Design Flow for Embedded System Device Driver Development and Verification

Ross Dickson  
Virtutech, Inc  
dickson@virtutech.com

Jason Andrews  
Cadence  
jasona@cadence.com

Jakob Engblom  
Virtutech, AB  
jakob@virtutech.com
Context and Motivation

• Bugs in device drivers are common
  – The majority of embedded system bugs are attributed to software
    • Possibly because software is less expensive to fix
• Cadence has proven utility of constrained random testing in the Incisive Software Extensions to provoke bugs
• Virtutech has proven the efficiency of the Simics Virtual Platform to nonintrusively isolate and identify bugs
• A combined environment provides the embedded device driver writer with an improved environment to provoke, isolate, and identify bugs in device drivers for embedded systems
• A driver for a hardware accelerator for the Rule 30 Cellular Atomata on a virtual Freescale MPC8641D SoC running Linux version 2.6.23 is used as a running example
Software Setup

Virtual MPC8641D Board

MPC8641D SoC

Busybox

Test program exercising device driver

Other program

Device driver under test

CPU cores 0 and 1

Linux 2.6.23 kernel

Virtual MPC8641D Board

Virtual serial console

Serial port

RAM

Ethernet

MemCtrl

Timers

PCI

PCIe

MPIC

Device model under test

Simics

Host operating system

Host computer
• **Determine the items that must be exercised and monitored**
  - **Software**
    - the software functions to be called
    - their input parameters, and their outputs
    - Variables and data structures to be monitored
  - **Hardware**
    - Registers modified
    - Interrupts triggered

• **Examples:**
  - **Software:** device driver functions
    - open and write
  - **Software:** kernel functions
    - interrupt handlers.
Rule 30 Verification Plan

Create Verification Plan

Describe Software to be Exercised

- Done using Word
- Saved as XML file
Introduction to Coverage Driven Verification

Automatic Generation + Coverage Measurement = Coverage-Driven Verification

Constrained Randomization

Metrics

Define Verification Goals and Let Machines do the Work
An engineer writes directed test for each item in Test Plan:

It’s work determine and follow the paths
It’s work to verify the goal was reached
Poor coverage of non-goal scenarios

Redo if design changes
…so directed tests are not enough

- Tedious to write and maintain
  - Tens of thousands of lines of test code
  - Reuse to get to state quickly causes poor new coverage
  - Design spec changes cause delays

- Difficult to think of all required verification cases
  - Many different ways to reach a certain state

- Can't implement all identified tests in test plan within project schedule
Where did CDV Come From?

Can the same technology be used for embedded software?

Can it be used by an embedded software verification engineer?

- Automated Constrained Random Stimulus Generation
- Coverage Collection (Functional, Assertion, Code)
- Independent Assertion & Data Checking
Software Setup with Coverage Driven Verification

Virtual MPC8641D Board

MPC8641D SoC

Busybox

Test program exercising device driver

Other program

Device driver under test

Helper

CPU cores 0 and 1

Linux 2.6.23 kernel

Device model under test

MPIC

RAM

UART

Ethernet

MemCtrl

Timers

PCI

PCle

Virtual serial console

Virtual network

Set test program parameters

Observe hardware/software interface

Change hardware parameters

Coverage Driven Verification

Tests

Verification Planning & Management

vPlan

Host operating system

Host computer
Rule 30 Driver Execution and Coverage Analysis

Run sequences and collect coverage

Generated sequences with random data

Achieved Coverage
Rule 30 Driver Regression and Results

Automated Execution and Analysis

- Reads XML file
- Analyzes Results
Rule 30 Results

- **Coverage**
  - Ability to measure hardware model parameters
    - `time_to_result` for algorithm completion time to make sure Linux driver is timing independent
  - Ability to measure generated stimulus
    - Length of rule30 line
  - Ability to measure coverage on software
    - Mode of device open() system call: read, write, read/write
    - Return values from system calls
  - Ability to cross all of the above and find out, “Did I test short line length with fast hardware response time?”
• Bug Hunting: hitting corner cases not thought writing manual tests
  – Operations out of order
  – Parameters outside of expected values
  – Memory corruption: “serial8250: too much work for irq42”

• Learning
  – Understanding driver behavior
  – When does a read block waiting for results
  – How interrupts work
Repeat any run trivially
   – No need to rerun and hope for bug to reoccur
Stop & go back in time
   – Instead of rerunning program from start
     – Breakpoints & watchpoints backwards in time
     – Investigate exactly what happened this time
This control and reliable repeatability is very powerful for parallel code!

On hardware, only some runs reproduce an error

On virtual hardware, debugging is much easier
Thank You
What Virtutech Does

• Provider of Simics: a high-performance, high fidelity, full system simulator

  – **High Performance** – fast enough to run *real* software loads (typically 100’s of MIPS, up to multiple GIPS)

  – **High Fidelity** – run full production software, including firmware, device drivers, hypervisor, RTOS/OS, application software

  – **Full System** – simulate entire systems, not just processor cores, or SoCs, or boards
    • Complete machines, networks, backplanes

• The true value of Simics is through enablement of process change: Virtualized Software Development
Traditional Product/Project Development

Pre-Silicon Testing

Post-Silicon Testing

System Integration
start time and duration

Production, Manufacture

Marketing / Feature Validation

End User Documentation & Translation

Product Specification

Board Bring-up

Platform Development

Application Development

Hardware Development

Software Development

Prototypes
Agility, Higher Quality, Faster time to Market

True Iterative Development

System Bring-up

Production, Manufacture

Marketing / Feature Validation

End User Documentation & Translation
What is a Virtual Platform?

- A piece of software
- Running on a regular PC, server, or workstation
- Functionally identical to a particular hardware
- Runs the same software as the physical hardware system