



EFOP-3.4.3-16-2016-00009

A felsőfokú oktatás minőségének és hozzáférhetőségének
együttes javítása a Pannon Egyetemen

EMBEDDED SYSTEM DEVELOPMENT (MISAM154R)



Created by Zsolt Voroshazi, PhD
voroshazi.zsolt@virt.uni-pannon.hu

Updated: 20 Nov. 2020.

SZÉCHENYI 2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Strukturális
és Beruházási Alapok



BEFEKTETÉS A JÖVŐBE

5. VIVADO – EMBEDDED SYSTEM

Adding GPIO peripherals to BSB from IP Catalog

SZÉCHENYI 2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Strukturális
és Beruházási Alapok



BEFEKTETÉS A JÖVŐBE

Topics covered

1. Introduction – Embedded Systems
2. FPGAs, Digilent ZyBo development platform
3. Embedded System - Firmware development environment (Xilinx Vivado – „EDK” Embedded Development)
4. Embedded System - Software development environment (Xilinx VITIS – „SDK”)
5. Embedded Base System Build (and Board Bring-Up)
- 6. Adding GPIO Peripherals (from IP database) to BSB**
7. Adding Custom (=own) Peripherals to BSB
8. Development, testing and debugging of software applications – Xilinx VITIS (SDK)
9. Design and Development of Complex IP cores and applications (e.g. camera/video/audio controllers)

Important notes & Tips

- Make sure that the path of the Vivado/VITIS project to be created does NOT contain **accented** letters or "White-space" characters!
- Have permissions on the drive you are working on:
 - If possible, DO NOT work on a network drive!
- The name of the project and source files should NOT start with a number, but they can contain a number! (due to VHDL)
- Use case-sensitive letters consistently within a source file and a project!
- If possible, the name of the project directory, project and source file(s) should be different and refer to their function for easier identification of possible error messages.



XILINX VIVADO DESIGN SUITE

Adding IP cores to the Embedded Base System

SZÉCHENYI  2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Strukturális
és Beruházási Alapok



BEFEKTETÉS A JÖVŐBE

Task

- Vivado – Block Designer
 - Add **IP (Intellectual Property) cores** to the formerly elaborated block design (Embedded Base System) from the IP Catalog,
 - Parameterize IP blocks, set connections, interfaces, address, and external ports (modify **.XDC** if needed),
- VITIS - SDK
 - Customize **compiler** settings,
 - Creating a software application (from pre-defined template)

Main steps to solve the task

- Create a new project based on previous laboratory (04.) by using the **Xilinx Vivado (IPI)** embedded system designer,
 - LAB01 project → Save as... → LAB02_A
- Select and add GPIO peripherals to the base system
- Parameterize and connect them, make external ports
- Overview of the created project,
 - *Implementation and Bitstream generation (.BIT) is now necessary, because PL side will also be configured!*
- Create peripheral „TestApp” software application(s) running on ARM by using the Xilinx VITIS environment (~SDK),
- Verify the operation of the completed embedded system and software application test on **Digilent ZyBo**.



XILINX VIVADO DESIGN SUITE

LAB02_A. PUSH BUTTONS, DIP SWITCHES (GPIOs)

SZÉCHENYI  2020



MAGYARORSZÁG
KORMÁNYA

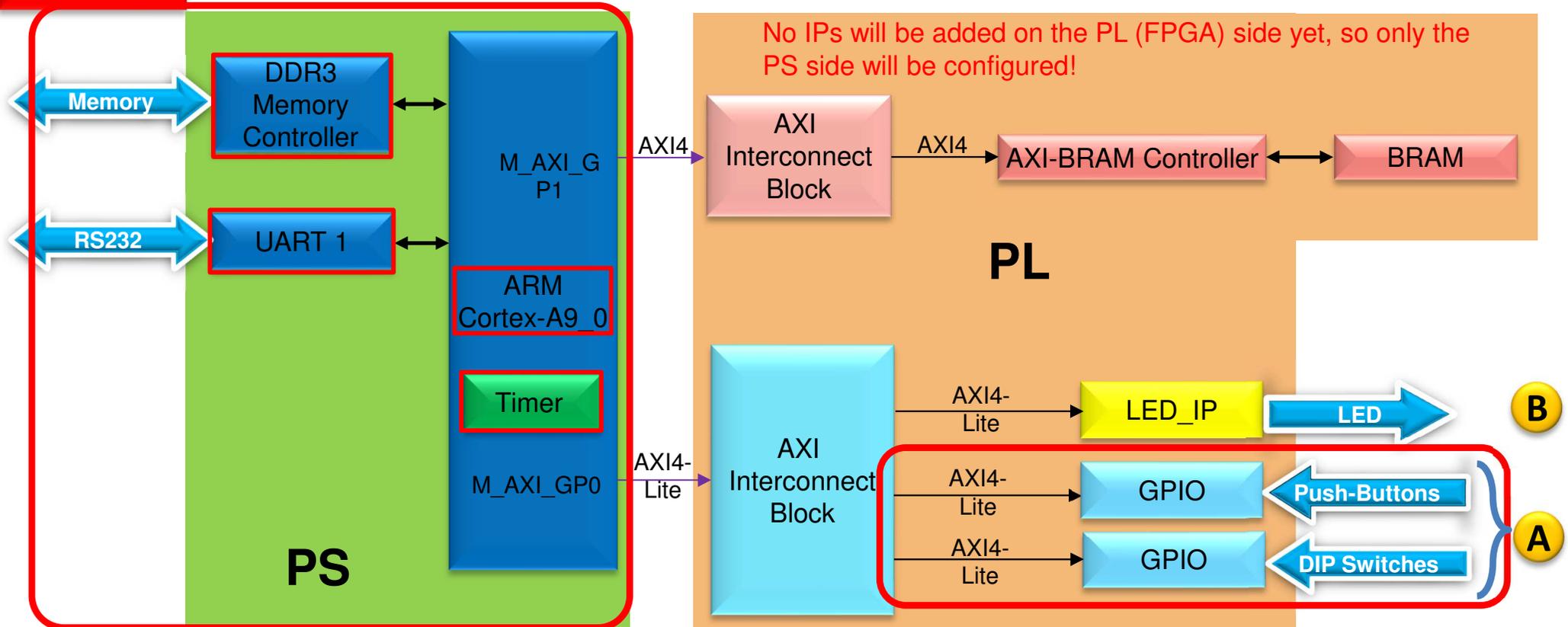
Európai Unió
Európai Strukturális
és Beruházási Alapok



BEFEKTETÉS A JÖVŐBE

Test system to be implemented

LAB02_A



PS side:

- ARM hard-processor (Core0)
- Internal OnChip-RAM controller
- UART1 (serial) interface
- External DDR3 memory controller
- Global Timer

PL (FPGA)

- (A) LAB02_A: GPIO inputs
 - PBSs: Push Button (nyomógomb kezelő)
 - DIPs: Switches (kapcsoló kezelő)
- (B) LAB02_B: GPIO outputs
 - Custom LED controller

Project – Open / Save as...

- Start Vivado
 - Start menu → Programs → Xilinx Design Tools → Vivado 2020.1
- Open the previous project! (LAB01)
 - File → Project → Open... / Open Recent...
 - `<projectdir>/LAB01/<system_name>.xpr` → **Open**
- **File → Project → Save As... → LAB02_A**
 - (This will save former project LAB01 as LAB02_A)

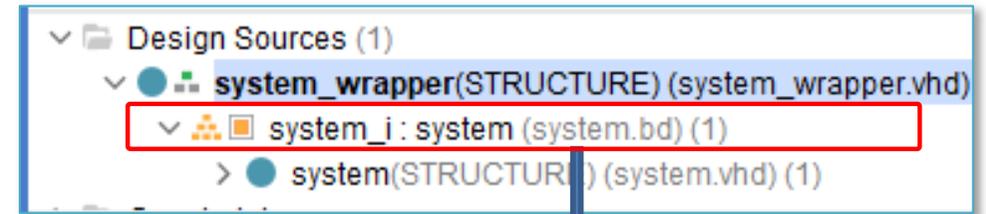
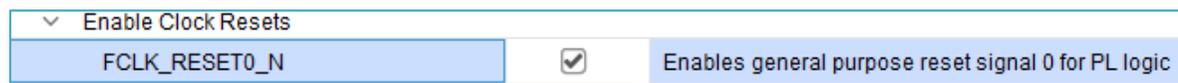
Modify Zynq PS settings

- Open Design Sources → *system.bd* blokk design (double click)

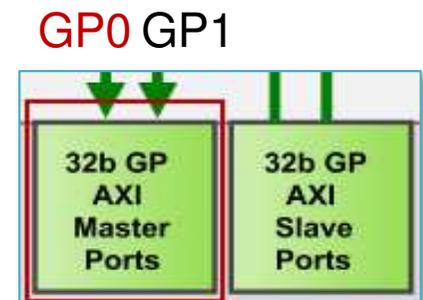
- Open Processing_system7_0

- PS-PL configuration:

- > General → Enable Clock Resets:
enable **FCLK_RESETO_N**



- > AXI Non Secure Enablement: GP Master AXI interface
enable **M_AXI_GP0** port!



