



BERGISCHE
UNIVERSITÄT
WUPPERTAL

BERGISCHE UNIVERSITÄT WUPPERTAL

ELEKTRONIK PRAKTIKUM

Versuch EP10 Digitalelektronik Teil 3: Mikrocontroller

Autoren:

Henrik JÜRGENS

Frederik STROTHMANN

Tutoren:

Hans-Peter KIND

Peter KNIELING

Marius WENSING

14. Januar 2015

Inhaltsverzeichnis

1	Einleitung	2
2	Messen der Arbeitsgeschwindigkeit des PIC18F4455	2
2.1	Schnellstmögliches Programm, ohne Verzögerungsbefehle	2
2.2	Programm mit Verzögerungsbefehlen aus for-Schleifen	4
2.3	Programm mit Compiler-Verzögerungsbefehlen	5
2.4	Programm mit fertigen Verzögerungsbefehlen	6
3	Benutzung der Taster	8
3.1	Einfache Tasterabfrage	9
3.2	Taster steuern Speicherfunktionen	9
3.3	Tasterabfrage und Verzögerungen	10
3.4	Taster steuern einen Zähler	11
3.5	Kontaktprellen der Taster – eine Lösung	11
4	Zähler für Oszillatortakte	12
4.1	Zähler ohne Vorteiler	12
4.2	Zähler mit Vorteiler	13
5	Verwendung des Analog-Digital-Konverters (ADC)	14
5.1	Anzeige des ADC-Wertes auf 8 LEDs	14
5.2	Anzeige des ADC-Wertes auf der Siebensegmentanzeige	15
5.3	Anzeige von zwei ADC-Kanälen auf der Siebensegmentanzeige	16
5.4	Anzeige des ADC-Wertes auf einer Balkenanzeige	17
6	Fazit	18

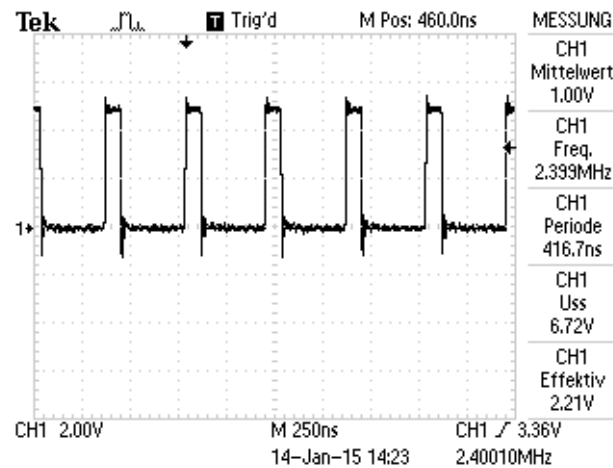


Abbildung 3: Aufnahme des Signals

2.2 Programm mit Verzögerungsbefehlen aus for-Schleifen

In diesem Versuchsteil soll wieder eine LED ein- und ausgeschaltet werden, jedoch mit einer verzögerung. Die Verzögerung soll mit Hilfe einer for-schleife realisiert werden. Zur erzeugen der Verzögerung wird der Befehl Nop() verwendet. Der Befehl sorgt dafür, das der Mikocontroller für einen Taktzyklus nichts macht.

Verwendete Geräte

Es wird ein Netzgerät, ein PC, ein Verbindungskabel, ein Oszilloskop und das Versuchsboard verwendet.

Quellcode

Quellcode zum testen der Geschwindigkeit des Boards mit einer for-Schleife und dem Nop()-Befehl.

```
void test_speed_for(){
    int i;
    TRISB = 0b00000000;

    while(1){
        for(i=0; i<10; i++){
            Nop();
        }

        LATB = 0b11111111;

        for(i=0; i<10; i++){
            Nop();
        }

        LATB = 0b00000000;
    } //end of while(1)
} //end of function test_speed_for()
```

Listing 2: Geschwindigkeitstest mit einer for-Schleife

Versuchsdurchführung

Es wird der Code 19 auf den Mikrokontroller geladen. Es wird der B Port mit dem LED Port über ein Flachband verbunden. Dann wird das Board an ein Oszilloskop angeschlossen und die Ausgangsfrequenz an den Pins gemessen.

Auswertung

Die LEDs leuchteten, es war keine flackern zu erkennen. Die Messung der Frequenz mit dem Oszilloskop ergab eine Frequenz von 28,9 kHz. Der Verlauf des Signals ist in Abbildung 4 zu sehen.

