Dokumentation – Digital-Thermometer

Testversion: Dieses Projekt wurde bisher erst einmal aufgebaut und bedarf ggf. an manchen Stellen noch etwas Überarbeitung. Durch den flexiblen Mikrocontroller bietet es jedoch viele Möglichkeiten für fortgeschrittene Bastler.

Inhaltsverzeichnis:

Funktionsprinzip

Temperaturfühler → Mikrocontroller → 7-Segment Display

Temperaturfühler:

Ein temperaturabhängigen Widerstand bildet in Reihe mit einem Widerstand einen Spannungsteiler. Da die so gewonnene Spannung zu klein ist um von einem Mikrocontroller ausgewertet zu werden, wird sie analog mittels eines Operationsverstärkers auf einen Pegel zwischen 0V und 5V gebracht. Der Analog-Digital-Wandler (ADC) kann diese Spannung nun als digitalen Wert einlesen. Bei einem 10 Bit ADC sind dies 1024 Stufen. Das Auflösungsvermögen beträgt somit ca. 5mV. Wenn man das Temperatur-Signal also für den Bereich -30°C bis 70°C zwischen 0V und 5V auspegelt, so erhält man eine Auflösung von 0,1°C.

7-Segment Display:

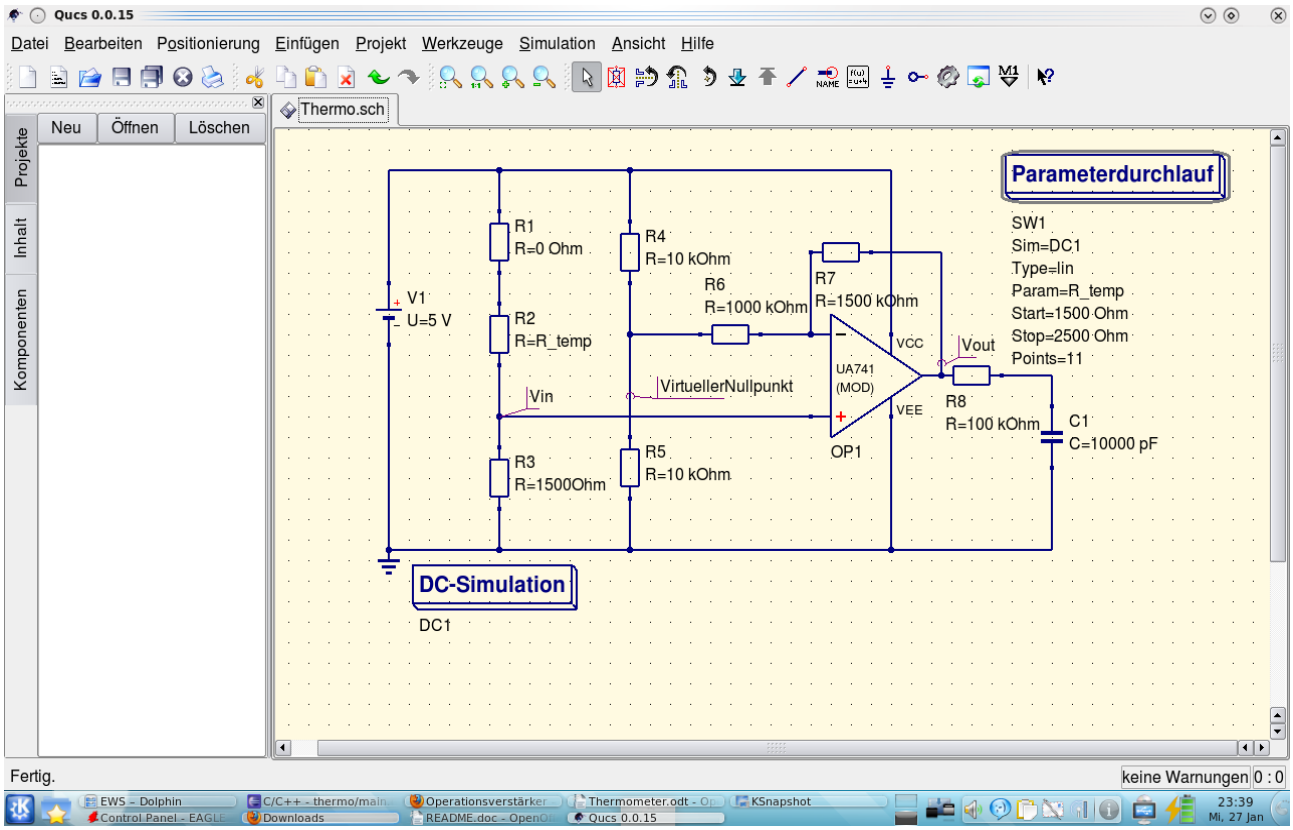
Das 7-Segment Display besteht aus mehreren 7-Segment Anzeigen, welche wiederum aus sieben LEDs, zu einer Anzeige angeordnet, bestehen. Die sieben LEDs haben je nach Modell eine gemeinsame Kathode oder Anode und sieben Anschlüsse für die jeweiligen Segmente. Da man zur Ansteuerung eines dreistelligen Display 21 Bit bzw. 21 I/O Ports bräuchte, verwendet man eine Art Multiplexing. Das heißt, dass jeweils immer nur eine 7-Segment Anzeige nach der anderen beschaltet wird. Nur dies geschieht so schnell, dass es für das menschliche Auge nicht wahrnehmbar ist. Dadurch werden nur noch 7+N Bit (N:= Anzahl der 7-Segment Anzeigen) benötigt.

