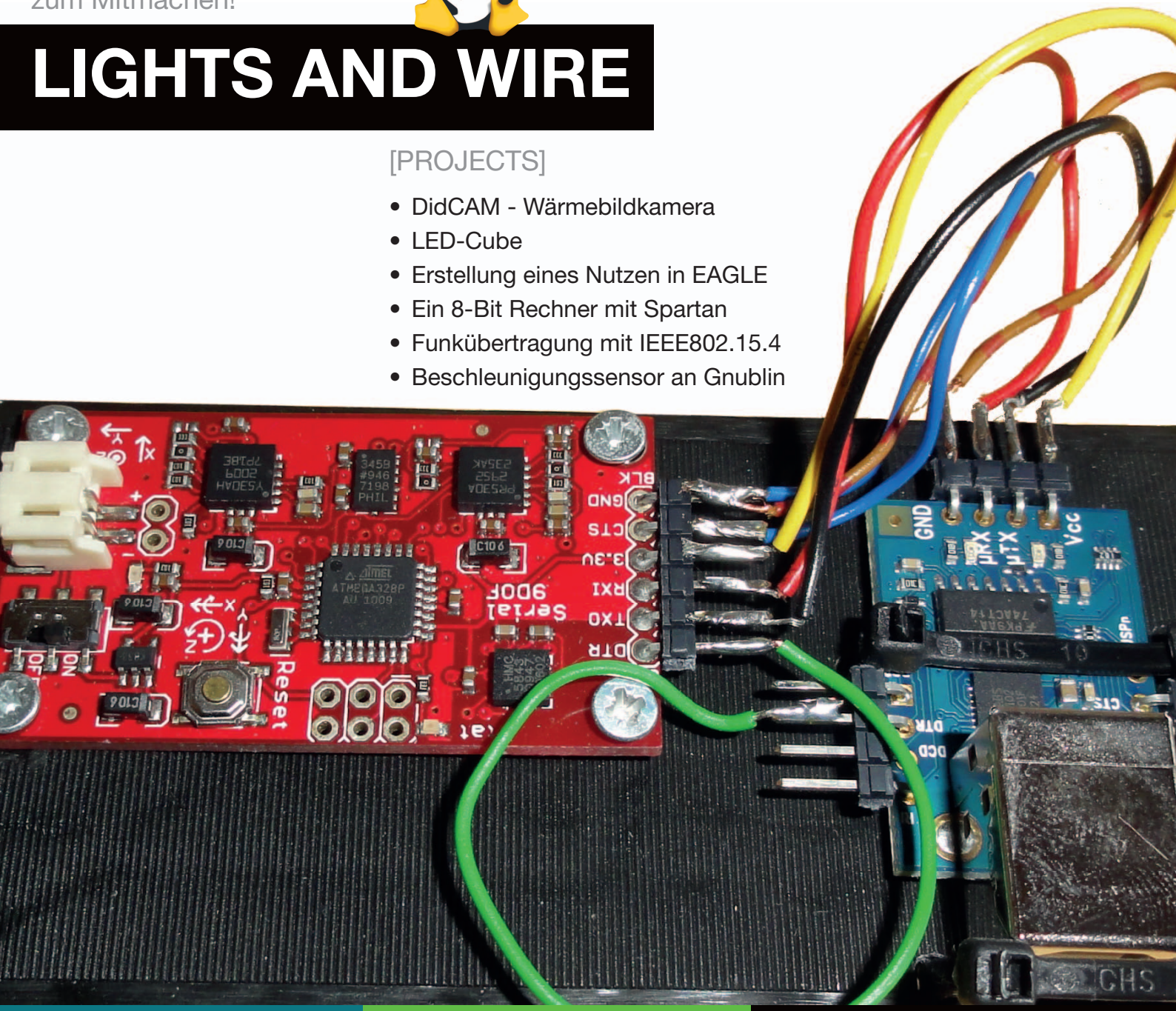
embedded projects
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Eine Open-Source Zeitschrift
zum Mitmachen!

LIGHTS AND WIRE

[PROJECTS]

- DidCAM - Wärmebildkamera
- LED-Cube
- Erstellung eines Nutzen in EAGLE
- Ein 8-Bit Rechner mit Spartan
- Funkübertragung mit IEEE802.15.4
- Beschleunigungssensor an Gnublin



Einleitung

Ausgabe 03/2012

Embedded Projects Journal - Ausgabe No. 14

Einleitung

Nachdem jetzt die kalte Jahreszeit startet, freut es uns besonders, in dieser Ausgabe den Bau einer eigenen Wärmebildkamera zeigen zu dürfen. Es ist doch immer wieder erstaunlich, wie man aus scheinbar sehr einfachen Bauteilen solche spannenden Sachen basteln kann :)

Viele weitere tolle Themen folgen und lassen das Journal zu einer guten Lektüre am Kaminfeuer werden.

Wir freuen uns weiterhin immer wieder über die verschiedensten Artikel und Themen von Euch.

Nur Mut zum Schreiben!

Dieses Jahr sind wir mit embedded projects zum ersten Mal auf der embedded world vertreten. Gerne kann sich jeder diesen Termin schon mal dick im Kalender notieren.

Eine weitere Neuigkeit wird es Rund um das Journal geben: Zu jedem Artikel wird in Zukunft ein Beitrag im Forum von mikrocontroller.net eröffnet werden. Dort können sich dann alle Interessierten Leser treffen und Erfahrungen, Ideen, etc. Rund um das Thema des jeweiligen Artikels austauschen.

Viel Spaß beim Lesen

wünschen Euch

Benedikt Sauter und

das embedded projects Team

→shop.embedded-projects.net

HARDWARE FOR YOUR PROJECTS – ONLINESHOP

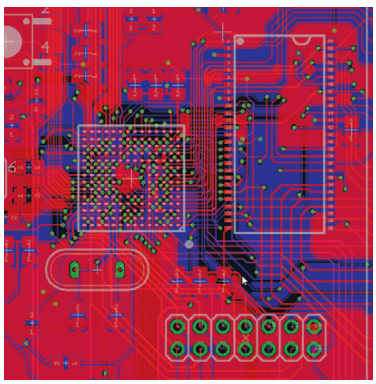
Design your GNUBLIN

Sie suchen ein Board, das fast so wie GNUBLIN ist und brauchen davon nur eine kleine Menge pro Jahr? Wir passen Ihnen auf kurzem Weg die Schaltung an und ergänzen diese nach Ihrem Wunsch. Dank unserer internen Bestückung können wir Ihnen bereits ab kleinen Mengen ähnliche Stückpreise wie bei den GNUBLIN Boards in diesem Shop liefern.

→ **www.gnublin.org/designer**

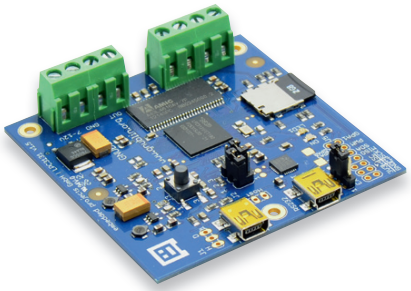
Die einfachste Möglichkeit an ein eigenes embedded GNU/Linux Board zu kommen!
Einfach und genial ist das Produkt „Design your GNUBLIN“. Für alle, die kleine Serien von Platinen mit embedded GNU/Linux benötigen. Basierend auf der Schaltung der Boards der GNUBLIN Familie können wir einfach und schnell kundenspezifische Lösungen erstellen. Innerhalb kürzester Zeit können wir Ihnen die Boards von 1 bis ca. 1000 Stück liefern.

NEU: online GNUBLIN-Designer mit Kalkulator



Wir zeichnen den Schaltplan und das Layout ...

... bestücken mit einem Automaten in Augsburg ...



... und nehmen die Schaltung für Sie in Betrieb.

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

DidCAM

Eine „low-cost“ Wärmebildkamera zum selber bauen

Helmuth Grötzebauch <h.groetz@fu-berlin.de>, Tomas Hahn <tomas.hahn@tu-dortmund.de>, Volkhard Nordmeier <volkhard.nordmeier@fu-berlin.de>

Eine Wärmebildkamera bietet aus didaktischer Perspektive interessante Möglichkeiten, viele Aspekte der Wärmelehre und eine breite Palette alltagsbezogener Kontexte auf eine völlig neue Weise experimentell zu untersuchen. Sie ‚misst' die Temperatur eines Objektes nicht wie ein Thermometer, sondern registriert die ausgesendete elektromagnetische Strahlung in einem bestimmten infraroten Spektralbereich, die sogenannte Wärmestrahlung. Die Interpretation der so generierten ‚Wärmebilder' in Form von Falschfarbendarstellungen erlaubt dann Aussagen zum Temperaturprofil. Darüber hinaus erweisen sich die farbigen Wärmebilder als ästhetisch sehr ansprechend. Obwohl professionelle Wärmebildkameras in den letzten Jahren deutlich preiswerter geworden sind, zählen sie noch lange nicht zur Standardausstattung einer physikalischen Schulsammlung, und eine Anschaffung sprengt derzeit noch (fast) jeden Physiketat. Um die zukünftige breitere Implementierung dieser Technik im Physikunterricht zu unterstützen, sollen in diesem Beitrag Anregungen für den Selbstbau einer „low-cost“ Wärmebildkamera gegeben und mögliche Anwendungsmöglichkeiten anhand von (einfachen) Versuchen mit einer solchen sehr preiswerten Alternative aufgezeigt werden. Auch mit dieser Kamera gelingt es, bislang ‚Unsichtbares' – die Wärmestrahlung – sichtbar zu machen.

Einleitung

Eine Wärmebildkamera ist heute noch immer eine kostspielige Anschaffung, und nur wenige Ausbildungseinrichtungen können sich diese Technik leisten. Die Kostenspanne bewegt sich zwischen ca. 1.500 € und mehr als 250.000 €. Dabei wird der Preisunterschied vom Detektortyp, der Anzahl der Messelemente (Auflösung in Pixel), der Empfindlichkeit des Detektors, vom einkalibrierten Messbereich und von der Qualität der Optik bestimmt. Ziel der hier vorgestellten Entwicklung war es, eine „low-cost“ Wärmebildkamera zu konstruieren, die im schulischen Umfeld ihren Einsatz finden kann. Sie sollte aus dem vorhandenen Etat beglichen werden können und trotzdem eine angemessene Qualität liefern. „Low-cost“ bedeutet in diesem Fall einen Betrag von unter 300 € für die selbstentwickelte Wärmebildkamera, gekoppelt mit einer in diesem Projekt entwickelten und frei erhältlichen Steuerungssoftware für Windows XP/ Windows 7. Bei „Low-cost“ Entwicklungen führt der günstige Preis in der Regel aber zu Kompromissen: Dies betrifft hier eine vergleichsweise lange Aufnahmezeit und eine niedrige Wärmebildauflösung. Als Kompensation zur Bildverbesserung wurde ein Hintergrundbild einer USB Kamera (Abb. 1) eingefügt. Ein Wärmebild benötigt zur Fertigstellung ca. 100 s. Das Bild besteht dabei aus ca. 2000 Pixeln (62x32 Pixel) und kann einfach geformte Gegenstände erfassen. Im Vergleich dazu: Eine professionelle Wärmebildkamera vom Typ ‚VarioCAM' (Jenoptik) weist 76800 Pixel (320x240 Pixel) pro Bild auf. Die Anschaffungskosten liegen dafür aber auch bei ca. 20000 - 25000 € je nach Ausstattung.

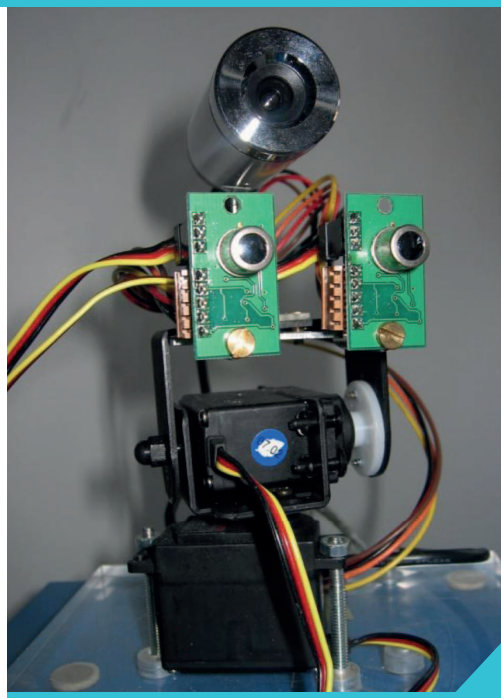


Abb. 1: DidCAM – eine Wärmebildkamera im Selbstbau

Physikalische Grundlagen Wärmestrahlung

a. Messprinzipien

Die Begriffe „Wärmestrahlung“ und „Infrarotstrahlung“ werden im Alltag und häufig auch in der Fachliteratur synonym verwendet, obwohl jeweils unterschiedliche Wellenlängenbereiche abgedeckt werden. Die sog. Infrarotstrahlung umfasst (gemäß DIN 5031) einen Wellenlängenbereich von ca. 800 nm bis 1 mm, die Wärmestrahlung einen Bereich von 800 nm bis 20µm. Im Folgenden beschränken wir uns bei unseren Betrachtungen auf den Bereich der Wärmestrahlung. Zur Messung von Wärmestrahlung werden die unterschiedlichsten Detektoren verwendet:

- Halbleiterdetektoren, z.B. Fotodioden, wandeln die auftreffenden Photonen direkt in ein elektrisches Signal um (photoelektrischen Effekt). Sie sind von der Konvertierungsgeschwindigkeit deutlich schneller als die passiven Bauelemente (Bolometer, Thermoelement) und liegen je nach Detektorsubstrat mit ihrer spektralen Bandbreite zwischen ca. 1 µm und 5µm.
- Beim Bolometer führt die einfallende Wärmestrahlung zu Widerstandsänderungen des Sensors (Thermistor), beim Thermoelement zu Spannungsänderungen (thermoelektrischer bzw. Seebeck-Effekt, s.u.), die messtechnisch erfasst werden. Bolometer und Thermoelement benötigen trägheitsbedingt eine längere Konvertierungszeit. Beide Detektoren zählen durch ihre berührungslose Messmethode zur Klasse

der sog. Pyrometer. Typische Anwendungsgebiete liegen im Messbereich oberhalb von ca. 7 µm Wellenlänge.

Die berührungslose Messmethode der Pyrometer unterscheidet sich deutlich gegenüber Temperaturfühlern, die einen direkten Kontakt zur Messstelle aufweisen, denn das Emissionsvermögen und der Absorptionsgrad von Strahlern sind sehr stark oberflächen- und wellenlängenabhängig. Bei der Messung mit Pyrometern sind daher die unterschiedlichen Emissionsgrade (materialabhängiges Wärmeabstrahlvermögen) der strahlenden Objekte zu berücksichtigen. Je nach Beschaffenheit des Materials ist die Wärmeabstrahlung bei einer gegebenen Temperatur unterschiedlich stark. So weist poliertes Gold einen Emissionsgrad von nur ca. 0,02 auf, ein gebrannter, roter Ziegelstein dagegen ca. 0,9 [1]. Zudem spielen das Transmissionsverhalten und auch die Reflexionen auf dem zu vermessenden Objekt eine wichtige Rolle. Darüber hinaus erweisen sich aber auch die Fokussierung der Kamera, die Abbildungsgröße des Objektes auf der Sensorebene, der Messwinkel „Field of view" (FOV) und die Objektentfernung (bzw. die damit einhergehenden atmosphärischen Beeinflussungen) als weitere wichtige Einflussgrößen für die Qualität von Messungen mit Pyrometern.

b. Thermoelement

Grundlage für die Messung von Wärmestrahlung mit Thermoelementen, wie sie bei der in diesem Beitrag vorgestellten Wärmebildkamera verwendet werden, ist der Seebeck-Effekt. Bei der Berührung zweier unterschiedlicher Metalle entsteht eine spezifische Kontaktspannung, die materialabhängig unterschiedlich hoch ausfällt. Diese Kontaktspannung ändert sich mit der Temperatur an der Kontaktstelle und lässt sich durch Thermodiffusionsströme an den Kontaktstellen der unterschiedlichen metallischen Leiter bei Temperaturänderungen erklären. Die Spannung ergibt sich aus dem Seebeck-Koeffizienten α und der Temperaturänderung ΔT bezogen auf eine Referenztemperatur T:

$U_{Seebeck} = \alpha \Delta T$ in V / K

Ändert sich die Temperatur am Kontaktpunkt um T durch Wärmezu- oder -abfuhr, so verändert sich auch die Thermospannung. Der prinzipielle Aufbau eines Thermoelementes sieht folgendermaßen aus (Abb. 2):

Als Thermospannung ergibt sich:

$$U_{AB} = \alpha A\, TU - TAB + \alpha B\, TAB - TBA + \alpha A\, TBA - TU$$

mit

αA: Seebeck-Koeffizient des Leiters A,

αB: Seebeck-Koeffizient des Leiters B,

TU: Umgebungstemperatur,

TAB: Temperatur am Übergang Metall A nach B,

TBA: Temperatur am Übergang Metall B nach A.

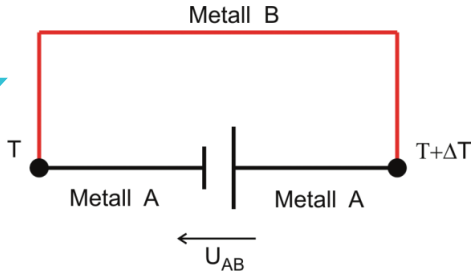


Abb. 2: Aufbau eines Thermoelementes

Aufbau der DidCAM

a. Thermoelemente und technische Daten

Für den hier vorgestellten Selbstbau einer Wärmebildkamera werden für die Aufnahme eines Wärmebildes zwei Thermoelement-Module verwendet (Abb. 1). Die Module vom Typ TPA81[2] basieren auf dem Sensor TPL08T von Perkin-Elmer [3] (vgl. Abschnitt „Komponenten zum Selbstbau"). Sie bestehen aus jeweils acht einzelnen Messzellen bzw. Thermoelementen. Die Elemente auf dem Sensor-Chip sind in einer 1 x 8 Einzelzeile angeordnet, die eine spektrale Empfindlichkeit zwischen 2 bis 22 µm aufweist, mit einer Transmission von ca. 50% über den gesamten Messbereich. [2] Das Messfenster (FOV) des einzelnen Elementes liegt bei 5,12° x 6°. Bedingt durch den Aufbau der Zeile und die über den 8 Messelementen liegende Linse, ergibt sich ein gesamter Öffnungswinkel in der Längsrichtung (5,12° x 8) von 41° x 6° (Abb. 3). Der Temperaturmessbereich liegt zwischen 4° C bis 100° C. Bei der Messtoleranz wird im Bereich von 4° bis 10° C ein Fehler von ± 3° C und im Bereich von 10° C bis 100° C ± 2° C angegeben. Nach unseren Erfahrungen kann die Zelle aber deutlich höhere Temperaturen detektieren, wobei dann die Messwerte im Toleranzband nicht mehr spezifiziert sind.

Die Datenkommunikation bezüglich Steuerung und Messwertübertragung zwischen Computer und Sensor erfolgt über einen I2C/USB Konverter [4].

b. Messbereich

Die Thermoelement-Module sind auf einem zwei-achsigen Servomotor (Typ HS-422) als „Pan und Tilt" (Dreh- und Kipp-) Bausatz vertikal befestigt. Mit 30 Schritten des horizontalen Motors und der Nullposition wird ein Messwinkel und Aufnahmebereich von 31° erreicht. Zur Erweiterung der Messauflösung des Arrays in vertikaler Richtung wurde ein weiterer Servomotor gleichen Typs auf den horizontalen aufgesetzt. Mit dieser Möglichkeit kann das Detektor-Modul auch in vertikalen Schritten weitere Messzeilen generieren. Die Erhöhung um einen ganzen vertikalen Messwinkelbereich (41° vertikal) hätte ein Messfenster mit 82° zur Folge, mit keiner verbesserten Auflösung. 82° ist im Normalfall nicht mehr der Bereich, der interessant ist, sondern der frontale Bereich, in dem sich das zu messende Objekt befindet. Folglich haben wir uns für die Beibehaltung des vertikalen 41° Messfeldes entschieden. Dafür wurde das Messfenster der einzelnen Detektoren in vertikaler Richtung (ca. 5°) verfeinert. Die Detektorzeile wird nun jeweils um 1° angehoben und erfasst die Zwischenwerte im Detektorfeld, so dass sich vier Messzeilen ergeben. Ergebnis ist eine um das vierfache verbesserte Auflösung im 41° Messfeld. Das Messfenster wird dadurch in Näherung auch um einen Betrag von 4° vergrößert. Der unterschiedliche Bezugspunkt von Drehachse und Sensor-

winkel (Abb. 3) führt zu einer näherungsweisen Messbereichserweiterung von 41°auf 45°.

Um den Messbereich auch in der horizontalen Richtung zu erweitern, wird das zweite Thermoelement-Modul verwendet (s.o.), das sich mit seinem Messfeld an das erste Messfeld idealerweise nahtlos anschließt und maximal zur Verdopplung des Messbereiches auf 62° führt. Hinzu kommt der halbe Sensoröffnungswinkel an beiden Messendpunkten von jeweils +3°, wenn der Sensormesswinkel symmetrisch zur Servomotor Messposition ausgerichtet wird. Daraus ergibt sich ein erweiterter Messwinkel von maximal 68° (Abb. 3).

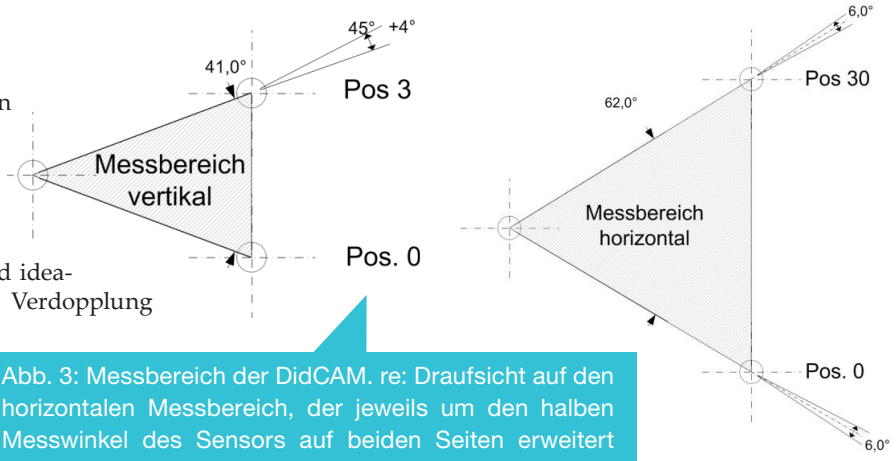


Abb. 3: Messbereich der DidCAM. re: Draufsicht auf den horizontalen Messbereich, der jeweils um den halben Messwinkel des Sensors auf beiden Seiten erweitert wird. li: Seitenansicht des vertikalen Messbereiches der Kamera mit der 4° Messbereichserweiterung.

