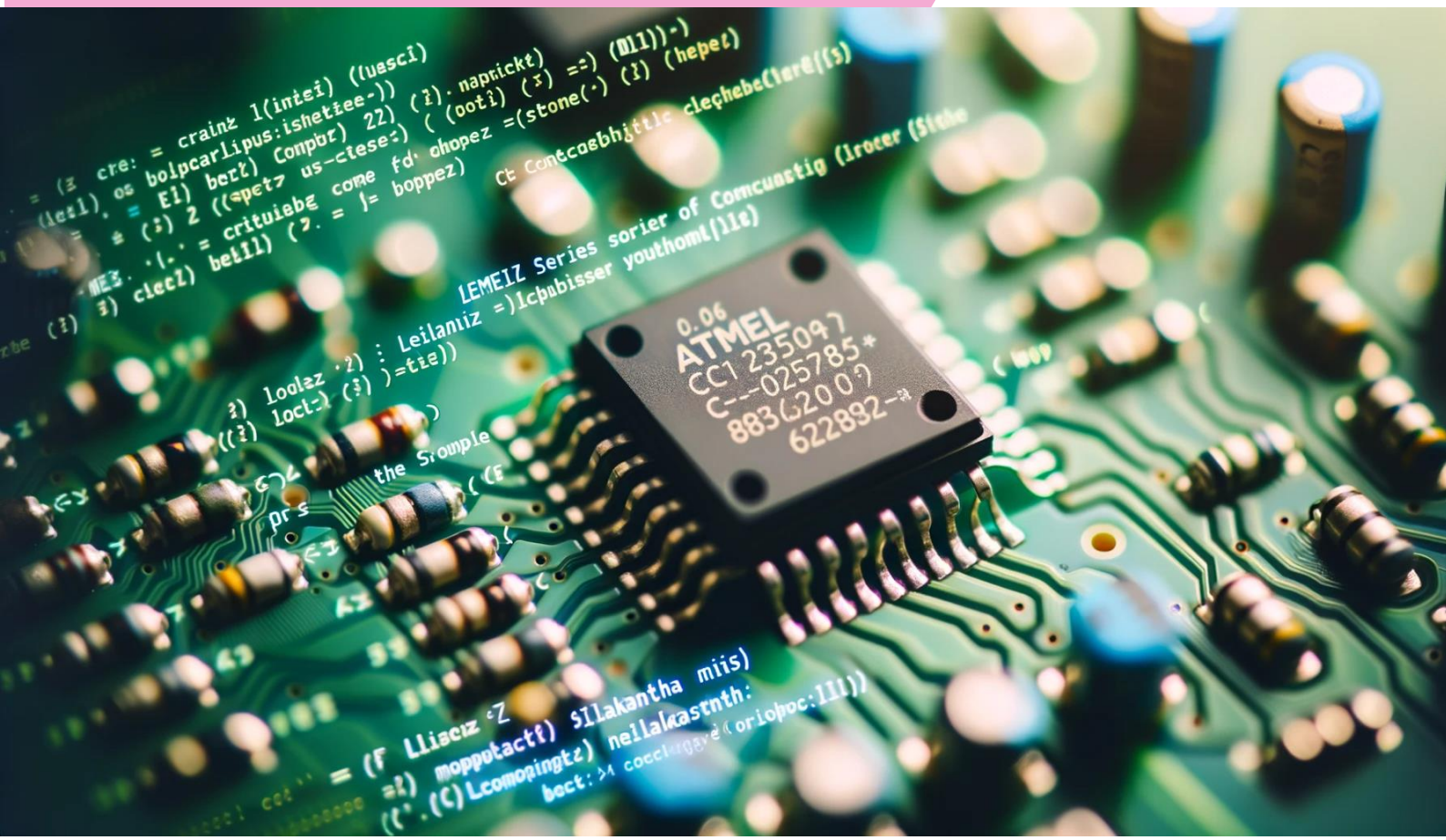
Juventus Schule

Calculating Pi



Marco Müller

E-Mail: marco.mueller1@juventus.schule

Juventus Schulen

INHALT

Einleitung

1.1 Hintergrund des Projekts

Die Berechnung von PI, der Kreiszahl, hat in der Mathematik und in der Geschichte der Wissenschaft eine lange Tradition. Es gibt zahlreiche Algorithmen, um PI zu berechnen, wobei einige schneller sind als andere. Ein einfacher Ansatz basiert auf der geometrischen Interpretation von PI: Wenn man in einem Quadrat mit der Seitenlänge 1 zwei zufällige Punkte wählt und deren Distanz zur linken unteren Ecke berechnet, kann man das Verhältnis der Punkte im Viertelkreis zum gesamten Quadrat nutzen, um ein Viertel von PI zu approximieren. Ein anderer Ansatz zur Berechnung von PI verwendet mathematische Reihen, wie die Leibniz-Reihe, die mit fortlaufender Berechnung gegen PI/4 konvergiert.

1.2 Zielsetzung der Dokumentation

Das Hauptziel dieser Übung ist es, die Berechnung von PI unter Verwendung der Leibniz-Reihe in einem Task zu realisieren und einen weiteren, aus dem Internet ausgewählten Algorithmus zur PI-Berechnung in einem separaten Task zu implementieren. Zusätzlich soll ein Steuertask erstellt werden, der die beiden Berechnungstasks kontrolliert. Während des Betriebs soll der aktuelle Wert von PI alle 500ms aktualisiert und angezeigt werden. Der Benutzer soll die Möglichkeit haben, den Berechnungsprozess mit Tastendrücken zu starten, zu stoppen, zurückzusetzen und zwischen den beiden Algorithmen zu wechseln. Die Kommunikation zwischen den Tasks kann entweder über EventBits oder TaskNotifications erfolgen. Schließlich soll das Programm um eine Zeitmessfunktion erweitert werden, um die benötigte Zeit zu messen, bis PI auf fünf Dezimalstellen genau berechnet ist.

2.1 Leibniz Formel

