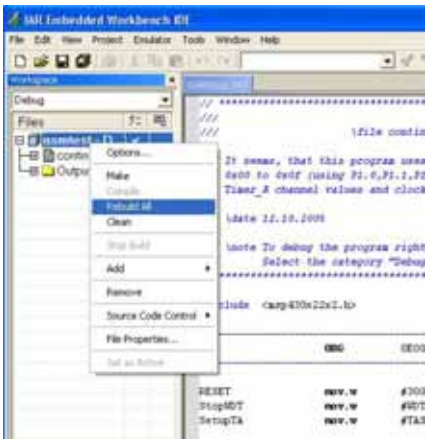




14. Nun müssen noch einige Einstellungen für den Linker und den Debugger vorgenommen werden. Man wählt im Menüpunkt **Project** den Unterpunkt **Options** und dem Anwender zeigt sich dieses Bild. In dem Punkt **General Options** *wählt* man den Target **MSP430F2272**.

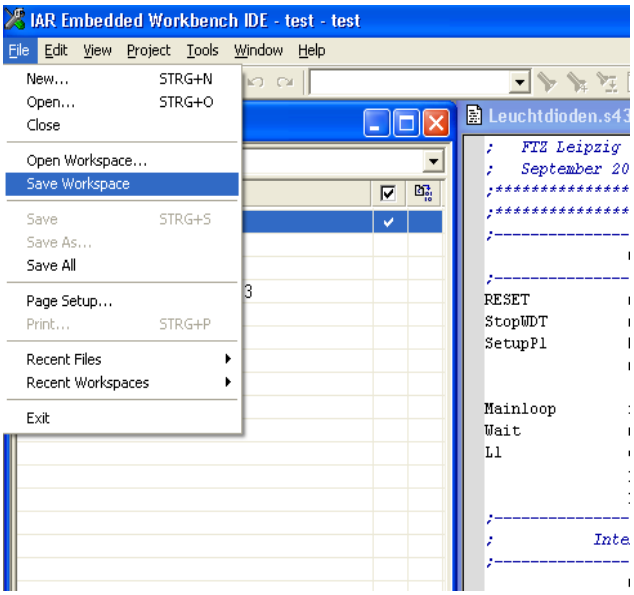
15. Damit das Projekt später ohne Fehler compilieren kann, muss man ein Häkchen in bei **Assembler-only project** machen.



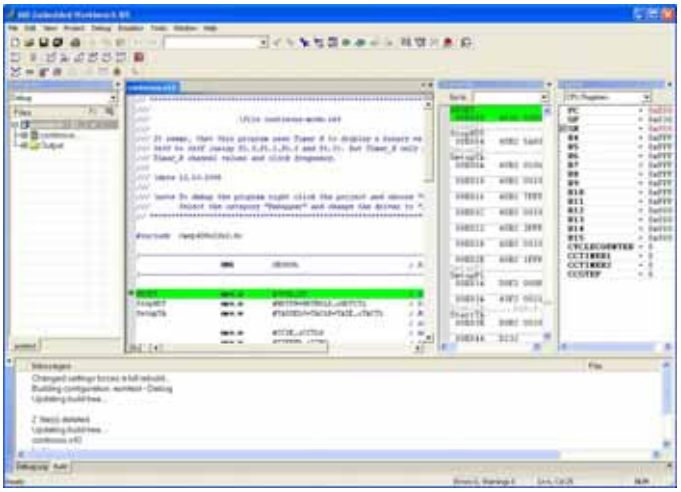


16. Nun wählt man den Unterpunkt **Debugger**. Im Setup Menü des Debugger Fensters kann man unter dem Punkt **Driver** zwischen 2 Debugmodi wählen. Standardmäßig ist hier der **Simulator** eingetragen. Um Programme auf den Mikrocontroller laden zu können, wählt man den Unterpunkt **FET Debugger**.

17. Nun kann man den Sourcecode compilieren. Dies kann man auf dreierlei Wegen tun. Durch Drücken auf den **Compile** Button, den **Make** Button oder auf den **Debug** Buttons. Der dritte Weg stellt gleichzeitig den Download auf den Mikrocontroller dar und öffnet den Debugger.



18. Nach dem erfolgreichen Compilieren sollen im **Messages** Fenster **warnings: 0** und **errors: 0** erscheinen. Um das ganze Projekt abzuspeichern, wählt man in der Menüleiste **File** und dort **Save Workspace**.



19. Nachdem der Sourcecode erfolgreich auf den Mikrocontroller geladen wurde, zeigt sich dem Anwender dieses Fenster. Es zeigt das **Debugger**-Fenster mit dem Sourcecode als zentralem Fenster. Des Weiten wurde hier das **Register** Fenster aktiviert, um die jeweiligen Registerinhalte darzustellen. Im **Build** Fenster wird der erfolgreiche Download der Software auf den Mikrocontroller bestätigt. Diejenigen, welche den nachfolgenden Sourcecode verwendet haben, sehen nach Drücken des *Go* Buttons vier Leuchtdioden blinken. Obwohl alle acht Leuchtdioden blinken sollten, kann man nur vier beobachten. Da hier noch der Debugger aktiv ist, werden die oberen vier Leuchtdioden durch die JTAG-Schnittstelle blockiert. Stoppt man das Debuggen, werden alle acht Leuchtdioden zusammen blinken. Das erste eigene Projekt wurde erfolgreich durchgeführt und beendet.

Nachfolgend der zusammenhängende Quelltext des ersten eigenen Programms:

```
// *************************************************************************
///                \file continous-mode.s43
///
/// It seems, that this program uses Timer A to display a binary value via LEDs. It counts from
/// 0x00 to 0x0f (using P1.0,P1.1,P1.2 and P1.3). But Timer_A only switches LEDs depending on
/// Timer_A channel values and clock frequency.
///
/// \date 12.10.2005
///
/// \note To debug the program right click the project and choose "Options ...".
///                Select the category "Debugger" and change the driver to "FET Debugger"
// *************************************************************************

#include  <msp430x22x2.h>

;-------------------------------------------------------------------------
            ORG             0E000h                  ; Program Start
;-------------------------------------------------------------------------
RESET       mov.w   #300h,SP                        ; Initialize stackpointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL          ; Stop WDT
SetupTA     mov.w   #TASSEL0+TACLR+TAIE,&TACTL      ; ACLK, Clear TAR,
                                                    ; interrupt enable
            mov.w   #CCIE,&CCTL0                    ; enable CCR0 compare interrupt
            mov.w   #07FFFh,&CCR0                   ; initialize CCR0
            mov.w   #CCIE,&CCTL1                    ; enable CCR1 compare interrupt
            mov.w   #03FFFh,&CCR1                   ; initialize CCR1
            mov.w   #CCIE,&CCTL2                    ; enable CCR2 compare interrupt
            mov.w   #01FFFh,&CCR2                   ; initialize CCR2
SetupP1     bis.b   #BIT0+BIT1+BIT2+BIT3,&P1DIR     ; P1.0-P1.3 output
            mov.b   #0FFh,&P1OUT                    ; P1 as output
StartTA     bis.w   #MC1,&TACTL                     ; Start timer_A continous mode
            eint                                    ; Enable interrupts
Mainloop    bis.w   #LPM3,SR                        ; MCLK is not required
            nop                                     ; Required only for C-spy Simulator
;-------------------------------------------------------------------------
TA_CCR0_ISR                                         ; ISR for CCR0
;-------------------------------------------------------------------------
            add.w   #07FFFh,&CCR0                   ; add CCR0 init value to CCR0
            xor.b   #BIT1,&P1OUT                    ; toggle P1.1
            reti
;-------------------------------------------------------------------------
TAX_ISR                                             ; Common ISR for CCR1-4 and overflow
;-------------------------------------------------------------------------
```

```
                add     &TAIV,PC                ; Add offset to Jump table
                reti                            ; Vector 0: No interrupt
                jmp     TIMMOD1                 ; Vector 2: chanel 1
                jmp     TIMMOD2                 ; Vector 4: chanel 2
                reti                            ; Vector 6: chanel 3 not implemented
                reti                            ; Vector 8: chanel 4 not implemented
TIMOVH          xor.b   #BIT0,&P1OUT            ; Vector 10: TIMOV Flag
                reti                            ; toggle P1.0
TIMMOD1         add.w   #03FFFh,&CCR1           ; Vector 2: Module 1
                xor.b   #BIT2,&P1OUT            ; add CCR1 init value to CCR1
                reti                            ; toggle P1.2
TIMMOD2         add.w   #01FFFh,&CCR2           ; Vector 4: Module 2
                xor.b   #BIT3,&P1OUT            ; add CCR2 init value to CCR2
                reti                            ; toggle P1.3
;-------------------------------------------------------------------------------
;    Interrupt Vectors Used MSP430F12x
;-------------------------------------------------------------------------------
                ORG     0FFFEh                  ; MSP430 RESET Vector
                DW      RESET
                ORG     0FFF0h                  ; Timer_AX Vector
                DW      TAX_ISR
                ORG     0FFF2h                  ; Timer_A0 Vector
                DW      TA_CCR0_ISR
                END
```

