OS Porting & Analysis for Dual Core ARM Cortex-A9 Based Systems

Simon Davidmann
Imperas
31 October 2012
Agenda

• Silicon without software is just sand...
  – Issues in embedded software development
  – OS porting, analysis and bring up

• What is a virtual platform?
  – Building a virtual platform
  – Requirements for a virtual platform development environment

• Case studies for virtual platform based software development
  – SMP Linux / Android
  – OS-related software: loadable kernel modules (LKM) for Linux
  – OS exception analysis
  – AMP system

• Summary, Q&A
Agenda

- Silicon without software is just sand...
  - Issues in embedded software development
  - OS porting, analysis and bring up
- What is a virtual platform?
- Case studies for virtual platform based software development
- Summary, Q&A
Silicon Without Software Is Just Sand

- Software development is costing more than chip development cost
- Embedded software is the critical path to system delivery
- Source code is doubling annually
- Software complexity is increasing dramatically with multi-core devices, multi-processor systems
- Products are defined by their software
Issues in Embedded Software Development

- Quality is critical
- Current development methodology breaks with increasing code complexity
- Time to market still counts!
- Management cannot manage the software development process: insufficient metrics
  - You cannot manage what you cannot measure
Focus for Today’s Presentation: Software Development, Porting, Bring Up

- Current development methodologies use hardware or host development systems
  - Actual hardware
  - Prototypes
  - x86 based development

- These methods lack controllability, visibility, accuracy
  - Controllability: can you test all relevant scenarios?
  - Visibility: if an error occurs, will it be observed by the test environment?
  - Accuracy: will software developed on x86 behave the same on an ARM-based device?

- Virtual platforms – software simulation – provide a complementary technology to the current methodology
  - Instruction accurate simulation promises controllability, visibility, ARM behavior
  - How to deliver on this promise?
Software Failures in Embedded Systems Are Bad!

This Car Runs on Code

February 5, 2010

The avionics system in the F-22 Raptor, the current U.S. Air Force frontline jet fighter, consists of about 1.7 million lines of software code. The F-35 Joint Strike Fighter, scheduled to be delivered to customers in 2010, will require about 5.7 million lines of software code to operate its onboard systems. And Boeing’s new 787 Dreamliner, scheduled to be delivered to customers in 2010, requires about 6.5 million lines of software code to operate its avionics and onboard support systems.

These are impressive amounts of software, yet if you bought a premium-class automobile recently, “it probably contains close to 100 million lines of software code,” says Manfred Broy, a professor of informatics at Technical University, Munich, and a leading expert on software in cars.

All that software executes on 70 to 100 microprocessor-based electronic control units (ECUs) networked throughout the body of your car.

FDA: Software Failures Responsible for 24% Of All Medical Device Recalls

June 20, 2012

Software failures were behind 24 percent of all the medical device recalls in 2011, according to data from the U.S. Food and Drug Administration, which said it is gearing up its labs to spend more time analyzing the quality and security of software-based medical instruments and equipment.

The FDA’s Office of Science and Engineering Laboratories (OSEL) released the data in its 2011 Annual Report on June 15, amid reports of a compromise of a Web site used to distribute software updates for hospital respirators. The absence of solid architecture and "principled engineering practices" in software development affects a wide range of medical devices, with potentially life-threatening consequences, the Agency said.

There is growing evidence that software security and integrity is a growing problem in the medical field. In October 2011, for example, security researcher Barnaby Jack demonstrated a remote, wireless attack on an implantable insulin pump from the firm Medtronic.

- Systems are getting more complex
- Software failures can be life-threatening
- Software failures now include security breaches

• Silicon without software is just sand...

