Versuch 5 – Timer/Counter

Dieser Versuch enthält drei Teile:

1. Inbetriebnahme eines Schrittmotors und Erzeugung eines Tons
2. Verwendung unabhängiger Timer/Counter für den Motor und den Lautsprecher
3. Inbetriebnahme eines Servomotors

Erst im den Teilen 2 und 3 stellen Sie selber Timer/Counter ein. In Teil 1 ist das nicht nötig. Die Reihenfolge sollte trotzdem eingehalten werden, da die Lösung aus Teil 1 weiterverwendet werden kann.

1 Informationen zur Hardware

1.1 Anschluss Audioplatine

Für die Tonausgabe verwenden Sie am besten die Audioplatine (rot). Eine Beschreibung finden Sie separat zum Herunterladen. Die Platine wird mit dem Piggyback (weiß) über drei Leitungen verbunden: GND, +5V und dem Audiosignal (3.3V Rechteckschwingung). Sie können für die Ausgabe einen Lautsprecher mit 8Ω Impedanz anschließen oder einen Kopfhörer.

Besonders bei dem Anschluss eines Kopfhörers: Lautstärke beachten!

Beim Anschluss sollte die Masseverbindung zuerst hergestellt werden. Sie können später auch im Betrieb jederzeit die +5V oder den Signaleingang trennen, wenn Sie keinen Dauerton hören wollen, aber auch das komplette Abstecken vermeiden wollen.

1.2 Schrittmotor

Der Schrittmotor wird über einen Verstärker (grüne Platine mit 4 LED) an das Piggyback angeschlossen. Der Verstärker ist nötig, da ein GPIO nicht den für den Motor benötigten Strom liefern kann. (Das Verstärker-IC ULN2003A enthält nur eine Pegelanpassung und einen Leistungstransistor pro Kanal. Es hat 7 Kanäle, von denen 4 benötigt werden.)
Der Schrittmotor ist für +5V ausgelegt. Die Verbindung mit dem Piggyback erfolgt über 6 Leitungen: GND, +5V, 4 Spulen (IN1-IN4 am Verstärker mit 4 GPIO am µC verbunden).
Pro Kanal gibt es eine Kontroll-LED, die auch ohne angesteckten Schrittmotor funktioniert. Für erste Tests ist es sinnvoll, den Motor vom Verstärker abzustecken (weißer Stecker), um zunächst nur an den LED zu beobachten, ob denn die eigene Ansteuerung sinnvoll ist. Damit kann man sinnlose Spulenströme vermeiden.
Für kurzfristiges „Stilllegen“ des Motors während der Vorbereitung ist es am besten, immer nur die +5V-Verbindung zu trennen, der Rest kann angeschlossen bleiben.

1.3 Servomotoren (Servos)

Die Servomotoren aus dem Kit haben einheitlich einen dreipoligen Stecker mit folgenden Zuordnung Kabelfarbe – Funktion: GND=braun, +5V=rot, PWM=orange.
Für den Anschluss ist der Servoblock auf dem Piggyback schon vorbereitet, hier werden keine weiteren Kabel benötigt. Über den Jumper (Steckbrücke) neben dem Block kann man die +5V direkt vom Netzteil holen. Er sollte daher gesetzt sein (d.h. aufgesteckt). Die Anschlussbelegung des Servoblocks können Sie der Beschreibung des Piggybacks entnehmen. Zur Kurzorientierung: Die Masse (braunes Kabel) befindet sich Richtung DIL-Schalter.
Werden die Servos nicht benötigt, kann man den Jumper abziehen – sie können dann angesteckt bleiben, ohne mit Spannung versorgt zu werden.

Versuch 5 – Timer/Counter

2 Teil 1: Tonerzeugung und Schrittmotor

Wechseln Sie in einen neuen, leeren Workspace. Importieren Sie dann aus dem Archiv *v5_p1_vorbereitung.zip* alle drei Projekte *lpc_chip_11uxx_lib*, *mch_aux_lib* und *v5_p1*.
Die Frequenz des Systick-Timers ist auf 1000 Hz eingestellt und **soll nicht geändert** werden.

2.1 Tonerzeugung

Die schnellste Methode für einen ersten Test besteht darin, in der Systick-ISR einfach bei jedem Aufruf einen GPIO umzuschalten. Welches Tastverhältnis[1] hat das entstehende Signal und welche Frequenz hat es hier ($f_{SYSTICK}$=1000 Hz)?

