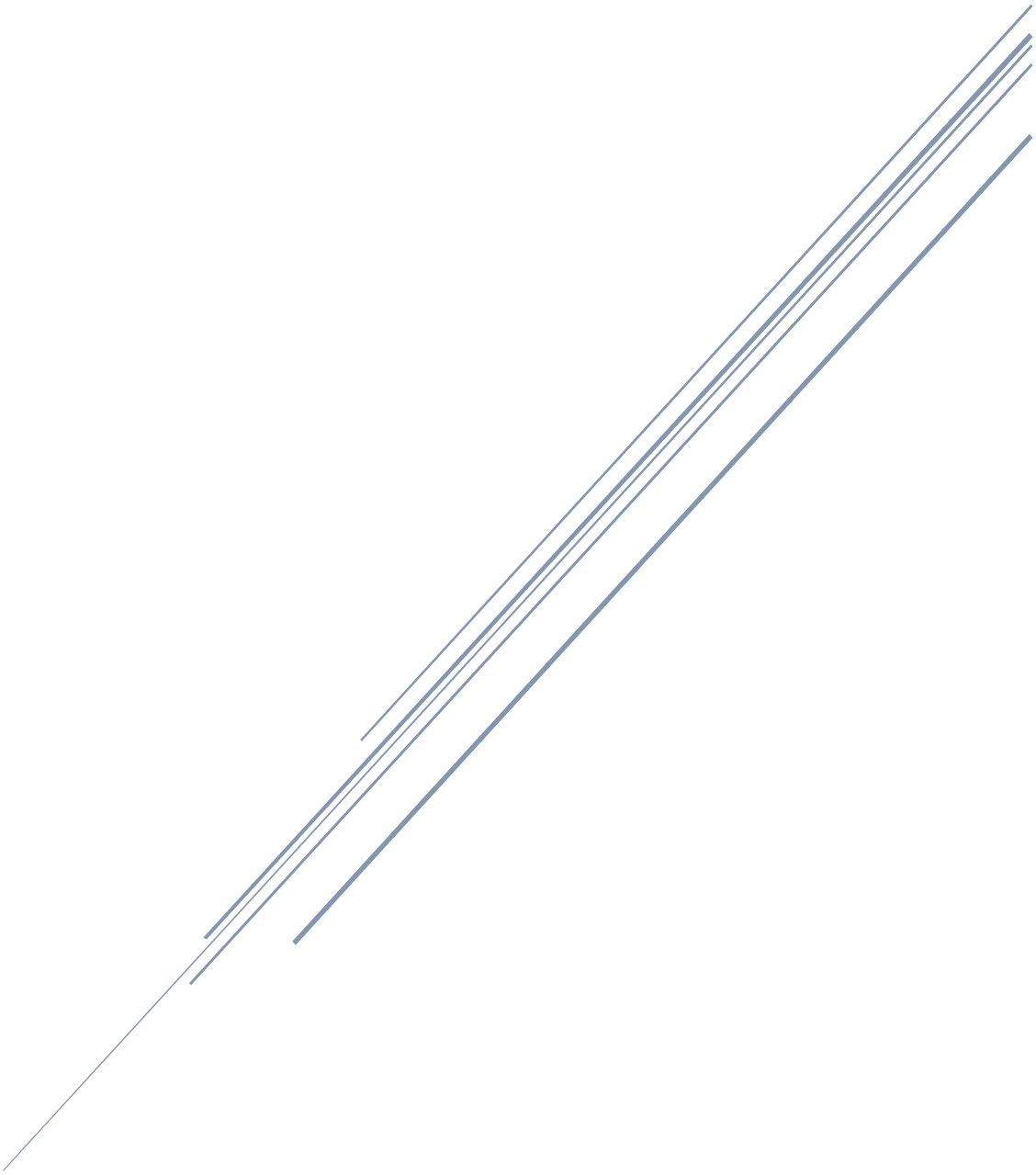
U_PICALC_HS2023

Merlin Unternährer



Juventus Technikerschule HF
Embedded Systems

Inhalt

1. Einleitung

Im Rahmen meines Embedded Systems Kurses wurde mir die faszinierende Aufgabe gestellt, den Wert von PI mithilfe des EDU-Boards und zweier unterschiedlicher Algorithmen zu berechnen. Um dieses Projekt umzusetzen, musste ich das Betriebssystem FreeRTOS verwenden, das mir die Möglichkeit bietet, mehrere Aufgaben gleichzeitig zu organisieren und auszuführen.