Der bereitgestellte Code ist Teil einer Methode zur Annäherung des Wertes von π (Pi) mit der Leibniz-Formel. Diese Formel ist eine unendliche Reihenmethode zur Berechnung von π und wird ausgedrückt als:

$$\pi = 4 \cdot (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots)$$

oder allgemeiner:

$$\pi = 4 \cdot \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$ wobei n von 0 bis unendlich reicht[1][2].

Im bereitgestellten Code-Schnipsel:

pythonCopy code

```
pi_approximation_leibniz += (sign / (2 * iterations + 1)) * 4;
```

- pi_approximation_leibniz ist eine Variable, die zur Speicherung der aktuellen Annäherung von π verwendet wird.
- sign ist eine Variable, die mit jeder Iteration zwischen 1 und -1 wechselt, um die wechselnden Vorzeichen in der Formel zu berücksichtigen.
- iterations ist eine Variable, die die aktuelle Iterationsnummer nachverfolgt, die zur Berechnung des Nenners des Bruchs in der Formel verwendet wird.
- Der Ausdruck (sign / (2 * iterations + 1)) * 4 berechnet den nächsten Term der Reihe basierend auf der aktuellen Iterationsnummer und addiert ihn zur aktuellen Annäherung von π.

Die Leibniz-Formel ist bekannt dafür, dass sie sehr langsam gegen π konvergiert. Beispielsweise sind etwa 5 Milliarden Terme erforderlich, um π auf 10 korrekte Dezimalstellen zu berechnen[3].

2.2 Nilkantha-Serie

Die Nilakantha-Serie wurde nach dem indischen Mathematiker und Astronomen Nilakantha (1445–1545) benannt und ist eine Methode zur Berechnung von π (Pi), die durch die folgende Formel gegeben wird:

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

Diese Formel nutzt die Multiplikation dreier aufeinanderfolgender Zahlen im Nenner, beginnend mit 2·3·42·3·4 für den ersten Term nach der 3, und erhöht die ersten zwei Zahlen im Nenner für jeden folgenden Term um 2[12].