• What is a virtual platform?
  – Building a virtual platform
  – Requirements for a virtual platform development environment

• Case studies for virtual platform based software development

• Summary, Q&A
Current Methodology, Software Debug on Prototype:

Run gdbserver on target and Eclipse on host to debug application on target.
Using a Virtual Platform Provides Exactly the Same Environment (with many of the same limitations)
Building the Virtual Platform

- The virtual platform is a set of models that reflects the hardware on which the software will execute
  - Subset / subsystem of a single device
  - Processor chip
  - Board
  - System
- Models are typically written in C or SystemC
- Models for individual components – interrupt controller, UART, ethernet – are connected just like in the hardware
- Peripheral components can be connected to the real world by using the host workstation resources: keyboard, mouse, screen, ethernet, USB, ...
- Models can be cycle accurate, cycle approximate, or instruction accurate, with instruction accurate models providing the highest simulation performance
To get the high speed required for real usage, processor hardware is modeled only to the minimum necessary level for correct or plausible instruction behavior so that software cannot tell it is not running on real hardware. Other features are approximated or omitted. Some examples:

- **Accurately modeled**
  - Most instructions
  - Exceptions
  - Structures, such as TLBs, required to allow OS boot

- **Approximated**
  - Tick timers – one “tick” per instruction
  - Random number generators (can affect, for example, TLB replacement algorithms)

- **Omitted**
  - Instruction pipelines
  - Speculative execution
  - Write buffers
  - Caches (can be added; not modeled by default)

- General rule – if a feature cannot be modeled with reasonable accuracy, don’t model it at all (no bogus pretence of accuracy)
Open Virtual Platforms Provides the Modeling Infrastructure

- Website community/portal/forum
- Over 7,000 people registered on the website

- Modeling APIs for processor, peripheral, and platform modeling

- Open source library of models (many are Apache 2.0 open source license)
  - Fast Processor Models (100+ by end 2012): ARM, MIPS, Renesas, ...
  - Peripheral models: UART, timer, interrupt, ethernet, DMA, I/O, ...
  - Working platforms: Linux, Nucleus, μC/OS II, bare metal applications, ...
  - OVP and SystemC/TLM2.0 native interfaces for all models
Website lists all available models (37 ARM + 17 ARM arch types)
For each specific model is a ‘variant’ page listing all relevant content: downloads, videos, presentations, documents
Open Virtual Platforms Provides a very fast simulator

- OVPsim™ simulator (models need the simulator to execute)
  - Runs processor models fast, 100s of mips
  - Interfaces to GDB via RSP
  - Encapsulation in Eclipse IDE for software and platform debug
Virtual Platform Requirements

- Performance near real time
- Run target binaries without change
- Repeatable results
- Multi-processor debug capability
- Software verification, analysis, profiling tools

![ARM Virtual Platform diagram]

- ARM Virtual Platform
  - Cortex-A15
  - Cortex-A15
  - GIC
  - GIC
  - Cortex-A7
  - Cortex-A7
- Memory
- Other Devices
- UART
- irq
- Keyboard
Virtual Platform Requirements Checklist

- Performance near real time
  - Instruction accurate virtual platforms run at 100s of MIPS

- Run target binaries without change
  - Use the same tool chain for compiling as for the real hardware

- Repeatable results
  - Simulation is a deterministic process, with repeatable results

- Multi-processor debug capability
  - Whether multiple processors on one device or board or system
  - Available either from virtual platform tool vendor or tool chain (IDE) vendor

- Software verification, analysis and profiling tools
  - Tools are needed so the virtual platforms can deliver on the simulation promise of complete controllability, visibility
Virtual Platforms Simulate the Software Running on the Hardware

Application Software & Operating System

Virtual Platform simulation engine

results(WH) = results(VP)
Software Analysis on Hardware (OS tracing, event scheduler analysis, ...)

Application Software & Operating System

New binaries, e.g. elf files

results(HW) = results(HW + instrumentation)

Add instrumentation, debug kernel, ...
Software Analysis on Virtual Platform can be Non-Intrusive

(code coverage, profiling, tracing, memory analysis, ...)

\[ \text{results(HW)} = \text{results(VP + instrumentation)} \]
Virtual Platform with Verification, Analysis and Profiling (VAP) Tools Plus Debugger

Imperas M*SIM simulation engine

Application Software & Operating System

Virtual Platform

Peripheral
Memory

OVP CPU

CPU HELPER

OS HELPER

VAP TOOLS

MULTI-PROCESSOR/MULTI-CORE DEBUGGER

Output Data
Trace Profile Coverage Schedule ...

