Juventus
Technikerschule HF
www.technikerschule.ch

Embedded Systems

Calculating Pi

Von **Erik Haubrich**

Inhaltsverzeichnis:

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.
Der aktuelle Wert von Pi soll stetig auf dem Display des EduBoard angezeigt werden. Die Anzeige muss alle 500ms aktualisiert werden.

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.

1 Algorithmen

1.1 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 erstmals veröffentlichte (1). Die Formel lautet:

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \ldots = \frac{\pi}{4}$$

Formel 1: Leibniz-Folge

1.2 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):

$$\pi = 3 + \frac{4}{2 \cdot 3 \cdot 4} - \frac{4}{4 \cdot 5 \cdot 6} + \frac{4}{6 \cdot 7 \cdot 8} - \frac{4}{8 \cdot 9 \cdot 10} + \cdots$$

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

Formel 2: Nilakantha-Folge

1.3 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 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.

2 Programm

2.1 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.

2.2 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.

2.3 Controller-/Interface-Task

Der Controller-/Interface-Task (im Folgenden als UI-Task bezeichnet) 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.

2.3.1 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 und das 'DisplayUpdateCounter' jeweils um eins dekrementiert wird - dies 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 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).

```
if (AlgorithmBits == EV_START_LEIBNIZ){
    vDisplayClear();
    vDisplayWriteStringAtPos(1,0, "1: Start");
    vDisplayWriteStringAtPos(2,0, "2: Stop");
    vDisplayWriteStringAtPos(2,10, "3: Reset");
    vDisplayWriteStringAtPos(3,0, "4: ~ Nilakantha");
    xEventGroupSetBits(evREADWRITE,EV_WRITE_IN_LEIBNIZSTRING);
    RWBits = xEventGroupGetBits(evREADWRITE);
    xEventGroupWaitBits(evREADWRITE,EV_READ_FROM_LEIBNIZSTRING,pdTRUE,pdTRUE,portMAX_DELAY);
    RWBits = xEventGroupGetBits(evREADWRITE);
    vDisplayWriteStringAtPos(0,0, "%s", LeibnizPiString);
    vDisplayWriteStringAtPos(0,16, "%s", LeibnizTimeString);
    vDisplayWriteStringAtPos(1,9, "%s", LeibnizExactTime);
}
```

Abbildung 1: 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.

2.3.2 Menüauswahl

Die Menüauswahl ist sehr einfach gehalten. Dieser Code-Abschnitt ermöglicht die Navigation zwischen den Hauptmenüs auf der Grundlage der vom Benutzer gedrückten Tasten. Die Auswahl erfolgt anhand des 'ButtonState'-Werts, der von einem '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.

2.3.3 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 auf acht Nachkommastellen genau aus der 'math.h'-Bibliothek angezeigt, ohne auf komplexe Berechnungsalgorithmen zurückgreifen zu müssen.

Von dort aus kann nun wieder zum Startbildschirm zurückgewechselt werden.

2.3.4 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.

2.3.5 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.

2.4 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.

2.5 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 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).

```
//Leibniz algorithm
Summe += (i % 2 == 0 ? 1 : -1) / (2.0 * i + 1);
PI = 4 * Summe;
n ++;
```

Abbildung 2: 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, 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, nämlich ob der aktuelle Wert von Pi sowie die aktuelle Zeit in die Strings geschrieben werden dürfen. Dieses Event wird vom UI-Task ausgelöst. Der UI-Task pausiert so lange, bis der Leibniz-Task die Werte in die Strings geschrieben hat und ihn wieder freigibt. Dieses Vorgehen verhindert sogenannte Race Conditions, bei denen mehrere Aufgaben gleichzeitig auf dieselben Speicherbereiche zugreifen und die Daten inkonsistent werden könnten (Abbildung 1). Ausserdem wird die vergangene Zeit um 500ms erhöht, da dies die Zeitspanne ist, in der der UI-Task diese Abfrage startet.

Zu guter Letzt, wenn Pi auf fünf Dezimalstellen genau ist, wird die genaue Berechnungszeit erfasst und in einen String geschrieben. Ein Codeblocker stellt sicher, dass dies nur einmal geschieht. Damit wird vermieden, dass die genaue Berechnungszeit mehrmals erfasst wird.

2.6 Nilakantha-Folge-Task

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

Dieser Task funktioniert sehr ähnlich wie der Leibniz-Task. Auch hier werden zuerst lokale Variablen initialisiert und der Task wartet auf das Nilakantha-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.

Für die erste Kalkulation wird ein 'if'-Case gemacht, erst danach beginnt die 'for'-Schleife. Wie auch im Leibniz-Task gibt es die Stopp- und Reset-Abläufe.

