AUFGABE 5: EXTENDED SCOPE

In dieser Aufgabe wird das Oszilloskop um die Ausgabe des Zeitbereichsignals und eine Befehlsschnittstelle erweitert. Hierfür wird eCos im ereignisgesteuerten Betrieb verwendet – vergeben Sie die Prioritäten nach dem RMA.

	Bezeichnung	Periode ms	WCET ms
T_1	Abtastung Signal	4	0,5
T_2	Prokrastinator	10	1,0
T_3	Analyse LDS	1000	?
T_4	Darstellung	250	?

T_1 Tastet den Soundkarteneingang ab und fügt die Werte in einen Ringpuffer mit 256 (`TIME_DOMAIN_LENGTH`) Elementen ein.

T_2 Simuliert nur Laufzeit.

T_3 Liest die Werte aus dem Ringpuffer und ruft die vorgegebene Funktion `ezs_easy_pds()` zur LDS-Analyse auf.

T_4 Stellt wahlweise das LDS oder das Signal auf dem Framebuffer dar (grundlegende Übung: nur LDS). Es nutzt dafür die ebenfalls vorgegebenen Funktionen `ezs_plot_pds()` und `ezs_plot_signal()`.

Für den Datenaustausch zwischen T_1, T_3 und T_4 bieten sich globale Arrays an. Die Funktion `ezs_easy_pds()` legt ihre Ausgangswerte in ein Array der Größe 128 ab. Dieses können Sie direkt in `ezs_plot_pds()` nutzen. Die genaue Schnittstellenbeschreibung können Sie den Headerdateien entnehmen. Sie finden das oben genannte Aufgabensystem bereits in weiten Teilen in der Vorgabe, auch die globalen Puffer und Daten sind von uns bereits angelegt worden. Die Vorgabe soll Ihnen die Aufgabe erleichtern, Sie müssen sich aber nicht zwingend daran halten.

☞ FFT_LENGTH

Aufgabenstellung

A. *Laufzeitmessung:* Ergänzen Sie die Vorgabe so, dass die Aufgaben ihre Funktion auch erfüllen (das LDS muss gezeichnet werden)

und simulieren Sie die WCET soweit oben angegeben. Passen Sie den Phasenversatz so an, dass zumindest T_1 und T_2 sich nicht überlappen. Sie müssen nicht den vollen Ablaufplan aufstellen – dieser wäre sehr groß. Messen Sie die Laufzeit von T_3 und T_4 und nehmen Sie das Maximum als Annäherung für die WCET. *Wie erreichen Sie eine möglichst störungsfreie und genaue Messung?*

	Bezeichnung	Min. Zwischenankunftszeit ms	WCET ms
T_5	Byteweiser Empfang	?	–
T_6	Dekodierung	?	0,5

T_5 baut aus den empfangenen Zeichen eine Zeichenkette auf. Hierfür wird die Funktion `packet_receive()` genutzt.

T_6 untersucht die vollständige Zeichenkette und extrahiert mittels `decode_command()` gültige Befehle.

Der Datenaustausch zwischen der ISR, T_5 und T_6 erfolgt wieder über globale Puffer. Zeichen dürfen in dieser Lösung also verloren gehen! Implementieren Sie die Funktionalität in den beiden Funktionen und nicht in Fäden/ISR/DSR, dies erleichtert Ihnen spätere Teilaufgaben.