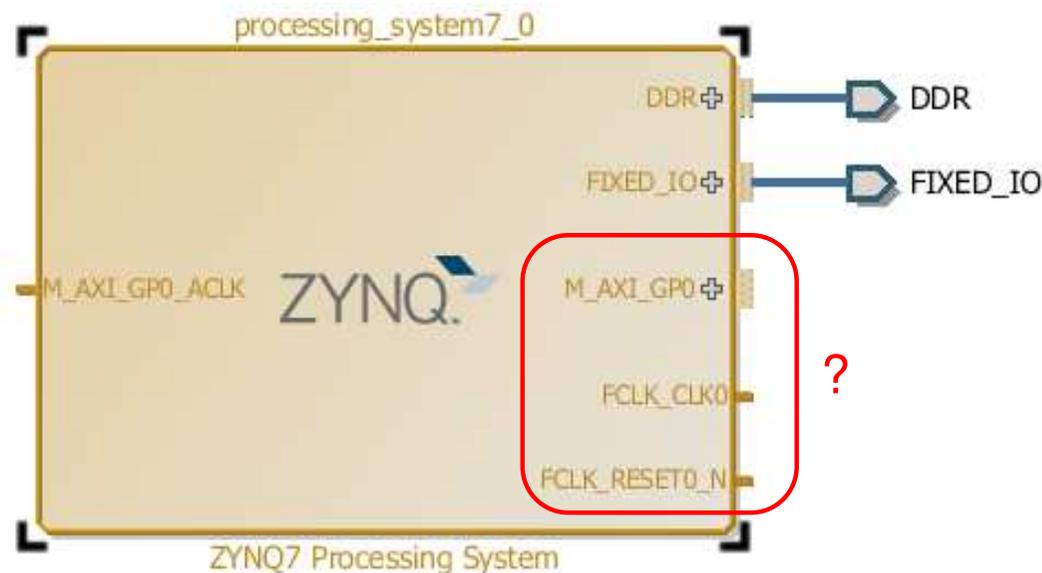
- Clock configuration:

- PL Fabric Clocks: enable **FCLK_CLK0** (100 MHz IO_PLL)



Zynq PS – Block diagram

- Examine, that the previously enabled ports:
 - GPO AXI Master interface,
 - FCLK_CLK0 PL-side clock,
 - FCLK_RESETO_N reset portare visible now?

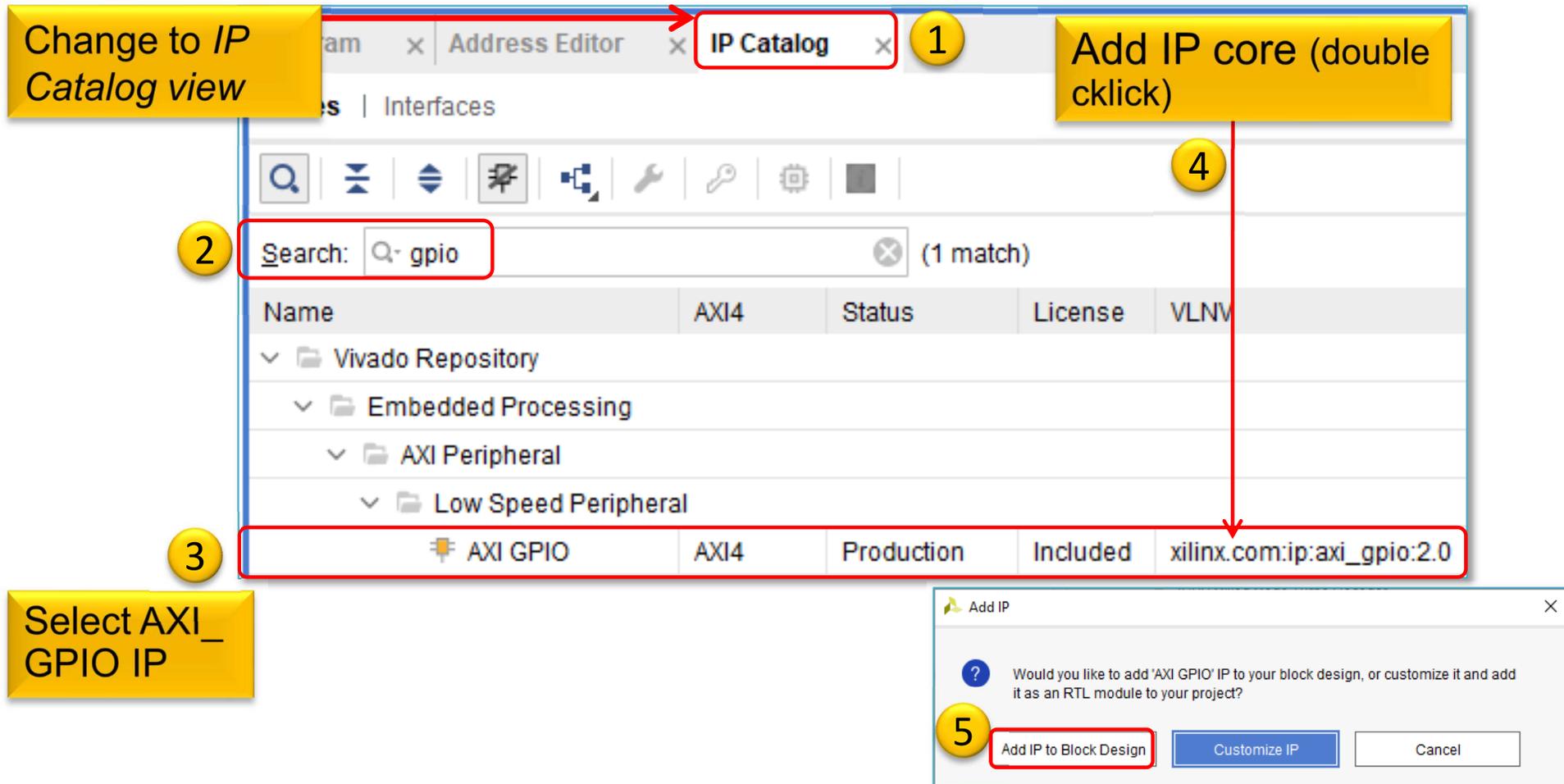


- What do you think about their functionality?

Adding and connecting AXI GPIO peripherals to the PL-side

2x
1x DIP
1x PB

- Adding new IP cores – possible ways:
 - a) Block Diagram View → Add IP  OR
 - b) Open IP Catalog → Select IP → Double click – Add IP to Block Design 
- Add 2 PL side AXI_GPIO peripherals to the processor system



The screenshot shows the Vivado IP Catalog window with the following annotations:

- 1**: IP Catalog window title bar.
- 2**: Search bar containing "gpio" with "(1 match)" next to it.
- 3**: The "AXI GPIO" entry in the table, which is highlighted with a red box.
- 4**: A yellow callout box pointing to the "AXI GPIO" entry with the text "Add IP core (double click)".
- 5**: A dialog box titled "Add IP" with the question "Would you like to add 'AXI GPIO' IP to your block design, or customize it and add it as an RTL module to your project?". The "Add IP to Block Design" button is highlighted with a red box.

Additional yellow callout boxes provide instructions:

- "Change to IP Catalog view" (pointing to the IP Catalog window).
- "Select AXI_GPIO IP" (pointing to the AXI GPIO entry in the table).

Name	AXI4	Status	License	VLNV
Vivado Repository				
Embedded Processing				
AXI Peripheral				
Low Speed Peripheral				
 AXI GPIO	AXI4	Production	Included	xilinx.com:ip:axi_gpio:2.0

Add IP

Would you like to add 'AXI GPIO' IP to your block design, or customize it and add it as an RTL module to your project?

