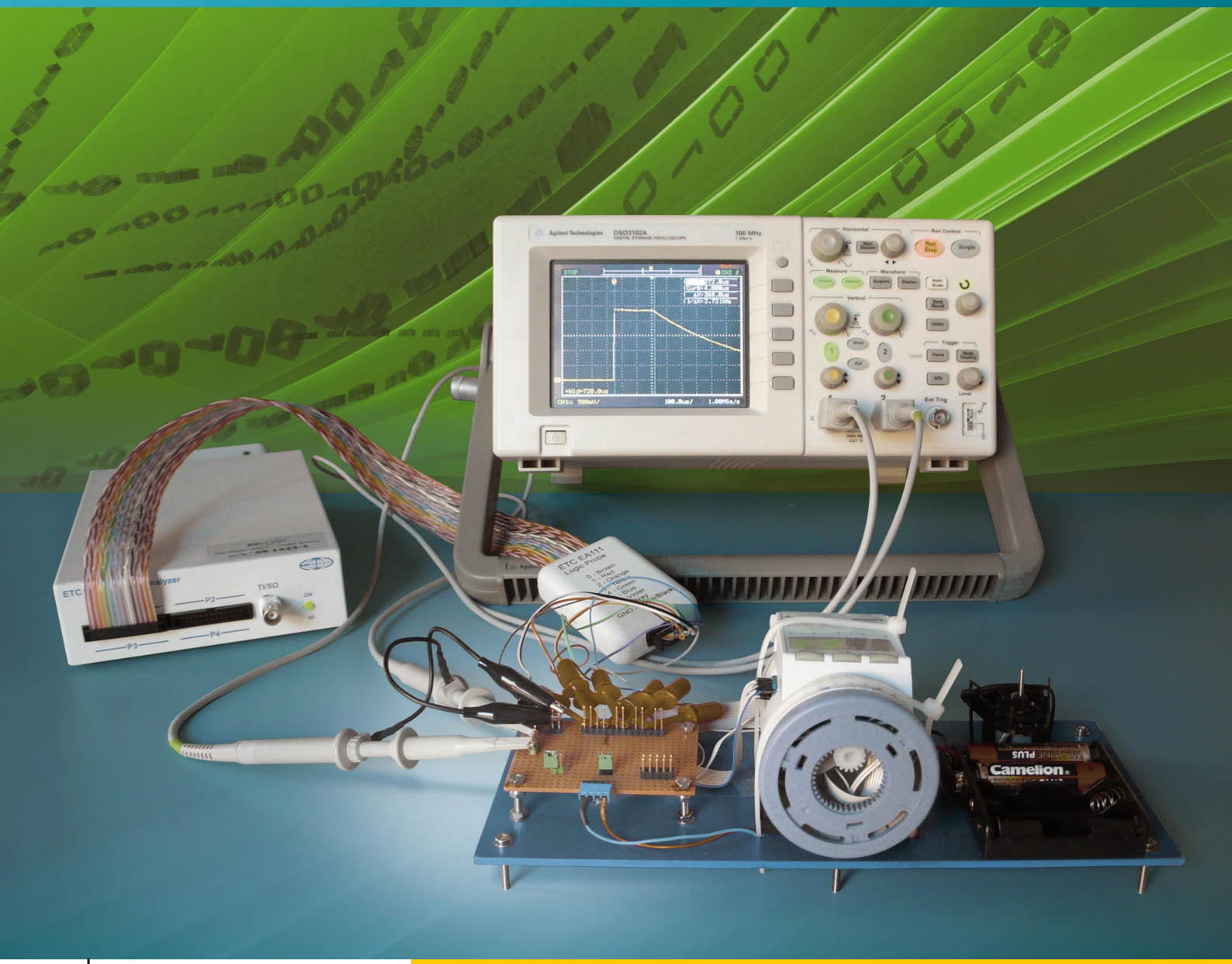
embedded-projects.net
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS



OPEN HR20

Open Soure Projekt:
Elektronisches Heizkörper-thermostat ansteuern

- **Embedded Linux, Python, eLua und ein Logikanalysator**
- **C-A-T Handsteuerinterface**
- **Grasshopper: GPIOs über ein Webinterface steuern**
- **FPGAs von A wie Actel bis X wie Xilinx**
- **Digitales PC-Spektrometer**

EPJ *No.* 06

WELCOME

Ausgabe No.6 - Embedded-Projects Journal

In der Kürze liegt die Würze

Eine Einleitung muss schnell her,
doch das Thema finden ist sehr schwer.

Über Fliegen und die Freien,
über Technik und Scharlatanereien.

Berichtet wurde bereits über vieles,
keines war jedoch so dringend wie dieses.

Wichtigste Botschaft dieses Heftes,
wir machen weiter und tun das Beste!

Etwas im Verzug für Leser,
wird es nächstes Jahr viel besser.

Eine Internetseite für die Zeitschrift,
hilft uns bei der Weitsicht.

Planung ist das halbe Leben,
so soll es mehr Aktivität um die Zeitschrift geben.

Artikel schreiben, Anzeigen kaufen,
Kommentare abgeben oder Sonstiges sich belaufen.

Eine Seite für uns alle,
vom Professor bis zum Kalle.

Hoffen wir, es wird uns helfen,
regelmäßig eine Ausgabe zu entwerfen.

In diesem Sinne: www.ep-journal.de

Eine Internetseite für das Open-Source Projekt Zeitschrift:

- Aboadresse ändern
- Artikel schreiben
- Spendenanzeigen kaufen
- Sponsor werden
- Stellenmarkt

Benedikt Sauter
sauter@embedded-projects.net

shop.embedded-projects.net

THE ONLINESHOP - HARDWARE FOR YOUR PROJECTS

ARM Boards

ARM USB Debugger (ARM-USB-Tiny)

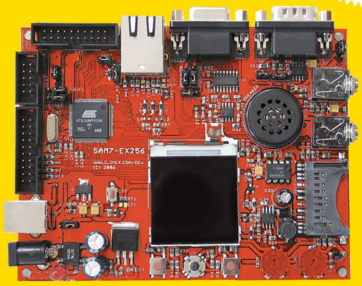
ARM7/ARM9/Cortex-M3 und XScale Debuginterface für OpenOCD und CrossWorks

ARM USB Debugger
44,90 €
embedded-projects SHOP

AT91SAM7X256+TFT+Ethernet (SAM7-EX256)

Bestückt mit einem 32 Bit ARM-Mikrocontroller mit 256 kB Flash, 64 kB RAM, 55 MHz, Ethernet 10/100, USB 2.0, RS232, CAN, MMC/SD-Card Slot und TFT-Display

AT91 SAM7X256 TFT+ Ethernet
119,90 €
embedded-projects SHOP



MSP430 Boards

MSP430F1611 Adapterplatine (MSP430-H1611)

MSP430F1611 mit 48K Bytes Programm Flash, 256 Bytes Daten Flash, 10K Bytes RAM

MSP430F1611 Adapterplatine
24,90 €
embedded-projects SHOP

MSP430 USB JTAG Adapter (MSP430-JTAG-TINY)

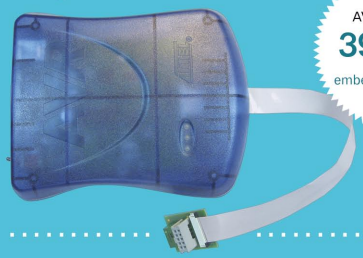
Programmierung/ Debugging für alle MSP430Fxxx Flash-Microcontroller

MSP430 USB JTAG Adapter
64,90 €
embedded-projects SHOP

AVR Boards

Atmel AVRISP mkII (USB)

Original AVRISP mkII In-System Programmer von Atmel.



Atmel AVRISP mkII
39,90 €
embedded-projects SHOP

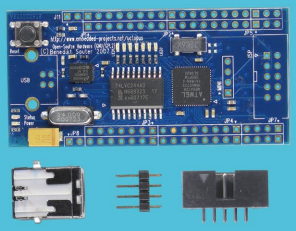
AVR Starterkit (inkl. USBprog, Netzteil und ATMega8)

AVR-Starterkit bestehend aus USBprog, AVR-Starterplatine, ATMega8 und Steckernetzteil.

SET AVR Starterkit
61,90 €
embedded-projects SHOP

Octopus USB

Octopus bietet viele bekannte Schnittstellen aus der Mikrocontrollerwelt über ein einfaches USB Gerät an.

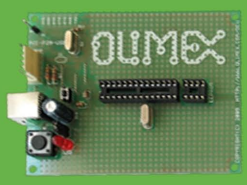


Octopus USB
39,00 €
embedded-projects SHOP

PIC Boards

PIC Entwicklungsplatine (PIC-P28-USB)

PIC Mikrocontroller Entwicklungsboard für 28-polige ICs + USB RS232.



PIC Entwicklungsplatine (PIC-P28-USB)
24,90 €
embedded-projects SHOP

AVR32 Boards

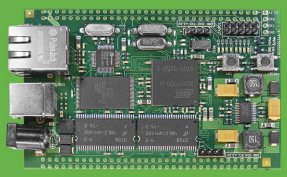
ATNGW100 Network Gateway Kit

ATNGW100 Network Gateway Kit
89,90 €
embedded-projects SHOP

Grasshopper AVR32-Board

*Open-Source-Version (ohne CD+Kabel)

freie Plattform für die AVR32 Entwicklung



Grasshopper AVR32-Board*
ab 85,00 €
embedded-projects SHOP

Jetzt schneller zum Shop:
www.eproo.net

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

Jetzt Neu:
Versand weltweit
3,95 Euro

Embedded Linux, Python, eLua und ein Logikanalysator

Hubert Högl, 15. Februar 2010, <Hubert.Hoegl@hs-augsburg.de>, http://www.hs-augsburg.de/~hhoegl

Es ist wieder mal Zeit, über ein paar Neuigkeiten aus meinem Umfeld in der Technischen Informatik an der Hochschule Augsburg zu berichten. Der Titel sagt schon grob um was es geht, zum Glück nicht in einem, sondern in den folgenden vier Projekten.

1. Embedded Linux und Python

In meiner Embedded Linux Veranstaltung sind öfter Leute, die sich erhoffen, mit einem Mikrocontroller-Board wie dem NGW100 unter Linux sofort loslegen zu können um Steuerungsaufgaben zu erledigen. So war das auch bei einem Mechatronik-Studenten letzten Sommer, der einen selbstgebauten Roboter (ein „WallE" Nachbau, [4]) damit zum Leben erwecken wollte. Dabei ging es hauptsächlich um die Ansteuerung von digitalen I/O Ports und um das Erzeugen von etwa einem Dutzend PWM Signalen für Servomotoren. Die Ernüchterung war gross, nachdem er sah, wie viele Hürden zu nehmen sind bis allein schon die von Linux bereits unterstützten Schnittstellen das Gewünschte machen. Dazu kommen dann noch die neu zu programmierenden Treiber für Sonderwünsche, wie z.B. „Soft-PWM" Ausgänge, die PWM über Software an gewöhnlichen Ausgangspins erzeugen. Das war nötig, weil über ein Dutzend PWM Ausgänge für Servos anzusteuern sind - soviel Hardware-PWM Ausgänge haben die Controller üblicherweise nicht.

Der Traum von der schnell realisierten Linux Anwendung ist also für viele schnell ausgeträumt. Hinzu kommt, dass häufig verwendete Embedded Linux Baukästen wie Buildroot oder OpenWRT für viele Neulinge am Anfang nur frustrierend sind. Es kommen einfach zu viele komplizierte Gebiete zusammen wie Kommandozeilenbedienung, Bootloader, Bootvorgang, Linux-Administration, Netzwerktechnik, Gerätetreiber, Kernel-Schnittstellen, User-Space Programmierung, autoconf/automake und so weiter. Zumindest versteht man nach dieser Erfahrung, warum Dienstleistungsunternehmen für Embedded Linux wie www.denx.de oder www.pengutronix.de ganz guten Zulauf haben.

Die Frage war nun, wie man (ohne Beratungsfirma) das Schreiben einer Anwendung deutlich vereinfachen und somit die Produktivität erhöhen könnte. Auf dem Desktop Rechner wäre die Antwort klar: Nimm doch eine richtige High-Level Sprache, wie z.B. Python! - könnte die Antwort lauten. Irgendwann in Python's Entstehungsgeschichte hat jemand das Motto „Batteries Included" erfunden. Damit ist die Standardbibliothek gemeint, die bei jeder Python Installation von Haus aus dabei ist [1]. Damit hat man interaktiv vom Python Prompt aus Zugriff auf eine riesige Menge an Algorithmen und Diensten.

Die Idee war nun naheliegend, die typischen Hardware-Schnittstellen bei einem Embedded Linux Rechner durch entsprechende Python Module nutzbar zu machen. Als wichtigste Schnittstellen wurden UART, I2C, SPI, GPIO, PWM, Software-PWM und CAN identifiziert.

Diese Aufgabe hat Volker Thoms in seiner Diplomarbeit bearbeitet, die nun fertig ist und unter [2], im Verzeichnis da-49 zur freien Verwendung steht. Er hat als Linux Baukasten die OpenWRT Distribution benutzt, die man an genannter Stelle im Verzeichnis trees/ findet. Was herauskam sieht tatsächlich ganz vielversprechend aus. Die meisten Schnittstellen lassen sich nun bequem aus Python ansteuern, ohne genauere Kenntnis des zugrunde liegenden Linux.

Die einzige Schnittstelle die noch externe Hardware benötigt ist die für CAN. Wir verwenden einen Microchip MPC2515 CAN Controller der an einer SPI Schnittstelle des AVR32 hängt. Die CAN Programmierschnittstelle sieht Dank des verwendeten Socketcan Projektes [3] ähnlich aus wie die altbekannte sockets Netzwerkschnittstelle. Was noch fehlt, ist die Anpassung des C Moduls socketmodule.c aus der Python Bibliothek für socketcan. Das sollte aber kein grösseres Problem sein, so dass es in Zukunft auf dem Python Prompt nur noch import socketcan benötigt, um mit CAN loszulegen.

Hat das ganze auch Nachteile, könnte man nun fragen. Mit Sicherheit ist die Geschwindigkeit einer Anwendung in Python um den Faktor 10 bis 100 langsamer als eine in C geschriebene. Grob gesagt passiert in C im Bereich einiger 10 Mikrosekunden schon ziemlich viel, bei Python landet man für die selbe Aufgabe schon im Bereich von ein paar Millisekunden, was aber sehr oft ohne Probleme zu verschmerzen ist. Natürlich fühlt sich der Python Interpreter um so wohler, je mehr Rechenleistung die CPU hat und je grösser der Arbeitsspeicher ist. Der AVR32 auf dem NGW100 liegt bei Python eher im minimal erforderlichen Bereich. Ein ordentlicher Leistungsschub wäre z.B. mit den neueren ARM926 Controllern zu erreichen, die mit bis zu 400 MHz laufen, z.B. mit dem Atmel SAM9G20 oder SAM9G45. Der Code sollte sich ohne grössere Schwierigkeiten auf diese ARM Architektur portieren lassen.

Mir gefällt dieser Ansatz so gut, dass ich gerade eine vor längerer Zeit in C selbstgeschriebene Heizungssteuerungs-Anwendung nach Python übertrage. Die zeitlichen Anforderungen sind dabei naturgemäss eher unkritisch. Wer könnte sich noch vorstellen, mit diesem Ansatz eine Anwendung zu programmieren? Bitte schickt mir eine E-mail.

