04. Februar 2015

MSP430 Education System v3.3

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.

© Copyright 2008 FTZ Leipzig e.V. an der HTWK Leipzig.
Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form ohne schriftliche Genehmigung des FTZ Leipzig unter Einsatz entsprechender Systeme reproduziert, verarbeitet, vervielfältigt oder verbreitet werden.

Kontaktadressen:

FTZ Leipzig e.V. an der HTWK Leipzig
Michael Eiserbeck / Daniel Käßler
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

- 21.03.2014 Initialversion
- 11.12.2014 Handbuch an die Hardware V3.3 angepasst
- 04.02.2015 Bemaßungen für das Board erstellt und hinzugefügt. Bilder aktualisiert.

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“ [0] 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 [0] oder direkt bei IAR Systems [0] bezogen werden.
Als Alternative steht ihnen auch der frei verfügbare und kostenlose GNU C-Compiler (GCC) [0] zur Verfügung, sowie weitere sehr gute Compiler der Firmen Rowley [0] und Imagecraft [0]. Über den Einsatz des entsprechenden Compilers kann jeder individuell entscheiden. Alle Compiler sind prinzipiell einsetzbar und lauffähig.

Dieses Handbuch beschreibt den Aufbau der Hardware 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.
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 den Versionen 3.3 und 2.0

Es hat sich einiges getan, es ist aber auch vieles geblieben.

Die Überarbeitung des MSP430 Education Systems hat einige Änderungen mit sich gebracht. Das Bild 1: *MSP430 Education System Version 2.0*zeigt die ältere Version 2.0, die auch im Buch „Mikrocontrollertechnik" von Prof. Matthias Sturm in der ersten Auflage beschrieben ist. Mit der zweiten Auflage wird im Buch auf das neue MSP430 Education Systems Version 3.3 (Bild 2) eingegangen.

Bild 1: MSP430 Education System Version 2.0

Bild 2: MSP430 Education System Version 3.3

Die wichtigsten Unterschiede gegenüber der älteren Version im Überblick:

- **Programmieren direkt über USB-Anschluss möglich!**
- **Virtuelle RS232-Schnittstelle** direkt im Rechner (Windows)
- Neuer Mikrocontroller **MSP430F2274**
- Stromversorgung über **USB** oder optional über 4x AA **Batterien**, welche mit Haltern einfach von unten an das Board gelötet werden können
- Versorgung des LCDs mit +5V über ein Spannungsregler.
- Kompaktere Bauweise
- Dreh-Encoder (24 Pulse /360 Grad)
- ADC-Eingang (Tiefpassfilter über Pfostenstecker erreichbar)
- 4x4 LED –Matrix mit Portextender-IC (I2C)
- Anschlussmöglichkeit für einen Servomotor (galvanisch entkoppelt, keine Stromversorgung über das Board vorgesehen!)
- Wegfall der RS232-Buchse (Rx/Tx Anschlüsse sind über Pfostenstecker erreichbar)
- Wegfall eines Potentiometers (Zweiter AD-Kanal ist über Pfostenstecker erreichbar)
- Wegfall der Infrarotschnittstelle mit Sender und Empfänger.

3 Features des MSP430 Education Systems 3.3

- **Controller**
  - MSP430F2274 (32kB + 256B Flash Memory / 1kB RAM)
  - Alle Pins des MSP430 sind über zwei im Rastermaß angeordneten Stiftleisten zugänglich (ausgenommen die Anschlüsse XIN und XOUT für Quarz).

- **Spannungsversorgung**
  - 3,3 V Spannungsversorgung und
  - 5,0 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 möglich (Wegfall Uhrenquarz)

- **Ein- und Ausgabe**
  - 8 LEDs an einem Port zur universellen Nutzung als Ausgabeelement
  - 4 Taster zur universellen Nutzung als Eingabeelement
  - 1 LC-Display (zweizeilig x 16 Zeichen) mit 4-Bit-Ansteuerung1 Potentiometer 100k linear
  - 1 analoger Eingang mit Tiefpassfilter beschalten
  - 1 Encoder 24 Pulse bei einer vollen Umdrehung
  - 1 Piezo-Lautsprecher mit Treiber IC
  - 4x4 LED Matrix mit I2C-Portextender (MCP23008)
  - 1 galvanisch getrennter Servomotoranschluss (open drain)

- **Programmierung**
  - Programmierung über einen eigenen OnBoard-Programmer (USB-Anschluss)

- **weitere Schnittstellen**
  - virtuelle RS-232 Schnittstelle über USB-Schnittstelle
  - I2C Schnittstelle mit Portextender (über extra Pin-Header abgreifbar inkl. 3V3 und 5V Spannungsanschluss)

- **Sonstiges**
  - Reset-Taster
  - LCD Kontrasteinstellung
  - Jumper zur Auswahl der Spannungsquelle (USB, Batterien)

- **Abmaße**
  - Hauptplatine 85 x 85 mm
  - Abmaße über Alles: 85 x 104 mm
  - Höhe ohne Batteriehalter 25 mm
  - Höhe mit Batteriehalter 40 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 P1 und P2 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 MSP430F2274 verwendet und über den auf dem Board befindlichen Programmieradapter als virtuelle Schnittstelle im Zielsystem eingebunden. 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 über das Board gibt das Bild 3. Für eine leichtere Entwicklung wurden auf der Platine alle verwendeten Pin-Belegungen der Peripherie mit aufgedruckt, ebenso wie die Pin Bezeichnung des MSP430F2274.

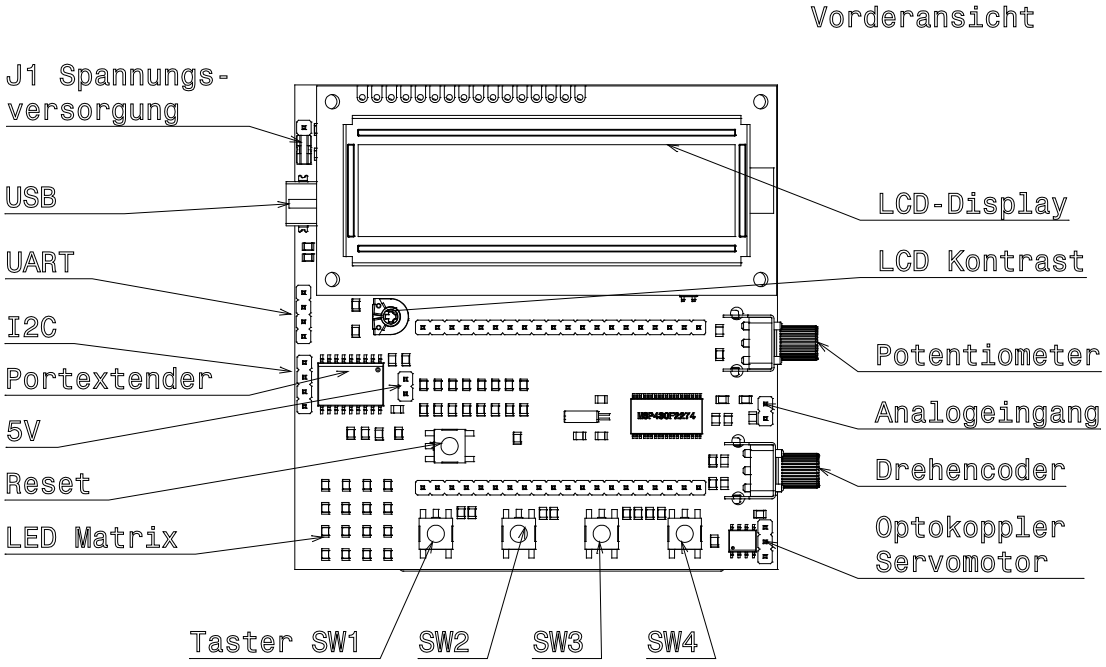


Bild 3: Hauptelemente des MSP430 Education Systems

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/SBWTCK	P1.2	
2	DV_{cc}	P1.3	
3	P2.5/R_{OSC}	P1.4	Taster SW1
4	DV_{ss}	P1.5	
5	XOUT/P2.7	~~P1.7~~	nicht verbunden
6	XIN/P2.6	~~P1.6~~	nicht verbunden
7	RST/NMI/SBWTDIO	P1.8	Taster RESET
8	P2.0/ACLK/A0/OA0I0	P1.9	Taster SW4
9	P2.1/TAINCLK/SMCLK/A1/OA0O	P1.10	Taster SW3
10	P2.2/TA0/A2/OA0I1	P1.11	Taster SW2
11	P3.0/UCB0STE/UCA0CLK/A5	P1.12	LCD RS-Leitung
12	P3.1/UCB0SIMO/UCB0SDA	P1.13	I^2C Datenleitung
13	P3.2/UCB0SOMI/UCB0SCL	P1.14	I^2C Taktleitung
14	P3.3/UCB0CLK/UCA0STE	P1.15	LCD Enable-Leitung
15	AV_{ss}	P1.16	
16	AV_{cc}	P1.17	
17	P4.0/TB0	P1.18	Servomotor
18	P4.1/TB1	P1.19	Drehencoder ENC A
19	P4.2/TB2	P1.20	Drehencoder ENC B
20	P4.3/TB0/A12/OA0O	P2.19	
21	P4.4/TB1/A13/OA1O	P2.18	
22	P4.5/TB2/A14/OA0I3	P2.17	
23	P4.6/TBOUTH/A15/OA1I3	P2.16	
24	P4.7/TBCLK	P2.15	
25	P3.4/UCA0TXD/UCA0SIMO	P2.14	Serielle Sendeleitung
26	P3.5/UCA0RXD/UCA0SOMI	P2.13	Serielle Empfangsleitung
27	P3.6/A6/OA0I2	P2.12	Analogeingang Stift P3
28	P3.7/A7/OA1I2	P2.11	Potentiometer
29	P2.3/TA1/A3/V_{REF-}/V_{eREF-} /OA1I1/OA1O	P2.10	Drehencoder ENC IRQ
30	P2.4/TA2/A4/V_{REF+}/V_{eREF+}/OA1I0	P2.9	Lautsprecher (Beeper)
31	P1.0/TACLK/ADC10CLK	P2.8	LED 0 (D2) (high activ) / LCD Datenleitung D4
32	P1.1/TA0	P2.7	LED 1 (D3) (high activ) / LCD Datenleitung D5
33	P1.2/TA1	P2.6	LED 2 (D4) (high activ) / LCD Datenleitung D6
34	P1.3/TA2	P2.5	LED 3 (D5) (high activ) / LCD Datenleitung D7
35	P1.4/SMCLK/TCK	P2.4	LED 4 (D6) (high activ)
36	P1.5/TA0/TMS	P2.3	LED 5 (D7) (high activ)
37	P1.6/TA1/TDI/TCLK	P2.2	LED 6 (D8) (high activ)
38	P1.7/TA2	P2.1	LED 7 (D9) (high activ)
		P1.1	+5V
		P2.20	GND

Im Bild 4 wird nochmals die Lage der Stiftleisten P1 und P2 dargestellt, durch die alle Pins des MSP430F1232 Mikrocontrollers frei verfügbar sind. Ausgenommen XIN und XOUT für den Quarz, da durch die erhöhte Kapazität und Einkopplungen von Störungen die Funktion beeinträchtigt wird.

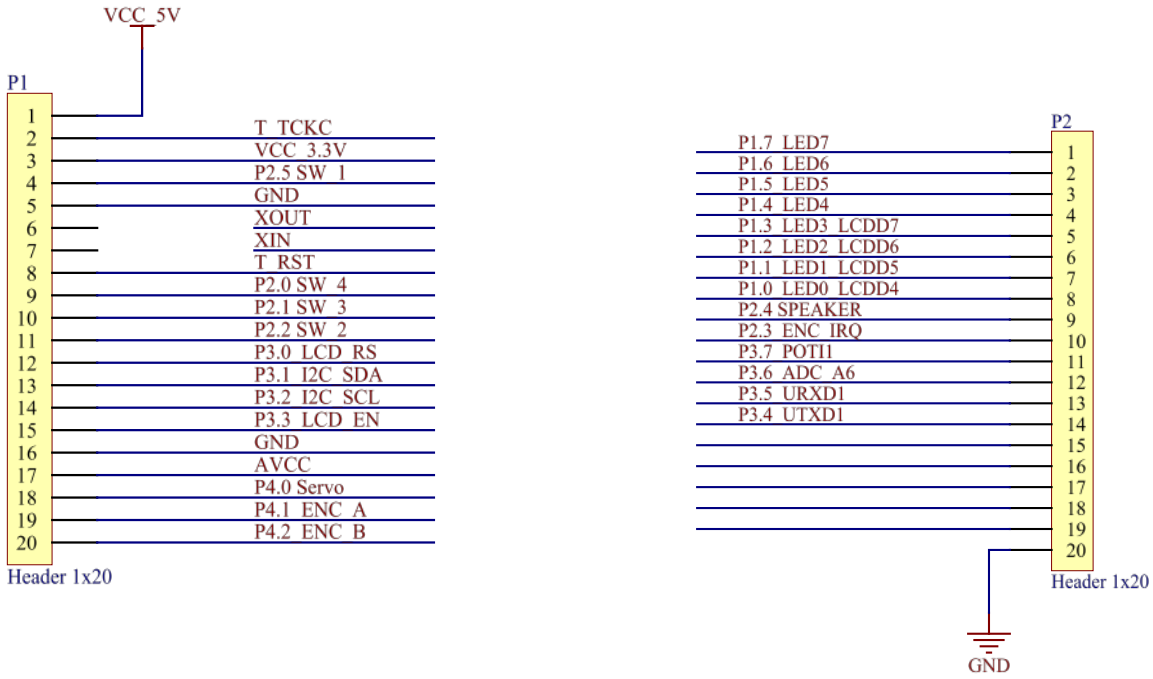


Bild 4: Belegung der Stiftleisten P1 und P2 mit Signalnamen

4.1 Der Mikrocontroller MSP430F2274

Der MSP430F2274 ist ein sehr leistungsfähiger 16-Bit Mikrocontroller und ein Nachfolgetyp des MSP430F1232 und kompatibel zum MSP430F2272. 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 in aktueller Form auf der Texas Instruments Homepage [0] gefunden werden.

Tabelle 2: Die wichtigsten Features des MSP430F2274 (Auszug aus dem Userguide von Texas Imstruments)

- 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 zugute 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 hergestellt werden. Der in dieser Lösung eingesetzte MSP430F2274 ist weiterhin mit 32 kByte Flashspeicher und 1 kByte RAM ausgestattet und ermöglicht damit sehr leistungsfähige Applikationen.

MSP430F22x4 Device Pinout, DA Package

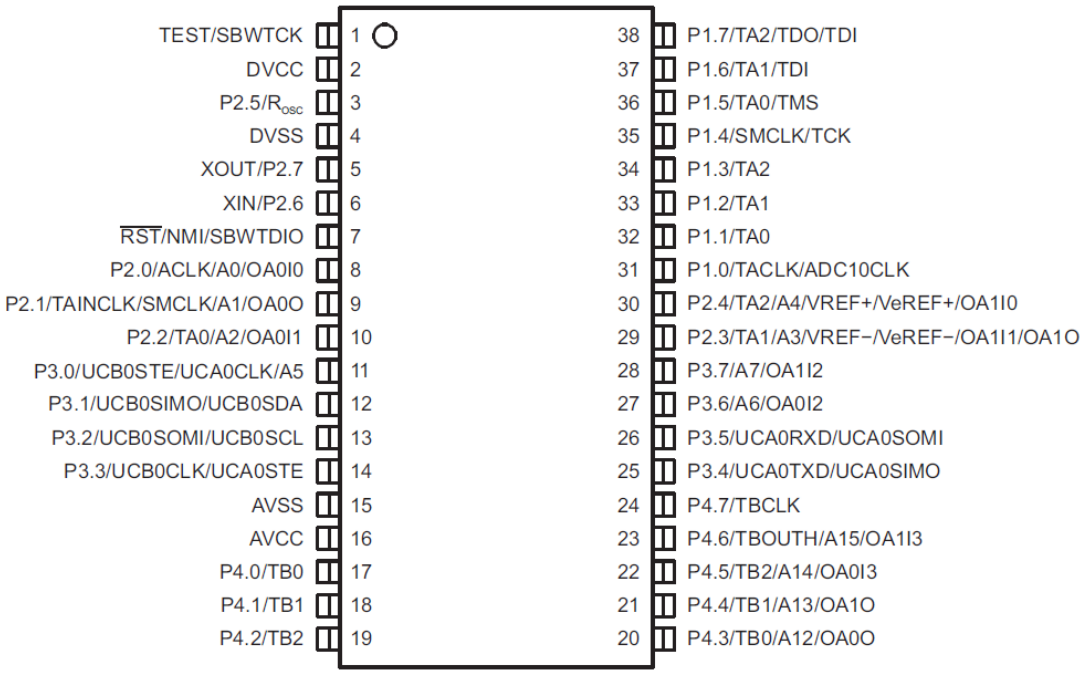


Bild 5: Pinbelegung des MSP430F2274 (Quelle: Datasheet SLAS504G Texas Instruments)

Im Bild 5 ist die komplette Pinbelegung des MSP430F2274 Mikrocontrollers dargestellt. An Hand dieses Bildes lassen sich die einzelnen Funktionseinheiten sowie die verwendeten Pins erkennen. In Bild 6 wird der interne Aufbau des MSP430F2274 mit Hilfe eines Blockschaltbildes dargestellt. Weitere Details können dem Datenblatt des MSP430F2274 [0] entnommen werden.