Mikrocontroller:

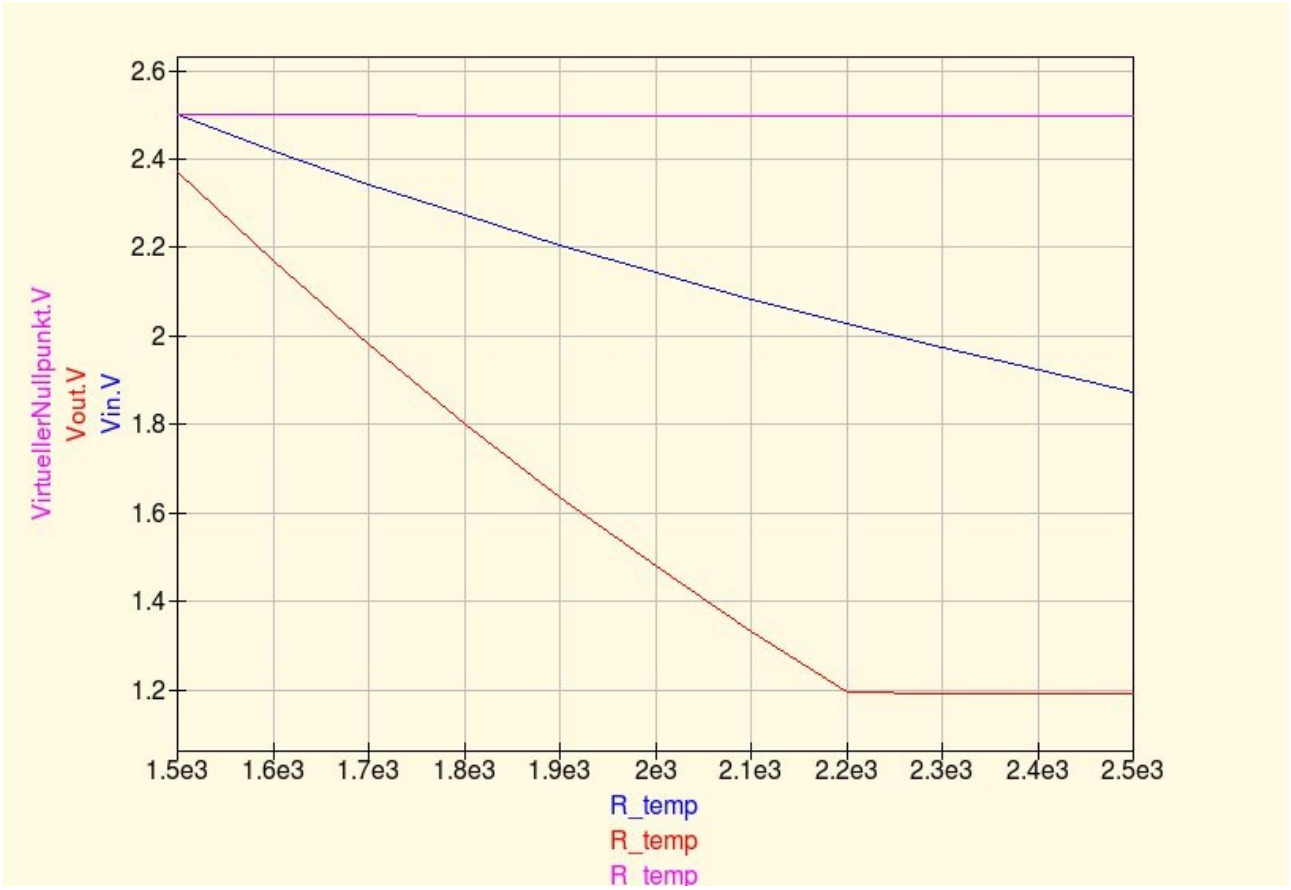
Der Mikrocontroller rechnet die gemessene Spannung in eine Temperatur um und stellt diese auf einem 7-Segment Display dar. Dies geht bei einer linearen Temperatur-Spannungs-Kennlinie über eine Geraden- Gleichung. Bei einem nicht linearen Verlauf könnte man