SW Development for Altera SoC Devices Workshop
SW Workshop #1 – Altera SoC SW Development Overview
SW Workshop #2 – Introduction to Linux on Altera SoC
SW Workshop #3 – Developing Drivers for Altera SoC Linux
Welcome. Here’s What You Can Expect Today

- An SoC SW development *quick-start* to quickly evaluate the SoC tool flow
  - SoC EDS tool suite and debug capability
  - Tool chain for bare metal development
  - Take-home lab to test drive the development tools
- An overview of the SoC HW for SW developers
- An overview the SoC SW ecosystem
- A useful lead-in to the detailed SoC SW training
Agenda

- Informational Resources
- SoC Device Overview
- Boot Process
- Hardware Development Flow and Tools
- Software Development Flow and Tools
  - DS-5 Altera Edition
- SoC Physical Address Maps
- Bare Metal Software Overview
- Linux Software Overview
- Supported OSes
- Preloader Generator and u-boot
- SD Card Manipulation
- System Console
- LAB overview
Information Resources
Several Ways to Learn!

**Instructor-led training**
- Face to face with an Altera expert Training Engineer
- 20+ courses to choose from (8 hour classes)

**Virtual classes (taught via WebEX)**
- Can ask questions to Altera expert Training Engineer
- Course content same as instructor-led classes
  (1/2 day sessions)

**Online training (free and always available)**
- 200+ topics available (~30 minutes in length)

**Videos (free and always available)**
- YouTube videos (~4 minutes each)
SoC Classes Available

Instructor-led or virtual classes
- Designing with an ARM-based SoC
- Developing Software for an ARM-based SoC

Online classes
- Hardware Design Flow for an ARM-based SoC
- Software Design Flow for an ARM-based SoC
- SoC Hardware Overview: the Microprocessor Unit
- SoC Hardware Overview: Interconnect and Memory
- SoC Hardware Overview: System Management, Debug, and General Purpose Peripherals
- SoC Hardware Overview: Flash Controllers and Interface Protocols
- SoC Bare-metal Programming and Hardware Libraries
- Getting Started with Linux for Altera SoCs
Essential SoC Hardware Documentation Resources

Hard Processor System Technical Reference Manuals
- Available in Device Handbooks:
- Contain Functional Descriptions Peripheral
- Contain Control Register Address Map and Definitions
  - These are also available online at the links above in HTML and PDF formats

HPS SoC Boot Guide
- Cyclone V SoC & Arria V SoC: [AN709 - HPS SoC Boot Guide](https://www.altera.com/products/soc/portfolio)
- Arria 10 SoC: included in HPS TRM in Arria 10 Device Handbook
  - Arria 10 SoC secure booting: [AN759 – Arria 10 SoC Secure Boot User Guide](https://www.altera.com/products/soc/portfolio)

ARM Documentation Site
- Documentation available for all ARM IP
  - Cortex-A9 & A53 MP Cores, FPU, NEON, GIC, ARM Peripherals, etc.
- Requires free registration
- Refer to HPS TRM for IP core names and revision information
- [http://infocenter.arm.com/help/index.jsp](http://infocenter.arm.com/help/index.jsp)
Essential SoC Software Documentation Resources

- **Altera SoC Embedded Design Software (SoC EDS) Tools**
  - User Guide:
  - Linux & Baremetal Software Development Tools Overview
  - HPS Preloader User Guide
  - HPS Flash Programmer User Guide
  - SD Card Boot Utility
  - Getting Started Guides: Preloader, Linux, Bare Metal, Debug, HW Library
  - SoC HPS Release Notes
  - SoC Abstraction Layer (SoCAL) API Reference
    - `<SoC EDS install dir>/ip/altera/hps/altera_hps/doc/socal/html/index.html`
  - Hardware Manager API Reference
    - `<SoC EDS install dir>/ip/altera/hps/altera_hps/doc/hwmgr/html/index.html`
  - GCC Documentation
    - `<SoC EDS install dir>/ds-5/documents/gcc/getting_started.html`
  - Bare Metal Compiler
    - `<SoC EDS installation directory>/host_tools/mentor/gnu/arm/baremetal/share/doc/sourceryg++-arm-altera-eabi`
Essential SoC Software Tools Online Videos

- ARM DS-5 Altera Edition Toolchain
  - https://youtu.be/HV6NHr6gLx0

- DS-5 Altera Edition: Bare-metal Debug and Trace
  - https://youtu.be/u_xKybPhcHI

- DS-5 Altera Edition: FPGA-adaptive Linux Kernel Debug and Trace
  - https://youtu.be/lrR-SfVZd18

- Debugging Linux applications on the Altera SoC with ARM DS-5
  - https://youtu.be/ZcGQEjkYWQc

- FPGA-adaptive debug on the Altera SoC using ARM DS-5
  - https://youtu.be/2NBcUv2TxbI

- Streamline Profiling on Altera SoC FPGA. Part 1 - Setup
  - https://youtu.be/X-k9ImXQTio

- Streamline Profiling on Altera SoC FPGA. Part 2 - Running Streamline
  - https://youtu.be/Tzbd7qldKqY
SoC Device Overview
Altera Investment in Embedded Technologies

- Altera established Austin Technology Center (ATC) in 2011
- Altera’s primary embedded engineering center
- Austin provides access to one of the richest embedded processing talent bases in the world
Altera SoC Product Portfolio

LOW END SoCs (Lowest Power, Form Factor & Cost)
- 28nm TSMC
- 925 MHz Dual ARM Cortex™-A9 MPCore™
- 5G Transceivers
- 400 MHz DDR3
- 25 to 110 KLE
- Up to 224 Multipliers (18x19)

LOW POWER

MID RANGE SoCs (High Performance with Low Power, Form Factor & Cost)
- 28nm TSMC
- 1.05 GHz Dual ARM Cortex™-A9 MPCore™
- 10G Transceivers
- 533 MHz DDR3
- Up to 462 KLE
- Up to 2136 Multipliers (18x19)

HIGH END SoCs (Highest Performance & System Bandwidth)
- 20nm TSMC
- 1.5 GHz Dual ARM Cortex™-A9 MPCore™
- 17G Transceivers
- 1333 MHz DDR4
- Up to 660 KLE
- Up to 3356 Multipliers (18x19)

14nm Intel Tri-Gate
- 64-bit Quad ARM A53 MP Core™
- Optimized for Max Performance per Watt
- Over 4000 KLE

DEVICE AVAILABILITY

SoC devices available across entire product portfolio ...

