Peter the Great St. Petersburg Polytechnic University

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[Institute of Computer Science and Technology](http://english.spbstu.ru/structure/institut_computernikh_nauk_i_tekhnologiy/)

Department of Computer Systems & Software Engineering

Lecture

*Using the Nios II Processor Hardware Development*

student:

Ivanov Il’ya

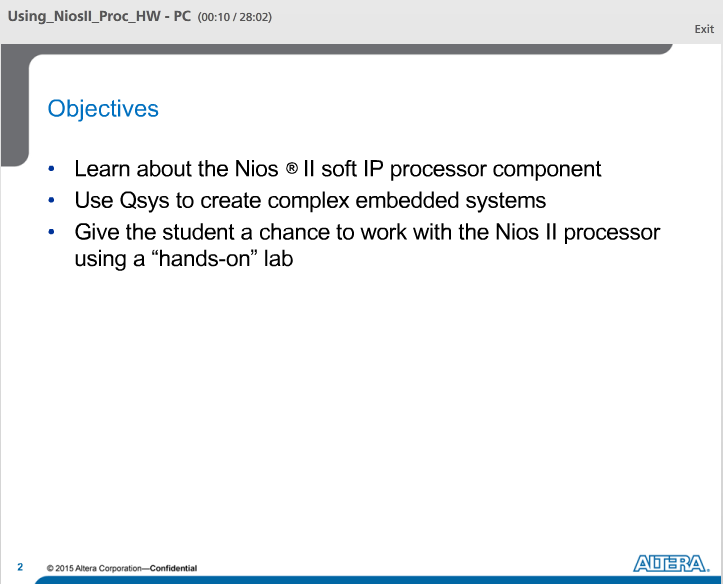
group: 3530901/70203

lecturer:

Antonov A.P.

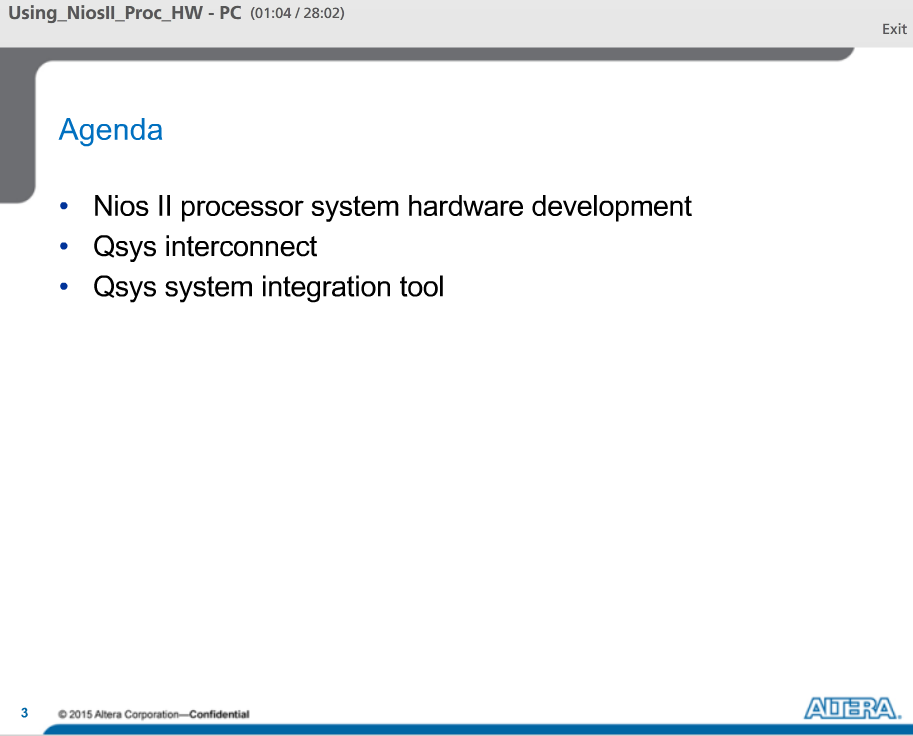
Saint-Petersburg

2020



Objectives for this lecture:

Describe what the Nios II soft core processer really is and how to use the Qsys system integration tool to create an embedded system with or without the Nios II processor.



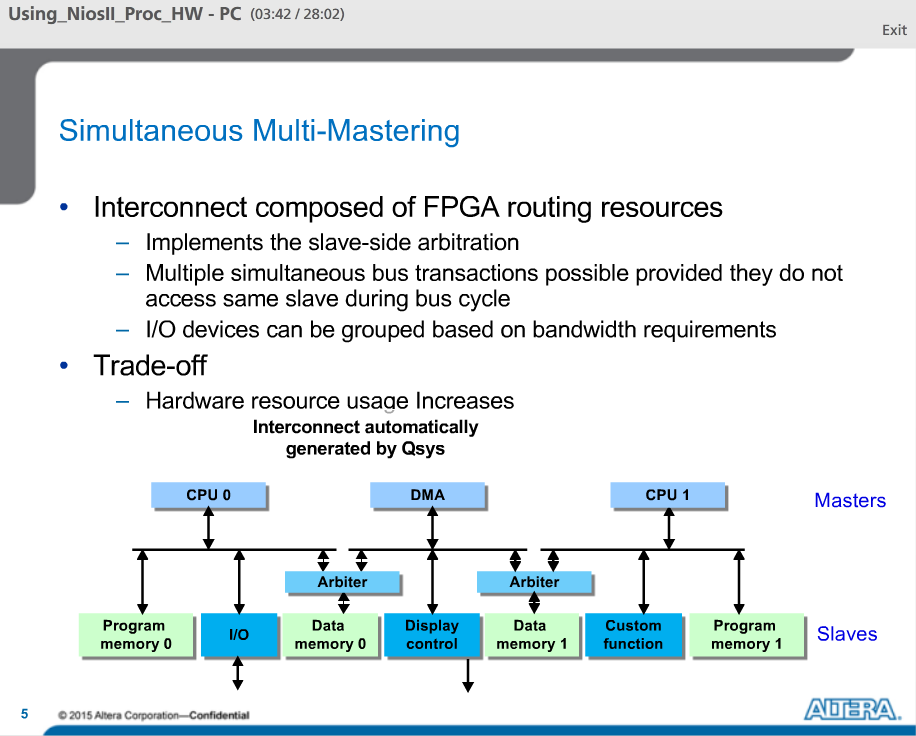
We will start by talking about the Nios II Processor and embedded systems inside an Altera FPGA. Next, we will talk a bit about the Nios II architecture, focusing on the system interconnect and its advantages over a traditional system bus. Finally, we will talk about Qsys sys integration tool, a tool that is run from Quartus that allows you to build embedded subsystems that may or may not include NII processor.



The Nios II Processor is developed internally by Altera and it employs the Harvard architecture which means it has a separate instruction and data bus. It is a royalty free product. It is a 32-bit RISC soft core microprocessor. It means that the processor and all the peripherals are written in either Verilog or VHDL. You can target the processor for any of the Altera’s FPGAs and synthesize the system using Quartus II integrated Synthesis Engine.

You can add multiple CPUs with multiple memory access controllers or even build a system without a processor and just build a network of peripherals to interact with an external processor.

The performance of the processor is related to number of recourses you will to use and to the performance of the FPGA family you are targeting.

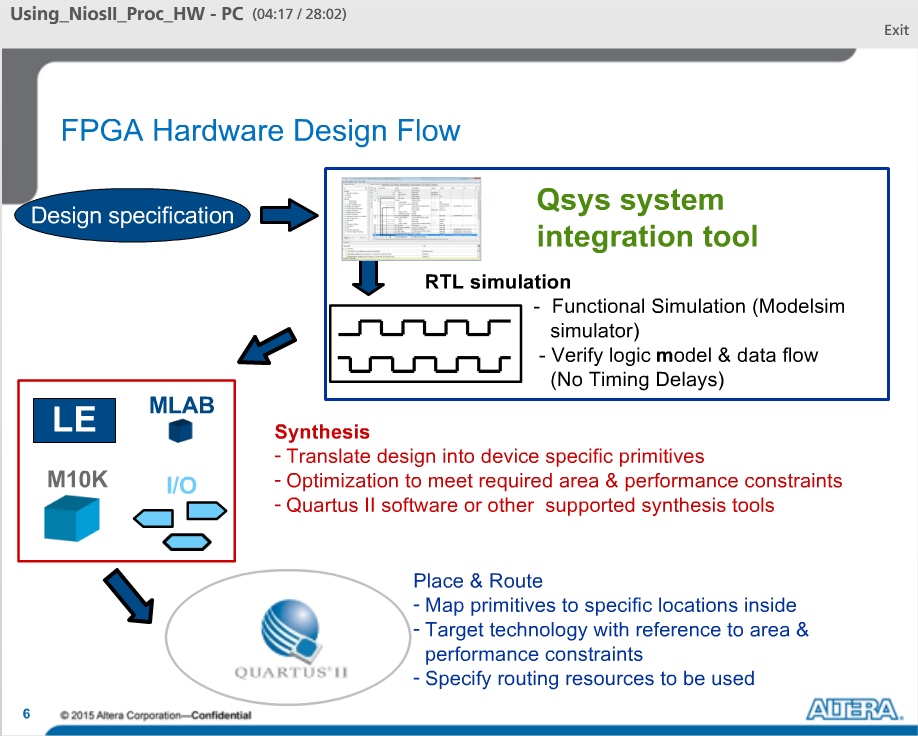


Aside from the flexibility, there is another intrinsic advantage to having a soft-core process. That is the simultaneous multi-mastering capabilities of the system interconnect.

The Qsys interconnect and Altera’s Avalon Standard interfaces allow multiple master ports to transfer data simultaneously. Arbitration logic stalls a master port only when multiple master ports attempt to access the same slave port during the same cycle.

Because everything is implemented in programmable logic, there are no unnecessary master-slave connections. No logic is wasted.

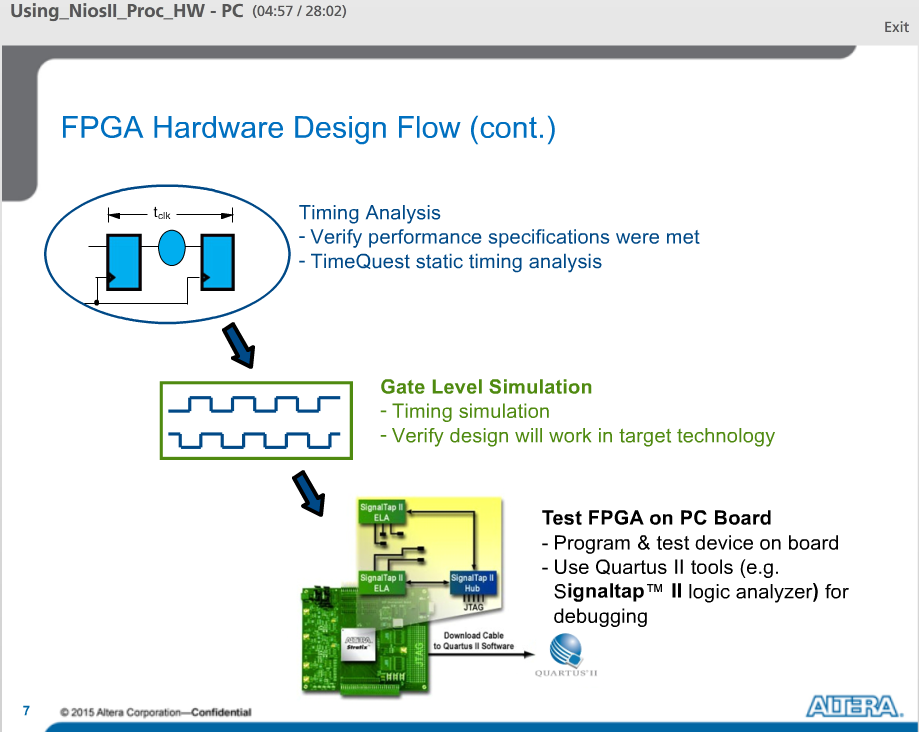
The Qsys tool provides configurable arbitration settings, and arbitration for each slave port is specified independently. The details of arbitration are encapsulated inside the interconnect.



