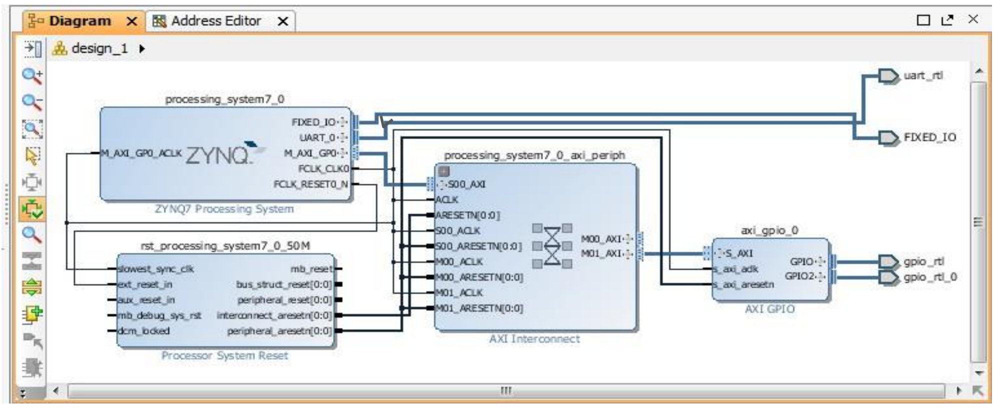
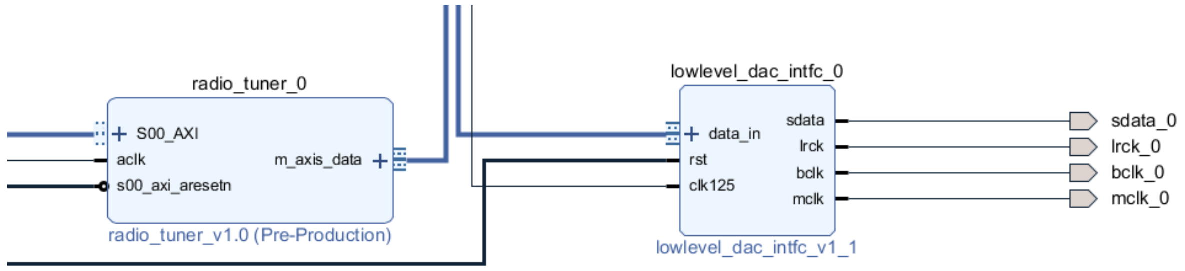
Johns Hopkins
Engineering for Professionals

System-on-Chip FPGA Design Lab Lecture
Topic – Custom AXI Peripherals



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

IP Packaging



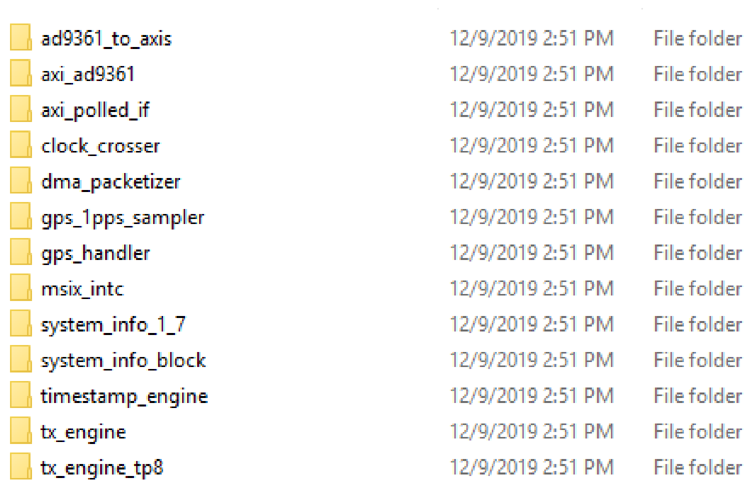
- Effective Design Re-Use Requires
  - Standard Interfaces (think AXI / AXI Stream) as much as possible
  - Well designed IP
    - Customizable for multiple uses
    - Bulletproof
  - Goal : a single file or directory that we can put in a catalog and people can use it without having to know anything about how it is constructed. We want a block that “just works”
  - With a catalog full of these, designs can be constructed and iterated rapidly
- Xilinx and other manufacturers come with a large suite of existing IP in the default catalog, but allow user defined IP “repositories” as a place to store IP purchased from other vendors, and IP built by you
- A project can be pointed to use an IP repository, and all of the IP which is stored in that repository will become accessible as part of your IP catalog window (which we are familiar with already)

*** note – packaging up a design into a piece of IP is likely something you will do after you test a bit. It adds a few steps, so it isn’t as agile of a process*

IP “Repository”

- IP Repository is just a directory which contains a bunch of IP
  - IP can be:
    - A directory folder for each packaged IP
    - A zip file which is essentially the same as the above
    - End result is the same
    - An IP repo can have a user-defined structure. The tools will browse through the entire tree of the repo looking for valid “packaged” IP
- Each IP in the repository when packaged is labelled by the person packing the IP with a few parameters which make up a unique identifier for the IP : “VLNV”
  - Vendor (e.g. “Johns Hopkins”)
  - Library (e.g. “DougsBlocks”)
  - Name (e.g. “lowlevel_dac_interface”)
  - Version (e.g. v1.0)
- A project can specify multiple IP Repositories and then the user of that project will have a catalog which contains the union of all of the IP in each repo
  - Only duplicate VLNV will possible cause problems

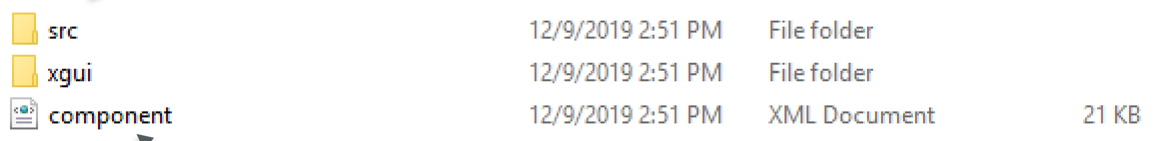
Example IP Repository



ad9361_to_axis	12/9/2019 2:51 PM	File folder
axi_ad9361	12/9/2019 2:51 PM	File folder
axi_polled_if	12/9/2019 2:51 PM	File folder
clock_crosser	12/9/2019 2:51 PM	File folder
dma_packetizer	12/9/2019 2:51 PM	File folder
gps_1pps_sampler	12/9/2019 2:51 PM	File folder
gps_handler	12/9/2019 2:51 PM	File folder
msix_intc	12/9/2019 2:51 PM	File folder
system_info_1_7	12/9/2019 2:51 PM	File folder
system_info_block	12/9/2019 2:51 PM	File folder
timestamp_engine	12/9/2019 2:51 PM	File folder
tx_engine	12/9/2019 2:51 PM	File folder
tx_engine_tp8	12/9/2019 2:51 PM	File folder

Directory (or zip) for each IP Core

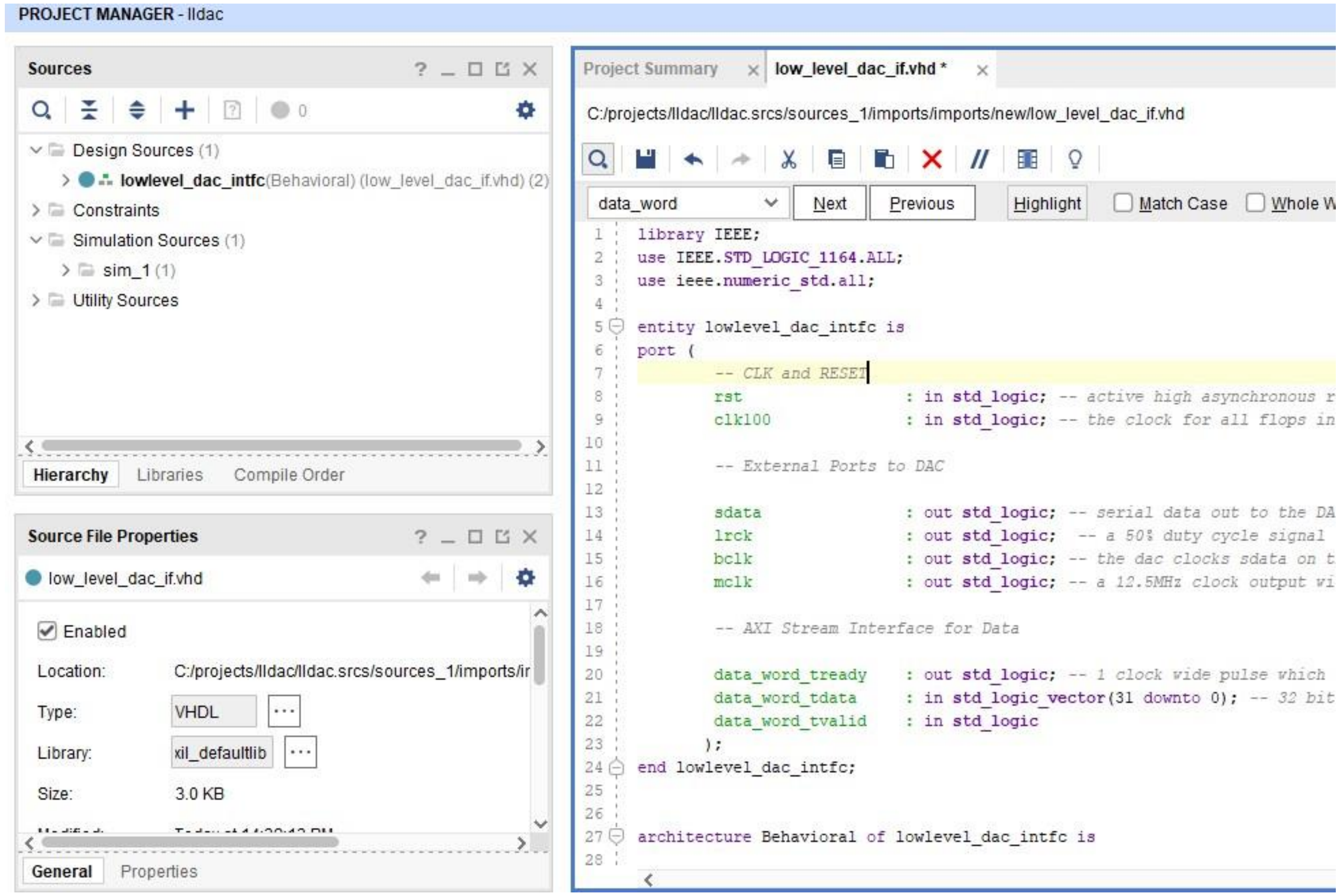
All the source files for that core



src	12/9/2019 2:51 PM	File folder	
xgui	12/9/2019 2:51 PM	File folder	
component	12/9/2019 2:51 PM	XML Document	21 KB

information about the IP, what the ports are, the VLNV…etc.