© 2016 Altera—Public
# ARM Public Processor Offering

<table>
<thead>
<tr>
<th>Application Processor</th>
<th>Real-time control</th>
<th>Microcontroller</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cortex-A57</td>
<td>Stratix 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-A53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-A17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-A15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-A12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-A9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-A8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-A5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-R7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-R5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-R4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-M4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-M3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-M1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-M0+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cortex-M0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC300</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cyclone V, Arria V, Arria 10

© 2016 Altera—Public

now part of Intel
28nm SoC System Architecture

- **Processor**
  - Dual-core ARM® Cortex™-A9 MPCore™ processor
  - Up to 5,250 MIPS (1050 MHz per core maximum)
  - NEON coprocessor with double-precision FPU
  - 32-KB/32-KB L1 caches per core
  - 512-KB shared L2 cache

- **Multiport SDRAM controller**
  - DDR3, DDR3L, DDR2, LPDDR2
  - Integrated ECC support

- **High-bandwidth on-chip interfaces**
  - > 125-Gbps HPS-to-FPGA interface
  - > 125-Gbps FPGA-to-SDRAM interface

- **Cost- and power-optimized FPGA fabric**
  - Lowest power transceivers
  - Up to 1,600 GMACS, 300 GFLOPS
  - Up to 25Mb on-chip RAM
  - More hard intellectual property (IP): PCIe® and memory controllers

**Notes:**
1. Integrated direct memory access (DMA)
2. Integrated ECC
Arria 10 HPS Block Diagram

Notes:
(1) Integrated direct memory access (DMA)
(2) Integrated ECC
(3) DDR3/4 SDRAM Support for HPS Memory
# A Comparison: Cyclone V SoC, Arria V SoC, Arria 10 SoC

<table>
<thead>
<tr>
<th>Metric</th>
<th>Cyclone V SoC</th>
<th>Arria V SoC</th>
<th>Arria 10 SoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>28nm</td>
<td>28nm</td>
<td>20nm</td>
</tr>
<tr>
<td>Processor Performance</td>
<td>925 MHz</td>
<td>1.05 GHz</td>
<td>1.5 GHz</td>
</tr>
<tr>
<td>Total Power Dissipation</td>
<td>100%</td>
<td>100%</td>
<td>60% (40% Lower)</td>
</tr>
<tr>
<td>Max PCI Express Hard IP</td>
<td>Gen 2 x4</td>
<td>Gen 2 x8</td>
<td>Gen 3 x8</td>
</tr>
<tr>
<td>Memory Devices Supported</td>
<td>DDR2, DDR3, DDR3L, LPDDR2</td>
<td>DDR2, DDR3, DDR3L, LPDDR2</td>
<td>DDR4/3, LPDDR2/3, QDRIV, RLDRAM III, Hybrid Memory Cube*</td>
</tr>
<tr>
<td>Max. HPS DDR Data-Width</td>
<td>40-bit (32-bit + ECC)</td>
<td>40-bit (32-bit + ECC)</td>
<td>72-bit (64-bit + ECC)</td>
</tr>
<tr>
<td>EMAC Cores</td>
<td>EMAC x 2</td>
<td>EMAC x 2</td>
<td>EMAC x 3</td>
</tr>
<tr>
<td>NAND Device Supported</td>
<td>8-bit</td>
<td>8-bit</td>
<td>8-bit and 16-bit</td>
</tr>
<tr>
<td>SD/MMC devices supported</td>
<td>SD/SDIO/MMC</td>
<td>SD/SDIO/MMC</td>
<td>SD/SDIO/MMC 4.5 with eMMC</td>
</tr>
<tr>
<td>FPGA Logic Density Range (LEs)</td>
<td>25 - 110K</td>
<td>370 - 450K</td>
<td>160 - 660K</td>
</tr>
<tr>
<td>FPGA Core Performance</td>
<td>260 MHz</td>
<td>307 MHz</td>
<td>500 MHz</td>
</tr>
</tbody>
</table>

* Listed memory protocols are supported by FPGA EMIF interface, the HPS EMIF interface supports a subset of these.
Learn more about the HPS hardware

Overview

SoC handbooks
- Choose the device family you are interested in from the links below and then locate the “Hard Processor System Technical Reference Manual” for that device.
Boot Process
Altera SoC FPGA Boot & Configuration Options
Boot Stages – Cyclone V & Arria V

General Boot Process

1. Reset
2. BootROM
3. Preloader SPL or MPL
4. Bootloader
5. OS or Bare Metal
6. FPGA or FLASH
Boot Stages – Cyclone V & Arria V

1. Reset
Boot Stages – Cyclone V & Arria V

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
Boot Stages – Cyclone V & Arria V

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Pre-loader into Onchip RAM
1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Pre-loader into Onchip RAM
5. Run Pre-loader
6. Setup HPS I/O and DDR
Boot Stages – Cyclone V & Arria V

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Pre-loader into Onchip RAM
5. Run Pre-loader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
Boot Stages – Cyclone V & Arria V

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Pre-loader into Onchip RAM
5. Run Pre-loader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
8. Copy User Bootloader into DDR RAM
Boot Stages – Cyclone V & Arria V

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Pre-loader into Onchip RAM
5. Run Pre-loader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
8. Copy User Bootloader into DDR RAM
9. Run User Bootloader
10. Configure FPGA (optional)
Boot Stages – Cyclone V & Arria V

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Pre-loader into Onchip RAM
5. Run Pre-loader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
8. Copy User Bootloader into DDR RAM
9. Run User Bootloader
10. Configure FPGA (optional)
11. Copy OS into DDR RAM
Boot Stages – Cyclone V & Arria V

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Pre-loader into Onchip RAM
5. Run Pre-loader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
8. Copy User Bootloader into DDR RAM
9. Run User Bootloader
10. Configure FPGA (optional)
11. Copy OS into DDR RAM
12. Run OS
13. Run applications
1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Pre-loader into Onchip RAM
5. Run Pre-loader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
8. Copy User Bootloader into DDR RAM
9. Run User Bootloader
10. Configure FPGA (optional)
11. Copy OS into DDR RAM
12. Run OS
13. Run applications
14. Configure FPGA (optional)

*User Bootloader is optional. Steps 8 - 14 optional
Non-Secure Boot Stages – Arria 10

General Boot Process for Non-Secure Boot Flow
– Arria 10 supports secure and non-secure boot flows

Reset → BootROM → Bootloader u-boot or UEFI → OS or Bare Metal

FPGA or FLASH

Enabled by larger on-chip RAM in Arria 10 HPS
Non-Secure Boot Stages – Arria 10

1. Reset
Non-Secure Boot Stages – Arria 10

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
Non-Secure Boot Stages – Arria 10

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Bootloader into Onchip RAM
Non-Secure Boot Stages – Arria 10

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Bootloader into Onchip RAM
5. Run Bootloader
6. Setup HPS I/O and DDR
Non-Secure Boot Stages – Arria 10

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Bootloader into Onchip RAM
5. Run Bootloader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
Non-Secure Boot Stages – Arria 10

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Bootloader into Onchip RAM
5. Run Bootloader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
8. Copy OS into DDR RAM
Non-Secure Boot Stages – Arria 10

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Bootloader into Onchip RAM
5. Run Bootloader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
8. Copy OS into DDR RAM
9. Run OS
10. Run applications
Non-Secure Boot Stages – Arria 10

1. Reset
2. Boot from ROM
3. Setup boot source from BSEL pins
4. Copy Bootloader into Onchip RAM
5. Run Bootloader
6. Setup HPS I/O and DDR
7. Configure FPGA (optional)
8. Copy OS into DDR RAM
9. Run OS
10. Run applications
11. Configure FPGA (optional)
Learn more about boot and configuration process

HPS SoC Boot Guide
- Cyclone V SoC & Arria V SoC: AN709 - HPS SoC Boot Guide
- Arria 10 SoC: included in HPS TRM in Arria 10 Device Handbook
  - Arria 10 SoC secure booting: AN759 – Arria 10 SoC Secure Boot User Guide

Booting the HPS
- Choose the device family you are interested in from the links below and then locate the “Hard Processor System Technical Reference Manual” for that device and locate the “Booting and Configuration” chapter in that document.
Hardware Flows & Tools
Traditional System Development Flow

FPGA Design Flow

- **Design**
  - Quartus II design software
  - Qsys system integration tool
  - Standard RTL flow
  - Altera and partner IP
- **Simulate**
  - ModelSim, VCS, NCSim, etc.
  - AMBA-AXI and Avalon bus functional models (BFMs)
- **Debug**
  - SignalTap™ II logic analyzer
  - System Console
- **Release**
  - Quartus II Programmer
  - In-system Update

Software Design Flow

- **Design**
  - ARM Development Studio 5
  - GNU toolchain
  - OS/BSP: Linux, VxWorks
  - Hardware Libraries
  - Design Examples
- **Simulate**
  - Virtual Platform
- **Debug**
  - GNU, Lauterbach, DS5
- **Release**
  - Flash Programmer
So... what exactly is Qsys?

