

AN 812: Platform Designer System Design Tutorial



ID: 683855 Version: 2018.04.02

AN-812

Contents

Platform Designer System Design Tutorial	3
Hardware and Software Requirements	
Download and Install the Tutorial Design Files	5
Build the Hardware Design	6
Open the Intel Quartus Prime Pro Edition Project	6
Build a Platform Designer System with a Top-Down Approach	7
Implement the Memory Tester Subsystem	
Build Software Applications and Download the Design	44
Hardware setup	44
Run the Bash Script	45
AN 812: Platform Designer System Design Tutorial Revision History	

intel

Platform Designer System Design Tutorial

The Platform Designer system integration tool for Intel FPGA and SoC devices automatically generates interconnect logic to connect intellectual property (IP) components and subsystems. Using Platform Designer saves time and effort in the design process. Platform Designer inherits the ease of use of Platform Designer (Standard). In addition, Platform Designer introduces hierarchical isolation between system interconnect and IP components. This tutorial is for users who have basic knowledge of Intel[®] Quartus[®] Prime Pro Edition software and Platform Designer (Standard), and want to experience the new features of Platform Designer.

This tutorial guides you through the following processes:

- Building systems in Platform Designer, and integrating those systems into an Intel Quartus Prime Pro Edition project.
- Explains the different user flows between Platform Designer (Standard) and Platform Designer.
- Demonstrates some of the new features of Platform Designer and how it increases efficiency and flexibility for team-based design.

The procedures in this tutorial provide you with a template to design a system that uses various test patterns to test an external memory device. The final system contains the following components:

- A processor subsystem which contains an Intel Nios[®] II/e core. The subsystem also includes an on-chip RAM to store the software code and a JTAG UART to communicate and display the memory test results in the host PC's console.
- A memory tester subsystem to interact with an SDRAM controller.
- The memory tester subsystem consists of a pattern generator subsystem, a pattern checker subsystem, a memory tester, a pattern writer, and a pattern reader.
- The pattern generator subsystem consists of a custom pattern generator, a pseudo random binary sequence (PRBS) pattern generator, along with a multiplexer (MUX) to select between these two.
- A data pattern checker subsystem consisting of a custom pattern checker, a pseudo random binary sequence (PRBS) pattern checker, along with a demultiplexer (DEMUX).
- Pattern writer and pattern reader subsystems that interacts with the SDRAM controller.
- A SDRAM controller to control the off-chip DDR SDRAM device under test.



Figure 1. Platform Designer System





There are four broad steps in this tutorial:

- 1. Build a processor subsystem from scratch in Platform Designer.
- 2. Build a top-level Platform Designer system with memory tester subsystem instantiated as a generic component.
- 3. Implement a generic component.
- 4. Create a Nios II software application and run the design on a FPGA.

Hardware and Software Requirements

This design targets the Intel Arria[®] 10 GX FPGA Development Kit (with DDR4 daughter card installed). To complete this tutorial, you need the following software and tools:

- Intel Quartus Prime Pro Edition 17.0 or later
- Nios II EDS (installs with the Intel Quartus Prime Pro Edition software)
- Board Test System (installs with the Intel Arria 10 GX FPGA Development Kit package)

Related Information

- Intel Quartus Prime Pro Edition Download Page
- Intel Arria 10 GX FPGA Development Kit
- Intel Arria 10 Board Test System

Download and Install the Tutorial Design Files

- 1. On the **Platform Designer Tutorial Design Example** page, under **Using this Design Example**, click **Platform Designer Tutorial Design Example** (.zip) to download and install the tutorial design files for the Platform Designer tutorial.
- 2. Extract the contents of the archive file to a directory on your computer. Do not use spaces in the directory path name.

The qsys_pro_tutorial_design_Arria_10_17p0.zip contains the following project files and is referred to as <project folder> in the rest of the document.

Table 1. Qsys Pro Design Tutorial Project Files

Folder Structure	Description
/complete_design	The final design. You can use this design as a reference and guidance while you follow the tutorial. You may also use the prebuilt systems in it if you want to skip certain steps of this tutorial.
/ip	The folder that stores IP component source files. The pattern_checker_system and pattern_generator_system are pre-generated for you.
/memory_tester_ip	The folder that contains source files for all custom components.
/software	This folder contains source code for building Nios II software applications and two scripts that automate this process for you.
A10.qpf	An Intel Quartus Prime Project file (. qpf).
	continued



Platform Designer System Design Tutorial 683855 | 2018.04.02



Folder Structure	Description
A10.qsf	An Intel Quartus Prime Settings file (.qsf), containing pre-defined pin assignments.
memory_tester_search_path.ipx	IP Index file (.ipx) that specifies the path to the source files of the custom components.
memory_tester_subsystem_bb.ipxact	The .ipxact file that defines the interfaces for your generic component.
my_constraints.sdc	A Synopsys Design Constraints, or SDC, file ($.sdc$) containing timing constraints.
pattern_checker_system.qsys	Pre-built Platform Designer System file (.gsys).
pattern_generator_system.qsys	Pre-built Platform Designer System file (.gsys).
top_level.v	Top-level Verilog Design file (. v) .

Related Information

- Platform Designer System Design Example
- Platform Designer System Design Example (.zip)

Build the Hardware Design

Open the Intel Quartus Prime Pro Edition Project

You must specify or create an Intel Quartus Prime Pro Edition project when you create or open a new Platform Designer system. Platform Designer inherits the device family or number from the Intel Quartus Prime Pro Edition software, which guarantees the or Platform Designer coherency. To open the Intel Quartus Prime Pro Edition project:

- 1. Launch Intel Quartus Prime Pro Edition software.
- 2. Click File ➤ Open Project.
- 3. Browse to the project directory.
- 4. Select A10.qpf and click **Open**.

The top-level RTL, pin assignments, and timing constraints have been created for you. The file references and pin assignments are saved in A10.qsf.



Figure 2. Intel Quartus Prime Pro Edition Project

ile Edit View Project Assignments Processing Tools Window Help Search Intel FPGA											
🗋 🗖 🖶 🗡 🛍 💼 つ C 🗛 10				• 🧭 🎸 🖉	S = F = K	🍳 🔕 🖪	* 🚸 🐉 🍙	•			
Project Navigator Q 🚇 🛛 🛛		C	ompilation Da	shboard 🗶	top_level.v	×	Assignmer	nt Editor 🌖	٢		
🃂 Files	<<	new>	> 🔻 🖌 Filte	er on node names	*			*	Category: All		\$
stell top_level.v		in tr	From	To	Accimmenthisme	Malue	Epobled	Fatitu	Commont	Teg	
my_constraints.sdc	-	.att	FIOIII	10	Assignment Name	value	Enabled	Enuty	comment	Lag	
	1				Location	PIN_F34	Yes				
	2	×.		reset_n[0]	Location	PIN_BD27	Yes				
	3			sdram_a[0]	Location	PIN_P132	Ves				
	4			sdram_a[1]	Location	PIN_L32	Ves				
	5	<u> </u>		sdram_a[2]	Location	PIN_N34	Yes				
	7			sdram_a[3]	Location	PIN_H35	Vec				
	/			sdram_a[4]	Location	PIN_L34	Ves				
	0	5		sdram_a[5]	Location	PIN_K34	Yes			_	
	9			sdram_a[8]	Location	PIN_M33	Yes				
	10			sdram_a[7]	Location	PIN_L33	Vas				
	12			sdram_a[0]	Location	PIN_333	Ves				
∧ H F P Com	12	-		sdrain_a[3]	Location	PIN U21	Vor				
Tasks 💷 🗵 🗷	14	5		 sdra[10] sdra. [11] 	Location	PIN_I31	Ves				
Project	15			 sdra[11] sdra. [12] 	Location		Ves		-		
Revisions	16	5		 sdra[12] sdra[12] 	Location	PIN H22	Vec				
Project Files	17	5		sdra[13]	Location	PIN G32	Ves		-		
🗋 New	18	5		 sdra[14] sdra. [15] 	Location	PIN E32	Ves				
Open	19	5		sdra [16]	Location	PIN F32	Yes				
🙀 Add/Remove Files in Project	20	5		sdra n[0]	Location	PIN P34	Ves		-		
- A Qsys Pro	21	5		sdra_n[0]	Location	PIN E35	Yes				
Assignments	22	5		sdraa[0]	Location	PIN F33	Yes				
- Device											
Settings	X										
🍼 Pin Planner	æ										
Assignment Editor											
8 0 8 🔬 🔥 /	•	2251	lary y		Ise Regular Expressions	An Eind	Find Next				
				•]□•	sae regula expressions		or more Ac				
Message								Mess	age ID		
	_								-		_
80											
S	_										_
Z System Processing											

Build a Platform Designer System with a Top-Down Approach

- 1. To launch Platform Designer, click **Tools ➤ Platform Designer**.
- 2. Click the **Create new Qsys system** button and name the new Platform Designer system **top_system.qsys**.

Figure 3. Create New System Dialog Box

💑 💿 Create New S	ystem <@sj-slscf013> 💿 🔿	×
System IP Variant]	
Select the Quartus Prir	me Pro project file and Qsys system file to create or open.	
Quartus project:	/data/jlsun/project/tt_qsys_design/reorg/A10.qpf	9
Revision:	A10	9
Device Family an	nd Part	_
Device family:	Set by Quartus project 🚽 Retrieve Values	
Device part:	Set by Quartus project	
Qsys system:	/data/jisun/project/tt_qsys_design/Qsys_Pro_tutorial_design_Arria_10_17p0_test/top_system.qsys 💽 🛄	•
	/	
	open Ca	ncel
	Create New System	

3. Click **Create**. The resulting system comes pre-populated with a clock bridge and a reset bridge.





- 4. Right-click the name of the **clock_in** component and click **Rename**. Type ext_clk.
- 5. In the parameter editor, change the **Explicit clock rate** to 100MHz (100,000).
- 6. Right-click the name of the **reset_in** component and click **Rename**. Type ext_reset.

Figure 4.Rename the Clock Bridge and Reset Bridge



Add a Processor Subsystem to the Top-Level

Using subsystems helps maintain design hierarchy. You can add a subsystem in Platform Designer and easily implement it.

