Semester/Gruppe/Team:	Abgabedatum:	Protokollführer:
BMT-4 Gruppe 1	12.06.19	YassineTourki
Versuchstag: 05.06.19		**Weitere Teilnehmer:** Thomas Schlundt Kokou Gaitu
Hochschullehrer: Prof. Dr.-Ing. Leutelt Dipl-Ing. Andreas Prosch		

MPP	V3: Digitalvoltmeter	

Inhalt

Einleitung & Laborgeräte

Einleitung

In diesem dritten Praktikumsversuch wird auf verschiedene Arten ein Digitalvoltmeter ausgeführt und realisiert. Für die verschiedenen Verfahren werden C-Programme erstellt und auf den Mikrocontroller TM4C1294 übertragen.

Laborgeräte

Das folgende Equipment wurde gebraucht und verwendet:

- Mikrocomputer (Tiva TM4C1294)
- Oszilloskop
- BCD 7-Segment Anzeige
- XY Joystick
- D/A – Umsetzer
- LED

Aufgabe 2: Wäge verfahren (Sukzessive Approximation)

Das Prinzip der Sukzessiven Approximation arbeitet wie folgt:
Zuerst wird das MSB von Uout auf '1' gesetzt und alle anderen Bits bleiben auf '0'. Dann wird mithilfe des Komparators ein Vergleich durchgeführt ob die zu ermittelnde Eingangsspannung größer oder kleiner als Uout ist. Ist die Spannung größer, dann wird das nächste Bit auch auf '1' gesetzt und das jetzige Bit auf '1' gelassen. Die Bitfolge von Uout lautet dann'1100:0000'. Wäre die Eingangsspannung kleiner gewesen, dann wäre das MSB auf '0' gesetzt worden und das nächste Bit auf '1' gesetzt ('0100:0000'). Dieses Prinzip wird bis zum letzten Bit fortgesetzt. Da hier immer die gleiche Anzahl an Bits überprüft wird, bleibt die Laufzeit immer dieselbe.

Aufbau

Folgender Aufbau ist zu realisieren:

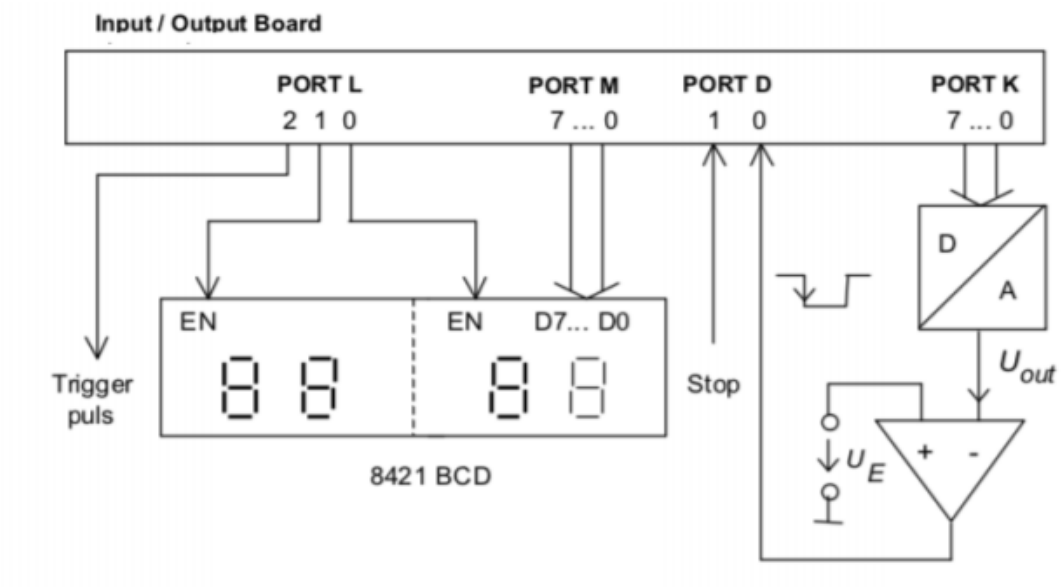

Abbildung 1: Schaltung zum Versuchsaufbau aus dem Aufgabenblatt

Messverlauf einer Spannungs Ermittlung durch das Wäge verfahren.

