

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

Lecture: Creating a System Design with Platform Designer

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

Оглавление

Creating a System Design with Platform Designer	4
Introduction to SystemVerilog Complexity	4
4. Part 4 – Connection Details	4
4.1. Clocks	4
4.2. Clock Domains and Domain Filtering	5
4.3. Clock Domain Crossing	6
4.4. Reset Control Options	7
4.5. Reset Connection Points	8
4.6. Reset Domain Filtering	9
4.7. Reset Adapter Insertion	10
4.8. Conduit Interfaces	11
4.9. Memory-Mapped Addressing	12
4.10. Address Map Tab	13
5. Part 5 – Review Connections	14
5.1. System Review and Modification	14
5.2. System Editing on System Contents tab	15
5.3. Basic Filtering	16
5.4. Device Family and Interconnected Requirements Tab	17
5.5. Messages Tab	18
6. Part 6 – Generate HDL for System	19
6.1. System Review and Modification	19
6.2. HDL Instantiation Template	20
7. Part 7 - Generalizing Basic Platform Designer system building	21
8. Intel Quartus Prime Design Flow	22
8.1. FPGA Hardware Design Flow	22
8.2. Platform Designer System Design Flow	23

List of illustrations

Figure 1 – Application of clock.....	4
Figure 2 – Clock Domains and Domain Filtering.....	5
Figure 3 – Clock Domain Crossing.....	6
Figure 4 - Reset Control Options	7
Figure 5 – Reset connect points	8
Figure 6 – Reset Domain Filtering.....	9
Figure 7 – Reset Adapter Insertion.....	10
Figure 8 – Conduit Interfaces.....	11
Figure 9 – Memory-Mapped Addressing	12
Figure 10 – Address Map Tab	13
Figure 11 – System Review and Modification.....	14
Figure 12 – System Editing on System Contents tab.....	15
Figure 13 – Basic Filtering.....	16
Figure 14 – Device Family Tab.....	17
Figure 15 – Messages Tab.....	18
Figure 16 – System Generation.....	19
Figure 17 – HDL Instantiation Template	20
Figure 18 – Simple QP Design Flow	22
Figure 19 – PD System Design Flow (1)	23
Figure 20 – PD System Design Flow (2)	23
Figure 21 – PD System Design Flow (3)	23

Creating a System Design with Platform Designer

Introduction to SystemVerilog Complexity

This lecture provides information about the usage of Platform Designer user interface to create a system design for implementation in an Intel FPGA device. Also, here explains how Platform Designer fits into the Intel Quartus Prime software design flow as another method of design entry.

4. Part 4 – Connection Details

In this part some information on establishing system connections through various interface types is provided. It covers the methods for making these connections, issues related to unconnected interfaces, and details about specific connections such as clocks, resets, conduits, and memory-mapped interface addressing.

4.1. Clocks

Clocks

All transactions in a system are synchronous, so all components require a clock

- New .qsys files include **Clock Source** component (clock aggregator)
 - Optional; can be removed
- Connect two interfaces
 - Clock Input interface fed from outside system (export)
 - Clock Output interface connects to Clock Input interfaces of other system components
- **Clock** column for quick clock signal connections

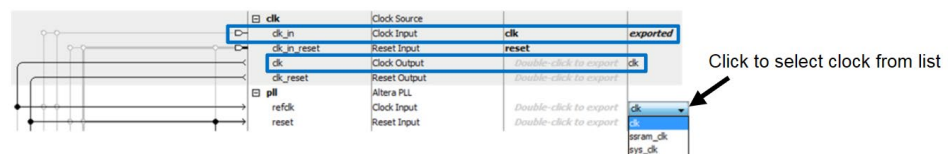
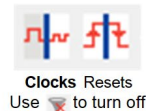


Figure 1 – Application of clock

In a system, each component needs a clock for synchronized operations. To streamline this process, we use a Clock Source component, which acts as a hub for distributing a single external clock to all components, eliminating the need to individually connect each one to an external source. You can customize the Clock Source by specifying the clock frequency for timing considerations. If your system has multiple incoming clocks, you can add more Clock Source components. However, if a component already directly connects to an external clock, you can skip using the Clock Source. Connecting clocks can be done through various strategies or by selecting options from a drop-down list in the System Contents tab for efficiency. This approach simplifies the management of clock connections in the system.

