Bergische Universität Wuppertal

Elektronik Praktikum

Versuch EP10 Digitalelektronik Teil 3: Mikrocontroller

Autoren:
Henrik Jürgens
Frederik Strothmann

Tutoren:
Hans-Peter Kind
Peter Knieling
Marius Wensing

15. Januar 2015

Inhaltsverzeichnis

1 Einleitung

In diesem Versuch geht es um Mikrocontroller und deren Programmierung. Es werden das ansteuern von LEDs, das verwenden von Tastern, die Verwendung eines Oszillators und die Verwendung des ACDs behandelt. Als Versuchsboard wird das Board in Abbildung 1 verwendet.

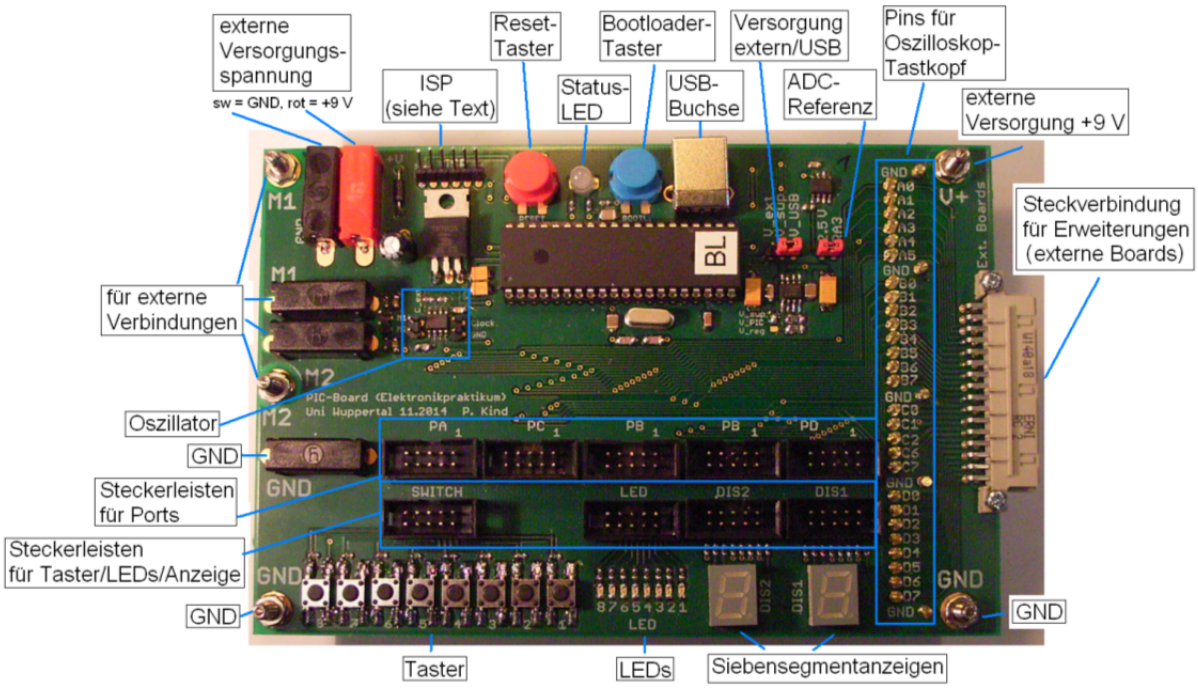


Abbildung 1: Versuchsboard[1]

2 Messen der Arbeitsgeschwindigkeit des PIC18F4455

In diesem Versuchsabschnitt wird die Arbeitsgeschwindigkeit des Mikrocontrollers untersucht, dafür wird eine LED mit verschiedenen Schleifen und einem delay-Befehlen an und aus geschaltet.

2.1 Schnellstmögliches Programm, ohne Verzögerungsbefehle

In diesem Versuchsteil wird einer LED abwechselnd der Zustand 'AN' und 'AUS' zugewiesen. Dies geschieht mit einer 'while(1)' Schleife.

