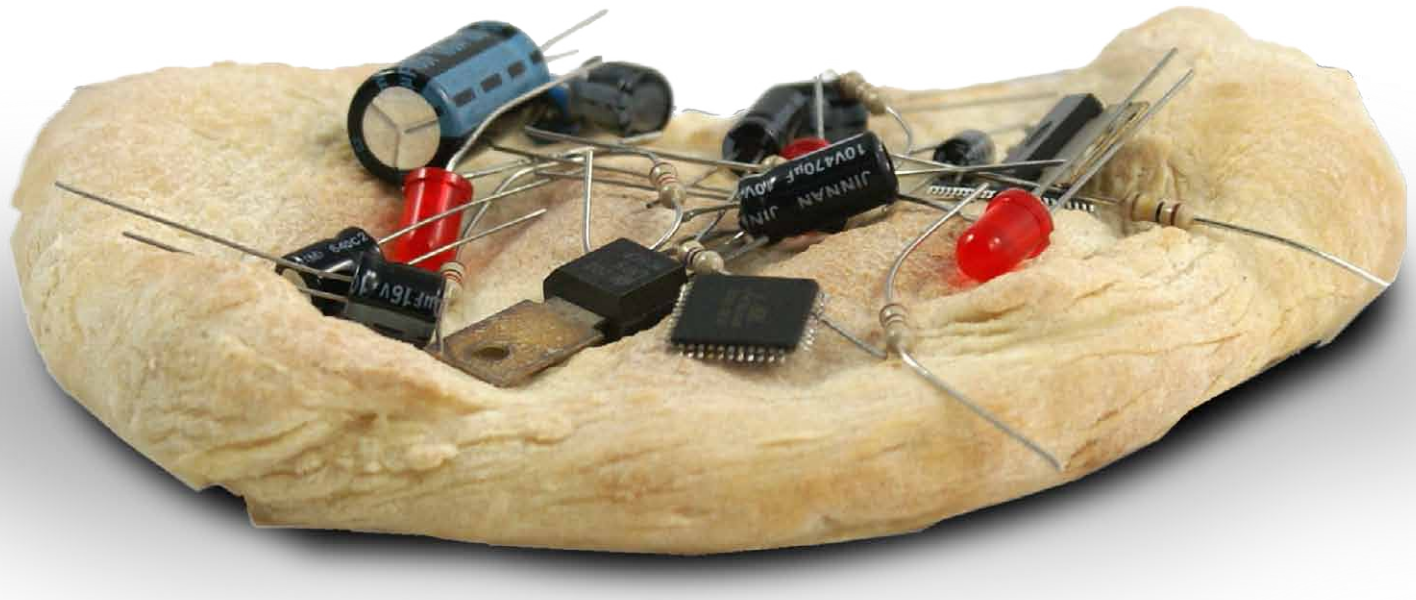
embedded-projects.net
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

PIZZA 2.0

Eine Open-Source Zeitschrift
zum Mitmachen!



[PROJECTS]

- Versenden von SMS mittels Mobiltelefon
- Einführung in den Kalman-Filter
- Bau einer günstigen CNC-Fräse
- fastboot Howto
- Rechner für Embedded Linux
- Web-basiertes Messen

Pizza 2.0

Der Neubeginn der aktiven Community Rund um das Journal!

Embedded Projects Journal - Issue 8

Pizza 2.0! Jeder kann mitmachen, jeder kann sie haben!Lecker und frisch, spannend und geladen! Mit frischen Mikcrocontroller und mehr für einen satten Belag.

Open-Source lebt von der Leidenschaft der Aktivisiten und der Möglichkeit aktiv mitzugestalten.

Endlich ist es soweit: Die passende Homepage für das embedded projects Journal ist online!

http://journal.embedded-projects.net

Die Seite ist eine Anlaufstelle, ein Archiv und eine Plattform für Kommunikation.

Die wichtigsten Features sind:

- Lieferadresse verwalten, bzw. Aboart auswählen (PDF, Post, Spendenabo)
- Artikel online schreiben (Autoren, ...)
- Einfach zu bedienener Flohmarkt für Mikrocontroller und Zubehör
- Kostenfreier Stellenmarkt für Firmen
- Branchenverzeichnis für Firmen

Im Wesentlichen erhoffen wir uns durch die Webseite, dass sich noch mehr Leute an der Arbeit für das Heft beteiligen. Aktuell gibt es beispielsweise noch viel zu wenig Helfer in meiner Kartei, die das Heft vor dem Druck nocheinmal kritisch durchforsten, um grobe Fehler zu entdecken.

Eine kleines Team ermöglicht so, dass Niveau auf ein sehr gutes Level zu bringen und zu halten. Schreibt mir einfach eine E-Mail, wenn Ihr Lust und Interesse habt, eich aktiv am Projekt zu beteiligen.

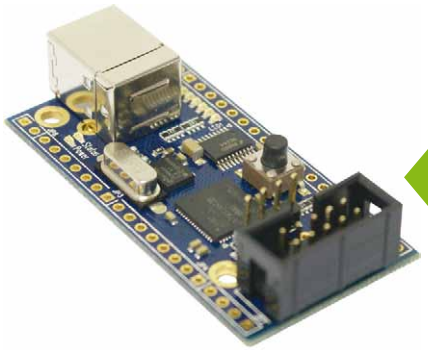
Ebenfalls wird das Titelbild ab jetzt nur noch selbst erstellt. Wenn jemand ein hübsches Bild oder eine gute Idee hat - wir freuen uns über Eure E-Mails.

Benedikt Sauter und das Team
sauter@embedded-projects.net

→shop.embedded-projects.net

HARDWARE FOR YOUR PROJECTS – ONLINESHOP

- großes Sortiment an Evaluations- und Testboards
- bekannte Open-Source-Projekte
- übernommener Artikelbestand von www.mikrocontroller.net
- faire Preise
- Produkte direkt vom Hersteller
- bequeme Zahlungsabwicklung und schneller Versand



Octopus USB IO-Schnittstelle (Octopus USB CAN)

- günstige Hardware für Zugriff von PC auf IO-Ports und Mikrocontrollerschnittstellen
- 38 x Digital Ein- Ausgabeports
- 8 x Analogeingänge, 2 x PWM, 1 x SPI
- 1 x I2C, 2 x UART, 1 x CAN
- Umfangreiche API mit Beispielen in vielen Sprachen vorhanden (C, C++, Java, Python, Ruby,...)
- Hardware, Firmware, Bibliothek und Beispiele komplett Open-Source

Best.-Nr.: 700008

Aktionspreis
€ 39,90*
inkl. 19 % MwSt.
zzgl. Versand

Einstieg in die Embedded Linux Welt mit dem Grasshopper

- Bekannt aus mikrocontroller.net und dem embedded projects Journal
- Sofort Einsatzbereit und fertig installiert
- Mit Anwendungsbeispielen: z.B. IOs schalten
- Open Source Entwicklungsumgebung
- 140MHz max. 200 MHz
- 64 MB SDRAM
- 8 MB Flash
- 10/100 MBit/s Netzwerk
- 1 USB Highspeed Device Anschluss
- 8 LEDs
- 1 Taster

Best.-Nr.: 700009

€ 85,00*
inkl. 19 % MwSt.
zzgl. Versand

OpenOCD USB Adapter

- 2 in 1 –> JTAG und RS232
- Debuggen und flashen von ARM Prozessoren
- USB zu JTAG Adapter für OpenOCD
- Nutzt die Standard 2x10 ARM JTAG Pin-Stecker
- RS232 Anschluss (direkt an SUB-D Stecker oder an Pins für TTL Pegel)

Best.-Nr.: 700004

€ 39,90*
inkl. 19 % MwSt.
zzgl. Versand

* Nur solange der Vorrat reicht!

Holzbachstraße 4, D-86152 Augsburg
Tel +49 (0) 821 279599-0, Fax +49 (0) 821 279599-20
shop@embedded-projects.net

embedded projects GmbH
HARDWARE FOR PROJECTS

Versenden von SMS mittels Mobiltelefon

Dipl.-Ing. (FH) Lars Neubert

Einleitung

Im Forum kommt immer häufiger die Frage „Wie werden Textnachrichten mittels eines handelsüblichen Mobilfunkgerätes versandt?“. Im Rahmen einer Projektarbeit entstand dabei eine recht umfangreiche Bibliothek zu diesem Thema. Diese soll hier nun kurz vorgestellt werden. Die Software beinhaltet unter anderem:

- Reaktion auf einen bestimmten Anrufer
- Versand von Textnachrichten im PDU Format
- Auslesen von Textnachrichten im PDU Format und Decodierung
- Auslesen des aktuellen Datums und der Zeit des Mobiltelefons
- Anzeigen von Text auf dem Display des Mobiltelefons
- Textphrasing von empfangenen Textnachrichten und Ausführen vorher eingestellter Aktionen/Funktionen auf bestimmte Schlüsselwörter.

Die einzelnen Algorithmen sind dabei sehr einfach gehalten. Es gibt sicherlich sehr viel kürzere Varianten gibt, um z. B. einen Text in das PDU Format zu transformieren. Ich habe mich aber lieber auf mehr und damit für andere auch verständlicheren Code beschränkt. Das verbraucht zwar mehr Speicher, trägt aber meines Erachtens zum Verständnis bei.

Hardware

Ich arbeite selber mit einem Siemens SL55 Mobiltelefon. In einschlägigen Auktionshäusern sind diese betagten Mobilfunktelefone sehr günstig zu erwerben und damit ideal zum Experimentieren geeignet. Neben dem Mobilfunktelefon wird natürlich auch ein entsprechendes serielles Datenkabel benötigt. Das Datenkabel wird über Pegelwander direkt mit dem Mobiltelefon und dem Mikrocontroller verbunden. Dadurch ist es möglich, die serielle Schnittstelle des Mobilfunkgerätes anzusprechen.

Als erstes sollte die Verbindung mit einem Computer überprüft werden. Dazu ist das Datenkabel an das Mobilfunkgerät anzuschließen. Mit einem Terminalprogramm wie hterm kann nun

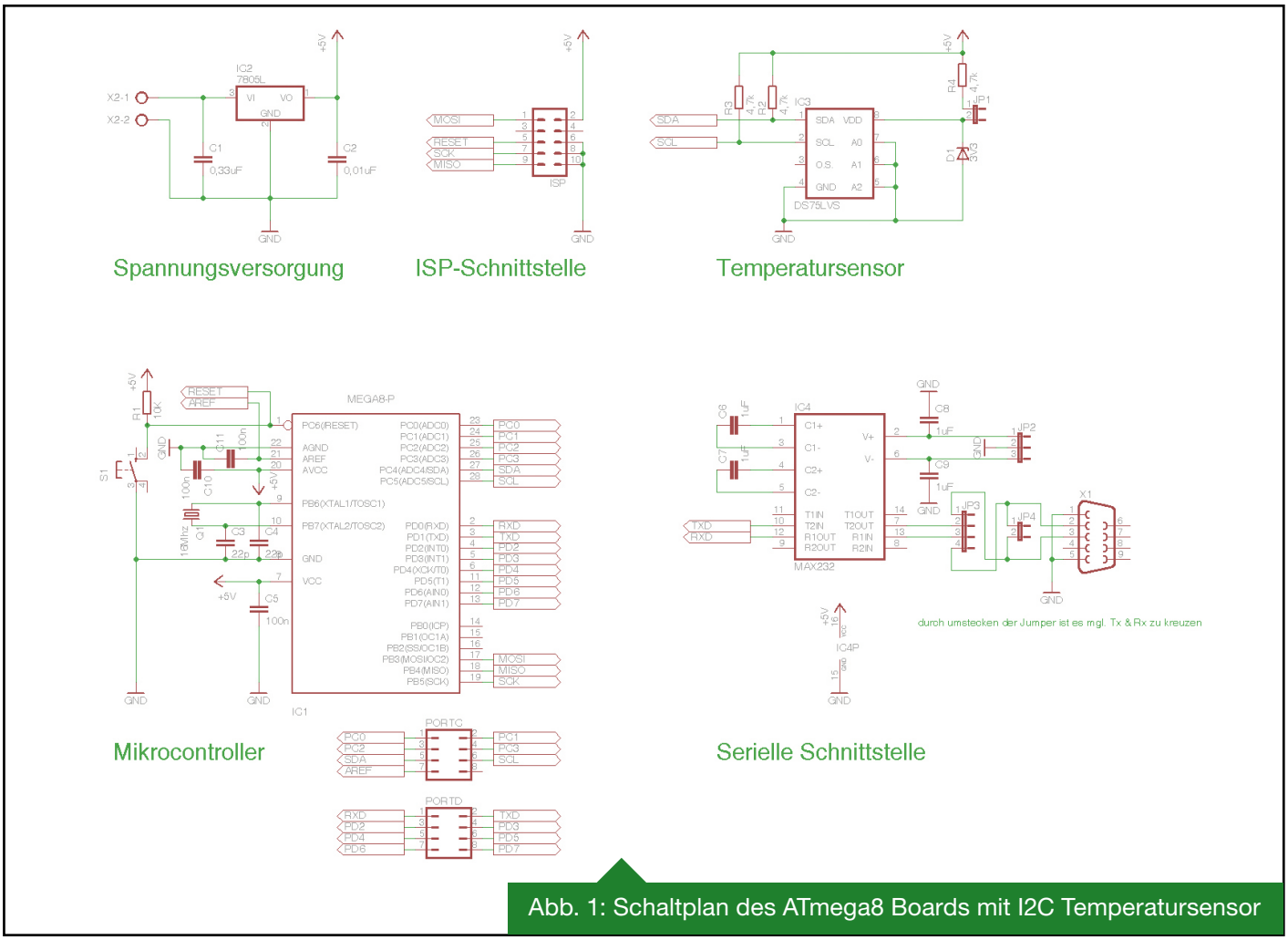


Abb. 1: Schaltplan des ATmega8 Boards mit I2C Temperatursensor

eine Datenübertragung erfolgen. Dazu muss die serielle Schnittstelle entsprechen eingestellt werden. Bei dem SL55 ist eine Baudrate von 19200 im 8N1 Modus erforderlich.

Bei der Kommunikation mit einem Mikrocontroller ist darauf zu achten, dass die meisten Datenkabel einen Pegelwandler enthalten. Daher müssen die Pegel des Mobilfunkgerätes (meist High-Pegel 3,3V) auf die Pegel der seriellen Schnittstelle angepasst werden (High-Pegel -15V). Ein Mikrocontroller arbeitet dagegen mit Spannungen von 5V bzw. 3,3V. Die Anpassung geschieht mittels eines Pegelwandlers wie dem MAX232. Dabei ist zu beachten, dass im Kommunikationsweg nun 2 Pegelwandler vorhanden sind (Datenkabel und am Mikrocontroller). Für eine Verbindung mit einem Mobilfunkgerät werden die Leitungen Rx und Tx daher gekreuzt. Dieser Umstand hat mir am Anfang relativ viel Zeit gekostet, da eine einfache Kommunikation mit einem Computer 1:1 geschehen kann. Es muss also zwischen Datenkabel und Mikrocontroller noch ein gekreuztes serielles Kabel dazwischen geschaltet werden.

Eine weitere Möglichkeit ist natürlich der direkte Anschluss der seriellen Schnittstelle des Mikrocontrollers am Mobiltelefon. Dabei ist zu beachten, dass im Fall einer Spannungsversorgung des Mikrocontrollers von 5V Pegelwandler für die Leitungen Tx und Rx nötig sind. Das können im einfachsten Fall 3,3V Z-Dioden sein.

