Speedster7t DDR4 Reference Design Guide (RD018)

Achronix
Data Acceleration

November 03, 2020 **Reference Design**

Introduction

The Speedster®7t DDR4 reference design illustrates how a user can perform data transactions between the FPGA core and the DDR4 subsystem. This design showcases both the network-on-chip (NoC) connection and direct-to-fabric connect (DC) interface for interaction between a design residing in the 7t1500 FPGA fabric and the DDR4 subsystem. The reference design includes an example testbench for simulation and scripts to implement the design on the FPGA. The user is able to simulate the design, run synthesis and execute the ACE flow using a batch build flow at a target frequency of 500 MHz for the FPGA core logic for the DDR4 interface.

The first release included bus functional models (BFMs) for the DDR4 subsystem and attached memory. This second release includes the RTL for both the DDR4 subsystem and interfaces to commonly available DDR4 RTL simulation models, providing a full, cycle-accurate simulation that includes functions such as memory initialization and training.

Description

The latest DDR4 design enables the user to simulate their design in two modes: bus-functional models (BFM) mode or RTL mode

- In BFM mode, the testbench incorporates an integrated BFM for the DDR4 subsystem and memories.
- In RTL mode, the DDR4 subsystem is represented by encrypted RTL and DDR4 cycle-accurate memory models. The reference design enables a user to switch between Micron or SKHynix memory models.

Note

This design supports only single-rank SoDIMM for ×8 data width in non-ECC mode at 3200 Mbps. Future releases will support more configurations, such as UDIMM, RDIMM or LRDIMM, different ranks, data widths, data rates and ECC.

The DDR4 reference design comprises the following design modules:

AXI Data Generator and Checker

The AXI data generator module generates transaction data of varying burst lengths and memory addresses which are sent to the DDR4 memory subsystem. This module enables the generation of sequential or random data and addresses which are then transmitted via the AXI bus. There are two generators in this reference design:

- The first generator sends its data and addresses to a designated AXI NoC access point (NAP). This AXI NAP pushes the data and addresses on to the NoC, where it is transmitted to the DDR4 subsystem.
- The second generator is connected to the DDR DC interface port, and drives the data and address directly to the DDR subsystem.

The two AXI checker modules fetch the NAP and DC interface memory address respectively, read the data and compare it against the expected data. If the read data matches the expected data, the test indicates a pass.

NAP Slave Module

The design instantiates an `ACX_NAP_AXI_SLAVE` macro to communicate with the NAP for read and write transactions. This macro connects the user logic in the FPGA through the NoC to the DDR4 subsystem. The NAP macro transmits the input data through the NoC, and hence, to the DDR4 subsystem. The NAPs can be placed and bound to any arbitrary location on the FPGA, allowing the user to test scenarios where the DDR4 subsystems can handle transactions from any NAP located in the fabric and meet the high data transfer speeds possible with this memory interface.

NoC Locations

For this reference design, an arbitrary location of column 3 and row 2 has been chosen for the slave NAP. This location is specified in both `/src/tb/tb_ddr4_ref_design.sv` and also in `/src/constraints/ace_placements.pdc`. By aligning the placement between simulation and the implemented build, the user ensures the most accurate correlation between the two flows.

DC Interface Logic Locations

The DC interface logic can be placed at any location with FPGA fabric and is not constrained within the reference design. However, in order to close timing between the DC interfaces and the user design, the logic will naturally be placed near to the DC interface ports on the south side of the FPGA fabric.

Data Flow Diagrams

The following figures illustrates the data flow from the user logic in the FPGA interacting with the DDR4 subsystem, via the NoC interface.

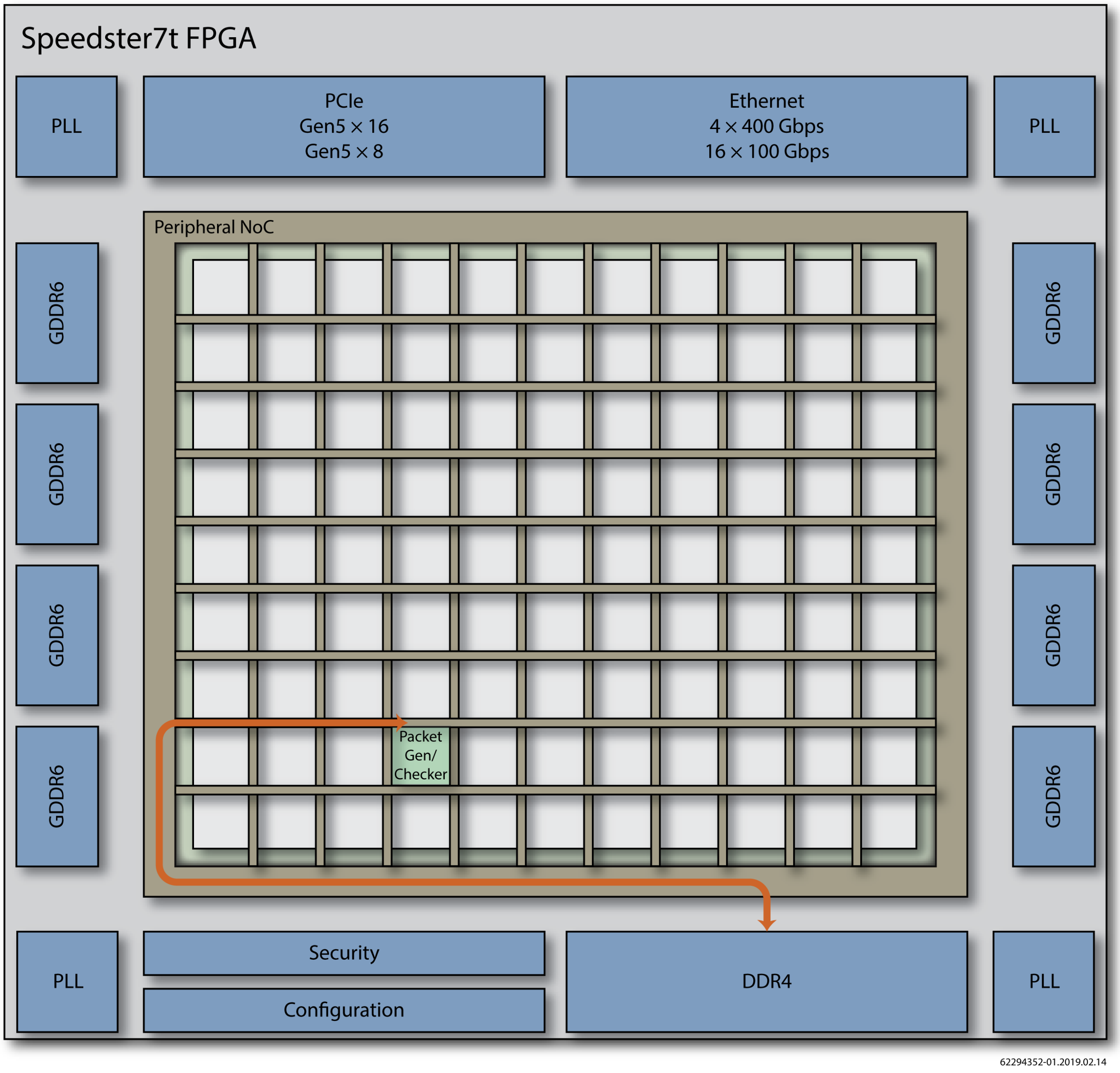

Figure 1: ***Data Generator-Checker Logic Communication Through the NoC***

The following figure illustrates the user logic in the FPGA interacting with the DDR4 subsystem via the DC interface port.