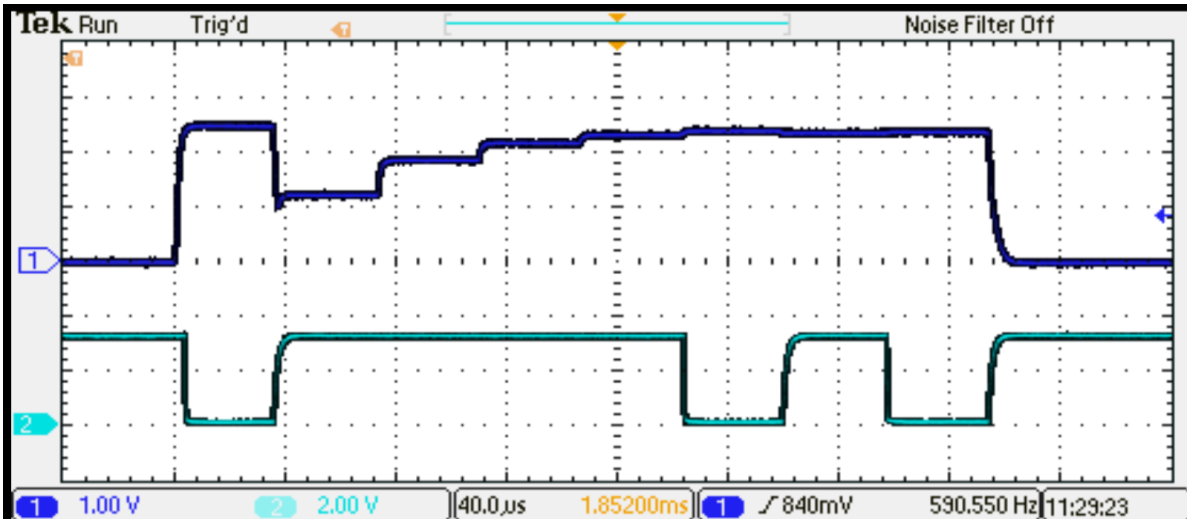

Abbildung 2: CH1: Ausgangsspannung Uout CH2: Komparatorsignal

Wie zu sehen ist, ist schon die erste Spannung zu groß, dies ist daran zu erkennen, dass das Signal von Channel 2 Low ging als die erste Spannung von Channel 1 High war. So sind die nächsten vier Bits High. Bis die Spannung überschritten worden ist und die Bits High und Low geschaltet werden um die Spannung UE anzunähern.

Berechnung der Spannung von Hand: (5/2)*0 + (5/4)*1 + (5/8)*1 + (5/16)*1 + (5/32)*1 + (5/64)*0 + (5/128)*1 + (5/256)*0 = 305/128V ~ 2,382V

Struktogramm

Anhand der Struktogramme wurde das Programm erstellt.

