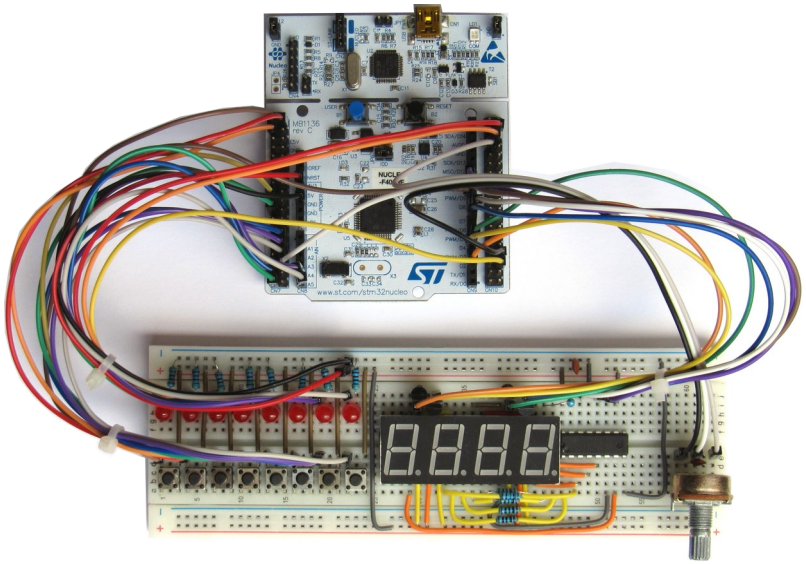
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

Mikroprozessoren-Labor



Versuch 4, Timer und Interrupts

Versuchsdurchführung

Stand: 21. Januar 2021

I. Aufgaben

1 Interrupt-gesteuertes Multiplexing

Die im Labor 2 implementierte 7-Segment-Anzeige ist darauf angewiesen, dass die Methode `activateNextDigit` zuverlässig alle 5 ms aus dem Super-Loop aufgerufen wird. Die Klasse soll in diesem Versuch so modifiziert werden, dass sie mit Hilfe eines Timers „autonom“ funktioniert.

Da für das Zeitmultiplexing lediglich ein periodische erzeugten Interrupt benötigt wird, soll der Timer 10 verwendet werden. Er hat (zusammen mit dem Timer 11) den geringsten Funktionsumfang. Durch diese Wahl stehen die Timer mit größerem Funktionsumfang weiter für anspruchsvollere Aufgaben zur Verfügung.

1.1 Vorbereitung

Kopieren Sie Ihre Klasse `CSevenSegmentDisplay` (und die von ihr benötigte Klasse `CShiftRegisterOutputExtender`) in Ihr Laborprojekt. Ändern Sie die Sichtbarkeit der Methode `activateNextDigit` auf `private`.

1.2 Timer-Initialisierung

Ergänzen Sie eine private Methode `void initTimer()`, die Sie am Ende des Konstruktors aufrufen. Diese neue Methode soll den Timer 10 so konfigurieren, dass er alle 5 ms einen Update-Interrupt auslöst. Die Schritte für das Konfigurieren des Timers 10 sind bis auf den Interrupt-Handler die gleichen wie für das Konfigurieren des Timers 3. Die Bezeichnungen der Register und Bit-Masken können in den meisten Fällen gefunden werden, indem man „3“ durch „10“ austauscht. Kontrollieren Sie im Zweifelsfall Ihre Anweisungen mit Hilfe des Reference Manuals [F401-RM] (beispielsweise befindet sich das Bit zum Einschalten des Takts in einem anderen RCC-Register als beim Timer 3).

In [F401-RM], Tabelle 38, finden Sie auch den zum Timer 10 gehörenden Vector (d. h. die Startadresse des Handlers) an Position 25. Wie Sie der Tabelle entnehmen können, gibt es offensichtlich einen gemeinsamen Interrupt-Handler für den „Timer 1 Update Interrupt“ und alle Timer 10 Interrupts. Da Sie den Timer 1 im Moment nicht benutzen, muss Sie das nicht weiter stören.

Den Namen des „Dummy-Eintrags“ in der Vector-Tabelle – und damit den Namen der Funktion, die Sie als Interrupt-Handler definieren müssen – finden Sie in der Datei `mbed-os/targets/TARGET_STM/TARGET_STM32F4/TARGET_STM32F401xE/TOOLCHAIN_GCC_ARM/startup_stm32f401xe.S`. In dieser Datei wird ab Zeile 145 der Vector-Table „gefüllt“. An Position 25 wird der Default-Handler unter dem Namen `TIM1_UP_TIM10_IRQHandler` eingetragen. Wenn Sie eine C-Funktion mit diesem Namen definieren, wird sie – anstelle des „Dummy-Handlers“ – bei einem Interrupt aufgerufen.

