|  |
| --- |
| Aufgabe Stoppuhr ETB3 Kapitel 14.2.10 |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Änderung | Name | Datum |
| 1.0 | Erstellung | Kenneth Mathis | 16. September 2019 |

Inhalt

[1 Aufgabenstellung 3](#_Toc18933017)

[2 Hardware-Struktur 3](#_Toc18933018)

[2.1 Systembeschreibung 3](#_Toc18933019)

[2.1.1 Beschaltung der Hardware 3](#_Toc18933020)

[2.1.2 Inputs 4](#_Toc18933021)

[2.1.3 Outputs 4](#_Toc18933022)

[2.1.4 Steuersignale & LED Positionen 4](#_Toc18933023)

[3 Software-Analyse 5](#_Toc18933024)

[3.1 Struktogramm des Hauptprogramms 5](#_Toc18933025)

[3.2 Struktogramm zur Flankenerkennung 5](#_Toc18933026)

[3.3 Struktogramm der Bitmaskenwandlung 6](#_Toc18933027)

[3.4 Struktogramm der 7-Segmentanzeige 6](#_Toc18933028)

[4 Source-Code 7](#_Toc18933029)

[5 Testprotokoll 10](#_Toc18933030)

[5.1 Testablauf 10](#_Toc18933031)

[5.2 Testbericht mit Fazit und Massnahmen 10](#_Toc18933032)

# Aufgabenstellung

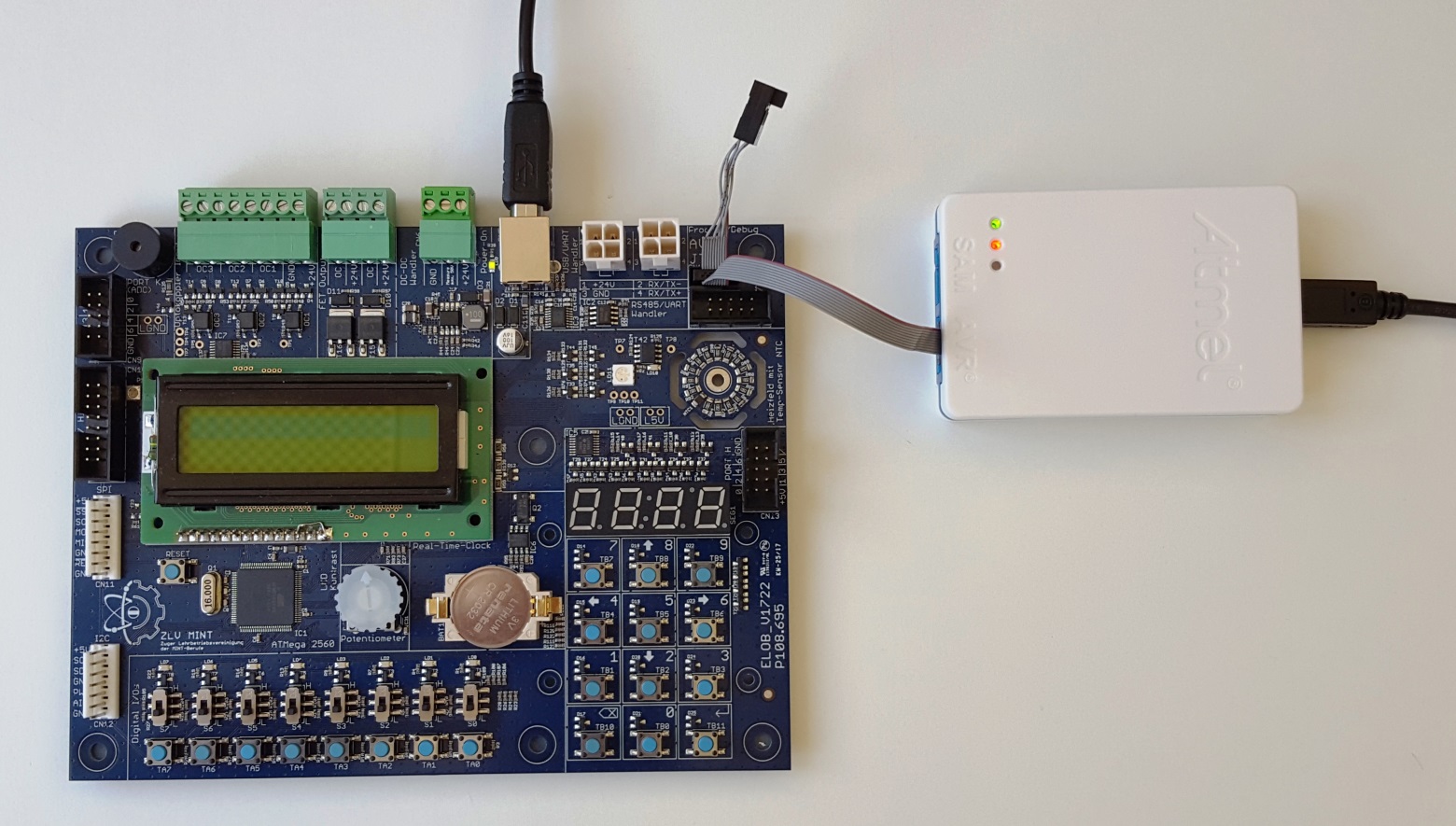
Es wird die Software einer Stoppuhr realisiert.