Ich habe mich dafür entschieden, gleich eine kleine Platine mit einem ATmega8 anzufertigen. Dabei hab ich darauf geachtet, dass ich über Jumper die Leitungen Tx und Rx 1:1 bzw. gekreuzt schalten kann. Dadurch kann ich ohne zusätzliches gekreuztes

ASCII Zeichen	H	a	l	l	o		W	e	l	t
8 Bit Binär	01001000	01100001	01101100	01101100	01101111	00100000	01010111	01100101	01101100	01110100
7 Bit Binär	1001000	1100001	1101100	1101100	1101111	0100000	1010111	1100101	1101100	1110100
8 Binär PDU	11001000	00110000	10011011	11111101	00000110	01011101	11001011		01101100	00111010
8 Bit Hex PDU	C8	30	9B	FD	06	5D	CB		6C	3A

Abb. 3: Umwandeln einer Zeichenkette in PDU Format

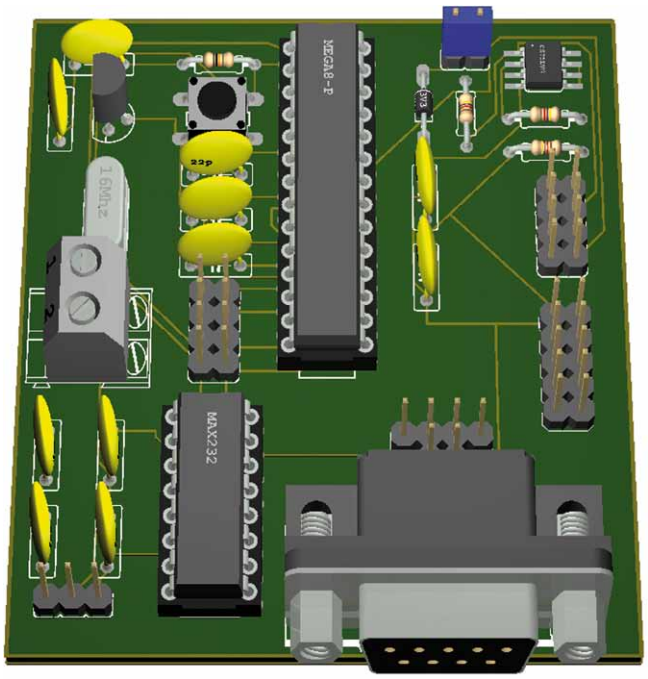


Abb. 2: 3D Ansicht des ATmega8 Boards

serielles Kabel eine Kommunikation mit dem Mobilfunkgerät aufbauen. Die aktuelle Software passt nicht mehr in einen ATmega8. Ich arbeite aktuell mit einem ATmega32. Mit ein paar Anpassungen sollte es aber problemlos möglich sein, den Code in der Größe stark zu reduzieren. So können z.B. die Bibliotheken zur Ansteuerung des Temperatursensors entfallen (twi.c, twi.h, ds75lv.c und ds75lv.h). Alternativ dazu wäre auch die Verwendung der pinkompatiblen Controller ATmega168 bzw. ATmega328 denkbar. Diese haben 16KiB bzw. 32KiB Flash und sind unwesentlich teurer als der ATmega8.

Software

Allgemein wird bei dem Softwarekonzept eine Mischung aus Interrupt und Polling genutzt. So werden z.B. Nachrichten vom Mobilfunkgerät über die serielle Schnittstelle des Mikrocontroller mittels UART Receive Interrupt empfangen, das Senden von Befehlen erfolgt aber mittels Polling. Wie der Datenaustausch mit der seriellen Schnittstelle des Mikrocontrollers funktioniert, ist in einem anderen Artikel nachzulesen (siehe Der UART) und soll an dieser Stelle vorausgesetzt werden. Für die einfachere Arbeit mit der seriellen Schnittstelle habe ich die Low-Level Funktionen auf die Standard Ein- und Ausgabe umgeleitet. Dadurch können die allseits bekannten Funktionen wie printf oder scanf für eine komfortable Kommunikation genutzt werden. Das vergrößert zwar den Code, ist aber dank einfacherer formatierten Ein- und Ausgabe mehr als gerechtfertigt.

c AT-Kommandos sind spezielle Steuerkommandos, welche es ermöglichen das Mobilfunkgerät zu steuern. Für jedes Mobilfunkgerät gibt es einen AT-Command Set in dem alle relevaten Kommandos erklärt sind. Diese Kommandos sollten vorab mit einem Terminalprogramm wie hterm am Computer getestet werden. Von Mobilfunkgerät zu Mobilfunkgerät kann es bei den einzelnen Kommandos kleinere Unterschiede geben, so dass evtl. auch der Quellcode angepasst werden müsste.

Das PDU Format

Viele Mobilfunkgeräte können nur im Packet Data Unit (PDA) Format Textnachrichten versenden. Bei neueren Mobilfunkgeräten ist es zum Teil möglich, auch im Klartext Nachrichten zu verschicken. Das PDU Format soll an dieser Stelle einmal am Beispiel des AT-Kommandos AT+CMGS zum versenden von Textnachrichten näher erläutert werden.

Eine Kurznachricht mit dem Inhalt „Hallo Welt“ an die Nummer 01621234567 sieht z. B. folgendermaßen aus:

```
AT+CMGS=22<cr>
0001000B811026214365F700000AC8309BFD065DCB6C3A
<subst>
```

Die Zeichen <cr> und <subst> sind dabei Steuerzeichen, welche das Mobiltelefon benötigt. Das Steuerzeichen <cr> stellt dabei ein new line (dezimal 10) mit anschließendem carriage return (dezimal 13) dar und <subst> ein so genanntes substitute (dezimal 26). Im Folgenden soll nun einzeln erläutert werden, inwieweit die Nachricht „Hallo Welt“ in das PDU Format kodiert werden kann. Nach dem AT Kommando folgt die Länge der Hex Paare der PDU Zeichenkette ohne die beiden führenden Nullen.

Das heißt, dass im obigen Beispiel 23 Hex Paare an das Mobiltelefon gesendet werden, aber lediglich 22 angegeben werden müssen. Die eigentliche PDU Zeichenkette beginnt mit „00“ und gibt die Länge der Short Massage Service Centre Nummer (SMSC) wieder. Die Längenangabe „00“ bedeutet, dass das Mobiltelefon selbst die eingespeicherte SMSC Nummer wählt. Mit Hilfe des zweiten Hex Paares können verschiedene Optionen eingestellt werden. Die Angabe von „01“ bedeutet, dass die SMS an das SMSC geschickt wird. Das nächste Hex Paar dient der Zuordnung weiterer Nachrichten wie z. B. Berichten und ist die Referenznummer der Nachricht. Die Angabe von „00“ veranlasst dabei das Mobiltelefon dazu, die Referenznummer selbst zu wählen. Als nächstes folgen die Angabe der Länge der Empfängernummer, der Nummerntyp und entsprechend nachfolgend die Nummer. Die Länge der Empfängernummer wird durch das Hex Paar „0B“ angegeben und ist somit 11 Zeichen lang. Das Hex Paar „81“ gibt an, dass die nachfolgende Empfängernummer eine nationale Nummer ist. Bei der Angabe der Empfängernummer ist zu beachten, dass die einzelnen Ziffern der Paare der Nummer vertauscht werden. Aus der Nummer 01 62 12 34 56 7 wird dementsprechend 10 26 21 43 65 F7. Wie im Beispiel zu sehen, werden unvollständige Paare mit „F“ aufgefüllt. Die nächsten beiden Hex Paare sind der so genannte Protocoll Identifier und das Data Coding Scheme. Hier genügt jeweils die Angabe von „00“. Es folgt die Angabe der Länge der zu sendende Nachricht, hier „0A“. Im Anschluss daran wird die eigentliche Nachricht als PDU codierter String angehängt.

Die Vorgehensweise ist dabei in Tabelle 1 dargestellt.

Bei der Umwandlung wird zunächst der ASCII Text in eine 7-Bit Darstellung überführt, indem das erste Bit gelöscht wird. Das bedeutet, dass nicht der gesamte ASCII Zeichensatz zur Verfügung steht (Zeichensatz siehe 3GPP 23.038). Als nächstes wird die 7-Bit Darstellung wiederum in eine 8-Bit Darstellung umgewandelt, indem beim ersten Zeichen das letzte Bit des nachfolgenden Zeichens am Anfang angefügt wird (siehe rote Markierungen in obiger Tabelle). Beim zweiten Zeichen werden entsprechend die letzten beiden Bits des nachfolgendenZeichens angefügt. Dies wird solange fortgeführt, bis die kompletten sieben Bits eines Zeichens an das vorhergehende angefügt worden und der Algorithmus von Anfang an beginnt. Stehen keine Zeichen mehr zur Verfügung, so wird mit Nullen aufgefüllt, bis das Oktett vollständig ist (siehe blaue Markierungen obiger Tabelle). Durch die Umwandlung von 8Bit ASCII Zeichen in eine PDU Zeichenkette konnte im obigen Beispiel ein Zeichen bei der Übertragung eingespart werden. Die zu übertragende Nachricht „Hallo Welt“ wird damit durch „C8309BFD065DCB6C3A“ repräsentiert. Beim zweiten Zeichen werden entsprechend die letzten beiden Bits des nachfolgenden Zeichens angefügt. Dies wird solange fortgeführt, bis die kompletten sieben Bits eines Zeichens an das vorhergehende angefügt worden und der Algorithmus von Anfang an beginnt.

Funktionsbeschreibung

Ich beschränke mich bei der Beschreibung lediglich auf die wesentlichen Funktionen in sms.c. Aus der Quelltextdokumentation sind die Funktionen der anderen Bibliotheken wie die der seriellen Schnittstelle, des Temperatursensors und der I2C-Schnittstellen zu entnehmen. Die Dokumentation und die damit verbundene HTML-Hilfe wird durch Doxygen tags realisiert.

sms_init(void)

Diese Funktion initialisiert die serielle Schnittstelle des Mikrocontrollers, leitet die Low-Level Funktionen der serielle Schnittstelle auf die Standard Ein- und Ausgabe um und nimmt einige Konfigurationen am Mobiltelefon vor. So wird z.B. das Echo beim Senden von AT-Kommandos deaktiviert und der Speicherplatz für empfangene Textnachrichten auf den internen Speicher des Mobiltelefons eingestellt. Zudem wird die new message indication eingestellt. Das bedeutet, dass bei einer eingehenden Textnachricht dies über die serielle Schnittstelle dem Mikrocontroller mitgeteilt wird. Das erspart lästiges Pollen nach neu eingetroffenen Textnachrichten. Des Weiteren werden ebenso die ersten Telefonnummern aus der SIM-Karte ausgelesen und zum späteren Vergleich in einer Datenstruktur abgelegt.

sms_time(unsigned char *year,unsigned char *month, unsigned char *day,unsigned char *hour, unsigned char *minute,unsigned char *second)

Diese Funktion ließt das Datum und die aktuelle Zeit des Mobiltelefons aus. Beispielaufruf:

```
unsigned char year=0;
unsigned char month=0;
unsigned char day=0;
unsigned char hour=0;
unsigned char minute=0;
unsigned char second=0;
sms _ time (&year, &month, &day, &hour,
&minute, &second);
```

sms_send(char* zielnr, char* nachricht)

Diese Funktion sendet ein Textnachricht an eine bestimmte Zielnummer. Beispielaufruf:

```
sms_send(SMS_TELNR,"Hallo Welt");
```

Dabei ist zu beachten, dass SMS_TELNR in der Headerdatei sms.h korrekt angegeben werden muss. Diese Nummer wird ebenso zur Reaktion auf eingehende Anrufe genutzt (siehe sms_state_machine). Der Aufruf kann auch direkt erfolgen:

```
sms_send(„01621234567","Hallo Welt");
```

sms_decode (char* pdu, SMS_DECODE_DATA_T* data)

Diese Funktion dekodiert empfangene Textnachrichten und legt diese in einer Datenstruktur ab. Diese ist folgendermaßen aufgebaut:

Abb. 4

```
/**
* @brief
* Struktur mit Datums- und Zeitangaben einer empfangenen Textnachricht
*/

typedef struct {
uint8_t year;     /**< Jahresangabe YY */
uint8_t month;    /**< Monatsangabe MM */
uint8_t day;      /**< Tagangabe DD */
uint8_t hour;     /**< Stundenangabe hh */
uint8_t minute;   /**< Minutenangabe mm */
uint8_t second;   /**< Sekundenangabe ss */
} DATE_T;

/**
* @brief
* Struktur zum speichern der decodierten Daten aus einer empfangenen Textnachricht
*/

typedef struct {
     char nachricht[SMS_MESSAGE_MAX_LEN+1];      /**< decodierte PDU Nachricht */
     char telnr[SMS_TELNR_MAX_LEN+1];            /**< Telefonnummer des Absenders */
     DATE_T date;                                /**< Struktur mit Datums- und Zeitelementen
                                                 (siehe \ref DATE_T) */
} SMS_DECODE_DATA_T;
```

Der Funktion muss dabei der komplette PDU String des AT-Kommandos AT+CMGR übergeben werden.

```
#define uart_maxstrlen 160
...
SMS_DECODE_DATA_T sms_dec_data;
char sms_tmp[UART_MAXSTRLEN+1]={,0'};
int sms_index=0;
int dummy=0;
...
printf(„AT+CMGR=%i\r\n",sms_index);
gets(sms_tmp); // führendes \r\n verwerfen
scanf(„+CMGR: %i,,%i\r\n%s\r\n",&dummy,&dummy,sms_tmp)
...
sms_decode(sms_tmp,&sms_dec_data);
...
```

sms_display_text(char* nachricht)

Diese Funktion stellt eine Nachricht auf dem Mobiltelefon-Display dar. Beispielaufruf:

```
sms_display_text(„Hallo Welt");
```

SMS_MSG sms_state_machine(void)

Diese Funktion wertet alle einkommenden Daten des Mobiltelefons aus und wird in der main loop fortlaufend ausgeführt. Der Rückgabewert der Funktion entspricht dem zuletzt empfangenen Datensatz.

```
/**
* zusammengefasste Auflistung als Rückgabewert der State Machine (\ref sms _ state _ machine)
*/

typedef enum {
     SMS _ OK,                  /**< OK von Mobiltelefon empfangen */
     SMS _ ERROR,               /**< ERROR von Mobiltelefon empgangen */
     SMS _ MSG _ DECODE,        /**< Textnachricht eingegangen, decodiert und in \ref
                                SMS _ DECODE _ DATA _ T Struktur gespeichert */
     SMS _ INCOMING _ RING,     /**< eingehender Anruf */
     SMS _ CANCEL _ RING,       /**< eingehender Anruf wurde abgewiesen */
     SMS _ REQUEST _ TELNR,     /**< Telefonnummer des eingehenden Anruft wurde abgefragt */
     SMS _ RETRIEVE _ TELNR,    /**< Anrufer wurde abgewiesen, Telefonnummer ermittelt und
                                korrekt mit \ref SMS _ TELNR bzw. \ref SMS _ TELNR _ LIST _ T
                                verglichen - über \ref sms _ last _ caller kann jetzt der
                                Anrufer identifiziert werden */
     SMS _ RETRIEVE _ nTELNR,   /**< Anrufer wurde abgewiesen, Telefonnummer ermittelt und
                                nicht korrekt mit \ref SMS _ TELNR bzw. \ref SMS _ TELNR _
                                LIST _ T verglichen - über \ref sms _ last _ caller kann jetzt
                                der Anrufer identifiziert werden */
     SMS _ IDLE                 /**< empfangene Zeichenkette wurde nicht in sms _ state _
                                machine „gefunden“ (kein Eintrag) */

} SMS _ MSG;
```

Abb. 5

Diese Funktion führt ebenso das Textphrasing bei empfangenen Textnachrichten durch. Die Zuordnung von Schlüsselwörtern erfolgt wiederum durch eine spezielle Datenstruktur:

```
typedef struct SMS _ CMD _ LIST _ S{
     char cmd[10];                  /**< Zeichenkette für Kommando */
     uint8 _ t (*fp)(uint8 _ t dat);    /**< Funktionspointer auf Funktion, welche Kommando
                                    \ref cmd zugeordnet werden soll */

}SMS _ CMD _ LIST _ T;
```

Die Parametrisierung der Datenstruktur erfolgt in der Datei sms.c. Beispielhaft könnte diese wie folgt aussehen:

```
const SMS _ CMD _ LIST _ T sms _ cmd _ list[] = {
     {„set",sms _ cmdset},
     {„test",sms _ cmdtest}
};
```

Die Schlüsselwörter müssen dabei immer aus **Kleinbuchstaben** bestehen, da die eintreffende Textnachricht in Kleinbuchstaben konvertiert wird. Das erleichtert den Vergleich und verhindert ungewollte Schreibfehler bei Groß- und Kleinschreibung. Die Definition der Trennzeichen zwischen den einzelnen Schlüsselwörtern erfolgt in der Headerdatei sms.h. Hier beispielhaft Komma, Semikolon, Doppelpunkt und Leerzeichen:

```
#define SMS _ DELIMITER „,;: „
```

Eine eingehende Textnachricht mit dem Inhalt „set, test“ würde demnach die Funktionen sms_cmdset und sms_cmdtest ausführen. Dabei ist zu beachten, dass die angelegten Funktion
en stets vom Typ

```
uint8 _ t foo (uint8 _ t dat);
```

sind. In der Bibliothek muss der entsprechende Prototyp in der Datei sms.h angegeben werden. Das könnte zum Beispiel folgendermaßen aussehen:

```
extern uint8 _ t foo (uint8 _ t dat);
```

Die eigentliche Funktion kann in die Datei main.c geschrieben werden.

Verwendung der Bibliothek in eigenen Projekten

Die Software wird als komplett kompilierbares Projekt zum Download bereitgestellt, in dem auch die Hauptfunktionen und deren Anwendung erläutert werden. Zunächst müssen natürlich einige Einstellungen vorgenommen werden.

