

# Speedster7t GDDR6 Reference Design (RD017)



---

**September 28, 2020****Reference Design**

---

## Introduction

---

The GDDR6 reference design demonstrates the ability to perform read/write transactions to all eight GDDR6 subsystems via the network-on-chip (NoC) interface and four of the GDDR6 subsystems through the direct-connect (DC) interface. The design can be used to test both channels for each GDDR6 subsystem, with the option to expand the design to all 16 channels from all the eight GDDR6 subsystems using the NoC interface and repeat the same for those that support the DC interface.

This design release primarily showcases how to interface between the NAP module or DC interface ports from the fabric to the GDDR6 memories and simulate the same. The user will also be able to take this design through synthesis and the ACE flow for place and route.

## Description

---

This design release enables the user to run simulation in one of two modes: bus-functional models (BFM) mode or RTL mode:

- In BFM mode, the testbench incorporates BFMs for the GDDR6 subsystem and memories.
- In RTL mode, the GDDR6 subsystem is represented by encrypted RTL and the GDDR6 memories are represented by cycle-accurate simulation models from the memory vendor.

The user can choose which mode to use on a per GDDR6 subsystem basis.

The design consists of the following logic blocks:

### GDDR6 Register Checker

Each GDDR6 subsystem needs to be configured by the FCU at startup. The GDDR6 register checker polls specific registers inside each active GDDR6 subsystem to determine if they have completed the configuration process. Once the status of the specific registers matches the expected value, the checker enables the function of the rest of the design to start GDDR6 related transactions.

This register checker uses one NAP module to read all active GDDR6 subsystems via the NoC.

### AXI Data Generator/Checker

The AXI data generator block generates the AXI data and addresses to be sent to the interfacing GDDR6 subsystem. The generated data and address can be either sequential or random packets to random or sequential addresses using the AXI bus. This data is then sent as inputs via the respective NAP/NoC to the corresponding GDDR6 subsystem. The design is configured to write 256-bit data via the NoC interface and 512-bit data through the DC interface to sequential addresses in the GDDR6 memory.

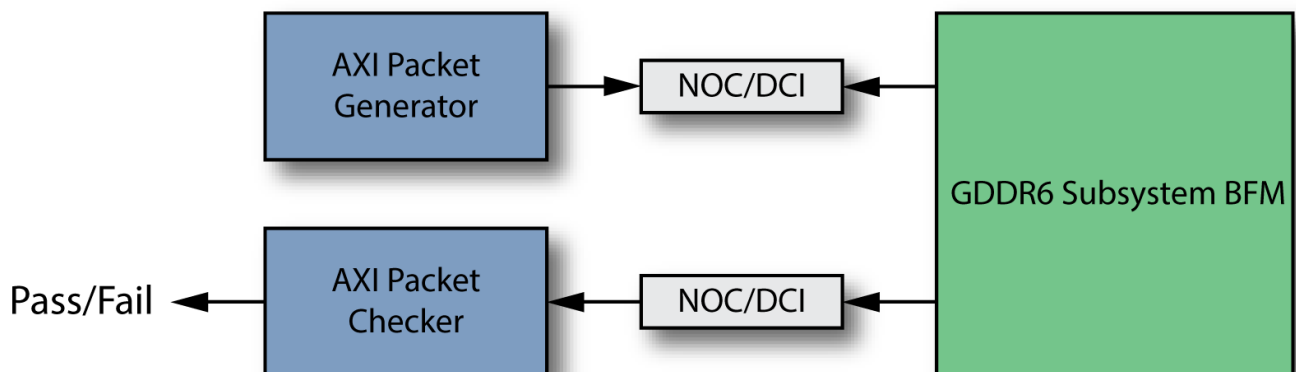
The AXI checker logic compares the data read from the GDDR6 subsystem for each generated address with the expected data. The test indicates a pass when read data from the memory matches the expected data. Both the packet generator and checker logic run at a rate of 500 MHz on the fabric.