* ***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

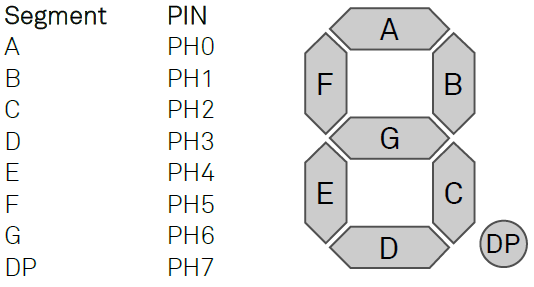
### Inputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bezeichner** | **uC-Pin** | **Bauteil** | **Logikpegel-Definitionen** | **Grobe Funktionsbeschreibung** |
| tstStartStop | PJ0 | TA0 | 1 = Gedrückt | Start/Stop Taster |
| tstReset | PJ1 | TA1 | 1 = Gedrückt | Reset Taster |

### Outputs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bezeichner** | **uC-Pin** | **Bauteil** | **Logikpegel-Definitionen** | **Grobe Funktionsbeschreibung** |
| enSevenSeg | PG2 | IC1 | 1 = Enabled | 7-Segment Enable Signal |
| segSel1 | PG0 | IC1 | Siehe Tabelle | 7-Segment Select Signal |
| segSel2 | PG1 | IC1 |
| 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 |

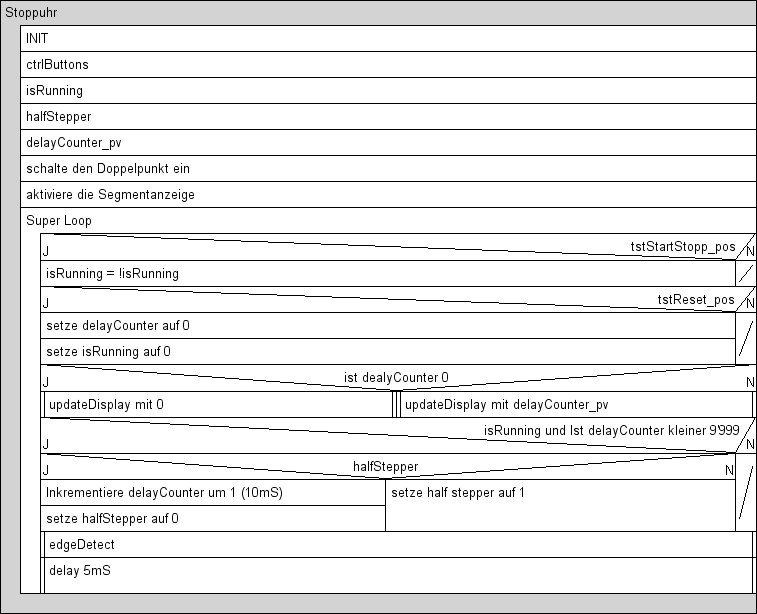
### Steuersignale & LED Positionen



|  |  |  |
| --- | --- | --- |
| **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

