

Calculating Pi

Hidinger Sven

Version 1.0

Kurs: Embedded Systems

Datum: 04.10.2020

Inhalt

[**1.** **Aufgabenstellung** 3](#_Toc53579829)

[**2.** **Erklärung des zusätzlichen Algorithmus** 4](#_Toc53579830)

[**3.** **Beschreibung der Tasks** 4](#_Toc53579831)

[3.1 ButtonTask 4](#_Toc53579832)

[3.2 Steuertask 5](#_Toc53579833)

[3.3 GetTime Task 8](#_Toc53579834)

[3.5 Leibniz Task 9](#_Toc53579835)

[3.2 Kellular Task 10](#_Toc53579836)

[**4.** **EventBits und TaskNotification** 11](#_Toc53579837)

[4.1 EventBits und TaskNotification im Leibniz Task 11](#_Toc53579838)

[4.2 EventBits und TaskNotification im Kellular Task 12](#_Toc53579839)

# **Aufgabenstellung**

* Realisiere die Leibniz-Reihe in einem Task
* Wähle einen weiteren Algorithmus aus dem Internet
* Realisiere den Algorithmus in einem weiteren Task
* Schreibe einen Steuertask, der die zwei erstellten Tasks kontrolliert

Dabei soll folgendes gegeben sein:

* Der aktuelle Wert soll angezeigt werden. Update alle 500ms
* Der Algorithmus soll mit einem Tastendruck gestartet werden und mit einem anderen Tastendruck gestoppt werden.
* Mit einer dritten Taste kann der Algorithmus zurückgesetzt werden
* Mit der vierten Taste kann der Algorithmus umgestellt werden.

(zwischen Leibniz und dem zweiten Algorithmus)

* Die Kommunikation zwischen den Tasks kann entweder mit Eventbits oder über TaskNotifications stattfinden.
* Folgende Eventbits können beispielweise verwendet werden:
  + EventBit zum Starten des Algorithmus
  + EventBit zum Stoppen des Algorithmus
  + EventBit zum Zurücksetzen des Algorithmus
  + EventBit für den Zustand des Kalkulationstask als Mitteilung für den Anzeige-Task
* Mindestens drei Tasks müssen existieren.
  + Interface Task für Buttonhandling und Display-Beschreiben
  + Kalkulation-Task für Berechnung von Pi mit Leibniz-Reihe
  + Kalkulation-Task für Berechnung von Pi mit anderer Methode
* Erweitere das Programm mit einem Zeitmess-Hardware-Timer (wie in der Alarm-Clock Übung) und messe die Zeit, bis Pi auf 5-Stellen hinter dem Komma stimmt. (Zeit auf dem Display mitlaufen lassen und bei Erreichen der Genauigkeit den Timer anhalten. Die Berechnung von Pi soll weitergehen)

# **Erklärung des zusätzlichen Algorithmus**

Ich habe einen Algorithmus gewählt, der für mich einfach zu implementieren war. Die Formel lautet:



Diese Formel hat der Mathematiker und Astronom Kelallur Nilakantha Somayaji entdeckt. Sie konvergiert nicht sehr schnell, aber die aufsummierten Brüche berechnen ziemlich genau die Nachkommastellen von Pi. Die drei ist somit immer am Anfang und läuft vorneweg. Die Berechnung läuft, wie bei der Leibniz Reihe ausser da die Näherung direkt an Pi geht.

# **Beschreibung der Tasks**

Hier werden die Implementierten Tasks beschrieben. In dieser Beschreibung werden die Abläufe und die Funktionsweise der Tasks beschrieben. Der Steuertask ist in mehreren Abbildungen dargestellt, da der Steuertask viele Schritte hat.

