Das Stone-Board, ein einfaches Mikrocontroller-Board

Dokumentation

Matthias Stonebrink, MSC

V1.2

Inhaltsverzeichnis

Einleitung

Das sogenannte Stone-Board ist ein einfaches Mikrocontroller-Board, das den Einstieg in die Embedded Systeme ermöglichen soll.

Das Board baut auf einem **Atmega88**-Mikrocontroller der Firma Atmel auf. Der Atmega88-Mikrocontroller kann aber auch gegen einen der pin-kompatiblen Typen **Atmega168** oder **Atmega328** getauscht werden, falls mehr Speicherplatz benötigt wird. Für die Programmierung und die Nutzung steht kostenlose Software zur Verfügung.

Für die Übertragung des Programms kann entweder ein ISP-Programmer (ISP = In System Programming) verwendet werden oder das Programm kann bequem über die USB-Schnittstelle übertragen werden, sofern sich ein Bootloader auf dem Chip befindet. Somit eignet sich das System auch gut für Einsteiger, da nur die Kosten des Boards zu tragen sind und keine zusätzlichen Ausgaben für Programmer und kostenpflichtige Software anfallen.

Um den Einstig so einfach wie möglich zu gestalten, wurden Bibliotheken geschrieben, die einen einfacheren Zugriff auf die I/Os und Schnittstellen zu ermöglicht. Der Mikrocontroller lässt sich selbstverständlich auch unabhängig von den Bibliotheken mit C, Assembler oder Basic (Bascom) programmieren.

Hardware

Das Stone-Board besitzt neben dem Mikrocontroller zusätzliche Baugruppen, um es vielfältig einsetzten zu können. Dies sind u. a. ein FT232 USB-UART-Converter, LEDs und Taster.

Der Controller

Das Board wurde für die Mikrocontroller-Familie AtmegaX8 entwickelt. Es handelt sich um einfach aufgebaute 8-Bit-RISC-(**R**educed **I**nstruction **S**et **C**omputer) Mikrocontroller, die über begrenzte Ressourcen und Peripherie verfügen. Im Datenblatt der Controller finden sich alle Details, weshalb hier nur auf einige Features eingegangen wird.

I/O Pins

Es stehen 23 I/O Pins zur Verfügung die beliebig konfiguriert werden können. Per Register oder Bibliothek können diese als Eingang oder Ausgang geschaltet werden. Ist der Pin als Eingang geschaltet, gibt es auch noch die Option, einen internen Pullup-Widerstand zu aktivieren, um auch bei offenem Eingang einen definierten Pegel zu erhalten.

Bei den Ausgängen handelt es sich um Push-Pull-Ausgänge d. h. ein Ausgang kann sowohl Lasten niederohmig auf Masse oder Vcc zu ziehen.

Es ist jedoch zu beachten, dass pro Pin ein Maximalstrom von 40 mA nicht überschritten wird, sonst kann es zur Beschädigung der Ausgangsstufe kommen. Alle Pins zusammen dürfen nicht mehr als 200 mA abgeben.

Timer

Es stehen drei Timer zur Verfügung: Timer0 (8-Bit), Timer1 (16-Bit), Timer2 (8-Bit). Diese können vielfältig konfiguriert werden. Einige Funktionen sind in den Bibliothek implementiert wie z. B. „Wait“-Befehle, Servosignal oder PWM. Ein synchroner oder asynchroner Betrieb ist möglich.

Analog-Digital-Converter (ADC)

Mit dem ADC ist die Umsetzung einer analogen Spannung in einen digitalen Wert möglich. Der AtmegaX8 verfügt über sechs 10-Bit-ADC-Kanäle, die unabhängig voneinander ausgewertet werden können. Als Referenzquellen können entweder die interne 1,1-V-Referenz, Vcc oder eine externe Referenz verwendet werden.

Die angelegte Analogspannung sollte nicht über der Referenzspannung und darf keinesfalls über der Betriebsspannung des Mikrocontrollers liegen.

Interfaces

Für die Kommunikation mit externer Peripherie, einem PC oder anderen Mikrocontrollern stehen verschiedene Interfaces zur Verfügung. Diese sind in Hardware realisiert d. h. das Senden und Empfangen geschieht unabhängig von der CPU des Mikrocontrollers. Auch hier wurden Bibliotheken geschrieben, die die Initialisierung und Benutzung der Schnittstellen vereinfacht.

Es stehen UART (**U**niversal **A**synchronous **R**eceiver **T**ransmitter), SPI-Schnittstelle (**S**erial **P**eripheral **I**nterface) , TWI (**T**wo **W**ire **I**nterface) bzw. I²C zur Verfügung.

Interrupts

Der AtmegaX8 verfügt über 26 Interrupt-Quellen. Die externen Interrupts wie Int0, Int1 und alle PCIs (**P**in **C**hange **I**nterrupt) können per Bibliothek aktiviert werden. Verschiedene Einstellungen sind möglich werden.

Stromversorgung

Die Stromversorgung ist zweiteilig realisiert. Zum Programmieren und bei Betrieb von Schaltungen mit nur geringem Stromverbrauch (weniger als 200 mA) kann das Stone-Board über den Mini-USB-Stecker mit 5 V versorgt werden. Zusätzlich verfügt das Stone-Board noch über eine eigenen Spannungsregler. An der entsprechenden Schraubklemme kann eine Spannung von 9 V bis 12 V AC/DC eingespeist werden. Durch die Gleichrichtung mit nachgeschalteter Spannungsregler werden die benötigten 5 V Betriebsspannung bereitgestellt. Der maximale Ausgangsstrom hängt von von der Eingangsspannung und der Kühlung des Spannungsreglers ab.

Über den Jumper JP4 kann zwischen USB und Klemmenversorgungsspannung gewechselt werden.

Wenn der Jumper die Pins 1 und 2 verbindet, wird das Board mit den 5 V USB-Spannung versorgt. Wenn der Jumper die Pins 2 und 3 verbindet wird das Board mit 5 V des Spannungsreglers versorgt.

Sowohl der USB-Eingang als auch der Klemmen-Eingang sind mit Sicherungen versehen. USB mit 200 mA flink (FUSE 1) und der Klemmen-Eingang mit 1.5 A mittelträge (FUSE 2).

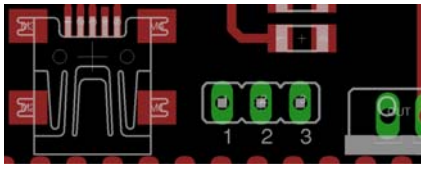


Abbildung 1: Jumper der Spannungsversorgung

FT232 USB-UART-Converter

Um eine möglichst einfache Kommunikation zwischen PC und Stone-Board zu ermöglichen, verfügt das Board über einen FT232-Chip von FTDI. Über ein Terminalprogramm (z. B. Hterm oder PuTTY) kann eine Verbindung mit dem Board aufgebaut werden; der FT232 gibt sich dabei als serielle Schnittstelle (COM-Port) aus. Damit es es möglich, über USB Daten an den PC zu übertragen. Weiterhin erlaubt der COM-Port das Debugging des Programms sowie die Steuerung des Board und ermöglicht viele andere hilfreiche Optionen.

Wenn auf dem Stone-Board ein Bootloader installiert ist, kann das Programm auch über die USB-Schnittstelle übertragen werden.

Eine erfolgreiche Datenübertragung kann anhand der LEDs festgestellt werden, die auf dem Board zwischen dem FTDI-Chip und der Mini-USB-Buchse angeordnet sind. Pro Datenpaket blinkt die RX- bzw. TX-LED einmal auf.

Quarz

Der AtmegaX8 läuft auf Werkseinstellung mit vier MHz (interner RC-Oszillator). Für genauere Taktfrequenzen sollte ein externer Quarz verwendet werden. Auf dem Board befindet sich eine 16-Mhz-Quarz. Sollten andere Taktfrequenzen benötigt werden, kann auch ein anderer Quarz eingelötet werden.

Taster und LEDs

Auf dem Stone-Board befinden sich drei Taster und zwei LEDs, die durch das Stone-Board angesprochen werden können. Zusätzlich befinden sich noch eine Power-LED und ein Reset-Taster auf der Platine.

Die Taster sind gegen GND geschaltet; für einen fehlerfreien Betrieb müssen die internen Pull-Up-Widerstände des AVR aktiviert werden. Bei gedrückter Taste liegt eine logische „0“ an dem jeweiligen Pin an. Die LEDs werden mit einer logischen „1“ am Ausgang des jeweiligen Pins aktiviert.

Die Taster sind nicht entprellt, die Entprellung muss in der Software vorgenommen werden.

Buchsenleisten

Auf dem Board befinden sich 4 Buchenleisten. Damit lassen sich alle Pins des AtmegaX8 separat abgreifen und nutzten. Es lassen sich passende Aufsteckplatinen entwerfen, die einen kompakten Gesamtaufbau eines Systems erlauben.

Manche Pins werden schon von anderer Peripherie genutzt, dies sollte beim Entwurf einer Aufsteckschaltungen beachtet werden.

Pins die nur mit Bedacht genutzt werden sollten sind:

- **PD0, PD1:** RX/TX wenn USB Bootloader oder Kommunikation genutzt werden soll.
- **PB3, PB4, PB5:** Taster und MOSI, MISO, SCK bei der Verwendung von ISP/Debug Wire.
- **PB6, PB7:** Externer Quarz

Software

Atmel-Studio 6

Das „Atmel-Studio“ ist die kostenlose Entwicklungsumgebung für nahezu alle 8-Bit und 32-Bit Mikrocontroller von Atmel.

Downloadbar ist die Software entweder über die Atmel-Seite selbst unter Angabe von persönlichen Daten (ggf. funktioniert auch erfundene Daten und eine Trashmail-Adresse) geladen werden oder per Direktlink von der Webseite

