JTAG to AXI Master Core Demo Script

Introduction

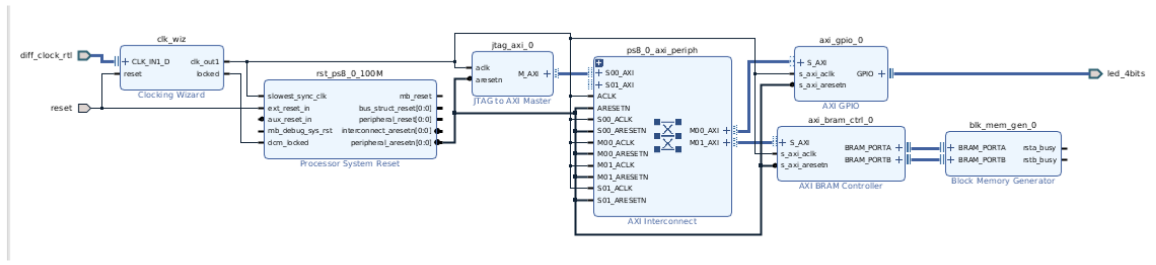
This demonstration introduces the JTAG to AXI debug core that can generate AXI transactions and drive internal AXI signals in a design at run time. The demonstration also highlights the Tcl commands used at runtime interaction with this core in the Vivado™ hardware manager environment.

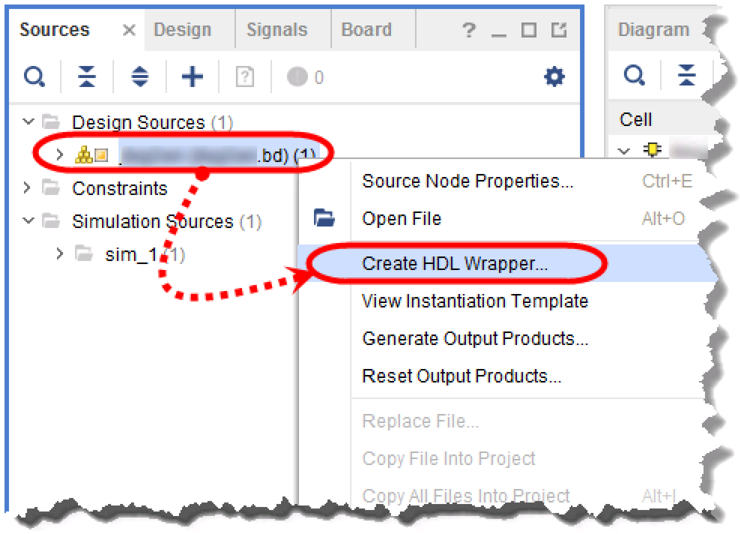
Preparation:

- Necessary files are located in the following directory:
  `$TRAINING_PATH/jtag2axi/demo/ZCU104`
- Required hardware: ZCU104 Evaluation Platform
- Required software: Vivado Design Suite (any edition)

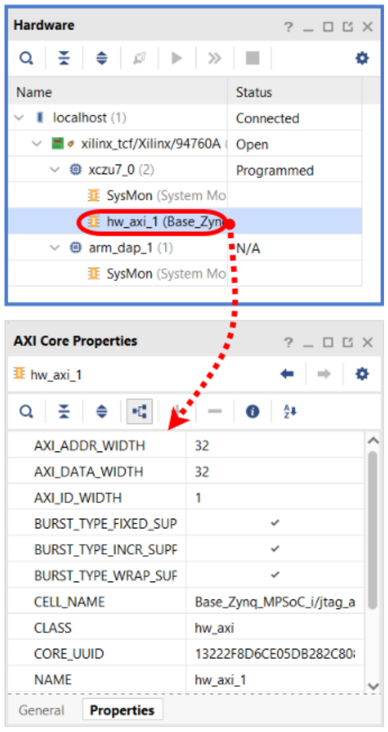
JTAG to AXI Master Debug IP Transactions

