National Technical University of Athens

Microprocessors and Digital Systems Laboratory

Why and how SoC (FPGA) components communicate ?

Introduction to AXI protocol

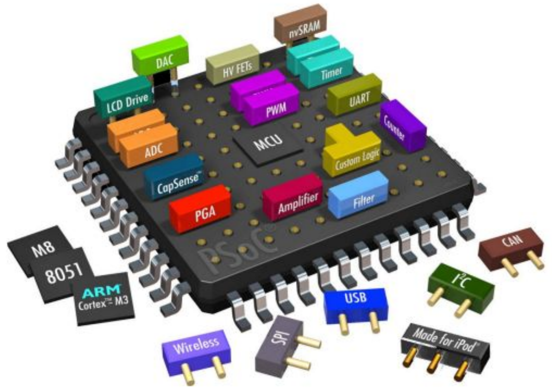
DVLSI 2025

Ilias Papalamprou – ipapalambrou@microlab.ntua.gr

What is SoC-FPGA ?

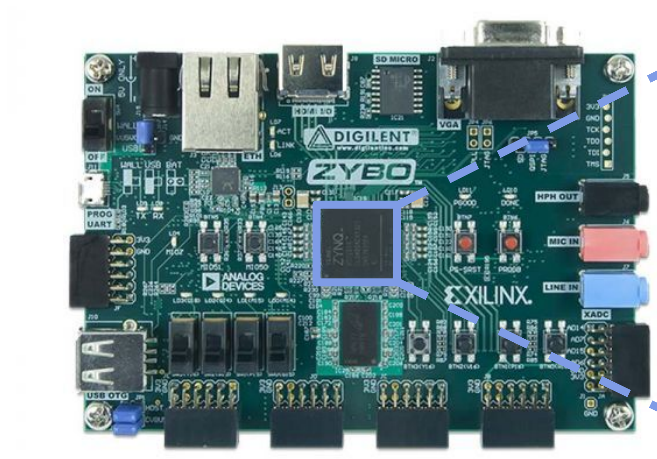
A **complex IC** that integrates digital, analog, radio frequency components (i.e. processor, hardware accelerators, ADCs/DACs, memory, interfaces and interconnection, peripherals etc.) **all in a single chip**

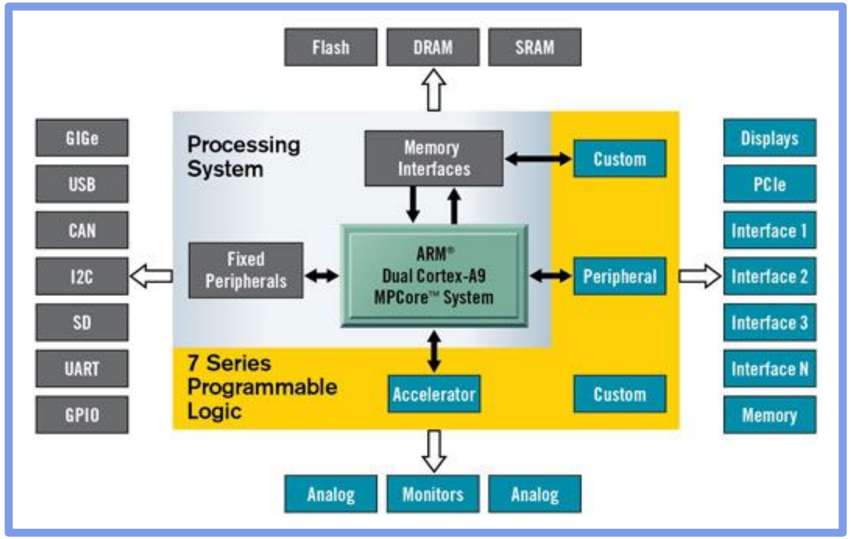
- Lower communication overhead
- Improved size, power consumption
- Efficient HW/SW co-processing
- Increased productivity



What is SoC-FPGA ?

A **complex IC** that integrates digital, analog, radio frequency components (i.e. processor, hardware accelerators, ADCs/DACs, memory, interfaces and interconnection, peripherals etc.) **all in a single chip**





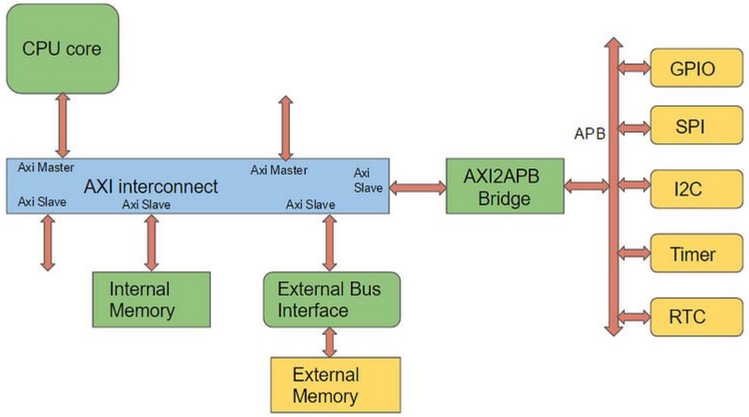
https://commons.wikimedia.org/wiki/File:Xilinx_Zynq-7000_AP_SoC.jpg
https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual

How the SoC components communicate ?

- A bus is a communication system that transfers data between components like:
  - CPU, RAM, Storage Devices, GPU, etc.
- Are needed to ensure “correct” communication
- Otherwise it would lead to unwanted interruption

★ Solution for SoCs:

Advanced eXtensible Interface - ***AXI***

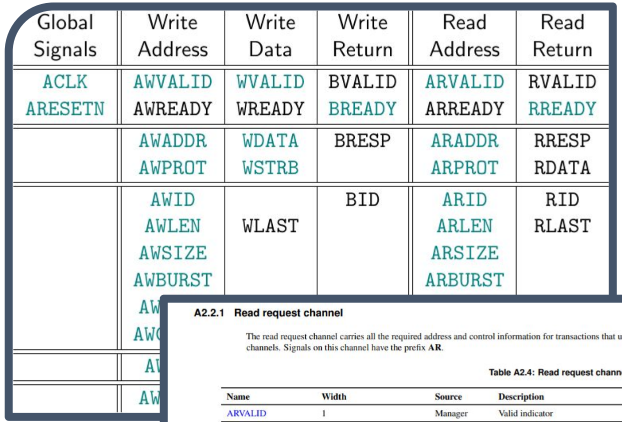


AXI Protocol

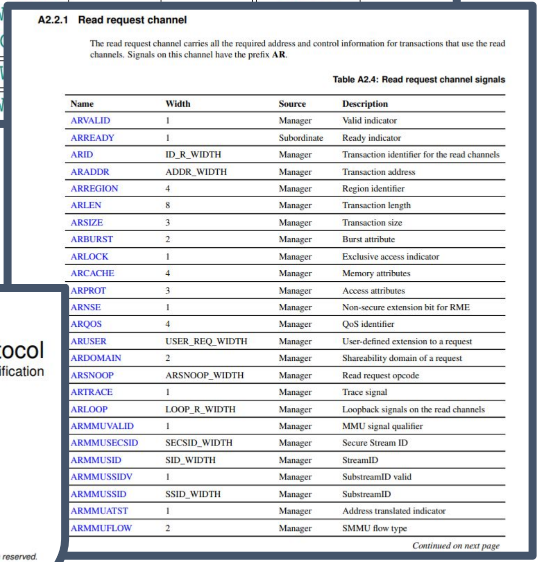
Different AXI variants are available:

