Cours de MICROINFORMATIQUE
Section de Microtechnique
Printemps 2020

Practical Exercises Corrections

Practical Exercise 1: Getting Started on e-puck2 - Corrections	
Title:	Programming simple functions on the e-puck2 miniature mobile robot.
Goal:	Learn how to program and debug the e-puck2 robot and familiarize with the STM32F4 microcontroller family.
Duration:	4 hours
Support:	Files available to download listed and explained in the appendix.
Equipment:	e-puck2 robot, programming environment tools

1 Home task 1: Result of the compilation

- `text` : Size of the code in the Flash memory in bytes.
- `data` : Size of the Non-Zero Initialized global and static data in bytes.
- `bss` : Size of the Zero Initialized and Uninitialized global and static data in bytes.
- `dec` : The sum of `text`, `data` and `bss` in bytes.

To know the total amount of Flash used, we must add `text` and `data` and to know the total amount of RAM used, we must add `data` and `bss`

Thus, this code is using 0.13 % of Flash (1336 bytes / 1 Mbytes) and 0.80 % of RAM (1544 bytes / 192 kbytes)

2 Task 1: Blinking LED7 at 1Hz using a delay function with NOP

Listing 1: main.c

```

#include <stm32f4xx.h>
#include <system_clock_config.h>
#include <gpio.h>
#include <main.h>

// Init function required by __libc_init_array
void _init(void) {}

// Simple delay function
void delay(unsigned int n)
{
    while (n--) {
        __asm__ volatile ("nop");
    }
}

int main(void)
{
    SystemClock_Config();

    // Enable GPIOD peripheral clock
    RCC->AHB1ENR    |= RCC_AHB1ENR_GPIODEN;
```

```

    // LED7 defined in main.h
    gpio_config_output_opendrain(LED7);

    while (1) {
        delay(SystemCoreClock/16);
        gpio_toggle(LED7);
    }
}
```

3 Task 2: Blinking the FRONT_LED at 1 Hz

By using FRONT_LED instead of LED7 with the previous code, FRONT_LED will not blink. GPIOD 14 drives FRONT_LED, but configured in open-drain, it can't push HIGH the gate of the transistor T12. With the external pull-down R_{PD} of 100 [kΩ] on this LED topology, it is necessary to use one of these 2 solutions:

1. Using the internal GPIO pull-up R_{PU} of 40±10 [kΩ]. The HIGH level will defined by the following equation :

$$V_{HIGH} \simeq \frac{V_{DD} \cdot R_{PD}}{R_{PU} + R_{PD}} \tag{1}$$

   Then the real levels we can obtain are :

$$V_{HIGHmin} \simeq \frac{3[V] \cdot 100[k\Omega]}{(50 + 100)[k\Omega]} = 2[V] \tag{2}$$

$$V_{HIGHmax} \simeq \frac{3[V] \cdot 100[k\Omega]}{(30 + 100)[k\Omega]} = 2.3[V] \tag{3}$$

   The Gate Threshold Voltage of IRLML6346TRPBF being $V_{GS(th)max} = 1.1[V]$, it is enough to switch the transistor and turn on the FRONT_LED.

   As measured (Fig. 1) the HIGH level is close to 2.2[V]. The falling time will be measured later, as it depends on the **OSPEEDR** configuration.
   Based on the HIGH level measured and equation (1), R_{PU} can be determined

$$R_{PU} \simeq \frac{V_{DD} \cdot R_{PD}}{V_{HIGH}} - R_{PD} \simeq 36[k\Omega] \tag{4}$$

   The rising time is only function of the RC characteristics on the GPIO and the time measured (Fig. 1) is $t_r \simeq 37.4[\mu s]$. By checking the 10% and 90% levels of 2.2[V] on the oscilloscope, t_r is closer to $35[\mu s]$.

