AUFGABE 1: HALLO WELT

Diese Aufgabe dient dem initialen Einrichten Ihrer Entwicklungsumgebung und dem ersten Kontakt mit dem Echtzeitbetriebssystem eCos und dem STM32F429-Board. Ziel ist es einen ersten Einblick in die Möglichkeiten der Entwicklungsumgebung zu erhalten. Im Laufe der Übungen werden Sie mit vielen verschiedenen Themenbereichen in Berührung kommen: Programmierung in C, Entwicklung auf eingebetteten Systemen, Interaktion mit der Außenwelt und zeitliche Aspekte von Echtzeitsystemen. Dabei stellt die Programmierung ein notwendiges Mittel zum Zweck dar und wird als Vorkenntnis vorausgesetzt. Gegen Ende der Veranstaltung werden die zeitlichen Aspekte zunehmend mehr betont. In diesem Aufgabenblatt werden alle genannten Aspekte kurz thematisiert.

Hinweis: Zeitliche Aspekte von Ein-/Ausgabevorgängen sind Bestandteil einer folgenden Tafelübung.

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 Vorbereitung:

Aufgabe 1

Kopieren und entpacken Sie die Vorgabe in ein beliebiges Arbeitsverzeichnis und setzen Sie die nötigen Umgebungsvariablen durch den Aufruf von **`source`** `ecosenv.sh`. Nun können Sie die Makefiles generieren und die noch funktionslose Anwendung erstmals kompilieren. Sie können die erzeugte Anwendung mit Hilfe des Debuggers in den Flash-Speicher des Boards laden und starten.

Hinweis: Zum Starten des Programms muss der schwarze Reset-Taster gedrückt werden!

☞ **source** ecosenv.sh
☞ **cd** build
☞ cmake ..
☞ make
☞ make flash

1.2 Fadensystem

Die Erzeugung eines geeigneten Threadsystems ist Bestandteil der folgenden Aufgabe.

Aufgabe 2

Implementieren Sie in `hello.c` die `cyg_user_start()`-Funktion, in der Sie mit Hilfe von `cyg_thread_create()` einen Thread (`test_thread()`) erzeugen. Initial erzeugt dieser die einmalige Ausgabe „`Hallo Welt!\n`“ auf der seriellen Schnittstelle. Sobald Sie das Board mit dem Mini-USB-Kabel an Ihrem PC angeschlossen haben, können Sie sich die serielle Ausgabe mit Hilfe von `cutecom` anzeigen lassen (`/tmp/$(whoami)-ezs-serial` oder auch `/dev/ttyACM1`, 115200 Baud, 8 Datenbits, ein Stopbit, keine Parität). Um serielle Schnittstellen außerhalb `/dev` nutzen zu können, sollten Sie dabei nicht die auf dem System installierte cutecom Version nutzen, sondern `/proj/i4ezs/tools/cutecom`.

1.3 Periodische Abarbeitung

Aufgabe 3

Die Testthreadfunktion soll nun periodisch, jede Sekunde, mittels `ezs_printf()` die Zeichenkette „`Hallo Welt!\n`“ auf der seriellen Schnittstelle ausgeben. Um einen periodischen Thread zu simulieren, verwenden Sie folgende Funktion:

```
ezs_delay_us()
```

Dieser Funktion können ganzzahlige Werte übergeben werden, welche als Mikrosekunden interpretiert werden und eine entsprechende Verzögerung der Programmausführung bewirken.

Aufgabe 4

Sehen Sie Nachteile, periodische Ausführung auf diese Art umzusetzen? Haben Sie eine Idee für einen sinnvolleren Ansatz? Beantworten Sie die Frage im Laufe dieses Übungsblatts.

1.4 Programmierung & Programmierumgebung

Aufgabe 5

Colorcycling ist ein einfacher Ansatz, um Animationen auf leistungsschwachen, speicherbeschränkten Systemen zu ermöglichen. Anstatt jeden Frame einer Animation abzuspeichern, wird nur eine Version des Bildes gespeichert, allerdings wird jedem Pixel nicht direkt eine Farbe, sondern ein Farbschlüssel zugewiesen. Für jeden Frame der Animation wird dann eine Farbpalette erzeugt, die eine Zuordnung von Farbschlüsseln zu Farben darstellt. Dadurch können Bilder durch einfaches Iterieren der Farbpalette animiert werden[1].

Um Ihre Kenntnisse in eingebetter C-Programmierung bezüglich Zeiger, Strukturen und Bitoperationen aufzufrischen, sollen Sie eine einfache Variante von Colorcycling implementieren. In der Funktion `ezs_plot_animation()` soll hier eine einfache Grafik (`logo`) über den boardeigenen LCD-Bildschirm (`ezs_lcd_set_pixel()`) ausgegeben werden. Die Paletten der einzelnen Schritte werden dabei in einer verketteten Liste gespeichert (`image.generate_palettes()`). Der Palettenwechsel zwischen Animationsschritten erfolgt dabei über wiederholtes Iterieren dieser Liste. Machen Sie sich hierzu mit der in `colorcycling.h` vorgegebenen Schnittstelle vertraut und imlementieren Sie die vorgegebenen Funktionsstümpfe der Funktionen `ezs_plot_animation()`, `color_key_to_color()` und `color_key_for_pixel()` in `hello.c`. Ergänzen Sie nun die Hauptschleife ihres periodischen Fadens um die Abarbeitung eines einzelnen Animationsschrittes, um so eine fortlaufende Animation zu erzeugen. Ergebnis sollte ein sich im Uhrzeigersinn drehender, roter Pfeil sein.

