MULTICORE DESIGN SIMPLIFIED

imperas

Agile for Embedded Conference 2017

Virtual Platform-based simulation for testing of Embedded Software in Continuous Integration flows

Lee Moore, Duncan Graham, **Simon Davidmann**, Larry Lapides
Imperas Software Ltd

For more information: info@imperas.com

Agenda

MULTICORE DESIGN SIMPLIFIED imperas

- Challenges in embedded software development
- Automation 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

The move to automation for software development

- Agile
- Continuous Integration
- DevOps

Manifesto for Agile Software Development

Values:

- ***Individuals and Interactions*** *over processes and tools*
- ***Working Software*** *over comprehensive documentation*
- ***Customer Collaboration*** *over contract negotiation*
- ***Responding to Change*** *over following a plan*

Based on twelve principles:

- Customer satisfaction by early and continuous delivery of valuable software
- Welcome changing requirements, even in late development
- Working software is delivered frequently (weeks rather than months)
- Close, daily cooperation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Face-to-face conversation is the best form of communication (co-location)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Best architectures, requirements, and designs emerge from self-organizing teams
- Regularly, the team reflects on how to become more effective, and adjusts accordingly

From Wikipedia

Continuous Integration

MULTICORE DESIGN SIMPLIFIED
imperas

- The practice of frequently integrating one's new or changed code with the existing code repository
  - should occur frequently enough that no intervening window remains between commit and build, and such that no errors can arise without developers noticing them and correcting them immediately
- Normal practice is to trigger these builds by every commit to a repository, rather than a periodically scheduled build
- Uses a version control system that supports atomic commits, i.e. all of a developer's changes may be seen as a single commit operation

Relies on the following principles:

- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
- Keep the build fast
- Test in a clone of the production environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

From Wikipedia

DevOps

MULTICORE DESIGN SIMPLIFIED

Code

- Code development and review, version control tools, code merging;

Build

- Continuous integration tools, build status;

Test

- Test and results determine performance;

Package

- Artifact repository, application pre-deployment staging;

Release

- Change management, release approvals, release automation;

Configure

- Infrastructure configuration and management, Infrastructure as Code tools;

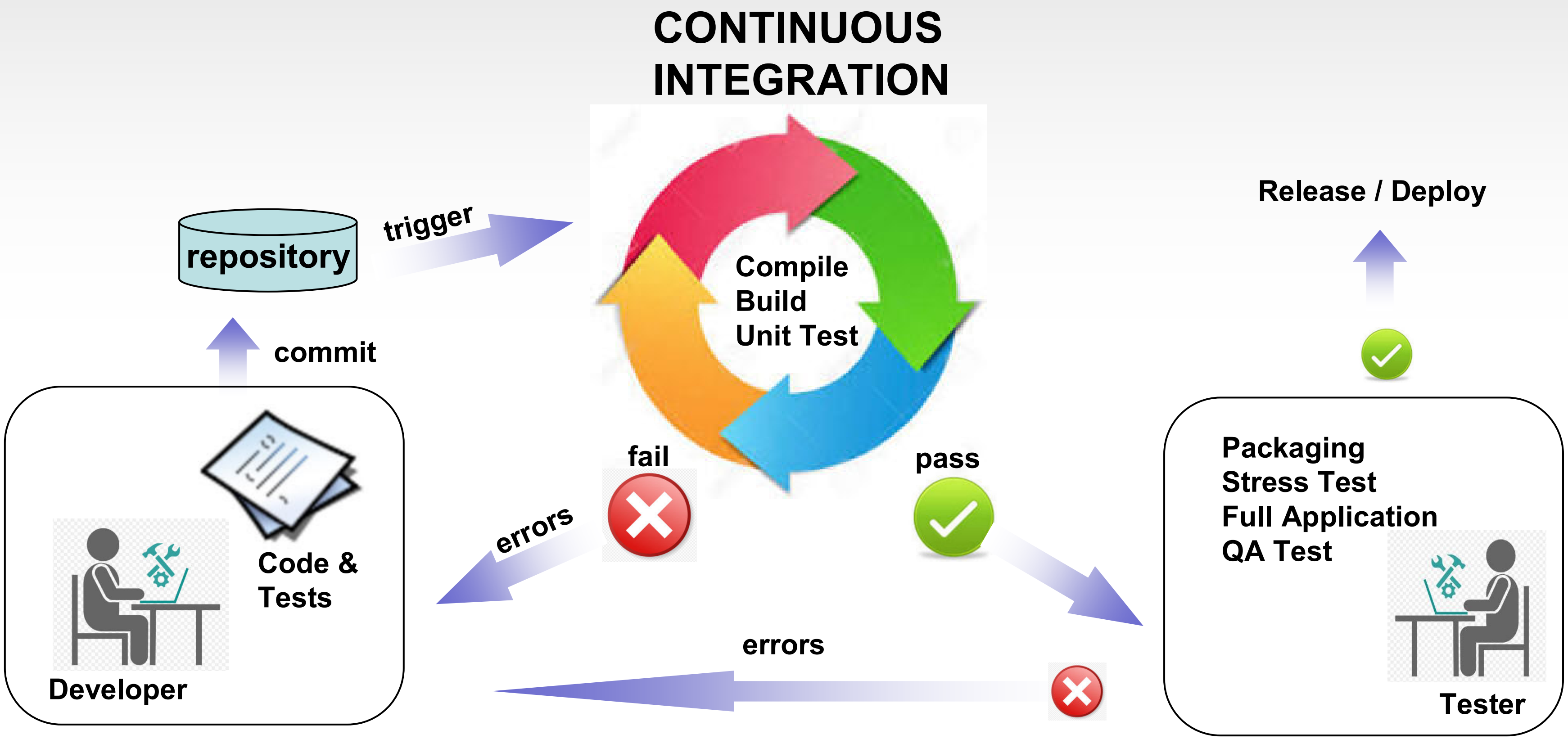
Monitor

- Applications performance monitoring, end–user experience.