Action with Description	Point of Emphasis and Key Takeaway
• Launch the Vivado Design Suite.	
• Unzip the project using the Tcl Console: `exec unzip $::env(TRAINING_PATH)/jtag2axi/demo/ZCU104.zip -d $::env(TRAINING_PATH)/jtag2axi/demo/ZCU104` • Open the project **`jtag2axi.xpr`** from the following directory using the Vivado IDE: `$TRAINING_PATH/jtag2axi/demo/ZCU104`	• The Open Project selection allows the designer to access existing projects.
• Notice that the design is empty. • You will create an IPI block design that has two peripherals: block RAM and GPIO (which connects to the LEDs on the board).	There are several ways to create a block design: • One way is to use a Tcl script to generate the required block design.

Action with Description	Point of Emphasis and Key Takeaway
• Enter the following command in the Tcl Console of the Vivado IDE to generate the required block design for this demonstration: `source /home/amd/training/jtag2axi/demo/ZCU104/Create_jtag2axi_block-design.tcl` • Right-click and select **Regenerate Layout**.	• The Tcl script is used for the interest of time in this demonstration. • Once the Tcl script finishes, you should be able to see the IPI block design as shown below.
 diff_clock_rtl reset clk_wiz CLK_IN1_D reset clk_out1 locked Clocking Wizard rst_ps8_0_100M slowest_sync_clk ext_reset_in aux_reset_in mb_debug_sys_rst dcm_locked mb_reset bus_struct_reset[0:0] peripheral_reset[0:0] interconnect_aresetn[0:0] peripheral_aresetn[0:0] Processor System Reset jtag_axi_0 aclk aresetn M_AXI JTAG to AXI Master ps8_0_axi_periph S00_AXI S01_AXI ACLK ARESETN S00_ACLK S00_ARESETN M00_ACLK M00_ARESETN M01_ACLK M01_ARESETN S01_ACLK S01_ARESETN M00_AXI M01_AXI AXI Interconnect axi_gpio_0 S_AXI s_axi_aclk s_axi_aresetn GPIO AXI GPIO led_4bits axi_bram_ctrl_0 S_AXI s_axi_aclk s_axi_aresetn BRAM_PORTA BRAM_PORTB AXI BRAM Controller blk_mem_gen_0 BRAM_PORTA BRAM_PORTB rsta_busy rstb_busy Block Memory Generator	
• View the **Address Editor** tab. • Note that the Vivado Design Suite assigns addresses to the PL slave peripherals: • AXI block RAM controller, which is connected to the dual-port BRAM block • AXI GPIO-LEDs_linear, which drives the board LEDs	• The Address Editor tab shows the base and high address along with the size of each peripheral. The base address of the block RAM controller is 0xC0000000 and the AXI GPIO is 0x40000000.
• View the IPI block design and quickly summarize the position of the JTAG to AXI core.	• The JTAG to AXI master core is an AXI-based embedded system that connects to the slave interface of the AXI interconnect that has slave peripherals. • The JTAG to AXI master core is placed before the AXI interconnect as a master to make data transactions to block RAM and drive the LEDs on the board.
• Validate and save the updated design. • Select **Tools** > **Validate Design** to scan for any errors and critical warnings. Resolve any issues. • Select **File** > **Save Block Design**.	• The updated IPI block design needs to be validated for any errors before you save it.

Action with Description	Point of Emphasis and Key Takeaway
• Create a HDL top-level wrapper for the block design by letting the Vivado Design Suite manage the wrapper and auto-update.	• The HDL top-level wrapper instantiates the final IPI block design with port mapping. • A top-level source file is required; it can be generated by the Vivado Design Suite.
 Sources Design Signals Board Diagram Cell Design Sources (1) [redacted].bd) (1) Constraints Simulation Sources (1) sim_1 (1) Source Node Properties... Ctrl+E Open File Alt+O Create HDL Wrapper... View Instantiation Template Generate Output Products... Reset Output Products... Replace File... Copy File Into Project Copy All Files Into Project Alt+I	
• Now that the HDL wrapper is generated, it is time to implement the design and generate the bitstream. • However, in the interest of time, the bitstream has already been generated and is available in the project directory.	
• Set up and connect the board or verify that this has properly been done before turning on the power. • All the connections are made to the board itself. • Power on the board.	• Powering up the board allows you to launch the Vivado hardware manager.
• Click **Program and Debug** > **Open Hardware Manager** from the Flow Navigator to establish a connection to the board. • Click **Open target** > **Auto Connect** to connect to the target board automatically with the default settings.	• A hardware session is the utility of the Vivado Design Suite that enables the monitoring of debug cores that were added to a design.

