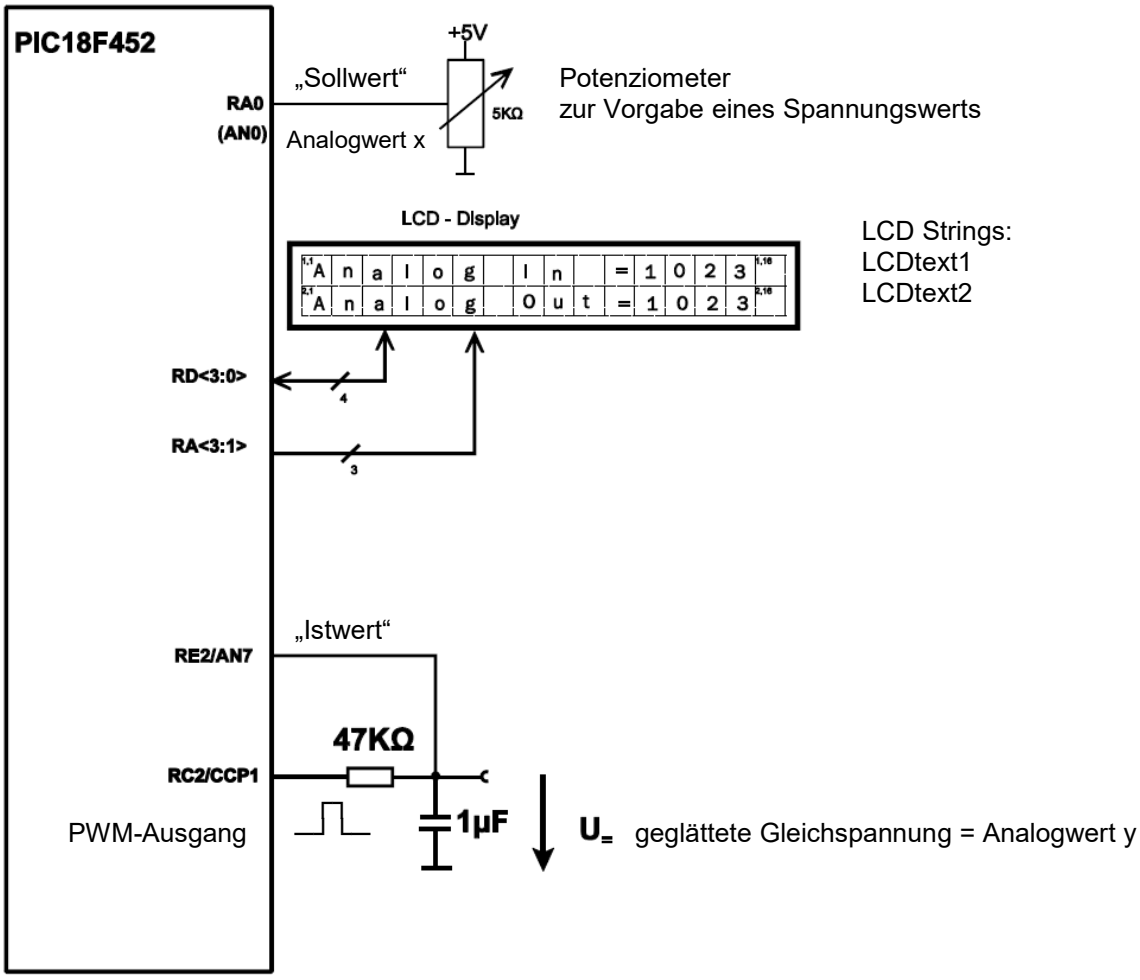
Laborversuch PIC2 PWM (Pulsweitenmodulation)

Version 5.12.2023

In diesem Versuch soll eine **Digital-Analog-Umsetzung** mittels des **Pulsweiten-Modulationsvefahrens (PWM)** realisiert werden.

Die folgende Schaltungsanordnung mit dem Mikrocontroller 18F452 ist gegeben:



Aufgabe

Entwickeln und testen Sie ein Programm in C, welches die an dem Analog-Eingang AN0 (Port RA0) angelegte Spannung mit dem 10-Bit A/D-Umsetzer digitalisiert und am Digitalausgang RC2/CCP1 als Duty Cycle eines pulsweitenmodulierten Rechtecksignals ausgibt. Die Pulsbreite ist dabei proportional zum Analogwert x, der durch ein Poti vorgegeben wird.

Leitet man unter Zielsetzung einer einfachen D/A-Umsetzung dieses PWM-Signal durch einen Tiefpass mit hinreichend großer Zeitkonstante, erhält man wieder eine Analogspannung y, die der vorgegebenen Analogspannung x sehr nahe kommt. Die Zeitkonstante des RC-Gliedes sollte mindestens das zehnfache der Periodendauer des PWM-Signals betragen, damit eine gut geglättete Gleichspannung am Ausgang des RC-Gliedes entsteht.

