ENCM 511 - Lab 0

Kyle Derby MacInnis
Maheen Hafeez

October 9, 2014

Abstract

The purpose of this lab is to examine the processes and methods involved in developing CCES projects for use with the Blackfin BF533 DSP. The projects include developing a static library, a 'customer' project, and a test suite.

1 Flash_LED Customer Project

1.1 Compiling Initial Customer Code

The code provided for the project was compiled, but errors were present. The errors stemmed from a LINKER error, and this was understood because in the console, the linker was complaining it couldn't reference symbols found within the file. In order to fix this problem, we included BF533 ISR(pre-emptive) routine in the libraries. This removed all errors relating to BlackfinBF533_AudioPremptive_Task.doj. Please see the figures below for screencaptures.

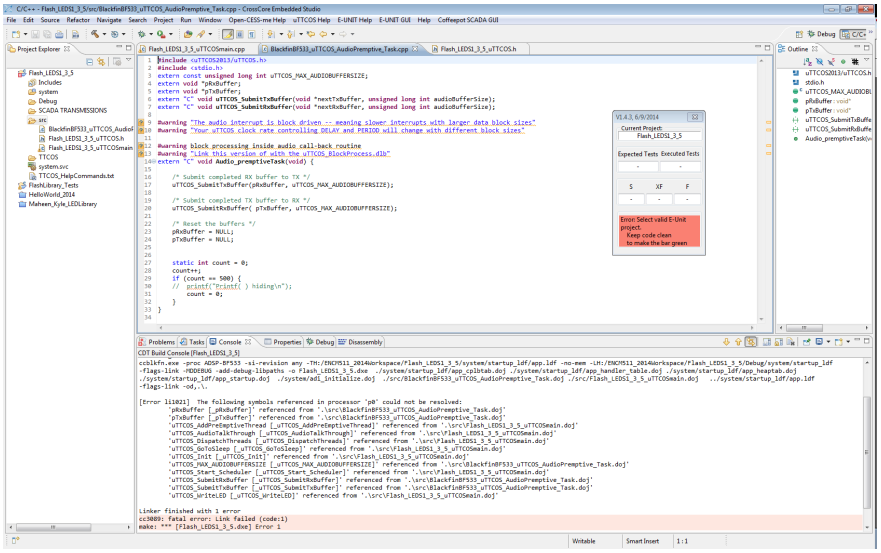

Figure 1: This screencapture was prior to including the Premptive Routine.

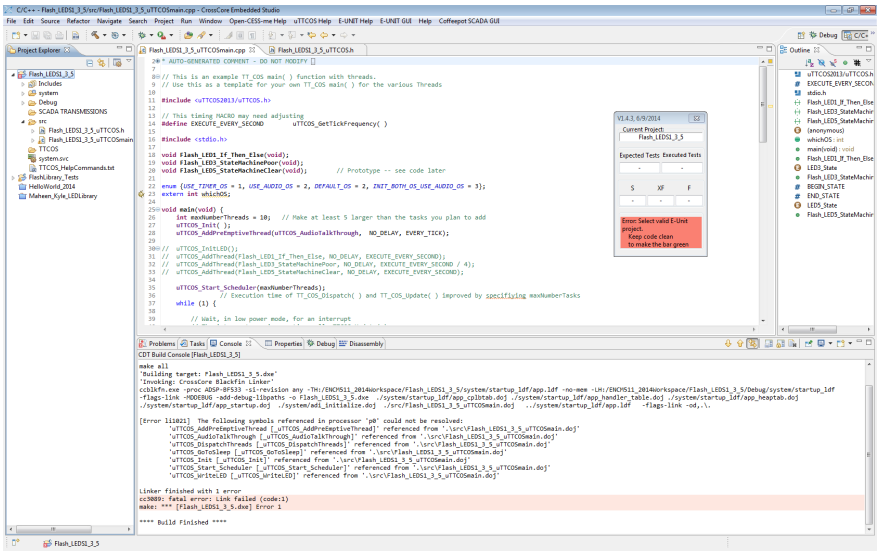

Figure 2: This screencapture was after including the Premptive Routine.

1.2 Testing Initial Customer Code

We compiled and built the initial form of the 'customer' project. The purpose of this project is to flash 3 LEDs at varying intervals. Using the customer provided Flash LED code the following was observed:

- Flash LED1 code worked as expected, and the LED flashed once every second.
- Flash LED3 code worked as expected, and the LED flashed 4 times every second.
- Flash LED5 code worked as expected, and the LED flashed on for 2 seconds, and off for one second.
- Flashing all three LEDs caused some issues however, as the flashing was intermittent and they did not synchronize properly. LED1 flashed fine, LED3 didn't flash four times every second but appeared to be restarting its loops, and LED5 was flickering on and off randomly.

There appears to be bugs in the code, and the details will be covered later in this report. Please see sign off sheet for confirmation of this task.

