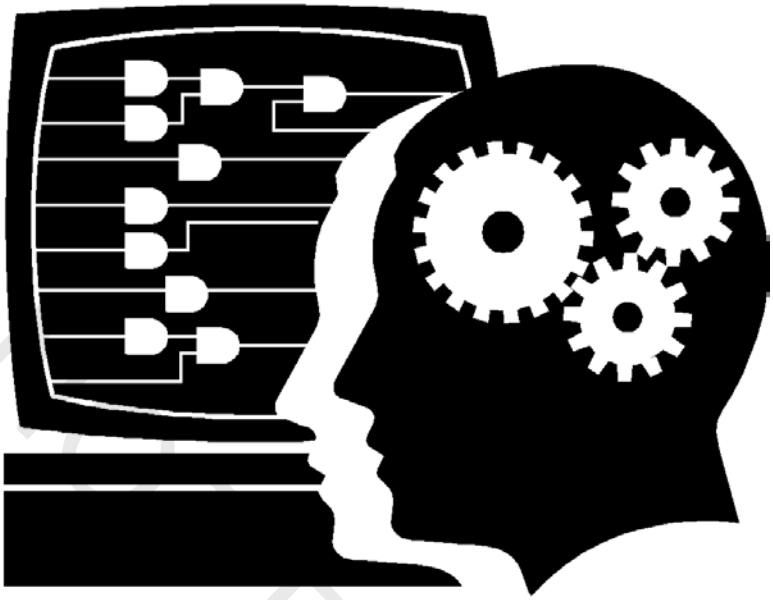


Customer Training

Designing with an ARM-based SoC

A-MNL-HW-SoC-15-0-v2

http://www.altera.com/customertraining/ILT/P-CSTN-HW-SOC-15-0-v2.zip

Table of Contents Designing with an ARM-based SoC	PAGES

Designing with an ARM-based System on a Chip

Objectives

- Explain the components that make up the SoC
- Create a Hard Processor System based Qsys system
- Describe the hardware to software file handoff
- Explain the Avalon® and AXI™ interface protocols
- Simulating an HPS-based Qsys system
- Debug an SoC with various Quartus® II tools

Agenda

- System on a Chip (SoC) overview
- Hard Processor System (HPS) overview
- Hardware Design
  - Hardware design flow
  - Avalon/Advanced eXtensible Interface (AXI) protocols
  - HPS configuration
  - SoC system simulation
  - SoC hardware system debug

Designing with an ARM-based System on a Chip

SoC Overview

Innovation Leader Across the Board

ALTERA

MAX	Cyclone FPGA • SoC	Arria FPGA • SoC	Stratix FPGA • SoC	Enpirion Power Solutions
CPLDs *Lowest Cost, Lowest Power*	***FPGAs*** *Cost/Power Balance SoC & Transceivers*	***FPGAs*** *Mid-range FPGAs SoC & Transceivers*	***FPGAs*** *Optimized for High Bandwidth*	***PowerSoCs*** *High-efficiency Power Management*

RESOURCES

Embedded Soft and Hard Processors

Nios II

ARM

Design Software

QUARTUS II

DSPBuilder

Development Kits

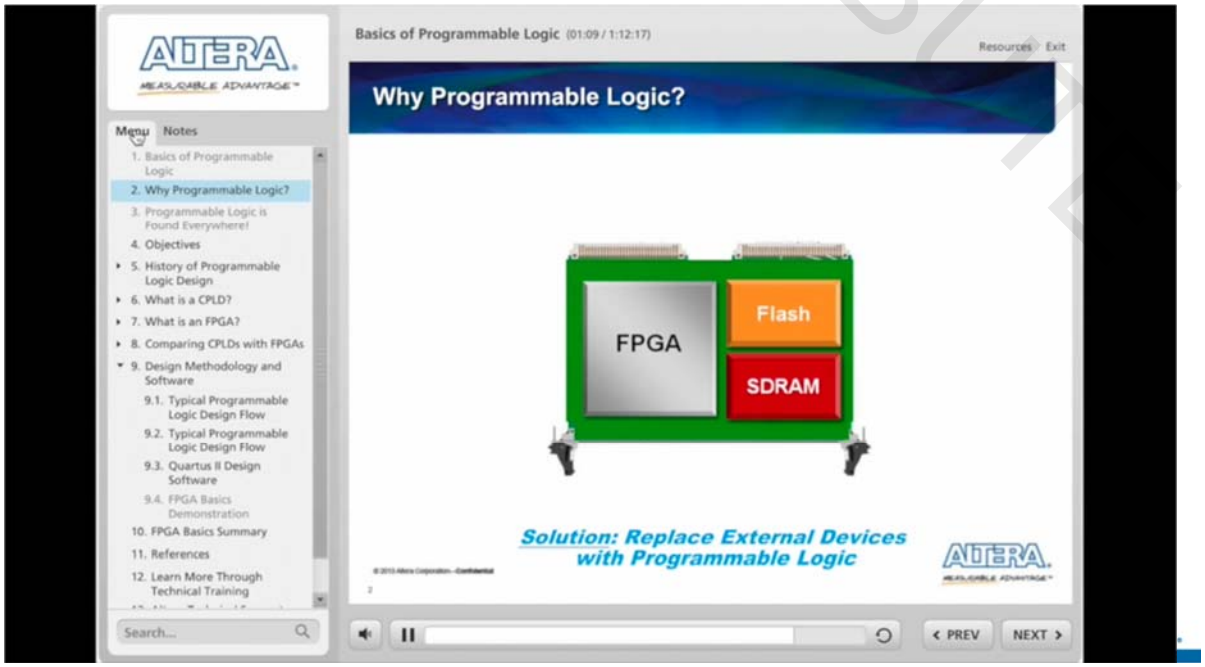
Intellectual Property (IP)

- Industrial
- Computing
- Enterprise

ALTERA

Register for Free Online Training!

- Always available on all your devices!
- Always free!



Altera & Silicon Convergence

General processors	SoC	Application-specific
µP, DSP	µP, ASIC, FPGA, DSP, ASSP	ASIC, ASSP
Need for Efficiency »		« Need for Flexibility
■Software programmable ■Great flexibility ■Poor power efficiency ■Few application specific features	■Hardware programmable ■Great flexibility ■Good power efficiency	■Hard-wired, not programmable ■Poor flexibility ■Great power efficiency ■Many contain embedded processors

ARM Cortex-A9 processor
Dedicated DSP blocks
Application-Specific IP
Programmable Logic Fabric

Altera SoC: The Best of Both Worlds

ARM

ARM Processor System
Dual Core ARM Cortex-A9 MPCore Processor
Hard Memory Controller
Peripherals

ALTERA

Cyclone V
Arria V
Arria 10

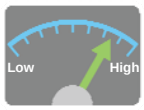
ALTERA
SoC

ARM Cortex-A9 + Altera FPGA

Quick Summary

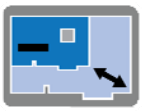
- FPGA:
  - Looks like an FPGA
  - Works like an FPGA
  - Standard FPGA development flow
  - Standard FPGA development tools
    - Quartus II, Qsys, SignalTap™ II Logic Analyser, System Console, USB-Blaster™, Programmer...
- ARM® HPS:
  - Looks like an ARM processor system
  - Works like an ARM processor system
  - Typical ARM processor development flow
  - Typical ARM processor development tools
    - ARM Cortex®-A9 compiler/debugger, JTAG tools, program trace...

System-Level Benefits of SoC



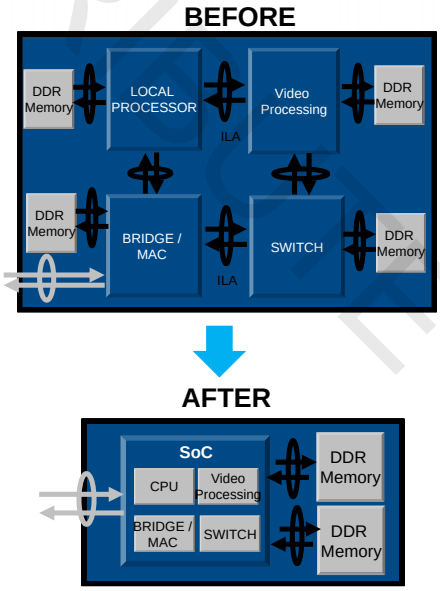
Increased system performance
- Over 4,000 DMIPS for under 1.8W
- Up to 1,600 GMACS, 300 GFLOPS DSP
- >120 Gbps processor to FPGA interconnect
- Cache coherent hardware accelerators

Reduced power consumption
- 28nm & 20nm process (processor+FPGA)
- Significant power savings vs. 2-chip solution

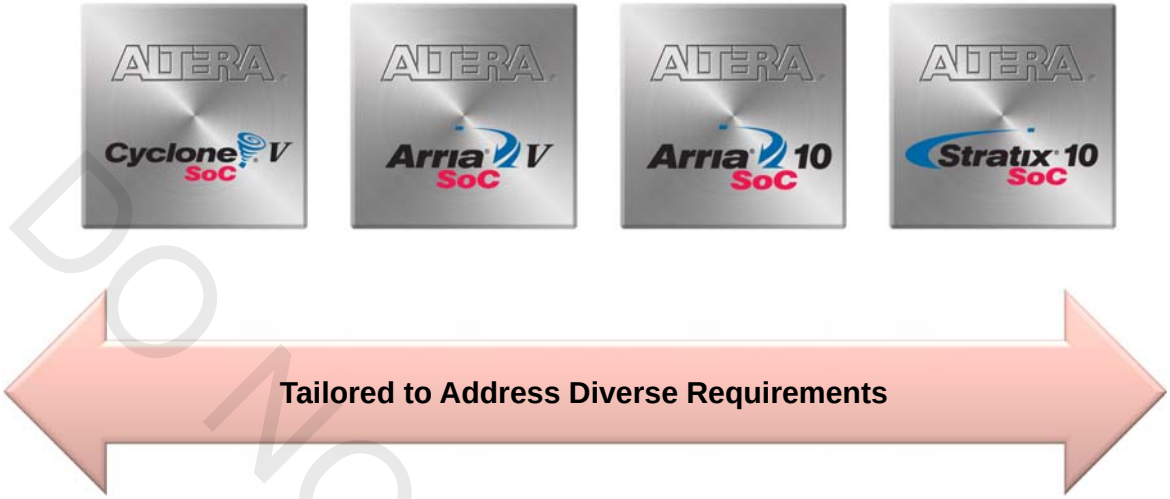


Reduced board size
- Up to 60% form factor reduction

Reduced system costs
- Lower component cost
- Reduction in PCB complexity and cost
  - Less routing with fewer layers



SoC Device Portfolio*



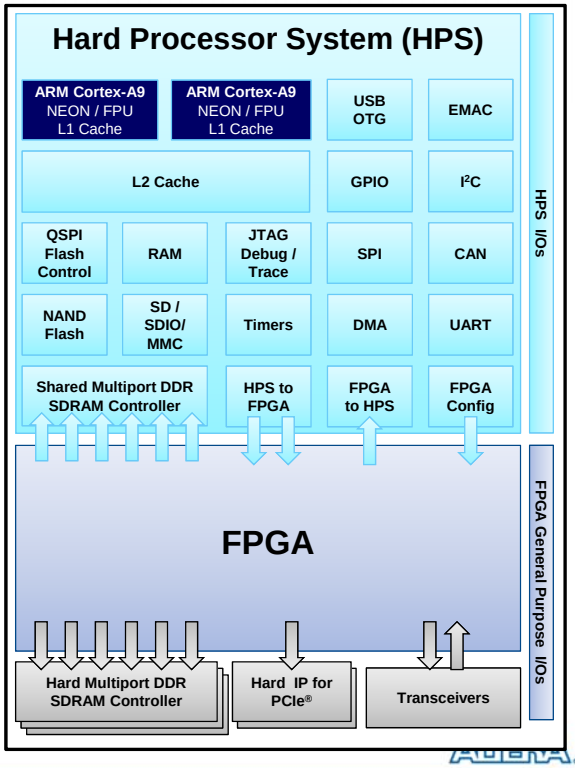
*Stratix® 10 SoCs not discussed in this class

SoC Key Features

Key Features	Cyclone V SoC	Arria V SoC	Arria 10 SoC
Process	28nm Low Power (28LP) process		20nm process
Processor Cache/coprocessors	Dual-core ARM Cortex-A9 MPCore processor L1, L2 Cache, NEON, Double Precision Floating Point Unit (FPU), Accelerator coherency port (ACP)		
Processor Performance	925 Mhz	1.05 GHz	1.5 GHz
Memory controllers support	Up to DDR3 400 MHz	Up to DDR3 533 MHz	Up to DDR4 1333MHz
Logic Density	25-110KLE	350-460KLE	160-660KLE
Transceivers	Up to 6 Gbps	Up to 10 Gbps	Up to 17 Gbps
Total Power Consumption	2W to 5W (Single core @ 300 MHz commercial temp to dual core @ 800 MHz industrial temp)	10W to 15W (Single core @ 300 MHz commercial temp to dual core @ 800 MHz industrial temp)	Up to 40% lower power compared to Arria V SoC

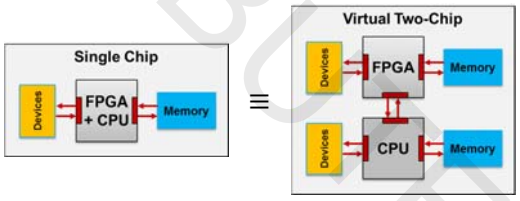
Other Features of SoC Devices

- Processor
  - Dual-core ARM® Cortex™-A9 MPCore™ processor
  - NEON™ coprocessor
  - Double-precision FPU
  - 32-KB L1 instruction and data caches per core
  - 512-KB shared L2 cache
  - 2.5 MIPS/MHz instruction efficiency
  - ARMv7-A
- FPGA Features
  - 8 input Adaptive Logic Modules (ALM)
  - Variable precision DSP blocks
  - Include hard floating-point on Arria® 10 devices
  - M10K/M20K + MLAB memory blocks
  - fPLLs
  - Hard IP for PCI Express®
- High-Bandwidth on-chip interfaces

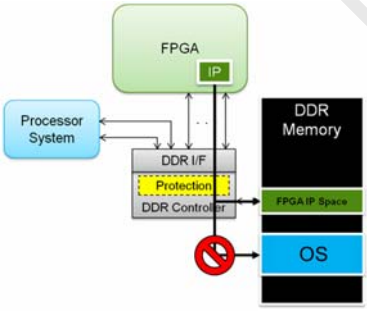


Architecture Matters

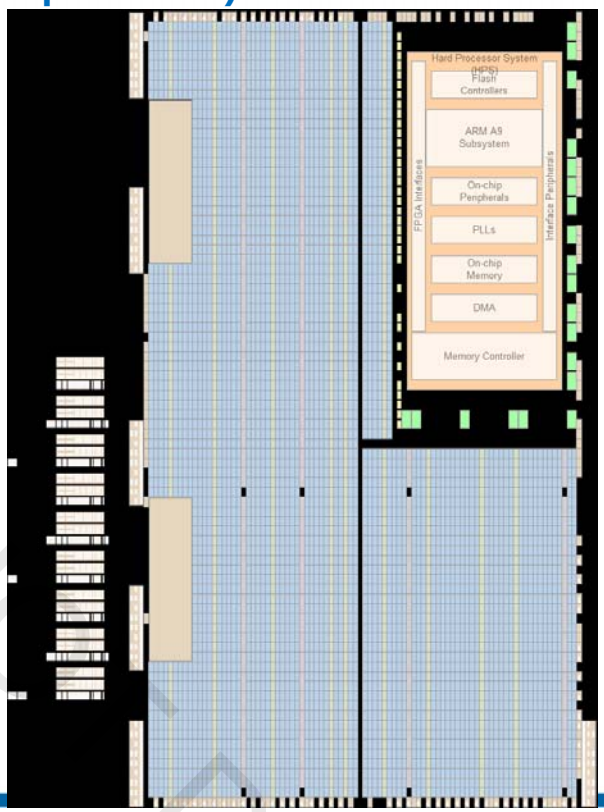
- Preserve Independence (virtual 2-chip operation)
  - Processor boot / FPGA configuration
  - FPGA operates even with CPU reset
  - Independent FPGA / CPU memories



- Protect Memory When Shared
  - CPU memory protected from FPGA IP



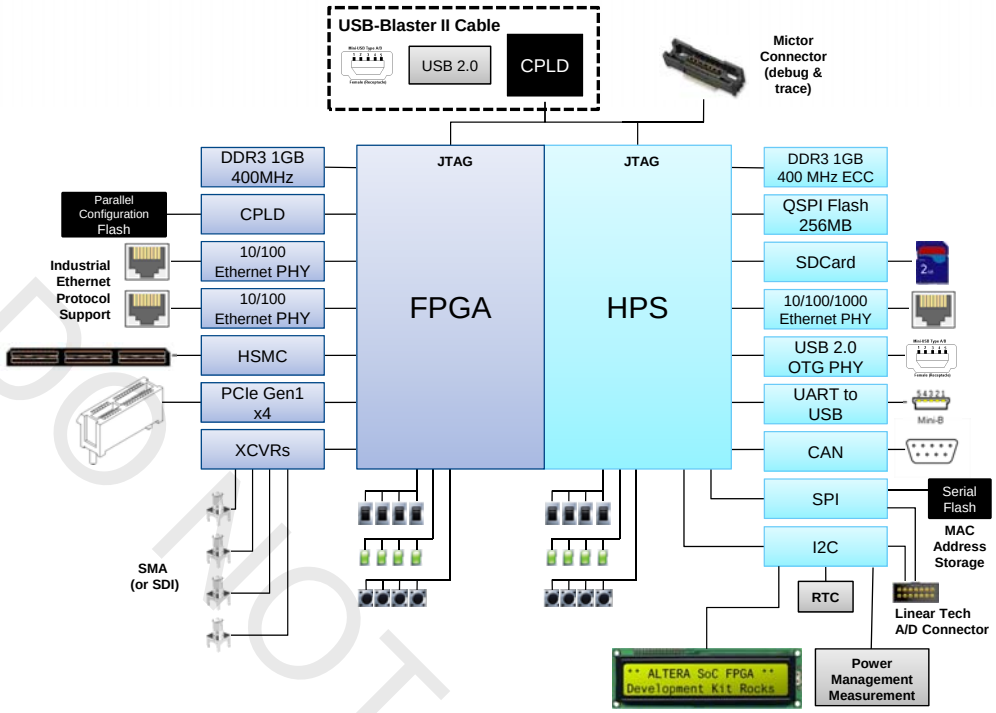
SoC Device (Chip Planner)



SoC Development Boards

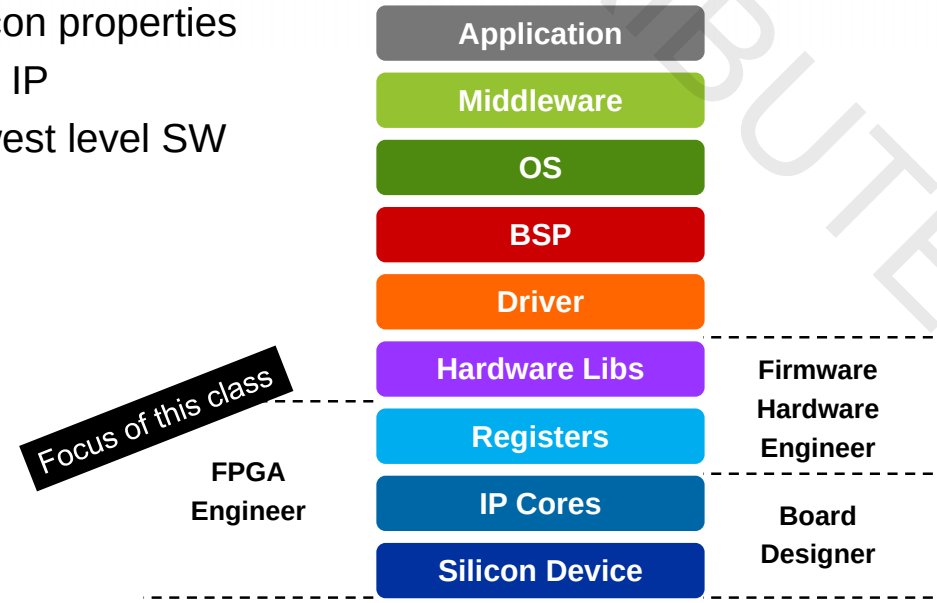
- Cyclone® V SoC
- Arria V SoC
- Arria 10 SoC
- DE1-SoC education board
- Arrow SOCKit
- Macnica Helio Board
- EBV SoCrates
- and many others

Cyclone V SoC Development Board Block Diagram



Hardware Development Perspective

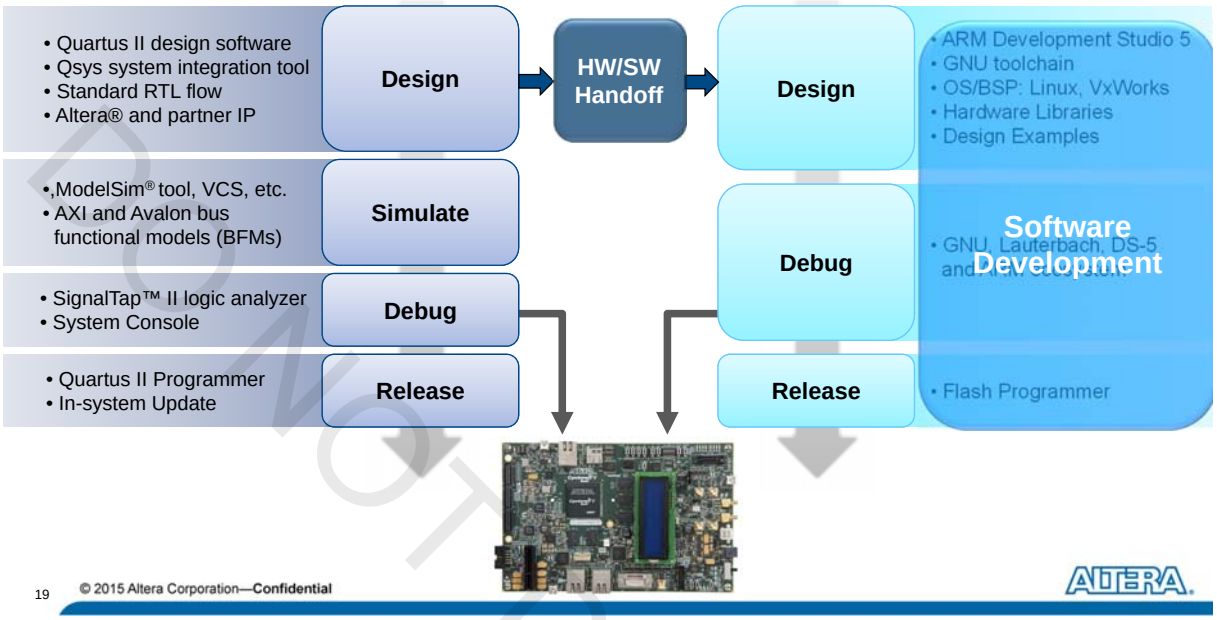
- Silicon properties
- Soft IP
- Lowest level SW



System Development Flow

FPGA Hardware Design Flow

Software Design Flow



Designing with an ARM-based System on a Chip

HPS Overview

HPS Overview Agenda

- HPS features
- System management
- Interconnect
- Memory and Memory Controllers
- DMA Controller

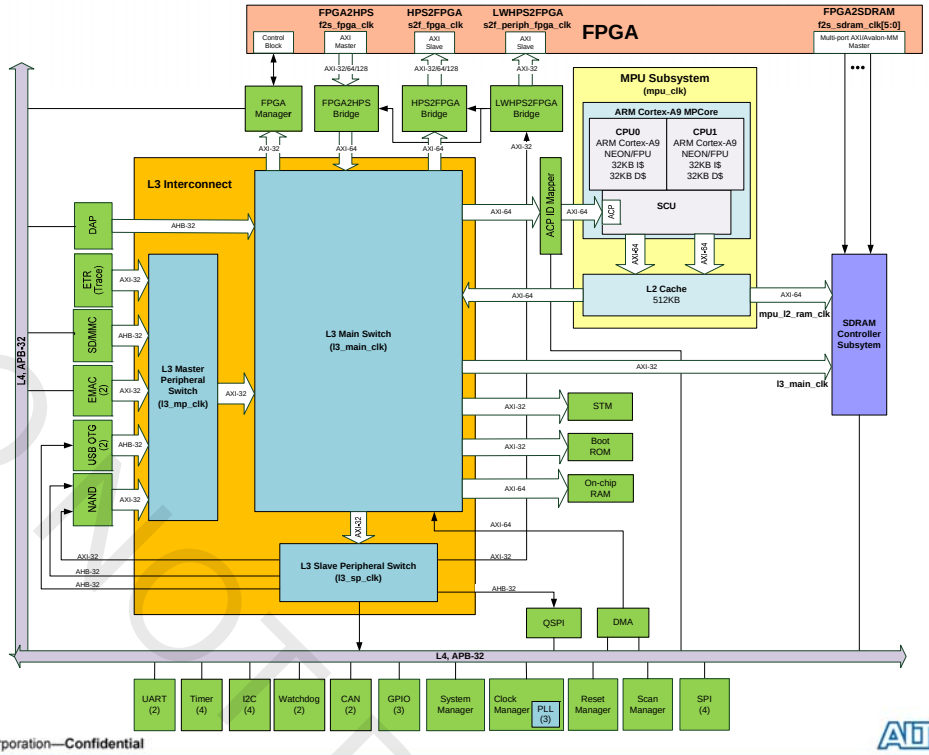
HPS IP Features

- Multi-Processor Unit (MPU) subsystem featuring dual ARM Cortex-A9 MPCore™ processor
- SDRAM controller subsystem/interconnect*
- General purpose direct memory access (DMA) controller
- 2 or 3* Ethernet media access controllers (EMACs)
- NAND, Quad SPI, Secure Digital (SD) and MultiMediaCard (MMC) flash controllers
- 2 USB 2.0 On-The-Go (OTG) controllers
- 2 Serial peripheral interface (SPI) master controllers
- 2 SPI slave controllers
- 4 or 5* Inter-integrated circuit (I^2C) controllers
- 2 Controller area network (CAN) controllers**
- 2 UARTs
- 3 GPIO interfaces

*Arria® 10 SoCs only

**Cyclone® V SoCs only

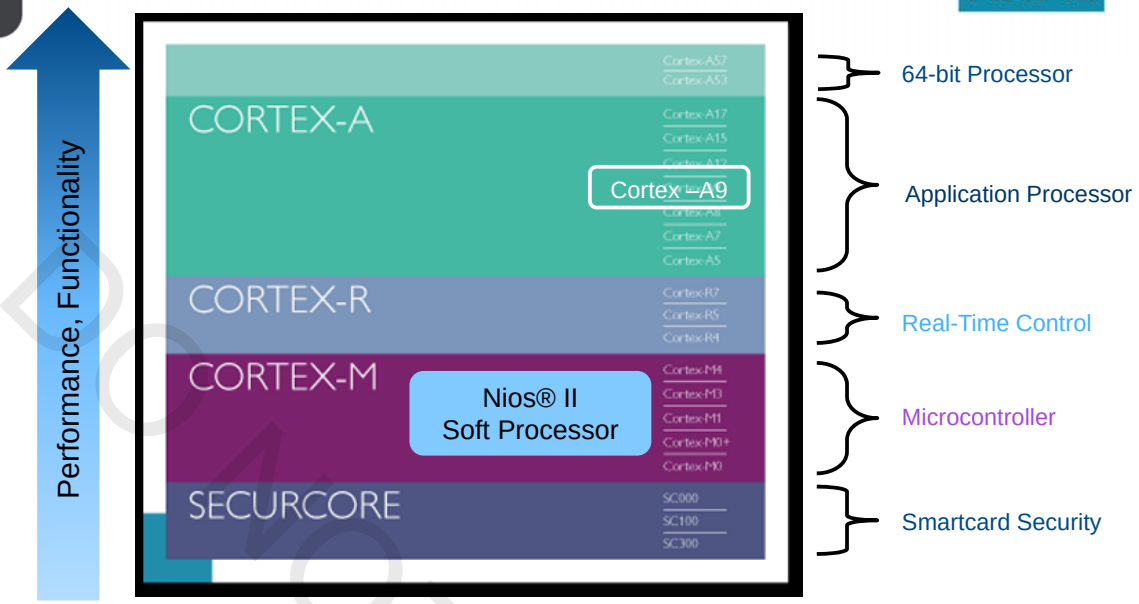
HPS Block Diagram



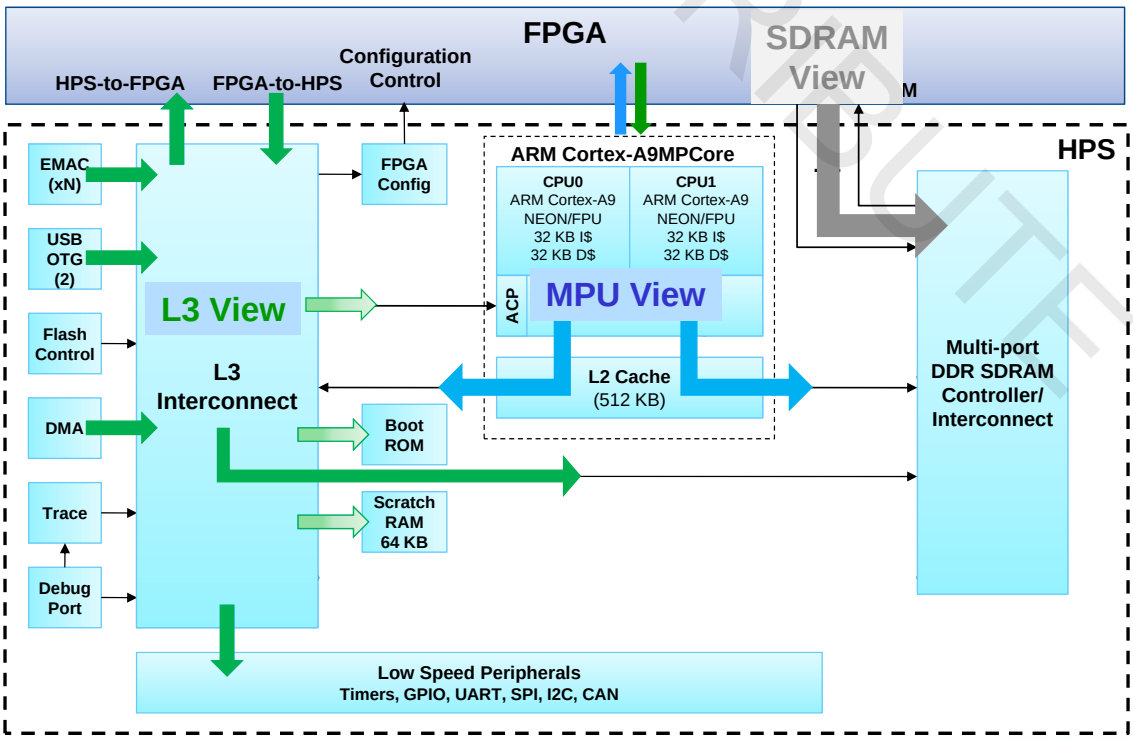
Overview

- Cortex-A9 MPU subsystem contains
  - Cortex-A9 MPCore processor
  - Level 2 cache
  - Debugging module
  - Accelerator coherency port (ACP) ID mapper
    - Allows memory coherency for other masters in system (DMA, L3 interconnect, FPGA peripherals, Debug Access Port)
- Cortex-A9 MPCore processor contains the following
  - Dual Cortex-A9 processor cores
  - Snoop control unit (SCU) with accelerator coherency port
  - Generic interrupt controller (GIC)
  - Global timer
  - Private timers and watchdogs
- Cortex-A9 processor core contains the following
  - ARM Cortex-A9 core
  - NEON™ Single Instruction Multiple Data (SIMD) engine with scalar Floating Point Unit (FPU)
  - Level 1 instruction and data caches
  - Memory management unit (MMU)

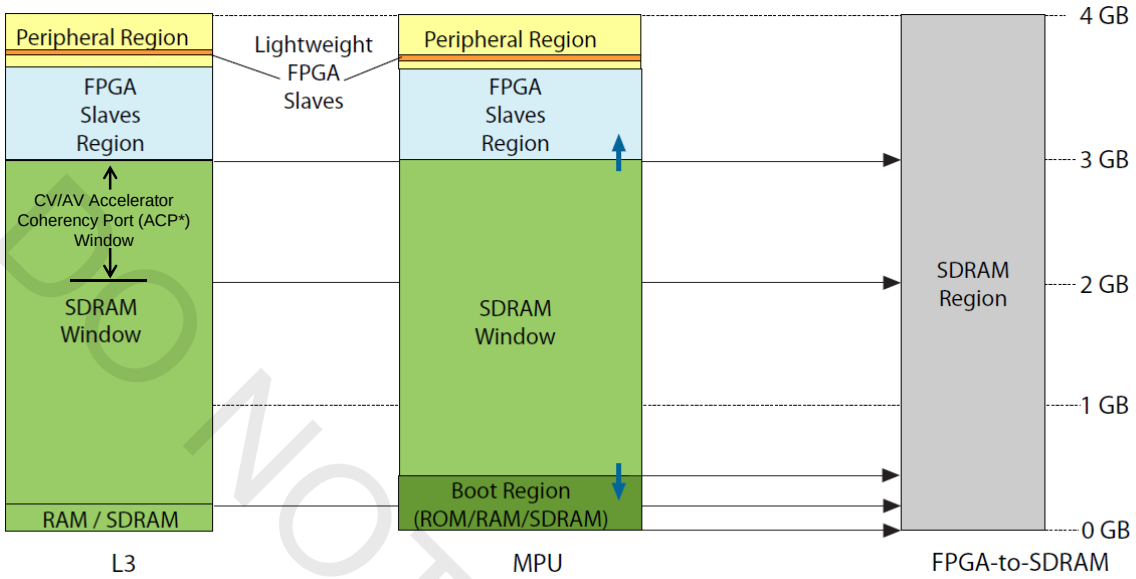
Cortex-A9 MPU Compared



HPS Memory Views



HPS Address Maps

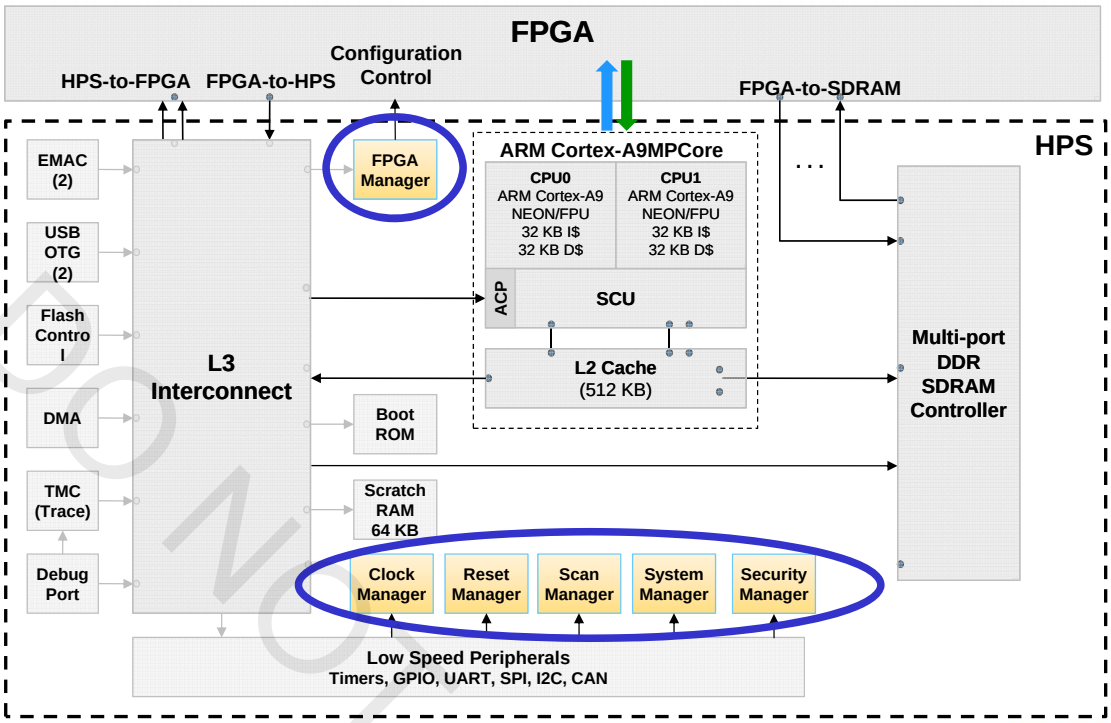


*Arria 10 ACP can be accessed for the entire SDRAM window if transaction is cacheable
*Arria V/Cyclone V ACP maps to the lowest 1G MPU SDRAM view by default

HPS Overview Agenda

- HPS features
- System management
- Interconnect
- Memory and Memory Controllers
- DMA Controller

System Management Components



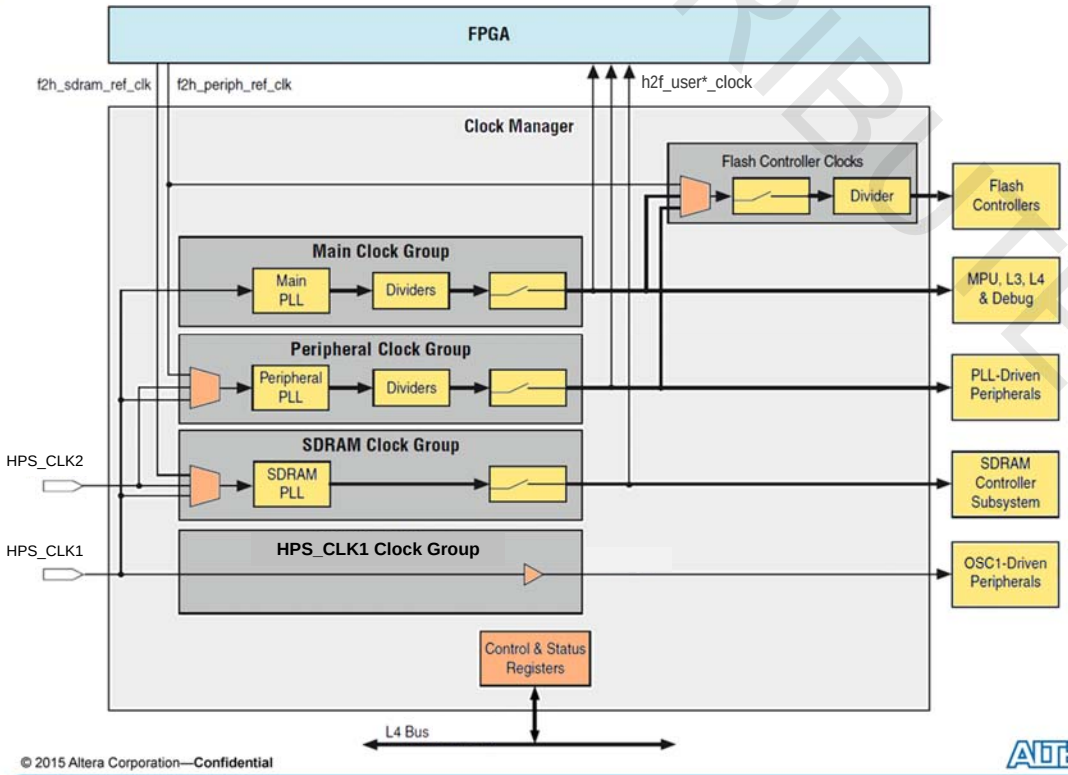
System Management

- Clocks and clock manager
- Resets and reset manager
- FPGA manager
- System manager
- Scan manager
- Security manager

Clock Manager Overview

- Manages Clocks in the HPS
  - Contains PLLs
  - Clock Frequency
  - Clock Muxing
    - Including sourcing clocks from the FPGA
  - Clock Gating
  - Controls sourcing of clocks to the FPGA
- Control and Status Register Access
  - Allows software control
  - Hardware libraries support

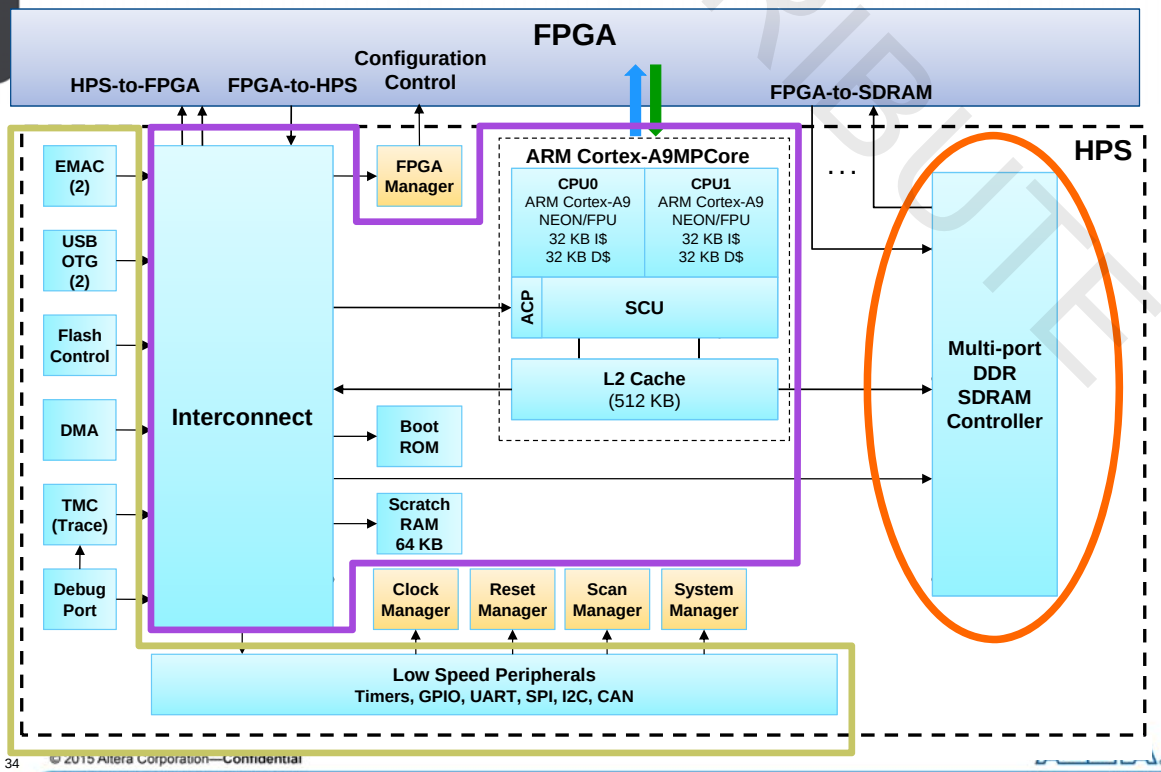
Cyclone V/Arria V HPS Clock Manager Block Diagram



Cyclone V & Arria V Clock Manager

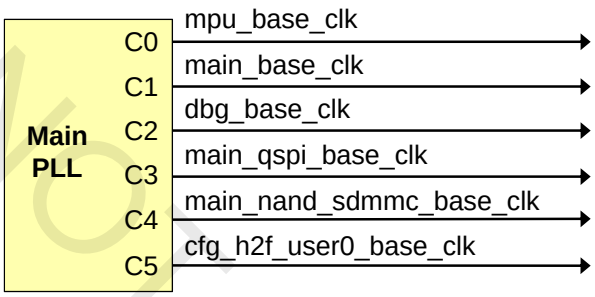
- Requires an external clock source
  - HPS_CLK1 pin
  - 10-50 MHz
  - Optional second source (HPS_CLK2 pin)
- Drives 3 clock groups, each with a separate PLL
  - Main PLL
  - Peripheral PLL
  - SDRAM PLL
- Optional clocks to and from FPGA
  - HPS-to-FPGA clocks 0/1/2, one from each PLL group
  - FPGA-to-HPS peripheral PLL reference clock
  - FPGA-to-HPS SDRAM PLL reference clock
- Hardware managed clocks
  - Main PLL C0, C1, and C2 are hardware-managed
    - All others are software-managed clocks

Cyclone V/Arria V: 3 HPS Clock Groups



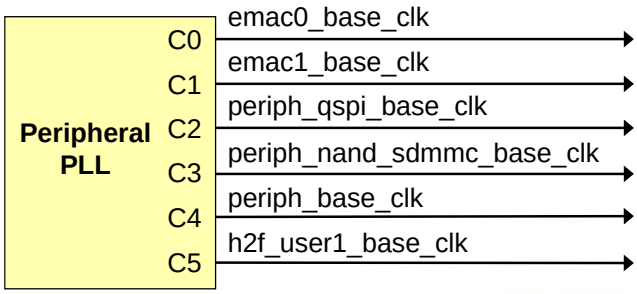
Cyclone V/Arria V Main PLL

- HPS_CLK1 input clock source pin
- Generates clocks for
  - MPU subsystem (C0)
  - L3 & L4 interconnect (C1)
  - Debug (C2)
  - Quad SPI flash (C3)
  - NAND & SD/MMC flash (C4)
  - Configuration & HPS-to-FPGA user clock 0 (C5)



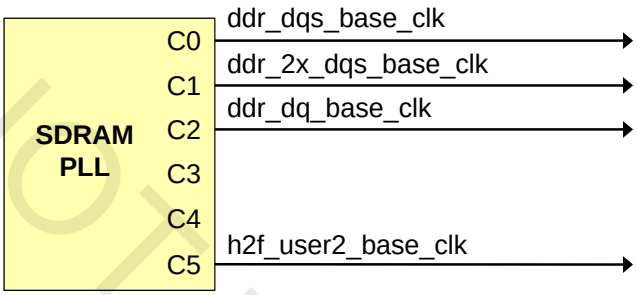
Cyclone V/Arria V Peripheral PLL

- Selectable Input sources
  - HPS_CLK1 input clock pin
  - HPS_CLK2 input clock pin
  - FPGA-to-HPS peripheral reference clock from FPGA fabric
- Generates clocks for
  - EMAC (C0 & C1)
  - Flash Controller (C2 & C3)
  - Main PLL group L4 interconnect (C4)
    - USB
    - SPI
    - CAN
    - GPIO
  - HPS-to-FPGA clock (C5)



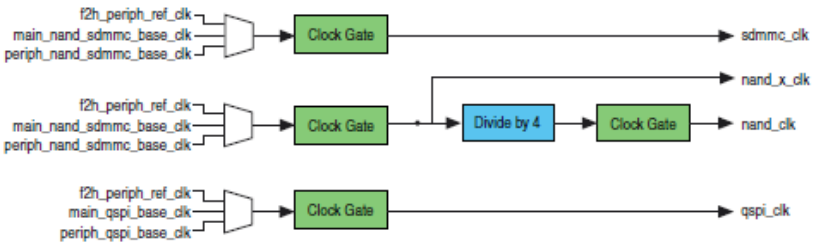
Cyclone V/Arria V SDRAM PLL

- Input sources
  - HPS_CLK1 input clock pin
  - HPS_CLK2 input clock pin
  - FPGA-to-HPS SDRAM reference clock from FPGA fabric
- Generates clocks for
  - SDRAM Controller Subsystem (C0-C2)
  - HPS-to-FPGA user clock 2 (C5)
- Only HPS PLL with phase and delay control



Cyclone V/Arria V Flash Controller Clocks

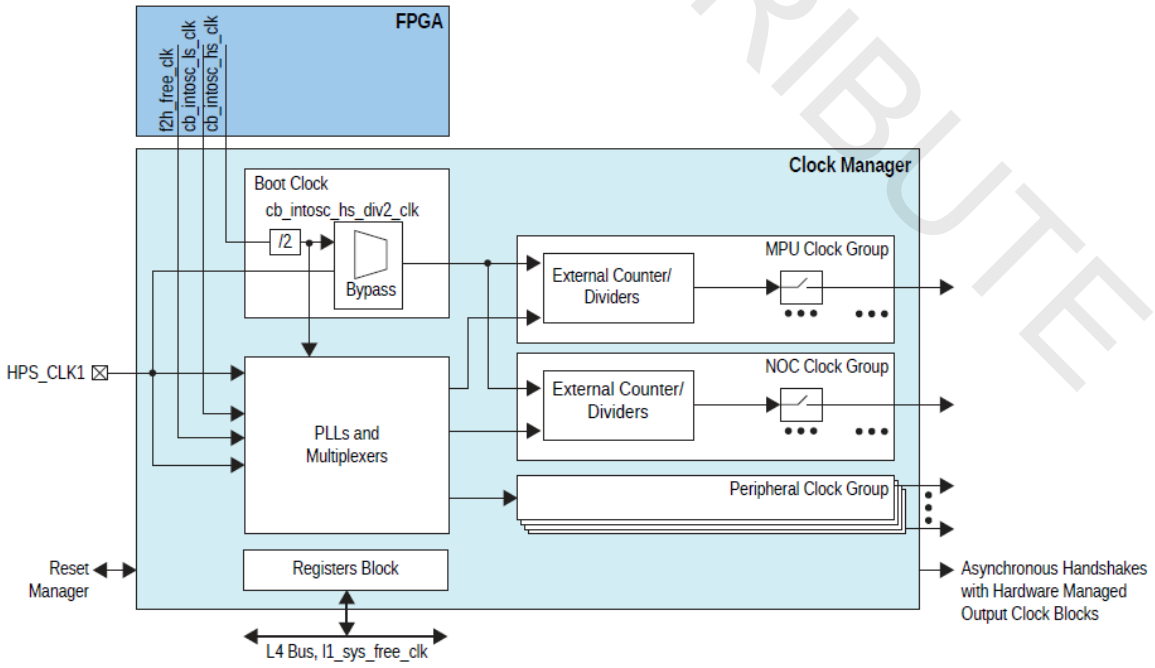
- Input sources
  - Main PLL
  - Peripheral PLL
  - FPGA Fabric
- Generates clocks for
  - QSPI Flash (Up to 108Mhz x4)
  - NAND Flash (Up to 250Mhz)
  - SD/MMC Flash (Up to 200Mhz)



Arria 10 Clock Manager

- 2 PLLs
- 3 Clock Groups
  - MPU
    - Hardware-sequenced clocks for the Cortex-A9 MPU subsystem
  - Network on chip (NoC)
    - Hardware-sequenced clocks for L3 interconnect, L4 peripheral bus, and debug
  - Peripherals
    - Software-sequenced clocks for PLL-driven peripherals
  - No SDRAM clock group as the SDRAM controller is located in the FPGA
- 4 possible clock source
  - HPS_CLK1 pin
  - FPGA Fabric PLL clock reference
  - FPGA configuration high speed internal oscillator
  - FPGA configuration low speed internal oscillator

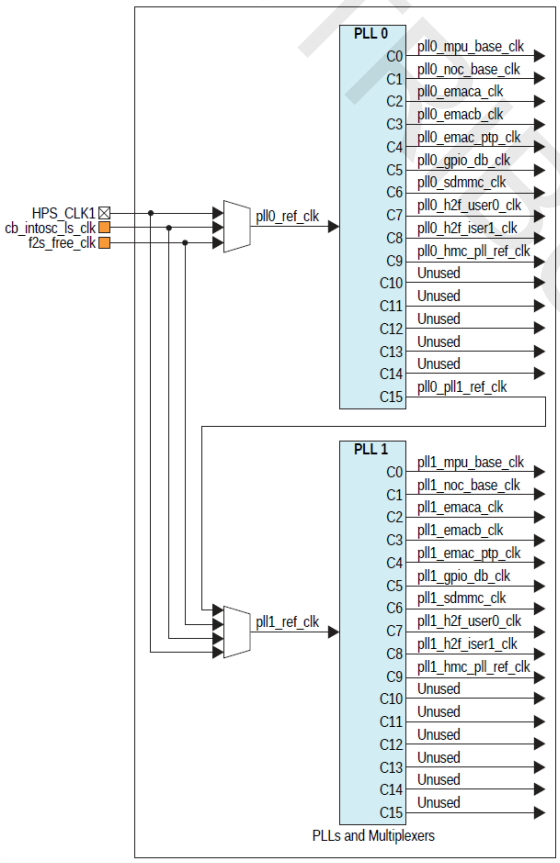
Arria 10 Clock Manager Diagram



Arria 10 Clock Manager Features

- Boot clock can be generated from 2 sources
  - FPGA cb_intosc_hs_clk /2 (Secure)
  - External HPS_CLK1 clock reference (non-secure)
- Each clock output block
  - Can be set to 2 modes with glitch-free external bypass multiplexer
    - Bypass
      - Uses Boot clock set at reset
    - Not-bypass (Uses one of 5 sources)
      - hps_clk1
      - f2s_free_clk (FPGA fabric clock reference)
      - cb_intosc_hs_div2_clk (FPGA control block clock divided by 2)
      - PLL0
      - PLL1
  - Has clock gates controlled by either hardware or software

Arria 10 PLLs



Software Management of Clocks

- Software sets clock manager using HW libraries
  - Output clock dividers
  - Clock selection / enable
  - Interrupts (Wake up MPU, lose/gain PLL lock)
  - Clock Sequencing (to prevent system lock-up)
- Most clocks require
  - SW manually gate off any clock affected by the change
  - Wait for any PLL lock if required
  - Gate the clocks back on
- Clocks must be very carefully sequenced or system can lock up
  - Use HW libraries to prevent this from happening (discussed later)

HPS Entry/Exit of 'Safe Mode'

- Put clocks into known safe state on cold or warm reset request from Reset Manager
  - Uses HPS_CLK1
  - Clock manager register settings, for reset PLL counter & dividers, bypassed to default values
  - Enables all clocks (turns clock gating off)
  - Flash controller clock multiplexer selects output from peripheral PLL
- Exit Safe Mode by resetting *Safe Mode* bit of *CTRL* register
  - Handled by the second stage bootloader stage *(discussed later)* during normal boot up
  - Otherwise, done by using the hardware libraries *(discussed later)*

System Management

- Clocks and clock manager
- Resets and reset manager
- FPGA manager
- System manager
- Scan manager
- Security manager