2. Noch ein paar Experimente mit dem NGW100

Andreas Waffler hat in seiner Diplomarbeit ([2], Verzeichnis da-48) einige Experimente mit dem NGW100 gemacht. Er beschreibt anschaulich, wie der Boot-Vorgang funktioniert und wie man die Schnittstellen des Boards aktiviert. Die folgenden I/O-lastigen Experimente hat er zunächst gemacht:

- Ein ATtiny44 wird über I2C als „neuer" Board-Controller angeschlossen. Mit ihm kann man die Versorgungsspannung des Boards als auch die Stromaufnahme messen. Letzteres wurde mit einem INA139 I-zu-U Konverter gemacht. Natürlich kann der AVR für beliebige Interface-Aufgaben verwendet werden, z.B. zum Einlesen eines Helligkeitswertes oder einer Temperatur. Beide Sensoren können an einen Analogeingang des AVR angeschlossen werden. Der AVR32 holt sich die Werte wie gewohnt über I2C.

- Der 1-wire Bus [8] wird eingehend untersucht, und zwar zunächst der GPIO Treiber w1-gpio, dann zwei Lösungen mit dem Maxim DS2482 I2C-zu-1wire Baustein und dem Maxim DS9097 Seriell-zu-1wire Baustein.

- Mit Hilfe der libconfig [7] wir eine Konfigurationsdatei erstellt, mit der die Eigenschaften der über I2C und 1-wire angeschlossenen Busse und Sensoren beschrieben werden. Das Programm read-sensor „weiss" dadurch, welche Sensoren es lesen kann. Die gemessenen Werte werden mit Hilfe eines Skriptes ausgewertet und über Gnuplot grafisch dargestellt.

Dann kommen die Experimente mit „Funk":

- Mit Hilfe von zwei Funkmodulen mit 802.15.4 Standard wird die UART Schnittstelle ttyS1 des NGW100 drahtlos mit der seriellen Schnittstelle eines PC verbunden. Am PC erwartet einen wie gewohnt der Login-Prompt des NGW100. Die Funkmodule heissen „ICradio Mini 2.4G", es gibt sie wie zu erwarten in Benedikt Sauter's Shop.

- Das GSM Modem („Handy-Modem") Wavecom M2106 wurde an die freie serielle Schnittstelle ttyS2 des AVR32 angeschlossen und damit demonstriert, wie man SMS- und FAX-Nachrichten verschickt und empfangt.

- Danach beschreibt die Arbeit, wie man einen DCF77 Empfänger von Pollin Elektronik, Nr 810054 am NGW100 betreibt. Damit hat man eine extrem genaue Zeitangabe, mit der man die Systemuhr stellen kann.

Danach untersucht Herr Waffler noch die Energiespareigenschaften des NGW100 mit Hilfe der oben genannten I-zu-U Messschaltung. Er geht auf ACPI ein, demonstriert die „suspend-to-ram" Betriebsart und weckt das Board wieder mit rtcwake auf. Am Schluss gibt die Arbeit eine Zusammenfassung über die Buildroot-Umgebung.

3. Lua auf dem SAM7X

Ähnlich wie bei dem vorherigen Projekt geht es in der Diplomarbeit von Matthias Hagl um die einfache Anwendbarkeit eines Mikrocontrollers durch eine interpretierte Sprache, die direkt auf dem Controller läuft. Die Sprache ist nun aber nicht Python, sondern Lua [5], genauer gesagt eLua [6], der „embedded" Variante von Lua. eLua hat im Vergleich zu Python den Vorteil, dass es sich mit einer wesentlich geringeren Hardware-Ausstattung zufrieden gibt. Man braucht weder einen riesigen SDRAM Speicher noch die Rechenleistung für ein grosses Betriebssystem. Aktuell läuft eLua auf vielen ARM7 und Cortex M3 Controllern die etwa 200 KByte Flash und ein paar 10 KByte RAM haben. Wir haben uns für den Atmel SAM7X256 entschieden, weil wir das Board „SAM7-EX256" von Olimex haben.

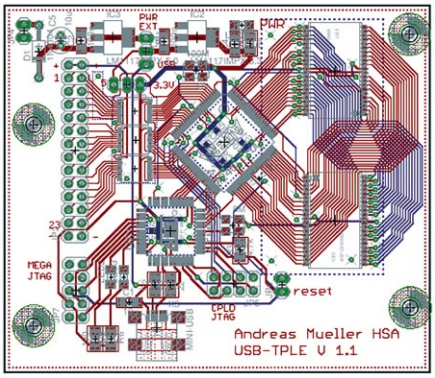
In der Diplomarbeit wurde der damals aktuelle Stand von eLua (Version 0.5) untersucht und Treiber für bisher noch nicht unterstützte Schnittstellen wie analoge Eingänge, SPI und TWI (= I2C) geschrieben. Daneben gab es bereits Treiber für USART, SSC, TIC, PWM, PIO und Netzwerk.

Die Programmierung des Mikrocontrollers mit eLua hat einen hohen Spassfaktor, da man die Programme im Quelltext sofort über die serielle Schnittstelle in den Controller laden und ausführen kann. Alternativ kann man die Programme auch direkt im Flash-Speicher ablegen. Der folgende Code zeigt einen kleinen Ausschnitt aus einem Lua Programm, bei dem Pins initialisiert werden:

```
function define_pins()
    backlight_lcd = pio.PB_20
    resetpin_lcd = pio.PA_2
    npcs0 = pio.PA_12
    miso = pio.PA_16
    mosi = pio.PA_17
    spck = pio.PA_18
end

function init_lcd()
    pio.output(backlight_lcd, resetpin_lcd, miso, mosi, spck)
    pio.set(backlight_lcd, resetpin_lcd)
    spi.setup(0, spi.MASTER, 125000, 1, 0, 9)
    spi.select(0, 0)
end

function init()
    define_pins()
    tmr.delay(0,1000)
    init_lcd()
end
```

Diese Arbeit können Sie unter [2] im Verzeichnis da-42 herunterladen. Wer sich in die Sprache Lua auf dem Desktop Rechner einarbeiten möchte, der braucht nur die Pakete lua und lua-doc unter Linux zu installieren.

4. Logikanalysator



Andreas Müller hat in seiner noch andauernden Bachelorarbeit eine Schaltung für einen einfachen USB-fähigen Logikanalysator entworfen. Es ist ein AVR Mega32U4 drauf, ein Altera MAX2 CPLD und ein paar Megabyte schnelles SRAM. Daraus soll ein Messgerät werden, das sich für elementare Messungen an digitalen Signalen in der Lehre eignet, vor allem denke ich an Timing-, Protokoll-, Logik- und Eventanalyse. Hier sind ein paar Anwendungsmöglichkeiten:

- Oft ist in meinem Labor eine Mikrocontroller-Platine an einen Server angeschlossen, auf dem man „remote" arbeitet. Eine Aufgabe könnte sein, an mehreren Ausgabepins ein „Lauflicht" zu programmieren. Mit dem angeschlossenen Messgerät könnte man sich den tatsächlichen zeitlichen Verlauf ansehen (Logik-Analyse).
- Bei einem kleinen Projekt wird eine UART-, SPI- oder I2C-Schnittstelle an einem Mikrocontroller in Betrieb genommen und man möchte sich über das generierte Timing durch eine Messung informieren (Protokoll-Analyse).
- Manchmal möchte man den zeitlichen Ablauf von Funktionsaufrufen genau untersuchen. Dazu fügt man am Anfang und am Ende von den interessierenden Funktionen sogenannten „Instrumentationscode" ein. Das sind oft nur ein paar winzige Ausgabebefehle (oft in Assembler), die von einem externen „Event-Recorder" mit Zeitstempeln aufgenommen werden. Später kann man den zeitlichen Verlauf z.B. in einem Gantt Diagramm genau rekonstruieren (Event-Analyse).
- Bei grösseren Betriebssystemen wie Embedded Linux sind die zeitlichen Abläufe - vor allem die Antwortzeiten auf Interrupts - eine Untersuchung wert. Ein simples Rechtecksignal, das man auf einem Parallelport ausgibt, kann einen deutlichen „Jitter" von hunderten Mikrosekunden aufweisen. Ausgestattet mit Echtzeit-Erweiterungen reduziert sich dieser Jitter auf wenige Mikrosekunden. Unser Messgerät soll auch solche Messungen erlauben (Timing-Analyse).

Herr Müller baut gerade die Platinen zusammen und kümmert sich um die VHDL-Programmierung des CPLD. Durch Ändern des CPLD Inhaltes über die USB Schnittstelle sollen die oben kurz vorgestellten Messfunktionen realisierbar sein. Wie die Software mit schicker GUI auf dem PC aussehen soll, ist noch ziemlich offen und gehörte erst mal nicht zum Umfang dieser Bachelorarbeit. Wer hat gute Ideen?

Das „Trac" zum Projekt ist hier:
http://sta.informatik.fh-augsburg.de/projects/sta/

5. Links

[1] http://www.python.org/doc/
[2] http://elk.informatik.fh-augsburg.de
[3] http://developer.berlios.de/projects/socketcan
[4] http://thomasboegle.de/walle.htm
[5] http://www.lua.org
[6] http://www.eluaproject.net
[7] http://www.hyperrealm.com/libconfig
[8] http://www.maxim-ic.com/products/1-wire

C-A-T Handsteuerinterface

Jugend Forscht Projekt von Phil Stelzer, Ole Stecker-Schürmann, Tobias Markus

Projektbetreuer: Herr Berthold Sommer
Aus dem Projektlabor des Berufskolleg der Stadt Rheine
Projektstand: In der Entwicklung

Berufskolleg der Stadt Rheine

Dieser Artikel beschreibt unsere bisherigen Ergebnisse mit dem CAT System, welches sich momentan in der Entwicklung befindet.

Die Computermaus hat sich, seit ihrer Vorstellung im Jahre 1968, als einfaches, universelles und intuitives Eingabemedium etabliert. Sie ist grundsätzlich als Eingabegerät für zweidimensionale Systeme konzipiert. Zur Steuerung von mehr als zweidimensionalen Systemen ist sie nur bedingt geeignet.

Für die Steuerung komplexer Anwendungen im dreidimensionalen Raum wird ein Eingabemedium benötigt, welches die Fähigkeit besitzt, auch mehrdimensionale Eingaben zu erfassen und weiterzuleiten.
Hier stellen wir ein universelles Computereingabesystem vor, mit dem intuitiv komplexe Objekte im dreidimensionalen Raum (X, Y, Z Richtung) gesteuert werden können.
Der „Mauszeiger" wird ähnlich wie bei einer klassischen Computermaus intuitiv über die Handposition navigiert. Zusätzlich zur Position ist auch die Lage (Drehen über die drei Achsen eines ausgedehnten Körpers) beeinflussbar.
Mit CAT können Körper oder Objekte im Raum einfach und intuitiv gesteuert werden.

Mehrdimensionale Eingabesysteme

Entweder werden drei oder bis zu sechs Freiheitsgrade gesteuert, die Position und/oder die Lage im Raum. Besonders im CAD-Bereich haben sich die 3D Mäuse (z.B. von Logitech) etabliert.

Diese Systeme sind Kabelgebunden und können somit nur an Tischen genutzt werden. Für Präsentationen und den mobilen Einsatz sind sie weniger geeignet. Andere Systeme können nur drei Freiheitsgrade erfassen und benötigen einen aufwändigen Sensoraufbau (so zum Beispiel: http://www.wpi.edu/news/20067/popsciaward.html).

Klein, Handlich, Flexibel

Mit dem hier vorgestellten System steuern wir sechs Freiheitsgrade und sind unabhängig von örtlichen Gegebenheiten.
Unser Eingabeinterface beschränkt sich auf ein kleines, handliches Modul, welches einfach an der Hand getragen wird. Die komplette Sensorik befindet sich in diesem Handinterface. So ist ein zusätzlicher Sensoraufbau für die Positionserfassung im Raum nicht erforderlich. Die Sensordaten überträgt das System per Funk an den Rechner. Dort werden die Daten mit einem speziellen Programm ausgewertet und an das zu steuernde System übergeben.
Weitere Funktionen können über Tasten und Schieber angesteuert werden.

1. Die Anforderungen

Als Grundlage für unsere Arbeit dient folgende Anforderungsliste.

Anforderungen an das Gesamtsystem

Folgende Funktionen werden gesteuert:
- Die Position von Objekten in 3- Dimensionen
- Die Rotation eines mehrdimensionalen Objektes um die X/Y/Z Achse
- Zusatzfunktionen durch Knöpfe und Schieber

Daraus ergeben sich folgende Funktionen der Hardware:
- Die X/Y/Z Position und die Drehung der Hand wird detektiert und an den Rechner übertragen
- Tasten und Schieber stehen zur Verfügung (frei konfigurierbar)
- Um Flexibilität und Bewegungsfreiheit für den Benutzer zu gewährleisten, werden alle Sensordaten per Funk übertragen
- Energiesparende Hardware, um eine maximale Laufzeit bei geringer Baugröße zu erreichen

Daraus ergeben sich folgende Anforderungen an die Software:
- Einfache Handhabung des Programms (selbsterklärend)
- Auslesen der USB Schnittstelle
- Übergeben der Handposition an den Mauszeiger

2. Funktionsbeschreibung

Die kleine Handelektronik erfasst die relative Positionsänderung und die Winkelgeschwindigkeit der Hand. Zusätzlich können durch Eingabeelemente weitere Funktionen gesteuert werden.

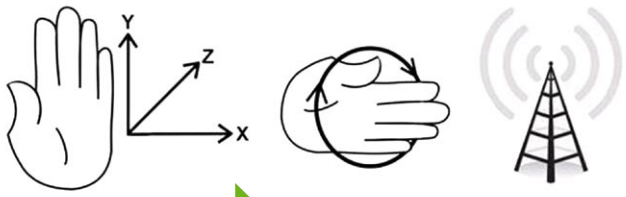
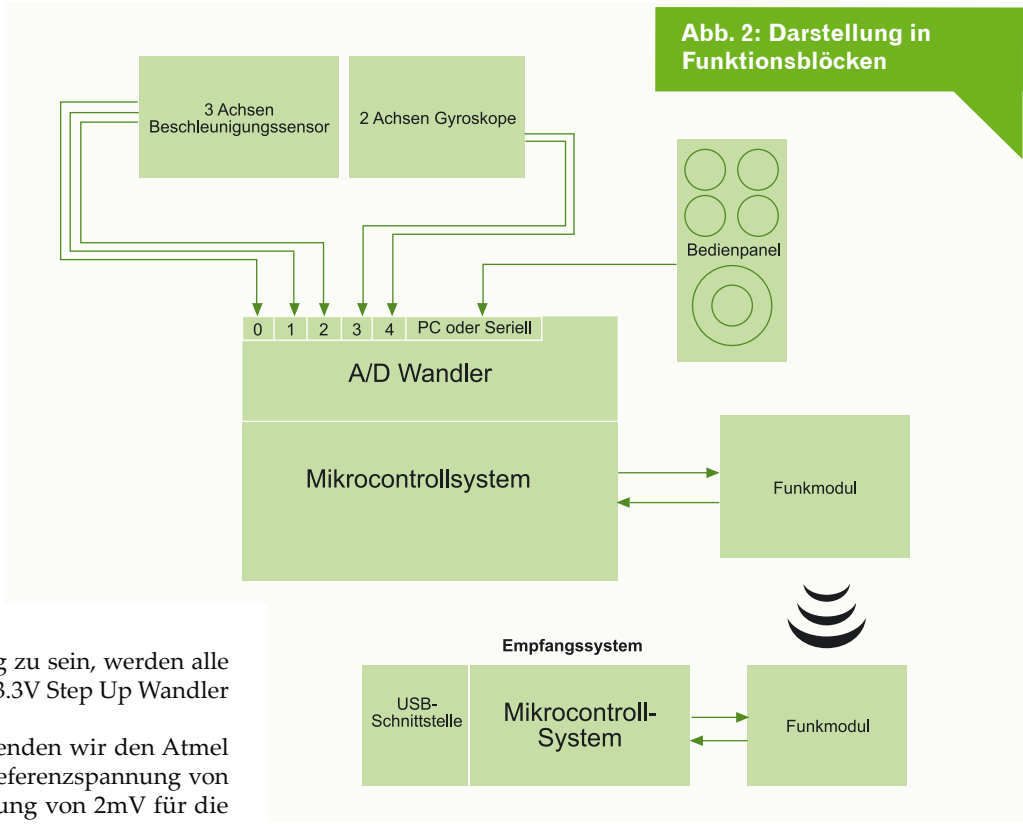


Abb. 1: Bildliche Darstellung der Funktionen

Alle Funktionen werden von einem Mikrocontrollersystem koordiniert. Zur Erfassung der relativen Positionsänderung wird ein 3-Achsen-Beschleunigungssensor verwendet. Die Winkelgeschwindigkeit wird von einem Gyrosensor gemessen. Alle diese Sensoren werden über den A/D-Wandler des Mikrocontrollers ausgelesen und die Daten per Funk an den Empfänger übermittelt. Die Spannungsversorgung wird von einer 3V-Knopfzelle bereitgestellt.

