

PI CALCULATOR

Nicolas Knopf

JUVENTUS SCHULE Embedded Systems

Inhalt

Aufgabenstellung.....	2
Algorihtmus	3
Tasks	4
Calculations Taks	4
Buttonhandler Task	5
Interface Task	6
Count Time Task	6
Pause Task	7
EventBits.....	8
Resultat der Zeitmessung	9
Abbildungsverzeichnis.....	10

Aufgabenstellung



Aufgabe:

Wähle einen Algorithmus (Es darf auch ein anderer sein. Es finden sich diverse im Netz.)

Realisiere den Algorithmus in einem auf FreeRTOS basierenden Programm.

Dabei soll folgendes stets gegeben sein:

- Der aktuelle Wert soll stets gezeigt werden. Update alle 500ms
- Der Algorithmus wird mit einem Tastendruck gestartet und mit einem anderen Tastendruck gestoppt.
- Mit einer dritten Taste kann der Algorithmus zurückgesetzt werden.
- Die Kommunikation zwischen den Tasks soll mittels EventBits stattfinden.
 - o EventBit zum Starten des Algorithmus
 - o EventBit zum Stoppen des Algorithmus
 - o EventBit zum Zurücksetzen des Algorithmus
 - o EventBit für den Zustand des Kalkulationstask als Mitteilung für den Anzeige-Task
- Mindestens zwei Tasks müssen existieren.
 - o Interface-Task für Buttonhandling und Display-Beschreiben
 - o Kalkulations-Task für Berechnung von PI
- 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.)

Algorihtmus

Als Algorihtmus zur Berechnung von π (Pi) habe ich die Leibniz Reihe gewählt. Mit der Anwendung dieser Reihe erhält man $\pi/4$. Das Ergebnis mit 4 multipliziert liefert den Wert von π (Pi). Die Reihe wird wie folgt beschrieben: $\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - \dots$. Um diese Reihe theoretisch ewig fortführen zu können, ohne alles eintippen zu müssen, musste die Reihe analysiert und eine Regelmässigkeit gefunden werden.

Nach genauer Analyse kann man erkennen die Zähler immer um 2 erhöhen bzw. jeder 2. Immer um 4 erhöht wird.


$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \dots$$


Abbildung 1

Mit dieser Voraussetzung und mit der Anwendung einer For Schleife war es nun möglich, die Reihe mit wenigen Ausdrücken anzuwenden und nur durch ändern der Variable i, sie beliebig zu erweitern.

$$dmyPi = dmyPi - \frac{1}{3 + 4 \cdot i} + \frac{1}{5 + 4 \cdot i}$$

Abbildung 2

Wie in Abb 2 zu erkennen ist, muss nur der Wiederholungswert von i verändert werden. Für dmyPi wird im Programm eine double Variable mit dem Anfangswert 1 erstellt.

Tasks

Da das Programm verschiedene Schritte durchführen muss wurden mehrere Tasks verwendet.

- Calculations Task
- Buttonhandler Task
- Interface Task
- Count Time Task
- Pause Task

Calculations Taks

```
void PiCalc (void *pvParameters)
{
    for (;;)
    {
        action= xEventGroupWaitBits( EventGroup, StartBit, pdTRUE, pdFALSE, portMAX_DELAY);
        if ((action & StartBit) !=0 )
        {
            dmyPi += 1.0;
            for ( i; i <300000 ; i++)
            {
                dmyPi = dmyPi - 1/(3+i*4) + 1/(5+i*4);
                if (i == 125500)
                {
                    xTimerStop(Timer1,0);
                }
            }
        }
    }
}
```

Abbildung 3

Hier wird die Zahl π (Pi) mittels der in Algorithmus beschriebenen For Schleife berechnet. Beim Erreichen der 5. Nachkommastelle (125500 Iterations) wurde über eine If-Abfrage der Timer, der zuständig ist um die Zeit zu stoppen, angehalten. Der Berechnungsvorgang läuft aber weiter, bis die vom Bediener eingestellten Iterations erreicht wurden (hier 300000).

