FreeRTOS PI\_Calculation

[Firmenname] | [Firmenadresse]

Juventus schule | Embedded system

Bryan Buonocore

2023

Inhaltsverzeichnis

[Inhaltsverzeichnis 1](#_Toc149591858)

[Aufgabenstellung: 2](#_Toc149591859)

[Algorithmus: 3](#_Toc149591860)

[Entscheid: 3](#_Toc149591861)

[Erklärung: 3](#_Toc149591862)

[Beschreibung der Tasks: 4](#_Toc149591863)

[Berechnung der Leibniz Reihe: 4](#_Toc149591864)

[Berechnung der Nilkantha Methode: 5](#_Toc149591865)

[Button Handling: 6](#_Toc149591866)

[InterfaceTask: 7](#_Toc149591867)

[Resultate und Erklärung 9](#_Toc149591868)

[Methoden vergleichen: 9](#_Toc149591869)

[Erklärung: 9](#_Toc149591870)

[Rückschluss bezüglich Rechenleistung zur tatsächlichen Prozessorleistung 9](#_Toc149591871)

[Kurzes Schlusswort 9](#_Toc149591872)

[Quellenverzeichnis: 10](#_Toc149591873)

# 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:

o EventBit zum Starten des Algorithmus.

o EventBit zum Stoppen des Algorithmus.

o EventBit zum Zurücksetzen des Algorithmus.

o EventBit für den Zustand des Kalkulationstask als Mitteilung für den Anzeige- Task.

- Mindestens drei Tasks müssen existieren.

o Interface-Task für Buttonhandling und Display-Beschreiben.

o Kalkulations-Task für Berechnung von PI mit Leibniz Reihe.

o 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.)

# Algorithmus:

Am Anfang der Arbeit habe ich mich für die Methode von Gregory entschieden. Mir wurde klar als ich mich anschliessend mir der Methode befasst habe, dass Gregory an der Leibniz Reihe mit entwickelt hat. Anschliessend habe ich den Algorithmus von Archimedes probiert. Es war sehr ungenau und ich landete nie auf PI mit fünf Kommastellen. Nachdem habe ich viele andere Methoden probiert, doch ich schaffte es nie richtig auf PI zu kommen.

## Entscheid:

Anschiessend bin ich auf die Methode von Nilakantha zugestossen. Ich fand diese Formel sehr verständlich und nach so vielen versuchen Pi auszurechnen, habe ich mich für diese Methode Entschieden. Die Formel habe ich in einem PDF-Dokument gefunden.



Die Formel:

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

Die Formel im Code:

## Erklärung:

Die Variable Pi hat ein Wert von drei, da der Startwert in dieser Formel drei ist. Sign hat den Wert eins und ist dazu da, um das Vorzeichen von Plus zu Minus und umgekehrt umzuwandeln. In der Nilakantha Formel wechselt das Vorzeichen bei jeder Term Berechnung. Das k ist auf null gesetzt und es wird für den Laufindex benutzt. "k" wird in jedem Schleifendurchlauf inkrementiert, um den nächsten Term in der Reihe zu berechnen. Es dient dazu, die verschiedenen Termen der Formel voneinander zu unterscheiden und Schrittweise zur näherung von Pi beizutragen. Der Rest vom Code ist einfach abgeschrieben und umgewandelt von der Formel, die ich im PDF-Dokument gesehen habe.

# Beschreibung der Tasks:

In diesem Abschnitt werde ich die einzelnen Tasks erklären. Ich werde die einzelnen Tasks mit Printscreen schneiden und in das Word kopieren. Anschliessend werde ich kleinere Kapitel machen, indenen ich die Tasks erkläre, um es Verständnisvoll zu machen.

## Berechnung der Leibniz Reihe:

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

**vLeibnizTask-Funktion:** Dies ist die Funktion, die als Task ausgeführt wird. Sie erwartet einen Zeiger auf Parameter (pvParameters), aber in diesem Fall werden keine Parameter verwendet.

**Schleife (for(;;)):** Dieser Task befindet sich in einer Endlosschleife, was bedeutet, dass er kontinuierlich ausgeführt wird, solange das Programm läuft.

