Agenda

- New challenges posed by heterogeneous architectures
- Comparison of hardware-based and Virtual Platform-based methodologies
- Continuous Integration and Virtual Platforms
- Case study: Linux bring up and testing on Altera Cyclone V SoC FPGA
- Demonstration
Observations On Embedded Software

1) As software complexity is increasing exponentially, companies need to adopt better ways to address problems, as *eventually the existing methods will no longer be sufficient*.

2) *One serious failure changes everything*.

3) There is a lesson to be learned from SoC design and verification: *a structured methodology provides predictable execution and measurable reduction of risk*.

⇒ Embedded software development domain needs to adopt a more formalized approach.
Agenda

- New challenges posed by heterogeneous architectures
- Comparison of hardware-based and Virtual Platform-based methodologies
- Continuous Integration and Virtual Platforms
- Case study: Linux bring up and testing on Altera Cyclone V SoC FPGA
- Demonstration
Heterogeneous SoC Architectures

- Heterogeneous can have various meanings for SoCs
  - Multiple different processors
  - Multiple computing elements, such as CPU plus GPU
- Why heterogeneous architectures?
  - Optimize the resources on the SoC for different tasks, e.g. application processor plus “minion” processors for power management, communication management, etc.

Mobileye EyeQ4 has quad core (8 threads) MIPS interAptiv plus MIPS M5150
Historically there was not a common operating system across the processors.

Recently processor IP companies have developed processor configurations that are similar enough to allow a common Linux OS to run.

Originally this was ARM big.LITTLE, with quad core Cortex-A15 plus quad core Cortex-A7, for power optimization of the application processor.

The different processors did not run simultaneously; the operating system switched automatically between the quad core processors depending on application load.

ARM big.LITTLE Operating Modes

Cluster Migration

2

1 1 1 1

2

2

CPU Migration

1 1

2 2

3 3

4 4

Global Task Scheduling

5

1

2

3

4

6

- OS can see one of two processor clusters.
- Only one cluster can be active at a time.
- Tasks run on either the LITTLE CPU cluster or the big CPU cluster.

- OS scheduler can see four CPU pairs, but only one CPU in each can be active at a time.
- Tasks run on either the big CPU or the paired LITTLE CPU.

- OS scheduler can see all CPUs and all can be active at any time.
- Tasks can run on or move between the LITTLE CPUs and the big CPUs as defined by the scheduler.

Source: ARM
MIPS I6500 and ARM DynamIQ Extend Heterogeneous Compute Paradigm

- Fully configurable architectures now enabled
  - Multiple heterogeneous cores per cluster
  - Multiple heterogeneous clusters per SoC
  - Linux and architecture now supporting > 1000 computing elements

Source: IMG
Linux Complexity Increases With Full Heterogeneity

- During boot, Linux needs to probe each core to determine characteristics so that correct hardware routines are installed
- Some Linux modules need to be updated
  - Cache initialization/handling: previously assumed homogeneous cache size, now needs to accommodate potentially different cache sizes for different cores/clusters
  - ...
- With increased complexity comes increased porting and bring up issues
Agenda

- New challenges posed by heterogeneous architectures
- Comparison of hardware-based and Virtual Platform-based methodologies
- Continuous Integration and Virtual Platforms
- Case study: Linux bring up and testing on Altera Cyclone V SoC FPGA
- Demonstration
Hardware-Based Software Development

- Has timing/cycle accuracy
- JTAG-based debug, trace
- Traditional breadboard / emulation based testing
  - Limited physical system availability
  - Limited external test access (controllability)
  - Limited internal visibility
- To get around these limitations, software is modified
  - printf
  - Debug versions of OS kernels
  - Instrumentation for specific analytical tools, e.g. code coverage, profiling
  - Modified software may not have the same behavior as clean source code
Virtual Platform Based Software Development

- Instruction accurate simulation
  - Runs actual hardware executables
    - e.g. MIPS code on x86 PC
  - Models require only functionality that is needed for software
  - Fast, enables quick turnaround and comprehensive testing
  - Simulation-based development provides visibility, controllability not available from hardware
  - Same or better debugger access than hardware
  - Access for the entire team
Advantages of Virtual Platform Based Software Development

- Earlier system availability
- Full controllability of platform both from external ports and internal nodes
  - Can corner cases be tested?
  - Can an error be made to happen?
- Full visibility into platform: if an error occurs, will it be observed by the test environment?
- Easy to replicate platform and test environment to support regression testing on compute farms
Virtual Platforms Complement Hardware-Based Software Development