Qsys in the FPGA design flow:

At the start of the FPGA design flow, you will define the design specs you want to achieve. But instead of coding the system manually the Qsys tools will automatically generate the HDL for your system and will also generate simulation models and testbench to allow you to easily perform RTL simulation.

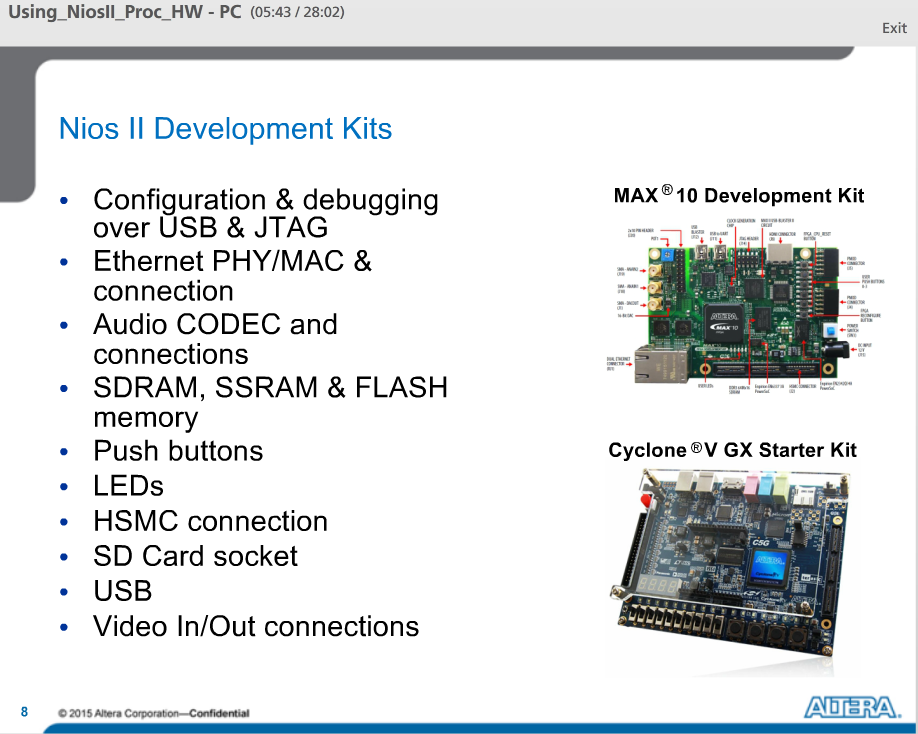
A designer can then run synthesis and place on route on the generated HDL to create the programming file.



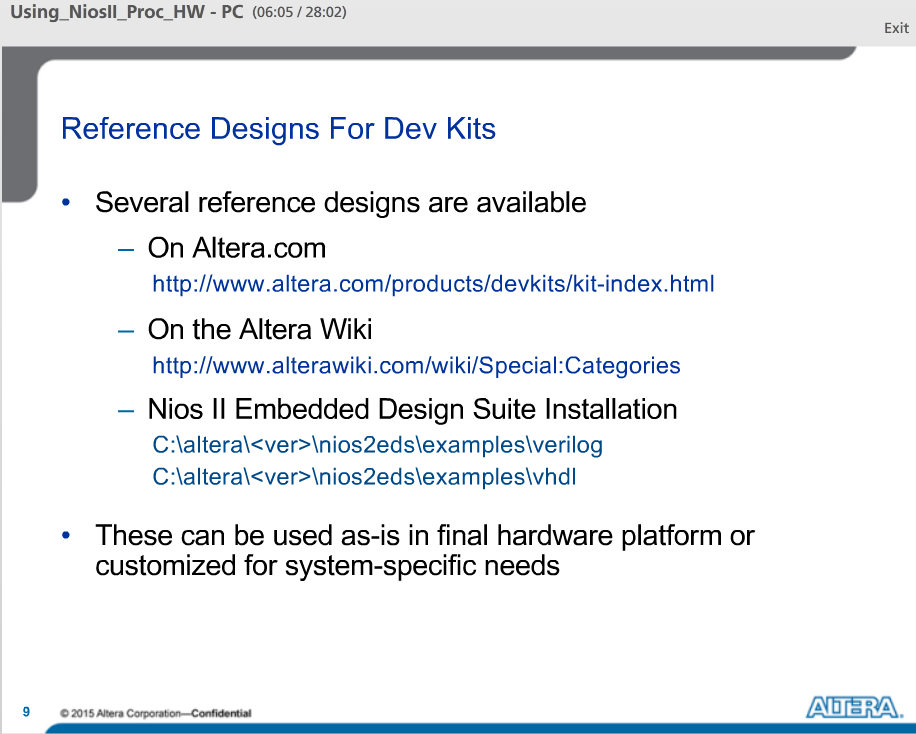
After place and route, timing analysis can be performed using tools like Altera’s TimeQuest static timing analyzer, to verify that timing specifications can be met by your system.

The optional Gate level simulations can be run to assure the design functions correctly once programmed into the FPGA by considering actual routing and cell delays.

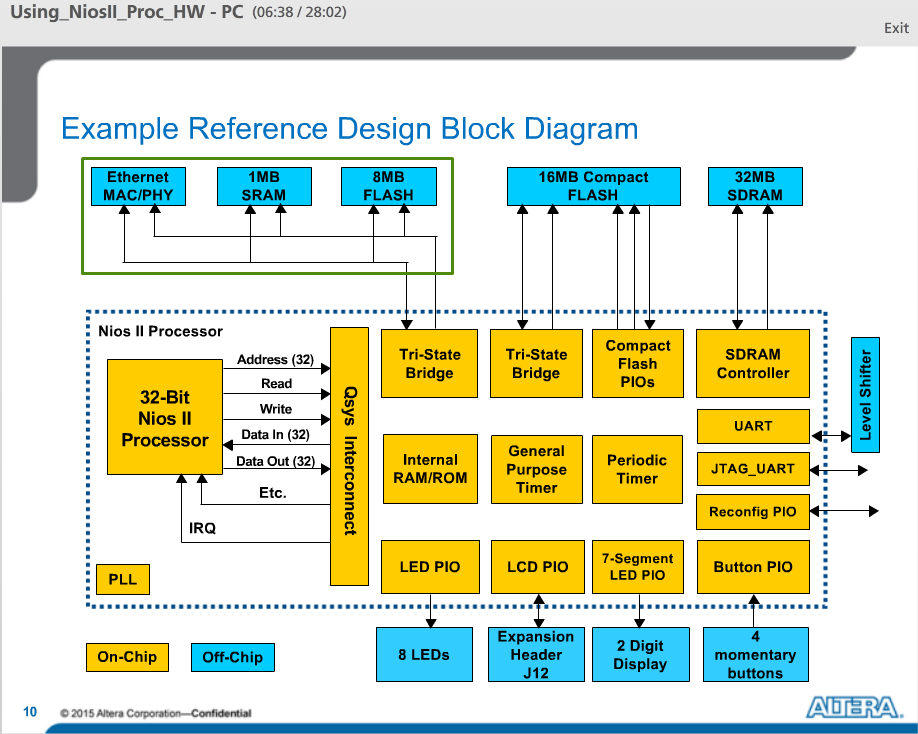
Then you can Program the FPGA on a board using the Quartus II Programmer to test in-system. Use tools like the SignalTap II embedded logic analyzer to probe internal nodes of your system to debug the design in system.



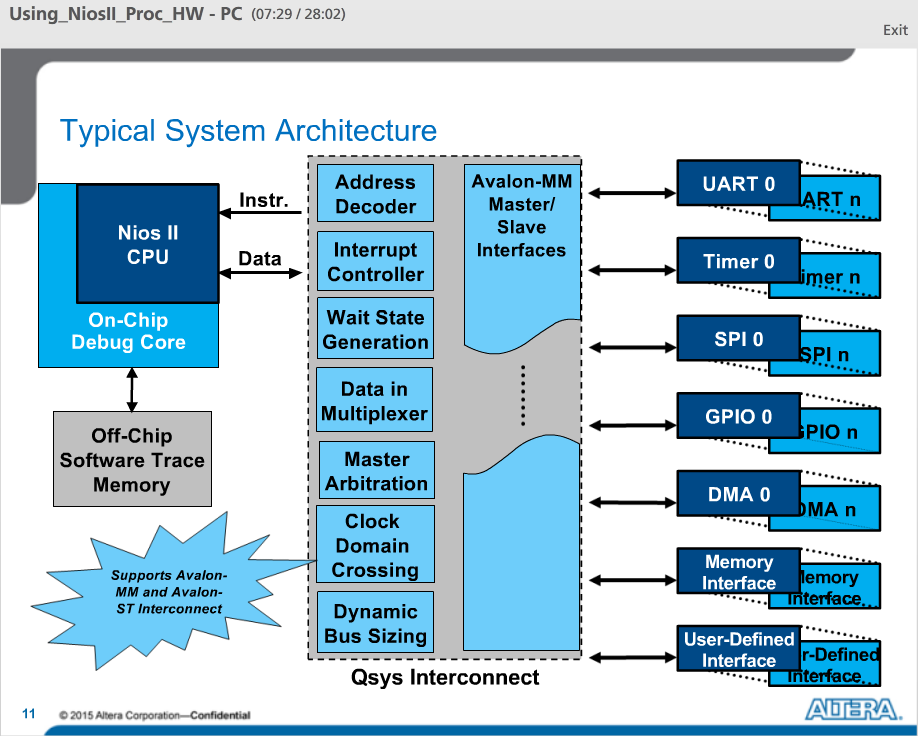
Development boards are great prototyping tools allowing you to take your design to the board level and examine how your FPGA design performs in real hardware. Altera has many development boards targeting Cyclone, Arria, and Stratix FPGA families. The MAX10 device also supports the Nios II processor. In a typical kit, in addition to the FPGA, depending on the board there are several peripherals available for prototyping, buttons, LEDs, LCDS, different types of memory, and expansion connectors to allow you to build daughter cards and attach them to the development board.



All development kits come with several types of reference designs. This allows a user to get a Nios design running on a development kit in a short time.



This is an example of a standard reference design. It consists of a processor and many other peripherals all connected through the Qsys interconnect.

