Implement a PCIe endpoint using Qsys System Integration Tool
Agenda

• What is Qsys?
• Qsys UI
• Using Qsys in FPGA design flow
• Qsys files
• Qsys Intellectual Property (IP)
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
Introducing Qsys
Automatic Interconnect Generation

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

• Simplifies complex system development
  – Automatic interconnect generation
• Raises the level of design abstraction
  – High level design and system visualization
• Provides a standard platform: IP integration, custom IP authoring, IP verification
• Enables design re-use
• Scales easily to meet the needs of end product
• Reduces time to market
  – Reduces design development time
  – Eases verification
Easy-to-Use System Integration UI

Library of Available IP
- Interface protocols
- Memory
- DSP
- Embedded
- Bridges
- PLL
- Custom systems

Connect IP and Systems
- IP 1
- Custom 1
- IP 2
- IP 3
- Custom 2

Accelerate Development

Simplify Integration

HDL

Automate Error-Prone Integration Tasks
Design Re-Use

- Qsys enables re-use of IP and systems with IP management capabilities
Target Qsys Applications

- Qsys can be used in almost *every FPGA design*
- Designs fall into two categories
  - Control plane
    - Memory mapped
    - Reading and writing to control and status registers
  - Data plane
    - Streaming
    - Data switching (muxing, demuxing), aggregation, bridges
- Applications include video and image processing, high-speed interfaces, embedded and memory
Processor Required?

*What if your design does not require a processor?*

- Qsys systems do not **require** a processor
- Other components can initiate transfer requests and exert system control
  - Memory-mapped components use **master \rightarrow slave**
    - e.g. state machine, direct memory access (DMA)
  - Streaming data uses **source \rightarrow sink**
    - e.g. video camera, pattern generator
- Soft Processors (e.g. Nios® II) or external processors may be connected to the Qsys Interconnect
Qsys vs. SOPC Builder – Similarities

- IP integration with switch fabric connectivity
- Dynamic system generation
- High level system visualization
- Custom IP authoring
- IP verification and bus functional models (BFMs)
- Simulation support for ModelSim
- Real-time system debug
In addition to SOPC Builder capabilities, Qsys adds:

- High-performance interconnect
- Hierarchy
- Industry standard interfaces
- IP management capabilities
Increasing the Level of Abstraction

Abstraction & Improved Productivity Level

Low

Medium

High

Design Block Integration

IP Integration
- Design a system with IPs
- IP re-use
- IP verification

System Integration
- Design a system with systems
- System re-use
- System verification

Schematic Entry Tool

SOPC Builder Tool

Qsys System Integration Tool
SOPC Builder Systems in Qsys

- Qsys can open SOPC Builder files
- Qsys will transform SOPC Builder system into Qsys system
  - .SOPC file maintained
  - Cleanup option available
Agenda

• What is Qsys?
• Qsys UI
  • Using Qsys in FPGA design flow
  • Qsys files
  • Qsys Intellectual Property (IP)
Qsys UI Overview

- Component Library
- System Contents
- System Inspector
- Address Map
- Clock Settings
- Project Settings
- Generation
- HDL Example
- Messages
Open Qsys from Quartus® II Software

- Quartus II software Tools Menu
- Select Qsys
- Open or create new Qsys system
- Can also open .qsys file from Quartus II File menu
Qsys System-Integration Tool
Component Library

- Lists available IP and systems
- Type search string to filter the list
- Reuse previous systems – hierarchy
- Expand categories to browse components
- Double-click component or click Add button to add selected component to system
Qsys UI: System Contents Tab

- Displays components and subsystems
- Use to add/remove/connect components

System Components

Enable/disable components
System Contents: Component Interfaces

- Number of independent interfaces with which component communicates with rest of system
- Clock and reset considered separate interfaces
System Contents: Connections Panel

- Use Connections panel to specify interface connectivity
  - Clocks, resets
  - Masters and slaves
  - Sources and sinks
  - Interrupt senders and receivers
  - Custom instruction senders and receivers

- Each dot represents a connection between two interfaces
  - Qsys generates system interconnect based on this configuration

- Direction shown with arrows

- Hide connections for added readability
  - Collapsing components
  - Using filters
System Contents: Exports

- Explicitly export interfaces
  - Choose interfaces to connect outside Qsys system
  - Any interface can be exported
- Managed in **Export** column

Exported interfaces displayed with “pin” icon
**System Contents: Clocks**

- **Clock Source** component
  - Defines input clock(s) to system
  - Connect two interfaces
    - **Clock Input** interface fed from outside system (export)
    - **Clock Output** interface connects to **Clock Input** interface of other system components
System Contents: Resets

1. Manually connect each reset interface
   – User has more control over reset implementation
     • Reset only subset of the components in system
     • Must use care to avoid reset loops and system lockup
       – MM Slave reset in middle of transaction, Master waits forever
   – Reset interfaces declared independent of clock interface
     • Still associated with a clock interface
     • Reset synchronized to associated clock
   – Multiple resets can enter the system, like clocks
     • Uses Reset Controller Block (later)

