SVEN Technology 1.0
What you will learn from this slide deck

• SVEN Technology for Linux*, Android* & Tizen™ targets

• Note: this information is general and works with all targets

• Please see subsequent slide decks for in-depth technical training on other components
<table>
<thead>
<tr>
<th>Phase</th>
<th>Component</th>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verify, Debug &amp; Flash</td>
<td>Intel® JTAG Debugger 20141</td>
<td>In-depth system and application debug</td>
<td>• In-depth debug insight into CPU, SoC and chipset for fast issue resolution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Core &amp; Intel® Xeon &amp; Intel® Quark support</td>
<td>• Leave trace instrumentation in production code for fast system-wide issue resolution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Source level debug of OS kernel software, drivers, firmware, BIOS, UEFI</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SVEN 1.0 - Ultra-fast software SoC trace infrastructure for debug</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GDB* Debugger</td>
<td>In-depth software analysis and tuning</td>
<td>• Detailed application debug and trace for fast issue resolution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Enhanced GDB* application debugger</td>
<td>• Data race detection in parallel software</td>
</tr>
<tr>
<td></td>
<td>Intel® Inspector 2014 for Systems</td>
<td>Memory &amp; threading analysis for improved code quality</td>
<td>• Increased productivity and code quality, and lowers cost, finds memory, threading, and security defects before they happen</td>
</tr>
<tr>
<td>Tune</td>
<td>Intel® VTune™ Amplifier 2014 for Systems</td>
<td>In-depth software analysis and tuning</td>
<td>• Fast in-depth analysis of SoC behavior</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SoC wide analysis</td>
<td>• Remove guesswork, saves time, makes it easier to optimize for power efficiency and find performance optimization opportunities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Power and performance profiling</td>
<td>• In-depth analysis on resource limited targets</td>
</tr>
<tr>
<td></td>
<td>Intel® Graphics Performance Analyzers</td>
<td>In-depth GPU analysis for Android*</td>
<td>• Optimize user experience of IA-based Android* devices</td>
</tr>
<tr>
<td>Build</td>
<td>Intel® C++ Compiler 14</td>
<td>Compiler and performance libraries</td>
<td>• Boost system performance for IA-based embedded designs and achieve scalability benefits of multicore and forward scale to many-core</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• High-performance C/C++ cross compiler; Intel® Cilk™ Plus threading runtime</td>
<td>• Cross platform development and integration into Eclipse for ease-of-use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Eclipse and sysroot support</td>
<td>• Speed up development &amp; performance with key software building blocks for signal, data, and media processing</td>
</tr>
<tr>
<td></td>
<td>Intel® Integrated Performance Primitives 8.1</td>
<td>A set of high-performance software building blocks for signal, media, and data processing</td>
<td>• GCC* compatibility</td>
</tr>
</tbody>
</table>

Enhance power efficiency, system reliability, and boost performance with Intel® System Studio 2014

1 Optional component
## Support for Latest Intel Processors & SoCs

<table>
<thead>
<tr>
<th>Software Suite</th>
<th>Intel® Quark SoC X1000</th>
<th>Intel® Atom™ Processors</th>
<th>Haswell microarchitecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® JTAG Debugger† - System Debug¹</td>
<td>✓</td>
<td>✓</td>
<td>--</td>
</tr>
<tr>
<td>Enhanced GDB* Debugger - Application Debug</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Inspector - Memory &amp; Threading Analysis</td>
<td>--</td>
<td>✓ Memory &amp; Thread Analysis</td>
<td>✓ Memory &amp; Thread Analysis</td>
</tr>
<tr>
<td>Intel® VTune™ Amplifier †† - Power &amp; Performance</td>
<td>--</td>
<td>✓ Hardware Events</td>
<td>✓ Hardware Events</td>
</tr>
<tr>
<td>Intel® Graphics Performance Analyzers</td>
<td>--</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® C++ Compiler</td>
<td>✓</td>
<td>✓ SSSE3</td>
<td>SSE, AVX, AVX2, FMA3</td>
</tr>
<tr>
<td>Intel® MKL library</td>
<td>--</td>
<td>--</td>
<td>SSE, AVX, AVX2, FMA3</td>
</tr>
<tr>
<td>Intel® IPP library</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

1. Hardware platform debug for new processors added as new processors ship
2. Hardware events for new processors added as new processors ship
3. optional

Copyright © 2013, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
# Intel® System Studio 2014

## What’s New?

### Broader host and target OS coverage

- New Tizen™ IVI, Embedded Android*
- New Yocto Project* 1.5; Wind River* Linux 6
- New Windows* host development for Linux*-based targets

