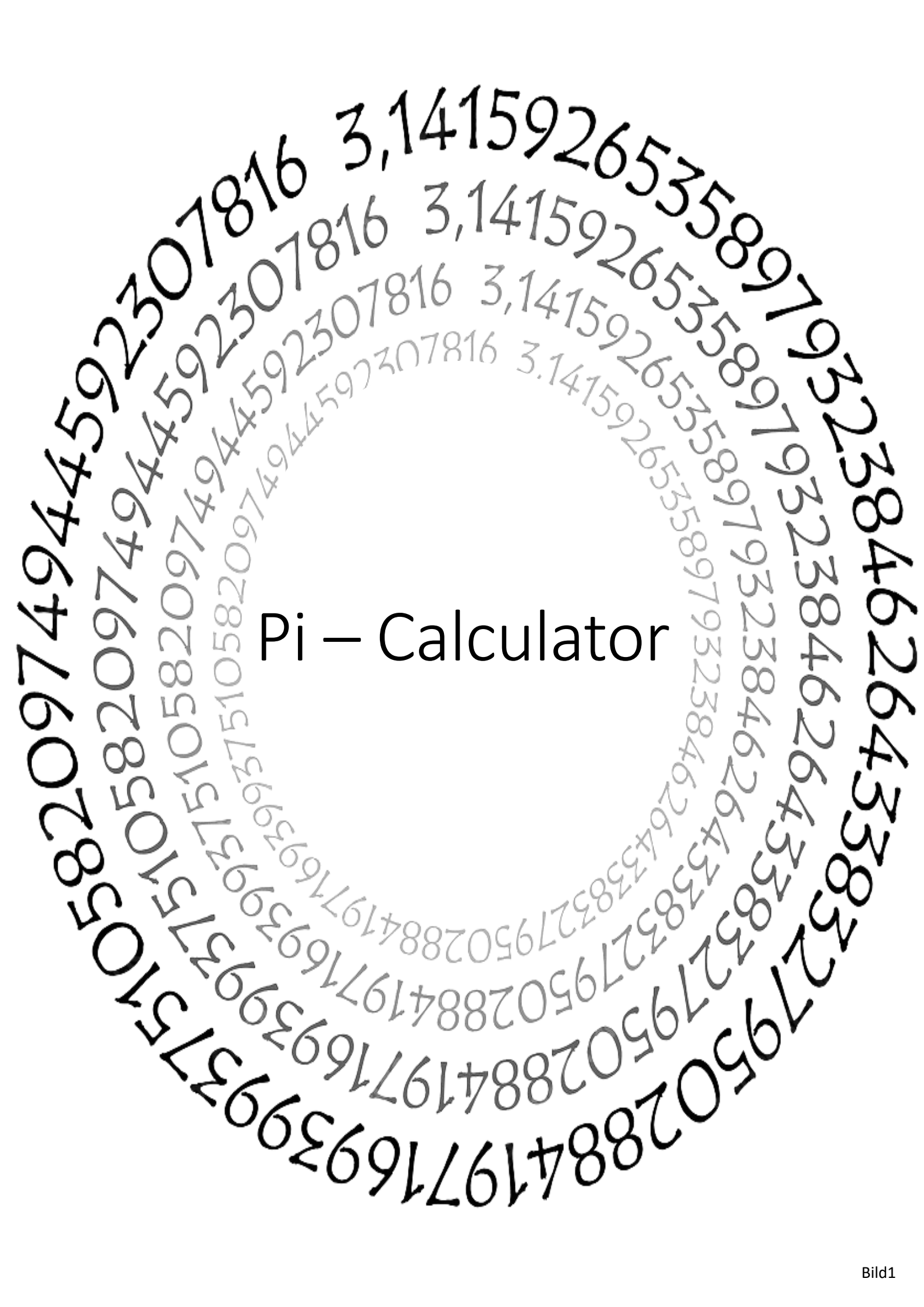


Bild1

Inhaltsverzeichnis

1. Pi Algorithmen

1.1. Leibnitz

$$\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}.$$

Bild2

Hier haben wir die Formel für die Leibnitz – Reihe, für die Vervollständigung.