Aufbau der Kamera

In diesem Abschnitt findet sich die Bauanleitung nebst der Beschreibung der notwendigen Komponenten für den Selbstbau der Kamera. (Die Dateien für den Betrieb und Servicefunktionen können auch von der Homepage der Autoren heruntergeladen werden.)

a. Komponenten zum Selbstbau

Die Anordnung der Komponenten zeigt Abbildung 4. Im Anhang finden sich zusätzlich schematische Aufbauhinweise (Abb. 23 und der zugehörige Schaltplan Abb. 26).

Die folgenden Abbildungen 5 und 6 zeigen weitere Details des Aufbaus.

Stck	Bezeichnung	Typ	Preis in €
2	Thermopile Zeile mit 8 Detektoren von „noDNA“ Art.Nr.: DEV-TPA81	TPA81	160 ,-
1	Schnittstelle Art.Nr.: USB-I2C von „noDNA“	USB nach I²C	30 ,-
1	Motorhalterung von „noDNA“ Art.Nr.: BPT-NS	Lynx BPan Tilt Bausatz	15 ,-
2	Analog Servomotor von „noDNA“ Art.Nr.: HS-422	HS-422	34 ,-
1	USB Kamera, 1,3 MPixel von „Conrad“ Best.Nr. 971975	USB-Webcam	20 ,-
	div. USB Leitungen, Stecker, Kabel, Winkel, Platine		20 ,-
	Gesamtsumme:		279 ,-

Tabelle 1: Komponenten zum Selbstbau (ungefähre Kostenkalkulation)

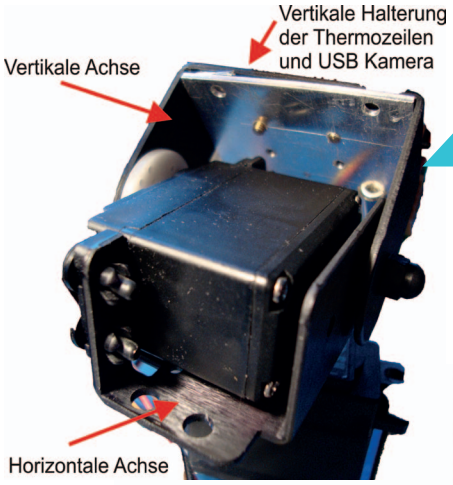


Abb. 5: Anordnung der beiden Servomotoren für den horizontalen- und vertikalen Messbereich

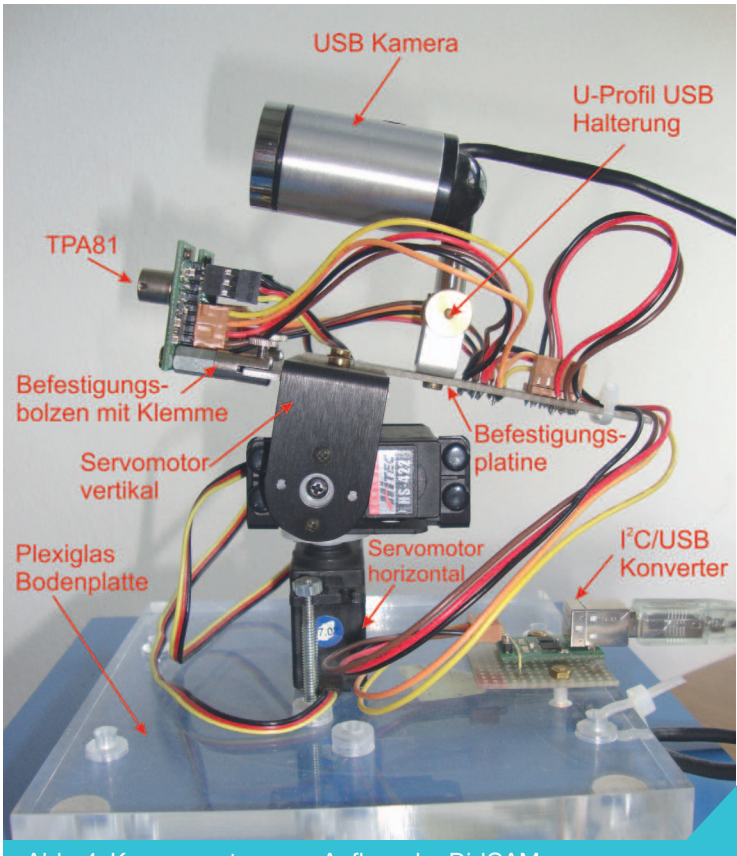
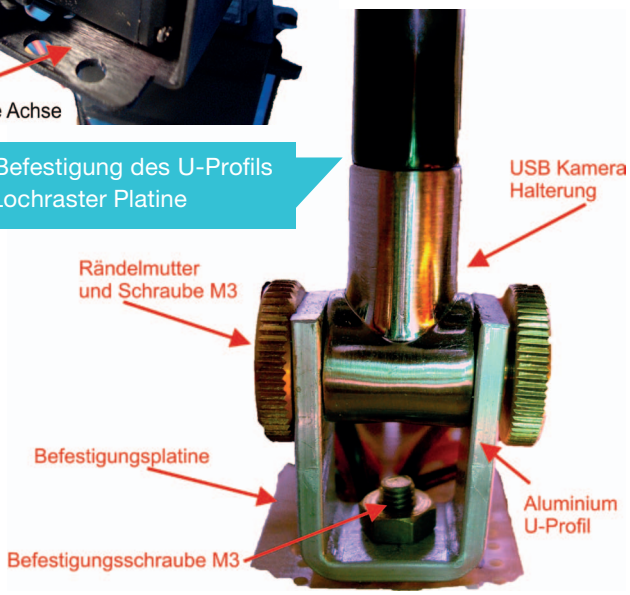


Abb. 4: Komponenten zum Aufbau der DidCAM

Abb. 6: Befestigung des U-Profils auf der Lochraster Platine



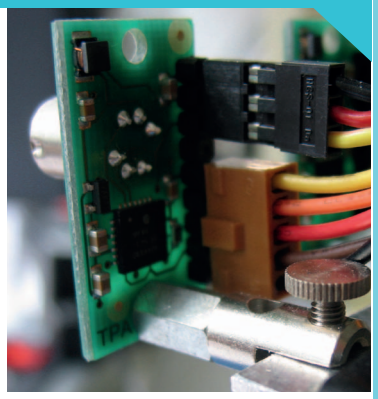
b. Servomotor

Die Servomotoren und die zugehörige Ansteuerungselektronik beziehen ihre Betriebsspannung über die USB-Schnittstelle. Über ein Pulsweiten-Modulationssignal (PWM) von 20 ms, das von der Ansteuerelektronik generiert wird, erhält der analoge Servomotor seine Winkelposition. Aufgrund des geringen Stromverbrauches im Verhältnis zum digitalen Servomotor fiel die Wahl auf analoge Servomotoren, die von der USB Schnittstelle versorgt werden können. Die Servomotoren werden von jeweils einem Thermosäulen- Modul, das mit einem Servomotor Treiber ausgestattet ist, betrieben. Der maximale Ausgangsstrom des Servotreibers liegt bei 70 mA. Zu den technischen Daten des Servomotors sei auf einen Link des Herstellers Hitec [5] verwiesen.

c. Mechanischer Messaufbau

Auf einer Bodenplatte aus Acrylglas, die genügend Eigengewicht für die Standfestigkeit bei der Servomotorbewegung haben sollte, wird der horizontale Servomotor verschraubt (Abb. 23). Auf seiner Montagefläche wird der zweite Servomotor für die vertikale Richtung montiert [6]. Mit Hilfe einer Aluminium Befestigungsverlängerung (Abb. 24) am vertikalen Motor wird eine Montageplatte mit zwei Befestigungsmulden für die Mess-Module verschraubt. An diesen Befestigungsmulden werden zwei Schraubklemmen mit Gewindestift so befestigt, dass sie arretiert, aber radial noch solange beweglich bleiben, bis sie ihre Endposition erhalten haben. Mit der zweiten Verschraubung der Klemme erhält der Thermozeilen Detektor seine Befestigung. Die Montage der Thermozeile erfolgt in senkrechter Anordnung, so dass die acht Messelemente übereinander angeordnet sind (Abb. 6). Diese Befestigungsart wiederholt sich für das zweite Modul. Die Stromversorgung und Datenübertragung erfolgt über genügend lange und flexible Litzen, die der Bewegungen des Schwenkkopfes mit beiden Servomotoren folgen können. Die Litzen münden in einem USB/ I2C Pfostenstecker, der auf der Bodenplatte montiert und mit einem Windows Rechner verbunden ist (Abb. 26).

Abb. 7: Montage TPA81 mit Befestigungsbolzen und Halogenklemme, die in die Mulde der Befestigungs- verlängerung des Servomotors geschraubt wird.



d. Mechanik der beiden Mess-Module

Mit der Verwendung von zwei Detektoren wird die genaue Justage beider Module zueinander wichtig. Da es schwierig ist, die jeweiligen Module winkelgenau, symmetrisch und nicht überlappend bzw. nicht lückenhaft zueinander einzustellen, um ein kontinuierliches Messbild zu erreichen, muss die Justage flexibler gehandhabt werden. Für diesen Zweck ist es erforderlich, dass die beiden Bilder der Detektoren sich leicht überlappen und bei der Auswertung am Bildschirm im Überlappungsbereich übereinander geschoben werden können. D.h. das Problem der Justage beider Module wird von der mechanischen auf die Softwareseite verlagert.

(Abb. 7, 8). Nachteil dieses Verfahrens ist der Verlust der sich überlappenden doppelt aufgenommenen Pixelspalten. Daraus folgt, je geringer die Überlappung gewählt wird, desto mehr Pixel stehen zur Auswertung zur Verfügung.

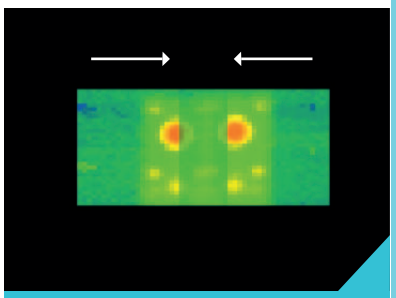


Abb. 8: Softwaremäßige Verschiebung der beiden Thermobilder (Justage)

e. Messwerterfassung und Steuerung

Zur Steuerung der Wärmebildkamera ist ein C++ (XP Windows) Programm entwickelt worden (s. Abschnitt Software.). Das Programm kommuniziert über eine USB Schnittstelle mit der Kamera und bestimmt den Schrittabstand der Servomotoren und folglich den maximalen Gesamtmesswinkel im horizontalen (abhängig von mechanischer Justage) und vertikalen Bereich. Die ermittelten Temperaturen werden in jeder Thermozeile in acht Registern abgelegt. Zusätzlich wird in einem weiteren Register die Umgebungstemperatur erfasst. Nach jedem im Steuerprogramm selbst definierten Servomotorschritt werden alle Messwerte vom Rechner ausgelesen. Ausgangspunkt der Programmierung ist eine eindeutige Definition des Messfeldes. Aus der schon definierten Winkelauflösung eines einzelnen Messelementes und des horizontalen Messfensters ergibt sich folgende Berechnung für die maximale Pixelanzahl Nmax bei den gegebenen Randbedingungen:

Nmax= 31 x 4 x 8 x 2 Pixel = 1984 Pixel

mit: 31 Messpunkte/Zeile horizontal, 4 Messzeilen vertikal, 8 Thermozeilen Detektoren und 2 Messmodule TPA81. Der Bildaufbau des Thermobildes erfolgt Zeilenweise. Nach wenigen Sekunden ergibt sich folgendes Bild (Abb. 9), das die zeilenweise Arbeitsweise der acht Thermoelemente veranschaulicht.

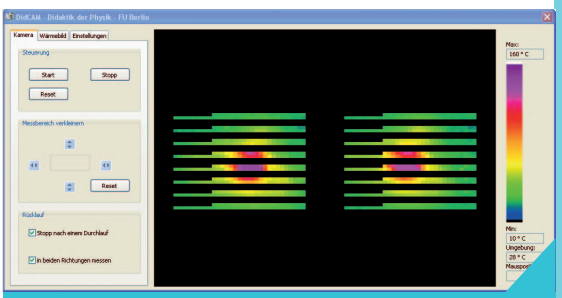


Abb. 9: Zeilenweises Scan Verfahren der acht senkrecht angeordneten Thermodetektoren von zwei Modulen

Software

a. Bedienoberfläche der Software

Das Programm zur Steuerung der Wärmebildkamera deckt einen Messbereich von +10C° bis 160°C ab. Werden diese Werte über- oder unterschritten, ist die Darstellung des Wärmepixels weiß bzw. schwarz. Anhand der Farben können optische Vergleiche zwischen der Skala und dem Wärmebild vorgenommen werden. Für genauere Ablesungen kann ein Cursor verwendet werden, der auf den ‚Wärmepixeln' positioniert wird. Diese Funktionen sind unabhängig von den eingestellten Bedienoberflächen, die sich in drei Bereiche aufteilen: Bedienung der Kamera Bearbeitung der Wärmebildaufnahme Hintergrundeinstellungen und Interface Eine Abbildung der Bedienoberflächen ist unter Abb. 21/22 zu finden.

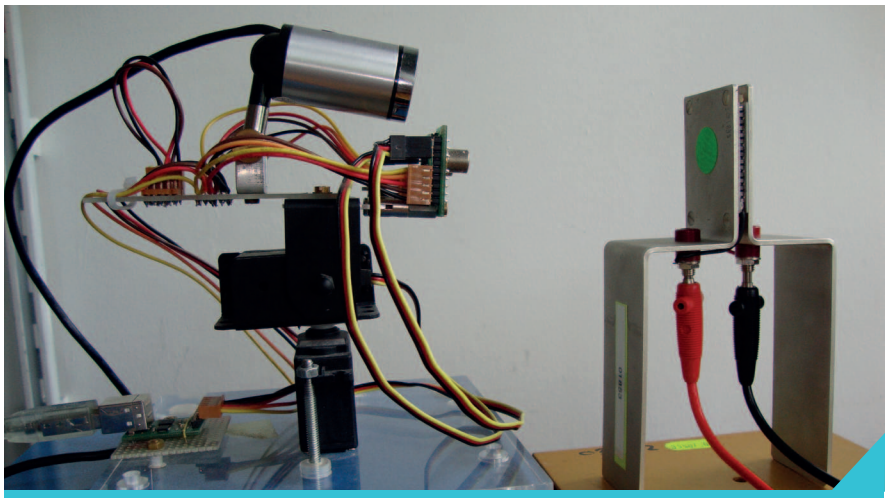


Abb.10: Funktionstest der DidCAM. Links: DidCAM mit Detektoren und USB-Kamera; rechts: Peltierelement mit eloxierter Oberfläche und grünem, aufgeklebten Messfleck aus Papier

Mit dem Aufruf des Programms muss als Erstes die COM-Schnittstelle der Wärmebildkamera bestimmt werden. Die COM-Schnittstelle wird über einen USB-Port aktiviert. Danach kann über den Menüpunkt „Kamera" die Messung gestartet werden.

Die USB Webkamera und die DidCAM werden jeweils separat über ihre USB-Schnittstellen angesprochen.

Zur Demonstration der Funktionsweise der hier vorgestellten Wärmebildkamera eignet sich beispielsweise ein Peltierelement (Abb. 10). An den Rändern ist es mit vier Kunststoffschrauben verbunden und in der Mitte wurde ein Messfleck mit verändertem Emissionsfaktor angebracht, der sich folglich von der restlichen Fläche im Messbild gut abheben sollte.

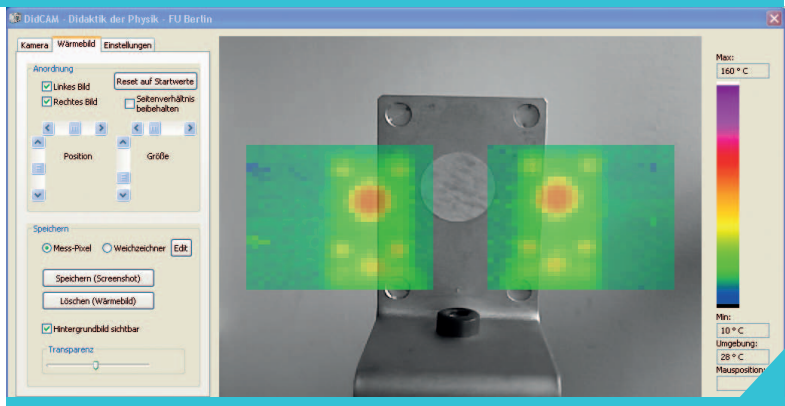


Abb. 11: Die transparentern Wärmebilder beider Thermozeilen im Grundzustand vor dem Foto der USB Kamera

Vor dem Beginn der Messung sollte der Bildausschnitt bestimmt werden, in dem gemessen werden soll. Dafür wird ein Bild mit der USB Kamera, die zum Messaufbau gehört, in der Auflösung 640x480 Pixel, in der Mittenposition des Messbereiches aufgenommen. „Bild Laden" legt die Aufnahme als Hintergrundbild fest.

Ergebnis sind zwei Wärmebilder, die transparent auf dem Foto des USB Kamera oder auch ohne Hintergrundbild dargestellt werden können (Abb. 11).

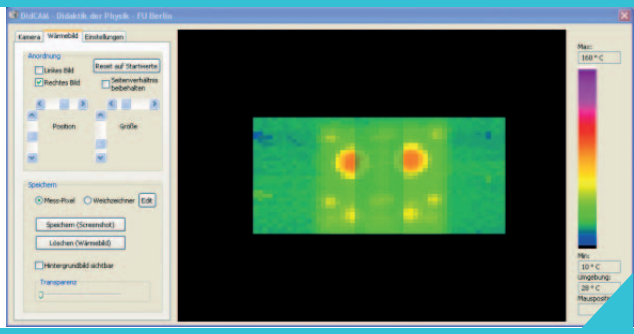


Abb. 12: Messbildjustage durch Bildverschiebung ‚per Hand' mit Hilfe der Software und Ausblendung des Hintergrundbildes

Da beide Kameras unterschiedliche Brennweiten aufweisen und Zoom-Funktionen fehlen, müssen die unterschiedlichen Abbildungsmaßstäbe zwischen beiden Aufnahmen angeglichen werden. Dazu werden zunächst die Wärmebildaufnahmen der beiden Detektoren ‚per Hand' mit Hilfe entsprechender Softwarefunktionen zur Deckung gebracht (Abb. 12), um danach über das Hintergrundbild gelegt zu werden.

Abbildung 13 zeigt zwei zur Deckung gebrachte Messbilder mit eingeblendetem Hintergrundfoto. Der Grad an Transparenz des Thermobildes kann per Software eingestellt werden. Die Auswertung der einzelnen, überlagerten Thermopixel erfolgt durch eine arithmetische Mittelwertbildung. Soll ein neues Wärmebild aufgenommen werden, besteht die Möglichkeit durch einen „Reset" die Startposition der Kamera wieder anzufahren. Ist nur ein bestimmter Bereich von Interesse („Area of Interest" - AOI), so kann mit dem Cursor ein Messrahmen definiert werden. Nur in diesem Rahmen erfolgen dann die weiteren Messungen. Dies kann die Messzeit erheblich reduzieren (Abb. 14). Im Extremfall kann der Rahmen auf ein Thermopixel eingeschränkt werden.

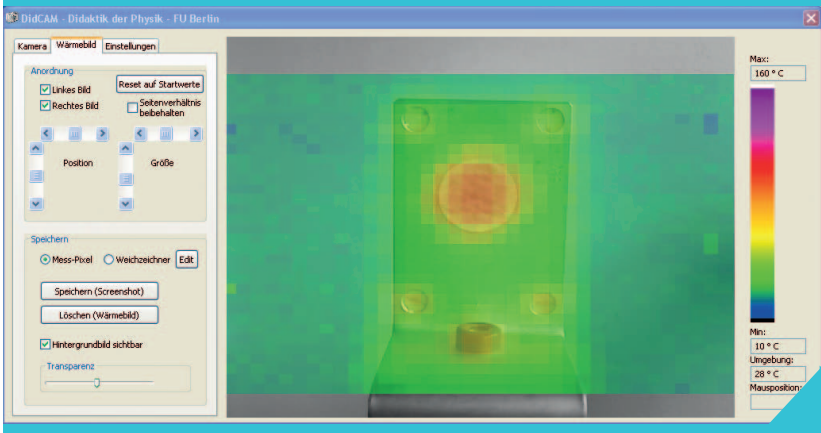


Abb. 13: Zur Deckung gebrachte Einzelwärmebilder der beiden Detektoren mit Hintergrundfoto.

Zur Abschätzung der Objekttemperatur können die Pixelfarben mit den Farben auf dem Farbbalken verglichen werden. In einem separaten Temperatur Messfenster wird die Umgebungstemperatur angezeigt. Zur Glättung bzw. ‚Weichzeichnung' der Wärmebilder wurde eine sog. Gauß-Glättung in die Software integriert (vgl. Abb. 16). Mit der Aktivierung des ‚Weichzeichners' erfolgt, abhängig von den Einstellparametern, z.B. folgende Ansicht (Abb. 15). Über eine entsprechende Parameterwahl können verschiedene Glättungseinstellungen vorgenommen werden. In Abbildung 16 ist ein einzelnes (links oben) Pixel in seiner beeinflussten Auswirkung zu erkennen. Die Auswirkung auf die Temperatur kann simuliert werden.