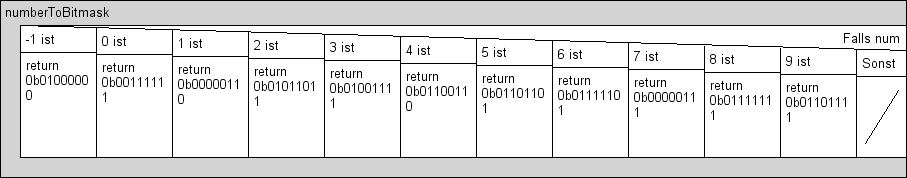
## Struktogramm des Hauptprogramms



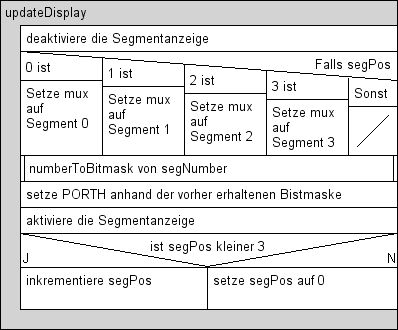
## Struktogramm zur Flankenerkennung



## Struktogramm der Bitmaskenwandlung



## Struktogramm der 7-Segmentanzeige



# Source-Code

/\*\*

\* Software for a stopwatch

\*

\* @Autor: Mathis Kenneth

\* @Version: 1.1

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

\* Datum     Vers.   Name            Kommentar / Änderung

\* 03.09.19  1.0     Mathis Kenneth  Erstellt

\* 16.09.19  1.1     Mathis Kenneth  General Revamp

\*/

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

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#include <math.h>

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

//\*\*Segment select defines\*\*

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

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

#define segSel0\_on (PORTG |= (1<<0))

#define segSel1\_on (PORTG |= (1<<1))

//\*\*Button press defines\*\*

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

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

//\*\*Set segment led's defines

#define setSeg(bit) (PORTH |= (bit))

#define delSeg(bit) (PORTH &= (bit))

//\*\*Misc\*\*

#define colonOn (PORTG |= (1<<4))

#define segEnable (PORTG |= (1<<2))

#define segDisable (PORTG &= ~(1<<2))

//\*\*\*STRUCT'S\*\*\*

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;

//\*\*\*FUNCTION PROTOTYPES\*\*\*

void updateSegment(unsigned int segNumber);

uint8\_t numberToBitmask(uint8\_t num);

void edgeDetect(edge \*port);

//\*\*\*MAIN ROUTINE\*\*\*

int main(void){

    //\*\*Init\*\*

    //\*\*Variables\*\*

    uint8\_t isRunning;      //Is 1 when the stopwatch is counting

    uint8\_t halfStepper;    //In order to run 5mS delay but increment delayCouner\_pv only every 10mS

    unsigned int delayCounter\_pv = 0; //Counts how many delays of 10mS have passed

    edge ctrlButtons = {0,0,&PINJ}; //Edge detect struct for control buttons

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

    //\*1=Output\*

    DDRG = 0xff; //MUX

    DDRH = 0xff; //Segment LED's

    DDRJ = 0; //Ctrl Buttons

    //\*\*Turn on the colon led and enable the segment\*\*

    colonOn;

    segEnable;