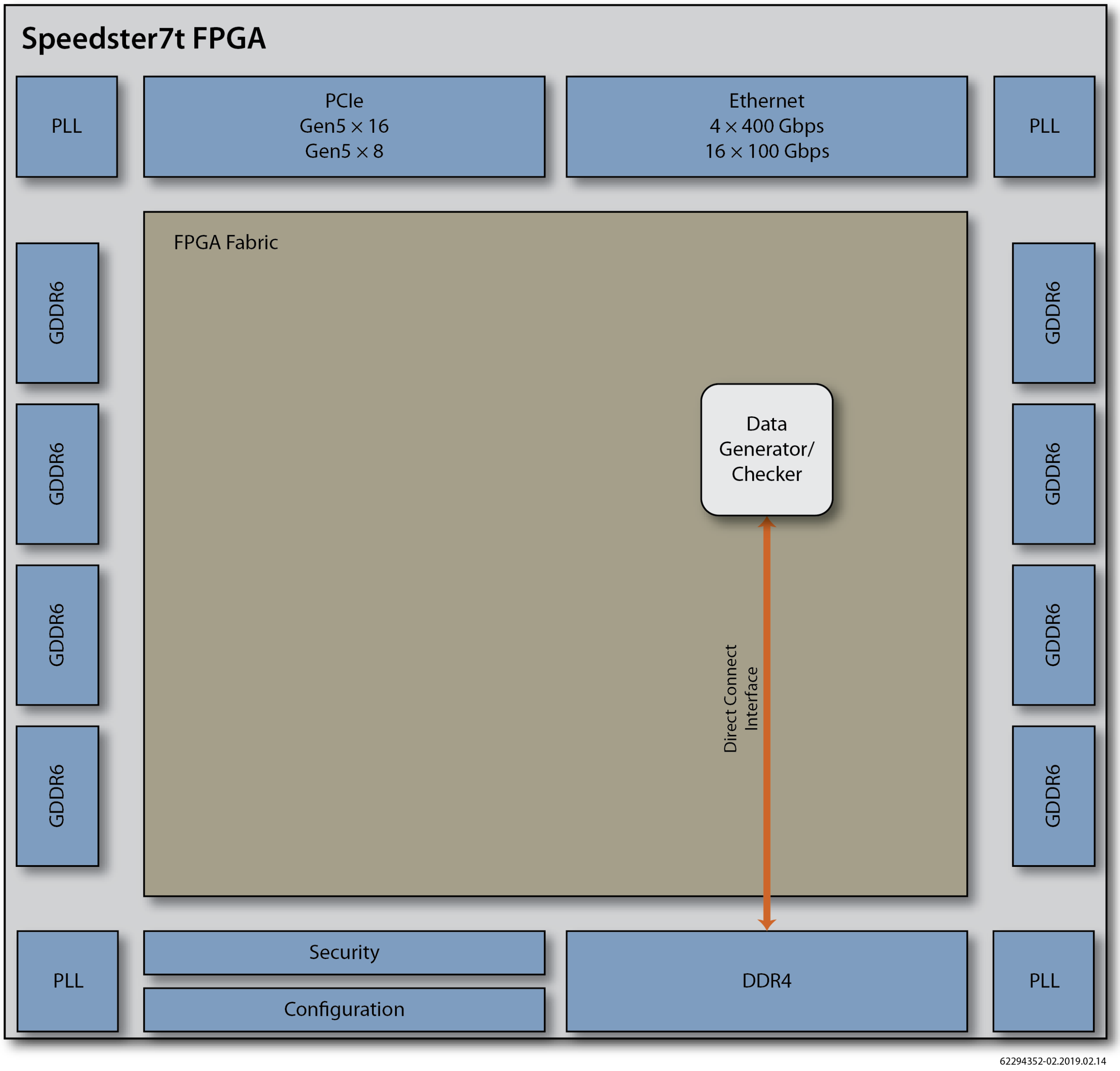

Figure 2: ***Data Generator-Checker Logic Communication Through DC Interface***

DDR4 Calibration and Training Module

The DDR4 controller subsystem requires a training and calibration phase on power-up. The reference design provides a suitable module, `/src/rtl/ddr4_training_polling_block.sv` to execute this training phase. The start-up sequence is described below.

During bitstream load, the FPGA configuration unit will preload many of the DDR4 internal registers. The bitstream file will contain the configuration selected by the user in ACE when initially defining the DDR4 subsystem.

Note

During simulation only, this function is performed by the `fcu_configure()` task, using the supplied `ddr4_memc_config.txt` file.

After initial bitstream load, the FPGA enters user mode. At this stage, the user design must complete the initialization process by monitoring and controlling the read and write leveling performed by the controller. The training block is started by de-asserting the `i_resetn` input to the training block. This process must start after the FPGA enters user mode, and no other transactions must be sent to the DDR4 subsystems until the training block has completed. During training the FSM in the training block writes and reads to addresses in the DRAM in addition to controlling the initialization and training of the DDR4 PHY and controller. Once the memory training is completed, the DDR training block `training_done` output is asserted. When this signal is asserted, the user design can start data transactions to the external DRAM

Notes

The training block in this reference design is connected to a dedicated reset input to allow the flexibility for the user design to be reset independently of the training block. Otherwise, upon each reset, the start-up sequence would be repeated. If the user design will only ever be reset at the start of operation, then it is possible for the training block to share the same global reset as the rest of the design. The only disadvantage to running the start-up sequence multiple times is that, particularly in simulation, it is a time consuming process which the user may wish to avoid.

Equally the training block is presented with a dedicated clock. The user design can modify the training block to use other clocks within the design if required.

Although the training block is only required by the RTL simulation, the user should retain the block in the design even when running BFM simulations, so that when the design is implemented in silicon, the training block is included. In the supplied testbench, the training block is held in reset so that it does not access the DRAM memory, and the output signal is presumed to be asserted when using BFM mode simulations.

Simulation Components

In addition to the components within the user portion of the reference design, there are some key components within the simulation environment that affect the performance of the design.

DDR4 BFM Module (for BFM mode)

This BFM for the DDR4 subsystem models the memory PHY, controller and attached memory device. The DDR4 BFM interfaces with the cycle-accurate NoC model. The DDR4 BFM incorporates delay models which have been derived from the delays measured from the cycle-accurate DDR4 RTL simulation.

Modelling Delays and Latencies

The advantage of using a BFM for simulation is the dramatic speed-up in simulation time, sometimes up to 10×. This speed-up is very apparent when simulating memories or SerDes interface with their long initialization sequences. However, the disadvantage is that a BFM is rarely cycle-accurate, instead it usually represents delays based on averages measured from a cycle-accurate simulation. For the majority of simulations, these average delays are acceptable, and particularly in the case of Speedster7t FPGAs where the transaction has to traverse the cycle-accurate NoC which adds additional delays and latencies, the differences in total system delays between a BFM and an RTL-based simulation is greatly reduced.

It is anticipated that the DDR4 BFM delays, coupled with the cycle-accurate NoC should give a good approximation to the true system delays while enabling users to achieve fast simulation and development turnaround times. The user is then able to use the RTL-based simulation flow later in their development to validate any systems requirements against a full, cycle-accurate simulation of NoC and DDR4.

Ports and Parameters

The DDR4 reference design top-level RTL module sets the following parameters:

Table 1: ***DDR4 Reference Design Top-Level RTL Parameters***

Parameter Name	Value	Description
DDR4_ADDR_ID	2'b01	Fixed value indicating the NoC DDR4 target ID.
NUM_TRANSACTIONS	256	Indicates the total number of AXI transactions to be completed. It can be set up to 8000. If set to 0, the transactions keep running.

The DDR4 reference design top-level RTL module has the following input output ports:

Table 2: ***DDR4 Reference Design Top-Level RTL Ports***

Signal Name	Width	Description
i_clk	1	Global clock that drives user logic interfacing with NoC interface.
i_reset_n	1	External active-low reset input.

Signal Name	Width	Description
i_training_clk	1	DDR4 PHY and controller training clock.
i_training_rstn	1	Active-low reset input for training block.
pll_*_lock	1	Active-high PLL lock signals (two instances).
i_start	1	Assert to start sending transactions on AXI slave NAP and DC interface signals.
o_fail	1	Asserted when data received does not match data expected.
o_fail_oe	1	Active-high output enable for o_fail pin; tied to 1'b1.
o_xact_done	1	Asserted when all transactions are completed.
o_xact_done_oe	1	Active-high output enable for o_xact_done pin; tied to 1'b1.
o_training_done	1	When asserted indicates memory training is complete.
o_training_done_oe	1	Active-high output enable for training done.
ddr4_1_clk	1	Clock that drives user logic interfacing with DC interface.
ddr4_1_clk_alt	2	Alternate clock signals for the DDR4 DC interface clock. The user just needs to connect this signal to the clock in the user logic and ACE will route the clocks automatically.
ddr4_1_dc*	-	Direct connection AXI interface signals. Reflects ACE generated <prefix>_* AXI signal names.

For simulation, the NAP locations are assigned using `ACX_BIND_NAP_AXI_SLAVE` macros in the testbench file. These macros specify the NAP location within the NoC for simulation purposes. For ACE, the locations of these NAPs are set in the `ace_placement.pdc` file. For correlation between simulation and implementation, the design should ensure alignment between these location assignments. In this design the NAP has been set to the slave NAP on column 3 and row 2.