   As defined in `https://en.wikipedia.org/wiki/RC_time_constant` and with R_{FRONT_LED} being the equivalent resistance of the line: $R_{FRONT\_LED} = R_{PU}//R_{PD} \simeq 26[k\Omega]$
   Then the equivalent capacitance of the output can be evaluated by:

$$C_{FRONT\_LED} \simeq \frac{t_r}{V_{HIGH} \cdot R_{FRONT\_LED}} = \frac{35[\mu s]}{2.2 \cdot 26[k\Omega]} \simeq 610[pF] \tag{5}$$

   The FRONT_LED line capacitance can also be **estimated** as the sum of the following capacitances:

   - IRLML6346TRPBF gate capacitance 270[pF] typical
   - μC I/O pin capacitance 5[pF] typical
   - PCB track capacitance estimated at less than 10[pF]

- HZO10 Oscilloscope probe capacitance 15[pF]

It gives about 300[pF]. We can clearly see there is a factor 2 between the result of equation (5) and the estimation above. Here are some reasons that could explain this situation:

- The datasheet of IRLML6346TRPBF gives a typical gate capacitance of 270[pf] for $V_{DS} = 24[V]$. At $V_{DS} = 3.3V$, it is bigger than 300[pF].
- More important is the load effect of the transistor on its gate capacitance. To validate this point, we took a plot of the same signal without the charge on the transistor (LED physically disconnected). Then we measured the new rising time $t_r \simeq 19[\mu s]$ (see fig. 2), which gives us $C_{FRONT\_LED} \simeq \frac{99[\mu s]}{2.2 \cdot 26[k\Omega]} \simeq 330[pF]$.

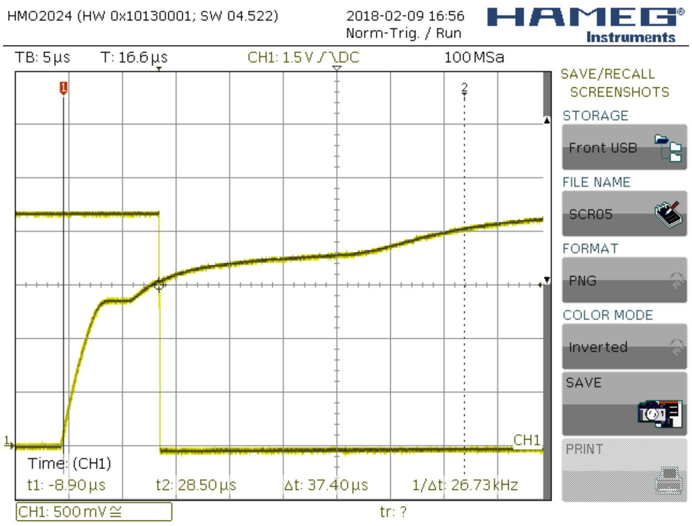

Figure 1: Measure of FRONT_LED test point with Open-Drain and internal Pull-up enabled. Both the rising and the falling edges are visible on this figure.

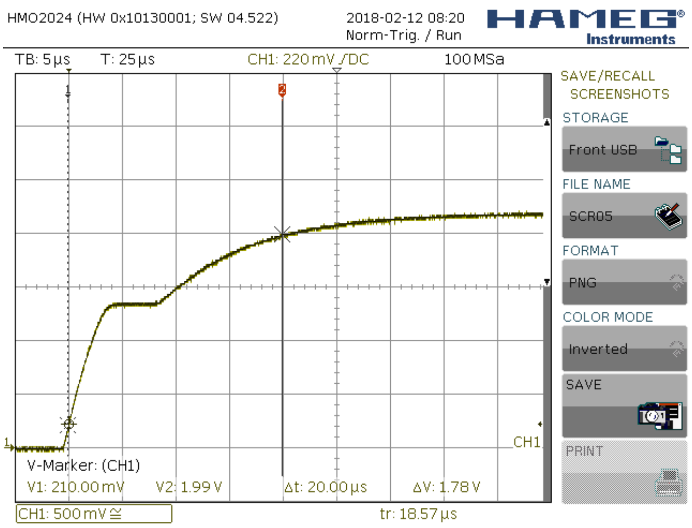

Figure 2: Measure of FRONT_LED test point with Open-Drain and internal Pull-up enabled but without charge on the transistor. Only the rising edge is visible on this figure.

For the code, you can use the same **main.c** than listing 1 and simply replace **LED7** by

LED_FRONT in the lines 26 and 30. Then add the pull-up configuration in the correct function of **gpio.c** in order to enable the pull-up like in line 12 of listing 2.