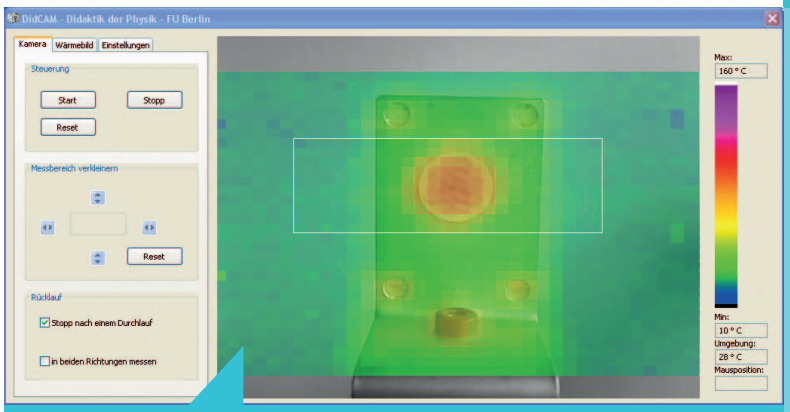


Abb. 14: Area of Interest – (AOI)

Abb. 15: (ohne Abb.)Temperaturverläufe auf dem Peltierelement in geglätteter Darstellung

b. Genauigkeit der Wärmebilder

Der in den technischen Daten angegebene Messbereich bis 100 °C wird deutlich überschritten. Er erreicht eine Temperatur von ca. 140 °C. Vergleichsmessungen mit der „VarioCAM" (Fa. Infratec) ergeben eine gute Übereinstimmung. Beim Vergleich der Aufnahmen zwischen DidCAM (Abb. 17) und VarioCAM (Abb. 18) sind die unterschiedlichen Farbskalen bei gleicher Temperaturspreizung zu beachten. Beide Aufnahmen zeigen bis 130°C gute Übereinstimmung. Beim Lampenschaft erreichen allerdings die Temperaturen, wie in Abbildung 18 ersichtlich, einen Wert von 172 °C. Diese Temperaturen sind von der DidCAM nicht mehr erfassbar und deshalb auch nicht darstellbar. Die Bereiche der niedrigeren Temperaturen stimmen überein.

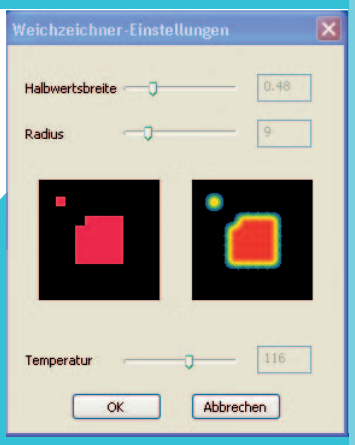


Abb. 16: Bei der Gauß-Glättung ist der Weichzeichnereffekt an einem einzelnen Pixel und einer Fläche zu sehen. Die Temperatur ist in dieser Voransicht frei wählbar.

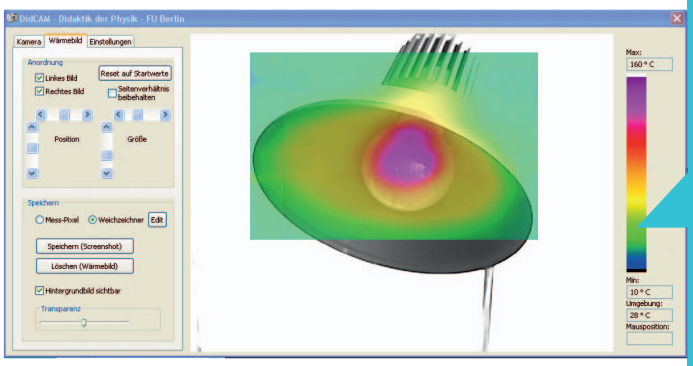


Abb. 17: DidCAM Darstellung der Schreibtischleuchte geglättet

Abb. 18: Zum Vergleich: Schreibtischleuchte mit der VarioCAM

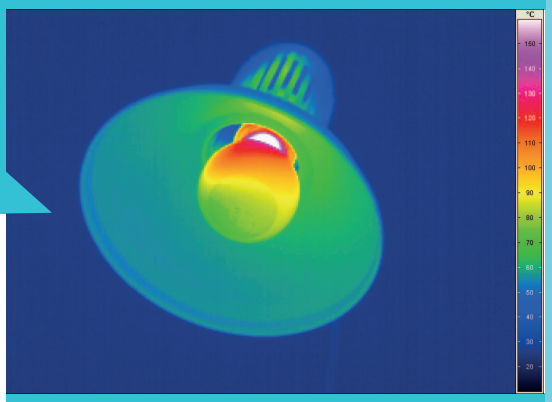


Abb. 19: Seltersflasche aus dem Kühlschrank als Speicherbild des Programms

c. Temperaturverläufe auf einer Selters Flasche

Bei einem weiteren Versuch untersuchten wir eine Selters Flasche aus dem Kühlschrank (Abb. 19). Zur genaueren Beurteilung wurde keine Gauß-Glättung verwendet. Die Temperaturverläufe des Wärmebildes wurden mit den Messungen eines Temperaturfühlers überprüft und stimmten auf 1°C überein. Die Messungen auf der Flaschenoberfläche ergaben beim Flüssigkeitsbereich einen Wert von 13°C und im oberen Teil 17°C.

d. Aufnahmen von gasförmigen Medien

Problematisch ist es, Temperaturverläufe richtig darzustellen, wenn es sich um gasförmige Medien wie z.B. Flammen handelt. Ursache ist die Unbestimmtheit des Emissionsgrades und die Dichte des Mediums, die auch schwanken kann und die Abkühlung der äußeren Gasschicht. Die eindeutige Bestimmung mit den beschriebenen Sensoren ist nicht mit gutem Ergebnis möglich. Dafür gibt es spezielle Gassensoren, die die Farbtemperatur des Mediums messen und daraus die Temperatur ermitteln [7]. Trotz dieser gravierenden Hindernisse können qualitative Aussagen mit Hilfe der Thermosensoren (Thermoelemente) vorgenommen werden, die zumindest eine Aussage des erwärmten Objektes in Relation zur Umgebungstemperatur erlauben.

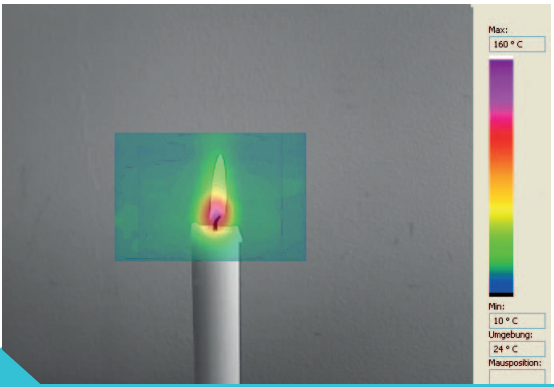


Abb. 20: Der schwarz/weiß Aufnahme der Kerze wurde das Wärmebild überlagert und mit dem Gauß Filter geglättet. Die gemessene Temperatur ist nicht die tatsächliche Temperatur.

Resümee

Die DidCAM ist eine Kamera, die mit recht geringer Auflösung ihren Informationsgehalt durch ein Hintergrundbild deutlich erhöhen kann. Der Preis von unter 300 € liegt in einem Preissegment, dass von einem Schuletat finanziert werden kann. Die Kamera vermittelt in der Anwendung auch mehr als nur ein Infrarotbild zu erzeugen. Durch den Scanmodus der beiden Servomotoren kann der Entstehungsprozess des Infrarotbildes Schritt für Schritt am Monitor verfolgt werden.

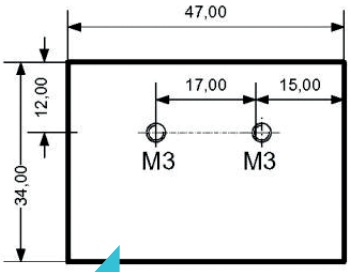


Abb. 24: Befestigungsverlängerung für die Detektoren TP81; Al, Plattendicke = 1,5mm

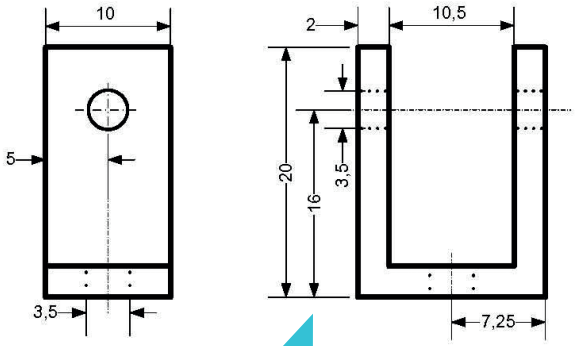


Abb. 25: U-Profil für USB Kamera

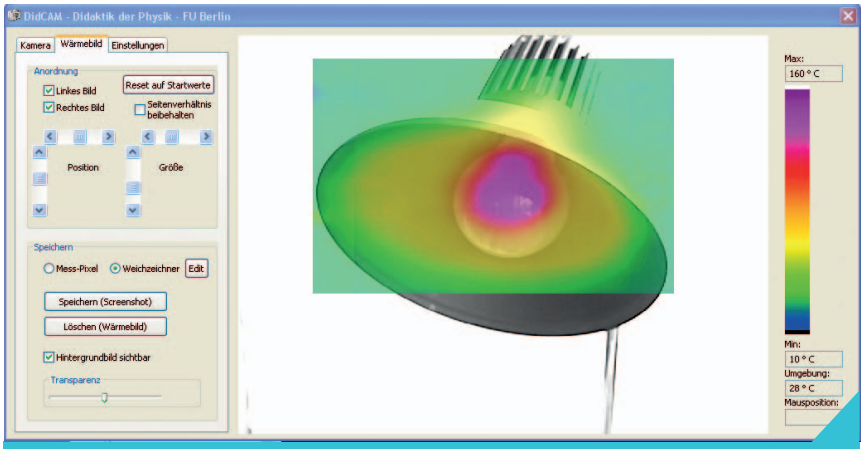


Abb. 21: Bedienoberfläche der DidCAM

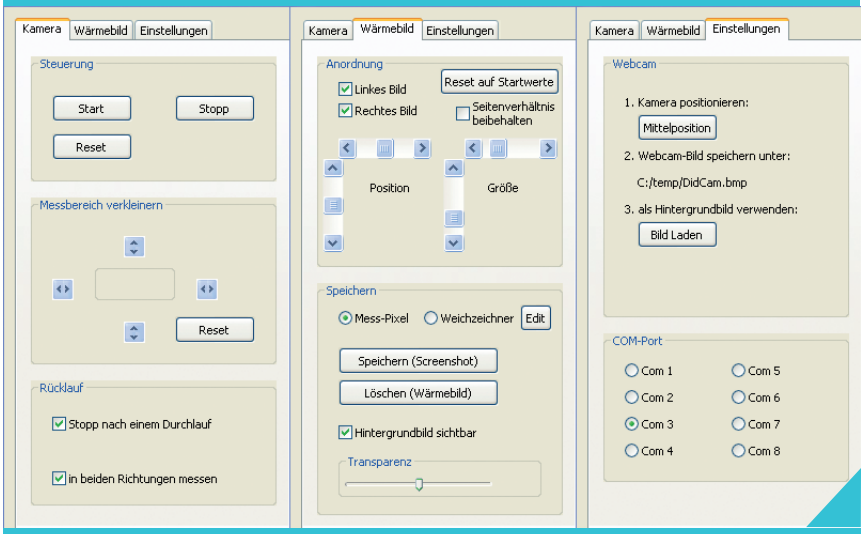


Abb. 22: Überblick über alle drei Bedienoberflächen

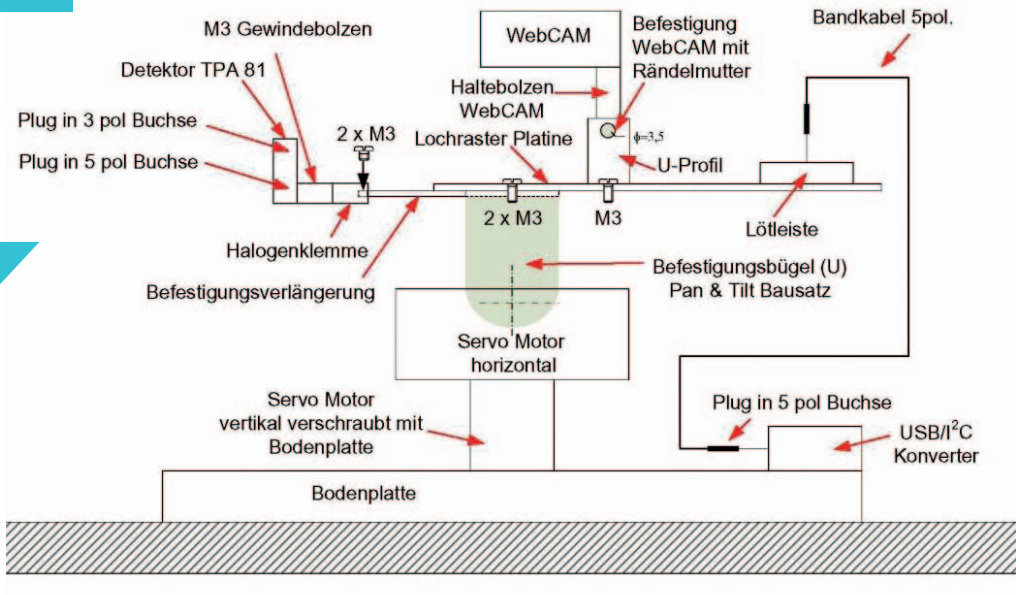


Abb. 23: Aufbauhinweise Aufbauschema DidCAM, FU Berlin, Didaktik der Physik, H. Grötzenbach

Literatur

[1] Infrarot Grundlagen: Optris. url: http://www.optris.de/fachartikel?file=tl_files/pdf/Downloads/IR-Grundlagen.pdf.

[2] TPA81. Thermopile Array Technical Specification: Robot Electronics. url: http://www.robot-electronics.co.uk/htm/tpa81tech.htm..

[3] INFRARED SENSING TECHNOLOGIES: PerkinElmer. url: http://www.perkinelmer.co.uk/CMSResources/Images/44-4345CAT_SensorsAndEmittersInfraredSensing.pdf.

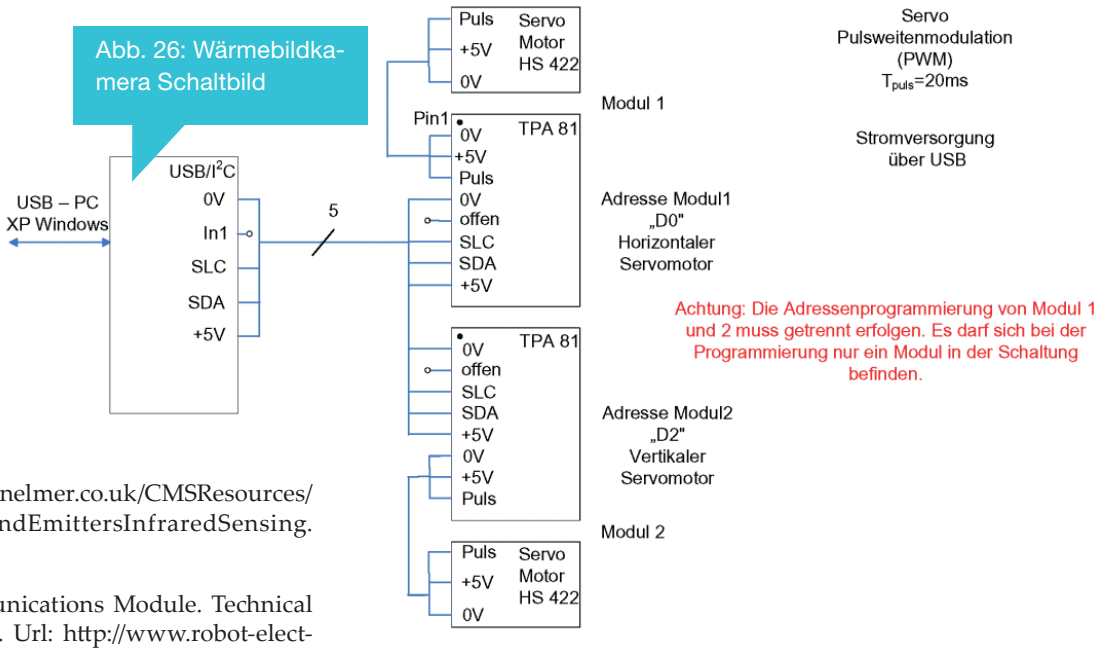
[4] USB-I2C USB to I2C Communications Module. Technical Specification: Robot Electronics. Url: http://www.robot-electronics.co.uk/htm/usb_i2c_tech.htm.

[5] Hitec. HS-422. url: http://www.hitecrc.de/store/product.php?productid=21140&cat=0&page=1.

[6] Lynx B - Pan and Tilt Assembly Instructions Rev. 1. url: http://www.lynxmotion.com/images/html/build16a.htm.

[7] IR imaging of gases: potential applications for CO2 cameras, M. Vollmer, K.-P. Möllmann (Inframation 2009, Proc. Vol 10, p.113 – 124)Anhang A –Auswertung der Wärmebild-aufnahmen und Bedienung des Programms

Abb. 26: Wärmebildkamera Schaltbild



LED-Cube

8x8x8 mit vielen Features

Dipl.-Ing. (FH) Florian Halfmann <florian.halfmann@googlemail.com>

Einführung

Anfang des Jahres 2011 dachte ich mir, dass es mal wieder Zeit wird ein neues Elektronik-Projekt anzugehen. Es sollte diesmal etwas mit visuellen Effekten zu tun haben. Da kam es mir gerade recht, dass einer meiner Freunde mir ein YouTube-Video mit einem 8x8x8 LED-Cube zeigte. Schnell war klar: So einer muß her :) (siehe Abb. 1)

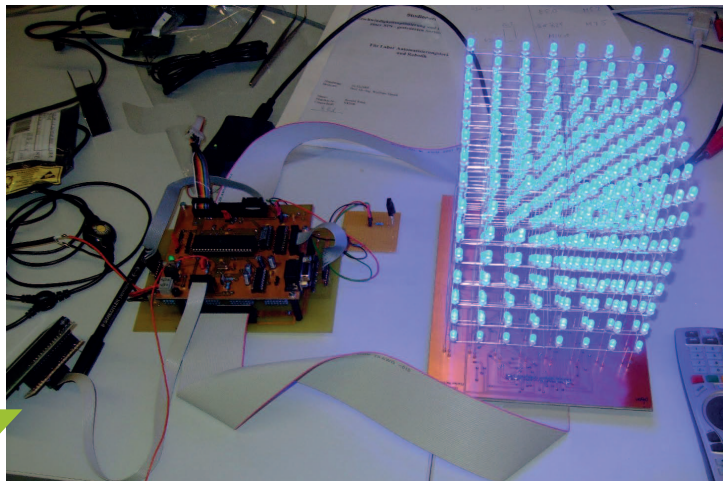


Abb. 1: Der LED-Cube

Doch was ist überhaupt ein LED-Cube?! Im Allgemeinen würde ich einen LED-Cube als exklusiven Einrichtungsgegenstand sehen. Er macht sich prächtig als Blickfang in jedem Wohnzimmer oder auf jeder Party.
Im engeren Sinne ist ein LED-Cube eine dreidimensionale Anordnung von LEDs, die einfache dreidimensionale Animationen darstellen kann.
Diese Cubes werden in allen erdenklichen Größen und Farben gebaut. Es gibt sogar Leute die RGB-LEDs verwenden. Um einen Eindruck zu bekommen einfach mal bei YouTube nach „LED Cube“ suchen und man bekommt etliche Videos präsentiert. Doch was hebt den hier vorgestellten Cube von anderen Lösungen ab? Nun, im Internet findet man immer wieder Cubes die immer wiederholend verschiedene Animationen darstellen. Um das ganze etwas aufzupeppen entschied ich mich dem Cube diverse Features mit auf den Weg zu geben: 4x20 Display zur Anzeige von Betriebszustand etc., RC5-Infrarot-Fernbedienung, SD-Card Interface zur Speicherung von Animationen, PC-Modus um Animationen die auf einem PC vorhanden sind über die serielle Schnittstelle abzuspielen, Dimm-Funktion um den Cube im Gesamten zu dimmen, Mikrofon um Amplituden oder vielleicht sogar das Spektrum von Musik darzustellen (FFT), Buchstaben/Laufschrift auf dem Cube darstellen.

Die Größe des Cubes und die passenden LEDs

Doch wie anfangen? Zuallererst stellte sich die Frage über die Größe des Cubes und die Farbe der LEDs. RGB-LEDs wollte ich nicht verwenden, da ich persönlich einfarbige Cubes schöner finde. Als Größe bot sich im Hinblick auf die Software 8x8x8 - also 512 LEDs - an, da dann prima und performant mit einem 8-bit Mikrocontroller gearbetet werden kann. Kurzum stöberte ich im Internet nach Shops für optoelektronische Bauelemente und wurde bei http://kt-elektronic.de/shop/ fündig. Dort orderte ich diverse diffuse LEDs als Muster. Diese waren auch kurze Zeit später schon da und ich konnte mit dem Ausmessen dieser mit Hilfe einer Konstantstromquelle beginnen. Am besten gefielen mir superhelle diffuse blaue LEDs. Kurzerhand wurden 600 Stück davon bestellt und damit konnte die Arbeit beginnen (Abb. 2) - ein Schaltplan musste her.

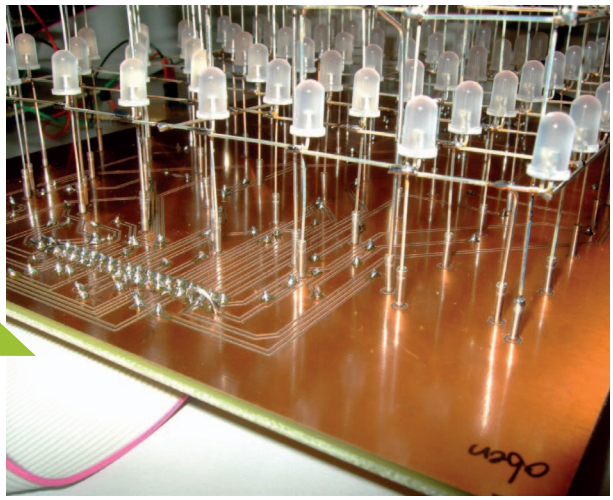


Abb. 2: Nahaufnahme

Zur Theorie eines LED-Cubes

Es ist möglich, später jede einzelne LED separat anzusteuern. Natürlich wäre es kaum möglich zu jeder einzelnen LED eine eigene Zuleitung zu legen. Dann würde der Cube fast nur noch aus Kabeln bestehen. Dafür gibt es bessere Alternativen wie z.B. das zeitliche Multiplexen einzelner Ebenen im Cube. Das bedeutet nichts anderes, als dass man die Kathoden aller 64 LEDs pro horizontaler Ebene im Cube miteinander verbindet und über einen FET auf Masse legt. Die Anoden aller vertikal übereinanderliegenden 8 LEDs (im folgenden „Säulen“ genannt) werden ebenfalls miteinander verbunden und über einen Quellentreiber wie z.B. UDN2981 und ein Schaltglied auf Vcc geschaltet. Man hat am Ende also 8 FETs für die 8 Ebenen und 64 Säulen, die unabhängig gegen Vcc geschaltet sind. Da das menschliche Auge ab ca. 100Hz keinerlei Flimmern mehr wahrnimmt, kann man sich diesen Effekt zu nutze machen, um per zeitlichem Multiplexing ein für das menschliche Auge stehendes Bild im LED-Cube zu erzeugen. In meinem Fall geschieht die Ansteuerung der Säulen über acht Schieberegister ICs vom Typ 74HC595 mit dahintergeschalteten UDN2981 Source-Treibern. In die Schieberegister werden die Bits einer Ebene

hereingeschoben, dann schaltet der FET die Ebene gegen Masse und die LEDs leuchten. Während die Ebene aktiv ist, werden in die Schieberegister schon die Daten für die nächste Ebene geladen - das ist möglich, da die Schieberegister noch ein Ausgangslatch besitzen. Man kann also Daten hineinschieben und später erst an den Ausgang übernehmnen. Jede Ebene meines Cubes wird mit 800Hz gemultiplext. Das bedeutet eine Bildwiederholfrequenz bei acht Ebenen von 100Hz.