From Wikipedia

Modern Development Methodology

MULTICORE DESIGN SIMPLIFIED
imperas



And the benefits… of Continuous Integration

MULTICORE DESIGN SIMPLIFIED
imperas

- 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
- When unit tests fail or a bug emerges, if developers need to revert the codebase to a bug-free state without debugging, only a small number of changes are lost (because integration happens frequently)
- Constant availability of a "current" build for testing, demo, or release purposes
- Frequent code check-in pushes developers to create modular, less complex code
- …

With continuous automated testing benefits can include:

- Enforces discipline of frequent automated testing
- Immediate feedback on system-wide impact of local changes
- 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
- …

And the challenge…

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

First, some of the problems

- Multiple code streams (release versions) to manage
  - Development, under test, in field
- With many hardware/OS targets, revisions (Linux xyz, 32/64, Windows 7/10, 32/64) and variants (ARM, MIPS, Power, …) 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, eg need coverage
- And then, need to run 1,000s of tests on many variants etc to validate software changes

- And with many common libraries, any change proliferates to many projects
  - need to re-validate ALL projects

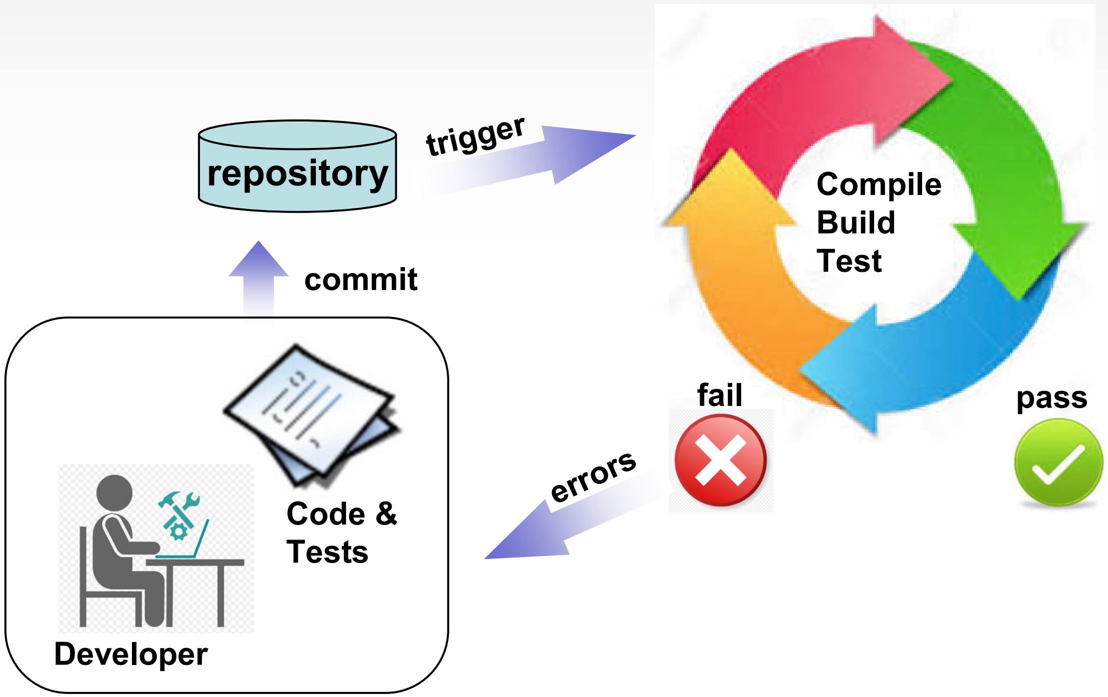
OK – so automation can address this

MULTICORE DESIGN SIMPLIFIED
imperas

- 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

MULTICORE DESIGN SIMPLIFIED
imperas



But how to test?

MULTICORE DESIGN SIMPLIFIED
imperas

- 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

MULTICORE DESIGN SIMPLIFIED imperas

- 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 requires 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

MULTICORE DESIGN SIMPLIFIED
Imperas

- 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 NIRA 6 months of road data need to run tests overnight)
    - (e.g. GPS chip sends position every second)
- …

Adopting CT for embedded needs simulation

MULTICORE DESIGN SIMPLIFIED imperas

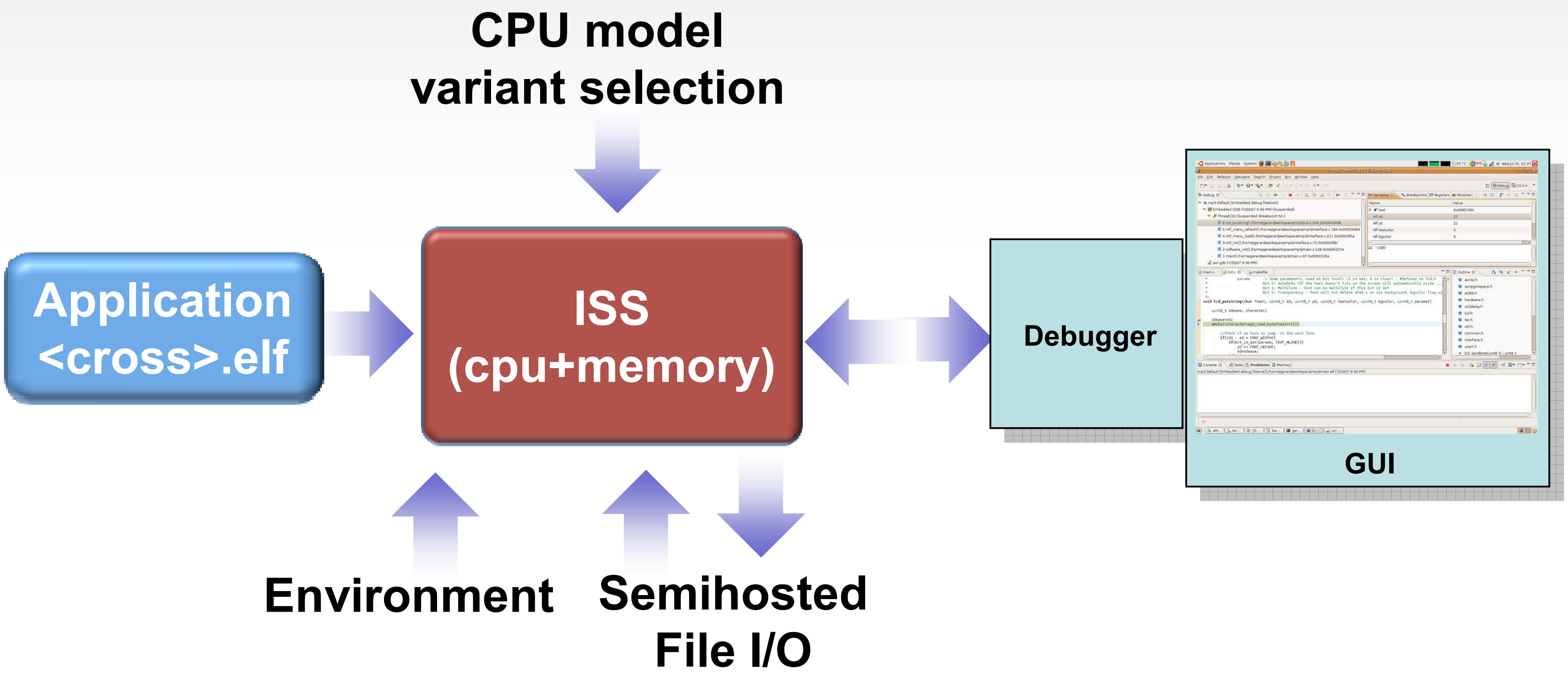
- Imagine a software build system without access to 'make' or 'ant'
  - they enable effective build automation
- Simulation enables the effective automation of testing embedded systems as part of a CI/CT environment
- Simulation enables full automation
  - with no manual intervention
- Use of hardware is just too hard

