|  |
| --- |
| Aufgabe Reflow Ofen |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Änderung | Name | Datum |
| 1.0 | Erstellung | Kenneth Mathis | 30. September 2019 |

Inhalt

[1 Aufgabenstellung 3](#_Toc20231418)

[2 Hardware-Struktur 3](#_Toc20231419)

[2.1 Systembeschreibung 3](#_Toc20231420)

[2.1.1 Beschaltung der Hardware 3](#_Toc20231421)

[2.1.2 Inputs 4](#_Toc20231422)

[2.1.3 Outputs 4](#_Toc20231423)

[2.1.4 Steuersignale & LED Positionen 4](#_Toc20231424)

[3 Software-Analyse 5](#_Toc20231425)

[3.1 Struktogramm zum Hauptprogramm 5](#_Toc20231426)

[3.2 Struktogramm zur Flankenerkennung 5](#_Toc20231427)

[3.3 Ramp Heat 6](#_Toc20231428)

[3.4 Heat Up 6](#_Toc20231429)

[3.5 Ramp Reflow 7](#_Toc20231430)

[3.6 Reflow 7](#_Toc20231431)

[3.7 Cooldown 8](#_Toc20231432)

[4 Source-Code 8](#_Toc20231433)

[4.1 Hauptprogramm 8](#_Toc20231434)

[4.2 Matrix Lib 12](#_Toc20231435)

[4.2.1 matrix.h 12](#_Toc20231436)

[4.2.2 matrix.c 12](#_Toc20231437)

[4.3 7Segment Lib 13](#_Toc20231438)

[4.3.1 sevenSegDisp.h 13](#_Toc20231439)

[4.3.2 sevenSegDisp.c 13](#_Toc20231440)

[4.4 ADC Lib 15](#_Toc20231441)

[4.4.1 adc.h 15](#_Toc20231442)

[4.4.2 adc.c 16](#_Toc20231443)

[4.5 Convert Lib 17](#_Toc20231444)

[4.6 Flankenerkennung Lib 18](#_Toc20231445)

[4.6.1 edge.h 18](#_Toc20231446)

[4.6.2 edge.c 18](#_Toc20231447)

[4.7 RGB Lib 19](#_Toc20231448)

[4.7.1 RGB.h 19](#_Toc20231449)

[4.7.2 RGB.c 19](#_Toc20231450)

[5 Testprotokoll 20](#_Toc20231451)

[5.1 Testablauf 20](#_Toc20231452)

[5.2 Testbericht mit Fazit und Massnahmen 21](#_Toc20231453)

# Aufgabenstellung

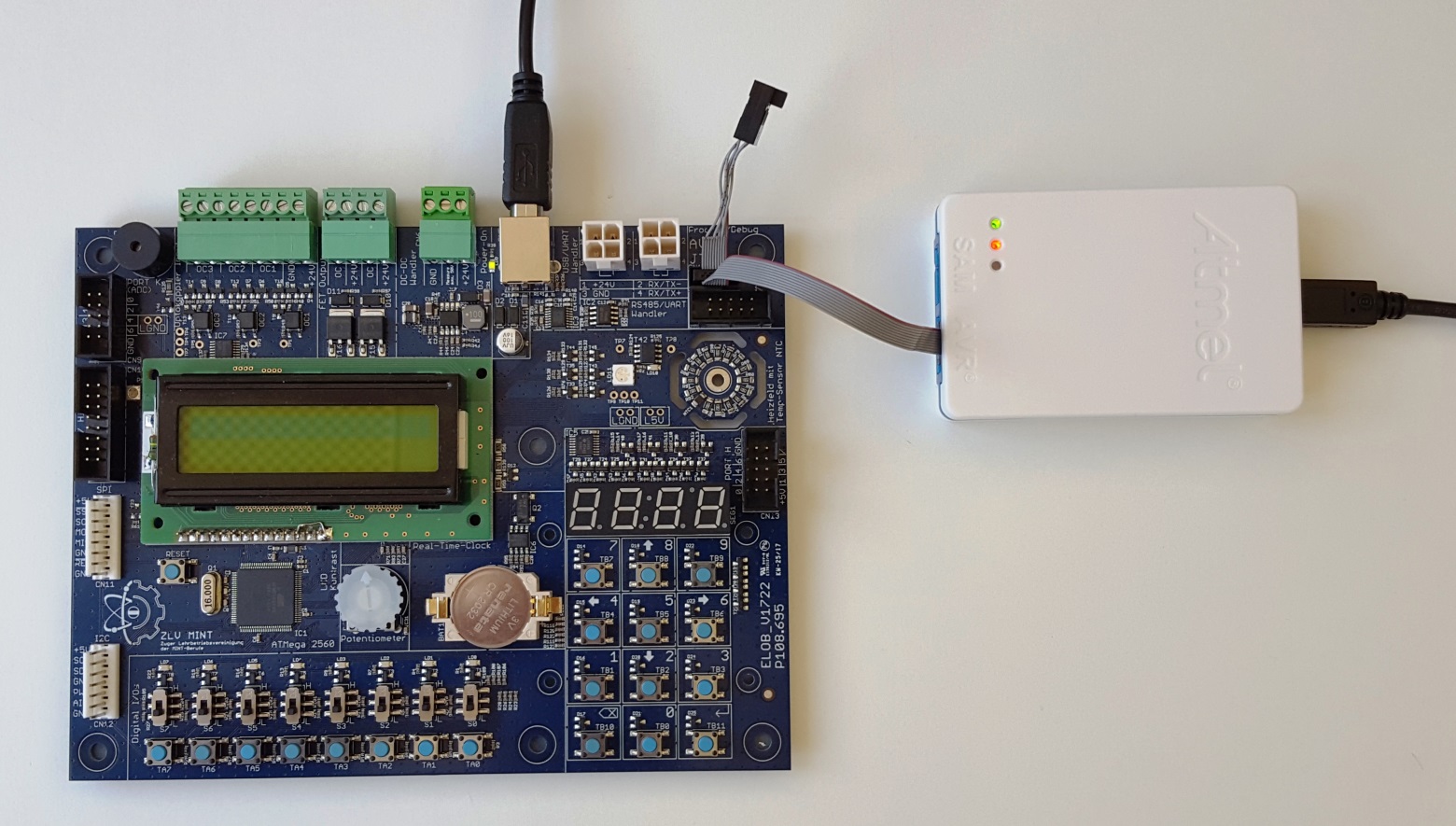
Es soll ein Programm zur Simulierung der Funktionsweise eines Reflow-Ofens erstellt werden.

