Trends in Embedded Software Development

Dan Smith
Principal Engineer, Barr Group
Topic: “Trends in Embedded Software”

Advances in hardware and software impact each other
About the Presenter

- Principal Engineer, Barr Group
  - Consulting – Architecture and Process
  - Product Development
  - Expert Witness

- B.S.E.E., Princeton University
  - Computer Architecture
  - Microprocessor Design

- Industry experience
  - Telecom & Datacom
  - Industrial Control
  - Low-power Devices & Consumer Electronics

- Range of processor architectures, RTOSs, etc.
About Barr Group

- Internationally recognized leader in embedded systems
  - CTO – Michael Barr (Netrino, Embedded Systems Design, ...)

- Focus on reliability and security
  - Training
  - Consulting on process and architecture
  - Design services

- Renesas RX62N Development board
  - Center of 5-day Embedded Bootcamp course
  - Also available: “Bootcamp in a Box”

- More information:
  - Andrew Girson, CEO, agirson@barrgroup.com
  - Web: www.barrgroup.com
  - Email: embed@barrgroup.com
  - Phone: (866) 65-EMBED
Foundation for this presentation

- **Personal experience**
  - 20+ years of embedded development
  - Wide variety of industries, code sizes, and processors

- **Barr Group experience**
  - Even wider range of experience and exposure
  - Survey of our customers and partners

- **Industry Research and Reports**
  - Necessary to stay on top of emerging trends
  - Identify emerging processes, architectures, standards...

- **Personal Connections & Contacts**
  - e.g. Michael Barr, former editor of “Embedded Systems Programming” magazine
Renesas Technology & Solution Portfolio
Role of firmware in embedded products

- Firmware is largest part of embedded product development...
  - By staff / headcount
  - By cost
  - By time

- Also considered to be riskiest
  - In terms of product security & stability
  - In terms of schedule slips

- Why?
  - Beginning without requirements
  - Changing requirements
  - We can always sneak in “one more feature”
  - We can always “fix it in the field”
Importance of firmware correctness

- Quoting Jack Ganssle:
  - “Firmware is the most expensive thing in the universe”

- Quoting Michael Barr:
  - “Neither reliability nor security can be tested, debugged, or patched into a product. They must be designed into embedded systems”

- The bottom line:
  - Important to get firmware right (as close to “right” as possible) – the first time
What’s driving the trends?

- More functionality being pushed into firmware
  - “Has to be done in hardware” – not necessarily!
  - Some products have a large up-front cost
  - Easier to update in the field
  - Easier to release feature upgrades ($$$)

- More powerful processors
  - Less focus on conserving bytes & CPU cycles
  - More focus on writing correct, robust, maintainable firmware

- Better tools
  - Easier than ever to prototype something quickly
  - More time focusing on product-specific requirements and functionality
Signal processing
Increased use of Signal Processing

- Not necessarily requiring DSP anymore
  - Microcontrollers are becoming more and more capable
  - Custom instructions, faster memories, bus architectures, etc.

- Multimedia is one driver
  - Audio encoding / decoding
  - Video encoding / decoding
  - Image processing

- Real-time control systems
  - Signal Conditioning / Filtering
  - Control loops
Renesas MCUs and Signal Processing Library

- Dual CPU (MCU + DSP) can often be replaced by single MCU
  - Cost reduction, simpler design

- Renesas MCUs
  - SuperH (SH2A-FPU, SH4A)
  - RX Family (RX600)
    - MCU/DSP Hybrid (FPU, MAC, Barrel Shifter)
    - “Digital Signal Controllers”
  - M16C Family (R32C / 100)

- Renesas Signal Processing Library
  - Collection of the most useful routines
  - Abstract away complexities of underlying implementation
    - Focus on algorithms & system-level design
  - Fine-tuned and highly-optimized for Renesas CPUs
  - Floating point and fixed point
  - Callable from C and C++
Test-driven development
Test Driven Development (TDD) – what is it?