## ButtonTask

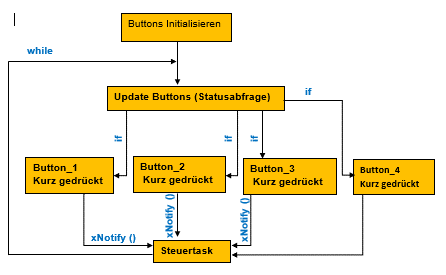


Abbildung 1

Der Ablauf von dem ButtonTask wird in Abbildung 1 dargestellt. Die Funktionsweise ist so, dass zuerst die Buttons initialisiert werden müssen. Dan wird abgefragt ob ein Taster gedrückt ist und wenn wie lange. Dies passiert in dem Button Handler. Mit jeweiligen if abfragen wird gesagt, dass wenn der Taster xxx gedrückt wird, dann den xTaskNotfy aus. Dieser leitet die Information an den Steuertask weiter.

## Steuertask

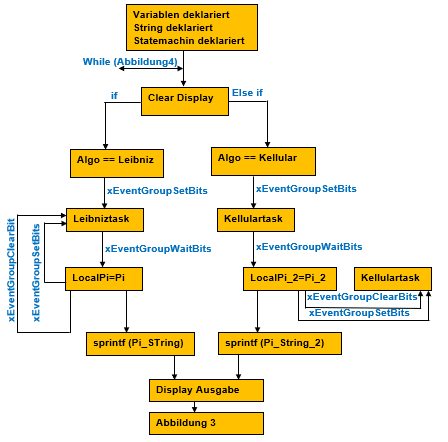


Abbildung 2

In der Abbildung 2 wird der Anfang des Ablaufes von dem Steuertask gezeigt. Zuerst werden die Variablen, Strings und Statemachine implementiert. Das keine Fehler auf dem Display angezeigt werden, wird der Display immer wieder gelöscht. Mit einer Abfrage wird festgestellt in welchem Algorithmus Task er sich befindet. Dan wird mit dem EventGroupSetBits eine Information gesendet an den Algorithmus Task, von dem Task bekommt er mit dem Befehl EventGroupWaitBits die Information zurück. Die Information, die er bekommen hat, ist Pi oder Pi\_2 je nach Algorithmus Task. Pi wird dan zu einer Lokalen Variabel definiert. Sobald dies erledigt ist, werden mit dem Befehl EventGroupClearBits die Bits beim an den Algorithmus Task gelöscht. Mit dem Befehl EventGroupSetBits sendet er, die Information AlgGo wieder an den folgenden Algorithmus Task zurück.

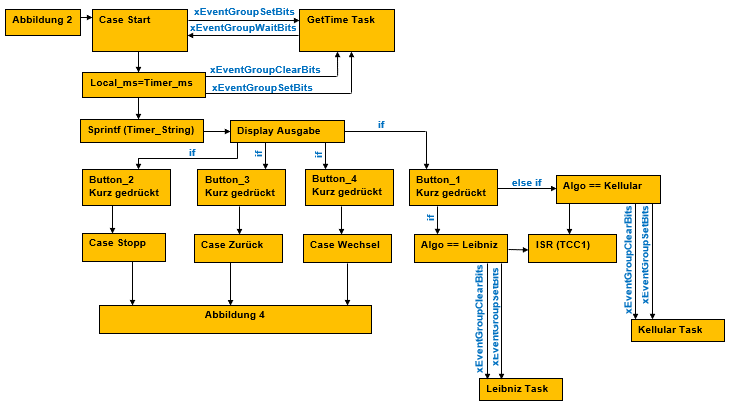


Abbildung 3

Die Abbildung 3 zeigt der Ablauf der Statemachine. Case ist der jeweilige Status der Statemachine. Im Case Start wird mit EventGroupSetBit eine Anfrage gesendet im GetTime Task. Wurde im GetTime Task die Anfrage erhalten, so übermittelt er die Information, in diesem Fall die Variabel Timer\_ms, zurück. Dan wird eine neue Variable Namens Local\_ms definiert. Mit dem Befehl EventGroupClearBits werden die Informationen wieder gelöscht. Dan wird über einen String die Zeit an das Display gesendet, so das die Stoppzeit ausgegeben wird.

