Embedded Systems

Calculating Pi

Von **Erik Haubrich**

**Inhaltsverzeichnis:**

[Aufgabenstellung 3](#_Toc149227018)

[1 Algorithmen 4](#_Toc149227019)

[1.1 Leibniz-Reihe 4](#_Toc149227020)

[1.2 Nilakantha-Reihe 4](#_Toc149227021)

[1.3 Entscheid 4](#_Toc149227022)

[2 Programm 5](#_Toc149227023)

[2.1 Aufbau 5](#_Toc149227024)

[2.2 Tasks 5](#_Toc149227025)

[2.3 Controller-/Interface-Task 5](#_Toc149227026)

[2.3.1 Benutzeroberfläche 5](#_Toc149227027)

[2.3.2 Menüauswahl 6](#_Toc149227028)

[2.3.3 Math.h Demo 7](#_Toc149227029)

[2.3.4 Leibniz’s Pi 7](#_Toc149227030)

[2.3.5 Nilakantha’s Pi 7](#_Toc149227031)

[2.4 Buttontask 8](#_Toc149227032)

[2.5 Leibniz-Folge-Task 9](#_Toc149227033)

[2.6 Nilakantha-Folge-Taks 9](#_Toc149227034)

[3 Messbericht Thema – 3 11](#_Toc149227035)

[3.1 Kapitel Aufgabenstellung 11](#_Toc149227036)

[3.2 Kapitel Theorie 11](#_Toc149227037)

[Persönliches Fazit 13](#_Toc149227038)

[Literaturverzeichnis 14](#_Toc149227039)

[Abbildungsverzeichnis 15](#_Toc149227040)

[Anhang 16](#_Toc149227041)

# Aufgabenstellung

Als benotete Übung ist folgende Aufgabenstellung gegeben.

«Es gibt diverse Algorithmen, wie man PI berechnen kann. Einige sind schneller als andere. Ein sehr simpler, jedoch auch langsamer Ansatz ist folgender: Wenn man in einem Quadrat mit der Seitenlänge 1 zwei zufällige Punkte wählt und deren Distanz zur linken unteren Ecke berechnet, dann bekommt man entweder einen Wert über oder unter 1.

Nun besagt der Ansatz, dass das Verhältnis der Punkte im Viertel-Kreis drin im Vergleich zum ganzen Quadrat einem Viertel PI entspricht. Diese Tatsache kann man sich zum Vorteil nehmen und Punkte in diesem Bereich berechnen.

Einen anderen Weg zu PI stellen Reihen dar. Eine einfache Reihe für diesen Zweck ist die Leibniz-Reihe. Diese konvergiert, je weiter man sie berechnet, immer mehr gegen PI/4.»

Die Aufgabe besteht darin, die Leibniz-Reihe in einem Task zu berechnen und in einem zweiten Task einen anderen, selbst gewählten Algorithmus zu implementieren. Diese Tasks werden von einem Steuertask koordiniert.

1. Der aktuelle Wert von Pi soll stetig auf dem Display des EduBoard angezeigt werden. Die Anzeige muss alle 500ms aktualisiert werden.
2. Die Algorithmen sollen durch Tastendruck gestartet, gestoppt und zurückgesetzt werden können. Ausserdem soll eine vierte Taste verwendet werden, um zwischen den beiden Algorithmen umzuschalten.

Die Kommunikation zwischen den Tasks kann entweder durch EventBits oder TaskNotifications erfolgen.

Es sind mindestens drei Tasks erforderlich: ein Task für das Buttonhandling und die Steuerung des Displays, ein Task für die Berechnung mit der Leibniz-Folge und ein Task für den dritten Algorithmus.

Das Programm soll ausserdem mit einer Zeitmessfunktion ausgestattet werden, um die Zeit zu messen, bis Pi auf fünf Nachkommastellen genau berechnet wurde. Hierzu wird xTaskGetTickCount verwendet. Diese Zeit wird dann auf dem Display angezeigt, während die Berechnung von Pi fortgesetzt wird. Die Zeit wird durchgehend aktualisiert.