MULTICORE DESIGN SIMPLIFIED
Requirements for Verification, Analysis, and Profiling Tools

- Non-intrusive: no modification of application source code
- Minimal overhead: simulations should still run fast
- Modular: can run one or more without tools stepping on each other
- Flexible: interactive or scripted use models
- Configurable: adjust for specific platform and focus
- Distributable: need to be shipped with virtual platform as integral part of SDK for specific platform/chip
Agenda

• Silicon without software is just sand...
• What is a virtual platform?
  • Case studies for virtual platform based software development
    – SMP Linux / Android
    – OS-related software: loadable kernel modules (LKMs) for Linux
    – OS exception analysis
    – AMP system
• Summary, Q&A
Agenda

• Silicon without software is just sand...
• What is a virtual platform?
• Case studies for virtual platform based software development
  – SMP Linux / Android
  – OS-related software: loadable kernel modules (LKM) for Linux
    – OS exception analysis
    – AMP system
• Summary, Q&A
Example 1: SMP Linux / Android on Dual Core ARM Cortex-A9

- **Goal:** in depth understanding of SMP Linux / Android operation on dual core processor
- **Virtual platform:** ARM Versatile Express
- **Processor model:** OVP ARM Cortex-A9 MPx2
- **Imperas VAP tools:**
  - OS task tracing: start OS analysis and debug at higher level of abstraction than with instruction tracing
    - Requires OS aware capability
  - OS scheduler profiling: process creation / deletion and context switching
    - Requires OS aware capability
This root FS contains most basic Linux utilities (implemented with busybox) and the Lynx web browser.

Kernel config is available through /proc/config.gz

Welcome to OVP simulation from Imperas

Log in as root with no password.
Imperas login:
ARM Versatile Express
Booting Android
OS-Aware Software Analysis

• Non-intrusive trace of
  – process creation
  – context switch
  – process deletion

• Captures communications between processes
OS-Aware Software Analysis Example: OS Task Tracing

- Non-intrusive: no instrumentation or modification of source code
- Multicore capable

1) Enables in-depth monitoring and analysis, even before console is available

2) Provides tracing at different levels of abstraction, granularity
   - ~ 700,000,000 instructions to boot Linux
   - ~ 700 tasks to boot Linux; i.e. Task Trace provides better ability to debug OS problems, such as bring up...

Can be done for any OS
- Linux
- Nucleus
- μC/OS
- μItron
- FreeRTOS
- ...
- Proprietary
Case 1: OS-Aware summary

- Modern complex Operating Systems/RTOS runs millions, billions of instructions before ‘interesting’ things happen
- Yes you need to simulate to see what is going on
- BUT – you need more than just instruction trace
- You need advanced OS-aware technologies
  - eg: task trace, scheduler trace, task profile
- AND they must be non-intrusive
  - Especially for multi-core, multi-processor
Agenda

• Silicon without software is just sand...
• What is a virtual platform?
• Example case studies for virtual platform based software development
  – SMP Linux / Android
  – OS-related software: loadable kernel modules (LKM) for Linux
    – OS exception analysis
    – AMP system
• Summary, Q&A
Example 2: Analysis & Debug of Loadable Kernel Module (LKM)

- LKMs are dynamic drivers used with Linux
  - Dynamic nature of the driver, plus complex interactions between cores, peripherals, OS and driver make analysis and debug difficult
  - What about timing-dependent bugs in driver with instruction accurate simulation?
    - For multicore devices, need to minimize timing dependencies in software
    - Instruction accurate virtual platforms are complementary to other development methodologies

- Platform: Cortex-A9MPx2
- Imperas Debug + VAP tools:
  - Multitprocessor debug enables simultaneous debug of software on peripherals as well as the processors
    - Set watchpoints, breakpoints on peripherals instead of processors
  - Functional coverage of test scenarios
  - Protocol verification
Abstraction, Operating System Application, Driver (LKM) and Peripheral