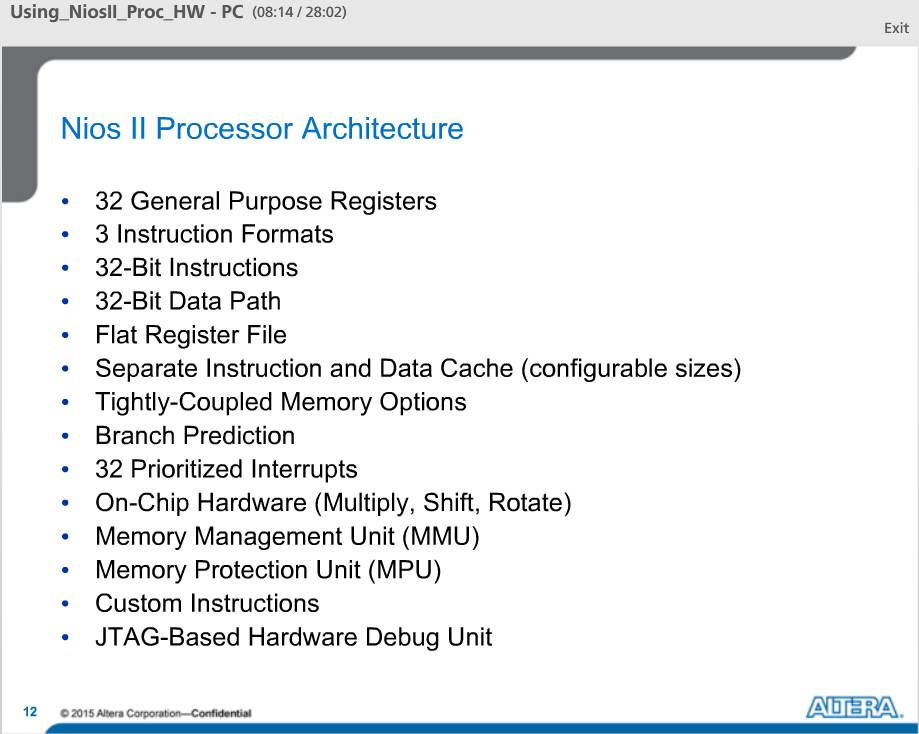


Typical NII system architecture.

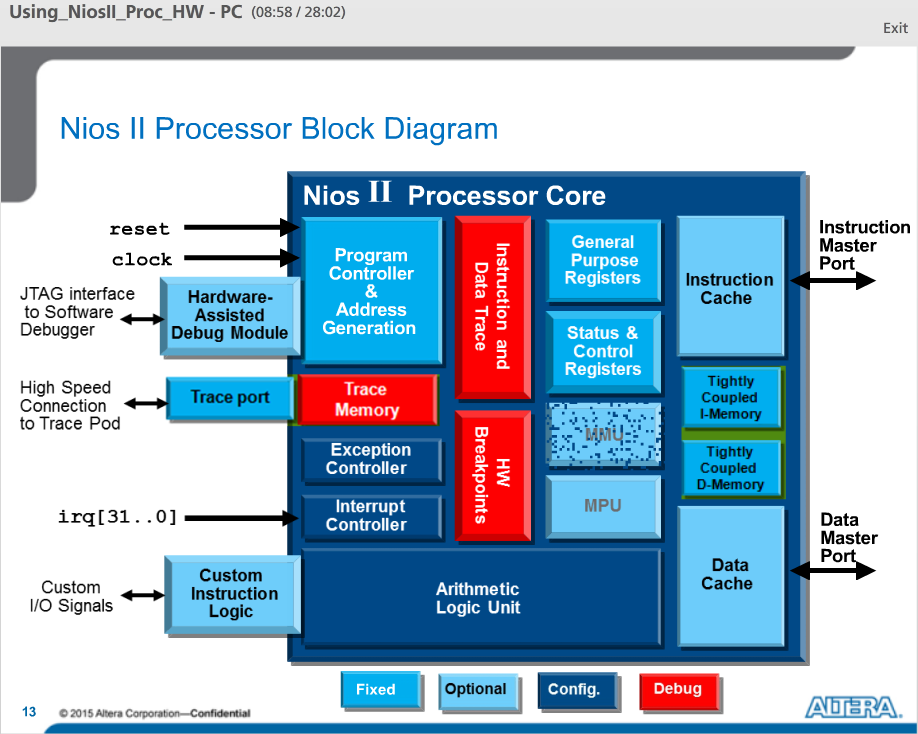
You can see the Nios II processor connected to a number of peripherals. The vector notation for the components on the right denotes you can have any number of the components instantiated in your system including different memory interfaces and User Defined interfaces to connect to your own logic. These peripherals are all connected to the Qsys interconnect which will automatically handle all of address decoding, arbitration, wait state generation, bus sizing, and clock domain crossing to make it possible for data to flow through your system.

Typically, you will also have an on-chip debug core attached to the Nios II as shown here which allows you to connect to your processor from the software development tools.

The Qsys interconnect supports both Avalon MM and memory mapped interfaces and Avalon ST or streaming interfaces.



The architecture of Nios II processor.

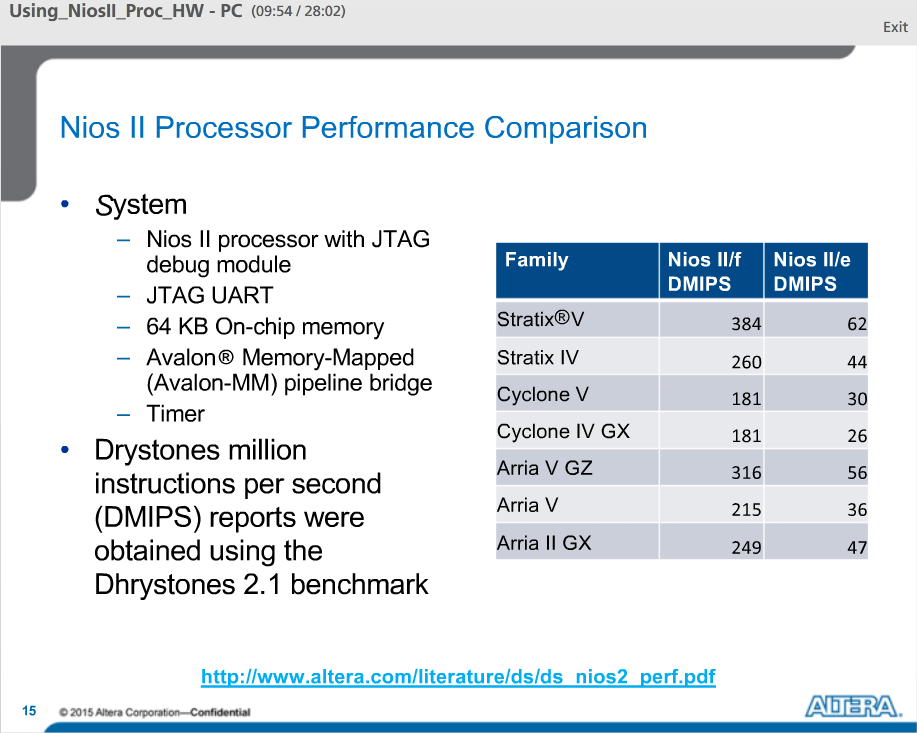


This is a block diagram of the Nios II processor showing the program controller, general purpose registers, exception handling unit, and the ALU.

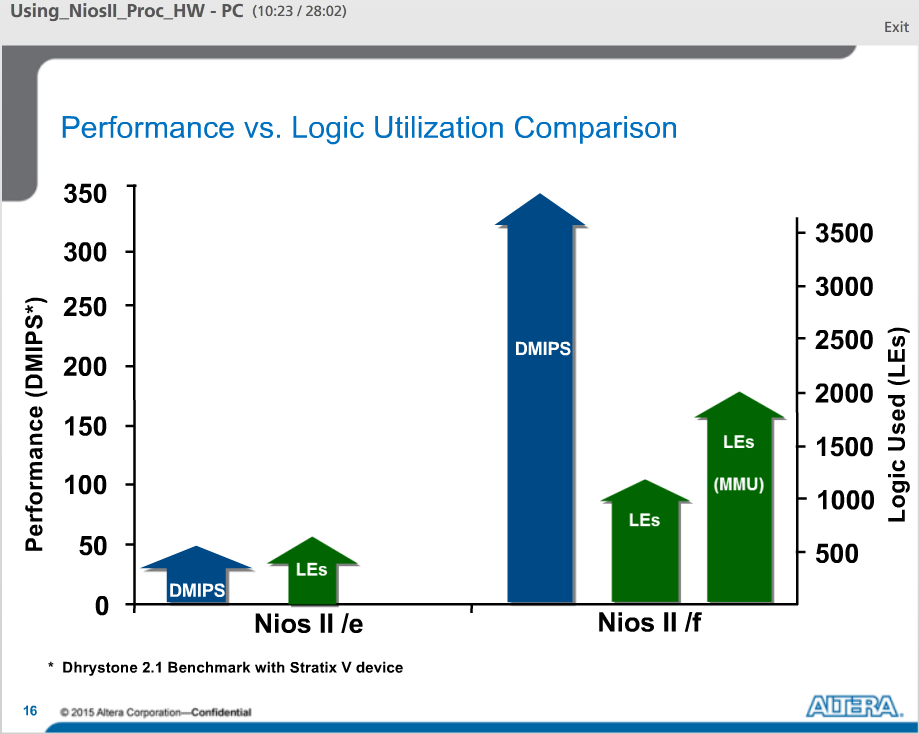
Optionally you can add an instruction or a data cache depending on which version of the core you choose. You can parameterize the core. You can include a HW assisted Debug module that gives you the ability to snoop on your system and optionally implement HW breakpoints. You can add instruction and data trace and take off chip trace data through a trace port. You can add custom instructions directly to the ALU to extend the functionality of the CPU and you can optionally add tightly coupled Instruction or data memories and if required, you can include an MMU or MPU.



There are two versions of Nios II processor to choose from depending on your requirements. The fast version is optimized for speed but takes the most FPGA resources. Economy takes fewer FPGA resources but does not provide the same level of performance as the fast processor. The fast core does required a license from Altera, but the economy core is free to use without a license.



This table compares the performance of the two Nios II processor cores for each of the FPGA families.

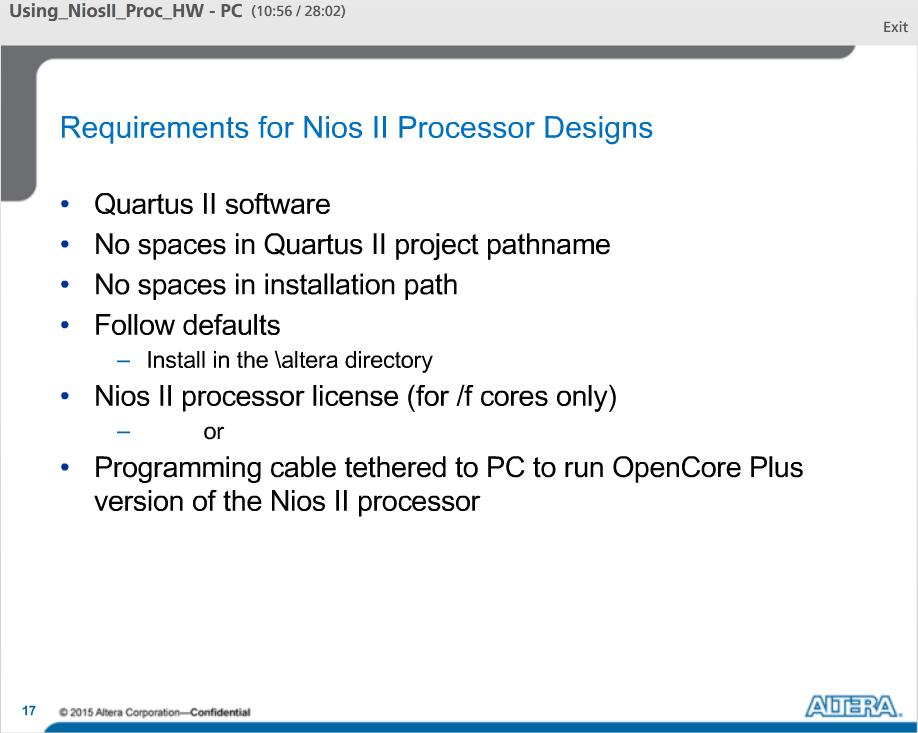


This table shows a comparison between logic utilization and performance for the two versions of Nios II processor.

