JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

IndInf03: Ampelsteuerung mit Interrupts

Systemtechnik Labor
5BHITT 2015/16, Gruppe X

Stefan Erceg

Version 1.0

Note:

Betreuer: Prof. Weiser

Begonnen am 2. Oktober 2015

Beendet am 30. Oktober 2015

Inhaltsverzeichnis

1 Einführung

Diese Übung zeigt wie eine Implementierung in C basierend auf Interrupts erfolgen kann.

1.1 Ziele

Das Ziel dieser Übung ist das Wissen über Interrupts in C, welche schon voriges Jahr in Systemtechnik vom Herr Professor Borko beigebracht wurden, zu erweitern. Der Unterschied zu der Übung vom letzten Jahr ist, dass das gesamte Programm nur mit (Timer-)Interrupts gesteuert wird und nicht nur ein bestimmter Teil.

1.2 Voraussetzungen

- STM32F3-Discovery Mikrocontroller
- Grundlagen über Interrupts
- Fertigstellung von IndInf01 (Ampelsteuerung)
- installierte Workbench von STMicroelectronics für STM32-Boards in Eclipse-CDT

1.3 Aufgabenstellung

Implementiere eine Ampel, welche rein mit Interrupts gesteuert wird:

a) Die Ampel möge von rot-orange-grün auf orange-blinken umschalten, wenn der Userbutton gedrückt wird.

b) Sowohl die rot/orange/grün-Phasen als auch die Phasen des Orange-Blinkens sollen mittels Timerinterrupts gesteuert werden.

Das Hauptprogramm besteht dann nur noch aus der Konfiguration des Systems, hernach folgt eine "Idle"-Phase (funktionslose Endlosschleife).

2 Ergebnisse

2.1 Implementierung der verschiedenen Zustände der LEDs

Zum Verwalten der verschiedenen Zustände der LEDs (rot, grün, grün blinkend usw.) habe ich ein c- und ein Header-File erstellt. Dort existieren folgende Funktionen:

- Funktion zum Zurücksetzen aller LEDs

  Mit folgender Anweisung wird eine bestimmte LED ausgeschaltet:

  ```
  BSP_LED_Off(LED_RED);
  ```

- Funktionen zum Einschalten einer bestimmten LED

  Mit folgender Anweisung wird eine bestimmte LED eingeschaltet:

  ```
  BSP_LED_On(LED_RED);
  ```

- Funktionen zum Umschalten einer bestimmten LED

  Mit folgender Anweisung wird eine bestimmte LED umgeschaltet:

  ```
  BSP_LED_Toggle(LED_RED);
  ```

Bei Funktionen, bei denen man eine bestimmte LED auf blinkend setzen muss, habe ich die LED zuerst ein- und danach ausgeschaltet:

```
BSP_LED_On(LED_GREEN_2);
BSP_LED_Off(LED_GREEN_2);
```

2.2 Implementierung der State-Centric State Machine

Da bei dieser Übung verlangt wurde nur eine State Machine umzusetzen, entschied ich mich für die State-Centric State Machine. Bei dieser Art von State Machine wird ein switch-case-Konstrukt aufgebaut, bei der zuerst der Zustand der LED geprüft wird und danach im case-Fall ebenfalls das Event, z.B.:

```
case RED_YELLOW:
     if (t_light->currentEvent == PREPAREFORGOING)
          setRedYellowLEDs();
```

In der Funktion habe ich ebenfalls einen Counter eingebaut, welcher in den einzelnen case-Fällen bis zu einer bestimmten Zahl (z.B. 2000 ms = 2 Sekunden) hochgezählt wird, bis dann zu dem nächstem Zustand gewechselt wird:

```
if (counter >= 2000) {
     counter = 0;
     t_light->currentEvent = PREPAREFORWAITING;
     t_light->currentState = GREEN_BLINK;
}
```

