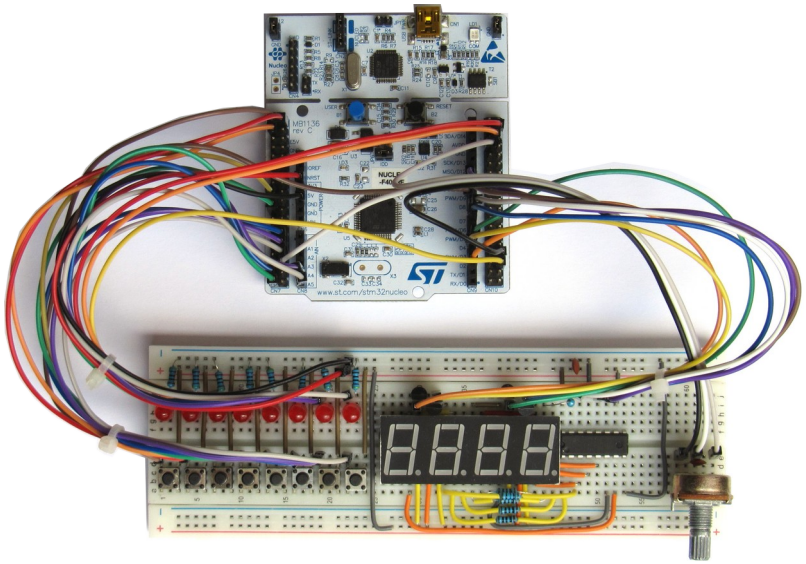
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

Mikroprozessoren-Labor



Versuch 2, 7-Segment-Anzeige und ADC

Versuchsdurchführung

Stand: 28. November 2020

I. Aufgaben

1 Modellierung der mehrstelligen Anzeige

Die mehrstellige Anzeige wird durch die Klasse `CSevenSegmentDisplay` modelliert. Die vorbereitete Klassendefinition und eine unvollständige Implementierung ist im Startprojekt enthalten.

Das nachfolgende UML-Klassendiagram illustriert das Zusammenspiel von `CSevenSegment-Display` mit den anderen Klassen.

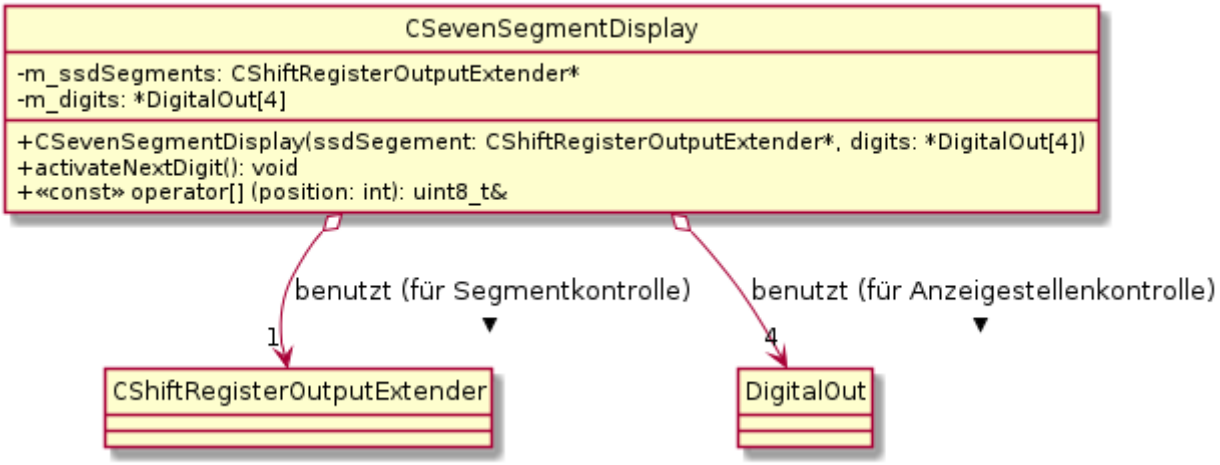


Bild 1: Zusammenspiel von `CSegmentDisplay` mit anderen Klassen

Vervollständigen Sie die Initialisierung der Klassenvariablen `CSevenSegmentDisplay::patterns` mit dem von Ihnen in der Vorbereitung erstellten Muster.

1.1 Methode `activateNextDigit`

Implementieren Sie die Methode `activateNextDigit` entsprechend den Hinweisen in der Implementierungsdatei der Klasse `CSevenSegmentDisplay`.