- A development methodology
  - Not a test methodology...
  - ... although when done properly, resulting code is well-tested

- Development is driven (guided) by tests
  - Not the other way around

- Tests are written before the code
  - Forces the question: “How will I test this?”
  - Emphasizes importance of separating interface from implementation
Test Driven Development (TDD) in one slide

- Short, iterative development cycles
  - Write test(s)
  - Run test(s) – FAIL (no logic in code)
  - Write code to make tests pass
  - Run tests – PASS

- Emphasis on writing the minimum code to pass tests
  - Tends to reduce YAGNI situations

- Emphasis on automation
  - Essential if battery of tests is to be run over & over again

- Refactoring done at end
  - Take a step back, improve internal structure without breaking anything
Test Driven Development benefits

Assuming the process is followed properly

1. All code is unit tested
   or at least what is feasible

2. Regression test suite is already developed
   Did we break anything?

3. Tests are automated
   No barrier to running a “health test” on current build
   Lack of automation is the single biggest obstacle

4. Write code and run tests before target HW available
   Test harness + removal of global state – simplifies testing
Decoupling software from target platform

Application

Hardware Abstraction Layer (HAL)

Host Implementation

Target Implementation

Host Platform

Target Platform
Middleware and 3rd party firmware
Increasing use of middleware

- Products are becoming more complex and capable

- Many capabilities are “enablers”
  - Network connectivity
  - Plugins / User Applications
  - Graphical User Interfaces

- Most companies aren’t in the business of developing user interface toolkits, network stacks and frameworks
  - But they want to use them

- Development -- more integration activities than ever
  - Final product - combination of company’s differentiating technology & 3rd-party enabling middleware
Rich set of middleware and 3rd-party firmware
Use of C and C++
Trends in Programming Languages

- C (today)
- C++ (today)
- C (expected)
- C++ (expected)
A natural fit...
- Ability to access & address memory directly
- “volatile” keyword
- Sometimes referred to as a “high level assembly language”
- Gives you “enough rope to hang yourself”

... but fewer graduates than ever are being taught C!
- And even fewer how write robust real-time firmware in C
- Is this a good thing or a bad thing for veteran C developers?

New C standard – C11
- Most features not specifically targeted at firmware
  - ... but still useful
- Toolset support still lagging
Use of C++ in firmware development

- Usage of C++ increases with project complexity
  - 32 kilobytes or less: 8%
  - Up to 256 kilobytes: 24%
  - Up to 1MB: 56%
  - Greater than 1MB: 71%

- Usage of C++ increases with processor architecture
  - 8-bit: 6%
  - 16-bit: 18%
  - 32-bit: 52%

- A couple of important notes
  - Numbers reflect percentage of projects using any C++
    - Virtually all of the projects above reported use of C as well
  - Much embedded C++ in use is very “C like”
Use of C++ going forward (2012 and beyond)

- New C++ standard (C++11)
  - Finalized in late 2011
  - Compiler support just rolling out now

- Now you can write firmware in C++ that is:
  - Safer (e.g. static (compile-time) assertions)
  - Faster (e.g. rvalue references / move semantics)
  - Smaller (e.g. constant expressions)
  - Simpler (e.g. type inference (“auto”), range-based for, lambda)

- Change in CPU, change in programming language
  - Move from 8/16 bit to 32-bit
    - “Now we can use C++”
  - Typically involves change in hardware, tools, etc.
    - Good time to consider change in implementation language
IDEs/toolsets and programming languages

- Use of both C and C++ supported
  - Possible to migrate from C to C++
  - Also possible to mix C & C++ in same application

- Major IDEs support both programming languages
  - e² Studio (Eclipse)
  - Renesas High-Performance Embedded Workshop (HEW)
  - IAR Embedded Workbench
  - Green Hills MULTI

- Not locked in to any particular toolset or language
  - Relatively easy to migrate to different IDE
  - C and C++ compilers mostly standards-compliant
Use of open-source software
Increased use of open-source software

- More and more projects using open source software

- Motivations:
  - Source code availability
  - Licensing cost
  - Control / freedom

