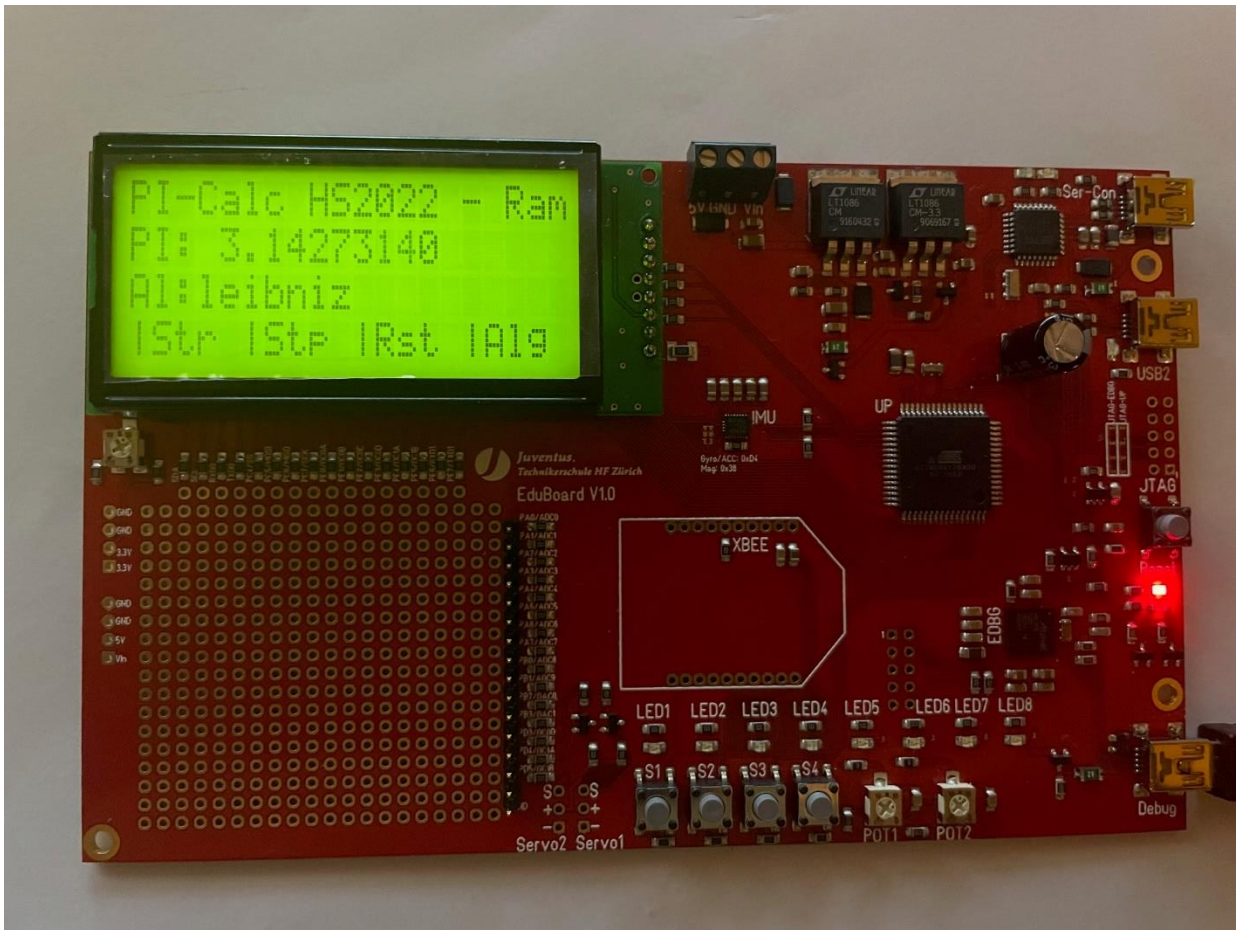
Juventus Technikerschule HF

Calculating PI



Von: Balaram Ramalingam

Fach: Embedded Systems

Betreuer: Martin Burger

TSE 2009 5.Semester

Inhaltsverzeichnis

1. Aufgabenstellung

- Realisiere die Leibniz-Reihen-Berechnung in einem Task.
- Wähle einen weiteren Algorithmus aus dem Internet.
- Realisiere den Algorithmus in einem weiteren Task.
- Schreibe einen Steuertask, der die zwei erstellten Tasks kontrolliert.