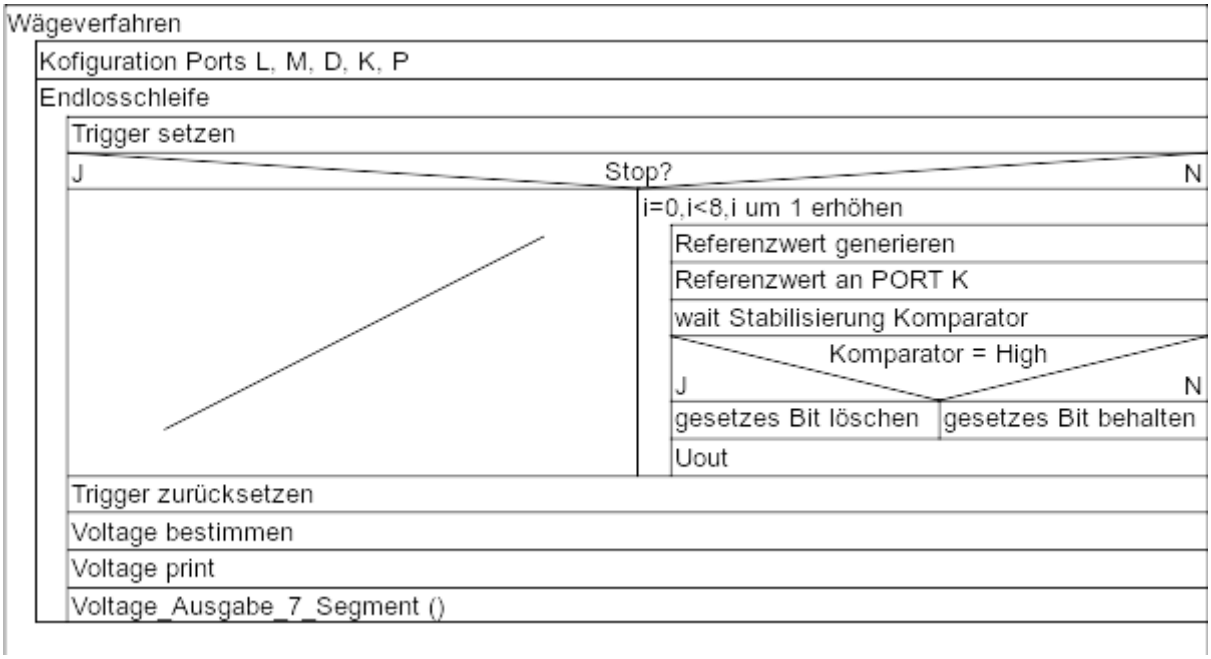

Abbildung 3: Struktogramm für Wägeverfahren

Das fertige Programm ist im Anhang unter Punkt Aufgabe 2.

Auswertung

Die Messwerte wurden in der darauffolgenden Tabelle mit Ist- & Sollwerten zusammengefasst. Das Messgerät True RMS Digital Multimeter 4150 von PeakTech wurde für die Auswertung benutzt. Man kann feststellen, dass Anhand der Tabelle1 eine Maximale Abweichung von 0,018V vorhanden ist.

IST-Wert (in V), mit Wegeverfahren	SOLL-Wert (in V), mit Multimeter
0,117	0,129
0,1952	0,213
0,292	0,303
0,410	0,414
0,507	0,506
0,996	1,010
2,031	2,038
3,164	3,172
4,062	4,075
4,550	4,564

Tabelle 1: Soll-Werte zu den abgelesenen Ist-Werten gegenübergestellt

Aufgabe 3: Interner A/D Umsetzer: Dimmen einer LED unter Zuhilfenahme eines analogen XY Joysticks

In diesem Aufgaben Teil sollte mit Hilfe des A/D Umsetzers und unter Verwendung eines XY Joysticks der momentane Spannungswert des Y Ausganges gemessen und auf der Konsole ausgegeben werden, sowohl auch mit Hilfe einer PWM Funktion und des Joysticks sollte die Helligkeit einer LED verändert werden. Zur Realisierung der PWM wurde ein Timer benutzt, welcher das Tastverhältnis einer Rechteckfrequenz im Bereich von 5% bis 95% verändern soll. Über diesen Timer wurde die Low Zeit und die High Zeit des Ausgangssignals festgelegt. Jedoch arbeitet der interne A/D Umsetzers mit 12-Bit, anstelle mit 8-Bit wie in Aufgabe 2. Somit ist dieser interne A/D Wandler Genauer.

Aufbau

XY Joystick

Die Ausgänge X und Y liefern jeweils ein Ausganssignal zwischen 0 und 4 Volt in

Abhängigkeit der Stellung des Joysticks.

