9th Workshop on Teaching Software Testing, Melbourne, Florida, January 2010

Experiences in using TDD as a key component in laboratories when teaching hardware / software co-design of embedded systems.

Presenter: Michael Smith,
Electrical and Computer Engineering,
University of Calgary,
Calgary, Alberta, Canada
Email: smithmr @ ucalgary.ca

James Miller,
Electrical and Computer Engineering,
University of Alberta,
Edmonton, Alberta, Canada
Email: jm @ece.ualberta.ca

Abstract: This paper provides background on why the presenter decided that taking a test driven development (TDD) approach offered advantages when teaching hardware / software co-design. A discussion is provided of the issues that arose while using TDD ideas with 3rd and 4th year University students designing and testing software interfaces for embedded system peripherals.

Introduction

This paper provides details of the personal experiences of a computer engineering trained instructor attempting to introduce test driven development (TDD) ideas to assist 3rd and 4th year University students to a better understanding of the concepts behind the “hands-on” laboratory components of an embedded system interfacing and a computer architecture course.

Background

Having used and taught microprocessors since the 80’s, the presenter’s attitude towards embedded software development drastically changed on becoming aware of the Personal Software Process (PSP) [Humphrey, 1997]. One key PSP epiphany remained with the instructor long after he was no longer formally recorded PSP metrics. If he was having a hard time in designing, coding, testing and releasing more than 30 lines of code per hour; should there be any expectation that a student’s productivity level be any higher? This led to a major modification of the instructor’s attitude of what could be accomplished, code-wise, in laboratories, assignments and exam questions.

A second issue that arose from this brief foray with PSP, was the realization that each time the course laboratory notes were updated to cover a new “term project”, the instructor’s code defect / error rate was initially (embarrassingly) high until the instructor once again remembered the peculiarities of the development environment, processor architecture and peripherals. However, such familiarity will not occur for the students taking a new course till the last part of the term. The late gaining of familiarity is especially true for some student since the laboratory experience involves a combination of software and hardware with which the students are totally unfamiliar. Thus fixing defects can be expected to impose a considerable time burden on the students; both in this and other courses involving code development.

A possible solution to this problem arose when this presenter was made aware of test driven development (TDD) by an experienced industrial software developer who returned for graduate studies under the presenter’s supervision and that student’s software engineering trained co-supervisor [Geras et al. 2004]. The remaining sections of this paper detail the steps used by the instructor to move the perceived business advantages of using TTD [e.g. Sanchez et al., 2007] into the new environment of designing and testing in embedded systems during 3rd and 4th year University courses.

Initial progress with introducing Embedded TDD

The Schulich School of Engineering program has three degree streams – Electrical Engineering, Computer Engineering and Software Engineering. The first attempt at using embedded test driven development (E-TDD) was within the electrical engineering stream with students exposed to minimal formalized testing during their program.

Three issues arose immediately: (A) a lack of student acceptance for the real need for testing for a course that involved only 5 (hands-on) laboratory periods (15 scheduled hours)); (B) a lack of a suitable testing framework for embedded development; and (C) a corresponding lack of mentorship for both instructor and students on how to use the TDD approach within the new environment.

The instructor’s existing approach was to treat the course laboratories as incremental, interdependent, prototypes directed towards a common final term project. It was felt that continuing this long-term orientation towards the laboratories would get the students to accept the need for testing in this course more than with their other early computing courses. The second issue was solved by identifying a light weight xUnit framework – *CppUnitLite* [Feathers, 2004] – that (i) looked easy to learn; (ii) looked small enough to be compiled and loaded within the limited memory of the standard embedded system evaluation board; and (iii) was non-scripted so that the instructions for the system to perform the tests did not disrupt the expectations of being able to develop code for a real time (time critical) system. Initial experiences with the resulting *Embedded UnitLite* running on an Analog Devices Blackfin (ADSP-BF533) evaluation board were reported in "embedded system" commercial magazines [Smith et al. 2005a] and at an Agile conference [Smith et al, 2005b].

