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

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>/<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

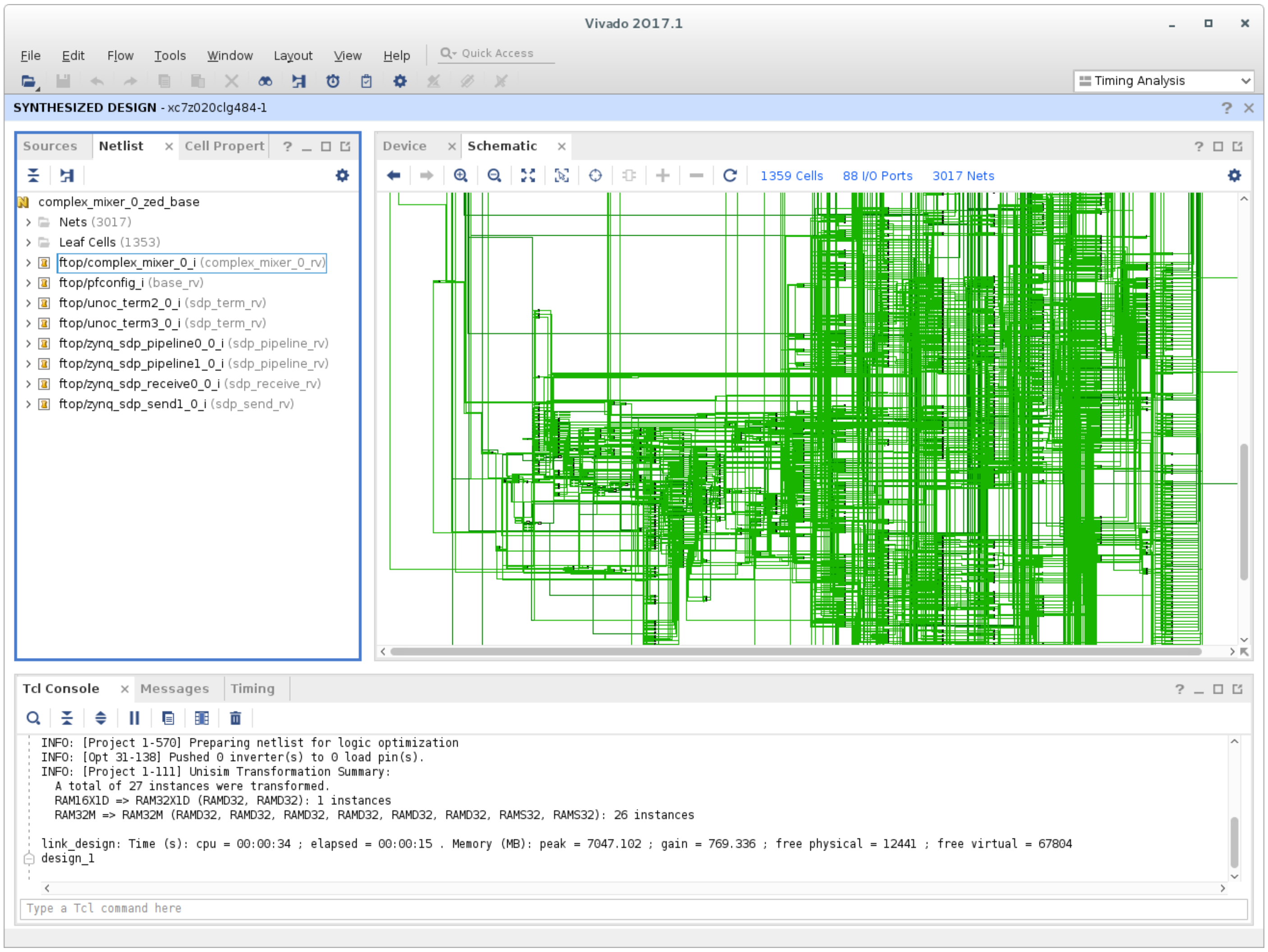
As explained in Xilinx's UG835, you can place a Tcl script at `$HOME/.Xilinx/Vivado/init.tcl` to be executed every time Vivado is launched. This is *not* recommended since it cannot be easily source-controlled with the rest of your project.

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).


