Übung  
Calculating Pi

**Übung**

**Calculating Pi**

im Studiengang Elektrotechnik HF,   
Fach: Embedded Systems

vorgelegt von **Michael Meier**TSD 1702 A

am 16. April 2019   
an der Juventus Technikerschule HF Zürich

Prüfer: Martin Burger

# Inhaltsverzeichnis

Inhaltsverzeichnis 2

Abbildungsverzeichnis 3

1 Erklärung des Gewählten Algorithmus 4

2 Beschreibung der Tasks 5

2.1 Displaytask 5

2.2 Buttontask 6

2.3 Pi-berechnen Task 7

3 Eventbits 8

3.1 1. EventBit 8

3.2 2. EventBit 8

3.3 3. EventBit 8

4 Zeitmessung 9

5 Fazit 10

# Abbildungsverzeichnis

Abbildung 1 Leibnetzreihe 5

Abbildung 2 Leibnetzreihe als summe 5

Abbildung 3 Displaytask 6

Abbildung 4 Buttontask 7

Abbildung 5 Ausschnitt aus Pi-Berechnungstask 8

# Erklärung des Gewählten Algorithmus

Zum Berechnen von Pi habe ich mich für die Leibnetzreihe entschieden. Dies aus dem Grund, da sie sehr einfach mit Software umzusetzen ist. In Abbildung 1 ist der Anfang der Reihe zu sehen welche immer weiter zu Pi-Viertel konvergiert. Abbildung 2 zeigt die Leibnetzreihe als Summe, was auch die einfache Umsetzung in Software erahnen lässt.

Wie schon erwähnt konvergiert die Leibnetzreihe immer weiter zu Pi-Viertel und zwar indem abwechslungsweise ein Teil weg und wieder dazu gerechnet wird. Diese Teile werden immer kleiner und so nähert sich das Ergebnis immer weiter an Pi-Viertel. Der Nachteil dieses Algorithmus sind die vielen Iterationen welche nötig sind, um eine vergleichsweise tiefe Genauigkeit von Pi zu erhalten

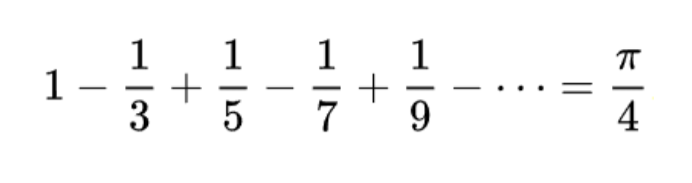


Abbildung 1 Leibnetzreihe

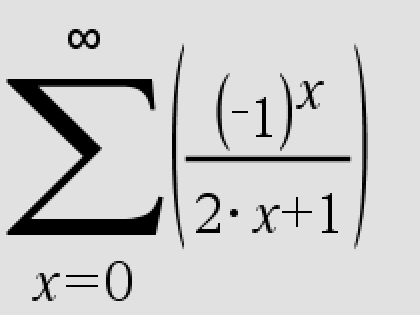


Abbildung 2 Leibnetzreihe als summe

# Beschreibung der Tasks

Mein Programm ist in Drei Tasks aufgeteilt, welche wie die jeweiligen Namen schon sagen für Display, Buttons und die Berechnung von Pi verantwortlich sind.