1.2. Nilakantha

Für den zweiten Algorithmus habe ich mich für den Nilakantha entschieden. Denn dies soll eine Rhein sein die sehr schnell konvergiert ist und dies sollte eine wichtige Eigenschaft sein wenn man Pi schon berechnen soll, mit der Genauigkeit natürlich.

$$\frac{\pi - 3}{4} = \frac{1}{2 \times 3 \times 4} - \frac{1}{4 \times 5 \times 6} + \frac{1}{6 \times 7 \times 8} - \cdots$$

Bild3

Die ist die Formel die Nilakantha entwickelt hat. Er war ein indischer Mathematiker und Astronom und hat diese Formel im 15. Jahrhundert entwickelt.

$$\frac{\pi - 3}{4} = \frac{1}{24} - \frac{1}{120} + \frac{1}{336} - \frac{1}{720} + \frac{1}{1320} - \frac{1}{2184} + \frac{1}{3360} - \frac{1}{4896} + \frac{1}{6840} - \frac{1}{9240}$$

Bild4

Wenn man die Formel soweit erweitert ist die Zahl Pi schon in diesem Moment auf drei Nachkommastellen genau, dementsprechend kann man sich selber denken wie schnell es dann auf weitere auszuführen ist.

2. Tasks

2.1. Leibnitz – Task

Im Leibnitz Task wird die Berechnung von Pi mit der Leibnitz – Reihe vorgenommen. Dazu wird hier auch noch die Zurücksetzung von den entsprechenden Variablen vorgenommen falls der Reset Knopf betätigt wurde. Darüber hinaus wird auch noch die benötigte Zeit abgefragt die die Reihe benötigt hat um Pi auf fünf stellen genau zu berechnen.

```
if(!(xEventGroupGetBits(xControlleventgroup) & SWITCH_ALGO_BIT)){
    //Reset
    if(xEventGroupGetBits(xControlleventgroup) & RESET_BIT){
        sign        = 1;
        counter     = 0;
        k           = 0;
        LeibnizPi   = 0.0;
        xEventGroupClearBits(xControlleventgroup, RESET_BIT);
    }
    if(xEventGroupGetBits(xControlleventgroup) & START_BIT) {
        LeibnizPi += sign / (2.0 * k + 1.0);
        //Wechseln des Vorzeichen für nächste Stelle
        sign = -sign;
        k++;
        counter++;
        //Abgleich für die Zeit
        if(Timefound == 0){
            if (fabs((LeibnizPi * 4) - 3.14159) < 0.00001) {
                clock_t end_time = clock(); // Endzeit erfassen
                double time_taken1 = (double)(end_time - start_time1) / CLOCKS_PER_SEC;
                printf("Benötigte Zeit Leibnitz: %.6f Sekunden\n", time_taken1);
                gctime_takenL = time_taken1;
                Timefound = 1;
            }
```

Bild5

Dies ist der Code – Ausschnitt des von den oben erwähnten Funktionen. Wie man sehen kann Funktionieren alle Funktionen nur dann wen auch dieser Task ausgewählt worden ist. Dementsprechend kann z.B. nur der Wert zurückgesetzt werden in dessen Task man sich gerade befindet, deshalb werden nie beider Werte zurückgesetzt.

2.2 Nilakantha – Task

```
while (1) {

if((xEventGroupGetBits(xControlleventgroup) & SWITCH_ALGO_BIT)){
    //Reset
    if(xEventGroupGetBits(xControlleventgroup) & RESET_BIT){
        n = 1;
        NilakanthaPi = 3.0;
        xEventGroupClearBits(xControlleventgroup, RESET_BIT);
}
    if(xEventGroupGetBits(xControlleventgroup) & START_BIT) {
        NilakanthaPi += (n % 2 == 0 ? -4.0 : 4.0) / ((2.0 * n) * (2.0 * n + 1.0) * (2.0 * n + 2.0));
        n++;
        //Abgleich für die Zeit
        if(Timefound == 0){
            if (fabs(NilakanthaPi - 3.14159) < 0.00001) {
                clock_t end_time = clock(); // Endzeit erfassen
                double time_taken2 = (double)(end_time - start_time2) / CLOCKS_PER_SEC;
                printf("Benötigte Zeit Nilakantha: %.6f Sekunden\n", time_taken2);
                gctime_takenN = time_taken2;
                //gctime_takenN = time_taken;
                Timefound = 1;
            }
        }
    }
}
```