Vivado 2017.1
File Edit Flow Tools Window Layout View Help
Quick Access
Timing Analysis
SYNTHESIZED DESIGN - xc7z020clg484-1
Sources Netlist Cell Propert
Device Schematic
1359 Cells 88 I/O Ports 3017 Nets
Tcl Console Messages Timing

```
complex_mixer_0_zed_base
  Nets (3017)
  Leaf Cells (1353)
  ftop/complex_mixer_0_i (complex_mixer_0_rv)
  ftop/pfconfig_i (base_rv)
  ftop/unoc_term2_0_i (sdp_term_rv)
  ftop/unoc_term3_0_i (sdp_term_rv)
  ftop/zynq_sdp_pipeline0_0_i (sdp_pipeline_rv)
  ftop/zynq_sdp_pipeline1_0_i (sdp_pipeline_rv)
  ftop/zynq_sdp_receive0_0_i (sdp_receive_rv)
  ftop/zynq_sdp_send1_0_i (sdp_send_rv)
```

```
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
  A total of 27 instances were transformed.
  RAM16X1D => RAM32X1D (RAMD32, RAMD32): 1 instances
  RAM32M => RAM32M (RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMD32, RAMS32, RAMS32): 26 instances

link_design: Time (s): cpu = 00:00:34 ; elapsed = 00:00:15 . Memory (MB): peak = 7047.102 ; gain = 769.336 ; free physical = 12441 ; free virtual = 67804
design_1
```

```
Type a Tcl command here
```

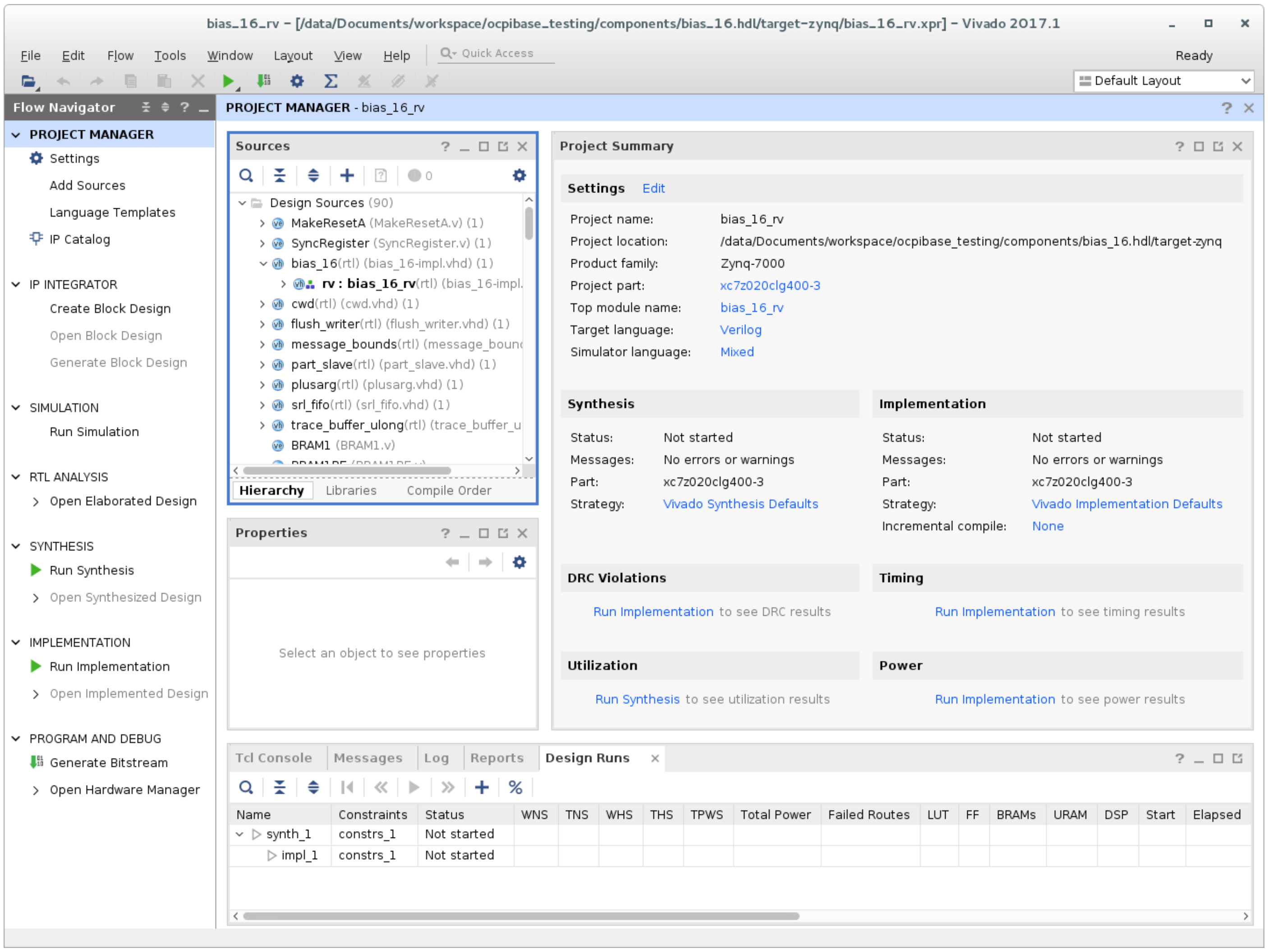
Figure 1: Xilinx Vivado Netlist

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.