* ***Weitere Details sind der Original-Aufgabenstellung zu entnehmen!***

# Hardware-Struktur

## Systembeschreibung

Es wird das ZLV-MINT Entwicklungs Board mit dem Mikrocontroller Atmel ATmega2560 verwendet, welches für die Programmierung sowie das Debugging mit dem Atmel-ICE-Debugger betrieben wird (per JTAG Schnittstelle).



### Beschaltung der Hardware

Die Ports für Taster, Schalter, LEDs und weitere Module sind direkt und fix auf dem Board verbunden.  
Bestimmte weitere Peripherie kann extern über entsprechende Anschlussleisten oder Stecker/Buchsen angeschlossen werden.

**Logik für Taster, Schalter und LEDs**

Taster 1 = gedrückt 0 = nicht gedrückt

Schalter 1 = Schalterposition oben 0 = Schalterposition unten

LED 1 = leuchtet 0 = dunkel

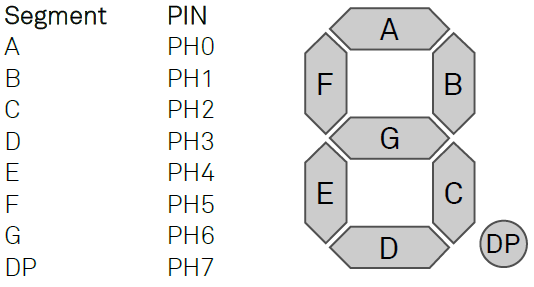
### Inputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bezeichner** | **uC-Pin** | **Bauteil** | **Logikpegel-Definitionen** | **Grobe Funktionsbeschreibung** |
| startTaster | PJ0 | TA0 | 1 = Gedrückt | Taster für Betriebsstart |
| stopTaster | PJ1 | TA1 | 1 = Gedrückt | Taster Stop |

### Outputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bezeichner** | **uC-Pin** | **Bauteil** | **Logikpegel-Definitionen** | **Grobe Funktionsbeschreibung** |
| ledRed | PB5 | LD11 | 1 = Leuchtet | Rote RGB-LED |
| ledGreen | PB6 | Grüne RGB-LED |
| ledBlue | PB7 | Blaue RGB-LED |
| enSevenSeg | PG2 | IC5A | 1 = Enabled | 7-Segment Enable Signal |
| segSel1 | PG0 | Siehe Tabelle | 7-Segment Select Signal |
| segSel2 | PG1 |
| segA | PH0 | SEG1 | 1 = Leuchtet | 7-Segment LED Signal (Siehe Schema) |
| segB | PH1 |
| segC | PH2 |
| segD | PH3 |
| segE | PH4 |
| segF | PH5 |
| segG | PH6 |
| segDP | PH7 |

