Programming Languages and Techniques for Today’s Embedded and IoT World

Speakers:

Nick Butler, Sr. Product Marketing Group Manager, National Instruments

Walter Capitani, Klocwork Product Manager, Rogue Wave Software

Moderator:

Curt Schwaderer, OpenSystems Media
Agenda

- Housekeeping
- Presentation
- Questions and Answers
- Wrap-up
Programming Languages and Techniques for Today’s Embedded and IoT World

Nicholas Butler
DAQ and Embedded Product Marketing
The Industrial Internet of Things

Enhanced Requirements for IIoT:
- Latency
- Synchronization
- Security
- Upgradeability
Software Development Challenges in the IIoT

**Tools**
- Math (.m file script)
- Simulation (Hybrid)
- User Interface (HTML)
- FPGA (VHDL, Verilog)
- Host Control (C, C++, .NET)
- DSP (Fix pt C, Assembly)
- H/W Driver (C, Assembly)
- System Debug

**Targets**
- FPGAs
- DSPs
- Multicore Processors

- Long learning curves
- Limited reuse
- Need for “specialists”

- Increased costs
- Increased time to result

---

*NATIONAL INSTRUMENTS*
The Effect of Moore’s Law on FPGAs

Heterogeneous, Massively Parallel SOCs

- Reduced power consumption
- Smaller overall footprint
- Improved re-configurability
- Lower Cost

![Diagram of ZYNQ 7100 with HARD CORE DSP SLICES, PROGRAMMABLE LOGIC & COMMUNICATION, and MICROPROCESSORS linked with EXTERNAL COMMUNICATION.]
ONE PLATFORM APPROACH

MEASUREMENT

TEST

MONITORING

EMBEDDED

CONTROL

LabVIEW™

3RD PARTY SOFTWARE

COMPACTDAQ

PXI AND MODULAR INSTRUMENTS

COMPACTRIO

DESKTOPS AND 3RD PARTY HARDWARE

NATIONAL INSTRUMENTS
Elements of a Platform-Based Approach

- Off the shelf, yet customizable

- Scalable, modular, and extensible – adapt and evolve through software

- Out-of-the-box integration of hardware and software

- Consistent software toolchain across:
  - Applications: test, measurement, design, control
  - Design phases: simulation, modeling, design, validation, deployment
  - Hardware platforms: CompactDAQ, CompactRIO, PXI
  - Programming Languages: C/C++, IEC-61131-3, .m, G dataflow

- A community and ecosystem of IP, add-ons, and toolkits
The LabVIEW RIO Architecture

Ideal for the IIoT

**Computation**
- Real-Time Analytics
- LabVIEW Math & Analysis Libraries
- Algorithms, Decision making

**Connectivity**
- Data Transfer Mechanisms
- Network Interface

**Computation**
- Signal Processing
- Data Reduction

**Connectivity**
- Custom Timing, Triggering and Synchronization
- Custom Protocols

**Control**
- Fast, Deterministic, Closed-Loop Control (MHz rates)
- Safety, reliability

**Connectivity**
- Any Sensor
- Any Protocol
- Industrially Rated
- Signal Conditioning
- Cameras, Drives, Motors, Actuators
LabVIEW System Design Software

- **Single Design Environment**
  Manage and organize all system resources, including I/O and deployment targets

- **Deployment Targets**
  Deploy LabVIEW code to HMIs, RT processors, and FPGA targets

- **Tools Network & Community**
  Leverage the partner ecosystem of more than 175 add-on products

- **Code Portability**
  Access to latest technologies with minimal code refactoring

- **Language Support**
  Augment with existing C/C++ development expertise and code

- **Systems Management**
  Includes multiple system management clients and an API for extensibility

- **Built-In Engineering IP**
  Over 900 control and analysis libraries designed for engineering and science

- **Code Reuse**
  Reuse existing libraries, .m files, HDL and state-based simulation

**Accelerates Your Success**
By abstracting low-level complexity and integrating all of the tools you need to build any measurement or control system
NI Linux Real-Time

Open and standard
- Built on standard Yocto/OpenEmbedded technologies using stable kernel versions
- OS source available: github.com/ni

Fully tested and supported by NI
- Complete software stack maintained by NI: OS, firmware, middleware, drivers
- Thousands of prevalidated open source applications and IP from the Linux community

Real-time reliability
- PREEMPT_RT is the standard for real-time performance on Linux through preemption, priority inheritance, and scheduling
- Real-time project founded by Linux Foundation, Google, NI, OSADL, TI, Altera, ARM, Intel, and IBM

The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Permission to use and/or modify the penguin image is granted by Larry Ewing lewing@isc.tamu.edu and The GIMP.
Any Programming Language
Integration with Flexible and Customizable Hardware

- CompactDAQ Controller
- Compact Vision System
- CompactRIO Single-Board Controller
- CompactRIO System on Module (SOM)
- NI Linux Real-Time
- myRIO

Controller for FlexRIO

CompactRIO Controller
Innovate with a Platform for the IIoT

- Eliminate the need to start from scratch
- Satisfy the computation, connectivity, and control requirements for Industrial IoT applications
  - Meet changing requirements over time with flexible, scalable, and field-programmable products
  - Choose from a variety of high-quality form factors, price points, and performance options
  - Leverage a consistent software environment for
    - Programming every element of the system
    - Simulating, modeling, prototyping, development, and deployment
    - Performing edge and end-to-end analytics
  - Integrate with existing “brown field” systems
Programming languages and techniques for today’s embedded and IoT world

Walter Capitani
Product manager, Klocwork
Presenter

Walter Capitani

Product manager, Klocwork
Rogue Wave Software
walter.capitani@roguewave.com
Twitter: @walter_capitani
Agenda

1. The problem
2. Choosing a language
3. Improve testing with SCA
4. Zero-tolerance for security issues
5. Use safe coding standards
The problem

• How to achieve secure and robust interoperability between embedded/IoT systems and back-end systems?
  – Want to use software development processes and tools that help us accomplish this mission
  – We want to select a language that enables the creation of robust software
  – We want to select a language that is supported by sophisticated automated tools that assist in the creation of more robust software

C/C++ is a programming language that meets the above criteria
New problem

How to use C/C++ to develop secure and robust embedded / IoT software?
Why the language choice?