4.2. Clock Domains and Domain Filtering

Clock Domains and Domain Filtering



Highlight and/or filter all components/interfaces in a particular clock domain

View menu → **Clock Domains - Beta**

Display only interfaces in selected domain

The current design has 3 clock domains

Selected clock domains:

- sm_transfer_system.clk - 50.0 MHz
- pll.outclk0
- sm_transfer_system.ssrsm

Selected element:

- sm_transfer_system

Component	Interface	Domain
clk	clk	Clock Source
clk	clk_reset	Clock Output
pll	refclk	Reset Output
pll	reset	Altera PLL
pll	outclk0	Clock Input
pll	outclk1	Reset Input
locked	locked	Clock Output
reset_debounce	reset_debounce	Clock Output
button_switch	button_switch	Reset Button Debounce
button_switch	button_switch	Clock Input
button_switch	button_switch	Button Switch PIO
button_switch	button_switch	Clock Input

Programmable Solutions Group

intel 6

Figure 2 – Clock Domains and Domain Filtering

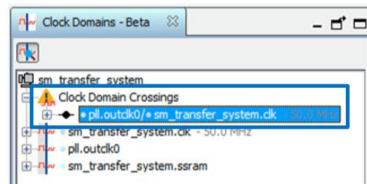
To manage different clock domains in your system, access the Clock Domains tab through the View menu. This tab organizes clock domains hierarchically, presenting each component and its interfaces within the respective clock region. By selecting specific clock domains in this tab, you can filter the System Contents tab to display only the components associated with the chosen domains. The System Contents tab will also use colors to indicate which interfaces belong to each clock domain. Additionally, you can refine the display further to show only the component interfaces within a particular domain, useful when a single component has interfaces across multiple domains.

4.3. Clock Domain Crossing

Clock Domain Crossings

Highlight connections where clock adapters must be added to interconnect

- Red dots indicate location
- Tooltip helps understand why adapter required
- Accept use of adapter or fix



button_switch slave is accidentally in different clock domain from master (indicated by coloring)

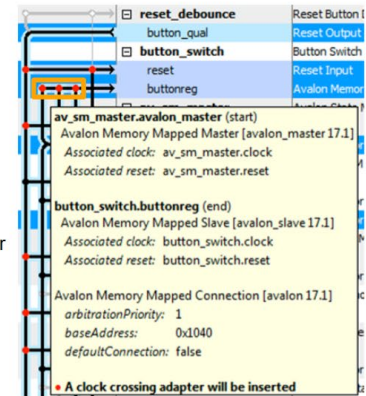


Figure 3 – Clock Domain Crossing

1. Clock Domains Tab Overview:
 - If your system has multiple clock domains (different timing zones), you can check them out in the Clock Domains tab. Here, you'll also see where any special clock crossing logic is placed. This logic helps data move smoothly between areas with different timing.
2. Automatic Clock Crossing Adapter:
 - Platform Designer does some clever work for you by automatically adding this clock crossing logic where necessary.
 - If you select a clock domain crossing from the top of the Clock Domains tab, red dots appear in the System Contents tab. These dots show where the tool plans to add the adapter.
3. Making Decisions:
 - When you hover over these red dots, you'll get little tooltips that explain why the adapter is needed. This helps you understand what's going on.
 - Now, you have a choice: you can let the tool insert the adapter as planned, or you can tweak your design to avoid needing it altogether.
4. Example Situation:
 - Let's say you have a component with two interfaces: one controlled by one clock (let's call it "buttonreg"), and another by a different clock ("avalon_master").
 - This setup might work fine, but if you're facing timing issues in this clock crossing scenario, switching both interfaces to use the same clock might solve the problem.

So, in a nutshell, the Clock Domains tab helps you understand and manage the timing differences in your system, while the tool takes care of adapting data flow between different clock domains automatically, giving you control over the process based on your design needs.

4.4. Reset Control Options

Reset Control Options

Platform Designer allows for complete control over the system's reset implementation

- Can choose to reset all (**System** menu → **Create Global Reset Network**) or only subset of the components in system
 - Must use care to avoid reset loops and system lockup
 - Example: slave reset in middle of transaction, master waits forever
- Reset interfaces declared independent of clock interface
 - But always associated with a clock interface
 - Reset synchronized to associated clock
- Multiple resets can enter the system, like clocks
 - Use **Reset Controller** or **Reset Sequencer** for advanced management



Figure 4 - Reset Control Options

In Platform Designer, reset interfaces are distinct from clock interfaces but are always linked to a specific clock domain. External resets can enter the system directly or through a Clock Source component, with control over their synchronization to the component's clock signal. Resets can also be generated internally by other components, providing reset control through an interface. Two methods exist for connecting resets: a global reset, automatically managed by Platform Designer, or manual connections for more precise control. Care must be taken to avoid reset loops or system lockups. If multiple external resets are present, a reset controller or reset sequencer can be used for efficient management.

In summary, Platform Designer provides options for managing reset interfaces, allowing you to synchronize resets with clock signals and control reset behavior either automatically or manually, depending on your system's requirements.