Designs get turned into IP (in this format) through a process called “Packaging”. The next few slides will show that process for our lowlevel_dac_interface.


PROJECT MANAGER - lldac

Sources

Design Sources (1)
lowlevel_dac_intfc(Behavioral) (low_level_dac_if.vhd) (2)
Constraints
Simulation Sources (1)
sim_1 (1)
Utility Sources

Hierarchy | Libraries | Compile Order

Source File Properties

low_level_dac_if.vhd

Enabled
Location: C:/projects/lldac/lldac.srcs/sources_1/imports/ir
Type: VHDL
Library: xil_defaultlib
Size: 3.0 KB

General | Properties

Project Summary | low_level_dac_if.vhd *

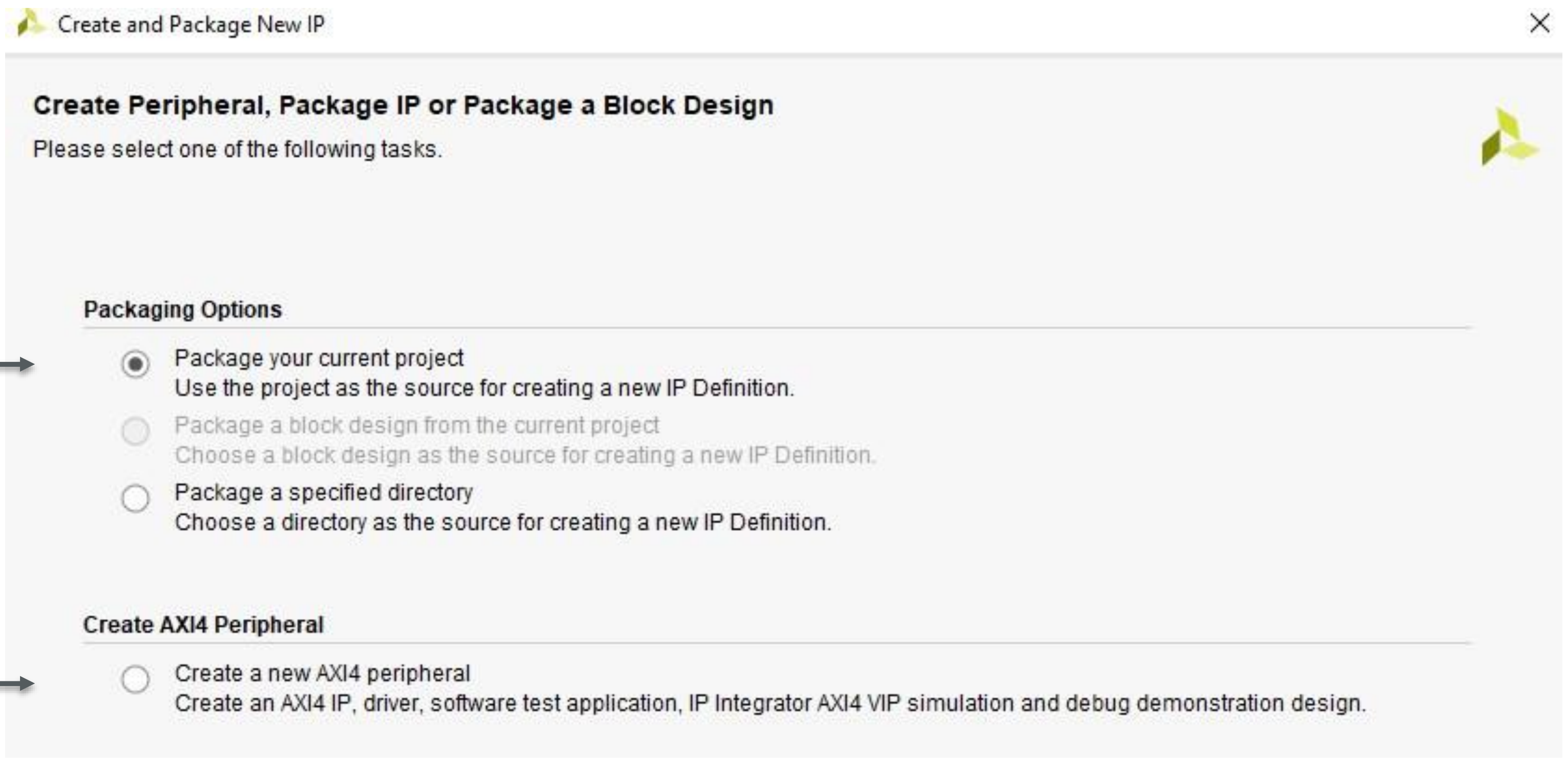
C:/projects/lldac/lldac.srcs/sources_1/imports/imports/new/low_level_dac_if.vhd

data_word | Next | Previous | Highlight | Match Case | Whole W

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity lowlevel_dac_intfc is
port (
        -- CLK and RESET
        rst                     : in std_logic; -- active high asynchronous r
        clk100                  : in std_logic; -- the clock for all flops in

        -- External Ports to DAC

        sdata                   : out std_logic; -- serial data out to the DA
        lrck                    : out std_logic;  -- a 50% duty cycle signal
        bclk                    : out std_logic; -- the dac clocks sdata on t
        mclk                    : out std_logic; -- a 12.5MHz clock output wi

        -- AXI Stream Interface for Data

        data_word_tready        : out std_logic; -- 1 clock wide pulse which
        data_word_tdata         : in std_logic_vector(31 downto 0); -- 32 bit
        data_word_tvalid        : in std_logic
       );
end lowlevel_dac_intfc;


