Embedded Systems

Calculating Pi

Von **Erik Haubrich**

**Inhaltsverzeichnis:**

[Aufgabenstellung 3](#_Toc149234562)

[1 Algorithmen 4](#_Toc149234563)

[1.1 Leibniz-Reihe 4](#_Toc149234564)

[1.2 Nilakantha-Reihe 4](#_Toc149234565)

[1.3 Entscheid 4](#_Toc149234566)

[2 Programm 5](#_Toc149234567)

[2.1 Aufbau 5](#_Toc149234568)

[2.2 Tasks 5](#_Toc149234569)

[2.3 Controller-/Interface-Task 5](#_Toc149234570)

[2.3.1 Benutzeroberfläche 5](#_Toc149234571)

[2.3.2 Menüauswahl 6](#_Toc149234572)

[2.3.3 Math.h Demo 6](#_Toc149234573)

[2.3.4 Leibniz’s Pi 7](#_Toc149234574)

[2.3.5 Nilakantha’s Pi 7](#_Toc149234575)

[2.4 Buttontask 8](#_Toc149234576)

[2.5 Leibniz-Folge-Task 9](#_Toc149234577)

[2.6 Nilakantha-Folge-Taks 9](#_Toc149234578)

[3 Zeitmessung 11](#_Toc149234579)

[3.1 Kapitel Aufgabenstellung 11](#_Toc149234580)

[4 Prozessorleistung 11](#_Toc149234581)

[4.1 Kapitel Aufgabenstellung 11](#_Toc149234582)

[4.2 Kapitel Theorie 11](#_Toc149234583)

[Persönliches Fazit 13](#_Toc149234584)

[Literaturverzeichnis 14](#_Toc149234585)

[Abbildungsverzeichnis 14](#_Toc149234586)

[Anhang 14](#_Toc149234587)

# 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 Event-Bits 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 : 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 : 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

Dieser Teil des Tasks ist dafür da, die entsprechende Informationen auf das Display zu schreiben. Es wird durch die Variable ‘DisplayUpdateCounter’ in regelmässigen Zeitabständen (alle 500ms) aktualisiert. Das funktioniert, indem der gesamte Task alle 10ms aufgerufen wird und das ‘DisplayUpdateCounter’ jeweils um eins dekrementiert wird, und das 50-mal.

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

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

Pi aus math.h: Dieses Menü zeigt den Wert von Pi aus der math.h-Bibliothek. Es gibt eine Option "4: Back", um zum Startbildschirm zurückzukehren.

Leibniz-Serie: In diesem Menü werden immer "1: Start", "2: Stop", "3: Reset", und "4: 🡪 Nilakantha" angezeigt. Sobald die Berechnung gestartet wurde, werden der aktuelle Wert von Pi sowie die bis dahin benötigte Zeit und auf dem Display angezeigt. Sobald Pi die Genauigkeit erreicht, wird auch die dafür benötigte Zeit angezeigt.

Nilakantha-Serie: Das Menü für die Nilakantha-Serie folgt dem gleichen Aufbau, enthält jedoch den Unterschied, dass stattdessen "4: 🡪 Leibniz" angezeigt wird.

Sobald ein Algorithmus gestartet wurde und die Anzeige aktualisiert wird, wird ein Event-Bit gesetzt. Dieses Bit signalisiert dem Algorithmus, dass er den aktuellen Pi-Wert und die bisher benötigte Zeit in separate Strings schreiben soll. Die Anzeige wartet, bis diese Daten aktualisiert sind. Nachdem die Strings erfolgreich überschrieben wurden, wird das Event-Bit zurückgesetzt, und die Anzeige zeigt den exakten Wert der Pi-Konvergenz und die Zeit auf dem Display an (Abbildung 1).

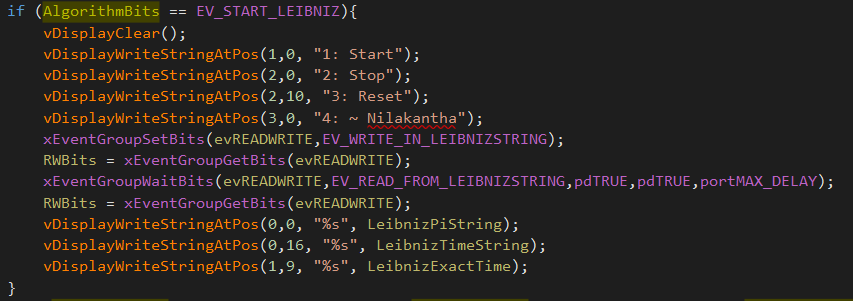


Abbildung : Warten auf Event-Bit für String

Falls die Berechnung gestoppt wird, wird der zuletzt angezeigte Wert aus den Strings beibehalten und auf dem Display angezeigt. Wenn ein Reset durchgeführt wird, werden vorübergehend Leerzeichen in die Strings geschrieben, und das Display wird gelöscht.

