Labor C Praktikum – FG Messtechnik

Praktikumsbericht über einen Aufgabenblock

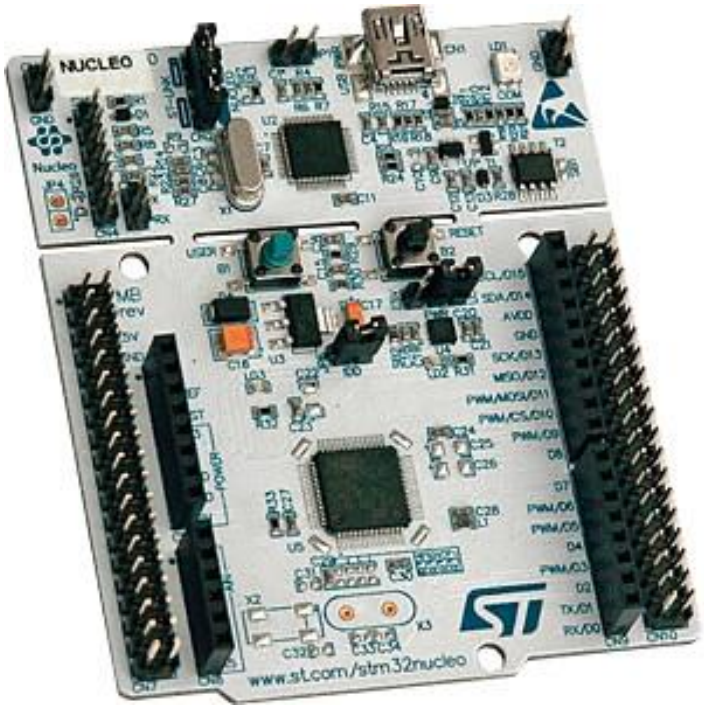
Parviz Moskalenko

Universität Kassel

Studiengang: Informatik

Matrikelnummer: 35827551

Es wurde gearbeitet mit dem



STM32 NUCLEO F446RE

Dieser Bericht handelt von: Block 1 – Ein- und Ausgaben

Block 1: Ein- und Ausgaben

Es gab zu diesem Aufgabenblock 3 Teilaufgaben. Die Aufgabenblöcke waren wie folgt unterteilt:

1. Blinkende LED
2. UART
3. Display

Ich werde jeden Aufgabenblock einzeln vorstellen und erklären. Folgende Bedingungen musste man erfüllen, um den Aufgabenblock 1 zu bestehen:

- Eine LED blinkt in einer definierten Frequenz
- Die Frequenz kann über die UART-Schnittstelle vorgegeben, verändert und auch dargestellt werden
- Die Frequenz wird im Display ausgegeben

Im Teilaufgabenblock 1 ging es darum, die integrierte LED auf dem Board mit einer gewissen Herzfrequenz blinken zu lassen. Dies könnte man mit der while-Schleife in der main-Funktion realisieren, damit ist das Board aber aufgrund vom endlosen Warten in der main-Funktion schon komplett ausgelastet, weswegen wir dies eleganter lösen mithilfe von einem Timer und dessen Interrupts.

Um dies zu realisieren müssen wir erstmal das .ioc file öffnen um die Peripherie zu initialisieren. Unter Pinout & Configuration findet man den Reiter Timer. Nun muss man eine geeigneten Timer aktivieren. Dies ist einem selbst überlassen, man muss nur darauf achten, dass man für den Timer Interrupts aktivieren kann. Wir haben dafür den Timer 2 gewählt. Um den Timer zu aktivieren, wählt man unter Clock Source Internal Clock aus. Der Timer 2 ist mit dem APB1 verbunden und hat somit, mit den Voreinstellungen, die wir bei der Projektanlegung getroffen haben über die Clock Configuration, eine Taktfrequenz von 90MHz. Nun aktivieren wir unter NVIC Settings bei dem Timer den globalen Interrupt. Diesen brauchen wir, um die LED zu aktivieren, bzw. zu deaktivieren. Um den Timer nun richtig einzustellen, müssen wir den Prescaler und den AutoReload richtig setzen, um somit die richtige Hz-Frequenz zu erreichen. Wir setzen nun den Prescaler auf 2^16-1. Der Timer ist ein 32 Bit Timer und zählt somit bis 2^32 mit 90MHz. Weil wir den Prescaler auf 2^16 setzen zählt der Timer nur noch mit 90.000.000/2^16Hz. Den Autoreload setzen wir erstmal auf 1000, dies wird aber später nochmal ausgeführt, da wir in diesem Aufgabenblock die Hz-Frequenz variierbar einstellen sollen. Da wir nun der Timer aktiviert haben, wird in der **stm324xx_it.c** Datei eine neue Interrupt-Funktion angelegt, welche ausgeführt wird, sobald der Timer überläuft. Da wir die LED zum Blinken bringen wollten, müssen wir diese auch noch initialisieren. Dies geschieht, indem man den Pin PA5 als GPIO-Output anlegt, denn dieser ist mit der LED verbunden. Um diesen Aufgabenteilblock abzuschließen, müssen wir nur noch zwei Zeilen in die **main.c** und in die **stm324xx_it.c** einfügen.