=> Virtual Platforms (simulation) enable CT for embedded

So what are we talking about here in terms of simulation

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

Instruction Set Simulator (ISS)



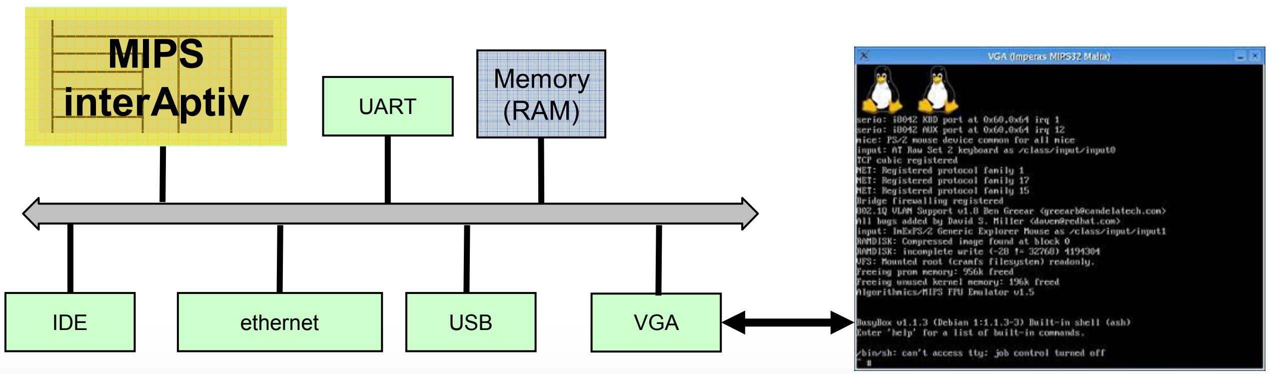
example
run fib

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

MULTICORE DESIGN SIMPLIFIED imperas

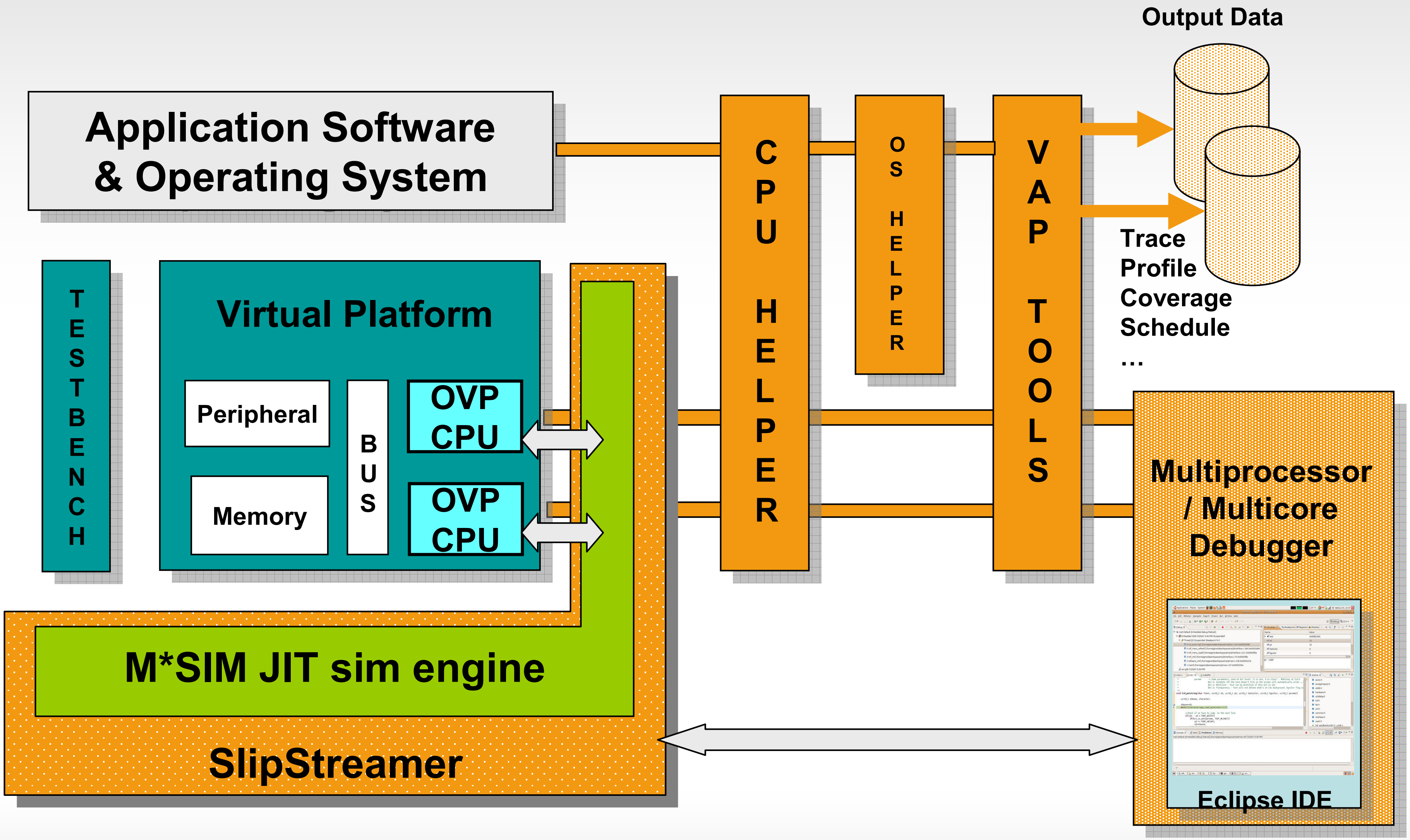
- The virtual platform is a set of instruction accurate models that reflect the hardware on which the software will execute
  - Could be 1 SoC, multiple SoCs, board, system; no physical limitations
- Run the executables compiled for the target hardware
- Models are typically written in C or SystemC
- 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, …

MIPS Malta Extendable Platform Kit (Linux)



