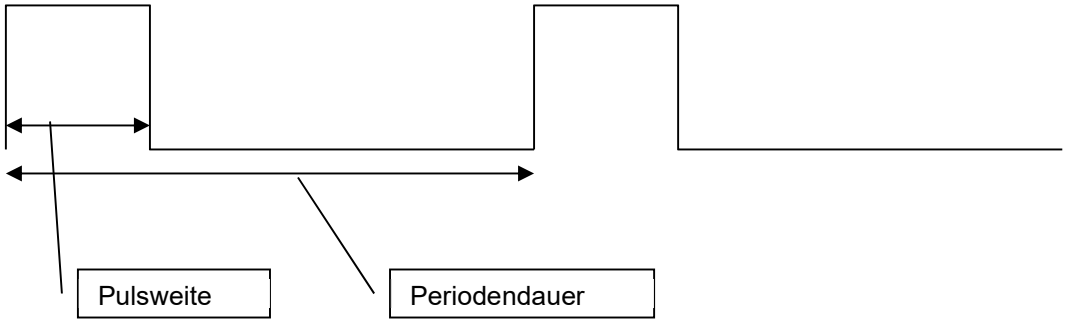
Versuch 3: Timer, PWM, DAC

Zu den wichtigsten I/O-Modulen gehören Timerbausteine. Ihre Anwendung reicht von der Erzeugung regelmäßiger Timerinterrupts über Funktionen zur Zeitmessung oder Ereigniszählung bis zur Ausgabe von zeitgesteuerten Signalen. Eine häufig verwendete zeitgesteuerte Signalform ist die Pulsweitenmodulation. In diesem Versuch werden mit Hilfe der Pulsweitenmodulation des Timers 3 ein Servomodul und eine RGB-LED angesteuert und mit Hilfe des Systick Timer Interrupts ein Schrittmotor betrieben.

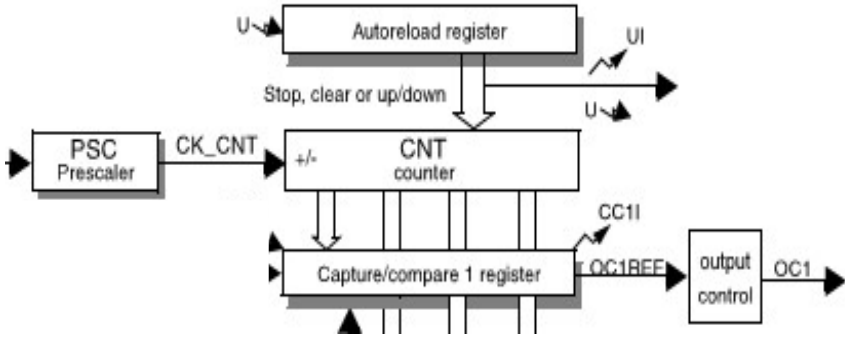
Pulsweitenmodulation (PWM):

Bei der PWM wird ein Puls regelmäßig mit einer bestimmten Periodendauer wiederholt, die Pulsbreite kann variiert werden. Durch das Verhältnis von Pulsbreite zu Periodendauer kann z.B. die Helligkeit einer LED bestimmt werden. Eine weitere Anwendung ist die Ansteuerung eines Servomotors. Die Periodendauer muss dabei 20 ms betragen, die Pulsweite kann von 1 ms bis 2 ms betragen. Die Position des Servomotors wird durch die Pulsweite gesteuert.

