Svante Ekholm Lindahl
Embedded Software Developer

maximatecc
svante.ekholm@maximatecc.com
www.maximatecc.com

Title: Hardware and Software Testing
Date: December 3rd 2013
Humans in control of industrial machinery in critical environments
Something about me

- 66% grown-up
  - Live in a house
  - Drive a Volvo

- VVV = “Villa, Vovve, Volvo”
- Thesis on firmware / BIOS

Embedded Software Developer
- Firmware design/coding:
  - Touch screen controllers
  - Battery controllers
  - System coprocessors
- Test application engineer
  - I write test software
  - (…not software tests?!)
maximatecc is a hardware and software company.

- We build rugged computers for use in industrial vehicles.
- We build firmware and software (drivers, API’s) for those computers.
- We provide software services to help customers build applications.
Humans in control of industrial machinery in critical environments

- **Sweden**
  - Alfta
  - Västerås
  - Uppsala

- **Spain**
  - Barcelona

- **United States**
  - Lancaster, PA

- **Mexico**
  - Juarez

- **Malaysia**
  - Penang

Something about maximatecc
Something about our customers

- A typical customer
  - Large company
  - Tough requirements
  - Low unit volumes
  - High unit costs

- Environment
- Reliability
- Robustness
- Performance
- Safety
This lecture is about widening your concept of testing from just software.

- How is hardware tested?
- How are software+hardware systems tested?
- How do hardware issues affect my software testing?
1. Introduction – this lecture in context with other course elements

2. Testing of systems composed of both hardware and software

3. Methods for testing the code of near-hardware software (firmware)
This guest lecture in context
Four questions:

- Does my software work?
- Does my software meet the specifications?
- Does my software still work after changes?
- Can Mr. Pearson make me a better programmer?
Four questions:

- Does my software work?
  - Functional testing
- Does my software meet the specifications?
  - Construction testing
- Does my software still work after changes?
  - Regression testing
- Can Mr. Pearson make me a better programmer?
  - Static code analysis for better code quality
Four questions:

- Does my software + hardware work?
  - Functional testing
- Does my software + hardware meet the specifications?
  - Construction testing
- Does my software + hardware still work after changes?
  - Regression testing
- Can Mr. Pearson make me a better programmer?
  - Static code analysis for better code quality

Humans in control of industrial machinery in critical environments
Humans in control of industrial machinery in critical environments

Four six questions:

- Does my software+hardware system work?
  - Functional testing
- Does my software+hardware system meet the specifications?
  - Construction testing
- Does my software+hardware system still work after changes?
  - Regression testing
- Can Mr. Pearson make me a better programmer?
  - Static code analysis for better code quality
- Is my software+hardware system robust enough?
- Is my software+hardware system reliable enough?
Humans in control of industrial machinery in critical environments
HWSW development

Humans in control of industrial machinery in critical environments
Humans in control of industrial machinery in critical environments

Definitions

- "HWSW system", or simply "system":
  - A design composed of both hardware and software.
  - Blueprints – something abstract.
  - Think of this as a class.

- "Unit"
  - An individual system.
  - Something concrete.
  - Think of this as an instance of the system class.
The HWSW system

- **Hardware**
  - Circuit boards
  - Paths and components
  - Integrated circuits
  - Processors, controllers
  - Mechanics
  - Chassis, cables, ...

- **Software**
  - User applications
    - GUI's, Angry birds, ...
  - Operating system
  - API's, libraries, drivers
  - Firmware
    - I/O controllers, sensor controllers, regulators
Humans in control of industrial machinery in critical environments

The HWSW system

- "Hardware and software are logically equivalent."
- "Yeah, right."
- Hardware is just **unpredictable** software
  - Each assembled device has unique properties
  - Physical inputs and outputs affected by environmental factors
  - Your software may get input data from the hardware
  - When environmental factors get worse, your hardware gets wonky
    - Then your software may fail.
Humans in control of industrial machinery in critical environments

**HWSW development cycle**

- **Traditional waterfall model**

  1. Conceptualize
  2. Design
  3. Prototype
  4. Verify
  5. Produce
  6. Sell
  7. Ship
Humans in control of industrial machinery in critical environments
Development activities

- **Software testing?**
  - Applying software to test software.