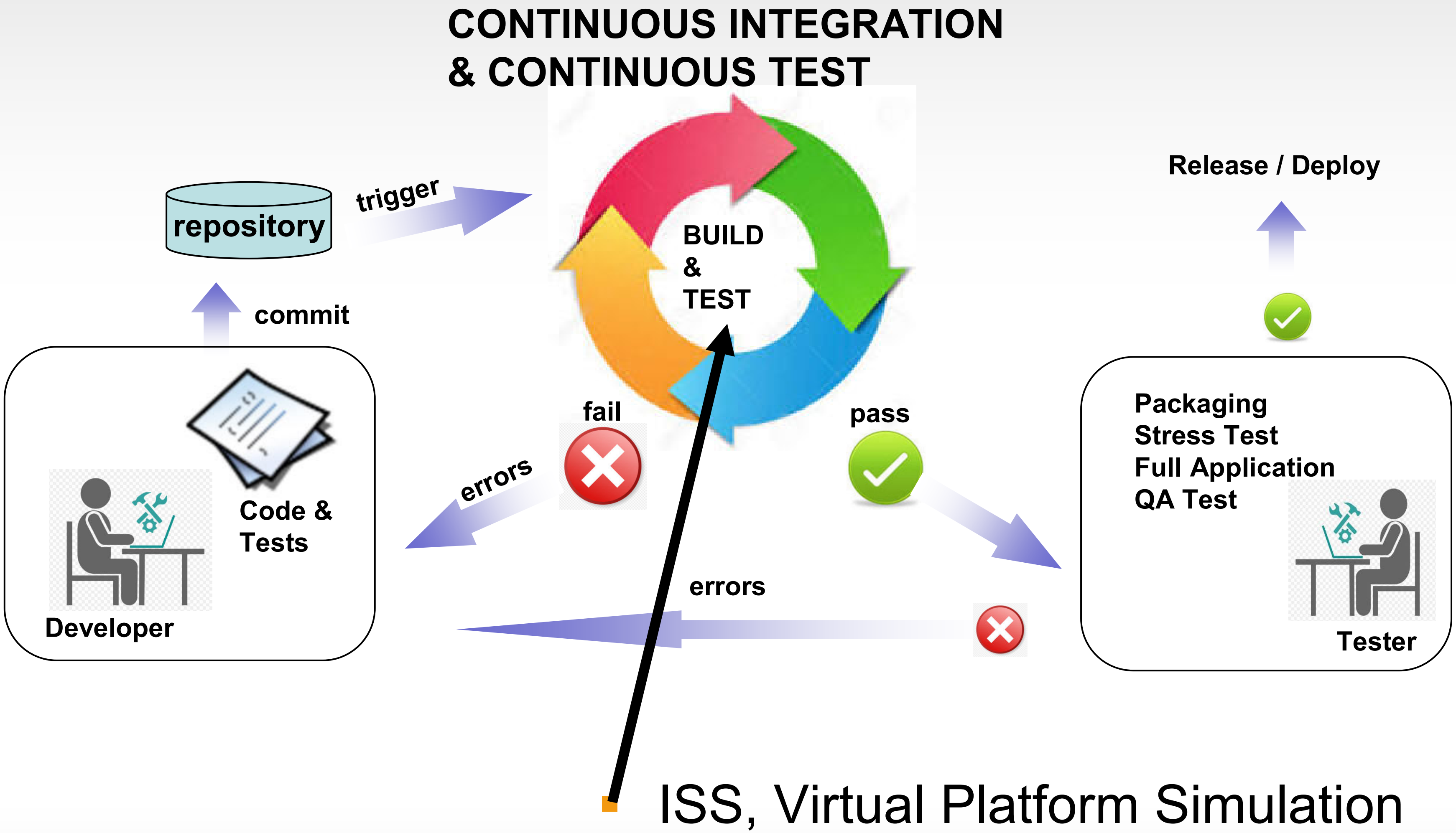
Simulation Architecture can include tools

MULTICORE DESIGN SIMPLIFIED
imperas



Simulation is a key component of embedded CI/CT environment

MULTICORE DESIGN SIMPLIFIED imperas



Demonstration

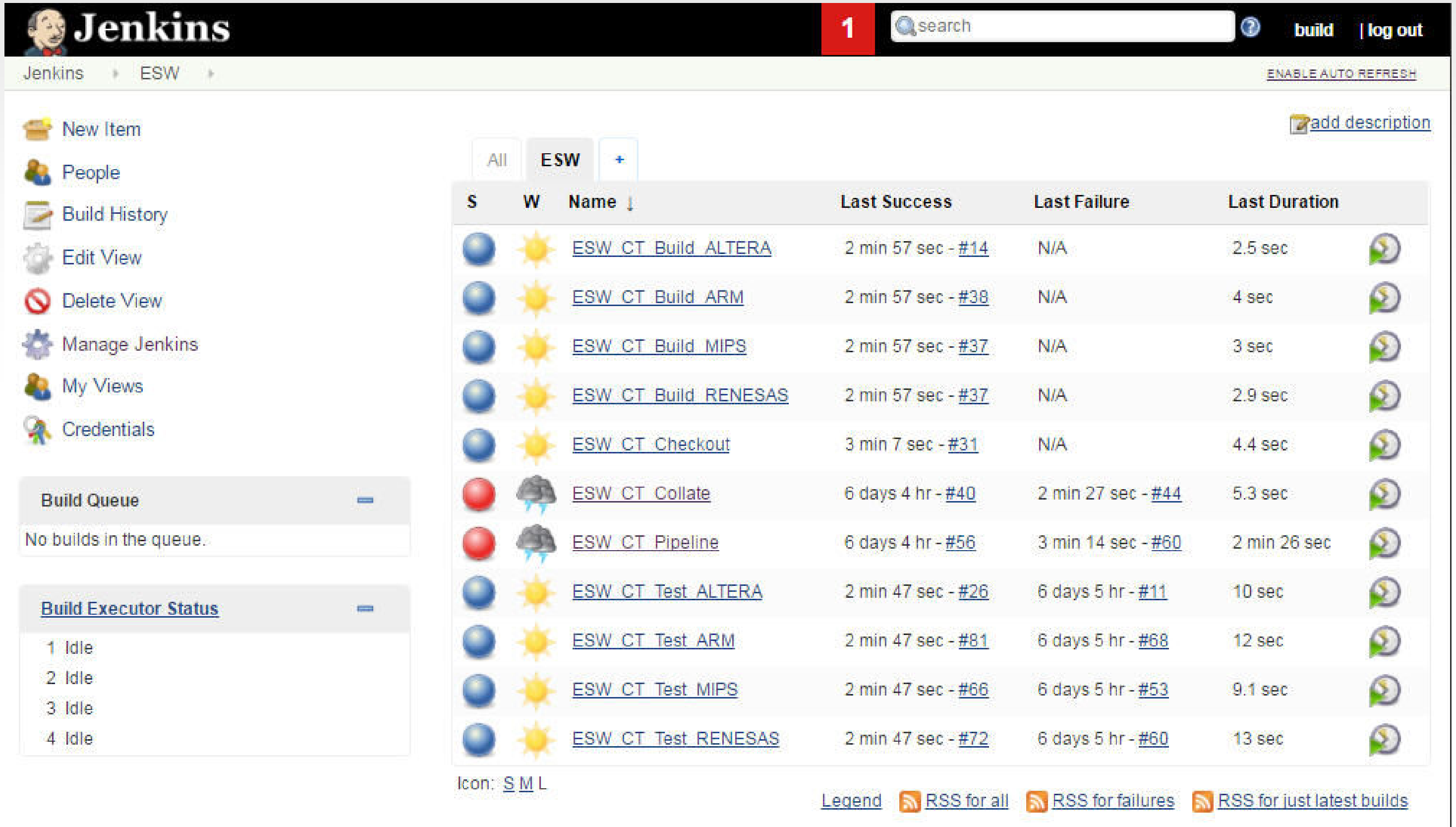
MULTICORE DESIGN SIMPLIFIED imperas

- Imperas ISS simulation used with Jenkins environment

- Edit, compile, local test, checkin -> triggers build
- Successful build -> triggers testing
- Testing completion -> triggers results