- AXI4-Full
- **AXI4-Lite** (“Simple” memory map)
- **AXI4-Stream** (Unlimited data)

Protocol variant is selected based on design requirements



Global Signals	Write Address	Write Data	Write Return	Read Address	Read Return
ACLK ARESETN	AWVALID AWREADY	WVALID WREADY	BVALID BREADY	ARVALID ARREADY	RVALID RREADY
	AWADDR AWPROT	WDATA WSTRB	BRESP	ARADDR ARPROT	RRESP RDATA
	AWID AWLEN AWSIZE AWBURST AW AWC	WLAST	BID	ARID ARLEN ARSIZE ARBURST	RID RLAST
	AV				
	AW				



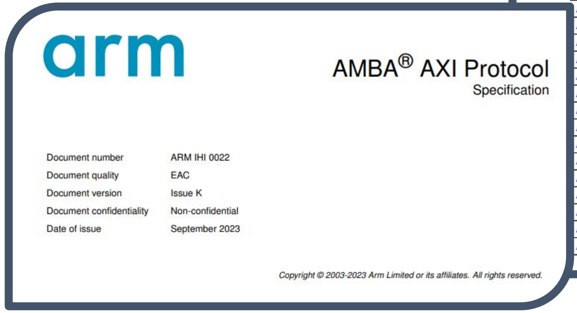
A2.2.1 **Read request channel**

The read request channel carries all the required address and control information for transactions that use the read channels. Signals on this channel have the prefix **AR**.

Table A2.4: Read request channel signals

Name	Width	Source	Description
ARVALID	1	Manager	Valid indicator
ARREADY	1	Subordinate	Ready indicator
ARID	ID_R_WIDTH	Manager	Transaction identifier for the read channels
ARADDR	ADDR_WIDTH	Manager	Transaction address
ARREGION	4	Manager	Region identifier
ARLEN	8	Manager	Transaction length
ARSIZE	3	Manager	Transaction size
ARBURST	2	Manager	Burst attribute
ARLOCK	1	Manager	Exclusive access indicator
ARCACHE	4	Manager	Memory attributes
ARPROT	3	Manager	Access attributes
ARNSE	1	Manager	Non-secure extension bit for RME
ARQOS	4	Manager	QoS identifier
ARUSER	USER_REQ_WIDTH	Manager	User-defined extension to a request
ARDOMAIN	2	Manager	Shareability domain of a request
ARSNOOP	ARSNOOP_WIDTH	Manager	Read request opcode
ARTRACE	1	Manager	Trace signal
ARLOOP	LOOP_R_WIDTH	Manager	Loopback signals on the read channels
ARMMUVALID	1	Manager	MMU signal qualifier
ARMMUSECSID	SECSID_WIDTH	Manager	Secure Stream ID
ARMMUSID	SID_WIDTH	Manager	StreamID
ARMMUSSIDV	1	Manager	SubstreamID valid
ARMMUSSID	SSID_WIDTH	Manager	SubstreamID
ARMMUATST	1	Manager	Address translated indicator
ARMMUFLOW	2	Manager	SMMU flow type

Continued on next page



arm

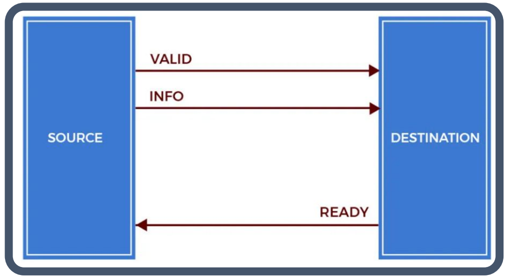
AMBA® AXI Protocol
Specification

Document number	ARM IHI 0022
Document quality	EAC
Document version	Issue K
Document confidentiality	Non-confidential
Date of issue	September 2023

AXI Protocol

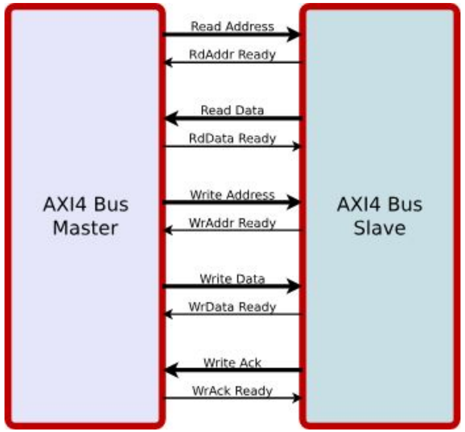
Master – Slave Handshake:

- *TREADY* & *TVALID*: Transaction Initiates



AXI4-Lite Channels:

- Global Signal (Reset, Clock)
- Read Address Channel
- Read Data Channel
- Write Address Channel
- Write Data Channel
- Write Response Channel



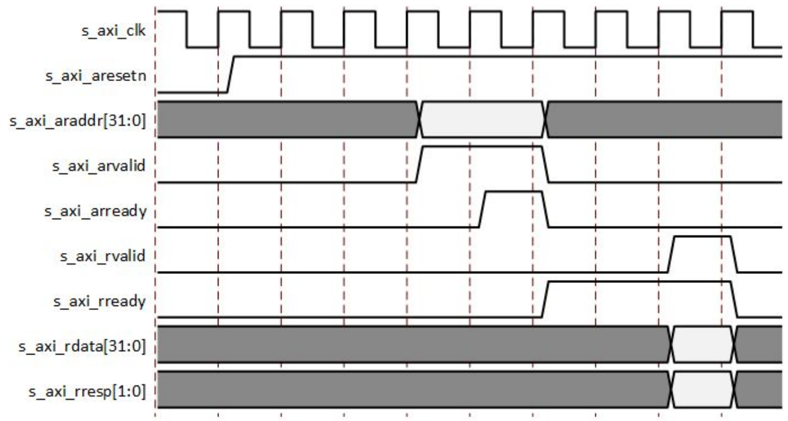
https://www.logic-fruit.com/blog/digital-interfaces/axi-full-axi-lite-interfaces/#:~:text=Five%20channels%20make%20up%20the,Write%20Address%2C%20and%20Write%20Address.&text=Each%20transaction%20burst%20length%20is,width%20as%20the%20data%20bus.