## Displaytask

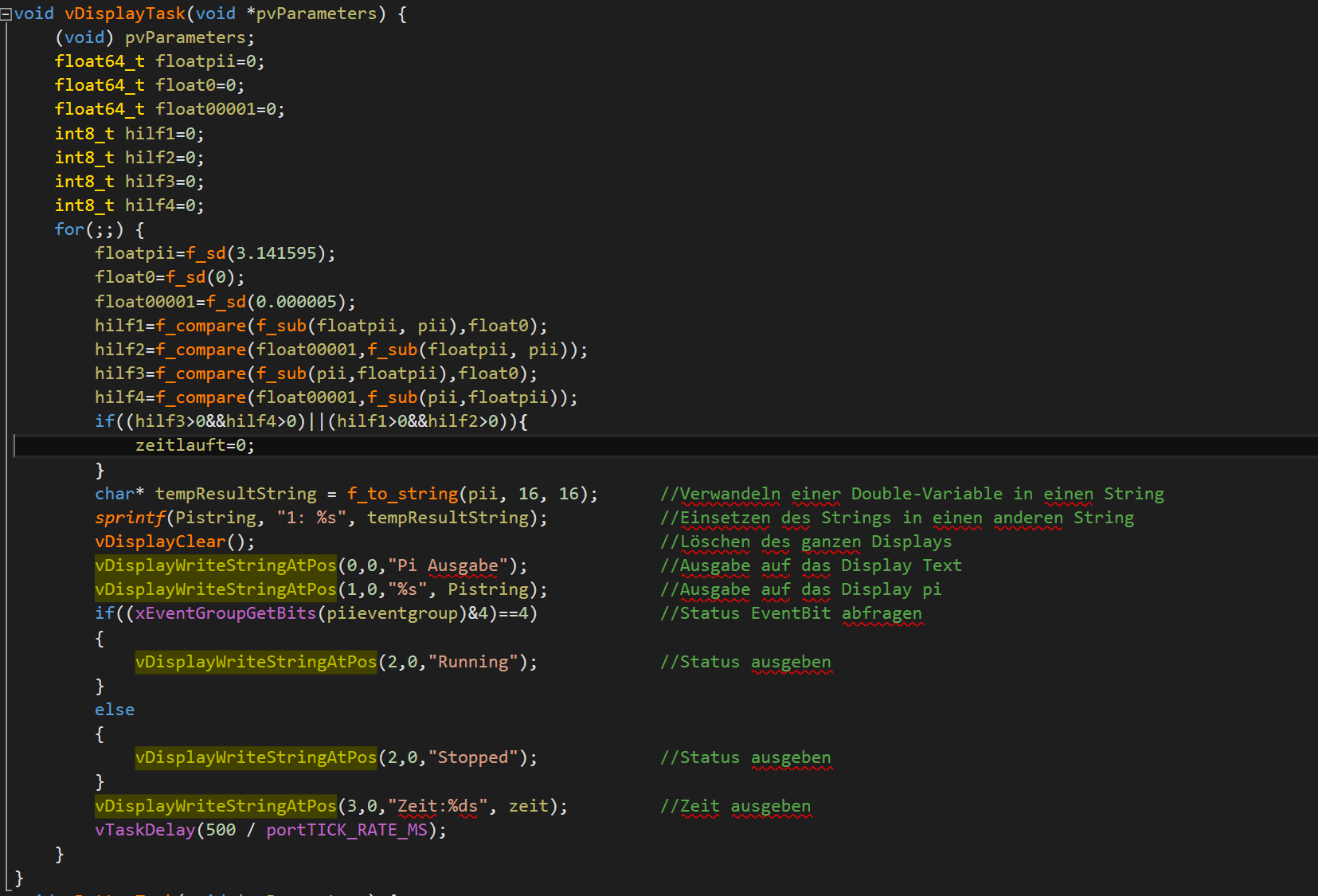


Abbildung 3 Displaytask

In diesem Task wird die Displayausgabe verwaltet. Ausgabe von Text, Berechnungsergebnis, Status und Zeit.

Ebenfalls in diesem Task, ist der Vergleich ob die geforderte Genauigkeit erreicht ist. Falls dies der Fall ist wird die Zeit angehalten.