- Current test methodology employs testing on hardware
  - Proven methodology
  - Has limitations
  - We are at the breaking point

- Virtual platform based methodology promises controllability, visibility, repeatability

- Virtual platforms – software simulation – provide a complementary technology to the current methodology
Building the Virtual Platform

The virtual platform is a set of models that reflects the hardware on which the software will execute:

- Could be 1 SoC, multiple SoCs, board, system; no physical limitations
- Functionally accurate, such that the software does not know that it is not running on the hardware, i.e. runs the target binaries - unmodified

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, …
Agenda

- New challenges posed by heterogeneous architectures
- Comparison of hardware-based and Virtual Platform-based methodologies
  - Continuous Integration and Virtual Platforms
- Case study: Linux bring up and testing on Altera Cyclone V SoC FPGA
- Demonstration
Modern Development Methodology: Agile, Not V-Shaped

CONTINUOUS INTEGRATION & TEST

- Compile
- Build
- Test

Developer:
- Code & Tests

Tester:
- Packaging
- Stress Test
- Full Application
- QA Test

Release / Deploy

Errors
- Fail
- Pass

Repository:
- Trigger
- Commit
Adopting Continuous Integration & Continuous Test for Embedded requires Simulation

- Imagine a software build system without access to ‘make’ or ‘ant’
  - they enable effective build automation

- Simulation enables the effective automation of testing embedded systems as part of a Continuous Integration / Continuous Test (CI/CT) environment

- Simulation enables full automation
  - with no manual intervention

- Use of hardware is just too hard

=> Virtual Platforms (simulation) enable CI / CT for embedded
Simulation is a key component of embedded CI / CT environment

CONTINUOUS INTEGRATION & CONTINUOUS TEST

Compile Build Test

Repository

Developer

Code & Tests

Tester

Packaging Stress Test Full Application QA Test

Release / Deploy

Virtual Platform Simulation

Errors

Commit

Trigger
Motivation for Change: Benefits of Continuous Integration

- Better code structure and quality
  - Frequent code check-in pushes developers to create modular, less complex code
  - Enforces discipline of frequent automated testing
  - Software metrics generated from automated testing and CI (such as metrics for code coverage, code complexity, and feature completeness) focus developers on developing functional, quality code, and help develop momentum in a team

- Easier debug
  - When unit tests fail or a bug emerges, if developers need to revert the codebase to a bug-free state only a small number of changes are lost

- Fewer major integration bugs
  - Immediate feedback on system-wide impact of local changes
  - Integration bugs are detected early and are easy to track down due to small change sets. This saves both time and money over the lifespan of a project.
  - Avoids last-minute chaos at release dates, when everyone tries to check in their slightly incompatible versions

- Constant availability of a "current" build for testing, demo, or release purposes
Agenda

- New challenges posed by heterogeneous architectures
- Comparison of hardware-based and Virtual Platform-based methodologies
- Continuous Integration and Virtual Platforms

- Case study: Linux bring up and testing on Altera Cyclone V SoC FPGA
- Demonstration
Altera Cyclone V SoC FPGA with ARM Cortex-A9MPx2

- Green background peripheral models fully functional
- Green diagonal pattern background have only the functionality necessary to boot the operating systems
- Blue background peripheral models just stubs

---

```
rtc-p303: rtc core: registered p303 as rtc0
mci-p100: ahb-mci: mci0: PLISS wafl 43 revd at 0x80000000 irq 41.42 (pio)
usbh: registered new interface driver usbhid
usbhid: USB HID core driver
AHB device list:
    No soundcards found
    omap: using am/arm7-c0
    IEP cubic registered
    IM: Registered protocol family 17
rtc-p303: rtc: rtc clock to 1570801418000000000 UTC 181
Freeing last memory 1789
input: AT7305 Keyboard as /dev/input/event5
input: dac7502 Generic Graphical Mouse as /dev/input/event3
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 QVP simulation from Imperas
Log in as root with no password.
Imperas login
```

---

![Diagram of Altera Cyclone V SoC FPGA with ARM Cortex-A9MPx2](image-url)
Linux Bring Up and Testing on Altera Cyclone V SoC FPGA

1) Linux boot on single core ARM Cortex-A9 (minimal peripheral models)
2) SMP Linux boot on dual core ARM Cortex-A9 (minimal peripheral models)
3) Add in peripheral models for Cyclone V SoC FPGA