- Often not considered:
  - Stability / maturity of software
  - Cost / value of engineer’s time
  - License requirements

- Most often cited reason for using open-source:
  - Not being dependent on software vendor for bug fixes / updates
A few examples: Open-source & embedded

**Host Side**
- Host operating system
  - Linux (also on target side for larger systems)
- Build tools
  - GCC / G++ / LLVM
- IDE
  - Eclipse

**Target Side**
- Bootloader
  - U-Boot Bootloader
- RTOS
  - FreeRTOS
- Filesystem
  - FatFs (ElmChan)
- TCP/IP
  - lwIP, uIP
Migration from 8/16 bit to 16/32 bit
8/16 bit CPUs

- **Not going away**
  - Extremely simple, high-volume, cost sensitive
  - But the cost difference is decreasing
    - Especially when considering roadmap

- The “hidden cost”
  - Extra effort spent on optimizing memory & CPU cycles

- CPU’s architecture is often exposed to programmer
  - Code is typically not portable
  - More time working around architecture (e.g. bank switching)

- Development tools often limited
  - Legacy compilers
  - Often no support for C++
    - or even updated C (C99, C11)!
The move to 32-bits

Key drivers

- Cost of 32-bit CPUs going down
  - Furthermore, CPU becoming smaller percentage of B.O.M.
  - Companies now consider indirect cost (development time)

- Products becoming software-intensive
  - Inexpensive, fast, integrated flash memory
  - Focus is on productivity, cycle time, debugging
  - More sophisticated development tools
  - Ability to program in higher-level languages
  - Less concern about memory and processing speed

8/16 bit MCUs not going away

- Extremely simple, high-volume, cost sensitive
- Larger memories, faster clock speeds
Renesas – 8/16/32 bit CPUs

- Renesas – rich MCU portfolio under one umbrella
  - One size does not fit all

- 8/16 bit CPU offerings
  - RL78, 78K, R8C
  - M16C, H8 / H8x, 720/740

- 32 bit offerings
  - RISC: RX Family (RX600, RX200, ...)
  - CISC: V850, SuperH

- Roadmap / migration path
  - Offerings for almost any power / performance requirement
  - Commonality in IDEs and peripherals reduces learning curve
Low Power Designs: The Firmware Aspect
Motivation for low-power design

- Entire track at this conference on low power design
- What is heat?
  - Wasted energy!
- Reduced heat
  - Reduced cooling costs and complexity
  - Better reliability
- Battery-powered devices
  - Longer run time
  - Smaller package
- Happier customers
  - Better for environment
  - Less expensive to operate
Evolving role of firmware in low power devices

- Certainly, hardware design plays a large role
  - Often overlooked: role of software!

- Software is much easier to:
  - Patch / Change / Reprogram
  - Adapt & Evolve
  - Optimize

- Earlier: design hardware, write firmware, measure, panic!

- Today: design low-power firmware from outset
  - Emphasis on Hardware / Software co-design
  - Component selection – power down modes
  - Also important – internal MCU peripherals and clocks
Renesas – high performance, low power

- Ultra-low voltage operation
  - 1.62V operation @ up to 20MHz, 31 DMIPS

- Low power consumption
  - 130μA/DMIPS (run mode), 1.3μA with RTC on
  - 0.45μA with RTC off

- High performance
  - 1.56 DMIPS/MHz, 78 DMIPS @ 50MHz, 2.7V to 5.5V

- Zero wait-state Flash
  - 2KB block size, Erase/Write operation down to 1.62V
  - Programmed at 1.62V
  - Data flash programmable while code is executed (BGO)

- Scalable
  - 36-145 pins, QFP, LGA, QFN
  - 32KB - 1MB
  - Multifunction pin controller
Modeling, Simulation, Code Generation, Frameworks
Modeling & Simulation - Hardware

- Even important for simple circuits and hardware
- The interfaces & behavior of individual components are well-defined
  - Perfect for simulation
- Answers the question: “When I put this all together, will things behave the way I expect?”
- Faster & easier to test and debug than a real circuit
Modeling & Simulation - Software

