JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

CORBA Overview

Systemtechnik Labor
4CHIT 2016/17, GruppeD

Martin Wölfer

Version 0.2
Note:
Betreuer: Th.Micheler
Begonnen am 9. Dezember 2016
Beendet am 27. Dezember 2016

Inhaltsverzeichnis

1 Einführung

1.1 Aufgabenstellung

Nach einem Tasterdruck soll n!(=1*2*3*...*(n-1)*n) für n=0-22 berechnet und mittels printf ausgegeben werden.

Der Taster wird durch einen einfachen Drahtkontakt zwischen Versorgungsspannung und einem entsprechend konfigurierten GPIO-Port realisiert. Der GPIO-Port soll softwaremäßig entprellt werden.

Unter Tasterprellen versteht man folgendes: Ein Taster kann beim Drücken und Loslassen kurze Zeit vibrieren und dabei öfters die Tasterpins verbinden und wieder trennen. Bei einer Interruptsteuerung, welche auf rising und/oder falling edges reagiert, kann dies zu oftmaligen Interrupts und damit zu undefinierten Software-Zuständen führen. Ein Taster ist entprellt, wenn solche Vibrationen keine Auswirkungen auf die Funktionalität des Gesamtsystems haben.

Wir gehen beim softwaremäßigen Entprellen des Tasters folgendermaßen vor:

- Es gibt ein Steuerflag calculateNFac. Dieses wird gesetzt vom Entprellsystem. Es wird vom Rechenalgorithmus (Funktion ´`nFac_0_22`) für n! gleich zu Beginn zurückgesetzt.
- Es gibt ein waitActive Flag und einen waitCount. Das waitActive Flag wird vom Falling Edge Interrupt des Tasters gesetzt. Gleichzeitig wird der waitCount auf 0 gesetzt. Der waitCount wird nun vom Systick-Interrupt hinaufgezählt. Sobald er den Wert 100 (0,1 Sekunden) erreicht hat, wird calculateNFac gesetzt und die Berechnung wird gestartet. Es wird dabei auch der waitCount auf 0 zurückgesetzt und der Timer deaktiviert (waitActive Flag wird zurückgesetzt). Der waitCount wird nun nicht weiter hinaufgezählt.
- Der Rising Edge Interrupt des Tasters deaktiviert einen laufenden waitCount ebenfalls und setzt den waitCounter ebenfalls auf 0 zurück.
- Da man bei einem Interrupt nicht zwischen Rising und Falling Edge unterscheiden kann, muss man den Taster mit 2 Pins verbinden und ein Pin auf falling Edge konfigurierenund das andere Pin auf Rising Edge!
- Die Berechnung von n! selbst sollte nicht im Interrupt, sondern im Hauptprogramm erfolgen. In einer Endlosschleife wird das Steuerflag calculateNFac abgefragt.

Diese Vorgehensweise soll folgendes bewirken: Die n! Berechnung wird erst nach dem Loslassen des Tasters aktiviert, und zwar ca. 0,1 Sekunden nach dem Loslassen. Vibrationen setzen den ensprechenden Wartealgorithmus wieder auf die Ausgangslage (Zähler waitCount = 0) zurück.

Für das Ausgeben der n!-Werte verwenden werden die Debug-Einheiten TPIU und ITM des Mikrocontrollers.

Schreibe wieder ein Protokoll. Erkläre darin den Code und füge auch einen Screenshot der n!-Ausgabe hinzu.

2 Ergebnisse

2.1 printf-Ausgabe

Die `printf-Ausgabe` über das `STMF32DISCOVERY-BOARD` läuft über das Programm `STM32 ST-Link Utility`. In das Programm, welches auf das Board geladen wird, wird lediglich eine Ausgabe mit `printf()` realisiert und es wurde folgender Code hinzugefügt:

```
// 0xE004 2004, Bit 5
SET_BIT(DBGMCU->CR, DBGMCU_CR_TRACE_IOEN);

TPI->CSPSR = 0x1;
TPI->FFCR = 0x102;

ITM->LAR = 0xC5ACCE55;
ITM->TCR = 0x00010005;
ITM->TER = 0x1;
ITM->TPR = 0x1;
```

und

```
int write(int file, char *ptr, int len);
int __io_putchar(int ch) {
    return(ITM_SendChar(ch));
}
```

Das `printf()` ist in einer endlosschleife mit einer Laufvariable welche jede sekunde hochzählt => `HAL_Delay()`.

Wenn man das Programm auf dem Board ausführt, mit dem bereits erwähnten Programm sich auf das Board verbindet und dann den `Serial Wire Viewer` öffnet kann man die ausgeführten `printf()` sehen.

