JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Ampel Intern und Extern

Systemtechnik Labor
4CHIT 2016/17, GruppeD

Martin Wölfer

Version 0.2
Begonnen am 2. Dezember 2016
Beendet am 12. Januar 2017

Note:
Betreuer: WEIJ

Inhaltsverzeichnis

1 Aufgabenstellung

Erstelle 2 Arten von Ampeln:

Variante a: Verwende externe LEDs sowie die Methode `HAL_Delay` um eine gewisse Zeit zu warten.

Variante b: Verwende Interrupts um gewisse Zeiten abzuwarten. Verwende ebenfalls externe LEDs. Zusätzlich verwende den Taster, um zwischen dem Normalbetrieb und dem Orange-Blinken einer Ampel umzuschalten. Der Umschaltvorgang sollte von der Orange in die Orange-Blink Phase bzw. in umgekehrter Richtung in die Rot-Phase erfolgen. Schließlich füge einen zusätzlichen externen Schalter hinzu, mit welchem man ebenfalls umschalten kann.

Dokumentiere dein Vorgehensweise in einem Protokoll und gib dieses zusammen mit dem Code ab.

Gehe Schritt für Schritt vor und dokumentiere diese Schritte:

- Realisiere zuerst ein leuchtendes internes und dann externes LED. - Erzeuge ein blinkendes LED mit `HAL_Delay`. - Erzeuge die Variante (a) der Ampel - Erzeuge ein LED, welches du mit dem Taster umschalten kannst. - Erzeuge ein blinkendes LED mit einem Timer-Interrupt - Erzeuge die Ampel der Variante b - Füge den externen Taster hinzu.

2 Ergebnisse

2.1 Erklärung Interrupts

In dem File `startup_stm32f303xc.c` im Startup-Ordner stehen für die Interrupts verschiedene Funktion zur Verfügung

```
g_pfnVectors:
    .word   _estack
    .word   Reset_Handler
    .word   NMI_Handler
    .word   HardFault_Handler
    .word   MemManage_Handler
    .word   BusFault_Handler
    .word   UsageFault_Handler
    .word   0
    .word   0
    .word   0
    .word   0
    .word   SVC_Handler
    .word   DebugMon_Handler
    .word   0
    .word   PendSV_Handler
    .word   SysTick_Handler
    .word   WWDG_IRQHandler
    .word   PVD_IRQHandler
    .word   TAMP_STAMP_IRQHandler
    .word   RTC_WKUP_IRQHandler
    .word   FLASH_IRQHandler
    .word   RCC_IRQHandler
    .word   EXTI0_IRQHandler
    .word   EXTI1_IRQHandler
    .word   EXTI2_TSC_IRQHandler
    .word   EXTI3_IRQHandler
    .
    .
    .
    ..
    ..
    ....
```


Zum Beispiel wird der `Reset_Handler` bei einem `Reset` aufgerufen oder der `NMI_Handler` beim größten Notfall.

Der `SysTick_Handler` allerdings wird jede Millisekunde aufgerufen!

Im File `stm32f3xx_it.c` sieht man folgende Methode:

```
void SysTick_Handler(void)
{
    HAL_IncTick();
    HAL_SYSTICK_IRQHandler();
#ifdef USE_RTOS_SYSTICK
    osSystickHandler();
#endif
}
```

Die Methode `HAL_IncTick()` sorgt dafür dass auch wirklich jede ms der `SysTick_Handler` aufgerufen wird, bei der Methode `HAL_SYSTICK_IRQHandler()` ist nicht so klar was sie macht.

Wenn man dieser mit `STRG + Doppelklick` folgt wird man folgenden Code erblicken:

```
/**
    * @brief  This function handles SYSTICK interrupt request.
    * @retval None
    */
void HAL_SYSTICK_IRQHandler(void)
{
    HAL_SYSTICK_Callback();
}

/**
    * @brief  SYSTICK callback.
    * @retval None
    */
__weak void HAL_SYSTICK_Callback(void)
{
    /* NOTE : This function Should not be modified, when the callback is needed,
              the HAL_SYSTICK_Callback could be implemented in the user file
    */
}
```

Dies sagt praktisch folgendes aus:

Die Methode`HAL_SYSTICK_Callback()` ist `__weak` definiert, dass heißt sie kann implementiert werden. Zusätzlich wird diese Methode jede ms vom Handler aufgerufen, was heißt dass wir in dieser Methode sehr leicht mit Interrupts arbeiten können!

Um nun tatsächlich mit der Methode `HAL_SYSTICK_Callback()` arbeiten zu können, muss eine globale Variable `int count = 0` definiert werden, welche jede ms inkrementiert wird und mit dieser kann man dann zeitlich überprüfen wann etwas passieren soll.

Ein leichtes Beispiel zum Blinken einer LED:

In `main()` steht lediglich folgender Code:

```
int main(void)
{
    HAL_Init();
    BSP_LED_Init(LED_RED);
}
```

In `HAL_SYSTICK_Callback()` steht nun folgendes:

```
void HAL_SYSTICK_Callback(void){
    count++;
    if(count == 1000){
        count = 0;
        BSP_LED_Toggle(LED_RED);
    }
}
```

Wenn man es ausführt wird man sehen dass die Rote LED am Board jede Sekunde wechselt zwischen leuchten und nicht-leuchten.

