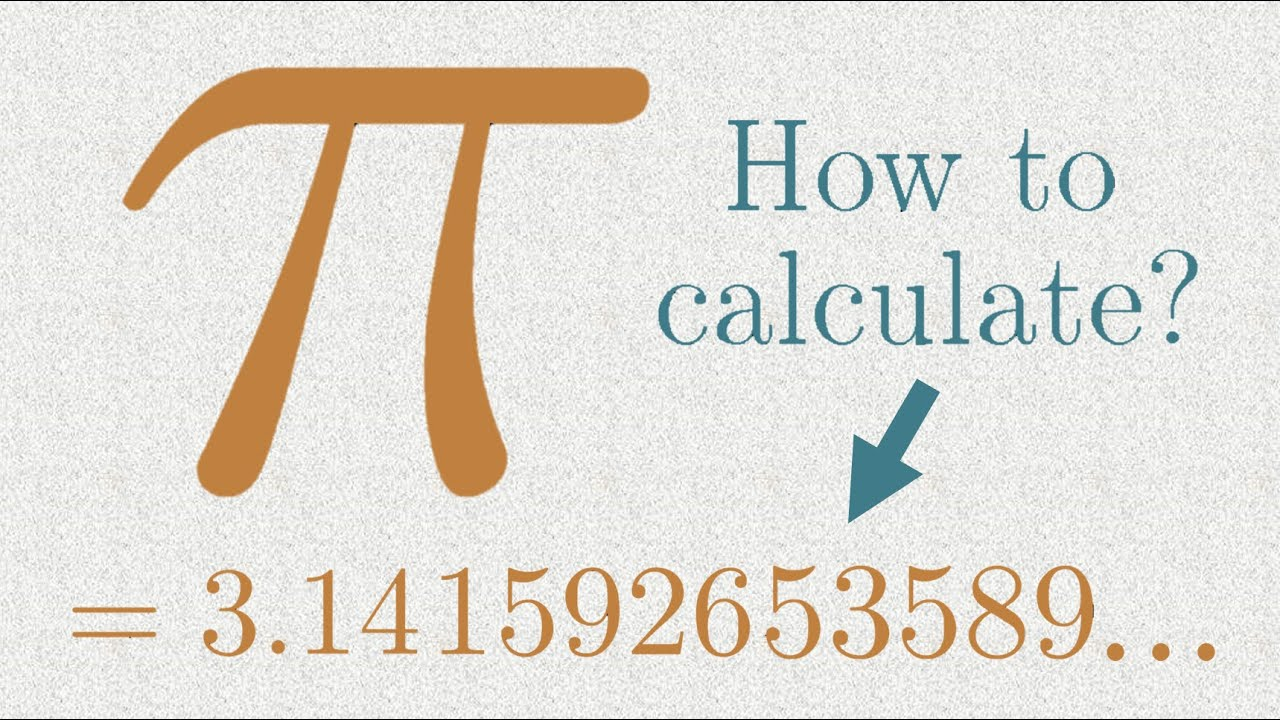
Calculating PI

mit dem EduBoard V1.0

Version 1.0

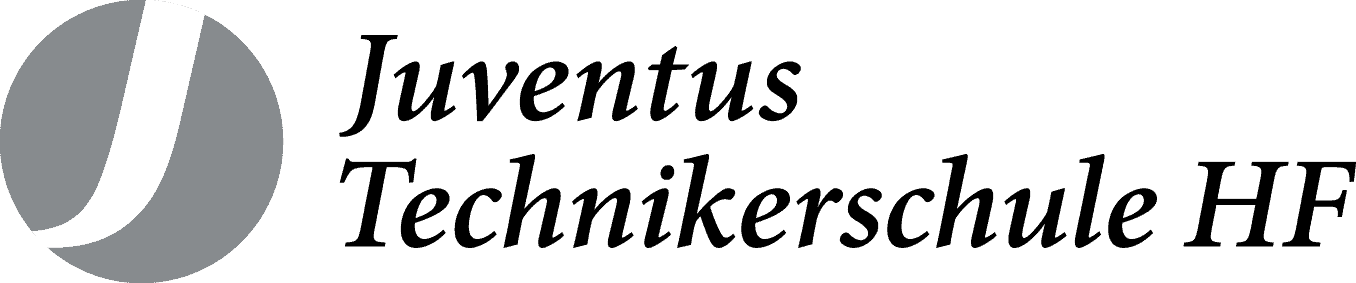


Von Kevin Perillo

Zürich, 24.10.2022

Embedded Systems

Juventus Techniker Schule



Inhaltsverzeichnis

[1 Einleitung 3](#_Toc117543263)