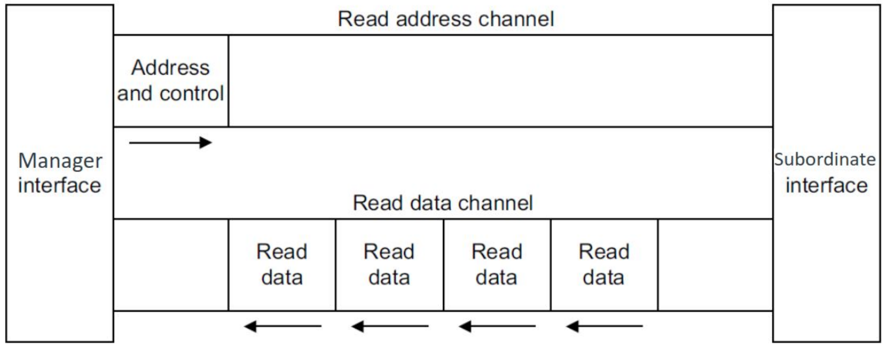
AXI Protocol

Read operation:

- Master sends read request
- Slave then transmits the data to the master

Figure: AXI4-Lite Read Timing Diagram

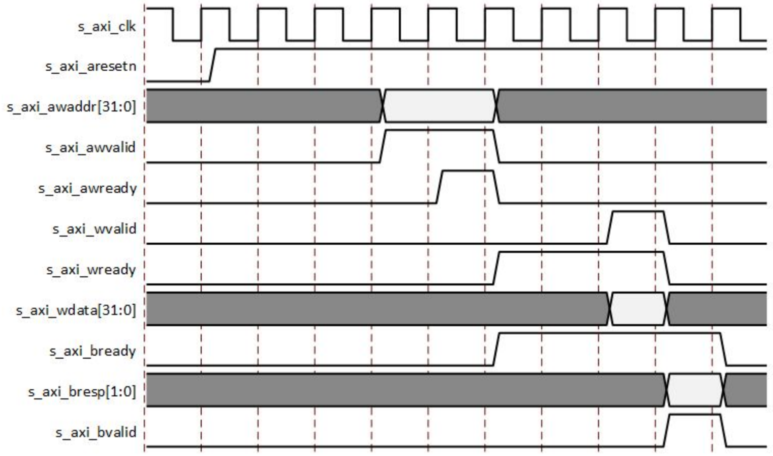


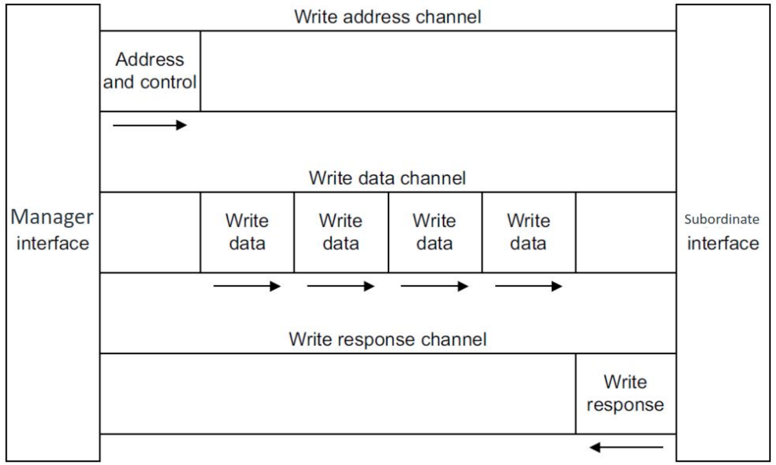


AXI Protocol

Write operation:

- Master sends write request
- Master writes the data to the slave
- Slave informs master if the transfer was successful





Vivado & Vitis Workflow

AXI4-Lite peripheral in practice…

Vivado:

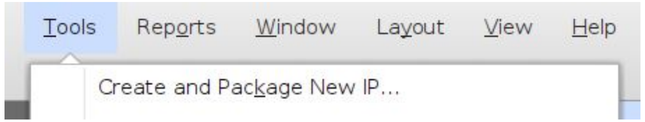
- Create and package new IP
- Generate the AXI4-Lite VHDL template
- Add user logic in the VHDL template
- Package IP
- Create block design & add custom IP

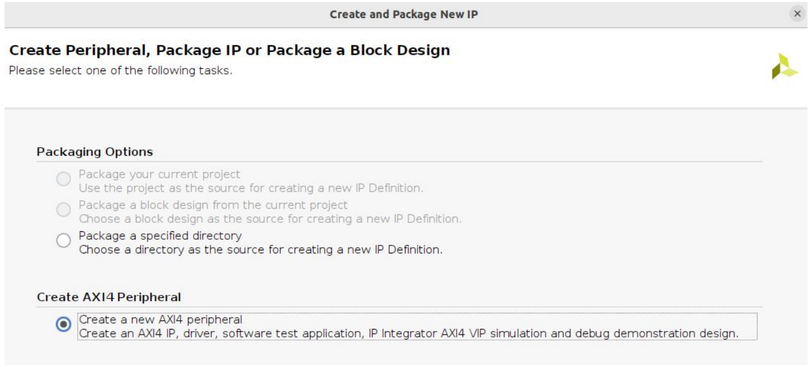
Vitis:

- Host Application (bare metal on ARM)
- *Xil_In32(…)*, *Xil_Out32(…) functions* for writing/reading data to/from AXI4-Lite custom IP.

Vivado Workflow

Step 1: Create new AXI IP




Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

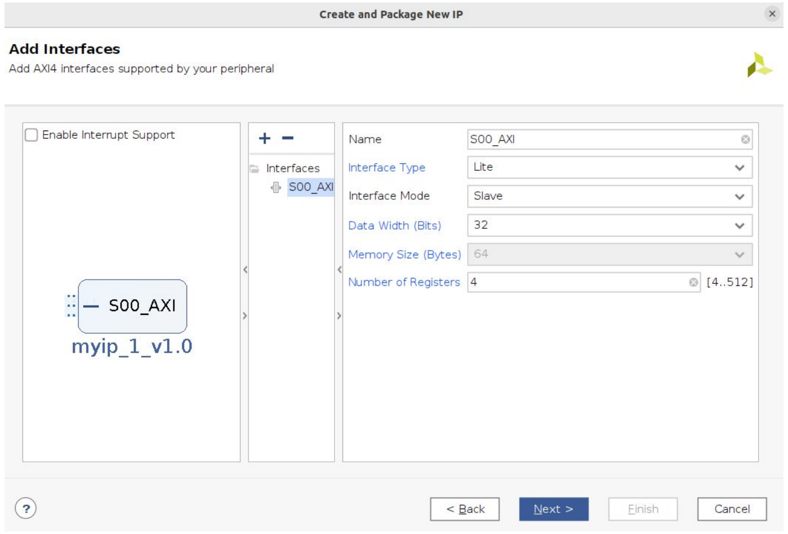
Please select one of the following tasks.

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.


