System Development for Embedded Linux* OS Running on Intel® Atom™ Processor

Robert Müller-Albrecht
Technical Consulting Engineer, Intel Corporation

Thorsten Möller
Product Marketing Engineer, Intel Corporation
Agenda

• Cross Development and Intel® Atom™ Processor

• System-On-Chip Challenge

• From Firmware to Operating System and Device Drivers

• Identifying and Fixing Performance Issues

• Identifying and Fixing Runtime Issues

• Summary & Questions
Many different devices and form factors

➤ Many flavors of Linux*
➤ Many screen sizes
➤ Many user interface types
➤ Many varying development requirements

Rich and varied ecosystem of Intel® Atom™ Processor powered intelligent systems
Cross Development - Targeting Intel® Atom™ Processor

- Single function device
- Custom Linux* OS
- Custom form-factor and user interface

⇒ development host ≠ development target

Flexible Requirements

- Targets: KVM, QEMU, Chroot +GUI redirection, physical device
- Target OS: Yocto Project*, Wind River Linux*, SourceForge*, MontaVista*, Custom OS based on standard distribution, and others
- Build Environment: Chroot, Sysroot (MADDE, Poky, and others)
- Debug and Deployment: USB, TCP/IP
- Flashing and System Debug:
  - JTAG (In-Target-Probe & Intel® eXtended Debug Port)

Plan the best development environment configuration for your project early
Cross-Build Flow Yocto Project*

Custom Embedded Linux* Build and Deployment
System-on-Chip Challenge

Intel® Atom™ Processor manages many execution units on SoC

**Debug Approach:** Monitor and log data flow across SoC
Instrumentation API
System level time stamps and triggers

One SoC with numerous different devices and device types
Layers of System Development

BIOS, Firmware & Bootloader

- Small hand-optimized code base
- Separate build and link step
- Bare metal code without OS dependencies
- Hex or bin file creation step (objcopy)
- Boot-NOR Flashing (JTAG, Boundary Scan, Flash Writer)

OS Kernel & Device Drivers

- Sysroot or chroot based cross-build environment
- Kernel version and kernel header dependencies
- SoC device dependencies
- High-level language debug requirement
- Interaction system and application layer
- Flashing file system, OS image (JTAG, USB software utility)

Bare-Metal or OS dependent ➔
Both require detailed platform awareness
Performance Considerations - Build

<table>
<thead>
<tr>
<th>Setting</th>
<th>GNU Compiler Collection (GCC)</th>
<th>Intel® C++ Compiler</th>
<th>Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimize performance for Atom™ Processor</td>
<td>-mtune=atom -msse2 -O3 -ffast-math</td>
<td>-xSSSE3_ATOM -O3 -no-prec-div -ipo</td>
<td>user interface, codecs, media &amp; signal processing</td>
</tr>
<tr>
<td>Optimize for memory footprint</td>
<td>-O1 + (custom libraries and linkage models)</td>
<td>-O1 + (custom libraries and linkage models)</td>
<td>resource constraints (e.g. uClibc* library)</td>
</tr>
<tr>
<td>Execution path Optimization</td>
<td>-fprofile-generate -fprofile-use</td>
<td>-prof_gen -prof_use</td>
<td>few user interactions, base station, router, manufacturing,</td>
</tr>
</tbody>
</table>

Key compiler optimizations for Intel® Atom™ Processor:

- Auto-Vectorization (-O2 or higher)
- Vectorization Report (-vec-report[n])

- Memory Access Optimization
- Load Effective Address (LEA)

- Instruction Scheduling
- Floating Point Optimizations
Key performance questions answered with *Performance Analysis*

**Where do I spend my time?**
- Hotspots
- CPU Clock Cycles
- OS Timers

**Resource Management?**
- Memory Access Latency
- Data Memory Alignment
- L1/L2 Data Cache Miss
- Instruction Cache Miss

**Coding Issues?**
- Branch Misprediction
- Data Alignment

Identify what is slowing the system down
Performance Considerations - *Tune*

**OProfile**
- System-wide processor event based sampling
- Multiple Architectures
- Eclipse* IDE Integration

```
$ opcontrol --vmlinux=/path/to/vmlinux
$ opcontrol --start
$ opreport -l foo
$ opannotate --source --output-dir=[directory] foo
$ opreport
```

**Intel® VTune™ Amplifier XE**

**Sampling Collector (sep)**
- System-wide processor event based sampling
- Local System
  - VTune™ Amplifier XE
  - Full user interface
- Remote System
  - Very lightweight command line collector (kernel module based)

```
$ ./insmod-sep3 -r -g [user_group]
$ sep -start -d 20 -info 4 -verbose -out foo
$ sfdump6 foo.tb6 –processes
```

=> View result file on host system

Low sampling overhead through event-based sampling and separation of sampling & analysis
Identify Runtime Issues - *Debug*

