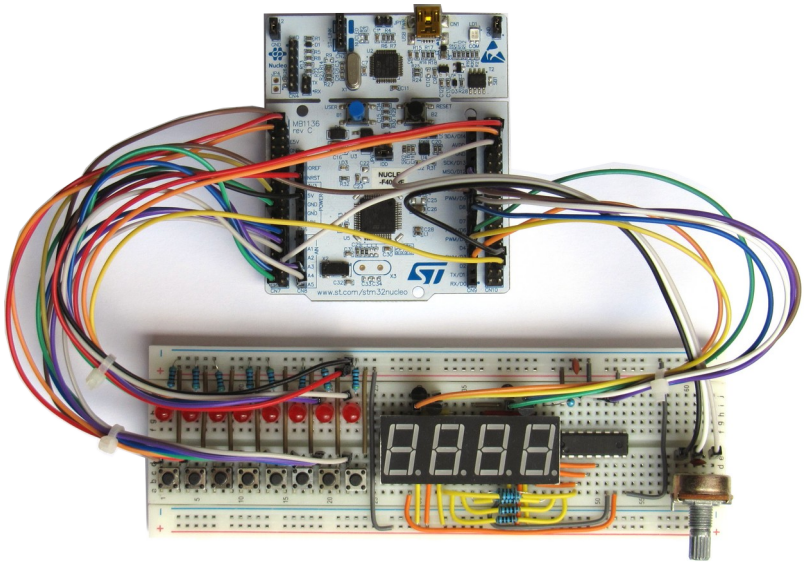
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

Mikroprozessoren-Labor



Versuch 4, Timer und Interrupts

Versuchsvorbereitung

Stand: 4. Februar 2021

I. Versuchsbeschreibung

In diesem Versuch soll ein Hardware-Timer für die Erzeugung eines PWM-Signals eingesetzt werden.

Danach wird das Multiplexing der 7-Segment-Anzeige auf die Ansteuerung durch einen Hardware-Timer umgestellt.

II. Aufgaben

1 Hardware-generiertes PWM-Signal

In der Vorlesung wurde der Timer 3 für die Generierung eines PWM-Signals konfiguriert. Dabei wurde der Code schrittweise als eine Funktion zusammengestellt, was für die Einführung nahe lag, aber für die Verwendung in einem Projekt zu wenig strukturiert ist.

1.1 Verbessern der Code-Struktur

rüdimentäre = sehr einfach und deshalb unvollständig

In einem ersten Schritt soll der in der Vorlesung erarbeitete Code strukturiert werden. Das vorbereitete Laborprojekt enthält die Klassendefinition (und rudimentäre Implementierung) der Klasse `CPwmGenerator1`. Kopieren Sie den auf der Folie „Output Compare – PWM1“ aus der Vorlesung zum Timer dokumentierten Code in die privaten Konfigurationsmethoden der Klasse. pg 32 in der Folie Timer

Wie in der Vorlesung greifen Sie im Folgenden direkt auf die GPIO-Pins mit Hilfe der CMSIS[1] `GPIO_TypeDef`[2] zu.

Wenn Sie anschließend im vorbereiteten Laborprojekt die Funktion `task1` aufrufen, sollten Sie das gleiche Verhalten beobachten können wie im Vorlesungsbeispiel.

1.2 Einstellen des Tastgrads

Natürlich ist das Generieren eines Signals mit festem Tastgrad (engl. duty cycle) ein wenig langweilig. Deshalb soll in diesem Aufgabenteil die Klasse so erweitert werden, dass der Tastgrad einstellbar ist.

Erstellen Sie eine neue Klasse `CPwmGenerator2`. Sie können die Klasse als Kopie von `CPwmGenerator1` erstellen, müssen aber darauf achten, dass Sie die Kommentare anpassen.

Ergänzen Sie in der neuen Klasse eine zusätzliche Methode `void setDutyCycle(uint8_t percent)`. Wie aus dem Namen leicht ersichtlich, stellt die Methode den Tastgrad ein. Der Wert des Arguments muss zwischen 0 und 100 liegen, Werte größer 100 werden wie 100 behandelt.

Die Implementierung der neuen Methode sollte offensichtlich sein. Sie muss `CCR1` einen Wert zuweisen, der in dem gewünschten Taktgrad resultiert. Verwenden Sie die Methode auch in den im Konstruktor aufgerufenen Konfigurationsmethoden, um den Tastgrad eines neu erzeugten `CPwmGenerator2` auf 0 zu setzen (und damit `CCR1` zu initialisieren).

Testen Sie die neue Klasse, indem Sie eine Funktion `task2` erstellen. In der Funktion erzeugen Sie vor der `while`-Schleife ein Objekt vom Typ `CPwmGenerator2`. In der `while`-Schleife kopieren Sie wie bei `task1` den Wert der Bits 15 bis 10 des Zählregisters von Timer 3 auf die LEDs 5 bis 0. Zusätzlich lesen Sie den Wert des ADC, glätten ihn mit Ihrem exponentiellen Glättungsfilter und

1 In Vorlesung 3 für den Zugriff auf die ADC-Register vorgestellt.
2 Die zu `ADC_TypeDef` analoge Struktur für den Zugriff auf die GPIO-Rgeister.

verwenden den gefilterten Wert, um den Tastgrad zwischen 0% (Potentiometer am linken Anschlag) und 100% (Potentiometer am rechten Anschlag) einzustellen.