Create and Package New IP

Add Interfaces

Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

Interfaces: S00_AXI

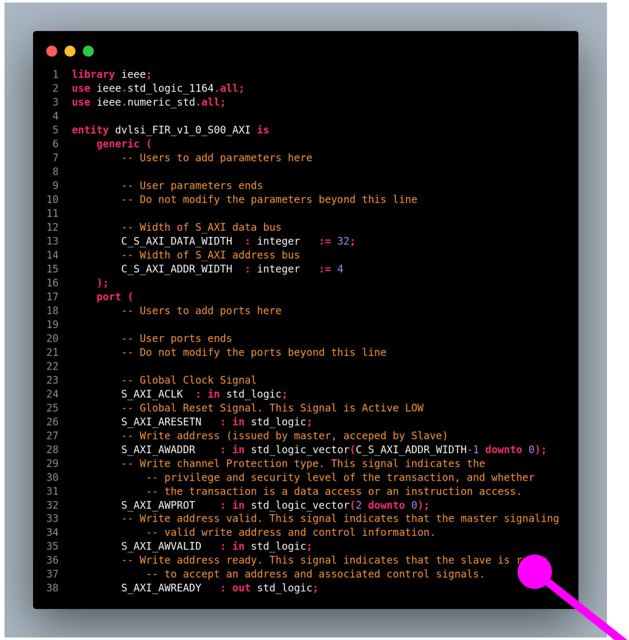
Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	4 [4..512]

AXI4-Lite IP configuration (e..g., number of registers)

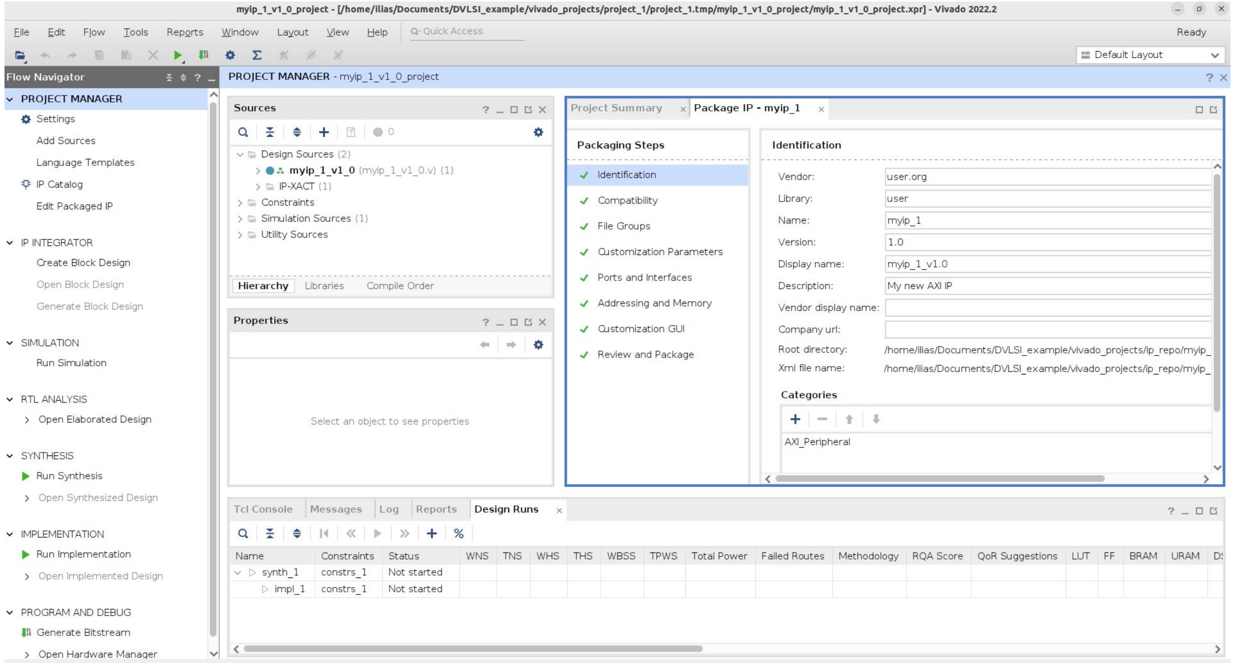
Note: *Set project language to VHDL to generate VHDL sources*

Vivado Workflow

Step 2: Modify the generated VHDL



```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dvlsi_FIR_v1_0_S00_AXI is
    generic (
        -- Users to add parameters here

        -- User parameters ends
        -- Do not modify the parameters beyond this line

        -- Width of S_AXI data bus
        C_S_AXI_DATA_WIDTH  : integer   := 32;
        -- Width of S_AXI address bus
        C_S_AXI_ADDR_WIDTH  : integer   := 4
    );
    port (
        -- Users to add ports here

        -- User ports ends
        -- Do not modify the ports beyond this line

        -- Global Clock Signal
        S_AXI_ACLK  : in std_logic;
        -- Global Reset Signal. This Signal is Active LOW
        S_AXI_ARESETN   : in std_logic;
        -- Write address (issued by master, acceped by Slave)
        S_AXI_AWADDR    : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
        -- Write channel Protection type. This signal indicates the
            -- privilege and security level of the transaction, and whether
            -- the transaction is a data access or an instruction access.
        S_AXI_AWPROT    : in std_logic_vector(2 downto 0);
        -- Write address valid. This signal indicates that the master signaling
            -- valid write address and control information.
        S_AXI_AWVALID   : in std_logic;
        -- Write address ready. This signal indicates that the slave is r
            -- to accept an address and associated control signals.
        S_AXI_AWREADY   : out std_logic;
```



myip_1_v1_0_project - [/home/ilias/Documents/DVLSI_example/vivado_projects/project_1/project_1.tmp/myip_1_v1_0_project/myip_1_v1_0_project.xpr] - Vivado 2022.2

PROJECT MANAGER - myip_1_v1_0_project

Sources

- Design Sources (2)
  - myip_1_v1_0 (myip_1_v1_0.v) (1)
  - IP-XACT (1)
- Constraints
- Simulation Sources (1)
- Utility Sources

Select an object to see properties

Packaging Steps

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

Identification

Vendor:	user.org
Library:	user
Name:	myip_1
Version:	1.0
Display name:	myip_1_v1.0
Description:	My new AXI IP
Vendor display name:	
Company url:	
Root directory:	/home/ilias/Documents/DVLSI_example/vivado_projects/ip_repo/myip_
Xml file name:	/home/ilias/Documents/DVLSI_example/vivado_projects/ip_repo/myip_

Categories

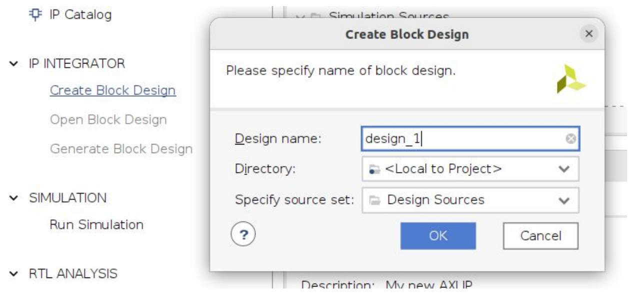
AXI_Peripheral

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	D
synth_1	constrs_1	Not started																
impl_1	constrs_1	Not started																

