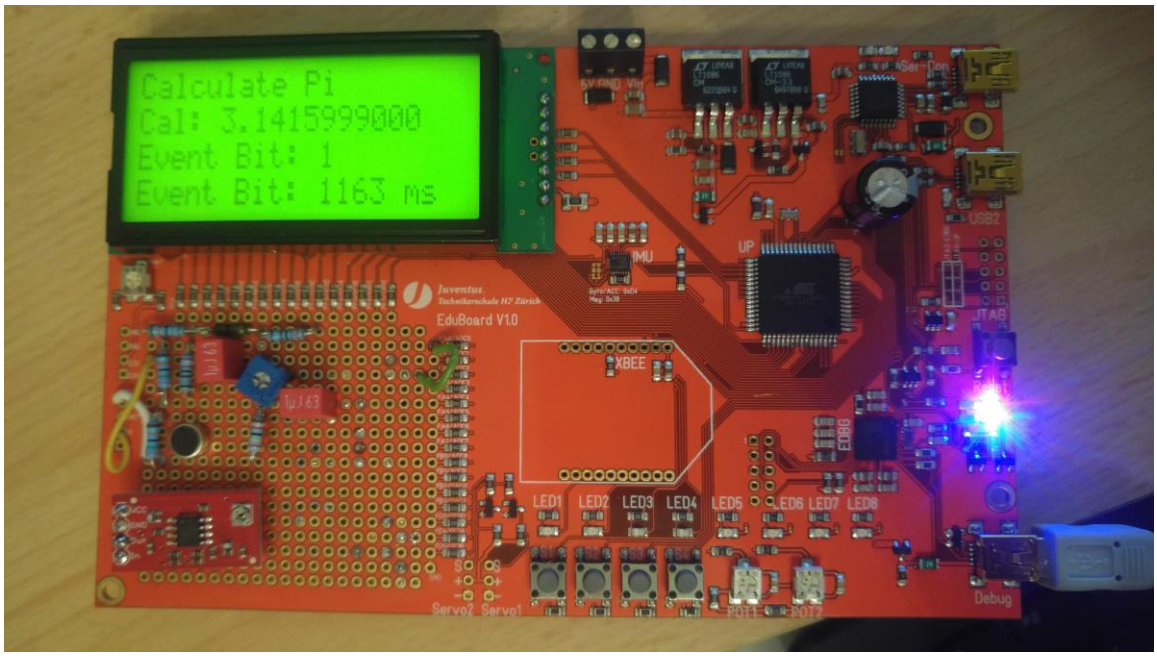

Calculate π

JUVENTUS SCHULE

| Embedded Systems | 15.04.2019

Inhalt

Algorithmus:

Um Pi zu berechnen gibt es mehrere Wege respektive Formeln die man anwenden kann. Ich habe mich für die folgende Formel entschieden:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

Durch die mathematische „Folgen und Reihen“ Technik kann man so Pi immer genauer berechnen. Und zwar wird bei dieser Formel der Nenner immer um +2 addiert und der Bruchterm abwechslungsweise addiert und subtrahiert. Im C-Code sieht das folgendermassen aus:

```
for(i=0;i<100;i++)
{
        dPi4 = dPi4 - 1/(3+n*4)+1/(5+n*4);          //Berechnung von Pi
        n++;
}
```

Es wird immer ein Additionsterm berechnet und von der Variablen dpi4 subtrahiert. Dies wird wiederum neu in der dpi4 Variable abgespeichert. Die variable n wird bei jeder Wiederholung um +1 vergrössert, damit die Reihe so wie sie in der Formel steht entstehen kann. Die „for“-Schleife kann beliebig oft durchgeführt werden. Je öfters sie durchläuft, desto genauer wird Pi berechnet.

Tasks

Zur Ausführung des Codes habe ich folgende Tasks erstellt.

```
xTaskCreate( vLedBlink, (const char *) "ledBlink", configMINIMAL_STACK_SIZE+10, NULL, 1, &ledTask);
xTaskCreate( my_calculation, (const char *) "ledBlink", configMINIMAL_STACK_SIZE+10, NULL, 1, &calculate_pi);
xTaskCreate( my_display, (const char *) "ledBlink", configMINIMAL_STACK_SIZE+10, NULL, 1, &display_task);
xTaskCreate( my_button_handler, (const char *) "ledBlink", configMINIMAL_STACK_SIZE+10, NULL, 1, &button_handler);
```

vLedBlink: Dieser Task dient zur Kontrolle, ob der Code durchläuft oder ob er irgendwo abstürzt. Falls er abstürzt hört das LED auf zu blinken.

My_calculation: In diesem Task wird die Berechnung von Pi gemacht mit der oben beschriebenen „for“-Schleife.

My_display: Dieser Task wird zum schreiben des Displays verwendet.

My_button_handler: Er ist für das Einlesen der Buttons zuständig und steuert somit die Befehle. Die Buttons haben folgende aufgabe:

Button 1: Beim Drücken dieser Taste wechselt der Berechnungszustand zwischen Run und Pause (Toggle)

Button 2: Hier wird das Zwischenresultat auf dem Display angezeigt.

Button 3: Setzt alle berechneten Daten zurück.

Eventgroup:

```
xCreatedEventGroup = xEventGroupCreate();
```

Mit Hilfe dieser EventGroup wird der Berechnungszyklus gesteuert. Es soll folgende Steuermöglichkeiten haben.

- Starten des Berechnungszyklusses (Bito wird gesetzt oder gelöscht)
- Stoppen des Berechnungszyklusses(Bit1 wird gesetzt oder gelöscht)
- Zurücksetzen (Reset) der Berechneten Werte (Bit2 wird gesetzt oder gelöscht)

Hardware Timer

Es soll die Zeit gemessen werden, wie lange der Prozessor benötigt um Pi bis auf die 5te Kommastelle zu berechnen. Dazu wird ein Hardware Timer (TCD0) eingesetzt. Der Timer startet, wenn die Berechnung gestartet wird. Sobald Pi = 3.14159 gross ist, stoppt die Zeit.

Resultate:

Nach dem der Code getestet und geprüft wurde, konnte die Messung analysiert werden. Es stellte sich heraus, dass die Berechnungszeit variert, wenn man die Wiederholungen der „for“ Schleife grösser respektive kleiner macht. Ich habe folgende Messungen gemacht

Iterationen der „for“ Schleife	Zeit (s)
1000	13.19
10000	11.75
50000	11.61
100000	11.61

Es wurden noch Optimierungsversuche gemacht:
Einen Task konnte gelöscht werden ohne dass es die Funktion stört. Delay Zeiten angepasst.

100000	11.53

Rückschluss:

Die Berechnung von Pi auf 5 Kommastellen genau benötigt ?? Berechnungsschritte. Für einen Berechnungsdurchlauf braucht er also:

$$1\ \text{Berechnunsschritt} = \frac{t_{tot}}{Iteration_{tot}} = \frac{11.53}{100000} = 11.5\text{us}$$

Für eine Berechnung benötigt der Microcontroller 11.5us. Dies ist auf die einzelne Berechnung schnell. Wenn dies jedoch auf die ganze Berechnung addiert wird, kommen 11s relative lange vor.