architecture Behavioral of lowlevel_dac_intfc is

```

- Start with a standard Vivado project
- Note that we've changed a few port names. This isn't required, but I want to package the dac_interface as an AXI stream component, so best practice is to name the signals with the convention. (It will also allow the tool to autodetect the AXI-S interconnect)
- Select Tools->Create and Package New IP

Create and Package New IP



Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

Please select one of the following tasks.

Packaging Options

[✓ Package your current project]
Use the project as the source for creating a new IP Definition.

Package a block design from the current project
Choose a block design as the source for creating a new IP Definition.

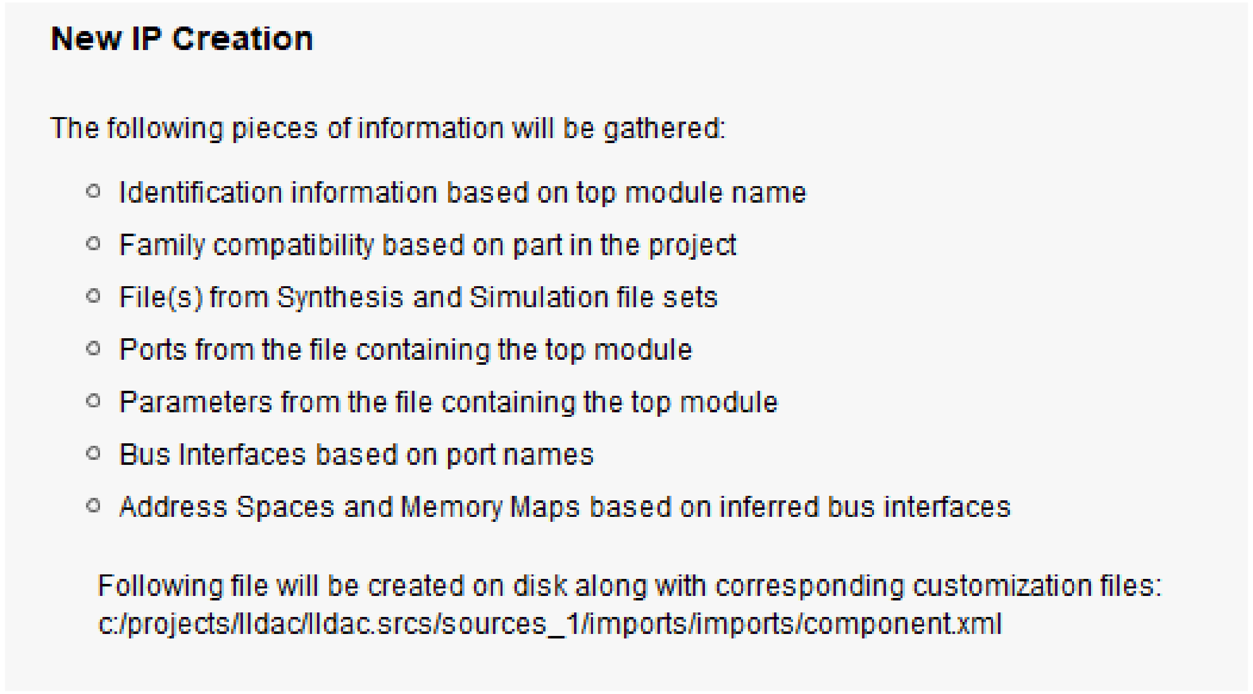
Package a specified directory
Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

Create a new AXI4 peripheral
Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

Choose this now

We will do this later today


New IP Creation

The following pieces of information will be gathered:

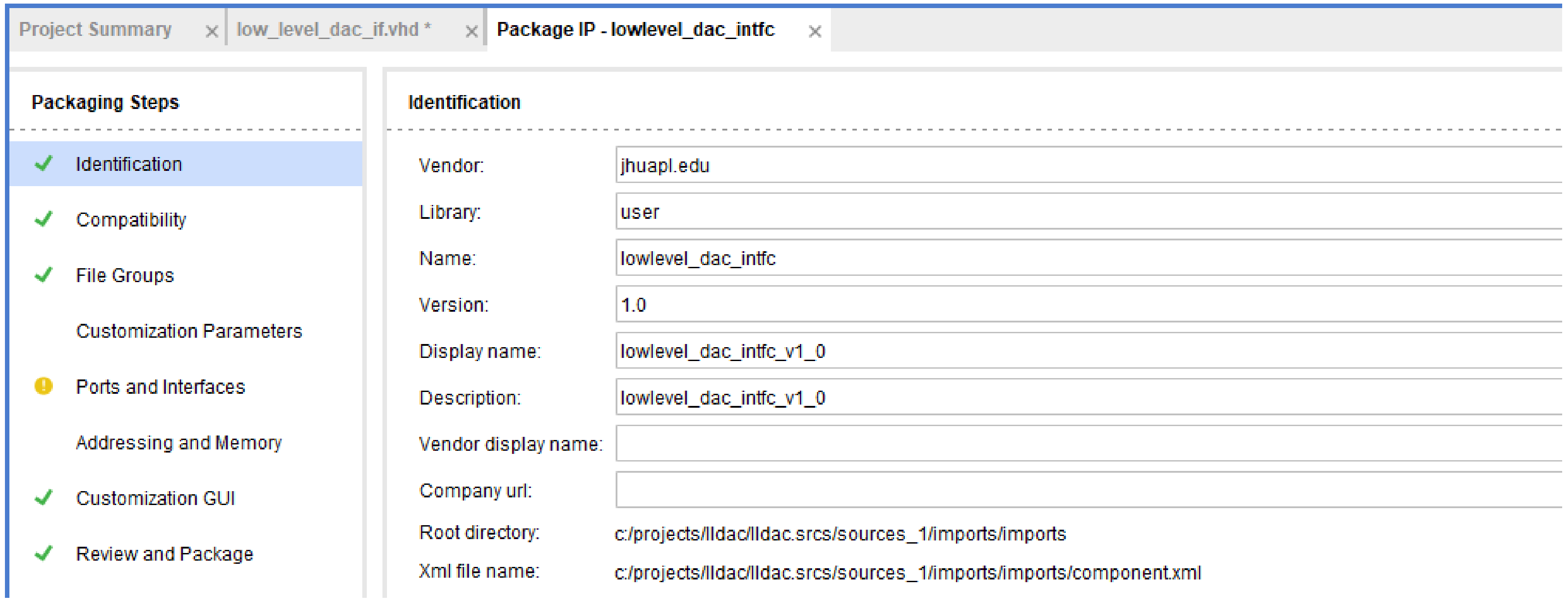
- Identification information based on top module name
- Family compatibility based on part in the project
- File(s) from Synthesis and Simulation file sets
- Ports from the file containing the top module
- Parameters from the file containing the top module
- Bus Interfaces based on port names
- Address Spaces and Memory Maps based on inferred bus interfaces

Following file will be created on disk along with corresponding customization files:
c:/projects/lldac/lldac.srcs/sources_1/imports/imports/component.xml

This dialog is a very good summary of the packaging process. In other words, what metadata about this core (and the use thereof) is required for it to be a plug-in block in IP integrator? Also, what files should go along with the core if we are sending it to be used by someone else? Instruction manual? Testbench? This is also the appropriate place for including things which would be useful even if not strictly required to be compiled into a design

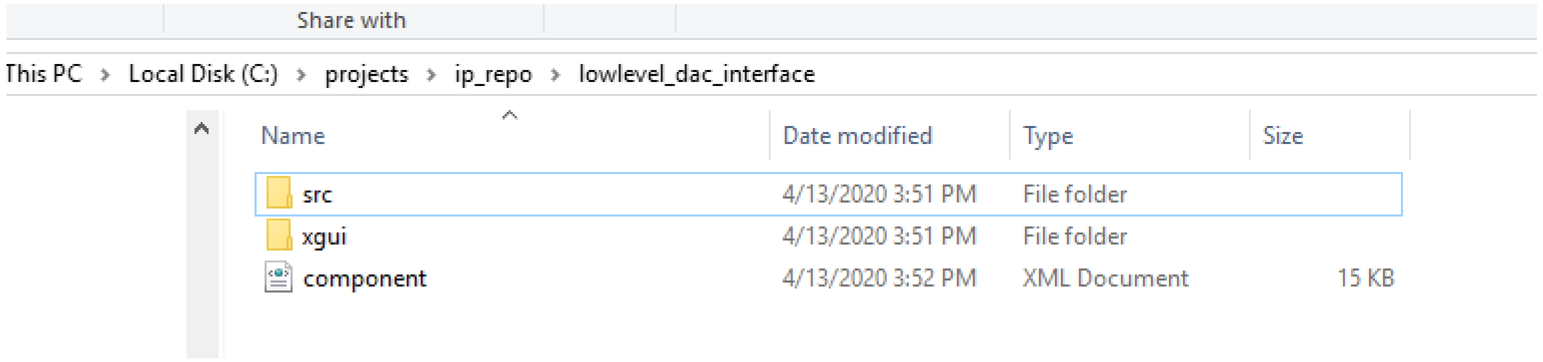
Main Packaging Dialog

- Project itself has a new file added that makes the project a “Packager Project”, meaning that this extra tab pops up when you open it
- Info about the IP is collected through all these fields (**walkthrough live done in lecture**)
- When you click “Package” the actual final product is saved off to a directory of your choice (it can be moved freely at any time to any repository location)



Vendor:	jhuapl.edu
Library:	user
Name:	lowlevel_dac_intfc
Version:	1.0
Display name:	lowlevel_dac_intfc_v1_0
Description:	lowlevel_dac_intfc_v1_0
Vendor display name:	
Company url:	
Root directory:	c:/projects/lldac/lldac.srcs/sources_1/imports/imports
Xml file name:	c:/projects/lldac/lldac.srcs/sources_1/imports/imports/component.xml

End Result – Packaged IP



This PC › Local Disk (C:) › projects › ip_repo › lowlevel_dac_interface

Name	Date modified	Type	Size
src	4/13/2020 3:51 PM	File folder	
xgui	4/13/2020 3:51 PM	File folder	
component	4/13/2020 3:52 PM	XML Document	15 KB

- In any project – part of the project settings are a list of IP repositories that you point to. If (in this case) c:\projects\ip_repo is selected, then “lowlevel_dac_interface” will show up as a choice in your IP catalog