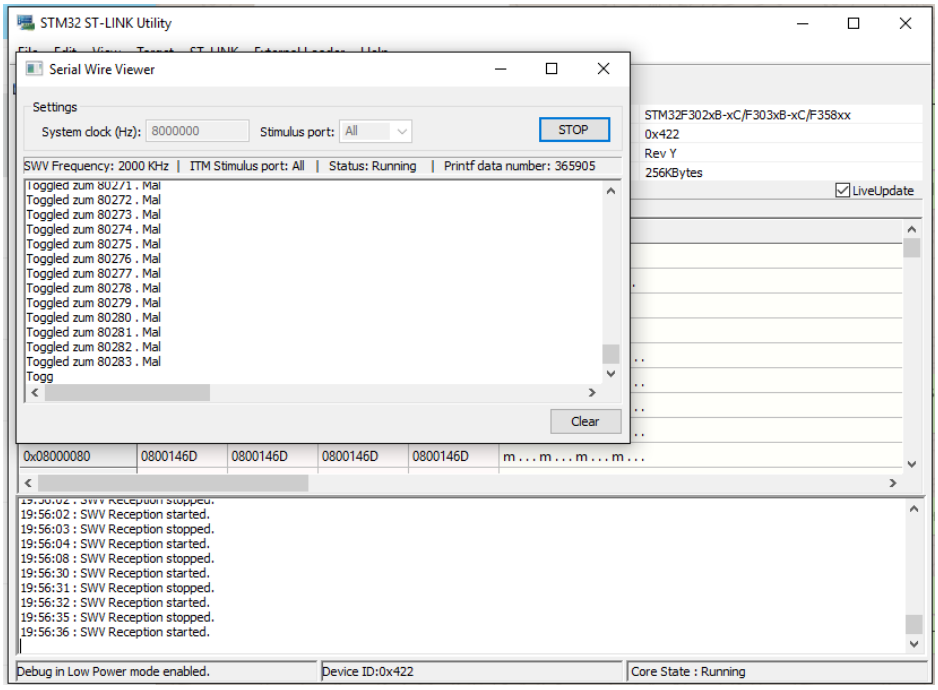

Abbildung 1: Es wird ausgegeben wie oft die Endlosschleife schon durchlaufen wurde

2.2 printf-Ausgabe mit Button

Um umzusetzen dass eine bestimmte Aufgabe ausgeführt wird wenn der User-Button gedrückt wird muss der `EXTI0-Handler` wieder zuerst implementiert werden:

```
void EXTI0_IRQHandler(void){
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
}
```

Nun muss nur mehr die Funktion `void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN)` implementiert werden und der User-Button intialisiert werden:

```
void BUTTON_Init(void){
    //Auf dem Port A (GPIOE) aktivieren
    __HAL_RCC_GPIOA_CLK_ENABLE();

    //Sagen welche Pins angesteurt werden sollen (1,2,3)
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    //Modus auf Push Pull setzen
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    //Pull Modus setzen auf NoPull
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;

    //GPIO Initalisieren mit GPIO (Port a) und der "Konfiguration"
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    HAL_NVIC_SetPriority(EXTI0_IRQn,2,0);
    HAL_NVIC_EnableIRQ(EXTI0_IRQn);
}
```

Die bereits genannte Funktion `HAL_GPIO_EXTI_Callback()` wird jedesmal aufgerufen wenn ein Interrupt durch den User-Button ausgeführt wird, in meiner Funktion steht folgendes `printf()`

```
printf("%d! = %llu \n",n,factorial(n));
```

Die Funktion `factorial()`:

```
unsigned long long factorial(unsigned long long f){
    if ( f == 0 )
        return 1;
    return(f * factorial(f - 1));
}
```

Zu beachten ist dass `unsigned long long` nur 20! noch genau speichern kann, 21! ist schon zu groß um es zu speichern!

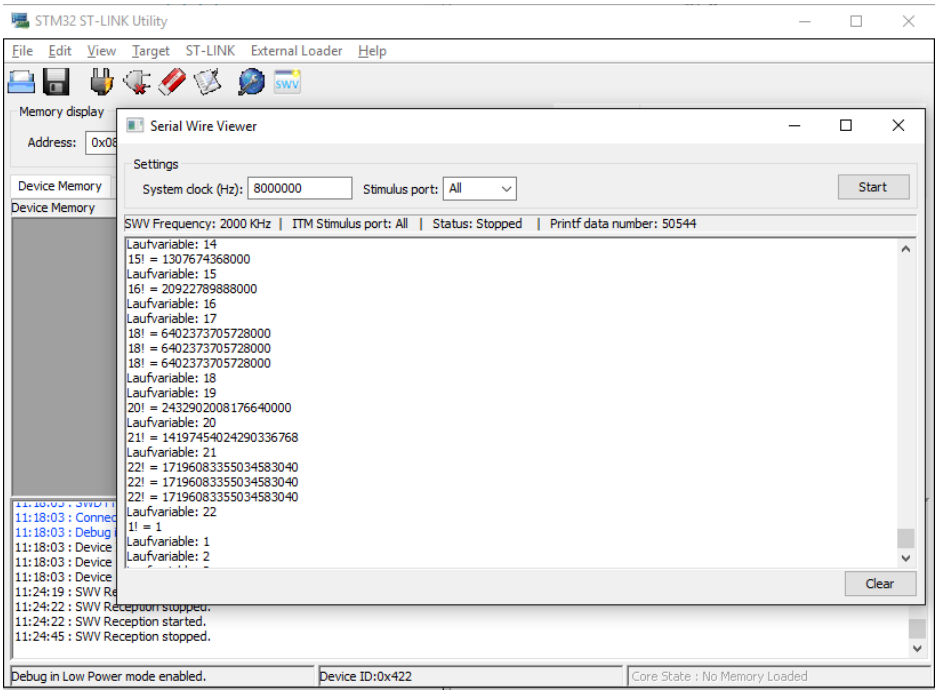

Abbildung 2: Eine Laufvariable läuft von 1-22 und wenn der Button gedrückt wird wird die Fakultät von der Laufvariable ausgegeben

Abbildungsverzeichnis