CT Praktikum: Timer und PWM

1 Einleitung

In diesem Praktikum setzen Sie die zwei 16-bit Timer TIM3 und TIM4 ein, um die Helligkeit der CT Board-LEDs zu kontrollieren. TIM4 erzeugt im Sekundentakt einen Interrupt. TIM3 produziert drei synchrone PWM Signale um die drei LED-Bänke LED7_0, LED15_8 und LED23_16 anzusteuern. Der Reload Wert in TIM3 bestimmt die gemeinsame Periode der PWM Signale. Die Werte in den zugehörigen Compare Registern legen den individuellen Tastgrad (Duty Cycle) der einzelnen PWM-Signale fest. Der Tastgrad wiederum bestimmt die Intensität, mit welcher die entsprechenden LEDs leuchten. Abbildung 1 zeigt das Prinzip:

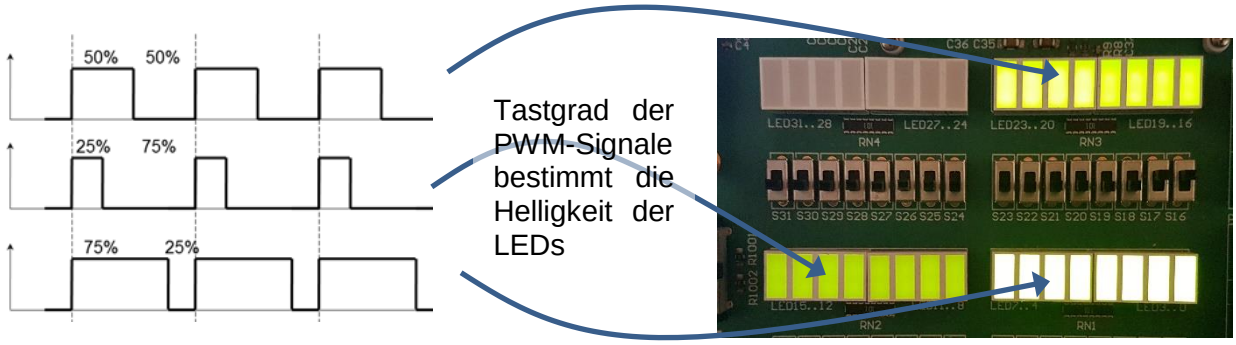

Abbildung 1: Helligkeitssteuerung von LEDs

Die LEDs 23-0 sind für die Darstellung der PWM-Signale reserviert und werden vom vorgegebenen Framework verwaltet. Bitte verwenden Sie nur die LEDs 31-24 für Ihre Zwecke.

Verwenden Sie für den Zugriff auf die LEDs und die Schiebeschalter des CT Boards die im File **`reg_ctboard.h`** definierten **`unions/structs`**.
Beispiele: **`CT_LED->BYTE.LED31_24`** und **`CT_DIPSW->BYTE.S7_0`**.

2 Lernziele

- Sie können mit einem Timer auf dem CT Board ein periodisches Interrupt erzeugen und eine zugehörige Interrupt Service Routine implementieren.
- Sie können mit einem Timer auf dem CT Board synchrone PWM Signale mit unterschiedlichen Duty Cycles erzeugen.
- Sie können die Funktion des Reload- und der Compare Register erklären.

3 Aufgaben

Verwenden Sie für die folgenden Aufgaben den zur Verfügung gestellten Projektrahmen. Dieser besteht aus einem Hauptprogramm **`main.c`**, einem Modul timer (**`timer.h`** und **`timer.c`**) sowie dem Modul pwm_sampler. Während das Hauptprogramm für die Ablaufsteuerung sorgt, stellt das Modul timer die Funktionen für die Zugriffe auf die verwendeten Timer zur Verfügung.

Das Modul pwm_sampler wird später verwendet, um die von Ihnen erzeugten PWM-Signale auf den LEDs des CT-Boards anzuzeigen.

3.1 TIM4 – Interrupt im Sekundentakt

Der Timer TIM4 soll im Folgenden so initialisiert werden, dass er jede Sekunde einen Interrupt auslöst. In der zugehörigen Interrupt Service Routine **`TIM4_IRQHandler()`** soll die LED31 im Sekundentakt blinken.

Initialisierung TIM4

Ergänzen Sie die Funktion **`tim4_init().`** Dabei müssen der Timer TIM4 konfiguriert und der zugehörige Interrupt freigeschaltet werden.