6 MSP430 JTAG-Adapter

Bild 6-1: MSP430 USB-JTAG-Adapter

Zur Programmierung des MSP430 kann ein Adapter für den Parallelport des PCs oder ein USB-Adapter verwendet werden. Letzterer ist eine Erweiterung des eZ430 USB-Sticks von TI.

Bild 6-2: MSP430 Parallel-JTAG-Adapter

Tabelle 20 gibt noch einmal die Belegung des JTAG-Steckverbinders wieder.

Tabelle 20: Pinbelegung der JTAG-Schnittstelle

JTAG-Pin	Signalname	Beschreibung und angeschlossene Peripherie
1	P1.7_TDO	Test Data Output Leitung der JTAG-Schnittstelle
2	VCC_IN	Versorgung des Targets aus der parallelen Schnittstelle (nicht verwendet in Hardware v2.0)
3	P1.6_TDI	Test Data Input Leitung der JTAG-Schnittstelle
4	VCC_OUT	Versorgung des JTAG-Steckverbinders vom Target
5	P1.5_TMS	Test Mode Select Leitung der JTAG-Schnittstelle
6	TCLK	Taktsignal aus der Parallelschnittstelle
7	P1.4_TCK	Test Clock Leitung der JTAG-Schnittstelle
8	TEST	Anschluss des JTAG-Moduls, schaltet den Controller zum Programmieren frei (SBWTDIO)
9	GND	Masse
10	NC	NC
11	RST	Reset-Leitung zum Controller (SBWTCK)
12	NC	NC
13	NC	NC
14	NC	NC

Die wichtigsten Signale und deren Funktionsweise wurden bereits in Kapitel 4.3 genau beschrieben. Die Beschreibung dieser Signale kann an dieser Stelle nachgelesen werden. Einzig die **TCLK-Leitung** wurde nicht weiter beschrieben, da sie in dieser Lösung nicht von Nutzen ist. Über die TCLK-Leitung ist es möglich einen Takt über die Parallelschnittstelle auf den Pin 6 des JTAG-Adapters zu geben und diesen auf eine angeschlossene Platine zu bringen. Damit kann zum Beispiel eine externe Taktversorgung über die JTAG-Schnittstelle realisiert werden.

A. Befehle des MSP430F2272

Die folgende Tabelle 21 zeigt alle Befehle des MSP430 in einer Übersicht. Alle Befehle stellen 16-Bit Worte dar. Die Befehle, die am Ende ein „(.B)“ besitzen, können optional mit dem Kürzel „.B“ auch als Byte-Befehle behandelt werden. Bestimmte Peripherie-Module können nur mit Byte-Befehlen behandelt werden, wie zum Beispiel die Ports. Alle Register, die mit einem Port in Verbindung stehen sind 8-Bit breit und können nur mit Byte-Befehlen beschrieben und ausgelesen werden. So auch das USART-Modul. Timer_A ist zum Beispiel ein Word-Befehl. Alle Befehle denen ein „*“ voransteht sind emulierte Befehle. Die genaue Beschreibung aller Befehle ist im User Guide [7] nachzulesen. Diese wird an kleinen Beispielen durchgeführt, um die genau Funktionsweise schnell erkennen zu können. In diesem findet man auch die genaue Registerbeschreibung sowie deren ausführliche Beschreibung.

Tabelle 21: Alle Befehle des MSP430 in Übersicht

	Mnemonic		Description		Status Bits			
					V	N	Z	C
*	ADC(.B)	dst	Add carry bit (C) to destination	dst + C –> dst	*	*	*	*
	ADD(.B)	src,dst	Add source to destination	src + dst –> dst	*	*	*	*
	ADDC(.B)	src,dst	Add source and carry to destination	src + dst + C –> dst	*	*	*	*
	AND(.B)	src,dst	AND source and destination	src .and. dst –> dst 0	0	*	*	*
	BIC(.B)	src,dst	Clear bits in destination	.not.src .and. dst –> dst	-	-	-	-
	BIS(.B)	src,dst	Set bits in destination	src .or. dst –> dst	-	-	-	-
	BIT(.B)	src,dst	Test bit in destination	src .and. dst	0	*	*	*
*	BR	dst	Branch to destination	dst-> PC	-	-	-	-
	CALL	dst	Call destination	PC+2 –> stack, dst –> PC	-	-	-	-
*	CLR(.B)	dst	Clear destination	0 -> dst	-	-	-	-
*	CLRC		Clear carry bit	0 -> C	-	-	-	0
*	CLRN		Clear negative bit	0 -> N	-	0	-	-
*	CLRZ		Clear zero bit	0 -> Z	-	-	0	-
	CMP(.B)	scr,dst	Compare Source and destination	dst – src	*	*	*	*
*	DADC(.B)	dst	Add C decimally to destination	dst + C –> dst (decimally)	*	*	*	*
	DADD(.B)	scr,dst	Add source and C decimally to dst	src + dst + C –> dst (decimally)	*	*	*	*
*	DEC(.B)	dst	Decrement destination	dst – 1 –> dst	*	*	*	*
*	DECD(.B)	dst	Increment destination	dst – 2 –> dst	*	*	*	*
*	DINT		Disable interrupt	0 -> GIE	-	-	-	-
*	EINT		Enable interrupt	1 -> GIE	-	-	-	-
*	INC(.B)	dst	Increment destination	dst + 1 -> dst	*	*	*	*
*	INCD(.B)	dst	Double-Increment destination	dst + 2 -> dst	*	*	*	*
*	INV(.B)	dst	Invert destination	.not.dst -> dst	*	*	*	*
	JC/JHS	label	Jump if C set/Jump if higher or same		-	-	-	-
	JEQ/JZ	label	Jump if equal/Jump if Zero-bit is set		-	-	-	-
	JGE	label	Jump to label if graeter or equal	(N .XOR. V) = 0	-	-	-	-
	JL	label	Jump to label if less	(N .XOR. V) = 1	-	-	-	-
	JMP	label	Jump to label unconditionally	PC +2 x offset -> PC	-	-	-	-
	JN	label	Jump to label if N set		-	-	-	-
	JNC/JLO	label	Jump if C not set/ Jump if lower		-	-	-	-