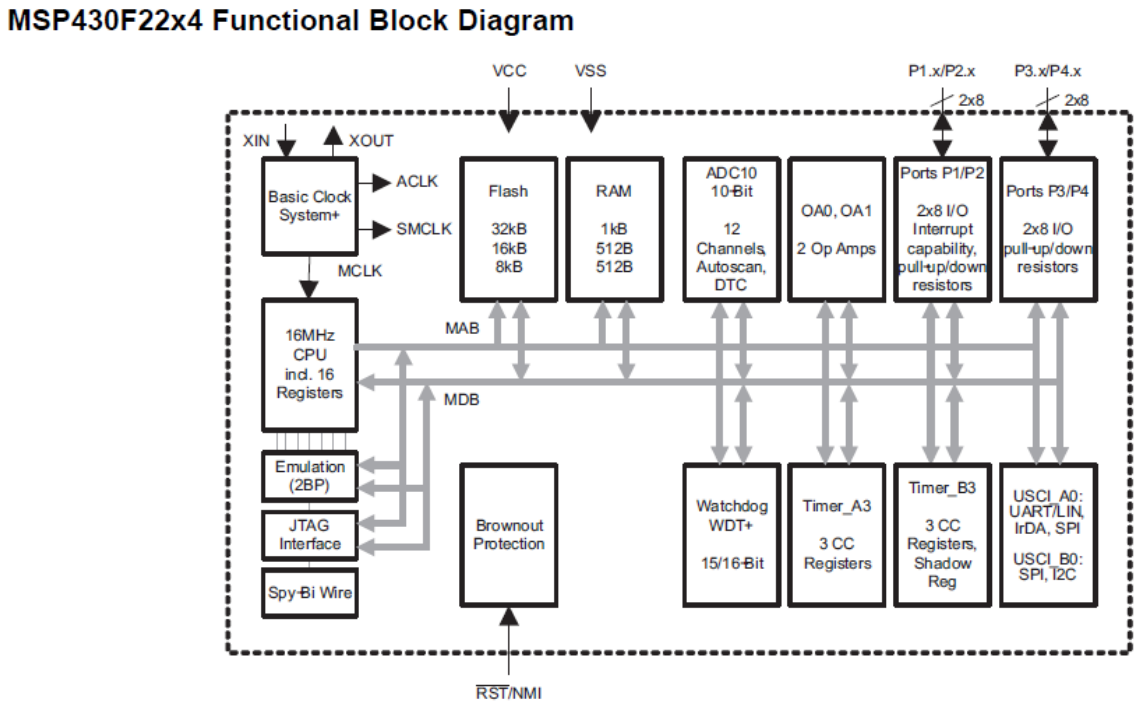


Bild 6: Blockdiagramm des MSP430F2274 (Quelle: Datasheet SLAS504G Texas Instruments)

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

zeigt alle möglichen Interrupts mit der jeweiligen Adresse nochmals in einer Übersicht. Weitere Funktionsweisen und Erklärungen können dem MSP430F2274 Datenblatt [0] und dem MSP430x2xx Family User's Guide [0] 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 (Quelle: Datasheet SLAS504G Texas Instruments)

INTERRUPT SOURCE	INTERRUPT FLAG	SYSTEM INTERRUPT	WORD ADDRESS	PRIORITY
Power-up External reset Watchdog Flash key violation PC out-of-range(1)	PORIFG RSTIFG WDTIFG KEYV(2)	Reset	0FFFEh	31, highest
NMI Oscillator fault Flash memory access violation	NMIIFG OFIFG ACCVIFG(2)(3)	(non)-maskable, (non)-maskable, (non)-maskable	0FFFCh	30
Timer_B3	TBCCR0 CCIFG(4)	maskable	0FFFAh	29
Timer_B3	TBCCR1 and TBCCR2 CCIFGs, TBIFG(2)(4)	maskable	0FFF8h	28
			0FFF6h	27
Watchdog Timer	WDTIFG	maskable	0FFF4h	26
Timer_A3	TACCR0 CCIFG (see Note 3)	maskable	0FFF2h	25
Timer_A3	TACCR1 CCIFG TACCR2 CCIFG TAIFG(2)(4)	maskable	0FFF0h	24
USCI_A0/USCI_B0 Receive	UCA0RXIFG, UCB0RXIFG(2)	maskable	0FFEEh	23
USCI_A0/USCI_B0 Transmit	UCA0TXIFG, UCB0TXIFG(2)	maskable	0FFECh	22
ADC10	ADC10IFG(4)	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (eight flags)	P2IFG.0 to P2IFG.7(2)(4)	maskable	0FFE6h	19
I/O Port P1 (eight flags)	P1IFG.0 to P1IFG.7(2)(4)	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
(5)			0FFDEh	15
(6)			0FFDCh to 0FFC0h	14 to 0, lowest

4.2 Spannungsversorgung

Das MSP430 Education System V3.3 arbeitet mit zwei verschiedenen Spannungsebenen. Der Mikrocontroller und die meisten Peripheriesysteme werden mit +3,3 V versorgt und das LC-Display wird mit +5 V versorgt.

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

Die Stromversorgung wird über den USB Anschluss realisiert. Dadurch kann bei nicht Beachtung der maximalen Stromaufnahme der USB Host beschädigt werden! Benutzung auf eigene Gefahr!

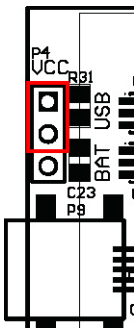
Ein gleichzeitiges betreiben mit USB-Spannung und Batteriespannung ist nicht vorgesehen! Es können dadurch Schäden am Board und am USB Host auftreten! Benutzung auf eigene Gefahr!

Ein externes Netzteil wird bei der neuen Version nicht weiter benötigt, es kann aber ein USB-Netzteil verwendet werden. Die Spannungsversorgung kann über zwei Wege hergestellt werden. Es kann einmal der USB-Anschluss genutzt oder die vorgesehenen Batteriehalter mit vier AA Mignon Batterien verwendet werden.

Die Spanungsquelle wird über einen Jumper eingestellt. Der Jumper dient darüber hinaus auch als sichere Möglichkeit das Board bei Batteriebetrieb auszuschalten. Ein gleichzeitiger betrieb mit USB-Spannung und Batteriespannung wird nicht unterstützt! Bei nicht beachten können Schäden auftreten!

Jumpereinstellung:

USB Versorgung



Jumpereinstellung:

Batterie Versorgung, wenn keine Batteriehalter bestückt sind Jumpereinstellung **AUS**.

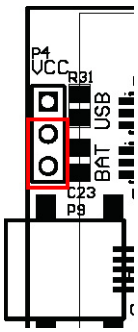


Bild 7: Jumpereinstellung MSP430 Education System

Ein Programmieren mit der Einstellung „Batterie“ wird nicht empfohlen, ist allerdings möglich.

Für eigene Erweiterungen können die beiden Spannungsebenen an mehreren Stellen abgegriffen werden. Für die 3,3V Spannung können die Pfostenstecker P6, P5 und P1 genutzt werden und für die 5,0V Spannung können die Pfostenstecker P1 und P7 genutzt werden.

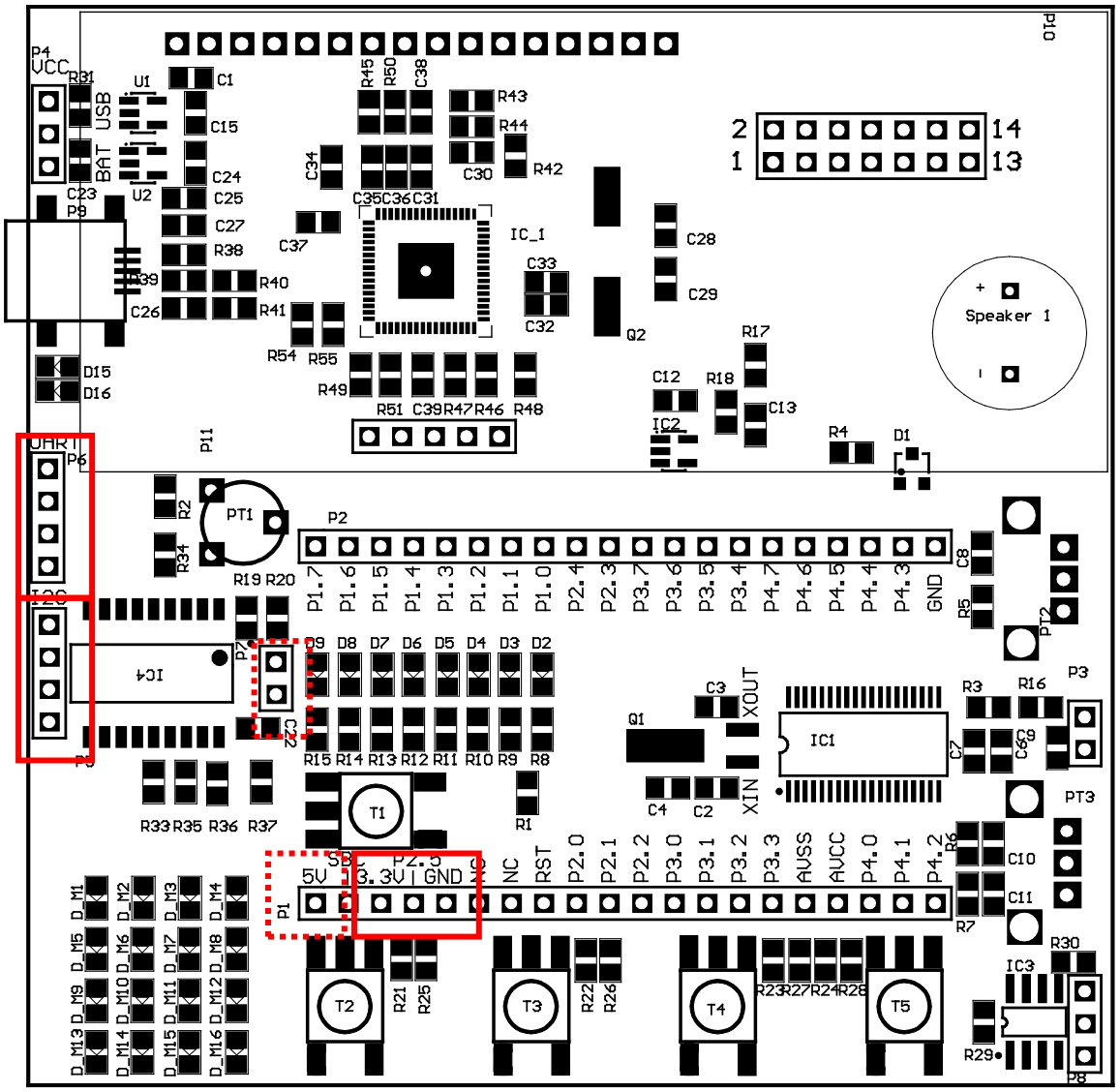


Bild 8: Möglichkeit zum Spannungsabgriff 3,3V und 5V (gestrichelt)

Hinweis: Die Batteriehalter sind optional und müssen unter Umständen vom Benutzer angelötet werden.

4.3 Programmier-Schnittstelle

Die JTAG-Schnittstelle dient zum einen als Programmierschnittstelle und 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.
Das Board verfügt über einen integrierten SBW-Programmerinterface. Das Interface stellt im Rechner über die USB-Schnittstelle eine Programmierschnittstelle und eine virtuelle Kommunikationsschnittstelle zur Verfügung. Über die Programmierschnittstelle wird das Board mittels Spy-by-Wire programmiert und auf Fehler überprüft. Über die virtuelle COM-Schnittstelle können Daten zwischen den MSP430 und dem Rechner über das UART-Protokoll ausgetauscht werden. Beide Schnittstellen benötigen keinen extra Treiber, er wird von Windows zur Verfügung gestellt bzw. durch die Programmierumgebung IAR. Der Vollständigkeit halber folgt nun noch eine kleine Beschreibung der JTAG- bzw. der SBW-Schnittstelle.

4.3.1 4-Draht-JTAG-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.
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 Befehlscodes 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 MSP430F2274 verfügt neben dem klassischen JTAG-Interface mit den Signalen TMS, TDI, TDO und TCK auch über eine neue serielle Version welche mit 2 Leitungen auskommt. Dieser wird als Spy-by-Wire bezeichnet und wurde speziell für MSP430-Mikrocontroller von Texas Instruments entwickelt. Die Programmiersignale werden über die TEST- und RESET-Leitung an den Mikrocontroller angeschlossen.

4.4 Serielle Schnittstelle RS-232

Die serielle Kommunikation wird über eine RS-232 Schnittstelle zur Verfügung gestellt. Durch Entfernen der Widerstände R49 und R51 kann der UART-USB-Wandler vom Mikrocontroller getrennt werden. Dadurch werden die entsprechenden Controllerpins für eine alternative Beschaltung frei. (Hinweis: Die Widerstände befinden sich unter dem LC-Display welches vorher entfernt werden muss. Das LC-Display ist nicht mit dem Board verlötet, es ist nur durch die vier Abstandshalter befestigt.)

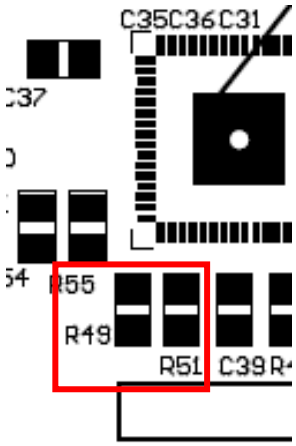


Bild 9: Trennen der USB-Seriell-Verbindung durch entfernen der Widerstände R49 und R51

4.5 Leuchtdioden

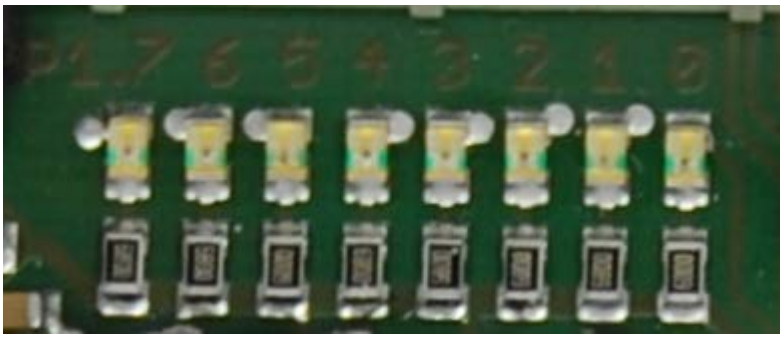


Bild 10: Installierte Leuchtdioden (Beispielbild)

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 die Ansteuerung des LCDs verwendet.

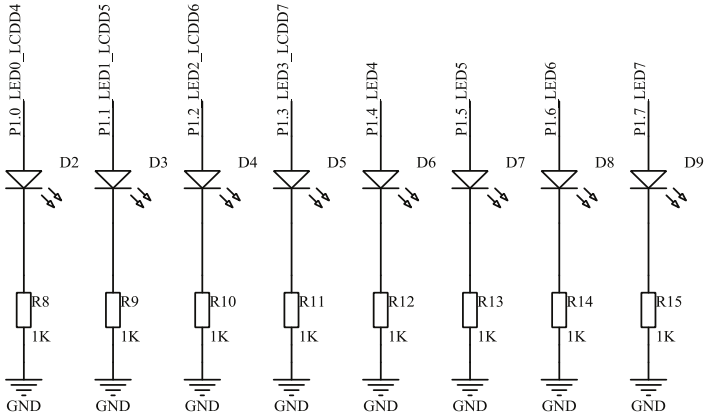


Bild 11: Beschaltung und Position der Leuchtdioden

Die Leuchtdioden wurden so an den Mikrocontroller angeschlossen, dass Sie bei High-Pegel (high activ) am Mikrocontroller leuchten.

