Modern Software Development Methodology for RISC-V Devices

Larry Lapides
Imperas Software Ltd
The size of the RISC-V market share will depend more on the software ecosystem than on specifics of RISC-V implementations.
Agenda

- Challenges in embedded software development
- New methodology is needed
  - Continuous Integration
  - Continuous Test
- Test with hardware – necessary evil? – help or hindrance?
- Adoption of Continuous Test for embedded
  - And how simulation is used
- Worked example
- Summary
Embedded Software Development Challenges

- Schedule
- Quality
- Security
- Safety certifications
- Predictability of the software engineering task: management accuracy on software resource and schedule requirements is +/- 50%
- Unknown / unmeasurable delivery risk

Jeep hacked in 2015
Embedded Software Methodology is Evolving

- Modern software development moving away from traditional V-shaped development flow to …
  - Agile
  - Continuous Integration
  - DevOps
Modern Development Methodology: Agile, Not V-Shaped

CONTINUOUS INTEGRATION

Developer

commit

repository

trigger

Compile

Build

Test

pass

fail

errors

errors

Tester

Packaging
Stress Test
Full Application
QA Test

Release / Deploy

Modern Development Methodology: Agile, Not V-Shaped
The Challenge…

- How to apply this methodology and gain these benefits in the Embedded Software world…

- In this talk the focus is only on the Continuous Integration piece of the Agile methodology
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
First, some of the problems

- Multiple code streams (release versions) to manage
  - Development, under test, in field
- Many hardware/OS targets: processor variants (ARM, MIPS, Renesas), OS versions (Linux xyz, 32/64, Windows 7/10, 32/64) and a large amount of common code between targets
- With many teams and tasks all in parallel

- Access/configuration of available hardware
  - (e.g. customer USAF 1 prototype, 2 weeks to get access, shift work)
  - (recall: old computers, card decks, or early timeshare 30 mins per day)
- Not just about testing something works
  - ensure what you think is being tested is being tested, i.e. need metrics
- And then, need to run 1,000s of tests on many targets to validate software changes

- And with many common libraries, any change proliferates to many projects
  - need to re-validate ALL projects
OK – so automation can address this

- Continuous Integration (CI)
  - Create a build server so that any change builds software
    - for multiple code streams
    - for multiple targets

- Then require Continuous Test (CT)
  - For each build for each target run N test cases
    - Quantify correctness
    - Coverage
    - Performance
Continuous Integration
Continuous Test (CICT)
Now, how to test?

- Use x86 PC native?
  - e.g. x86 compile and run – works well for simple code
  - What about binary libraries, e.g. for ARM CMSIS
  - What about CPU architectures with restrictions e.g. reduced address space, available memory, ...

- For embedded, real target code should be used
  - Cross compile
    - Use correct binary libraries
    - Use correct instruction streams
  - Need to run
    - on real cpu architecture
    - with real data
    - with real stimulus
  - Need to capture real outputs
But using real hardware is a problem

- Need Device Under Test and environment (world) both available in hardware
  - How to stimulate the environment, sensors, buttons
  - How to monitor responses and measure correctness in both value and time
- Hardware may require manual intervention, which is prone to errors
- Can a hardware testbench do everything you need?
  - For example, trigger interrupt after 15msecs of xyz event
- Can hardware be set into the correct state to start a test sequence?
- It can be hard to model the real world, and hard to make reproducible
And there are more issues

- How much access can you get for your testing
  - Including setup and versioning of the hardware
- And can you have several users using in parallel
- And prototypes are costly to acquire and maintain
- And they only run in real time
  - Can they run faster to get more testing done?
    - (e.g. customer Audi – 6 months of road data need to run tests overnight)
    - (e.g. GPS chip sends position every second)
- ...