Reset Manager Overview

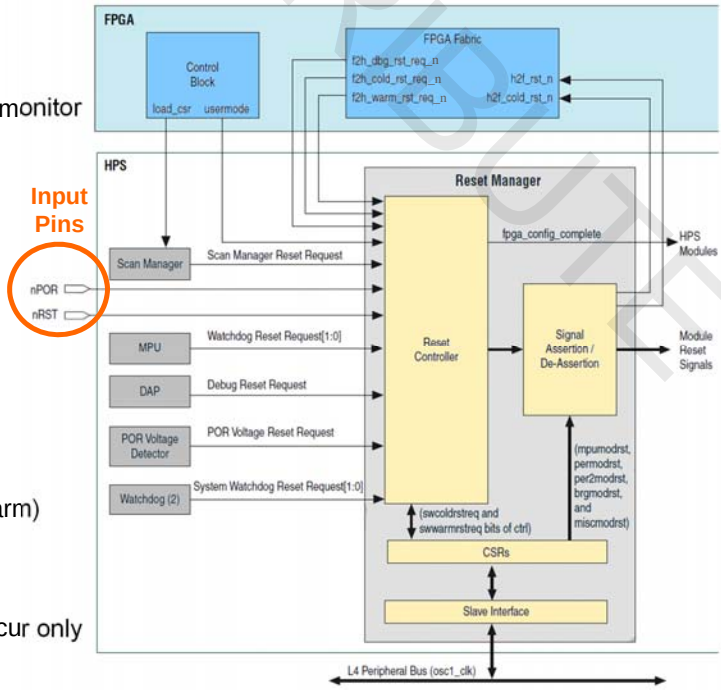
- Generates module reset signals based on reset requests from various sources
- Clocked by HPS_CLK1 clock pin
- Three separate reset domains
  - HPS system
  - JTAG port
  - Debug system
- Accepts reset requests from following
  - Software writing module reset registers
  - FPGA control block
  - FPGA fabric
  - External Reset pins
- Implements Cold and Warm reset

Cold / Warm / Debug Resets

Cold Reset	Warm Reset	Debug Reset
Affects all reset domains (JTAG, Debug, System)	Happens after HPS has already been cold reset	Only affects debug reset domain
Places hardware-managed clocks into safe mode	Used to recover system from a non-responsive condition	
Places software managed clocks into their default states	Resets a subset of the HPS	
Asynchronously resets all registers in the clock manager	Debug & JTAG reset domain unaffected	
Resets SDRAM so memory contents and setup lost	SDRAM unaffected so memory contents preserved	

Reset Manager Integration

- Reset Sources
  - Power-On-Reset voltage monitor (cold reset)
  - nPOR pin (cold reset)
  - nRST pin (warm reset)
  - Debug (Debug Reset)
  - Watchdogs (SW control)
  - FPGA
    - f2h_cold_rst_req_n
    - f2h_warm_rst_req_n
    - f2h_dbg_rst_req_n
    - h2f_cold_rst_n
    - h2f_rst_n (cold or warm)
    - load_csr (cold reset)
- FPGA monitoring
  - Reset from FPGA can occur only if FPGA is configured

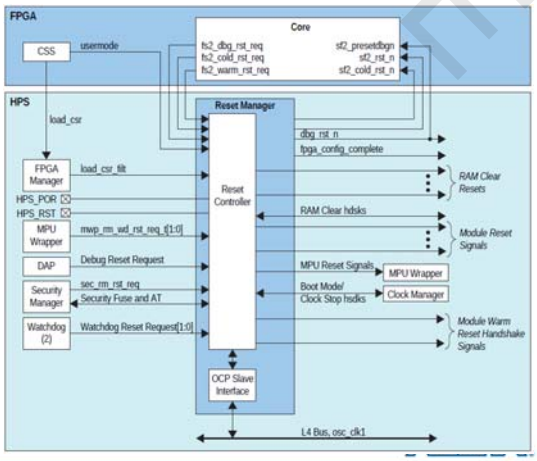


Warm Reset is Software Configurable

- Warm Reset can be configured to wait if
  - FPGA needs to finish outstanding transactions
  - SDRAM not yet in self refresh mode
  - Trace Module needs to Stall and or Finish outstanding transactions
  - Scan Manager interfaces not quiescent (may generate clock glitches)
- Source of last warm reset can be logged
  - Warm reset log must be cleared by software

Arria 10 Reset Manager Additional Features

- Handles anti-tamper reset
- Power on Reset handled by Security Manager and FPGA configuration subsystem
- Additional RAM clear domain
  - Clears the following RAM blocks on cold or warm
    - On-chip RAM
    - NAND read, write, and ECC RAMs
    - MPU RAMs



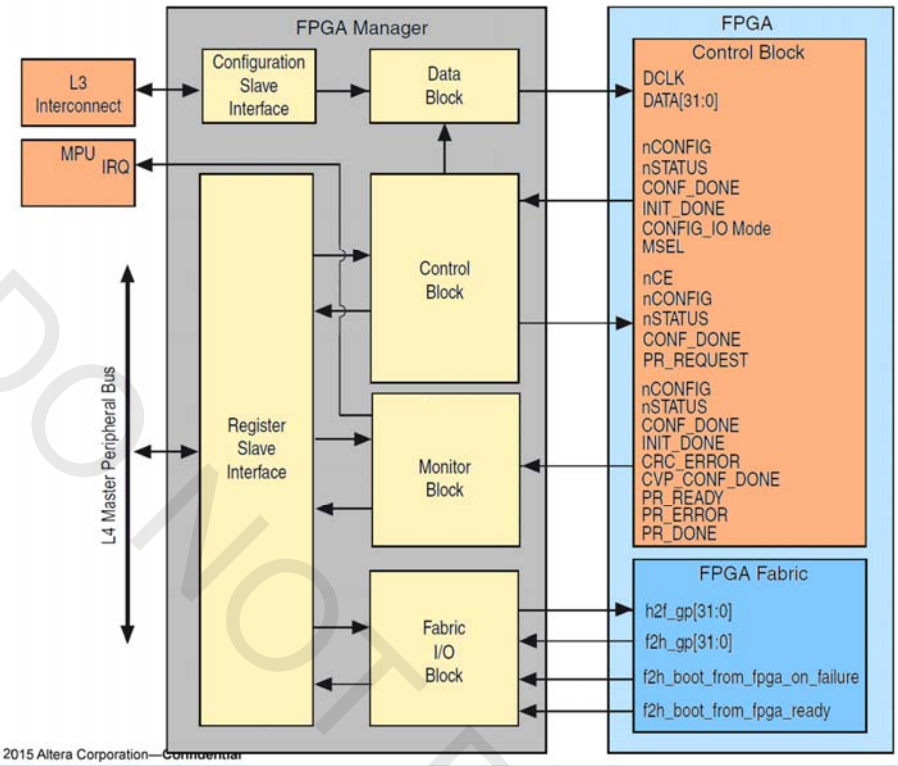
System Management

- Clocks and clock manager
- Resets and reset manager
- FPGA manager
- System manager
- Scan manager
- Security manager

FPGA Manager Overview

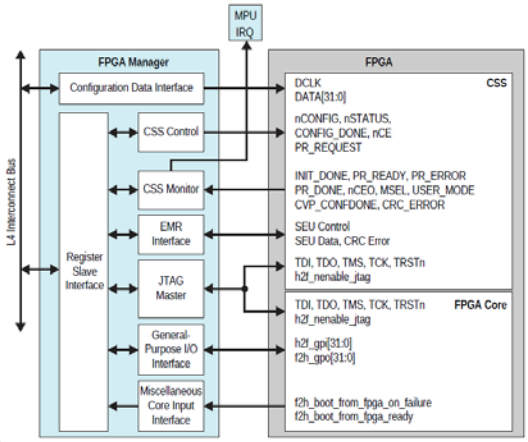
- Manages and monitors the FPGA portion of the SoC
  - Configure the FPGA fabric
    - Partial reconfiguration, encryption, decompression supported
    - MSEL pins needs to be set appropriately
  - Monitor FPGA configuration and power status
    - *INIT_DONE, CRC_ERR, PR_DONE,* etc.
    - Software configurable interrupts to MPU
  - Can reset the FPGA
  - Drive/Receive 32 GPIOs from the FPGA
  - Boot from FPGA handshake signals
    - Used for booting the HPS from the FPGA fabric
  - Supported by the Hardware Libraries

FPGA Manager Block Diagram

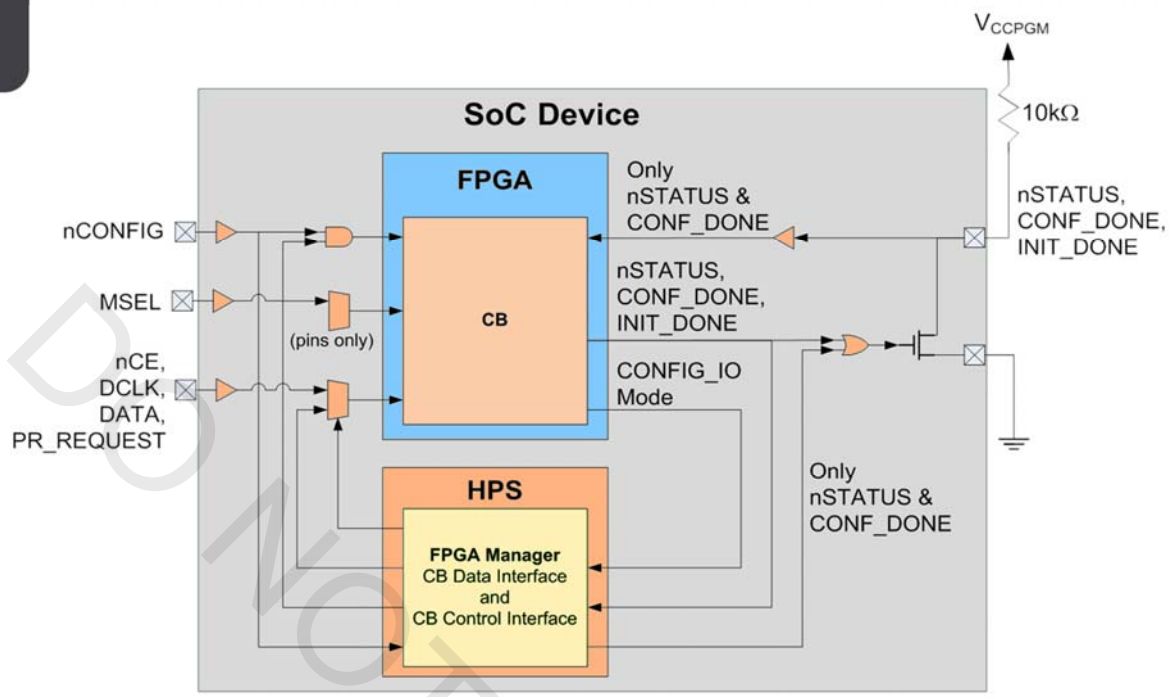


Arria 10 FPGA Manager Additional Feature

- Error Message Register (EMR) Interface
  - Error message extraction in case of CRC errors in the FPGA
- FPGA JTAG Hosting
  - Allows HPS to take control of configuration subsystem or provide JTAG functionality to the FPGA
  - Does not take control of the JTAG I/O



HPS Configuring FPGA Fabric



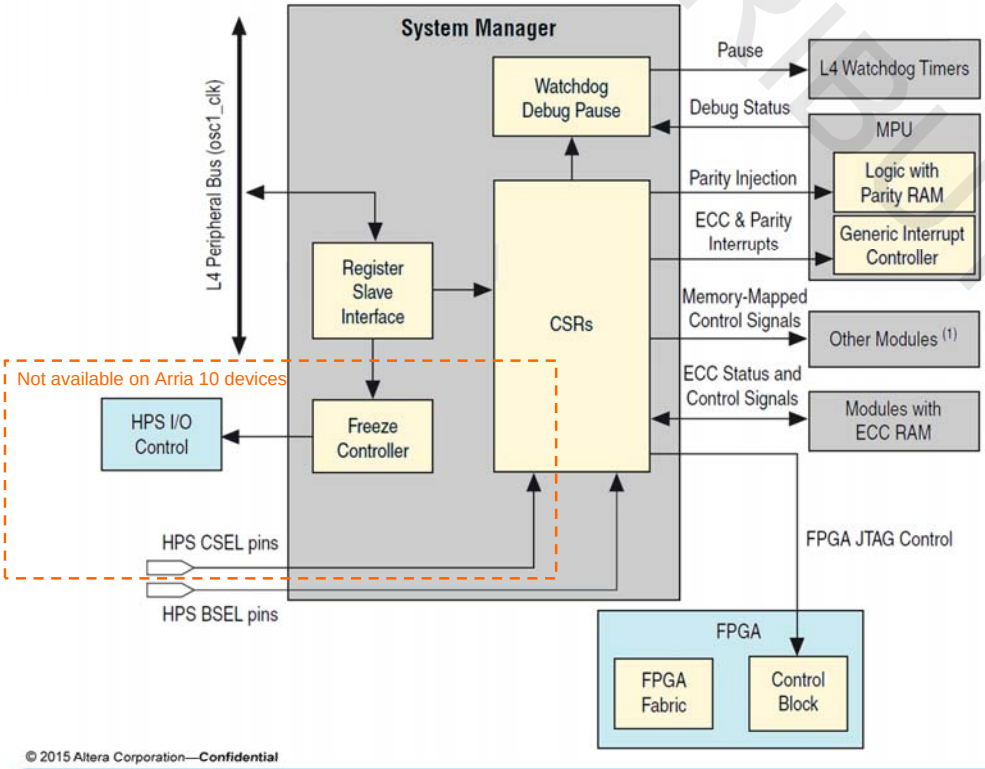
System Management

- Clocks and clock manager
- Resets and reset manager
- FPGA manager
- System manager
- Scan manager
- Security manager

System Manager Overview

- Contains memory-mapped control and status registers
- Manage HPS I/O features
- Enable/disable other HPS peripherals
- Provide access to boot configuration information
- Provide access to status signals in other HPS modules
- Enable and controls ECC and parity in HPS modules
- Provide registers to pass information between warm boots
- Pause watchdog timers during debug mode

System Manager Block Diagram



HPS System Manager I/O Features

- Tri-states HPS-configurable I/O pins during configuration (Freeze controller)
  - Not on Arria 10 devices
- Controls HPS peripheral I/O pin multiplexing
- Enables/disables various HPS I/O peripherals interfaces
- Enables/disables HPS-FPGA interfaces

System Manager – Managed Peripherals

- EMACs
- USB controllers
- SD/MMC controller
- NAND controller
- SPI masters
- Quad SPI controller
- DMA controller
- On-chip RAM
- CAN controllers
- Debug core
- Reset manager

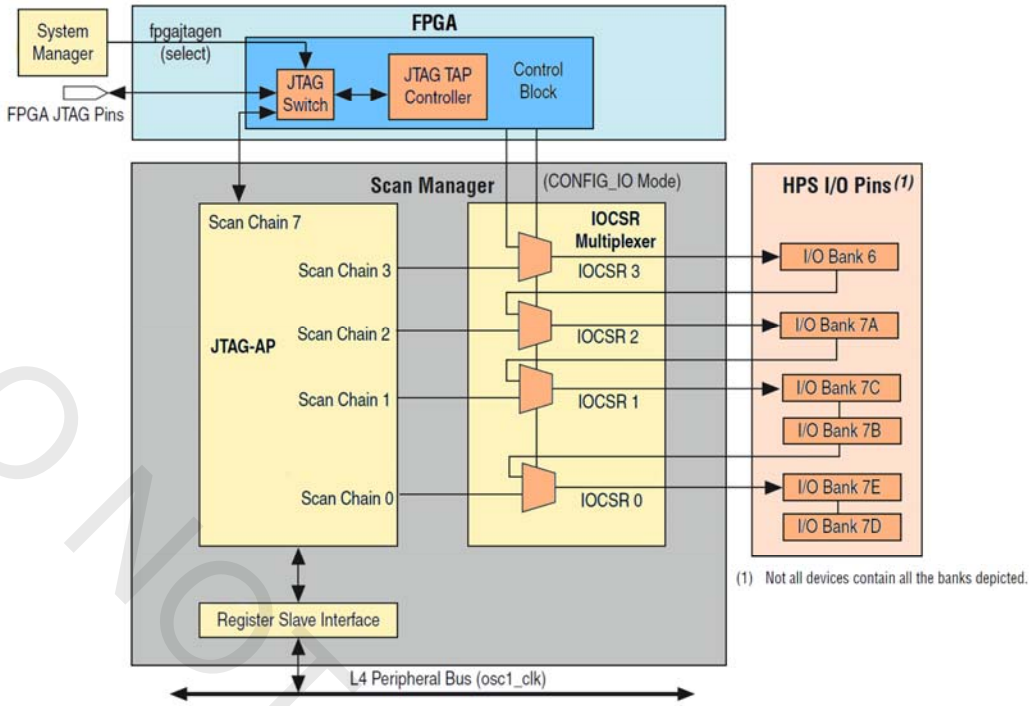
System Management

- Clocks and clock manager
- Resets and reset manager
- FPGA manager
- System manager
- Scan manager
- Security manager

Scan Manager Overview

- Cyclone V and Arria V SoCs only
- Configures and manages HPS I/O pins
  - Configures I/O bank settings (voltage, drive, pull up, etc.)
  - HPS I/O must be frozen by System Manager before configuration
- Gives HPS access to FPGA JTAG
  - Can perform any FPGA JTAG operation
  - Disables external FPGA JTAG when it takes over
- Contains ARM JTAG Access Port (JTAG-AP)
  - Multiple scan chain JTAG master interfaces
    - One connects to FPGA JTAG (standard JTAG signals)
    - Four connect to HPS I/O banks

Scan Manager Block Diagram



Scan Manager

- Drives serial scan-chains connected to:
  - HPS configuration IOCSR chains
    - Configures HPS I/Os
    - SDRAM DDR PHY (DLL/OCT)
  - FPGA JTAG
    - Allows HPS to monitor single event upsets (SEUs)
- From cold reset will trigger freeze sequence
  - All HPS I/O will be considered un-configured
    - IO Buffers are in tri-state
    - Weak pull-up enabled
  - During Boot phase
    - On-chip boot code uses Scan Manager to configure boot device's IOCSR
    - Initial software will use Scan Manager to configure I/Os
    - When configuration complete, the freeze manager will receive an indicator from System Manager to unfreeze I/Os
- Not involved in Design for Test (DFT) or boundary-scan testing

System Management

- Clocks and clock manager
- Resets and reset manager
- FPGA manager
- System manager
- Scan manager
- Security manager

Security Manager Overview

- Available on Arria 10 SoCs
- Centralized device security control
- All top-level clocks and reset signals go through the security manager for validation
- Features
  - Allows Secure Boot through authentication and/or decryption
  - Anti-Tamper protection
  - ARM TrustZone® Technology (firewall with security manager features)

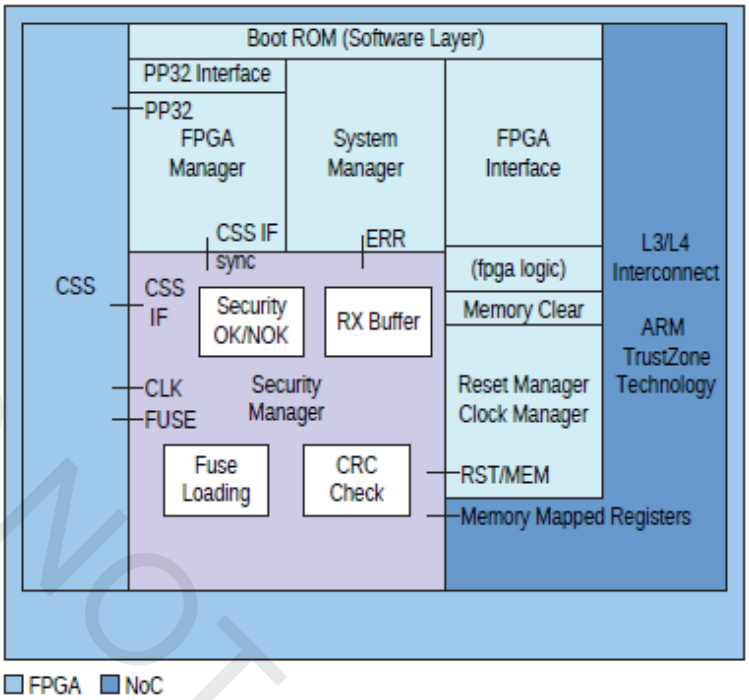
Authentication and Encryption

- When Secure Boot is enabled, the device only accepts authenticated software
  - BootROM authenticates and/or decrypts 2nd Stage Bootloader
  - 2nd Stage Bootloader authenticates and/or decrypts Secure Micro OS
  - Through secure APIs, Micro OS lets applications in Standard OS establish trusted services
- Arria 10 SoCs support hierarchical public key infrastructure
- Once authenticated the HPS software can configure the FPGA with encrypted bitstream

Security Manager Components

- CSS (Configuration Sub-System) interface
  - Receives decrypted data from the CSS engine
- Fuse interface
  - Request and receive fuse data from the CSS block
    - Data stored in one-time programmable fuses on the device
- Register interface
  - Software read and/or writes to security state bits, status, errors, enable interrupt or DMA interface
- Anti-tamper – trigger memory clearing of all internal memories

Security Manager Block Diagram



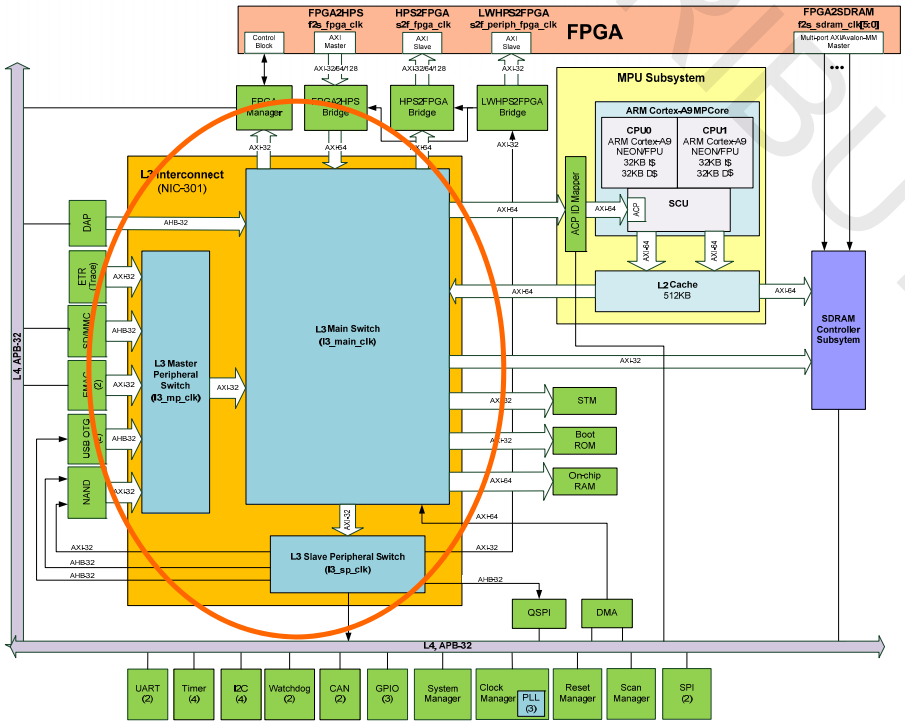
HPS Overview Agenda

- HPS features
- System management
- Interconnect
- Memory and Memory Controllers
- DMA Controller

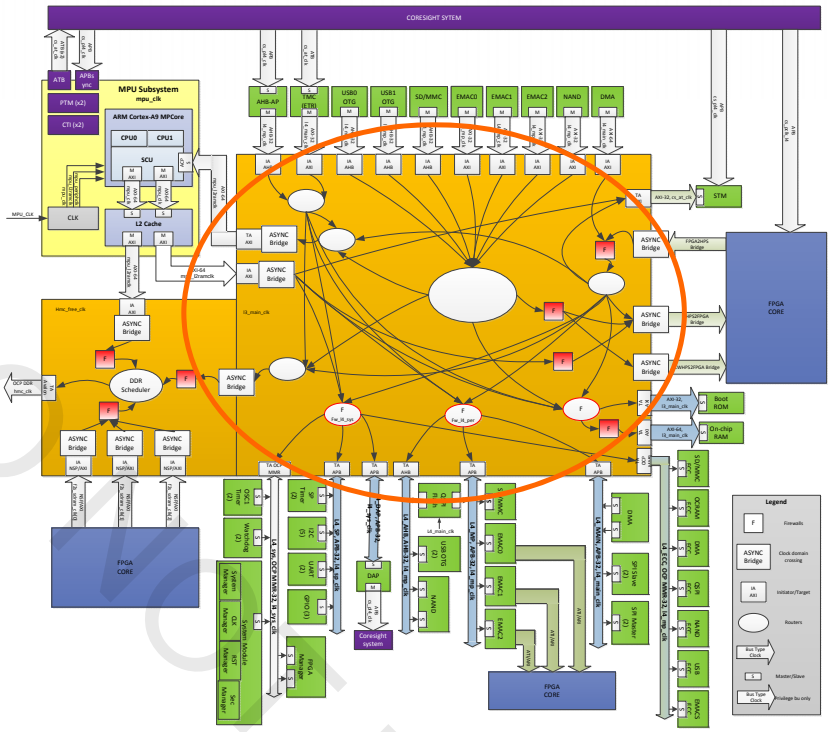
Interconnect Overview

- Cyclone V/Arria V SoCs: ARM CoreLink™ Network Interconnect (NIC-301)
- Arria 10 SoC: Arteris FlexNoC™ Interconnect
- Based on AMBA® AXI, AHB™, and APB™ protocols
- Level 3 is Multi-layer, nonblocking architecture comprised of three switches
  - Main switch
    - 64 bits of connectivity to AXI bridges, on-chip memories, SDRAM and FPGA manager
    - Half the MPU main clock frequency
  - Master peripheral switch
    - 32 bits connecting memory mastering peripherals to main switch
    - Half the main switch clock frequency
  - Slave peripheral switch
    - 32 bits connecting L3, L4 slave interfaces for masters of main and master peripheral switches
    - Operates at a much slower speed

Cyclone V/Arria V Interconnect Diagram



Arria 10 Interconnect Diagram



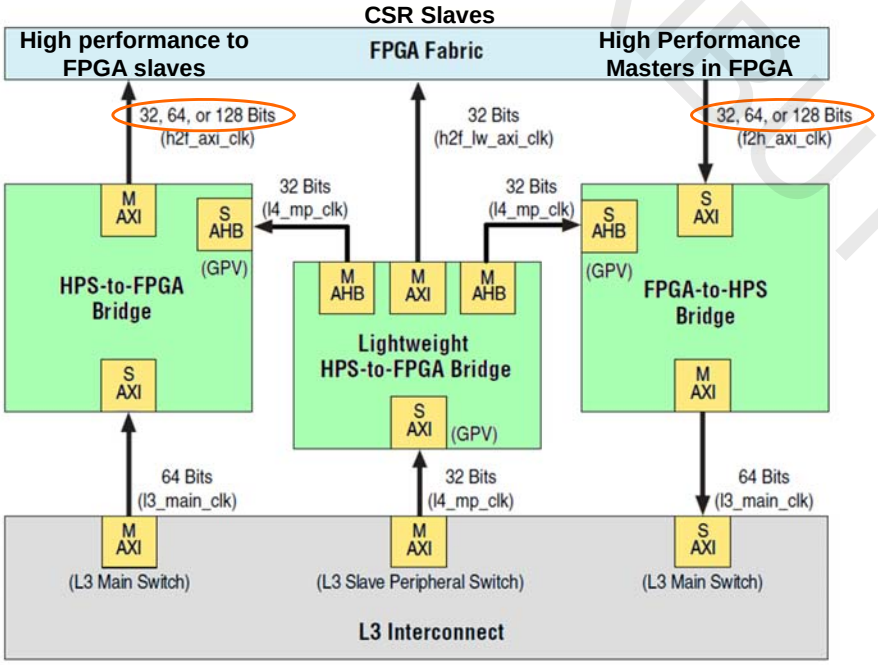
Arria 10 Interconnect Features

- Security firewall support
  - Configure secure or non-secure access per peripheral
  - Configure privilege and user access for some of the peripherals
  - Per transaction security
- ECC

Level 3 Interconnect Up/Downsizing

- Level 3 interconnect different width interfaces
  - 64-bit main switch
  - 32-bit master peripheral switch
  - 32-bit slave peripheral switch
- Sizing logic used at boundaries
  - Between L3 interconnect and module
  - Between 32-bit and 64-bit sub switches
- Sizing logic only packs cacheable accesses
- Sizing logic attempts to minimize 64-bit traffic
  - 32→64 bit: pack two 32-bit words into a single 64-bit word
  - 64→32 bit: perform one 64-bit access and split into two 32-bit words

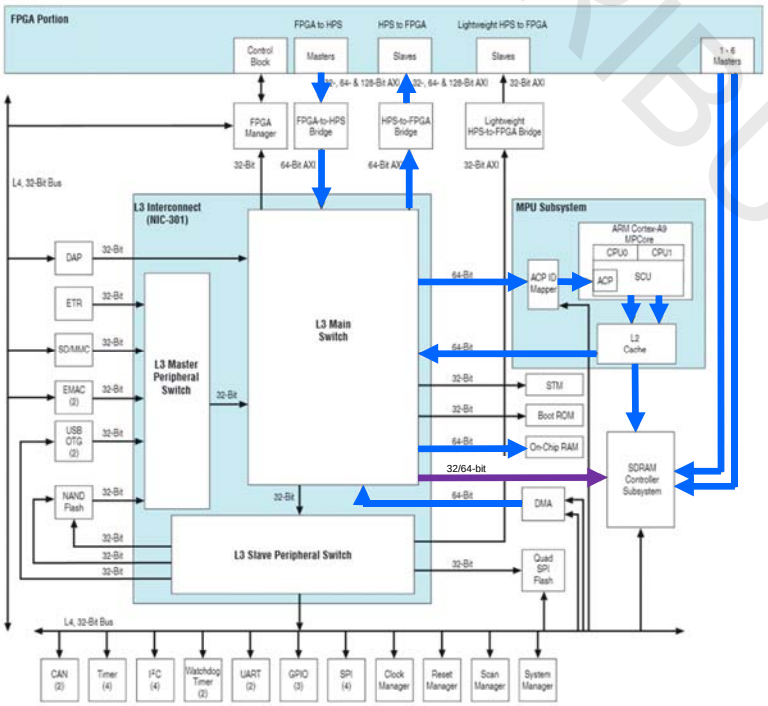
AXI Bridges Architecture



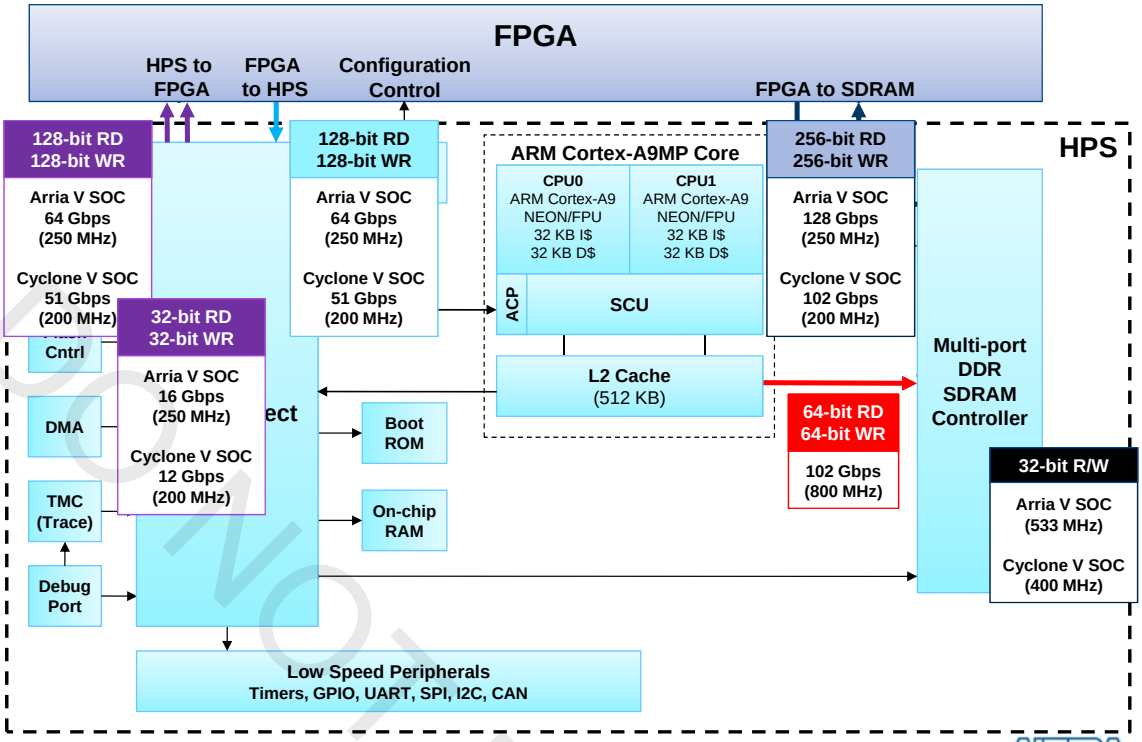
Global Programmers View (GPV)

- GPV exposes register set of the L3 interconnect to control bridge properties and behavior
- Only three masters can access the GPV
  - MPU subsystem
  - FPGA-to-HPS bridge
  - Debug access port (DAP)
- GPV registers are used to access the following
  - Slave security
  - Address remapping
  - Arbitration priority
  - Write tide mark (buffer fill level before issuing transaction to slave)
- Some GPV accesses must be secure
  - Slave security and address remapping require secure accesses

High Performance Paths



Example System Throughput

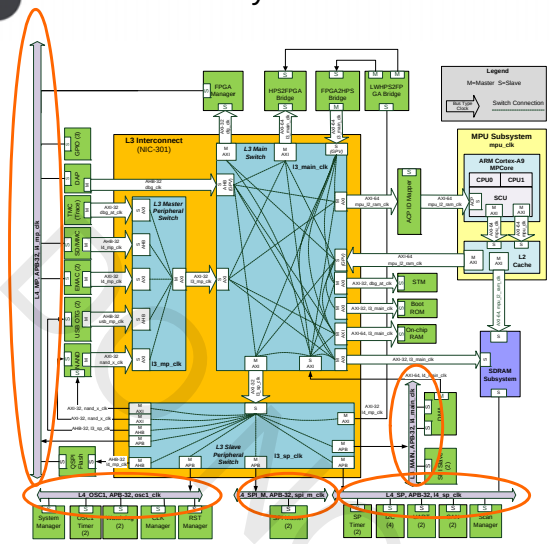


FPGA-to-HPS Bridge Drawbacks

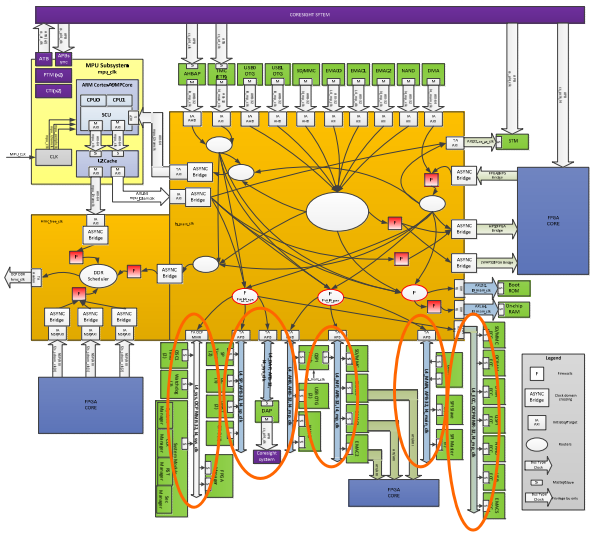
- Additional latency through the L3 interconnect
- Access to HPS SDRAM interface is only 32-bit
  - On Cyclone V and Arria V SoCs
- Access to ACP is throughput limited
  - Due to cache coherency logic
- Access to secure slaves require FPGA master to support security
  - Built into the AXI protocol
  - For Avalon masters, Qsys supports hard coded security scheme
- Use direct FPGA-to-SDRAM interface instead of FPGA-to-HPS bridge when cache coherency is not needed
  - Minimize latency
  - Improve throughput

Level 4 Peripheral Bus Interconnect

Arria V and Cyclone V L4 Buses



Arria 10 L4 Buses



Level 4 Peripheral Bus

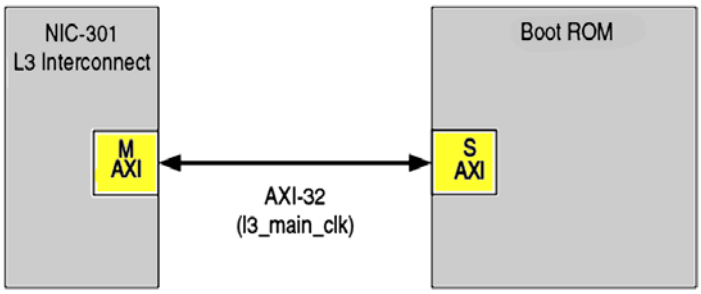
- Level 4 buses used for control and status access
  - Configuring modules
  - Enabling/clearing module interrupts
  - Reading back module status
- HPS contains a virtual level 4 bus
  - 5 physical APB buses connected to level 3 interconnect
    - 7 for Arria 10 Devices
- Each physical APB bus
  - Operates on a unique clock domain (L3 handles clock crossing)
  - Connects to multiple APB slave interfaces

HPS Overview Agenda

- HPS features
- System management
- Interconnect
- Memory and Memory Controllers
- DMA Controller

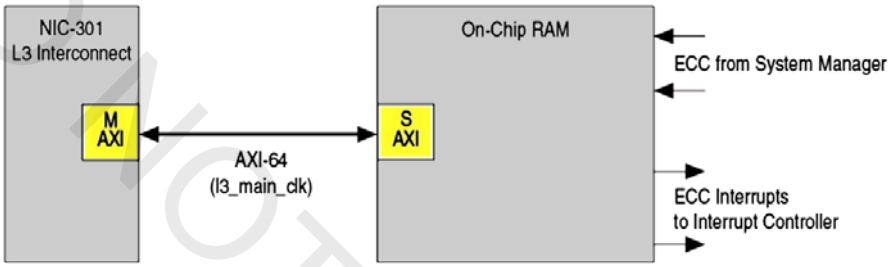
On-Chip ROM Features

- 64 KB for Cyclone V and Arria V SoCs
  - 128 KB for Arria 10 SoCs
- 32-bit AXI interface
- Accessed after warm or cold reset
  - Stores initial bootloader
  - Programmed by Altera



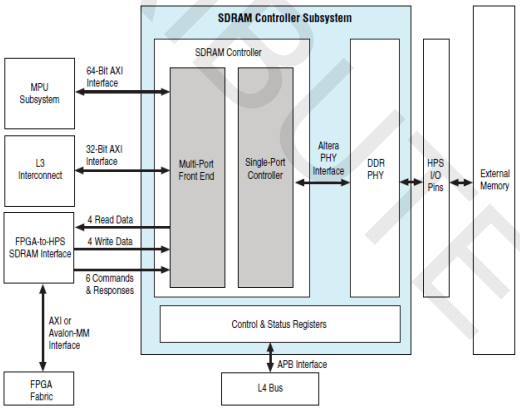
On-Chip RAM Features

- 64 KB (28nm SoCs) or 256 KB (Arria 10 SoCs)
  - Used as initial bootloader scratch pad
  - Runs 2nd stage bootloader and possibly user bootloader
- 64-bit AXI interface
- ECC support
  - Single bit correction
  - Double bit detection



HPS DDR Memory Controller

- One 64-bit AXI port for CPU
- One 32-bit AXI port for L3
- Available direct FPGA Ports
  - Up to 6 for Cyclone V/Arria V SoC
    - Avalon and AXI
  - Up to 3 for Arria 10
    - AXI only
- Variable DRAM port width
  - 8-bit, 16-bit, 32-bit
  - 64-bit (Only Arria 10)
- Optional 8-bit ECC
- Up to 4GB of address map for DDR
- Firewall and security support (Arria 10 SoC)



SDRAM Controller Features (1)

- SDRAM device support
  - Supports DDR2 (Cyclone V/Arria V only), DDR3, and LPDDR2(Cyclone V/Arria V SoC only) SDRAM
  - DDR4 and LPDDR3 support with Arria 10 SoC
  - Up to 4GBit density parts and 2 chip selects
- Error correcting code (ECC) support
  - Hamming single-error correct and double-error detect reads
  - ECC write-backs
  - User notification of ECC errors
- Operation ordering
  - Command reordering (bank look ahead management)
  - Data reordering (transaction reordering)

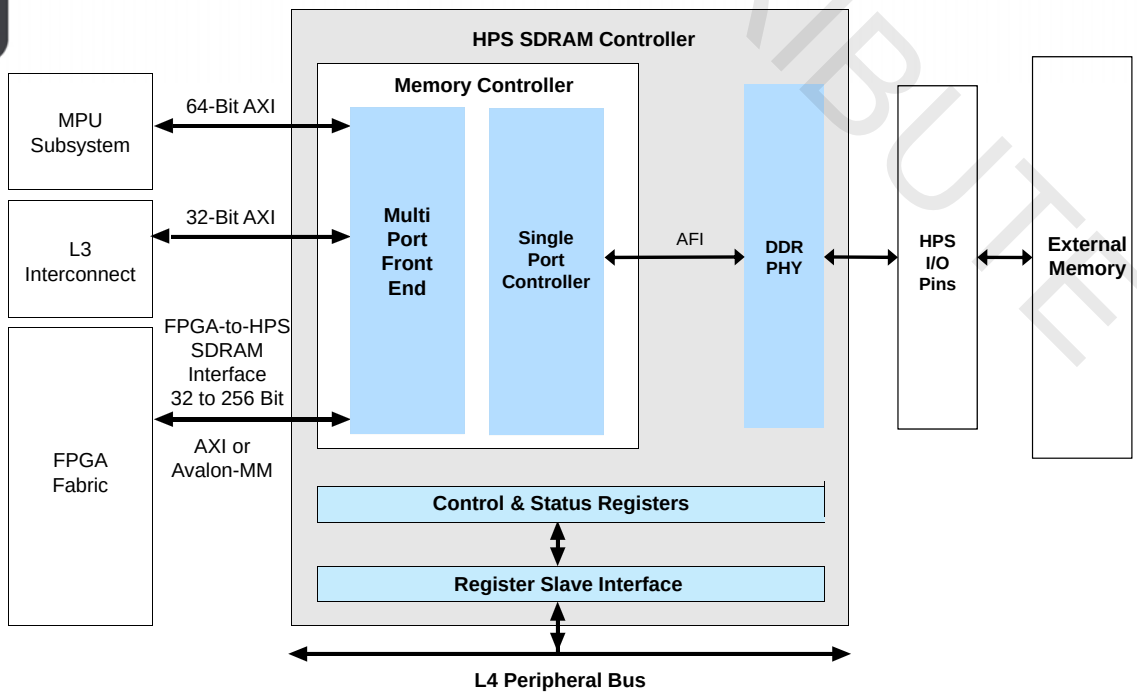
SDRAM Controller Features (2)

- Port scheduling
  - Absolute priority per port with eight levels
  - Relative weight per port
    - Deficit weighted round robin scheduling for ports of same priority
  - Priority and weight can be configured dynamically
- Burst adaptation
  - Burst alignment support for efficient unaligned burst access
  - Burst adaptation support of chopping and merging for user burst bigger or smaller than memory burst length
- Memory device power management
  - Support for self refresh command and partial array self refresh
  - Power down support for user defined length or based on inactivity
  - Deep power down support for LPDDR

SDRAM Controller Features (3)

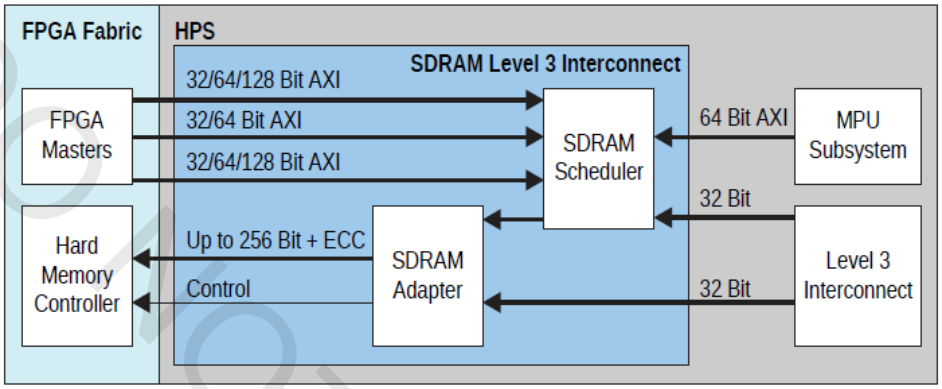
- AXI support
  - AMBA AXI4 ordering rules supported
- Memory protection
  - If designed exclusively with AXI masters TrustZone® is supported
- Exclusive support for shared memory accesses
  - Tracking logic for up to 16 exclusive write operations

SDRAM Controller Diagram (Cyclone V/Arria V SoC)



SDRAM Interconnect (Arria 10 SoCs)

- Uses HMC (Hard Memory Controller) located in the FPGA but through the SDRAM interconnect inside the HPS



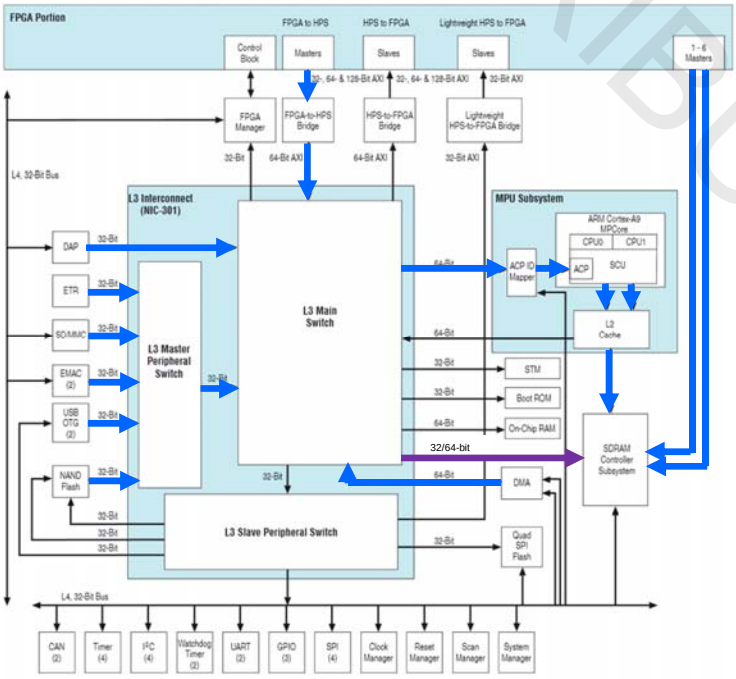
HPS SDRAM Controller Configuration (28nm SoCs)

- MPU subsystem must initialize interface
  - Program I/O scan chain bits (voltage, delay chain settings)
  - Program memory controller settings (width, timing)
  - Perform calibration
- Implications
  - HPS 2nd stage bootloader software will be responsible for controller initialization and calibration
  - Quartus II software will provide appropriate settings and bit stream
- For Arria 10 SoCs
  - SDRAM Controller is done through FPGA peripheral configuration
    - Can be performed by software

Maximizing SDRAM Performance

- When possible give FPGA masters direct access
  - Using the FPGA-to-HPS bridge will limit throughput to 32-bit data
  - Traffic through the FPGA-to-HPS bridge has to compete with other masters
  - FPGA SDRAM ports are wider and typically will be lower latency
- Carefully determine MPFE weighting
  - Absolute and relative weighting
  - Ensure that a performance critical path has appropriate weighting
  - Remember that FPGA masters are also competing for bandwidth and may swamp the memory controller

Paths Into HPS SDRAM



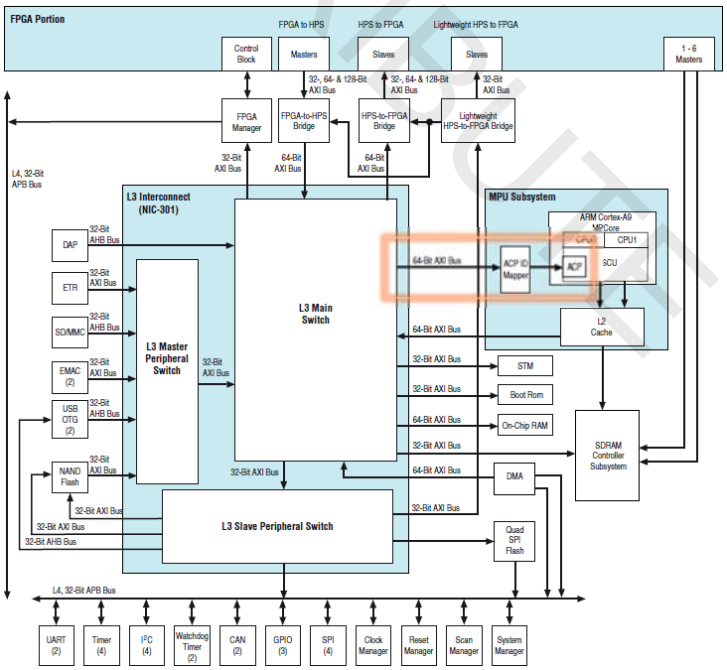
Considerations when Accessing HPS SDRAM from FPGA

- HPS SDRAM FPGA ports are highly configurable
  - AXI standards supported
  - Avalon-MM* support can be configured to be read/write only
- Use multiple FPGA ports instead of Qsys arbitration
  - ex: 3 FPGA masters connected to HPS SDRAM
  - Give each FPGA master an independent port
  - Goal: use arbitration built into the controller as apposed to soft logic
- Keep the FPGA masters and HPS SDRAM port on the same clock domain
  - SDRAM FPGA ports have built in clock crossing that is always enabled
  - Using different clock domains will only increase memory latency

Only directly supported in Arria V and Cyclone V SoCs

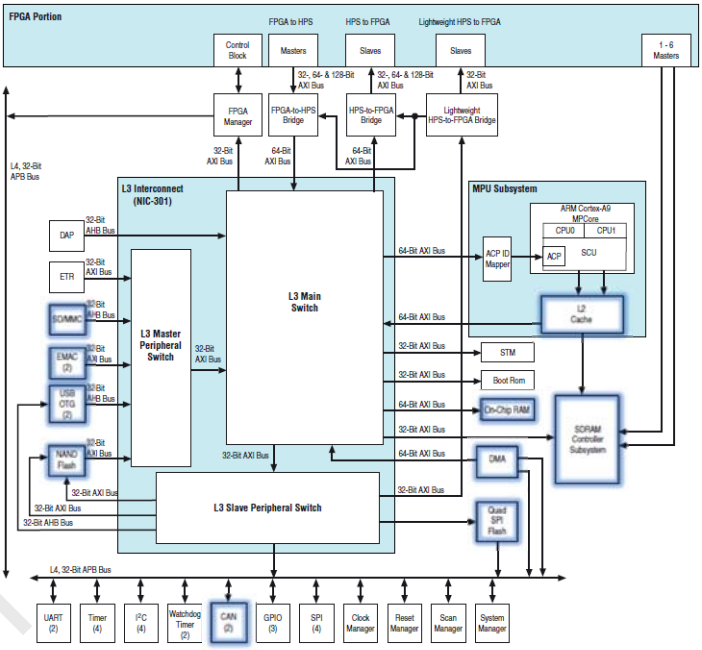
Coherent Memory Sharing

- Coherent memory support for *all* masters
  - FPGA and HPS
- ACP ID Mapper
  - Supports up to 6 concurrent L3 Masters
  - Unlimited transactions pending
- High Bandwidth
  - 64-bit port running at ½ CPU clock



Memory Protection: ECC Support

- DDR controller (16-bit, 32-bit, 64-bit)
- L2 cache
- On-chip RAM
- QSPI, SD/MMC, NAND
- DMA
- USB, EMAC, CAN



HPS Overview Agenda

- HPS features
- System management
- Interconnect
- Memory and Memory Controllers
- DMA Controller

Direct Memory Access Controller (DMAC) Overview

- ARM Corelink™ Controller (DMA-330)
- Microcode architecture for various transfer types
  - Small instruction set providing flexibilty
  - Memory to memory, memory to/from peripheral
- Support for up to eight DMA channels
  - Eight concurrently executed threads (AXI read/writes)
  - One additional manager thread
- Provides interrupt lines into the MPU subsystem
  - For thread abort and events
- Dedicated support for secure & non-secure accesses
  - Dual slave interface
- Contains multi-FIFO data buffer

DMAC Peripheral Flow Control Features

- Peripherals can request transfers
  - Simple request and acknowledge interfaces provided
  - Most peripherals require two interfaces for receive and transmit
  - Peripherals can request single or burst transfers
- Flow control with 32 peripheral request interfaces
  - 4 for FPGA
  - 4 shared between FPGA and CAN controller (2 RX and 2 TX)
  - 8 for I^2C controller (4 RX and 4 TX)
  - 8 for SPI masters and slaves (4 RX and 4 TX)
  - 2 for QSPI flash controller (RX and TX)
  - 1 for System Trace Macrocell
  - 4 for UART (2 RX and 2TX)
  - 1 for FPGA manager
- Conditional code execution support based on peripheral request type

References

- For more information refer to the Cyclone V, Arria V, or Arria 10 handbooks
  - https://www.altera.com/products/soc/overview.html
- SoC Hardware Overview online trainings
  - Microprocessor Unit
    - http://wl.altera.com/education/training/courses/OEMB5500
  - Interconnect and Memory
    - http://wl.altera.com/education/training/courses/OEMB5500INT
  - System Management, Debug, and General Purpose Peripherals
    - http://wl.altera.com/education/training/courses/OEMB5501
  - Flash Controllers and Interface Protocols
    - http://wl.altera.com/education/training/courses/OEMB5501FLASH
- External Memory Interface Handbook
- External Memory Interfaces instructor led training

Quiz: Appropriate Bridge to Use

Scenario	Which Bridge?
ARM MPU accessing FPGA SGDMA status register	
ARM MPU writing to FPGA SGDMA descriptor memory	
FPGA SGDMA master access to HPS memory	
HPS writing a character to the FPGA JTAG UART	
ARM MPU reading the FPGA System ID	
HPS DMA transfer to FPGA SDRAM	
ARM MPU executing code from FPGA SDRAM	
Nios MPU writing to HPS SDRAM to share data with HPS	

Depending on the cache coherency needs of the access, either the FPGA-to-HPS bridge or the FPGA-to-SDRAM interface will be accessed.