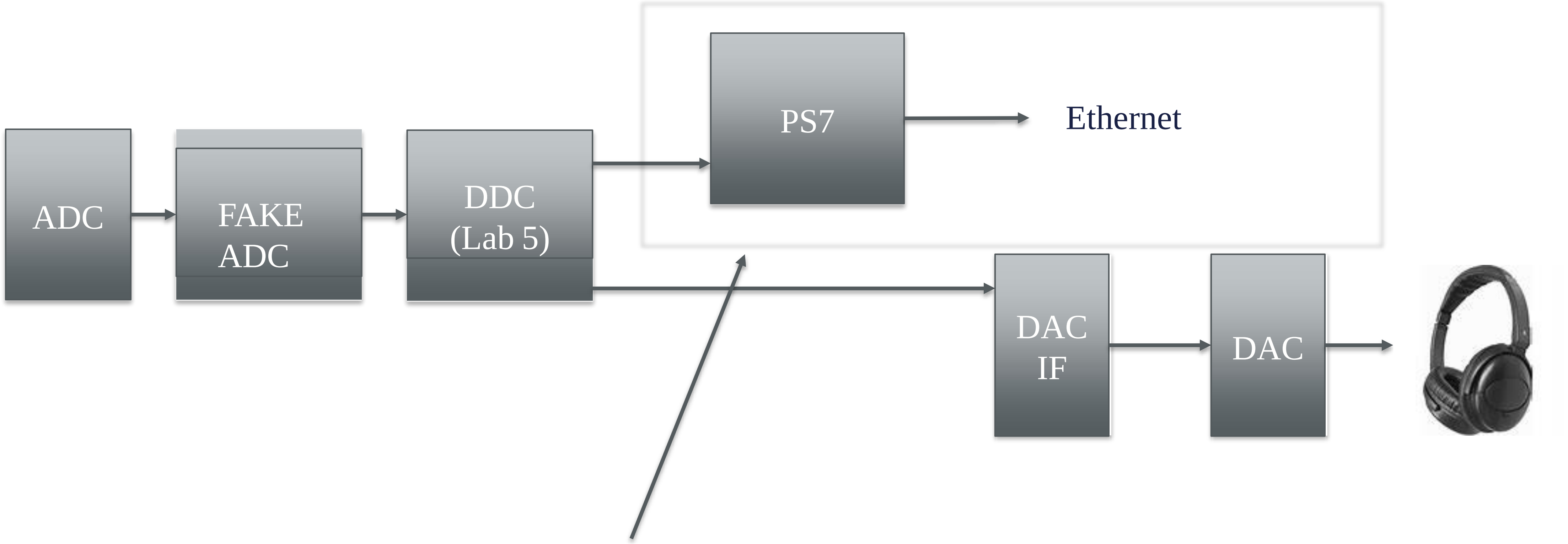
- If you put it in the wrong spot – no worries at all, you can grab the created files and move them around wherever you want as long as you move the whole IP together

- Lets look at these live:
  - An ip_repo
  - Project Settings to point to a repo
  - The associated catalog

Section 2

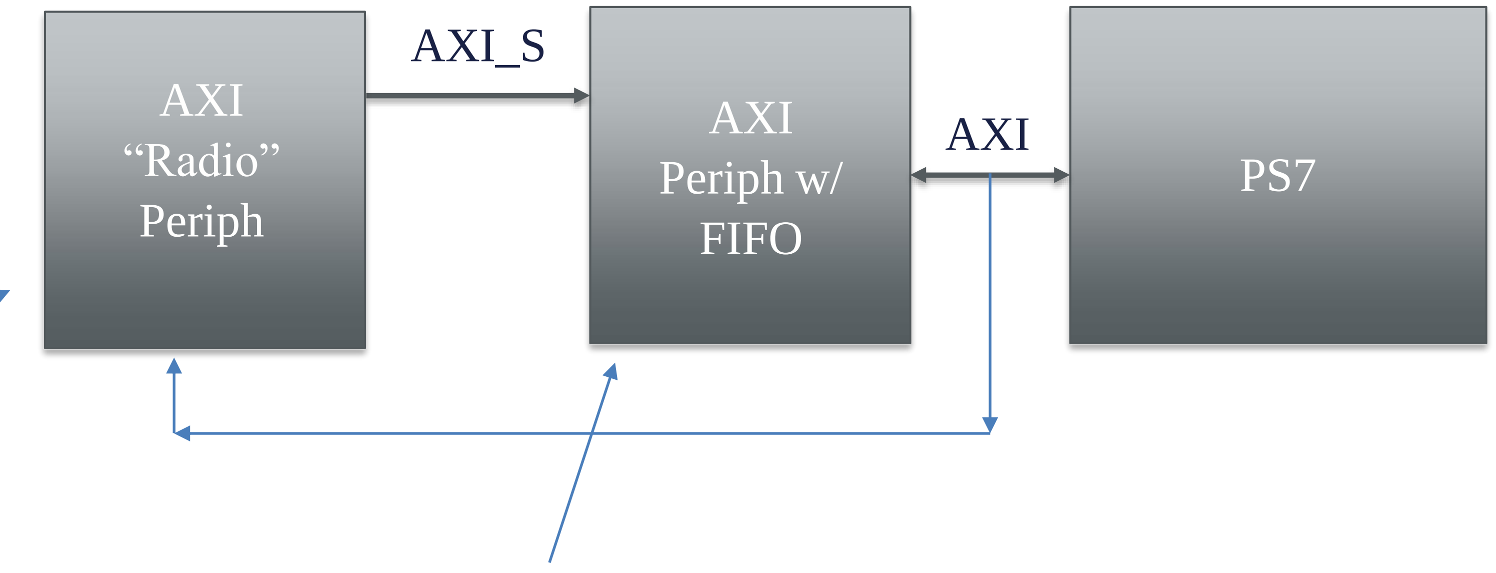
MAKING AXI CUSTOM PERIPHERALS

Simplified View of SDR



How do we do this? How do we get our samples to the PS7?
(There is no in-built AXI stream interface)

Proposed Solution (for that small piece)



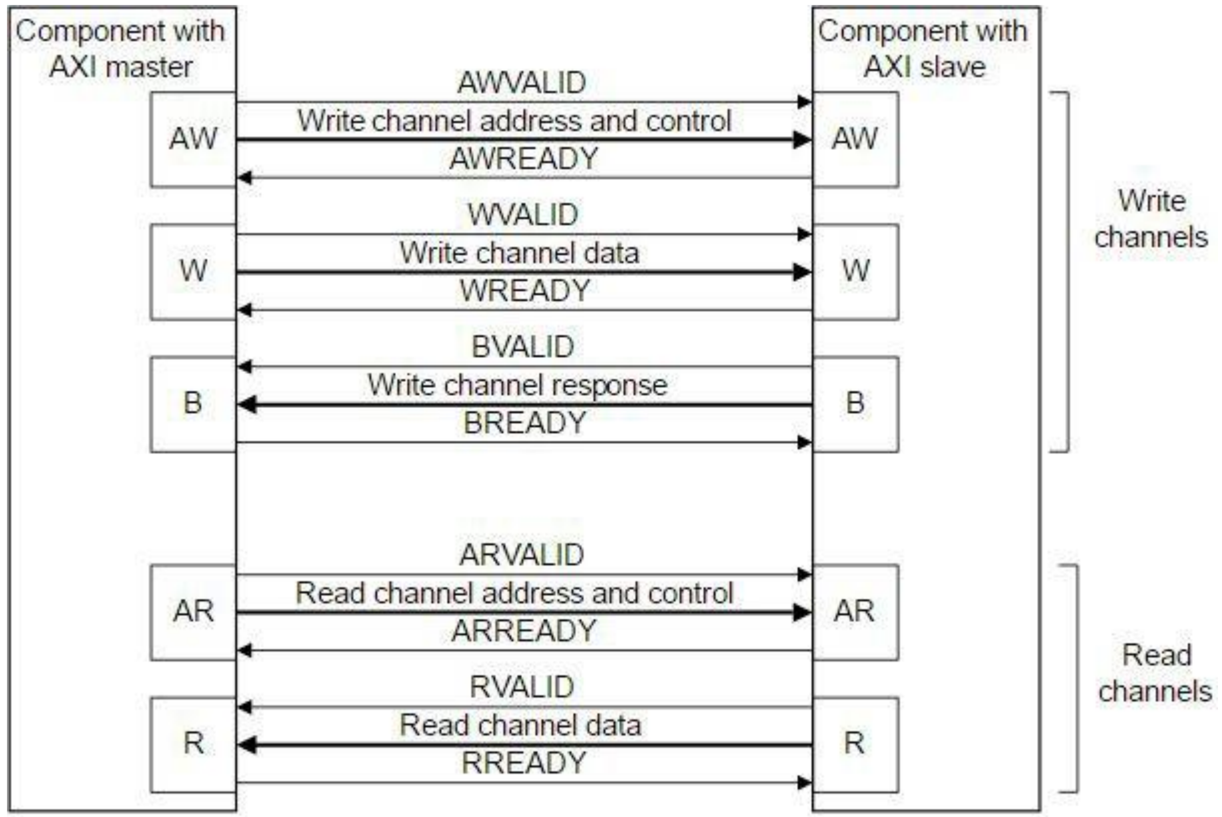
Contains the whole radio (DDSs, mixer, filters). Processor tunes DDS values with a register write over AXI

Contains a FIFO that holds data written from an AXI stream (your filter output). The processor can query the number of words in the FIFO at any time, and read them out by doing an AXI read

AXI Bus

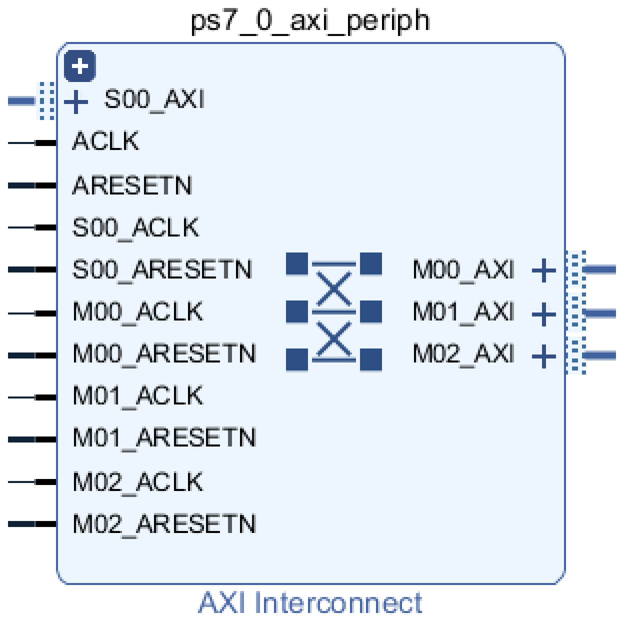
- Specification available from ARM with registration
- Xilinx implementation
  - http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf
- Provides a standard way to connect blocks, guaranteeing interoperability (at the bus interface level) in Systems-on-chip
- Xilinx provides a wizard driven template process to show you how to build an axi compliant peripheral. The AXI details are abstracted away into a simpler IPIF (IP Interface)

AXI Protocol



AXI Protocol /Interconnect

This Interconnect allows Master to be connected to many slaves



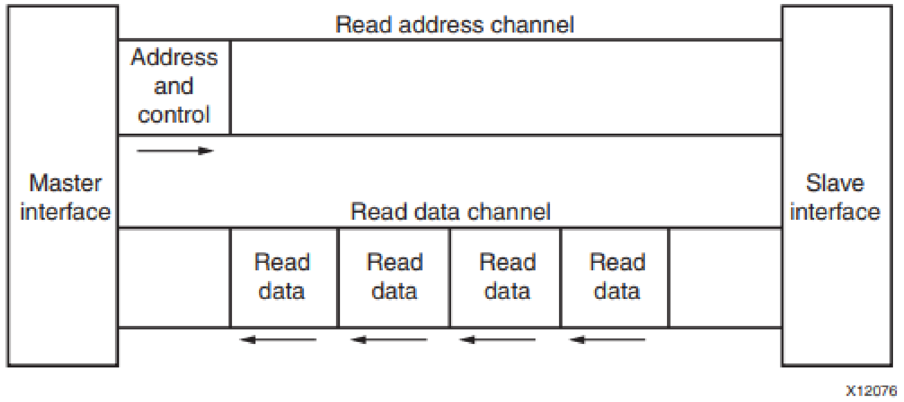


Figure 1-1: **Channel Architecture of Reads**

Figure 1-2 shows how a Write transaction uses the Write address, Write data, and Write response channels.

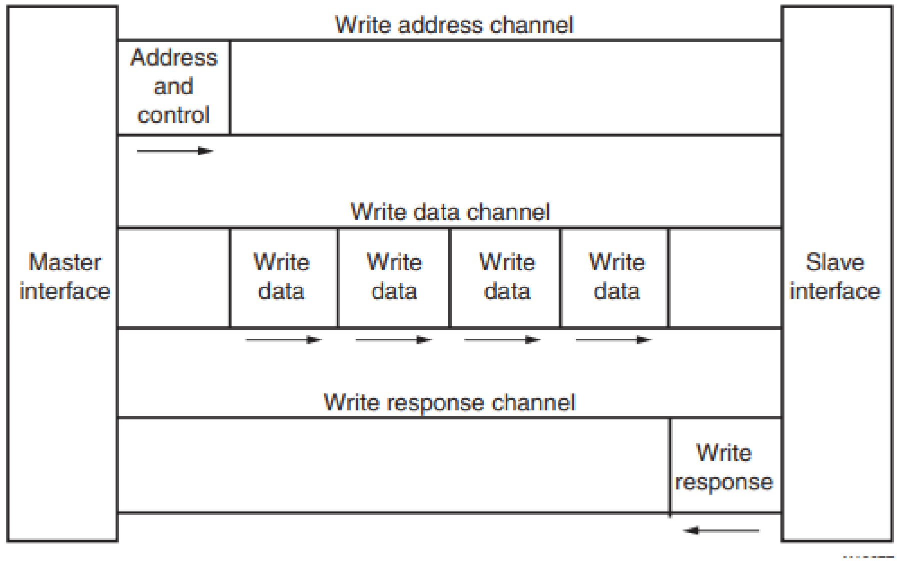
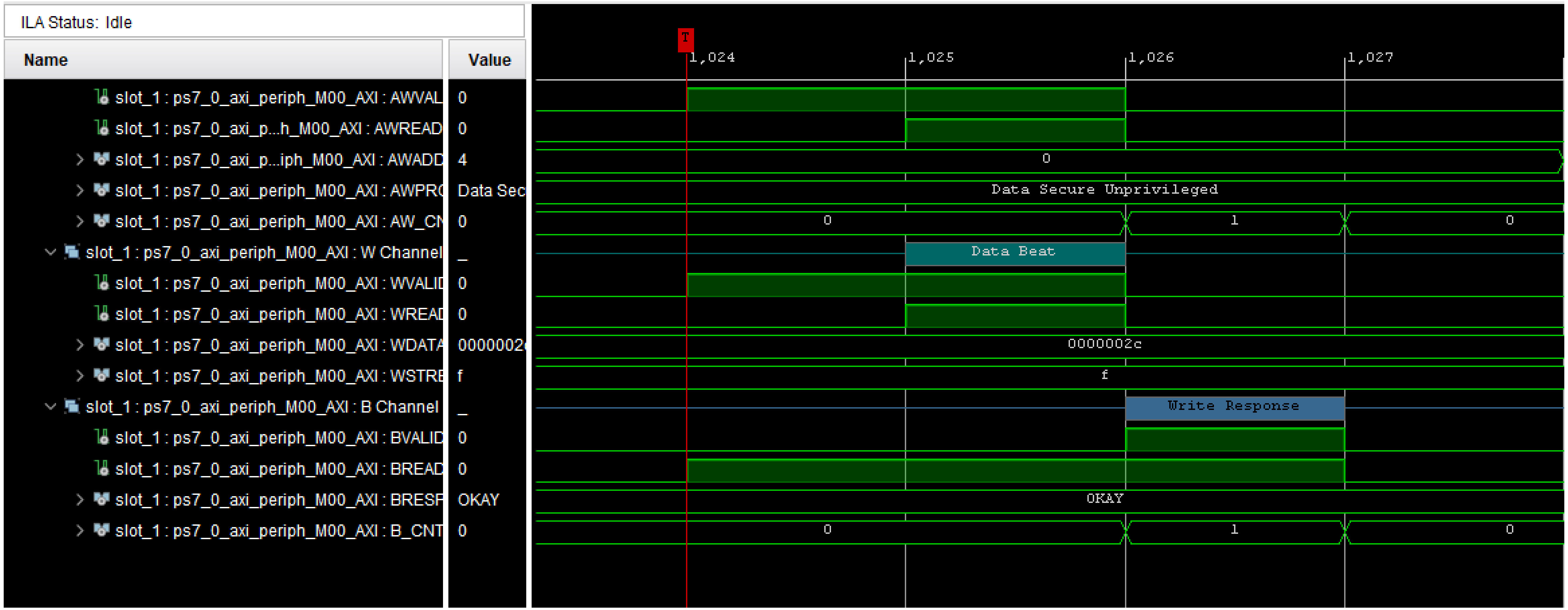


Figure 1-2: **Channel Architecture of Writes**

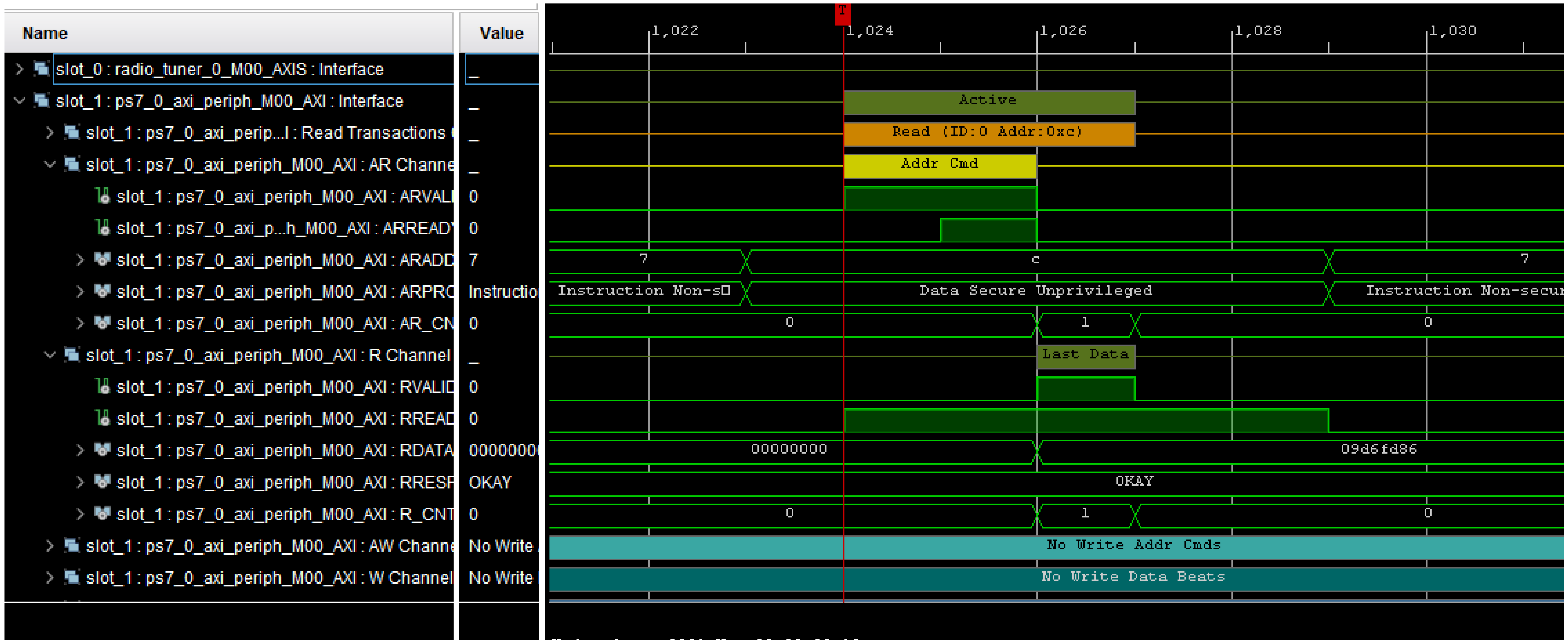
https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

AXI Timing, a Write to address x43c00000, with value 44


ILA Status: Idle

Name	Value
slot_1 : ps7_0_axi_periph_M00_AXI : AWVAL	0
slot_1 : ps7_0_axi_p...h_M00_AXI : AWREAD	0
slot_1 : ps7_0_axi_p...iph_M00_AXI : AWADD	4
slot_1 : ps7_0_axi_periph_M00_AXI : AWPRO	Data Sec
slot_1 : ps7_0_axi_periph_M00_AXI : AW_CN	0
slot_1 : ps7_0_axi_periph_M00_AXI : W Channel	_
slot_1 : ps7_0_axi_periph_M00_AXI : WVALID	0
slot_1 : ps7_0_axi_periph_M00_AXI : WREAD	0
slot_1 : ps7_0_axi_periph_M00_AXI : WDATA	0000002
slot_1 : ps7_0_axi_periph_M00_AXI : WSTRB	f
slot_1 : ps7_0_axi_periph_M00_AXI : B Channel	_
slot_1 : ps7_0_axi_periph_M00_AXI : BVALID	0
slot_1 : ps7_0_axi_periph_M00_AXI : BREAD	0
slot_1 : ps7_0_axi_periph_M00_AXI : BRESP	OKAY
slot_1 : ps7_0_axi_periph_M00_AXI : B_CNT	0

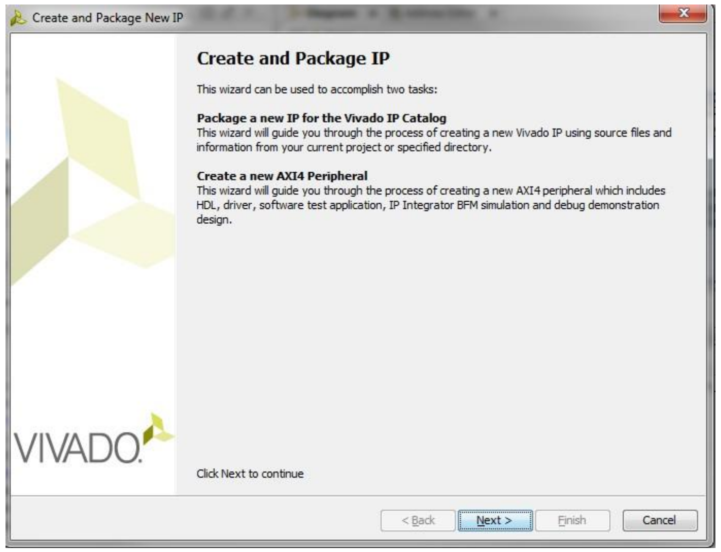
AXI Timing, a Read from address 0x43C0000c, returned 09d6fd86



Name	Value
slot_0 : radio_tuner_0_M00_AXIS : Interface	_
slot_1 : ps7_0_axi_periph_M00_AXI : Interface	_
slot_1 : ps7_0_axi_perip...l : Read Transactions	_
slot_1 : ps7_0_axi_periph_M00_AXI : AR Channe	_
slot_1 : ps7_0_axi_periph_M00_AXI : ARVALI	0
slot_1 : ps7_0_axi_p...h_M00_AXI : ARREADY	0
slot_1 : ps7_0_axi_periph_M00_AXI : ARADD	7
slot_1 : ps7_0_axi_periph_M00_AXI : ARPRO	Instructio
slot_1 : ps7_0_axi_periph_M00_AXI : AR_CN	0
slot_1 : ps7_0_axi_periph_M00_AXI : R Channel	_
slot_1 : ps7_0_axi_periph_M00_AXI : RVALID	0
slot_1 : ps7_0_axi_periph_M00_AXI : RREAD	0