2. Choose **Create Global Reset Network** from **System** menu
   – Automatically connects all **Reset Inputs**
Reset Connection Points

External reset input interface

Reset Output interface drives Reset Input interfaces of other system components
System Contents: Conduit Interfaces

- Conduits are used for any non-standard interfaces (groups of signals)
  - User must manually specify signals that make up conduit interface
- Like-typed conduits can be connected in the UI
  - Same signals with opposite directions
- May be exported outside system
System Contents: Addressing

• Each memory-mapped master interface has own address map
  – When slave ports are shared, the address map converges
  – Maximum 32-bit address space (4GB) for each master interface

• Master address map is a collection of the following
  – Connected slave interface base addresses
  – Connected slave interface address spans (determines end address)
  – Lowest and highest slave addresses make up the address space of the master

• Manually assign slave addresses
  – Double-click or let Qsys auto-assign

Master interface

Master address space based on connected slaves
Qsys UI: Address Map Tab

- Table of memory-mapped addresses
- Double-click cell to manually edit slave addressing
- Supports per-master addressing for shared slaves
  - Single slave represented by different address ranges for different masters

Master interfaces represented by
Slave interfaces represented by rows
Qsys UI: Clock Settings Tab

- Use for further clock management
- Add new clocks to system
- Rename system clocks for readability
- Specify clock frequencies
  - Must also create proper TimeQuest SDC constraints

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>External</td>
<td>50.0</td>
</tr>
<tr>
<td>sys_clk</td>
<td>pll.c0</td>
<td>100.0</td>
</tr>
<tr>
<td>ssram_clk</td>
<td>pll.c1</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Double-click to change name or frequency (non-PLL)
Qsys UI: Project Settings Tab

- Control interconnect implementation

Handshake, FIFO, or Auto clock domain crossing logic

Pipelining in interconnect
Instance Parameters

- Define parameters and associated script that configures instantiations of current system
  - Discussed in the Advanced Qsys System Integration Tool Methodologies class

Used in hierarchical systems
Qsys UI: System Inspector Tab

- Review system and component details
  - System hierarchy
  - Top-level system connection
  - Component interfaces
  - Connections between components
  - Component details

- Edit component settings
System Inspector – System View

Top-level symbol diagram

System level settings
System Inspector – Exported Interface

All exported interfaces

Selected exported interface
System Inspector – Connections

Connection type

Selected connection properties

All interface connections
System Inspector – Submodules

* Some wizard settings can be edited
System Inspector – Submodule Interface

Components interfaces

Selected interface settings including timing waveforms

Selected interface type
Qsys UI: HDL Example Tab

- Creates Verilog or VHDL instantiation template
- Use to instantiate Qsys system as a submodule in design

```verilog
sm_transfer_system u0 {
    .reset_n   (connected-to-reset_n),   // clk_clk_in_reset
    .clk       (connected-to-clk),      // clk_clk_in
    .led_out_export (connected-to-led_out_export), // led_out
    .ext_mem_bus_tcm_begintransfer_n_out (connected-to-ext_mem_bus_tcm_begintransfer_n_out), // ext_mem_bus_tcm_begintransfer
    .ext_mem_bus_tcm_data_out     (connected-to-ext_mem_bus_tcm_data_out), // ext_mem_bus_tcm_data_out
    .ext_mem_bus_tcm_byteenable_n_out (connected-to-ext_mem_bus_tcm_byteenable_n_out), // ext_mem_bus_tcm_byteenable
    .ext_mem_bus_tcm_chipselect_n_out (connected-to-ext_mem_bus_tcm_chipselect_n_out), // ext_mem_bus_tcm_chipselect
    .ext_mem_bus_tcm_outputenable_n_out (connected-to-ext_mem_bus_tcm_outputenable_n_out), // ext_mem_bus_tcm_outputenable
    .ext_mem_bus_tcm_address_out    (connected-to-ext_mem_bus_tcm_address_out), // ext_mem_bus_tcm_address_out
    .ext_mem_bus_tcm_write_n_out   (connected-to-ext_mem_bus_tcm_write_n_out), // ext_mem_bus_tcm_write
    .ssram_clk_clk                 (connected-to-ssram_clk_clk), // ssram_clk_clock
    .start_pushbutton_export       (connected-to-start_pushbutton_export) // start_pushbutton
}    
```
Qsys UI: Generation Tab

- Choose what files to generate (Simulation, synthesis, or symbol)
- Choose **Output Directory Path** (default is subdirectory)
- Click **Generate** button (bottom of tab)
Qsys UI: Messages

- Documents system error, warning and information messages
- Includes summary of error and warning counts