## NAP Modules

In addition to being used to read GDDR6 registers, the NAP modules interface with the data generation logic to transfer data via the NoC to the respective GDDR6 subsystem. The design uses all eight GDDR6 subsystems available in the device to test the NoC interface and four GDDR6 subsystems for the direct-connect interface. Each GDDR6 interface has its own data generation and checker logic and corresponding NAP. Each of these NAPs can be bound to any arbitrary location in simulation and during placement in ACE. This flexibility enables the user to test scenarios where the GDDR6 subsystems can handle transactions from any NAP located in the fabric.

## GDDR6 BFM (BFM Mode Only)

This module is the bus functional model for the entire GDDR6 subsystem that mimics the GDDR6 controller, PHY plus memory setup. The BFM delay models are also designed to account for delays in each of these corresponding blocks. Each of the eight instantiated GDDR6 BFMs for the NoC use a unique NAP target address that indicates the NAP/NoC structure where each of the data packets need to be routed. The target address (indicated at the top-level design file) also has a designated bit to indicate which of the two channels in the GDDR6 subsystem is being used for the transactions. The GDDR6 BFM runs at a data rate of 16 Gbps and hence uses a clock source of 1 GHz.

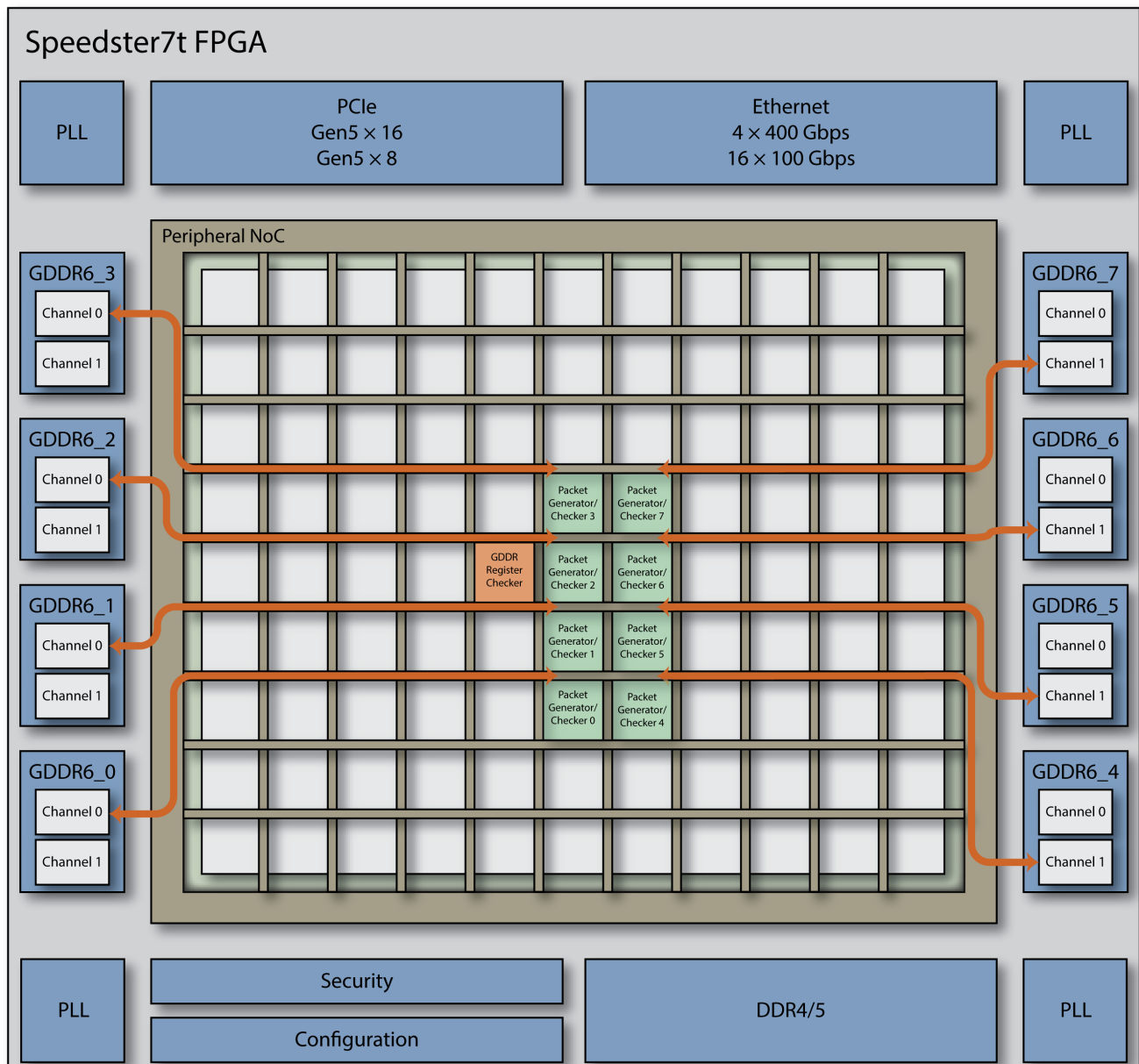


62293531-01.2020.26.08

**Figure 1: Block Diagram of a Single GDDR6 Subsystem with NoC/DC Interface Test Structure**

## Test Structure Using the NoC-GDDR6 Interface

The diagram below shows how each of these modules interact with the GDDR6 subsystems via the NoC. The user can choose the number of NoC or DC interface-enabled GDDR6 subsystems and set the desired Controller ID for each of them in the top-level module.