Mittlere Ruhelage = 2.0 V

Hebel nach rechts = 4 V

Hebel nach links = 0 V

Taster SW in Ruhe = 0V gedrückt = 3.3 V

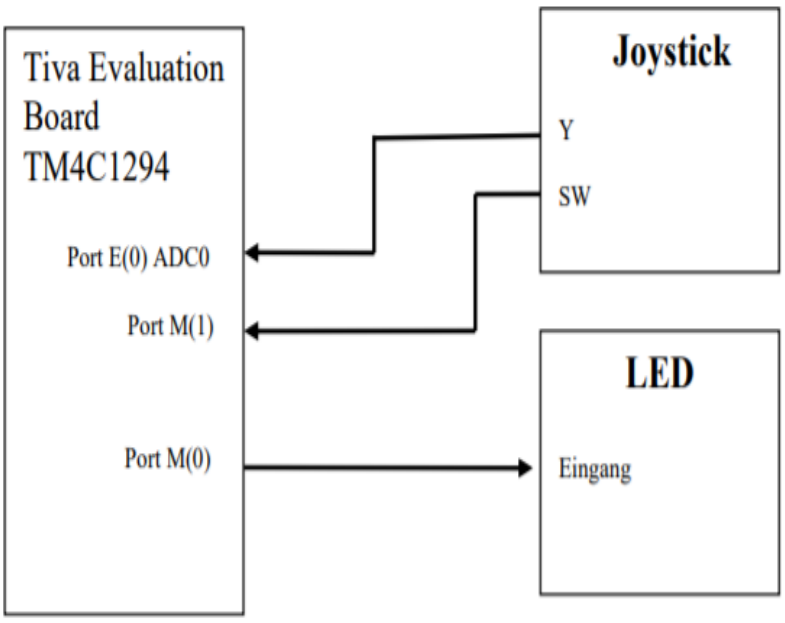

Abbildung 4: Versuchsaufbau für das Joystick aus dem Aufgabenblatt

Durchführung Aufgabe 3.1

Nachdem der ADC Wandler konfiguriert ist, kann schon der Wert abgelesen werden. Doch dieser entspricht nicht der angelegten Spannung so muss dieser Wert mit dem Faktor $5V/2^{12}$ multipliziert werden um auf die Spannung zu kommen.

Durchführung Aufgabe 3.2

Es wurde das Beispiel aus der Vorlesung für PWM Signale [2] genommen und modifiziert. So wurde der Matchwert zwischen **MAX**=48000*0.95 und **MIN**=48000*0.05 eingestellt. Um nun linear interpolieren zu können wurde eine Zahl nun hoch oder runter addiert. Wenn der Y Wert größer oder kleiner als 2V war. Wie es in Zeile 163 und 168 zu sehen ist. Damit das hochzählen nicht zu schnell erfolgt wurde eine Warteschleife hinzugefügt.

Außerdem wurde beim Knopf druck ein Boolescher Wert getoggelt um zwischen MIN und MAX hin und her zu wechseln.