<table>
<thead>
<tr>
<th>Messages</th>
<th>Description</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>•</td>
<td>dma_srammm_to_led.control_port_slave (0x20..0x3f) overlaps pll pll_slave (0x20..0x2f)</td>
<td>System.avalon-master.avalon_master</td>
</tr>
<tr>
<td>1 Warning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>•</td>
<td>This memory is not initialized during device programming.</td>
<td>System.test_mem</td>
</tr>
<tr>
<td>4 Info Messages</td>
<td></td>
<td></td>
</tr>
<tr>
<td>•</td>
<td>User is required to provide memory initialization files for memory.</td>
<td>System.source</td>
</tr>
<tr>
<td>•</td>
<td>Memory will be initialized from C:\Designs\my_test_designs\qsys\qsys_no_nios_ver_sramsource2.hex</td>
<td>System.source</td>
</tr>
<tr>
<td>•</td>
<td>For Avalon-ST and Avalon-MM combinations, data width is 32 bits.</td>
<td>System.led_fifo</td>
</tr>
<tr>
<td>•</td>
<td>PIG inputs are not hardwired in test bench. Undefined values will be read from PIG inputs during simulation.</td>
<td>System.start_pushbutton</td>
</tr>
</tbody>
</table>

1 Error, 1 Warning
Additional UI Features

- System editing
- Filtering
- Auto-assignments and insertions
System Editing

- Remove components from system
  - Highlight component and click X button
  - Highlight component and hit Delete key

- Change order of components in system
  - Arrange components in System Contents tab to make system and component connections more easily understandable
  - Does not change addressing or connections
Filtering

• Filter System Contents display based on
  – Component name
  – Interface type
  – Connection source/destination
• Default filter is all components and all interfaces except interrupts
• Select pre-defined filters by right-clicking on any component or interface in System Contents
• Create new custom filters by clicking on Filter button and defining filter criteria
Other Useful Qsys Commands

• Accessible from the **File** menu
  – **Refresh System**
    • Refreshes all IP and components to detect any changes

• Accessible from **System** menu
  – **Assign Base Addresses**
    • Automatically eliminates conflicts in slave addressing
  – **Create Global Reset Network**
  – **Insert Avalon®-ST Adapters**
    • Automatically inserts adapters between source and sinks to compensate for differences (e.g. timing, data width, data format, error flags)
  – **Remove Dangling Connections**
    • Removes unconnected connection lines (interfaces) from active components in the System Contents tab
Agenda

• What is Qsys?
• Qsys UI
• Using Qsys in FPGA design flow
  • Qsys files
  • Qsys Intellectual Property (IP)
FPGA Hardware Design Flow

**Synthesis**
- Translate design into device specific primitives
- Optimization to meet required area & performance constraints
- Quartus II software or other supported synthesis tools

**Place & Route**
- Map primitives to specific locations inside
- Target technology with reference to area & performance constraints
- Specify routing resources to be used

**RTL Simulation**
- Functional Simulation (Modelsim, Quartus® II software)
- Verify Logic Model & Data Flow (No Timing Delays)
FPGA Hardware Design Flow

Timing Analysis
- Verify performance specifications were met
- TimeQuest static timing analysis

Gate Level Simulation
- Timing simulation
- Verify design will work in target technology

Test FPGA on PC Board
- Program & test device on board
- Use Quartus II tools (e.g. Signaltap® II logic analyzer) for debugging
Qsys System Generation

1. Create Qsys system
2. Generate HDL files for synthesis on Generation tab
   – Generates HDL files into a synthesis subdirectory
3. Set top-level project entity
   – Instantiate Qsys system as sub-block in another design file or specify Qsys system name as top-level design entity
4. Add generated .qip file to Quartus II project
   – Adds all required generated HDL files
5. Constrain design
6. Compile design

• Use Qsys to make changes to system, rather than editing any generated HDL
Agenda

• What is Qsys?
• Qsys UI
• Using Qsys in FPGA design flow
• Qsys files
  • Qsys Intellectual Property (IP)
Qsys Source Files

• .qsys file
  – Each .qsys file represents a single Qsys system (components, connections and parameterizations)
  – All other files for Qsys system created during system generation
  – Can use multiple Qsys files in a single Quartus II project (hierarchy)
Qsys Output Files

• Top-level folder files
  – <system_name>.sopcinfo
    • XML file describing Qsys system used for software development tools
  – <system_name>.bsf
    • Symbol file for Quartus II schematic editor (optional)
  – <system_name>.html
    • Generation report including output files generated, component list, etc.

• Files for synthesis
  – Located in <system_name>/synthesis folder
    • <system_name>.qip
      – Script file that adds all files needed for synthesis to Quartus II project
    • <system_name>.v
      – Qsys system top-level file connecting all components together (Verilog only)
  • Submodule files for synthesis
    – Located in submodule folder
    – Combination of Verilog, SystemVerilog and/or VHDL files representing system
    – .SDC timing constraint files may also be generated (add to Quartus II project)

• Files for simulation
  – Discussed later
Qsys Output Files Example

- **project (folder)**
  - sm_transfer_system.qsys
  - sm_transfer_system.sopcinfo
  - sm_transfer_system.bsf
  - sm_transfer_system.html
  - **sm_transfer_system (folder)**
    - synthesis (folder)
      - sm_transfer_system.qip
      - sm_transfer_system.v
    - **submodules (folder)**
      - component.v, .vhd and .sv files
      - component.sdc
Agenda

• What is Qsys?
• Qsys UI
• Using Qsys in FPGA design flow
• Qsys files
• Qsys Intellectual Property (IP)
Component Library Review