	Mnemonic		Description		Status Bits			
					V	N	Z	C
	JNE/JNZ	label	Jump if not equal/Jump if Z not set		-	-	-	-
	MOV(.B)	src,dst	Move source to destination	src –> dst	-	-	-	-
*	NOP		No operation		-	-	-	-
*	POP(.B)	dst	Pop item from stack to destination	@Sp -> dst, SP+2 -> SP	-	-	-	-
	PUSH(.B)	scr	Push source onto stack	SP – 2 -> SP, src -> @SP	-	-	-	-
*	RET		Return from subroutine	@SP → PC, SP + 2 → SP	-	-	-	-
	RETI		Return from interrupt		*	*	*	*
*	RLA(.B)	dst	Rotate left arithmetically		*	*	*	*
*	RLC(.B)	dst	Rotate left through carry		*	*	*	*
	RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
	RRC(.B)	dst	Rotate right through carry		*	*	*	*
*	SBC(.B)	dst	Subtract not (carry) from destination	dst + 0FFFFh + C -> dst	*	*	*	*
*	SETC		Set carry bit	1 -> C	-	-	-	1
*	SETN		Set negative bit	1 -> N	-	1	-	-
*	SETZ		Set zero bit	1 -> C	-	-	1	-
	SUB(.B)	src,dst	Subtract source from destination	dst + .not.src + 1 -> dst	*	*	*	*
	SUBC(.B)	src,dst	Subtract source an not © from dst.	dst + .not.src + C -> dst	*	*	*	*
	SWPB	dst	swap bytes		-	-	-	-
	SXT	dst	Extend sign		0	*	*	*
*	TST(.B)	dst	Test destination		0	*	*	1
	XOR(.B)	src,dst	Exclusive OR source and destination	src .xor. dst -> ds	*	*	*	*