- **Test software?**
  - The software used for testing of other software.
  - ...or for testing hardware.
  - ...or for testing hardware and software systems.
Does my **hardware** meet the specifications?
- Does it not explode when voltage is applied?
- Are voltage levels on the board within limits?
- Does Ethernet work?
- Does CAN bus work?
- Does serial port work?
- etc...
Construction testing

- Does my *software* meet the specifications?
  - This leads to *software testing.*
Humans in control of industrial machinery in critical environments

Environmental testing

- Does my system meet the environmental and EMC requirements?
  - Is it still functional at 80 °C?
  - Is it still functional at -20 °C?
  - Does the heater work at -40°C?
  - Does basic functionality withstand EM frequency bombardment?
  - Is the hardware radiating low enough amounts of EM radiation on outputs and power supply?
Unit testing

- The unit test is a black box functional test.
- Test sequence run by test computer.
- Some tests are performed on the system using hardware.
- **Some tests are performed on the system using test software.**
Production testing

- **Unit testing**

  1. Test query
  2. Test result

**Unit under test**
Running test software

**Test computer performing test sequence**
Humans in control of industrial machinery in critical environments

Production testing

- Test computer running test sequence
  - TestStand (industrial standard)
  - Test commands sent via Ethernet or via CANbus to unit.
  - Connected to digital in/out to perform measurements directly on the hardware being tested.
Production testing

- **Test computer running test script**
  - Less advanced
  - Scripting tool for quick test sequence generation
  - Ethernet/CANbus interface only
    - Voltages may be measured, but if so they are measured by the unit under test itself.

(1 – Test script, 2 – Controls, 3 – Results pane, 4 – Log window)
Humans in control of industrial machinery in critical environments
Production testing

- Idea: 90% of faulty units will fail within the first few hours of operation.
- Adding heat helps detect failing circuits.

ESS / Burn-in tests
- Each unit is put into a heating chamber for 24 hours.
- Functional tests of external interfaces performed during this time.
- Unit restarts every half hour.
<table>
<thead>
<tr>
<th>Type of test</th>
<th>SW/HW construction test</th>
<th>Environmental tests</th>
<th>Production tests, Burn-in tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s tested</td>
<td>SW/HW design</td>
<td>SW/HW design</td>
<td>Individual HW+SW systems (units)</td>
</tr>
<tr>
<td>Goal</td>
<td>Verify requirements</td>
<td>Verify requirements</td>
<td>Screen defective units</td>
</tr>
<tr>
<td>How</td>
<td>• Voltage levels</td>
<td>• Functional tests</td>
<td>• Simple functional tests</td>
</tr>
<tr>
<td></td>
<td>• Current levels</td>
<td>during EMC screening</td>
<td>• Voltage levels</td>
</tr>
<tr>
<td></td>
<td>• Load tolerances</td>
<td>• Functional tests</td>
<td>• High temp screening</td>
</tr>
<tr>
<td></td>
<td>• Simple functional</td>
<td>in high+low temps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tests</td>
<td>• Vibration and shock</td>
<td></td>
</tr>
</tbody>
</table>
Tests so far

Are all these tests enough?
Consider a WLAN test that loads the WLAN driver, connects to the WLAN network.
- What if the WLAN driver would fail 0.1% of times, causing a kernel panic and a system reset.
  - Problem may be caused by a voltage spike in the WLAN controller circuit, causing the driver to crash.

Pop quiz: What type of tests would detect this issue?
- Code unit tests?
- Regression tests?
- Static code analysis?
- Test-driven development?
This WLAN driver issue was real, and we found it through ESS / burn-in tests.