1. Right-click in the **System Contents** tab and click **Add New Subsystem to Current System**.

Figure 5. Add a New Subsystem to the Current System



- 2. In the **Confirm New System Filename** dialog box, click the sysA subsystem and rename it by typing cpu_subsystem.gsys.
- 3. Click **OK**.





Figure 6. Confirm New System Filename Dialog Box

👗 💿 Confirm New System Filenar 😔 🛛 🛞							
1 ton system [ton system asys*]							
♀ · ··································							
- 💷 Instance: sysA_0							
OK Cancel							

- To rename the instance from sysA_0 to cpu_subsystem, right-click the name of the new subsystem in System Contents and click Rename. Type cpu_subsystem.
- To implement the cpu_subsystem component, right-click the name and click Drill into Subsystem. Alternatively, you can double-click cpu_subsystem in the Subsystems folder in the Hierarchy list.

Figure 7. Drill into Subsystem Command to Modify a New Subsystem



This opens <code>cpu_subsystem.qsys</code> as a new Platform Designer project where you can add components.

Build the Processor Subsystem

To build the cpu_subsystem subsystem, you add IP components from the IP Catalog:







- 1. Type clock in the search box of the IP Catalog and double-click **Clock Bridge** to add that component.
- 2. Type reset in the search box of the IP Catalog and double-click **Reset Bridge** to add that component.
- 3. Right-click the name of the clock bridge and click **Rename**. Type mem_clk to rename the clock bridge.
- 4. Right-click the name of the reset bridge and click **Rename**. Type mem_reset to rename the reset bridge.
- 5. To add a second clock bridge, type clock in the search box of the IP Catalog and double-click **Clock Bridge** to add that component.
- 6. To add a second reset bridge type reset in the search box of the and double-click **Reset Bridge** to add that component.
- 7. Right-click and rename the new clock bridge and reset bridge to cpu_clk and cpu_reset, respectively.
- 8. Connect the out_clk signal of mem_clk to the clk signal of mem_reset.
- 9. Connect the out_clk signal of cpu_clk to the clk signal of cpu_reset.
- 10. Edit the exported interface by double-clicking the name in the **Export** column, from the following table:

Table 2.Export Rename Values

Component Name	Description	Export Value
mem_clk	Clock Input	mem_clk
mem_reset	Reset Input	mem_reset
cpu_clk	Clock Input	cpu_clk
cpu_reset	Reset Input	cpu_reset





Your results should match those in the following figure:

Figure 8. Clock and Reset Components

dock	System	cpu_subsystem Path: Name In_clk in_clk out_clk im_mr.eset clk in_reset out_reset out_reset in_clk	cpu_clk Description Clock Bridge Clock Input Clock Input Reset Bridge Clock Input Reset Input Reset Output Reset Output Clock Bridge	Export mem_clk Double-click to export Double-click to export mem_reset Double-click to export	System: cpu_subsystem Path: cpu_clk. HDL entity name (cpu_subsystem) IP file: [p/(cpu_subsystem/cpu_ Any changes here will be immediately written out to disk.
Toplet New Companent Jbran P Bröges ein Adaptors Cicks Ci	Use Connections	Name Name Name Name Name Name Name Name	Description Clock Bridge Clock Input Clock Output Reset Bridge Clock Input Reset Input Reset Output Clock Bridge	Export mem_cik Double-click to export Double-click to export mem_reset Double-click to export	HDL entity name: cpu_subsystem Any changes here will be immediately written out to disk.
OFF O	System Messages	IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	Clock Mouth Clock Mouth Clock Moud Reset Output Reset Output	(pu.dk Double-citick to export Double-citick to export (pu.reset Double-citick to export (pu.reset Double-citick to export (pu.reset Double-citick to export (pu.reset) Double-citick to export	Crock bridge Denvisit Caker are: O Dervised Caker are: O Dervised Caker are: O Number of Clock Outputs: Type Messages Type Message Olo messages)

Add a Nios II Processor

- 1. Type nios in the search box of the IP Catalog and double-click $\ensuremath{\text{Nios II}}$ $\ensuremath{\text{Processor}}$
- 2. In the **Select an Implementation** parameter editor, select the **Nios II/e** processor.
- 3. To add the Nios II/e processor to the design, click Finish.
- 4. Right-click the name of the Nios II processor component and click **Rename**. Type cpu to change the name.
- 5. In the **Export** column, double-click the entry corresponding to the **Reset Output** for the **cpu** component and rename it cpu_jtag_debug_reset.

Errors regarding reset and exception slaves can be resolved after you add connections.





Figure 9. cpu_subsystem Export Naming



Add RAM, JTAG UART, and Avalon-MM Pipeline Bridge

The final components you'll need to add and configure are an On-Chip RAM, a JTAG UART, and an Avalon-MM Pipeline Bridge.

- 1. Type ram in the IP Catalog search box and double-click **On-Chip Memory (RAM** or **ROM)**.
- 2. In the **On-Chip Memory (RAM or ROM)** parameter editor, in the **Size** box, set the **Total memory size** to 8192 bytes.
- 3. To add the **On-Chip Memory (RAM or ROM)** component to your design, click **Finish**.
- 4. Right-click the name of the **On-Chip Memory (RAM or ROM)** component and click **Rename**. Type onchip_ram to change the name.
- 5. Type jtag uart in the IP Catalog search box and double-click JTAG UART.
- 6. To add the **JTAG UART** component to your design with default settings, click **Finish**.
- 7. Right-click the name of the **JTAG UART** component and click **Rename**. Type jtag_uart to change the name.
- 8. Type pipeline bridge in the IP Catalog search box and double-click Avalon-MM Pipeline Bridge.
- 9. In the **Avalon-MM Pipeline Bridge** parameter editor, change the following settings:
 - Set the **Address width** to 16.
 - Set the Maximum pending read transactions to 1.
- 10. To add the Avalon-MM Pipeline Bridge to your design, click Finish.
- 11. Right-click the name of the **Avalon-MM Pipeline Bridge** component and click **Rename**. Type pipeline_bridge to change the name.
- 12. In the **Export** column, double-click the entry that corresponds to the m0 signal for the **pipeline_bridge** component and type master.

The Avalon-MM Pipeline Bridge allows the processor subsystem cpu_subsystem to export a single Avalon-MM master interface. Your design can then access the slave interfaces in a higher-level system, and handle address offsets automatically. The bridge also improves timing performance.





All the required components are now included in this subsystem. Compare the settings in your design with the following figure and make sure your components and exported interfaces are named correctly.



Figure 10. Export Names for cpu_subsystem Components

Connect cpu_subsystem Components

Connect the component signals below by clicking the dots across from the appropriate signals, or by right-clicking the signal and choosing from the drop-down menu.

Follow these steps to connect the components:

Figure 11. Illustrated Clock and Reset Component Connections for cpu_subsystem





Source Component/Signal	Target Component/Signal
mem_clk/out_clk	pipeline_bridge/clk
<pre>cpu_clk/out_clk</pre>	 cpu_reset/clk cpu/clk onchip_ram/clk1 jtag_uart/clk
mem_reset/out_reset	 cpu/reset onchip_ram/reset1 jtag_uart/reset pipleline_bridge/reset
<pre>cpu_reset/out_reset</pre>	 cpu/reset onchip_ram/reset1 jtag_uart/reset pipleline_bridge/reset
cpu /data_master	 onchip_ram/s1 jtag_uart/avalon_jtag_slave pipleline_bridge/s0
cpu /instruction_master	onchip_ram/s1

Table 3. Component Connections for cpu_subsystem

Compare the finished connections to the following figure:

Figure 12. Component Connections for cpu_subsystem



System Connectivity Error appears in the **System Messages** tab. To access this tab, click **View ➤ System Messages**. The **System Connectivity Error** occurs because when the base address of the Avalon-MM slaves are not assigned, which can cause address overlap.





Follow these steps to assign the **Base** address to the value shown in the following figure. Click the "lock" icon to lock the address.

- 1. In the **Base** column, click the value for **Avalon Memory Mapped Slave** (**Description** column) of the **cpu** component and type 12000.
- 2. Find the **Avalon Memory Mapped Slave** entry for the **onchip_ram** component and type 10000 as the value in the **Base** column.
- 3. Find the **Avalon Memory Mapped Slave** entry for the **jtag_uart** component and type 12800 as the value in the **Base** column.

Figure 13. Base Address Assignments for cpu_subsystem Components

		Name	Description	Export	Clock	Base	End	
		mem_clk	Clock Bridge					
		in_clk	Clock Input	mem_clk	exported			
		out_clk	Clock Output	Double-click to export	mem_clk			
		mem_reset	Reset Bridge					
\rightarrow		clk	Clock Input	Double-click to export	mem_clk			
D-		in_reset	Reset Input	mem_reset	[clk]			
\vdash		out_reset	Reset Output	Double-click to export	[clk]			
		cpu_clk	Clock Bridge					
	1	in_clk	Clock Input	cpu_clk	exported			
<u> </u>		out_clk	Clock Output	Double-click to export	cpu_clk_o			
	Ξ	cpu_reset	Reset Bridge					
\rightarrow		clk	Clock Input	Double-click to export	cpu_clk_o			
		in_reset	Reset Input	cpu_reset	[clk]			
\vdash		out_reset	Reset Output	Double-click to export	[clk]			
	Ð	cpu	Nios II Processor					
\rightarrow		clk	Clock Input	Double-click to export	cpu_clk_o			
		custom_instruction	Custom Instruction Master	Double-click to export				
Ť		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
\rightarrow		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_2000	0x0001_27ff	
9	1	debug_reset_requ	Reset Output	cpu_jtag_debug_reset	[clk]			
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
		irq	Interrupt Receiver	Double-click to export	[clk]		IRQ O	IRQ 31
\rightarrow		reset	Reset Input	Double-click to export	[clk]			
-		onchip_ram	On-Chip Memory (RAM or ROM)					
\rightarrow		clk1	Clock Input	Double-click to export	cpu_clk_o			
\rightarrow		reset1	Reset Input	Double-click to export	[clk1]			
\rightarrow		\$1	Avalon Memory Mapped Slave	Double-click to export	[CIK1]	0x0001_0000	0x0001_1fff	
		jtag_uart	JI AG UART					
		avaion_jtag_slave	Avaion Memory Mapped Slave		[CIK]	 0x0001_2800 	0x0001_2807	
		CIK	Clock Input		cpu_cik_o			
		irq	Interrupt Sender		[CIK]			
- 3		reset	Keset Input	Double-click to export	[CIK]			
		pipeline_bridge	Avaion-MM Pipeline Bridge	Double slide to expect	mam all			
		CIR.	Avelop Memory Manned Master	COURDER CHER ED EXPORT	felki			
		rocot	Reset leput	Double click to expert	[CIN]			
		c0	Amin Memory Manned Slove	Double click to export	[CIN]	A 020000 0000	OVODOD FFFF	
		50	Avaluti methory mapped stave	L'UGUIE-LIILK EU EXPUIL	[[CIN]	= 0x0000_00000	0.0000_1111	

- 4. To resolve any remaining system connectivity errors, in the **System Messages** tab, click **Sync All System Info** in the bottom of the GUI. This synchronizes the component instantiations with their .ip files.
- 5. To resolve errors in the parameterization of the cpu component (the name of the component is still red), double-click cpu and you can see the Parameterization Messages in the Parameters tab. Platform Designer separates the messages for system connectivity and component parameterization, which simplifies the error and resolution compared to the combined messaging in Platform Designer (Standard).