4.5. Reset Connection Points

Reset Connection Points

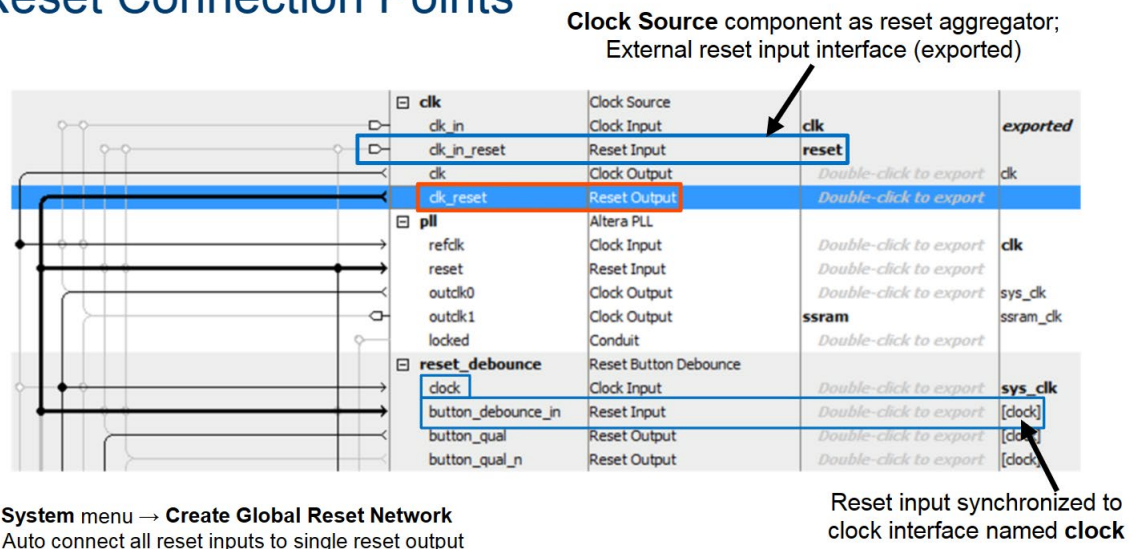


Figure 5 – Reset connect points

1. Manual Reset Connection:

- In this scenario, resets are connected manually.
- First, the reset input interface of the Clock Source component is exported to link with an external reset signal.
- Then, the reset output interface of the Clock Source component connects to the reset input interfaces of other components across the system.

2. Clock Synchronization:

- In the system, each interface, including reset interfaces, is synchronized by a specific clock interface.
- The Clock column in the system indicates which clock interface is responsible for synchronizing each interface.

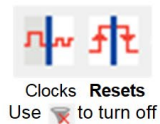
3. Example Highlight:

- For example, let's consider the highlighted "button_debounce_in" reset input interface.
- This interface's synchronization is managed by the clock input interface named "clock" on the "reset_debounce" component.

By manually connecting resets and ensuring synchronization with specific clock interfaces, the system maintains proper timing coordination between different components.

4.6. Reset Domain Filtering

Reset Domain Filtering



Highlight and/or filter all components/interfaces in a particular reset domain

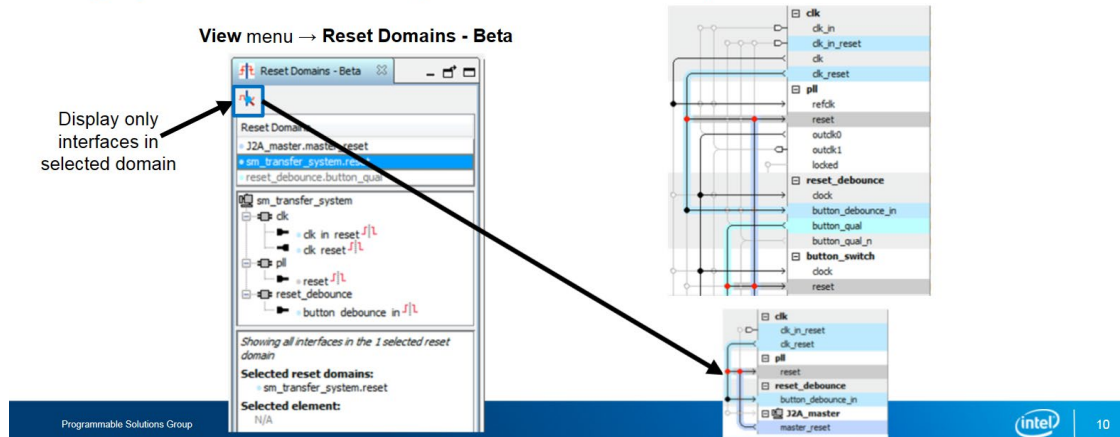


Figure 6 – Reset Domain Filtering

Just like the Clock Domains tab, you can access the Reset Domains tab from the View menu to view components and interfaces grouped into reset domains. This is useful, especially if you've made custom reset connections instead of using the global reset option. Additionally, you can apply further filtering to show only reset interfaces, simplifying the view. One distinction from the Clock Domains tab is that when selecting multiple reset domains, the System Contents tab displays components and interfaces connected to both domains, rather than just those connected to either one.