***** **-** Emulierte Befehle

src - Abkürzung für Source, Quelle

dst - Abkürzung für Destination, Ziel

label - Bezeichnung für die eingesetzte Sprungmarke,

B. Schaltpläne

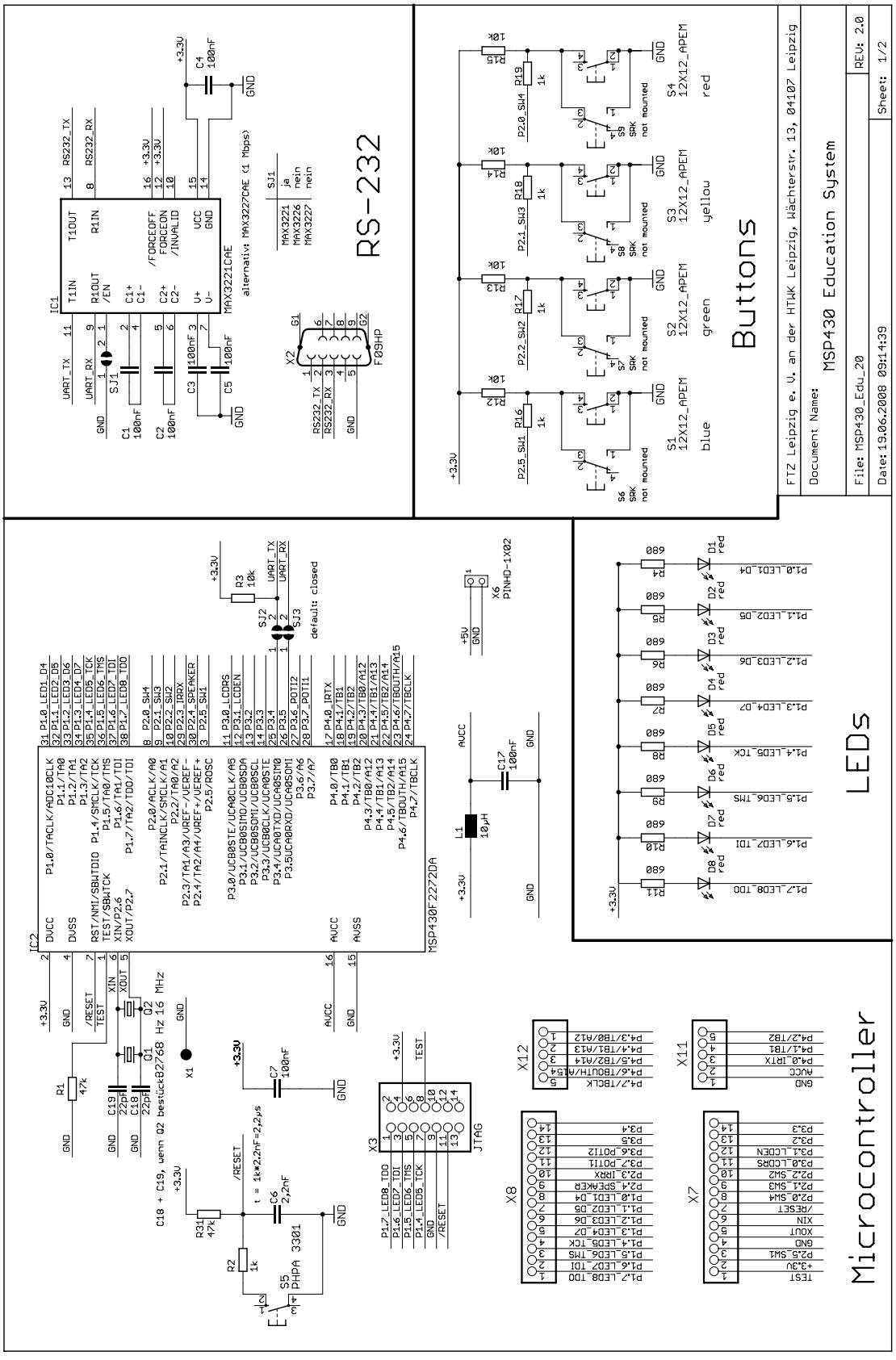


Bild B-1: Schaltplan MSP430 Education System 2.0 Seite 1/2

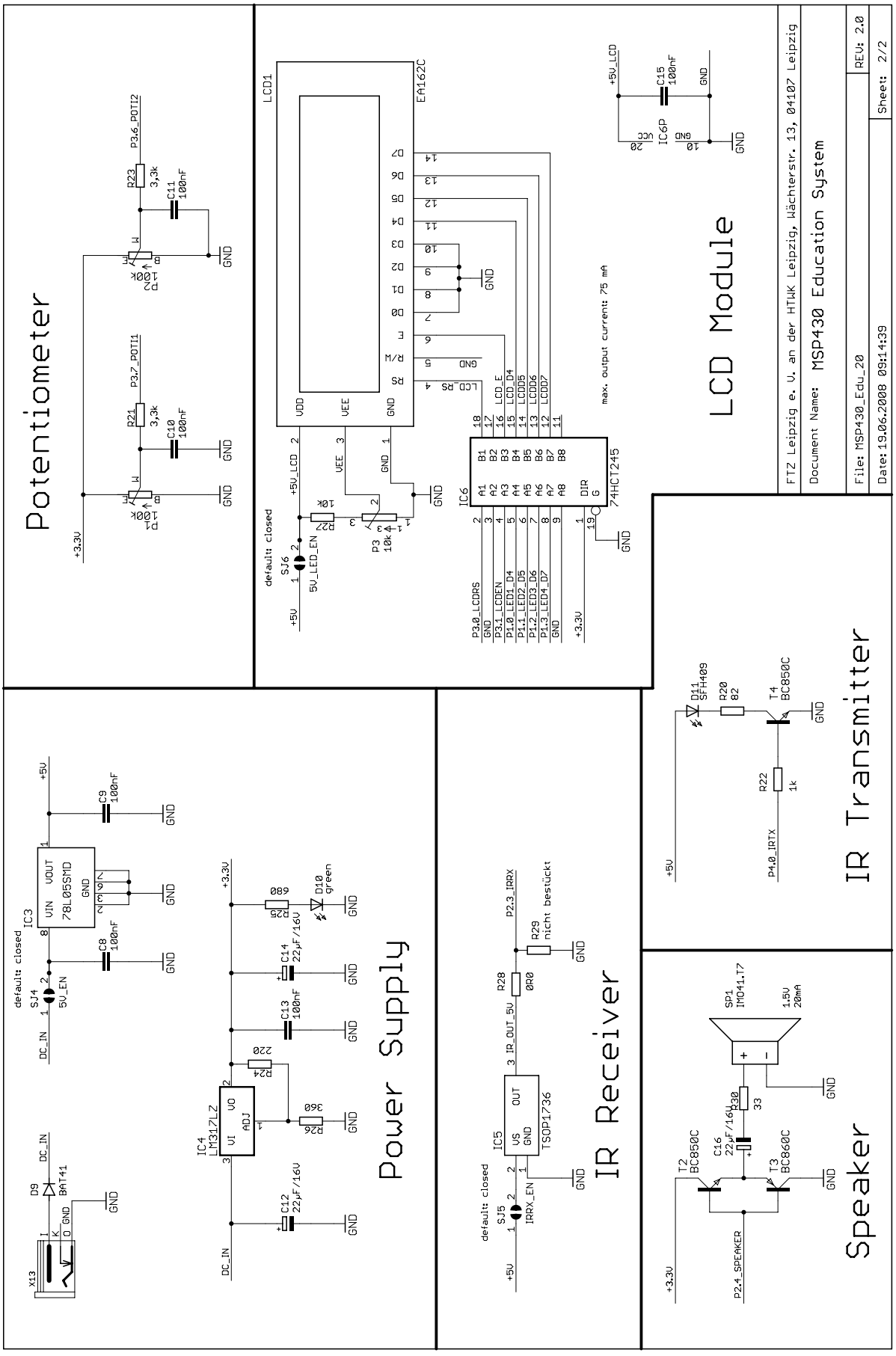


Bild B-2: Schaltplan MSP430 Education System 2.0 Seite 2/2

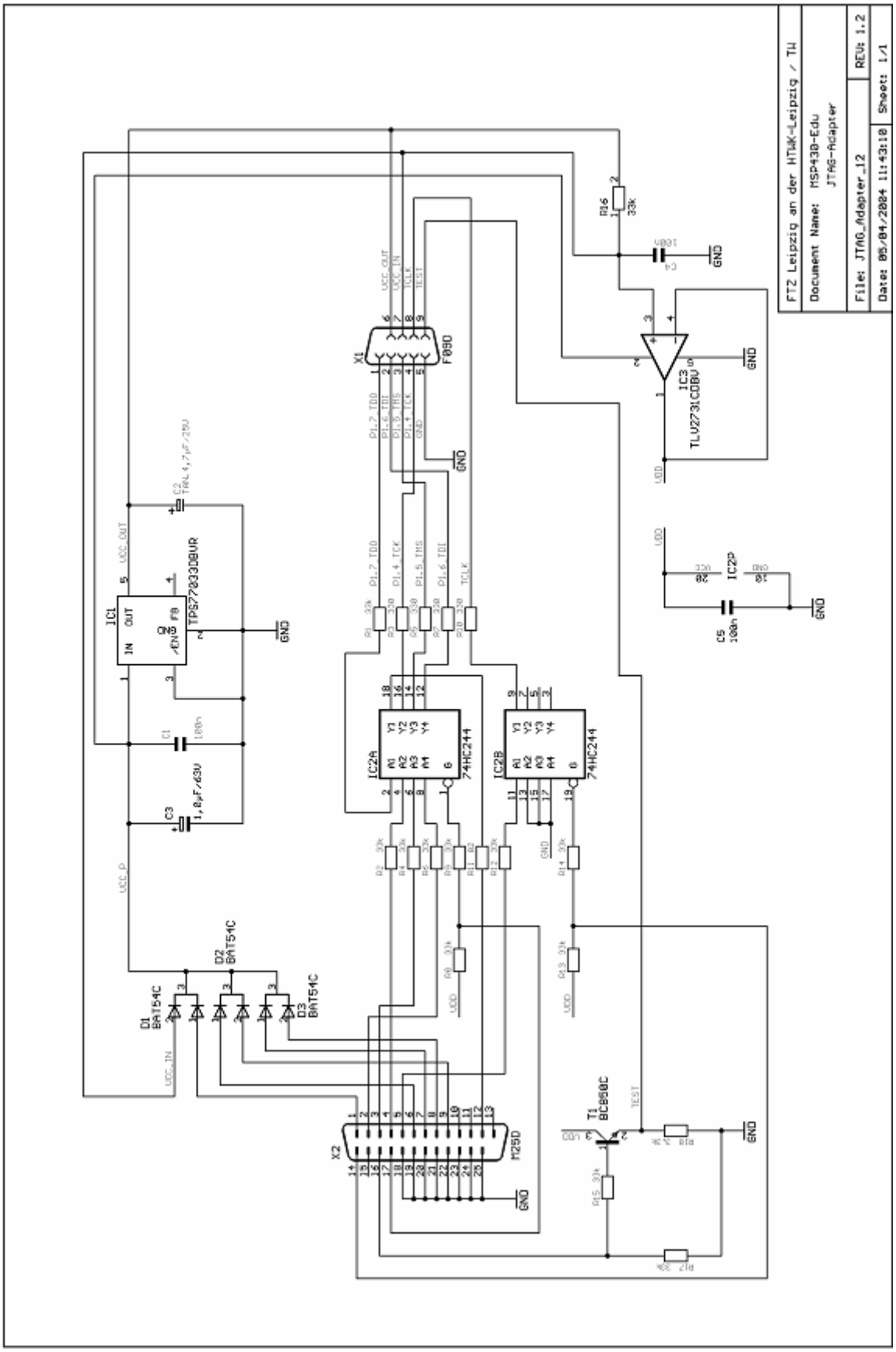


Bild B-3: Schaltplan eines MSP430 Parallel-JTAG-Adapters

C. Bestückungsplan

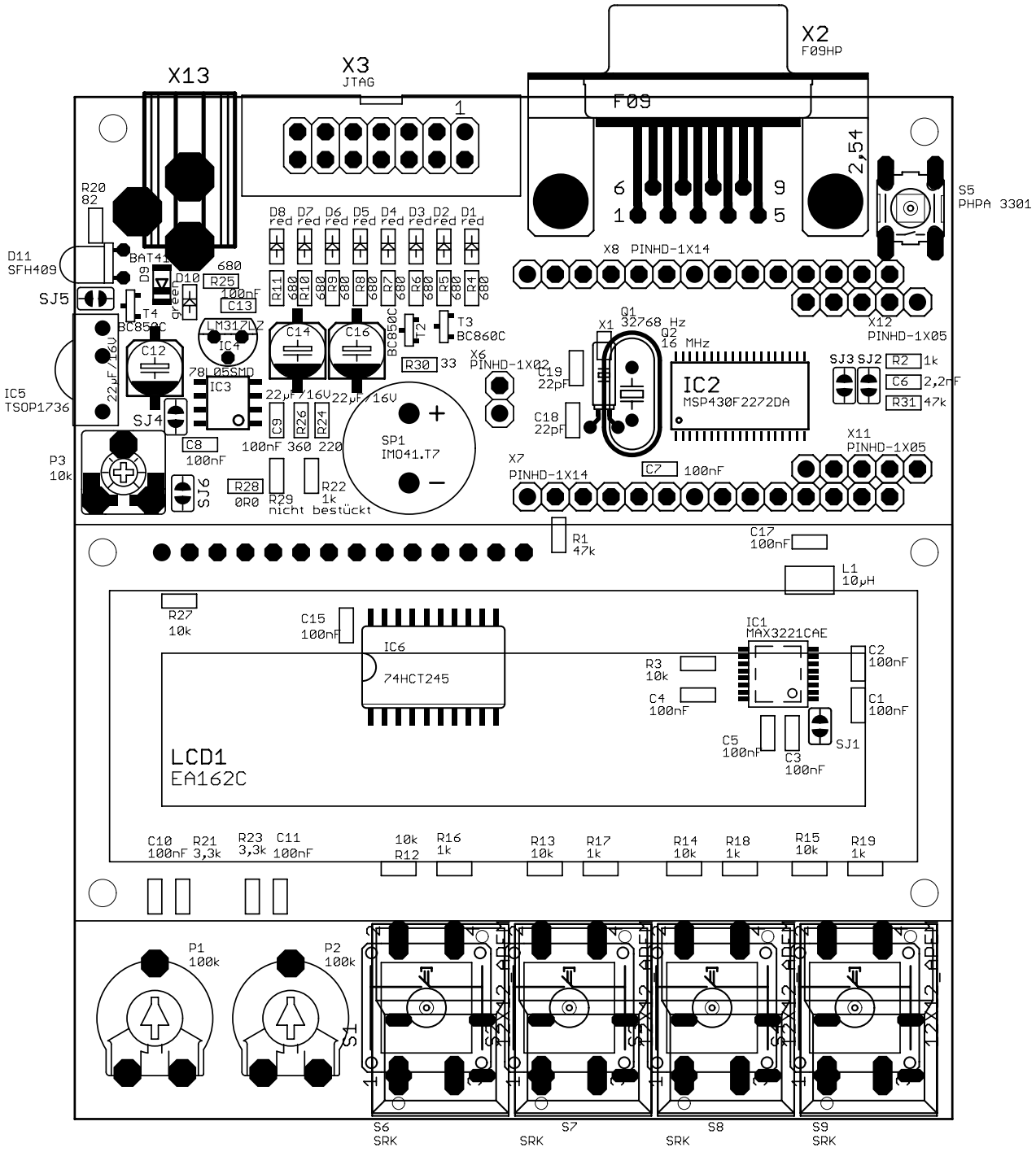


Bild C-1: Bestückungsplan MSP430 Education System 2.0 top

D. Mechanische Abmessungen

