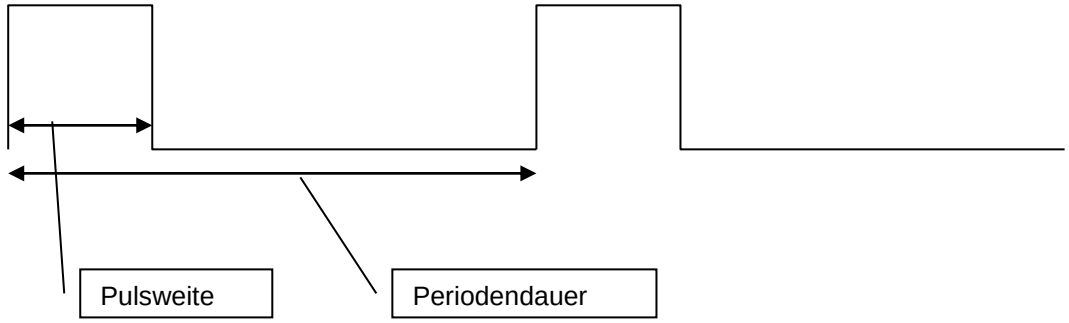
Versuch3: Timer, PWM, Schrittmotor

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 wird mit Hilfe der Pulsweitenmodulation des Timers2 ein Servomodul sowie eine RGB-LED angesteuert und mit Hilfe des Systick Timer Interrupts ein Schrittmotor betrieben.

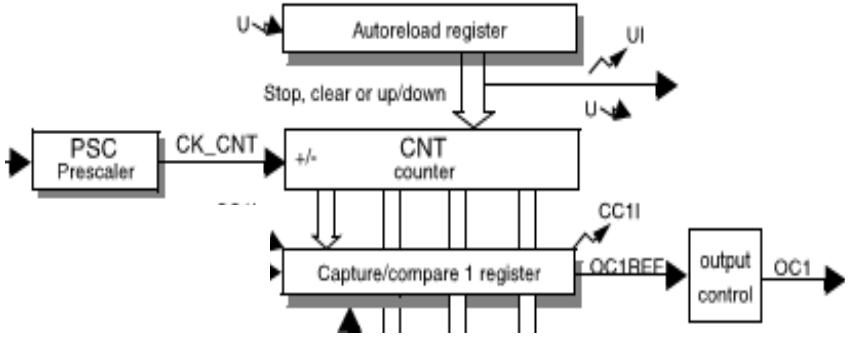
Pulsweitenmodulation:

Bei der PWM wird ein Puls regelmäßig mit einer bestimmten Periodendauer wiederholt, die Pulsbreite kann variiert werden. Durch das Verhältnis von Pulsbreite zur 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 Servomotor 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 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 (pwm2) wird OC1 beim Zählerstart gesetzt und beim Erreichen des Vergleichswertes zurückgesetzt. Erst beim Erreichen des Reload-Wertes wird OC1 wieder gesetzt.