1.3 Fixed Customer Code

After running our tests, we realized that the provided code had defects present within it. The main problem lay with the part of the code which handled the LED State Machine (for LED3 and LED5). The state machine was being updated separately from the hardware itself, and under certain circumstances, the State Machine could become outdated and hold inaccurate states. This was compounded by the fact that the code for the LEDs also was resetting not just the LED it was supposed to control, but was in fact overwriting the entire LED control register.

The solution to fixing this defect was quite simple, and involved changing the method by which the control register was updated and how the state machine was maintained in memory.

- The code was modified to include a BITMASK which was then ANDed or ORed with the Control Register to ensure only the correct bits were being changed.
- The code was modified to change the State Machine Variables from Static variables within the individual LED functions, to globabl variables within the main project. This allowed for the variables to be updated, by their respective functions, but stopped them from accidently falling out of sync.
- The code was modified to read the Current state of the LEDs each time a function was called, and this allowed for accurate bitmasking.

Please see Appendix A for source code.

2 EUNIT Test Project

2.1 Building the EUNIT Test Project

The next step in the lab was to build the basic EUNIT Test framework, and this code was initially designed to fail, but we modified the code to pass. Please see the figures below for screencaptures of the tests.

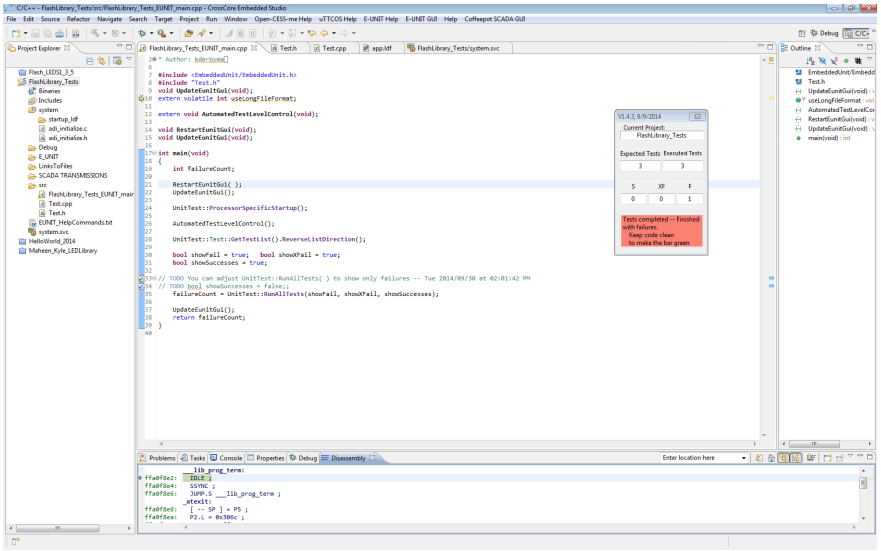

Figure 3: This screencapture shows the initial test framework failing.

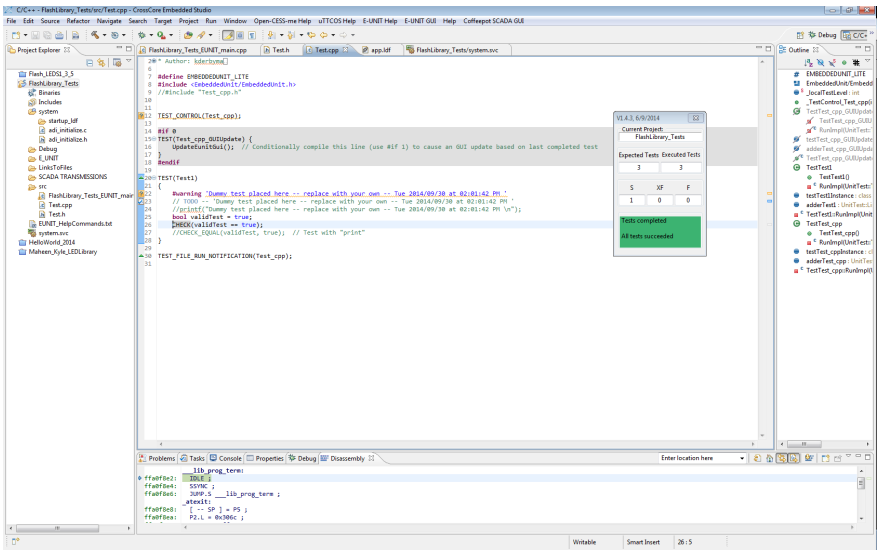

Figure 4: This screencapture shows the test passing, after we modified the test framework. This uses the CHECK macro for comparison.