Dabei soll folgendes stets gegeben sein:

- Der aktuelle Wert soll stets gezeigt werden. Update alle 500ms
- Der Algorithmus wird mit einem Tastendruck gestartet und mit einem anderen Tastendruck gestoppt.
- Mit einer dritten Taste kann der Algorithmus zurückgesetzt werden.
- Mit der vierten Taste kann der Algorithmus umgestellt werden. (zwischen Leibniz und dem zweiten Algorithmus)
- Die Kommunikation zwischen den Tasks kann entweder mit EventBits oder über TaskNotifications stattfinden.
- Folgende Event-Bits könnte man beispielsweise verwenden:
    - EventBit zum Starten des Algorithmus
    - EventBit zum Stoppen des Algorithmus
    - EventBit zum Zurücksetzen des Algorithmus
    - EventBit für den Zustand des Kalkulationstask als Mitteilung für den Anzeige-Task

- Mindestens drei Tasks müssen existieren.
    - Interface-Task für Buttonhandling und Display-Beschreiben
    - Kalkulations-Task für Berechnung von PI mit Leibniz Reihe
    - Kalkulations-Task für Berechnung von PI mit anderer Methode

- Erweitere das Programm mit einer Zeitmess-Funktion (verwende xTaskGetTickCount) und messe die Zeit, bis PI auf 5 Stellen hinter dem Komma stimmt. (Zeit auf dem Display mitlaufen lassen und beim Erreichen der Genauigkeit die Zeit berechnen. Die Berechnung von PI soll weitergehen.)

2. Einleitung

Es gibt viele verschiedene Algorithmen, um die PI zu berechnen. Es gibt einfache sowie sehr komplizierte Methoden, diese zu berechnen. Was den Unterschied ausmacht, ist, wie schnell das eine Methode PI ausrechnet.

2.1 Leibniz-Reihen Berechnungsmethode

Eine einfache und verständnisvolle Methode ist die Leibniz-Methode.

Diese Methode ist bereits im 14. Jahrhundert dem Mathematiker bekannt. Officiel veröffentlichte Gottfried Wilhelm Leibniz im Jahr 1682 in der Zeitschrift Acta Eruditorum.

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

Je mehr man berechnet desto näher kommt man an Zahl π / 4

Aus diesem haben die Schweizer Mathematiker Nicolas Fatio und Euler die schneller konvergente Reihe erzeugt.

$$\frac{\pi}{4} = \frac{1}{2}\left(1 + \frac{1}{1*3} + \frac{1*2}{1*3*5} + \frac{1*2 \ldots * n}{1*3*5 \ldots (2n+1)} + \cdots\right.$$

$$\frac{\pi^2}{8} = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{(n+2)^2} + \cdots$$

2.2 Nilakantha-Reihen Berechnungsmethode

Kelallur Nilakantha Somayaji war ein indischer Mathematiker, Astrologe und Astronom. Nur von ihm gibt es mehrere Formular für die PI-Berechnungen, davon habe ich für meine Arbeit dieses untenstehende Formular verwendet. Ausserdem war der erste PI-Berechnung durch eine unendliche Reihe in einem Sanskrit Vers vermerkt. Diese Methode ist effizienter als Leibniz Reihe.

$$\pi = 3 + \frac{4}{3^3 - 3} - \frac{4}{5^3 - 5} + \frac{4}{7^3 - 7} - \frac{4}{9^3 - 9} + \frac{4}{(n+2)^3 - (n+2)} \cdots$$

3. Beschreibung der Tasks

Um die Berechnungen, Taster Funktion, UI-Anzeige der Texte und die Steuerung der Tasks zu handhaben, sind die folgenden 4 FreeRTOS-Tasks implementiert.

3.1 ControllerTask - Schnittstellenaufgaben

ControllerTask führt die folgenden zwei Hauptfunktionen aus

1. Steuern Sie die Ausführung der beiden Berechnungsaufgaben basierend auf den Ereignissen, die durch die vier Taster Klicks generiert werden (Start | Stopp | Zurücksetzen | Algorithmus).

2. Schreiben Sie die statischen (Titel, Tastenbeschreibungen) und dynamischen Texte (Wert von PI, aktuell verwendeter Algorithmus) in die Anzeige auf dem EduBoard basierend auf den Tastenklicks und dem Ergebnis der Berechnungen.