- Lists available IP and systems
  - Type search string to filter the list
  - Reuse previous systems – hierarchy
  - Expand categories to browse components
  - Double-click component or click Add button to add selected component to system
Component Library Parameter Editors

- After clicking **Add**, configure component options and interfaces before adding to system
IP Components

- Basic components
- Streaming components
- Memory components
- Tristate components
- Bridge components
- High-speed interface components
- Processor components
Basic IP Components

- Clock source
  - Defines external clock inputs to the system
- Reset bridge
  - Use when component-generated reset must be connected to internal logic and be exported
- Qsys does not allow interfaces to be exported and connected inside system
Streaming IP Components

- Use to manipulate data flows in streaming systems
- Demultiplexer
- Multiplexer
- Channel adapter
- Data format adapter
- Timing adapter
- Delay
- Splitter
Memory IP Components

• On-chip
  – On-chip RAM/ROM
  – On-chip FIFO

• Off-chip
  – SDRAM Controller
  – DDR/DDR2/DDR3 Controller
  – QDRII/QDRII+ SRAM Controller
  – RLDRAM II Controller
  – Flash Interfaces
Direct Memory Access (DMA)

• DMA
  – Perform bulk data transfers between Avalon-MM address ranges

• Scatter-gather DMA
  – Perform bulk data transfers and merges between non-contiguous memory and continuous address space
Tri-State IP Components

- Generic tri-state controller
  - Represents any controller block that uses tri-states
- Tri-state pin sharer
  - Allows multiple off-chip devices to share FPGA pins
- Tri-state bridge
  - Bridges unidirectional core signals and tri-state I/O signals
  - Must be used to connect to off-chip tri-state devices
Bridge IP Components

• **Avalon-MM clock crossing bridge**
  – Uses FIFOs for buffered high-throughput clock domain crossing from Avalon-MM master to slave

• **Avalon-MM pipeline bridge**
  – Specify pipelining of command and response segments
  – Control interface topology with or without pipelining
    • Allow exporting of multiple MM interfaces through one aggregate interface

• **JTAG to Avalon master bridge**
  – Uses JTAG commands to access and control Qsys components

• **SPI slave to Avalon master bridge**
  – Uses SPI interface to access and control Qsys components
High-Speed Interface IP

- IP_Compiler for PCI Express*
- 10/100/1000 Mb (Triple-Speed) Ethernet
- 10Gb Ethernet
- Interlaken
- RapidIO

* All except for IP_Compiler for PCI Express require additional licensing to use in finished product.
IP_Compiler for PCI Express®

• Configures Hard IP for PCI Express and embedded transceivers
  – Performs transaction, data link, PHYMAC layer and functionality
  – Supports PCI Express Gen 2 (5.0 Gbps) & Gen 1 (2.5 Gbps)
  – Supports root port and endpoint applications

• Connects directly to FPGA logic using Avalon-MM master and slave interfaces
Processor IP Components

• Nios® II processor
  – Altera’s second generation soft-core 32 bit RISC microprocessor
  – Licenses MegaCore® function

• Nios II instructor-led classes:
  – “Designing with the Nios II Processor”
  – “Developing Software for the Nios II Processor”
Information for Exercise

PCIe transactions targeting bar0 translated to Avalon-MM transfers sent to the on-chip memory.
PCIe transactions targeting bar1 translated to Avalon-MM transfers sent to DMA control and PCIe core CRA.
Programmed DMA can transfer data between on-chip memory and PCIe core (tx_out) to be converted to PCIe transactions.
Step 1: Create Quartus II project(1)
Step 1: Create Quartus II project(2)
Step 1: Create Quartus II project(3)
Step 2: Start Qsys
Step 3: add a Hard IP for PCI Express function to the system(1)
Configure the general, system-related settings. In the PCIe System Settings section of the IP_Compiler for PCI Express editor, choose the following options:

- Select x1 for the number of Lanes.
- Select 100 MHz for the Reference clock frequency.
- Select 64bits for the Test out width.
- Leave both the Gen2 Lane Rate Mode and Enable 62.5 Mhz application clock options de-selected.
Step 3: add a Hard IP for PCI Express function to the system(3)

Configure the PCIe Base Address Registers (BARs) for this endpoint. Go to the PCI Base Address Registers (Type 0 Configuration Space) section of the IP_Compiler for PCI Express editor and choose the following options:

- Verify BAR 0 (the first row) has been set to 64 bit Prefetchable. *The BAR type for BAR 1 should read: 1 - Occupied.*
- Click on the BAR Type field for BAR 2 (i.e. row 3). From the drop-down menu that appears, select 32-bit Non-Prefetchable.
- Leave the BAR Size and Avalon Base Address fields set to their defaults.
Step 3: add a Hard IP for PCI Express function to the system(4)

Configure the PCIe Read-Only Registers for this endpoint. Go to the Device Identification Registers section of the IP_Compiler for PCI Express editor and ensure the following options are set:

• The Vendor ID to 0x1172.
• The Device ID to 0xE001.
• The Revision ID to 0x09.
• The Class code to 0xFF0000.
• The Subsystem vendor ID to 0x1172.
• The Subsystem ID to 0x01144.
Step 3: add a Hard IP for PCI Express function to the system (5)