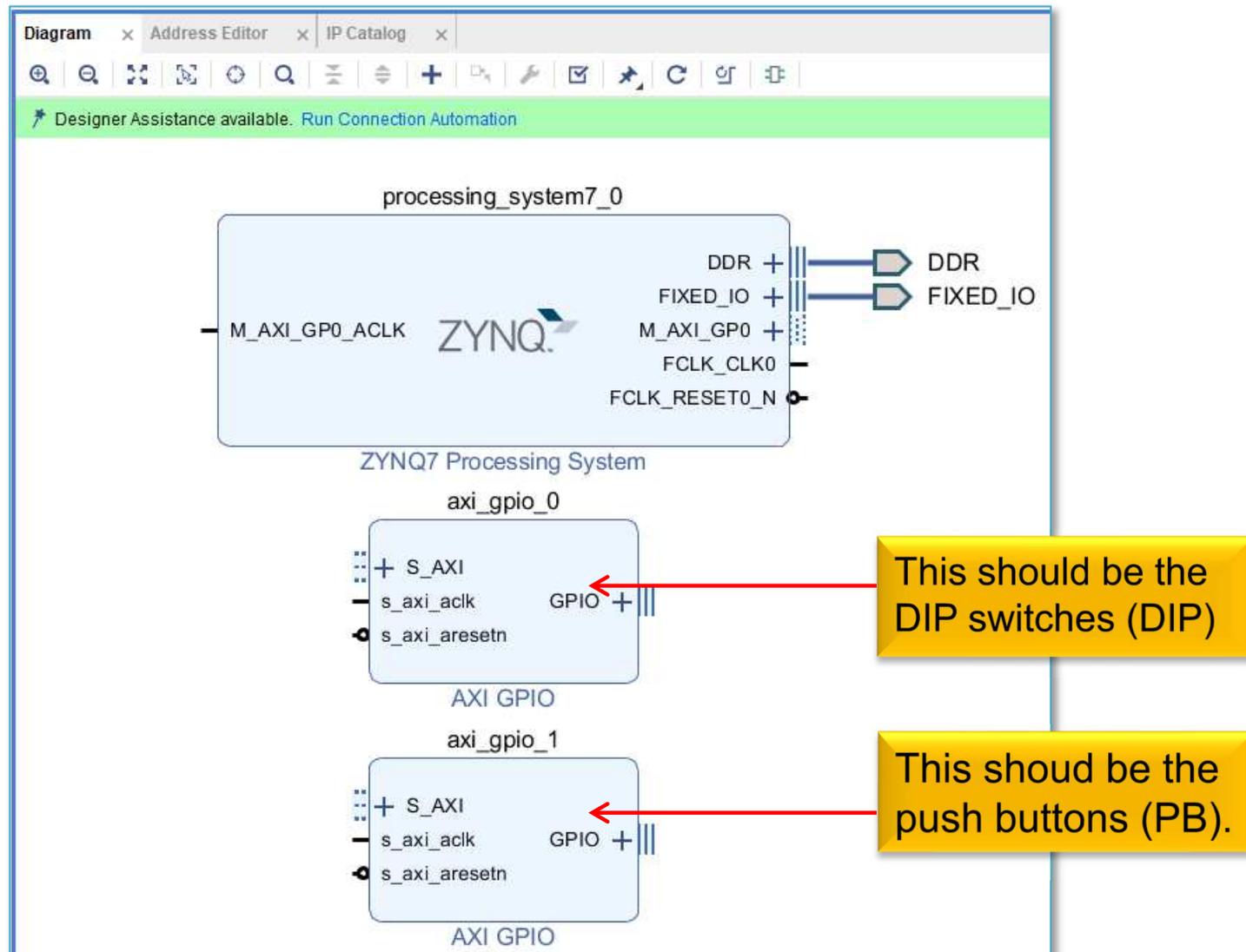
5 Add IP to Block Design

Adding and connecting AXI GPIO peripherals to the PL-side (cont.)

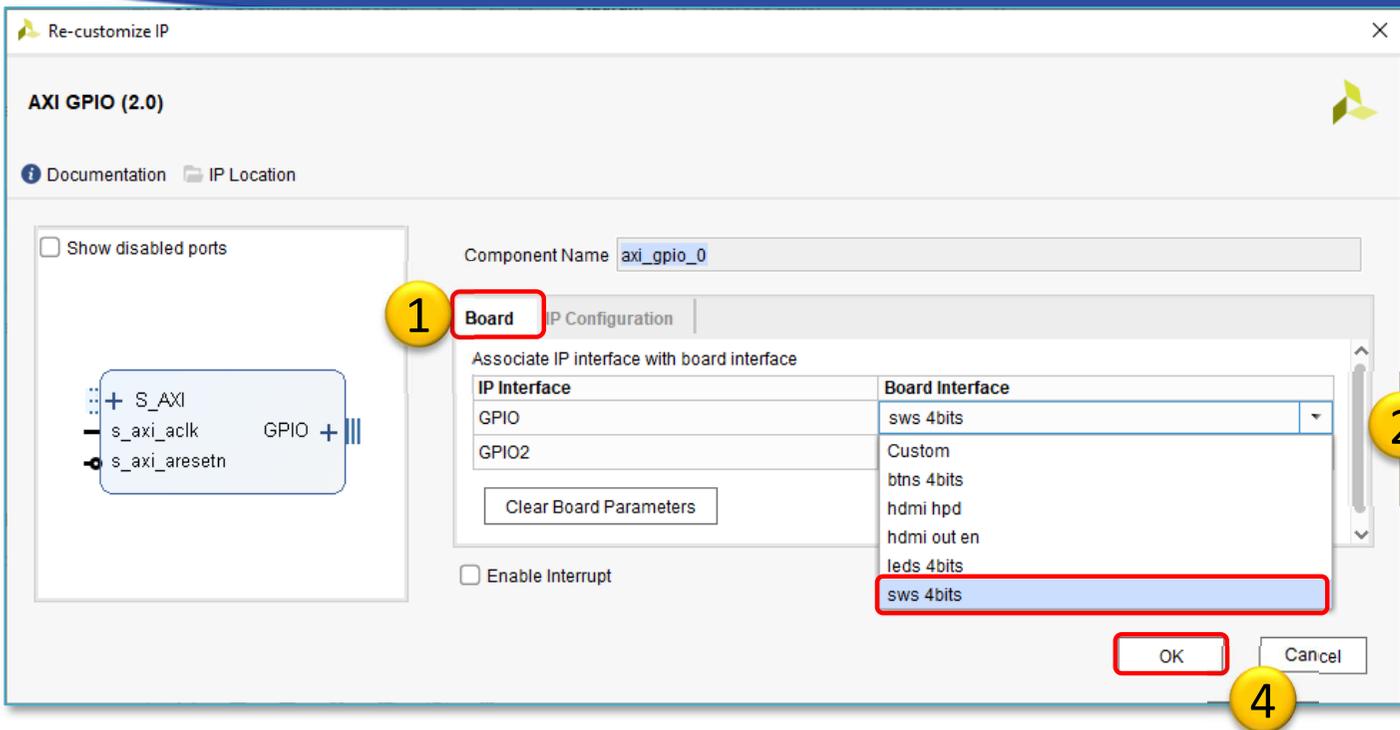
- For each IP modul (e.g. **AXI_GPIO**) the following should be set:
 - a.) *interface connection* between the IP modul and bus system (AXI),
 - b.) **mapping** IP modul to the PS **address space** (Base-High Addresses),
 - c.) assigning I/O ports of IP modules **to external ports**,
 - d.) finally, assigning external ports to **physical FPGA pins** (.XDC editing) – I/O planning.

Adding GPIOs – Block Diagram view

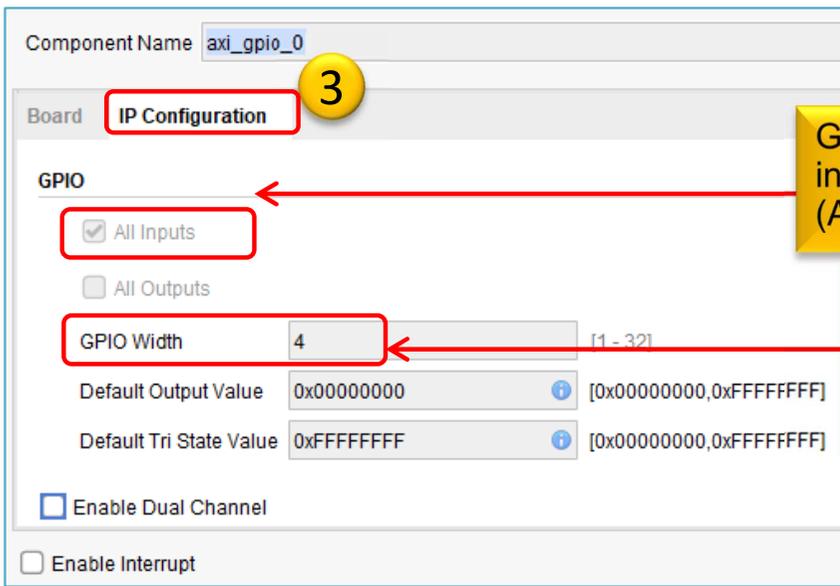
AXI GPIO_0 as DIP switches (DIP) and AXI_GPIO_1 as push buttons (PB).



Set AXI_GPIO peripheral - Dip switches



Board interface: select „sws_4bits”.



GPIO channel only inputs for dip switches (All inputs)

Check GPIO channel width = 4 (because ZyBo has 4 dip switches)

Set AXI_GPIO peripheral - Push buttons

Re-customize IP

AXI_GPIO (2.0)

Documentation IP Location

Show disabled ports

Component Name axi_gpio_1

Board IP Configuration

Associate IP interface with board interface

IP Interface	Board Interface
GPIO	btns 4bits
GPIO2	Custom
	btns 4bits
	hdmi hpd
	hdmi out en
	leds 4bits
	sws 4bits

Clear Board Parameters

Enable Interrupt

OK Cancel

Board interface: select „btns_4bits”.