Exercise 1

Configuring the HPS Component

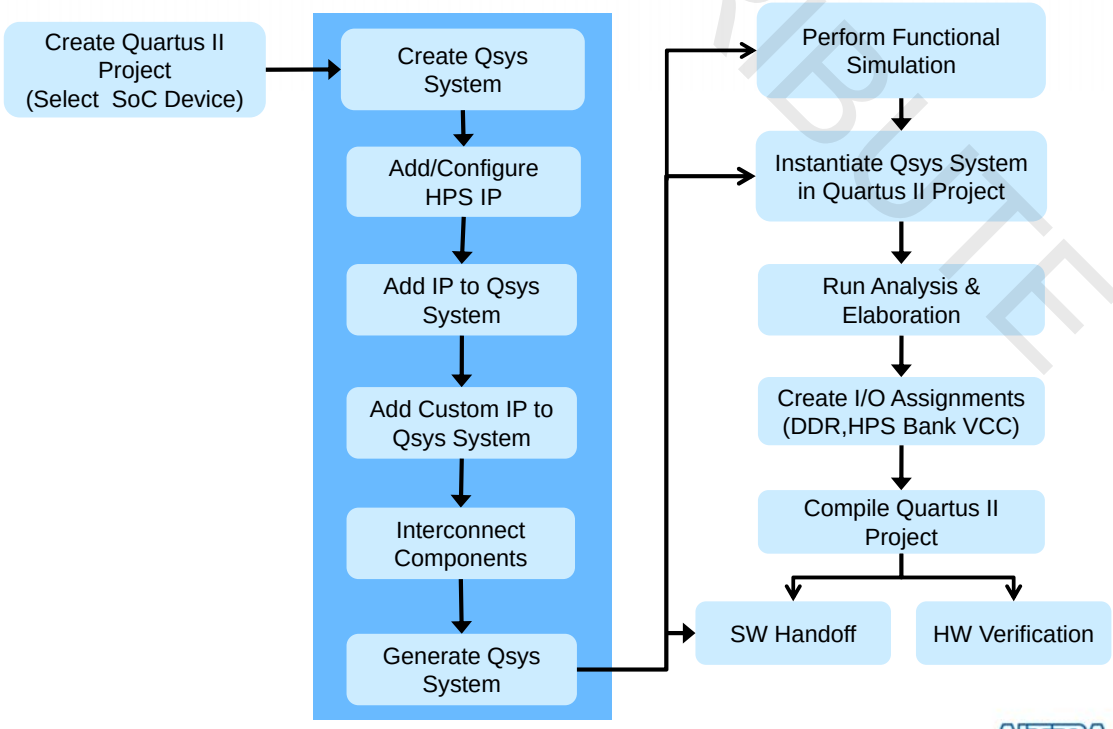
Designing with an ARM-based System on a Chip

Hardware Design

Hardware Design Agenda

- Hardware design flow with Qsys
- Configuring the HPS IP
- Software handoff
- Avalon/AXI overview
- HPS simulation
- HPS configuration and booting
- SoC debug

Typical Design Flow

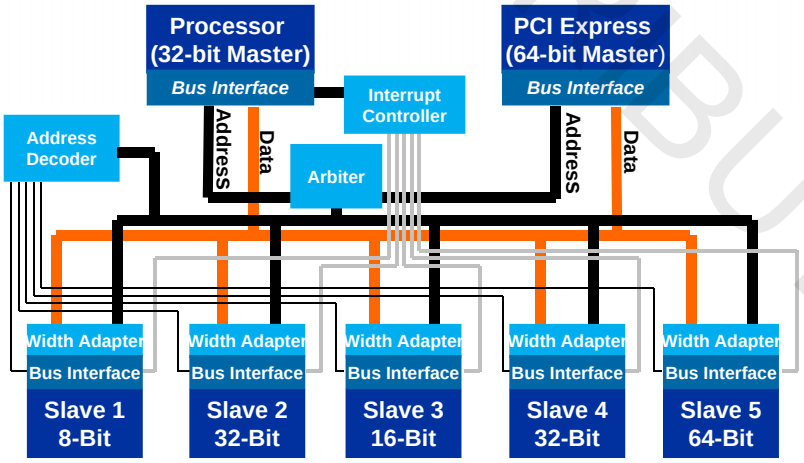


Qsys Flow

Qsys Tool

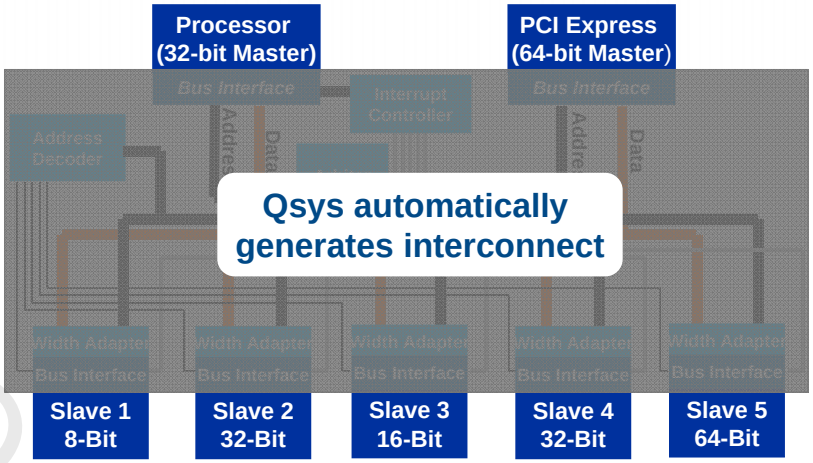
- GUI based tool for system design using IPs
- Qsys advantages
  - Simplifies complex system development
  - Reduces time to market
  - Raises the level of design abstraction
  - Enables design re-use
  - Scales easily to meet the needs of end product
- Provides a standard platform:
  - IP integration
  - Custom IP authoring
  - IP verification
- Generates fast and scalable network-on-a-chip interconnect
  - Allows standard interface interoperability
  - Allows simultaneous multi-mastering

Traditional System Design



- Components in system use different interfaces to communicate (some standard, some non-standard)
- Typical system requires significant engineering work to design custom interface logic
- Integrating design blocks and intellectual property (IP) is tedious and error-prone

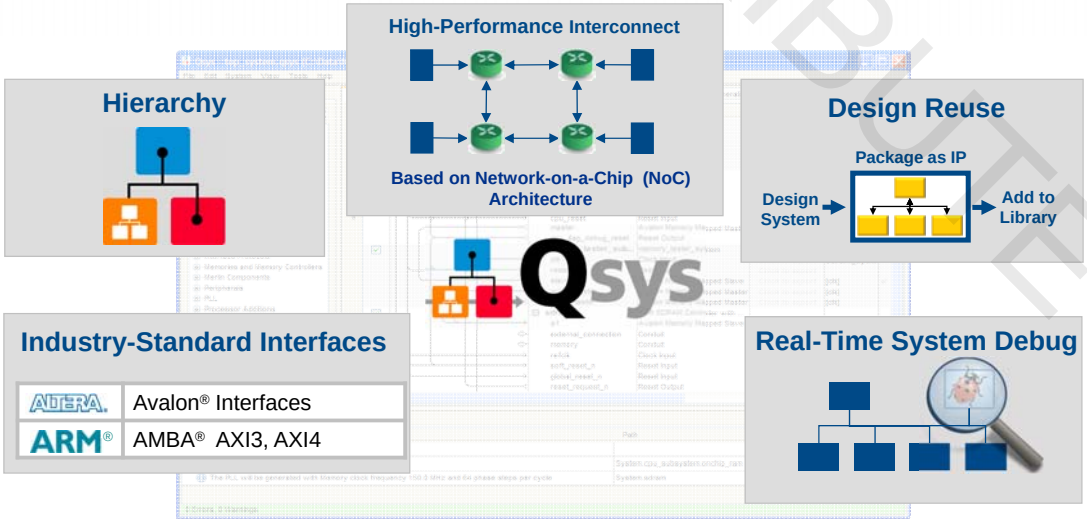
Automatic Interconnect Generation



- Avoids error-prone integration
- Saves development time with automatic logic & HDL generation
- Enables you to focus on value-add blocks

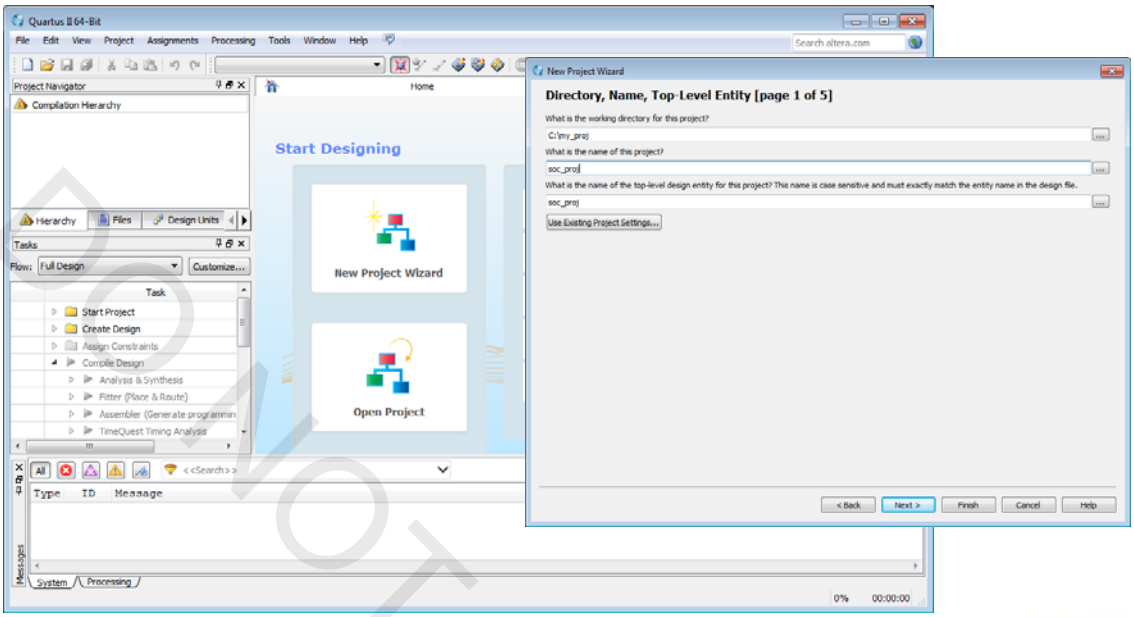
Qsys improves productivity by automatically generating the system interconnect logic

Qsys Features



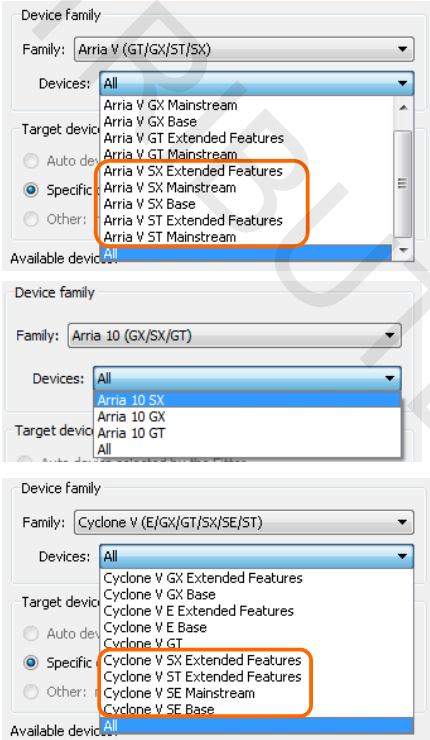
Create Quartus II Project for SoC Device

- Start with a New Quartus II Project



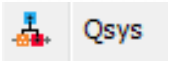
SoC Devices

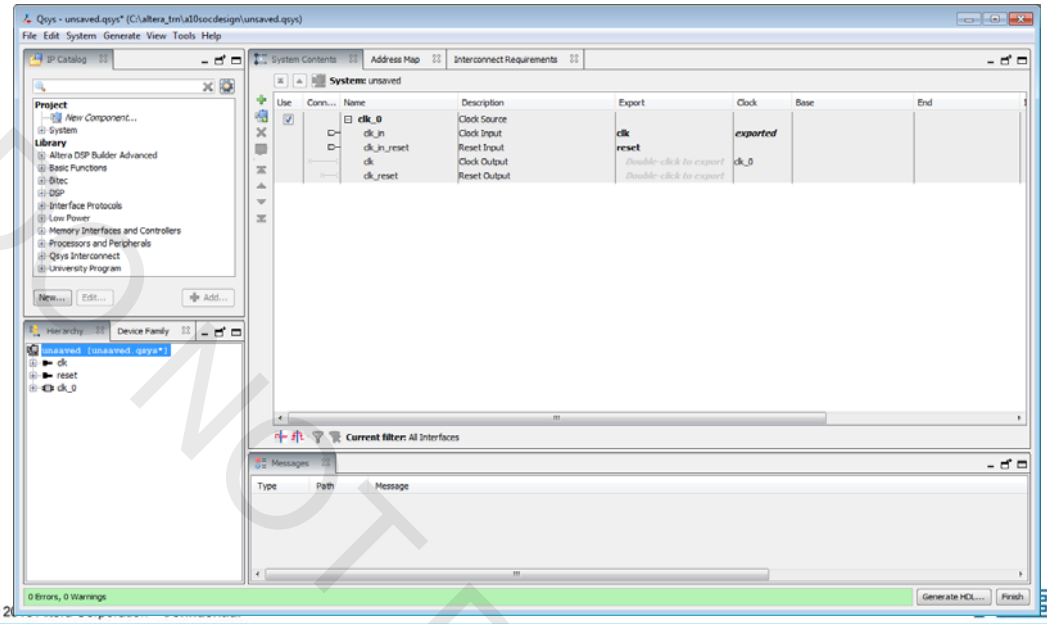
- Arria V SX
  - 6.5 Gbps transceivers
- Arria V ST
  - 10.3125 Gbps transceivers
- Arria 10 SX
  - 17.4 Gbps transceivers
- Cyclone V SX
  - 3.125 Gbps transceivers
- Cyclone V ST
  - 5 Gbps transceivers
- Cyclone V SE
  - No transceivers



Start a New System in Qsys

- Start Qsys from the Quartus II software
  - Menu, toolbar, or task pane





Qsys Component Library

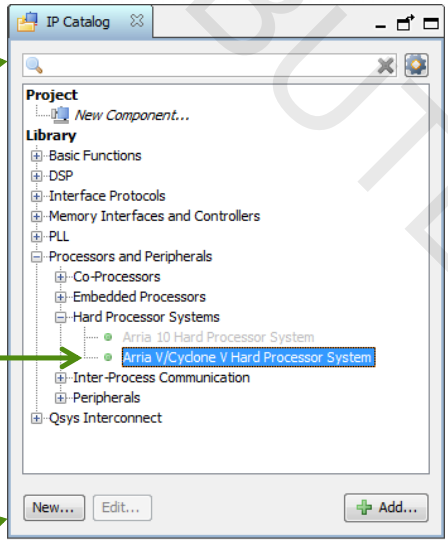
- Lists available IP and systems

Type search string to filter the list

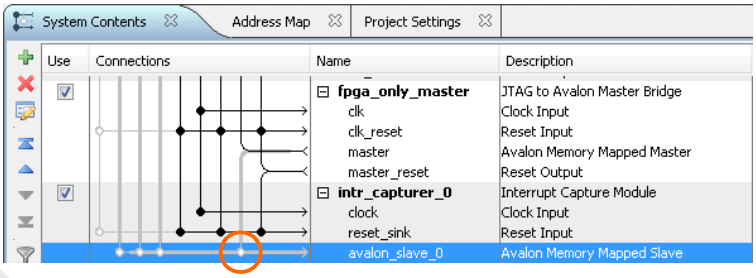
Expand categories to browse components

Double-click component or click **Add…** button to add selected component to system

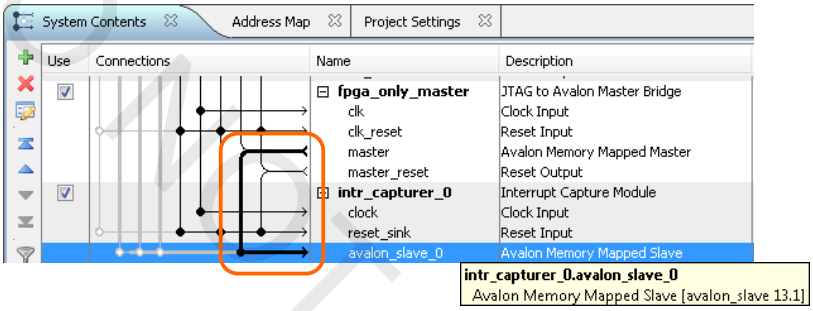
Bring in custom IP



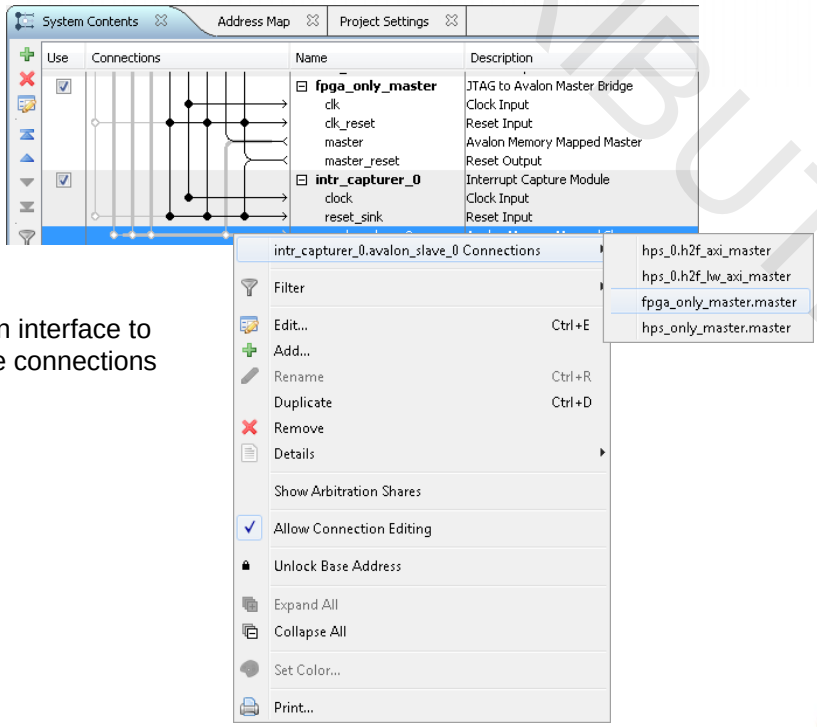
Connect the Components: Method 1



Click the open dot to make a connection

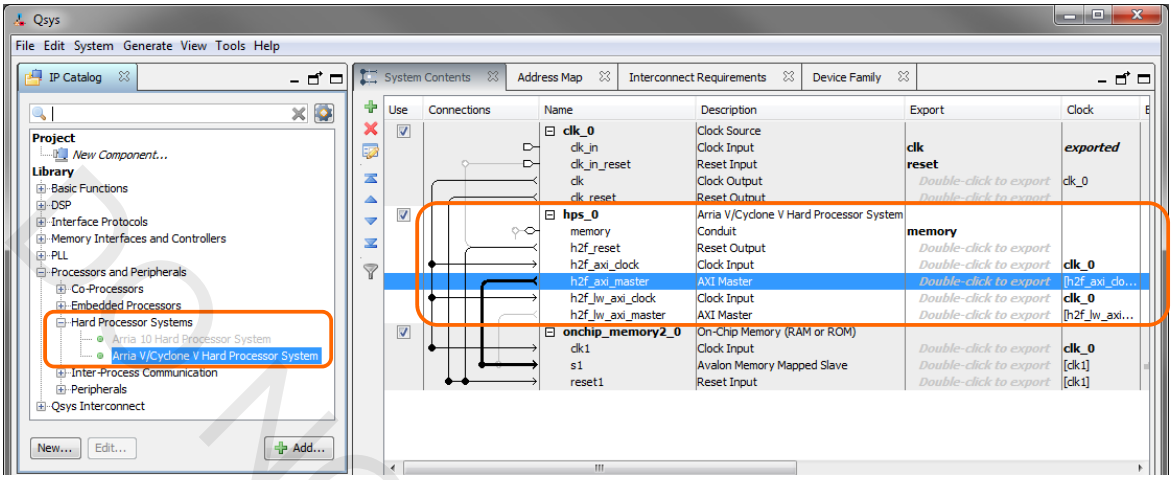


Connect the Components: Method 2



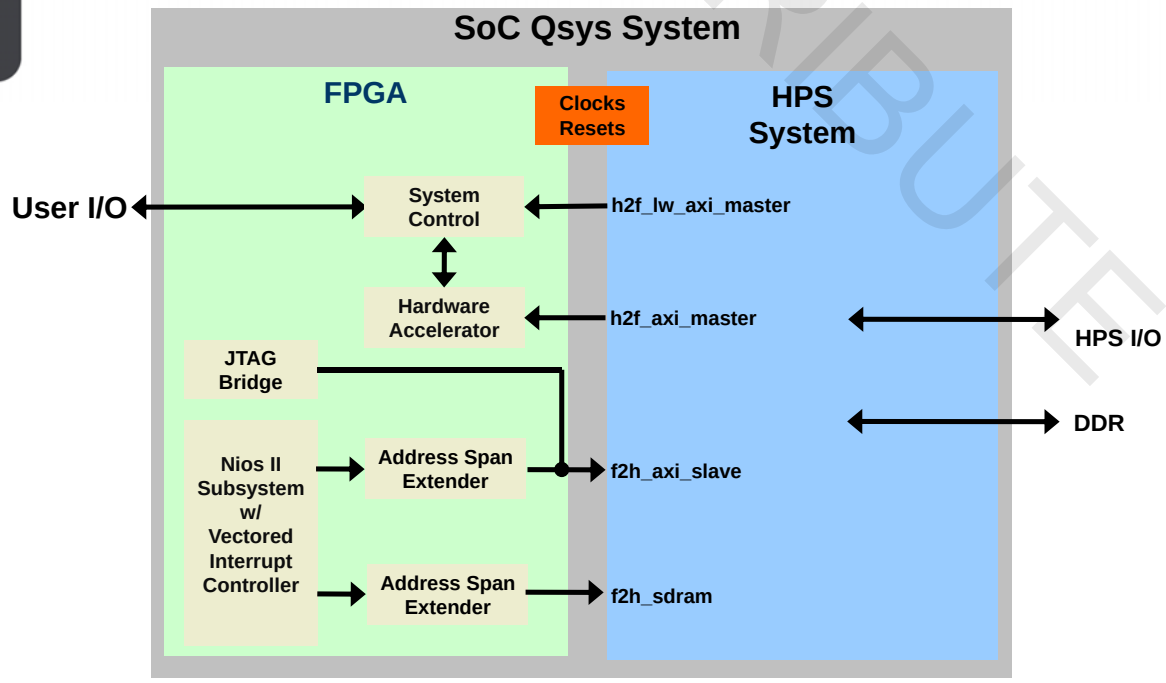
Right-click an interface to make eligible connections

Instantiating HPS in Qsys



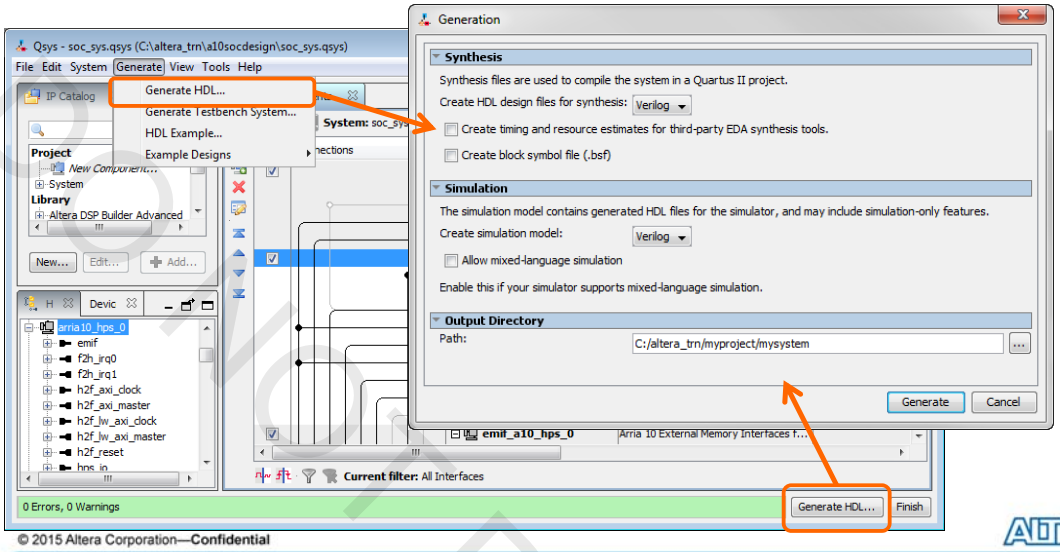
Back

Create Complete FPGA System in Qsys



Generate Completed Qsys System

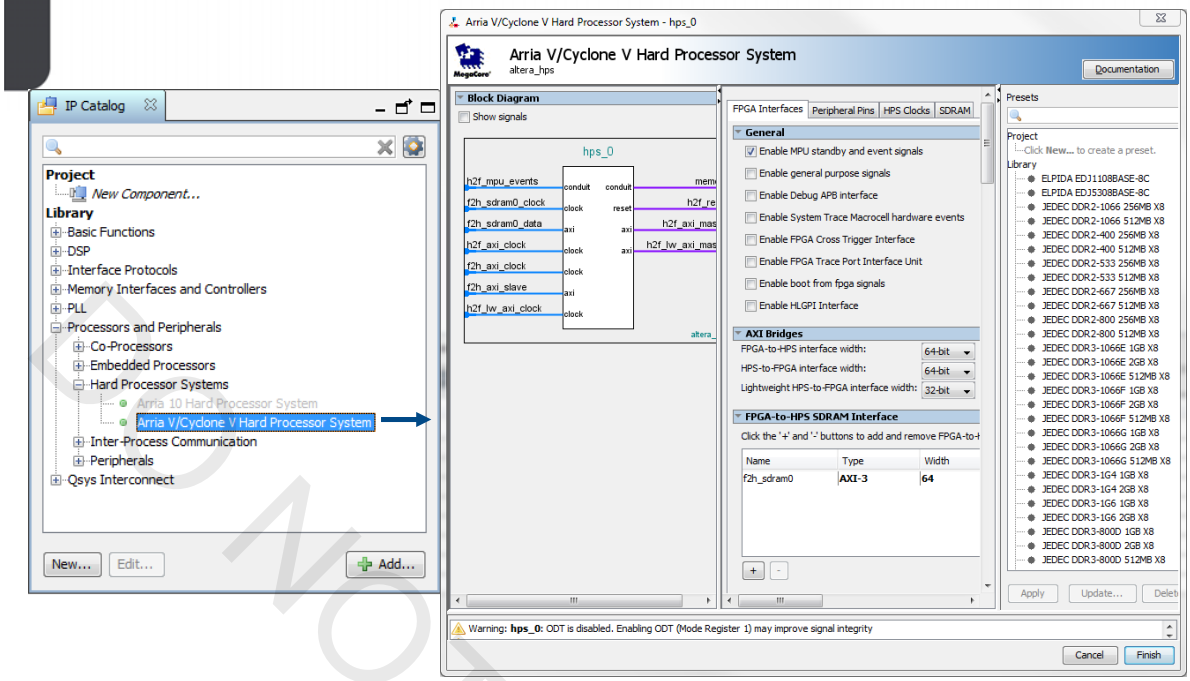
- Creates the Qsys interconnect
- Generates source files for synthesis and/or simulation
- Creates software handoff files



Hardware Design Agenda

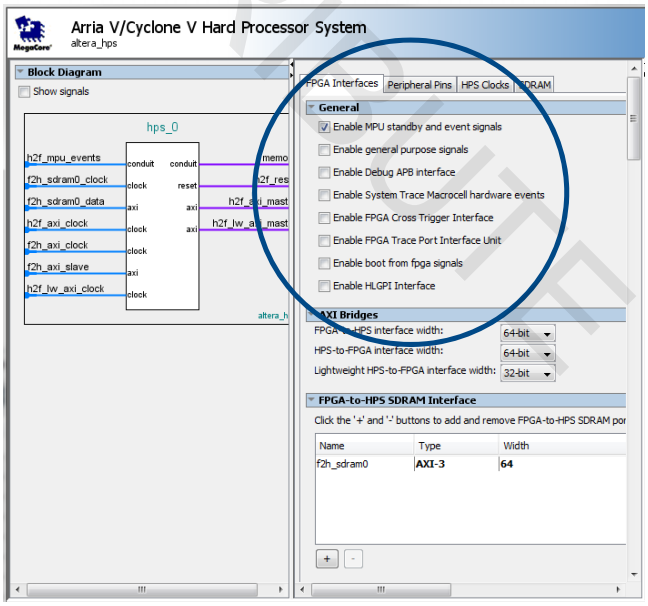
- Hardware design flow with Qsys
- Configuring the HPS IP
- Software handoff
- Avalon/AXI overview
- HPS simulation
- HPS configuration and booting
- SoC debug

Hard Processor System Component



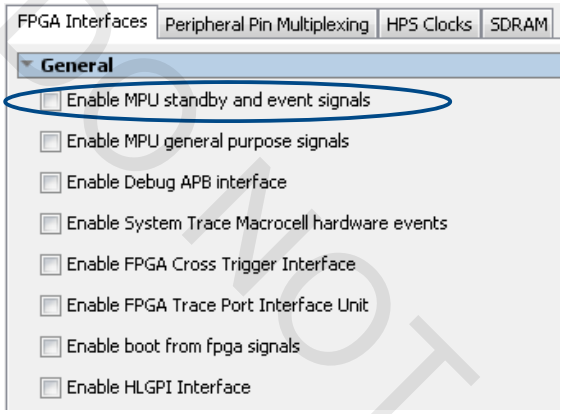
General Options & Boot Control

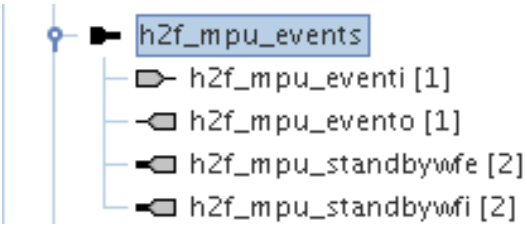
- Events
  - Event in and out
  - Wait for event condition
  - Wait for interrupt condition
- GPIO
- Debug interfaces
- Boot from FPGA



Events

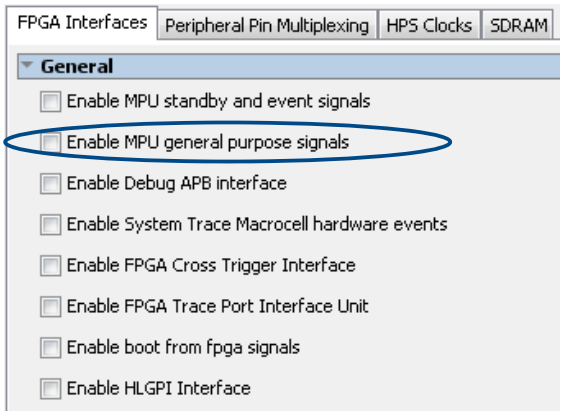
- Event in – sends event to both Cortex-A9 cores
- Event out – either cores has executed single event
- Wait for event condition (status bit)
- Wait for interrupt condition (status bit)

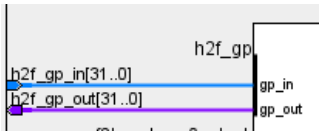




General Purpose IO

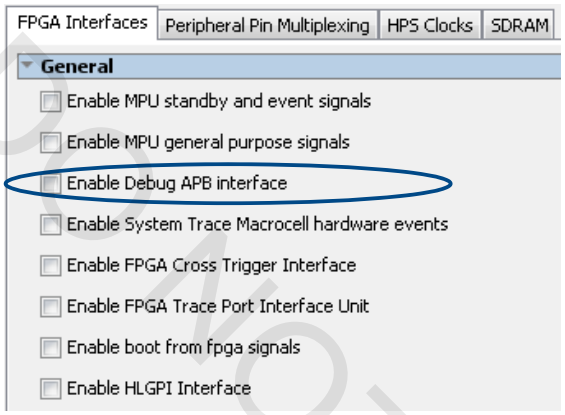
- Input bus 32 bits
- Output bus 32 bits
- Separate GPIO than shared HPS I/O pins
- Controlled through FPGA Manager





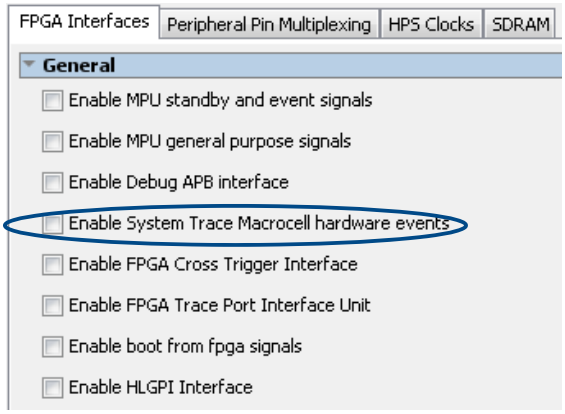
Debug APB

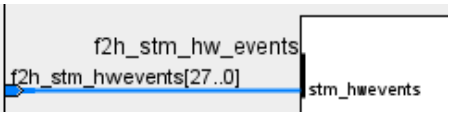
- Clock input and reset output
- APB interface
  - Address 31 indicates CPU or DAP access



System Trace Macrocell *(discussed later)*

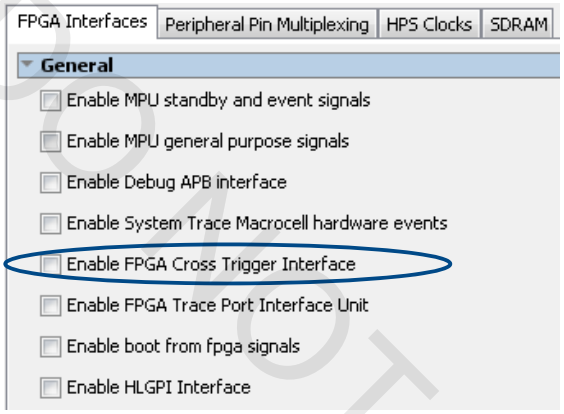
- Allows FPGA HW to insert messages into the trace
- Event bus - 28 bits
- Captures rising edge events

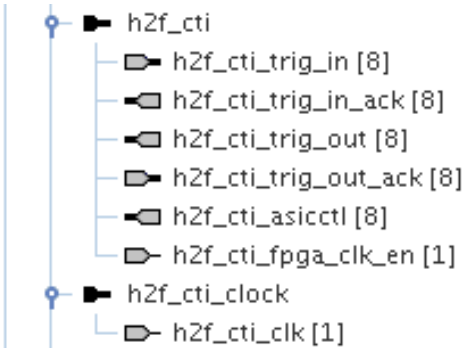




Cross Trigger Interface *(discussed later)*

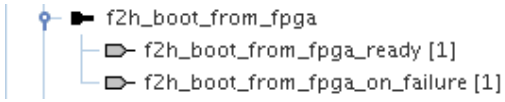
- Allows cross-trigger capability between FPGA hardware and software debugger
- Sends & receives triggers
  - Trigger input/output buses
  - Handshake interface

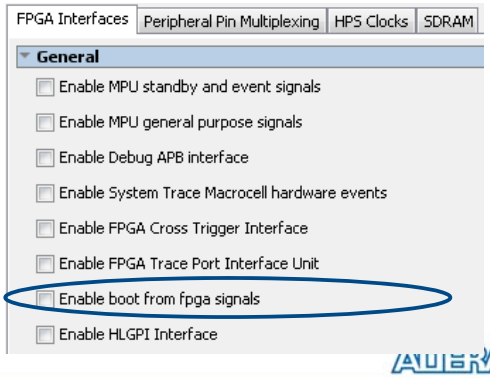




Boot from FPGA Signals

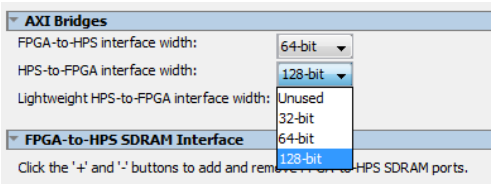
- Provides inputs monitored by Boot ROM
  - FPGA Boot device ready
  - Allow boot from FPGA if selected boot device fails
- FPGA system must be active and ready
  - PLL driving boot device must be locked
- Boot ROM will pass SW control to FPGA



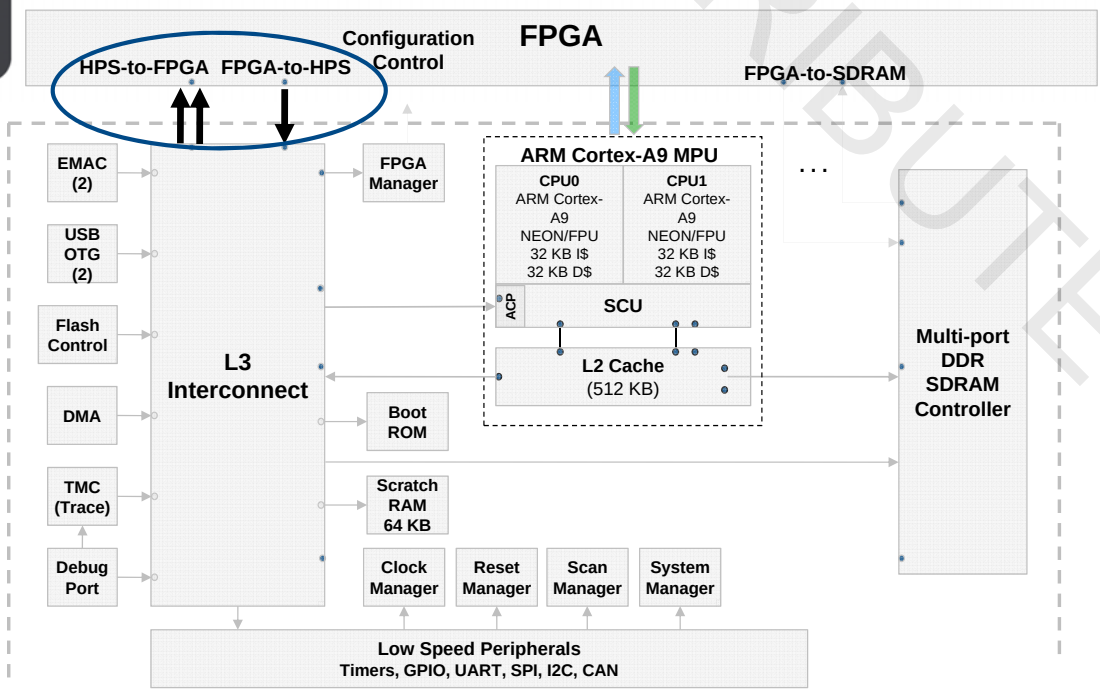


AXI Bridges

- FPGA-to-HPS
  - Access peripherals & memory
  - 4 GB space
  - 32, 64, or 128 bits wide
- HPS-to-FPGA
  - 960 MB space
  - 32, 64, or 128 bits wide
- Lightweight HPS-to-FPGA
  - Lower performance (32 bits)
    - Latency sensitive
    - No bursting
  - Accessing CSRs
  - 2 MB space
- Support soft Avalon connections

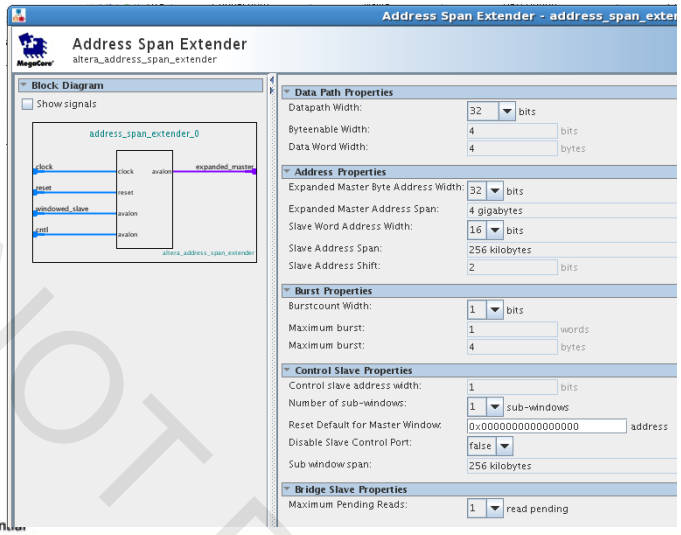


FPGA-HPS Bridge Interfaces



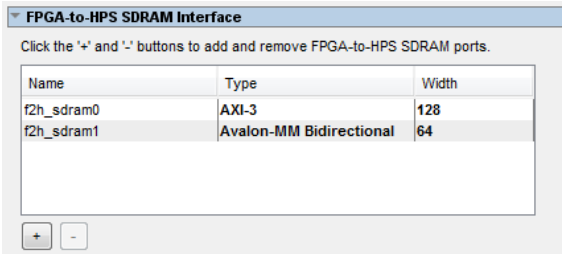
Accessing HPS Memory Space from FPGA

- 4 GB HPS memory map (FPGA to HPS Bridge)
- Less capable FPGA masters need Address Span Extender (Windowed Bridge)



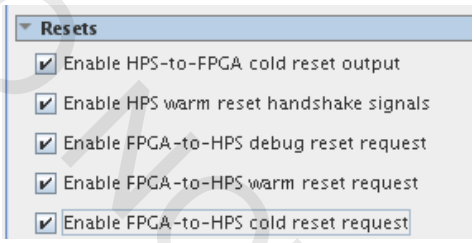
FPGA-to-HPS SDRAM Interface

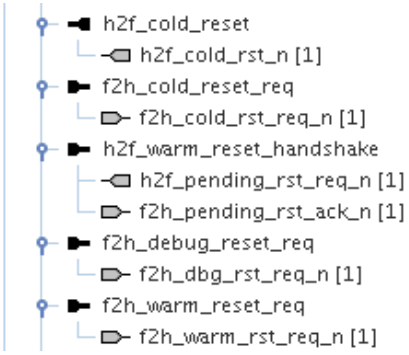
- Cyclone V and Arria V SoCs
  - AXI3 or Avalon-MM ports
  - Supports up to 6 ports
    - Maximum of 3 AXI3 ports or
    - Maximum of 6 Avalon-MM ports
  - Data widths: 32, 64, 128, 256
- Arria 10 HPS supports up to 3 ports
  - AXI protocol only
  - 32, 64, or 128 bit wide



Resets

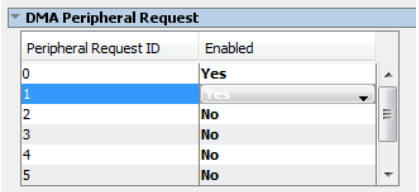
- Different reset domains
  - Cold
  - Warm
  - Debug
- HPS can drive resets to FPGA
- FPGA can drive resets

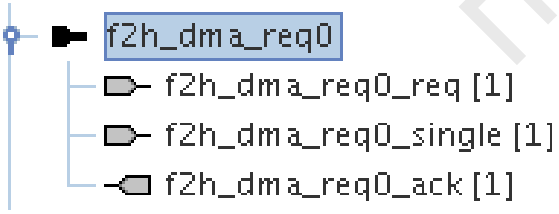




DMA Control

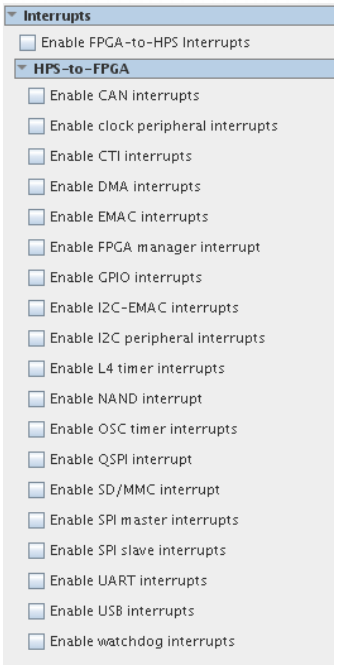
- Eight logical channels
  - Four available exclusive for FPGA
  - Four shared between CAN or FPGA
    - Becomes FPGA exclusive on parts without CAN
- Request single or burst transfer





Interrupts

- FPGA-to-HPS (64 interrupt inputs to GIC)
- HPS peripheral interrupt outputs to FPGA

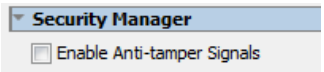


Generic Interrupt Controller (GIC) Overview

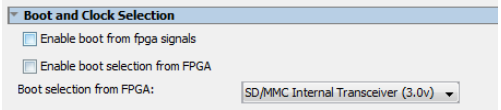
- Up to 180 interrupt sources
  - Shared by both Cortex-A9 processors
  - 64 FPGA interrupts, DMA, Peripherals, Parity/ECC, Debug
- 16 banked Software Generated Interrupts (SGIs) per A9 core
  - Used for issuing events to other core
    - ex: waking up processor core from sleep
  - Priority set by SGI receiver
- 16 banked private peripheral interrupts
  - Watchdog timers, general-purpose timers
- Programmable priority levels for each interrupt
  - 16 levels for non-secure
  - 16 levels for secure
  - Two levels deep
  - Priority mask will filter out lower priority interrupts
  - Highest priority is 0

Additional Arria 10 Interface Options

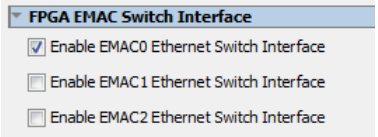
- Enable Anti-tamper signals
  - Allows communication between FPGA anti-tamper logic and the HPS security manager



- Enable Boot selection from FPGA
  - Allow boot selection (BSEL) tie-off from the FPGA instead of the BSEL pins

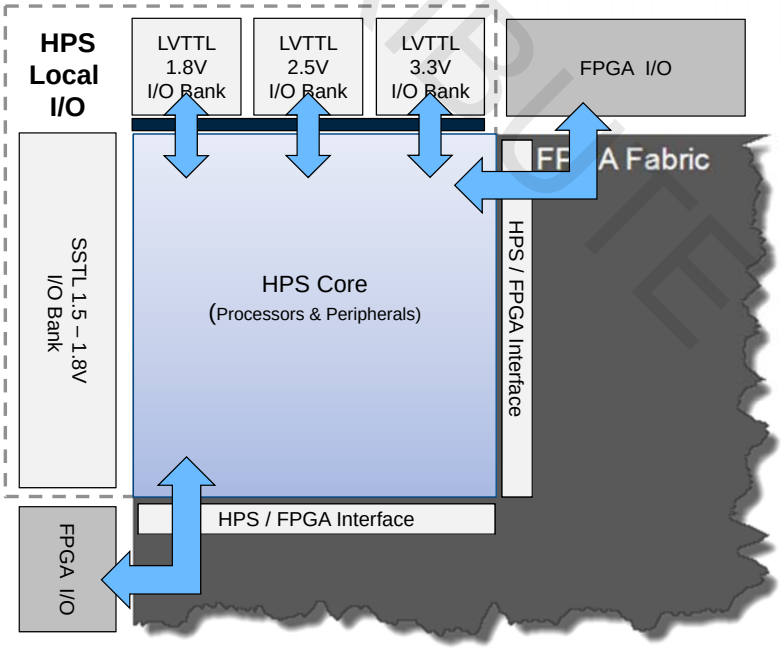


- Enable FPGA EMAC(1-3) Switch Interface
  - Provides direct connectivity from FPGA soft logic to any of the HPS EMAC peripherals, bypassing the L3 interconnect



HPS and FPGA IOs (Cyclone V/Arria V SoCs)

- A portion of device I/O are allocated to the HPS peripherals
- HPS peripherals assigned to HPS I/O
- HPS peripherals can be routed to FPGA
- FPGA logic can be routed to HPS I/O



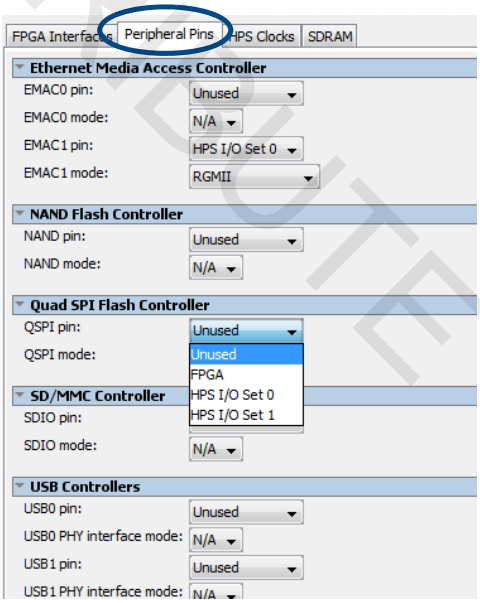
Arria 10 HPS IOs

- 17 Dedicated HPS IO
  - For Clk, Resets, Flash devices, I2C, UART, SPI
  - Configured by software
- 48 Shared IOs for peripherals
  - Configured as part of FPGA IO
  - Shared with FPGA logic
- Pins for 64bit DDR with ECC
  - Configured as part of FPGA IO
- Compared with Cyclone V/Arria V SoCs
  - Where all HPS IOs are dedicated

Peripheral Pins Options

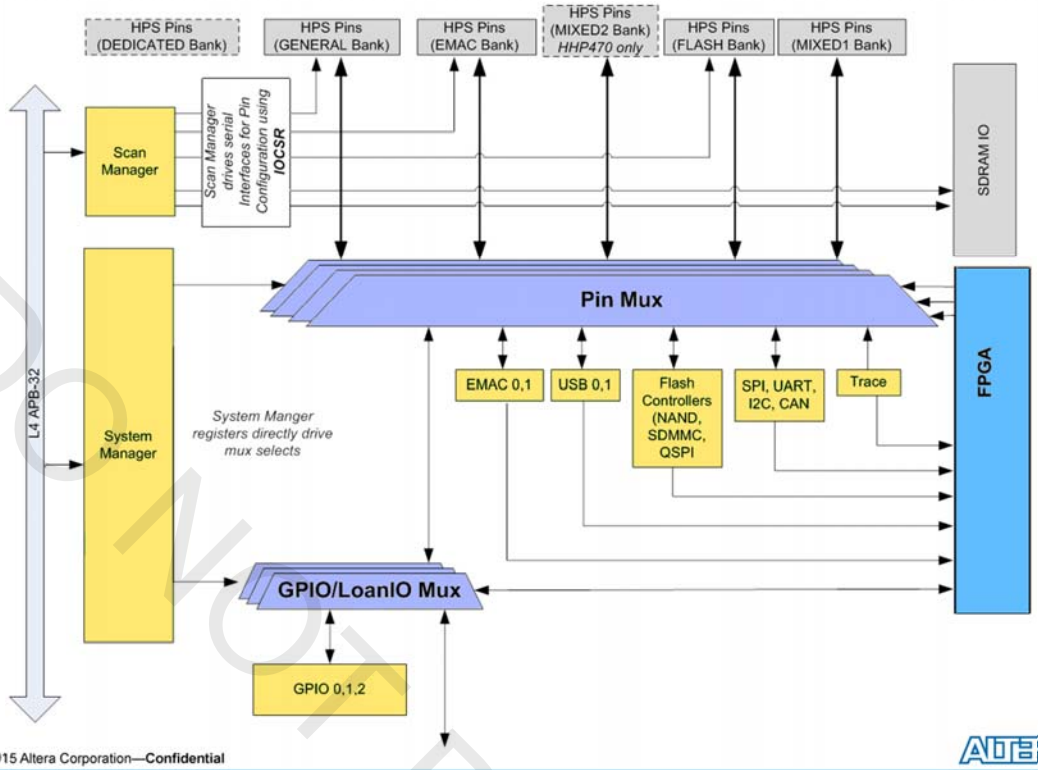
For Arria V and Cyclone V SoCs

- Enable peripheral interfaces
- Choose peripheral modes
- Select I/O set
  - More peripherals than available I/O
  - GPIO pins shared with peripherals



HPS I/O Pin Muxing Diagram

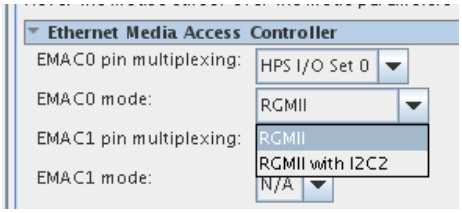
For Arria V and Cyclone V SoCs



Ethernet

For Arria V and Cyclone V SoCs

- Two Ethernet cores
- I/O Select
  - Arria V SoC
    - EMAC0 one
    - EMAC1 two
  - Cyclone V SoC – one for each EMAC core
- Presently only RGMII support
- Optionally select MDIO or I^{2}C PHY management interfaces



Other Peripheral Options

For Arria V and Cyclone V SoCs

- QSPI
  - Two I/O Sets
  - Choose 1, 2 or 4 slave selects (up to 4 devices)
- SPI master
  - Single or dual slave selects
- UART
  - Enable/Disable flow control

Pin Usage & Conflicts

For Arria V and Cyclone V SoCs

- View pin mux usage

Peripherals Mux Table

PinName	mux_select_1	mux_select_2	mux_select_3	* GPIO	* LoanIO
RGMII0_TX_CLK			EMAC0.TX_CLK (Set0)	GPIO00	LOANIO00
RGMII0_TXD0		USB1.D0 (Set0)	EMAC0.TXD0 (Set0)	GPIO01	LOANIO01
RGMII0_TXD1		USB1.D1 (Set0)	EMAC0.TXD1 (Set0)	GPIO02	LOANIO02
RGMII0_TXD2		USB1.D2 (Set0)	EMAC0.TXD2 (Set0)	GPIO03	LOANIO03
RGMII0_TXD3		USB1.D3 (Set0)	EMAC0.TXD3 (Set0)	GPIO04	LOANIO04

- Enable GPIO

Peripherals Mux Table

PinName	mux_select_1	mux_select_2	mux_select_3	* GPIO
RGMII0_TX_CTL			EMAC0.TX_CTL (Set0)	GPIO09
RGMII0_RX_CLK		USB1.CLK (Set0)	EMAC0.RX_CLK (Set0)	GPIO10
RGMII0_RXD1		USB1.STP (Set0)	EMAC0.RXD1 (Set0)	GPIO11
RGMII0_RXD2		USB1.DIR (Set0)	EMAC0.RXD2 (Set0)	GPIO12
RGMII0_RXD3		USB1.NXT (Set0)	EMAC0.RXD3 (Set0)	GPIO13

- View conflicts

Peripherals Mux Table

PinName	mux_select_1	mux_select_2	mux_select_3
NAND_ALE	QSPI.SS3 (Set1) (Set0)	EMAC1.TX_CLK (Set0)	NAND.ALE (Set0)
NAND_CE	USB1.D0 (Set1)	EMAC1.TXD0 (Set0)	NAND.CE (Set0)
NAND_CLE	USB1.D1 (Set1)	EMAC1.TXD1 (Set0)	NAND.CLE (Set0)

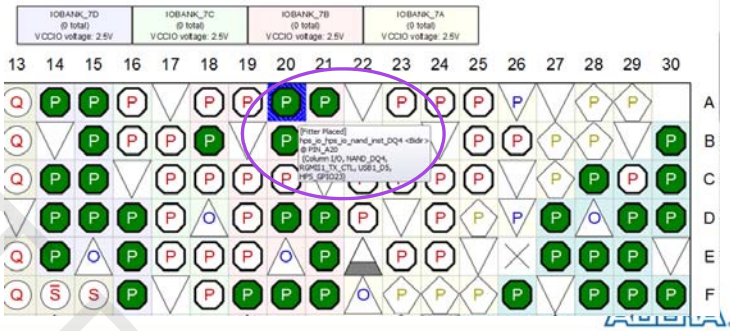
Error: **hps_0**: Refer to the Peripherals Mux Table for more details. The selected peripherals 'EMAC1' and 'NAND' are conflicting.