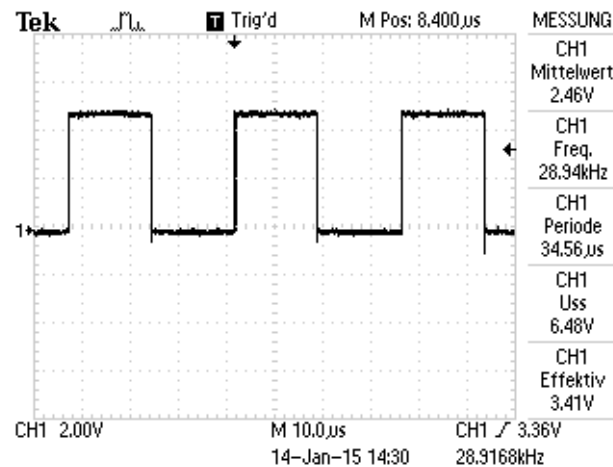


Abbildung 4: Aufnahme des Signals

2.3 Programm mit Compiler-Verzögerungsbefehlen

In diesem Versuchsteil soll eine Verzögerung des Ein- und Ausschaltvorgangs der LED mit Compiler-Verzögerungsbefehlen realisiert werden. Bei dieser Art der Verzögerung wird eine bestimmte Anzahl von Taktzyklen ausgesetzt. Zur erzeugung der Verzögerung werden die Befehle (unit kann Werte von 0 bis 255 annehmen):

- Delay1TCY() oder Nop() delay von einem instruction cycle
- Delay10TCYx(unit) delay von $\text{unit} \times 10$ instruction cycle
- Delay100TCYx(unit) delay von $\text{unit} \times 100$ instruction cycle
- Delay10KTCYx(unit) delay von $\text{unit} \times 1000$ instruction cycle
- Delay10KTCYx(unit) delay von $\text{unit} \times 10000$ instruction cycle

Verwendete Geräte

Es wird ein Netzgerät, ein PC, ein Verbindungskabel, ein Oszilloskop und das Versuchsboard verwendet.

Quellcode

Quellcode zum testen der Compiler-Verzögerungsbefehle.

```
void test_speed_delay_1(){
    int i;
    TRISB = 0b00000000;

    while(1){
        Delay10TCYx(100);
        LATBbits.LATB0 = 1;
        Delay100TCYx(100);
        LATBbits.LATB1 = 1;
        Delay1KTCYx(100);
        LATBbits.LATB2 = 1;
        Delay10KTCYx(100);
        LATBbits.LATB3 = 1;
        Delay10TCYx(255);
    }
}
```

```

LATBbits.LATB4 = 1;
Delay100TCYx(255);
LATBbits.LATB5 = 1;
Delay1KTCYx(255);
LATBbits.LATB6 = 1;
Delay10KTCYx(255);
LATBbits.LATB7 = 1;
Delay10KTCYx(100);

LATB = 0b00000000;

} //end of while(1)
} //end of function test_speed_delay_1()

```

//Wurde im ersten Durchlauf auskommentiert

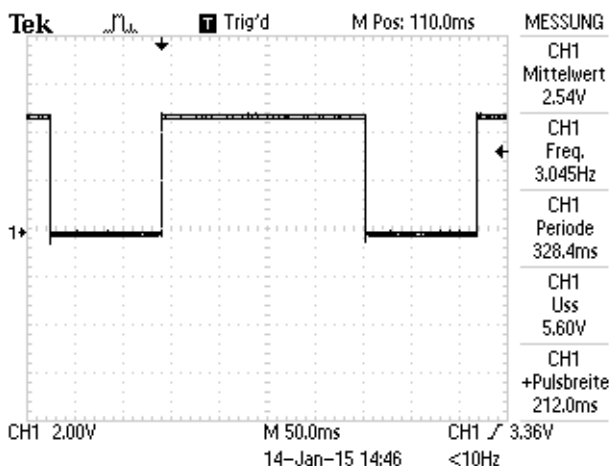
Listing 3: Geschwindigkeitstest mit Compiler-Verzögerungsbefehlen

Versuchsdurchführung

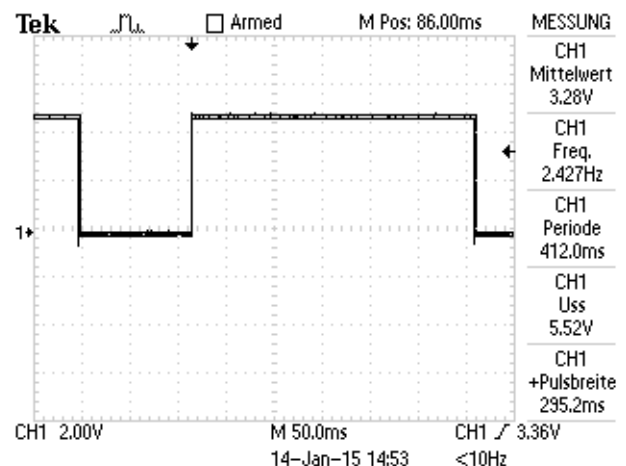
Es wird der Code 28 auf den Mikrokontroller geladen. Es wird Port B mit Port LED, über ein Flachbandkabel verbunden. Dann wird das Board an ein Oszilloskop angeschlossen und die Ausgangsfrequenz an den Pins gemessen. Dann wurde die in Code 28 erwähnte Zeile auskommentiert und Frequenz mit dem Oszilloskop gemessen.