- Easier to get started
  - Not battling with nuances and quirks of new hardware
  - Friendly, robust, powerful environment
  - CPU & memory are essentially “unlimited”
  - Less intimidating

- Initial focus is not concerned with final target hardware
  - Task partitioning
  - Algorithm correctness
  - Data flow throughout system

- Getting these things right up front makes integration with new hardware easier (already proven correct on known HW)
Development / modeling sophistication

- No Design ("just code")
- Design (Visio, MS Word, simple CASE)
- Model (Quantum Modeler, UMLet, etc.)
- Executable Model (Rhapsody, Enterprise Architect, etc.)

- Code
- Drawings & Text
- UML
- xUML
UML (Unified Modeling Language)

- UML is the “lingua franca” in the modeling world
  - Statecharts (behavior)
  - Sequence Diagrams (interaction)
  - Class Diagrams (structure)
UML & Embedded Systems

- UML used increasingly for modeling embedded systems
  - Myth: UML only useful for OO / C++ systems

- UML statecharts: powerful, widely-used
  - Most embedded systems are event-driven at their core
  - Support for hierarchy in states
    - Reduce complexity and code size
  - Entry & exit actions
    - Enforce invariants
  - Guard conditions
    - Evaluate extended state variables at run time

- UML sequence diagrams
  - Show interactions between objects / tasks / threads
  - Describe data exchange (and often timing) between actors
One size does **not** fit all

- Wide range of tools (from Visio to Rhapsody)
  - Visio: pure drawing tool
    - Not even UML aware
    - Better than nothing, but not by much!
  - Rhapsody: full-blown modeling tool
    - And all the complexity and “ceremony” that comes with it

- Different tools have different capabilities and features

- Consider the Costs
  - Financial
  - Learning Curve
  - Maintenance
Modeling & Design Tools

- Heavyweight tools
  - Typically support round-trip engineering and executable UML
  - Examples:
    - Rhapsody (IBM)
    - Enterprise Architect (Sparx Systems)
    - Visual Paradigm (only C++)
  - These tools usually supply (or require) a framework

- Lighter weight tools
  - Quantum Modeler (Quantum Leaps)
  - VisualState (IAR)
  - Focused more on creating correct, robust hierarchical state machines than full-blown UML system engineering
  - Code generation in C as well as C++
    - Targeted more at embedded systems
Simulation and Modeling – Hidden Costs

- False sense of security
  - Not shaking out hardware
  - Timing issues are largely undiscovered
  - Memory/CPU limitations often not exposed
  - Less intimidating development & debugging

- Learning Curve
  - Complex tools impose a serious learning curve
  - Easy to become bogged down in options & drawing / layout
  - Beware of analysis paralysis!

- Move to target
  - Use architecture-independent data types (e.g. uint32_t)
  - Abstract away underlying hardware
Code Generation

- One of the biggest motivations for modeling
  - UML-aware drawing tools are far better than Visio...
    - ...but code generation is even better!

- With code generation, the design is the code
  - “Design Rot” not possible with majority of application logic
    - Fix/enhance code by modifying design & re-generating

- Testing burden doesn’t disappear...
  - ... but at least the tedium of coding up diagrams does
  - Just because a model simulation works, there is no guarantee that the target code will
    - Code generation can have flaws just like compilers
    - Integration issues with marginal hardware
Code Generation Example (Quantum Modeler)
Security and Reliability
Greater Need For Reliability And Security

- More embedded systems than ever impact safety
  - Medical Devices (defibrillators, insulin pumps, radiation)
  - Transportation (avionics, rail, road vehicles)
  - Industrial Control (food processing, robotics, smelting)
  - Energy (electric grid, nuclear power plants)

- But the question is:
  - Are these industries and products as safe as they could be?
    - And safe as they **should** be?