Info: **hps_0**: ECC will be enabled in the preloader because an interface width of 24 or 40 has been chosen.

HPS Pin Assignments

For Arria V and Cyclone V SoCs

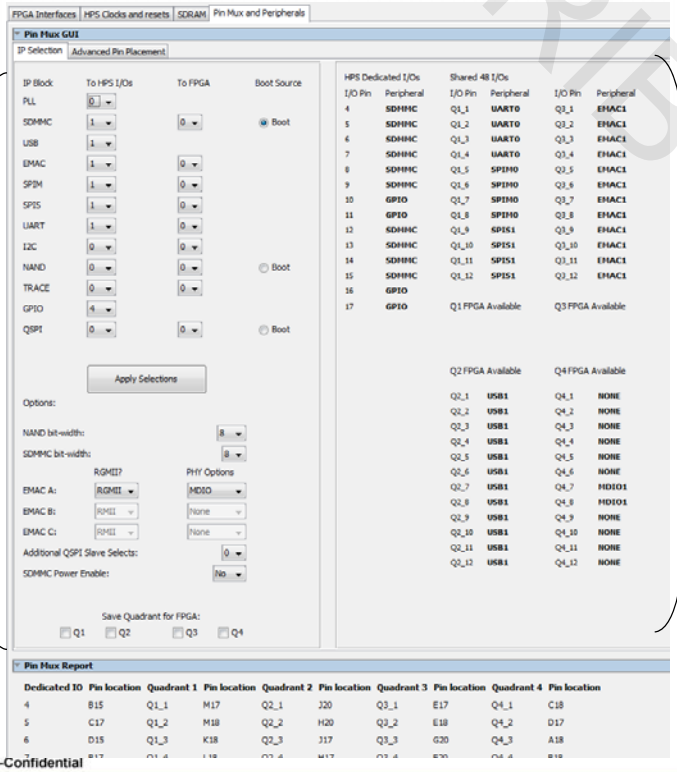
- Qsys automatically assigns HPS pins
  - pin mux settings transferred to the Quartus II project
- Run `hps_sdram_p0_pin_assignments.tcl` in Quartus to set up SDRAM I/O assignments
  - Located in the `<project>\<qsys system>\synthesis\submodules` directory
  - Tcl file generated during Qsys system generation
- Check Assignments in Quartus II pin planner
  - Location
  - Drive strength
  - VCCIO for banks



Arria 10 Pin Mux GUI

For Arria 10SoCs

Enable Peripherals and Options



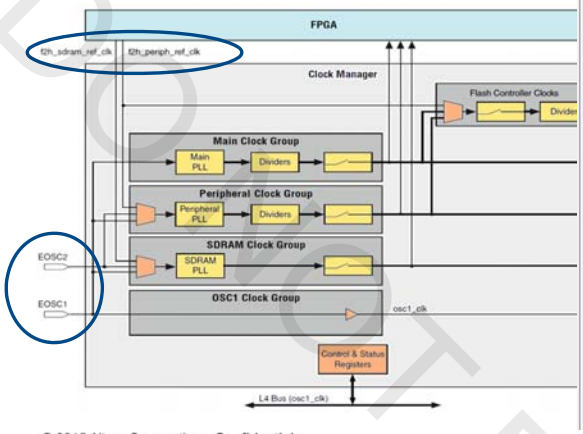
I/O Usage displayed

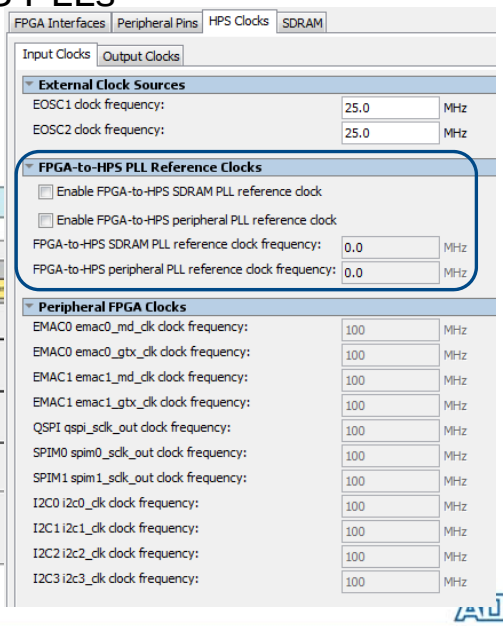
Finer I/O selection options available under Advanced Pin Placement tab

HPS Input Clocks Options

- Specify HPS clock pin frequency
- Drive FPGA clocks into HPS PLLs
  - Peripherals
  - SDRAM

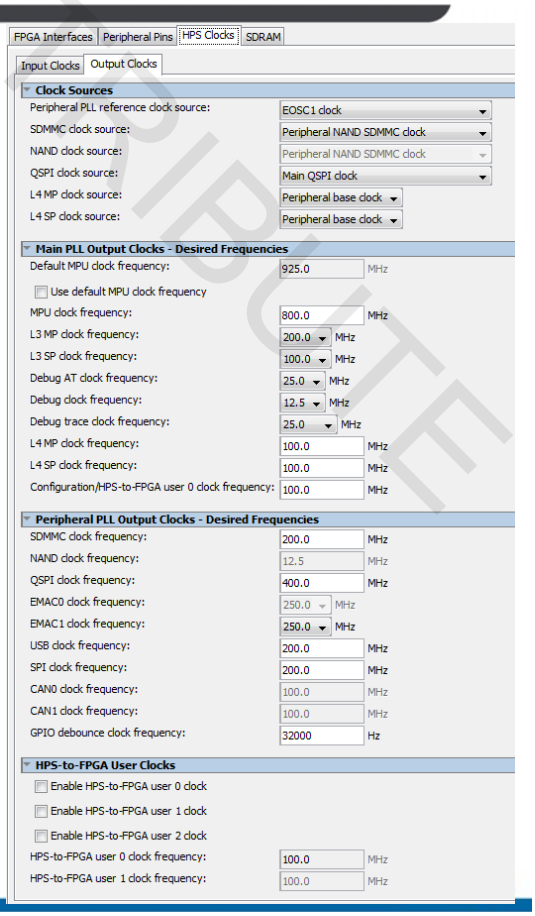
Options transferred to 2nd stage bootloader





HPS Output Clocks Options

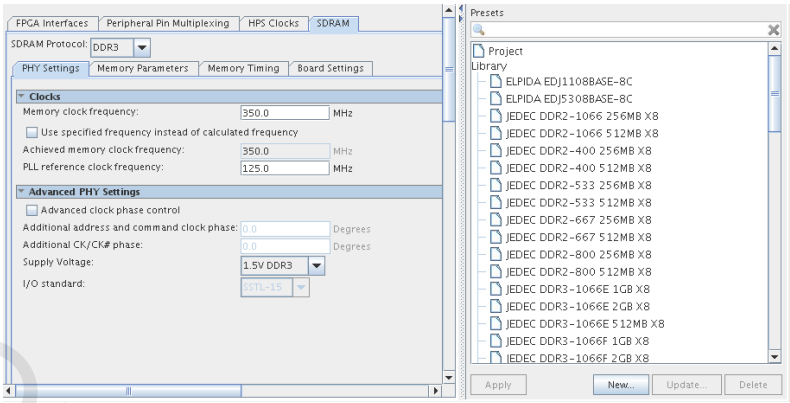
- Specify clock mux options
- Specify peripheral clock frequencies
- Enable HPS clocks into the FPGA



Cyclone V/Arria V SDRAM Configuration GUI

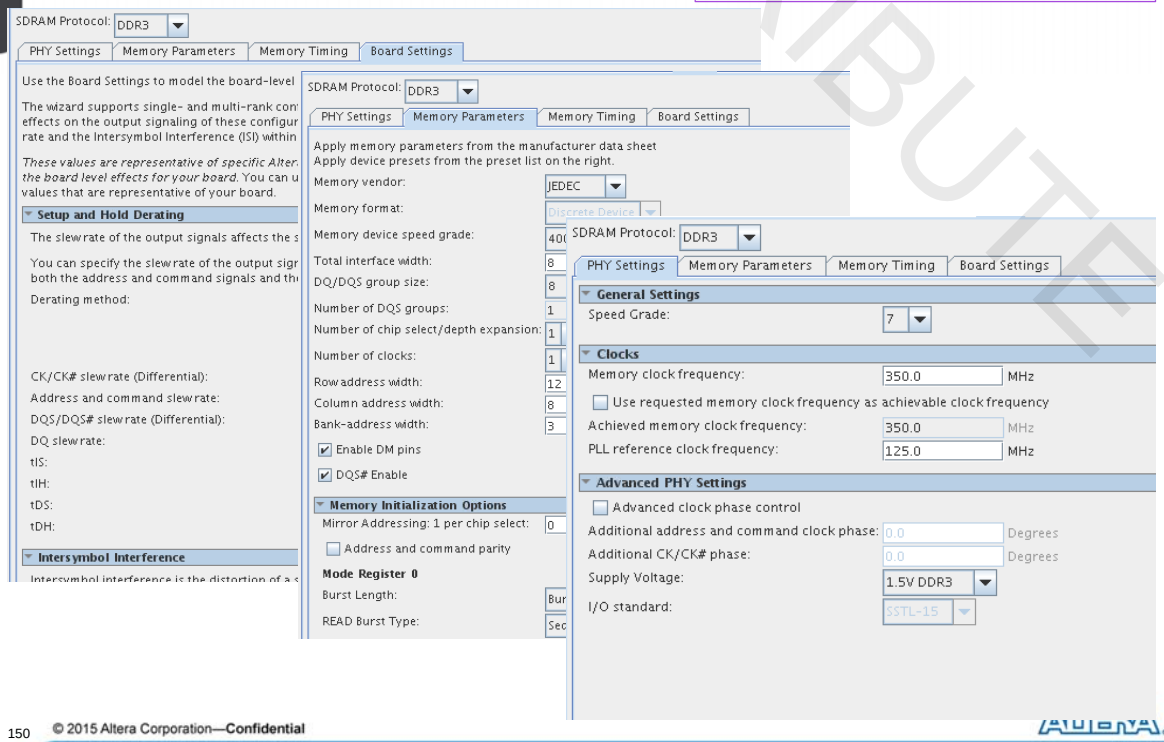
For Arria V and Cyclone V SoCs

- Consistent with SDRAM Controller MegaWizard™ GUI
- Supported memory devices
  - DDR3
  - DDR2
  - LPDDR2
- Configure clock & initial settings



SDRAM Embedded Memory Interface

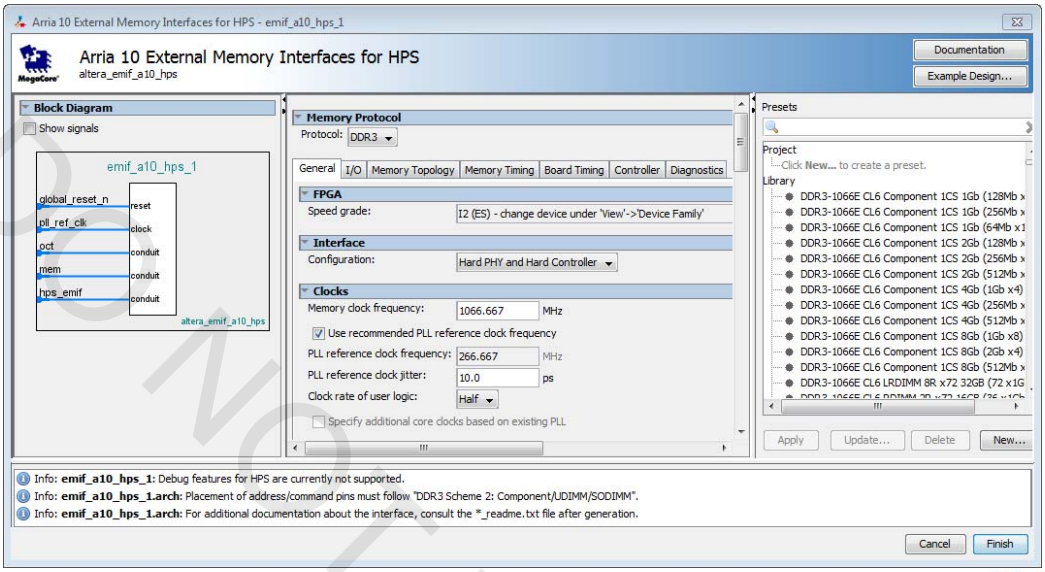
For Arria V and Cyclone V SoCs



Arria 10 SDRAM Parameterization

For Arria 10SoCs

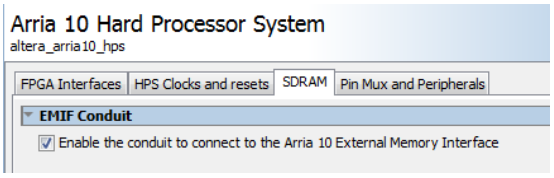
- Must use Arria 10 External Memory Interfaces
  - Separate Qsys component to be connected to the HPS



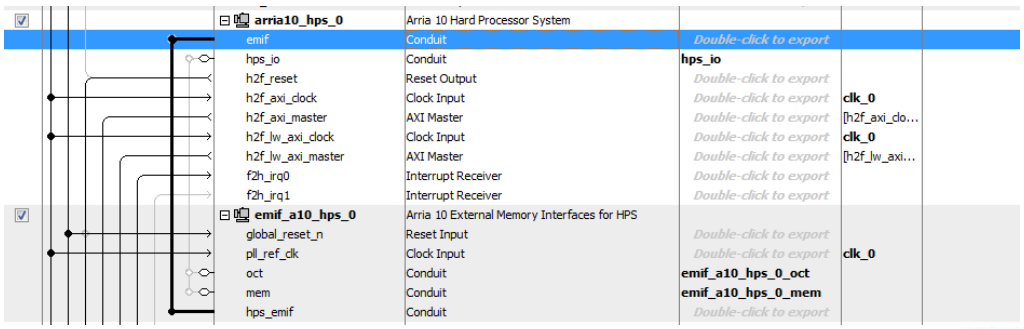
Arria 10 SDRAM Connection

For Arria 10SoCs

- Enable conduit to the Arria 10 External Memory Interface
  - Allows connection to the Arria 10 External Memory Interface for HPS



- Make the conduit connection in the Qsys tool



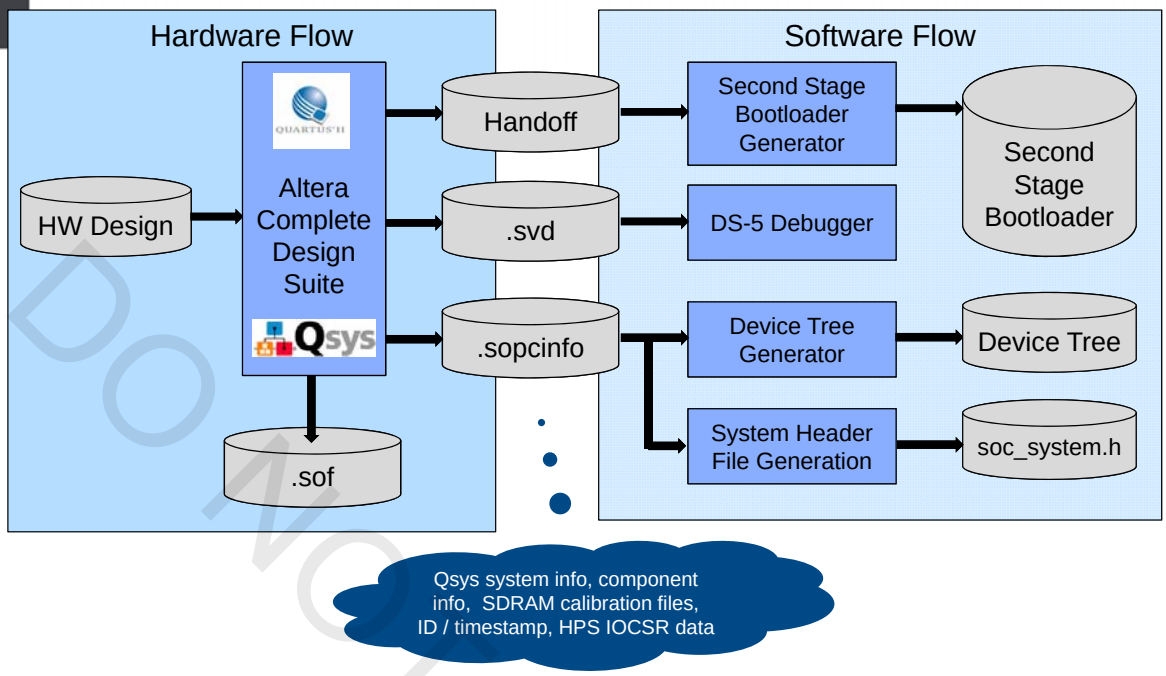
Arria 10 SDRAM Considerations

- Unnecessary to run SDRAM assignment script as relevant assignments are included in the .qip file
- SDRAM Controller is part of the FPGA
  - FPGA peripheral must be configured prior to software using the SDRAM
- Arria 10 bootloader can be generated to provide FPGA configuration support

Hardware Design Agenda

- Hardware design flow with Qsys
- Configuring the HPS IP
- Software handoff
- Avalon/AXI overview
- HPS simulation
- HPS configuration and booting
- SoC debug

Hardware/Software Design Flow Overview



Generated Software Handoff Files

- Created when Quartus project is compiled
  - *<quartus project>/hps_isw_handoff/*
- Used to create the second stage bootloader
- Information contained
  - SDRAM parameters
  - HPS modules and I/O usage
  - Checksum
  - For Cyclone V / Arria V SoC's
    - Binaries of IOCSR, SDRAM sequencing source software

Additional Generated Files

- Created when Qsys system is generated

File	Description
<qsys_system_name>_<hps instance>_hps.svd (located in the ***<system>\synthesis*** directory)	System View Description (SVD) file • Allows visibility into the register maps of Qsys peripherals from a debugging tool such as DS-5*
<QSYS_SYSTEM_NAME>.sopcinfo (located in the same directory as the .qsys file)	XML file describes FPGA hardware system 1. Used to create a system header file used to abstract away FPGA peripheral addresses 2. Used to generate device tree for Linux

SVD File for Custom Components

- IP component designer can create .svd file and attach to an interface in the `<component>_hw.tcl` file

```
set_interface_property <slave interface> CMSIS_SVD_FILE <file path>
```

- Ability to pass variable into .svd file from component instantiation through hw.tcl file

```
set_interface_property <slave interface> \
        CMSIS_SVD_VARIABLES "<Variable> <Variable value>"
```


Example System View Description File

- Create SVD file according to ARM Cortex Microcontroller Software Interface Standard (CMSIS) XML schema specification

```
<?xml version="1.0" encoding="utf-8"?>
<device schemaVersion="1.1" xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" xs:noNamespaceSchemaLocation="CMSIS-SVD_Schema_1_1.xsd" >
  <peripherals>
    <peripheral>
      <name>altera_avalon_sysid</name><baseAddress>0x00000000</baseAddress>
      <addressBlock>
        <offset>0x0</offset>
        <size>8</size>
        <usage>registers</usage>
      </addressBlock>
      <registers>
        <register>
         <name>ID</name>
         <displayName>System ID</displayName>
           <description>A unique 32-bit value that is based on the contents of the QSys system. The id is similar to a check-sum value; QS
         <addressOffset>0x0</addressOffset>
         <size>32</size>
         <access>read-only</access>
         <resetValue>${sysid_id_value}</resetValue>
         <resetMask>0xffffffff</resetMask>
         <fields>
           <field><name>id</name>
            <bitOffset>0x0</bitOffset>
            <bitWidth>32</bitWidth>
            <access>read-only</access>
        </field>
       </fields>
     </register>
```

Component Information

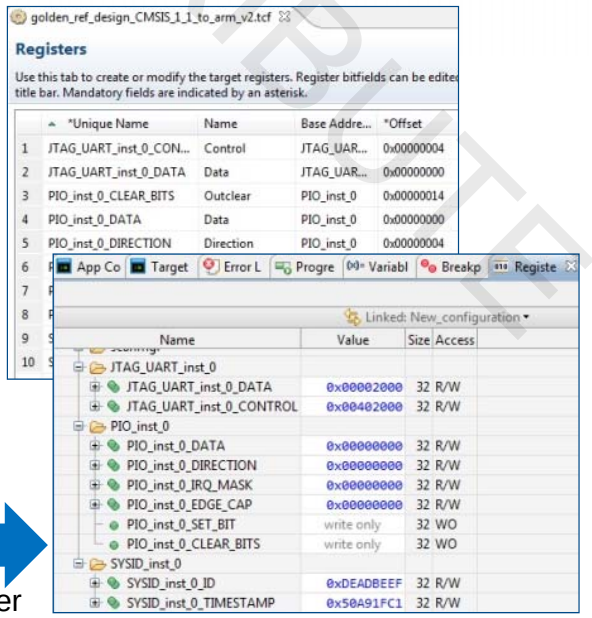
Register information

Register description

Value from component instantiation through hw.tcl

Visualization of SoC Peripherals

- Register views of FPGA peripherals
  - Point DS-5 debugger to Qsys-generated svd file
- Allows for debug of software drivers
  - Self-documenting
  - Grouped by peripheral, register and bit-field



Qsys

CMSIS SVD File

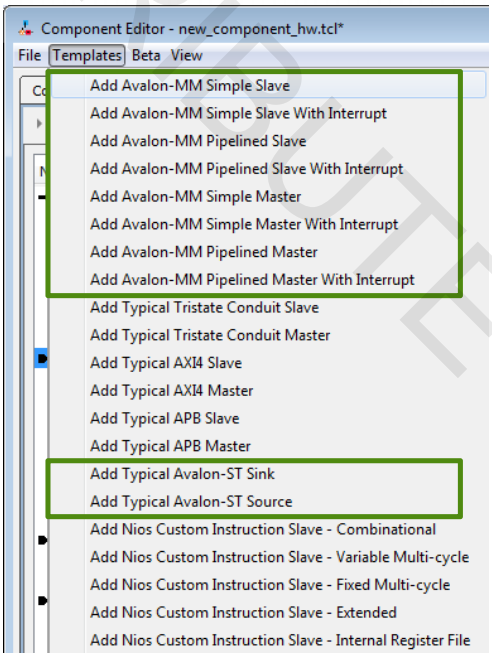
Peripheral register descriptions

Hardware Design Agenda

- Hardware design flow with Qsys
- Configuring the HPS IP
- Software handoff
- Avalon/AXI overview
- HPS simulation
- HPS configuration and booting
- SoC debug

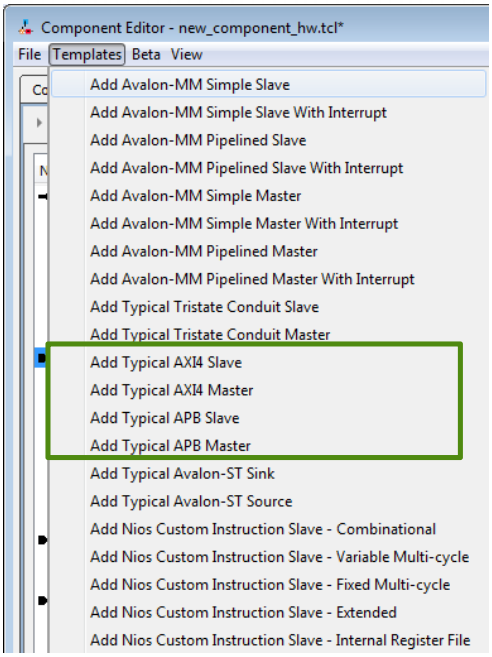
Qsys - Supported Interfaces – Altera Avalon

- Altera Avalon-MM
  - Memory mapped
  - Control plane
  - Master initiates requests to slave
- Altera Avalon-ST
  - Streaming
  - Data plane
  - Source interface sends data to sink interface (point-to-point)



Qsys - Supported Interfaces – ARM AMBA

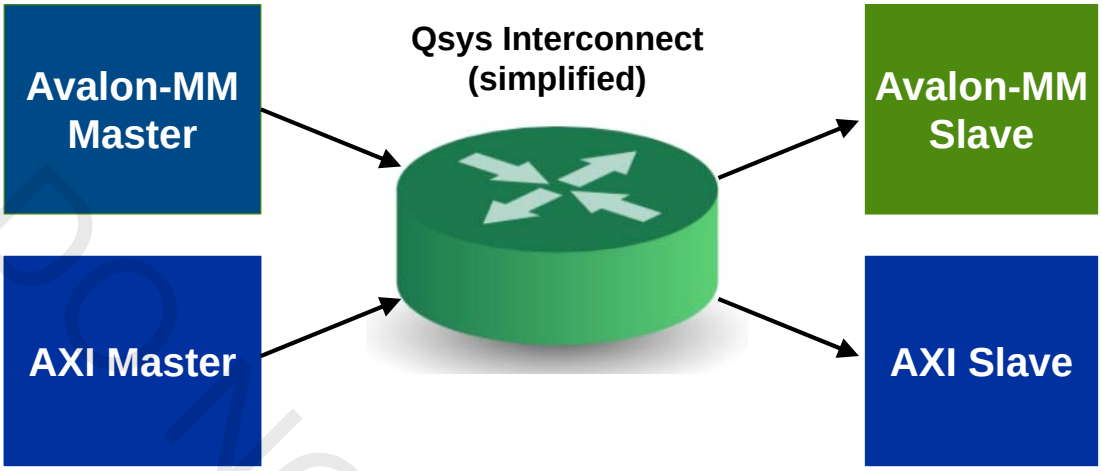
- ARM AMBA AHB interface
  - Memory mapped
  - High bandwidth control plane
  - Multiple bus masters
  - Burst and split transactions
- ARM AMBA APB interface
  - Memory mapped
  - Low bandwidth control plane
  - Similar to AHB interface, with a less complex signal list
- ARM AMBA AXI interface
  - Memory mapped
  - Includes support for AXI-Lite and AXI-Streaming interfaces
  - Both control and data plane
  - Burst and unaligned transactions



Advantages of Using Standard Interfaces

- Ensure compatibility between IP blocks from different design teams or vendors
  - Any component supporting interface can be connected
- Simplify design entry and team-based design
  - Signal behavior defined by interface
  - Improved understanding, simplified documentation
  - No manual wiring or mapping of control, data, and status signals
  - Fast system-level integration
  - Easy system changes
- Simplify interface verification
  - Use verification infrastructure to verify against standard
    - Bus functional models, interface compliance assertions and monitors, functional coverage

Standard Interface Example

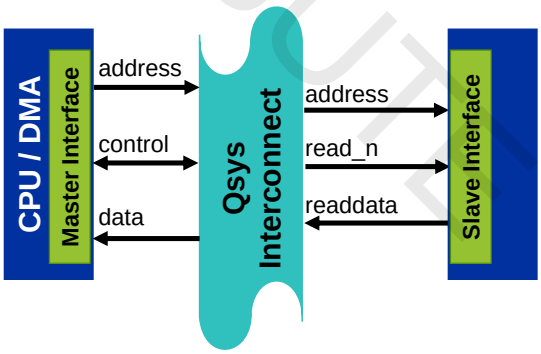


Any master interface can communicate with any slave interface

Avalon-MM Interfaces

- Master interfaces
  - Initiate read/write transfers to Qsys interconnect targeting slaves in its address space
- Slave interfaces
  - Respond to transfer requests from Qsys interconnect
- Qsys interconnect handles
  - Address decoding
  - Data width matching
  - Arbitration
  - Clock crossing
  - Timing adaptation
- All interfaces must be associated with a clock and reset

Example master/slave connections



Basic Avalon-MM Master Interface Signals

Signal Type	Width	Direction	Required	Description
address	1-64	Output	Y	Byte address corresponding to slave for transfer request
waitrequest waitrequest_n	1	Input	Y	Forces master to stall transfer until deasserted (other Avalon-MM signals must be held constant)
read read_n	1	Output	N	Indicates master issuing read request
readdata	1-1024	Input	N	Data returned from read request
write write_n	1	Output	N	Indicates master issuing write request
writedata	1-1024	Output	N	Data to be sent for write request
byteenable byteenable_n	1, 2, 4, …, 128	Output	N	Specifies valid byte lanes for readdata or writedata (width = data width / 8)

Basic Avalon-MM Slave Interface Signals

Signal Type	Width	Direction	Required	Description
address	1-64	Input	N	Word address of slave for transfer request
waitrequest waitrequest_n	1	Output	N	Allows slave to stall transfer until deasserted
read read_n	1	Input	N	Indicates slave should respond to read request
readdata	1-1024	Output	N	Response data provided to the Qsys interconnect
write write_n	1	Input	N	Indicates slave should respond to write request
writedata	1-1024	Input	N	Data from the Qsys interconnect for a write request
byteenable byteenable_n	1, 2, 4, … 128	Input	N	Specifies valid byte lanes for readdata or writedata (width = data width / 8)

Avalon Interface Specification

- Defines the entire Avalon Interface standard
- Provides reference information on additional transfer types
  - Use cases
  - Waveform diagrams

www.altera.com/literature/manual/mnl_avalon_spec.pdf

Advanced eXtensible Interface (AXI) Overview

- ARM standard has been licensed by Altera
  - Altera supports AXI3 and AXI4 specifications
  - HPS is AXI3 spec compliant
- Suitable for high-bandwidth and low-latency transfers
- Separate Read and Write channels
  - 3 Write Channels, 2 Read Channels
- Separate Address/Control and Data phases
- Identical handshake mechanisms for all channels
  - VALID/READY handshake
    - Source generates the VALID
    - Destination generates the READY
    - Handshake can be
      - VALID before READY
      - READY before VALID
      - VALID with READY

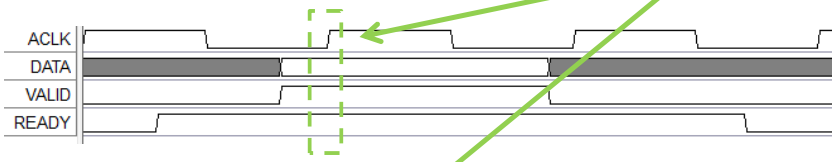
Additional AXI features

- Transaction ID's for out of order responses
- AwCACHE & ArCACHE
  - Define memory type (i.e. read, write, cacheable, bufferable)
- AwPROT & ArPROT
  - Indicates the privilege and security level of the transaction
  - Indicates whether the transaction is a data access or an instruction access
- AxLock
  - Allows atomic accesses to AXI slaves (i.e. locked, exclusive, normal)
  - Response signaling notifies master if can't be accessed

Individual Channel Handshake Examples

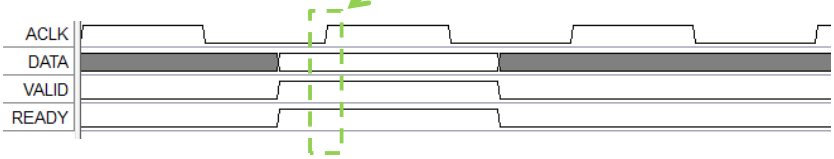
- **READY** active after **VALID**

- **READY** active before **VALID**

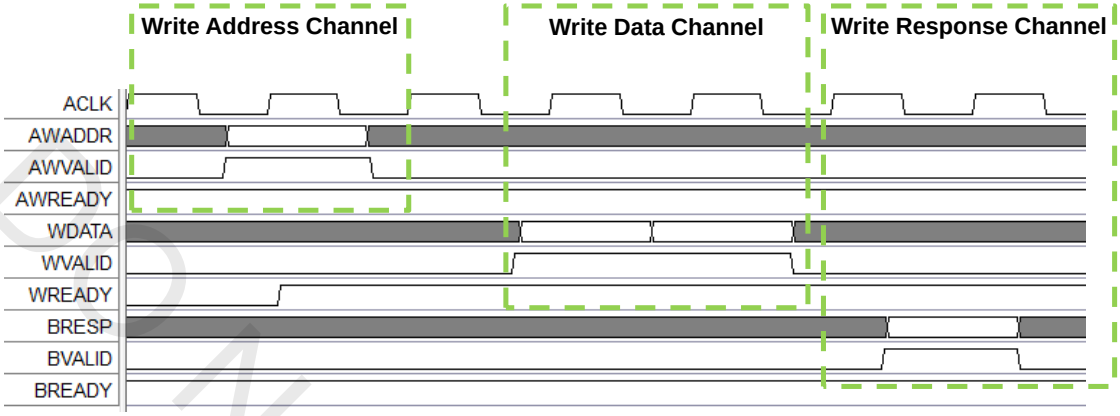


Information Capture

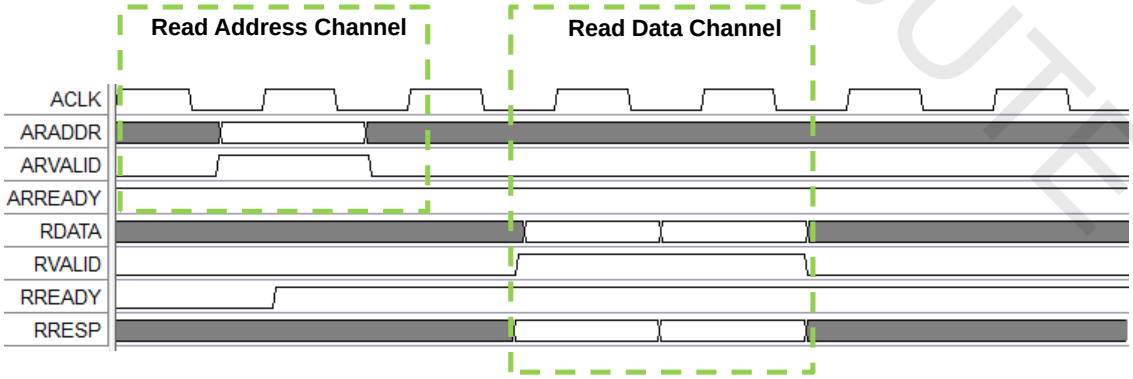
- **READY** active at the same time as **VALID**



AXI Write Transaction

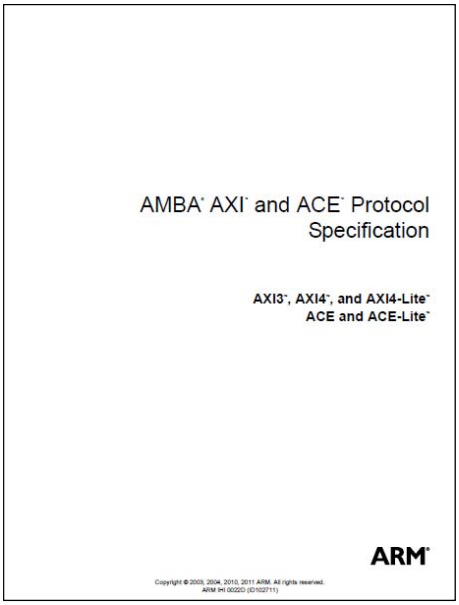


AXI Read Transaction



AXI Interface Specification

- Defines the entire AXI Interface standard
- AXI3 and AXI4 interfaces
- AXI specification available from ARM


AMBA AXI and ACE Protocol Specification

AXI3, AXI4, and AXI4-Lite
ACE and ACE-Lite

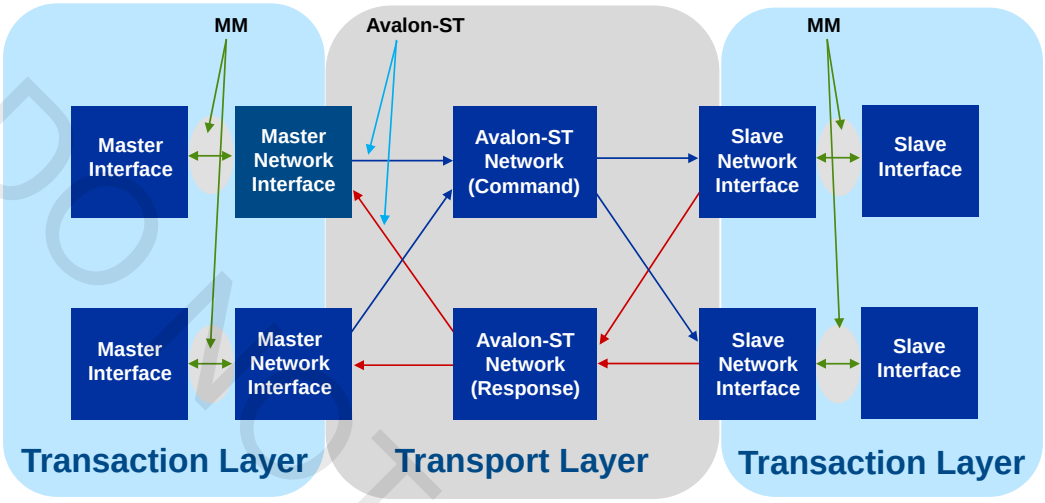
ARM

Qsys Packets

- All memory mapped transactions automatically converted to Qsys packets
- The Qsys tool uses a wide packet format
  - Contains complete transaction in a single clock cycle
    - Writes in 0 cycle
    - Reads with a round-trip latency of 1 cycle
  - Project Settings assignment allows tradeoff between latency and maximum frequency
- Separate command and response network
  - Increases concurrency
    - Command traffic and Response traffic don't have to compete for throughput
    - Networks tailored individually to system topology

NoC Architecture

- Memory-mapped packet transactions and transport
  - Each transfer/request encapsulated in packet and sent to slave
  - Each response encapsulated in packet and sent back to master



Qsys Memory-Mapped Packet Format (1/2)

Packet Field	Description
Address	Byte address of lowest byte in packet
Size	Describes the segment of the payload that contains valid data for a beat
Address Sideband	Up to 8 bit signals for rd/wr address channels; valid for each beat in a packet
Cache	AXI cache signals
Transaction (Exc)	Indicates exclusive access (read, compressed read, write, posted, lock)
Transaction (Posted)	Indicates non-posted writes (require response)
Data	Write - data to be written; Read - data that has been read
Byte Enables	Which bytes of data in packet are valid
Source ID	Command - ID of the master; Response - ID of the slave
Dest ID	Command - ID of the slave; Response - ID of the master
Response	AXI response signals
Thread ID	AXI transaction ID values

Note: Fields in yellow are for AXI interface support and are ignored or removed for Avalon interfaces

Qsys Memory-Mapped Packet Format (2/2)

Packet Field	Description
Byte Count	Number of remaining bytes in the transfer
Burst Wrap	Defines the wrapping behavior during bursting
Protection	Access level protection 0 - normal access; 1 – privileged access
QoS	AXI4 std: 4 bit field carries QoS info from AXI master to slave AXI3 std: 4'b0000 indicates not participating in QoS scheme QoS bits are dropped by slaves that do not support QoS
Data sideband	On Write, signals map to WUSER register On Read, signals map to RUSER register On Write response, signals map to BUSER register

Note: See Qsys Interconnect chapter of the Quartus II Handbook for more details on the packet fields.

Which Protocol to Choose: Avalon or AXI?

- No right answer….

Reasoning	Which Bus?
Desire for simpler interfaces	Avalon
Working with existing Avalon interface-based systems	Avalon
Have legacy AXI IP	AXI
Require secure transactions	AXI
Need the ability to lock or have exclusive access to slaves (i.e. mutex):	AXI*

- Ability to mix and match protocols among interfaces

*Avalon interface supports locked transactions, but does not support exclusive accesses

Test Your Knowledge

- True or False, The HPS can be instantiated directly in HDL without the Qsys tool
  - False
- Which if these is not a hardware to software handoff tool?
  a) Device Tree Generator
  b) Second Stage Bootloader Generator
  c) Linux Application Generator
  d) System Header File generator
- True or False, Qsys tool will automatically handle HPS AXI master to Avalon slave interface translations
  - True

Exercise 2

Complete the HPS Qsys System

Hardware Design Agenda

- Hardware design flow with Qsys
- Configuring the HPS IP
- Software handoff
- Avalon/AXI overview
- HPS simulation
- HPS configuration and booting
- SoC debug

HPS System Simulation Support

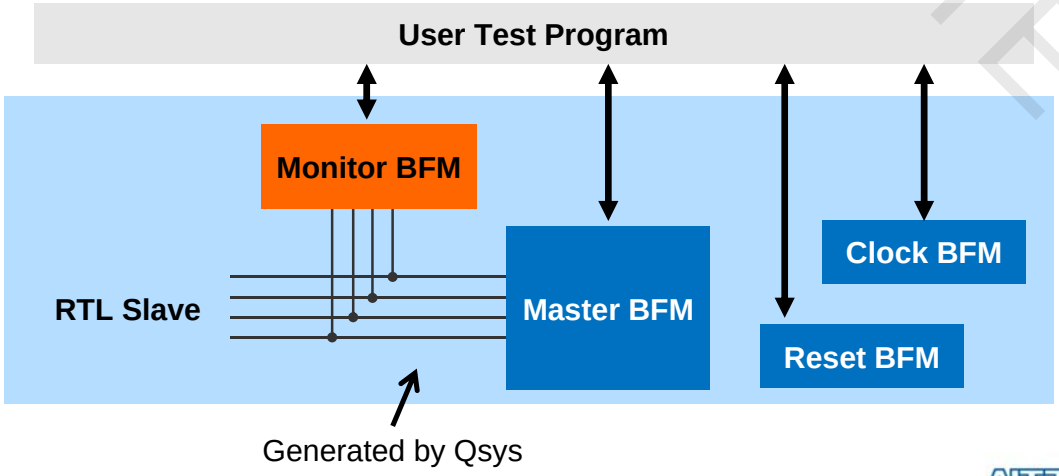
- AXI Bus Functional Models (BFMs) provided
  - Licensed through Mentor Graphics® Corporation
  - Implemented in SystemVerilog
    - VHDL wrapper components provided
  - Master BFM, Slave BFM, and Inline Monitor are available
  - Use to test AXI components
- Avalon BFMs
  - Used to test any Avalon interface
    - Master, Slave, or Streaming
  - Written in SystemVerilog with available VHDL wrapper
- Allows rapid verification of your FPGA IP
  - Instantiate appropriate BFM for each interface
    - Validate IP before integrating with system
  - Access the BFMs through standard function calls in the BFM API
- HPS component not simulatable

Simulation Flow

1. Create HPS system in Qsys
   - Can also simulate individual Avalon/AXI components standalone
2. Generate simulation model or testbench system
3. Write top-level test program
4. Build and run simulation script

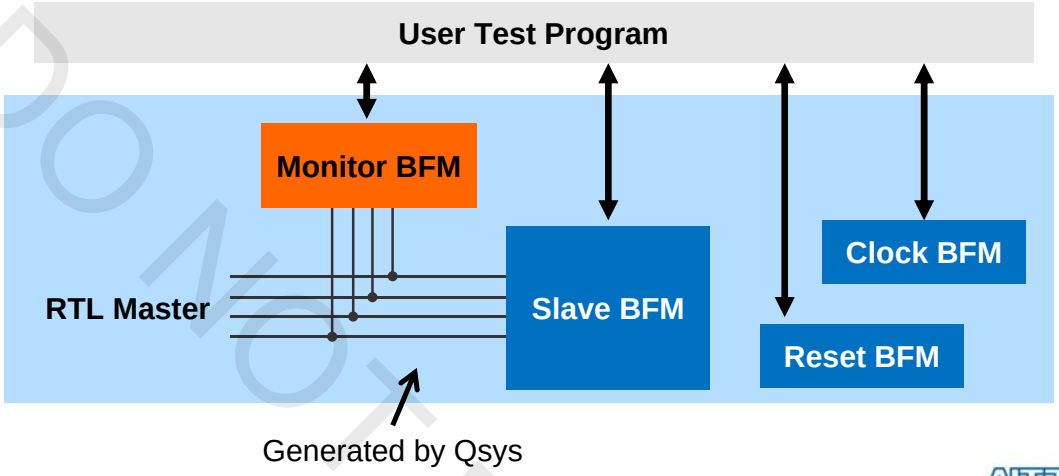
Slave Component Testing

- Build Qsys system with RTL slave (DUT) components interfaces exported or connected to BFMs
- AXI/Avalon Master BFM generates transactions
- Monitor BFM watches traffic and does checking



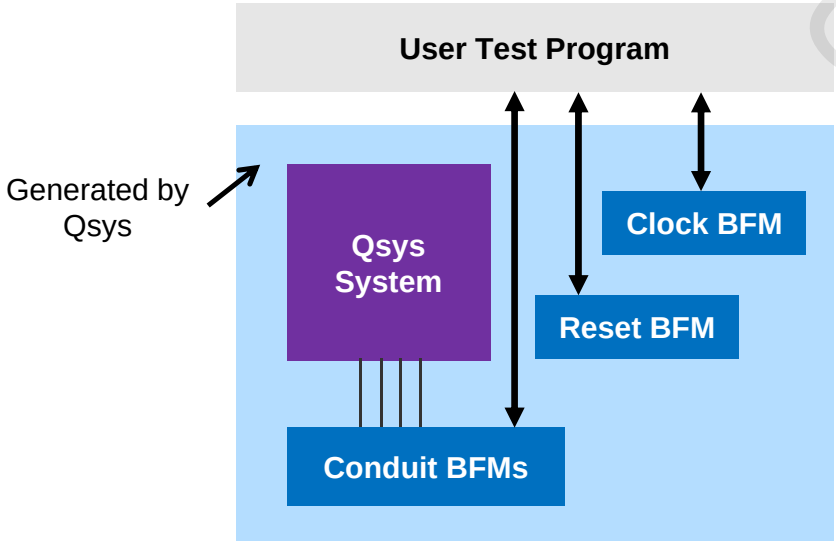
Master Component Testing

- Qsys system with RTL master (DUT) components interfaces exported or connected to BFMs
- AXI/Avalon Slave BFM generates responses
- Monitor BFM watches traffic and does checking



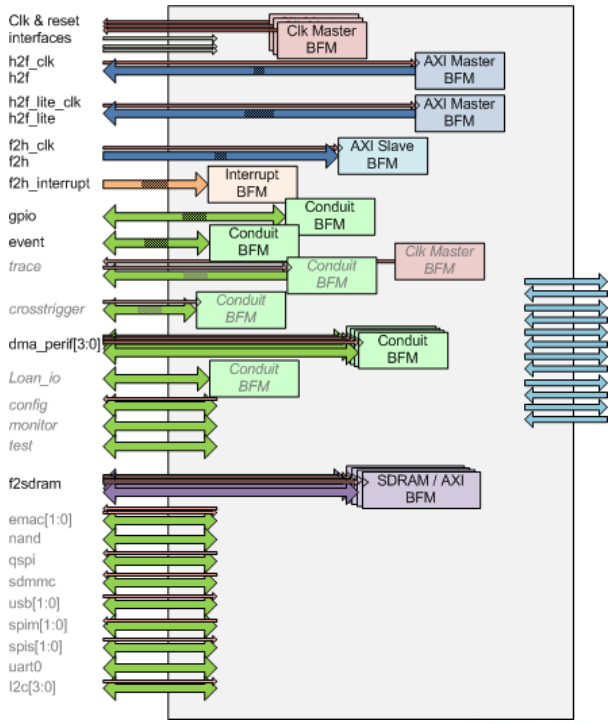
HPS System Testing

- Completed HPS system in Qsys
  - HPS component represented by appropriate BFMs



HPS Simulation Support - Interfaces

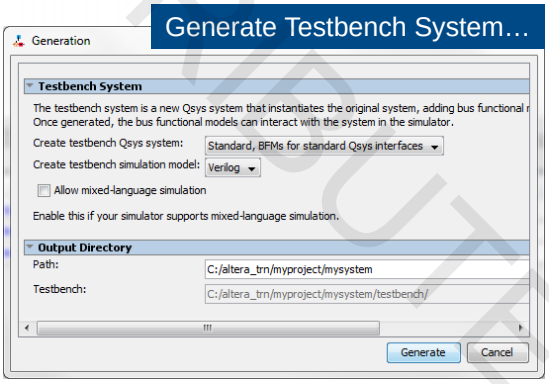
- Each HPS interface will be represented by an interface BFM
- Clock output interfaces will be driven by Clock Master BFMs
- Pin-side interfaces will be unconnected within the HPS simulation model
- Applies to both simulation model and testbench generation



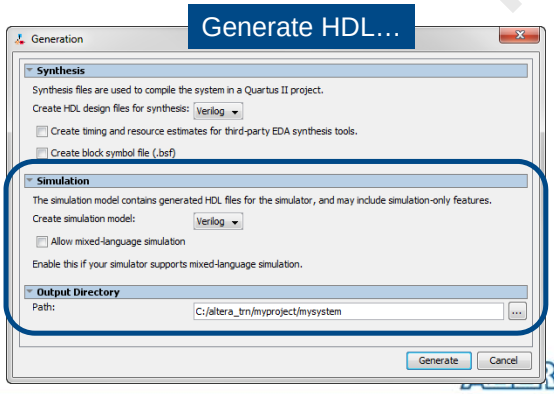
Generate Qsys System for Simulation

- Testbench Generation
  - Standard
    - For component testing
    - BFMs created for every exported interface
  - Simple
    - For system testing
    - Only clock and reset BFMs created
- Simulation Model Generation
  - Generates simulation model of system, no BFMs added for exported interfaces

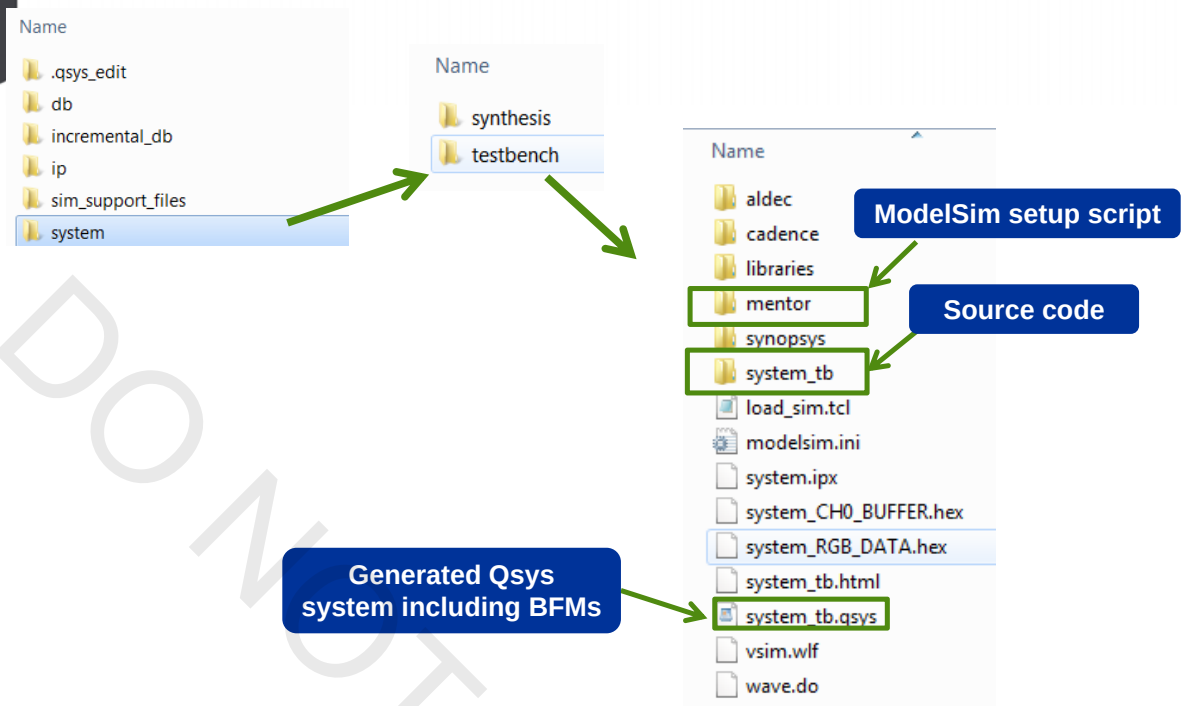
Generate Testbench System…



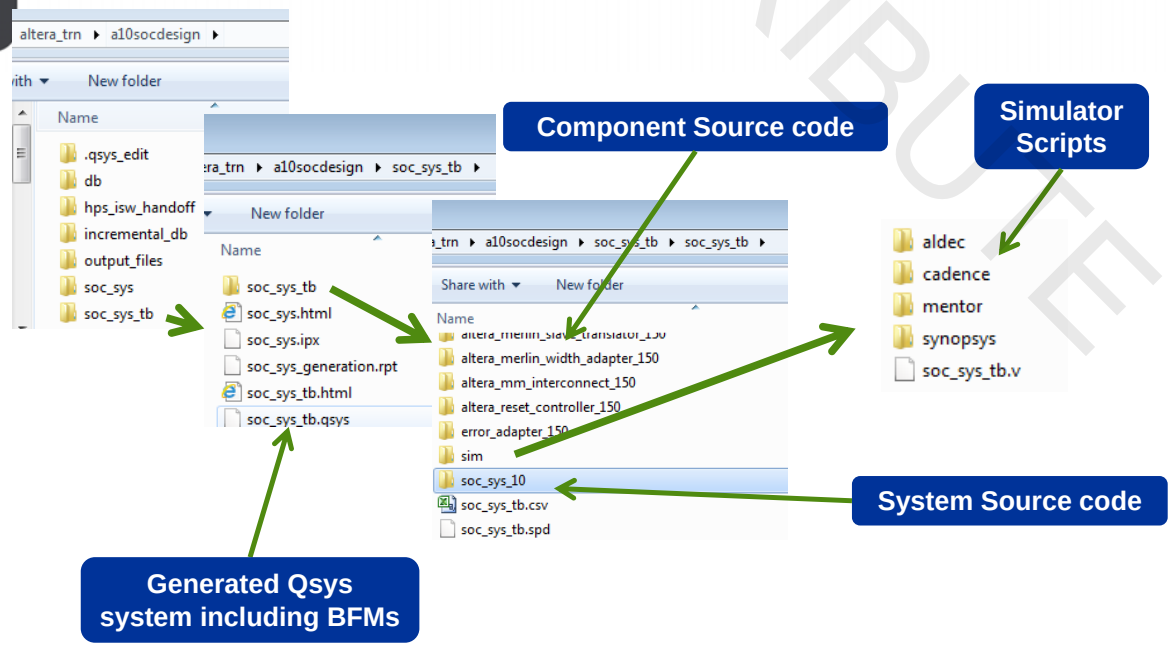
Generate HDL…



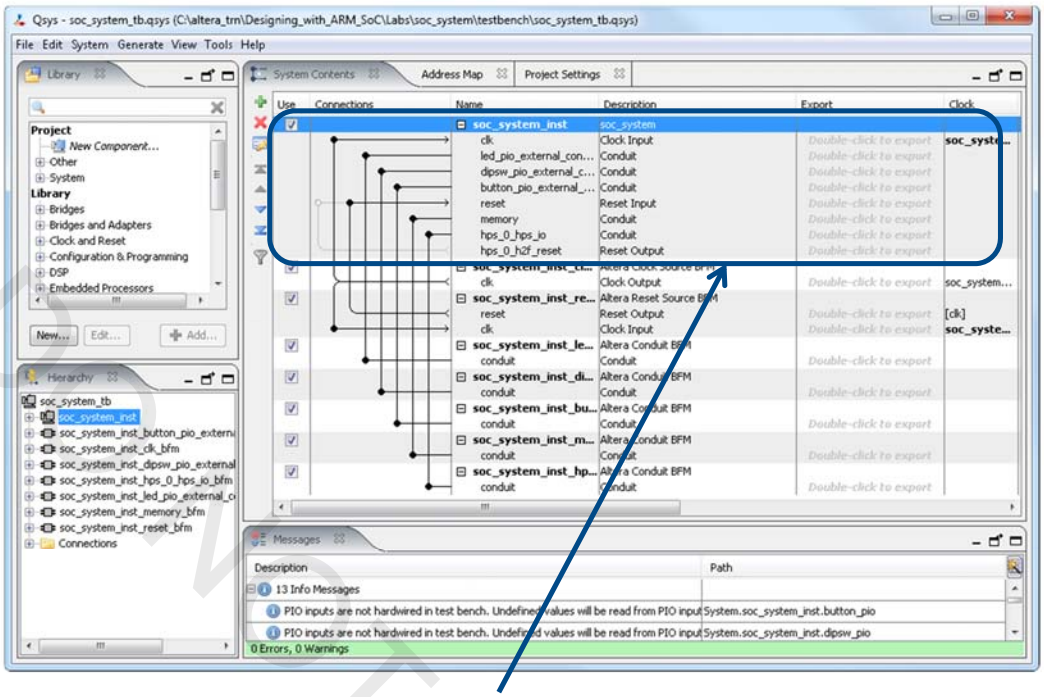
Testbench Directory Structure for 28nm Devices



