Speedster7t Device Manager User Guide (UG115)

Speedster FPGAs

Achronix
Data Acceleration

Copyrights, Trademarks and Disclaimers

Notice of Disclaimer

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

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

Table of Contents

Chapter 1 : Description

The Achronix device manager (ADM) provides automatic control of hard IP components such as GDDR6 , DDR4/5, Ethernet, and PCIe, where control is complex for typical production systems. The ADM can also act as a bridge between the JTAG controller and the 2D NoC, allowing access to control and status registers (CSRs) via the JTAG interface. The status of the ADM itself, including errors during startup, can also be queried via the JTAG interface using Tcl commands in ACE.

On AC7t1500 devices, the ADM is implemented in the FPGA core and contains a soft microcontroller unit (MCU). On AC7t800 devices, the ADM is a hard IP located in the I/O ring. A block diagram of the ADM is shown in the following figure:

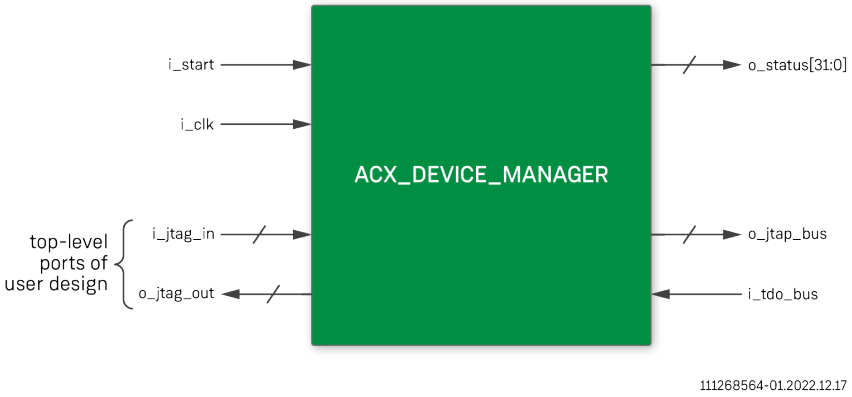

Figure 1 • ADM Block Diagram

The ADM operates in two phases: startup and background. In the startup phase, the ADM performs the necessary steps to configure hard IP interfaces. Many interfaces require some form of pre-calibration or link training. Before and during the startup phase, communication to these interfaces is not possible. The user design must wait until the startup phase is complete before accessing the interface. The end of the startup phase is indicated by an output status signal, `o_status`.

Following the startup phase, the ADM enters the background phase. In this phase, the ADM acts as bridge between the JTAG interface and the 2D NoC, and may perform periodic maintenance tasks such as temperature sensor monitoring, etc. The ADM contains an embedded NAP to communicate via the 2D NoC. The core of the ADM is a 32-bit embedded MCU with non-volatile firmware. The firmware is pre-programmed and cannot be modified by the user.

The ADM controls the configuration and operation of the GDDR6, DDR, Ethernet, and PCIe interfaces. Thus, any design that utilizes these IP blocks must also include a ADM instance. It is also recommended that all user designs include the ADM to support periodic monitoring tasks such as the reading of the temperature sensor.

Chapter 2 : Configuration

AC7t1400/1500 ADM (ACX_DEVICE_MANAGER) Configuration

The ADM soft IP can be found in the ACE IP Libraries under **Speedster7t → Core → Device Management → Device Manager**. The configurator has the following options:

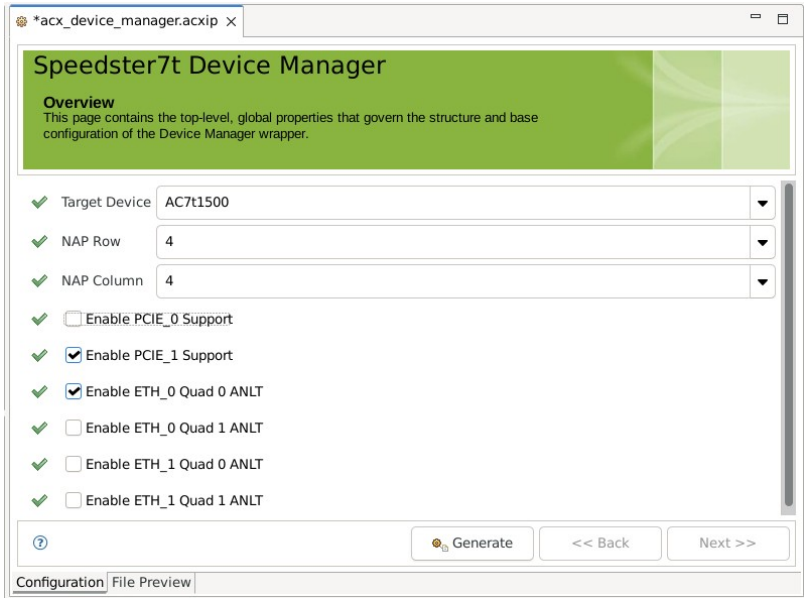

Figure 2 • Speedster7t Device Manager Soft IP Configuration

Table 1 • AC7t1400/1500 ADM Configuration Options

Name	Default	Range	Description
Target Device	AC7t1500	AC7t1400/AC7t1500	Target device of the project
NAP Row (1)	4'bx	1–4	Row coordinate for the ADM's built-in NAP
NAP Column (1)	4'bx	1–10	Column coordinate for the ADM's built-in NAP
Enable PCIE_0 Support	disabled	disabled/enabled	Enables support for PERST, Hot Reset, Gen2 De-emphasis, DBI Gateway, and FPGA Reconfiguration pin control for the PCIE_0 or PCIE_1 subsystem respectively. Required for correct PCIe operations.
Enable PCIE_1 Support	disabled	disabled/enabled	

Name	Default	Range	Description
Enable ETH_0 Quad 0 ANLT	disabled	disabled/enabled	Enables the Ethernet Auto-Negotiation & Link Training for Quad 0 on the ETH_0, Quad 1 ETH_0, Quad 0 ETH_1, and Quad 1 ETH_1 subsystems respectively. The Ethernet ANLT function only supports up to 4 lanes: 200G-CR4, 100G-CR4, 40G-CR4, 100G-CR2, 50G-CR1, and 25G-CR1. When enabled, top-level RTL wire connections between the ADM and the Ethernet subsystem are required. See Instantiation Example (page 46) section for details.
Enable ETH_0 Quad 1 ANLT	disabled	disabled/enabled	
Enable ETH_1 Quad 0 ANLT	disabled	disabled/enabled	
Enable ETH_1 Quad 1 ANLT	disabled	disabled/enabled	

Table Notes

1. The ADM must have the NAP row and column parameters set in the RTL. This differs from most other IP where it is recommended that the row and column location be set via the `.pdc` file. This requirement results from the fact that the ADM is encrypted. Thus, the path to the NAP instance in the RTL is also encrypted, making application of the row and column parameters difficult from a `.pdc` file.

File Generation

The configuration settings in the previous table prompt the generation of a ADM template design file in the selected location as shown in the following example:

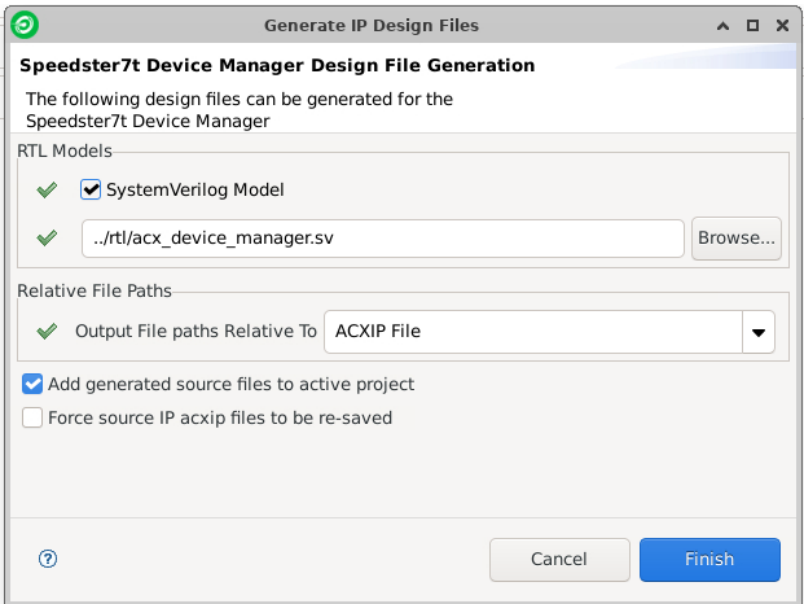

Figure 3 • ADM Design File Generation

Ports

Table 2 • ACX_DEVICE_MANAGER Ports

Name	Direction	Description
i_clk	Input	100 MHz clock required for ACX_DEVICE_MANAGER
i_start	Input	Start signal, must be kept high for the ACX_DEVICE_MANAGER to start. It is recommended to simply tie this high (1'b1), or to drive it with a PLL lock signal.
o_status[31:0]	Output	Ready and error status, see table below for status bits.
i_pcie_0_perstn	Input	When enabled, it is required to connect PCIE_0 PERSTN (from the PCIe connector), via GPIO or CLKIO input, to the exposed i_pcie_0_perstn pin.
i_pcie_1_perstn	Input	When enabled, it is required to connect PCIE_1 PERSTN (from the PCIe connector), via GPIO or CLKIO input, to the exposed i_pcie_1_perstn pin.
i_pcie_0_ltssm_state [5:0]	Input	When enabled, it requires the exposed i_pcie_0_ltssm_state input pin to be connected to the DCI LTSSM state pins for the PCIE_0 interface.
i_pcie_0_ltssm_state [5:0]	Input	When enabled, it requires the exposed i_pcie_1_ltssm_state input pin to be connected to the DCI LTSSM state pins for the PCIE_1 interface.
o_pcie_0_reconfig_fpga_n	Output	The active-low output signals to the board manager that the FPGA must be reprogrammed. This requires board support, and that the bitstream is stored in flash memory. This output is only enabled if both ENABLE_PCIE_0_HOT_RSTN and ENABLE_PCIE_RECONFIG_FPGA are set, otherwise this output will be 1.
o_pcie_1_reconfig_fpga_n	Output	The active-low output signals to the board manager that the FPGA must be reprogrammed. This requires board support, and that the bitstream is stored in flash memory. This output is only enabled if both ENABLE_PCIE_1_HOT_RSTN and ENABLE_PCIE_RECONFIG_FPGA are set, otherwise this output will be 1.
t_JTAG_INPUT i_jtag_in	Input	JTAG input: must be connected to a top-level port of the same t_JTAG_INPUT type. i_jtag_in[0] is the JTAG clock.
t_JTAG_OUTPUT o_jtag_out	Output	JTAG output: must be connected to a top-level port of the same t_JTAG_OUTPUT type.
t_JTAP_BUS o_jtap_bus	Output	JTAP bus output, to share the JTAG interface.
i_tdo_bus	Input	JTAP bus input, to share the JTAG interface. When the JTAG interface is not shared, tie this input to 1'b0.