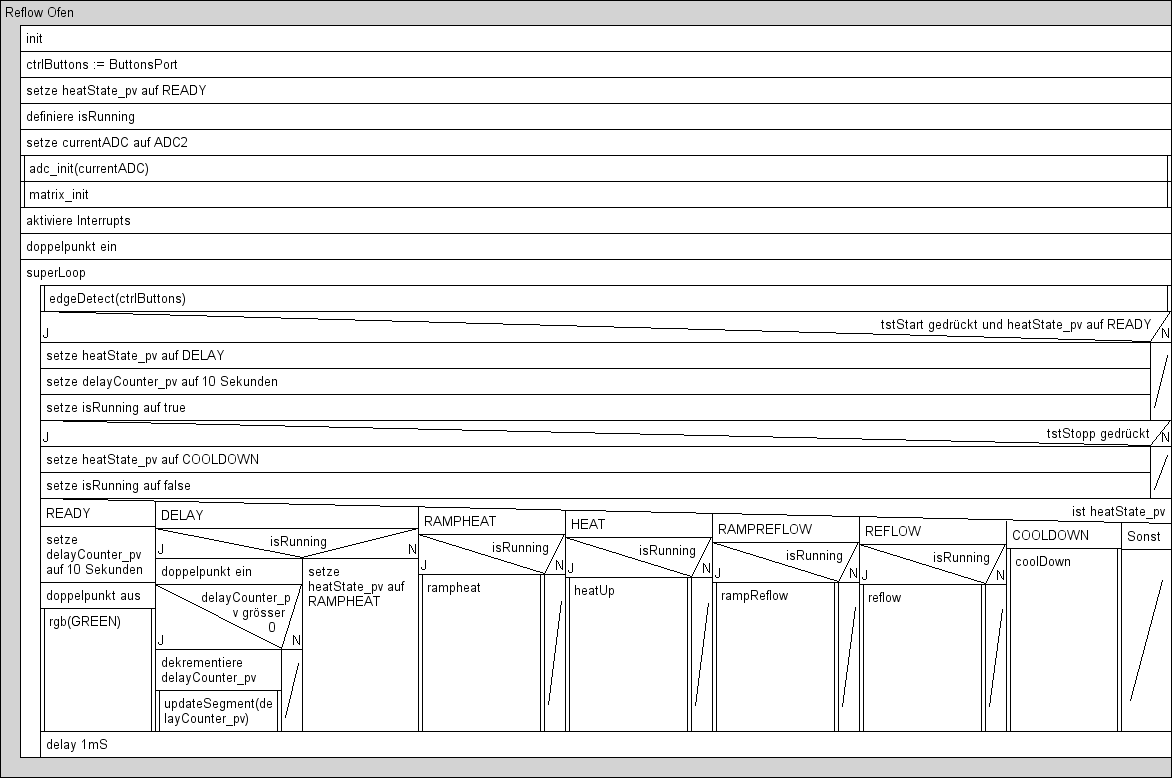
### Steuersignale & LED Positionen



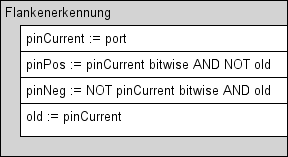
|  |  |  |
| --- | --- | --- |
| **PG1** | **PG0** | **Segment** |
| 0 | 0 | 00:0**0** |
| 0 | 1 | 00:**0**0 |
| 1 | 0 | 0**0**:00 |
| 1 | 1 | **0**0:00 |