Diese Herausforderung eröffnete mir nicht nur die Gelegenheit, mein Wissen im Bereich Embedded Systems zu vertiefen, sondern auch praktische Erfahrungen mit FreeRTOS zu sammeln. In diesem Bericht werde ich genauer auf die Aspekte meines Projekts eingehen, die in der Aufgabenstellung vorgegeben sind. Ich werde die verschiedenen Schritte und Erkenntnisse auf dem Weg zur Berechnung des PI-Werts beschreiben und dabei auch erläutern, wie FreeRTOS in diesem Prozess eine Rolle gespielt hat. Dieses Projekt hat meine Fähigkeiten in der Softwareentwicklung und im Umgang mit Hardware gestärkt, und ich werde dies im weiteren Verlauf ausführlich dokumentieren.

1.1 Aufgabenstellung

Die Aufgabenstellung verlangt die Entwicklung eines Embedded Systems-Programms, das die Berechnung von PI mithilfe von zwei verschiedenen Algorithmen in separaten Tasks ermöglicht. Dabei sind folgende Anforderungen zu erfüllen:

- Es soll ein Task zur Implementierung der Leibniz-Reihen-Berechnung erstellt werden.
- Ein weiterer Task soll zur Implementierung eines Algorithmus dienen, der aus dem Internet ausgewählt wurde.
- Ein Steuertask ist zu erstellen, der die beiden oben genannten Tasks steuert und kontrolliert.

Folgende Funktionalitäten müssen stets gewährleistet sein:

- Der aktuelle Wert von PI soll alle 500 Millisekunden angezeigt werden.
- Der Start des Algorithmus erfolgt durch einen Tastendruck, während ein anderer Tastendruck den Algorithmus stoppt.
- Eine dritte Taste ermöglicht das Zurücksetzen des Algorithmus.
- Eine vierte Taste erlaubt das Umschalten zwischen der Verwendung des Leibniz-Algorithmus und des aus dem Internet ausgewählten Algorithmus.

Das Programm muss mindestens drei Tasks enthalten:

- Ein Interface-Task, der für das Buttonhandling und das Beschreiben des Displays zuständig ist.
- Ein Kalkulations-Task für die Berechnung von PI mithilfe der Leibniz-Reihe.
- Ein weiterer Kalkulations-Task für die Berechnung von PI mithilfe des aus dem Internet ausgewählten Algorithmus.

Zusätzlich soll das Programm eine Zeitmess-Funktion implementieren, die xTaskGetTickCount verwendet, um die Zeit bis zur Genauigkeit von PI auf fünf Stellen hinter dem Komma zu messen. Dabei soll die Zeit auf dem Display angezeigt werden, und sobald die gewünschte Genauigkeit erreicht ist, wird die Zeit für die Berechnung von PI erfasst. Die Berechnung von PI soll trotzdem weitergehen.

2. Algorithmen

2.1 Leibniz-Reihe

Die Leibniz-Reihe ist eine alternierende unendliche Reihe, die zur Näherung der Zahl Pi (π) verwendet wird. Sie lautet:

Formel 1: Leibniz-Reihe

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

In dieser Reihe wechseln sich positive und negative Terme ab, wobei die Nenner der Brüche die ungeraden natürlichen Zahlen sind. Jeder Term in der Reihe wird abwechselnd addiert und subtrahiert.

Die Leibniz-Reihe konvergiert gegen π/4. Das bedeutet, wenn man unendlich viele Terme dieser Reihe summiert, nähert sich die Summe dem Wert von π/4. Je mehr Terme in der Berechnung berücksichtigt werden, desto genauer wird die Näherung an π/4.