Configure the Buffer Configuration settings. Go to the Buffer Configuration section of the IP Compiler for PCI Express editor and choose the following options:

- For Maximum Payload Size, use the drop-down menu to select 256 Bytes.
- For the Desired performance for received requests, select High.
Step 3: add a Hard IP for PCI Express function to the system (6)

Configure the Avalon-MM interface to the embedded system. Go to the Avalon –MM Settings section of the IP_Compiler for PCI Express editor. Choose the following:

- For Peripheral Mode, use the drop-down menu to select Requester/Completer.
- Enable the Control Register Access (CRA) Avalon slave port option.
- Leave the Auto Enable PCIe Interrupt (enabled at power-on) and Disable Auto Reordering for Rx Completion TLP’s options disabled.
Step 3: add a Hard IP for PCI Express function to the system(7)

Configure the Avalon address translation.
Go to the Address Translation section of the IP_Compiler for PCI Express editor. Choose the following options:
• For the Address Translation Table Configuration, use the drop-down menu to select Dynamic translation table.
• Select 2 as the Number of address pages.
• Select 1 MByte – 20 bits for the Size of address pages.
Step 3: add a Hard IP for PCI Express function to the system(8)
Step 4: Add DMA Controller(1)
Step 4: Add DMA Controller(2)
Step 4: Add DMA Controller(3)
Step 5: Add on-chip Memory(1)
Step 5: Add on-chip Memory(2)
Step 5: Add on-chip Memory(3)
Step 6: Connect the component interfaces(1)
Step 6: Connect the component interfaces(2)

<table>
<thead>
<tr>
<th>From Component</th>
<th>From Interface Name</th>
<th>To Component</th>
<th>To Interface Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcie_hard_ip_0</td>
<td>pcie_core_clk</td>
<td>dma_0</td>
<td>clk</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>pcie_core_reset</td>
<td>on-chip_memory_0</td>
<td>reset1</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>pcie_core_reset</td>
<td>dma_0</td>
<td>reset</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>bar1_0</td>
<td>on-chip_memory_0</td>
<td>s1</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>bar2</td>
<td>dma_0</td>
<td>control_port_slave</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>bar2</td>
<td>pcie_hard_ip_0</td>
<td>cra</td>
</tr>
<tr>
<td>dma_0</td>
<td>read_master</td>
<td>on-chip_memory_0</td>
<td>s1</td>
</tr>
<tr>
<td>dma_0</td>
<td>read_master</td>
<td>pcie_hard_ip_0</td>
<td>txs</td>
</tr>
<tr>
<td>dma_0</td>
<td>write_master</td>
<td>on-chip_memory_0</td>
<td>s1</td>
</tr>
<tr>
<td>dma_0</td>
<td>write_master</td>
<td>pcie_hard_ip_0</td>
<td>txs</td>
</tr>
</tbody>
</table>
Step 6: Connect the component interfaces(3)
### Step 6: Connect the component interfaces(4)

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
<th>Export</th>
<th>IRQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>bar1_0</td>
<td>Avalon Memory Mapped Master</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bar2</td>
<td>Avalon Memory Mapped Master</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cra</td>
<td>Avalon Memory Mapped Slave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rx_in</td>
<td>Conduit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tx_out</td>
<td>Conduit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reconfig_tgxkb</td>
<td>Conduit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reconfig_gxbck</td>
<td>Clock Input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reconfig_fromgxb0</td>
<td>Clock Input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fixedck</td>
<td>Conduit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dma_0</td>
<td>DMA Controller</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clk</td>
<td>Clock Input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reset</td>
<td>Reset Input</td>
<td></td>
<td></td>
</tr>
<tr>
<td>control_port_slave</td>
<td>Avalon Memory Mapped Slave</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 6: Connect the component interfaces(5)

Export the calibration block interface of the Hard IP for PCIe. In the System Contents tab, highlight the `cal_blk_clk` interface of the `pcie_hard_ip_0` component. Click in the Export column and confirm the exported name for this interface is `pcie_hard_ip_0_cal_blk_clk`. Type the name, if necessary. Hit Enter when finished.
Step 6: Connect the component interfaces(6)

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>Exported Port Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>refclk</td>
<td>pcie_hard_ip_0_refclk</td>
</tr>
<tr>
<td>test_in</td>
<td>pcie_hard_ip_0_test_in</td>
</tr>
<tr>
<td>pcie_rstn</td>
<td>pcie_hard_ip_0_pcie_rstn</td>
</tr>
<tr>
<td>clocks_sim</td>
<td>pcie_hard_ip_0_clocks_sim</td>
</tr>
<tr>
<td>reconfig_busy</td>
<td>pcie_hard_ip_0_reconfig_busy</td>
</tr>
<tr>
<td>pipe_ext</td>
<td>pcie_hard_ip_0_pipe_ext</td>
</tr>
<tr>
<td>test_out</td>
<td>pcie_hard_ip_0_test_out</td>
</tr>
<tr>
<td>rx_in</td>
<td>pcie_hard_ip_0_rx_in</td>
</tr>
<tr>
<td>tx_out</td>
<td>pcie_hard_ip_0_tx_out</td>
</tr>
<tr>
<td>reconfig_togxb</td>
<td>pcie_hard_ip_0_reconfig_togxb</td>
</tr>
<tr>
<td>reconfig_gxbclk</td>
<td>pcie_hard_ip_0_reconfig_gxbclk</td>
</tr>
<tr>
<td>reconfig_fromgxb_0</td>
<td>pcie_hard_ip_0_reconfig_fromgxb_0</td>
</tr>
<tr>
<td>fixedclk</td>
<td>pcie_hard_ip_0_fixedclk</td>
</tr>
</tbody>
</table>
Step 6: Connect the component interfaces(7)