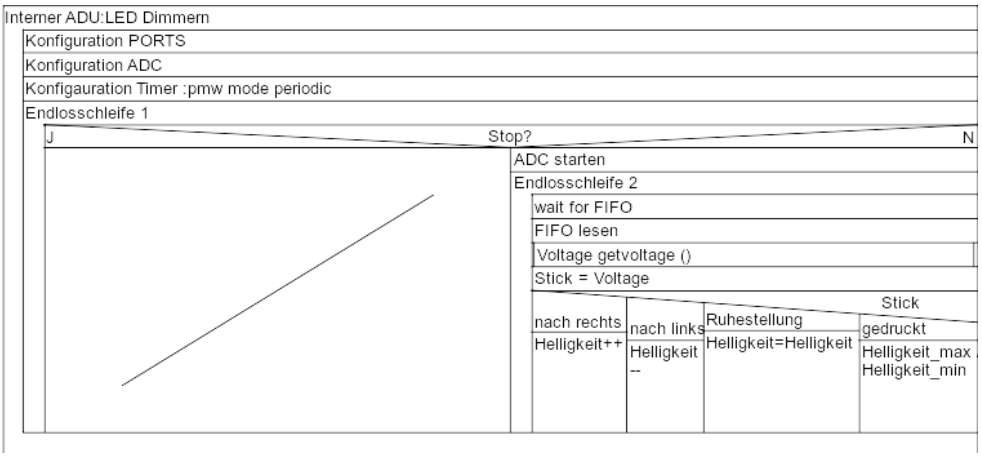

Abbildung 5: Struktogramm des Programms

Mit dem Oszilloskop wurden zwei Abbildungen eines Rechteckfrequenz im Bereich von 5% bis 95% aufgenommen. Wo über ein Matchvalue des Timers die High Zeit des Ausgangssignals festgelegt wurde.