In this generated VHDL file from Vivado, you add your custom logic and make the appropriate changes to connect them with AXI-4 Lite Registers…

Vivado Workflow

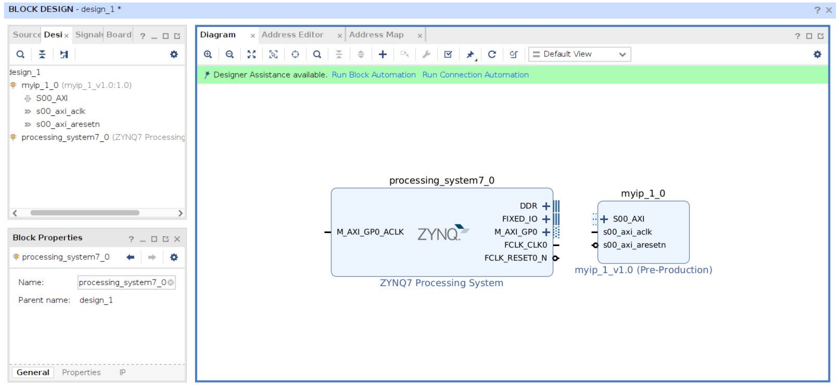
Step 3: Package IP & Integration in Block Design (BD)

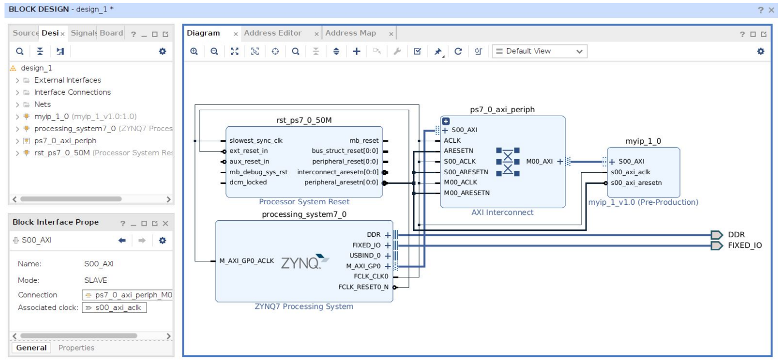


Create Block Design

Please specify name of block design.

Design name:	design_1
Directory:	<Local to Project>
Specify source set:	Design Sources

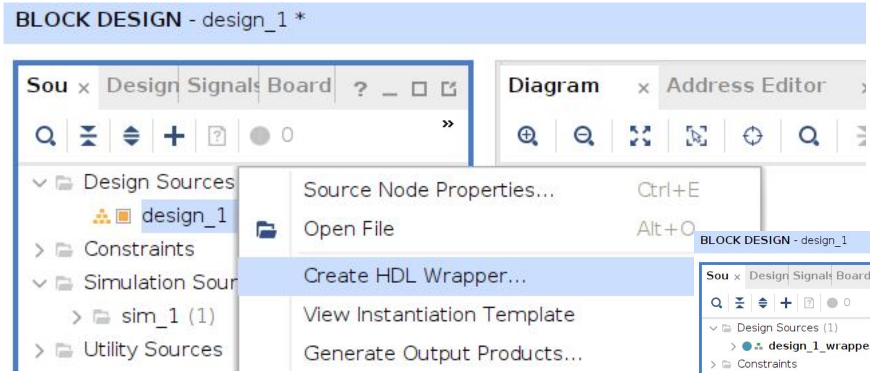


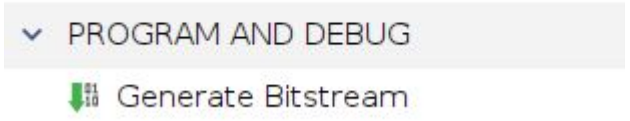


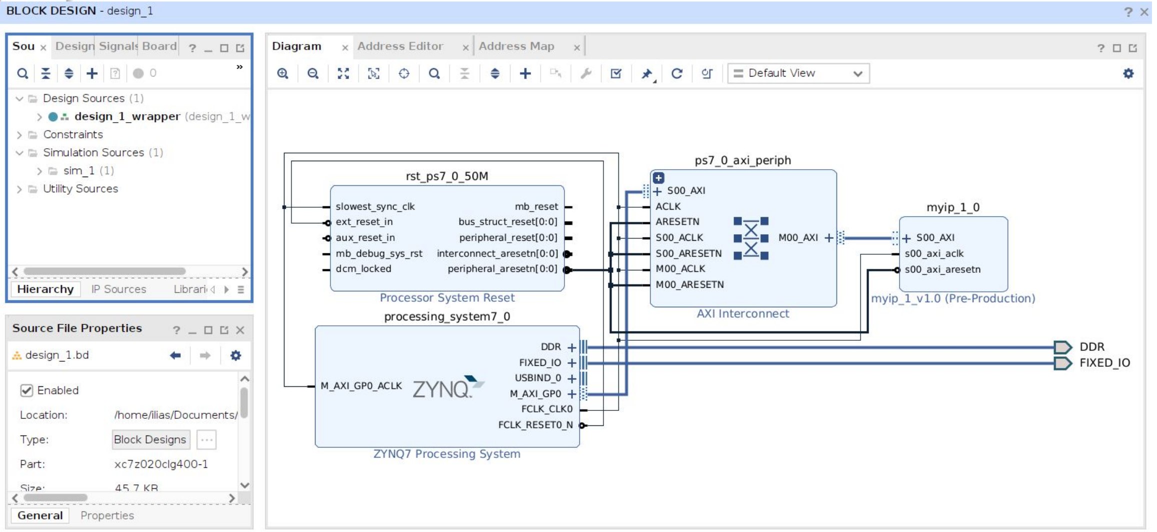
Add in the BD the ZYNQ Processing system & the custom IP

Vivado Workflow

Step 4: Validate BD, HDL Wrapper, Bitstream







Enabled

Location: /home/ilias/Documents/

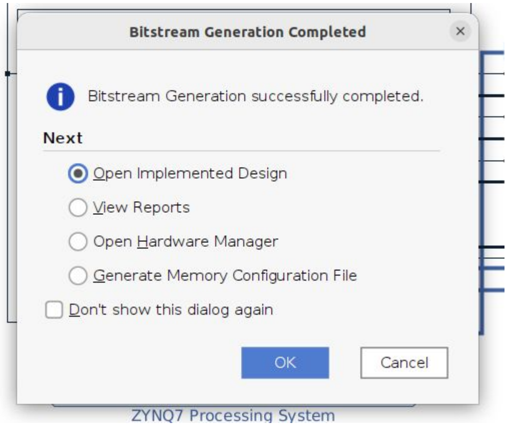
Type: Block Designs

Part: xc7z020clg400-1

Size: 45.7 KB

Vivado Workflow

Step 5: Export .xsa (hardware) & Launch Vitis



Bitstream Generation Completed

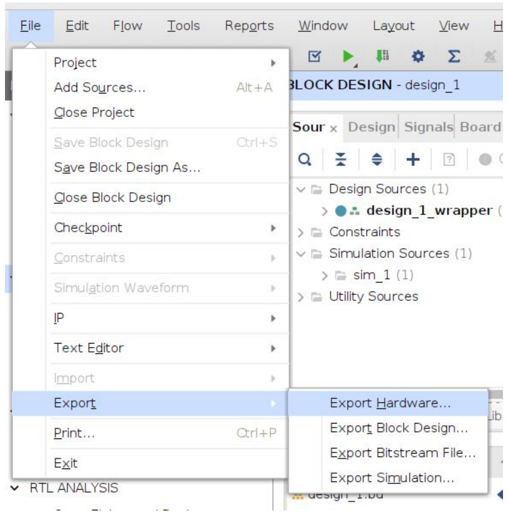
Bitstream Generation successfully completed.