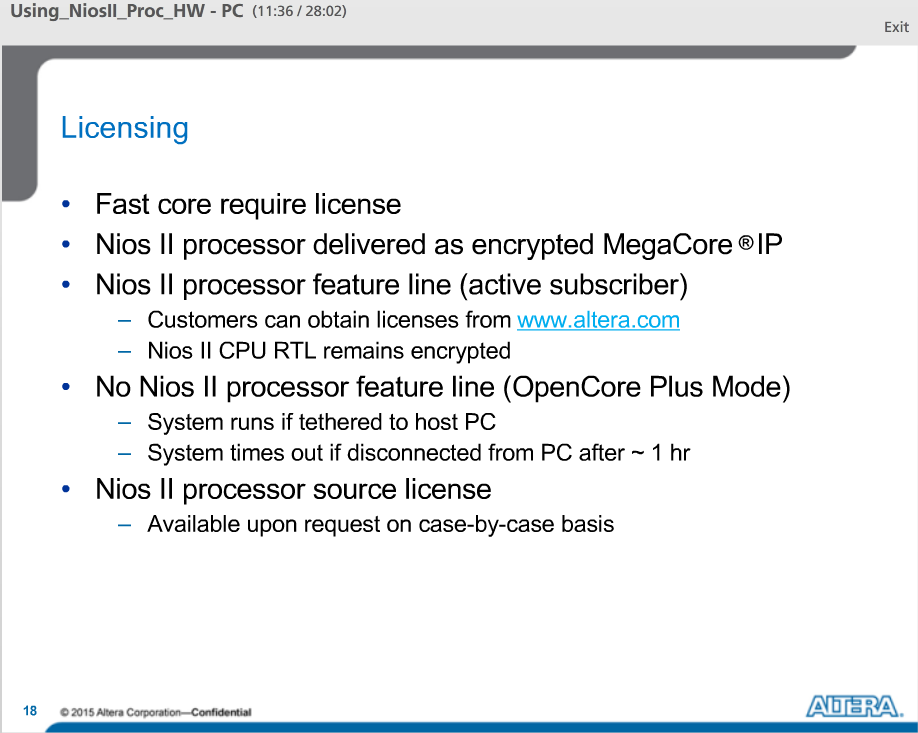
The fast core requires about 1800 Logic Elements without an MMU and provides approximately 350 DMIPS. While the economy processor uses only 600 Logic elements but provides only about 50 DMIPs. With an MMU the gate count for the fast processor jumps to 3200 Logic Elements. The MMU is not available for the economy processor.

HW multiplication and barrel sifting can be done in 1 cycle with the fast processor. This is done with the Stratix Family DSP block. With standard, it is 3 cycles, and with economy it needs to be handled in SW.

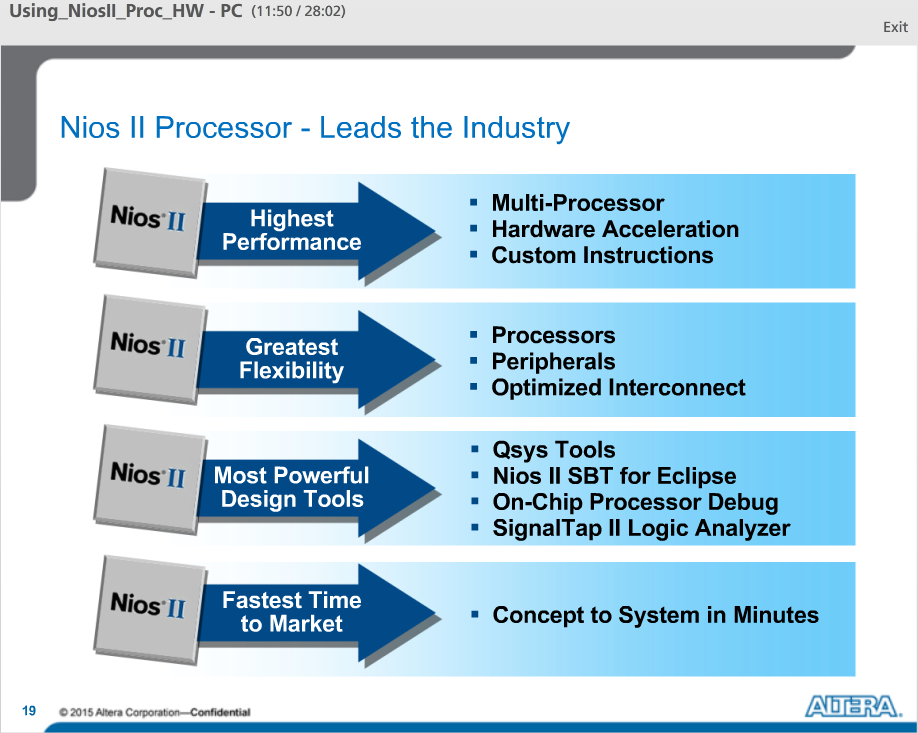
With fast you get configurable Instruction and Data cache; with standard you only get instruction cache, in economy you do not get any caches at all.



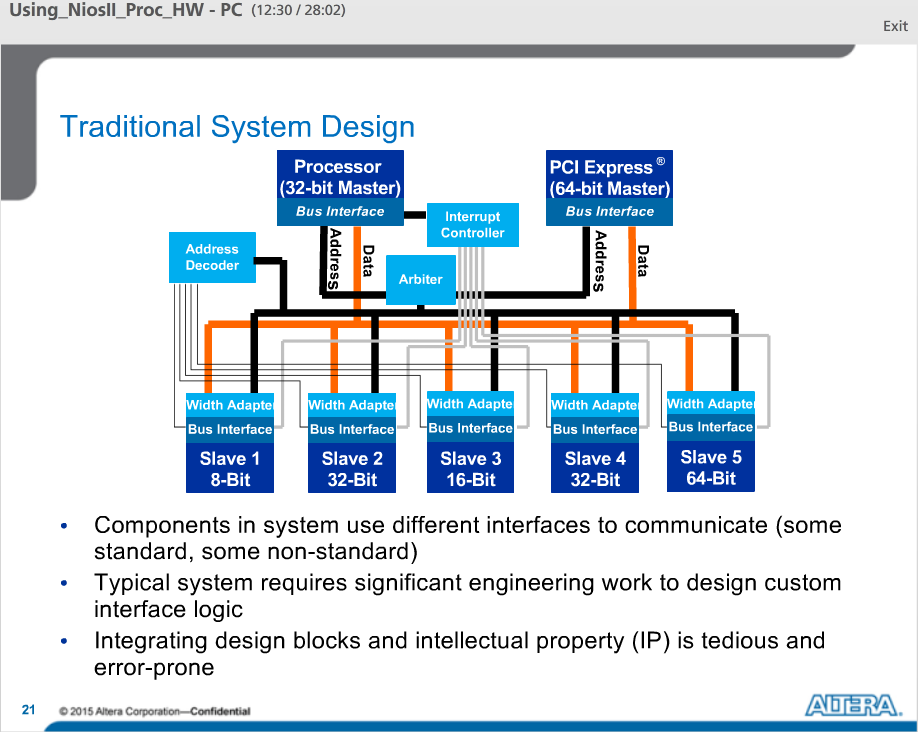
Requirements and recommendations for running Nios II processor designs.



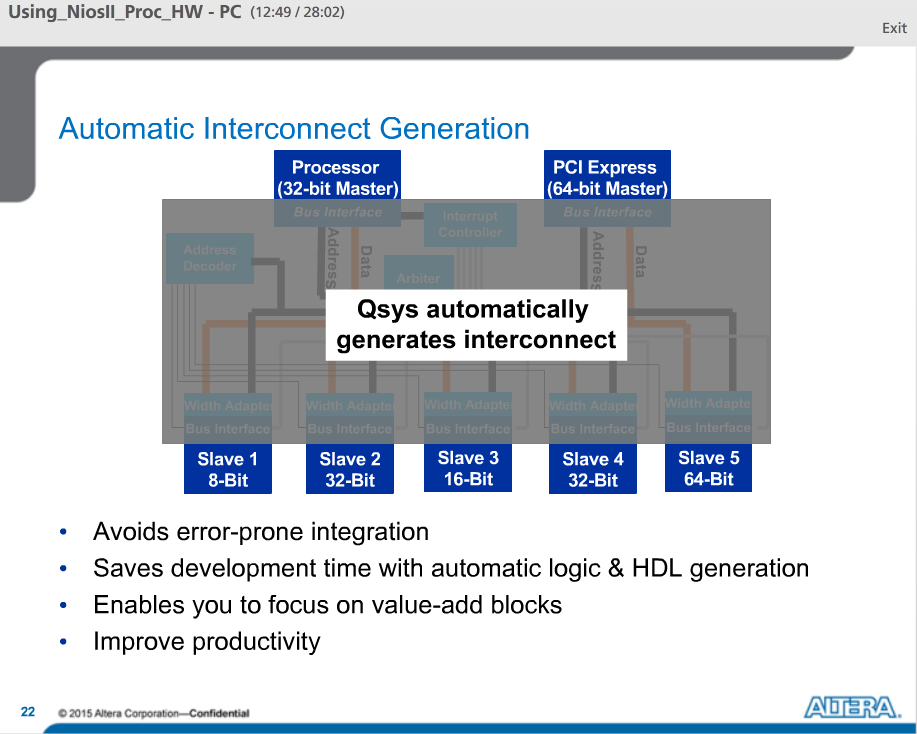
To use the Nios II Processor fast core you will need a license. The core itself is delivered as an encrypted megacore, meaning its non-readable HDL code.



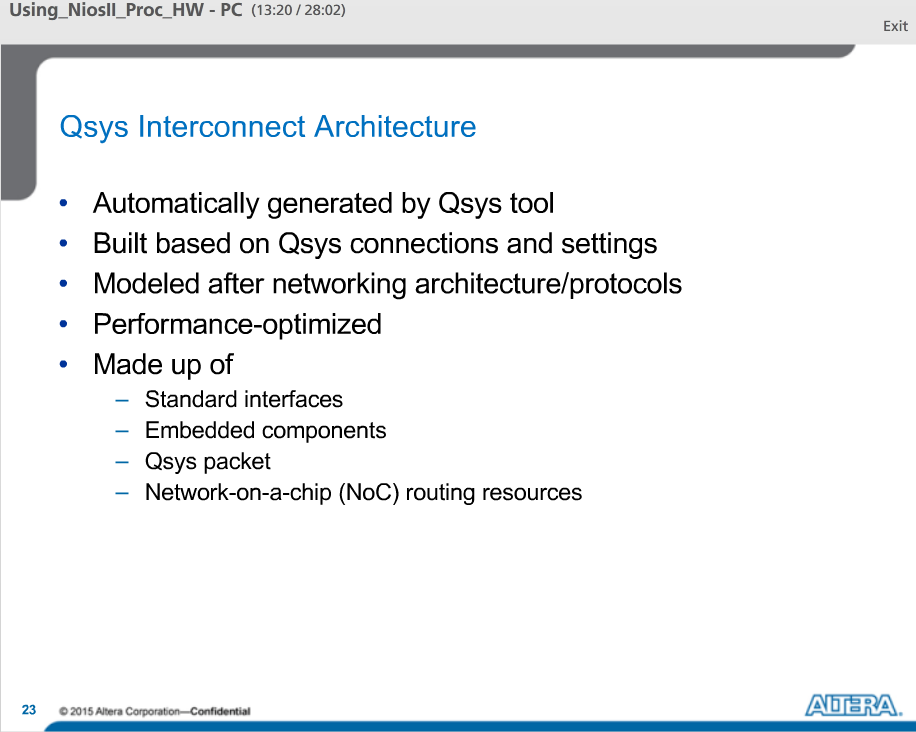
Nios II leads the industry, with the highest performance soft core processor available and creates flexibility through a powerful set of design tools.