# Algorithmen

## Leibniz-Reihe

Die Leibniz-Reihe ist eine Formel (Formel 1) zur Annäherung an die Kreiszahl Pi, die von Gottfried Wilhelm Leibniz zwischen 1673 und 1676 entwickelt wurde und 1682 in der Zeitschrift [Acta Eruditorum](https://de.wikipedia.org/wiki/Acta_Eruditorum) erstmals veröffentlichte. Die Folmel lautet (1):

Formel 1: Leibniz-Folge

## Nilakantha-Reihe

Die Nilakantha-Reihe ist eine verbesserte und schnellere Formel (Formel 2) zur Annäherung an die Kreiszahl Pi. Sie geht auf den indischen Mathematiker und Astronomen Kelallur Nilakantha Somayaji zurück. Witzigerweise berechnen die aufsummierten Brüche aber genau die Nachkommstellen von Pi, die 3 läuft gewissermassen vorneweg. Die Formel kann auf verschieden Weise dargestellt werden (2), (3):

Formel 2: Nilakantha-Folge

## Entscheid

Ich habe mich für den Nilakantha-Algorithmus entschieden, weil er sich als eine einfache und effiziente Methode zur Berechnung von Pi in meinem C-Code erwies. Die Gründe dafür sind vielfältig.

Zunächst einmal ist die Implementierung des Nilakantha-Algorithmus unkompliziert. Dies bedeutet, dass ich keine komplexen mathematischen Operationen oder Funktionen in meinen Code integrieren musste. Das hat die Entwicklung meines Codes erheblich erleichtert.

Darüber hinaus zeichnet sich der Nilakantha-Algorithmus durch seine schnelle Konvergenz aus. Bereits nach wenigen Iterationen kann man äußerst präzise Näherungen von Pi erhalten. Dies ist darum besonders von Vorteil, da die Rechenleistung begrenzt ist.

Ein weiterer Pluspunkt ist der geringe Speicherbedarf des Algorithmus. Im Vergleich zu anderen Pi-Berechnungsmethoden benötigt der Nilakantha-Algorithmus weniger Speicherplatz, da er keine umfangreichen Tabellen oder Arrays speichert. Dies ist in Ressourcenbeschränkten Umgebungen besonders wichtig.

# Programm

## Aufbau

Das Programm basiert auf einer Zustandsmaschine (Finite State Machine) und zeigt immer das aktuelle Menü an, in dem sich der Benutzer gerade befindet.

Zunächst wird ein Startbildschirm angezeigt, auf dem der Benutzer zwischen verschiedenen Untermenüs auswählen kann. Die Auswahl erfolgt über die Tasten des EduBoards.

Im ersten Untermenü wird Pi mithilfe der math.h-Bibliothek angezeigt. Mit einem weiteren Tastendruck kann der Benutzer zum Startbildschirm zurückkehren.

Die zweiten und dritten Untermenüs führen zur Leibniz- bzw. Nilakantha-Folge. Diese sind grundsätzlich gleich aufgebaut. Sie zeigen an, mit welchen Tasten die Berechnung gestartet, gestoppt, zurückgesetzt und der Algorithmus gewechselt werden kann. In der ersten Zeile wird die aktuell konvergierte Zahl sowie die bis dahin vergangene Zeit angezeigt. In der zweiten Zeile wird angezeigt, wie viel Zeit seit dem Starten vergangen ist, bis Pi auf fünf Nachkommastellen genau berechnet wurde.

## Tasks

Das gesamte Programm ist in vier Tasks unterteilt. Ein Task ist für die Ausgabe auf dem Display und die Steuerung der Berechnungstasks zuständig. Ein weiterer Task kümmert sich ausschliesslich um das Buttonhandling, und die anderen beiden sind jeweils für einen Algorithmus und das Timing verantwortlich.