Diese mathematische Reihe wurde nach dem deutschen Mathematiker Gottfried Wilhelm Leibniz benannt und ist ein Beispiel für eine alternierende konvergente Reihe.[1]

[1] Leibniz, G. W. (1674). Nova Methodus pro Maximis et Minimis. In G. W. Leibniz, Sämtliche Schriften und Briefe (Vol. 6, pp. 229-259). Akademie Verlag.

2.2 Nilakantha-Methode

Die Nilakantha-Reihe ist eine Methode zur Annäherung an den Wert von π (Pi), die nach dem indischen Mathematiker Nilakantha Somayaji benannt wurde. Er lebte im 15. Jahrhundert und war ein Hauptvertreter der Kerala-Schule der Astronomie und Mathematik.

Die Reihe lautet:

Formel 2: Nilakantha-Reihe

$$\pi = 3 + \sum_{n=1}^{\infty} \frac{4 * (-1)^n}{2n(2n+1)(2n+2)}$$

Hier wird π als die Summe einer unendlichen Reihe von Brüchen dargestellt, wobei der Nenner jedes Bruchs aus einem Produkt von drei aufeinanderfolgenden natürlichen Zahlen besteht.

Um das genauer zu verstehen:

Die Reihe startet mit 3.

Der Summationsausdruck verwendet eine abwechselnde Reihe, was bedeutet, dass die Vorzeichen der Terme sich von positiv zu negativ und umgekehrt ändern. Das erreicht man durch den Faktor (−1)^n. Wenn n gerade ist, ist der Term positiv; wenn n ungerade ist, ist der Term negativ.

Im Nenner sieht man das Produkt von drei aufeinanderfolgenden Zahlen, die erste ist 2n, die zweite 2n+1 und die dritte ist 2n+3.

Einige der ersten Terme dieser Reihe sind:

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

Wenn man diese Terme summiert und 3 hinzufügt, erhaltet man eine Annäherung an π. Je mehr Terme man hinzufügt, desto genauer wird die Annäherung.[2]

[2]Jyesthadeva. (16. Jahrhundert). Yuktibhāṣā. Südwesten Indiens.

3. Erklärung der Tasks

3.1 vControllerTask

Diese Aufgabe überwacht die Tastenereignisse und legt entsprechende Ereignis-Flags fest. Es initialisiert und aktualisiert den Zustand der Tasten und legt Ereignis-Flags für kurze Tastendrücke fest. Diese Ereignisse werden später von der Benutzeroberfläche verwendet, um verschiedene Aktionen wie das Starten/Stoppen der Berechnung oder das Umschalten zwischen den Algorithmen auszulösen.

3.2 vCalculationTaskLeibniz

Diese Aufgabe führt die Pi-Berechnung basierend auf der Leibniz-Reihe aus. Es enthält eine Zustandsmaschine mit den Zuständen RUN (Ausführung) und WAIT (Warten). Im RUN-Zustand berechnet die Aufgabe das nächste Glied der Leibniz-Reihe und aktualisiert die Pi-Näherung. Im WAIT-Zustand setzt die Aufgabe ein Ereignis-Flag, um anzuzeigen, dass sie wartet.

3.3 vCalculationTaskNilakantha

Genau wie die vCalculationTaskLeibniz-Aufgabe, führt diese Aufgabe die Pi-Berechnung basierend auf der Nilakantha-Reihe aus. Dieser Task verwendet ebenfalls eine Zustandsmaschine mit RUN- und WAIT-Zuständen.

3.4 vUi_task

Dieser Task verwaltet die Benutzeroberfläche (UI). Dieser zeigt Informationen über den aktuellen Algorithmus, die berechnete Pi-Näherung und die benötigte Zeit auf einem Display an. Die Aufgabe verwendet eine Zustandsmaschine, um zwischen den verschiedenen UI-Modi Leibniz-Reihe und der Nilakantha-Reihe zu wechseln. Er reagiert auch auf Tastendruckereignisse, die von der vControllerTask-Aufgabe erzeugt werden, um die Pi-Berechnung zu starten/stoppen oder zwischen den Algorithmen zu wechseln

3.4.1 vUi_task Finite State Machine

UIMODE_INIT:

Initialisierungszustand. Der Zweck dieses Zustands ist es, die Zustandsmaschine auf einen Startzustand zu setzen. Hier wird der UI-Modus direkt auf UIMODE_LEIBNIZ_CALC gesetzt. Es wäre möglich hier gewisse Start-Prozesse durchzuführen, in diesem Fall war das nicht nötig.