1.3 `ActivateNextDigit` aufrufen

Als letzten Schritt der Umstellung müssen Sie `activateNextDigit` aus dem Interrupt Handler heraus aufrufen. Das könnten Sie grundsätzlich wie in der Vorbereitung machen, indem Sie den Typ `function` (also die Kombination aus Zeiger auf Objekt und Zeiger auf Methode) aus der C++-Standardbibliothek benutzen. Allerdings sind in dem hier zu implementierenden Fall Objekt und Methode fest vorgegeben, so dass Sie auch eine einfachere Variante realisieren können.

Ergänzen Sie in der Klasse `CSevenSegmentDisplay` ein privates Klassenattribut[1] `m_instance` vom Typ `CSevenSegmentDisplay*` und weisen Sie ihm im Konstruktor den Zeiger auf das erzeugte Objekt zu. Deklarieren Sie weiterhin `TIM1_UP_TIM10_IRQHandler` als `friend`-Funktion der Klasse `CSevenSegmentDisplay` (beachten Sie den Hinweis im nächsten Absatz). Damit können Sie im Interrupt Handler (`TIM1_UP_TIM10_IRQHandler`) auf das Klassenattribut zugreifen, erhalten den Zeiger auf Ihr Objekt vom Typ `CSevenSegmentDisplay` und können die Methode `activateNextDigit` aufrufen.

Eine Funktion im C-Namensraum (`extern "C"`) kann leider nicht mit einer einzigen Anweisung als Funktion deklariert und zu einem Freund gemacht werden wie das bei C++-Funktionen möglich ist (erinnern Sie sich z. B. an die Funktion `operator<<` zum Überladen der Ausgabe auf der Konsole). Sie müssen die C-Funktion explizit vor der Klassendefinition deklarieren und sie dann in der Klassendefinition „zum Freund machen“.

Die für `friend`-Deklaration relevanten Zeilen sind zusammengefasst:

```
// ...
extern "C" void TIM1_UP_TIM10_IRQHandler();
// ...


class CSevenSegmentDisplay {
     // ...
     friend void TIM1_UP_TIM10_IRQHandler();
}
```

Zum Testen der verbesserten Implementierung der Klasse `CSevenSegmentDisplay` schreiben Sie eine Funktion `task5`, die ein Objekt vom Typ `CSevenSegmentDisplay` erzeugt, als anzuzeigende Ziffernfolge die letzten vier Stellen Ihrer Matrikelnummer einstellt und dann nichts mehr macht, d. h. der Super-Loop ist leer.

✎ **Schriftliche Aufgabe D-1**: Kontrollieren Sie das Timing, indem Sie vier Kanäle des Logic-Analyzer an die GPIO-Pins anschließen, die die Stellen der 7-Segment-Anzeige ansteuern, und einen vollständigen Ansteuer-Zyklus protokollieren und in Ihrer Ausarbeitung dokumentieren (verwenden Sie die Pin-Namen als Labels im Diagramm). Lassen Sie den Logic-Analyzer angeschlossen, Sie werden ihn gleich nochmal benötigen.

1 Sie haben in Labor 2 schon einmal ein Klassenattribut verwendet, dort war es in der Klasse `CSevenSegmentDisplay` im Ausgangsprojekt bereits vorgegeben. Diesmal müssen Sie es selbst deklarieren und definieren.

2 Energiesparen

Da der Super-Loop leer ist, d. h. die CPU „aktiv nichts tut“ liegt es nahe, Energie zu sparen, indem man die CPU stilllegt bis der nächste Interrupt zur Bearbeitung ansteht.

Dafür kennt der ARM-Prozessor den Assembler-Befehl „Wait for Interrupt“, der auch als C-Funktionsaufruf `__WFI()` zur Verfügung steht. Diesen Aufruf können Sie in die Endlosschleife einfügen. Sollten Sie über einen USB-Zwischenstecker zum Messen des Stroms verfügen, können Sie auch tatsächlich einen – wenn auch nicht sehr großen – Unterschied messen.

Mit einem kleinen Trick können Sie den Effekt auch mit dem Logik-Analysator abschätzen. Dazu schalten Sie einen Ausgang (z. B. LED0) vor dem Aufruf von `__WFI()` aus und nach dem Aufruf wieder an. Der Super-Loop würde also wie folgt aussehen:

```
DigitalOut led0(BB_LED_0);
while (true) {
      led0 = 0;
      __WFI();
      led0 = 1;
}
```

Damit müsste die LED ausgeschaltet sein, so lange der Prozessor auf einen Interrupt wartet, d. h. so lange er im Energiesparmodus ist. Leider funktioniert es so einfach nicht, denn der Interrupt wird sofort nach der „Rückkehr“ von `__WFI()` und damit noch vor dem Einschalten der LED bearbeitet.