Buttonhandler Task

```
void vButtonTask(void *pvParameters) {
    initButtons();
    vTaskDelay(3000);
    vDisplayClear();
    for(;;) {
        updateButtons();

        if(getButtonPress(BUTTON1) == SHORT_PRESSED) {
            vDisplayClear();
            vDisplayWriteStringAtPos(0,0, "Start Calculation");
            xEventGroupSetBits(EventGroup, StartBit);
            xTimerStart(Timer1, 0);
            //vTaskResume(PiCalc_task);
        }
        if(getButtonPress(BUTTON2) == SHORT_PRESSED) {
            vDisplayClear();
            vDisplayWriteStringAtPos(0,0, "Pause Calculation");
            xEventGroupSetBits(EventGroup, PauseBit);
            xTimerStop(Timer1, 0);
        }
        if(getButtonPress(BUTTON3) == SHORT_PRESSED) {
            vDisplayClear();
            vDisplayWriteStringAtPos(0,0, "Start Calculation");
            xEventGroupSetBits(EventGroup, WeiterBit);
            xTimerStart(Timer1, 0);
        }
        if(getButtonPress(BUTTON4) == SHORT_PRESSED) {
            vDisplayClear();
            vDisplayWriteStringAtPos(0,0, "Reset");
            xEventGroupSetBits(EventGroup, ResetBit);
            xTimerReset(Timer1, 0);
            xTimerStop(Timer1, 0);
        }
        vTaskDelay((1000/BUTTON_UPDATE_FREQUENCY_HZ)/portTICK_RATE_MS);
    }
}
```

Abbildung 4

Dieser Task verwaltet die implementierten Buttons. Jeder Button setzt ein anderes Eventbit, welches dann je nachdem einen anderen Task aufruft.

Button 1: Start (Setzt das StartBit und startet den Timer)

Button 2: Pause (Setzt das PauseBit und stoppt den Timer)

Button 3: Weiter (Setzt das WeiterBit und startet den Timer ebenfalls)

Button4: Reset (Setzt das ResetBit, stoppt und setzt den Timer zurück)

Alle 4 Button geben jeweils ihre Funktion auf dem Display aus(Start, Pause, Weiter und Reset)

Interface Task

```
void Interface (void *pvParameters)
{
    for (;;)
    {
        dtostrf(dmyPi*4,1,7,Pi_string);
        dtostrf(i,1,0,I_string);
        vDisplayWriteStringAtPos(1,0,"Pi=%s",Pi_string);
        vDisplayWriteStringAtPos(2,0,"Iterations=%s",I_string);
        vDisplayWriteStringAtPos(3,0,"Duration:%d",sec);
        vTaskDelay(500/portTICK_RATE_MS);
    }
}
```

Abbildung 5

Im interface Task wird neben der Ausgabe der gewünschten Parameter noch die double Variable von π (Pi) in einen String umgewandelt. Dies ist nötig, dass die errechnete Zahl auf dem Display ausgegeben werden kann. Der Interface Task wird durch das Delay alle 500ms aufgerufen und aktualisiert.

Count Time Task

```
void Count_Time (void *pvParameters)
{
    for (;;)
    {
        xEventGroupWaitBits( EventGroup, TimeBit, pdTRUE, pdFALSE, portMAX_DELAY);
        count++;
        if (count == 100)
        {
            sec++;
            count = 0;
        }
    }
}
```

Abbildung 6

Im Count Time Task wird mittels eines Timers, der nach Starten alle 10ms auslöst eine Variable (count) immer um 1 inkrementiert. In der If Abfrage wird, wenn count 100 erreicht die Variable sec um 1 inkrementiert (1s ist vergangen). Diese Variable (sec) dient zur Zeiterfassung der Rechnung.

Pause Task

```
void Pause (void *pvparameters)
{
    for (;;)
    {
        stop= xEventGroupWaitBits( EventGroup, PauseBit|WeiterBit|ResetBit, pdTRUE, pdFALSE, portMAX_DELAY);
        if((stop & PauseBit) !=0 )
        {
            vTaskSuspend (PiCalc_task);
        }
        else if ((stop & WeiterBit) !=0 )
        {
            vTaskResume(PiCalc_task);
        }
        else if ((stop & ResetBit) !=0 )
        {
            dmyPi = 0.0;
            i = 0.0;
            sec = 0;
        }
    }
}
```

Abbildung 7

Im Pause Taks befinden sich 3 If Abfragen, die auf die Eventbits von Button 2,3&4 reagieren.

Button 2 Suspended den Calculations Task.