- Reliability and security are the cornerstones of safety
  - And they have to be considered from the outset

- Yet systems are becoming **larger** and **more complex**
  - What can be done to make and keep products safe and secure?
Reliability & its role in embedded systems

- Reliability – one definition
  - A product’s resistance to malfunction or failure in the presence of any single failure

- Statistical analysis must demonstrate that a double fault is infinitesimally small and improbable
  - Not all faults are independent and unrelated!
  - Some industries even require mitigation of double faults

- A reliable system starts with reliable hardware
  - But how software handles hardware problems is key

- An unreliable system can never be a safe system
  - Is your system reliable? Does it need to be?
Failure Mode and Effects Analysis (FMEA)

- Identify and understand failures and their consequences

- Risk = Likelihood x Consequences

- FMEA: Assign values for failure likelihoods and consequences
  - Identify most cost-effective areas for improving reliability
  - Highlight areas of greatest likelihood / consequences

- Minimize impact of failure on system
  - Redundancy
  - Fault Identification & Isolation

- Firmware must be robust in the face of uncertainty
  - Hardware faults, user input, harsh environments, etc.
Renesas & Safety: IEC 60730

- IEC 60730 (International Standard)
  - Intended to increase quality & robustness of devices
  - Defines classifications for automatic electronic controls
    - Class A - not intended ... for equipment safety
    - Class B - prevent unsafe operation of controlled equipment

- Renesas RX200 family
  - Surpass requirements for Class B
  - Built in functions to ensure safe automatic electronic control of class B appliances
  - Examples: CPU registers, interrupt handling, memory tests, communication, I/O peripherals, A/D tests
Security - The Risk

Why?
- Corporate espionage
- Blackmail
-Hackers (misuse leads to flakey or dangerous product)

Security often an afterthought
- “Closing the barn door after the horses have escaped”

Difficult to retrofit security and patch holes
- But the alternative requires more time up front

Do you want to pay now?
- Or 10x as much later?
Increased Connectivity = Increased Vulnerability

- Connectivity everywhere in embedded devices
  - Wireless
    - WiFi, 3G/4G, Bluetooth, Zigbee, Satellite, etc.
  - Wired
    - Ethernet, CANbus, DeviceNET, USB, etc.

- Each of these is a potential attack vector
  - Medical devices (wireless attack on ICDs and insulin pumps)
  - Industrial control & automation (Stuxnet, Duqu, Flame,...)
  - Vehicles (OBD-II exploit)

- Don’t forget about physical access
  - JTAG port, soldered memory
  - Other attacks: side channel, brownout/glitch, thermal...
  - *But we’re talking about software right now*...
What can be done?

- Embedded Devices have an advantage
  - Don’t typically run arbitrary code

- Don’t write encryption software yourself
  - What business are you in?
  - Focus and competence
  - Keeping up-to-date

- Use a coding standard that prioritizes correctness over form

- Don’t do in software what can be done in hardware

- Renesas addresses security in multiple ways
  - Renesas middleware library (DES, AES, RSA, key exchange)
  - Secure MCUs – AE-x series and RS-4 series
Wrap Up & Parting Thoughts

- Importance of firmware has never been greater
  - Develop more in less time without compromising correctness

- Modeling, code generation, frameworks
  - Design becomes code, no need to “re-invent the wheel”

- Security and reliability have never been more important
  - Difficult / impossible to retrofit

- Renesas is well-positioned to address all of these trends
  - Wide offering of MCUs and hardware platforms
  - Extremely broad “ecosystem”
    - Compilers, IDEs, middleware, RTOS, etc.
  - Be sure to work with FAEs, Salespeople, Distributors
Reading and references

- Available for download:

- Books:
  - “Embedded C Coding Standard”, Michael Barr
  - “The CERT C Secure Coding Standard”, Robert Seacord
  - “Test Driven Development for Embedded C”, James Grenning
  - “C++ Primer, 5th Ed.”, Lippman, Moo, LaJoie
Questions?
Please provide your feedback...

- Please utilize the “Guidebook” application to leave feedback

or

- Ask me for the paper form for you to use

Thank you!