# Software-Analyse

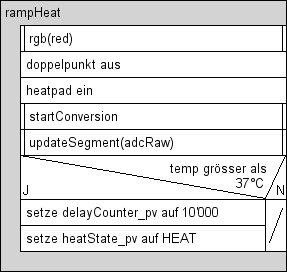
## Struktogramm zum Hauptprogramm



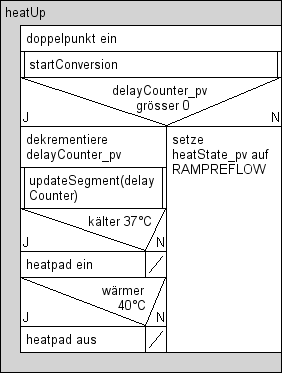
## Struktogramm zur Flankenerkennung



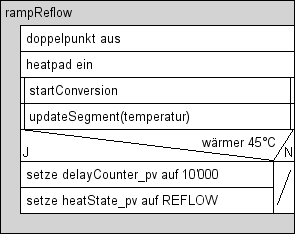
## Ramp Heat



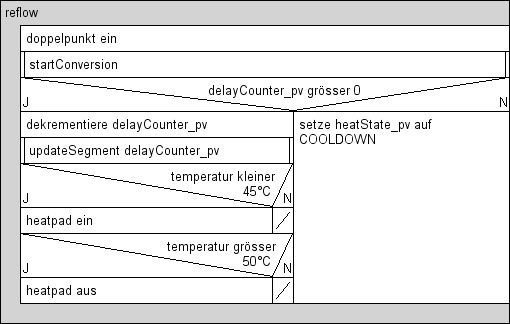
## Heat Up



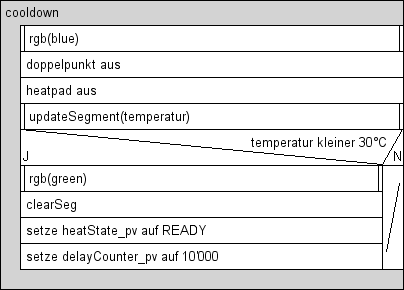
## Ramp Reflow



## Reflow



## Cooldown



# Source-Code

## Hauptprogramm

/\*\*

\* @file main.c

\* @author Kenneth Mathis

\* @version 1.0

\* @brief Program for simulating a reflow oven's workflow

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 17.09.19 1.0 Kenneth Mathis Created

\*/

//\*\*\*INCLUDES\*\*\*

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include "matrix.h"

#include "sevenSegDisp.h"

#include "adc.h"

#include "convert.h"

#include "edge.h"

#include "RGB.h"

//\*\*\*DEFINES\*\*\*

//\*\*Control Buttons\*\*

#define tstStart\_pos (ctrlButtons.pinPos & (1<<0))

#define tstStop\_pos (ctrlButtons.pinPos & (1<<1))

//\*\*Heatpad\*\*

#define heatPad\_on (PORTB |= (1<<4))

#define heatPad\_off (PORTB &= ~(1<<4))

//\*\*\*GLOBAL VARIABLES\*\*\*

adc\_t currentADC;

enum heatingStates {READY, DELAY, RAMPHEAT, HEAT, RAMPREFLOW, REFLOW, COOLDOWN};

enum heatingStates heatState\_pv;

*uint16\_t* delayCounter\_pv;

//\*\*Predefined Colours\*\*

color red = {1,0,0};

color green = {0,1,0};

color blue = {0,0,1};

//\*\*\*PROTOTYPES\*\*\*

void rampHeat(void);

void heatUp(void);

void rampReflow(void);

void reflow(void);

void coolDown(void);

/\*\*

\* @brief Main routine, this is where the magic happens.

\* @param void

\*/

int main(void){

//\*\*VARIABLES\*\*

edge ctrlButtons = {0,0,&PINJ}; //Struct for edge detection

heatState\_pv = READY; //Variable to store the current process state

*uint8\_t* isRunning;

//\*\*Init\*\*

currentADC.usingADC = ADC2;

adc\_init(&currentADC);

matrix\_init();

sei();

colon\_on;

//\*\*Data Direction Registers\*\*

DDRB = 0x10; //Set the 4th bit for the heatpad to output

//\*\*\*SUPER LOOP\*\*\*

while (1){

//\*\*Control button detection\*\*

edgeDetect(&ctrlButtons);

if(tstStart\_pos && heatState\_pv == READY){

if(heatState\_pv == READY){

heatState\_pv = DELAY;

delayCounter\_pv = 10000; //Set a 10s Countdown

isRunning = 1;

}

}

if(tstStop\_pos){

heatState\_pv = COOLDOWN;

isRunning = 0;

}

switch(heatState\_pv){

case READY:

delayCounter\_pv = 10000; //Use a 10s delay on startup if the heatpad has cooled

colon\_off;

rgb(green);

break;

case DELAY: //Countdown from 10s

if(isRunning){

colon\_on;

if(delayCounter\_pv > 0){

delayCounter\_pv--;

updateSegment((*uint16\_t*)(delayCounter\_pv/10), 4);

}

else heatState\_pv = RAMPHEAT;

}

break;

case RAMPHEAT:

if(isRunning) rampHeat();

break;

case HEAT:

if(isRunning) heatUp();

break;

case RAMPREFLOW:

if(isRunning) rampReflow();

break;

case REFLOW:

if(isRunning) reflow();

break;

case COOLDOWN:

coolDown();

break;

default:

break;

}

//\*\*\*GLOBAL DELAY\*\*\*

*\_delay\_ms*(1);