Action with Description	Point of Emphasis and Key Takeaway
Now that connection to the board is established, your first task is to typically download a bitstream to your board. • Right-click the **xczu7_0** device and select **Program Device**. • Select the bitstream (**jtag2axi_wrapper.bit**) and probe (**jtag2axi_wrapper.ltx**) files from the project directory.	• A bitstream programming file is used to download to your hardware device, whereas debug probe files contain details of the probing signals for cores like VIO and ILA. • You can ignore the *debug.ltx* probe file in this case as you are not monitoring any internal design signals. • **Note:** The JTAG to AXI master core that you added to the design appears in the Hardware window under the target device. If you do not see the JTAG to AXI core appear, right-click the device and select **Refresh Device**. This re-scans the FPGA device and refreshes the Hardware window.
The hardware is now programmed, and you can see the AXI debug core in the device tree. • Select **hw_axi_1(AXI)** and explore the AXI core properties.	• The core is AXI4 Full type, which is a master to the AXI4 Lite peripherals on the other side of the interconnect.
• Review the properties of JTAG to AXI master core in the AXI Core Properties window.	• The JTAG to AXI master debug core is a customizable core that can generate the AXI transactions and drive the AXI signals internal to an FPGA at run time. • The core supports all memory-mapped AXI and AXI-Lite interfaces and can support a 32-bit or 64-bit wide data interface. • The JTAG to AXI master core can only be communicated with using Tcl Console commands. • You can create and run AXI read and write transactions using these Tcl Console commands.

Action with Description	Point of Emphasis and Key Takeaway
	
• Reset the JTAG to AXI master debug core. • Before creating and issuing transactions, it is important to reset the JTAG to AXI master core using the following Tcl command: `reset_hw_axi [get_hw_axis hw_axi_1]`	• The AXI side of the logic is reset when the JTAG-to-AXI master core samples this as low on the rising edge of the input clock. You need to use the `reset_hw_axi` Tcl command to reset the core.

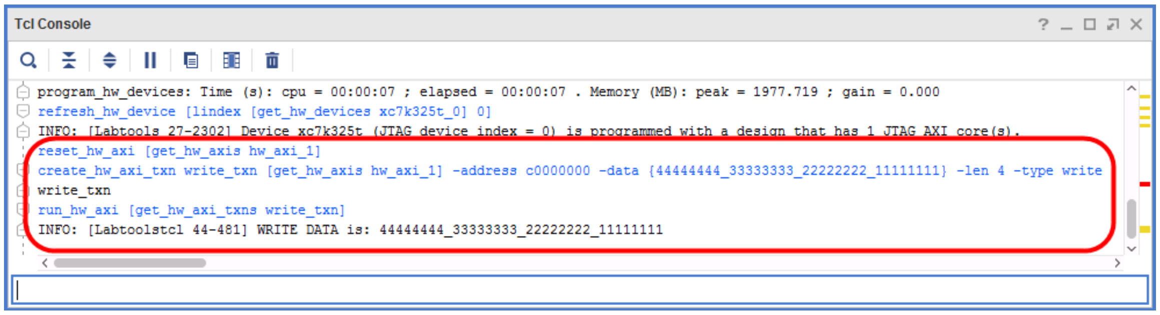
Hardware

Name	Status
localhost (1)	Connected
xilinx_tcf/Xilinx/94760A	Open
xczu7_0 (2)	Programmed
SysMon (System Mo	
hw_axi_1 (Base_Zyn	
arm_dap_1 (1)	N/A
SysMon (System Mo	