Design Sources (90)
- MakeResetA (MakeResetA.v) (1)
- SyncRegister (SyncRegister.v) (1)
- bias_16(rtl) (bias_16-impl.vhd) (1)
  - rv : bias_16_rv(rtl) (bias_16-impl.
- cwd(rtl) (cwd.vhd) (1)
- flush_writer(rtl) (flush_writer.vhd) (1)
- message_bounds(rtl) (message_bound
- part_slave(rtl) (part_slave.vhd) (1)
- plusarg(rtl) (plusarg.vhd) (1)
- srl_fifo(rtl) (srl_fifo.vhd) (1)
- trace_buffer_ulong(rtl) (trace_buffer_u
- BRAM1 (BRAM1.v)

Settings Edit

Project name:	bias_16_rv
Project location:	/data/Documents/workspace/ocpibase_testing/components/bias_16.hdl/target-zynq
Product family:	Zynq-7000
Project part:	xc7z020clg400-3
Top module name:	bias_16_rv
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	Not started
Messages:	No errors or warnings
Part:	xc7z020clg400-3
Strategy:	Vivado Synthesis Defaults

Implementation

Status:	Not started
Messages:	No errors or warnings
Part:	xc7z020clg400-3
Strategy:	Vivado Implementation Defaults
Incremental compile:	None

DRC Violations

Run Implementation to see DRC results

Timing

Run Implementation to see timing results

Utilization

Run Synthesis to see utilization results

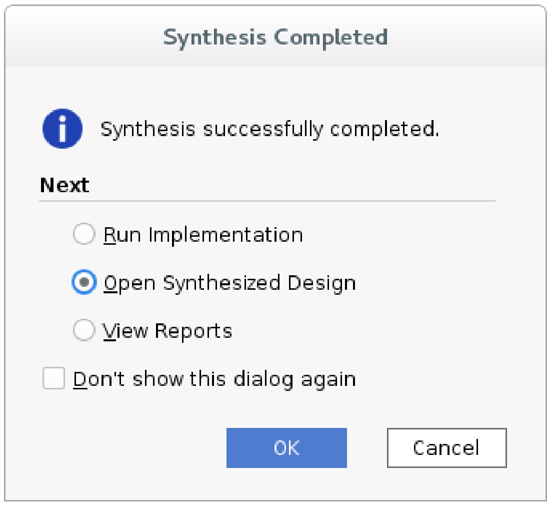
Power

Run Implementation to see power results

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed
synth_1	constrs_1	Not started														
impl_1	constrs_1	Not started														

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 successfully completed.

