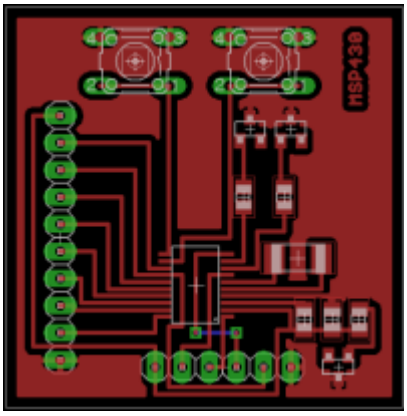
MSP430dev



Für die vielen herumliegenden MSP430F1111A und MSP430F1101A wurde ein einfaches Entwicklungskit gebaut. Einsatzmöglichkeit sind kleine Projekte ohne viel Peripherie (Nur Timer und Komparator sind zusätzlich vorhanden) und das Lernen von Assembler.

- **Version 1:** erste Auflage.
- **Version 2:** TODO
  - Pullup an RESET und Pulldown an TEST vorsehen,
  - Inverter an TEST vorsehen
  - Pullups an den Tastern vorsehen

Hardware

Es handelt sich um den MSP430F1101A oder MSP430F1111A. Die 8 GPIOs von Port 1 sind herausgeführt auf einer 10poligen Stiftleiste (zusammen mit 3,3V und GND), auf die die üblichen Steckmodule passen. Zwei Taster und zwei LEDs sind an GPIOs von Port 2 angebracht. Zur Temperaturstabilisierung des internen Oszillators ist ein 100k-Widerstand an P2.5 vorhanden. P1.1 und P2.2 sowie RESET und TEST werden für die Programmierung per BSL verwendet, können aber im normalen Programmablauf auch verwendet werden.

Datenblatt: http://www.ti.com/lit/ds/symlink/msp430c1111.pdf

Family Guide: http://www.ti.com/lit/ug/slau049f/slau049f.pdf

Example Code: http://www.ti.com/lit/zip/slac013

Programmierung

Die Programmierung kann einfach über eine serielle Schnittstelle (mit TTL-Pegeln) geschehen. Mit der IDE seiner Wahl wird eine HEX(A43)-Datei erzeugt, welche dann mit MSPFET oder mspdebug auf den Controller geladen wird. Der verwendete BootstrapLoader (BSL) ist hier dokumentiert.

Folgende Einstellungen sind dabei zu wählen:

- Tools → Setup
  - „Current Adapter Settings“ auf BSL
  - COM-Port korrekt einstellen
  - Keep Port Open auf True
  - RST invert auf „True“, TST invert auf False
  - links bei Autoprogram
    - Reload File ein
    - Erase ein
    - Blank Check aus
    - Program ein
    - Verify ein
- Device: MSP430F11x1A
- Dann File → Open, Hexfile wählen → Auto

Zu beachten: Nur bei „Auto“ (also automatischem Programmiervorgang) wird die Programmierdatei neu geladen, wenn man manuell programmiert, muss man sie manuell neu laden (File → Open).

Die Belegung des Programmierheaders (von MSP-Seite/von Seriell-Seite) ist:

- GND / GND
- TXD / RXD
- RXD / TXD
- TEST / RTS
- RESET / DTR
- Vcc / 3V3

und damit passend zum TUSB3410 USB-Seriell-Wandler. An dem 10Pin-Port ist Port 1 komplett herausgeführt, sowie 3,3V und GND. Treiber für den TUSB3410 für Windows sind hier zu finden: http://www.dell.com/support/drivers/us/en/04/driverdetails?driverid=R286478 - **falls die automatische Installation trotzdem fehlschlägt:** „manuelle Treibersuche“ auswählen → „Gerät aus einer Liste wählen“ → „Schnittstellen“ → „Texas Instruments“ → „TUSB3410 Device“.

GNU/Linux

Unter Debian wurde msp430-bsl.py aus python-nmsp430-tools getestet.

Der Funktionsaufruf zum flashen lautet:

```
# msp430-bsl.py -e -P -r -p $GERAET $HEXFILE --invert-reset [--swap-reset-test]
```

Makefile

makefile.txt

Schaltplan / Layout

Version 1.0

- Schaltplan: PDF, Eagle SCH
- Layout: msp430dev.pdf ,Eagle BRD
- Layout, gespiegelt und nur-schwarz: PDF

Erste Schritte mit IAR Workbench

- Herunterladen von http://supp.iar.com/Download/SW/?item=EW430-EVAL
- Installieren und starten
- Project → Create New Project → Als Vorlage → „asm“ → „asm“ wählen
- Ordner anlegen, die angefragte EWP-(Projekt-)-Datei mit Namen „Projekt“ speichern
- Project → Options
  - Device „MSP430F1111A“
  - Linker → Output → „Other“ → Output Format: intel-standard
- Mit F7 Assemblieren, den Workspace unter „Workspace“ speichern (EWW-Datei)
- Mit MSPFET die entstehende a43-Datei (Im Ordner „bin“) öffnen und programmieren

Der Sourcecode der ersten Programmierversuche ist hier zu finden: erste-schritte.txt