## Controller-/Interface-Task

Der Controller-/Interface-Task ist grundsätzlich in verschiedene Teile unterteilt. Ein Teil ist für die Anzeige verantwortlich, ein anderer Teil für die Menüauswahl, ein weiterer Teil für die Auswahl des ‘math.h’-Menüs und je ein Teil für die Steuerung der beiden Algorithmen.

### Benutzeroberfläche

Der Code ist Teil eines wiederholten Prozesses, der die Anzeige auf dem Display aktualisiert. Die Aktualisierung erfolgt in regelmäßigen Zeitabständen (alle 500 ms) und wird durch die Variable DisplayUpdateCounter gesteuert.

Die Benutzeroberfläche hat mehrere Menüs, und je nachdem, welches Menü ausgewählt ist (durch die Variable Menu), werden verschiedene Informationen auf dem Display angezeigt.

Hier sind die Hauptteile des Codes:

Menu\_Main: Dies ist der Startbildschirm. Es zeigt den Titel "Pi-Calculator" und die Optionen "1: Pi aus math.h", "2: Leibniz-Serie" und "3: Nilakantha-Serie" an.

Menu\_Demo: Dieses Menü zeigt den Wert von Pi aus der math.h-Bibliothek mit acht Dezimalstellen an. Es gibt eine Option "4: Back", um zum Startbildschirm zurückzukehren.

Menu\_Leibniz: In diesem Menü haben Sie verschiedene Optionen, abhängig von den Werten in der Variable Bits. Wenn die Berechnung gestartet wurde (durch Drücken von "1: Start"), werden der aktuelle Wert von Pi, die Zeit und andere Informationen auf dem Display angezeigt. Es gibt auch Optionen zum Anhalten und Zurücksetzen der Berechnung.

Menu\_Nilakantha: Ähnlich wie im Menu\_Leibniz gibt es auch hier verschiedene Optionen, abhängig von den Werten in Bits. Dieses Menü ist für die Nilakantha-Serie vorgesehen, mit Start-, Stop-, und Zurücksetzen-Optionen.

Der Code verwendet sprintf, um den Wert von Pi aus der math.h-Bibliothek in einem String zu formatieren und auf dem Display anzuzeigen.

Die Variable DisplayUpdateCounter wird verwendet, um sicherzustellen, dass das Display alle 500 ms aktualisiert wird, und sie wird in jedem Schleifendurchlauf verringert.

Insgesamt handelt es sich um eine Benutzeroberfläche, die dem Benutzer ermöglicht, zwischen verschiedenen Pi-Berechnungsmethoden zu wählen und den Fortschritt der Berechnungen auf einem Display anzuzeigen. Je nach ausgewähltem Menü und Status werden verschiedene Informationen angezeigt.

### Menüauswahl

Die Menüauswahl ist sehr einfach gehalten. Die Auswahl erfolgt anhand des ‘ButtonState’-Werts, der von ein ‘switch-case’ ausgewertet wird, um in das entsprechende Menü zu wechseln. Die Menüauswahl steht nur zur Verfügung, wenn sich der Benutzer im Startbildschirm befindet.

Menüüberprüfung:

Zuerst wird überprüft, ob das aktuelle Menü Menu das Hauptmenü (Menu\_Main) ist.

Schalterabfrage:

Der Code überprüft den aktuellen ButtonState, der die Statusinformationen der Schaltflächen enthält.

Fallunterscheidung basierend auf den Schaltflächen:

Fall 1 (ButtonState = 1): Wenn die Taste 1 gedrückt wird, wird das Menü auf das Demo-Menü (Menu\_Demo) umgeschaltet. Dies ermöglicht dem Benutzer, die Pi-Demo anzuzeigen.

Fall 2 (ButtonState = 2): Wenn die Taste 2 gedrückt wird, wird das Menü auf das Leibniz-Menü (Menu\_Leibniz) umgeschaltet. Hier kann der Benutzer die Leibniz'sche Pi-Berechnung durchführen.