Je nach Tastendruck wird mit einer wenn Abfrage geschaut in welchen Status er gehen muss. Bei kurzem Tastendruck der Taste 1 wird entschieden in welchem Algorithmus Task er sich befindet. Mit dieser Taste wird der Start Befehl erteilt, dann wird der Inerrupt der Stoppuhr gestartet. Es werden mit dem Befehl EventGroupBitsClear und EventGroupSetBits weitere Befehle zu den jeweiligen Algorithmus Tasks gesendet.

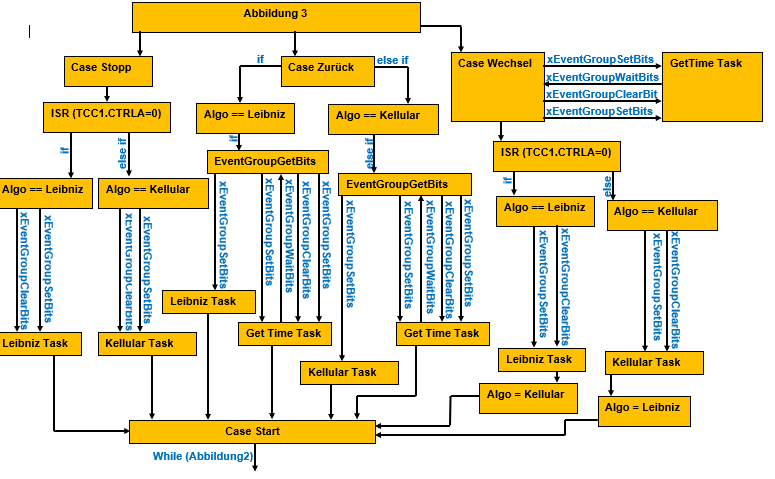


Abbildung 4

Wird die Taste 2 kurz betätigt, geht die Statemachine in den Status Stopp. Dann wird der Timer auf null gestellt und abgefragt in welchem Algorithmus Task er sich befindet. Dan wird in den jeweiligen Task die Information gelöscht mit dem Befehl EventGroupClearBits und eine Information gesendet mit dem Befehl EventGroupSetBits.

Die Statemachine befindet sich im Status Zurück, wenn die Taste 3 kurz betätigt wird. Wenn dies passiert wird auch hier abgefragt in welchem Algorithmus Task er sich befindet. Dan wird mit den EventGroup den Timer in dem GetTime Task zurückgestellt, sowie auch den Algorithmus.

Falls die Taste 4 kurz betätigt wird geht die Statemachin in den Status Wechsel. Es wird wieder eine Abfrage stattfinden, dass er weiss in welchem Algorithmus Task er sich befindet. Danach werden mit den EventGroups die jeweiligen Informationen an die Algorithmus Tasks weitergeleitet. Und je nach Zustand zeigt er den Leibniz Algorithmus oder den Kellular Algorithmus auf dem Display an.

Wenn einer von diesen drei Status am Ende angelangt ist, geht er wieder in den Status Start zurück.

