AUFGABE 1: HALLO WELT

Diese Aufgabe dient dem initialen Einrichten Ihrer Entwicklungsumgebung und dem ersten Kontakt mit dem Echtzeitbetriebssystem *eCos*. Ziel ist es einen ersten Einblick in die Möglichkeiten der Entwicklungsumgebung zu erhalten.

Aufgabenstellung

A. Kopieren und entpacken Sie die Vorgabe in ein beliebiges Arbeitsverzeichnis und setzen Sie die nötigen Umgebungsvariablen durch Aufruf von *source* `ecosenv.sh` Sie können den Inhalt der Datei auch in Ihre `~/.bashrc` einfügen, um den Vorgang zu automatisieren. Fügen Sie den von uns zur Verfügung gestellten Treiber für den Sound Blaster 16 zum Applikationsrumpf `hello.c` und zur `CMakeLists.txt`-Datei hinzu. Nun können Sie die Makefiles generieren und die noch funktionslose Anwendung erstmals kompilieren.

☞ Übungsfolien

B. Die Erzeugung eines geeigneten Threadsystems ist Bestandteil aller folgenden Übungsaufgaben.
Implementieren Sie in `hello.c` die `cyg_user_start()`-Funktion, in der Sie einen Thread mit Hilfe von `cyg_thread_create()` erzeugen. Die Threadfunktion soll periodisch, jede Sekunde, mittels `printf()` die Zeichenkette „`Hallo Welt!\n`“ auf der seriellen Schnittstelle ausgeben. Um einen periodischen Thread zu simulieren, bietet sich die Funktion `cyg_thread_delay()` an. Ein Tick entspricht in unserem Fall 10 ms.

C. Mittels des periodischen Threads kann nun ein erstes Signal erzeugt werden. Der Sound-Blaster-Treiber bietet die Funktion `ezs_sb16_play_sample()`, mit deren Hilfe Sie analoge Werte im Bereich von 0 bis $2^8 - 1$ ausgeben können. Erzeugen Sie ein Sinussignal und verwenden Sie für diesen Zweck die Funktion `sin()`. *Welche Frequenz hat das erzeugte Signal? Was passiert, wenn Sie die Verzögerung des Threads verändern?* Sehen Sie sich auch die Aufzeichnung der Werte an.

☞ `math.h`

D. Für das Echtzeitverhalten ist neben der eigenen Implementierung immer das Gesamtsystem zu betrachten. In der letzten

Teilaufgabe soll daher die Aufrufhierarchie eines `printf()`-Aufrufs analysiert werden. Öffnen Sie hierfür die Emulation im Debugging-Modus (`make debug`) und setzen Sie einen Breakpoint auf die *tiefste* Funktion `cyg_hal_plf_serial_putc()` – ein Breakpoint lässt sich über die grafische Oberfläche oder die Befehlszeile (unten) per `break <Funktionsname>` setzen.
Starten Sie dann die Ausführung mit `continue`. *Wichtig:* Der Befehl `run` funktioniert hier nicht. Sobald die Ausführung an dem gesetzten Breakpoint gestoppt hat, können Sie die Aufrufhierarchie über den Befehl `backtrace` ausgeben. *Erkunden Sie nun mit Hilfe des Debuggers den Aufrufgraphen der* `printf()`*-Funktion und erklären Sie grob die einzelnen Funktionen. Welche aus Echtzeitsicht kritischen Stellen fallen Ihnen hier besonders auf? Wie sieht die Situation im Vergleich dazu bei* `ezs_sb16_play_sample()` *aus?*

Hinweise

- Erforderliche Dateien: wie vorgegeben.
- Bearbeitung: Gruppe mit je drei Teilnehmern.
- Abgabezeit: 8.11.2012
- Fragen bitte an `i4ezs@lists.cs.fau.de`
- Die Funktionsreferenz der *eCos* API findet sich unter `http://ecos.sourceware.org/docs-latest/ref/kernel.html`
- Eigene Quelldateien müssen Sie in die `CMakeLists.txt` eintragen. Bei jeder Änderung müssen Sie im Build-Verzeichnis `cmake` erneut aufrufen.
- *Wichtig:* Bei nachträglichen Änderungen an der Vorgabe (z. B. zur Fehlerkorrektur) *muss* der Inhalt des Build-Verzeichnisses gelöscht und `cmake` erneut aufgerufen werden. Dieses Vorgehen empfiehlt sich auch bei nicht nachvollziehbaren Fehlern.