<table>
<thead>
<tr>
<th>Connections</th>
<th>Name</th>
<th>Description</th>
<th>Export</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcie_hard_ip_0</td>
<td>pcie_core_clk</td>
<td>Clock Output</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>pcie_core_reset</td>
<td>Reset Output</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>cal_blk_clk</td>
<td>Clock Input</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>txs</td>
<td>Avalon Memory Mapped Slave</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>refclk</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>test_in</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>pcie_rstn</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>clocks_sim</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>reconfig_busy</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>pipe_ext</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>test_out</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>bar1_0</td>
<td>Avalon Memory Mapped Master</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>bar2</td>
<td>Avalon Memory Mapped Master</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>cra</td>
<td>Avalon Memory Mapped Slave</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>rx_in</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>tx_out</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>reconfig_togxb</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>reconfig_gxclk</td>
<td>Clock Input</td>
<td>Click to export</td>
</tr>
<tr>
<td>pcie_hard_ip_0</td>
<td>reconfig_fromgxclk_0</td>
<td>Conduit</td>
<td>Click to export</td>
</tr>
<tr>
<td>fixedclk</td>
<td></td>
<td>Clock Input</td>
<td>pcie_hard_ip_0_fixedclk</td>
</tr>
</tbody>
</table>
Step 7: Adjust System Addresses(1)

Set the address of the pcie_hard_ip_0 component’s Control Register Access port to 0x00000000. On the System Contents tab, locate the Base address column. For the cra interface of the pcie_hard_ip_0 component, confirm that the address is set to 0x00000000. Change, if necessary, by double-clicking in the Base address field and typing in the address.

<table>
<thead>
<tr>
<th>Component</th>
<th>Interface</th>
<th>Base Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcie_hard_ip_0</td>
<td>txs</td>
<td>0x00000000</td>
</tr>
<tr>
<td>dma_0</td>
<td>control_port_slave</td>
<td>0x00004000</td>
</tr>
<tr>
<td>onchip_mem_0</td>
<td>s1</td>
<td>0x00200000</td>
</tr>
</tbody>
</table>
Step 7: Adjust System Addresses (2)

<table>
<thead>
<tr>
<th>Connections</th>
<th>Name</th>
<th>Description</th>
<th>Export</th>
<th>Clock</th>
<th>Base</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pcie_hard_ip_0</td>
<td>IP Compiler for PCI Express</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pcie_core_clk</td>
<td>Clock Output</td>
<td>Click to export</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>pcie_core_reset</td>
<td>Reset Output</td>
<td>Click to export</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cal_blk_clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>txa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>retck</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>test_in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>test_out</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bar1_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bar2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cra</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>rx_in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tx_out</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>reconfig_togxb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>reconfig_gxbck</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>reconfig_fromgxb_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>fixedclck</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dma_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>reset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>control_port_slave</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>read_master</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>write_master</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>onchip_memory_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>clk1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>reset1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Intel xRCG, System Design Center
Step 8: Choose final settings and generate the Qsys system(1)

• Review the system clock settings. On the Clock Settings tab, verify that the pcie_hard_ip_0 component is generating a 125 MHz clock for the system.

• Establish final Qsys settings. On the Project Settings tab, review the block diagram to make sure all of your exported interfaces are visible. In the Parameters section, choose the following options:
  - Device Family: Arria II GX
  - Clock crossing adapter type: FIFO
  - Limit interconnect pipeline stages to: 2
  - Generation ID: 0

• Generate the system. On the Generation tab, leave all options in the Simulation section set to None. In the Synthesis section, enable both options to Create HDL design files for synthesis and to Create block symbol file (.bsf). Click Generate.
Step 8: Choose final settings and generate the Qsys system(2)

- Generate the system. On the Generation tab, leave all options in the Simulation section set to None. In the Synthesis section, enable both options to Create HDL design files for synthesis and to Create block symbol file (.bsf). Click Generate.
Step 9: Incorporate into top-level file and compile(1)
Step 9: Incorporate into top-level file and compile(2)
Step 9: Incorporate into top-level file and compile(3)
Step 9: Incorporate into top-level file and compile(4)
Step 9: Incorporate into top-level file and compile(5)
Step 9: Incorporate into top-level file and compile(6)
Step 9: Incorporate into top-level file and compile(7)
Step 9: Incorporate into top-level file and compile(8)
Step 9: Incorporate into top-level file and compile(9)
Digital Video Design and Implementation Skills Using FPGA
Basic Digital TV Design