Voreinstellungen

Die Baudrate für die Kommunikation mit dem Mobiltelefon kann in der Datei uart.h über

```
/**
* definiert Baudrate für serielle
Schnittstelle
*/
#define UART _ BAUD 19200UL
```

geändert werden. Die Einstellung von 19200 Baud sollte aber bei den meisten Mobiltelefonen korrekt sein. Des Weiteren kann in der Headerdatei auch die Größe des Empfangspuffer über

```
/**
* Definiert die maximale Anzahl von Zeichen,
welche über die Rx-Interruptroutine
empfangen werden können.
* <br> Dient damit auch zum Festlegen der
Größe der globalen Variable /ref uart _
string.
*/

#define UART _ MAXSTRLEN 380
```

eingestellt werden. Der relativ groß eingestellte Puffer kommt durch das PDU Format eingehender Textnachrichten zustande. Neben der eigentlichen codierten Nachricht wird noch der Header mit Telefonnummer, Zeitstempel und anderen Angaben zum Mikrocontroller übertragen. Die Textnachricht kann maximal 160 Zeichen enthalten. Im PDU Format werden 240 Zeichen für diese Textnachricht übertragen. Die minimale Puffergröße sollte

$$minAnz = 2 \cdot \left(160 - \frac{160}{8}\right) + 60 + 5 = 345$$

sein, um korrekt Textnachrichten bis zu einer Länge von 160 Zeichen empfangen zu können (60 ist die maximale Größe des PDU Header und 5 eine kleine Reserve für Steuerzeichen wie new line).

Die nächsten Einstellungen sind in der Headerdatei sms.h vorzunehmen. Hier muss zunächst die Telefonnummer über

```
/**
* Adressat der SMS
* <br> (immer führende 0 mitschreiben,
NICHT +49)
*/

#define SMS _ TELNR „01623456789"
```

angepasst werden.

Dieses Makro kann zum einen genutzt werden, um mit Hilfe von sms_send() Textnachrichten zu verschicken, zum anderen dient es aber auch der Anrufererkennung in sms_state_machine(). Ein weiteres wichtiges Makro ist:

```
/**
* definiert die maximale Anzahl der
Telefonnummern in der Telefonnummernliste
* <br> Dies kann zur
Anrufernummernerkennung genutzt werden.
Dabei werden immer die ersten \ref SMS _
TELNR _ LIST _ MAX Nummern der SIM-Karte
verwendet (siehe \ref sms _ init)
*/

#define SMS _ TELNR _ LIST _ MAX 5
```

Darüber lässt sich steuern, wie viele Telefonnummern aus der SIM-Karte zum Vergleich mit herangezogen werden sollen. Ein Wert von fünf bedeutet demnach, dass die ersten fünf Telefonnummern der SIM-Karte analog zu SMS_TELNR in der sms_state_machine ausgewertet werden. Ein korrekter Vergleich mit den Telefonnummern kann über die Rückgabewerte der Funktion sms_state_machine realisiert werden. Neben der Telefonnummer können in der Headerdatei auch die Trennzeichen für die Schlüsselwörter einer empfangenen Textnachricht eingestellt werden.

```
/**
* Trennzeichen um Befehle aus empfangenen
Textnachrichten zu selektieren
*/

#define SMS _ DELIMITER „,;: „
```

Dies sind die wichtigsten Einstellungen welche vorab überprüft werden müssen.

Nutzung der Funktionen

Neben der bereits erfolgten kurzen Funktionsbeschreibung, soll an dieser Stelle noch einmal auf die Besonderheit der Funktion sms_state_machine() eingegangen werden. Die Funktion wird zyklisch in der main loop aufgerufen und hat als Rückgabewert folgenden Aufzählungstyp SMS_MSG:

```
siehe Abb. 5
```

Über diesen Aufzählungstyp kann der zuletzt empfangene Datensatz vom Mobiltelefon verifiziert werden. So bedeutet der Rückgabewert SMS_RETRIEVE_TELNR z. B. dass ein eingehender Anruf mit der in SMS_TELNR angegebenen Telefonnummer bzw. mit den aus der SIM-Karte ausgelesenen Telefonnummern (siehe SMS_TELNR_LIST_MAX) erfolgte. Auf diesen Anruf kann nun in der main loop entsprechend reagiert werden. Das erfolgt nach dem Schema:

```
uint8 _ t flag;
// main loop
while(1){
     flag = sms _ state _ machine();
     if (flag == SMS _ RETRIEVE _ TELNR){
            // mache irgend etwas
     }
}
```

Die Telefonnummer des letzten Anrufers kann mit Hilfe der globalen Variable sms_last_caller weiter verarbeitet werden. So wäre es z. B. denkbar, eine Textnachricht an diesen zu senden.

Neben SMS_RETRIEVE_TELNR ist ebenso der Rückgabewert SMS_MSG_DECODE von Relevanz. Er zeigt an, dass eine Textnachricht empfangen, decodiert und in der Datenstruktur sms_dec_data abgelegt wurde. Die Struktur der globalen Variable sieht dabei wie folgt aus:

```
siehe Abb. 4
```

So kann über sms_dec_data.nachricht die zuletzt eingegangene Textnachricht oder über sms_dec_data.date.day der Tag ausgelesen werden.

Ein weiterer wichtiger Aspekt der Bibliothek ist das Textphrasing empfangener Textnachrichten und das Ausführen entsprechender Funktionen bei bestimmten Schlüsselwörtern. Nähere Erläuterungen zur Vorgehensweise und Einstellung der Schlüsselwörter sind dazu bereits in der obigen Funktionsbeschreibung zu sms_state_machine erfolgt.

Links

[1] http://twit88.com/home/utility/sms-pdu-encode-decode

[2] http://www.dreamfabric.com/sms/

[3] http://www.mdetroy.de/reload.html?/hp_files/prog/s45/pdu.htm

[4] http://www.allpinouts.org/index.php/Siemens_Slim

[5] _Lumberg_%28A55,_C55,_S55,_SL55,_ST55,_M55%29

[6] http://www.3gpp.org/ftp/Specs/html-info/23038.htm

[7] https://bugs.cihar.com/file_download.php?file_id=86&type=bug

[8] http://www.atmel.com/dyn/resources/prod_documents/doc8016.pdf

Einführung in den Kalman-Filter von Steffen Mauch

Der folgende Artikel ist eine kurze Einführung in den Kalman-Filter. Zunächst wird das allgemeine Filterungproblem skizziert, anschließend werden die Randbedingungen für den Kalman-Filter aufgeführt und näher diskutiert. Schließlich wird das Kalman-Filter vorgestellt und ausführlich erläutert. Die Herleitung des Kalman-Filter ist nicht Gegenstand dieses Artikels, hier sei zum Beispiel auf [GA08] beziehungsweise [Kal60], [Bau10] verwiesen.
Zuletzt wird ein einfaches Beispiel, der freie Fall, modelliert und das Kalman-Filter für diesen Fall entworfen und in Matlab realisiert. Anhand den Simulationsergebnissen wird der Kalman-Filter exemplarisch erläutert.

Filter Problem

Das allgemeine Filterproblem wird für $k \geq 0$ wie folgt angegeben:

$$x_{k+1} = \Phi_k x_k + B_k u_k + w_k \quad (1)$$

$$z_k = H_k x_k + v_k \quad (2)$$

In Abbildung 1 ist das 'basic signal model' in zeitvarianter Form abgebildet. Bei den Signalen v_k und w_k handelt es sich mittelwertfreies, normalverteiltes, weißes Rauschen[1]. Ein weißes Signal enthält gleiche Leistung für alle Frequenzen und zudem ist der Erwartungswert[2] des Signals null. v_k und w_k sind zusätzlich gaußsche Prozesse mit einer bekannten Kovarianz. Folglich ist $E\{v_k\} = 0$, $E\{w_k\} = 0$, $E\{v_k v_k^T\} = R_k$ and $E\{w_k w_k^T\} = Q_k$, wobei $E\{\cdot\}$ der Erwartungswert ist.

$$v_k \in \mathcal{N}\{0, R_k\} \; ; \; w_k \in \mathcal{N}\{0, Q_k\} \quad (3)$$

$x_k \in \mathbb{R}^n$ ist der Zustandsvektor, $w_k \in \mathbb{R}^n$ ist das unbekannte Systemrauschen, $v_k \in \mathbb{R}^q$ das unbekannte Messrauschen, $\Phi_k \in \mathbb{R}^{n \times n}$ die Zustandsübergangsmatrix (state transition matrix), $B_k \in \mathbb{R}^{n \times p}$ die Störmatrix (forcing function matrix/input matrix), $H_k \in \mathbb{R}^{q \times n}$ die Messmatrix, $z_k \in \mathbb{R}^q$ der mit Störungen behaftete Ausgang und $u_k \in \mathbb{R}^p$ der Steuerungsvektor (control vector).

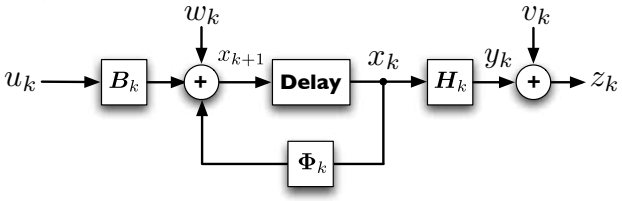


Abbildung 1: Basic signal model

Um vereinfachte Filter Gleichungen zu erhalten, wird das Filterungsproblem umgeschrieben, so dass für x_k nur Messungen bis zur Zeit $k-1$ benötigt werden. Dies ist ein *one-step predicition Problem* welches jedoch trotzdem als Filterproblem gehandhabt wird.

Kalman-Filter

Der Kalman-Filter ist ein rekursiver Algorithmus zur Schätzung des Zustandes eines linearen Systems. Er wurde 1960 von Rudolf Emil Kalman unter dem Namen 'A new approach to linear filtering and prediction problems' [Kal60] publiziert. Bei dem Kalman-Filter werden Messwerte verarbeitet, welche in linearer Weise mit dem Systemzustand zusammenhängen müssen. Falls der Zusammenhang zwischen Messwerten und Systemzustand oder das beobachtete System nichtlinear ist, kann ein erweiteres Kalman-Filter (EKF) eingesetzt werden, siehe [AM05].
Der Kalman-Filter kann als rekursiver Algorithmus verstanden werden, der die Prinzipien der Ausgleichsrechnung nach der Methode der kleinste Quadrate auf dynamische Systeme erweitert. Es werden keine konstanten Parameter geschätzt sondern zeitvariable Zustandsgrößen $\hat{x}$ und deren Kovarianzmatrix. Der Kalman-Filter ist ein diskreter Filter, die zeitkontinuierliche Variante wird als Kalman-Bucy Filter bezeichnet.
Um die Zustandsschätzung durchführen zu können, wird das folgende zeitvariante Modell für das beobachtende System zugrunde gelegt:

$$\hat{x}_k^- = \Phi_k \cdot \hat{x}_{k-1} + B_k \cdot u_{k-1} \quad (4)$$

Der tatsächliche Systemzustand wird mit x_k, der geschätzte mit $\hat{x}_k$ beschrieben. Der obere Index

1 Als weißes Rauschen wird ein Zufallssignal mit einer flachen spektralen Leistungsdichte bezeichnet.

2 Der Erwartungswert ist ein Begriff der Stochastik. Der Erwartungswert einer Zufallsvariablen ist jener Wert, der sich (in der Regel) bei oftmaligem Wiederholen des zugrunde liegenden Experiments als Mittelwert der Ergebnisse ergibt.

$^{+/-}$ gibt an, ob der aktuelle Messwert bereits verarbeitet wurde $^{+}$ oder die Bearbeitung noch aussteht $^{-}$. Ersterer Fall wird als a posteriori, zweiterer als a priori Schätzwert bezeichnet. In Abbildung 2 ist das zeitvariante Modell visualisiert, wobei hier $B_k \cdot u_{k-1} = 0$ angenommen wird, folglich sind also keine Steuerungseingänge vorhanden.

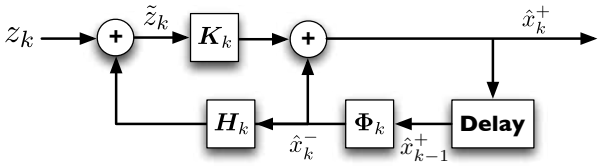


Abbildung 2: Kalman-Filter Modell ohne Eingänge

Eine sehr gute Anlaufstelle für weiterführende Informationen bezüglich des Kalman-Filters ist die Webseite `http://www.cs.unc.edu/~welch/kalman/`, beziehungsweise die angegebene Literatur. Im folgenden werden die Modellgleichungen (4) zeitinvariant behandelt, somit werden die Parameter Φ_k, B_k, ... unabhängig vom Index $_k$, sind also konstant.

Gleichungen des Kalman-Filter

Die Kalman-Filter Gleichungen werden in Estimation und Propagation unterteilt. Bei der Estimation werden die zum aktuellen Zeitpunkt vorliegenden Messwerte verarbeitet. Mit diesen Werten wird die Schätzung des Systemzustands verbessert und die Kovarianzmatrix des Schätzfehlers angepasst. Als Startwert wird

$$\hat{x}_0 = E\{x_0\} \tag{5}$$
$$P_0 = E\{[x_0 - \hat{x}_0][x_0 - \hat{x}_0]^T\} \tag{6}$$

angenommen. Der Initialwert P_0 gibt an, ob die exakten Werte von x_0 bekannt. Ist dies der Fall, so wird die Kovarianzmatrix P_0 als 0-Matrix definiert, ansonsten sollte die Matrix mit großen Werten L auf der Diagonale besetzt werden.

$$P_0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \; ; \; P_0 = \begin{bmatrix} L & 0 \\ 0 & L \end{bmatrix} \tag{7}$$

Bei großen Werten auf der Diagonale, wird der Kalman-Filter die Messwerte stärker gegenüber den Werten des Modells gewichten.
Der geschätzte Systemzustand wird propagiert, das heißt aus dem posteriori Schätzwert zum Zeitpunkt $k-1$ wird der a priori Schätzwert zum Zeitpunkt k berechnet. Die Unsicherheit bezüglich der Zustandsschätzung wächst und die Schätzfehlerkovarianzmatrix muss entsprechend angepasst werden. Die a priori Kovarianzmatrix P_k^- wird wie folgt berechnet:

$$P_k^- = \Phi \cdot P_{k-1} \cdot \Phi^T + Q \tag{8}$$

Der Kalman-Gain K_k ist das Resultat eines mathematischen Beweises:

$$K_k = P_k^- \cdot H^T (H \cdot P_k^- \cdot H^T + R)^{-1} \tag{9}$$

Ist ein Element von K_k null, so wird in der folgenden Schätzgleichung dieses Element ausschließlich aus dem vorhergehenden Schätzwert ermittelt. Wird ein Element von K_k jedoch eins, so wird bei der Berechnung dieses Elements in der Schätzgleichung ausschließlich der Messung geglaubt. Ist die Systemmatrix Φ zeitinvariant, so kann der Kalman-Gain offline berechnet werden, unter der Voraussetzung dass A bekannt ist. Der a posteriori Schätzwert $\hat{x}_k$ ebenso wie die dazugehörige Fehlervarianz P_k

$$\hat{x}_k^+ = \hat{x}_k^- + K_k(z_k - H \cdot x_k^-) \tag{10}$$
$$P_k^+ = (I - K_k \cdot H) \cdot P_k^- \tag{11}$$

werden in Abhängigkeit von K_k ermittelt. Die Kovarianzmatrix P_k ist ein Maß für die Genauigkeit der Zustandschätzung. Unter der Bedingung dass die Varianzen R und Q sich während des gesamten Prozesses nicht ändern, konvergieren K_k und P_k sehr schnell, normalerweise in weniger als zehn Iterationen. Der Kalman-Filter liefert nicht nur eine Schätzung, sondern zudem auch ein Maß für die Güte der Schätzung.

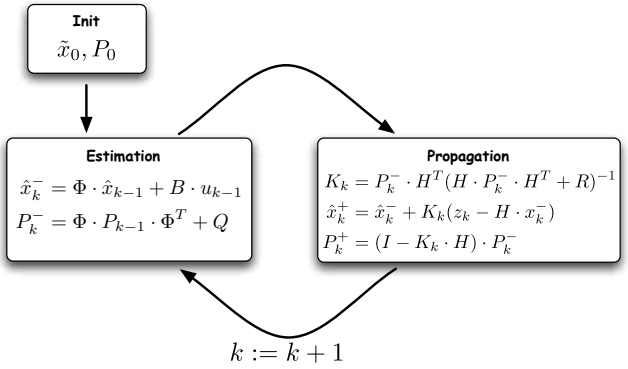


Abbildung 3: Überblick Kalman-Filter

Ist das Modell exakt und geben $\hat{x}_0$ und P_0 ebenso den exakten initialen Systemzustand an, kann man den Fehler e_k folgendermaßen angeben:

$$e_{k_1} = x_k - \hat{x}_k^+ \; ; \; e_{k_2} = x_k - \hat{x}_k^- \tag{12}$$
$$E\{e_{k_1}\} = 0 \; ; \; E\{e_{k_2}\} = 0 \tag{13}$$

Die Kovarianzmatrizen[3] entsprechen ebenso exakt den geschätzen:

$$P_k^+ = cov[x_k - \hat{x}_k^+] \quad (14)$$
$$P_k^- = cov[x_k - \hat{x}_k^-] \quad (15)$$

Freier Fall, siehe [Bau10]

Man nehme folgendes an: ein Greifarm soll mit Hilfe einer Kamera einen Ball verfolgen (tracken) und diesen Ball mit seinem Greifarm fangen. Der Einfachheit halber wird angenommen, dass der Ball sich nur in y-Richtung bewegt. Dem Prozessor wird die detektierte Höhe, ermittelt durch die Kamera, zur Verfügung gestellt. Diese Höhe unterliegt jedoch einem Messfehler und weist eine gewisse Varianz auf.
Durch den Einsatz des Kalman-Filters soll die Positionsgenauigkeit verbessert werden, sodass der Greifarm den Ball immer zuverlässig fassen kann.
Der Zustandsvektor ist folgendermaßen definiert:

$$x_k = \begin{pmatrix} h & v \end{pmatrix}^T \quad (16)$$

wobei h für die Höhe steht und v für die Geschwindigkeit.