**Button-Überprüfung:** Zunächst wird überprüft, ob ein "Button" gedrückt wurde, um die Berechnung zu starten. Dies wird durch eine Event-Gruppe (vStartBitLeibniz) und die Funktion xEventGroupGetBits überwacht. Wenn die Bedingung erfüllt ist (wenn das Bit auf 1 gesetzt ist), wird der folgende Code ausgeführt.

**Zeitmessung starten:** Dieser Abschnitt stellt sicher, dass die Zeitmessung nur einmal gestartet wird. Wenn runningLeibniz gleich 1 ist, wird die aktuelle Systemzeit mit xTaskGetTickCount erfasst und runningLeibniz auf 0 gesetzt, um sicherzustellen, dass dies nur einmal geschieht.

**Leibniz-Formel zur Berechnung von Pi:** Hier wird die Leibniz-Formel für die Näherung von Pi verwendet. Die Formel wird in jedem Schritt der Schleife berechnet, um immer näher an Pi zu gelangen. Die Variable Pi4 ist auf eins gesetzt und die Schleifenvariable n wird um vier erhöht. Die Ergebnisse der Formel werden in pi\_Leibniz gespeichert.

**Überprüfung und Zeitmessung stoppen:** Nach jedem Berechnungsschritt wird überprüft, ob pi\_Leibniz im Bereich von 3.14159 und 3.14160 liegt. Wenn dieser Bereich erreicht wird, wird die Zeitmessung gestoppt, indem die aktuelle Systemzeit von der zuvor gespeicherten Zeit abgezogen wird.

**Stop-Signal:** Wenn das Stopsignal betätigt wird (wenn ein "Stop-Button" gedrückt wird), wird die Event-Gruppe vStartBitLeibniz auf 0 gesetzt, was bedeutet, dass die Berechnung gestoppt wird.

## Berechnung der Nilkantha Methode:

Ein Bild, das Text, Screenshot, Software, Display enthält.

Automatisch generierte Beschreibung

**vNilkanthaTask-Funktion:** Dies ist die Funktion, die als Task ausgeführt wird. Sie erwartet einen Zeiger auf Parameter (pvParameters), aber in diesem Fall werden keine Parameter verwendet.

**Schleife (for(;;)):** Dieser Task befindet sich in einer Endlosschleife, was bedeutet, dass er kontinuierlich ausgeführt wird, solange das Programm läuft.

**Button-Überprüfung:** Innerhalb der Schleife wird zunächst überprüft, ob ein "Button" gedrückt wurde, um die Berechnung zu starten. Dies erfolgt durch das Überprüfen einer Event-Gruppe (vStartBitNilkantha) mithilfe der Funktion xEventGroupGetBits. Wenn die Bedingung erfüllt ist (wenn das Bit auf 1 gesetzt ist), wird der folgende Code ausgeführt.

**Zeitmessung starten:** Dieser Abschnitt stellt sicher, dass die Zeitmessung nur einmal gestartet wird. Wenn runningNilkantha gleich 1 ist, wird die aktuelle Systemzeit mit xTaskGetTickCount erfasst und runningNilkantha auf 0 gesetzt, um sicherzustellen, dass dies nur einmal geschieht.

**Berechnung der Pi-Approximation:** In diesem Abschnitt wird die Nilakantha-Reihe verwendet, um eine Näherung von Pi zu berechnen. Die Variable Pi startet mit dem Wert drei und wird schrittweise aktualisiert, indem sie um den Wert (4.0 / ((2\*k+2) \* (2\*k+3) \* (2\*k+4))) erhöht oder verringert wird. Die Variable k wird ebenfalls inkrementiert und das Vorzeichen (sign) wird gewechselt, um die Berechnung fortzusetzen.

**Überprüfung und Zeitmessung stoppen:** Nach jedem Berechnungsschritt wird überprüft, ob pi\_Nilkantha im Bereich von 3.14159 und 3.14160 liegt. Wenn dieser Bereich erreicht wird, wird die Zeitmessung gestoppt, indem die aktuelle Systemzeit von der zuvor gespeicherten Zeit abgezogen wird.