Auswertung

Die Spannung wurde bei B6 abgegriffen, im ersten Durchlauf wurde eine Frequenz von 3,04Hz gemessen. Der Verlauf des Signals ist in Abbildung 6c zu sehen. Im zweiten Durchlauf wurde eine Frequenz von 2,42Hz gemessen. Der Verlauf des Signals ist in Abbildung 6b zu sehen.



(a) Aufnahme des Signals, mit Auskommentierung



(b) Aufnahme des Signals, ohne Auskommentierung

Bei beiden Versionen leuchteten die ersten drei LEDs durch und die restlichen blingen.

2.4 Programm mit fertigen Verzögerungsbefehlen

In diesem Versuchsteil soll das An- und Ausschalten der LED mit den verschiedenen Verzögerungsbefehlen realisiert werden. Verzögerungsbefehle sorgen für eine Verzögerung von einer bestimmten Zeitspanne und nicht wie Vorher für ein Aussetzen einer Anzahl von Taktzyklen. Es werden die Befehle (unit kann Werte von 0 bis 255):

- delay us(unit) verzögert um unit Mikrosekunden
- delay 10us(unit) verzögert um unit × 10 Mikrosekunden
- delay ms(unit) verzögert um unit Millisekunden

Verwendete Geräte

Es wird ein Netzgerät, ein PC, ein Verbindungskabel, ein Oszilloskop und das Versuchsboard verwendet.

Messergebnisse

Befehl	Pulsbreite/ μs
delay_us(1)	1,66
delay_us(2)	2,66
delay_10us(0)	1,91
delay_10us(1)	12,4
delay_ms(0)	3,00
delay_ms(1)	995,9

Tabelle 1: Pulsbreiten der Verschiedenen Befehle

Quellcode

Quellcode zum testen der fertigen Verzögerungsbefehlen.

```
void test_speed_delay_3(){
    TRISB = 0b00000000;

    while(1){
        delay_us(25);
        LATBbits.LATB0 = 1;
        delay_us(25);
        LATBbits.LATB1 = 0;
    } //end of while(1)
} //end of function test_speed_delay_3()
```

Listing 4: Geschwindigkeitstest mit den fertigen Verzögerungsbefehlen

Versuchsdurchführung

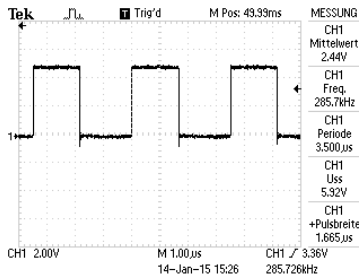
Es wird der Code 11 auf den Mikrokontroller geladen. Es wird der B Port mit dem LED Port verbunden. Dann wird das Board an ein Oszilloskop angeschlossen und die Pulsbreite der Signale an den Pins gemessen. Dieser Prozess wird mit verschiedenen Verzögerungsbefehlen wiederholt.

Auswertung

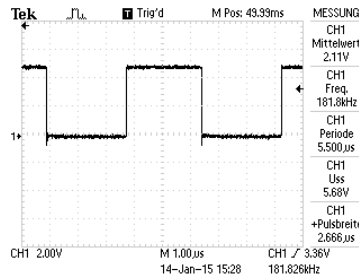
Es wurde Quellcode 11 verwendet für den Befehl **delay_us(25)** wurde der Reihe nach durch:

- delay_us(1)
- delay_us(2)
- delay_10us(0)
- delay_10us(1)
- delay_ms(0)
- delay_ms(1)

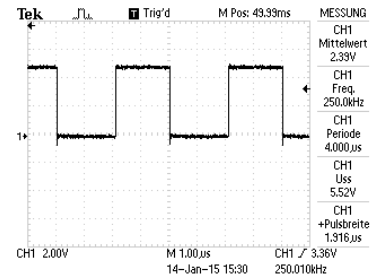
ersetzt und jeweils die Frequenz gemessen. Dabei ergaben sich die Werte in Tabelle 1. Die Aufnahmen der jeweiligen signale sind in Abbildung 6 und Abbildung 7 zu sehen. Bei allen Befehlen ist die benötigte zeit 0,6 bis 3 μs länger als gewünscht, dies kommt daher, dass intern berechnet werden muss wie viele Zyklen ausgesetzt werden müssen, um die gewünschte Verzögerung zu erreichen. Ein Zyklus dauert 83,33 ns. Der letzte Befehl hat eine geringere Pulsbreite als 1ms.