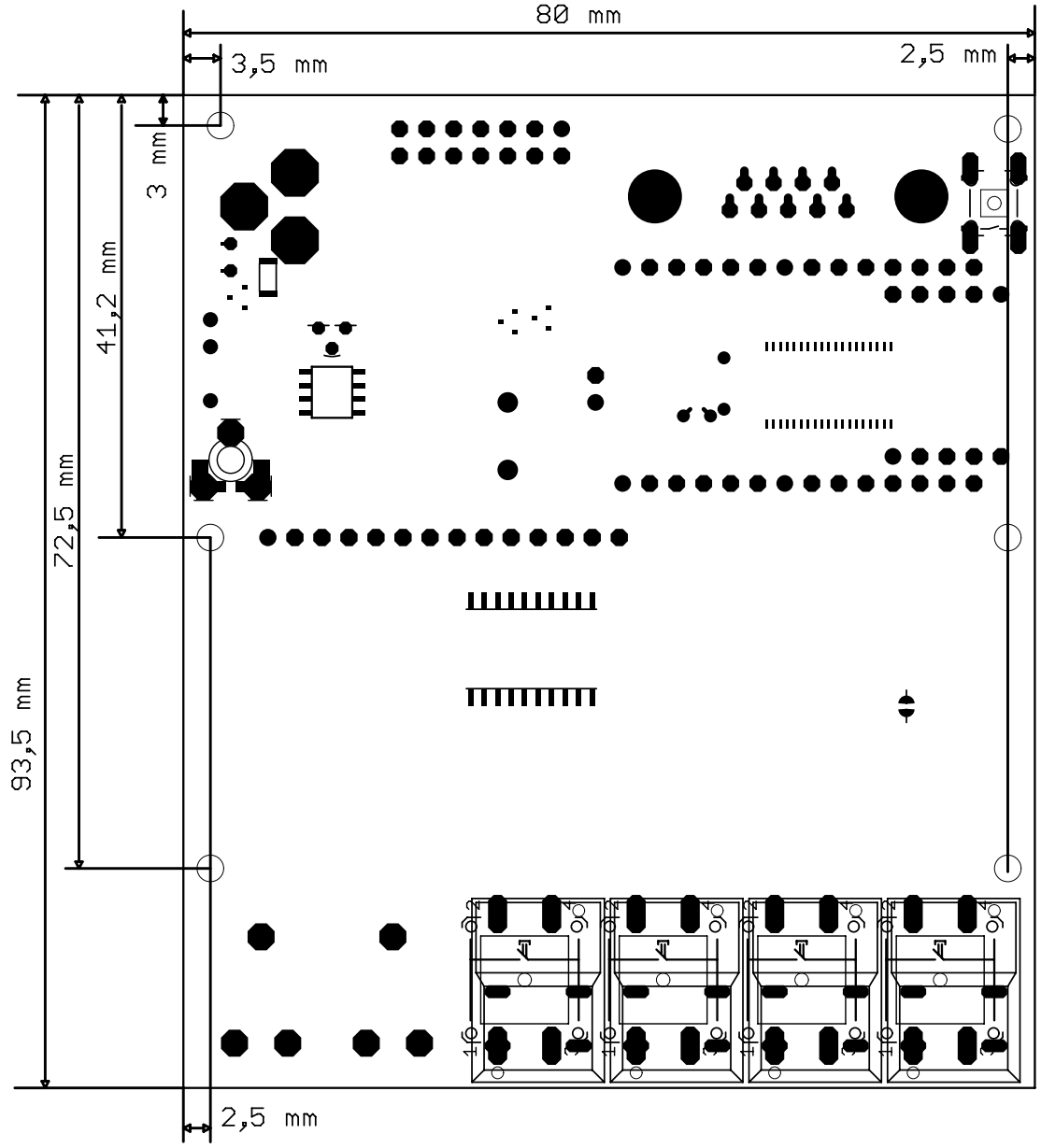


Bild D-1: Mechanische Abmessungen und Lage der Bohrungen

E. Übersicht Beispielprogramme (TI)

Nachfolgend eine Übersicht der von Texas Instruments zur Verfügung gestellten Test-Programme, die frei von der Texas Instruments Homepage [1] bezogen werden können. Die Beispielprogramme wurden in zwei Kategorien unterteilt und zwar in Assembler- und C-Programmbeispiele. Durch Pflege und Aktualisierung der Programme können Verschiebungen vorkommen, der größte Teil wird aber über diese Liste abgedeckt. Die entsprechend verwendete Funktionseinheit wird im Namen mitgeführt und kann mit der dazugehörigen Datei getestet werden.