Die Zeitkonstante des vorliegenden RC-Glieds ist $\tau = R*C = 47ms$

Die PWM-Periodendauer errechnet sich wie folgt:

$$PWM_{Period} = [(PR2)\ +1\ ]\ *\ 4\ *\ T_{osc}\ *\ (TMR2_{Prescale\ Value})$$

Timer 2 Period Register

(Siehe auch PWM-Beschreibung aus den PIC Datenblättern von Microchip, Seite 111, 112, 117, 122, 123)

Bei einem Vorteiler von 1 ergibt sich die Periodendauer zu: _____

Bei einem Vorteiler von 16 ergibt sich die Periodendauer zu: _____

Vorgaben zur Initialisierung des A/D-Umsetzers:

- Umsetzungstakt mit **Fosc/32**
- Format linksbündig
- AN0..AN7 als Analogkanäle

Anmerkung: Da die Analogeingänge AN1..AN3 auf den vom LCD benutzten Anschlüssen RA1..RA3 liegen, werden vor dem Aufruf der LCD-Funktionen AN1..AN7 zu Digitalkanälen umkonfiguriert und danach wieder zu Analogkanälen.

Aufgabe zur Simulation Die Simulation sollte schon zuhause durchgeführt werden.

Das vorgegebene Projekt mit Programmvorlage ist sowohl für die Simulation (einschließlich LCD-Text) als auch für den Gebrauch mit Hardware und LC-Display geeignet. Die Umschaltung geschieht in einer einzigen Zeile:

Betrieb im Simulator (MPLAB SIM):

`#define Simulator` ⇨ Standardvorgabe

Hierbei sind alle LCD-Ausgaben des Quelltexts quasi auskommentiert, sonst würde die Simulation darin hängen bleiben.
Die LCD-Ausgabe wird simuliert durch Anzeige der mit sprintf() erzeugten Strings **LCDtext1** und **LCDtext2** in einem Watch-Fenster.

Betrieb mit der Hardware (MPLAB ICD3):

// `#define Simulator` ⇨ auskommentiert für Hardwaregebrauch / **LCDtext1** und **LCDtext2** werden an die 2 Zeilen des LCD geschickt.

Diese Umschaltung ist möglich durch **bedingte Kompilierung**: Durch die Anweisungen `#ifdef` oder `#ifndef` d.h. „if (not) defined“ und `#else` / `#endif` werden die dazwischen befindlichen Quelltextzeilen in Abhängigkeit vom Vorhandensein oder nicht Vorhandensein eines beliebigen, durch `#define` definierten Schlüsselworts für die Kompilierung herein oder heraus genommen (wie auskommentiert).
Die LCD-Ausgaben sind fertig vorgegeben.

Ihre Aufgabe:
Tragen Sie, wie in den Kommentaren angedeutet, alle nötigen Registerinitialisierungen ein, sowie die Übertragung des Analogwerts in den Duty Cycle der PWM. Testen Sie zunächst mit **Prescaler 1**, d.h. kein Vorteiler. Wenn Sie alles richtig gemacht haben, sehen Sie im **Logic Analyzer Fenster** nach einigen Testmustern der PWM den Durchlauf einer kompletten **Linksdrehung des Potenziometers.** Die Analogwerte können wir im Logic Analyzer leider nicht darstellen. Wenn Sie auf Prescaler 16 umstellen, kann aufgrund der längeren Zeit nur ein Teil des Gesamtverlaufs im Timing dargestellt werden. Achten Sie bei der Implementierung auch besonders auf die richtige Übertragung der beiden **niederwertigen Bits DC1B1 + DC1B0** des Duty Cycle, die nicht viel zum Gesamtwert beitragen, mit denen aber erst die volle Auflösung (und auch der Maximalwert) von 10 Bit erreicht wird.

Als Ersatz für die Eingangsspannungen an den Analogeingängen AN0 (x) und AN7 (y) wird bei der Simulation eine Register Injection aus der Textdatei „`Stimulus ADRESL pic2pwm.txt`“ in das ADRES-Register durchgeführt, d.h. beim Abschluss jeder A/D-Umsetzung wird zeilenweise ein neuer Hexwert aus der Datei gelesen und in ADRES eingetragen.

Nur, wenn es Sie näher interessiert: Folgende Wertemuster enthält die Textdatei „`Stimulus ADRESL pic2pwm.txt`“:

- Wenn Sie in diese Datei schauen, sehen Sie aufeinander folgende Wertepaare, die jeweils x und y entsprechen.
- Da später durch die Belastung des hochohmigen Tiefpassausgangs zu erwarten ist, dass es zu einem leichten Spannungsverlust bei der Messung von y kommt, sind die y-Werte auch für die Simulation mit angenommenen Verlusten versehen. Für den praktischen Einsatz einer solchen Schaltung empfiehlt es sich daher, ggf. ausgangsseitig einen Impedanzwandler vorzusehen.
- 1. Teil: einzelnes Bit auf 1 gesetzt, diese 1 wandert zur Stellenwertkontrolle vom MSB bis zum LSB
  und damit Test auf vollständige Übertragung des AD-Werts inklusive der niederwertigen Bits 1+0 in den Duty Cycle
  2. Teil: Linksdrehung des Potentiometers (Maximum bis Minimum)
  Die Abschnitte sind durch gut erkennbare 0%-Marken des Duty Cycle getrennt.
- Die Register Injection ist im Stimulus Workbook „`Stimulus ADRESL.sbs`“ definiert.