Jenkins ‘Items’

MULTICORE DESIGN SIMPLIFIED
imperas



- 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

MULTICORE DESIGN SIMPLIFIED

imperas

Build

Execute shell

Command

```
#!/bin/bash

#
# Setup the environment
#
. /etc/bashrc

isetup /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}"

PDIR=/home/build/jenkins/workspace/Project/Build/ESW_CT_Project

pushd ${PDIR}
    make build ARM
    rc=$?
popd

exit $rc
```

Can use scripts, like bash e.g. Test

MULTICORE DESIGN SIMPLIFIED

imperas

```
# usage: unknown_program_name [switches] [inputfile]
# Switches (names may be abbreviated):
#   -quality N     Compression quality (0..100; 5-95 is useful range)
#   -grayscale     Create monochrome JPEG file
#   -optimize      Optimize Huffman table (smaller file, but slow compression)
#   -progressive   Create progressive JPEG file
#   -targa         Input file is Targa format (usually not needed)
# Switches for advanced users:
#   -dct int       Use integer DCT method (default)
#   -dct fast      Use fast integer DCT (less accurate)
#   -dct float     Use floating-point DCT method
#   -restart N     Set restart interval in rows, or in blocks with B
#   -smooth N      Smooth dithered input (N=1..100 is strength)
#   -maxmemory N   Maximum memory to use (in kbytes)
#   -outfile name  Specify name for output file
#   -verbose  or  -debug   Emit debug output
# Switches for wizards:
#   -baseline      Force baseline quantization tables
#   -qtables file  Use quantization tables given in file
#   -qslots N[....]    Set component quantization tables
#   -sample HxV[,...]  Set component sampling factors
#   -scans file    Create multi-scan JPEG per script file

switches[0]="-dct int"
switches[1]="-dct fast"
switches[2]="-dct float"
switches[3]="-grayscale -dct int"
switches[4]="-grayscale -dct fast"
switches[5]="-grayscale -dct float"
switches[6]="-quality 10"
switches[7]="-quality 20"
switches[8]="-quality 50"
switches[9]="-quality 75"
switches[10]="-quality 100"

#
# Cortex-M3 RH850G3M M5100
# --processorvendor imgtec.ovpworld.org --processorname mips32 --variant M5100
#
for i in 1 2 3; do
    variant=Cortex-M3
    if [ "$VARIANT" = "$variant" ] || [ -z "$VARIANT" ]; then
        for op in {0..10}; do
            image=image.${variant}.${i}.${op}.jpg
            sw=${switches[$op]}
            echo "Run $variant $image : $sw"
            iss.exe --quiet --nobanner \
                --processorvendor arm.ovpworld.org --processorname armm --variant ${variant} \
                --parameter UAL=1 --parameter endian=little \
                --numprocessors 1 \
                --program outdir/ARM_CORTEX_M3/cjpeg.elf \
                    -argv ${sw} -outfile ${RESDIR}/${image} test/image.${i}.bmp
            diff ${RESDIR}/${image} test/image.out.${i}.${op}.jpg
            rc=$?
            add_info ${image} ${rc}
        done
    fi

--More--(72%)
```

Jenkins Pipeline
Create 'Stages' from items

Pipeline script

Script

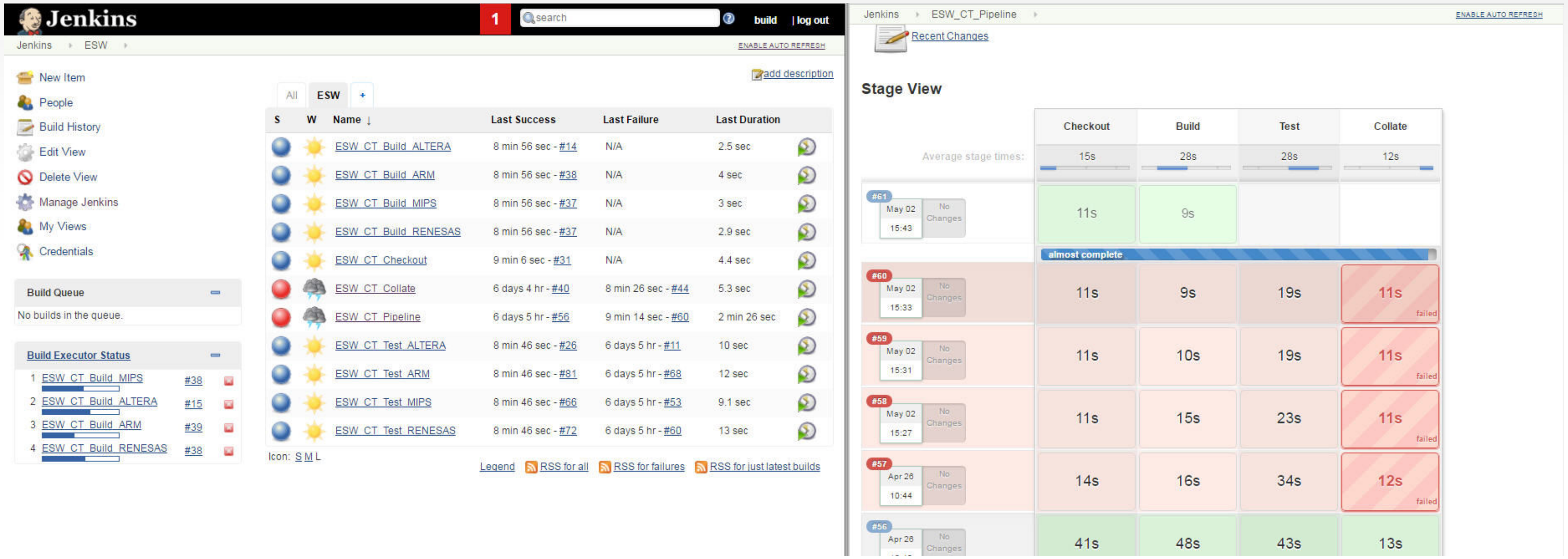
```
stage "Checkout"
build('ESW_CT_Checkout')

stage "Build"
parallel(
    Task1: {
        build('ESW_CT_Build_ARM')
    },
    Task2: {
        build('ESW_CT_Build_MIPS')
    },
    Task3: {
        build('ESW_CT_Build_RENESAS')
    },
    Task4: {
        build('ESW_CT_Build_ALTERA')
    },
)

stage "Test"
parallel(
    Task5: {
        build('ESW_CT_Test_ARM')
    },
    Task6: {
        build('ESW_CT_Test_MIPS')
    },
    Task7: {
        build('ESW_CT_Test_RENESAS')
    },
    Task8: {
        build('ESW_CT_Test_ALTERA')
    },
)

stage "Collate"
build('ESW_CT_Collate')
```