Component Name axi_gpio_1

Board IP Configuration

GPIO

All Inputs

All Outputs

GPIO Width 4 [1 - 32]

Default Output Value 0x00000000 [0x00000000,0xFFFFFFFF]

Default Tri State Value 0xFFFFFFFF [0x00000000,0xFFFFFFFF]

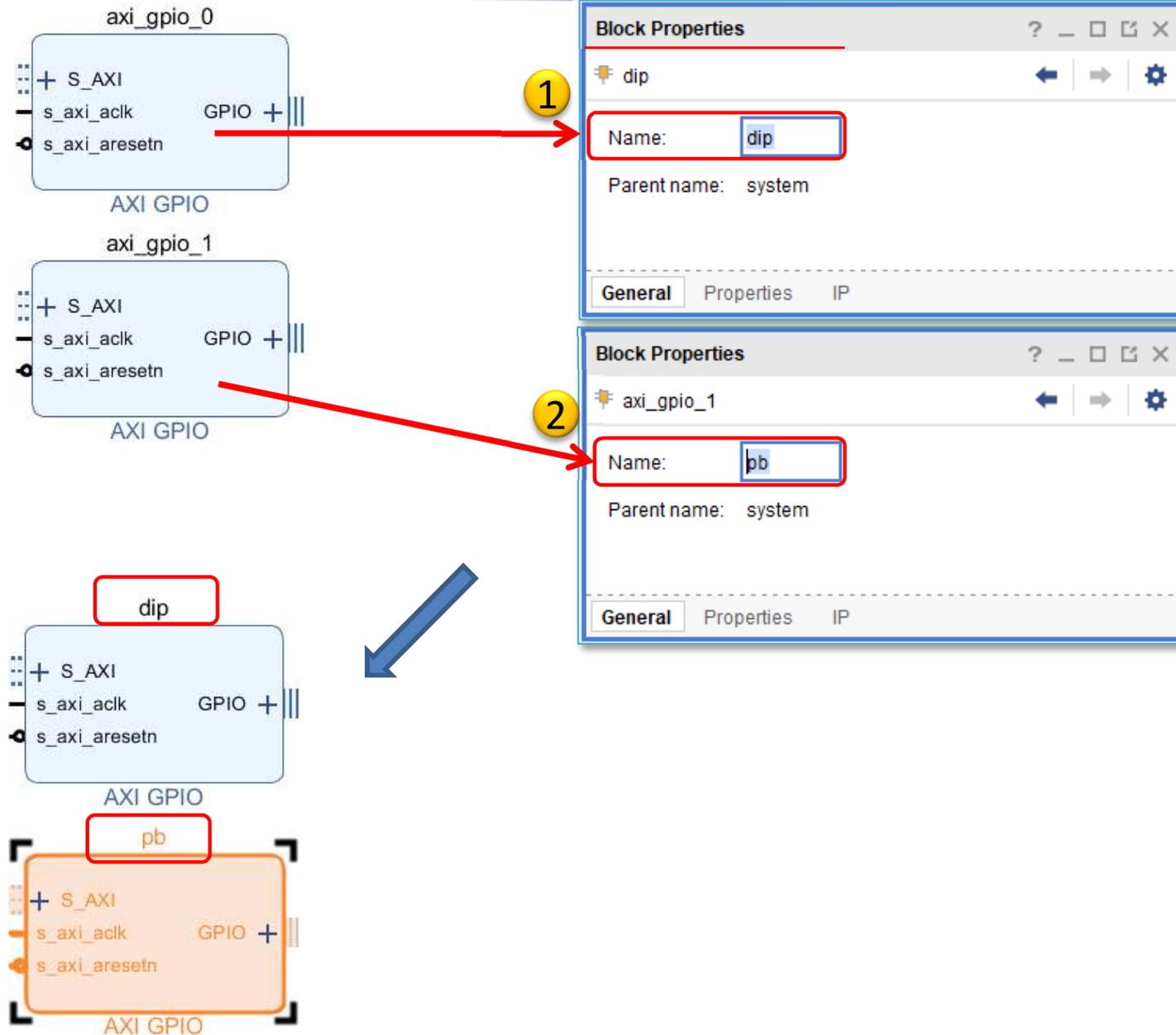
Enable Dual Channel

Enable Interrupt

GPIO channel only inputs for push buttons (All inputs)

Check GPIO channel width = 4 (because ZyBo has 4 Push buttons)

Renaming AXI GPIO peripherals



Block Properties
→ IP instance
name would be
dip and **pb**.

Connecting AXI GPIOs (autorouter)

Diagram → Run Connection Automation

Designer Assistance available. **Run Connection Automation** 1

processing_system7_0

M_AXI_GP0_ACLK ZYNQ M_AXI_GP0 FCLK_CLK0 FCLK_RESET0_N

DDR DDR
FIXED_IO FIXED_IO

DIP → enable S_AXI interface,
PB → enable S_AXI interface

Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

All Automation (2 out of 4 selected)

- dip
 - GPIO
 - S_AXI
- pb
 - GPIO
 - S_AXI

Description

Connect Slave interface (/pb/S_AXI) to a selected Master address space.

Options

Master	/processing_system7_0/M_AXI_GP0
Bridge IP	New AXI Interconnect
Clock source for driving Interconnect IP	Auto
Clock source for Master interface	Auto
Clock source for Slave interface	Auto

OK Cancel

2 3 4

Analysis – Processing_system7_axi_periph

Analyze the parameters of AXI peripheral interface:

- How many slave-, and master interfaces does it have?

Re-customize IP

AXI Interconnect (2.1)

Documentation IP Location

Component Name ps7_0_axi_periph

1 Top Level Settings Slave Interfaces Master Interfaces

Number of Slave Interfaces 1

Number of Master Interfaces 2

Interconnect Optimization Strategy Custom

AXI Interconnect includes IP Interconnect. When the endpoint IPs attach to the interconnect in width, clock or protocol. If a converter IP is inserted into the interconnect, it configures the converter to convert the endpoint IP to the interconnect protocol. To see which conversion IP is used, click the 'expand hierarchy' buttons.

NOTE: Addressing information for the interconnect is defined in the component's configuration file.

Enable Advanced Configuration

2 Top Level Settings Slave Interfaces Master Interfaces

Slave Interface	Enable Register Slice	Enable Data FIFO
S00_AXI	None	None

3 Top Level Settings Slave Interfaces Master Interfaces

Master Interface	Enable Register Slice	Enable Data FIFO
M00_AXI	None	None
M01_AXI	None	None

OK Cancel

Analyzis – Rst_Processing_system7 (Reset)

Analyze the parameters of AXI reset generator:

- How many low-/and high assertion **reset** signals does it have?

Re-customize IP

Processor System Reset (5.0)

Documentation IP Location

Show disabled ports

Component Name:

External Reset

Ext Reset Logic Level (Auto):

Ext Reset Active Width:

Auxillary Reset

Aux Reset Logic Level:

Aux Reset Active Width:

Active High Reset

Bus Structure:

Peripherals:

Active Low Reset

Interconnect:

Peripherals:

slowest_sync_clk

ext_reset_in

aux_reset_in

mb_debug_sys_st

dom_locked

mb_reset

bus_struct_reset[0:0]

peripheral_reset[0:0]

interconnect_aresetn[0:0]

peripheral_aresetn[0:0]

OK Cancel

Set memory addresses – AXI GPIO

- Block Design → select „Address Editor” tab
- Map „unmapped” GPIO IP peripherals into the memory address space (automatically or manually)

0X4000_0000
because GP0
port was
enabled!

a.) Automatic address
generation
(right click →
Auto Assign
Address)

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/processing_system7_0					
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])					
/dip	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
/pb	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF

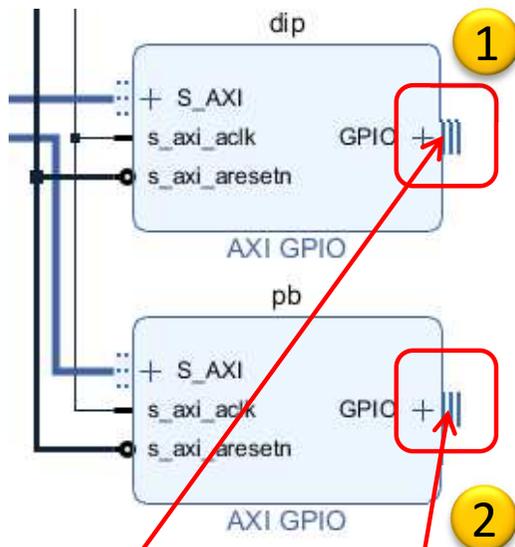
b.) Base address manual set *
pb: 0x4121_0000 (64K)
dip: 0x4120_0000 (64K)

* Address ranges must be limited to 2^n in size and must not overlap!