Figure 14. Parameterization Messages

irchy 🛛 🖰 IP Catalog 🖇 🛛 🗕 🗗 🗖	🎞 Syste	🕅 Address 🕅 Inter	conr 🖾 Details 🖾 🔤 🖬 🗖	👌 🦄 Par	ameters 🕅			- 6
IP Catalog		System: cpu_subs	stem Path: cpu	System:	cpu_subsystem	Path: cpu		
		Name	Description	HDL er	itity	IP		
A New Component		🖯 mem_clk	Clock Bridge	name:	cpu_subsyst	em_nios2_g fil	e: ip/cpu_subsystem	/cpu_subsystem_n
New Component	- D-	in_clk	Clock Input	Any cha	anges here will be i	mmediately writt	en out to disk.	
ary Issic Eurotions	× —	out_clk	Clock Output					
Bridges and Adaptors		🖻 mem reset	Reset Bridge	Nios	II Processor			
Memory Manned	\rightarrow	clk	Clock Input	altera_r	nios2_gen2			Details
 Avalon-MM Clock Crossing Brit 	×	in reset	Reset Input					
Avalon-MM Pipeline Bridge		out reset	Reset Output	Aufahu		MARKED AND AND AND AND AND AND AND AND AND AN	Services V mag pub	
Avalon-MM Unaligned Burst Ex		E cou_clk	Clock Bridge	Anth	metic instructions	MMU and MPU	Settings JI AG Dec	iug Advanced Fi
ITAG to Avalon Master Bridge		in clk	Clock Input		Main	Vectors		Cac
Streaming	1 x — (out clk	Clock Output	T Por	at Vactor			
Avalon Packets to Transaction		E cou reset	Reset Bridge	Kes	et vettor			
Avalon-ST Adapter		clk	Clock Input	Kese	et vector memory.		None	-
Avalon-ST Ruter to Packets Co	0	in recet	Reset Input	Rese	et vector offset:		0x00000000	
Avalon-ST Oyles to Fackets Co Avalon-ST Channel Adapter		out recet	Recet Output	inter inter	a sector officies		0x00000000	
Avalon-ST Channel Adapter Avalon ST Data Format Adapter		out_reset	Nesel Output	Rese	et vector:			
 Avaion-ST Data Format Adapt 		es cpu	NIOS II Processor					
Avaion-ST Delay		CIK	Clock input	✓ Exc	eption Vector			
 Avalon-ST Demultiplexer 		custom_instruction.	Custom Instruction Master	Exce	ption vector memo	iry:	None	-
 Avalon-ST Error Adapter 		data_master	Avalon Memory Mapped Master				TTOTIC	-
 Avalon-ST Multiplexer 	\rightarrow	debug_mem_slave	Avalon Memory Mapped Slave	Exce	ption vector offset		0x00000020	
 Avalon-ST Packets to Bytes Co 	-0-	debug_reset_requ.	. Reset Output	Exce	ption vector:			
 Avalon-ST Pipeline Stage 		instruction_master	Avalon Memory Mapped Master					
 Avalon–ST Splitter 		ing	Interrupt Receiver	T Fas	t TLR Miss Except	tion Vector		
 Avalon-ST Timing Adapter 	\longrightarrow	reset	Reset Input	Fort	TLB Micc Exception	worter memory		
Configuration and Programming		E onchip_ram	On-Chip Memory (RAM or ROM)	r d Si	TLD MISS Exception	r vector memory.	None	-
 Avalon-MM Partial Reconfiguration 	\rightarrow	clk1	Clock Input	Fast	TLB Miss Exception	h vector offset:	[0x00000000]	
Avalon-ST Partial Reconfiguration	$ \longrightarrow $	reset1	Reset Input		THE MEN PROPERTY			
- On Chip Memory		<1	Avalon Memory Manned Slave	Fast	ILB MISS Exception	n vector:		
 Avalon EIEO Memory 		E itag wart	IT AC HAPT					
Avalon-ST Dual Clock EIEO		in Jung danc	A plan Manan (Manned Slave					
Avalon-ST Multi-Channel Shared I		avaion_jcag_siave	Avaion Memory Mapped Slave					
Avalon ST Reund Robin Scheduler		CIK	CIOCK INPUT					
Avalon-ST Round Robin Scheduler		ing	Interrupt Sender					
 Avaion-ST Single Clock FIFO 		reset	Reset Input					
 On-Chip Memory (KAM or KOM) 		🗄 pipeline_bridge	Avalon-MM Pipeline Bridge					
Simulation; Debug and Ventication	\rightarrow	clk	Clock Input					
 Debug and Performance 		m0	Avalon Memory Mapped Master 🥃					
 Performance Counter Unit 	4		Passed launa					
 System ID Peripheral 								
 Simulation 	n~ 51	t 📑 🤝 🛒 Current fi	iter:	4				
 Altera Avalon Interrupt Sink 	-							
 Altera Avalon Interrupt Source 	System	Messages 😂	- 6 0	Parame	terization Messa	jes		
 Altera Avalon-MM Master BFM 				Tur	0		Morcoa	0
 Altera Avalon-MM Monitor 	Type		Path	T yp	c		Messay	c
 Altera Avalon-MM Slave BFM 	የ 🔺	2 Component Instantial	tion Warnings	N 🔀				
 Altera Avalon-ST Monitor 	A				Please Sync	All System Infos	before attempting	to resolve the fo
 Altera Avalon-ST Sink BFM 	<u> </u>	cpu_subsystem.cpu						
 Altera Avalon-ST Source BFM 	A 1	cpu_subsystem.jtag_u	Jart		Reset slave is	not specified. Pi	ease select the reset	slave
 Altera Clock Source BFM 	0 4	1 System Connectivity (Karping	11 6	Exception slar	ve is not specifie	d. Please select the e	xception slave
Altera Reset Source REM	· · ···	a system connectivity i	warning					
	A 1	cpu_subsystem.jtag_	Jart					
				11				
n ► ► ► ► ► ► ► ► ► ► ► ► ► ► ► ► ► ► ►								

- 6. In the **Vectors** tab, set **Reset vector memory** and **Exception vector memory** both to **onchip_ram.s1** to resolve the error messages.
- Click File ➤ Save to save the project. There is no need to generate the RTL for the Platform Designer system at this time. Click Move up one level of hierarchy to return to top_level.qsys system.

Figure 15. Move Up One Hierarchy Level



Platform Designer and Platform Designer (Standard) Differences

Platform Designer introduces a hierarchical isolation between system interconnect and IP components by saving the parameters of each IP component in a .ip file under <project folder>/ip/<Platform Designer system name> and saving of the





system interconnect in a .qsys file under <project folder>. The RTL of each .ip or .qsys file can be generated in isolation as it contains the full information required to reproduce the state of the RTL. There are no unresolved dependencies between files.

For example, Platform Designer saves the Nios II processor parameterization in <project folder>/ip/cpu_subsystem/cpu_subsystem_ nios2_gen2_0.ip, and the system interconnect in <project folder>/cpu_subsystem.qsys.

Figure 16. File Location for Nios II Processor IP File

🦄 Parameters 🕺	-				
System: cpu_subsystem Path: cpu					
HDL entity name:	(IP file: subsystem/cpu_subsystem_nios2_gen2_0.ip) en out to disk.				
Nios II Processor altera_nios2_gen2	Details				
Arithmetic Instructions MMU and MPU Main Vectors	Settings JTAG Debug Advanced Features Caches and M				
Reset Vector					
Reset vector memory.	onchip_ram.s1				
Reset vector offset:	0x0000000				
Reset vector:	0x00010000				
Exception Vector					
Exception vector memory.	onchip_ram.s1				
Exception vector offset:	0x0000020				
Exception vector:	0x00010020				
Fast TLB Miss Exception Vector					
Fast TLB Miss Exception vector memory.	None				
Fast TLB Miss Exception vector offset:	0x0000000				
Fast TLB Miss Exception vector:	0x0000000				

Platform Designer and Platform Designer (Standard) differ also differ in how they handle IP files:

- Platform Designer requires that you include the .qsys file along with a list of .ip files associated with that Platform Designer project. The Intel Quartus Prime Pro Edition software manages this for you after you save your Platform Designer project.
- The older Platform Designer (Standard) tool saves both component instantiation and system interconnects in a .qsys file. When integrating a Platform Designer (Standard) system to a Intel Quartus Prime project, you only need to include a single Intel Quartus Prime IP file (.qip).





Add External Memory Interface

The next step is to add an **Arria 10 External Memory Interfaces** component and use presets to configure the parameters.