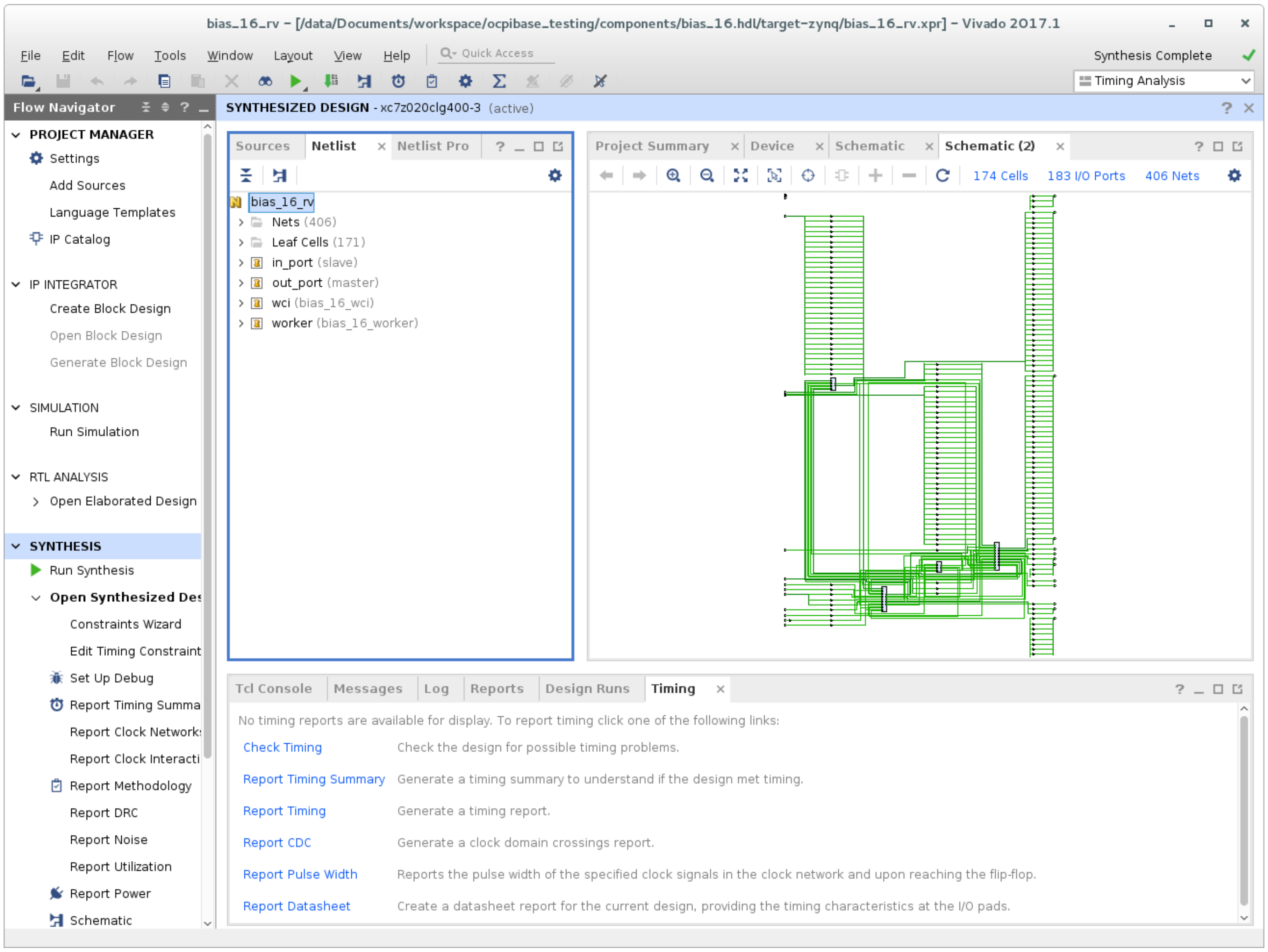
Next

- Run Implementation
- [✓ Open Synthesized Design]
- View Reports

Don't show this dialog again

Figure 3: Xilinx Vivado Open Synthesized Design

After this, netlists and sources can be viewed together.



No timing reports are available for display. To report timing click one of the following links:

Check Timing	Check the design for possible timing problems.
Report Timing Summary	Generate a timing summary to understand if the design met timing.
Report Timing	Generate a timing report.
Report CDC	Generate a clock domain crossings report.
Report Pulse Width	Reports the pulse width of the specified clock signals in the clock network and upon reaching the flip-flop.
Report Datasheet	Create a datasheet report for the current design, providing the timing characteristics at the I/O pads.