ControllerTask hört auf die Ereignisse (Start | Stop | Reset | Algorithmus) mit einer unendlichen For-Schleife mit der Verzögerung von vTaskDelay(500/portTICK_RATE_MS), die 500 ms beträgt. Weitere Erläuterungen, wie die Steuerung der Berechnungstasks gehandhabt wird, finden Sie im Abschnitt „Kommunikation zwischen Tasks“ auf den folgenden Seiten.

3.2 ButtonTask - Aufgabe zum Generieren von Taster Ereignissen

Anfänglich wurden während der Implementierung die Tastendrücke unter der controllerTask. Aber um die Lesbarkeit, Modularität und vor allem die Ereignisbits für jeden der vier Tastendrücke zu verbessern, wird eine separate FreeRTOS-Aufgabe buttonTask implementiert.

Als Teil der buttonTask wird eine Event-Gruppe für die Button-Events mit dem Namen egButtonEvents erstellt und dann werden separate Event-Bits für jeden der folgenden Tastendrücke gesetzt:

1. Taste 1 - kurzes Drücken (Start)
2. Taste 2 - kurzes Drücken (Stop)
3. Taste 3 - kurzes Drücken (Reset)
4. Taste 4 - kurzes Drücken (Algorithmus wechseln)
5. Taste 1 - langes Drücken
6. Taste 2 - langes Drücken
7. Taste 3 - langes Drücken
8. Taste 4 - langes Drücken

Tastendrücke werden erfasst und Ereignisbits werden in einer Endlosschleife mit Verzögerung gesetzt vTaskDelay((1000/BUTTON_UPDATE_FREQUENCY_HZ)/portTICK_RATE_MS).

3.3 LeibnizTask - Berechnungstask -1

Implementiert die Schritte zur Berechnung des PI auf Basis der 'Leibniz-Reihe' mittels einer unendlichen For-Schleife.

Der Codeblock der Berechnung:

```
uint32_t n = 3;
float piviertel = 1;
for (;;) {
        piviertel = piviertel - 1.0/n + 1.0/(n+2);
        pi = piviertel * 4;
        n = n + 4;
        vTaskDelay(200/portTICK_RATE_MS);

}
```

Abbildung 1 LeibnizTask Code

3.4 NilakanthaTask – Berechnungstask -2

Implementiert die Schritte zur Berechnung von PI basierend auf der „Nilakantha-Reihe“ unter Verwendung einer endlosen For-Schleife.

Der Codeblock der Berechnung:

```
uint32_t n = 3;
float pi_local = 3.0;
for (;;) {
        pi_local = pi_local + 4.0/(pow(n,3) - n) + 4.0/(pow(n+2,3) - (n+2));
        pi = pi_local;
        n = n + 4;
        vTaskDelay(200/portTICK_RATE_MS);
}
```

Abbildung 2 NilakanthaTask Code

4. Kommunikation zwischen Tasks

Die Kommunikation zwischen verschiedenen Tasks wird unter Verwendung von Ereignisbits und globalen Variablen in der Datei main.c integriert.

Wird eine Event-Gruppe für Button-Events angelegt buttonTask und darunter separate Event-Bits für jedes der 8 Button-Events gesetzt (4 Buttons * kurzes Drücken + 4 Buttons * langes Drücken).

Durch Tastendruck generierte Ereignisbits werden unter ControllerTask

1. BUTTON1_SHORT (Start) – Startet die Berechnung basierend auf dem aktuell ausgewählten Algorithmus.
2. BUTTON2_SHORT (Stop) – Stoppt die Berechnung basierend auf dem aktuell ausgewählten Algorithmus.
3. BUTTON3_SHORT (Reset) – Setzt das Ergebnis der aktuellen Berechnung zurück.
4. BUTTON4_SHORT (Algorithmus) – Schaltet den Wert des aktuellen Algorithmus zwischen „ Leibniz“ und „Nilakantha“.

Langes Drücken einer der 4 Tasten wird nicht unter der ControllerTask, obwohl die relevanten Ereignisbits unter der buttonTask gesetzt werden.

Unter der Hauptfunktion wird jede der 4 Tasks (LeibnizTask, NilakanthaTask, controllerTask und buttonTask) erstellt. Leibniz Task und NilakanthaTask werden anhand der unmittelbarer Erstellung ausgesetzt, um die Kontrolle über die Ausführung ControllerTask und der 4 Button-Events.

