Peter the Great St. Petersburg Polytechnic University
Institute of Computer Science and Cybersecurity
Graduate School of Computer Technologies and Information Systems

Lecture: Hardware Design Flow for ARM-based System on Chip

Subject: Automation of discrete device design (in English)

Completed by student of group 5130901/10101 _____ Nepomnyaschiy M.T.
(signature)

Lecturer _____ Antonov A.P.
(signature)

Saint Petersburg
2024

Table of contents

List of illustrations

Introduction

This lecture provides some information about the System Development Flow for FPGA-based Systems. It explores the design and development processes using Field-Programmable Gate Arrays (FPGAs), renowned for their flexibility across various applications.

2. SoC Overview

The first part focuses on introducing the components and flow of system development, emphasizing the hardware aspect. It covers tools like Quartus for design synthesis, Qsys for system integration, and discusses the integration of hard processor systems with FPGA fabric. The aim is to ensure familiarity for both traditional FPGA designers and ARM software engineers, maintaining their comfort within the respective design environments.

2.1. Hardware Development Perspective

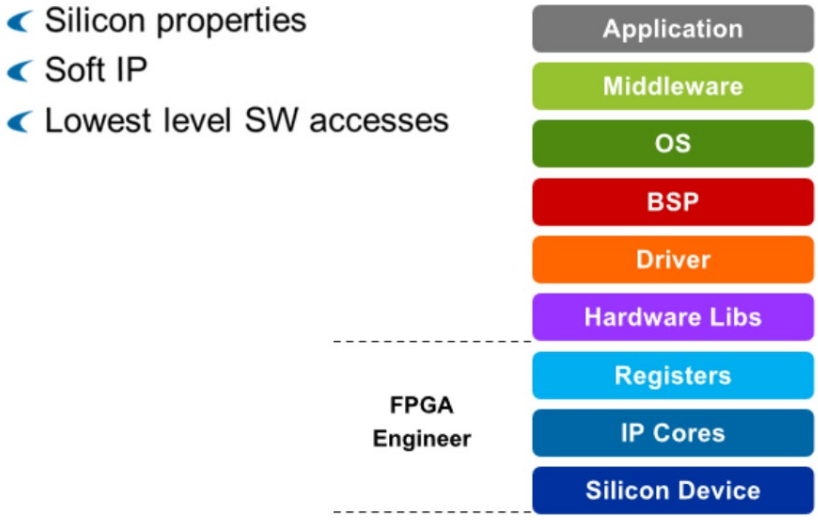

Figure 1 – Hardware Development Perspective

In system development, there are different layers. At the foundation are hardware components like silicon devices. Inside these devices, there are IP cores that do specific tasks. Above that, we have a control and status register view of these components. Moving into the software side, there are hardware libraries that help firmware developers by hiding some technical details. Then, there are drivers for specific IP components.

Beyond that, we have a Board Support Package, an operating system, and middleware like Ethernet stacks. Finally, at the top, there's the user application software.

For FPGA engineers, we focus on the bottom layers: the devices where IPs are implemented, the IPs themselves (including soft IPs in FPGAs), and the register view that allows the lowest level of software access.

2.2. System Development Flow

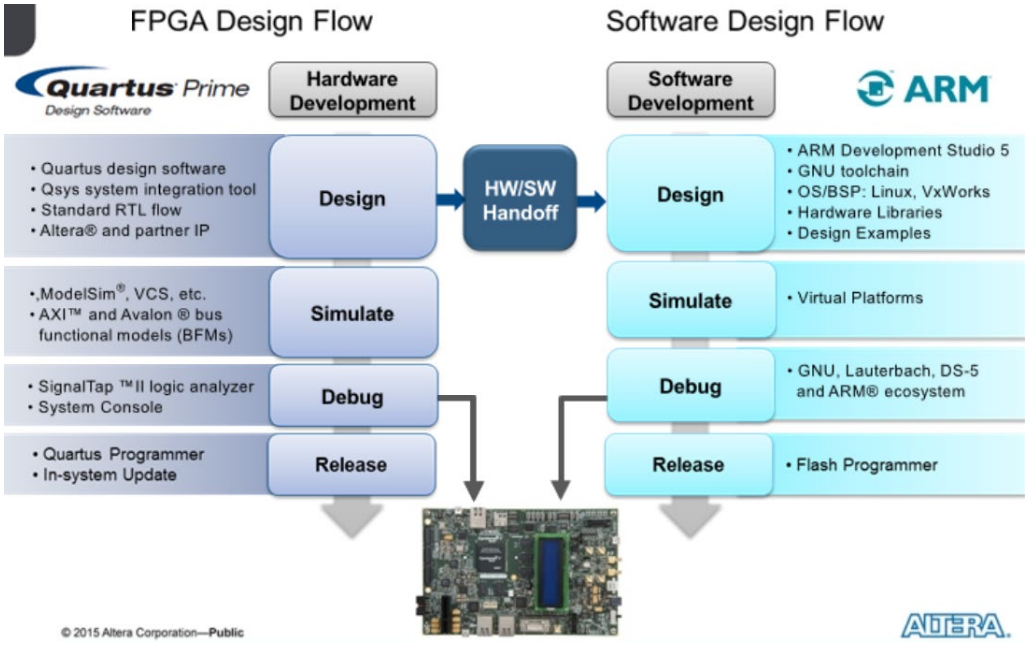

Figure 2 – System Development Flow

In hardware development, we focus on the features in the left column of the system development flow. The Quartus design software is our starting point, used for synthesizing, placing, and routing RTL (Register Transfer Level) source files.

Next, the Qsys system integration tool comes into play. It generates RTL sources and allows for the quick assembly of various IPs (Intellectual Properties), automatically creating synthesizable interconnects.

After creating the hardware system, functional simulation becomes crucial. Bus functional models, supported by Qsys, make this process easy.

For hardware verification, FPGA tools such as Signal Tap II embedded logic analyzer or system console are used. These tools can also be utilized in the SoC (System-on-Chip) environment.

Finally, the FPGA hardware is programmed into flash for release. Qsys generates handoff files to simplify the creation of second-stage boot software and board support packages, making the software design flow smoother. Although we'll briefly touch on these tools, they're the main focus of another online training.

