AUFGABE 2: ANTWORTZEIT

In den ersten beiden Teilaufgaben dieser Aufgabe sollen Sie nützliche Funktionen für den Umgang mit Zeit in einem Rechensystem implementieren, die wir im weiteren Verlauf der Übung immer wieder verwenden werden. Die beiden letzten Teilaufgaben sollen Ihnen dann ein erstes Gefühl für die Probleme beim Zusammenspiel von periodischen und nicht-periodischen Ereignissen in einem Echtzeitsystem vermitteln.

1 Aufgabenstellung

1.1 Zeitmessung mit der libEZS:

Um die zeitlichen Abläufe im System messen zu können, muss zunächst die `libEZS` erweitert werden. Daher beziehen sich die meisten Dateinamen in dieser Aufgabe auf Dateien im Unterverzeichnis `libEZS`. 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. Lesen Sie die Dokumentation der von uns bereitgestellten Funktionen!

☞ libEZS/...

☞ include/ezs_counter.h

☞ make doc

☞ ezs_counter_get_resolution

Teilaufgabe 1.

Implementieren Sie die Funktionen `ezs_watch_start()` und `ezs_watch_stop()` in `libEZS/src/ezs_stopwatch.c`. Beachten sie hierbei auch die vorgegebene Signatur und Kommentare Was bedeuten die angegebenen Datentypen für Ihre zukünftigen Messungen?

☞ include/ezs_stopwatch.h

Antwort:

Hinweise: 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, die diesen Zustand abbildet, muss daher später in der Anwendung global[1] angelegt werden. Die Funktion `ezs_watch_stop()` liefert das Ergebnis der Messung in Form von Zeitgeber-Ticks zurück. Die Dauer eines solchen Ticks ist prinzipiell hardwareabhängig und wird wie in der Tafelübung besprochen von einer Struktur aus Dividend und Divisor beschrieben.

☞ ezs_counter_get_resolution

[1] Gültigkeit von Variablen in C, Folie 37ff: `https://www4.cs.fau.de/Lehre/SS15/V_SP1/Vorlesung/Folien/SP1-02-A4.pdf`

1.2 WCET-Simulation:

Um in dieser und den folgenden Aufgaben möglichst reale Anwendungssysteme betrachten zu können, müssen wir den Rechenzeitaufwand komplexer Anwendung simulieren können. Hierfür soll die Funktion `ezs_lose_time()` genutzt werden. In der von uns vorgegebenen Implementierung nutzt sie den EZS-Zeitgeber um aktiv zu warten, bis die vorgegebene Anzahl Ticks vergangen ist.

☞ ezs_stopwatch.c

Teilaufgabe 2.

Nutzen Sie den in der Tafelübung präsentierten Algorithmus um die Funktion so zu erweitern, dass sie auch mit *Unterbrechungen* umgehen kann. Ignorieren Sie hierfür zunächst den zweiten Parameter `percentage`. Weshalb ist das Beachten der Unterbrechungen notwendig?

Antwort:

Stellen Sie sicher, dass Ihre Funktion nie *mehr* Zeit verbraucht als der übergebene Zeitwert vorschreibt. Woraus ergibt sich diese Anforderung?

Antwort:

Teilaufgabe 3.

Testen Sie mithilfe der in der vorherigen Aufgabe entwickelten Funktionen, ob Ihre `ezs_lose_time()` die Zeitvorgabe einhält!

Teilaufgabe 4.

Erweitern Sie `ezs_lose_time()` nun so, dass ein zufällig gewählter Anteil der angefragten Zeit *nicht* verbraucht wird. Die maximale Grösse dieses zufälligen Anteils soll über den zweiten Parameter `percentage` der Funktion wählbar sein.

☞ `rand()%100`

1.3 Signalerzeugung:

Die Wiedergabe (Rekonstruktion) von kontinuierlichen Signalen, beispielsweise von Musikstücken, ist eine typische Aufgabe eines Echtzeitsystems. Hierbei gilt das Nyquist-Shannon-Abtasttheorem[2], wonach für eine korrekte Wiedergabe eines zeitdiskreten

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

Signals die Abtastfrequenz f_{sample} so zu wählen ist, dass sie größer $2 \cdot f_{\text{max}}$ (Maximalfrequenz des Signals) ist. Bei der Wiedergabe digitalisierter Musikstücke ist diese Abtastfrequenz (engl. sampling rate) von entscheidender Bedeutung für die verzerrungsfreie Rekonstruktion analoger Signale. Der Einhaltung des Abtasttheorems fällt also eine zentrale Bedeutung bei der Implementierung eines solchen Echtzeitsystems zu.

Teilaufgabe 5.