slot_1 : ps7_0_axi_periph_M00_AXI : RDATA	0000000
slot_1 : ps7_0_axi_periph_M00_AXI : RRESP	OKAY
slot_1 : ps7_0_axi_periph_M00_AXI : R_CNT	0
slot_1 : ps7_0_axi_periph_M00_AXI : AW Channe	No Write
slot_1 : ps7_0_axi_periph_M00_AXI : W Channel	No Write

Creating a Peripheral : (Getting Started)

- Vivado provides a wizard which will give you a framework peripheral for you
  - Will contain some standard features like registers and FIFOs
  - You edit sections of the HDL to put your custom logic in
  - **AXI interfacing is abstracted a bit, which is nice!**
- Get Started with Tools -> Create and Package IP



Create and Package IP

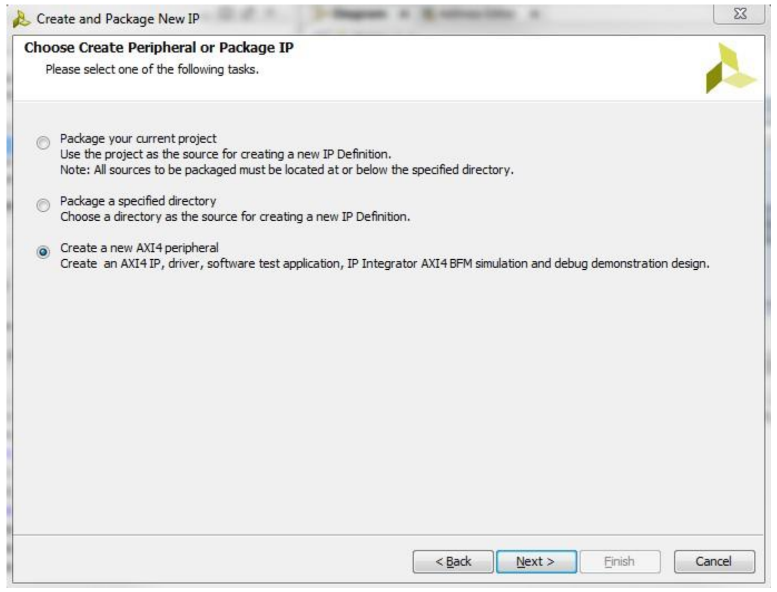
This wizard can be used to accomplish two tasks:

Package a new IP for the Vivado IP Catalog
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project or specified directory.

Create a new AXI4 Peripheral
This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator BFM simulation and debug demonstration design.

Click Next to continue

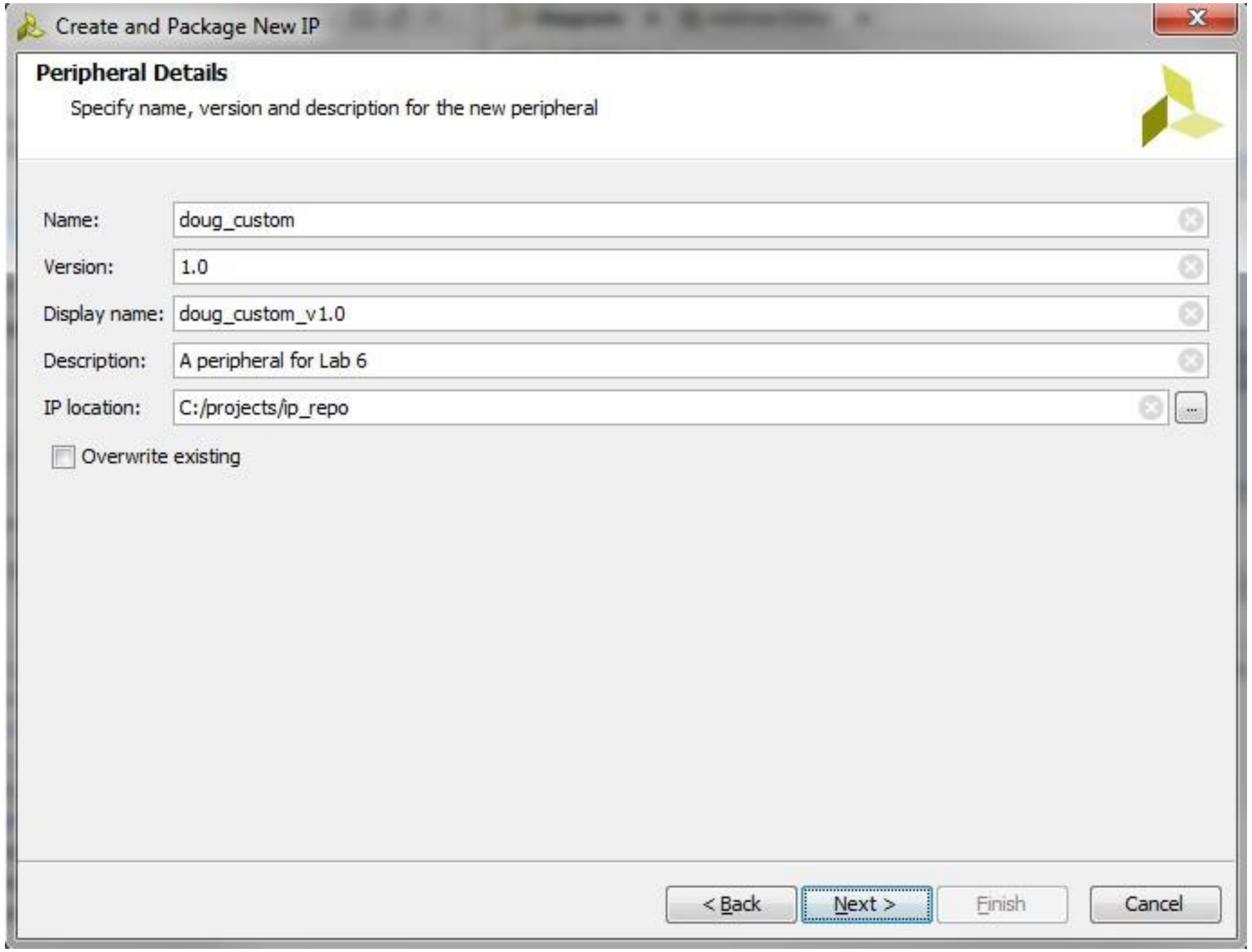
→



Choose Create Peripheral or Package IP
Please select one of the following tasks.

- Package your current project
  Use the project as the source for creating a new IP Definition.
  Note: All sources to be packaged must be located at or below the specified directory.
- Package a specified directory
  Choose a directory as the source for creating a new IP Definition.
- [✓ Create a new AXI4 peripheral]
  Create an AXI4 IP, driver, software test application, IP Integrator AXI4 BFM simulation and debug demonstration design.

Name, Location, Version, etc.


Create and Package New IP
Peripheral Details
Specify name, version and description for the new peripheral

Name: doug_custom

Version: 1.0

Display name: doug_custom_v1.0

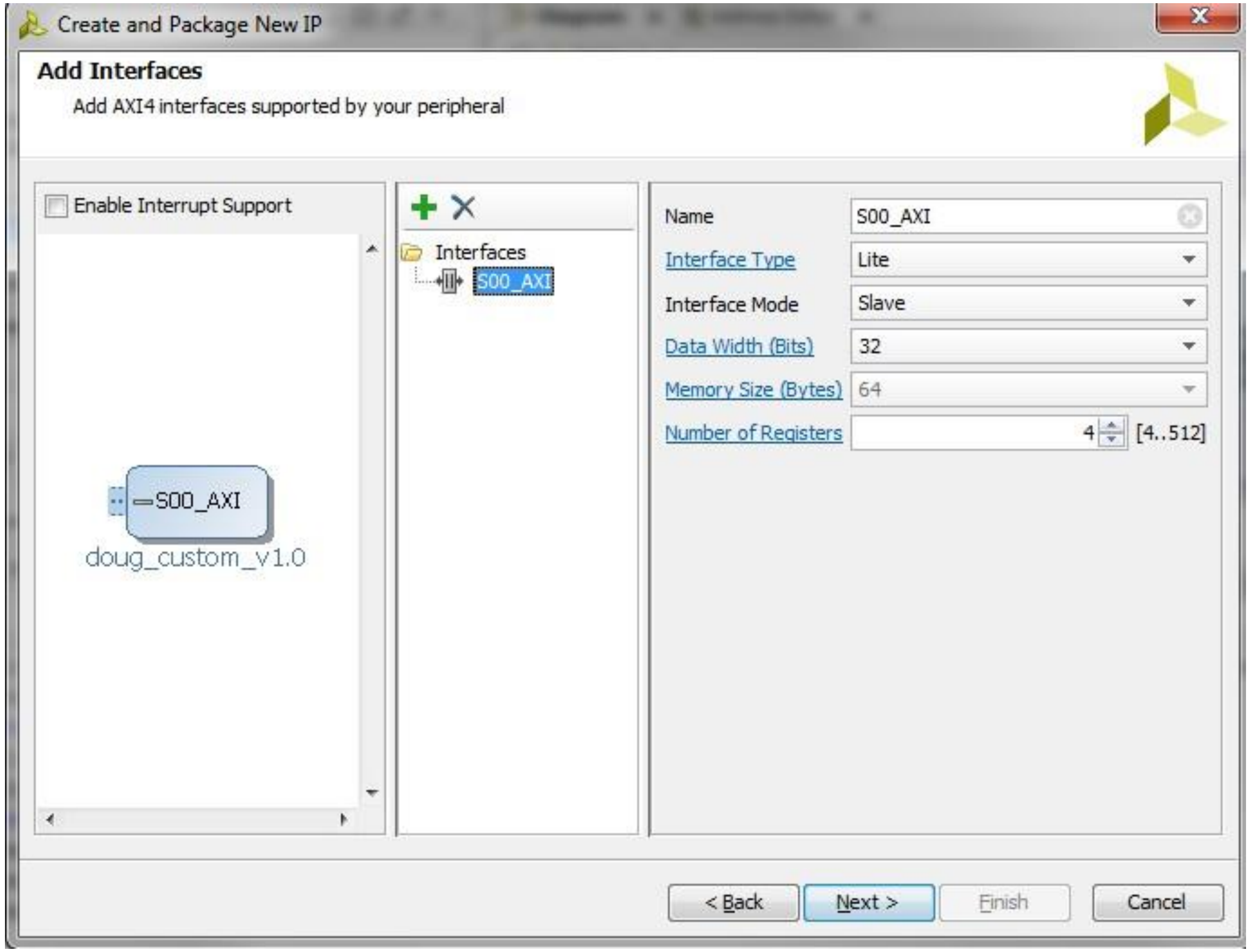
Description: A peripheral for Lab 6

IP location: C:/projects/ip_repo

[ ] Overwrite existing

< Back
Next >
Finish
Cancel

Interface Details


Create and Package New IP

Add Interfaces

Add AXI4 interfaces supported by your peripheral

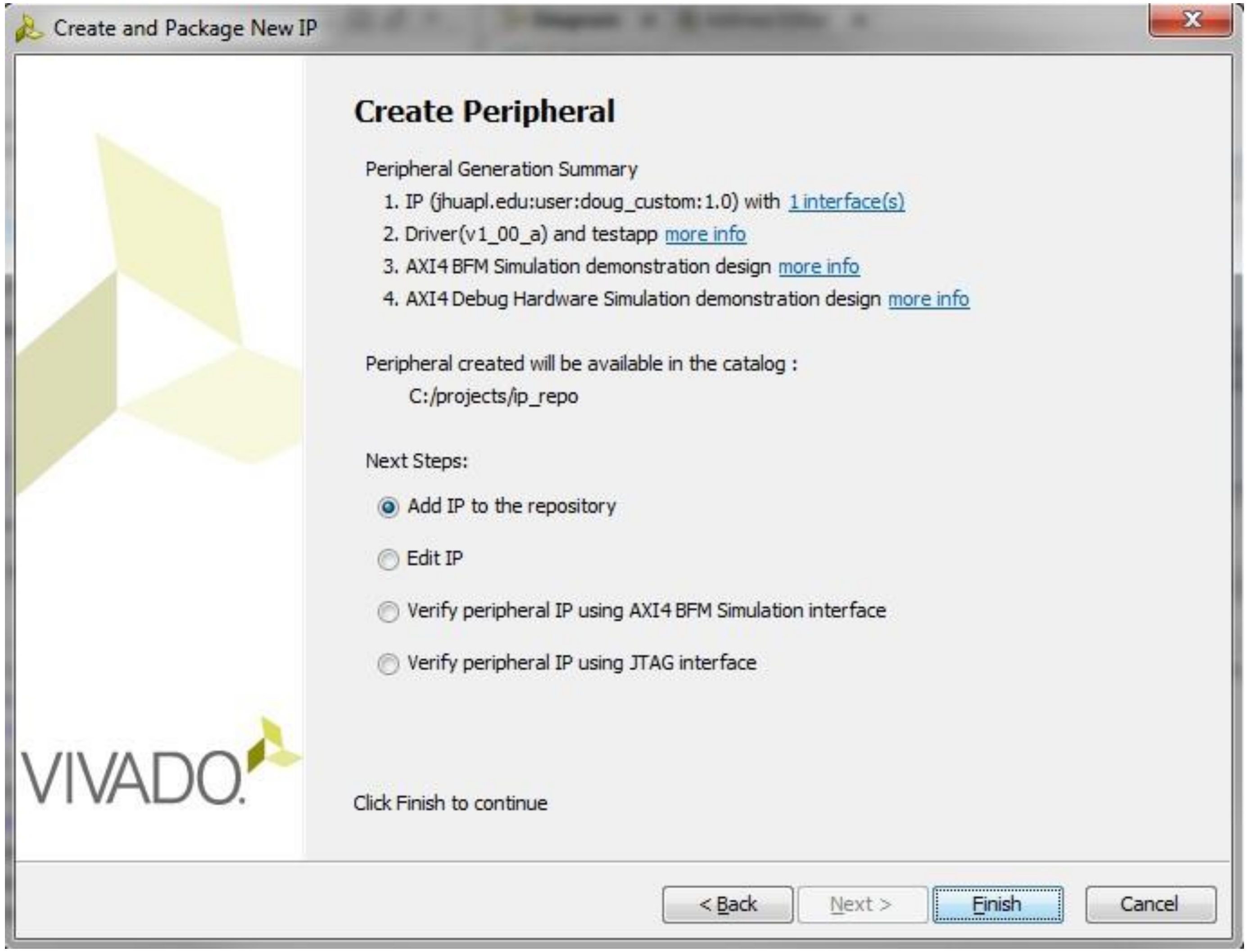
Enable Interrupt Support

Interfaces
S00_AXI

Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	4 [4..512]

We will build a sample peripheral which is an AXI slave, and has 4 registers

Finalize



Create Peripheral

Peripheral Generation Summary
1. IP (jhuapl.edu:user:doug_custom:1.0) with 1 interface(s)
2. Driver(v1_00_a) and testapp more info
3. AXI4 BFM Simulation demonstration design more info
4. AXI4 Debug Hardware Simulation demonstration design more info

Peripheral created will be available in the catalog :
C:/projects/ip_repo

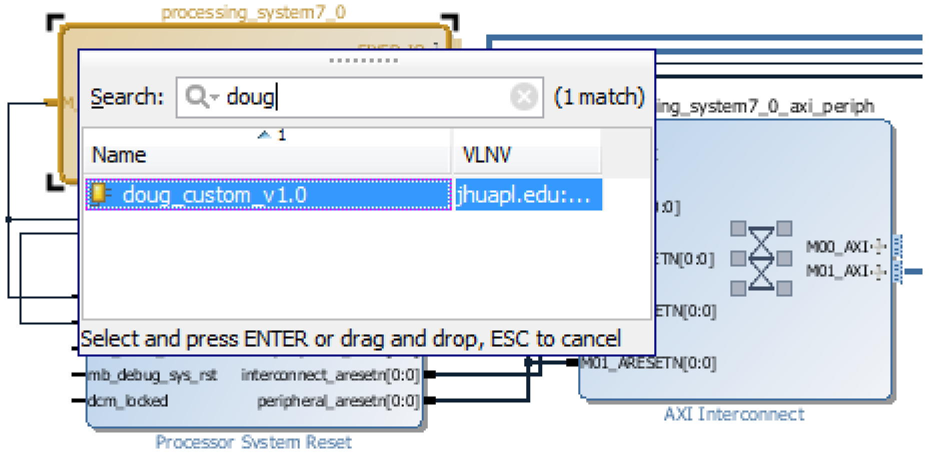
Next Steps:
- [✓ Add IP to the repository]
- Edit IP
- Verify peripheral IP using AXI4 BFM Simulation interface
- Verify peripheral IP using JTAG interface

Click Finish to continue

Take this IP and give it to all your friends!

Instantiating

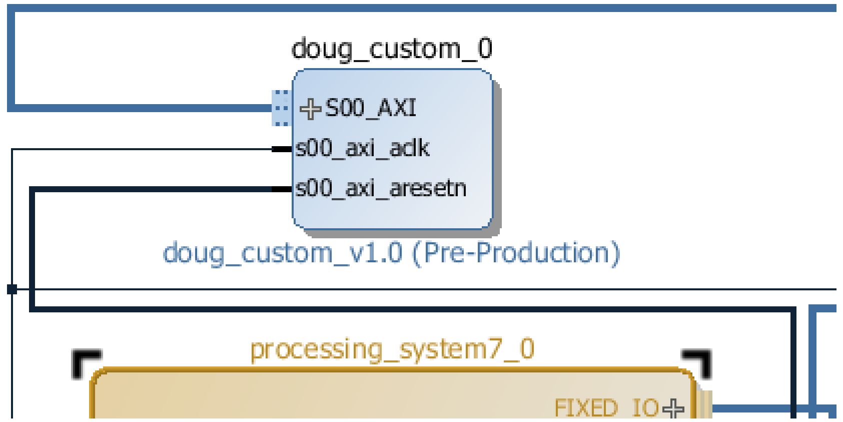
"Add IP" Dialog now shows our peripheral, and will let us add it to the design.

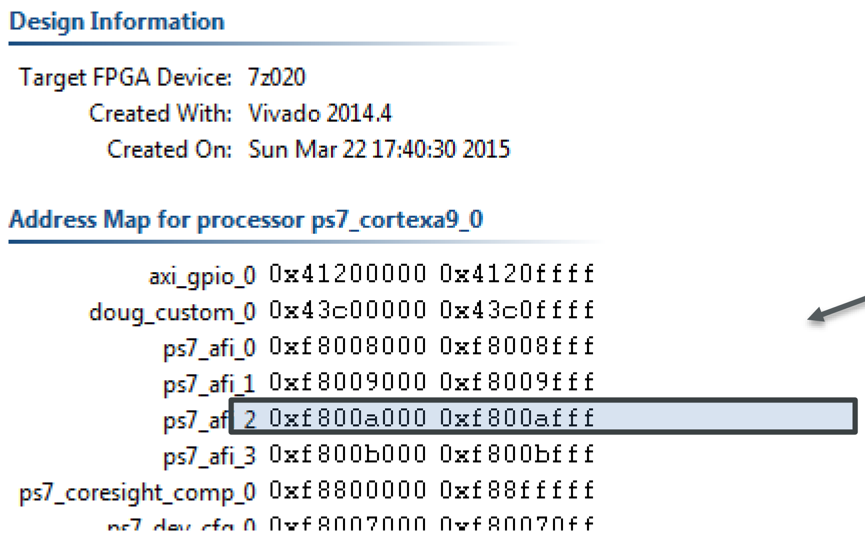


Search: doug (1 match)

Name	VLNV
doug_custom_v1.0	jhuapl.edu:...

Select and press ENTER or drag and drop, ESC to cancel





Design Information

Target FPGA Device: 7z020
Created With: Vivado 2014.4
Created On: Sun Mar 22 17:40:30 2015

Address Map for processor ps7_cortexa9_0