2.3. SoC System Architecture

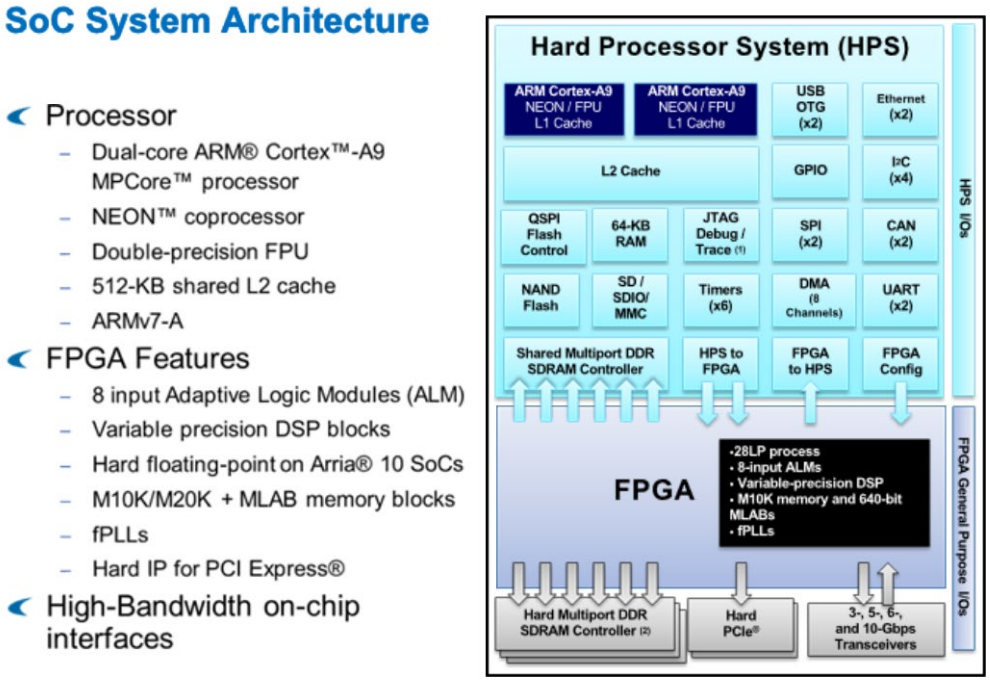

Figure 3 – SoC System Architecture

The SoC device consists of two main parts: the Hard Processor System (HPS) shown in blue and the FPGA fabric shown in green.

The HPS contains a dual-core processor, each with its own floating-point unit and NEON coprocessor, along with shared L1 caches. These processors share a 512-KB L2 cache and various peripherals, including Ethernet ports, USB ports, flash memory interfaces, and general-purpose embedded peripherals like UART, SPI, CAN, and I2C. The HPS also features a multiport SD DRAM controller supporting DDR2, DDR3, and LPDDR2 with ECC.

Fast interfaces connect the HPS and FPGA fabric, allowing for efficient communication. Additionally, there's a direct connection from the FPGA to the SDRAM controller in the HPS. The processor can even control the FPGA configuration directly through software.

The FPGA fabric includes typical components and offers additional hard IP functions for custom logic, such as multiport SDRAM controllers, PCI Express ports, and high-speed serial transceivers. Both the HPS and FPGA have dedicated I/O, though they can share some I/O functions when needed.

2.4. 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, SD/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

Figure 4 – HPS IP Features

The Hard Processor System (HPS) includes more than just processors. It features dual ARM Cortex-A9 processors with additional components. These include a hard SDRAM controller, a general-purpose DMA controller from ARM, Ethernet controllers from Synopsis, multiple flash interfaces, and various serial interfaces like USB, SPI, I2C, and CAN.

Moreover, the HPS houses UARTs and GPIO interfaces, enhancing its functionality and connectivity.

2.5. SoC Design Methodology Goals

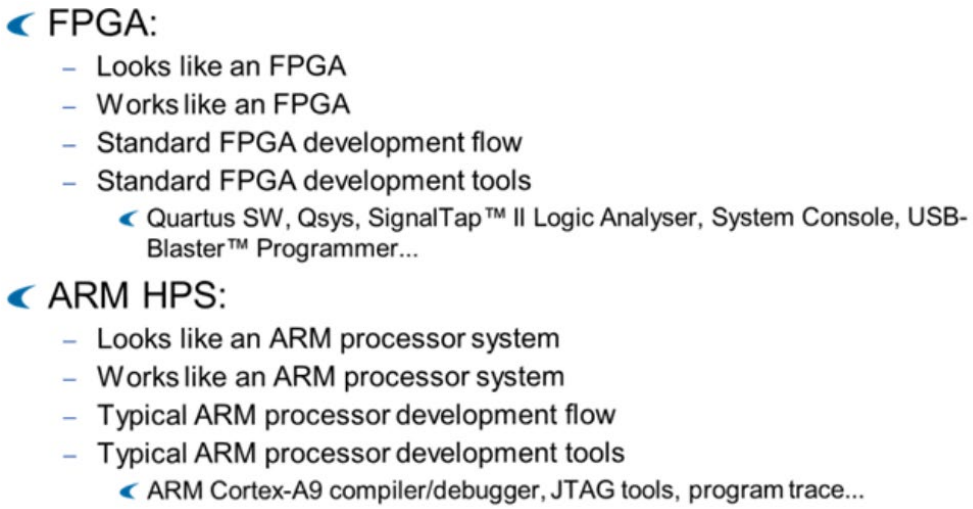

Figure 5 – SoC Design Methodology Goals

As an FPGA designer going through this training, you'll find the hardware portion very familiar. The goal in designing these devices was to ensure that both traditional FPGA designers and ARM designers feel at home.

For FPGA engineers, the integration of the hard processor core won't impact their normal FPGA environment. All the usual development and debugging tools remain available, ensuring a seamless experience.

Similarly, for ARM software engineers, the typical ARM environment they're accustomed to is the same one they'll use to develop for the embedded ARM SoC. It's designed to be transparent, as if working with two separate chips.

In essence, the integration of the hard processor core into the FPGA design environment aims to maintain familiarity and ease of use for both FPGA and ARM designers.

3. Quartus and Qsys Tool Flow

Now that we've seen what the SoC devices are all about, lets dive in to the Hardware design flow, and we'll first examine Quartus and Qsys tool flows including HPS component instantiation and custom components that the Hard processor system interacts with.

3.1. SoC Design Flow

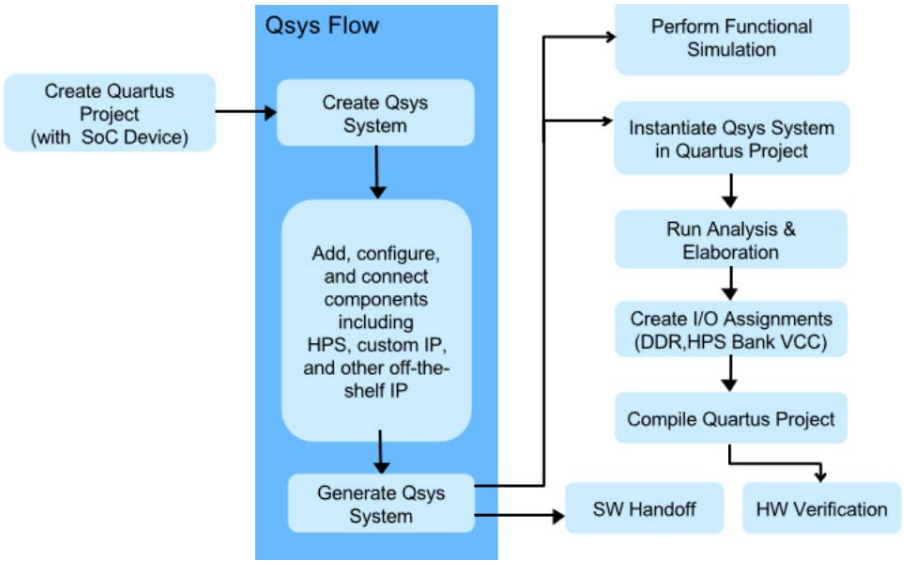

Figure 6 – SoC Design Flow

The Quartus and Qsys design flow for SoC devices involves several steps. First, you start with a Quartus project. Within this project, you create a Qsys system that includes the Hard Processor System (HPS), custom IP, and other off-the-shelf IP available from Altera or third-party vendors.

Next, you generate the system, which creates simulatable or synthesizable HDL versions of the system depending on the generation options chosen.