62293531-02.2020.08.08

**Figure 2: Generator/Checker Logic Interacting with all GDDR6 Subsystems via the NoC**

## Addressing

Each packet generator/checker logic is bound to a NAP to interface with an associated GDDR6 subsystem. The following values are used in the control ID part of the GDDR6 address scheme to indicate to each NAP to send and receive transactions to a particular GDDR6 subsystem. More details on the NAP address scheme can be found in the *Speedster7t Network on Chip User Guide* (UG089).

**Table 1: GDDR6 Subsystem Control IDs**

GDDR6 Controller	Control ID [36:33]
GDDR6_0	Channel 0 - 4'b1100
	Channel 1 - 4'b1101
GDDR6_1	Channel 0 - 4'b0100
	Channel 1 - 4'b0101
GDDR6_2	Channel 0 - 4'b0000
	Channel 1 - 4'b0001
GDDR6_3	Channel 0 - 4'b1000
	Channel 1 - 4'b1001
GDDR6_4	Channel 0 - 4'b1111
	Channel 1 - 4'b1110
GDDR6_5	Channel 0 - 4'b0111
	Channel 1 - 4'b0110
GDDR6_6	Channel 0 - 4'b0011
	Channel 1 - 4'b0010
GDDR6_7	Channel 0 - 4'b1011
	Channel 1 - 4'b1010

Since the GDDR6 address width is 9 bits, these values are defined as 9-bit values in the `GDDR6_ID_NOC` local parameter in the top-level RTL file.

```
// GDDR6 target address ID. Pages are defined in NoC User Guide, Address Mapping
// Defined as 9 bit field. 9th bit(LSB) controls channel selection. All NoC interfaces are set to
// channel 1
localparam [71:0] GDDR6_ID_NOC = {9'd10, 9'd2, 9'd6, 9'd14, 9'd9, 9'd1, 9'd5, 9'd13};
```

**Note**

The GDDR6 controllers on the east side use even addresses for channel 1, whereas the west side uses odd addresses. Therefore, four `GDDR6_ID_NOC` values are even, and four are odd.