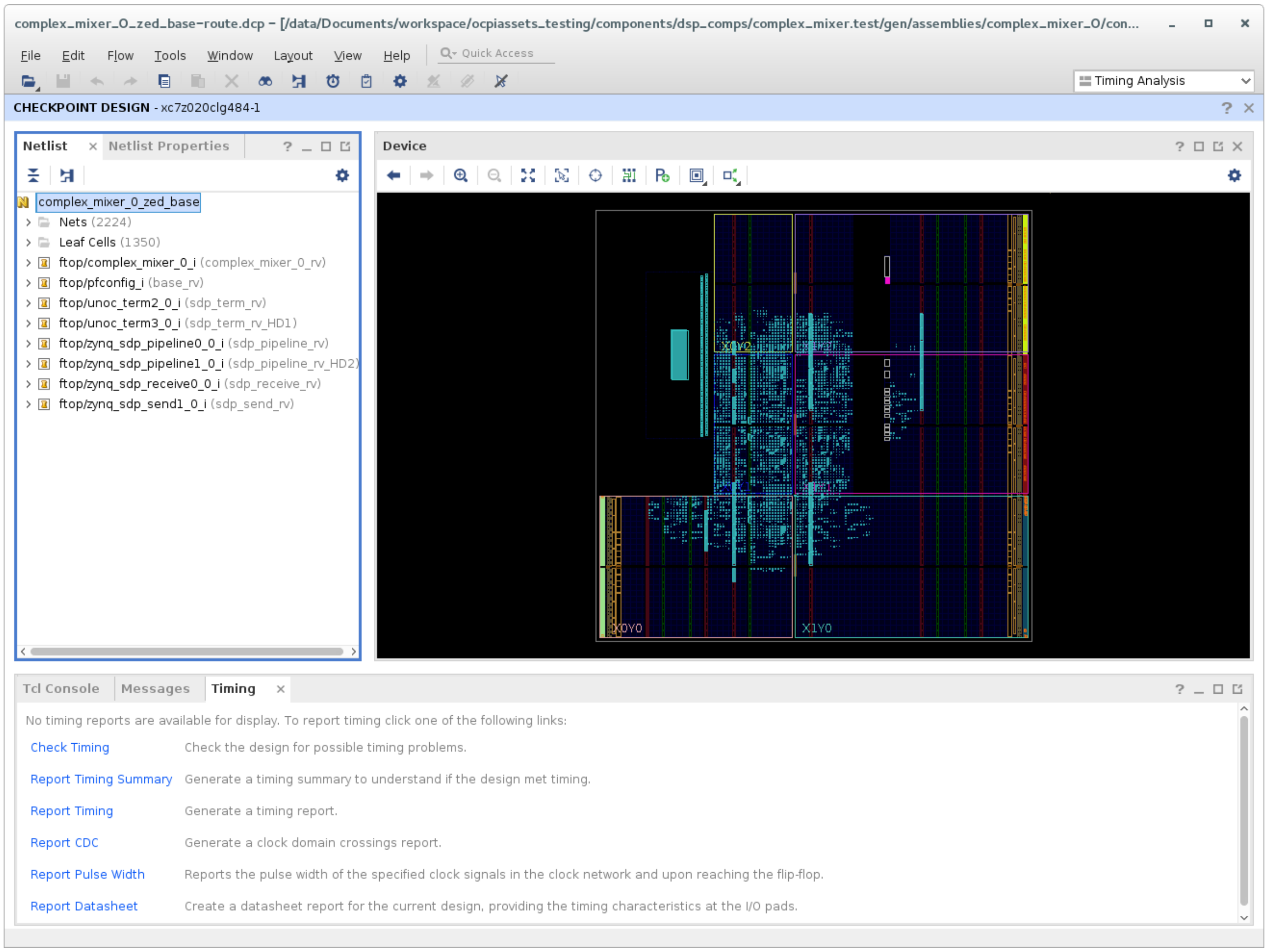
Figure 4: Xilinx Vivado Netlist View

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`



complex_mixer_0_zed_base-route.dcp - [/data/Documents/workspace/ocpiassets_testing/components/dsp_comps/complex_mixer.test/gen/assemblies/complex_mixer_0/con...

CHECKPOINT DESIGN - xc7z020clg484-1

```
complex_mixer_0_zed_base
  Nets (2224)
  Leaf Cells (1350)
  ftop/complex_mixer_0_i (complex_mixer_0_rv)
  ftop/pfconfig_i (base_rv)
  ftop/unoc_term2_0_i (sdp_term_rv)
  ftop/unoc_term3_0_i (sdp_term_rv_HD1)
  ftop/zynq_sdp_pipeline0_0_i (sdp_pipeline_rv)
  ftop/zynq_sdp_pipeline1_0_i (sdp_pipeline_rv_HD2)
  ftop/zynq_sdp_receive0_0_i (sdp_receive_rv)
  ftop/zynq_sdp_send1_0_i (sdp_send_rv)
```

No timing reports are available for display. To report timing click one of the following links:

Check Timing	Check the design for possible timing problems.
Report Timing Summary	Generate a timing summary to understand if the design met timing.
Report Timing	Generate a timing report.
Report CDC	Generate a clock domain crossings report.
Report Pulse Width	Reports the pulse width of the specified clock signals in the clock network and upon reaching the flip-flop.
Report Datasheet	Create a datasheet report for the current design, providing the timing characteristics at the I/O pads.