```
            axi_gpio_0 0x41200000 0x4120ffff
        doug_custom_0 0x43c00000 0x43c0ffff
             ps7_afi_0 0xf8008000 0xf8008fff
             ps7_afi_1 0xf8009000 0xf8009fff
             ps7_afi_2 0xf800a000 0xf800afff
             ps7_afi_3 0xf800b000 0xf800bfff
ps7_coresight_comp_0 0xf8800000 0xf88fffff
       ps7_dev_cfg_0 0xf8007000 0xf80070ff
```

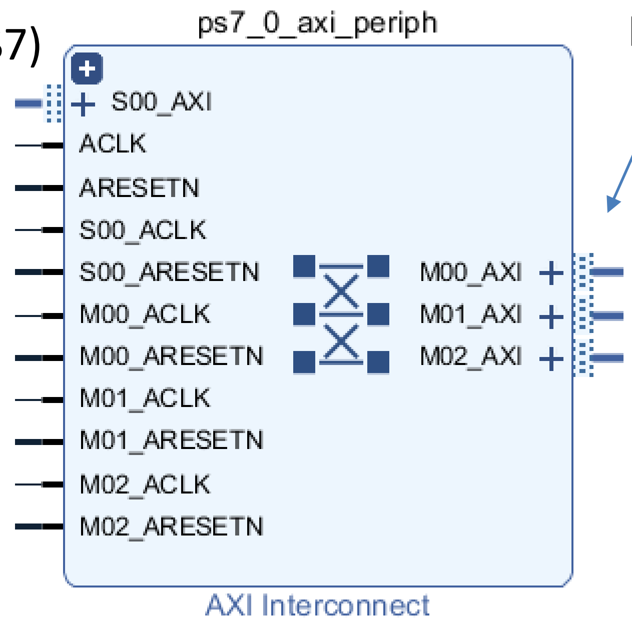
After exporting to SDK -

<In xparameters.h….>
#define XPAR_DOUG_CUSTOM_0_S00_AXI_BASEADDR 0x43C00000

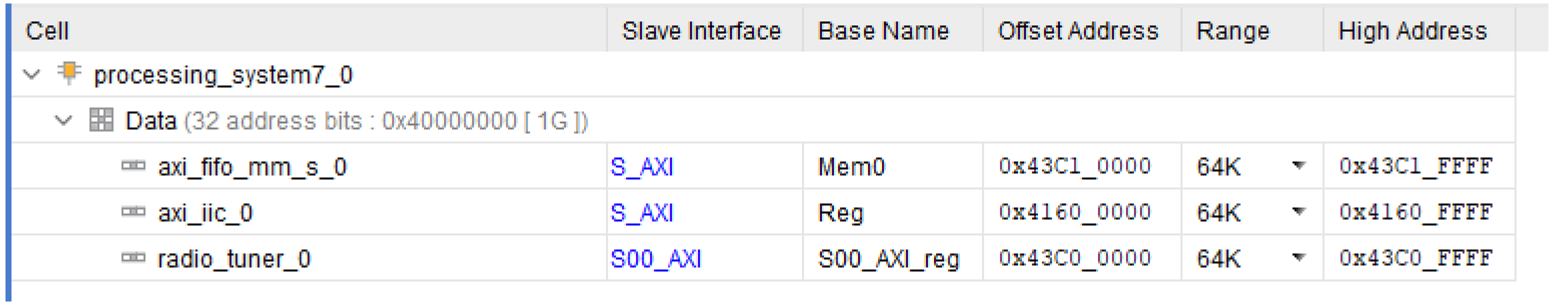
The AXI Interconnect in our system

Single Master (PS7)

Multiple slaves (our peripherals)

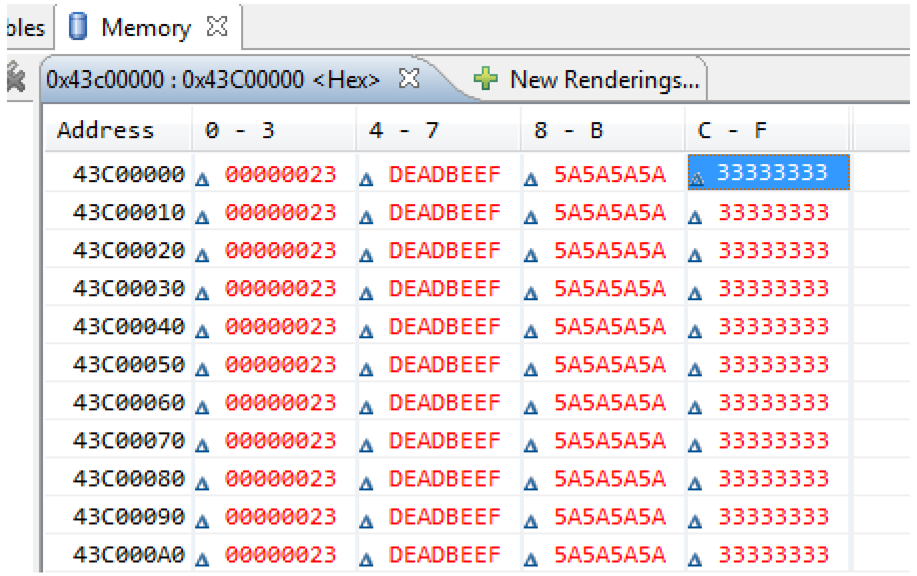


Transactions from the PS7 are routed to one of 3 locations dependent on the address. Check this out in your own design.



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_fifo_mm_s_0	S_AXI	Mem0	0x43C1_0000	64K	0x43C1_FFFF
axi_iic_0	S_AXI	Reg	0x4160_0000	64K	0x4160_FFFF
radio_tuner_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF

In the debugger : our peripheral in action!



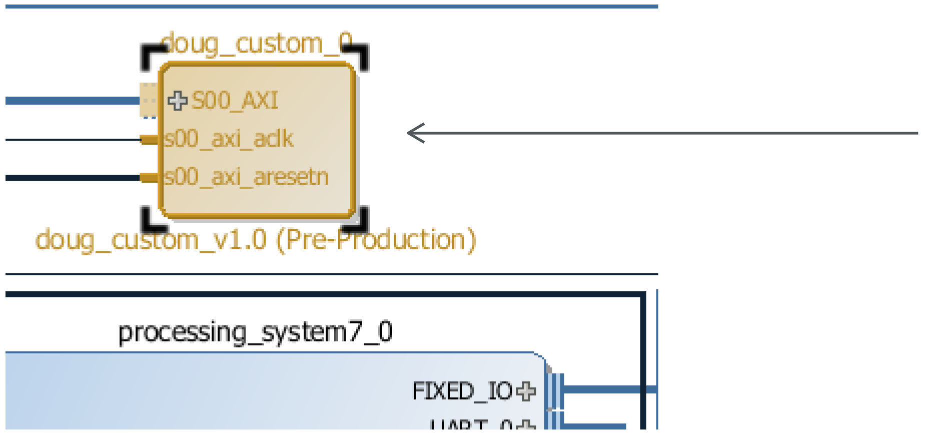
```
Address   0 - 3     4 - 7     8 - B     C - F
43C00000  00000023  DEADBEEF  5A5A5A5A  33333333
43C00010  00000023  DEADBEEF  5A5A5A5A  33333333
43C00020  00000023  DEADBEEF  5A5A5A5A  33333333
43C00030  00000023  DEADBEEF  5A5A5A5A  33333333
43C00040  00000023  DEADBEEF  5A5A5A5A  33333333
43C00050  00000023  DEADBEEF  5A5A5A5A  33333333
43C00060  00000023  DEADBEEF  5A5A5A5A  33333333
43C00070  00000023  DEADBEEF  5A5A5A5A  33333333
43C00080  00000023  DEADBEEF  5A5A5A5A  33333333
43C00090  00000023  DEADBEEF  5A5A5A5A  33333333
43C000A0  00000023  DEADBEEF  5A5A5A5A  33333333
```

4 independent registers which can be read/written

Mirroring of the registers reflects the large address space reserved for the peripheral while it only has 4 real registers.

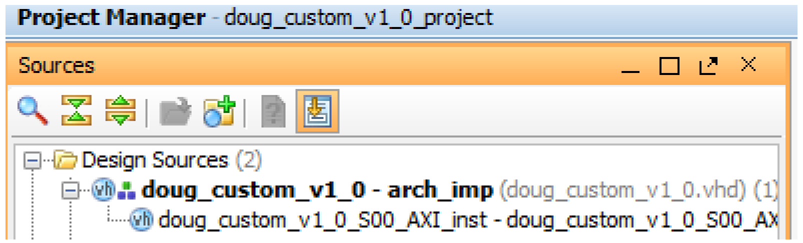
In Class Exercise : Adding some custom Logic

- For demonstration we will…
  - Make it so that reads from one of our 4 locations always reads back 0xDEADBEEF no matter what you write to that location (ADR Base+4)
  - Add a DDS core to our peripheral whose phase increment can be set by writing to the location at the base address of the peripheral



To get started editing the HDL, right click on the block and select “Edit in IP Packager”

Structure of the Peripheral


Project Manager - doug_custom_v1_0_project

Sources

Design Sources (2)

doug_custom_v1_0 - arch_imp (doug_custom_v1_0.vhd) (1)

doug_custom_v1_0_S00_AXI_inst - doug_custom_v1_0_S00_AX

Just a wrapper, not much here

The AXI bus logic, and the 4 registers
We will edit this first

Structure of the Peripheral (Reading)

Useful! Just because the address is set a certain way doesn't mean the processor is reading! This is important when reads have side-effects *like with a FIFO*