UIMODE_LEIBNIZ_CALC:

Dieser Zustand zeigt Informationen über die Pi-Berechnung basierend auf der Leibniz-Reihe an. Die Benutzeroberfläche zeigt den aktuellen Wert von Pi, die benötigte Zeit und bietet Optionen, um die Berechnung zu starten, anzuhalten oder zurückzusetzen.

UIMODE_NILAKANTHA_CALC: Dieser Zustand zeigt Informationen über die Pi-Berechnung basierend auf der Nilakantha-Reihe an. Er ähnelt dem Zustand von UIMODE_LEIBNIZ_CALC.

3.5 Taskhandling durch eTaskGetState

Innerhalb des vUi_task, der für die Benutzeroberfläche zuständig ist, wird der Zustand der beiden Berechnungstasks über die eTaskGetState-Funktion überprüft. Dies wird verwendet, um zu bestimmen, welchen Text auf dem Display angezeigt wird, je nachdem, ob der jeweilige Task gerade läuft oder suspendiert ist.

Abhängig vom Zustand der Tasks und den ausgelösten Ereignissen (durch Tastendrücke) können bestimmte Aktionen ausgeführt werden, z. B. das Starten oder Stoppen eines Tasks.

Dadurch wird zusätzlich jegliche Race-Condition zwischen den Berechnungstasks ausgeschlossen, weil es nur einem dieser Tasks erlaubt im laufenden Zustand zu sein.

4. Zeitmessung

Für die Zeitmessung gibt es zwei globale variablen, startTime und endTime, vom Typ TickType_t. Diese werden verwendet, um die Start- und Endzeitpunkte der Berechnung festzuhalten.

Wenn die Berechnung gestartet oder zurückgesetzt wird, wird der aktuelle Zeitstempel aus der Funktion xTaskGetTickCount in die Variable startTime übertragen.

Während der Berechnung überprüft der laufende Rechentask kontinuierlich den Wert von pi_approx. Sobald dieser Wert zwischen 3.14159 und 3.1416 liegt, wird der Zeitstempel aus der Funktion xTaskGetTickCount in die variabel endTime geschrieben.

Die Differenz zwischen endTime und startTime gibt die Anzahl der Ticks während der Berechnung. Um dies in Millisekunden zu konvertieren, wird dieser Wert mit portTICK_PERIOD_MS multipliziert.

4.1 Resultate der Zeitmessung

Leibniz-Reihe:

≈ 9169 ms

Nilakantha-Reihe:

≈ 1 - 4 ms

Auf diese Resultate bezogen stellt sich die Frage: Warum ist die Leibniz-Reihe so viel langsamer? Die Antwort findet sich in der Menge der benötigten Durchläufe, deshalb habe ich die Algorithmen, genau so wie ich diese in meinem Projekt angewendet habe, in CLion simuliert:

Leibniz-Reihe:

135055 Durchläufe

Nilakantha-Reihe:

34 Durchläufe

Man erkennt, dass die Nilakantha-Reihe sehr schnell konvergiert, dagegen konvergiert die Leibniz-Reihe um einiges langsamer.

4.1.1 Auswirkungen auf die Prozessor-Leistung

Macht man sich Gedanken dazu, wie sich die zwei Algorithmen auf die Prozessorleitung auswirken, ist der Faktor Multiplikation enorm wichtig. Schauen wir uns dafür den 34ten Durchlauf der Nilakantha-Reihe an:

$$n = 1$$

$$n = 2 + (33 * 2) = 68$$

$$\frac{4 * (-1)^{68}}{2 * 68 * (2 * 68 + 1) * (2 * 68 + 2)} = \frac{4}{2'571'216} \approx 1.56^{-6}$$

Wir haben also nach nur 34 Durchläufen schon enorm grosse Zahlen. Das führt zu einem sehr schnellen Anstieg der Rechenleistung.