- Need to set up test infrastructure such that Continuous Integration (CI) testing can be performed
Cyclone V SoC FPGA Virtual Platform

- Top level virtual platform built using Open Virtual Platforms (OVP, www.OVPworld.org) platform API
- ARM Cortex-A9MPx2 processor core model from the OVP Library
- Peripheral models
  - Some models available in the OVP Library
  - Remaining models of peripheral components developed using OVP APIs
- OVP APIs written for C language
- Simulation engine: Imperas M*SDK

- All OVP processor and peripheral models include both native OVP and native SystemC/TLM2 interfaces, so all the following results could have been achieved using the OSCI SystemC simulator plus Imperas M*SDK product
  - Peripheral models could have been written in SystemC
  - M*SDK tools require OVP processor core models to enable tools
1a) Linux Boot on Single Core ARM Cortex-A9

- Use Linux from Altera: Altera-3.4
- Use default configurations
- Use default device trees
  - Comment out a few peripherals not yet modeled
- Bug found in Linux kernel preemptive scheduling
  - Running multiple applications under Linux part of standard Imperas bring up testing
  - Linux boots and runs, but does not switch tasks properly

- Approximately 2 weeks engineering effort to build virtual platform able to boot Linux
- Virtual platform boots Linux in under 5 sec on standard PC, Windows or Linux
1b) OS-Aware Tools Used to Find the Bug

- Use OS tracing [task, execve, schedule, context, …] to trace at the OS level, not instruction level
- OS-aware tools debug in hours, once the bug was observed
- Simulation overhead due to OS-aware tools < 10%

![Imperas Multicore Debugger Screenshot]
OS-Aware Software Analysis
Example: OS Task Tracing

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

1) OS-aware tools enable in-depth monitoring and analysis, even before console is available

2) Provides tracing at appropriate levels of abstraction, granularity
   - ~ 1,000,000,000 instructions to boot SMP Linux: instruction tracing to find OS problem would be painfully slow and complicated
   - ~ 700 tasks to boot Linux: task tracing provides better starting point for debugging OS problems during bring up
OS-Aware Software Analysis

Example: OS Scheduler Tracing

- Process creation
- Process deletion
- Context switching
- Non-intrusive
- Multicore capable
2a) SMP Linux Boot on Dual Core ARM Cortex-A9

- Use Linux from Altera: Altera-3.6
- Use default configurations
- Use default device trees
  - Comment out the peripherals not yet modeled

- Bug found in Linux accesses of GIC registers
- Virtual platform debug took 2 days versus 2 weeks on hardware platform (5x improvement)

- Also need to ensure that operating systems do not access forbidden memory segments
Callbacks on events

- Non-intrusive trace/callback of
  - Selected changes/events in the hardware of system
  - Selected events in OS/software
- Add C code to monitor and check what has happened – add protocols, rules, assertions
2b) Custom Memory Access Monitor Accelerates Platform Debug

- Memory access monitor is just C code, less than 350 lines, loaded into simulation environment
- When simulation is run, monitor produces warning if memory access rules are violated

```c
// Define watch areas for memory and peripherals defined in the platform

memWatchT amcWatch[] = {
    { "Linux memory", 0x0, 0x2fffffff, LINUX_CPU },
    { "gmac0", 0xff700000, 0xff700fff, LINUX_CPU },
    { "emac0_dma", 0xff701000, 0xff701fff, LINUX_CPU },
    { "gmac1", 0xff702000, 0xff702fff, LINUX_CPU },
    { "emac1_dma", 0xff703000, 0xff703fff, LINUX_CPU },
    { "uart0", 0xffc02000, 0xffc02fff, LINUX_CPU },
    { "CLKMGR", 0xffd04000, 0xffd04fff, LINUX_CPU },
    { "RSTMGR", 0xffd05000, 0xffd05fff, LINUX_CPU },
    { "SYSMGR", 0xffd08000, 0xffd08fff, LINUX_CPU },
    { "GIC", 0xfffec000, 0xfffedfff, LINUX_CPU },
    { "L2", 0xfffff000, 0xffffeffff, LINUX_CPU },
    { 0 } /* Marks end of list */
};
```