Software handoffs generated from this process go to the firmware engineer. The generated Qsys system can then be functionally simulated or included in the Quartus project.

In Quartus, you run analysis and elaboration to allow the tool to understand the hierarchy of the design. You then create IO assignments using a script generated by Qsys and manually set IO assignments for other general-purpose FPGA IOs.

Finally, you run a full compile. Once an FPGA programming bitstream is completed, you can perform hardware verification before product release.

3.2. Create Quartus Project for a SoC Device

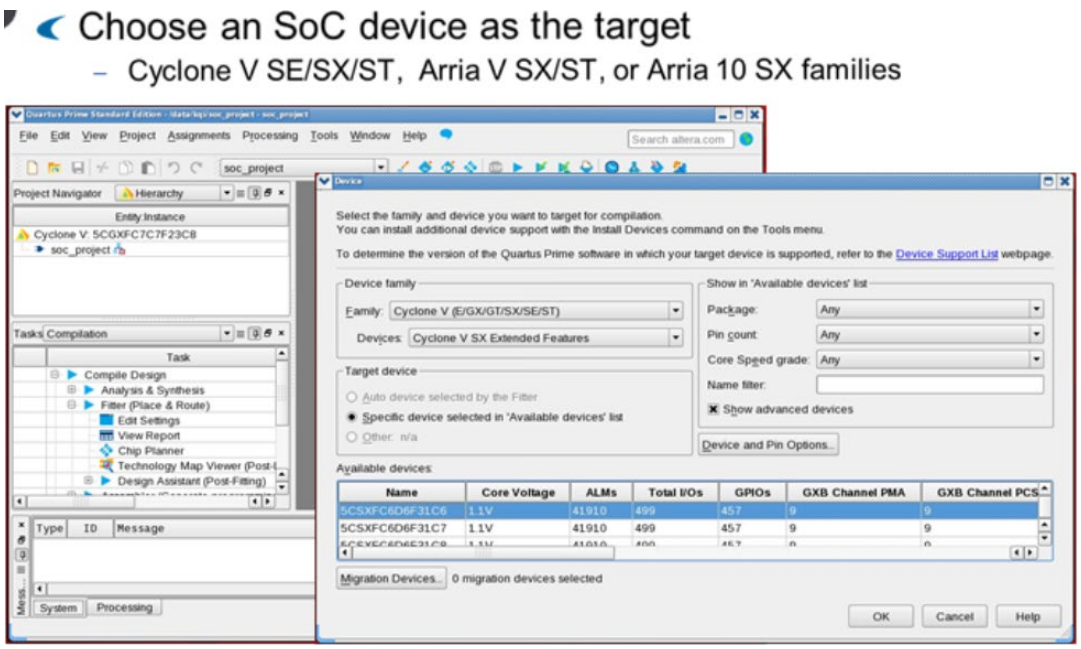

Figure 7 – Create Quartus Project for a SoC Device

To create a Quartus project, you typically use the New Project Wizard, which guides you through the process. A Quartus project is essentially a collection of all source files and settings related to a specific design.

When creating the project for use with an SoC device, it's crucial to select a family that includes the Hard Processor System (HPS). These families usually start with an 'S'. There are several subfamilies within these families that contain the HPS.

For example, in the Cyclone V family, you'll find variants like SE, SX, and ST. In the Arria V family, there are subfamilies like SX and ST. Lastly, in the Arria 10 family, the SX subfamily includes the HPS.

Selecting the appropriate family and subfamily ensures compatibility with the SoC device and enables you to utilize the Hard Processor System effectively.

3.3. Qsys System Integration Tool

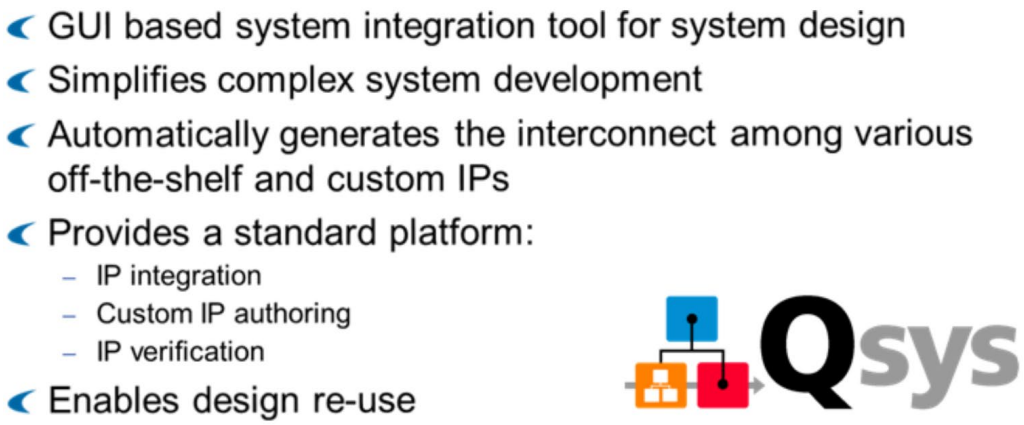

Figure 8 – Qsys System Integration Tool

After creating the SoC project in Quartus, you use the Qsys tool to connect all the important parts like the HPS and other components. Qsys saves time by automatically connecting these parts. It works by making sure all components follow a standard way of communicating. This makes it easy for Qsys to connect them together. You can use Qsys to add new parts, create custom ones, and check that everything works. It uses special technology to make things work faster and better on the FPGA chip.

3.4. Start a new System in Qsys

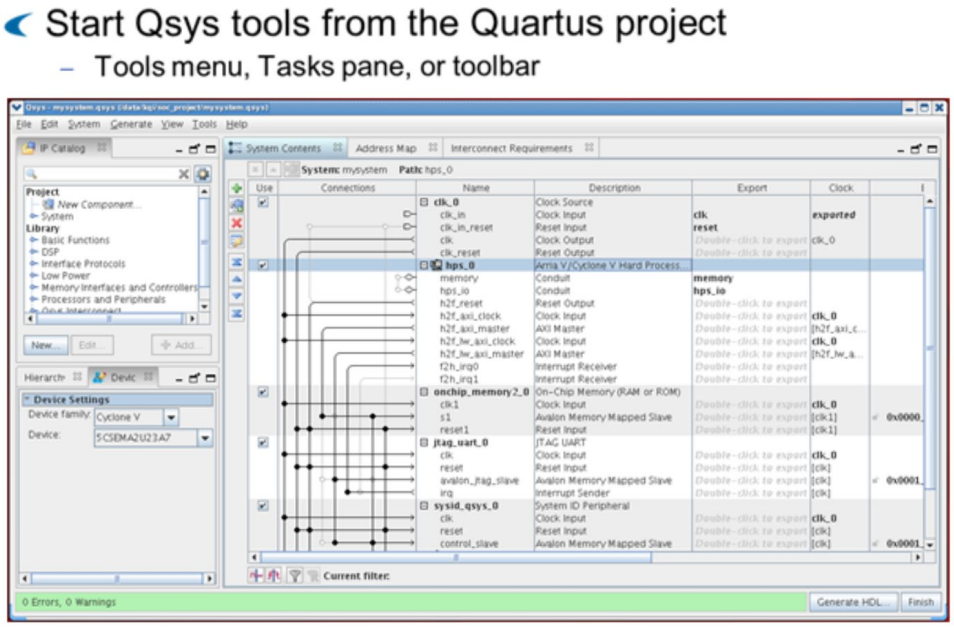

Figure 9 – Start a new System in Qsys

To begin using the Qsys tool, you can find it in the Quartus tools menu, task pane, or toolbar. When you open Qsys, you'll see a window where you can add different IP components, including the HPS or custom IP, from the component library pane.

