## AUFGABE 2: HALLO ZEIT

In dieser Übungsaufgabe geht es nun erstmals um die Interaktion mit einem (imaginären) physikalischen System und damit dem Faktor Zeit. Wie in der Vorlesung gezeigt, gibt es Berührungspunkte zwischen dem physikalischen Objekt und dem kontrollierenden Echtzeitsystem. Die resultierenden Aufgaben und Terminvorgaben müssen von dem Echtzeitsystem ausgeführt beziehungsweise eingehalten werden. Ziel dieser Aufgabe ist es, einen ersten Eindruck über die zeitlichen Abläufe und Abhängigkeiten zu gewinnen.

### *Aufgabenstellung*

A. *Zeitmessung mit der* `libEZS`*:* Um die zeitlichen Abläufe im System messen zu können, muss zunächst die `libEZS` geeignet erweitert werden. Einen hardwareunabhängigen Zähler haben wir bereits vorgegeben. Sie können auf diesen mittels der Funktion `ezs_counter_get()` zugreifen und den aktuellen Wert in Ticks auslesen. *Implementieren Sie nun die Funktionen* `ezs_watch_start()` *und* `ezs_watch_stop()` *in* `ezs_stopwatch.c`. Damit mehrere Zeitmessungen parallel und über Funktionsgrenzen hinweg erfolgen können, erhalten beide Funktionen einen Zeiger auf den Zustand der Messung durch den Aufrufer – die Zustandsvariable muss in der Anwendung später global[1] angelegt werden. Die Funktion `ezs_watch_stop()` liefert das Ergebnis der Messung in Form von Ticks zurück.

☞ `libEZS/...`

☞ `ezs_counter.h`

B. *WCET-Simulation:* Um zukünftig den Rechenzeitaufwand einer komplexeren Anwendung zu simulieren, ist ein weiteres Instrument notwendig. *Implementieren Sie hierfür die Funktion* `ezs_watch_wcet()`, *welche aktiv wartet und erst mit dem Erreichen der gewünschten Zeit zurückkehrt.* Setzen sie zur Zeitkontrolle wiederum den bereitgestellten Zähler ein und implementieren Sie den in der Tafelübung präsentierten Algorithmus – ihre Funktion muss mit Unterbrechungen umgehen können.

[1] Gültigkeit von Variablen in C, Folie 37ff: `http://www4.cs.fau.de/Lehre/SS11/V_SP1/Folien/SP-02a-A4.pdf`

c. *Das Abtasttheorem:* Die Rekonstruktion von kontinuierlichen Signalen, beispielsweise von Musikstücken, ist eine typische Aufgabe eines Echtzeitsystems. Hierbei gilt das Nyquist-Shannonsche Abtasttheorem[2], wonach für eine korrekte Rekonstruktion des zeitdiskreten Signals die Abtastfrequenz $f_{sample}$ so zu wählen ist, dass sie mindestens $2 \cdot f_{max}$ (Maximalfrequenz des Signals) ist. Bei der Wiedergabe digitalisierter Musikstücke ist diese Abtastfrequenz (sampling rate) von entscheidender Bedeutung für die verzerrungsfreie Rekonstruktion. Der Einhaltung des Abtasttheorems fällt also eine zentrale Bedeutung bei der Implementierung eines Echtzeitsystems zu. In dieser Aufgabe lassen wir Qemu einen überlagerten Sinus erzeugen (3 Hz und 29 Hz), den Sie mit Hilfe des Soundblasters abtasten können. *Lesen Sie die Abtastwerte mit der korrekten Abtastfrequenz ein (*`ezs_sb16_record_sample()`*), geben Sie diese umgehend über die Soundkarte (*`ezs_sb16_play_sample()`*) aus und betrachten Sie das Ergebnis. Mit welcher Rate müssen Sie die Abtastwerte wiedergeben? Was passiert, wenn Sie diese Rate verdoppeln? Was passiert, wenn Sie die Rate mit einer zufälligen Abweichung von ±20% nicht einhalten?*
Die Auflösung des Zeitgebers haben wir herfür auf 1 ms erhöht. `cyg_thread_delay(1)` entspricht also 1 ms Pause.