http://www.mikrocontroller.net/articles/Atmel_Studio

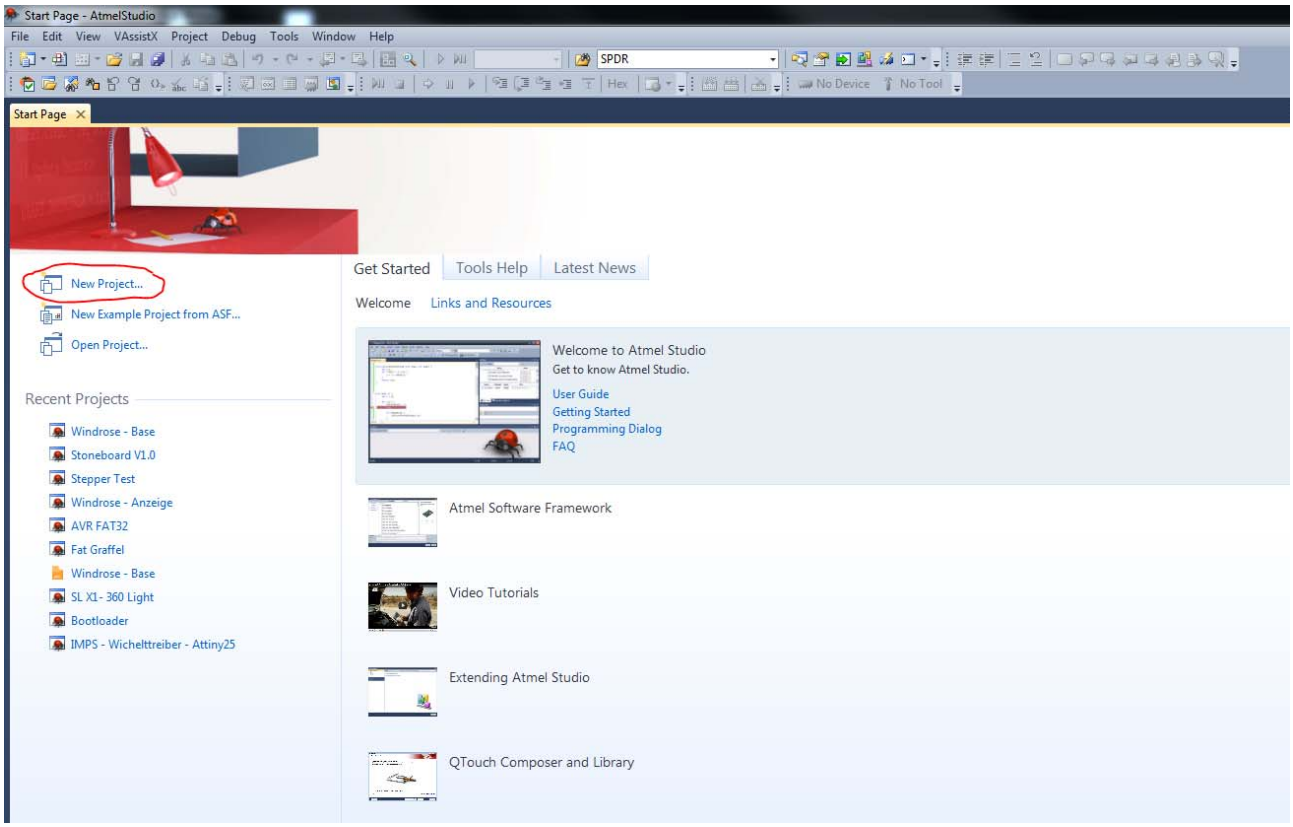
Durch die Microsoft-Visual-Studio-Umgebung funktioniert das Atmel-Studio nur unter Windows. Es gibt auch Alternativen für Linux wie z. B. Eclipse. In der Anleitung wird aber nur auf den Umgang mit Atmel-Studio eingegangen. Im Internet finden sich genug Anleitungen für andere Entwicklungsumgebungen. Die Installation erfolgt nach der für Windows typischen Vorgehensweise.

Projekt öffnen

Nach der erfolgreichen Installation kann nun ein Projekt geöffnet werden.

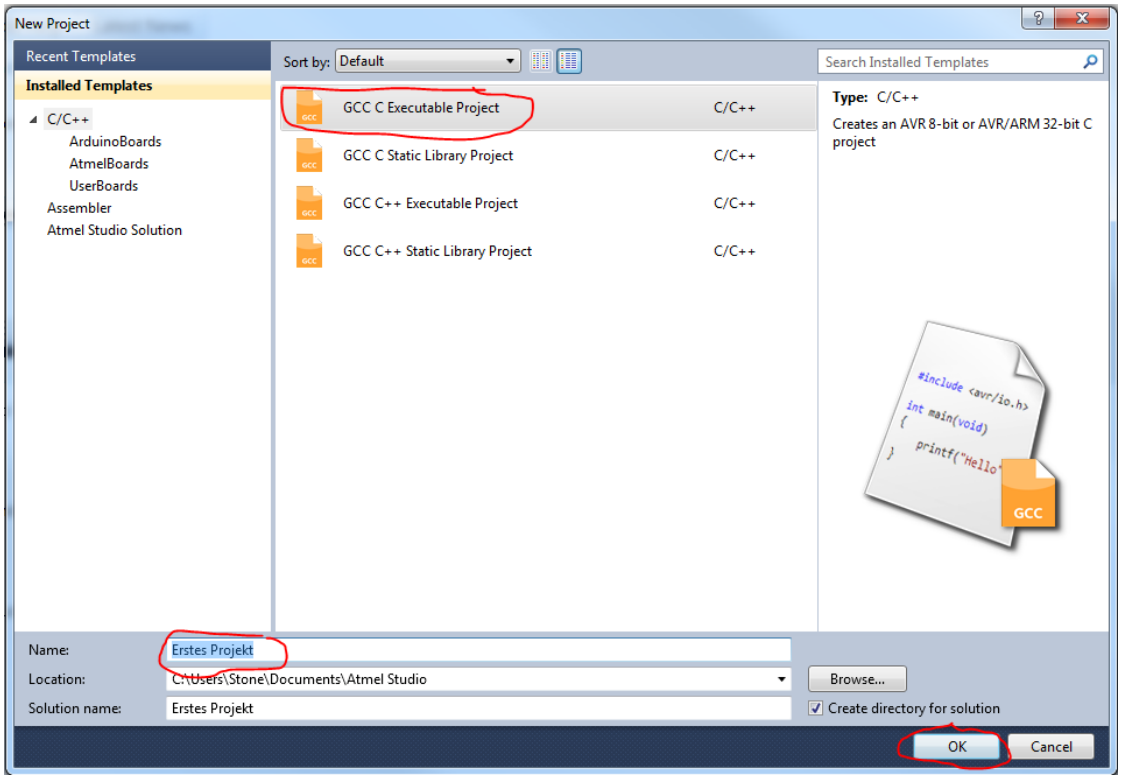
Nach dem Öffnen sollte der Startbildschirm sichtbar sein. Nun auf „New Projekt“ klicken.

Alternativ: *File → New → Projekt*

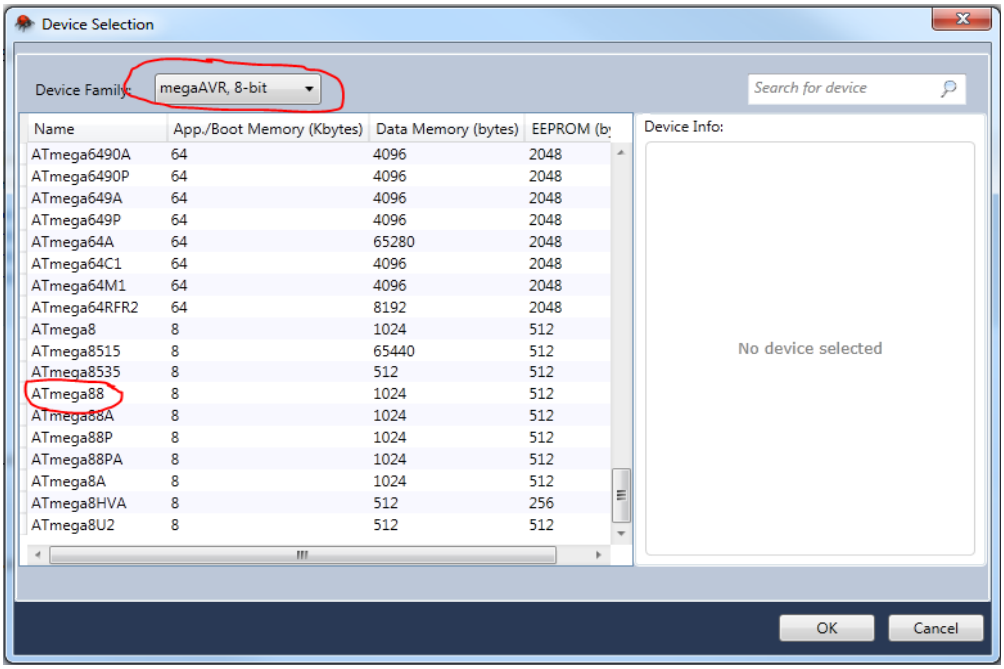


Nun den Compiler Auswählen:

- GCC C Executable Project
- Name und, falls erwünscht, anderen Pfad eingeben **! Keine Umlaute verwenden !**
- mit OK bestätigen



Unter *Device Family* „megaAVR, 8-bit“ und verwendeten Chip auswählen, in diesem Fall z. B. ATmega88. Mit „OK“ bestätigen. Es wird dann ein Projekt erstellt:



Programm oder Bootloader auf den Mikrocontroller übertragen

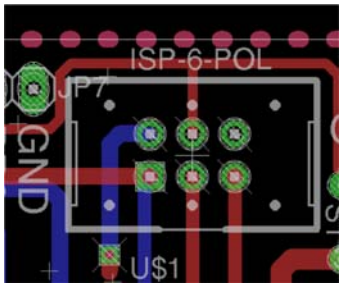
Um das Programm oder einen Bootloader auf den Mikrocontroller zu laden, wird ein ISP-Programmer benötigt, z. B. der AVR MKII oder das Jtag IC3.

Abbildung 1: MK II (Quelle: Atmel.com)

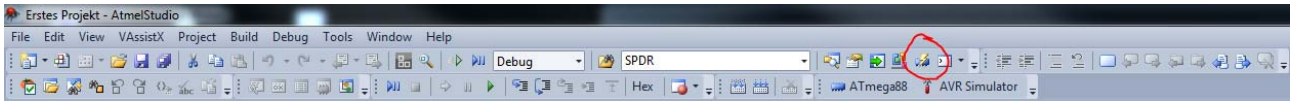
Abbildung 2: JTAGIC3 (Quelle: Atmel.com)

Um das Board zu programmieren, muss das Board mit Spannung versorgt sein und der 6-polige ISP-Stecker eingesteckt sein. Dabei handelt es sich um einen verpolungssicheren Wannenstecker.

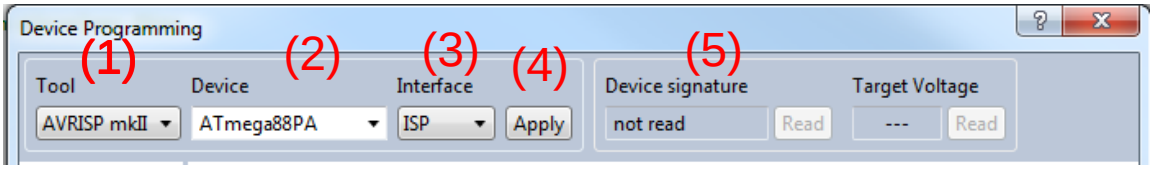
Um in das Programming-Menü zu gelangen, klicken Sie auf „Device Programming“.



Tools → Device Programming



Nun öffnet sich das *Device-Programming-Menü*



Es müssen nun einige Einstellungen vorgenommen werden.

(1) Einstellung des Tools, in diesem Fall das AVRISP MkII

(2) Einstellen des Device-Typs (auf die genaue Bezeichnung achten)

(3) Interface wählen, in diesem Fall ISP

(4) mit „Apply“ übernehmen

(5) Nun kann die Signatur des Chips ausgelesen werden. Das ist ein guter Test, ob alles korrekt angeschlossen und eingestellt ist.

Nun muss die zu übertragende .hex-Datei unter „Memories“ ausgewählt werden. Die Datei befindet sich im Ordner:

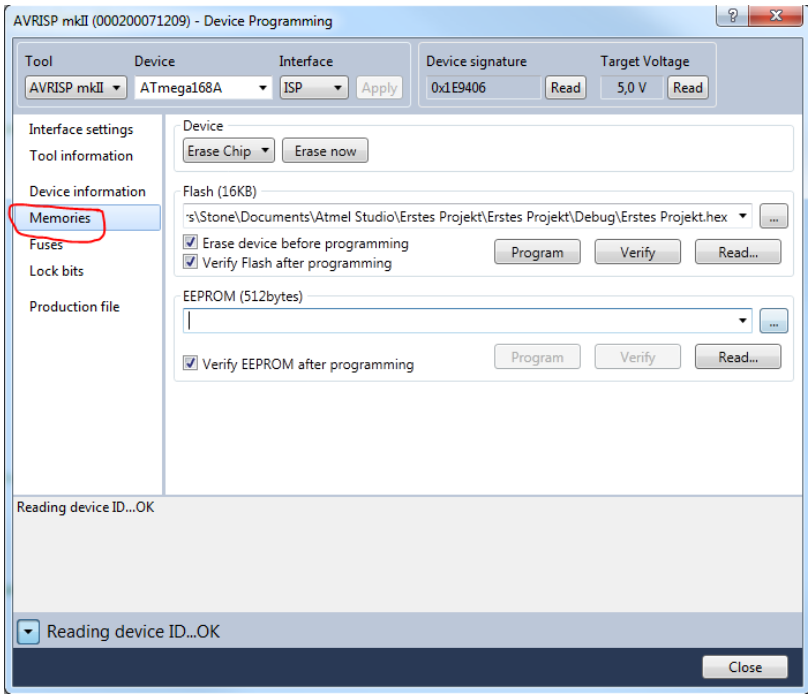
.../Projektordner/Debug/

Ein alternativer Ablageort ist das folgende Verzeichnis für die .hex-Datei, wenn auf Release kompiliert wurde:

.../Projektordner/Release/

Zum Übertragen muss nur noch auf „Program“ geklickt werden.