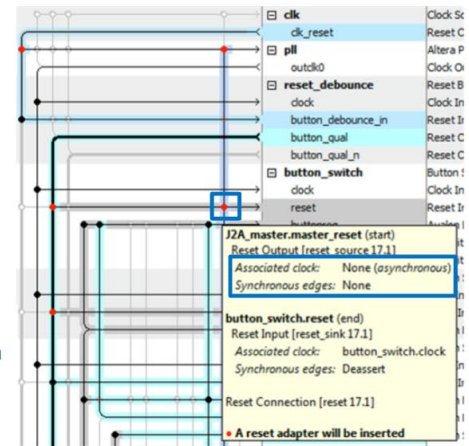
4.7. Reset Adapter Insertion

Reset Adapter Insertion

Like clock domain crossings, adapters automatically added and highlighted in filtered reset domains

- Red dots indicate location
- Tooltip helps understand why adapter required
 - Multiple reset outputs connected to single input
 - Resets associated with different clock domains
 - Reset output/input have different synchronicity

master_reset is asynchronous from JTAG to Avalon® master bridge



Programmable Solutions Group

intel 11

Figure 7 – Reset Adapter Insertion

1. Reset Synchronizers:

- Like clock crossing adapters, you can view where reset synchronizers are inserted in the system by selecting a domain crossing from the Reset Domains tab.
- Red dots on the System Contents tab show where these synchronizers are placed, with tooltips explaining why they're necessary.

2. Reasons for Adapter Insertion:

- Three common situations prompt the insertion of an adapter.
- Like clock domain crossings, you can choose to keep the adapter or adjust the design to avoid needing it, although sometimes it's unavoidable.

3. Example Scenario:

- In the provided example, the system features a JTAG to Avalon master bridge, a special component allowing JTAG access for hardware debugging.
- This bridge uses an asynchronous reset and is connected to the reset input of the **button_switch** component.
- An adapter is required to synchronize the asynchronous reset output of the bridge with the synchronous reset input of the **button_switch** component.

In summary, reset synchronizers are inserted similarly to clock crossing adapters, with red dots indicating their placement. Understanding when and why adapters are needed helps ensure proper system functionality, even in scenarios involving components with different reset types.

4.8. Conduit Interfaces

Conduit Interfaces

Components use conduit interfaces for any individual or grouped signals that don't conform to a supported standard interface

- Recall the supported standard interfaces: Avalon®, Arm® AMBA® AXI standard
- For custom components, user must manually specify signals that make up conduit
- Like conduits *can* be connected in the UI
 - Same signals with opposite directions
- Most often exported outside system
- View signals that make up conduit in **Hierarchy** or **Block Symbol** (component schematic block view) tabs

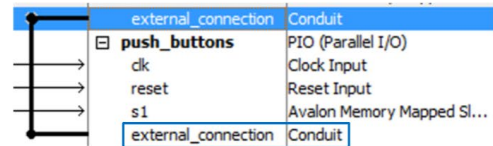


Figure 8 – Conduit Interfaces

Conduit interfaces serve as communication channels between components within a system. They facilitate the exchange of data, signals, or control information between different modules or blocks. These interfaces act as pipelines, allowing seamless transmission of data streams or signals from one component to another. Conduit interfaces play a crucial role in enabling interaction and coordination between various elements of a system, enhancing its functionality and efficiency.

4.9. Memory-Mapped Addressing

Memory-Mapped Addressing

Each memory-mapped master interface has own address map

- Maximum 64-bit address space for each master interface
- Master address map is a collection of:
 - Connected slave interface **Base** addresses
 - Connected slave interface address spans (determines **End** address; spans cannot overlap)
 - Lowest to highest slave addresses make up the address space of the master
- Manually assign non-overlapping base addresses or use **System** menu → **Assign Base Addresses**

Master interface		Lock to prevent auto-assignment		Address map based on connected slaves (0x0 to 0x6f)	
<input checked="" type="checkbox"/> av_sm_master	Avalon State Machine Master		sys_clk	<input checked="" type="checkbox"/> 0x00000000	0x00000007
<input checked="" type="checkbox"/> led_fifo	On-Chip FIFO Memory		sys_clk	<input checked="" type="checkbox"/> 0x00000020	0x0000003f
<input checked="" type="checkbox"/> dma_ssramm_to_led	DMA Controller		sys_clk	<input checked="" type="checkbox"/> 0x00000040	0x0000005f
<input checked="" type="checkbox"/> dma_source_to_ssram	DMA Controller		sys_clk	<input checked="" type="checkbox"/> 0x00000060	0x0000006f
<input checked="" type="checkbox"/> start_pushbutton	PIO (Parallel I/O)		sys_clk		

Figure 9 – Memory-Mapped Addressing

The final item to configure on the System Contents tab is memory-mapped addressing. It allows you to define addressing for the memory-mapped section of your system, supporting up to 64-bit addressing. Each memory-mapped master interface has its unique address map to access connected slave interfaces. The address span of a master interface is determined by the collective address spans of its connected slave interfaces. If there are gaps in the address map, the interconnect automatically handles accesses to them. If multiple master interfaces connect to the same slave interface, their address maps converge. Platform Designer automatically assigns base and ending addresses for slave interfaces as connections are made, but you can manually change these addresses if needed. This can simplify decoding logic, resolve conflicts, or accommodate pre-written software or hardware logic. Slave address spans must not overlap, and auto-assignment of non-overlapping addresses is available. You can prevent auto-assigned addresses from changing by locking them manually.