B. *Implementierung der aperiodische Steuerung:* In dieser Teilaufgabe sollen Sie das Oszilloskop um eine externe Steuerung per Tastatur erweitern. Die Verarbeitung der eingelesenen Befehle ist eine typische aperiodische Aufgabe, zu deren Lösung Sie die Konzepte aus der Vorlesung anwenden können. Das System soll folgende Kommandos auswerten können:

Befehl	Parameter	Beschreibung
`display`	`<signal, pds>`	Umschaltung der Anzeige
`trigger`	`<off, on>`	Signaltrigger ein- / ausschalten
`tlevel`	`<rise, fall>`	Flanke auf die getriggert werden soll

Lesen Sie in der ISR jedes Zeichen einzeln aus und machen Sie dieses der zugehörigen DSR bzw. T_5 zugänglich. Die DSR nutzt die von Ihnen zu implementierende Funktion `int packet_receive(char`

byte) um einzelne Zeichen bis zum nächsten Zeilenvorschubzeichen '\n' zu puffern.
Halten Sie die Implementierung so kurz wie möglich. Dimensionieren Sie den Puffer entsprechend der maximalen Befehlslänge (+2) und führen Sie diesen als Ringpuffer aus. Ersetzen Sie unbedingt das Zeilenvorschubzeichen '\n' mit '\0', um die Zeichenkette zu terminieren. Die Funktion kehrt bei erfolgreichem Paketempfang mit 1 zurück und setzt den Ringpuffer zurück. Ist noch kein Zeilenvorschubzeichen empfangen worden, so kehrt sie mit 0 zurück.
Nachdem ein Kommando vollständig empfangen wurde, muss es dekodiert werden. Implementieren Sie hierfür die Funktion `enum Command decode_command(void)`. Sie versucht die empfangene Zeichenkette zu interpretieren und daraus gültige Kommandos zu dekodieren. Wir empfehlen Ihnen als Rückgabewert der Funktion das vorgegebene Enum[1] `Command` zu nutzen, welches die sechs möglichen Befehlskombinationen als Bitmaske kodiert.
Die Funktion wird erst in der nachfolgenden Teilaufgabe tatsächlich aufgerufen. Verwenden Sie die Funktion `strncmp()`, um Zeichenketten zu vergleichen. Vergessen Sie die WCET-Simulation nicht.

☞ `man 3 strncmp`

Vor der weiteren Umsetzung der Steuerung müssen Sie nun zunächst die minimale Zwischenankunftszeit der aperiodischen Aufgaben T_5 bestimmen. Ermitteln Sie diese durch Zeitmessung. *Wie können Sie eine minimalen Zwischenankunftszeit im Falle von T_5 erzwingen? Welche minimale Zwischenankunftszeit ergibt sich daraus für T_6? Wie lässt sie sich deutlich vergrößern?*

c. *Abbildung der aperiodische Steuerung:*

Rufen sie nun `decode_command()` an geeigneter Stelle auf und realisieren Sie nacheinander die drei verschiedenen Ausführungskonzepte für aperiodische Aufgaben (Unterbrecher-, Hintergrundbetrieb, periodischer Zusteller). Messen Sie jeweils die Antwortzeit von T_6. *Welches Problem ergibt sich aus der Nutzung der globalen Puffer zum Datenaustausch? Wie könnte man dieses Problem vermeiden?*
Es genügt die Funktion `decode_command()` als Implementierung von T_6 an geeigneter Stelle aufzurufen. Für zwei der drei Konzepte

[1] `http://en.wikipedia.org/wiki/Enumerated_type#C_and_syntactically_similar_languages`

benötigen Sie jetzt einen eigenständigen Faden. Beachten Sie die ermittelte Zwischenankunftszeit und nutzen Sie das Prioritätsgefüge aus. Belassen Sie die nicht genutzten Varianten als Kommentar in Ihrem Code.

D. *Umschaltung der Anzeige:* Im letzten Schritt soll nun noch tatsächliche Umschaltung der Anzeige umgesetzt werden. Stellen Sie für diese Teilaufgabe T_6 auf Hintergrundbetrieb ein. Um T_4 die Umschaltung zu signalisieren, eignet sich das eCos-Event-Konzept. Setzen Sie die Eventmaske in T_6 mittels `cyg_flag_setbits()` und prüfen Sie diese in der weiterhin periodischen Aufgabe T_4. *Welche Variante zur Überprüfung der Events muss hier verwendet werden? Welchem Konzept ähnelt diese Konstellation? Welche Probleme können hier entstehen?*

Sichern Sie Ihre Implementierung für die spätere Abgabe.