Der Empfänger übergibt die Daten über den USB- Anschluss an den Rechner, wo sie dann weiterverarbeitet werden.

Abb. 2: Darstellung in Funktionsblöcken



3. Hardware

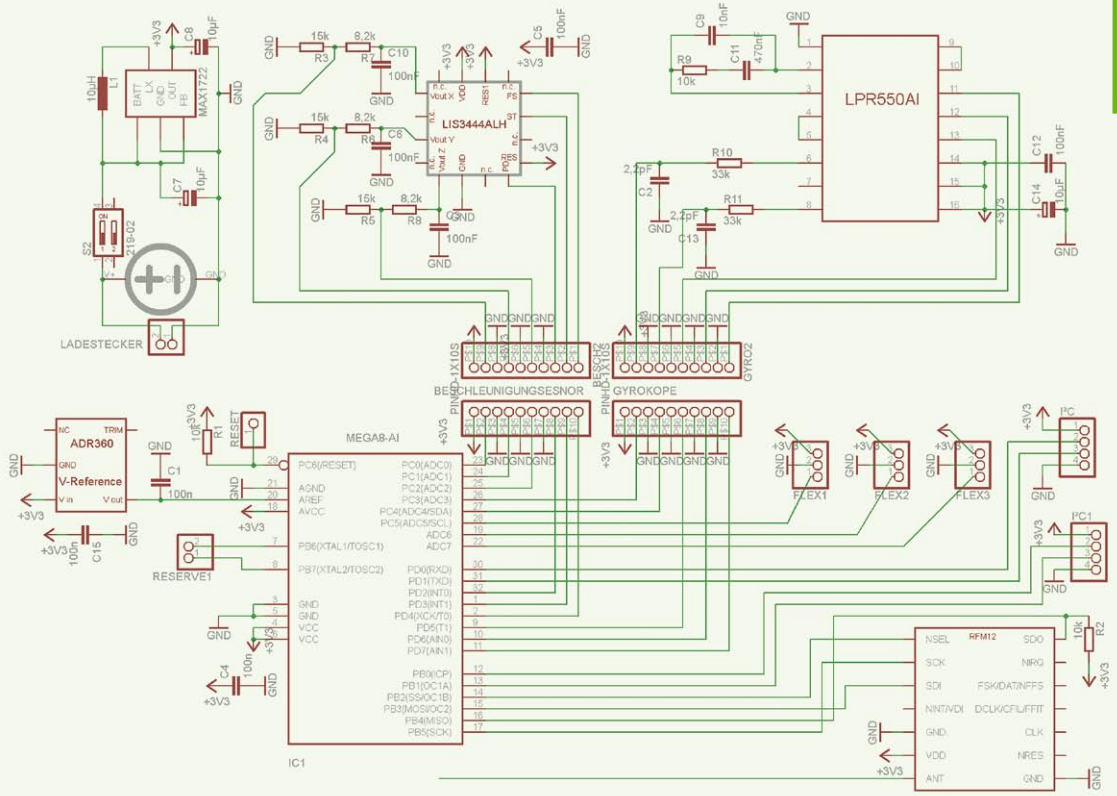
Um unabhängig von der Batteriespannung zu sein, werden alle Komponenten von dem MAX1722, einem 3.3V Step Up Wandler von Maxim, mit Spannung versorgt.
Als zentrale Mikrocontrollereinheit verwenden wir den Atmel Mega8. Für die A/D Wandler wird eine Referenzspannung von 2,048V genutzt, welches uns eine Auflösung von 2mV für die Messung ermöglicht. Die Referenzspannung wird durch den ADR360 geliefert.
Als Beschleunigungssensor nutzen wir den LIS344ALH einen 3-Achsen-Beschleunigungssensor, welcher auf die Messbereiche ±2g oder ±6g eingestellt werden kann. Zum Erfassen der Winkelgeschwindigkeit findet ein 2-Achsen-Gyroskop der LPR550AL mit einem Messbereich von ±500°/s Verwendung.

Als Funkmodul verwenden wir das RFM12 Modul (mit aufgelöteten 1k Ohm Pull Up Widerstand zwischen VDD und FSK), welches über das SPI vom Mikrocontroller angesteuert wird.

Außerdem wurden alle weiteren zur Verfügung stehenden Pins des Controllers für Erweiterungen rausgeführt.

Der Empfänger besteht ebenfalls aus dem Mega8 und dem RFM12 Funkmodul. Die USB-Kommunikation wird durch den FT232RL von FTDI Chips realisiert. Die so generierte virtuelle Serielle Schnittstelle kann bequem über C# oder andere Programmiersprachen ausgelesen werden. Die Spannungsversorgung für dieses Modul wird über den USB Port zur Verfügung gestellt.

Abb. 3: Schaltplan Handsteuer-interface



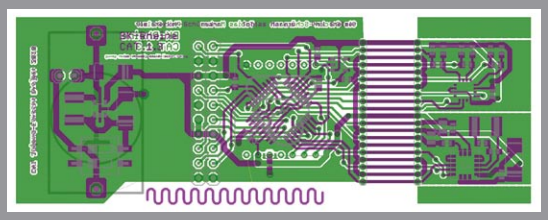


Abb. 4: Layout Handinterface

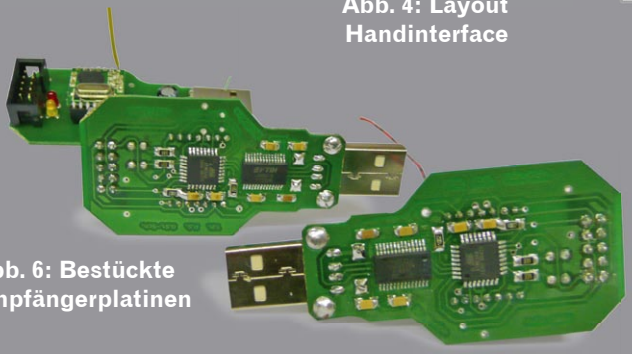


Abb. 6: Bestückte Empfängerplatinen

Abb. 5: Empfänger

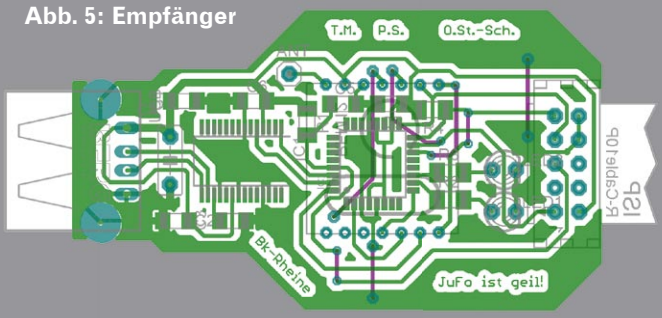
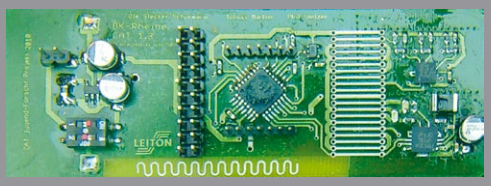


Abb. 7: Platine Handsteuerinterface



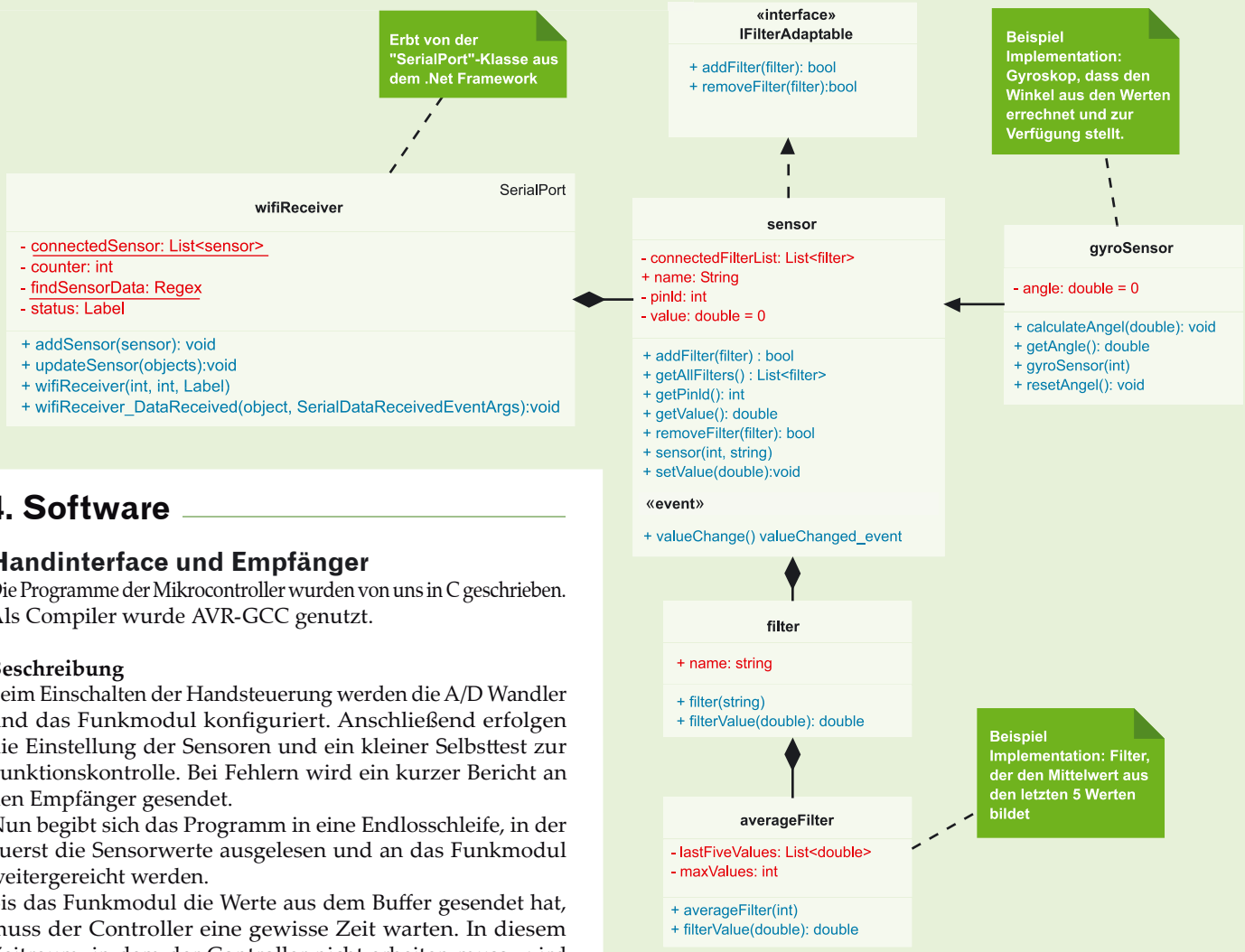


Abb. 8: UML- Klassendiagramm

4. Software

Handinterface und Empfänger

Die Programme der Mikrocontroller wurden von uns in C geschrieben. Als Compiler wurde AVR-GCC genutzt.

Beschreibung

Beim Einschalten der Handsteuerung werden die A/D Wandler und das Funkmodul konfiguriert. Anschließend erfolgen die Einstellung der Sensoren und ein kleiner Selbsttest zur Funktionskontrolle. Bei Fehlern wird ein kurzer Bericht an den Empfänger gesendet.

Nun begibt sich das Programm in eine Endlosschleife, in der zuerst die Sensorwerte ausgelesen und an das Funkmodul weitergereicht werden.

Bis das Funkmodul die Werte aus dem Buffer gesendet hat, muss der Controller eine gewisse Zeit warten. In diesem Zeitraum, in dem der Controller nicht arbeiten muss, wird er von uns in einen Sleep Modus geschickt. Hierdurch kann der Strombedarf des Controllers während dieser Zeit auf 1/3 gesenkt werden.

Anwendung

Die Clientanwendung wurde von uns in C# programmiert. Als Programmierumgebung nutzen wir Visual Studio 2008.

Beschreibung

Die Anwenderseite unserer CAT-Handsteuerung wird durch Klassen bereitgestellt, die den einfachen und schnellen Zugriff auf verschiedene Sensoren zulassen. Außerdem können Filter definiert werden, welche empfangene Werte aufbereiten, bevor sie am Sensorobjekt ankommen.
Die Klasse „wifiReceiver" hat die Hauptaufgabe, den ankommenden Datenstrom des Serial Ports zu entschlüsseln und die geschickten Daten an die entsprechenden Sensoren zu übergeben, die vorher als Objekte einer Liste im „wifiReceiver" hinzugefügt wurden.

Funktionstest

Im Folgenden führen wir ein Beispielprogramm auf, welches die Funktionen des Handinterfaces visualisiert. (Abb.9)

Ein kleines Video von ersten Tests der Teilfunktionen.
http://www.youtube.com/watch?v=QLsyTELEIgM

Abb. 9: Programm zum Prüfen der Funktionen

5. Resümee

Wir haben es bisher geschafft, einen Prototyp zu fertigen, zu programmieren und die wichtigsten Funktionen in Betrieb zu nehmen.
Aufbauend auf diese erreichten Ziele können wir als nächstes konkrete Anwendungen programmieren, aber auch noch einige Auffälligkeiten am Platinenlayout verbessern.

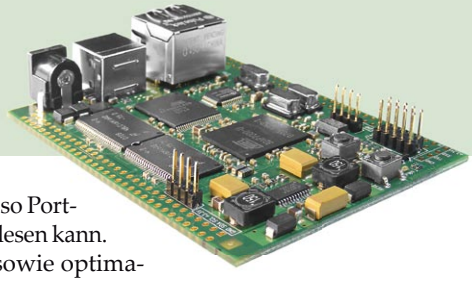
Bei Fragen, Ideen, Vorschlägen und auch Kritik zu dem Projekt können wir unter info@messfix.bkr-projekt.de erreicht werden.

6. Quellen

[1] http://de.wikipedia.org/wiki/Computermaus
[2] http://www.maxim-ic.com/quick_view2.cfm/qv_pk/3024
[3] http://www.atmel.com/dyn/resources/prod_documents/doc2486.pdf
[4] http://www.analog.com/static/imported-files/data_sheets/ADR360_361_363_364_365_366.pdf
[5] http://www.st.com/stonline/products/literature/ds/14337/lis344alh.pdf
[6] http://www.st.com/stonline/products/families/sensors/datasheets/lpr550al.pdf
[7] http://www.hoperf.com/pdf/rfm12.pdf
[8] http://www.mikrocontroller.net/articles/RFM12
[9] http://www.3dconnexion.de/

Grasshopper: GPIOs über ein Webinterface steuern

Christian Zietz



Die folgende Anleitung soll zeigen, wie man die GPIOs (General Purpose Inputs/Outputs, also Port-Pins) des Grasshopper- bzw. ICnova-P7000-Base-Boards über ein Webinterface setzen und auslesen kann. Vorausgesetzt werden Kenntnisse über die grundlegenden Linux-/Unix-Kommandos sowie optimalerweise minimale Kenntnisse des einzigen auf dem Board installierten Editors vi.
Die in dieser Anleitung erwähnten Dateien finden sich nach Kapiteln sortiert in grasshopper-gpio.tgz.