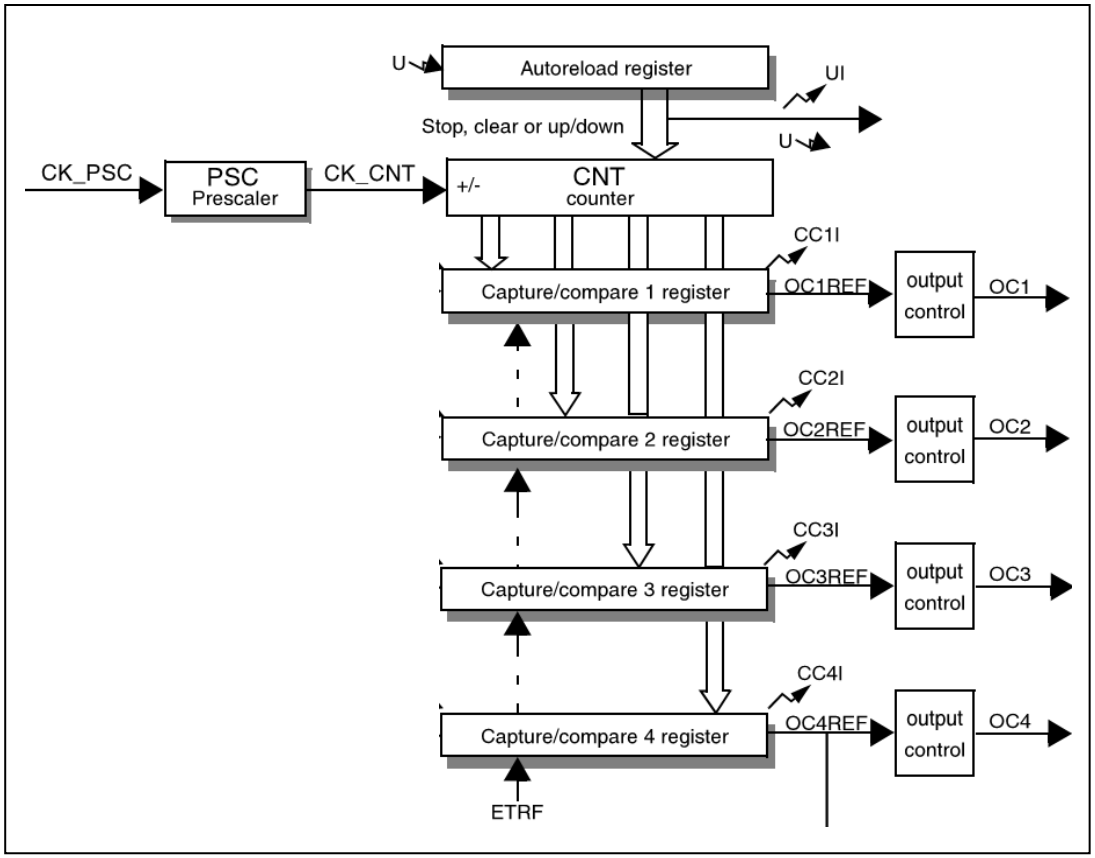
Der Timer2 des STM32 ist ein universeller Timer, er arbeitet nach dem Capture/Compare-Modus.

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. In diesem Versuch werden wir nur den Output-Modus verwenden.

Der Timer2 verfügt über vier Compareregister sowie vier Ausgangssignale, es können also vier PWM- Signale mit gleicher Periodendauer, aber unterschiedlicher Pulsweite erzeugt werden.

In der folgenden Registerübersicht sind die für die PWM notwendigen Register markiert. Die Timerausgänge liegen auf den Pin 0-3 des PortA, d.h. PortA0 –PortA3 müssen auf den Modus „alternate Function“ geschaltet werden.



Aufgabe 3_1 : Servomotor

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

Programmieren Sie nun den Timer2 so, dass an OC1 (entspricht PortA0) das PWM Signal zum Steuern eines Servomotors generiert wird. Die Position wird wie in Versuch 3 mit „putty“ über die serielle Schnittstelle eingegeben. Dabei wird nach einer Kennung „s“ der Positionswert als Prozentangabe des Vollauschlags eingegeben, ein Punkt beendet die Eingabe (z.B. „s50.“ = Servo fährt auf Mittelstellung).
Nachdem der Timer das PWM Signal autark ohne den Prozessor erzeugt, kann weiterhin die Analogausgabe aus Versuch 2 laufen, die USART –Schnittstelle wird per Interrupt eingelesen. Orientieren Sie sich am Beispielprojekt V3.1_PWM_Stepper.
Überzeugen Sie sich von der korrekten PWM –Ausgabe mit Hilfe des Oszilloskops.

Vorbereitung: Berechnen Sie den Prescaler-Wert, den Preload-Wert für 20 ms Periodendauer und den Wert für 1ms bei einem Prozessortakt von 24 MHz. Der Bereich von 1ms bis 2ms soll in 100 Schritte aufgeteilt werden. Verwenden Sie keine Komma-Zahlen!

Hinweis: Alle Zugriffe auf den Timer2 sollen im Modul *TIM2_PWM* erfolgen, so dass im main-Programm keine Hardwarezugriffe erscheinen.

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 Timer2. Erweitern Sie das Programm so, dass die Helligkeit der 3 Farben der LED durch Pulsweitenmodulation gesteuert wird (Ausgang PortA1-PortA3). Bleiben Sie bei einer Periodendauer von 20 ms, die Pulsweite soll nun aber von 0 ms bis 20 ms reichen. Durch die Einstellung von unterschiedlichen Helligkeiten (0-255) der 3 Farben lassen sich beliebige Farben erzeugen

Erweitern Sie Ihre Eingabe so, dass verschiedene Werte für alle 3 Farben einstellbar sind (z.B. „r200g150b0.“) sollte die LED gelb leuchten lassen.

Achtung: Die LEDs sind im Gegensatz zum Servo low active, d.h. sie leuchten, wenn das Ausgangssignal des Timers „low“ ausgibt. Sie müssen daher die Polarität der Ausgänge für Kanal 1 – 3 ändern. (Siehe Register CCER).