The **Presets** tab displays a list of applications consisting of different protocols and development kits. You can choose from the list and apply a pre-defined set of parameters to the selected IP components. The DDR4 component from the list of **Presets** implements a pre-configured module. Modify the following parameters to help meet timing for this design:

- 1. Type external memory in the IP Catalog search box and double-click Arria 10 External Memory Interfaces to add it to the system.
- In the Arria 10 External Memory Interfaces parameter editor, select the Arria 10 GX FPGA Development Kit with DDR4 HILO from the Preset library and click Apply.

Memory Protocol	
Protocol: DDR4	•
General Memory Mem I,	O FPGA I/O Mem Timing Board Controller D
T EDC A	
Speed grade:	D. (Descharder) - shares destas under Milaud - (Destas 5
speed grade.	12 (Production) – change device under view –> Device F
Interface	
Configuration:	Hard PHY and Hard Controller
	nara minana nara controller
Instantiate two controllers	sharing a Ping Pong PHY
T. Clasks	
Mamony clock froquency	
Memory clock frequency.	800.0 MHz
📃 Use recommended PLL re	ference clock frequency
PLL reference clock frequenc	V. 100.0 🔽 MHz
PLL reference clock jitter	10.0
Clock rate of user logic:	10.0
Clock rate of user logic.	Quarter 🔽
Core clocks sharing:	No Sharing 👻
Specify additional care d	asks based on existing PLI
Specify additional core cit	JERS Dased on existing FLE

Figure 17. Arria 10 External Memory Interfaces Pane

3. In the **Clocks** section of the **General** tab, change **Memory clock frequency** to 800MHz and the **PLL reference clock frequency** value to 100MHz.





Figure 18. Memory Tab

ļ	General Memory Mem I/O FP	GAI/O Mem Timing Board Controller Diagnos
	Topology	
	Memory format:	UDIMM 👻
	DQ width:	32
	DQ pins per DQS group:	8 🗸
	Number of DQS groups:	4
	Number of clocks:	1 🗸
	Number of DIMMs:	1 -
	Number of physical ranks per DIMM:	1 -
	Number of chip selects per DIMM:	1
	Row address width:	15 🖵
	Column address width:	10 -
	Bank address width:	2 -
	Bank group width:	1 -
	🗹 Data mask	
	🔲 Write DBI	
	Read DBI	
	ALERT# pin placement:	I/O Lane with DQS Group
	DQS group of ALERT#:	0 -

- 4. Click the **Memory** tab and specify the following:
 - Change the **DQ width** to 32.
 - Turn off **Read DBI**.
 - Select '0' from the **DQS group of ALERT#** list.



Figure 19. Diagnostics Tab

-	Memory Protocol					
	Protocol: DDR4					
Ĺ	General Memory Mem I/O FPGA I/O Mem Timing Board Co	ontroller Diagnostics				
	Simulation Options					
	Calibration mode:	Skip Calibration 👻				
	Abstract phy for fast simulation					
	Calibration Debug Options					
	Quartus Prime EMIF Debug Toolkit/On-Chip Debug Port:	Add EMIF Debug Interface				
	Enable Daisy-Chaining for Quartus Prime EMIF Debug Toolkit/On-Chip	Debug Port				
	Interface ID:	0 💌				
Skip address/command leveling calibration						
	Skip address/command deskew calibration					
	Skip VREF calibration					
	Use Soft NIOS Processor for On-Chip Debug:	Disabled 👻				
ľ	* Example Design					
	Number of core clocks sharing slaves to instantiate in the example design:	1 -				
	Use a separate RZQ resistor for every sharing interface					
	✓ Enable In-System-Sources-and-Probes					

- 5. Click the **Diagnostics** tab and specify the following:
 - Turn on Skip address/command leveling calibration.
 - Turn on Skip address/command deskew calibration.
- 6. Click Finish.
- Right-click the name of the top_system_emif_0 component and click Rename. Type emif_0.
- In the Export column, double-click the mem, oct, and status conduit interfaces and rename them emif_0_mem, emif_0_oct, and emif_0_status, respectively.



Figure 20. Export Names for emif_0 Signals

ext_clk in_clk	Clock Bridge Clock Input	cik
out_cik	Clock Output	Double-click to export
🗆 ext_reset	Reset Bridge	
clk	Clock Input	Double-click to export
in_reset	Reset Input	reset
out_reset	Reset Output	Double-click to export
🗆 🖳 cpu_subsystem 🚽	cpu_subsystem	
cpu_clk	Clock Input	Double-click to export
cpu_jtag_debug_r	Reset Output	Double-click to export
cpu_reset	Reset Input	Double-click to export
master	Avalon Memory Mapped Master	Double-click to export
mem_clk	Clock Input	Double-click to export
mem_reset	Reset Input	Double-click to export
🗆 emif_0	Arria 10 External Memory Interf	
ctrl_amm_0	Avalon Memory Mapped Slave	Double-click to export
emif_usr_clk	Clock Output	Double-click to export
emif_usr_reset_n	Reset Output	Double-click to export
global_reset_n	Reset Input	Double-click to export
mem	Conduit	emif_0_mem
oct	Conduit	emif_0_oct
pll_ref_clk	Clock Input	Double-click to export
status	Conduit	emif_0_status

Instantiate a Memory Tester Subsystem as a Generic Component

At this stage of memory tester subsystem design, you can use Platform Designer to assemble the whole design to verify and debug components such as the Nios II logic resource usage and DDR4 Calibration.

You can instantiate the memory tester subsystem as a generic component (an empty entity with only interfaces defined). When integrating a memory tester subsystem with a processor subsystem and an EMIF controller only the interfaces of the memory tester subsystem are significant.

Instantiation of a generic component does not prevent the completion of other parts of the design. This feature provides a lot of flexibility in the design, and is especially beneficial for large and team-based designs. You need only verify that, when adding the entity implementation, the entity interfaces match the interfaces defined for the generic component.

To instantiate a generic component:

1. In the IP Catalog, double-click Generic Component.





Figure 21. IP Catalog Generic Component



The **Component Instantiation** tab contains three implementation types: **IP**, **HDL**, and **Blackbox**. When you add a generic component, **Blackbox** is the default.

2. Change the **HDL entity name** and **HDL compilation library** to memory_tester_subsystem.





Figure 22. Component Instantiation Tab for memory_tester_subsystem Generic Component

🎦 Component Instantiation 🛛		- d' 0
System: top_system Path: memory_tester_st	ubsystem	
Templates View Advanced		
Implementation Type:	HDL	Blackbox
Compilation Info 🛛		- 5 0
About Compilation Info		
HDL entity name: memory_tester_sub HDL compilation library: memory_tester_sub Assignments: Edit Signals & Ir System Infc Block Syn About Signals About Signals	osystem osystem mk 🕸 Implement 🕮 Export	× _ f □
About Signals Name clk Clock Input clk_clk [1] clk reset Reset Input - reset_reset [1] reset < add signal>> < add interface>>	Name: reset_re Signal Type: reset Width: 1 VHDL Signal Type: Standar Direction: input	set

- 3. To add the component, click **Finish**.
- 4. Right-click the name of the **top_system_generic_component_0** component and click **Rename**. Type memory_tester_subsystem.
- *Note:* When implementing a generic component with the **Blackbox** option, you don't have to provide the HDL implementation during component instantiation. Simply customize the interfaces and signals and generate an empty HDL file. Then, connect the generic component to other components in Platform Designer, generate interconnects, and finally, compile the project with this empty entity. When you finish the implementation of the generic component, simply replace the generic component with the actual implementation to complete the design. In other words, the generic component functions as a placeholder for the actual component you plan to use.

Platform Designer provides many features to help you add interfaces and signals for a generic component. The following steps, 1-11, showcase how to add signals manually, by using **Mirror** or **Clone**, and how to change parameters. In the final steps, you are going to import a complete interface definition from an .ipxact file.





Platform Designer provides many ways to help you add interfaces easily and efficiently.

- 1. Click View > Component Instantiation.
- 2. Select the **memory_tester_subsystem** component. The instantiation information appears in the **Component Instantiation** tab.
- 3. Click the **Signals & Interfaces** tab. You can add interfaces manually, **Import** from an IP-XACT file, **Mirror**, or **Clone** from existing interfaces in the system.
- 4. Click **<< add interface>>** and select **Clock Input** from the drop down list.
- 5. To change the name of the interface, in the **Name** field, type clk.
- 6. Click <<add signal>> and choose clk.
- 7. Repeat steps 4-6 to add a **Reset Input** interface and signal, and rename it reset.
- 8. Click Apply.

Figure 23. Signal and Interface Options for memory_tester_subsystem

🧏 Component Instantiation 🛛 🖾	- 6 5				
System: top_system Path: memory_test	er_subsystem				
Templates <u>V</u> iew <u>A</u> dvanced					
Implementation Type:	HDL Blackbox				
Compliation Info 88 ト About Compilation Info	- ď 🗆				
HDL entity name: memory_tester_subsystem HDL compilation library: memory_tester_subsystem Assignments: Edit					
Signals & Ir ⊠ System Infc ⊠ Bloc → About Signals	k Symit 🕮 Implement 🕮 Export 🕮 🗕 🗗 🗖				
Name Clk Clock Input clk_clk [1] clk reset Reset Input < <add signal="">> <<add interface="">></add></add>	Name: reset_reset Signal Type: reset Width: 1 VHDL Signal Type: standard logic Direction: input				
Import Clone Mirri	or				

Apart from clock and reset, the design also requires an Avalon-MM slave interface to communicate with the processor subsystem. It could be tedious to add Avalon-MM slave interface manually since there are address bus, data bus, and many other parameter settings to configure. An easier way is to use the **Mirror** feature.

9. Click **Mirror** and choose the master interface of **cpu_subsystem** to add a slave interface.



Signals & Ir 💥 System Info 🖾 Block Symb 🖾 Implement 🖾 Export 🖾 - 5 🗆 About Signals Name - clk Clock Input ➡ clk_clk [1] clk 🖿 reset Reset Input - reset_reset [1] reset < <add signal>> < <add interface>> Mirror.. Import... Clone.. cpu_subsystem cpu_clk Instantiation Messages 23 - d' 🗆 cpu_jtag_debug_reset Туре Path ge cpu_reset 1 Info Message የ 🕕 master No error mem_clk Mirror the interface 'master'. mem_reset emif_0 ctrl_amm_0 emif_usr_clk emif_usr_reset_n global_reset_n mem oct

Figure 24. Create Mirror of Interface 'master'

10. You can resolve the errors that appear in the **Instantiation Messages** box by assigning **Associated Clock** and **Associated Reset** to the clk and reset interfaces in the parameter editor.