AXI Core Properties

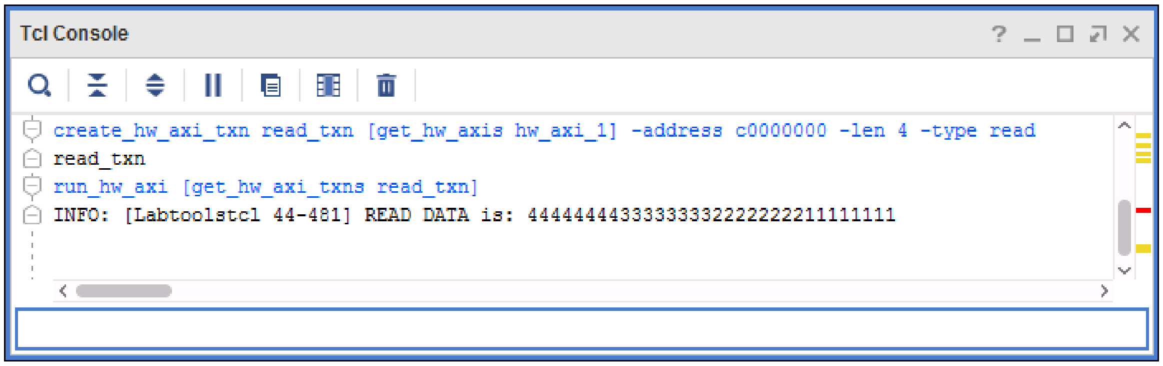
hw_axi_1

AXI_ADDR_WIDTH	32
AXI_DATA_WIDTH	32
AXI_ID_WIDTH	1
BURST_TYPE_FIXED_SUP	✓
BURST_TYPE_INCR_SUPF	✓
BURST_TYPE_WRAP_SUF	✓
CELL_NAME	Base_Zynq_MPSoC_i/jtag_a
CLASS	hw_axi
CORE_UUID	13222F8D6CE05DB282C80
NAME	hw_axi_1

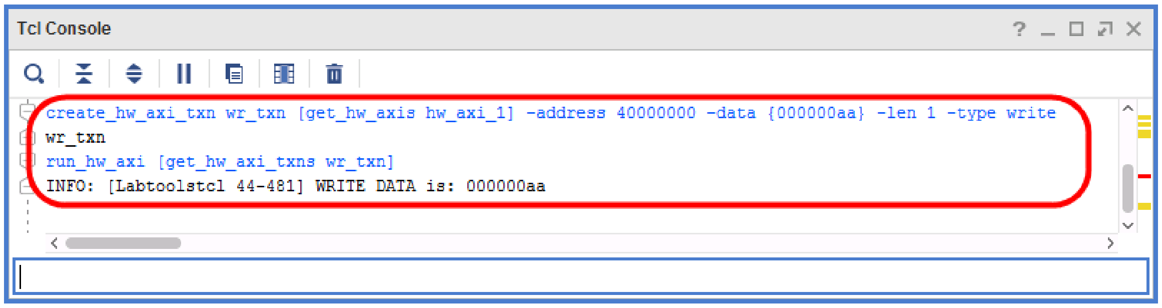
Action with Description	Point of Emphasis and Key Takeaway
• Create a write transaction into the block memory in the design. • The following Tcl command creates a 4-word AXI write burst transaction to the memory controller base address: `create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -address c0000000 -data {44444444_33333333_22222222_11111111} -len 4 -type write` where: ▪ *write_txn* is the user-defined name of the transaction. ▪ *[get_hw_axis hw_axi_1]* returns the hw_axi_1 object. ▪ *-address c0000000* is the base address of bram controller. ▪ *-data {44444444_33333333_22222222_11111111}* The -data direction is LSB to the left (i.e., address 0) and MSB to the right (i.e., address 3). ▪ *-len 4* sets the AXI burst length to four words.	• The 4-word AXI burst transaction writes to the locations of the block RAM starting from the base address.
• Run the write transaction that was just created by using the `run_hw_axi` command: `run_hw_axi [get_hw_axi_txns write_txn]`	• After creating the transaction in the above step, you ran it as a write transaction by using the `run_hw_axi` command. • The data *44444444_33333333_22222222_11111111* is now written to the memory.
	

```
program_hw_devices: Time (s): cpu = 00:00:07 ; elapsed = 00:00:07 . Memory (MB): peak = 1977.719 ; gain = 0.000
refresh_hw_device [lindex [get_hw_devices xc7k325t_0] 0]
INFO: [Labtools 27-2302] Device xc7k325t (JTAG device index = 0) is programmed with a design that has 1 JTAG AXI core(s).
reset_hw_axi [get_hw_axis hw_axi_1]
create_hw_axi_txn write_txn [get_hw_axis hw_axi_1] -address c0000000 -data {44444444_33333333_22222222_11111111} -len 4 -type write
write_txn
run_hw_axi [get_hw_axi_txns write_txn]
INFO: [Labtoolstcl 44-481] WRITE DATA is: 44444444_33333333_22222222_11111111
```