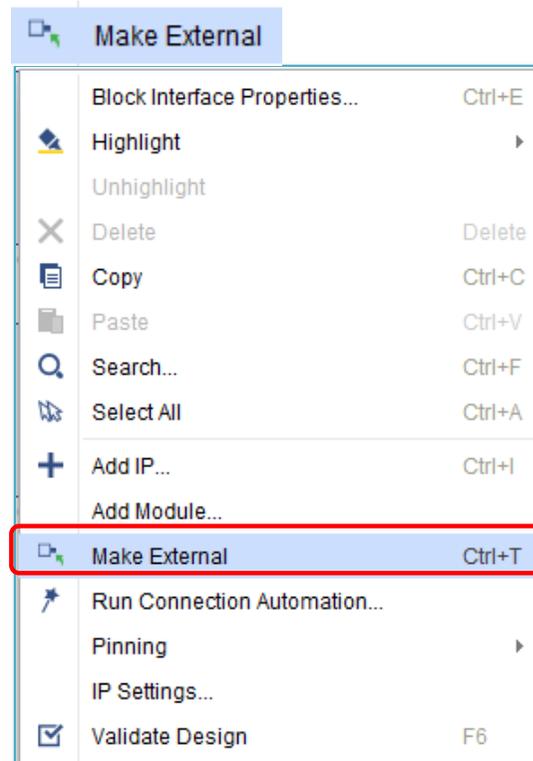
Making external ports – AXI GPIO

The dip and pb GPIO instances must be connected to the physical FPGA pins of the (dip) switches and (pb) pushbuttons on the ZyBo platform:

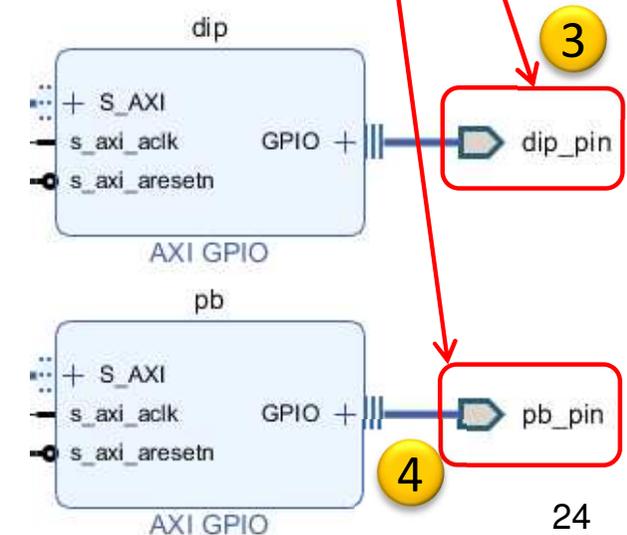
1. The data ports of GPIO instances must also be connected to external FPGA pins,
2. We also define the names of the external ports (e.g. ending in `_pin`),
3. In the `<system>` `.XDC` file the proper FPGA pin must be specified.



Right Click → Make external ... (CTRL+T)



Rename of external ports for **dip_pin**, and **pb_pin** !
(Right click → External interface properties (CTRL+E))



Block Design – Layout synthesis

- Update the Block Design:

- Regenerate Layout



- Validate Design (DRC)



- Flow Navigator → Run Synthesis

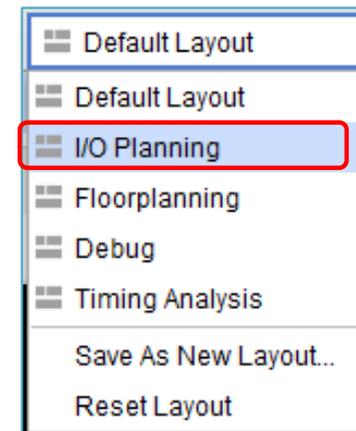


Run Synthesis

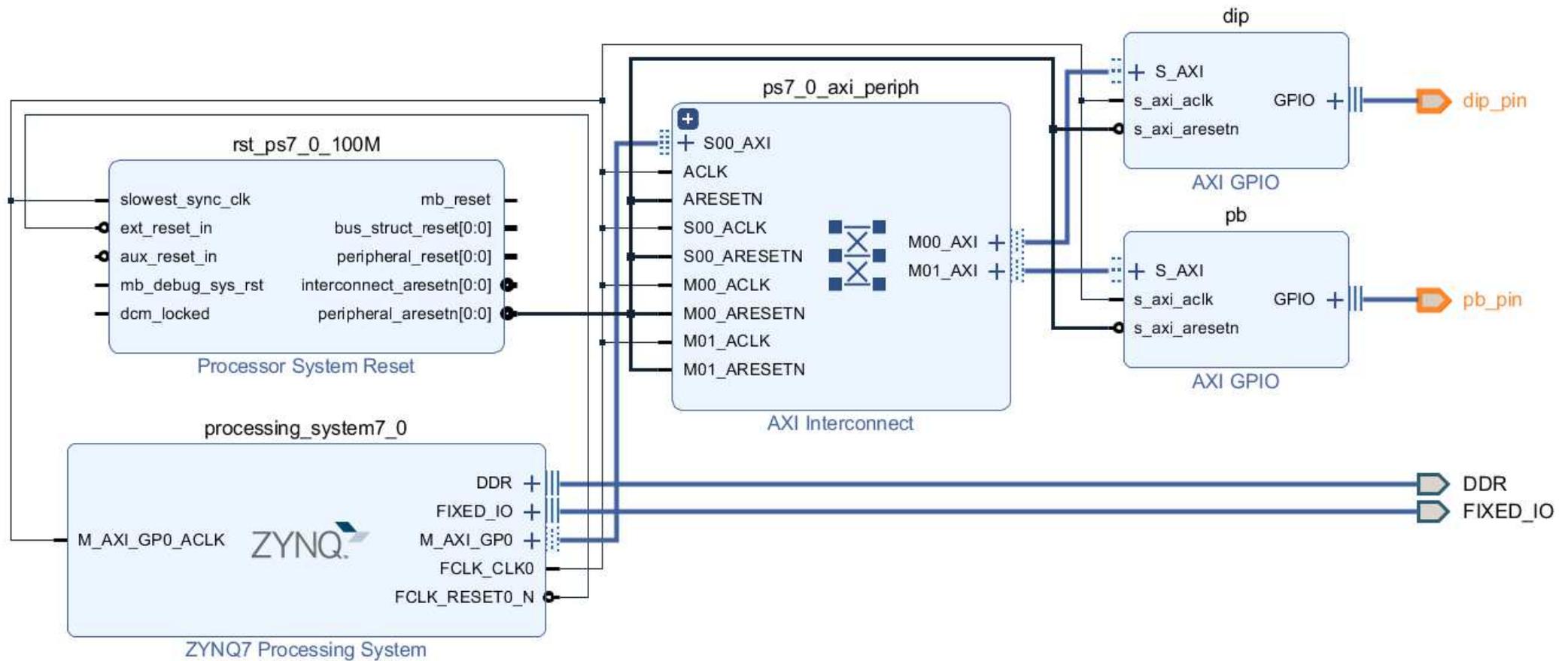
- Open Synthesized Design, OK

- At the final step, the FPGA I/O pins must also be assigned to the two external ports (`dip_pin` and `pb_pin`, respectively)!

- Layout menu → I/O Planning layout view



Block Design – Full view



I/O Planning – Pin assignment

Automatic pin assignment can be checked based on Zybo_master.xdc file or Zybo schematic.

The screenshot displays the I/O Planning tool in Vivado. The top window shows a grid of pins with various colors and symbols indicating their status and assignments. The bottom window shows a table of I/O ports with columns for Name, Direction, Board Part Pin, Board Part Interface, Neg Diff Pair, Package Pin, Fixed, Bank, I/O Std, Vcco, Vref, Drive Strength, Slew Type, Pull Type, and Off-Chip Termination.

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination
GPIO_44577 (4)	IN					<input checked="" type="checkbox"/>	34	LVCN0333*	3.300				NONE	NONE
pb_dip_tri_i (4)	IN					<input checked="" type="checkbox"/>	34	LVCN0333*	3.300				NONE	NONE
pb_dip_tri_i[3]	IN	btns_4bits_tr...			Y16	<input checked="" type="checkbox"/>	34	LVCN0333*	3.300				NONE	NONE
pb_dip_tri_i[2]	IN	btns_4bits_tr...			V16	<input checked="" type="checkbox"/>	34	LVCN0333*	3.300				NONE	NONE
pb_dip_tri_i[1]	IN	btns_4bits_tr...			P16	<input checked="" type="checkbox"/>	34	LVCN0333*	3.300				NONE	NONE
pb_dip_tri_i[0]	IN	btns_4bits_tr...			R18	<input checked="" type="checkbox"/>	34	LVCN0333*	3.300				NONE	NONE
Scalar ports (0)														
GPIO_51240 (4)	IN					<input checked="" type="checkbox"/>	(Multiple)	LVCN0333*	3.300				NONE	NONE
dip_pin_tri_i (4)	IN					<input checked="" type="checkbox"/>	(Multiple)	LVCN0333*	3.300				NONE	NONE
dip_pin_tri_i[3]	IN	sws_4bits_tr...			T16	<input checked="" type="checkbox"/>	34	LVCN0333*	3.300				NONE	NONE
dip_pin_tri_i[2]	IN	sws_4bits_tr...			W13	<input checked="" type="checkbox"/>	34	LVCN0333*	3.300				NONE	NONE
dip_pin_tri_i[1]	IN	sws_4bits_tr...			P15	<input checked="" type="checkbox"/>	34	LVCN0333*	3.300				NONE	NONE
dip_pin_tri_i[0]	IN	sws_4bits_tr...			G15	<input checked="" type="checkbox"/>	35	LVCN0333*	3.300				NONE	NONE
Scalar ports (0)														

Implementation and Bitstream generation

- Flow Navigator menu → **Run Implementation**



- It can filter out possible wrong assignments / errors,
- Warning messages are allowed (the design can be implemented),
- Some floating wires are also allowed (e.g. Peripheral Reset, etc.).
- **While Vivado is working you can check out the synthesis/implementation reports!**

Finally, run the Bitstream generation:

- Flow Navigator → **Generate Bitstream**



Q & A 1.)