Sollte es bei der Programmierung zu Problemen kommen, arbeiten Sie zunächst einmal die folgende Checkliste ab.

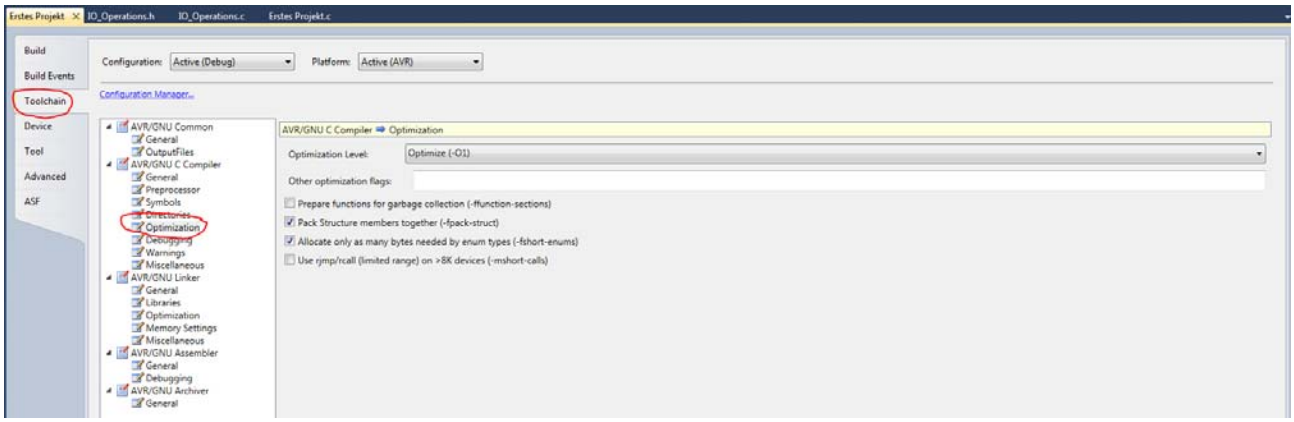


- Stone-Board mit dem ISP-Programmer verbunden und mit Strom versorgt?
- Keine störende Peripherie an den ISP-Anschlüssen?
- Kurzschlüsse oder offene Stellen auf der Platine?
- ISP-Frequenz innerhalb der Spezifikationen (ISP-Frequenz sollte kleiner oder gleich FCPU/4 sein)? Unter „Interface settings“ kann die Frequenz geändert und dann mit „Set“ gespeichert werden. 125 kHz sind ein guter Wert (wenn FCPU >= 1 MHz)

Optimierung

Unter *Project → XXX Properties*

kann die Optimierung eingestellt werden. Wenn man nicht debuggen möchte, am Besten auf „-os“ stellen.



Tipps und Tricks für Atmel-Studio

Neue .c oder .h Files erstellen:

Project → „Add New Item“

Bestehende Files in das Projekt einbilden

Project → „Add Existing Files“

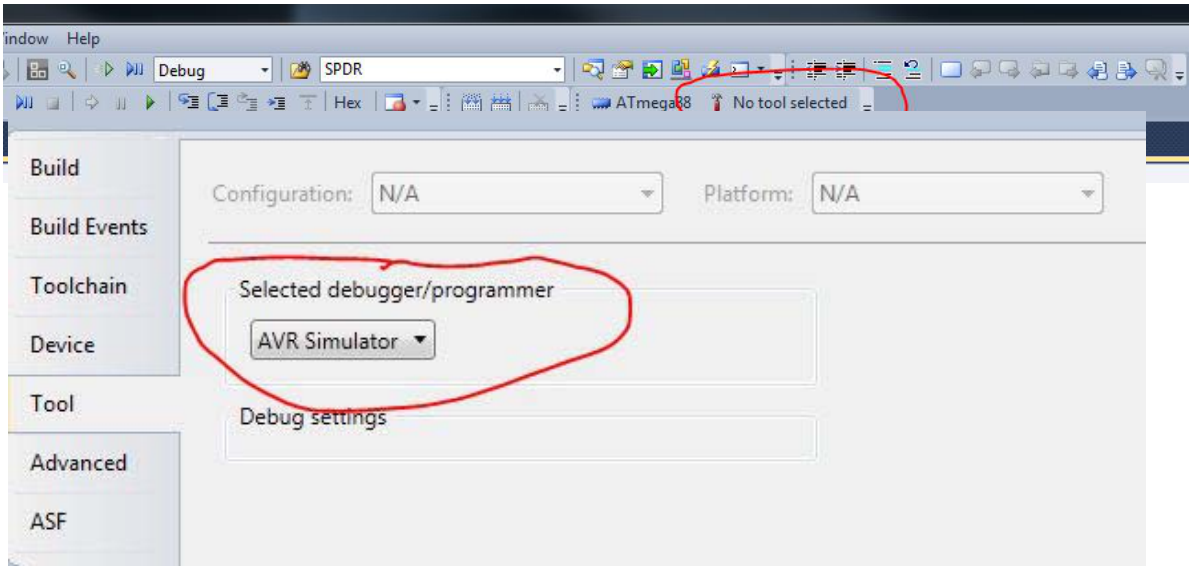
Simulation

Auswahl Debugger

Es folgt ein kleines Programm, um den Debugger zu demonstrieren. Das Programm hat die folgende Aufgabe: Sobald Taster 1 gedrückt ist, leuchtet LED 1 auf, beim Drücken von Taster 2 leuchtet LED 2 auf. Zuvor werden noch die I/O-Pins initialisiert.

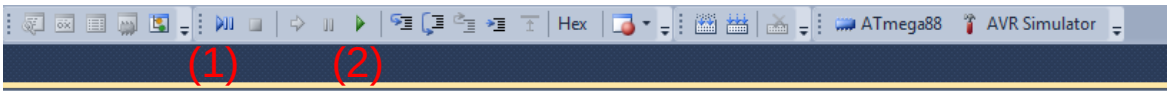
```
/*
 * Erstes_Projekt.c
 *
 * Created: 15.02.2013 14:38:59
 *  Author: Stone
 */


#include <avr/io.h>
#include "IO_Operations.h"

int main(void)
{
    IO_IniPin(IO_PD5, Out); //PD5 und PD6 als Ausgang
    IO_IniPin(IO_PD6, Out);

    IO_IniPullUp(IO_PB3, 1); //Pullups an PB3 und PB4 Aktivieren
    IO_IniPullUp(IO_PB4, 1);


    while(1)
    {
        if(IO_GetPin(IO_PB3) == 0) //Wenn Taster gedrückt
            IO_SetPin(IO_PD5, 1); //Led An
        else
            IO_SetPin(IO_PD5, 0); //Sonst Led aus

        if(IO_GetPin(IO_PB4) == 0) // Taster gedrückt
            IO_SetPin(IO_PD6, 1); //Led An
        else
            IO_SetPin(IO_PD6, 0); //Sonst Led aus


    }
}
```

Zum „Entwanzen“ den Debugger auswählen und anschließend speichern.

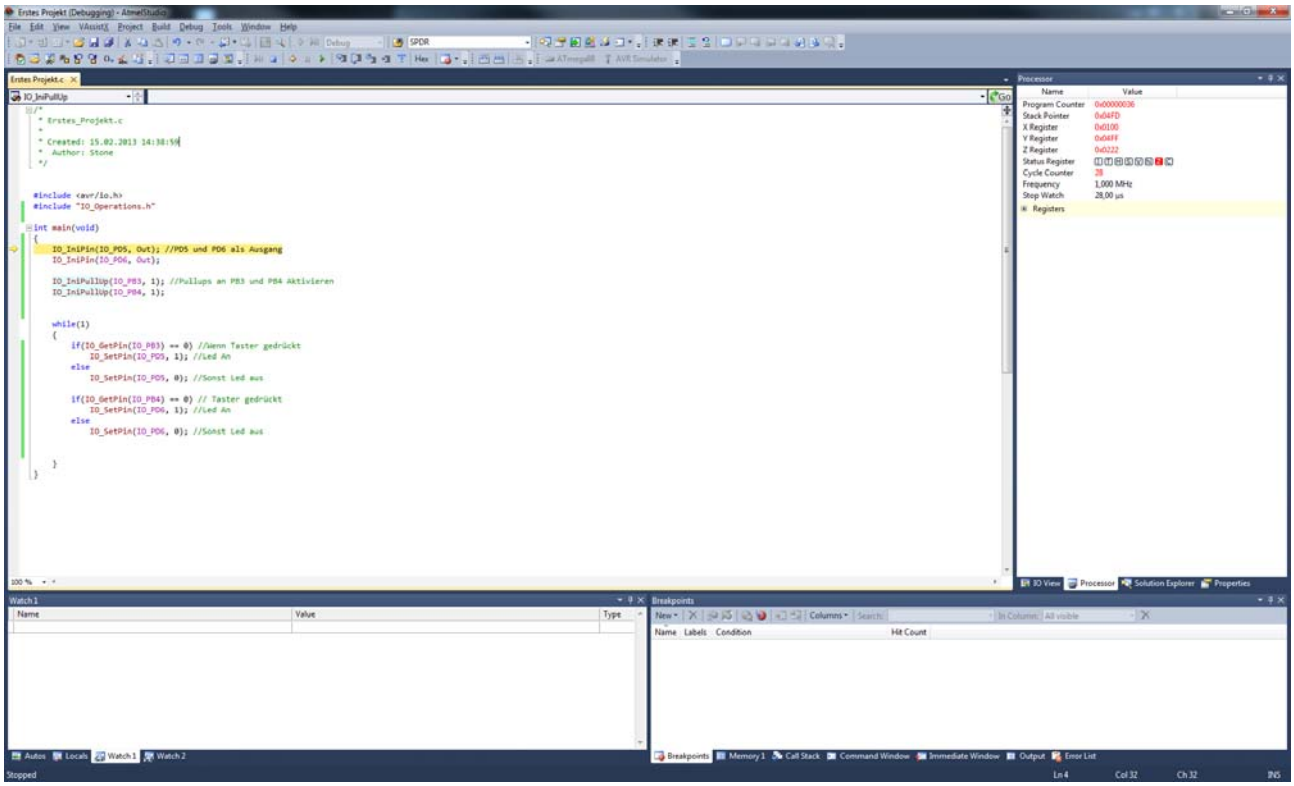


Starten eines Programms mittels Debugger

Zum Starten des Debuggers (1) auswählen, um das Programm zu laden und direkt zu pausieren, oder (2) um das Programm zu Laden und sofort zu starten.