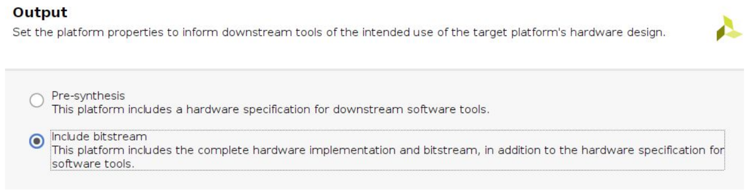
Next

[✓ Open Implemented Design]
View Reports
Open Hardware Manager
Generate Memory Configuration File

☐ Don't show this dialog again

OK   Cancel



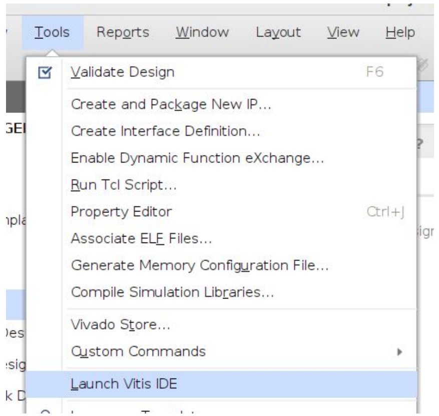


Output

Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.

Pre-synthesis
This platform includes a hardware specification for downstream software tools.

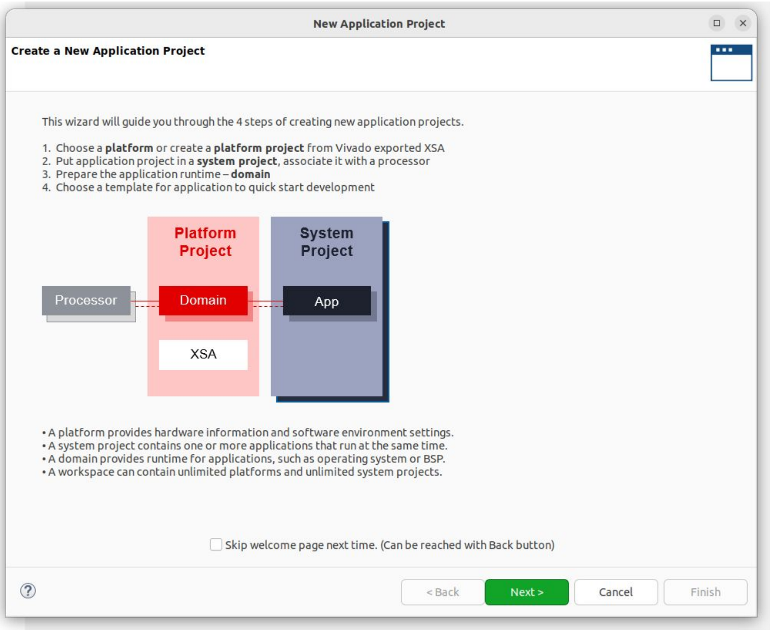
[✓ Include bitstream]
This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.


Tools
Reports
Window
Layout
View
Help
Validate Design F6
Create and Package New IP...
Create Interface Definition...
Enable Dynamic Function eXchange...
Run Tcl Script...
Property Editor Ctrl+J
Associate ELF Files...
Generate Memory Configuration File...
Compile Simulation Libraries...
Vivado Store...
Custom Commands
Launch Vitis IDE

After exporting the .xsa file, the Vivado part is complete

Vitis Workflow

Step 6: Application Project


New Application Project

Create a New Application Project

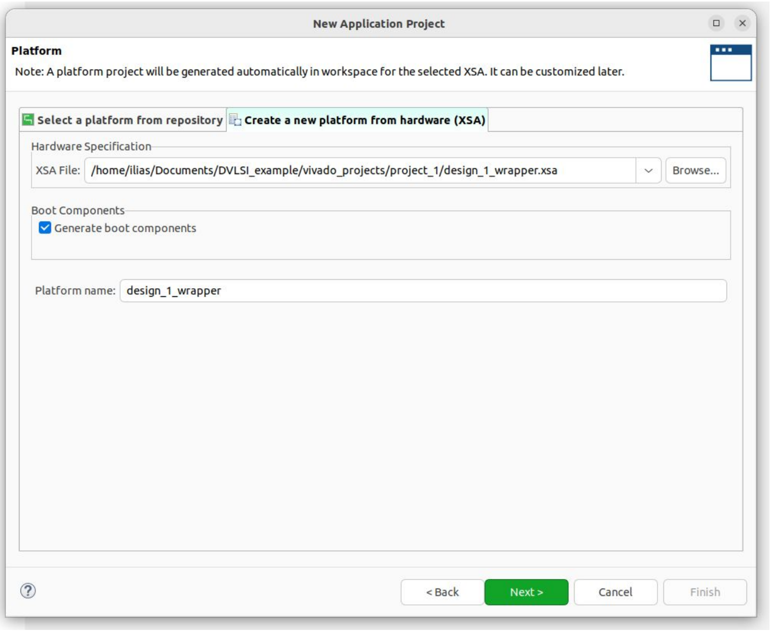
This wizard will guide you through the 4 steps of creating new application projects.

1. Choose a **platform** or create a **platform project** from Vivado exported XSA
2. Put application project in a **system project**, associate it with a processor
3. Prepare the application runtime – **domain**
4. Choose a template for application to quick start development

• A platform provides hardware information and software environment settings.
• A system project contains one or more applications that run at the same time.
• A domain provides runtime for applications, such as operating system or BSP.
• A workspace can contain unlimited platforms and unlimited system projects.

☐ Skip welcome page next time. (Can be reached with Back button)

< Back
Next >
Cancel
Finish


New Application Project

Platform

Note: A platform project will be generated automatically in workspace for the selected XSA. It can be customized later.