}

}

/\*\*

\* @brief Pre-Heat the reflow oven

\* @param void

\*/

void rampHeat(void){

rgb(red);

colon\_off;

heatPad\_on;

startConversion();

updateSegment(((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)\*100)), 2);

//Temperature over 35°C go to the the heating phase

if((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)) >= 37){

delayCounter\_pv = 10000; //Use a 10s heating period

heatState\_pv = HEAT;

}

}

/\*\*

\* @brief Heat the PCB

\* @param void

\*/

void heatUp(void){

colon\_on;

startConversion();

if(delayCounter\_pv > 0){

delayCounter\_pv--;

updateSegment((*uint16\_t*)(delayCounter\_pv/10), 4);

//Temperature holding state

if((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)) < 37) heatPad\_on;

if((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)) > 40) heatPad\_off;

}

else heatState\_pv = RAMPREFLOW;

}

/\*\*

\* @brief Ramp up to reflow temperature

\* @param void

\*/

void rampReflow(void){

colon\_off;

heatPad\_on;

startConversion();

updateSegment(((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)\*100)), 2);

if((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)) >= 45){

delayCounter\_pv = 10000; //Use a 10s reflow period

heatState\_pv = REFLOW;

}

}

/\*\*

\* @brief Reflow dat board

\* @param void

\*/

void reflow(void){

colon\_on;

startConversion();

if(delayCounter\_pv > 0){

delayCounter\_pv--;

updateSegment((*uint16\_t*)(delayCounter\_pv/10), 4);

//Temperature holding state

if((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)) < 45) heatPad\_on;

if((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)) > 50) heatPad\_off;

}

else heatState\_pv = COOLDOWN;

}

/\*\*

\* @brief Cool down everything

\* @param void

\*/

void coolDown(void){

rgb(blue);

colon\_off;

heatPad\_off;

static *uint16\_t* adcClearOffset; //ADC reads garbage after reflow part, this counteracts the problem

updateSegment(((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)\*100)), 2);

if((*uint16\_t*)(toTemeperatuere(currentADC.adcRaw)) >= 30 || adcClearOffset <= 10000){

rgb(blue);

heatState\_pv = COOLDOWN;

adcClearOffset++;

}

else{

rgb(green);

clearSeg;

heatState\_pv = READY;

delayCounter\_pv = 10000; //Use a 10s delay on startup if the heatpad has cooled

}

}

//\*\*\*INTERRUPTS\*\*\*

/\*\*

\* @brief Interrupt for ADC-Conversion complete

\* @param void

\*/

ISR(ADC\_vect){

currentADC.adcRaw = ADC;

}

## Matrix Lib

### matrix.h

/\*\*

\* @file matrix.h

\* @brief LIB to control the mux and enable signal to the seven segment display and keypad.

\* @author Kenneth Mathis

\* @version 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 09.09.19 1.0 Kenneth Mathis Created

\*/

#ifndef MATRIX\_H

#define MATRIX\_H

//Enable and Disable the segment and keypad

#define mtxEnable (PORTG |= (1<<2))

#define mtxDisable (PORTG &= ~(1<<2))

//Mux signals for selecting a segment/keypad row

#define segSel0\_off (PORTG &= ~(1<<0))

#define segSel1\_off (PORTG &= ~(1<<1))

#define segSel0\_on (PORTG |= (1<<0))

#define segSel1\_on (PORTG |= (1<<1))

//\*\*\*PROTOTYPES\*\*\*

/\*\*

\* @brief Preps the seven segment display and keypad for use.

\* @param Void

\*/

void matrix\_init(void);

#endif

### matrix.c

/\*\*

\* @file matrix.c

\* @brief LIB to control the mux and enable signal to the seven segment display and keypad.

\* @author Kenneth Mathis

\* @version 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 09.09.19 1.0 Kenneth Mathis Created

\*/

#include <avr/io.h>

#include "matrix.h"

void matrix\_init(void){

//\*\*Data Direction Registers\*\*

//\*1=Output\*

DDRH = 0xff; //DDR of the individual segments

DDRG |= 0x1f; //DDR of the multiplexing/control signals

DDRD &= 0x70; //DDR of the keypad read pins

mtxEnable; //Enable the matrix

}

## 7Segment Lib

### sevenSegDisp.h

/\*\*

\* @file sevenSegDisp.h

\* @brief LIB to control the seven segment display.