Gehen wir nun einmal in die Datei **stm324xx_it.c**. Dort finden wir die Funktion, welche ausgeführt wird, sobald der Timer über läuft. Diese Funktion nennt sich **TIM2_IRQHandler**. In diese Funktion fügen wir nun die Funktion ein, welche die LED aktiviert, bzw. deaktiviert. Um dies zu realisieren nutzen wir die Funktion **HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin)** und übergeben der Funktion der Port und den Pin. Nun wird jedes Mal, wenn der Timer überläuft die die LED an- bzw. ausgeschaltet, je nachdem wie der aktuelle Status ist. Um den Aufgabenteil zu vervollständigen, müssen wir nur noch den Timer starten, um die ganze Sache zum Laufen zu bringen. Dies machen wir in der **main.c**, indem wir einfach in den Block **USER CODE BEGIN 2** mit der Funktion **HAL_TIM_Base_Start_It(&htim2)**, welcher wir den Timer den wir starten möchten übergeben. Drücken wir nun „Run“ lädt er alles auf das Board und wir sehen nun, wie die LED blinkt.

In Teilaufgabenblock 2 geht es nun darum, mithilfe der UART-Schnittstelle, Werte an das Board zu übergeben, um dann mit den eingegebenen Werten die LED mit verschiedenen Hz-Frequenzen blinken zu lassen.

Um per UART Daten zu übersenden und zu empfangen, nutzen wir das Terminal Programm **HTerm**. Die benötigte UART-Schnittstelle **UART2** ist bereits voraktiviert, sodass wir diese nicht mehr aktivieren müssen. Da man per UART Daten empfangen, sowie senden kann, nutzen wir dafür die bereits vorhandenen HAL-Funktionen **HAL_UART_Recieve_IT**, zum Empfangen von Daten, sowie **HAL_UART_Transmit** um Daten per UART zu senden. Um UART nun nutzen zu können müssen wir nur noch die Callback-Funktion manuell einfügen, welche dann aufgerufen wird, sobald ein UART-Interrupt ausgelöst wird. Diese kann man dem Skript entnehmen und in die **main.c** einfügen.

Zuerst müssen wir im Device Configuration Tool das Interrupt für das UART aktivieren. Dafür gehen wir einfach auf **USART2** unter Connectivity und aktivieren in den NVIC Settings das Interrupt. Um zu prüfen, ob die übersendeten Daten per UART auch ankommen, übergeben wir jedes Mal am Anfang eine kleine Begrüßungsnachricht aus. Dies erreichen wir mithilfe der Transmit-Funktion. Wie das ganze konkret aussieht schauen wir uns nun an. Als Beispiel wollen wir nun den Projektnamen „Block1“ per UART übersenden. Wie wir bereits wissen, tun wir dies per **HAL_UART_Transmit** Funktion. Für unsere Nachricht benötigen wir ein char-Array, welches wir mit „**Block1\r\n**“ anlegen. Die letzten vier Zeichen sind für den Zeilenumbruch da, da wir nicht alles Darauffolgende in einer Zeile stehen haben wollen. Nun wollen wir unser char-Array transmitten. Dies sieht wie folgt aus:
HAL_UART_Transmit(&huart2, &projectName, sizeof(projectName), HAL_MAX_DELAY).
Die Übergabeparameter sind, &huart2, das ist die bereits vorangelegte UART-Schnittstelle, die wir ansprechen möchten, danach folgt unser char-Array, welches wir hier projectName genannt haben, daraufhin müssen wir dem UART noch sagen, wie viele Bytes wir übersenden möchten und zuletzt übergeben wir HAL_MAX_DELAY welches das größte vordefinierte Makro für den Timeout ist. Um nun die Daten noch per HTerm zu empfangen, müssen wir in HTerm den richtigen Port auswählen und HTerm noch verbinden. Dies wurde ausführlich im Skript erklärt, weshalb ich hierzu keine weiteren Erklärungen anstellen

möchte. Haben wir nun HTerm richtig konfiguriert und verbunden und starten das Board, wird uns angezeigt: „Block1“.