About Signals Name Islave					
Name Slave					
De cik_cik [1] cik Freset Reset Input De reset reset Input Siave_datdress [16] address De slave_datdress [16] address De slave_byteenable [4] byteenable De slave_byteenable [4] byteenable De slave_readdata [32] readdata De slave_readdata [32] readdata De slave_writedata [32] readdata Siave_writedata [32] writedata < <add signal="">> <<add signal="">> </add>> </add> > > > > > > >					
Instantiation Messages 🛛 📃 🗖 🗖					
Type Path Message					
Y U L INTO MESSAGE					
vo errors or warnings.					

- 11. Locate the **Maximum pending read transactions** box under **Pipelined Transfers** and change that value to 4.
- 12. Click **Import** and choose memory_tester_subsystem_bb.ipxact to add the interfaces.
- 13. To complete the import step, click **Apply**.





Figure 25. Results of Importing memory_tester_subsystem_ bb.ipxact

nplementation Type:	IP	HDL	HDL Blackbox				
Compilation Info 🛛 😂 📘 F About Compilation II HDL entity name:	nfo memory_tester_subsystem				- = = :		
HDL entry name. Internory_tester_subsystem HDL compilation library. memory_tester_subsystem Assignments: Edit							
Signals & Inter 🙁 Sy 🕨 About Signals	stem Inform 🛛 🛛 Block Symbo	ol 🖾 Implement	atio ⊠ E>	port 🛛	- d` I		
 clk Clock Input clk_clk [1] clk read_master_axalon read_master_axalon read_master_axalon read_master_re read_master_re read_master_re read_master_re read_master_re read_master_re reset_reset [1] <add signal="">></add> slave_davlaces [slave_burstcour slave_burstcour slave_debugacc slave_read [1] / Import 	Name n Memory Mapped Master ddress [32] address yteenable [32] byteenable addata [326] readdata addata [256] readdata addatavalid [1] readdatavalid aitrequest [1] waitrequest reset ry Mapped Slave 13] address tt [1] burstcount le [4] byteenable exess [1] debugaccess read Clone Mirn	Name: Type: Associat Assignm Block	ted Clock: c ted Reset: r nents:	ead_master valon Memory Ma lk eset Edit read_master read_master read_master_ read_master_by read_master_by read_master_by	address[3] address[3] (d_master_i addat251 r_readdata teenable[3] ter_waitreg		
Instantiation Messages 🖾 🗕 🗗 🗖							
? 🕕 1 Info Mess	Vype Path Message Info Message						
0	No errors or warnings.						

Click the **Implementation** tab to create a Platform Designer template with interface requirements setup to implement the memory tester subsystem. You can also create an HDL template with ports defined.

- 14. Click **Create Platform Designer System Template ➤ Save** to create the memory_tester_ subsystem.gsys file in the <project folder>.
- 15. Click **Create HDL Template ➤ Save** to create the memory_tester_ subsystem.v file in the <project folder>.

The **Export** tab allows you to export the interfaces and requirements to an .ipxact or a _hw.tcl file, however, this feature is not used in this project.





Figure 26. Create Platform Designer System Template and HDL Template Options

Signals & Inte 🛛	System Inforn 🛛	Block Symbol 🛛	Files	X	Implementati	x	Export	X	=
Platform Designer System Template When you create a Platform Designer system template, an empty Platform Designer system will be created for you to use to implement this Generic Component. The Platform Designer system will be setup to have Interface Requirements which mirror the requirements of this Generic Component. In that way, when constructing the Platform Designer system, you can know whether what you've created matches the interface expectations of the parent system. It is important that the Generic Component and the template Platform Designer system's interfaces match exactly so that the fabric and connections created by the parent system work as expected. Create Platform Designer System Template									
HDL Template This will export an empty HDL entity with ports matching those of the Generic Component. Create HDL Template									

Related Information

Implement the Memory Tester Subsystem on page 31

Connect and Generate IP Files

You can make connections once the component instantiation is complete. Connect the source and target components with the entries in the following table:

Table 4. Top Level Platform Designer Connections

Source Component/Signal	Target Component/Signal
<pre>ext_clk/out_clk</pre>	 ext_reset/clk cpu_subsystem/cpu_clk emif_0/pll_ref_clk
ext_reset/out_reset	 cpu_subsystem/cpu_reset cpu_subsystem/mem_reset memory_test_subsystem/reset emif_0/global_reset_n
cpu_subsystem/master	 memory_test_subsystem/slave
cpu_subsystem /cpu_jtag_reset	 cpu_subsystem/cpu_reset cpu_subsystem/mem_reset memory_test_subsystem/reset emif_0/global_reset_n
memory_test_subsystem/read_master	• emif_0/ctrl_amm_0
<pre>memory_test_subsystem/write_master</pre>	• emif_0/ctrl_amm_0
emif_0/emif_usr_clk	 cpu_subsystem/mem_clk memory_test_subsystem/clk
emif_0/emif_usr_reset_n	• cpu_subsystem/mem_reset





Compare your completed system to the following figure:

Figure 27. Top Level Platform Designer System Connections



If there are any errors, read the error message and fix the error.

- 1. Click **File > Save** to save the top-level system.
- Click Generate ➤ Generate HDL and click Generate to generate RTL for each component, including components in the cpu_subsystem.
- Close Platform Designer. New files appear in the in the Project Navigator ➤ Files tab in the Intel Quartus Prime project. You must add another file memory_tester_subsystem.v. Adding this provides an empty entity for memory_tester_subsystem so Intel Quartus Prime Pro Edition can elaborate the hierarchy.
- 4. In the **Tasks** window, double-click **Add/Remove Files in Project** to open the **Settings** dialog box.
- 5. To add an empty memory_tester_subsystem.v file, type memory_tester_subsystem.v in the **File name** box.
- 6. Click Add.



Figure 28. top_system.qsys IP Files



 Compile the project by clicking Processing ➤ Start Compilation. If there are any errors, verify that all required files are present, and that you correctly name the exported ports in the Platform Designer system.

After compilation completes successfully, check the Compilation Reports (**Processing** ➤ **Compilation Report**) for **Logic Resource Usage**, **I/O Bank Usage**, **Clock tree**. You can also upload the A10.sof file generated during compilation to a board to check the calibration status of the DDR4 RAM. In the top_level.v file, sdram_cal_success, and sdram_cal_fail are connected to LED3 on the board. A green light indicates that calibration was successful. A red light indicates that calibration failed.

This design flow allows you to verify and debug DDR4 RAM calibration, while maintaining the system structure, before finishing the implementation of the memory tester subsystem.

Examples of Platform Designer Generic Components

You can instantiate components in Platform Designer using generic components.

Generic components fall into one of the three implementation types: **IP**, **HDL**, or **Blackbox**. Each type is selectable by the corresponding button in the **Component Instantiation** tab. All the _hw.tcl based IP components found in the IP Catalog, such as On-chip Memory and External Memory Interfaces (EMIF), belong to the IP type. If you want to add a custom component written in RTL, you can use the HDL type and link the source files in the **Component Instantiation** tab.





Figure 29. Example of an IP Component Instantiation

Implement the Memory Tester Subsystem

Next, you implement the memory tester subsystem (previously instantiated as a generic component) using the Platform Designer template.

You typically perform this process as a member of a remote team with a need to implement the memory tester subsystem. The remote team member receives a .qsys file which serves as the requirement hand off for an implementation. This .qsys file contains the details needed for designing a block for the larger design, without access to the top level.

To implement the memory tester subsystem you must add components from the IP Catalog to this Platform Designer project, make connections, and export interfaces to match what is defined for the generic component. Once those processes are complete, replace the generic component in the top level system with this subsystem implementation.

To implement the subsystem, complete the following steps:

- 1. To launch Platform Designer, click **Tools > Platform Designer**.
- 2. Browse to the memory_tester_subsystem.qsys file and click **Open**.

Platform Designer opens and displays an empty project. However, it embeds the interface requirements you defined in the generic component representation within the top level system used as a guide to implement this subsystem.

3. To view these interfaces, click **View ➤ Interface Requirements**.

The left column shows the interfaces instantiated in the current Platform Designer (Standard) Pro system. The right column shows the requirements you define in previous steps. Since there are no components or exported interfaces, all the interface names are highlighted in green, denoting missing items.

Figure 30. Interface Requirements Dialog Box

System Contents 🛛 😂	Address Map	x 🛃	Interface	Requirements	83	Details	83	
 About Interface Requirements 								
Current System >> Interface Requirements								
(Current system has no exports) Clk Clock Input Clk_Clk [1] Clk read_master_Avalon Memory Mapped Maste read_master_byteenable [32] byteenable read_master_byteenable [32] byteenable read_master_read(1) read read_master_read(1) readdata read_master_readdata (256) readdata read_master_readdata (256) readdata read_master_readdata (1) readdata read_master_waitrequest [1] waitrequest < add signal>> reset Reset Input reset reset [1] reset < add signal>> slave_Avalon Memory Mapped Slave slave_byteenable [4] byteenable clave_downeeses debuggeeses debuggeeses								
Legend: missing items	s different iten	75		Import Interfa	ace R	Requireme	ents	
(No selection)								
Parameter Differenc	es							
Parameter	Name	Cu	rent Syst	em Value	li	nterface l	Requ	irement Value
(No differences)								
Parameters	************						.*.*.*.*.*.*.*.*	

Add Clock, Reset, and Avalon-MM components

In order to resolve missing items in the **Interface Requirements** list, add a clock bridge, reset bridge, and Avalon-MM pipeline bridge first:

- 1. In the IP Catalog, type clock in the search box and double-click **Clock Bridge**.
- 2. Click **Finish** to add the clock bridge.
- 3. Type reset in the search box and double-click **Reset Bridge**.
- 4. Click **Finish** to add the reset bridge.
- 5. In the System Components tab, right-click the name of the clock bridge and click **Rename**. Type clk.
- 6. In the System Components tab, right-click the name of the reset bridge and click **Rename**. Type reset.
- 7. In the **Export** column, double-click the entry corresponding to the clock input for the clk component and rename it clk.
- 8. In the **Export** column, double-click the entry corresponding to the reset input for the reset component and rename it reset.