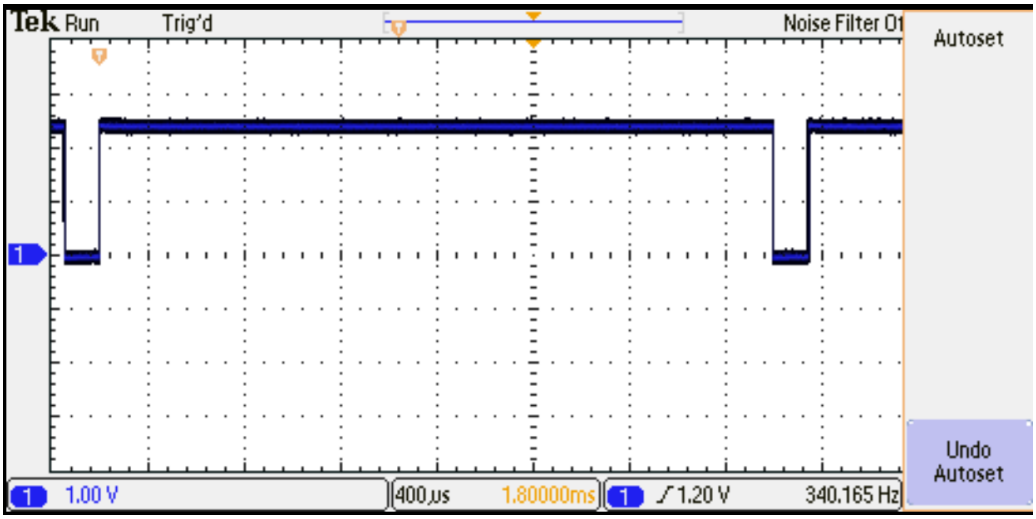

Abbildung 6: 95% High

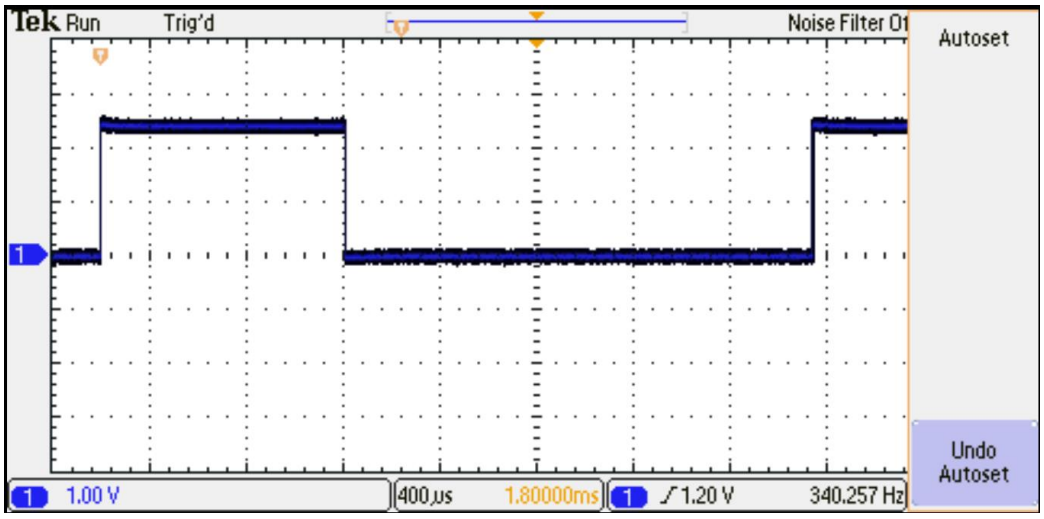

Abbildung 7: ca. 30% High

Fazit

Mit dem Wäge verfahren kann die Spannung genau, kostengünstig und schnell ermittelt werden. Außerdem kann mit einem ADC Wandler Analoge Bewegungen die zu einer Spannungsenderung sorgen in Digitale Veränderungen in diesem Fall Helligkeit einer LED umgewandelt werden. Dabei muss ein PWM Signal erzeugt werden welches durch den Timer realisiert werden kann.

Literaturverzeichnis

[1] Prof. Dr.-Ing Lutz Leutelt, " T3_Nr_1-DigVoltmeter.pdf „, Februar 2019.

[2] Prof. Dr.-Ing Lutz Leutelt, "General Purpose Timers.pdf „ Seite. 81-82, Februar 2019.

Anhang

Quellcode Aufgabe 2

Mit https://tohtml.com/cpp/ erstellt.

```
01. #include "tm4c1294ncpdt.h"
02. #include "stdio.h"

03. #define D0 (GPIO_PORTD_AHB_DATA_R & 0x01)
04. #define stop (GPIO_PORTD_AHB_DATA_R & 0x02)
05. #define portK GPIO_PORTK_DATA_R
06. #define uLSB (5000.0/256)
07. #define portM GPIO PORTM DATA R

08. //EN0 = PL0
09. //EN1 = PL1
10. //D0.. = PM0
11. //..D7 = PM7
12. //input = PD0
13. //MSB.. = PK7
14. //..LSB = PK0


15. // Warteschleife
16. void wait(unsigned long time){
17. unsigned long j;
18. for(j=0;j<time;j++);
19. }

20. //Funktion zur Ausgabe
21. void output(double value){
22. //portM = 0x00;
23. int pos1 = value/1000;
```

```
int pos2 = (value - (pos1*1000))/100;
int pos3 = (value - (pos1*1000)- (pos2*100))/10;
int pos4 = value - (pos1*1000) - (pos2*100) - (pos3*10);
GPIO_PORTL_DATA_R = 0x1; //enable EN0
        portM = (pos3 << 4) | pos4;
wait(1000);
GPIO_PORTL_DATA_R = 0x2; //enable EN1
portM = (pos1 << 4) | pos2;
wait(1000);
}

void main(void){
// configuration port D
SYSCTL_RCGCGPIO_R |= 0x0008;            //clock port D
while(!(SYSCTL_PRGPIO_R & 0x0008));     //wait for port D clock
GPIO_PORTD_AHB_DEN_R  = 0x03;           //enable Pins 1&0
GPIO_PORTD_AHB_DIR_R  = 00000000;               //Pin 1&0 => Input
