ESS: Exercise Set 3

Software Design

Question 1:

Design an embedded system to control a hot-water tank/boiler. Start from a set of requirements and follow through the Vee diagram, looking at architecture, block-diagram, modules, modelling, V&V and deployment. How would your system be different for a domestic boiler, as compared with an industrial plant?

Question 2:

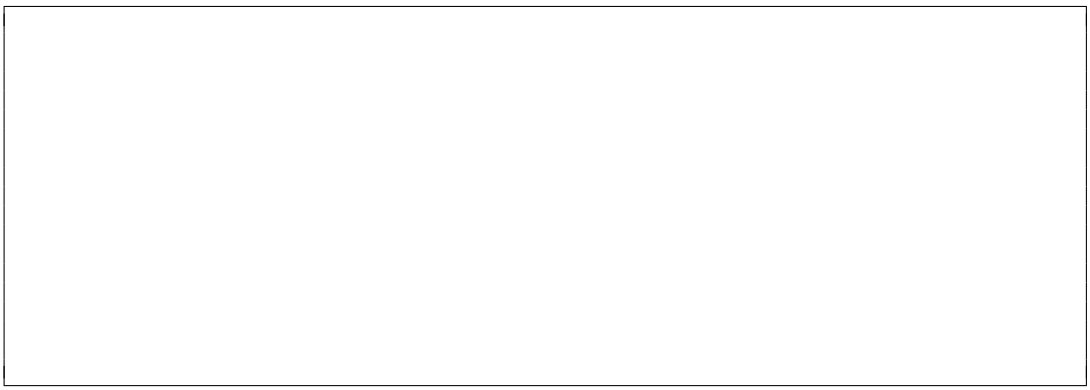
Show how to wrap the led driver from Lab 1 in a harness to do unit testing. The full API is as follows:

```
// initialize led
void led_init(LED_t * led, volatile uint32_t * port, uint32_t pin);
// turn led on
void led_on(LED_t * led);
// turn led off
void led_off(LED_t * led);
// toggle led from off to on and vice-versa
void led_toggle(LED_t * led);
// returns whether led is set (1) or clear (0)
uint8_t led_read(LED_t * led);
```

The harness should not depend on the hardware itself - how could we decouple the logic from the actual hardware (e.g. PORTD?).

(a) Write some tests[1] e.g. `led_init()` should set the bit for the LED to off, but leave the rest of the port register unaltered. Write some sample assertions that you could use to check these.

[1] See `http://throwtheswitch.org/` for a site dedicated to TDD on embedded platforms



(b) To do the testing, we can use `assert()` statements, or if we have support for `printf`)), a very minimal test suite (minunit[3])is defined by the following macro:

```
/* file: minunit.h */
#define mu_assert(message, test) do { if (!(test)) return message; } while (0)
#define mu_run_test(test) do { char *message = test(); tests_run++; \
                               if (message) return message; } while (0)
extern int tests_run;
```

This shows how to run some tests:

```
/* file test_suites.c */

#include <stdio.h>
#include "minunit.h"

int tests_run = 0;

int foo = 7;
int bar = 4;

static char * test_foo() {
    mu_assert("error, foo != 7", foo == 7);
    return 0;
}

static char * test_bar() {
    mu_assert("error, bar != 5", bar == 5);
    return 0;
}

static char * all_tests() {
    mu_run_test(test_foo);
    mu_run_test(test_bar);
    return 0;
}

int main(int argc, char **argv) {
    char *result = all_tests();
    if (result != 0) {
        printf("%s\n", result);
    }
```

[3] http://www.jera.com/techinfo/jtns/jtn002.html

```
    else {
        printf("ALL TESTS PASSED\n");
    }
    printf("Tests run: %d\n", tests_run);

    return result != 0;
}
```

Using the minunit framework, show how to write a few of the tests you specified above.

Question 3:

Write a test strategy for an alarm clock module which triggers an alarm when the time exceeds the preset alarm time. The API for the alarm clock is as follows:

```
#include "real_time.h"
// initialize alarm module - initially no alarm is set
void alarm_init(void);
// set the alarm
void set_alarm(clock_time_t alarm_time);
// clear the alarm.
void clear_alarm(void);
```

The relevant API for the dependency file real_time.h is as follows:

```
struct clock_time
{
  uint8_t year;
  uint8_t month;
  uint8_t day;
  uint8_t hour;
  uint8_t minute;
  uint8_t second;
};
typedef struct clock_time clock_time_t;

// provide a pointer to a clock_time_t struct, this function will update it with the current tim
void get_current_time(clock_time_t * my_time);
```