Der Mikrocontroller und weitere Bauteile

Zur Versorgung des Cubes verwende ich ein universal Notebook-Netzteil von Reichelt mit 9,5V. Als Mikrocontroller hatte ich noch einen Atmel ATmega644 in meiner Bastelkiste, der sich für dieses Vorhaben perfekt eignet. Er bietet genug Flashspeicher um viele Animationen direkt in der uC-Software zu implementieren und hat auch ansonsten alle nötige Peripherie an Bord. Getaktet wird er von einem Quarz mit 14,7456 MHz. Die FETs der Ebenen steuere ich über einen 3zu8 Decoder 74HC138 an. Diesem Baustein gibt man über 3 Portpins eine Ziffer zwischen 0 und 7 im Dualsystem und er schaltet den entsprechenden Ausgang dann aktiv. Da der 74HC138 low-Aktive Ausgänge hat, wurde noch ein inverter-IC vom Typ 74HC540 dahintergeschaltet, an dem dann die Gates der Ebenen-FETs hängen. Wie schon erwähnt werden die Daten einer Ebene in Schieberegister vom Typ 74HC595 geschoben. Dazu werden ebenfalls drei Portpins verwendet: RCK, SER und SCK. Am SER-Pin wird das Bit angelegt, das bei einem Takt an SCK dann ins Schieberegister übernommen wird. Wenn alle Bits eingeschoben wurden, werden mit einem Impuls an RCK die Daten an den parallelen Ausgang übernommen. Zusätzlich bieten die Schieberegister auch praktischerweise noch einen G-Pin, mit dem man den Ausgang komplett an- und abschalten kann ohne die gespeicherten Daten zu beeinflussen. Diesen G-Pin mache ich mir zu Nutze, um den Cube später -mittels einer PWM- zu dimmen. Als weitere Bauelemente finden sich ein MAX232 zur Pegelwandlung des UART, ein TSOP1736 für die Infrarot-Fernbedienung, ein 7805 Festspannungsregler zur Versorgung des 5V-Pfades der Elektronik und ein zusätzlicher FET, der die Hintergrundbeleuchtung des LCD Displays schaltet. Das Display ist ein standard 4x20 Zeichen-LCD mit HD44780 Controller und wird im 4-bit Modus angeschlossen.

Die Umsetzung der Ansteuerplatinen

Ich enschloss mich die Elektronik auf zwei Platinen zu verteilen (Abb. 3 und Abb. 4). Auf die erste Platine kamen die Schieberegister, die UDN2981 und die Ebenen-FETs, also alle Bauteile, die zur Ansteuerung der LEDs selbst nötig sind. Auf die andere Platine kam der Mikrocontroller mit allem drumherum. Somit könnte man später einfach eine andere Mikrocontrollerplatine mit ganz anderem Controller nehmen, ohne gleich die Ansteuerelektronik der LEDs mit zu tauschen. Ein weiterer postiver Effekt ist, dass die Platinen später in einem schicken Sockel unter den LEDs verschwinden sollen und jede Platine fast Euro-Größe hat. Somit kann ich die Platinen jetzt „huckepack“ stapeln und spare Platz. Die Platinen an sich sind doppelseitig. Ich habe auf der Arbeit die Möglichkeit diese professionell zu ätzen - ein Aufbau auf Lochraster oder Streifenraster ist jedoch auch grundsätzlich möglich, da ich komplett auf SMD-Bauelemente verzichtet habe. Durchkontaktierungen wurden per Hand gemacht (Draht durchlöten), alle ICs sind gesockelt.

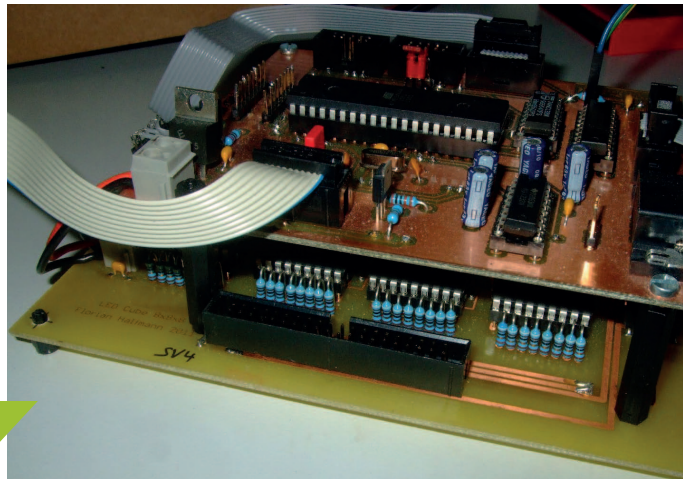


Abb. 3: Platine

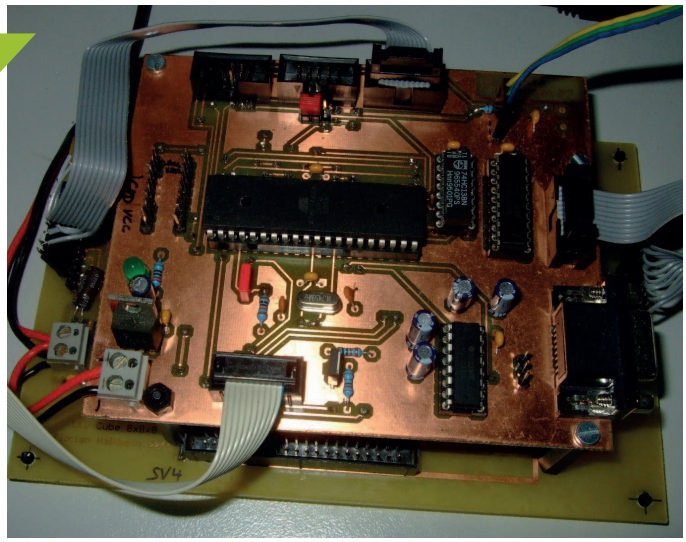


Abb. 4: Die Platine - Draufsicht

Das Löten der LED-Matrix

Dieser wohl handwerklich kniffeligste Teil des Projekts ist zeitlich nicht zu unterschätzen. Ich habe über 20 Stunden benötigt, bis der Cube fertig war. Man sollte zumindest über grundlegende Lötkenntnisse und vernünftiges Handwerkszeug verfügen. Der größte Feind einer sauberen Arbeit ist die Ungeduld. Lieber mal längere Pausen machen als später einen krummen und schiefen Cube zu haben. Wenn man 5mm LEDs benutzt sollte man einen Abstand von 20mm für die LEDs wählen, da sich dann die Länge der Anode und Kathode gut ausnutzen lässt. Eine meiner Meinung nach perfekte Art des Aufbaus hat die Firma qube-solutions beschrieben. Auf deren Homepage kann man sich unter

http://www.qube-solutions.de/qube-5/downloads

ein PDF-Dokument herunterladen in dem der Aufbau Schritt für Schritt beschrieben ist. Die Schablone für das Verlöten der LEDs hat ein Arbeitskollege mir praktischerweise mit einer CNC-Fräse aus einem Stück Multiplexholz gefertigt.

Berechnung der Vorwiderstände für die LEDs

Da es 64 Säulen gibt und immer nur eine Ebene gleichzeitig aktiv ist, benötigen wir auch entsprechend nur 64 Vorwiderstände. Hier ist aber zu beachten, dass durch das Zeitliche Multiplexing der Ebenen die Leuchtstärke bei Nennstrom auf 1/8 der Nennleuchtstärke reduziert, da jede Ebene nur 1/8 der Zeit leuchtet. Um diesen Effekt auszugleichen betreibt man die LEDs mit viel höherem Strom als dem Nennstrom. Effektiv kommt dann nach dem Multiplexen später wieder eine vernünftige Leuchtstärke heraus. Als Grenze für den Strom habe ich 350mA pro UDN2981 eingeführt, damit diese nicht zur Heizung werden. Da an jedem Quellentreiber 8 LEDs hängen, die theoretisch alle gleichzeitig leuchten können, bleibt pro LED 43,75mA (350mA/8 LEDs) übrig. Durch das Multiplexing ergibt sich eine effektive Stromstärke von 5,47mA (43,75mA/ 8 Ebenen), die in Licht umgewandelt werden. Für den Vorwiderstand wird aber 43,75mA als Berechnungsgrundlage genommen, da dies die tatsächlich fließende Stromstärke ist. Die LEDs vertragen dies ohne Probleme, da dieser Strom ja nur 1/8 der Zeit fließt und die LED sich die restlichen 7 Ebenenzyklen „erholen“ kann. Jetzt benötigt man die Flußspannung der LED bei einer Stromstärke von 43,75mA. Diese Spannung kann man messen, wenn man die LED an einer Konstantstromquelle betreibt oder aber auch im Datenblatt ablesen, falls man die Möglichkeit hat. die UDN2981 liegen an ihren Vertärkerstufen direkt an den 9,5V, die vom Netzteil kommen. Es fallen grob gesagt 2V an den Vertärkerstufen ab und somit bleiben noch 7,5V hinter den Quellentreibern übrig. Jetzt muss die Differenz dieser Spannung zu der Flußspannung der LED am Vorwiderstand vernichtet werden. Der FET, der die Ebene auf Masse schaltet kann vernachlässigt werden, da an diesem quasi keine Spannung abfällt. Bei meinen LEDs ergab sich eine Flußspannung von ca. 3,5V bei 43,75mA.

Rechnung:

((9,5V - 2V) - 3,5V) / 43,75mA = 91,43 Ohm.

Ich habe dann 91 Ohm Widerstände bei reichelt geordert.

Hinweis: Günstiger wäre es die UDN2981 mit einer kleineren Spannung als 9,5V zu versorgen, um weniger Verluste an den Vorwiderständen zu haben. Wenn alle LEDs im Cube über längere Zeit ununterbrochen leuchten, werden die UDN2981 und die Vorwiderstände sehr heiß (über 100 °C). Da man in der Praxis aber meistens Muster darstellt, bei denen nur wenige LEDs simultan leuchten, habe ich diesen Aspekt vernachlässigt. Wenn ich es nochmal machen würde, würde ich mich aber nach einem Netzteil mit ca. 7V umsehen. Eventuell rüste ich auch noch einen Temperatursensor nach, der bei zu hoher Temperatur den Cube automatisch dimmt um die Wärmeentwicklung zu reduzieren. Doch das steht noch in den Sternen...

Organisation der Software

Ein Frame besteht aus 512Bit, da es 512 LEDs bzw „Pixel“ gibt. Diese gilt es nun effektiv im Speicher des Mikrocontrollers zu organisieren. Es würde sich anbieten ein dreidimensionales Array cubeDate[8][8][8] zu verwenden, doch dann würde für jedes Pixel ein komplettes Byte verschwendet werden und RAM ist bei den meisten Mikrocontrollen knapp. Effektiver ist es, ein zweidimensionales Array zu nehmen, bei dem die dritte Dimension die einzelnen Bits eines Bytes sind. So habe ich es auch realisiert: uint8_t[][] cubeData[8][8]. Die erste Dimension ist der Layer und die zweite Dimension ist die „Reihe“. An dieser Position liegt im Speicher dann ein Byte, das die einzelnen Pixel der Reihe darstellt. Somit benötigt man nur 64 Byte im RAM des Controllers um einen Frame vorzuhalten. Die Animationen, die dargestellt werden arbeiten dann in der Regel direkt auf diesem Array. Falls eine Animation mal länger zur Berechnung benötigt, würde man im Cube unschöne Effekte sehen, da die Berechnungsdauer der Animation vom Menschen wahrnehmbar ist. In diesem Fall wird der Frame der Animation in einem temporären Puffer aufgebaut und anschließend per memcpy in den realen Puffer kopiert. Die Multiplexroutine besteht aus einem Timer der mit 800Hz einen Interrupt feuert. (100Hz Bildwiederholrate bei 8 ebenen). In der Interrupt Service Routine wird der aktuelle Layer inkrementiert, der ensprechende FET durchgeschaltet und anschließend werden die acht Bytes für die nächste Ebene schonmal aus dem CubeDate[8][8] Array in die Schieberegister geladen. Wenn dann der nächste Interrupt kommt, können diese Daten in die Ausgangslatches der Schieberegister übernommen werden und diese Ebene kann direkt aktiv geschaltet werden. Somit hat man keine „Leerlaufzeit“ um die Schieberegister zu befüllen die man hätte, wenn man dies am Anfang der ISR machen würde. Es ist peinlichst genau darauf zu achten, dass diese ISR perfekt funktioniert. Wenn eine Ebene mal länger oder sogar dauerhaft eingeschaltet bleibt, bekommen die LEDs dauerhaft den überhöhten Strom ab und könnten zerstört werden. Also: Niemals den Interrupt fürs Multiplexen deaktiveren oder ähnliches. Die Routine ist im Listing dargestellt. Des Weiteren laufen zwei weitere Timer. Einer von diesen kümmert sich um die Auswertung des RC5 Codes mittels einer Routine von Peter Dannegger (mikrocontroller.net). Der andere Timer erzeugt eine PWM, die auf die G-Eingänge der Schieberegister gegeben wird. Somit ist ein dimmen des Cubes möglich, indem dem Multiplexing eine PWM mit höherer Frequenz überlagert wird. Das Tastverhältnis der PWM kann über die Software verändert werden.

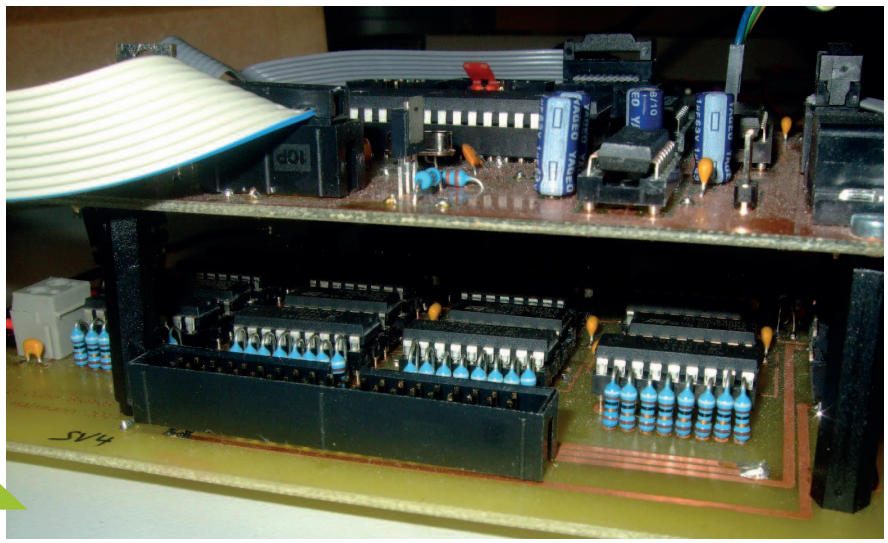


Abb. 5: ICs unterhalb der Platine

Animationen

Mittlerweile sind eine ganze Reihe von tollen Animationen entstanden wobei ich gestehen muss, dass ich einige Algorithmen beim Instructables 8x8x8 Cube abgeguckt habe (http://www.instructables.com/id/Led-Cube-8x8x8/). Doch warum das Rad neu erfinden wenn es schon effektive Algorithmen gibt. Dem Autor der Software vom Instructables Cube möchte ich deswegen hier ein Kompliment aussprechen. Die Basis für die Animationen bilden einige Low-Level Funktionen wie z.B. setLayer_z(), das die LEDs einer kompletten Ebene setzt oder togglePixel(x,y,z), das den Zustand eines Pixels im Raum umkehrt. Diese Reihe von Low-Level Funktionen arbeitet direkt auf dem cubeDate[8][8] array, das von der Multiplexroutine auf dem Cube dargestellt wird. Die eigentlichen Animationen machen sich die Low-Level Funktionen zu Nutze und kombinieren diese in einer Weise, die für den Menschen ein beeindruckendes Lichtschauspiel ergibt. Die Bezeichnung der aktuell dargestellten Animation und die Helligkeit werden ständig auf dem Display dargestellt. Auch der ASCII-Code wurde größtenteils implementiert. Die Bitmuster der Buchstaben sind im EEPROM abgelegt und können mittels einer Animation z.B. als eine Art Laufschrift dargestellt werden. Um das Starten der Animationen leichter zu gestalten, habe ich eine Art „Launcher“ implementiert, der den Animationen Nummern zuordnet. Das macht es auch einfacher mit der Fernbedienung Nummern durchzuschalten. Die PC-Schnittstelle ist auch bereits implementiert. In diesem Modus arbeitet der Cube als eine Art „Slave“ und stellt nur vom Master (PC) über RS232 gesendete Frames dar. Der SD-Card Modus ist bisher noch nicht implementiert, doch die Hardware-Schnittstelle über das SPI ist fertig implementiert. Vielleicht werde ich in dieser Richtung auch noch aktiv, doch die bisher funktionierenden Animationen sind schon sehr zufriedenstellend.

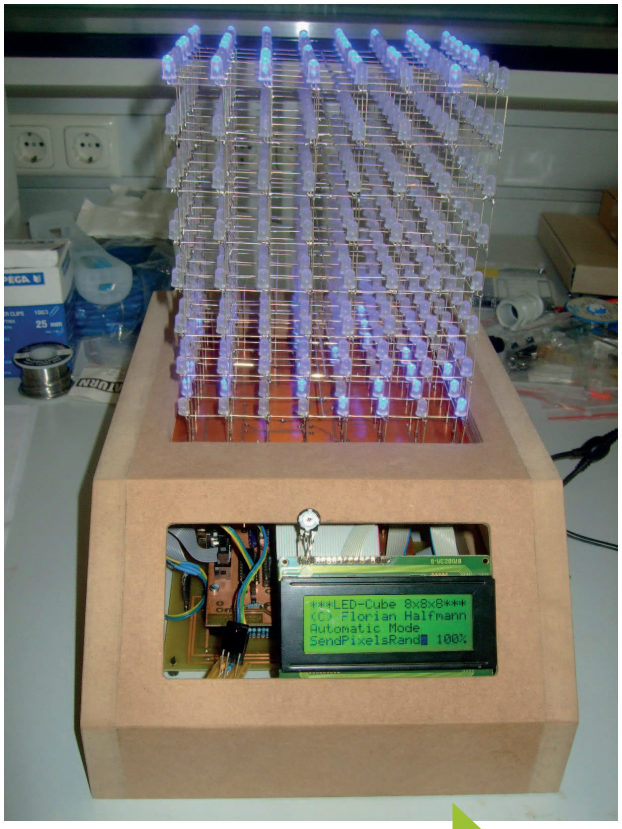


Abb. 6: LED-Cube mit Sockel

Der Sockel

Damit das ganze zu einem Schmuckstück wird bzw die Platinen und der Cube nicht einfach lose auf dem Tisch herumliegen, wird ein Sockel aus MDF-Holz gebaut. In diesen wird eine Aluplatte mit Display und ein paar Bedienelementen eingesetzt.

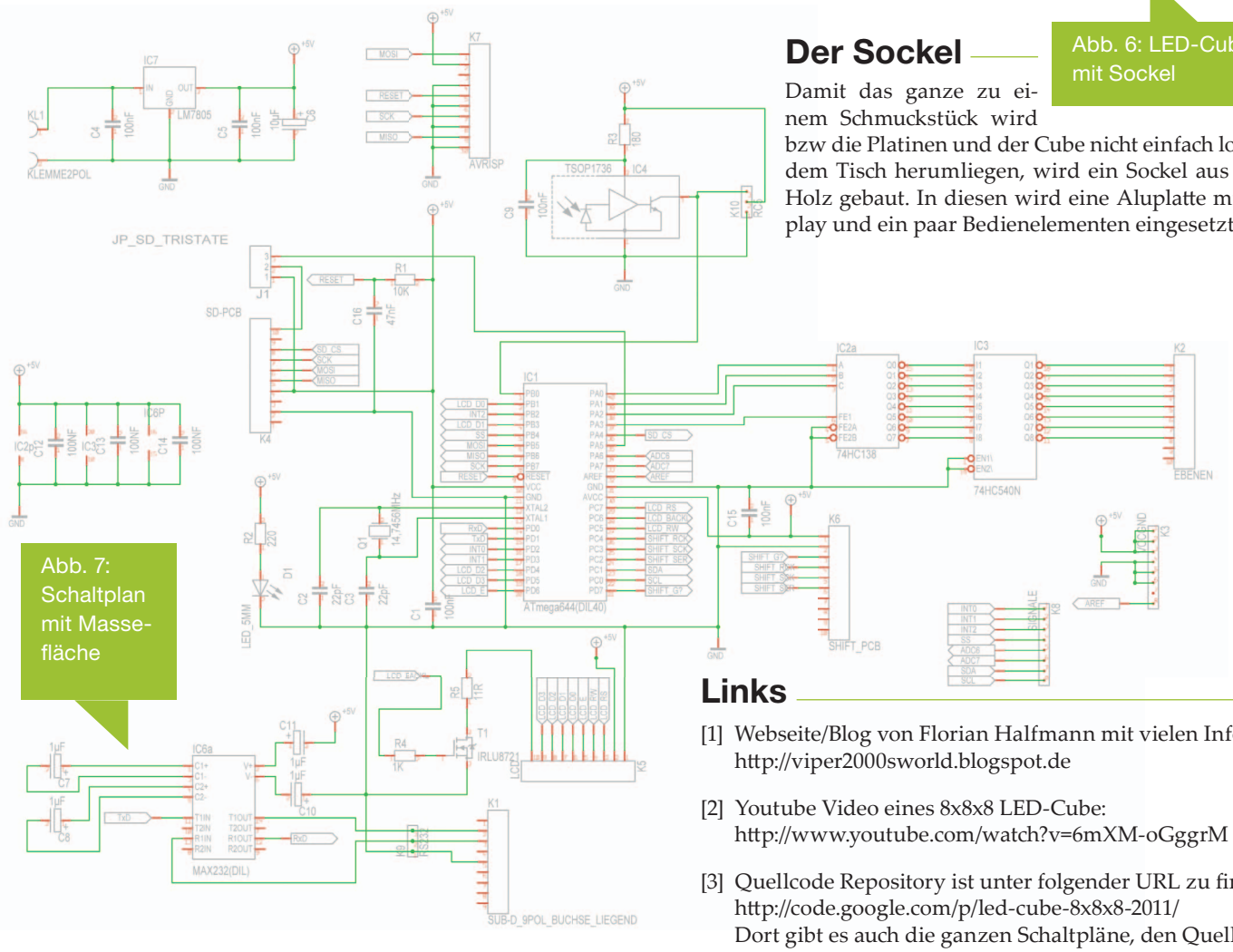


Abb. 7: Schaltplan mit Masse-fläche

Links

[1] Webseite/Blog von Florian Halfmann mit vielen Infos: http://viper2000sworld.blogspot.de

[2] Youtube Video eines 8x8x8 LED-Cube: http://www.youtube.com/watch?v=6mXM-oGggrM

[3] Quellcode Repository ist unter folgender URL zu finden: http://code.google.com/p/led-cube-8x8x8-2011/ Dort gibt es auch die ganzen Schaltpläne, den Quellcode für den AVR usw.

