Praktikum Informatik 1

PIN1 Aufgabe Stopuhr

Kneißl/ Mandl - FH Regensburg

April 2008

Fachhochschule Regensburg
Fakultät Elektro und Informationstechnik

Inhaltsverzeichnis

7. Stopuhr

In dieser Praktikumseinheit wird wieder ein Mikroprozessor-Evaluations-Board benutzt. Wir programmieren das Board so, dass es als Stopuhr funktioniert.

Im Gegensatz zur Aufgabe *EvalBoard* ist die Baugruppe aber nicht emuliert, sondern real.

Wir verwenden ein Testboard mit dem Mikroprozessor ATMEGA32 der Firma Atmel. Auf der Schaltung befinden sich ein LCD-Display, Leuchtdioden, Taster und ein Potentiometer für Eingaben über einen Analog-Digital-Konverter (ADC).

Die Baugruppe kann über USB mit einem PC verbunden werden und wird so mit Strom versorgt. Auf dem PC editieren und übersetzen wir auch das C-Programm, das der Mikroprozessor abarbeiten soll. Es lässt sich über die USB-Schnittstelle auf die Baugruppe laden.

7.1 Prozessor und Evaluation Board

Dokumentation zum Prozessor ATMEGA32 und dem Evaluation-Board liegt auf `K:\Mar\IN1\Atmel\documentation\`. Dort liegt auch der Schaltplan aus Abbildung 7.3 auf Seite 12 als `.pdf`-Datei.

7.1.1 Ports

Die vier 8-Bit-Ports vom ATMEGA32-Prozessor werden mit Buchstaben A-D bezeichnet.

Im Programm verwenden wir nur die Ports A und B. In der Schaltung aus Bild 7.3 ist zu sehen, welche Pins von Port A mit den Tastern verbunden sind und dass die Leds an Pins von Port B angeschlossen sind.

Port-Register

Für jede Bitposition in den Ports ist individuell konfigurierbar, ob sie als Ein- oder Ausgang arbeiten soll. Daher gibt es für jeden Port x mit $x \in \{A, B, C, D\}$ jeweils drei 8-Bit-Register:

PORTx	wird benutzt, um Ausgänge mit Signalen zu belegen.
PINx	wird benutzt, um Eingänge ins Programm zu übernehmen.
DDRx	wird benutzt, um für jedes Bit festzulegen, ob Aus- oder Eingang.

Ausgänge setzt man im Programm durch Wertzuweisung an `PORTx`, z.B.

```
PORTB = schalter;
```

Eingänge überträgt man ins Programm durch Zugriff auf `PINx`, z.B.

```
schalter = PINA;
```

Zur ***Konfiguration*** in Aus- Und Eingänge setzt man `DDRx`, z.B.

```
DDRA = 0xe0;
```

durch eine 1 wird ein Ausgang konfiguriert.

7.1.2 Taster, Leds

Für Eingabepins gibt es eine zusätzliche Konfigurationsmöglichkeit durch setzen der zugehörigen Bits in `PORTx`. Mit dem Wert 1 bekommt man für den Eingang einen „Pull Up"-Widerstand auf dem Chip nach VCC geschaltet. Das bedeutet, dass bei offenem Pin eine

1 ansteht und man durch Verbindung mit Masse eine 0 erzeugen kann. Mit anderen Worten:
Die **Taster arbeiten invers**: Nicht gedrückt ≅ 1 und gedrückt ≅ 0.

Übung:

Versuchen Sie das folgende Programmstück aus `Hilfs.c` für die Initialisierung der Ports mit der Schaltung in Bild 7.3 in Einklang zu bringen:

```
// configure port a
PORTA = 0x1e; // enable switch pullups PA1..PA4
DDRA = 0xe0; // RS,E,R/W out, PA1..PA4 in (switches), PA0 in (ADC)

// configure port b
PORTB = 0x00;
DDRB = 0xff; // all bits out (Leds 0..7)
```

Sie sollten jetzt auch erklären können, warum die Schalter im Demo-Programm mit

```
schalter = ~PINA&0x1e;
```

übernommen werden und vor der Anzeige geshiftet werden:

```
// Anzeige im lcd display
schalter = schalter>>1;
```

7.1.3 Analog-Digital-Converter (ADC)

Der ATMEGA32 enthält einen Analog-Digital-Converter, über den Analoge Werte eingegeben und als Zahlenwerte im Programm verfügbar sind.

Wie in Abb. 7.3 ersichtlich, ist er auf dem Evaluation Board über Port A, Pin0 mit einem Drehpotentiometer beschaltet.