(e.g. Vellonis)
Adopting Continuous Test for Embedded Needs 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 CICT environment

- Simulation enables full automation
  - with no manual intervention

- Use of hardware is just too hard

=> Virtual Platforms (simulation) enable CICT for embedded
Modern Software Methodology: Virtual Platforms Complement Hardware-Based Software Development

- Virtual platform based methodology delivers controllability, visibility, repeatability, automation, access
  - 75-90% of bugs are functional, and can be found using software simulation testing
- Testing of timing sensitive software, and final testing, still needs to be done on hardware

Virtual platforms – software simulation – provide a complementary technology to the current methodology
So what are we talking about here in terms of simulation

- An Instruction Set Simulator (ISS) or
- A Virtual Platform (Virtual Prototype) simulation
  - CPUs, memories, peripherals
  - Test components, stimulus generation
  - Models of the world/environment
  - Verification/validation tools
Instruction Set Simulator (ISS)

CPU model variant selection

Application <cross>.elf

ISS (cpu+memory)

Environment

Semihosted File I/O

Debugger

GUI
Virtual Platforms Provide a Simulation Environment Such That the Software Does Not Know That It Is Not Running On Hardware

- The virtual platform is a set of instruction accurate models that reflect the hardware on which the software will execute
  - Could be 1 SoC, multiple SoCs, board, system; no physical limitations
- Runs the executables compiled for the target hardware
- 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, …
- Typical performance is 200-500 million instructions per second
Simulation Architecture Can (Should) Include Tools

Software Verification, Analysis & Profiling (VAP) tools
• Trace
• Profile
• Coverage
• Schedule
• Memory monitor
• Protocol checker

• ...

M*SIM JIT sim engine

Multiprocessor / Multicore Debugger

Eclipse IDE
Simulation is a key component of embedded CICT environment.

CONTINUOUS INTEGRATION & CONTINUOUS TEST

Repository

Developer

Code & Tests

Trigger

Commit

Compile

Build

Test

Pass

Fail

Errors

Virtual Platform Simulation

Tester

Packaging

Stress Test

Full Application

QA Test

Release / Deploy

Errors
Demonstration

- Imperas ISS simulation used with Jenkins environment
  - Jenkins is a widely used, open source CI/CT environment for general software
- Edit, compile, local test, check in -&gt; triggers build
- Successful build -&gt; triggers testing
- Testing completion -&gt; triggers results
Simple example is jpeg encoder targeting 4 different target ISA (ARM, MIPS, Renesas, Altera) with 11 different algorithm arguments to test.
Items can use ‘make’

e.g. Build

```
#!/bin/bash
#
# Setup the environment
#
/etc/bashrc

setup /home/build/DailyBuild/HEAD/Linux/Opt/Imperas
DIR=$(pwd)
declare -i rc=0
RUN_NAME=$(JOB_NAME)_$(BUILD_NUMBER)
echo "Starting Shell in DIR=$(DIR) RUN_NAME=$RUN_NAME"

FDIR=/home/build/jenkins/workspace/Project/Build/ESN_GT_Project
pushd $(FDIR)
  make_build ARM
rc=$?
popd
exit $rc
```
Can use scripts, like bash
e.g. Test

```bash
# usage: unknown_program_name [switches] [inputfile]
# Switches (names may be abbreviated):
# -q quality N Compression quality (0,100): 5-95 is useful range
# -g grayscale Create monochrome PNG file
# -o optimize Optimize Huffman table (smaller file, but slow compression)
# -p progressive Create progressive JPEG file
# -t targa Input file is Targa format (usually not needed)
# Switches for advanced users:
# -d dct Allow integer DCT method (default)
# -d dct fast Use fast integer DCT (less accurate)
# -d dct float Use floating-point DCT method
# -s setrows N Set rows in file, in blocks of 6
# -m smooth N Smooth dithered input (N=100 is strength)
# -m maxmemory N Maximum memory to use (in kbytes)
# -o outfile name Specify name for output file
# -verb or -debug Limit debug output
# Switches for wizards:
# -baseline Force baseline quantization tables
# -qtable file Use quantization tables given in file
# -qtable N[...][...][...] Set component quantization tables
# -s sample H[V[...][...]] Set component sampling factors
# -c scans file Create multi-scan JPEG per script file

switches[0]="-d dct int"
switches[1]="-d dct fast"
switches[2]="-d dct float"
switches[3]="-g grayscale -d dct int"
switches[4]="-g grayscale -d dct fast"
switches[5]="-g grayscale -d dct float"
switches[6]="-m quality 10"
switches[7]="-m quality 20"
switches[8]="-m quality 30"
switches[9]="-m quality 50"
switches[10]="-m quality 100"

# Cortex M3 RHSEGQM M510
# --processor vendor imtc.oepworld.org --processorname nips32 --variant RS100
# for i in 1 2 3; do
  variant=`Cortex M3`
  if [ "$VARIANT" = "$variant" ] || [ $i -eq "$VARIANT" ]; then
    for op in (0,10): 0
      Image=Images[$variant][i][op].jpg
      ppm=$(switches[op])
      echo "No variant $image: $ppm"
      imagemagick --quiet --noassert - -processor vendor imtc.oepworld.org --processorname nips32 --variant "$variant" -parameter UCLA=-parameter endian=little
      -parameter processors 1
      --program output/ARM Cortex M3/jpeg.cif
      --arg "[$ppm]" --arg "false" --arg "$image"
      test/image.out.$i.$op.jpg
diff $image test/image.out.$i.$op.jpg
      r=$?
      ndc_info "$image" $r
  fi
done
```