Verwendete Geräte

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

Versuchsaufbau

In allen Versuchsteilen wird das Board in Abbildung 2 verwendet. Der Versuchsaufbau ändert sich bis auf die Verkabelung nicht.

[1] Abbildung entnommen von http://www.atlas.uni-wuppertal.de/~kind/ep10_14.pdf am 10.01.2015

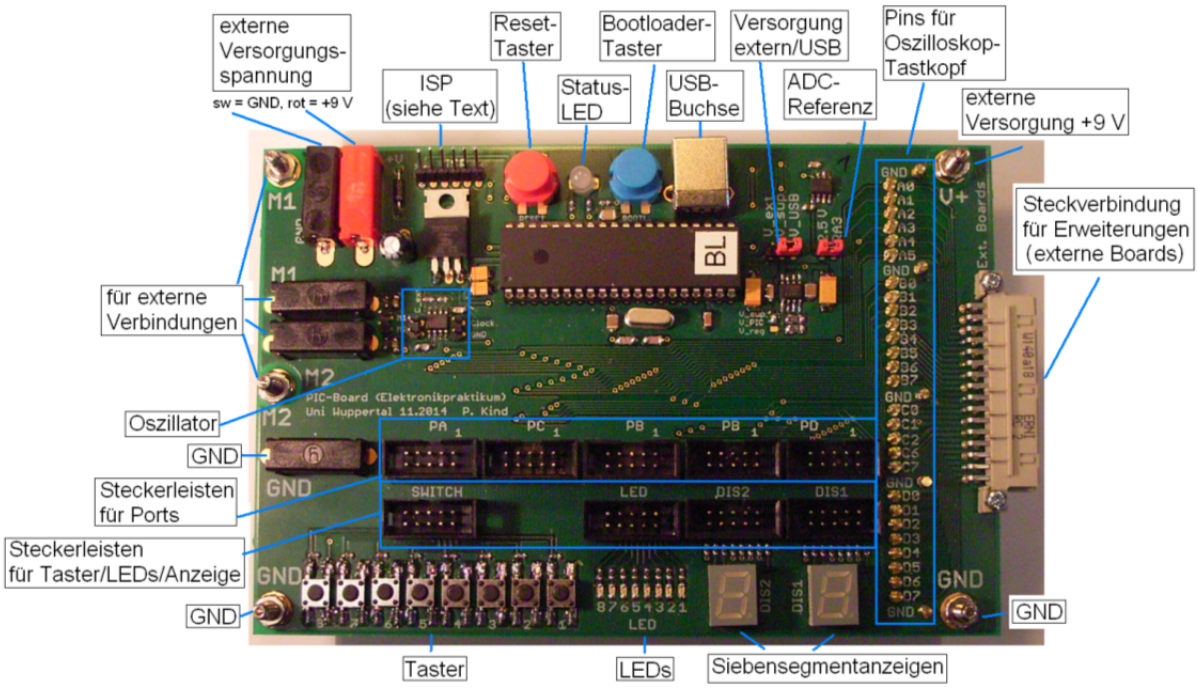


Abbildung 2: Versuchsboard[2]

Quellcode

Der Quellcode zum einfachen testen der Geschwindigkeit des Boards.

```
// Implementierung der eigenen Funktion
void test_speed_1(){
    TRISB = 0b00000000;

    while(1){
            LATB = 0b1111111;               // In dieser Schleife sollen die LEDs moeglichst
            LATB = 0b0000000;               // schnell an und aus geschaltet werden.
    }//end of while(1)
}//end of function test_speed_1()
```

Listing 1: Einfacher Geschwindigkeitstest

Versuchsdurchführung

Der Code 1 wird auf den Mikrocontroller geladen. Danach wird Port B über ein Flachbandkabel mit dem LED Port verbunden. Das Board wird an ein Oszilloskop angeschlossen und die Ausgangsfrequenz an den Pins gemessen.

Auswertung

Die LEDs leuchteten, es war kein Flackern zu erkennen. Die Messung der Frequenz mit dem Oszilloskop ergab 2,4 GHz. Der Verlauf des Signals ist in Abbildung 3 zu sehen.