Die Anschlussbelegung der Leuchtdioden zeigt das Bild 11. Tabelle 4 zeigt die Anschlussbelegung der Leuchtdioden am Mikrocontroller, der Stiftleiste P2 und den verwendeten Port.

Tabelle 4: Anschlussbelegung der Leuchtdioden

Port	Stiftleiste	µC Pin	zugehörige Leuchtdioden
P1.0	P2.8	31	LED 0 (D2) (high activ) / LCD Datenleitung D4
P1.1	P2.7	32	LED 1 (D3) (high activ) / LCD Datenleitung D5
P1.2	P2.6	33	LED 2 (D4) (high activ) / LCD Datenleitung D6
P1.3	P2.5	34	LED 3 (D5) (high activ) / LCD Datenleitung D7
P1.4	P2.4	35	LED 4 (D6) (high activ)
P1.5	P2.3	36	LED 5 (D7) (high activ)
P1.6	P2.2	37	LED 6 (D8) (high activ)
P1.7	P2.1	38	LED 7 (D9) (high activ)

4.6 Taster

Da Taster wichtige Funktionen in vielen unterschiedlichen Systemen übernehmen, wurden auf dem MSP430 Education System vier Taster verwendet. 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.

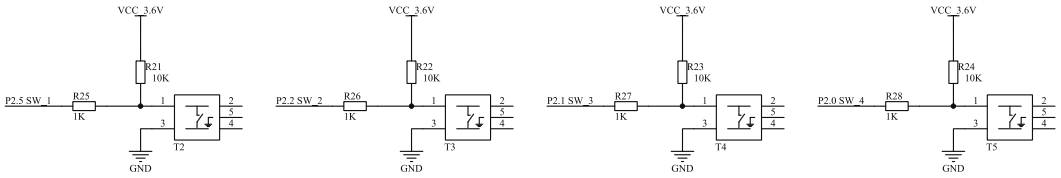


Bild 12: Beschaltung der Taster

Wie in Bild 12: zu sehen ist, liegen die angeschlossen Pins des Mikrocontrollers (über Vorwiderstände R21-R24) direkt an der Versorgungsspannung (VCC). Die Widerstände R25-R28 dienen der Strombegrenzung beim Drücken der Taster. Die Pins 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 Pins 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 Pin, erkannt werden.

Da die Taster an den externen Interruptleitungen angeschlossen sind, ist auch eine interruptgesteuerte Entprellung möglich. Die Widerstände R25-R28 dienen der Absicherung der Mikrocontrollereingänge, um eine Überlastung dieser zu verhindern.

Die Tabelle 5 stellt die Anschlussbelegung noch einmal übersichtlich dar.

Tabelle 5: Anschlussbelegung der Taster

Port	Stiftleiste	µC Pin	Peripheriesystem
P2.5	P1.4	3	Taster SW1
P2.2	P1.11	10	Taster SW2
P2.1	P1.10	9	Taster SW3
P2.0	P1.9	8	Taster SW4

Dreh-Encoder

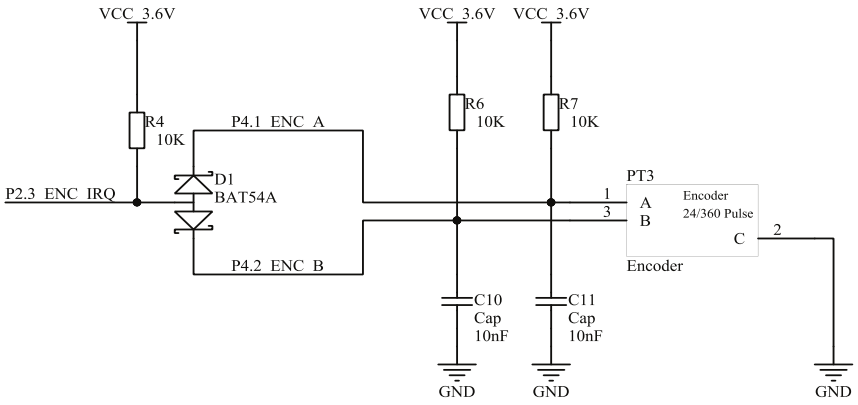


Bild 13: Dreh-Encoder

Das MSP430 Education System ist mit einem Dreh-Encoder ausgestattet. Der Drehencoder besitzt drei Anschlüsse und erzeugt bei einer vollständigen Umdrehung 24 Impulse an zwei seiner Anschlüsse. Der dritte Anschluss wird als Bezugspotential (VCC) genutzt und ergibt daraus ein *„low active"* verhalten. Um Störungen zu vermeiden sind beide Ausgänge mit einem Tiefpassfilter ausgestattet, um so Fehlimpulse zu vermeiden.

Seine Funktionsweise ist relativ einfach. Bei jedem Impuls wird das Signal beider Anschlüsse von VCC nach Masse gezogen. Je nachdem welcher der beiden Anschlüsse zuerst auf VCC gezogen wird kann neben dem eigentlichen Impuls auch die Drehrichtung erkannt werden.

Der Dreh-Encoder wurde aus Mangel an interruptfähigen Ports an einen nicht interruptfähigen Port angeschlossen. Nun ist das ständige Auslesen der beiden Pin-Eingänge aufwändig und außerdem können auch Impulse verloren gehen, wenn der Controller anderweitige Routinen abarbeiten muss. Um den Drehencoder dennoch über einen Interrupt abfragen zu können, werden die beiden Signale, mittels einer BAT-Diode auf einen freien interruptfähigen Pin (P2.3) gelegt. Die beiden internen Dioden trennen die Signale voneinander und geben ein entsprechendes Signal an den Eingang weiter. So kann der Drehencoder über nur einen Interrupt-Pin ausgelesen werden. Die Pinbelegung ist in folgender Tabelle zu sehen.

Tabelle 6: Anschlussbelegung des Drehencoders

Port	Stiftleiste	µC Pin	Peripheriesystem
P4.1	P1.19	18	Drehencoder ENC A
P4.2	P1.20	19	Drehencoder ENC B
P2.3	P2.10	29	Drehencoder ENC IRQ

4.7 LED-Matrix / Portextender (MCP23008)

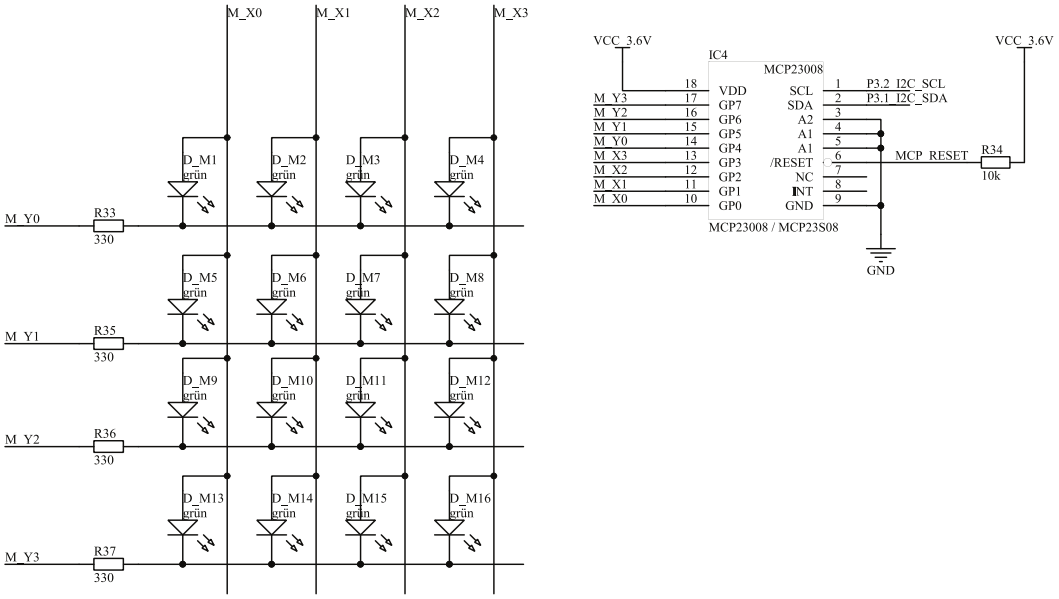


Bild 14: LED Matrix 4x4 / Portextender MSP23008

Das MSP430 Education System ist mit einer LED 4x4 Matrix ausgestattet. Für die Ansteuerung der LED-Matrix werden die LED, wie in Bild 14 gezeigt, angeordnet. So entsteht eine Matrix, die mit acht Steuersignalen alle 16 LED angesteuert werden kann.

Ein kleines Beispiel:
Die LED *D_M1* oben links (Bild 14: LED Matrix 4x4) soll angesteuert werden. Damit die LED *D_M1* leuchtet, muss am Anschluss *M_Y0* GND und am Anschluss *M_X0* VCC anliegen. Zugleich muss aber auch dafür gesorgt werden, dass an den Anschlüssen *M_X1*, *M_X2* und *M_X3* kein VCC anliegt sondern GND oder aber die Anschlüsse nicht beschalten sind, da sonst die LED D_M6, D_M7 und D_M8 auch leuchten würden.

Nach diesem Schema kann jede LED angesprochen werden. Möchte man aber eine diagonale Linie darstellen, kommt man schnell zu der Erkenntnis, dass es ein Problem gibt. Es leuchten alle LED. Für eine Lösung können sie gerne in das entsprechende Beispielprogramm schauen.

Der MSP430 hat keine acht Pins mehr zur Verfügung deshalb kann die Matrix so nicht angesteuert werden, auch wenn der MSP430 das prinzipiell könnte. Damit die Matrix trotzdem angesteuert werden kann, wird ein Portextender genutzt. Mit einem Portextender ist es möglich mit einigen Steuerleitungen mehrere IO-Leitungen zu bedienen. Der Protextender, der hier verwendet wird, nutzt als Übertragungsprotokoll den I2C-Bus, der im MSP430 als Hardwarebaugruppe zur Verfügung steht. So werden nur zwei Pins verwendet.

Tabelle 7: Anschlussbelegung des LED Matrix Portextender

Port	Stiftleiste	µC Pin	Peripheriesystem
P3.1/UCB0SDA	P1.13	12	I^2C Datenleitung
P3.2/UCB0SCL	P1.14	13	I^2C Taktleitung

Das I2C-Protokoll ist relativ einfach und funktioniert bei einer einfachen Master–Slave-Kommunikation wie folgt:

- Der Master(MSP430) sendet ein Startzeichen (definierter Pegelwechsel auf der Daten- und Takt-Leitung)
- Der Master sendet anschließend die Slave-Adresse, hier die vom Portextender. Die Adresse ist eindeutig und vom Hersteller vorgegeben, sie kann meist durch Adress-Pins (A0, A1, A2) am IC, um einige Bit hardwareseitig geändert werden.
- Der Slave bestätigt seine eigene Adresse durch ein „ACK“ und der Master kann Daten senden oder anfordern (abhängig vom R/W-Bit in der Slave-Adresse). Sollte sich kein IC auf dem Bus angesprochen fühlen, dann bleibt das „ACK“ aus und der MSP430 meldet ein „NACK“, der über einen Interrupt abgefangen werden kann.
- Ist die Übertragung abgeschlossen sendet der Master ein Stoppzeichen (definierter Pegelwechsel auf der Daten- und Takt-Leitung) und der BUS ist für die nächste Kommunikation frei.

Daraus resultiert, dass für jede Kommunikation immer ein Startzeichen, ein Adressbyte, ein oder mehrere Datenbytes und ein Stoppzeichen verwendet werden müssen (Siehe dazu Bild 15**: I2C Start- und Stoppbedingung (Quelle: Datasheet MCP23008)**).

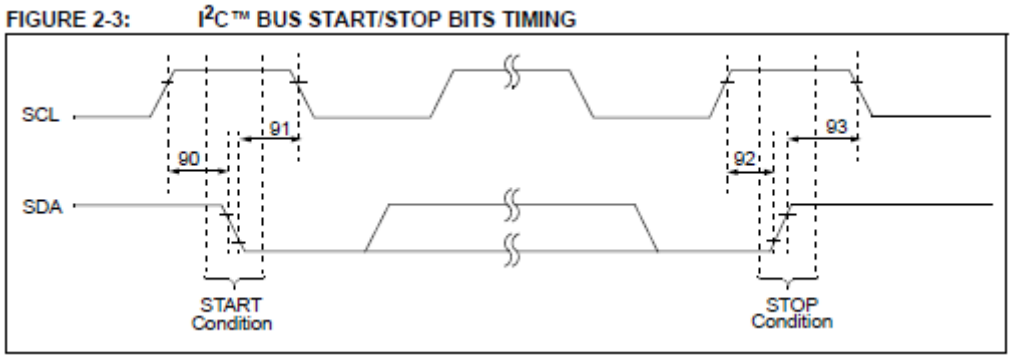


Bild 15: I2C Start- und Stoppbedingung (Quelle: Datasheet MCP23008)

Die Adresse des Protextenders lautet in Hex 0x20h bzw. in Binär 0b0010 0000. Es ist wichtig zu wissen, dass ein zu lesendes Byte auf den I2C-Bus mit einem R/W-Bit (0b0010 000x, hier als „X“ markiert) in der Slave-Adresse angezeigt wird. Das bedeutet allerdings auch, dass dieses Bit irgendwer mal setzen muss (Bild 16: **Aufbau Slave-Adresse (Quelle: Datasheet MCP23008)**.

Dazu gibt es zwei Methoden.
Einmal kann der Benutzer einfach zwei Adressen definieren und diese entsprechend senden. Zum Lesen wäre dass in unserem Beispiel eine 0x2**1**h bzw. 0b0010 000**1** und zum Schreiben eine 0x2**0**h bzw. 0b0010 000**0**. Anschließend wird beim Schreiben ein Datenbyte gesendet und beim Lesen entsprechende Takte auf der Taktleitung ausgegeben um ein Byte empfangen zu können, da nur der Master die Taktleitung bedient.
Auf der Anderen Seite kann dass die I2C-Baugruppe übernehmen und dort einfach die Slave-Adresse angeben, es muss aber immer angeben werden ob man lesen oder schreiben möchte und anschließend einfach die entsprechenden Bytes in das Senderegister schreiben bzw. aus dem Empfangsregister lesen.

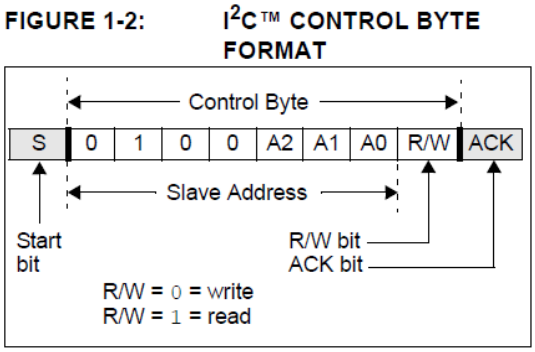


Bild 16: Aufbau Slave-Adresse (Quelle: Datasheet MCP23008)

Der Portextender besitzt wie der MSP430 Register, zehn um genau zu sein. Für eine einfache Ausgabe sind nur zwei Register interessant, einmal das **IODIR** und das **GPIO**-Register.
Das **IODIR**-Register gibt an, ob sich das Ausgaberegister (GPIO) als Eingang oder als Ausgang verhalten soll. Das Register muss einmal mit 0x00h beschrieben werden, um alle Pins im Ausgaberegister als Ausgang zu konfigurieren, denn es sind alle nach dem Anschalten standartmäßig als Eingang konfiguriert.
Und das **GPIO**-Register in dem die Ausgabewerte (Byte) geschrieben werden. Ist ein neuer Wert in das Ausgaberegister geschrieben worden, nehmen die Ausgangspins am Portextender die Werte an.

Im Bild 17 ist ein Auszug aus dem Beispielprogramm zu sehen, indem die verwendeten Register mit einer #define Anweisung, für eine bessere Übersicht im Programmcode, die entsprechenden Werte aus dem Datenblatt zugewiesen worden sind.