(a) Aufnahme des Signals, für delay_us(1)

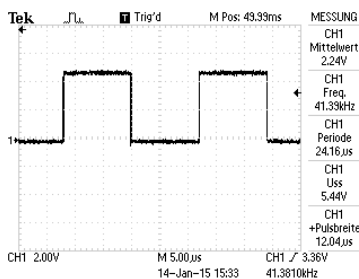


(b) Aufnahme des Signals, für delay_us(2)

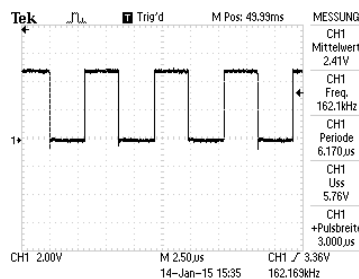


(c) Aufnahme des Signals, für delay_10us(0)

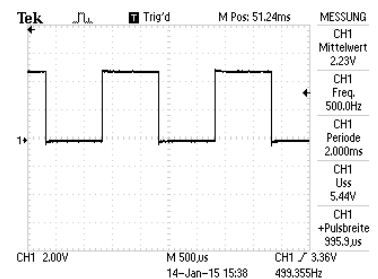
Abbildung 6: Oszilloskopaufnahmen für delay_us(1), delay_us(2) und delay_10us(0)



(a) Aufnahme des Signals, für delay_10us(1)



(b) Aufnahme des Signals, für delay_ms(0)



(c) Aufnahme des Signals, für delay_ms(1)

Abbildung 7: Oszilloskopaufnahmen für delay_10us(1), delay_ms(0) und delay_ms(1)

Diskussion

In dem Versuchsteil liessen sich die verschieden Methoden zur Verzögerung einer Ausführung gut untersuchen. So wie die Vor- und Nachteile der einzelnen Möglichkeiten der Verzögerung. Mit den Delay*TCY*() befehlen lassen sich ziemliche genaue Verzögerungen realisieren, jedoch muss man immer die Anzahl der ausgelassenen Zyklen noch in eine Zeit umrechnen. Eine bestimmte Zeit lässt sich einfacher mit den delay_*() Befehlen umsetzen, diese Befehle haben bei sehr kurzen Verzögerungen jedoch eine große Ungenauigkeit, da der Mikrokontroller noch etwas Zeit zur Berechnung braucht.

3 Benutzung der Taster

In diesem Versuchsteil sollen Taster und deren Funktionsweise untersucht werden. Es sollen LEDs mit den 8 Tastern an- und ausgeschaltet werden, dafür wird die PollingMethode verwendet.

3.1 Einfache Tasterabfrage

In diesem Versuchsteil soll die einfachste Variante eines Tasters implementiert werden. Das Programm prüft, ob der Taster gedrückt wurde und schaltet die LED ein, wenn der Taster gedrückt wurde sonst nicht.

Verwendete Geräte

Es wird ein Netzgerät, ein PC, ein Verbindungskabel, ein Oszilloskop und das Versuchsboard verwendet.

Quellcode

Quellcode für einen einfachen Taster.

```
void taster_1(){
    byte i;
    TRISB = 0b00000000;
    TRISA = 0b111111;

    while(1){
        i = PORTA;

        LATB = i;
    } //end of while(1)
} //end of function taster_1()
```

Listing 5: Einfachster Version eines Schalters

Versuchsdurchführung

Es wird Code 12 auf das Versuchsboard geladen, Port A wird mit den Tastern und Port B mit dem LED Port verbunden. Dann wird das Verhalten des Boards untersucht.

Auswertung

Beim Druck eines Tasters außer 4,7 und 8 leuchtete die erste LED auf.

3.2 Taster steuern Speicherfunktionen

In diesem Versuchsteil soll der Zustand der LED gewechselt werden, wenn der Taster gedrückt wird, sonst nicht.

Verwendete Geräte

Es wird ein Netzgerät, ein PC, ein Verbindungskabel, ein Oszilloskop und das Versuchsboard verwendet.