Aufgabe 3_3 : Schrittmotor
In diesem Versuch wird der Schrittmotor verwendet, den Sie aus dem Praktikum Digitaltechnik bereits kennen. Schließen Sie die 4 Motorleitungen an PortB8..11 an (siehe Versuch1 Lauflicht). Der Systick-Timer löst regelmäßig einen Interrupt aus, der zur Ansteuerung des Schrittmotors verwendet.
Im Interrupthandler wird jeweils ein Schritt ausgegeben, die Geschwindigkeit des Motors wird also durch die Interruptrate bestimmt. (Tipp: Definieren Sie das Schrittmuster in einem Array und geben Sie mit Hilfe einer Indexvariablen einen Schritt aus. Der Index wird pro Interrupt hoch- oder runtergezählt, je nach Drehrichtung des Motors)

a) 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 werden. Verwenden Sie dazu Ihr Programm von Versuch3 und erweitern Sie das Programm entsprechend .

b) Nun soll der Schrittmotor nicht mehr permanent drehen, sondern eine bestimmte Position anfahren. Die Zielposition zwischen 0 und 400 wird über putty vorgegeben (z.B. „m250.“).
Hinweis: Deklarieren Sie die Variablen ziel_pos und akt_pos (= aktuelle Position). Die Variable ziel_pos wird über putty vorgegeben, akt_pos wird vom Interrupthandler verwaltet. Im Interrupthandler wird aus der Differenz zwischen ziel_pos und akt_pos die Drehrichtung ermittelt. Bei Übereinstimmung bleibt der Motor stehen!

Hinweis: Alle Funktionen für den Schrittmotor sollen im Modul Stepper implementiert werden.

Die Register des Timer-Bausteins (TIM2) im Überblick

Table 73. TIMx register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1	Reserved																						CKD [1:0]		ARPE	CMS [1:0]		DIR	OPM	URS	UDIS	CEN
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x04	TIMx_CR2	Reserved																								TI1S	MMS[2:0]			CCDS	Reserved		
	Reset value																									0	0	0	0	0			
0x08	TIMx_SMCR	Reserved																ETP	ECE	ETP S [1:0]		ETF[3:0]				MSM	TS[2:0]			Reserved	SMS[2:0]		
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0
0x0C	TIMx_DIER	Reserved																	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Reserved	TIE	Reserved	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	Reset value																		0	0	0	0	0	0	0		0		0	0	0	0	0
0x10	TIMx_SR	Reserved																			CC4OF	CC3OF	CC2OF	CC1OF	Reserved		TIF	Reserved	CC4IF	CC3IF	CC2IF	CC1IF	UIF
	Reset value																				0	0	0	0			0		0	0	0	0	0
0x14	TIMx_EGR	Reserved																								TG	Reserved		CC4G	CC3G	CC2G	CC1G	UG
	Reset value																									0			0	0	0	0	0

Table 73. TIMx register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x18	**TIMx_CCMR1** *Output Compare mode*	Reserved																OC2CE	OC2M [2:0]			OC2PE	OC2FE	CC2S [1:0]		OC1CE	OC1M [2:0]			OC1PE	OC1FE	CC1S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TIMx_CCMR1 *Input Capture mode*	Reserved																IC2F[3:0]				IC2PSC [1:0]		CC2S [1:0]		IC1F[3:0]				IC1PSC [1:0]		CC1S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	**TIMx_CCMR2** *Output Compare mode*	Reserved																O24CE	OC4M [2:0]			OC4PE	OC4FE	CC4S [1:0]		OC3CE	OC3M [2:0]			OC3PE	OC3FE	CC3S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	TIMx_CCMR2 *Input Capture mode*	Reserved																IC4F[3:0]				IC4PSC [1:0]		CC4S [1:0]		IC3F[3:0]				IC3PSC [1:0]		CC3S [1:0]	
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x20	**TIMx_CCER**	Reserved																		CC4P	CC4E	Reserved		CC3P	CC3E	Reserved		CC2P	CC2E	Reserved		CC1P	CC1E
	Reset value																			0	0			0	0			0	0			0	0
0x24	**TIMx_CNT**	Reserved																CNT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	**TIMx_PSC**	Reserved																PSC[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	**TIMx_ARR**	Reserved																ARR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x34	**TIMx_CCR1**	Reserved																CCR1[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x38	**TIMx_CCR2**	Reserved																CCR2[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x3C	**TIMx_CCR3**	Reserved																CCR3[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x40	**TIMx_CCR4**	Reserved																CCR4[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x44	Reserved																																