Hinweise

- Verwenden Sie für die Zugriffe auf die Kontrollregister des Timers, das in **`reg_stm32f4xx.h`** definierte **`struct reg_tim_t`** sowie das im gleichen File definierte Macro für die Basisadresse des Timers.
- Der Clockeingang **`CK_INT`** der verwendeten Timer wird bereits vor dem Start von **`main()`** auf 84 MHz gesetzt. Ebenso ist die Initialisierung einiger Timer Register am Anfang von **`tim4_init()`** bereits vorgegeben. Diese müssen nicht verändert werden.
- Wählen Sie den Prescaler Wert so, dass TIM4 in Zählschritten von 100 us (10 kHz) zählt.
- Setzen Sie den Wert des Reload Registers so, dass jede Sekunde ein Überlauf erfolgt.
- Laden Sie für das Register CR1 den im Reference Manual beschrieben Defaultwert 0x0000. Wählen Sie zusätzlich die Richtung Downcounter.
- Enablen Sie im Timer TIM4 das Auslösen eines Interrupts bei Überlauf (Bit UIE in Register DIER). Disablen Sie die andern Interruptquellen und DMA Requests indem Sie alle anderen Bits auf null setzen.
- Enablen Sie ganz am Schluss den Timer TIM4 über das Bit CEN.
- Das Enablen des TIM4 Interrupts im NVIC am Ende der Funktion ist bereits vorgegeben (Register ISER0 – SETENA Bits).

Interrupt Service Routine (ISR)

Wenn der Timer TIM4 überläuft wird das UIF Bit gesetzt und die zugehörige Interrupt Service Routine **`TIM4_IRQHandler()`** aufgerufen. Ergänzen Sie den vorgegebenen Funktionsrahmen. In der ISR müssen das UIF Bit gelöscht werden (verwenden Sie **`tim4_reset_uif()`**) und die LED31 angesteuert werden.

Verifikation

Rufen Sie im Hauptprogramm **`main()`** die von Ihnen ergänzte Intialisierungsfunktion für den Timer TIM4 auf. Danach erstellen Sie eine endlose **`while(1){}`** Schleife, welche auf den Interrupt wartet. Überprüfen Sie die korrekte Funktion Ihres Programmes auf dem CT Board.

3.2 TIM3 – Setzen der Helligkeits-Werte

In diesem Schritt ergänzen wir das bestehende Programm um die drei synchronen PWM Signale für die Ansteuerung der LEDs auf dem CT Board zu erzeugen. Die gemeinsame Periode der drei Signale soll auf 5 ms (200 Hz) eingestellt werden. Die Wahl des Periodenwertes ist ein 'Design Entscheid'. Sie basiert auf Eigenschaften des verwendeten LED Treiberbausteines und den Teilerverhältnissen im Microcontroller. Wichtig ist lediglich, dass die resultierende Frequenz über 50 Hz liegt, sonst flackern die LEDs wahrnehmbar.

Von den Schiebeschaltern sollen die Duty Cycles der einzelnen PWM-Channels als 4-bit Werte eingelesen werden:

- S3..S0 steuert den Duty Cycle von PWM-Channel 1, Ausgabe auf GPIOB Pin 4
- S11..S8 steuert den Duty Cycle von PWM-Channel 2, Ausgabe auf GPIOB Pin 5
- S19..S16 steuert den Duty Cycle von PWM-Channel 3, Ausgabe auf GPIOB Pin 0

Die eingestellten Werte 0-15 werden durch eine Multiplikation skaliert und in die entsprechenden Compare Register (CCR1 – CCR3) geladen. Bei einem Schalterwert 0x0 soll ein Duty Cycle von 0% entstehen, wogegen beim Schalterwert 0xF ein Duty Cycle von 100% entstehen soll. Der Bereich dazwischen soll in 15 gleich grosse Schritte eingeteilt werden. So entsteht beispielsweise bei der Schalterstellung 0x3 ein Duty Cycle von 3 / 15 ≙ 20 %.

Initialisierung TIM3

Hinweise

- Die Konfiguration der GPIO Pins ist bereits vorgegeben. Sie sind als PWM Ausgänge für den Timer TIM3 geschaltet.
- Ebenso ist die Initialisierung einiger Timerregister bereits vorgegeben.
- Konfigurieren Sie den Timer TIM3 als Upcounter. Alle anderen Bits im CR1 Register sollen auf 0 gesetzt werden.
- Wählen Sie entsprechende Werte für Prescaler und Reload damit bei möglichst grosser Ausnützung des Zählbereiches eine Frequenz von 200 Hz entsteht.
- Setzen Sie die Output Compare Modes der drei Channels auf Mode 1 (Bits OCxM im Register CCMR1 bzw. CCMR2)
- Enablen Sie die Outputs der drei Channels (Bits CCxE im CCER Register). Alle anderen Bits des Registers sollen auf 0 gesetzt werden.
- Enablen Sie den Timer TIM3 am Ende der Funktion.