Bild6

Wie man hier unschwer erkennen kann ist es ziemlich genau gleich aufgebaut wie beim Leibnitz Task nur das hier die Formel des Nilakantha verwendet wird.

2.3. Steuerung

```
void ControllingTask(void *param){
    for(;;) {
        if(button_get_state(SW0, true) == SHORT_PRESSED) {
            xEventGroupSetBits(xControlleventgroup, START_BIT);
            xEventGroupClearBits(xControlleventgroup, STOP_BIT);
        }
        if(button_get_state(SW1, true) == SHORT_PRESSED) {
            xEventGroupSetBits(xControlleventgroup, STOP_BIT);
            xEventGroupClearBits(xControlleventgroup, START_BIT);
        }
        if(button_get_state(SW2, true) == SHORT_PRESSED) {
            xEventGroupSetBits(xControlleventgroup, RESET_BIT);
            Timefound = 0;
        }
        if(button_get_state(SW3, true) == SHORT_PRESSED) {
            if(AlgoBit == 0){
                xEventGroupSetBits(xControlleventgroup, SWITCH_ALGO_BIT);
                AlgoBit = 1;
            }
            else{
                xEventGroupClearBits(xControlleventgroup, SWITCH_ALGO_BIT);
                AlgoBit = 0;
            }
        }
        vTaskDelay(100/portTICK_PERIOD_MS);
    }
```

Bild7

Wie man hier erkennen kann ist es beim Steuerungstask sehr simpel gehalten worden. Denn es sind einfache Tastenabfragen und wenn diese erfüllt sind wird das entsprechende Event Bit gesetzt, sodass danach die oben entsprechenden Task darauf reagieren können und ihre vorausgesetzten Bedingungen verfolgen können.

2.4. Interface

Zum Ende hin habe ich noch einen zusätzlichen Task hinzugefügt, dieser Regelt die Datenübertragung auf das LCD sodass man auch etwas sehen kann.

```
    }
    if(!(xEventGroupGetBits(xControlleventgroup) & SWITCH_ALGO_BIT)){
        lcdDrawString(fx32M, 10, 30, "PI-Calculator", WHITE);
        lcdDrawString(fx24M, 10, 100, "Leibnitz-Pi:", GREEN);
        lcdDrawString(fx24M, 10, 130, "Calculate-Time:", YELLOW);
        sprintf(Leibniz, "%.10f", LeibnizPi*4);
        lcdDrawString(fx24M, 200, 100, Leibniz, GREEN);
        sprintf(TimeLeibniz, "%f sek", gctime_takenL );
        lcdDrawString(fx24M, 200, 130, TimeLeibniz, YELLOW);
    }

    lcdDrawRect(0, 219, 119, 319,   GRAY);
    lcdDrawString(fx16M, 10, 280, "S0 = Ein", WHITE);
    lcdDrawRect(120, 219, 239, 319, GRAY);
    lcdDrawString(fx16M, 130, 280, "S1 = Aus", WHITE);
    lcdDrawRect(240, 219, 359, 319, GRAY);
    lcdDrawString(fx16M, 250, 280, "S2 = Reset", WHITE);
    lcdDrawRect(360, 219, 479, 319, GRAY);
    lcdDrawString(fx16M, 370, 280, "S3 = Switch", WHITE);

    lcdUpdateVScreen();
    vTaskDelay(500/portTICK_PERIOD_MS);
```

Bild8

Hier werden die verschiedenen Anzeigen erstellt. Damit nicht immer der Leibnitz und der Nilakantha angezeigt werden wurde hier zusätzlich noch eine if abfrage mit den Event Bits erstellt sodass sich diese 2 niemals in die Querer kommen können. Für die Sachen die durchgehend angezeigt werden und es dementsprechend nicht nötig ist wurde einfach die entsprechenden Positionen bestimmt und ausgegeben.