### Menüauswahl

Die Menüauswahl ist sehr einfach gehalten. Dieser Codeabschnitt ermöglicht die Navigation zwischen den Hauptmenüs auf der Grundlage der vom Benutzer gedrückten Tasten. Die Auswahl erfolgt anhand des ‘ButtonState’-Werts, der von ein ‘switch-case’ ausgewertet wird, um den entsprechend definierten Wert zu setzen und in das Menü zu wechseln.

Die Menüauswahl steht nur zur Verfügung, wenn sich der Benutzer im Startbildschirm befindet. Nachdem eine der Algorithmen ausgewählt wurde, besteht nur noch die Möglichkeit zwischen diesen beiden hin- und her zu schalten.

### 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’ Pi

Im Leibniz-Menü erfolgt die Steuerung des Algorithmus basierend auf den Tastendrücken. Die Auswahl des jeweiligen Algorithmus-Betriebsmodus erfolgt anhand des Werts "ButtonState". Dieser Wert wird in einem "Switch-Case"-Konstrukt ausgewertet, um den entsprechenden Betriebszustand zu setzen und die damit verbundene Funktion auszuführen.

Wenn die Taste 1 gedrückt wird, soll der Algorithmus gestartet werden. Die Reaktion darauf ist abhängig von der vorherigen Aktivität. Der Code überprüft, ob und welches Bit in der Event-Gruppe "evStartStopEvents" (im Folgenden als Event-Bits bezeichnet) gesetzt ist. Wenn noch kein Algorithmus zuvor ausgeführt wurde, ist kein Event-Bit gesetzt. In diesem Fall wird das Start-Bit gesetzt, und der Algorithmus beginnt seine Berechnung.

Wenn die Berechnung bereits lief und gestoppt wurde oder der Wechsel von der Nilakantha-Serie erfolgte, werden die zuvor gesetzten Event-Bits gelöscht. Dann wird das Start-Bit erneut gesetzt, und die Berechnung wird an der Stelle fortgesetzt, an der sie zuvor unterbrochen wurde.

Im Falle eines Resets wird der Code ebenfalls nur an der unterbrochenen Stelle fortgesetzt. Dieser Einstiegspunkt ist im Leibniz-Task definiert, und die Berechnung setzt dort fort, wo sie zuvor pausiert wurde.

Wird die Taste 2 gedrückt, soll die Berechnung pausiert werden. Dies beinhaltet das Löschen der Event-Bits und das Setzen eines Reset-Bits. Das Stopp-Bit wird im Leibniz-Task ausgewertet, und die Berechnung wird an einer im Task definierten Stelle unterbrochen.

Beim Drücken der Taste 3 erfolgt ein Reset. Event-Bits werden gelöscht und ein Reset-Bit wird gesetzt. Zusätzlich werden Leerzeichen in die Strings geschrieben, um die Anzeige Leer erscheinen zu lassen. Im Leibniz-Task wird das Reset-Bit ausgewertet und entsprechen darauf reagiert, einschliesslich des endgültigen Löschens der Strings.

Wenn die Taste 4 betätigt wird, erfolgt eine Unterbrechung der Berechnung, und es wird zum Nilakantha-Menü gewechselt. Dieser Ablauf wird durchgeführt, indem die Event-Bits gelöscht und ein Stopp-Bit gesetzt werden. Das Menü-Define wird ebenfalls aktualisiert, um das Nilakantha-Menü anzuzeigen. Zusätzlich werden die Event-Bits der Tasteneingaben gelöscht, um zu verhindern, dass mit dem Tastendruck sofort wieder zurückgewechselt wird. Dieser Vorgang sorgt für einen reibungslosen Übergang vom Leibniz- zum Nilakantha-Berechnungsalgorithmus.

### Nilakantha’s Pi

Das Nilakantha-Menü weist die gleiche Struktur auf, und die Steuerung des Nilakantha-Algorithmus funktioniert genauso wie die Steuerung des Leibniz-Algorithmus. Der Hauptunterschied besteht darin, dass die verwendeten "if-Cases" jeweils auf unterschiedliche Bits reagieren, die spezifisch für den Nilakantha-Algorithmus festgelegt sind. Dies ermöglicht es, die Berechnungen für die beiden Algorithmen getrennt zu steuern und sicherzustellen, dass die richtigen Aktionen ausgeführt werden.

## Buttontask

Der Buttontask wurde von einem Template übernommen, das überprüft, welche Taste gedrückt wurde. Dabei wird zwischen kurzen Tastendrücken (< 100ms) und langen Tastendrücken (> 500ms) unterschieden. Im Code wird ausschließlich auf kurze Tastendrücke reagiert.