6.1 MSP-FET430P120 Assembler Examples slac143a.zip (81k)

- msp430x22x4_1.asm - Software Toggle P1.0
- msp430x22x4_1_vlo.asm - Software Toggle P1.0, MCLK = VLO/8
- msp430x22x4_lpm3.asm - Basic Clock, LPM3 Using WDT ISR, 32kHz ACLK
- msp430x22x4_lpm3_vlo.asm - Basic Clock, LPM3 Using WDT ISR, VLO ACLK
- msp430x22x4_p1_02.asm - Software Port Interrupt Service on P1.3 from LPM4
- msp430x22x4_oa_01.asm - OA0, Comparator Mode
- msp430x22x4_oa_02.asm - OA0, General-Purpose Mode
- msp430x22x4_oa_03.asm - OA0, Inverting PGA Mode
- msp430x22x4_oa_04.asm - OA0, Non-Inverting PGA Mode
- msp430x22x4_oa_05.asm - OA0, Unity-Gain Buffer Mode
- msp430x22x4_adc10_01.asm - ADC10, Sample A0, AVcc Ref, Set P1.0 if A0 > 0.5*AVcc
- msp430x22x4_adc10_02.asm - ADC10, Sample A0, 1.5V Ref, Set P1.0 if A0 > 0.2V
- msp430x22x4_adc10_03.asm - ADC10, Sample A10 Temp, Set P1.0 if Temp ++ ~2C
- msp430x22x4_adc10_04.asm - ADC10, Sample A0, Signed, Set P1.0 if A0 > 0.5*AVcc
- msp430x22x4_adc10_05.asm - ADC10, Sample A11, Lo_Batt, Set P1.0 if AVcc < 2.3V
- msp430x22x4_adc10_06.asm - ADC10, Output Internal Vref on P2.4 & ADCCLK on P1.0
- msp430x22x4_adc10_07.asm - ADC10, DTC Sample A0 64x, AVcc, Repeat Single, DCO
- msp430x22x4_adc10_08.asm - ADC10, DTC Sample A0 64x, 1.5V, Repeat Single, DCO
- msp430x22x4_adc10_10.asm - ADC10, DTC Sample A2-0, AVcc, Single Sequence, DCO
- msp430x22x4_adc10_13.asm - ADC10, DTC Sample A1 32x, AVcc, TA0 Trig, DCO
- msp430x22x4_adc10_14.asm - ADC10, DTC Sample A1-0 32x, AVcc, Repeat Seq, DCO
- msp430x22x4_adc10_15.asm - ADC10, DTC Sample A10 32x to Flash, Int Ref, DCO
- msp430x22x4_adc10_19.asm - ADC10, DTC Sample A0 64x, AVcc, HF XTAL
- msp430x22x4_adc10_20.asm - ADC10, DTC Sample A0 2-Blk Cont. Mode, AVcc, HF XTAL
- msp430x22x4_flashwrite_01.asm - Flash In-System Programming, Copy SegC to SegD
- msp430x22x4_flashwrite_04.asm - Flash In-System Programming w/ EEI, Copy SegD to A/B/C
- msp430x22x4_clks.asm - Basic Clock, Output Buffered SMCLK, ACLK and MCLK/10
- msp430x22x4_fll_01.asm - Basic Clock, Implement Auto RSEL SW FLL
- msp430x22x4_fll_02.asm - Basic Clock, Implement Cont. SW FLL with Auto RSEL
- msp430x22x4_hfxtal.asm - Basic Clock, MCLK Sourced from HF XTAL
- msp430x22x4_rosc.asm - DCOCLK Biased with External Resistor Rosc
- msp430x22x4_dco_flashcal.asm - DCO Calibration Constants Programmer
- msp430x22x4_ta_01.asm - Timer_A, Toggle P1.0, TACCR0 Cont. Mode ISR, DCO SMCLK
- msp430x22x4_ta_02.asm - Timer_A, Toggle P1.0, TACCR0 Up Mode ISR, DCO SMCLK
- msp430x22x4_ta_03.asm - Timer_A, Toggle P1.0, Overflow ISR, DCO SMCLK
- msp430x22x4_ta_04.asm - Timer_A, Toggle P1.0, Overflow ISR, 32kHz ACLK
- msp430x22x4_ta_05.asm - Timer_A, Toggle P1.0, TACCR0 Up Mode ISR, 32kHz ACLK
- msp430x22x4_ta_06.asm - Timer_A, Toggle P1.0, TACCR1 Cont. Mode ISR, DCO SMCLK
- msp430x22x4_ta_16.asm - Timer_A, PWM TA1-2, Up Mode, DCO SMCLK

- msp430x22x4_ta_17.asm - Timer_A, PWM TA1-2, Up Mode, 32kHz ACLK
- msp430x22x4_tb_01.asm - Timer_B, Toggle P1.0, TBCCR0 Cont. Mode ISR, DCO SMCLK
- msp430x22x4_tb_04.asm - Timer_B, Toggle P1.0, Overflow ISR, 32kHz ACLK
- msp430x22x4_tb_05.asm - Timer_B, Toggle P1.0, TBCCR0 Up Mode ISR, 32kHz ACLK
- msp430x22x4_tb_06.asm - Timer_B, Toggle P1.0, TBCCR1 Cont. Mode ISR, DCO SMCLK
- msp430x22x4_tb_09.asm - Timer_B, Toggle P4.0-2, Cont. Mode ISR, HF XTAL ACLK
- msp430x22x4_tb_11.asm - Timer_B, PWM TB1-2, Up Mode, 32kHz ACLK
- msp430x22x4_wdt_01.asm - WDT, Toggle P1.0, Interval Overflow ISR, DCO SMCLK
- msp430x22x4_wdt_02.asm - WDT, Toggle P1.0, Interval Overflow ISR, 32kHz ACLK
- msp430x22x4_wdt_04.asm - WDT+ Failsafe Clock, DCO SMCLK
- msp430x22x4_wdt_05.asm - Reset on Invalid Address fetch, Toggle P1.0
- msp430x22x4_wdt_06.asm - WDT+ Failsafe Clock, 32kHz ACLK
- msp430x22x4_uscia0_uart_01_115k.asm - USCI_A0, 115200 UART Echo ISR, DCO SMCLK
- msp430x22x4_uscia0_uart_01_115k_lpm.asm - USCI_A0, 115200 UART Echo ISR, DCO SMCLK, LPM4
- msp430x22x4_uscia0_uart_01_9600.asm - USCI_A0, 9600 UART Echo ISR, DCO SMCLK
- msp430x22x4_uscia0_uart_05_9600.asm - USCI_A0, Ultra-Low Pwr UART 9600 Echo ISR, 32kHz ACLK
- msp430x22x4_uscia0_uart_08_9600.asm - USCI_A0, UART 9600 Full-Duplex Transceiver, 32kHz ACLK
- msp430x22x4_uscia0_spi_09.asm - USCI_A0, SPI 3-Wire Master Incremented Data
- msp430x22x4_uscia0_spi_10.asm - USCI_A0, SPI 3-Wire Slave Data Echo
- msp430x22x4_uscia0_irda_01.asm - USCI_A0 IrDA External Loopback Test, 8MHz SMCLK
- msp430x22x4_uscia0_irda_02.asm - USCI_A0 IrDA Monitor, 8MHz SMCLK
- msp430x22x4_uscib0_i2c_08.asm - USCI_B0 I2C Master TX multiple bytes to MSP430 Slave
- msp430x22x4_uscib0_i2c_09.asm - USCI_B0 I2C Slave RX multiple bytes from MSP430 Master
- msp430x22x4_uscib0_i2c_10.asm - USCI_B0 I2C Master RX multiple bytes from MSP430 Slave
- msp430x22x4_uscib0_i2c_11.asm - USCI_B0 I2C Slave TX multiple bytes to MSP430 Master

