Vivado Usage Notes

Version 1.5

Revision History

Revision	Description of Change	Date
v1.2	Initial creation for Release 1.2	8/2017
v1.4	Update for Release 1.4	9/2018
v1.5	Update for Release 1.5	4/2019

Table of Contents

1	References	4
2	Migrating an OpenCPI Platform from ISE to Vivado	4
3	Reverting an OpenCPI Platform from Vivado to ISE	4
4	Vivado Constraints Files	5
5	Using PreBuilt Cores/Netlists with Vivado and OpenCPI	5
6	Simulating Vivado IP or PreBuilt Cores with XSIM in OpenCPI	5
7	Using Vivado IP with OpenCPI	5
8	Makefile options for Vivado/XSIM compilation 8.1 Incremental Compilation - Place/Route 8.2 Synthesis Options : Applies to primitives, workers, platforms, configs, assemblies, containers 8.3 Enabling Optimization Stages 8.4 Implementation Options : Applies to Containers 8.5 XSIM Options : Applies to primitives, workers, platforms, configs, assemblies, containers	6 6 6 7 7
9	Global Tcl Initialization Scripts	7
10	Opening up designs in the GUI 10.1 EDIF Netlist 10.2 Project File 10.3 Implementation Design Checkpoint 10.4 Interactive Timing Report 10.5 Elaborated XSIM design 10.6 Open XSIM Wave Database	7 7 8 10 11 12 13
11	OpenCPI Output Files for Vivado	14
12	OpenCPI Output Files for XSIM	15

1 References

This document assumes a basic understanding of the Linux command line (or "shell") environment. A working knowledge of OpenCPI is required for understanding what vendor tools are necessary to perform various OpenCPI operations. The reference(s) in Table 1 can be used as an overview of OpenCPI and may prove useful.

Title	Link
OpenCPI Overview	Overview.pdf
Acronyms and Definitions	Acronyms_and_Definitions.pdf
Getting Started	Getting_Started.pdf
Installation Guide	RPM_Installation_Guide.pdf
HDL Development Guide	OpenCPI_HDL_Development.pdf

 Table 1: References

This document explains usage of Xilinx Vivado in the context of OpenCPI. For further information regarding Xilinx Vivado, consult Xilinx's documentation (*e.g.* UG835).

2 Migrating an OpenCPI Platform from ISE to Vivado

Note: This section explains how to migrate an OpenCPI Platform *already created in a previous version of OpenCPI* to use Vivado. Documentation for defining a *new* platform can be found in the *HDL Development Guide*, referenced in Table 1.

- 1. Modify hdl/platforms/<platform>/<platform>.mk to use a target part that maps to Vivado (*e.g.* HdlPart_matchstiq_z1=xc7z020-1-clg484).
- 2. Port the platform's UCF file to an XDC file. Reference Xilinx's document Vivado Migration (UG911) for assistance.
- 3. Port the UT file to an XDC file ending in "*_bit.xdc". Reference Xilinx's document Vivado Migration (UG911) for assistance.
- 4. Modify <platform>/Makefile to export both the XDC files ("*.xdc" and "*_bit.xdc") instead of the UCF and UT files, via ExportFiles=.
- 5. Build for the platform using the platform name (HdlPlatform=<platform>) or the target-part (HdlTarget=zynq).

3 Reverting an OpenCPI Platform from Vivado to ISE

This process is described in assets/hdl/platforms/matchstiq_z1/ise_constraints/README for the "matchstiq_z1" platform. To summarize more generically:

- 1. Modify hdl/platforms/<platform>.mk to use the "ise alias" of the target part (*e.g.* HdlPart_matchstiq_z1=xc7z020_ise_alias-1-clg484).
- 2. Port the platform's XDC file ("*.xdc") to a UCF file. Reference Xilinx's document Vivado Migration (UG911) for assistance.
- 3. Port the platform's XDC configuration file ("*_bit.xdc") to a UT file. Reference Xilinx's document Vivado Migration (UG911) for assistance.
- 4. Modify <platform>/Makefile to export the UCF and UT files instead of the XDC files via ExportFiles=.
- 5. Build for the platform using the platform name (HdlPlatform=<platform>) or the target-part's "ise alias" (HdlTarget=zynq_ise).