Es existiert ebenfalls ein Header-File, in dem Enums und Structs für die verschiedenen Zustände der LEDs definiert wurden. In diesem Enum werden z.B. die verschiedenen Events beschrieben:

```
typedef enum {
      STOP,              /**< Enum fuer das Event "gelb zu rot" */
      PREPAREFORGOING,   /**< Enum fuer das Event "rot zu rot-gelb" */
      GO,                /**< Enum fuer das Event "rot-gelb zu gruen" */
      PREPAREFORWAITING, /**< Enum fuer das Event "gruen zu gruen-blinkend" */
      CAUTION,           /**< Enum fuer das Event "gruen-blinkend zu gelb" */
      ERR                /**< Enum fuer den Status "Error" falls etwas schief
                              gegangen ist */
} LEDevent;
```

2.3 Verwalten der Interrupts

Die Funktionen, die zum Verwalten der Interrupts zuständig sind, befinden sich bei mir alle im main-File. 3 Funktionen sind hier wesentlich:

- EXTI0_Config

  In dieser Funktion werden alle für die Interrupts relevanten Eigenschaften gesetzt.

  Die GPIOA-Clock wird mit dem Befehl `__GPIOA_CLK_ENABLE();` aktiviert.

  Zum Aktivieren des User-Buttons müssen einige Eigenschaften gesetzt werden, z.B. muss man den Modus auf Rising Edge (wo das Clock Signal von 0 auf 1 geht) mit dem Befehl `GPIO_InitStructure.Mode = GPIO_MODE_IT_RISING;` einstellen.

  Der EXTI0-Interrupt wird mit folgenden Befehlen auf die niedrigste Priorität gesetzt und danach aktiviert:

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

- HAL_GPIO_EXTI_Callback

  Hier wurde definiert, was das Programm machen soll nachdem der User-Button gedrückt wurde. Es wird mit dem bool „night" überprüft, ob der Nachtmodus aktiviert ist. Wenn dies so ist, wird wieder zum Ausgangszustand (rote LED aktiv, Event = STOP) gewechselt, andernfalls wird der Nachtmodus aktiviert.

- HAL_SYSTICK_Callback

  Hier wird lediglich die Funktion, die die State-Centric State Machine umsetzt, aufgerufen.

2.4 Main-Funktion

In der Main-Funktion wird der STM zu Beginn initialisiert:

```
SystemInit();
SystemCoreClockUpdate();
SysTick_Config(SystemCoreClock / 1000);
```

Danach werden die LEDs, die benötigt werden, initialisiert:

```
BSP_LED_Init(LED_RED);
BSP_LED_Init(LED_ORANGE);
BSP_LED_Init(LED_GREEN_2);
```

Die EXTI0_Config-Methode, die wie oben beschrieben alle relevanten Eigenschaften für Interrupts setzt, wird ebenfalls in der Main-Funktion aufgerufen.

Zum Schluss werden noch Default-Werte für die Variablen des Structs, welcher den aktuellen Zustand, das aktuelle Event, den Counter und die Zustände „Unterbrochen" und „Nachtmodus" ablegt, gesetzt:

```
pointerForTrafficLight->currentEvent = STOP;
pointerForTrafficLight->currentState = RED;
pointerForTrafficLight->blink_counter = 0;
pointerForTrafficLight->night = false;
```

3 Zeitaufwand

Leider ging sich diese Übung im Labor-Unterricht, welche von 8 Uhr bis 10:40 h am 2. Oktober 2015 stattfand, nicht einmal ansatzweise aus, da das Schreiben des Protokolls, das Generieren von Doxygen und einige andere Aspekte doch eine gewisse Zeit benötigen. Somit musste ich in meiner Freizeit zusätzliche 2 Stunden und 30 Minuten investieren, um die Aufgabe fertigzustellen.

4 Lessons learned

- Vertiefung der Kenntnisse über Interrupts
- Fähigkeit erweitert, einen bestimmten Code wiederzuverwenden