Name	Direction	Description
i_eth0_quad0_an_link_good[3:0]	Input	Drive by PMA , indicates AN has negotiated a link speed for the corresponding lane, and is attempting to get link training/PCS working. Connect to the Ethernet subsystem at ETH0 Quad 0.
i_eth0_quad0_link_status[3:0]	Input	PCS lock achieved for 10-100G MAC. Driven from the Ethernet controller. 4 bits correspond to link status of each of the 4 lanes. Connect to the Ethernet subsystem at ETH0 Quad 0.
i_eth0_quad0_link_status_ored	Input	PCS lock achieved for 200G MAC. Driven from the Ethernet controller. Connect to the Ethernet subsystem at ETH0 Quad 0.
o_eth0_quad0_an_link_status[63:0]	Output	The confirmation of negotiated technology driven to PMA. Connect to the Ethernet subsystem at ETH0 Quad 0.
i_eth0_quad1_an_link_good[3:0]	Input	Same as above but connect to the Ethernet subsystem at ETH0 Quad 1.
i_eth0_quad1_link_status[3:0]	Input	
i_eth0_quad1_link_status_ored	Input	
o_eth0_quad1_an_link_status[63:0]	Output	
i_eth1_quad0_an_link_good[3:0]	Input	Same as above but connect to the Ethernet subsystem at ETH1 Quad 0.
i_eth1_quad0_link_status[3:0]	Input	
i_eth1_quad0_link_status_ored	Input	
o_eth1_quad0_an_link_status[63:0]	Output	
i_eth1_quad1_an_link_good[3:0]	Input	Same as above but connect to the Ethernet subsystem at ETH1 Quad 1.
i_eth1_quad1_link_status[3:0]	Input	
i_eth1_quad1_link_status_ored	Input	
o_eth1_quad1_an_link_status[63:0]	Output	

Status Signals

The `o_status` output signal shows the status after the startup phase is complete, and indicates if there were any errors during startup. Most designs need only monitor `o_status[0]` or `o_status[1:0]` to decide when the design can be started. The `o_status` value can also be read in ACE Tcl console with the `mcu_status` command. If errors occur, other ACE Tcl commands may give additional information.

Table 3 • AC7t1500 ACX_DEVICE_MANAGER Status Output

Status Bits	Meaning When Asserted
0	Startup is complete and successful. This signal can be used as the start signal for the user design.
1	Startup is complete. If there were no errors, `o_status[0]` is set as well. Otherwise, an error bit is also set.
3:2	ADM internal error. These bits should always read `2'b00` to indicate no error. If either bit is asserted, contact Achronix for assistance.
5:4	PCIE_1, PCIE_0 startup error
6	DDR4 startup error
14:7	GDDR_7...GDDR_0 startup error
16:15	ETH_1...ETH_0 startup error
17	Temperature alarm 0: Set when the FPGA temperature is ≥ 85°C
18	Temperature alarm 1: Set when the FPGA temperature is ≥ 100°C
31:19	Reserved

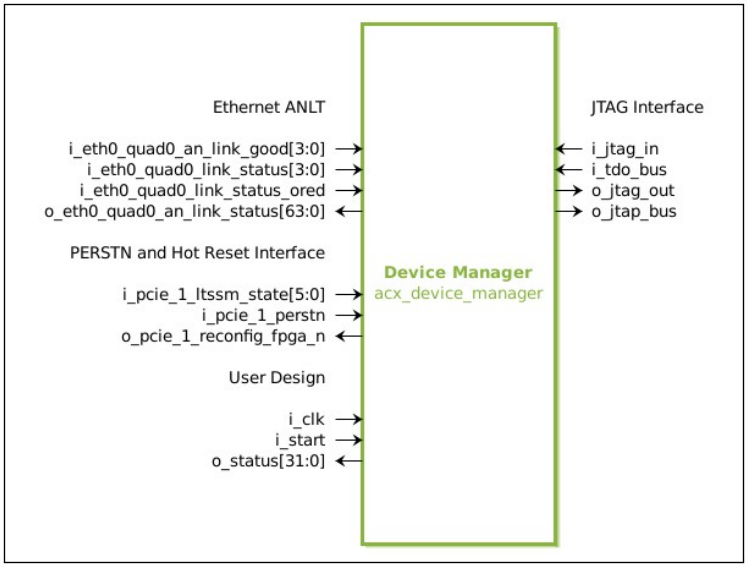

Figure 4 • The ACE IP Diagram of the Generated ADM Instance.

AC7t700/800 ADM System (DEV_MGR) Configuration

The ADM System configurator has the following options:

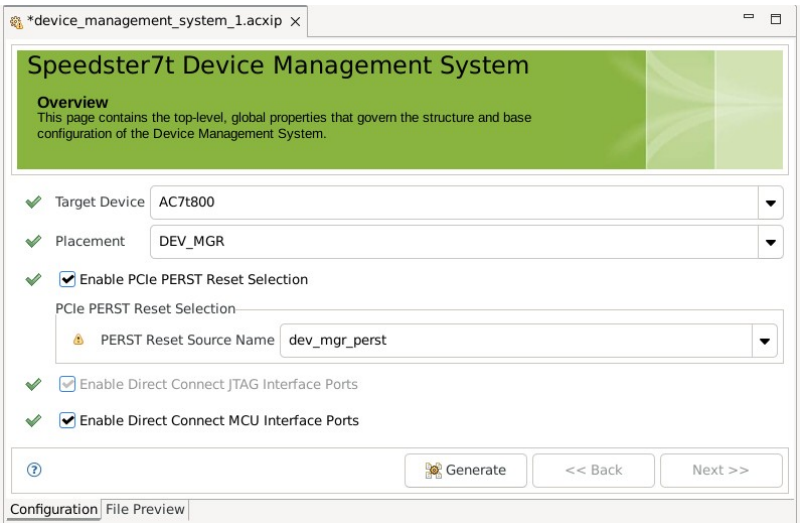

Figure 5 • Speedster7t ADM System Configuration

Table 4 • AC7t700/800 ADM System Configuration Options

Name	Default	Range	Description
Target Device	AC7t800	AC7t700/AC7t800	Set to match the target device of the project.
Placement	DEV_MGR	DEV_MGR	Location of hard ADM. Only one is available.
Enable PCIe PERST Reset Selection	disabled	disabled/enabled	When using PCIe, PERST Reset Selection must be enabled with a valid reset signal.
Enable Direct Connect JTAG Interface Ports	enabled	enabled	JTAP ports are always enabled. To be connected to the ADM gateway.
Enable Direct Connect MCU Interface Ports	disabled	disabled/enabled	Enable Direct Connect MCU Interface on the fabric boundary. Enables `o_status` port.

File Generation

The DEV_MGR configuration settings in the previous table prompt the generation the I/O ring in the selected location as shown in the following example:

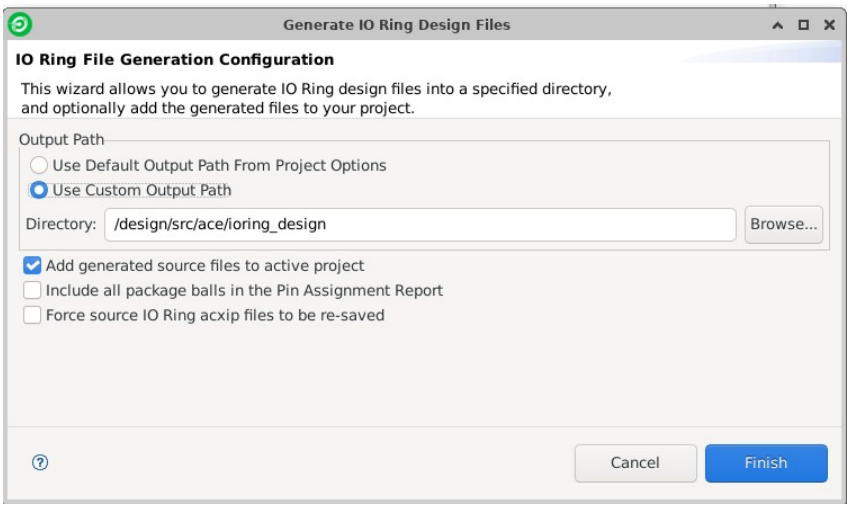

Figure 6 • I/O Ring Design File Generation

Ports

Table 5 • DEV_MGR Ports on the I/O Ring

Name	Direction	Description
<dev_mgr instance>_o_status[31:0]	Output	Ready and error status. Input into the user top level. See table below for status bits.
<dev_mgr instance>_jtag_tck	Input	JTAP clock to be connected to the ADM gateway o_jtag_bus_tck port.
<dev_mgr instance>_jtag_bus*	Input/Output	JTAP inputs and outputs. To be connected to the ADM gateway o_jtag_bus* ports

Status Signals

The o_status output signal shows the status after the startup phase is complete, and indicates if there were any errors during startup. Most designs need only monitor o_status[0] or o_status[1:0] to decide when the design can be started. The o_status value can also be read in ACE Tcl console with the mcu_status command. If errors occur, other ACE Tcl commands may give additional information.