1. Inbetriebnahme

Für diese Anleitungen verbinden wir das Board sowohl über USB mit dem PC als auch über Ethernet mit dem Netzwerk. Über den USB-Anschluss wird das Board mit Strom versorgt. Außerdem bietet eine virtuelle serielle Schnittstelle dar¨uber sowohl die Boot-Meldungen als auch eine Konsole. Details zur Nutzung der seriellen Schnittstelle nennt die Projektseite des Grasshoppers. Die Ethernetschnittstelle wird standardmäßig nicht beim Booten aktiviert. **ACHTUNG:** So lange das Board über USB am PC angeschlossen ist, ist eine externe Stromversorgung über die vorhandene Buchse weder notwendig noch sinnvoll. Soll das Board jedoch später einmal an einem Netzteil betrieben werden, weist In-Circuit darauf hin, dass der Aufdruck 7-20V auf der Platine falsch ist.Es dürfen keinesfalls mehr als 10V angelegt werden.

Im Folgenden wird davon ausgegangen, dass das Board nun über die Konsole auf der virtuellen seriellen Schnittstelle angesprochen werden kann.Wir loggen uns mit dem Benutzernamen default (ohne Passwort) ein und geben dann das Kommando su ein, um als root zu arbeiten. Es gibt noch ein paar weitere Kleinigkeiten, die wir zunächst korrigieren wollen:

- Damit die Ethernetschnittstelle beim Booten aktiviert wird und sich das Board von einem DHCP-Server automatisch eine IP-Adresse besorgt, muss die Datei /etc/network/interfaces editiert werden (mit vi). Wir entfernen die Raute (#) vor der Zeile

```
auto eth0
```

  Einziger Nachteil dieser Änderung: Ist kein DHCP-Server auffindbar, verzögert sich der Boot-Vorgang leicht. Andernfalls müssten wir jedoch nach jedem Login auf der seriellen Konsole ein ifup eth0 eingeben, um die Ethernetschnittstelle zu aktivieren.

- Um die Meldung can't access tty: job control turned off beim Login loszuwerden, ändern wir in /etc/inittab die Zeile

```
null::respawn:/sbin/getty -L console 115200 vt100
```

  in

```
null::respawn:/sbin/getty -L /dev/ttyS0 115200 vt100
```

  Ein weiterer Vorteil: Nun ist an der seriellen Konsole auch der direkte Login als root möglich. **ACHTUNG:** Sollte die inittab beschädigt werden, ist evtl. kein regulärer Login mehr möglich. Die Lösung in diesem Fall: Beim Booten bei der Meldung Press SPACE to abort autoboot in 3 seconds die Leertaste drücken und dann im Bootloader U-Boot die folgenden Kommandos eingeben:

```
setenv bootargs $bootargs init=/bin/sh
boot
```

  Damit landet man nach dem Booten einmalig direkt in einer Shell und kann die inittab reparieren.

- Bei manchen Grasshoppern ist offenbar die Datei /var/www/index.html beschädigt. Dies führt beim Booten zu der MeldungUnknown node type: e002 len 766 offset 0x508da8. Auch liefertder Webserver dann keine Startseite aus. Eine reparierte index.html ist in den Dateien zu dieser Anleitung enthalten.

2. Dateien mit dem Board austauschen

Es gibt verschiedene Wege, Dateien mit dem Board auszutauschen. Standardmäßig läuft ein Webserver, der die Dateien unterhalb von /var/www bereitstellt. Auf dem Board ist wget installiert, mit dem Dateien von einem Webserver auf das Board geladen werden können. Wir wollen jedoch den ebenfalls vorhandenen TFTP-Server verwenden. (Hinweis: Außer dem Namen hat TFTP (Trivial File Transfer Protocol) nichts mit dem bekannteren FTP gemeinsam.) Standardmäßig läuft der Server nicht. Wir starten ihn mit folgendem Kommando und begrenzen den Zugriff sicherheitshalber auf Dateien in /tmp:

```
in.tftpd -l -c -s /tmp
```

Windows und die gängigen Linux-Distributionen bringen einen TFTP-Client mit, so dass wir nun Dateien austauschen können. Um beispielsweise die Datei test.bin vom PC zum Board zu senden ist unter Windows folgender Aufruf auf der Kommandozeile nötig (IP-Adresse dabei durch die tatsächliche Adresse des Boards ersetzen):

```
tftp -i IP-Adresse PUT test.bin
```

Sollte im LAN bereits ein NFS-Server (Network File System) existieren, kann auch er zum Datenaustausch mit dem Grasshopper dienen. Dazu muss auf dem Board nur portmap gestartet werden. Danach ist ein Mounten von NFS-Freigaben möglich.

3. Einrichtung der GPIOs

Der Zugriff auf die Port-Pins erfolgt nach Unix-Art über eine Device-Datei. Diese legen wir mit dem Kommando

```
mknod /dev/gpio0 c 254 0
```

an. Außerdem muss nach jedem Neustart konfiguriert werden, welcher Port und welche Pins über diese Datei als Ein- und Ausgänge angesprochen werden sollen. In Anleitung werden wir die Pins PORTA00 (kurz PA00), PA01 und PA02 als Ausgänge und die Pins PA03, PA04 und PA05 als Eingänge konfigurieren. Diese Konfiguration erfolgt über Dateien in einem noch anzulegenden Unterverzeichnis von /config/gpio:

```
mkdir /config/gpio/gpio0
```

ACHTUNG: Die Zuordnung zwischen der Device-Datei und der eben angelegten Konfiguration erfolgt **nicht** über den Namen sondern über die Reihenfolge der Erstellung der Unterordner. Unsere Device-Datei /dev/gpio0 spricht immer die Konfiguration an, die als erste unter /config/gpio angelegt wird, unabhängig von deren Namen.

Der Ordner /config/gpio/gpio0 enthält vier Dateien, in die wir unsere Konfiguration schreiben müssen:

- gpio_id gibt den Port an. Hierbei steht 0 für Port A, 1 für Port B usw.
- pin_mask gibt an, welche Pins des Ports überhaupt angesprochen werden sollen. Die entsprechenden Bits sind zu setzen, der resultierende Wert muss als Hexadezimalzahl in pin_mask geschrieben werden. In unserem Fall wollen wir mit 0x3f die untersten 6 Pins ansprechen.
- oe_mask gibt an, welche Pins Ausgänge sein sollen. In dieser Anleitung steht 0x7 für die untersten 3 Pins.
- Schließlich muss in die Datei enabled eine 1 geschrieben werden, um die Konfiguration zu aktivieren.

Insgesamt ergeben sich damit folgende Kommandos:

```
mkdir /config/gpio/gpio0
cd /config/gpio/gpio0
echo 0 > gpio id
echo 0x3f > pin mask
echo 0x7 > oe mask
echo 1 > enabledcd /config/gpio/gpio0
echo 0 > gpio id
echo 0x3f > pin mask
echo 0x7 > oe mask
echo 1 > enabled
```

Da diese Kommandos nach jedem Neustart ausgeführt werden müssen, enthalten die Dateien zu dieser Anleitung das Shell-Skript gpio0. Dieses kopieren wir nach /etc/init.d, machen es mit chmod a+x gpio0 ausführbar und legen dann eine Verknüpfung darauf in /etc/rc.d an, damit das Shell-Skript automatisch beim Booten aufgerufen wird:

```
cd /etc/rc.d
ln -s ../init.d/gpio0 S20gpio0
```

Wenn das Blinken der LED1 stört, so ist dies der richtige Zeitpunkt es abzustellen. Hierfür geben wir ein bzw. erweitern /etc/init.d/gpio0 um:

```
echo none > /sys/class/leds/led1n:green/trigger
```

4. GPIOs setzen und lesen

Die gewählten Port-Pins können nun über die angelegte Device-Datei (/dev/gpio0) angesprochen werden. Beim Lesen dieser Datei werden nach dem Öffnen und danach nach jeder Änderung der mittels pin_mask gewählten Port-Pins vier Bytes (= 32 Bits) übermittelt, die den Status des Ports enthalten. Die über oe_mask als Ausgänge definierten Port-Pins können durch das Schreiben der Device-Datei gesetzt oder gelöscht werden. Auch hier müssen vier Bytes an binären Daten übergeben werden.

Um den Umgang mit GPIOs zu vereinfachen, liegt dieser Anleitung ein kleines C-Programm als Quellcode (gpio.c) sowie passend für das Board compiliert (gpio) bei. Wir kopieren die compilierte Variante auf dem Board in das Verzeichnis /usr/bin. Hier ein paar Beispiele für mögliche Aufrufe des Programms:

- `gpio /dev/gpio0 on 0`

  setzt das niederwertigste (nullte) Bit des konfigurierten Ports (in unserem Fall Port A) auf logisch high. Dies funktioniert natürlich nur bei als Ausgang konfigurierten Pins.

- `gpio /dev/gpio0 off 1`

  setzt den Pin Nummer 1 des konfigurierten Ports auf logisch low, sofern der entsprechende Pin ein Ausgang ist.

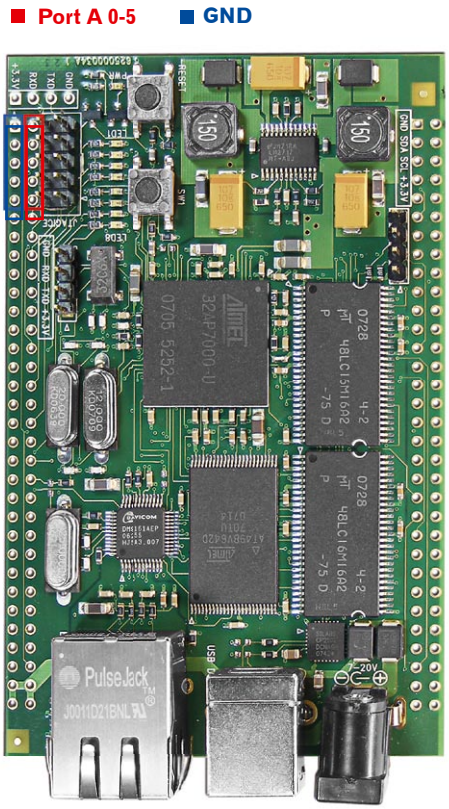
- `gpio /dev/gpio0 query 2`

  fragt den Zustand des Pins 2 ab. Dies funktioniert sowohl für Ausgänge als auch für Eingänge. Eingänge haben einen (schwachen) internen Pull-Up. Der Rückgabewert von gpio ist in diesem Fall 0, wenn der Pin high ist und 1, falls er low ist. (Hinweis: Diese scheinbare Verdrehung der Logik des Rückgabewerts erlaubt dann wieder die logisch schlüssigere Verwendung in if-Abfragen in Shell-Skripts.)

5. CGI-Skript für den Webserver einrichten

Der auf dem Board vorhandene und automatisch gestartete Webserver (bereitgestellt von busybox) unterstützt CGIs, d.h. die Generierung interaktiverWebseiten. Es ist allerdings wenig sinnvoll, wie auf einem "großen" Webserver Skripte in php, perl oder python für CGI-Programme zu verwenden. Wie jedoch bereits das auf dem Board vorhandene Beispiel (/var/www/cgi-bin/led.sh) zeigt, kann jede beliebige ausführbare Datei, also auch ein Shell-Skript, als CGI-Programm dienen. Die Kommunikation zwischen Webserver und Skript erfolgt, wie beim CGI üblich, über Umgebungsvariablen sowie über die Standardausgabe.

Dieser Anleitung liegt mit gpio.sh ein passendes Skript bei, um entsprechend der vorhin vorgenommenen Konfiguration die Pins 0 bis 2 des Ports A zu setzen und den Logikpegeln an den Pins 3 bis 5 zu lesen. Nachdem wir das Skript nach /var/www/cgi-bin kopiert und mittels chmod a+x gpio.sh ausführbar gemacht haben, können wir es auf dem PC im Browser als http://IP-Adresse/cgi-bin/gpio.sh aufrufen. Das beigefügte Foto zeigt die Lage der entsprechenden GPIOs, die sich nun über das Netz schalten und auslesen lassen:



6. Rechtliches

FPGAs von A wie Actel bis X wie Xilinx

von Hagen Sankowski

Auch in Embedded Systems sind sie bereits nicht mehr wegzudenken, Field Programmable Gate Arrays (FPGA). Sie kommen zum Einsatz in Nullserien für z.Bsp. Mobilfunkgeräte oder für Kleinauflagen wenn sich eine ASIC-Entwicklung aus Kosten- und/oder Zeitgründen verbieten würde. Oder, wenn Entwicklungsabteilungen noch in der höchst dynamischen Standardisierungsphase neuer Funktechnologien, z.Bsp. LTE, erste Prototypen testen wollen. Ihr Vorteil ist ihre Flexibilität. Dort wo gerade noch ein Microcontroller eine Finite State Machine abarbeitete, werkelt nun nach einem Firmwareupdate zum Beispiel ein FIR Filter. FPGAs profitieren massiv von der technologischen Entwicklung innerhalb der Dekaden zählenden Gültigkeit von Moore's Law. Aus der Anfängen der Glue-Logik, wie sie Progammable Logic Devices (PLDs) boten, längst entwachsen, sind in aktuellen FPGAs ganze Systeme, so genannte Systems-on-Chip, mit leistungsfähigem Controller plus zahlreicher Peripherie wie Memory Controller und High-Speed Serial Links, z.Bsp. PCIe oder XAUI, per Firmware unterzubringen.

Wir wollen mit diesem Artikel einen kühlen Kopf bewahren und Orientierungs- bzw. Entscheidungshilfen anhand übersichtlicher Tabellen bei der Auswahl des richtigen FPGAs geben.

Marktübersicht

Vor reichlich 20 Jahren herrschte Goldgräberstimmung. Patente wurde eingereicht, viele verschiedene Hersteller versuchten sich am Markt, wurden aufgekauft, abgewickelt, oder entwickelten sich weiter. In diesem Sinne gibt das FPGA-Kochbuch von Wannemacher [1] unfreiwillig nur noch einen historischen Überblick. Seit nunmehr einer guten Dekade sind vier höchst unterschiedliche Hersteller auf dem Markt aktiv, angeführt von zwei Platzhirschen und flankiert von zwei Spezialisten. Xilinx [2] sieht sich selbst dabei in der Vorreiterrolle, produziert ausschließlich SRAM basierte FPGAs und möchte immer wieder mit neuen Features zu trumpfen. Altera [3] ist oft gleichauf, ebenso im SRAM basierten FPGA Geschäft, und erreicht vergleichbare Größen an programmierbarer Logik erst mit der nächsten Technologiegeneration, bleibt aber dadurch weniger stromhungrig und vorallem preiswerter. Lattice [4] bietet bei den SRAM basierten den Flash gleich mit im Gehäuse; während bei der Konkurrenz die Konfiguration erst von einem externen Speicher gelesen werden muss und dabei mitgelesen werden kann, können ein paar Lattice dies intern. Als vierter in der Runde agiert Actel [5], mit einem Faible für Aeronautic und Military mit der mutmaßlich längsten Verfügbarkeit und neuerdings dem Ehrgeiz extremer Energieeffizienz.