- TV Decoder
- ITU-R 656 Decoder
- De-Interlace
- Scaler
- VGA Timing
- YCbCr to RGB
- Composite Input
- VGA DAC
- Altera FPGA
- SDRAM
- SCLK, SDATA
- I²C Controller
- Display
Basic Design on DE2-115
Basic Digital TV Design

- TV Decoder
- ITU-R 656 Decoder
- De-Interlace
- Scaler
- YCbCr to RGB
- VGA Timing
- VGA DAC
- Composite Input
- SCLK
- SDATA
- I²C Controller
- SDRAM
- Altera FPGA
- VS
- HS
- RGB
- VGA

Intel

ESDC

InterCap Embedded System Design Center
➤ TV Decoder

- **Composite (Analog)**
- **A/DC**
- **Digital Decoder**
- **VSync**
- **HSync**
- **Field**
- **Y**
- **Cb**
- **Cr**
TV Decoder Programming Output

13.5MHz

- \( \text{xx} \quad \text{xx} \quad \text{xx} \quad Y_0 \quad Y_1 \quad \ldots \)
- \( \text{xx} \quad \text{xx} \quad \text{xx} \quad C_b_0 \quad C_r_1 \quad \ldots \)
- + Hsync + Vsync + Field

27MHz

- \( \text{xx} \quad \text{xx} \quad \text{xx} \quad \text{xx} \quad \text{xx} \quad \text{xx} \quad C_b_0 \quad Y_0 \quad C_r_1 \quad Y_1 \quad \ldots \)
- + Hsync + Vsync + Field

27MHz

- \( \text{xx} \quad \text{xx} \quad \text{FF} \quad 00 \quad 00 \quad \text{SAV} \quad C_b_0 \quad Y_0 \quad C_r_1 \quad Y_1 \quad \ldots \)
Basic Digital TV Design

- Composite Input
- TV Decoder
- ITU-R 656 Decoder
- De-Interlace
- Scaler
- YCbCr to RGB
- VGA Timing
- VGA DAC
- Altera FPGA

Inputs:
- SCLK
- SDATA
- I²C Controller
- SDRAM

Outputs:
- VS
- HS
- R
- G
- B

Monitor
Input Format: ITU-R656

FF0000: Control Code Header
XY: Field/Vertical-Active/Vertical-Active Indicator

Video Data

EAV End
H BLANK
SAV Start

HSE[10:0]  HSB[10:0]
Decoder ITU-R656 Data Stream

ITU656 Data → 2-bit Counter

2-bit Counter:

- 0 → Cb
- 1, 3 → Y
- 2 → Cr

SAV (Reset) 27MHz → 2-bit Counter
Basic Digital TV Design

- Composite Input
  - TV Decoder
  - ITU-R 656 Decoder
  - De-Interlace
  - SDRAM
  - I²C Controller

- Altera FPGA
  - VGA Timing

- Scaler
  - YCbCr to RGB
  - R, G, B

- VGA DAC
  - VS
  - HS

- VGA

This diagram illustrates the basic components of a digital TV system, including decoding, de-interlace, memory management, and output to a display device.
What is Interlaced Scan?

Upper Field

Interlace Scan

Lower Field

Progressive Scan
Why is Interlaced Scan?

<table>
<thead>
<tr>
<th>Spatial domain</th>
<th>Temporal domain</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Spatial diagram" /></td>
<td><img src="image2" alt="Temporal diagram" /></td>
</tr>
</tbody>
</table>

- **Spatial domain**
  - Shows a grid with rows and columns.
  - Rows are labeled 1 to 10, and columns are labeled 1 to 10.

- **Temporal domain**
  - Represents the sequence of frames over time.
  - Frames are labeled n-2, n-1, and n.
  - Shows the progression of frames in the temporal domain.
De-interlaced Method

- **Video mode.**
  - Intra-Field (Bob or Interpolation or ELA)
  - Inter-Field (Blend or Weave)
  - Motion Adaptive
  - Motion Compensation

- **Film mode.**
  - 3:2 Pull-Down (NTSC)
  - 2:2 Pull-Down (PAL)
Video Mode

**Intra-Field: Bob**

**Advantage:**
Video no loss. Line buffer only. No extra judgment.

**Disadvantage:**
Video Mode

- **Intra-Field: Interpolation**

- **Advantage:**
  Video no loss. Line buffer only. No extra judgment.

- **Disadvantage:**
  Less resolution. Jagged.

![Diagram of Intra-Field Interpolation]

- Source Line n+1 → Line Buffer → Output Line 2n-3/2n-2/2n-1
- Line Buffer → Line 2n-1/2n/2n+1

Method:
Interpolate the new line from the scan lines for each field.

Advantage:
Video no loss. Line buffer only. No extra judgment.

Disadvantage:
Less resolution. Jagged.
Video Mode

Inter-Field: Weave

Method:
Merge the odd and even field.

Source

Output
Video Mode

Inter-Field: Blend

Method:
Resize the average odd and even field.