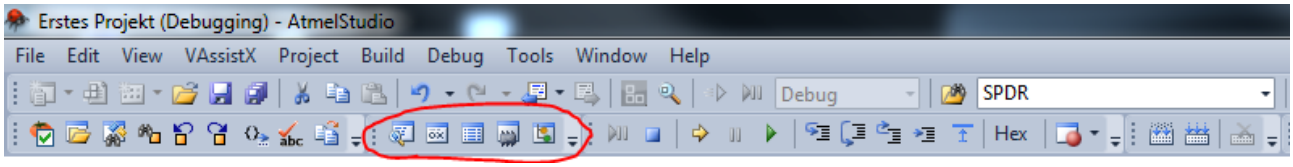


Das Layout sollt jetzt ungefähr so aussehen. Wichtig sind „I/O-View“, „Processor“ und „Watch“:



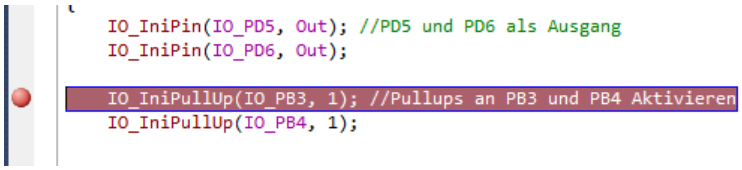
Sollte der IO-Viewer und Prozessor nicht sichtbar sein, können diese über die jeweiligen Buttons aktiviert werden.

Alternativ: *Debug → Windows*



Breakpoints und Debugger-Controls

Um das Programm an einer bestimmten Stelle automatisch anhalten zu lassen, gibt es sogenannte Breakpoints. Diese können durch Anklicken des linken Balkens an der gewünschten Zeile oder durch Rechtsklick in die gewünschte Zeile *Breakpoints → Insert Breakpoint* aktiviert werden.

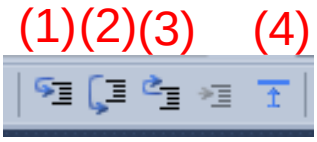


(1) Jump In (Springt in Unterfunktionen)

(2) Step Over (führt die Funktion aus und Stopt wieder nach deren Ausführung)

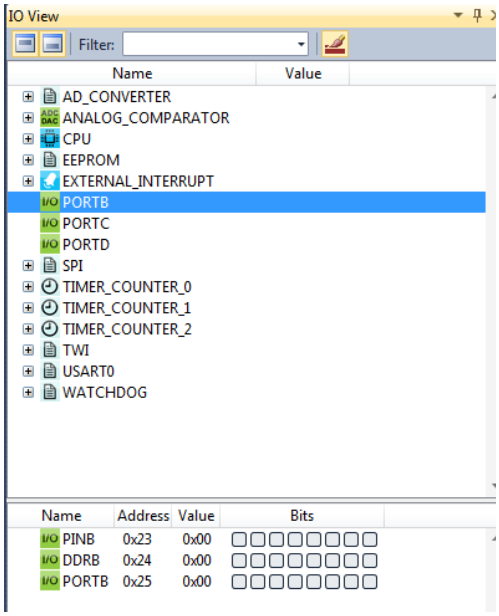
(3) Step Out (Springt aus der Unterfunktion nach deren Ausführung)

(4) Reset (Springt an der Anfang des Programms)



I/O-View

Im I/O View können alle nicht zum Prozessor gehörenden interne Register eingesehen und verändert werden.



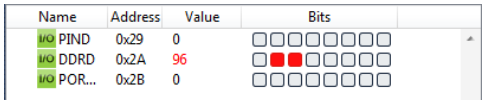
Änderungen und Ansicht stimmt nur im pausierten Modus.

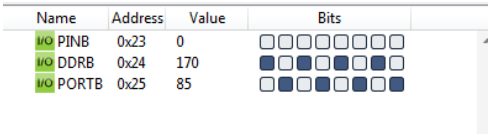
Als Beispiel wird PORTB verwendet. Um nun den Registerinhalt zu ändern, gibt es zwei Möglichkeiten:

- In das Feld „Value“ klicken und den Wert ändern.
- Mit der Maus auf das jeweilige Bit klicken, um es zu ändern

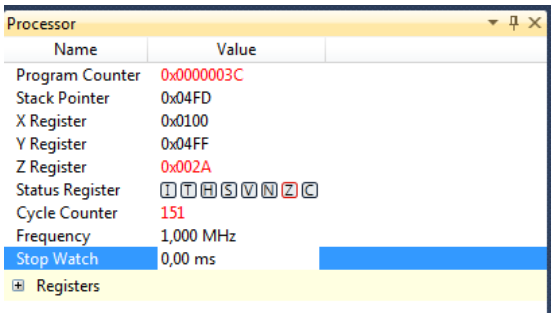
Mit einem Rechtsklick auf „Value“ kann noch zwischen hexadezimaler und dezimaler Darstellung gewechselt werden.

Eine logische „0“ wird mit weißer Füllung eine logische „1“ mit dunkelblauer Füllung angezeigt. Sollte sich seit dem letzten pausieren ein Wert geändert haben, wird dieser Rot hinterlegt.





Processor



Im „Processor“-Fenster können alle Register sowie alle Flags des Prozessors eingesehen werden. Zusätzlich können hier Laufzeiten und „Ausführungszyklen“ gemessen werden.

Die „Stop Watch“-Anzeige kann z. B. genutzt werden, um die Aufrufhäufigkeit einer Timer-ISR zu messen, oder die Zeit, die ein Programmteil zur Ausführung benötigt. Per Rechtsklick kann die Stoppuhr zurück gesetzt werden und zwischen Milli- und Mikrosekunden-Darstellung gewechselt werden.

Für korrekte Ergebnisse muss drauf geachtet werden, dass „Frequency“ auf die korrekte Prozessorfrequenz der realen Hardware gestellt ist.

Watch

Im Fenster „Watch“ können Variablen angezeigt und verändert werden. Alternativ kann man auch während des pausierten Programms mit dem Mauszeiger auf die Variable gehe. Dann wird deren Wert auch angezeigt.

Sollte das Watch-Fenster oder Unterpunkte davon nicht angezeigt werden (z. B. Lokal, Auto, …), können diese aktiviert werden(nur wenn der Debugger aktiv ist).

Unter: Debug → Windows

Teilweise werden (lokale) Variablen vom Compiler auf Register gelegt und können nicht vom Debugger angezeigt werden. Durch den Zusatz „volatile“ kann dies umgangen werden (z. B. : volatile uint8_t).

Terminalprogramme

HTerm

HTerm ist ein einfaches Terminalprogramm. Es kommt ohne Installation aus und kann kostenlos von *http://www.der-hammer.info/terminal/index.htm* heruntergeladen werden.

Es eignet sich sehr gut im Mikrocontroller-Bereich, da es mehrere Darstellungsformen (Hexadezimal, Binär, ASCII, Dezimal) bei Ein- und Ausgabe ermöglicht. Zudem können das Zeichen für ein „Newline“, Datenformat, Baudrate usw. eingestellt werden.

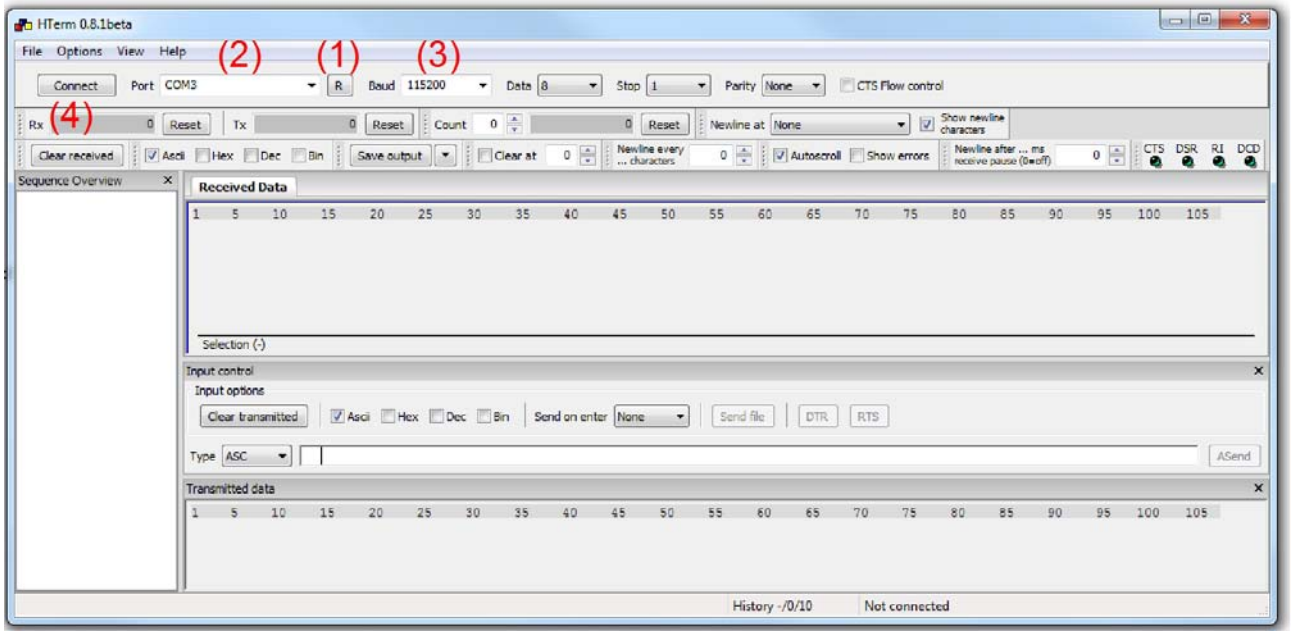


Abbildung 3: HTerm

Um eine Verbindung mit dem Stone-Board herzustellen empfiehlt sich folgendes Vorgehen

- HTerm öffnen.
- Unter „Port“ (2) nachschauen welche COM-Ports verfügbar sind.
- Stone-Board über USB anstecken.
- Prüfen ob die Power-LED auf dem Stone-Board leuchtet.
- Gegebenenfalls warten, bis die Treiberinstallation für den FT232 abgeschlossen ist.
- Refresh-Button (1) drücken sowie überprüfen, ob ein neuer COM-Port erschienen ist und diesen auswählen.
- Die richtige Baudrate einstellen (3) (so wie kompiliert wurde).
- Nun auf „Connect“ (4) klicken.
- Testweise können jetzt Zeichen gesendet werden (RX-LED sollte aufblitzen).

PuTTY

PuTTY (von *http://www.putty.org/*) ist ein sehr universelles Terminalprogramm, das auch Verbindungen über Netzwerke und Internet ermöglicht. Jedoch ist es durch seine reine ASCII-Darstellung nicht so für alle Zwecke geeignet. Es besitzt auch deutlich weniger Einstellmöglichkeiten für diesen Einsatzzweck als HTerm.

USB-Bootloader

Von der Webseite *http://www.chip45.com* kann ein kostenloser Bootloader heruntergeladen werden. Er besteht aus den vorkompilierten Hexdateien für viele AVR Mikrocontroller.

Über die Boot-GUI kann die Programm-Hex-Datei und, falls vorhanden, die EPP-Datei (EEPROM) in den Controller über tragen werden.

Die Datei sollte sich im Ordner

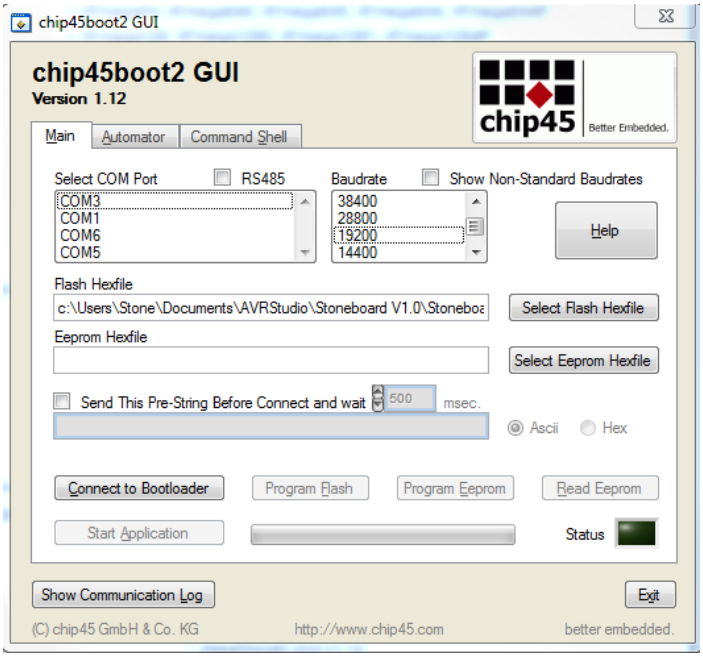
.../Projektordner/Debug/

befinden. Alternativ, wenn auf Release kompiliert wurde, sollte sie im Ordner

.../Projektordner/Release/

liegen.