- Add hardware/peripheral to platform
- Needs driver adding to Linux

- LKM sits as part of the Kernel and provides services to the User Apps
- Need to verify interactions with Devices
  - AlphaNumeric Display peripheral added
- Debugging
  - Device driver and Peripheral model

- Challenges related to
  - Different components (CPU+peripheral)
  - Timing – sequencing of events
  - Repeatability
Imperas MP Debugger provides easy development

- Integrated debugger with simulator
  - Full control of running/stoping cores/processors
- Debug of both hardware and software - together
- Spatial, temporal, and abstraction
- Full reproducibility and determinism

- Can see everything in the platform
  - All hardware components
  - All software on all processors
  - [normal software debuggers only ever see the processor and its software – no access to platform components, behavioral subsystems]
  - Be able to single step/control any code – hardware model or software app
- Powerful MP features
  - e.g. place watchpoint on memory – triggers whenever any processor or component accesses
  - Global state, access, views, control, scripting...
  - Control any processor – in SMP cluster or hetero AMP system
Case 2 summary: using complex OSes requires sophisticated tools

- Imperas MP Debugger provides simultaneous access to hardware and software with OS-awareness
- Sorts out issues with dynamic symbol loading
- Allows very efficient development of software that interacts with hardware
  - Even on top of complex OS
Agenda

• Silicon without software is just sand...
• What is a virtual platform?
• Example case studies for virtual platform based software development
  – SMP Linux / Android
  – OS-related software: loadable kernel modules (LKMs) for Linux
    – OS exception analysis
    – AMP system
• Summary, Q&A
Example 3: In Depth OS Behavior Analysis

• **Goal:** Use virtual platform visibility to measure the number of instructions from exception entry to return

• **Custom tool developed for analyzing exception handler instruction counts**
  
  – Utilizes Imperas VAP Tools infrastructure
    
    • Interested in callbacks on exceptions and their returns
      
      – VapHelper provides callbacks on entry and return from exception
      – CpuHelper detects and provides details of exceptions

  – Adds new command to simulation environment to turn on/off tracing

  – Simply reports entries and returns with elapsed instruction counts

  – Could easily be enhanced to provide statistical analysis, report worst case occurrences, provide call stack snapshot at exception, provide OS process information, etc.
Imperas Simulation Infrastructure
Enables Tool Definition

Application Software & Operating System

Virtual Platform simulation engine

binaries, e.g. elf files

OVP CPU

results
Imperas Simulation Infrastructure Enables Tool Definition

Application Software & Operating System

binaries, e.g. elf files

Virtual Platform simulation engine

OVP CPU

results

Simulation Infrastructure

OVP Fast Processor Model:
CPU functionality, predefined views, events, actions

Simulation Engine:
Just In Time (JIT) code morphing (binary translation)
Imperas Simulation Infrastructure Enables Tool Definition

Application Software & Operating System

Virtual Platform simulation engine
- OVP CPU
- instrumentation

results

Simulation Infrastructure
- CPU and OS Helpers:
  CPU and OS specific information
- OVP Fast Processor Model:
  CPU functionality, predefined views, events, actions
- Simulation Engine:
  Just In Time (JIT) code morphing (binary translation)
Imperas Simulation Infrastructure Enables Tool Definition

Application Software & Operating System

Virtual Platform simulation engine
- OVP CPU instrumentation

Tool Helper:
API enabling user-definition of software analysis tools

CPU and OS Helpers:
CPU and OS specific information

OVP Fast Processor Model:
CPU functionality, predefined views, events, actions

Simulation Engine:
Just In Time (JIT) code morphing (binary translation)

Results
Imperas Simulation Infrastructure Enables Tool Definition

Application Software & Operating System

binaries, e.g. elf files

VAP Tool:
Definition of the tool, written in C (from library or can be user written)

Tool Helper:
API enabling user-definition of software analysis tools

CPU and OS Helpers:
CPU and OS specific information

OVP Fast Processor Model:
CPU functionality, predefined views, events, actions

Simulation Engine:
Just In Time (JIT) code morphing (binary translation)
Exception Analysis Tool