4.10. Address Map Tab

Address Map Tab

Use **Address Map** tab to manage all addressing

- Double-click cell to manually edit slave base addressing
- Supports per-master addressing for shared slaves
 - Single slave can be represented by different address ranges for different masters

System Contents | **Address Map** | Interconnect Requirements

System: sm_transfer_system Path: test_mem

	J2A_master.master	av_sm_master.avalon_master	dma_ssram_to_led.write_master
button_switch.buttonreg	0x0000_1040 - 0x0000_104f	0x0000_1040 - 0x0000_104f	
dma_source_to_ssram.control_port_sl...		0x0000_0040 - 0x0000_005f	
dma_ssram_to_led.control_port_slave		0x0000_0020 - 0x0000_003f	
led_fifo.in	0x0000_1050 - 0x0000_1057	0x0000_0000 - 0x0000_0007	0x0000_0000 - 0x0000_0007
led_out.led	0x0000_1030 - 0x0000_103f	0x0000_1030 - 0x0000_103f	
source.s1			
ssram_controller.uas			
test_mem.s1			0x0010_00

Slave interfaces

Master interfaces

led_fifo: Avalon FIFO Memory, clk_in, reset_in, in, out, Avalon Memory Mapped Slave, Avalon Streaming Source, mixed, mixed

Programmable Solutions Group | intel | 14

Figure 10 – Address Map Tab

In Platform Designer, there's a detailed Address Map tab in addition to the System Contents tab. This tab offers a comprehensive table of all memory-mapped device addresses, simplifying address mapping and configuration. Each column represents a memory-mapped master interface, while each row represents a slave interface. When a slave is connected to a master, the corresponding cell in the table displays the slave's address span for that master. You can edit these addresses directly in the table. If a slave is connected to multiple masters, its address may vary for each master, and the Address Map tab allows you to manage these variations efficiently.

5. Part 5 – Review Connections

Next step in Basic Platform Designer System Building Flow after adding all components to the system and after this connection to each other is to review their connections.

5.1. System Review and Modification

System Review and Modification

Parameters tab

- Displays parameter editor settings
- Edit parameter values without directly opening component parameter editor

Schematic tab

- Displays schematic of connected system
- Review, connect, disconnect, or export component interfaces

Block Symbol tab

- View block diagram of the entire Platform Designer system or the selected component
 - All exported interfaces appear as port connections

Hierarchy tab

- Review system, component, interface, and connection details
 - Overall system hierarchy
 - Top-level system connections
 - Component interfaces and signals
 - Connections between components

Programmable Solutions Group



16

Figure 11 – System Review and Modification

Use tabs like System Contents, Clock, or Reset Domains to check your progress in building a Platform Designer system. You can access these tabs from the View menu.

1. Additional Tabs for Assistance:

- Apart from the mentioned tabs, there are others to assist you.
- The Parameters tab allows you to review and edit component parameters without reopening the editor.
- Simply select a component in any tab and switch to the Parameters tab to make changes.

2. Schematic tab:

- The Schematic tab offers a visual representation of your system, helping you understand data flow and control logic.
- You can filter the display and make or break interface connections directly in this tab, providing a bird's eye view of complex designs.

3. Block Symbol View:

- The Block Symbol tab displays a block diagram of selected components or the entire system.
- It highlights external connections and shows exported interfaces.
- Enabling the Show Signals option reveals individual signals within each interface.

4. Hierarchical Review:

- The Hierarchy tab presents system connections hierarchically, similar to the Project Navigator in Intel® Quartus® Prime software.

- It allows you to examine the entire system, its components, exported interfaces, component interfaces, signals, and connections between interfaces easily.

These tabs help you navigate, understand, and make adjustments to your Platform Designer system efficiently.

5.2. System Editing on System Contents tab

System Editing on System Contents tab

- Remove components from system
 - Highlight component and click button
 - Highlight component and press **Delete** key
- Change order of components in system
 - Move to top/bottom, move up/down
 - Make system and component connections more easily understandable
 - Does not change addressing or connections




Figure 12 – System Editing on System Contents tab

In Platform Designer, there's a toolbar for editing systems. It lets you remove or edit components and change their order. Rearranging components mainly helps with organization and readability. For instance, you can group components with similar interfaces together. Importantly, changing the order doesn't affect interface connections or memory-mapped slave addresses. It's all about making your system easier to understand and manage.