\* @author Kenneth Mathis

\* @version 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 09.09.19 1.0 Kenneth Mathis Created

\* 24.09.19 1.1 Kenneth Mathis Added clearSeg

\*/

#ifndef SEVENSEGDISP\_H

#define SEVENSEGDISP\_H

#define colon\_on (PORTG |= (1<<4))

#define colon\_off (PORTG &= ~(1<<4))

#define apostrophe\_on (PORTG |= (1<<3))

#define apostrophe\_off (PORTG &= ~(1<<3))

#define segEnable (PORTG |= (1<<2))

#define segDisable (PORTG &= ~(1<<2))

#define clearSeg (PORTH = 0)

/\*\*

\* @brief Update the seven segment display with a number

\* Note: This function takes care of the incremeting

\* so no individual segment can be set.

\* @param segNumber The Number to display on the segment.

\* @param dotPos The position of the dot. Set it to 4 or higher for no dot.

\*/

void updateSegment(unsigned int segNumber, *uint8\_t* dotPos);

#endif

### sevenSegDisp.c

/\*\*

\* @file sevenSegDisp.c

\* @brief LIB to control the seven segment display.

\* @author Kenneth Mathis

\* @version 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 09.09.19 1.0 Kenneth Mathis Created

\*/

//\*\*\*INCLUDES\*\*\*

#include <avr/io.h>

#include <math.h>

#include "sevenSegDisp.h"

#include "matrix.h"

//\*\*\*DEFINES\*\*\*

#define setSeg(bit) (PORTH |= (bit))

#define delSeg(bit) (PORTH &= (bit))

//\*\*\*PROTOTYPES\*\*\*

*uint8\_t* numberToBitmask(*uint8\_t* num, *uint8\_t* dotted);

void updateSegment(unsigned int segNumber, *uint8\_t* dotPos){

static *uint8\_t* segPos;

segDisable;

//\*\*Select which segment is currently being displayed\*\*

switch(segPos){

case 0:

segSel0\_off;

segSel1\_off;

break;

case 1:

segSel0\_on;

segSel1\_off;

break;

case 2:

segSel0\_off;

segSel1\_on;

break;

case 3:

segSel0\_on;

segSel1\_on;

break;

default:

segSel0\_off;

segSel1\_off;

break;

}

//\*\*Create the bitmask for the given number and set the PORT to it\*\*

*uint8\_t* bitmask;

if(segNumber == 0){

bitmask = numberToBitmask(0, 0);

} //If the given number is zero, display it as such

else{

*uint8\_t* segDigit;

segDigit = ((*uint8\_t*)(segNumber/*pow*(10, segPos)))%10; //Get the digit at segPos from segNumber

if(dotPos == segPos) //Determine if the current segment is dotted

bitmask = numberToBitmask(segDigit, 1);

else bitmask = numberToBitmask(segDigit, 0);

}

setSeg(bitmask);

delSeg(bitmask);

segEnable;

if(segPos < 3) segPos++; //increment the segment position till the last segment

else segPos = 0; //set the segment position to zero if its past the last one

}

/\*\*\*NUMBER TO BITMASK\*\*\*

\* @brief Turns a number into a seven segment bitmask

\* @param: num the number to turn into a 7-segment bitmask

\* @param: dotted If the wanted number should be dotted

\* @return: the bitmask of the given number

\*/

*uint8\_t* numberToBitmask(*uint8\_t* num, *uint8\_t* dotted){

switch(num){

case 0:

return 0b00111111 | dotted<<7;

case 1:

return 0b00000110 | dotted<<7;

case 2:

return 0b01011011 | dotted<<7;

case 3:

return 0b01001111 | dotted<<7;

case 4:

return 0b01100110 | dotted<<7;

case 5:

return 0b01101101 | dotted<<7;

case 6:

return 0b01111101 | dotted<<7;

case 7:

return 0b00000111 | dotted<<7;

case 8:

return 0b01111111 | dotted<<7;

case 9:

return 0b01101111 | dotted<<7;

default:

return 0b01000000 | dotted<<7;

}

}

## ADC Lib

### adc.h

/\*\*

\* @file adc.h

\* @brief LIB to control the ADC on the ZLV-Mint Board

\* @author Kenneth Mathis

\* @version 1.1

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 09.09.19 1.0 Kenneth Mathis Created

\* 17.09.19 1.1 Kenneth Mathis Added interrupts

\*

\*/

//\*\*\*FILE GUARD\*\*\*

#ifndef ADC\_H

#define ADC\_H

//A list of all the single ended input ADC's and their mux "id"

enum tblADC {ADC0=0,ADC1,ADC2,ADC3,ADC4,ADC5,ADC6,ADC7,ADC8=32,ADC9,ADC10,ADC11,ADC12,ADC13,ADC14,ADC15};

//A struct for storing all the adc revlevant data

typedef struct{

enum tblADC usingADC;

*uint16\_t* adcRaw;

*uint8\_t* isLeftAdjusted;

*uint8\_t* is8BitMode;

} adc\_t;

//\*\*\*PROTOTYPES\*\*\*

/\*\*

\* @brief Initializes the ADC for use and defines the wanted MUX config.