☞ make doc

1.5 Signalerzeugung und zeitliche Eigenschaften

Deaktivieren sie nun vorübergehend sowohl Animation als auch die periodische serielle Ausgabe (Auskommentieren der Funktionsaufrufe in der Hauptschleife).

Mittels des periodischen Threads kann weiterhin ein periodisches elektisches Signal erzeugt werden. Sie können einen Timer des STM32F429-Boards mit Hilfe der Funktion `ezs_dac_write()` als Digital-Analog-Wandler verwenden (Wertebereich 0 bis $2^8 - 1$) und sich das erzeugte Signal am Oszilloskop anschauen. Verbinden Sie hierzu die Masseklemme des Tastkopfes mit dem Massepin der Filterplatine. Das Signal liegt an **Pin PD13** am EZS-Board an, der Masse-Pin (**GND**) ist am Platinenen-

[1] Eine anschauliche Erklärung finden Sie unter: `https://blog.prototypr.io/color-cycling-in-pixel-art-c8f20e61b4c4`

de angebracht (siehe Beschriftungen). Sie können mit Hilfe der Klemmprüfspitze das ausgegebene ungefilterte und gefilterte Signal an den entsprechenden Pins ansehen. Da die Ausgabe mittels Pulsweitenmodulation erfolgt, empfiehlt sich für diese Aufgabe die Betrachtung der tiefpassgefilterten Signale.

Aufgabe 6

Erzeugen Sie in der Funktion `sine_wave()` ein Sinussignal und verwenden Sie für diesen Zweck die Funktion `sinf()`. Verfolgen Sie bei Ihrer Implementierung einen systematischen Ansatz bei der Erzeugung des Signals und sorgen Sie dafür, dass die relevanten Parameter der Sinusschwingung flexibel einstellbar sind. Experimentieren Sie mit verschiedenen Variationen der Parameter. Die Periode der Ausgabe des Sinus soll dabei unabhängig von der Periode des Sinus selbst sein, d.h. der `delay`-Parameter und die Frequenz sollen unabhängig voneinander frei wählbar sein. Sie haben die Möglichkeit, die Ergebnisse Ihrer Implementierung automatisiert gegen einen Satz Referenzwerte zu vergleichen.

☞ `math.h`

☞ `make sanity-test`

Hinweis: Achten Sie hier darauf, dass Sie (hinter dem Tiefpassfilter) ein sauberes, klar erkennbares Sinussignal erzeugen.

Aufgabe 7

Verringern Sie nun die Periode des Fadens drastisch (z. B. 2 ms). Was beobachten Sie? Wie müssen Sie Abtastrate und Parameter des Sinus wählen um ein Signal einer bestimmten Frequenz zu erzeugen?

Antwort:

Aufgabe 8

Betrachten Sie nun die Fragestellung nach der Periodizität der Ausgaben aus Aufgabe 4 erneut. Bestimmen Sie dazu Periode bzw. Frequenz des Signals mit dem Oszilloskop. Aktivieren Sie nun die periodische Ausgabe (Aufgabe 3) und Animation (Aufgabe 5) und bestimmen Sie die Kenngrößen erneut.

Welche Effekte beobachten Sie durch diese Art der Umsetzung periodischer Ausführung? Haben Sie eine Idee für einen sinnvolleren Ansatz (Implementierung ist *nicht* erforderlich)?

Antwort:

1.6 Debugging & Zeitliche Aspekte der Ein-/Ausgabe

Aufgabe 9

Für das Echtzeitverhalten ist neben der eigenen Implementierung immer das Gesamtsystem zu betrachten. Um ein Gefühl hierfür zu bekommen, setzen Sie im Debugger einen Breakpoint auf die Funktion

☞ `make debug`
☞ break

```
stm32_serial_putc_polled()
```

und setzen Sie die Ausführung fort. Sobald die Ausführung an dem gesetzten Breakpoint gestoppt hat, können Sie die Aufrufhierarchie im Backtrace-Fenster betrachten. Erkunden Sie den Aufrufgraph der Funktion `stm32_serial_putc_polled()`. Wieso ist diese Funktion für Ihr Programm relevant?

☞ backtrace

Antwort:

Aufgabe 10

Lassen Sie sich den Quellcode von `stm32_serial_putc_polled()` im Debugger anzeigen. Welchen Zweck erfüllt die Funktion? Wie erbringt sie diese Leistung (abstrakt)? Welche Schritte werden hier im Einzelnen durchgeführt?

☞ list

Antwort:

Aufgabe 11

Welche aus Echtzeitsicht kritischen Stellen fallen in der Funktion besonders auf? Die Verwendung welcher Art von Funktionen ist also aus Sicht eines Echtzeitsystems im Allgemeinen problematisch? Welches zusätzliche Wissen ist hier für den Einsatz erforderlich, wie kann dieses Wissen gewonnen werden?

Aufgabe 12

Betrachten Sie nun die Funktionen `ezs_dac_write()` und `timer_set_oc_value()` im Debugger. Wie unterscheiden sie sich von `stm32_serial_putc_polled()` im Bezug auf das Echtzeitverhalten?

1.7 Abgabe

Aufgabe 13

Geben Sie nun Ihre Lösung durch den Befehl `make submit` ab. Sie können Ihre Lösung vor dem Abgabetermin (d.h. vor der ersten Rechnerübung am 06.05.2024) jederzeit aktualisieren. Mittels `make diff` können Sie überprüfen, ob lokale Änderungen vorliegen.

Hinweise

- Bearbeitung: Gruppe mit je drei Teilnehmern.
- Abgabefrist: 06.05.2024 (23:59) ☞ `make submit`
- Fragen bitte an `i4ezs@lists.cs.fau.de`