5.3. Basic Filtering

Basic Filtering

Filter what you see in System Contents by component name, interface type, or connection source/destination

- Default filter is all components and all interfaces
- Select pre-defined filters by right-clicking on any component or interface in System Contents
- Create new custom filters by clicking on Filters button and defining filter criteria
- Remove filters with 

List of built-in and custom filters

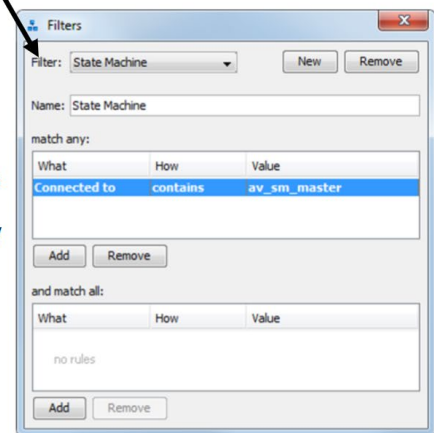


Figure 13 – Basic Filtering

In Platform Designer, you can filter the System Contents tab by clocks and resets, which helps in understanding your system's contents and connections. For more detailed filtering, you can open the Filters dialog box from the toolbar. This allows you to filter components and connections based on criteria such as interface type and component name. You can choose from pre-defined filters or create custom ones, making it easier to focus on specific aspects of your design and store them for future use.

5.4. Device Family and Interconnected Requirements Tab

Device Family and Interconnect Requirements Tabs

Specify general settings for the entire system

Device info brought in if .qsys file created from open Intel® Quartus® Prime software project (required in Pro edition)

Specify other settings for whole system or individual interfaces; see Intel Quartus Prime software handbook for details

Handshake, FIFO, or Auto select for clock crossing logic

Identifier	Setting	Value
\$system	Clock crossing adapter type	Handshake
\$system	Limit interconnect pipeline stages to	2
\$system	Enable instrumentation	TRUE
av_sm_master.avalon_master	Add performance monitor	TRUE
button_switch.buttonreg	Add performance monitor	TRUE
button_switch.buttonreg	Security	Non-secure
led_fifo.in	Add performance monitor	TRUE
led_out.led	Add performance monitor	TRUE
led_out.swpb	Add performance monitor	TRUE
source.s1	Add performance monitor	TRUE

Figure 14 – Device Family Tab

Once your system is ready, you'll want to generate it with its custom interconnect. To do this, you need to set up some important details in two tabs: the Device Family tab and the Interconnect Requirements tab.

- On the Device Family tab, you need to make sure you've picked the right type of device you're targeting, like which family it belongs to and its specific part number. This helps the tool understand what kind of hardware it's working with.
- Then, on the Interconnect Requirements tab, you can tweak settings that affect how the connections between different parts of your system are handled. For example, you can decide how much extra processing the interconnect should do and how it should handle timing differences between different parts of your system. It's like giving instructions on how you want things to be connected inside your system.
- These tabs might seem a bit complicated, but they're important to make sure everything works smoothly. If you're not sure about something, you can always check the handbook for more details.

5.5. Messages Tab

Messages Tab

Review **error**, **warning**, and **informational** messages about the system

- Includes summary of errors, if any, and warning counts
- Double-click message to highlight referenced interface/connection
- Must be “clean” (no errors) in order to generate

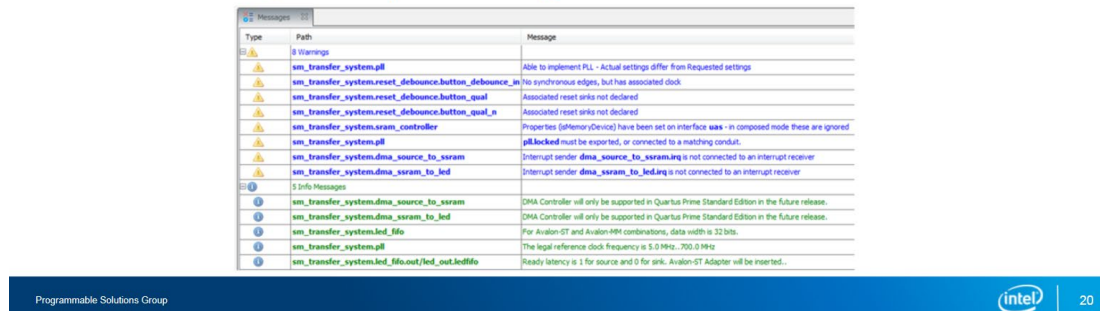


Figure 15 – Messages Tab

At the bottom of Platform Designer, there's a Messages tab. It shows different types of messages: info, warnings, and errors. These messages tell you about things you've done or haven't done in your system. If you click on a message, it will show you what it's talking about in your system. Red errors need fixing before you can finish your work, while blue warnings are just reminders to double-check things. It's like a little helper that tells you if something needs attention before you're done.

6. Part 6 – Generate HDL for System

The final step is to generate the HDL code for the system.