GUI based system integration tool for HW system design using IP blocks.

- Simplifies complex system development
- Raises the level of design abstraction
- Provides a standard platform:
  - IP integration
  - Custom IP authoring
  - IP verification
- Enables design re-use
- Scales easily to meet the needs of end product
- Reduces time to market
Qsys System Integration Platform

- Fastest way to build, modify, & optimize complex systems
- Flexibly optimize for performance or area
- Seamlessly integrate with Altera’s embedded solutions
Qsys connects masters and slaves together
- Configures components
- Defines memory maps
Hard Processor System Component
Learn more Qsys and HPS configuration

✿ Avalon interface specification

✿ Qsys resource page

✿ Configuring the HPS in Qsys
  - Choose the device family you are interested in from the links below and then locate the “Hard Processor System Technical Reference Manual” for that device and locate the “Instantiating the HPS Component” chapter in that document.
Software Flows and Tools
### Traditional System Development Flow

#### FPGA Design Flow

- **Design**
  - Quartus II design software
  - Qsys system integration tool
  - Standard RTL flow
  - Altera and partner IP

- **Simulate**
  - ModelSim, VCS, NCSim, etc.
  - AMBA-AXI and Avalon bus functional models (BFMs)

- **Debug**
  - SignalTap™ II logic analyzer
  - System Console

- **Release**
  - Quartus II Programmer
  - In-system Update

#### Software Design Flow

- **Design**
  - ARM Development Studio 5
  - GNU toolchain
  - OS/BSP: Linux, VxWorks
  - Hardware Libraries
  - Design Examples

- **Simulate**
  - Virtual Software Development

- **Debug**
  - GNU, Lauterbach, DS5

- **Release**
  - Flash Programmer

---

![FPGA-Adaptive Debugging Diagram](image)
Altera SoC Embedded Design Suite

Comprehensive Suite SW Dev Tools
- Hardware / software handoff tools
- Bare-metal application development
  - SoC Hardware Libraries
  - Bare-metal compiler tools
- FPGA-adaptive debugging
  - ARM DS-5 Altera Edition Toolkit
- Linux application development
  - Yocto Linux build environment
  - Pre-built binaries for Linux / u-boot
  - Work in conjunction with the Community Portal – www.Rocketboards.org

Design examples
Software Design Flow

Software Design Flow

- Standard development environment
  - ARM DS-5 and/or partner IDE

- Standard software enablement
  - HWLIBs for use with or without OS

- Standard design flow
  - No proprietary or additional tools required

Altera SoCs provide high SW developer productivity
Linux HW/SW Handoff – Cyclone V SoC and Arria V SoC

Qsys system info, SDRAM calibration files, ID / timestamp, HPS IOCSR data

Hardware

system.iswinfo

system.sopcinfo

board info

Software

Preloader Generator

Device Tree Generator

.c & .h source files (u-boot spl)

Linux Device Tree

Preloader Generator

Device Tree Generator

system.iswinfo

system.sopcinfo

board info

Linux Device Tree

Qsys system info, SDRAM calibration files, ID / timestamp, HPS IOCSR data

Hardware

Software
Linux HW/SW Handoff – Arria 10 SoC

Hardware Project → Quartus II → Handoff Folder → Board Info → Device Tree Generator: sopc2dts

Bootsloader Generator: bsp-editor → Bootloader DT Source → DTC → Bootloader DT Blob

Regenerate when HW project is recompiled

Regenerate only when user options (boot source, etc.) change

Makefile → Make → u-boot Src Code → u-boot Binary

mkpimage → u-boot Image

Board Info

Linux DT Blob

Different DTBs
DS-5 Altera Edition
ARM DS-5 Altera Edition Overview

- ARM-Altera strategic partnership with unique OEM arrangement
- Low cost and included in many SoC development kits
- Complete multi-core debug and ARM CoreSight compliant trace
- Includes ARM Pro bare-metal compiler and Altera GCC compiler
- Industry-only FPGA-Adaptive debug support
- Uses USB-Blaster cable
DS-5 Altera Edition - One Tool, Three Usages

1. JTAG-Based Debugging
   - Board Bring-up
   - OS porting, Drivers Dev,
   - Kernel Debug

2. Application Debugging
   - Linux User Space Code
   - RTOS App Code

3. FPGA-Adaptive Debugging
   - System Integration
   - System Debug

© 2016 Altera – Public
**One Device, Two Debugging Tools?**

**ARM® DS-5™ Toolkit**
- Dedicated JTAG connection
- Visualize & control CPU subsystem

**Altera Quartus™ II Software**
- Dedicated JTAG connection
- Visualize & control FPGA
Industry First: FPGA-Adaptive Debugging

ARM® Development Studio 5 (DS-5™) Altera® Edition Toolkit
- Removes debugging barrier between CPUs and FPGA
- Exclusive OEM agreement between Altera and ARM
- Result of innovation in silicon, software, and business model
Visualization of SoC Peripherals

- Register views assist the debug of FPGA peripherals
  - File generated by FPGA tool flow
  - Automatically imported in DS-5 Debugger

- Debug views for debug of software drivers
  - Self-documenting
  - Grouped by peripheral, register and bit-field

CMSIS

Peripheral register descriptions
FPGA-Adaptive, Unified Debugging

- FPGA connected to debug and trace buses for non-intrusive capture and visualization of signal events
- Simultaneous debug and trace connection to CPU cores and compatible IP
- Correlate FPGA signal events with software events and CPU instruction trace using triggers and timestamps
Cross-Domain Debug 1

Trigger from software world to FPGA world
Cross-Domain Debug 2

- Trigger from FPGA world to software world

**HARDWARE TRIGGER**

**EXECUTION STOP OR HW TRACE TRIGGER**

**EXECUTION STOP OR SW TRACE TRIGGER**
Correlate HW and SW Events

- Debug event trigger point set from either:
  - SignalTap™ II Logic Analyzer
  - DS-5 debugger

- Captured trace can then be analyzed using timestamp-correlated events
## SoC EDS Summary with DS-5 Editions Summary

<table>
<thead>
<tr>
<th>Component</th>
<th>Key Feature</th>
<th>Web Edition</th>
<th>Subscription Edition</th>
<th>30-Day Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware/Software Handoff Tools</td>
<td>Preloader Image Generator</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Flash Image Creator</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Device Tree Generator (Linux)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ARM DS-5 Altera Edition</td>
<td>Eclipse IDE</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Debugging over Ethernet (Linux)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Debugging over USB-Blaster II JTAG</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Automatic FPGA Register Views</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Hardware Cross-triggering</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>CPU/FPGA Event Correlation</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Compiler Tool Chains</td>
<td>Linaro Tool Chain (Linux)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>GCC EABI (Bare-metal)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>ARM Pro Compiler</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Hardware Libraries</td>
<td>Bare-metal programming Support</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SoC Programming Examples</td>
<td>Golden System Reference Design</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Everything needed for Linux application development is free.
Learn more about DS-5