Der Bootloader wird per ISP-Schnittstelle auf den Mikrocontroller geladen. Siehe *Programm oder Bootloader auf den Mikrocontroller übertragen.*



Um die Hex-Datei eines Programms bei installierten Bootloader zu übertragen, gehen Sie folgendermaßen vor:

- Stone-Board per USB an den PC anschließen und mit Strom versorgen.
- Neu erschienenen COM-Port wählen.
- Baudrate wählen (der Bootloader passt sich automatisch an).
- Flash-File auswählen.
- Reset-Taster auf dem Stone-Board drücken.
- Innerhalb von 3s auf „Connect to Bootloader“ drücken.
- „Programm Flash“ drücken.
- „Start Applikation“ drücken, dann läuft das Programm.

Libraries

Die Libraries sollen einen einfachen Einstieg in die Mikrocontroller-Welt ermöglichen, ohne sich Anfangs mit vielen Registern und korrekten Bit-Verknüpfungen zu beschäftigen. Für kleine Projekte reichen die Libraries vollkommen aus. Bei wirklich aufwendigen und zeitkritischen Aufgaben sollte lieber auf klassischen C- oder Assembler-Code zurückgegriffen werde.

Um eine Library in das Projekt einzubinden, müssen deren Header- und Quell-Datei eingefügt werden. Um bestehende Files in das Projekt einzubinden, wählen Sie das Menü

Project → „Add Existing Files“

Zusätzlich muss noch die Datei im Quelltext inkludiert werden, z. B. mittels

```
#include "IO_Operations.h"
```

Eine Zip-Datei mit den Libraries ist unter *http://www.netzmafia.de/skripten/hardware/Stone_Board/* abgelegt.

Wer ohne die Librarys auskommen will, findet unter *http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial* ein gutes Einsteigertutorial für die C-Programmierung von AVR-Mikrocontrollern.

I/Os

Die I/O-Library ist für die einfache Ansteuerung der I/Os zuständig.

Wichtig: Einer der Übergabeparameter ist „Io_Number“ hier muss immer „IO_PXX“ übergeben werden, nicht „PXX“

Richtiges Beispiel:

```
IO_IniPin(IO_PD5, Out);
```

Falsches Beispiel:

```
IO_IniPin(PD5, Out);
```

```
void IO_IniPin(Io_Number IoNumber, Io_Mode IoMode);
```

Mit der IO_IniPin-Funktion kann ausgewählt werden ob der Pin als Ausgang oder Eingang verwendet werden soll. Nach einem Reset des Mikrocontroller sind die Pins immer als Eingang initialisiert d.h. es müssen nur Ausgänge explizit eingestellt werden.

Beispiel:

```
IO_IniPin(IO_PD5, Out);

IO_IniPin(IO_PD5, In);
```

```
uint8_t IO_GetPin(Io_Number IoNumber);
```

Mit `IO_GetPin` kann der Zustand eines Eingans-Pins abgefragt werden.

Beispiel:

```
if(IO_GetPin(IO_PB2) == 1) // Wenn PB2 High ist tuDies sonst tuDas
```

```
  { tuDies(); }
else
  { tuDas(); }
```

```
void IO_SetPin(Io_Number IoNumber, uint8_t IoValue);
```

Mit `IO_SetPin` kann ein Ausgang auf „High“ oder „Low“ gesetzt werden.

Beispiel:

```
IO_SetPin(IO_PD5, 1); //Led An
IO_SetPin(IO_PD5, 0); //Led Aus
```

```
void IO_TogglePin(Io_Number IoNumber)
```

Mit dieser Funktion kann man einen bestimmten Ausgang toggeln. Sie funktioniert nur, wenn Optimierung auf `-os` steht. Siehe Optimierung.

Beispiel:

```
IO_TogglePin(IO_PD5); //Toggle LED
```

```
void IO_IniPullUp(Io_Number IoNumber, uint8_t IoValue);
```

Der AtmegaXX besitzt an jedem Pin einen internen Pull-Up-Widerstand dieser kann mit der Funktion `IO_IniPullUp` aktiviert oder deaktiviert werden. **Default ist „0“ also deaktiviert.**

Beispiel:

```
IO_IniPullUp(IO_PD2, 1); //Aktiviert den Pull-Up An PB2
IO_IniPullUp(IO_PD2, 0); //Deaktiviert den Pull-Up An PB2
```

```
void IO_IniPort(Io_Port IoPort, uint8_t IoValue);
```

Um bei einem ganzen Port nicht immer umständlich jeden Pin einzeln konfigurieren zu müssen, gibt es die Funktion `IO_IniPort`. Damit kann ein ganzer Port auf einmal konfiguriert werden.

Beispiel:

```
IO_IniPort(IO_PORTB, 0b11010011); //Binäre Konfiguration LSB=PB0 und MSB=PB7, „1“ = Ausgang.
IO_IO_IniPort(IO_PORTD, 0x08); //Hexadezimale Konfiguration
IO_IO_IniPort(IO_PORTC, 132); //Dezimale Konfiguration
```

```
uint8_t IO_GetPort(Io_Port IoPort);
```

Es ist nicht nur möglich, einen ganzen Port zu konfigurieren sondern auch auszulesen.

Beispiel:

```
uint8_t uiTempPortC;
...
uiTempPortC = IO_GetPort(IO_PORTC);
```

```
void IO_SetPort(Io_Port IoPort, uint8_t IoValue);
```

Auch das Setzen eines kompletten Ports ist möglich.

Beispiel:

```
IO_SetPort(IO_PORTD, 0b00011100);
```

UART

```
void Uart_Ini(void);
```

Vor Benutzung der UART-Schnittstelle muss diese erst konfiguriert werden. Das erfolgt über den einmaligen Aufruf von `Uart_Ini();`

```
void Uart_Puts (volatile char*);
```

Per `Uart_Puts` wird ein String über die UART-Schnittstelle verschickt.

Beispiel:

```
Uart_Puts(„Hallo Welt“);

char cTest[11] = „Hallo Welt“;
...
Uart_Puts(cTest);
```

```
void Uart_Putc(unsigned char);
```

Sendet nur ein Zeichen bzw. Byte über die UART-Schnittstelle.

Beispiel:

```
Uart_Putc('H');
Uart_Putc(115);
```

```
uint8_t Uart_Gets(char*, uint8_t);
```

Liest einen String ein. Sobald dieser komplett eingetroffen ist, ist der Rückgabewert „1“ sonst „0“. Ein kompletter String wird durch ein `'\r'` oder `'\n'` abgeschlossen. Übergabewert ist der Puffer-String in dem die empfangenen Zeichen gespeichert werden sollen und dessen Länge.

Beispiel:

```
char sPuffer[20];
...

if(Uart_Gets(sPuffer, 20)  // Wenn neuer String eingetroffen
  { tuDies(); }
```

```
void Uart_NewLine(void);
```

Sendet ein DOS/Windows-Zeilenende (`'\r'` gefolgt von `'\n'`) über die UART-Schnittstelle.

```
uint8_t Uart_Received(void);
```

Prüft, ob ein neues Zeichen auf der UART-Schnittstelle eingetroffen ist. „1“: neues Zeichen eingetroffen, „0“: kein neues Zeichen eingetroffen.

Beispiel:

```
if(Uart_Received())
   tuDiesMitDemNeuenZeichen();
```

Time

```
void TIME_Ini();
```

Vor Benutzung der Timer-Funktionen muss der Timer erst Konfiguriert werden. Das erfolgt über den einmaligen Aufruf von `TIME_Ini()`. **Für die Timer-Funktion wird Timer1 verwendet!**

```
uint64_t TIME_Millis(void);
```

Gibt die Zeit in Millisekunden seit dem Systemstart zurück.

```
uint64_t TIME_Micros(void);
```

Gibt die Zeit in Mikrosekunden seit dem Systemstart zurück.

```
void TIME_Delay_ms(uint16_t);
```

Die Funktion wartet X Millisekunden (max. 65536 ms (65 s). Für längere Zeiten bitte die Funktion `TIME_Delay_s(uint16_t)` verwenden).

```
void TIME_Delay_us(uint16_t);
```
Wartet X Mikrosekunden (max. 1000 µs. Für längere Zeiten bitte die Funktion `TIME_Delay_ms (uint16_t)` verwenden)

```
void TIME_Delay_s(uint16_t);
```

Wartet X Sekunden (max. 2^16 Sekunden; das entspricht 18 Tagen).

Die drei Delay-Funktionen können natürlich auch kombiniert werden, um auf „krumme“ Zeiten zu kommen.

Beispiel:

```
TIME_Delay_us(500);
TIME_Delay_ms(2);  //zusammen für 2.5ms Wartezeit
```

Die Mikrosekunden-Zeiten sollten nur mit Bedacht genutzt werden. Für wirklich genaues Timing sind diese ungeeignet, da sie je nach Auslastung des Mikrocontroller nicht zwingend zum Aufrufzeitpunkt direkt ausgeführt werden. Es sollte auch nie auf Gleichheit mit einem vorgegebenen Wert geprüft werden, sondern immer auf „größer gleich Wert“.

ADC

```
void ADC_Ini(ADC_RefType RefType);
```

Vor Benutzung des ADC-Wandlers muss diese erst konfiguriert werden. Das erfolgt über den einmaligen Aufruf von `ADC_Ini()`. Es stehen drei Referenzquellen zur Verfügung:

- ADC_RefInt1V1 (Interne 1.1-V-Referenz)
- ADC_RefVcc (Vcc als Referenz)
- ADC_RefExt (Externe Referenz, die am Ref-Pin angeschlossen ist)

```
uint16_t ADC_GetADCPin(Adc_Number AdcNumber);
```

Führt eine Analog-Digital-Wandlung an dem ausgewählten ADC-Pin aus.

Beispiel:

```
uint16_t uiAdcVal0, uiAdcVal1;
...
uiAdcVal0 = ADC_GetADCPin(ADC_PIN0); // ADC-Wandlung an PC0
uiAdcVal1 = ADC_GetADCPin(ADC_PIN1); // ADC-Wandlung an PC1
if(uiAdcVal0 > uiAdcVal1)
   tuDies();
else
   tuJenes();
```

Servo

Für die Ansteuerung von Modellbauservos wurde eine spezielle Library geschrieben. Mit dieser

können bis zu acht Servos unabhängig angesteuert werden. Dazu werden Impulse zwischen 1 und 2 ms Breite und 20 ms Wiederholrate erzeugt. Zur Erzeugung wird Timer2 benötigt. Es kann jeder beliebige I/O-Pin zur Ausgabe genutzt werden. **Nicht zwingend kann jede Position angefahren werden, an den Grenzen kann der Servo an den mechanischen Anschlag geraten. Die maximalen Stellgrenzen sollte vorab getestet werden um Schäden zu vermeiden.**

```
void SERVO_Ini(void)
```

Die Funktion `SERVO_ini()` muss einmalig aufgerufen werden, um den Timer2 zu initialisieren.

```
void SERVO_Add_Servo(Io_Number IoNumber, uint8_t IniValue);
```

Mit dieser Funktion wird ein Pin in die „Servoliste“ eingetragen.