Testbench Directory Structure for Arria 10 Devices



Qsys Testbench System – HPS System



HPS System Under Test

Writing the Test – BFM API Overview

- The BFM API is a System Verilog interface with VHDL wrapper functions
  - Application Programming Interface (API) contains tasks and functions
  - Organizes transactions into commands and responses
    - Abstracts away signal level details
- The tasks/functions are used to communicate between the API and the underlying AXI implementation
  - The tasks and functions often contain a handle to a transaction

Testbench Example

```
module test_program;

   import mgc_axi_pkg::*;

   logic [31:0]       temp_register;
   axi_transaction    master_trans, slave_trans;

   example_tb system();

   initial begin
      //initial reset - h2f_reset follow the tb reset
      `h2f_reset.reset_assert();
      wait (`tb_reset.reset == 1);
      `h2f_reset.reset_deassert();

      fork: axi_transaction_thread
         begin: master_thread
            // s2f axi master to configure read command
            master_trans       = `s2f.create_read_transaction(32'h8);
            master_trans.id    = 8'h1;
            master_trans.size  = AXI_BYTES_8;
            master_trans.burst = AXI_INCR;
            master_trans.lock  = AXI_NORMAL;
            master_trans.cache = AXI_NONCACHE_NONBUF;
            master_trans.prot  = AXI_PRIV_SEC_DATA;

            // send read data command and wair for read date and read response
            // read data and read response will be contained inside master transaction
            `s2f.execute_transaction(master_trans);
         end  //master_thread

         begin: slave_thread
            // f2s axi slave wait and receive the read command
            slave_trans = `f2s.create_slave_transaction();
            slave_trans.set_address_ready_delay(4);
            `f2s.get_read_addr_phase(slave_trans);

            // configure read data and read response and send it output
            slave_trans.resp[0]        = AXI_OKAY;
            slave_trans.data_words[0] = 64'h32;
            `f2s.execute_read_data_burst(slave_trans);
         end //slave_thread
      join //joins axi_transaction_thread
```

Import AXI BFM package

Instantiate Qsys testbench system

Setup values for master transaction

Send transaction

Wait for transaction

Setup response

Send response

Using Conduit BFMs

- Every signal on a conduit BFM gets its own set & get function

Set value on "gp_in" signal of BFM

Retrieve value from "gp_in"

```
      //coduit transaction
      //drive signal to mpu_gp's gp_in and retrieve the value
      `conduit.set_gp_in(32'h5);
      #1;
      temp_register = `mpu_gp.get_gp_in();

      //drive signal to mpu_gp's gp_out
      `mpu_gp.set_gp_out(32'h10);

      wait(2) @posedge `tb_clk.clk);
      $finish;
   end
endmodule
```

Set value on "gp_out"

Run Simulation Script

- Scripts for simulators created automatically by Qsys
  - Supports tools from Mentor Graphics, Synopsys®, Cadence®, Aldec®
  - For simulation model
    - ***<project_directory>/<Qsys design name>/simulation/<vendor>***
  - For simulation testbench
    - ***<project_directory>/<Qsys design name>/testbench/<vendor>***
- Sets up simulation variables
- Creates and compiles device libraries and source files
- Simulates the design

Learn More

- Mentor Verification IP Altera Edition User Guide
  - http://www.altera.com/literature/ug/mentor_vip_ae_usr.pdf
- Quartus II handbook Qsys chapter
  - http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf
- Instruction-led trainings
  - Introduction to the Qsys System Integration Tool
  - Advanced Qsys System Integration Methodologies
- Free Altera Online trainings
  - Introduction to Qsys
  - Advanced System Design Using Qsys
  - Custom IP Development Using Avalon and AXI Interfaces

Hardware Design Agenda

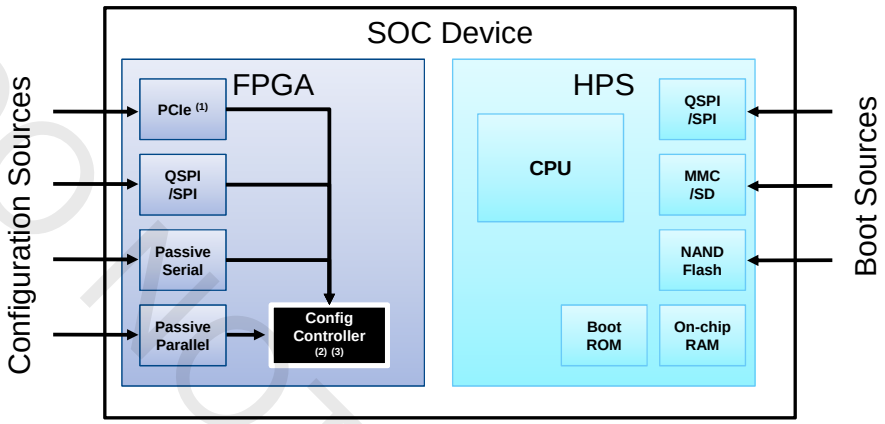
- Hardware design flow with Qsys
- Configuring the HPS IP
- Software handoff
- Avalon/AXI overview
- HPS simulation
- HPS configuration and booting
- SoC debug

SoC Configurations & Boot Sequences

- Independent FPGA configuration and HPS boot
- FPGA configured then HPS boots through FPGA
- HPS boots and configures the FPGA

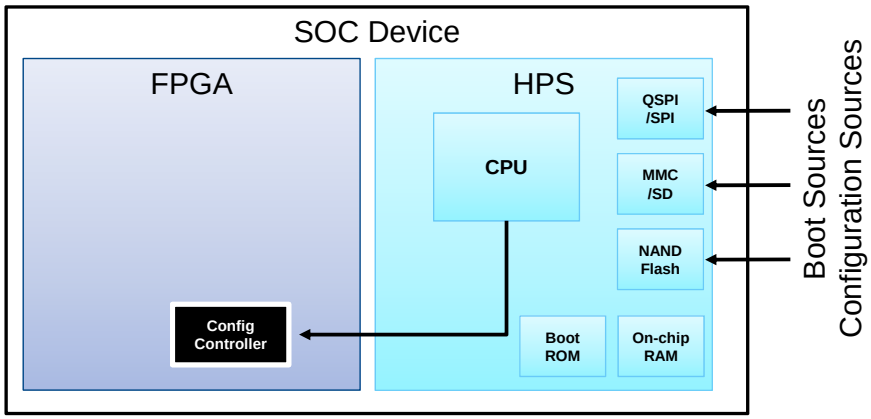
System Boot Schemes – Independent

- FPGA configured from standard non-HPS sources
  - Configuration device, flash memory, etc
- HPS obtains 2nd stage bootloader from boot source
  - Flash memory



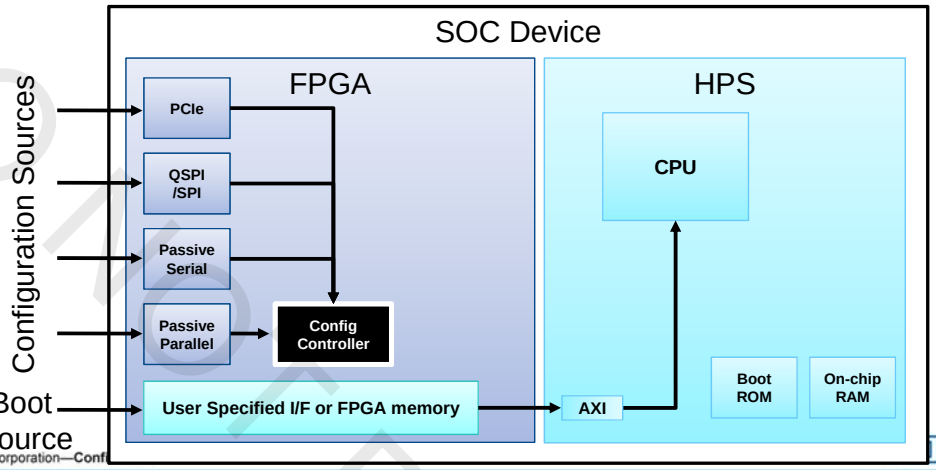
System Boot Schemes – HPS First

- HPS boots first through a non-FPGA source
- Software running on the HPS configures the FPGA through the FPGA Manager
  - FPGA image from flash memory or any communication interface

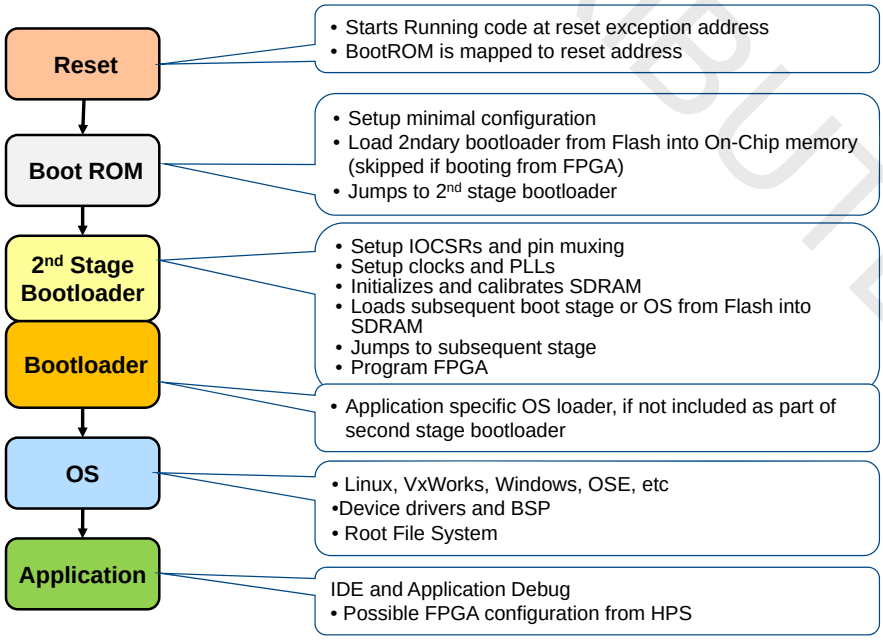


System Boot Schemes – FPGA First

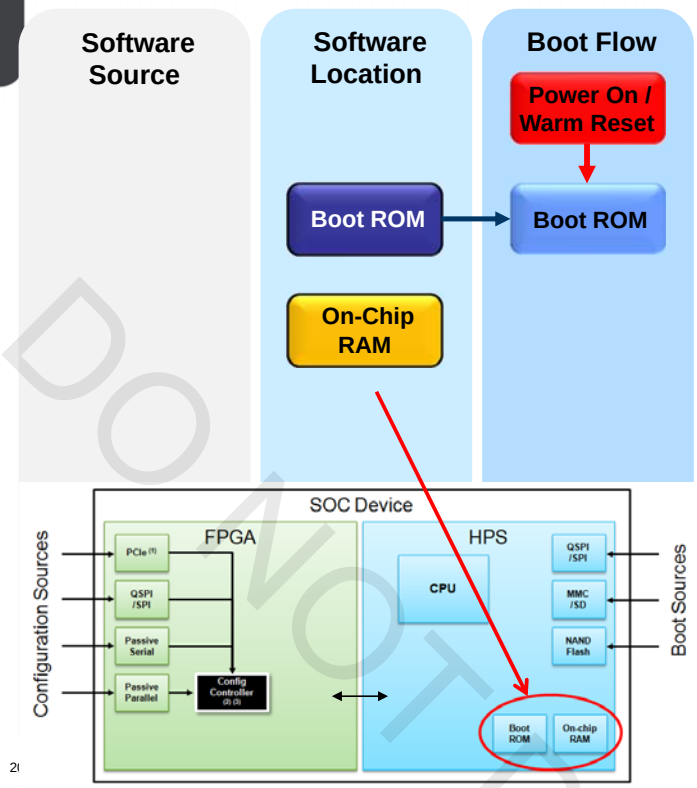
- FPGA configured from standard non-HPS sources
- HPS boots from FPGA fabric
  - Waits for various control signals (init_done and boot_from_fpga signal)
  - Boot ROM launches software running across the HPS-to-FPGA bridge
    - 2nd stage bootloader or other software can be in FPGA RAM or other sources



HPS Typical Boot Stages

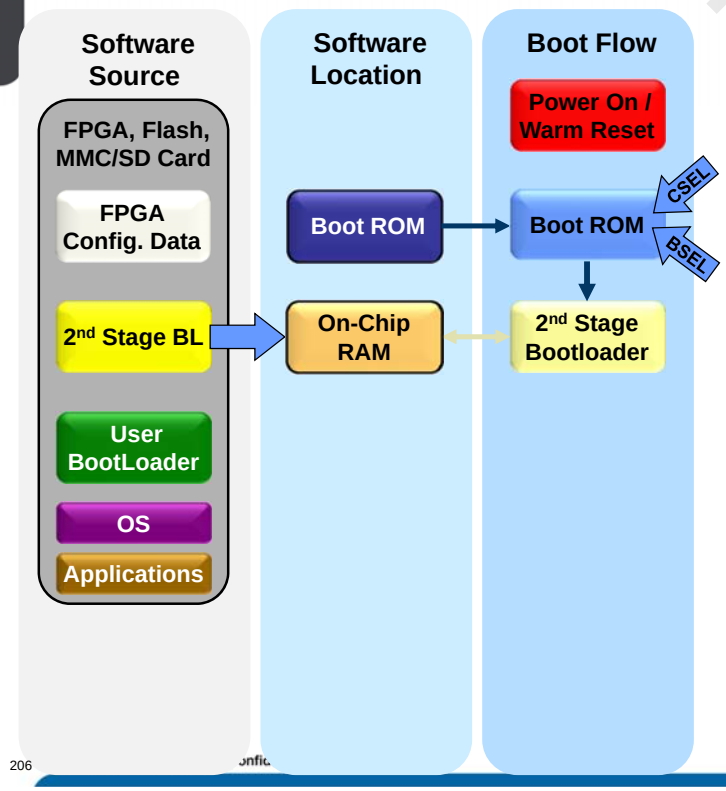


HPS Power On / Reset



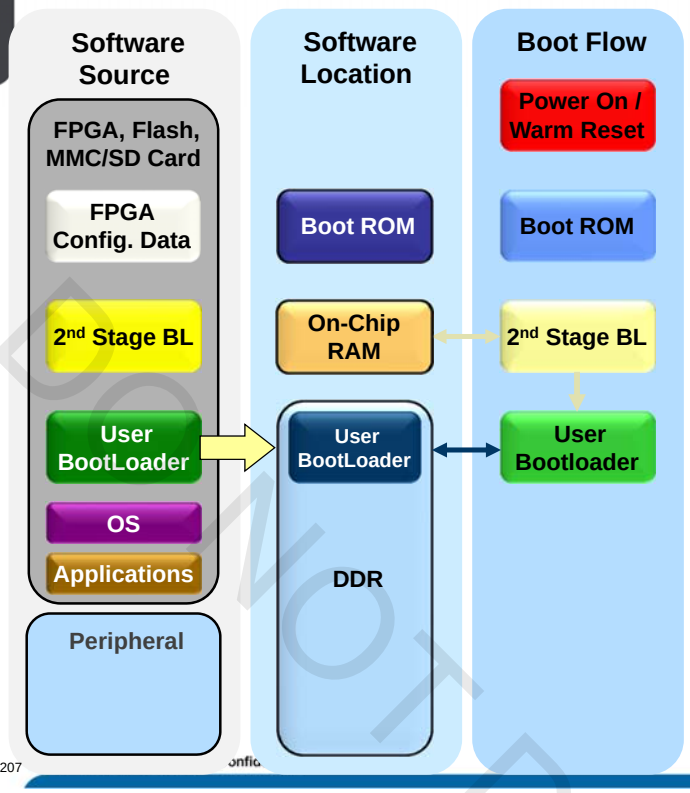
- After reset/power up system runs Boot ROM code on CPU0
- CPU1 is held in reset
- Boot ROM code hard programmed as primary bootloader
- Boot ROM code uses On-chip RAM space for data storage

HPS Boot ROM



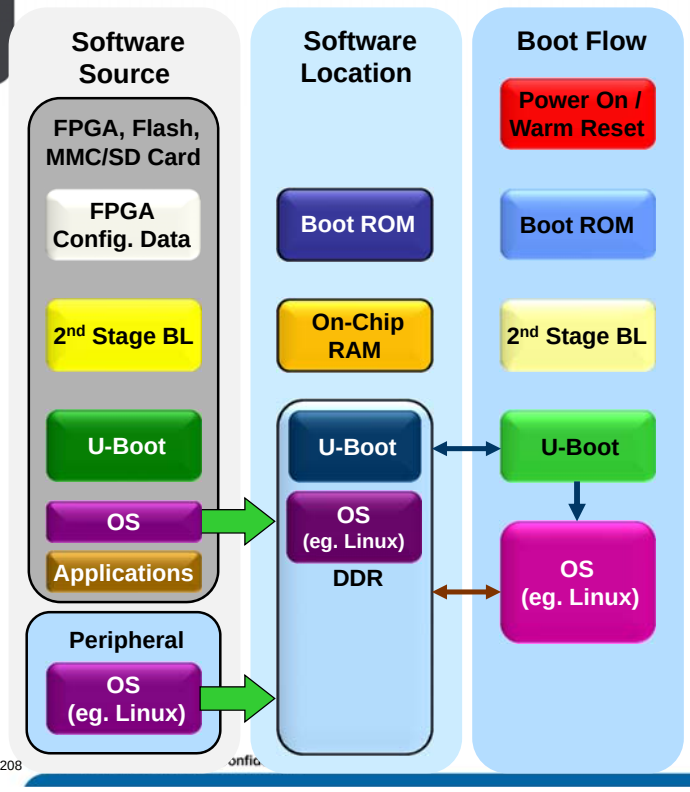
- Boot ROM code scans boot-select and clock-select pins to determine flash clock setup and boot source
- Configures minimal set of HPS I/O pins to read boot source using I/O config. data stored in Boot ROM
- Performs CRC check & loads 2nd stage bootloader (Preloader) software from boot source into On-Chip RAM
- Boot ROM hands off program control to the 2nd stage bootloader

Second Stage Bootloader



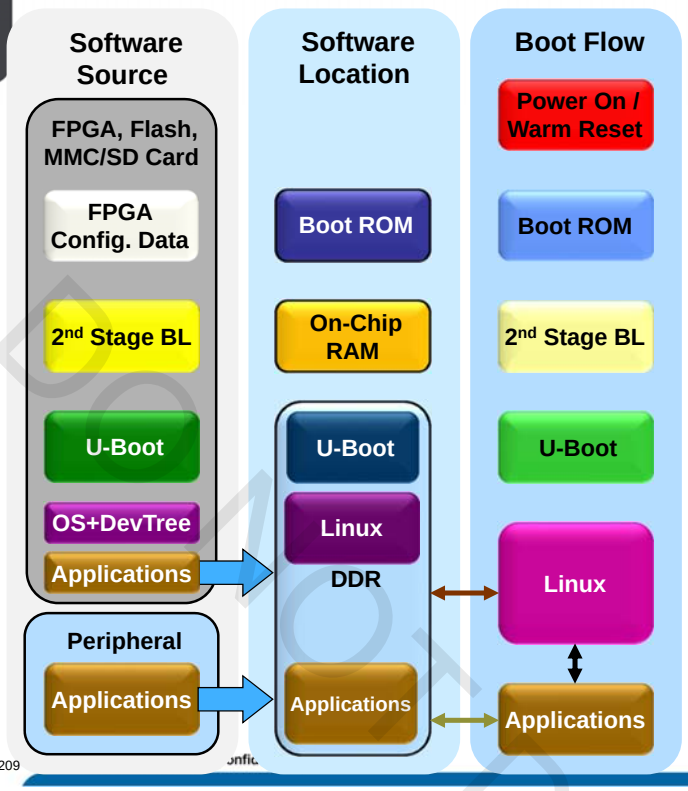
- Built from Qsys handoff files
- Limited by size of On-Chip memory
- HPS I/O and SDRAM configuration data compiled into 2nd Stage Bootloader
  - As software (Cyclone V and Arria V SoCs) or FPGA bitstream (Arria 10 SoCs)
- Sets HPS pin configuration
- Initializes, calibrates and verifies SDRAM setup
- Copies next stage software (e.g. U-Boot or OS) into SDRAM from boot source
- Hands off control to the next stage

HPS User Bootloader



- Application/OS specific
- Bootloader Copies Operating System from non-volatile RAM (or Peripheral) to SDRAM
  - Linux, VxWorks, etc
- Runs any other processes specified
- Hands off control to OS
- For Arria 10 devices, the functionality of the User Bootloader is built into the 2nd Stage Bootloader

HPS Linux OS Start Up

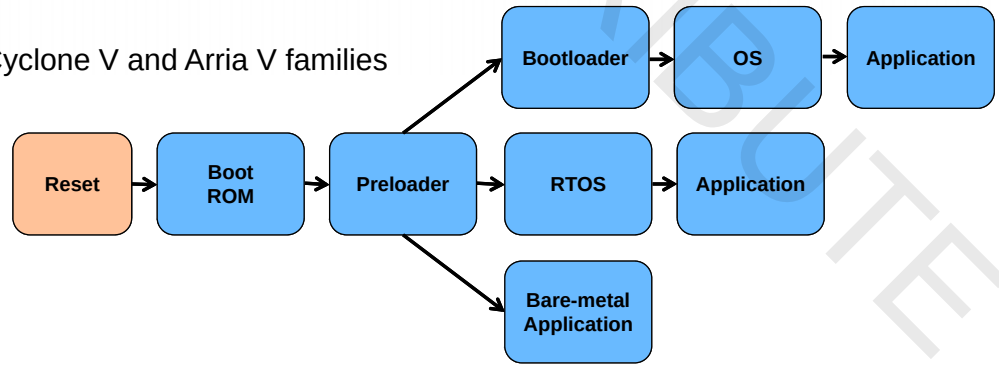


- OS launches
- Runs BSP/device driver initialization routines
  - Processor setup (memory, interrupts etc.)
  - CPU 1 initializes and runs
  - Peripheral hardware setup (detect, verify and initialize)
- Creates OS specific resources
  - e.g. Root File System
- Same OS runs on both processors
- Loads and runs Applications

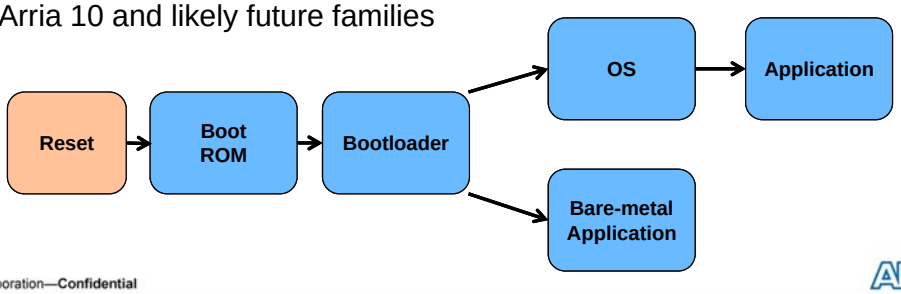
209

Typical Boot Scenarios

Cyclone V and Arria V families



Arria 10 and likely future families



210

"Bare Metal" Programming

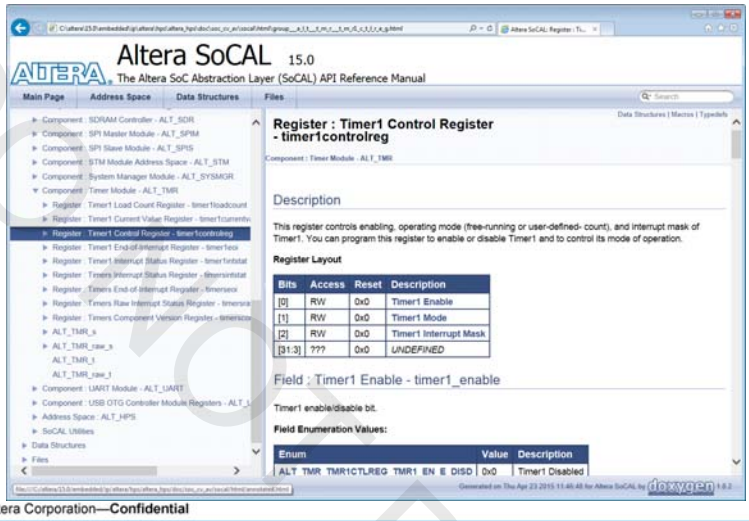
- Bare metal: the actual register interfaces and hardware features of the processor system
- Bare metal programming: code that reads and writes direct to the hardware with no intervening software functions/code/abstraction layer
- The less operating system resources are used, the closer you are to bare-metal
- Requires in-depth knowledge of hardware to write

HWLibs Components

- SoC Abstraction Layer (SoCAL) - (low level HAL)
  - Macro based abstraction layer to access low-level hardware IP registers
  - Header files
  - C code generated from chip RTL
  - Decouples software from hardware
  - Each type of HPS has its own SoCAL
- Hardware Manager (HWMgr)
  - Collection of C and some assembly language APIs for more complex higher level access to SoC hardware
  - #includes SoCAL header files
  - Written by Altera engineers

HWLibs Documentation

- SoC Abstraction Layer:
  - <SoC EDS folder>/ip/altera/hps/altera_hps/doc/soc_<fam>/socal/html/index.htm
- Hardware Manager
  - <SoC EDS folder>/ip/altera/hps/altera_hps/doc/hwmgr/html/index.htm



Creating System Header File

- Create system header file from Qsys .sopcinfo file
  - Abstract away FPGA component properties from software
  - Run `sopc-create-header-files` from embedded command shell
    - Executable part of the Quartus II software install

```
sopc-create-header-files <qsys system>.sopcinfo
```

  - Outputs various .h files
    - For the system
    - For the processors
    - For each of the masters in the system

Hardware Design Agenda

- Hardware design flow with Qsys
- Configuring the HPS IP
- Software handoff
- Avalon/AXI overview
- HPS simulation
- HPS configuration and booting
- SoC debug

SoC Debug Agenda

- Debug Overview
- System Console
- FPGA Adaptive Software Debug

Traditional Software and FPGA Debug

- All traditional FPGA and ARM software debug methods still apply
- ARM software debug
  - JTAG tools, compiler, debugger, profiler, trace, etc
- FPGA
  - SignalProbe, SignalTap II Logic Analyzer, System Console, USB-Blaster Cable, Programmer, etc

HPS Software Debug Interfaces Supported

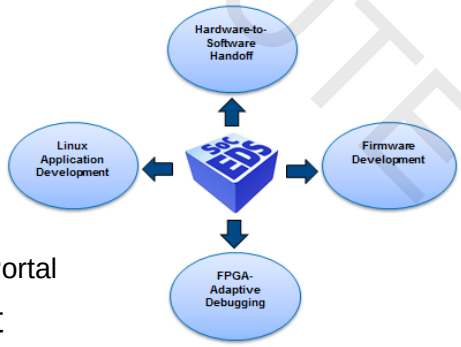
- ARM DS-5 Altera Edition toolchain
- JTAG
  - Direct connection to hardware
  - Halts processor, can master system via control of processor
  - Debug any code at low level – driver, OS kernel, bare metal application
  - Dual core SMP processor debugging
  - OS aware debugger display optional
- Ethernet
  - Requires running software to drive it
  - Processor/OS <u>cannot</u> be halted
  - OS Application <u>can</u> be halted
  - Good for high level programming (fast, easy)
  - Debuggers usually have OS aware GUIs and features
  - Cannot do low level debug (kernel, driver, bare metal) as OS cannot be stopped

Hardware Debug with SignalTap II Logic Analyzer

- Part of the Quartus II software
- Built in logic analyzer capability for internal FPGA logic
- Captures the state of signals when running at speed
- Uses standard JTAG connection to PC
- User defines target signals and trigger conditions to capture data
- Data read back from FPGA memory and displayed on PC
- Works on the FPGA side of the SoC

Altera SoC Embedded Design Suite

- Comprehensive software / firmware development environment
- FPGA-adaptive software debugging capabilities
  - ARM DS-5 Altera Edition Toolkit
- Hardware / software handoff tools
- Linux application development
  - Yocto Linux build environment
  - Pre-built binaries for Linux / U-Boot
  - Work in conjunction with the Community Portal
- Bare-metal application development
  - SoC Hardware Libraries
  - Bare-metal compiler tools
- Design examples



FPGA-Adaptive Debugging

ARM DS-5 Altera Edition Features

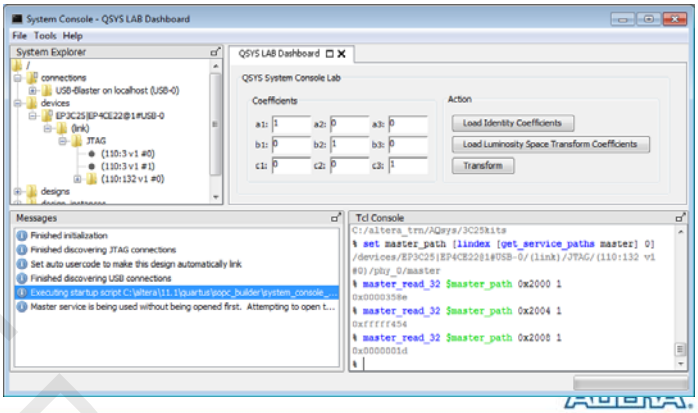
- Removes debugging barrier between CPUs and FPGA
- Single USB-Blaster target connection for SW and HW debug
- Hardware cross-triggering between the CPU and FPGA domains
- FPGA events inserted non-intrusively into HPS program trace
- Debugger with visibility into FPGA registers
  - Automatic creation of register views

SoC Debug Agenda

- Debug Overview
- System Console
- FPGA Adaptive Software Debug

What is System Console?

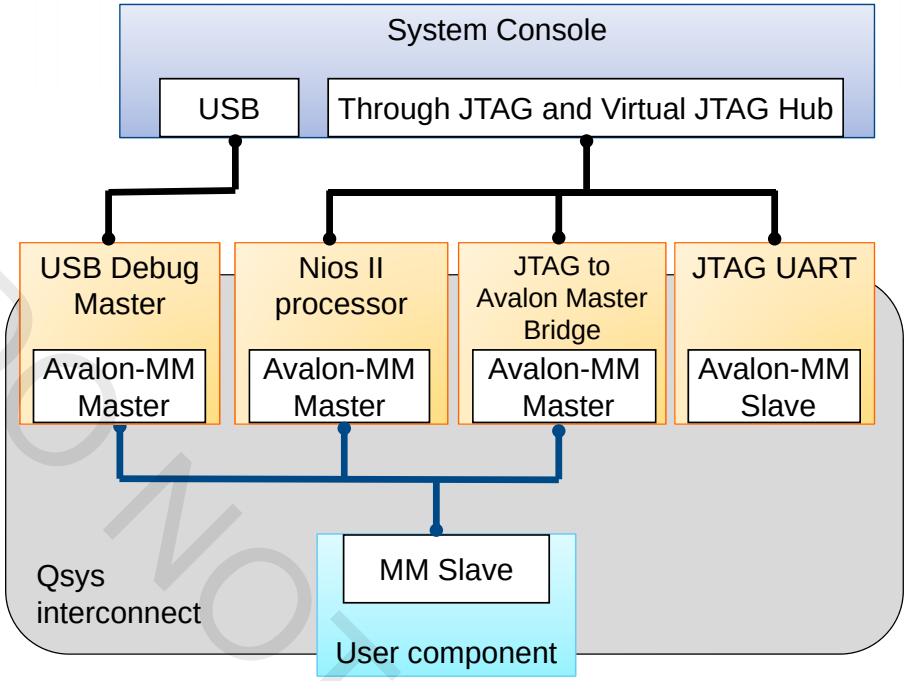
- Quick debug of Qsys systems through an interactive Tcl console
  - Opens as a separate window
  - Or in the Nios® II and Embedded Command Shell
- Debug over various available communication channels
  - JTAG, TCP/IP, or USB
- Read from or write to memory-mapped components
- No processor required



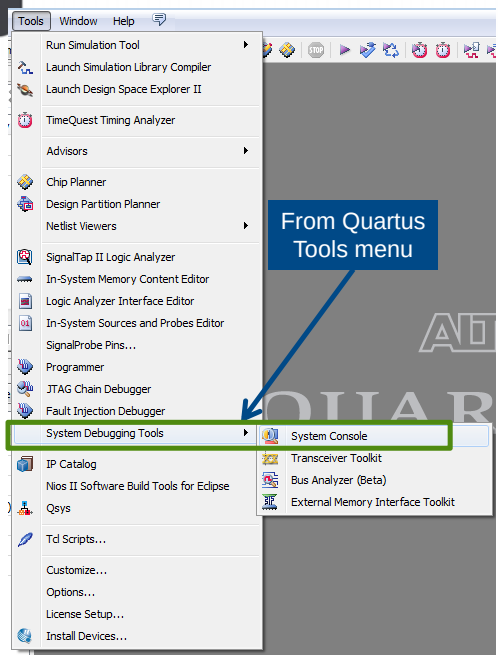
Usage Examples

- System-level debug
  - Board bring-up and interface testing
  - System clock, reset and JTAG chain validity testing
- Bug Isolation
  - HW/SW bug isolation
  - Replicate MM access pattern and ensure proper response
  - Dissecting locked systems while in the locked state
- Process Automation
  - Automate production tests
- Custom Visualization
  - Create interactive Dashboards customized for a system
  - Used to drive manual processes or get immediate feedback

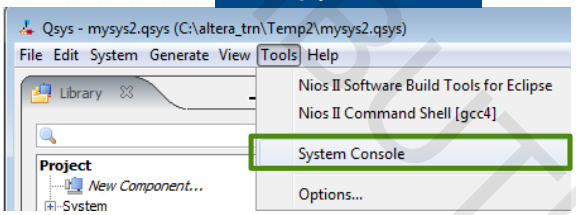
System Console Interfaces



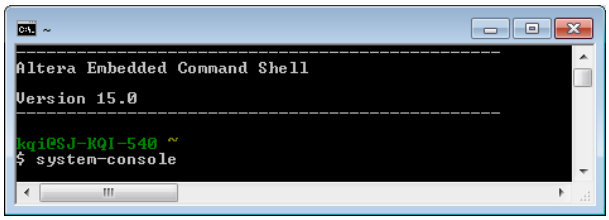
System Console GUI Launch



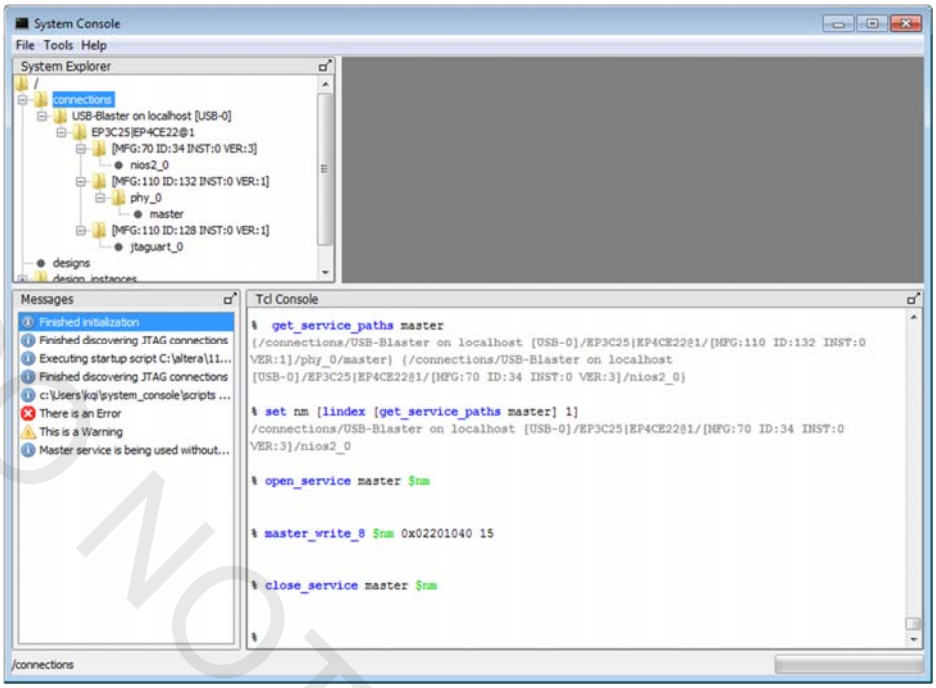
From Qsys Tools Menu



From Nios II or Embedded Command Shell:
`system-console`



System Console GUI



System Console Services

- Collection of functions to accomplish a type of tasks
- Accessed through a service instance which may be associated with a particular component
- Run `get_service_types` to return all available services
- Example Services
  - jtag_debug
    - Perform jtag chain, system reset, and system clock debug
  - master
    - Provide control over a master interface to perform reads and writes on slaves
  - monitor
    - Efficiently and regularly read from a range of addresses
  - dashboard
    - Easily add GUI elements such as buttons and displays to control the system
  - And many others

Usage Flow – Summary

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

Complete master write and read example script

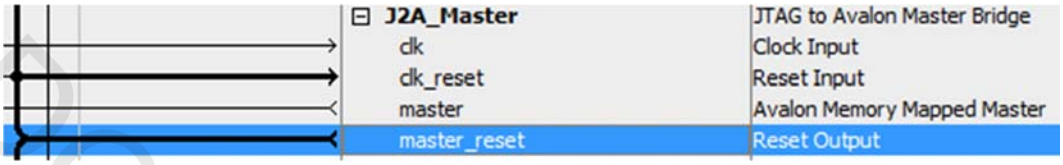
```
set mpath [lindex \
    [get_service_paths master] 0]
set claim_path \
 [claim_service master $mpath “”]


master_write_memory $claim_path \
    0x2000 [list 0x01 0x02]
master_read_memory $claime_path \
   0x2000 2

close_service master $claim_path
```


jtag_debug Service Type

- Verify functionality and signal integrity of the JTAG chain
- Some functions requires JTAG to Avalon Master Bridge
- Verify the clock and reset signals
- Issue reset(s) to the system

Issue JTAG Reset

- `jtag_debug_reset_system <path>`
  - Issues a reset to connected components in the system through the **master_reset** output of the JTAG to Avalon Master Bridge




```
% jtag_debug_reset_system $claim_path
```

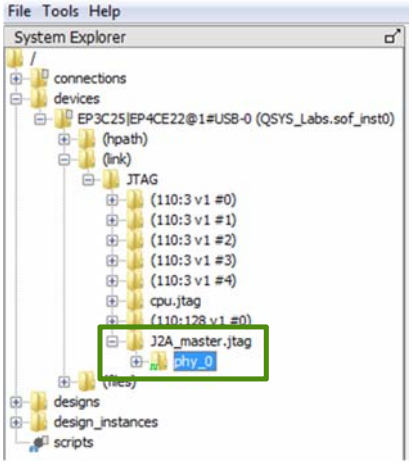

Verify Clock and Reset

- `jtag_debug_sample_clock <path>`
  - Returns the value of the asynchronously sampled system clock
  - May require multiple samplings to see toggling
- `jtag_debug_sample_reset <path>`
  - Returns the value of reset signal
- `jtag_debug_sense_clock <path>`
  - Returns 1 if clock has ever toggled

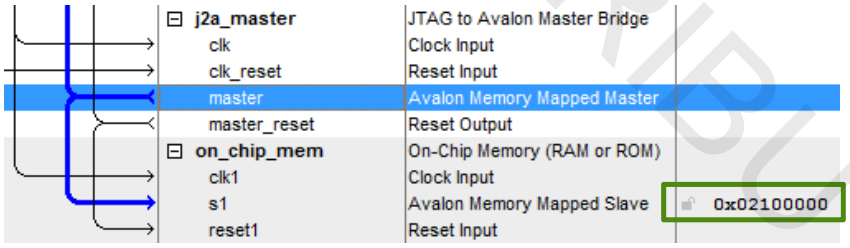

```
% jtag_debug_sample_clock $claim_path
0

% jtag_debug_sample_clock $claim_path
1

% jtag_debug_sense_clock $claim_path
1
```


Clock and Reset Visibility

- Badge in explorer window nodes indicate clock and reset status
- Status updated with Refresh Connections (Tools menu)
- Green for OK
- Red for no clock
- Green with line for reset asserted and held



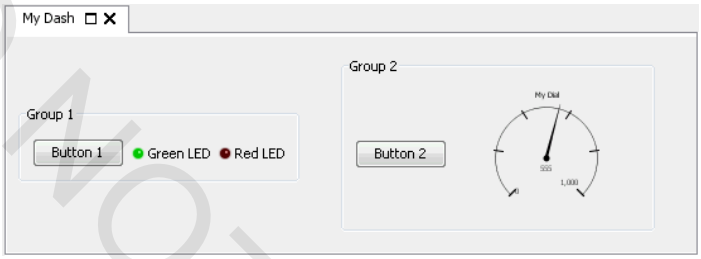
Master Service Complete Example




```
% set m_path [lindex [get_service_paths master] 0]
/connections/USB-Blaster on localhost [USB-0]/EP3C25|EP4CE22@1/[MFG:110 ID:132
INST:0 VER:1]/phy_0/master

% set claim_path [claim_service master $m_path {} {}]


% master_write_memory $claim_path 0x02100000 32


% master_read_memory $claim_path 0x02100000 1
0x20

% close_service master $claim_path
```


Dashboards

- Create widgets in System Console GUI
- Customize the dashboard using Tcl commands
- Interact with live instances of an IP core
- Refer to the "System Console" online training or "Advanced Qsys System Integration Tool Methodologies" Instructor Led Training



SoC Debug Agenda

- Debug Overview
- System Console
- FPGA Adaptive Software Debug

Cross Triggering

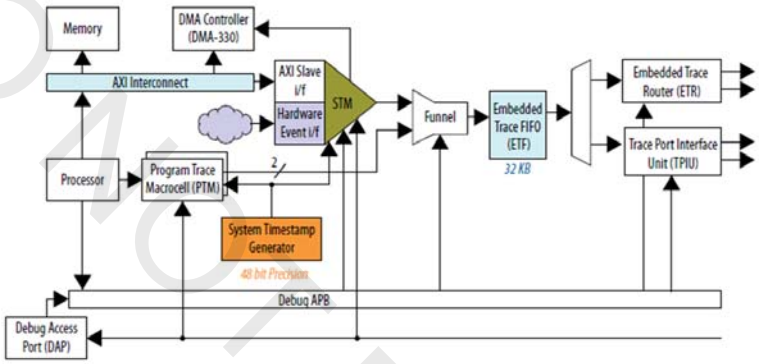
- Synchronization of debug actions on the HPS with hardware events in the FPGA (including SignalTap II instances)
  - Facilitate the debugging of software interacting with hardware
- Features provided
  - FPGA to HPS cross trigger
    - Hardware signal or SignalTap II logic analyzer trigger out breaking HPS execution
  - HPS to FPGA cross trigger
    - Breaking code execution in the HPS cores acts as a input trigger for the SignalTap II logic analyzer or other hardware

Cross Triggering Interface (CTI)

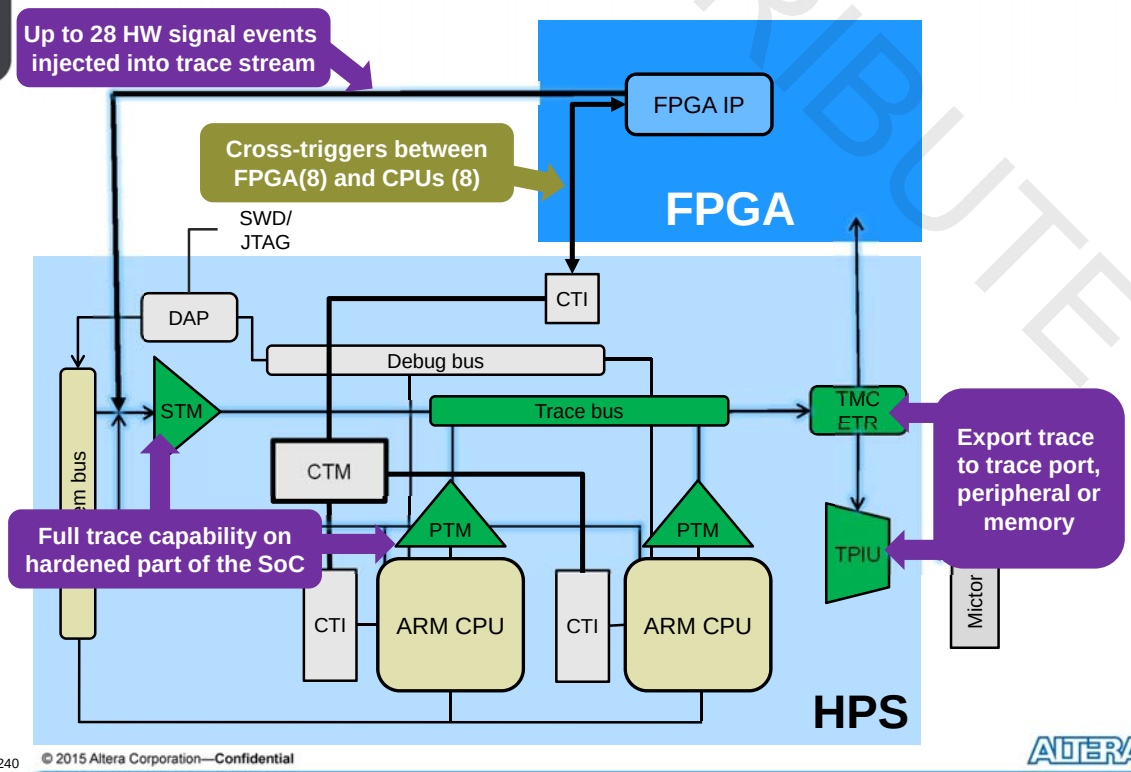
- Standard ARM CoreSight™ debug component in HPS
  - Documented in the CTI CoreSight specifications
- Debug Cross Triggers between FPGA and MPU
  - 8 trigger inputs + 8 ACK signals
  - 8 trigger outputs + 8 ACK signals
- Runs on FPGA clock source
- Trigger handshaking option is supported
  - Trigger output to FPGA will remain asserted until ACK signal received from FPGA
- CTI connected to Cross Trigger Matrix (CTM)
  - Configures trigger connections to debug infrastructure

Synchronizing Hardware Events with HPS Trace

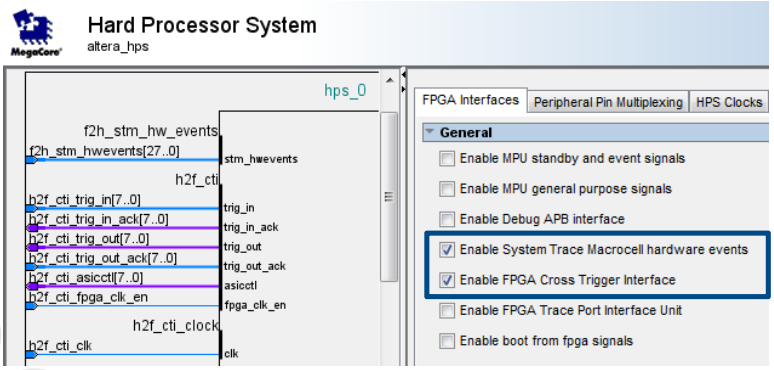
- Correlate hardware event or SignalTap II trigger out with the HPS trace stream without stopping processor
- Hardware signals or SignalTap II trigger out drive the event inputs of the HPS System Trace Macrocell (STM)
- Hardware events generated are stored in trace buffer along with program trace generated by the processors



Altera SoC Debug Architecture

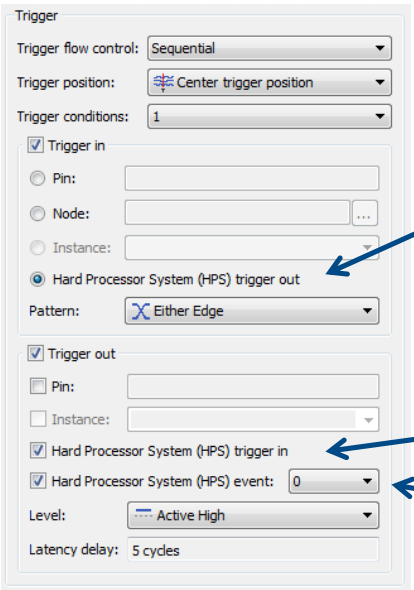


Using Cross Trigger and/or Trace with Custom Hardware



- Enable HPS events and cross trigger interfaces
  - Enabling STM hardware events allow FPGA modules to synchronize FPGA events with the program trace
  - Enabling the Cross Trigger Interface allows FPGA modules to halt the processor or take the HPS trigger out
  - Connect to FPGA hardware
- Must be disabled if to be used with SignalTap II Logic Analyzer
  - Configure cross triggering and events from within SignalTap II setup

SignalTap II Configuration for Cross Trigger



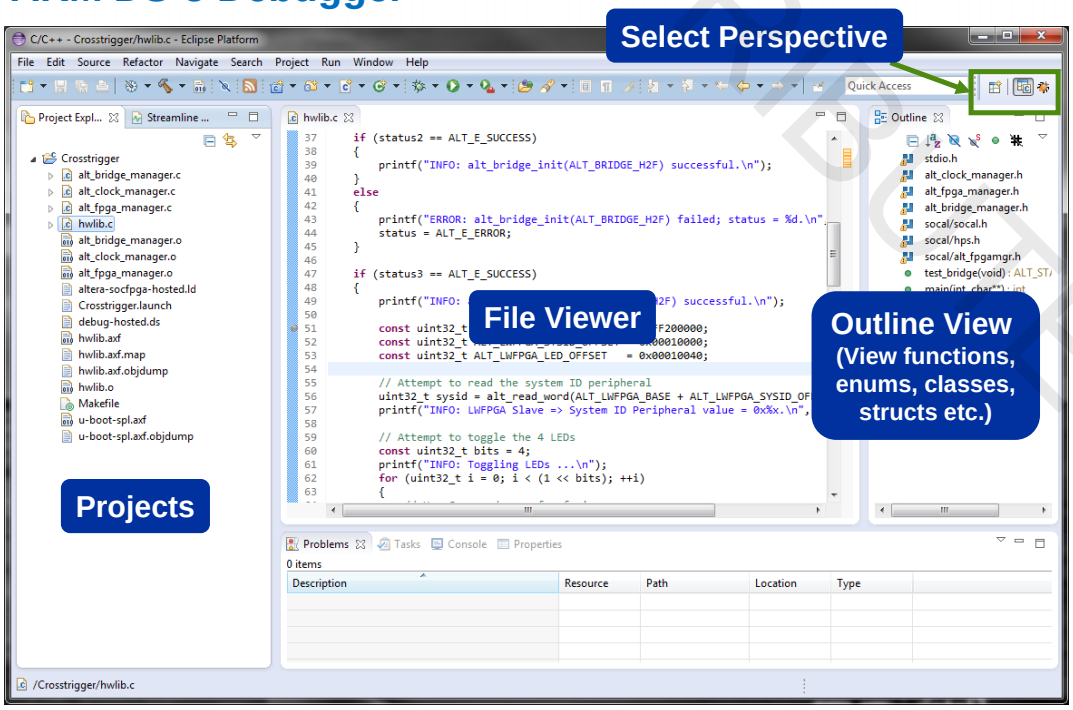
- HPS -> FPGA Cross Trigger
  - Allow the breaking in the running of the HPS core to act as the trigger in condition for SignalTap II Logic Analyzer
- FPGA -> HPS Cross Trigger
  - Allows SignalTap II trigger out to break the execution of the HPS core
  - Allow trigger out to generate STM hardware event in the HPS trace

Qsys HPS Component Export of Cross Trigger Interface must be disabled

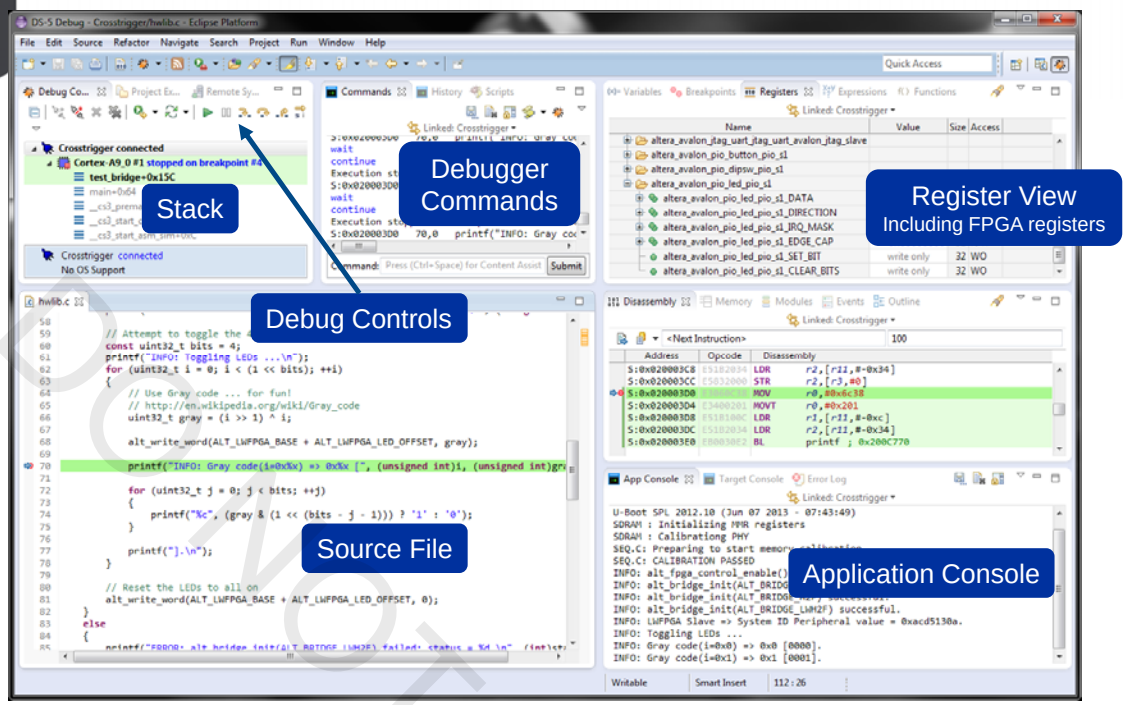
Before Running Debugger with Cross Triggering Enabled