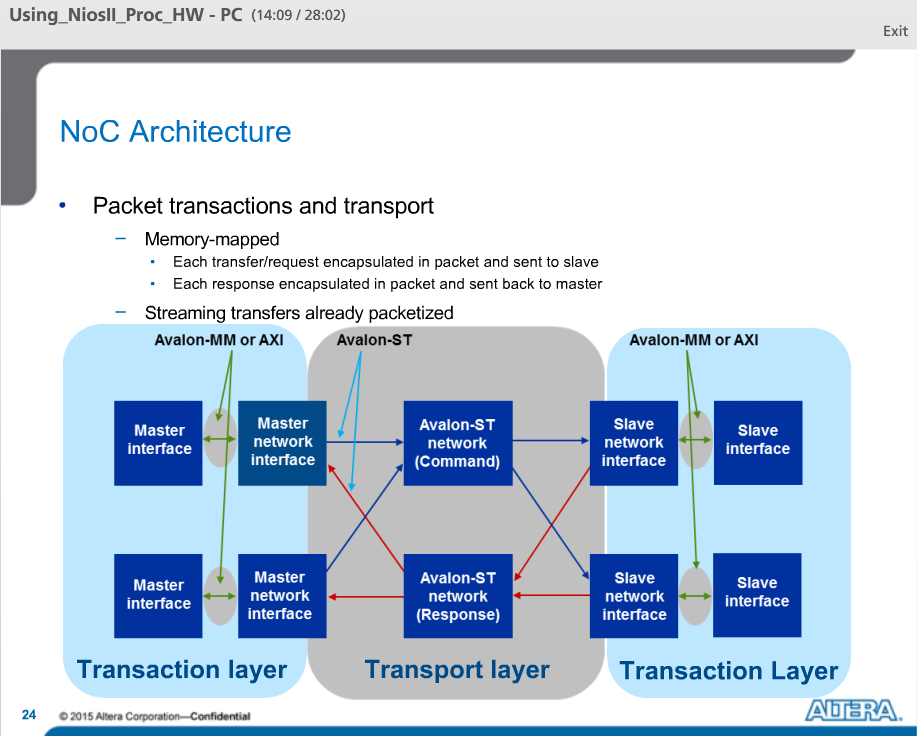
Example of a typical system. You have different devices in your system that have different interfaces that need to communicate in order for the system to work. Connectivity logic such as arbitration and width matching require a lot of engineering effort to design and maintain. This type of logic does not add a lot of value, but it is just necessary to get the system connected.



When you use Qsys to design your system, Qsys generates a custom-built interconnect structure that facilitates communication between all of your system components. This abstracts away from the designer the manual process of manipulating interconnects and signals and allows the designer to instead focus on creating blocks of high value.



The Qsys interconnect is what connects all Qsys components together. It is a custom-generated interconnect structure generated automatically by Qsys and is based on the connections and settings chosen in the Qsys GUI. The interconnect is made up of standard interfaces, embedded system structures and network on a chip routing resources.

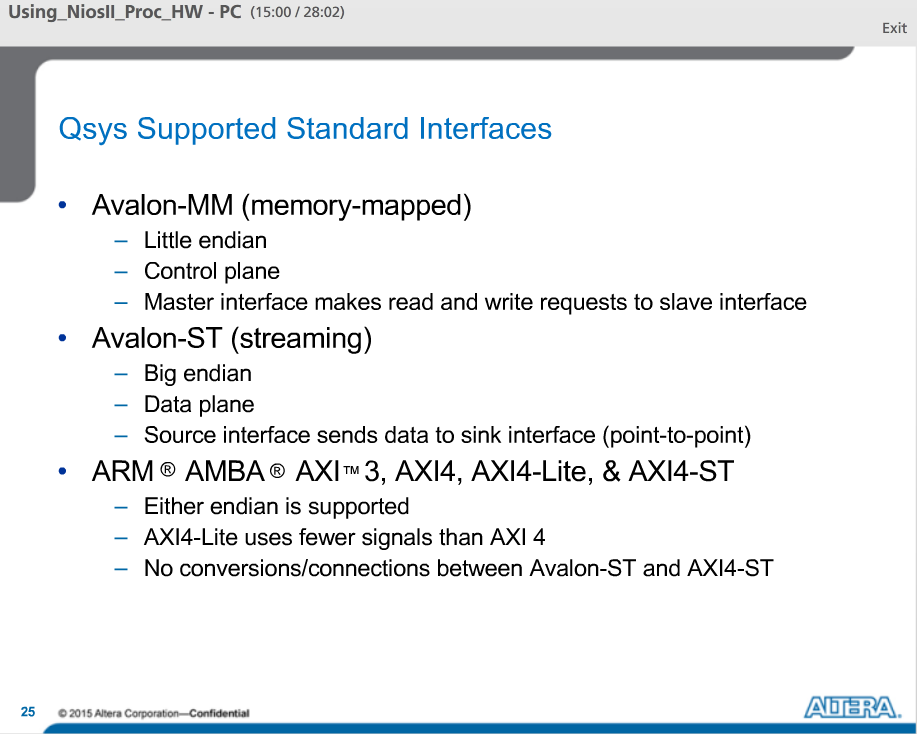


Qsys system interconnect uses the Network-on-a-chip, or NoC architecture. The NoC serves to transfer packets of data through the system to facilitate communication.

For memory-mapped interfaces, read and write transfers are converted into packets and sent through the system from master to slave where they are converted back into memory-mapped transfer requests. Responses from slaves are again converted into packets and sent through the system back to the requesting master where they are again converted back into memory-mapped transfer responses.

Because the Qsys network on a chip architecture uses a wide packet format, latency is not an issue. Writes can be accomplished in one cycle while reads have a round trip latency of 1 cycle.

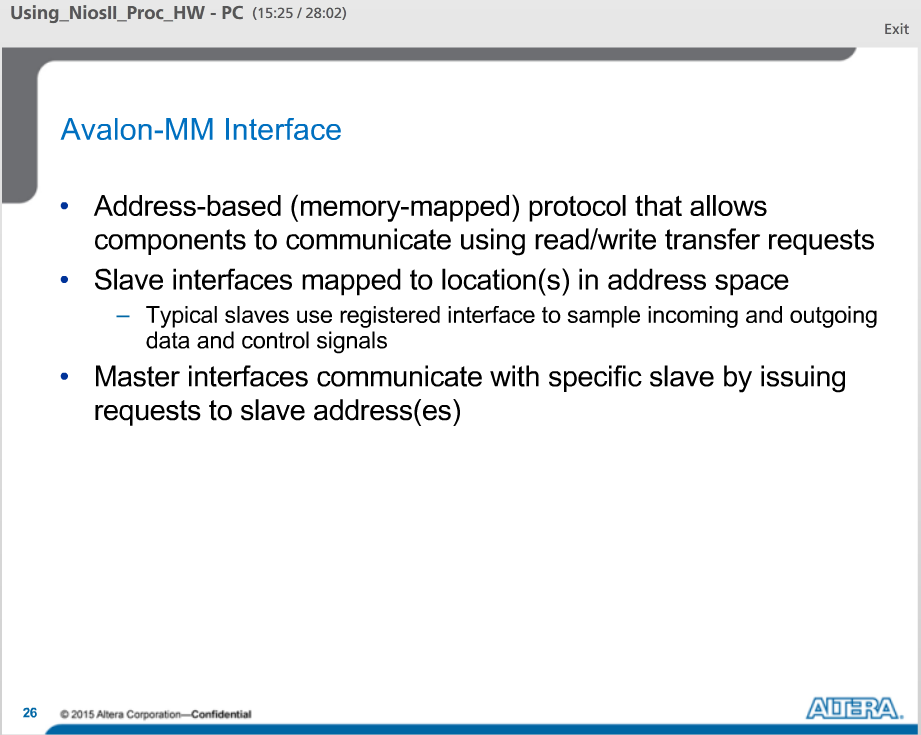
Since streaming transfers are already packetized, they are transferring through the system as is.



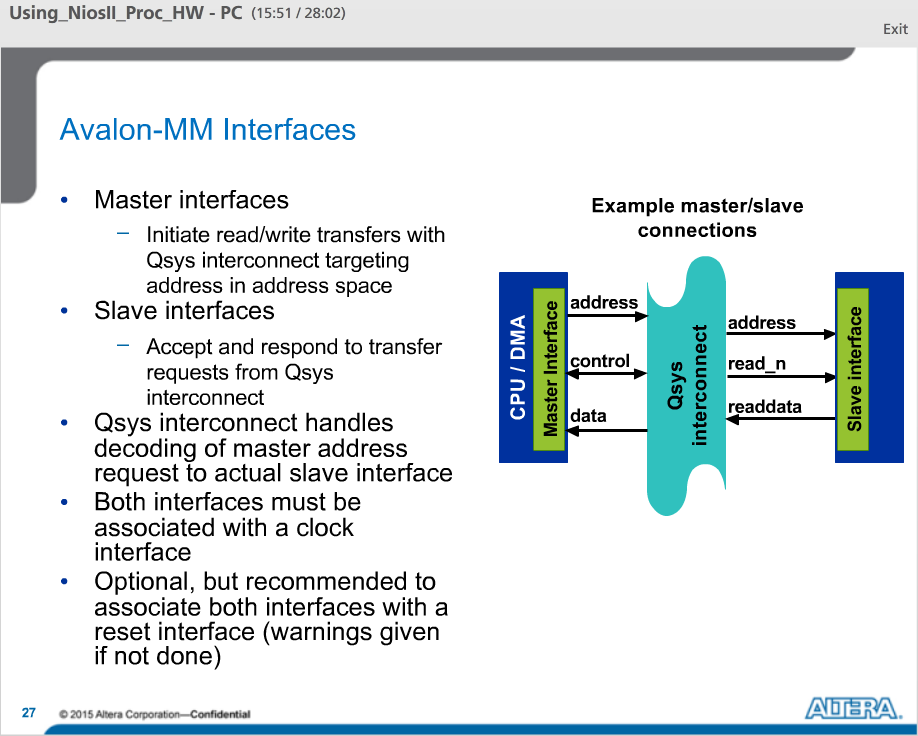
Qsys supported standard interfaces include Altera’s Avalon-MM and Avalon-ST as well as several interfaces from ARM. These interfaces can be grouped into tow general categories called memory-mapped and streaming.