[1.1 Aufgabenstellung 3](#_Toc117543264)

[1.2 Berechnung von Pi 3](#_Toc117543265)

[1.2.1 Archimedes Verfahren / Exhaustionsmethode 3](#_Toc117543266)

[1.3 Algorithmen zur Berechnung von Pi 4](#_Toc117543267)

[1.3.1 Gewällter Algorithmus 5](#_Toc117543268)

[1.4 Entscheid 5](#_Toc117543269)

[2 Firmware 6](#_Toc117543270)

[2.1 Aufbau / Task 6](#_Toc117543271)

[2.2 Ui Task 6](#_Toc117543272)

[2.3 Controll Task 9](#_Toc117543273)

[2.4 Calculate Leibniz Task 9](#_Toc117543274)

[2.5 Calculate Nilakantha Task 9](#_Toc117543275)

[2.6 Calc-Timer Task 9](#_Toc117543276)

[2.7 LED handler Task 10](#_Toc117543277)

[2.8 Event-Bits 11](#_Toc117543278)

[2.9 Auslastung der CPU 11](#_Toc117543279)

[3 Die Zeitmessung 12](#_Toc117543280)

[3.1 Leibniz Task 12](#_Toc117543281)

[3.2 Nilakanatha Task 12](#_Toc117543282)

[4 Fazit 13](#_Toc117543283)

[5 Verzeichnisse 14](#_Toc117543284)

[5.1 Quellenverzeichnis 14](#_Toc117543285)

[5.2 Abbildungsverzeichnis 14](#_Toc117543286)

**Dokumentenhistorie**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Autor | Änderungsgrund/Bemerkungen |
| 1.0 | 24.10.2022 | Kevin Perillo | Initiale Version |

# Einleitung

## Aufgabenstellung

Zu beginn dieser Arbeit haben wir die folgende Aufgabenstellung erhalten:

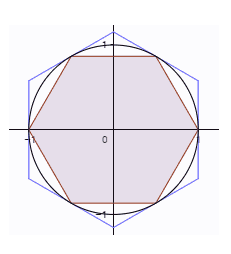
Aufgabe:

* 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.)

## Berechnung von Pi

Aufgrund seiner Transzendenz und Irrationalität weiß man seit langem, dass π nicht nur eine unendlich lange Zahlenfolge darstellt, sondern dass es auch keine einfache Formel für Pi geben kann, die nur aus dem Radius oder dem Durchmesser und ein paar Divisionen und Multiplikationen den Wert von PI berechnet. Auf der anderen Seite hat man Formeln und Algorithmen entdeckt, die von verblüffender Einfachheit und Eleganz sind. Doch alle diese Formeln haben eines gemeinsam. Ohne schwere Rechenarbeit gibt es keinen Lohn. Zum Glück nehmen uns seit Mitte des Zwanzigsten Jahrhunderts moderne Rechenknechte diese Aufgabe ab. Doch angefangen hat es schon vor über 2000 Jahren mit Archimedes von Syracus.

### Archimedes Verfahren / Exhaustionsmethode

Archimedes wählte für seine Berechnung von Pi einen geometrischen Ansatz. Angefangen mit zwei regelmäßigen Sechsecken, die einem Einheitskreis (Kreis mit dem Radius 1) umschrieben bzw. einbeschrieben waren, hangelte er sich über 12-, 24- und 48-Ecke bis hin zu zwei 96-Ecken. Deren Umfang berechnete er mit Hilfe der anderen Zwischenergebnisse und fand so am Ende eine untere und eine obere Grenze für deren Kreisumfang und damit auch für die Zahl Pi. Mit Hilfe der Fläche des Kreises wäre Archimedes zu ähnlichen Ergebnissen gekommen, mit wahrscheinlich etwas schwächeren Schranken. (vgl. Pi Faszination in Ziffern 2022)

Abbildung ‑: Archimedes Verfahren



Abbildung ‑: Formel Archimedes

Damit war Pi auf 2 Nachkommastellen genau berechnet und 3,14 für Jahrhunderte als erster Näherungswert von Pi etabliert. Eine starke Leitung, denn mehr als der Satz des Pythagoras und den Satz des Thales und ein paar ganz elementare geometrische Regeln standen Archimedes nicht zu Verfügung.