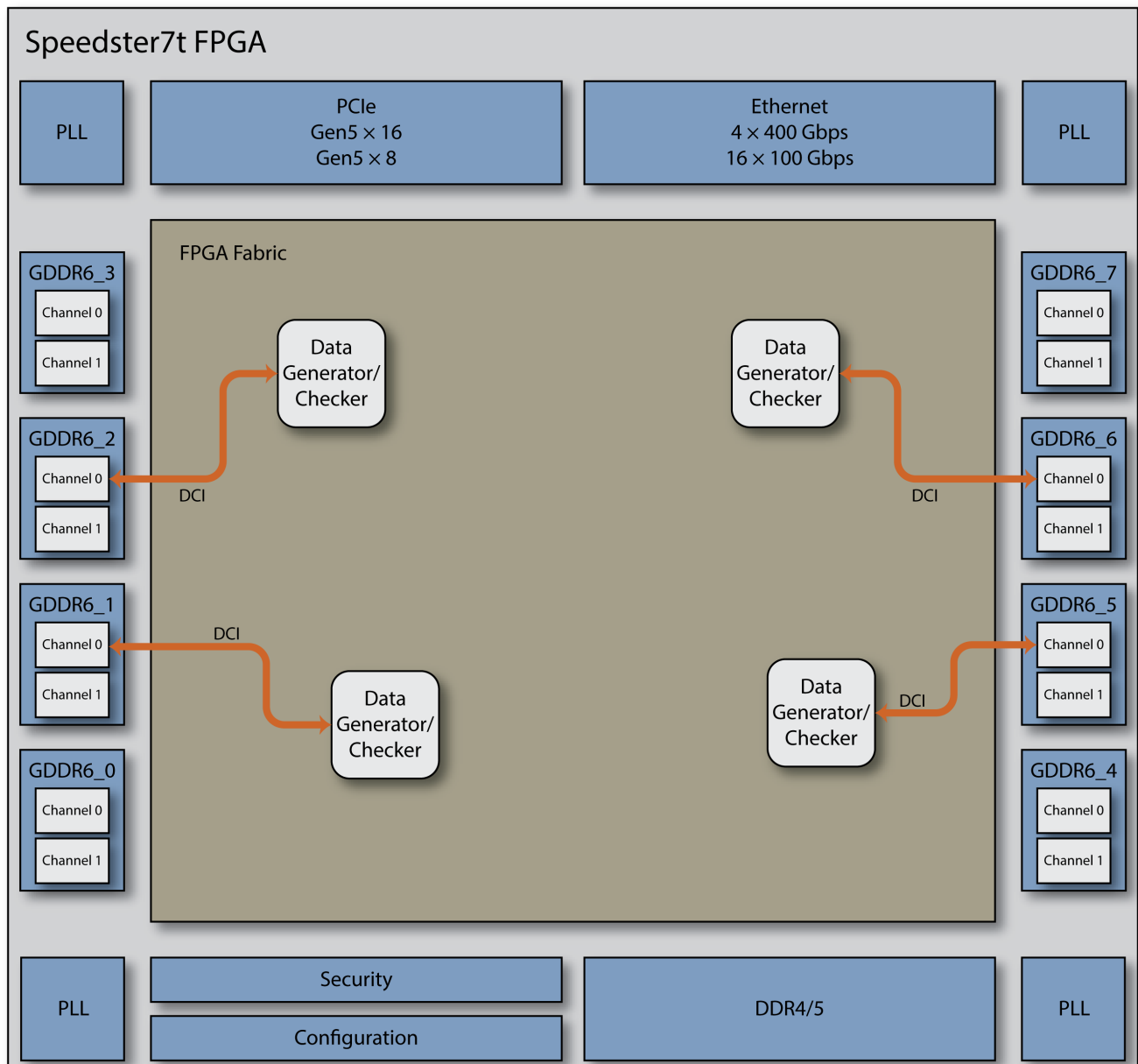
## Placement

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 implementation, the location 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.

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.

## Test Structure Using the DC-GDDR6 Interface

The diagram below shows how each of these modules interact with the GDDR6 subsystems via the DC interface. There are four GDDR6 subsystems namely GDDR6 1, 2, 5 and 6 that support the direct-to-fabric connect interface as shown in the figure below. The user can specify the desired subsystem as shown in the table [GDDR6 Subsystem Control ID \(see page 35\)](#). In this reference design, all DC interfaces are connected to channel 0 of GDDR6 subsystems. For further details on the DC interface connection, refer to the *Speedster7t GDDR6 User Guide* (UG091).