- **What is the physical package pin value of the push buttons (pb)?**

- R18 = pb_pin_tri_i[0]
- P16 = pb_pin_tri_i[1]
- V16 = pb_pin_tri_i[2]
- Y16 = pb_pin_tri_i[3]

- **What is the physical package pin value of the dip switches (dip):**

- G15 = dip_pin_tri_i[0]
- P15 = dip_pin_tri_i[1]
- W13 = dip_pin_tri_i[2]
- T16 = dip_pin_tri_i[3]

- **What are their directions?**

- All „IN” as INput direction

Layout Editor – Floorplanning

IMPLEMENTED DESIGN - xc7z010clg400-1 (active)

Physical Cor Device x ? _ □ ▢

Netlist ? _ □ ▢

Project Summary x Device x system_wrapper.vhd x

Internal VREF

- 0.6V
- 0.675V
- 0.75V
- 0.9V
- NONE (2)
- I/O Bank 34
- I/O Bank 35

Drop I/O banks on voltages or the "NONE" folder to set/unset Internal VREF.

Cell Prop x Clock Regi ? _ □ ▢

dip_pin_tri_i_IBUF[0]_inst

Name: dip_pin_tri_i_IBUF[0]_inst

Reference name: IBUF

Type: IO

BEL: INBUF_EN

Site: G15

General Properties Nets

Leaf Cells (8)

- dip_pin_tri_i_IBUF[0]_inst (IBUF)
- dip_pin_tri_i_IBUF[1]_inst (IBUF)
- dip_pin_tri_i_IBUF[2]_inst (IBUF)
- dip_pin_tri_i_IBUF[3]_inst (IBUF)
- pb_dip_tri_i_IBUF[0]_inst (IBUF)
- pb_dip_tri_i_IBUF[1]_inst (IBUF)
- pb_dip_tri_i_IBUF[2]_inst (IBUF)
- pb_dip_tri_i_IBUF[3]_inst (IBUF)

system_i (system)

Timing

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 2,712 ns	Worst Hold Slack (WHS): 0,071 ns	Worst Pulse Width Slack (WPWS): 4,020 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 1701	Total Number of Endpoints: 1701	Total Number of Endpoints: 885

All user specified timing constraints are met.



XILINX VIVADO DESIGN SUITE

LAB02_A. ANALYZING BLOCK DIAGRAM AND RIPOITS

SZÉCHENYI  2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Strukturális
és Beruházási Alapok



BEFEKTETÉS A JÖVŐBE

Analyzing Block Diagram

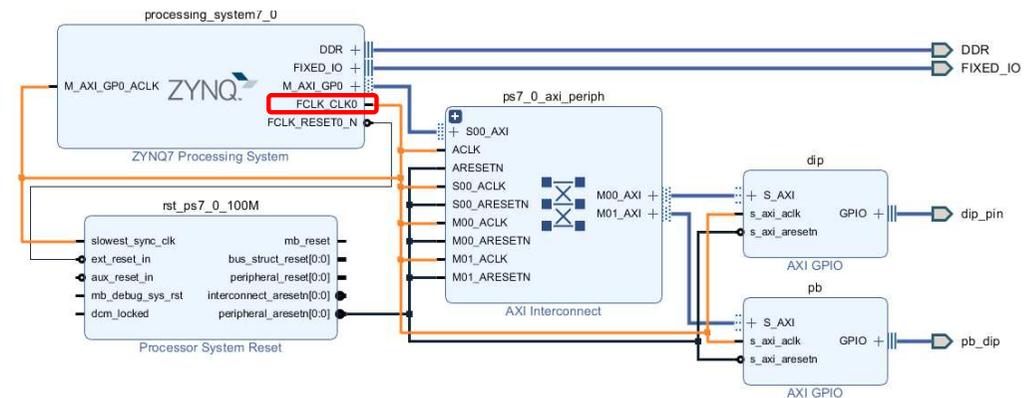
- Question 1.) (buses, internal signals)
 - Which IP peripheral instance(s) are associated with the clock named `processing_system7_0_FCLK_CLK0`?
 - What is its frequency?
 - Which IP peripheral instance(s) are connected to the AXI Lite bus interface?
- Question 2.) (addresses)
 - Outline a full memory space / map of the system by specifying instance names!
- Question 3.) (resource utilization)
 - How many resources are utilized on the PL/FPGA side?

Analyzing Block Diagram (cont.)

- Question 1.) Solution

- Which IP peripheral instance(s) are associated with the clock named `processing_system7_0_FCLK_CLK0` ?

- `processing_system7` (feedback)
- `rst_processing_system7`
- `dip`
- `pb`
- `ps7_axi_periph`

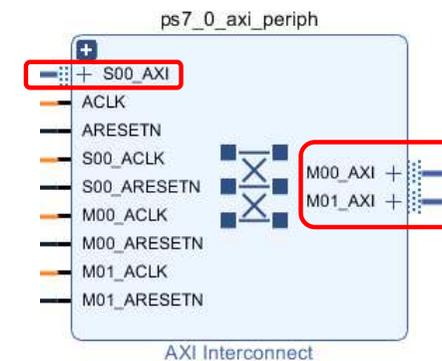


- Clock `processing_system7_0_FCLK_CLK0` is:

- **100 MHz!** (Just check it: ZynqPS → Clock Configuration → PL Fabric Clock)

- Which IP peripheral instance(s) are connected to the AXI Lite bus interface?

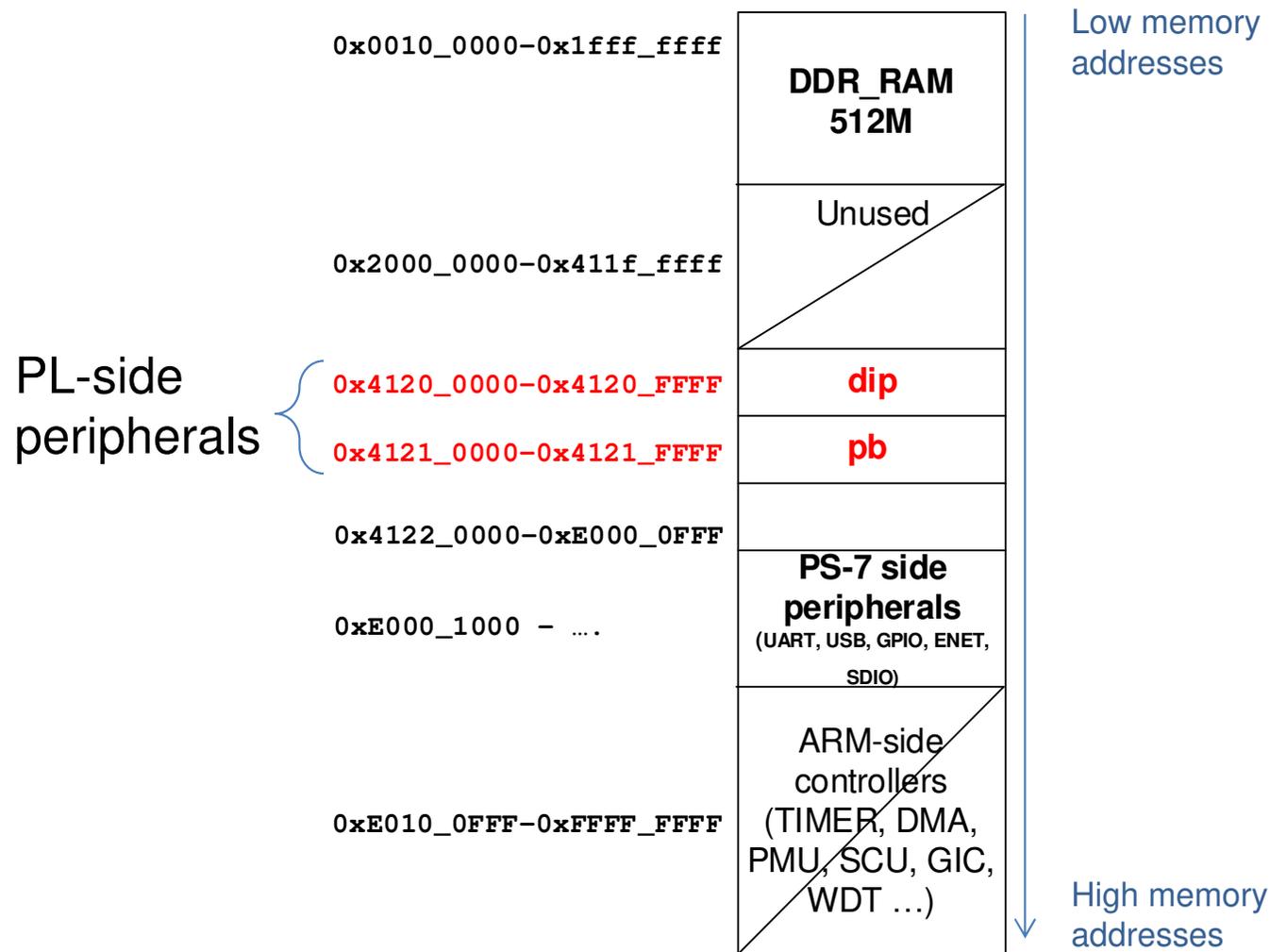
- `processing_system7`
- `dip`
- `pb`



Analyzing Block Diagram (cont.)