GPIO_PORTD_AHB_DATA_R = 00000000;               //reset data

// configuration port K
SYSCTL_RCGCGPIO_R |= 0x0200;                //clock port K
while(!(SYSCTL_PRGPIO_R & 0x0200));         //wait for port K clock
GPIO_PORTK_DEN_R  = 0xFF;                   //enable all pins of port K
GPIO_PORTK_DIR_R  = 0xFF;                   //all output-pins
GPIO_PORTK_DATA_R = 00000000;           //reset data

// configuration port L
SYSCTL_RCGCGPIO_R |= 0x0400;            //clock port L
while((SYSCTL_PRGPIO_R & 0x0400) == 0); //wait for port L clock
GPIO_PORTL_DEN_R  = 0x07;               //enable pins 2,1,0
GPIO_PORTL_DIR_R  = 0x07;               //pins 2,1,0 => Output
GPIO_PORTL_DATA_R = 0x00000000;                 //reset data

// configuration port M
SYSCTL_RCGCGPIO_R |= 0x0800;                //clock port M
while ((SYSCTL_PRGPIO_R & 0x0800) ==0);     //wait for port M clock
GPIO_PORTM_DEN_R  = 0xFF;                   //enable all pins of port M
GPIO_PORTM_DIR_R  = 0xFF;                   //all output-pins
GPIO_PORTM_DATA_R = 00000000;               //reset data

// configuration port P
SYSCTL_RCGCGPIO_R |= 0x2000;                //clock port P
while ((SYSCTL_PRGPIO_R & 0x2000) ==0);     //wait for port P clock
GPIO_PORTP_DEN_R  = 0x01;                   //enable pins 0
GPIO_PORTP_DIR_R  = 0x01;                   //pin 0 => output
GPIO_PORTP_DATA_R = 00000000;               //reset DATA
int i = 0;
char uOut = 0;
double voltage = 0;

while(1){                                                   //Endlosschleife
       GPIO_PORTL_DATA_R |=0x04;                                        //trigger setzten
            while(!stop && !(GPIO_PORTD_AHB_DATA_R & (1<<1))){
                    uOut = 0;
for(i=0; i<=7; i++){
uOut |= (0x80 >> i);
portK = uOut;
//timer(1);
wait(55);
if(!D0){
uOut &= (~(0x80 >> i));
portK = uOut;
}
}
GPIO_PORTL_DATA_R &= ~0x04;                                 //trigger löschen
portK = 0x00;
voltage = uOut * uLSB;
//printf("U: %f\n",voltage);
output(voltage);
}

}
}
```