Script

- Note use of parallel & serial

Stages running (1)

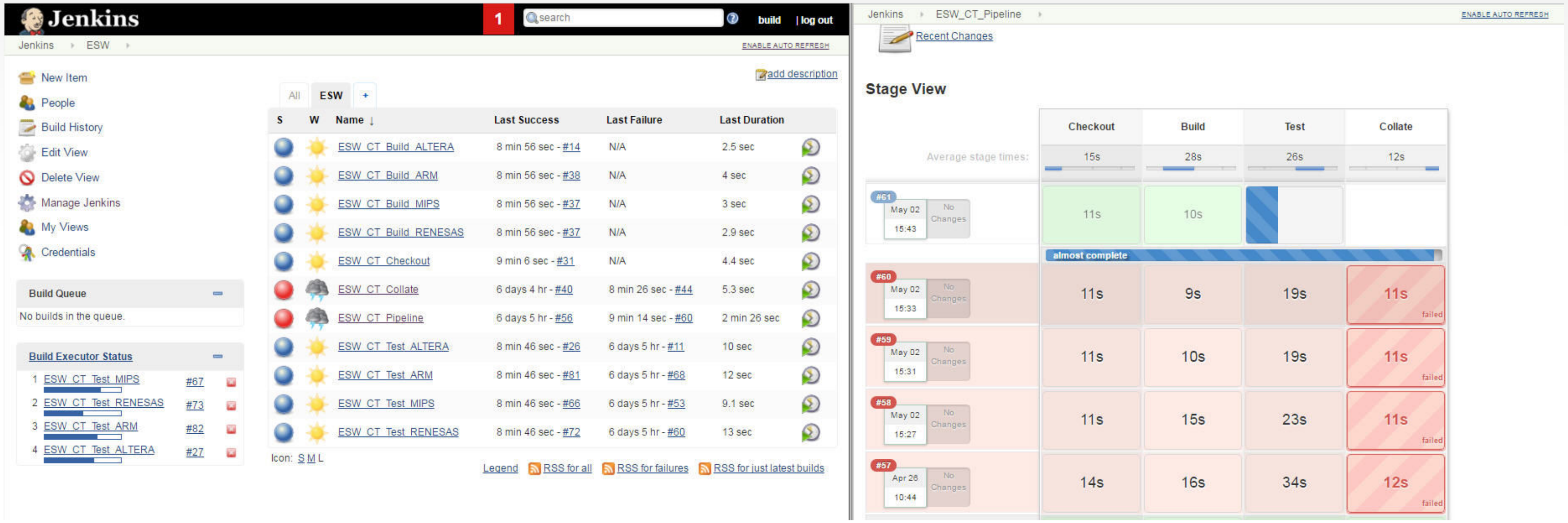
MULTICORE DESIGN SIMPLIFIED
imperas



- Can trigger start of run from code check-in or directly

Stages running (2)

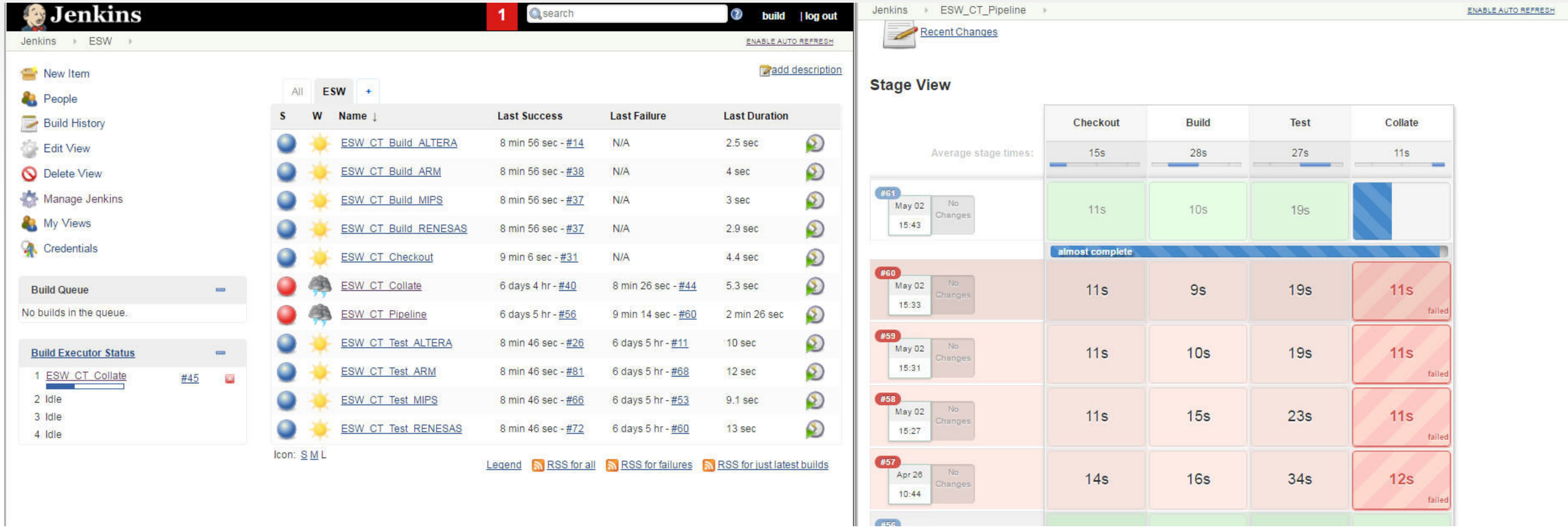
MULTICORE DESIGN SIMPLIFIED imperas



- Can run tests in parallel on available resources (executors)