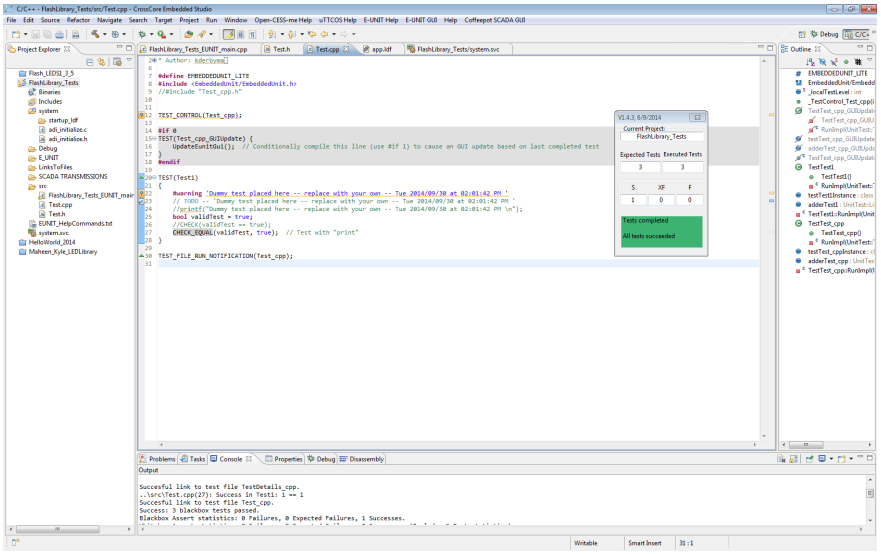

Figure 5: This screencapture shows the test passing, after we modified the test framework. This uses the CHECK_EQUAL macro for comparison.

Please see the sign off sheet for confirmation of our completion of the task.

2.2 Excluding Test Files

After adjusting the basic test framework to pass, we excluded the test files in order to add our own customized tests. We excluded the files from the project so that they would not be compiled in future, and then we did a BLLR and verified that the test files were not being compiled. See the figure below for screen capture.

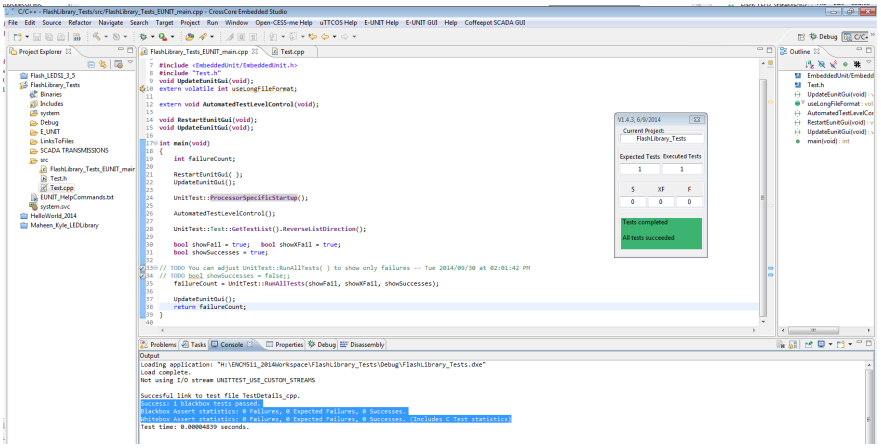

Figure 6: This screencapture shows the test passing, after we excluded the initial test framework.

2.3 Adding Flash_LED Test Code

Following the outline of the lab, the next step was to add in code that enabled us to test whether or not, the customer code we had been given, was functioning correctly.

2.3.1 Flash_LED1_Investigation Test

The first test was to check if LED1 was flashing correctly. Code was provided for this test, and a code defect was present. The defect involved a UNSIGNED CHAR in place of an UNSIGNED INT. The function UTTCOS_READLED() returns a UNSIGNED INT which is 16 bits long, whilst an UNSIGNED CHAR is 8 bits long. Therefore the code would behave oddly and may or may not provide problems in the future if left unchecked.

After correcting the defects and including our own tests, we did a BLLR to see if our test was functioning. The code worked, and our test returned true. Please see the figure below for screen capture.