Storagetechnologien

Die Art und Weise der Speicherung von Konfigurationen hat deutlichen Einfluss auf viele Designentscheidungen. Am weitesten verbreitet sind die SRAM basierten FPGA. D.h. es gibt neben dem FPGA selbst, meist noch in der gleichen JTAG-Chain, einen Flashspeicher aus welchem sich der FPGA die Konfiguration in den SRAM lädt. Dieser bootähnliche Vorgang ist natürlich mit Zeit verbunden, in welchem die Schaltung nach dem Reset noch nicht einsatzbereit ist. Derzeit sind Konfigurationen im Bereich mehrere Millionen Bytes völlig normal, da kann schon mal ein PCI-Bus die Einsteckkarte übersehen, weil der Timeout längst abgelaufen ist. Und, dieses Laden der Konfiguration erfolgt jedes mal nach dem Einschalten; mit einfachen Mittel lässt sich der Datenstrom zumindest mithören und klonen, mithin ein Sicherheitsrisiko, das nur dadurch abgemildert wird, das keiner der Hersteller den so genannten Bitstreams hinreichend dokumentiert hat um die Funktion nachvollziehen zu können. Wirklich richtig ist aber die bei einigen Familien angebotene Verschlüsselung des Bitsreams mit AES. Neben dem erhöhten Stromverbrauch für die SRAM-Zellen, ohne Strom verlieren die Zellen sofort ihren Inhalt und der FPGA damit seine komplette Konfiguration, fallen bei den SRAM basierten FPGA auch die zusätzliche Bauelemente wie z.Bsp. den Flashspeicher auf der BOM negativ auf. Es ist daher konsequent, den Flash gleich mit zu integrieren, wie es Lattice bei den XP Familien tut. Noch ein Plus in Sachen Abhörsicherheit, wenn die Konfiguration nicht mehr vom externen Speicher in den FPGA geladen wird.

SRAM basierte FPGA böten hingegen dann ein riesiges Potential, wenn es möglich wäre, zumindest Teile der Konfiguration quasi im laufenden Betrieb auszutauschen. Leider fehlen dafür die Tools, die dieses könnten; aufgrund der bereits genannten fehlenden Dokumentation wird es dabei wohl auch bleiben. Wenn aber FPGAs nicht mehr ständig umkonfiguriert werden müssen, sondern dies vielleicht nur bei Firmwareupdates passiert, also eher in überschaubarer Häufigkeit, dann macht es Sinn, statt vom Flash in den SRAM zu laden, gleich Flashzellen für die Konfiguration zu verwenden. Dies machte lange Zeit technologische Probleme, doch gerade Actel hat hier sein Alleinstellungsmerkmal gefunden. Einmal konfiguriert halten die FPGAs diese Konfiguration, auch wenn mal der Strom ausfällt, und sie haben die Konfiguration mit dem nächsten Einschalten noch immer im Speicher, sind also vom ersten Moment an einsatzbereit. Für kritische Anwendungen mit hoher Zuverlässigkeit geben die Datenblätter zumindest 500 mögliche Konfigurationszyklen für die Flashzellen an; im Regelfall reicht dies aus für gewöhnliche Produktlebenszyklen. Aus den Anfangszeiten von CPLDs und Mikrokontrollern hat sich auch das Prinzip erhalten, auf dem Silizium mit der Programmierspannung kleine Sicherungen aufzuschmelzen, den FPGAs also eine OTP-Eigenschaft zu geben. Bei dieser Konfiguration mit Antifuses ist eine Abänderung der Konfiguration also prinzipbedingt später nicht mehr möglich. Mit der moderneren Möglichkeit über Flashzellen zu konfigurieren teilen sich die Antifuse basierten FPGA die Eigenschaft sofort

		PLL	SerDes	CPU
Actel	IGLOO	1MHz .. 160MHz (250MHz)	-	RMv6-M enabled
	IGLOO nano	1MHz .. 160MHz (250MHz)	-	-
	IGLOO PLUS	1MHz .. 160MHz (250MHz)	-	-
	ProASIC3	1MHz .. 350MHz	-	ARMv6-M enabled
	ProASIC3 nano	1MHz .. 350MHz	-	-
	ProASIC3L	1MHz .. 250MHz (350MHz)	-	ARMv6-M enabled
	RT ProASIC3	1MHz .. 250MHz (350MHz)	-	-
	Axcelerator	1GHz+	-	-
	RTAX-S	-	-	-
	RTAX-DSP	-	-	-
	Fusion	40MHz .. 250Mhz	-	ARMv6-M enabled
	SX-A	-	-	-
	eX	-	-	-
	MX	-	-	-
	ProASIC+	24MHz .. 180MHz	-	-
Altera	Arria II GX	472.5MHz	155Mbps .. 3.75Gbps	-
	Arria GX	550MHz	600Mbps .. 3.175Gbps	-
	Stratix IV GT	717MHz (800Mhz)	2.488Gbps .. 11.3Gbps	-
	Stratix IV GX	717MHz (800Mhz)	600Mbps .. 8.5Gbps	-
	Stratix IV	717MHz (800Mhz)	-	-
	Stratix III	375Mhz .. 600Mhz	-	-
	Stratix II GX	550MHz	600Mbps .. 3.375Gbps	-
	Stratix II	550MHz	-	-
	Stratix GX	500MHz	500Mbps .. 3.1875Gpbs	-
	Stratix	500MHz	-	-
	Cyclone II	10MHz .. 400MHz (500Mhz)	-	-
	Cyclone	15MHz .. 275Mhz (320MHz)	-	-
Lattice	LatticeECP3	2MHz .. 420MHz	150Mbps .. 3.2Gbps	-
	LatticeECP2M/S	20MHz .. 420MHz	250Mbps .. 3.125Gbps	-
	LatticeECP2	20MHz .. 420MHz	-	-
	LatticeECP & EC	25MHz ..420MHz	-	-
	LatticeSC/M	1.5MHz .. 1GHz	600Mbps .. 3.8Gbps	bus interface
	LatticeXP2	10MHz .. 435MHz	-	-
	LatticeXP	25MHz .. 375MHz	-	-
Xilinx	Virtex 6	400MHz .. 1600MHz	150Mbps .. 6.5Gbps / 2.488Gbps .. 11Gbps	-
	Virtex 5	400MHz .. 1000MHz 1440MHz)	100Mbps .. 6.5Gbps	PPC440
	Virtex 4	350mHz	622Mbps .. 6.5Gbps	-
	Virtex II Pro	24MHz .. 360MHz (450MHz)	600Mbps .. 3.125Gbps (4.25Gbps)	PPC405
	Virtex II	24MHz .. 360MHz (450MHz)	-	-
	Virtex E	60MHz .. 275MHz (350MHz)	-	-
	Virtex	60MHz .. 180MHz (200MHz)	-	-
	Spartan 6	400MHz .. 1000MHz	622Mbps .. 3.125Gbps	-
	Spartan 3	5MHz .. 300MHz	-	-

mit dem Einschalten einsatzbereit zu sein. Darüber hinaus bieten sie mehr Sicherheit gegen kippende Bits unter dem Einfluss von harter Strahlung, ein unschätzbarer Vorteil beispielsweise in Satelliten.

Power Consumption

FPGAs sind sicherlich nicht immer die erste Wahl, wenn es zum Beispiel bei mobilen Endgeräten um die Energieeffizienz und lange Akkulaufzeiten geht. Gegenüber einem richtigen ASIC enthalten FPGAs in etwa 10 mal soviel Gatter bzw. Transistoren; entsprechend größer ist ihr Energiehunger. Ihr Energieverbrauch ist abhängig von der Taktgeschwindigkeit, je höher die Frequenz, umso häufiger müssen die Transistorstufen aus pMOS und nMOS Transistor umgeladen werden, der Togglerate aller Signale und eben von der Versorgungsspannung. Gerade die SRAM basierten FPGA haben aufgrund ihre Speicherzellen für die Konfiguration ein paar Gatter mehr, die gefüttert werden wollen. Entsprechend nehmen sie die Spitzenreiterposition im Energieverbrauch ein, Wattzahlen von 10 bis 15 Watt sind leider keine Seltenheit, gefolgt von den Flash basierten. Die Antifuse Familien kommen am sparsamsten daher. Fast jeder Hersteller bietet ein Tool zum Download an, mit welchem der mutmaßliche Energieverbrauch bereits im Vorfeld kalkuliert werden könnte. Derzeitig am sparsamsten sind die Bausteine aus Actels IGLOO Reihe, die hunrigstens sicherlich die Xilinx Virtex.

HDLs

Nun, wie kommt die gewünschte Funktion aufs Silizium? Sofern eine Bibliothek wirklich alle Komponenten enthält, die gebraucht werden, läßt sich da sicherlich etwas zusammen klicken. Im Regelfall aber wird aber vieles in einer so genannten Hardwarebeschreibungssprache (HDL) wie Verilog oder VHDL beschrieben, simuliert und synthetisiert. VHDL hat in Europa, speziell in Deutschland seine größte Verbreitung weil es hier gern aufgrund seines akademischen Potentials an der Universität gelehrt wird. Im Alltag zieht VHDL, bewusst vom amerikanischen DoD an ADA angelehnt, jedoch in Sachen Produktivität oftmals den Kürzen gegenüber Verilog, dessen Ursprung bei einem Toolhersteller für Chipdesign liegt. VHDL ist aufgrund seines zugrundeliegenden Simulationsmechanismus prinzipiell durchaus in der Lage, auch anderes wie zum Beispiel das Wetter zu simulieren, ein entsprechendes Modell vorausgesetzt. Verilog kommt einfacher daher, dort sind die Objekte wie Gatter, Transistoren und Drähte eben schon im Sprachumfang enthalten und kein eingebundenes Modell. Letztlich entscheiden Projektvorgaben oder persönliche Vorlieben über die Wahl der Sprache; Verilog oder VHDL. Ein Chipdesigner muss beide HDLs können und wird in seiner Arbeit stets mit beiden konfrontiert. Wer sich als Anfänger in die Materie einarbeiten möchte oder selbst das eine oder andere Design simulieren möchte, dem sei Verilog empfohlen; dafür existiert ein flotter Open Source Simulator mit dem Namen Icarus Verilog [6]. Die beiden Open Source Äquivalente für VHDL bleiben auf halbem Weg stecken. Offensichtlich bereitet die korrekte Implementierung des gesamten Sprachumfanges von VHDL Probleme.

		Windows	Linux	Solaris	
Actel	Libero IDE	Windows XP 32bit Vista Business	RHEL 4 WS 64bit RHEL 5 WS 64bit	-	kostenpflichtig
Altera	Quartus II Web Edition	Windows XP 32bit Windows Vista 32bit	RHEL 4 WS 32bit RHEL 5 WS 32bit CentOS 4 32bit CentOS 5 32bit SLE 9 WS 32bit SLE 10 WS 32bit	-	kostenfreier Download eingeschränkt
	Quartus II Subcription Edition	Windows XP 32bit Windows XP 64bit Windows Vista 32bit Windows Vista 64bit	RHEL 4 WS 32bit RHEL 4 WS 64bit RHEL 5 WS 32bit RHEL 5 WS 64bit CentOS 4 32bit CentOS 5 64bit SLE 10 WS 32bit SLE 10 WS 64bit	-	kostenpflichtig vollständig
	Nios II Embedded Design Suite	Windows XP 32bit Windows Vista 32bit	RHEL 4 WS 32bit RHEL 4 WS 64bit RHEL 5 WS 32bit RHEL 5 WS 64bit SLE 10 WS 32bit SLE 10 WS 64bit	-	kostenpflichtig unterstützt Programmentwicklung für Nios uC
Lattice	ispLEVER starter	Windows XP 32bit Windows Vista 32bit Windows 2000	-	-	kostenfreier Download eingeschränkt
	ispLEVER	Windows XP 32bit Windows Vista 32bit Windows 2000	RHEL 3 WS 32bit RHEL 3 WS 64bit RHEL 4 WS 32bit RHEL 4 WS 64bit RHEL 5 WS 32bit RHEL 5 WS 64bit	Solarix 2.8 , Solaris 2.10	kostenpflichtig
Xilinx	ISE Webpack	Windows XP 32bit Windows Vista 32bit	RHEL 4 WS 32bit RHEL 5 WS 32bit SLE 10 WS 32bit	-	kostenfreier Download eingeschränkt
	ISE Foundation	Windows XP 32bit Windows XP 64bit Windows Vista 32bit Windows Vista 64bit	RHEL 4 WS 32bit RHEL 4 WS 64bit RHEL 5 WS 32bit RHEL 5 WS 64bit SLE 10 WS 32bit SLE 10 WS 64bit	-	kostenpflichtig vollständig
	System Edition	Windows XP 32bit Windows Vista 32bit	RHEL 4 WS 32bit RHEL 4 WS 64bit RHEL 5 WS 32bit RHEL 5 WS 64bit SLE 10 WS 32bit SLE 10 WS 64bit	-	kostenpflichtig unterstützt Programmentwicklung für Embedded uC

IPCores

Das Schreiben von anwendungsspezifischen Modulen oder gar Microcontrolern in einer HDL ist eine Sysiphusaufgabe, erst recht weil erfahrungsgemäß der Aufwand mit Faktor 5 für die Beschreibung von Testcases und deren Verifizierung zu veranschlagen ist. Schön, wenn bei der Entwicklung bereits fertige und gut getestete Module zur Verfügung stehen. Zahlreiche Firmen leben davon, so genannte IP Cores bereit zu stellen. Wer nicht tief in die Tasche greifen möchte oder vermag, kann sich IP Cores in guter Open Source Manier auch im Netz mit Lizenzen ähnlich der GPL oder BSD herunter laden. Angeraten ist hier an erster Stelle aufgrund seiner Fülle die OpenCores Site anzusehen [7]. Leider wird dort gerade die Verifizierung nur klein geschrieben, die Cores lassen oft eine ordentliche Methodik vermissen. Dennoch, es gibt Controller in vielen Geschmacksrichtungen, Peripherie und nochmehr Vorhaben. Übrigens halten sich hier Verilog und VHDL Designs in etwa die Waage, gut durchmischt mit ein paar exotischen Sprachansätzen. Eine Erleichterung bei der Entwicklung von Anwendungen in FPGAs sind die IP Cores auf alle Fälle. Im Umfeld von OpenCores hat sich übrigens der Wishbone Interconnet Bus als Bussystem etabliert, patentfrei im Gegensatz zum kommerziellen Umfeld mit dem AMBA Bus.

Toolchains

Grundsätzlich gilt bei der Entwicklung von FPGAs, wie eigentlich bei allen leistungshungrigen Anwendungen: der Rechner kann nicht schnell genug sein, der Speicher ist immer zu wenig und das System sollte schlank und gut gewartet sein. Aufgrund der Adressierbarkeit von großen Speichermodulen, 32bit Systeme kennen hier eine hinderliche Grenze, sind 64bit angeraten und RAM im Gigabytes. Umfangreiche Simulationen von kompletten Chips mit Timing Annotation erreichen schnell Zeiträume von einigen Stunden bis einigen Tagen, umso ärgerlicher wäre es, wenn in dieser Zeit die Simulation aufgrund von Stabilitätsproblemen abbricht und wiederholt werden muss.

Alle vier Hersteller bieten auf ihren Webseiten Tools für ihre FPGAs zum (kostenfreien) Download an. Alle vier tun dies ausschließlich für die Windowsversion verbunden mit einer Registrierung. Wer Vorteile eines unixoiden Betriebssystems nutzen möchte, von den alten klassischen Workstationsystemen ist nurnoch Solaris übrig, beziehungsweise Linux, muss zahlen. Die Preise sind recht unterschiedlich und gehen je nach Lizenzmodell in die Tausender. Angeboten wird die Software in mehreren Ausbaustufen. Zum einen die kostenlosen Packs für Windows, dann für die Workstationsystem und letztens mit Unterstützung der Programmentwicklung für einen eingebetteten Microcontroler. Für Freunde eines Unix-artigen Betriebssystems ist für alle vier Hersteller RedHat Enterprise Linux 4 (RHEL 4) der kleinste gemeinsame Nenner. Das gibt es auch in der 64bit Version.