ARM
  - http://ds.arm.com

Altera Edition

SoC EDS user guide
SoC Physical Address Map
Cyclone V & Arria V SoC HPS Physical Memory Map

L3 (Default)
- HPS Slaves
- H2F FPGA Slaves
- ACP Window
- SDRAM Region

MPU
- HPS Slaves
- H2F FPGA Slaves
- SDRAM Region
- Boot Region
- RAM/ROM/SDRAM

FPGA to SDRAM
- SDRAM

Address Ranges:
- 0x0000_0000
- 0x8000_0000
- 0xC000_0000
- 0xFF20_0000

Default remap to 0x0
Remaps as RAM & ROM or SDRAM
Arria 10 SoC HPS Physical Memory Map

- **L3 (Default)**
  - HPS Slaves
  - H2F FPGA Slaves
  - SDRAM Region
  - RAM/SDRAM

- **MPU**
  - HPS Slaves
  - H2F FPGA Slaves
  - SDRAM Region
  - Boot Region
  - Remaps as RAM/ROM or SDRAM

- **FPGA to SDRAM**
  - 4 GB
  - 3 GB
  - 2 GB
  - 1 GB
  - 0 GB
Arria 10 SoC HPS Physical Memory Map

FPGA to HPS Bridge

- HPS Slaves
- H2F FPGA Slaves
- ACP Window or SDRAM direct

MPU

- HPS Slaves
- SDRAM Region
- Boot Region RAM/ROM/SDRAM

FPGA to SDRAM

- 4 GB
- 3 GB
- 2 GB
- 1 GB
- 0 GB

ACP/SDRAM selected by AxCACHE
**Physical Address Mapping**

- FPGA Masters have access to full 4GB of SDRAM address space
  - Subject to MPFE MPU rules
  - No coherency
  - No virtual addressing
MPU can only access the lower 3GB of SDRAM
- MPU can access 960MB of FPGA address space via HPS to FPGA Bridge
- MPU can access 2MB of FPGA address space via HPS to FPGA Lightweight bridge
- See Developing Drivers for Altera SoC Linux Workshop
### Physical Address Mapping

- **FPGA to HPS (F2H)**
  - masters see 4 GB address space
    - 2GB SDRAM
    - 1GB ACP Window
    - 960MB H2F
    - 64MB HPS I/O

- F2H bandwidth to SDRAM limited vs. FPGA to SDRAM bridge
Cyclone V SoC Dev. Kit Memory Map Example
Golden Hardware Reference Design (GHRD)

Complete hardware design

- Most BSP use this design
- Simple custom logic design in FPGA
- All source code and Quartus II / Qsys design files for reference
- Includes JTAG Master(s) for System Console access.
- Similar for each Board

Diagram:

- HPS
  - ARM A9 NEON/FPU
  - DDR/SDRAM
- FPGA
  - TMC/Trace
  - USB OTG
  - Ethernet
  - SD/MMC
  - NAND flash
- LED PIO in FPGA
- HPS2FPGA LW Bridge
- JTAG Master
- SYSID
- PIO
- Memory
Cyclone V SoC Dev. Kit Memory Map Example

Default remap to 0x0

- **FPGA F2H Master**
  - HPS Slaves
  - H2F FPGA Slaves
  - ACP Window
  - Undecoded
  - SDRAM Region

- **MPU**
  - HPS Slaves
  - Undecoded
  - SDRAM Region

- **FPGA to SDRAM**
  - 4 GB
  - 3 GB
  - 2 GB
  - 1 GB
  - 0 GB

0xFFF20_0000
Memory Map through Qsys

- `led_pio` is at Address `0x0001_0040` in Qsys
Memory Map through Qsys

- It is connected to 3 Masters

<table>
<thead>
<tr>
<th>Use</th>
<th>Connections</th>
<th>Name</th>
<th>Description</th>
<th>Export</th>
<th>Clock</th>
<th>Base</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>hps_0</strong></td>
<td>Arria V/Cyclone V Hard Processor</td>
<td>Double-click</td>
<td>clk_0</td>
<td>multiple</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_axi_master</td>
<td>AXI Master</td>
<td>to clk_0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f2h_axi_slave</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_lw_axi_master</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>hps_only_master</strong></td>
<td>TAG to Avalon Master Bridge</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sysid_qsys</td>
<td>Avalon Memory Mapped Master</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>control_slave</td>
<td>System ID Peripheral</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>button_pio</strong></td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s1</td>
<td>PIO (Parallel I/O)</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>dipsw_pio</strong></td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s1</td>
<td>PIO (Parallel I/O)</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>led_pio</strong></td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click</td>
<td>clk_0</td>
<td>0x0001_0040</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s1</td>
<td>PIO (Parallel I/O)</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>onchp_memory</strong></td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s1</td>
<td>On-Chip Memory (RAM or ROM)</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>jtag_uart</strong></td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>avalon_jtag_slave</td>
<td>JTAG UART</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>fpga_only_master</strong></td>
<td>TAG to Avalon Master Bridge</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>intr_capturer_0</strong></td>
<td>Interrupt Capture Module</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>avalon_slave_0</td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s0</td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>to_ddr</td>
<td>Avalon Memory Mapped Slave</td>
<td>Double-click</td>
<td>clk_0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© 2016 Altera—Public

now part of Intel
Memory Map through Qsys

Each master sees the slave at a different address:
- HPS master addresses offset from base location of bridge slave in HPS

ARM sees LED through the H2F bridge:
H2F bridge: 0xff20_0000
+ LEDPIO base: 0x0001_0040
= 0xff21_0040

JTAG Master sees LED H2F:
LEDPIO base: 0x0001_0040
= 0x0001_0040
Altera SoC
Bare-Metal Software Development

© 2016 Altera—Public
Software For Bare-Metal Programming

- **Hardware Libs (HWLIBs)**
  - SoC Abstraction Layer (SoCAL) – Low-Level HAL
    - Header files
    - Documentation
  - Hardware Manager (HWMgr)
    - Adds C and some assembly language
    - #includes SoCAL header files

- **SoC Embedded Development Suite (EDS)**
  - IDE for development/debug of bare-metal application
  - Includes tools for flash programming, boot-image generation, and configuring and generating Preloader (Initial Program Loader)

- **Bare-Metal Compilers**
  - Altera GCC EABI Compiler
  - ARM Pro compiler (also included as of SoC EDS 14.0.2)
Hardware Libs Usage

- **NO O/S (or BM App)**
  - Simple executive
  - Polling Loop
  - Can use Bare-Metal tools:
    - SoCAL, HW Mgr, Compiler

- **Low-Level O/S**
  - No device driver model
  - i.e. ThreadX, uC/OS
  - Can use some Bare-Metal:
    - SoCAL, HW Mgr

- **High-Level O/S**
  - Device Driver Model
  - e.g. Linux, VxWorks
  - Can use some BM tools:
    - SoCAL, HW Mgr
SoCAL Overview