Im bereitgestellten Code-Schnipsel:

pythonCopy code

```
pi_approximation_nilkantha += sign * (4.0 / ((2 * iterations + 2) * (2 * iterations + 3) * (2 * iterations + 4)));
```

- **pi_approximation_nilkantha ist eine Variable, die zur Speicherung der aktuellen Annäherung von π verwendet wird.**
- **sign ist eine Variable, die mit jeder Iteration zwischen 1 und -1 wechselt, um die wechselnden Vorzeichen in der Serie zu berücksichtigen.**
- **iterations ist eine Variable, die die aktuelle Iterationsnummer nachverfolgt, welche zur Berechnung des Nenners des Bruchs in der Serie verwendet wird.**
- **Der Ausdruck sign * (4.0 / ((2 * iterations + 2) * (2 * iterations + 3) * (2 * iterations + 4))) berechnet den nächsten Term der Serie basierend auf der aktuellen Iterationsnummer und addiert ihn zur aktuellen Annäherung von π.**

Die Nilakantha-Serie konvergiert schneller gegen π im Vergleich zu anderen Reihen, wie der Leibniz-Formel, und ist daher nützlicher für die Berechnung von Stellen von π[3]. Darüber hinaus gibt es Berichte, dass die Nilakantha-Serie transformiert werden kann, um ihre Konvergenz zu beschleunigen, und dass sie in Beziehung zu anderen mathematischen Formeln und Konzepten steht, wie zum Beispiel der Stirling-Formel[4].

3.1 Übersicht der Tasks

Im vorliegenden Code werden verschiedene Tasks (Aufgaben) definiert, die im Rahmen des FreeRTOS Betriebssystems ausgeführt werden. Diese Tasks umfassen:

1. vControllerTask: Steuert die Ausführung der Pi-Berechnungstasks und die Anzeige basierend auf den Tastendrücken.
2. vPiCalcLeibnizTask: Berechnet die Annäherung an Pi mithilfe der Leibniz-Reihe.
3. vPiCalcNilkanthaTask: Berechnet die Annäherung an Pi mithilfe der Nilkantha-Reihe.
4. vButtonHandler: Verarbeitet Tastendrücke und setzt entsprechende Ereignisbits.

3.2 Ablauf der Tasks

Die Tasks werden vom FreeRTOS Scheduler gesteuert und laufen potenziell unendlich in ihren jeweiligen Schleifen. Der Ablauf und die Steuerung der Tasks erfolgen hauptsächlich über Semaphoren und Ereignisgruppen, die zwischen den Tasks geteilt werden.

3.3 Funktionsweise der einzelnen Tasks

- **vControllerTask:**
  - **Dieser Task liest den Zustand der Buttons über eine Ereignisgruppe und steuert die Ausführung der Pi-Berechnungstasks mithilfe von Semaphoren.**
  - **Er aktualisiert auch die Anzeige, um die aktuelle Pi-Annäherung und die verstrichene Zeit anzuzeigen.**
- **vPiCalcLeibnizTask und vPiCalcNilkanthaTask:**
  - **Diese Tasks berechnen die Annäherung an Pi mithilfe der jeweiligen Reihen.**
  - **Sie prüfen die Semaphoren, um zu bestimmen, ob sie starten, stoppen oder zurücksetzen sollen.**
  - **Sie aktualisieren globale Variablen mit den aktuellen Pi-Annäherungen und überprüfen die Genauigkeit der Berechnung.**
- **vButtonHandler:**
  - **Dieser Task initialisiert die Button-Handler, aktualisiert die Button-Zustände und setzt Ereignisbits in der Ereignisgruppe basierend auf den Tastendrücken.**
  - **Es gibt eine Verzögerung zwischen den Button-Updates, um die CPU-Nutzung zu minimieren.**

4.1 Definition und Grundlagen

In dem bereitgestellten Code werden Semaphoren als Synchronisationsmechanismen verwendet, um den Ablauf verschiedener Tasks zu koordinieren. Sie sind essentielle Elemente in einem Real-Time Operating System (RTOS) wie FreeRTOS, um sicherzustellen, dass Ressourcen korrekt geteilt und Zugriffe darauf synchronisiert werden.