Links

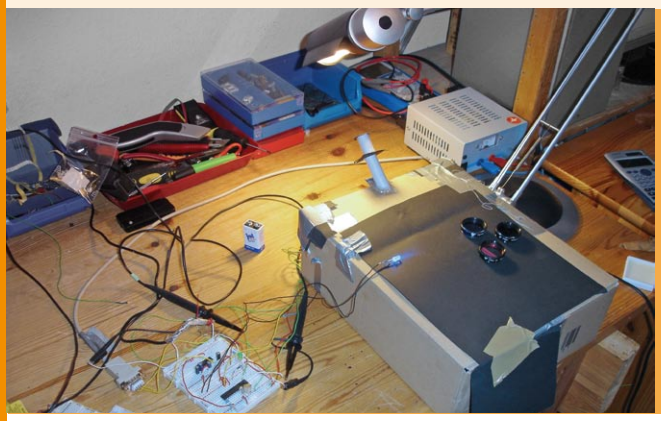
Jedem sei beim Einsatz von FPGAs das unvermeidliche Studium der Datenblätter angeraten.

[1] Wannemacher, Markus: Das FPGA Kochbuch. mitp 1998.
[2] www.xilinx.com
[3] www.altera.com
[4] www.latticesemi.com
[5] www.actel.com
[6] www.icarus.com/eda/verilog
[7] www.opencores.org

Digitales PC-Spektrometer

Ein Schukarton und ein Plastikrohr

Moritz Greif



Ein Schuhkarton, ein Plastikröhrchen aus dem Baumarkt, Pappe, Klebeband, eine Handvoll Kabel, und ein paar Elektronikbauteile, mehr braucht man nicht, um ein modernes Messgerät zu bauen, welches in fast allen naturwissenschaftlichen und technischen Forschungen Verwendung findet: ein digitales Spektrometer.

Hier soll kurz der Aufbau eines Spektrometers erklärt werden, das man sich, mit entsperechenden Elektronikkenntnissen, selbst bauen kann. Es hat natürlich gewisse Einschränkungen gegenüber professionellen Messgeräten.

Was ist ein Lichtspektrum?

Licht wird in der Physik u.a. durch ein Wellenmodell beschrieben. Ein Lichtstrahl stellt man sich als elektromagnetische Welle vor, die sich mit Lichtgeschwindigkeit ausbreitet. Dabei enspricht die Farbe des Lichts der Wellenlänge (Distanz Wellenberg-Wellenberg).
Dabei nutzen Wissenschaftler die einzelnen Intensitäten der Lichtfarben (Wellenlängen), die sie von Objekten erhalten, um viele wichtige Informationen zu gewinnen. Z.B. erfährt man über die Farben von Sternen sehr viel über deren Zusammensetzung. Ein Messgerät zur Gewinnung der Intensität einer Wellenlänge in einem Lichtstrahl, heisst Spektrometer.
Ein Diagramm, welches mehrere Wellenlängen und deren Intensität in Beziehung setzt, nennt man Lichtspektrum.

Das Schuhkarton-Spektrometer, welches ich hier vorstelle, kann z.B. benutzt werden, um die Spektren einzelner Glühbirnen/Energiesparlampen zu vergleichen, Aquariumlampen, Planzenlampen, Gewächshausbeleuchtungen u.ä. zu untersuchen, Messungen zum Sonnenspektrum anzustellen, spektroskopische Astronomie zu betreiben, etc. Man kann auch die Lichtabsorbierende Wirkung verschidener Materialien beobachten.
Ich hatte vor einiger Zeit die Gelegenheit ein kommerzielles PC-Spektrometer zu benutzen, um damit in einem Laser-Labor Laserstrahlen auf die Bandbreite hin zu untersuchen. Das (sehr teure Messgerät) koppelte Licht per Lichtleiter ein, und zeigte dann am PC-Monitor das Spektrum live an. Bei der Arbeit kam mir die Idee, so ein Live-PC-Spektrometer nachzubauen.

Abb. 1: Digitales PC-Spektrometer

Prinzip

Durch einen Spalt fällt das Licht durch ein Gitter, Prisma, oder wird von einem Gitterspiegel reflektiert. Das Licht wird dadurch in seine Spektralanteile (=Farben) aufgesplittet, und fällt auf einen Linien-CCD. Der CCD wird von einem μC ausgelesen, und schickt die Daten an den PC. Auf dem PC läuft eine selbstgeschriebene Software, die die Daten im Empfang nimmt, und in ein Diagramm zeichnet, das man dann speichern kann. Natürlich kann man mit den Daten beliebige Messungen machen, oder beliebig am PC weiterbearbeiten.

Aufbau

Um aus einer Lichtquelle einen vernünftigen Lichtstrahl zu erhalten, muss man das Licht erstmal entweder bündeln oder so „zurechtformen“, bis man einen annähernd parallelen, dünnen, exacten Lichstrahl bekommt. Ich habe viel mit Linsen experimentiert, da jedoch ein heimischer Basteltisch keine optische Werkbank ist, nur mit mäßigem Erfolg. Also hab ich einen anderen Weg gewählt. Ein Plasikröhrchen (1cm Durchmesser, 10cm lang) aus dem Baumarkt, wird zweimal durchgetrennt, und jeweils mit eine Scheibe aus schwarzem Tonpaier dazwischen, wieder zusammengeklebt. Das Tonpapier hat in der Mitte einen möglichst dünnen Schlitz. Man muss genau darauf achten, die Schlitze direkt untereinander anzuordnen. Nun kommt eine dritte Tonpapier-Scheibe auf das eine Ende des Plastikröhrchens, gewissermaßen als Deckel. So hat man drei Spalte hintereinander, das ist eine unglaublich gute Streulichtabsorbtion. Das Licht fällt dadurch annähernd parallel durch die drei Spalte, bzw. es kommt nur Licht in den Karton, welches genau parallel zu der Achse der drei Schlitze ist.

Abb. 2: Pappspalt-Lichteinleitung Röhrchen für Licht

Um das Spektrum zu erhalten, habe ich das Licht auf einen Gitterspiegel aus einem Monochromator geworfen (ebay). Der zurückreflektierte Lichtstrahl ist nun nicht mehr dünn, sondern aufgefächert in seine einzelnen Spektralanteile. Genausogut könnte man eine CD-Rückseite benutzen, ein Foliengitter, ein geeignetes Prisma, etc.

Nun muss man nichts anderes machen, als die einzelnen Farben auf ihre Intensität hin zu untersuchen, und zwar möglichst kontinuierlich. Was eignet sich dazu besser als ein Digitalkamerachip? Allerdings interessiert hier nur die Intensität einer beleuchteten Fläche in einer Richtung, deswegen bediente ich mich eines Linien-CCD-Chips. Der Sony ILX554B (Firma Framos) eignet sich dazu gut, es gibt aber auch eine Handvoll anderer. Der Chip ist ein paar Centimeter lang und wird einfach in das projezierte Spektrum gebaut.

Alle diese Komponenten müssen nun geschickt in eine Kiste, oder den besagten Schuhkarton gebaut werden, dabei sollte das Gerät ABSOLUT lichtdicht sein! Aber natürlich, bevor man das tut, sollte die Elektronik zumindest ansatzweise fertigprogrammiert sein, der CCD-Chip schon mal im dunklen Zimmer getestet worden sein, das Spektrum mit dem Oszilloskop vielleicht schon einmal betrachtet, usw.

Die Bauteile

Die Bauteile, die ich verwendet habe, sind folgende. Der Code und das PC-Programm sind darauf ausgelegt. Sicher kann man das Gerät auch mit anderen CCD's, Controllern, Optiken, etc. realisieren.

Desweiteren ist eine Steckplatine ganz nützlich, dann braucht man noch die Bauteile zur Beschaltung des Mikrocontrollers und des MAX232's [http://www.mikrocontroller.net].

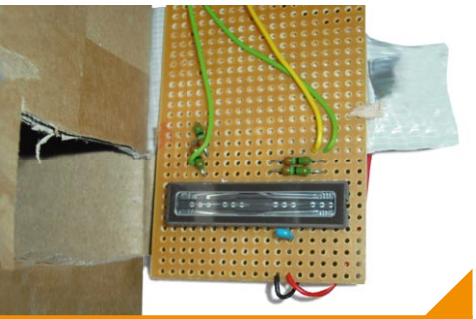


Abb. 3: CCD-Chip von Sony-Kamera

...braucht man:	..habe ich verwendet:	..bekommt man bei:
Controller	ATmega8	z.B. Reichelt [http://www.reichelt.de], oder andere Elektronikfirma
Beugungs/ Brechungsoptik	Gitterspiegel (aus Monochromator)	ebay, oder Optikfirma, wie z.B. Edmund optics [http://www.edmudoptics.de]
Gehäuse	Karton, Lichtdicht verklebt, oder Holzkistchen	hat man bestimmt irgendwo rumfliegen, selber bauen
PC-Connection	seriell, per MAX232-Converter (oder FDTI-USB-UART-Chip)	auch Reichelt (Max232) [http://www.reichelt.de], oder USB-UART-Platinchen von FTDI [http://www.ftdichips.com] (recht billig)
CCD	Sony ILX554B	Framos [http://www.framos.eu/sensors_line.html]
PC	Windows-PC mit serieller (oder USB) Schnittstelle	...hat jeder

Die Schaltung

Die Schaltung ist so einfach wie möglich gehalten. Ein ATmega8-Mikroconroller ist das Herzstück, er ist in der Standardbeschaltung mit einem 16Mhz Quarz betrieben. Es würde sich allerdings lohnen, gleich mit einem ATmega644 (4096Byte SRAM, max. 20MHZ) einzusteigen, ich kam mit dem ATmega8 (1024Byte SRAM, max 16Mhz) schnell an die SRAM-Grenzen.

Der Mikrocontroller wird folgendes leisten:

- Initalisieren
- CCD abfragen, dabei schnell die analogen Spannungswerte der einzelnen Pixel in digitale Werte umwandeln und in den eigenen Speicher (SRAM) zwischenspeichern
- Ist das ganze Spektrum erfasst, wird es Pixel für Pixel per serieller Schnittstelle an der PC übertragen, wo es angezeigt wird
- CCD wird von neuem Abgefragt

Die Elektronik ist im Wesentlichen die Grundschaltung zum Betrieb eines ATmega-Mikrocontrollers von Atmel, dazu noch die Anbindung an die Serielle Schnittstelle mit Hilfe des spannungskonverter-IC's MAX232. Diese Schaltungen sind z.B. auf mikrocontroller.net ausführlich erklärt. Der Vout-Ausgang des CCD's geht in den ADC0 (PortC 0). Der ROG-Pin des CCD's geht an PORTB 0 und der CLOCK-PIN des CCD's an PORTB 1 des Mikrocontrollers. Das war's!

Die Programmierung des µC's und der PC-Software

Ich programmiere AVR's in Assembler. Die PC-Software zum Empfangen der seriellen Daten erstellte ich mit Delphi. Ich benutzte die Version Delphi 2005. Diese war kostenfrei im Internet erhältlich. Inzwischen muss man wahrscheinlich auf alte CT'magazin-Hefte mit CD-Beilage zurückgreifen, oder eine richtige Version kaufen.

Um den COM-port abzufragen benötigt man die comport library 3.1.

Nützliche Links dazu:
Forumsbeitrag: Wie steuer ich den RS232-port mit Delphi an?
http://www.delphipraxis.net/topic102188_uart+terminalprogramm+zur+kommunikation+mit+atmega8+c.html&highlight=

Hier kann man die Komponente runterladen
http://sourceforge.net/projects/comport/

Assembler

Der Assemblercode hat verschiedene Aufgaben: - Initalisiere den ADC, event. LCD, und das UART - Setze die Steuerbits des CCD's - Wandle die einzelnen analogen Pixelwerte des CCD's in digitale um - speicher sie im SRAM zwischen - wenn der CCD ausgelesen ist, sende sie an den PC - fange von vorne an mit einem Auslesevorgang.

Die Programmierung der Ausleseroutine ist nicht ganz einfach. Man muss sich genau das Datenblatt anschauen, und die Zeiten genau einhalten. Deshalb ist es wahrscheinlich auch sinnvoll, hier mit der Programmiersprache Assembler zu arbeiten, da man dort jegliche Taktzeiten für jeden Befehl kennt. Der CCD besitzt drei Pins, einen Steuerpin (ROG), einen CLOCK-Pin für den Takt, und einen V-out für die Spannungswerte der einzelnen Pixel. Man schaltet den ROG-Pin auf LOW, gibt dann ein Taktsignal an den Takt-Pin, der bewirkt, dass pro Takt ein Pixelwert weitergeshiftet wird, und am Ende immer ein neuer Pixelwert an dem V-Out-Pin anliegt, den man dann ausliest. Das muss man sich aber noch mal genau nach dem Datenblatt klar machen.

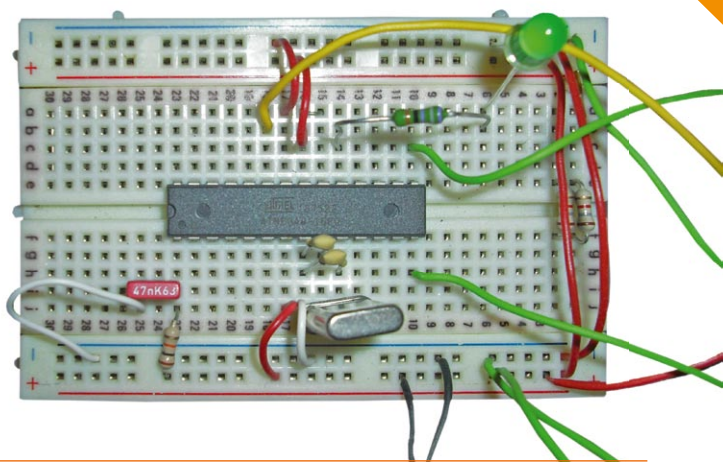
Das Problem beim Auslesen des CCD's bestand hauptsächlich darin, die Auslesezeit genau einzustellen. Macht man sie zu kurz, wird der CCD zu wenig belichtet (kommt nicht oft vor) macht man sie zu lang, belichtet der CCD über.
Dabei muss man unterscheiden zwischen der Auslesezeit, und der Belichtungszeit. BELICHTET wird der CCD grundsätzlich immer, die Ladung der belichteten Pixel wird bei dem NEGATIVEN ROG-Impuls in das Ausleseregister geladen. Dort ist die Ladung erstmal sicher, und kann theoretisch beliebig langsam ausgelesen werden. Aber nur theoretisch. Das Ausleseregister entlädt sich auch nach einer gewissen Zeit. Mann muss deshalb nach dem neg. Rog-Impuls die Dummy-Pixel so schnell als möglich durchrattern. Bei mir hat sich erwiesen, das ein kompletter Auslesevorgang 110ms dauern darf. Die Dummy-Pixel-Clock-Perioden dauern deshlab 2μs (32zyclen bei 16Mhz). Ein auszulesendes-Pixel-clock-Signal braucht für die High (und für die Low-Zeit) 110μS. Damit benötigt die komplette Ausleseroutine wie in der Datenblattskizze oben gezeigt, 110ms.
In dieser Zeit ist der CCD natürlich wieder dolle belichtet worden. deshalb wird vor der nächsten Ausleseroutine eine „Dummy-Ausleseroutine" gestartet: Diese macht nichts anderes als 2100 sehr kurze Rechtecksignale in die Clock-Leitung zu schicken, und danach den Rog-Pin wieder Low zu machen - damit wurden die Pixel neu belichtet, und zwar mit der perfekten Belichtungszeit! Man müsste diese Rechtecksignal-Frequenz jetzt praktisch verändern, um die Belichtungszeit anzupassen. Bei mir hat sich erwiesen, dass eine Gesamt-Belichtungszeit von 4,2ms gute Ergebnisse bringt.
Zum eigentlichen Auslesen der 2048 Pixel: Es werden immer 3 Pulse schnell als „dummy" rausgeschickt, dann eine High-Low-Peridode mit wirklicher Messung. Sowit lese ich jeden 4.Pixel aus (ATmega8 hat zu wenig SRAM, langsamen ADC). Bei dem 4. Rechtecksignal wird nur in der High-Phase der gemessene Wert tatsächlich gespeichert. Da die High-Zeit aber genau gleich der Low-Zeit sein muss, messe ich einfach zweimal per ADC den Vout-Ausgang des CCD's. Somit sind die gleichen Zeiten gewährleistet. Gespeichert wird der Wert dann nur in der High-Zeit. (Der Befehl **`ST X+, adcwert`** benötigt vernachlässigbar wenig Zeit). Hier das Stückchen Code, das 512 mal wiederholt wird:

```
cbi CCD_PORT, clk
rcall kurz  ; braucht 16 Zyclen = 2µs
sbi CCD_PORT, clk
rcall kurz

cbi CCD_PORT, clk
rcall kurz
sbi CCD_PORT, clk
rcall kurz

cbi CCD_PORT, clk
rcall kurz
sbi CCD_PORT, clk
rcall kurz

cbi CCD_PORT, clk
rcall messe  ; braucht gemessene 110 µs
sbi CCD_PORT, clk
rcall messe_und_save ;braucht auch gemessene 110 µs
```

Den weiteren Code erläutere ich hier nicht in jeder Einzelheit. Dazu gibts die Code-Kommentare, ausserdem ist er recht simpel. Viel kann man direkt aus dem AVR-Tutorial übernehmen (Initalisierung der ADC's, UARTS).

