AUFGABE 6: EXTENDED SCOPE

Hinweis: *Auf studentischen Wunsch wird diese Aufgabe vorzeitig, vor der zugehörigen Tafelübung veröffentlicht. Diese findet entsprechend am 17.12. bzw. 18.12.2019 statt.*

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 Rate Monotonic Algorithm (RMA).

	Bezeichnung	Periode ms	WCET ms
T_1	Abtastung Signal	4	0,5
T_2	Flankenerkennung	10	1,0
T_3	Analyse PDS	1000	?
T_4	Darstellung Signal	250	?
T_5	Darstellung PDS	1000	?

Die Deadline entspricht jeweils der Periode.

T_1 Tastet den ADC ab (`ezs_adc_get`) und fügt die Werte in einen Ringpuffer der Größe 64 (`TIME_DOMAIN_LENGTH`) ein.

T_2 Simuliert in der einfachen Übung nur Laufzeit.

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

T_4 Stellt das abgetastete Signal mithilfe der vorgegebenen Funktion `ezs_plot()` auf dem Framebuffer dar.

T_5 Stellt die Ergebnis der PDS-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 32 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 einige grundlegende globale Puffer und Daten sind von uns bereits angelegt worden. Die

☞ PDS_LENGTH

☞ make doc

Vorgabe soll Ihnen die Aufgabe erleichtern, Sie müssen sich aber nicht zwingend daran halten. Globale Definitionen finden sich zur besseren Lesbarkeit in der Datei `config.h`, welche Sie ebenfalls im Projektverzeichnis vorfinden.

Übernehmen Sie bei Bedarf – wie in den vorherigen Übungen – Ihre Implementierung der Hilfsfunktionen `ms_to_cyg_ticks()`, `ms_to_ezs_ticks()` und `ezs_simulate_wcet()` in die Vorgabe. Mittels `make sanity-test` haben Sie wieder die Möglichkeit, Ihre Implementierung grundlegend zu prüfen.

1 Aufgabenstellung

Nach dem erfolgreichen Aufsetzen des `build`*-Verzeichnisses können Sie sich mittels* `make doc` *eine Übersicht über alle in* `libEZS` *bereitgestellten Funktionen inklusive deren Dokumentation erzeugen.*

Denken Sie daran, Ihren Quellcode nach vollständiger Bearbeitung noch vor dem Beginn der Rechnerübung abzugeben. Rufen Sie hierzu in Ihrem `build`*-Verzeichnis* `make submit` *auf.*

1.1 Laufzeitmessung:

Aufgabe 1

Ergänzen Sie die Vorgabe so, dass die Aufgaben ihre Funktion auch erfüllen (das PDS muss gezeichnet werden) und simulieren Sie die WCET soweit oben angegeben.

Aufgabe 2

Messen Sie die Laufzeit von T_3, T_4 und T_5 (sinnvoll!) und nehmen Sie das Maximum als Annäherung für die WCET.

$$e_3 \approx \_\_\_\_\_$$

$$e_4 \approx \_\_\_\_\_$$

$$e_5 \approx \_\_\_\_\_$$

Antwort:

1.2 Implementierung der aperiodischen Steuerung:

In dieser Teilaufgabe sollen Sie das Oszilloskop um eine externe Steuerung über die serielle Verbindung 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

Aufgabe 3

Bestimmen Sie auch für diese neuen Aufgaben die WCET durch eine Annäherung mittels Laufzeitmessung.

Aufgabe 4

☞ ezs_serial_getc()

Lesen Sie in der ISR jedes Zeichen einzeln aus und machen Sie dieses der zugehörigen DSR bzw. T_6 zugänglich. Sie können dazu auf ihre Lösung aus der Aufgabe "Antwortzeit" zurückgreifen. Die DSR bzw. T_6 nutzt die von Ihnen zu implementierende Funktion `enum CommandStatus packet_receive(char byte)` um einzelne Zeichen bis zum nächsten Zeilenvorschubzeichen `'\n'` zu puffern. Halten Sie die Implementierung der Funktion so kurz wie möglich. Achten Sie dabei auf speicherkorrekte Implementierung und die Repräsentation von Zeichenketten (engl. *strings*) in C.