Listing 2: gpio.c

```
1  ...
2
3  void gpio_config_output_opendrain(GPIO_TypeDef *port, unsigned int pin)
4  {
5      // Output type open-drain : OTy = 1
6      port->OTYPER |= (1 << pin);
7
8      // Output data low : ODRy = 0
9      port->ODR &= ~(1 << pin);
10
11     // Pull-up : PUPDRy = 01
12     port->PUPDR = (port->PUPDR & ~(3 << (pin * 2))) | (1 << (pin * 2));
13
14     // Output speed highest : OSPEEDRy = 11
15     port->OSPEEDR |= (3 << (pin * 2));
16
17     // Output mode : MODERy = 01
18     port->MODER = (port->MODER & ~(3 << (pin * 2))) | (1 << (pin * 2));
19 }
20
21 ...
```

2. Using the push-pull output configuration, such that the HIGH level will be $\sim V_{DD} = 3[V]$. For the code, use the previous code (main.c, main.h, gpio.c and gpio.h) and add a function in gpio library that configures an output in push-pull mode (listings 3 and 4) and use it instead of the open-drain one (listing 5).

Listing 3: gpio.h

```
#ifndef GPIO_H
#define GPIO_H

#include <stm32f407xx.h>

void gpio_config_output_opendrain(GPIO_TypeDef *port, unsigned int pin);
void gpio_config_output_pushpull(GPIO_TypeDef *port, unsigned int pin);
void gpio_set(GPIO_TypeDef *port, unsigned int pin);
void gpio_clear(GPIO_TypeDef *port, unsigned int pin);
void gpio_toggle(GPIO_TypeDef *port, unsigned int pin);

#endif /* GPIO_H */
```

Listing 4: gpio.c

```
...

void gpio_config_output_pushpull(GPIO_TypeDef *port, unsigned int pin)
{
    // Output type pushpull : OTy = 0
    port->OTYPER &= ~(1 << pin);

    // Output data low : ODRy = 0
    port->ODR &= ~(1 << pin);

    // Floating, no pull-up/down : PUPDRy = 00
    port->PUPDR &= ~(3 << (pin * 2));

    // Output speed highest : OSPEEDRy = 11
    port->OSPEEDR |= (3 << (pin * 2));

    // Output mode : MODERy = 01
    port->MODER = (port->MODER & ~(3 << (pin * 2))) | (1 << (pin * 2));
}

...
```

Listing 5: main.c

```
...

    // FRONT_LED defined in main.h
    gpio_config_output_pushpull(FRONT_LED);

    while (1) {
        delay(SystemCoreClock/16);
        gpio_toggle(FRONT_LED);
    }
...
```

By playing with the **OSPEEDR** configuration with the help of `EmbSys Registers` tabular, we can see the influence of the configured speed on the rising and falling edges of the GPIO (Figures 3 and 4).

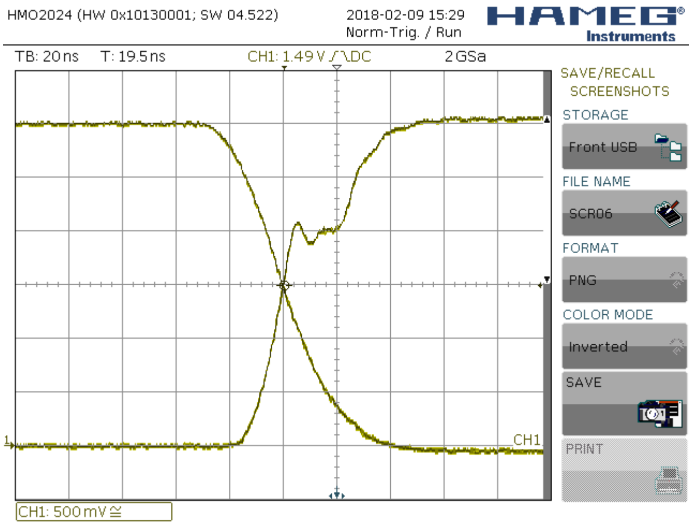

Figure 3: Measure of FRONT_LED test point with Push-pull and Low speed configuration. Both the rising and the falling edges are visible on this figure.