62293531-03.2019.14.02

**Figure 3: Generator/Checker Logic Interacting with Four GDDR6 Subsystems that Have DC Interface Connections**

## Ports and Parameters

The GDDR6 reference design top-level RTL module needs to set the following parameters:

**Table 2: GDDR6 Reference Design Top-Level RTL Parameters**

Parameter Name	Default Value	Description
GDDR6_NOC_CONFIG	8'b11111111	Signifies which ones of the 8 gddr6 subsystems will be enabled for NoC interface testing, 1 – enable; 0 – disable. bit[0]: gddr6_0 bit[1]: gddr6_1 ... bit[7]: gddr6_7
GDDR6_DCI_CONFIG	4'b1111	Signifies which ones of the 4 gddr6 subsystems will be enabled for DC interface testing, 1 – enable; 0 – disable. bit[0]: gddr6_1 bit[1]: gddr6_2 bit[2]: gddr6_5 bit[7]: gddr6_6
NUM_TRANSACTIONS	256	Indicates the total number of AXI transactions to performed. It can be set up to 8000. If set to 0, the transactions keep running.

The GDDR6 reference design top-level RTL module has the following input output ports:

**Table 3: GDDR6 Reference Design Top-Level RTL Ports**

Signal Name	Width	Description
i_clk	1	Clock that drives logic interfacing with the NAP.
gddr6*_dc0_clk	1	Clock that drives logic interfacing with the DC interface.
i_reset_n	1	Active-low reset.
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; 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; tied to 1'b1.
gddr6*_dc0_*	–	DC interface AXI signals.

## Design Considerations

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

### Clocking

Within the supplied ACE GUI project, there are three PLLs, named `p11` (for all the reference design logic clocks) and `p11_gddr_NE` and `p11_gddr_NW` (for all the GDDR6 subsystem and DC interface clocks), created as internal IP for the design. The GDDR6 subsystems' clocks can only be supplied from PLLs located on the same edge of the FPGA; therefore, two separate PLLs are needed to supply the clocks for the GDDR6 subsystems located on the East and West edges of the FPGA.

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 4: Reference Design Clocks**

Clock	Direction	Frequency (MHz)	Description
<b>pll Clocks</b>			
<code>ref_clk_NE</code>	Input	200	External reference input clock for pll.
<code>noc_clk</code>	Output	200	Reference clock to NOC. <sup>(†)</sup>
<code>i_clk</code>	Output	500	Global clock for user logic.
<b>pll_gddr_NE and pll_gddr_NW Clocks</b>			
<code>ref_clk_NE</code>	Input	200	External reference input clock for pll_gddr_NE.
<code>ref_clk_NW</code>	Input	200	External reference input clock for pll_gddr_NW.
<code>gddr_ctrl_clk_NE</code>	Output	1000	Reference clock to GDDR6 controller subsystems on the East side of the FPGA. <sup>(†)</sup>
<code>gddr_ctrl_clk_NW</code>	Output	1000	Reference clock to GDDR6 controller subsystems on the West side of the FPGA. <sup>(†)</sup>
<code>gddr6_*_dci_clk</code>	Output	500	Reference clock to the corresponding GDDR6 direct-connect interface. <sup>(†)</sup> This clock is the source of the GDDR6 DC interface <code>gddr6_*_dc0_clk</code> input to the design.



#### 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 each NAP interface and one for each DC interface. Each NAP interface is clocked by the `i_clk` clock domain, and each DC interface is clocked by the corresponding `gddr6*_dc0_clk` clock domain. These clocks are asynchronous to one another; therefore, 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 testbench file `/src/tb/tb_gddr_ref_design.sv`.



### Note

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

## Resets

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

**Table 5: Reset Sources**

Source	Description
<code>i_reset_n</code>	External asynchronous input.
<code>gddr6*_dc0_aresetn</code>	Output from each GDDR6 DC interface. Synchronous to the corresponding <code>gddr6*_dc0_clk</code>
<code>pll_lock</code>	PLL locked.
<code>pll_gddr*_lock</code>	GDDR6 PLLs locked.

Good design practice requires that resets are carefully synchronized and processed. The reference design uses multiple instances of the `reset_processor` module which combines multiple sources, correctly synchronizes them to the specified 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 five instances of the `reset_processor`:

- One instance is for the NoC circuits, using `i_reset_n`, the PLL lock signals, and generates a reset, (`nap_rstn`), that is synchronous to `i_clk`.
- One instance per DC interface, resulting in four instances in total. These instances each synchronize the same signals as the NoC instance, along with their respective DC interface `resetn`, producing a local signal, `dci_output_rstn`, per DC interface instance.