Die Funktion kehrt bei erfolgreichem Paketempfang mit `CommandComplete` zurück und setzt den Puffer zurück. Ist noch kein Zeilenvorschubzeichen empfangen worden, so kehrt sie mit `CommandIncomplete` zurück. Achten Sie drauf, dass Ihr `cutecom` (oder das Terminalprogramm für serielle Schnittstellen Ihrer Wahl) ausgehende Befehle mit genau einen Zeilenvorschubzeichen terminiert (cutecom Einstellung: "LF line end"). Zudem muss wahrscheinlich eine Zeichenverzögerung ("char delay") von einigen Millisekunden konfiguriert werden, um die Befehle korrekt zu empfangen.

Antwort:

Aufgabe 5

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 zwei möglichen Befehlskombinationen als Bitmaske kodiert.

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

Die Funktion wird erst in der nachfolgenden Teilaufgabe tatsächlich aufgerufen. Verwenden Sie die Funktion `strncmp()`, um Zeichenketten zu vergleichen. Wieso ist es grundsätzlich sinnvoll `strncmp()` anstatt `strcmp()` zu verwenden? Welche Größe muss `strncmp()` als Länge übergeben werden? Wieso?

☞ man 3 strncmp

Antwort:

Aufgabe 6

Vor der weiteren Umsetzung der Steuerung müssen Sie nun zunächst die minimale Zwischenankunftszeit der aperiodischen Aufgaben T_6 bestimmen. Wie können sie diese ermitteln? Welchen Wert für die minimale Zwischenankunftszeit erhalten Sie? *Hinweis:* Es existieren verschiedene Ansätze um diese Parameter zu ermitteln. Wovon hängt Ihre Entscheidung ab?

Antwort:

Aufgabe 7

Welche minimale Zwischenankunftszeit ergibt sich daraus für T_7? Haben Sie als Entwickler Möglichkeiten diesen Entwurfsparameter zu vergrößern?

Antwort:

1.3 Abbildung der aperiodischen Steuerung:

Aufgabe 8

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.

Unterbrecherbetrieb: _____

Hintergrundbetrieb: _____

Periodischer Zusteller: _____

Falls Sie den Task T_7 aufgrund des Aufrufkontextes nicht ausführen können, so genügt es die Funktion `decode_command()` als Implementierung von T_7 an geeigneter Stelle direkt aufzurufen. Für zwei der drei Konzepte benötigen Sie jetzt einen eigenständigen Faden. Beachten Sie die ermittelte Zwischenankunftszeit und nutzen Sie das Prioritätsgefüge aus. **Achten Sie darauf, dass Ihnen der Code der jeweiligen Variante erhalten bleibt.**
Es bietet sich hierfür an, für die verschiedenen Varianten entsprechende Konfigurationsoptionen in `config.h` anzulegen, um das gewünschte Verhalten zum Übersetzungszeitpunkt festlegen zu können.

Aufgabe 9

Welches Problem ergibt sich aus der Nutzung der globalen Puffer zum Datenaustausch? Bedenken Sie auch logische Abhängigkeiten zwischen den beteiligten Aufgaben. Wie könnte man dieses Problem vermeiden (Implementierung nicht erforderlich)?

Antwort:

1.4 Umschaltung der Anzeige:

Aufgabe 10

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` zu verwenden. 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. *Nützliche*

Fragestellungen während der Implementierung: Welche Variante zur Überprüfung der Events muss hier verwendet werden? Welche Betriebszustände können auftreten und müssen von T_8 abgebildet werden? Wie müssen die weiterhin periodischen Aufgaben T_1 bis T_5 in Abhängigkeit vom Betriebszustand verhalten?
Hinweis: Sicher Sie diese Lösung für die spätere Abgabe als `app_no_modeswitch.c`!

1.5 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.