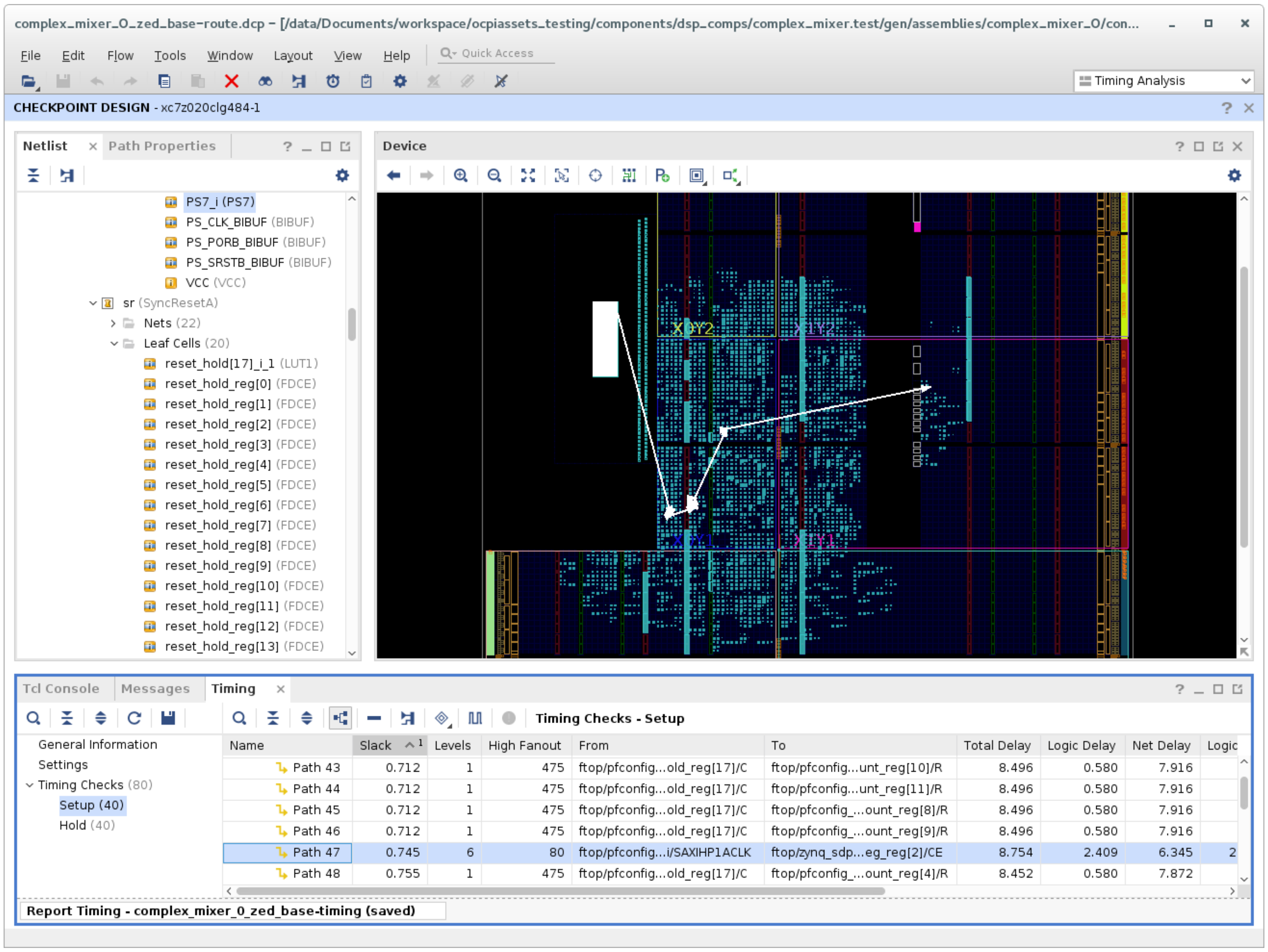
Figure 5: Xilinx Vivado Post-Route Design Checkpoint

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 → Open Interactive Report → `<container-name>-timing.rpx`


complex_mixer_0_zed_base-route.dcp - [/data/Documents/workspace/ocpiassets_testing/components/dsp_comps/complex_mixer.test/gen/assemblies/complex_mixer_0/con...
File Edit Flow Tools Window Layout View Help
Quick Access
Timing Analysis
Netlist
Path Properties
Device
X0Y2
X1Y2
X0Y1
X1Y1
Tcl Console
Messages
Timing

CHECKPOINT DESIGN - xc7z020clg484-1

- PS7_i (PS7)
- PS_CLK_BIBUF (BIBUF)
- PS_PORB_BIBUF (BIBUF)
- PS_SRSTB_BIBUF (BIBUF)
- VCC (VCC)
- sr (SyncResetA)
  - Nets (22)
  - Leaf Cells (20)
    - reset_hold[17]_i_1 (LUT1)
    - reset_hold_reg[0] (FDCE)
    - reset_hold_reg[1] (FDCE)
    - reset_hold_reg[2] (FDCE)
    - reset_hold_reg[3] (FDCE)
    - reset_hold_reg[4] (FDCE)
    - reset_hold_reg[5] (FDCE)
    - reset_hold_reg[6] (FDCE)
    - reset_hold_reg[7] (FDCE)
    - reset_hold_reg[8] (FDCE)
    - reset_hold_reg[9] (FDCE)
    - reset_hold_reg[10] (FDCE)
    - reset_hold_reg[11] (FDCE)
    - reset_hold_reg[12] (FDCE)
    - reset_hold_reg[13] (FDCE)

- General Information
- Settings
- Timing Checks (80)
  - Setup (40)
  - Hold (40)