die Temperatur anhand einer eingespeicherten Tabelle bestimmen.
Der Mikrocontroller kann in C programmiert werden und ist somit sehr flexibel. Es wäre auch Möglich das 7-Segment Display zusätzlich als Uhr zu nutzen.

Aufbau:

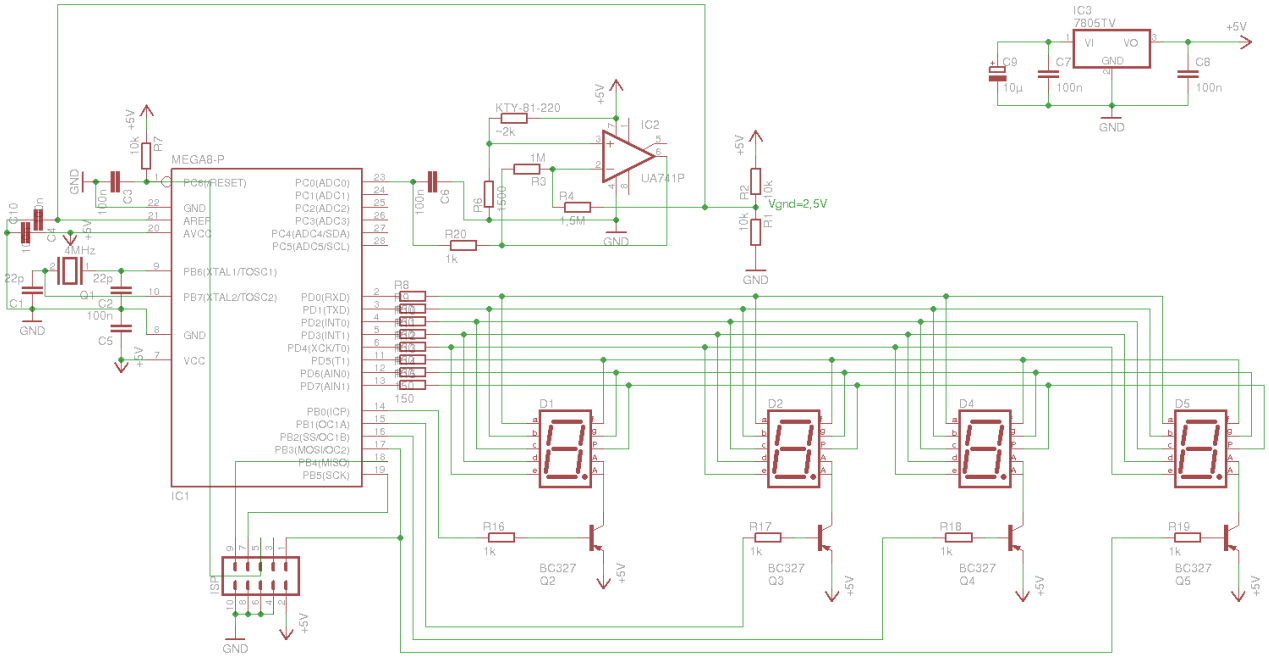
Temperaturfühler - Schaltplan und Simulation:



Realisiert wurde der temperaturabhängige Widerstand hier durch einen Transistor. Dies ist zwar nicht die beste Wahl, aber in jeder Bastelkiste zu finden. Das Temperatur-Signal des Spannungsteilers(Vout) wird von einem Operationsverstärker, welcher als nicht invertierender Verstärker beschaltet ist (Details sind bei Wikipedia nachzulesen), um den Faktor 8 verstärkt(Vout2). Praktisch ist es nicht möglich das Signal zwischen 0 und 5V auszupegeln, da der Operationsverstärker nahe der Versorgungsspannung sonst in die Sättigung gerät. Da der Operationsverstärker normalerweise mit einer Bipolaren Spannung versorgt wird, bilden R2 und R4 eine 'virtuelle Masse', wobei das Verhältnis R2 zu R4 zusätzlich noch einen Offset enthält. Wenn die Kennlinie sich als nicht ausreichend linear herausstellen sollte, so kann man über einen industriellen Thermowiderstand nachdenken.


VirtuellerNullpunkt.V
Vout.V
Vin.V
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1.5e3
1.6e3
1.7e3
1.8e3
1.9e3
2e3
2.1e3
2.2e3
2.3e3
2.4e3
2.5e3
R_temp
R_temp
R_temp

Schaltplan



Erläuterungen:

Links des Mega8 ist hauptsächlich die standardmäßige Beschaltung des Mega8 zu sehen. (Mehr dazu unter http://www.mikrocontroller.net/articles/AVR-Tutorial:_Equipment o.ä.)
Ebenso ist die ISP – Schnittstelle in fast jeder Schaltung mit dem Mega8 so zu finden.
PortD ist über 150 Ohm Vorwiderstände mit den 7 Segmenten und dem Punkt verbunden. Die Segmentanzeigen haben eine gemeinsame Anode (+) , also leuchten die Segmente bei einem Low-Pegel. Die Pins PB0 -PB3 über 1 kOhm mit den PNP-Transistoren zum Schalten der gemeinsamen Anoden verbunden. Die Transistoren schalten ebenfalls bei logisch „0“ durch, und ermöglichen so das Leuchten der Anzeigen.
Die Temperaturwiderstand bildet zusammen mit R6 einen Spannungsteiler. Diese Spannung wird über einen Operationsverstärker (UA741) in nicht invertierende Verstärkerschaltung (mehr: Wikipedia: Operationsverstärker) um den Faktor V=R4/R3 +1 =2,5 verstärkt. Da der Operationsverstärker normalerweise eine Bipolare Spannungsversorgung benötigt, gibt es einen Virtuellen Nullpunkt von 2,5V. Somit ist der Opamp quasi mit +/- 2,5V versorgt.
Über einen Tiefpass geht die Ausgangsspannung auf den ADC0 des Mega8.
Oben rechts ist die Spannungsversorgung zu sehen, welche für konstante 5V sorgt.