- C/C++ used in many mission-critical and safety-critical environments
- Allows reuse of well-known software development methodologies which have been proven in safety-critical environments
- Large choice of development tools and libraries to aid in code generation
- Wide selection of tools available to increase security and robustness of software

C/C++ is in top 10 most popular languages*
(based on GitHub + Stack Overflow popularity)

*RedMonk Programming Language Rankings, January 2017
Developers are not trained for security

- Computer science and software engineering graduates are not taught how to code against security threats
- Typically reformed hackers are helpful to test for vulnerabilities, but can only do so after the code is compiled
- This means security issues must be first introduced into the code base before can be detected and fixed
- Security training is expensive and reduces developer productivity
- Security training must be repeated as new threats emerge
The conclusion of a recent study by the Ponemon Institute is that developers want – but do not have—the skills necessary to combat software security threats and they do not feel they are properly trained.

- 45% indicate that their development processes include any activity that supports security requirements.
- Only 41% agree that secure software is a priority for their company.
- 69% believe that securing the applications are difficult/very difficult.
When are defects introduced?

80% of defects are introduced in development phase

- Most defects are introduced when the software is actually written
- However, most defects are only detected in QA or production
- This increases the costs associated with detection and correction of defects
- Each defect reported by QA or Customer Support touches many people across the organization, and derails developers
- The “hidden costs” of these defects can be equal or greater to the effort required to actually correct the defect
The solution

• Improve testing by using static code analysis (SCA)
• Implement a zero-tolerance policy on security issues
• Use safe coding standards such as MISRA and NASA to make code more bulletproof
How can static code analysis help?

• Find common issues in code
  – Buffer overflows (*security exploit or program crashes*)
  – Null pointer dereferences (*your program crashes*)
  – Memory leaks (*processor runs out memory and locks up*)
  – Uninitialized data usage (*data injection*)
  – Platform/OS specifics (*privilege escalation, etc...*)
  – Concurrency (*deadlock*)

• Not easy to spot with the human eye
  – Not generally found by code review
  – Many are traditionally found with dynamic testing *after a failure has occurred in testing or the field*
How does static code analysis work?

- Requires source code
  - Klocwork must be able to compile the code
  - No changes to your existing build flow

- Different types of analysis
  - Klocwork walks down every path of your code
  - Inter-procedural
  - Inter-file

- Klocwork runs the tests that your developers don’t (or won’t) write

- **Klocwork will find defects that other testing won’t**
Klocwork static analysis engine

- Hundreds of checkers for C, C++, C# and Java
- Support for numerous standards
- Customizable:
  - Turn checkers on or off
  - Change the severity of identified defects
  - Add custom checkers

Security
- Buffer overflow
- Un-validated user input
- SQL injection
- Path injection
- File injection
- Cross-site scripting
- Information leakage
- Vulnerable coding practices
- And many more...

Reliability
- Memory and resource leaks
- Concurrency violations
- Infinite loops
- Dereferencing NULL pointers
- Usage of uninitialized data
- Resource management
- Memory allocation errors
- And many more...

Coding Standards & Maintainability
- MISRA, DISA, CWE, CERT, etc.
- Dead code
- Unreachable code Calculated values that are never used
- Unused function parameters
- And many more...
SmartRank defect triage & filtering

- Use SmartRank to prioritize issues in your projects
- Assists developers to select which defects to work on first
- Recommended issues are at the top of the list
- SmartRank is based on a sophisticated analysis of each individual defect (regardless of defect type or severity)
- SmartRank can be used with views, modules and filters
Flexible workflow

Incremental CI analysis for the fastest defect detection
- Quickly and easily integrate Klocwork into an organization’s CI build process

Desktop analysis for all users with integration analysis done whenever you do your standard team-level build
- Developers see and fix defects prior to check in
- Team leads review status with just a simple browser, no software to install

Complete stand-alone mode for very small projects
- Common for smaller microcontroller-based embedded projects where there isn't a "team level" integration build

Traditional post-development analysis
- Geared for audit type functions
Zero-tolerance policy on security issues

• Leading software development organizations are moving to this model
• If changed code contains security issues, the code cannot be checked in
• Security issues are caught as early as possible
• Security issues in the integration build mean the build is “broken”
Use safe coding standards

Significantly reduces the cost of reliable, secure software

- Complements existing testing approaches
- Automated and repeatable analysis

Enforce key industry standards

- DISA STIG, CWE, MISRA
- CERT, SAMATE
- OWASP, DO-178B, FDA validation
- ...and more
Audience Q & A

Nick Butler,
Sr. Product Marketing Group Manager,
National Instruments

Walter Capitani,
Klocwork Product Manager,
Rogue Wave Software
Thanks for joining us

Event archive available at:

http://ecast.opensystemsmedia.com/

E-mail us at: jgilmore@opensystemsmedia.com