Design Verification
COMS30026

WEEKLY STATUS UPDATE – W3

Kerstin Eder

Trustworthy Systems Lab

University of BRISTOL

Department of COMPUTER SCIENCE

Topics W1

- ✓ Introduction to DV
- ✓ Verification Hierarchy
- ✓ Fundamentals of Simulation-based Verification
  - Driving & Checking

- **Lab W1:**
  - get remote access to the EDA software
  - teach yourself Verilog ☺

Topics W2

- ✓Verification Tools
- ✓Hardware Design Languages
- ✓Verification Cycle, Methodology & Plan

- **Lab W2:**
  - Introduction to ModelSim/Questa
    - installed on linux lab machines
  - Work through mux testbench from Exercise 2

    https://uobdv.github.io/Design-Verification/

Topics W3

- ✓ Stimuli Generation Part I: Foundations
- Stimuli Generation Part II: Test Automation
- Checking

- **Lab W3:**
  - Introduction to **Practical 1** (Weeks 2-4)
    - Specification review
    - Debug calculator design using Modelsim/Questa
    - Example testbench on Blackboard

DESIGN VERIFICATION Practical 1:
Debugging a Calculator Design

This practical requires you to find bugs in a calculator design. The calculator specification is given below. The design HDL code is not available for this practical. You are expected to use a black box verification approach.

For simulation with ModelSim, the calculator design has been provided in the pre-compiled library `calc1_black_box`. The library is available from BlackBoard as a ZIP file. BlackBoard also provides you with instructions on how to set up your workspace with this library. The top-level module of the calculator design is called `calc1`. Please use the example testbench file provided on BlackBoard to check that you have set everything up correctly.

Calculator Specification

Input/Output Specification

The calculator has four commands: add, subtract, shift left and shift right. It can handle (but not process) four requests in parallel. All four requestors use separate input signals. All requestors have equal priority. Figure 1 shows the input output ports of the calculator.

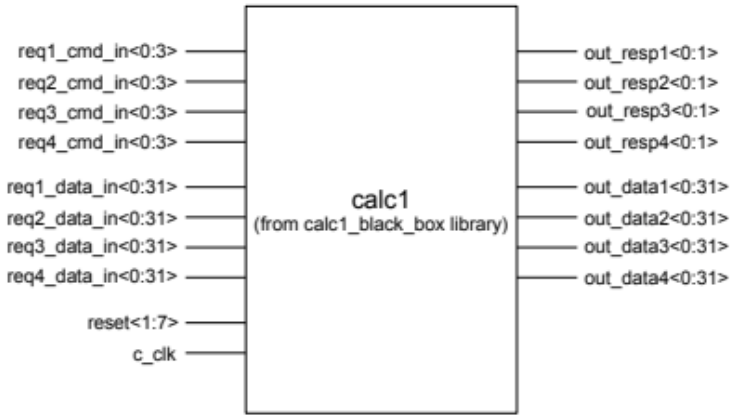

Figure 1: Top-level of Calculator Design

Opportunities – Week 3

- 2020 Wilson Research Group Verification Survey Results
  - Full recording by Harry Foster on Blackboard

Challenge

Photo from
https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

- Find out about the work of Edsger W. Dijkstra.
- In particular, find what he said about testing and its limitations.

Challenge

Photo from
https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

- Find out about the work of Edsger W. Dijkstra.
- In particular, find what he said about testing and its limitations.

"Program testing can be used to show the presence of bugs, but never to show their absence!"

— Edsger W. Dijkstra

Four necessary and sufficient conditions for deadlock

System Deadlocks

E. G. COFFMAN, JR.

Pennsylvania State University, University Park, Pennsylvania

M. J. ELPHICK

University of Newcastle upon Tyne, Newcastle upon Tyne, England

A. SHOSHANI

System Development Corporation, Santa Monica, California

E. G. Coffman, M. Elphick, and A. Shoshani. 1971. System Deadlocks. ACM Comput. Surv. 3, 2 (June 1971), 67–78.

DOI: https://doi.org/10.1145/356586.356588

A problem of increasing importance in the design of large multiprogramming systems is the, so-called, deadlock or deadly-embrace problem. In this article we survey the work that has been done on the treatment of deadlocks from both the theoretical and practical points of view.

Key words and phrases: deadlocks, deadly embraces, system deadlocks, multiprogramming, interlock problems

CR categories: 4.10, 4.32

1 INTRODUCTION

One of the objectives of recent developments in operating systems—incorporating multiprogramming, multiprocessing, etc.—has been to improve the utilization of system resources (and hence reduce the cost to users) by distributing them among many concurrently executing *tasks*. In any operating system of this type the problem of deadlock must be considered. Requests by separate tasks for resources may possibly be granted in such a sequence that a group of

But if we assume that T_2 is in the middle of some file on disk, then, because of hardware or software limitations, a preemption of the disk from T_2 can be effectively tantamount to a sacrifice in the processing already accomplished by T_2. Under these circumstances, if T_1 and T_2 have no (temporary) alternative to waiting, they will be deadlocked and never able to proceed.

As shown by this example, deadlocks, or "deadly embraces" as E. W. Dijkstra has called them, can arise even though no single task requires more than the total resources

This offers you an opportunity to learn material beyond what has been taught in this unit.

Next

- Recordings of lectures for **Week 3:**
  - ✓ Stimuli Generation Part I: Foundations
  - ✓ Stimuli Generation Part II: Test Automation
  - ✓ Checking
  - uobdv.github.io/Design-Verification/ shows a **weekly schedule of topics** to watch BEFORE the next session, ideally
  - Recordings are available from Blackboard unit page
- Tasks for you this week:
  - **Attend the lab session** with Xuan to get help with using ModelSim/Questa so you can do **Practical 1**
  - Submit issues found during your **Specification review** to Blackboard Discussion Forum

Questions

https://www.bristol.ac.uk/tsl