4 Vivado Constraints Files

By default, all constraints files in a platform's directory with the extension ".xdc" are applied during the opt_design stage (the first post-synthesis implementation stage), *except* for files ending with "_bit.xdc". Files ending with "_bit.xdc" are applied later during bitstream generation (write_bitstream).

Options such as pin assignments, clock constraints, I/O standards, etc. *can* be placed in an ".xdc" file that does *not* end in "_bit.xdc". For example:

set_property PACKAGE_PIN V9 [get_ports lime_spi_sdo];

Constraints relating to project/chip/board configuration as well as bitstream settings can be placed in the "_bit.xdc" file. This file is the equivalent of Xilinx ISE's ".ut" file. For example: set_property BITSTREAM.CONFIG.TCKPIN PullUp [current_design];

5 Using PreBuilt Cores/Netlists with Vivado and OpenCPI

While Vivado generates netlists in the EDIF or DCP format, it can also read netlists in NGC format. So, NGC cores prebuilt with ISE (*e.g.* the ddc_4243_4ch_v5 primitive or the complex_mixer's debug cores) can be used in the same way a Vivado EDIF is used.

When including a core using Cores= (as seen with the complex_mixer) worker, the core can be either an EDIF, NGC, or DCP file for usage with Vivado. For NGC and EDIF netlists, you will also need to include a VHDL stub file. DCP files, on the other hand, contain both a netlist and a stub, and you therefore do not need to explicitly include a stub file (via SourceFiles=).

To include cores at the worker level, you can set the Cores make variable as follows: Cores="netlist1.ngc mynetlist2.edf mynetlist3.dcp"

As noted above, for NGC and EDIF netlists, you will also need to include a VHDL stub file: SourceFiles="netlist1_stub.vhd mynetlist2_stub.vhd"

Another option for including prebuilt cores with OpenCPI is to create an OpenCPI primitive core. In the primitive core's makefile, you would set "PreBuiltCore=mynetlist2.edf". You would then create a VHDL package file. This would comprise of a VHDL package containing the core's component declaration. You would then be able to include this core for any worker using "Cores=<core-name>".

6 Simulating Vivado IP or PreBuilt Cores with XSIM in OpenCPI

You may be able to build for XSIM with OpenCPI by including the stub VHDL file mentioned in 5, but omitting the netlist. If this does not work, you will have to generate a simulation netlist.

A simulation netlist can be generated by opening up the post-synthesis IP or core and running the following TCL command:

> write_vhdl <ip_name>_sim.vhd

Now include that as a source file in your worker Makefile: SourceFiles=<ip_name>_sim.vhd

The corresponding *synthesis* stub and netlist files (if present) will need to be removed from the SourceFiles and Cores variables in the Makefile before building for XSIM.

7 Using Vivado IP with OpenCPI

To use Vivado's IP within OpenCPI, you can follow these steps:

• Create a new Vivado RTL project with no sources

- Window→IP Catalog
- Choose IP, Customize IP
- Generate IP output products in Global mode (stubs, test bench, xci)
- Run synthesis and Open Synthesized Design
- Generate the necessary netlist/source files:
 - Generate the EDIF netlist and VHDL stub:
 - > write_edif -security_mode all <ip_name>.edf
 - > write_vhdl -mode synth_stub <ip_name>.vhd
 - Or, generate the DCP (checkpoint file):
 - > write_checkpoint <ip_name>.dcp

Note: Reference section 5 for more information on using Vivado pre-built cores with OpenCPI

- Generate a simulation netlist for use with XSIM:
 - > write_vhdl <ip_name>_sim.vhd

Note: For more information on simulating Vivado IP with ANGRYIPER, reference 6.

• Reference the Instantiation Template (*.vho) file when instantiating the module in your design.

8 Makefile options for Vivado/XSIM compilation

8.1 Incremental Compilation - Place/Route

Setting "VivadoIncrementalCompilation=true" (false by default) enables Vivado's incremental compilation for place and route. This applies during Container compilation only. If enabled, Vivado will attempt to reuse the results of previous place/route runs for this Container. This is very useful when making small source changes (or changes to comments) and then rebuilding.