Beobachten Sie, wie verschiedene Einstellungen des Potentiometers das Verhalten der LED 6 verändern.

✎ **Schriftliche Aufgabe V-1**: Entsprechend der Vorgabe ist der Taktgrad in Stufen von 1% einstellbar. Wären auch Stufen von 1 ‰ (Promille) sinnvoll (d. h. messbar)? Begründen Sie Ihre Antwort (vollständige Sätze).

1.3 Helligkeitssteuerung

Um eine Helligkeitssteuerung der LED 6 zu realisieren, muss die Periodendauer des PWM-Signals deutlich verkürzt werden. Damit vom menschlichen Auge kein Flackern mehr wahrgenommen werden kann, darf die Periodendauer höchsten 20 ms betragen.

Erstellen Sie eine neue Klasse `CPwmGenerator3`. Sie können die Klasse als Kopie von `CPwmGenerator2` erstellen, müssen aber darauf achten, dass Sie die Kommentare anpassen.

Der Konstruktor der neuen Klasse bekommt einen Parameter vom Typ `uint16_t`, mit dem die Periodendauer im Bereich von 1 µs (Wert des Arguments: 0) bis 65536 µs (Wert des Arguments: 65535) eingestellt werden kann. Ansonsten verhält sich die Klasse wie `CPwmGenerator2`.

Hinweis: Eine Implementierung ohne zusätzliches Attribut ist möglich. Wenn Ihnen die Implementierung unter Nutzung eines zusätzlichen Attributs für die Speicherung der Periodendauer leichter fällt, dürfen Sie aber ein entsprechendes Attribut einführen.

Testen Sie die neue Klasse, indem Sie eine Funktion `task3` erstellen. In der Funktion erzeugen Sie vor der `while`-Schleife ein Objekt vom Typ `CPwmGenerator3` mit einer Periodendauer von 20 ms. In der `while`-Schleife lesen Sie den Wert des ADC, filtern ihn mit Ihrem exponentiellen Glättungsfilter und verwenden das Ergebnis, um den Tastgrad zwischen 0% (Potentiometer am linken Anschlag) und 100% (Potentiometer am rechten Anschlag) einzustellen. Beobachten Sie die Helligkeitsänderungsänderung der LED.

✎ **Schriftliche Aufgabe V-2**: Kontrollieren Sie mit dem Logic-Analyzer (ein Kanal an PC6) die Einhaltung der Zeiten. Erstellen Sie einen Screen-Shot mit ca. 10% Tastgrad und einen Screen-Shot mit ca. 90% Tastgrad und fügen Sie die Screen-Shots in Ihr abzugebendes PDF ein.

Um zu testen, dass sich Ihre Methode `setDutyCycle` korrekt verhält, ändern Sie jetzt die Periodendauer auf 15 ms.

✎ **Schriftliche Aufgabe V-3**: Kontrollieren Sie mit dem Logic-Analyzer die geänderten Zeiten. Erstellen Sie einen Screen-Shot mit ca. 10% Tastgrad und einen Screen-Shot mit ca. 90% Tastgrad und fügen Sie die Screen-Shots in Ihr abzugebendes PDF ein.

Ändern Sie die Periodendauer wieder auf 20 ms.

2 Interrupt-generiertes PWM-Signal

Die Verwendung des Hardware-generierten PWM-Signals ist natürlich nur möglich, wenn das von der Logik generierte Signal als Alternate Function des GPIO-Pins ausgewählt werden kann, an den das anzusteuernde Gerät angeschlossen ist. Ist das nicht der Fall, kann man sich behelfen, indem man im Interrupt-Handler des Timers den gewünschten GPIO-Pin auf 0 bzw. 1 setzt.

Die Nutzung eines Interrupt-Handlers für einen Timer wurde in der Vorlesung bereits vorgestellt. Um diesen Ansatz in unseren neu strukturierten PWM-Code zu integrieren, soll die PWM-Generator-Klasse um eine Methode ergänzt werden, die den Interrupt behandelt.

2.1 Methode für die Interrupt-Behandlung

Erstellen Sie eine neue Klasse `CPwmGenerator4`. Sie können die Klasse als Kopie von `CPwmGenerator3` erstellen, müssen aber darauf achten, dass Sie die Kommentare anpassen.

Ergänzen Sie ein Attribut vom Typ `DigitalOut` und eine entsprechende Setter-Methode.