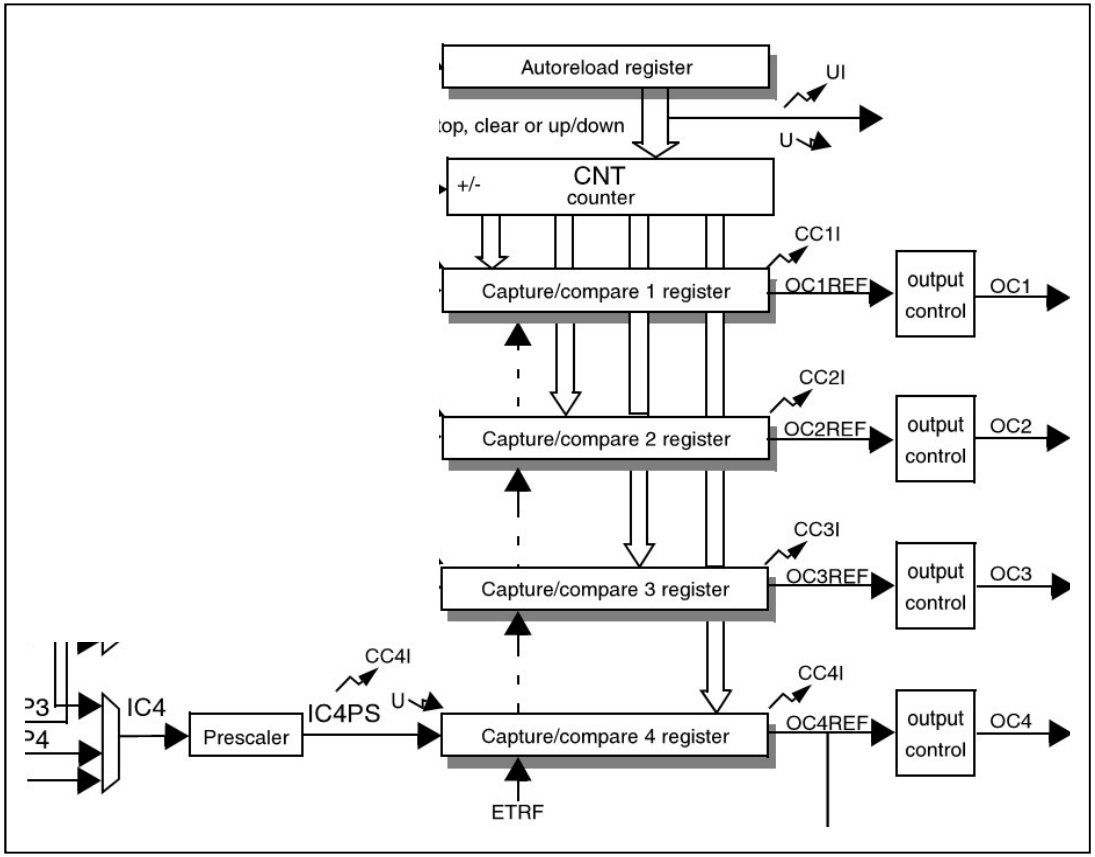


Timerbaustein:

Ein Timerbaustein besteht im Wesentlichen aus einem Zähler, der von einem Takt angesteuert wird. Der Zähler zählt von 0 bis zu einem einstellbaren Reload-Wert und beginnt dann wieder von vorn. Der Takt kann über einen Prescaler variiert werden. Der Timer 3 des STM32 ist ein universeller Timer, er arbeitet im Capture/Compare-Modus.
Capture bedeutet, dass abhängig von einem Triggersignal der aktuelle Zählerwert im Capture-Register gespeichert wird und somit die Zeit zwischen Zählerstart und Triggersignal gemessen werden kann, der Zähler wird im Input-Modus betrieben.



Im Output-Modus wird das Compareregister als Vergleichswert für die Erzeugung einer zeitgesteuerten Signalform verwendet. Der Zählerwert wird kontinuierlich mit dem Wert in einem Compare-Register verglichen. Bei Übereinstimmung von Zähler und Vergleichswert wird das Ausgangssignal OC1 gesetzt bzw. zurückgesetzt.
Im PWM-Betrieb (Modus pwm1) wird OC1 beim Zählerstart gesetzt und beim Erreichen des Vergleichswertes zurückgesetzt. Erst beim Erreichen des Reload-Wertes wird OC1 wieder gesetzt.
Der Timer 3 verfügt über vier Compareregister sowie vier Ausgangssignale, es können also vier PWM-Signale mit gleicher Periodendauer, aber unterschiedlicher Pulsweite erzeugt werden.
Die Timerausgänge liegen an den Pins PA6 und 7 sowie PC 8 und 9, d.h. diese müssen auf den Modus „alternate function“ geschaltet werden, so dass sie nicht mehr als GPIOs arbeiten, sondern durch den Timer gesteuert werden (siehe Tabelle im Dokument „Alternate Functions“ im Bereich Unterlagen).



Aufgabe 3.1: Servomotor

Ein Servomotor wird häufig im Modellbau verwendet. Ein Elektromotor fährt gesteuert durch eine interne Regelung eine bestimmte Position innerhalb eines 90°-Winkels an. Die Position wird mit einer Pulsweitenmodulation vorgegeben. Dabei muss eine Periodendauer von 20 ms eingehalten werden. Die Pulsweite liegt zwischen 1 und 2 ms, sie bestimmt die Position, z.B. 1 ms entspricht dem rechten Anschlag, 2 ms steht für den linken Anschlag. Eine Pulsweite dazwischen wird in die entsprechende Position umgesetzt.

Binden Sie die Datei `TIM3_PWM.C` aus den Projektvorlagen in Ihr Projekt ein und ergänzen Sie dort den Code.

Vervollständigen Sie die Funktion `void TIM3InitPWM (void)` so, dass an OC1 (entspricht Port A6) das PWM-Signal zum Steuern eines Servomotors generiert wird. Die Position wird wie in Versuch 1 mit „putty“ über die serielle Schnittstelle eingegeben. Dabei wird mit dem Befehl `s` der Positionswert als Prozentangabe des Vollausschlags eingegeben (z.B. `s50` = Servo fährt auf Mittelstellung) und damit die Funktion `void TIM3Servo (int pos)` aufgerufen, die Sie noch vervollständigen müssen. Die Befehlsauswertung ergänzen Sie in der Funktion `ExecCmd`.

Überzeugen Sie sich von der korrekten PWM-Ausgabe mit Hilfe des Oszilloskops.

Vorbereitung: Berechnen Sie den Prescaler-Wert, den Auto-Reload-Wert für 20 ms Periodendauer und den Wert für 1 ms Periodendauer (Compare) bei einem Prozessortakt von 16 MHz. Der Bereich von 1 ms bis 2 ms soll in 100 Schritte aufgeteilt werden. Verwenden Sie keine Komma-Zahlen!