d. *Latenzzeit:* Unter Latenz versteht man die Zeit zwischen dem Auftreten eines Ereignisses (z. B. eines Interrupts) und der Reaktion (z. B. Öffnen eines Ventils) durch das Echtzeitsystem. Die Latenzzeit hängt dabei nicht alleine vom Rechenzeitbedarf der Ereignisbehandlung sondern von vielen Faktoren ab.
In dieser Aufgabe setzen wir für die Erzeugung von Ereignissen den Tastaturinterrupt ein. Zu diesem Zweck wird in der Vorgabe bereits der entsprechende Interrupt initialisiert und bearbeitet. Die `printf()`-Ausgabe im Interrupthandler dient nur zur Visualiserung und sollte bei späteren Messungen auskommentiert werden. Ebenfalls in der Vorgabe enthalten ist die Interruptbehandlung sowie zwei Handler-Funktionen die von Ihnen ausgefüllt bzw. angepasst werden müssen.
*Implementieren Sie einen weiteren Thread (Keyboard-Thread) der die eingehenden Daten von der Tastatur per* `printf()` *ausgibt.* (Sie müs-

[2] `http://de.wikipedia.org/wiki/Abtasttheorem`

sen die Keycodes nicht auswerten! Es genügt den Datenwert darzustellen.) Der Thread wird durch die Interruptbehandlung aktiviert und gibt nach erfolgter Ausgabe die Kontrolle wieder ab. Vergeben Sie die Prioritäten so, dass der Thread aus Teilaufgabe C (Abtast-Thread) eine niedrigere Priorität (> Zahl) aufweist. Verwenden Sie die Funktionen zur Zeitmessung aus Teilaufgabe A um die Latenz zwischen dem Auftreten des Interrupts und der Ausgabe durch den Thread zu bestimmen. Für die Umsetzung müssen Sie die Funktionen `keyb_isr_handler()` und `keyb_dsr_handler()` der Interruptbehandlung geeignet (Aktivierung, Messung) erweitern. *Was beobachten Sie?*

☞ `cyg_thread_resume()`
☞ `cyg_thread_suspend()`

Als letztes soll der Abtast-Thread eine höhere Priorität (< Zahl) als der Keyboard-Thread erhalten. Darüber hinaus soll er zusätzliche Rechenzeit benötigen. Verwenden sie hierfür die Funktion `ezs_watch_wcet()` und wählen sie als Zeit 1/5 der Periode des Threads (z. B. eine Verzögerung von 1 ms → WCET von 200 µs. Beachten Sie `libEZS`-Ticks ≠ *eCos*-Ticks!). Wie verhält es sich jetzt mit der Latenz? Wodurch kommt der Unterschied zur vorherigen Messung zustande?

## *Erweiterte Aufgabe*

A. *Angewandte digitale Signalverarbeitung:* Bitte teilen Sie dem Übungsleiter mit Abgabe dieser Übung mit, ob Sie an den erweiterten Übungen teilnehmen wollen – die folgende Teilaufgabe ist dann verpflichtend. Wir empfehlen den Teilnehmern der grundlegenden Übungen sich ebenfalls an dieser Teilaufgabe zu versuchen.
Im Folgenden werden wir eine realistischere Anwendung in Form einer digitalen Signalverarbeitung betrachten, konkret geht es um die Filterung des Signals mittels eines Tiefpassfilters[3]. Die Auswirkungen von Fehlern sind hier deutlich subtiler.
Aus dem bereits in der grundlegenden Übung abgetasteten Signal sollen Sie im folgenden die höhere der beiden Frequenzen herausgefiltern. Die Implementierung und Parametrierung des Tiefpassfilters ist abhängig von den Signalparametern. Verwenden

[3] `http://de.wikipedia.org/wiki/Tiefpass`

sie den bereitgestellten[4] Generator um ein angepasstes Filter zu erhalten. Verwenden Sie 0.1 für den `beta`-Parameter und bestimmen Sie die Impulslänge anhand der angegebenen Formel, alle weiteren Parameter ergeben sich aus Ihren Einstellungen (in unserem Beispiel ist eine Grenzfrequenz von 15 Hz sinnvoll). Kopieren Sie den generierten C-Code in die bestehende Entry-Funktion des Abtast-Thread und passen Sie ihn gegebenenfalls an.
Ist alles korrekt eingestellt, wird die höhere der beiden Frequenzen herausgefiltert. *Was passiert wenn Sie jetzt die Periode des Abtast-Thread verändern?*

B. *Volle Kontrolle:* Nutzen Sie abschließend den Keyboard-Thread um zwischen einer gefilterten und einer ungefilterten Ausgabe hin- und herschalten zu können (z. B. durch Drücken eines bestimmten Buchstabens). Macht die Sache deutlich einfacher, oder?

## *Hinweise*

- Erforderliche Dateien: wie vorgegeben.
- Bearbeitung: Gruppe mit je zwei/drei Teilnehmern.
- Abgabezeit: 22.11.2012
- 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.

[4] `http://www-users.cs.york.ac.uk/~fisher/mkfilter/racos.html`