Source

Odd1 001
Odd1 002
... Odd1 239
Odd1 240

Odd1 001
... Odd1 239
Odd1 240

Odd2 001
... Odd2 239
Odd2 240

Even2 001
... Even2 239
Even2 240

Time

Avg (Odd1 001/Even1 001)
Avg (Odd1 001/Even1 001)
Avg (Odd1 001/Even1 001)
Avg (Odd1 002/Even1 002)
Avg (Odd1 002/Even1 002)
... Avg (Odd1 239/Even1 239)
Avg (Odd1 239/Even1 239)
Avg (Odd1 240/Even1 240)
Avg (Odd1 240/Even1 240)

Avg (Odd2 001/Even1 001)
Avg (Odd2 001/Even1 001)
Avg (Odd2 001/Even1 001)
Avg (Odd2 002/Even1 002)
Avg (Odd2 002/Even1 002)
... Avg (Odd2 239/Even1 239)
Avg (Odd2 239/Even1 239)
Avg (Odd2 240/Even1 240)
Avg (Odd2 240/Even1 240)

Avg (Odd2 001/Even2 001)
Avg (Odd2 001/Even2 001)
Avg (Odd2 001/Even2 001)
Avg (Odd2 002/Even2 002)
Avg (Odd2 002/Even2 002)
... Avg (Odd2 239/Even2 239)
Avg (Odd2 239/Even2 239)
Avg (Odd2 240/Even2 240)
Avg (Odd2 240/Even2 240)
Film Mode

 ➢ 3:2 Pull Down Detection

Original film frames 24 frames/sec

Interlaced fields 60 fields/sec

10 Interlaced fields for every 4 film frames

odd even odd even odd even odd even odd even odd even

Scan Code = 000001

?? Scan Code = 000011
Film Mode

**PAL: 2:2 Pull Down**

- Original film frames: 24 frames/sec
- Interlaced fields: 50 fields/sec
- Assembled into progressive frames
- Displayed in a 2-2 pattern
Film Mode

2:2 Pull Down Detection

If `sawtooth_even1 > threshold`, then sawtooth detected!

```
Sawtooth 0 1 0 1
odd1 even1 odd2
odd1 even1 odd2
```
Basic Digital TV Design

- Composite Input
  - TV Decoder
  - ITU-R 656 Decoder
  - De-Interlace
  - SDRAM
  - I²C Controller
  - SCLK, SDATA

- Altera FPGA
- De-Interlace

- Scaler
- YCbCr to RGB

- VGA Timing
- VGA DAC
- VS, HS
- R, G, B

- Display

- Intel Basic Digital TV Design
Scaler

Digital Video Source → Buffer → Digital Rescaling Algorithm → LCD Display

Rescaling
4:3 => 16:9

Original 4:3 Image

Simple Linear Scaling

4:3 Image Fit on 16:9 Display

Non-Linear Horizontal Scaling
Basic Digital TV Design

TV Decoder
ITU-R 656 Decoder
De-Interlace
Scaler
VGA Timing
VGA DAC

Composite Input

SDRAM

I²C Controller

SCLK
SDATA

Altera FPGA

YCbCr to RGB

R
G
B

VS
HS

intel
Color Space Conversion

◆ YCbCr to RGB

\[
R = 1.164(Y-16) + 1.596(Cr-128) \\
G = 1.164(Y-16) - 0.391(Cb-128) - 0.813(Cr-128) \\
B = 1.164(Y-16) + 2.018(Cb-128)
\]

\[
\begin{align*}
1.164 &= 1 + 1/2^3 + 1/2^5 + 1/2^7 \\
1.596 &= 1 + 1/2 + 1/2^4 + 1/2^5 \\
0.391 &= 1/2^2 + 1/2^3 + 1/2^6 \\
0.813 &= 1/2 + 1/2^2 + 1/2^4 \\
2.018 &= 2 + 1/2^6
\end{align*}
\]
Basic Digital TV Design

- TV Decoder
- ITU-R 656 Decoder
- De-Interlace
- Scaler
- YCbCr to RGB
- VGA Timing
- VGA DAC
- Composite Input
- I²C Controller
- SDRAM
- SCLK, SDATA
- Altera FPGA
- VS, HS
- R, G, B
- Intel
- Altera FPGA
- VGA DAC
- VS, HS
- R, G, B
- Basic Digital TV Design
VGA Timing

- Video Line
- Horizontal Blanking Interval
- Pixel RGB Levels
- Horizontal Blanking Interval
- Front Porch
- Back Porch
- Vertical Sync
- Horizontal Sync

VGA MONITOR
VGA Controller Architecture

- PLL
- Source Clock
- H-Sync Generator
  - H-Counter
- Pixel Clock
- V-Sync Generator
  - V-Counter
- VGA Data Control
  - DATA
- Memory
  - X
  - Y
- VGA Data
- Request
- H-Sync
- V-Sync
- X
- Y
- R
- G
- B
Video Quality?

Composite Input

TV Decoder

ITU-R 656 Decoder

De-Interlace

Scaler

VGA Timing

VGA DAC

YCbCr to RGB

R

G

B

VS

HS

I^2C Controller

SDATA

SCLK

SDRAM