Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE

Shuvra S. Bhattacharyya[1], William Plishker[1],

***Chung-Ching Shen[1]**, and Ayush Gupta[2]*

Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies[1],
Department of Physics[2]
University of Maryland, College Park, MD 20742, USA

Taipei, Taiwan, October 13, 2011

Outline

- Introduction: test-driven design practices for embedded systems education
- The DSPCAD Integrative Command Line Environment (DICE)
- Case Studies – Courses Offered at the University of Maryland
- Summary and Future Directions

Introduction

- For high productivity, high reliability hardware/ software development, testing should be integrated deeply into the design process.
  - Providing effective tests is an important part of overall project design.
  - Automated test execution is critical to practical use of testing methods.
- Embedded and high performance software must often utilize multiple languages and multiple platforms and transcoding between an initial application specification and software for the final implementation.
  - E.g., MATLAB, C, C++, CUDA, Verilog, and VHDL
  - Heterogeneous language are often involved in the same project at different stages of the design flow.

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Unit Testing I

- Tests are written concurrently with application code.
  - Tests are not written as an afterthought to the development process (e.g., code-driven approach)
- Tests are maintained separately from the application
  - Clean separation of testing code and application code
  - Testing code is normally not distributed/shipped with the application
  - Similar to the *testbench* concept in use of hardware description languages

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Unit Testing 2

- Tests are not invoked as part of the application; they are run separately.
- Tests exercise *isolated* objects and features.
- Tests are independent. Tests are not dependent on the results of other tests.
- Tests are short and simple.
- Complex designs are tested through top-down, hierarchical use of short and simple tests.

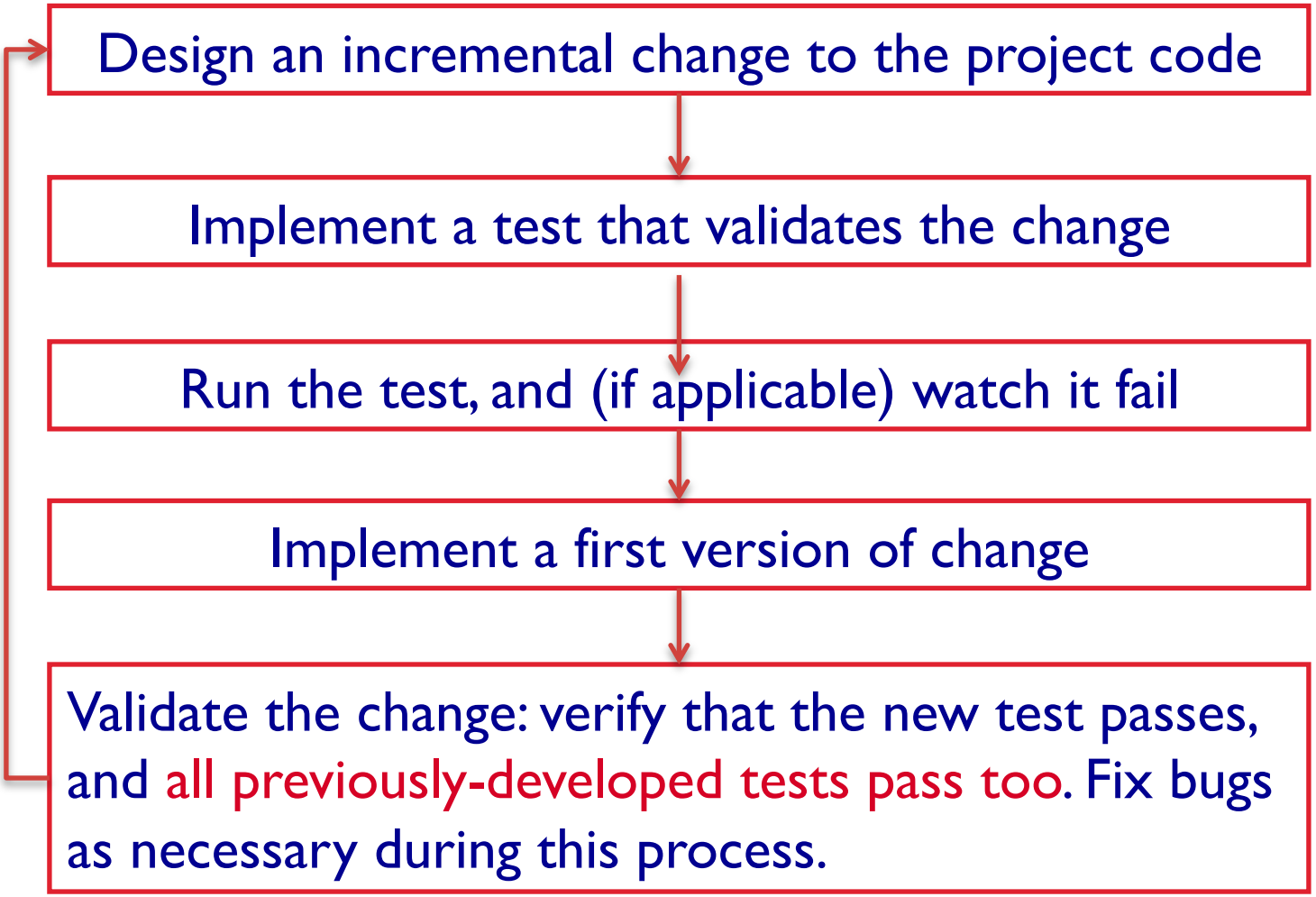
S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Application of Unit Testing When Changing Code [Hamill 2005]