Programmierung

Der Atmega8 wurde in C programmiert. Wie der Quelltext compiliert und auf den µC geschrieben wird ist stark Software abhängig. Genaueres findet man entsprechenden Tutorial im Internet oder erfragt es beim EWS- Team.

Quelltext

```
/*
 * main.c
 *
 *  Created on: 15.12.2009
 *      Author: axel
 */



#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>
#include <avr/interrupt.h>


/* 7-Segment-Anzeige Zuordnung: */
#define seg0 PB3
#define seg1 PB2
#define seg2 PB1
#define seg3 PB0
#define EP PORTB
#define SegPort PORTD

/*Eichungs Parameter Scale ca. 20; Offset ca. 2200 */
#define TEMPSCALE 21
#define TEMPOFFSET 3000
//Speicherarry für "Bilder" der 7-Segment Anzeigen
volatile uint8_t display[]= { 0, 0, 0, 0 };
//Speicherarray für grafische Umsetzung der Zahlen "~" invertiert, da Segmente bei low leuchten
volatile const uint8_t segments[] = { ~0x3f, ~0x6, ~0x5b, ~0x4f, ~0x66, ~0x6d, ~0x7d, ~0x7,
~0x7f, ~0x6f };
//Zähler für task_segments()
volatile uint8_t pos = 0;
//Variable zur Verwaltung der Kommastelle
volatile int dp[] = {0,0,1,0};

// Multiplexing der 7-Segment Anzeigen
void task_segments(void)
{
	switch(pos)
	{
	case 0: EP=~0x01; SegPort=display[pos]; pos++; break;
	case 1: EP=~0x02; SegPort=display[pos]; pos++; break;
	case 2: EP=~0x04; SegPort=display[pos]; pos++; break;
	case 3: EP=~0x08; SegPort=display[pos]; pos=0; break;
	}

}

//speichern eines int in das Segment Array + Setzen der Kommastelle
```