Beispiel:

```
SERVO_Add_Servo(IO_PD2, 128); //Servo an PD2, 128 entspricht der Mittelstellung
```

```
void SERVO_Change_Servo_Value(Io_Number IoNumber, uint8_t NewValue);
```

Mit dieser Funktion wird die Position des Servos geändert.

Beispiel:

```
SERVO_Change_Servo_Value(IO_PD2, 0); //Ändert die Position des Servos auf das linke max.
```

PWM

```
void PWM_Ini(Pwm_Use_Timer Timer);
```

Es stehen vier Hardware-PWM-Kanäle über die Library zur Verfügung. Es handelt sich um 8-Bit PWMs, damit liegt der Wertebereich zwischen 0 und 255. Zur Auswahl stehen PD5 und PD6 auf Timer0 und PB3 und PD3 auf Timer2. Um die PWM-Kanäle nutzen zu können, müssen erst die Timer initialisiert werden.

Beispiel:

```
PWM_Ini(Timer0); // Initialisiert den Timer0 für 8-Bit PWM damit stehen PD5 und PD6 …
                 //  … für eine PWM zur Verfügung
```

```
void PWM_Add_Pin(Pwm_Number IoNumber, uint8_t uiIniValue);
```

Um die PWM an dem Pin tatsächlich zu aktivieren, muss die Funktion `PWM_Add_Pin()` aufgerufen werden. Dadurch wird der Pin auf Ausgang geschaltet und die Pincontrolle an das Compare-Register übertragen. Es muss noch ein Initialisierungswert zwischen 0 und 255 übergeben werden.

Beispiel:

```
PWM_Add_Pin(PWM_PD6, 128); //Aktiviert die PWM an PD6 und Initialisiert den PWM Wert auf 128
```

```
void PWM_Del_Pin(Pwm_Number IoNumber);
```

Um die PWM an einem Pin wieder zu deaktivieren kann die Funktion `PWM_Del_Pin()` genutzt werden. Dabei wird dem Compare-Register die Kontrolle entzogen und der Pin kann wieder normal angesteuert werden. Der Pin wird auch wieder als Eingang konfiguriert.

Beispiel:

```
PWM_Del_Pin(PWM_PD6); // Sperrt den Pin für die PWM
```

```
void PWM_Change_Value(Pwm_Number IoNumber, uint8_t uiNewValue);
```

Mit dieser Funktion wird der PWM-Wert geändert. Übergabeparameter sind die *IoNumber* und der neue PWM-Wert (0-255).

Beispiel:

```
PWM_Change_Value(PWM_PD6, 255); //Ändert den Wert an PD6 auf 255
```

Interrupt

Der AtmegaX8 verfügt über mehrere externe Interrupts wie Int0 (PD2), Int1 (PD3) und alle I/O-Pins verfügen über PCIs (**P**in **C**hange **I**nterrupt). Diese können einfach per Library aktiviert bzw. deaktiviert und konfiguriert werden.

Int0 und Int1 können auf verschiedene Trigger-Arten konfiguriert werden wie z. B. fallende und steigende Flanken, Pegelwechsel oder Low-Pegel. Für Int0 und Int1 steht jeweils eine eigene ISR zur Verfügung, die nur aufgerufen wird, wenn der jeweilige Interrupt aufgetreten ist.

PCIs stehen an jedem Pin zur Verfügung. Im Gegensatz zu Int1 und Int0 können diese nicht vielfältig konfiguriert werden. Auch sind diese in Gruppen zusammen gefasst und es wird nur ein Interrupt pro Gruppe ausgelöst. Der Interrupt wird bei Pegelwechsel von einem der aktivierten Pins ausgeführt. Um die Trennung, welcher Pin ausgelöst hat und wie dessen Zustand ist muss sich der Benutzer selbst kümmern. So ergibt sich folgende Gruppierung der PCI Pins:

- PB0-PB7 PCI0
- PC0-PC5 PCI1
- PD0-PD7 PCI2

Alle ISRs befinden sich in der INTERRUPT_Library.c. In diese muss der eigene Code eingetragen werden.

```
void INTERRUPT_Activate_Int0(INTERRUPT_Mode IntMode);
```

Diese Funktion aktiviert INT0.
Modes:

- LOW
- CHANGE

- RISING
- FALLING

Beispiel:

```
INTERRUPT_Activate_Int0(FALLING); //aktiviert INT0 getriggert auf fallende Flanke
```

```
void INTERRUPT_Deactivate_Int0(void);
```

Deaktiviert den Int0 wieder.

```
void INTERRUPT_Activate_Int1(INTERRUPT_Mode IntMode);
```

Diese Funktion aktiviert INT1. Siehe `INTERRUPT_Activate_Int0`.

```
void INTERRUPT_Deactivate_Int1(void);

Deaktiviert den Int1 wieder.
```

```
void INTERRUPT_Activate_Pci(Io_Number IoNumber);

Aktiviert den PCI an dem angegebenen Pin.
```

```
void INTERRUPT_Deactivate_Pci(Io_Number IoNumber);

Deaktiviert den PCI an dem angegebenen Pin.
```

NUM-Operations

```
int32_t NUM_Limit(int32_t uiValue, int32_t uiMinValue, int32_t uiMaxValue);
```

Begrenzt `uiValue` auf einen Wertebereich: `uiMinValue <= uiValue <= uiMaxValue`

```
uint16_t NUM_Map(uint16_t uiValue, uint16_t uiMinValue1, uint16_t uiMaxValue1, uint16_t
uiMinValue2, uint16_t uiMaxValue2);
```

Mapping eines Datenbereichs auf einen anderen. `Value1`: Wertebereich von `uiValue`, `Value2`: Wertebereich des Ergebnisses.

Beispiel:

```
uint8_t uiPWMVal; //PWM Wert von 15-230
uint16_t uiAdcVal; // Adc Wert von 0-1023
…
uiPWMVal = NUM_Map(uiAdcVal, 1023, 0, 230, 15);
```

```
void NUM_Srand(uint16_t uiStartwert);
```

In Kombination mit `NUM_Random()` dient die Funktion zum Erzeugen eines Pseudozufallswerts. Mit `NUM_Srand()` wird der Startwert der Pseudozufallszahlenreihe festgelegt.

```
uint16_t NUM_Random(void);
```

Bei jedem Aufruf von `NUM_Random()` wird ein neuer 16-Bit Pseudozufallswerts erzeugt.

Serial

Die Serial-Library besteht nur aus einer Funktion, die Daten über einen beliebigen Data-Pin und Clock-Pin schiebt. Die Geschwindigkeit ist fest vorgegeben. Sollten weitere Anforderungen an die serielle Übertragung gestellt werden, ist die SPI-Library zu verwenden. Diese ist jedoch auf die SPI-Schnittstelle begrenzt und die Pins sind festgelegt.

```
void SERIAL_Shift(Io_Number ClockPin, Io_Number DataPin, Serial_Order Order, uint8_t Data);
```

Schiebt "Data" über den "DataPin" raus Clock wird an "ClockPin" ausgegeben. Mit „Order“ kann bestimmt werden ob das Datenbyte MSB (**M**ost **S**ignificant **B**it) → LSB (**L**east **S**ignificant **B**it) oder LSB → MSB gesendet wird.

Optionen

- MsbFirst
- LsbFirst

Beispiel:

```
SERIAL_Shift(IO_PD5, IO_PD6, MsbFirst, 42);
```

Sound

```
void SOUND_Beep(Io_Number SoundPin, uint16_t Frequenz, uint32_t BeepTime);
```

Die Funktion dient dazu um eine Ton an einem beliebigen Pin mit einer beliebigen Frequenz (in Hz) für eine bestimmte Zeit (in ms) auszugeben.

Beispiel:

```
SOUND_Beep(IO_PC3, 2000, 500); //erzeugt einen 2kHz-Ton für eine halbe Sekunde
```

SPI

SPI (**S**erial **P**eripheral **I**nterface) ist sehr verbreitet im Mikrocontroller-Bereich. Eine Vielzahl von Speichern, Sensoren und anderer Peripherie verwendet SPI. Es können mehrere Geräte an einem SPI-Bus hängen, die Auswahl mit welchem Gerät kommuniziert, geschieht über eine jeweilige Chip-Select-Leitung.

Sollten mehrere SPI-ICs am Bus hängen, die jeweils unterschiedliche Konfigurationen der Schnittstelle benötigen, kann vor jedem Wechsel die Schnittstelle per `SPI_Master_Ini()` neu initialisiert werden, ein `SPI_Disable()` ist nicht nötig.

Die Pins der SPI-Schnittstelle sind `SCK (PB5), MISO(PB4), MOSI(PB5).`

```
void SPI_Master_Ini(Io_Number ChipSelect_Pin, SPI_Master_DataOrder Order,
SPI_Master_ClockPolarity Polarity, SPI_Master_ClockPhase ClockPhase, uint8_t Prescaler);
```

Die Funktion dient der Initialisiertung des SPI-Masters. Es stehen etliche Konfigurationsmöglichkeiten zur Verfügung. Welche Einstellungen für einen bestimmten Chip benötigt werden, findet sich im jeweiligen Datenblatt.

1. ChipSelect_Pin: Initialisiert den Pin als Ausgang und Deaktiviert den Chipselect (wird immer erst vor dem Senden automatisch aktiviert).
2. Mit DataOrder kann bestimmt werden ob das Datenbyte MSB (**M**ost **S**ignificant **B**it) → LSB (**L**east **S**ignificant **B**it) oder LSB → MSB gesendet wird.

   Optionen: `MSB_First` oder `LSB_First`
3. Mit ClockPolaritiy wird entschieden, ob der Clockpin im IDLE-Zustand „high“ oder „low“ ist.

   Optionen: `IDLE_Low` oder `IDLE_High`
4. Mit ClockPhase kann entschieden werden, bei welcher Flanke das Signal übernommen wird.

   Optionen: `Leading_Edge_Sample` oder `Leading_Edge_Setup`
5. Mit Prescaler kann ein Vorteiler für den SPI-Clock eingestellt werden. Manche Peripherie-Bausteine kommen nur mit einer gewissen SPI-Frequenz zurecht. Der Prescaler kann nur Zweierpotenzen als Wert annehmen (2-128) F_SPI= F_CPU/Prescaler

Beispiel:

```
SPI_Master_Ini(IO_PD2, MSB_First, IDLE_Low, Leading_Edge_Sample, 64);
```

```
uint8_t SPI_Send_Message(Io_Number ChipSelect_Pin, uint8_t Message);
```

Mit dieser Funktion wird jeweils ein Byte übertragen. Der Rückgabewert ist die Antwort auf das gesendete Datenpaket. Als Übergabeparameter muss der ChipSelect-Pin mit dem der IC angesteuert und das Datenbyte angegeben werden.

```
void SPI_Disable(void);
```

Wird genutzt, um die SPI-Schnittstelle wieder zu deaktivieren.