## Algorithmen zur Berechnung von Pi

Mit den zuvor genannten Aufgaben, die uns gestellt wurden, habe ich mich auch die Suche nach einem Algorithmus gemacht. In der Aufgabe müssen wir zwei verwenden, davon ist einer mit der Leibniz-Reihe schon vorgegeben.

Sie wird Gottfried Wilhelm Leibniz zugeschrieben, soll aber schon viel früher in Indien benutzt worden sein.

leibniz-reihe

Abbildung ‑: Leibniz-Reihe

Die Reihe stellt einen Sonderfall der Arcustangens Reihe dar (π/4=arctan 1). Als Rechenformel ist sie aber auf Grund ihrer schlechten Konvergenz denkbar ungeeignet. Mathematiker schufen im Laufe der Zeit viele besser geeignete Abwandlungen der Arcustangens Reihe, mit deren Hilfe Pi auf Abermillionen von Stellen berechnet werden konnte. (vgl. Pi Faszination in Ziffern 2022)



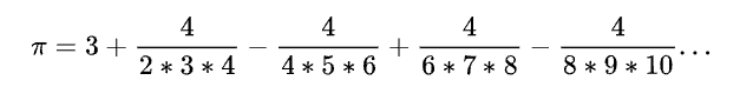
Abbildung ‑: Formel John Machin 1706

Zuerst entschied ich mich für eine Algorithmus der Pi auf sehr viele Stellen nach dem Komma berechnen kann. Im Jahr 2010 stellte der Franzose Fabrice Bellard mit seiner Formel auch den Weltrekord auf. Er berechnete mit seiner Formel Pi auf 2.7 Billionen stellen nach dem Komma. (vgl. Spiegel Wissenschaft 2010)

Während dem Programmieren habe ich mich aber gegen diese Methode entschieden. Warum ich mich dagegen entschieden habe, folgt in einem späteren Abschnitt.

### Gewällter Algorithmus

Ich entschied mich für die Nilakanatha-Reihe.



Die Formel geht auf den indischen Mathematiker und Astronomen Kelallur Nilakanatha Somayaji (1444-1544) zurück und konvergiert nicht sonderlich schnell, witzigerweise berechnen die aufsummierten Brüche aber genau die Nachkommstellen von Pi.

Ich habe mich für diese Reihe entschieden, da sie sich ohne zusätzlichen Mathematischen Funktionen in C berechnen lässt. Da wir auf dem EduBoard im Speicher sehr limitiert sind und ich mit der ersten Rechnung Speicherplatz-Probleme bekommen habe, entschied ich mich für die Nilakanatha-Reihe. Entsprechend der Schwierigkeit der Formel, lässt sich die Zahl Pi sehr schnell und mit wenig Ressourcen berechnen.

## Entscheid

Da ich wie oben angesprochen habe mit Platz-Problemen auf dem Eduboard zu kämpfen hatte. Passte ich meine anfänglichen Ziele etwas an. Ich verzichtete auf die Verwendung der 64-Bit Float Library. Dies aus dem Grund, dass ich auf dem Display keinen Platz habe, um eine so grosse Zahl anzeigen zu können. Die Umsetzung für diesen Variabeltyp ist nicht gross. Wichtig hier ist, man darf nicht vergessen, dass die Zahl nach der Berechnung in einen String gewandelt werden muss, ansonsten kann man die Zahl im Debugger nicht anschauen. Ebenso kann die Zahl nicht einfach so berechnet werden. Für die Umsetzung muss die Library Funktionen zur Addition, usw. verwendet werden. Dies weil es sich beim 64-Bit Float nicht um eine echte Zahl handelt mit dem das System arbeiten kann.

Auf dem Display wurden auch ein paar Platzsparmassnahmen umgesetzt. So wurde:

* Start | Stop | Reset | Toggle Anzeige auf der untersten Zeile auf ein Minimum beschränkt
* Die Zeitanzeige wurde mit ‘T’ abgekützt

Ebenso wurde die Funktion configASSERT( task\_status == pdPASS ); bei der Erstellung eines neuen Task auskommentiert. Dies aus dem Grund, dass diese Funktion, relativ viel Speicherplatz benötigt und diese Applikation nicht genügen Speicher und auch nicht genügend kritisch ist, um diese Funktion drin zu lassen.