In a memory mapped interface, the master issues read or write commands to the interconnect and the interconnect relays those commands to the slave. Every slave has an address in the memory map of each master that is connected to it.

Streaming interfaces are point-to-point data interfaces. Designed to handle the high bandwidth low latency requirements of a data plane. They are unidirectional with data flowing from source to sink.



The Avalon-MM protocol is used for standard data and control transfers. It uses an address-based, or memory-mapped, methodology.



The Avalon-MM protocol consists of master and slave interfaces. Master interfaces must initiate transfers to the Qsys interconnect. Slave interfaces respond to transfers from the Qsys interconnect. Qsys decode the master’s address request and forward it to the correct slave interfaces.

Both master and slave interfaces must be associated with both clock and reset interfaces.

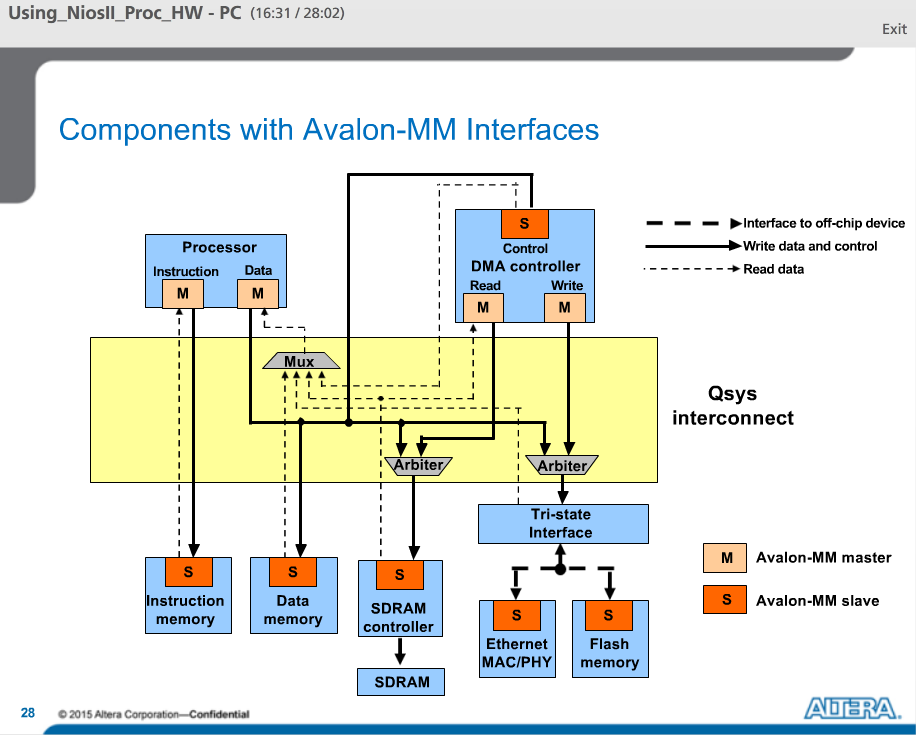
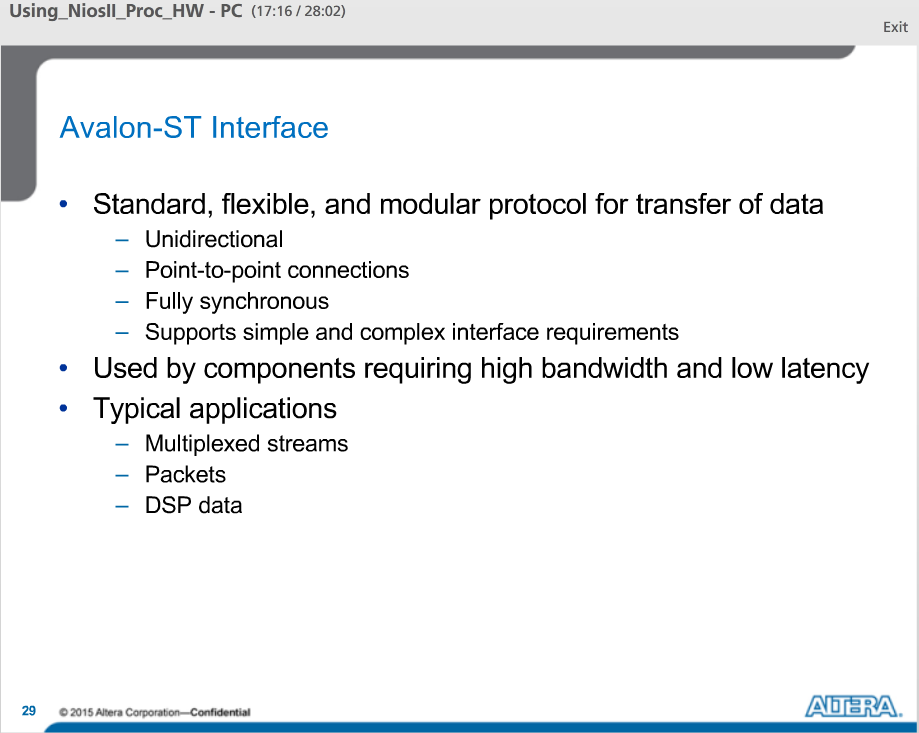


Diagram of a system with components using Avalon-MM interfaces.

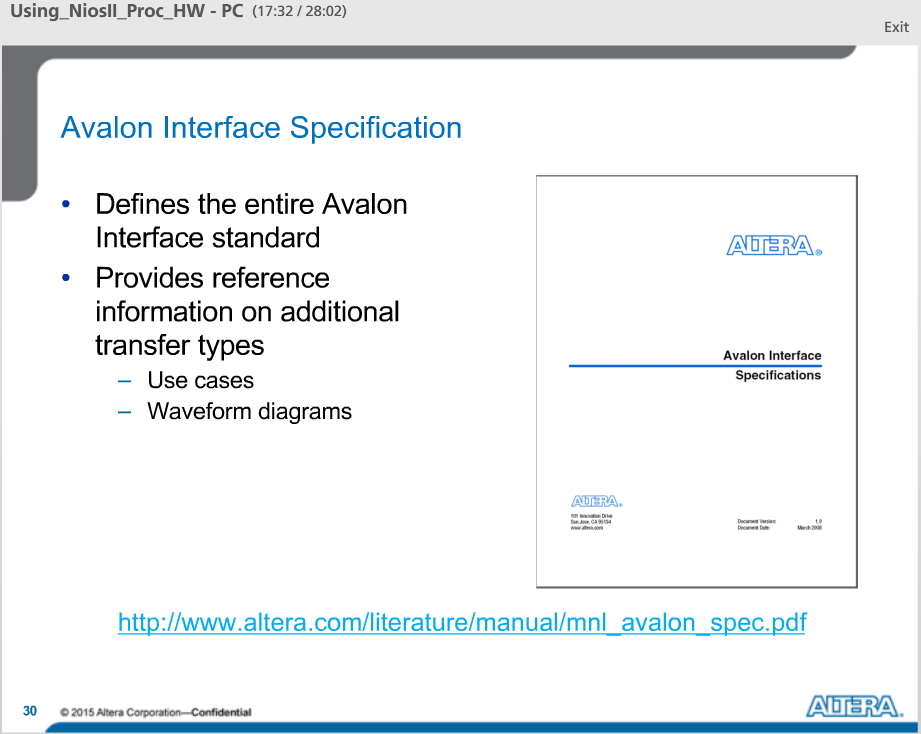
The Processor and the DMA controller are the masters of this system initiating transfers to the slave components on the bottom. DMA has both slave and master interfaces where it can receive commands from the CPU through the slave port while issuing data transfers from its master read and write ports.



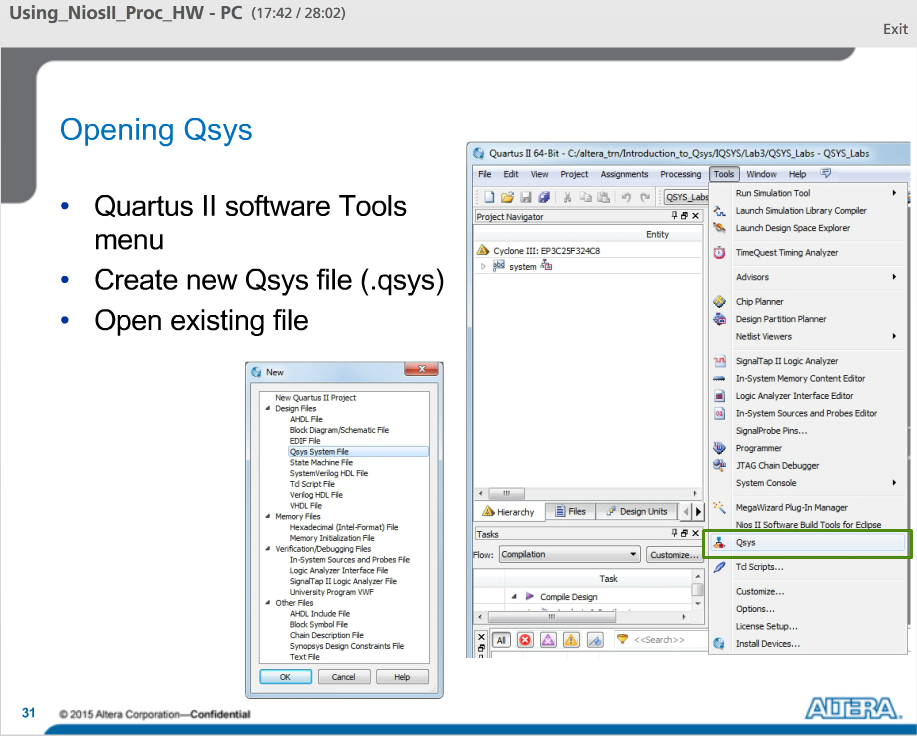
The Avalon-ST protocol is a standard protocol used for high-speed data transfers. It supports unidirectional, point-to-point transfers. It is fully synchronous and is flexible enough to support both simple and more complex transfer requirements.

The big benefit of the Avalon-ST protocol is its high bandwidth and low latency.

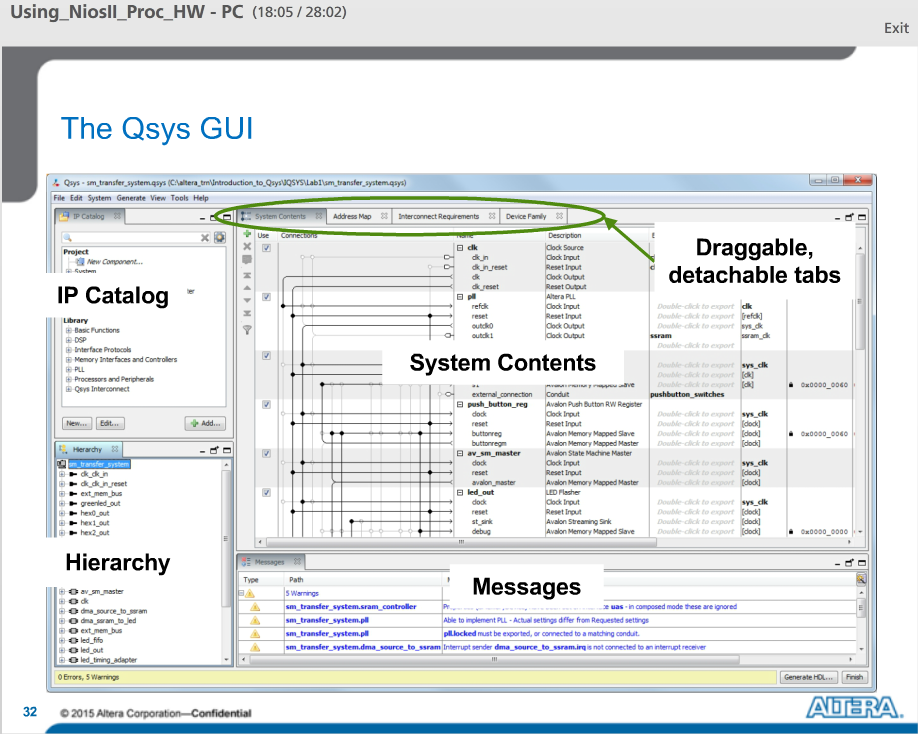
It is important to remember that Avalon ST and MM interfaces are complimentary, in that MM is used to define the control logic while ST is used for the data path.



