

Calculating Pi

Hidinger Sven

Version 1.3

Kurs: Embedded Systems

Datum: 14.10.2020

Inhalt

[**1.** **Aufgabenstellung** 3](#_Toc53932341)

[**2.** **Erklärung des zusätzlichen Algorithmus** 4](#_Toc53932342)

[**3.** **Beschreibung der Tasks** 4](#_Toc53932343)

[3.1 ButtonTask 4](#_Toc53932344)

[3.2 Steuertask 5](#_Toc53932345)

[3.3 GetTime Task 8](#_Toc53932346)

[3.5 Leibniz Task 9](#_Toc53932347)

[3.2 Kellular Task 10](#_Toc53932348)

[**4.** **EventBits und TaskNotification** 11](#_Toc53932349)

[4.1 EventBits und TaskNotification im Leibniz und Kellular Task 11](#_Toc53932350)

[4.2 EventBits und TaskNotification im Steuer Task 12](#_Toc53932351)

[4.2.1 Display Kommunikation 12](#_Toc53932352)

[4.2.2 Statemachine 12](#_Toc53932353)

[4.3 EventBits und TaskNotification vom GetTime Task 15](#_Toc53932354)

[4.4 EventBits und TaskNotification vom Button Task 16](#_Toc53932355)

[4.5 EventBits und TaskNotification in der ISR Funktion 16](#_Toc53932356)

[**5.** **Zeitmessung der beiden Algorithmen** 16](#_Toc53932357)

[5.1 Zeitmessung Leibniz Algorithmus 16](#_Toc53932358)

[5.2 Zeitmessung Kellular Algorithmus 17](#_Toc53932359)

[**5** Geschwindigkeitsvergleich 17](#_Toc53932360)

[5.4 Rückschluss zwischen Rechenleistung und Prozessorleistung 17](#_Toc53932361)

[**6** **Abbildungsverzeichnis** 18](#_Toc53932362)

[**7** **Tabellenverzeichnis** 18](#_Toc53932363)

# **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

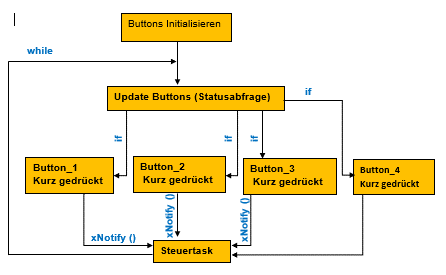


Abbildung 1

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

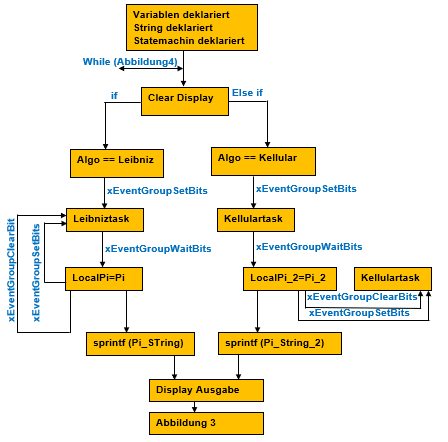


Abbildung 2

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.

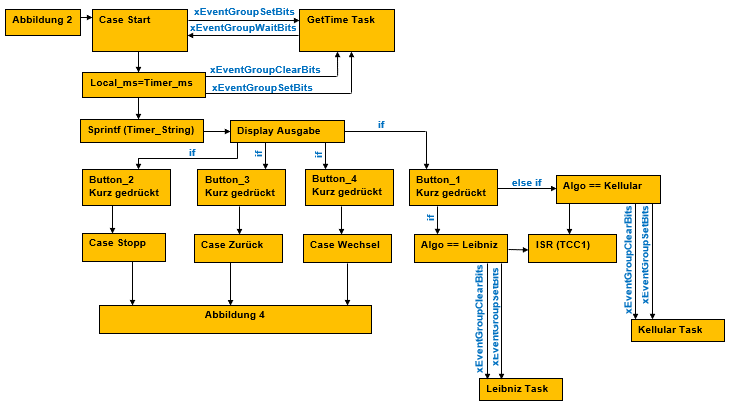


Abbildung 3

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.