## GetTime Task

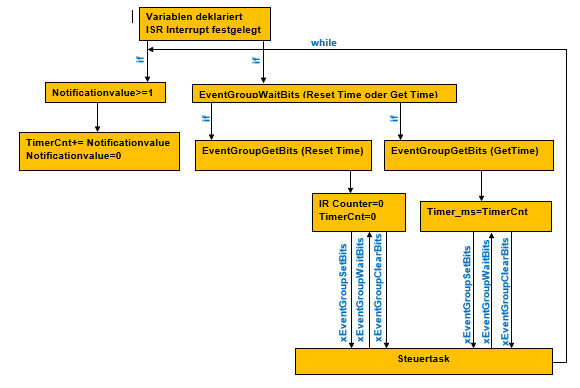


Abbildung 5

In diesem Task wird die Stoppuhr gemanagt. Zuerst werden die Hilfsvariablen definiert und der Interrupt festgelegt in welcher Zeit die Impulse durchgeführt werden. Mit deiner if Abfrage geht es je nach Zustand in den Abschnitt, wenn der Variabel wer grösser gleich eins ist dann wird der Counter gestartet und von Null an aufgezählt. Falls die Information Reset Time oder Get Time kommt, wird ebenfalls abgefragt ob die Information entweder Reset Time oder GetTime gesendet wurde. Wenn Reset gesendet wurde wird der Interrupt Counter und der Timer Counter zurückgesetzt, danach wird die Information mit den Event Groups an den Steuertask weitergeleitet. Falls die Get Time Information gesendet wurde wird die Variabel Timer\_ms definiert und mit den EventGroups die Information weitergeleitet.

## Leibniz Task

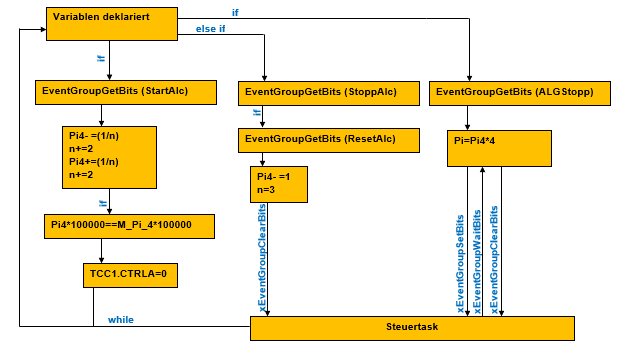


Abbildung 5

Abbildung 5 zeigt den Ablauf des Leibniz Algorithmus. Da werden zuerst die Hilfsvariabel deklariert. Wenn die Information StartAlc kommt fängt er an zu berechnen. Anschliessend wird Pi Viertel auf 5 nach Kommastellen auf die Genauigkeit verglichen, wenn dies der Fall ist, wird die Stoppuhr gestoppt.

Die Information ALGStopp ist anstehend, wird das Pi Viertel mal vier gerechnet, dass man Pi hat, danach wird die Information mit den Event Groups weitergeleitet an den Steuertask. Bei der EventGroupWaitBits wird auf eine Information vom Steuertask gewartet.

Wenn Stopp Alc ansteht als Information und wenn Reset Alc, dann wird Pi4 und n auf den Anfangswert zurückgestellt.

## Kellular Task

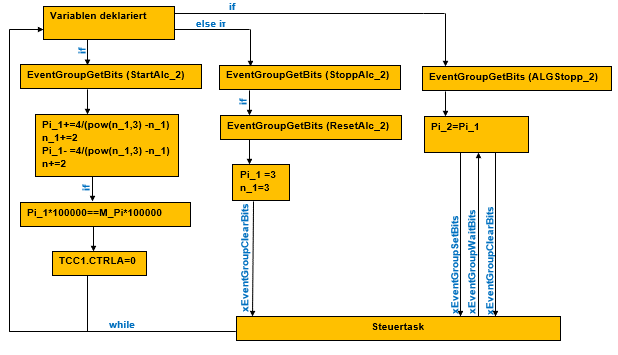


Abbildung 6

In dieser Abbildung 6 wird der Ablauf des Kellular Algorithmus aufgezeigt. Am Anfang werden die benötigten Variablen deklariert. Wenn die Information StartAlc 2 kommt, dann fängt der an zu Rechnen. Anschliessend wird verglichen ob Pi ausgerechnet mit Pi Funktion aus Math Bibliothek, sind diese gleich wird die Stoppuhr gestoppt.

Ist die Information StoppAlc\_2 anstehend und wenn Informatin ResetAlc\_2 ist, werden die Start Variabel auf den Anfangs wert zurückgestellt. Mit einem Event Group Bit Clear werden am Steuertask die Informationen wieder gelöscht.