```
-- Implement memory mapped register select and read logic generation
-- Slave register read enable is asserted when valid address is available
-- and the slave is ready to accept the read address.
slv_reg_rden <= axi_arready and S_AXI_ARVALID and (not axi_rvalid) ;

process (slv_reg0, slv_reg1, slv_reg2, slv_reg3, axi_araddr, S_AXI_ARESETN, slv_reg_rden)
variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
begin
    -- Address decoding for reading registers
    loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
    case loc_addr is
      when b"00" =>
        reg_data_out <= slv_reg0;
      when b"01" =>
        reg_data_out <= slv_reg1;
      when b"10" =>
        reg_data_out <= slv_reg2;
      when b"11" =>
        reg_data_out <= slv_reg3;
      when others =>
        reg_data_out  <= (others => '0');
    end case;
end process;
```

What value goes to the data bus when the processor reads from a particular address?

Structure of the Peripheral (Writing)

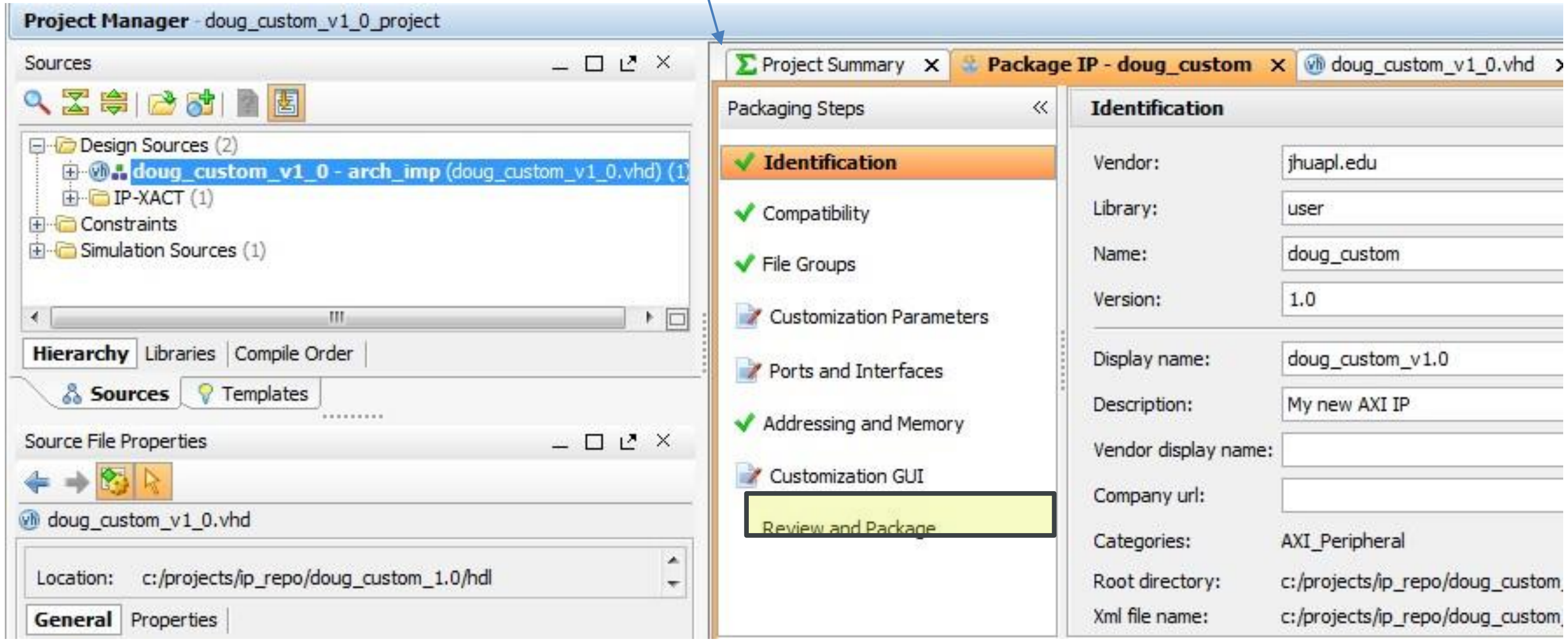
```
-- Implement memory mapped register select and write logic generation
-- The write data is accepted and written to memory mapped registers when
-- axi_awready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted. Write strobes are used to
-- select byte enables of slave registers while writing.
-- These registers are cleared when reset (active low) is applied.
-- Slave register write enable is asserted when valid address and data are available
-- and the slave is ready to accept the write address and write data.
slv_reg_wren <= axi_wready and S_AXI_WVALID and axi_awready and S_AXI_AWVALID ;