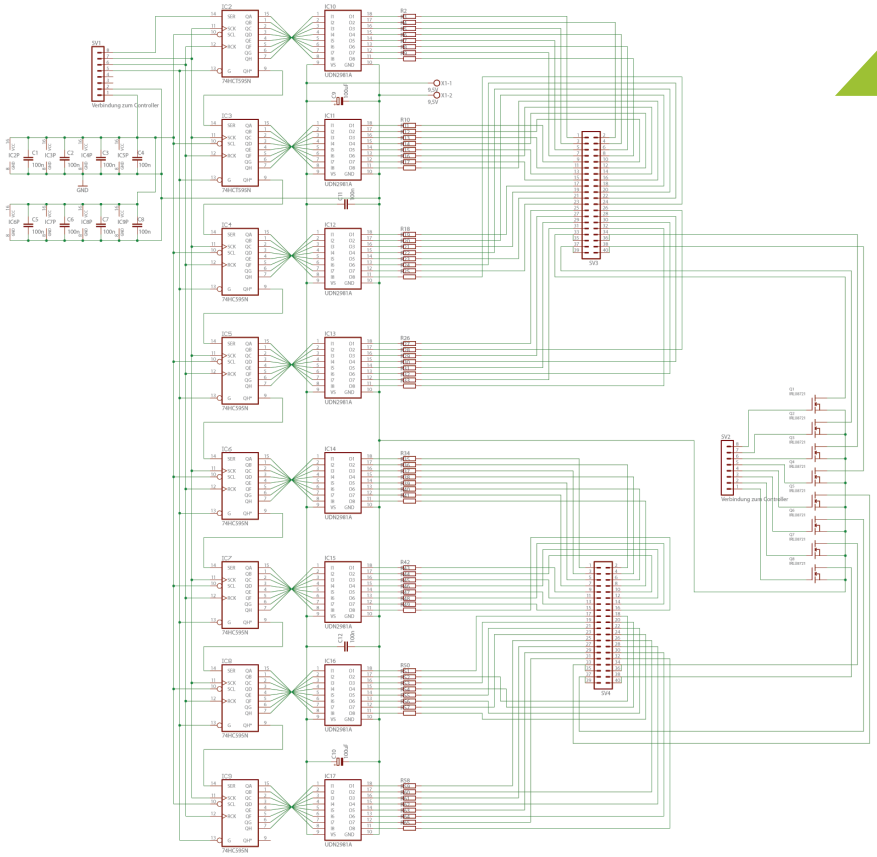


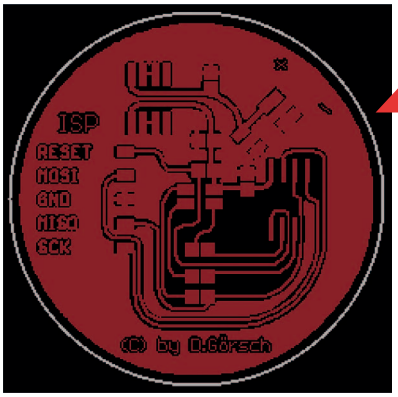
Abb. 6: Schaltplan Leistungsteil. Die Quelltexte sind vom Autor zu erhalten.

Hinweis: Aus Platzgründen konnte leider nicht alles in Originalgröße abgedruckt werden. Deshalb sei an dieser Stelle auf die PDF-Version verwiesen, hier kann man sich die Pläne in der Vergrößerung nochmals ansehen.

HowTo: Erstellung eines Nutzen in EAGLE

D. Görsch <mail@dgoersch.info>

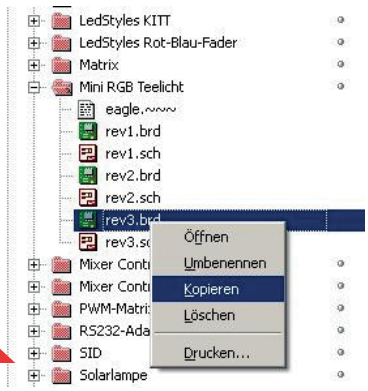
Hin und wieder taucht die Frage auf, wie man in EAGLE einen s.g. Nutzen erstellt. Da ich mich vor Kurzem selber erst darüber informiert hatte, fasse ich die nötigen Schritte in ein kurzes HowTo zusammen:



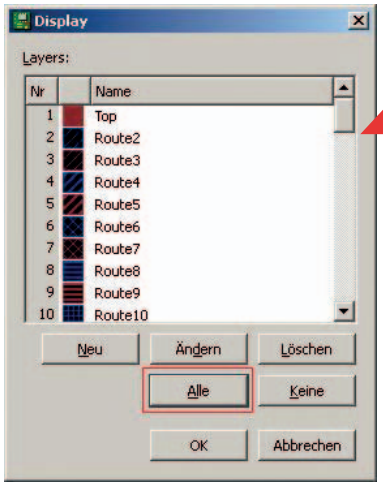
Als Vorlage dient hier eine kleine Platine von mir, für ein RGB-Licht als Teelicht-Replacement

Schritt 1:

Als erstes erstellen wir eine Kopie der .brd-Datei

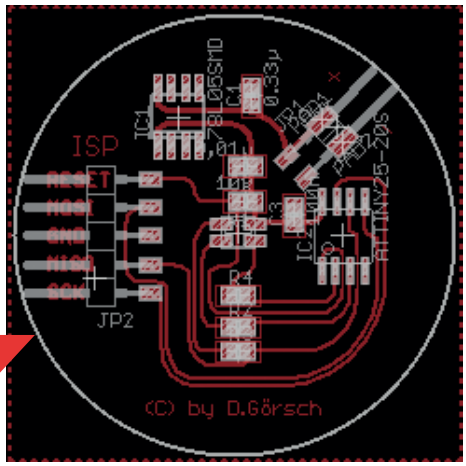


Schritt 2:



Nach dem Öffnen der Kopie aktivieren wir alle Layer und Ansichten

Die Masseflächen lassen wir nicht freirechnen (nicht RATSNEST ausführen!), die Ansicht sollte dann in etwa so aussehen



Schritt 3:

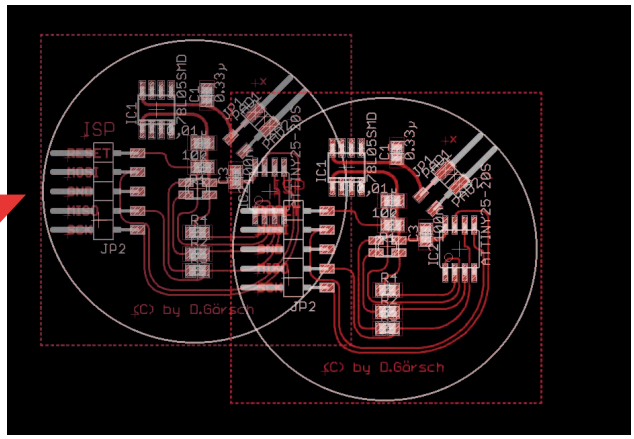
Um alle Bauteile, Leiterbahnen, Pads, etc. auszuwählen, geben wir GROUP ALL ein

Schritt 4:

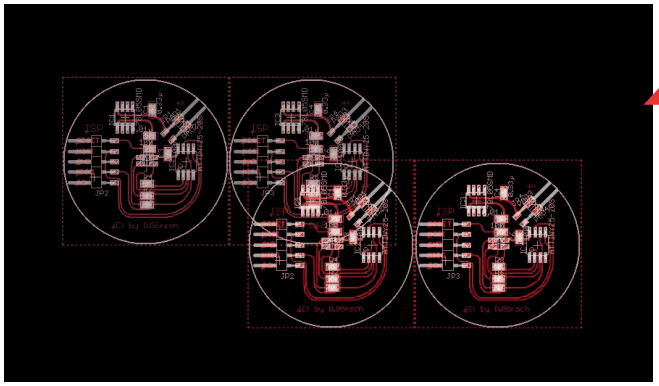
Danach erstellen wir mit dem Befehl CUT und einem Klick irgendwo in den markierten Bereich eine Kopie in der Zwischenablage

Schritt 5:

Tippen wir nun PASTE haben wir diese Kopie frei schwebend unter dem Mauszeiger und können sie frei Platzieren



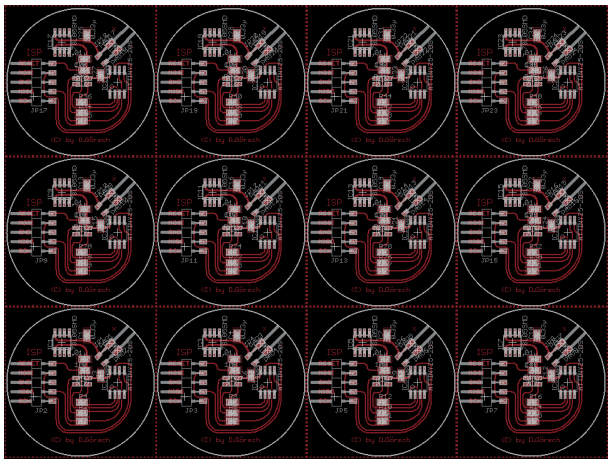
Schritt 6:



Nachdem wir die Kopie passend platziert haben, führen wir die Schritte 3-5 erneut aus und Platzieren die Kopie wieder passend

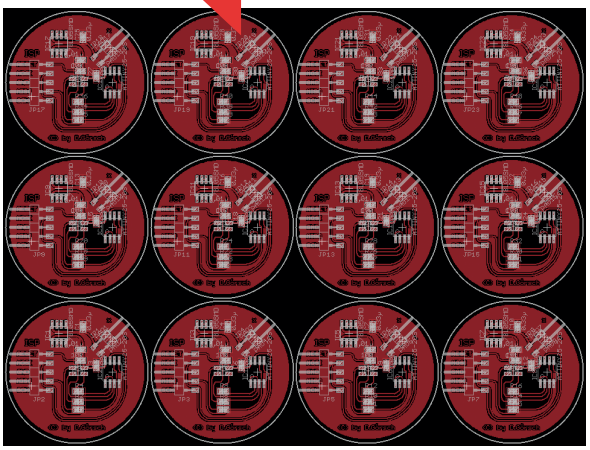
Schritt 7:

Das Spiel Markieren – Kopieren – Einfügen wiederholen wir sooft bis das gewünschte Nutzen erstellt ist



Schritt 8:

Zu guter Letzt lassen wir freirechnen



Anmerkungen

- Ich bin kein Freund von unnötiger Mausschubserei, daher tippe ich die Befehle direkt ein. Es sollte sich aber nichts Grundlegendes bei Mausbenutzung ändern.
- Ich übernehme keine Garantie auf Vollständigkeit und Korrektheit der Anleitung. Jedoch habe ich die Schritte genau so während des Erstellens des HowTo ausgeführt.
- Fragen und Anregungen nehme ich gerne per eMail an mail@dgoersch.info entgegen. Ich biete aber keinen Support für EAGLE.

Ein 8bit-Rechner

auf dem Spartan-3A-Starterkit

Josef Gnadl <gnadlregensburg@kabelmail.de>

In diesem Artikel wird das Projekt eines Bastlers vorgestellt. Ausführliche und exakte Informationen sowie Downloads sind zu finden unter http://www.bomerenzprojekt.de

Die CPU

Die CPU ist vielleicht am ehesten vergleichbar mit historischen Prozessoren wie 6502 oder 8085. Sie hat einen vollständigen Befehlssatz mit 256 OpCodes. Die Befehle sind 1 oder 2 Byte lang.

Die CPU (Abb. 1) unterstützt die Adressierung mehrer 64KByte-Seiten. Dies geschieht so: Jede ausgegebene Adresse kommt von einem der Adressregister P,X,Y,Z. Auf den Steuerleitungen wird angezeigt, von welchem dieser Register die Adresse kommt. Dadurch hat eine externe MMU die Möglichkeit, jedem der Adressregister eine andere Speicherseite zuzuordnen. Die CPU kann ausserdem spezielle Steuersignale ausgeben, welche die MMU zu einer Seitenumschaltung veranlassen sollen. Alle Speicherzugriffe sind 8bit-Zugriffe. 16bit-Zugriffe können daraus aufgebaut werden nach dem Muster: GTMX => IXE => GTMX. GTMX lädt (X) in den Akku A, IXE inkrementiert X und tauscht die Inhalte des Akku und des Erweiterungs-Registers B, GTMX lädt das zweite Byte in den Akku. (Es gibt hierfür Assembler-Makros.) X,Y,Z sind gleichberechtigt. Dekrementieren/Inkrementieren im Bereich -8..+8 geht schnell. Einen Stackpointer gibt es nicht. Für lange Sprünge lädt man das Sprungziel absolut oder relativ nach AB. Dies erfolgt byteweise durch zwei Befehle. (Auch dazu gibt es Assembler-Makros.) Der Sprungbefehl lädt AB nach P und rettet die Rückkehradresse P+1 nach AB und gleichzeitig in ein Register Q. Er wird verwendet für Sprünge, Unterprogrammaufrufe und Rückkehr aus Unterprogrammen.

Abb. 1: CPU

Verzweigungs-Befehle arbeiten mit 8bit-langer Distanz. Es wird zwischen Vorwärts- und Rückwärts-Sprüngen unterschieden. Für schnelle Schleifen gibt es die Schleifen-Startadresse R. Repeat-Befehle gibt es kombiniert mit Zähler-Dekrementieren und Adresse Inkrementieren/Dekrementieren.

Der Befehl H ist der einzige und universell einsetzbare I/O-Befehl. Es wird AB auf dem Adressbus ausgegeben und der Datenbus nach A eingelesen. Wenn auf H ein Sprungbefehl folgt, wirkt H stattdessen als Präfix und bewirkt, dass der Sprung mit einer Memory-Page-Umschaltung kombiniert wird.

Einziges Flag ist das Carry-Flag. Anstelle eines Zero-Flags kann der aktuelle Zustand zB. des Akku abgefragt werden, ebenso dessen höchstes Bit. Zweierkomplement-Arithmetik ist trotz des fehlenden Overflow-Flags möglich. Man geht dazu den Umweg über die Umwandlung von Zweierkomplement-Zahlen in Offset-Zahlen durch Negation des Vorzeichenbits.

Der Zeitablauf ist eine Folge von Vollzyklen, welche ihrerseits aus den Halbzyklen tA und tB bestehen. Speicherzugriffe erfolgen während tA. Es gibt einen Repeat-Eingang zum Wiederholen eines Halbzyklus. Lese-Vorgänge werden dabei wiederholt. Alle Operationen dauern 1, 2 oder 3 Vollzyklen. Letzteres gilt für Verzeigungen, wenn der Sprung ausgeführt wird. Ausnahme ist ein spezieller langer No-Op-Befehl.

Die Startadresse wird bei RESET vom Adressbus eingelesen. Die CPU hat keine Interrupts. Für Anwendungen, welche zwingend Interrupts erfordern, bräuchte man eine zweite CPU, welche die erste mittels Repeat-Signal anhält und die Interrupt-Routine ausführt. Denkbar wäre auch eine geänderte Version der CPU mit einem zweiten Registersatz.

Das Gesamtsystem

Das Gesamtsystem besteht aus den 64KByte-Seiten ROM/Video-RAM, Haupt-RAM, Zusatz-RAM, aktiver Steckplatz. Dazu kommt noch die Tastatur-Schnittstelle und ein spezieller Taster (DG-Taster). Es gibt 8 Steckplätze. Der aktive Steckplatz wird durch einen 3bit-Speicher ausgewählt. Die Steckkarten sollen aus Hardware (externe Schnittstellen, ...) und mitgebrachter ROM-Software (Treiber, ...) bestehen. Wegen des CPU-Timings mit inaktivem tB-Halbzyklus ist es denkbar, auf den Steckkarten eigene CPUs mit 180 Grad Zeitversatz laufen zu lassen.

Repeat-Halbzyklen werden stets in gerader Anzahl eingefügt. Das Display ist 512 Pixel breit und 256 Pixel hoch. Die Pixel sind zweiwertig. Das Schreiben des Bitmusters der ausgegebenen Zeichen ins Video-RAM erfolgt per Software durch die CPU. Das Auslesen des Video-RAM zum Display erfolgt während der tB-Phasen und stört nicht den Lauf der CPU. Alternativ kann als Video-RAM ein Dual-Port-RAM verwendet werden. Eine Besonderheit ist der Zeichensatz. Er beinhaltet einen hexadezimalen Ziffernsatz, bei welchem für die Ziffern A..F spezielle Zeichen verwendet werden.

Abb. 3: CPU-Timing

Die Software

Die Tastatur-Display-Routinen verwenden die Text-Bereiche C für Kommando-Eingabe, D für tabellarische Ein/Ausgabe, E für Text-Bearbeitung. Der Aufruf der Routinen erfolgt im Bereich C oder D. Der Benutzer kann dann innerhalb dieser Routinen in den Bereich E wechseln. Die Kommando-Eingabe erfolgt nur dann über die Tastatur, wenn ein spezieller 16 Zeilen großer FIFO-Speicher leer ist. Andernfalls wird das Kommando aus diesem Speicher ausgelesen. So können Abläufe automatisiert werden.

Die ROM-Software enthält eine Compiler-Sprache und einen Assembler. Der Compiler erzeugt einen Zwischencode, welcher interpretativ abgearbeitet wird. Der Zwischencode enthält veränderliche Zeiger und ist nur im RAM lauffähig.

Die Compilersprache besteht nur aus Struktur-Elementen und wenigen einfachen Routinen wie Inkrementieren/Dekrementieren. Die übrigen Routinen werden von den Steckkarten geliefert. Vom Anwender erstellte Assembler-Routinen werden über eine sauber definierte Schnittstelle eingebunden. Mit Hilfe eines Datentyps Data-Box lassen sich Arrays und Strukturen (Records) realisieren. Programme können Umgebungs-Variable acquirieren und dann wie gewöhnliche Variable verwenden. Programme können so geschrieben werden, dass die Zahl der Taktzyklen für alle Teile exakt berechenbar ist, sofern die von den Steckkarten gelieferten Routinen dafür ausgelegt sind.

Die Software beinhaltet ein Sicherheits-Konzept, wonach bei fehlerhafter Programmierung auf Hochsprachen-Ebene oder bei Fehlbedienung eine versehentliche Änderung von Zeigern oder geschützten Daten wie zB. Programmcodes ausgeschlossen sein sollte. Dies setzt voraus, dass alle Maschinenroutinen nur zulässige Aktionen ausführen. Das Konzept wird durchbrochen, wenn man selbst-erstellten Maschinencode als ausführbar kennzeichnet. Als Sicherheit wird deshalb dabei der DG-Taster abgefragt.