In the System contents tab, you can connect these components together using compatible interfaces. Once you've made all the connections, you can move to the generation tab.

In the generation tab, you can generate the necessary HDL (Hardware Description Language) and other supporting files for the system. This includes everything needed to implement the design on the FPGA.

3.5. Add Components to Qsys System

- All IP including HPS, off-the-shelf IP, and custom components located in the IP Catalog pane of the Qsys tools
- Use search feature to easily find IP
- Double click or click add to instantiate the component

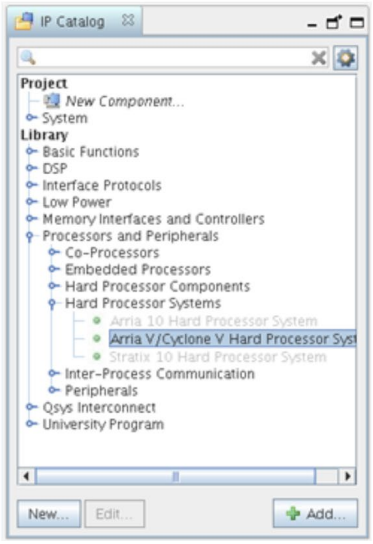

Figure 10 – Add Components to Qsys System

In the Qsys tool, the Component Library lists all available IP and subsystems that can be added to your current Qsys system. Off-the-shelf IP are listed under the Library section, while subsystems and any custom IP or components available are listed under the Project section. The Component Library is organized by categories to make it easier to find the IP you need. You can expand categories and subcategories and highlight the component you want to add to your system. To add a component, simply double-click on the component name or click on the Add button with the component highlighted. You can also use the search field to filter the component library by name. As you type, Qsys dynamically filters the list, making it easier to find what you need. For example, the Hard Processor System component can be found inside the Embedded Processors folder of the component pick list.

3.6. Connect the Components

- Click the appropriate dots in the connections column
- Alternatively right click on a specific interface to see and make connections

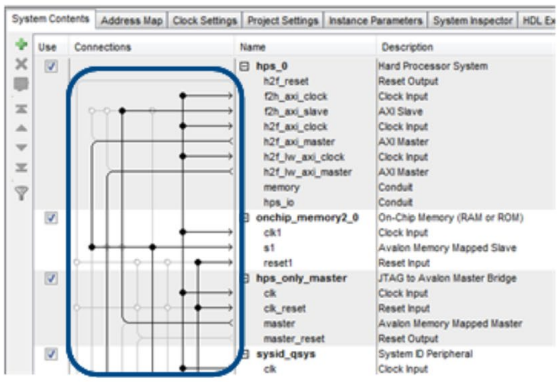

Figure 11 – Connect the Components

In the System Contents tab of Qsys, after you've added components, you need to make connections between them. There are two main ways to do this.

First, you can use the Connections column. Compatible connections are shown in gray with a hollow circle. Clicking on a hollow circle will darken the connection, indicating that a particular set of interfaces is connected.

If you have many components and it's hard to find the correct connections, you can right-click on a particular interface. This will show you a list of legal connections, and you can choose the one you need to make the connection.

3.7. HPS FPGA Interface Options

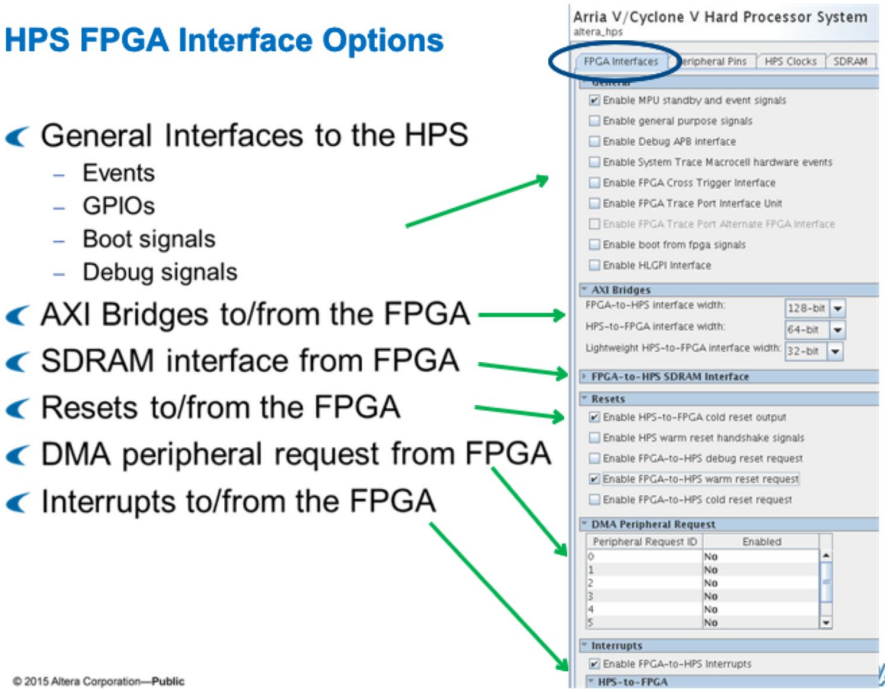

Figure 12 – HPS FPGA Interface Options

When configuring the HPS, most options are related to its interfaces. The first tab of the HPS configuration focuses on the interface to and from the FPGA.

In the general section, you can enable options to send events to and from the FPGA, direct GPIOs to and from the FPGA, enable boot signals from the FPGA, and connect debug interfaces to the HPS, such as Cross Trigger, System Trace Macrocell, and Debug APB bus interfaces.

You can also enable the three bridges connecting to the FPGA side: FPGA to HPS bridge, HPS to FPGA bridge, and the latency-sensitive lightweight HPS to FPGA interface. For high-performance bridges, you can set the bridge width from the FPGA side.

If you choose to enable and configure direct FPGA access of the HPS SDRAM, it allows the fastest data transfer from the FPGA side.

In the reset section, you can allow the FPGA to reset the HPS or have the HPS send reset signals to the FPGA.

In the DMA peripheral request section, you can enable peripheral request signals to allow FPGA components to request DMA transfers using the ARM DMA inside the HPS.

Lastly, you can enable peripheral interrupts to and from the FPGA to the HPS.