# Firmware

## Aufbau / Task

Für den Aufbau dieser Firmware habe ich mich dazu entschieden folgende Task zu erstellen:

* Ui Task
* Controll Task
* Calculate Leibniz Task
* Calculate Nilakantha Task
* Calc-Timer Task
* LED handler Task

Zudem habe ich versucht den Code möglichst modular und unabhängig voneinander zu programmieren. Da wir noch keinen Mutex, Semaphoren oder Queues verwenden sollten, habe ich versucht die Variablen nur an einem Ort zu schreiben und an anderen Orten im Code dieselben Variablen nur zu lesen. So kann man auch ohne blockieren der Variable einen Crash verhindern. Das Problem ist, man muss sehr aufpassen, dass man nicht gleichzeitig an eine Adresse schreiben will. Ansonsten fängt an diesem Punkt die Fehlersuche an.

## Ui Task

Der Ui Task befindet sich im Ui-Modul (ui.c/ui.h). Der Ui-Task ist sehr simpel aufgebaut. Zuerst werden alte Daten auf dem Display entfernt. Danach wir der Titel des Projektes (Pi Calc V1.0 PEK) auf dem Display dargestellt.

Im Anschluss werden in einer Fine-State Machine verschiedene Modis durchlaufen. Beim Aufstarten wird immer zuerst der Idle-Modus dargestellt. In diesem Modus wird gewartet, bis eine der beiden Reihen ausgewählt wurde. Dies wird mit dem Taster 4 ausgewählt, Wenn eine Reihe ausgewählt wird, wir die Auswahl als «Selection: XXX» auf dem Display dargestellt. Der Modus bleibt aber noch im Idle so lange bis die Berechnung gestartet wurde.

Ein Bild, das Text, drinnen enthält.

Automatisch generierte Beschreibung

Abbildung ‑: Idle Mode nach dem Aufstarten

Idle Mode nachdem eine Reihe ausgewählt wurde:

Ein Bild, das Text, drinnen enthält.

Automatisch generierte Beschreibung

Abbildung ‑: Idle Mode mit Selektion LBZ

Die andere Reihe als Auswahl:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung ‑: Idle Mode mit Selekton NLK

Bei einer gestarteten Berechnung ändert sich die Anzeige auf «Mode: Calc xxxxx»

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung ‑: Mode Berechnung

Bei der Berechnung steht als Zeit zuerst 0s. Dies bleibt so lange bis Pi berechnet wurde. So kann daraus geschlossen werden, dass Pi noch berechnet wird.

Ist die Berechnung abgeschlossen, wird die benötigte Zeit neben Pi dargestellt.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung ‑: Abgeschlossene Berechnung

## Controll Task

Der Controll Task steuert die Tasks. Die Buttons werden alle 10ms eingelesen. In diesem Projekt wurde die Button Library für das EduBoard verwendet.

## Calculate Leibniz Task

Der Leibnitz Task berechnet die Zahl Pi mit der Leibniz Reihe. Direkt nach dem Aufstarten, wartet der Task nach der Initialisierung wartet der Task auf das «Select LBZ» Event-Bit. Danach wartet der Task auf das «Start» Event-Bit. Wir dieses Bit durch den Tastendruck aktiviert beginnt die Berechnung.

Nach dem die Berechnung beendet wurde, wird geprüft, ob das «Reset» Bit gesetzt wurde. Wird das Bit gesetzt, werden die Variablen, welche zur Berechnung verwendet werden, auf ihren Ursprung zurückgesetzt.

## Calculate Nilakantha Task

Dieser Task berechnet die Zahl Pi mit der Nilakantha-Reihe. Beide Berechnungstasks sind vom Aufbau her gleich aufgebaut. Somit wartet dieser Task zuerst auch auf seine Selektierung und danach auf das «Start» Event-Bit. Danach berechnet auch dieser Task in einer Endlosschleife die Zahl Pi ohne Verzögerung.

Wurde die Berechnung gestoppt, so wird in diesem Task auch auf das «Reset» Bit gewartet.

## Calc-Timer Task

Der Calculation Timer Task misst die Zeit, welche für die Berechnung von Pi auf 6 Stellen nach dem Komma benötigt werden. Ich entschied mich die 6. Kommstelle zu prüfen, da wir keine Vorgabe hatten und ich auf dem Display auch nicht mehr Platz habe, um mehr Stellen anzeigen zu können.