### New Intel® Architecture

- New Intel® Quark SoC X1000-based platforms
- New support for Intel® Atom™ processor E3xxx & C2xxx series
- New support for 4th generation Intel® Core™ processor

### Enhanced Eclipse* integration & cross-build

- Automated Eclipse* IDE Integration on Linux* and Windows* hosts
- Enhanced cross-build sysroot support and Wind River* Workbench integration
- OpenEmbedded* 3rd party toolchain layer recipes

### New features across all key components

- Intel® C++ Compiler and libraries generated code compatible with Wind River Simics*
- Intel® JTAG Debugger 2014 – New support for Intel® Core™, Intel® Xeon™ & Intel® Quark processors; Agent based UEFI debug
- GNU* GDB - Branch Trace Store (btrace) for Intel® Atom™ or 4th generation Intel® Core™ Processors
- Intel® VTune™ Amplifier 2014 for Systems - Adds system wide event based sampling of uncore and memory bandwidth
# Integrated & Comprehensive Development Suite

<table>
<thead>
<tr>
<th>Accelerate Time To Market</th>
<th>Strengthen System Reliability</th>
<th>Boost Power Efficiency and Performance</th>
<th>Debuggers &amp; Trace</th>
<th>Analyzers</th>
<th>Compiler &amp; Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td></td>
<td>Intel® JTAG Debugger 2014(^1)</td>
<td>Intel® VTune™ Amplifier 2014 for Systems</td>
<td>Intel® C++ Compiler 14.0</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td></td>
<td>GDB* Debugger 7.5</td>
<td>Intel® Graphics Performance Analyzers (Android*)</td>
<td>Intel® Integrated Performance Primitives 8.1</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>Intel® Inspector 2014 for Systems</td>
<td>SVEN Technology 1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Deep system-level insights into power, reliability and performance which help accelerate time to market of Intel Architecture-based embedded and mobile systems.

\(^1\) Optional
SVEN - A Stethoscope for your System System & SoC trace through JTAG

Trace Visualization
- Advanced navigation, search & filter
- Graphical and textual event display
- User controlled trace line grouping

Smart Event Triggers
- Live JTAG system debug with event tracing
- Smart breakpoints that interrupt execution on trace event calls
- Set smart breakpoints for in-depth analysis
  For example:
  - Break on any event from the USB driver
  - Break on any Debug String that starts with "ERROR"
  - Break if register X is accessed
  - Break if register X bits [7-9] have value 0b101

Enhance system stability through powerful JTAG & event tracing
System Visible Event Nexus (SVEN) 1.0
What is it?

A ultra-fast trace recorder for SoC and CPU events

Detailed SoC & CPU System View
- Multiple cores (IA, DSP, other)
- User and kernel code

Ultra-low Overhead Sampling
- Can remain in production builds
- Around 1/10 of a microsecond
- Well defined event structure

Ultra-fast software event tracing recorder for fast issue resolution
Key Features

- **New**: JTAG debug for Intel® Core™, Xeon® & Quark SoC-based platforms
- **New**: Agent based UEFI debug
- JTAG system debug with event tracing (SVEN)
- Bitfield editor with full register description
- EFI/UEFI Firmware, bootloader debug, Linux® OS awareness
- Flashing and peripheral register support
- Access to page translation and descriptor tables
- Dynamically loaded kernel module debug
- LBR On-Chip instruction trace support, SMP run control support
Windows* Host Build and Debug Support with Eclipse* Integration

Windows* Host Eclipse* integration for SVEN, GDB and Intel® C++ Compiler
Intel® JTAG Debugger (XDB)
System Software Debug Solution

IPT Support

- EFI/UEFI Firmware Debug
- ELF Dwarf / PDB symbol info support
- Complex Software Breakpoints
- Source File Bookmarks
- Memory Layout and Page Table views
- Descriptor Table Views (GDT, LDT, IDT)
- Linux* OS awareness
  - Kernel Thread Views, Kernel Module Debug
- Hardware/Platform register access
- Hardware Threads
- LBR based Hardware trace capability support
- Advanced Scripting
- Flash Writer

System Software Debug with in-depth register and memory configuration awareness – from Firmware to OS
Debug Firmware and Bootloaders

- Debug SEC, PEI, DXE phase EFI debug support
- Source level debug for modules in flash, RAM and shadowed mode
- Tree view of firmware modules
- Automatic load of source files
- Step and set breakpoints as in any high level language debugger

JTAG assisted firmware and bootloader debug made easy
What is SVEN?