[2]Abbildung entnommen von http://www.atlas.uni-wuppertal.de/~kind/ep10_14.pdf am 10.01.2015

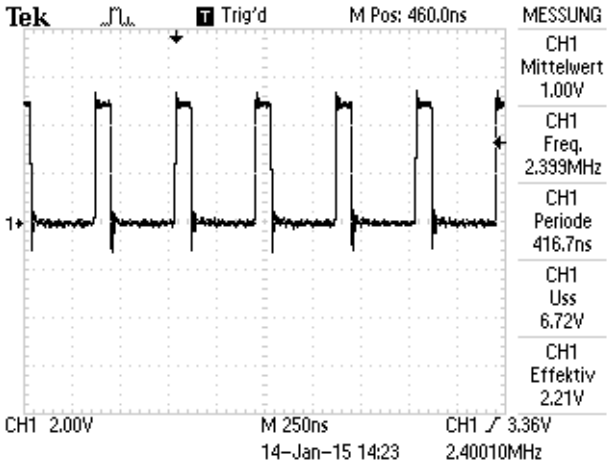


Abbildung 3: Aufnahme des Signals

2.2 Programm mit Verzögerungsbefehlen aus for-Schleifen

In diesem Versuchsteil soll eine LED mit Verzögerung ein- und ausgeschaltet werden. Die Pausen werden mit Hilfe einer for-Schleife realisiert. Um die Länge der Verzögerung einzustellen wird der Befehl Nop() verwendet. Dieser sorgt dafür, dass der Mikrocontroller für einen Taktzyklus aussetzt.

Verwendete Geräte

Es werden 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();                  // zum Verzoegern von 10 Taktzyklen
                }

                LATB = 0b1111111;

                for(i=0; i<10; i++){
                        Nop();                  // zum Verzoegern von 10 Taktzyklen
                }