The location of these NAPs can be assigned to any arbitrary location. However, NAPs on the same row share the same overall NoC data bus, so if high utilization is expected, then it may be beneficial to place NAPs with high traffic requirements onto separate rows. Refer to the *Speedster7t Network on Chip User Guide* (UG08) for more details.

Design Considerations

Within the design there are a number of considerations that the user must take into account.

Clocking

Using the supplied ACE GUI project, it can be seen that there are two PLLs, named `pll_1` and `pll_2`, created as internal IP for the design. The source acxip files for these are located in the `/src/acxip` directory. The clocks for these PLLs are detailed in the tables below.

Table 3: ***Reference Design Clocks***

Clock	Direction	Frequency (MHz)	Description
pll_1 clocks			
ref_clk	Input	200	External reference input clock for pll_1.
ddr_clk	Output	800	Reference clock to DDR controller subsystem. (†)
ddr_dci_clk	Output	400	Reference clock to DDR DC interface.(†) This clock is the source of the DDR DC interface `ddr4_1_clk` input to the design that drives the user logic for DC interface.
noc_clk	Output	200	Reference clock to NoC. (†)
pll_2 clocks			
sys_clk	Input	100	External system input clock for pll_2
i_clk	Output	500	Global clock driving user logic for NoC interface
i_training_clk	Output	250	Training clock driving APB interface

Table Note

† These clock outputs connect directly from the reference PLL to their respective destinations. The clocks are not routed via the FPGA fabric, and therefore, are not available to be used by the user logic.

Clock Domains

The design consists of two parallel traffic generator and checker blocks, one for the NAP interface and one for the DC interface. The NAP interface is clocked by the `i_clk` clock domain, and the DC interface is clocked by the `ddr4_1_clk` clock domain. These two clocks are asynchronous to one another, and so clock domain crossing circuits are employed for signals interfacing between these two domains. In addition they are defined as separate clock groups in `/src/constraints/ace_constraints.sdc` and `/src/constraints/synplify_constraints.sdc`.

The frequencies of these two clocks can be defined by these parameters in the top level test bench file `/src/tb/tb_ddr4_ref_design.sv`.

Table 4: ***Testbench Parameters***

Parameter Name	Default Value	Description
DESIGN_CLOCK_PERIOD	2000	Reference design general clock period (in ps). Input to the design as `i_clk`. Provided by `pll_2` with a frequency of 500 MHz.

Parameter Name	Default Value	Description
TRAINING_CLOCK	4000	Memory training clock period. Input to the design as `i_training_clk`. Provided by `pll_2`, with a frequency of 250 MHz.
DDR4_INTERFACE_CLOCK_PERIOD	1250	Clock period (in ps) of the internal DDR4 interface clock rates. These are the clocks that are applied directly from the PLL to the DDR4 controller interfaces, both the NAP and the DC interface. The DC interface operates at half this frequency. The clock speed needs to be sufficient to support the full bandwidth at the internal DDR controller interface. Within the user design, for the DC interface, this clock, at half speed, is the `ddr_dci_clk` provided by `pll_1`. DDR data rate can be 3200, 2666, 2400 Mbps: • For the data rate of 3200 Mbps, set the period to 1250 (800 MHz). • For the data rates of 2666 Mbps and 2400 Mbps, the interface clock period needs be fast enough to support the data rate chosen. Setting it to 800 MHz will guarantee that. • Currently, data rates below 3200 Mbps are not supported
DDR4_CONTROLLER_CLOCK_PERIOD	1250	Clock period (in ps) of the internal clock provided to the DDR4 controller core. This represents `ddr_clk` provided by `pll_1`

Note

For consistency, maintain the same frequency/period values in both the testbench and synthesis constraint files.

ACE I/O Designer

Warning!

The DDR4 IP in the ACE I/O Designer currently forces the user to provide input clocks for both the NoC and DC interface, even if the user chooses to utilize just one of the interface connections.

The above mentioned issues will be addressed in future ACE releases.

Resets

There are a number of reset sources into the design as detailed in the table below.

Table 5: *Reset Sources*

Source	Description
i_reset_n	External asynchronous input.

Source	Description
ddr4_1_rstn	Output from DDR DC interface. Synchronous to `ddr4_1_clk`
pll_1_lock	PLL_1 locked.
pll_2_lock	PLL_2 locked.
i_training_rstn	External asynchronous reset for DDR training block.

Good design practice requires that resets are carefully synchronized and processed. The reference design uses the `reset_processor` module which combines all of the above sources, correctly synchronizes them to each of the two clock domains and subsequently pipelines the resets to allow for fan-out control and re-timing. The output from the `reset_processor` module is a synchronous reset signal per clock domain.

The design consists of two instances of the reset_processor:

- One instance is for the NoC circuits, using `i_reset_n`, the PLL lock signals, and generates the `nap_rstn` that is synchronous to `i_clk`.
- Another instance is for DC interface which utilizes the `ddr4_1_rstn` and generates the `dci_rstn` used to reset the logic for DC interface.

In addition `i_training_rstn` is synchronized to `i_training_clk` before being input to the training module.

Unused Pins

The DDR DC interface consists of a number of pins that are unused by this reference design. In order for correct project compilation, these pins must exist as ports in the top level of the design. When the design netlist is input to ACE, it requires that these ports are present in order to correctly connect between the DDR subsystem (defined in `/src/acxip/ddr4_1.acxip`) and the user design.

A number of these unused pins are outputs from the user design; in all instances these unused signals should be driven to 0.

Simulation Configuration

Simulation Modes

This DDR4 user design release supports standalone, BFM and RTL modes of simulation)BFM is the default). The RTL mode simulation can be run by either appending `"FLOW=FULLCHIP_RTL"` to the simulation `make` command as shown below:

```
> make (other options) FLOW=FULLCHIP_RTL
```

Or, if the user will be regularly running RTL simulations, they can edit the `FLOW` variable in the `/sim/<simulator>/Makefile` as below:

```
FLOW := FULLCHIP_RTL
```

Simulation File lists

Reviewing the Makefile, the user will note that the primary differences when selecting between the BFM and RTL flows are the use of either the `system_files_bfm.f` or `system_files_rtl.f` simulation file lists respectively. These files allow the user to enable any defines or override any parameters for a particular simulation. In addition any additional include directories can be specified here.

Note

The file for the user design and the simulation environment are normally specified in `/src/filelist.tcl`.

For the reference design the key difference between these two files is:

- The enabling of the RTL subsystem using the `ACX_DDR4_FULL` define.
- The addition of the memory model specific defines and associated include directories.

Obtaining the DDR4 Memory Simulation Models

The DDR4 RTL mode requires the user to access the DDR4 memory RTL Verilog simulation models. Achronix does not redistribute memory models directly to the user. However, the user can download these models directly from preferred vendor websites.

This reference design was developed using models from Micron and SKHynix (downloadable via the links below):

- Micron_ddr4_verilog_models
- SKhynix_ddr4_sim_models

To obtain simulation models from other vendors, users should contact their technical support teams directly.

Including Vendor Simulation Models into RTL Simulation

A user should place the relevant DDR4 memory model files to the provided vendor specific directories `/src/tb/micron and /src/tb/skhynix` for Micron and SKHynix respectively. While simulating in RTL mode, the user can choose the DDR4 memory model from their preferred memory vendor by enabling the right memory models in the files below. The following example shows the simulation being configured for Micron memory model and VCS as the simulator:

- In the `/src/filelist.tcl` file uncomment the paths `micron/protected_vcs/*`

```
set tb_verilog_files {
tb_noc_memory_behavioural.sv
micron/protected_vcs/arch_package.sv
micron/protected_vcs/proj_package.sv
micron/protected_vcs/dimm_interface.sv
micron/protected_vcs/StateTable.svp
micron/protected_vcs/MemoryArray.svp
micron/protected_vcs/ddr4_model.svp
# micron/protected_modelsim/arch_package.sv
# micron/protected_modelsim/proj_package.sv
# micron/protected_modelsim/dimm_interface.sv
# micron/protected_modelsim/StateTable.svp
# micron/protected_modelsim/MemoryArray.svp
# micron/protected_modelsim/ddr4_model.svp
# skhynix/ddr4_vcs.vp
# skhynix/ddr4_modelsim.vp
tb_ddr4_ref_design.sv
}
```

- Turn on the corresponding +define+ in the `/sim/<simulator>/system_files_rtl.f` file

```
# Turn-on below defines for Micron Model
+define+ACX_USE_MICRON_MODEL
+define+ACX_DDR4_3200
+define+DDR4_X8
+define+DDR4_2G

# Turn-on below defines for SKHynix Model
//+define+ACX_USE_HYNIX_MODEL
//+define+DDR4_8Gx8
//+define+DDR4_3200AA
//+define+ACX_DDR4_3200

# ACE libraries must be defined first as they are referenced by the fullchip files that
follow
+incdir+$ACE_INSTALL_DIR/libraries/

# Turn-on below library to enable Micron model and appropriate simulator
+incdir+../../src/tb/micron/protected_vcs/
//+incdir+../../src/tb/micron/protected_modelsim/

# Turn-on below library to enable SKHynix model
//+incdir+../../src/tb/skhynix

$ACE_INSTALL_DIR/libraries/device_models/7t_simmodels.v
```

Installing the Reference Design

Downloading

The design is available for download on the Achronix self-service FTP site (https://secure.achronix.com), located in the folder `/public/Achronix/Reference_Designs/Speedster7t`.