Die Abtastzeit beträgt $\Delta t = 4\text{ms}$, woraus sich eine Abtastfrequenz von 250 Hz ergibt. Die Erdbeschleunigung wird mit $g = 9.81\frac{kg \cdot m}{s^2}$ angenommen. Die Zustandsgleichung lautet folglich in Matrix-Notation:

$$\begin{pmatrix} h \\ v \end{pmatrix}_k = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} h \\ v \end{pmatrix}_{k-1} + \begin{pmatrix} -\frac{1}{2} \cdot g \cdot (\Delta t)^2 \\ -g \cdot \Delta t \end{pmatrix} + \begin{pmatrix} w_h \\ w_v \end{pmatrix}_k \quad (17)$$

Die Kovarianzmatrix wird zu Beginn der Simulation

$$Q = \begin{pmatrix} 0.02 & 0 \\ 0 & 0.01 \end{pmatrix} \quad (18)$$

angenommen. Die Messmatrix lautet

$$H = \begin{pmatrix} 1 & 0 \end{pmatrix} \quad (19)$$

da lediglich die Höhe benötigt wird. Auf das Testsignal wird weißes Rauschen addiert, mit einer Varianz von $\sigma_s^2 = 0.1$.

Die Varianz des Messrauschen R ist $\sigma_m^2 = 9$ und P_0 beträgt

$$P_0 = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}. \quad (20)$$

In Abbildung 4 sind die verrauschten, ebenso die wahren Messwerte (gestrichelt) abgebildet. Zusätzlich sind noch die geschätzten Werte des Kalman-Filters eingezeichnet.

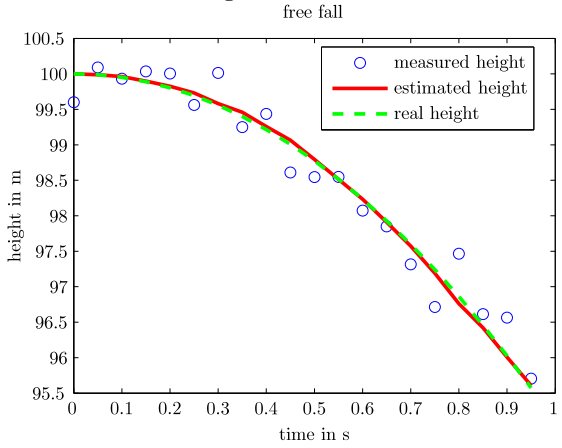


Abbildung 4: Simulation: freier Fall - Höhe

Man erkennt die signifikante Verbesserung der Genauigkeit der Höhe. Mit Hilfe dieser geschätzten Werte sollte ein zuverlässiges greifen des Balls möglich sein. In Abbildung 5 wurde die Varianz des Messrauschen auf $\sigma_m^2 = 0.1$ reduziert. Der Filter reagiert hierdurch stärker auf das Rauschen des Signals. Es wird stärker der Messung geglaubt, welche nach wie vor die gleiche Varianz aufweist.

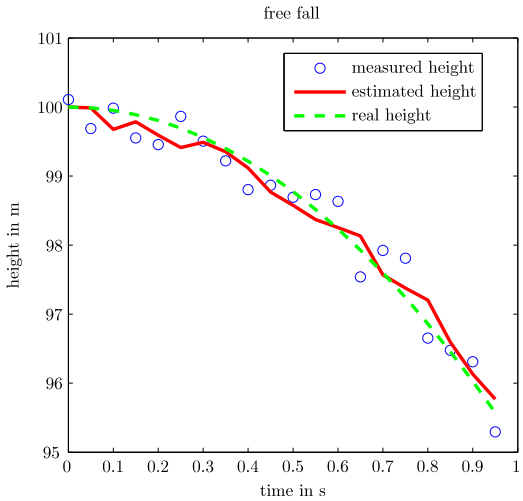


Abbildung 5: Simulation: freier Fall - Höhe mit $\sigma_m^2 = 0.1$ (R Messrauschen)

In Abbildung 6 hingegen wurden einige Messwerte gelöscht und zudem Störungen, wie zum Beispiel ein starker Luftzug simuliert. Die Varianz des Messrauschen wurde wieder auf $\sigma_m^2 = 9$

3 cov(·) enspricht der Kovarianz, wobei die Kovarianz wie folgt definiert ist: $\text{cov}(x) = E\{[x - E\{x\}][x - E\{x\}]^T\}$

gesetzt. Gut ersichtlich ist, dass das System eine gewisse Zeit braucht, bis es den Messungen traut. Dies liegt daran, dass Q relativ niedrig gewählt ist.

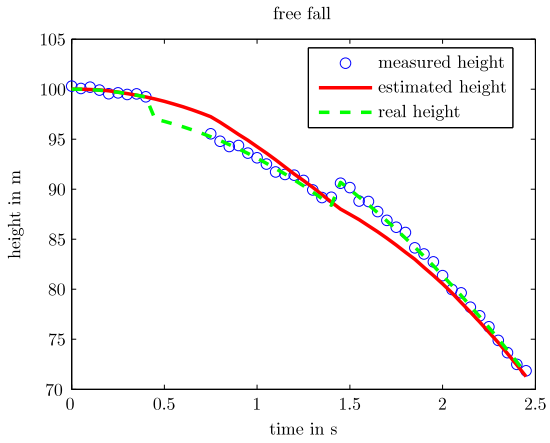


Abbildung 6: Simulation: freier Fall - mit Abweichung

Wird Q jedoch generell höher gewählt, so kann das Rauschen als Modellabweichung interpretiert werden. In Abbildung 7 kann man einen weiteren Vorteil des Kalman-Filters erkennen. Bleiben Messungen zum Beispiel auf Grund etwa einer Verdeckung des Objekts aus, schätzt der Kalman-Filter die neue Position auf Basis der alten Daten und mit dieser geschätzten Position kann weiter agiert werden.

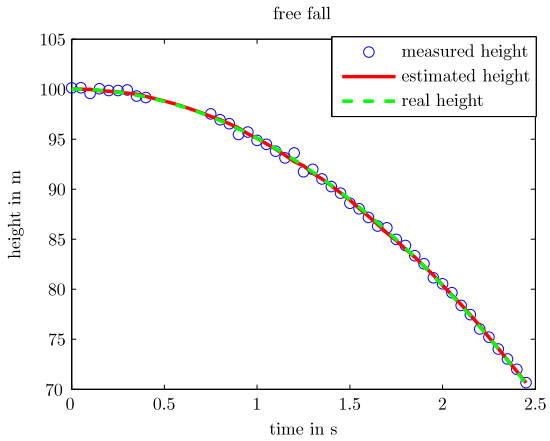


Abbildung 7: Simulation: freier Fall - mit Lücken

Für die nachfolgende Signalverarbeitung bedarf es keiner zwingenden gesonderten Behandlung wenn Messdaten ausbleiben, da der Kalman-Filter den ausbleibenden Messwert kompensiert.

Fazit

Der Kalman-Filter ist ein sehr mächtiger Filter, welcher optimale Ergebnisse in Bezug auf die Minimierung der Varianz $J_k = E[(x_k - \hat{x}_k)^2]$ liefert. Die Implementierung des Filters ist auch auf sehr leistungsschwachen Plattformen möglich, so wurde der erste Kalman-Filter bereits bei dem Apollo Programm eingesetzt. Er eignet sich ideal für den Einsatz in Echtzeitsystemen. Aus dem Kalman-Filter kann der Wiener-Filter als Spezialfall für stationäre Signale und Störungen abgeleitet werden. Der Kalman-Filter hingegen kann sich selbstständig an nicht-stationäre Umgebungen adaptieren. Ist das Signal als auch das Rauschen gaußverteilt, so ist der Kalman-Filter optimal im Sinne des MSE Kriteriums[4].
Ist das Signal und/oder das Rauschen nicht gaußverteilt, dann ist der Kalman-Filter der beste lineare Schätzer, welcher den MSE unter allen möglichen linearen Schätzer minimiert ([AM05], Seite 49).
Gerade bei der Sensordatenfusion[5] wird der Kalman-Filter sehr häufig eingesetzt, etwa bei Fahrerassistenzsystemen oder bei der Lokalisierung des Balls beziehungsweise der Roboter untereinander im Roboterfußball.

Literatur

[AM05] Anderson, Brian D. O. ; Moore, John B.: *Optimal Filtering.* Dover Publications, 2005

[Bau10] Baumann, Claude: Das Kalman-Filter wird 50. (2010). `http://www.convict.lu/htm/rob/Das%20Kalman%20Filter%20wird%2050_v_1.4.pdf`

[GA08] Grewal, Mohinder S. ; Andrews, Angus P.: *Kalman Filtering: Theory and Practice Using MATLAB.* 3. Wiley-IEEE Press, 2008

[Kal60] Kalman, R.E.: A New Approach to Linear Filtering and Prediction Problems. In: *Transaction of the ASME, Journal of Basic Engineering, Page 35-45* (1960)

4 mittlere quadratische Abweichung oder mittlerer quadratischer Fehler (mean squared error), $MSE(f,g) = E[\|f(X) - g\|^2]$

5 Die Sensordatenfusion wird allgemein als die Verknüpfung von Ausgabedaten mehrerer Sensoren bezeichnet. Ziel ist die Gewinnung von Informationen besserer Qualität.

Matlab - Sourcecode

Listing 1: Beispiel: Kalman-Filter - Freier Fall

```
% Kalman-Filter free fall
% masterthesis
%
% example implementation for learning
%
% (c) 2010 - Steffen Mauch
% steffen.mauch@gmail.com

clear all;
close all;

    % gravitational acceleration
    g = 9.81;
    % time intervall in seconds
    deltaT = 0.05;
    % initial heigth of free fall
    height = 100;
    % number of discrete points
    count = 50;
    % variance of y
    variance = 0.1;
    % initial value for speed and height
        estimated!
    height_est = 100;
    initial_speed_est = 0;

y = height + 0.5 * -g * ( (0:count-1)*
    deltaT ).^2;

% disturb the measured signal
%y(10:end) = y(10:end)-2;
%y(30:end) = y(30:end)+3;

y_ideal = y;
% add some noise
y = y + randn(1, count)*sqrt(variance);

% kill some measurements to show
    interpolation
%y(10:15) = -999;

% constant definitions for kalman
    % state transition matrix
    Phi = [ [1 deltaT];[0 1] ];
    % forcing function matrix or input
        matrix
    u = [-0.5*g*(deltaT).^2 -g*deltaT
        ].';
    B = [ [1 0];[0 1] ];
    % covariance matrix of w
    Q = [ [0.02 0];[0 0.01] ];
    % measure matrix
    H = [1 0];
    % variance of measure
    R = 9;
    P_zero = [ [1 0];[0 2] ];

% init hatP and hatx
hatP = P_zero;
hatx = [height_est initial_speed_est].';

% create vectors
xout = zeros (count, 2);
xout(1,:) = hatx';

% calculation
for k=2:count

    % estimation
    starP = Phi * hatP * Phi' + Q;
    starx = Phi * hatx + B*u;

    % propagation
    K = starP * H' * ( H * starP * H' +
        R)^-1;

    % get actual value (not when value
        equals -999)
     if ( y(k)==-999 )
         y(k)=NaN;
         hatx = starx;
     else
         hatx = starx + K * ( y(k) - H *
             starx);
     end

    hatP = starP - K*H*starP;

    xout(k,:) = hatx.';
end

% plot height
    figure();
    scale = 0:deltaT:count*deltaT-deltaT
        ;
    plot(scale, y,'o');
    hold on;
    plot(scale, xout(:,1),'red','
        linewidth',2);
    plot(scale, y_ideal,'--g','linewidth
        ',2);
    hold off;
    title('free fall');
    ylabel('height in m');
    xlabel('time in s');
    legend('measured height','estimated
        height','real height');

% plot error
    figure();
    scale = 0:deltaT:count*deltaT-deltaT
        ;
    plot(scale, xout(:,1).'-y_ideal,'-')
        ;
    title('error of estimated and real
        height');
    ylabel('difference in m');
    xlabel('time in s');

% END
```

Eine günstige CNC-Fräse

Bau einer günstigen CNC-Fräse zum Fräsen von Platinenlayouts. Bericht eines Projekts.

Jacob Seibert

Da mir das Belichten, Entwickeln und Ätzen von Leiterplatten zu umständlich und von zu vielen Misserfolgen begleitet war, beschloss ich eine kleine günstige CNC-Fräse zu bauen, die einfache Platinen fräsen können sollte. Ähnliche Projekte sind im Internet zu finden, auch auf dem Portal http://www.roboternetz.de sind einige vorgestellt und/oder diskutiert worden. [1]

Die Fräse muss natürlich eine gewisse Genauigkeit haben und die gesamte Konstruktion muss steif sein, denn der Abstand zwischen Leiterbahnen auf der Platine kann schon bei relativ einfachen Layouts schnell mal unter einem Millimeter liegen.

Die geforderte Genauigkeit steht leider im Gegensatz zu meiner zweiten Anforderung an das Projekt: Anders als bei vielen Fräsen, die mehrere hundert Euro kosten, habe ich stets versucht einfache und damit kostengünstige Lösungen zu finden. Selbstverständlich müssen dann Abstriche an anderer Stelle (z.B. Genauigkeit, Geschwindigkeit) gemacht werden. Doch mich hat es gereizt, es mit einfachen Mitteln zu versuchen.

Aufbau der Fräse (Mechanik)

Ich habe mich für folgende Bauart der Fräse entschieden: Das Werkstück wird auf einem sogenannten XY-Koordinatentisch befestigt, der in zwei der drei Dimensionen (X, Y und Z) bewegt wird. Darüber befindet sich der Fräsmotor mit dem Fräswerkzeug, der in der dritten Richtung (Z) bewegt wird.

Als Führungen der drei Richtungsachsen habe ich Schubladenführungen verwendet, die erstaunlich spielfrei sind. Der Antrieb der Achsen erfolgt über M5-Gewindestangen und Schrittmotoren.

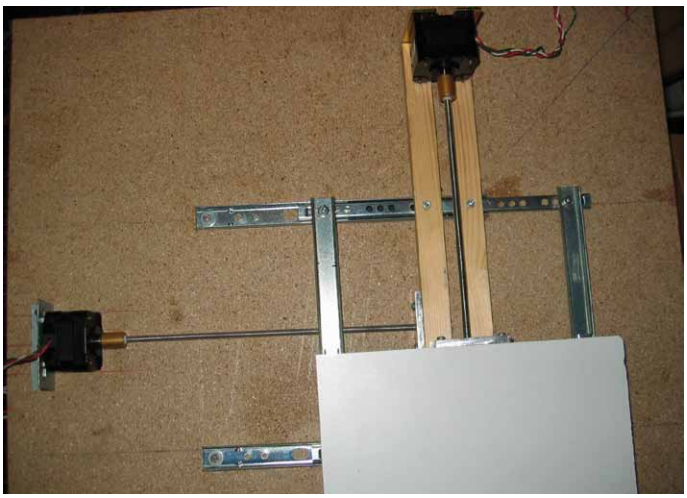


Abb. 1: Der XY-Koordinatentisch

Am XY-Koordinatentisch bzw. an der Trägerplatte der Z-Achse befinden sich 5mm dicke Aluminiumteile mit einem Innengewinde, in denen die antreibende Gewindestange läuft. Dreht der Schrittmotor die Gewindestange, bewegt sich die Achse entsprechend der Steigung des Gewindes in die eine oder die andere Richtung.

Abb. 2: Die Z-Achse

Die Z-Achse ist nach dem selben Prinzip aufgebaut. Aus Stabilitätsgründen habe ich hier aber drei Schubladenführungen verwendet, wovon eine um 90° längs zu den anderen gedreht ist. Aufgrund ihrer Länge entstehen bei der Z-Achse nämlich größere Hebelkräfte, die ein „Ruckeln“ bei nur zwei Führungen verursachen würden.

Diese gesamte Konstruktion ist auf einen 50cm x 50cm x 15cm Holzkasten gebaut, der als Grundplatte dient und der die Stromversorgung enthält. Auch die Höhe der Fräse beträgt etwa einen halben Meter. Die Stromversorgung wird der Einfachheit halber von einem gebrauchten PC-Netzteil sichergestellt, außerdem ist noch eine Stromanschluss für den Fräsmotor vorhanden.

