Task Based Training - 2: Experiment #1

e-Yantra Summer Internship 2017

Vinay Manjunath

Contents

Part 1

```
#include "../../../../Unity-master/src/unity.h"
#include "../../../../CMock/lib/mocks/Mock_ms_delay.h"
#include <stdio.h>

void main_test(void);
void port_init(void);
void led_on_topfour(void);
void led_off_topfour(void);
void led_on_bottomfour(void);
void led_off_bottomfour(void);
```

To begin writing any Unit Test, the proper Header files must be included. The essentials Unit Testing Embedded C programs are:

- Unity header file
- Mocked Function header file
- IO Header file.
- Forward declarations of all functions

Depending on the program to be tested, certain variables from the Main program must be accessed, and in order to do so they must be declared in the Unit Test with the storage class "extern". In a Unit Test where the Main program is IO based, the functions to be tested execute without interruption and hence a variable that is to be observed is over written multiple times. It is important for us to know how many times the variables are over written and what value are written to them. Lets look at how to do this.

- An "extern" array is generally initialized in the Unit Test to allow us to observe the values being written to a port.
- The array is initialized in the Main program and is used to capture the values being written to the port under watch.
- The length of the array depends on how many values the user wishes to record and the nature of the function under testing.

The array "value [5]" is an example of such an array.

Part 2

```
/*
* Function Name: clear_ports
* Input: none
* Output: none
* Logic: Code to clear the values that have been written to PORTJ and DDRJ
Example Call: clear_ports.
*/

void clear_ports()
{
DDRJ = 0;
PORTJ = 0;
}

/*
* Function Name: test_port_init
* Input: none
* Output: none
* Logic: Code to test the initialization of desired I/O port using IO port registers
            viz. DDRJ and PORTJ
*/

void test_port_init()
{
port_init();
TEST_ASSERT_EQUAL_INT(0xFF, DDRJ);
TEST_ASSERT_EQUAL_INT(0x00, PORTJ);
clear_ports();
}
```

Before we being Unit Testing it is important to have a function to clear the ports that are commonly written to. If the ports are not cleared between tests then assignment of a port in one test could lead to a failure in a test, even though it should have passed.

In order to test a function using the Unity framework, a test function has to be created as shown above. Let us look at the flow of a unit test:

- The function that is to be tested has to be called.
- The user can test various variables that may have been altered during execution
- The user can "expect" certain functions to be called by the function under test.
- Variables values are tested using assert statements.

In the Port Init test function, DDRJ and PORTJ are compared with their expected values using an assert statement to verify that the port initialization function in the main program initializes them properly.

Part 3

```
/*
* Function Name: test_led_on_topfour
* Input: none
* Output: none
* Logic: Code to Test wether the top four BAR LEDs are turned on
Example Call: RUN_TEST(test_led_on_topfour)
*/

void test_led_on_topfour()
{
led_on_topfour();
TEST_ASSERT_EQUAL_INT(0xF0, PORTJ);
clear_ports();
}

/*
* Function Name: test_led_on_bottomfour
* Input: none
* Output: none
* Logic: Code to Test wether the bottom four BAR LEDs are turned on
Example Call: RUN_TEST(test_led_on_bottomfour)
*/

void test_led_on_bottomfour()
{
led_on_bottomfour();
TEST_ASSERT_EQUAL_INT(0x0F, PORTJ);
clear_ports();
}
```

The Experiment involves turning on and off the Bar LEDs on the FireBird V. The Bar LEDs are connected to PORTJ. The following steps are used to determine if the correct number of LEDs have been turned on or off:

- Call the program under test.
- Verify the values being written to PORTJ.
- If all values are as expected the test passes.

As seen above, the functions testing whether the top/bottom four Bar LEDs have been turned on first. Verification is achieved by asserting the expected value of PORTJ after the respective LED on/off functions have been called. The led off functions can be verified in a similar manner.

Part 3 continued on next page...

Part 4

```
void test_main_1()
{
  expect_delay(1000);
  expect_delay(1000);
  main_test();
  if (value[1]== 0x00)
     {
          if(value[2]== 0xF0)
               {
                    if (value[3]== 0x00)
                     {
                        TEST_ASSERT_EQUAL_INT(0x0F,value[4]);
                     }
                    else
                     {
                         TEST_FAIL();
                     }
               }

          else if (value[2]== 0x0F)
               {
                    if (value[3]== 0x00)
                     {
                        TEST_ASSERT_EQUAL_INT(0xF0,value[4]);
                     }
                    else
                     {
                         TEST_FAIL();
                     }
               }

          else
              {
                    TEST_FAIL();
               }
     }
 else
     {
          TEST_FAIL();
     }
```

Experiment 1 has the following problem statement: "Your task is to turn on and turn off a set of 4 LEDs connected to upper nibble of Port J and then turn on and turn off another set of 4 LEDs connected to lower nibble of Port J, alternately with an ON time of 1 second."