Fall 4 (ButtonState = 4): Wenn die Taste 4 gedrückt wird, wird das Menü auf das Nilakantha-Menü (Menu\_Nilakantha) umgeschaltet. Dies ermöglicht dem Benutzer die Nilakantha'sche Pi-Berechnung.

Zusammenfassung:

Dieser Codeabschnitt ermöglicht die Navigation zwischen den Hauptmenüs auf der Grundlage der vom Benutzer gedrückten Tasten. Je nach Auswahl wird das Menü entsprechend aktualisiert, um die gewünschten Funktionen und Berechnungen anzuzeigen. Dies trägt zur Benutzerinteraktion und -steuerung des Pi-Calculators bei.

### Math.h Demo

Obwohl das 'math.h'-Demo nicht in der ursprünglichen Aufgabenstellung vorgesehen war, stellt es eine informative Ergänzung dar. In diesem Menü wird ausschließlich Pi aus der 'math.h'-Bibliothek angezeigt, ohne auf komplexe Berechnungsalgorithmen zurückgreifen zu müssen.

Von dort aus kann nur wieder zum Startbildschirm zurückgewechselt werden.

### Leibniz’s Pi

Menüüberprüfung:

Zuerst wird überprüft, ob das aktuelle Menü Menu das Leibniz-Menü ist.

Schalterabfrage:

Der Code überprüft den aktuellen ButtonState, der die Statusinformationen der Schaltflächen enthält.

Fallunterscheidung basierend auf den Schaltflächen:

Fall 1 (ButtonState = 1): Wenn die Taste 1 gedrückt wird, wird überprüft, ob Bits (ein Statusbit) den Wert 0 hat. Dies deutet darauf hin, dass der Benutzer die Berechnung starten möchte. In diesem Fall werden entsprechende Statusbits gesetzt, und die Leibniz-Berechnung wird gestartet. Wenn Bits andere bestimmte Werte hat (z. B. nach dem Wechsel von Nilakantha's Pi), wird die Berechnung ebenfalls gestartet oder fortgesetzt.

Fall 2 (ButtonState = 2): Wenn die Taste 2 gedrückt wird, wird überprüft, ob die Berechnung bereits läuft. In diesem Fall wird die Berechnung gestoppt, indem die entsprechenden Statusbits gelöscht werden.

Fall 4 (ButtonState = 4): Wenn die Taste 4 gedrückt wird, wird die Leibniz-Berechnung zurückgesetzt. Dies bedeutet, dass alle zugehörigen Statusbits gelöscht werden und die Anzeige auf dem Display gelöscht wird.

Fall 8 (ButtonState = 8): Wenn die Taste 8 gedrückt wird, wird die Leibniz-Berechnung gestoppt, und das Menü wird auf Nilakantha's Pi umgeschaltet (Menu = Menu\_Nilakantha). Dies verhindert ein sofortiges Zurückwechseln zwischen den Menüs.

Zusammenfassung:

Dieser Codeabschnitt ermöglicht die Steuerung der Leibniz'schen Pi-Berechnung, einschließlich Starten, Stoppen und Zurücksetzen. Er sorgt auch dafür, dass zwischen den Menüs hin- und hergewechselt werden kann, wenn der Benutzer dies wünscht. Die Verarbeitung erfolgt durch das Festlegen und Löschen von Statusbits und das Anzeigen von Informationen auf dem Display.

### Nilakantha’s Pi

Menüüberprüfung:

Zuerst wird überprüft, ob das aktuelle Menü Menu das Nilakantha-Pi-Menü (Menu\_Nilakantha) ist.

Schalterabfrage:

Der Code überprüft den aktuellen ButtonState, der die Statusinformationen der Schaltflächen enthält.