Da hier mit Netzspannung gearbeitet wird, muss auf die Sicherheitsbestimmungen geachtet und mit größter Vorsicht vorgegangen werden! Wer nicht entsprechende Kenntnisse für solche Arbeiten besitzt, sollte auf fertige Lösungen zurückgreifen!

Als Fräsmotor kommt ein bereits vorhandener „Feinbohrschleifer“ zum Einsatz. Prinzipiell können sicher alle kleinen Handbohrmaschinchen verwendet werden. Die Aufnahme des Fräsmotors stammt aus einem passenden Bohrständer.

Die gesamte Fräse habe ich dann auf das Rollgestell eines alten Bürostuhls geschraubt, was der Fräse eine sehr angenehme Mobilität verschafft.

Auf Bild 3 und 4 sieht man den Fräsmotor nicht, denn er ist auf der anderen Seite der Z-Achs-Trägerplatte angebracht.

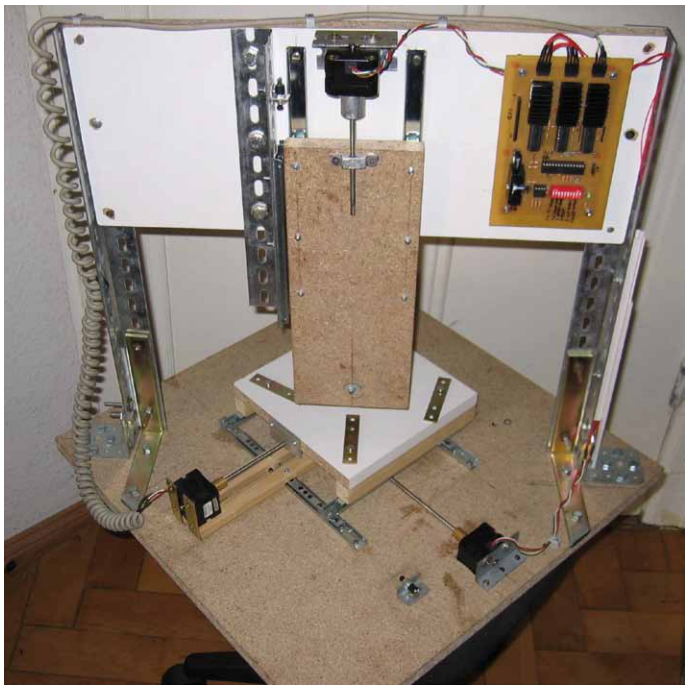


Abb. 3: Übersicht über die Fräse

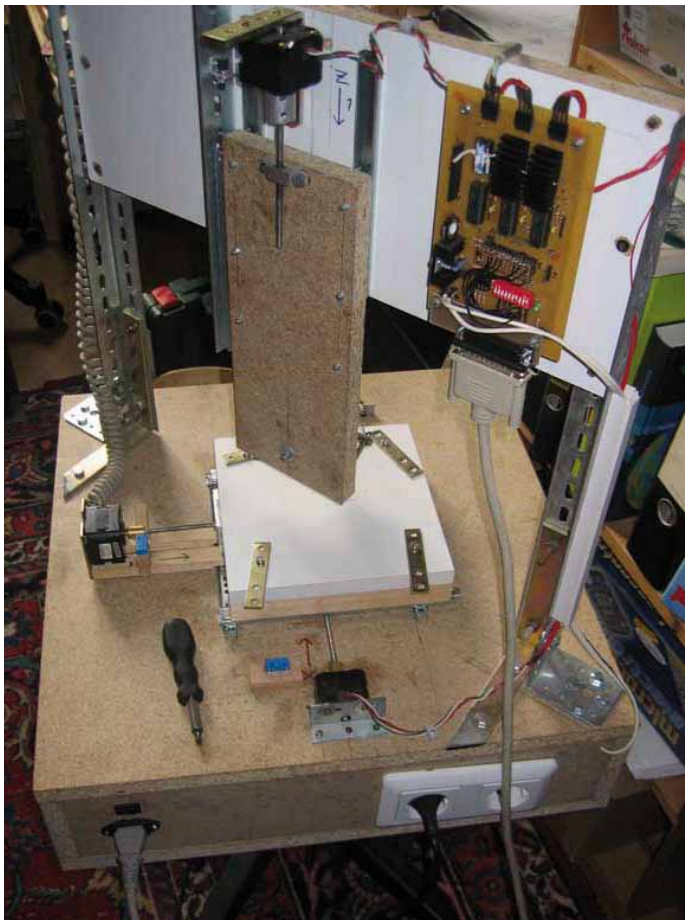


Abb. 3: Übersicht über die Fräse (fertig)

Um die Arbeit der Fräse auch bei schlechten Lichtverhältnissen gut verfolgen zu kön nen, habe ich aus einer ausgedienten LED-Fahrradlampe eine regelbare Beleuchtung gebaut. Die LED-Platine ist an einem Draht befestigt, sodass man sich das Licht dort „hinbiegen“ kann, wo man es benötigt.

Eine andere kleine Erweiterung der Fräse ist die Anschlussmöglichkeit eines Staubsaugers (über einen Schlauch), der den beim Fräsen entstehenden Staub und Dreck einsaugt. Die Plastikbauteile für diesen Anschluss habe ich übrigens mit der Fräse ausgeschnitten. Auf dem linken Bild sieht man die Halterung für den Fräsmotor.

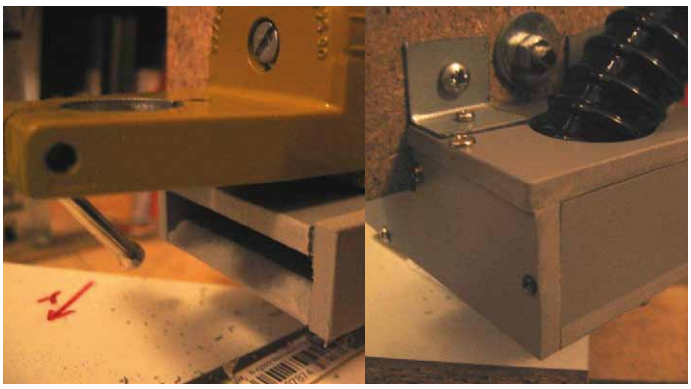


Abb. 5: Staubsaugeranschluss

Die Steuerung der Fräse/Elektronik

Die Steuerung der Fräse erfolgt durch den Computer, dazwischen benötigt es aber noch eine Treiberschaltung. Für diesen Zweck benutze ich die bekannten integrierten Schaltungen L297 und L293D. Damit kann die Fräse über ein Richtungs- und ein Taktsignal pro Motor gesteuert werden. Nähere Informationen zu diesen IC's gibt es in den Datenblättern. [2]

Des weiteren ist auch ein Speicherbaustein für Konfigurationsdaten, Fräsdaten oder sonstige Daten an den Mikrocontroller angeschlossen, aber bisher noch nicht in Verwendung.

Die Schaltung ist so ausgelegt, dass die Fräse zu Testzwecken auch mit einem eingesteckten ATmega8 gesteuert werden konnte. In den Sockel des Mikrocontrollers wird aber normalerweise ein selbstgebauter Adapter gesteckt, der den Anschluss des Druckerkabels zum Computer ermöglicht.

Software

Auf dem angeschlossenen Computer läuft Ubuntu mit EMC². [3]

Dies ist eine Open-Source Software um CNC-Fräsen, aber auch Maschinen im Allgemeinen mit dem Computer zu steuern. Die Konfiguration der Software wird durch einen Assistenten einfach gehalten, wenn man eine andere Hardware hat, wird die Konfiguration allerdings komplizierter. Als Eingangsdaten kann man GCode oder auch Graustufenbilder verwenden. Die Graustufenbilder werden als sogenannte „heightmaps“[4] behandelt und von EMC² in G-Code umgerechnet.

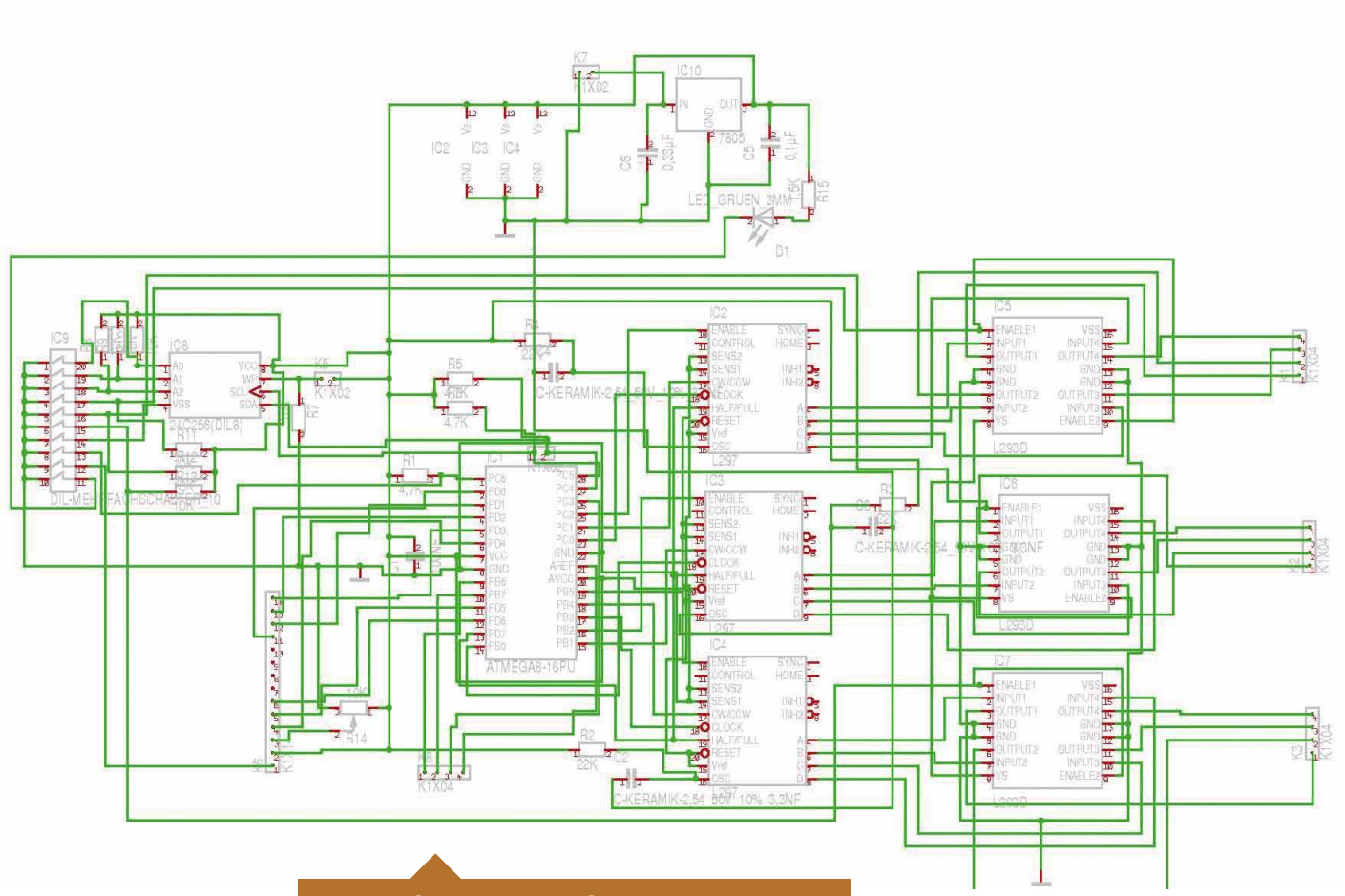


Abb. 6: Schaltplan der Steuerungsschaltung

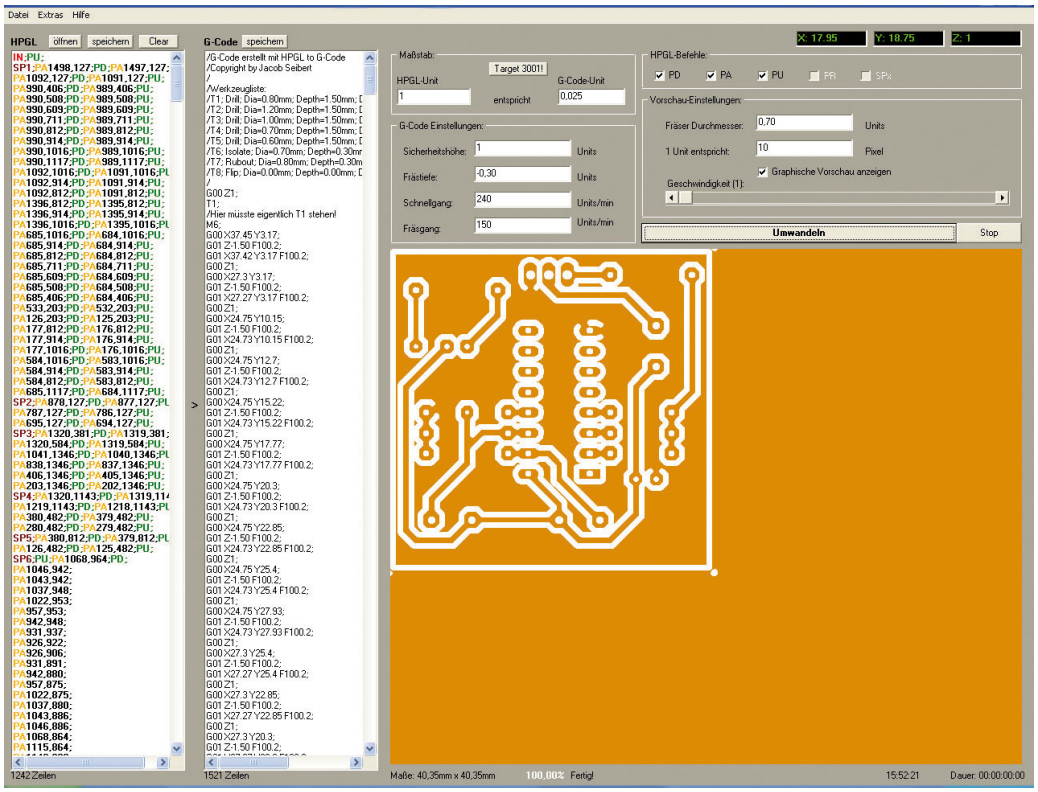


Abb. 7: Eigene Konvertierungssoftware von HPGL zu G-Code

Das Layoutprogramm, mit dem ich meine Platinenlayouts mache („Target 3001!“), kann die Fräsdaten aber leider nur als HPGL exportieren. Da ich im Internet keine für mich passende Konvertierungssoftware von HPGL nach G-Code fand, habe ich mir kurzerhand selbst ein Programm geschrieben. Erfahrungsgemäß ist es sowieso besser genau zu wissen, wie die Dinge funktionieren, weil dann Fehler schneller gefunden werden können.

Soweit ist also die Beschreibung der Fräse fertig. Doch jetzt ist es natürlich interessant, was letztlich herauskommt und ob damit etwas anzufangen ist.

Ergebnisse

Da der Zweck der Fräse das Platinenfräsen sein sollte, waren dies auch die ersten Tests. Dazu musste ich mir erst das richtige Fräswerkzeug, einen Gravierstichel, kaufen, das eine Fräsbreite von ca. 0,5mm hat. Schon die ersten Versuche waren recht vielversprechend (links), die späteren Versuche (rechts) überraschten mich positiv!