Ein Hinweis zur Verwendung einzelner Pins in einem Register. Sollen einzelne Bit eines Registers manipuliert werden empfiehlt es sich das ganze Register zu lesen, die entsprechenden Bits zu ändern und das Register mit den geänderten Daten zu schreiben. Nur das Schreiben eines Registers, kann zur Folge haben, dass unbeabsichtigt andere Bits überschrieben werden, besonders wenn der Portextender von sich aus Status-Bits ändert.

```
//Register und Adressen MCP23008
#define MCP23008        0x20 //SlaveAddress

#define IODIR           0x00 //I/O DIRECTION (IODIR) REGISTER
#define IPOL            0x01 //INPUT POLARITY (IPOL) REGISTER
#define GPINTEN         0x02 //INTERRUPT-ON-CHANGE CONTROL (GPINTEN) REGISTER
#define DEFVAL          0x03 //DEFAULT COMPARE (DEFVAL) REGISTER FOR INTERRUPT-ONCHANGE
#define INTCON          0x04 //INTERRUPT CONTROL (INTCON)REGISTER
#define IOCON           0x05 //CONFIGURATION (IOCON)REGISTER
#define GPPU            0x06 //PULL-UP RESISTOR CONFIGURATION (GPPU) REGISTER
#define INTF            0x07 //INTERRUPT FLAG (INTF) REGISTER
#define INTCAP          0x08 //INTERRUPT CAPTURE (INTCAP) REGISTER
#define GPIO            0x09 //PORT (GPIO) REGISTER
#define OLAT            0x0A //OUTPUT LATCH REGISTER (OLAT)
```

Bild 17: Auszug Beispielprogramm Portextender Register

4.8 LC-Display

Bild 18: Installiertes Dotmatrix Display (Beispielbild)

Das eingesetzte Display (Bild 18) 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 19 zeigt die Beschaltung des Displays auf dem MSP430 Education System.

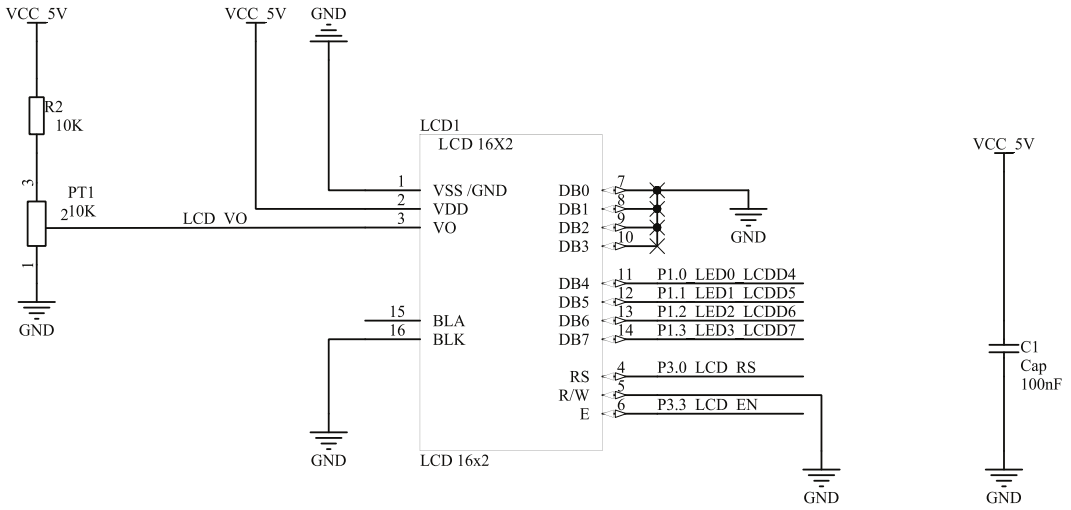


Bild 19: Beschaltung des LC-Displays

Es wurde der 4-Bit Modus gewählt, damit nicht alle LED-Pins indirekt durch das Display angesprochen werden.

Das Display wird mit 5V betrieben und benötigt eigentlich auch Signalpegel von 5V, um eine logische Eins zu erkennen. Da das Display den High-Pegel laut Datenblatt schon ab 2.0V erkennt wurde auf ein Pegelwandler verzichtet und die Pins direkt am MSP430 angeschlossen. Durch die gewählte Beschaltung ist es nicht möglich Daten vom Display zu lesen. Die R/W-Leitung des Displays ist fest auf Massepegel gelegt und somit auf „schreiben“ codiert.

Über das Potentiometer PT1 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	P1.12	11	LCD RS-Leitung
P3.3	P1.15	14	LCD Enable-Leitung
P1.0	P2.8	31	LCD Datenleitung D4 / LED 0 (D2) (high activ)
P1.1	P2.7	32	LCD Datenleitung D5 / LED 1 (D3) (high activ)
P1.2	P2.6	33	LCD Datenleitung D6 / LED 2 (D4) (high activ)
P1.3	P2.5	34	LCD Datenleitung D7 / LED 3 (D5) (high activ)

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 Befehlssatz 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 [0] nochmals in ausführlicher Form nachgelesen werden. Die wichtigsten Eigenschaften des Displaycontrollers (HD44780) im Überblick:

- 4-Bit oder 8-Bit MPU-Interface (hier 4-Bit verwendet)
- Integriertes Display RAM für 80 Zeichen
- Zeichengenerator ROM 5x7: 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 HD44780 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.

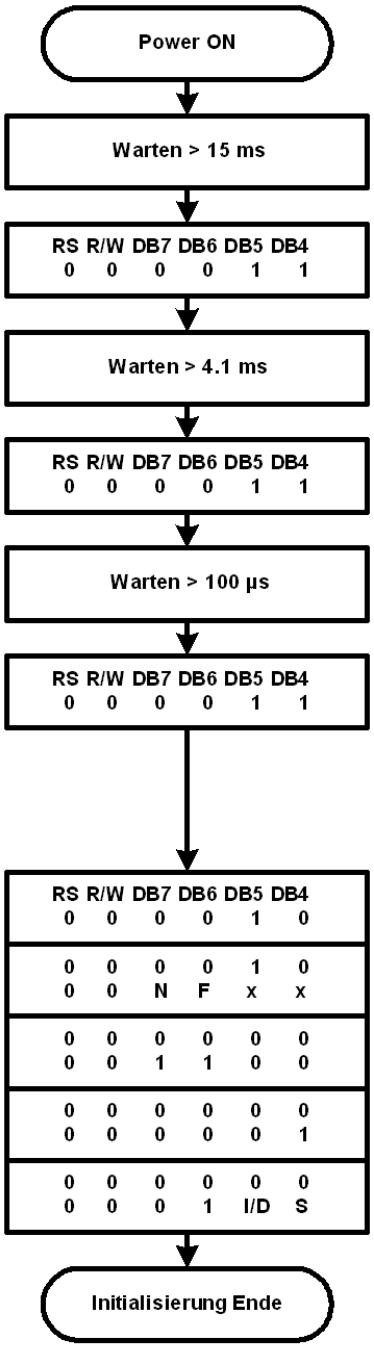
Mit dem Signal RS wird dem Displaycontroller mitgeteilt, ob Anweisungen, Instruktionscodes (RS=0) oder Daten (RS=1) übertragen werden. Bei der fallenden Flanke des Enable Signals (E) übernimmt der Display-Controller die Daten.

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 schreiben**	DL	= 0	4-Bit Datenbusbreite
		N	= 1	2-Zeilen Display
		F	= 0	5x7 Zeichensatz
3.	**Display On/Off**			
4.	**Funktion schreiben**	D	= 0	Display aus
		C	= 0	Cursor aus
		B	= 0	Blinkfunktion aus
5.	**Entry mode set**	I/D	= 1	Inkrementiert die DD-RAM Adresse nach dem Lesen/Schreiben eines Zeichens
		S	= 0	Kein schieben des Displays
6.	**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 Funktionen der in Tabelle 10 dargestellten Befehle, werden nachfolgend erläutert.

Clear Display

Das Display wird gelöscht.

***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 20 zeigt den Standardzeichensatz in einer Übersicht.

Lower 4 bit \ Upper 4 bit	0000 ($0x)	0010 ($2x)	0011 ($3x)	0100 ($4x)	0101 ($5x)	0110 ($6x)	0111 ($7x)	1010 ($Ax)	1011 ($Bx)	1100 ($Cx)	1101 ($Dx)	1110 ($Ex)	1111 ($Fx)
xxxx0000 ($x0)	CG RAM (0)		0	@	P	`	p		ー	タ	ミ	α	p
xxxx0001 ($x1)	(1)	!	1	A	Q	a	q	。	ア	チ	ム	ä	q
xxxx0010 ($x2)	(2)	"	2	B	R	b	r	「	イ	ツ	メ	β	θ
xxxx0011 ($x3)	(3)	#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
xxxx0100 ($x4)	(4)	$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
xxxx0101 ($x5)	(5)	%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
xxxx0110 ($x6)	(6)	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111 ($x7)	(7)	'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
xxxx1000 ($x8)	CG RAM (0)	(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
xxxx1001 ($x9)	(1)	)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010 ($xA)	(2)	*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
xxxx1011 ($xB)	(3)	+	;	K	[	k	{	ォ	サ	ヒ	ロ	x	万
xxxx1100 ($xC)	(4)	,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
xxxx1101 ($xD)	(5)	−	=	M	]	m	}	ュ	ス	ヘ	ン	₺	÷
xxxx1110 ($xE)	(6)	.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
xxxx1111 ($xF)	(7)	/	?	O	_	o	←	ッ	ソ	マ	°	ö	■

Bild 20: Standardzeichensatz des Displays

Weitere Einstellungen und Informationen zu diesem Display können aus dem zugehörigen Datenblatt [0] entnommen werden. In diesem Datenblatt wird ebenfalls die genaue Vorgehensweise beim Entwerfen und Darstellen eigener Zeichen beschrieben.

4.9 Lautsprecher

Der Lautsprecher wird mit einem Buffer IC angesprochen. Dies ist eine einfache 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.

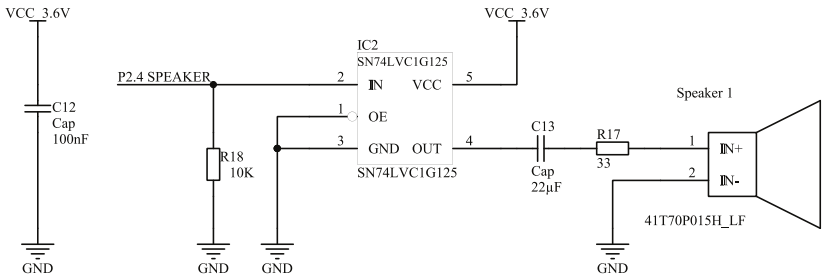


Bild 21: Beschaltung des Lautsprechers

Tabelle 12: Anschlussbelegung des Lautsprechers

Port	Stiftleiste	µC Pin	Peripheriesystem
P2.4	P2.9	30	Lautsprecher (Beeper)

4.10 Servo-Ansteuerung

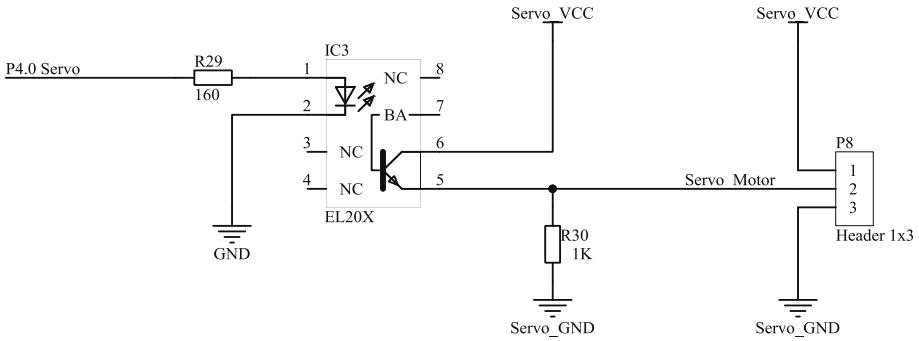


Bild 22: Beschaltung der Servo-Ansteuerung

Ein Modellbauservo kleiner Leistung wird über eine Pulsweitenmodulation (PWM) angesprochen. Mit der Pulsweite im Signal kann so der Servomotor auf einen bestimmten Winkel gefahren werden. Der MSP430 bietet für solche Anwendungen mit der Timer-Baugruppe einen sehr bequem Weg an, einen Grundtakt und mit der entsprechenden

Modulation zu erzeugen. Der hier natürlich genutzt werden kann. Ist die Timer-Baugruppe entsprechend konfiguriert so gibt diese selbstständig das PWM-Signal an den Pin P4.0 aus und steuert den Optokoppler an. Der Optokoppler wird genutzt um Schäden am MSP430 bzw. am Board zu verhindern. Durch ihn wird das Signal galvanisch getrennt und ermöglicht so auch das treiben einer größeren Spannung mit mehr Leistung. Das Board ist **nicht** in der Lage genügend Strom für ein Servomotor zur Verfügung zu stellen! Sie müssen eine externe Spannungsquelle nutzen.
Der Servomotor bekommt sein Steuersignal(2) von der Stiftleiste P8 wobei an P8 die Spannung(1) und das Bezugspotential(3) des Servomotors angeschlossen werden muss.

Tabelle 13: Anschlussbelegung des Servomotors

Port	Stiftleiste	µC Pin	Peripheriesystem
P4.0/TB0	P1.18	17	Servomotor

4.11 Potentiometer

Das Potentiometer PT2 ist ein linear einstellbarer Widerstand mit einem Widerstand von 100 kOhm. Mit diesem Potentiometer soll die Verwendung des Analog-Digitalwandlers vom MSP430 erlernt werden. In einer Applikation können mit dem Potentiometer bestimmte Parameter dynamisch verändert werden. 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 mit anderen Parametern belegt werden. Bei einem festen Aufbau, wäre dies nicht ohne erheblichen Aufwand möglich.

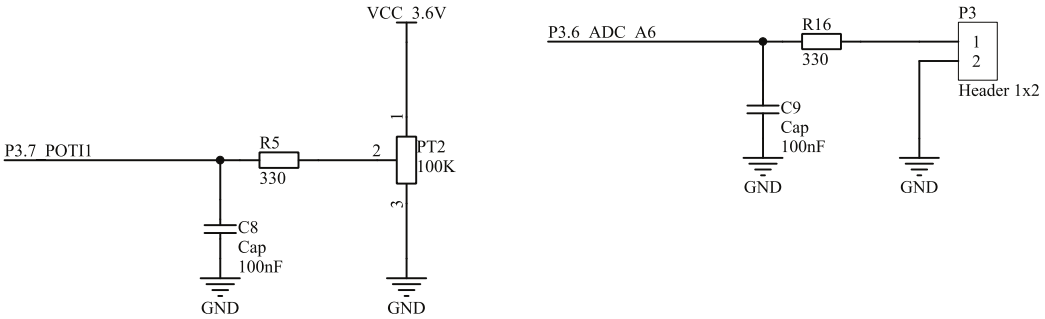


Bild 23: Beschaltung Potentiometer und AD-Eingangsfilter

Der Widerstände R5 und R16 dienen, neben der Eigenschaft als Filter (zusammen mit C8 bzw. C9), auch als Schutz des Mikrocontrollers gegen Kurzschluss.

Tabelle 14: Anschlussbelegung der Potentiometer

Port	Stiftleiste	µC Pin	Peripheriesystem
P3.6/A6	P2.12	27	Analogeingang Stift P3
P3.7/A7	P2.11	28	Potentiometer

4.12 Reset-Beschaltung des MSP430

Der MSP430F2274 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 Reset wird im Controller ein Reset ausgelöst.

5 Softwarebeschreibung

In diesem Kapitel wird die Entwicklungsumgebung IAR Embedded Workbench von IAR Systems [0] kurz beschrieben und erklärt. Die Software kann als Kickstart Version von der Texas Instruments Homepage [0] oder der IAR Systems Homepage [0] 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. Alle Beschreibungen beziehen sich z.Z. auf die IAR Version 4.41 und gelten auch weitgehend für die Version 6.10. Bei Konkreten Änderungen wird gezielt darauf eingegangen.