Erstellen Sie zum Testen eine Funktion `task3`. Definieren Sie in der Funktion, wie schon bei `task2`, ein Objekt `ssdSegments` vom Typ `CShiftRegisterOutputExtender`. Zusätzlich definieren Sie ein Objekt `ssd` vom Typ `CSevenSegmentDisplay`. Setzen Sie die von `ssd` anzuzeigenden Ziffern auf die letzten vier Stellen Ihrer Matrikelnummer.

Definieren Sie weiterhin, entsprechend dem Vorgehen in Labor 1, ein Objekt `digitTimer` vom Typ `CPolledTimer` mit einer Taktperiode von 5 ms.

In der Endlosschleife prüfen Sie, ob der Timer abgelaufen ist. Wenn ja, rufen Sie die Methode `activateNextDigit` von `ssd` auf. Kontrollieren Sie, dass Sie eine flackerfreie Darstellung der letzten vier Stellen Ihrer Matrikelnummer auf der Anzeige sehen. Falls das nicht der Fall sein sollte

und Sie die Ursache nicht finden können, bitten Sie einen Betreuer im Labor, sich Ihren Code anzuschauen.

1.2 Ergänzung Dezimalpunkt

Ergänzen Sie in der Klasse `CSevenSegmentDisplay` eine Methode `setDecimalPoint(int digit)`, deren Aufruf dafür sorgt, dass an der angegeben Anzeigestelle (0 ganz rechts, 3 ganz links) zusätzlich der Dezimalpunkt leuchtet (vergessen Sie nicht die Dokumentation in der Header-Datei!). Wird als Argument ein Wert außerhalb des zulässigen Bereichs angegeben, wird kein Dezimalpunkt angezeigt.

Die Methode speichert zunächst nur den übergebenen Wert in einem neuen, von Ihnen zu ergänzenden Attribut der Klasse. Die eigentliche Funktionalität (Anzeige des Dezimalpunkts) implementieren Sie durch Änderungen in der Implementierung der Methode `activateNextDigit`.

Testen Sie Ihre Methode, indem Sie sie in `task3` aufrufen.

1.3 Komfort-Anzeige für Ganzzahlwerte

Ergänzen Sie in der Klasse `CSevenSegmentDisplay` eine Methode `void setValue(int value)`, die den angegebene Wert anzeigt. Ist der Wert negativ oder größer 9999, soll „EEEE“ angezeigt werden. Es wird kein Dezimalpunkt angezeigt.

Um die Methode einfach verwenden zu können, überladen Sie außerdem den Operator „`CSevenSegmentDisplay& operator= (int value)`“ so, dass er die Methode `setValue(int)` aufruft.

Die Methode muss nicht extra getestet werden, da sie im Rahmen der Implementierung der folgenden Methode getestet wird.

1.4 Komfort-Anzeige für Fließkommazahlen

Ergänzen Sie in der Klasse `CSevenSegmentDisplay` eine Methode `void setValue(float value)`, die den angegeben Wert mit korrekt gesetztem Dezimalpunkt und unter Ausnutzung aller Stellen anzeigt (d. h. keine führenden Nullen)[1]. Für Werte kleiner 0,001 wird 0,000 angezeigt, für negative Werte oder Werte größer „9999,“ soll „EEEE“ angezeigt werden. Implementieren Sie die korrekte Positionierung des Dezimalpunkts mit einer Schleife (nicht mit drei „`if`“ Bedingungen)! Verwenden Sie bei der Implementierung die Methode `setValue(int)` (bzw. den entsprechenden Zuweisungsoperator).

Lösungsansatz: Stellen Sie sich vor, Sie würden nur den Ganzzahlanteil des Wertes anzeigen, d.h. der Dezimalpunkt ist an Position 0 (ganz rechts). Woran können Sie dann erkennen, dass der angezeigte Wert führende Nullen hat? Überlegen Sie, was Sie an den angezeigten Ziffern und der Position des Dezimalpunktes verändern müssen, um zunächst eine führende Null weg zu bekommen. Dann wiederholen Sie die Prüfung auf führende Nullen. Verwenden Sie im Labor nicht mehr als eine halbe Stunde auf diese Aufgabe. Wenn Sie keine Lösung finden, Stellen Sie die Zahl mit dem

1 Ein letztes Mal die explizite Erinnerung: Vergessen Sie nicht die Dokumentation in der Header-Datei!

Dezimalpunkt ganz links dar (darstellbar sind dann Werte zwischen 0,0000 und 9,999) und denken Sie bei der Nachbereitung weiter über eine Lösung nach.

Um die Methode `setValue(float value)` einfach verwenden zu können, überladen Sie außerdem den Operator „`CSevenSegmentDisplay& operator= (float value)`“ so, dass er die Methode `setValue(float)` aufruft.