Damit bei der Simulation der Systemtakt dem der Hardware entspricht, geben Sie die **Quarzfrequenz** ein unter **>Debugger >Settings...** Processor Frequency: von 20MHz (Vorgabe) umstellen auf **4MHz**

Gehen Sie Ihr Programm erst einzelschrittweise durch, um die Registerinhalte kontrollieren zu können. Später durchlaufen Sie größere Zyklen durch geeignetes Platzieren eines Haltepunkts. Schließlich setzen Sie den Haltepunkt auf den `Nop()`-Befehl (NOP = No Operation) nach der Abfrage des Hexwerts 0x1FA. Damit erreichen Sie, dass genau nach einem Lesedurchlauf der Register Injection Datei das Programm anhält, womit die Aufgabe gelöst ist:

Hier bei Nop Haltepunkt setzen ➡

```
#else   // Simulation: PWM-Periode abwarten + Haltepunkt bei bestimmtem Analogwert ermöglichen
        while(!PIR1bits.TMR2IF);    // Eine Periode der PWM (Timer 2) abwarten
        PIR1bits.TMR2IF=0;          // bis der nächste Analogwert gelesen wird.

        if(y==0x1FA){       // Letzter aus "Stimulus ADRESL pic2pwm.txt" zu lesender AD-Wert
            Nop();      // <-- hier einen Haltepunkt zum Anhalten nach einem Datenzyklus setzten!
        }                   // Ohne Haltepunkt wird die Injaktionsdatei zyklisch wiederholt gelesen.
#endif
```

Aufgabe an der Hardware

Für die Screenshots am Oszilloskop ist ein **USB-Stick** (FAT- oder FAT32-formatiert) hilfreich.

Um das Programm auf der Hardware mit LCD-Funktionen auszuführen, kommentieren Sie folgende Zeile aus:

```
// #define Simulator        // zum Gebrauch mit Hardware auskommentieren
```

Nach Umstellen des Debuggers auf ICD3 kann das Programm direkt auf der Hardware getestet werden.

Bevor Sie mit der Oszilloskopmessung beginnen, können Sie durch Aufstecken des Lautsprecher-Jumpers J9 sehr schnell kontrollieren, ob überhaupt ein PWM-Signal ausgegeben wird. Durch Drehen des Potis sollten nicht nur die Anzeigewerte beeinflusst werden, sondern auch die Phase (und Stärke) des hörbaren Tons.

Wenn das funktioniert, schließen Sie das Oszilloskop folgendermaßen an:

GND ⇨ Masse-Strippe
RC2 (digital, PWM-Ausgang) ⇨ **CH1** **10x DC** Testweise Lautsprecher-Jumper J9 aufstecken, später abziehen!
RE2 (analog, DA-Ausgang) ⇨ **CH2** → Einstellungen nach Tabelle s.u.

Oszilloskopmessungen

Für die Messungen darf der Lautsprecher-Jumper nicht aufgesteckt sein, da durch die Last die Ausgangsspannungen verfälscht werden.

Führen Sie das Programm mit jeweils mit Vorteiler 1 und 16 aus und oszilloskopieren Sie den PWM-Ausgang und den Analog-Ausgang nach folgenden Anweisungen:

Tragen Sie im **MEASURE-Menü** (automatische Messung) folgende Messungen ein:

CH1 Periodendauer
CH1 +Pulsbreite (=High-Zeit)
CH1 Maximalspannung (=High-Pegel)
CH2 Mittelwert (bei DC = Ausgangsspannung der DA-Wandlung per PWM)
CH2 U_{SS} (bei AC = Welligkeit der Gleichspannung)

Wählen Sie einen **AD-Wert** von etwa mittlerer Größe (Wert ca. zwischen 250 und 650),
den Sie **notieren** und für alle Oszilloskop-Messungen konstant lassen.

Speichern Sie auf einem USB-Stick folgende Bildschirmaufnahmen:
Mit **Prescaler 1** und mit **Prescaler 16** jeweils einen Screenshot zur Messung des **Analogwerts** und dessen **Schwankung** am Tiefpassausgang (zusammen 4 Aufnahmen):

	Prescaler 1	**Prescaler 16**
CH2: Eingangskopplung (CH2 Menü) und Tastkopf-Einstellung	DC 10x Aufnahme 1	DC 10x Aufnahme 2
	AC 1x Aufnahme 4	AC 1x Aufnahme 3

Beachten Sie bitte:
1.) dass für die Messung der richtigen Spannung die Tastkopfeinstellung (**1x/10x**) von CH2 am Tastkopf und im CH2-Vertikal-Menü übereinstimmen müssen!

2.) dass die Messungen (**MEASURE**-Menü) eingeblendet sind!

3.) dass der digitale PWM-Ausgang RC2 auf CH1 unverändert immer mit 10x und DC-Kopplung gemessen wird.

Legen Sie die Aufnahmen im Unterverzeichnis "Oszilloskop" der Projektvorlage ab.