What’s the chance at a single unit will fail?
- 1 in x chance the test fails, x = 1000.
- Number of reboots, y = 49
- \( P = 1-(\frac{x}{x+1})^y = 4.7\% \)

What’s the chance at least one unit will fail?
- 5% = 1 in 20 chance, so x = 20.
- y = 20 units in each batch
- \( P = 1-(\frac{x}{x+1})^y = 60.6\% \)
The WLAN driver problem

- ESS test was designed to screen defective units.
  - Instead we detected a general implementation flaw!
  - What can we learn from this?
The toughest test imaginable is to put your hardware or software into the hands of the end user.

- Very bad things happen when your end user discovers a flaw / a bug / a weakness in the implementation.
- Simply because it’s very much too late to do anything about it.

Idea: Simulate an end user during development

- This leads to reliability and robustness testing.
Reliability

- "Ability to perform as required, without failure, for a given time interval, under given conditions."
- Software is theoretically 100% predictable.
- Again, software running on hardware is more like 99,9% predictable.

Robustness

- "Degree to which a system can continue to operate under the presence of exceptional inputs or stressful environment."
- I won’t go into details here because it mainly concerns the hardware.
- Goal is to discover where, when and how it breaks by subjecting the hardware to extreme conditions.
## Reliability and robustness

<table>
<thead>
<tr>
<th>Type of test</th>
<th>SW/HW construction test</th>
<th>Environmental tests</th>
<th>R&amp;R</th>
<th>Production tests, Burn-in tests</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>What's tested</strong></td>
<td>SW/HW design</td>
<td>SW/HW design</td>
<td>Individual HW+SW systems (units)</td>
<td></td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>Verify requirements</td>
<td>Verify requirements</td>
<td>Screen defective units</td>
<td></td>
</tr>
</tbody>
</table>
| **How** | • Voltage levels  
• Current levels  
• Load tolerances  
• Simple functional tests | • Functional tests during EMC screening  
• Functional tests in high+low temps  
• Vibration and shock | • Simple functional tests  
• Voltage levels  
• High temp screening |
<table>
<thead>
<tr>
<th>Type of test</th>
<th>Reliability tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s tested</td>
<td>HWSW system</td>
</tr>
</tbody>
</table>
| Goal | Test beyond requirements:  
- Simulate end user testing to find hidden problems early  
- Identify design weaknesses |
| How | Long-time automated tests (set time or 1000+ cycles):  
- Performance/load tests with high/low temp  
- Power cycling  
- Reboot tests  
- Interface on/load/off tests  
- Early ESS tests |
Tests so far

**Now** are all these tests enough?

I.e. will we find all problems we want to find?
Humans in control of industrial machinery in critical environments
Humans in control of industrial machinery in critical environments

Problem occurrence

- Problem is that we use an individual unit to verify the design.
  - The problem may only manifest itself on a fraction of all units.
- If time and budget allows, test not one but several units.
Embedded software testing
Embedded Software Testing - Challenges

- How to apply software testing to near-hardware software?
  - Driver software
  - Cross-compiled software
  - Firmware
Cross-compiled software

- Cross-compilation:
  - Software is compiled and linked on another architecture
  - E.g. built on X86 but executable on ARM
  - Static code analysis still possible
Problem:
• Tests can’t run on the same system where it’s built
• Tests can’t be automatically integrated into Makefile
Possible solutions #1:

- Make application cross-platform
- Simulate the underlying hardware by creating stubs returning dummy values. Build the stubs on the host machine.
- Use common API’s wherever possible. Think of Linux as an abstraction layer.
Cross-compiled software

Possible solutions #2:

- Simulate inputs on the real hardware
- Build your software in a special test configuration where it partially runs on real hardware and partially is simulated in situ.
I/O drivers:
- Often cross-compiled, close to hardware
- Static code analysis is most useful here.
- Interface simulation is an option.
- Reliability testing of complete system is useful.
Firmware testing #1

- Firmware = microcontroller code:
  - Always cross-compiled, extremely close to hardware
  - Static code analysis is again most useful here.
  - Partial simulation of underlying hardware in situ is useful.
Unit (code) testing may be possible, using stubs and test code.