Es funktioniert aber, wenn Sie die Bearbeitung des Interrupts zunächst verhindern.

```
DigitalOut led0(BB_LED_0);
while (true) {
      // Ausführung von Interrupt Handlern verhindern und LED ausschalten.
      __disable_irq();
      led0 = 0;

      // Auf einen Interrupt warten. Wenn er auftritt wird die CPU wieder
      // aktiviert, der Interrupt Handler wird aber noch nicht ausgeführt.
      // Der Interrupt "steht aus" ("is pending").
      __WFI();

      // LED einschalten und die Ausführung von Interrupt Handlern wieder
      // zulassen. Der Handler wird damit sofort nach __enable_irq()
      // aufgerufen.
      led0 = 1;
      __enable_irq();
}
```

✎ **Schriftliche Aufgabe D-2**: Übernehmen Sie den o. a. Code in Ihr Programm. Schließen Sie einen weiteren Kanal des Logic-Analyzers an LED0 an. Protokollieren Sie einen vollständigen Ansteuerzyklus und übernehmen Sie das Ergebnis in Ihre Ausarbeitung. Bestimmen Sie aus den High/Low-Zeiten der LED die CPU-Auslastung in Prozent.

3 Helligkeitssteuerung

3.1 Funktionsweise

Auch der „einfache“ Timer 10 verfügt über ein Compare-Register. Damit lässt sich sehr einfach eine Helligkeitssteuerung für die 7-Segment-Anzeige realisieren. Wird in das `CCR1` ein Wert geladen, der kleiner ist als der Wert im `ARR`, kann der Compare Interrupt benutzt werden, um die gerade aktive Stelle der Anzeige „vorzeitig“ (d. h. vor dem Wechsel zur nächsten Anzeigestelle) auszuschalten. Damit wird effektiv für jede Anzeigestelle eine Pulsweitenmodulation zwischen 0% und 25% realisiert (bezogen auf den Ansteuerzyklus für alle 4 Stellen).

3.2 Realisierung

Ergänzen Sie in der Klasse `CSevenSegmentDisplay` die öffentlichen Methoden `void setBrightness(int percent)` und `int getBrightness()`. Mit der setter-Methode soll die Helligkeit zwischen 5% und 100% eingestellt werden können (bezogen auf die maximal mögliche Helligkeit). Argumente kleiner 5 werden wie 5 behandelt, Argumente größer 100 wie 100.

Natürlich benötigen Sie eine neue private Methode, um bei einem Compare-Interrupt aus dem Interrupt Handler heraus alle Stellen der Anzeige ausschalten zu können.

Testen Sie die Implementierung der Methoden, indem Sie die Helligkeit vor dem Ausführen des Super-Loop in `task5` auf verschiedene Werte setzen.

✎ **Schriftliche Aufgabe D-3**: Kontrollieren Sie das Timing, indem Sie den Logic-Analyser wie in der vorherigen Aufgabe anschließen, einen Helligkeitswert von 33% einstellen und einen vollständigen Ansteuer-Zyklus protokollieren und in Ihrer Ausarbeitung dokumentieren.

3.3 Interaktive Steuerung

Erstellen Sie eine Funktion `task6` als Kopie von `task5` und entfernen Sie die für die Messung eingeführte Ansteuerung der LED 0. In dieser neuen Funktion soll jetzt für die Einstellung der Helligkeit eine Ein-Knopf-Steuerung realisiert werden.

Hat die Anzeige maximale Helligkeit und wird der Taster ganz links gedrückt, reduziert Ihr Programm die Helligkeit alle 50 ms um 1% solange der Knopf gedrückt bleibt und das Minimum noch nicht erreicht ist.

Hat die Anzeige nicht die maximale Helligkeit und wird der Knopf ganz links gedrückt, erhöht Ihr Programm die Helligkeit alle 50 ms um 1% solange der Knopf gedrückt bleibt und das Maximum noch nicht erreicht ist.

Für die Ausführung einer Aktion alle 50 ms genügt der schon im 1. Labor eingesetzte `CPolledTimer`. Sie können ihn auch in Kombination mit der Energiespar-Variante im Super-Loop abfragen, da alle 5 ms ein Interrupt ausgelöst und die Schleife einmal durchlaufen wird. Im „schlimmsten Fall“ verzögert sich das Einstellen der nächsten Helligkeitsstufe damit um 5 ms (also 10%), was nicht auffallen wird.

- **Schriftliche Aufgabe D-4 (freiwillig)**: Während die Helligkeit sinkt (insbesondere wenn sie sich dem Minimum nähert) sollten Sie manchmal beobachten können, dass eine (zufällige) Anzeigestelle kurz hell aufblinkt (maximale Helligkeit). Überlegen Sie, warum das passieren kann.
- **Schriftliche Aufgabe D-5 (freiwillig)**: Wenn Sie verstanden haben, was die Ursache für das kurze Aufblinken ist, können Sie leicht nachvollziehen, warum es im Timer-Register CCMR1 das Flag OC1PE gibt. Lesen Sie den entsprechenden Abschnitt in [RM-401RE]. Setzen Sie das Flag versuchsweise in der Timer-Initialisierung. Behebt es das Problem?