JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll
GK10.1: Priorität von Interrupts

Systemtechnik Labor
5BHIT 2017/18

Martin Wölfer

Version 0.1
Begonnen am 15.März.2018
Beendet am 22. März 2018

Note:
Betreuer: WEIJ

Inhaltsverzeichnis

1 Aufgabenstellung

Die Aufgabe ist im Detail am Ende des Dokumentes ÏnterruptAdvanced.pdf"beschrieben. In diesem Dokument ist davor auch die technik der Interrupt-Priorisierung einigermaßen detailloiert beschrieben.

Führe die Aufgaben wie im Dokument beschrieben durch und schreibe ein zugehöriges Protokoll. GIb den Sourceode und das Protokoll ab.

1.1 Teil (a): Interrupts für Blinken und Toggeln

Erstelle ein einfaches Programm, wobei zwei LEDs blinken. Verwende hierbei für eine LED die Funktion HAL_Delay im Hauptprogramm. Diese nutzt wiederum implizit den SystickInterrupt. Die zweite LED wird direkt über den Systick-Callback gesteuert. Eine weitere LED wird mit dem Taster getoggelt. Alles zusammen funktioniert einwandfrei. Beide InterruptRoutinen (Systick und EXTI0) werden schnell wieder verlassen, sodass es dabei zu keinen Problemen kommt.

1.2 Teil (b): Blinken innnerhalb einer Interruptroutine

Wenn der Taster des Boards gedrückt wird, so möge eine LED drei Mal blinken. Die Steuerung für das Blinken möge in der Interruptroutine des Tasters selbst erfolgen. Das Blinken soll mit HAL_Delay gesteuert werden. Erst nach dem Blinken soll die InterruptRoutine verlassen werden. Die Priorität und des Taster-Interrupts muss man ordentlich konfigurieren. In HAL_Init wird die Priorität des Systick-Interrupts auf niedrig (15) gesetzt. Man muss die Priorität von Systick auf hoch setzen (z.B.: 0) und die Priorität des Taster - Interrupts auf einen etwas niedrigeren Wert (z.B.: 1). Ändert man die Priorität nicht, so bleibt das Programm ewig in der Interrupt-Routine des Buttons hängen!

1.3 Teil (c): Eigenen Interrupt definieren und via Software auslösen

Trage deine eigene Interrupt-Routine (Name könnte sein <deinName>_IRQHandler) in einer feien Stelle der Interrupttabelle ein. Definiere lokal die zugehörige Interrupt-Nummer (<deinName_IRQn). Die Interrupt-Routine möge einfach nur ein LED toggeln. Aktiviere dein Interrupt in der Interrupttabelle. Löse nun den Interrupt in der Callback-Routine für den Taster aus. Wenn die Priorität des selbstdefinierten Interrupts höher ist als die des Tasters, so wird er sofort ausgeführt. Wenn die Priorität gleich oder niedriger ist, so wird der selbstdefinierte Interrupt erst ausgeführt, nachdem der Taster-Interrupt beendet ist.

1.4 Teil (d): Externen Schalter entprellen

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 entprellen

einen externen Taster folgendermaßen: Wir warten im Interrupt wenige Millisekunden und prüfen dann den Zustand des zugehörigen Pins. Ist der Zustand auf 1, dann haben wir einen rising-edge Interrupt, ist der Zustand auf 0 so handelt es sich um einen falling-edge Interrupt. Wir ignorieren letzteren! Mit einem solchen entprellten externen Schalter toggeln wir eine LED. Wiederum muss die Priorität des Systick-Interrupts höher sein als die Priorität des Interrupts des externen Tasters.

1.5 Teil (e): 2 Schalter mit unterschiedlichen Prioritäten

Realisiere folgendes:

- Ein Blinklicht in der main-Funktion mittels HAL_Delay.
- den Systick-Interrupt mit hoher Priorität
- Bei Betätigen des Taster am Board soll ein zweites LED 5 Mal blinken, realisiert mittels HAL_Delay.
- Einen externen (entprellten) Schalter, welcher ein weiteres LED 5 Mal blinken lässt, wiederum mit HAL_Delay realisiert. Die Priorität ist hier niedriger als beim Taster am Board.