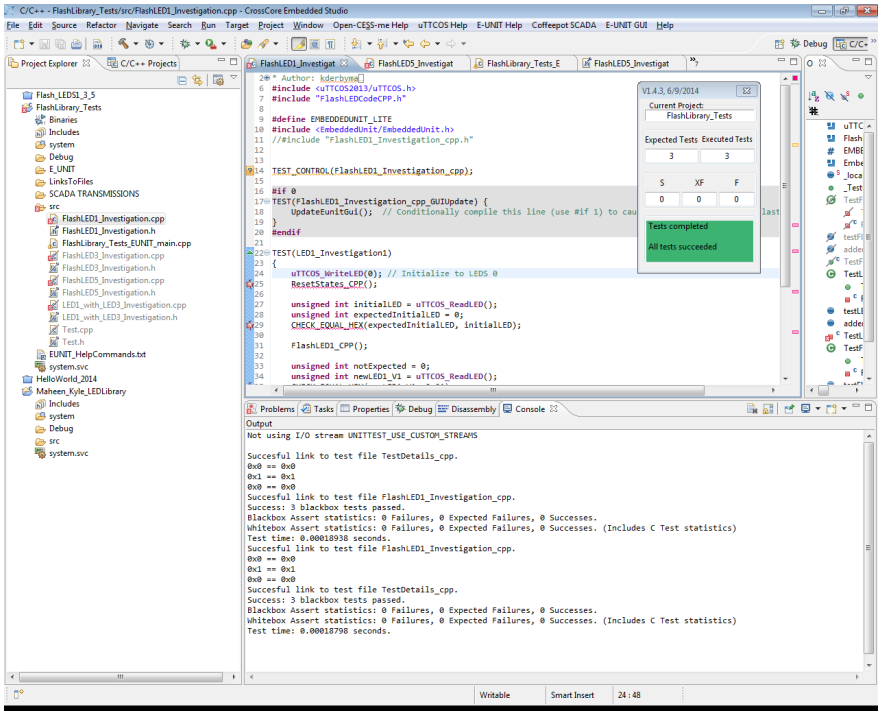

Figure 7: This screencapture shows the test results for LED1 flashing function

2.3.2 LED3 and LED5 Test Investigations

After checking and verifying the test for LED1, we did likewise for LED3 and LED5. Please see figures below for screen captures.

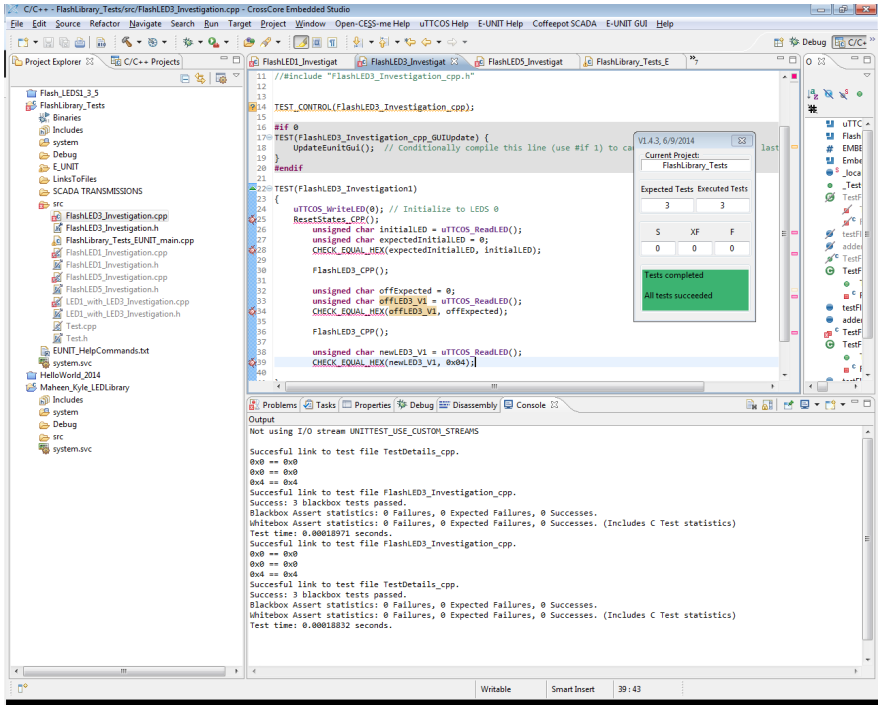

Figure 8: This screencapture shows the test results for LED3 flashing function

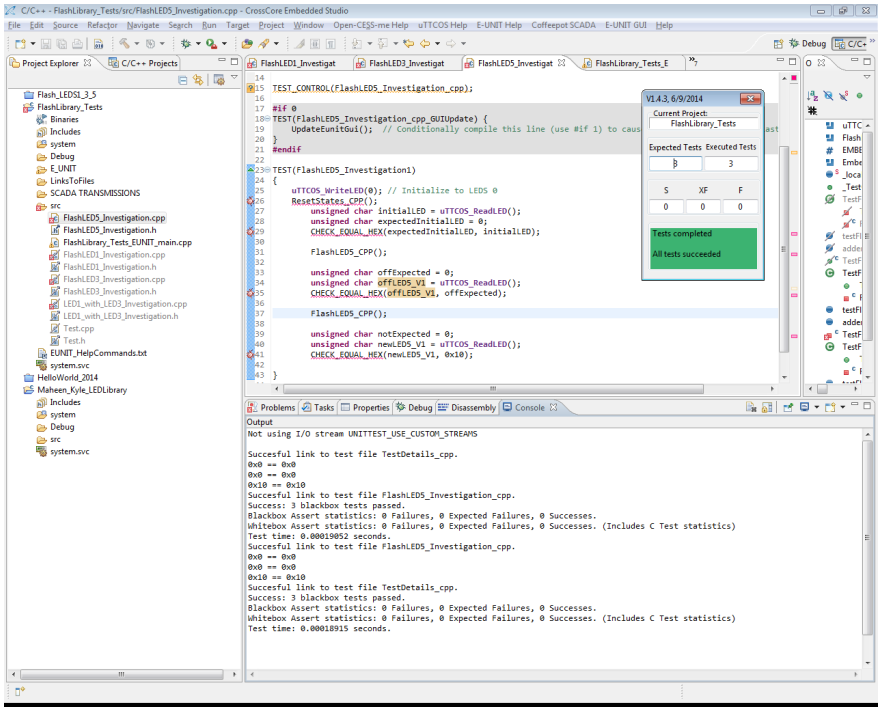

Figure 9: This screencapture shows the test results for LED5 flashing function

2.3.3 Multiple LEDs at Once Test Investigations

After checking and verifying the test for all the LEDs separately, we designed a new set of tests to examine how they worked alongside other LEDS. These tests looked at LED1 with LED3, LED1 with LED5, and LED1 with LED3 and LED5.