Quellcode

Quellcode zum verwenden eines Tasters als Schalter.

```
void taster_2(){
    byte i;
    TRISB = 0b00000000;
    TRISA = 0b111111;

    while(1){
        if(PORTA == 0b000001){
            if(LATB == 0b00000001)
                LATB = 0b00000000;
            else
                LATB = 0b00000001;
        }
        delay_ms(25);
    } //end of while(1)
} //end of function taster_1()
```

```

void taster_3(){
    byte i;
    TRISB = 0b00000000;
    TRISA = 0b111111;

    while(1){
        if(PORTAbits.RA1 == 0b1){
            if(LATB == 0b00000001)
                LATB = 0b00000000;
            else
                LATB = 0b00000001;
        }
        delay_ms(25);
    } //end of while(1)
} //end of function taster_3()

```

Listing 6: Taster steuert Speicherfunktion

Versuchsdurchführung

Es wird Code 32 auf das Versuchsboard geladen, Port A wird mit den Tastern und Port B mit dem LED Port verbunden. Dann wird das Verhalten des Boards untersucht.

Auswertung

Beim verwenden der Funktion `taster_2()` war ein schnelles Blinken der ersten LED zu sehen, wenn noch ein anderer Taster neben dem ersten gedrückt wurde blinkte die LED nicht. Beim verwenden der Funktion `taster_3()` war auch ein schnelles blinken der LED zu sehen. Es war auch ein Blinken zu sehen, wenn ein anderer Taster zusätzlich gedrückt wurde.

3.3 Tasterabfrage und Verzögerungen

In diesem Versuchsteil soll eine Verzögerung implementiert werden, da sonst kein sauberes umschalten möglich ist und man nur ein flackern sehen kann.

Quellcode

Quellcode für einen Schalter mit Verzögerung.

```

void taster_4(){
    byte i;
    TRISB = 0b00000000;
    TRISA = 0b111111;

    while(1){
        if(PORTAbits.RA0 == 0b1){
            if(LATB == 0b00000001)
                LATB = 0b00000000;
            else
                LATB = 0b00000001;
        }
        delay_ms(500);
    } //end of while(1)
} //end of function taster_4()

```

Listing 7: Tastabfrage und Verzögerung

Versuchsdurchführung

Es wird Code 16 auf das Versuchsboard geladen, Port A wird mit den Tastern und Port B mit dem LED Port verbunden. Dann wird das Verhalten des Boards untersucht.

Auswertung

Mit dem hinzufügen der Verzögerung konnte der Taster mit einer Frequenz von 1Hz als Schalter verwendet werden.

3.4 Taster steuern einen Zähler

In diesem Versuchsteil soll der Taster einen Zähler steuern und bei jedem Tasterndruck einen hochzählen.

Quellcode

Quellcode zum steuern eines Zählers mit einem Taster.

```
void taster_5(){
    byte taster_alt = 0b00000000;
    byte taster_neu = 0b00000000;
    byte counter = 0;
    TRISB = 0b00000000;
    TRISA = 0b111111;

    while(1){
        taster_neu = PORTA;
        if(taster_neu != taster_alt){
            taster_alt = taster_neu;
            counter++;
            LATB = counter;
        } //end of if(flag%2 == 1)
    } //end of if(taster_neu != taster_alt)
} //end of while(1)
} //end of function taster_5()

void taster_6(){
    byte taster_alt = 0b00000000;
    byte taster_neu = 0b00000000;
    byte counter;
    byte flag = 0;
    TRISB = 0b00000000;
    TRISA = 0b111111;

    while(1){
        taster_neu = PORTA;
        if(taster_neu != taster_alt){
            flag++;
            taster_alt = taster_neu;
            if(flag%2 == 0){
                counter++;
                LATB = counter;
            } //end of if(flag%2 == 1)
        } //end of if(taster_neu != taster_alt)
    } //end of while(1)
} //end of function taster_6()
```

Listing 8: Code zum steuern eines Zählers mit einem Schalter

Versuchsdurchführung

Es wird Code 39 auf das Versuchsboard geladen, Port A wird mit den Tastern und Port B mit dem LED Port verbunden. Dann wird das Verhalten des Boards untersucht.

Auswertung

Bei der Implementierung von `taster_5()` wurde beim runterdrücken und beim loslassen der counter um einen erhöht, dies konnte man auch an den LEDs sehen. Bei der Implementierung von `taster_6()` wurde nur beim loslassen des Tasters der counter einen hochgezählt, dies was auch bei den LEDs zu sehen.

3.5 Kontaktprellen der Taster – eine Lösung

In diesem Versuchsteil soll der Effekt des Kontaktprellen verhindert werden. Kontaktprellen ist, wenn der Taster-Kontakt nicht sauber schließt und dadurch mehrfaches drücken registriert wird.