## Buttontask

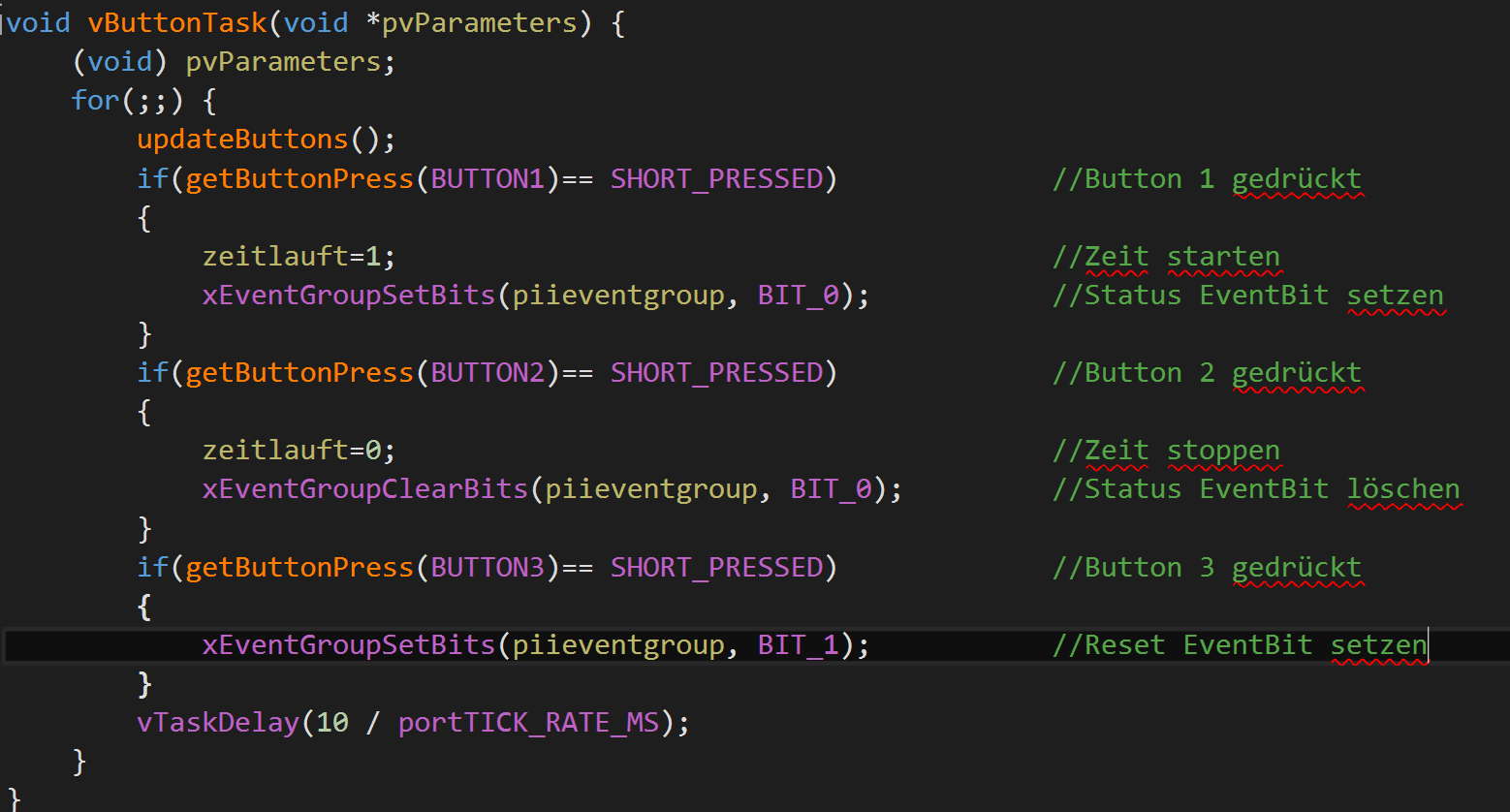


Abbildung 4 Buttontask

In diesem Task werden alle Knöpfe abgefragt und die jeweiligen EventBits gesetzt oder gelöscht.