Initially, these new tests failed, and this was due to the errors present in the code which had been provided for flashing the LEDs. The main reason was that they were resetting all the LEDs at once, rather than resetting individual LEDS. After the corrections were made, the code test passed. Please see the figure below for confirmation.

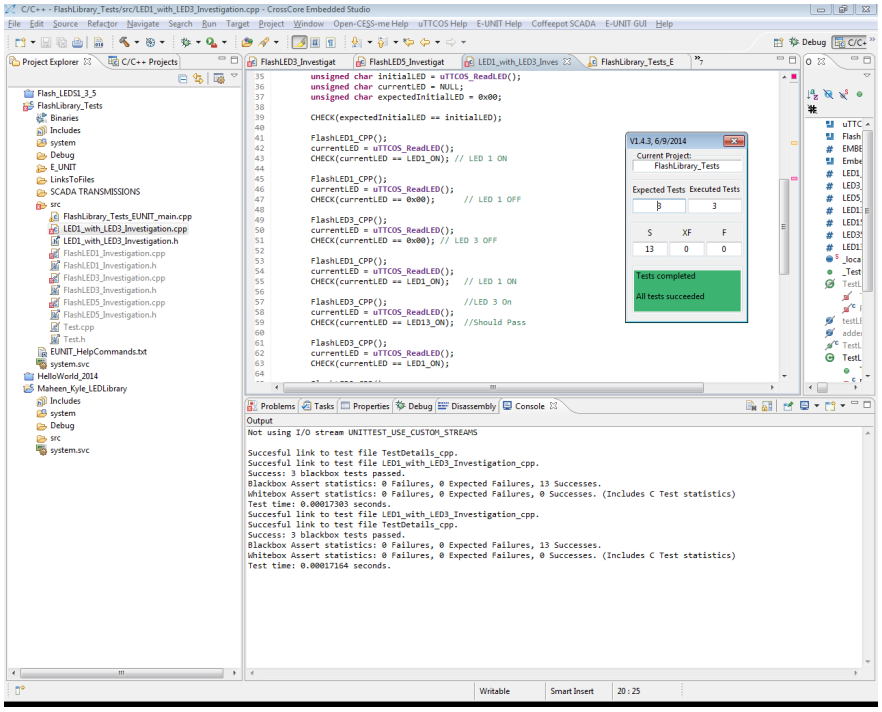

Figure 10: This screencapture shows the test results for LED1 flashing with LED3 and LED5. As can be seen, they passed.

Please see Sign off sheet for comfirmation of completion.

3 Static Library Project

3.1 Building Static Library

After having run all the tests, and modifying the customer code, we decided to compile the correct functions into a static library for linkage with future projects and labs. This was done by taking our modified customer CPP and header files and then building the project as a library instead of an executable. This compiling went off without errors and we were left with a library.

3.2 Testing Static Library

Now that our library had been built, we needed to ensure that it was still functioning correctly. To do this, we decided to exclude the previous code from the EUNIT test project, and the Customer project, and replace it by linking our library within the project properties.

The library was linked correctly with no errors, and our EUNIT and Customer Project functioned as expected. Please see figure below for screen capture.