SVEN = “System Visible Event Nexus”

- Infrastructure for low overhead software event tracing
- Based on SW instrumentation using an TX-API
  - Inserted code to indicate what is happening when
- SoC/Platform scope, collects traces for
  - multiple cores (IA, DSP, ...)
  - user and kernel code

Think of SVEN as the SW analog to an airplane black box recorder
What is a SVEN event

- 32 byte in size
- Indicates good or poor health of the module
- Standardized set of events
- User defined module or API events
What is debug instrumentation?

Code inserted to indicate WHAT a component is doing:

- I just got a buffer at my input!
  - Physical address = 0xCAFEF00D
  - Timestamp = 6006
  - Buffer_id = 57

- Bad frame in stream, skipping...
  - Workload = 0xC0DED0010
  - Timestamp = 3003
  - Buffer_id = 31

- Whoops, frame DROP!
  - Physical_addr = 0xABADFEED
  - Timestamp = 140
  - Buffer_id = 13

- Audio Decode complete (ac3)
  - Timestamp = 3003
  - Buffer_id = 23
  - Num_samples = 1536
Software Instrumentation Example

```c
static void ipc_message_received(
    struct HostIPC_Handler *hipc,
    struct Host_IPC_ReceiveQueue *rcv_q,
    const struct _IPC_MessageHeader *mh,
    const char *message,
    unsigned int message_size )
{
    struct Host_IPC_QueuedMessage *msg;

    DEVH_FUNC_ENTER(hipc->devh);

    /** Queue the message for reading with HostIPC_GetNextInboundMessage() */
    DEVH_FUNC_ENTER(hipc->devh, (message_size > 0));
    DEVH_ASSERT( hipc->devh, (message_size <= CONFIG_IPC_MESSAGE_MAX_SIZE) );
    DEVH_ASSERT( hipc->devh, (mh->ipc_mh_dst_qnum < CONFIG_IPC_HOST_MAX_RX_QUEUES) );

    devh_ReadReg32( hipc->devh, CONFIG_IPC_ROFF_DOORBELL_STATUS );

    if ( NULL != (msg = HostIPC_GetFreeMessage(hipc)) )
    {
        if ( message_size > CONFIG_IPC_MESSAGE_MAX_SIZE )
            message_size = CONFIG_IPC_MESSAGE_MAX_SIZE;

        msg->mh = *mh; /* copy the header */
        memcpy( msg->msg, message, message_size );

        /* Add to inbound messages */
        OS_LIST_ADD_TAIL( &msg->node, &rcv_q->inbound_msgs );
    }
    else
    {
        DEVH_WARN( hipc->devh, "HIPC_RX_OVF" );
    }

    DEVH_FUNC_EXIT(hipc->devh);
}
```
SVEN Architecture

- Tracing infrastructure with a SW instrumentation API and visualization tools
- 3 Layer architecture
  - Event TX API for user/kernel/firmware code
  - DRAM based event ring buffer (nexus)
  - Trace receiving API and tools
  - Multiple trace writers/ multiple readers
- Very low event TX overhead
  - < .5 microseconds on Intel CE HW
  - Can remain in retail builds
- Open and extensible architecture
  - Well defined event structure
  - Independent tools development
  - Split between TX and receivers

Low overhead Event Trace infrastructure for SoC SW event tracing
SVEN SDK

- Easy adaptable SVEN device side SW stack
  - Source code distribution (no binary components)
  - SVEN kernel module
  - User mode tools like the csven trace console
  - Build system
    - Works on any Linux distribution with kernel module build environment
    - Basically just “make && sudo make install” needed for setup

- C/C++ Headers and libraries for doing SW instrumentation

- Documentation for installing and adapting to own needs

- Intel Internal version for Linux available via git
  http://wiki.ith.intel.com/display/SVEN/Accessing+the+SVEN+SDK+using+git

The SDK contains everything for SVEN enabling of a Linux platform and for starting with instrumenting code
SVEN Tools Environment

SVEN SDK enabled Linux System

CSVEN recorded Trace Data file

Trace Data in BBR format (BlackBoxRecorder)

Trace configuration File (htuple)

Trace Visualisation Layout

Upload of SVEN trace data from memory via JTAG

XDB JTAG Debugger

Trace visualisation
SVEN Debug Tools

Trace Visualization Framework

- Embeddable into JTAG Debugger and Eclipse
- Advanced navigation and search capabilities
- Graphical and textual event displays
- Extensible to other trace inputs (OMAR support available)
- User controlled trace line grouping