Natürlich habe ich nicht jedes SRAM-Init-Stückchen selbst eingetippt, deswegen sind Codeteile hier aus dem Forum kopiert und werden wiederverwendet.

Abb. 4: ATMega8 auf Steckbrett



Fazit

Das Ergebnis ist schon eindrucksvoll. Man hat eine sekündlich Aktualisierung des Spektrums, und kann losexperimentieren! Man kann verschiedene LED's vor den Eingangsspalt halten, mit einem Graufilter war auch ein Sonnenspektrum möglich. Man sieht physikalische Prinzipien, wie Linienabsorbtion, Durchlassbreiten bei verschiedenen Filtern, und das Sonnenspektrum errinnert entfernt an das eines „schwarzen Körpers"- Das temeraturabhängige, charakteistische Spektrum das ein Körper emmitieren würde, der jegliche elektromagnetische Welle vollständig absorbiert. [http://de.wikipedia.org/w/index.php?title=Datei:BlackbodySpectrum_lin_150dpi_de.png&filetimestamp=20060609125107]

Auch das Sonnenspektrum ist annähernd solch ein schwarzer Körper. Viel Spass haben auch Experimente mit verschiedenen lichtdurchlässigen Medien gemacht, wie .z.B. Apfelsaft, Rotfilter aus dem Teleskop-Okularkoffer, Bastelfolien, Glas, etc.

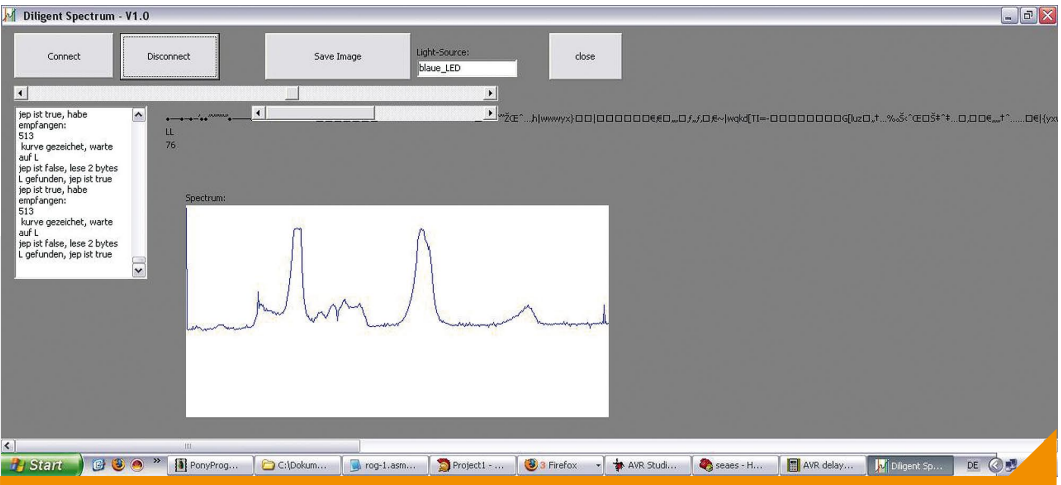
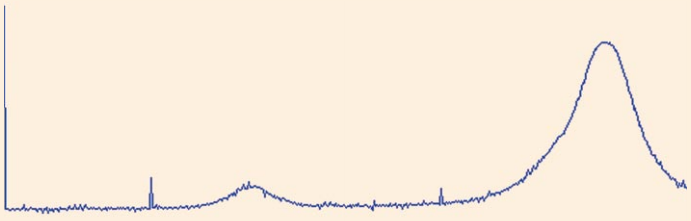


Abb. 5: Screenshot Diligent Spectrum

Natürlich ist mein Spektrometer nicht kalibriert, und so sind die Aussagen nur qualitativ. Eine Kalibirierung ist tatsächlich kniffelig, da:

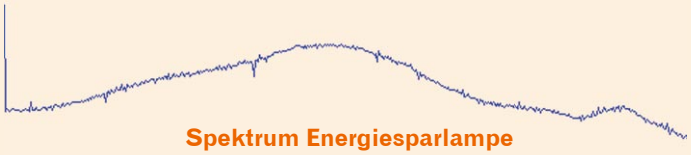
1. Das Prisma/Der Gitterspiegel ein möglichst ebenes Bild liefern muss
2. Der Linien-CCD eine charakteristische Empfindlichkeit bei verschiedenen Wellenlängen hat
3. Man eine Referenzquelle braucht

Dennoch denke ich, ist es ein interessantes Projekt, was einem auch mal ein bisschen Wissenschaftsluft im eigenen Wohnzimmer schnuppern lässt. Das Projekt hat neben den spannenden K onstruktionsherausforderungen auch noch ein enormes Potential- man kann es mit genügend Zeit und Engagement sicher noch viel weiter verbessern!

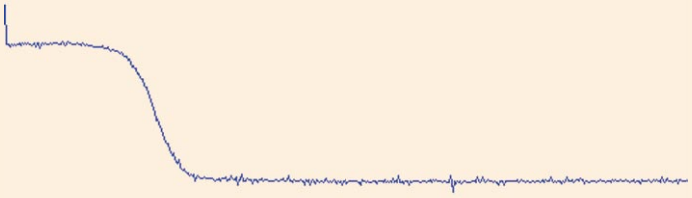


Spektrum LED Blau

Spektrum Energiesparlampe



Spektrum Energiesparlampe



Spektrum Rotfilter

Konstruktionsmöglichkeiten für die Zukunft

Ich plane weitere Verbesserungen, um das Spektrometer noch genauer und schneller zu machen:

- **ATmega644**
  mit 4096 Bytes SRAM erlaubt alle 2048 Pixel zwischenzuspeichern, und erhöht somit die Auflösung (mit dem ATmega8 kann ich maximal 512 Pixel auslesen). Ausserdem läuft der 644 mit 20Mhz-Quarzen, bedeutet die Aktualisierung kann noch schneller werden, ebenso die Baud-Rate.
- **externer ADC, z.B. ADC0803 oder ADC0820**
  Damit wird die „Conversion“ viel schneller. Der AVR-ADC braucht bei 16Mhz ca. 110µs für eine Komplett-Messung, der ADC0803 gemesssene 20µs.
- **Linsen zur Strahlaufweitung/Abbildung des Spalts**
  Damit kann man die komplette Gitterfläche nutzen, um die optische Auflösung zu erhöhen. Das geht natürlich mit einem besseren Gehäuse einher. Dabei will ich auch den Fokus perfekt einstellen.
- **Eichung des Spektrometers anhand bekannter Spektrallinien**
  Ich werde einige Messungen mit Natriumdampflampen oder Laserpointern durchführen, um das Spektrometer zu eichen/kalibrieren. Idealerweise wird dann jedem Pixel eine ganz bestimmte Wellenlänge zugewiesen.
- **Belichtungszeit-Steuerung vom PC aus**
  Damit kann man exakter messen, und hat es leicheter, verschieden helle Lichtquellen zu untersuchen. Ist nur eine kleine Code-Veränderung am PC-Programm, ebenso im Assembler. Man muss eben nur die Clock Frequency und einige ähnliche Werte parametrisieren.

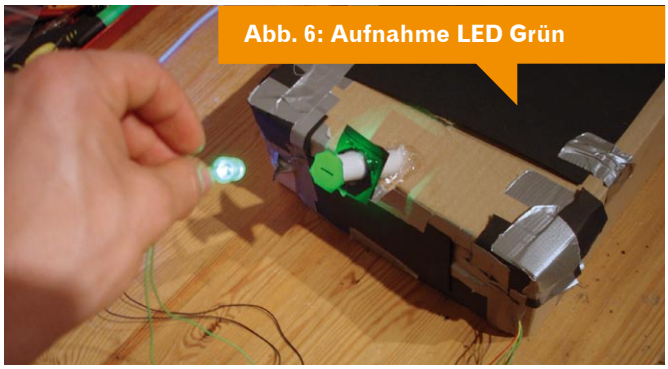


Abb. 6: Aufnahme LED Grün

OpenHR20

Elektronisches Heizkörperthermostat ansteuern

Dario Carluccio

Ein handelsüblicher mechanischer Thermostat, besteht aus einem mit Gas gefüllten Metallbalg, welches sich bei Wärme ausdehnt und über einen Übertragungsstift ein Ventil und damit den Zufluss von Heißwasser des Heizkörpers steuert. Dieses Regelverfahren ist nicht sehr genau, aus diesem Grund zeigt die Skala solcher Thermostaten selten Temperaturen, sondern in der Regel Ziffern zwischen 1 und 7.
In der beiliegenden Bedienungsanleitung wird dann angegeben, dass die Einstellung „3“ etwa 20°C entspricht und jede Veränderung der Einstellung um eine Zahl nach oben oder unten eine Temperaturänderung von etwa 4 °C bewirkt. Genau ist das nicht.

Was der Markt so bietet

Umso erfreulicher ist es, dass seit ein paar Jahren batteriebetriebene elektronische Heizkörperthermostate zum Nachrüsten für die eigenen (oder gemieteten) vier Wände angeboten werden. Diese kleinen Geräte werden anstelle der konventionellen Heizkörperventile angebracht und fortan regeln sie die Raumtemperatur der jeweiligen Zimmer. Dise elektronische Thermostate regeln die Temperatur weitaus genauer als die mechanischen. Neben der höheren Genauigkeit der elektronischen Thermostate ist der besondere Vorteil, dass die Solltemperatur programmierbar ist und so automatisch zu vorgegeben Zeiten geändert wird. So ist es möglich die Temperatur an Arbeitstagen von 7:00 bis 9:00 und von 17:00 bis 23:00, sowie am Wochenende von 10:00 bis 23:00 auf 22°C und in der verbleibenden Zeit auf 18°C zu stellen um so Energie zu sparen.

Wer nicht so diszipliniert ist und alle Thermostate runterstellt, wenn er die Zimmer nicht nutzt kann so leicht über 300Euro pro Jahr an Heizkosten sparen. So rechnet sich die Investition von ca. 30 Euro pro Thermostat schon im ersten Jahr.

Der Regler der Wahl

Ein solches elektronisches Heizkörperthermostat ist das Modell HR20 von Honeywell. Der Nachteil an diesem und fast allen anderen Thermostaten dieser Preisklasse ist der eingeschränkte Funktionsumfang. So ist die Einstellung der Schaltzeiten auf zwei Intervalle pro Wochentag beschränkt, ferner erlaubt es nur zwischen zwei unterschiedlichen Temperaturen (Komforttemperatur und Spartemperatur) umzuschalten. Auch ist das Eingeben der Intervalle über die Bedienelemente des Thermostaten sehr aufwändig, was besonders nervenaufreibend ist, wenn die eigene Wohnung mit mehreren Heizkörpern ausgestattet ist. Wenn zum Beispiel am folgenden Tag die betriebliche Weihnachtsfeier ist und der Bewohner kommt erst gegen 22:00 nach Hause kommt, oder an einem Donnerstag ist ein Feiertag und die Wohnung soll auch tagsüber geheizt werden, ist es erforderlich jeden einzelnen Thermostaten dementsprechend umzuprogrammieren, was leicht 20 Minuten dauern kann.

Wünschenswert wäre es daher den Funktionsumfang zu erweitern, um so zusätzliche Einstellungen zu ermöglichen und die Thermostate über eine zentrale Steuerung zu programmieren. Zusätzlich könnte die Ist-Temperatur an einer anderen Stelle im Raum gemessen um so den Messfehler in der Nähe der Heizung auszugleichen.

Das OpenHR20 Projekt

Ziel des OpenHR20 Projektes ist es, das Thermostat um eine Kommunikationsschnittstelle zu erweitern, sodass eine zentrale Steuerung angeschlossen werden kann. Die Kommunikation sollte hierbei frei wählbar sein, denkbar wären kabelgebundene Lösungen wie RS232, I²C oder CAN aber auch eine Funkschnittstelle, wie zum Beispiel 433 oder 866 MHz.

Hardware

Ein wichtiger Grund, dass es dieses Projekt überhaupt gibt ist die im HR20 eingesetzte CPU, dort verrichtet nämlich ein ATmega169 Mikrocontroller von ATMEL [AT169] seinen Dienst. Zu diesem Prozessor gibt es nicht nur sehr viele Informationen in Form von Datenblättern direkt vom Hersteller, auch die komplette Entwicklungsumgebung zum Programmieren eigener Software ist kostenlos verfügbar.

Der erste Schritt in diesem Projekt war es die Hardware, also den Schaltplan und insbesondere die Beschaltung der CPU zu ermitteln und zu dokumentieren [HW]. Dazu wurde das Thermostat zerlegt und Photos von Ober- und Unterseite der Platine im inneren geschossen.
Am Ende der mehrtägigen Fleißarbeit stand dann der fast vollständige Schaltplan des Thermostaten. So war bekannt, dass die ISP Programmierschnittstelle nicht ohne weiteres zugänglich ist, jedoch der JTAG Port so ausgeführt ist, dass man ihn kontaktieren kann ohne an der Schaltung im Thermostaten etwas ändern zu müssen. Als JTAG eignet sich jedes AVR kompatibles JTAG Interface, wie zum Beispiel der JTAG ICE Nachbau von Olimex [JTAG]

Im Prinzip ist es möglich über JTAG auch den aktuellen Inhalt des Flash Speichers, also des Programms auszulesen, wenn die entsprechende Sicherung, welche das verhindern soll nicht aktiviert wurde. Der Hersteller hat, wie nicht anders zu erwarten war diese Sicherung jedoch aktiviert. Es war jedoch sowieso nie das Ziel den

Code auszulesen, da man dadurch ja nur den kompilierten Code bekommt, das ist eigentlich nur interessant, wenn man kleine Änderung vornehmen will. Schön wäre der Code gewesen, weil man dann die Thermostate, nachdem man sie selber programmiert hat, wieder in den Auslieferungszustand versetzen könnte, das geht nun leider nicht mehr.
Per JTAG muss man zuerst den kompletten Inhalt des Controllers löschen, dadurch werden auch alle Sicherungen in den Ausgangszustand versetzte und der Chip ist wieder programmierbar.