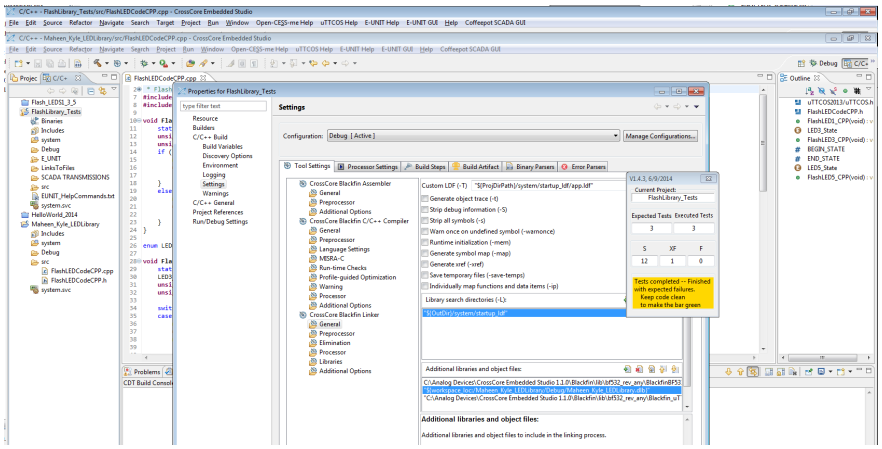

Figure 11: This screencapture shows the addition of our static library into the test project.

```
 2  * FlashLEDCodeCPP.h
 7 #include <stdio.h>
 8
 9 #ifndef FLASHLEDCODECPP_H_
10 #define FLASHLEDCODECPP_H_
11
12 void FlashLED1_CPP(void);
13 void FlashLED3_CPP(void);
14 void FlashLED5_CPP(void);
15 void ResetStates_CPP(void);
16
17 #define CHECK_EQUAL_HEX(A,B) \
18     if (A == B) printf("0x%x == 0x%x\n", A, B);\
19 else printf("0x%x != 0x%x\n", A, B);\
20
21 #endif /* FLASHLEDCODECPP_H_ */
22
```

```
 2  * FlashLEDCodeCPP.cpp
 7 #include <uTTCOS2013/uTTCOS.h>
 8 #include "FlashLEDCodeCPP.h"
 9
10 // State Machine Variables (global)
11 bool LED1_is_on = false;
12 enum LED3_State {JUST_STARTED, IS_OFF, IS_ON};
13 LED3_State currentState = JUST_STARTED;
14 enum LED5_State {LED5_JUST_STARTED, LED5_IS_OFF, LED5_IS_ON, LED5_STAYS_ON};
15 LED5_State currentState_LED5 = LED5_JUST_STARTED;
16
17 // Reset to Base States
18 void ResetStates_CPP(void) {
19     LED1_is_on = false;
20     currentState = JUST_STARTED;
21     currentState_LED5 = LED5_JUST_STARTED;
22 }
23 #define LED1_BITMASK 0x01
24 void FlashLED1_CPP(void) {
25     unsigned char LEDCurrentState = uTTCOS_ReadLED();
26     unsigned char nxt_State = NULL;
27     if (!LED1_is_on) {
28         nxt_State = LED1_BITMASK | LEDCurrentState;
29         uTTCOS_WriteLED(nxt_State);  // Turn on LED 1
30         LED1_is_on = true;
31     }
32     else {
33         nxt_State = ~LED1_BITMASK & LEDCurrentState;
34         uTTCOS_WriteLED(nxt_State);  // Turn off LED 1
35         LED1_is_on = false;
36     }
37 }
38
39 #define LED3_BIT_MASK  0x04
40 void FlashLED3_CPP(void) {
41     LED3_State nextStateToDo = currentState;
42     unsigned char LEDCurrentState = uTTCOS_ReadLED();
43     unsigned char nxt_State = NULL;
44
45     switch (currentState) {
46     case JUST_STARTED:
47         nxt_State = ~LED3_BIT_MASK & LEDCurrentState;
48         uTTCOS_WriteLED(nxt_State); // Turn off LED 3
49         nextStateToDo = IS_OFF;
50         break;
51     case IS_OFF:
52         nxt_State = LED3_BIT_MASK | LEDCurrentState;
53         uTTCOS_WriteLED(nxt_State);  // Was off turn it on
54         nextStateToDo = IS_ON;
55         break;
56     case IS_ON:
57         nxt_State = ~LED3_BIT_MASK & LEDCurrentState;
58         uTTCOS_WriteLED(nxt_State); // Turn off LED 3
59         nextStateToDo = IS_OFF;
60     break;
61     }
62
```

```
63      currentState = nextStateToDo;
64  }
65
66  // Making some C++ extensions to handle "LED state machine"
67
68  #define BEGIN_STATE case
69  #define END_STATE break;
70
71  #define LED5_BITMASK 0x10
72
73  void FlashLED5_CPP(void) {
74      unsigned char LEDCurrentState = uTTCOS_ReadLED();
75      unsigned char nxt_State = NULL;
76      LED5_State nextStateToDo = currentState_LED5;
77
78      switch (currentState_LED5) {
79      BEGIN_STATE LED5_JUST_STARTED:
80          nxt_State = ~LED5_BITMASK & LEDCurrentState;
81          uTTCOS_WriteLED(nxt_State);              // Turn off LED 5
82          nextStateToDo = LED5_IS_OFF;
83      END_STATE
84
85      BEGIN_STATE LED5_IS_OFF:
86          nxt_State = LED5_BITMASK | LEDCurrentState;
87          uTTCOS_WriteLED(nxt_State);  // Was off turn it on
88          nextStateToDo = LED5_IS_ON;
89      END_STATE
90
91      BEGIN_STATE LED5_IS_ON:
92          nextStateToDo = LED5_STAYS_ON;        // Just on -- keep on
93      END_STATE
94
95      BEGIN_STATE LED5_STAYS_ON:
96          nxt_State = ~LED5_BITMASK & LEDCurrentState;
97          uTTCOS_WriteLED(nxt_State);                 // Was off turn it on
98          nextStateToDo = LED5_IS_OFF;
99      END_STATE
100     }
101
102     currentState_LED5 = nextStateToDo;