Nun haben wir fürs Erste alles erledigt und richtig eingerichtet im Bezug zum UART und widmen uns nun dem Teilaufgabenblock 3.

Im Teilaufgabenblock 3 geht es darum, das Display **SSD1306** richtig einzurichten und uns die eingegebene Herzfrequenz anzeigen zu lassen. Um das Display anzusprechen, nutzen wir die Datenschnittstelle **I2C**.

Um das Display nutzen zu können müssen wir zuerst das Display richtig anschließen. Die Praktikumsleitung war so nett und hat bereits die ganze benötigte Hardware auf einem extra Board mit dem Board verlötet und somit bereits Pins vorgegeben. Wir nutzen einfach 2 Klemmen und schließen das Display somit an das Board mit den Pins PB9 und PB8. Um die Pins richtig anzulegen, geht man wieder in das Device Configuration Tool und sucht unter Connectivity „I2C1“. Man wählt den Modus „I2C“. Es müssten nun unter den GPIO Settings automatisch die Pins PB9 und PB8 ausgewählt worden sein. Nun haben wir die I2C Schnittstelle richtig konfiguriert und können schon fast das Display ansprechen. Um das Display anzusprechen, müssen wir nun noch den Schalter auf „**ON**“ stellen, der auch mit auf dem Board verlötet ist und die Bibliotheken einbinden. Die Bibliotheken wurden uns von der Praktikumsleitung zur Verfügung gestellt und wir können diese herunterladen. Es gibt 4 Dateien, die wir nun richtig einfügen müssen. Die Dateien **ssd1306.c** und **fonts.c** müssen in die Projektunterordner „**\Core\Src**“ eingebunden werden, sowie die Dateien **ssd1306.h** und **fonts.h** in den Projektunterordner „**\Core\Inc**“, wodurch uns nun Funktionen und Schriftarten zur Verfügung stehen, damit wir das Display nun einfach ansprechen können. Um die Dateien noch nutzen zu können müssen wir noch per Präprozessoranweisung die Dateien einbinden, indem wir in die **main.c** schreiben: „**#include “ssd1306.c**“„ was wir für jede eingefügte Datei machen müssen. Nun sind wir soweit und können das Display relativ leicht im Code ansprechen.

Um den Aufgabenblock zu beenden, müssen wir nun alle Aufgabenteilblöcke “zusammenführen“ und ein paar Kleinigkeiten einfügen.

Zuerst kümmern wir uns darum, die LED in einer definierten Frequenz blinken zu lassen. Wie wir bereits vorher erwähnt haben bestimmt man die Frequenz, indem man den Autoreload und Prescaler richtig einstellt, wodurch der Timer in einer bestimmten Hz-Frequenz überläuft und die LED triggert. Wir möchten unsere in HTerm eingegebene Hz-Frequenz per UART empfangen und den Autoreload des Timer 2 so setzen, dass er mit der richtigen Hz-Frequenz blinkt. Fangen wir erstmal an und schauen uns an, wie wir Daten per UART empfangen können. Um die empfangenen Daten verarbeiten und nutzen zu können benötigen wir eine globale Variable, damit diese auch in der Callback-Funktion genutzt werden kann. Diese Variable speichert für uns die empfangenen Bytes die wir von der UART-Schnittstelle erhalten haben. Wir nennen diese Variable „**buffer**“ und legen sie global als ein char-Array an. Wir müssen diese Variable als char-Array anlegen, weil wir per UART-Schnittstelle ASCII-Zeichen empfangen. Dies führt aber nicht zu Problemen, da wir die empfangenen chars einfach zu Zahlenwerten casten können mithilfe einer Funktion, dazu

gleich mehr. Die bereits erwähnte Funktion **HAL_UART_Receive_IT(…)** löst ein UART-Interrupt aus, wodurch die UART-Callback Funktion aufgerufen wird, in der man dann die empfangenen Daten verarbeiten kann. Wir schreiben also nun in unsere **main.c** in den **USER CODE BLOCK 2** den Funktionsaufruf **HAL_UART_Receive_IT(&huart2, &buffer, 2)**, wobei wir mit &huart2 unsere gewählte UART-Schnittstelle übergeben, mit &buffer, das global angelegte char-Array zu Datenverarbeitung, und mit 2 die Anzahl der Bytes die wir empfangen möchten.