                LATB = 0b0000000;
        }//end of while(1)
}//end of function test_speed_for()
```

Listing 2: Geschwindigkeitstest mit einer for-Schleife

Versuchsdurchführung

Der Code 2 wird auf den Mikrocontroller geladen. Port B wird mit dem LED Port über ein Flachbandkabel verbunden. Dann wird das Board an ein Oszilloskop angeschlossen und die Ausgangsfrequenz an den Pins gemessen.

Auswertung

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

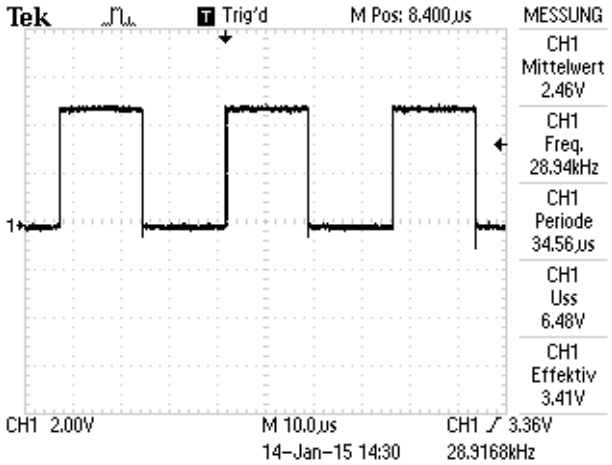


Abbildung 4: Aufnahme des Signals

2.3 Programm mit Compiler-Verzögerungsbefehlen

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

- Delay1TCY() oder Nop() delay von einem instruction cycle
- Delay10TCYx(unit) delay von unit × 10 instruction cycle
- Delay100TCYx(unit) delay von unit × 100 instruction cycle
- Delay10KTCYx(unit) delay von unit × 1000 instruction cycle
- Delay10KTCYx(unit) delay von unit × 10000 instruction cycle

Verwendete Geräte

Es werden 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;       // es wurden dierekt mehrere Verzoegerungsbefehle verwendet
            Delay10KTCYx(100);        // wurde im ersten Durchlauf auskommentiert

            LATB = 0b00000000;

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

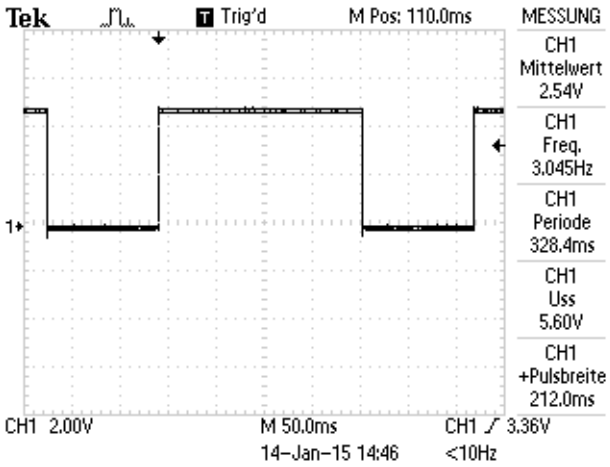
Listing 3: Geschwindigkeitstest mit Compiler-Verzögerungsbefehlen

Versuchsdurchführung

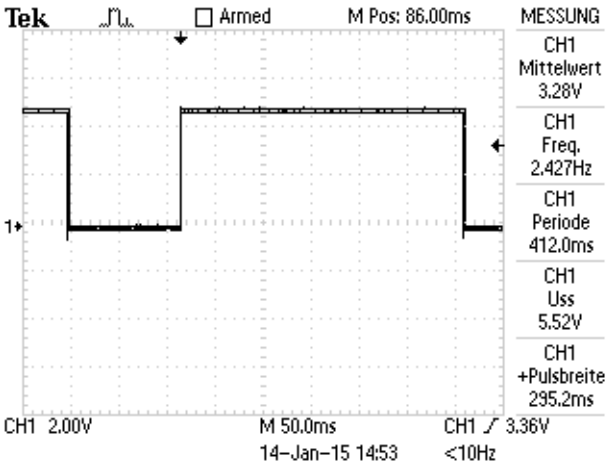
Der Code 3 wird auf den Mikrocontroller geladen. Port B wird mit Port LED, über ein Flachbandkabel verbunden. Das Board wird an ein Oszilloskop angeschlossen und die Ausgangsfrequenz an den Pins gemessen. Dann wird die in Code 3 erwähnte Zeile einkommentiert und die Frequenz mit dem Oszilloskop gemessen.

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 zweitem 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 und die Restlichen blinkten.

2.4 Programm mit fertigen Verzögerungsbefehlen

In diesem Versuchsteil soll das An- und Ausschalten der LED mit 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 folgenden Befehle (unit kann Werte von 0 bis 255) verwendet.

- delay us(unit) verzögert um unit Mikrosekunden
- delay 10us(unit) verzögert um unit $\times$ 10 Mikrosekunden
- delay ms(unit) verzögert um unit Millisekunden

Verwendete Geräte

Es werden 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;             // Quellcode war vorgegeben

        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

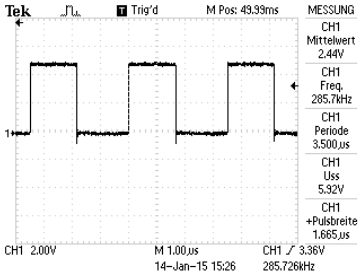