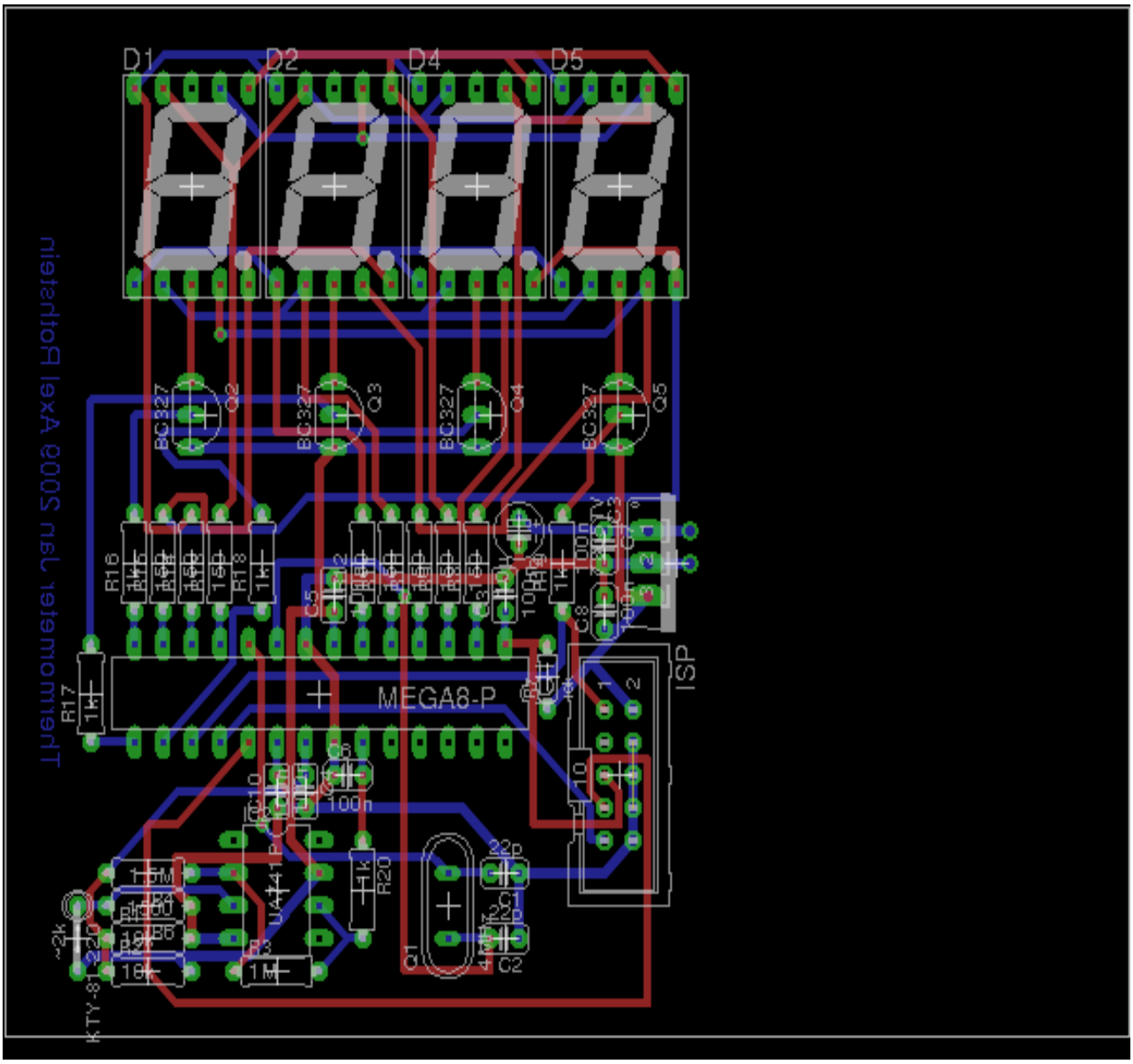
```
void set_segments(int16_t val)
{
	//negative Zahlen fehlen noch!!!
	uint8_t minus =0;
	if(val<0)
	{
		val=-val;
		minus=1;
	}
	display[seg0]=segments[val%10];
	if(dp[seg0])display[seg0]&= ~(1<<7);
	display[seg1]=segments[(val%100)/10];
	if(dp[seg1])display[seg1]&= ~(1<<7);
	display[seg2]=segments[(val%1000)/100];
	if(dp[seg2])display[seg2]&= ~(1<<7);
	display[seg3]=segments[(val%10000)/1000];
	if(dp[seg3])display[seg3]&= ~(1<<7);
	if(minus)
		{
		if(val%10000==0)
		display[seg3]=0b10111111;
		else display[seg3]=0b10000110; //Setzt "E" als Fehlermeldung
		}


}

//Initialisierung des Timers
void timer_enable(void)
{
	TCCR0=2; //Timer0 wird aktiviert mit CPU-Takt / 8
	TCNT0=0; //Timer0 wird auf 0 gesetzt
	TIMSK|=(1<<TOIE0); // Timer Overflow Interrupt enable
}
//Initialisierung der Interrupt Routine bei Timer0 Overflow
ISR(TIMER0_OVF_vect)
{
	task_segments();
}

void adc_init(void)
{
	/*
	 * ADEN ADC-enable
	 * ADSP Prescaler für ADC eigene Frequenz
	 * ADFR "free-run" - ADC erneuert sich fortlaufend
	 * ADSC start conversion (muss für free run gesetzt sein)
	 */
	ADCSRA |= (1<<ADEN)  | (1<<ADPS2) | (1<<ADPS2) | (1<<ADFR) | (1<<ADSC);
	ADMUX|= (1<<REFS0)|(1<<REFS1); //Interne 2,56 Volt als Referrenzspannung : entfällt
bei neuem Layout!!!
```

```
}

int temp(void)
{
	uint16_t temp_raw =0;
	uint8_t i;
	for(i = 0; i<50 ; i++)
	{
		temp_raw +=ADC;
		_delay_ms(100);
	}
	temp_raw=temp_raw/50;

	return 10*(1024000/(temp_raw) - TEMPOFFSET)/ TEMPSCALE ;
}

void init(void)
{
	/*Ports auf Ausgang setzen*/
		DDRD =0xFF;
		DDRB =0xF;

		timer_enable();
		adc_init();

}

int main(void)
{

	init();

	/*globale Interrupts zulassen*/
	sei();

	for(;;)
	{
		set_segments(temp());
	}
}
```