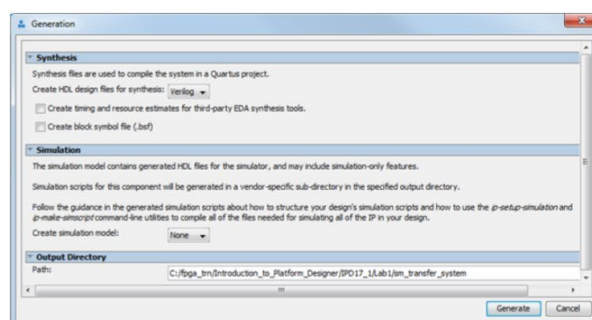
6.1. System Review and Modification

System Generation

When you've finished building your system, generate the files required for synthesis or simulation

- Choose file types to generate and HDL language
- Choose **Output Directory** (default is subdirectory named after **.qsys** file)
- Click **Generate**

Generate menu → **Generate HDL**



Programmable Solutions Group



22

Figure 16 – System Generation

Imagine you've just finished building your system using Platform Designer, and now it's time to turn it into real computer code. Here's what you need to do:

1. Go to the menu and select "Generate HDL" – this tells the computer to create the code for your system.
2. Decide what kind of files you want the computer to make. If you're planning to make your system real and run it on actual hardware, you'll want "synthesis optimized" files. If you just want to test it out on the computer, you'll need "simulation model" files. You can also get a "symbol" file to use in a schematic if you want.
3. Choose which computer language you want the code to be written in – it's like picking the language you speak best.
4. Tell the computer where to save all these new files. You can choose a main folder, and the computer will make subfolders inside it for the different types of files.
5. Finally, click "Generate" and watch the computer work its magic – it'll create all the code you need to make your system come to life!

6.2. HDL Instantiation Template

HDL Instantiation Template

Display a Verilog or VHDL instantiation template based on exported interfaces to easily instantiate system in an Intel® Quartus® Prime software project

Generate menu → Show Instantiation Template

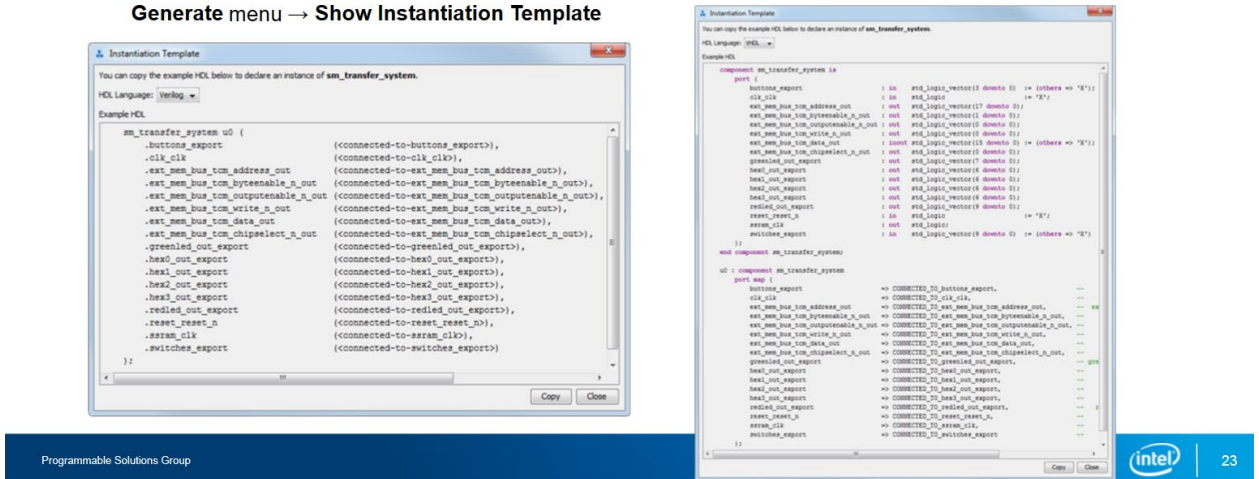


Figure 17 – HDL Instantiation Template

If you want to add another design file, Platform Designer makes it super easy. Here's how:

1. Go to the menu and pick "Show Instantiation Template" – this gives you a special kind of template, like a guide, to help you put your system into the other design file.
2. Choose whether you want the guide to be in VHDL or Verilog – it's like picking the language you want the guide to be written in.
3. Copy the code into your design file.
4. You'll see some parts in the guide that you need to change to fit your design. Editing the port mapping sections in the template involves specifying how each part of your system connects to other components in the overall design.

7. Part 7 - Generalizing Basic Platform Designer system building

Here are the key steps for creating a system design in Platform Designer:

1. First, choose the components you need from the IP Catalog and configure them using their parameter editors.
2. Then, utilize the Platform Designer interface to connect or export interfaces.
3. Next, review your connections to ensure they align with your design's goals and functionality.
4. And lastly, generate Verilog or VHDL files to represent your system, ready to integrate into your Intel® Quartus® Prime software project for compilation.