- Question 2.) Solution

- memory space: Block Diagram → Address Editor or VITIS .XSA



Analyzing Block Diagram (cont.)

- Question 3.) Solution

- How many resources are utilized on the PL/FPGA side?

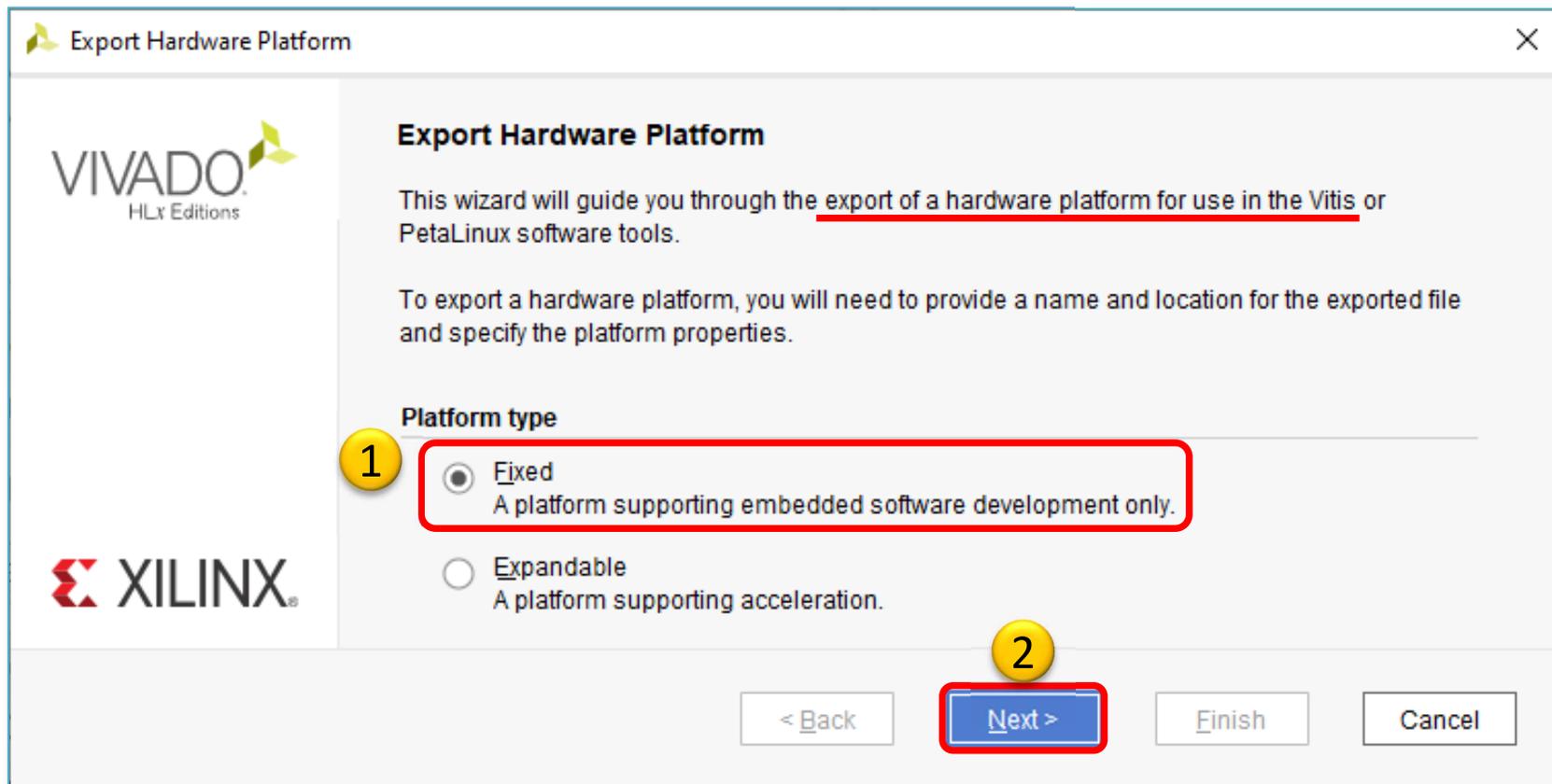
- Reports tab → Report Utilization (vagy Project Summary )

Site Type	Used	Fixed	Available	Util%
Slice LUTs	566	0	17600	3.22
LUT as Logic	504	0	17600	2.86
LUT as Memory	62	0	6000	1.03
LUT as Distributed RAM	0	0		
LUT as Shift Register	62	0		
Slice Registers	815	0	35200	2.32
Register as Flip Flop	815	0	35200	2.32
Register as Latch	0	0	35200	0.00
F7 Muxes	0	0	8800	0.00
F8 Muxes	0	0	4400	0.00

VIVADO Export HW → VITIS (~SDK)

- **File → Export → Export Hardware...**

2020.x: at least an Elaborated Design must be able to be exported to HW!



VIVADO Export HW → VITIS (cont.)

Select „Include bitstream” option as output:

Export Hardware Platform

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.

1 **Include bitstream**
This platform includes the complete hardware implementation and bitstream, in addition to the hardware specification for software tools.

2

< Back **Next >** Finish Cancel

Hence the PL (FPGA) side has been configured, a bitstream (.BIT) file generation is required!

Export HW → VITIS (cont.)

Set **XSA*** file name and export directory path:

Export Hardware Platform

Files

Enter the name of your hardware platform file, and the directory where the XSA file will be stored.

XSA file name: system_wrapper

Export to: F:\Vivado_2020.1\lab02_a

The XSA will be written to: F:\Vivado_2020.1\lab02_a\system_wrapper.xsa

< Back Next > Finish Cancel

Export Hardware Platform

Exporting hardware platform...

Background Cancel



USING XILINX VITIS

LAB02_A. Creating a software test application

SZÉCHENYI  2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Strukturális
és Beruházási Alapok



BEFEKTETÉS A JÖVŐBE

VITIS – General steps of application development

- 
1. Creating a Vivado project, then Export HW → VITIS, ✓
 2. Creating a new application or an application generated from a C/C++ template (e.g. *TestApp* peripheral test):
 - a. Importing **.XSA**
 - b. Generating and compiling an application project containing a platform and a domain inside (~**BSP**: Board Support Package),
 - c. Generating a Linker Script (specifying memory sections, **.LD**),
 - d. Writing / generating and compiling the SW application
 3. Setup a Serial terminal/Console (USB-serial port),
 4. Connecting and setup a JTAG-USB programmer,
 - Configuring the FPGA (**.BIT** if PL-side existing)
 5. Creating a 'Debug Configuration' for hardware debugging
 6. Debug (insert breakpoints, stepping, run, etc.)

