Embedded Systems

Pi Berechnung EduBoard

der **Juventus Technikerschule HF**

**Pascal Iten 2022 Embedded Systems PI Calc.** 23.10.2022

**Inhaltsverzeichnis:**

[Abbildungsverzeichnis 3](#_Toc117442161)

[Einleitung zum Bericht 4](#_Toc117442162)

[1 Leibniz-Reihe 5](#_Toc117442163)

[2 Wallissches-Produkt Algorithmus 5](#_Toc117442164)

[3 Eventbits 6](#_Toc117442165)

[3.1 STARTSTOPP 6](#_Toc117442166)

[3.2 RESET 6](#_Toc117442167)

[3.3 PI\_COLLECT 6](#_Toc117442168)

[3.4 PI\_EVEN 6](#_Toc117442169)

[3.5 BREAK 6](#_Toc117442170)

[3.6 ALGORITHMUS 6](#_Toc117442171)

[4 Tasks 7](#_Toc117442172)

[4.1 vController 7](#_Toc117442173)

[4.2 vDisplay 8](#_Toc117442174)

[4.3 vLeibniz 8](#_Toc117442175)

[4.4 vWallissches 9](#_Toc117442176)

[4.5 vZeit 10](#_Toc117442177)

[5 Zeitmessung 11](#_Toc117442178)

[Persönliches Fazit 12](#_Toc117442179)

[Literaturverzeichnis 13](#_Toc117442180)

[Anhang 14](#_Toc117442181)

# Abbildungsverzeichnis

[Abbildung 1 Leibniz-Reihe (1) 5](#_Toc117442182)

[Abbildung 2 Wallissches-Produkt (1) 5](#_Toc117442183)

[Abbildung 3 Ausschnitt Controller Task 7](#_Toc117442184)

[Abbildung 4 Ausschnit Display-Task 8](#_Toc117442185)

[Abbildung 5 Ausschnitt Leibniz-Task, Bereich Rechnen 8](#_Toc117442186)

[Abbildung 6 Wallissche-Task, Bereich Rechnen 9](#_Toc117442187)

[Abbildung 7 Ausschnitt Pausenaufzeichnung Zeitmessung 10](#_Toc117442188)

[Abbildung 8 Annäherung von unten her 11](#_Toc117442189)

[Abbildung 9 Annäherung von oben her 11](#_Toc117442190)

# Einleitung zum Bericht

Der Bericht beschreibt die Eigenschaften einer Softwarelösung, welche im Rahmen einer Schulischen Arbeit im Fach «Embedded Systems\* an der Juventus Technikerschule durchgeführt wurde. Es beinhaltet die Verwendung von Tasks, EventBits wie auch einfache Bausteine wie Schleifen, if/else usw. Im Projekt wurde ein vorgegebener Algorithmus verwendet, dies wäre die sogenannte «Leibniz-Reihe». Auch wurde ein frei gewählter Algorithmus gewählt, welcher in diesem Falle das «Wallissche-Produkt» ist. Diese beiden Algorithmen sind ähnlich und einfach verständlich, jedoch kommt «Leibniz-» dem effektiven Pi immer näher, während «Wallissche-» immer um Pi herumspringt und nicht zur Ruhe kommt.

# Leibniz-Reihe

Dieser Algorithmus wurde um 1671-1674 entdeckt von James Gregory bspw. von Gottfried Wilhelm Leibniz. In der Formel wird vom Startwert 1 abwechselnd ein Wert mit sich um zweivergrössernden Nenner abgezogen bspw. dazu genommen, siehe in Abbildung 1. Wird dieser Wert vervierfacht nähert man sich an Pi an.

Ein Bild, das Text, Uhr, Messanzeige enthält.

Automatisch generierte Beschreibung

Abbildung Leibniz-Reihe (1)

# Wallissches-Produkt Algorithmus

Das Wallissche-Produkt wurde im Jahre 1655 von Jahn Wallis entdeck und nähert sich sehr schnell an Pi/2 heran. Die Art des Algorithmus jedoch führt auch dazu, dass sich Pi auch immer wieder weiter von Pi entfernt, um dann wieder näher zu kommen. Der Algorithmus ist in Abbildung 2 ersichtlich.