Der Code 4 wird auf den Mikrocontroller geladen und Port B wird mit dem LED Port verbunden. Dann wird das Board an ein Oszilloskop angeschlossen und die Pulsbreite der Signale an den Pins gemessen. Dieser Prozess wir mit verschiedenen Verzögerungsbefehlen wiederholt.

Auswertung

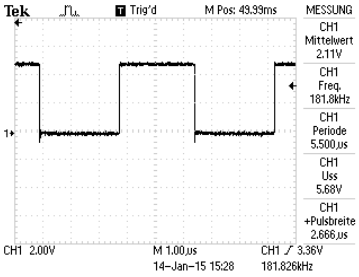
Es wurde Quellcode 4 verwendet. Der Befehl **delay_us(25)** wurde der Reihe nach durch die Folgenden ersetzt.

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

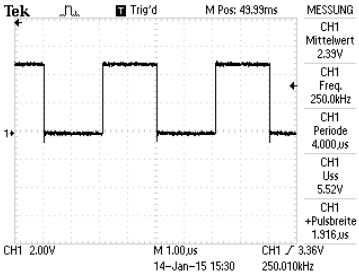
Die Frequenz wurde bei jedem dieser Befehle gemessen. Dabei ergaben sich die Werte in Tabelle 1. Die Aufnahmen der jeweiligen Signale sind in Abbildung 6 und Abbildung 7 zu sehen. Bei allen Befehlen ist die benötigte Zeit 0,6 bis 3 μs länger als gewünscht.(Bei sehr kleinen Zeiten ergeben sich große relative Abweichungen von der gewünschten Verzögerung.) Dies Begründet sich dadurch, dass die Anzahl der Zyklen, die ausgesetzt werden müssen um die gewünschte Verzögerung zu erreichen, intern berechnet wird. Ein Zyklus dauert 83,33 ns. Der letzte Befehl hat eine geringere Pulsbreite als 1ms. Bei längeren Verzögerungszeiten sind die Abweichungen zu vernachlässigen.



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

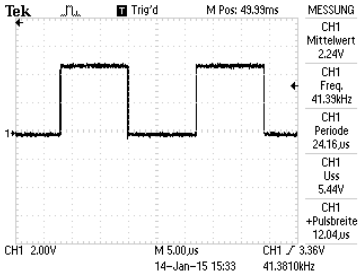


(b) Aufnahme des Signals, für delay_us(2)

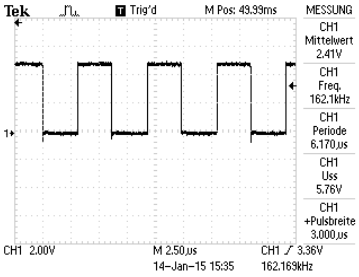


(c) Aufnahme des Signals, für delay_10us(0)

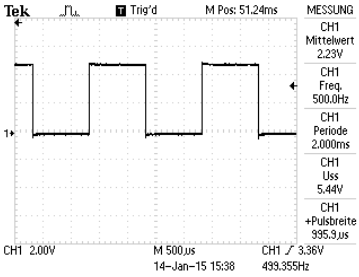
Abbildung 6: Oszilloskopaufnahmen für delay_us(1), delay_us(2) und delay_10us(0)



(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 wurden die Methoden zur Verzögerung erfolgreich untersucht. Die Vor- und Nachteile der einzelnen Möglichkeiten zur Verzögerung konnten festgestellt werden. Mit den Delay*TCY*() Befehlen lassen sich genaue Verzögerungen realisieren, jedoch muss die Anzahl der ausgelassenen Zyklen(83,33ns) aufsummiert werden, um die Gesamtverzögerung zu erhalten. Eine bestimmte Zeit lässt sich einfacher mit den delay_*() Befehlen umsetzen, welche bei sehr kurzen Verzögerungen eine große Ungenauigkeit haben, da der Mikrocontroller zusätzliche Zeit zur Berechnung braucht.

3 Benutzung der Taster

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

3.1 Einfache Tasterabfrage

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

Verwendete Geräte

Es werden 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;       // ueber die Variable i wird das Eingangssiganl an den Ausgang weitergegeben
        }//end of while(1)
}//end of function taster_1()
```