Verwenden Sie, ähnlich wie in der vorangegangenen Aufgabe, die Funktion `sinf()` um diesmal ein überlagertes Sinus-Signal zu erzeugen. Dieses soll eine Komponente mit einer Frequenz von 2 Hz und eine mit 13 Hz enthalten. Geben Sie den errechneten Signalverlauf auf dem Analog-Digital-Umsetzer so aus, dass das erwünschte analoge Signal korrekt rekonstruiert wird. *Mit welcher minimalen Rate müssen Sie die Abtastwerte wiedergeben?*

Antwort:

Was passiert, wenn Sie diese Rate verdoppeln?

Antwort:

Verwenden Sie Ihre `ezs_lose_time()` um zusätzlich zu Ihrer mit `ezs_delay()` eingebrachten Verzögerung Rechenlast in der Größenordnung Ihrer Abtastrate zu simulieren. Nutzen Sie den Parameter `percentage` um bis zu 90% der Arbeitslast nicht zu verbrauchen. *Wie wirkt sich die so eingebrachte Rechenlast auf die Periodizität Ihrer Aufgabe aus? Betrachten Sie auch die Fouriertransformierte des so wiedergegebenen Signals am Oszilloskop.*

Antwort:

1.4 Antwortzeit:

Unter Antwortzeit versteht man die Zeit zwischen dem Auftreten eines Ereignisses (z. B. eines Interrupts) und dem Bereitstellen eines Ergebnisses (z. B. Öffnen eines Ventils) durch das Echtzeitsystem.

In dieser Aufgabe setzen wir für die Erzeugung von Ereignissen die serielle Schnittstelle ein. Diese teilt durch das Auslösen eines Interrupts mit, wenn ein Zeichen eingelesen

wurde. Wir haben den entsprechenden Interrupt bereits in der Vorgabe für Sie aufgesetzt und eine rudimentäre Interruptbehandlung implementiert.

Teilaufgabe 6.

Inwiefern ist die von uns bereitgestellte Interruptbehandlung für ein komplexes Echtzeitsystem nicht geeignet?

Antwort:

Teilaufgabe 7.

Was wird durch den Aufruf der Interrupt-Acknowledge-Funktion wem signalisiert?

Antwort:

Implementieren Sie unter Nutzung der in der Übung vorgestellten Konzepte *DSR* und *Faden* eine bessere Interruptbehandlung, die die eingelesenen Zeichen mit Hilfe der Funktion `ezs_printf()` auf der seriellen Schnittstelle im Faden-Kontext ausgibt. Vergeben Sie die Prioritäten für diese neue Aufgaben so, dass die Signalerzeugung aus der vorangegangenen Teilaufgabe eine niedrigere Priorität (größere Zahl) erhält.

Teilaufgabe 8.

Verwenden Sie die zuvor implementierte Zeitmessung per Systemzeitgeber, um die Antwortzeit zwischen dem Auftreten des Interrupts und der Ausgabe zu ermitteln. *Wo beginnt die Messung? Wo ist sie zu Ende?*

Antwort:

Geben Sie die Ergebnisse Ihrer Messung in Nanosekunden wiederum über die serielle Schnittstelle aus.

☞ ezs_print_measurement

Teilaufgabe 9.

Nutzen Sie die cutecom-Funktion „Send File“, um die serielle Schnittstelle auszulasten. *Speichern Sie die in beiden Fällen gemessenen Werte nach der Messung über die serielle Schnittstelle auf Ihrem Desktop-PC ab. Wie verhält sich das Sinussignal während der Übertragung? Wie verhalten sich die Antwortzeiten?*

Antwort:

Teilaufgabe 10.

Vertauschen Sie nun die Prioritäten so, dass die Wiedergabe des Sinus-Signals die höchste Priorität(niedrigste Zahl) erhält. *Wie verhält es sich jetzt mit der Antwortzeit? Wodurch kommt der Unterschied zur vorherigen Messung zustande?*

Antwort:

2 Erweiterte Aufgabe

Die Erweiterten Übungsaufgaben sind nur für TeilnehmerInnen verpflichtend, die das 7,5-ECTS-Modul belegen.

Teilaufgabe 11.

Bestimmen Sie den zusätzlichen Rechenaufwand, der durch die Nutzung der Funktionen `ezs_watch_start()` und `ezs_watch_stop()` eingebracht wird und dadurch die Messung verfälscht. *Entwerfen Sie einen entsprechender Messaufbau. Wie muss dieser aussehen?*

Antwort:

Implementieren Sie Ihren Messaufbau und werten Sie Ihre Ergebnisse statistisch aus. *Wie hoch ist der durch die Messung induzierte Fehler?*

Antwort:

Teilaufgabe 12.

Wie kann der Messaufwand durch entsprechende Programmierung vermindert werden?

Antwort:

Hinweise

- Bearbeitung: Gruppe mit je drei Teilnehmern.
- Abgabezeit: 18.11.2015
- Fragen bitte an `i4ezs@lists.cs.fau.de`