# DEBUGGING WITH USART

INDINF09

18. JUNI 2015

STEFAN ERCEG
4AHITT

# Inhalt



*Github-Link: https://github.com/serceg-tgm/4AHITT-SYT/tree/indinf/IndInf09*

*Github-Tag: erceg_indinf09*

# 1. Aufgabenstellung

**Debugging mit USART**

Geben Sie die Zeichenkette "Button gedrueckt!" aus, wenn der User-Button gedrückt wird. Regeln Sie diese Ausgabe mittels Interrupts.

**Vertiefung**

Verbinden Sie zwei Mikrokontroller miteinander und programmieren Sie eine vereinfachte Ping-Pong Implementierung mittels des USART. Schicken Sie jeweils gegengleich ein Ping und Pong hin und zurück und zeigen Sie entsprechend die Information mit der LED4 und LED5. Beschreiben Sie die notwendige Verwendung und Konfiguration des USART in einem Protokoll.

# 2. Zeitabschätzung

| Teilaufgabe | benötigte Zeit |
|---|---|
| Programm implementieren | 180 Minuten |
| Protokoll schreiben | 60 Minuten |
| | |
| *Gesamt* | **240 Minuten (4 h)** |

Dazu muss erwähnt werden, dass die Vertiefung mittels der Ping-Pong Implementierung nicht bei der Zeitabschätzung miteinberechnet wurde!

# 3. Tatsächlicher Zeitaufwand

| Teilaufgabe | benötigte Zeit |
|---|---|
| Programm implementieren | 135 Minuten |
| Protokoll schreiben | 30 Minuten |
| | |
| *Gesamt* | **165 Minuten (2 h 45 min)** |

Der tatsächliche Zeitaufwand für die Übung betrug um 1 Stunde und 15 Minuten weniger als der geplante. Da diese Übung im Großen und Ganzen „nur“ ein Zusammenmergen der beiden von Herr Professor Borko zur Verfügung gestellten Interrupt- und UART-Übungen war, fiel es nicht bedeutend schwer, diese Übung durchzuführen.

# 4. Arbeitsdurchführung

## 4.1 Allgemeines

Zu Beginn wurde das Github-Repository vom Herr Professor Borko [1] geklont:

```
git clone https://github.com/mborko/stm32f3-template.git
```

In diesem Repository befindet sich bereits im Verzeichnis `src/interrupt` eine Interrupt- und im Verzeichnis `src/uart` eine UART-Übung. Somit habe ich die beiden Übungen „zusammengemergt“ und damit die Aufgabenstellung gelöst. Um diese Übung funktionsfähig zu machen, musste im Nachhinein noch eine Funktion im File `stm32f30x_it.c` hinzugefügt werden (siehe Punkt 4.3).

Als erstes habe ich mal einen eigenen Ordner `indinf09` in meinem Projekt-Ordner erstellt, wo ich das Makefile und alles, was sich im Verzeichnis `src/interrupt` befindet, kopiert habe.

Somit musste man im Makefile die Zeile `PROJ := src/blink` auf `PROJ := .` ändern, damit das richtige Programm ausgeführt wird.

## 4.2 Änderungen an der Datei „main.c“

```
int main(void) {

	STM_EVAL_PBInit(BUTTON_USER,BUTTON_MODE_EXTI);
	EXTI0_Config();

	RCC_Configuration();
	GPIO_Configuration();
	USART1_Configuration();

	while (1);

}
```

Zu Beginn wird in der main-Funktion PA0 als Interrupt-Modus aktiviert und die Funktion `EXTI0_Config()` aufgerufen, welche zur Konfiguration des Interrupts dient.

In der Funktion `EXTI0_Config(void)`, die, wie vorher erwähnt, zur Konfiguration des Interrupts dient, wird in der Zeile

```
	EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
```

noch ein `Falling` hinzugefügt, daher sieht die Zeile folgendermaßen aus:

```
	EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
```

Somit wird das Interrupt bei Falling Edge (sinkende Flanke; von 1 auf 0) und Rising Edge (steigende Flanke; von 0 auf 1) ausgelöst und der Zustand des Buttons überprüft.

Die 3 Funktionen `RCC_Configuration(void)`, `GPIO_Configuration(void)` und `USART1_Configuration(void)` wurden aus der UART-Übung herauskopiert:

- In der Funktion `RCC_Configuration(void)` wird die Clock enabled.
- In der Funktion `GPIO_Configuration(void)` legt man fest, welcher Pin für das Senden und welcher für das Empfangen zuständig sein sollen.
- In der Funktion `USART1_Configuration(void)` werden die Ressourcen für USART konfiguriert. Dabei stellt man u.a. die Datenübertragungsrate, die Wortlänge, die Stop-Bits und das Parity-Bit ein.

Zum Schluss der main-Funktion wird noch die wiederkehrende Anweisung `while(1);` durchgeführt.

## 4.3 Änderungen an der Datei „stm32f30x_it.c“

```
void EXTI0_IRQHandler(void) {

	/* Print the message after clicking on the button */
	char* msg = "Button gedrueckt!";
	printmessage(msg);

	/* Clear the EXTI line 0 pending bit */
	EXTI_ClearITPendingBit(USER_BUTTON_EXTI_LINE);

}

void printmessage(char* message) {

	for(i=0; message[i]; i++) {

		while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
		USART_SendData(USART1, message[i]);

	}

}
```

In der Funktion `EXTI0_IRQHandler(void)` wird zu Beginn die von mir erstellte Funktion `printmessage(char* message)`, dem als Parameter der Text „Button gedrueckt!“ als ein Pointer auf ein Char-Array übergeben wurde, aufgerufen. In dieser wird zu Beginn überprüft, ob man das nächste Zeichen schon senden darf. Wenn dies zutrifft, wird das bestimmte Zeichen mittels `USART_SendData` übertragen.

Mit der Zeile `EXTI_ClearITPendingBit(USER_BUTTON_EXTI_LINE);` wird das für das Interrupt zuständige Bit gecleart. Somit kann beim Drücken des Buttons erneut reagiert werden.

## 5. Lessons learned

- Verwendung von USART
- Wissen über Interrupts erweitert

## 6. Quellenangaben

[1] Github-User „mborko” (2014, 2015). Github-Repository „stm32f3-template” [Online]. Available at: https://github.com/mborko/stm32f3-template [zuletzt abgerufen am 18.06.2015]