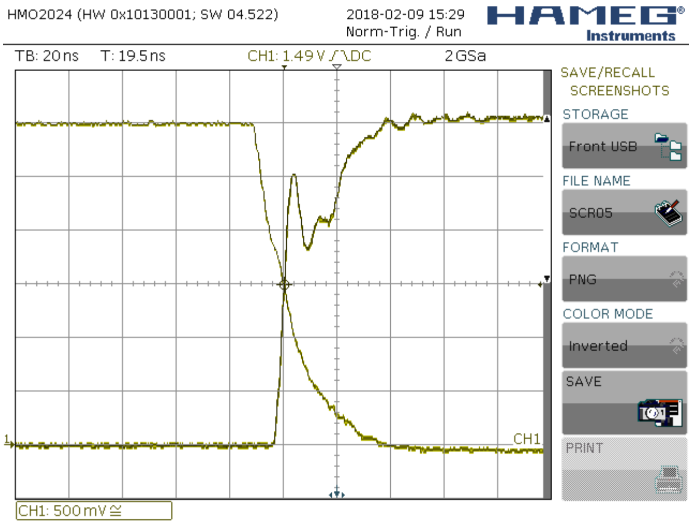

Figure 4: Measure of FRONT_LED test point with Push-pull and High speed configuration. Both the rising and the falling edges are visible on this figure.

4 Task 3: Blinking of the four body LED

BODY_LED is like FRONT_LED and needs a push-pull output topology or at less an open-drain with pull-up.
But as defined in **main.h** this LED is not on the port D but on the port B. So it's necessary to enable the clock for this port too, as done in line 4 of listing 6.
Both LEDs need a transistor because the currents, about 4 x 25[mA] for the body one ($V_F \simeq 2.1[V]$) and 24[mA] for the front one ($V_F \simeq 1.94[V]$) are too important to be driven directly by the μC I/Os (±8 to 20[mA]).

Listing 6: main.c

```
1  ...
2
3      // Enable GPIOB and GPIOD peripheral clock
4      RCC->AHB1ENR    |= RCC_AHB1ENR_GPIOBEN | RCC_AHB1ENR_GPIODEN;
5
6      // LED7 defined in main.h
7      gpio_config_output_pushpull(BODY_LED);
8
9      while (1) {
10         delay(SystemCoreClock/16);
11         gpio_toggle(BODY_LED);
12     }
13 ...
```

5 Task 4: Blinking many LEDs automatically with a circular pattern

Listing 7: main.c

```
#include <stm32f4xx.h>
#include <system_clock_config.h>
#include <gpio.h>
#include <selector.h>
#include <main.h>

// Init function required by __libc_init_array
void _init(void) {}

// Simple delay function
void delay(unsigned int n)
{
    while (n--) {
        __asm__ volatile ("nop");
    }
}

/**************** LED SEQUENCES ****************/

// LEDs sequences with order LED7, LED5, LED3, LED1
static const uint8_t seq1[8][4] = {
    {0, 0, 0, 1},   // ON1
    {0, 0, 0, 0},   // OFF1
    {0, 0, 1, 0},   // ON3
    {0, 0, 0, 0},   // OFF3
    {0, 1, 0, 0},   // ON5
    {0, 0, 0, 0},   // OFF5
    {1, 0, 0, 0},   // ON7
    {0, 0, 0, 0},   // OFF7
};

static const uint8_t seq2[8][4] = {
    {0, 0, 0, 1},   // ON1
    {0, 0, 1, 1},   // ON3
```

```
    {0, 1, 1, 1},   // ON5
    {1, 1, 1, 1},   // ON7
    {1, 1, 1, 0},   // OFF1
    {1, 1, 0, 0},   // OFF3
    {1, 0, 0, 0},   // OFF5
    {0, 0, 0, 0},   // OFF7
};

/**
 * @brief   Updates the LED states
 *
 * @param[in] out       pointer to the table containing the state
 *
 */

static void LEDs_update(const uint8_t *out)
{
    /* LEDs */
    out[3] ? gpio_clear(LED1) : gpio_set(LED1);
    out[2] ? gpio_clear(LED3) : gpio_set(LED3);
    out[1] ? gpio_clear(LED5) : gpio_set(LED5);
    out[0] ? gpio_clear(LED7) : gpio_set(LED7);
}

int main(void)
{
    int selector, old_selector = 0;
    int sequence_pos = 0;
    SystemClock_Config();

    // Enable GPIOB and GPIOD peripheral clock for the LEDs
    RCC->AHB1ENR    |= RCC_AHB1ENR_GPIOBEN | RCC_AHB1ENR_GPIODEN;

    // LEDs defined in main.h
    gpio_config_output_opendrain(LED1);
    gpio_config_output_opendrain(LED3);
    gpio_config_output_opendrain(LED5);
    gpio_config_output_opendrain(LED7);

    init_selector();

    while (1) {
        delay(SystemCoreClock/32);
        selector = get_selector();
        if (selector != old_selector) {
            sequence_pos = 0;
            gpio_set(LED1);
            gpio_set(LED3);
            gpio_set(LED5);
            gpio_set(LED7);
        }
        old_selector = selector;
        switch (selector)
        {
            case 0: // All LEDs off
                gpio_set(LED1);
                gpio_set(LED3);
                gpio_set(LED5);
                gpio_set(LED7);
                break;
            case 1:
                gpio_toggle(LED1);
                break;
            case 2:
                gpio_toggle(LED3);
                break;
            case 4:
                gpio_toggle(LED5);
                break;
            case 8:
                gpio_toggle(LED7);
```

```
                break;
            case 9: // Use Sequence 1
                LEDs_update(seq1[sequence_pos]);
                sequence_pos++;
                sequence_pos %= 8;
                break;
            case 10: // Use Sequence 2
                LEDs_update(seq2[sequence_pos]);
                sequence_pos++;
                sequence_pos %= 8;
                break;
            default:
            break;
        }
    }
}
```