Weil die Ansteuerung etwas komplizierter ist, als bei den Tastern, besorgen wir Werte vom ADC nicht durch direkte Portzugriffe, sondern mit Hilfe eines Hilfsprogramms, das im Programmrahmen (s. Kap. „Vorbereiteter Programmrahmen“ unten). enthalten ist.

7.1.4 LCD-Anzeige

Auf dem Evaluation Board befindet sich eine zweizeilige LCD-Anzeige. In jeder Zeile können 16 Zeichen dargestellt werden.

Weil die Ansteuerung etwas komplizierter ist, als bei den LEDs, geben wir Werte nicht durch direkte Portzugriffe auf die Anzeige aus, sondern mit Hilfe eines Hilfsprogramms, das im Programmrahmen (s. Kap. „Vorbereiteter Programmrahmen“ unten). enthalten ist.

7.2 Vorbereiteter Programmrahmen

Um Ihnen die Programmierung zu erleichtern, gibt es einen Programmrahmen, der sich in der Entwicklungsumgebung übersetzen und starten lässt. Sie können diesen Rahmen als Ausgangsmaterial benutzen und gemäß Aufgabenstellung ändern und ergänzen.

Folgende Dateien sind enthalten und bilden das AVR-Studio-Projekt für diese Praktikumseinheit:

Hilfs.c	Unterprogramme für Initialisierung, Ausgabe auf LCD, Eingabe von Analogwerten über AD-Wandler
Hilfs.h	Headerfile für die Verwendung der Unterprogramme aus Hilfs.c damit sind auch die Adressen PORTA, PINA etc. deklariert
Main.c	Hauptprogramm mit Demo, das Sie ausbauen können

7.2.1 Aufrufe

Initialisierung

Die Komplette Initialisierung des ATMEGA32 einschließlich Ports und der LCD-Anzeige erledigt ein Aufruf

```
initAll();
```

LCD-Anzeige

die zweizeilige Anzeige kann man mit

```
LCD_clear();
```

löschen.

Eine Zeichenkette lässt sich mit einem Aufruf

```
lcdPrint(zeile, spalte, Text);
```

ausgeben.
Als Zeilennummern sind 0 und 1 zulässig, gültige Spaltennummern sind 0 bis 15.

Text ist eine Zeichenkette, *keine Interndarstellung*, sondern Text. Wenn man also Zahlen in Interndarstellung ausgeben will, benötigt man einen Textpuffer, z.B.

```
char text[128];
```

und muss die Interndarstellung vor der Ausgabe erst einmal in Text umwandeln, z.B.

```
sprintf(text, "%4d", schalter);
lcdPrint(1, 3, text);
```

`sprintf` arbeitet ähnlich wie `printf`, gibt aber den Text nicht in die Ausgabe (z.B. in ein Fenster), sondern legt den Text als Zeichenkette in den Zeichenvektor, der als 1. Parameter (hier `text`) angegeben wurde.

Generell zu beachten ist die langsame Ausgabe für die LCD-Anzeige. Das bedeutet insbesondere, dass Aufrufe wie `LCD_clear();` nicht in eine Schleife gehören, sondern so selten wie möglich aufgerufen werden sollten, wenn man Flimmern vermeiden will.

Am Besten gibt man immer nur so viele Zeichen aus, wie zur Aktualisierung der Anzeige nötig sind.

Verzögerung

Mit den folgenden Aufrufen lässt sich die Abarbeitung des Programms um `anz` Sekunden/ Millisekunden/ Mikrosekunden verzögern.

```
delay_s(anz);
delay_ms(anz);
delay_us(anz);
```

ADC-Eingabe

Über das Potentiometer lässt sich einstellen, welche Spannung an Pin 0, Port A anliegt. Mit einem Aufruf

```
adcWert = getAdc();
```

lässt sich der `unsigned short`-Variablen `adcWert` ein Wert zuweisen, der der angelegten Spannung entspricht. Die Auflösung ist 10 Bit, d.h. es werden Werte zwischen 0 und 1023 geliefert, je nach Stellung des Potentiometers.

7.2.2 Hauptprogramm, Main.c

In `Main.c` wird exemplarisch gezeigt, wie man das Board initialisiert, wie man Taster ausliest, LEDs setzt, Analogwerte einliest und Text in die LCD-Anzeige ausgibt.

Polling-Betrieb

In dieser Praktikumseinheit betreiben wir den Prozessor nicht im Interrupt-, sondern im Polling-Betrieb. Das Hauptprogramm enthält eine Endlosschleife, in der die Eingänge zyklisch abgefragt werden. Dabei wird nicht auf eine Veränderung der Eingänge „gewartet“, sondern der aktuelle Zustand wird bei jedem Schleifendurchgang erneut gelesen.