Dies sind zwar „nur“ normale Layouts, keine SMD, aber es funktioniert gut. Für Prototypen einfacherer Layouts absolut ausreichend! Wenn man sich die Layouts näher ansieht, erkennt

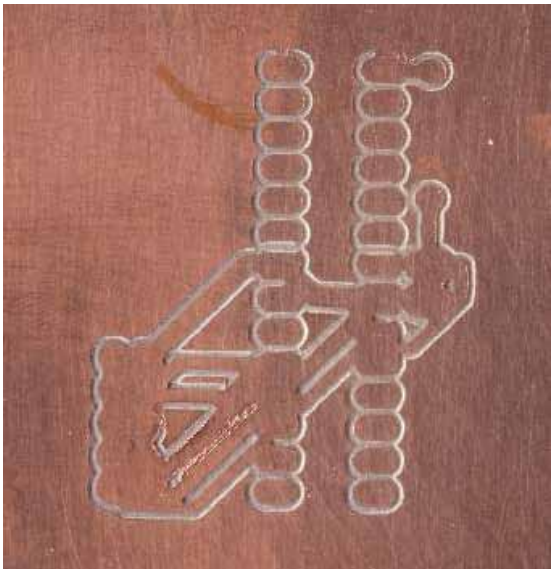


Abb. 8: Erster Versuch

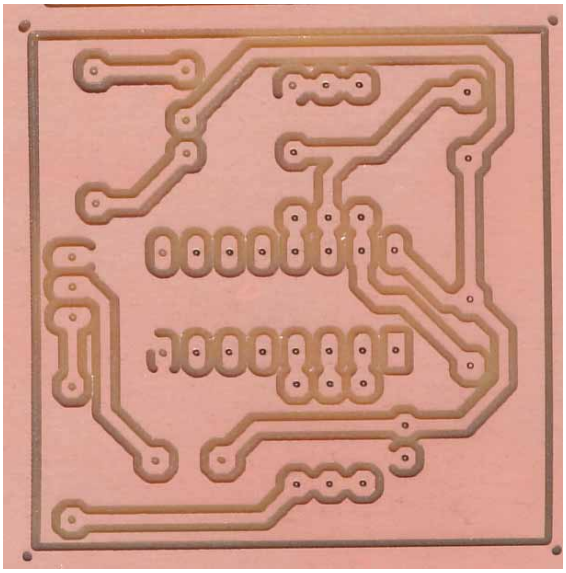


Abb. 9: Späterer Versuch

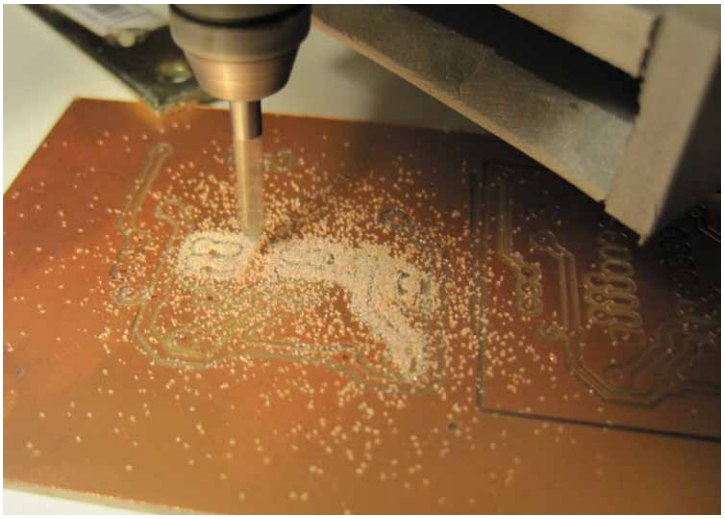


Abb. 10: Fräsen eines Layouts

Abb. 11: In Sperrholz gefrästes Foto

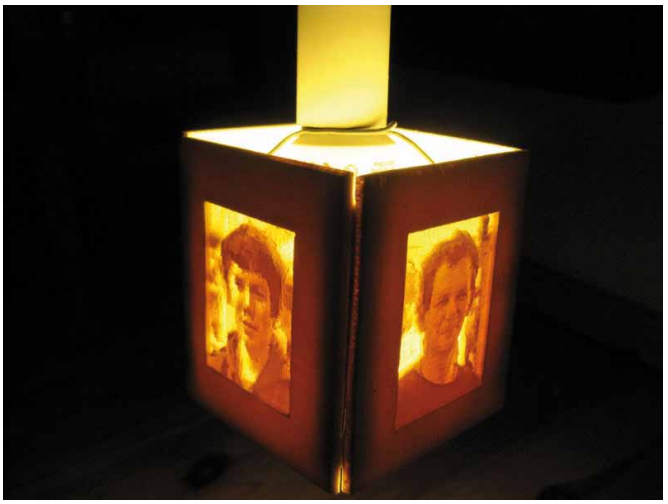


Abb. 12: Lampe aus gefrästem Sperrholz

man, dass manche Ecken leicht abgerundet sind. Dort sieht man also die Grenzen dieser Fräse, die letztlich eine Genauigkeit von etwa 0,2mm hat (exakt lässt sich das aber schwer sagen).

Aus einer spontanen Idee heraus entstand außerdem Folgendes: Ich habe Schwarz-Weiß-Bilder mit EMC² in Fräsdaten umwandeln lassen und dann in 4 mm Sperrholz gefräst.

Diese Lampe auf der Terrasse bzw. dem Balkon sorgt an lauen Sommerabenden für sehr stimmungsvolles Licht... Von diesem Fräsergebnis war ich sehr erstaunt und fasziniert! Die Bilder sind fotorealistisch und sehr detailreich.

Fazit

Das ist also der bisherige Stand meines Projektes. Nach gut zwei Jahren Arbeit daran, kann ich nun sagen, dass es sich gelohnt hat! Trotz „low-budget“ (die Projektkosten blieben unter 100,-€) kann man mit der Fräse brauchbare und schöne Sachen machen. Außerdem konnte ich Erfahrungen aller Art sammeln, die für zukünftige Projekte brauchbar sein können.

Wer noch Einzelheiten erfahren oder das Projekt weiterverfolgen möchte, sei auf meine Website http://www.roboterbastler.lima-city.de verwiesen. Dort gibt es auch noch andere Basteleien von mir zu sehen.

Links

[1] http://www.roboternetz.de/phpBB2/zeigebeitrag.php?t=39353&highlight=cncfr%E4se
http://www.roboternetz.de/phpBB2/zeigebeitrag.php?t=52563&highlight=cncfr%E4se
http://www.roboternetz.de/phpBB2/zeigebeitrag.php?t=38739&highlight=cncfr%E4se

[2] http://www.st.com/stonline/books/pdf/docs/1334.pdf
http://www.st.com/stonline/books/pdf/docs/1330.pdf

[3] http://linuxcnc.org/

[4] http://de.wikipedia.org/wiki/H%C3%B6henfeld

fastboot Howto

Wernfred Zolnhofer <zoli@augusta.de>

Abstract

Es muss nicht immer ein teurer Programmieradapter sein, der einen Mikrocontroller "zum Leben erweckt." Im Fall der ATMega-Serie von Atmel genügen 2 freie Portpins in Verbindung mit einer RS232 Schittstelle und etwas Software. Wie dabei vorzugehen ist, wird im nachfolgenden Beitrag beschrieben. Einer kurzen Einführung zum Bootloaderkonzept folgt eine Schritt für Schritt Anleitung zur Programmierung von AVR-Mikrocontrollern mit dem Bootloader fastboot von Peter Dannegger [1].

Die Anleitung bezieht sich soweit nichts anderes angegeben ist auf fastboot_build26 [2] für die gcc Toolchain. Die Toolchain (avrdude, gnu-gcc, avrlibc, bootloader) ermöglicht es, fastboot unter Linux einzusetzen. Die Beispiele wurden unter Sabayone (Gentoo-Linux) ausgeführt. Mikrocontrollerboard war das KCPU- Modul, das bereits in Ausgabe 4/2009 [3] auf Seite 23 vorgestellt wurde. Ebenso können andere Boards mit Mikrocontrollern der Atmel ATMega-Serie eingesetzt werden.

Bootloader

Wie kommt ein Programm in den Mikrocontroller?

Ein Mikrocontroller ist zwar sehr universell einsetzbar, erhält aber erst durch die Anwendersoftware (Applikation) seine spezielle Funktionalität. Das Übertragen der Software auf den Mikrocontroller erfolgt dabei meist mit einem ISP-Dongle in der Zielumgebung (In System Programming). Dazu gibt es diverse Programmieradapter (z.B. USBProg) für verschiedene Schnittstellen (Parallelschnittstelle, SPI, JTAG,...). Die Programmieradapter werden entweder direkt durch ein entsprechendes Programm des PC gesteuert oder sie bestehen selbst wieder aus einem Mikrocontroller, der die Daten und Kommandos des Hostrechners entgegennimmt und die Anwendersoftware im Speicher des Zielsystems ablegt (flasht).

Was ist ein Bootloader?

Mit einem Bootloader (verkürzte Form des ursprünglichen Wortes bootstrap loader) kann diese Aufgabe aber auch von unserem Zielsystem erledigt werden. Man muss im Prinzip dazu lediglich etwas Flashspeicher für den Programmcode des Bootloaders abzwacken, kann dafür aber dann über Standartschnittstellen (z.B.RS232, USB) die Anwendersoftware in den Mikrocontroller einspielen, ohne dass man dazu noch einen Programmieradapter dazwischenschalten muss. Die serielle Schnittstelle kann nach dem Flashen des Programms ganz normal zur Ein- u. Ausgabe verwendet werden. Ein Bootloader ist also ein kleines Programm das beim Start des Mikrocontrollers ausgeführt wird, um dann die eigentliche Applikation nachzuladen.

Was sind die Vorteile eines Bootloaders?

- man braucht zur Übertragung der Anwendersoftware in den Mikrocontroller keinen Programmieradapter (kostengünstig und simpel).
- die Programmierung durch einen Bootloader ist relativ schnell
- man gewinnt zusätzliche Pins (JTAG und ISP)
- die Anwendersoftware kann verschlüsselt sein

Die Benutzereinstellungen im Makefile von fastboot_build27(zoli) sind für das K-CPU-Modul mit einem Atmel Mega32 zugeschnitten. Eine Anpasssung an andere Mikrocontroller und Boards der Atmel AVR-Reihe ist meist problemlos möglich. Eine Liste der derzeit unterstützten Mikrocontroller ist unter "user-presets" im Makefile zu finden.

Installation der Toolchain

Soweit noch nicht bereits erfolgt, sollte man sich jetzt für seine Linuxdistribution aus der AVR-Entwicklungsumgebung die Toolchain (Werkzeugsammlung), bestehend aus mehreren Kommandozeilen-Programmen herunterladen und installieren.

Die Bestandteile im einzelnen:

- Binutils: Assembler, Linker und weitere Hilfsprogramme.
- GCC: Der eigentliche C(++)-Compiler.
- AVR-LIBC: Die C-Standardbibliothek mit AVR-spezifischen Headerdateien und Funktionen.
- (AVRDUDE: universelle AVR-Programmiersoftware, kein eigentlicher Teil der Toolchain, aber oft verwendet)
- (evt. auch gdbg ....)

Anschließendd holt man sich bei mikrocontroller.net eine aktuelle Version von fastboot [2] oder build29 [4] des Boootloaders.

Nach dem Entpacken wird der Ordner nach kboot umbenannt:

```
tar -zxvf fastboot _ build29.tar.gz && mv
fastboot _ build29 kboot
```

Uploader

Der Uploader, das Linux-Konsolenprogramm auf der PC-Seite, kann bei avfreaks [5] oder hier [6] heruntergeladen werden.

Anschließend muss es für das Zielsystem (Hostrechner) noch kompiliert werden.

```
tar -zxvf bootloader-1.1.tar.gz; cd
bootloader-1.1
make clean && make all
```

Das Programm heißt dann schlicht “bootloader”. Entweder sie kopieren sich es jeweils in ihren Projektordner oder besser gleich nach /usr/bin bzw. nehmen es in ihre Pathvariable mit auf.

```
PATH=$PATH':/meinpfad/zu/bootloader' && echo
$PATH
```

Bei letzterem ist sichergestellt, dass “device.txt” mit den Kennumern der Mikrocontroller gefunden wird.

Dann brauchen wir noch die Definitionfiles [7] für die verwendeten Mikrocontroller die man hier [8] herunterladen kann:

Im Fall des K-CPU-Moduls betrifft das folgende Files,

- m16def.inc für ATMEGA16
- m32def.inc für ATMEGA32
- m64def.inc für ATMEGA64

die wir nach dem Entpacken des Zip-Files in den vorher angelegten Ordner kboot verschieben/kopieren.:

```
uzip AVR000.zip; mv m16def.inc kboot; mv
m32def.inc kboot; mv m64def.inc kboot
```

Anpassen des Bootloaders

Wenn wir das alles haben, können wir mit der Anpassung des Bootloaders weitermachen. Die Einstellungen werden komplett im Makefile vorgenommen. Also, ihren Lieblingseditor aufrufen und im Makefile im Abschnitt “user-presets” folgende Änderungen durchführen:

```
MCU = atmega32
```

wenn Sie einen ATMEGA32 verwenden das dazugehörige ATMEL Definition File

```
ATMEL_INC = m32def.inc
```

den Prozessortakt

```
F_CPU = 16000000
```

sowie Receive- und Transmit-Pin (muss nicht UART-Pin sein, da der integrierte Software-UART auf allen Portpins lauschen kann).

```
STX_PORT = PORTD
STX = PD1

SRX_PORT = PORTD
SRX = PD0
```

Eventuell ist noch der Debugger anzugeben

```
DEBUG = stabs+
```

Die Änderungen abspeichern und einen neuen Bootloader mit

```
make clean && make
```

generieren. Wenn alles geklappt hat, sollte dann der aktuelle Bootloader als “bootload.hex” im Intel Hex-Format vorliegen. Um ihn später der spezifischen Mikrocontrollerkonfiguration zuordnen zu können, benennen wir ihn sinnvoll um, z.B. in der Form boot_m32-16.hex für einen ATMEGA32 bei 16MHz Taktfrequenz. Das kann man nun für jeden eingesetzten Mikrocontroller und Taktfrequenzkombination wiederholen. Wer es sich zutraut, kann ja das Makefile entsprechend modifizieren, damit die diversen Hexfiles automatisch erzeugt werden.:

```
mv booloader.hex boot_m32-16.hex
```

Vorbereiten des Mikrocontrollers

Fusebits einstellen

Damit der Bootloader zu Beginn gestartet wird, müssen die entsprechenden Fusebits gesetzt werden. Möglich ist dies u.a. mit der AVRDude GUI. Auf derWebsite Fusecalc kann man sich die Fuses elegant zusammenstellen und erhält gleich die Parameter für avrdude.

Neben den bereits bekannten Fusebits zur Einstellung der Taktfrequenz etc. sind bei Verwendung eines Bootloaders zusätzlich folgende Fuses von Interesse:

BOOTRST: legt Sprungziel nach einem Reset fest; muss auf 0 programmiert werden, damit der Bootloader aktiviert wird (Fuse setzen)

BOOTSZ0 und **BOOTSZ1**: legen die Größe des Bootbereichs (BLS - Boot Loader Section) fest; wird auf 256Worte (= 512 Byte) oder dem minimal möglichen eingestellt (Beim ATMega644 1kByte)

Die Bootstartadressen liegen dann beim

ATMega8 0xF00

ATMega16 0x1F00

ÁTMega32 0x3F00

Bei einigen Controllern wie z. B. dem ATMega 48 wird das Flag “**Selfprogramming enabled**” gesetzt.

Fuse-Übersicht am Beispiel eines ATMega32 mit folgender Konfiguration

- 512 Byte Bootloader,
- 16MHz externer Quarz, 4ms Startup,
- Störungsarme Umgebung CKOPT=0
- EEPROM wird überschrieben

```
#
# hfuse
# ============================================================
#  OCDEN JTAGEN SPIEN CKOPT EESAVE BOOTSZ1 BOOTSZ0 BOOTRST
#    1      1     0     0     1       1       0       0    = 0xCC
#
# lfuse
# ============================================================
#  BODLEVEL BODEN SUT1 SUT0 CKSEL3 CKSEL2 CKSEL1 CKSEL0
#     1       1    1    0     1      1      1      1       = 0xEF
#
# lock
# ============================================================
#    XXX    XXX  BLB12 BLB11 BLB02 BLB01  LB2   LB1
#     0      0     0     0     1     1     1     1         = 0x0F
#                  +-----+--- SPM is not allowed to write
#                             to the Boot Loader section
```

Zum Brennen der Fusebits brauchen wir einen Programmieradapter (siehe auch unter Abschnitt “Bootloader einspielen”) mit passender Software. Die Beispiele wurden mit avrdude durchgeführt.

Fuses:

```
avrdude -c avrispv2 -P usb -p m32 -U
lfuse:w:0xef:m -U hfuse:w:0xcc:m
```

Lcckbits:

```
avrdude -c avrispv2 -P usb -p m32 -U
lock:w:0x0F:m
```

Beim K-CPU-Modul können für den ATMega16, ATMega32 und ATMega64 die gleichen Low-/High- Fusebitseinstellungen verwendet werden. Der kleinste Bootloaderbereich ist beim ATMega644 1kByte groß.

Der ATMega644 hat zusätzliche Extension Fuses:

```
avrdude -c avrispv2 -P usb -p 644 -U
efuse:w:0xff:m
```

Anmerkung: Das Setzen der Fusebits und Lockbits kann ggf. in das Makefile des Boootloaders mit integriert werden: “make fuses” und “make lockbits” - wer von den Lesern fühlt sich angesprochen?

