Übung 5

Interrupts, Memory Pool

Aufgabe 5.1 Button-0 Interrupt

Der Taster T0 auf dem CARME-M4 Kit soll interruptgesteuert ausgelesen und anschliessend in einem Task verarbeitet werden. Für die Synchronisation zwischen ISR und Task wird eine Semaphore verwendet. Der Task soll anschliessend einen kurzen Text auf dem Display ausgeben.

Aufgabenstellung:

- Initialisieren Sie als Erstes den Taster T0 so, dass bei Betätigung ein Interrupt ausgelöst wird. Den Code kenne Sie aus dem Modul “Hardwarenahe Softwareentwicklung“. Taster T0 ist mit GPIO C7 (Port C Pin 7) des Prozessors verbunden.

```
#include <stm32f4xx_exti.h>
#include <carme_io1.h>

void btn_InitInterrupt(void) {

  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Connect EXTI7  to C7 (Button0) */
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOC, EXTI_PinSource7);

  /* Configure EXTI7 line */
  EXTI_InitStructure.EXTI_Line = EXTI_Line7;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Enable and set EXTI9_5 Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 8;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

- Schreiben Sie einen Interrupt-Handler für den Taster-Interrupt. Da auf dem Cortex-M4 mehrere Interrupts auf derselben Vektoradresse zusammengefasst werden (EXTI5 bis EXTI9) müssen Sie im Handler die Quelle des Interrupts noch prüfen. Der Code dazu lautet:

```
void EXTI9_5_IRQHandler(void) {
  /* Button T0 interrupt on Port C Pin (Line) 7 */
  if (EXTI_GetITStatus(EXTI_Line7) != RESET) {
    /* Clear Interrupt and call Interrupt Service Routine */
    EXTI_ClearITPendingBit(EXTI_Line7);
    btn_IrqT0();
  }
}
```

Wenn Sie mit Atollic TrueSTUDIO ein FreeRTOS-Projekt für das CARME-Kit erstellen, wird in der Datei “stm32f4xx_it.c“ ein EXTI9_5_IRQHandler()“ erstellt. Sie können diesen Code für den Tasterinterrupt ergänzen, oder Sie können einen neuen Handler in Ihrem Projekt definieren, dann müssen Sie aber den Handler in der Datei “stm32f4xx_it.c“ auskommentieren.

- Programmieren Sie die ISR btn_IrqT0(), welche die Semaphore semBtn setzt. Zudem soll diese ISR eine LED toggeln.
- Programmieren Sie einen Task, welcher auf die Semaphore semBtn wartet und anschliessend den Text auf das Display ausgibt (counter mit Anzahl Betätigungen).

Tipps:

Beachten Sie folgende Punkte bei der Implementation:

- Innerhalb der ISR dürfen in freeRTOS nur API-Funktionen aufgerufen werden, welche die Endung “FromISR“ oder “FROM_ISR“ besitzen.
- In den Folien zum Unterricht oder im Dokument “freeRTOS auf dem CARME-M4“ im Kapitel 4.6 (Interrupt Handling) wird beschrieben, wie eine ISR korrekt aufgebaut wird.

Aufgabe 5.2 Log-Meldungen mit Memory-Pool

Ergänzen Sie Übung 4-2 mit einem Memory Pool.

Aufgabenstellung:

- Bevor Sie die Nachricht über die Message Queue an den Gatekeeper Task versenden, müssen Sie einen Memory-Block (welcher die Grösse einer Log-Nachricht hat) beim Memory Pool Manager anfordern.
- Den angeforderten Memory-Block können Sie nun mit den gewünschten Daten abfüllen.
- Anstelle einer kompletten Log-Nachricht legen Sie nur noch einen Pointer auf den Memory-Block in die Message Queue.
- Innerhalb des Gatekeeper Tasks empfangen Sie nun einen Pointer auf eine Log-Nachricht, welche Sie auf der seriellen Schnittstelle ausgeben können.
- Zum Schluss muss der Memory-Block dem Memory-Pool zurückgegeben werden.

Beantworten Sie folgende Fragen:

- Was muss gemacht werden, wenn kein freier Memory Block verfügbar ist?
- Bei der Erzeugung des Memory-Pools kann die Anzahl Blöcke angegeben werden. Wie viele Blöcke sind sinnvoll?
- Ab welcher Blockgrösse ist die Erstellung eines Memory-Pools sinnvoll?