- Firmware code modules must be compiled natively and linked to stubs on the host system in order for tests to run.
Firmware testing #3

- Firmware runs in controllers
  - Controllers performs tasks

- If the task is complex:
  - How to automate testing of correct behavior?
Example 1: Simulating the hardware

Humans in control of industrial machinery in critical environments
Partial simulation in-situ

- Your software implementation
- Hardware abstraction layer (HAL) API’s
- Your HAL implementation
- Driver API
- Magic driver
- Magic
- Other component

We want to test this (with automation)

Humans in control of industrial machinery in critical environments
Humans in control of industrial machinery in critical environments

Simulating the hardware

We can only test this (with automation) by expanding or replacing the HAL with a simulation layer and compiling the application for the host computer.
Humans in control of industrial machinery in critical environments

Simulating the hardware

• Instead of this...

Component
Simulating the hardware

• You do this:

SW

Simulated component
Simulating the hardware

• Instead of compiling the software for the target architecture, you compile the software (and your tests) for the X86 build machine you are working on.

• The hardware can be replaced by simulated hardware.

• Then you can test your software using the simulated hardware.
Example 2: Partial simulation in situ
Example 2:
Partial simulation in situ

Your Software

Simulated component

Magic

Humans in control of industrial machinery in critical environments
Partial simulation in situ

- You compile for the target architecture
- Some of the hardware can be replaced by simulated hardware.
- Build a test application that executes on the target device and performs all tests
Case study: System Supervisor

• System coprocessor
• Supervises the board status
  • Temperatures
  • Voltages
  • Signals in general
    ▪ Heartbeats
    ▪ EEPROM
    ▪ ADC

• This project was built with a simulated hardware platform
Humans in control of industrial machinery in critical environments

Case study: System Supervisor

- Temp sensor
- EEPROM
- ARM µC
- System reset signal
- Warning signal
- OK signal
- Input voltages
- Output signals
- 3.3V
- GND

maximatecc
Humans in control of industrial machinery in critical environments

Case study: System Supervisor

- System reset signal
- Warning signal
- OK signal

- Input voltages
- Output signals

- How to test the software running here?
Case study: System Supervisor

• This project was built for two platforms
  • Two entirely different IDE projects and targets

- ARM IDE Project
  - real_IO.c
  - handler.c
  - main.c
  - headers.h

- VS IDE Project
  - simulated_IO.c
  - task.c
  - headers.h

Humans in control of industrial machinery in critical environments
Case study: System Supervisor

- Code uses C pre-processor directives in the HAL layer

```c
#ifdef WIN32
    #include "simulated_IO.h"
#elseif defined(ARM)
    #include "real_IO.h"
#endif

uint32_t Set_DigIO(pin_t pin, bool value) {
{
    #ifdef WIN32
        foo_simulated_Set_IO (pin, value);
    #elseif defined(ARM)
        foo_Set_IO(pin,value);
    #endif
}
```

VS IDE Project
Defines **WIN32**

ARM IDE Project
Defines **ARM**

Humans in control of industrial machinery in critical environments
Humans in control of industrial machinery in critical environments

Case study: System Supervisor

Simulator window

<table>
<thead>
<tr>
<th>Temperature</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEPROM data</td>
<td>0xDE 0xAD</td>
</tr>
<tr>
<td>Voltage #1</td>
<td>3320</td>
</tr>
<tr>
<td>Voltage #2</td>
<td>1521</td>
</tr>
</tbody>
</table>

Results window

- System reset signal
- Warning signal
- OK signal

Virtual µC

VS IDE Project
Humans in control of industrial machinery in critical environments

Case study: System Supervisor

Automated test input window

- Open script
- Start tests

Test script

Automated results window

- Test results
- OK

Virtual μC

VS IDE Project
Humans in control of industrial machinery in critical environments

Summary #1

- Testing HWSW systems
  - Functional, construction, regression, and reliability tests
- Many units must be used to verify the design
- Unit tests

1. Test query
2. Test result
Testing near-hardware software (firmware)

- **Easy:**
  - Static code analysis – just like ordinary SW
- **Harder:**
  - Everything else
  - **Simulation** of the hardware and I/O is required to test code functionality and system behavior
  - Building for two platforms is very helpful but cumbersome
  - Special test applications must be used