- 1. Write a test of the new code and see it fail.
  - (The "seeing it fail" part generally applies to tests that affect functionality, not organizational changes)
  - Whenever applicable, don't skip this step!
- 2. Write the new code
  - Make a simple change to the code: build the new code incrementally through multiple iterations of this 3-step process.
- 3. Repeat testing and bug fixing until the new test succeeds and all previously-developed tests (in your "test suite") succeed as well.
  - This gives confidence that the new code works and does not break any of the other code.

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Incremental Development Methodology



S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Approaches to Unit Testing

- Various language-specific frameworks are available to facilitate unit testing
  - For example, JUnit, CppUnit.
- However, unit testing frameworks can be applied effectively in user-designed ways also through
  - Test organization and coding conventions
  - Separate compilation
  - Scripts
  - **diff**

Note: It is important that students understand distinctions between design methods and tools that assist in applying these methods.

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

DICE

DICE = The DSPCAD Integrative Command Line Environment [Bhattacharyya 2011]

Available from:
http://www.ece.umd.edu/DSPCAD/projects/dice/dice.htm

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

What is DICE?

- Website: http://www.ece.umd.edu/DSPCAD/projects/dice/dice.htm
- A **Bash**-based project development environment that emphasizes
  - Cross-platform, command-driven operation
  - Language-agnostic operation; integration across heterogeneous design languages
  - Support for model-based design
  - Unit testing, and test-driven design
  - Ease of learning, use, and interoperability for interdisciplinary design teams

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

What DICE is *not*

- A shell
- A software synthesis tool
- A compiler
- A replacement for language-specific development tools and IDEs
- A debugger, simulator, or transcoder

Instead, DICE is a command-line solution to utilize all of these existing kinds of tools more effectively, especially for cross-platform design.

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Example: Utility Scripts Provided in DICE for Efficient Directory Navigation

- The DICE utilities for directory navigation allow one to label directories with arbitrary (user-defined) identifiers,
- … and to move to directories by simply referencing these identifiers (rather than the complete directory path).
- This makes it very easy to "jump" from one directory to another.
- The main DICE utility related to directory navigation is **dlk** (the Directory LinKing utility)
- Other navigation-related utilities include **rlk**, and **plk**.

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

dlk Example

- Example usage:

  cd ~/disps/proj/proj1

  dlk p1

  cd ~/disps

  g p1

- After the above sequence of commands, the user will end up in **~/disps/proj/proj1** (assuming that this directory exists).

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

dxtest

- **dxtest** is a DICE utility (*not* a standard UNIX utility) for running all tests in a given test suite (or sub-suite)
- **dxtest** recursively traverses the directory tree (rooted at the current working directory), and runs all tests
  - directory names that start with "test" are identified as belonging to the test suite
  - … only these subdirectories are visited in the recursive traversal
- When used properly, **dxtest** can save an enormous amount of effort in the test validation process
- When executing, the actual output of the module under test is compared with the correct expected behavior

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Individual Test Suite

- Each test for a software unit should be placed in a separate directory that is organized in a certain way according to the DICE-based conventions for testing.
- The directory associated with a given test is called an individual test subdirectory (**ITS**).
- The name of an ITS must begin with "test" (e.g., test01, test02, test-A, test-B, test-simple1, etc.)