Packaging

The design is packaged in a zip file archive, using the following format:

```
<design_name>_<design_version>_<date_of_packaging>.zip
```

The archive contains all the source code, scripts to build the design, simulation script files, and optionally, GUI-based project files.

In addition the root of the archive contains release notes identifying the change history of the design.

Operating System

Linux

The design scripts and build flow are natively designed for Linux and have been built and tested using CentOS 7 and Ubuntu 16.04LTS. The flows use Makefiles, so are Linux shell agnostic.

Windows

To enable the scripted simulation or implementation flows to operate under Windows 10, the user must install one of the following;

- A Linux environment installed under Windows, such as www.cygwin.com The installation should include a Tcl interpreter and the `make` executable.
- A Tcl Interpreter and a `make` executable for Windows. There are many variants available, including both no cost and licensed versions.

If the user is unable to install any of the options above, it is still possible to run some of the flows under Windows:

- Implementation – GUI projects are provided for both Synplify and ACE, enabling builds to be done using the tools directly in GUI mode.
- Simulation – A Tcl script is provided which can be executed in the QuestaSim Tcl console. This script enables simulation using QuestaSim under Windows. For further details, refer to the Simulation section.

Note

For correct operation of any script flow, ACE should be installed in a directory without spaces in the path name, e.g., `C:\achronix\8.1.1\Achronix_CAD_Environment`. Additionally the environment variable `ACE_INSTALL_DIR` must use "/" as path separators rather than "\", for example:

`ACE_INSTALL_DIR = C:/achronix/8.1.1/Achronix_CAD_Environment/Achronix`

Tool Versions

All designs were tested with the following tools:

Table 6: ***Minimum Tool Versions***

Software	Version
ACE	8.2.1
ACE IO BFM Simulation Package	8.2.update2
Synplify Pro	Q-2020.03X
Mentor Questa	10.7c-1
Synopsys VCS	O-2018.09-SP1-1

Environment Variables

ACE_INSTALL_DIR

In order to support relocatable projects, the designs make use of an environment variable, `ACE_INSTALL_DIR`. This variable should be set to point to the ACE installation directory (where `ace.exe` or `ace` is installed). This variable is then used by both synthesis and simulation to correctly locate the ACE library files.

Note

For correct operation of any script flow, ACE must be installed in a directory without spaces in the path name. Examples of suitable paths are:

- Windows – `C:\achronix\8.2.1\Achronix_CAD_Environment`
- Linux – `/opt/achronix/ace/8.2.1`

Additionally when installed under Windows, the environment variable `ACE_INSTALL_DIR` must use "/" as path separators rather than "\", for example:

```
ACE_INSTALL_DIR = C:/achronix/8.2.1/Achronix_CAD_Environment/Achronix
```

Directory Structure

The design has a directory structure that allows for easy navigation and separation of source and generated files. The directory structure can be easily modified to suit a users preferred layout; however, if the structure is modified, then the necessary makefiles and build scripts will need to be modified to suit. To support portability, relative paths are used, as opposed to absolute paths; with the use of environment variables to select the root directory.

Table 7: *Design Directory Structure*

Directory			Decription
<design_name>			Root directory. Contains release notes.
	/build		Synthesis and place-and-route building
	/doc		Documentation and user guide
	/scripts		Scripts used for building and simulation
	/sim		Simulation area
		/vcs	Synopsys VCS simulation files
		/questa	Mentor QuestaSim simulation files
	/src		Source code
		/ace	ACE GUI project
		/acxip	ACE .acxip configuration files
		/constraints	Placement and timing constraint files
		/include	RTL include files
		/ioring	ACE generated ioring files
		/rtl	RTL source files
		/syn	Synplify Pro GUI project
		/tb	RTL testbench files
		filelist.tcl	Filelist used for building and simulation

Language Support

The reference designs support both Verilog, SystemVerilog and VHDL RTL languages. These can be used for both building, and standalone simulation. If the full-chip BFM simulation is used, that environment requires that the top-level testbench is Verilog or SystemVerilog. However, the design under test (DUT) may be written in VHDL.

Constraint Files

The design is supplied with a full set of constraint files, located under `/src/constraints`. These files demonstrate how various constraints and directives may be applied to the design. The constraint files, and their usage is detailed below.

Table 8: ***Constraint File Details***

File Name	Usage
`ace_constraints.sdc`	Timing constraints used by ACE. More than one SDC file can be included in an ACE project.
`ace_options.sdc`	Control ACE settings, such as flow mode, speed grade, reporting of unconstrained paths.
`ace_placements.pdc`	Fix locations of elements within the ACE fabric, and creation of placement regions. **Note** Pin placements between the fabric and the I/O ring are automatically created by the I/O ring designer, and provided in the ioring PDC files.
`synplify_constraints.fdc`	Synplify FPGA design constraints. Set attributes such as compile points, or default memory styles.
`synplify_constraints.sdc`	Synplify timing constraints. Clock and timing constraints. Should match those set in `ace_constraints.sdc`.
`synplify_options.tcl`	Control Synplify settings, such as top module. Create synthesis specific parameters, generics and defines.

I/O Ring Constraint Files

In addition to the constraint files listed above, the I/O ring generates constraint files specific to the interface between the fabric core and the I/O ring containing the interface subsystems. These constraint files can be auto-generated by ACE from the respective `.acxip` files; however, to aid the build flow, pre-generated files are provided for projects that require configuration of the I/O ring interface subsystems. These files are located under `/src/ioring`. The purpose of each file is detailed below.

Table 9: ***IO Ring Constraint File Details***

File Name	Usage
`<design_name>_ioring.sdc`	I/O timing constraints for direct connection interfaces, between the fabric and the I/O ring.
`<design_name>_ioring.pdc`	Placement of the fabric I/O pins, to assign them to the direct connection interfaces in the I/O ring.
`<design_name>_ioring_util.xml`	Used by ACE to generate a combined utilization report, combining the fabric and I/O ring resources.

I/O Ring Simulation Package

Many designs require a simulation overlay named I/O Ring Simulation Package. This package combines the full RTL of the network on chip (NoC) with bus functional models (BFMs) of the interface subsystems that surround the NoC and FPGA fabric. This combination of true RTL for the NoC and models for the interface subsystems allows users to develop their designs within a fast responsive simulation environment, while achieving cycle-accurate interfaces from the NoC, and representative cycle responses from the hard interface subsystems. This simulation environment allows a designer to iterate rapidly to develop and debug their design.

Description

The I/O ring simulation structure provides full RTL code for the NoC and BFM models of the surrounding interface subsystems. This structure is wrapped within a SystemVerilog module named per device, i.e., ac7t1500. The user needs to instantiate one instance of this module within their top-level testbench.

In addition, the simulation package provides binding macros such that the user can bind between elements of their design and the same elements within the device. For example, the design may instantiate a NoC access point (NAP). It is then necessary to bind this NAP instance to the NAP in the correct location within the NoC by using the `ACX_BIND_NAP_SLAVE, `ACX_BIND_NAP_MASTER, `ACX_BIND_NAP_HORIZONTAL, or `ACX_BIND_NAP_VERTICAL macro, whichever is appropriate for the design.

Similarly it is necessary to bind between the ports on the design and the direct-connection interface (DCI) for the interface subsystem. Each DCI within the device is connected to a SystemVerilog interface. This interface can then be directly accessed from the top-level testbench, and signals assigned between the SystemVerilog interface and the ports on the design.

Version Control

The I/O ring simulation package is version controlled. Within a release, new functions may be added and older functions may be deprecated or replaced. The release is indicated both in the package name (`ACE_<major>.<minor>.<patch>_IO_BFM_sim_<update>.zip/tgz`) and in the `readme` file placed in the root directory of the package.