Das Signal soll am Pin *PIO0_21* erzeugt werden. Dieser Pin ist schon als GPIO mit Richtung Ausgang eingestellt (siehe Feld *pins_v5*). Sorgen Sie in der Systick-ISR *SysTick_Handler* für das Umschalten bei jedem Aufruf (Kommentar *V5_2.1* suchen). Hören Sie sich dann den Ton an.

2.2 Schrittmotor

Schrittmotoren eignen sich besonders für Anwendungen, in denen es nicht auf die Geschwindigkeit oder Leistung sondern auf eine genaue Änderung der Achsenstellung ankommt. Die Achse wird dabei in Einzelschritten (daher der Name) gedreht. Abbildung 1 zeigt das Prinzip eines Schrittmotors, hier wie im Praktikum die unipolare Ausführung.

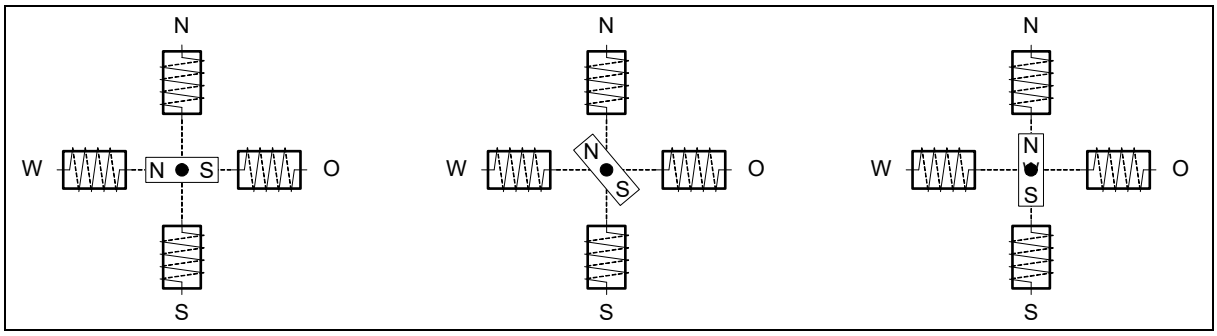

Abbildung 1: Prinzip eines (unipolaren) Schrittmotors

Es gibt vier Spulen (hier N, O, S, W). Der Anker richtet sich nach dem Magnetfeld aus, das von den Spulen erzeugt wird. Links ist nur die Spule W eingeschaltet, in der Mitte die Spulen N und W und rechts nur die Spule N. Die Bilder zeigen die resultierende Ankerstellung.
Wie viele dieser Schritte werden hier für eine vollständige Umdrehung benötigt?

[1] Zeitdauer GPIO=1 zu Zeitdauer GPIO=0 des Signals

Versuch 5 – Timer/Counter

Schreiben Sie tabellarisch die benötigen „Spulenmuster“ für alle Schritte auf.

Schritt	**N**	**O**	**S**	**W**
GPIO	*PIO1_25*	*PIO1_26*	*PIO1_27*	*PIO1_28*
Motorplatine	*IN1*	*IN2*	*IN3*	*IN4*
1	1	0	0	0
2				
…				

Der Schrittmotor soll über die GPIOs *PIO1_25* - *PIO1_28* angesteuert werden. Diese Anschlüsse sind schon als GPIOs mit Richtung Ausgang konfiguriert (siehe Feld *pins_v5*). Schreiben Sie die Funktion *v5_stepper* fertig (Kommentar *V5_2.2* suchen). Sie soll bei jedem Aufruf das Muster für den nächsten Schritt an den GPIOs ausgeben. Sie muss sich also den jeweils aktuellen Schritt merken. Verwenden Sie keine globalen Variablen.

Hinweis:
Das Prinzip ist exakt dasselbe wie bei der Ansteuerung der Multiplex-Anzeige – sowohl das Erzeugen der Muster als auch das Weiterschalten bei jedem Aufruf. Nutzen Sie die schon gefundene Lösung mit entsprechender Anpassung.

Für einen ersten Test sollten Sie den Schrittmotor noch abgesteckt lassen. Sie sehen ja an den LED, welche Spule eingeschaltet wird. So können Sie prüfen, ob die Muster so erscheinen, wie Sie das wollen.
Für den ersten Test ist es sinnvoll, die Funktion *v5_stepper* mir nur 1 Hz aufzurufen. Dann können Sie in Ruhe sehen, ob die Reihenfolge stimmt. Das ist jetzt schon so eingestellt (Kommentar *V5_P2.2a*). Wenn beides (Muster, Reihenfolge) wie gewünscht an den LED erscheint, stecken Sie den Motor an.