Warum haben wir 2 Bytes gewählt? Da wir mit dem menschlichen Auge sowieso keine 99 von 100 Herz unterscheiden können, reichen uns 2 Bytes um unsere Zahl zu empfangen. Wir gehen in diesem Projekt davon aus, das uns Hz-Frequenzen bis 26Hz ausreichen. Jede einzelne Zahl wird mit einem Byte übertragen. Somit haben wir in der Zahl 26, zwei Zahlen einmal die 2 und einmal die 6. Um dies zu übertragen, geben wir also nun 2 Bytes an. Wie läuft es nun ab, wenn man eine Hz-Frequenz von 4 haben möchte, aber die 4 hat so gesehen nur ein Byte? Würden wir nur die „4“ übertragen, füllt die **HAL_UART_Receive_IT** -Funktion automatisch mit Nullen auf. Also geben wir nur „4 “ ein, so übertragen wir die „40“. Wir lösen dies einfach, indem wir immer, wenn wir eine einstellige Hz-Frequenz übertragen, eine „0“ davor schreiben. Möchte man nun also eine Hz-Frequenz von 4, gibt man in HTerm ein: „04“.

Da wir die Funktion jetzt aufgerufen haben, wird gewartet bis die Bytes empfangen wurden und es wird ein UART-Interrupt ausgelöst. Geben wir nun „10“ in HTerm ein, wird die „10“ in unseren **buffer** gespeichert und die die Callback-Funktion welche wir zuvor in die **main.c** eingefügt haben wird aufgerufen. Da wir den **buffer** global angelegt haben, können wir ihn nun auch in der Funktion nutzen. Wir möchten nun unsere gespeicherten Daten vom **buffer** dafür nutzen, die richtige Hz-Frequenz für die LED zu setzen.

Um nachzuschauen, ob unser eingegebener Wert auch richtig übertragen wurde, schicken wir per UART direkt einmal den Buffer zurück und können ihn dann somit bei HTerm auslesen. Dafür nutzen wir erneut die **HAL_UART_Transmit(&huart2, buffer, 2, HAL_MAX_DELAY)** Funktion und übergeben unseren buffer.

Um nun den Autoreload so zu setzen, dass der Timer mit einer bestimmten Hz-Frequenz überläuft, müssen wir eine kleine Berechnung tätigen. Um den Wert zu speichern, legen wir eine neue int-Variable an mit dem Namen „hz“. Wir haben den Timer 2 bereits mit dem Prescaler von 2^16-1 runterdividiert. Den Autoreload möchten wir nun auf **(10000/buffer)/2** setzen. Dies entspricht dann der Hz-Frequenz mit der der Timer überläuft. Dabei müssen wir noch beachten, dass **buffer** ja aktuell noch ein char-Array darstellt und wir dieses char-Array zu einem Integer casten müssen. Dafür gibt es die schöne Funktion **atoi()** welche uns dann einen int-Wert zurückgibt, mit dem wir dann rechnen können. Konkrekt sieht das dann wie folgt aus: **int hz = round((10000/atoi(&buffer))/2);** . Die Funktion **round()** rundet den Wert. Nun müssen wir nurnoch den Autoreload auf **hz** setzen. Da wir nicht einfach während der Timer läuft, den Autoreload neu setzen können, müssen wir den Timer stoppen, den aktuellen Wert des Timers zurücksetzen, den Autoreload auf **hz** setzen, und den Timer wieder starten. Dies realisieren wir mit 4 Funktionsaufrufen, die wir von der HAL-Bibliothek gestellt bekommen. Im Code sieht das folgendermaßen aus:

Timer stoppen: **HAL_TIM_Base_Stop_IT(&htim2);**

Autoreload auf **hz** setzen: **__HAL_TIM_SET_AUTORELOAD(&htim2, hz);**

Aktuellen Timer Counter resetten bzw. auf 0 setzen: **__HAL_TIM_SET_COUNTER(&htim2, 0);**

Timer wieder starten mit neuen Werten: **HAL_TIM_Base_Start_IT(&htim2);**

Die Funktionsaufrufe sind selbsterklärend. Wir müssen jedes Mal darauf achten, dass wir auch unseren Timer 2 übergeben. Nun haben wir die Blinkfrequenz der LED auf unsere gewünschte Hz-Frequenz geändert. Um den Aufgabenblock zu beenden, müssen wir nur noch die Ausgabe auf dem Display realisieren. Dies geht relativ einfach, Dank der Bibliotheken.

