Juventus
Technikerschule HF
www.technikerschule.ch

Semesterarbeit

# PI Berechnung und Unterschied Leibniz | Nilakantha

Microchip Studio

# Inhaltsverzeichnis

# 1 Aufgabenstellung

- Realisierung der Reihe von Gottfried Wilhelm Leibniz
- Realisierung einer weiteren Methode Pi zu berechnen.
- Tasks mittels Eventbits in den Code realisieren.
- Steuertask erstellen, um zwischen den beiden Methoden zu switchen.

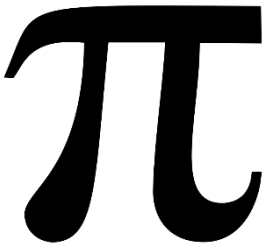

**Abbildung 1**

## 1.1 Leibniz Serie gegenüber Nilakantha

Als ich mir einen weiteren Algorithmus herausgesucht habe, erstellte ich mir von allen Möglichkeiten ein Vergleich. Dabei habe ich angeschaut welche Rechnungsmethode die vereinzelten Reihen verwenden. Zum Entschluss bin ich dann mittels der Effizienz der Methode von Somayaji Nilakantha gekommen.

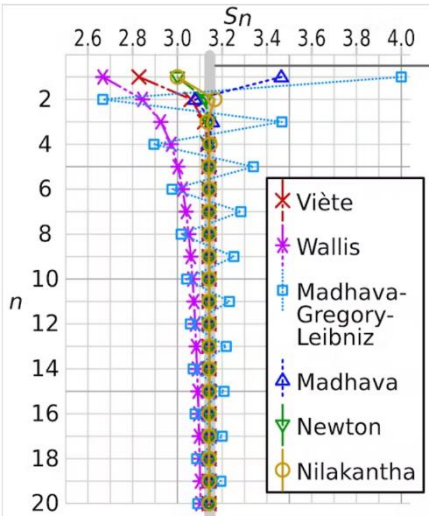


**Abbildung 2**

# 2 Sollzustand

Es soll per Auswahl durch Eventbits zweier verschiedener Algorithmen ausgewählt werden, um eine Annäherung auf PI berechnen zu können.

Mittels eines Timer soll die Zeit gemessen werden, bis die Annäherung fünf Stellen nach dem Komma erreicht hat.

Algorithmus muss via jeweiligem Tastendruck eine Start und Stop Funktion haben.

Reset wird durch einem weiteren Tastendruck gewährleistet.

Das wechseln beider Algorithmen soll mittels Switch funktion über einen Taster erfolgen.

# 3 Nilakantha Reihe

## 3.1 Mathematische Berechnung

Die mathematische Berechnung ähnelt sich sehr stark der Leibniz Reihe, höchst wahrscheinlich deswegen, weil diese ca. 120 Jahre vor dem Mathematiker der Leibniz Reihe berechnet wurde.

Um die Berechnung besser zu verstehen erstellte ich ein Excel in der ich banal die Rechenschritte simulierte.

**Abbildung 3**

Das Folgende Bild zeigt der Anfang der Annäherung:

$$\pi = 3 + \frac{4}{2*3*4} - \frac{4}{4*5*6} + \frac{4}{6*7*8} - \frac{4}{8*9*10} \cdots$$