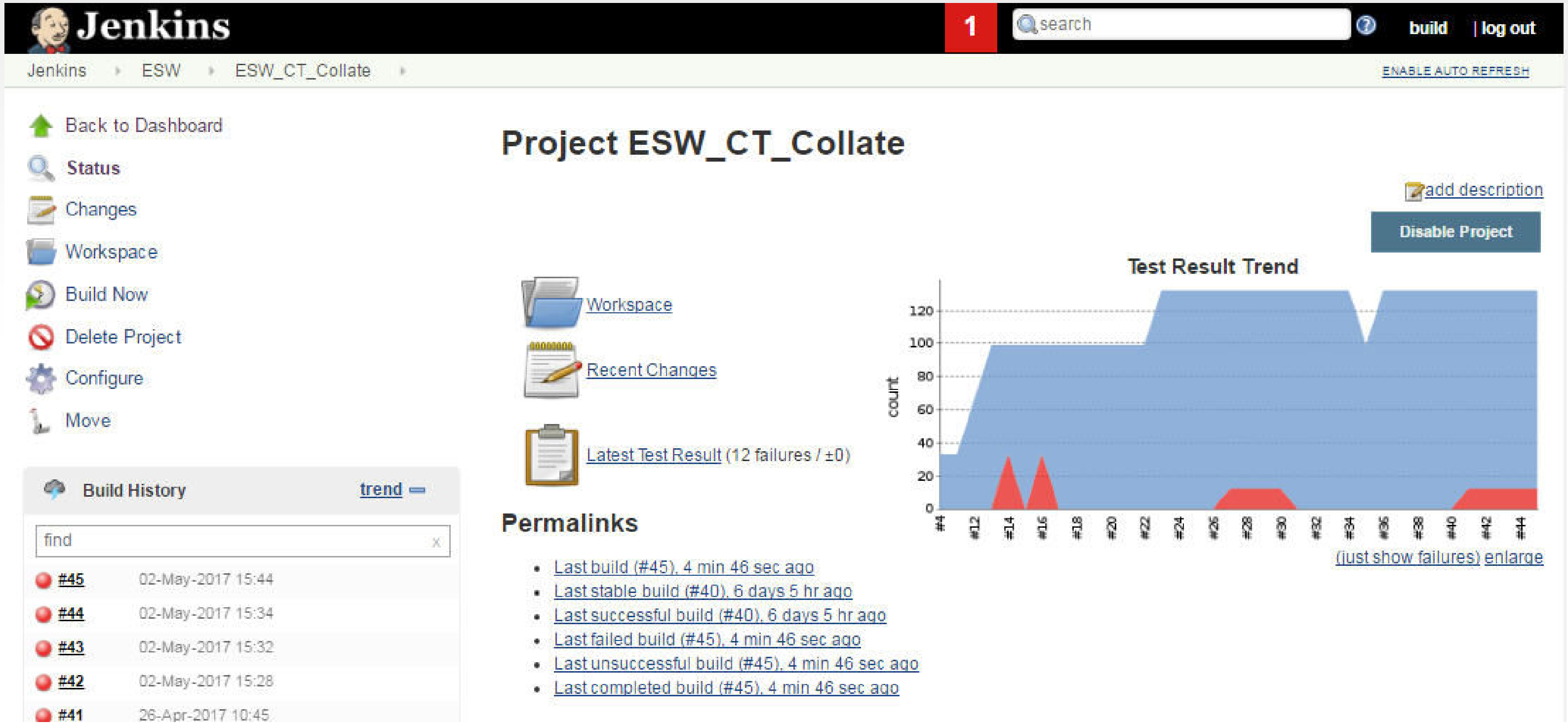
Final Stage is collate results

MULTICORE DESIGN SIMPLIFIED
imperas



- Each test run records test results from its group
- Final task stage in pipeline collates

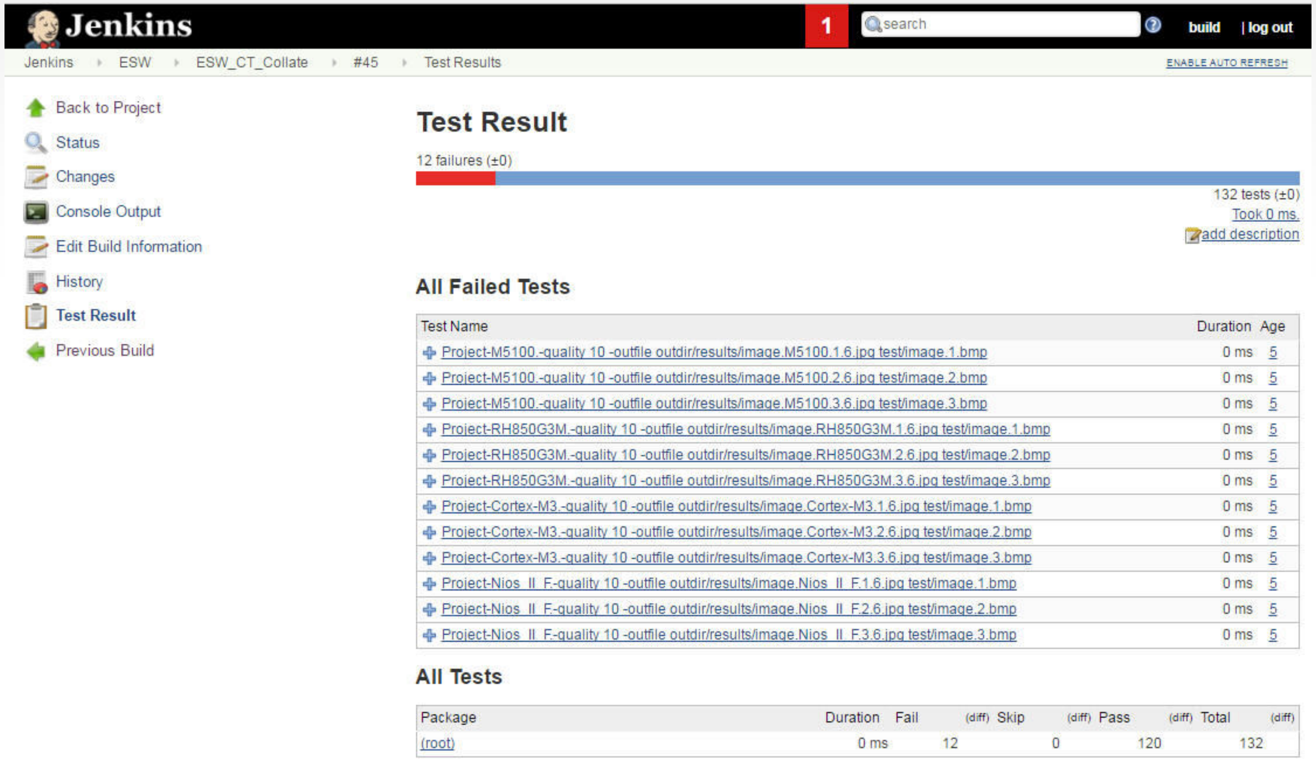
Can see results at end



- See how tests perform over each run
- Management get a dashboard for visibility of project status

Drill down to see failures

MULTICORE DESIGN SIMPLIFIED



Demo Wrap up

MULTICORE DESIGN SIMPLIFIED
imperas

- This showed simple example of developing and testing code for embedded targets using cross compilers to build and ISS to execute
- Used CI/CT 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

MULTICORE DESIGN SIMPLIFIED imperas

- Challenges in embedded software development
- Automation is needed
  - Continuous Integration
  - Continuous Test
- 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

Advanced Modeling *Infrastructure*

MULTICORE DESIGN SIMPLIFIED
imperas

Open Virtual Platforms™ (OVP™) infrastructure and iGen model template generator: Platform creation technology

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 simulator for running models