3.8. Peripheral Pins Options

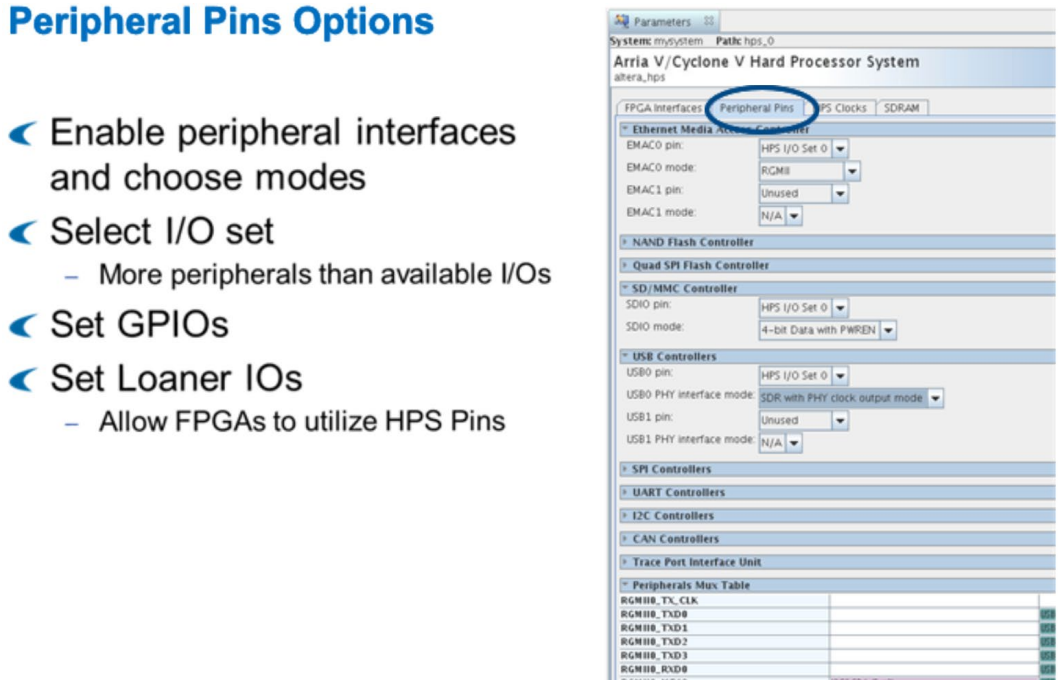

Figure 13 – Peripheral Pins Options

In the IO interface settings, specifically on the Peripheral Pin tab, you configure pin assignments for specific peripherals. Here, you enable peripheral interfaces and in some cases, select modes for the peripherals.

For example, with the QSPI peripheral, you can select 1, 2, or 4 chip select outputs. Each peripheral connects to at least one pin mux, which is controlled by selecting a specific HPS I/O set. Choosing an IO set determines which I/O pins will be used for this interface.

If you need more details on which pins can be used for a specific peripheral, you can refer to the Pin Connection Guidelines for the SoC Devices.

For Arria 10 devices, the GUI may look different because most IOs are shared between the FPGA and HPS. However, the concept remains the same: you need to assign modes and pins to the peripherals.

If there's a conflict during pin assignments, the Qsys tool will show an error, and you'll need to ensure the HPS component is error-free before generating the system.

3.9. HPS Clocks

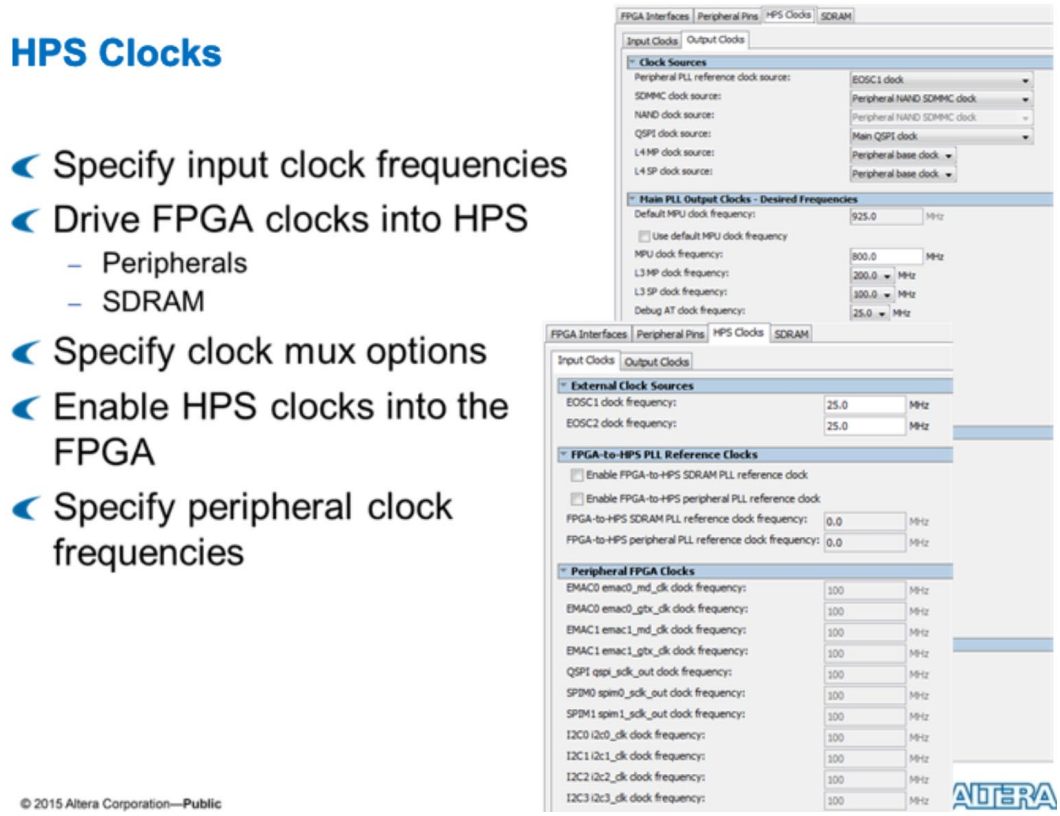

Figure 14 – HPS Clocks

In the HPS Clocks tab, you can adjust all options related to the clock. These settings are transferred to the 2nd Stage bootloader software as part of the HW-SW handoffs.

Here, you can set the frequencies of the processors and peripherals. The frequencies of the PLLs are set by software using specific API calls available in the software library.

You can also create connections for FPGA clocks to drive the HPS for use by the peripherals and SDRAM. Additionally, you can select clock sources and send HPS clocks into the FPGA.

It's often a good idea to synchronize clocks between the HPS and the FPGA to speed up data transfer. Without synchronization, a clock crossing bridge must be implemented, which can impact the performance of the bridges.

3.10. Cyclone V / Arria V SDRAM Configuration

- Consistent with Altera SDRAM controller configuration GUI
- Supported Memories:
  - DDR3
  - DDR2
  - LPDDR2
- Configure clock & initial settings

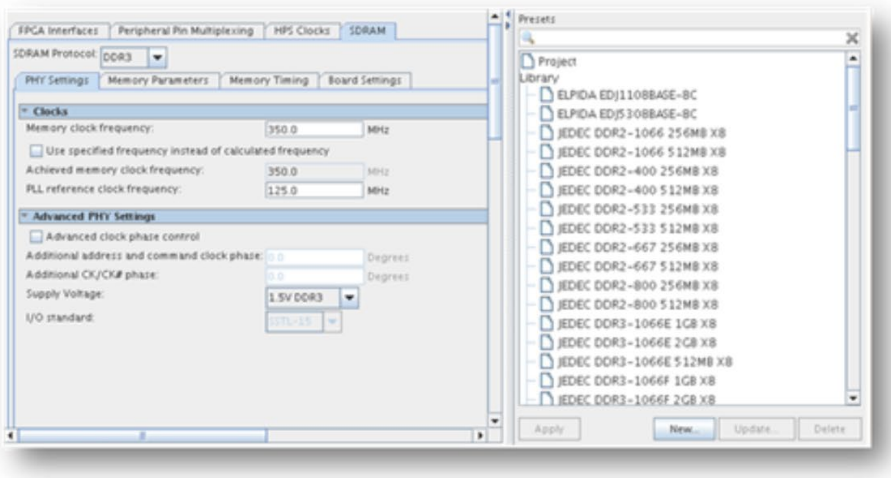

Figure 15 – Cyclone V / Arria V SDRAM Configuration

In the HPS Clocks tab, you control all clock-related settings. These settings are passed to the 2nd Stage bootloader software during hardware-software handoffs. You can adjust the frequencies of processors and peripherals here, while PLL frequencies are managed through specific software API calls.