Sie werden keine Bewegung der Achse wahrnehmen können. Sie können aber (vermutlich) hören, dass etwas im Motor geschieht und man kann das auch fühlen, wenn man den Motor mit zwei Fingern hochhebt.

Die Ursache für das **scheinbare** Nichtfunktionieren ist die erhebliche Untersetzung, die im Motor realisiert ist. Dieser Motor benötigt 4096 Schritte für eine Umdrehung der Achse!
Jeder hier verwendete Schritt wird als Halbschritt bezeichnet, da auch die Stellungen mit nur einer Spule unter Strom benutzt werden. Nutzt man nur die Stellungen mit je zwei Spulen unter Strom für einen Schritt, dann spricht man von Vollschritten. Regelt man zusätzlich noch die Spulenleistungen unterschiedlich, dann spricht man von Mikroschritten.

Rufen Sie also nun die Funktion *v5_stepper* mit 1000 Hz auf (Kommentar *V5_2.2b*). Dann wird auch die Drehung gut sichtbar.

Sie können auch testweise die Drehrichtung umkehren, dazu müssen ja die Schritte nur in der umgekehrten Reihenfolge ausgeführt werden.

Versuch 5 – Timer/Counter

3 Tonerzeugung/Schrittmotor per Hardware und eigenem Timer

Bisher sind die Frequenzen für den Schrittmotor und den Ton an den SysTick-Timer gekoppelt, der aber auch für die Multiplex-Anzeige benötigt wird. Natürlich möchte man die Tonhöhe und die Geschwindigkeit des Motors unabhängig von anderen Aufgaben einstellen können. Dazu wird jetzt je ein eigener Timer benutzt.
Wechseln Sie in einen neuen, leeren Workspace. Importieren Sie dann aus dem Archiv *v5_p2_vorbereitung.zip* alle drei Projekte *lpc_chip_11uxx_lib*, *mch_aux_lib* und *v5_p2*.

3.1 Timer für den Schrittmotor

Übertragen Sie zunächst aus Ihrer Lösung zum Teil 1 des Versuchs die Funktion *v5_stepper*.
Für den Motor soll der 32 Bit-Timer *CT32B0* (*UM*, *Kap. 16*) eingesetzt werden. Dieser Timer kann die ISR *TIMER32_0_IRQHandler* auslösen. Die ISR ist bereits im Programm enthalten.

Für das Zurücksetzen des Timers soll das Match-Register (MR) 0 benutzt werden. Zunächst wird der Timer einmalig für seine Aufgabe vorbereitet. Dazu dient die Funktion *v5_ct32b0_init*. Schreiben Sie jetzt mit Hilfe der Bibliotheksfunktionen (siehe *timer_11xx.h*) diese Funktion fertig. Sie finden schon Kommentare für die nötigen Aktionen vor.

Hinweise:

- Das Handle für den Timer CT32B0 heißt LPC_TIMER32_0
- Die interne Taktquelle, die Sie hier benötigen, heißt *PCLK* (siehe auch *UM, Fig. 59* bzw. *UM, Table 333*, Feld *CTM*). Wenn Sie den internen Takt wählen, dann hat der Parameter *capnum* der entsprechenden Funktion keine Bedeutung – setzen Sie ihn auf 0.
- Die Funktion zum Setzen des Vorteilers erwartet als Parameter den gewünschten Wert minus 1. Möchten Sie den Vorteiler auf den Teilerfaktor 1 einstellen, dann hat der Parameter also den Wert 0.
- Die Funktion *Chip_TIMER_ClearMatch* löscht das Flag, das das Ereignis „Timer-Wert = Match-Wert) anzeigt.

Nun kann der Timer mit der Funktion *v5_ct32b0_set_freq* auf die gewünschte Frequenz gestellt und gestartet werden. Schreiben Sie auch diese Funktion fertig.

In dieser Funktion sollen Sie nach dem Ändern des Werts im Match-Register den Timer auf 0 zurücksetzen (siehe Kommentar in der Funktion). Warum ist das wichtig?
Überlegen Sie sich dazu, wann das Ereignis „Timer-Wert=neuer Match-Wert“ eintreten kann, wenn der Timer schon läuft und dann der neue Wert eingestellt wird.

Der Motor wird wieder wie in V5_P1 laufen, aber jetzt können Sie die Geschwindigkeit unabhängig vom SysTick-Timer einstellen. Probieren Sie das auch aus.