Select a platform from repository
Create a new platform from hardware (XSA)

Hardware Specification

XSA File: /home/ilias/Documents/DVLSI_example/vivado_projects/project_1/design_1_wrapper.xsa

Boot Components

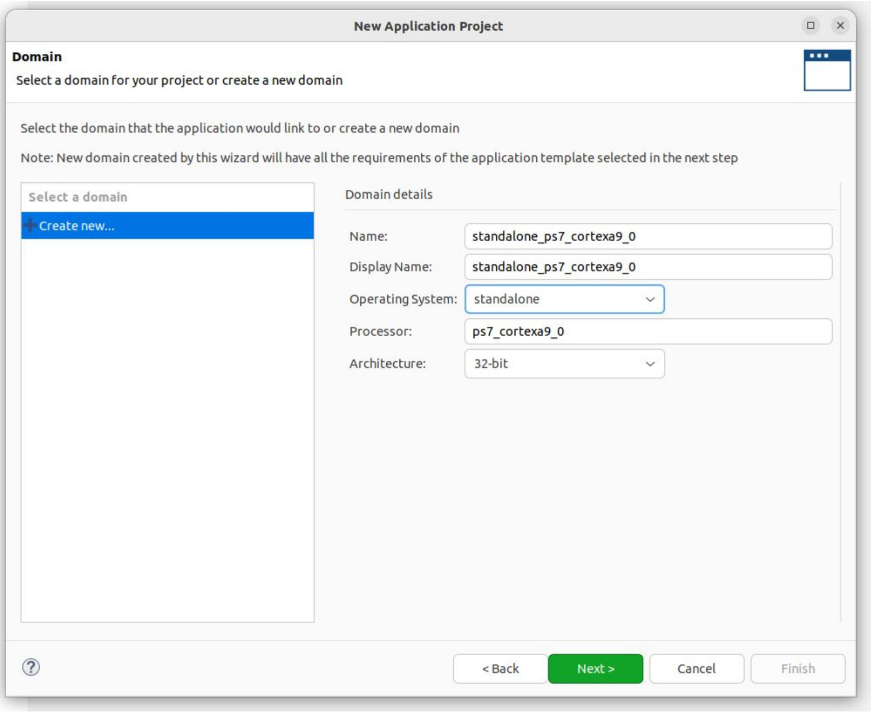
☑ Generate boot components

Platform name: design_1_wrapper

< Back
Next >
Cancel
Finish

Vitis Workflow

Step 6: Generate Application Project



Domain

Select a domain for your project or create a new domain

Select the domain that the application would link to or create a new domain

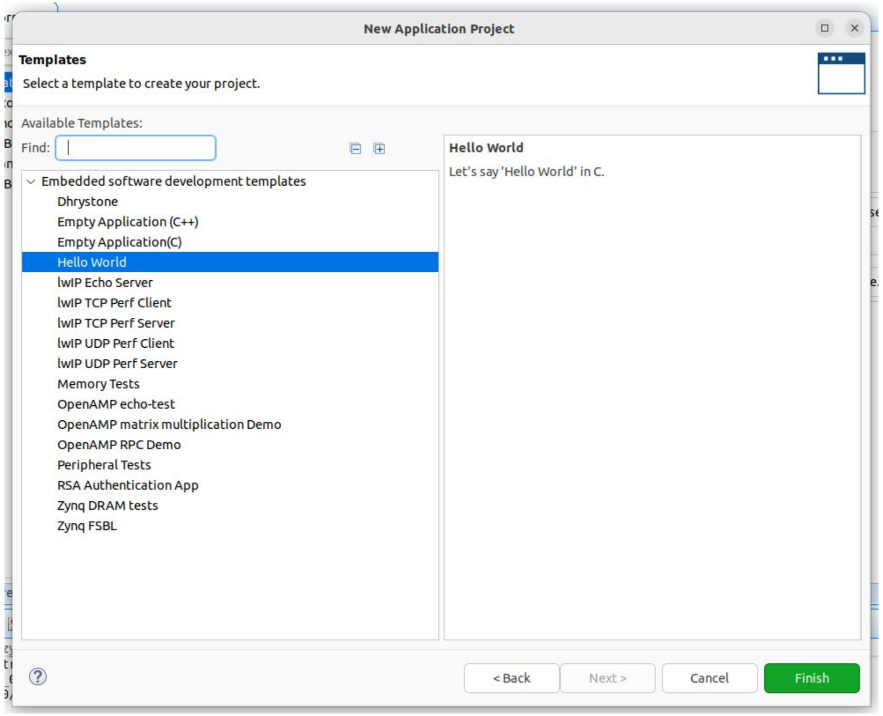
Note: New domain created by this wizard will have all the requirements of the application template selected in the next step

Select a domain

Create new...

Domain details

Name:	standalone_ps7_cortexa9_0
Display Name:	standalone_ps7_cortexa9_0
Operating System:	standalone
Processor:	ps7_cortexa9_0
Architecture:	32-bit



Templates

Select a template to create your project.

Available Templates:

Find:

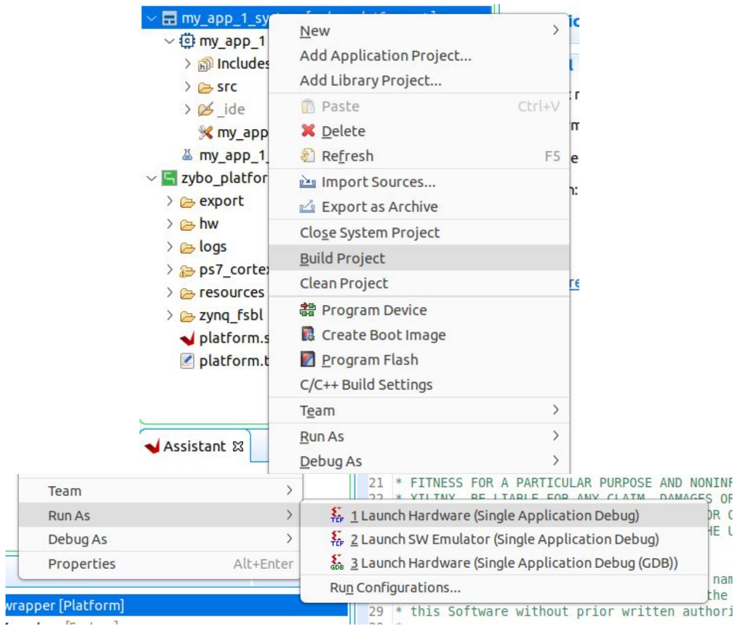
- Embedded software development templates
  - Dhrystone
  - Empty Application (C++)
  - Empty Application(C)
  - Hello World
  - lwIP Echo Server
  - lwIP TCP Perf Client
  - lwIP TCP Perf Server
  - lwIP UDP Perf Client
  - lwIP UDP Perf Server
  - Memory Tests
  - OpenAMP echo-test
  - OpenAMP matrix multiplication Demo
  - OpenAMP RPC Demo
  - Peripheral Tests
  - RSA Authentication App
  - Zynq DRAM tests
  - Zynq FSBL

