Pi Rechner

Von Hüseyin Dogrucam

Aufgabenstellung

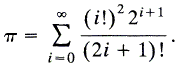
Für das Fach Embedded Systems wurde uns die Aufgabe erteilt, mithilfe eines Algorithmus die Kreiszahl Pi auf unserem Edu-Board zu berechnen. Dabei soll der aktuelle Pi Werts alle 500ms angezeigt werden. Durch Tastendrucks soll der Rechner gestartet, gestoppt und zurückgesetzt werden. Die Kommunikation zwischen den Tasks soll mittels EventBits und/oder TaskNotification stattfinden und soll mindestens 2 Tasks beinhalten. Damit wir sehen können wie schnell der Algorithmus ist, sollen wir einen Hardware Timer einprogrammieren, welcher die Zeit messen soll, bis wir Pi auf genau die 5.Stellen nach dem Komma berechnet haben. Danach soll der Rechner aber immer noch weiter rechnen

Der Algorithmus

Bevor ich ans Programmieren ging, suchte ich nach einem Algorithmus, welcher für die Berechnung von Nachkommastellen von Pi auf einem Mikroprozessor geeignet ist. Als Vorgabe gab ich mir, dass ich nicht mit Potenzen und Wurzeln arbeiten wollte, weil ich nicht wusste, wie schnell der Mikroprozessor damit rechnen konnte.

Auf meiner Suche stoss ich auf den Tröpfelalgorithmus bzw. Spigot Algorithmus auf Englisch. Der Tröpfelalgorithmus für Pi wurde 1991 von Stanley Rabinowitz entdeckt und 1995 von Stanley Rabinowitz und Stan Wagon untersucht und veröffentlicht.

Die Mathematische Formel für den Tröpfelalgorithmus für Pi sieht so aus:



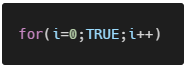
Dieser kann noch vereinfacht für Pi Halbe dargestellt werden.



So sieht sieht Pi Halbe in der vereinfachten Horner Form aus.



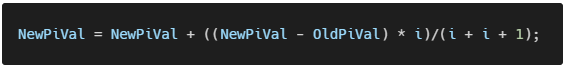
Um Pi zu berechnen nehme ich die vereinfachte Form, welche Pi Halbe darstellt. Zu sehen ist, dass es sich um eine unendlich lange Summe handelt. Begonnen wird bei «i = 0» und bei jeder neuen Addition soll «i» um eins grösser werden. Um dies in Code zu realisieren eignet sich eine For-Schleife.



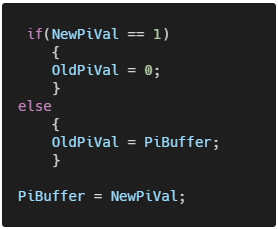
Das Ausrufezeichen neben dem «i» steht in der Mathematik für die Fakultät. Die Fakultät ist eine Funktion, die einer natürlichen Zahl das Produkt aller natürlichen Zahlen (ohne Null) kleiner und gleich dieser Zahl zuordnet, Die Fakultät von Null ist Eins. Das heisst für unseren Algorithmus, den Bruch der vorherigen Berechnung müssen wir mit dem Bruch welcher durch das neue «i» ohne Fakultät entsteht multiplizieren. Dieser neue Bruch wird dann zum bestehenden Wert dazu addiert.

1.Durchgang 2.Durchgang 3.Durchgang



In Code sieht das wie es wie folgt aus.

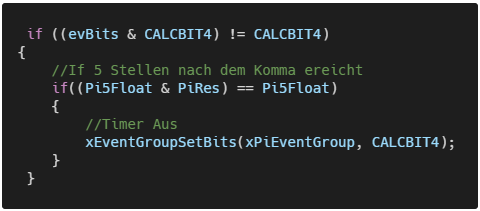
«NewPiVal» beginnt bei einem Reset mit dem Wert 1 während alle anderen Werte 0 sind. Damt wir beim ersten durchlauf auf eins kommen, verwende ich eine bedingte Anweisung, welche «OldPiVal» auf null setzt, wenn «NewPiVal» noch eins ist. Ansonsten wird «OldPiVal» auf den vorherigen «NewPiVal» gesetzt durch einen Buffer.