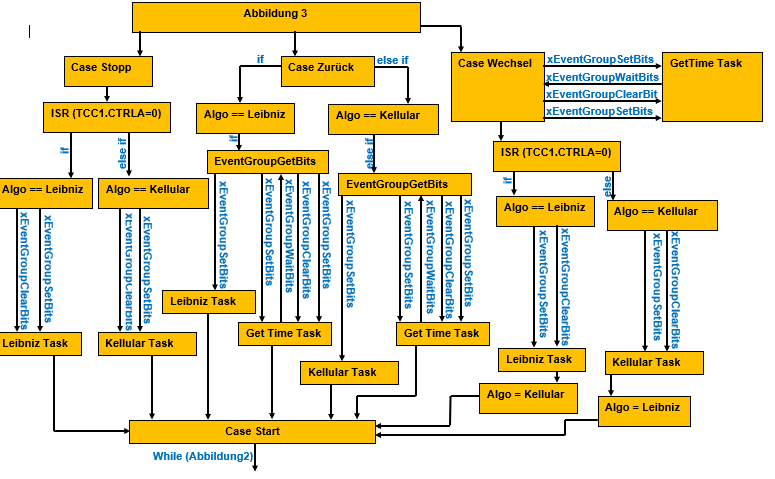


Abbildung 4

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

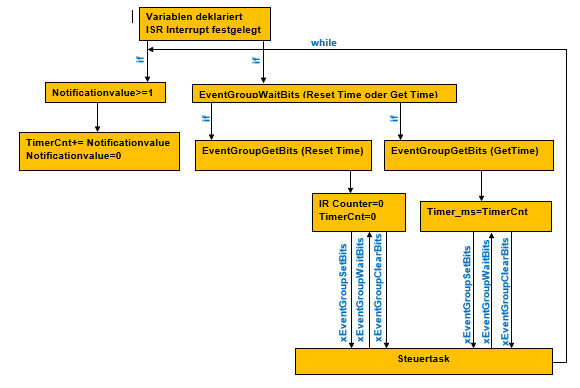


Abbildung 5

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

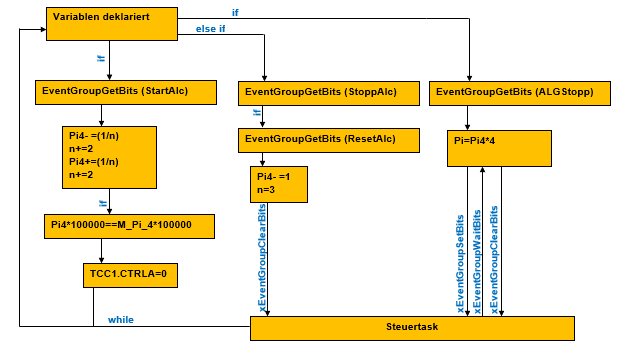


Abbildung 6

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

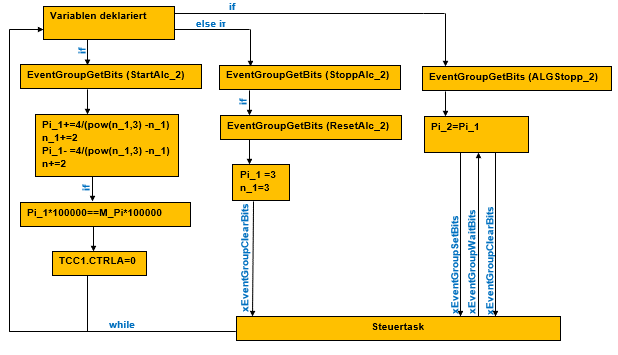


Abbildung 7

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 und Kellular Task

xEventGroupGetBits(xKommunikation)&STARTCALC / STARTCALC\_2)

Dieser Event Group wird in dem Leibniz oder im Kellular 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 zwischen dem Algorithmus Task und dem Steuertask zuständig. Das STARTALC oder STARTALC2 ist das Bit, das gesetzt werden musss. In diesem Fall bedeutet es das der Algorithmus starten darf mit der Berechnung.

xEventGroupGetBits(xKommunikation)&STOPPCALC/STOPPALC\_2)

Falls die beiden Algorithmen zurückgesetzt werden muss die Rechnung gestoppt sein. Er wartet bis das Bit kommt. Das Bit Lautet STOPPALC/STOPPALC\_2, dieses Bit wird von dem Steuertask übermittelt.

xEventGroupGetBits(xKommunikation)&RESETCALC/RESETCALC\_2)

Wenn Event Bit STOPPALC/STOPPCALC\_2 übermittelt wurde und wenn er das RESETCALC/RESETCALC\_2 Bit erhalten hat, dann wird die Berechnung zurückgesetzt. RESETCALC/ RESETCALC\_2 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 Bits ResetALC/RESETALC\_2 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)