Action with Description	Point of Emphasis and Key Takeaway
• Create a read transaction to the block memory in the design. • The following Tcl command creates a 4-word AXI read burst transaction from the memory controller base address that was written in the previous steps: `create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -address c0000000 -len 4 -type read` where: ▪ *read_txn* is the user-defined name of the transaction. ▪ *[get_hw_axis hw_axi_1]* returns the hw_axi_1 object. ▪ *-address c0000000* is the base address of bram controller. ▪ *-len 4* sets the AXI burst length to fourwords.	• The 4-word AXI burst transaction reads from the locations of the block RAM starting from the base address.
• Run the read transaction that was just created by using the `run_hw_axi` command: `run_hw_axi [get_hw_axi_txns read_txn]`	• After creating the transaction in the above step, you ran it as a read transaction by using the `run_hw_axi` command. • The data *44444444333333332222222211111111* is now read back from the memory.



```
create_hw_axi_txn read_txn [get_hw_axis hw_axi_1] -address c0000000 -len 4 -type read
read_txn
run_hw_axi [get_hw_axi_txns read_txn]
INFO: [Labtoolstcl 44-481] READ DATA is: 44444444333333332222222211111111
```

Action with Description	Point of Emphasis and Key Takeaway
• Create a write data transaction on the board user 8-bit LEDs that are enabled in the design. • The following Tcl command creates a one-word AXI write transaction to the AXI GPIO-LEDs base address: `create_hw_axi_txn wr_txn [get_hw_axis hw_axi_1] -address 40000000 -data {000000aa} -len 1 -type write`	• The 1-word AXI data transaction writes to the base address of the user LEDs.
• Run the write transaction on user LEDs that was just created by using the `run_hw_axi` command: `run_hw_axi [get_hw_axi_txns wr_txn]`	• After creating the transaction in the above step, you ran it as a write transaction by using the `run_hw_axi` command. • The written data "000000aa" will be displayed on the board user LEDs.
	
• Close the hardware manager and exit the Vivado Design Suite.	• Exit the GUI and applications.
• Power off the evaluation board.	

```
create_hw_axi_txn wr_txn [get_hw_axis hw_axi_1] -address 40000000 -data {000000aa} -len 1 -type write
wr_txn
run_hw_axi [get_hw_axi_txns wr_txn]
INFO: [Labtoolstcl 44-481] WRITE DATA is: 000000aa
```

Summary

This demonstration showed how to add and integrate the JTAG to AXI debug IP core available to an embedded or non-embedded system and how to perform essential connections to the AXI peripherals in the IPI block design.

Launching the hardware server to open, set up a JTAG connection, and program the device was also shown. The use of the Vivado logic analyzer Tcl Console interface for run-time interaction with the target hardware was demonstrated as well. Note that the JTAG to AXI master debug core can only be communicated via Tcl commands.