The Avalon Interface Specification defines all of the interfaces discussed here.



You can launch the Qsys tool from the Quartus II tools menu, by creating a new qsys file from the file menu or opening an existing Qsys project file.

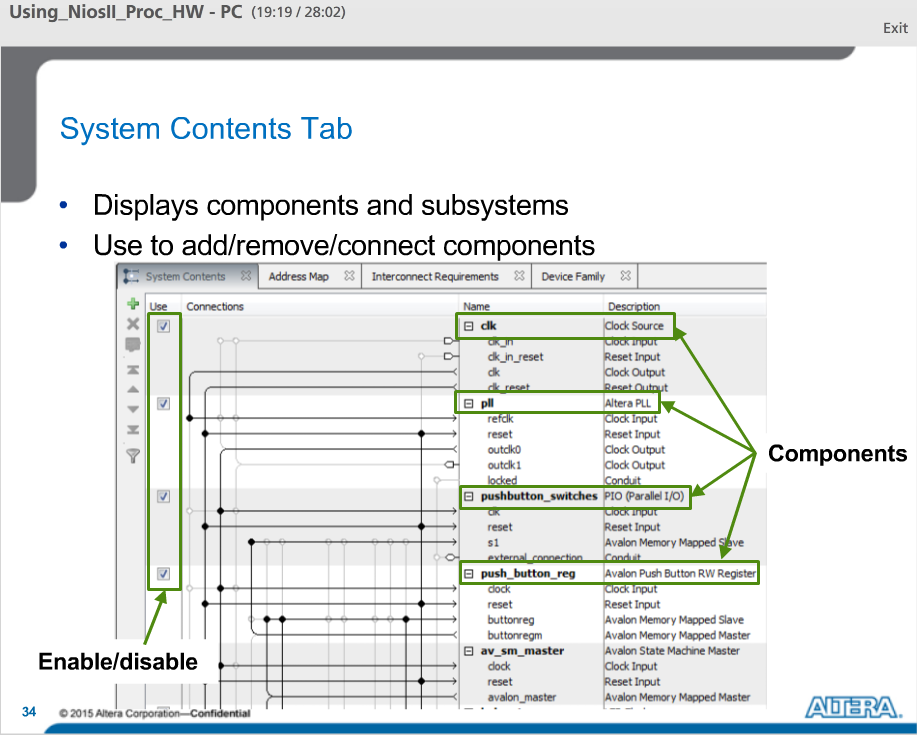


Here is what the Qsys tool looks like.

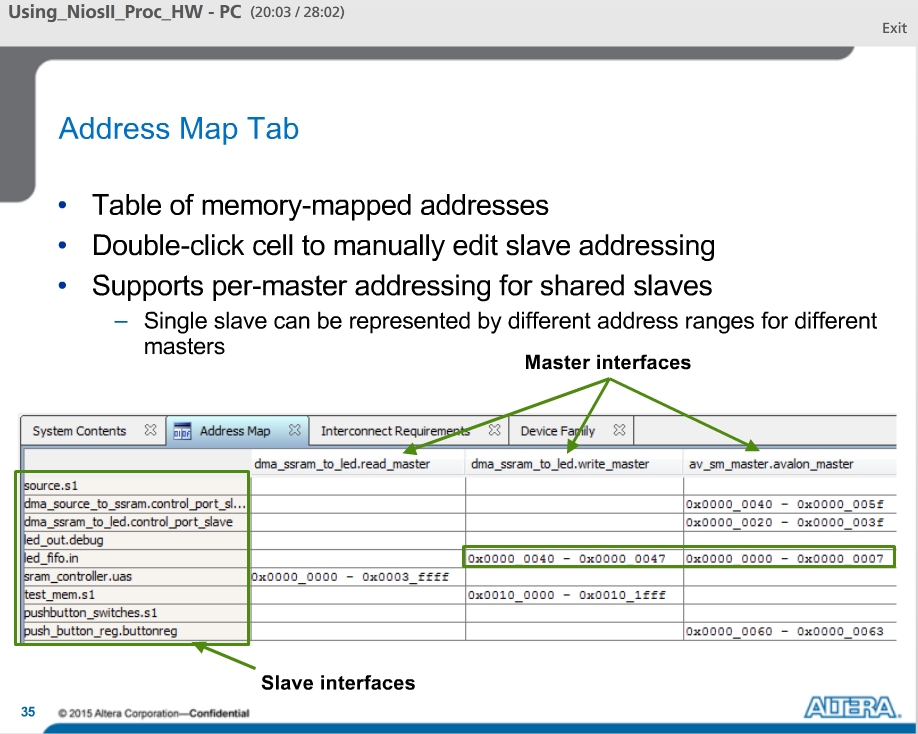


The IP catalog lists all of the available IP and subsystems that are available to add to your current Qsys system.

The component library is organized by categories to make it easier to find the IP you need.

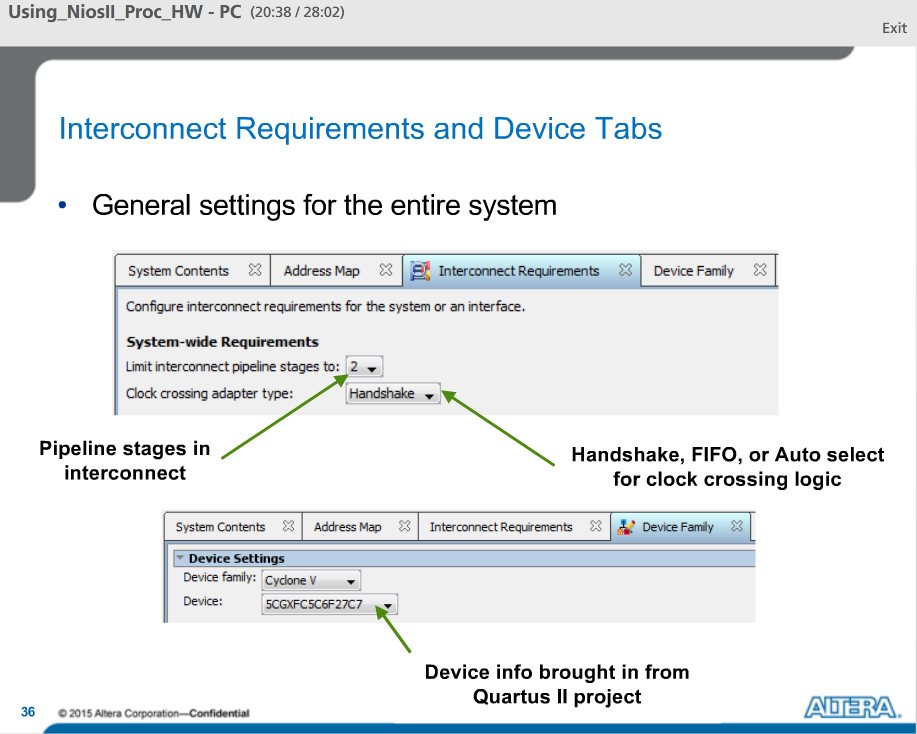


The System Contents tab displays all of the components and subsystems that have been added to the current system. Use this window to easily manipulate a component’s relationship to the rest of the system, including connections to other components, connections external to system and location in address space.



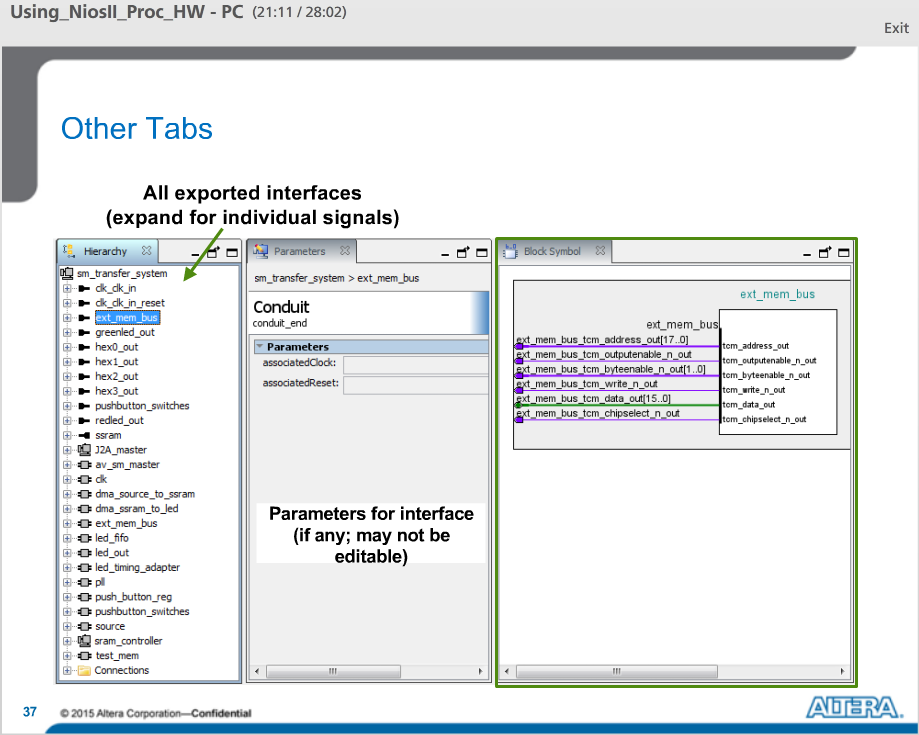
The Address Map tab contains a table of all memory mapped device addresses, making it easier to learn the address mappings and address spans in your system. Here each memory mapped master is indicated by a column and each memory mapped slave is indicated by a row. If they are connected, then the cell representing their intersection is filled with the slaves address as that master port sees it.

You can double-click on a cell and edit the address for that slave. If connected to multiple masters, the slave address is only changed for the master in whose column the cell is located.



On the Interconnect Requirements tab, you can choose options that affect the implementation of your Qsys interconnect.