7.3 Entwicklungsumgebung

Auf dem PC arbeiten wir mit der grafischen Entwicklungsumgebung *AVR Studio*, die einen Cross-Compiler enthält, der auf dem PC läuft, aber Code für den Mikroprozessor erzeugt.

7.3.1 Installation der Entwicklungsumgebung

Die zu installierende Software findet man auf `K:\Mar\IN1\Atmel\Software\`. Unter `K:\Mar\IN1\Atmel\avrusbeval\hidboot\bootloadHID.exe` gibt es das Programm `bootloadHID.exe` zum Hochladen von Kompilaten auf das Board.
Zu den original-Webseiten zum Herunterladen kommt man über

- `http://winavr.sourceforge.net/`
  WINAVR (C-Compiler für den Atmel Controller)
- `http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725`
  AVR Studio (Entwicklungsumgebung mit Assembler und Simulator, bindet WINAVR ein)

Installationsreihenfolge

Zuerst installiert man den C-Compiler *WINAVR*, anschließend die Entwicklungsumgebung *AVR Studio*, die den Compiler voraussetzt.

7.3.2 Projekt einrichten

Erstellen Sie ein geeignetes Verzeichnis und kopieren Sie die Dateien des vorbereiteten Programmrahmens `Hilfs.c`, `Hilfs.h` und `Main.c` hinein.

Neues Projekt erstellen

Im AVR Studio brauchen wir ein Projekt mit Typ *AVR GCC*, wobei man den Haken bei *Create initial File* entfernt, weil die Datei `Main.c` schon existiert.

> Der Name des Projekts darf nicht mit `Stop` beginnen, weil man sonst Build-Errors bekommt

Die benötigte Plattform ist *AVR Simulator*, Prozessor *ATMEGA32*.

Konfiguration einstellen

Über *Project/ Configuration Options* stellt man wie in Abb. 7.1 die Frequenz 1.2MHz ein und schaltet die Optimierungen ab (*-O0*).

Dateien hinzufügen und Übersetzen

Zum Projekt fügt man die `.c` und `.h` Dateien hinzu (Rechte Maustaste auf *Source Files/ Add Existing Source Files* bzw. *Header Files/ Add Existing Header Files*). Das Projekt sollte sich jetzt mit *Build* übersetzen lassen.

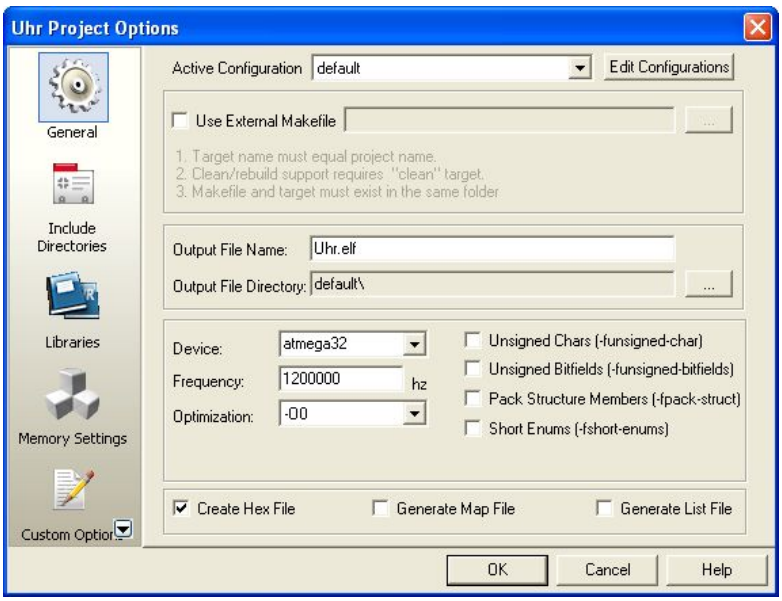

Abbildung 7.1: Konfigurationseinstellungen

7.3.3 Hochladen und Starten von kompilierten C-Programmen

Hochladen

Dazu muss man die Baugruppe in den Programmiermodus bringen. Das erreicht man durch gleichzeitiges Drücken von *Reset* und *Boot* und Loslassen von *Reset* bei noch gedrückter *Boot* Taste. Den Programmiermodus erkennt man am Leuchten der Boot-Led.

Mit Hilfe des Progamms `bootloadHID.exe` kann man dann die erzeugte `.hex`-Datei auf die Baugruppe laden. z.B. gibt man dazu in der DOS-Box das Kommando `bootloadHID.exe Demo.hex` ein. Dabei müssen sich die `.exe` und die `.hex`-Datei im aktuellen Verzeichnis der DOS-Box befinden.