- Make sure FPGA is programmed
- Ensure CTI interface enabled
  - Interface enabled in Qsys HPS parameterization or in SignalTap II setup
  - Interface connected to appropriate logic
- Otherwise HPS will be stuck in an unknown state trying to read the CTI register requiring a reboot

ARM DS-5 Debugger

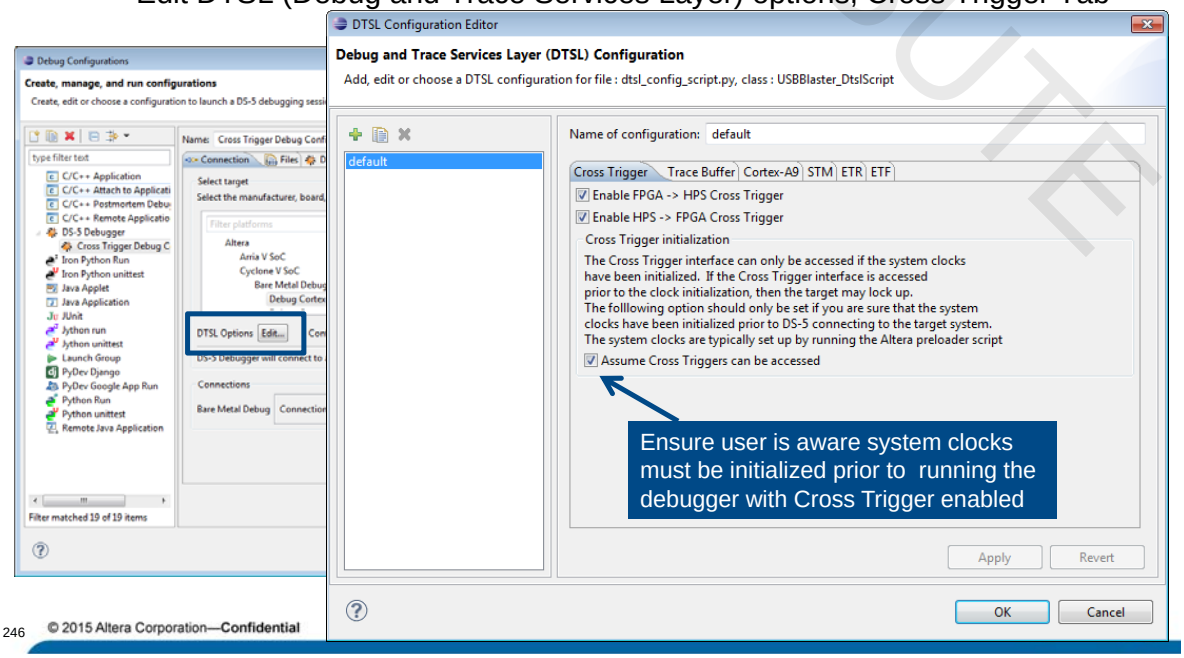


Debug Perspective



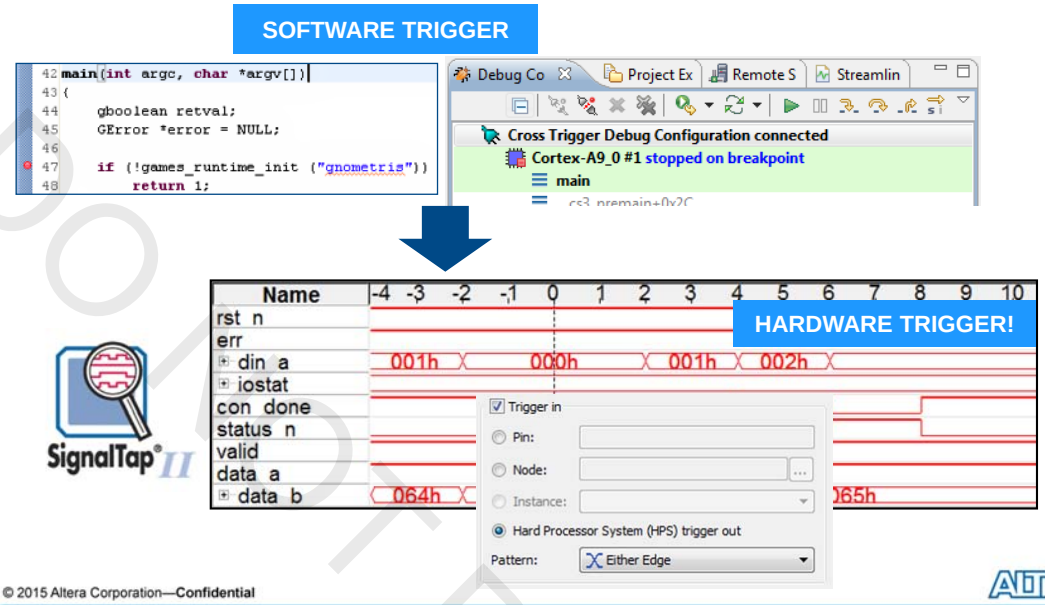
Debugger Configuration for Cross Trigger

- From DS-5 Debug Configurations
  - Edit DTSL (Debug and Trace Services Layer) options, Cross Trigger Tab



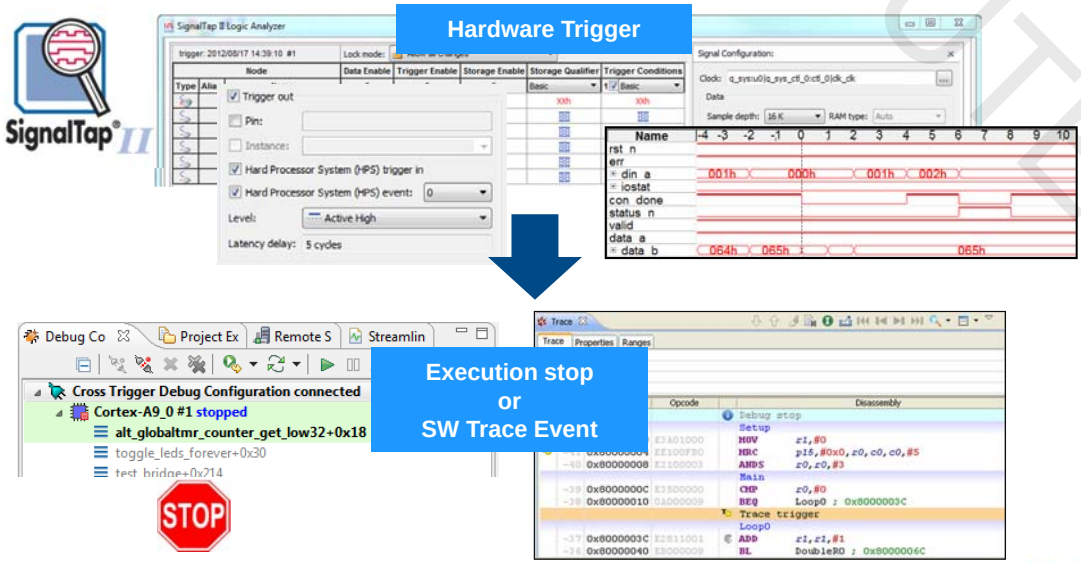
Software to FPGA Cross-Domain Debug

- Trigger from the software world to the FPGA world
  - DS-5 running with cross trigger enabled



FPGA to Software Cross-Domain Debug

- Trigger from the FPGA world to the software world
  - DS-5 running and enable to wait for cross-trigger



Test Your Knowledge

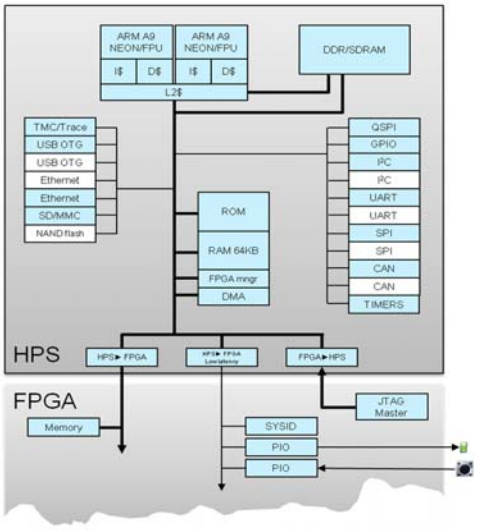
- True or False, users can alter the boot ROM code
  - False
- What's the best way to resolve low-level driver issues?
  a) Always blame the software engineer
  b) Always blame the hardware engineer
  c) Use FPGA-adaptive software debugging capabilities of the SoC to efficiently find and solve the problem

Hardware Design Agenda

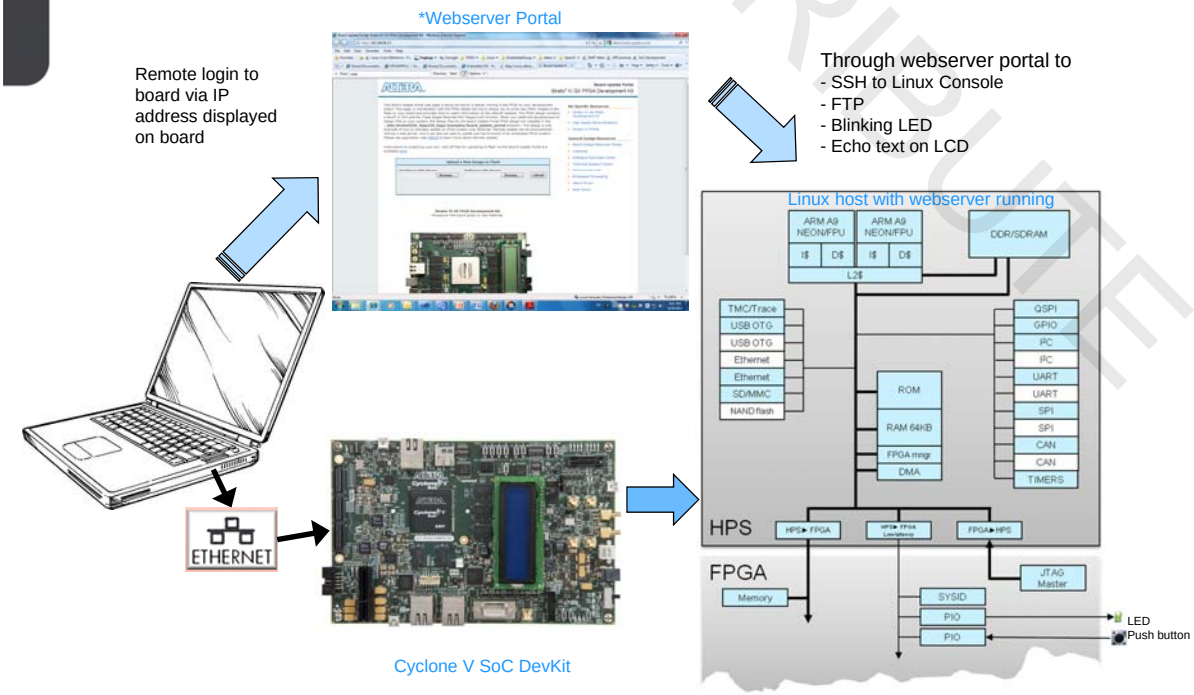
- Hardware design flow with Qsys
- Configuring the HPS IP
- Software handoff
- Avalon/AXI overview
- HPS simulation
- HPS configuration and booting
- SoC debug
- Conclusion

Golden Reference Design

- Complete system design with Linux software support
  - Simple custom logic design in FPGA
  - All source code and Quartus II / Qsys design files for reference
  - Include all compiled binaries- example can run on an Altera SoC Development Kit to jumpstart development



SoC Out-of-Box Reference Design Demo



*Part of the software included with the Golden Hardware Reference Design

SoC Development Boards

- Cyclone V SoC
- Arria V SoC
- Arria 10 SoC
- DE1-SoC education board
- Arrow SOCKit
- Macnica Helio Board
- EBV SoCrates
- and many others

Follow-on Training

- Altera
  - Developing Software for an ARM-based SoC
- Free Altera Online Trainings
  - SoC Hardware Overview
    - The Microprocessor Unit
    - Interconnect and Memory
    - System Management, Debug and GP Peripherals
    - Flash Controllers and Interface Protocols
  - SoC Hardware Design Flow
  - SoC Software Design Flow
- Doulos SoC hardware and software training courses
- ARM training courses:
  - ARM Cortex-A9 MPCore Software Development (3 days)
- AC6 training courses

Summary

You should now be able to

- Explain the pieces HPS in the SoC devices
- Use Qsys to instantiate and configure an HPS component
- Explain the similarities and differences with the Avalon and AXI protocols
- Debug an SoC device using the System Console and SignalTap II logic analyser in conjunction with DS-5
- Explain the hardware handoff files for the software flow

Many Ways to Learn

FREE
Always available
~4 minutes long
YouTube videos

Videos

FREE
Always available
~30 minutes long
>200 topics available

Online Training

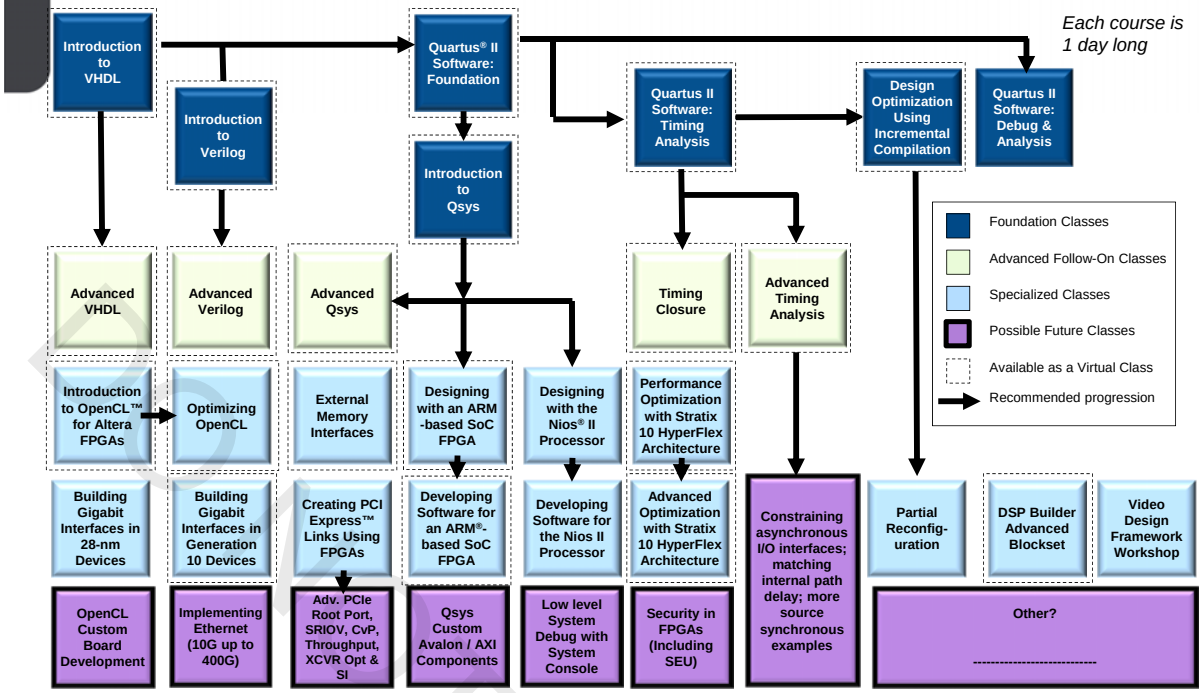
Live over Webex
Ask questions to Altera expert
Taught in ½ day sessions
Class schedules at
www.altera.com/training

Virtual Classes

In-person
Ask questions to Altera expert
1 day long
Class schedules at
www.altera.com/training

Instructor-led Training

Instructor-Led and Virtual Training Curriculum



http://www.altera.com/training

Exercise 3

Debugging

Thank You

Designing with the ARM-Based SoC

Exercise Manual

Software Requirements:

Quartus® II Software v15.0 with the Cyclone® V family installed
SoC Embedded Development Suite 15.0

Hardware Requirements:

Terasic® DE- SoC development kit

http://www.altera.com/customertraining/ILT/Designing_with_ARM_SoC_15_0_v2.zip

Exercise 1

Instantiate the HPS Component in Qsys

Objectives:

- *Add a Hard Processor System (HPS) component to an existing Qsys System*
- *Configure the HPS interfaces and other parameters*

As you proceed through the exercises, be sure to completely read the instructions for each step and sub-step in this lab manual. Use the lines next to each step (____) to keep track of your progress or to check off completed steps in the exercises.

If you have any questions or problems, please ask the instructor for assistance.

Step 1: Set Up an Embedded Hardware Design Project

____ 1. Find the lab materials on your training computer by navigating in the windows explorer to the **C:\altera_trn\Designing_with_ARM_SoC** directory. *This directory will be referred to throughout these exercises as the **project folder**.*

____ 2. If there are already existing subdirectories in the project directory, **delete** them before continuing. This will ensure you are starting fresh with clean files and not files created from a previous class. There should be nothing in the directory other than the self-extracting zip file.

____ 3. Double-click on the file **Designing_with_ARM_SoC_15_0_v2.exe**

____ 4. Select **Unzip** to extract its contents to the **Designing_with_ARM_SoC** folder.

____ 5. Click **Close** when complete.

____ 6. Change directory into the **<*project_folder*>\Labs** directory.

In this folder, you will find the Quartus II project that you will use today.

____ 7. Start the Quartus II version **15.0** Software from the windows start menu

____ 8. Open the **soc_system.qpf** by selecting **File -> Open Project** from the menu bar and then selecting the <*project_folder*>\Labs**soc_system.qpf** file.

Be sure to use File -> Open Project and not File -> Open.

____ 9. Click **Open.**

Next, you will start building your system by instantiating the HPS component.

____ 10. From the Quartus II **Tools** menu, choose **Qsys**

This opens Qsys system integration tool which is required to design with the Hard Processor System, we will talk more about the advantages of using this tool later in the presentation. The Qsys tool is used to generate the HDL file for the system which will then be compiled.

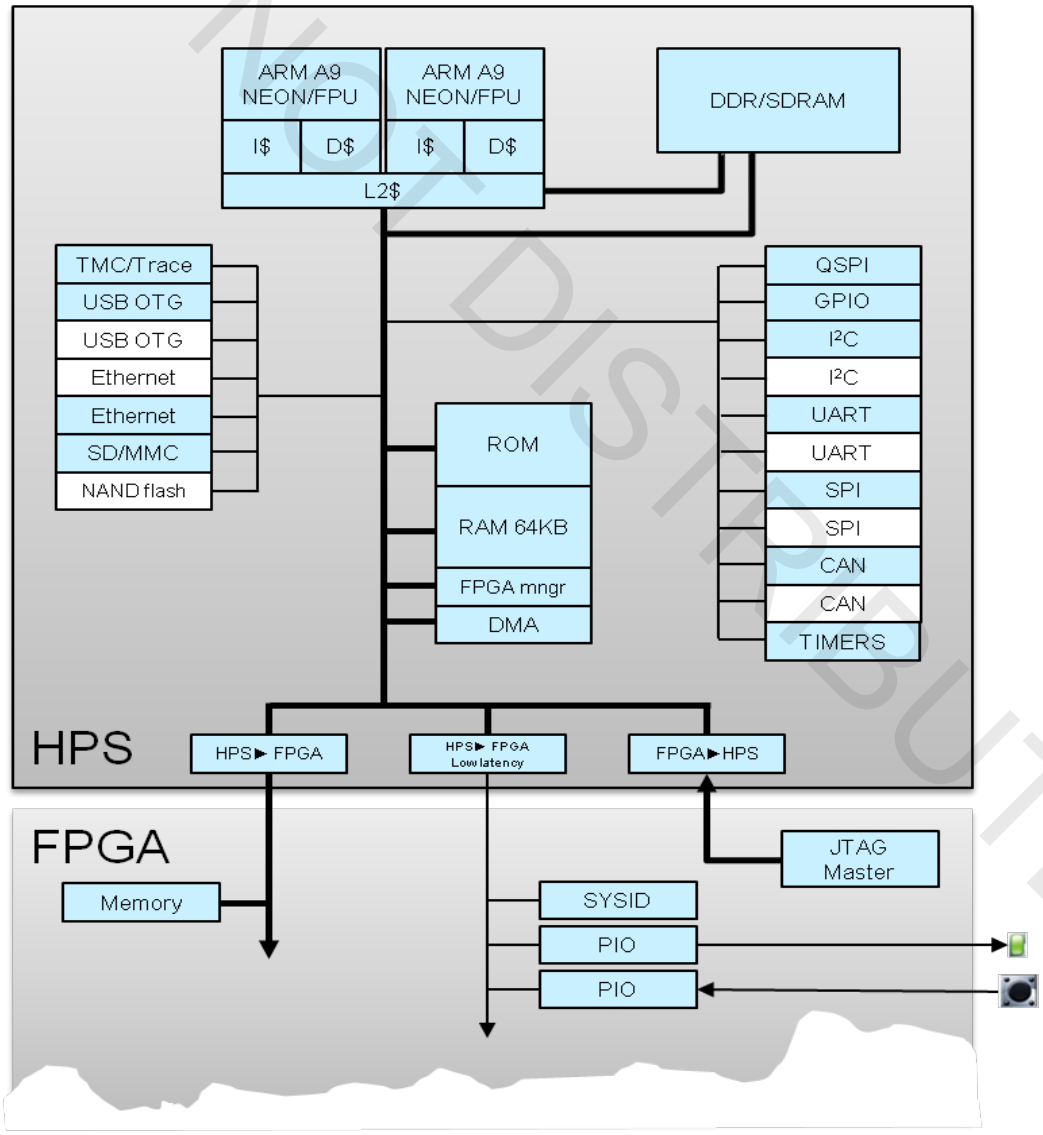
____ 11. Double-click the **soc_system.qsys** file when prompted to open.

*In the interest of speeding your creation of the system along, the Qsys system already contains several components and a **Clock Source** component. If you were to create your own Qsys system from scratch, only the **Clock Source** component would be present at first.*

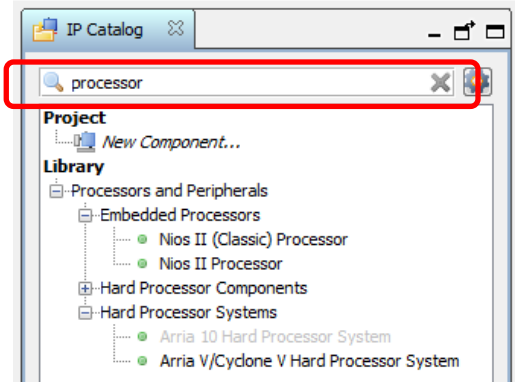
Step 2: Add Hard Processor System (HPS) Component

The HPS component consists of the dual ARM® Cortex™- A9 processor with various peripherals that can be enabled for use in the system. The block diagram below shows the system, divided up into HPS and FPGA portions. The items in the upper HPS portion will be configured now.

There are multiple tabs used to configure the HPS component. These tabs are FPGA interfaces, Peripheral Pin Multiplexing, HPS Clocks, and SDRAM. Each of these tabs will be looked at in sequence to configure them.



____ 1. In the Search box under the **IP Catalog** tab, type "processor".



The search feature is very useful when locating components in the library.

____ 2. Double-click **Arria V/Cyclone V Hard Processor System**.

By double clicking, we are creating an instance of the IP into our current system. This also opens the HPS component dialog box allowing us to customize the component.

We will be discussing every one of the options in the presentation later.

____ 3. On the **FPGA Interfaces** tab, **disable** the **MPU standby and event signal**, which should be enabled by default.

These are internal signals that indicate if the microprocessor is in standby mode and can wake up the CPU. We are disabling these because our design does not have any logic in the FPGA fabric that can take advantage of these features.

Note: It may take up to 2 seconds for the GUI to respond to your selections.

____ 4. Ensure that the **general purpose signals** are **disabled** (default).

These are signals which produce a pair of 32-bit unidirectional general purpose interfaces between the FPGA and HPS. For this exercise, these signals are not needed

____ 5. Enable **System Trace Macrocell hardware events**

This allows custom hardware to inject trace events to the HPS trace bus

____ 6. In the **AXI Bridges** section, ensure that the **FPGA-to-HPS interface width** is set to **64-bit**.

Enabling the FPGA to HPS interface allows masters within the FPGA to access to HPS peripherals. 64-bits is the width of the interface to the hardware logic.

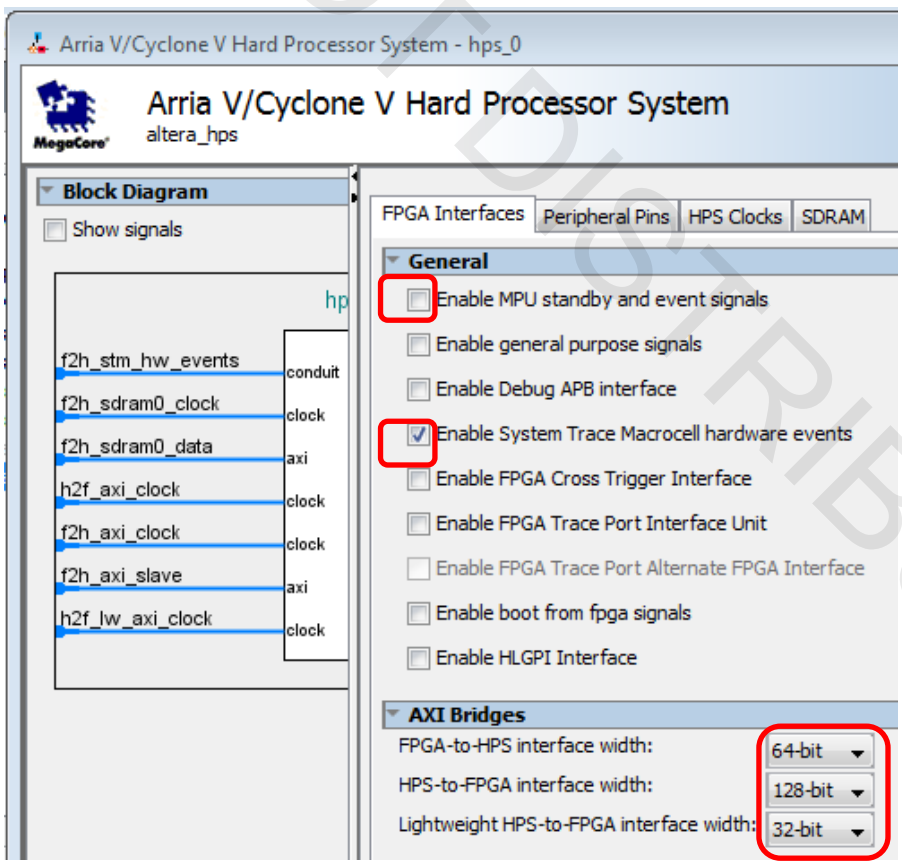
____ 7. Set the **HPS-to-FPGA interface width** to 128-bit

Enabling the HPS to FPGA interfaces allows the HPS master to access the FPGA peripherals. Setting it to a wider width allows the interface to run on a slower clock but maintaining throughput.

____ 8. Ensure that the **lightweight HPS-to -FPGA interface width** is set to **32-bit**.

Unlike the regular HPS to FPGA bridge which is tuned for throughput, the lightweight HPS-to-FPGA bridge is tune for latency. Using two HPS to FPGA bridges allows us to differentiate traffic type to increase performance. Control and status type of access can now use the lightweight bridge while data transfers will be assigned to the regular high throughput HPS to FPGA Bridge.

The settings should now look like this:

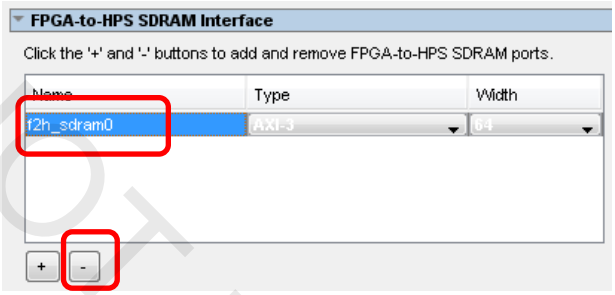


Scrolling down the FPGA interface tab, there are still more options available to set. There are the FPGA-to-HPS SDRAM interface settings, Reset settings and DMA Peripheral Request settings.

____ 9. Scroll down the FPGA interface window until you see the **FPGA-to-HPS SDRAM Interface** and select the **f2h_sdram0** interfaces in the window.

The FPGA-to-HPS SDRAM interface allow FPGA masters to directly read/write from the HPS SDRAM. If you don't need cache coherency support this is the fastest way to access the SDRAM. If you do need cache coherency then you can use the FPGA-to-HPS Bridge through the Accelerator Coherency Port to access the SDRAM.

____ 10. Click the **"–"**button to remove the interface since we won't be using it.



____ 11. In the **Resets** section enable the three FPGA-to-HPS reset requests while ensure HPS-to-FPGA cold reset and warm reset handshake are disabled.

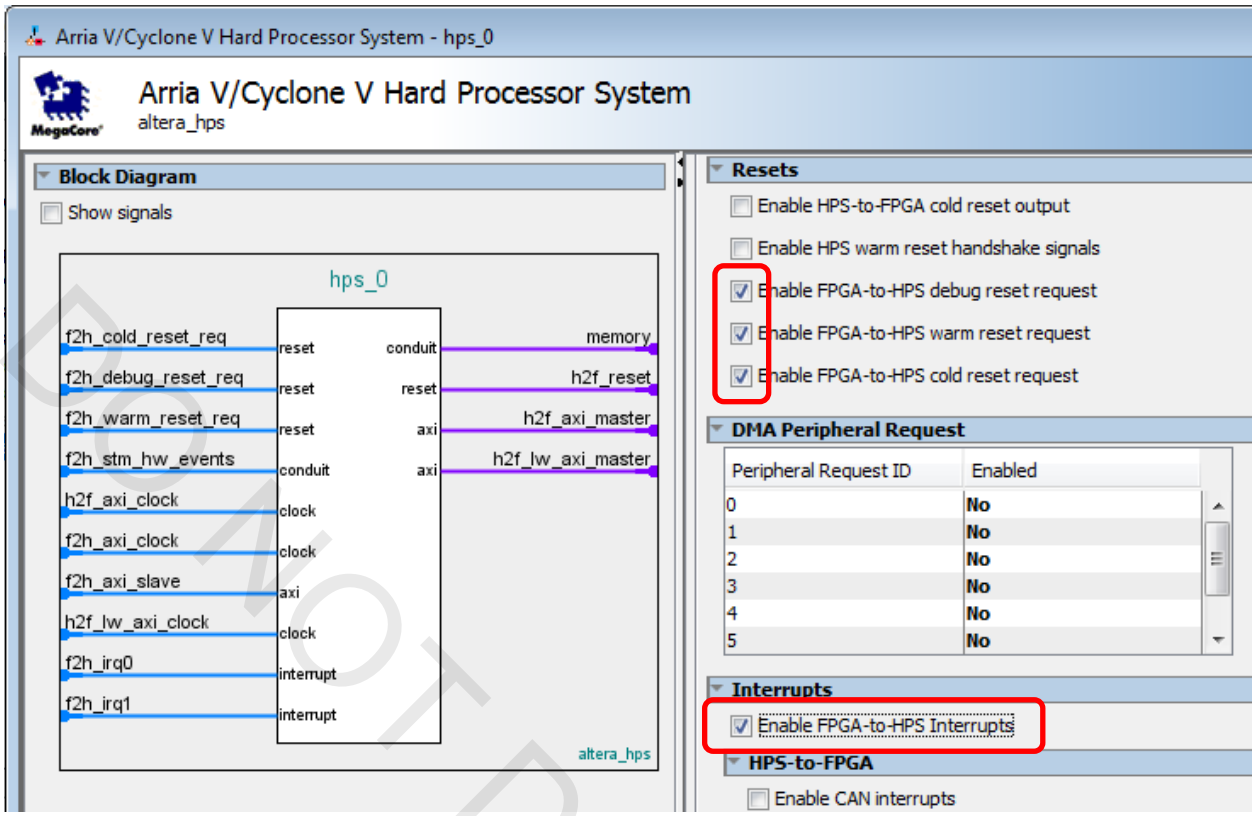
Our FPGA components will be able to reset the HPS.

____ 12. Scroll down to the **DMA Peripheral Request** section and verify that all rows indicate "**No**" under the Enabled column.

Enabling the DMA peripheral request would allow soft logic in the FPGA fabric to communicate with the DMA controller in the HPS through one of the eight request IDs. Our design does not use this capability.

____ 13. Scroll down to the **Interrupts** section and **enable** the **FPGA-to-HPS interrupts** option.

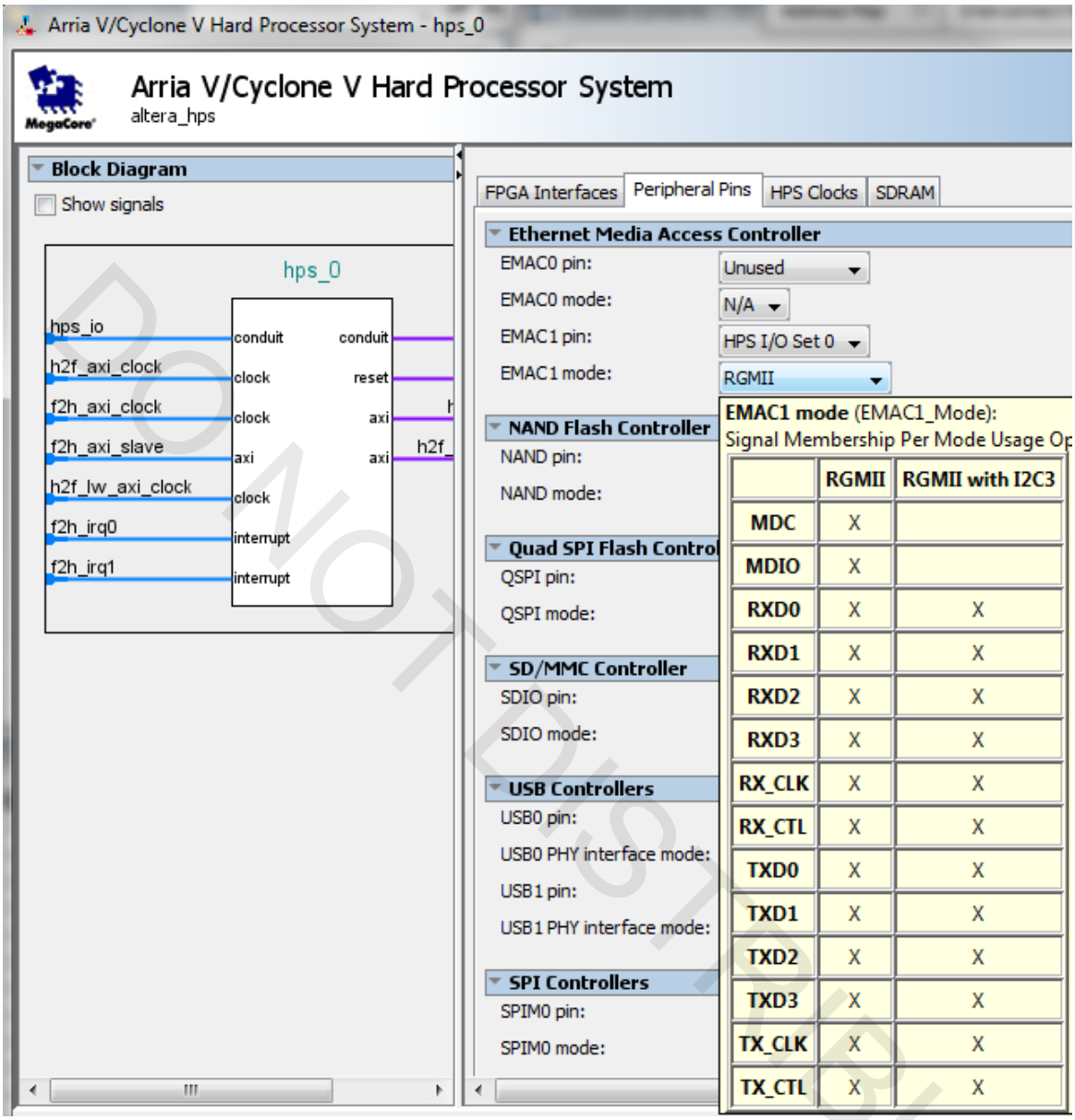
This will provide 64 bits of interrupts for the FPGA components to send interrupts to the Generic Interrupt Controller in the HPS.



Step 3: Configure HPS Peripherals (MAC, NAND, QSPI, SDIO, USB)

Under the Peripheral Pins tab, there are options to enable the HPS peripherals. There are more peripherals than there are available IOs to support them, so choices will have to be made regarding which HPS component uses which pins. To accommodate this there may be multiple I/O sets that we can use for each component.

By hovering with a mouse over each interface in the Peripheral Pins tab (EMAC1 mode in the example below), a list of the signals used in that interface pops up.



____ 1. Select the **Peripheral Pins** tab.

This tab allows us to enable HPS components as well as to choose which one of the IOs are assigned to those enabled components.

____ 2. Under the **Ethernet Media Access Controller** section, set **EMAC1 pin** to **HPS I/O Set 0** and ensure the **EMAC1 mode** is set to **RGMII**.

We have now enabled Ethernet MAC 1 using the MDIO PHY management interface.

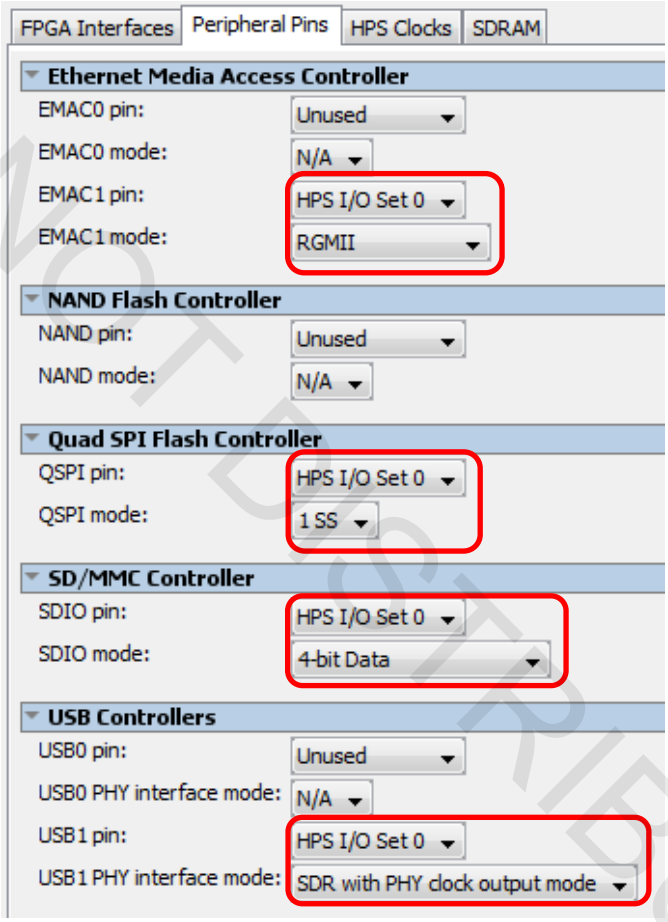
____ 3. Under the **QSPI Flash Controller** section, set **QSPI pin** to **HPS I/O Set 0.**

____ 4. Ensure the **QSPI mode** is set to **1 SS** (slave select) for use with 1 device.

____ 5. In the **SDMMC/SDIO Controller** section, set **SDIO pin** to **HPS I/O Set 0**.

____ 6. Set the **SDIO mode** to **4-bit data**.

____ 7. Under the **USB Controllers** section, set **USB1** pin multiplexing to **HPS I/O Set 0**, and ensure that the **USB1 PHY interface mode** is set to **SDR with PHY clock output mode**.

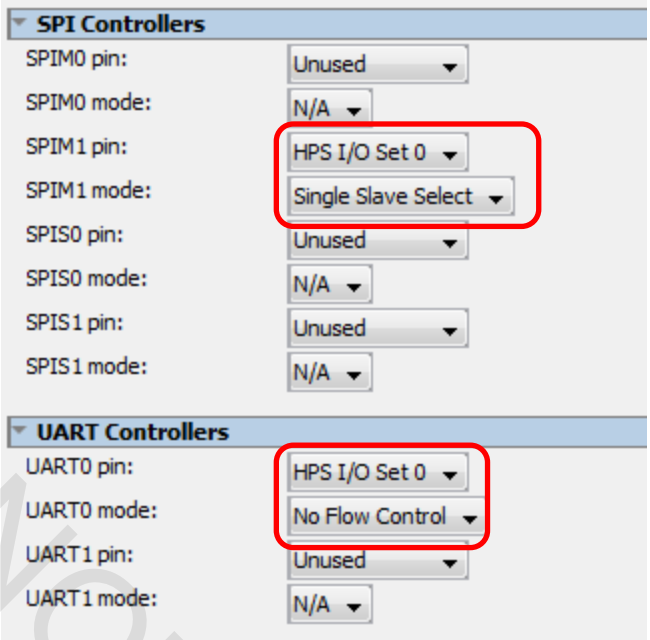


____ 8. Under the **SPI Controllers** section, set **SPIM1 pin** to **HPS I/O Set 0.**

____ 9. Set the **SPIM1** mode to **Single Slave Select.**

____ 10. Under the **UART Controllers** section, set **UART0 pin** to **HPS I/O Set 0**.

____ 11. Set the **UART0 mode** to **No Flow Control.**



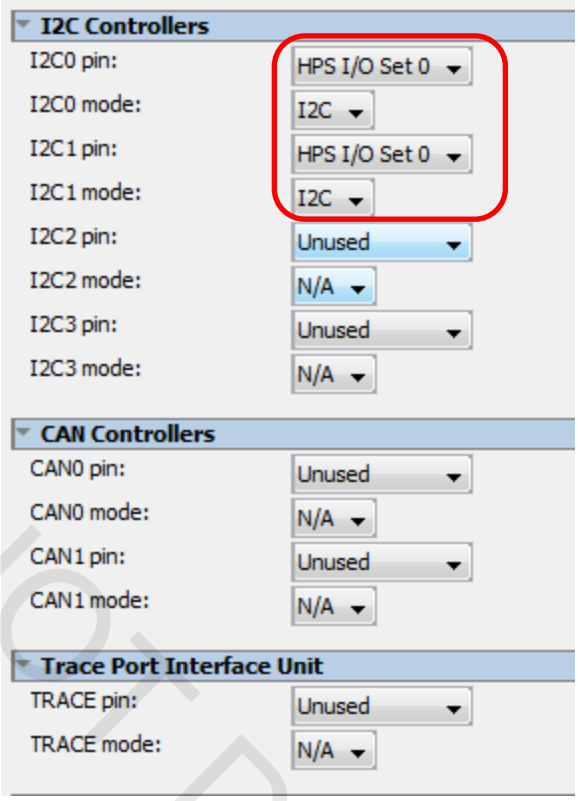
____ 12. Under the **I2C Controllers** section, set **I2C0 pins** to **HPS I/O Set 0.**

____ 13. Ensure that the **I2C0 mode** is set to **I2C.**

____ 14. Set **I2C1 pins** to **HPS I/O Set 0.**

____ 15. Ensure that the **I2C1 mode** is set to **I2C.**

____ 16. Ensure CAN Controllers and the Trace Port Interface unit are all Unused



____ 17. Check the messages window in the HPS component configuration window and verify that there are **NO errors** regarding conflicts.

If there are errors, they would appear at in the message window. Here's an example

Error: hps_0: Refer to the Peripherals Mux Table for more details. The selected peripherals 'EMAC1' and 'NAND' are conflicting.

Two interfaces cannot share the same pins. The peripherals mux table at the bottom shows if there are pins with invalid assignments, such as having multiple interfaces. (The bold outlines shown in the screen shot below indicate which signals are in use.)

Peripherals Mux Table

PinName	mux_select_1	mux_select_2	mux_select_3
NAND_ALE	QSPI.SS3 (Set1) (Set0)	EMAC1.TX_CLK (Set0)	NAND.ALE (Set0)
NAND_CE	USB1.D0 (Set1)	EMAC1.TXD0 (Set0)	NAND.CE (Set0)
NAND_CLE	USB1.D1 (Set1)	EMAC1.TXD1 (Set0)	NAND.CLE (Set0)
NAND_RE	USB1.D2 (Set1)	EMAC1.TXD2 (Set0)	NAND.RE (Set0)
NAND_RB	USB1.D3 (Set1)	EMAC1.TXD3 (Set0)	NAND.RB (Set0)

If an error similar to the one above appears, check the peripherals mux table to find out which interfaces are in conflict and correct them.

Only one signal per row should have a bold outline in the peripherals mux table as follows:

Peripherals Mux Table

PinName	mux_select_1	mux_select_2	mux_select_3
NAND_ALE	QSPI.SS3 (Set1) (Set0)	EMAC1.TX_CLK (Set0)	NAND.ALE (Set0)
NAND_CE	USB1.D0 (Set1)	EMAC1.TXD0 (Set0)	NAND.CE (Set0)
NAND_CLE	USB1.D1 (Set1)	EMAC1.TXD1 (Set0)	NAND.CLE (Set0)
NAND_RE	USB1.D2 (Set1)	EMAC1.TXD2 (Set0)	NAND.RE (Set0)
NAND_RB	USB1.D3 (Set1)	EMAC1.TXD3 (Set0)	NAND.RB (Set0)

____ 18. In the **Peripherals mux** table, find **GPIO9** under the **GPIO** column (5th column of the table).

Since this particular pin RGMII0_TX_CTL is not being used by another component, we're able to configure it as a General Purpose IO.

____ 19. **Enable** GPIO09 by clicking the **GPIO09** button.

Peripherals Mux Table

PinName	mux_select_1	mux_select_2	mux_select_3	* GPIO
RGMII0_RX_CTL		USB1.D7 (Set0)	EMAC0.RX_CTL (Set0)	GPIO08
RGMII0_TX_CTL			EMAC0.TX_CTL (Set0)	GPIO09
RGMII0_RX_CLK		USB1.CLK (Set0)	EMAC0.RX_CLK (Set0)	GPIO10

____ 20. Repeat the above 2 steps for these additional GPIO pins:

GPIO35
GPIO40
GPIO48
GPIO53
GPIO54
GPIO61

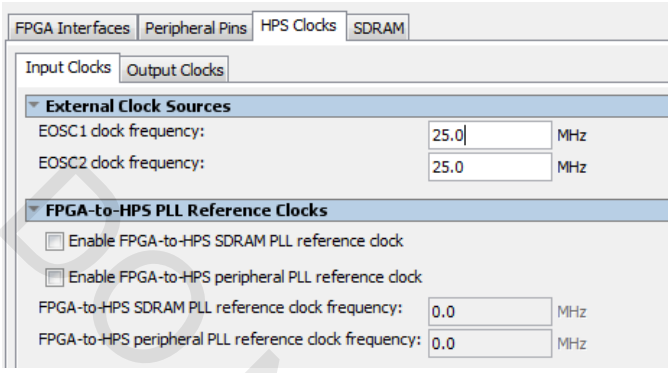
Step 3: Configure HPS Clocks

On the HPS Clocks tab the specific clock sources and frequencies are specified. Remember from the presentation that these properties are all managed by the Clock Manager Component. When you make the selections on this tab the information is used to generate the 2nd stage bootloader software which executes these selections.

____ 1. Click on the **HPS Clocks** Tab

____ 2. Click on the **Input Clocks** Tab

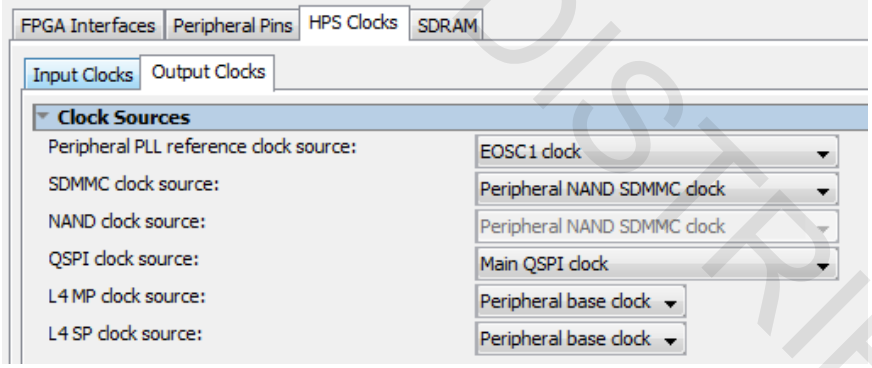
____ 3. Ensure EOSC1 and EOSC2 clock frequencies are set to 25Mhz and all FPGA-to-HPS References clocks are disabled.



In this design we will not be sourcing any clocks from the FPGA fabric nor sending any clocks to the FPGA fabric.

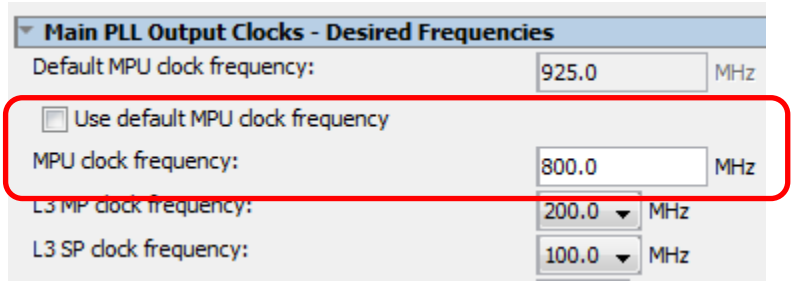
____ 4. Click on the Output Clocks Tab.

____ 5. Ensure reference is set to EOSC1 clock and clock sources are set as the screen capture.



____ 6. Disable "**Use default MPU clock frequency**"

____ 7. Instead set the MPU clock frequency to 800 MHz



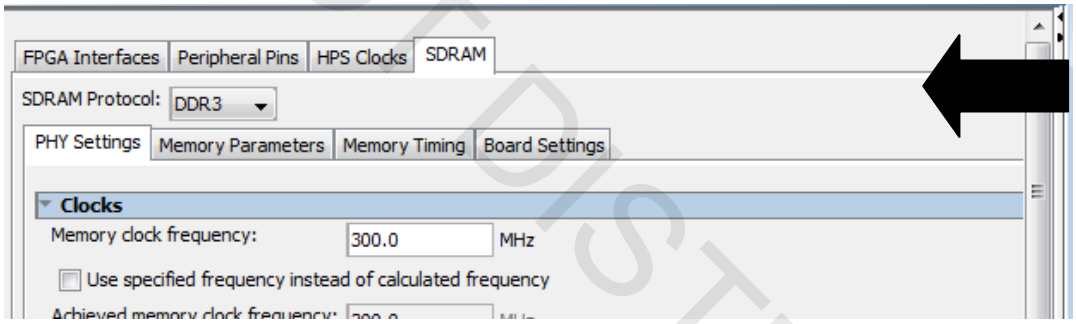
Step 4: Configure SDRAM

Under the SDRAM tab, there are options to set the SDRAM parameters for the DDR3 on the board. There are four tabs for the SDRAM configuration, PHY Settings, Memory Parameters, Memory Timing, and Board Settings.

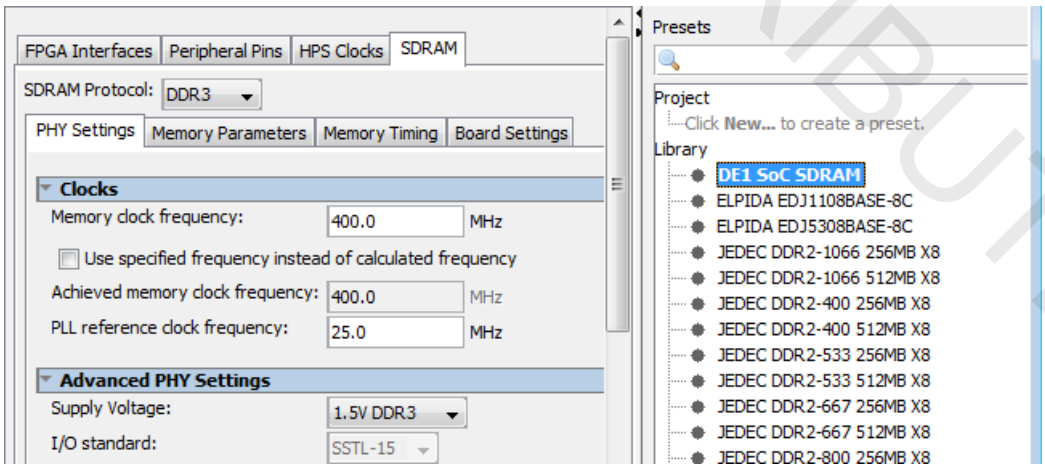
The settings need to match the datasheet of the Micron DDR3 device on the board. In the interest of time, you will use a preset rather than change all the settings manually. The preset for that configuration stores all of the relevant settings. You can always create your own preset by clicking the ***New...*** *button then save the preset with a custom name.*

____ 1. Click the **SDRAM** tab in the HPS component wizard.

Ensure the Preset window is visible. If not, the window is hidden to the right of the wizard. Make the window visible by dragging the right border of the dialog box to the left.