Sobald das Erreignisbit aufgerufen wird fängt er an, die variabel Pi zu beschreiben. Das heiss er wartet auf das Bit ALGSTOPP/ALGSTOPP2 vom Steuertask.

xEventGroupSetBits(xKommunikation,ALG\_AM\_WARTEN/ALG\_AM\_WARTEN\_2)

Mit der Event Group Set Bit wird das Bit ALG\_AM\_WARTEN/ALG\_AM\_WARTEN\_2 an den Steuer Task gesendet, so wird die Variabel an den Steuertask übermittelt.

xEventGroupWaitBits(xKommunikation,ALG\_GO/ALG\_GO\_2,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/ALG\_GO\_2 ist die Information zum Weiterfahren, das man ihm sagt, er hat den Austausch erledigt ist und fortfahren kann.

xEventGroupClearBits(xKommunikation, ALG\_AM\_WARTEN/ALG\_AM\_WARTEN\_2)

Dieser Bit ALG\_AM\_WARTEN/ALG\_AM\_WARTEN\_2 wird hier wieder zurückgesetzt. So dass keine Fehlübermittlungen stattfinden kann. Es ist zur Sicherheit.

## EventBits und TaskNotification im Steuer Task

### Display Kommunikation

xTaskNotifyWait(0,0xfffffff,&Buttonvalue,1/portTICK\_RATE\_MS)

Mit dem xTaskNotifyWait wird auf einen Wert gewartet in diesem Fall auf den Buttonvalue. Je nach Tasten druck verändert sich der Wert. Er wartet eine gewisse Zeit ab bis er den Wert erhalten hat.

In diesem Abschnitt wird die Kommunikation zwischen Steuertask und den beiden Algorithmus Tasks. Da es die gleichen Befehle sind, wird es einmal erklärt für beide Algorithmus Tasks.

xEventGroupSetBits(xKommunikation,ALGSTOPP/ALGSTOPP\_2)

Mit dem Bit ALGSTOPP/ALGSTOPP\_2 wird dem Algorithmus Task übermittelt das er starten kann die Variabel zu schreiben.

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

Er bekommt die Variabel zugesendet. Und wartet bis alle Bits gesendet wurden oder die Zeit abgelaufen ist. Die Information bekommt er von dem Algorithmus Task.

xEventGroupClearBits(xKommunikation,ALGSTOPP/ALGSTOPP2)

Die Bits ALGSTOPP/ALGSTOPP2 werden hier gelöscht und wieder auf Null gesetzt.

xEventGroupSetBits(xKommunikation,ALG\_GO/ALG\_GO\_2)

ALG\_GO/ALG\_GO\_2 diese Bits werden verwendet das er wieder fertig ist mit dem Sendevorgang. Er sendet das Bit an den Algorithmus Task.

### Statemachine

Status Start: xEventGroupSetBits(xTimeKom,GET\_TIME)

Mit dem GET TIME Bit sendet der Steuertask den Bit weiter an den GetTime Task.

xEventGroupWaitBits(xTimeKom,WAIT\_TIMER,pdTRUE,pdTRUE,5/portTICK\_RATE \_MS)

Bekommt die Bit Information WAIT\_TIMER von dem Get\_Time Task, es wird die Time\_ms variabel übermittelt.

Status Start:

xEventGroupClearBits(xTimeKom,GET\_TIME)

Der Bit GET\_TIME wird zurückgesetzt in dem Handle Time Kom.

xEventGroupSetBits(xTimeKom,RUN\_TIMER)

* 1. Das Bit RUN\_TIMER wird an Get Time Task weitergeleitet. So kann der Timer beginnen zu starten.

Beide nachfolgenden Event Groups werden gleich gebraucht in den beiden Algorithmus Tasks. Der Unterschied ist die Bits die versendet werden, deshalb werden beide zusammen genommen.

xEventGroupSetBits(xKommunikation,STARTCALC/STARTCALC\_2)

Die Bits STARTCALC und STARTCALC\_2 werden an den Algorithmus Task versendet. So wird der Befehl Start der Berechnung gegeben.

xEventGroupClearBits(xKommunikation,STOPPCALC/STOPPCALC\_2)

Das kein Ereignis sich im Stopp vorgang befindet werden die Bits STOPPCALC und STOPPCALC\_2 zurückgesetzt.