Bei der Leibniz Reihe erreichen wir nach 135055 Durchläufen ein Term mit dem Wert:

$$\frac{1}{2k + 1} = \frac{1}{2 * 135'055 + 1} = \frac{1}{270'111} \approx 3.7^{-6}$$

Der Nenner im Bruch unterscheidet sich enorm, deshalb benötigt auch die Nilakantha-Reihe nach nur 34 Durchläufen schon mehr Prozessorleistung.

Aus Interresse habe ich noch die Anzahl möglicher Durchläufe der Nilakantha-Reihe und der Leibniz-Reihe mit einem 32 Bit Float errechnet:

Kleinstes Value eines Floats beträgt ~ 1.175 x 10^(-38)[3].

Gleichung für die Nilakantha-Reihe:

$$\frac{4}{x} = 1.175^{-38}$$

$$x = \frac{4}{1.175^{-38}} \approx 3.4^{38}$$

Mit der folgenden Ungleichung wird berechnet, ab wann n grösser als der Wert x von oben ist:

$$(2 * n)(2 * n + 1)(2 * n + 2) > \approx 3.4^{38}$$

$$n \approx 3.5^{12}$$

[3] https://www.khronos.org/opengl/wiki/Small_Float_Formats

Gleichung für die Leibniz-Reihe:

$$\frac{1}{x} = 1.175^{-38}$$

$$x = \frac{1}{1.175^{-38}} \approx 8.5^{37}$$

Die Ungleichung, um k zu berechnen:

$$2 * k + 1 > \approx 8.5^{37}$$

$$k \approx 4.25^{37}$$

1. Fazit

Diese Aufgabe bot eine spannende Gelegenheit, sich tiefgehend mit der Programmierung von Embedded Systems auseinanderzusetzen und das Betriebssystem FreeRTOS in der Praxis anzuwenden. Durch die Implementierung der beiden Algorithmen zur Berechnung von PI war es möglich, die theoretischen Kenntnisse im Bereich Embedded Systems in der Praxis anzuwenden und die Feinheiten des Multitaskings in FreeRTOS zu verstehen.

Die Wahl der Algorithmen, insbesondere der Unterschied in ihrer Konvergenzgeschwindigkeit, hat deutlich gemacht, dass nicht alle Methoden zur Annäherung an eine Zahl gleich effizient sind. Während die Leibniz-Reihe trotz ihrer Einfachheit eine hohe Anzahl von Durchläufen benötigte, um eine akzeptable Genauigkeit zu erreichen, zeigte die Nilakantha-Reihe eine bemerkenswerte Schnelligkeit in ihrer Annäherung an PI.

Darüber hinaus verdeutlichte mir das Ausführen dieser Arbeit die Bedeutung von Multitasking in Embedded Systems und die Vorteile sowie Problematiken, die ein Betriebssystem wie FreeRTOS bieten kann. Durch den Einsatz von FreeRTOS konnte ich mehrere Tasks gleichzeitig ausführen und deren Zustände effizient verwalten.

Die Untersuchung der Auswirkungen der Algorithmen auf die Prozessorleistung zeigte zudem die praktischen Herausforderungen bei der Implementierung von Berechnungen in Echtzeit-Systemen auf. Die Tatsache, dass der Prozessor bei der Ausführung der Nilakantha-Reihe stärker beansprucht wurde, obwohl sie schneller konvergierte, unterstreicht die Notwendigkeit, nicht nur die Konvergenzgeschwindigkeit, sondern auch andere Aspekte wie die Rechenkomplexität in Betracht zu ziehen, wenn man sich für einen Algorithmus entscheidet.

Literaturverzeichnis

Jyesthadeva. (16. Jahrhundert). *Yuktibhāṣā.* Südwesten Indiens.

Leibniz, G. W. (1674). *Nova Methodus pro Maximis et Minimis. In G. W. Leibniz, Sämtliche Schriften und Briefe (Vol. 6, pp. 229-259).* Akademie Verlag.