Alternativ kann man durch *Tools/ Customize/ Tools/ New* (siehe Bild 7.2) einen Menüpunkt für das Tools-Menü der Entwicklungsumgebung definieren, der das Hochladen startet.

Starten

Nach dem Hochladen drückt man zum Starten die Taste *Reset* auf der Baugruppe.

7.4 Durchzuführende Vorversuche

In diesem Abschnitt werden einige Vorversuche durchgeführt. Die Vorversuche sollen nach Erstellung *nicht* aus der Quelldatei gelöscht werden, sondern mit

```
#ifdef ...
  ...
#endif
```

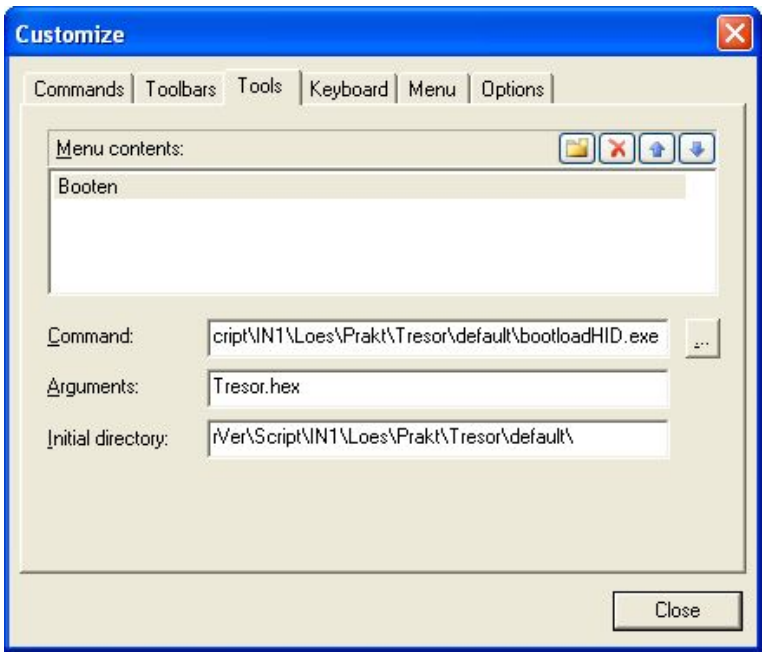

Abbildung 7.2: Boot Menüeintrag erzeugen

geklammert werden, so dass sie auch nach Weiterentwicklung des Programmstandes vorgeführt werden können.

Kommentieren Sie nach einem ersten Test der Demo `#define _DEMO` aus!

7.4.1 LCD

Geben Sie in Zeile 0, Spalte 5 „Test“ aus und sehen Sie sich das Ergebnis an!

Fügen Sie einen Löschaufruf für die Anzeige ein. Sehen Sie sich den Unterschied an, wenn der Löschaufruf

1. Vor der Schleife
2. In der Schleife

steht!

7.4.2 ADC

Kopieren Sie das Programmstück aus dem Demo-Teil in die Schleife, das einen ADC-Wert besorgt und im LCD-Display anzeigt. Überzeugen Sie sich vom Wertebereich durch Drehen des Potentiometers.

Anzeigeformat

Ändern Sie jetzt das Format für den `sprintf`-Aufruf von `%4d` in `%d`. Klären Sie, warum die Ausgabe jetzt falsche Zahlen anzeigt!

7.4.3 Polling Betrieb

In diesem Versuch wird Eingabe im Polling-Betrieb untersucht.
Definieren Sie eine Zählvariable. Gestalten Sie die Endlos-Schleife im Hauptprogramm so um, dass

- der Zählerstand bei jedem Durchlauf erhöht wird, wenn der Taster „PA1“ gedrückt ist
- der Zählerstand unverändert bleibt, wenn alle Taster ungedrückt sind
- der Zähler auf 0 gesetzt wird, wenn irgendeine andere Tastenkombination gedrückt ist.

Der Wert der Zählvariablen ist auf dem LCD-Display anzuzeigen.

Hier können Sie auch die Verzögerungs-Funktionen testen, und z.B. immer 250 Millisekunden warten, bevor die Zählvariable erhöht wird.

7.5 Stopuhr

7.5.1 Einfache Stopuhr

Unsere Stopuhr soll nun mit der Taste PA1 gestartet und mit PA2 angehalten werden können (Pause). PA3 dient zum Rücksetzen der Stopuhr.