## Leibniz-Folge-Task

In diesem Task wird die Annäherung von Pi mithilfe der Leibniz-Folge durchgeführt.

Zuerst werden lokale Variablen initialisiert, und der Task wartet auf das Leibniz-Start-Bit (xEventWaitBits). Sobald das Start-Bit gesetzt ist, beginnt der Task und führt den Code aus.

Es wurde ein Wiedereinstiegspunkt für den Reset-Fall festgelegt. Dort werden bestimmte Startwerte in die Variablen geschrieben, und die aktuelle Anzahl Ticks, die für die genaue Zeitbestimmung benötigt wird, erfasst.

Der Task betritt eine 'for'-Schleife, um die Pi-Annäherung durchzuführen. Zuerst wird erneut die aktuelle Anzahl Ticks ausgelesen, um den Sekundenzähler zu aktualisieren zu können.

Während der Schleife wird auf Events geprüft, die den Berechnungsprozess steuern. Es wird überprüft, ob ein Stopp- oder Reset-Bit aktiv ist. Im Fall eines Stopps wird die aktuelle Zeit festgehalten, ein Bit gesetzt, um anzuzeigen, dass die Berechnung gestoppt wurde, und die Start- sowie Reset-Bits werden gelöscht. Der Task wird durch 'vTaskSuspend' angehalten. Wenn währenddessen ein Reset ausgelöst wird, kehrt der Code zum zuvor festgelegten Wiedereinstiegspunkt zurück, wo alle lokalen Variablen auf den Startwert zurückgesetzt werden. Wenn kein Reset erfolgt, wird der Code fortgesetzt, und die aktuelle Tick-Zeit wird erneut aufgenommen, um die Zeitmessung korrekt zu halten.

Im Falle eines Resets werden die Event-Bits gelöscht, das Stopp-Bit gesetzt, die Strings geleert, und der Task wird unterbrochen. Nach der erneuten Aktivierung des Tasks beginnt er erneut am zuvor festgelegten Wiedereinstiegspunkt mit den Startwerten.

Wenn weder ein Stopp- noch Reset-Bit gesetzt ist, wird im Wesentlichen nur der Berechnungsalgorithmus ausgeführt (Abbildung 2).

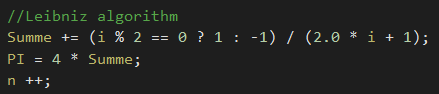


Abbildung : Leibniz Algorithmus

(i % 2 == 0 ? 1 : -1) ist ein Ausdruck, der je nachdem, ob ‘i’ eine gerade oder ungerade Zahl ist, entweder 1 oder -1 ergibt. Dies entscheidet, ob der jeweilige Term in der Reihe addiert oder subtrahiert wird.

(2.0 \* i + 1) ist der Nenner des Bruchs. Hier wird 2 zu ‘i’ multipliziert und dann 1 addiert, um den Nenner für den aktuellen Schleifendurchlauf zu berechnen.

Die Anweisung "Summe +=" ist äquivalent zu "Summe = Summe + Bruch". Dies liegt daran, dass nach jedem Schleifendurchlauf der aktuelle Bruch zum bisherigen Ergebnis der Summe hinzugefügt wird.

Die Berechnung "Pi = 4\*Summe" erfolgt, da das Resultat von 'Summe' in der Leibniz-Folge ein Viertel von Pi ist. In der Leibniz-Folge wird Pi durch die Approximation mithilfe der berechneten Summe angenähert, wobei jedes Summenelement ein Viertel von Pi repräsentiert. Daher wird das Ergebnis mit 4 multipliziert, um die Approximation von Pi zu erhalten.

Anschließend wird 'n' inkrementiert, was für die Wahrheitsprüfung der 'for'-Schleife notwendig ist. 'i' muss nicht explizit inkrementiert werden muss, da dies bereits von der 'for'-Schleife selbst geregelt wird. Die Schleife erhöht 'i' automatisch bei jedem Durchlauf um 1.

Während der Schleife wird auf ein weiteres Event geprüft, und zwar ob der aktuelle Wert von Pi sowie die aktuelle Zeit in die Strings geschrieben werden dürfen. Dieses Event wird vom Controller-/Ineterface-Task ausgelöst. Der Task pausiert solange, bis der Leibniz-Task die werte geschrieben haben und den Task wieder freigibt. Das wird gemacht, damit keine race-conditions entstehen. Auch die vergangene Zeit wird noch um 500ms erhöht, da das die Zeit ist in der Controller-/Interface-Task diese abfrage startet.

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

Schleifenführung und Wiederaufnahme:

## 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.

# Zeitmessung

## Kapitel Aufgabenstellung

# Prozessorleistung

## Kapitel Aufgabenstellung

## Kapitel Theorie

# 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

[Abbildung 1: Warten auf String 6](#_Toc149234588)

# Anhang

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