Timing Checks - Setup

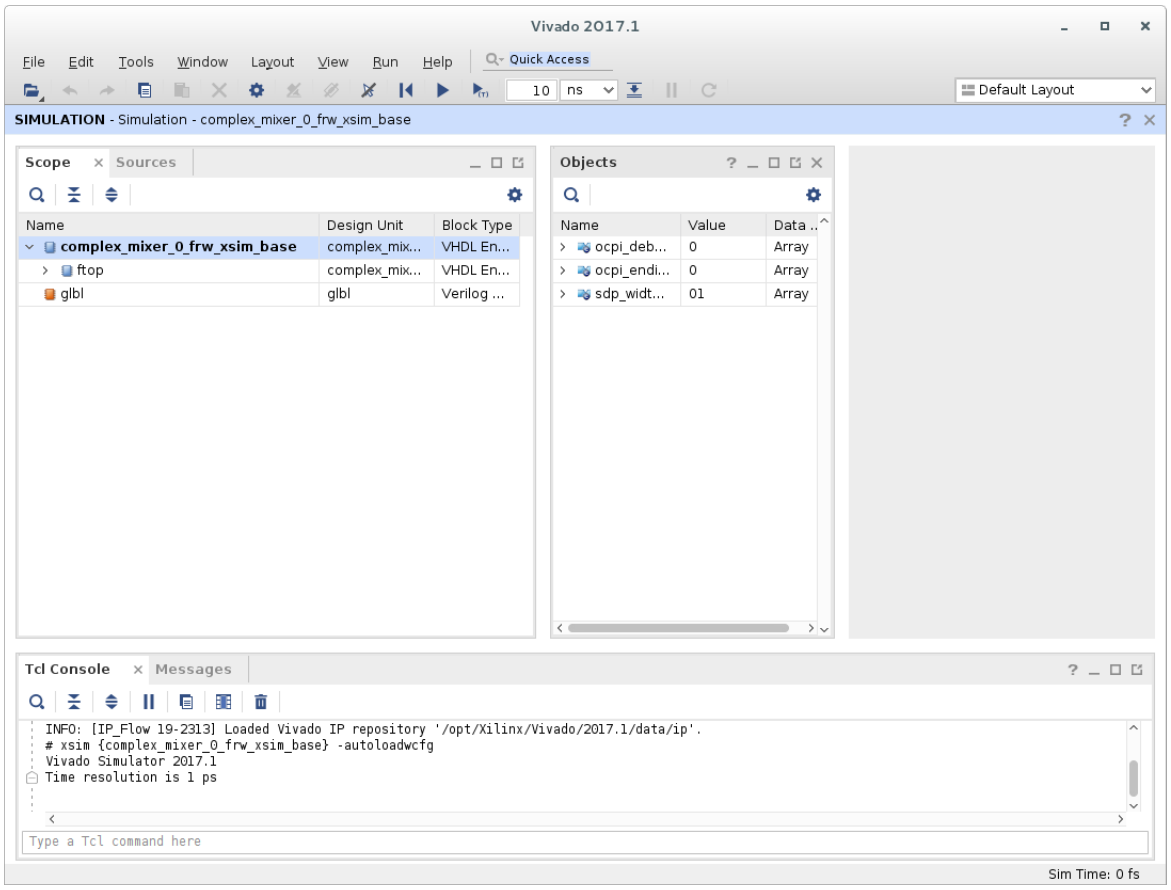
Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic
Path 43	0.712	1	475	ftop/pfconfig...old_reg[17]/C	ftop/pfconfig...unt_reg[10]/R	8.496	0.580	7.916	
Path 44	0.712	1	475	ftop/pfconfig...old_reg[17]/C	ftop/pfconfig...unt_reg[11]/R	8.496	0.580	7.916	
Path 45	0.712	1	475	ftop/pfconfig...old_reg[17]/C	ftop/pfconfig_...ount_reg[8]/R	8.496	0.580	7.916	
Path 46	0.712	1	475	ftop/pfconfig...old_reg[17]/C	ftop/pfconfig_...ount_reg[9]/R	8.496	0.580	7.916	
Path 47	0.745	6	80	ftop/pfconfig...i/SAXIHP1ACLK	ftop/zynq_sdp...eg_reg[2]/CE	8.754	2.409	6.345	2
Path 48	0.755	1	475	ftop/pfconfig...old_reg[17]/C	ftop/pfconfig_...ount_reg[4]/R	8.452	0.580	7.872	

Report Timing - complex_mixer_0_zed_base-timing (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>`



Name	Design Unit	Block Type
complex_mixer_0_frw_xsim_base	complex_mix...	VHDL En...
ftop	complex_mix...	VHDL En...
glbl	glbl	Verilog ...