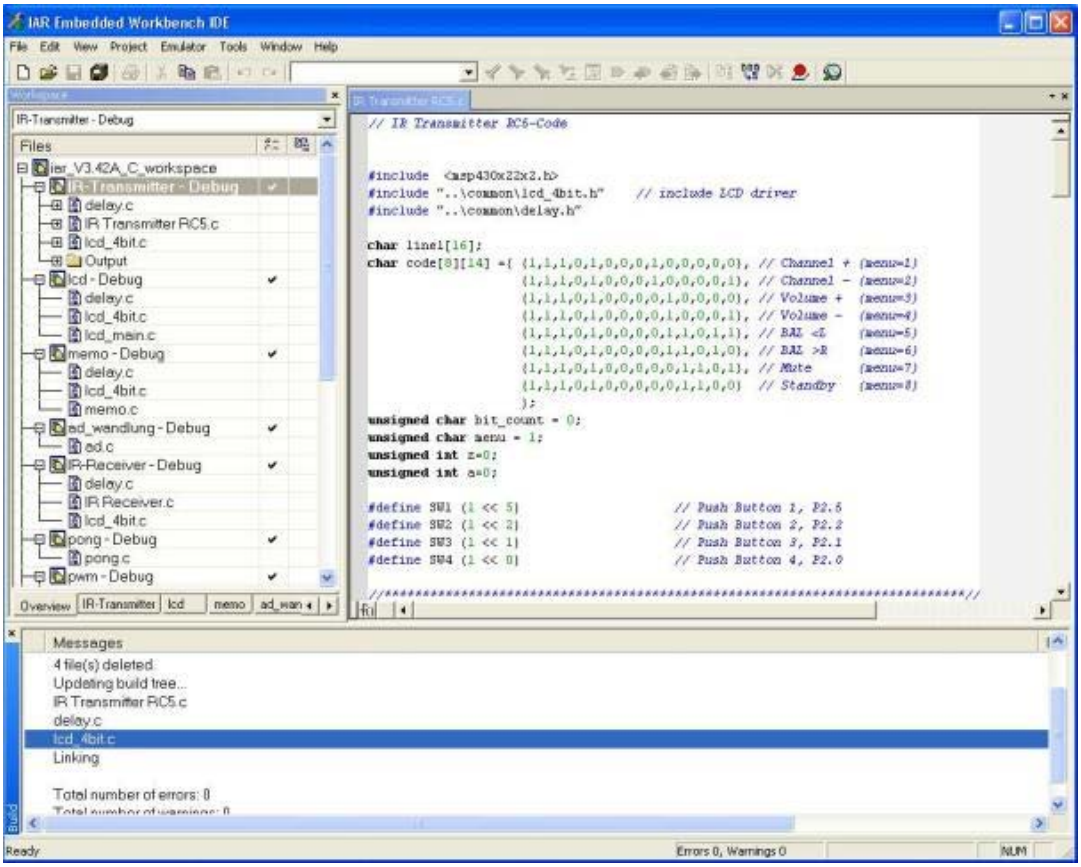


Bild 24: Entwicklungsumgebung IAR Embedded Workbench

5.1 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.

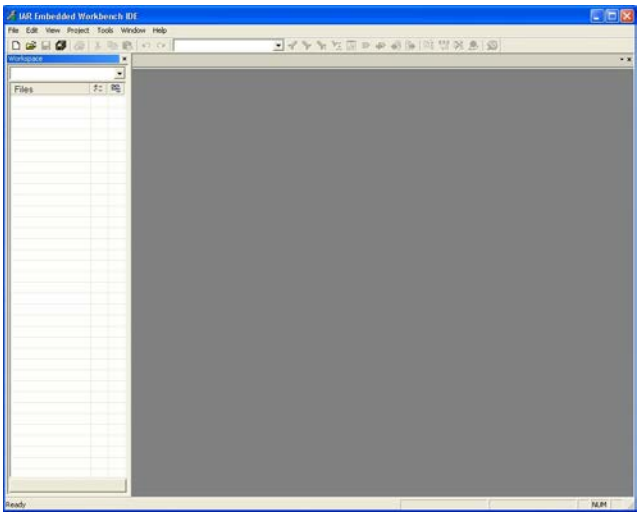


Bild 5-22: Bildschirm nach Start der Software

Nach dem Start der Software durch klicken auf das IAR Embedded Workbench Symbols im Startmenü zeigt sich dem Benutzer folgender Bildschirm wie in Bild 5-22 zu sehen.

Die Programmieroberfläche besteht aus 3 großen Fenstern wie in das Bild 5-23 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 Vereinbarungen zur Verfügung stellen.

Bild 5-23: 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 [0] und [0] 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 5-24. Alle Funktionen der Buttons sind ebenfalls im zugehörigen Handbuch nachzulesen.

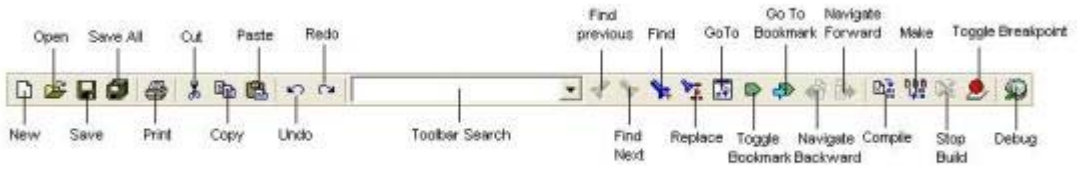


Bild 5-24: Funktionen der Toolbar Buttons

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 5-25 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 [0] und [0].

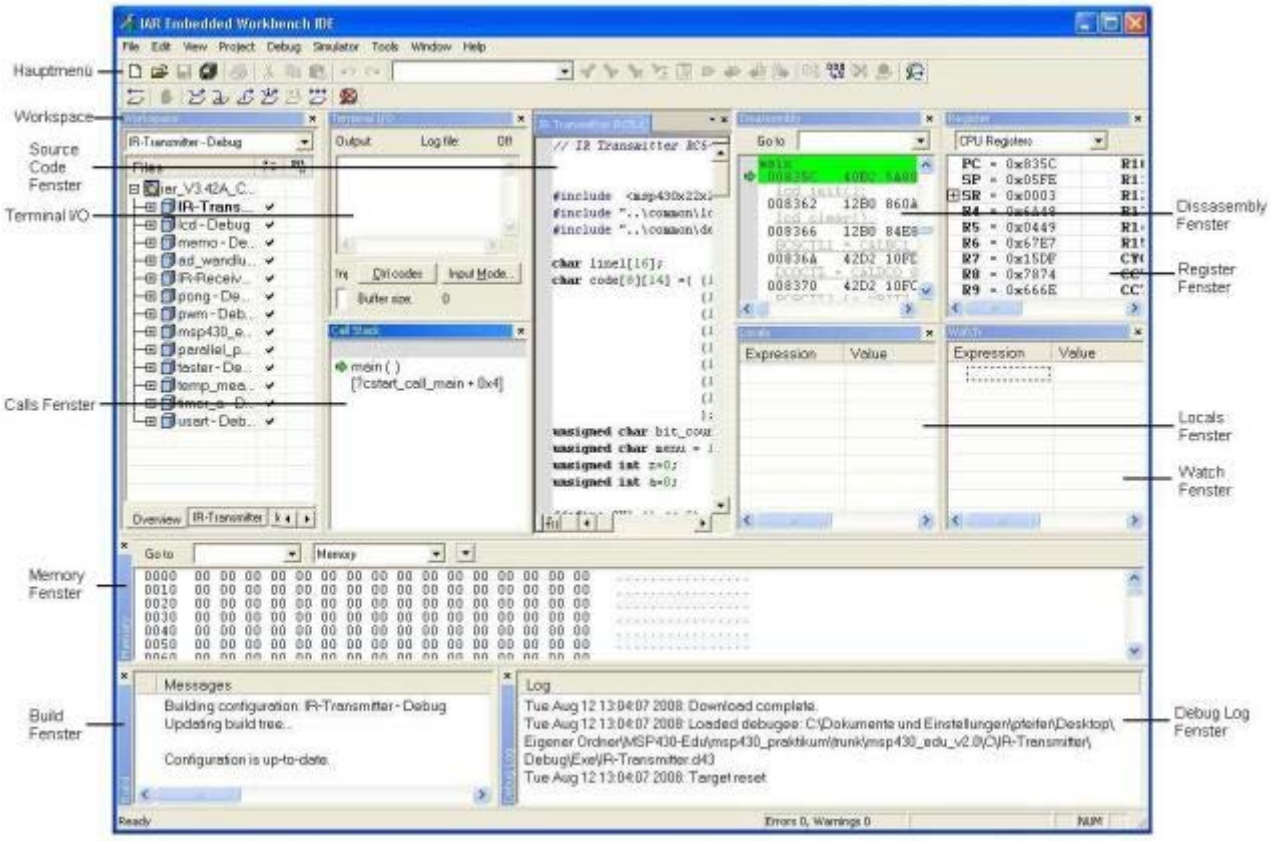


Bild 5-25: Hauptbestandteile des Debuggers

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 lässt 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 lässt 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 5-26 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.

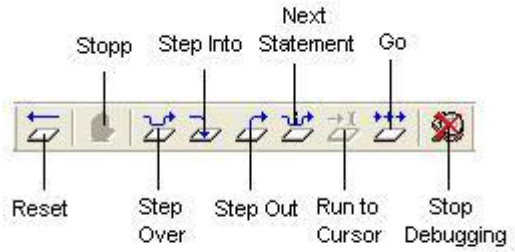


Bild 5-26: Funktionen der Toolbars des Debuggers

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 Fehler! Verweisquelle konnte nicht gefunden werden. 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.

5.2 Erste Schritte

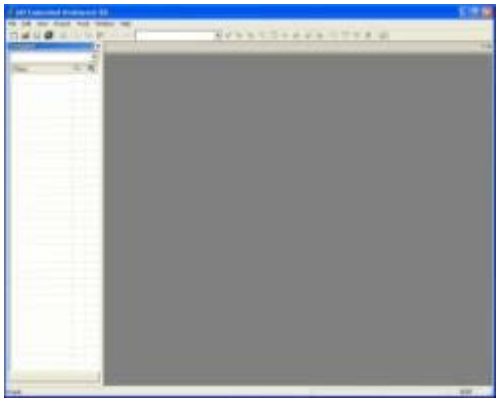
Nachdem in den vorherigen Kapiteln die Hard- und Software genau beschrieben wurde, steht in diesem Kapitel die Anwendung beider Bereiche im Vordergrund. Nachfolgend werden der richtige Aufbau und die richtige Verwendung des MSP430 Education System genau beschrieben. Die Kenntnisse der vorherigen Kapitel 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.

5.3 Laden eines Beispielprojektes

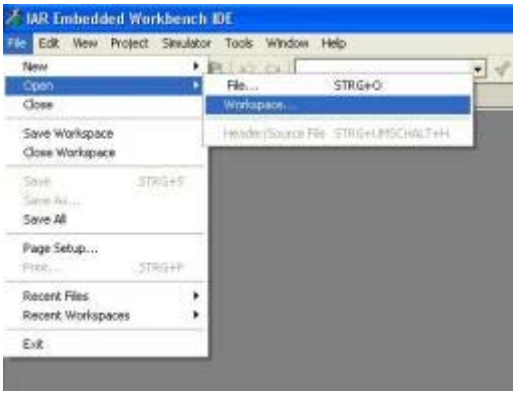
Um die Softwareinstallation nun zu testen, kann ein Beispielprojekt aus dem IAR Programmordner geladen, bearbeitet ausgeführt und werden. Eine Fülle von Beispielprogrammen wird während der Installation in den IAR Programmordner installiert. Auf der Homepage zum MSP Education System stehen weiter Beispielprogramme 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 16 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 16: Laden eines vorhanden Beispielprojektes

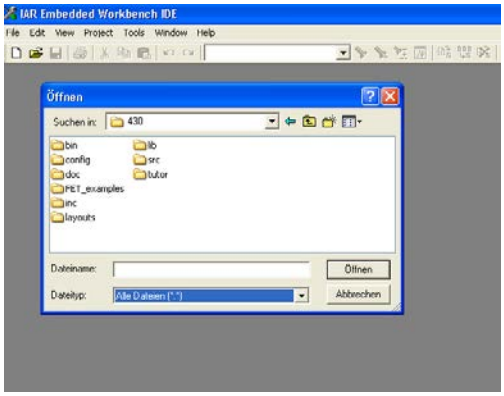
1. Man startet die **IAR Embedded Workbench** aus dem Startmenü heraus.



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



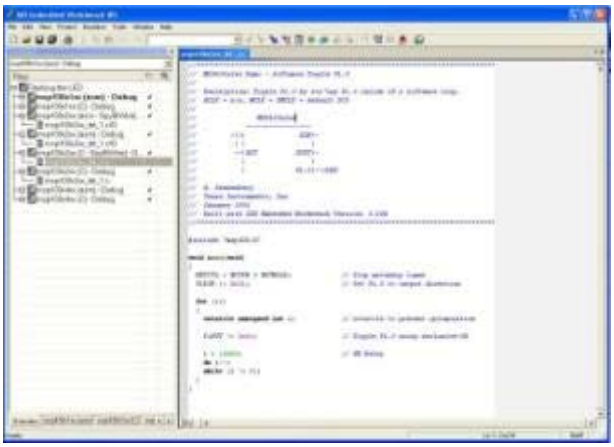
3. Im Menü wählt man **File** um dann im Untermenü **Open** einen neuen **Workspace** zu öffnen.



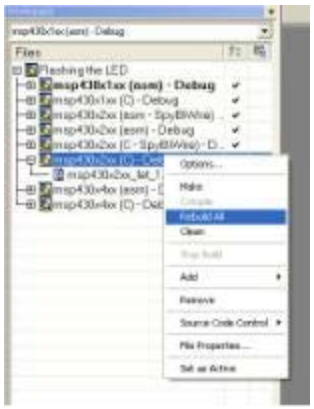
4. Es erscheint folgendes Fenster. Man wählt den **IAR Systems** Ordner im Windows Programme Ordner. Voraussetzung ist ebenfalls, dass die Installation wie in Kapitel 4.1 beschrieben durchgeführt wurde. Man klickt doppelt auf den **IAR Systems** Programmordner.

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**.

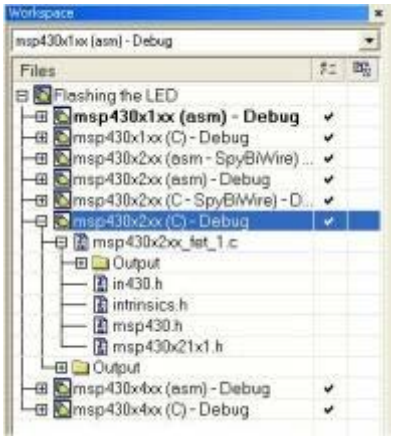
6. In diesem Ordner findet man den Workspace unter dem Namen „**fet_projects.eww**“. Dieses File beinhaltet alle Beispielprogramme, die in C und in Assembler geschrieben sind.