This tab also allows you to connect FPGA clocks to drive the HPS for use by peripherals and SDRAM. You can select clock sources and send HPS clocks into the FPGA. It's beneficial to synchronize clocks between the HPS and the FPGA to accelerate data transfer. Without synchronization, a clock crossing bridge must be implemented, affecting bridge performance.

3.11. Arria 10 SDRAM Parametrization

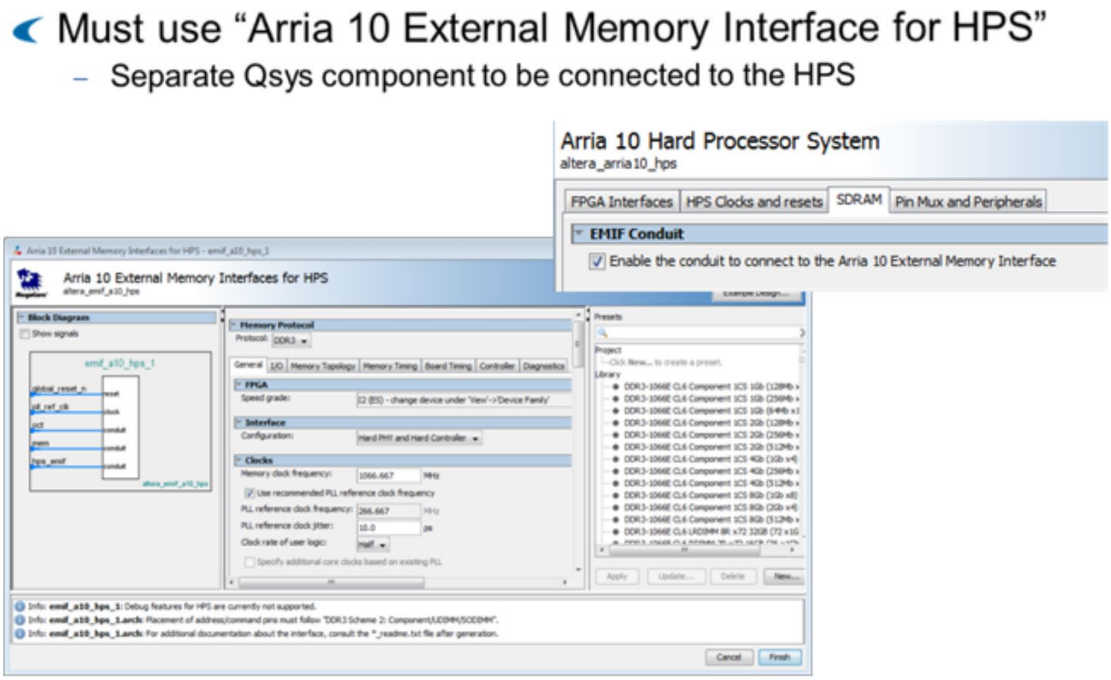

Figure 16 – Arria 10 SDRAM Parametrization

For Arria 10, the instantiation of the SDRAM differs slightly because the external memory controller resides within the FPGA. Instead of directly configuring the SDRAM through the second stage bootloader (SSBL), it's managed by a separate component called the Arria 10 external memory interface for HPS. This component can be connected to the Arria 10 HPS using a dedicated conduit connection in Qsys.

The Arria 10 SDRAM is configured by the FPGA bitstream. While the bitstream can be loaded by the SSBL, the SDRAM itself isn't configured directly by the second stage bootloader.

3.12. Generate Completed System

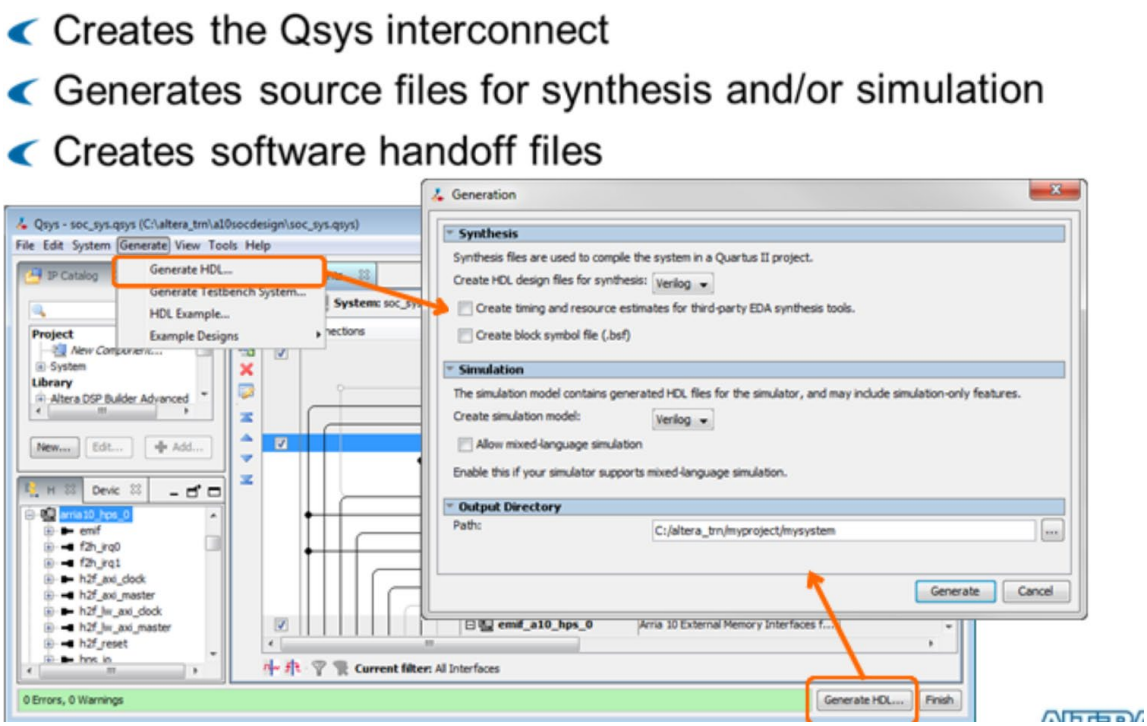

Figure 17 – Generate Completed System

Once you've completed configuring the HPS system, you can generate the necessary HDL (Hardware Description Language) files. Depending on your choices, you can generate files for simulation, testbench, or synthesis, in either Verilog or VHDL format. These files will be stored in the appropriate directory.

In addition to the HDL files for the system and the interconnect, the generation process also creates software handoff files. These files are used by the software debugger or as input for the Linux device tree. We'll discuss these further later on.