Table 6 • ACX7t800 DEV_MGR Status Output

Status Bits	Meaning When Asserted
0	Startup is complete and successful. This signal can be used as the start signal for the user design.
1	Startup is complete. If there were no errors, `o_status[0]` is set as well. Otherwise, an error bit is also set.
3:2	ADM internal error. These bits should always read `2'b00` to indicate no error. If either bit is asserted, contact Achronix for assistance.
4	PCIe startup error
5	Reserved
6	DDR5 startup error
9:7	GDDR_2...GDDR_0 startup error
14:10	Reserved
15	ETH startup error
16	Reserved
17	Temperature alarm 0: Set when the FPGA temperature is ≥ 85°C
18	Temperature alarm 1: Set when the FPGA temperature is ≥ 100°C
31:19	Reserved

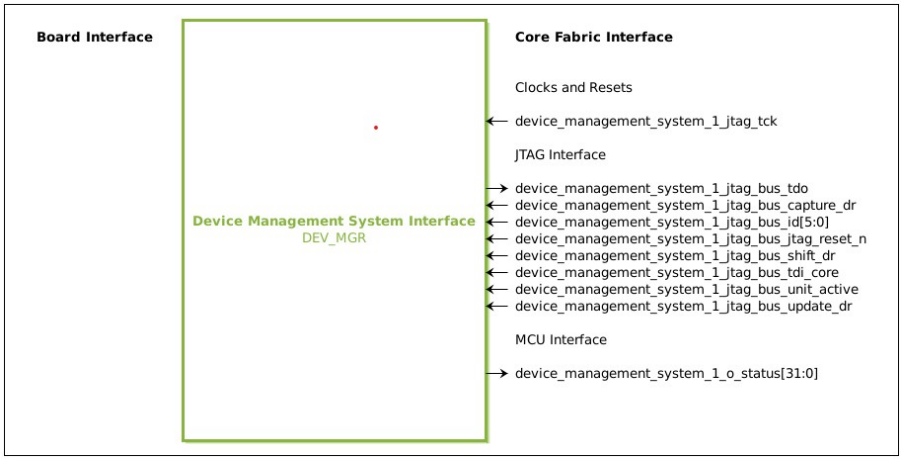

Figure 7 • ADM System IP Diagram

AC7t700/800 ADM Gateway Configuration

The ADM gateway (ACX_DEVICE_MANAGER_GATEWAY) is a soft IP component that can be instantiated on AC7t700 and AC7t800 devices to get access to the PCIe data bus interface (DBI), MCU SRAM for RPC functionality, or an I^2C master to initiate DDR5 RDIMM training procedures.

On AC7t1500 devices, these functionalities are part of the ADM soft IP component. On AC7t800 devices, the ADM is a hard I/O ring IP so the gateway needs to be instantiated to gain access to these features.

Configuration

The ADM gateway IP can be configured with the following options:

Figure 8 • ADM Gateway Soft IP Configuration

Table 7 • AC7t700/800 ADM Gateway Configuration Options

Parameter	Description
Target Device	Target device of the project
NAP Row(1)	Row coordinate for the ADM gateway's built-in NAP
NAP Column(1)	Column coordinate for the ADM gateway's built-in NAP
Enable PCIe DBI Gateway	Enables PCIe DBI gateway access
Enable DDR5 I^2C Master	Enables the I^2C master interface required for interfacing with DDR5 RDIMM components

Parameter	Description

Table Notes

1. The ADM gateway must have the NAP row and column parameters set in the RTL. This differs from most other IP blocks where it is recommended that the row and column location be set via the `.pdc` file. This requirement results from the fact that the ADM is encrypted. Thus, the path to the NAP instance in the RTL is also encrypted, making application of the row and column parameters difficult from a `.pdc` file.

Ports

Table 8 • ACX_DEVICE_MANAGER_GATEWAY Ports

Name	Direction	Description
`i_clk`	Input	Clock.
`i_rstn`	Input	Active-low reset.
`o_status`	Output	Gateway status indication. Used for debugging only. Can be left unconnected. The gateway is encrypted; the `o_status` signal is available to allow debugging of the module.
`t_JTAG_INPUT i_jtag_in`	Input	JTAG input: Should be connected to top-level ports with the same declaration.
`i_tdo_bus`	Input	Pass-through the JTAG bus to connect to Snapshot. Tie to 0 if unused.
`t_JTAG_OUTPUT o_jtag_out`	Output	JTAG output: Should be connected to top-level ports with the same declaration.
`t_JTAP_BUS o_jtap_bus`	Output	JTAP bus output, to share the JTAG interface.
`o_jtag_bus_jtag_reset_n`	Output	JTAP Interface. Must be connected at the top level to the direct connect JTAP interface ports from ADM system in the I/O ring.
`o_jtag_bus_tck`	Output	
`i_jtag_bus_tdo`	Input	
`o_jtag_bus_capture_dr`	Output	
`o_jtag_bus_id[5:0[`	Output	
`o_jtag_bus_shift_dr`	Output	
`o_jtag_bus_tdi_core`	Output	
`o_jtag_bus_unit_active`	Output	

Name	Direction	Description
o_jtag_bus_update_dr	Output	
io_i2c_sda	Inout	I2C Interface signals. Used if DDR5 I2C master is enabled.
o_i2c_sda_oe	Output	
o_i2c_scl	Output	

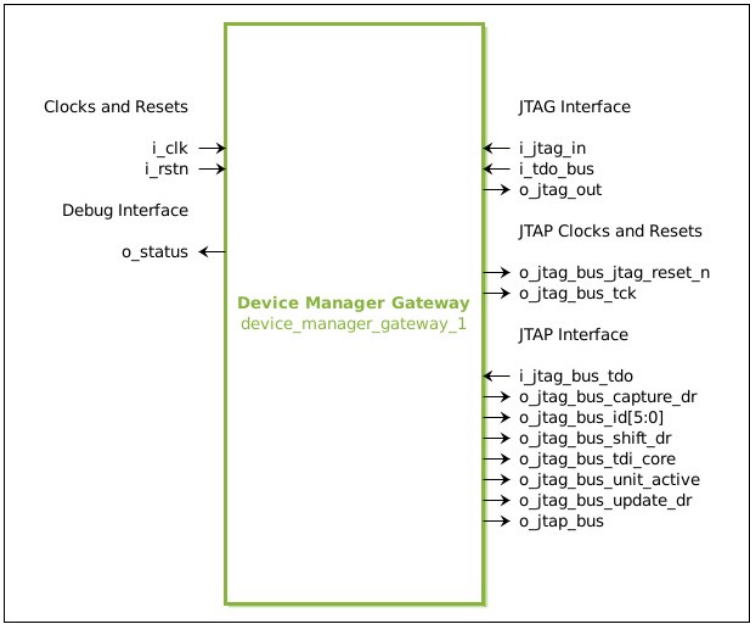

Figure 9 • Generated ADM Gateway Instance IP Diagram

Memory Map

The ADM gateway allows access to the following sub-blocks from the NoC address space.

Sub-Block	Address[27:0]	Description
PCIe DBI Gateway	[0x000_0000 : 0x03F_FFFF]	Access to PCIe controller's control and status registers.
I2C	[0x040_0000 : 0x04F_FFFF]	I2C initiator to interface with DDR5 RDIMM components.
ADM MCU SRAM Gateway	[0x080_0000 : 0x081_FFFF]	Access to MC SRAM for RPC functionality.

Chapter 3 : Using the JTAG Interface

JTAG Connection

The AC7t1500 ADM (ACX_DEVICE_MANAGER) and the AC7t800 ADM gateway (ACX_DEVICE_MANAGER_GATEWAY) connect to the JTAG interface via top-level ports in the design. These ports have an RTL type (i.e., `t_JTAG_INPUT`, `t_JTAG_OUTPUT`) that must be used in the RTL port and wire declarations. Declarations are illustrated in the **instantiation example** (page 0).

The placement of the JTAG interface is built into the macro, and no additional placement statements are required. ACE requires that the JTAG ports are top-level ports of the design. For this reason, it is most convenient to instantiate the ACX_DEVICE_MANAGER/ACX_DEVICE_MANAGER_GATEWAY at the top-level of the design.

In addition, for AC7t800 devices, the ACX_DEVICE_MANAGER_GATEWAY must also be connected to the I/O ring ADM JTAP interface.

The Speedster7t FPGA supports two JTAG communication models, depending on whether the design includes an ACX_DEVICE_MANAGER/ACX_DEVICE_MANAGER_GATEWAY instance. Both models are used in exactly the same manner, but the correct model must be selected with the `use_acx_device_manager` command defined in the following example. This command must be issued before programming the FPGA whenever a design has an ACX_DEVICE_MANAGER/ ACX_DEVICE_MANAGER_GATEWAY instance.

Sharing the JTAG Interface with Snapshot

The Snapshot debug tool, used to observe signals in a design, is independent of the ADM, but also uses the JTAG interface to interact with ACE. The AC7t1500 ACX_DEVICE_MANAGER and AC7t800 ACX_DEVICE_MANAGER_GATEWAY have two ports (`o_jtap_bus` and `i_tdo_bus`) that pass the JTAG signals through so that the interface can be shared. The ACX_SNAPSHOT_UNIT module has matching ports (`i_jtap_bus` and `o_tdo_bus`) that should be connected to the ADM.

> **Note**
>
> For more information on Snapshot, the real-time design debugging tool for Achronix FPGAs, see the ***Snapshot User Guide*** **(UG016)**[1].

1 https://achronix.com/documentation/snapshot-user-guide-ug016

Caution!