- Logical interface abstraction to a physical device or registers
- Low-level HAL
  - API closest to actual hardware
- Decouples software from hardware
  - Isolates client software from hardware changes
- Designed to be used with or without an OS
- Generated code
  - Preprocessor macros, enum, and struct declarations
  - Commented inline
- Targets C and assembly language programmers
- Defines programmatic access to hardware:
  - HPS address space
  - Hard IP components
  - Peripheral registers
  - Register fields
- Tested with GCC and ARM Pro bare-metal compiler tool-sets
Functional APIs that implement more complex configuration and operational control over SoC hardware resources

- Satisfy specific timing constraints
- Error checking through parameter constraints and validation checks
- Provides a level of precondition assertion checking
  - For example, checking that the FPGA is powered on and in USER mode prior to initializing an FPGA bridge
- Provides a level of post condition assurance
  - For example, if the configuration of a scan chain fails

Designed to be used with or without an OS

- Open source BSD license (non GPL)
- No namespace clash
- Thread-safe
APIs in HWLIBs HWMgr

MPU Subsystem
- Memory Map Cntl
- Address Filters
- Mem Coherence

FPGA Manager
- Full Configuration

Clock Manager
Reset Manager

SoCAL Layer (non ARM IP)

Timers
- Watchdog
- General Purpose

Bridge Management
- FPGA2HPS
- HPS2FPGA
- LWHPS2FPGA

Cache/MMU
- Cache Mgmt
- MMU Mgmt

Flash Memory
- QSPI
- NAND
- SD/MMC

Serial
- UART
- SPI
- I2C
- CAN

Minimal Preloader
- Interrupt Ctrl
- Pin I/O Cnf Mgmt
- ECC Mgmt
- Parity Mgmt

Current
Future rel.
Software Delivery

- HWLIBs and Bare-Metal application development components are included with SoC Embedded Development Suite (SoC EDS)

- SoC EDS is distributed through the Altera Download Center

- Updates for HWLIBs are also be available between SoC EDS releases
  - See Altera.com -> Download Center -> “SoC RTOS and HWLIBs Support”
HWLIB Examples

- Hello World
- UART/Interrupt processing
- FPGA Programming
- Error Correction Code (ECC)
- Web Server/Ethernet LWIP (through codetime.com)
- Minimal Preloader (MPL) – Non-GPL Preloader
- Many more examples available to download
  - https://www.altera.com/support/support-resources/design-examples.html#soc-design-examples
HWWLIB Examples – Minimal Preloader (MPL)

- Non-GPL bootloader for Cyclone V and Arria V
  - Replaces default GPL Preloader
  - UEFI-based preloader will be available for Arria 10

- Runs in Preloader boot stage
  - First programmable image loaded by bootrom

- Loads the next stage bootloader or executable image
  - Typically loads a bootloader or IPL (Initial Program Loader)
  - Can load image from QSPI Flash or SD/MMC Flash
  - Boot from FPGA option coming soon.

- Configures clocks, PLLs, memory controller, IOCSR

- Distributed as HWLIBs example starting in SoC EDS 14.0.1
Learn More

- SoC EDS user guide

- Bare metal compiler
  The bare-metal compiler comes with full documentation, located at:
  `<SoC EDS installation directory>/host_tools/mentor/gnu/arm/baremetal/share/doc/sourceryg++-arm-altera-eabi`
  The documentation is offered in four different formats to accommodate various user preferences: Html files, Info files, Man pages, PDF files

- HWLIBs and SoCAL
  Reference documentation for the SoCAL API and HW Manager API is distributed as part of the SoC EDS Toolkit. This reference documentation is provided as online HTML accessible from any web browser.
  The locations of the online SoC FPGA Hardware Library (HWLIB) Reference Documentation are:
  - SoC Abstraction Layer (SoCAL) API Reference Documentation:
    `<SoC EDS installation directory>/ip/altera/hps/altera_hps/doc/socal/html/index.html`
  - Hardware Manager (HW Manager) API Reference Documentation:
Linux for Altera SoCs

- High Quality Linux Support
- Modern release strategy
- Multiple Kernel Versions
- Community Enablement
Altera SOC Linux Provides Customers Kernel Choices

Linux for Altera SoCs supports the latest stable kernel

Linux for Altera SoCs Linux also supports latest LTSI kernel

LTSI kernel available with or without Preempt Real Time Patches
Industry-leading Linux support

- Altera keeps up with the Linux community
  - Kernel is upgraded every 3 months against every new kernel.org release
  - Altera is the maintainer for the SoCFPGA architecture folder (mach-socfpga)

- Altera maintains the LTSI kernel
  - Altera SOC Linux LTSI v3.10, more stable code base for 24 months
  - Fixes, improvements and new features back-ported from latest kernel
  - Seamless transition to commercial Linux vendors, Wind River, etc.

- Altera’s SoC drivers have high quality
  - Altera SoC Hard IP acquired from EDA vendors, use community Linux drivers
  - Altera SoC Linux drivers used by many SOC vendors, therefore drivers have more support and higher quality
  - Altera upstreams fixes and improvements to code to the kernel and u-boot

- Altera supports a modern release strategy
  - Updates Public GIT trees every 2 weeks – complete transparency
  - Published on community site: See git.rocketboards.org
  - See the NEWS section of RocketBoards.org latest updates
Altera SoC Linux Support Model

Rocketboards.org
- SoC & Nios II Linux documentation
- SoC & Nios II SoC Linux reference & example designs

Rocketboards.org RFI & Linux Community
- Kernel/RFS/u-boot questions
- SoC/Nios II subsystem and driver questions

Altera.com and Rocketboards.org
- SoC EDS & Quartus/QSys documentation and questions
- SoC Preloader questions
- SoC HPS implementation specific questions
- Use myAltera for service requests

Support from Altera is focused on SoCFPGA and NiosII Linux Board Support Package

Altera enables Linux development on SoCFPGA & Nios II
Linux Resources
Linux Documentation Resources

**GIT**
- Distributed revision control system to enable distributed collaboration
- On-line documentation & training:
  - [http://git-scm.com/doc](http://git-scm.com/doc)
  - [https://training.github.com](https://training.github.com)

**Denx u-boot Manual**
- Complete documentation from the folks who wrote Das u-boot

**Free-Electrons:**
- Complete training materials posted free

**Device Tree for Dummies**
Linux Documentation Resources

- **Yocto Project**
  - [https://www.yoctoproject.org/documentation](https://www.yoctoproject.org/documentation)

- **Angstrom Distribution**

- **Open Embedded**
  - [http://www.openembedded.org/wiki/Main_Page](http://www.openembedded.org/wiki/Main_Page)
The Two Best Sources for Linux Development Information

Linux Kernel Documentation
- The most complete and most essential Linux kernel documentation
- Included with the Linux kernel source code
  - <local GIT repo>/Documentation

```
/data/linux_repos/linux-socfpga/documentation> grep -ir tickless *
timers/highres.txt: The implementation leaves room for further development like full tickless
timers/NO_HZ.txt: tickless*. The remainder of this document will use "dyntick-idle mode".
```

```
NO_HZ.txt - KWrite

NO_HZ: Reducing Scheduling-Clock Ticks

This document describes Kconfig options and boot parameters that can reduce the number of scheduling-clock interrupts, thereby improving energy efficiency and reducing OS jitter. Reducing OS jitter is important for some types of computationally intensive high-performance computing (HPC) applications and for real-time applications.

There are three main ways of managing scheduling-clock interrupts (also known as 'scheduling-clock ticks' or simply "ticks"):

1. Never omit scheduling-clock ticks (CONFIG_HZ_PERIODIC=y or CONFIG_NO_HZ=n for older kernels). You normally will not want to choose this option.
```
An open source OS breeds open source information.