Button 3 Resumend den Task wieder

Button 4 setzt alle Variablen auf 0 zurück

EventBits

```
#define StartBit 1<<0
#define PauseBit 1<<1
#define WeiterBit 1<<2
#define ResetBit 1<<4
#define TimeBit 1<<5
```

Um die einzelnen Tasks kontrolliert in richtiger Reihenfolge aufzurufen, wurden Eventbits genutzt. Diese werden je nach Aktion oder Task gesetzt bzw gelöscht. In den einzelnen Task befinden sich Befehle die eine Abfrage nach bestimmten Bit durchführen. Ist nun eine solche Abfrage erfolgreich d.h das entsprechende Eventbit wurde gesetzt führt der Task seine Befehlskette aus. War die Abfrage nicht erfolgreich wird gewartet und keine Aktion ausgeführt. Folgende 5 Bits wurden genutzt.

- StartBit wird durch drücken des Button 1 gesetzt und im Calculation Task abgefragt. Bei gesetztem Bit startet die Rechnung.
- PauseBit wird durch drücken des Button 2 gesetzt und im Pause Task abgefragt. Bei gesetztem Bit pausiert die Rechnung.
- WeiterBit wird durch drücken des Button 3 gesetzt und im Pause Task abgefragt. Bei gesetztem Bit wird die Rechnung fortgesetzt.
- ResetBit wird durch drücken des Button 4 gesetzt und im Pause Task abgefragt. Bei gesetztem Bit werden alle definierten Werte wie: Pi, i & sec auf 0 zurückgesetzt.
- TimeBit wird von einem Timer alle 10ms gesetzt und im Count Time Task abgefragt

Mit der Abfragefunktion `xEventGroupSetBits` können die einzelnen Bits gesetzt werden und durch `xEventGroupWaitBits` kann die Abfrage realisiert werden, welche nach ausführen des Task mit dem Befehl `pdTrue` an dritter Stelle auch praktischerweise die Bits wieder löscht.

Resultat der Zeitmessung

Meine Resultate um auf fünf Nachkommastellen (3,14159) genau zu rechnen sind:

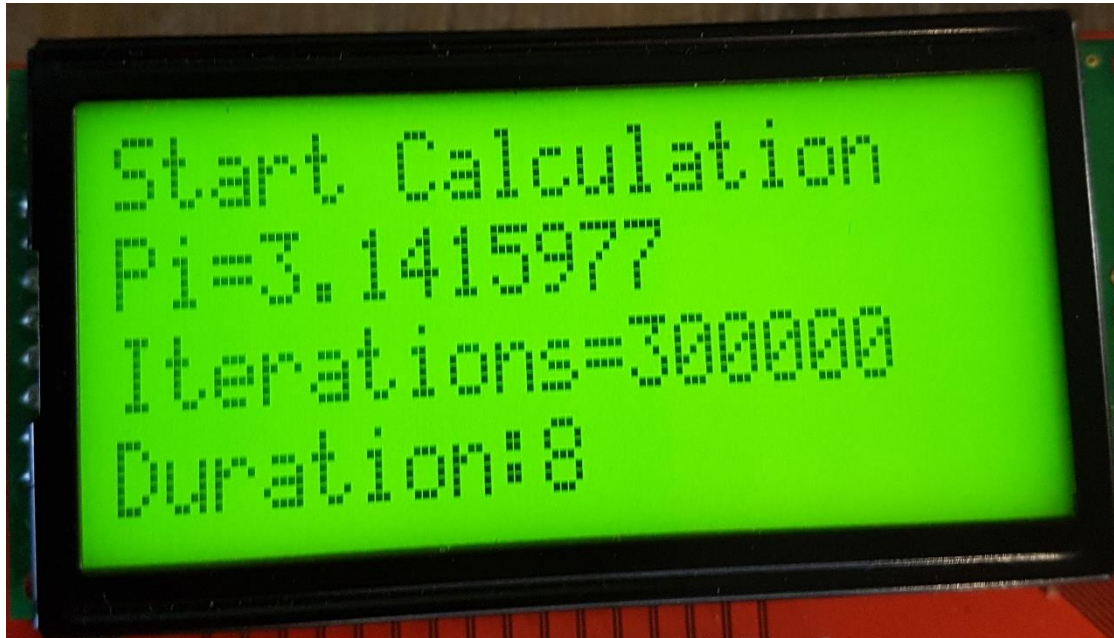


Abbildung 8

Duration: 8 Sekunden

Iterations: 125500 (Die Iterations im oberen Bild zeigen die Gesamtanzahl der Iterations)

Abbildungsverzeichnis

Abbildung 1	3
Abbildung 2	3
Abbildung 3	4
Abbildung 4	5
Abbildung 5	6
Abbildung 6	6
Abbildung 7	7
Abbildung 8	9