Further Embedded TDD development

Two things prevented *Embedded UnitLite* from ending up as an instructor's passing whim. The first was the chance reading of a comment that demonstrated the common concepts between the use of the scientific method in other aspects of engineering and the use of TDD in code development [Mugridge, 2003]. This suggested the possible easy acceptance of the TDD concepts by students because of existing familiarity. The second was the fact that a major defect in the actual processor silicon was uncovered by the first 3rd year computer engineering class that was trained to use a prototype automated testing framework adapted for the embedded environment (see Fig. 1).

Given that this silicon defect had been unrecognized by industrial developers working with "tens of thousands" of the processor, it was considered that *Embedded UnitLite* showed considerable potential as both a teaching and an industrial development tool.

With the instructor's and student's interest in biomedical engineering, research work was undertaken in applying TDD across all four stages of an extreme programming inspired (XPI) life cycle for biomedical instrument development [Chen et al., 2005] [Miller et al., 2006], [Smith et al., 2009a, 2009b].

In addition, the testing framework was extended to a wide variety of processor families to better support its industrial application and facilitate its use across a wide range of student 4th year design embedded projects.

Initial Educational experiences

Initial educational experiences with the prototype Embedded Unit have been published [Smith et al, 2005a] [Miller and Smith, 2007]. The prototype *EmbeddedUnit* permitted the standard basic testing of algorithms: e.g. CHECK macros to perform black box testing on returned or modified variables. In addition, hardware assisted embedded specific extensions were added to track whether required low-level processor activities were performed by student's who were not experienced in concepts associated with the importance of test-coverage. For example WATCH_MEMORY_RANGE macros, included as part of "customer provided acceptance tests", were used to determine whether the students were properly initializing memory mapped external device registers to zero. This was intended to identify coding that would lead to immediate system crashes if students assumed that these control registers of the evaluation board peripheral retained their default (cold-start) zero values when used across multiple downloads of their programs to the evaluation board.

In hindsight, it has been recognized that there was always the potential for using the existing *CppUnitLite* testing framework within other electrical engineering courses directly involving programming or when students handle team design projects. This offered the possibility of dramatically increasing the exposure to testing concepts using one testing framework. However considering the over-loaded course content present within engineering programs, the practical application of TDD concepts in other courses never happened.

Recent Educational experiences

One of the biggest difficulties when adapting and adopting TDD into the teaching environment are the equivalent of that expected in a similar situation industry – the learning curve associated with a new "formal" testing approach. Compounding this problem are the additional educational issues of (a) the students are unfamiliar with testing; (b) the students are experiencing a learning curve associated with using a industrial strength development tool environment rather than a command-line based gcc compiler running within Unix; (c) the students are not using code to control a simulation but a real embedded system that will crash and lock-up if not treated correctly; and the most important (d) the time associated with attempting to successfully complete five other engineering courses in the same term.

To reduce the learning curve and time commitment we developed a GUI that connected to Analog Devices IDE to better automate the production and interpretation of the tests in the new embedded environment (see left side of Fig. 1).

In addition, we modified the testing environment to provide the ability to report on "Test successes" as well as the standard "Test failures". We would recommend that all testing frameworks used in teaching or initial industrial training be modified to support this mode of operation. This mode appeared to motivate the students to use the testing framework to produce "many" tests during the initial laboratories. However, in the later labs involving regression testing, this enthusiasm for reporting success became problematic. It appeared that students were not seeing new failing tests amongst the large number of reported successes. It was difficult to convince some students of the advantages of turning off this mode of reporting test metrics even when it was pointed out that they could still their success rates reported in a simple fashion in the "STOP-CAUTION-GO" traffic light bar present in the GUI (see Fig. 1).