Verwendete Geräte

Es werden ein Netzgerät, das Versuchsboard, Verbindungskabel, ein PC und ein Oszilloskop verwendet.

Quellcode

Quellcode für die Verwendung eines Tasters als Schalter mit unterdrückung von Kontaktprellen.

```
void taster_7(){
    byte taster_alt = 0b00000000;
    byte taster_neu = 0b00000000;
    byte counter = 0;
    TRISB = 0b00000000;
    TRISA = 0b111111;

    while(1){
        delay_ms(10);
        if(PORTA){
            delay_ms(10);
            while(PORTA){
                //end of while (!PORTA)
                counter++;
                LATB = counter;
            } //end of if (taster_neu != taster_alt)
        } //end of while(1)
    } //end of function taster_7()
```

Listing 9: Verhindern von Kontaktprellen

Versuchsdurchführung

Es wird Code 19 auf das Versuchsboard geladen, Port A wird mit den Tastern und Port B mit dem LED Port verbunden. Dann wird das Verhalten des Boards untersucht.

Auswertung

Bei der Implementierung von Code 19 wurde beim loslassen der counter um einen erhöht, was auch an den LEDs zu sehen war.

Diskussion

In diesem Versuchsteil konnten gut die Verwendungsmöglichkeiten, sowie verschiedene Implementierungen dieser untersucht werden. In allen Versuchsabschnitten ließen sich die gewünschten Ergebnisse erzielen.

4 Zähler für Oszillatortakte

In diesem Versuchsabschnitt soll ein Zähler programmiert und implementiert werden.

4.1 Zähler ohne Vorteiler

In diesem Versuchsteil wird ein Zähler ohne Vorteiler implementiert.

Verwendete Geräte

Es werden ein Netzgerät, das Versuchsboard, Verbindungskabel, ein PC und ein Oszilloskop verwendet.

Quellcode

Quellcode für den Zähler ohne Vorteiler.

```
void oszi_1(){
    byte counter = 0;
    TRISB = 0b00000000;
    TRISA = 0b111111;

    while(1){
        while(PORTAbits.RA0){
            counter++;
            LATB = counter;
        }
    }
}
```

```

    } //end of while (LATAbits.LATA0)
  } //end of function oszi_1()
} //end of while (1)

```

Listing 10: Zähler ohne Vorteiler

Versuchsdurchführung

Code 13 wird auf das Versuchboard geladen. Dann wird Pin 1 von Port A mit dem Oszillator und Port B mit dem LED Port verbunden.

Auswertung

Da der Oszillator mit 60kHz taktet, kann man nur ein leuchten aller LEDs beobachten. Auf dem Oszilloskop ist der Verlauf in Abbildung 8 zu sehen. Da der Mikrocontroller eine viel höhere Taktrate als der Oszillator hat kann man wärem einem Taktsinal des Oszillators mehrere Zyklen des Mikrocontrollers sehen.

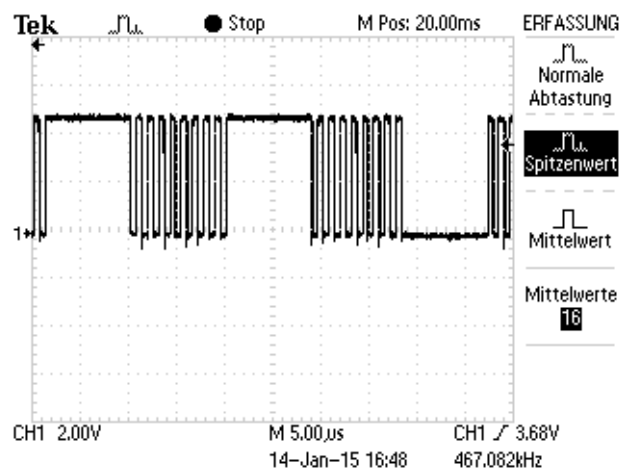


Abbildung 8: Aufnahme des Signals

4.2 Zähler mit Vorteiler

In diesem Versuchsteil soll noch ein Vorteiler implementiert werden, damit nicht nur ein blinken der LEDs zu beobachten ist. Die Frequenz soll auf 5Hz geregelt werden.

Verwendete Geräte

Es werden ein Netzgerät, das Versuchboard, Verbindungskabel, ein PC und ein Oszilloskop verwendet.

Quellcode

Quellcode für den Zähler mit Vorteiler.

```

void oszi_2(){
  int i;
  byte counter = 0;
  TRISB = 0b00000000;
  TRISA = 0b111111;

  while(1){
    while(PORTAbits.RA0){
      for(i = 0; i < 52428; i++)
        Nop();
      counter++;
      LATB = counter;
    } //end of while (1)
  }
}

```

```
//end of function oszi_2()
```

Listing 11: Zähler mit Vorteiler

Versuchsdurchführung

Code 15 wird auf das Versuchboard geladen. Dann wird Pin 1 von Port A mit dem Oszillator und Port B mit Port LED verbunden.