- 9. Type pipeline in the IP Catalog search box and double-click Avalon-MM Pipeline Bridge.
- 10. When the **Avalon-MM Pipeline Bridge** parameter editor opens, in **Address**, set the **Address width** to 13.
- 11. To add the component, click **Finish**.
- 12. Right-click the name of the **Avalon-MM Pipeline Bridge** and click **Rename**. Type mm_bridge.

Add Pre-Built Systems and Memory Test Microcore Components

The files listed in the *Design Files* topic contain two pre-built systems: a pattern checker system, and a pattern generator system. These pre-built Platform Designer systems appear in the IP Catalog in the System folder. The IP Catalog also contains a list of available Memory Test Microcores. The source files of these custom IP cores are located in cproject folder>/memory_tester_ip. The

memory_tester_search_path.ipx file included in the project provides this path to
Platform Designer.

Figure 31. IP Catalog Memory Test Microcore and System Components







To add these pre-built Platform Designer systems, complete the following steps:

- 1. In the IP Catalog, expand the **System** folder and double-click **pattern_checker_system** to add the pattern checker component.
- 2. To add the component, click **Finish**.
- 3. To rename the pattern checker, right-click the system in the **Name** column and type pattern_checker_subsystem.
- 4. In the IP Catalog, double-click **pattern_generator_system** to add the prebuilt pattern generator component.
- 5. To add the component, click **Finish**.
- 6. To rename the pre-built pattern generator, right-click the system in the Name column and type pattern_generator_subsystem.
- 7. In the IP Catalog, expand the **Memory Test Microcores** folder, and double-click **Pattern Writer**.
- 8. In the **Pattern Writer** parameter editor, turn on **Burst Enable**.
- 9. To add the component, click Finish.
- 10. To rename the pattern writer component, right-click the system in the **Name** column and type pattern_writer.
- 11. In the IP Catalog, double-click **Pattern Reader**.
- 12. In the Pattern Reader parameter editor, turn on Burst Enable.
- 13. Click **Finish** to add the component.
- 14. In the IP Catalog, double-click to begin adding a RAM Test Controller.
- 15. Click **Finish** to add the component.

Related Information

Download and Install the Tutorial Design Files on page 5

Export Signals, Set Base Address Assignments, and Connect Memory Tester Interface Components

To export signals, set base address assignments, and connect components, perform the following steps:

- To export the Avalon Memory Mapped Master interface for Pattern Writer, in the Export column double-click the row adjacent to the Avalon Memory Mapped Master and type write_master.
- To export the Avalon Memory Mapped Master interface for Pattern Reader, in the Export column, double-click the row adjacent to the Avalon Memory Mapped Master and type read_master.
- 3. Make connections for the system based on the following table:





Source Compont/Signal	Target Component/Signal
clk /out_clk	 reset/clk mm_bridge/clk pattern_generator_subsystem/clk pattern_checker_subsystem/clk pattern_writer/clock pattern_reader/clock ram_test_controller/clock
reset /out_reset	 mm_bridge/reset pattern_generator_subsystem/reset pattern_checker_subsystem/reset pattern_writer/reset pattern_reader/reset ram_test_controller/reset
mm_bridge/m0	 pattern_generator_subsystem/slave pattern_checker_subsystem/slave ram_test_controller/csr
pattern_generator_subsystem/st_data_out	• pattern_writer/st_data
pattern_reader/st_data	 pattern_checker_subsystem/st_data_in
ram_test_controller/read_command	• pattern_reader/command
ram_test_controller//write command	• pattern_writer/command

Table 5. Memory Tester Interface Component Connections

- 4. Assign base addresses for Avalon Memory Mapped Slave interfaces:
 - a. In the **Base** column, click the value for slave signal of the **pattern_generator_subsystem** component and type 0000.
 - b. In the Base column, click the value for slave signal of the pattern_checker_subsystem component and type 1000.
 - c. In the **Base** column, click the value for csr signal of the ram_test_controller component and type 800.

The following figure shows the completed system:



Description Clock Bridge Clock Input Clock Output Reset Bridge Clock Input Reset Input Reset Output Avaion-MM Pipeline Bridge Clock Input Clock Use Connections Name Export Base End 🗆 dk Cik in_cik out_cik reset cik in_reset out_reset mm_bridge clk exported clk_out_clk V clk_out_clk 0reset [clk] [clk] V Avaion-MM Pipeline Bridge Clock Input Avalon Memory Mapped M. Reset Input Avalon Memory Mapped SI. clk m0 reset clk_out_clk [clk] [clk] [clk] Avaion Memory Mapped SL. pattern_generator_system Clock Input Reset Input Avaion Memory Mapped SL. Avaion Streaming Source pattern_checker_system Clock Input Reset Input Avaion Memory Mapped SL. Avaion Streaming Sink Pattern Writer Clock Input V 🗉 🛄 pattern_generator_subsystem clk_out_clk clk reset slave [clk] [clk] 0x0000 x07ff e 별 pattern_checker_subsystem V clk_out_clk reset [clk] [clk] ▲ 0×1000 0x17ff st data in V pattern_writer Pattern writer Clock Input Avalon Streaming Sink Avalon Memory Mapped M.. Reset Input clock clk_out_clk command [clock] á [clock] mm_data write_master reset Reset Input Avaion Streaming Sink Pattern Reader Clock Input Avaion Streaming Sink Avaion Streaming Sink Avaion Memory Mapped SI. Avaion Streaming Source RAM Test Controller Clock Input Avaion Memory Mapped SI. [clock] st_data pattern_reader [clock] V clock command mm_data reset st_data clk_out_clk [clock] [clock] [clock] [clock] 9 read_master st_data Tam_test_controller clock csr V rik out rik [clock] [clock] 0x0800 0x081f read command Avalon Streaming Source reset write_command Reset Input [clock] Avalon Streaming Source [clock]

Figure 32. Connections and Base Address Values for the memory_tester_sybsystem

Resolve Interface Requirements and Value Mismatches

In the **Interface Requirements** tab you can verify that the exported interfaces meet the interface requirements.

1. Click the Interface Requirements tab in Platform Designer.

The exported interfaces in the tutorial system appear in the **Current System** list. The **Interface Requirements** list shows the definition of the generic component. A green highlight indicates a missing item. A blue highlight indicates an item with parameter mismatches.



System Contents 🛛 Address Map 🔅 Interconnect Requirements 🖇 🛃 Interface Requirements 🕮 - d' 🗆 About Interface Requirements >> Interface Requirements Current System - clk Clock Inpu - clk Clock Inp 🕞 clk_clk [1] clk ➡ clk_clk [1] clk read_master Avalon Memory Mapped Master read_master_address [32] address read_master_Avalon Memory Mapped Master memory master_address [32] address read_master_burstcount [2] burstcount 🛥 read_master_burstcount [2] burstcoum read_master_byteenable [32] byteenable read_master_byteenable [4] byteenable - read_master_read [1] read - read_master_read [1] read read_master_readdata [256] readdata read_master_readdatavalid [1] readdatavalid 🖙 read_master_readdata [32] readdata - read_master_readdatavalid [1] readdatavalid - read_master_waltrequest [1] waltrequest - read_master_waitrequest [1] waitrequest reset Reset Input < <add signal>> - reset_reset_n [1] reset_n reset Reset Input write_master Avalon Memory Mapped Master - reset_reset_n [1] reset_n - write_master_address [32] address < <add signal>> - write master_burstcount [2] burstcount slave Avalon Memi v Manned Slave - write_master_byteenable [4] byteenable - slave_address [13] address write_master_waitrequest [1] waitrequest write_master_write [1] write slave_burstcount [1] burstcount slave_byteenable [4] byteenable 🕶 write_master_writedata [32] writedata - slave_debugaccess [1] debugaccess - slave_read [1] rea - slave_readdata [32] readdata - slave_readdatavalid [1] readdatavalid - slave_waitrequest [1] waitrequest slave_write [1] write slave_writedata [32] writedata add sionals s write_master Avalon Memory Mapped Master write_master_address [32] address - write_master_burstcount [2] burstcount - write_master_byteenable [32] byteenable write_master_waitrequest [1] waitrequest write_master_write [1] write 🖛 write_master_writedata [256] writedata < <add signal>. <add interface>> Legend: missing items different items Import Interface Requirements...

Figure 33. Missing Components and Value Mismatches

- 2. View the **Interface Requirements** list for missing items. What appears in the figure indicates a missing slave interface of the pipeline bridge. Fix the missing items by exporting the appropriate signal.
- 3. In the **System Contents** tab, double-click the entry in the **Export** column corresponding to the s0 for the **mm_bridge** component and rename it to slave.