Status Stopp:

xEventGroupSetBits(xKommunikation,STOPPCALC/STOPPCALC\_2)

Je nach Algorithmus wird der Bit STOPCALC oder STOPPCALC\_2 verwendet. Mit diesem Bit sendet er an den Algorithmus Task das er die Berechnung Stoppen soll.

xEventGroupClearBits(xKommunikation,STARTCALC/STARTCALC\_2)

Das kein STARTCALC/STARTCALC\_2 Bit vorhanden ist wird diese zur Sicherheit zurückgesetzt. So wird ein Konflikt ausgeschlossen.

Status Zurück

xEventGroupGetBits(xKommunikation)&STOPPCALC/STOPPCALC\_2)

Wenn das Bit STOPPCALC/STOPPCALC\_2 aufgerufen wurde bekommt er diese Information und geht in den nächsten Schritt.

xEventGroupSetBits(xKommunikation,RESETCALC/RESETCALC\_2)

Mit dem Senden der Bit RESETCALC/RESETCALC\_2 zum Algorithmus wird die Berechnung wieder zurückgesetzt.

xEventGroupSetBits(xTimeKom,RESET\_TIME)

Get Time bekommt das gesendete Bit RESET\_Time und setzt den Timer zurück.

Status Zurück:

xEventGroupWaitBits(xTimeKom, WAIT\_TIMER,pdTRUE,pdTRUE, 5/portTICK\_RATE\_MS)

Wenn die Zeit zurückgesetzt wurde bekommt er die Information mit dem Bit WAIT\_TIMER vom Get Time Task zurück.

xEventGroupClearBits(xTimeKom,RESET\_TIME)

Sobald der Timer zurückgestellt wurde und er das Bit WAIT\_TIMER erhalten hat wird das Bit RESET\_TIME zurückgesetzt.

xEventGroupSetBits(xTimeKom,RUN\_TIMER)

Nach dem zurücksetzen des Bit RESET\_TIME kann er wieder weiterlaufen, dies wird im mit dem Bit RUN\_TIMER gesagt.

Status Wechsel:

xEventGroupSetBits(xTimeKom,RESET\_TIME)

Zuerst wird an den Get Time Task das Bit RESET\_TIME gesendet so das der Timer wieder zurückgesetzt wird.

xEventGroupWaitBits(xTimeKom,WAIT\_TIMER,pdTRUE,pdTRUE, 5/portTICK\_RATE\_MS)

An dieser Stelle bekommt er von dem Get Time Task das Bit Wait\_Timer zurück. Das ihm sagt das der Timer zurückgestellt wurde.

xEventGroupClearBits(xTimeKom,RESET\_TIME)

Dass es kein Konflikt gibt wird das Bit RESET\_TIME zurückgesetzt.

xEventGroupSetBits(xTimeKom,RUN\_TIMER)

Es wird ihm wieder in den Get Time Task ein Bit RUN\_TIMER gesendet das die Information enthaltet, du kannst wieder weiter.

xEventGroupSetBits(xKommunikation,STOPPCALC/STOPPCALC\_2)

Zum Wechseln der Task muss die Berechnung gestoppt werden für dies wird dem Algorithmus Task das Bit STOPPCALC/STOPPCALC\_2 weitergeleitet.

xEventGroupClearBits(xKommunikation,STARTCALC/STARTCALC\_2)

Das Bit STARTCALC/STARTCALC\_2 wird hier zurückgesetzt.

## EventBits und TaskNotification vom GetTime Task

xTaskNotifyWait(0,0xffffffff,&NotificationValue, pdMS\_TO\_TICKS(20))

Mit dem xTaskNotifyWait wird auf einen Wert gewartet in diesem Fall auf den NotificationValue. Die Variabel NotificationValue ist der Wert des auf Zählen von dem Timer Counter. Er wartet eine gewisse Zeit ab bis er den Wert erhalten hat.

xEventGroupWaitBits(xTimeKom,RESET\_TIME|GET\_TIME,pdFALSE,pdFALSE,pdMS\_TO\_TICKS(100))

Er bekommt die Bits RESET\_TIME oder das Bit Get\_TIME vom Steuer Task. Wenn er eins von beiden bekommt geht er in die nächste Abfrage.

xEventGroupGetBits(xTimeKom)&RESET\_TIME)

Wenn das Bit RESET\_TIME im Ereignis vorgekommen ist, dann wird der Timer Counter zurückgesetzt.

xEventGroupSetBits(xTimeKom,WAIT\_TIMER)

Der Gesetzte Wert wird an den Steuertask mit dem Bit WAIT\_TIMER gesendet.

xEventGroupWaitBits(xTimeKom,RUN\_TIMER,pdTRUE,pdTRUE,5/ portTICK\_RATE\_MS)