CPU Helper
• When “exception” event occurs:
  • Determines all the addresses this exception might return to
  • Produces a description string for the event

Tool Helper
• When notified of an “exception” event
  • Determines and saves the current context of the processor
  • Registers intercepts on all possible return addresses
• When exception return address is intercepted
  • Determines if context matches a previously observed exception

Exception Analysis Tool
• Adds a user command to enable/disable exception tracing
• When notified of an exception entry
  • Creates data structure, including instruction count on entry
• When notified of an exception exit
  • Determines elapsed instructions since entry
• Provides report of data collected about the exception event
Exception Analysis Tool: Results

- Platform is booting Linux
- Exception analysis tool is used interactively as OS is running
  - Could be used in script
- Reports where exception was taken and returned
- Calculates instructions between exception entry and return
Case 3: Exception Analysis

• With complex operating systems hard to gain visibility
  – Not just for exceptions, but for all operations
• Requires advanced tools with abstractions
  – CPU-aware, OS-aware
• Cannot expect tool vendor to know all types of analysis that is appropriate for your system
• Need ability for user created extensions that are
  – Easy, well documented, efficient, abstraction, fast
  – And of course – non-intrusive
  – And with no requirement for app recompile
Agenda

- Silicon without software is just sand...
- What is a virtual platform?
- Example case studies for virtual platform based software development
  - SMP Linux / Android
  - OS-related software: loadable kernel modules (LKM)s for Linux
  - OS exception analysis
  - AMP system
- Summary, Q&A
Example 4: AMP System Analysis

- Goal: in depth understanding of AMP operation on heterogeneous multiprocessor system
- Zynq virtual platform with dual core ARM Cortex-A9 processor plus Xilinx MicroBlaze processor
Heterogeneous Platform
ARM Cortex-A9MPCore + Xilinx MicroBlaze

Keyboard / Mouse

Xilinx MicroBlaze

ARM Cortex-A9MPCore

UART
Memory
Peripheral

Bridge

SysControl
Sysregs
Timer SP804
RTC PL031
USB
DDR
SRAM
RAM
Flash

LCD Controller PL011
UART PL111
Keyboard/Mouse PL050

LAN9118
PL041

USB
RAM
Flash

RTCP
dr
dr
dr

ARM Cortex-A9MPCore

Keyboard
Mouse
Xilinx Zynq™-7000 EPP System Level Block Diagram

✓ Cadence virtual platform includes Imperas OVP Fast Processor Model of ARM Cortex-A9MPx2
Imperas VAP Tools

• OVP Fast Processor Models enable use of VAP tools
• CPU and OS aware
  – Almost 100+ CPU cores supported
  – OS support: Linux, Nucleus, uCLinux, FreeRTOS, µC/OS II, eCos, µLtron, proprietary, ...
  – Used for hardware-dependent software development
    • Early software development
    • Software testing
    • System analysis
• 25+ M*VAP tools: code coverage, profiling (function, OS events), tracing (instruction, function, event, OS task, OS kernel), memory analysis, ...
• Non-intrusive
  – No instrumentation or modification of application code
  – No change to instruction ordering
• Execute as native host code for minimal overhead
• Can be used interactively or scripted
• Multiple tools can be loaded simultaneously
• User defined tools enabled: fault injection, protocol verification, software behavior analysis, ...
  – Users write tools in C
  – Documented API
Agenda

- Silicon without software is just sand...
- What is a virtual platform?
- Example case studies for virtual platform based software development
  - SMP Linux / Android
  - OS-related software: loadable kernel modules (LKMs) for Linux
    - OS exception analysis
    - AMP system
- Summary, Q&A
Virtual Platform Based Software Development

- Simulation (virtual platforms) enables full visibility, controllability of software
- Tools are needed – more than just simulation – to deliver on the promise of visibility, controllability
- Verification, analysis and profiling tools for virtual platforms provide complementary capability to existing development methodology

Take away: using Virtual Platforms with advanced tools enhances software development in terms of quality, timescales, efforts, and results