- Intel® Atom™ Processor supports *Branch Trace Store* (BTS) using cache-as-RAM or system DRAM
- Set breakpoint in OS signal event handler
- Unroll execution flow leading up to stack overflow or segmentation fault
- Follow execution backwards to where it deviated from expectation
- Rerun to that point and analyze memory accesses

**Unroll past execution flow:**
Where did things start to go wrong?
**Identify Memory Configuration Issues - Debug**

<table>
<thead>
<tr>
<th>Memory Issue</th>
<th>Possible Root Causes</th>
<th>Detection / Symptoms</th>
<th>Possible Resolutions</th>
<th>Intel® Architecture Exception Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bus Errors</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Memory Leaks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Segmentation Faults</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Memory Issue**

**Possible Root Causes**

- Bus Errors
  - Read/write access to non-existent memory address or page fault
  - Unaligned memory access
  - Write access to read-only memory

**Detection / Symptoms**

- SIGBUS signal raised

**Possible Resolutions**

1. Verify page table runs
2. Verify descriptor table settings

**Intel® Architecture Exception Handling**

- Uses programmable interrupt controller instead of direct memory mapped access.
- Interrupt controller entries (exception vectors) are mapped to Linux* OS signals.

**Memory Leaks**

- Faulty memory allocation and release management
  - Constantly increasing memory use
  - Can eventually lead to segmentation fault

**Possible Resolutions**

- Implement garbage collector,
- Avoid memory allocation in conditionals

**Segmentation Faults**

- A buffer overflow
- Using uninitialized pointers
- Dereferencing NULL pointers
- Attempting to access memory the program does not own
- Attempting to alter memory the program does not own (storage violation)
- Exceeding the allowable stack size (possibly due to runaway recursion or an infinite loop)

**Detection / Symptoms**

- SIGSEGV signal raised

**Possible Resolutions**

1. Verify which access triggered segmentation fault (trace, breakpoints),
2. Verify memory configuration for location, continue backtracking execution flow and memory accesses

**Intel® Architecture Exception Handling**

- Uses programmable interrupt controller instead of direct memory mapped access.
- Interrupt controller entries (exception vectors) are mapped to Linux* OS signals.

**Identify Memory Configuration Issues - Debug**

**Transparency of page table runs and descriptor table entries → Easier identification of failed memory access**
Identify Device Driver Issues - *Debug*

- Kernel module awareness of debug solution
- Notification of kernel module load
- Information about memory location of initialization method
- Access to device configuration and device data registers on SoC

<table>
<thead>
<tr>
<th>Common Device Driver Issues</th>
<th>Symptoms</th>
<th>Possible Resolutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory leaks</td>
<td>Constantly increasing memory use</td>
<td>Implement garbage collector or system-wide memory cleaner</td>
</tr>
<tr>
<td>Device mode switching failure</td>
<td>Sudden loss of device control or access</td>
<td>Correct configuration register read/write and timing</td>
</tr>
<tr>
<td>OS communication handshake</td>
<td>Device does not show as active or registered</td>
<td>Debug kernel module load and unload</td>
</tr>
<tr>
<td>Device port mapping</td>
<td>Cannot write to device, access failure</td>
<td>Remove other devices and slowly add back in. Change interface used</td>
</tr>
</tbody>
</table>

Device driver debug simplified with SoC register access
Summary

System Development for Linux* OS on Intel® Atom™ Processor

- Cross development setup similar to other architectures
- Use of chroot and UI redirection reduces need for physical cross-development setup and virtualization
- Rich ecosystem of build and debug tools for custom development needs
- Use best-known-methods for build optimization
- Performance Monitoring Unit (PMU) simplifies performance tuning of SoC bound applications
- Insight into processor and SoC architecture facilitates debug of runtime and configuration issues
Questions?
Additional Resources & References

Books

- **Embedded Intel® Architecture Chipsets Web Site**
- **Intel® Atom™ Performance for DSP Applications**, Tim Freeman and Dave Murray Birkenhead, UK, N.A. Software Ltd. 2009
  http://www.nasoftware.co.uk/home/attachments/019_Atom_benchmarks.pdf
- **JTAG 101; IEEE 1149.x and Software Debug**, Randy Johnson and Stewart Christie, Santa Clara, CA: Intel Corporation 2009

Products

- **Intel® Embedded Design Center** http://edc.intel.com
- **Intel® Embedded Software Development Tool Suite for Intel® Atom™ Processor**
  http://www.intel.com/software/products/atomtools
- **Intel® Embedded Software Development Tool Suite Forum**
- **Yocto Project** http://www.yoctoproject.org
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS
DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR
IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES
RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY
PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on
Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using
specific computer systems, components, software, operations and functions. Any change to any of
those factors may cause the results to vary. You should consult other information and performance
tests to assist you in fully evaluating your contemplated purchases, including the performance of that
product when combined with other products.

Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk
are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that
are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and
other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on
microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended
for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for
Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information
regarding the specific instruction sets covered by this notice.

Notice revision #20110804