Nachdem das «Start» Event-Bit gesetzt wurde wird die Zeit mit der Funktion aufgenommen:

pst\_time->start\_time = xTaskGetTickCount( );

Die Variable wird hier auch wieder Global gespeichert, damit ich den Wert aus einem anderen Task lesen kann. Danach wird die aktuelle Zahl Pi durch eine Multiplikation in eine Ganzzahl gewandelt und verglichen. Stimmt das multiplizierte Pi mit dem definierten Wert, weiss ich das Pi genau genug ist. In diesem Fall wird die Endzeit aufgenommen und das Delta von den beiden Werten wird berechnet.

pst\_time->stop\_time = xTaskGetTickCount( );

pst\_time->calc\_time = pst\_time->stop\_time - pst\_time->start\_time;

Danach wird der LED Task und der Ui Task über die Event-Bits informiert, dass die Zahl bereit ist. Die benötigte Zeit wird auf dem Display dargestellt.

## LED handler Task

Der LED-Task wartet, bis das jeweilige «Status» Bit der Berechnungen gesetzt wurden. Wird eines dieser Bits gesetzt, wird entweder die LED 1 oder LED 2 gesetzt. Die LED 1 zeigt an, dass die Leibniz-Berechnung genau genug ist. Die LED 2 ist dementsprechend den anderen Berechnungs-Task.

Ein Bild, das Text, Elektronik, Schaltkreis enthält.

Automatisch generierte Beschreibung

Abbildung ‑: LED-Anzeige Leibniz-Task genau

## Event-Bits

Für dieses Programm habe ich mich dazu entschieden, die maximale Anzahl an verfügbaren Event-Bits. Da wir in diesem Fall mit einem AtMega128A3U arbeiten, einem Mikrokontroller mit einer 8-Bit Architektur, sind max. 8 Event-Bits möglich.

Dabei habe ich mich für die folgenden Event-Bits entschieden:

/\*--- Event Bits --------------------------------------------------------------- \*/

#define START\_CALC ( 1U << 0UL ) // Start Calculation

#define STOP\_CALC ( 1U << 1UL ) // Stop Calculation

#define RESET\_CALC ( 1U << 2UL ) // Reset Calculation task

#define CALC\_SEL\_LBZ ( 1U << 3UL ) // Select Calculation algorithm

#define CALC\_SEL\_NLK ( 1U << 4UL ) // Select Calculation algorithm

#define BIT5 ( 1U << 5UL ) // unused

#define LBZ\_STATE ( 1U << 6UL ) // Leibnitz Task state

#define NLK\_STATE ( 1U << 7UL ) // Bellard Task state

Das «CALC\_SEL\_LBZ» Bit selektiert den Task welchen Pi mit der Leibniz-Reihe berechnet wird. Das Bit «CALC\_SEL\_NLK» selektiert den anderen Berechnungstask. Es ist jedoch nicht möglich, dass beide Bits gleichzeitig gesetzt wurden. Mit dem «START\_CALC» Bit wird der Selektierte Task gestartet. Mit dem «STOP\_CALC» wird der laufende Task gestoppt. Die beiden «STATE» Bits (Bit 6 & Bit 7) zeigen den Status der Pi-Berechnung. Der Status bestätigt dem LED-Task, dass die LED für die jeweilige Berechnung darstellt. Nach der Anzeige wird das Bit wieder gelöscht.

Das «RESET\_CALC» Bit setzt die Berechnungen der Werte, sowie die Zeit und den Zustand der LED.

Das Bit 5 wurde nicht gebraucht. Es wurde als Platzhalter definiert.

## Auslastung der CPU

Formal wird die Prozessorauslastung als Verhältnis von nutzbarer Arbeitszeit zu geleisteter Arbeitszeit des Prozessors berechnet.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung ‑: Formel zur Berechnung der CPU-Auslastung

Eine CPU kann maximal zu 100% ausgelastet werden. Somit liegt ᶯ zwischen 0 und 1.

Die Berechnung der Leibniz-Reihe benötigt Additionen von Brüchen, welche mehrere CPU-Takte für die Berechnung. Beim Nilakanatha Task besteht die Berechnung aus einem Bruch und Multiplikationen. Vom Zeitaufwand denke ich, die beiden Tasks benötigen ca. gleich viel Auslastung. Wie viel genau kann ich aktuell nicht Berechnen. Ich schätze aber, dass die Auslastung mit dieser Rechnung noch nicht gross sein wird.