Several students independently made an observation that has been reported many times in the literature – the futility of using an automated testing framework when the test code needed to generate the results used for testing was essentially the same as the code under test. This is particular true for the short code segments that are typically needed to initialize the embedded device peripherals. The problem was enhanced by the desire of the instructor to ensure that the student did not need to write more 40 -- 50 lines of code / hour across the 3 hour laboratory session. This was considered a reasonable extension to the instructor's own 20 lines or code per hour capability given that the students were presented with a nearly complete design and that a certain number of remaining code defects could be tolerated in a prototype developed during the limited laboratory time available.

In principle this issue would be reduced if the student laboratory pair split its effort with one student writing the test and the other student writing the code. However given the learning curve associated with the new embedded system course material, the team members were collaborating so closely that any misconceptions would remain common and reflected in both code and test.

Given the time constraints and lack of student experience, a common observation was that the students were unable to conjure up a mental model of what they were attempting to accomplish. This meant that even when the students wrote tests that found faults in the system, they were unable to interpret the reported failures to solve the system problems.

This inability to interpret and then respond to the meaning of the reported failures had an unanticipated impact on the use of prebuilt (customer) acceptance tests provided by the instructor to cut down the physical amount of typing necessary to develop a full range of tests. The students were taught that the test suite was a living document subject to upgrade as additional knowledge about the system was gained. Therefore the students would consider it reasonable to modify the tests to pass given that the students now thought they understood the low level characteristics of the processor interface.

These problems were less evident in the 4th year course which explored the impact that processor architecture had on the developer's ability to ensure real-time performance when implementing typical digital signal processing algorithms (DSP). In this situation, there is a wide range of data available for testing, with the students familiar with DSP ideas from other courses. In addition, the tests were used in more of a "refactoring mode for speed' with the same tests repeatedly used to validate a wide range of algorithm implementations.

Conclusion

In this paper we have discussed the personal experiences of a computer engineering trained instructor attempting to introduce the advantages seen in business world application of TDD concepts into the environment of embedded system development. Considerable successes were found during laboratories, but the lack of mentorship (prior examples) is still considered a major factor in limiting the instructor ability to use properly use TDD to full advantage.

The embedded system course discussed in this paper has recently become a required course for computer and software engineering students. It will be interesting to watch whether the students' attitude to using the *EmbeddedUnit* testing framework changes since these new groups have both been exposed *JUNIT* while developing Java code.

Acknowledgements

The authors acknowledge the financial support from Analog Devices (US), University of Calgary and NSERC, the Natural Sciences and Research Council of Canada. Mike Smith is Analog Devices University Ambassador. Feedback was obtained from many undergraduate students who have used Embedded UnitLite during ENCM415, ENCM511and ENCM515 courses at the University of Calgary, with particular thanks to R. Brey, M. Hafez, and J. Woehr.

References

[Geras et al., 2004] A. Geras, A, M. R. Smith and J. Miller. "*More or Different University Testing Courses: The Implications of a Recent Survey of Alberta Software Organizations*", I.E.E.E. Canadian Conference on Computer and Software Engineering Education, Calgary, Canada, March 2004.

[Chen et al., 2005] J. Chen, M. Smith, A. Geras, J. Miller., "*Making Fit/FitNesse Appropriate for Biomedical Engineering Research,*" Proc. 7th Int'l Extreme Programming and Agile Processes in Software, LNCS 4044, Springer, 2006, pp. 186–190.

[Feathers, 2004] *CppUnit Wiki*; cppunit.sourceforge.net/cppunit-wiki. last accessed January, 2010.

[Humphrey, 1997] W. S. Humphrey, "*Introduction to the Personal Software Process*", Addison-Wesley, 1997.

[Miller et al., 2006] J. Miller, M. Smith, S. Daenick, J. Chen, J. Qiao, F. Huang, A. Kwan, M. Roper, "*An XP inspired test-oriented life-cycle production strategy for building embedded biomedical applications*", TAIC PART, 2006.

[Miller and Smith, 2007] J. Miller, M. Smith, *"A TDD Approach to Introducing Students to Embedded Programming"*, The 12th Annual Conference on Innovation Technology in Computer Science Education (ItiCSE), 2007.