**Abbildung 4**

```
(-1)^(AX+1)/(2*AX*(2*AX+1)*(2*AX+2))
```

**Abbildung 5**

| | A | B | C | |
|---|---|---|---|---|
| Nilkantha Reihe Calc. PI | | | | |
| 1 | | 0.04166667 | 3 | |
| 2 | | -0.0083333 | 3.16666667 | CX+4*BX |
| 3 | | 0.00297619 | 3.13333333 | CX+4*BX |
| 4 | | -0.0013889 | 3.1452381 | CX+4*BX |
| 5 | | 0.00075758 | 3.13968254 | CX+4*BX |
| 6 | | -0.0004579 | 3.14271284 | CX+4*BX |
| 7 | | 0.00029762 | 3.14088134 | CX+4*BX |
| 8 | | -0.0002042 | 3.14207182 | CX+4*BX |
| 9 | | 0.0001462 | 3.14125482 | CX+4*BX |
| 10 | | -0.0001082 | 3.14183962 | CX+4*BX |
| 11 | | 8.2345E-05 | 3.14140672 | CX+4*BX |
| 12 | | -6.41E-05 | 3.1417361 | CX+4*BX |
| 13 | | 5.0875E-05 | 3.14147969 | CX+4*BX |
| 14 | | -4.105E-05 | 3.14168319 | CX+4*BX |
| 15 | | 3.3602E-05 | 3.14151899 | CX+4*BX |
| 16 | | -2.785E-05 | 3.14165339 | CX+4*BX |
| 17 | | 2.3343E-05 | 3.14154199 | CX+4*BX |
| 18 | | -1.976E-05 | 3.14163536 | CX+4*BX |
| 19 | | 1.6869E-05 | 3.14155633 | CX+4*BX |
| 20 | | -1.452E-05 | 3.14162381 | CX+4*BX |

**Abbildung 6**

Die Berechnung habe ich fünfhundert mal durchgeführt und dabei bin ich zum Entschluss gekommen, dass das Ergebnis bei ± 3 stagniert.

Die Annäherung mittels der Excelliste kann man mit zwei einfachen Formeln berechnen.

$$3 + 4 * \sum_{j=1}^{n} \frac{(-1)^{j+1}}{2j(2j+1)(2j+2)}$$

**Abbildung 7**

| | | | |
|---|---|---|---|
| 479 | 477 | 1.1481E-09 | 3.141592651 |
| 480 | 478 | -1.141E-09 | 3.141592656 |
| 481 | 479 | 1.1338E-09 | 3.141592651 |
| 482 | 480 | -1.127E-09 | 3.141592656 |
| 483 | 481 | 1.1198E-09 | 3.141592651 |
| 484 | 482 | -1.113E-09 | 3.141592656 |
| 485 | 483 | 1.1059E-09 | 3.141592651 |
| 486 | 484 | -1.099E-09 | 3.141592656 |
| 487 | 485 | 1.0923E-09 | 3.141592651 |
| 488 | 486 | -1.086E-09 | 3.141592656 |
| 489 | 487 | 1.0789E-09 | 3.141592651 |
| 490 | 488 | -1.072E-09 | 3.141592656 |
| 491 | 489 | 1.0657E-09 | 3.141592651 |
| 492 | 490 | -1.059E-09 | 3.141592656 |
| 493 | 491 | 1.0528E-09 | 3.141592651 |
| 494 | 492 | -1.046E-09 | 3.141592656 |
| 495 | 493 | 1.04E-09 | 3.141592652 |
| 496 | 494 | -1.034E-09 | 3.141592656 |
| 497 | 495 | 1.0275E-09 | 3.141592652 |
| 498 | 496 | -1.021E-09 | 3.141592656 |
| 499 | 497 | 1.0152E-09 | 3.141592652 |

**Abbildung 8**

## 3.2 Nilakantha in C

Damit ich die Nilakantha Berechnung ins C Programm implementieren konnte, musste ich verschiedene Hilfsvariabeln in mein NilakanthaTask setzen.

```
float PIH = 0;
double n = 2;
double sign = 1;
long compare = 0;
const long pisix = 314159;
```

**Abbildung 9**

Somit ist sichergestellt, dass ich immer den aktuellsten PI-Wert (piact) entzehen kann und auf dem Display ausgeben kann. Das selbe bei der Leibniz Variante.

```
PIH = PIH + (sign/(((n)*(n+1)*(n+2))));
sign = sign * (-1);
n += 2;
pi = (PIH*4)+3;
```

**Abbildung 10**

| |
|---|
| 0 = 0 + (1/(((2)*(n+1)*(n+2)))) |
| 1 = 1 * (-1) |
| 2 += 2 |
| 1 = (0*4)+3 |
| |
| 0 = 1*100000 |

**Abbildung 11**

Da am Anfang bei der Nilakantha Methode direkt PI = 3.14159 gesetzt wurde, machte ich mich auf die Problemsuche. Der erste Versuch war der Optimizer aus dem Compiler über Properties zu verbannen, jedoch ohne Erfolg. Bei weiterem durchsuchen und debuggen mit Breakpoints bemerkte ich, dass die Berechnung nie durchlaufen wird und dadurch nicht ausgegeben wird. Nach weiterem durchsuchen der Dokumentationen im Internet, kam ich nochmals auf das Diagramm von 1.1 Leibniz Serie gegenüber Nilakantha. In diesem Diagramm wird aufgezeigt, dass Nilakantha einer der schnellsten Methode um eine Annäherung von PI zu erreichen.