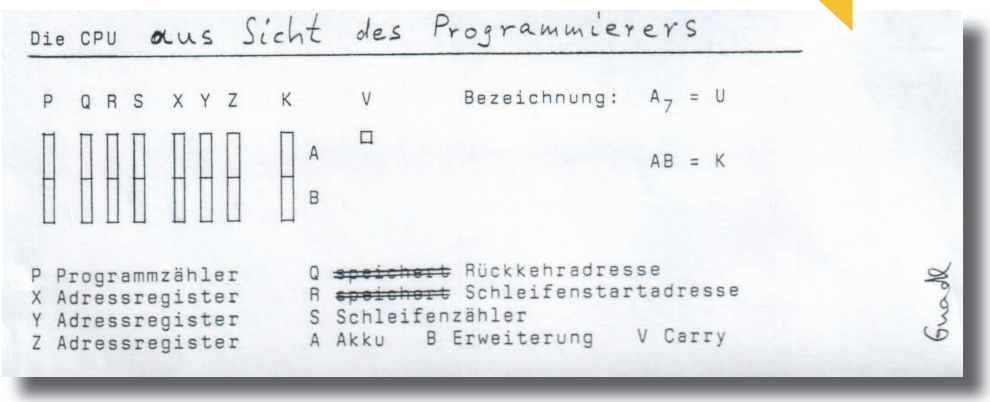
Die Realisierung

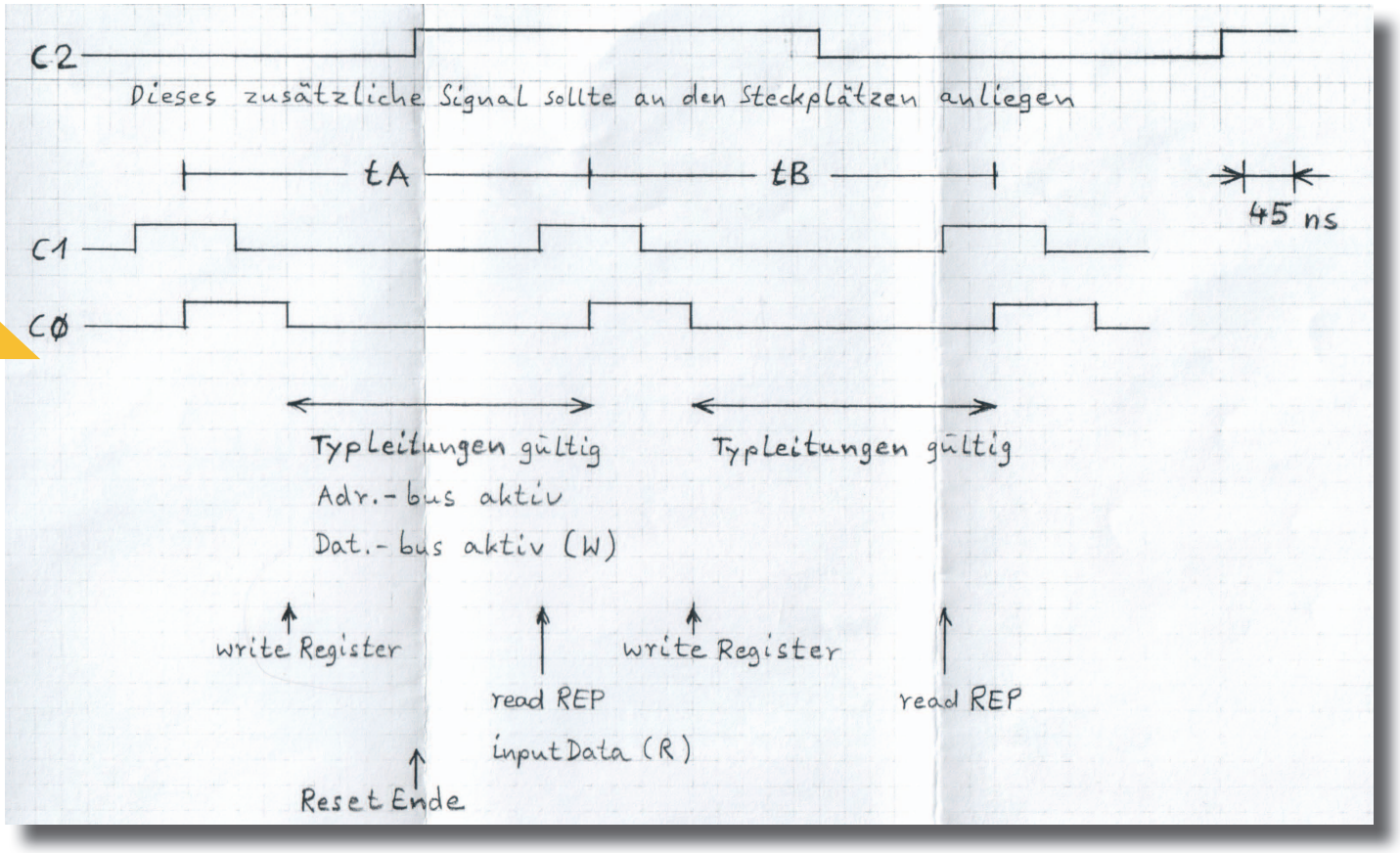
Eine besondere Schwierigkeit ist die Verwendung des DDR2-RAM. Wegen des diffizilen Timings verwendet man normalerweise das Xilinx-Werkzeug MIG. Die zu realisierende Architektur kann mit Burst-Zugriffen nichts anfangen. Die vorliegende Konfiguration umgeht dies, indem die vier Datenworte eines Bursts als ein einziges Datenwort behandelt werden. Beim Schreiben erhalten alle vier Worte denselben Wert. Das Lesen erfolgt irgendwo im Inneren des Bursts. Es entfällt die Verwendung von DDR-Flipflops, ebenso die Schwierigkeit, beim Lesen in jedem der vier Datenworte zeitlich die Mitte zu treffen. Ein dabei theoretisch denkbares Glitch-Problem ist bisher nicht aufgetreten. Ebenfalls schwierig ist das Schalten des 133MHz-Dataclock. Der Refresh erfolgt während der tB-Phasen ohne Störung des streng berechenbaren Zeitverhaltens der CPU.

Die Konfiguration wurde mittels der ISE-Version 11.1 realisiert. Wegen des DDR2-RAM ist es denkbar, dass bei anderen Versionen Probleme auftreten, welche aber durch geeignete Einstellungen bei der Synthese lösbar sein sollten. Die gesamte Konfiguration kommt ohne vorgefertigte Module von Xilinx aus. Es wird nur die kostenlose ISE-WebPack-Lizenz gebraucht. Mit Ausnahme des Block-RAM lastet die Konfiguration das FPGA zu etwa 20 Prozent aus. Von der auf dem Spartan-3A-Starterkit angebotenen Peripherie werden verwendet der VGA-Ausgang für den Monitor, die PS/2- Schnittstelle für die Tastatur, die RS232-Schnittstelle für Kommunikation mit PC, eine Buchsenleiste zum Anschluss einer SD-Karte, außerdem das LCD-Display zur taktgenauen Anzeige von Programmlaufzeiten.

Das auf dem Board vorhandene Parallel-Flash kann zum Sichern einer Steckkarten-Software verwendet werden, muss aber nicht. Es ist möglich, die Konfiguration ohne bleibende Veränderung auf dem Board zu testen. Die CPU ist ein sauber abgegrenzter Teil der Konfiguration und sollte unverändert auch für andere Designs übernommen werden können. Sie war ürsprünglich für eine Realisierung mit Latches konzipiert. Es gibt deshalb zwei Taktsignale. Kein Register der CPU wird gleichzeitig geschrieben und gelesen. Ein Vollzyklus, bestehend aus tA und tB, dauert 720 ns.

Abb. 2: Die CPU aus Sicht des Programmierers



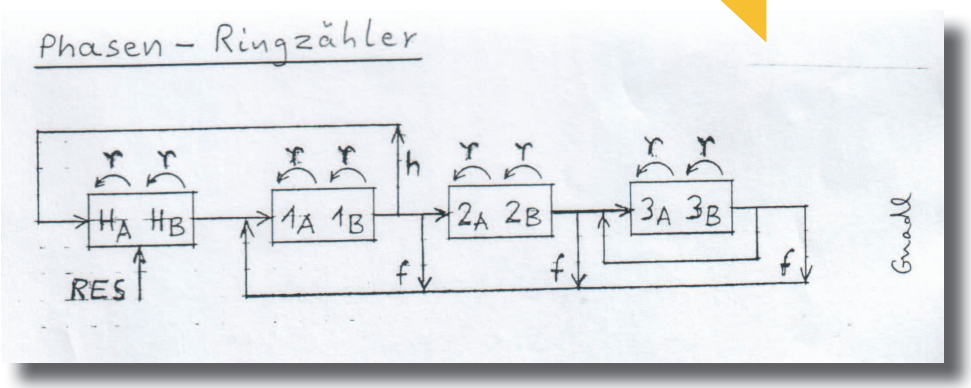


Hilfs-Programme für PC

Zur Veranschaulichung der internen Arbeitsweise der CPU gibt es ein C-Programm, welches für jede Taktphase innerhalb eines Befehlszyklus die aktiven Steuersignale in der CPU anzeigt. Damit kann man die im VHDL-Code der CPU enthaltenen Logik-Netze zur Bildung der Steuersignale manuell überprüfen.

Zum Testen von Software gibt es ein Emulationsprogramm für Linux-PC, welches den Maschinencode der CPU interpretativ abarbeitet. Wer kein Spartan-3A-Starterkit hat, kann sich so einen Eindruck vom Look&Feel des Systems verschaffen.

Abb. 4: Phasen-Ringzähler



Schlussworte

Das System ist konzipiert als eigenständig arbeitender Rechner nach Art der historischen Heimcomputer wie AppleII (Steckplatzkonzept) oder C64. Die CPU ist nicht gedacht als speziell für FPGAs optimierter Softcore, sondern um vielleicht einmal als Einzel-Baustein erhältlich zu sein. Die Realisierung auf dem Spartan-3A-Starterkit sollte ursprünglich nur dazu dienen, den VHDL-Code der CPU zu verifizieren. Es ist nun aber ein eigenes Gerät für Hobbyisten daraus geworden.

Vielleicht findet sich jemand, der das Gerät als speziell gefertigte Hardware realisiert, bei welcher die Steckplätze auch physikalisch Steckplätze sind. Erforderlich wäre noch die Entwicklung einer Mindest-Ausstattung an Steckkarten. Als CPU könnte man vielleicht ein Antifuse-FPGA verwenden.

Rückmeldungen an den Autor wären willkommen, Kontaktdaten finden sich auf der Projekt-Website. Hingewiesen sei auch auf das dort verlinkte Forum.

Links

http://www.bomerenzprojekt.de

Funkübertragung mit IEEE 802.15.4 Transceivern - nicht nur für Profis

Daniel Thiele, Axel Wachtler

Im Artikel wird ein Überblick über die IEEE-802.15.4 Funktechnik gegeben und die derzeit verfügbaren Atmel-Transceiver werden vorgestellt. In einem Beispielprojekt wird mit Hilfe von zwei Radio-Boards und der µracoli-Library gezeigt, wie ein einfacher Temperatur-Webserver in Python programmiert wird.

Einleitung

Der Funkstandard IEEE-802.15.4 „ieee154“ [6] wurde speziell für batteriebetriebene Sensoranwendungen mit niedrigen Datenraten im Bereich von 20 bis 250 kbit/s spezifiziert, wodurch mit diesen Funkknoten eine Batterielebensdauer von mehreren Jahren erreicht werden kann [3]. Im Standard sind die physikalische Datenübertragung (PHY-Layer) und der Zugriff auf den Funkkanal (MAC-Layer) beschrieben. Das Zigbee-Protokoll setzt auf dem 802.15.4-MAC-Layer auf und erlaubt es, Daten über mehrere Knoten zu routen. Charakteristisch für IEEE-802.15.4 basierte Anwendungen ist, dass es einen zentralen Koordinator-Knoten gibt, an dem sich die Sensor-Knoten anmelden.

Bereits an dieser Stelle fragt man sich, ob der Einarbeitungsaufwand in diese Funktechnologie für den Laien nicht zu aufwändig ist und das Projekt für eine einfache drahtlose Sensoranwendung nicht doch besser mit RFM12-Bauteilen aufgebaut wird. In den folgenden Abschnitten wird beschrieben, wie mit geringem Aufwand, basierend auf der µracoli-Library, ein einfacher Temperaturserver mit IEEE-802.15.4-Modulen realisiert werden kann.

Transceiver

Die Transceiver der Firma Atmel zeichnen sich durch geringen Stromverbrauch und ein großes Linkbudget und damit eine große Reichweite aus (siehe Tabelle 1). Sie bieten eine einfache und flexible Programmierschnittstelle auf Registerebene und implementieren Teile des IEEE-802.15.4-Standards bereits in Hardware. Derzeit sind Transceiver für den Frequenzbereich 700/800/900 MHz und für das 2,4 GHz Band verfügbar. Eine Sonderstellung nimmt der IC ATmega128RFA1 ein, bei dem Mikrocontroller und Transceiver in einem Bauteil integriert sind.

IC	Frequenzband	Interface	Sleep-Strom	RX-Strom	TX-Strom
AT86RF212	700/800/900 MHz	SPI	200 nA	9,0 mA	17,0 mA
AT86RF230	2,4 GHz	SPI	20 nA	15,5 mA	16,5 mA
AT86RF231	2,4 GHz	SPI	20 nA	12,3 mA	14,0 mA
AT86RF232	2,4 GHz	SPI	400 nA	11,8 mA	13,8 mA
AT86RF233	2,4 GHz	SPI	200 nA	11,8 mA	13,8 mA
ATmega128RFA1	2,4 GHz	MCU+TRX	250 nA	16,6 mA	18,6 mA

Tabelle 1: Transciever der Firma Atmel

Die Transceiver-Bausteine sind SMD-Bauteile mit geringen Pinabständen, so dass das Selbstlöten in aller Regel ausscheidet. Zusätzlich wird einige Erfahrung beim UHF-Design der Leiterplatten vorausgesetzt. Für ein Hobbyprojekt greift man deshalb besser auf fertige Module zurück. Dass die Eigenentwicklung einer Leiterplatte für diese Transceiver in der Hobbywerkstatt dennoch möglich ist, wird in [4] und [11] gezeigt.

Module

Für die Integration von IEEE-802.15.4-Funk in eine eigene Schaltung bieten sich komplett bestückte und zertifizierte Module an. Auf diesen Modulen befinden sich der Transceiver und ein Mikrocontroller. Ein sehr populärer Vertreter ist das Zigbit-Modul von Atmel (ehemals Meshnetics). Eine ausführliche Analyse dieses Moduls findet man in [12]. In nachfolgender Tabelle (Tabelle 2) sind Module weiterer Hersteller aufgeführt:

Modul	Hersteller	Transceiver	MCU
ANY2400SC-3	AN-Solutions	ATmega128RF1	ATmega128RF1
ZigBit ATZB-A24-UFL/U0	Atmel	AT86RF230	ATmega1281
ZigBit ATZB-900-B0	Atmel	AT86RF212	ATmega1281
deRFmega128	dresden elektronik	ATmega128RFA1	ATmega128RFA1
WiMOD im240a	IMST GmbH	AT86RF231	ATmega328
IC-RadioModul	In-Circuit GmbH	AT86RF230	ATmega1281

Tabelle 2: Module verschiedener Hersteller

Radiofaro Board

Das Radiofaro-Board [9] entstand aus der Idee heraus, das Arduino-Diecimila-Design [1] mit einen modernen Prozessor mit Funkschnittstelle auszustatten. Um die Notwendigkeit einer kostenintensiven Funkzulassung zu umgehen, fiel die Wahl auf das damals verfügbare Modul derfmega128 [5] der Firma dresden elektronik ingenieurtechnik gmbh. Das Modul ist FCC und ETSI zertifiziert und kann daher ohne Bedenken in eigene Schaltungen integriert werden.

Warum wurde das derfmega128 und nicht das Zigbit-Modul verwendet? Gegenüber einem ATmega1281 mit 8KByte SRAM verfügt der ATmega128RFA1 über die doppelte Menge SRAM. Weiterhin erfolgt der Zugriff auf den Transceiver beim ATmega128RFA1 direkt über interne Register anstatt über einen SPI-Link wie beim ZigBit Modul. Außerdem wurde ein Transceiver der zweiten Generation integriert, der zusätzliche Möglichkeiten bietet, u.a. ist ein asynchroner MAC-Symbol-Counter verfügbar, mit dem sehr lange Sleep-Intervalle realisiert werden können. Neben dem für ATmega-Controller untypischen AES-Block kommt im Beispielprojekt der integrierte Temperatursensor zur Anwendung.

Das Radiofaro-Board (siehe Abb. 1) sieht auf den ersten Blick zwar aus wie ein Arduino-Board, ist aber leider nur mechanisch mit seinem italienischen Verwandten kompatibel. Das liegt u.a. daran, dass der ATmega128RFA1 nur Spannungen bis maximal 3.8V an den Port-Pins erlaubt. Die Port-Pins wurden daher durch Vorwiderstände geschützt. Allerdings kann man die Funktion mancher Shields damit nicht sicherstellen. Eine weitere Inkompatibilität betrifft den „falsch platzierten“ ISP Header, der bei allen offiziellen Arduino-Boards an der Vorderseite, gegenüber der USB-Buchse zu finden ist und den manche Shields verwenden. Dieser digitale Anschluss würde sich aber im Antennenbereich des PCB befinden und hätte somit Einfluss auf die HF-Eigenschaften.

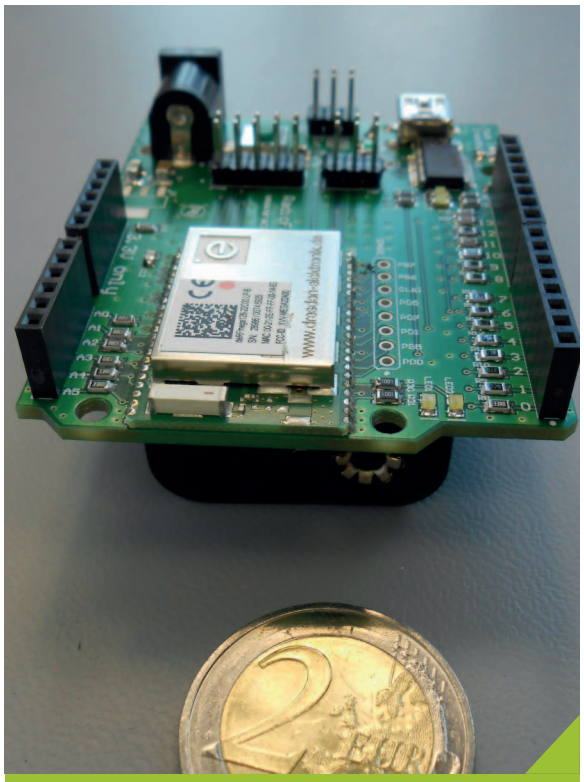


Abb. 1: Radiofaro Board

Es gibt aber auch eine ganze Reihe von Vorteilen:

- einen JTAG-Connector, mit dem man komplexere Software debuggen kann
- ein Batterie-Halter für 2 AA-Batterien, damit das Board auch längere Zeit autark arbeiten kann
- die Möglichkeit der Auswahl zwischen drei Versorgungsspannung durch einen Jumper (DC-Buchse, USB und Batterie)
- 2 Status LEDs befinden sich auf dem Board
- ungenutzte IO-Pins des Mikrocontrollers sind auf einer extra Stiftleiste herausgeführt

Software

a. AVR-Toolchain

Das eigentliche Ziel des Radiofaro-Projektes ist es, die Firmware mit der Arduino-IDE zu programmieren, um Einsteigern die ersten Schritte im Bereich der Funktechnik zu erleichtern. Leider ist in den aktuellen IDE-Paketen eine Toolchain vom Jahr 2008 enthalten, in der der ATmega128RFA1 noch nicht unterstützt wird. In einem der nächsten Artikel wird daher das Update der Arduino-IDE beschrieben. Für das nachfolgend vorgestellte Beispielprojekt wird eine reguläre AVR-Toolchain, bestehend aus

avr-gcc, avr-binutils, avr-libc und avrdude (alternativ auch Atmel Studio unter Windows) und die µracoli-Library verwendet.

Die Installation der AVR-Toolchain unter den verschiedenen Betriebssystemen würde allein schon einen eigenen Artikel füllen,daher sei an dieser Stelle auf folgende Quellen verwiesen:

- http://www.mikrocontroller.net/articles/AVR-GCC
- http://www.mikrocontroller.net/articles/AVR_und_Linux

Eine weitere Kurzanleitung für Windows und Linux in englischer Sprache findet man bei avrtools [2].

b. Python

Für die Implementierung des Webservers wird Python2.x [8] und das Modul pyserial [7] verwendet. Unter Windows muss zusätzlich noch das Modul win32com installiert werden.

c. Radio-Library

Die µracoli-Library beinhaltet Funktionen zur Ansteuerung des Transceivers sowie verschiedene Hilfsfunktionen zur Timer-, UART-, LED- und Tasteransteuerung. Da es sich um eine Bibliothek handelt, kann man die zur Verfügung gestellten Hilfsfunktionen verwenden, muss es aber nicht tun. Es werden derzeit ca. 80 verschiedene Boards und Module von der Library unterstützt. Für das Beispielprojekt wird das aktuelle Paket uracoli-src-0.3.0.zip von http://uracoli.nongnu.org/download.html verwendet und in einem Arbeitsverzeichnis entpackt. Man erhält damit die folgende Verzeichnisstruktur:

```
uracoli-src-rel _ 0.3.0/
|-- bin ................ vorkompilierte Images
|-- doc ................ Dokumentation als Text und PDF
|-- inc ................ Include Dateien
|-- lib ................ vorkompilierte Bibliotheken
|-- sniffer ............ Quellcode der Sniffer-Firmware
|-- src ................ Quellcode der Library
|   |-- libioutil
|   |-- libradio
|-- wibo ............... Quellcode Wireless Bootloader
|-- wuart .............. Quellcode Wireless UART
`-- xmpl ............... Beispiele zur Verwendung der
Library
```

Das Projekt

Es soll ein einfacher Temperatur-Webserver realisiert werden. Als Hardware werden zwei oder mehr Transceiver-Boards benötigt, ein Gateway-Board und ein oder mehrere Sensor-Boards. Das Gateway-Board wird über eine serielle- oder USB-Schnittstelle mit dem PC verbunden. Als Sensor-Knoten kommen Radiofaro-Boards oder andere Boards mit ATmega128RFA1 zum Einsatz, da der integrierte Temperatursensor verwendet werden soll. Als Sensor-Knoten kann aber prinzipiell jedes von µracoli unterstützte Board verwendet werden, wenn es mit einem Temperatursensor bestückt ist oder nachgerüstet wird.

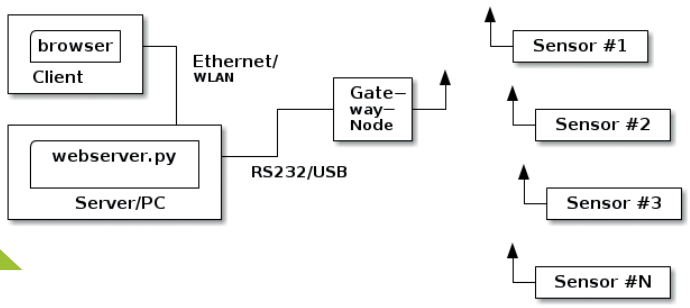


Abb. 2: Softwareübersicht

a. Gateway-Knoten

Auf dem Gateway-Board wird die Wireless-UART Firmware (WUART) installiert. Der Transceiver des Gateways ist ständig auf Empfang (Zustand RX_ON). Da das Gateway-Board über USB mit Strom versorgt wird, ist der permanente Stromverbrauch von ca. 14mA akzeptabel.

Die Firmware wird wie folgt gebaut und installiert:

```
cd uracoli-src-0.3.0
make -C src radiofaro              # compilieren von lib/liburacoli _ radiofaro.a
make -C wuart radiofaro            # compilieren von bin/wuart _ radiofaro.hex
avrdude -P usb -c dragon _ jtag -p m128rfa1 -U bin/wuart _ radiofaro.hex
```

Eine Liste von Boards, die als Gateway-Knoten eingesetzt werden können, erhält man mit dem Befehl:

```
make -C wuart list
```

Die Firmware für ein alternatives Gateway-Board wird entsprechend dem obigen Beispiel gebaut und auf dem Mikrocontroller installiert.

b. Sensor-Knoten

Auf den Sensor-Boards läuft eine spezielle Firmware zur Temperaturerfassung und Messwertübertragung. Da die Sensor-Knoten autark arbeiten und von einer Batterie versorgt werden, muss in der Firmware Powermanagement implementiert sein um eine lange Laufzeit zu erreichen. In [clt2011] werden verschiedene Temperatur- und Feuchtesensoren vorgestellt und miteinander verglichen. Der Source-Code zur Ansteuerung dieser Sensoren befindet sich im Paket uracoli-sensorlogger-1.0.zip.