[Mugridge, 2003] R. Mugridge. "*Test driven development and the scientific method*". Proceedings of the Agile Development Conference, 2003.

[Sanchez et al., 2007]. J Sanchez, Williams L., Maximilien E.M., "*A Longitudinal Study of the Use of a Test-Driven Development Practice in Industry*", Proc. Agile 2007 Conference, IEEE CS Press, 2007.

[Smith et al., 2005a] M. R. Smith, A. Martin, L. Huang, M. Bariffi, A. Kwan, W. Flaman, A. Geras, J. Miller, "*A look at test driven development (TDD) in the embedded environment: Part 1 and Part 2*", Circuit Cellar magazine, Vol. 176, pp 34 – 39, March 2005; Vol. 177, pp 60 – 67, April 2005.

[Smith et al., 2005b] M. Smith, A. Kwan, A. Martin, J. Miller, "*E-TDD – Embedded Test Driven Development: A Tool for Hardware-Software Co-design Projects*", 6th International Conference on eXtreme Programming and Agile Processes in Software, 2005

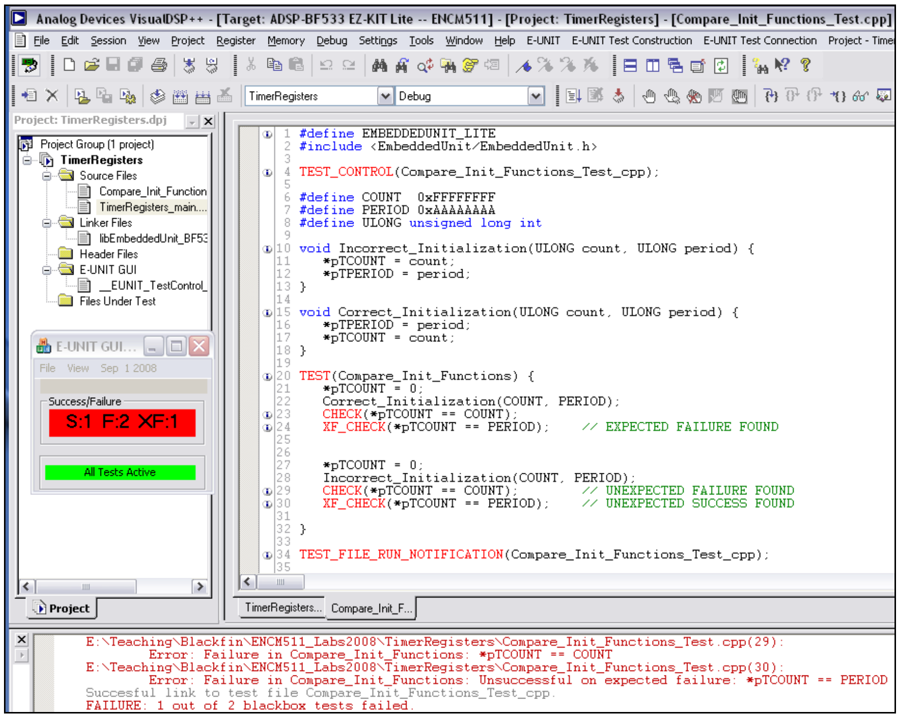
[Smith et al, 2009a] M. Smith, J. Miller, L. Huang, A. Tran, *"A More Agile Approach to Embedded System Development"*, IEEE Software, Vol. 26 no. 3, pp 50-57 May/June 2009.

[Smith et al, 2009b] M. Smith, J. Miller, S. Daeninck, "*A Test-oriented Embedded System Production Methodology*", Journal of Signal Processing Systems Volume 56, Number 1, 69-89, July 2009.