- When implementing Snapshot in a design that also requires the ADM, ACX_SNAPSHOT_JTAG_UNIT must be used rather than ACX_SNAPSHOT.
- The signal `o_jtap_bus` is not a simple wire, but instead, is type `t_JTAP_BUS`. This type must be used in the wire declaration. When connected in this manner, the Snapshot debug tool operates normally but with the caveat that follows in the next point (this caveat may change in future versions of ACE).
- To use the Snapshot debug tool, the ACE JTAG connection must be closed using the `<device_namespace>::close_jtag` Tcl command. This requirement is because Snapshot establishes its own connection to the JTAG driver in a different way, and the driver cannot have both connections open simultaneously. When a snapshot has been taken, the JTAG interface connected can be opened again with the `<device_namespace>::open_jtag` command, to allow use of Tcl commands via JTAG. The JTAG connection can be opened and closed repeatedly without affecting the running design.

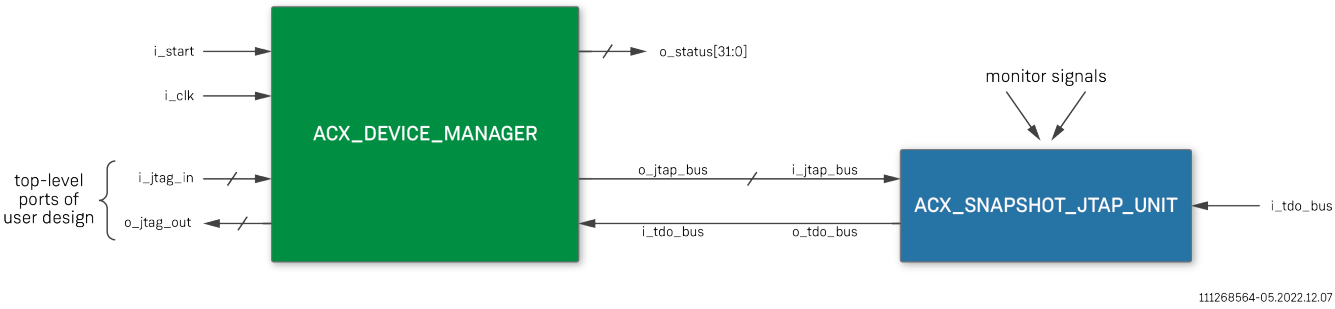

Figure 10 • JTAG Connection Shared Between AC7t1500 ACX_DEVICE_MANAGER and Snapshot

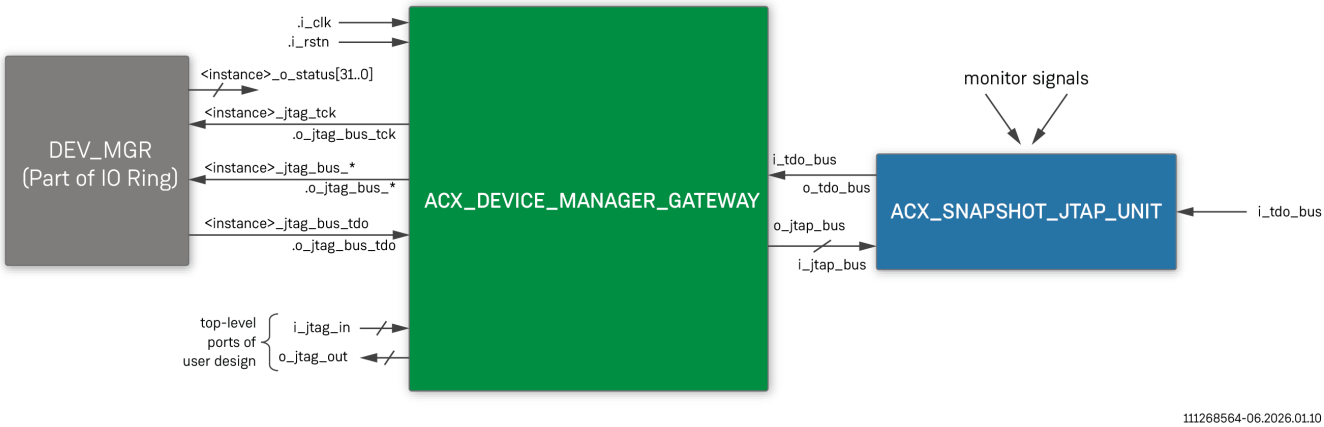

Figure 11 • JTAG Connection Shared Between AC7t800 ACX_DEVICE_MANAGER_GATEWAY and Snapshot

Accessing the ADM with ACE

ACE has several Tcl commands that can be used to access the ADM via the JTAG interface. To enable these commands, use the command `package require mcu`. Before these commands can be used, the JTAG connection must be established, and the device must be programmed. A typical sequence of instructions follows:

1. `use_device_namespace ac7t1500`
2. `use_acx_device_manager`
3. `set_jtag_id`
4. `open_jtag`
5. `program_hex_file <design>.hex`

Start using MCU commands, for instance::

- `package require mcu`
- `mcu_status`

These commands are described in the following table:

Table 9 • ADM JTAG Interface Tcl Command Details

Command	Description
`use_device_namespace ac7t1500`	Some commands (i.e., `program_hex_file`) are device-specific and are, therefore, in a special `ac7t1500::` namespace. Thus, the command would be called as: "`ac7t1500::program_hex_file`". As a convenience, the `use_device_namespace` command imports commands from the namespace, so that the namespace prefix can be omitted.
`use_acx_device_manager`	This command establishes that the ACX_DEVICE_MANAGER is used for JTAG communication and must be issued before programming the FPGA.
`set_jtag_id`	JTAG commands must have the name of the JTAG device driver. For most commands, this name is assumed to be in the global variable, `$jtag_id`. The `set_jtag_id` command finds the name of the device and sets the global variable. If there is more than one device (not common), it lists the possible choices. In such cases, or alternatively, the `jtag_id` variable can be assigned manually.
`open_jtag`	The `open_jtag` command establishes the connection with the JTAG driver. A `close_jtag` command is also provided.
`program_hex_file <design>.hex`	The `program_hex_file` command programs the FPGA via the JTAG interface. The full path name of the hex file must be specified (typically: `<userdesign>/impl_1/output/<design>.hex`).

Command	Description
`package require mcu`	The `package` command loads a set of Tcl macros into ACE. It must be issued at least once, but using it multiple times is harmless.
`mcu_status`	The `mcu_status` command displays the status of the ADM `o_status` output. Details are provided in the following section.

Chapter 4 : Achronix Device Manager Commands

The following ACE commands interact with the ACX_DEVICE_MANAGER. They are enabled with the `package require mcu` Tcl command.

> **Note**
> Unless otherwise specified, all arguments are optional.

use_acx_device_manager

Description

The `use_acx_device_manager` Tcl command specifies that the ACX_DEVICE_MANAGER handles JTAG communication with the 2D NoC. It must be specified at least once, before programming the FPGA. If this command is not used, the startup phase of the ACX_DEVICE_MANAGER does not complete (if JTAG is never used, as is typical in a production environment, this command is not required).

Example

```
use_acx_device_manager [-enable] [-disable]
```

Arguments

Table 10 • use_acx_device_manager Arguments

Argument	Description
`-enable`	Specifies that the ACX_DEVICE_MANAGER handles JTAG communication with the 2D NoC (default).
`-disable`	Specifies legacy mode, where the FCU handles JTAG communication with the 2D NoC. This argument should not be used with the ACX_DEVICE_MANAGER, because it interferes with the startup phase.

mcu_status

Description

The `mcu_status` Tcl command prints the value of the ACX_DEVICE_MANAGER `o_status` output, with interpretation.

Example

```
mcu_status [-wait] [-quiet] [-timeout <seconds>] [-debug]
```

Arguments

Table 11 • mcu_status Arguments

Argument	Description
`-wait`	Waits for the startup phase to complete.
`-quiet`	Instead of printing the result, this argument returns the status value (an integer).
`-timeout int`	Specifies a timeout (in seconds) for the `-wait` option (default is 50 seconds). If a timeout occurs, most likely the `use_acx_device_manager` command was not issued.
`-debug`	Prints additional information that might be useful when filing a support ticket.

mcu_info

Description

The `mcu_info` Tcl command prints ACX_DEVICE_MANAGER version information.

Example

```
mcu_info [-quiet]
```

Arguments

Table 12 • mcu_info Arguments

Argument	Description
`-quiet`	Instead of printing the result, this argument returns the information as a list.

noc_delay

Description

The `noc_delay` Tcl command prints the round-trip delay when accessing a CSR. This command confirms that the ACX_DEVICE_MANAGER can communicate with the 2D NoC. However, its main purpose is to print whether the CSR clock is the system clock or the JTAG clock. For the ACX_DEVICE_MANAGER to operate correctly, the system clock should be used. If the clock is reported as the JTAG clock, most likely the `use_acx_device_manager` command was omitted. The clock can be switched with the `set_csr_clock` Tcl command, but a `use_acx_device_manager` command must also be used.

Example

```
noc_delay
```

Arguments

This command has no arguments.

set_csr_clock

Description

The `set_csr_clock` Tcl command sets the clock used for CSRs. For the ACX_DEVICE_MANAGER to operate correctly, the CSR clock should be set to the system clock. When the `use_acx_device_manager` command is specified, the clock selection should be automatic. If the CSR clock remains set to the JTAG clock even after specifying the system clock, the board is misconfigured (the `FCU_CFG_CLKSEL` pin of the FPGA must be tied to 0). This condition must be corrected for the design to operate correctly.

Example

```
set_csr_clock [-system] [-jtag]
```

Arguments

Table 13 • set_csr_clock Arguments

Argument	Description
-system	Sets the system clock to the CSR clock (default).
-jtag	Sets the JTAG clock to the CSR clock.

fpga_temp

Description

The fpga_temp Tcl command prints the current temperature, as well as the maximum temperature measured so far. See FPGA Temperature (page 44) section for more details.