Danach können wir «NewPiVal» mit sich selbst addieren und erhalten Pi.

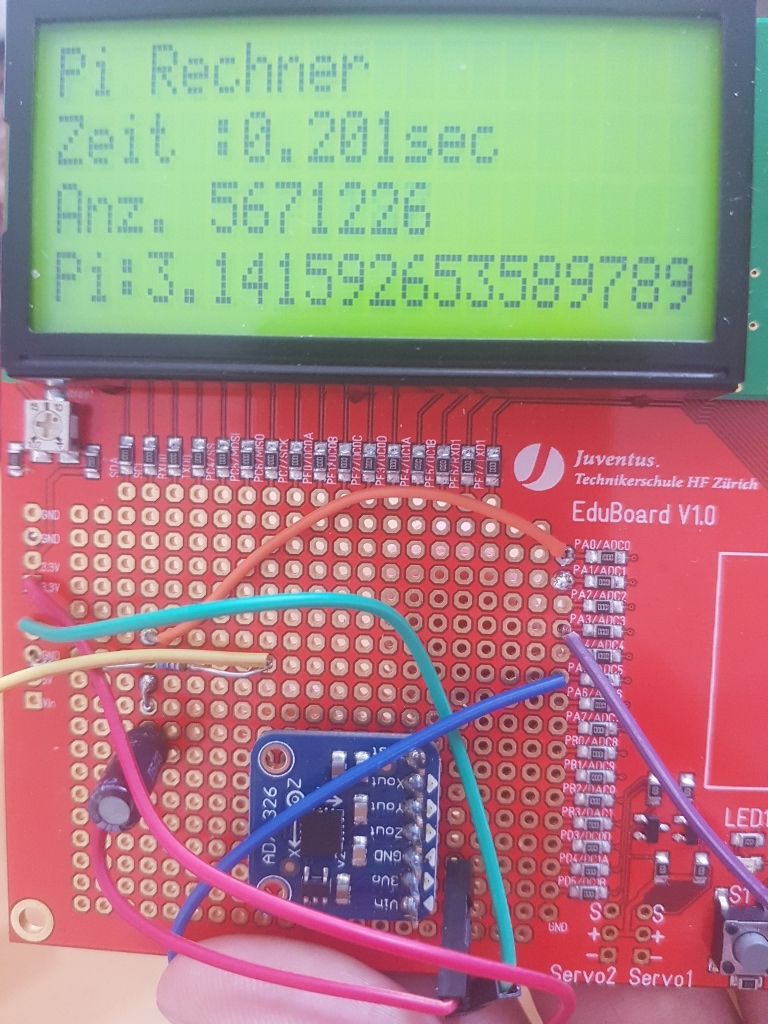
Die Tasks und Eventbits

Für dieses Projekt arbeite ich mit 3 Tasks, «vCalcPi», «vInterface» und «vTimeCountTask». «vCalcPi» ist für die Berechnung von Pi zuständig. Dieser Task hat die Priorität 1 und damit die Tiefste, weil er für die Berechnungen am meisten Ressourcen braucht. Zu beginn des Tasks, bevor er in eine Schleife fällt, initialisiert er alle Variablen, welche wir für diesen Task brauchen. Sobald er in die Schleife gefallen ist, wartet er auf das Eventbit «CalcBit1». Wenn dieses noch nicht da ist, geht er in das «TaskDelay». Sobald «CalcBit1» gesetzt ist, geht der Task in den Algorithmus. Um den Algorithmus zu pausieren muss man das «CalcBit1» clearen. Wenn das Eventbit «CalcBit0» gesetz wird, werden alle Variablen zurückgesetzt und es wird wieder auf «CalcBit1» gewartet. In der Schleife des Algorithmus, wird kontrolliert, ob das berechnete Pi bis auf 5 Stellen nach dem Komma korrekt ist. Wenn die zutrifft, wird das Eventbit «CalcBit4» gesetzt und die Kontrolle wird übersprungen.