6.2 MSP-FET430P120 "C" Examples slac123b.zip (136k)

- msp430x22x4_1.c - Software Toggle P1.0
- msp430x22x4_1_vlo.c - Software Toggle P1.0, MCLK = VLO/8
- msp430x22x4_lpm3.c - Basic Clock, LPM3 Using WDT ISR, 32kHz ACLK
- msp430x22x4_lpm3_vlo.c - Basic Clock, LPM3 Using WDT ISR, VLO ACLK
- sp430x22x4_nmi.c - Configure RST/NMI as NMI
- msp430x22x4_p1_01.c - Software Poll P1.3, Set P1.0 if P1.3 = 1
- msp430x22x4_p1_02.c - Software Port Interrupt Service on P1.3 from LPM4
- msp430x22x4_p1_03.c - Poll P1 With Software with Internal Pull-up
- msp430x22x4_p1_04.c - P1 Interrupt from LPM4 with Internal Pull-up
- msp430x22x4_oa_01.c - OA0, Comparator Mode
- msp430x22x4_oa_02.c - OA0, General-Purpose Mode
- msp430x22x4_oa_03.c - OA0, Inverting PGA Mode
- msp430x22x4_oa_04.c - OA0, Non-Inverting PGA Mode
- msp430x22x4_oa_05.c - OA0, Unity-Gain Buffer Mode
- msp430x22x4_oa_06.c - OA1, Comparator Mode
- msp430x22x4_oa_07.c - OA1, General-Purpose Mode
- msp430x22x4_oa_08.c - OA1, Inverting PGA Mode
- msp430x22x4_oa_09.c - OA1, Non-Inverting PGA Mode
- msp430x22x4_oa_10.c - OA1, Unity-Gain Buffer Mode
- msp430x22x4_adc10_01.c - ADC10, Sample A0, AVcc Ref, Set P1.0 if A0 > 0.5*AVcc
- msp430x22x4_adc10_02.c - ADC10, Sample A0, 1.5V Ref, Set P1.0 if A0 > 0.2V
- msp430x22x4_adc10_03.c - ADC10, Sample A10 Temp, Set P1.0 if Temp ++ ~2C
- msp430x22x4_adc10_04.c - ADC10, Sample A0, Signed, Set P1.0 if A0 > 0.5*AVcc
- msp430x22x4_adc10_05.c - ADC10, Sample A11, Lo_Batt, Set P1.0 if AVcc < 2.3V
- msp430x22x4_adc10_06.c - ADC10, Output Internal Vref on P2.4 & ADCCLK on P1.0
- msp430x22x4_adc10_07.c - ADC10, DTC Sample A0 64x, AVcc, Repeat Single, DCO
- msp430x22x4_adc10_08.c - ADC10, DTC Sample A0 64x, 1.5V, Repeat Single, DCO
- msp430x22x4_adc10_09.c - ADC10, DTC Sample A10 64x, 1.5V, Repeat Single, DCO

- msp430x22x4_adc10_10.c - ADC10, DTC Sample A2-0, AVcc, Single Sequence, DCO
- msp430x22x4_adc10_11.c - ADC10, Sample A0, 1.5V, TA1 Trig, Set P1.0 if > 0.5V
- msp430x22x4_adc10_12.c - ADC10, Sample A7, 1.5V, TA1 Trig, Ultra-Low Pwr
- msp430x22x4_adc10_13.c - ADC10, DTC Sample A1 32x, AVcc, TA0 Trig, DCO
- msp430x22x4_adc10_14.c - ADC10, DTC Sample A1-0 32x, AVcc, Repeat Seq, DCO
- msp430x22x4_adc10_15.c - ADC10, DTC Sample A10 32x to Flash, Int Ref, DCO
- msp430x22x4_adc10_16.c - ADC10, DTC Sample A0 -> TA1, AVcc, DCO
- msp430x22x4_adc10_17.c - ADC10, DTC Sample A0 -> TA1, AVcc, HF XTAL
- msp430x22x4_adc10_18.c - ADC10, DTC Sample A1/0 -> TA1/2, 2.5V, HF XTAL
- msp430x22x4_adc10_19.c - ADC10, DTC Sample A0 64x, AVcc, HF XTAL
- msp430x22x4_adc10_20.c - ADC10, DTC Sample A0 2-Blk Cont. Mode, AVcc, HF XTAL
- msp430x22x4_adc10_temp.c - ADC10, Sample A10 Temp and Convert to oC and oF
- msp430x22x4_flashwrite_01.c - Flash In-System Programming, Copy SegC to SegD
- msp430x22x4_flashwrite_03.c - Flash In-System Programming w/ EEI, Copy SegC to SegD
- msp430x22x4_flashwrite_04.c - Flash In-System Programming w/ EEI, Copy SegD to A/B/C
- msp430x22x4_clks.c - Basic Clock, Output Buffered SMCLK, ACLK and MCLK/10
- msp430x22x4_fll_01.c - Basic Clock, Implement Auto RSEL SW FLL
- msp430x22x4_fll_02.c - Basic Clock, Implement Cont. SW FLL with Auto RSEL
- msp430x22x4_hfxtal.c - Basic Clock, MCLK Sourced from HF XTAL
- msp430x22x4_hfxtal_nmi.c - Basic Clock, LFXT1/MCLK Sourced from HF XTAL, NMI
- msp430x22x4_rosc.c - DCOCLK Biased with External Resistor Rosc
- msp430x22x4_dco_flashcal.c - DCO Calibration Constants Programmer
- msp430x22x4_ta_01.c - Timer_A, Toggle P1.0, TACCR0 Cont. Mode ISR, DCO SMCLK
- msp430x22x4_ta_02.c - Timer_A, Toggle P1.0, TACCR0 Up Mode ISR, DCO SMCLK
- msp430x22x4_ta_03.c - Timer_A, Toggle P1.0, Overflow ISR, DCO SMCLK
- msp430x22x4_ta_04.c - Timer_A, Toggle P1.0, Overflow ISR, 32kHz ACLK
- msp430x22x4_ta_05.c - Timer_A, Toggle P1.0, TACCR0 Up Mode ISR, 32kHz ACLK
- msp430x22x4_ta_06.c - Timer_A, Toggle P1.0, TACCR1 Cont. Mode ISR, DCO SMCLK
- msp430x22x4_ta_07.c - Timer_A, Toggle P1.0-3, Cont. Mode ISR, DCO SMCLK
- msp430x22x4_ta_08.c - Timer_A, Toggle P1.0-3, Cont. Mode ISR, 32kHz ACLK
- msp430x22x4_ta_09.c - Timer_A, Toggle P1.0-3, Cont. Mode ISR, HF XTAL ACLK
- msp430x22x4_ta_10.c - Timer_A, Toggle P1.1/TA0, Up Mode, DCO SMCLK
- msp430x22x4_ta_11.c - Timer_A, Toggle P1.1/TA0, Up Mode, 32kHz ACLK
- msp430x22x4_ta_12.c - Timer_A, Toggle P1.1/TA0, Up Mode, HF XTAL ACLK
- msp430x22x4_ta_13.c - Timer_A, Toggle P1.1/TA0, Up/Down Mode, DCO SMCLK
- msp430x22x4_ta_14.c - Timer_A, Toggle P1.1/TA0, Up/Down Mode, 32kHz ACLK
- msp430x22x4_ta_15.c - Timer_A, Toggle P1.1/TA0, Up/Down Mode, HF XTAL ACLK
- msp430x22x4_ta_16.c - Timer_A, PWM TA1-2, Up Mode, DCO SMCLK
- msp430x22x4_ta_17.c - Timer_A, PWM TA1-2, Up Mode, 32kHz ACLK
- msp430x22x4_ta_18.c - Timer_A, PWM TA1-2, Up Mode, HF XTAL ACLK
- msp430x22x4_ta_19.c - Timer_A, PWM TA1-2, Up/Down Mode, DCO SMCLK
- msp430x22x4_ta_20.c - Timer_A, PWM TA1-2, Up/Down Mode, 32kHz ACLK
- msp430x22x4_ta_21.c - Timer_A, PWM TA1-2, Up/Down Mode, HF XTAL ACLK
- msp430x22x4_ta_22.c - Timer_A, Ultra-Low Pwr Pulse Accumulator
- msp430x22x4_tb_01.c - Timer_B, Toggle P1.0, TBCCR0 Cont. Mode ISR, DCO SMCLK
- msp430x22x4_tb_02.c - Timer_B, Toggle P1.0, TBCCR0 Up Mode ISR, DCO SMCLK
- msp430x22x4_tb_03.c - Timer_B, Toggle P1.0, Overflow ISR, DCO SMCLK
- msp430x22x4_tb_04.c - Timer_B, Toggle P1.0, Overflow ISR, 32kHz ACLK
- msp430x22x4_tb_05.c - Timer_B, Toggle P1.0, TBCCR0 Up Mode ISR, 32kHz ACLK
- msp430x22x4_tb_06.c - Timer_B, Toggle P1.0, TBCCR1 Cont. Mode ISR, DCO SMCLK
- msp430x22x4_tb_07.c - Timer_B, Toggle P4.0-2, Cont. Mode ISR, DCO SMCLK
- msp430x22x4_tb_08.c - Timer_B, Toggle P4.0-2, Cont. Mode ISR, 32kHz ACLK
- msp430x22x4_tb_09.c - Timer_B, Toggle P4.0-2, Cont. Mode ISR, HF XTAL ACLK
- msp430x22x4_tb_10.c - Timer_B, PWM TB1-2, Up Mode, DCO SMCLK
- msp430x22x4_tb_11.c - Timer_B, PWM TB1-2, Up Mode, 32kHz ACLK
- msp430x22x4_tb_12.c - Timer_B, PWM TB1-2, Up Mode, HF XTAL ACLK