8.2 Synthesis Options : Applies to primitives, workers, platforms, configs, assemblies, containers

To set options for the synthesis stage of compilation:

VivadoExtraOptions_synth="-<myoption1> -<myoption2>"

Only use the quotes at the command line. If setting this variable *inside* a Makefile, omit the quotes. If you are setting this variable in an assembly Makefile and wish to apply it during container synthesis (as opposed to assembly synthesis), you must prepend the command with 'export '. For example: export VivadoExtraOptions_synth=-directive runtimeoptimized

8.3 Enabling Optimization Stages

Setting "VivadoPowerOpt=true" enables Vivado's power_opt_design stage. This is run directly after opt_design during container implementation.

Vivado's optional phys_opt_design stage of implementation can be run after place_design or after route_design. In each case, different optimizations are performed. The following options can be used to enable the phys_opt_ design stage in on or both of the position:

- "VivadoPostPlaceOpt=true" enables Vivado's phys_opt_design stage after place_design
- "VivadoPostRouteOpt=true" enables Vivado's phys_opt_design stage after route_design

8.4 Implementation Options : Applies to Containers

To set options for a specific implementation stage of compilation:

VivadoExtraOptions_<stage>="-<myoption1> -<myoption2>"

Note: If setting these variables *inside* an assembly Makefile, you must **prepend the command with 'export**' and omit the quotes. Here, stage can be: opt, place, post_place_phys_opt, route, post_route_phys_opt, timing, bit.

8.5 XSIM Options : Applies to primitives, workers, platforms, configs, assemblies, containers

To set options for the XSIM xvhdl and xvlog commands: XsimExtraArgs=" -<myoption1> -<myoption2> "

To set options for the XSIM elaboration stage (xelab): XsimXelabExtraArgs=" -<myoption1> -<myoption2> " Note: Only use the quotes at the command line. If setting these variables *inside* a Makefile, omit the quotes.

9 Global Tcl Initialization Scripts

As explained in Xilinx's UG835, you can place a Tcl script at **\$HOME/.Xilinx/Vivado/init.tcl** to be executed every time Vivado is launched. This is *not* recommended since it cannot be easily source-controlled with the rest of your project.

10 Opening up designs in the GUI

Prior to running any vivado/xsim commands, you must source <path-to-vivado>/settings64.sh. Because these settings interfere with OpenCPI's environment, you should *always* do this in a separate terminal.

10.1 EDIF Netlist

To open up an EDIF (or NGC) netlist in Vivado, navigate to the directory containing the netlist and run: vivado Once the GUI opens up, run the following Tcl commands: read_edif <netlist-filename>; link_design; You can then navigate to the "Netlist" tab, right click the file, and choose "Schematic" (Figure 1).



Figure 1: Xilinx Vivado Netlist

Another option for viewing EDIF netlists in Vivado involves creating a Post-Synthesis project and including the netlist as a source file. You can then "Open Synthesized Design" to view the netlist in the GUI.

10.2 Project File

To open up a Vivado project at any level, run: vivado target-<tgt>/<asset-name>.xpr

Because the framework runs compilation in Non-Project Mode, the source files and synthesis results will not be opened side-by-side.

