AUFGABE 2: HALLO DIY

Diese Aufgabe dient dem initialen Einrichten Ihrer Entwicklungsumgebung, dem ersten Kontakt mit dem Echtzeitbetriebssystem eCos und dem STM32F411-Board. Bei der Aufgabe gibt es folgende *Lernziele*:

- Erzeugen von Fäden in eCos
- Debuggen mittels GDB
- Umgang mit Realzeit auf einem Mikrocontroller
- Digital-Analog-Umwandlung mittels Pulsdauermodulation (PWM)[1] und Rekonstruktion analoger Signale

Hinweis: Sie können diese Lernziele auch anhand Ihres Gruppenprojekts demonstrieren.

1 Aufgabenstellung

1.1 Vorbereitung:

1. Aufgabe Die Vorgabe befindet sich im Ordner `01-HalloDIY` des Vorgaben-Repositories:

`git@gitlab.cs.fau.de:diy/diy-vorgabe-ss19.git`

Klonen Sie die Vorgabe 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 Make-Targets in den Flash-Speicher des Boards laden und starten (siehe Anweisungen rechts).

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

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

Sobald Sie das Board mit dem Mini-USB-Kabel an Ihrem PC angeschlossen haben, können Sie sich Ausgaben auf der seriellen Schnittstelle mit Hilfe von `cutecom` anzeigen lassen.

☞ /dev/ttyACM1
☞ 115200 Baud
☞ 8 Datenbits
☞ ein Stopbit
☞ keine Parität
☞ kein Handshake

1.2 Fadensystem:

Die Erzeugung eines geeigneten Threadsystems ist Bestandteil der folgenden Aufgaben.

2. Aufgabe Am Ende dieser Teilaufgabe soll periodisch, einmal jede Sekunde die Zeichenkette „`Hallo Welt!\n`“ auf der seriellen Schnittstelle ausgegeben werden.
Erzeugen Sie dazu in der Funktion `cyg_user_start()` in `hello.c` mit Hilfe von

`cyg_thread_create()`

[1] `https://de.wikipedia.org/wiki/Pulsdauermodulation`

einen neuen Thread und verwenden Sie idealerweise `hello_thread_entry()` als Einsprungfunktion. Dieser Thread soll sich um die Ausgabe der Zeichenkette kümmern.
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.

Hinweis: Eine ausführliche Dokumentation zu eCos finden Sie in der Befehlsreferenz[2]. Die Dokumentation für die `libEZS` *können Sie sich selbst erzeugen.*

☞ `make doc`

3. Aufgabe *Sehen Sie Nachteile, periodische Ausführung auf diese Art umzusetzen? Haben Sie eine Idee für einen sinnvolleren Ansatz?*

Antwort:

1.3 Debugging mittels GDB:

4. Aufgabe 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` (für Dashboard), oder `make gdb`

```
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. Lassen Sie sich den Quellcode von der Funktion im Debugger anzeigen.
Welchen Zweck erfüllt die Funktion?

Antwort:

[2] `http://ecos.sourceware.org/docs-latest/ref/kernel.html`

1.4 Signalerzeugung:

Mittels des periodischen Threads können nun Abtastwerte eines Signals berechnet werden, aus denen sich ein analoges Signal rekonstruieren lässt. Die Funktion `ezs_dac_write()` (Wertebereich 0 bis $2^8 - 1$) stellt eine Implementierung eines einfachen Digital-Analog-Wandlers zur Verfügung, die einen Timer des STM32F411-Prozessors benutzt, um ein PWM Signal zu erzeugen. Das Signal liegt an **Pin PD15** an, der Masse-Pin ist direkt daneben (siehe Beschriftung). Wir stellen eine Platine mit einem einfachen Rekonstruktionsfilter zur Verfügung, das eingangsseitig an diese beiden Pins angeschlossen wird.

Hinweis: Verbinden Sie den Masse-Pin des EZS-Boards mit dem eingangsseitigen Massepin der Filterplatine und die Masseklemme des Tastkopfes des Oszilloskops mit dem ausgangsseitigen.

5. Aufgabe Erzeugen Sie mit Hilfe der Funktion `sinf()` periodisch Abtastwerte eines Sinussignals mit der Frequenz 7 Hz und übergeben Sie diese an `ezs_dac_write()`. ☞ `math.h`
Sorgen Sie dafür, dass die relevanten Parameter des Signals nachvollziehbar und übersichtlich einstellbar sind. Insbesondere soll die Abtastfrequenz unabhängig von der Periode des Sinus einstellbar sein.

Hinweis: Implementieren Sie die Berechnung in der Funktion `signal_thread_entry()` *und übergeben Sie diese statt* `hello_thread_entry()` *als Einsprungfunktion an* `cyg_thread_create()`.

Welche Parameter sind bei einer sinusförmigen Schwingung von Interesse?[3]
Wie müssen Sie die Abtastrate und die Parameter des Sinus wählen um ein Signal einer bestimmten Frequenz zu erzeugen?

Antwort:

6. Aufgabe Verbinden Sie das EZS-Board wie oben beschrieben mit der Filterplatine.
Greifen Sie nun mit Hilfe der Klemmprüfspitze nacheinander das ungefilterte und das gefilterte Signal an den entsprechenden Pins ab und betrachten Sie es am Oszilloskop.
Wie unterscheidet sich das gefilterte vom ungefilterten Signal?
Welche Abtastrate(n) ergibt/ergeben sich bei diesem System?
Was ist die theoretische obere Grenzfrequenz bis zu der in diesem System analoge Signale (mit nicht-periodischem Spektrum) rekonstruiert werden können?

[3] `https://de.wikipedia.org/wiki/Schwingung#Harmonische_Schwingung`

Antwort:

7. Aufgabe Verringern Sie nun die Periode des Fadens drastisch, um die resultierende Abtastfrequenz zu erhöhen (z. B. 500 Hz).
Was beobachten Sie?
Wie entsteht die Abweichung?

Antwort:

Hinweise

- Bearbeitung: in Gruppen
- Abgabefrist: bis Ende nächster Woche
- Fragen bitte an `diy-orga@fablab.fau.de`