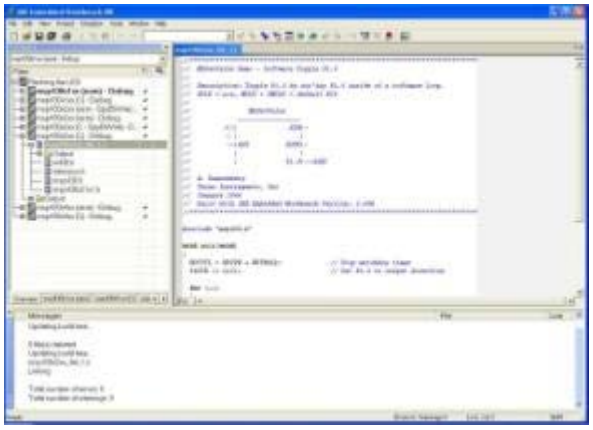
7. In diesem Workspace findet man mehrere Projekte mit dem Namen **msp430x**. Diese Projekte sind für verschiedene MSP430 Typen vorgesehen. Da hier ein MSP430F2272 zum Einsatz kommt, wählt man ein Projekt unter dem Namen **msp430x2xx.** Diese Beispielprogramme können beim Entwurf einer eigenen Applikation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



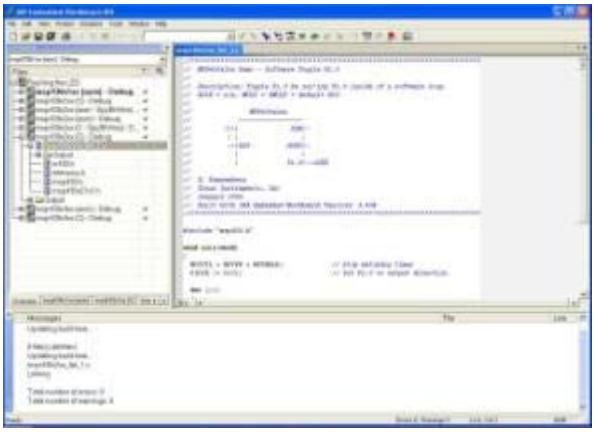
9. Bei dem Compilieren des Projekts werden die Bibliotheken in das Projekt einbezogen und in dem Fenster angezeigt



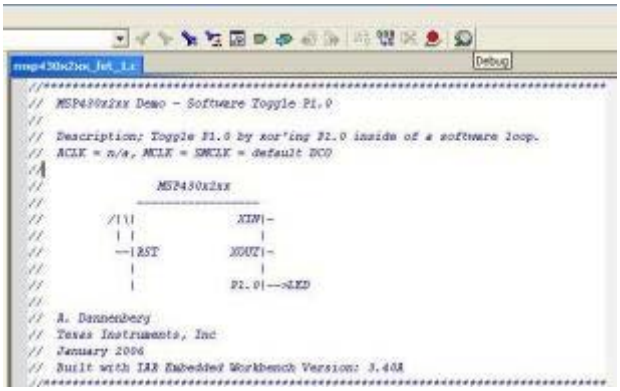
10. Durch einen Doppelklick auf die File **msp430x2xx_fet_1.c**, öffnet sich das Programmierfenster mit dem C-Code. Nach erfolgreicher Durchführung dieser Schritte zeigt sich dem Anwender dieses Bild.

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 17. Es werden alle wichtigen Schritte übersichtlich beschrieben und erklärt.

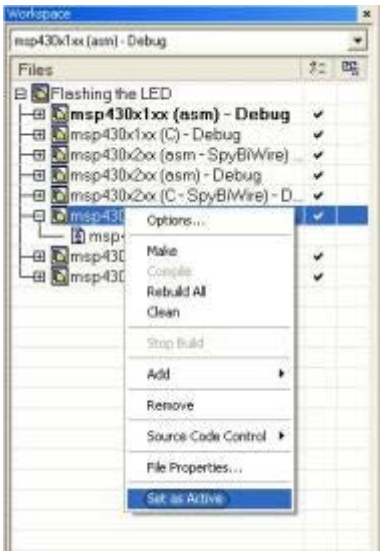
Tabelle 17: Programmieren des Mikrocontrollers



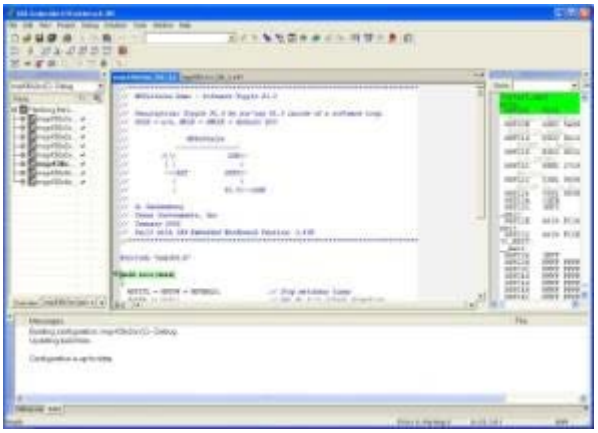
1. Nach dem Laden des Beispielprogramms zeigt sich dem Anwender folgender bekannter Bildschirm. Hat man eine Verbindung zum MSP430 Education System aufgebaut, wie in Kapitel 5.1 beschrieben, kann man nun auf den **Debug**-Button der Toolbar klicken und der Debugger wird geladen.



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 5-27) 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.**

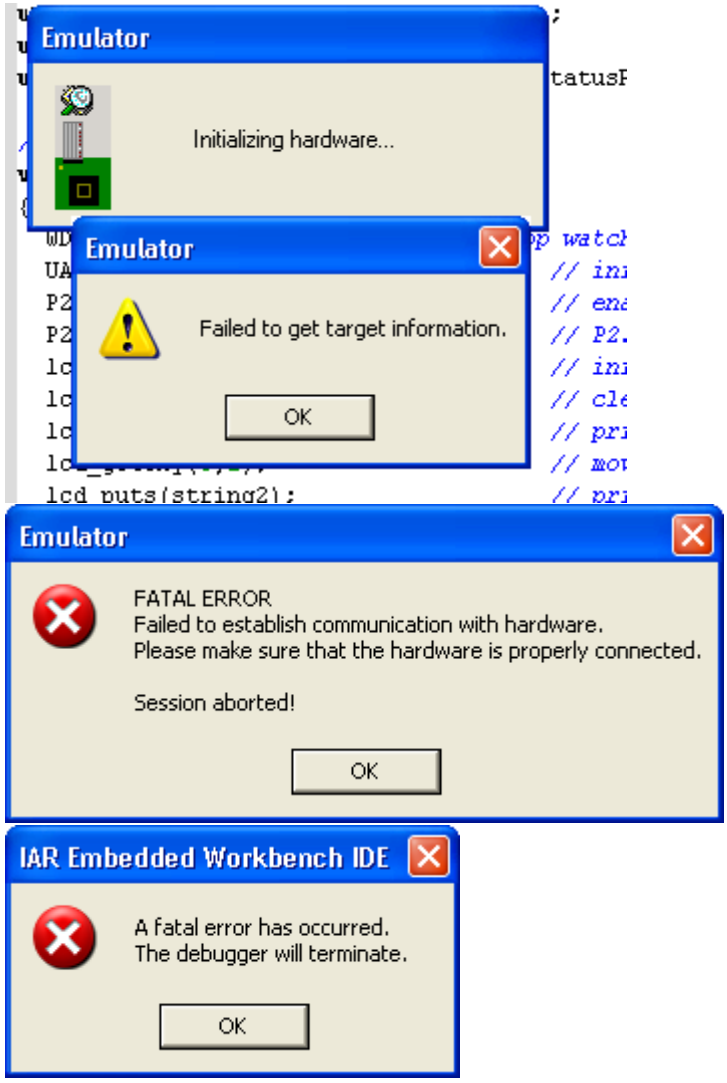


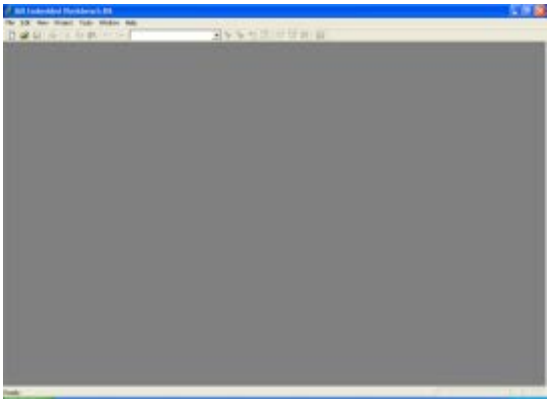
Bild 5-27: Fehlermeldung bei nicht erfolgreichem Download

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

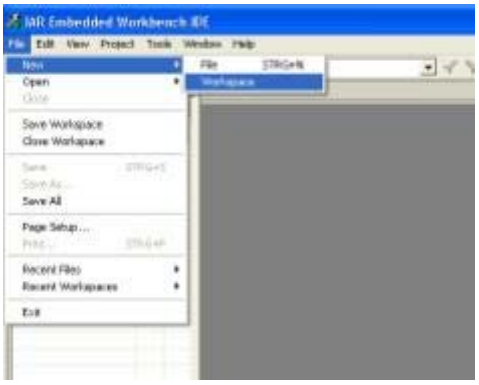
Um nun ein eigenes Projekt zu erstellen, sind wichtige Abläufe und Einstellungen zu beachten. Die genaue Vorgehensweise beim Erstellen und Konfigurieren eines eigenen Projektes zeigt die Tabelle 18. In ihr werden auch wichtige Projekteinstellungen an Hand von Bildern beschrieben. Hierdurch gelingt es 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. 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.

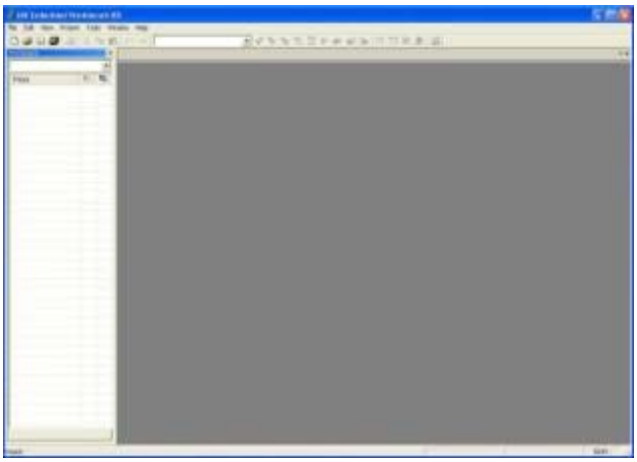
Tabelle 18: Das erste eigene Projekt



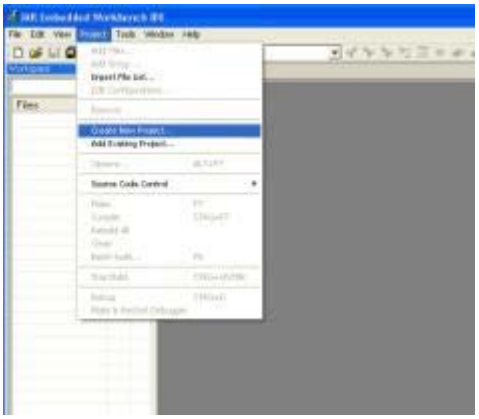
1. Man startet wie gewohnt die IAR Embedded Workbench. Nach dem Laden sieht man dieses Fenster.



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



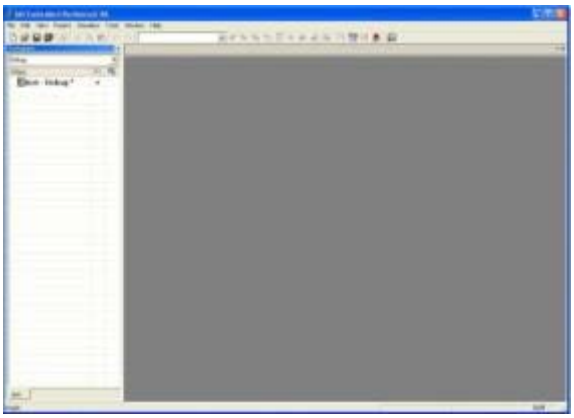
3. Dem Anwender zeigt sich nun dieses Bild. Das Workspace-Fenster ist geöffnet. und zeigt somit das erfolgreiche Anlegen einer neuen Arbeitsfläche.



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

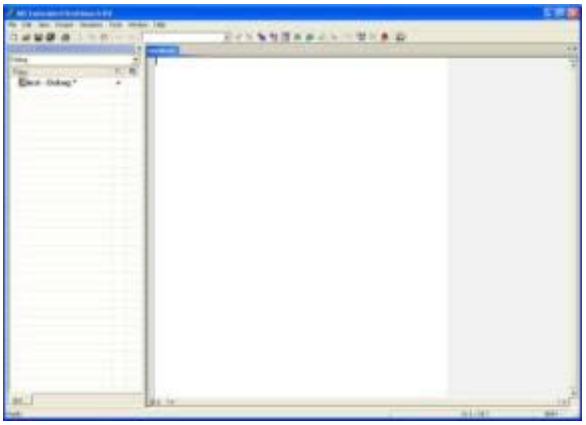
5. Dem Anwender zeigt sich nun dieses Bild. Man kann zwischen verschiedenen Project templates wählen. Für dieses erste eigene Project wählt man nun ein **Empty Project**.

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.

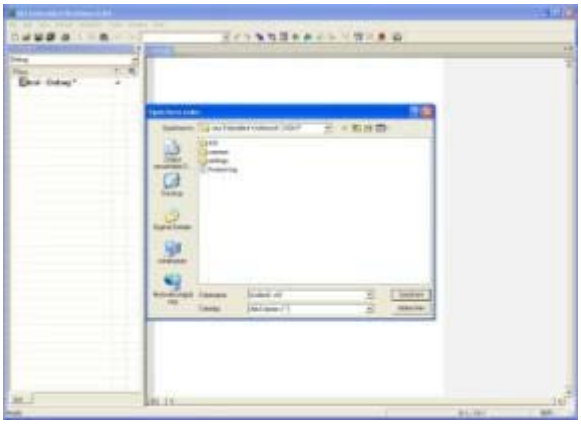


7. Dem Anwender zeigt sich nun dieses Bild. Das Projekt-Fenster ist geöffnet und zeigt somit das erfolgreiche Anlegen eines neuen Projektes. Um nun ein Programm schreiben zu können muss man noch ein neues Sourcefile anlegen.

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

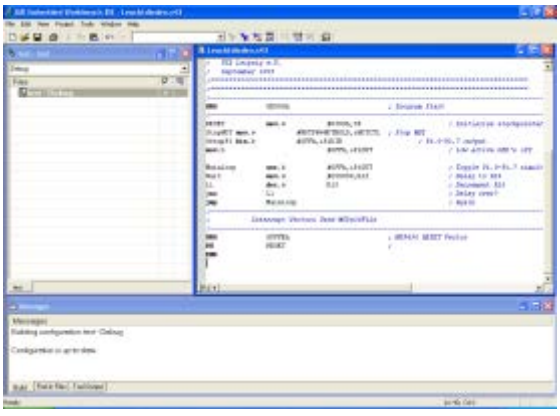


9. Es öffnet sich ein neues Fenster mit dem Namen **Untitle1**. Um das neue Sourcefile zu speichern, drückt man nun auf den **Speichern** Button.

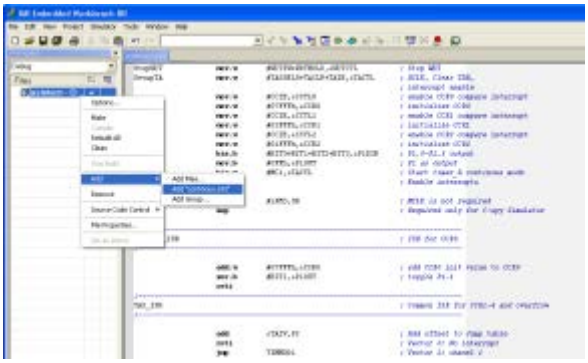


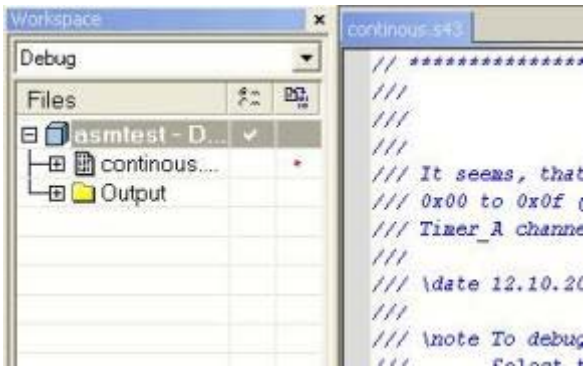
10. Da hier ein Assemblerprogramm entwickelt werden soll, wählt man als Namenserweiterung „**.s43**“. Dies ist die Bezeichnung für eine IAR EW Assemblerfile. Die Datei nennt man nun **xxx.s43** wobei xxx für einen selbst gewählten Namen steht. Danach drückt man auf **Speichern**.

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.

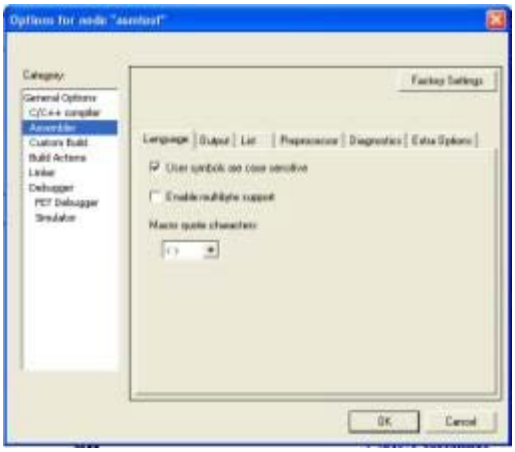


12. Dieser Sourcecode zählt binär von 0x00 zu 0x0f und in Abhängigkeit davon leuchten die LED's an P1.0 bis P1.3. Um das Projekt compilieren und ausführen zu können, müssen sehr viele Grundeinstellungen vorgenommen werden. Alle Einstellungen werden in den folgenden Fenstern genau beschrieben.