Listing 8: selector.h

```
#ifndef SELECTOR_H
#define SELECTOR_H

void init_selector(void);
int get_selector(void);

#endif /*SELECTOR_H*/
```

Listing 9: selector.c

```
#include <gpio.h>
#include <selector.h>

#define Sel0        GPIOC, 13
#define Sel1        GPIOC, 14
#define Sel2        GPIOC, 15
#define Sel3        GPIOD, 4


void init_selector(void)
{
    // Enable GPIOC and GPIOD peripheral clock
    RCC->AHB1ENR    |= RCC_AHB1ENR_GPIOCEN | RCC_AHB1ENR_GPIODEN;

    gpio_config_input_pd(Sel0);
    gpio_config_input_pd(Sel1);
    gpio_config_input_pd(Sel2);
    gpio_config_input_pd(Sel3);

}

int get_selector()
{
    return gpio_read(Sel0) + 2 * gpio_read(Sel1) + 4 * gpio_read(Sel2) + 8 * gpio_read(Sel3);
}
```

Listing 10: gpio.h

```
#ifndef GPIO_H
#define GPIO_H

#include <stm32f407xx.h>
#include <stdbool.h>

void gpio_config_input_pd(GPIO_TypeDef *port, unsigned int pin);
void gpio_config_output_opendrain(GPIO_TypeDef *port, unsigned int pin);
void gpio_config_output_pushpull(GPIO_TypeDef *port, unsigned int pin);
void gpio_set(GPIO_TypeDef *port, unsigned int pin);
void gpio_clear(GPIO_TypeDef *port, unsigned int pin);
void gpio_toggle(GPIO_TypeDef *port, unsigned int pin);
bool gpio_read(GPIO_TypeDef *port, unsigned int pin);
```

```
#endif /* GPIO_H */
```