Fallunterscheidung basierend auf den Schaltflächen:

Fall 1 (ButtonState = 1):

Wenn die Taste 1 gedrückt wird und Bits (ein Statuswert) 0 ist, bedeutet dies, dass es der erste Start seit dem Hauptbildschirm ist. In diesem Fall werden die Startereignisse für die Nilakantha-Berechnung initialisiert.

Fall 2 (ButtonState = 2):

Wenn die Taste 2 gedrückt wird und die Nilakantha-Berechnung bereits läuft oder die Leibniz-Berechnung gestoppt ist, wird die Nilakantha-Berechnung gestoppt.

Fall 4 (ButtonState = 4):

Wenn die Taste 4 gedrückt wird und die Nilakantha-Berechnung bereits läuft oder gestoppt wurde, wird die Nilakantha-Berechnung zurückgesetzt (auf 0). Gleichzeitig werden die Anzeigetexte für die Nilakantha-Berechnung gelöscht.

Fall 8 (ButtonState = 8):

Wenn die Taste 8 gedrückt wird, wird die Nilakantha-Berechnung gestoppt, und das Menü wird auf das Leibniz-Pi-Menü (Menu\_Leibniz) umgeschaltet. Dabei wird sichergestellt, dass der Benutzer nicht unmittelbar nach dem Umschalten zurückkehren kann.

Zusammenfassung:

Dieser Codeabschnitt ermöglicht dem Benutzer, zwischen verschiedenen Aktionen im Nilakantha-Pi-Menü zu wählen. Dies beinhaltet den Starten, Stoppen und Zurücksetzen der Nilakantha-Berechnung sowie den Wechsel zum Leibniz-Pi-Menü.

## Buttontask

Initialisierung der Schaltflächen:

Die Funktion initButtons() initialisiert die Schaltflächen, um sicherzustellen, dass sie korrekt erkannt und verarbeitet werden.

Schleife für die Schaltflächenverarbeitung:

Es wird eine Endlosschleife (for(;;)) verwendet, um kontinuierlich auf Schaltflächenereignisse zu warten und sie zu verarbeiten.

Aktualisierung der Schaltflächenzustände:

Die Funktion updateButtons() wird aufgerufen, um die aktuellen Zustände der Schaltflächen zu erfassen.

Lesen des Schaltflächenzustands und Setzen von Event-Bits in der Event-Gruppe:

Der Code überprüft den Zustand jeder Schaltfläche, um festzustellen, ob sie kurz (SHORT\_PRESSED) oder lange (LONG\_PRESSED) gedrückt wurde.

Abhängig vom Zustand der Schaltflächen werden entsprechende Event-Bits in der Event-Gruppe (evButtonEvents) gesetzt. Dies ermöglicht es anderen Teilen des Programms, auf Schaltflächenaktionen zu reagieren.

Verzögerung für die Schaltflächenaktualisierung:

Nachdem die Schaltflächen verarbeitet wurden, gibt es eine Verzögerung, um die Schaltflächen nicht zu häufig zu überprüfen. Diese Verzögerung basiert auf BUTTON\_UPDATE\_FREQUENCY\_HZ und wird mit vTaskDelay implementiert.

Zusammenfassung:

Dieser Codeabschnitt ist für die Erfassung und Verarbeitung von Schaltflächenereignissen verantwortlich. Er prüft die Zustände der Schaltflächen, erkennt kurze und lange Drücke und setzt entsprechende Event-Bits in der Event-Gruppe, um auf Schaltflächenaktionen reagieren zu können.

## Leibniz-Folge-Task

Lokale Variablen und Event-Erwartung:

Es werden lokale Variablen für die Berechnung initialisiert, darunter PI und Summe.

Die Funktion erwartet ein Event (EV\_START\_LEIBNIZ), um mit der Berechnung zu beginnen.

Einige Variablen für die Zeiterfassung werden initialisiert.

Berechnung der Leibniz-Reihe:

Die Berechnung erfolgt in einer Schleife, die mit n-Schritten durchgeführt wird.