Figure 34. Export and Rename Avalon-MM Slave



4. Re-examine the **Interface Requirements** tab. The **Current System** list contains the slave interface with no green highlight. Next you resolve the different item highlighted in blue.



urrent System	>> Interface Requirements
- dk Clock Input	clk Clock Input clk clk clk l11 clk
read master Avalon Memory Manned Master	read master Avalon Memony Manned Master
read master address [32] address	read master address [32] address
read_master_burstcount [2] burstcount	read master burstcount [2] burstcount
read master byteenable [4] byteenable	read master byteenable [32] byteenable
- read master read [1] read	- read master read [1] read
read master readdata [32] readdata	read master readdata [256] readdata
- read_master_readdatavalid [1] readdatavalid	read_master_readdatavalid [1] readdatavalid
read_master_waitrequest [1] waitrequest	read_master_waitrequest [1] waitrequest
reset Reset Input	< < add signal>>
reset_reset_n [1] reset_n	reset Reset Input
 slave Avalon Memory Mapped Slave 	reset_reset_n [1] reset_n
slave_address [13] address	< <add signal="">></add>
slave_burstcount [1] burstcount	slave Avalon Memory Mapped Slave
slave_byteenable [4] byteenable	slave_address [13] address
slave_debugaccess [1] debugaccess	slave_burstcount [1] burstcount
slave_read [1] read	slave_byteenable [4] byteenable
📹 slave_readdata [32] <i>readdata</i>	slave_debugaccess [1] debugaccess
- slave_readdatavalid [1] readdatavalid	slave_read [1] read
- slave_waitrequest [1] waitrequest	slave_readdata [32] readdata
slave_write [1] write	slave_readdatavalid [1] readdatavalid
slave_writedata [32] writedata	slave_waitrequest [1] waitrequest
write_master Avaion Memory Mapped Master	slave_write [1] write
write_master_address [32] address	slave_writedata [32] writedata
write_master_burstcount [2] burstcount	< add signal>>
write_master_byteenable [4] byteenable	- write_master Avaion memory mapped master
write_master_watrequest [1] watrequest	write_master_address [32] address
Write_master_writedate_[23] writedate	write master byteenable [23] byteenable
- wheelmaster_wheelata [52] whitebala	write_master_pyteenable [32] byteenable
	- write master write [1] write
	Write master writedata [256] writedata
	< cadd signals >
	< < add interface > >
	COMPACTION OF ACCOUNT OF A

Figure 35. Current System / Interface Requirement Value Mismatch

5. Click the signal name highlighted in blue to display more information in the **Parameter Differences** pane. Typically, you change the **Current System Value** to match the **Interface Requirement Value** by editing the parameters of that component.

Figure 36. Changing Current System Value

stem Contents 🛛 🕅	Address Map 🛛 Ir	terconnect Require	ments 🖾	🖆 Interface Requirer	nents 🖾	_ _ _
About Interface R	equirements					
urrent System				>> Interface Requi	rements	
 dk Clock Input clk_clk [1] clk read_master Ava read_master. read_master. read_master. read_master. read_master. read_master. read_master. read_master. read_master. 	Ion Memory Mapped Mas address [32] address burstount [2] burstount byteenable [4] byteenab read [1] read readdata [32] readdata readdata [32] readdata waitrequest [1] waitrequ	ter t le ttavalid est		► clk Clock Inp ► clk clock Inp ► clk.clk [1 ◄ read_master ◄ read_master ► read_master	ut) CIK Avalon Memory Mapped Maste ster_bartcount [2] burstcount ster_bystcount [2] burstcount ster_steract [1] byseenably ster_read/data [256] readdata ster_read/data [256] readdata ster_read/data [1] readdata ster_waitrequest [1] waitreques ab>>	r e tvalid t
- reset_reset_n - slave Avalon Men - clave addrese - slave missing iter - gnal: read master	(1) reset_n nory Mapped Slave (112) address msdifferent kems read_master_readdata		-	Import Interface	et_n [1] reset_n	
- reset_reset_n - slave Avalon Mer - slave Avalon Mer - slave addres - segend: missing iter - gnal: read_master Parameter Differer	(1) reset,n nory Mapped Slave e 1121 ordenoce ms different items read_master_readdata tces			► reset_res	et_n [1] reset_n	
reset_reset_n slave Avalon Mer slave Avalon Mer grad: missing iter gnal: read_master Parameter Differen Pa	(1) reser, n nony Mapped Slave e. 1121 address msdifferent items read_master_readdata tices arameter Name		Current	Preset_res Import Interface	et_n [1] reset_n :Requirements Interface Requ	irement Value
reset_reset_n slave Avalom Mer slave Avalom Mer dese Avalom Mer gnal: read_master. Parameter Differer Parameter Differer Parameter Differer	(1) reset.n nory Mapped Slave (12) addense ms_different Rems read_master_readdata tces arameter Name	32	Current	Import Interface	er_n [1] reset_n Requirements Interface Requ 256	irement Value
preset_reset_n slave Avalon Mer slave	La reser, n norw Mapped Slave Slave Slave man different items read, master_readdata nces urameter Name	32	Current	D- reset, res c ended clina Import Interface	er_n [1] reset_n Requirements Interface Requ 256	irement Value
reset_reset_n save Avalom Mer save Avalom Mer dens Avalom Mer gnal: read_master. Parameter Differen Pr Width Current Parameter Name:	14) reser,n norw Napped Slave Slave Slave statistics manual states read,master_readdata tices wameter Name s read,master_readdata	32	Current	System Value	Requirements Requirements Interface Requ 256	jrement Value
reset_reset_n reset_reset_reset_n reset_reset_reset_n reset_reset_reset_n reset_reset_reset_n reset_reset_reset_reset_n reset_r	I I rest.n norw Mapped Slave Slave Slave read_master_readdata rccs read_master_readdata rccs read_master_readdata readdata	32	Current	System Value	er_n [1] reset_n	irement Value
reset_reset_n setup exection setup exection development development reset_reset_n development reset_reset_n reset_reset_reset_n reset_reset_reset_n reset_reset_reset_n reset_reset_reset_n reset_reset_reset_reset_reset_n reset_reset	14 / rest.n nory Mapped Slave 5121. addresse mail different Rems read_master_readdata rccs read_master_readdata feaddata 32	32	Current	System Value	er_n [1] reset_n	irement Value
seset_reset_n seset_reset_n slave_Avalon Mer slave_Avalon Mer dawa_valon Mer dawa_valon Mer gradtmaster gradtmaster parameter Differer Width Current Parameter Name: Signal Type: Width. VHOL Signal Type:	I I rest.n norv Mapped Slave CI32. nddesce. ms: different Rems read_master_readdata rces read_master_readdata readdata 32 standard logic vector	32	Current	System Value	et_n [1] reset_n	irement Value



- Click read_master_readdata[32] and examine the Parameter Differences pane. In the top_system, the data width of the Avalon Memory Mapped Master of the EMIF controller is 256. The data width of the memory_tester_subsystem must match with a value of 256. Adapters inserted to handle data width mismatch may become the bottle-neck of a design.
- This exported interface comes from the Pattern Writer. To alter its width, alter the parameters of that IP core. To change the data width of **Pattern Writer**, doubleclick the **pattern_writer** component. Change the **Data Width** in the parameter editor to 256.

Figure 37. Pattern Writer Settings Dialog Box

pattern_writer	<u>D</u> etails
* Parameters	
Transfer Length Register Width: 21 👻	
Data Width: 256 👻	
Internal FIFO Depth: 128 👻	
🖌 Burst Enable	
Maximum Burst Count: 2 👻	
🗹 Enable Burst Re-alignment	

Repeat the Step 7 for the **pattern_reader** component.

Figure 38. Pattern Reader Settings Dialog Box

Pattern Reader pattern_reader	Details
 Parameters 	
Transfer Length Register Width:	21 🗸
Data Width:	256 👻
Internal FIFO Depth:	128 👻
🖌 Burst Enable	
Maximum Burst Count:	2 👻
🗹 Enable Burst Re-alignment	

These parameter changes alter the width of *_byteenable signals accordingly.

8. Verify that your Interface Requirements tab contains no missing items or mismatched items. In cases where you want to keep the current system value, you can click the **Copy** button to copy items from the left table to the right.



Figure 39. Completed Interface Requirements

ystem Contents 🖾 Address Map 🖾 Interco	iect Requirements 🛛 🕅	12	Interface Requirements 🛛 🎘	
 About Interface Requirements 				
Current System		>>	Interface Requirements	
dk Clock input Clock input Clk, clk [1] (ik read_master_avalon Memory Mapped Master read_master_isvalon Memory Mapped Master read_master_isvalon Memory Mapped Master read_master_isvalon Memory Mapped Master read_master_burstount [2] burstount read_master_isvalon Memory Mapped Master read_master_readdta[256] readdata read_master_readdta[256] readdata read_master_readdta[1] waitrequest read_master_readdta[1] waitrequest read_master_readdta[1] waitrequest read_master_readdtasul[1] waitrequest read_master_readdtasul[1] waitrequest slave_address [13] address idave_burstount [1] burstount diave_taddtasul[1] readdataval diave_taddtasul[1] readdataval diave_waitrequest [1] waitrequest slave_read[1] readdataval diave_waitrequest [1] waitrequest slave_waitrequest [1] waitrequest diave_waitrequest [1] waitrequest slave_waitrequest [1] waitrequest write_master_burstount [2] burstount write_master_datasul[2] readdatavall diave_vaitrequest [1] waitrequest write_master_datasul[2] predatasul write_master_burstount [2] burstount write_master_datasul[2] predatasul diave_vaitrequest [2] burstount write_master_burstount [2] burstount write_master_burstount [2] burstount write_master_burstount [2] burstount write_master_waitrequest [3] waitrequest write_master_waitrequest [3] waitrequast write_master_wai			dk Clock Input bclk_Ck[1] (k) read_master_axilon read_master_axilon read_master_axilon read_master_axilon read_master_byter read_master_byter read_master_read read_master_read read_master_read read_master_read read_master_read read_master_read read_master_set slave_adoress[13] slave_adoress[13] slave_adoress[13] slave_adoress[13] slave_adoress[13] slave_adoress[13] slave_read[1] read sl	lemory Mapped Master ress [32] address toom [2] bustcomt enable [32] byteenable [1] read data [256] readdata ddatavalid [1] readdatavalid request [1] waitrequest reset_n Mapped Slave] address [1] durstcount 4] byteenable [1] durstcount 4] byteenable [2] readdatavalid [1] waitrequest te 2] writedata Memory Mapped Master ress [32] address ttoom [2] burstcount enable [32] byteenable request [1] waitrequest tcl) write data

This completes the editing of component parameters to validate interface requirements.

- 9. Save and close the project. There is no requirement to generate HDL because we are replacing the generic component in top_system.qsys with the implemented subsystem.
- 10. Close Platform Designer and inspect the **Files** tab in the Project Navigator. Files for the memory_tester_subsystem are present in the Intel Quartus Prime Pro Edition project.