SVEN Event Debugger triggering

- Unique combination of life JTAG system debugging with event tracing
- Smart breakpoints that interrupt execution on SVEN event TX
- Support for any possible SVEN event from generic to very specific ones
- Examples:
  - Break on any event from the USB driver
  - Break on any Debug String that starts with “ERROR”
  - Break if register X is accessed
  - Break if register X bits [7-9] have value 0b101
SVEN Event Example, Register Read 32'

```c
/* SVEN Major Event Types */
enum SVEN_MajorEventType_t
{
   /* NO ZEROS ALLOWED */
   SVEN_EVENT_INVALID, /* 1 */
   SVEN_EVENT_TRIGGER, /* 2 */
   SVEN_EVENT_DEBUG_STRING, /* 3 */
   SVEN_EVENT_REGISTER IO, /* 4 */
   SVEN_EVENT_MODULE_ISR, /* 5 */
   SVEN_EVENT_OS_ISR, /* 6 */
   SVEN_EVENT_OS_THREAD, /* 7 */
   SVEN_EVENT_SMD, /* 8 */
   SVEN_EVENT_MODULE_SPECIFIC, /* 9 */
   SVEN_EVENT_PMU, /* 10 */
   SVEN_EVENT_PERFORMANCE, /* 11 */
   SVEN_EVENT_MAX
};

struct SVENEventTag
{
   unsigned int et_gencount : 2;
   unsigned int et_module : 10;
   unsigned int et_unit : 4;
   unsigned int et_type : 8;
   unsigned int et_subtype : 8;
};

/* Event Tag for SVEN Events, allows decoding */
/* of event owner and event type/subtype */
struct SVENEventTag
{
   unsigned int et_gencount : 2;
   unsigned int et_module : 10;
   unsigned int et_unit : 4;
   unsigned int et_type : 8;
   unsigned int et_subtype : 8;
};

/* SVEN_event_type_register_io SUBTypes */
enum SVEN_EV_RegIo_t
{
   SVEN_EV_RegIo_invalid,
   SVEN_EV_RegIo32_Read,
   SVEN_EV_RegIo32_Write,
   SVEN_EV_RegIo32_ORBits,
   SVEN_EV_RegIo32_ANDBits,
   SVEN_EV_RegIo32_SetMasked,
   SVEN_EV_RegIo16_Read,
   SVEN_EV_RegIo16_Write,
   SVEN_EV_RegIo16_ORBits,
   SVEN_EV_RegIo16_ANDBits,
   SVEN_EV_RegIo16_SetMasked,
   SVEN_EV_RegIo8_Read,
   SVEN_EV_RegIo8_Write,
   SVEN_EV_RegIo8_ORBits,
   SVEN_EV_RegIo8_ANDBits,
   SVEN_EV_RegIo8_SetMasked,
   SVEN_EV_RegIo64_Read,
   SVEN_EV_RegIo64_Write,
   SVEN_EV_RegIo64_ORBits,
   SVEN_EV_RegIo64_ANDBits,
   SVEN_EV_RegIo64_SetMasked,
   SVEN_EV_REGIO_MAX
};
```
Intel System Debugger Environment

JTAG Debugging

• Host target IO via JTAG probe (Intel XDP or Macraigor* usb2demon)
• No network/usb/Serial connection to system required
• Boot phase and kernel debugging

Macraigor* "usb2demon*"
or Intel® XDP3 probe

USB  JTAG
Linux* OS Awareness – System Debug

- Monitor kernel modules and system threads
- Access status information
- Debugging of Linux* memory images

Be aware of all relevant platform software stack interactions
SVEN Event Breakpoints in Intel Debugger

New breakpoint type based on SVEN events

• Trigger is comparison of event properties against values and mask
• Mask defined what bits in a sven event are considered for comparison

• Supports generic event properties like
  - “any event from graphics card driver”
  - “audio driver performs 32 bit write access to any register”

• and specific event properties that depend on the event kind
  - “INT_ENABLE bit is being set in register INTR”
  - “SVEN event with debug string “ABCD” written”

Workflow example on next slides
Matching SVEN instrumented code with Debugger Event Triggers

Driver code example

Clearing bit 7 in a register
//reset bit in the int_en register
//
devh_WriteReg32(handle,
ROFF_MFD_HOST_INT_EN,
regval & ~(0x00000040));

Trigger definition can be matched precisely with the corresponding source code
Intel® System Studio 2014

http://intel.ly/system-studio

The next step in Intelligent Systems Software Development
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 #: 28112204