The Two Best Sources for Linux Development Information

Embedded Linux boot time reduction - Free Electrons
free-electrons.com/services/boot-time/
Making your embedded Linux systems boot faster: Investigating boot time issues and applying optimization techniques that don't require a redesign.

Free Electrons: Embedded Linux Experts
free-electrons.com/
Embedded Linux, kernel and Android: development, training and consulting services. ... Buildroot commercial support: Embedded Linux boot time reduction ... Offering our broad embedded Linux development experience through our ... 42,295 views; How to boot an uncompressed Linux kernel on ARM - 38,199 views ....

[PDF] Update on boot time reduction techniques, with figures - T...
events.linuxfoundation.org/.../open HACKER-boot-time... Linux Foundation
That's where you will find extensive...
RocketBoards.org – Altera SoC Linux Community Portal

- The source for SoC FPGA Linux info
  - Golden System Reference Design (GSRD)
  - Updates on latest releases
  - Step-by-step getting started guides

- SoC FPGA Mailing List - RFI
  - Active community participation in answering SoC FPGA and Linux questions

- Example Projects, Applications, and Designs
  - From Altera and the SoC community

- Enables the SoC community to support Linux
Learn More about SoCFPGA Linux

- SW Workshop #1 – Altera SoC SW Development Overview
- SW Workshop #2 – Introduction to Linux on Altera SoC
- SW Workshop #3 – Developing Drivers for Altera SoC Linux
Altera SoC
Operating System Support
## Embedded OS Availability

<table>
<thead>
<tr>
<th>Vendor</th>
<th>OS/RTOS</th>
<th>Development Tools</th>
<th>Available From</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Source Linux</td>
<td>Linux (current and 3.10 LTSI)</td>
<td>Linaro compiler</td>
<td>rocketboards.org</td>
</tr>
<tr>
<td>Wind River Systems VxWorks 6.9.3 and 7.0</td>
<td>Wind River Workbench</td>
<td>Wind River</td>
<td></td>
</tr>
<tr>
<td>Micriµm µC/OS-II, µC/OS-III</td>
<td>GNU compiler</td>
<td>Micriµm</td>
<td></td>
</tr>
<tr>
<td>Enea</td>
<td>OSE 5.5.3</td>
<td>Optima 2.6</td>
<td>ENEA</td>
</tr>
<tr>
<td>Express Logic ThreadX G5.5.5.0</td>
<td>GNU compiler</td>
<td>Express Logic</td>
<td></td>
</tr>
<tr>
<td>Wind River Systems Wind River Linux 5 and 7</td>
<td>Workbench/GNU</td>
<td>Wind River</td>
<td></td>
</tr>
<tr>
<td>QNX QNX/Neutrino 6.5.3 and 6.6</td>
<td>Momentics</td>
<td>QNX</td>
<td></td>
</tr>
<tr>
<td>Fujisoft Android</td>
<td>GNU compiler</td>
<td>Fujisoft</td>
<td></td>
</tr>
<tr>
<td>Green Hills INTEGRITY</td>
<td>Multi/Green Hills</td>
<td>Green Hills</td>
<td></td>
</tr>
<tr>
<td>DDC-I Deos</td>
<td>DDC-I</td>
<td>DDC-I</td>
<td></td>
</tr>
<tr>
<td>Code Time Multicore Abassi</td>
<td>ARMCC/GCC</td>
<td>Code Time</td>
<td></td>
</tr>
<tr>
<td>Mentor Nucleus</td>
<td>GCC</td>
<td>Mentor</td>
<td></td>
</tr>
<tr>
<td>eCosCentric ECOSPRO (eCos)</td>
<td>GCC</td>
<td>eCosCentric</td>
<td></td>
</tr>
<tr>
<td>Vendor</td>
<td>OS/RTOS</td>
<td>Development Tools</td>
<td>Available From</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------</td>
<td>------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>MRA Digital</td>
<td>Android</td>
<td>GCC</td>
<td>MRA Digital</td>
</tr>
<tr>
<td>FreeRTOS</td>
<td>RTE</td>
<td>ARM DS-5 and GCC</td>
<td>Freertos.org</td>
</tr>
<tr>
<td>Monta Vista</td>
<td>CGE7 Linux</td>
<td>Monta Vista/GCC</td>
<td>Monta Vista</td>
</tr>
<tr>
<td>AUTOSAR</td>
<td>AUTOSAR 4.0.3 MCAL</td>
<td>Elektrobit Tresos Studio</td>
<td>Altera</td>
</tr>
<tr>
<td>Microsoft</td>
<td>Windows Embedded 7</td>
<td>Microsoft/Studio</td>
<td>Coming</td>
</tr>
<tr>
<td>Quadros</td>
<td>RTXC</td>
<td>GCC</td>
<td>Coming</td>
</tr>
<tr>
<td>rtems.org</td>
<td>RTEMS</td>
<td>GCC</td>
<td>Coming</td>
</tr>
</tbody>
</table>
# Embedded SW Operating System Ecosystem (Japan)

<table>
<thead>
<tr>
<th>Vendor</th>
<th>OS/RTOS</th>
<th>Development Tools</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSOL</td>
<td>eT-Kernel (uITRON4.0)</td>
<td>eBinder</td>
<td>eSOL</td>
</tr>
<tr>
<td>eForce</td>
<td>uC3 (uITRON 4.0)</td>
<td>ARMCC/SoCEDS</td>
<td>eForce</td>
</tr>
<tr>
<td>Toppers</td>
<td>Toppers (uITRON extended)</td>
<td>EDS/Toppers</td>
<td>Toppers</td>
</tr>
<tr>
<td>Mispro</td>
<td>NORTi(uITRON 4.0 standard)</td>
<td>ARMCC</td>
<td>Mispro</td>
</tr>
</tbody>
</table>
# Broad JTAG Debugging Tools Support

<table>
<thead>
<tr>
<th>Company</th>
<th>Debugger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera</td>
<td>USB Blaster II</td>
</tr>
<tr>
<td>Lauterbach</td>
<td>Trace32</td>
</tr>
<tr>
<td>ARM</td>
<td>DSTREAM</td>
</tr>
<tr>
<td>Wind River</td>
<td>ICE II, Probe</td>
</tr>
<tr>
<td>Green Hills</td>
<td>Probe</td>
</tr>
<tr>
<td>Yokogawa Digital Computer</td>
<td>AdviceLUNA</td>
</tr>
<tr>
<td>Kyoto Microcomputer</td>
<td>Partner-Jet</td>
</tr>
<tr>
<td>Computex</td>
<td>PALMiCE3</td>
</tr>
<tr>
<td>Segger</td>
<td>J-Link</td>
</tr>
<tr>
<td>iSystem</td>
<td>Coming Soon</td>
</tr>
<tr>
<td>Ronetix</td>
<td>PEEDI</td>
</tr>
</tbody>
</table>
OS support

List

Preloader Generation

For Cyclone V and Arria V SoC
Preloader

- Runs after the boot ROM
- Configures the HPS IO
- Configures and Calibrates the DDR controller
- Optionally loads the FPGA image
- Is just a small version of u-boot called u-boot-spl
Generating the preloader

(GUI and Command line options)
- [Link](http://www.rocketboards.org/foswiki/Documentation/GSRD131Preloader)

Launch in embedded shell
- "bsp-editor &"

Choose File -> New

Select the folder under the handoff folder
bsp-editor - Choose Soc System

Select OK
Bsp-editor - Choose Next Boot location

Choose the Device that contains the next executable. Note: Only select one.
Preloader - Advanced Settings

Choose the advanced settings you may want.

i.e. you may not want watchdog for bare metal system.