Damit ich eine möglichst genaue Berechnung zustande bringen kann, habe ich die Float64 Library implementiert. Diese hat zurfolge, dass ich keine normalen Operationen ausführen kann, weswegen ich die Float64 spezifischen Operationen verwenden musste. Der Wert für «i» wird immer mittels einer Queue zum «vInterface» Task gesendet. Wenn die Anzahl Durchgänge(«i») das Maximum einer uint\_32\_t Variable erreicht hat(2^32), wird der Algorithmus gestoppt und das Eventbit «CalcBit2» gesetzt. Dieses Eventbit sorgt dafür, dass auf dem Monitor das Wort Max neben der Anzahl von Durchgängen steht.

Die Ansteuerung des Monitors geschieht im Task «vInterface». Auf dem Monitor wird der Titel, die aktuelle Zeit, die aktuelle Anzahl Durchgänge und der aktuelle Wert von Pi angezeigt. Ebenfalls wird neben der Anzahl Durchgänge noch angezeigt, ob man den Algorithmus pausiert oder zurückgesetzt hat oder er das Maximum erreicht hat.



Starten, pausieren und zurücksetzten kann man mit dem Buttons 1,2 und 3. Wenn diese gedrückt werden, setzten und clearen sie Eventbits.

Im Task «vTimeCountTask» initialisieren wir den Timer. Dieser wird ohne einen Clock initialisiert, damit er nicht anfängt zu zählen, bevor wir überhaupt Rechnen wollen. Sobald die der Button 1 gedrückt wird, wird der Eventbit «CalcBit3» gesetzt. Dieses Eventbit lässt uns das «CalcBit0» setzen. Wenn wir noch nicht 5 Stellen nach dem Komma von Pi berechnet haben und damit «CalcBit4» noch nicht gesetzt haben, schalten wir ebenfalls einen Clock ein. Nun wird immer nach einer Millisekunde ein Interrupt erzeugt, in welchem wir eine Float Variable um 0.001 hochzählen. Wenn «CalcBit4» gesetzt wird, wird der Clock ausgeschalten und die Float Variable wird nicht mehr grösser. Weil wir jetzt den Clock nicht mehr brauchen erhöhen wir das Delay dieses Tasks.

Erkenntnisse

Mit meinem Algorithmus braucht der Mirkrokontroller durchschnittlich 0.206 Sekunden, um Pi auf die 5.Stelle nach dem Komma genau zu berechnen. Dies liest darauf schliessen, dass seine Rechenleistung für diese Aufgabe, mehr als genug ist. Theoretisch sollte er in der Lage sein, innert 10 Sekunden 250 Stellen nach dem Komma zu berechnen. Das Problem dabei ist, dass ab einem gewissen Punkt die Variablen zu klein sind und er nur noch mit 0 rechnet.

Einige Probleme ergaben sich doch, durch die Rechenleistung. Globale Integer scheinen durch die hohe beanspruchte Leistung des Algorithmus nicht korrekt zu speichern. Sie hatten zum Beispiel ständig den gleichen Wert obwohl sie sich verändern sollten. Ebenfalls fror mir das Programm ein, sobald ich ein Eventbit aus dem ISR setzen wollte oder die Funktion «sprintf» im ISR abrufen wollte. Solche Fehler zwangen mich dazu eine Queue für «i» zu verwenden oder eine Float für den Timer Counter. Alles in allem bin ich sehr zufrieden mit meiner Arbeit.