103 }
104
```

```
 2 * Author: kderbyma
 6
 7 #include <EmbeddedUnit/EmbeddedUnit.h>
 8 #include "LED1_with_LED3_Investigation.h"
 9 #include "FlashLED5_Investigation.h"
10 #include "FlashLED3_Investigation.h"
11 #include <uTTCOS2013/uTTCOS.h>
12 #include "FlashLED1_Investigation.h"
13 #include "Test.h"
14 void UpdateEunitGui(void);
15 extern volatile int useLongFileFormat;
16
17 extern void AutomatedTestLevelControl(void);
18
19 void RestartEunitGui(void);
20 void UpdateEunitGui(void);
21
22 int main(void)
23 {
24     int failureCount;
25
26     uTTCOS_InitLED();   // Modified
27
28     RestartEunitGui( );
29     UpdateEunitGui();
30
31     UnitTest::ProcessorSpecificStartup();
32
33     AutomatedTestLevelControl();
34
35     UnitTest::Test::GetTestList().ReverseListDirection();
36
37     bool showFail = true;   bool showXFail = true;
38     bool showSuccesses = false;
39
40 // TODO You can adjust UnitTest::RunAllTests( ) to show only failures -- Tue 2014/09/30 at
   02:01:42 PM
41 // TODO bool showSuccesses = false;;
42     failureCount = UnitTest::RunAllTests(showFail, showXFail, showSuccesses);
43
44     UnitTest::Test::GetTestList().ReverseListDirection();
45     failureCount = UnitTest::RunAllTests(showFail, showXFail, showSuccesses);
46
47     UpdateEunitGui();
48     return failureCount;
49 }
50
```

```
 2 * Author: kderbyma
 6 #include <uTTCOS2013/uTTCOS.h>
 7 #include "FlashLEDCodeCPP.h"
 8
 9 #define EMBEDDEDUNIT_LITE
10 #include <EmbeddedUnit/EmbeddedUnit.h>
11 //#include "FlashLED1_Investigation_cpp.h"
12
13
14 TEST_CONTROL(FlashLED1_Investigation_cpp);
15
16 #if 0
17 TEST(FlashLED1_Investigation_cpp_GUIUpdate) {
18     UpdateEunitGui();  // Conditionally compile this line (use #if 1) to cause an GUI update
   based on last completed test
19 }
20 #endif
21
22 TEST(LED1_Investigation1)
23 {
24     uTTCOS_WriteLED(0); // Initialize to LEDS 0
25     ResetStates_CPP();
26
27     unsigned int initialLED = uTTCOS_ReadLED();
28     unsigned int expectedInitialLED = 0;
29     CHECK_EQUAL_HEX(expectedInitialLED, initialLED);
30
31     FlashLED1_CPP();
32
33     unsigned int notExpected = 0;
34     unsigned int newLED1_V1 = uTTCOS_ReadLED();
35     CHECK_EQUAL_HEX(newLED1_V1, 0x01);
36
37     FlashLED1_CPP();
38
39     unsigned int offExpected = 0;
40     unsigned int offLED1_V1 = uTTCOS_ReadLED();
41     CHECK_EQUAL_HEX(offLED1_V1, offExpected);
42
43 }
44
45
46 TEST_FILE_RUN_NOTIFICATION(FlashLED1_Investigation_cpp);
47
```

```
 2 * Author: kderbyma
 6 #include <uTTCOS2013/uTTCOS.h>
 7 #include "FlashLEDCodeCPP.h"
 8
 9 #define EMBEDDEDUNIT_LITE
10 #include <EmbeddedUnit/EmbeddedUnit.h>
11 //#include "FlashLED3_Investigation_cpp.h"
12
13
14 TEST_CONTROL(FlashLED3_Investigation_cpp);
15
16 #if 0
17 TEST(FlashLED3_Investigation_cpp_GUIUpdate) {
18     UpdateEunitGui();  // Conditionally compile this line (use #if 1) to cause an GUI update
   based on last completed test
19 }
20 #endif
21
22 TEST(FlashLED3_Investigation1)
23 {
24     uTTCOS_WriteLED(0); // Initialize to LEDS 0
25     ResetStates_CPP();
26         unsigned char initialLED = uTTCOS_ReadLED();
27         unsigned char expectedInitialLED = 0;
28         CHECK_EQUAL_HEX(expectedInitialLED, initialLED);
29
30         FlashLED3_CPP();
31
32         unsigned char offExpected = 0;
33         unsigned char offLED3_V1 = uTTCOS_ReadLED();
34         CHECK_EQUAL_HEX(offLED3_V1, offExpected);
35
36         FlashLED3_CPP();
37
38         unsigned char newLED3_V1 = uTTCOS_ReadLED();
39         CHECK_EQUAL_HEX(newLED3_V1, 0x04);
40
41 }
42
43 TEST_FILE_RUN_NOTIFICATION(FlashLED3_Investigation_cpp);
44
```

```
 2 * Author: kderbyma
 6
 7 #include <uTTCOS2013/uTTCOS.h>
 8 #include "FlashLEDCodeCPP.h"
 9