To ensure that the correct version of the I/O ring simulation package is used, a task must be included within the design testbench to confirm the version compatibility. This function should be instantiated as detailed below:

Code

```
// For this example the FPGA instance is ac7t1500
initial begin
    // Ensure correct version of sim package is being used
    // This design requires 8.1.2.update2 as a minimum
    ac7t1500.require_version(8, 1, 2, 2);
end
```

require_version() Task

The require_version task has four arguments. In order

- Major Version – Will match the major version of the release
- Minor Version – Will match the minor version of the release

- Patch – Will match the patch version of the release (optional)
- Update – Will match the update number of the release (optional)

If either of patch or update is not specified, then these should be set to 0; for example, for the 8.3 release, the arguments would be set as 8,3,0,0.

Note

The values can be expressed either a numbers (0-9) or as strings ("0" – "9") or as letters ("a/A", "b/B"), with the letters "a" and "b" represent alpha or beta releases. When deciding on the priority of a release, a number represents a more recent release than a letter; therefore, 8.3.alpha (defined as as 8,3,"a",0) will precede the full 8.3 release (designated as 8,3,0,0).

Example Design

An example structure of the I/O ring simulation, combined within a testbench, and with a design under test is shown in the diagram (see page 18) below. This example shows the macros required for the slave NAPs, and the DCIs for two instances of the GDDR6 subsystem. For other forms of NAPs, or for other DCI types, such as DDR, consult the Bind Macros (see page 20) and Direct-Connection Interfaces (see page 22) tables.

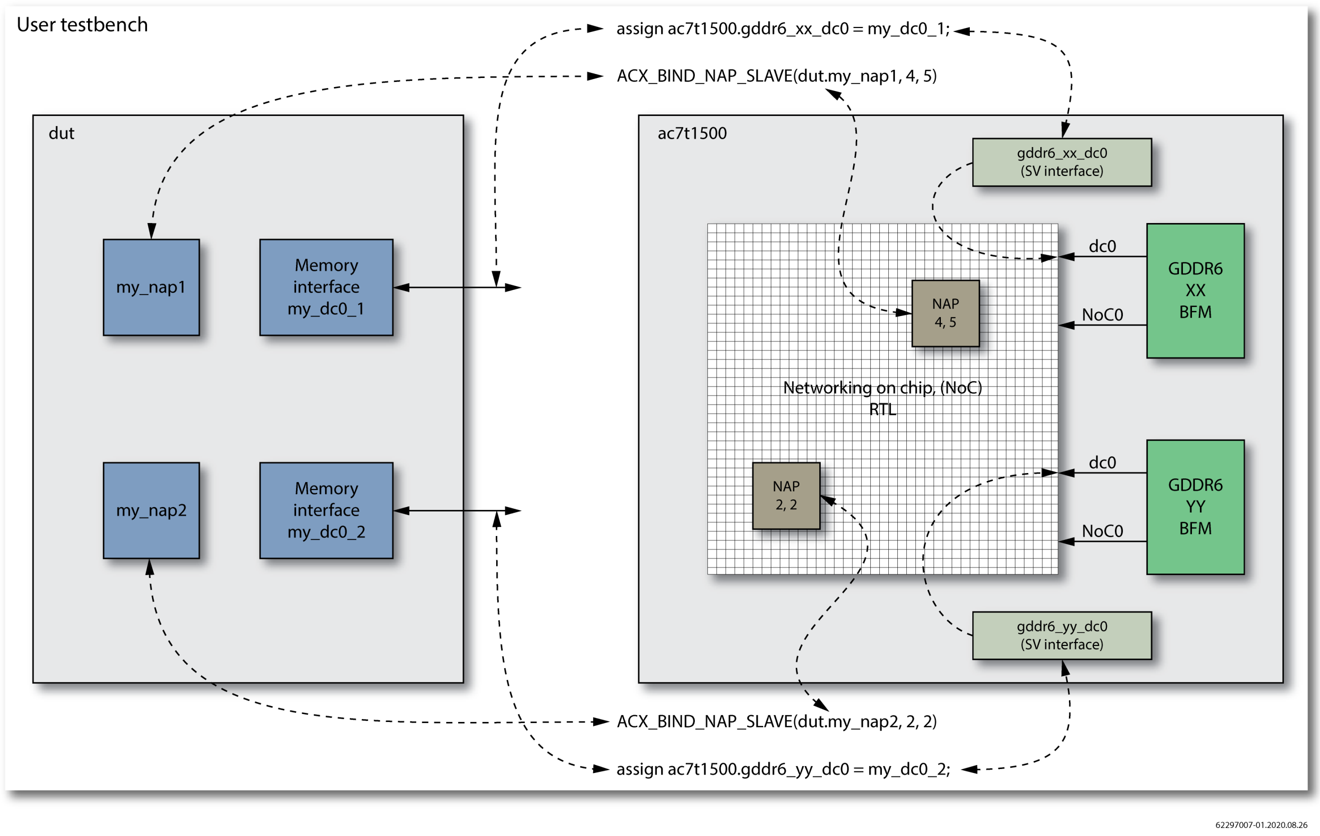
User testbench
assign ac7t1500.gddr6_xx_dc0 = my_dc0_1;
ACX_BIND_NAP_SLAVE(dut.my_nap1, 4, 5)
dut
my_nap1
Memory interface my_dc0_1
my_nap2
Memory interface my_dc0_2
ac7t1500
gddr6_xx_dc0 (SV interface)
dc0
NoC0
GDDR6 XX BFM
NAP 4, 5
Networking on chip, (NoC) RTL
NAP 2, 2
dc0
NoC0
GDDR6 YY BFM
gddr6_yy_dc0 (SV interface)
ACX_BIND_NAP_SLAVE(dut.my_nap2, 2, 2)
assign ac7t1500.gddr6_yy_dc0 = my_dc0_2;
62297007-01.2020.08.26

Figure 3: *Example I/O Ring Simulation Structure*

In the example above, there are two NAPs, `my_nap1` and `my_nap2`. In addition there are two direct-connect interfaces, `my_dc0_1` and `my_dc0_2`. In the top-level testbench bindings are made between the NAPs in the design and the NAPs within the device using the ACX_BIND_NAP_SLAVE macro. This macro supports inserting the coordinates of the NAP within the NoC in order that the simulation is aligned with physical placement of the NAP on silicon.

The DCIs are ports on the user design; these ports are then assigned to the appropriate signals within the device direct-connect SystemVerilog interface.

The Verilog code to instantiate the above is shown below. This example is based on using the ac7t1500 device

```
// ----------------------------------------------
// Instantiate Speedster7t1500
// ----------------------------------------------
// Connect the chip ready port
// Note : All ac7t1500 ports are defined, so can be directly connected if required
ac7t1500 ac7t1500( .FCU_CONFIG_USER_MODE (chip_ready ) );

// Set the verbosity options on the messages
initial begin
    ac7t1500.verbosity = 3;
end

// ----------------------------------------------
// Bind NAPs
// ----------------------------------------------
// Bind my_nap1 to location 4,5
`ACX_BIND_NAP_AXI_SLAVE(dut.my_nap1,4,5);
// Bind my_nap2 to location 2,2
`ACX_BIND_NAP_AXI_SLAVE(dut.my_nap2,2,2);

// ----------------------------------------------
// Connect to DC interfaces
// ----------------------------------------------
// Create signals to attach to direct-connect interface
logic                         my_dc0_1_clk;
logic                         my_dc0_1_awvalid;
logic                         my_dc0_1_awaddr;
logic                         my_dc0_1_awready;
.....
logic                         my_dc0_2_clk;
logic                         my_dc0_2_awvalid;
logic                         my_dc0_2_awaddr;
logic                         my_dc0_2_awready;
.....

// Connect signals to gddr6_xx_dc0 interface within ac7t1500 device
// Inputs to device
assign ac7t1500.gddr6_xx_dc0.awvalid  = my_dc0_1_awvalid;
assign ac7t1500.gddr6_xx_dc0.awaddr   = my_dc0_1_awaddr;
....
// Outputs from device
assign my_dc0_1_awready = ac7t1500.gddr6_xx_dc0.awready;
....

// Connect signals to gddr6_xx_dc0 interface within ac7t1500 device
```

```
    // Inputs to device
    assign ac7t1500.gddr6_yy_dc0.awvalid  = my_dc0_2_awvalid;
    assign ac7t1500.gddr6_yy_dc0.awaddr   = my_dc0_2_awaddr;
    ....
    // Outputs from device
    assign my_dc0_2_awready = ac7t1500.gddr6_yy_dc0.awready;
    ....

    // ----------------------------------------------
    // Remember to connect the clock!
    // ----------------------------------------------
    assign my_dc0_1_clk = ac7t1500.gddr6_xx_dc0.clk;
    assign my_dc0_2_clk = ac7t1500.gddr6_yy_dc0.clk;
```

Note

When using bind macros, the user is able to specify the column and row coordinates of the target NAP. To ensure consistency between simulation and silicon, the user should add matching placement constraints to the ACE placement `.pdc` file, for example:

In simulation

```
`ACX_BIND_NAP_AXI_SLAVE(dut.my_nap1,4,5);
```

In place and route

```
set_placement -fixed {i:my_nap} {s:x_core.NOC[4][5].logic.noc.nap_s}
```

Chip Status Output

From initial simulation start, the device operates similarly to its silicon equivalent with an initialization period when the device is in reset. In hardware this occurs during configuration as the bitstream is loaded. After this initialization period, the device asserts the `FCU_CONFIG_USER_MODE` signal to indicate that it has entered user mode, whereby the design starts to operate.