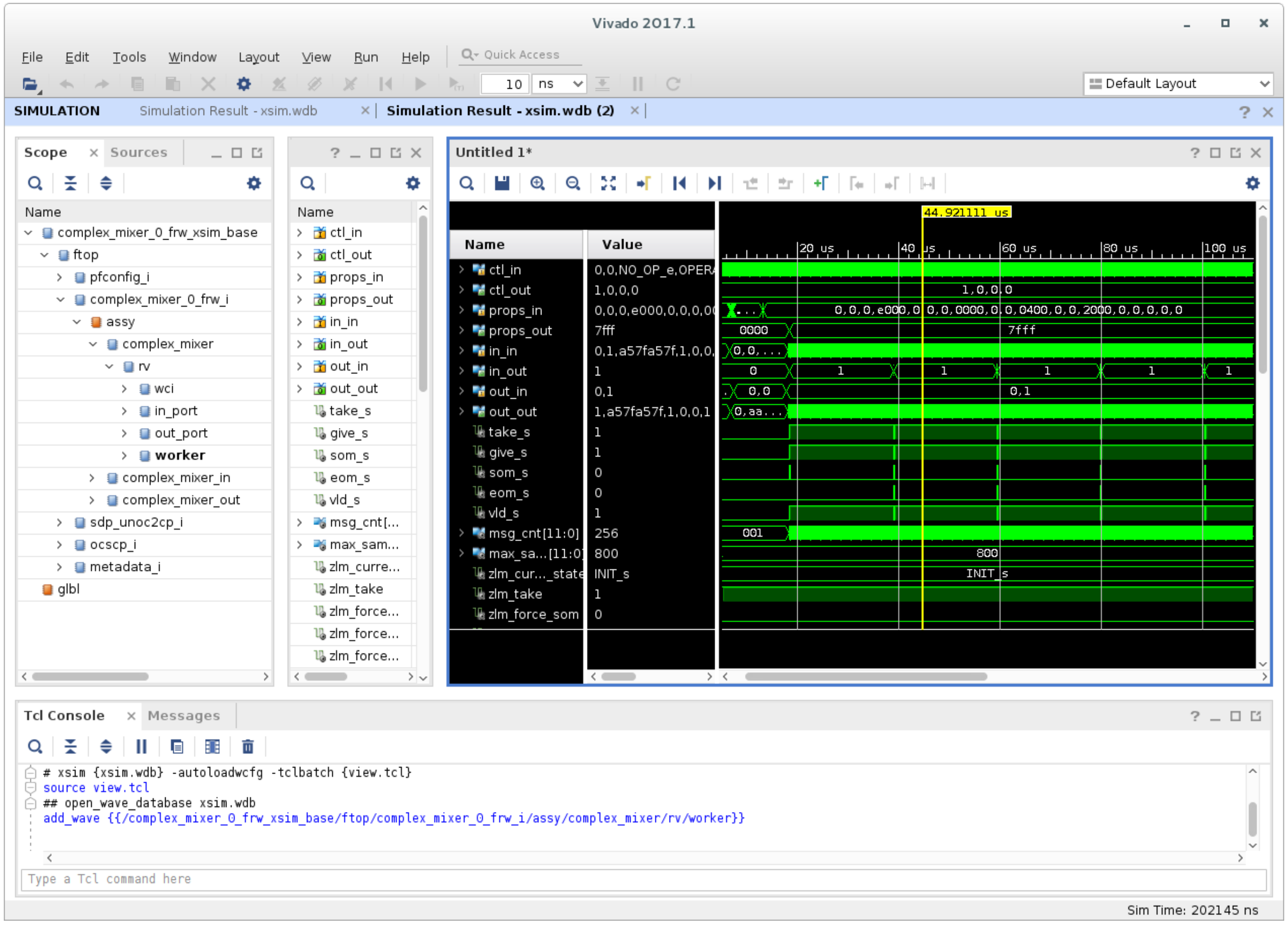
Name	Value	Data ...
ocpi_deb...	0	Array
ocpi_endi...	0	Array
sdp_widt...	01	Array

```
INFO: [IP_Flow 19-2313] Loaded Vivado IP repository '/opt/Xilinx/Vivado/2017.1/data/ip'.
# xsim {complex_mixer_0_frw_xsim_base} -autoloadwcfg
Vivado Simulator 2017.1
Time resolution is 1 ps
```

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`


```
Name                Value
ctl_in              0,0,NO_OP_e,OPER
ctl_out             1,0,0,0
props_in            0,0,0,e000,0,0,0,0
props_out           7fff
in_in               0,1,a57fa57f,1,0,0,
in_out              1
out_in              0,1
out_out             1,a57fa57f,1,0,0,1
take_s              1
give_s              1
som_s               0
eom_s               0
vld_s               1
msg_cnt[11:0]       256
max_sa...[11:0]     800
zlm_cur..._state    INIT_s
zlm_take            1
zlm_force_som       0
```

```
# xsim {xsim.wdb} -autoloadwcfg -tclbatch {view.tcl}
source view.tcl
## open_wave_database xsim.wdb
add_wave {{/complex_mixer_0_frw_xsim_base/ftop/complex_mixer_0_frw_i/assy/complex_mixer/rv/worker}}
```

Sim Time: 202145 ns

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