2.4. Task Priorisierung

```
xTaskCreate(LeibnizTask,        //Subroutine
        "LeibnizTask",          //Name
        2*2048,                 //Stacksize
        NULL,                   //Parameters
        1,                      //Priority
        NULL);                  //Taskhandle

xTaskCreate(NilakanthaTask,     //Subroutine
        "NilakanthaTask",       //Name
        2*2048,                 //Stacksize
        NULL,                   //Parameters
        1,                      //Priority
        NULL);                  //Taskhandle

xTaskCreate(ControllingTask,    //Subroutine
        "ControllingTask",      //Name
        2*2048,                 //Stacksize
        NULL,                   //Parameters
        10,                     //Priority
        NULL);                  //Taskhandle

xTaskCreate(InterfaceTask,      //Subroutine
        "ControllingTask",      //Name
        2*2048,                 //Stacksize
        NULL,                   //Parameters
        7,                      //Priority
        NULL);                  //Taskhandle
```

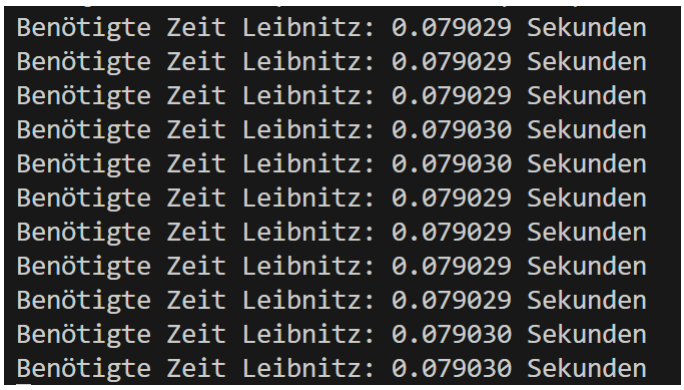
Bild9

Wie man hier erkennen kann haben die Berechnungs Task die niedrigste Priorisierung und der Steuerungstask die höchste. Da ich nicht genau wusste wie schnell der LCD Task upgedatet werden muss ist dort einfach eine 7 geworden.

3. Event Bits

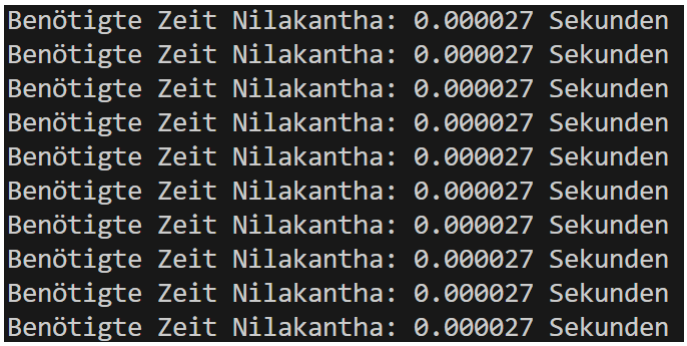
Ich habe es bei den Event Bits sehr simpel gehalten, denn ich habe diese eingesetzt das sie für die Entsprechenden Sachen eingeschaltet oder ausgeschaltet werden. Dementsprechend erscheinen diese nachher nur in abfragen wo geschaut wird ob sie gesetzt oder eben nicht gesetzt sind. In den entsprechenden Voraussetzung wird dann etwas anderes ausgeführt. Zum Beispiel für die Entscheidung welcher Algorithmus nun ausgeführt werden soll ist ein Event Bit entstanden. Wenn es nicht gesetzt ist wird der Leibnitz Algorithmus ausgeführt und sonst der Nilakantha.