Die Leibniz-Reihe wird zur Berechnung von Pi verwendet, und PI wird aktualisiert.

Die Schleife enthält auch die Zeiterfassung, um festzustellen, wie lange die Berechnung gedauert hat.

Event-Verarbeitung:

Während der Schleife wird auf Events geprüft, die den Berechnungsprozess steuern.

Wenn das Event EV\_STOP\_LEIBNIZ ausgelöst wird, wird die Berechnung gestoppt, die Zeit gemessen, und das Event EV\_STOPPED\_LEIBNIZ wird ausgelöst.

Wenn das Event EV\_RESET\_LEIBNIZ ausgelöst wird, wird die Berechnung gestoppt und zurückgesetzt.

Wenn Pi auf 5 Dezimalstellen genau ist, wird die genaue Berechnungszeit aufgezeichnet und ein Codeblocker verhindert, dass dies mehrmals geschieht.

Schleifenführung und Wiederaufnahme:

Die Schleife läuft fort, bis n den vorgegebenen Wert erreicht hat.

Die Berechnung kann gestoppt oder zurückgesetzt werden, und die Schleife wird fortgesetzt, falls erforderlich.

Insgesamt handelt es sich bei dieser Funktion um eine Implementierung der Leibniz-Reihe zur Berechnung von Pi, die auf Events reagiert, um gesteuert und überwacht zu werden. Die Berechnungszeit und das Ergebnis werden in Strings geschrieben und aktualisiert, um sie auf einem Display anzuzeigen.

## Nilakantha-Folge-Taks

Lokale Variablen und Event-Erwartung:

Es werden lokale Variablen für die Berechnung initialisiert, darunter PI und Zaehler.

Die Funktion erwartet ein Event (EV\_START\_NILA), um mit der Berechnung zu beginnen.

Einige Variablen für die Zeiterfassung werden initialisiert.

Berechnung des Nilakantha-Verfahrens:

Die Berechnung erfolgt in einer Endlosschleife, in der die Nilakantha-Reihe Schritt für Schritt berechnet wird.

Das Verfahren beginnt mit einem Startwert für Pi und ändert ihn schrittweise.

Die Schleife enthält auch die Zeiterfassung, um festzustellen, wie lange die Berechnung gedauert hat.

Event-Verarbeitung:

Während der Schleife wird auf Events geprüft, die den Berechnungsprozess steuern.

Wenn das Event EV\_STOP\_NILA ausgelöst wird, wird die Berechnung gestoppt und die Zeit gemessen, und das Event EV\_STOPPED\_NILA wird ausgelöst.

Wenn das Event EV\_RESET\_NILA ausgelöst wird, wird die Berechnung gestoppt und zurückgesetzt.

Schleifenführung und Wiederaufnahme:

Die Schleife läuft fort, bis die Berechnung gestoppt oder zurückgesetzt wird.

Die Berechnung kann gestoppt oder zurückgesetzt werden, und die Schleife wird fortgesetzt, falls erforderlich.

Insgesamt handelt es sich bei dieser Funktion um eine Implementierung des Nilakantha-Verfahrens zur Berechnung von Pi, die auf Events reagiert, um gesteuert und überwacht zu werden. Die Berechnungszeit und das Ergebnis werden in Strings geschrieben und aktualisiert, um sie auf einem Display anzuzeigen.

# Messbericht Thema – 3

## Kapitel Aufgabenstellung

## Kapitel Theorie

Initialisierung und Aufbau des Interfaces:

Zuerst wird das Display gelöscht und dann eine Pause von 500 ms eingelegt, um sicherzustellen, dass der Benutzer alles sehen kann.

Die Funktion initButtons() wird aufgerufen, um die Schaltflächen (Buttons) zu initialisieren.

DisplayUpdateCounter wird auf 50 gesetzt, um sicherzustellen, dass das Display alle 500 ms aktualisiert wird.