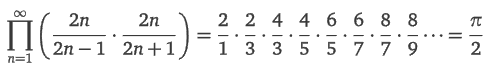


Abbildung Wallissches-Produkt (1)

# Eventbits

Damit das Programm läuft und alle Tasks die nötigen Informationen zuhanden haben, sind die nachfolgenden EventBits bereitgestellt. Diese wurde mit dazugehörigen Befehlen abgeffragt, gesetzt oder auch gelöscht.

xEventGroupSetBits(); Setzt das gewünschte Bit auf «1»

xEventGroupClearBits(); Setzt das gewünschte Bit zurück auf «0»

xEventGroupGetBits(); Fragt den aktuellen Status des Bits ab

xEventGroupWaitBits(); Wartet bis das folgende Bit gesetzt wird

## STARTSTOPP

Dieses EventBit wurde dazu benutzt, eine Freigabe zu erteilen für den Rechenvorgang und damit hergehend Information für alle Aktivitäten, welche mit diesem Zustand zu tun haben. Wird dieses gesetzt starten zum Beispiel die Rechnung, damit zusammenhängend die Darstellung auf dem Display wie auch die Zeitmessung währenddessen.

## RESET

Das ResetBit wurde dafür gebraucht, um nach einer erfolgten Berechnung oder auch währenddessen die Werte zurückzusetzen damit ein Neustart möglich ist.

## PI\_COLLECT

In dem Display-Task wurde das Bit benötigt, damit dieser wusste, ab wann mögliche Daten zugänglich sind, um abgegriffen zu werden, ohne fehlerhaft zu sein, weil diese im gleichen Moment abgeändert werden könnten. Nach dem Auslesen der Daten, wird es wieder zurückgesetzt.

## PI\_EVEN

Dieses Bit wurde dafür verwendet, um zwischen dem Berechnendem Task und dem Zeitmess-task die nötigen Informationen zu teilen, wann genau Pi beim Berechnen erreicht wurde.

## BREAK

Das BreakBit wurde, wie das bereits erwähnte CollectBit dazu verwendet, um die Daten vom Berechnenden abzugreifen. Diese hier jedoch signalisiert dem Berechnenden Task, dass Daten abgegriffen werden möchten, von Seiten Display-Task. Dieses wird

## ALGORITHMUS

Der Algorithmus-Task ist eine reine Speicherung, welcher der beiden Algorithmen aktuell im Berechnenden Modus ist. Ist es gesetzt, ist es der Leibniz-Task, ist es nicht gesetzt, ist es der Wallissche-Task.

# Tasks

Es werden in diesem Konstrukt fünf verschiedene Tasks benötigt, die Tasks sind nicht in einer State-Machine eingebettet, da dies in dieser Anwendung auch ohne realisiert werden kann.

## vController

Der Controller-Task ist dazu da, Tasten, welche gedrückt werden zu erkennen und darauffolgend die nötigen Änderungen am Ablauf vorzunehmen. Dies hat mitunter dem Zeit-Task die höchste Priorität, da Tasten immer erkannt werden sollten. Für die Erkennung der Tasten wurde die Library ButtonHandler.c/.h im Projekt eingefügt, welches die Erkennung von nicht gedrückt, kurz gedrückt wie auch lang gedrückt ermöglicht. In diesem Projekt wurden jedoch nur die Kurz gedrückten Tastenerkennungen genutzt.

In Abbildung 3 sind alle möglichen Aufgaben ersichtlich welche in diesem Task auch genutzt wurden, aus Einfachheit und Übersichtlichkeit wurde nur dieser Ausschnitt gewählt.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung Ausschnitt Controller Task

eTaskState state = eTaskGetState(); Fragt den aktuellen Status eines Tasks ab

if(state == eSuspended) {} je nach Status eine Aktion ausführen

vTaskSuspend(); setzt einen Task aus, bis wieder aufgenommen

vTaskResume(); nimmt einen Task wieder auf

dauer = 0; Variable «dauer» soll den Wert «0» erhalten

vTaskDelay(); setzt den Task aus bis Zeit X abgelaufen ist

* Taste 1 (links) startet den Algorithmus wie auch die Zeitmessung
* Taste 2 unterbricht den Algorithmus und die Zeitmessung
* Taste 3 löscht die aktuellen Werte und macht, dass die Berechnung von vorne begonnen werden kann.
* Taste 4 ist für den Wechsel zwischen den beiden Algorithmen zuständig

Dieser Tasks wird im Abstand von 10ms durchgeführt und fragt diese einfach alle nacheinander immer wieder ab.