Example

```
fpga_temp [-clear] [-fout channelId] [-quiet]
```

Arguments

Table 14 • fpga_temp Arguments

Argument	Description
-clear	Resets the maximum temperature value, after returning the previous value.
-fout channelId	Print to the open channel (file descriptor) instead of to the console.

Argument	Description
-quiet	Instead of printing the result, return a list with items {max_temp, time_max_temp, current_temp, time_current_temp}, where: • Temperatures are in degrees C • Times are in seconds since the start of the design

fpga_temp_to_c

Description

The fpga_temp_to_c Tcl command converts the sample value to degrees C, returned as an integer. The sample argument is required. See FPGA Temperature (page 44) section for more details.

Example

```
fpga_temp_to_c <sample>
```

Arguments

Table 15 • fpga_temp_to_c Arguments

Argument	Description
sample	16-bit sample value.

fpga_c_to_temp

Description

The fpga_c_to_temp Tcl command converts an integer temperature in degrees C to a sample value range. The c argument is required. See FPGA Temperature (page 44) section for more details.

Example

```
fpga_c_to_temp <c> [-quiet]
```

Arguments

Table 16 • fpga_c_to_temp Arguments

Argument	Description
`c`	Temperature in degrees C.
`-quiet`	Instead of printing the result, returns a list with `{min, max}` sample values.

Chapter 5 : Remote Procedure Calls

The remote procedure call (RPC) mechanism is available on all Speedster7t FPGA devices to invoke functions executed on the embedded MCU in the Achronix device manager (ADM).

An RPC, at a high level, is a way for a program to execute a function on another computer. For Speedster7t devices, RPCs allow the user to execute functions running on the MCU in the ADM to perform tasks such as Ethernet link training or speed changes. RPCs simplify these tasks by removing the need for complex direct interactions with low-level interface IP control and status registers. On AC7t700 and AC7t1400 devices, where direct access to SerDes and PCS CSRs is not available, RPCs are the required mechanism to perform these tasks.

> **Note**
>
> On AC7t700 and AC7t1400 devices, direct access to SerDes and PCS registers in the CSR address space of the NoC is not supported. All tasks involving those register must be performed through RPC functions.

The RPC functions themselves are made available as firmware, which is pre-programmed in the read-only memory of the ADM MCU. That firmware cannot be modified or extended by the user.

Users make RPCs by sending messages to the ADM through a dedicated BRAM. That BRAM can be accessed via JTAG or NoC access points of either the ADM soft IP (AC7t1400/1500) or the ADM gateway soft IP (AC7t700/800). RPC functions can be invoked from three sources: JTAG, FPGA fabric logic, or PCIe interface

How to Invoke an RPC

RPCs are made through a shared BRAM accessible from both the NoC and the ADM MCU.

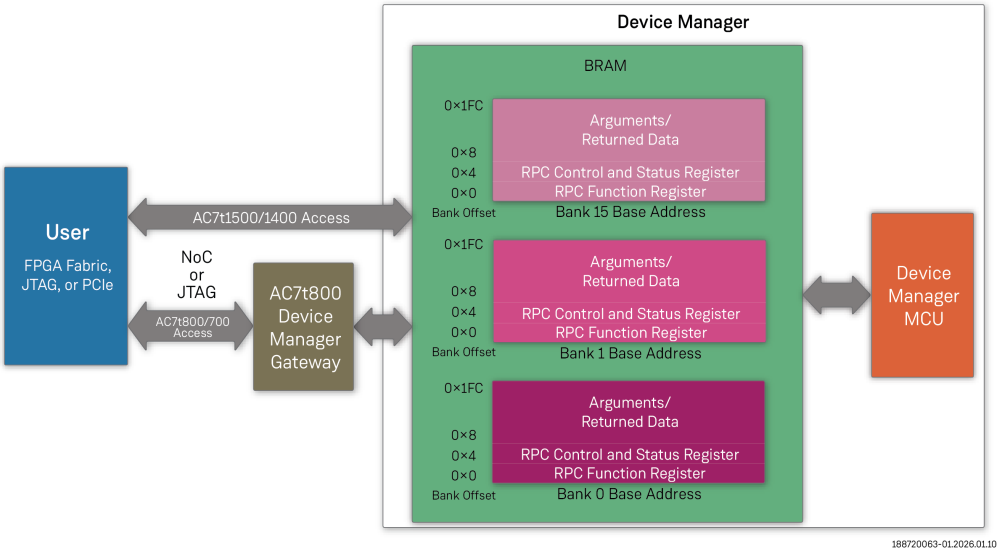

Figure 12 • RPC Access Points

BRAM Access Protocol

The BRAM is divided into 16 separate banks where each bank can contain control, arguments, and return data for one RPC function call. To make an RPC, the user will need to write the control and arguments for one of the defined commands (see list of RPC functions below) into a bank and then monitor the status of execution in the same BRAM bank. With 16 available banks, up to 16 commands can be queued at the same time.

To invoke an RPC, the following steps are taken:

1. The user writes the desired RPC function ID and arguments into the RPC function register (bank offset `0x0`) and arguments registers (bank offset `0x8` and `0xC`) of one of 16 separate RPC banks in the BRAM.

   > **Note**
   >
   > A new command can be written into the same bank only after the previous command for that bank has completed (either `error` or `done`)

2. The user writes `0x3` to the status and control bits of the RPC CSR (bank offset `0x4`) to indicate function has been queued and execution is being requested.
3. When the RPC request has been completed the ADM MCU will write the RPC status (bank offset `0x04`) along with returned data (bank offset `0x8`) to the same bank.
4. The user polls the RPC CSR to check the status of the RPC request.

   > **Note**
   >
   > Function is complete when status is set to either `error` or `done` at bank offset `0x4`.

5. Once the request is complete, the user can read the returned data, if appropriate, at bank offset `0x8`.

   > **Note**
   >
   > Currently supported functions all use `0x8` to return data, `0x08–0x1FC` may be used by future functions.

BRAM Access Notes

- RPCs can be made any time after the MCU is in its idle stage after the I/O ring components have been initialized.
- Control logic uses a round-robin priority algorithm to switch the BRAM access between the user and the ADM MCU running the firmware.
- The user can enter RPC commands into 15 of the 16 available RPC banks (bank #0 is reserved). The MCU arbitrates between the different banks in a round-robin fashion.
- Users are responsible for following the protocol outlined above.
- Uses are responsible for implementing their own BRAM bank allocation and arbitration policies to prevent collisions between concurrent users or processes.

RPC Bank Structure

RPC Bank Base Addresses

Each RPC register bank is `0x1000` words in length, and available at the following offset from the base address of the ADM soft IP (AC7t1400/1500) or the ADM gateway soft IP (AC7t700/800).

Table 17 • RPC Bank Base Addresses for the 16 RPC Banks

RPC Bank Number	RPC Bank Base Address
0 (†)	0x800000
1	0x801000
2	0x802000
3	0x803000
4	0x804000
5	0x805000
6	0x806000
7	0x807000
8	0x808000
9	0x809000
10	0x80A000
11	0x80B000
12	0x80C000
13	0x80D000
14	0x80E000
15	0x80F000

Table Notes

† Bank 0 is reserved for ACE; users should utilize any of the other Banks.

RPC Bank Register Offsets

Within an RPC bank, the user would read/write the following offset locations to provide argument or control for a particular RPC function.

Table 18 • Register Offsets Within an RPC Bank

Bank Address Offset	R/W	Function
RPC Function Register		
0x000	W	• [31:8] – Function arguments. Format vary per function. (†) (See the section, RPC Functions (page 27) below) • [7:0] – RPC function ID (See the section, RPC Functions (page 27) below)
RPC Control and Status Register		
0x004	R/W	• [31:2] – Return data. Reserved for future use. • [1:0] – RPC status and control: ◦ 00 – Error. RPC function has returned an error, or user has called an invalid RPC function. ◦ 01 – In Progress. RPC function actively running. ◦ 10 – Done. RPC function completed with no errors. ◦ 11 – Ready/Queued for Execution. RPC Function has been called but has not started. Written by user.
Extended Arguments/Return Data		
0x008 – 0x1FC	R/W	Usage dependent on the RPC function. **Caution!** Due to sharing of space, returned data may overwrite arguments when the function is run.

Table Notes

† [12:8] often used to select among the 32 SerDes lanes.

RPC Functions

This section describes the currently supported list of RPC function along with their associated arguments and return data. Arguments must be written to the appropriate location prior to queuing the function by writing to RPC CSR. If appropriate, return data is available once the RPC CSR shows done status. In certain cases, return data may overwrite arguments if those are at the same location offset, e.g., `0x008`.

> **Note**
>
> In the following tables, any bit locations with no specified values are "don't cares".

List of Current RPC Functions Available

General Functions

NoC Read

Function ID: `0x01`

Description: Read a 32-bit value from the provided NoC address.

Table 19 • NoC Read Arguments

Offset	Bits	Description
0x000	[7:0]	Function ID, 0x1
	[31:8]	NoC address[41:28]
0x008	[27:0]	NoC address[27:0]

Table 20 • NoC Read Return Data

Offset	Bits	Description
0x008	[31:0]	rdata[31:0]

NoC Write

Function ID: `0x02`

Description: Write a 32-bit value to the provided NoC address.

Table 21 • NoC Write Arguments

Offset	Bits	Description
0x000	[7:0]	Function ID, 0x2
	[31:8]	NoC address[41:28]
0x008	[27:0]	NoC address[27:0]
0x00C	[31:0]	wdata[31:0]

Ethernet SerDes Transmitter Functions

Get or Set TX FIR

Function ID: `0x03`

Description: Get or set the transmitter equalization (TX-FIR) coefficients.

Table 22 • Get or Set TX FIR Arguments

Offset	Bits	Description
0x000	[31]	• 1 – Set/write values • 0 – Get/read values
	[12:8]	SerDes Lane
	[7:0]	Function ID, 0x3
0x008 (†)	[20:16]	c1
	[15:10]	c0
	[9:6]	cm1
	[5:3]	cm2
	[2:0]	cm3

Note

† Data to be written when setting values.