In einem Experiment habe ich den MCU mit einer etwas schwierigeren Formel laufen lassen, welche ich im Internet gefunden hatte. Dabei wurde der Chip spürbar wärmer, somit auch etwas mehr ausgelastet wie in diesem Projekt. Die Dokumentation dieses Experimentes würde aber den Rahmen dieser Arbeit sprengen. Den Versuch findet man auch auf Github.

# Die Zeitmessung

Mit dem Zeit-Mess-Task konnte ich die benötigte Zeit messen. Daraus konnte ich folgendes schliessen:

## Leibniz Task

Beim Leibniz Task habe ich eine Berechnungszeit von ca. 10s gemessen. Die oben beschriebene Reihe

## Nilakanatha Task

Beim Nilakanatha Task habe ich eine Berechnungszeit von ca. 10ms gemessen. Ich denke einen Grund, warum diese Berechnung so viel schneller geht wie die Leibnitz-Reihe, ist das die Berechnung um einiges einfacher ist. Mit wenigen Rechnungen kommt man mit dieser Reihe auf die benötigte Genauigkeit.

Die Leibnitz-Reihe ist somit 1000x langsamer in der Berechnung.

# Fazit

In diesem Projekt war die Aufgabe Pi zu berechnen. Beim Programmieren hatte ich keine Probleme. Für mich persönlich, war das Schwierigste ein Programm zu schreiben, dass für mich keinen brauchbaren Sinn hat. Zu dieser Aussage kam ich, weil Pi in der Math Library (math.h) bereits auf mehrere Stellen nach dem Komma definiert ist. Ich hätte eine freiere Aufgabe viel besser gefunden. So hätte man auch die verschiedenen Level an Programmierkenntnissen besser auslasten können. In dem man z.B. die Sensoren ansteuert, oder etwas dazu bauen hätte können.

Mit dem Resultat dieser Arbeit bin ich zufrieden. Es Erfüllt die Anforderungen gemäss der Aufgabe. Ich fand es dennoch interessant wie schnell man an die Grenzen dieses Chips kommt. Bestes Beispiel hier war das hinzufügen des Calc-Timer Task. Als ich diesen hinzugefügt hatte funktionierte plötzlich die Display Anzeige nicht mehr. Auch die Buttons wurden nicht korrekt eingelesen. Zu meinem Glück war mir dieses RTOS-Problem schon etwas bekannt von früheren Projekten. So konnte ich das Problem lösen, indem ich die Stack Grössen für alle Tasks neu definiert habe, bzw. die Grösse aller Stacks angepasst habe.

# Verzeichnisse

## Quellenverzeichnis

* Leibniz-Reihe (2022): <https://de.wikipedia.org/wiki/Leibniz-Reihe>
* FreeRTOS (2022): [www.freertos.org](http://www.freertos.org)
* Wie Bellard gerechnet hat (2010): [Spiegel Wissenschaft](https://www.spiegel.de/wissenschaft/technik/pi-rekord-wie-bellard-gerechnet-hat-a-685679.html)
* Pi Faszination (2022): <https://3.14159265358979323846264338327950288.eu/pi-berechnen>

## Abbildungsverzeichnis

[Abbildung 1‑1: Archimedes Verfahren 3](file:///C:\Entwicklung\Projekte\U_ES_PiCalc\Doku_Pi_Calc.docx#_Toc117543246)

[Abbildung 1‑2: Formel Archimedes 4](#_Toc117543247)

[Abbildung 1‑3: Leibniz-Reihe 4](#_Toc117543248)

[Abbildung 1‑4: Formel John Machin 1706 4](#_Toc117543249)

[Abbildung 2‑1: Idle Mode nach dem Aufstarten 6](#_Toc117543250)

[Abbildung 2‑2: Idle Mode mit Selektion LBZ 7](#_Toc117543251)

[Abbildung 2‑3: Idle Mode mit Selekton NLK 7](#_Toc117543252)

[Abbildung 2‑4: Mode Berechnung 7](#_Toc117543253)

[Abbildung 2‑5: Abgeschlossene Berechnung 8](#_Toc117543254)

[Abbildung 2‑6: LED-Anzeige Leibniz-Task genau 10](#_Toc117543255)

[Abbildung 2‑7: Formel zur Berechnung der CPU-Auslastung 11](#_Toc117543256)