Bootloader einspielen (flashen)

Um den Bootloader einzuspielen brauchen wir noch einmal einen Programmieradapter. Die nachfolgenden Beispiele wurden mit einem USBProg von Benedikt Sauter (usbprog ist ein freier Programmieradapter und ist im Shop [9] von Bendikt Sauter erhältlich. Es gibt jedoch auch zahlreiche Nachbauvorschläge im Netz.) ausgeführt. Als Programmiersoftware wurde avrdude verwendet, Zielsystem ist ein ATMega32 mit 16 MHz Taktfrequenz:

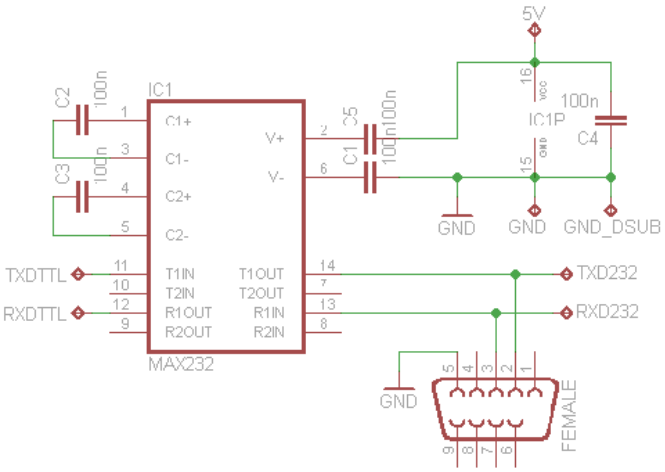
```
avrdude -c avrispv2 -P usb -p m32 -U
flash:w:boot _ m32-16.hexV
```

Das war es dann aber schon. Wer es noch bequemer haben möchte, kann hier [6] eine bereits für das K-CPU-Modul angepasste Version von fastboot herunterladen.

Anwenderprogramm flashen

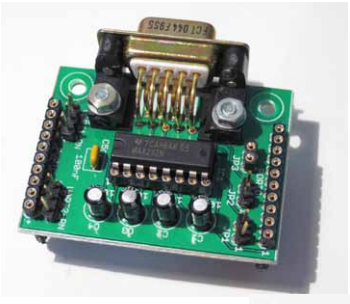
Die Hardwareschnittstelle

Zur Programmierung des Mikrocontrollers kann jeder freie Portpin benutzt werden (siehe hierzu fastboot mikrocontroller.net). Das K-CPU-Modul wurde mit der Hardware-UART-Schnttstelle (PD0 RxD und PD1 TxD) getestet. Die Pegelanpassung an die serielle (V24) Schnittstelle des PCs kann z.B. wie in der folgenden Standartschaltung gezeigt mit einem MAX232A oder MAX202 erfolgen.



Bei Verwendung des einfachen MAX232 (ohne “A” am Ende) sollte man jedoch bei den Kondensatoren C1, C2, C3 u. C5 für die Ladungspumpe an Stelle der 0,1uF Keramikkondensatoren, 1uF bis 10 uF Elektrolytkondensatoren einsetzen. Dabei ist auf richtige Polung zu achten.

Mit dem K-UART-Modul [10] klappte die Verbindung auch bei 115kBaud zum Host problemlos und es braucht nur auf das KCPU-Modul aufgesteckt werden. Platinen für die beiden Module können beim Autor bezogen werden. Die Anschlussleitung wird 1:1 durchverbunden, d.h. es wird kein Nullmodemkabel verwendet! Die 1-Draht-Ansteuerung wurde bislang noch nicht getestet.



Die Programmiersoftware

Um den Bootloader2 des Mikrocontrollers ansprechen zu können wird eine von Andreas Butti modifizierte Version des originalen Bootloader-Steuerungsprogramms verwendet. Durch Aufruf des Programms erhält man eine Kurzübersicht zu den möglichen Programmoptionen und -parametern

```
zoli@phecda ~/develop/mc/k-software/boot/bootloader-1.1 $ ./bootloader

================================================
|             BOOTLOADER, Target: V2.1          |
================================================
No hexfile specified!
./booloader [-d /dev/ttyS0] [-b 9600] [-r '\n\nRESET\n'] [-s] [-a 100] -[v|p] file.hex
-d Device
-b Baudrate
-v Verify
-p Programm
-r Character sequence to reboot AVR (with std. C escape codes)
-s Silent run - exits 0 on success, -1 on failure, errors to stderr
-a Number of AVR bootloader connection attempts (def.: 0 = infinite)

Author: Andreas Butti (andreasbutti at bluewin dot ch)
Modified by: Ilya Goldberg (igg at cathilya dot org
```

Zum Flashen des Mikrocontrollers im K-CPU-Modul ist normalerweise ein

```
bootloader -d /dev/ttyUSB0 -b 115200 -p aplikation-file.hex -v
```

ausreichend. Bei 115200 Baud geht das Flashen auch größerer Programme recht flott.

Falls schon ein Anwenderprogramm aktiv ist, wartet das Bootloaderprogramm auf einen Reset des Zielsystems

```
Waiting for device... Press CTRL+C to exit.
```

Nun den hoffentlich vorhanden Resetbutton am Board drücken. Ihre Konsolenausgabe sollte dann etwa wie folgt aussehen:

```
zoli@dubhe $ bootloader -d /dev/ttyUSB0 -b 115200 -p main.hex -v

================================================
|             BOOTLOADER, Target: V2.1          |
================================================
COM dev  : /dev/ttyUSB0
Baudrate : 115200
Program  : main.hex
------------------------------------------------
Waiting for device... Press CTRL+C to exit. |
...Connected!
Bootloader    : V2.1
Target        : 1E9502 ATmega32
Buffer        :   1024 Bytes
Size available:  31744 Bytes
CRC enabled and OK.
Reading main.hex... File read.
Highest address in hex file: 0x00183 (387 Bytes)
Writing program memory...
Programming "main.hex": 00000 - 00183 SUCCESS

CRC: o.k.
Elapsed time: 0.119 seconds
...starting application

zoli@dubhe ~/develop/mc/k-software/cpu-modul/Demo $
```

Mit CTRL+C kann das Bootloaderprogramm abgebrochen werden. Sofern noch Fragen oder weiterer Informationsbedarf bestehen, können sie unter den genannten Links Hilfe finden.

Weitere Unterlagen und Links

- **Homepage der Mikrocontroller Projektgroup (mcPG):**
  http://localhost/~zoli/mcpg/data/index.html

  Die K-Modul Serie:
  Allgemeines und Grundlagen zu den K-Modulen

  K-CPU-Modul Aufbauanleitung V2.0:
  Anleitung zum Aufbau K-CPU-Moduls V3.02

  K-CPU-Modul Boardbeschreibung (*ENTWURF*):
  Beschreibung der Funktionen und Schnittstellen des K-CPU-Moduls

  K-UART-Modul Boardbeschreibung (*ENTWURF*):
  Beschreibung der Funktionen und Schnittstellen des K-UART-Moduls

  USB-Prog Howto:
  Kurzeanleitung zur Programmierung des USBprog von Benedikt Sauter

- **Im Internet:**

  AVR-Fuse-Calkulator:
  Internetseite zum berechnen des Werte für die Fuse-Bytes
  http://www.engbedded.com/fusecalc/

  AVR-Tutorial:
  http://www.wiki.elektronik-projekt.de/mikrocontroller/avr/avrdude

[1] Bootloader fastboot von Peter Dannegger:
http://www.mikrocontroller.net/articles/AVR_Bootloader_FastBoot_von_Peter_Dannegger

[2] fastboot_build26:
http://www.mikrocontroller.net/topic/146638#1578698

[3] EPJ Ausgabe 4/2009:
http://www.ep-journal.de/archiv/EPJ_04_download.pdf

[4] build29:
http://mcpg.augusta.de/mcprog/kboot/fastboot_build29.tar.gz

[5] http://www.avrfreaks.net/index.php?module=Freaks%20Files&func=viewFile&id=3839&showinfo=1

[6] http://mcpg.augusta.de/mcprog/kboot/index.html

[7] Definitionfiles:
http://attiny.com/definitions.htm

[8] http://attiny.com/software/AVR000.zip

[9] http://www.embedded-projects.net/

[10] http://mcpg.augusta.de/data/kmodule/k-uart-modul/kuartA.html

Wir bauen einen Rechner für Embedded Linux!

Hubert Högl <Hubert.Hoegl@hs-augsburg.de>

Vorgeschichte

Seit ein paar Jahren verwende ich an unserer Hochschule zur praktischen Ausbildung im Fach „Embedded Linux“ das NGW100 Board [1] mit dem AVR32 „AP7000“ Controller und war damit auch immer ziemlich glücklich. Neben den technischen Eigenschaften war vor allem der Preis von etwa 50 Euro im Hochschulprogramm der Firma Atmel eine ideale Voraussetzung, damit sich Studenten das Board kaufen und selber nach Herzenslust damit experimentieren können.

Leider wird der AP7000 von Atmel nicht mehr weiterentwickelt, da er auch sowas wie eine hauseigene Konkurrenz zu den Controllern mit ARM9 Kern darstellte. Den AP7000 als einziges Mitglied einer anfangs grösser angelegten Familie wird es weiterhin zu kaufen geben, auch das NW100 (nun teurer!), aber das war's dann auch. Stark vermisse ich zum Beispiel einen USB Host Controller und auch eine etwas höhere Taktfrequenz wäre schön, um zum Beispiel den Übergang von cross- auf native Kompilierung zu ermöglichen. Auch schwindet die Anwendergemeinde um den AP7000, was in einem Teufelskreis natürlich auch nicht motivierend auf mögliche neue Nutzer wirkt.

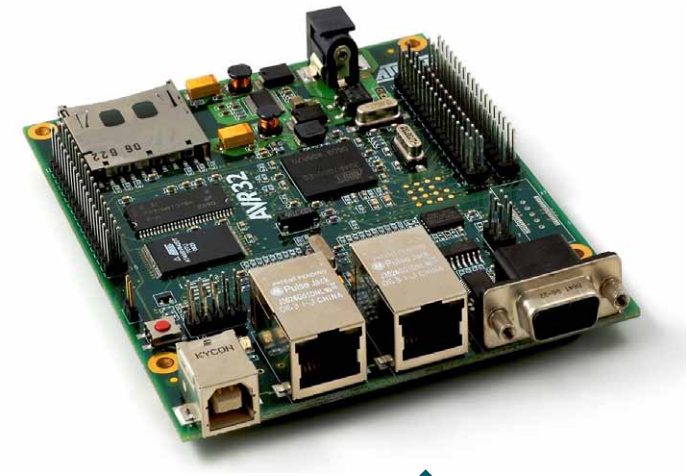


Abb. 1: Das „NGW100“ Board von Atmel

Der Plan

Im vergangenen Sommer war der Plan dann schnell gefasst. Wie wollen aus eigener Anstrengung ein Board bauen, hier sind in Stichpunkten die Rahmenbedingungen:

- Es soll ein freies Gemeinschaftsprojekt werden, das auch bei den Entwicklungswerkzeugen das Wörtchen „frei“ ernst nimmt. Entwurfsdaten und Software werden mit einer freien Lizenz veröffentlicht.
- Der Mikrocontroller soll schnell sein (200 bis 400 MHz) und mit einem ARM Kern ausgestattet sein, weil darauf die meisten Projekte aus der Embedded Linux Welt portiert sind, z.B. Echtzeit Erweiterungen wie die RT-Preemption oder Xenomai. Diese gab es leider für den AP7000 nicht.
- Der Controller soll viele Schnittstellen haben wie UARTs, SPI, I2C, Ethernet und wenn möglich auch CAN.
- Das Board soll sich gut für nachträgliche Erweiterungen eignen. Es sollen nur elementare Pfostenfeldleisten im 2.54 mm Rastermass Verwendung finden, so dass man mit „Flugdrähten“ oder „wire-wrap“ Drähten als Bastler einfach Erweiterungen anschliessen kann.
- Das Board soll mit etwa 7 x 7 cm ziemlich klein werden, so dass man auch kleinere tragbare Geräte damit bauen kann die aus einem Akku gespeist werden.
- Das Board soll im einfachsten Fall über eine einzige USB Schnittstelle zu betreiben sein. Darüber läuft die Spannungsversorgung, die Konsole (UART) und das Debuggen über JTAG (optional). Das wird durch einen separaten Board Controller AVR Mega32U4 mit USB device Anschluss möglich.
- Die Schnittstellen nach aussen sollen möglichst üppig sein: USB host, USB device, Ethernet, CAN, I2C, SPI, mehrere UARTs.
- Es soll mindestens 64M SDRAM vorhanden sein. Der Flash Speicher besteht aus einem seriellen DataFlash für das U-Boot (Bootloader) und einer Micro-MMC/SD Karte für den Linux Kern und das Root Filesystem.
- Der Preis soll bei 50 Euro liegen, das ist meine gefühlte Akzeptanzgrenze bei Dingen, die sich Studenten, ohne grösser darüber nachzudenken, kaufen (vergleichbar mit einem etwas teureren Buch).

Die praktische Umsetzung

Wir haben uns für den Controller i.MX287 von Freescale [7] entschieden. Der i.MX28 hat einen ARM926 Kern der bis zu 450 MHz schnell ist. Die Peripherie ist sehr üppig ausgestattet, wie man in der folgenden Abbildung 2 sieht. Ethernet und CAN sind sogar jeweils doppelt vorhanden. Bei USB gibt es einen USB high-speed OTG Anschluss und einen USB high-speed Host Anschluss.

Im Bereich „Power Management“ fällt auf, dass der Controller für die drei benötigten Spannungen 3,3V, 1,8V und 1,2V sowohl Spannungsregler (LDO) als auch DC/DC-Schaltregler und einen Laderegler für Li-Ionen Akkus bereits integriert hat. Damit könnte die Realisierung eines mit Akku betriebenen Gerätes, zumindest vom Hardwareaufwand gesehen, sehr einfach werden. Es gibt auch einen LCD Controller für TFT Displays und

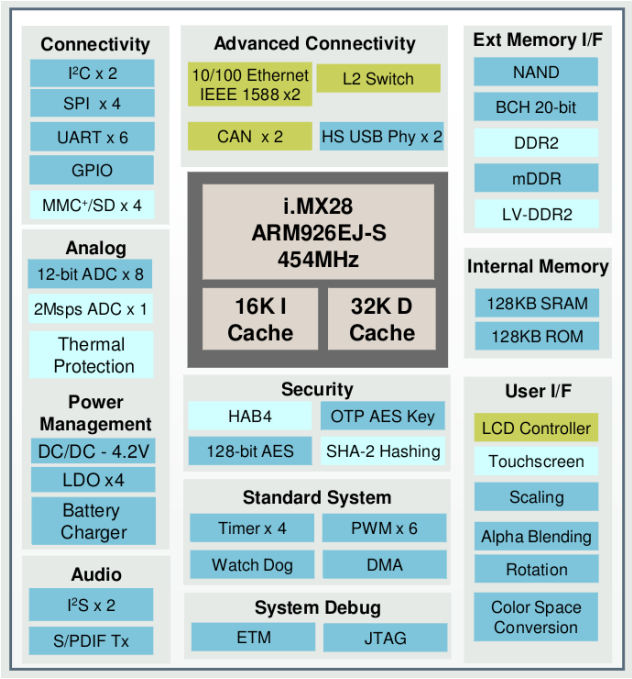


Abb. 2: Das „NGW100“ Board von Atmel

einen Anschluss für Touch Sensoren.

Das folgende Blockschaltbild (Abb. 3) zeigt, was alles auf das Board soll. Es wird nur einen DDR2 SDRAM Baustein geben mit 64 MByte Kapazität. Beim Flash Speicher gibt es nur einen seriellen DataFlash Baustein, der den U-Boot Bootloader enthalten wird, optional noch den Linux Kern und ein minimales Root Filesystem. Das „richtige“ Betriebssytem, z.B. Debian für ARM wird man immer aus einer Micro SD Karte starten, dafür ist eine Kartenhalterung vorgesehen.

Jede der beiden Ethernet Schnittstellen hat einen PHY Baustein auf dem Board, eine geht auf eine Ethernet Buchse (mit Pins für „Power over Ethernet“, PoE), die andere geht mit den Ethernet Signalen auf einen Erweiterungsstecker. Eine der CAN Schnittstellen hat einen 3,3V Transceiver dessen CAN Signalen auf einen Erweiterungsstecker gehen. Der andere CAN Kanal geht direkt auf den Stecker.

Es gibt 6 Pfostenfeldleisten, jede mit 2 x 20 Pins, insgesamt hat man also 240 Pins (120 oben, 120 unten).

Das Board könnte mal in etwa so aussehen, wie es die folgende Abbildung zeigt (leider ist das Platinenlayout noch nicht fertig). Die Pfostenfeldleisten für die Erweiterungen sind oben und unten jeweils aussen auf drei Seiten der Platine angeordnet. Oben sind andere Signale als unten!