Table 23 • Get or Set TX FIR Return Data

Offset	Bits	Description (†)
0x008	[20:16]	c1
	[15:10]	c0
	[9:6]	cm1
	[5:3]	cm2
	[2:0]	cm3

Note

† Data returned, when getting values..

Get or Set TX Polarity

Function ID: `0x04`

Description: Get or set the transmitter's P/N polarity setting.

Table 24 • Get or Set TX Polarity Arguments

Offset	Bits	Description
0x000	[31]	• 1 – Set/write values • 0 – Get/read values
	[30]	Transmitter polarity (used if [31] is set to '1'): • 0 – Normal • 1 – Inverted
	[12:8]	SerDes Lane
	[7:0]	Function ID, 0x4

Table 25 • Get or Set TX Polarity Return Data

Offset	Bits	Description (†)
0x008	[30]	Transmitter polarity: • 0 - Normal • 1 - Inverted

Note

† Data returned, when getting values..

Get or Set PAM4 Eye Ratio

Function ID: `0x05`

Description: Get or set the transmitter's PAM4 eye ratio setting.

Table 26 • Get or Set PAM4 Eye Ratio Arguments

Offset	Bits	Description
0x000	[31]	• 1 – Set/write values • 0 – Get/read values
	[30:24]	Top eye. (†)
	[23:16]	Mid eye. (†)
	[12:8]	SerDes lane
	[7:0]	Function ID, 0x5

Note

† Used if [31] is set to '1'

Return Data:

Table 27 • Get or Set PAM4 Eye Ratio Return Data

Offset	Bits	Description (†)
0x008	[30:24]	Top eye.
	[23:16]	Mid eye.

Note

† Data returned, when getting values..

Get or Set TX Gray Code

Function ID: 0x06

Description: Get or set the transmitter's gray code setting.

Table 28 • Get or Set TX Gray Code Arguments

Offset	Bits	Description
0x000	[31]	• 1 – Set/write values • 0 – Get/read values
	[30]	Use default. (†)
	[22:21]	el3 symbol. (†)
	[20:19]	el1 symbol. (†)
	[18:17]	eh1 symbol. (†)
	[16:15]	eh3 symbol. (†)
	[12:8]	SerDes lane
	[7:0]	Function ID, 0x6

Note

† Used if [31] is set to '1'

Table 29 • Get or Set TX Gray Code Return Data

Offset	Bits	Description (†)
0x008	[27:24]	el3 symbol
	[23:20]	el1 symbol
	[19:16]	eh1 symbol
	[15:12]	eh3 symbol
	[8]	Use default

Note

† Data returned, when getting values..

Get or Set TX Enable

Function ID: `0x07`

Description: Disable or enable the transmitter or to check if the transmitter is enabled or disabled.

Table 30 • Get or Set TX Enable Arguments

Offset	Bits	Description
0x000	[31]	• 1 – Set/write values • 0 – Get/read values
	[30]	• 1 – Enable transmitter (†) • 0 – Disable transmitter (†)
	[12:8]	SerDes lane
	[7:0]	Function ID, 0x7

Note

† Used if [31] is set to '1'

Table 31 • Get or Set TX Enable Return Data

Offset	Bits	Description (†)
0x008	[30]	• 1 – Enable transmitter • 0 – Disable transmitter

Note

† Data returned, when getting values.

Ethernet SerDes Receiver Functions

Get RX CDR Lock Status

Function ID: `0x08`

Description: Get receiver lane CDR lock status, after link training or RX equalization.

Table 32 • Get RX CDR Lock Status Arguments

Offset	Bits	Description
0x000	[12:8]	SerDes lane
	[7:0]	Function ID, 0x8

Table 33 • Get RX CDR Lock Status Return Data

Offset	Bits	Description
0x0008	[30]	• 1 – CDR is locked • 1 – CDR is unlocked

Get or Set RX Polarity

Function ID: `0x09`

Description: Get or set receiver polarity for a SerDes lane.

Table 34 • Get or Set RX Polarity for a SerDes Lane Arguments

Offset	Bits	Description
0x000	[31]	• 1 – Set/write values • 0 – Get/read values
	[30]	Receiver polarity: (†) • 0 - Normal • 1 - Inverted
	[12:8]	SerDes lane
	[7:0]	Function ID, 0x9

Note

† Used if [31] is set to '1'

Table 35 • Get or Set RX Polarity for a SerDes Lane Return Data

Offset	Bits	Description (†)
0x008	[30]	Receiver polarity: • 0 - Normal • 1 - Inverted

Note

† Data returned, when getting values..

Do RX Equalization

Function ID: `0x0A`

Description: Perform receiver equalization.

Table 36 • Do RX Equalization Arguments

Offset	Bits	Description
0x000	[30]	Equalization type: • 0 – Runs full equalization mode, i.e,. it runs all the complete equalization from start to finish. This is the most common case. • 1 – Runs evaluation equalization mode, i.e., it runs the minimum equalization required to evaluate link partner's TX-FIR settings during link training. This mode is normally run iteratively.
	[23:16]	Specifies number of iterations used by the CDR lock polling routine.
	[12:8]	SerDes lane
	[7:0]	Function ID, 0xA

Request PMA State Toggle

Function ID: `0x0B`

Description: This procedure requests the PCS to clear set of PMA flags there by unblocking the PMA of running RX-EQ/LT processes. Running this function is a prerequisite for receiver equalization (RX-EQ), and link training (LT).

Table 37 • Request PMA State Toggle Arguments

Offset	Bits	Description
0x000	[12:8]	SerDes Lane
	[7:0]	Function ID, 0xB

Trigger Link Training

Function ID: `0x0C`

Description: Triggers link training on the selected lane and immediately returns control to command line (non blocking). Users must call `Get Link Training Status` to monitor if LT had completed. This procedure can be in a loop to trigger LT on multiple SerDes lanes.

Table 38 • Trigger Link Training Arguments

Offset	Bits	Description
0x000	[7:0]	Function ID, 0xC
	[12:8]	SerDes lane

Get Link Training Status

Function ID: `0x0D`

Description: Returns success/failure status of link training (LT) process. This procedure should be called repeatedly as LT sometimes takes time to complete.

Table 39 • Get Link Training Status Arguments

Offset	Bits	Description
0x000	[23:16]	Specifies number of iterations used by the CDR lock polling routine.
	[12:8]	SerDes lane
	[7:0]	Function ID, 0xD

Table 40 • Get Link Training Status Return Data

Offset	Bits	Description
0x008	[30]	• 1 – Link training successful • 0 – Link training unsuccessful

Switch SerDes Rate

Function ID: `0x0E`

Description: To switch SerDes operating rate to a new rate stored in the SRAM.

Table 41 • Switch SerDes Rate Arguments

Offset	Bits	Description
0x000	[31:24]	Specifies the number of iterations. The iterations parameter is proportional to the polling period for acknowledge of the rate-change operation.
	[23:20]	Specifies the width of the data bus at SerDes/fabric interface. This value is also selected when creating raw SerDes IP. Allowed data widths and mapping: • 0010 – 16 bits • 0011 – 20 bits • 0100 – 32 bits • 0101 – 40 bits • 0110 – 64 bits • 0111 – 128 bits
	[19:16]	Rate ID – specifies the rate preset corresponding to the new rate. Users can configure up to eight presets during raw SerDes ACXIP creation. In case of multiple presets are present, the SerDes lane will be powered up using rate-0 after FPGA enters user mode.
	[12:8]	SerDes lane
	[7:0]	Function ID, 0xE

Table 42 • Switch SerDes Rate Return Data

Offset	Bits	Description
0x008	[30]	• 1 – Rate setting successful • 0 – Rate setting unsuccessful

Set Loopback

Function ID: `0x0F`

Description: Sets the loopback mode for a SerDes lane.

Table 43 • Set Loopback Arguments

Offset	Bits	Description
0x000	[31:30]	Mode: • 1 – Enable near-end serial (NES) loopback • 2 – Enable far-end serial (FES) loopback • 3 – Enable near-end parallel (NEP) loopback
	[12:8]	SerDes lane
	[7:0]	Function ID, 0xF

Examples

Using JTAG Tcl Commands

The following example RPC function call use JTAG Tcl commands to access the RPC memory space via the NoC. For this example, the ADM or ADM gateway is assumed to be placed at column 6. row 4, and the call is using RPC bank 1 (`0x801000` from **table** (page 25) above).

For more information on JTAG Tcl commands, refer to section, "JTAG Programming using the Tcl Library API" of the ***Speedster7t Configuration User Guide*** (UG094)[2].

2 https://achronix.com/documentation/speedster7t-configuration-user-guide-ug094

Retrieve Transmitter FIR Coefficient Example.

The following example uses the RPC, **Get or Set TX FIR** (page 28) (ID `0x03`), to get the current TX equalization settings. The function is executed in 4 steps using the Tcl commands, **nap_axi_write** and **nap_axi_read**, with the following syntax:

```
nap_axi_write NAP_SPACE <column> <row> <address> <value>
nap_axi_read NAP_SPACE <column> <row> <address>
```

Where for this example:

- Column = 6
- Row = 4
- Address = <RPC bank address offset> & <offset of the register>
- RPC bank address offset = 0x801000
- Value = <RPC arguments> or <RPC Control>

Step 1. Set the Control Register with Function Arguments

This command sets up the control register (offset 0x000) for the RPC call to reads the transmitter equalization (ID 0x03) settings for SerDes lane 12 (0x0C).

```
ac7t1500::nap_axi_write NAP_SPACE 6 4 801000 00000C03;
```

Step 2. Set the Status Bits of the CSR

Set the status bits of the CSR (offset `0x004` from the **table** (page 26) above) to "ready" (0x03) to indicate that the arguments are set and the function can be executed.

```
ac7t1500::nap_axi_write NAP_SPACE 6 4 801004 3;
```

Step 3. Poll the Status Bits of the CSR

Next, continue to query the status bits of the CSR (offset 004) until the status is set to "done" (value of 2), signaling the RPC function has finished executing.

```
ac7t1500::nap_axi_read NAP_SPACE 6 4 801004;
```

Step 4. Read the Results

Read the transmitter coefficients from the returned data register (0x008). See the table under **Get or Set TX FIR** (page 28) for bit encoding of the coefficients:

```
ac7t1500::nap_axi_read NAP_SPACE 6 4 801008;
```

Using Achronix SDK

RPCs can be made from the host across PCIe using C functions available in the **Achronix Software Development Kit**[3] (Version 2.2 and newer) through the **Demonstration Design for the Achronix SDK**[4] (Version 1.3 and newer) loaded on a VectorPath board. See ***Software Development Kit User Guide* (UG107)**[5] for more details.