Also kam ich zum Entschluss einen Delay rein zu hauen, obwohl das eine Verzögerung generiert in dem das Messen der Zeit ungenauer wird, habe ich mich für diese Variante entschieden.

```
compare = pi * 100000;
vTaskDelay(10);

if (compare == pisix)
```

**Abbildung 12**

# 4 Code

## 4.1 Finite State Machine Diagramm

Um das Verfahren des ganzen Programms visualisieren zu können habe ich, wie in der Aufgabenstellung erwähnt ein Finite State Machine Diagramm erstellt.

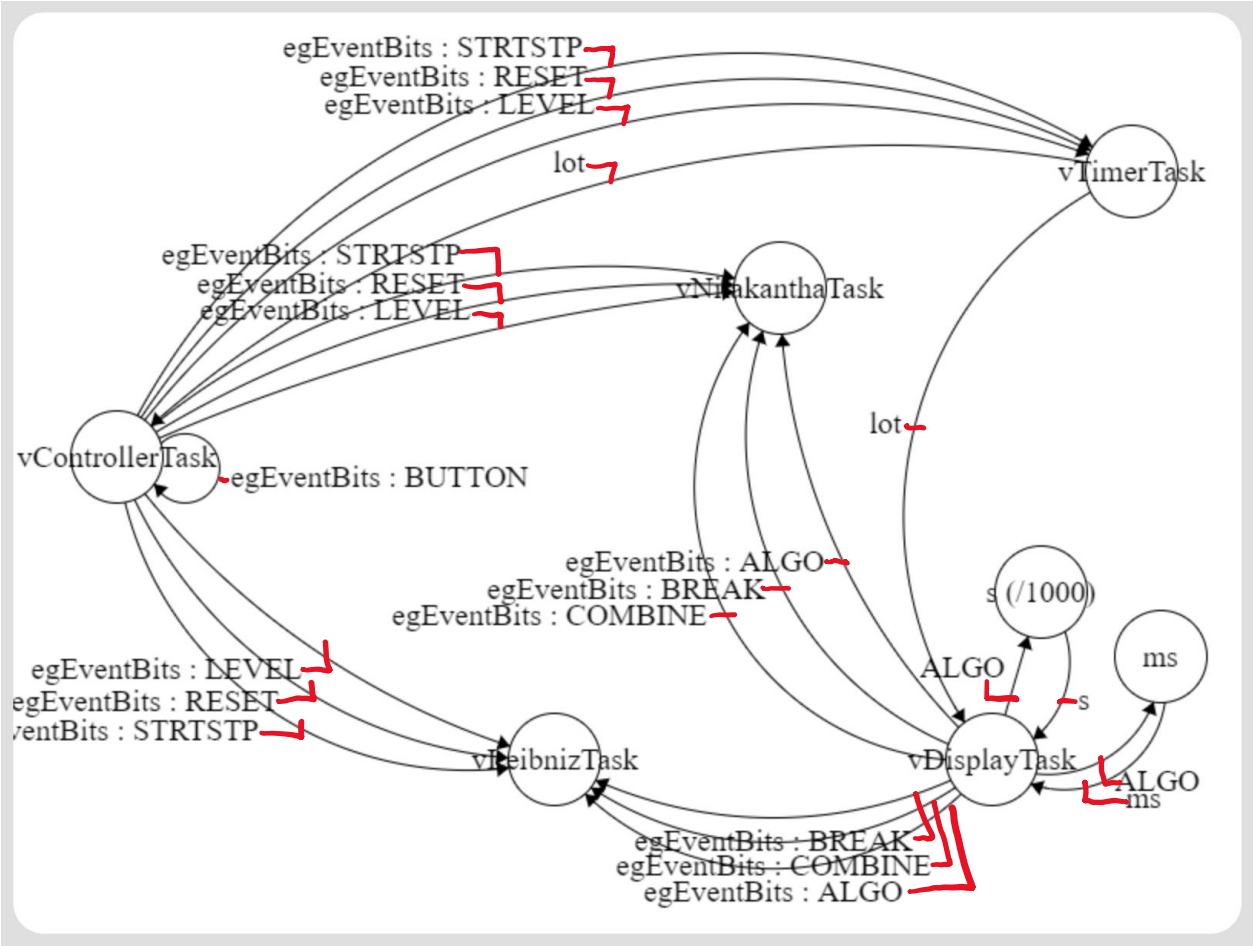


**Abbildung 13**

Vom Diagramm kann man daraus schliessen, dass der Controller Task die Top Entity ist wärend der Display Task nur mit den beiden Algorithmen interagiert. Die Lengh of Time (lot) kehrt vom Timer Task über Controller Task und wird danach als String zum Display geschrieben.

Das Diagramm habe ich mit einem FSM Designer aus dem Internet erstellt.

Link: https://madebyevan.com/fsm/

## 4.2 Aufgabe EventBits und Tasks

| | |
|---|---|
| STRTSTP | = Wird für Button1 und Button 2 die StartStop func verwendet. |
| RESET | = Reset func um den Timer und die Berechnung auf Basis zu setzen. |
| COMBINE | = Func damit die Berechnung zusammengefasst werden kann. |
| LEVEL | = EventBit vergleicht vorgegebener pisix mit COMBINE. |
| BREAK | = Bricht Berechnung ab, womit der Wert für COMBINE ausgelesen wird. |
| ALGO | = Switched vom Leibniz zu Nilakantha und umgekehrt. |
| BUTTON | = EventBit für Controller Task Button einlesen. |

| | |
|---|---|
| vDisplayTask | = Dafür zuständig, alle nötigen Informationen um die Aufgabenstellung auf das Display zu schreiben. |
| vTimerTask | = Stellt die Struktur des Timer dar. |