I²C

I²C, auch TWI(**T**wo **W**ire **I**nterface) genannt, ist ebenfalls sehr verbreitet im Mikrocontroller-Bereich. Eine Vielzahl von Speichern, Sensoren und anderer Peripherie verwendet I²C. Mit I²C ist es möglich, bis zu 112 Knoten (ICs) mit nur zwei Leitungen anzusprechen.

```
void I2C_Master_Ini_Speed(uint32_t uiSpeed);
```

Mit dieser Funktion kann die Übertragungsgeschwindigkeit in Hz eingestellt werden.

```
uint8_t I2C_Master_StartSending(uint8_t Adresse, I2C_Master_Aktion Aktion);
```

Senden der Startbedingung und Adresse des Slaves. Wenn erfolgreich (Ack vom Slave), Rückgabewert '1' sonst '0'. Option des Parameters Aktion: `WRITE` oder `READ`

```
uint8_t I2C_Master_SendMessage(uint8_t Message);
```

Sendet "Message" an den adressierten Slave. Wenn erfolgreich (Ack vom Slave): Rückgabewert '1' sonst '0'.

```
void I2C_Master_StopSending(void);
```

Sendet Stopbedingung auf den Bus.

```
uint8_t I2C_GetMessageAck(void);
```

Holt Daten vom Slave ab und sendet Ack. Rückgabewert: Message des Slave.

```
uint8_t I2C_GetMessageNoAck(void);
```

Holt Daten von Slave ab, ohne Ack zu senden (letztes Paket bevor die Stopbedingung gesendet wird). Rückgabewert: Message des Slave.

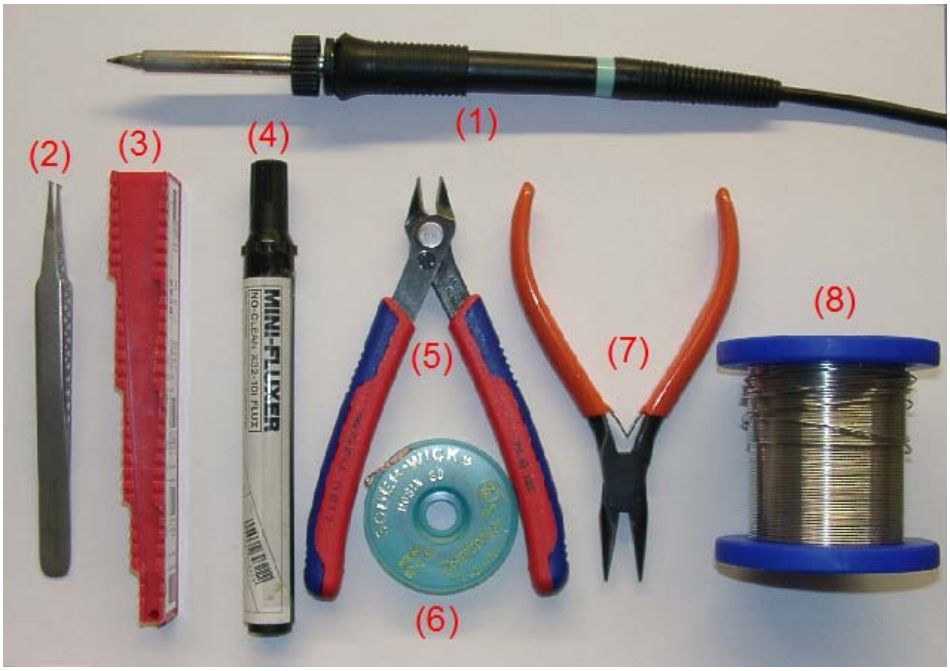
```
I2C_Status_Master_Codes I2C_GetI2cState(void);
```

Holt den aktuellen I²C Status ab.

Löten ist vonnöten

Das Stone-Board soll nicht nur den Einstieg in die Mikrocontroller-Welt ermöglichen sondern auch die praktischen Fähigkeiten verbessern. USB-Buchse, RX-TX-LEDs und der FT232 sind schon vorbestückt, alle anderen Komponenten müssen von Hand bestückt werden.

Das Tutorial ist für bleihaltiges Lötzinn mit Flussmittelseele ausgelegt.



Alle benötigten Werkzeuge sind am Lötplatz vorhanden.

(1) Lötkolben (Weller WSP80 an Lötstation PU81)

(2) SMD-Pinzette

(3) Biegevorrichtung für Widerstände

(4) Flussmittelstift

(5) Elektronik-Seitenschneider

(6) Entlötlitze

(7) Flachzange

(8) bleihaltiges Lötzinn 0.5 mm

Lötkolben

Im Labor steht ein Lötstation des Types „Weller PU81“ zur Verfügung mit einem Lötkolben des Types „Weller WSP 81“. Es stehen eine Vielzahl von verschieden Spitzen und drei Hülsen (1) zur Verfügung. Das Austauschen der Spitzen ist sehr einfach und schnell erledigt. Es ist auch ohne Werkzeug möglich, die Spitzen im eingeschalteten Zustand zu wechseln. Dafür muss nur die Kunststoff-Rändel-Mutter (1.1) an der Hülse gelöst werden; dann die heiße Spitze mit der Hülse in die Halterung des Lötkolbenhalters stecken. Eine kalte Hülse nehmen und mit der gewünschten Lötspitze bestücken und aufschrauben.

Es sollte immer eine möglichst breite Spitze verwendet werden, da diese einen besseren Wärmeübergang schafft als eine schmale Spitze. Eine große Spitze hat auch eine höhere Wärmekapazität als eine kleine Spitze. Für das Stone-Board ist eine Spitze der Größe (4) bis (6) perfekt.

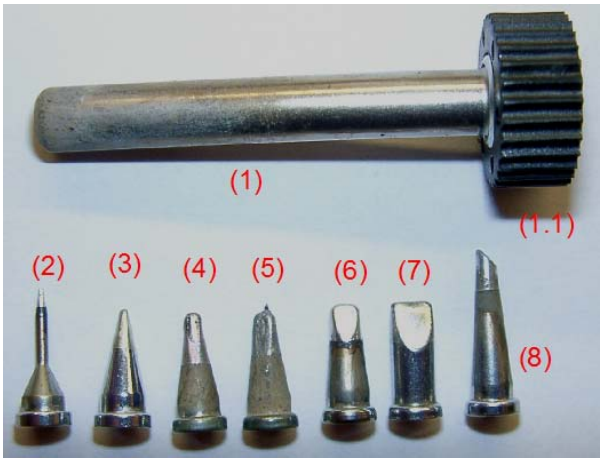


Abbildung 4: Lötspitzen und Hülse

Die Spitze der Größe (2) ist auf keinen Fall zu verwenden, diese eignet sich nur für Fine-Pitch-SMD-Anwendungen oder schwer zugängliche Stellen auf Platinen.

Die Temperatur sollte auf 325 - 350°C ein gestellt werden. Sollte sich eine Lötstelle mit der Temperatur nicht gut löten lassen **muss** eine dickere Spitze verwendet und **nicht** die Temperatur erhöht werden !

Um die Spitze von Flussmittelresten oder Lötzinn zu reinigen, befindet sich im Lötkolbenhalter ein Schwamm, dieser sollte leicht feucht (nicht „schwimmen“, nicht trocken) sein.

Die Lötspitzen dürfen auf keinen Fall mit Feile oder Sandpapier gereinigt werden, das führt zur Zerstörung der Oberflächenbehandlung und damit der Spitze.

Löten von THT-Bauteilen

THT (**T**hrough **H**ole **T**echnology)-Bauteile oder auch „bedrahtete Bauelemente“ genannt, lassen sich sehr einfach verlöten.

Erst muss das Bauteil auf das richtige Rastermaß gebogen werden. Dazu nimmt man am einfachsten die Biege vorrichtung. Nun wird das Bauteil auf die Platine gesteckt und mit Tesakrepp oder dem Finger (kann heiß werden) fixiert.

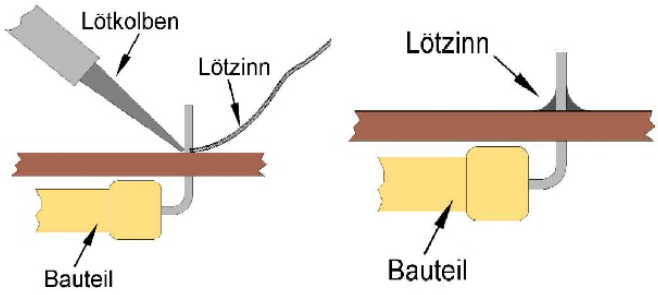


Abbildung 5: Quelle:www.stayathome.ch

Nun wird eine Seite angelötet. Dazu nimmt man den Lötkolben und setzt diesen so an, dass sowohl das Bauteil als auch das Pad von der Spitze berührt werden. Nun führt man Lötzinn von der Seite zu. Dieses sollte nun sowohl auf dem Pad wie auch an dem Bauteilpin gleichmäßig fließen und wie in Abbildung 5 rechts hohlkehlig verlaufen. Die ganze Aktion sollte mit ewas Übung nicht mehr als zwei bis drei Sekunden dauern. Nun wird der gleiche Vorgang an den anderen Pins des Bauteils wiederholt.

Eine gute Lötstelle sollte folgende Eigenschaften haben:

- Glänzende Oberfläche
- gleichmäßiger Verlauft des Lötzinns über das Pad und das Bauteilbein

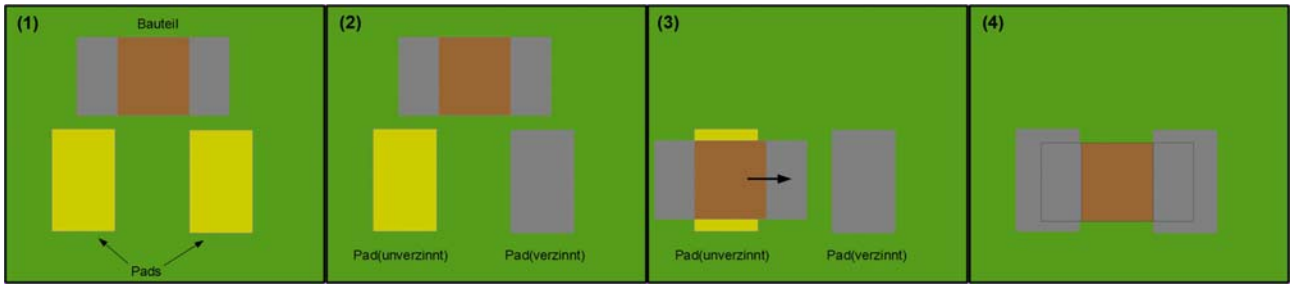
Sollte die Oberfläche nicht gläzend oder der Verlauft nicht gut sein, kann die Lötstelle nochmals erhitzt und neues Zinn zugefügt werden. Alternativ kann mit dem Flussmittelstift neues Flussmittel auf die Lötstelle aufgetragen und diese nochmalig erhitzt werden.

Wenn es gar nicht klappt

- Größere Spitze verwenden, da das Bauteil oder die Platine zuviel Hitze entzieht
- Bauteil und Platine reinigen (Glasfaser-Pinsel, Platinenradier)
- Flussmittel auftragen um ggf. aufgetretene Oxidationen zu entfernen.
- Falsche Temperatur

Löten von SMD-Bauteilen

Das Löten von SMD (**S**urface **M**ounted **D**evice)-Bauteilen ist für Anfänger etwas anstruchsvoller als von THT-Bauteilen, aber mit etwas Übung geht es sogar deutlich schneller, da z. B. das Biegen und Ablängen der Bauteile wegfällt.



Am Beispiel eines Kondesators soll das Vorgehen beim SMD-Löten gezeigt werden. Bei nahezu allen handlötbaren SMD-Bauformen ist das vorgehen das gleiche, sei es nun ein SOT-23-Transistor oder ein SO-16-IC.