Erweiterte Aufgabe

A. *Implementierung der Signal-Trigger-Funktion:* Verwenden Sie für die erweiterte Aufgabe folgendes Task-System:

	Bezeichnung	**Periode** ms	**WCET** ms
T_2	Signal Trigger	4	0,5
T_4	Darstellung LDS/Signal	1000	?

Ein Trigger wird dazu verwendet die Ausgabe eines Oszilloskops auf die Frequenz des Signals zu synchronisieren und so eine stabile Anzeige des Signalverlaufs zu erreichen – d. h. das Signal „wandert“ nicht mehr. Ziel der erweiterten Aufgabe ist es, Aufgabe T_2 um eine Trigger-Funktion für das von der Soundkarte eingelesene Signal zu ergänzen. Implementieren Sie den Trigger so, dass er bei einer einstellbaren Flanke auslöst. Sie können davon ausgehen, dass eine steigende Flanke vorliegt, wenn der aktuelle Wert des Signals größer als 128 und der vorherige Wert kleiner als 128 ist. Der Zusammenhang für eine fallende Flanke verhält sich umgekehrt.

	Bezeichnung	**Min. Zwischenankunftszeit** ms	**WCET** ms
T_7	Darstellung Trigger	?	-

Die Darstellung im Trigger-Betrieb unterscheidet sich von der Bisherigen und erfolgt aperiodisch. Implementieren Sie die Aufgabe T_7, welche die bis zum Trigger-Ereignis aufgezeichneten Werte von T_1 mittels `ezs_plot_signal()` darstellt. Nutzen Sie Events um die Aufgaben T_1, T_2 und T_7 geeignet zu koordinieren. Beachten Sie hierbei, dass T_1 in jedem Fall weiter Daten aufzeichnen muss und somit T_7 nicht auf denselben Daten arbeiten kann. Nutzen Sie den Mailbox-Mechanismus von eCos um dieses Problem zu lösen.

B. *Vollständige Steuerung:* Vervollständigen Sie nun die Steuerung der Oszilloskopfunktionen. Aus den möglichen Optionen ergeben sich zwei Betriebsmodi: Bei aktivierter Trigger-Funktion erfolgt die Anzeige (Zeitsignal) aperiodisch, sonst periodisch (Zeitsignal oder LDS). Nutzen Sie beliebige Mechanismen (Schaltervariablen, Events, Mailboxen, ...), um die verschiedenen Betriebsarten umzusetzen. *Welche Probleme ergeben sich generell aus den verschiedenen Betriebsmodi? Welche Vor- beziehungsweise Nachteile sehen Sie bei den zur Verfügung stehenden Mechanismen?*
Sichern Sie die Implementierung dieser Teilaufgabe für die Abgabe!

C. *Betriebswechsel:* Für die verschiedenen Betriebsmodi sind unterschiedliche Teilmengen von Aufgaben notwendig. Es bietet sich also an, einen Betriebswechsel durchzuführen um das System optimal auf seine jeweilige Aufgabe auszurichten. Implementieren Sie die Rekonfiguration des Aufgabensystems mit Hilfe der zusätzlichen aperiodischen Aufgabe T_8 („Moduswechsel"). Suspendieren Sie alle unnötigen Aufgaben und stoppen Sie gegebenenfalls die aktivierenden Alarme.
Wo sehen Sie die Vorteile eines expliziten Betriebswechsels? Wo liegen die Herausforderungen?

Hinweise

- Erforderliche Dateien: wie vorgegeben
- Bearbeitung: Gruppe mit je zwei/drei Teilnehmern.
- Abgabezeit: 17.01.2013
- 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.
- 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.