| vControllerTask | = Kontrolliert und koordiniert die anderen Tasks. |
| vLeibnizTask | = Berechnet Annäherung von PI mittels Leibniz Reihe. |
| vNilkanthaTask | = Berechnet Annäherung von PI mittels Nilakantha Reihe. |

## 4.3 Berechnungszeit

Schlussendlich habe ich die Zeit der beiden Algorithmen mittels des TimeTasks gemessen.
Die Ergebnisse:

| Mode: Leibniz | Mode: Nilakantha |
|---|---|
| Time: 14s | Time: 440ms |
| PI: 3.141597 | PI: 3.1415927 |

**Abbildung 14**

Zu erkennen ist offensichtlich, die Berechnung von Leibniz ist im Mittelwert ca. 13 mal langsamer als die Nilakantha Reihe. Die Schlussfolgerung der zweier Berechnungen kann mehrere Punkte bedeuten;

- Nilakantha lässt sich einfacher und weniger komplex berechnen
- Mein Leibniz Code habe ich nicht optimiert
- Die allgemeine Aufstellung meines Programms könnte man effizienter gestallten

## Persönliches Fazit

Im Grossem und Ganzem ist das Projekt gut verlaufen, dazu verschweige ich nicht welche Probleme mir auf diesem Weg angefallen sind.

+ Neues erlernt
+ Umgang mit dem MicrochipStudio (wenn es dann mal tut was es sollte)
+ Verständnis der C Programmsprache
+ Nilakantha konnte mit ein oder zwei Schwierigkeiten implementiert werden

Verbesserungsvorschläge für nächstes Mal.

- Mehr in die Basics vertiefen
- Immernoch grosse Verständnisschwierigkeiten mit den Grundlagen von C im Allgemeinen
- Struktogramm am Anfang erstellen
- Timemanagement (ab wann starte ich mit welchen Punkten)
- Von Anfang an eine **klare** Struktur im Code

# Literaturverzeichnis

Die genutzte Literatur dieser Arbeit wurden durch die Juventus Technikerschule, uns abgegebenen, Skripte zur Verfügung gestellt.