Kapitel 7

Mini-Betriebssystem

Das folgende Kapitel führt durch die Entwicklung eines kleinen Betriebsystems, das für die quasi-nebenläufige Abarbeitung mehrerer Tasks geeignet ist. Es sollen dabei verschiedene Möglichkeiten der Realisierung vorgestellt werden. Allen gemein ist dabei, dass kein wirklicher Wechsel des Prozessorkontextes vorgenommen wird, während dies bei „richtigen“ Betriebssystemen eine elementare Funktion darstellt. Auch weitere Aufgaben eines Betriebssystems wie beispielsweise die Verwaltung von Ressourcen, sollen an dieser Stelle nicht betrachtet werden.

Die Grundidee der Mini-Betriebssysteme ist wie auch bei ihren großen Vorbildern die Verwendung von Interrupts, um einen laufenden Task unterbrechen zu können. Um die Notwendigkeit eines Kontextwechsels zu vermeiden, sollen zunächst aber für jeden Task ein separater Interrupt verwendet werden. Daraus ergibt sich unmittelbar ein erster Lösungsansatz, nämlich die Verwendung von Interrupt-Service-Routinen als Tasks. Die Interrupts sind untereinander priorisiert, daher lassen sich prioritätenbasierte Scheduling-Verfahren mit so einem System abbilden.

Beim Nios II besteht ferner die Möglichkeit, laufende Interrupt-Service-Routinen durch höherpriore Interrupts zu unterbrechen. Dieses Konzept wird als „verschachtelte Interrupts“ bzw. „nested interrupts“ bezeichnet. Dies kann innerhalb jeder ISR kann mittels der Funtion `alt_irq_interruptible` angezeigt werden. Ein Aufruf von `alt_irq_non_interruptible` beendet diesen Modus wieder. Üblicherweise werden diese Aufrufe direkt am Anfang bzw. Ende der ISR durchgeführt. *Wichtig:* Die Funktionen müssen immer genau paarweise verwendet werden, jeder Aktivierung der Verschachtelung muss innerhalb der selben ISR die Deaktivierung folgen.

Die Prototypen der Funktionen sind die folgenden:

```
alt_u32 alt_irq_interruptible(alt_u32 id)
void    alt_irq_non_interruptible(alt_u32 status);
```

Ein Beispiel zeigt ihre Verwendung, dabei sind im Hinblick auf die Aufgaben in diesem Kapitel die Funktionsaufrufe per Präprozessoranweisung abschaltbar.

```
#if (PREEMPT == 1)
    alt_u32 int_status;

    int_status = alt_irq_interruptible(id);
#endif

...

/* ISR code */

...

#if (PREEMPT == 1)
    alt_irq_non_interruptible(int_status);
#endif
```

Die Auslösung der Interruptrequests kann nun durch Geräte erfolgen, für zeitbasierte Anforderungen können Timer verwendet werden.

Aufgabe 1

Erstellen Sie mittels der oben vorgestellten Idee ein eigenes kleines Betriebssystem, welches die Abarbeitung von mindestens vier Tasks ermöglicht. Um einfacher an reproduzierbare Ergebnisse zu gelangen, sollen alle notwendigen Interrupts von unabhängigen Timern erzeugt werden. Um die volle Programmierbarkeit der Timer zu erreichen, sollen diese im „full-featured“ Modus eingebunden werden. Passen Sie Ihr SOPC-Design entsprechend an.

Erstellen Sie dann die entsprechende Software, bei der die Tasks direkt in den ISRs ablaufen. Für einen ersten Versuch soll jeder Task einen eigenen Zähler bei jeder Ausführung inkrementieren. In der main()-Funktion sollen zunächst die Timer konfiguriert werden und dann in einer Schleife ein Idle-Zähler inkrementiert sowie periodisch eine Übersicht über die Zähler auf der Konsole ausgege-

ben werden. Funktionen für den Zugriff auf die Timer finden Sie in der Datei `altera_avalon_timer_regs.h`.

Experimentieren Sie mit verschiedenen Perioden, überprüfen Sie die Einhaltung der Prioritäten und ermitteln Sie mittels verschachtelten Interrupts den Unterschied zwischen präemptivem und nicht-präemptivem Multitasking. Innerhalb der ISRs können Sie durch aktives Warten mittels `usleep()` Prozessorauslastung generieren und somit unterschiedlich lange Ausführungszeiten simulieren. Weiterhin kann es sinnvoll sein, erweiterte (statistische) Analysen der Zählerwerte in der Hauptschleife durchzuführen.

Aufgabe 2

Sollen die Tasks nicht innerhalb der ISRs, sondern im Kontext des Hauptprogramms abgearbeitet werden, beispielsweise, weil blockierende Funktionen verwendet werden sollen, so bietet sich eine zweite Möglichkeit für die Implementierung eines Mini-Betriebssystems an.

Die jeweiligen ISRs werden nur dafür benutzt, die Tasks auf „aktiv“ zu setzen, während in der main()-Funktion eine Schleife abläuft, welche die Tasks sortiert nach ihrer Priorität auf eben jene Aktivitätsanforderung überprüft und dann bei Bedarf die entsprechende Funktion des Tasks aufruft. Zuvor muss verständlicherweise das Signal zurückgesetzt werden. Wird nach der Abarbeitung eines Tasks sichergestellt, das die Schleife direkt wieder von vorn mit der Überprüfung beginnt, so ist sichergestellt, dass immer der Task mit der höchsten aktiven Priorität läuft.

Setzen Sie nun diese Idee in einem eigenen System um. Mit geeigneter Wahl einer Datenstruktur für die Verwaltung der Tasks sowie der Verwendung von Funktions-Pointern können Sie die Scheduler-Schleife sehr einfach und unabhängig von der Anzahl an Tasks gestalten.

Führen Sie wie in Aufgabe 1 verschiedene Experimente durch und untersuchen Sie das Verhalten des Systems. Handelt es sich bei Ihrer Implementierung um ein präemptives oder ein nicht-präemptives System?

Freiwillige Sonderaufgabe

Bei zeitgesteuerter Taskaktivierung ist es sehr unschön, auf Grund unterschiedlicher Perioden für jeden Task einen eigenen Hardware-Timer verwenden zu müssen. Entwickeln Sie daher eine möglichst flexible Lösung, um alle zeitbasierten Aktivierungen von Tasks von nur einem Timer erledigen zu lassen. Dabei sollen die Perioden auch während der Laufzeit konfigurierbar sein. Beachten Sie die Bedeutung der Granularität des entsprechenden Timers.