3.13. Hardware/Software Handoff Files

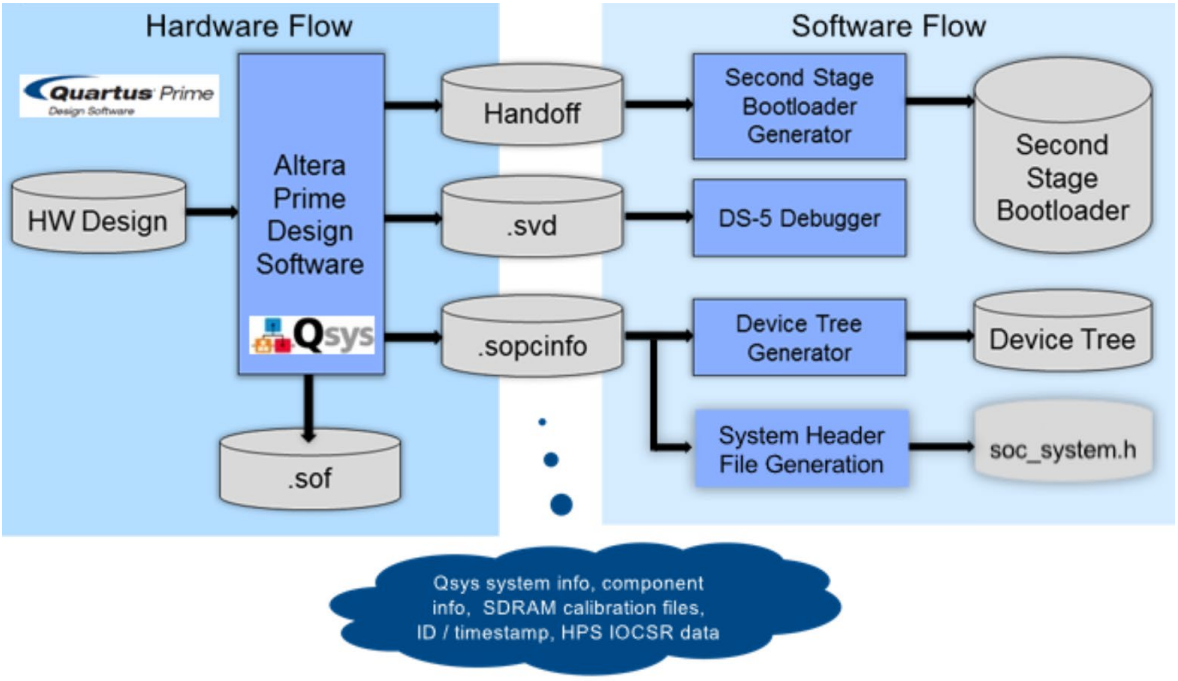

Figure 18 – Hardware/Software Handoff Files

Here we see the different files generated by Qsys and Quartus and how they are used by the software tools.

- SOF FPGA Bitstream: This file is generated by Quartus and is used to configure the FPGA portion of the SoC. It contains the configuration information needed to set up the FPGA hardware.
- Handoff Directory: Quartus creates this directory for use by the Second Stage Bootloader Generator. The Second Stage Bootloader generates software source files necessary to configure clocks, IO pin mux, and SDRAM. Depending on the family, these configurations can be done via software or FPGA image.
- .svd File: This file is a description of soft IP peripheral registers in the FPGA. It functions similarly to symbol files in the software debugger, providing access to hardware registers directly from the ARM software debugger. Each FPGA peripheral, whether custom or off-the-shelf, needs its own .svd file. During Qsys generation, these files are aggregated. They can then be imported into ARM DS5 debuggers, allowing you to see FPGA registers in the debugger register view along with associated help on functionality.
- .sopcinfo File: This file is output by Qsys and is used by the Device Tree generator. The Device Tree standard specifies hardware connectivity so that the Linux kernel can load drivers without rebuilding the kernel. This makes the Linux kernel flexible regarding changing FPGA hardware. The .sopcinfo file can also be used by the system header file generation tool to create soc_system.h, which contains MACROS representing the FPGA Qsys system.

In summary, these files play crucial roles in configuring and debugging the SoC system, providing necessary information for software tools to interact with the FPGA hardware effectively.

4. Custom Components

Now that we under stand basic Quartus and Qsys tools including the HPS component, now, lets take a look at how implement custom components in the FPGA that interacts with the HPS.

4.1. Qsys-Supported Standard Interfaces

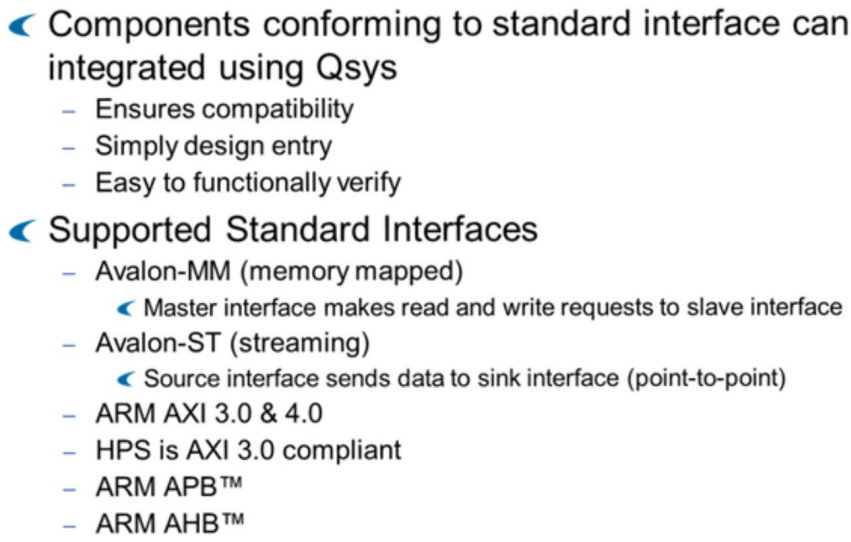

Figure 19 – Qsys-Supported Standard Interfaces

Qsys serves as a powerful tool for integrating various components, but these components must adhere to supported standard interfaces. There are numerous benefits to conforming custom IP to a standard interface. Firstly, it ensures compatibility with other components, simplifies design entry through tools like Qsys, and facilitates easy verification.

Qsys supports several standard interfaces, including:

1. Avalon memory-mapped interfaces: These interfaces involve a master requesting reads and writes from a slave.
2. Avalon streaming: Designed for point-to-point high-throughput applications.
3. ARM AMBA interfaces: These include AXI 3, AXI 4, APB, and AHB. Notably, the HPS bridges are AXI 3.0 compliant.

If your custom component conforms to one of these standards, it can be seamlessly integrated with other IP using Qsys tools. This ensures smooth interoperability and efficient system development.