The above statement can be interpreted in various ways by a given user. This leads to multiple solutions to the same problem. In order to account for the various solutions that might arise, the Unit test for the function that satisfies the problem statement may become complex. The multiple solutions that arise must be individually accounted for and hence there arises multiple tests for the same function. In the case of this experiment it is the main function. There are three ways that the main function of this

Part 4 continued on next page...

experiment is tested, and they have been separated into three separate test functions. If any of the three functions that test the main function pass, then the main function is said to pass the Unit Test.

The test for the main function consists of three parts:

- Expectation of the required delay
- Running the Main function
- Verifying the Main function

The logic of the Unit test is as follows:

- It is expected that the delay function is called twice in the main function with the argument of 1000.
- The extern array value was updated each time the LEDs were turned on or off. checking this array we can see if the LEDs were turned on or off.
- Assumption: 1. LEDs are first turned off.
  2. Top/Bottom four LEDs turned on.
  3. LEDs turned off.
  4. Other four LEDs turned on.
- The Unit Test fails if the Assumption is false.

Part 5

```
void test_main_2()
{
  expect_delay(1000);
  expect_delay(1000);
  if (value[1]== 0xF0)
     {
          if(value[2]== 0x00)
               {
                    if (value[3]== 0x0F)
                     {
                        TEST_ASSERT_EQUAL_INT(0x00,value[4]);
                     }
                    else
                     {
                         TEST_FAIL();
                     }
               }
          else if (value[2]== 0xF0)
               {

                    if (value[3]== 0x0F)
                     {
                        TEST_ASSERT_EQUAL_INT(0x0F,value[4]);
```

Part 5 continued on next page...

```
                    }
                else
                    {
                        TEST_FAIL();
                    }
            }
        else
        {
            TEST_FAIL();
        }
    }
 else
    {
        TEST_FAIL();
    }
}
```

The second Main function Test is similar to the one explained above, except the order in which the LEDs are turned on and off is different.

- It is expected that the delay function is called twice in the main function with the argument of 1000.
- The extern array value was updated each time the LEDs were turned on or off.
- Assumption A : 1. Top Four LEDs are first turned off.
  2. LEDs turned off Bottom four turned on.
  3. Bottom four turned on.
  4. LEDs turned off.
- Assumption B : 1. Top Four LEDs are first turned off.
  2. Bottom four LEDS turned off
  3. Bottom four turned on.
  4. Top four LEDS turned off
- The Unit Test fails if all the Assumptions are false.

Part 6

```
void test_main_3()
{
  expect_delay(1000);
  expect_delay(1000);
  if (value[1]== 0x0F)
    {
        if(value[2]== 0x00)
            {
                if (value[3]== 0xF0)
                {
```

```
                    TEST_ASSERT_EQUAL_INT(0x00,value[4]);
                }
                else
                {
                    TEST_FAIL();
                }
            }
        else if (value[2]== 0x0F)
            {

                if (value[3]== 0xF0)
                {
                   TEST_ASSERT_EQUAL_INT(0xF0,value[4]);
                }
                else
                {
                    TEST_FAIL();
                }
            }
        else
        {
            TEST_FAIL();
        }
    }
  else
    {
     TEST_FAIL();
    }
}
```

The third Main function Test is similar to the previous two, except the order in which the LEDs are turned on and off is different.

- It is expected that the delay function is called twice in the main function with the argument of 1000.
- The extern array value was updated each time the LEDs were turned on or off and hence by
- Assumption A : 1. Bottom Four LEDs are first turned off.
  2. LEDs turned off Bottom four turned on.
  3. Top four turned on.
  4. LEDs turned off.
- Assumption B : 1. Bottom Four LEDs are first turned off.
  2. Top four LEDS turned off
  3. Top four turned on.
  4. Bottom four LEDS turned off.
- The Unit Test fails if all the Assumptions are false.

Part 7

```
int main(void)
{
UNITY_BEGIN();
RUN_TEST(test_port_init);
clear_ports();
RUN_TEST(test_main_1);
RUN_TEST(test_main_2);
RUN_TEST(test_main_3);
clear_ports();
RUN_TEST(test_led_on_topfour);
RUN_TEST(test_led_on_bottomfour);
}
```

The Main function of the Unit Test is where the different tests that were developed are called and run. The Main function should always begin with the Unity Begin function, following which the individual tests to be run are called. Any function developed in this file an be called and run in the Main function as well.