## Pi-berechnen Task

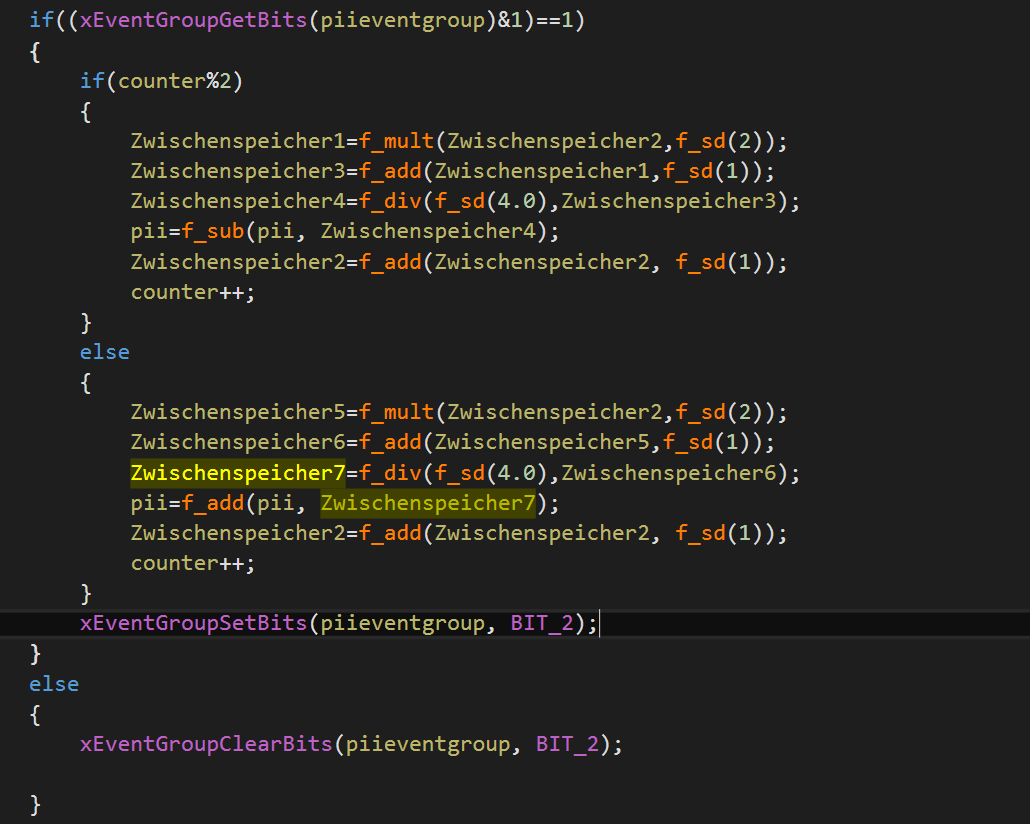


Abbildung 5 Ausschnitt aus Pi-Berechnungstask

In diesem Ausschnitt ist die Berechnung von Pi zu sehen. Da ich Float64 nutze sind die Operationen sehr umständlich und langsam. Hier ebenfalls zu sehen, ist das Setzen und Löschen der Status EventBits. Um bessere Übersicht zu erhalten habe ich diverse Zwischenspeicher Variablen erstellt. Natürlich währe es auch mit weniger lauffähig. Die Übersichtlichkeit und Wartbarkeit würden aber stark leiden.

# Eventbits

Ich habe eine Eventgroup mit dem Namen «Piieventgroup» erstellt. Darin sind alle meine Eventbits gespeichert. Ich nutze drei Eventbits.

## 1. EventBit

Mit diesem EventBit starte ich die Berechnung und stoppe diese auch wieder. Es wird im «Buttontask» gesetzt und gelöscht. Genutzt wird es im «Pi Berechnung Task».

## 2. EventBit

Mit diesem EventBit löse ich ein Reset aus. Ich setze es im «Buttontask», genutzt und gelöscht wird es im «Pi Berechnung Task».

## 3. EventBit

Dieses EventBit repräsentiert den aktuellen Status der Berechnung. In meinem Fall gibt es zwei, und zwar running und stopped. Gesetzt und gelöscht wird dieses Bit im «Pi Berechnung Task». Die Auswertung geschieht im «Displaytask».

# Zeitmessung

Wie schon erwähnt ist meine Berechnung relativ langsam. Ich habe es auch viel schneller hinbekommen mit der Leibnetzreihe, indem ich alles als Double gerechnet habe. Durch die beschränkten Stellen nach dem Komma hat sich aber nach der fünften stelle des Ergebnisses eine Ungenauigkeit eingeschlichen. Für unsere Aufgabe währe es ausreichend gewesen, ich war aber damit nicht zufrieden. Deshalb habe ich es mit float64 gelöst, was langsam aber dafür genau ist. Es werden ca. 160 Sekunden benötigt, um Pi auf fünf Stellen nach dem Komma zu berechnen. Anders gesagt es braucht 136122 Durchläufe mit der Leibnetzreihe, um die geforderte Genauigkeit zu erreichen. Mit anderen Worten, es werden 850 Durchläufe pro Sekunden berechnet. Damit braucht ein Durchlauf 38000 Taktzyklen, was nicht gerade effizient ist. Schneller würde es mit Doublewerten oder natürlich mit einem anderen Algorithmus gehen.

# Fazit

Zwar erfüllt meine Software meiner Meinung nach, alle Anforderungen. Ich hätte aber gerne noch mehr gemacht. Das Programmieren macht mir grossen Spass und «FreeRTOS» vereinfacht einiges und eröffnet ganz neue Möglichkeiten. Ich habe im Verlauf dieser Arbeit sehr viel dazugelernt. Vor allem der Umgang und die Anwendung von Fliesskommazahlen. Beim nächsten Mal würde ich nur Kleinigkeiten anders machen. Ich bin mit meinem Vorgehen und dem Ergebnis im Grossen und Ganzen zufrieden.