\* @param theADC An adc\_t stcut is used to determine all the init values.

\*/

void adc\_init(adc\_t \*theADC);

/\*\*

\* @brief Start the conversion when an interrupt is used

\* @param Void

\*/

void startConversion(void);

/\*\*

\* @brief Reads the current value on the ADC value and stores its via "reference".

\* @param theADC adc\_t struct for storing the adcRaw into.

\*/

void getValue(adc\_t \*theADC);

#endif

### adc.c

/\*\*

\* @file adc.c

\* @brief LIB to control the ADC on the ZLV-Mint Board

\* @author Kenneth Mathis

\* @version 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 09.09.19 1.0 Kenneth Mathis Created

\* 17.09.19 1.1 Kenneth Mathis Added interrupts

\*

\*/

//\*\*\*INCLUDES\*\*\*

#include <avr/io.h>

#include "adc.h"

void adc\_init(adc\_t \*theADC){

ADCSRA |= (1<<7); //Enable ADC

ADCSRA |= (1<<3); //Enable ADC conversion complete interrupts

if(theADC->isLeftAdjusted || theADC->is8BitMode) ADMUX |= (1<<5); //Produce left adjusted result (8-Bit Mode)

ADMUX |= (theADC->usingADC & 0x1f); //Set the mux to the right DAC, Note: The mux Bits are in two Registers!

ADCSRB |= ((theADC->usingADC>>1) & 0x08);

}

void startConversion(void){

ADCSRA |= (1<<6); //Start the adc conversion

}

void getValue(adc\_t \*theADC){

ADCSRA |= (1<<6); //Start the adc conversion

while(!(ADCSRA & 0x10)); //Wait till the conversion has ended

if(theADC->is8BitMode) theADC->adcRaw = ADCH; //Return an 8-Bit value if wanted

else theADC->adcRaw = ADC;

}

## Convert Lib

/\*\*

\* @file convert.c

\* @brief LIB to covert raw ADC values to more useful stuff

\* @author Kenneth Mathis

\* @version 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 17.09.19 1.0 Kenneth Mathis Created

\*

\*/

//\*\*\*INCLUDES\*\*\*

#include "convert.h"

//\*\*\*FUNCTIONS\*\*\*

float toTemeperatuere(*uint16\_t* adcValue){

//Linear Temperature conversion

//float temp;

//temp = (26.32\*toVoltage(adcValue))-40.8;

//Conversion via Simplified Steinhart-Hart B-Parameter Equation

//See: https://learn.adafruit.com/thermistor/using-a-thermistor

float steinhart;

steinhart = (toResistance(adcValue))/10000.0; // (R/Ro)

steinhart = *log*(steinhart); // ln(R/Ro)

steinhart /= 3510; // 1/B \* ln(R/Ro)

steinhart += 1.0 / (25.0 + 273.15); // + (1/To)

steinhart = 1.0 / steinhart; // Invert

steinhart -= 273.15; // convert to C

return steinhart;

}

float toVoltage(*uint16\_t* adcValue){

float volt;

volt = adcValue\*0.0048875; //voltage = adcVal\*(5V/1023)

return volt;

}

float toResistance(*uint16\_t* adcValue){

float resistance;

resistance = (50000.0/toVoltage(adcValue))-10000; //resistance = (Vcc\*R2/Vadc)-R2

return resistance;

}

## Flankenerkennung Lib

### edge.h

/\*\*

\* @file edge.h

\* @brief LIB to detect positive and negative edges

\* @author Kenneth Mathis

\* @version 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 17.09.19 1.0 Kenneth Mathis Created

\*

\*/

#ifndef EDGE\_H

#define EDGE\_H

//\*\*\*INCLUDES\*\*\*

#include <avr/io.h>

//\*\*\*STRUCT DEFINITIONS\*\*\*

typedef struct{ //A struct to store the edge detection data

*uint8\_t* pinPos; //Positive edge

*uint8\_t* pinNeg; //Negative edge

*uint8\_t* \*port; //The port to do edge detection on

*uint8\_t* old; //Previous value of the port

} edge;

//\*\*\*PROTOTYPES\*\*\*

/\*\*

\* @brief: A function used to detect edges on a port

\* @param: An edge struct to stores all the necessary stuff from the edge detection

\* @return: Return via "reference"

\*/

void edgeDetect(edge \*port);