**Stop-Signal:** Wenn das Stopsignal angezeigt wird (wenn ein "Stop-Button" gedrückt wird), wird die Event-Gruppe vStartBitNilkantha auf 0 gesetzt, was bedeutet, dass die Berechnung gestoppt wird.

## Ein Bild, das Text, Screenshot, Software, Display enthält. Automatisch generierte BeschreibungButton Handling:

**Button-Initialisierung (initButtons):** Die Funktion initButtons() wird zu Beginn aufgerufen, um die Tasten zu initialisieren. Dies bedeutet, dass die Tasten auf die richtige Weise konfiguriert und vorbereitet werden, um ihre Zustände zu überwachen.

**Endlosschleife (for(;;)):** Der Task wird in einer Endlosschleife ausgeführt, was bedeutet, dass er kontinuierlich läuft und die Zustände der Tasten überwacht werden, solange die Anwendung aktiv ist.

**Button-Aktualisierung (updateButtons):** In jedem Schleifendurchlauf wird die Funktion updateButtons() aufgerufen, um die aktuellen Zustände der Tasten zu aktualisieren. Dies bedeutet, dass der Task die Zustände der Tasten ausliest, um zu erkennen, ob sie gedrückt wurden oder nicht.

**Tastenstatus überprüfen und Event-Bits setzen:** Nachdem die Tasten aktualisiert wurden, wird der Zustand jeder Taste überprüft, um festzustellen, ob sie kurz gedrückt wurden (SHORT\_PRESSED). Wenn dies der Fall ist, wird das entsprechende Event-Bit in einer Event-Gruppe (evButtonEvents) gesetzt. Diese Event-Bits dienen dazu, auf Tastendrücke zu reagieren und bestimmte Aktionen im Programm auszulösen. Es gibt separate Event-Bits für jede Taste (z.B., EVBUTTONS\_S1, EVBUTTONS\_S2, EVBUTTONS\_S3, EVBUTTONS\_S4) und sie werden je nach gedrückter Taste gesetzt.

**Button-Update-Verzögerung:** Am Ende jeder Schleife wird vTaskDelay verwendet, um eine Verzögerung einzuführen. Diese Verzögerung ermöglicht es, die Häufigkeit der Tastenaktualisierungen auf eine bestimmte Frequenz (BUTTON\_UPDATE\_FREQUENCY\_HZ) zu begrenzen.

## InterfaceTask:

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

**Schleife (for(;;)):** Diese Task wird in einer Endlosschleife ausgeführt, was bedeutet, dass sie kontinuierlich läuft solange die Anwendung aktiv ist.

**Button-Status ermitteln:** Zuerst wird der Status der Tasten überprüft, indem die Bits in der Event-Gruppe evButtonEvents gelesen werden. Mithilfe von Bitmaskierung (& 0x000000FF) wird sichergestellt, dass nur die unteren 8 Bits verwendet werden, um den Tastenstatus zu ermitteln. Danach werden die Event-Bits mit xEventGroupClearBits zurückgesetzt, um festzustellen, welche Tasten betätigt wurden.

**Switch-Anweisung für Button-Status:** Die Schaltanweisung (Switch-Case) wird verwendet, um je nach erkanntem Tastenstatus verschiedene Aktionen auszuführen:

**EVBUTTONS\_S1** (Button 1 kurz gedrückt): In Abhängigkeit vom aktuellen Algorithmus (für die Berechnung von Pi) werden die entsprechenden Task-Steuerungsbits (vStartBitLeibniz oder vStartBitNilkantha) gesetzt, um die Berechnung zu starten. Die Variablen runningLeibniz oder runningNilkantha werden ebenfalls auf eins gesetzt (Für die Zeitberechnung).

**EVBUTTONS\_S2** (Button 2 kurz gedrückt): Das Bit vStopBit wird gesetzt, um die Berechnung zu stoppen.

**EVBUTTONS\_S3** (Button 3 kurz gedrückt): Je nach aktuellem Algorithmus werden verschiedene Variablen zurückgesetzt, um die Berechnung neu zu starten. Dies schließt die Zeitmessung und die Ergebnisse ein.