Hello World

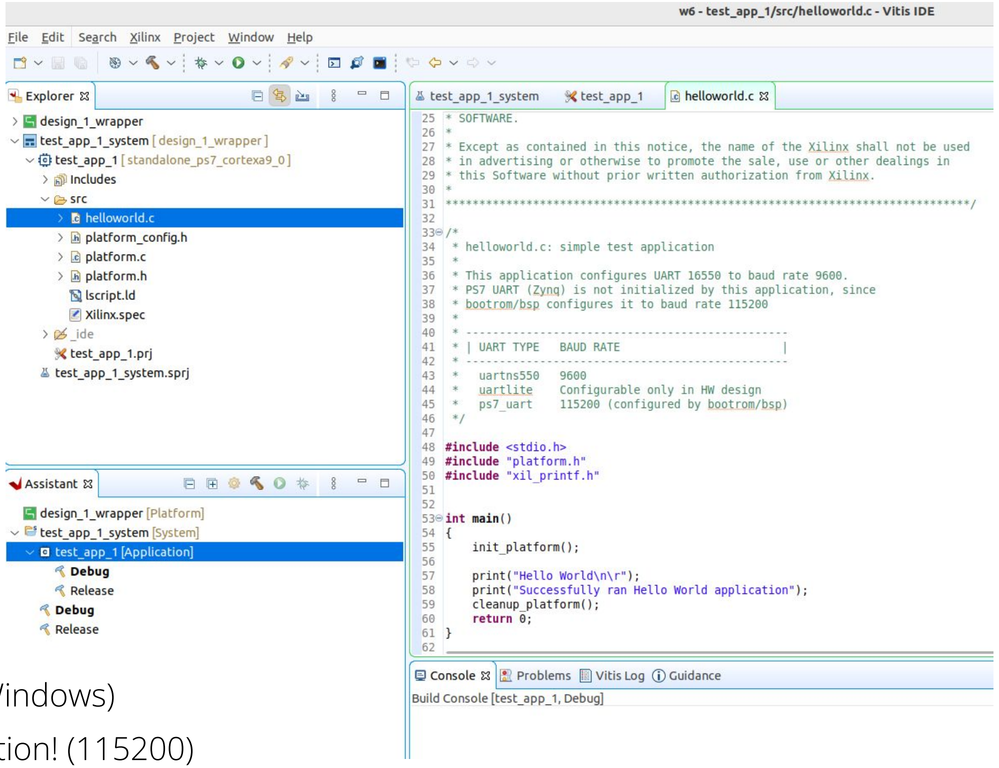
Let's say 'Hello World' in C.

Vitis Workflow

Step 7: Build Project and Program Zybo – JTAG mode



```
* FITNESS FOR A PARTICULAR PURPOSE AND NONINF
* this Software without prior written authori
```



```
 * SOFTWARE.
 *
 * Except as contained in this notice, the name of the Xilinx shall not be used
 * in advertising or otherwise to promote the sale, use or other dealings in
 * this Software without prior written authorization from Xilinx.
 *
******************************************************************************/

/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
    init_platform();

    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    cleanup_platform();
    return 0;
}
```

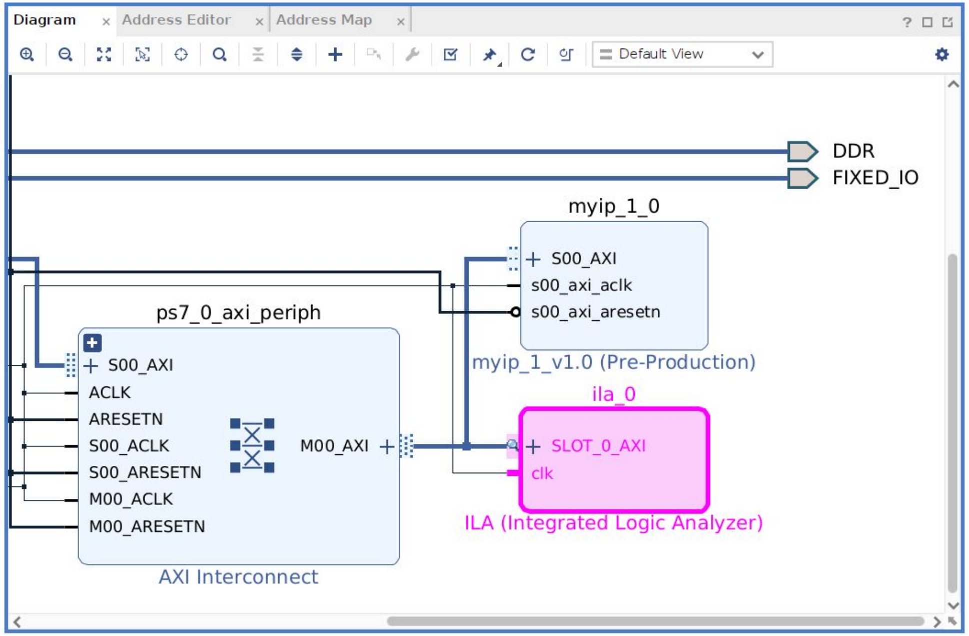
See serial output with: minicom, screen (Linux), putty (Windows)

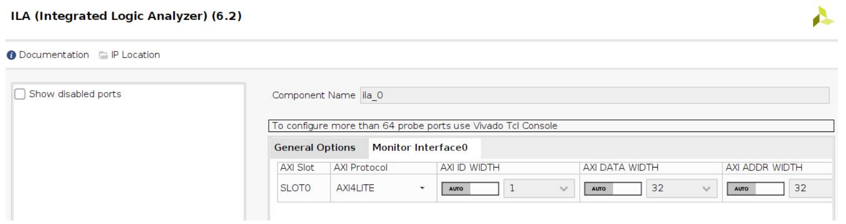
Be sure to set the correct baud rate in uart communication! (115200)

Backup Slides

Vivado Workflow

Advanced Debugging: Integrated Logic Analyzer (ILA)





ILA (Integrated Logic Analyzer) (6.2)

Documentation IP Location

Show disabled ports

Component Name: ila_0

To configure more than 64 probe ports use Vivado Tcl Console

General Options | Monitor Interface0

AXI Slot	AXI Protocol	AXI ID WIDTH	AXI DATA WIDTH	AXI ADDR WIDTH
SLOT0	AXI4LITE	AUTO 1	AUTO 32	AUTO 32

ILA is a way to capture and monitor live signals on the design running on the FPGA.

Results from ILA are viewed in the *Hardware Manager* (in Vivado)

Setup is required for correct trigger (*Hint: What is the basic handshake rule in AXI?*)

References / Useful links

- https://docs.amd.com/r/en-US/ug1165-zynq-embedded-design-tutorial/Example-11-Creating-Peripheral-IP

- https://docs.amd.com/r/en-US/ug1165-zynq-embedded-design-tutorial/Configuring-the-Zynq-7000-Processing-System-with-Presets-in-Vivado

- https://docs.amd.com/r/en-US/oslib_rm/Register-IO-interfacing-APIs

- https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_1/ug995-vivado-ip-subsystems-tutorial.pdf