Wenn mehrere Sensor-Knoten verwendet werden sollen, müssen diese durch Adressen unterschieden werden. Die Adressen können entweder im Flash-Speicher oder im EEPROM des Mikrocontrollers abgelegt werden. In der µracoli-Library gibt es Methoden um einen Konfigurationsdatensatz zu erzeugen und auszuwerten. Dieser Datensatz beinhaltet u.a. eine 16-Bit-Adresse und die 16-Bit-PAN-ID (Netzwerkadresse) des Knotens. Mit dem folgenden Befehl kann der Konfigurationsdatensatz erzeugt und in das EEPROM der MCU programmiert werden:

```
python nodeaddr.py -a <addr> -p <panid> -c <channel> -O 0 > addr.hex
avrdude -P usb -p m128rfa1 -c jtag2 -U ee:w:addr.hex:i
```

Im Beispiel-Projekt sendet der Sensor-Knoten die Messwerte als Textstrings, die in der Payload von IEEE-802.15.4-Datenrahmen eingebettet sind. Es wurde die redundantere ASCII-Übertragung gegenüber der einem effizienteren binären Format gewählt, da damit die WUART-Firmware im Gateway-Knoten eingesetzt werden konnte:

```
ADDR=0003, T=26.7 [degC], Vcore=3,25 [V]
ADDR=0004, T=18.2 [degC], Vcore=2,93 [V]
ADDR=0003, T=26.7 [degC], Vcore=3,25 [V]
ADDR=0004, T=18.4 [degC], Vcore=2,92 [V]
```

c. Server

Der Temperatur-Webserver ist in Python implementiert und damit unter Linux, Windows oder auch MacOS lauffähig. Ein Thread des Webservers liest die ASCII-Daten von der seriellen Schnittstelle und parst die Werte. Die Adressen werden auf Strings gemappt und die lokale Uhrzeit des Server-PCs wird hinzugefügt. Anschließend werden diese Daten als HTML-Tabelle aufbereitet und über einen HTTP-Socket an anfragende Web-Browser ausgeliefert. In [clt2011] wurde dieser einfache Webserver-Ansatz weiter ausgebaut, u.a. wurden die Daten in einer Round-Robin-Database [rrdtool] gespeichert und das Python-Script kann als Server-Dienst unter Linux vom Init-Prozeß gestartet werden.

Bilder

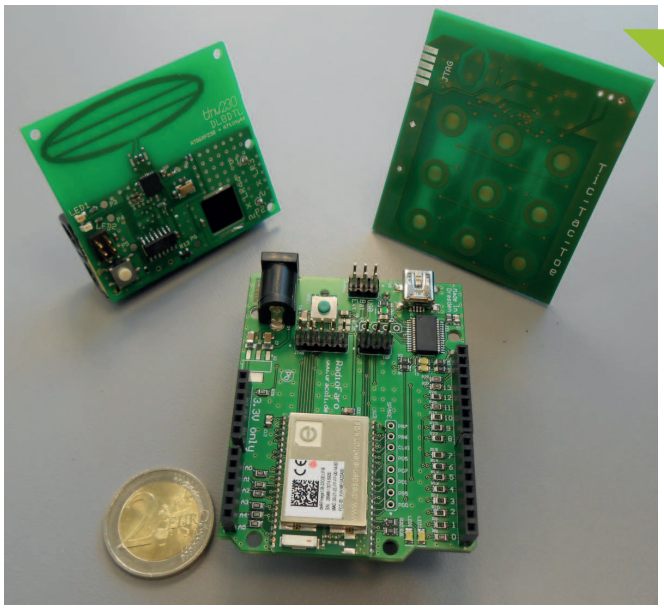


Abb. 3: DIY-Boards
links: tiny230, Joerg Wunsch ATtiny84 + AT86RF230/231
rechts: Tic-Tac-Toe, Joerg Wunsch ATmega128RFA1
mitte: Radiofaro, Daniel Thiele

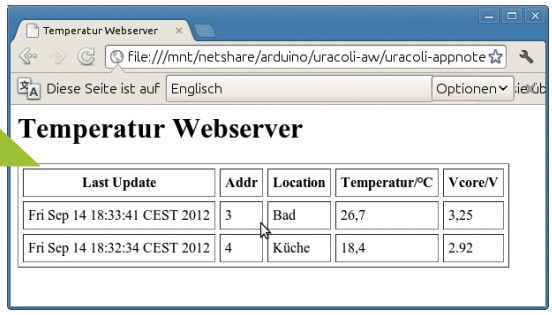


Abb. 4: Screenshot des Beispielprojektes

Ausblick

Das vorgestellte Beispiel-Projekt zeigt, wie mit geringem Aufwand eine einfache Sensor-Webanwendung realisiert werden kann. Durch den Einsatz von Funkmodulen reduziert sich der Aufwand bei der Hardware-Entwicklung. Das vorgestellte Radiofaro-Board wird in kommenden Beiträgen als IEEE-802.15.4-Sniffer verwendet und über die Arduino-IDE programmiert. Die Software zum Artikel findet man unter http://uracoli.nongnu.org/download.html in den Paketen uracoli-src-0.3.0.zip und uracoli-epj00-1.0.zip.

Literatur

- [1] Internet, Arduino Diecimila http://arduino.cc/en/Main/ArduinoBoardDiecimila
- [2] Internet, µracoli Documentation - AVR Tools, http://uracoli.nongnu.org/avrtools.html
- [3] Axel Wachtler, Jörg Wunsch, Matthias Vorwerk, Drahtlose Sensordatenerfassung und -verarbeitung mit Linux, http://uracoli.nongnu.org/clt2011/
- [4] Wunsch, Wachtler & Vorwerk. Tic Tac Toe Reloaded, 2012, http://uracoli.nongnu.org/clt2012/
- [5] dresden elektonik GmbH, Evaluations-Module deRFmega128 http://www.dresden-elektronik.de/funktechnik/products/radio-modules/eval-derfmega/
- [6] IEEE, IEEE 802.15 Standards, http://standards.ieee.org/about/get/802/802.15.html
- [7] Internet, pyserial Documentation, http://pyserial.sourceforge.net/
- [8] Internet, Python Programming Language - Official Website, http://www.python.org/
- [9] Daniel Thiele, Radiofaro, http://uracoli.nongnu.org/radiofaro.html
- [10] Tobias Oetiker, RRDtool, http://oss.oetiker.ch/rrdtool/
- [11] Jörg Wunsch, The tiny230 Radio Board, http://www.sax.de/~joerg/tiny230/
- [12] Wiki-Autorenkollektiv, Meshnetics Zigbee, http://www.mikrocontroller.net/articles/Meshnetics_Zigbee

□

Betreiben eines Beschleunigungssensors am Gnublin-Board

Markus Lösch, Benjamin Zimmermann

Dieser Artikel beschreibt den Einsatz eines „Sparkfun 9DOF Razor IMU“ an einem Gnublin. Das embedded-Board „Gnublin“, über das bereits in einigen vergangenen Ausgaben dieses Magazins berichtet wurde, stellte sich als sehr gutes Gegenstück zum 9DOF heraus. Beim Sparkfun 9DOF handelt es sich um einen vielseitig einsetzbaren Sensor mit eingebautem Beschleunigungssensor, Gyroskop und Magnetometer. Ziel des Projektes war es, die Daten des Sensors mit einem Gnublin auszulesen und anderen Diensten und Funktionen zur Verfügung zu stellen. Ein Dienst der diese Funktion nutzt und auf die geschaffene Schnittstelle zugreift wurde implementiert. Bei diesem Dienst handelt es sich um einen Webserver, welcher die Daten somit über HTTP für den User erreichbar macht. Ein weiteres Ziel des Projekts war es den Zugriff auf das Webinterface für den Benutzer möglichst einfach und konfigurationsfrei zu ermöglichen. So soll es ausreichen, wenn sich der Benutzer nur zu einem von Gnublin bereitgestellten WLAN-Netzwerk verbindet und einen Webbrowser öffnet um die Daten des Sensors angezeigt zu bekommen.

Folgende Schritte bzw. Bestandteile des Projekts waren nötig um die gesteckten Ziele zu erreichen

- Erfolgreiches verbinden des 9DOF an das Gnublin-Board
- einen Dämon, der kontinuierlich die Daten des Sensors ausliest und anderen Programmen zur Verfügung stellt
- ein CGI-Programm, welches die aktuellen Daten per Web abrufbar macht
- eine HTML-Seite die den Verlauf der Sensorwerte grafisch darstellt
- Eine Netzwerkkonfiguration die konfigurationsfreies verbinden mit dem Gnublin-Board ermöglicht und sofort das Webinterface anzeigt

Betreiben des 9DOF an einem Gnublin-Board

Der Sparkfun 9DOF kombiniert 4 Sensoren auf einem Board, womit 9 Freiheitsgrade erreicht werden. Die Daten aller vorhandenen Sensoren werden von einem auf dem gleichen Board sitzenden ATmega328 verarbeitet und über ein serielles Interface ausgegeben. Dadurch eignet sich der der 9DOF z.B. als Kontrollmechanismus für autonome Fahrzeuge oder Bildstabilisierungssysteme etc. Die auf dem 9DOF befindliche Firmware kann zudem durch eigene ersetzt werden. Dieser Schritt war für unser Projekt jedoch nicht erforderlich, die ursprünglich installierte Firmware reichte vollkommen aus. Um den Sensor an das Gnublin-Board anzuschließen wurde eine USB-B UART Bridge verwendet. Mithilfe dieses Bausteins ist es möglich den Sensor über USB anzuschließen. Diese Brücke verbindet den seriellen Ausgang des 9DOF mit dem (Host-)USB-Port des Gnublin. Angesprochen werden kann der Sensor nach erfolgreichen verbinden der 3 Komponenten mittels eines geeigneten Terminalprogramms (z. B. picocom oder screen) über das Gerät /dev/ttyUSB0. Vorher muss jedoch auf dem Gnublin das Kernelmodul cp210x geladen werden. Dieses dient als Treiber für die UART-Bridge und ist im Auslieferungszustand des Gnublin vorhanden. Das Kernelmodul muss also nicht, zuerst kompiliert werden.

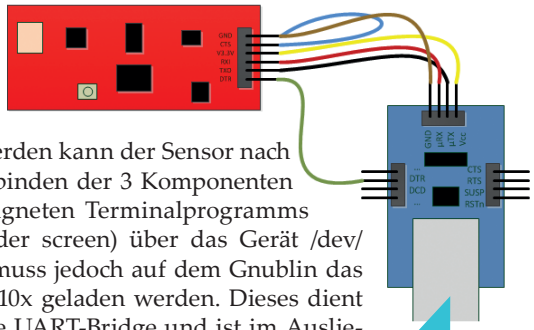


Abb. 1: Aufbau

Der Dämon

Der Dämon bildet das Herzstück der gesamten Softwarebestandteile. Hier werden die Daten live aus dem Sensor gelesen und über diverse Schnittstellen anderen Diensten und Programmen zur Verfügung gestellt. Beliebige Erweiterungen zu diesem Projekt können so über verschiedene Schnittstellen jeweils Zugang zu den jeweils aktuellen Werten des Sensors Erlangen. Außerdem bringt der Einsatz einer zentralen SW-Komponente zum einen den Vorteil der Abstraktion der Daten von der Datenquelle und zum anderen stehen die Daten für mehrere Prozesse gleichzeitig zur Verfügung. Würden ein Clientprogramm direkt auf das serielle Interface des 9DOF zugreifen, würde das Terminal den Zugriff für einen zweiten Prozess verweigern. Der Dämon bietet die Möglichkeit die Daten in einer Datei zu loggen, über einen Netzwerksocket zur Verfügung zu stellen oder für andere lokale Prozesse in einem Shared-Memory-Bereich zur Verfügung zu stellen. Um diese Aufgaben möglich unabhängig erfüllen zu können haben wir uns dazu entschieden für jede dieser Aufgaben einen nebenläufigen POSIX-Thread einzusetzen.

Thread	Aufgabe
readValues	Liest die aktuellen Werte des Sensors von der seriellen Schnittstelle und schreibt diese in einen gemeinsam genutzten Datenbereich
printValuesSTDOUT	Schreibt die Daten auf die Standardausgabe
printSocket	Stellt die aktuellen Sensordaten per TCP-Netzwerksocket bereit
fileWriter	Schreibt die Sensordaten in eine Logdatei
memoryWriter	Schreibt die Daten für das Webinterface in einen Shared-Memory

Tabelle 1: Eckdaten

Einrichten des Webserver und aktivieren von cgi

Um eine Webseite auf dem Gnublin bereitzustellen, wird ein Webserver benötigt. Der Aufwand diesen aus den Quellen für den Gnublin zu übersetzen entfällt, da bereits im Auslieferungszustand ein durchaus brauchbarer Webserver enthalten ist. Der Webserver lighttpd wird in diesem Projekt dazu verwendet, die jeweils aktuellen Sensordaten über ein per CGI angebundenes Programm per HTTP auszuliefern. CGI (Common Gateway Interface) bietet eine sehr einfache Möglichkeit dynamische Webseiten zu generieren oder mit dem Gerät zu kommunizieren und es dadurch ggf. auch zu steuern. Mit CGI lassen sich mit nahezu jeder beliebigen Programmier- oder Skriptsprache dynamische Webseiten erstellen. Bei Aufruf des Programms über den Webserver wird eine Instanz des Programms gestartet und dessen Ausgaben auf stdout per HTTP an den Client übermittelt. Dies können komplette HTML-Seiten, Plain-Text, JSON-Objekte, Binär-, oder auch beliebige andere Daten sein. Starten lässt sich der mitgelieferte Webserver durch ausführen eines Skripts lighttpd-init.sh, welches im Auslieferungszustand im Heimatverzeichnis von root hinterlegt ist. Anschließend lassen sich (eine zuvor konfigurierte Netzwerkschnittstelle vorausgesetzt) HTML-Seiten unter /srv/www/htdocs/ von einem Client per Webbrowser aufrufen. Häufig werden Skriptsprachen wie Python, PHP oder Perl zur Generierung von dynamischen Seiten eingesetzt. Dieser Ansatz wäre auf dem Gnublin-Board zwar durchaus machbar, allerdings starten diese Interpreter auf dem Gnublin-Board aufgrund des geringen Arbeitsspeichers mit extremer Verzögerung. Daher ist es sinnvoller, native Programme wie C/C++ o. ä. zu verwenden. Im jetzigen Zustand beherrscht der Webserver zwar schon das Ausliefern von statischen Seiten, die CGI-Schnittstelle um nun endlich dynamische Seiten erzeugen zu können muss jedoch erst aktiviert werden. Um das passende Modul des lighttpd zu laden, muss die die Konfigurationsdatei /etc/lighttpd/lighttpd.conf angepasst werden. Hier muss der bereits enthaltene include-Befehl für das CGI-Moduls lediglich noch einkommentiert werden.

Anschließend kann entsprechend den Anforderungen die Konfiguration für das CGI-Modul selbst vorgenommen werden. Da Binärcode ausgeführt werden soll, für den kein Interpreter erforderlich ist, muss das Mapping von Interpreterprogrammen zu Dateiendungen in der Datei /etc/lighttpd/conf.d/cgi.conf wie folgend angepasst werden.

```
cgi.assign = ( „.pl“ => „/usr/bin/perl“,
„.rb“ => „/usr/bin/ruby“,
„.erb“ => „/usr/bin/eruby“,
„.py“ => „/usr/bin/python“,
„.cgi“ => „“ )
```

Da zuerst der Interpreter /usr/bin/perl für die Dateiendung .cgi registriert ist, stehen Perl-Skripte nach anpassen der Konfiguration nicht mehr zur Verwendung über CGI zur Verfügung. Aufgrund der Tatsache, dass Skripte wegen der langen Startzeit der Interpreter auf dem Gnublin in unserem Projekt keine Verwendung finden, ist dies kein Problem. Alternativ könnte man für Perl oder Binärprogramme natürlich auch eine andere Dateiendung vergeben. Damit die Konfiguration aktiv wird, ist es notwendig, den Webserver über /etc/init.d/lighttpd restart neu zu starten. Nun ist der Webserver startklar für CGI.

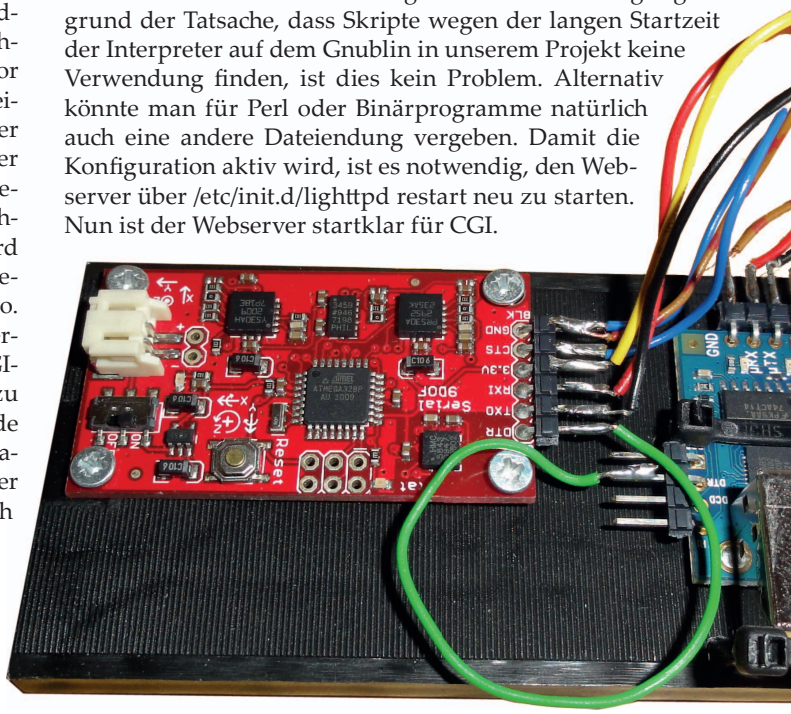


Abb. 2: Versuchsaufbau

Das CGI-Programm

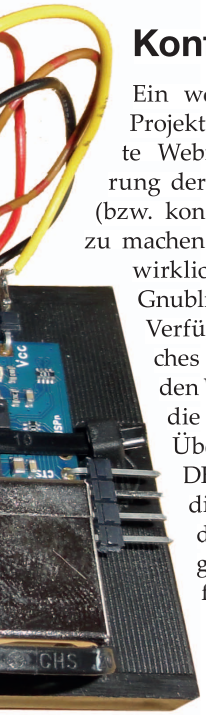
Um die Daten über einen Webserver abrufbar zu machen verwenden wir ein CGI-Programm (readmemory.cgi), das die Daten über eine Schnittstelle vom Dämon liest und über HTTP ausliefert. Bei der Kommunikation des cgi-Programm mit dem Dämon haben wir uns für SharedMemory entschieden. Hierbei werden bestimmte Datenstrukturen eines Prozesses, mittels eines definierten Schlüssels, für einen anderen freigegeben. Um den Datenschutz und -Integrität sicher zu stellen, ist es einerseits möglich den Bereich mit Zugriffsrechten zu versehen, andererseits ist es möglich die Daten mit Semaphoren vor konkurrierenden Zugriffen zu schützen. Da der memoryWriter-Thread des Server immer nur die aktuell gültigen Daten des Sensor in den Shared-Memory schreibt können wir über das Programm immer die aktuellen Daten abrufen. Das CGI-Programm liefert die vom Dämon erhaltenen Daten in das, im Webbereich sehr beliebte, JSON-Format um. Dies gewährleistet zum einen, dass die Daten von beliebigen Clientprogrammen über HTTP abruf- und verarbeitbar sind (für alle gängigen Programmier-/Skripsprachen gibt es JSON-Anbindungen) zum anderen erleichtert es dem Entwickler durch die einfache Lesbarkeit die Arbeit beim Entwickeln. Eine weitere wichtige Teilaufgabe ist somit erledigt, es können die jeweils aktuellen Sensordaten per HTTP abgefragt und im Browser dargestellt werden. Eine Beispielhafte Ausgabe beim Aufruf des CGI-Programms wäre z. B.

```
{
„accX“: 900,
„accY“: 537,
„accZ“: 434,
„magneto1“: 105,
„magneto2“: 821,
„magneto3“: 848,
„gyroX“: 916,
„gyroY“: 995,
„gyroZ“: 25
}
```

Da dieses Format zwar einen Einblick in die gelieferten Daten gibt, aber dennoch nicht gerade benutzerfreundlich ist gibt es einen weiteren Bestandteil der das Abfragen der Daten beim readmemory.cgi für uns übernimmt und die Daten grafisch ansprechend aufbereitet.

Die Visualisierung mittels HTML+JS (data.html)

Das Gnublin-Board soll die Daten des Sensors über ein Webinterface für den Benutzer zugänglich machen. Prinzipiell wird diese Aufgabe bereits von readmemory.cgi übernommen, die Anzeige der Daten als „rohes" JSON-Objekt ist jedoch nicht sehr komfortabel. Um die Daten ansprechend darzustellen ist per Webserver eine HTML-Seite aufrufbar, welche die per readmemory.cgi verfügbaren Daten grafisch aufbereitet. Diese Aufgabe übernimmt eine HTML-Seite (data.html) die ebenfalls vom Webserver abgerufen wird. Ein in der HTML-Seite enthaltener aktiver JavaScript-Teil ruft periodisch die aktuellen Daten über readmemory.cgi ab und legt diese mit dem aktuellen Zeitstempel in einem Array ab. Damit dies ohne weiteren weitere SW auf dem Server und auch ohne erneutes Laden der Seite geschehen kann werden die Daten über AJAX nachgeladen. Die gesammelten Daten werden anschließend durch die JavaScript-Library JQuery.Flot grafisch dargestellt. Dem Benutzer wird ermöglicht das Updateintervall über einen Schieberegler interaktiv zu bestimmen, sodass er auf die Häufigkeit der Aktualisierung Einfluss hat. Weiterhin kann der Benutzer über einen Schieberegler vorgeben wie viele Werte gleichzeitig im Graphen dargestellt werden sollen. Der Updatevorgang geschieht periodisch, in dem jeweils nach, dem vom Benutzer gewählten, Zeitabschnitt eine JavaScript-Funktion (update()) aufgerufen wird, welche die Aktualisierungs- und Darstellungsfunktion übernimmt. Durch Einsatz von AJAX beim Nachladen der Daten und JQuery.Flot bei der Darstellung lassen sich die Sensordaten „live" im Browser betrachten, ohne dass ein erneutes Laden der gesamten Seite erforderlich ist.



Konfigurationsloser Zugriff auf das Webinterface

Ein weiterer Schwerpunkt des Projekts war es, das bereitgestellte Webinterface zur Visualisierung der Daten möglichst einfach (bzw. konfigurationslos) erreichbar zu machen. Um dieses Ziel zu verwirklichen, wurde mithilfe des Gnublin-Board ein WLAN zur Verfügung gestellt, über welches andere Geräte Zugriff auf den Webserver (und damit auf die Seite data.html)haben. Über die Netzwerkdienste DHCP und DNS wurden die verbundenen Clients dann dahingehen konfiguriert, dass beliebige Anfragen an Netzwerkgeräte an unser Gnublin-Board gesendet werden. Dadurch wurde der konfigurationsfreie Zugriff erreicht.