process (S_AXI_ACLK)
variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
begin
  if rising_edge(S_AXI_ACLK) then
    if S_AXI_ARESETN = '0' then
      slv_reg0 <= (others => '0');
      slv_reg1 <= (others => '0');
      slv_reg2 <= (others => '0');
      slv_reg3 <= (others => '0');
    else
      loc_addr := axi_awaddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
      if (slv_reg_wren = '1') then
        case loc_addr is
          when b"00" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 0
                slv_reg0(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto…
              end if;
            end loop;
          when b"01" =>
            for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
              if ( S_AXI_WSTRB(byte_index) = '1' ) then
                -- Respective byte enables are asserted as per write strobes
                -- slave registor 1
                slv_reg1(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto..
              end if;
```

What happens when the processor writes to a particular address? Signals slv_reg0-3 get set to the contents of the data bus.

Re-packaging New Peripheral

- At this point, we've modified all of the VHDL, but we want need to "package" the IP again

Just like earlier in the class when we were packaging the lowlevel dac interface, we want to turn everything to green checks



Project Manager - doug_custom_v1_0_project

Sources

- Design Sources (2)
  - doug_custom_v1_0 - arch_imp (doug_custom_v1_0.vhd) (1)
  - IP-XACT (1)
- Constraints
- Simulation Sources (1)

doug_custom_v1_0.vhd

Location: c:/projects/ip_repo/doug_custom_1.0/hdl

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- Customization Parameters
- Ports and Interfaces
- ✓ Addressing and Memory
- Customization GUI
- Review and Package

Identification

Vendor:	jhuapl.edu
Library:	user
Name:	doug_custom
Version:	1.0
Display name:	doug_custom_v1.0
Description:	My new AXI IP
Vendor display name:	
Company url:	
Categories:	AXI_Peripheral
Root directory:	c:/projects/ip_repo/doug_custom
Xml file name:	c:/projects/ip_repo/doug_custom