RISC-V Workshop © 2017 Imperas Software Ltd.
Jenkins Pipeline
Create ‘Stages’ from items

- Note use of parallel & serial
- Can trigger start of run from code check-in or directly
Stages running (2)

- Can run tests in parallel on available resources (executors)
Final Stage is collate results

- Each test run records test results from its group
- Final task stage in pipeline collates
Can see results at end

- See how tests perform over each run
- Management get a dashboard for visibility of project status
Drill down to see failures
Demo Wrap up

- This showed simple example of developing and testing code for embedded targets using cross compilers to build and ISS to execute
- Used CICT system (Jenkins) to manage processes, data, and results
- Very simple to set up / manage
- Automates build/test – and can provide high level monitoring and results to developers
- Easily extends to full platforms using Virtual Platform simulations
  - e.g. testing applications under operating systems
Agenda

- Challenges in embedded software development
- New methodology is needed
  - Continuous Integration
  - Continuous Test
- Test with hardware – necessary evil? – help or hindrance?
- Adoption of Continuous Test for embedded
  - And how simulation is used
- Worked example
- A little more about Imperas solutions
- Summary
Open Virtual Platforms (OVP)

Library of 170+ High-Performance Processor Models

- **ARM®**: Models for ARMv4™, v5™, v6™, v7™ and v8™ architectures
  - Including MMU, MPU, TCM, Thumb™, Thumb-2™, Jazelle™, SIMD, VFPv3, NEON™, TrustZone®, hardware virtualization instructions, ...

- **MIPS®**: Models for microMIPS, MIPS32 and MIPS64 architectures
  - Verification, licensing, and distribution relationship
  - Including MMU, MPU, DSP, FPU, MT, MSA, VZ architecture subsets

- **Renesas**: Models for RH850, V850 architectures; 16 bit microcontroller cores
  - RH850G3, V850 ES, E1, E1F, E2; RL78, M16C cores

- **Synopsys (ARC)**: ARC6xx, ARC7xx, EM, EMS families

- **Altera Nios II, Xilinx Microblaze**

- **PowerPC**

- **RISC-V**

“OVP is addressing key issues in software development for embedded systems. By supporting the creation of virtual platforms, OVP is enabling early software development and helping expand the ARM user community.”