4.2 Aufgabe im Projekt

In diesem Projekt dienen die Semaphoren dazu, den Ablauf der Pi-Berechnungstasks zu steuern. Drei Semaphoren xResetSemaphore, xStartSemaphore, und xStopSemaphore werden definiert, um die Zustände "Start", "Stop" und "Reset" für die Berechnungstasks zu signalisieren. Diese Semaphoren werden in den Tasks vPiCalcLeibnizTask und vPiCalcNilkanthaTask verwendet, um zu prüfen, ob ein bestimmtes Ereignis signalisiert wurde, und dementsprechend die Ausführung zu steuern. In vControllerTask werden die Semaphoren auf Basis der Tastendrücke gegeben.

4.3 Verwendung und Implementierung

Die Semaphoren werden im Code wie folgt implementiert und verwendet:

- Erstellung der Semaphoren:
    - Die Semaphoren werden im main-Funktion durch Aufrufe von xSemaphoreCreateBinary() erstellt.

xResetSemaphore = xSemaphoreCreateBinary(); xStartSemaphore = xSemaphoreCreateBinary(); xStopSemaphore = xSemaphoreCreateBinary();

- Verwendung in den Berechnungstasks:
    - In den Tasks vPiCalcLeibnizTask und vPiCalcNilkanthaTask werden die Semaphoren verwendet, um den Ablauf zu steuern.
    - Zum Beispiel, in vPiCalcLeibnizTask:

Signalisierung der Semaphoren:

- **Die Semaphoren werden in vControllerTask auf Basis der Tastendrücke signalisiert.**

EVBUTTONS_S1: // Start xSemaphoreGive(xStartSemaphore);
EVBUTTONS_S3: // Reset xSemaphoreGive(xResetSemaphore);

Durch diese Implementierung und Verwendung von Semaphoren wird eine klare und effektive Synchronisation und Koordination zwischen den verschiedenen Tasks erreicht, was für die korrekte Funktionalität des Projekts entscheidend ist.

5.1 Methodik der Zeitmessung

Die Zeitmessung in diesem Projekt wird durch die Verwendung der Funktion xTaskGetTickCount() von FreeRTOS ermöglicht. Diese Funktion gibt die Anzahl der Ticks seit dem Start des Schedulers zurück. Bei jedem Start einer Pi-Berechnung wird die aktuelle Tick-Zahl erfasst und als Startzeitpunkt gespeichert. Im Code sind dies die Variablen startTimeLeibniz und startTimeNilkantha.

startTimeLeibniz = xTaskGetTickCount(); // Capture start time ...
startTimeNilkantha = xTaskGetTickCount(); // Capture start time

Die verstrichene Zeit wird dann berechnet, indem die aktuelle Tick-Zahl abgerufen und die Startzeit subtrahiert wird. Dies erfolgt in der vControllerTask:

elapsedTimeLeibniz = xTaskGetTickCount() - startTimeLeibniz; ...
elapsedTimeNilkantha = xTaskGetTickCount() - startTimeNilkantha;

5.2 Darstellung der Ergebnisse

Die Darstellung der Ergebnisse erfolgt über eine Display-Schnittstelle, die durch die Funktionen vDisplayWriteStringAtPos() und vDisplayClear() gesteuert wird. Die aktuelle Pi-Annäherung und die verstrichene Zeit werden auf dem Display dargestellt. Die Formatierung der Ausgabe erfolgt durch die Funktion sprintf, welche die Float-Werte in Zeichenketten konvertiert, die dann auf dem Display angezeigt werden können:

5.3 Interpretation und Analyse

Die Interpretation und Analyse des Codes und der Ergebnisse würde typischerweise darauf abzielen, die Genauigkeit und Effizienz der beiden Pi-Berechnungsmethoden zu bewerten. Durch den Vergleich der verstrichenen Zeit und der erreichten Genauigkeit kann eine Einschätzung darüber getroffen werden, welche Methode schneller konvergiert, oder weniger Rechenzeit benötigt, um eine bestimmte Genauigkeit zu erreichen. Des Weiteren könnte analysiert werden, wie die Systemressourcen während der Ausführung der Tasks genutzt werden, und ob Optimierungen möglich sind, um die Performance zu verbessern.

6.1 Vergleich zwischen Leibniz und Nilkantha

Die Berechnung von Pi mithilfe der Leibniz- und Nilkantha-Formeln erfolgt durch unterschiedliche mathematische Ansätze, die im Code als separate Tasks implementiert sind. Die Leibniz-Reihe konvergiert langsamer zur tatsächlichen Zahl Pi, wie durch die benötigte Zeit und die Anzahl der Iterationen im Code ersichtlich wird. Im Gegensatz dazu stellt die Nilkantha-Methode eine schnellere Konvergenz dar, was in der Regel zu einer schnelleren Berechnung von Pi führt.

```
// Leibniz formula for pi approximation pi_approximation_leibniz += (sign / (2 * iterations + 1)) * 4; ...

// Nilkantha formula for pi approximation pi_approximation_nilkantha += sign * (4.0 / ((2 * iterations + 2) * (2 * iterations + 3) * (2 * iterations + 4)));
```

Die Zeitmessung im Code zeigt die Effizienzunterschiede zwischen den beiden Methoden, wobei die Nilkantha-Methode, wie angegeben, die schnelleren Ergebnisse liefert.

```
elapsedTimeLeibniz = xTaskGetTickCount() - startTimeLeibniz; ...
elapsedTimeNilkantha = xTaskGetTickCount() - startTimeNilkantha;
```

6.2 Diskussion der Resultate

Die Resultate des Codes zeigen deutlich, dass die Nilkantha-Methode eine effizientere Wahl für die Pi-Berechnung ist, zumindest in Bezug auf die Konvergenzgeschwindigkeit. Dies ist ein wichtiger Faktor für Echtzeitanwendungen, wo die Reaktionszeit und die Effizienz kritisch sind.

7.1 Vergleich der Rechenleistung

Die Nilkantha-Methode konvergiert schneller zur tatsächlichen Zahl Pi im Vergleich zur Leibniz-Methode, was auf ihre unterschiedliche mathematische Struktur zurückzuführen ist. In der Nilkantha-Formel werden Terme mit einem größeren Nenner hinzugefügt, was zu einer schnelleren Annäherung an Pi führt, während die Leibniz-Formel mit kleineren Nennern operiert, die eine langsamere Konvergenz zur Folge haben.

Die schnelle Konvergenz der Nilkantha-Methode führt zu einer geringeren Anzahl von benötigten Iterationen und einer kürzeren Rechenzeit, um eine ähnliche Genauigkeit zu erreichen, was in den gemessenen Zeiten im Code reflektiert wird.

Softwareverwaltung mit GIT

8.3 Link zum Repository auf Github

https://github.com/annoyedmilk/piCal/

Fazit

Das vorgelegte Projekt stellt eine umfassende Implementierung dar, die sich mit der Berechnung von Pi mithilfe zweier unterschiedlicher Algorithmen befasst. Durch die sorgfältige Organisation in Form von verschiedenen Tasks, die Verwendung von Semaphoren zur Synchronisation und die Einrichtung einer Zeitmessmethode wird eine solide Basis für die Untersuchung der Effizienz und Genauigkeit der beiden Algorithmen geschaffen. Der direkte Vergleich zwischen der Leibniz- und der Nilkantha-Methode hat gezeigt, dass die Nilkantha-Methode eine schnellere Konvergenz und damit eine effizientere Berechnung von Pi bietet.