## GDDR6 Channel Assignments for NoC and DC Interfaces

The reference design is configured to use channel 1 for all the NoC interfaces and channel 0 for all the DC interfaces, so that the packet generator/checker does not create an error if both were writing different values to the same memory space. However, in a user design it may be desirable for both these interfaces to access the same address space in the memory, i.e., the NoC writing to a memory location, and the DC interface reading from the same address or vice versa.

For the DC interfaces, only the `gddr6_*_chan1` signals are driven. The NoC access is controlled by memory addressing (see [Addressing \(see page 3\)](#)).

## Number of GDDR6 Subsystems

As the table "[GDDR6 Reference Design Top-Level RTL Parameters \(see page 22\)](#)" shows, the user can use parameters to configure the number of gddr6 subsystems used in this design. The default configuration is to enable all GDDR6 subsystems.

If a user configures the design to use only some of the GDDR6 subsystems, the default placement constraint file (`/src/constraints/ace_placement.pdc`) must also be edited to only place the GDDR6 subsystems used.

# Simulation Configuration

## Simulation Modes

This design supports both BFM and RTL mode of simulation. The default mode is BFM. To run the simulation in RTL mode, either append "FLOW=FULLCHIP\_RTL" to the simulation make command:

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

Or, edit the flow variable in the Makefile to ensure that all future simulation runs are using RTL mode

```
FLOW := FULLCHIP_RTL
```

Furthermore, the user can choose which GDDR6 subsystems are to be replaced with a cycle-accurate RTL model in RTL mode simulation by modifying the applicable simulation `/sim/<simulator>/system_files_rtl.f` file and enabling the corresponding `+define+`. The GDDR6 subsystems that are not enabled in this file will remain in BFM mode even during an RTL mode simulation. The following example shows GDDR subsystems 1 and 5 configured to use RTL, all other GDDR subsystems will continue to use BFM models.

```
# Turn on GDDR RTL
# Enable the desired GDDR memory controllers below
# Any undefined controllers will use their BFM model
+define+ACX_GDDR6_1_FULLL
+define+ACX_GDDR6_5_FULLL
```

Understandably the more GDDR6 subsystems are enabled during RTL simulation, the longer the run time will be.



### Warning

#### How to Obtain the GDDR6 Memory Simulation Models

RTL mode simulation requires use of the GDDR6 memory simulation models. Achronix is not able to provide these models directly to the user. Rather, the user needs to acquire these models directly from their preferred vendor.

The testbench and reference design were developed using models from Micron Technology Inc. To obtain these models, the user should contact Micron Sales or Technical Support directly.

## How to Use GDDR6 Memory Simulation Models

Once the GDDR6 memory models have been obtained, they should copy the relevant simulation files to the `/src/tb/gddr_model/` directory. The user should then compare the instantiation of their selected memory models to the instantiation of the default models in `/src/tb/tb_gddr_ref_design.sv` to align the pin-out of their model with the testbench. To improve readability of the testbench, the GDDR6 model pin-out and connections

are provided by two macros, `ACX_GDDR6_MODEL_WIRE` and `ACX_GDDR6_PORT_CONNECT`. These macros are defined in `/src/include/gddr_model_names.svh`. The user may either edit these macros to suit their model, or else instantiate the model directly in the testbench.

**Note**

As the testbench was developed using models from Micron, if these models are used, no changes should be necessary.

Once satisfied with the connectivity of the GDDR6 models, include these files into the simulation by editing `/src/filelist.tcl`. Under `tb_verilog_files` then add `./gddr_model/<gddr6_model_filename(s)>`. For example, using the Micron models, the the local `/src/filelist.tcl` should have the following entry:

```
set tb_verilog_files {  
  tb_noc_memory_behavioural.sv  
  tb_gddr_ref_design.sv  
  gddr_model/micron_gddr6_bfm.v  
}
```

## GDDR6 Data Rate

User can also define GDDR6 data rate of the simulation by enabling the corresponding lines in these files:

- BFM mode – `/sim/<simulator>/system_files_bfm.f`
- RTL mode – `/sim/<simulator>/system_files_rtl.f`

In RTL mode simulation, the GDDR6 subsystems must be configured with a configuration file. Once the data rate is set in the above files, the simulation will automatically use the corresponding bitstream files stored here:

```
/sim/gddr_config_16Gbps.txt  
/sim/gddr_config_14Gbps.txt  
/sim/gddr_config_12Gbps.txt
```

# 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](http://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.update1
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\Achronix_CAD_Environment`
- Linux – `/opt/achronix/ace/8.2`

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/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		Description
<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. <div>  <b>Note</b>  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. </div>
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.

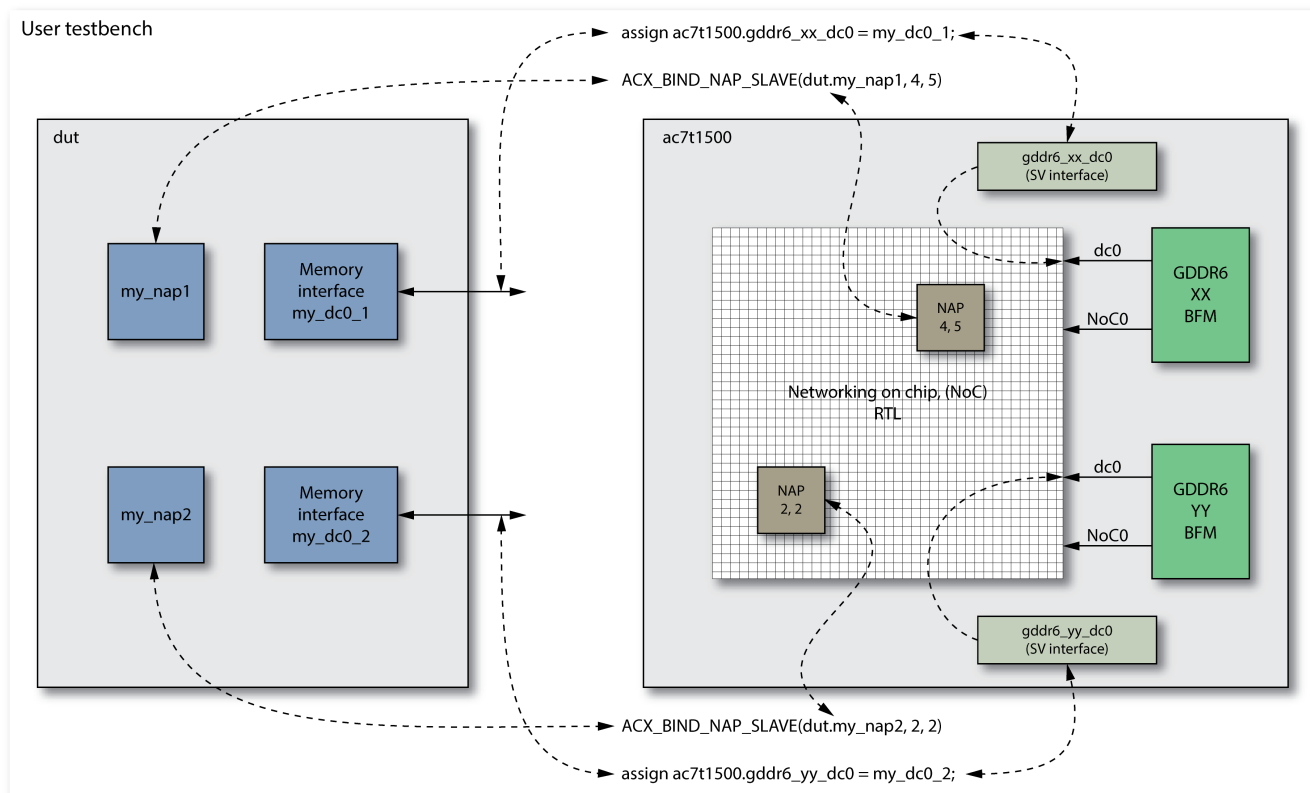


#### Note

The values can be expressed either as 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 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 \(see page 35\)](#) and [Direct-Connection Interfaces \(see page 22\)](#) tables.