It is suggested that the top-level testbench monitor `FCU_CONFIG_USER_MODE` and only starts to drive stimulus into the device once this signal is asserted (shown in the example above by use of a testbench `chip_ready` signal).

Bind Macros

The following bind statements are available.

Table 10: *Bind Macros*

Macro	Arguments	Description
ACX_BIND_NAP_HORIZONTAL	user_nap_instance, noc_colunm, noc_row	To bind a horizontal streaming NAP, instance ACX_NAP_HORIZONTAL.
ACX_BIND_NAP_VERTICAL	user_nap_instance, noc_colunm, noc_row	To bind a vertical streaming NAP, instance ACX_NAP_VERTICAL.

Macro	Arguments	Description
ACX_BIND_NAP_AXI_MASTER	user_nap_instance, noc_colunm, noc_row	To bind an AXI master NAP, instance ACX_NAP_AXI_MASTER.
ACX_BIND_NAP_AXI_SLAVE	user_nap_instance, noc_colunm, noc_row	To bind an AXI slave NAP, instance ACX_NAP_AXI_SLAVE.

Direct-Connect Interfaces

The following direct-connect interfaces are available.

Table 11: *Direct-Connect Interfaces*

Subsystem	Interface Name	Physical Location	GDDR6 Channel	SystemVerilog Interface Type	Data Width	Address Width
GDDR6	gddr6_1_dc0	West 1	0	t_ACX_AXI4	512	33
GDDR6	gddr6_1_dc1	West 1	1	t_ACX_AXI4	512	33
GDDR6	gddr6_2_dc0	West 2	0	t_ACX_AXI4	512	33
GDDR6	gddr6_2_dc1	West 2	1	t_ACX_AXI4	512	33
GDDR6	gddr6_5_dc0	East 1	0	t_ACX_AXI4	512	33
GDDR6	gddr6_5_dc1	East 1	1	t_ACX_AXI4	512	33
GDDR6	gddr6_6_dc0	East 2	0	t_ACX_AXI4	512	33
GDDR6	gddr6_6_dc1	East 2	1	t_ACX_AXI4	512	33
DDR	ddr4_dc0	South	NA	t_ACX_AXI4	512	40

Note

Not all interfaces are available in all devices. Please consult the appropriate device datasheet to understand which interfaces are present in the selected device.

Clock Frequencies

In addition to binding to the interfaces, it is possible to control the frequencies of the clocks generated by these interfaces. For design integrity the clock frequencies set within simulation should match the desired design operating frequencies. For design implementation, the frequencies are configured within the ACE IO Designer tool. For simulation, the `set_clock_period` function is provided.

The example below shows setting the GDDR6 East 1 controller to an operating frequency of 1 GHz (suitable for 16 Gbps operation). As the DC interface operates at half the controller frequency, it is then configured for 500 MHz.

Using this method the user can first ensure that the simulation operates at the correct frequencies. Second, they are able to operate each subsystem at a different frequency if required.

```
// Set default GDDR6 clock frequency to 1000 ps = 1GHz
localparam GDDR6_CONTROLLER_CLOCK_PERIOD = 1000;

// Configure the NoC interface of GDDR6 E1 to 1GHz
ac7t1500.clocks.set_clock_period("gddr6_5_noc0_clk", GDDR6_CONTROLLER_CLOCK_PERIOD);

// Configure the DC interface of GDDR6 E1 to 500MHz, (double the period of the NoC interface)
ac7t1500.clocks.set_clock_period("gddr6_5_dc0_clk", GDDR6_CONTROLLER_CLOCK_PERIOD*2);
```

Note

The `set_clock_period` function is within the ac7t1500 model. This model has a default timescale value of 1 ps; therefore, the specified clock period will be applied in picoseconds, irrespective of the timescale value of the calling module.

The following clock frequency interfaces are available

Table 12: *Clock Frequency Interfaces*

Subsystem	Interface Name	Physical Location	GDDR6 Channel
GDDR6	gddr6_0_noc0_clk	West 0 NoC	0
GDDR6	gddr6_0_noc1_clk	West 0 NoC	1
GDDR6	gddr6_1_noc0_clk	West 1 NoC	0
GDDR6	gddr6_1_noc1_clk	West 1 NoC	1
GDDR6	gddr6_2_noc0_clk	West 2 NoC	0
GDDR6	gddr6_2_noc1_clk	West 2 NoC	1
GDDR6	gddr6_3_noc0_clk	West 3 NoC	0
GDDR6	gddr6_3_noc1_clk	West 3 NoC	1
GDDR6	gddr6_4_noc0_clk	East 0 NoC	0
GDDR6	gddr6_4_noc1_clk	East 0 NoC	1
GDDR6	gddr6_5_noc0_clk	East 1 NoC	0
GDDR6	gddr6_5_noc1_clk	East 1 NoC	1
GDDR6	gddr6_6_noc0_clk	East 2 NoC	0
GDDR6	gddr6_6_noc1_clk	East 2 NoC	1
GDDR6	gddr6_7_noc0_clk	East 3 NoC	0
GDDR6	gddr6_7_noc1_clk	East 3 NoC	1

Subsystem	Interface Name	Physical Location	GDDR6 Channel
GDDR6	gddr6_1_dc0_clk	West 1 DCI	0
GDDR6	gddr6_1_dc1_clk	West 1 DCI	0
GDDR6	gddr6_2_dc0_clk	West 2 DCI	0
GDDR6	gddr6_2_dc1_clk	West 2 DCI	1
GDDR6	gddr6_5_dc0_clk	East 1 DCI	0
GDDR6	gddr6_5_dc1_clk	East 1 DCI	1
GDDR6	gddr6_6_dc0_clk	East 2 DCI	0
GDDR6	gddr6_6_dc1_clk	East 2 DCI	1
DDR	ddr4_noc0_clk	South NoC	NA
DDR	ddr4_dc0_clk	South DCI	NA
PCIe	pciex16_clk	Gen5 PCIe x16	NA
Configuration	cfg_clk	System wide configuration clock	NA

Configuration

A number of the interface subsystems require configuration at power-up. In the physical device, this configuration would be performed by the bitstream pre-programming the relevant configuration registers. Within the simulation environment, there are tasks that can read configuration files and apply those files to the relevant interface subsystem. An example of applying a configuration is shown in the code snippet below

```
// -------------------------
// Configuration
// -------------------------

// Call function within device to configure the registers
// By using fork-join, the two configurations will be run in parallel, configuring both
// Ethernet blocks.  This saves overall simulation time.
// Both blocks are configured the same, hence the use the same file
initial
begin
    fork
        ac7t1500.fcu.configure( "ethernet_cfg.txt", "ethernet0" );
        ac7t1500.fcu.configure( "ethernet_cfg.txt", "ethernet1" );
    join
end
```

Startup Sequence

While the task `fcu.configure()` is processing the configuration (including waiting for any polling to return a valid value), the Chip Status Output (see page 20) is not asserted. This behavior mirrors that of the device where the device will only enter user mode once configuration is completed.