Listing 11: gpio.c

```
#include <stm32f407xx.h>
#include <gpio.h>
#include <main.h>

void gpio_config_input_pd(GPIO_TypeDef *port, unsigned int pin)
{

    // Pull-down : PUPDRy = 10
    port->PUPDR = (port->PUPDR & ~(3 << (pin * 2))) | (2 << (pin * 2));

    // Output speed highest : OSPEEDRy = 11
    port->OSPEEDR |= (3 << (pin * 2));

    // Input mode : MODERy = 00
    port->MODER &= ~(3 << (pin * 2));
}


void gpio_config_output_opendrain(GPIO_TypeDef *port, unsigned int pin)
{
    // Output type open-drain : OTy = 1
    port->OTYPER |= (1 << pin);

    // Output data low : ODRy = 0
    port->ODR &= ~(1 << pin);

    // Pull-up : PUPDRy = 01
    port->PUPDR = (port->PUPDR & ~(3 << (pin * 2))) | (1 << (pin * 2));

    // Output speed highest : OSPEEDRy = 11
    port->OSPEEDR |= (3 << (pin * 2));

    // Output mode : MODERy = 01
    port->MODER = (port->MODER & ~(3 << (pin * 2))) | (1 << (pin * 2));
}

void gpio_config_output_pushpull(GPIO_TypeDef *port, unsigned int pin)
{
    // Output type pushpull : OTy = 0
    port->OTYPER &= ~(1 << pin);

    // Output data low : ODRy = 0
    port->ODR &= ~(1 << pin);

    // Floating, no pull-up/down : PUPDRy = 00
    port->PUPDR &= ~(3 << (pin * 2));

    // Output speed highest : OSPEEDRy = 11
    port->OSPEEDR |= (3 << (pin * 2));

    // Output mode : MODERy = 01
    port->MODER = (port->MODER & ~(3 << (pin * 2))) | (1 << (pin * 2));
}

void gpio_set(GPIO_TypeDef *port, unsigned int pin)
{
    port->BSRR = (1 << pin);
}

void gpio_clear(GPIO_TypeDef *port, unsigned int pin)
{
    port->BSRR = (1 << (pin + 16));
}

void gpio_toggle(GPIO_TypeDef *port, unsigned int pin)
{
```

```
    if (port->ODR & (1<<pin)) {
        gpio_clear(port, pin);
    } else {
        gpio_set(port, pin);
    }
}

bool gpio_read(GPIO_TypeDef *port, unsigned int pin)
{
    return (port->IDR & (1<<pin));
}
```

Listing 12: Makefile

```
...

CSRC = main.c gpio.c selector.c

...
```

6 Task 5: Configure the timer TIM7 to have interrupts at 1Hz and blink LED7

Listing 13: main.c

```

#include <stm32f4xx.h>
#include <system_clock_config.h>
#include "gpio.h"
#include "timer.h"

/* init function required by __libc_init_array */
void _init(void) {}

/* simple delay function */
void delay(unsigned int n)
{
    while (n--) {
        __asm__ volatile ("nop");
    }
}

/* LED: GPIOD pin 11 */
#define LED_PIN     11

int main(void)
{
    SystemClock_Config();

    /* LED init */
    /* Enable GPIOD peripheral clock */
    RCC->AHB1ENR    |= RCC_AHB1ENR_GPIODEN;
    gpio_config_output_pushpull(GPIOD, LED_PIN);

    timer7_start();

    while (1) {
        ;
    }

}
```

Listing 14: timer.c

```
#include <stm32f4xx.h>
#include <gpio.h>
```

```c
#include <main.h>

#define TIMER_CLOCK         84000000    // APB1 clock
#define PRESCALER_TIM7      8400        // timer frequency: 10kHz
#define COUNTER_MAX_TIM7    10000       // timer max counter -> 1Hz

void timer7_start(void)
{
    // Enable TIM7 clock
    RCC->APB1ENR |= RCC_APB1ENR_TIM7EN;

    // Enable TIM7 interrupt vector
    NVIC_EnableIRQ(TIM7_IRQn);

    // Configure TIM7
    TIM7->PSC = PRESCALER_TIM7 - 1;      // Note: final timer clock  = timer clock / (prescaler + 1)
    TIM7->ARR = COUNTER_MAX_TIM7 - 1;    // Note: timer reload takes 1 cycle, thus -1
    TIM7->DIER |= TIM_DIER_UIE;          // Enable update interrupt
    TIM7->CR1 |= TIM_CR1_CEN;            // Enable timer
}

// Timer 7 Interrupt Service Routine
void TIM7_IRQHandler(void)
{
    /*
    *
    *   BEWARE !!
    *   Based on STM32F40x and STM32F41x Errata sheet - 2.1.13 Delay after an RCC peripheral clock
     enabling
    *
    *   As there can be a delay between the instruction of clearing of the IF (Interrupt Flag) of
     corresponding register (named here CR) and
    *   the effective peripheral IF clearing bit there is a risk to enter again in the interrupt if
     the clearing is done at the end of ISR.
    *
    *   As tested, only the workaround 3 is working well, then read back of CR must be done before
     leaving the ISR
    *
    */

    /* do something ... */
    gpio_toggle(BODY_LED);

    // Clear interrupt flag
    TIM7->SR &= ~TIM_SR_UIF;
    TIM7->SR;   // Read back in order to ensure the effective IF clearing
}
```

Listing 15: timer.h

```c
#ifndef TIMER_H
#define TIMER_H

void timer7_start(void);

#endif /* TIMER_H */
```