> **Note**
>
> Support account is needed to access the Achronix SDK. See the KB article, **How Do I Register for an Achronix Support Account?**[6] for details.

Supported Functions

Here are currently supported functions defined in <Achronix SDK Install Dir>/include/Achronix_IP/Achronix_RPC.h and implemented in the SDK to abstract away the low-level RPC call mechanism.

```
// Initializes the ACX_IP_RPC ip_block in the device
uint32_t acx_rpc_init(ACX_DEV_PCIe_device *device, ACX_IP_block *ip_block, void
 *details);

// Cleans up the ACX_IP_RPC ip_block in the device
void acx_rpc_cleanup(ACX_DEV_PCIe_device *device, ACX_IP_block *ip_block);

// Runtime override for the BAR index and offset for the RPC interface
ACX_SDK_STATUS acx_rpc_overide_location(ACX_DEV_PCIe_device *device, int bar_index,
uint32_t bar_offset);

// Set the RPC bank to use
ACX_SDK_STATUS acx_rpc_bank_select(int bank);

// Write a 32-bit value to a NOC location using the ADM RPC
ACX_SDK_STATUS acx_rpc_noc_write(ACX_DEV_PCIe_device *device, uint64_t noc_address,
uint32_t value);

// Read a 32-bit value from a NOC location using the ADM RPC
ACX_SDK_STATUS acx_rpc_noc_read(ACX_DEV_PCIe_device *device, uint64_t noc_address,
uint32_t *value_p);

// Retrieve the TX FIR coefficients for a serdes lane
ACX_SDK_STATUS acx_rpc_get_tx_fir(ACX_DEV_PCIe_device *device, int lane,
ACX_RPC_FIR_COEFFICIENTS *fir_up);
```

3 https://support.achronix.com/hc/en-us/articles/15234194799252-How-do-I-Download-the-Achronix-SDK
4 https://support.achronix.com/hc/en-us/articles/38065028969236-How-do-I-Download-Demonstration-and-Reference-Designs
5 https://www.achronix.com/documentation/software-development-kit-user-guide-ug107
6 https://support.achronix.com/hc/en-us/articles/4404014677268-How-Do-I-Register-for-an-Achronix-Support-Account

```
// Set the TX FIR coefficients for a serdes lane
ACX_SDK_STATUS acx_rpc_set_tx_fir(ACX_DEV_PCIe_device *device, int lane,
ACX_RPC_FIR_COEFFICIENTS fir_u);

// Retrieve TX polarity for a serdes lane
ACX_SDK_STATUS acx_rpc_get_tx_polarity(ACX_DEV_PCIe_device *device, int lane, int
 *polarity_p);

//Set TX polarity for a serdes lane
ACX_SDK_STATUS acx_rpc_set_tx_polarity(ACX_DEV_PCIe_device *device, int lane, int
 polarity);

//Retrieve TX PAM4 eye ratio coefficients for a serdes lane
ACX_SDK_STATUS acx_rpc_get_tx_pam4_eye_ratio(ACX_DEV_PCIe_device *device, int lane,
ACX_RPC_EYE_RATIO *ratio_up);

// Set TX PAM4 eye ratio coefficients for a serdes lane
ACX_SDK_STATUS acx_rpc_set_tx_pam4_eye_ratio(ACX_DEV_PCIe_device *device, int lane,
ACX_RPC_EYE_RATIO ratio_u);

// Retrieve TX graycode coefficients for a serdes lane
ACX_SDK_STATUS acx_rpc_get_tx_graycode(ACX_DEV_PCIe_device *device, int lane,
ACX_RPC_GRAYCODE *graycode_up);

// Set TX graycode coefficients for a serdes lane
ACX_SDK_STATUS acx_rpc_set_tx_graycode(ACX_DEV_PCIe_device *device, int lane,
ACX_RPC_GRAYCODE graycode_u);

// Retrieve TX lane enable state
ACX_SDK_STATUS acx_rpc_get_tx_enable(ACX_DEV_PCIe_device *device, int lane, int
 *enable_p);

// Set TX lane enable state
ACX_SDK_STATUS acx_rpc_set_tx_enable(ACX_DEV_PCIe_device *device, int lane, int enable);

// Get RX lane CDR lock status
ACX_SDK_STATUS acx_rpc_get_cdr_lock_status(ACX_DEV_PCIe_device *device, int lane, int
 *status_p);

// Retrieve RX polarity for a serdes lane
ACX_SDK_STATUS acx_rpc_get_rx_polarity(ACX_DEV_PCIe_device *device, int lane, int
 *polarity_p);

// Set RX polarity for a serdes lane
ACX_SDK_STATUS acx_rpc_set_rx_polarity(ACX_DEV_PCIe_device *device, int lane, int
 polarity);

// Perform RX equalization for a serdes lane
```

```
ACX_SDK_STATUS acx_rpc_do_rx_equalization(ACX_DEV_PCIe_device *device, int lane, int
 eval_eq, int iterations);

// Perform RX PMA state toggle for a serdes lane
ACX_SDK_STATUS acx_rpc_do_rx_pma_state_toggle(ACX_DEV_PCIe_device *device, int lane);

// Initiate RX/TX link training for a serdes lane
ACX_SDK_STATUS acx_rpc_do_rxtx_training(ACX_DEV_PCIe_device *device, int lane);

// Retrieve RX/TX link training status for a serdes lane
ACX_SDK_STATUS acx_rpc_get_get_rxtx_training_status(ACX_DEV_PCIe_device *device, int
 lane, int iterations, int *status_p);

// Change RX/TX data rate for a raw serdes lane</font></pre>
ACX_SDK_STATUS acx_rpc_set_serdes_rate(ACX_DEV_PCIe_device *device, int lane, int
 iterations, ACX_RPC_DBUS_WIDTH width, int rate);

// Set loopback mode for a serdes lane
ACX_SDK_STATUS acx_rpc_set_serdes_loopback(ACX_DEV_PCIe_device *device, int lane,
ACX_RPC_LOOPBACK_MODE mode);
```

Example

Example of RPC usage is shown in `<Achronix SDK Install Dir>/example/Serdes_example/serdes_example.c`. By default, the code is using Bank 1.

Below is a short code snippet showing how Ethernet FIR coefficients are set.

```
// Open the PCIe device with device id or bus-device-function.
ACX_DEV_PCIe_device *device = NULL;
if (id >= 0)
    device = acx_dev_init_pcie_device_idx(id);
else if (bdf)
    device = acx_dev_init_pcie_device_bdf(bdf);

// Derive lane start and limit based on if using Ethernet Port 0 and Port 1
int lane_start = port ? 8 : 0;
int lane_limit = lane_start + 4;
int lane;
ACX_SDK_STATUS rv;

ACX_RPC_FIR_COEFFICIENTS fir;
fir.c1 = c1;
fir.c0 = c0;
fir.cm1 = cm1;
fir.cm2 = cm2;
fir.cm3 = cm3;

// Set coefficients for each lane and check for status
```

```
for (lane = lane_start; lane < lane_limit; lane++) {
    rv = acx_rpc_set_tx_fir(device, lane, fir);
    if (rv != ACX_SDK_STATUS_OK) {
        printf("ERROR: acx_rpc_set_tx_fir(lane:%2u) failed %s\n", lane,
acx_sdk_status_to_string(rv));
        return 1;
    }
}
```

Chapter 6 : FPGA Temperature

At startup, the ADM configures the FPGA built-in temperature sensor for continuous sampling. The ADM status output has two bits that are set high when pre-set temperature limits are exceeded. These can be observed by the design, or brought out via GPIO pins to throttle the design or possibly shut down the board (the ADM merely observes the temperature; it does not shut down the board or FPGA).

Table 44 • ADM Temperature Status Outputs

Bit	Description
`o_status[17]`	Temperature alarm 0 – high when the FPGA temperature is ≥ 85°C.
`o_status[18]`	Temperature alarm 1 – high when the FPGA temperature is ≥ 100°C.

When a JTAG connection is present, the `fpga_temp` Tcl command can be used.

The current temperature can also be observed directly by reading the CSR, `0x080c00000d8` on AC7t1500 devices and `0x080a0000138` on AC7t800 devices. CSRs can be read with Tcl via JTAG:

```
ac7t1500::noc_read 080c00000d8, ac7t800::noc_read 080a0000138;
```

The register can also be read through the 2D NoC via a NAP. This way, a design can implement its own temperature alarms.

The return value of the CSR is a 16-bit sample value. It can be converted to degrees C with the Tcl command `fpga_temp_to_c`, or by computing:

$$a + b \times (sample/4096 - 0.5)$$

where a = 71.4 and b = 283.

The reverse calculation is performed by the `fpga_c_to_temp` Tcl command, to facilitate comparison of sample values.

Chapter 7 : Simulation

Device simulation model support for the ACX_DEVICE_MANAGER soft IP is available in the full-chip RTL flow. During simulation, the ADM waits 16 cycles then sets `o_status` to `0x3` to indicate successful completion of the startup phase. This value is appropriate for most simulations, which typically do not include full simulation models for all of the interfaces and external devices.