The simulation testbench can issue configuration processes as shown above, and once the Chip Status Output (see page 20) is asserted, the testbench will know the device is correctly configured. The testbench can then proceed to apply the necessary tests.

fcu.configure() Task

The task `fcu.configure` has the following arguments:

```
fcu.configure ( <configuration filename>, <interface subsystem name> );
```

The following interface subsystem names are supported:

Table 13: *Configuration Subsystem Names*

Subsystem	Interface Name	Physical Location
GDDR6	gddr6_0	West 0
GDDR6	gddr6_1	West 1
GDDR6	gddr6_2	West 2
GDDR6	gddr6_3	West 3
GDDR6	gddr6_4	East 0
GDDR6	gddr6_5	East 1
GDDR6	gddr6_6	East 2
GDDR6	gddr6_7	East 3
DDR	ddr4	South
Ethernet	ethernet0	North
Ethernet	ethernet1	North

Table Note

Configuration subsystem interface names are case sensitive

Configuration File Format

The configuration file has the following format:

```
# ------------------------------------------
# Config file
# Supports both # and // comments
# ------------------------------------------

# A comment line
// Another comment line

# Format is <cmd> <addr> <data>

# Commands are
 "w" - write
 "r" - read
 "v" - read and verify

# Address is 28-bit, (7 hex characters).  This supports the configuration
# memory space of an interface subsystem

# Data is 32-bit, (8 hex characters).

# For reads, put 0x0 for the data
# For verify put the expected data value

# Examples

# Writes
w 00005c0 76543210
w 0000014 00004064

# Reads
r 00005c0 00000000
r 0000014 00000000

# Verify
v 00005c0 76543210
v 0000014 00004064
```

SystemVerilog Interfaces

The following SystemVerilog interfaces are defined, and are used for DCI assignments.

Note

The interface below is only available in the simulation environment. For code that must be synthesized, users need to define their own SystemVerilog interfaces, or use one of the interfaces predefined within the reference designs.

```
interface t_ACX_AXI4
    #(DATA_WIDTH = 0,
      ADDR_WIDTH = 0,
      LEN_WIDTH  = 0);

   logic                       clk;       // Clock reference
   logic                       awvalid;   // AXI Interface
   logic                       awready;
   logic [ADDR_WIDTH -1:0]     awaddr;
   logic [LEN_WIDTH -1:0]      awlen;
   logic [8 -1:0]              awid;
   logic [4 -1:0]              awqos;
   logic [2 -1:0]              awburst;
   logic                       awlock;
   logic [3 -1:0]              awsize;
   logic [3 -1:0]              awregion;
   logic [3:0]                 awcache;
   logic [2:0]                 awprot;
   logic                       wvalid;
   logic                       wready;
   logic [DATA_WIDTH -1:0]     wdata;
   logic [(DATA_WIDTH/8) -1:0] wstrb;
   logic                       wlast;
   logic                       arready;
   logic [DATA_WIDTH -1:0]     rdata;
   logic                       rlast;
   logic [2 -1:0]              rresp;
   logic                       rvalid;
   logic [8 -1:0]              rid;
   logic [ADDR_WIDTH -1:0]     araddr;
   logic [LEN_WIDTH -1:0]      arlen;
   logic [8 -1:0]              arid;
   logic [4 -1:0]              arqos;
   logic [2 -1:0]              arburst;
   logic                       arlock;
   logic [3 -1:0]              arsize;
   logic                       arvalid;
   logic [3 -1:0]              arregion;
   logic [3:0]                 arcache;
   logic [2:0]                 arprot;
   logic                       aresetn;
   logic                       rready;
   logic                       bvalid;
   logic                       bready;
   logic [2 -1:0]              bresp;
   logic [8 -1:0]              bid;

   modport master (input  awready, bresp, bvalid, bid, wready, arready, rdata, rlast, rresp,
rvalid, rid,
                   output awaddr, awlen, awid, awqos, awburst, awlock, awsize, awvalid, awregion,
                          bready, wdata, wlast, rready, wstrb, wvalid,
                          araddr, arlen, arid, arqos, arburst, arlock, arsize, arvalid, arregion);

   modport slave  (output awready, bresp, bvalid, bid, wready, arready, rdata, rlast, rresp,
rvalid, rid,
                     input  awaddr, awlen, awid, awqos, awburst, awlock, awsize, awvalid, awregion,
                            bready, wdata, wlast, rready, wstrb, wvalid,
                            araddr, arlen, arid, arqos, arburst, arlock, arsize, arvalid, arregion);

endinterface : t_ACX_AXI4
```

Installation

There is a simulation package per device, available for both Linux and Windows. The packages are named `<device>_IO_BFM_Sim_<version>.tgz` for Linux and `<device>_IO_BFM_Sim_<version>.zip` for Windows. The packages are available on the Achronix self-service FTP site at secure.achronix.com, located in the `/Achronix/ACE/Speedster7t` folder. As each device package is independent, it is possible to either download only the device the user is targeting, or all device packages.

Any package is only required to be installed once — it is common for all designs targeting the selected device.

ACE Integration

Upgrading an Existing Installation

If a version of the simulation package was previously installed into ACE, it is recommended to first delete the existing simulation package before upgrading to ensure the integrity of the new installation. To delete an existing package, navigate to `<ACE_INSTALL_DIR/system/data/yuma-alpha-rev0` and remove the `/sim` directory. Then return to the root of the ACE installation and proceed with the instructions for First Installation (see page 27) below.

First Installation

The recommended installation method is to merge the contents of the package into the current ACE installation. The package contains a root directory `/system`. The contents of this folder should be merged with the selected ACE installation `/system` folder.

Warning!

The contents of the simulation package consists of files that are not present in the base ACE installation. These files should not replace or overwrite any existing files. However, if the user has already downloaded an earlier version of the simulation package, then they should select "overwrite" to ensure the latest version of the simulation files are written to the ACE installation.

Standalone

In certain instances it may not be possible for a user to modify their existing ACE installation. In these cases it is possible to install the package separately and to simulate using files from both this simulation package and the existing simulation files within ACE.

To install as standalone, simply uncompress the package to a suitable location.

Note

All reference designs are configured for the simulation package to be integrated within ACE. If the standalone method is selected, the user must edit the necessary environment variables in the reference design makefiles.

Environment Variables

The locations of both ACE and the simulation package are controlled by two environment variables. For all reference designs these two variables must be set before simulating.

ACE_INSTALL_DIR

The environment variable ACE_INSTALL_DIR must be set to the directory location of the `ace`, or `ace.exe` executable. This variable is used by both simulation and synthesis to locate the correct device library files.

ACX_DEVICE_INSTALL_DIR

The environment variable ACX_DEVICE_INSTALL_DIR is used to select the device I/O ring simulation files. It should be set to the base directory of the device files within the I/O ring simulation package. For example if the ac7t1500 device is selected, then the device base directory is yuma-alpha-rev0.

When the installation is done as ACE integration mode, then the following setting should be used:

```
ACX_DEVICE_INSTALL_DIR = $ACE_INSTALL_DIR/system/data/yuma-alpha-rev0
```

When the installation is done as standalone, the the following setting should be used:

```
ACX_DEVICE_INSTALL_DIR = <location of standalone package>/system/data/yuma-alpha-rev0
```

Simulating the Reference Design

Supported Simulators

The designs have a consistent simulation environment, providing scripts for Mentor QuestaSim and Synopsys VCS simulators.

Location

All designs have a `/sim` directory located in the design root directory. Within this directory there are `/vcs` and `/questa` directories for each of the simulators.

Simulation Flows

Where applicable the simulation supports a number of flow options, which offer a balance between speed and accuracy. Not all flow options are available with all reference designs; the relevant makefiles will list what flow options can be set.

Standalone

Any NAP in the design uses a standalone model bound to the NAP, modelling memory behavior. This mode is the quickest simulation to run, but is the least cycle accurate. The NAP will only interact with its own memory model; therefore, this mode does not support multiple NAPs designed to access a common memory.

Full-Chip BFM

This uses a model of the full chip, with cycle-accurate NoC. There are then bus functional models (BFMs) for all the hardened interfaces around the NoC. These BFMs have representative delays, allowing this mode to offer near cycle-accurate simulations. This mode does not require the interface subsystems to perform initialization and calibration steps, offering a quicker iterative time compared to a full cycle-accurate simulation.

The full-chip BFM simulations require the I/O Ring Simulation Package to have been downloaded and installed.

Full-Chip RTL

This mode uses the full RTL of the subsystem combined. if necessary, with a cycle-accurate model of any necessary external component (such as a memory). This configuration gives a fully cycle-accurate simulation representing the final silicon operation. For most of these simulations it will be necessary to configure the relevant subsystems using the provided configuration files. As these simulations are using the full RTL of the subsystem, they run slower than the BFM equivalent simulations, while offering complete timing accuracy.

Note

To obtain the encrypted RTL of the various subsystems, a second licensed simulation package is required. Please contact Achronix Support to arrange licensing and access to this package.