Durch kurzes Drücken der Taste „Str“ wird die Ausführung einer Berechnungsaufgabe wieder aufgenommen, während die Taste „Stp“ die Ausführung unterbricht. Die Taste „Rst“ würde das aktuelle Berechnungsergebnis auf „0,0“ zurücksetzen, indem die aktuelle Aufgabe gelöscht wird, und die Taste „Alg“ würde die derzeit ausgeführte Berechnungsaufgabe unterbrechen, während die andere Berechnungsaufgabe fortgesetzt/erstellt wird.

Die folgenden FreeRTOS-Funktionen werden verwendet, um die Ausführung der Aufgaben zu steuern.

- xTaskCreate
- vTaskSuspend
- vTaskResume
- vTaskDelete

FreeRTOS-Task-Zustandsdiagramm:

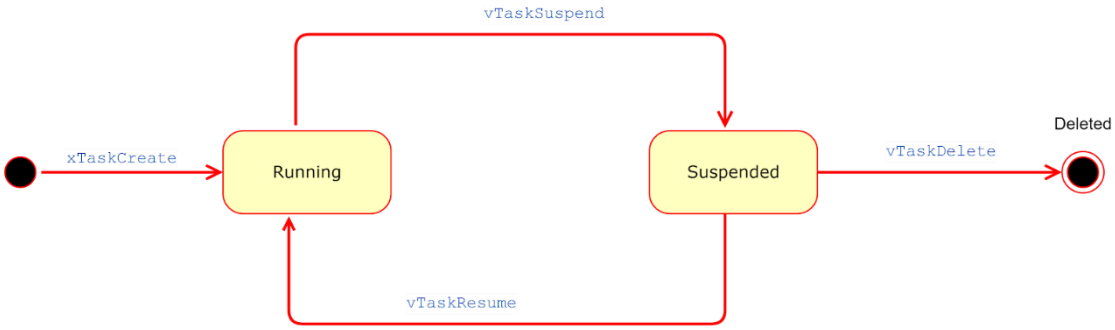


Abbildung 3 Task-state diagram

5. Ergebnisse der Zeitmessung

Die Zeit, die eine Berechnungsaufgabe benötigt, wird mit der FreeRTOS-Funktion xTaskGetTickCount gemessen.

Die Startzeit wird erfasst, wenn die Ausführung der Aufgabe beginnt (z. B., wenn die Taster „Start“ gedrückt wird), und dann wird die Stoppzeit erfasst, wenn der Wert von PI wie der Aufgabe (Ergebnis) eine Genauigkeit von 5 Stellen hinter dem Dezimalpunkt erreicht (3.14159).

Die Zeitdifferenz wird durch Subtrahieren der Startzeit von der Stoppzeit erhalten.

```
void nilakanthaTask(void* pvParameters) {

        TickType_t xStartTimeNilakantha, xStopTimeNilakantha;

        xStartTimeNilakantha = xTaskGetTickCount();
        uint32_t n = 3;
        float pi_local = 3.0;
        for (;;) {
                pi_local = pi_local + 4.0/(pow(n,3) - n) - 4.0/(pow(n+2,3) - (n+2));
                pi = pi_local;
                n = n + 4;
                xStopTimeNilakantha = xTaskGetTickCount();
                if ((floorf(pi *100000) / 100000) != 3.14159f) {
                        xTimeDifferenceNilakantha = xStopTimeNilakantha - xStartTimeNilakantha;
                }

                vTaskDelay(200/portTICK_RATE_MS);
        }
}
```

Abbildung 4 Messzeit Codeblock

6. Vergleich der Rechengeschwindigkeiten

Basierend auf der obigen Zeitmessung nimmt die Berechnung des PI auf 5 Stellen hinter dem Dezimalkomma mit der Nilakantha-Reihe viel weniger Zeit in Anspruch als die Berechnungsaufgabe mit der Leibniz-Reihe.