Key Technology: Library of High-Performance Processor Models

MULTICORE DESIGN SIMPLIFIED
imperas

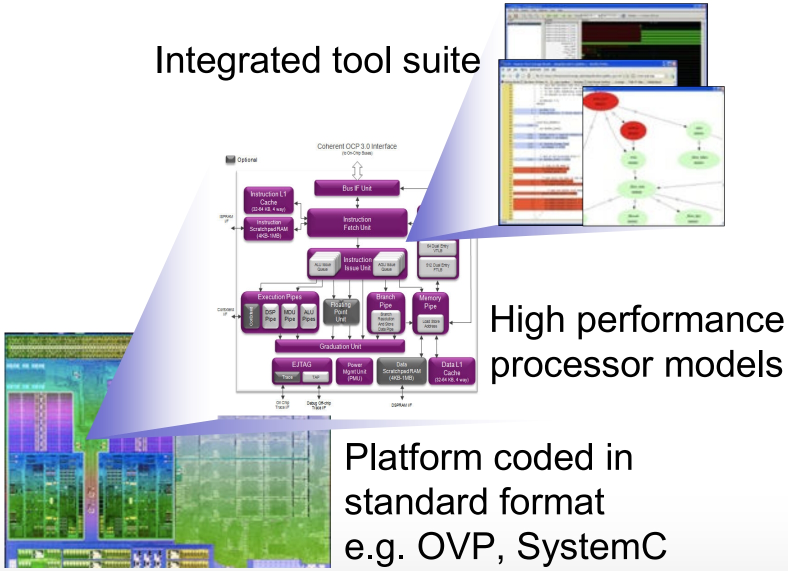
- Over 170 Fast Processor Models in OVP Library
- ARM®: Models for ARMv4™, v5™, v6™, v7™ and v8™ architectures
  - Including MMU, MPU, TCM, Thumb™, Thumb-2™, Jazelle™, SIMD, VFPv3, NEON™, TrustZone®, hardware virtualization instructions, …
- MIPS®: Models for microMIPS, MIPS32 and MIPS64 architectures
  - 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 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

Key Technology: Multicore Development, Debug & Test Tools

MULTICORE DESIGN SIMPLIFIED
Imperas

- 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



"We're delighted to be working with Imperas to deliver a high-performance Instruction Accurate (IA) simulation solution for our many MIPS partners. The performance and capability of the system enables our customers to rapidly produce high-quality code, using features such as Imperas' OVP processor models and M*SDK. This is a clear benefit when facing tight production schedules."

Tony King-Smith, EVP of Marketing, Imagination Technologies

Imagination

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

MULTICORE DESIGN SIMPLIFIED
imperas

Nagravision

NAGRA
KUDELSKI

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

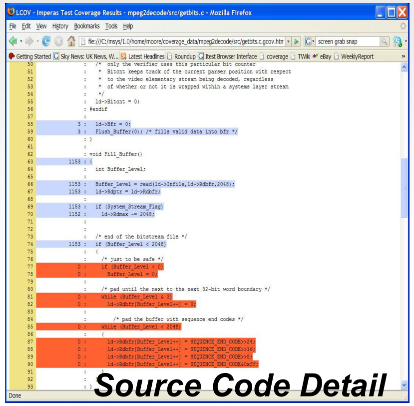
- Imperas use model
  - Virtual platform peripheral models are built by Nagravision (proprietary models) and Imperas (standard I/O, e.g. USB)
  - Use Imperas debugger for software debug and for driver-peripheral model software-hardware co-debug
  - Use VAP tools such as OS-aware tools, code coverage,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

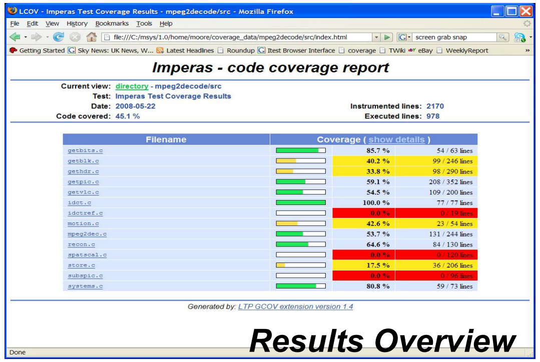
> "At **NAGRA**, we have adopted the Imperas virtual platform-based software development and test tools for our application and firmware teams. The simulation performance, and the tools for software analysis, have added significant value to our daily Agile Continuous Integration (CI) methodology. Our view is that **software simulation is mandatory** to reach metrics required for high quality secured products."

Advanced Software Tools: Code Coverage

MULTICORE DESIGN SIMPLIFIED

- Application
  - Use virtual platform observability to analyze effectiveness of software tests
- Software test and analysis
  - Non-intrusive: no instrumentation or modification of source code
  - Multicore capable
  - Overview and detailed source code analysis reports
  - High performance critical for comprehensive testing



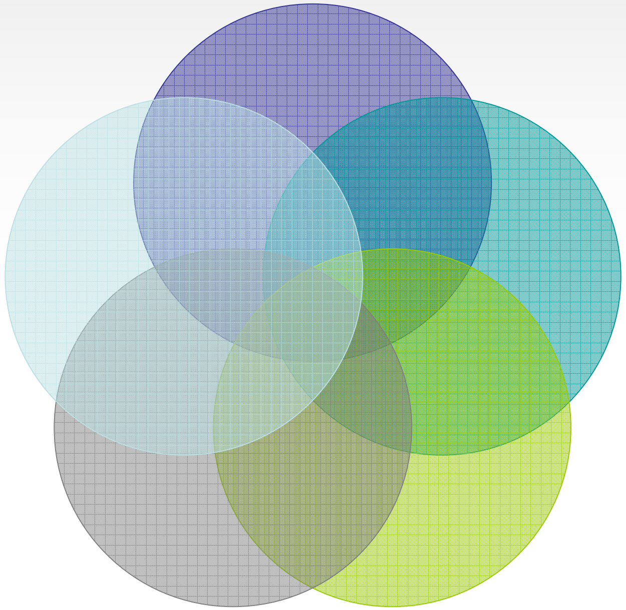


"Imperas with its OVP Fast Processor Models is addressing key issues in software development for embedded systems. We are happy to work with Imperas to ensure that high quality models are easily available to our worldwide customers, helping them to develop and test software faster and more easily using virtual platforms."
Hirohiko Ono, senior manager of the MCU Tools Marketing Department, Renesas Electronics

Imperas Solution Contents

MULTICORE DESIGN SIMPLIFIED
imperas

Methodology
Collaboration with customers, vendor ecosystem



Models
170+ CPU models
200+ peripheral models
30+ platforms

Tools
Leading simulation, debug, software verification tools

Resources
Imperas and partners
Model development
Tool development

Training
Imperas and partners
On-site, customized agenda

Agenda

MULTICORE DESIGN SIMPLIFIED
imperas

- Challenges in embedded software development
- Automation is needed
  - Continuous Integration
  - Continuous Test
- 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 Profiling (VAP) tools
- Solutions for embedded use cases: custom CPU, semiconductor vendors, embedded systems developers
- Experience with Continuous Integration and Continuous Test usage

MULTICORE DESIGN SIMPLIFIED
imperas

- Thank you

- For more information:
  - www.imperas.com
  - www.ovpworld.org

- Contact us: info@imperas.com

MULTICORE DESIGN SIMPLIFIED

imperas