Wichtig zu beachten ist, dass man in `main()` mit dieser Methodik keine Endlosschleife benötigt, da diese schon im Handler definiert ist:

```
Default_Handler:
Infinite_Loop:
b    Infinite_Loop
```

2.2 Umsetzung Ampel interne LEDs

Theoretisch kann man diese Aufgabe umsetzen nur durch mit `BSP_LED_...` aber um mich mit den Ports bekannt zu machen habe ich diese Aufgabe gleich mit den Ports umgesetzt.

Auf dem `Discovery-Board` belegen auf Port E die Pins 8-15 die internen LEDs.

Um mit diesen zu Arbeiten müssen diese natürlich zuerst initialisiert werden:

```
void LED_Init(void){
    //Auf dem Port E (GPIOE) aktivieren
    __HAL_RCC_GPIOE_CLK_ENABLE();

    //Sagen welche Pins angesteurt werden sollen (Alle)
    GPIO_InitStruct.Pin = PinsInt;
    //Modus auf Push Pull setzen
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    //Pull Modus setzen auf NoPull
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

    //GPIO Initalisieren mit GPIO (Port E) und der "Konfiguration"
    HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
}
```

Wobei `PinsInt` lediglich definiert wurde als:

```
#define PinsInt GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_13 |
    GPIO_PIN_14 | GPIO_PIN_15
```

und `GPIO_InitStruct` als:

```
GPIO_InitTypeDef GPIO_InitStruct;
```

Nach der Initialisierung kann man nun auf diese Ports zugreifen mit
`HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_9);`

Um die Lesbarkeit des Codes zu verbessern habe ich folgende Methoden geschrieben:

```
void toggleRed(void){
    HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_9 | GPIO_PIN_13);
}

void toggleBlue(void){
HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_8 | GPIO_PIN_12);
}

void toggleOrange(void){
    HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_10 | GPIO_PIN_14);
}

void toggleGreen(void){
    HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11 | GPIO_PIN_15);
}
```

Die Ampel nun umzusetzen war eine reine Logik-Überlegung in der Methode `HAL_SYSTICK_Callback()`:

```
void HAL_SYSTICK_Callback(void){
    count++;
    if(count == 1){
        toggleRed();
    }

    if(count == 3000){
        toggleOrange();
    }

    if(count == 4500){
        toggleOrange();
        toggleRed();
        toggleGreen();
    }

    if(count >= 10000 && count <= 12000){
        if(count%200 == 0){
            toggleGreen();
        }
    }

    if(count == 12000){
        toggleOrange();
    }

    if(count == 14000){
        toggleOrange();
        count = 0;
    }
}
```

2.3 Umsetzung Ampel externe LEDs

Daher ich schon bei der Umsetzung der internen LEDs mit den Ports gearbeitet habe, war es kein Problem externe Spannungen auszugeben.

Für die Farbe Rot habe ich auf dem Port `A` den Pin `1` ausgesucht.

Für die Farbe Gelb habe ich auf dem Port `A` den Pin `2` ausgesucht.

Für die Farbe Grün habe ich auf dem Port `A` den Pin `3` ausgesucht.

Auf dem Steckbrett habe ich nur die bestimmten LEDs mit den jeweiligen Ports verbunden und alle schließlich mit Masse verbunden. Um sicherzugehen wurde zusätzlich jedem LED ein Vorwiderstand vorgeschalten.