Der Schleifendurchlauf soll nicht mehr als 250ms verzögert werden, damit die Tastereingaben auch bei kurzem Druck erkannt werden.

Die Anzeige soll am LCD Display in Sekunden erfolgen.

7.5.2 Zwischenzeit erfassen

PA4 soll die Anzeige anhalten und die Zwischenzeit anzeigen. Während der Zwischenzeitanzeige soll aber die Zeit weiter unsichtbar mitgezählt werden.

Damit man diesen Zustand vom Pausenzustand (mit PA2) unterscheiden kann, soll eine LED blinken.

Wenn die Start-Taste gedrückt wird, soll die Zeit wieder normal angezeigt werden.

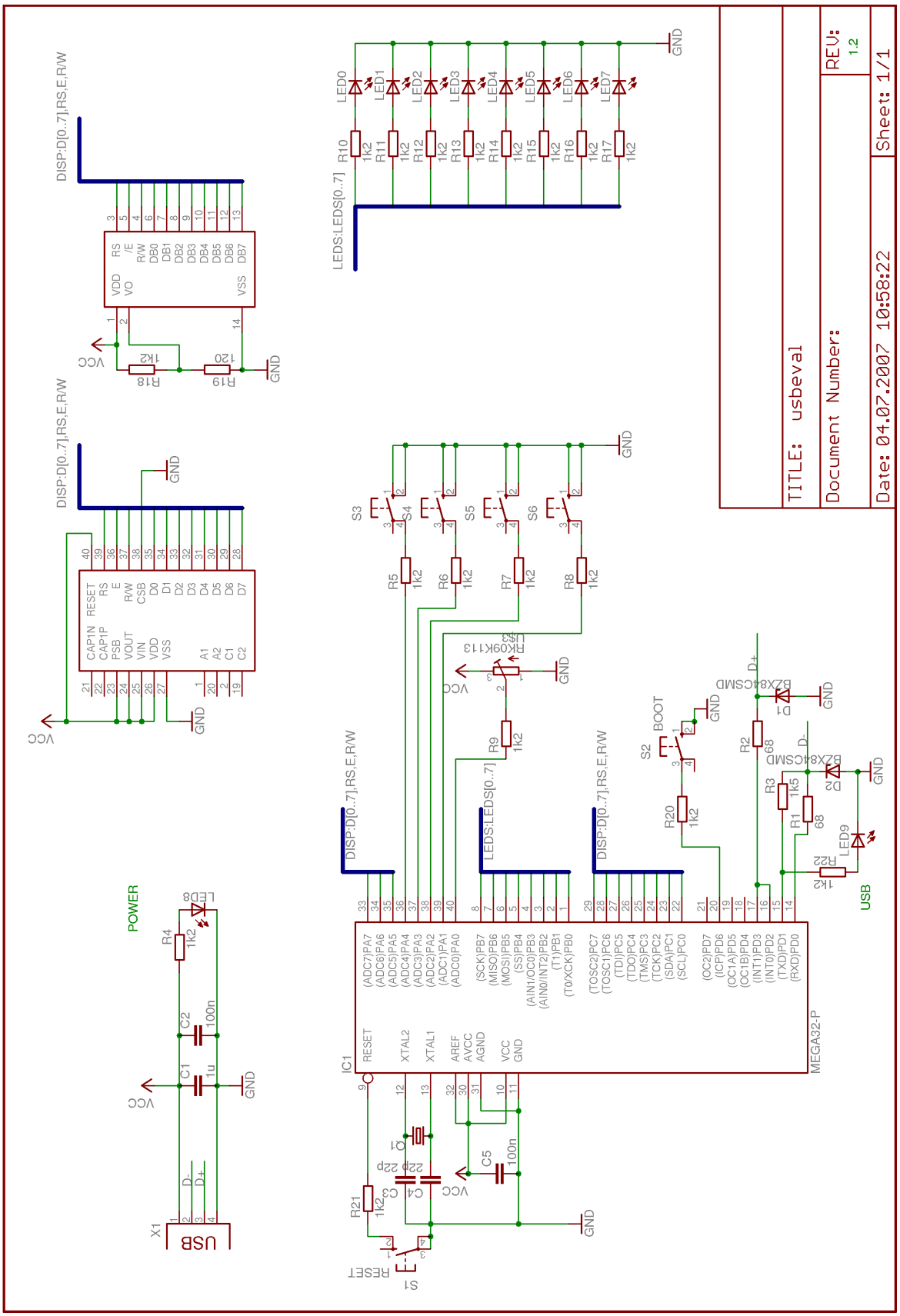

Abbildung 7.3: Evaluation Board: Schaltung