*Noel Hurley, VP Business Development, ARM*
Support for RISC-V

- Imperas is a member of RISC-V Foundation
- Platforms / OS Combinations available for free download from Open Virtual Platforms (OVP) website, www.OVPworld.org
  - Bare metal application examples – processor + memory
  - Extendable Platform Kits (EPKs)
    - Operating systems on implementations of actual hardware platforms
- Processor models available as open source downloads
  - RV32I, RV32G
  - RV64I, RV64G
  - Others under development (contact Imperas for schedule)
- Models can be used in C, C++, SystemC, and TLM2 platforms and with 3rd party simulators/environments
Advanced Modeling Infrastructure for Virtual Platform Creation

Open Virtual Platforms™ (OVP™) infrastructure and iGen model template generator

- Model Library
  Extensive (300+), comprehensive open source model collection

- OVP Modeling
  Easy-to-code modeling API

- Environment
  Third party interfaces to SystemC, GDB, etc

- Reference Simulator: OVPsim
  Useful (free) simulator for running models

OVPworld.org
Key Technology: Multicore Development, Debug & Test Tools

- Verification, Analysis & Profiling (VAP) software tools
  - Non-intrusive: no modification of software or model source code
  - Users can easily define custom tools
- 3Debug™ capability for debug of software on complex, heterogeneous platforms
- Tools at the appropriate and valuable levels of abstraction, granularity
  - Instruction tracing shows everything at lowest level of abstraction, no granularity
  - Function tracing and OS tracing show higher levels of abstraction
  - Instruction subset tracing, e.g. SIMD or hardware virtualization, show finer granularity
Exhaustive Testing Solution
When Failsafe Quality is Key

Audi (NIRA Dynamics)

- Application
  - Application that tests tire pressure using ABS data
  - Software runs on different processors (different ABS systems) such as ARM Cortex-M/R, Renesas, PowerPC
- Software test and analysis
  - Collect months of road test data for use as stimuli
  - Data ran in regression suite, 1,000s of tests nightly
  - Memory analysis ensures stack and heap behavior
- Imperas M*SDK and OVP Fast Processor Models
  - High performance critical for comprehensive testing
  - Multiple processor support (multiple ABS systems) key

“Imperas M*SDK helps us not only to find bugs in our code, but also in the compilers we use. We will not ship software without testing with Imperas tools.”

Peter Lindskog, Head of Development, NIRA Dynamics AB
Security is Critical

Nagravision

- Application
  - Devices that protect streaming video
  - Attach to smart tv or set top box
  - Build SoC, end user device and software

- Imperas use model
  - Virtual platform peripheral models are built by Nagravision (proprietary models) and Imperas (standard I/O, e.g. USB)
  - Use Imperas debugger for software debug and for driver-peripheral model software-hardware co-debug
  - Use VAP tools such as OS-aware tools, code coverage, memory analysis, …
  - High performance simulation is critical for Continuous Integration (CI) testing

“At NAGRA, we have adopted the Imperas virtual platform-based software development and test tools for our application and firmware teams. The simulation performance, and the tools for software analysis, have added significant value to our daily Agile Continuous Integration (CI) methodology. Our view is that software simulation is mandatory to reach metrics required for high quality secured products.”
1) Custom/proprietary processor modeling
   - Build an ISS for internal use and/or delivery to customers
   - This is a make/buy decision: Imperas customers achieve lower initial and ongoing costs, higher quality

2) Early (pre-silicon) software development
   - Accelerate software schedules by months over conventional methodologies

3) Software test
   - Simulation is a complementary tool to hardware based software development
   - Achieve higher quality; reduce risk; accelerate safety/security certifications

4) Software/system power and timing estimation
   - Power and timing are key embedded product characteristics
Agenda

- Challenges in embedded software development
- New methodology is needed
  - Continuous Integration
  - Continuous Test
- Test with hardware – necessary evil? – help or hindrance?
- Adoption of Continuous Test for embedded
  - And how simulation is used
- Worked example
- Summary
Imperas Users Benefit From Improved Software Quality, and Reduced Schedules & Cost

- Key technologies: 170+ processor model library, large peripheral model library, fastest simulator, advanced Verification, Analysis and Profiling (VAP) tools
- Solutions for embedded use cases: custom CPU, semiconductor vendors, embedded systems developers
- Experience with Continuous Integration and Continuous Test usage
Thank you

For more information:

- [www.imperas.com](http://www.imperas.com)
- [www.OVPworld.org](http://www.OVPworld.org)

Contact us: info@imperas.com