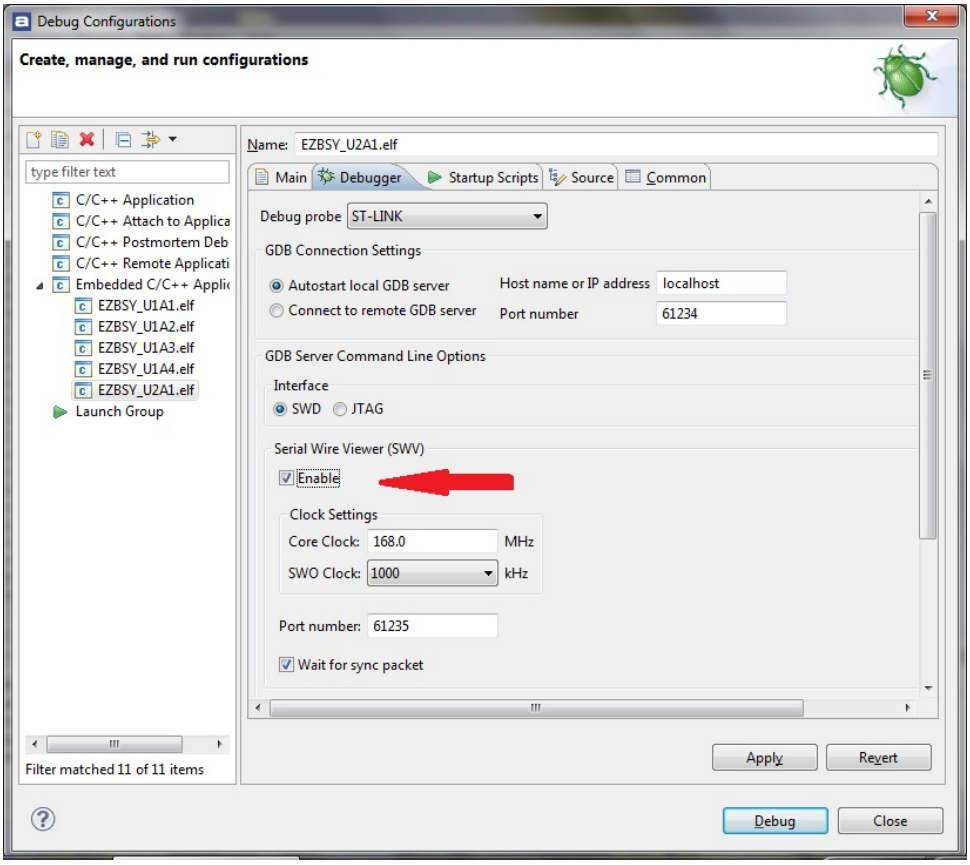
Übung 2

Tasks, Context-Switch, Kernel Awarness

Aufgabe 2.1 FreeRTOS Kernel Awareness

Mit der eingesetzten Attolic IDE ist es möglich, den Zustand von Tasks, Semaphoren, Message Queues sowie Software Timer zu verfolgen. Verwenden Sie als Ausgangslage Übung 1 Aufgabe 1.3 “LED Lauflicht” und gehen Sie folgendermassen vor.

- Lesen Sie Application Note “AN1202: FreeRTOS Kernel Awareness“, welche Sie auf Moodle finden. Interessant sind vor allem die Seiten 4-7 sowie 9+10.
- Geben Sie den beiden Tasks einen eindeutigen beschreibenden Namen (mit weniger als 16 Zeichen).
- Aktivieren Sie für die aktuelle Applikation den SWV (Serial Wire Viewer) unter Run → Debug Configurations...



- Debuggen Sie die Applikation und verfolgen Sie die Task-Zustände mit Hilfe der Kernel -Awareness (Task-Liste wird immer nur aktualisiert wenn der Debugger angehalten wird, betätigen Sie das Pausen-Icon “Suspend“). Beantworten Sie folgende Fragen:
  - Wieso ist (fast) immer nur der Idle-Task im RUNNING Zustand? Wie könnten Sie forcieren, dass ein anderer Task öfter im RUNNING Zustand ist? Was hat das für Konsequenzen?

- Wie kann aus den Informationen die Stackauslastung der einzelnen Task berechnet werden? Wie gross ist die Stackauslastung der einzelnen Tasks in Prozent?

Aufgabe 2.2 Stack-Informationen auf dem LCD ausgeben

Der Zustand der Stacks verschiedener Tasks soll auf dem LCD ausgegeben werden. Verwenden Sie als Ausgangslage Aufgabe 1.3 “LED Lauflicht“. Ergänzen Sie diese Aufgabe mit einem zusätzlichen Task, welcher folgende Ausgaben auf dem LCD macht:

- Stack-Zustände (freier Speicher in Byte) für jeden Task. Benutzen Sie hierzu die FreeRTOS API Funktion *uxTaskGetStackHighWaterMark()*.
- Anzahl der laufenden Tasks innerhalb der Applikation. Diese erhalten Sie mit der Funktion uxTaskGetNumberOfTasks().
- Den noch verfügbaren Heap Speicher, welcher mit der Funktion xPortGetFreeHeapSize() abgefragt werden kann.

Aktivieren Sie auch für diese Applikation das Kernel Aware Debugging aus Aufgabe 2.1 und vergleichen Sie den Stackzustand welchen Sie auf dem Display ausgeben mit jenem welcher durch die Kernel Awareness dargestellt wird. Ändern Sie anschliessend die Stackgrösse des LED-Tasks und prüfen Sie, welchen Einfluss dies auf den freien Stack des Tasks hat.

Welche Aussagen können Sie bezüglich der Anzahl Tasks machen?

Aufgabe 2.3 Software Timer

Aufgabe 1.4 aus Übung 1 soll mit Software Timern realisiert werden. Folgende Software Timer Callback-Funktionen sind zu implementieren:

- Switch State: Die Callback-Funktion wird alle 100ms (100 Ticks) aufgerufen, liest die Zustände der Schalter ein und speichert diese in der globalen Variablen *u8SwitchState*.
- Button State: Diese Callback-Funktion pollt die Zustände der Taster alle 50ms und legt diese in der Variablen *u8ButtonState* ab.
- LED Lauflicht: Diese Callback-Funktion wird alle 400ms aufgerufen und implementiert ein Lauflicht auf dem CARME-M4.

Zusätzlich zu den Timer Callback-Funktionen ist ein Task zu implementieren, welcher die Zustände der Schalter und der Taster auf dem LCD ausgibt.

Mit Hilfe der Kernel-Awareness können in der Attolic IDE auch die Zustände der Timer sichtbar gemacht werden.

Tipps zur Implementation:

Für diese Aufgabe brauchen Sie die FreeRTOS-API-Funktionen xTimerCreate() und xTimerStart(). Mehr Informationen zu diesen Funktionen finden Sie auf der Webseite von FreeRTOS (www.freertos.org) unter API-Reference.

Beantworten Sie folgende Fragen:

- Was ist der Vorteil einer Timer-Callback Funktion gegenüber der Erstellung eines eigenen Tasks?
- Was sind die Nachteile von Timer-Callback Funktionen?
- Dürfen innerhalb der Timer-Callback Funktion blockierende Funktionen aufgerufen werden? Begründen Sie ihre Antwort.
- Welche Informationen zu den Timer Callback-Funktionen werden mit Hilfe der Kernel Awareness innerhalb der Attolic IDE sichtbar?