    //\*\*\*SUPER LOOP\*\*\*

    while (1){

        //\*\*Detect positive Edges on StartStop and Reset\*\*

        if(tstStartStop\_pos) isRunning = !isRunning;

        if(tstReset\_pos){

            delayCounter\_pv = 0;

            isRunning = 0;

        }

        //\*\*Update the segment\*\*

        if(delayCounter\_pv == 0)updateSegment(0);

        else updateSegment(delayCounter\_pv);

        //\*\*Do half steps and stop at a 99'990mS\*\*

        if(delayCounter\_pv < 9999 && isRunning){

            if(halfStepper){

                halfStepper = 0;

                delayCounter\_pv++;

            }

            else halfStepper = 1;

        }

        //\*\*Detect the edges\*\*

        edgeDetect(&ctrlButtons);

        //\*\*Global Delay\*\*

        \_delay\_ms(5);

    }

}

/\*\*\*UPDATE 7-SEGMENT\*\*\*

 \* 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

\*/

void updateSegment(unsigned int segNumber){

    static uint8\_t segPos\_pv;

    segDisable; //Disable the segment in order to avoid ghost digits

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

    switch(segPos\_pv){

        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 PORTH to it\*\*

    uint8\_t bitmask;

    if(segNumber == 0) bitmask = numberToBitmask(0); //If the given number is zero, display it as such

    else{

        uint8\_t segDigit;

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

        bitmask = numberToBitmask(segDigit);

    }

    //\*\*Update the segment\*\*

    setSeg(bitmask);

    delSeg(bitmask);

    segEnable; //Make the segment visible again

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

    else segPos\_pv = 0; //return to first segment after the last segment

}

/\*\*\*NUMBER TO BITMASK\*\*\*

 \* Turns a number into a seven segment bitmask

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

 \* @return: the bitmask of the given number

\*/

uint8\_t numberToBitmask(uint8\_t num){

    switch(num){

        case 0:

            return 0b00111111;

        case 1:

            return 0b00000110;

        case 2:

            return 0b01011011;

        case 3:

            return 0b01001111;

        case 4:

            return 0b01100110;

        case 5:

            return 0b01101101;

        case 6:

            return 0b01111101;

        case 7:

            return 0b00000111;

        case 8:

            return 0b01111111;

        case 9:

            return 0b01101111;

        default:

            return 0b01000000;

    }

}

// \*\*\*EDGE DETECTION\*\*\*

/\*\*

\*@param: An edge struct is needed which defines stores all the necessary stuff for the edge detection

\*@return: Return via "reference"

\*/

void edgeDetect(edge \*port){

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

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

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

    port->old = pinjCurrent;

} // \*\*EOF\*\*

# Testprotokoll

## Testablauf

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nr.** | **Testfall** | **Vorgang** | **Erwartung** | **Befund** |
| 01 | Einschalten/Startup | Speisung einschalten | Die Segmentanzeige zeigt 00:00 | Passed |
| 02 | Taster Start/Stopp gedrückt | Taster Start/Stopp drücken | Die Segmentanzeige fängt an in 10mS Schritten zu zählen. | Passed |
| 03 | Taster Start/Stopp gedrückt | Taster Start/Stopp drücken | Die Segmentanzeige stopp das Zählen. | Passed |
| 04 | Taster Reset gedrückt | Taster Reset drücken | Die Segmentanzeige zeigt 00:00 | Passed |
| 05 | Taster Start/Stopp gedrückt | Taster Start/Stopp drücken | Die Segmentanzeige fängt an in 10mS Schritten zu zählen. | Passed |
| 06 | Taster Reset gedrückt | Taster Reset drücken | Die Segmentanzeige zeigt 00:00 | Passed |

## Testbericht mit Fazit und Massnahmen

Die Software erfüllt die gegebenen Anforderungen.

Ort, Datum und Unterschrift:

RDI Rotkreuz, 09-SEP-2019, K. Mathis

Rotkreuz, 16. September 2019, Kenneth Mathis