Select Generate

Then at the embedded prompt cd to the BSP directory and type “make”
Build Preloader from the command line

```bash
make -C $(PRELOADER_DIR)
```

```bash
bsp -create --bsp-dir "./qspi_preloader" --preloader-settings-dir "$(SOCEDS_ROOT)/examples/hardware/cv_soc_devkit_grhd/hps_isw_handoff/soc_system_hps_0" --settings "./qspi_preloader/settings.bsp" --type spl --set spl.CROSS_COMPILE arm-altera-eabi --set spl.PRELOADER_TGZ "$(SOCEDS_ROOT)/host_tools/altera/preloader/uboot-socfpga.tar.gz" --set spl.boot.BOOTTROM_HANDSHAKE_CFGIO 1 --set spl.boot.BOOT_FROM_QSPI 10 --set spl.boot.BOOT_FROM_RAM 0 --set spl.boot.BOOT_FROM_SDMC 1 --set spl.boot.CHECKSUM_NEXT_IMAGE 1 --set spl.boot.EXE_ON_FPGA 0 --set spl.boot.FPGA_DATA_BASE 0xffff0000 --set spl.boot.FPGA_MAX_SIZE 0x10000 --set spl.boot.QSPI_NEXT_BOOT_IMAGE 0x60000 --set spl.boot.SDMMC_NEXT_BOOT_IMAGE 0x60000 --set spl.boot.SDMMC_NEXT_BOOT_IMAGE 0x40000 --set spl.boot.STATE_REG_ENABLE 1 --set spl.boot.WARMRST_SKIP_CFGIO 1 --set spl.boot.WATCHDOG_ENABLE 0 --set spl.debug.DEBUG_MEMORY_ADDR 0xfffffd00 --set spl.debug.DEBUG_MEMORY_SIZE 0x200 --set spl.debug.DEBUG_MEMORY_WRITE 0 --set spl.debug.HARDWARE_DIAGNOSTIC 0 --set spl.debug.SEMIHOSTING 0 --set spl.debug.SKIP_SDRAM 0 --set spl.performance.SERIAL_SUPPORT 1 --set spl.reset_assert.DMA 0 --set spl.reset_assert.GPIO0 0 --set spl.reset_assert.GPIO1 0 --set spl.reset_assert.GPIO2 0 --set spl.reset_assert.L4WD1 0 --set spl.reset_assert.OSCTIMER1 0 --set spl.reset_assert.SDR 0 --set spl.reset_assert.SPTIMER0 0 --set spl.reset_assert.SPTIMER1 0 --set spl.warm_reset_handshake.ETR 1 --set spl.warm_reset_handshake.FPGA 1 --set spl.warm_reset_handshake.SDRAM 0 --set spl.boot.SDRAM_SCRUBBING 1 --set spl.boot.SDRAM_SCRUB_BOOT_REGION_START 0x1000000 --set spl.boot.SDRAM_SCRUB_BOOT_REGION_END 0x3000000 --set spl.boot.SDRAM_SCRUB_REMAIN_REGION 1
```
Preloader more info

SoC EDS Users guide – Chapter 7

Rocketboards
- http://www.rocketboards.org/foswiki/Documentation/PreloaderUbootCustomization131
u-boot Generation
u-boot

- Linux workshop – WS2
- Altera SoC 2 day training
- u-boot can be downloaded from GitHub.org…
  - https://github.com/altera-opensource
  - http://www.rocketboards.org/foswiki/Documentation/GitGettingSTarted
- …or built along with the preloader
  - Type “make uboot” in the preloader directory and u-boot will be created
Working with GSRD SD cards
Creating an SD card from GSRD on Linux Host

The steps required to create the SD card for the Cyclone V Development board are:

1. Download the GSRD release binaries

2. Extract the compressed Linux SD card image from archive
   - $ tar -xzf linux-socfpga-gsrd-14.1-cv-bin.tar.gz

3. Expand the compressed Linux SD card image
   - $ gunzip linux-socfpga-gsrd-14.1-cv-bin/sd_image_cyclone5.bin.gz

4. Determine the device associated with the SD card on the host by running the following command before and after inserting the card in the reader:
   - $ cat /proc/partitions
   - Let's assume it is /dev/sdx.

5. Use dd utility to write the SD image to the SD card:
   - $ sudo dd if=linux-socfpga-gsrd-14.1-cv-bin/sd_image_cyclone5.bin of=/dev/sdx bs=1M
   - Note we are using sudo so we can access the device node
Creating an SD card from GSRD on Windows Host

- Download and unzip the sd-image
- Insert an micro-SD card in a microSD->USB adapter
- A new drive letter will show up, assuming F:
- Run Win32DiskImager
- Select the GSRD sd_image for your board
- Click 'write' and confirm 'yes'
- Wait for the completion
- Eject the micro-SD card
## Partition Layout and Contents

**http://www.rocketboards.org/foswiki/Documentation/GSRD141SdCard**

<table>
<thead>
<tr>
<th>Location</th>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition 1</td>
<td>socfpga.dtb</td>
<td>Device Tree Blob file</td>
</tr>
<tr>
<td></td>
<td>soc_system.rbf</td>
<td>FPGA configuration file</td>
</tr>
<tr>
<td></td>
<td>u-boot.scr</td>
<td>u-boot script for configuring FPGA</td>
</tr>
<tr>
<td></td>
<td>zImage</td>
<td>Compressed Linux kernel image file</td>
</tr>
<tr>
<td>Partition 2</td>
<td>various</td>
<td>Linux root filesystem</td>
</tr>
<tr>
<td>Partition 3</td>
<td>n/a</td>
<td>Preloader image</td>
</tr>
<tr>
<td></td>
<td>n/a</td>
<td>u-boot image</td>
</tr>
</tbody>
</table>

**Diagram**

- **Unused**
- **Partition 3**
  - Type=A2 (raw)
- **Partition 2**
  - Type=83 (EXT Linux)
- **Partition 1**
  - Type=B (FAT32 Windows)
- **U-boot Environment Settings**
- **Master Boot Record (MBR)**
Updating an SD Card Linux

The following table presents how each item can be updated individually. Replace "sdx" in the command below with the device name of the SD card on your host system. You can find out the device name by running “$ cat /proc/partitions” before and after plugging in the card reader into the host.

