Kapitel 8

MicroC/OS-II

In diesem Kapitel wollen wir uns nun einem „richtigen“ Betriebssystem zuwenden, dem Echtzeitbetriebssystem MicroC/OS-II (eigentlich „µC/OS-II“) von Jean J. Labrosse, welches er über seine Firma Micriμm vermarktet. Das System wird ausführlich (jede Zeile des Quellcodes) im gleichnamigen Buch des Verlags CMP Books erläutert.

Für die Nutzung mit der Nios II CPU wurde das System von Altera portiert und der Altera-HAL entsprechend angepasst. So werden beispielsweise einige Makros bereitgestellt, die je nach Konfiguration im Multitasking-Betrieb Betriebssystemfunktionen und im Singletask-Betrieb Funktionen des HAL verwenden. Weitere Informationen entnehmen Sie bitte Kaptel 10 des Nios II Software Developer's Handbook.

Aufgabe 1

Erstellen Sie ein System auf Grundlage des MicroC/OS. Das zu Grunde liegende Nios II System benötigt dafür nur noch einen Timer, den Systemtimer, der die höchste Interrupt-Priorität besitzen sollte und in diesem Fall eine Periode von 1 ms (also eine Frequenz von 1 kHz). Er muss nicht programmierbar sein. Im Nios II IDE erstellen Sie dann ein neues Projekt basierend auf dem Beispiel „Hello MicroC/OS-II“. Bitte ändern Sie die Prioritäten der Tasks auf 4 und 5, da die obersten sowie untersten vier Prioritäten eigentlich für das Betriebssystem reserviert sind.

Starten Sie das System und versuchen Sie, die Funktionsweise nachzuvollziehen.

Aufgabe 2

Der Software-Debugger des Nios II IDE erkennt die Tasks eines MicroC/OS Systems und kann sie und ihren Funktions-Stack im pausierten Modus anzeigen. Ein manuelles „Umschalten“ zwischen den Tasks ist aber verständlicherweise nicht möglich.

Führen Sie im Debugger den Start des Systems Schritt für Schritt aus (Stepping). Können Sie den Beginn des Multitasking-Betriebs erkennen? Es gibt außer den explizit erstellen Tasks noch den Idle-Task, der mit der niedrigsten Priorität läuft sowie je nach Konfiguration noch einen Statistik-Task, für den die zweitniedrigste Priorität vorgesehen ist. Können Sie die einzelnen Tasks identifizieren?

Experimentieren Sie auch mit Breakpoints innerhalb der Tasks.

Aufgabe 3

Der Statistik-Task stellt jede Sekunde die aktuelle Auslastung der CPU (in Prozent) in der globalen Variable `OSCPUUsage` bereit. Dies funktioniert allerdings erst nach der Initialisierung mittels der Funktion `OSStatInit()`, für deren korrekte Verwendung das Beispielprogramm etwas modifiziert werden muss.

Die Auslastung des Prozessors soll natürlich nur die Benutzer-Tasks umfassen, da der Idle- und Statistiktasks nicht wichtig für die Funktion des Systems sind. Daher muss die Initialisierung mittels `OSStatInit()` genau dann erfolgen, wenn der Multitasking-Betrieb bereits aktiviert ist, aber noch keine Benutzerfunktionen ablaufen. Ausser den Idle- und Statistik-Tasks soll daher zur Zeit der Initialisierung nur ein einziger weiterer Task existieren, nämlich genau der, welcher die Initialisierungsfunktion aufruft. Üblicherweise erstellt dieser dann zunächst die weiteren Tasks und führt dann (bei Bedarf) eigene Aufgaben aus.

Passen Sie Ihr System entsprechend an, um die Auslastung der CPU regelmäßig ausgeben zu können. Erstellen Sie dafür einen weiteren Task, der wie beschrieben nach dem Beginn des Multitasking-Betriebs zunächst die Funktion `OSStatInit()` aufruft und dann die restlichen Tasks erstellt. Seine weitere Funktionalität könnte dann beispielsweise die regelmäßige Ausgabe der Systemauslastung sein. Achten Sie in diesem Fall darauf, die Priorität des Tasks nicht zu hoch zu wählen. Da die Prioritäten `OS_LOWEST_PRIO` bis `OS_LOWEST_PRIO - 3` für das Betriebssystem reserviert sind, ist die niedrigste, Ihnen zur Verfügung stehende Priorität `OS_LOWEST_PRIO - 4`.

Experimentieren Sie mit unterschiedlichen Auslastungen des Systems, beispielsweise, indem einige Ihrer Tasks einen Teil mit aktivem Warten verbringen.

Aufgabe 4

Übertragen Sie die Experimente aus dem vorigen Kapitel auf das Echtzeitberiebssystem. Wo gibt es Gemeinsamkeiten, wo Unterschiede im Ablauf?