4. Die Zeitmessung



Die sind die Resultate der Zeitmessung für den Leibnitz Algorithmus. Wie man hier sehen kann braucht es im Schnitt etwa 79ms um den Wert auf 5 Kommastellen zu berechnen.

Bild10



Dies sind die Ergebnisse für den Nilakantha Algorithmus. Wie man hier unschwer erkennen kann braucht er immer 27µs um auf 5 Komastellen genau zu rechnen.

Bild11

Ich nehme an das der Wert noch genauer Ausgerechnet werden könnte, indem man die Tick rate verändern würde, indem man sie zum Beispiel erhöhen würde, denn im Moment läuft diese auf 1ms genau. Dementsprechend kommt dieser wahrscheinlich gar nicht hinterher um ein genaues Ergebnis zu ermöglichen. Aber mit dem jetzigem Stand sind dies die Ergebnisse die ausgegeben werden.

Zu sehen ist aber deutlich das der Nilakantha um ein vielfaches schneller ist als der Leibnitz. Wen man so sagen will ist der Nilakantha etwa 292000-mal schneller als der Leibnitz.

5. Fazit

Im Grunde war es das erste mal das ich die Art mit Task zu schaffen wirklich angewendet habe und ich muss sagen als man endlich den Überblick hate wie man dies anweden muss war es sehr aufschlussreich.

Die Berechnung Task waren am simpelsten was mir besonders mühe gemacht hat war am Anfang die Event Bit einzugliedern und anzuwenden da ich mit denen so überhaupt nicht umgehen konnte. Dementsprechend habe ich auf ChatGPT zurückgegriffen, bedauerlicher weise konnte dies mir auch nicht weiter helfen, sodass ich sehr viel Zeit bei diesem Bereich vertrödelt hatte.

Die Zeitmessung hatte ich relativ schnell, doch nach mehrmaligem widerholen viel mir auf das sich die Zeit nie zurücksetzte sodass, im optischen, der Rechner immer länger brauchte um Pi zu berechnen. Das Problem war das ich diesen Befehl xTaskGetTickCount() nur einmal gesetzt hatte und er dementsprechend nie zurückgesetzt wurde.

Das grösste Problem war allerdings, dass bei mir nach einiger Zeit das Raufladen auf Git nicht mehr funktionierte wie auch das Complimieren, am Schluss ging es so weit das ich den ganzen Code in ein neues Kopiert habe und von dort aus weitergearbeitet habe. Dementsprechend habe ich nun 2 Git repository.

Quellenverzeichnis

- https://de.wikipedia.org/wiki/Kreiszahl
- https://chatgpt.com/
- http://www.maeckes.nl/Formule%20voor%20pi%20(Nilakantha)%20DE.html
- https://de.wikipedia.org/wiki/Leibniz-Reihe

Bildverzeichnis

- Bild1: https://as2.ftcdn.net/v2/jpg/02/24/78/63/1000_F_224786373_7iWeGLUFhTmMyM3lZvW6vXPJOBgsQkCN.jpg
- Bild2: https://wikimedia.org/api/rest_v1/media/math/render/svg/ef36d20e884bc257c537b4610511fedac46f2e48
- Bild3: http://www.maeckes.nl/clips/(%CF%80%20%E2%88%92%203)%20%E2%88%95%204%20=%201%20%E2%88%95%202%C3%973%C3%974%20%E2%88%92%201%20%E2%88%95%204%C3%975%C3%976%20+%20%E2%8B%AF%20.gif
- Bild4: http://www.maeckes.nl/clips/(%CF%80%20%E2%88%92%203)%20%E2%88%95%204%20=%201%20%E2%88%95%2024%20%E2%88%92%201%20%E2%88%95%20120%20+%201%20%E2%88%95%20336%20%E2%88%92%201%20%E2%88%95%20720%20+%20%E2%8B%AF%20.png
- Bild5: Eigen Bild
- Bild6 : Eigen Bild
- Bild7: Eigen Bild
- Bild8: Eigen Bild
- Bild9: Eigen Bild
- Bild10: Eigen Bild
- Bild11: Eigen Bild