62297007-01.2020.08.26

**Figure 4: 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_column, noc_row	To bind a horizontal streaming NAP, instance ACX_NAP_HORIZONTAL.
ACX_BIND_NAP_VERTICAL	user_nap_instance, noc_column, noc_row	To bind a vertical streaming NAP, instance ACX_NAP_VERTICAL.

Macro	Arguments	Description
ACX_BIND_NAP_AXI_MASTER	user_nap_instance, noc_column, noc_row	To bind an AXI master NAP, instance ACX_NAP_AXI_MASTER.
ACX_BIND_NAP_AXI_SLAVE	user_nap_instance, noc_column, 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
"p" - poll until bit set

# 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
# For poll, set relevant bits to 1.
# Poll will complete when all bits set to 1 are asserted

# Examples

# Writes
w 00005c0 76543210
w 0000014 00004064

# Reads
r 00005c0 00000000
r 0000014 00000000

# Verify
v 00005c0 76543210
v 0000014 00004064

# Poll
# Waiting for bits [17:16] and [0] to be asserted
p 0123456 00030001

```

## 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](https://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 file.
$ 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\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/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	<ul style="list-style-type: none"> <li>Initial Release.</li> </ul>
1.1	03 Mar 2020	<ul style="list-style-type: none"> <li>Added Design Considerations section.</li> <li>Changes to the block diagram to point to the correct channel.</li> <li>Other content updates.</li> </ul>
1.2	06 Mar 2020	<ul style="list-style-type: none"> <li>Updated notes on autogenerated SDC based on ACE 8.1.1 fix.</li> </ul>
1.3	21 May 2020	<ul style="list-style-type: none"> <li>Added description on how clock periods are defined in testbench.</li> <li>Corrected launch of QuestaSim GUI command.</li> <li>Updated example instantiation of ac7t1500.</li> </ul>
2.0	22 Jul 2020	<ul style="list-style-type: none"> <li>Added description of RTL mode simulation, reformatted BFM simulation description.</li> <li>Updated signal/port names and clock description to match design changes due to ACE 8.2 I/O Designer change.</li> <li>Removed warning for autogenerated constraint files because issues have been fixed.</li> <li>Removed table for testbench parameters because they are now being controlled by defines in <code>system_files_*.f</code>.</li> </ul>
2.1	23 Jul 2020	<ul style="list-style-type: none"> <li>Corrected the polarity description of *_oen signals listed in <a href="#">Table: GDDR6 Reference Design Top-Level RTL Ports (see page 7)</a>.</li> </ul>
2.2	22 Sep 2020	<ul style="list-style-type: none"> <li>Added description of GDDR6 Register Checker logic block.</li> <li>Updated diagrams to reflect accurate placement and updated IP block names and signal names.</li> <li>Added description to detail RTL simulation mode related files and scripts.</li> </ul>
2.3	28 Sep 2020	<ul style="list-style-type: none"> <li>Added details on why two separate PLLs are needed for GDDR6 subsystems.</li> <li>Updated minimum ACE version needed to 8.2.1.</li> </ul>



Achronix Semiconductor Corporation

2903 Bunker Hill Lane  
Santa Clara, CA 95054  
USA

Website: [www.achronix.com](http://www.achronix.com)  
E-mail : [info@achronix.com](mailto:info@achronix.com)

---

Copyright © 2020 Achronix Semiconductor Corporation. All rights reserved. Achronix, Speedcore, Speedster, and ACE are trademarks of Achronix Semiconductor Corporation in the U.S. and/or other countries All other trademarks are the property of their respective owners. All specifications subject to change without notice.

## Notice of Disclaimer

The information given in this document is believed to be accurate and reliable. However, Achronix Semiconductor Corporation does not give any representations or warranties as to the completeness or accuracy of such information and shall have no liability for the use of the information contained herein. Achronix Semiconductor Corporation reserves the right to make changes to this document and the information contained herein at any time and without notice. All Achronix trademarks, registered trademarks, disclaimers and patents are listed at <http://www.achronix.com/legal>.