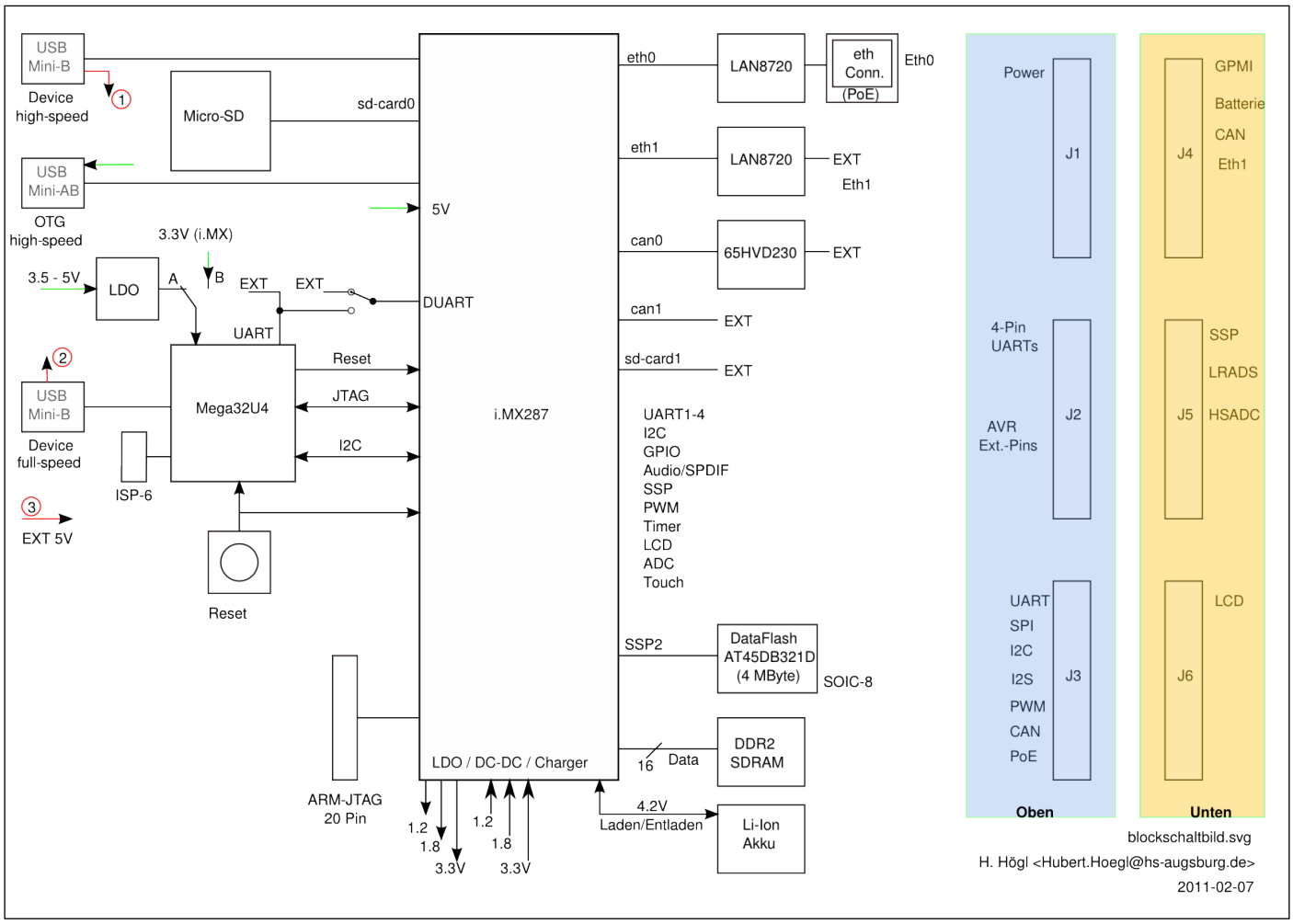


Abb. 3: Das Blockschaltbild unseres geplanten Boards

Linux für den i.MX28

Mit Linux für den i.MX28 sieht es sehr gut aus. Freescale erachtet die Unterstützung für Linux als sehr wichtig. Auf der Freescale Homepage zum i.MX gibt es die Seite „i.MX28 Software and Development Tool Resources“, dort findet man alles Wichtige zum Download für Linux.

Die Firma Pengutronix [5] kümmert sich um die notwendigen Änderungen für diesen Controller und pflegt diese auch vorbildlich in den Mainline Kernel ein. Den aktuellen Pengutronix-Kernel gibt es in diesem git Repository:

git://git.pengutronix.de/git/imx/linux-2.6.git

Zum Erstellen des Root Filesystems verwendet Freescale die Bau Umgebung „LTIB“ (Linux Target Image Builder) [6]. Daneben kann man auch jedes andere Buildsystem verwenden, das Root Filesysteme für ARM Controller erzeugen kann, also z.B. Buildroot und OpenEmbedded. Auch die etablierten grossen PC Distributionen wie Debian [8], Fedora und Ubuntu gibt es mittlerweile mit einem grossen Paketumfang für Controller mit ARM9 Kern.

Auch der U-Boot Bootloader versteht sich mit der i.MX Controller Familie, das Repository ist hier:

git://git.denx.de/u-boot-imx.git

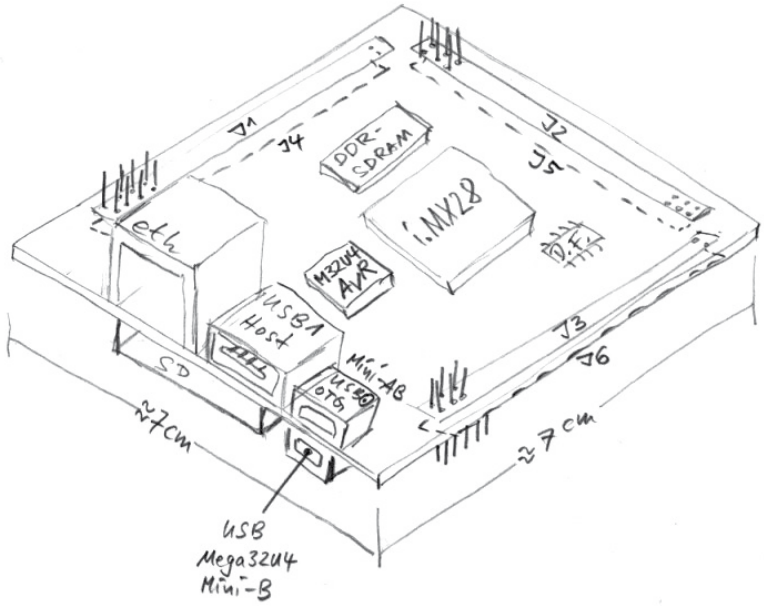


Abb. 4: Handzeichnung des Boards, bitte nicht zu ernst nehmen

Aktueller Stand und weiteres Vorgehen

Zur Zeit ist der Schaltplan [3] schon ziemlich weit fortgeschritten. Vermutlich könnte im März der Platinenplan fertig sein. Wir arbeiten übrigens mit dem freien EDA Programm „KiCad“ [4].

Der Zeitplan dürfte in etwa so sein:

- Einige Musterplatinen herstellen lassen (März).
- Manuelle Bestückung von ein bis zwei Musterplatinen. Auf Lötgerät für BGA Sockel (Bausteine i.MX28 und DDR-SDRAM) können wir zugreifen.
- Inbetriebnahme und Test der Musterplatinen (April, Mai).

- Falls wir zu viele Fehler eingebaut haben, ist wahrscheinlich eine neue Iteration mit KiCad notwendig.
- Automatische Bestückung einer grösseren Menge bei einer Bestückungsfirma. Hier ist noch alles offen.

Wer will mitmachen?

Wer in irgendeiner Form mitmachen möchte, vielleicht auch helfen kann, darf sich gerne bei mir melden. Alle weiteren Informationen stehen im Wiki [2]. Es gibt auch eine Mailing-Liste, siehe [9].

Wir haben ausserdem noch keinen richtigen Namen für das Projekt. Verschiedene Leute haben „Gnublin“, „Elinuxboard“, „Kaktux“ (meine Idee, das ging in die Hose :-) vorgeschlagen. Wer einen schönen Namen weiss, der darf ihn mir gerne verraten.

Links

[1] Das „reife“ NGW100 Board:
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4102

[2] Das „Gnublin“ Wiki:
http://elk.informatik.fh-augsburg.de/hhwiki/Gnublin

[3] Aktueller Schaltplan:
http://elk.informatik.fh-augsburg.de/kaktux-cdrom/eda/index.html

[4] KiCad:
http://kicad.sourceforge.net/wiki/Main_Page

[5] Pengutronix:
http://www.pengutronix.de

[6] LTIB:
http://www.ltib.org

[7] Freescale i.MX28:
http://www.freescale.com/webapp/sps/site/overview.jsp?code=IMX28_FAMILY

[8] Debian für ARM:
http://www.debian.org/ports/arm/index.en.html

[9] Mailing-Liste des Projektes: elinuxboard@fh-augsburg.de:
https://www.rz.fh-augsburg.de/sympa/info/elinuxboard

Danksagung

Die Firma Freescale hat mir für das Projekt freundlicherweise ein i.MX287 EVK gespendet. Dankeschön!

Freut euch auf die nächste Ausgabe:

Web-basiertes Messen, Steuern, Regeln mit dem eNet-sam7X

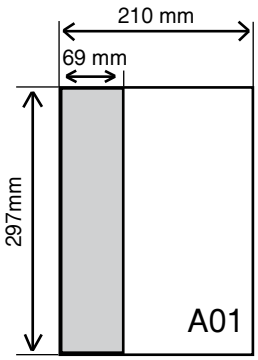
Wer glaubt das Internet besteht nur aus eMail, Facebook und YouTube etc, der verpasst die aufregende Welt der embedded Ethernet Geräte. Wie wäre es, wenn man seine Geräte von überall auf der Welt kontrollieren kann? Die Solaranlage mit dem Smart-Phone vom Strand aus steuern, die Heizung zum Feierabend vom Büro aus einschalten und die Rolläden automatisch steuern? [...]

Und genau darum geht es hier...

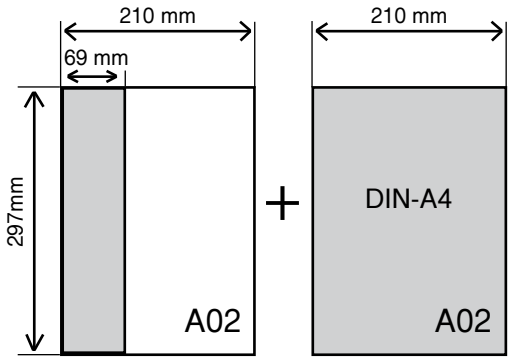
Ihre Anzeige im embedded projects Journal

Ansprechpartner:
embedded projects GmbH
Holzbachstraße 4
D-86152 Augsburg
Tel. +49(0)821 / 279599-0
Fax: +49(0)821 / 279599-20
Mail: journal@embedded-projects.net

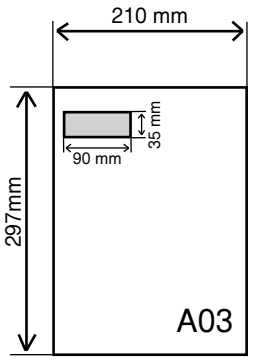
A01: 1/3 Seite



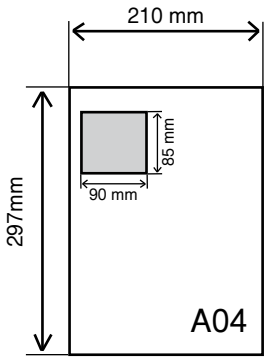
A02: Hauptsponsor



A03: Jahressponsor



A04: Würfel



embedded projects Journal:

Das embedded projects Journal ist eine Zeitschrift, die unter einer Open-Source Lizenz steht. Alle Artikel und die Zeitschrift können so unter Einhaltung dieser Lizenz weiter verarbeitet werden. Das besondere an dem Open-Source Projekt Journal ist, dass die Zeitschrift nicht nur frei als PDF-Version verfügbar ist, sondern von den Lesern kostenlos abonniert werden kann. Druck und Porto werden durch Anzeigen von Sponsoren finanziert.

Aktuell wird jede Ausgabe ca. 8000 mal online und ca. 2000 mal als Papierversion gelesen.

Anzeigenformate und Preisliste:

Bezeichnung	Bezeichnung	Format in mm Breite x Höhe	Preis*	Erscheinung in
A01	1/3 Seite	69 x 297 mm	150 €	1 Ausgabe
A02	Hauptsponsor	69 x 299 mm + 210 x 297 mm	400 €	1 Ausgabe
A03	Jahressponsor **	90 x 35 mm	100 €	4 Ausgaben
A04	Würfel	90 x 85 mm	100 €	1 Ausgabe

* Alle Preise zzgl. Ust.
** Im Angebot inbegriffen: 1. Frei Haus Lieferung jeder Ausgabe.
2. Ihre Anzeigen wir zusätzlich auf unserer Homepage unter: http://www.ep-journal.de im Firmenverzeichnis veröffentlicht.

Top Angebot

1/3tel Seite
69 x 297 mm
150€ pro Ausgabe *
bei Buchung von 4 Ausgaben 20% Rabatt

Ausgabe	Anzeigenschluss	Erscheinungsmonat
01 / 2011	15.02.2011	März
02 / 2011	15.05.2011	Juni
03 / 2011	15.08.2011	September
04 / 2011	15.11.2011	Dezember
01 / 2012	15.02.2012	März
02 / 2012	15.05.2012	Juni
03 / 2012	15.08.2012	September
04 / 2012	15.11.2012	Dezember

Bestellfax an +49(0)821 / 279599-20

☐ Anzeige in nächster/n _____ Ausgabe veröffentlichen
☐ A01 ☐ A02 ☐ A03 ☐ A04

☐ Anzeige veröffentlichen in / ab Ausgabe ___ / ____

Firma: _____
Vor- / Nachname: _____
Anschrift: _____
_____
Tel. / E-Mail: _____

embedded - projects.net
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Bitte frei machen

embedded projects GmbH
Holzbachstraße 4
D - 86152 Augsburg

Name / Firma

Straße / Hausnummer

PLZ / Ort

Email / Telefon / Fax

○ Ich möchte jede zukünftige Ausgabe erhalten

○ Wir möchten als Hochschule / Ausbildungsbetrieb jede weitere Ausgabe bekommen.
Bitte gewünschte Anzahl der Hefte pro Ausgabe ankreuzen. 5 10

○ Ich möchten im embedded projects Journal werben oder eine Stellenanzeige aufgeben. Bitte schicken Sie mir Infomaterial, Preisliste etc. zu.

Werdet aktiv!

Das Motto: Von der Community für die Community !
Das Magazin ist ein Open Source Projekt.
Falls du Lust hast, Dich an der Zeitschrift durch einen Beitrag zu beteiligen, würden wir uns darüber sehr freuen. Schreibe deine Idee an:
sauter@embedded-projects.net

Regelmäßig

Die Zeitschrift wird über mehrere Kanäle verteilt. Der erste Kanal ist der Download als PDF - Datei. Alle Ausgaben sind auf der Internetseite [1] verfügbar. Diejenigen, die lieber eine Papierversion erhalten möchten, können den zweiten Kanal wählen. Man kann sich dort auf einer Internetseite [1] in eine Liste für die gesponserten Abos eintragen. Beim Erscheinen einer neuen Ausgabe wird dank Sponsorengeldern an jeden auf der Liste eine Ausgabe des aktuellen Journal versendet. Falls man den Versandtermin verpasst hat, kann man das Heft auch zum Preis von einem Euro über einen Online - Shop [2] beziehen.
[1] Internetseite www.embedded-projects.net/epjournal
[2] Online - Shop: http://www.eproo.de/

Sponsoren gesucht!

Damit wir weiterhin diese Zeitschrift für jeden frei bereitstellen können, suchen wir dringend Sponsoren für Werbe- und Stellenanzeigen. Bei Interesse meldet Euch bitte unter folgender Telefonnummer: 0821 / 2795990 oder sendet eine E-Mail an die oben genannte Adresse.

Impressum

embedded projects GmbH
Holzbachstraße 4
D-86152 Augsburg
Telefon: +49(0)821 / 279599-0
Telefax: +49(0)821 / 279599-20

Veröffentlichung: 4x / Jahr
Ausgabenformat: PDF / Print
Auflagen Print: 2500 Stk.
Einzelverkaufspreis: 1 EUR

Layout / Satz:
Akzente Media | www.akzente-media.de

Titelfoto: fotolia.de, ioannis kounadeas

BESSER GLEICH ONLINE KALKULIEREN.

STARRE UND FLEXIBLE LEITERPLATTEN.

ab 12 h Leiterplatten-Express! Pünktlich oder gratis!

LEITON
RECHNEN SIE MIT BESTEM SERVICE

Endlich wird's einfach und Sie bleiben flexibel. Den umständlichen und aufwändigen Kalkulationsprozessen machen wir einen dicken Strich durch die Rechnung. Sie kalkulieren Ihre Leiterplatten online – ganz bequem, ganz schnell. **Einzigartig: Die Online-Kalkulation gilt auch für Serien und flexible Leiterplatten.** Das macht uns weltweit so schnell keiner nach. Einmalig ist zudem der **Leiterplatten-Expressdienst** von LeitOn mit unserer Garantie: Wenn wir bei Expressdiensten den vereinbarten Übergabetermin an den Versender nicht einhalten können, schenken wir Ihnen die bestellten Platinen! Sie möchten mehr darüber wissen? Wir bieten persönliche Beratung am Telefon und einen kompetenten Außendienst. Sie können bei LeitOn immer mit dem besten Service rechnen.

www.leiton.de | **kontakt@leiton.de** | Info-Hotline **+49 (0)30 701 73 49 0**