Ungefähr eine auf der Leibniz-Reihe basierende Berechnung dauert: Über 1000000ms und eine auf der Nilakantha-Reihe basierende Berechnung dauert: 3500 ms

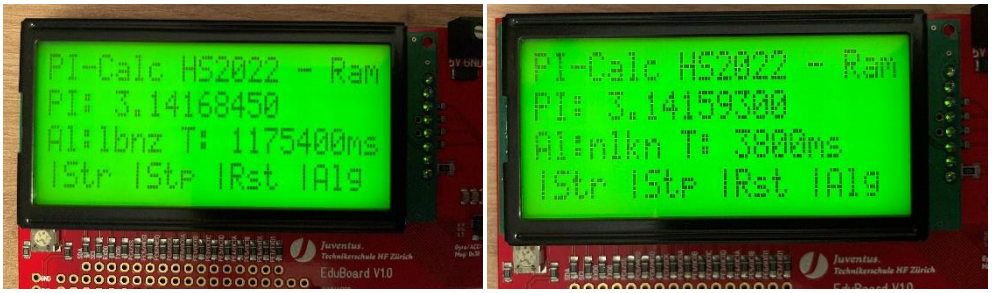


Abbildung 5 Das Ergebnis

7. Code-Repository

Die funktionierende Codebasis ist verfügbar unter:
https://github.com/ram2yk/U_PiCalc_HS2022_BR/tree/develop

8. Bedienungsanweisungen

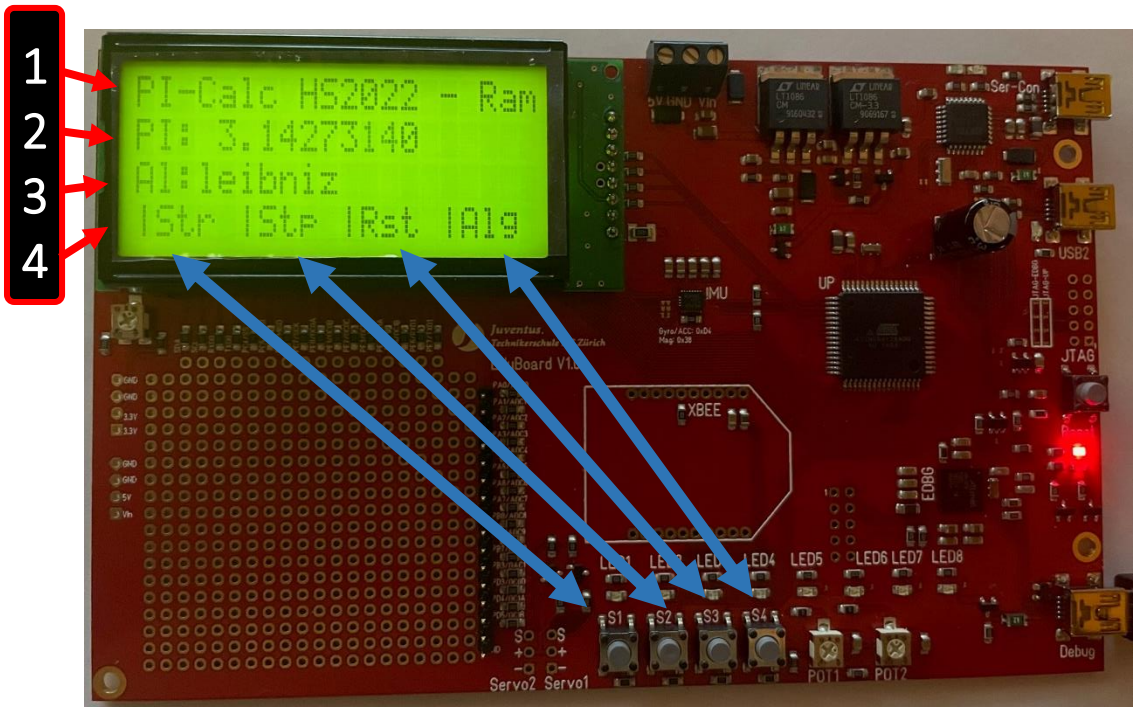


Abbildung 6: EduBoard V1.0

1. DISPLAY Zeile 01: Titel
2. DISPLAY Zeile 02: Berechnete π-Wert
3. DISPLAY Zeile 03: Name des Algorithmus | Zeit [ms]
4. DISPLAY Zeile 04: Taster Funktion

Str = Start = S1 kurz drucken = Berechnung des PI starten mit entsprechende Algorithmus
Stp = Stop = S2 kurz drucken = Berechnung des PI stoppen
Rst = Reset = S3 kurz drucken = PI-Wert zurücksetzen
Alg = Algorithmus = S4 kurz drucken = PI-Wert zu berechnen entsprechende Algorithmus-Formular wechseln.

9. Herausforderungen

1. Um geeignete Zeitverzögerungen über verschiedene Tasks hinweg zu haben (Control , Tasten und Berechnungsaufgaben), musste mit verschiedenen Werten experimentiert werden, um einen Satz geeigneter Endwerte zu erhalten.