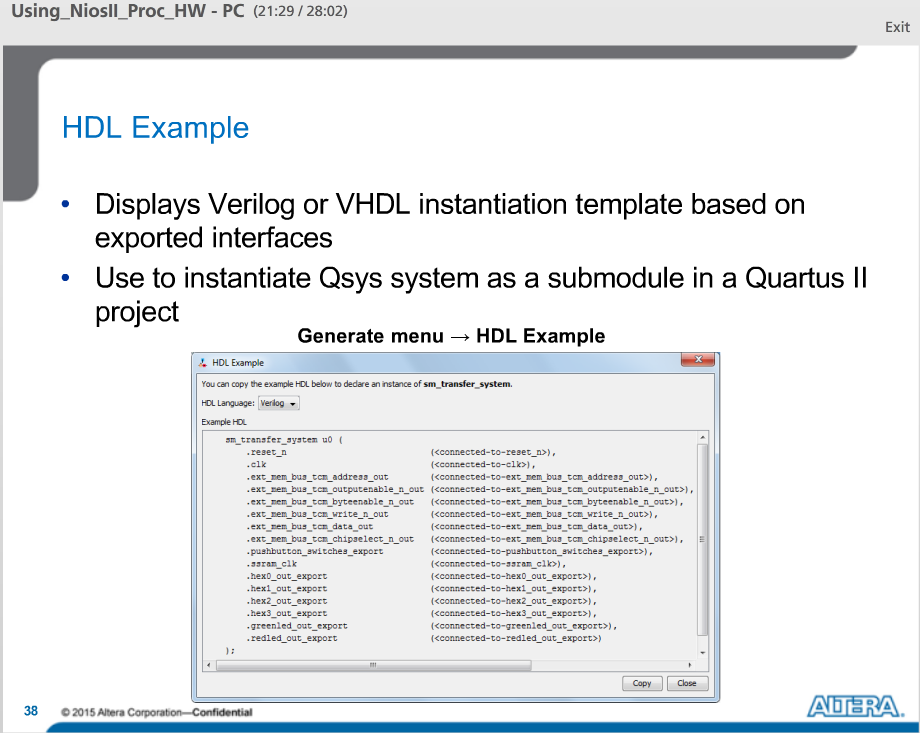
On the Device tab, you can see the device that was imported from Quartus II project or can choose a different target device.



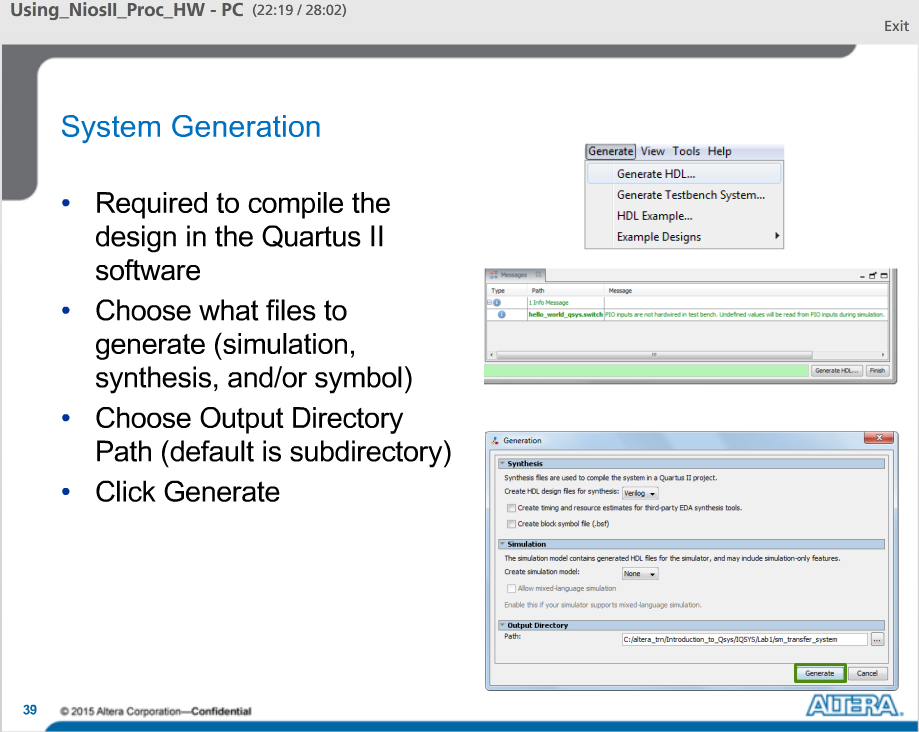
The Hierarchy Tab is a good way of viewing details about the system. This includes interface parameters, component parameters, connections etc. By selecting a particular component, you can see the parameter editor settings chosen when that component was added to the system.

The Parameter Tab also appears when a user double-clicks on a Qsys component in the System View Menu. Users can sometimes change the parameters of that component using the parameters tab.

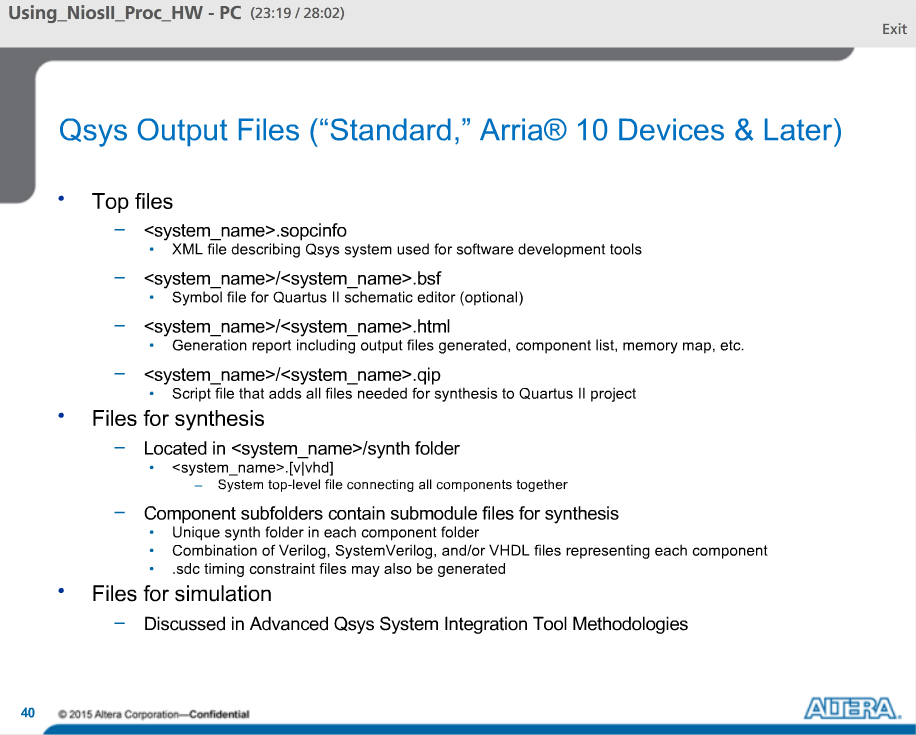
The block symbol tab shows a graphical view of component’s top level signals.



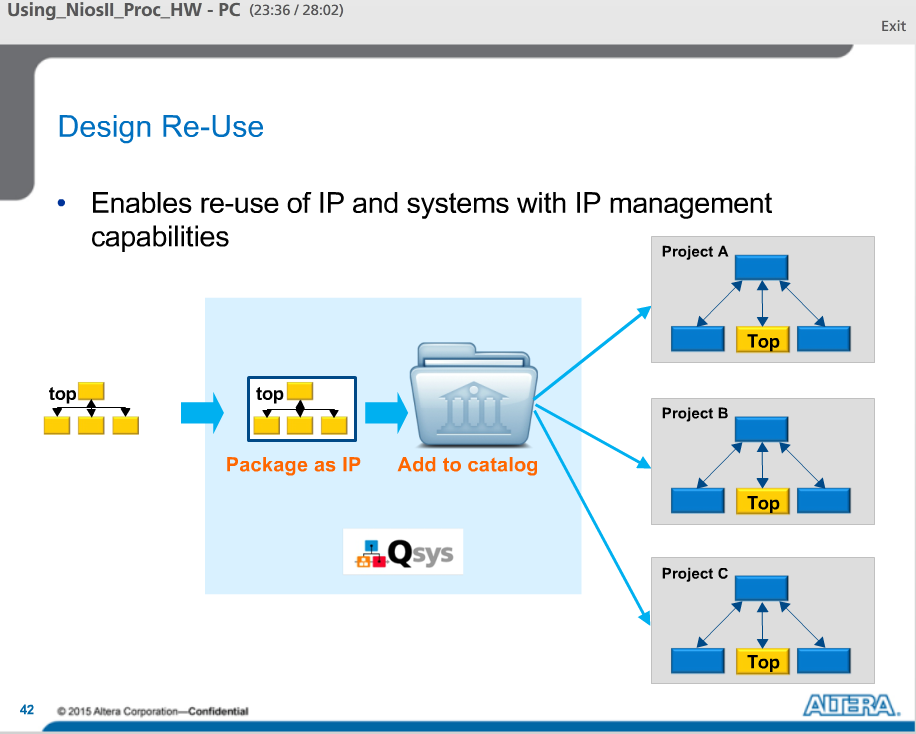
The Qsys HDL Example provides an instantiation template to allow a designer to instantiate a Qsys System as a sub-module in another design file.



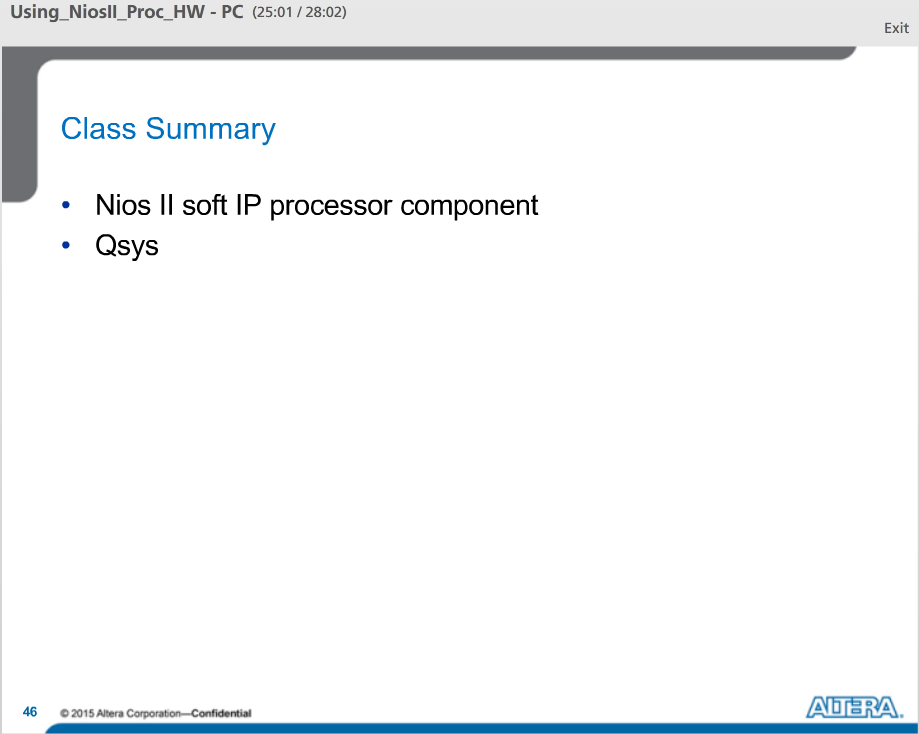
In order to build an FPGA using a design, a designer must generate HDL files from Qsys for use in the Quartus II Compilation.



Here are the files generated by Qsys after you press the generate button.



Qsys improves design re-use by allowing designers to package their entire Qsys system as an IP and adding it to their library for others to use.



In this lecture, we talked about the features and functionally of the Nios II processor IP component as well as some example applications.

We also discussed the Qsys System generation tool, the Qsys interconnect and how to use Qsys to create embedded systems using the Nios II processor.