**EVBUTTONS\_S4** (Button 4 kurz gedrückt): Je nach aktuellem Algorithmus wird entweder die Berechnung mit dem Leibniz-Algorithmus oder dem Nilkantha-Algorithmus gestoppt und der aktive Algorithmus gewechselt.

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

**Anzeige der Ergebnisse auf dem Display:** Abhängig vom aktuellen Algorithmus (aktuellAlgorithmus) werden Pi- und Zeitinformationen auf dem Display angezeigt. Dazu werden Strings für Pi-Werte und Zeitmessungen erstellt und mit vDisplayWriteStringAtPos auf dem Display platziert.

**Aktualisierung der Anzeige:** Um sicherzustellen, dass die Anzeige nicht zu schnell aktualisiert wird, wird eine Verzögerung von 500 Millisekunden (0,5 Sekunden) mit vTaskDelay eingeführt. Dies ermöglicht es dem Benutzer, die auf dem Display angezeigten Informationen in Ruhe zu betrachten.

# Resultate und Erklärung

## Methoden vergleichen:

Der Leibniz Task braucht ein bisschen mehr als 12 Sekunden damit Pi genau ausgerechnet wird. Der Nilkantha hingegen braucht nur 4 Millisekunden, bis Pi genau berechnet wird.

## Erklärung:

**Schnellere Konvergenz:** Der Nilkantha-Algorithmus konvergiert schneller gegen den exakten Wert von Pi. Dies bedeutet, dass er mit weniger Iterationen eine genauere Näherung erreicht. Der Leibniz-Algorithmus konvergiert im Vergleich dazu langsamer.

**Günstige Annäherung von Anfang an:** Der Nilkantha-Algorithmus beginnt mit einer anfänglichen Näherung von Pi, die näher am tatsächlichen Wert liegt (3), während der Leibniz-Algorithmus mit einer anfänglichen Näherung (1) beginnt, die weiter vom Ziel entfernt ist. Dies bedeutet, dass der Nilkantha-Algorithmus weniger Iterationen benötigt, um eine genauere Näherung zu erreichen.

# Rückschluss bezüglich Rechenleistung zur tatsächlichen Prozessorleistung

**Algorithmus-Effizienz:** Die gemessene Laufzeit hängt stark von der Effizienz des verwendeten Algorithmus ab. Ein effizienter Algorithmus führt Aufgaben mit weniger Rechenressourcen (CPU-Zeit) schneller aus. In Ihrem Fall ist der Nilkantha-Algorithmus effizienter als der Leibniz-Algorithmus, was zu kürzeren Laufzeiten führt.

**Prozessorlast**: Andere laufende Aufgaben oder Prozesse auf demselben Prozessor können die Prozessorleistung beeinträchtigen. Wenn der Prozessor stark ausgelastet ist, kann dies zu längeren Laufzeiten führen.

**Weniger Berechnungen pro Iteration:** Der Nilkantha-Algorithmus erfordert weniger Berechnungen pro Iteration im Vergleich zum Leibniz-Algorithmus. Weniger Berechnungen bedeuten weniger Rechenaufwand, beanspruchen weniger Prozessorleistung und ergeben somit auch eine schnellere Berechnung.

# Kurzes Schlusswort

Für mich war dieses Projekt sehr lehrreich und erfolgreich. Ich konnte mich in diesem Bereich weiterentwickeln und viel Wissen dazu gewinnen. Da ich noch nie richtig programmiert habe, war das eine Challenge für mich, die ich aber dank der Hilfe meines Dozenten gut gemeistert habe. Für kleinere Fehler konnte ich ebenfalls Chat GPT in Anspruch nehmen.

# Quellenverzeichnis:

Pi Formel Nilkantha: PDF wird in GitHub dazu gelegt.

Pi Formel Leibniz: [Leibniz-Reihe – Wikipedia](https://de.wikipedia.org/wiki/Leibniz-Reihe)

Chat GPT: Um Fehler im Code zu analysieren.