## vDisplay

Der Display-Task umfasst die Bereiche für die Darstellung der berechneten Werte nach Starten der Anwendung. Hat die Berechnung nicht begonnen, bleibt die Anzeige so, wie diese im Main gesetzt wurde. Der Task ist auch so ausgelegt, dass er eine Pause des Berechnens mit Bestätigung von den Tasks fordert, damit er die richtigen Daten abrufen kann, danach gibt er die Berechnung wieder frei. Dies geschieht mit «xEventGroupWaitBits()» siehe Abbildung 4, Zeile 3. Dieses Bit wird vom Berechnungs-Task gesetzt.

Die Informationen werden via «vDisplayWriteStringAtPos()» auf die Anzeige ausgegeben. Erste Zahl beschreibt die Zeile, die zweite spalte. Je nach Algorithmus wird ein anderer Modus am Display angezeigt. Die Zeit und der berechnete Pi-Wert ändern sich, alles andere bleib bestehen. Die Anzeige wird alle 500ms erneuert.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung Ausschnit Display-Task

## vLeibniz

Zuerst werden bei einem Reset alle Werte zurückgesetzt oder gesetzt, dadurch sollten alle Werte wieder auf Startformation sein. Dies wird mit der Überprüfung des «RESET\_SHORT» EventBits durchgeführt. Als nächstes wird das «STARTSTOPP» Bit überprüft ob die Berechnung überhaupt freigegeben ist. Das «BREAK» Bit ist dazu da, um bei einem gewollten Lesezugriff das Rechnen zu unterbrechen und das «PI\_COLLECT» Bit zu setzen, damit ein Auslesen der Daten erlaubt ist, ersichtlich in Abbildung 5. Damit immer der Vollständige Wert vorhanden ist, werden beide Operatoren «+» & «-» in einem Durchlauf durchgeführt, dafür ist eine zusätzliche Variable «piHilfe» nötig für die zwischenspeicherung.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung Ausschnitt Leibniz-Task, Bereich Rechnen

Sind die Berechnungen durchgeführt, werden diese mit 100’000 multipliziert und als «int» gespeichert, dadurch gehen alle nicht benötigten Nachkommastellen verloren. Dieser Wert wird dann mit einer vorgespeicherten Variablen verglichen. Ist der Wert derselbe, wird ein EventBit gesetzt, welches signalisiert, dass ein Treffer gefunden wurde.

## vWallissches

Bei diesem Task werden wie im vorherigen «vLeibniz» alle Werte zuerst zurückgesetzt bspw. gesetzt. Der Task ist gleich aufgesetzt, hat auch dieselbe Priorität zugesprochen bekommen. Die einzigen Unterschiede belaufen sich auf die Art der Berechnung, diese sieht im vWallisschen so aus wie in Abbildung 6 ersichtlich und der andere Unterschied ist, dass dieser immer dann suspended ist, wenn vLeibniz nicht ist und umgekehrt. Auch hier wird am Ende der Wert verglichen und bei einem Treffer das EventBit «PI\_EVEN» gesetzt. Dies wird im Zeit-Task weiterbenötigt. Diese Beiden Tasks haben keine delay verbaut, da diese am besten jede mögliche Zeit nutzen sollten, diese die Niedrigste Priorität besitzen und somit immer den anderen Vorrang geben müssen.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung Wallissche-Task, Bereich Rechnen

## vZeit

Dieser Task wurde nicht mit einem allgemeinem delay versehen, sondern mit einem «vTasDelayUntil()». Dies aus dem Grund, da dieser Task genau sein sollte, heisst es muss ein planbarer Moment dauern damit diese Zeitstabil bleiben. Dies wurde nach dem im Unterricht behandelten «ES\_T04\_TimeManagement\_V1.3» Dokument aufgebaut (2).

Es wurden diverse neue Variablen eingeführt, «start» & «stop» für die aktuellen Tickwerte, wann diese Gestartet wurden und wann Pi erreicht wurde. «start» wird im ersten Durchlauf neu geschrieben, wenn dieser vorher «0» war, danach nicht mehr, «stopp» wird jede rRunde neu geschrieben. Weitere sind «pause», «pauseKlein», «pauseGross» und «hilfe» welche benötigt werden um bei einem pausieren der Berechnung, diese Zeit wieder abzählen zu können, da diese ja die Berechnung nicht betreffen sollten. Diese werden bei einem Stoppen der Berechnung gestartet und bei Weiterfahren abgeschlossen und in der Variable «pause» gespeichert, diese wird bei jedem neuen pausieren erweitert.