ы	as_16_rv - [/data/Documents/workspace/ocpibase.	.testing/components/bias_16.hdl/target-zynq/bias	s_16_rv.xpr] - Vivado 2017.1 _ 🛛 🗙					
Eile Edit Flow Tools Window Layout View Help Q- Quick Access								
			🔤 Default Layout 🗸 🗸					
Flow Navigator	PROJECT MANAGER - bias_16_rv		? ×					
 PROJECT MANAGER Settings Add Sources Language Templates IP Catalog IP INTEGRATOR Create Block Design Open Block Design 	Sources ? _ □ Ľ × Q ¥ ♦ + ? ● 0 ↓ > Design Sources (90) ↓ > MakeResetA (MakeResetA.v) (1) ↓ > Ø 5yncRegister (SyncRegister.v) (1) ↓ > Ø bias_16(rtl) (bias_16-impl.vhd) (1) ↓ > Ø cwd(rtl) (cwd.vhd) (1) ↓ > Ø flush_writer(rtl) (flush_writer.vhd) (1) ↓	Project Summary Settings Edit Project name: bias_16_rv Project location: /data/Documents/works Product family: Zynq-7000 Project part: xc7z020clg400-3 Top module name: bias_16_rv Target language: Verilog	? 미 대 X pace/ocpibase_testing/components/bias_16.hdl/target zynq					
Generate Block Design	 > @ message_bounds(rth) (message_bound > @ part_slave(rth) (part_slave.vhd) (1) > @ plusarg(rth) (plusarg.vhd) (1) > @ srl_ffo(rth) (srl_ffo.vhd) (1) 	 > @ message_bounds(rtl) (message_bounds) > @ part_slave(rtl) (part_slave.vhd) (1) > @ plusarg(rtl) (plusarg.vhd) (1) > @ staffo(rtl) (staffo, vhd) (1) Svnthesis 						
Run Simulation RTL ANALYSIS Open Elaborated Design Structuresis	> @ trace_buffer_ulong(rtl) (trace_buffer_u @ BRAM1 (BRAM1.v) Hierarchy Libraries Compile Order Properties ? _ □ Ľ ×	Status: Not started Messages: No errors or warnings Part: xc7z020clg400-3 Strategy: Vivado Synthesis Defaults	Status: Not started Messages: No errors or warnings Part: xc7z020clg400-3 Strategy: Vivado Implementation Defaults Incremental compile: None					
 Run Synthesis 	$\leftarrow \Rightarrow \diamondsuit$	Timing						
 Open Synthesized Design IMPLEMENTATION 		Run Implementation to see DRC results	Run Implementation to see timing results					
 Run Implementation Open Implemented Design 	Select an object to see properties	Utilization Run Synthesis to see utilization results	Power Run Implementation to see power results					
 PROGRAM AND DEBUG III Generate Bitstream > Open Hardware Manager 	Tcl ConsoleMessagesLogReports $Q_{\parallel} \stackrel{\star}{\Rightarrow} \diamondsuit \And \And \And \gg \Rightarrow \bigstar \% $	Design Runs X	? _ 🗆 🛛					
	Name Constraints Status WNS ∨ ▷ synth_1 constrs_1 Not started ▷ impl_1 constrs_1 Not started	TNS WHS THS TPWS Total Power Failed	Routes LUT FF BRAMS URAM DSP Start Elapsed					

Figure 2: Xilinx Vivado Project

However, after a project file is opened in the GUI, synthesis can be rerun in Project Mode. The synthesized design can then be opened, and the netlist and source can be viewed together.

Synthesis Completed						
i Synthesis successfully completed. Next						
O <u>R</u> un Implementation						
Open Synthesized Design						
○ View Reports						
Don't show this dialog again						
OK Cancel						

Figure 3: Xilinx Vivado Open Synthesized Design

After this, netlists and sources can be viewed together.

bia	s_16_rv - [/data/Documents/workspace/ocpibase_testing/components/bias_16.hdl/target-zynq/bias_16_rv.xpr] - Vivado 2017.1	_ = ×
<u>F</u> ile <u>E</u> dit F <u>l</u> ow <u>T</u> ools <u>W</u> in	ndow Layout View Help Q- Quick Access	Synthesis Complete 🛛 🗸
	× ∞ ▶ 時 月 ② ② ◇ ∑ ※ ※	Timing Analysis
Flow Navigator	SYNTHESIZED DESIGN - xc7z020clg400-3 (active)	? ×
 PROJECT MANAGER Settings Add Sources Language Templates IP Catalog IP INTEGRATOR Create Block Design Open Block Design Generate Block Design SIMULATION Run Simulation RTL ANALYSIS 	Sources Netlist × Netlist Pro ? C X bias_16_rv >	? 🗆 🗹 8 I/O Ports 406 Nets 🌩
 > Open Elaborated Design > SYNTHESIS > Run Synthesis > Open Synthesized Des Constraints Wizard 		
Edit Timing Constraint		
🐞 Set Up Debug	Tcl Console Messages Log Reports Design Runs Timing x	2 _ 0 6
ö Report Timing Summa	No timing reports are available for display. To report timing alick one of the following links:	L L
Report Clock Network:	Check Timing reports are available for display. To report timing click one of the following links:	
Report Clock Interacti	crieck the design to possible drining problems.	
🖻 Report Methodology	Report liming Summary Generate a timing summary to understand if the design met timing.	
Report DRC	Report Timing Generate a timing report.	
Report Noise	Report CDC Generate a clock domain crossings report.	
Report Utilization	Report Pulse Width Reports the pulse width of the specified clock signals in the clock network and upon reaching the flip-flop.	
📽 Report Power	Report Datasheet Create a datasheet report for the current design, providing the timing characteristics at the I/O pads.	~

