Übung 09: Embedded Betriebssysteme

Hardware:

- Basis: Arduino, Steckbrett, Kabel
- Taster

Hinweise:

(1) Datenblatt ATmega2560
(2) FreeRTOS Reference Manual: https://www.freertos.org/wp-content/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf
(3) FreeRTOS Kernel Developer Docs: https://www.freertos.org/features.html

In diesem Praktikum geht es um ein erstes Kennenlernen des Betriebssystems FreeRTOS. **Für Aufgabe 2 bis 5 ist je bereits Code vorgegeben. Nur Stellen mit `//TODO` ergänzen!**

Aufgabe 1: Einbindung von FreeRTOS

a) Starten Sie die Arduino IDE und installieren Sie die Bibliothek `FreeRTOS`:
→ Menü „*Werkzeuge – Bibliothek verwalten*“, nach „`FreeRTOS`“ suchen und installieren. Das Paket stammt von Richard Barry, aktuelle Version ist 10.3.0-9 (Stand 02.07.2020).

b) Wieviel Speicher in Byte benötigt ein leeres Programm mit leerer `setup`- und `loop`-Methode?
→ Ausgabe beim Kompilieren bzw. Hochladen ablesen!

c) Wieviel Speicher in Byte benötigt ein Programm, falls Sie die FreeRTOS Bibliothek einbinden? Was ist somit ein entscheidender Nachteil eines Betriebssystems?
→ Menü „*Sketch – Bibliothek einbinden*“, dort „`FreeRTOS`“ auswählen.

Aufgabe 2: Programm mit 1 Task

Der vorgegebene Code erzeugt nur 1 FreeRTOS Task. Ziel ist es, das die interne LED (***PB 7, Digital Pin 13***) jede Sekunde Ihren Zustand ändert, also entweder aus- bzw. angeschaltet wird.

a) Welche Priorität hat dieser Task? Wie groß ist sein Stack? → Hinweis (3), Seite 34

b) Was macht das Kommando `vTaskDelay(pdMS_TO_TICKS(1000))`? In welchem Zustand ist der Task unmittelbar vor und unmittelbar nach dem Ausführen dieses Kommandos?

c) Im Task fehlen noch 2 Kommandos. Es muss für die interne LED (siehe oben) das entsprechende Bit im `DDR`-Register auf Ausgang geschaltet werden und das entsprechende Bit im `PORT`-Register muss „getoggelt“ werden. Ergänzen Sie das und testen Sie! (gute Wiederholung!)

Aufgabe 3: Zwei Tasks mit verschiedenen Prioritäten

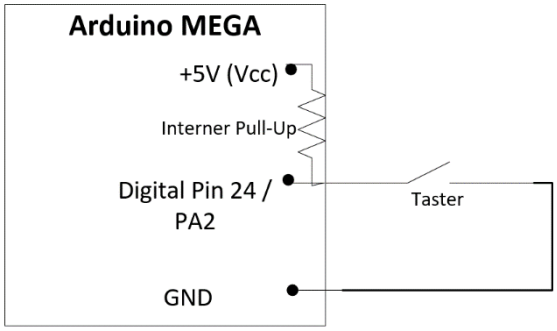
Der komplette Code ist bereits vorgegeben. Es gibt 2 Tasks. Der „*Busy Task*“ wacht jede Sekunde auf, blockiert dann aber sogleich wieder. Der „*Blinking Task*“ toggelt jede Sekunde den Zustand der internen LED, dazwischen blockiert er.

Testen Sie mit verschiedenen Prioritäten für die Tasks und beobachten Sie jeweils ob die interne LED blinkt oder nicht. Versuchen Sie Ihre Beobachtung zu erklären! Lesen Sie zur Begründung auch den folgenden Abschnitt: https://www.freertos.org/RTOS-task-priority.html

	Priorität „*Blinking Task*“	Priorität „*Busy Task*“
a)	3	2
b)	2	2
c)	2	3
d)	2	3 + zusätzlich `vTaskDelay(1000)` durch `delay(3000)` ersetzen)

Aufgabe 4: Kommunikation zwischen Tasks - Semaphore

Es gibt weiterhin 2 Tasks: Der “*Blinking Task*” aus 3) bleibt erhalten, der „*Busy Task*“ wird durch einen Task „*Button Task*“ ersetzt, der das Betätigen eines Tasters durch Polling erkennt. Die LED soll nur getoggelt werden, nachdem der Taster gedrückt wurde. Das signalisiert der „*Button Task*“ dem „*Blinking Task*“ über einen Semaphor. Es ist akzeptabel, wenn die LED mit einer maximalen Zeitverzögerung von ca. 1 Sekunde auf das Drücken des Tasters reagiert. Der Code ist größtenteils vorgegeben.



a) Schließen Sie den Taster an. Liegt am Eingang `PA2` des Mikrocontrollers LOW oder HIGH an, wenn der Taster gedrückt wird?

b) Konfigurieren Sie den Pin `PA2` **(=*Digital Pin 24*)** im „Button Task“ als Eingang!
- Entsprechendes `DDR`-Register konfigurieren
- Internen Pull-Up aktivieren: → Handbuch S. 68, Abschnitt 13.2.1 beachten!

c) Ergänzen Sie den Semaphor und testen Sie!
- Code erzeugt bereits binären Semaphor: `xSemaphoreCreateBinary()`
- Geben Sie den Semaphor frei, wenn der Taster gedrückt wird: → Hinweis (3), Seite 236
- Der „Blinking Task“ belegt den Semaphore → Hinweis (3), Seite 244

d) Warum wird der Taster im Code implizit entprellt?

Aufgabe 5: Kommunikation zwischen Tasks - Queues

Ihr Mikrocontroller weiß nicht, wann und wie oft er Nachrichten über `UART0` empfängt. Um nichts zu verpassen, implementieren Sie den Zeichenempfang über einen `UART RX Complete Interrupt.` Die ISR bleibt kurz, Sie legen das empfangene Zeichen nur in eine Queue! Ein extra „`UARTTask`“ kümmert sich zu einem beliebigen späteren Zeitpunkt um das empfangene Zeichen (*Deferred interrupt processing*). In unserem Test wird das empfangene Zeichen gespiegelt und (an den PC) zurückgesendet. Es gibt also nur 1 Task und 1 ISR!

Der Code ist bereits fast vollständig, versuchen Sie ihn zu verstehen! Sie müssen nur noch im „*UART Task*“ prüfen, ob etwas in der Queue ist und es dann zurücksenden → Hinweis (3), Seite 186

Testen Sie, in dem Sie über die serielle Konsole ein Zeichen an den Mikrocontroller senden.