Wenn sich die Preemptive-Priorität sieht man hier folgendes:

- Das Betätigen eines Tasters unterbricht den Blinker der main-Funktion.
- Das Betätigen des Board-Tasters unterbricht das Blinken des externen Schalters, aber nicht umgekehrt. Es wird hier angenommen, dass sich die Preemptive-Priorität der beiden Schalter unterscheidet.

Führe nun dieselbe Aufgabe noch einmal durch. Nun möge die preemptive-Priorität der beiden Schalter gleich sein, sie sollen aber unterschiedliche Sub-Priorität haben. In diesem Fall wird ein laufender Interrupt nicht unterbrochen, der andere Interrupt kommt erst nach Beendigung des ersten Interrupt s an die Reihe. Die unterschiedliche Sub-Priorität kann man folgendermaßen feststellen:

- Betätige einen Schalter, sodass der entsprechende Blinker läuft.
- Betätige nun beide Taster je einmal.
- Nachdem die eine Blinksequenz fertig ist, wird der Interrupt mit der höheren Subpriorität ausgewählt, unabhängig von der Reihenfolge, in welcher die beiden Schalter betätigt wurden.

1.6 Teil (f): Drei Blinklichter mit Systick-Interrupt-Steuerung

Realisiere die obige Aufgabe (drei Blinklichter, ein ewiges und zwei auf Tastendruck) nun noch einmal, dieses Mal jedoch ohne HAL_Delay. Alle Blinker sollen mit dem SystickInterrupt gesteuert werden. In diesem Fall ist die Priorität der Interrupts egal, dabei jedem Interrupt der Mikrocontroller nur ganz kurze Zeit im Interrupt-Modus bleibt. Die mainFunktion enthält am Ende eine leere Endlosschleife, davor müssen die Interrupts und die LEDs konfiguriert werden.

2 Ergebnisse

2.1 Aufgabe a

Der erste Schritt war es in `stm32f3xx_it.c` den `EXTI0_IRQHander` zu implementieren:

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

Weiters wurde eine Funktion erstellt, welche den Port bzw. den Pin für den User-Button initialisiert. Zusätzlich werden für den Interrupt, welcher beim Drücken des Userbuttons ausgelöst wird, auf die Priorität 2 gesetzt:

```
void BUTTON_Init(void){
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    HAL_NVIC_SetPriority(EXTI0_IRQn,2,0);
    HAL_NVIC_EnableIRQ(EXTI0_IRQn);
}
```

Danach wurde die Funktion `HAL_GPIO_EXTI_Callback` implementiert. Dieser Callback wird jedesmal aufgerufen, sobald der Userbutton gedrückt wird:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN){
BSP_LED_Toggle(LED_RED);
}
```

Der nächste Schritt ist es nun 2 LEDs zum Blinken zu bringen. Eine davon soll mit `HAL_Delay` in der main Funktion blinkend gemacht werden, die andere soll auf den `SYSTICK_Callback` reagieren und somit die andere LED zum blinken zu bringen. Der `SYSTICK_Callback` wird automatisch jede Millisekunde aufgerufen, und es ermöglicht es somit Sachen „Parallel“ zur main Methode auszuführen. Daher im `SYSTICK_Callback` nicht gewartet werden kann, muss ein globaler Counter angesetzt werden, welcher bei jedem Aufruf des Callbacks, d.h. jede Sekunde, den Counter hochzählt.

```
void HAL_SYSTICK_Callback(void) {
    counter++;
    // Jede 250ms die LED togglen
    if (counter % 250 == 0) {
        BSP_LED_Toggle(LED_ORANGE);
    }
}
```

In der main Methode wurden zuerst die benötigten Init Methoden ausgeführt, und danach in einer While-True Schleife alle 500ms eine LED getoggled:

```
int main(void) {
    HAL_Init();
    BUTTON_Init();
    BSP_LED_Init(LED_BLUE);
    BSP_LED_Init(LED_RED);
    BSP_LED_Init(LED_ORANGE);

    for(;;){
        HAL_Delay(500);
        BSP_LED_Toggle(LED_BLUE);
    }
}
```

2.2 Aufgabe B

Bei dieser Aufgabe soll beim Betätigen der Taste in eine Interruptroutine gewechselt werden, in welcher eine LED blinkt. Während dieser Routine steht alles still, außer der Systick-Interrupt. Um zu gewährleisten dass dieser auch funktioniert während der Interruptroutine des Buttons, muss diesem eine höhere Priorität als dem EXTI0 Interrupt gegeben werden:

```
int main(void) {
    HAL_Init();
    // Systick interrupt prioritaet hoeher als die vom Button
    HAL_NVIC_SetPriority(SysTick_IRQn,0,0);
    HAL_NVIC_SetPriority(EXTI0_IRQn,1,0);
    HAL_NVIC_EnableIRQ(EXTI0_IRQn);


    BUTTON_Init();
    BSP_LED_Init(LED_BLUE);
    BSP_LED_Init(LED_RED);
    BSP_LED_Init(LED_ORANGE);

    for(;;){
        HAL_Delay(500);
        BSP_LED_Toggle(LED_BLUE);
    }
}
```

Der Callback des Buttons wurde folgendermaßen definiert:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN){
    int i;
    for(i = 0; i < 20; i++){
        BSP_LED_Toggle(LED_RED);
        HAL_Delay(250);
    }
}
```

Falls der Systick Interrupt **NICHT** höhere Priorität als der EXTI0 Interrupt hätte, würde in der Interruptroutine des Buttons festgehangen blieben werden, daher `HAL_Delay` intern Systick abfragt um die gewartete Zeit zu bestimmen.

2.3 Aufgabe C

Bei dieser Aufgabe geht es darum, seinen eigenen Interrupt zu definieren. Der erste Schritt ist es im File `startup/startup_stm32f303xc.s` seinen eigenen Interrupt in der Tabelle zu definieren. Hierbei muss man darauf achten, seinen eigenen Interrupt in einem freien Slot zu definieren. Ich habe meinen Interrupt in der Zeile 232 nach dem `COMP7_IRQHandler` definiert:

```
229     .word   COMP1_2_3_IRQHandler
230     .word   COMP4_5_6_IRQHandler
231     .word   COMP7_IRQHandler
232     .word   Woelfer_IRQHandler
```

Abbildung 1: Woelfer_IRQHandler wurde definiert

Jeder Interrupt hat eine gewisse Nummer, diese sind im File `CMSIS/device/stm32f303xc.h` einzusehen. Relevant hierbei ist die Nummer von jenem Interrupt, nach welchem mein Interrupt definiert wurde, also `COMP7`:

```
155   COMP4_5_6_IRQn              = 65,     /*!< COMP4, COMP5 and COMP6 global Interrupt v
156   COMP7_IRQn                  = 66,     /*!< COMP7 global Interrupt via EXTI Line33
157   USB_HP_IRQn                 = 74,     /*!< USB High Priority global Interrupt
```

Abbildung 2: Interrupt Nummer des COMP7 Interrupts

Nun muss der selbstdefinierte Interrupt diese Nummer plus 1 besitzen:

```
1 #define Woelfer_IRQn (COMP7_IRQn+1)
```

Nun muss der Interrupt mit der beschrieben Nummer initialisiert werden:

```
1 HAL_NVIC_EnableIRQ(Woelfer_IRQn);
```

Der nächste Schritt ist jene Funktion zu implementieren, welche aufgerufen wird sobald der Interrupt gefeuert wird:

```
1 void Woelfer_IRQHandler(void){
      BSP_LED_Toggle(LED_GREEN);
3 }
```

Der letzte Schritt ist den Interrupt auszulösen, dieser soll ausgelöst werden sobald auf den Userbutton gedrückt wird:

```
1 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN){
      BSP_LED_Toggle(LED_RED);
3     HAL_NVIC_SetPendingIRQ(Woelfer_IRQn);
  }
```

Literatur

[1] A.S. Tanenbaum and M. Van Steen. *Verteilte Systeme: Prinzipien und Paradigmen*. Pearson Studium. Addison Wesley Verlag, 2007.

Abbildungsverzeichnis