4.2. Avalon or AXI

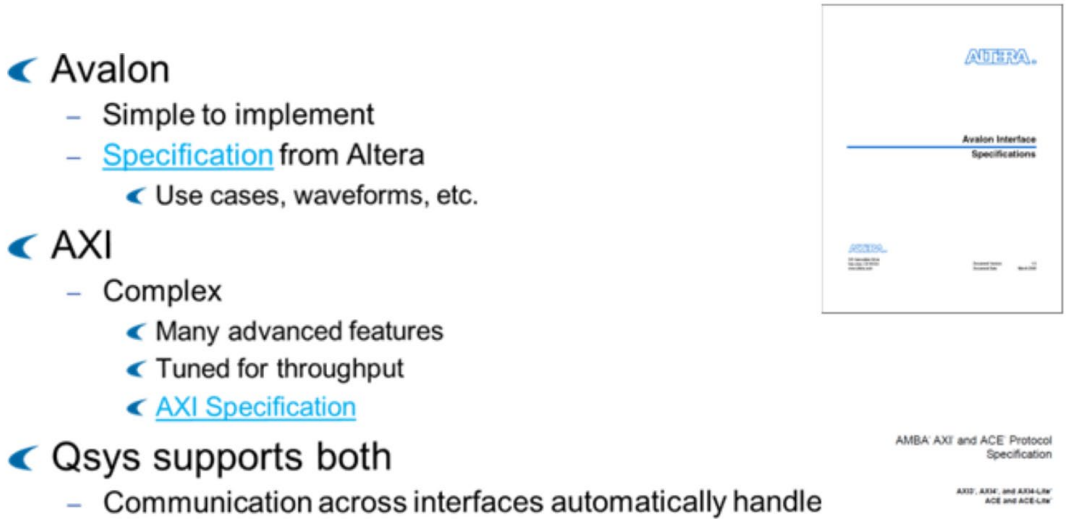

Figure 20 – Avalon or AXI

Before choosing an interface standard, it's important to understand it well. You can find specifications for Avalon (by Altera) and AXI (by ARM). Avalon is simpler, while AXI is more complex and optimized for speed.

Qsys supports both standards, and it can handle communication between them. You can use AXI with its advanced features or stick with Avalon for simplicity.

Ultimately, the choice depends on your needs. If you want something straightforward, go with Avalon. If you need more advanced features like security, AXI is better.

4.3. Add Custom Components to Qsys

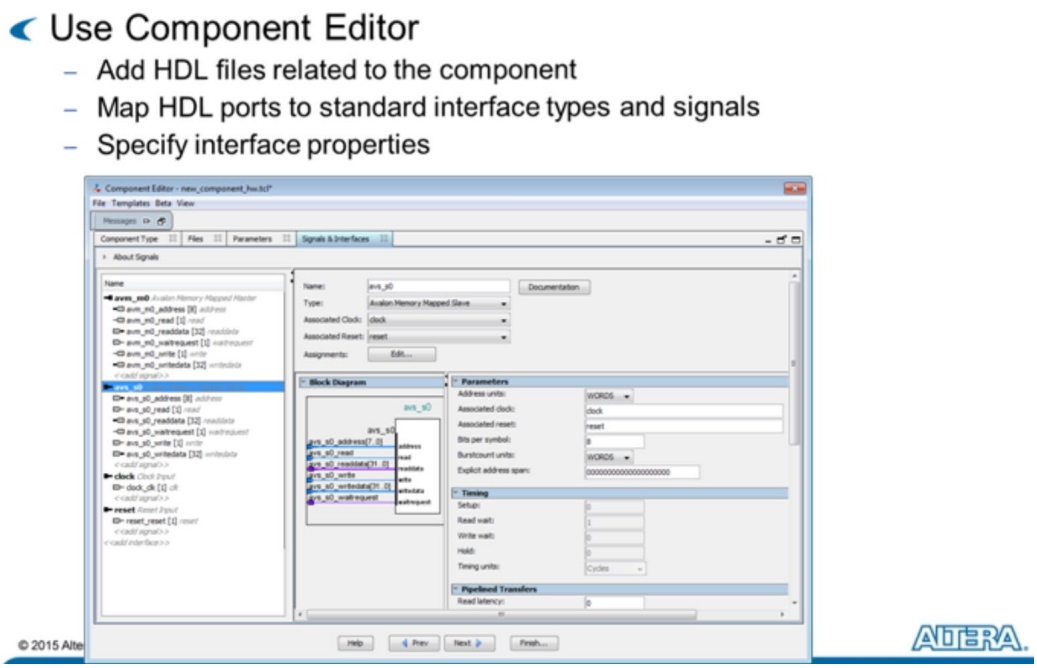

Figure 21 – Add Custom Components to Qsys

Before you can use custom components in Qsys, you must create them. The simplest method is to use the Component Editor. In this tool, you define the HDL (Hardware Description Language) for your component. You can also specify instantiation parameters, similar to Verilog parameters or VHDL generics, to make each instance of your IP different, enhancing its reusability.

However, the most critical task of the Component Editor is to map the HDL ports to standard interface types and signals, and then define the properties of these interfaces. Once Qsys recognizes the interfaces, it can connect your custom component to others, providing the necessary components to bridge the interfaces.

4.4. Visualization of FPGA

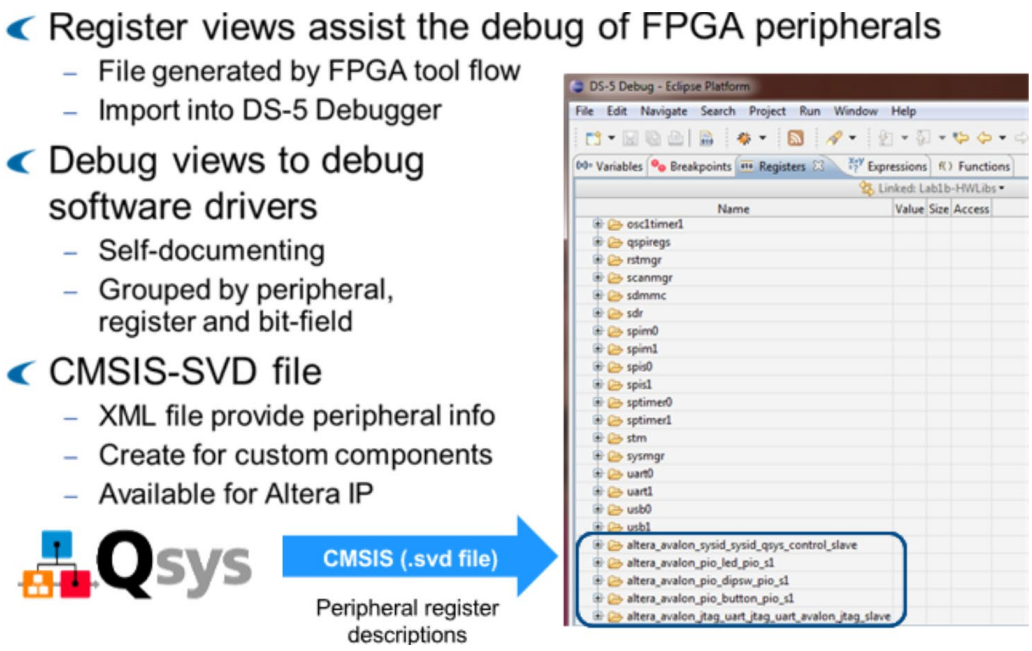

Figure 22 – Visualization of FPGA

One important consideration when creating custom components is to include a System View Description (SVD) file. This file describes the register set of your custom FPGA peripheral to the ARM DS5 debugger. With this file, the debugger can view and modify custom IP in the FPGA fabric.

The SVD file, formatted in XML, is available for off-the-shelf Altera IPs, and it can be created for custom components as well. This ensures self-documenting and easy debugging, enhancing the overall development process.

Conclusion

This lecture has covered essential aspects of FPGA system development, from designing custom components to integrating hardware and software. Adhering to standard interface protocols and including System View Description files can streamline the development process and ensure system reliability.