Schaltplan

Zurück zum Schaltplan.
Über diesen ist zu erkennen, wie die Tasten an die CPU angeschlossen sind und wie der Motor angesteuert werden muss, allerdings fehlten noch einige Kleinigkeiten, welche nicht im Schaltplan abzulesen sind.

Das LCD Display ist ein speziell für dieses gerät hergestelltes Display, ein Datenblatt ist nicht öffentlich verfügbar. Es besteht aus 61 Segmenten, die über die prozessorinternen Register angesteuert werden. Die Zuordnung der Segnete des LCD zu den CPU internen Registern ist konnte nur ermittelt werden, indem die CPU mit einem entsprechenden Testprogramm programmiert wurde.

Auch die Temperaturmessung, welche über einen temperaturempfindlichen Widerstand und den CPU internen A/D Wandler erfolgt musste kalibriert werden. Hier wurde die Tatsache genutzt, dass es in der Originalsoftware eine Funktion gab um die Ist-Temperatur anzuzeigen. So wurde ein Widerstand per Umschalter an zwei Thermostaten betrieben, einem mit original Software und einen mit einem Testprogramm, welches die Werde des A/D-Wandlers anzeigte. So konnte die Temperaturkennlinie gemessen werden.

Treiber

Nachdem alle Einzelheiten bekannt waren, konnten die ersten Treiber programmiert werden. Das sind keine Programme, welche die Aufgabe haben die einzelnen Hardware-Komponenten des Thermostaten zu steuern. So wurden die Treiber programmiert:

- Displayansteuerung
- interne Uhr mit Datum und Uhrzeit, automatischer Sommer-/ Winterzeitumstellung
- Abfrage der Eingabetasten
- Messung der Temperatur- sowie der Batteriespannung.
- Motoransteuerung
- Abfrage der Sensoren (Thermostat montiert, Bewegung des Motors)
- Ansteuerung des EEPROM Speichers

Alles in allem waren nun alle Hürden genommen um die Thermostat Anwendung neu zu programmieren und dabei direkt die ersten Verbesserungen in die Software mit einfließen zu lassen.

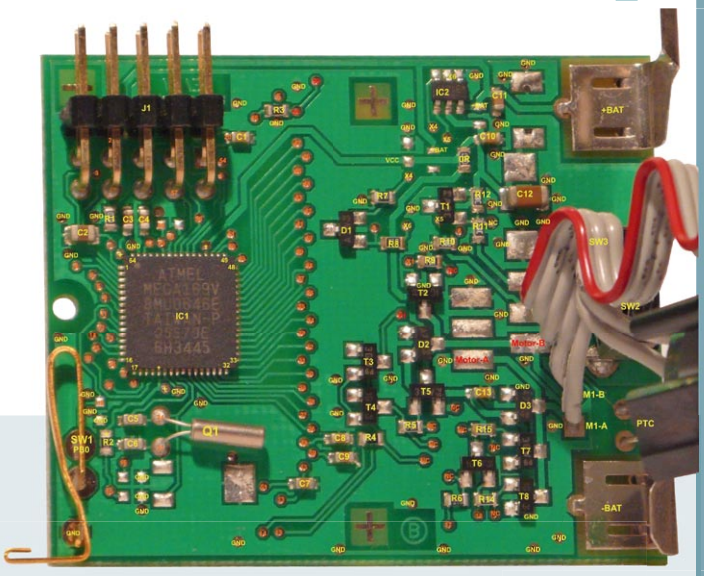
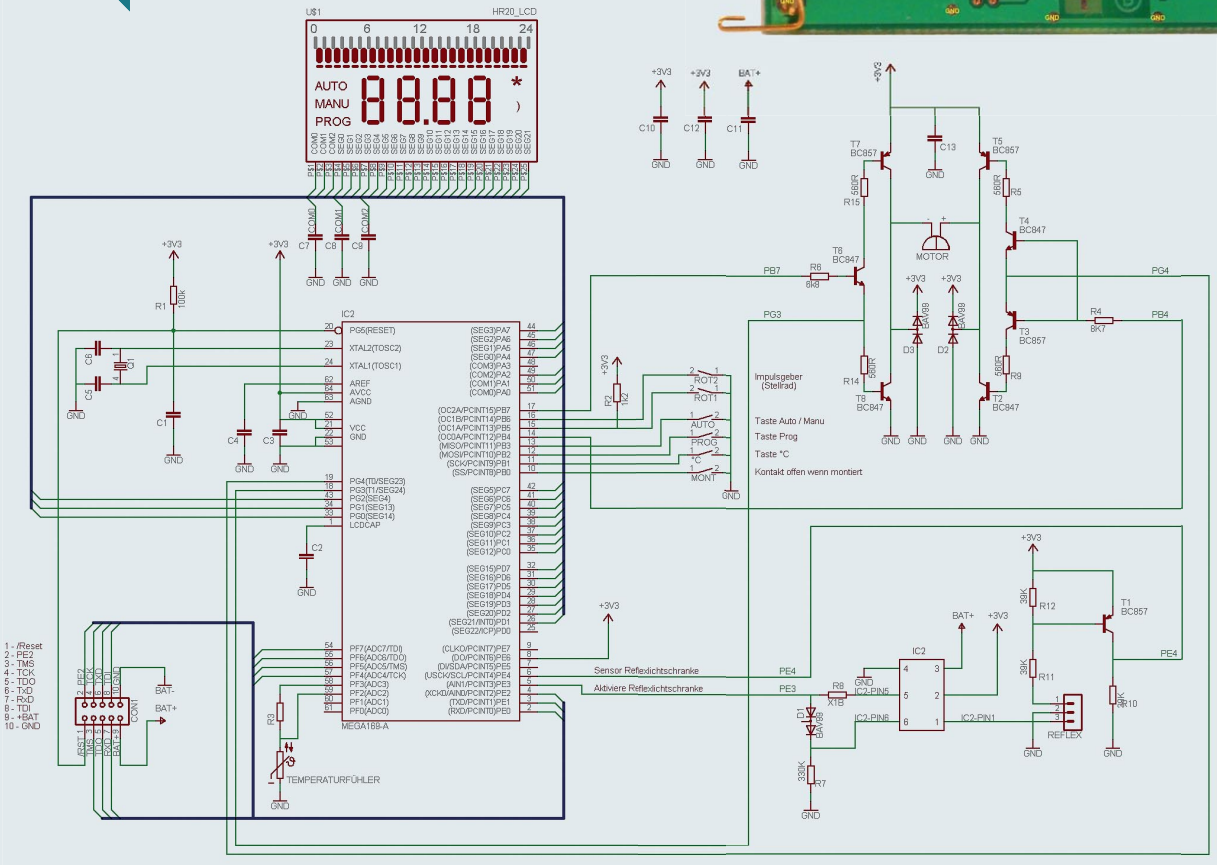


Abb. 2: Die Schaltung



Hilfe der Community

Genau an hier hat der Initiator des Projektes keine Energie mehr gehabt die einzelnen Codeschnipsel zu einem lauffähigen Programm zu verbinden. Auch fehlten ihm das Wissen über effektive und batterieschonende Programmierung, wie auch die Zeit sich weiter darum zu kümmern.
Also wurde für den Quellcode ein Repository bei sourceforge.net [SF] beantragt und das Projekt auf mikrocontroller.net [MCN] veröffentlicht. In dem Mikrocontroller Forum fand eine rege Diskussion statt.
Es fand sich schnell ein sehr kompetenter und engagierter Entwickler, der die einzelnen Funktionen zu einer lauffähigen Anwendung verbunden hat. So war kaum ein Jahr nachdem das Projekt gestartet ist die erste lauffähige Version verfügbar. Diese war zwar nicht komplett fehlerfrei, aber sie bot doch deutlich mehr Funktionen als die originale Software. So können zum Beispiel pro Tag vier anstelle von zwei Intervallen programmiert werden und auch vier anstelle von zwei Temperaturen sind nun wählbar.
Leider hat der oben genannte Entwickler auch nur wenig Zeit und daher ist das Projekt nun etwas verwaist, da sich bisher noch kein Maintainer für den Code gefunden hat.

Kommunikation

Erfreulich hingegen ist aber, dass eben dieser Entwickler das Projekt nicht komplett verlassen hat, vielmehr arbeitet er daran die Thermostate über Funk von einem zentralen Server zu steuern. Dazu wird ein RFM12 Funkmodul mit dem Thermostaten verbunden. Das geht entweder direktes Einlöten des Moduls in den Thermostaten, oder durch aufstecken einer kleinen Zusatzplatine auf den externen Stecker. Bei der letztere Variante verliert man die Möglichkeit das Thermostat per JTAG zu debuggen.

Da Batteriesparen an oberster Stelle steht wurde ein Protokoll entworfen, welches den Strombedarf der einzelnen Thermostate nach Möglichkeit nur minimal mehr Belasten soll, sodass die Batterien gleich lange halten und wie bisher maximal einmal pro Jahr gewechselt werden müssen.

Sicherheit

Da über Funk jeder die gesendeten Daten mithören kann und sogar auch selber senden kann soll verhindert werden, dass:

- aus dem Funkverkehr ein möglicher Einbrecher feststellen kann, dass niemand zu Hause ist.
- der Nachbar über der Wohnung die Temperatur hochstellt und sich so eine günstige Fußbodenheizung bastelt.

Daher wurde das Funkprotokoll um die entsprechenden Sicherheitsaspekte erweitert. Genutzt wird als Verschlüsselungsalgorithmus XTEA, da der Codespeicher schon sehr stark ausgelastet ist und die XTEA Implementierung sehr klein ist. Auch werden zum Verschlüsseln und Entschlüsseln, sowie zum Erzeugen und Prüfen der MAC Integritätscodes (kryptographische Prüfsummen) derartige Operationsmodi verwendet, dass nur eine XTEA Verschlüsselung nötig ist und auf die Implementierung der XTEA Entschlüsselung verzichtet werden kann.

Mit diesen Mitteln wird basierend auf an den Thermostaten eingegebenen geheimen Schlüsseln erreicht, dass die komplette Kommunikation für Außenstehende nicht entschlüsselt werden kann und die Thermostate nur Daten akzeptieren, welche von dem Absender kommen, der den selben geheimen Schlüssel kennt.

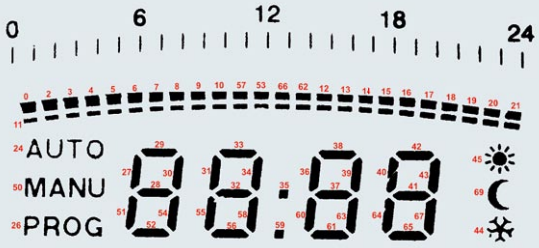
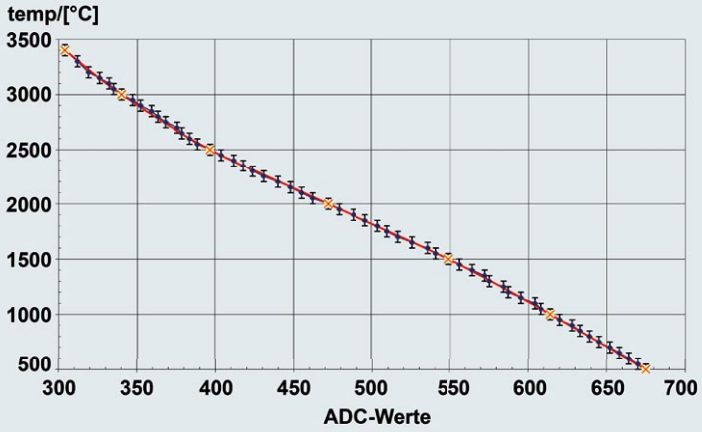


Abb. 3: Segmente des LCDs

Abb. 4: Temperaturkennlinie HR20



Eine kleine Platine mit einem ATMega 32 und einem RFM12 Funkmodul dient dabei als Kommunikationsmaster. Der ATMega 32 ist verantwortlich für das komplette Protokoll, sowie für die Verschlüsselung. Betrieben wird er an einem Linux Server (oder Linux Router), der die Daten zyklich abfragt, speichert und über ein Web-Frontent grafisch aufbereitet.

Unterstützung gesucht

Auch wenn schon sehr viel Arbeit gemacht wurde werden dringend Helfer gesucht. Zum einen wird ein Programmierer gesucht, der sich um die Hauptlinie des Sourcecodes kümmert, der Verbesserungen und Fehlerbehebungen prüft und die Änderungen in das Repository einpflegt.
Auch an der Dokumentation des Quellcodes (über Doxygen) ist noch viel zu pflegen. Auch fehlt noch eine Bedienungsanleitung für die neue Software. Nicht zuletzt wäre es schon, wenn es eine Web-Präsenz für das Projekt geben würde.
Also wer sich angesprochen fühlt und sich gerne mit einbringen will kann sich gerne beim Autor melden.

Links

[AT169] Datasheet ATmega169, Version 2514P–AVR–07/06
www.atmel.com/dyn/resources/prod_documents/doc2514.pdf
[HW] http://carluccio.de/images/e/e1/Hr20-analyse.pdf
[JTAG] http://olimex.com/dev/avr-jtag.html
http://olimex.com/dev/avr-usb-jtag.html
[MCN] http://www.mikrocontroller.net/topic/17603
http://embdev.net/topic/118781
http://www.mikrocontroller.net/articles/Heizungssteuerung_mit_Honeywell_HR20
[SF] http://sourceforge.net/projects/openhr20

*** HABEN SIE EINE BESSERE ANZEIGE?**
Open-Source lebt von Engagement!

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

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
Wir werden dann gemeinsam sehen, was wir daraus machen können.

Regelmäßig lesen!

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 (Anmeldeformular gesponserte Abos)
http://www.embedded-projects.net/journal
2. Online-Shop für Journal (Preis 1 EUR + Versand)
http://www.eproo.de/journal

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: 0821/27 95 990 oder sendet eine E-Mail an die oben genannte Adresse.

Bitte frei machen

Embedded Projects
Holzbachstraße 4
D-86152 Augsburg

Bitte in Druckbuchstaben gut lesbar ausfüllen, damit wir Ihre Bestellung bearbeiten können.

Name / Firma

Straße / Hausnummer

PLZ / Ort

E-Mail / Telefon / Fax

☐ **Bestellung des Embedded Projects Journal:**
Ich möchte jede zukünftige Ausgabe erhalten

☐ **Bestellung des Embedded Projects Journal für Hochschulen/ Ausbildungsbetriebe**
Wir möchten jede zukünftige Ausgabe zu Ausbildungszwecken bestellen
Anzahl der Hefte pro Ausgabe [5] [10]
(zutreffendes bitte ankreuzen)

☐ **Anforderung Infomaterial zur Anzeigenschaltung:**
Wir sind eine Firma und sind an Werbe- und Stellenanzeigen interessiert.
Bitte schicken Sie uns kostenlos und unverbindlich Infomaterial, Preisliste, ect. zur Anzeigenschaltung zu.

[IMPRESSUM]

embedded projects GmbH
Holzbachstrasse 4
D-86152 Augsburg
Telefon: +49 (0) 821 / 27 95 99 - 0
Telefax: +49 (0) 821 / 27 95 99 - 20
Mail: journal@embedded-projects.net

Anzeigemöglichkeiten und Preisliste auf Anfrage via Mail.

Herausgeber: Benedikt Sauter
Gestaltung/Satz: Medienkollektiv.de

Veröffentlichung: 4x / Jahr
Ausgabeformate: PDF / Print
Auflage Print: 2500 Stk.
Einzelverkaufspreis: 1,00 EUR

Dies ist ein Open-Source Projekt.
Informationen zum ABO - www.embedded-projects.net/journal

Titelfoto: Projekt OPEN HR20
Fotocomposing: Medienkollektiv.de

GESUCHT

Sponsor für Open-Source Projekt Zeitschrift