<table>
<thead>
<tr>
<th>File</th>
<th>Update Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>zImage</td>
<td>Mount /dev/sdx1 (FAT) on the host machine and update files accordingly: $ sudo mkdir sdcard</td>
</tr>
<tr>
<td>soc_system.rbf</td>
<td>$ sudo mount /dev/sdx1 sdcard/</td>
</tr>
<tr>
<td>soc_system.dtb</td>
<td>$ sudo cp &lt;file_name&gt; sdcard/</td>
</tr>
<tr>
<td>u-boot.scr</td>
<td>$ sudo umount sdcard</td>
</tr>
<tr>
<td>preloader-mkpimage.bin</td>
<td>$ sudo dd if=preloader-mkpimage.bin of=/dev/sdx3 bs=64k seek=0</td>
</tr>
<tr>
<td>u-boot-socfpga_cyclone5.img</td>
<td>$ sudo dd if=u-boot-socfpga_cyclone5.img of=/dev/sdx3 bs=64k seek=4</td>
</tr>
<tr>
<td>root filesystem</td>
<td>Mount /dev/sdx2 (ext3 FS) on the host machine and update files accordingly</td>
</tr>
</tbody>
</table>
Updating an SD Card on Windows or Linux Hosts

SD Card Boot Utility
- Useful for updating an existing sd card image
- SoC EDS user guide – Chapter 11

```
$ alt-boot-disk-util
Altera Boot Disk Utility
Copyright (C) 1991-2014 Altera Corporation

Usage:
#write preloader to disk
  alt-boot-disk-util -p preloader -a write disk_file

#write bootloader to disk
  alt-boot-disk-util -b bootloader -a write disk_file

#write BOOTloader and PREloader to disk
  alt-boot-disk-util -p preloader -b bootloader -a write disk_file

#write BOOTloader and PREloader to disk drive 'E'
  alt-boot-disk-util -p preloader -b bootloader -a write -d E
```
System Console
System Console

- Provides direct system access at runtime from TCL command line interface
- No processor code to write
  - Access via Nios® II processor, JTAG bridge, JTAG UART or custom Agent
  - Supports an Avalon-MM or Avalon-ST Master
  - Use for board bring-up, debug, diagnostics, system profiling and configuration
What is System Console?

Quick system-level debug of Qsys systems

- Interactive Tcl Console
  - Opens as a separate window
  - Launched from the Embedded Command Shell
    - “system-console”
- Dashboard components available.
- Debug over various communication channels
  - JTAG or TCP/IP
- Read or write memory mapped components
- Sink or source streaming data
- No processor required
Usage Examples

System-level debug

- Board bring-up and interface testing
- System clock, reset and JTAG chain validity testing
- Qsys component functionality testing
- Loopback testing of Avalon Streaming interfaces
- Inject test vectors, retrieve responses
- Peek and Poke at the address map.
System Console Interfaces

The System Console on Host is connected to the Qsys Interconnect through JTAG and Virtual JTAG Hub. The following components are involved:

1. **Nios II Processor**
   - Avalon-MM Master
2. **JTAG to Avalon Master Bridge**
   - Avalon-MM Master
3. **JTAG UART**
   - Avalon-MM Slave
4. **Avalon-ST JTAG Interface**
   - Avalon-ST Source and Sink

These components interact through Avalon-MM Master and Slave interfaces.
Usage Flow – Summary

1. Add required component to Qsys
2. Connect board and program FPGA
3. Launch System Console
4. Locate and open service path
5. Perform desired operation(s) with the service
6. Close the service

Complete master write and read example script

```bash
set m_path [lindex [get_service_paths master] 0]
open_service master $m_path
master_write_memory $m_path 0x2000 [list 0x01 0x02]
master_read_memory $m_path 0x2000 2
close_service master $m_path
```
JTAG Masters in GHRD

- GHRD has multiple JTAG Masters
  - To HPS
  - FPGA slaves
  - To SDRAM (optional)
System Console Resources

» System Console home page

» On alterawiki.com

» Online training courses available.
Creating FPGA specific header files for software development
Creating FPGA-Specific Header Files

“sopc-create-header-files <project>.sopcinfo”

- Creates a header file for each master in the Qsys system.
- Use this in your software flow to keep everything updated.
- Call from an embedded command shell or in your own make file.
- Hardware changes made in Qsys get generated into the SOCINFO file and this utility extracts that information into C header files for your software project.
- `sopc-create-header-files` is installed in with the ACDS tool chain.
Labs Overview
LABs Overview

- FPGA hardware already built
  - Based on GHRD
  - Added FFT megacore
  - Added 2 SGDMA to move the data in and out of the FFT
  - Added on chip memory to make transfers easy.
  - Self contained in its own Qsys subsystem.
  - LW bridge and JTAG Master connect to all FFT slaves

- Need Quartus, Cyclone V devices and SoC EDS loaded on your machine.

- No licenses are required for these LABs

- *If you recompile the hardware in Quartus you will need a full Quartus license or 60 day evaluation license.*
FFT Implementation Details

FFT Component is Composed of two items
- The FFT megacore
- An adapter to wrap it with standard Avalon-ST interfaces

Inputs are 16-bit real and 16-bit imaginary so it would make a nice 32-bit word.
- They could be larger, but then you would just make the real and imaginary their own 32-bit words.

Output is 24-bit of real and 24-bit of imaginary data
- So map it into 64 bits (2 x 32 bit words);
1- Processor Writes waveform to Onchip Memory.
1. Processor Writes waveform to Onchip Memory.
2. Processor Writes Descriptor to SGDMAs to start data flowing.
1- Processor Writes waveform to Onchip Memory.
2- Processor Writes Descriptor to SGDMAs to start data flowing.
3- SGDMAs move data through the FFT.
1- Processor Writes waveform to Onchip Memory.
2- Processor Writes Descriptor to SGDMAs to start data flowing.
3- SGDMAs move data through the FFT.
4- Processor reads the FFT data back.
JTAG Master also has access to all these peripherals so System Console can run the FFT
LABs

LAB 1 – Generate and Build a Preloader
LAB 2 – System Console, FFT Hardware Manipulation
LAB 3 – BareMetal Application, FFT Hardware Manipulation
LAB 4 – Linux Application, FFT Hardware Manipulation
LAB 1 – Generate and Build a Preloader

- Load SD card with a default image.
- Build Preloader from prebuilt hardware.
- Load new Preloader into SD Card
- Boot target with new Preloader
- Restore original Preloader
LAB 2 – System Console, FFT Hardware Manipulation

Use the FPGA JTAG Master to load the input sample waveforms, trigger the SGDMAs to process the FFT and retrieve the resultant waveforms (no processor involvement required)

- Open system_console
- Load Design SOF
- Run fft dashboard script
- Observe controls and waveforms in dashboard
- This essentially validates the hardware
LAB 3 – BareMetal Application, FFT Hardware Manipulation

- Review code
- Use the alt_read_word, alt_write_word to insure the data does not get cached by the processor
- Compile in an embedded_command shell
  - Paths to the hwlibs, socal and compilers
- Load code onto SD card and run from u-boot
- Program the Preloader from Lab 1 and the bare metal application into 0xA2 partition to observe auto boot into the bare metal application.
Linux requires that a driver must be present to talk to hardware.

There are at least three drivers that can be used:
- /dev/mem, ready to go out of the box
- uio driver, some assembly required
- Custom written driver, you own it

Not a Linux class so we use the simplest /dev/mem.

Linux also uses the MMU so all address are virtual.

Linux drivers discussed in detail in Developing Drivers for Altera SoC Linux Workshop.
LAB 4 – Linux Application, FFT Hardware Manipulation

- Build the Linux application by running the make file
- Copy the fpga_fft application over to the target
- Run the application on the target
- Observe the FFT input and output data graphically through the target based web server:
  - Two files fft.sh and MyGraphTest1_2.js facilitate this on the target
  - fft.sh is the initial script that is run when you type the below in your browser
    - `<ipaddress>/fft.sh`
  - MyGraphTest1_2.js is a JavaScript which plots the FFT input and output and runs in the browser
  - When fpga_fft is runs, it produces output files with the input samples and output result values of the FFT.
  - This JavaScript code reads it in and plots the graph.
Thank You