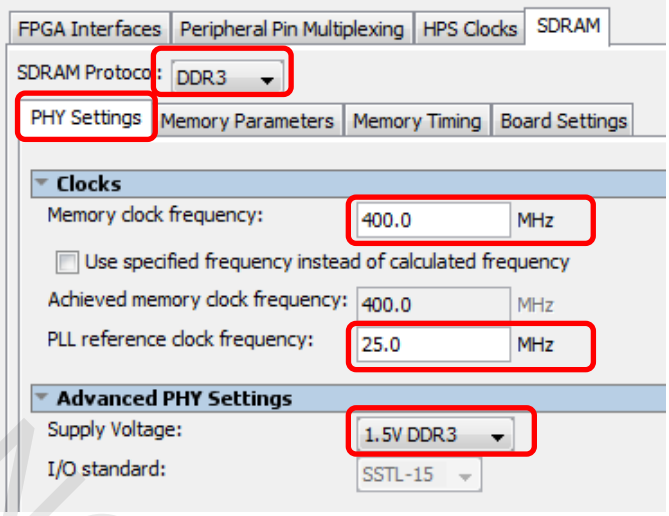
____ 2. Select the **DE1 SoC SDRAM** preset in the presets window.



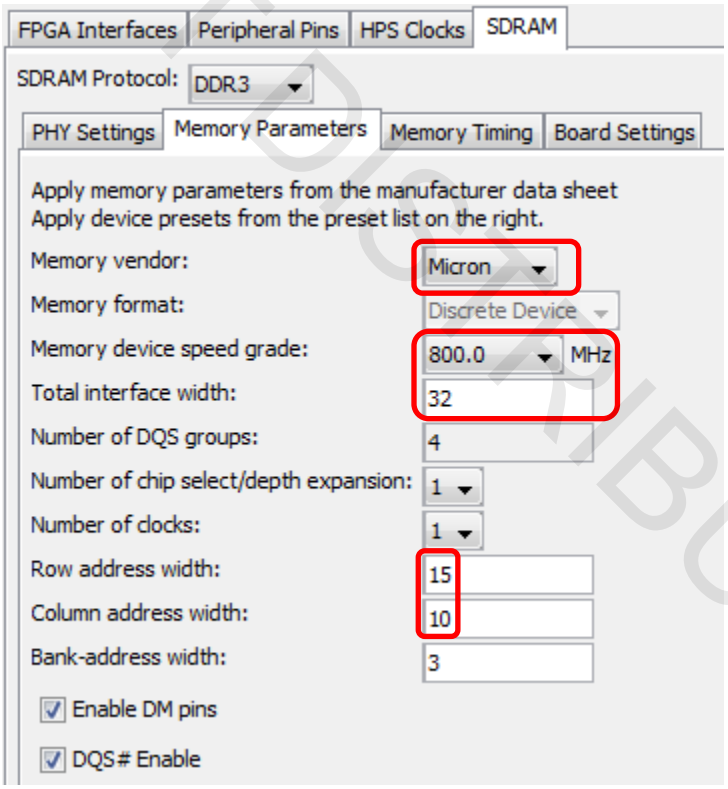
____ 3. Click **Apply**. You should see **DE1 SoC SDRAM** preset appear in bold

When a preset is in bold, it signifies all of the settings in the preset have been applied.

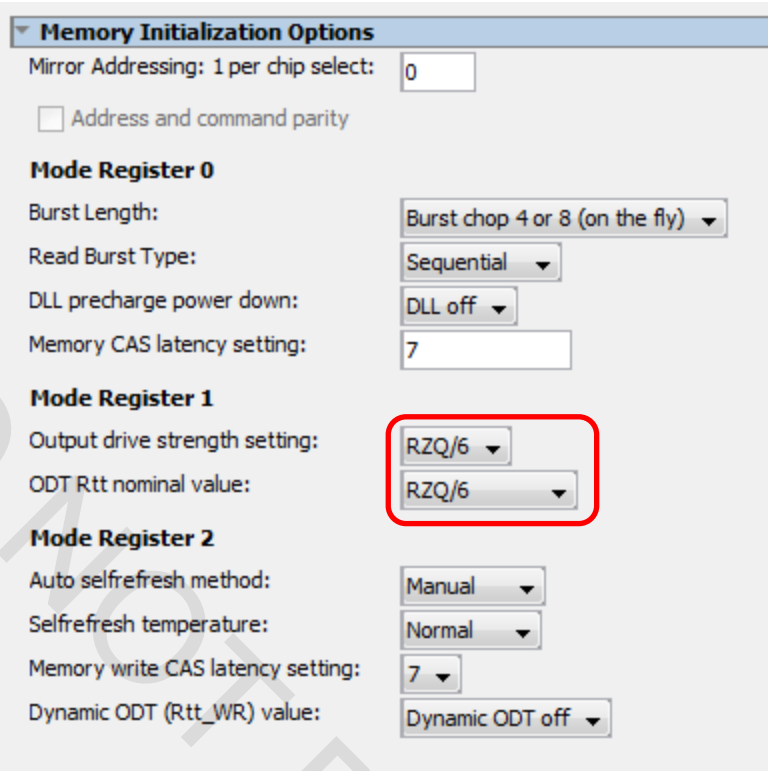
____ 4. Click the **PHY Settings** tab and verify that the circled settings match the following.



____ 5. Click the **Memory Parameters** and verify that the circled settings match the following.

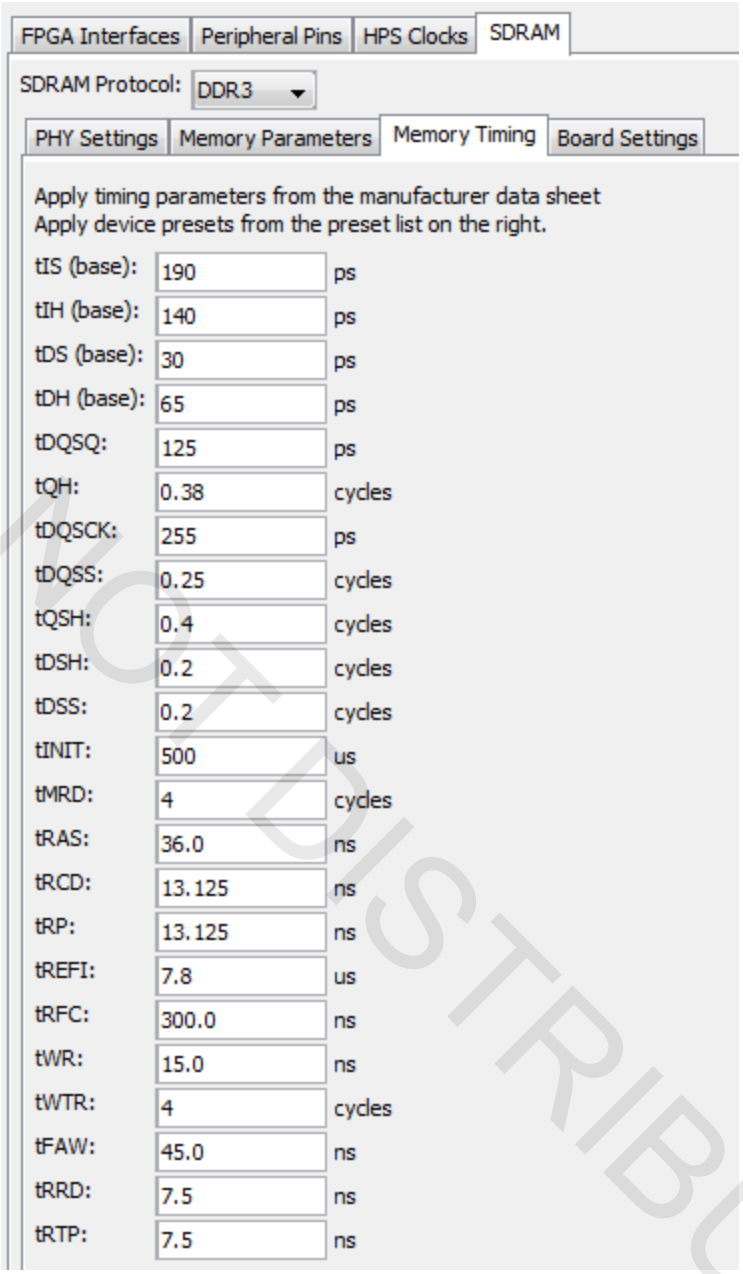


____ 6. Scroll down to the **Memory Initialization Options** section and verify that **ODT Rtt nominal value** is set to **RZQ/6**.



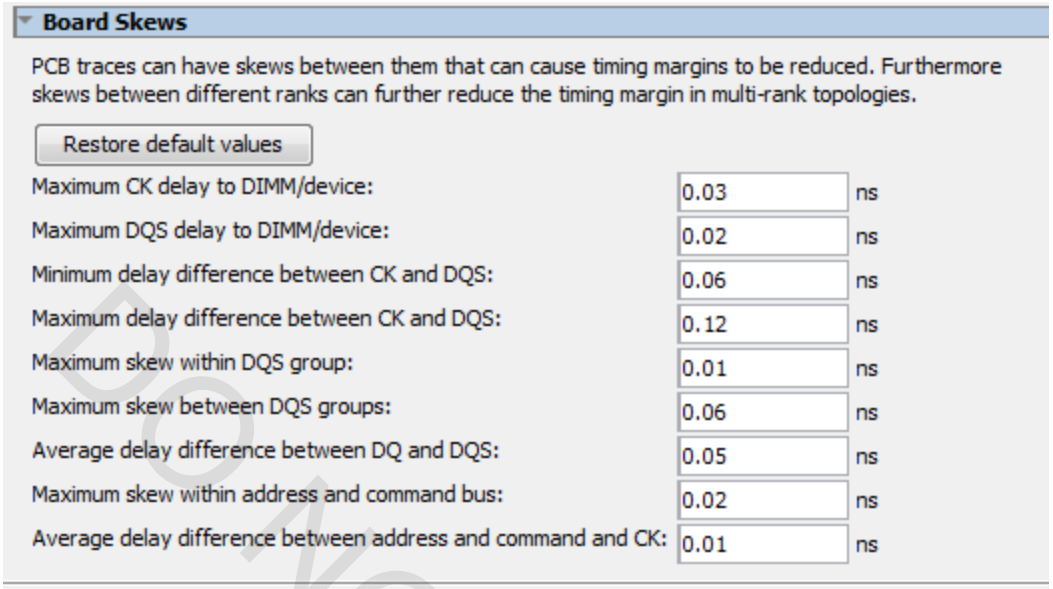
____ 7. Click on the **Memory Timing** tab and verify that the circled settings match the following screen shot.

(These values are found on the data sheet for the Micron part. If further knowledge on these is required, please attend our instructor led External Memory Interfaces course or one of our free Memory Interface online trainings where we will discuss in detail how to implement a successful modern high speed memory controller.



____ 8. Click the **Board Settings** tab and verify that "**Use Altera's default settings"** is selected under both the **Setup and Hold Derating** section and the **Channel Signal Integrity** section.

____ 9. Scroll down to the **Board Skew** section and verify that the board skews are set as follows:



____ 10. Click **Finish** in the HPS configuration window to accept the configuration settings and close the window.

There will be errors in the Qsys system as we haven't made any of the necessary connection to the HPS yet, we will resolve those in exercise 2.

Exercise Summary

- Added an HPS component to a Qsys System
- Configured an HPS component

END OF EXERCISE 1

Exercise 2

Complete the HPS Qsys System

Objectives:

- *Connect the HPS instantiation with the FPGA system in Qsys*
- *Instantiate additional FPGA components in Qsys*
- *Generate the Qsys system*

Our completed system will include the following components:

Hard Processor System ***On-chip memory*** ***Clock Source***

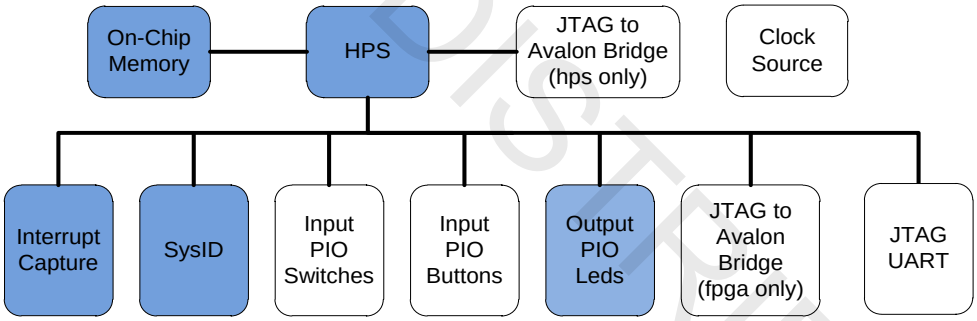
JTAG to Avalon® Master Bridge (To Master HPS Components)

JTAG to Avalon Master Bridge (To Master FPGA Components)

Interrupt Capture Module ***System ID peripheral*** ***JTAG UART***

Parallel IOs for DIP switch, push buttons, and LEDs

You will be building the following system:



This system will have a processor and a number of embedded peripherals, including interrupt capturer, PIOs, and JTAG bridges. It also has a sysid register used to identify the system built. The specific components you will be adding are colored dark in the diagram above. Note that we've already instantiated the HPS component in the previous lab.

Throughout this lab, as we add components, reordering components in the Qsys system view can make verification easier. The screenshots provided will often be organized to make the connections more obvious, but components do not have to be ordered that way.

Step 1: Connect the HPS Component

____ 1. Back in the Qsys **System Contents** window, find the row associated with the HPS component we just added, it should be at the bottom and named **hps_0**

____ 2. Export the **h2f_reset** Reset Output port by double clicking in the **Export** column and accepting the default name, **hps_0_h2f_reset** by pressing the enter key.

By exporting we're making the signal available outside of the Qsys system. This way we can connect the signal to pin or to other non-Qsys FPGA logic.

____ 3. Export the **f2h_cold_reset_req** Reset input port by double clicking in the **Export** column and accepting the default name, **hps_0_f2h_cold_reset_req** by pressing the enter key.

____ 4. Export the **f2h_debug_reset_req** Reset input port by double clicking in the **Export** column and accepting the default name, **hps_0_f2h_debug_reset_req** by pressing the enter key.

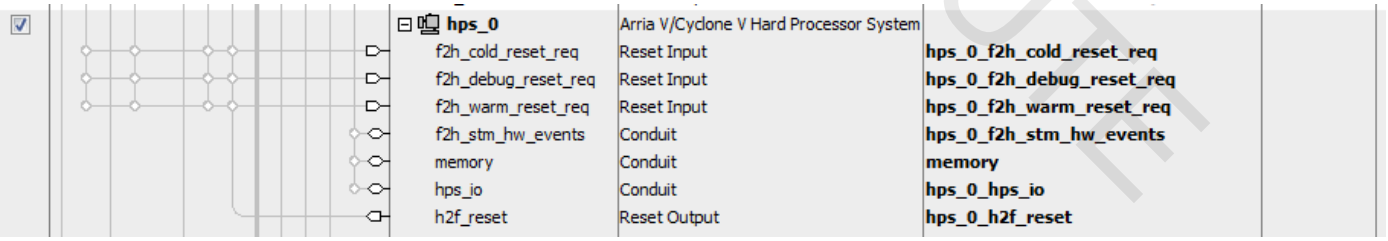
____ 5. Export the **f2h_warm_reset_req** Reset input port by double clicking in the **Export** column and accepting the default name, **hps_0_f2h_warm_reset_req** by pressing the enter key.

____ 6. Export the **f2h_stm_hw_events** conduit port by double clicking in the **Export** column and accepting the default name, **hps_0_stm_hw_events** by pressing the enter key.

____ 7. Rename the exported name of the **hps_io** conduit port of the HPS component by selecting **hps_io** in the **Export** column and typing **hps_0_hps_io**.

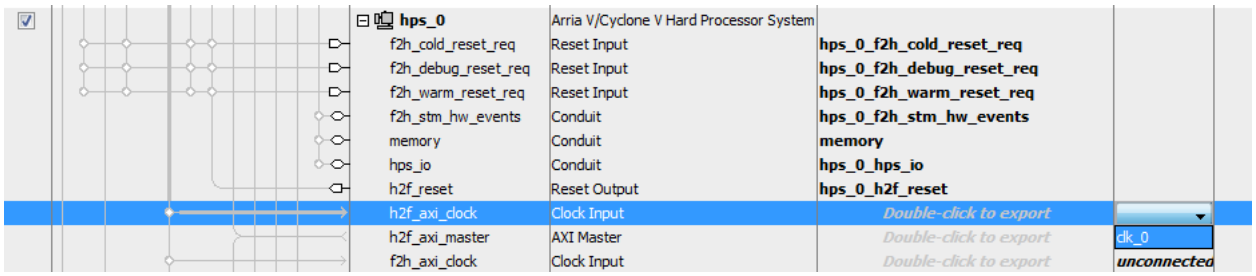
____ 8. Verify that the **memory conduit port** is exported and named **memory**. If it is not, export that interface with the name **memory**.

These exported connections can be seen below. It needs to appear Exactly as shown.



hps_0	Arria V/Cyclone V Hard Processor System	
f2h_cold_reset_req	Reset Input	**hps_0_f2h_cold_reset_req**
f2h_debug_reset_req	Reset Input	**hps_0_f2h_debug_reset_req**
f2h_warm_reset_req	Reset Input	**hps_0_f2h_warm_reset_req**
f2h_stm_hw_events	Conduit	**hps_0_f2h_stm_hw_events**
memory	Conduit	**memory**
hps_io	Conduit	**hps_0_hps_io**
h2f_reset	Reset Output	**hps_0_h2f_reset**

____ 9. Connect the **Clock Input** interface, **h2f_axi_clock,** on the HPS by choosing **clk_0** in the drop-down menu in the **Clock** column of the HPS instance.

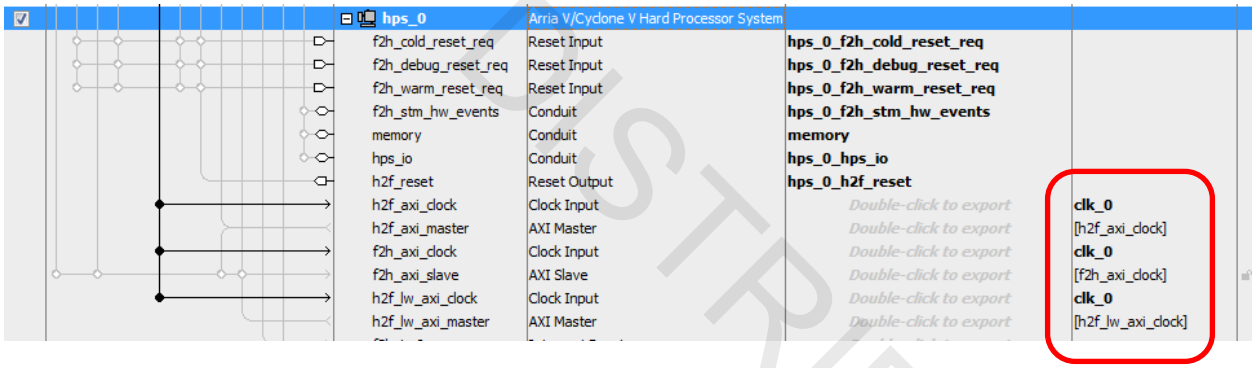


There are many ways to connect the clock. You could have also right clicked on the interface or used the connections panel.

____ 10. Connect the **Clock Input** interface, **f2h_axi_clock,** on the HPS by choosing **clk_0** in the drop-down menu in the **Clock** column of the HPS instance.

____ 11. Connect the **Clock Input** interface, **h2f_lw_axi_clock,** on the HPS by choosing **clk_0** in the drop-down menu in the **Clock** column of the HPS instance.

The clocks on the HPS should be connected as shown in the following picture.



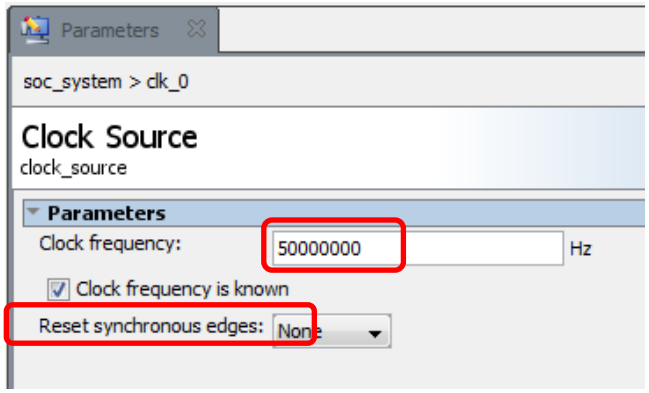
All errors should be resolved now, there are still some warnings which we will resolve soon.

<u>Step 2</u>: <u>Adding Additional System Components</u>

____ 1. Double-clock the **Clock Source** component (named **clk_0**) and ensure that the **Clock Frequency** is set to **50 MHz** to match the oscillator on the development board.

We're using the clock source component to bring a clock into the Qsys system and we've already connected it to the FPGA side of the HPS bridges.

____ 2. Ensure that **clock frequency is known is** checked.



____ 3. Close the Clock Source **Parameters** window

____ 4. Erase "processor" from the Search box under the **IP Catalog** tab. If that's still there.

Now we're going to add components by browsing in the library.

____ 5. Add a **System ID Peripheral** to the Qsys system:

a. Double-click the **System ID Peripheral** in the **Basic Functions > Simulation;Debug and Verification > Debug and Performance** folder in the Qsys IP Catalog pick-list window to open up its configuration window.

b. Enter **0xacd51302** as the **32 bit system ID.**

c. Click **Finish** to add the component to the Qsys system.

The System ID peripheral component contain registers for the ID as well as the Qsys generation timestamp, it can be used to identify the current system programmed in the FPGA.

*For more information on this component and others, please consult the **Embedded Peripheral IP User Guide** found on www.altera.com*

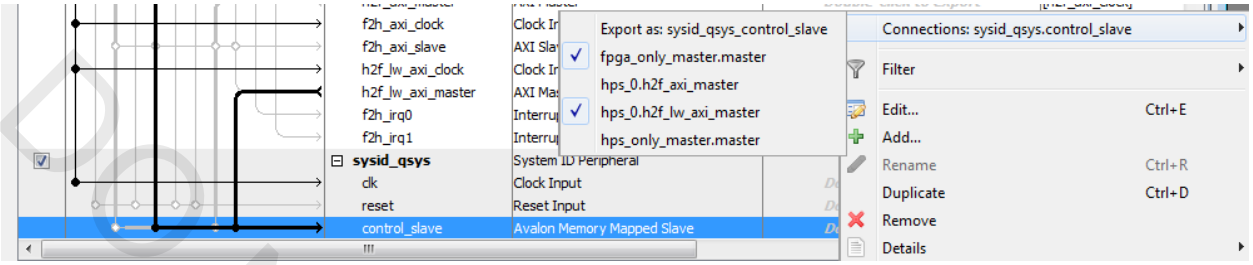
____ 6. Connect the System ID component to the Qsys system.

a. Connect the **clk input** port of the System ID Peripheral to **clk_0** by selecting it in the clock column drop down menu.

b. Rename the System ID component by clicking on its name in the **Name** column and typing **sysid_qsys.**

c. Connect the System ID as shown below using the right click menu or connections panel

Component	Port		Component	Port
sysid_qsys	control_slave	To	hps_0	h2f_lw_axi_master
sysid_qsys	control_slave		fpga_only_master	master

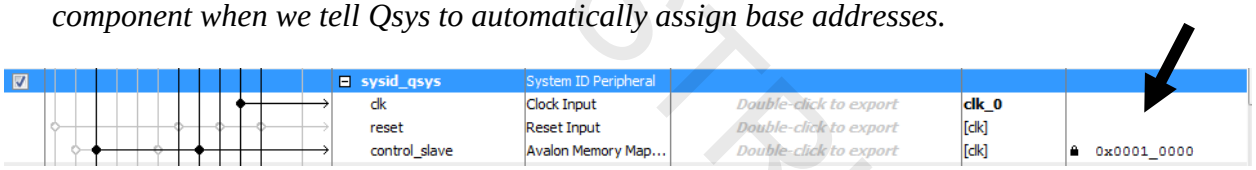


Notice we're connecting the System ID slave interface to the HPS to FPGA lightweight bridge as accesses to the System ID component is consider a Control/Status access.

d. Set the **base address** for the System ID Peripheral by double-clicking the address for the System ID peripheral in the **Base** address column and typing **0x0001_0000**.

e. Click the **lock icon** next to the base address to lock it.

By locking the address, we won't allow Qsys to assign any other address to this component when we tell Qsys to automatically assign base addresses.

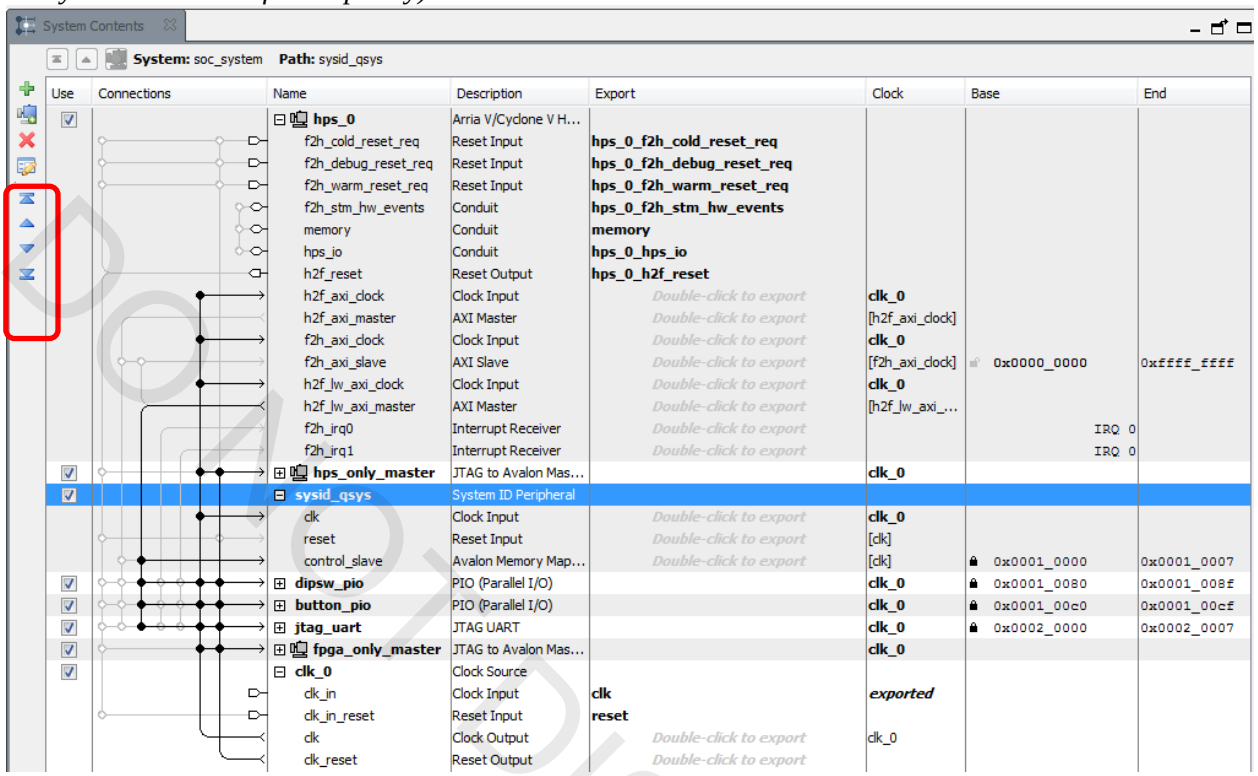


____ 7. Move the **HPS component** to the top of the Qsys system by selecting the hps_0 component and clicking the "**move to top**" button, on the Qsys tool bar to the **left** of the System Contents window.

When we move the component in the System Contents window, we're not altering any connections, only the appearance of the system.

____ 8. Move the **System ID component** right below the **hps_only_master** component as shown in the following screen shot.

Your systems should resemble the following: (This diagram shows components already added to the system minimized for simplicity)



____ 9. Add the On-Chip RAM component:

a. Double click the **On-Chip Memory (RAM or ROM)** component from the IP Catalog in the **Basic Functions > On Chip Memory** folder to open the configuration window.

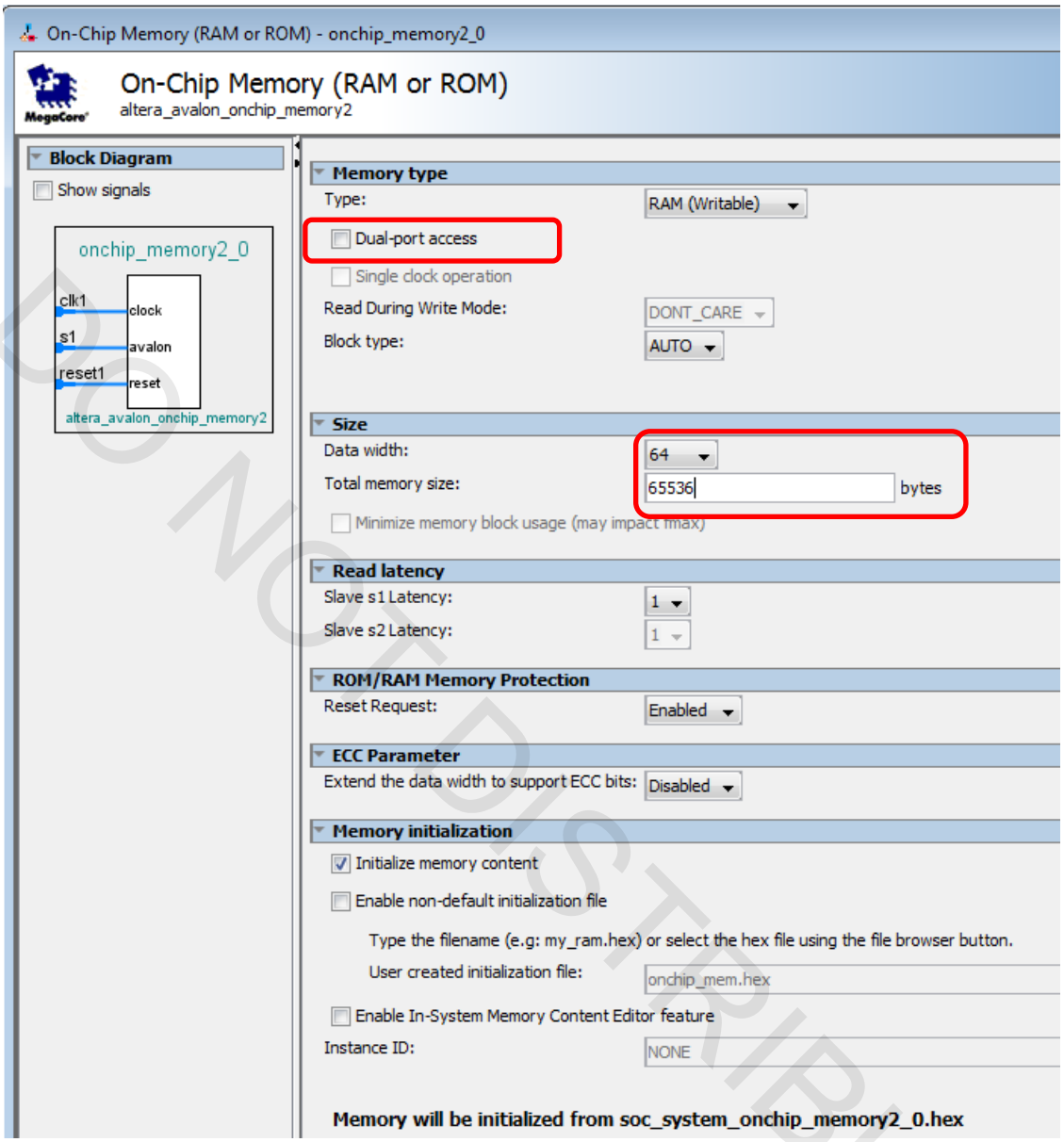
b. Set the **Data width** to **64 bits.**

c. Set the **Total memory size** to **65536 bytes.**

d. Ensure **Dual-Port Access** is **disabled**.

e. Ensure **Initialize memory content** is **Enabled.**

f. Click **Finish.**



The on-chip ram component can utilize the memory blocks in the FPGA

____ 10. Connect the On-chip RAM component to the Qsys system:

a. Verify that the On-Chip memory component is named **onchip_memory2_0.** *If not, click the name and change it.*

b. Connect the **clk1** port of the On-chip RAM to **clk_0** by selecting **clk_0** in the **Clock** column drop down menu.

c. Connect the On-Chip memory slave port **s1** as shown below:

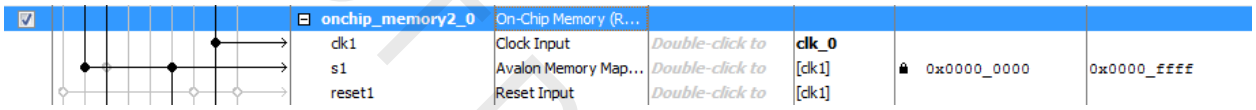
Component	Port		Component	Port
onchip_memory2_0	**s1**	**To**	**hps_0**	**h2f_axi_master**
onchip_memory2_0	**s1**		**fpga_only_master**	**master**

Notice we're connecting the on-chip RAM to the high throughput HPS to FPGA bridge as this connection is used for data movement.

d. Ensure the **base address** for the On-Chip memory is set to **0x0000_0000**. *If not, double-click the address for the On-Chip Memory in the address column and type 0x0000_0000.*

e. Click the **lock icon** next to the base address to lock it.

____ 11. Move the On-Chip RAM memory component so that it is located just below the HPS component.



____ 12. Add Interrupt Capture Module:

a. Double click the **Interrupt Capture Module** component from **Project -> Other** folder to open the configuration window for our custom component.

This component is in the Project folder as this is a custom component.

b. Verify the **NUM_INTR** is set to **32**

c. Click **Finish.**

The Interrupt Capture Module captures the interrupts coming in and makes the information available on a memory mapped interface.

____ 13. Connect the Interrupt Capture Module to the Qsys system:

a. Connect the **clk** port of the Interrupt Capturer to **clk_0**

b. Connect the **avalon_slave_0** interface on the **Interrupt Capturer** component to the **master** port of the **fpga_only_master** component**.**

c. Set and **lock** the **base address** for the Interrupt Capture Module to **0x0003_0000**

____ 14. Add the LED Parallel IO Component

a. In the IP Catalog, find **Processors and Peripherals → Peripherals → PIO** and double click it to instantiate it.

b. Set the Width to 10

c. Set direction to Output

d. Set Output Port Reset Value to 0xF

e. Click Finish

____ 15. Rename the component **led_pio** by single click on the name or using the right click menu

____ 16. Connect the led_pio

a. Set the led_pio clk input to clk_0

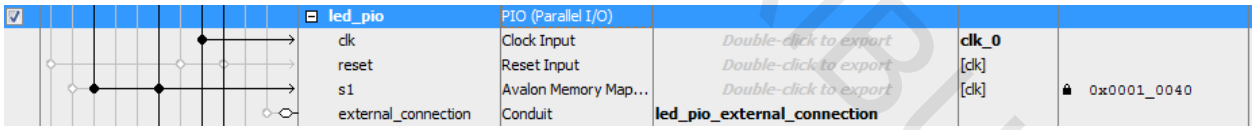
b. Connect the led_pio s1 slave port to fpga_only_master.master

c. Connect the led_pio s1 slave port to hps_0.h2f_lw_axi_master

d. Set the address of the led_pio s1 port to 0x10040 and lock it

e. Export the external_connection Conduit interface as the default **led_pio_external_connection**

f. Move the led_pio component to be just below button_pio



____ 17. Ensure these components clocks are connected as described below:

hps_only_master	**clk_0**
jtag_uart	**clk_0**
button_pio	**clk_0**
dipsw_pio	**clk_0**
led_pio	**clk_0**

____ 18. Connect the remaining components as shown:

Component	Port		Component	Port
hps_only_master	**master**	**To**	**hps_0**	**f2h_axi_ slave**
jtag_uart	**avalon_jtag_slave**		**hps_0**	**h2f_lw_axi_ master**
button_pio	**s1**		**hps_0**	**h2f_lw_axi_master**
dipsw_pio	**s1**		**hps_0**	**h2f_lw_axi_master**
led_pio	**s1**		**hps_0**	**h2f_lw_axi_master**

____ 19. Connect all the reset interfaces in the design by selecting **Create Global Reset Network** from the **System** menu.

This will connect OR together all of the reset outputs in the Qsys system and connect them to all reset inputs in the Qsys system

____ 20. Auto-assign the base addresses for all the components so that there are no overlapping addresses by selecting **Assign Base Addresses** in the **System** menu.

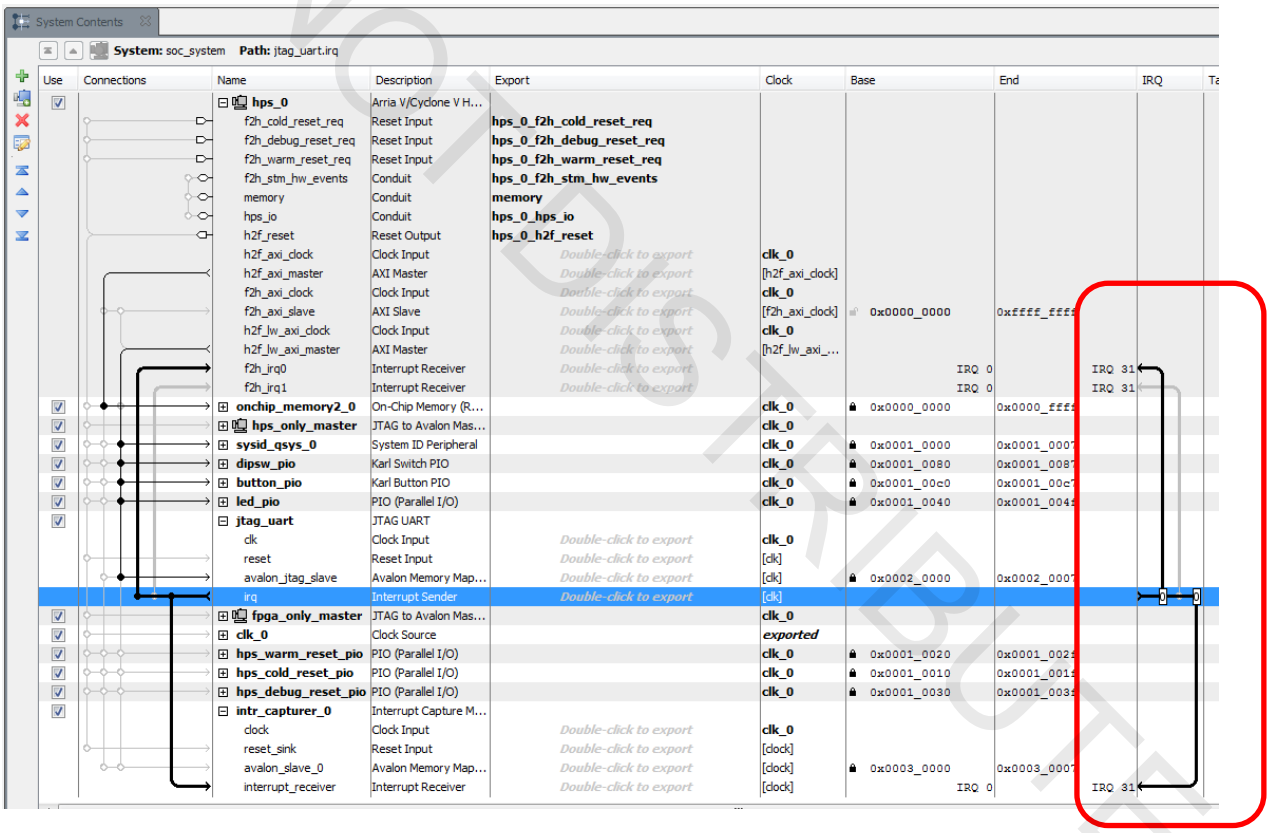
This should have no effect in our case since we've manually assigned and locked all addresses.

Step 3: Establish IRQ Priorities

____ 1. In the IRQ Column (You'll need to scroll to the right), connect the **IRQ** line on the **jtag_uart** component to the HPS f2h_irq0 and Interrupt Capturer in the IRQ column.

Doing this allows the components to interrupt processor 0 in the HPS as well as sending a signal to the interrupt capturer custom component so FPGA Only Master has access to the information.

The IRQ assignments should match the screen shot below. Ensure the jtag_uart gets priority ***0****(highest priority).*



Step 4: Verify Your System

____ 1. Verify that you connections are correct by checking against the following table:

*Your **Qsys System Contents** page should now be connected as shown in the following table. The order of the components doesn't matter, just their connectivity. Ensure the **led_pio_external_connection, dipsw_pio_external_connection** and **button_pio_external_connection** conduits are exported and named properly to ensure correct mapping in the Quartus II software.*

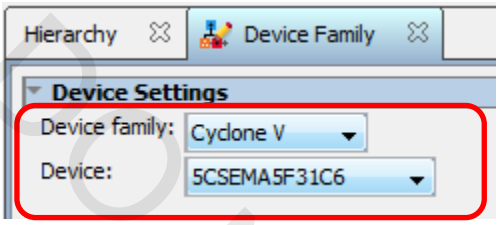
*It's easiest to verify by right click on these ports and choose **Connections.***

Component	Port	Connections
clk_0	clk_in	exported as **clk**
	clk_in_reset	exported as **reset**
	clk	All Components
	clk_reset	All Components except the hps0 exported ones
dipsw_pio	conduit_in	Exported as **dipsw_pio_conduit_in**
	conduit_out	Exported as **dipsw_pio_conduit_out**
button_pio	conduit_in	Export as **button_pio_conduit_in**
	conduit_out	Exported as **button_pio_conduit_out**
led_pio	external_connection	exported as **led_pio_external_connection**
HPS	h2f_axi_master	onchip_memory2_0.s1
	h2f_lw_axi_master	jtag_uart.avalon_jtag_slave
		sysid_qsys.control_slave
		button_pio.s1
		dipsw_pio.s1
		led_pio.s1
	f2h_axi_slave	hps_only_master.master
fpga_only_master	master	jtag_uart.avalon_jtag_slave
		intr_capture_0.avalon_slave_0
		sysid_qsys.control_slave
		button_pio.s1
		led_pio.s1
		dipsw_pio.s1
		onchip_memory2_0.s1
		hps_debug_reset_pio.s1
		hps_cold_reset_pio.s1
		hps_warm_reset_pio.s1

____ 2. Ensure the **Device Family** tab exists, if not enable it from the Qsys view menu.

____ 3. Verify that the **Device Family and Device** and other project settings match the screen shot.

Qsys should have pulled this information from your Quartus II .qsf file. (If it has not done so, please consult your instructor.)



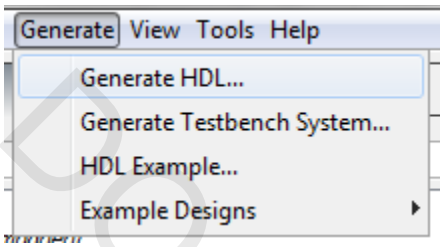
The Device Family tab communicate to Qsys which device you're generating the system for.

____ 4. In the **Message** box, ensure there are no more remaining errors and warnings. *If so, you must fix them before proceeding.*

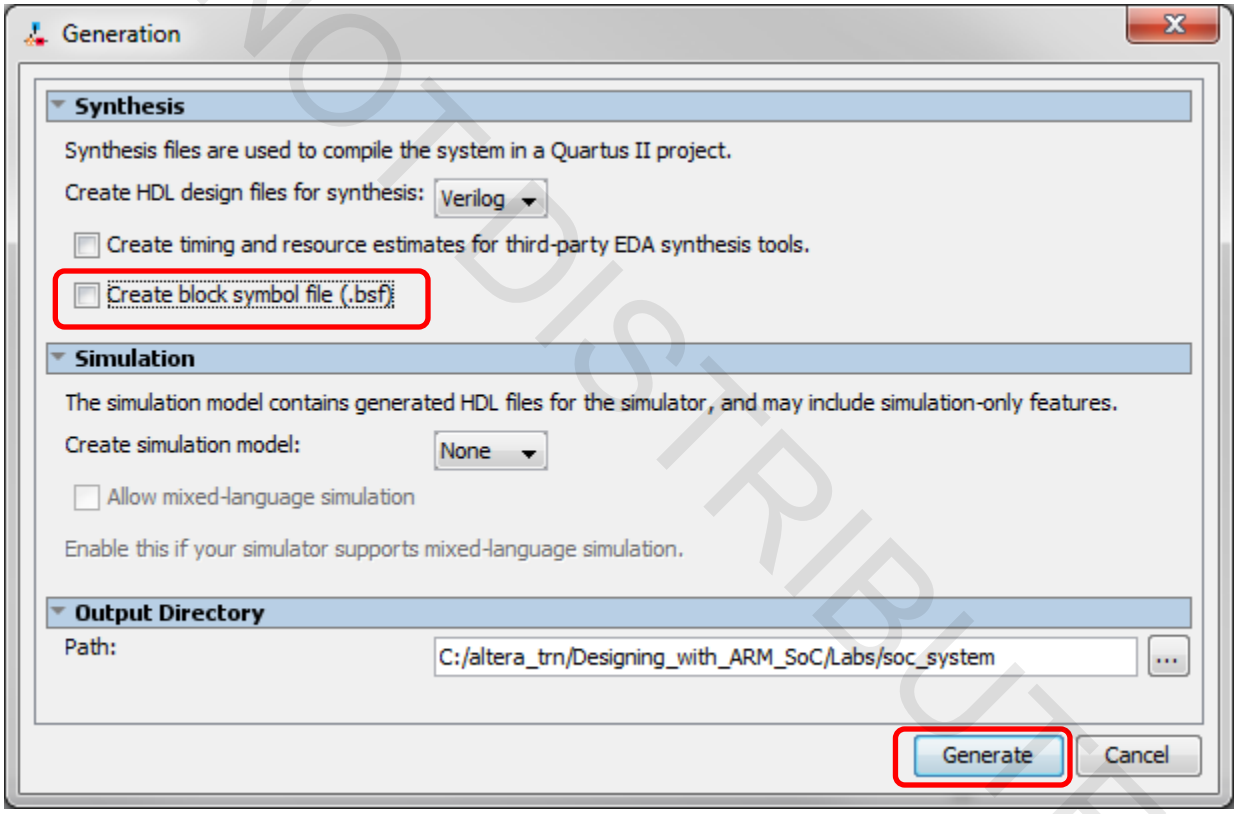
Step 5: Generate the Qsys System

____ 1. Save your Qsys system by selecting **Save** from the **File** menu.

____ 2. Click the **Generate HDL…** option under the **Generate** menu.



____ 3. Uncheck **Create block symbol file** in the Generate dialog box if it's checked.



____ 4. Leave everything else to their defaults and press the **Generate** button.

This will take a few minutes.

Qsys will now create the parameterized hardware system. Qsys can generate either VHDL or Verilog model of the system although we're using Verilog today.

____ 5. After the Qsys software has finished generating the system, close Qsys and return to the **Quartus II software**.

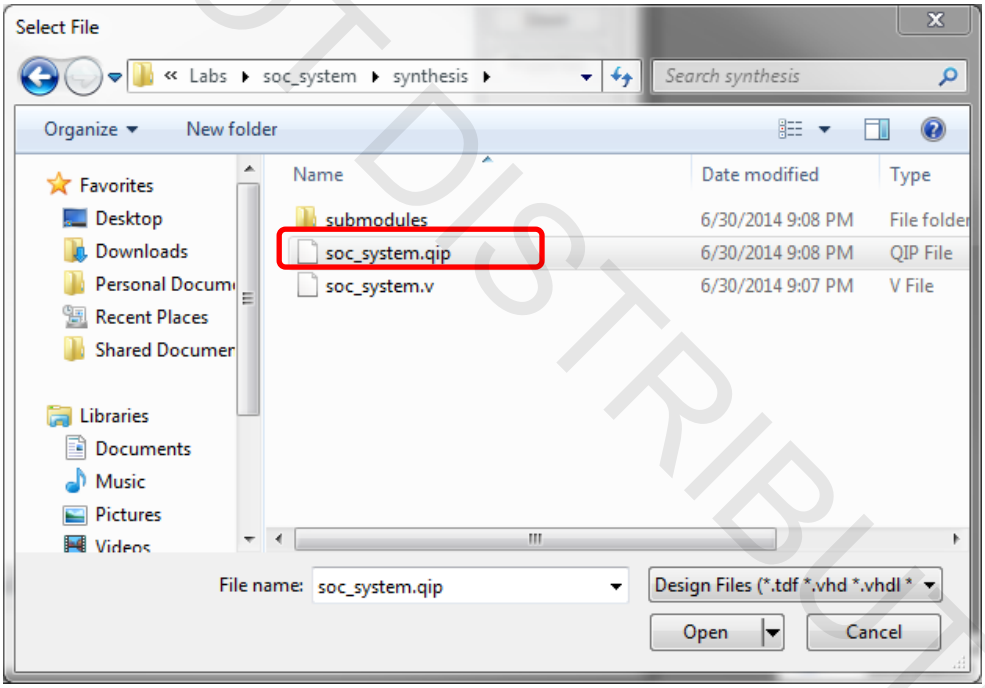
____ 6. Click OK if you see a window telling you you've created an IP Variation in the .qsys file.

____ 7. From the **Project** menu **select Add/Remove Files in Project**.

____ 8. Browse to the **<project_folder>/Lab/soc_system/synthesis** folder by pressing the button next to the **File Name** field in the Quartus II **Settings** page.

____ 9. Select the **soc_system.qip** file.

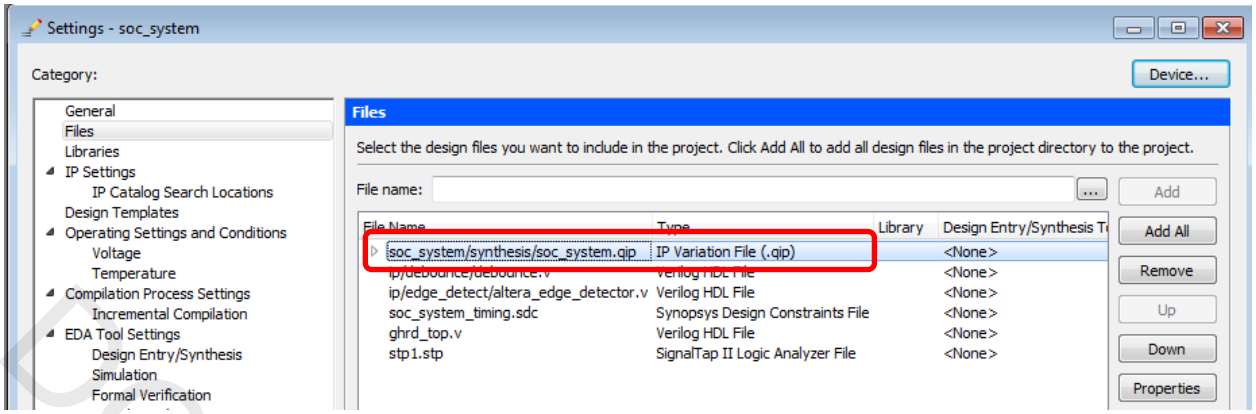
The .qip file is an index file created by Qsys to configure the Quartus II project with HPS pin outs and also loads the variety of source files needed to compile the Qsys system without having to add them all individually.



____ 10. Click **Open**.

____ 11. In the Quartus II Settings page click **Add** to actually add the file to the project.

If you forget to do this, the file will not actually be part of the project.



*<u>Note</u>: The **soc_system_timing.sdc** file used for timing analysis purposes and few other verilog source files has already been added for you.*

____ 12. Click **OK** to close the Settings page.

<u>Step 6</u>: <u>Run the Pin Assignments Script and Compile the Design</u>

Signals entering or exiting the FPGA device need to be assigned physical locations and other pin properties on the device I/O. Since the HPS is a hard core IP, pin assignments, other than SDRAM memory pins, do not need to be specified. The pin assignments were made when the HPS was instantiated, i.e. when the peripheral IO muxing options was set.

In the following steps, you will source a Tcl script created by the Qsys tool to set up a number of I/O assignments for the SDRAM device. A synthesized netlist is required in order to run this script.

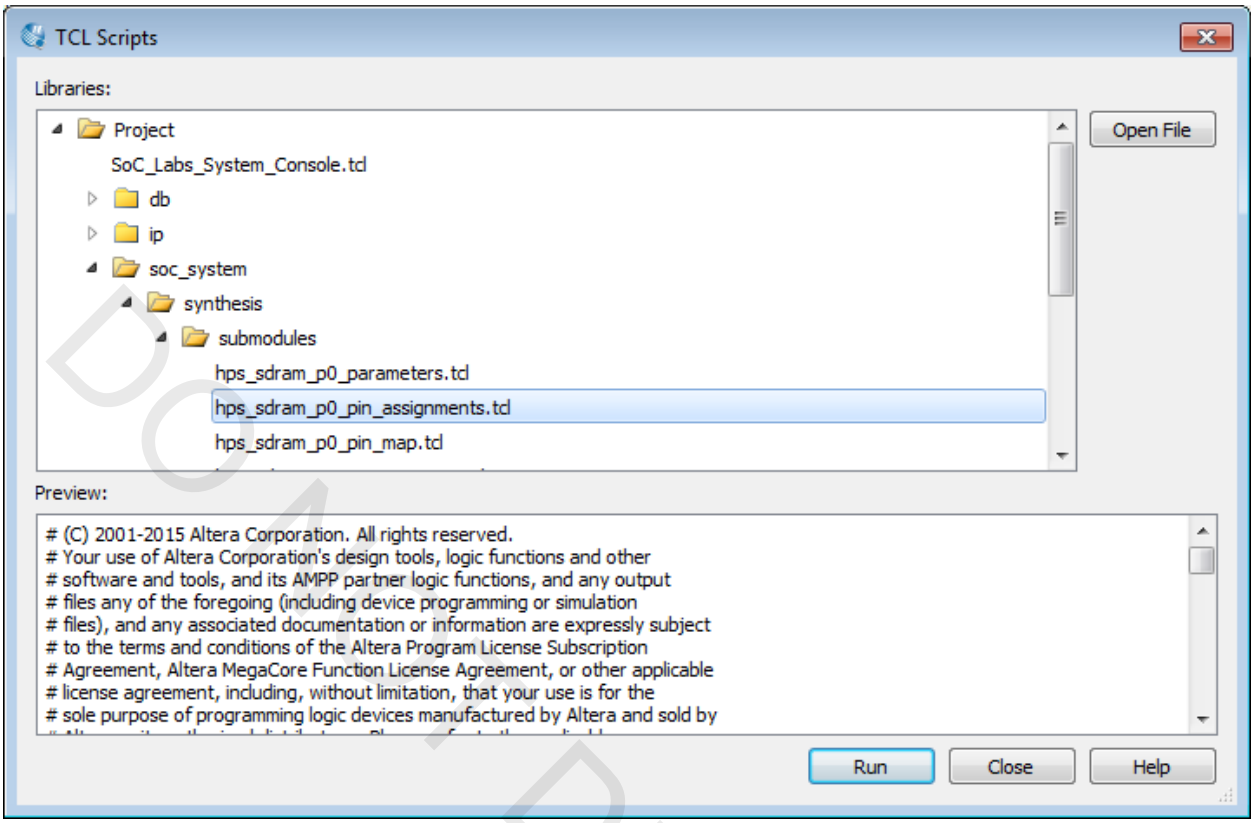
____ 1. Synthesize the design by selecting **Processing** > **Start** > **Start Analysis & Synthesis.**

____ 2. Click **OK** when synthesis is complete.

This may take 5 minutes. Make sure there weren't any errors. Ignore any warnings.

