AUFGABE 6: 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 Signal	250	?
T_5	Darstellung LDS	250	?

T_1 Tastet den ADC 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 abgetastete Signal mithilfe der vorgegebenen Funktion `ezs_plot()` auf dem Framebuffer dar.

T_5 Stellt die Ergebniss der LDS-Analyse durch Aufruf der ebenfalls gegebenen Funktionen `ezs_plot_pds()` auf dem Framebuffer dar.

Für den Datenaustausch zwischen T_1, T_3, T_4 und T_5 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

1. *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 sowohl T_1 und T_2 als auch T_4 und T_5 sich nicht überlappen. Sie müssen nicht den vollen

Ablaufplan aufstellen – dieser wäre sehr groß. Messen Sie die Laufzeit von T_3, T_4 und T_5 und nehmen Sie das Maximum als Annäherung für die WCET.

2. *Implementierung der aperiodischen 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, die durch folgendes Aufgabensystem bewältigt werden soll:

	Bezeichnung	**Min. Zwischenankunftszeit** ms	**WCET** ms
T_6	Byteweiser Empfang	?	–
T_7	Dekodierung	?	0,5
T_8	Zustandsverwaltung	?	–

T_6 baut aus den empfangenen Zeichen eine Zeichenkette auf. Hierfür wird die Funktion `packet_receive()` genutzt.

T_7 untersucht die vollständige Zeichenkette und extrahiert mittels `decode_command()` gültige Befehle.

T_8 wertet die Befehle aus und verwaltet eine Zustandsmaschine für das System.

Der Datenaustausch zwischen der ISR, T_6 und T_7 erfolgt wieder über globale Puffer. Zeichen dürfen in dieser Lösung also verloren gehen! Implementieren Sie die komplette Funktionalität von T_6 und T_7 in den beiden angegebenen Funktionen, also nicht direkt in Fäden/ISR/DSR, dies erleichtert Ihnen spätere Teilaufgaben.

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_6 zugänglich. Für die Umsetzung der Tastatur-Scancodes in lesbare ASCII-Zeichen steht Ihnen die Funktion `ezs_get_char_from_keycode()` zur Verfügung. Die DSR nutzt die von Ihnen zu implementierende Funktion `int packet_receive(char byte)` um einzelne Zeichen bis zum nächsten Zeilenvorschubzeichen `'\n'` zu puffern.

☞ ezs_keycodes.h

Halten Sie deren 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. ☞ `man 3 strncmp`
Vergessen Sie bei allen implementierten Aufgaben die WCET-Simulation nicht.
Vor der weiteren Umsetzung der Steuerung müssen Sie nun zunächst die minimale Zwischenankunftszeit der aperiodischen Aufgaben T_6 bestimmen. Ermitteln Sie diese durch Zeitmessung. *Wie können Sie eine minimalen Zwischenankunftszeit im Falle von T_6 erzwingen? Welche minimale Zwischenankunftszeit ergibt sich daraus für T_7? Wie lässt sie sich deutlich vergrößern?*

3. *Abbildung der aperiodischen 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_7. *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_7 an geeigneter Stelle aufzurufen. Für zwei der drei Konzepte 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.*

4. *Umschaltung der Anzeige:* Verwenden Sie nun T_8 für die Implementierung einer Zustandsmaschine. Diese prüft einkommende Befehle und passt eine globale Zustandsvariable an. Stellen Sie hierfür Teilaufgabe T_7 auf Hintergrundbetrieb ein. Auch für diese Zustandsvariable bietet es sich an ein `Enum` Da T_8 nur bei Empfang eines gültigen Kommandos ausgeführt werden soll, eignet sich das *eCos-Event-Konzept* für das Aufwecken dieses Fadens und die Weitergabe des empfangenen Befehls. *Welche Variante zur Überprüfung der Events muss hier verwendet werden? Welche Betriebszustände können auftreten und müssen von T_8 abgebildet werden?*

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

Wie müssen die weiterhin periodischen Aufgaben T_1 bis T_5 in Abhängigkeit vom Betriebszustand verhalten?

5. *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 und somit Ressourcen zu sparen. Nutzen Sie T_8 für einen Betriebswechsel und suspendieren Sie, bei einem Zustandswechsel, alle unnötigen Aufgaben und stoppen Sie gegebenenfalls die aktivierenden Alarme. **Sichern Sie Ihre Implementierung für die spätere Abgabe! Falls Sie die Fäden T_1 bis T_5 für diese Angabe angepasst haben, erhalten Sie die alte Implemtierung bitte ebenfalls für die spätere Abgabe, beispielsweise in Kommentaren.** *Wo sehen Sie die Vorteile eines expliziten Betriebswechsels? Wo liegen die Herausforderungen?*

Erweiterte Aufgabe

1. *Implementierung der Signal-Trigger-Funktion:* Verwenden Sie für die erweiterte Aufgabe folgende angepasste Version des Task-Systems:

	Bezeichnung	**Periode** ms	**WCET** ms
T_1	Abtastung Signal	4	0,5
T_2	Signal Trigger	4	0,5
T_3	Analyse LDS	1000	?
T_4	Darstellung Signal	250	?
T_5	Darstellung LDS	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 vom ADC 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. Die Darstellung im Trigger-Betrieb unterscheidet sich von der Bisherigen und erfolgt aperiodisch:

	Bezeichnung	**Min. Zwischenankunftszeit** ms	**WCET** ms
T_9	Darstellung Trigger	?	-

Implementieren Sie diese Aufgabe T_9, welche die bis zum Trigger-Ereignis aufgezeichneten Werte von T_1 mittels `ezs_plot()` darstellt. Nutzen Sie Events um die

Aufgaben T_1, T_2 und T_9 geeignet zu koordinieren. Beachten Sie hierbei, dass T_1 in jedem Fall weiter Daten aufzeichnen muss und somit T_9 nicht auf denselben Daten arbeiten kann. Nutzen Sie den *Mailbox-Mechanismus* von eCos um dieses Problem zu lösen.

2. *Vollständige Steuerung:* Vervollständigen Sie nun die Steuerung der Oszilloskopfunktionen. Aus den möglichen Optionen ergeben sich zusätzliche Betriebsmodi: *Welche?* Bei aktivierter Trigger-Funktion erfolgt die Anzeige (Zeitsignal) aperiodisch, sonst periodisch (Zeitsignal oder LDS). *Welche Vor- beziehungsweise Nachteile sehen Sie bei den zur Verfügung stehenden Mechanismen zur Bereitstellung von Ereignissen unter eCos? Was muss bei ihrer Verwendung beachtet werden?*

3. *Ereignissteuerung: Welche der bisher periodischen Aufgaben bieten sich noch für die Aktivierung durch Ereignisse an?* Bauen Sie die in Frage kommenden Aufgaben um und minimieren Sie damit den Einsatz von Alarmen, die Sie in T_8 aktivieren bzw. suspendieren müssen.

Hinweise

- Bearbeitung: Gruppe mit je zwei/drei Teilnehmern.
- Abgabezeit: 12.01.2015
- Fragen bitte an `i4ezs@lists.cs.fau.de`