Falls die Information ALGStopp ansteht wird die variabel P\_2 definiert und anschliessend mit den Event Groups die Informationen an den Steuer Task weitergleitet. Mit dem Befehl EventGroupWaitBits wird vom Steuertask eine Information erwartet.

# **4. EventBits und TaskNotification**

In diesem Abschnitt werden die Verwendeten EventBits und TaskNotification erklärt. Es wird erklärt wie die verschiedenen EventsBits oder TaskNotitification verwendet werden und welche Aufgabe sie haben. Die verschiedenen EventBits werden in jedem Task einzeln Beschrieben.

## 4.1 EventBits und TaskNotification im Leibniz Task

xEventGroupGetBits(xKommunikation)&STARTCALC

Dieser Event Group wird in dem Leibniz Task verwendet zur Abfrage. Bei der Event Group Get Bits bekommt er die Information. Das die richtige Information kommt muss ihm in Klammern folgendes gesagt werden. Zuerst welcher Event Group Handle er ist. Hier ist es xKommunikation, dieses Handle ist für die Weiterleitung der Informationen des Algorithmus Task und dem Steuertask zuständig. Das STARTALC ist das Bit, das gesetzt werden musss. In diesem Fall bedeutet es das der Algorithmus starten darf mit der Berechnung.

xEventGroupGetBits(xKommunikation)&STOPPCALC

Hier wird der Fall abgehandelt, wenn die Berechnung gestoppt wird. Auch hier wartet er auf das Bit, das er bekommen soll. Das Bit Lautet STOPPALC, dieses Bit wird von dem Steuertask übermittelt.

xEventGroupGetBits(xKommunikation)&RESETCALC

Wenn Event Bit STOPPALC übermittelt wurde und RESETCALC, dann wird die Berechnung zurückgesetzt. RESETCALC wird im Steuertask gesetzt und gesendet. Das Setzen und Senden dieses Bit ist in der Statemachine im Status Zurücksetzen vorhanden.

xEventGroupClearBits(xKommunikation,RESETCALC)

Wenn zurückgesetzt wurde wird mit dieser Event Group Bits, die Bit ResetALC zurückgesetzt, also gelöscht. Somit kann wird einen Fehler bei der Informationsübertragung vermieden. Auch hier muss in Klammern das Handle angegeben werden und welcher Bit er zurücksetzen soll.

xEventGroupGetBits(xKommunikation)&ALGSTOPP

Das Bit ALGSTOPP ist für die Übertragung von der Variable Pi zu dem Steuer Task verantwortlich. Hier wartet er auf das Bit ALGSTOPP vom Steuertask.

xEventGroupSetBits(xKommunikation,ALG\_AM\_WARTEN)

Mit der Event Group Set Bit wird das Bit ALG\_AM\_Warten an den Steuer Task gesendet, so wird übermittelt das er bereit ist.

xEventGroupWaitBits(xKommunikation,ALG\_GO,pdTRUE,pdTRUE,5/ portTICK\_RATE\_MS)

In dieser Event Group werden noch ein paar andere Angaben benötig. Beim ersten pdTRUE werden die Bit im Ereignis gelöscht, wenn alle Bits übertragen wurden. Beim zweiten ist es auf pdTRUE das er auf alle Bits, das er erhaltet, wartet oder wenn das Timeout abgelaufen ist weitergeht. Das ALG\_GO ist die Information zum Weiterfahren, das man ihm sagt, er hat den Austausch erledigt fahre fort.

xEventGroupClearBits(xKommunikation,ALG\_AM\_WARTEN)

Dieser Bit ALG\_AM\_WARTEN wird hier wieder zurückgesetzt. So dass keine Fehlübermittlungen stattfinden kann. Es ist zur Sicherheit.

## EventBits und TaskNotification im Kellular Task