Hinweise: Für den Timer müssen der Takt eingeschaltet und die Werte für Prescaler und Auto-Reload-Register gesetzt werden, Zählmodus ist aufwärts. Für die einzelnen Kanäle (Output Compare, OC) müssen Polarität, Modus (PWM1) und der Compare-Wert gesetzt werden. Die Einschaltfunktion für die Kanäle ist nicht unter OC, sondern unter CC (Capture Compare) zu finden. Zum Abschluss muss der gesamte Zähler eingeschaltet werden.

Aufgabe 3.2: Ansteuern einer 3-farbigen LED
Zum unterschiedlichen Dimmen einer 3-farbigen LED mit den Farben Rot, Grün und Blau verwenden wir nun die Ausgänge OC2…OC4 des Timers 3. Erweitern Sie die Funktion `void TIM3RGB(int r, int g, int b)` und die Initialisierungsfunktion `TIM3InitPWM` so, dass die Helligkeit der drei Farben der LED durch Pulsweitenmodulation gesteuert wird (Ausgänge PA7, PC8, PC9). Die Periodendauer von 20 ms ist für alle vier Kanäle gleich und muss wegen des Servos bei 20 ms bleiben, die Pulsweite für die RGB-Kanäle soll nun aber von 0 ms bis 20 ms reichen (nicht nur von 1…2 ms wie beim Servo). Durch die Einstellung von unterschiedlichen Helligkeiten (0…255) der drei Farben lassen sich beliebige Mischfarben erzeugen. Erweitern Sie Ihre Eingabe so, dass verschiedene Werte für alle 3 Farben einstellbar sind (z.B. `r150g150b0` sollte die LED gelb leuchten lassen).

Achtung: Die LEDs sind im Gegensatz zum Servo low-aktiv, d.h. sie leuchten, wenn das Ausgangssignal des Timers „low“ ausgibt. Sie müssen daher die Polarität der Ausgänge für Kanal 2…4 gegenüber Kanal 1 ändern.

Aufgabe 3.3: Schrittmotor
In diesem Versuch wird der Schrittmotor verwendet, den Sie aus dem Praktikum Digitaltechnik bereits kennen. Der Schrittmotor ist an den Ausgängen PB0…3 angeschlossen (also parallel zu den unteren vier LEDs) und benötigt im Halbschrittmodus 400 Schritte für eine ganze Umdrehung. Das Weiterschalten der Schritte erfolgt über ein vorgegebenes Schrittmuster für die Ausgänge, das sie gelb markiert im Dokument Schrittmotor.pdf im Ordner Unterlagen in Moodle finden. Der Systick-Timer soll regelmäßig einen Interrupt auslösen (typischerweise mit ca. 300…400 Hz), der zur Ansteuerung des Schrittmotors verwendet wird. Im Interrupthandler wird jeweils eines der Schrittmuster ausgegeben. Die Geschwindigkeit des Motors wird also durch die Häufigkeit der Auslösung des SysTick-Interrupts bestimmt. (Tipp: Definieren Sie das Schrittmuster in einem Array und geben Sie mit Hilfe einer Indexvariablen jeweils einen Schritt aus. Der Index wird pro Interrupt hoch- oder runtergezählt, je nach Drehrichtung des Motors).
Binden Sie bitte die Projektvorlage `Stepper.c` in Ihr Projekt ein.

Aufgabe 3.3.1: Permanentes Drehen
Steuern Sie den Schrittmotor mit Hilfe des SystickTimers an. Der Motor soll sich permanent drehen, die Drehrichtung wird durch die Eingabe von `+` bzw. `-` am putty-Terminal umgeschaltet, durch Eingabe einer `0` wird der Motor wieder gestoppt. Ergänzen Sie dazu die Funktionen `StepperInit`, `StepOut` und `SysTick_Handler` in `stepper.c` und `ExecCmd` in `main.c`. Beachten Sie bei der Ausgabe des Schrittmusters an den Port B, dass Sie auch nur die unteren vier Bits ändern, da die oberen vier Bits ja von der Potentiometerstellung vorgegeben wird.

Aufgabe 3.3.2: Positionieren
Nun soll der Schrittmotor nicht mehr permanent drehen, sondern eine bestimmte Position anfahren. Die Zielposition wird über putty vorgegeben (z.B. `m250`) und kann auch mehrfache Umdrehungen bedeuten (z.B. `m800` = zwei Umdrehungen).

Hinweis: Die Variablen `ZielPos` (global) und `AktPos` (= aktuelle Position, lokal) sind schon vordefiniert. Die Variable `ZielPos` wird über den Befehl in putty vorgegeben, `AktPos` wird vom Interrupthandler verwaltet. Im Interrupthandler wird abhänig davon, welche der beiden Positionen größer ist, die Drehrichtung ermittelt und ein Schritt in die jeweilige Richtung ausgegeben. Bei Übereinstimmung bleibt der Motor stehen.

Alle Funktionen für den Schrittmotor sollen im Modul `Stepper.c` implementiert werden.