Listing 5: Einfachster Version eines Schalters

Versuchsdurchführung

Der Code 5 wird 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 Taster 4,7 und 8 leuchtete die entsprechende LED auf.

3.2 Taster steuern Speicherfunktionen

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

Verwendete Geräte

Es werden 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){   // hier wird der gesamte PORT abgefragt,
                                         // sodass das Druecken von zwei Tastern gleichzeitig nichts bewirkt
                        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){               // nun wird nurnoch der erste Taster abgefragt,
                                                        // unabhaengig von dem Zustand anderer Taster
                        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

Der Code 6 wird 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, falls Taster 1 gedrückt war. Wenn noch ein anderer Taster gedrückt war, während der erste Taster gedrückt wurde, blinkte die LED nicht. Beim Verwenden der Funktion taster_3() war auch ein schnelles Blinken der LED zu sehen falls Taster 1 gedrückt war. Es war auch ein Blinken zu sehen, falls ein anderer Taster gleichzeitig 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);          // um das Flackern zu verhindern wurden 500ms Delay eingebaut
        }//end of while(1)
}//end of function taster_4()
```

Listing 7: Tastabfrage und Verzögerung

Versuchsdurchführung

Der Code 7 wird 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 steuert der Taster einen Zähler, welcher bei jedem Tastendruck eine Stelle hoch zählen soll.

Quellcode

Quellcode zum steuern eines Zählers mit einem Taster.

```
void taster_5(){
        byte taster_alt = 0b00000000;           // taster_alt wird deklariert und definiert
        byte taster_neu = 0b00000000;           // taster_neu wird deklariert und definiert
        byte counter = 0;
        TRISB = 0b00000000;
        TRISA = 0b111111;

        while(1){
                taster_neu = PORTA;             // taster_neu wird der Zustand von PORTA zugewiesen
                if(taster_neu != taster_alt){   // falls taster_neu ungleich taster_alt
                                                // 'falls ein Taster oder mehrere gedrueckt oder losgelassen werden'
                        taster_alt = taster_neu;// setze taster_alt = taster_neu
                                                //(damit auch beim Loslassen umgeschaltet wird)
                        counter++;              // erhoehe den counter
                        LATB = counter;         // und lasse das 8-Bit-Muster von counter am Ausgang aufleuchten
                }//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;                  // eine Variable flag wird deklariert und definiert
        TRISB = 0b00000000;
        TRISA = 0b111111;

        while(1){
                taster_neu = PORTA;
                if(taster_neu != taster_alt){   // falls taster_neu ungleich taster_alt
                        flag++;                 // erhoehe flag um einen
                                                // falls flag = 255 bzw. 11111111 ist, wird flag nun auf 0 schalten
                        taster_alt = taster_neu;// und setze taster_alt = taster_neu
                        if(flag%2 == 0){        // wenn dann flag modulo 2 = 0 ist
                                counter++;      // erhoehe den counter
                                LATB = counter; // und gebe das Muster von counter aus
                        }//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 stuern eines Zählers mit einem Schalter

Versuchsdurchführung

Der Code 8 wird 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 sowie beim Loslassen der counter um einen erhöht, dies konnte man an den LEDs sehen. Bei der Implementierung von taster_6() wurde nur beim Loslassen des Tasters der counter einen hochgezählt, dies war bei den LEDs zu sehen.

3.5 Kontaktprellen der Taster – eine Lösung

In diesem Versuchsteil soll der Effekt des Kontaktprellen verhindert werden. Kontaktprellen bedeutet, dass falls der Taster-Kontakt nicht sauber schließt 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);                           // warte 10ms
                if(PORTA){                              // falls PORTA ungleich 0
                        delay_ms(10);                   // warte nochmal 10ms
                        while(PORTA){                   // solange PORTA ungleich 0 mache nichts
                        }//end of while(!PORTA)
                        counter++;                      // erhoehe danach counter um 1
                        LATB = counter;                 // und gebe das Muster von counter am Ausgang aus
                }//end of if(taster_neu != taster_alt)
        }//end of while(1)
}//end of function taster_7()
```

Listing 9: Verhindern von Kontakprellen

Versuchsdurchführung

Der Code 9 wird 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 9 wurde beim loslassen der counter um einen erhöht, was auch an den LEDs zu sehen war.

Diskussion

In diesem Versuchsteil konnten die Verwendungsmöglichkeiten, sowie verschiedene Implementierungen dieser untersucht werden. In allen Versuchsabschnitten liesen 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){    // solange an PORTA Spannung anliegt
                        counter++;       // erhoehe den counter
                        LATB = counter; // und gebe das Muster von conter am Ausgang(LEDs) aus
```

```
                }//end of while(PORTAbits.RA0)
        }//end of function oszi_1()
}//end of while(1) und gebe das Muster von counter
```

Listing 10: Zähler ohne Vorteiler

Versuchsdurchführung

Code 10 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 getaktet ist, 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 während eines Taktsignals des Oszillators mehrere Zyklen des Mikrocontrollers sehen.