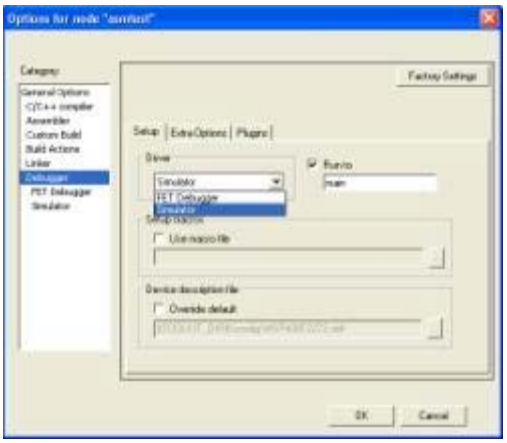


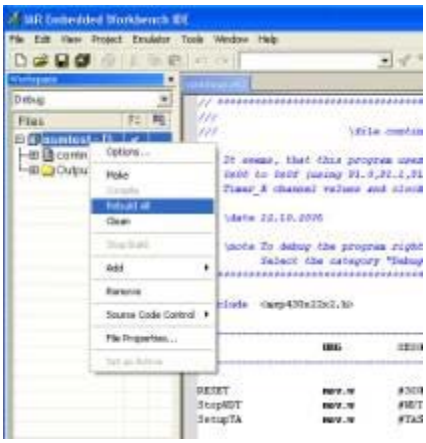
13. Die Quelldatei muss nun in das Projekt eingefügt werden. Dazu macht man einen Rechtsklick auf das Projekt und wählt im Kontextmenü **Add** und wählt zwischen **Add Files ..** und **Add xxx.s43** . Danach ist das Sourcefile in das Projekt eingebunden.



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**.

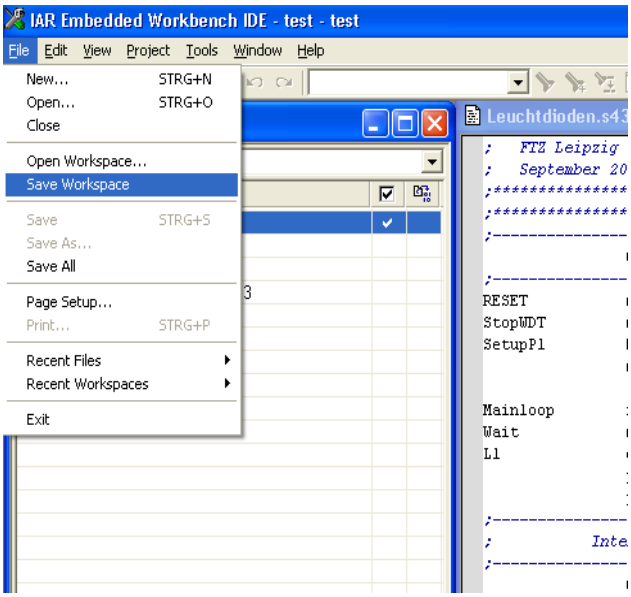
15. Damit das Projekt später ohne Fehler compilieren kann, muss man ein Häkchen in bei **Assembler-only project** machen.



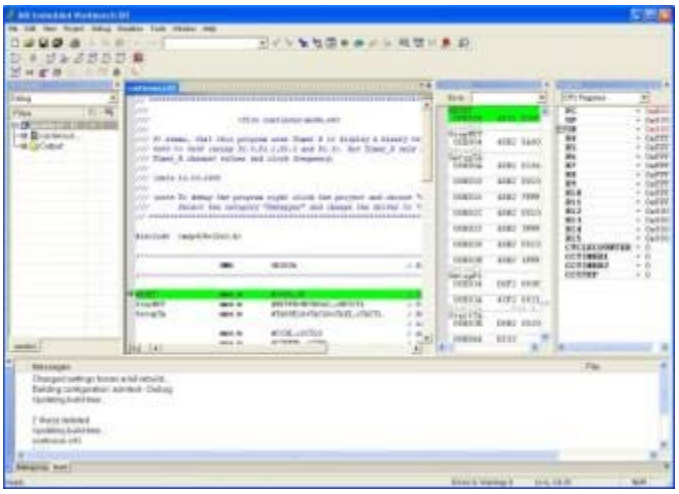


16. Nun wählt man den Unterpunkt **Debugger**. Im Setup Menü des Debugger Fensters kann man unter dem Punkt **Driver** zwischen 2 Debugmodi wählen. Standardmäßig ist hier der **Simulator** eingetragen. Um Programme auf den Mikrocontroller laden zu können, wählt man den Unterpunkt **FET Debugger**.