Der Eigentliche Algorithmus setzt sich aus wenigen Zeilen Code (Abbildung 3) zusammen.

```
//Nilakantha Algorithm
Zaehler *= -1;
PI += (Zaehler * 4 / (n * (n + 1) * (n + 2)));
n += 2;
```

Abbildung 3: Nilakantha Algorithmus

Zaehler *= -1 ändert durch diese Operation das Vorzeichen, um die Berechnung entweder zu addieren oder zu subtrahieren, abhängig vom aktuellen Schleifendurchlauf.

Der Ausdruck (Zaehler * 4 / (n * (n + 1) * (n + 2))) entspricht dem aktuellen Term. Durch die Verwendung von PI += wird dieser Term zu Pi addiert, um die Näherung an Pi bei jedem Schleifendurchlauf zu aktualisieren.

Die Inkrementierung von n mit n += 2 erhöht den Wert von n bei jedem Schleifendurchlauf um 2, um sicherzustellen, dass der Term beim nächsten Durchlauf korrekt berechnet wird.

Wie in der Leibniz-Folge wird auch hier überprüft, ob das Event ausgelöst wurde, um den aktuellen Wert von Pi sowie die aktuelle Zeit in die Strings schreiben zu dürfen. Ebenfalls wird die genaue Berechnungszeit erfasst, sobald Pi auf fünf Dezimalstellen genau ist. Diese Zeit wird in einen String geschrieben und dann auf dem Display angezeigt.

3 Zeitmessung

3.1 Allgemein

Die Zeitmessung erfolgt durch die Verwendung von Ticks. Hierbei wird die aktuelle Anzahl der Ticks über die Funktion xTaskGetTickCount ausgelesen und in Variablen vom Typ TickType_t gespeichert. Wenn die vergangene Zeit benötigt wird, erfolgt erneut das Auslesen der Ticks und die Differenz zwischen den Ticks zu Beginn und am Ende der Zeitspanne wird berechnet. Diese Differenz entspricht der vergangenen Zeit in Ticks. Bei Bedarf kann diese Zeitdifferenz in Millisekunden umgerechnet und aufaddiert werden, um die Gesamtzeit zu ermitteln (Abbildung 4). Die Anzahl der Ticks entspricht der Zeit in Millisekunden.

```
Starttime  = xTaskGetTickCount();
//ausgeführter Code
Endtime = xTaskGetTickCount() - Starttime;
Elapsedtime += Endtime;
```

Abbildung 4: TickCount

3.2 Leibniz

Die Zeit, die benötigt wird, um Pi mithilfe der Leibniz-Folge anzunähern, liegt im Bereich von 11'687 ms bis 11'718 ms. Die meisten Messungen ergeben eine Zeit von 11'717 ms, die als die Standardzeit für diesen Code angesehen werden kann.

3.3 Nilakantha

Die Annäherung an Pi mit der Nilakantha-Folge ist wesentlich schneller. Hier liegt die Zeit normalerweise zwischen <1ms und 4ms. In den meisten Fällen beträgt sie 1ms.

4 Prozessorleistung

Die Prozessorauslastung sollte im Falle, dass ein Berechnungs-Task aktiv ist, vergleichsweise hoch sein, da dieser ununterbrochen rechnet. Wenn nur der UI- und der Button-Task aktiv sind, sollte die Auslastung wesentlich geringer sein, da diese nur all Millisekunde beziehungsweise alle 10ms aufgerufen werden.

5 Persönliches Fazit

Anfangs schien die Aufgabenstellung nicht allzu kompliziert. Da ich jedoch keinerlei Erfahrung im Programmieren hatte, insbesondere im Bereich der Embedded Systems, erwies sich diese Annahme als Trugschluss. Das Handhaben der verschiedenen Tasks, das Starten und Stoppen der Aufgaben sowie die unterschiedlichen Ausgangssituationen haben die Aufgabe erheblich verkompliziert. Die eigentliche Berechnung von Pi hingegen ist vergleichsweise einfach und erfordert nur wenige Zeilen Code. Das Handhaben der Zeit hingegen bereitete mir weniger Schwierigkeiten.
Nach geschätzten 100 Stunden des Programmierens habe ich zweifellos eine Menge gelernt, aber es besteht noch erhebliches Verbesserungspotenzial. Während der Dokumentation sind mir einige Fehler im Code aufgefallen. Die meisten von ihnen tragen lediglich zur Lesbarkeit des Codes bei und haben keinen Einfluss auf die Funktionalität. Der Code ist keinesfalls perfekt, erfüllt jedoch seinen beabsichtigten Zweck.
Alle in der Aufgabenstellung gestellten Anforderungen wurden erfüllt.

Literaturverzeichnis

1. **Wikipeida.org.** ***https://books.google.it/books?id=WNUNAQAAIAAJ&pg=PA118#v=onepage&q&f=false.*** **[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. 3.14159265358979323846264338327950288419716939937510582097494459 2.eu. [Online] [Zitat vom: 25. 10 2023.] https://3.14159265358979323846264338327950288419716939937510582097494459 2.eu/pi-berechnen-formeln-und-algorithmen/.

Abbildungsverzeichnis

Anhang

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