Ergänzen Sie zum Testen eine Funktion `task4`, die zunächst die gleichen Objekte wir `task3` erzeugt. Ergänzen Sie dann einen weiteren `CPolledTimer` mit Namen `showNext` und einer Periode von einer Sekunde. Immer wenn dieser Timer abgelaufen ist, setzen Sie den nächsten anzuzeigenden Wert aus der Folge 0,0001234; 0,001234; 0,01234; 0,123; 1,234; 12,34; 123,4; 1234,0; 12340,0. D. h. Sie starten mit dem Wert 0,0001234, multiplizieren diesen Wert bei jeder Iteration mit 10 und setzen ihn wieder auf 0,0001234, nachdem Sie 12340,0 angezeigt haben (bzw. versucht haben, 12340,0 anzuzeigen).

✎ **Schriftliche Aufgabe D-1**: Begründen Sie (vollständige Sätze!), warum es sinnlos ist, den Parameter von `setValue` als `double` zu deklarieren.

2 Analog-Digital Konverter

2.1 Direkte Anzeige des gemessenen Wertes

Erstellen Sie eine Funktion `task5` als Kopie von `task3`. Entfernen Sie das explizite Setzen der Werte der Anzeigestellen und des Dezimalpunkts, so dass nur die Definition von `ssd` (und der dafür erforderlichen Objekte), die Definition von `digitTimer` und die Endlosschleife mit der Abfrage des Timers und der Aktualisierung von `ssd` übrig bleiben.

Ergänzen Sie vor der Endlosschleife die Definition eines Objekts `poti` vom Typ `AnalogIn` mit einer Referenzspannung von 3,3 V. Definieren Sie einen zusätzlichen `CPolledTimer` mit Namen `measurementTimer` und einer Taktperiode von 20 ms.

Weisen Sie in der Endlosschleife nach Ablauf des Timers `measurementTimer` der Anzeige die am Potentiometer gemessene Spannung zu.

Testen Sie, dass Sie mit Drehen des Potentiometers Spannungswerte zwischen 0 und 3,3 (ca.) angezeigt bekommen.

2.2 Grenzwert-Überwachung

Ergänzen Sie den in der Vorlesung vorgestellten Code zur Überwachung eines unteren (25%) und oberen Grenzwerts (75%) mit Hilfe der ADC-Hardware und Anzeige des Unter- bzw. Überschreitens mit LED 0.

2.3 Messwert glätten

Bei der direkten Anzeige des gemessenen Wertes dürfte die Anzeige mindestens an der letzten Stelle stark schwanken. Das wird durch das Rauschen auf der Messleitung verursacht. Der Wert muss daher durch einen Filter geglättet werden.

2.3.1 Gleitender Mittelwert

Der gleitende Mittelwert wird über die letzten N Messwerte gebildet. Dazu werden N Messwerte gespeichert. Jeder neue Messwert ersetzt den jeweils ältesten Messwert. Bei der Abfrage des gefilterten Wertes wird die Summe aller Werte gebildet und durch N geteilt.

Ein effizienterer Algorithmus besteht darin, das Array mit 0-Werten zu initialisieren und zusätzlich die Summe aller Werte im Array in einer Variablen zu speichern. Liegt ein neuer Wert vor, wird der Wert des ältesten Eintrags im Array von der Summe subtrahiert. Der neue Wert wird zur Summe addiert und im Array statt des ältesten Wert eingetragen. Bei der Abfrage des gefilterten Werts muss damit nur die immer aktuell gehaltene Summe durch N geteilt werden. Damit keine kumulierenden Rundungsfehler auftreten, muss der Typ der Variablen, die die Summe speichert, mindestens die N-fache Genauigkeit des Typs der Einträge im Array haben. Werden die Werte im Array beispielsweise als `uint16_t` gespeichert und die Summe als `uint32_t`, könnte das Array bis zu 2^{16} Werte enthalten.

Implementieren Sie die Klasse `CMovingAverageFilter` entsprechend dem nachfolgenden Klassendiagramm, mit den Typen wie oben beschrieben.

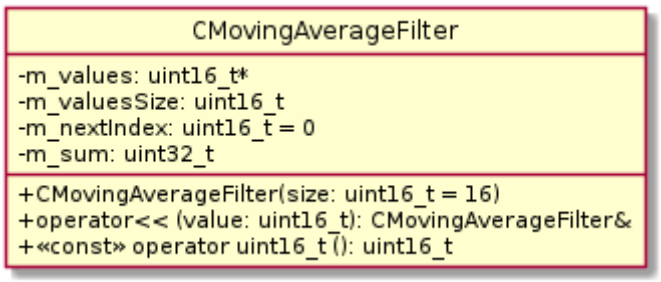


Bild 2: Klasse `CMovingAverageFilter`

Der Konstruktor erzeugt einen neuen Filter, der den gleitenden Mittelwert über N Werte bildet. Mit dem überladenen <<-Operator werden neue Werte in den Filter hinein gegeben, der überladene `operator uint16_t ()` liefert den gefilterten Wert (d. h. den gleitenden Mittelwert). Das Attribut `m_nextIndex` zeigt gleichzeitig auf den ältesten Wert und auf die Stelle, an der ein neuer Wert gespeichert werden muss.