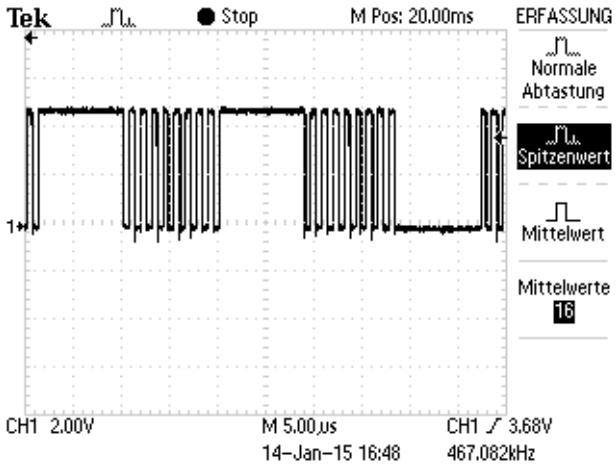


Abbildung 8: Aufnahme des Signals

4.2 Zähler mit Vorteiler

In diesem Versuchsteil soll 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 Versuchsboard, 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();                          // warte ca. 200ms
                        counter++;                              // und erhoehe dann ...
                        LATB = counter;                         //gebe den neuen Zustand aus
        }//end of while(1)
```

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

Listing 11: Zähler mit Vorteiler

Versuchsdurchführung

Code 11 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.

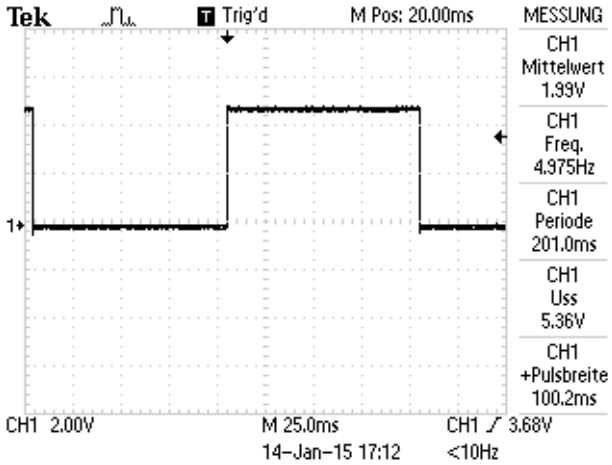


Abbildung 9: Aufnahme des Signals

Diskussion

In diesem Versuchsabschnitt konnte die Verwendungsmöglichkeit eines Oszillators beobachtet werden.

5 Verwendung des Analog-Digital-Konverters (ADC)

Der Mikrocontroller besitzt einen 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 Versuchsboard, Verbindungskabel, ein PC und ein Netzgerät verwendet.

Quellcode

Quellcode für den ADC mit LEDs.

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

        //adc initialisieren
        setup_adc_ports(AN0_TO_AN1);
        setup_adc(ADC_CLOCK_DIV_32);
        set_adc_channel(0);
        read_adc(ADC_START_ONLY);

        //Mess- und Ausgaberoutine starten
        while(1){
                delay_ms(10);                   // warte 10ms
                LATB = (read_adc(ADC_READ)>>2); //gemessene Spannung ausgeben
        }//end of while(1)
}//end of function adc_1()
```

Listing 12: adc mit LEDs

Versuchsdurchführung

Der Code 12 wird 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 halben maximalen Wert anzeigt.(Abhängig von den LEDs.)