Zum Aufspannen eines WLAN-Netzwerks wird normalerweise ein WLAN-Stick benötigt, welcher sich in den sog. Master-Modus versetzen lässt. Da wir lediglich Basisfunktion benötigen und keine Infrastruktur aufbauen möchten genügt es auch mit dem Gnublin einfach ein Ad-hoc-Netzwerk zu eröffnen. Bei einem Ad-hoc-Netzwerk sind alle Teilnehmer gleichberechtigt und die Daten werden mittels Algorithmen von Station zu Station zum Ziel weitergeleitet. Damit nun andere Teilnehmer, die dem Netzwerk beitreten, eine IP-Adresse bekommen und so überhaupt mit anderen Netzwerkgeräten kommunizieren zu können, müssen IPs über DHCP verteilt werden. Diese Aufgabe übernimmt das Programm udhcpd welches sich bereits auf dem Gnublin-Board befindet aber noch entsprechend konfiguriert werden muss. Konfigurationsfreier Zugriff auf das Webinterface bedeutet, dass der User nur einen Browser öffnen muss und sofort auf der von uns gewünschten Webseite landet. Damit dies überhaupt möglich ist, müssen beliebige Hostnamen (z.B. google.de) in die IP-Adresse des Gnublin aufgelöst werden. Hierzu muss auf dem Gnublin ein DNS-Server laufen, der Anfragen von den Clients im WLAN beantwortet. In kleinen Netzwerken wird die DNS-Auflösung meist vom Dienst dnsmasq zur Verfügung gestellt. Da dnsmasq dbus benötigt, dies jedoch auf dem Gnublin-Board noch nicht zur Verfügung stand, haben wir uns dazu entschlossen einen eigenen DNS-Server zu programmieren. Da der Server auf alle Anfragen mit derselben IP, der des

Gnublin-Board, antworten soll, war dies nicht sehr aufwändig. Damit jedoch die DNS-Anfragen der Clients überhaupt bei unserem Gnublin landen muss der entsprechende Eintrag in der Konfigurationsdatei des DHCP-Server angepasst werden.

Ein Aufruf der Webseite http://www.hs-augsburg.de/fakultaet/informatik/index.html wird jetzt dank unseres Mockup-DNS-Servers in http://<IP-des-Gnublin>/fakultaet/informatik/index.html umgewandelt. Das Problem ist jedoch, dass das Verzeichnis bzw. die Datei fakultaet/informatik/index.html auf dem Webserver des Gnublin-Board nicht existiert und vom Webserver mit einem HTTP-404 Fehler quittiert wird. Um dieses Problem zu umgehen benötigt man einen serverseitigen Redirect, der alle Aufrufe an den Webserver auf unsere Seite umleitet. Dazu muss im Webserver lighttpd das Modul mod_redirect aktiviert werden. Mit dem Eintrag url.redirect = („^(?!/logger/.*)“ => „/logger/index.html“) in der Konfigurationsdatei leitet man alle Anfrage auf die Seite /logger/index.html weiter. Da die Seite des Gnublin wiederum Bilder, JavaScript und cgi-bin-Programme nachlädt müssen diese Anfragen ausgefiltert werden. Anderenfalls würde die Anfrage nach readmemory.cgi wieder die Seite index.html liefern, welche dann wiederum Ressourcen nachladen würde. Da dies zu einer Endlosschleife führen würde leiten wir die URLS die mit /logger/.* beginnen nicht um.

Programm als Dämon laufen lassen

Damit der Dämon tatsächlich wie ein richtiger Dämon (und damit als Kindprozess von init) läuft, sind zu Programmstart einige Schritte notwendig. Mit fork() wird ein Abbild des aktuellen Prozesses angefertigt und als Kindprozess ausgeführt. Direkt anschließend wird der Elternprozess beendet. Um den Kindprozess vom Elternprozess zu lösen und an init anzuhängen wird die Funktion setsid() aufgerufen. Abschließend wird das Arbeitsverzeichnis auf das root-Verzeichnis (/) gesetzt, die umask auf den Wert 0 gesetzt und die standardmäßig geöffneten Filehandels stdin, stdout und stderr auf /dev/null umgelenkt, da diese im Serverbetrieb keine Verwendung finden. Wird einer dieser Teilschritte des Kindprozesse nach fork() nicht aufgerufen, ist es möglich, dass der Dämon nicht startet oder nur fehlerhaft funktioniert. Das Ausführen als Dämon wird in der Methode daemonize() abgewickelt.

```
void daemonize(void){
if(getppid() == 1) return; /* if a daemon,
then parent ist init */
pid=fork(); /* Fork off the parent */
if(pid>0){ /* if we got a good PID, then we
can exit the parent process. */
printf(„parent exit\n“);
exit(0);
}
/* now we are child */
...
}
```

Um den Dämon später mit einem init-script sauber beenden, oder neu starten zu können, wird noch ein PID-File benötigt. In dieser Datei steht nur die Prozess-ID des Dämon. Mit Hilfe des Befehl kill kann jetzt an diese PID das SIGTERM Signal gesendet werden. Damit der Prozess auf das Signal reagiert, muss für jedes abzufangende Signal ein handler definiert werden.

```
signal(SIGINT, exitHandler); /* ctrl-c */
signal(SIGQUIT, exitHandler); /* ctrl-
<igendwas> */
signal(SIGTERM, exitHandler); /* normale
terminierung */
```

In der Manpage zu signal() wird von dessen Benutzung abgeraten, da sich dieser Befehl aus historischen Gründen in unterschiedlichen Versionen unter UNIX und LINUX unterscheidet. Dies führt dazu, dass es beim Portieren der Software zu unerwarteten Problemen kommen kann. Als bessere Alternative wird sigaction() empfohlen.

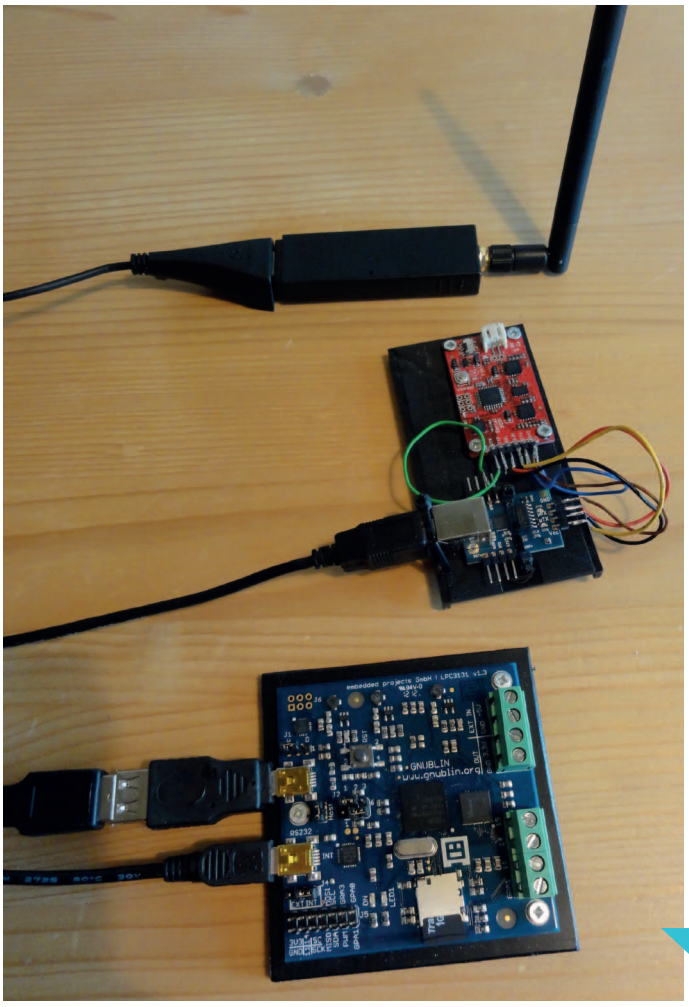
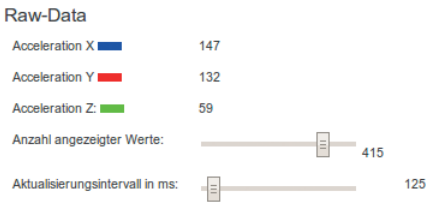


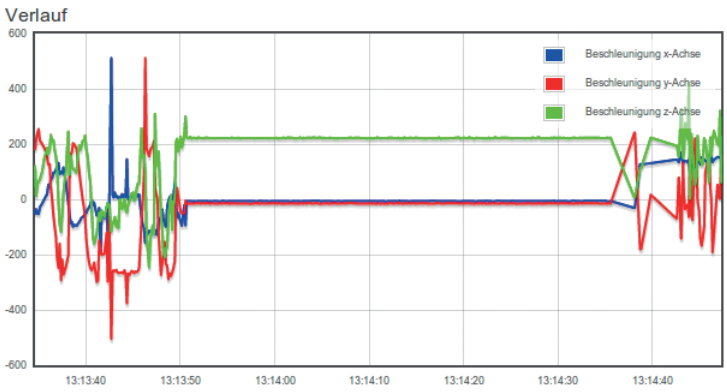
Abb. 3: Gnublin, WLAN und 9dof

Links

http://elk.informatik.fh-augsburg.de/pub/rtlabor/elinux/sommer12

Beschleunigungsmesser:





Magnetometer:

Value 1: 387 Value 2: 395 Value 3: 355

Gyroskop:

Gyro X: -351 Gyro Y: -417 Gyro Z: -87

Abb. 4: Anzeige Magnetometer und Gyroskop

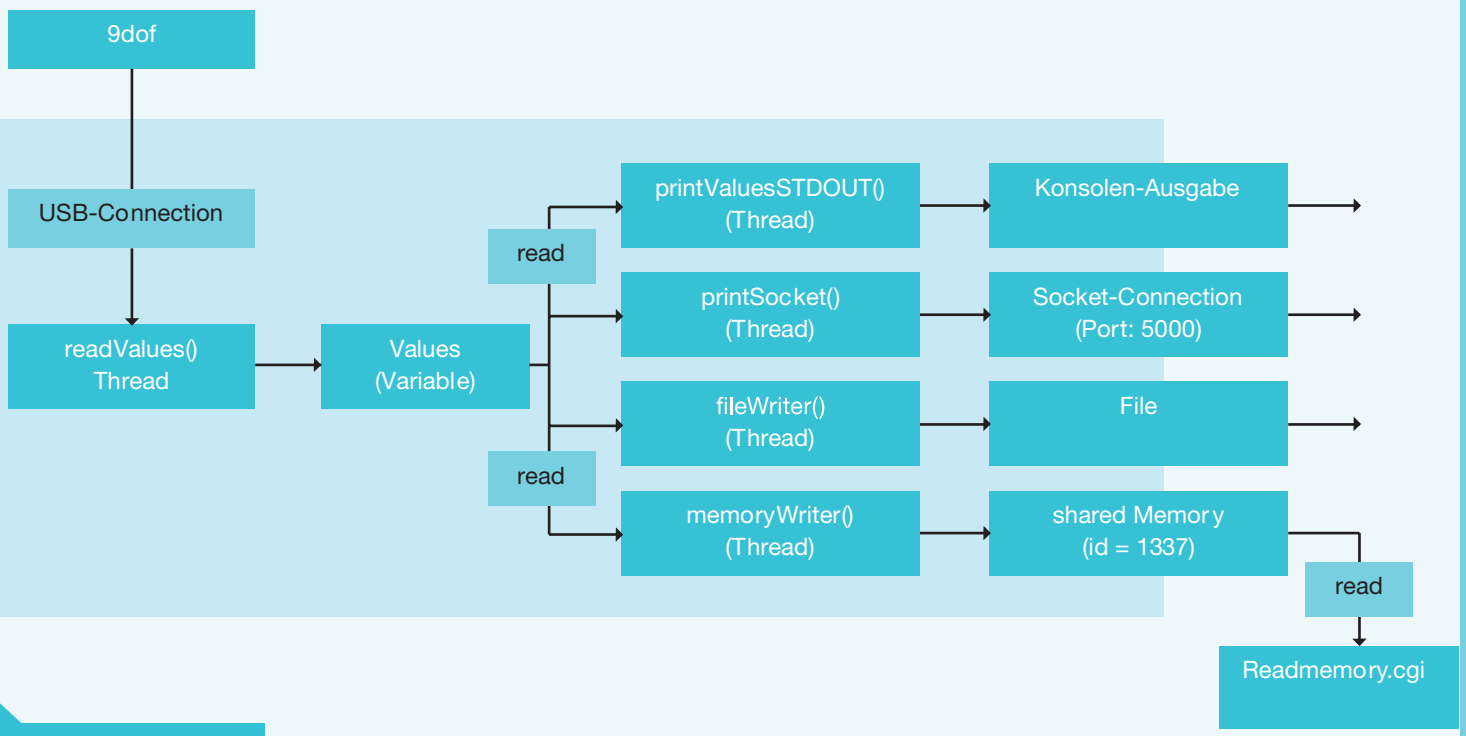


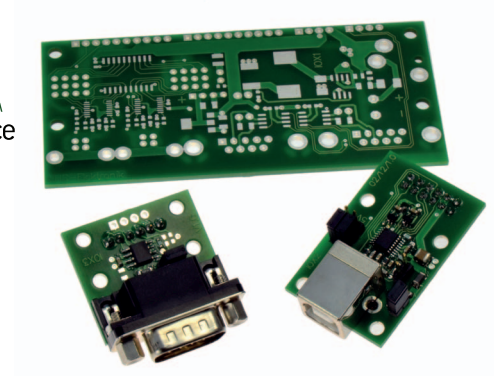
Abb. 5: 9dof-Logger

Marktplatz / Neuigkeiten

Die Ecke für Neuigkeiten und verschiedene Produkte

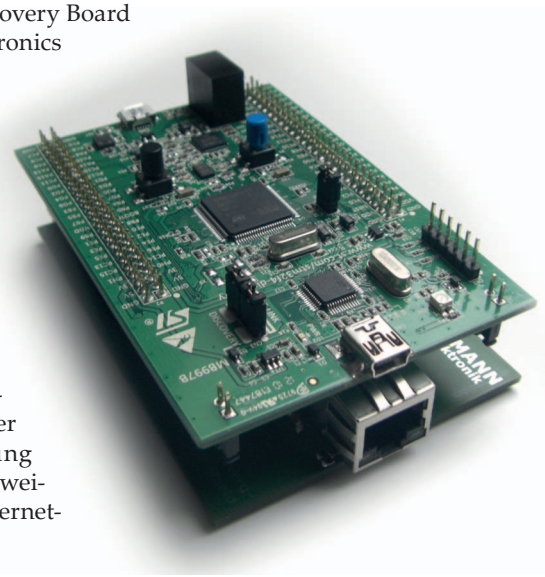
Die IOX-Module - perfekt für CAN-Entwicklungen

Mit den neuen IOX Module von MHS-Elektronik wird die Entwicklung von CAN-Projekten in der Prototypenphase stark erleichtert. Die Module bieten CAN-Treiber, Versorgungsspannungen und USB-UART Schnittstellen zur einfachen Verwendung mit Prototypen Plattformen an. Mehr dazu in der nächstens Ausgabe des embedded projects Journal.



STM32F4-Discovery-Board-Shield mit Ethernet oder WLAN

Mit dem STM32F4 Discovery Board bietet ST Microelectronics ein sehr günstiges Entwicklungsboard mit integriertem Programmier- und Debug-Adapter sowie einem leistungsfähigen Mikrocontroller. Der Artikel (nächstes Journal Ausgabe 15) beschreibt die ersten Schritte mit dem Entwicklungsboard und der Entwicklungsumgebung CooCox sowie die Erweiterung um eine Ethernetschnittstelle.

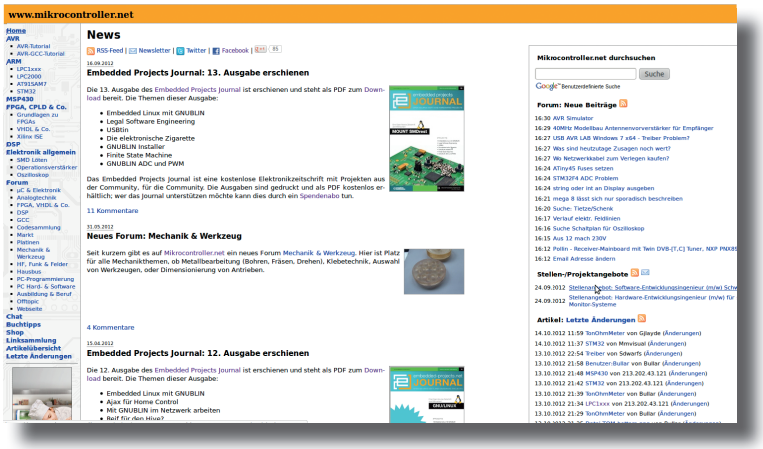


Mikrocontroller.net - über jeden Artikel jetzt einzeln diskutieren

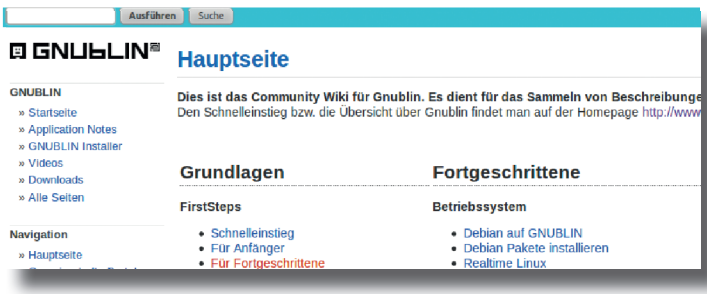
Gemeinsam mit mikrocontroller.net bieten wir einen neuen Service zu unserem embedded projects Journal an. Zu jedem Artikel wird es ab sofort ein eigenen Thread im Forum von mikrocontroller.net geben. So können sich Interessierte sammeln und Themen Rund um den Artikel einfach und unkompliziert austauschen.

Die Links zu den Artikeln werden im Forum bekannt gegeben.

http://www.mikrocontroller.net



Gnublin-Wiki



In den letzten Wochen hat sich einiges im Community Wiki zu unserem Projekt Gnublin getan.

So findet man jetzt viele Beschreibungen, vom Display über die Kamera bis zu Temperatursensoren.

http://wiki.gnublin.org

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:

journal@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 [2] in eine Liste für die gesponserten Abos oder ein Spendenabo 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 über einen Online - Shop [2] beziehen.

[1] Internetseite (Anmeldeformular gesponserte Abos): http://journal.embedded-projects.net

[2] Online - Shop für Journal:
http://www.embedded-projects.net

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

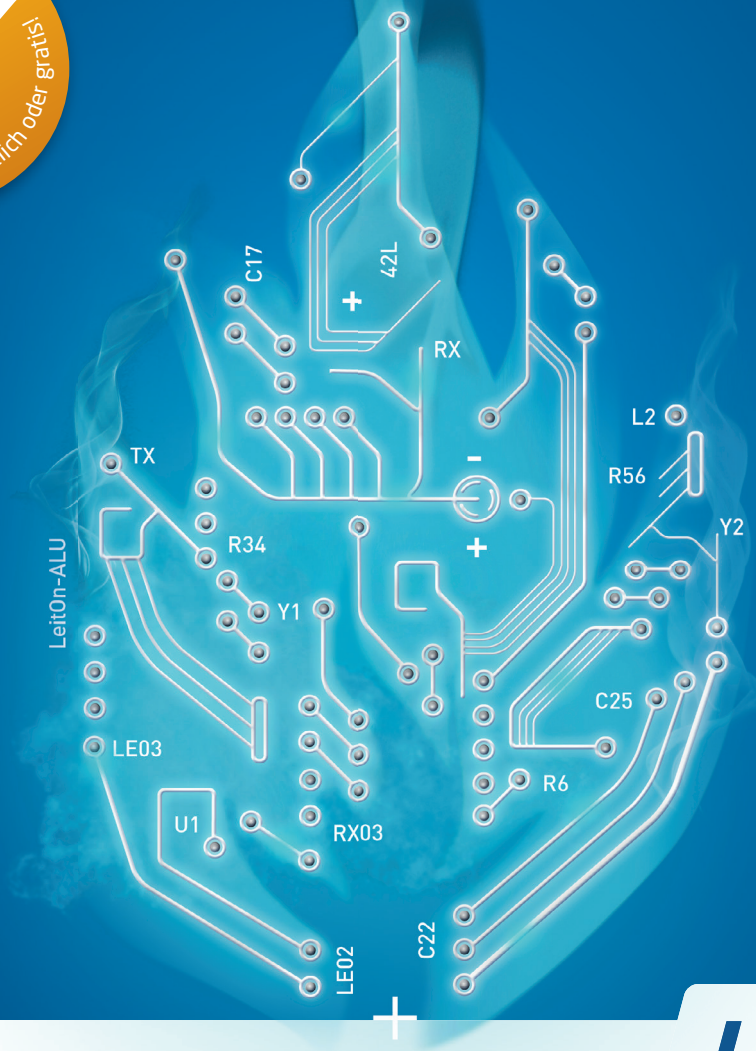
Veröffentlichung: 4x / Jahr
Ausgabenformat: PDF / Print
Auflagen Print: 2500 Stk.
Einzelverkaufspreis: 1 €

Layout / Satz: EP
Druck: flyeralarm GmbH
Titelfoto: Claudia Sauter

GERADE WENN'S MAL HEISS HERGEHT.

LEITERPLATTEN AUS ALUMINIUM ONLINE BESTELLEN.

ab 12 h Leiterplatten-Express! Pünktlich oder gratis!



www.H1QN.de

LEITON
RECHNEN SIE MIT BESTEM SERVICE

Vorsicht, heiß! Starke Hitzeeinwirkung ist eine echte Herausforderung – besonders in der Hochleistungs-LED-Technik oder wenn wichtige Leistungsbauteile permanent hohen Temperaturen ausgesetzt sind. Aluminium-Platinen von LeitOn überzeugen durch **hohe Hitzeresistenz** sowie einen Wärmeleitwert von bis zu 3,0 W/mK in der Premiumvariante „Polytherm TC-Lam 3.0". Die vielen Vorteile liegen auf der Hand: **Kostenreduktion** durch höhere Lebensdauer und Zuverlässigkeit, **Platzersparnis** dank Integration der aufwendigen Kühlmechanismen sowie **Performancesteigerung** durch höhere Leistungsdichte der Anwendung. 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**