3.2 Timer für die Tonerzeugung

Es wäre offensichtlich ein Leichtes, mit derselben Methode die unabhängige Tonerzeugung zu erreichen. Sie würden einfach einen anderen Timer genauso initialisieren und dann in dessen ISR eben den GPIO für den Ton umschalten.
Hier soll aber noch einen Schritt weiter gegangen werden: Die Tonerzeugung soll ganz ohne ISR auskommen, so dass der Ton dann **ohne jeden Verbrauch** an Rechenzeit durch eine ISR von der Hardware erzeugt wird.
Das ist möglich, weil man alle Timer-Module so einstellen kann, dass beim Erreichen eines Match-Wertes im *MR n* der Anschluss *CTxxBy_MATn*[2] umgeschaltet (engl. toggle) wird.

Hier sollen der Timer *CT16B1* und das *MR 1* eingesetzt werden.

Definieren Sie nun das Symbol *V5_P2_AUDIO* in *v5_p2.h*, damit Sie gleich die vorgesehenen Stellen für den neuen C-Code finden.
Ergänzen Sie dann zunächst die Initialisierung für den Pin *PIO0_22*. Sehen Sie im *UM* nach, welche Funktionsnummer Sie hier benötigen.

Schreiben Sie jetzt die Funktion *v5_ct16b1_init* fertig (Kommentare!). Der Vorteiler wird hier zur Vereinfachung der Programmierung einmalig fest auf 16 eingestellt. Damit kann dann später der Frequenzbereich [50 Hz, 5000 Hz] ohne weitere Änderung des Vorteilers abgedeckt werden – die Audioplatine ist auf diesen Bereich eingestellt.

Schreiben Sie dann auch Funktion *v5_ct16b1_set_freq* fertig (Kommentare!)

Hinweise:

- Sie können 90% Code vom Timer *CT32B0* übernehmen, aber **das Handle** ist natürlich an anderes (beliebter Fehler bei copy&paste) und auch **das benutzte MR** ist ein anderes. Sehen Sie in *chip.h* nach, wie das Handle für den Timer *CT16B1* heißt.
- Bei der Funktion *Chip_TIMER_ExtMatchControlSet* ist der zweite Parameter (*initial_state*) unwesentlich, da sich der Zustand des Pins ja ohnehin später dauernd ändert. Setzen Sie am besten 0 ein.

Hören Sie sich den Ton am Pin *PIO0_22* an. Sie können zum Vergleich der Frequenzen auch die Lösung aus Teil 1 (SysTick) mit einbauen. Die Pins *PIO0_21* und *PIO0_22* liegen direkt nebeneinander, so dass Sie per Umstecken leicht akustisch vergleichen können.
Nehmen Sie dann aber die SysTick-Variante wieder aus dem Code, da diese Lösung Rechenzeit verbraucht.

Test/Spielerei:

Sie könnten nun in der noch leeren Endlos-Schleife eine Sirene oder eine Tonfolge programmieren, indem Sie die Frequenz in gewissen Zeitabständen neu einstellen. Zur Vereinfachung empfehle ich hier für die Wartezeit zwischen zwei Frequenzwechseln die Methode „zählen“ wie in V1. Das ist nicht elegant und verbraucht dann die ganze restliche Rechenzeit, ist aber am schnellsten programmiert. Denken Sie daran, dass Sie nach einem Frequenzwechsel so lange warten, bis ein Mensch den neuen Ton auch bewusst wahrnehmen kann – zu schnelle Wechsel führen sonst dazu, dass man gar nichts hört.

[2] xx: 16 oder 32, y: 0 oder 1, n: 0,1,2 oder 3

Versuch 5 – Timer/Counter

4 PWM-Erzeugung (Servos)

Wechseln Sie in einen neuen, leeren Workspace. Importieren Sie dann aus dem Archiv *v5_p3_vorbereitung.zip* alle drei Projekte *lpc_chip_11uxx_lib*, *mch_aux_lib* und *v5_p3*.

Eine der häufigsten Aufgaben eines Timers ist die Erzeugung von PWM-Signalen. Die Servos benötigen üblicherweise ein Signal mit einer Frequenz von 50 *Hz*.
Der Winkelstellung [0°, 180°] entspricht dabei eine Pulsweite im Bereich [1 *ms*, 2 *ms*]. Der Puls ist dabei durch ein H (High) gegeben (Abbildung 2, Mittelstellung des Servo bei 1.5 *ms*).