Der Get Time Task bekommt das Bit RUN\_TIMER von dem Steuertask. Diese Information heisst für den Get Time Task, das der Steuer Task die Information erhalten hat und er fortfahren kann.

xEventGroupClearBits(xTimeKom,WAIT\_TIMER)

Das Bit WAIT\_TIMER wird wieder zurückgesetzt.

xEventGroupGetBits(xTimeKom)&GET\_TIME)

Wenn das Bit GET\_TIME gesetzt wurde im Steuer Task, geht er in diese Abfrage und setz die Variable Timer\_ms.

xEventGroupSetBits(xTimeKom,WAIT\_TIMER)

Dieses Bit WAIT\_TIMER wird an den Steuertask gesendet, so bekommt der Steuertask seine Information, die er benötigt.

xEventGroupWaitBits(xTimeKom,RUN\_TIMER,pdTRUE,pdTRUE,15/ portTICK\_RATE\_MS)

Der Steuertask meldet mit dem Bit RUN\_TIMER dem Get Time Task die Information zurück, dass er weiterfahren kann.

xEventGroupClearBits(xTimeKom,WAIT\_TIMER)

Das Bit WAIT\_TIMER wird zurückgesetzt.

## EventBits und TaskNotification vom Button Task

xTaskNotify(xSteuerTask,BUTTON1SHORTPRESSEDMASK,eSetValueWithOverwrite)

Mit dem TaskNotify wird der Wert an den Steuer Task weitergeleitet. So werden dem Steuertask die Werte direkt übermittelt, so merkt er welcher Taster betätigt wurde.

## EventBits und TaskNotification in der ISR Funktion

xTaskNotifyFromISR( xGetTime,IR\_counter, eIncrement, &Priority )

Dieses Task Notify wird für den Interrupt verwendet. Er gibt die Variabel des IR\_Counter weiter an den Get Time Task. Es wird eine Priorität gegeben so das keine Fehler passieren.

# **Zeitmessung der beiden Algorithmen**

## Zeitmessung Leibniz Algorithmus

|  |  |
| --- | --- |
| **Zeitmessung** | **Messergebnis in ms** |
| 1 | 12703ms |
| 2 | 12701ms |
| 3 | 12706 ms |
| 4 | 12681 ms |
| 5  Tabelle 1 | 12706 ms |

Die Zeiten kommen zustande von der Dauer der Berechnung und der Übermittlung zwischen den Task. Sie weichen an da auch der andere Algorithmus Task sich im Stopp zustand ist und eine Wartezeit hat. Die Warten Zeiten spielen auch eine Rolle.

## Zeitmessung Kellular Algorithmus

|  |  |
| --- | --- |
| **Zeitmessung** | **Messergebnis in ms** |
| 1 | 10 ms |
| 2 | 11 ms |
| 3 | 11 ms |
| 4 | 9 ms |
| 5  Tabelle 2 | 11 ms |

Den Zeit Unterschied in diesem Task erklärt sich dadurch, dass im Ablauf des Programms sich Wartezeiten befindet und je nach längerem oder kürzerem Warten ist er schneller oder langsamer. Die Tasks zusammen müssen ebenfalls kommunizieren und dies braucht eine Gewisse Zeit.

## Geschwindigkeitsvergleich

Der Kellular Algorithmus ist einiges schneller als der Leibniz Algorithmus. Das ist aus dem Grund das beim Kellular die 3 schon am Anfang der Berechnung vorgeschoben wird. So ist diese wesentlich schneller. Der Leibniz brauch da wesentlich länger für die Berechnung.

## Rückschluss zwischen Rechenleistung und Prozessorleistung

Die Prozessorleistung nimmt ab. Da mehrere Tasks miteinander laufen und die auch Warten oder weiterlaufen im Hintergrund. Das geht auf die Prozessorleistung. Da die Tasks auch noch Wartezeiten wird diese langsamer.

# **Abbildungsverzeichnis**

Abbildung 1 Ablauf Button Task Seite 4/18

Abbildung 2 Ablauf Steuer Task (Display Ausgabe) Seite 5/18

Abbildung 3 Ablauf Steuertask Seite 6/18

Abbildung 4 Ablauf Steuertask Seite 7/18

Abbildung 5 Ablauf Get Time Task Seite 8/18

Abbildung 6 Ablauf Leibniz Task Seite 9/18

Abbildung 7 Ablauf Kellular Task Seite 10/18

# **Tabellenverzeichnis**

Tabelle 1 Zeitaufnahme Leibniz Seite 16/18

Tabelle 2 Zeitaufnahme Kellular Seite 17/18