Build Options

Within each simulator directory is a makefile. This makefile can be edited by the user to configure the simulation to their needs. The following variables need to be set:

- `FLOW` – To match the selected simulation flow. The options (detailed in the makefile) are STANDALONE, FULLCHIP_BFM or FULLCHIP_RTL.
- `TOP_LEVEL_MODULE` – Preset for the supplied design. However, if the user ports the design to their own testbench, this variable must be updated.
- `ACX_DEVICE_INSTALL_DIR` – Points to the directory (normally under ACE) where the target device files are stored, for example, `$ACE_INSTALL_DIR/system/data/yuma-alpha-rev0`. For further information consult the I/O ring simulation installation instructions.

Prerequisites

Before running any installation, the user must ensure the following are configured:

- `ACE_INSTALL_DIR` environment variable: This should point to the ACE installation directory, where the ACE executable is located.
- Path to the required simulator. For VCS this should also include the VCS_HOME environment variable.

Auto File List Generation

The simulation file list is auto generated from the `../../src/filelist.tcl` file. The script to create the simulation file list is `../../scripts/create_sim_project.tcl`, and it uses a template file `../scripts/sim_template.f` or `../../scripts/sim_template_bfm.f` to define the general simulation options. The create script is called by the specific simulator makefile as detailed below. The resultant file, `sim_filelist.f`, combines the template contents with the specific list of files in `../../src/filelist.tcl`.

Files

In each `/sim/vendor` directory the following scripts are located:

- `makefile` – Makefile supporting the various simulation flows. Default target is to compile and run the simulation.
- `system_files_bfm.f` – (Full-chip BFM flow only) List of system files used by the full-chip BFM flow. Any user defines can also be added to this file.
- `system_files_rtl.f` – (Full-chip RTL flow only) List of system files used by the full-chip RTL flow. Also any defines required to specify which subsystems should be cycle-accurate RTL rather than BFM. The defines are named as <subsystem name>_FULL, i.e., GDDR6_W2_FULL. Any user defines can also be added to this file.

VCS Only

- `fullchip_bfm_vcs_waiver.cfg` – (Full-chip BFM flow only) Waiver file to remove benign warnings.
- `session.sim_output_pluson.vpd.tcl` – Session file for DVE waveform viewer.

QuestaSim Only

- `wave.do` – Waveform file.
- `qsim_<design_name>.do` – Non-makefile GUI flow. OS independent.

Running the Simulation

For all simulator flows there is a `makefile` located in the simulation directory. The makefiles support the following build options

VCS

code

```
$ make             : Default flow. Compile with no debug options, run in batch mode writing the output to a VPD file.
$ make run       : Same as "make".
$ make debug     : Build with debug options (increased visibility of lower level variables). Run in batch mode writing to a VPD file.
$ make open_dve : Open the DVE waveform viewer and load the VPD file and viewing session file.
$ make clean      : Delete all generated and temporary files.
```

QuestaSim

code

```
$ make            : Default flow. Compile with no debug options, run in batch mode writing the
output to a WLF file.
$ make run      : Default flow. Same as "make".
$ make debug    : Build with debug options (increased visibility of lower level variables). Open
GUI with wave.do and allow user to run interactively.
$ make open_wave: Open the GUI waveform viewer. Load the generated WLF file and viewing wave.do fi
le.
$ make clean      : Delete all generated and temporary files.
```

QuestaSim Non-Makefile Flow

To support environments that do not natively support makefiles (such as Windows), there is an additional QuestaSim non-makefile flow using QuestaSim `.do` files. The following steps are required to use this flow:

1. Navigate to `sim/questa`
2. Launch QuestaSim GUI. Normally:

   ```
   $ vsim
   ```

   Within the QuestaSim GUI launch the script

   ```
   $ do qsim_<design_name>.do
   ```

Note

The QuestaSim script uses the ACE_INSTALL_DIR environment variable. For correct operation of this, (or any other), script flow, ACE should be installed in a directory without spaces in the path name, e.g., `C:\achronix\8.2.1\Achronix_CAD_Environment`. Additionally the environment variable `ACE_INSTALL_DIR` must use "/" as path separators rather than "\", for example:

`ACE_INSTALL_DIR = C:/achronix/8.2.1/Achronix_CAD_Environment/Achronix`

Note

The do script is configured for the `FULLCHIP_BFM` flow. It can be modified to match the `STANDALONE` flow by selecting the appropriate options commented within the script.

Results Verification

All the designs make use of a self-checking testbench which compares the results generated from the RTL to a verified output. The verified output can come from a number of sources, either a math package, a software model, or an RTL behavioral model. The details of the applicable verification source is given in the detail of each individual design.

Building the Reference Design

The designs make use of a consistent build environment, using a makefile and scripts to run both Synplify Pro and ACE in batch mode.

Prerequisites

Before running any installation, the user must ensure the following are configured:

- `ACE_INSTALL_DIR` environment variable. This variable must point to the ACE installation directory where the ACE executable is located.
- ACE should be in the environment path. The easiest method is to add `$ACE_INSTALL_DIR` to the path variable.
- Synplify Pro should be in the environment path.

Batch Flow

In the root directory, there is a `/build` directory, within which there is a `makefile`. Before running the makefile, ensure the prerequisites above have been met. The relative paths within the makefile are intended to be run from the `/build` directory. If the makefile is moved to a new location or called outside of this directory, then the paths will require amending accordingly.

When the makefile is run (with the default options), it will create the following;

- `/build/results/syn` directory – Synplify Pro is executed in batch mode, synthesizing the design, which generates a netlist in `/results/syn/rev_1/<design_name.vm`. If synthesis is unsuccessful, the user should consult `/results/syn/rev_1/<design_name>.srr` for details of any synthesis failure. Options to the generated Synplify Pro project file are controlled by `/src/constraints /synplify_options.tcl.`
- `/build/results/ace` directory – After synthesis, ACE is run in evaluation mode (meaning that no I/O pins need be specified). If any options are required for the ACE-generated project, these are controlled by the `/src/constraints/ace_impl_options.tcl` file.

Makefile Options

The makefile supports multiple build flow options

code

```
$ make             : Default flow. Synthesize and build a single implementation with ACE
$ make run       : Same as "make"
$ make syn_only    : Synthesis only
$ make pnr_only : Run ACE place and route only. This requires synthesis to have previously been
run.
$ make run_mp   : Run multiprocess. Synthesize and build multiple implementations with ACE
multiprocess.
$ make clean      : Delete all generated and temporary files
```

Constraint Files

In addition to the constraint files listed above, additional files may be used in any build flow. All constraint files are located in `/src/constraints`:

- `ace_timing.sdc, <design_name>.sdc` – timing constraint files used by both synthesis and ACE.
- `<design_name>.fdc` – FPGA constraints used by Synplify Pro to set non-timing related directives and attributes, such as compile points.
- `<design_name>.pdc` – placement constraints used by ACE.

The full list of what files are used in the flow, and by which tool, can be determined by referring to the relevant `/src/filelist_xx.tcl` file.

GUI Flow

The design has pre-generated GUI project files for both ACE and Synplify Pro. These files are located in `/src/ace` and `/src/syn` directories respectively. The user can open these to interactively edit or run builds.

Note

When using the GUI projects, any generated files will be placed beneath the GUI project file directory.

- For Synplify Pro, the revision directory, rev_1 etc. will be generated in `/src/syn/rev_1`.
- For ACE, any implementation directory will be generated as `/src/ace/impl_<name>`.
- For ACE, if I/O Designer is used to generate new constraint files, the default directory for those files is `/src/ace/ioring_design`.

When builds are done using the batch flow, the flow writes out both the relevant project files under the `/build/results` directory. Within this directory are the generated project files for both ACE and Synplify Pro. The user can open both of these project files in GUI mode and interactively re-run or edit the builds.

Revision History

Version	Date	Description
1.0	24 Feb 2020	• Initial Release.
1.1	28 Feb 2020	• Added Design Considerations (see page 7) section. • Provided more detail under AXI Data Packet Generator and Checker. (see page)
1.2	21 May 2020	• Added description of clock frequency function. • Removed clock frequency warning message. • Corrected launch of QuestaSim GUI command. • Updated example instantiation of ac7t1500.
2.0	23 Jul 2020	• Updated and added signal/port names that reflect changes from ACE 8.2 I /O Designer
3.0	16 Oct 2020	• Updated the design to include RTL mode that support transactions for cycle accurate memory models • Updated to reflect changes for ACE 8.2.1 I/O Designer
3.1	03 Nov 2020	• Additional details on BFM versus RTL delays • Enhanced usage and description of training and polling module

Data Acceleration

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

Notice of Disclaimer