Aufgabe 11

Nutzen Sie T_8 für einen Betriebswechsel und stoppen Sie die aktivierenden Alarme aller unnötigen Aufgaben bei einem Zustandswechsel.

Aufgabe 12

Wo sehen Sie die Vorteile eines expliziten Betriebswechsels? Wo liegen die Herausforderungen?

Antwort:

2 Erweiterte Aufgabe

Die Erweiterten Übungsaufgaben sind nur für Teilnehmer verpflichtend, die das 7,5-ECTS-Modul belegen. Wir werden Sie natürlich auch dann bei der Bearbeitung unterstützen, wenn Sie diese Teilaufgaben freiwillig bearbeiten.

Passen Sie Ihr Aufgabensystem an die folgende leicht veränderte Version an:

	Bezeichnung	Periode ms	WCET ms
T_1	Abtastung Signal	4	0,5
T_2	Flankenerkennung	4	0,5
T_3	Analyse PDS	1000	?
T_4	Darstellung Signal	250	?
T_5	Darstellung PDS	1000	?

Übernehmen Sie die aperiodischen Aufgaben $T_6 - T_8$ direkt aus Ihrer bisherigen Implementierung.

2.1 Implementierung der Triggerfunktionalität:

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 ist es, Aufgabe T_2 um eine entsprechende Flankenerkennung für das vom ADC eingelesene Signal zu ergänzen.

Aufgabe 13

Hinweis: Sicher Sie zunächst ihre bisherige Lösung für die spätere Abgabe als `app_basic_ex.c`!

Implementieren Sie die Flankenerkennung in T_2 so, dass sie bei einer fallenden oder steigenden Flanke ein Trigger-Ereignis erkennt. Anstatt eine einstellbare Pegelhöhe zu implementieren, können Sie davon ausgehen, dass eine steigende Flanke vorliegt, wenn der aktuelle Wert des Signals größer als 188 (`TRIGGER_LEVEL`) und der vorherige Wert kleiner als 188 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	?	-

Aufgabe 14

Implementieren Sie diese Aufgabe T_9, welche die bis zum Trigger-Ereignis von T_1 aufgezeichneten Werte in zeitlich korrekter Reihenfolge mittels `ezs_plot()` darstellt. Nutzen Sie *Events* um die Aufgaben T_1 und T_2 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. Ist ein periodisches Aufwecken der Aufgabe T_2 über einen Alarm notwendig?

Antwort:

2.2 Steuerung der Funktionalität:

Um die Nutzung der Triggerfunktionalität konfigurierbar zu machen, soll die Oszilloskopsteuerung um folgende Kommandos erweitert werden:

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

Aufgabe 15

Vervollständigen Sie nun die Steuerung der Oszilloskopfunktionen. Aus den neuen Kommandos (`trigger`, `tlevel`) ergeben sich zusätzliche Betriebsmodi – welche?

Antwort:

Bei aktivierter Trigger-Funktion erfolgt die Anzeige (Zeitsignal) aperiodisch, sonst periodisch (Zeitsignal oder PDS).

2.3 Mögliche Entwurfsalternativen:

Aufgabe 16

Das von uns vorgeschlagene Aufgabensystem und die implizite/explizite Umsetzung der enthaltenen Abhängigkeiten stellen nur eine Entwurfsmöglichkeit dar. Entwerfen Sie eine weitere Variante des kompletten Aufgabensystems und versuchen Sie hierbei möglichst alle Abhängigkeiten, die bisher implizit koordiniert wurden, auf eine andere Art und Weise umzusetzen. Welche Aufgaben müssen zwingend

von Alarmen aktiviert werden? Welche lassen sich durch logische Abhängigkeiten realisieren?

Antwort:

Aufgabe 17

Bauen Sie Ihre Implementierung gemäß des zuvor gewählten Entwurfsmusters um.

Antwort:

Hinweise

- Bearbeitung: Gruppe mit je drei Teilnehmern.
- Abgabefrist: 23.01.2020 (vor der ersten Rechnerübung) ☞ `make submit`
- Fragen bitte an `i4ezs@lists.cs.fau.de`