- msp430x22x4_tb_13.c - Timer_B, PWM TB1-2, Up/Down Mode, DCO SMCLK
- msp430x22x4_tb_14.c - Timer_B, PWM TB1-2, Up/Down Mode, 32kHz ACLK
- msp430x22x4_tb_15.c - Timer_B, PWM TB1-2, Up/Down Mode, HF XTAL ACLK
- msp430x22x4_wdt_01.c - WDT, Toggle P1.0, Interval Overflow ISR, DCO SMCLK
- msp430x22x4_wdt_02.c - WDT, Toggle P1.0, Interval Overflow ISR, 32kHz ACLK
- msp430x22x4_wdt_04.c - WDT+ Failsafe Clock, DCO SMCLK
- msp430x22x4_wdt_05.c - Reset on Invalid Address fetch, Toggle P1.0
- msp430x22x4_wdt_06.c - WDT+ Failsafe Clock, 32kHz ACLK
- msp430x22x4_uscia0_uart_01_115k.c USCI_A0, 115200 UART Echo ISR, DCO SMCLK
- msp430x22x4_uscia0_uart_01_115k_lpm.c USCI_A0, 115200 UART Echo ISR, DCO SMCLK, LPM4
- msp430x22x4_uscia0_uart_01_19200.c USCI_A0, 19200 UART Echo ISR, DCO SMCLK
- msp430x22x4_uscia0_uart_01_19200_2.c USCI_A0, UART 19200 Echo ISR, HF XTAL SMCLK
- msp430x22x4_uscia0_uart_01_9600.c USCI_A0, 9600 UART Echo ISR, DCO SMCLK
- msp430x22x4_uscia0_uart_05_9600.c USCI_A0, Ultra-Low Pwr UART 9600 Echo ISR, 32kHz ACLK
- msp430x22x4_uscia0_uart_06_9600.c USCI_A0, Ultra-Low Pwr UART 9600 String, 32kHz ACLK
- msp430x22x4_uscia0_uart_07_9600.c USCI_A0, Ultra-Low Pwr UART 9600 RX/TX, 32kHz ACLK
- msp430x22x4_uscia0_uart_08_9600.c USCI_A0, UART 9600 Full-Duplex Transceiver, 32kHz ACLK
- msp430x22x4_uscia0_spi_01.c USCI_A0, SPI Interface to HC164 Shift Register
- msp430x22x4_uscia0_spi_02.c USCI_A0, SPI Interface to HC165 Shift Register
- msp430x22x4_uscia0_spi_03.c USCI_A0, SPI Interface to HC165/164 Shift Registers
- msp430x22x4_uscia0_spi_09.c USCI_A0, SPI 3-Wire Master Incremented Data
- msp430x22x4_uscia0_spi_10.c USCI_A0, SPI 3-Wire Slave Data Echo
- msp430x22x4_uscib0_spi_01.c USCI_B0, SPI Interface to TLC549 8-Bit ADC
- msp430x22x4_uscib0_spi_02.c USCI_B0, SPI Interface to TLV1549 10-Bit ADC
- msp430x22x4_uscia0_irda_01.c USCI_A0 IrDA External Loopback Test, 8MHz SMCLK
- msp430x22x4_uscia0_irda_02.c USCI_A0 IrDA Monitor, 8MHz SMCLK
- msp430x22x4_uscia0_irda_03.c USCI_A0 IrDA Physical Layer Comm, 8MHz SMCLK
- msp430x22x4_uscib0_i2c_01.c USCI_B0 I2C Master to TMP100, Set P5.1 if Temp > 28C
- msp430x22x4_uscib0_i2c_02.c USCI_B0 I2C Master Interface to PCF8574, Read/Write
- msp430x22x4_uscib0_i2c_03.c USCI_B0 I2C Master Interface to DAC8571, Write
- msp430x22x4_uscib0_i2c_04.c USCI_B0 I2C Master RX single bytes from MSP430 Slave
- msp430x22x4_uscib0_i2c_05.c USCI_B0 I2C Slave TX single bytes to MSP430 Master
- msp430x22x4_uscib0_i2c_06.c USCI_B0 I2C Master TX single bytes to MSP430 Slave
- msp430x22x4_uscib0_i2c_07.c USCI_B0 I2C Slave RX single bytes from MSP430 Master
- msp430x22x4_uscib0_i2c_08.c USCI_B0 I2C Master TX multiple bytes to MSP430 Slave
- msp430x22x4_uscib0_i2c_09.c USCI_B0 I2C Slave RX multiple bytes from MSP430 Master
- msp430x22x4_uscib0_i2c_10.c USCI_B0 I2C Master RX multiple bytes from MSP430 Slave
- msp430x22x4_uscib0_i2c_11.c USCI_B0 I2C Slave TX multiple bytes to MSP430 Master

F. Literatur- und Quellenverzeichnis

[1] Texas Instruments Homepage
www.ti.com

[2] IAR Embedded Workbench für MSP430
http://www.iar.com/

[3] freie GNU Programmierumgebung für MSP430 (toolchain for MSP430)
http://mspgcc.sourceforge.net/

[4] Rowley CrossWorks für MSP430 Programmierumgebung
http://www.rowley.co.uk/

[5] Imagecraft ICC430 Programmierumgebung
http://www.imagecraft.com/software/

[6] msp430f2272, Texas Instruments MSP4302272 Datenblatt

[7] MSP-FET430 Users Guide.pdf, Texas Instruments MSP430x1xx Users Guide

[8] a430.pdf, MSP430 ASSEMBLER, LINKER, AND LIBRARIAN Programming Guide

[9] cs430.pdf, MSP430 C-SPY ROM-Monitor Supplement

[10] cw430.pdf, MSP430 C-SPY User Guide

[11] ew430.pdf, MSP430 WINDOWS WORKBENCH Interface Guide

[12] icc430,pdf, MSP430 C COMPILER Programming Guide

[13] MSP-FET430 Users Guide.pdf, MSP-FET430 FLASH Emulation Tool (FET)
Users Guide

[14] TI Simulator Users Guide.pdf, IAR CSPY w/TI Simulator User.s Guide

[15] Tutor.pdf, IAR CSPY w/TI Simulator User.s Guide

[16] xlink.pdf, IAR Linker and Library Tools, Reference Guide

[17] Datenblatt HD44780U, Hitachi Semiconductor