Warning (AMPCHK_MWV) LINUX_CPU: AMP write access violation in uart1 area. PA: 0xfffff000 VA: 0xfffff000
Warning (AMPCHK_MWV) LINUX_CPU: AMP write access violation in uart1 area. PA: 0xfffff00c VA: 0xfffff00c
Warning (AMPCHK_MWV) LINUX_CPU: AMP write access violation in uart1 area. PA: 0xfffff010 VA: 0xfffff010
2b-2) Custom Memory Access Monitor Accelerates Platform Debug (2nd CPU)

- Memory access monitor is just C code, less than 350 lines, loaded into simulation environment
- When simulation is run, monitor produces warning if memory access rules are violated

```c
// Define watch areas for memory and peripherals defined in the platform
//
memWatchT amcWatch[] = {
  // name            watchLow watchHigh   allowedCPUs
  { "Linux memory", 0, 0x2fffffff, LINUX_CPU },
  { "CPU2 memory", 0x30000000, 0x31ffffff, CPU2_CPU },
  { "gmac0", 0xff700000, 0xff700fff, LINUX_CPU },
  { "emac0_dma", 0xff701000, 0xff701fff, LINUX_CPU },
  { "gmac1", 0xff702000, 0xff702fff, LINUX_CPU },
  { "emac1_dma", 0xff703000, 0xff703fff, LINUX_CPU },
  { "uart0", 0xffc02000, 0xffc02fff, LINUX_CPU },
  { "uart1", 0xffc03000, 0xffc03fff, CPU2_CPU },
  { "CLKMGR", 0xffd04000, 0xffd04fff, LINUX_CPU },
  { "RSTMGR", 0xffd05000, 0xffd05fff, LINUX_CPU },
  { "SYSMGR", 0xffd08000, 0xffd08fff, LINUX_CPU },
  { "GIC", 0xfffec000, 0xfffedfff, LINUX_CPU },
  { "L2", 0xfffef000, 0xfffeffff, LINUX_CPU },
};
```

Warning (AMPCHK_MWV) LINUX_CPU: AMP write access violation in uart1 area. PA: 0xffc03008 VA: 0xffc03008
Warning (AMPCHK_MWV) LINUX_CPU: AMP write access violation in uart1 area. PA: 0xffc0300c VA: 0xffc0300c
Warning (AMPCHK_MWV) LINUX_CPU: AMP write access violation in uart1 area. PA: 0xffc03010 VA: 0xffc03010
Warning (AMPCHK_MRV) CPU2_CPU: AMP read access violation in Linux memory area. PA: 0x00000020 VA: 0x00000020
Agenda

- New challenges posed by heterogeneous architectures
- Comparison of hardware-based and Virtual Platform-based methodologies
- Continuous Integration and Virtual Platforms
- Case study: Linux bring up and testing on Altera Cyclone V SoC FPGA
- Demonstration
Demonstrations

- Linux boot on single ARM Cortex-A9
- Linux boot on Altera Cyclone V
- Linux boot on multicore MIPS I6400
- SMP Linux boot on single core ARM Cortex-A9
  - OS-aware tools
- Memory Monitoring
Linux boot on single ARM Cortex-A9

ARM Versatile Express Cortex-A9MP / SMP Linux

Keyboard / Mouse

Run
Linux boot on Altera Cyclone V

Altera Cyclone V SoC FPGA
ARM Cortex-A9MPx2
Linux OS-Aware tools

Imperas ARM platform
ARM Versatile Express Cortex-A9MP

Run
PrintKernel
TraceTasks
Memory Monitor

- Non-intrusive trace/callback of
  - Selected changes/events in the hardware of system
  - Selected events in OS/software

![Diagram showing ARM CPU, CPU-Aware Layer, Runtime Analysis/checker, and Report]

Run
Virtual Platform Adoption*

- As systems become more complex, organizations are turning to modeling and simulation tools to improve their software development environments.
- Virtual platform solutions are being adopted as a mechanism to improve system quality and to accelerate software development and testing.
- Engineering teams whose projects align with Agile and Continuous Integration (CI) product development methodologies are more likely to use virtual platform solutions.

⇒ If you need to build complex products with high quality in shorter schedules you need to adopt virtual platform based solutions.

* Trends from VDCresearch reports 2014
Summary

- Virtual platforms – software simulation – provide a complementary technology to hardware-based testing of software

- Linux bring up on virtual platforms should be done incrementally
  - Minimizing platform degrees of freedom adds productivity

- OS-aware tools provide additional productivity, efficiency

- Custom tools provide more robust software test environment
Thank you

www.imperas.com
www.OVPworld.org