2. Die Aufgabenbehandlung, insbesondere vTaskDelete, erforderte eine Aktualisierung der FreeRTOSconfig.h-Datei im Includes-Verzeichnis, dass erst nach Durchlaufen der FreeRTOS-API-Dokumentationen identifiziert wurde.

3. Musste die API-Dokumentationen für die verschiedene APIs von FreeRTOS lesen und verstehen.

10. Ergebnis

Leider hat das Code weiterhin ein paar Fehler, die ich nicht beheben kann. Wie zum Beispiel nach dem Reset muss man zuerst stopp drucken und dann wieder Start. Mit mehr Zeit ist es eventuell möglich dies zu korrigieren. Auf dem Bildschirm funktioniert PI-Berechnung vollständig.
Taster funktionieren ebenfalls. Die Zeitmessung funktioniert so weit. Das Ergebnis ist, dass die Leibniz Methode langsamer ist als das von Nilakantha. Das entspricht ebenfalls meine Erwartungen.

11. Fazit

Im Allgemeinen habe ich keine Erfahrung im Programmieren.

Trotzdem finde ich das Thema sehr spannend und interessant. Ich fand es schön und sehr hilfreich, dass Martin schnell bei Fragen und Problemen half und erklärte. Auch während dem Unterricht, dass er sein Tempo für die Anfänger wie ich anpasste, dass schätze ich sehr.
Ich habe auf jeden Fall einige neue Sachen gelernt. Dank Martin sein GitHub Übungs-link habe ich in Historie Schritt für Schritt gelernt, wo ich anfangen muss (Allgemein Reihenfolgen).

Es war manchmal sehr deprimierend und demotivierend, bei Fehlermeldungen, den die Fehler konnten nicht schnell behoben werden.
Für manche Fehler fand ich weder im Skript noch in der Suchmaschine eine Lösung. In solchen Fällen musste ich den Code anders schreiben. Auf der anderen Seite ist die Freude gross, wenn ich den Fehler selbst fand.

Ich sehe mein Projekt als einen grossen Erfolg! Natürlich habe ich viel mehr Zeit gebraucht als ein Profi oder ein erfahrener Student, trotzdem bin ich froh, dass ich es so weit geschafft habe.
Am Anfang war es für mich nicht selbsterklärend. Das ist ein gutes Gefühl. Ich selbst fühle als hätte ich etwas Grosses gemeistert. Ich werde sicher in der Zukunft weiter an EduBoard üben.

Vielen Dank.

12. Abbildungsverzeichnis

13. Quellen

01 Allgemein alle Übungen und Unterlagen von Dozent: Martin Burger
02 Kreiszahl: https://de.wikipedia.org/wiki/Kreiszahl
03 Nilakantha_Somayaji: https://mathshistory.st-andrews.ac.uk/Biographies/Nilakantha/
04 Annäherung von Pi: https://www.youtube.com/watch?v=dyYpzBQNB5k
05 Beispiel Annäherung von Pi: https://geogebra.org/classic/huhgbksu
06 Kreiszahl Pi berechnen: https://3.14159265358979323846264338327950288419716939937510582097494459 2.eu/pi-berechnen-formeln-und-algorithmen/
07 Madhava of Sangamagrama: https://en.wikipedia.org/wiki/Madhava_of_Sangamagrama
08 Archimedes: https://www.dom-gymnasium.de/mathpage/10/archimed/archimed.html
09 GitHub: https://github.com/Juventus-Technikerschule-HF/U_PiCalc_HS2022/projects?query=is%3Aopen
10 GitHub: https://github.com/Juventus-Technikerschule-HF/U_AlarmClock_SolutionHS2022/commits/master
11 GitForum: https://stackoverflow.com/questions/4089430/how-to-determine-the-url-that-a-local-git-repository-was-originally-cloned-from
12 FreeRTOS: https://www.freertos.org/a00125.html
13 Learn X in Y minutes Where X=C: https://learnxinyminutes.com/docs/c/
14 How do I properly compare strings in C?: https://stackoverflow.com/questions/8004237/how-do-i-properly-compare-strings-in-c
15 Leibniz-Reihe: https://de.wikipedia.org/wiki/Leibniz-Reihe
16 Eulersche Reihentransformation: https://de.wikipedia.org/wiki/Eulersche_Reihentransformation