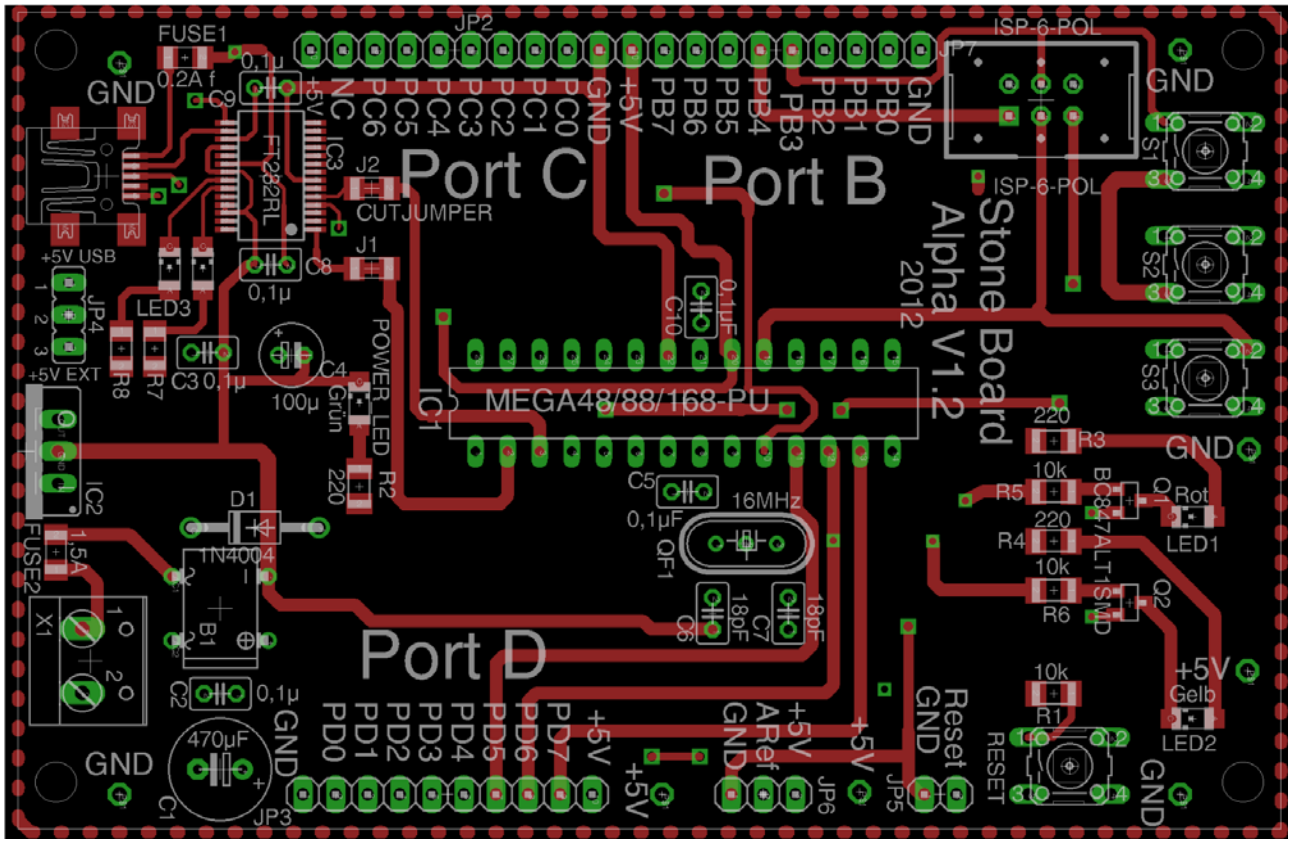
(1) Bauteil in die Nähe der Lötstelle legen

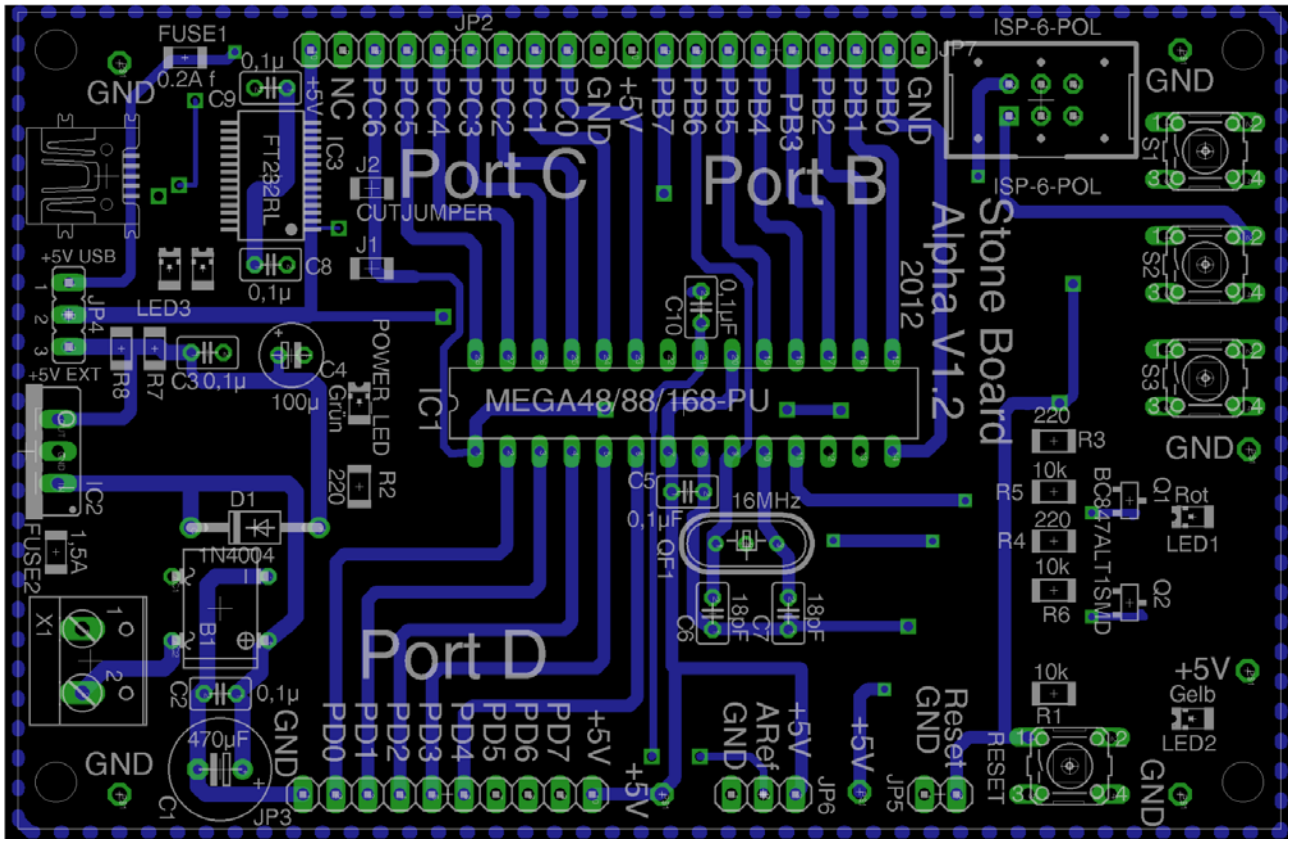
(2) Ein Pad (“Fixier-Pad”) verzinnen, dazu mit dem Lötkolben das Pad erwärmen und Lötzinn zuführen.

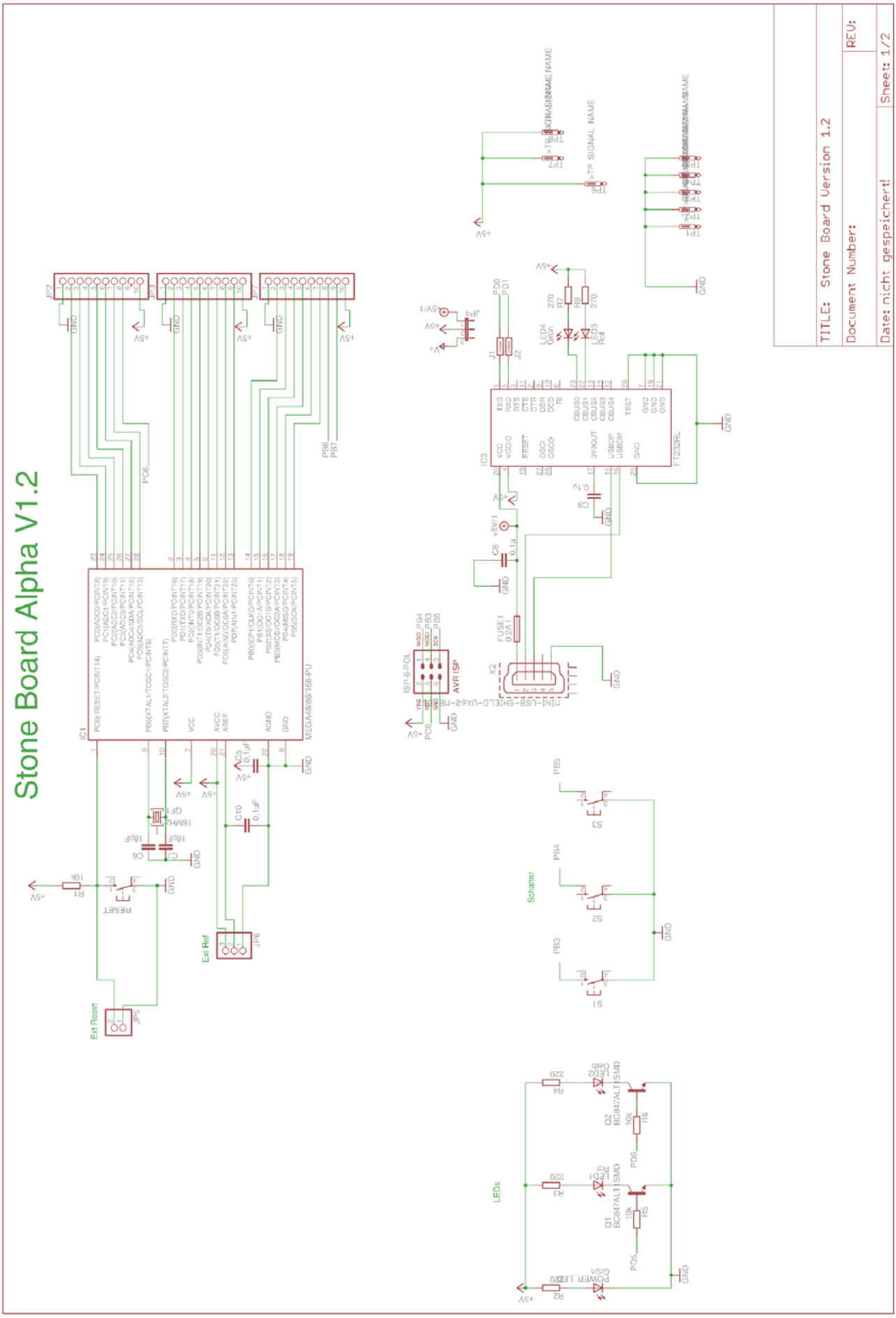
(3) Das Bauteil mit SMD-Pinzette greifen. Dann das verzinnte Pad wieder erhitzen bis das Lötzinn schmilzt und das Bauteil auf das Pad schieben und positionieren. Nun den Lötkolben vom Pad nehmen und warten bis das Lötzinn erstart ist. Sollte man mit der Position des Bauteils nicht zufrieden sein, Pad erneut erhitzen und neu ausrichten.

(4) Nun alle Pads anlöten, dazu mit dem Lötkolben Pad und Bauteil-Pad berühren und Lötzinn zugeben. **Als Abschluss das “Fixier-Pad” nochmal nachlöten. Entweder durch das Erhizten und Zugeben von neuem Lötzinn oder durch Auftragen von Flussmittel und Erhitzen des Pads.**

Platinenlayout






Stone Board Alpha V1.2
TITLE: Stone Board Version 1.2
Document Number:
REV:
Date: nicht gespeichert!
Sheet: 1/2
MEGA48/88/168-PU
FT232RL
MINI-USB-SHIELD-UX60-MB
AVR ISP
ISP-6-PCB
Schalter
LEDs
Ext Ref
Ext Reset
RESET
16MHz
+5V
GND