REQUIRED Components of an ITS (1/2)

- A **README.txt** file that provides a brief explanation of what is being tested by the test.
  - That is, what is distinguishing about this test compared to the other tests in the test suite.
- A **makeme** file that performs all necessary compilation steps (e.g. driver program) that are needed for the test.
  - Note: that this typically does *not* include compilation of the source code that is under test.
- A **runme** file that runs the test and directs all normal output to standard output and all error output to standard error.

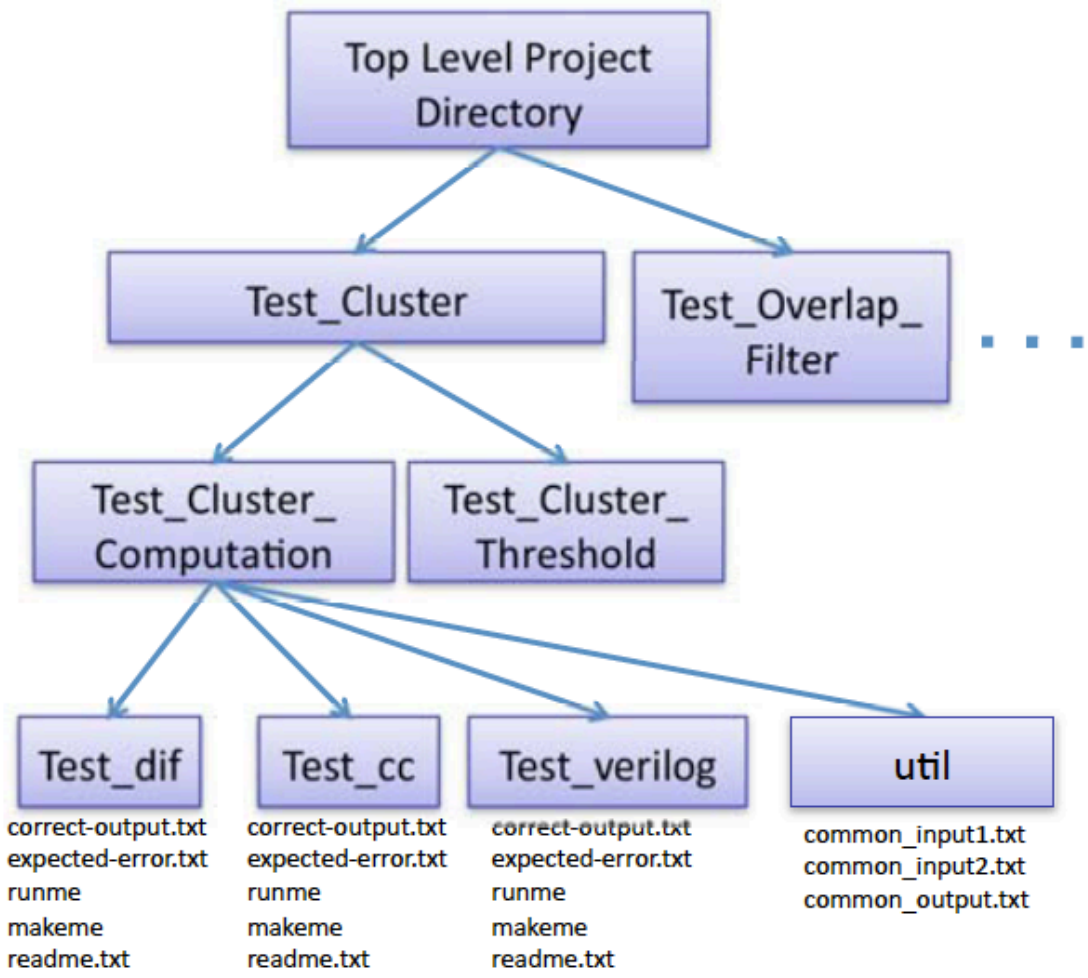
S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

REQUIRED Components of an ITS (2/2)

- Any input files that are needed for the test
- A file called **correct-output.txt** that contains the standard output text that should result from the test.
- A file called **expected-errors.txt** that contains the standard error text that should result from the test.
- NOTE: If one or more output files are created when running the test, and one wants to verify the correctness of these output files as part each test run, then the output files should be concatenated to standard output (e.g., using the **cat** command) by runme.