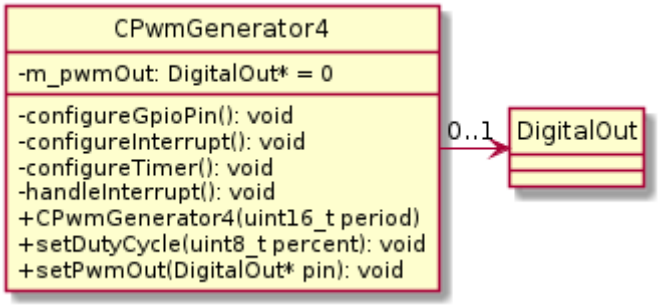


Bild 1: Klassendiagramm `CPwmGenerator4`

(Je nachdem., welchen Lösungsansatz Sie in den vorherigen Aufgaben gewählt haben, kann die Methode `configureTimer` auch einen Parameter haben und/oder es kann ein weiteres Attribut geben.)

Die Methode `configureInterrupt` wird im nächsten Abschnitt implementiert.

Implementieren Sie zunächst die Methode `handleInterrupt` entsprechend dem Code-Beispiel auf der Folie „Compare Register – PWM mit Interrupt“ aus der Vorlesung zum Timer. Wenn mit `setPwmOut` ein `DigitalOut` für die Ausgabe des PWM-Signals eingestellt wurde, wird dieser Ausgang beim Update-Interrupt auf 1 und beim Compare-Interrupt auf 0 gesetzt. Denken Sie daran, dass der Interrupt bereits aufgerufen werden kann, wenn `m_pwmOut` noch den Initialwert (0) hat und dass die Flags, die den Interrupt ausgelöst haben, in jedem Fall zurückgesetzt werden müssen, d. h. unabhängig davon, ob ein anzusteuernder Ausgang „gesetzt“ wurde oder nicht.

2.2 Aufruf der Methode für die Interrupt-Behandlung

Die neue Methode `configureInterrupt` muss den in der Vorlesung präsentierten Code für das Einschalten des Timer-Interrupts am Timer und am NVIC enthalten. Zusätzlich muss sie dafür sor-

gen, dass die Methode `handleInterrupt` von der in der Vektortabelle eingetragenen C-Funktion `TIM3_IRQHandler` aufgerufen wird.

Leider reicht die in der Vorlesung vorgestellte Lösung, den „effektiven“ Interrupt-Handler mit Hilfe eines Funktionszeigers einzustellen, nicht aus, denn `handleInterrupt` ist keine C- oder C++ Funktion sondern eine Methode (also eine Funktion eines Objekts) und zum Aufruf benötigt man nicht nur die Information, welche Methode aufgerufen werden soll, sondern auch noch die Information für welches Objekt diese Methode aufgerufen werden soll.

Die C++-Standartbibliothek stellt für diesen Anwendungsfall mit der Klasse „`function`“ eine Art erweiterten Funktionszeiger zur Verfügung, der auch auf eine Methode eines Objekts verweisen kann. Die Variable für den Zeiger (`activeTIM3_IRQHandler`) ist bereits in der Datei main.cpp des vorbereiteten Laborprojekts definiert, ebenso wie der Aufruf in der C-Funktion `TIM3_IRQHandler`. Sie müssen lediglich den „erweiterten Funktionszeiger“ `activeTIM3_IRQHandler` auf die Methode `handleInterrupt` Ihres Exemplars von `CPwmGenerator4` setzen.

Dazu inkludieren Sie in der Implementierungsdatei von `CPwmGenerator4` „main.h“ und ergänzen in `configureInterrupt` die Anweisung

```
activeTIM3_IRQHandler = bind(&CPwmGenerator4::handleInterrupt, this);
```

Die für den Aufruf Ihrer Methode zur Behandlung des Interrupts relevanten Zeilen sind also zusammengefasst:

```
// ...
#include "main.h"
// ...


void CPwmGenerator4::configureInterrupt() {
     activeTIM3_IRQHandler = bind(&CPwmGenerator4::handleInterrupt, this);
     // ...
}
```

2.3 Test

Testen Sie die neue Klasse, indem Sie eine Funktion `task4` erstellen. Diese Funktion enthält die gleichen Anweisungen wie `task3`, Sie verwenden lediglich `CPwmGenerator4` (statt `CPwmGenerator3`) und erzeugen vor der `while`-Schleife ein Objekt vom Typ `DigitalOut`, das die LED 7 ansteuert und das Sie bei Ihrem Generator als anzusteuernden Ausgang setzen.

- **Schriftliche Aufgabe V-4**: Messen Sie mit dem Logic-Analyzer möglichst genau, um welche Zeit das über den Interrupt erzeugte Signal gegenüber dem mit der Hardware erzeugten Signal verzögert ist. Fügen Sie in das abzugebende PDF einen Screen-Shot ein, auf dem Ihre Messung zu sehen ist. Geben Sie die gemessenen Werte an.

Kontrollieren Sie mit dem Logic-Analyser, ob die LED 7 bei der Null-Stellung des Potentiometers wirklich komplett aus ist. Wenn Sie in Ihrer Methode `handleInterrupt` den in der Vorlesung vorgestellten Code verwendet haben, sollte das nicht der Fall sein. Überlegen Sie, warum dieser Effekt auftritt und verbesser Sie Ihren Code entsprechend.