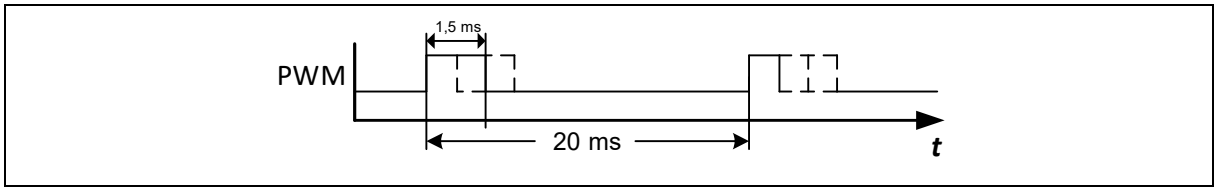

Abbildung 2: PWM-Signal für Servo

Beide Werte sind bei den billigen Servos (unter 10€) nicht kalibriert. Die Zuordnung Winkel zu Pulsbreite kann schon einmal um 30% abweichen. Das muss man dann ausprobieren.

Für die Servos sind 5 Anschlüsse vorbereitet. Die zugehörigen Pins (*PIO0_13* - *PIO0_14*, *PIO1_13* - *PIO1_15*) haben Funktionen, die zu den Timern *CT16B0* und *CT32B1* gehören.

Ergänzen Sie zunächst die nötige Initialisierung der Pins (Funktionsnummern beachten, *UM*).

Die Systemtaktfrequenz beträgt 48 *MHz*. Die Initialisierung für die beiden Timer kann vollständig gleich gehalten werden, wenn man zweierlei beachtet: Die Frequenz wird mit dem MR 3 eingestellt und der Vorteiler für beide auf einen Wert größer 14. Hier soll der Wert 48 für den Vorteiler gewählt werden.
Warum muss der Vorteilerwert bei diesem Vorgehen größer als 14 sein und warum erscheint der Wert 48 als besonders günstig?

Hinweis:
Berechnen Sie die nötigen Endwerte für die 20 *ms* (50 *Hz*) Periodendauer einmal für den Vorteilerfaktor 14 und einmal für den Vorteilerfaktor 48.

Schreiben Sie jetzt die Funktion *v5_timer_pwm_init* fertig. Der Parameter *tmr* ist dabei der Handle, mit dem die Funktion später aufgerufen wird.

Hinweis:
Die Funktion *Chip_TIMER_Set_PWMmode* legt für ein MR fest, ob es zur PWM-Erzeugung verwendet wird oder nicht. Diese Funktion finden Sie in *mch_timer.c* der Bibliothek *mch_aux_lib*. Den benötigten Typ für den Parameter finden Sie auch in *mch_aux.h*.

Schreiben Sie dann die Funktion *v5_timer_pwm_setwidth* fertig. Diese Funktion bekommt ebenfalls den Handle, dazu aber noch das MR *matchnum* und die Pulsbreite *pw*. Die Pulsbreite kann zwischen 0 und dem Endwert in MR 3 liegen. Sie bezeichnet die Zeit (eigentlich: Anzahl der Zähltakte), in der das Signal H sein soll.

Versuch 5 – Timer/Counter

Jetzt können Sie die Servos mit der Funktion *v5_set_servo* an verschiedenen Anschlüssen und mit verschiedenen Pulsbreiten ausprobieren.
Im Versuch wird in der Endlosschleife als Beispiel der Servo 1 bewegt.

Lernziele

- Umgang mit Timern
- PWM-Modulation

Material zur Vorbereitung

- User Manual UM10462, Revision 5.3 für den µC
- Projektarchive v5_px_vorbereitung.zip für MCUXpresso
- Dokumentation zum Piggyback (nur für Anschlussbelegung)
- Dokumentation zur Audioplatine (nur für Anschlussbelegung)

Zusatzinformationen bei Interesse

- Datenblätter Schrittmotor, Servo, Verstärker-IC ULN2003A
- Schaltplan Audioplatine

Material zur Durchführung

- Basisplatine, Piggyback weiß, Flachbandkabel, LED-Display, Netzteil
- Audioplatine, ggf. mit Lautsprecher
- Schrittmotor mit Verstärkerplatine
- 1-3 Servomotoren
- 6 einzelne Verbindungskabel (3-5 für Audio, 6 für Schrittmotor)
- Entwicklungswerkzeug (hier MCUXpresso Version 10.0.0)