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Example: The DICE Testing Approach



S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Case Studies: Courses Offered at the University of Maryland

- 2007, 2008, 2009: Intermediate Programming Concepts for Engineers
  - Strengthen the exposure to programming concepts for undergraduates with EE majors.
  - Began integrating testing-driven design practices in the curriculum.
  - Established a foundation for our instructional approaches to testing-driven design concepts and application of DICE for exercising these concepts.

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Case Studies: Courses Offered at the University of Maryland

- Graduate courses in 2010: VLSI architectures (aimed at research-oriented students), FPGA system design (aimed at professional Master's students)
  - The objectives was introducing students to efficient verification and test planning as well as advanced programming models for embedded systems design targeted to multicore platforms and FPGAs.
  - Focus on hands-on design experience.
  - Covered design flows and tools available for arriving at efficient implementations on customized programmable platforms including digital signal processors, GPUs, and FPGAs.

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Case Studies: Courses Offered at the University of Maryland (continued)

- DICE facilitated these courses by providing a single umbrella under which other design environments were used, giving students a more unified programing experience than a conventional approach would provide.
- Example of a **makeme** script for a C-based Gaussian filter project.

```
gcc -c -Wall -pedantic -I$UXSRC driver.c
gcc -Wall -pg -pedantic -o driver driver.o \
$UXBIN/bmp_file_write.o \
$UXBIN/gfilter.o \
...
```

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Case Studies: Courses Offered at the University of Maryland (continued)

- Example of a **makeme** script for a CUDA-based Gaussian filter project

```
nvcc -I$UXSRC driver.c -o driver.o
nvcc -o driver \
$UXBIN/driver.o
$UXBIN/gfilter_cuda.o \
…
```

- Example of a **runme** script for an ITS

```
./driver coeffs.txt ../util/lena512.bmp out.bmp > \
        diagnostics.txt
```

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Case Studies: Courses Offered at the University of Maryland (continued)

- Example of a **makeme** script for a Verilog project

```
function makeone {
    if ! [ -d ../bin ]; then
        mkdir ../bin
    fi
    vlib ../bin
    vmap work ../bin
    vlog -work ../bin \$1.v
}
makeone fifo
makeone fir
makeone pea
```

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Case Studies: Courses Offered at the University of Maryland (continued)

- Example of a **makeme** script for a Verilog testbench

```
if ! [ -d work ]; then
    vlib work
    vmap work work
fi
vlog testbench.v
```

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Case Studies: Courses Offered at the University of Maryland (continued)

- Example of a **runme** script for an ITS of a Verilog project

```
UXLIB=../../../bin
if [ -f out.txt ]; then
    rm -r out.txt
Fi

vsim -c -L \$UXLIB -do "run -all" testbench > \
        transcript
cat out.txt 2> err.txt
```

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Summary and Future Directions

- Motivated the deep integration into embedded systems education of testing-driven design and methods for cross-platform and language-independent, test suite development.
- Introduced a software package called the DSPCAD Integrative Command Line Environment (DICE).
- DICE was applied to the courses at UMD aimed at a diverse groups of students were offered at UMD.
  - An open source project under active development that is publicly available
  - Also being used by two community colleges near UMD.
- Systematic research on the instructional effectiveness of the test-driven approach using DICE across various courses.
  - Traditional instruction versus testing-driven approach

S. S. Bhattacharyya, W. Plishker, C. Shen, and A. Gupta. *Teaching Cross-Platform Design and Testing Methods for Embedded Systems using DICE*, Taipei, Taiwan, October 13, 2011.

Questions

Acknowledgements

- This work was supported in part by the US National Science

References

- [Hamill 2005]. *Unit Test Frameworks*. O'Reilly & Associates, Inc., 2004.
- [Bhattacharyya 2011] The DSPCAD integrative command line environment: Introduction to DICE version 1.1. Technical Report UMIACS-TR-2011-10, Institute for Advanced Computer Studies, University of Maryland at College Park, 2011. http://drum.lib.umd.edu/handle/1903/11422.