Versuch 1: GPIO und ADC

Aufgabe 1.1: Lauflicht

An Port C0…7 sind 8 Schalter, an Port B0…7 sind 8 LEDs angeschlossen. In der ersten Aufgabe wird ein Lauflicht programmiert, die Geschwindigkeit wird durch die Funktion `wait` realisiert. Analysieren Sie das vorgegebene Programm `V1.1_Lauflicht` und lassen Sie dieses Programm im Debugger ablaufen.

Ablauf:

- In der Reset-and-Control-Logik (RCC) wird jeder I/O-Baustein separat eingeschaltet und damit mit Takt versorgt
- Initialisierung des Port B0…7 als Ausgang (General Purpose Push Pull, 50 MHz)
- Initialisierung des Port C0…7 als Eingang (Floating Input)
- In einer Endlosschleife wird das Lauflicht ausgegeben

Ergänzen Sie nun das Programm so, dass das Lauflicht abhängig von der Schalterstellung langsamer oder schneller läuft. Verwenden Sie dazu beim Aufruf der Funktion `wait` einen Parameter, der die Länge der Wartezeit bestimmt. Initialisieren Sie den Port C0…7 als Eingang (Floating Input), die nicht verwendeten Schalter müssen ausmaskiert werden. Realisieren Sie 4 verschiedene Geschwindigkeiten mit zwei Schaltern.

Aufgabe 1.2: Lauflicht mit SysTickTimer-Interrupt

Verwenden Sie nun den SysTickTimer, um die Geschwindigkeit des Lauflichts zu steuern. Vom SysTickTimer wird regelmäßig nach Ablauf des Timers ein Interrupt ausgelöst. Die Ausgabe an den Port B erfolgt nun in der Interruptroutine `void SysTick_Handler` und nicht mehr in der while-Schleife. Der Prozessortakt beträgt **16 MHz** (vordefiniert durch die Variable `SystemCoreClock`).

Hinweise:

- Definieren Sie die Funktion `void SysTick_Handler(void)`, in der Sie das Weiterschalten des Lauflichts realisieren. Die Funktion muss nur definiert, aber nie explizit aufgerufen werden; der Aufruf erfolgt automatisch beim Auslösen des Interrupts durch den SysTick über die Vektortabelle.
- Zur Initialisierung des SysTick schreiben Sie zwei Funktionen: `void SysTickTime(int ms)`, in der Sie im CTRL-Register das Bit `CLKSOURCE` korrekt setzen (Wertebereich des LOAD-Registers beachten, es müssen auch Zeiten bis 1s einstellbar sein!) und das `LOAD`-Register des SysTick-Timers so belegen, dass er alle `ms` Millisekunden ausgelöst wird. Mit einer zweiten Funktion `void SysTickEnable(void)` starten Sie den Timer und lassen den Interrupt zu (über das `CTRL`-Register).

- Die Abfrage der Schalter und Anpassung der Geschwindigkeit erfolgt weiterhin in der Endlos-while-Schleife in `main`, dabei sollen die LEDs entsprechend der Schalterstellung mit den folgenden Wartezeiten weitergeschaltet werden: 00 → 1s, 01 → 0,75s, 10 → 0,5s, 11 → 0,25s.

Aufgabe 1.3: Analog-Digital-Wandler (ADC)
Mikrocontroller besitzen häufig einen Analog-Digital-Konverter (ADC), der analoge Messsignale in digitale Werte zur Weiterverarbeitung im Programm umwandelt. Der im Praktikum verwendete F446 hat zum Beispiel drei ADCs mit insgesamt 24 Kanälen, an denen Analogsignale angelegt werden können. Diese Kanäle können von allen drei ADCs eingelesen werden, dabei sind auch Sequenzen möglich, so dass ein ADC mehrere Kanäle nacheinander abfragen kann.

Auf dem Board im Praktikum befindet sich neben den Schiebeschaltern auch ein Potentiometer (weißer Drehknopf), der mit dem ersten Analogkanal verbunden ist. Durch Drehen des Potentiometers können damit beliebige Spannungen zwischen 0 und 3,3V an den ADC-Kanal angelegt werden. Diese Spannung soll eingelesen und in sinnvoller Form mit den linken vier LEDs angezeigt werden, so dass die Stellung des Potentiometers an den LEDs erkennbar ist.

Verwenden Sie die Projektvorlage `V1-4`, in die wir auch die folgenden Versuche integrieren. Ab sofort werden die Register nicht mehr wie bisher direkt beschrieben/geändert, sondern es wird dazu die Low-Level-Library des Controller-Herstellers benutzt. Die Dokumentation dazu finden Sie als PDF im Bereich „Allgemeine Unterlagen“ in Moodle (Dokumentation LL-Bibliothek) oder unter: http://www.disca.upv.es/aperles/arm_cortex_m3/llibre/st/STM32F439xx_User_Manual/group__stm32f4xx__ll__driver.html. Bitte verwenden Sie daraus nur die Funktionen, die mit LL beginnen, nicht die mit HAL.

Wir haben die Funktion `void ADCInit(int ADCChannel)` schon fertig programmiert und zur Verfügung gestellt; als Übergabeparameter sind nur 0 oder 1 für die beiden auf dem Board verdrahteten ADC-Kanäle zulässig. Das Potentiometer ist an Kanal 0 angeschlossen. Ergänzen Sie die Funktion `int ADCRead()` zum Einlesen eines Werts vom ADC. Verwenden Sie dazu Funktionen aus der LL-Bibliothek; gehen Sie dabei wie folgt vor:

Wert einlesen
- Wandlung starten
- Warten, bis das End-Of-Conversion-Flag gesetzt ist, erst dann ist der Wert vollständig gewandelt und gültig
- Einlesen des Werts

Danach implementieren Sie in der `while`-Schleife in der Funktion `main()` die Anzeige der Stellung des Potentiometers (die ADC-Werte liegen dabei zwischen 0…4095) mit Hilfe der LEDs 4..7 (die linken vier der LEDs auf dem Board; die rechten vier benötigen wir noch bei einem späteren Versuch). Überlegen Sie sich dazu vorher, wie Sie die Anzeige sinnvoll gestalten können.

In der Datei `LED.C` sind für die LED-Anzeige schon Funktionen vordefiniert, die Sie dazu nutzen können. Außerdem können Sie sich darin schon etwas mit der Struktur der Bibliothek vertraut machen.