Ergänzen Sie die Funktion **`tim3_init()`** mit der Konfiguration der Kontrollregister für den Timer TIM3.

Setzen der Duty Cycles

Lesen Sie im Hauptprogramm innerhalb der **`while`** -Endlosschleife die Werte der Schiebeschalter ein, skalieren Sie diese und setzen Sie die drei Duty Cycles mit der Funktion **`tim3_set_compare_register()`**. Diese ist als Rahmen vorgegeben und muss ergänzt werden.

Welcher Wert muss für einen Duty Cycle von 100% ins Compare Register geladen werden?

Welcher Wert muss für einen Duty Cycle von 0% ins Compare Register geladen werden?

Der Bereich dazwischen soll in 15 gleich grosse Schritte eingeteilt werden. Wie gross ist die Schrittweite und damit der Skalierungswert?

Verwenden Sie diesen Wert für die Skalierung der eingelesenen 4-bit Werte.

Verifikation

Das zusätzliche Framework-Modul pwm_sampler dient der Verifikation Ihres Programms und der Darstellung der PWM-Signale auf den CT-Board-eigenen LEDs.
Es enthält eine gleichnamige Prozedur, die die 3 PWM-Pins von GPIOB abfragt und deren Wert auf die LEDs 7-0 (Kanal 1), 15-8 (Kanal 2) und 23-16 (Kanal 3) schickt. Anhand der Helligkeiten der LED-Bänke können Sie sehen, welche Duty-Cycles Ihr Programm auf den drei Kanälen produziert.

Das Modul pwm_sampler enthält eine gleichnamige Prozedur, die die 3 PWM-Pins von GPIOB abfragt und deren Wert auf die LEDs 7-0 (Kanal 1), 15-8 (Kanal 2) und 23-16 (Kanal 3) schickt. Anhand der Helligkeiten der LED-Bänke können Sie sehen, welche Duty-Cycles Ihr Programm auf den drei Kanälen produziert.

- Fügen Sie das Modul pwm_sampler zu Ihrem main-Modul hinzu, indem Sie das entsprechende **`#include`**-Statement schreiben.
- Rufen Sie am Schluss Ihrer **`while(1){}`**-Schleife die Prozedur pwm_sampler() auf:

```
while (1) {
    …
    (hier ist Ihr Code)
    …
    pwm_sampler();    // map pwm signals to the onboard leds
}
```

Überprüfen Sie die korrekte Funktion Ihres Programmes. Lassen sich die Helligkeiten wie gewünscht einstellen?

3.3 Periodischer Wechsel der Helligkeiten

Im dritten Schritt soll die Helligkeit der LEDs im Sekundentakt wechseln. Über den Schiebeschalter S31 soll der Benutzer zwischen den Aufgaben 3.2 und 3.3 umschalten können.

Ergänzen Sie dazu die in Aufgabe 3.1 erstellte ISR um einen zyklischen 4-bit Zähler in der vordefinierten Variable **`cycle_counter_4bit`**. Bei jedem Interrupt wird der Zähler inkrementiert.

Verwenden Sie den Zählerwert im Hauptprogramm. Der Zählerwert soll direkt den Duty Cycle von Channel 1 bestimmen. Der Duty Cycle von Channel 2 soll dagegen durch (0xF minus Zählerwert) entsprechen. Dadurch zählt Channel 1 hoch, während Channel 2 runterzählt. Channel 3 soll weiterhin durch den Wert an den Schiebeschaltern S19..S16 kontrolliert werden.

3.4 Bewertung

Die lauffähigen Programme müssen präsentiert werden. Die einzelnen Studierenden müssen die Lösungen und den Quellcode verstanden haben und erklären können.

Bewertungskriterien	**Gewichtung**
Das Programm erfüllt die in Aufgabe 3.1 geforderte Funktionalität.	1/2
Das Programm erfüllt die in Aufgabe 3.2 geforderte Funktionalität.	1/4
Das Programm erfüllt die in Aufgabe 3.3 geforderte Funktionalität.	1/4