Zuerst einmal müssen wir das Display initialisieren und den Hintergrund mit dunkler Farbe „füllen“. Da wir dies nicht immer wieder machen wollen, wenn die Callback-Funktion aufgerufen wird, weil dies einfach Zeit kostet und unnötig ist, reicht es, wenn wir dies einmal machen, in der main-Funktion. Dafür gibt es nun zwei einfache Funktionsaufrufe:

ssd1306_Init(&hi2c1); , **ssd1306_Fill(DISPLAY_COLOR_DARK);**

Um das Display zu initialisieren, übergeben wir der Init-Funktion unseren I2C1-Handler, der automatisch erzeugt wurde, als wir im Device Configuration Tool den I2C1 aktiviert haben. Danach füllen wir das Display dunkel. Nun da wir das Display initialisiert haben, können wir wieder in die Callback-Funktion springen und das Display ansprechen. Um an der oberen linken Ecke anzufangen, müssen wir den Startpunkt setzen, wo wir unseren Text schreiben wollen. Wir könnten den Text beliebig **verschieben**, indem man den „Cursor“ verschiebt. Da das Bildschirm oben links die Koordinaten (0,0) hat, setzen wir unseren Cursor auf 0,0 mit dem Funktionsaufruf **ssd1306_SetCursor(0, 0)**. Wir möchten auf dem Display stehen haben: „**hz: *buffer***“. Nun gibt es eine einfache Funktion, um Text auf das Display zu übertragen. Dieser Funktion übergibt man ein „String“, womit beim Cursor gestartet wird und der Text geschrieben wird nach rechts verlaufend. Zuerst müssen wir das „**hz:** “ schreiben, was wie folgt passiert: **ssd1306_WriteString(„hz: “, Font_7x10, DISPLAY_COLOR_LIGHT);**. Wir übergeben der Funktion den String den wir schreiben möchten, eine beliebig gewählte Schriftart und danach die Farbe, in welcher der Text erscheinen soll. Da wir nun dahinter noch unseren **buffer** schreiben möchten, müssen wir dies noch mit dem selben Funktionsaufruf machen, jedoch übergeben wir anstatt **„hz: “,** unser char-Array **buffer**. Damit das ganze dann auch noch wirklich auf das Display geschrieben wird, müssen wir das ganze Display einmal updaten mit **ssd1306_UpdateScreen()**. Nun wird uns unser gewünschter Text auf dem Display angezeigt.

Wir befinden uns nun immer noch in der Callback-Funktion. Da wir immer wieder Werte vom UART empfangen wollen, müssen wir eine „Endlosschleife“ erstellen, damit das Board immer wieder Werte vom UART erwartet, um die Frequenz der LED zu ändern. Dafür schreiben wir einfach ans Ende der Callback-Funktion erneut die Funktion **HAL_UART_Recieve_IT(…)**, wodurch wieder ein **UART-Interrupt** ausgelöst wird, sobald 2 Bytes eingegeben wurden und wir sind somit erneut in der Callback-Funktion gelandet. Dadurch können wir immer wieder neue Werte eingeben.

Schauen wir nun nach, ob wir alle Aufgaben erfolgreich erfüllt haben, dazu schauen wir uns nochmal die Anforderungen an den Block 1 an:

- Eine LED blinkt in einer definierten Frequenz
- Die Frequenz kann über die UART-Schnittstelle vorgegeben, verändert und auch dargestellt werden
- Die Frequenz wird im Display ausgegeben

Die erste Anforderung haben wir mittels Timer und dessen Interrupt realisiert, indem wir den Prescaler und den Autoreload richtig setzen, das Intervall des Überlaufs so timen, sodass wir die gewünschte Herzfrequenz erreichen.

Die zweite Anforderung haben wir auch erfolgreich erfüllt, indem wir per UART-Schnittstelle 2 Bytes empfangen, diese 2 Bytes in einen Integer-Wert umwandeln und mit diesem Wert den Timer anpassen. Des Weiteren lassen wir uns auch per UART-Schnittstelle unseren übergebenen Wert wieder ausgeben, um so zu sehen, welche Hz-Frequenz wir eingestellt haben.

Für die letzte Anforderung haben wir das Display richtig initialisiert, den Cursor gesetzt und lassen uns dort unsere gewünschte Hz-Frequenz ausgeben, indem wir unsere Hz-Frequenz per WriteString-Methode auf das Display schreiben.

Nun haben wir alle Aufgaben des Blocks erfüllt und können per UART-Schnittstelle Hz-Frequenzen an das Board übergeben, was die LED zu dieser Frequenz blinken lässt und den eingegebenen Wert per Display und UART-Schnittstelle ausgibt.

Parviz Moskalenko