For more details, seethe ***Simulation User Guide*** (UG072)[7].

7 https://achronix.com/documentation/simulation-user-guide-ug072

Chapter 8 : Instantiation Example

This section covers the instantiation of the ADM for AC7t1400/AC7t1500 devices as well as the ADM gateway for AC7t700/AC7t800 devices.

AC7t1400/AC7t1500 Instantiation

AC7t1400/AC7t1500 Example Template

The following example shows the ACE-generated ADM template.

```
`include "speedster7t/macros/ACX_DEVICE_MANAGER.svp"

module device_manager_test
(
   // JTAG Interface
   input  t_JTAG_INPUT  i_jtag_in,      // Should be connected to top-level ports with
the same declaration
   input                i_tdo_bus,      // Pass-through the JTAG bus to connect to
Snapshot. If not used, this input should be tied to 1'b0
   output t_JTAG_OUTPUT o_jtag_out,     // Should be connected to top-level ports with
the same declaration
   output t_JTAP_BUS    o_jtap_bus,     // Pass-through of the JTAG bus to connect to
Snapshot (or other JTAG components)

   // User Design
   input                i_clk,          // 100 MHz Clock input for Device Manager block.
   input                i_start,        // A high input starts the Device Manager. In
most cases this signal is tied to 1'b1,
                                        // but it can also be tied to a PLL lock signal
if necessary.
   output [31:0]        o_status        // Progress indication, error status, alarms
);

   wire [1023:0] not_used;

  ACX_DEVICE_MANAGER #
  (
  .NAP_ROW      (6),            // NAP Row
  .NAP_COLUMN   (3)             // NAP Column
  )
  x_dev_mgr
  (
```

```
   // JTAG Interface
   .i_jtag_in   (i_jtag_in),    // Should be connected to top-level ports with the same
declaration
   .i_tdo_bus   (i_tdo_bus),    // Pass-through the JTAG bus to connect to Snapshot. If
not used, this input should be tied to 1'b0
   .o_jtag_out  (o_jtag_out),   // Should be connected to top-level ports with the same
declaration
   .o_jtap_bus  (o_jtap_bus),   // Pass-through of the JTAG bus to connect to Snapshot
(or other JTAG components)

   // User Design

   .i_clk       (i_clk),        // 100 MHz Clock input for Device Manager block.
   .i_start     (i_start),      // A high input starts the Device Manager. In most cases
this signal is tied to 1'b1,
                                // but it can also be tied to a PLL lock signal if
necessary.
   .o_status    (o_status)      // Progress indication, error status, alarms
  );

endmodule : device_manager_test
```

AC7t1400/AC7t1500 Instantiation Without Snapshot

The following examples show how the ACE-generated ADM template can be utilized in a design.

```
module top_level
(
   // JTAG Interface
   input  t_JTAG_INPUT   i_jtag_in,     // Should be connected to top-level ports with
the same declaration
   output t_JTAG_OUTPUT  o_jtag_out,    // Should be connected to top-level ports with
the same declaration

   // User Design
   input  wire           i_clk          // 100 MHz Clock input for Device Manager block.
);

   // signals for shared JTAG bus
   wire  t_JTAP_BUS      jtap_bus;      // shared JTAG bus
   wire                  tdo_bus;       // tie to 1'b0 if unused

   // Other ADM signals
   logic [32 -1:0]       adm_status;    // Status from the ADM

   device_manager_test # ()
   i_acx_device_manager
   (
```

```
        // JTAG Interface
       .i_jtag_in   (i_jtag_in),        // Should be connected to top-level ports with
the same declaration
       .i_tdo_bus   (tdo_bus),          // Pass-through the JTAG bus to connect to
Snapshot. If not used, this input should be tied to 1'b0
       .o_jtag_out  (o_jtag_out),       // Should be connected to top-level ports with
the same declaration
       .o_jtap_bus  (jtap_bus),         // Pass-through of the JTAG bus to connect to
Snapshot (or other JTAG components)

       // Ethernet0 Quad0 Interface
       .i_eth0_quad0_an_link_good ({ethernet_0_m0_an_link_good_ln3,
ethernet_0_m0_an_link_good_ln2, ethernet_0_m0_an_link_good_ln1,
ethernet_0_m0_an_link_good_ln0}),
       .i_eth0_quad0_link_status (ethernet_0_quad0_link_status),
       .i_eth0_quad0_link_status_ored (ethernet_0_quad0_link_status_ored),
       .o_eth0_quad0_an_link_status ({ethernet_0_m0_an_link_status_ln3,
ethernet_0_m0_an_link_status_ln2, ethernet_0_m0_an_link_status_ln1,
ethernet_0_m0_an_link_status_ln0}),

       // Ethernet1 Quad0 Interface
       .i_eth1_quad0_an_link_good ({ethernet_1_m0_an_link_good_ln3,
ethernet_1_m0_an_link_good_ln2, ethernet_1_m0_an_link_good_ln1,
ethernet_1_m0_an_link_good_ln0}),
       .i_eth1_quad0_link_status (ethernet_1_quad0_link_status),
       .i_eth1_quad0_link_status_ored (ethernet_1_quad0_link_status_ored),
       .o_eth1_quad0_an_link_status ({ethernet_1_m0_an_link_status_ln3,
ethernet_1_m0_an_link_status_ln2, ethernet_1_m0_an_link_status_ln1,
ethernet_1_m0_an_link_status_ln0}),

       // User Design
       .i_clk       (i_clk),            // 100 MHz Clock input for Device Manager block.
       .i_start     (1'b1),             // A high input starts the Device Manager. In
most cases this signal is tied to 1'b1,
                                        // but it can also be tied to a PLL lock signal
if necessary.
       .o_status    (adm_status)        // Progress indication, error status, alarms
   );

endmodule : top_level
```

AC7t1400/AC7t1500 Instantiation with Snapshot

```
`include "speedster7t/common/speedster7t_snapshot_v3.sv"

module top_level
(
   // JTAG Interface
   input  t_JTAG_INPUT   i_jtag_in,      // Should be connected to top-level ports with
the same declaration
   output t_JTAG_OUTPUT  o_jtag_out,     // Should be connected to top-level ports with
the same declaration

   // User Design
   input                 i_clk           // 100 MHz Clock input for Device Manager
block.
);

   // signals for shared JTAG bus
   wire   t_JTAP_BUS     jtap_bus;       // shared JTAG bus
   wire                  tdo_bus;        // tie to 0 if unused

   // Other ADM signals
   logic [32 -1:0]       adm_status;     // Status from the ADM

   device_manager_test # ()
   i_acx_device_manager
   (
        // JTAG Interface
        .i_jtag_in   (i_jtag_in),        // Should be connected to top-level ports with
the same declaration
        .i_tdo_bus   (tdo_bus),          // Pass-through the JTAG bus to connect to
Snapshot. If not used, this input should be tied to 1'b0
        .o_jtag_out  (o_jtag_out),       // Should be connected to top-level ports with
the same declaration
        .o_jtap_bus  (jtap_bus),         // Pass-through of the JTAG bus to connect to
Snapshot (or other JTAG components)

        // User Design
        .i_clk       (i_clk),            // 100 MHz Clock input for Device Manager
block.
        .i_start     (1'b1),             // A high input starts the Device Manager. In
most cases this signal is tied to 1'b1,
                                         // but it can also be tied to a PLL lock signal
if necessary.
        .o_status    (adm_status)        // Progress indication, error status, alarms
   );

   ACX_SNAPSHOT_UNIT #(....)
```

```
    x_snapshot
    (
         .i_jtap_bus  (jtap_bus),
         .i_tdo_bus   (1'b0),             // Tie to 1'b0 if not used
         .o_tdo_bus   (tdo_bus),
         ... other Snapshot ports ...
    );

endmodule : top_level
```

AC7t800 / AC7t700 Instantiation

For AC7t800 devices, the ADM is part of the I/O ring and is included when the I/O ring is generated. The ADM gateway, however, must be instantiated and connected to the top level, including to the ADM system as in the example below.

```
device_manager_gateway #()
u_adm_gateway
(
    .i_clk (i_nap_clk),
    .i_rstn (reg_rstn),

    //Top-level JTAG interface connections
    .i_jtag_in (i_jtag_in),                      // Should be connected to top-level
ports with the same declaration
    .o_jtag_out (o_jtag_out),                    // Should be connected to top-level
ports with the same declaration

    // JTAG connections to/from FPGA core
    .i_tdo_bus (1'b0),                           // Pass-through the JTAG bus to
connect to Snapshot. tie to 0 if not used

    //JTAP Interface must be connected to Direct Connect JTAG Interface ports from
Device Manager System in the IO Ring.
    .i_jtag_bus_tdo (device_management_system_1_jtag_bus_tdo),
    .o_jtag_bus_tck (device_management_system_1_jtag_tck),
    .o_jtag_bus_capture_dr (device_management_system_1_jtag_bus_capture_dr),
    .o_jtag_bus_id (device_management_system_1_jtag_bus_id),
    .o_jtag_bus_jtag_reset_n (device_management_system_1_jtag_bus_jtag_reset_n),
    .o_jtag_bus_shift_dr (device_management_system_1_jtag_bus_shift_dr),
    .o_jtag_bus_tdi_core (device_management_system_1_jtag_bus_tdi_core),
    .o_jtag_bus_unit_active (device_management_system_1_jtag_bus_unit_active),
    .o_jtag_bus_update_dr (device_management_system_1_jtag_bus_update_dr)

    //.o_status    (adm_status)        // Progress indication, error status, alarms

    // DDR5 I2C Master Interface
    //.i_i2c_sda_in (1'b0),
```

```
    //.o_i2c_sda_oe (),
    //.o_i2c_sda_out (),
    //.o_i2c_scl_oe  (),
    //.o_i2c_scl  ()
);
```

Chapter 9 : Revision History

Version	Date	Description
1.0	22 Jan 2026	• Initial Achronix release.