Benutzeroberfläche anzeigen:

Der Code beginnt mit der Überprüfung von DisplayUpdateCounter. Wenn er null ist, wird die Benutzeroberfläche auf dem Display aktualisiert.

Die Benutzeroberfläche hängt von der aktuellen Auswahl (Menu) ab. Es gibt verschiedene Menüs, die auf dem Display angezeigt werden:

Menu\_Main: Dies ist der Startbildschirm, der dem Benutzer die Auswahl zwischen verschiedenen Berechnungsmethoden bietet, einschließlich Pi aus math.h, Leibniz-Serie und Nilakantha-Serie.

Menu\_Demo: In diesem Menü wird der Wert von Pi aus der math.h-Bibliothek angezeigt. Der Benutzer kann zur vorherigen Ansicht ("Back") zurückkehren.

Menu\_Leibniz: Dieses Menü ist für die Leibniz-Pi-Berechnung vorgesehen. Je nach Status (Bits) des Algorithmus werden verschiedene Optionen angezeigt, einschließlich "Start", "Stop", "Reset" und "Zu Nilakantha wechseln". Der Fortschritt der Berechnung wird ebenfalls auf dem Display angezeigt.

Menu\_Nilakantha: Dieses Menü ist für die Nilakantha-Pi-Berechnung konzipiert und ähnelt dem Leibniz-Menü. Es zeigt Optionen zum Starten, Anhalten, Zurücksetzen und Wechseln zum Leibniz-Menü an.

Menüauswahl:

Nachdem die Benutzeroberfläche angezeigt wurde, wird überprüft, ob der Benutzer auf eine der Schaltflächen gedrückt hat.

Wenn sich der Benutzer im Menu\_Main befindet, kann er zwischen den verschiedenen Berechnungsmethoden auswählen.

Pi Demo-Funktionen:

Im Menu\_Demo kann der Benutzer mit der Schaltfläche "Back" zum Startbildschirm zurückkehren.

Leibniz-Pi-Funktionen:

Im Menu\_Leibniz stehen Optionen zum Starten, Anhalten, Zurücksetzen und Wechseln zum Nilakantha-Menü zur Verfügung.

Die Berechnung wird gestartet, angehalten und wieder aufgenommen, je nach Status.

Nilakantha-Pi-Funktionen:

Im Menu\_Nilakantha gibt es ähnliche Optionen wie im Leibniz-Menü, um die Nilakantha-Pi-Berechnung zu steuern.

Die Berechnung kann gestartet, angehalten, zurückgesetzt und zu Leibniz gewechselt werden.

Der Haupt-Task wird alle 10 ms für 10 ms pausiert, um die Ausführung zu verlangsamen und das System nicht zu überlasten.

Insgesamt handelt es sich um eine Benutzeroberfläche für die Auswahl und Steuerung von Pi-Berechnungsalgorithmen. Je nach Auswahl des Benutzers werden Informationen auf dem Display angezeigt, und die Berechnungen werden gestartet, gestoppt oder zurückgesetzt.

# Persönliches Fazit

Die Aufgabenstellung erschien mir anfangs nicht all zu kompliziert. Da ich aber keine Ahnung von Programmieren hatte, speziell nicht von Embedded Systems, erwies sich das aber als Trugschluss.

# Literaturverzeichnis

1. **Wikipeida.org. [Online] [Zitat vom: 21. 10 2023.] https://de.wikipedia.org/wiki/Leibniz-Reihe.**

**2. opengenus.org. [Online] [Zitat vom: 21. 10 2023.] https://iq.opengenus.org/different-ways-to-calculate-pi/.**

**3. 3.141592653589793238462643383279502884197169399375105820974944592.eu. [Online] [Zitat vom: 25. 10 2023.] https://3.141592653589793238462643383279502884197169399375105820974944592.eu/pi-berechnen-formeln-und-algorithmen/.**

# Abbildungsverzeichnis

Text

# Anhang

Das Dokument wurde auf Grammatik- und Rechtschreibefehler durch ChatGPT überprüft.