____ 3. Select **Tools** > **Tcl Scripts…**

____ 4. Select the **hps_sdram_p0_pin_assignments.tcl** file as seen below in the **soc_system** > **synthesis** > **submodules** folder.



____ 5. Click **Run**.

This Tcl script creates I/O assignments for the DDR3 pins. If you like, open the Tcl scripts and examine them. It takes approximately 30 seconds to run.

____ 6. After the Tcl script successfully executes, click **OK**.

____ 7. Close the Tcl scripts window.

____ 8. Start compilation in the Quartus II software by selecting **Processing** > **Start Compilation**.

This compile takes approximately 10 minutes.

____ 9. When compilation completes, click **OK**.

Step 7: Examine Output Files

____ 1. Using Windows Explorer, navigate to **<project_folder>\Labs**

The soc_system.sof file generated will be used in later labs to program the FPGA.

Using Windows Explorer, navigate to the software handoff file directory:

<project_folder>*Labs\hps_isw_handoff\soc_system_hps_0*

Here you'll find the handoff files generated by the tools that software will need.

Name	Date modified	Type	Size
alt_types.h	4/8/2013 3:56 PM	C/C++ Header	4 KB
emif.xml	4/8/2013 3:56 PM	XML Document	9 KB
hps.xml	4/8/2013 3:56 PM	XML Document	6 KB
id	4/8/2013 3:56 PM	File	1 KB
sdram_io.h	4/8/2013 3:56 PM	C/C++ Header	2 KB
sequencer.c	4/8/2013 3:56 PM	C Source	312 KB
sequencer.h	4/8/2013 3:56 PM	C/C++ Header	21 KB
sequencer_auto.h	4/8/2013 3:56 PM	C/C++ Header	8 KB
sequencer_auto_ac_init.c	4/8/2013 3:56 PM	C Source	1 KB
sequencer_auto_inst_init.c	4/8/2013 3:56 PM	C Source	2 KB
sequencer_defines.h	4/8/2013 3:56 PM	C/C++ Header	4 KB
soc_system_hps_0.hiof	4/8/2013 3:56 PM	HIOF File	3 KB
system.h	4/8/2013 3:56 PM	C/C++ Header	1 KB
tclrpt.c	4/8/2013 3:56 PM	C Source	32 KB
tclrpt.h	4/8/2013 3:56 PM	C/C++ Header	17 KB

Exercise Summary

- Instantiated additional components and connect them to the HPS
- Generated the Qsys System
- Examined the output files generated by the process

END OF EXERCISE 2

Exercise 3A

Exercise the FPGA Using the System Console Tool

Objectives:

- *Program the board using System Console*
- *Use System Console to verify JTAG signal integrity, system clock and reset functionality*
- *Perform simple master read and write operations from System Console (switches and LED PIO)*
- *See that status of switches button and LEDs using the Dashboard*

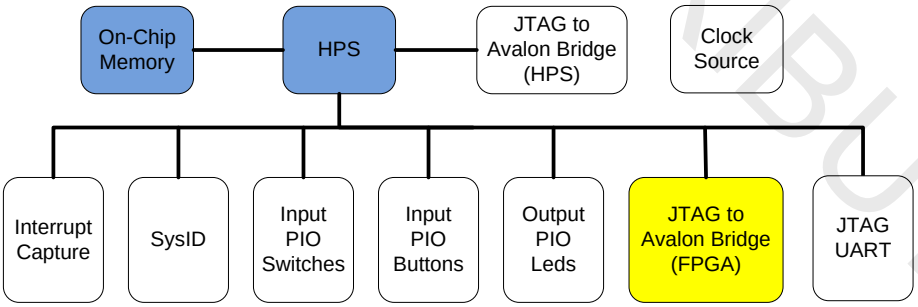
Introduction:

An ARM processor located inside the SoC behaves the same as any other ARM processor. From the FPGA designer's point of view, FPGA is independent of the processor.

This lab is intended to prove that your IP (in this case the LED and switch PIO IP) is connected correctly and responds to Avalon™ bus transactions.

In this exercise, you will use the System Console to control the system. You can communicate with the Avalon slave interface from System Console and act as the master to the system through the JTAG- to-Avalon Master Bridge (FPGA) that is part of your system. This block is shaded on the bottom row of the diagram below. You will then use the System Console to check the system reset and clock signals. After that, you will perform simple master reads and writes from the System Console. Finally, you will use the Dashboard GUI to control the system from the System Console.

While the HPS core has been instantiated in the Qsys system, it's not running because we haven't provided any software.



Step 1: Connect the Development Board

____ 1. Take the DE-1 development board out of the box and connect the power supply

____ 2. Connect the USB to Ethernet dongle into the laptop. No ethernet cable is needed since this is only used for the ARM DS-5 tool licensing.

____ 3. Connect the USB cable between the host PC and the USB-Blaster™ II port on the board.

____ 4. The MSEL DIP switches on the back of the board should already be set as the following. Verify the settings.

SW10 (MSEL0..4)	ON-OFF-ON-OFF-ON-OFF (ON is 0)

____ 5. If the the Micro SD card is inserted in the slot J11 then remove it.

____ 6. Turn on the development board using the power switch.

Step 2: Launch System Console and Program the FPGA

System Console is a tool that allows low level access to the memory mapped peripherals in the system over JTAG. We will be using this tool to perform low level board bring up and testing.

____ 1. In the Quartus II software, make sure the **soc_system** project is open.

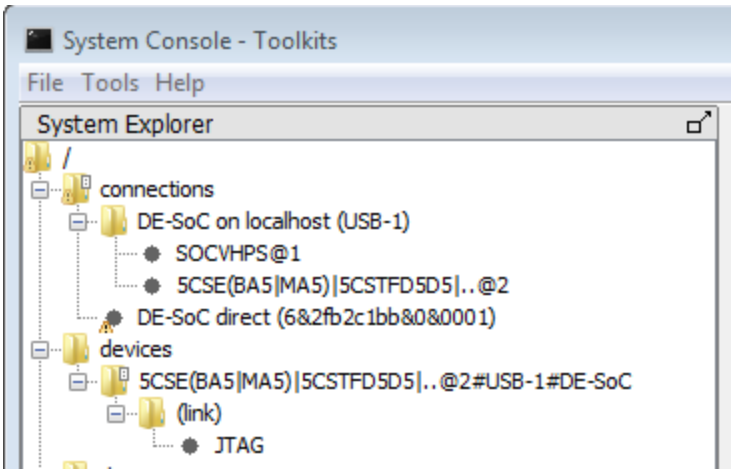
____ 2. In the Qsys tool, open the **soc_system.qsys** file if it's not already open

We will use this to verify the addresses.

____ 3. Launch the System Console tool by selecting **System Console** from the Qsys or Quartus **Tools** menu.

____ 4. Verify that System Console can see the connections in the JTAG chain in the System Console **System Explorer** window.

*If you don't see the connections and device shown in the picture below, go to the System Console **Tools** menu, and select **Refresh Connections**. If this still doesn't work, let the instructor know.*



____ 5. View the device services available by typing **get_service_paths device** in the Tcl Console.

System console will respond with a list of the paths to all of the devices found in the chain. In our case we have only the one as shown below:

```
% get_service_paths device
/devices/5CSE(BA5|MA5)|5CSTFD5D5|..@2#USB-1#DE-SoC
```

____ 6. Program the board in the System Console tool using the following steps:

a. **set d_path [get_service_paths device]**

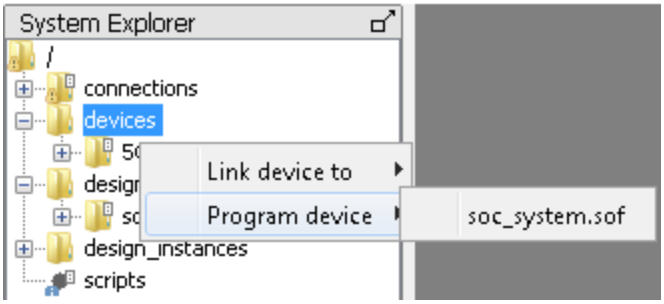
This command sets the device service path to d_path so we can easily access it. "set" is a Tcl command assigns values to a variable.

b. **device_download_sof $d_path soc_system.sof**

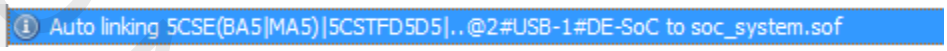
This command programs the device from the previous step with the soc_system.sof file. The device service is unusual in that it doesn't require being opened or closed before and after use.

```
% set d_path [get_service_paths device]
/devices/5CSE(BA5|MA5)|5CSTFD5D5|..@2#USB-1#DE-SoC
% device_download_sof $d_path soc_system.sof
```

It is also possible to program the device by right clicking on the device, selecting Program device submenu and selecting soc_system.sof

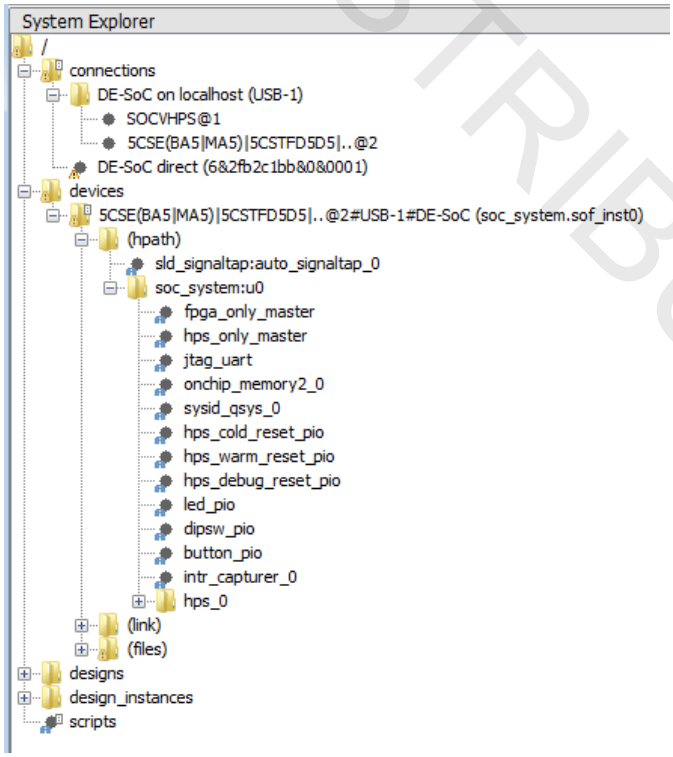


____ 7. Confirm that System Console has programmed the FPGA and linked to the project by verifying that the following message appears in the messages window.



Expand the devices folder in the System Explorer window and then the subfolders to see all of the components that have been connected to the *JTAG to Avalon components in the Qsys system and the now linked soc_system.sof file. Notice all of the compoenent names match those we specified in Qsys, this is possible because we opened System Console from our open Quartus project and all of the information is automatically transferred to the System Console session.*

If you do not see this, please consult your instructor.



Step 3: Verify Clock and Reset From the System Console Tool

____ 1. View the JTAG debug services available by typing **get_service_paths jtag_debug** in the Tcl Console pane

System Console responds by listing the two possible devices available to perform jtag_debug tasks. These are the JTAG to Avalon Bridges that were added and connected to the HPS component (hps_only_master) and the rest of the FPGA components in the Qsys system (fpga_only_master).

```
% get_service_paths jtag_debug
/devices/5CSE(BA5|MA5)|5CSTFD5D5|..@2#USB-1#DE-SoC/(link)/JTAG/alt_sld_fab_sldfabric.node_1/phy_0
/devices/5CSE(BA5|MA5)|5CSTFD5D5|..@2#USB-1#DE-SoC/(link)/JTAG/alt_sld_fab_sldfabric.node_2/phy_1
```

____ 2. Create a variable that points to the fpga_only_master JTAG to Avalon bridge component by typing **set jd_path [lindex [get_service_paths jtag_debug] 0]**

As we saw in the previous step there are two JTAG debug paths available, so we can access each one independantly by creating a variable to point to them, as we did with the device service earlier. In this case we will create a variable called jd_path to point to the fpga_only_master which was the first in the list. For this JTAG debug purposes we can use either JTAG to Avalon master.

```
% set jd_path [lindex [get_service_paths jtag_debug] 0]
/devices/5CSE(BA5|MA5)|5CSTFD5D5|..@2#USB-1#DE-SoC/(link)/JTAG/alt_sld_fab_sldfabric.node_1/phy_0
%
```

____ 3. Verify that the signal integrity of the JTAG chain by passing a test pattern through it and making sure you get the same values back from the JTAG chain. Type**:**

jtag_debug_loop $jd_path [list 1 2 3 4 5 6 7 8 9 10] in the System Console Tcl Console window. *This is a useful sanity check to make sure the JTAG chain is passing through the bits expected and not corrupting them. This can be useful for large JTAG chains.*

```
% jtag_debug_loop $jd_path [list 1 2 3 4 5 6 7 8 9 10]
0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a
```

____ 4. Verify that the clock is toggling by entering the following commands in the System Console Tcl Console window.

a. **jtag_debug_sense_clock $jd_path**

This command simply senses if the clock has ever toggled, returning a 1 if it has.

b. **jtag_debug_sample_clock $jd_path**

This command asynchronously samples the clock signals. You will likely have to run this command several times to witness a change.

Note: Pressing the up arrow will bring up the previous System Console command and tab will bring up a list of commands that can be entered. When tab completing the command, use the up and down arrows to select the command and then press enter to make the selection.

```
% jtag_debug_sense_clock $jd_path
1
% jtag_debug_sample_clock $jd_path
0
% jtag_debug_sample_clock $jd_path
0
% jtag_debug_sample_clock $jd_path
0
% jtag_debug_sample_clock $jd_path
1
```

____ 5. Verify that the reset signal has been released by typing **jtag_debug_sample_reset $jd_path** command in the Tcl Console window to sample the current value of the reset signal.

The reset signal is active low so we should expect the result of the sampling to be 1 denoting the reset is released. We could also issue a reset to all the components whose reset line is connected to the JTAG to Avalon Master component in Qsys by typing ***jtag_debug_reset_system $jd_path***

```
% jtag_debug_sample_reset $jd_path
1
% jtag_debug_reset_system $jd_path
```


Step 4: Perform Master Reads and Writes to Peripherals in the FPGA

____ 1. View the Avalon master services available in the FPGA by entering the following command **get_service_paths master**

This command returns all the possible masters on the JTAG chain. Notice that again the paths available are the hps_only_master and fpga_only_master JTAG to Avalon bridges.

```
% get_service_paths master
/devices/5CSE(BA5|MA5)|5CSTFD5D5|..@2#USB-1#DE-SoC/(link)/JTAG/alt_sld_fab_sldfabric.node_1/phy_0/fpga_only_master.master
/devices/5CSE(BA5|MA5)|5CSTFD5D5|..@2#USB-1#DE-SoC/(link)/JTAG/alt_sld_fab_sldfabric.node_2/phy_1/hps_only_master.master
```

____ 2. Create a variable that points to the fpga_only_master JTAG to Avalon bridge component by typing **set m_path [lindex [get_service_paths master] 0]**

This command sets a variable, m_path, to the master service path. We use the index 0 here because you can see from the master services listed in the previous step that the FPGA only master is the first one listed and therefore index 0.

```
% set m_path [lindex [get_service_paths master] 0]
/devices/5CSE(BA5|MA5)|5CSTFD5D5|..@2#USB-1#DE-SoC/(link)/JTAG/alt_sld_fab_sldfabric.node_1/phy_0/fpga_only_master.master
```

____ 3. Claim the master service to allow exclusive access to the master device by typing

set c_path [claim_service master $m_path “”] in the Tcl console window.

```
% set c_path [claim_service master $m_path ""]
/channels/local/(lib)/master_1
%
```

____ 4. Change the state of the four FPGA LEDs using the **master_write** command as shown below:

a. **master_write_32 $c_path 0x10040 0x3ff** will turn on all the LEDs.

b. **master_write_32 $c_path 0x10040 0x0** will turn off all the LEDs.

c. **master_write_32 $c_path 0x10040 0xc** will turn on LEDs 2 & 3.

d. **master_write_32 $c_path 0x10040 0xa** will turn on LEDs 0 & 2.

*Remember that the **led_pio** is located at address 0x10040 in the Qsys system and are “on” when driven with a ‘1’. Feel free to play with different values.*

*Because the **led_pio** component is 10 bits wide, if you used master_write_8 or master_write_memory you’ll need to write to multiple words.*

____ 5. Change the dip switch settings of the FPGA side of SW0-9 and read their value with the **master_read_32** command of the **dip_sw_pio** located at address **0x10080** using the command **master_read_32 $c_path 0x10080 1**, you may also try out various different read widths and see their effects.

```
% master_read_32 $c_path 0x10080 1
0x00000181
% master_read_16 $c_path 0x10080 1
0x0181
% master_read_8 $c_path 0x10080 1
0x81
```

Notice that read display size matches the command size.

____ 6. Hold down one or more of the buttons and read the **button_pio** component at address **0x100C0** with a master_read command, **master_read_32 $c_path 0x100C0 1**

Note: These signal are active low. And feel free to try out different combinations.

```
% master_read_8 $c_path 0x100C0 1
0x0b
% master_read_16 $c_path 0x100C0 1
0x000c
% master_read_32 $c_path 0x100C0 1
0x0000000f
```

____ 7. Close the master service by typing **close_service master $c_path**

Step 5: Run the System from the System Console Dashboard GUI

____ 1. From the Quartus II software **File** menu, open **SoC_HW_SysCon.tcl** found in the project directory and examine it.

You may need to change the file type to script files in the dialog box. The script contain the dashboard commands that sets up the dashboard GUI and function calls that respond to button pushes..

____ 2. In the System Console tool, run the **SoC_HW_SysCon.tcl** by typing

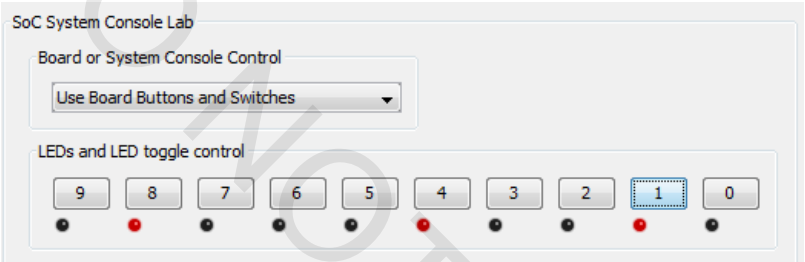
source SoC_HW_SysCon.tcl command in the Tcl console window

Note: Remember that Tcl is case sensitive. You can also run scripts from the File menu.

```
% source SoC_HW_SysCon.tcl
```

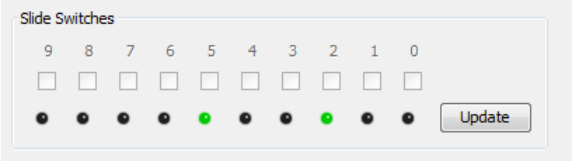
____ 3. Push the **LED Toggle** buttons on the Dashboard and confirm that the corresponding FPGA LED changes state on the development board.

Everytime you press the buttons, the script sends a write command to the LED PIO register and also updates the display in System Console.



____ 4. Flip some of the 10 switches, and push the update button. Verify that the corresponding Switch Setting LED changes on the dashboard window.

Everytime UPDATE is pressed, the script performs a master read on the switch PIO component and updates the display.



This script has many more features if you're curious what it does, please ask the instructor

____ 5. Exit System Console

Exercise Summary

- Performed low level verification of the jtag chain and programmed the FPGA
- Performed low level verification of the system clock and reset signals
- Performed master read and write commands using the System Console
- Ran the system hardware from the Dashboard GUI and observed the results

END OF EXERCISE 3A

Exercise 3B

Debugging Hardware and Software Using SignalTap II Logic Analyzer and ARM DS-5 Development Studio

Objectives:

- *Use the DS-5 tool to write from the HPS to the LED PIO and cause the SignalTap™ II logic analyzer to trigger on that transaction*
- *Use a break point in the software running on the HPS to trigger the SignalTap II logic analyzer in the FPGA*

Introduction

In lab 3A, we demonstrated that instantiating an HPS component in a Qsys system has no effect on the normal debug process of an FPGA design. The HPS component had no effect on the system because the there was no software running and the bridges to/from the FPGA weren't enabled.

In order to prove the connectivity between the FPGA design and the HPS component, this lab will show you how to access the HPS from the FPGA design a number of different ways. The HPS to FPGA bridge connectivity will be proven by initiating a register write to the LED PIO device located in the FPGA and triggering on that transaction using the SignalTap II logic analyzer. The SignalTap II logic analyzer can also be triggered from a break point in software on the HPS.

Step 1: Launch the Software Project

We need to initiate the bridges on the HPS component and the easiest ways to do that is to use the DS-5 tool to load the preloader software onto the HPS component and get the HPS component to initialize the bridges for us.

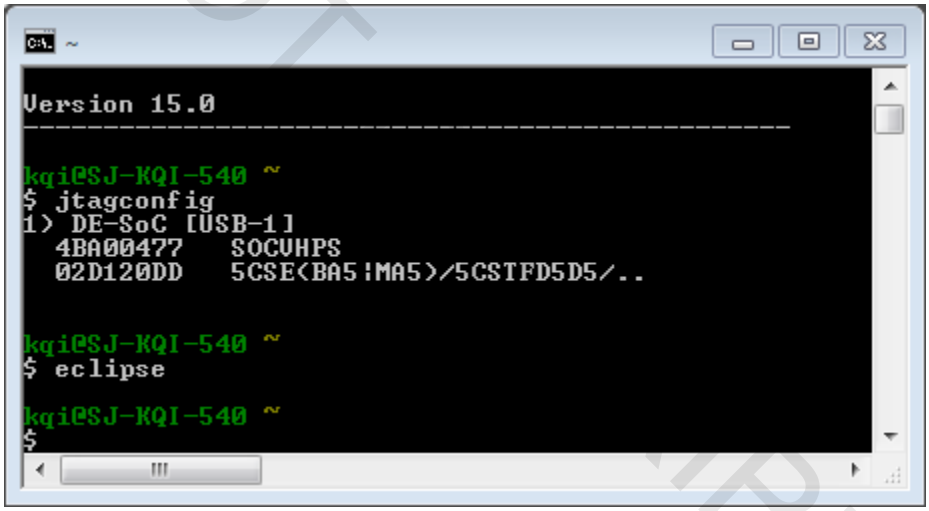
____ 1. If you powered off your board, reprogram the FPGA with the Quartus II programmer tool using the .sof file from the project directory.

____ 2. In the Windows Explorer, navigate to the **C:\altera\15.0\embedded** directory and double-click the file **Embedded_Command_Shell.bat** to run it.

This is a Cygwin shell that allows you to perform many SoC related tasks.

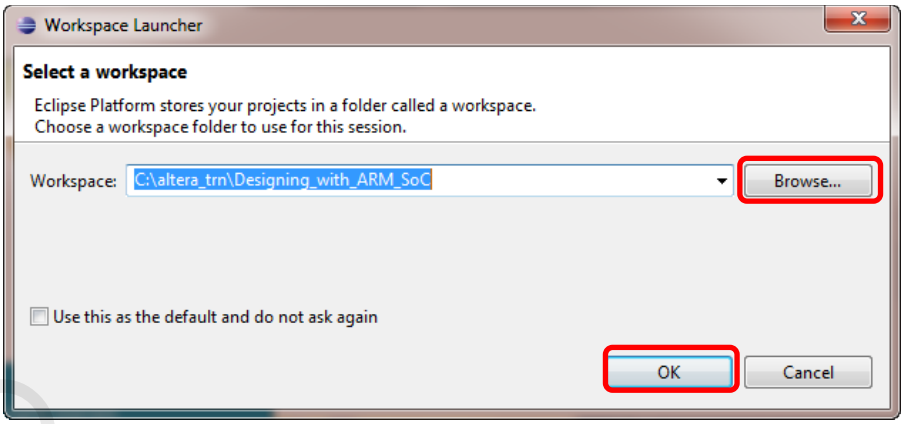
____ 3. Type ***jtagconfig*** at the command prompt to scan and see the devices on the JTAG chain.

This will ensure the HPS and the FPGA can be seen in the JTAG chain.



____ 4. Open the **Eclipse for DS-5** software by typing ***eclipse&*** in the Embedded Command Shell.
Launching DS-5 from the shell bring in various SoC related settings.

____ 5. When the **Workspace Launcher** window appears, CAREFULLY select the workspace as **C:\altera_trn\Designing_with_ARM_SoC** (Not Software)



Make sure you're using the Designing_with_ARM_SoC workspace and ***NOT*** *Developing_Software_For_ARM_SoC*

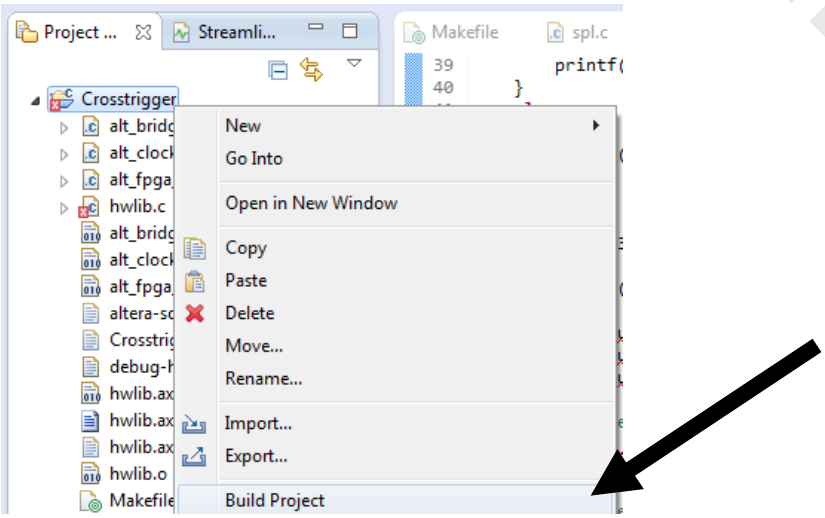
____ 6. Click the **OK** button to accept the workspace.

Make sure the Crosstrigger project is loaded into the workspace

____ 7. Look over the software if you wish, in the Crosstrigger project, the main source code is located in hwlib.c (open that by double clicking on it within the Crosstrigger Project)

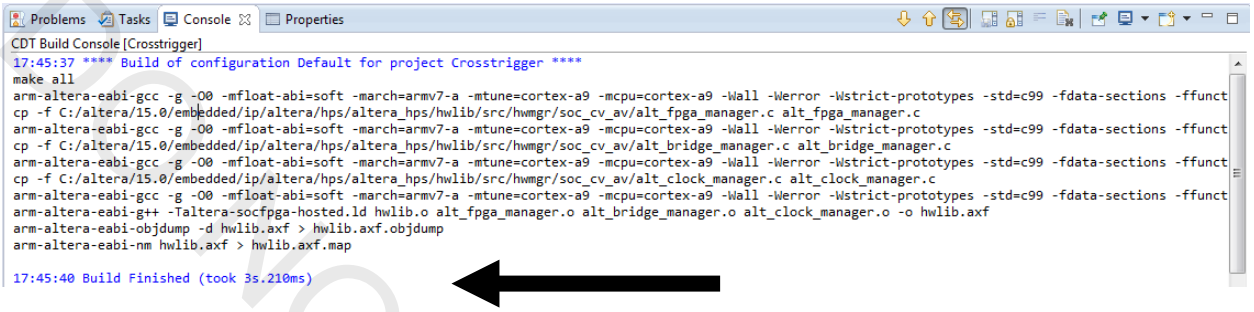
This is a very simple program, and most of the action is performed with in the test_bridge function where it will first initialize the FPGA to HPS, the HPS to FPGA, and the lightweight HPS to FPGA bridges. Then the software will run a gray code pattern on the LEDs by writing across the lightweight HPS to FPGA bridge to the LED PIO component.

____ 8. Compile the program by right clicking the Crosstrigger project and choose Build Project

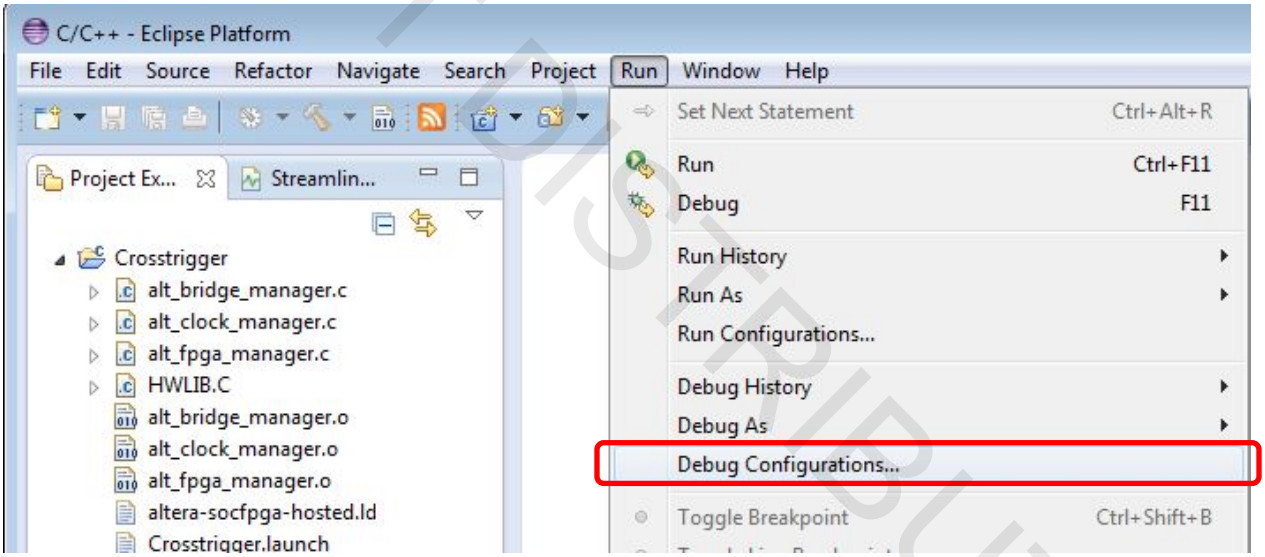


____ 9. Verify that the project has been successfully compiled by looking in the Console window.

Make sure the build finished without any errors and that the hwlib.axf executable linkable file is generated correctly. Don't worry about any errors in the Problems pane or the source code, these errors are caused by the Eclipse GUI indexer not having a chance to run on all the files that's copied into the directory during build..



____ 10. From the **Run** menu select **Debug Configurations…**



The debug configuration is a way to create a launcher script that specifies a variety of setup scripts to speed up the connecting to the HPS and performing basic debug functions.

____ 11. Make sure the **Crosstrigger** configuration launcher script is selected in the list on the left and the **Connection** tab is selected.

____ 12. In the **Select target** window, choose **Debug Cortex-A9_0** under **Altera → Cyclone V SoC (Dual Core) → Bare Meta Debug** (see diagram below)

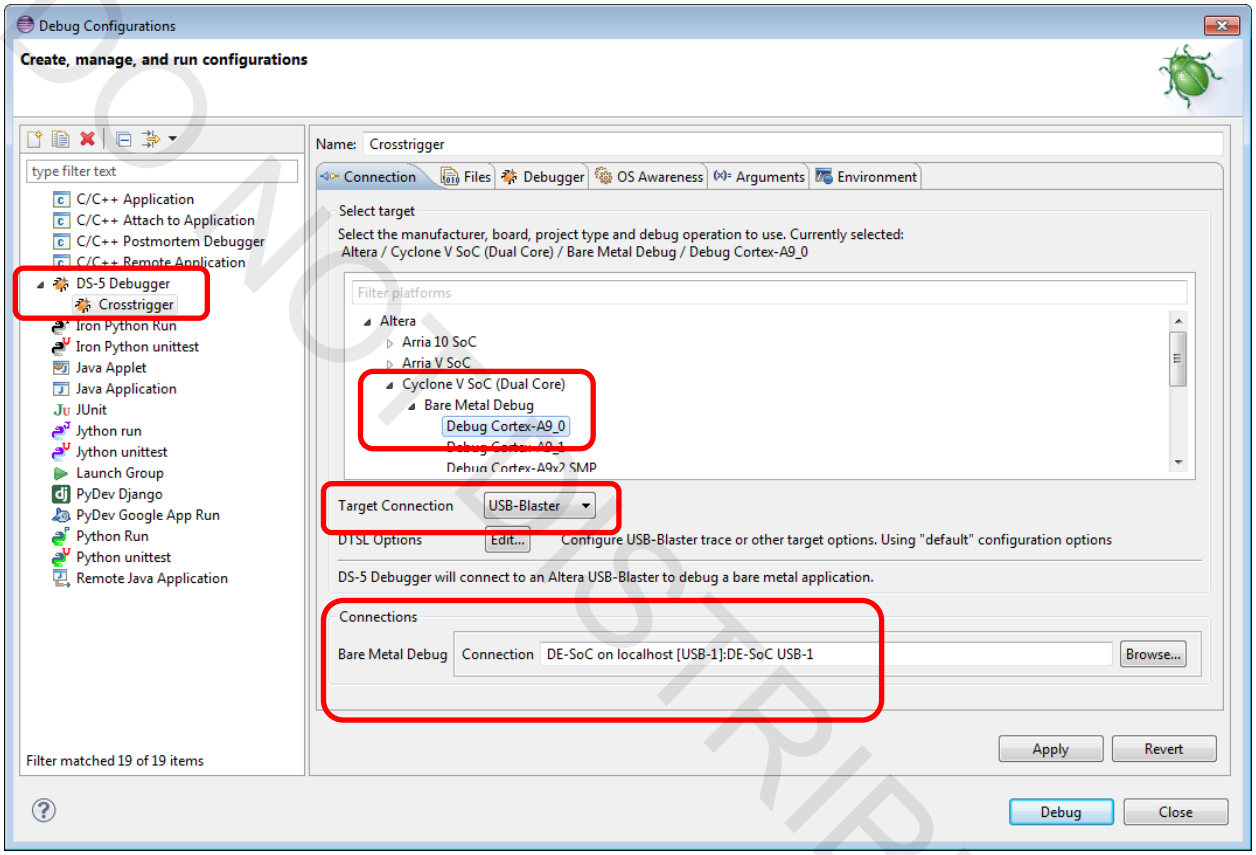
This specification allows us to run software on the ARM processor on the board.

____ 13. For **Target Connection**, choose **USB-Blaster** from the pulldown menu

This step tells the configuration which debug cable to use. In addition to the ARM debug tools such as DSTREAM, DS5 also recognizes Altera's USB-Blaster as a valid JTAG debug cable to gain access to the Cortex A9 CPU.

____ 14. For **Connection -> Bare Metal Debug**, click **Browse…**, and select **DE-SoC on localhost**

This steps looks for the valid components across the USB-Blaster II download cable.

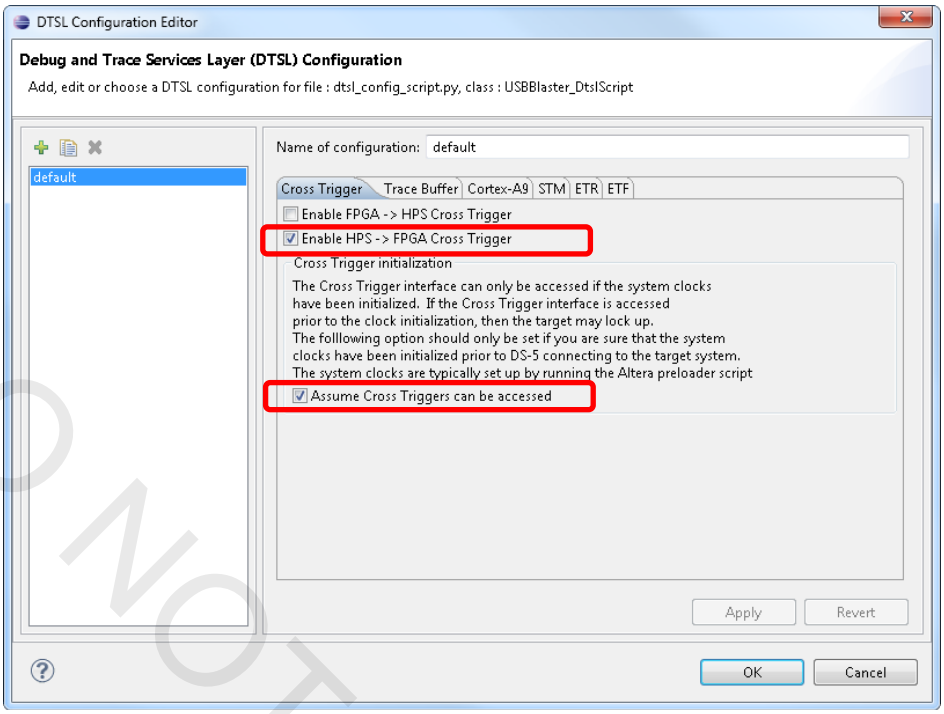


____ 15. Click the **Edit** button to modify the **DSTL Options**.

____ 16. On the **DSTL Configuration Editor** window, click the **Cross Trigger** tab.

____ 17. Click **Enable HPS -> FPGA Cross Trigger** to enable it.

____ 18. Click **Assume Cross Triggers** can be accessed to enable it.



This setting is required in order to enable the cross-triggering from the HPS component to the FPGA domain. This causes a trigger to the FGPA in the event of a break in the software code running on the HPS.

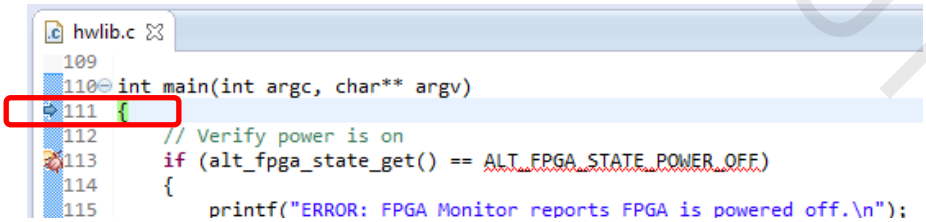
____ 19. Press **OK** to close the **DSTL Configuration Editor** window.

____ 20. Click **Debug** to start the debug process

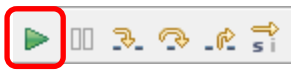
____ 21. Click **Yes** if prompted to switch to the debug perspective.

The start-up process will take a second and it will stop at main (line 111 of file hwlib.c)

If you encounter errors, please see the troubleshoot guide at the end of this document.



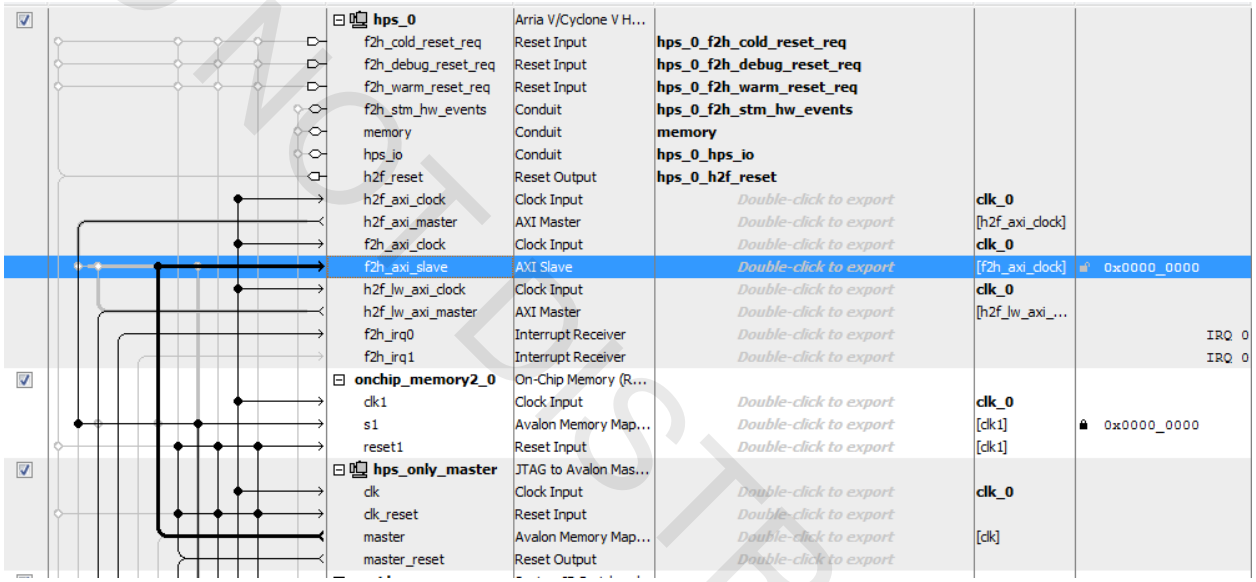
____ 22. Press the play button, ▶, in the DS-5 for Eclipse GUI to continue running code until it stops at a breakpoint on line 51. *This breakpoint has already been set for you.*



If you look at the the App Console located in the lower right hand corner of the DS-5 for Eclipse window you can see that the bridges between the HPS and the FPGA have been initiated successfully. It is now possible to communicate between the HPS and the FPGA using the Avalon bus.

```
INFO: alt_bridge_init(ALT_BRIDGE_F2H) successful.
INFO: alt_bridge_init(ALT_BRIDGE_H2F) successful.
INFO: alt_bridge_init(ALT_BRIDGE_LWH2F) successful.
```

____ 23. In the Qsys tool, open it if it's been closed, click on the **f2h_axi_slave** port of the hps_0 component to highlight it.



You can see that the hps_only_master connects to the HPS component through the FPGA to HPS bridge. This connection allows access the to the HPS registers from System Console through the JTAG to Avalon Master Bridge.

____ 24. Click on **the h2f_axi_master** and the **h2f_lw_axi_master**, these are the connections that will allow the HPS to control FPGA component. In this lab, the softwrae will mostly exercise the lightweight bridge to write to the LED PIO component.

Step 2: Have HPS trigger a SignalTap II logic analyzer capture of HW state

____ 1. Open the SignalTap II logic analyzer file using the following steps

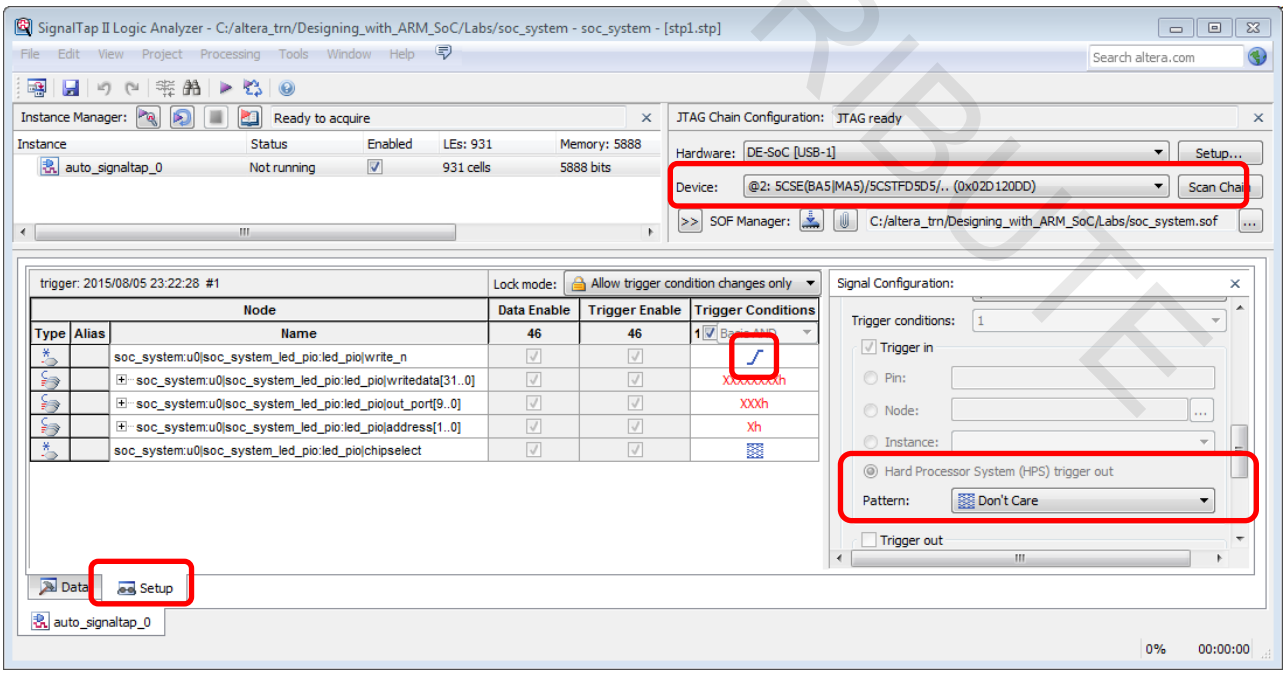
- In the Quartus II software, select **File-> <u>Open.</u>**.
- Select **SignalTap II Logic Analyzer Files (.stp)** in the "**Files of type:**" drop down menu.
- Select the **stp1.stp** file from the **<project_folder>\Labs** directory.
- Click **Open.**

SignalTap II is Altera's Embedded Logic Analyzer used to debug FPGA logic in real time. For more information, consult the Quartus II Handbook or view the free SignalTap II online training available at http://www.altera.com/training

____ 2. Make sure the **Setup** tab is in the foreground to see how the triggers have been configured.

The SignalTap II logic analyzer file taps signals of the Avalon Memory-Mapped interface that is connected to the LED PIO block. The signal write_n rises when a transaction ends and the SignalTap II file is setup to trigger on that event.

The SignalTap II logic analyzer also has the ability to trigger on an event from the HPS. The "trigger in" option in the SignalTap II Signal Configuration window, is currently set to HPS trigger out. Note that this input is currently set to "don't care". That will change in a later section about FPGA to HPS cross triggering.



____ 3. Ensure that the 5CS device is selected under the **Device:** drop down menu.

SignalTap works with logic in the FPGA and not the HPS component directly.

____ 4. Set the lock mode of the SignalTap II analyzer to "Allow trigger condition changes only".

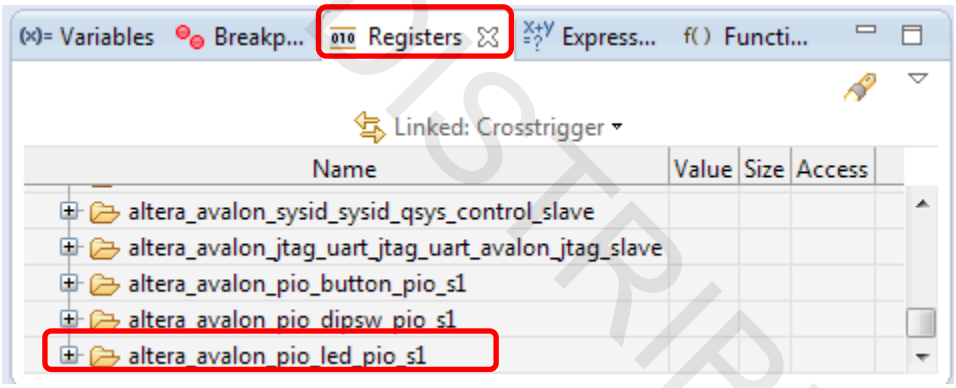
This option is in the middle of the window and will prevent us from making changes to the setup that will require a Quartus recompile.

____ 5. Arm the SignalTap II logic analyzer by clicking the **Run Analysis** button  .

____ 6. In the DS-5 tool, go to the **Registers** tab and scroll to the bottom.

If you do not see a registers tab, you can enable it from the DS-5 Windows menu → Show View → Registers

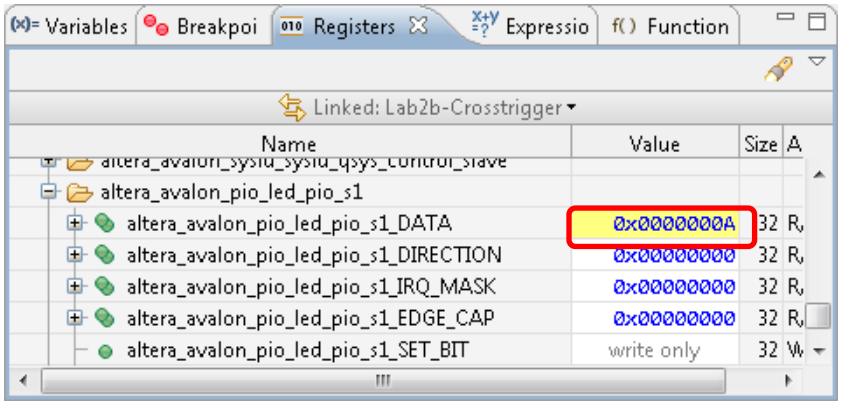
____ 7. Expand the Peripherals folder and find the LED PIO



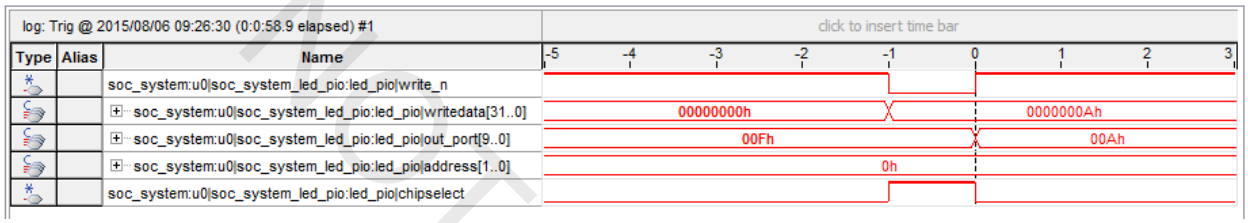
____ 8. Expand the **altera_avalon_led_pio_s1** register set by clicking on the "+" symbol next to it.

____ 9. Change the value of the **DATA** register to **0xA** by clicking in the **Value** column, editing the value and hitting enter.

The LEDR3 and LEDR1 on the board should turn on and SignalTap II should trigger.



____ 10. Examine the waveform in the SignalTap II logic analyzer **Data** tab

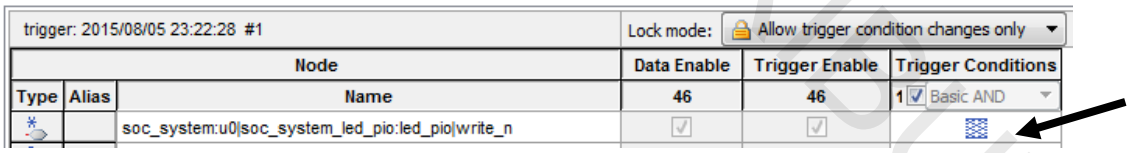


This shows the Avalon MM write transaction from the HPS to the LED PIO in the FPGA.

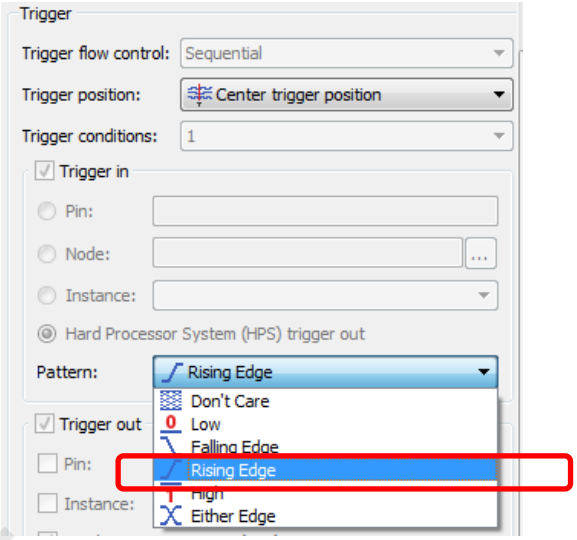
____ 11. Minimize the led_pio_s1 register in the register view of the DS5 tools.

____ 12. Click the **Setup** tab in the SignalTap II window .

____ 13. Change the Trigger Conditions on the write_n signal to be don't care by right-clicking the Trigger conditions column of the write_n signal.



____ 14. In the Signal Configuration pane of the setup tab, scroll down to just above the **Trigger out** section change the HPS Trigger out to "**Rising Edge**" using the drop down menu.



Now instead of triggering on the write signal, SignalTap will trigger when the Cortex-A9 processor stops. Had we left the write_n trigger condition in, then we would've gotten a 2 staged trigger condition where the write must happen after the Processor break.

____ 25. Arm SignalTap II logic analyzer by pushing the **Run Analysis** button  .

____ 26. Press the play button in the DS-5 tool to continue to the next break point (line 70). *SignalTap should trigger*

____ 27. Examine the waveform in the **Data** tab in the SignalTap II logic analyzer tool.

Notice all signals are stable at the trigger (Time 0), this is because this trigger was caused by a software breakpoint and not any hardware trigger conditions.

____ 28. Examine the LEDs on the board and write down state of each LED.

The value in SignalTap may not agree with the value shown in the System Console because at the time of SignalTap capture, the LEDs wasn't written to yet.

LEDR[9..0] _____

____ 29. Arm SignalTap II logic analyzer again.

____ 30. Press the play button again in the DS-5 tool to repeat the loop and stop at the breakpoint again. SignalTap tool should trigger again.

____ 31. Examine the waveform again.

Now SignalTap out_port should match the previous LED value

____ 32. What is the state of the LEDs now on the Board?

_____

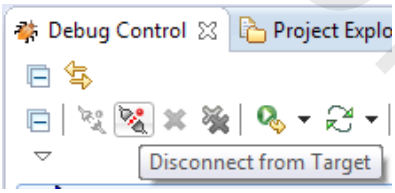
____ 33. Repeat Arming the SignalTap trigger and runinng the software a few more times to see more gray code values.

____ 34. Remove the breakpoint on line 70 by double clicking the left most column where the red dot is indicating the breakpoint.

____ 35. Press the play button to finish running the software.

You should see "RESULT: All tests successful in the App Console"

____ 36. Disconnect the DS5 from the Target. By pressing in debug controls.



____ 37. Exit from DS-5, Embedded Command Shell, Qsys, SignalTap II, and Quartus tools.

In this section, you were able to use the trigger output from the HPS as a trigger input to capture exactly the transaction of interest in SignalTap II logic analyzer.

Exercise Summary

- Triggered SignalTap II analyzer from a transaction generated by the HPS using the DS-5 tool
- Triggered the SignalTap II logic analyzer using a break point set in software.

END OF EXERCISE 3B