10 #define EMBEDDEDUNIT_LITE
11 #include <EmbeddedUnit/EmbeddedUnit.h>
12 //#include "FlashLED5_Investigation_cpp.h"
13
14
15 TEST_CONTROL(FlashLED5_Investigation_cpp);
16
17 #if 0
18 TEST(FlashLED5_Investigation_cpp_GUIUpdate) {
19     UpdateEunitGui();  // Conditionally compile this line (use #if 1) to cause an GUI update
   based on last completed test
20 }
21 #endif
22
23 TEST(FlashLED5_Investigation1)
24 {
25     uTTCOS_WriteLED(0); // Initialize to LEDS 0
26     ResetStates_CPP();
27         unsigned char initialLED = uTTCOS_ReadLED();
28         unsigned char expectedInitialLED = 0;
29         CHECK_EQUAL_HEX(expectedInitialLED, initialLED);
30
31         FlashLED5_CPP();
32
33         unsigned char offExpected = 0;
34         unsigned char offLED5_V1 = uTTCOS_ReadLED();
35         CHECK_EQUAL_HEX(offLED5_V1, offExpected);
36
37         FlashLED5_CPP();
38
39         unsigned char notExpected = 0;
40         unsigned char newLED5_V1 = uTTCOS_ReadLED();
41         CHECK_EQUAL_HEX(newLED5_V1, 0x10);
42
43 }
44
45 TEST_FILE_RUN_NOTIFICATION(FlashLED5_Investigation_cpp);
46
```

```
 2 * Author: kderbyma
 6 #include <uTTCOS2013/uTTCOS.h>
 7 #include "FlashLEDCodeCPP.h"
 8
 9 #define EMBEDDEDUNIT_LITE
10 #include <EmbeddedUnit/EmbeddedUnit.h>
11 //#include "LED1_with_LED3_Investigation_cpp.h"
12
13 // LED STATES
14 #define LED1_ON     0x01
15 #define LED3_ON     0x04
16 #define LED5_ON     0x10
17 #define LED13_ON    0x05
18 #define LED15_ON    0x11
19 #define LED35_ON    0x14
20 #define LED135_ON   0x15
21
22
23 TEST_CONTROL(LED1_with_LED3_Investigation_cpp);
24
25 #if 0
26 TEST(LED1_with_LED3_Investigation_cpp_GUIUpdate) {
27     UpdateEunitGui();  // Conditionally compile this line (use #if 1) to cause an GUI update
   based on last completed test
28 }
29 #endif
30
31 TEST(LED1_with_LED3_Investigation1)
32 {
33     uTTCOS_WriteLED(0); // Initialize to LEDS 0
34     ResetStates_CPP(); // Reset LED State Machines
35         unsigned char initialLED = uTTCOS_ReadLED();
36         unsigned char currentLED = NULL;
37         unsigned char expectedInitialLED = 0x00;
38
39         CHECK(expectedInitialLED == initialLED);
40
41         FlashLED1_CPP();
42         currentLED = uTTCOS_ReadLED();
43         CHECK(currentLED == LED1_ON); // LED 1 ON
44
45         FlashLED1_CPP();
46         currentLED = uTTCOS_ReadLED();
47         CHECK(currentLED == 0x00);      // LED 1 OFF
48
49         FlashLED3_CPP();
50         currentLED = uTTCOS_ReadLED();
51         CHECK(currentLED == 0x00); // LED 3 OFF
52
53         FlashLED1_CPP();
54         currentLED = uTTCOS_ReadLED();
55         CHECK(currentLED == LED1_ON);   // LED 1 ON
56
57         FlashLED3_CPP();                //LED 3 On
58         currentLED = uTTCOS_ReadLED();
59         CHECK(currentLED == LED13_ON);  //Should Pass
60
```

```
61          FlashLED3_CPP();
62          currentLED = uTTCOS_ReadLED();
63          CHECK(currentLED == LED1_ON);
64
65          FlashLED3_CPP();
66          currentLED = uTTCOS_ReadLED();
67          CHECK(currentLED == LED13_ON);
68
69          FlashLED1_CPP();
70          currentLED = uTTCOS_ReadLED();
71          CHECK(currentLED == LED3_ON);
72
73          FlashLED5_CPP();
74          currentLED = uTTCOS_ReadLED();
75          CHECK(currentLED == LED3_ON);
76
77          FlashLED5_CPP();
78          currentLED = uTTCOS_ReadLED();
79          CHECK(currentLED == LED35_ON);
80
81          FlashLED5_CPP();
82          currentLED = uTTCOS_ReadLED();
83          CHECK(currentLED == LED35_ON);
84
85          FlashLED5_CPP();
86          currentLED = uTTCOS_ReadLED();
87          CHECK(currentLED == LED3_ON);
88
89
90
91  }
92
93  TEST_FILE_RUN_NOTIFICATION(LED1_with_LED3_Investigation_cpp);
94
```