Figure 4: Xilinx Vivado Netlist View

At this point, you can right click on elements of the netlist and "Go To Source".

The various stages of implementation also generate project files. These project files can be opened, and implementation can be run in Project-Mode via the Vivado GUI. The other option for viewing implementation results is to open an implementation-stage's checkpoint as described in 10.3.

10.3 Implementation Design Checkpoint

To open up a Vivado Design Checkpoint resulting from any post-synthesis implementation stage, run: vivado <path-to-OCPI-container-dir>/target-<tgt>/<OCPI-container-name>-<stage>.dcp



Figure 5: Xilinx Vivado Post-Route Design Checkpoint

10.4 Interactive Timing Report

To open up an interactive timing report (result of timing stage of implementation):

- 1. Open up the Design Checkpoint for the route stage (shown in 10.3).
- 2. Open the interactive timing report:

 $File \rightarrow Open \ Interactive \ Report \rightarrow \texttt{<container-name>-timing.rpx}$

complex_mixer_O_zed_base-ro	oute.dcp – [/data/Docu	iments/work	space/oo	piassets_test	ing/components/dsp_comps/c	omplex_mixer.test/gen/asseml	blies/complex	_mixer_0/cor	1	• ×
<u>F</u> ile <u>E</u> dit F <u>l</u> ow <u>T</u> ools <u>W</u>	(indow Layout ⊻ie	w <u>H</u> elp	Q ≁ Quid	k Access						
	X 👁 🚽 🔯	0 💠 :	<u>k</u>	×				📰 Timin	ıg Analysis	~
CHECKPOINT DESIGN - xc7z020c	lg484-1									? ×
Netlist × Path Properties	? _ 0 6	Device							? [- C ×
· 문 · 년	•	← →	Ð,	Q 🔀 🕅	🕀 🖽 Ro 💷 🔍					¢
ii PS7_ ii PS_C ii PS_C ii PS_C ii PS_S ii VCC v ii sr (SyncResetA) > Nets (22) v ii Leaf Cells (2 ii reset_ho ii reset_ho	i (PS7) IK_BIBUF (BIBUF) ORB_BIBUF (BIBUF) ORB_BIBUF (BIBUF) RSTB_BIBUF (BIBUF) NCC) id[17]_i1 (LUT1) id[reg[0] (FDCE) id[reg[1] (FDCE) id[reg[2] (FDCE) id[reg[3] (FDCE) id[reg[4] (FDCE) id[reg[5] (FDCE) id[reg[6] (FDCE) id[reg[10] (FDCE) id[reg[11] (FDCE) id[reg[12] (FDCE) id[reg[12] (FDCE) id[reg[12] (FDCE)									
reset_nd	id_reg[13] (FDCE)	/								ĸ
Tcl Console Messages 1	iming ×								? .	_ 0 6
Q ≚ ≑ C Ш	Q, , , , , , , , , , , , , , , , , , ,	– H	▲	Timin	g Checks - Setup					
General Information	Name	Slack 🔨 1	Levels	High Fanout	From	То	Total Delay	Logic Delay	Net Delay	Logic
Settings	l> Path 43	0.712	1	475	ftop/pfconfigold_reg[17]/C	ftop/pfconfigunt_reg[10]/R	8.496	0.580	7.916	-
Setup (40)	ly Path 44	0.712	1	475	ttop/pfconfigold_reg[17]/C	ftop/pfconfigunt_reg[11]/R	8.496	0.580	7.916	
Hold (40)	l Path 45	0.712	1	475	ttop/pfconfigold_reg[17]/C	ttop/pfconfigount_reg[8]/R	8.496	0.580	7.916	
1010 (107	Path 46	0.712	1	475	ttop/ptcontigold_reg[17]/C	πop/ptcontigount_reg[9]/R	8.496	0.580	7.916	2
	Path 4/	0.745	6	80	ftop/prconfigi/SAXIHPTACLK	ftop/zynq_sapeg_reg[2]/CE	8.754	2.409	5.345	2
	48 ×	0.755	1	4/5	itop/prconfigoia_reg[17]/C	ntop/prcontigount_reg[4]/R	8.452	0.580	/.8/2	>
Report Timing - complex_mix	er_0_zed_base-timin	g (saved)								