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  <msp430.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
```

A. Befehle des MSP430F2274

Die folgende Tabelle 20 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 [0] 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 20: 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	*	*	*	*

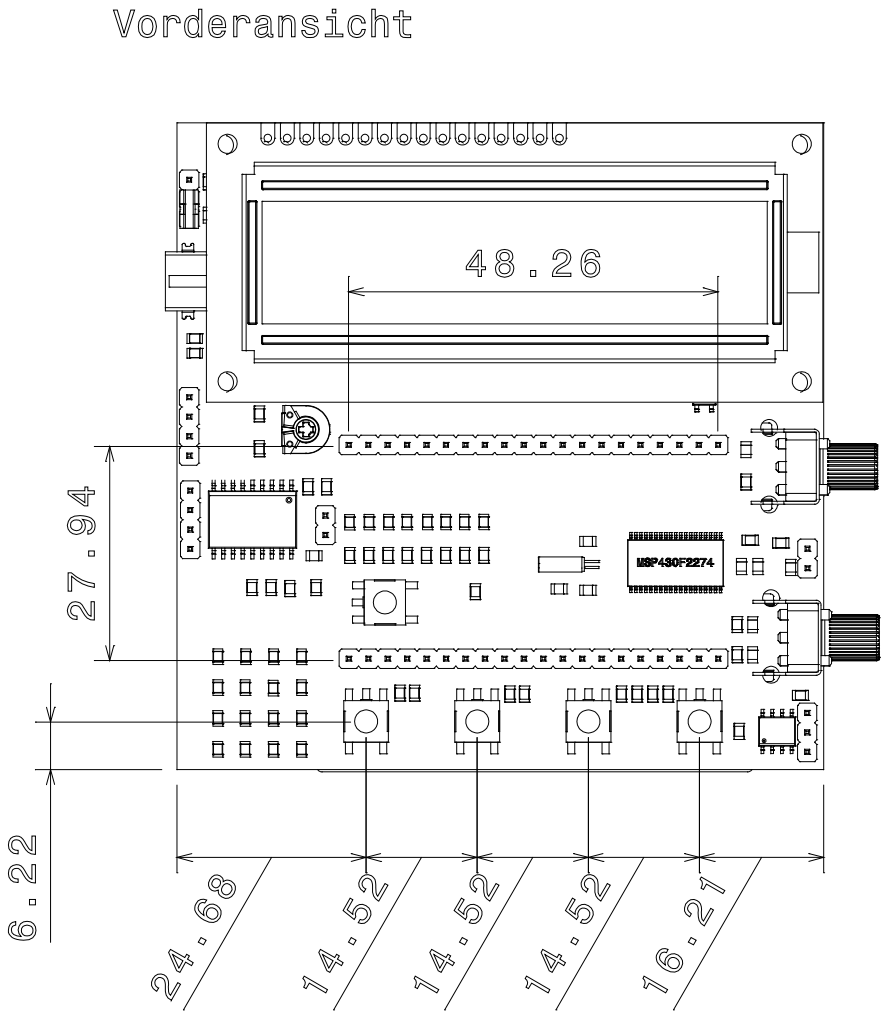
***** **-** Emulierte Befehle

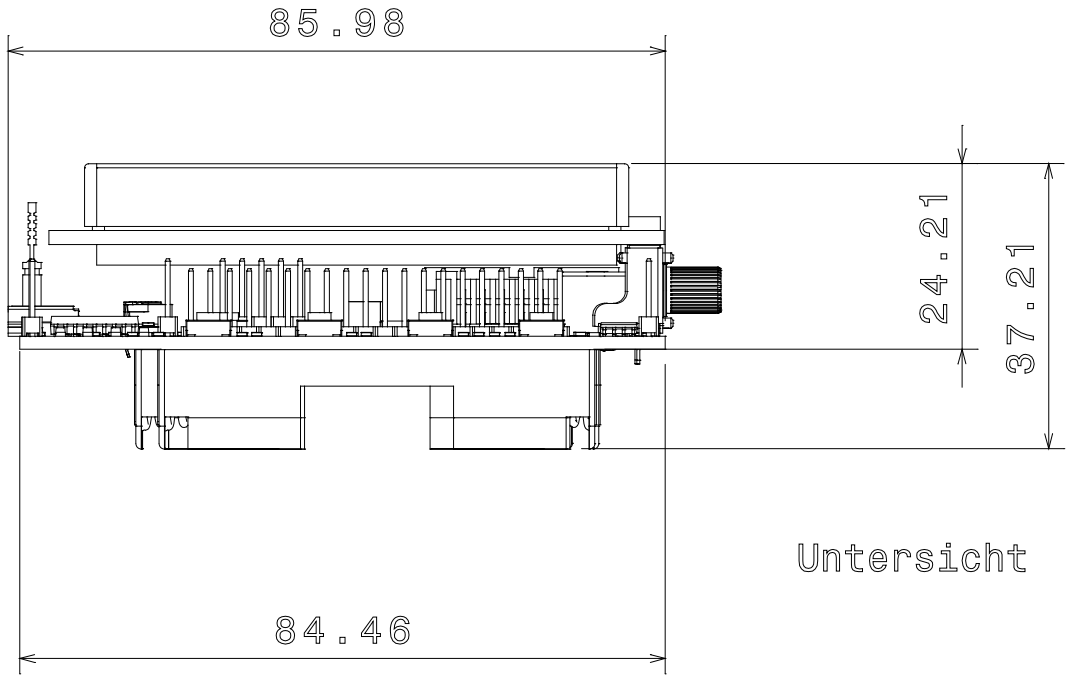
src - Abkürzung für Source, Quelle

dst - Abkürzung für Destination, Ziel

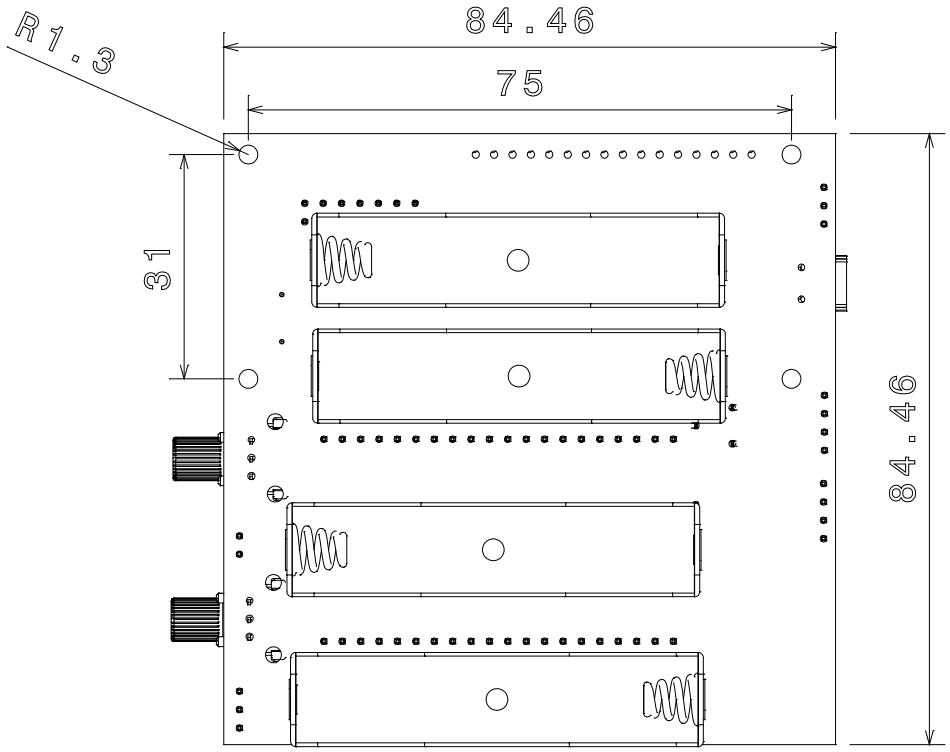
label - Bezeichnung für die eingesetzte Sprungmarke,

B. Bemaßung


Vorderansicht
48.26
27.94
6.22
24.68
14.52
14.52
14.52
16.21
MSP430F2274



Rückansicht



C. Schaltpläne

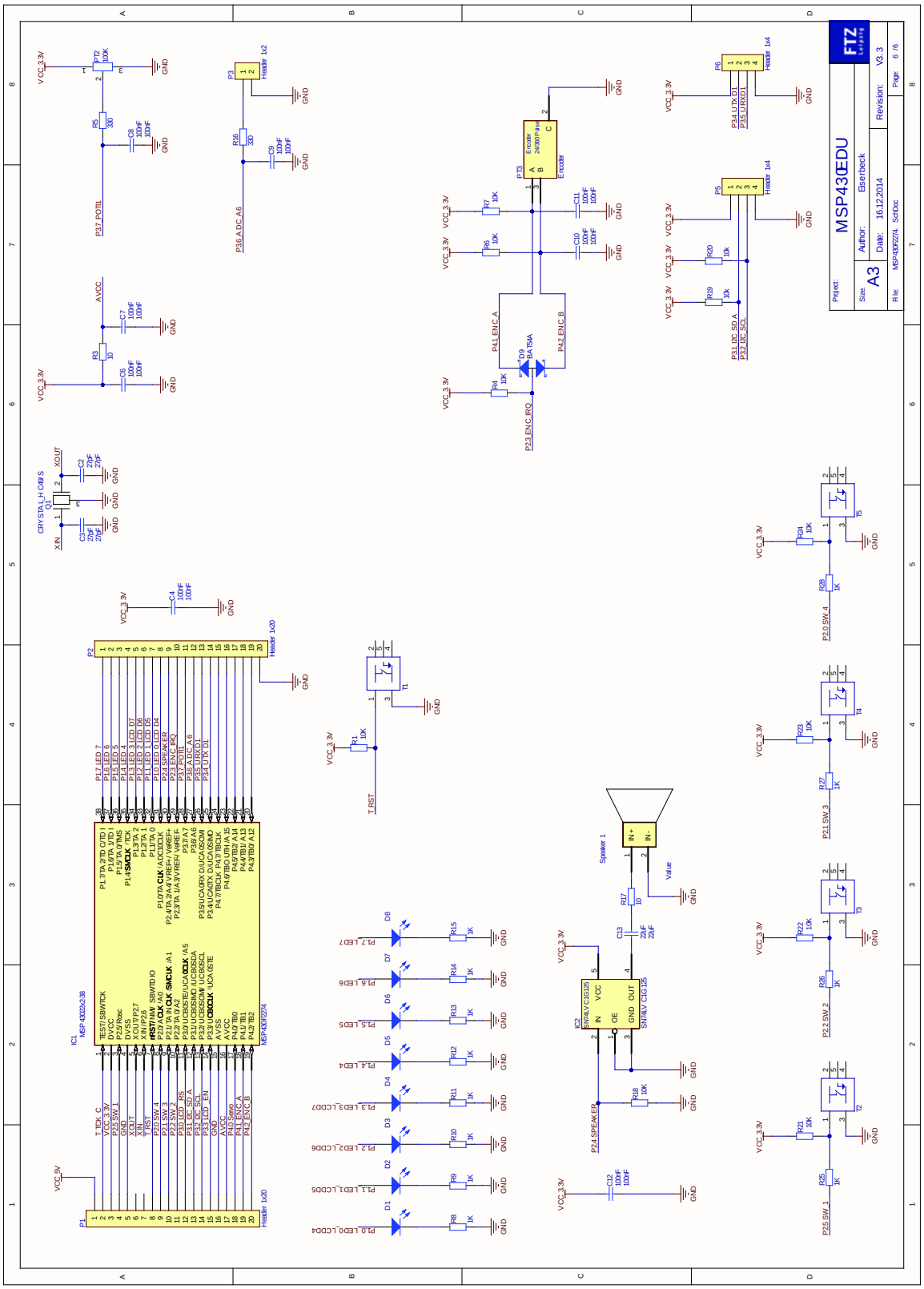


Bild 25: Schaltplan MSP430F2274

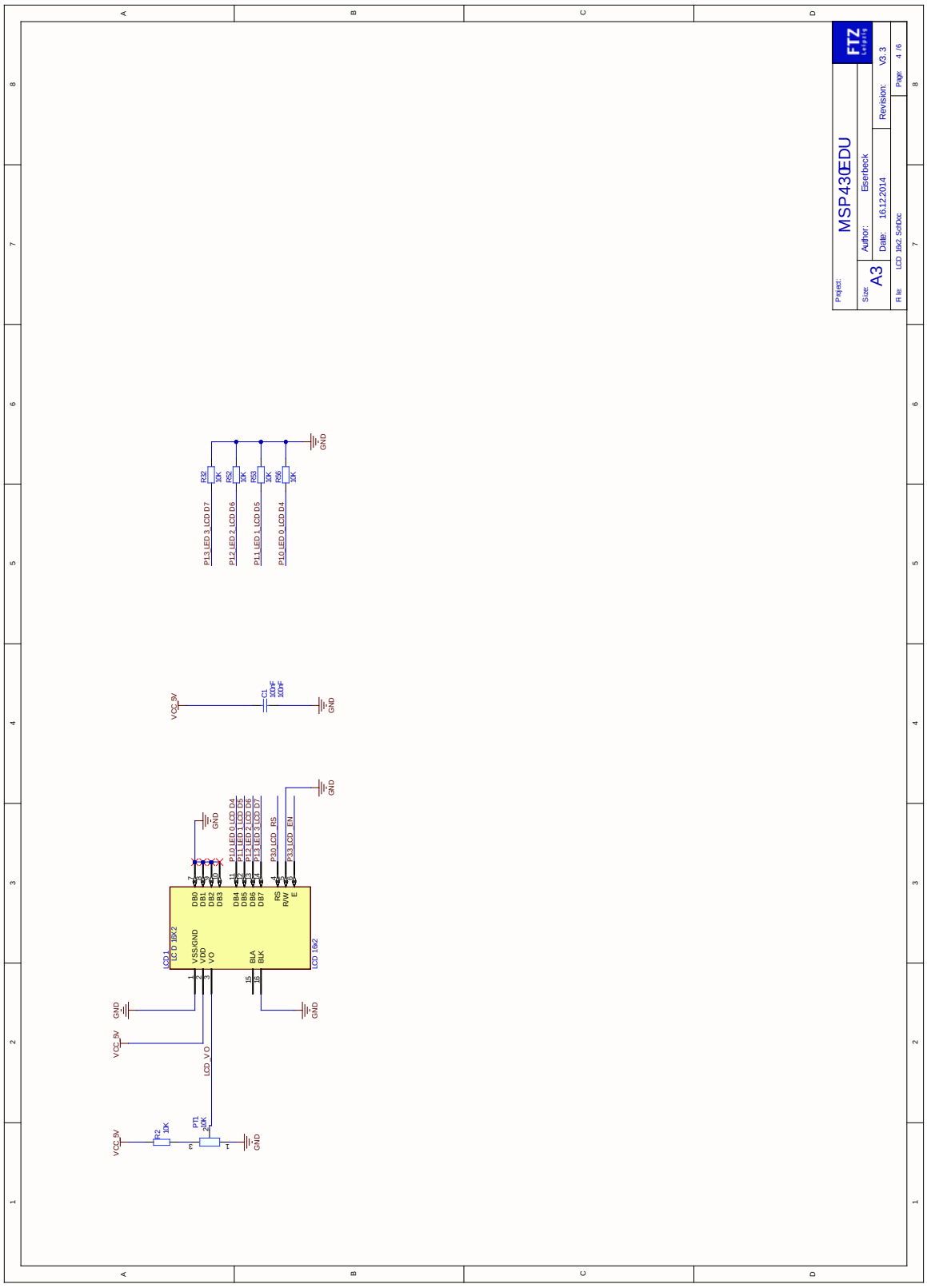


Bild 26: Schaltplan LCD Display

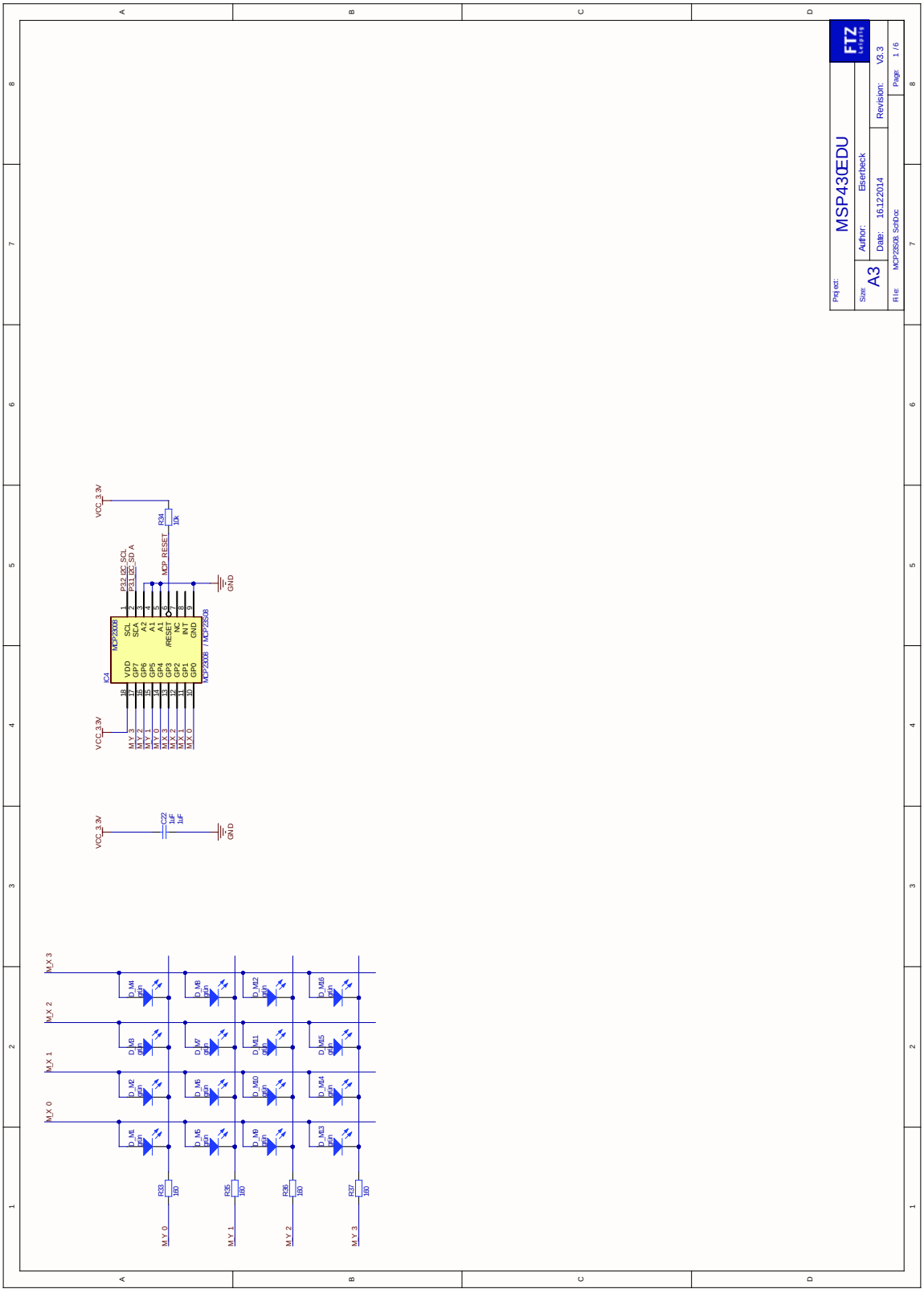


Bild 27: Schaltplan LED Matrix

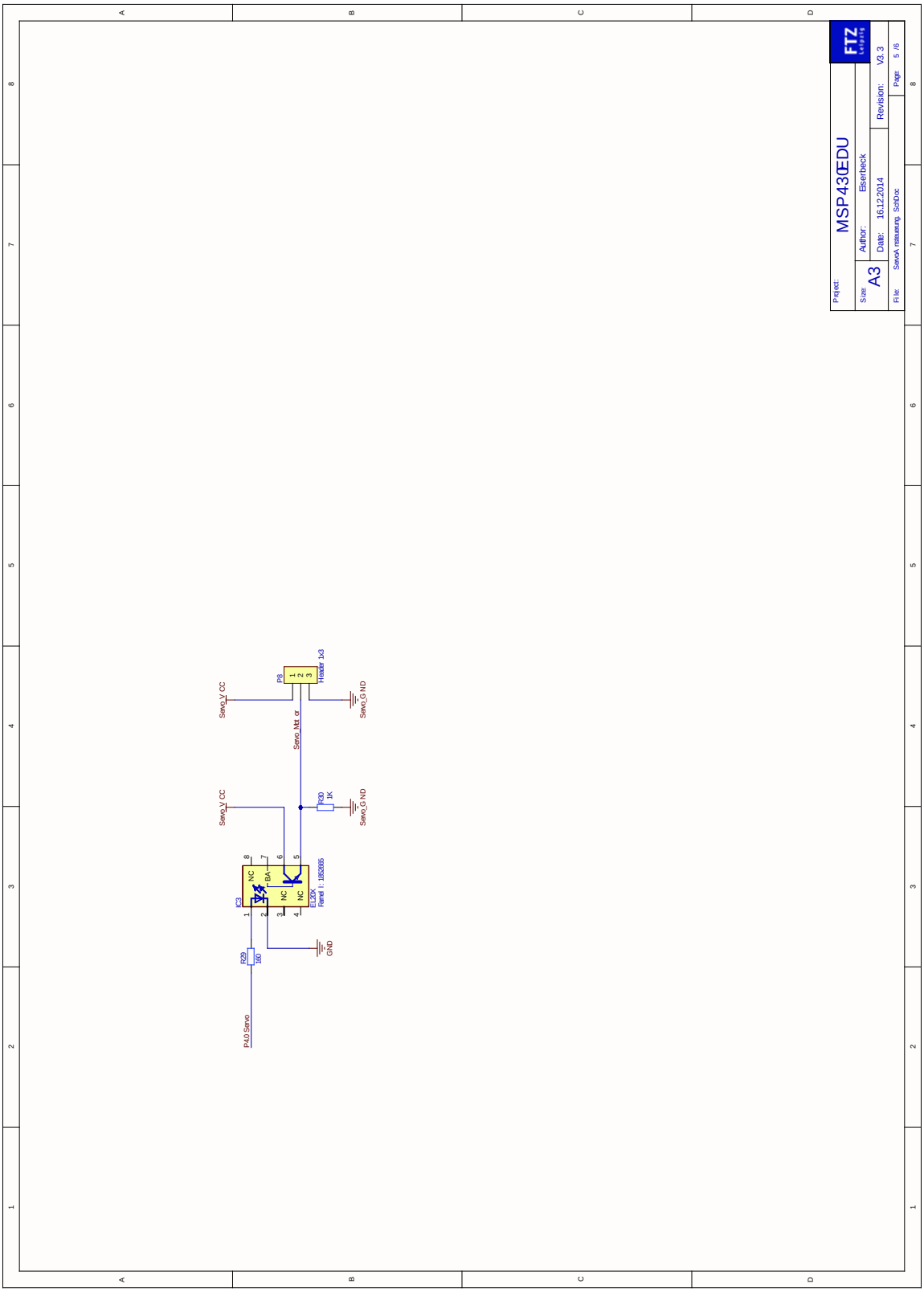


Bild 28: Schaltplan Optokoppler / Servo

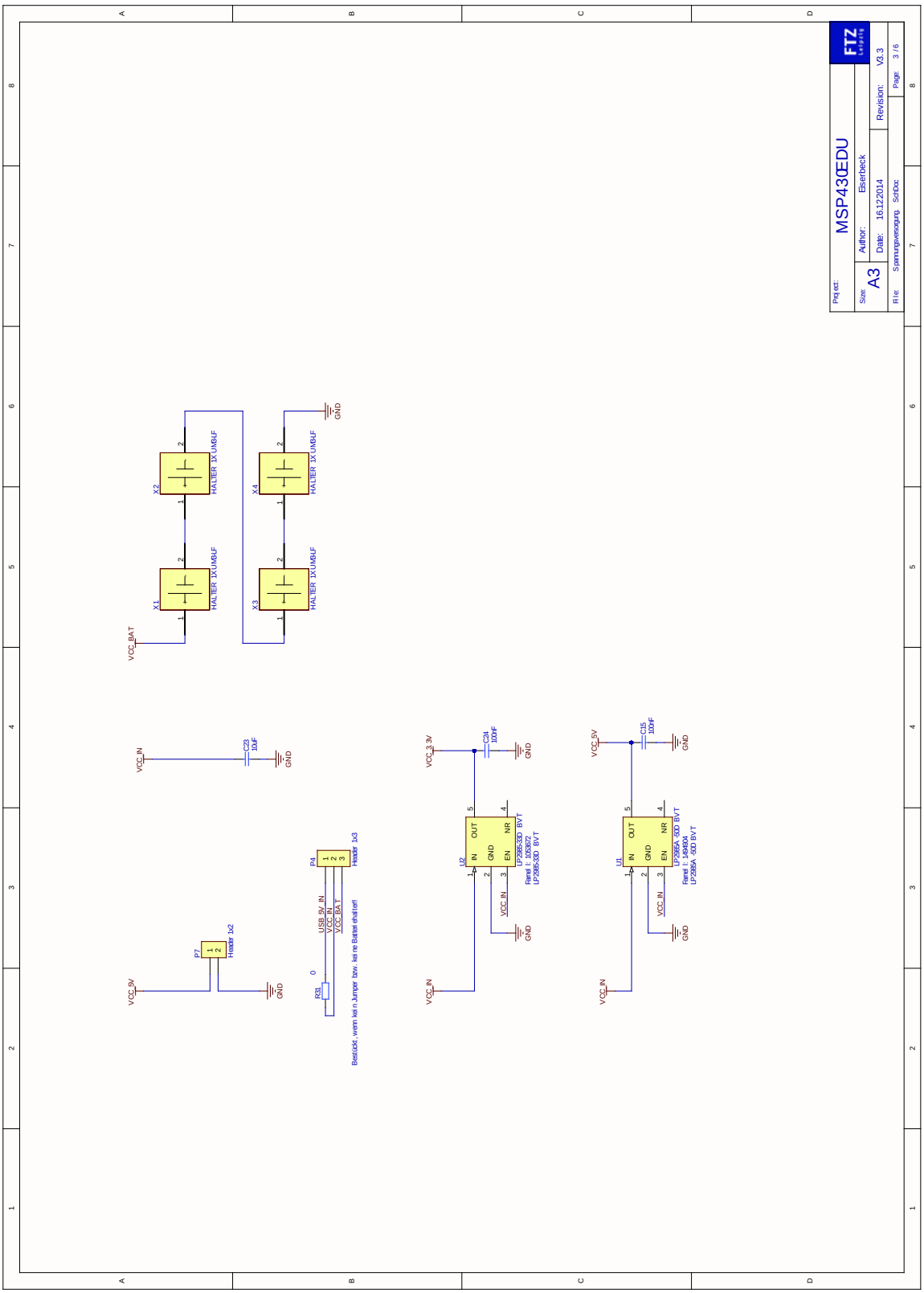


Bild 29: Schaltplan Spannungsversorgung

A. Literatur- und Quellenverzeichnis

[0] Texas Instruments Homepage
www.ti.com

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

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

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

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

[0] msp430f2272, Texas Instruments MSP4302272 Datenblatt

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

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

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

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

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

[0] icc430.pdf, MSP430 C COMPILER Programming Guide

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

[0] TI Simulator Users Guide.pdf, IAR CSPY w/TI Simulator Users Guide

[0] Tutor.pdf, IAR CSPY w/TI Simulator Users Guide

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

[0] Datenblatt HD44780U, Hitachi Semiconductor