Fusebits (optional)

Damit der Atmega8 den Quarz zur Takterzeugung benutzt müssen die entsprechenden Fusebits gesetzt werden.
Welche das genau sind kann dem Datenblatt des Atmega 8 entnommen werden.
Die Benutzung des Quarzes ist für die Thermometer-Anwendung sicherlich nicht zwingen notwendig, wäre aber sehr bei z.B. einer Uhr zu empfehlen.

Zusammenbau



Hinweise

Hier noch ein paar Dinge, die beim Löten beachtet werden sollten. Eigentlich kommt man auch von alleine drauf, meist aber erst nachdem man es falsch gemacht hat:

-Bauteile zuerst auf der Oberseite der Platine Löten oder sicherstellen das dies später noch geht. Bsp.: Sockel für Segment Anzeigen oder ISP-Buchse müssen soweit wie möglich nach Oben herausstehen um auf der Oberseite löten zu können.
-Temperatursensor zuerst über ein Kabel mit der Platine verbinden, dies erleichtert das Eichen.
-Überprüfen von später nicht mehr zugänglichen Leiterbahnen, ggf. nachbessern.
-Liste ergänzen falls etwas auffällt.

Eichen

1.)Zum Eichen benötigt man ein Glas mit Eiswasser und ein möglichst genaues Thermometer. Zunächst setzt ihr den TEMPSCALE auf 1 um eine möglichst große Auflösung um den Nullpunkt zu haben. TEMPOFFSET sollte sollte ca. bei 2250 gesetzt sein.
2.)Dann kompiliert ihr euer Programm und übertragt es auf den Atmega8.
3.)Jetzt haltet ihr den Temperaturfühler in das Eiswasser. Nun könnt ihr Raten, Schätzen oder Rechnen wie ihr den Offset anpassen müsst. Wer es ausrechnen will, sollte sich die Funktion temp() im Quelltext genauer anschauen.
4.) TEMPOFFSET anpassen und Schritte 2.) bis 4.) wiederholen bis der Offset „hinreichend genau“ eingestellt ist.
5.)Setzt TEMPSCALE auf ca. 20
6.) Gleicht das Thermometer bei Raumtemperatur mit dem Refferenzthermometer ab, und passt TEMPSCALE solange an bis es „ausreichend genau“ genug ist. TEMPSCALE verhält sich antiproportional zur angezeigten Temperatur.

Erweiterungsmöglichkeiten:

Die Flexibilität des Mikrocontrollers ermöglicht es aus dieser Schaltung nur durch eine neue Programmierung eine Uhr oder einen Temperaturlogger zu machen. Bei einer Uhr wäre es praktisch noch Taster (gegen Masse) zum einstellen der Zeit an PC1 bis PC5 zu löten. Ein Temperaturlogger könnte z.b. stündlich die Werte in den EEPROM des Mega8 schreiben, welche über die ISP ausgelesen werden kann.
Man kann aber auch eine Laufschrift o.ä. realisieren.