Auswertung

Bei 0V waren alle LEDs aus, entspricht einer dezimalen 0. Bei 5V waren alle LEDs an, was einem dezimalen Wert von 255 entspricht. Dies war gleichzeitig der Maximalwert. 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 Siebensegmentanzeige ausgegeben werden.

Verwendete Geräte

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

Quellcode

Quellcode für die Ausgabe des ADC mit einer Siebensegmanetanzeige.

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

        // Porteigenschaften festlegen
        TRISD = 0b00000000;
        TRISC = 0b00000000;
        TRISB = 0b00000000;
        TRISAbits.TRISA0 = 1;

        // ADC initialisieren
        setup_adc_ports(AN0_TO_AN1);
        setup_adc(ADC_CLOCK_DIV_32);
        set_adc_channel(0);
        read_adc(ADC_START_ONLY);

        // Mess- und Ausgaberoutine
        while(1){
                delay_ms(10);

                value = read_adc(ADC_READ)>>2;

                valH = value>>4;                        // die ersten 4 Bits abspeichern
```

```
                valL = (value & 0b00001111)>>4;          // die letzten 4 Bits abspeichern

                //Spannung mit der Siebensegmentanzeige ausgeben
                LATD = dec_7seg[valH];
                LATC = dec_7seg[valL];

        }//end of while(1)
}//end of function adc_2()
```

Listing 13: adc mit Siebensegmentanzeige

Versuchsdurchführung

Der Code 13 wird 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 halben maximalen Wert anzeigt.

Auswertung

Bei 0V wurde der Wert 0 auf der Siebensegmentanzeige angezeigt. Der Maximale Wert F wurde zum ersten mal bei 4,6V und damit auch bei 5V 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 die Funktion implementiert werden, mit der per Tasterdruck zwischen den Kanälen AN0 und AN1 umgeschaltet werden kann.

Verwendete Geräte

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

Quellcode

Quellcode für ADC Messung mit Umschaltfunktion. Der Wechsel zwischen den Channels wird über die flag-Variable gesteuert. In Abhängigkeit davon wird der 0-te oder 1-te Channel ausgelesen.

```
void adc_3(){
        // Variablen deklarieren und definieren
        byte flag = 0;

        // Porteigenschaften festlegen
        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;                // flag wechselt, falls der Taster gedrueckt wird
                        }//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);        // abhaengig von flag wird der ADC-Channel gewechselt
                }

                delay_ms(100);

                LATB = read_adc(ADC_READ)<<2;            // Ausgabe des Spannungswertes
```

```
                delay_ms(1000);          // Delay von 1s

        }//end of while(1)
}//end of function adc_3()
```

Listing 14: ADC mit Umschaltfunktion

Versuchsdurchführung

Der Code 14 wird 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 wird, gewechselt. Aufgrund der langen Verzögerung musste immer einen 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(){
        // Variablen initialisieren
        byte val;

        // Porteigenschaften festlegen
        TRISB = 0b00000000;
        TRISAbits.TRISA0 = 0b1;

        // ADC konfigurieren
        setup_adc_ports(AN0_TO_AN1);
        setup_adc(ADC_CLOCK_DIV_32);
        set_adc_channel(0);
        read_adc(ADC_START_ONLY);

        // Mess- und Ausleseroutine starten
        while(1){
                delay_ms(10);
                val = (read_adc(ADC_READ)>>7);  // es werden nur die letzen drei bits verwendet,
                                                // da sich darus 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

Der Code 15 wird 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 trotzdem lauffähig sein und aus der LED-Anzeige eine Balkenanzeige machen.

6 Fazit

Im Versuch wurden verschiedene Einsatzmöglichkeiten für Mikrocontroller untersucht, sowie einige Varianten der Implementierung. Die implementierten und getesteten Funktionen haben alle wie erwartet funktioniert. Mikrocontroller sind aufgrund ihrer Programmierbarkeit und der Möglichkeit verschiedene Bausteine anzuschließen, vielseitig einsetzbar.