Quellcode Aufgabe 3

```
90. #include <stdint.h>
91. #include <stdlib.h>
92. #include <stdio.h>
93. #include <stdbool.h>
94. #include <math.h>
95. #include "tm4c1294ncpdt.h"

96. #define uLSB (5000.0/4096)
97. #define portM GPIO_PORTM_DATA_R

98. #define MIN_TIME 48000*0.05
99. #define MAX_TIME 48000*0.90

100.  // Warteschleife
101.  void wait(unsigned long time){
102.  unsigned long j;
103.  for(j=0;j<time;j++);
104.  }

105.  //Funktion zur Ausgabe
106.  void output(double value){
107.  //portM = 0x00;
108.  int pos1 = value/1000;
109.  int pos2 = (value - (pos1*1000))/100;
110.  int pos3 = (value - (pos1*1000)- (pos2*100))/10;
111.  int pos4 = value - (pos1*1000) - (pos2*100) - (pos3*10);
112.  GPIO_PORTL_DATA_R = 0x1; //enable EN0
113.  portM = (pos3 << 4) | pos4;
114.  wait(1000);
115.  GPIO_PORTL_DATA_R = 0x2; //enable EN1
116.  portM = (pos1 << 4) | pos2;
117.  wait(1000);
118.  }

119.  double getVoltage()
120.  {
121.  unsigned long ADCvalue = 0;

122.  ADC0_PSSI_R |= 0x01; //start ADC
123.  while(ADC0 SSFSTAT0 R & (1<<8)); //warten solange FIFO leer
124.  ADCvalue = (unsigned long) ADC0_SSFIFO0_R;
125.  return ADCvalue*uLSB+0.5;
126.  }

127.  void tast31()
128.  {
129.  while(1){
130.  printf("%f\n", getVoltage());
131.  wait(1000000);
132.  }
133.  }

134.  void tast32()
135.  {
136.  // configuration port D
137.  SYSCTL_RCGCGPIO_R |= 0x0008;                    //clock port D
138.  while(!(SYSCTL PRGPIO R & 0x0008));             //wait for port D clock
139.  GPIO_PORTD_AHB_DEN_R  |= 0x07;                  //enable Pins 1
140.  GPIO_PORTD_AHB_AFSEL_R  |= 0x01;
141.  GPIO_PORTD_AHB_PCTL_R  = 0x03;

142.  //configure Timer 0
143.  SYSCTL_RCGCTIMER_R |= (1<<0);                   //timer 0
144.  while(!(SYSCTL_PRTIMER_R) & (1<<0));            //wait for timer 0 activation
145.  TIMER0_CTL_R &= ~0x01;                          //disable Timer 0
146.  TIMER0_CFG_R = 0x04;                            //2 x 16-bit mode

147.  //compare mode, down
148.  TIMER0_TAMR_R |= (1<<3)                         //TAAMS: PWM - mode
149.  | 0x02;                                                    //TAMR: periodic
150.  TIMER0_CTL_R |= (1<<6);
151.  TIMER0_TAILR_R = 48000-1;
152.  TIMER0_TAMATCHR_R = 32000-1;
153.  GPIO_PORTD_AHB_DATA_R |= (1<<1);
154.  TIMER0_CTL_R |= 0x01;
```

```
155.  unsigned char count = 128;
156.  bool togell = false;
157.  bool down = false;

158.  while(1){
159.  double stickY = getVoltage();
160.  if(stickY > 2500 && count != 255)
161.  {
162.  count++;

163.  TIMER0_TAMATCHR_R = MIN_TIME + (MAX_TIME/255) * count;
164.  }
165.  else if(stickY < 1500 && count != 0)
166.  {
167.  count--;

168.  TIMER0_TAMATCHR_R = MIN_TIME + (MAX_TIME/255) * count;
169.  }
170.  else if(GPIO_PORTD_AHB_DATA_R & (1<<2) && !down)
171.  {
172.  togell = !togell;
173.  down = true;

174.  if(togell)
175.  {
                                      i.  TIMER0_TAMATCHR_R = MIN_TIME + MAX_TIME;
176.  }
177.  else
178.  {
                                     ii.  TIMER0_TAMATCHR_R = MIN_TIME;
179.  }
180.  }
181.  else
182.  {
183.  down = false;
184.  wait(100000);
185.  }



186.  wait(10000);
187.  }
188.  }

189.  void main(void){
190.  //ADC0
191.  SYSCTL_RCGCGPIO_R |= (1<<4);               // PE (AIN3 belong to Port E)
192.  while(!(SYSCTL_PRGPIO_R & 0x10));          // Ready ?
193.  SYSCTL_RCGCADC_R |= (1<<0);                // ADC0 digital block
194.  while(!(SYSCTL_PRADC_R & 0x01));           // Ready ?
195.  // configure AIN3 (= PE(0)) as analog inputs
196.  GPIO_PORTE_AHB_AFSEL_R |= 0x01;                  // PE0 Alternative Pin
    Functionenable
197.  GPIO_PORTE_AHB_DEN_R &= ~0x01;                  // PE disabledigital IO
198.  GPIO_PORTE_AHB_AMSEL_R |= 0x01;                 // PE enable analog function
199.  GPIO_PORTE_AHB_DIR_R &= ~0x01;                  // AllowInput

200.  // ADC0_SS0 configuration
201.  //magic code
202.  ADC0_ACTSS_R &= ~0x0F; // disable all 4 sequencers of ADC0
203.  SYSCTL_PLLFREQ0_R |= (1<<23); // PLL Power
204.  while (!(SYSCTL_PLLSTAT_R & 0x01)); // until PLL has locked
205.  ADC0_CC_R |= 0x01;  // PIOSC für analog block
206.  SYSCTL_PLLFREQ0_R &= ~(1<<23); // PLL Power off
207.  // end of magic code
208.  ADC0 SSMUX0 R |= 0x0000003; // sequencer0, channel AIN3
209.  ADC0_SSCTL0_R |= 0x000002; // END2 set, sequence length = 0
210.  ADC0_ACTSS_R |= 0x01; // enable sequencer 0 ADC0

211.  //tast31();
212.  tast32();
213. }
```