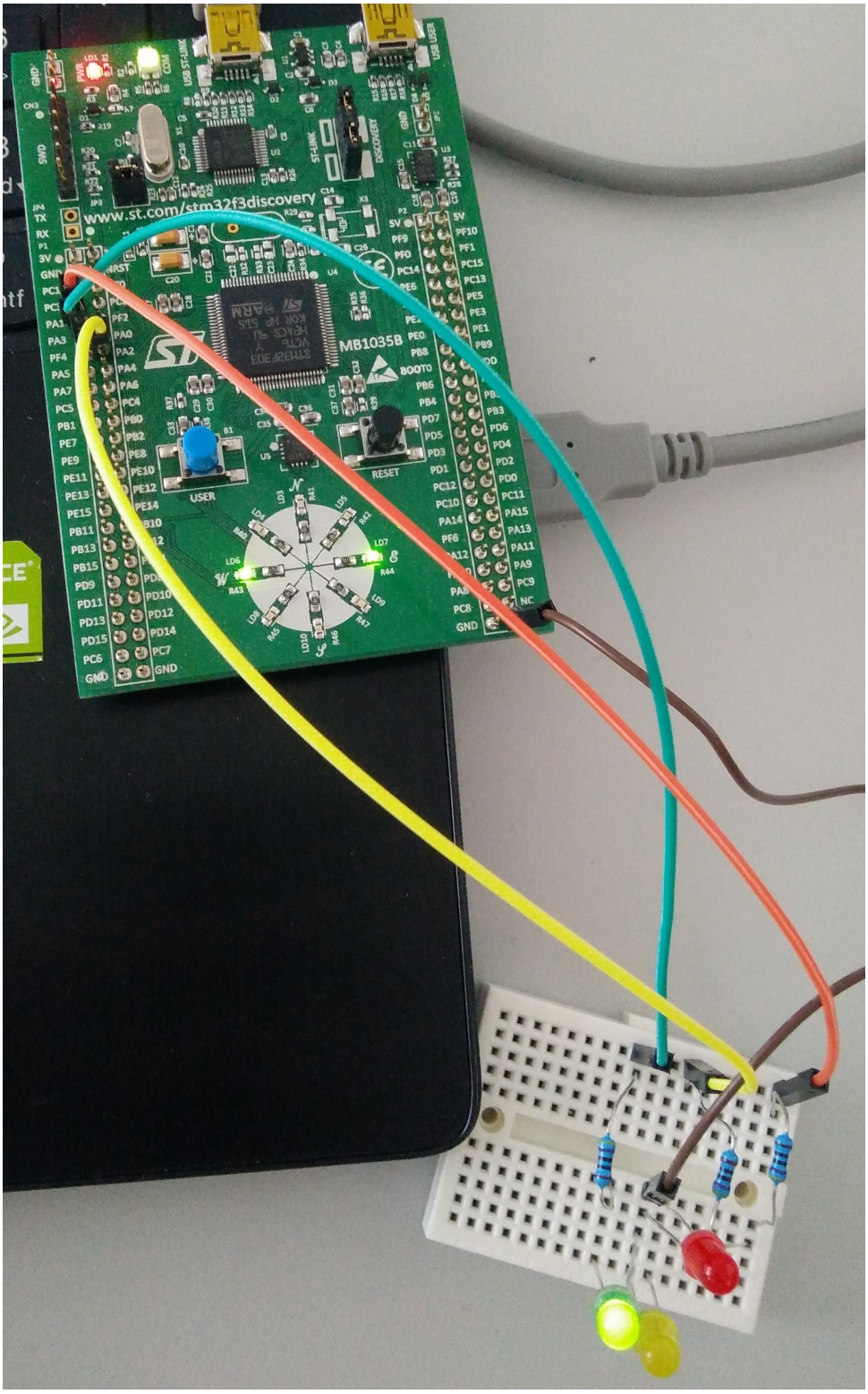

Abbildung 1: LEDs wurden mit Ports und Masse verbunden

Im Code musste zuerst die Funktionalität hinzugefügt werden dass die genannte Pins initialisiert werden:

```
void PIN_Init(void){
    //Auf dem Port A (GPIOA) aktivieren
    __HAL_RCC_GPIOA_CLK_ENABLE();

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

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

Wobei `PinsExt` lediglich definierte wurde als:

```
#define PinsExt GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3
```

Der nächste Schritt war in den bereits genannten Funktionen hinzuzufügen dass diese auch die externen Pins ansteuern:

```
void toggleRed(void){
    HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_9 | GPIO_PIN_13);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_1);
}

void toggleOrange(void){
    HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_10 | GPIO_PIN_14);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_2);
}

void toggleGreen(void){
    HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11 | GPIO_PIN_15);
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_3);
}
```

Am restlichen Code musste sonst nichts verändert werden.

2.4 Umsetzung Ampel mit Button

Der erste Schritt war in dem File `stm32f3xx_it.c` folgenden Code hinzuzufügen:

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

Dadurch wird jedes mal ein Interrupt ausgelöst wenn der User-Button gedrückt wird bzw. die Funktion `void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN)` wird aufgerufen.

Wenn der Button nun gedrückt wird, wird zuerst die `count` Variable zurückgesetzt und alle LEDs werden ausgeschalten mit `GPIO_PIN_RESET`.

```
void turnAllOff(void){
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, PinsInt, GPIO_PIN_RESET);
}
```

Um den Betriebsmodus (Normale Ampel oder blinkendes Orange) gibt es die globale Variable `button`:

```
int button = 0;
```

Diese Variable wird jedes mal beim drücken des Buttons getoggled:

```
if(button == 0){
    button = 1;
} else{
    button = 0;
}
```

Zum Schluss muss nur mehr die Funktion verändert werden die sich um das Verhalten der Ampel kümmert (`HAL_SYSTICK_Callback()`) da je nach Betriebsmodus andere LEDs leuchten sollen:

```
void HAL_SYSTICK_Callback(void){
    count++;
    if(button == 0){
        if(count == 1){
            toggleRed();
        }

        if(count == 3000){
            toggleOrange();
        }

        if(count == 4500){
            toggleOrange();
            toggleRed();
            toggleGreen();
        }

        if(count >= 10000 && count <= 12000){
            if(count%200 == 0){
                toggleGreen();
            }
        }

        if(count == 12000){
            toggleOrange();
        }

        if(count == 14000){
            toggleOrange();
            count = 0;
        }
    }else{
        if(count%1000 == 0){
            toggleOrange();
        }
    }
}
```

Abbildungsverzeichnis