Figure 40. Files List for memory_tester_subsystem.v



Replace the memory_tester_subsystem Generic Component

Next, replace the generic component with the <code>memory_tester_subsystem</code> implementation:

- Click Tools ➤ Platform Designer to launch Platform Designer. Browse to the top_system.qsys file and click Open.
- 2. Right-click the memory_tester_subsystem component and click **Remove**.
- In the IP Catalog, browse to the System folder and double-click to memory_tester_subsystem. Keep the same name and update the connections.
- 4. Right-click the name of the top_system_subsystem_0 and click Rename. Type memory_tester_subsystem.
- 5. Verify and complete connections based on the following table:

Source Component/Signal	Target Component/Signal
<pre>ext_clk/out_clk</pre>	 ext_reset/clk cpu_subsystem/cpu_clk emif_0/pll_ref_clk
ext_reset/out_reset	 cpu_subsystem/cpu_reset cpu_subsystem/mem_reset memory_tester_subsystem/reset emif_0/global_reset_n







Source Component/Signal	Target Component/Signal
cpu_subsystem /cpu_jtag_debug_reset	 cpu_subsystem/cpu_reset cpu_subsystem/mem_reset memory_tester_subsystem/reset emif_0/global_reset_n
cpu_subsystem/master	• memory_tester_subsystem/slave
<pre>memory_tester_subsystem/read_master</pre>	• emif_0/ctrl_amm_0
<pre>memory_tester_subsystem/write_master</pre>	• emif_0/ctrl_amm_0
emif_0/emif_usr_clk	 cpu_subsystem/mem_clk memory_tester_subsystem/clk
emif_0/emif_usr_reset_n	• cpu_subsystem/mem_reset

6. Compare the connections to the following figure:

Figure 41. memory_tester_subsystem Implementation Connections



- 7. Click **File ≻ Save**.
- 8. Click Generate ➤ Generate HDL.
- 9. Click Generate.

10. Close the current Platform Designer project when generation is done.

The files in included with this design are Verilog (.v) files, but you can also use VHDL (.vhdl) in your design if you prefer.

Synchronize IP Results

When you synchronize IP files, Platform Designer checks IP file references.

- 1. In the Intel Quartus Prime Pro Edition, click **Files** in the Project Navigator and browse to memory_tester_subsystem.v.
- Delete the empty entity RTL memory_tester_subsystem.v since we now have the actual memory_tester_subsystem implementation.

Send Feedback



The files included in memory_tester_subsystem are not complete though. We are missing IP components in the pattern generator system and pattern checker system.

3. Open the pattern_generator_system.qsys and pattern_checker_system.qsys in Platform Designer and save them without generating HDL. This designates the IP components in these systems for elaboration during compilation.

Each time you open a Platform Designer project, Platform Designer automatically checks the IP file references and opens a dialog box if there is any mismatch. In the following figure, **IP Synchronization** detects the Platform Designer system includes these IP, but the Intel Quartus Prime Pro Edition project does not. This dialog box informs you when you must add these files to the project..

Figure 42. IP Synchronization Dialog Box

A 🕢 IP Synchronization Result <@sj-slscf014>	\odot	×
Synchronizing IP files between Qsys system and Quartus project		
Modified Modules Image: System's IP file reference added to Quartus project revision (.qsf). Image: Custom_pattern_generator Image:)_17p0/ip/ 0_17p0/ip/	patte
Export Report To File	Can	cel

- 4. Click **OK** and the Intel Quartus Prime Pro Edition synchronizes the file references.
- 5. Examine the Project Navigator and these new files appear:

Figure 43. Files Added through IP Synchronization

- 🖶 📥 ip/pattern_generator_system/pattern_generator_system_mm_bridge.ip
- 🗈 🚣 ip/pattern_generator_system/pattern_generator_system_two_to_one_st_mux.ip
- 🖶 📥 ip/pattern_generator_system/pattern_generator_system_clk_0.ip
- 🗄 📥 ip/pattern_generator_system/pattern_generator_system_prbs_pattern_generator.ip
- 🖻 👗 ip/pattern_generator_system/pattern_generator_system_custom_pattern_generator.ip
- 🗄 💑 ip/pattern_checker_system/pattern_checker_system_custom_pattern_checker.ip
- ip/pattern_checker_system/pattern_checker_system_clk_0.ip
- 🗄 👗 ip/pattern_checker_system/pattern_checker_system_one_to_two_st_demux.ip
- 🐵 📥 ip/pattern_checker_system/pattern_checker_system_mm_bridge.ip
- 🗄 💑 ip/pattern_checker_system/pattern_checker_system_prbs_pattern_checker.ip
- 6. Click **Processing ➤ Start Compilation** to compile the project. The Intel Quartus Prime Pro Edition software may return missing file errors, for example:

"Instance ` abc|def|ghi ' instantiates undefined entity ` xyz ' " This type of error is caused when an expected IP file is missing. Resolve it by adding the xyz.ip file to the project.







Build Software Applications and Download the Design

The final steps in this tutorial show how to download your design onto a Dev Kit board.

Hardware setup

First, you reprogram the clock generator chip on the board.

The default clock resource in this design runs at 133.33 MHz. Program the clock to run at 100 MHz.

- 1. Connect the board to the host PC with a USB cable and apply power to the board.
- Run the ClockController.exe executable that installs with the Dev Kit package. This executable installs to <package installation folder>/ examples/board_test_system by default.
- 3. Click the **Si5338(U26)** tab.
- 4. Change the frequency setting for CLK3 to 100MHz.
- 5. Click Set.

Figure 44. Clock Controller Settings

😔 Clock Cor	ntroller			×
SI570(X3) S	5338(U26) Si533	8(U14)		
Register		Frequence	cy (MHz)	Disable All
CLK0	270.0003	CLK0	270.0003	Disable CLK0
CLK1	644.5313	CLK1	644.5313	Disable CLK1
CLK2	644.5313	CLK2	644.5313	Disable CLK2
CLK3	99.9998	CLK3	99.9998	Disable CLK3
F_vco: 2578	.1250 MHz			
	Default	Read	Set	Import
Messages Connected to the target				



Related Information

Hardware and Software Requirements on page 5

Run the Bash Script

Nios II EDS enables you to build board support packages (device drivers, HAL) and applications based on the top_system.sopcinfo file, an output file of top_system.qsys generation.

- 1. Copy top_system.sopcinfo from /top_system to /<project folder>.
- To launch the Nios II Command Shell from Platform Designer, click Tools ➤ Nios II Command Shell (gcc4).
- In the Nios II Command Shell, browse to <project folder>/software, and run batch_script.sh.

Figure 45. Run the nios2_command_shell.sh

🕅 🕐 nios2_command_shell.sh <@sj-slscf014> 💿 🔗 🛞
Altera Nios2 Command Shell [GCC 4]
Version 17.0, Build 290
/data/jlsun/project/tt_qsys_design/Qsys_Pro_tutorial_design_Arria_10_17p0\$ cd so ftware /data/jlsun/project/tt_qsys_design/Qsys_Pro_tutorial_design_Arria_10_17p0/softwa re\$./batch_script.sh nios2-bsp: Using /swip_build/releases/acds/17.0/290/linux64/nios2eds/sdk2/bin/bs p-set-defaults.tcl to set system-dependent settings. nios2-bsp: Creating new BSP because ./bsp/settings.bsp doesn't exist. nios2-bsp: Running "nios2-bsp-create-settingssocp/top_system.sopcinfoty pe halsettings ./bsp/settings.bspbsp-dir ./bspscript /swip_build/releas es/acds/17.0/290/linux64/nios2eds/sdk2/bin/bsp-set-defaults.tcl default_sections _mapping cpu_subsystem_onchip_ramset hal.max_file_descriptors 4set hal.sys _clk_timer noneset hal.timestamp_timer noneset hal.enable_exit trueset hal.enable_c_plus_plus falseset hal.enable_clean_exit trueset hal.enable_r educed_device_drivers trueset hal.enable_lightweight_device_driver_api true - set hal.enable_small_c_library trueset hal.enable_sim_optimize falseset hal. al.make.bsp_cflags_optimization -02"

The batch_script.sh script calls commands in Nios II EDS to build a board support package and applications. The script then configures the FPGA with the A10.sof file that you generate during Intel Quartus Prime software compilation, runs the software applications, and establishes a terminal connection with the board. The test software performs test sweeps, such as Walking Ones, Walking Zeros, and PRBS, on the SDRAM and the output values appear in the command terminal.





Figure 46. Terminal Connection Console

	-	×
Info (209017): Device 1 contains JTAG ID code 0x02E060DD Info (209007): Configuration succeeded 1 device(s) configured Info (209011): Successfully performed operation(s)		^
Info (2090b1): Ended Programmer operation at wed May 1/ 16:21:18 201/ Info: Quartus Prime Programmer was successful - 0 errors - 0 warnings		
Info: Peak virtual memory: 1502 megabytes		
Info: Processing ended: Wed May 17 16:21:18 2017		
Info: Elapsed time: 00:00:33		
Info: Total CPU time (on all processors): 00:00:16		
Using cable "USB-BlasterII [USB-1]", device 1, instance 0x00		
Resetting and pausing target processor: OK		
Initializing CPU cache (1+ present)		
Downloaded 7KB in 0 0s		
Verified OK		
Starting processor at address 0x00010020		
nios2-terminal: connected to hardware target using JTAG UART on cable		
nios2-terminal: "USB-BlasterII [USB-1]", device 1, instance 0		
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)		
Starting test.		
0x10000000 bytes transferred in 0x8000126 clock cycles		
0x10000000 bytes transferred in 0x800022a clock cycles		
0x10000000 bytes transferred in 0x800045a clock cycles		
0x10000000 bytes transferred in 0x800080a clock cycles		
0x10000000 bytes transferred in 0x000104 clock cycles		
0x10000000 bytes transferred in 0x800210e total cycles		
0x10000000 bytes transferred in 0x800869c clock cycles		
0x10000000 bytes transferred in 0x83c297e clock cycles		~

The <project folder>/software folder contains a rerun.sh script. You can run this script when you already have the Nios II board support package and applications built, and don't need to build them again. This script downloads only the .sof file and runs Nios II applications.

AN 812: Platform Designer System Design Tutorial Revision History

Document Version	Changes
2018.04.02	Updated for terminology change from Qsys Pro to Platform Designer.
2018.05.04	Updated
2017.08.15	Initial release.