Auswertung

Die LED-Anzeige zählte mit einer Frequenz von 5Hz hoch. Auf dem Oszilloskop war der Verlauf in Abbildung 9 zu sehen.

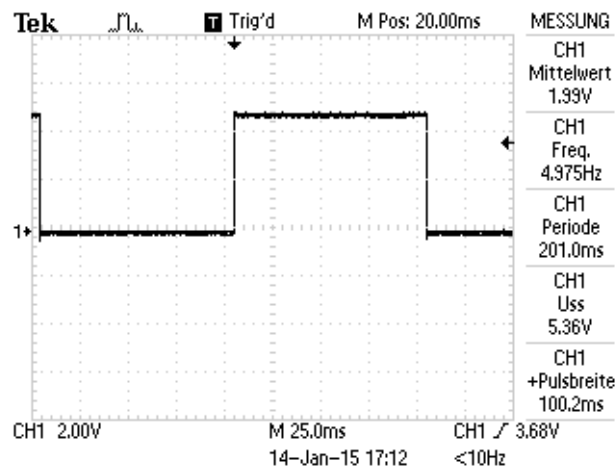


Abbildung 9: Aufnahme des Signals

Diskussion

In diesem Versuchsabschnitt liesen sich gut die Verwendungsmöglichkeit eines Oszillators untersuchen.

5 Verwendung des Analog-Digital-Konverters (ADC)

Der Mikrocontroller besitzt eine ADC, mit dem Spannungen von 0 bis 5V gemessen werden können. In diesem Versuchsabschnitt soll damit ein Voltmeter gebaut werden.

5.1 Anzeige des ADC-Wertes auf 8 LEDs

In diesem Versuchsteil soll die einfachste Version eines Voltmeters implementiert werden. Zur Ausgabe werden die 8 LEDs verwendet.

Verwendete Geräte

Es werden das Versuchboard, Verbindungskabel, ein PC und ein Netzgerät verwendet.

Quellcode

Quellcode für den adc mit LEDs.

```
void adc_1(){
    TRISB = 0b00000000;
    TRISAbits.TRISA0 = 0b1;

    setup_adc_ports(AN0_TO_AN1);

    setup_adc(ADC_CLOCK_DIV_32);

    set_adc_channel(0);

    read_adc(ADC_START_ONLY);

    while(1){
        delay_ms(10);
        LATB = (read_adc(ADC_READ)>>2);
    } //end of while(1)
} //end of function adc_1()
```

Listing 12: adc mit LEDs

Versuchsdurchführung

Es wird Code 18 auf den Mikrocontroller geladen und eine Spannung an AN0 angelegt. Die externe Spannung wird auf 0 und 5 V eingestellt und dann mit dem Board gemessen. Dann soll noch die Spannung ermittelt werden, bei dem das Board den maximalen und den mittleren Wert anzeigt.

Auswertung

Bei 0V waren alle LEDs aus, entspricht also einer decimalen 0. Bei 5V waren alle LEDs an, was einem decimalen Wert von 255 entspricht. Der halbe Maximalwert wurde bei 2,4V angezeigt.

5.2 Anzeige des ADC-Wertes auf der Siebensegmentanzeige

In diesem Versuchsteil soll die gemessene Spannung mit einer Siebensegmentanzeigen ausgegeben werden.

Verwendete Geräte

Es werden das Versuchsboard, Verbindungskabel, ein PC und ein Netzgerät verwendet.

Quellcode

Quellcode für die Ausgabe des adc mit einer Siebensegmentanzeige.

```
void adc_2(){
    char value;
    byte valH;
    byte valL;

    TRISD = 0b00000000;
    TRISC = 0b00000000;
    TRISB = 0b00000000;
    TRISAbits.TRISA0 = 1;

    setup_adc_ports(AN0_TO_AN1);

    setup_adc(ADC_CLOCK_DIV_32);

    set_adc_channel(0);

    read_adc(ADC_START_ONLY);

    while(1){
        delay_ms(10);

        value = read_adc(ADC_READ)>>2;

        valH = value>>4;
        valL = (value & 0b00001111)>>4;

        LATD = dec_7seg[valH];
    }
```



```

        LATC = dec_7seg[valL];
    } //end of while(1)
} //end of function adc_2()

```

Listing 13: adc mit Siebensegmentanzeige

Versuchsdurchführung

Es wird Code 32 auf den Mikrocontroller geladen und eine Spannung an AN0 angelegt. Die externe Spannung wird auf 0 und 5 V eingestellt und dann mit dem Board gemessen. Dann soll noch die Spannung ermittelt werden, bei dem das Board den maximalen und den mittleren Wert anzeigt.