✎ **Schriftliche Aufgabe D-2**: Speicher in einem Mikrocontroller ist endlich. Berechnen Sie mit Hilfe des Datenblatts, wie viele Werte Sie in Ihrem gleitenden Mittelwertfilter maximal speichern könnten, wenn Ihnen das gesamte RAM ausschließlich für diesen Zweck zur Verfügung stünde.

✎ **Schriftliche Aufgabe D-3**: Häufig gibt es im Umfeld der Embedded-Programmierung Vorgaben, dass Puffergrößen Zweierpotenzen sein sollen. Welchen arithmetischen Ausdruck in Ihrer Implementierung könnten Sie durch welche Bit-Operation ersetzen (und damit effizienter machen) wenn die Größe des Puffers immer eine Zweierpotenz wäre?

Erstellen Sie eine Funktion `task6` als Kopie von `task5`. Definieren Sie vor der Endlosschleife einen gleitenden Mittelwertfilter mit Namen `filter`. Weisen Sie in der Endlosschleife den gemessenen Wert nicht mehr direkt der Anzeige zu sondern übergeben Sie ihn an den Filter und aktualisieren die Anzeige mit dem neuen, gefilterten Wert. Verwenden Sie, um unnötige Fließkommaarithmetik zu vermeiden, den von `AnalogIn::read_u16()` gelieferten Wert und rechnen Sie den berechneten gleitenden Mittelwert auf eine Fließkommazahl zwischen 0 und 3,3 um.

Experimentieren Sie mit verschiedenen Werten für N, d. h. bilden Sie den gleitenden Mittelwert über eine unterschiedliche Anzahl von Werten.

✎ **Schriftliche Aufgabe D-4**: Geben Sie an, bei welchem Wert für N die Anzeige bei Ihnen stabil wird.

2.3.2 Exponentielle Glättung

Ein alternatives Verfahren für die Glättung des Messwerts ist die exponentielle Glättung (ein ausführliche Beschreibung finden Sie z. B. in der Wikipedia: https://de.wikipedia.org/wiki/Exponentielle_Gl%C3%A4ttung).

Das Grundprinzip ist sehr einfach. Die in der Wikipedia angegebene Formel ist für das Verständnis etwas unglücklich. Wenn man sie umformt erhält man:

$$y_t^* = y_{t-1}^* + \alpha \cdot (y_t - y_{t-1}^*)$$

In Worten: der neue geglättete Wert ergibt sich aus dem alten geglätteten Wert plus der Differenz aus neuem und altem Wert gewichtet mit dem Faktor α. Je größer der Faktor ist, desto mehr „Einfluss“ hat der neue Wert auf den geglätteten Wert.

Implementieren Sie die Klasse `CExponentialFilter` entsprechend dem nachfolgenden Klassendiagramm, mit den Typen wie oben beschrieben.

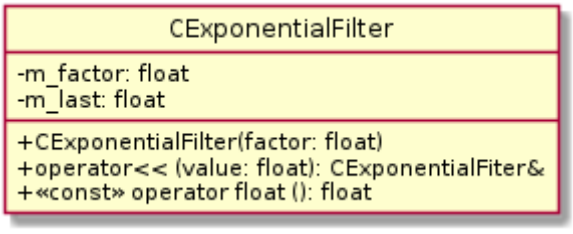


Bild 3: Klasse `CExponentialFilter`

Der Konstruktor erzeugt einen neuen Filter mit dem angegebenen Wert für den Faktor (α). Mit dem überladenen <<-Operator werden neue Werte in den Filter hinein gegeben, der überladene `operator float ()` liefert den geglätteten Wert.

Erstellen Sie eine Funktion `task7` als Kopie von `task6`. Ändern Sie den Typ des Filters. Verwenden Sie jetzt wieder den Messwert „in Volt“ als Eingabewert für den Filter.

Experimentieren Sie mit verschiedenen Werten für den Faktor.

- **Schriftliche Aufgabe D-5**: Geben Sie an, bei welchem Wert für den Faktor die Anzeige bei Ihnen stabil wird.