Bei Erreichen des Pi Werts wie auch bei einem Resetsignal wird nach Rücksetzen aller Werte der Task vZeit suspendet, da er bis zum neuen Start der Berechnung nicht benötigt wird. Dies ist alles ersichtlich in Abbildung 7.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung Ausschnitt Pausenaufzeichnung Zeitmessung

# Zeitmessung

Die beiden verschiedenen Mess-Algorithmen wurden mehrfach immer wieder durchlaufen gelassen und die Werte notiert. Die Werte bei den verschiedenen Messungen der jeweiligen Algorithmen weichen nicht gross ab, jedoch zueinander sind massive Unterschiede ersichtlich.

Leibniz: 13 … 14 Sekunden (nach dem plus Auswerten)

Leibniz: 8 … 9 Sekunden (nach dem minus Auswerten)

Wallissche: 1 … 2 Sekunden

Grundsätzlich ist der Wallissche-Task nicht schneller, durch seine Rechenart jedoch hat er direkt nach Begin die 5.-stellige Nachkommazahl getroffen, danach entfernt dieser sich schnell wieder davon. Wäre es auf z.B. 10 Nachkommastellen genau geplant, wäre dieser Task vermutlich um einiges länger dran, wenn er diesen Wert überhaupt erreichen kann.

Dadurch das wir uns von zwei Seiten her beim Leibniz annähern, wurde durch eine Exceltabelle der Nenner der jeweiligen Berechnungen herausgefunden, bei welcher die 3.14159 zuerst erreicht werden. Ergebnis folgend

(Zeile – 1 = Berechnungszahl; Nenner; +/-; gerechneter Wert; effektiv Pi):

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Abbildung Annäherung von unten her

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Abbildung Annäherung von oben her

Ersichtlich ist das Pi auf 5 Nachkommastellen bei der Annäherung von oben um mehr als den Faktor 2.5 früher eintritt, deshalb wurde dies im Nachhinein im Code noch angepasst, der Ursprüngliche Ort der anderen Anzeige jedoch drin gelassen, jedoch auskommentiert.

Der Prozessor läuft auf 32MHz und hat 1’000Ticks/Sekunde, dies führt zu 32'000 Zyklen pro Tick. Bei 8 Sekunden sind dies im Besten Fall, wenn es nur Berechnungen wären, 17'000 Berechnungen/Sekunde und somit 17 Berechnungen pro Tick.

Pro Berechnung fallen ~1'800 Zyklen an. Jedoch sind noch andere Tasks nötig welche hier nicht besser aufgeschlüsselt werden können.

# Persönliches Fazit

Die Arbeit an sich war ok, Neuerungen mit dem Bekanntmachen der Tasks und das ein einsetzten ist sicher sehr wertvoll und ich denke im kleinen Rahmen ist dies mit dem Eduboard auch ohne Probleme möglich. Ich denke, wenn es dann grössere Projekte gibt, wird dies sicher ein relevanter Punkt welchem viel Beachtung geschenkt werden muss.

Software allgemein ist ein sehr interessantes Projekt, jedoch reichen meine Kenntnisse leider nicht wirklich weit genug. Auch sind mir Grundregeln im Bereich der Software eher fremd, vieles kommt einfach von Lernvideos oder Stack-Overflow Vorschlägen, welche halt direkt umgesetzt wurden, bspw. einen gewissen brauchbaren Wert hatten.

# Literaturverzeichnis

1. **Hemmerich, W.A.** die Kreiszahl Pi. [Online] [Zitat vom: 21. 10 2022.] https://matheguru.com/allgemein/die-kreiszahl-pi.html.

2. **Burger, Martin.** OpenOlat. [Online] V1.3. https://lernen.juvecampus.ch/auth/1%3A1%3A27000093015%3A3%3A0%3Aserv%3Ax%3A\_csrf%3A06fea5ef-3eb0-45b8-bc53-e609a8196e32/Unterlagen/ES\_T04\_TimeManagement\_V1.3.pdf.

# Anhang

Ablage Software und Bericht: <https://github.com/itenpascal/U_PiCalc_ES>