Auswertung

Bei 0V wurde der Wert 0 auf der Siebensegmentanzeige angezeigt. Der Maximale Wert F wurde zum ersten mal bei 4,6V gemessen, bei 5V wurde auch F angezeigt. F entspricht der dezimalzahl 15. Der halbe maximale Wert wurde bei 2,3V angezeigt und entspricht einer Dezimalzahl von 7;

5.3 Anzeige von zwei ADC-Kanälen auf der Siebensegmentanzeige

In diesem Versuchsteil soll noch die Funktion implementiert werden, mit der per Tasterdruck zwischen den Kanälen AN0 und AN1 umschalten kann.

Verwendete Geräte

Es werden das Versuchsboard, Verbindungskabel, ein PC und ein Netzgerät verwendet.

Quellcode

Quellcode für adc Messung mit Umschaltfunktion.

```

void adc_3(){
    byte flag = 0;

    TRISC = 0b11111111;
    TRISB = 0b00000000;
    TRISAbits.TRISA0 = 1;
    TRISAbits.TRISA1 = 1;

    //setting up the adc
    setup_adc_ports(AN0_TO_AN1);
    setup_adc(ADC_CLOCK_DIV_32);
    set_adc_channel(0);
    read_adc(ADC_START_ONLY);

    //starting the main routine
    while(1){
        if(PORTCbits.RC0 == 0b1){
            if(flag == 0)
                flag = 1;
            else
                flag = 0;
        } //end of if(PORTCbits.RC0 == 0b1)
        if(flag == 0){
            set_adc_channel(0);
            read_adc(ADC_START_ONLY);
        } else {
            set_adc_channel(1);
            read_adc(ADC_START_ONLY);
        }

        delay_ms(100);

        LATB = read_adc(ADC_READ)<<2;

        delay_ms(1000);
    } //end of while(1)
} //end of function adc_3()

```

Listing 14: acd mit Umschaltfunktion

Versuchsdurchführung

Es wird Code 40 auf den Mikrocontroller geladen und überprüft, ob das Programm wie erwartet funktioniert.

Auswertung

Beim drücken des Schalters wurde wie erwartet der Kanal an dem die Spannung gemessen wurde gewechselt. Aufgrund der langen Verzögerung musste immer ein Moment gewartet werden, bis der Kanal wieder gewechselt werden konnte.

5.4 Anzeige des ADC-Wertes auf einer Balkenanzeige

In diesem Versuchsteil soll die gemessene Spannung mit einer Balkenanzeige angezeigt werden.

Verwendete Geräte

Es werden das Versuchsboard, Verbindungskabel, ein PC und ein Netzgerät verwendet.

Quellcode

Quellcode für adc mit Balkenanzeige.

```
void adc_4(){
    byte val;

    TRISB = 0b00000000;
    TRISAbits.TRISA0 = 0b1;

    setup_adc_ports(AN0_TO_AN1);
    setup_adc(ADC_CLOCK_DIV_32);
    set_adc_channel(0);
    read_adc(ADC_START_ONLY);

    while(1){
        delay_ms(10);
        val = (read_adc(ADC_READ)>>7); //Es werden nur die letzten drei bits verwendet,
                                        //da sich daraus acht moegliche Zustaende ergeben
                                        //Die Zustaende muessen alle einzeln ausgewertet werden

        if(val == 0b00000000)
            LATB = 0b00000000;
        if(val == 0b00000001)
            LATB = 0b00000001;
        if(val == 0b00000010)
            LATB = 0b00000011;
        if(val == 0b00000011)
            LATB = 0b00000111;
        if(val == 0b00000111)
            LATB = 0b00001111;
        if(val == 0b00000100)
            LATB = 0b00011111;
        if(val == 0b00000101)
            LATB = 0b00111111;
        if(val == 0b00000110)
            LATB = 0b01111111;
        if(val == 0b00000111)
            LATB = 0b11111111;
    } //end of while(1)
} //end of function adc_4()
```

Listing 15: adc mit Balkenanzeige

Versuchsdurchführung

Es wird Code 37 auf den Mikrocontroller geladen und das Verhalten untersucht.

Auswertung

Da nicht genug Zeit zur Verfügung stand, konnte der Code nicht am Board getestet werden. Der Code sollte lauffähig sein und aus der LED-Anzeige eine Balkenanzeige machen.

6 Fazit