#endif

### edge.c

/\*\*

\* @file edge.c

\* @brief LIB to detect positive and negative edges

\* @author Kenneth Mathis

\* @version 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 17.09.19 1.0 Kenneth Mathis Created

\*

\*/

//\*\*\*INCLUDES\*\*\*

#include "edge.h"

void edgeDetect(edge \*port){

*uint8\_t* pinjCurrent = \*(port->port);

port->pinPos = pinjCurrent & ~port->old;

port->pinNeg = ~pinjCurrent & port->old;

port->old = pinjCurrent;

}

## RGB Lib

### RGB.h

/\*\*

\* @file RGB.h

\* @brief LIB to control the RGB-LED on the ZLV-Mint Board

\* @author Kenneth Mathis

\* @version: 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 17.09.19 1.0 Kenneth Mathis Created

\*

\*/

#ifndef RGB\_H

#define RGB\_H

//\*\*\*INCLUDES\*\*\*

#include <avr/io.h>

//\*\*\*STRUCT DEFINITIONS\*\*\*

typedef struct{

*uint8\_t* r;

*uint8\_t* g;

*uint8\_t* b;

} color;

//\*\*\*PROTOTYPES\*\*\*

/\*\*

\* @brief Set the RGB-LED's color based on an color struct.

\* @param newColor Color struct to represent the led's color.

\*/

void rgb(color newColor);

#endif

### RGB.c

/\*\*

\* @file RGB.c

\* @brief LIB to control the RGB-LED on the ZLV-Mint Board

\* @author Kenneth Mathis

\* @version 1.0

\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Date Vers. Name Comment/Change

\* 17.09.19 1.0 Kenneth Mathis Created

\*

\*/

#include "RGB.h"

// \*\*\*RGB LED\*\*\*

void rgb(color newColor){

PORTB |= (newColor.r<<5)|(newColor.g<<6)|(newColor.b<<7);

//Because a 0 turns anything off with an "and-set", its necessary to set every bit outside of

//the RGB bits to 1. Thats why there is a |0x8f (|0b1000 1111) at the end.

PORTB &= (newColor.r<<5)|(newColor.g<<6)|(newColor.b<<7)|0x1f;

}

# Testprotokoll

## Testablauf

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Testfall** | **Vorgang** | **Erwartung** | **Befund** |
| 01 | Startup | Speisung Einschalten | RGB-LED Leuchtet grün | Passed |
| 02 | Taster Start gedrückt | Taster Start drücken | Segmentanzeige zählt von 10 im 10mS-Takt herunter. | Passed |
| 03 | Timer Auslauf | Timer auslaufen lassen | * Segmentanzeige ändert auf aktuelle Temperatur * Heizfeld erwärmt sich. * RGB LED wechselt auf Rot | Passed |
| 04 | Heizfeld erwärmt sich auf 37-40°C | Heizfeld erwärmen lassen | Temperatur steigt auf 37-40°C. | Passed |
| 05 | Neuer Timer Startet | Timer auslaufen lassen | Ab 37°C startet ein neuer Timer vom 10s auf der Segmentanzeige. | Passed |
| 06 | Temperatur halten | Temperatur halten | Die Temperatur wird zwischen 37-40°C gehalten | Passed |
| 07 | Timer Auslauf | Timer auslaufen lassen. | * Segmentanzeige ändert auf aktuelle Temperatur * Heizfeld erwärmt sich | Passed |
| 08 | Heizfeld erwärmt sich auf 45-47°C | Heizfeld wärmen lassen | Temperatur steigt auf 45-50°C. | Passed |
| 09 | Neuer Timer Startet | Timer auslaufen lassen | Ab 45°C startet ein neuer Timer vom 10s auf der Segmentanzeige. | Passed |
| 10 | Timer Auslauf | Timer auslaufen lassen | * RGB-LED wechselt auf blau * Segmentanzeige zeigt Temperatur | Passed |
| 11 | Abkülen | Abkühlen | Wenn das Heizfeld 30°C unterschritten hat wechselt die RGB-LED auf grün. Ein neuer start kann durchgeführt werden. | Passed |
| 12 | Stopp Funktion | Nach beliebiger Zeit Stopp-Taster drücken. | * RGB-LED wechselt auf blau * Segmentanzeige zeigt Temperatur * Wenn das Heizfeld 30°C unterschritten hat wechselt die RGB-LED auf grün. Ein neuer start kann durchgeführt werden. | Passed |

## Testbericht mit Fazit und Massnahmen

Die Software erfüllt die gegebenen Anforderungen.

Ort, Datum und Unterschrift:

Rotkreuz, 30. September 2019, Kenneth Mathis