Figure 6: Xilinx Vivado Interactive Timing Report

10.5 Elaborated XSIM design

```
To open up the results of XSIM's xelab:
xsim <path-to-OCPI-container-dir>/target-<tgt>/<OCPI-container-name>
```

	_ = ×				
File Edit Tools Window Layout					
		10 ns 🗸 🖭	С	😑 Default Layout 💊	
SIMULATION - Simulation - complex mixer 0	frw xsim base			2 3	k
Scope × Sources	_ 🗆 🖸	Objects	? _ 🗆 🖒 X		
Q ≚ ♦	0	Q	0		
Name	Design Unit Block Type	Name	Value Data^		
v 🛾 complex_mixer_0_frw_xsim_base	complex_mix VHDL En	> 👒 ocpi_deb	0 Array		
> 🏮 ftop	complex_mix VHDL En	> 👒 ocpi_endi	0 Array		
🧧 glbl	glbl Verilog	> 👒 sdp_widt	01 Array		
		<			
					i.
ICICONSOLE X Messages				? _ 0 6	
Q, X ≑ II 🗉 🖩 🗖					
INFO: [IP_Flow 19-2313] Loaded Vivad	o IP repository '/opt/Xilinx/V	/ivado/2017.1/data/ip	5. Contraction (1997)	^	1
Vivado Simulator 2017.1	er -autotoadworg				L
— Time resolution is 1 ps					L
Type a Tel command hare				>	
Type a recommand here					
				Sim Time: 0 fs	

Figure 7: Xilinx Vivado Simulator Elaborated Design

10.6 Open XSIM Wave Database

As with any other simulator, you can run:

ocpiview <simulations-directory>

For example, for the complex_mixer.test Unit Test, after running case00.00 with ocpidev's --keep-simulations option (or Make's KeepSimulations=1) set, you can run:

ocpiview run/xsim/case00.00.complex_mixer.hdl.simulation



Figure 8: Xilinx Vivado Simulator Wave Database

11 OpenCPI Output Files for Vivado

- .jou: Vivado journal file
- . jou.bak: Backup of the previously generated Vivado journal file
- <asset-name>-vivado.out: OpenCPI and Vivado output for synthesis of an asset
- <impl-stage>.out: OpenCPI and Vivado output for a stage of implementation
- .xpr: Every stage of compilation for every OpenCPI asset results in a Vivado project file. This project file can be opened in Vivado to observe the source files associated with that stage.
- .edf: Vivado's netlist format. This is the artifact of building any OpenCPI asset except primitive libraries.
- .dcp: After the container is synthesized, implementation begins. From then on, DCP (Vivado's Design Checkpoint) files are used as the result of each implementation stage.
- .rpx: Vivado's Interactive Timing Report. This file is generated as a result of the timing stage which is run after route.
- .libs, .sources, .cores: The OpenCPI files used to store information regarding the libraries, sources, and cores that an asset depends on
- *.hw, *.ip_user_files, *.cache directories: Various directories generated by Vivado when creating a project or running synthesis/implementation stages

12 OpenCPI Output Files for XSIM

- .log: XSIM log file for xelab, xvhdl, and xvlog commands
- .vdb: Output of XSIM's source parser
- xsim.dir: Files generated by XSIM during setup and elaboration
- .pb: Message information for Vivado's "Messages" window