8. Intel Quartus Prime Design Flow

8.1. FPGA Hardware Design Flow

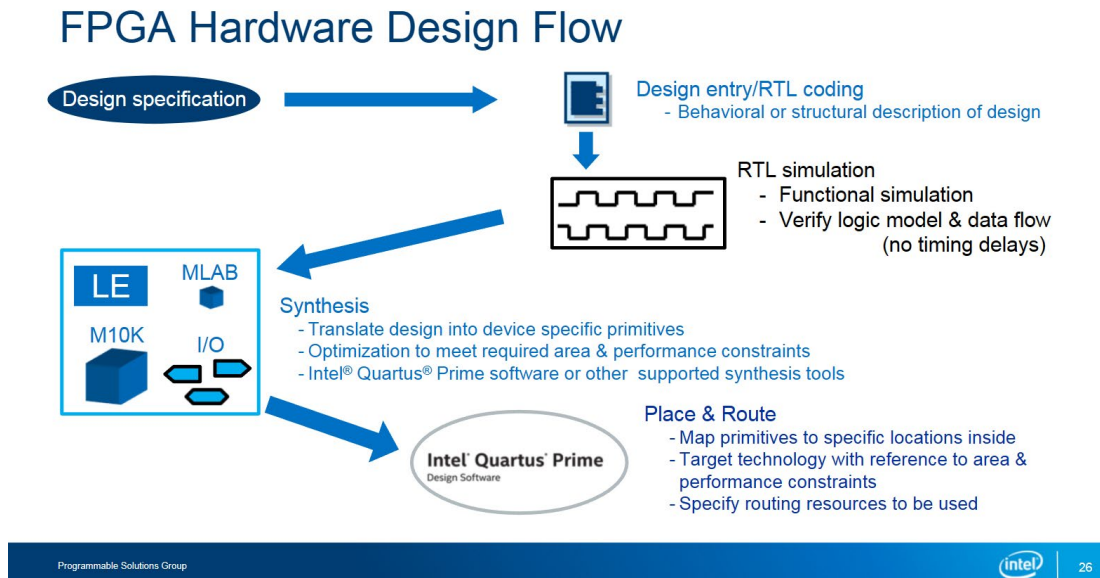


Figure 18 – Simple QP Design Flow

On the figure above we can see the typical FPGA design process:

1. First, you write code that describes how your design should work.
2. Then, you run a simulation to make sure it behaves as expected.
3. Next, synthesis transforms your code into specific instructions for the FPGA.
4. After that, the Intel Quartus Prime software figures out where to put these instructions and connects them together. Platform Designer fits into this process as a tool for building systems, creating code that fits into this flow just like any other method of design entry.

8.2. Platform Designer System Design Flow

Platform Designer System Design Flow

It's easy to integrate a Platform Designer system into an Intel® Quartus® Prime software project

1. Create a system, making sure to target correct device
 - Standard edition: device set correctly if PD started from open project
 - Pro edition: system *must* be associated with a project when created
2. Decide what will be top-level project entity and where system will exist in hierarchy
 - Top-level Platform Designer system: set top-level entity to system name and add **.qsys** or **.qip** file to project (*see next step*)
 - Lower in project hierarchy in other design file (**.v**, **.vhd**, etc.): instantiate system in file using instantiation template
 - Required if I/O pin names are to be different from exported signal names

Programmable Solutions Group



Figure 19 – PD System Design Flow (1)

Platform Designer System Design Flow (cont.)

3. Either generate system files manually or allow Intel® Quartus® Prime software compile generate during synthesis
 - Generate manually: **Generate** menu → **Generate HDL**
 - Placed by default into <system name>/synthesis subdirectory
4. Add either **.qsys** or **.qip** (Intel Quartus Prime IP) file to project
 - Can add only one or the other
 - **.qsys**: system generated automatically every time project is compiled (synthesized)
 - **.qip**: adds all required generated HDL files to project but system must be manually generated before project compilation to create all files (including **.qip**)

Figure 20 – PD System Design Flow (2)

Platform Designer System Design Flow (cont.)

5. Constrain design: timing constraints (**.sdc**), I/O locations, & other assignments added to **.qsf**
 - Many off-the-shelf components and the system generate their own **.sdc** pointed to by **.qip**
 - Still should create top-level **.sdc** for clock inputs to system (`create_clock`, `create_generated_clock`) and for PLL-generated clocks in system (`derive_pll_clocks`)
6. Compile design
 - Use Platform Designer to make changes to system, rather than editing the generated HDL

Figure 21 – PD System Design Flow (3)

To integrate a Platform Designer system into the usual Intel Quartus Prime software design process, follow these steps:

1. Build your system using Platform Designer and set the target device.
2. Decide if your system will be the main part of your project or a part of it.
3. If your system is the main part, specify its name as the top-level design entity in the Intel Quartus Prime project. If it's lower down, use the instantiation template to place it correctly.
4. If you want different names for your system's signals and the I/O pins, create a wrapper to translate them.
5. Set up your project to reference the generated system. You can add the .qip file manually or include the .qsys file and let the project compile it.
6. Add project constraints like I/O assignments and timing constraints.
7. Compile your project.

Remember, if you need to change your Platform Designer system, do it in Platform Designer itself. Don't edit the generated HDL directly, as your changes won't be saved back to the .qsys file and will be overwritten when the system is regenerated.