About the authors

Mike Smith is a full professor in Electrical and Computer Engineering at the Schulich School of Engineering, University of Calgary, Calgary, Canada. His research interests can be described as "Finding reliable methods for (A) developing biomedical algorithms and then (B) moving the developed algorithm down to a biomedical instrument (embedded systems)" For links to lessons and laboratories using EmbeddedUnit in 3rd and 4th year Embedded system courses -- see *enel.ucalgary.ca/People/Smith*

James Miller is a full professor in Electrical and Computer Engineering at the University of Alberta, Edmonton, Canada. He has published over one hundred refereed journal and conference papers on Software and Systems Engineering (see www.steam.ualberta.ca for details on recent directions); and regularly appears in the Journal of Systems and Software list of "top scholars". He currently serves on the program committee for the IEEE International Symposium on Empirical Software Engineering and Measurement; and sits on the editorial board of the Journal of Empirical Software Engineering.


Analog Devices VisualDSP++ - [Target: ADSP-BF533 EZ-KIT Lite -- ENCM511] - [Project: TimerRegisters] - [Compare_Init_Functions_Test.cpp]
File Edit Session View Project Register Memory Debug Settings Tools Window Help E-UNIT E-UNIT Test Construction E-UNIT Test Connection Project - Timer
TimerRegisters
Debug
Project: TimerRegisters.dpj
Project Group (1 project)
TimerRegisters
Source Files
Compare_Init_Function
TimerRegisters_main....
Linker Files
libEmbeddedUnit_BF53
Header Files
E-UNIT GUI
__EUNIT_TestControl_
Files Under Test
E-UNIT GUI...
File View Sep 1 2008
Success/Failure
S:1 F:2 XF:1
All Tests Active
Project
TimerRegisters...
Compare_Init_F...

```
 1 #define EMBEDDEDUNIT_LITE
 2 #include <EmbeddedUnit/EmbeddedUnit.h>
 3
 4 TEST_CONTROL(Compare_Init_Functions_Test_cpp);
 5
 6 #define COUNT  0xFFFFFFFF
 7 #define PERIOD 0xAAAAAAAA
 8 #define ULONG unsigned long int
 9
10 void Incorrect_Initialization(ULONG count, ULONG period) {
11     *pTCOUNT = count;
12     *pTPERIOD = period;
13 }
14
15 void Correct_Initialization(ULONG count, ULONG period) {
16     *pTPERIOD = period;
17     *pTCOUNT = count;
18 }
19
20 TEST(Compare_Init_Functions) {
21     *pTCOUNT = 0;
22     Correct_Initialization(COUNT, PERIOD);
23     CHECK(*pTCOUNT == COUNT);
24     XF_CHECK(*pTCOUNT == PERIOD);     // EXPECTED FAILURE FOUND
25
26
27     *pTCOUNT = 0;
28     Incorrect_Initialization(COUNT, PERIOD);
29     CHECK(*pTCOUNT == COUNT);         // UNEXPECTED FAILURE FOUND
30     XF_CHECK(*pTCOUNT == PERIOD);     // UNEXPECTED SUCCESS FOUND
31
32 }
33
34 TEST_FILE_RUN_NOTIFICATION(Compare_Init_Functions_Test_cpp);
35
```

```
E:\Teaching\Blackfin\ENCM511_Labs2008\TimerRegisters\Compare_Init_Functions_Test.cpp(29):
        Error: Failure in Compare_Init_Functions: *pTCOUNT == COUNT
E:\Teaching\Blackfin\ENCM511_Labs2008\TimerRegisters\Compare_Init_Functions_Test.cpp(30):
        Error: Failure in Compare_Init_Functions: Unsuccessful on expected failure: *pTCOUNT == PERIOD
Succesful link to test file Compare_Init_Functions_Test_cpp.
FAILURE: 1 out of 2 blackbox tests failed.
```

Figure 1: Screen capture of the *Embedded-Unit* tests for initializing the Blackfin core timer via two apparently equivalent C++ routines (Lines 11 – 12 and 16 -- 17). Undocumented silicon level behaviour resulted in one assert unexpectedly failing (Line 29), and another unexpectedly passing (Line 30). (From [Smith et al, 2009a]) .