Starting VITIS



Xilinx Vitis 2020.1

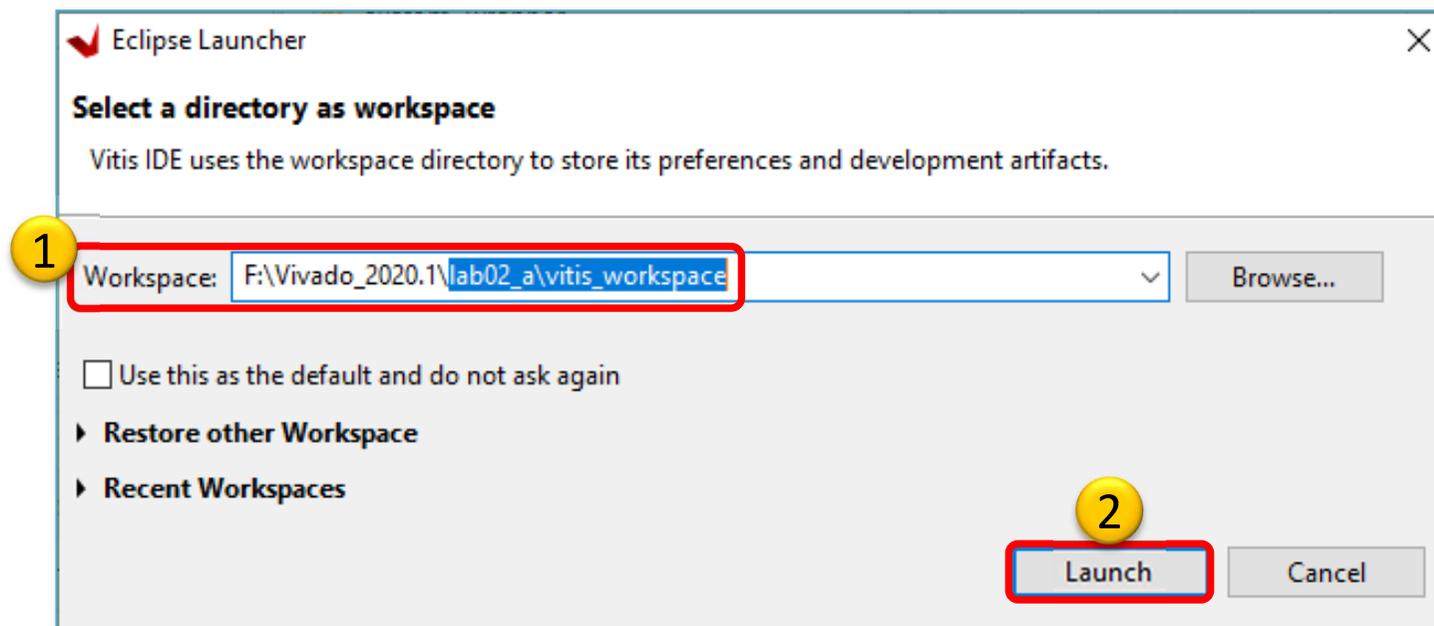
From Vivado: Tools menu → Launch VITIS IDE

OR externally

Start menu → Programs → Xilinx Design Tools → Xilinx VITIS 2020.1

Do Not run Xilinx VITIS HLS 2020.1 !

- Set workspace directory properly (*lab02_a*):
 - Recommended to use *vitis_workspace* as a subdirectory in your lab folder. Then Launch...



Xilinx VITIS – Create Application

Recall the steps of the former LAB01!

1. Create a new application project

- File → New → Application Project...

2. Platform – Create a new platform from HW (XSA)

- Browse... for LAB02 `system_wrapper.xsa`. Open it.
- Do not select the „*Generate boot components*”

3. Application project details

- Type „`TestApp`” as project name
- Type „`Zybo_test_system`” as system project name
- Select `ps7_cortexa9_0` as target ARM core 0

4. Domain: leave settings as default (standalone)

Create Application (cont)

New Application Project

Application Project Details
Specify the application project name and its system project properties

1 Application project name:

System Project
Create a new system project for the application or select an existing one from the workspace

Select a system project

System project details
2 System project name:

Target processor
Select target processor for the Application project.

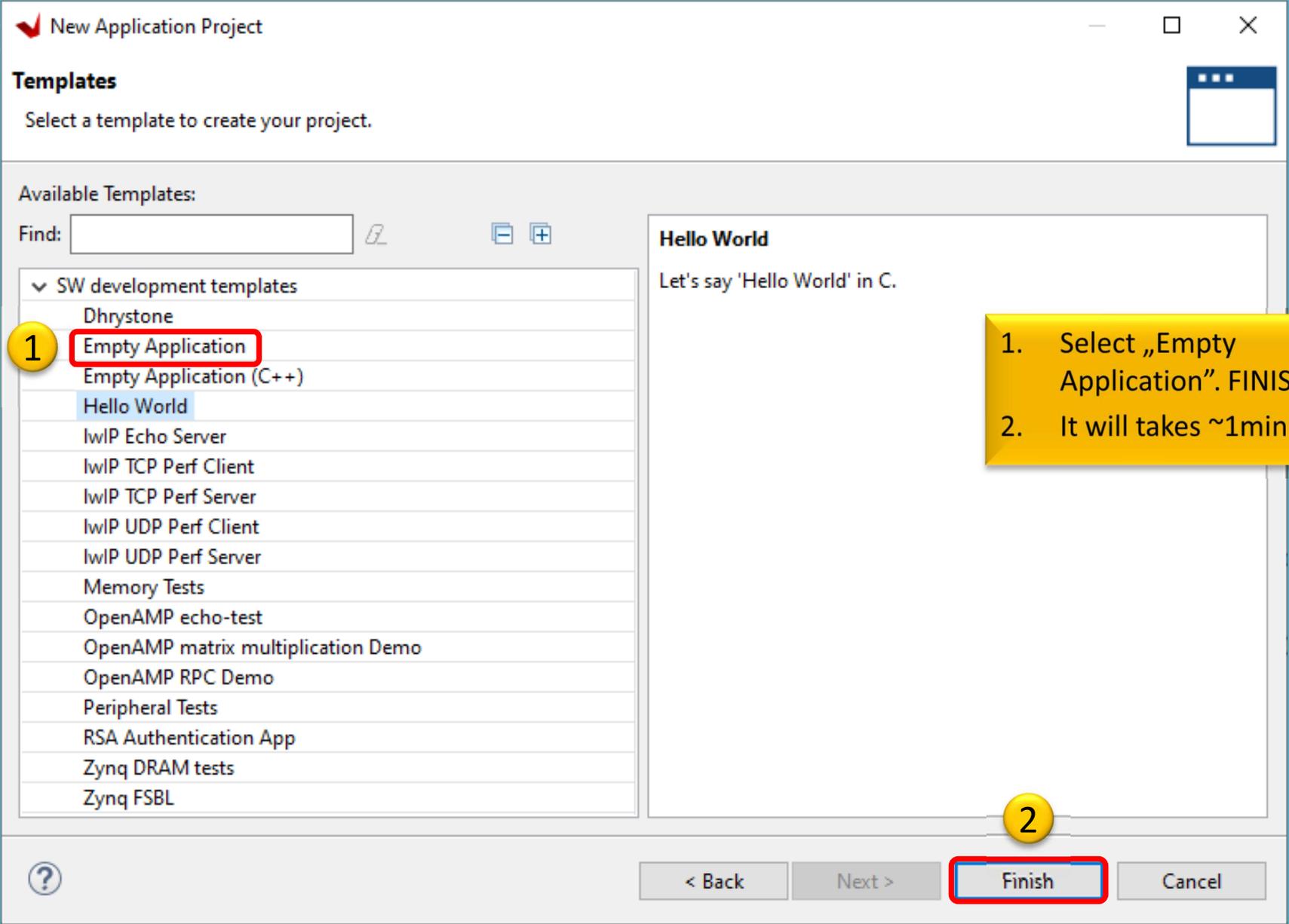
Processor	Associated applications
3 ps7_cortexa9_0	TestApp
ps7_cortexa9_1	
ps7_cortexa9 SMP	

Show all processors in the hardware specification

4

1. Type app project name: „TestApp”
2. Type system project name: „Zybo_test_system”
3. Select ARM CortexA-0 core for TestApp
4. NEXT.

Example I.) Creating TestApp application

- 

New Application Project

Templates
Select a template to create your project.

Available Templates:

Find:

SW development templates

 - 1 **Empty Application**
 - 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.

 1. Select „Empty Application“. FINISH.
 2. It will takes ~1min time 😊

2

< Back Next > **Finish** Cancel

VITIS GUI – Main window

The screenshot shows the Vitis IDE interface. The Explorer on the left shows the project structure with 'TestApp.prj' highlighted. The Application Project Settings window is open, showing the following configuration:

- Project name: TestApp
- Platform: system_wrapper
- Runtime: cpp
- Domain: domain_ps7_cortexa9_0
- CPU: ps7_cortexa9_0
- OS: standalone

The 'Hardware Specification' link is highlighted, with a callout box explaining that 'Standalone' is a simple, low-level software layer. The callout box contains the following text:

Standalone is a simple, low-level software layer. It provides access to

- **basic processor features:** \$ caches, IRQ interrupts and exceptions
- and **basic features of a hosted environment**, e.g. standard I/O, profiling, abort and exit

VITIS – HW platform

1 platform.spr

2 system_wrapper

3 IP Instance

4 Address Map for processor ps7_cortexa9[0-1]

HW platform from Vivado, description of elaborated embedded system

4G Memory address map (PS)
Check dip and pb addresses!

List and versions of used PS peripherals (below)

Hardware Platform Specification

Design Information

Target FPGA Device: 7z010
Part: xc7z010clg400-1
Created With: Vivado 2020.1
Created On: Fri Aug 21 00:16:52 2020

Note: To view ip parameters, double-click on the cell containing ip name in any of the below tables.

Address Map for processor ps7_cortexa9[0-1]

Cell	Base Address	High Address	Slave Interface	Addr Range	Type
dip	0x41200000	0x4120ffff	S_AXI		register
pb	0x41210000	0x4121ffff	S_AXI		register
ps7_uart_1	0xe0001000	0xe0001fff	-		register
ps7_iop_bus_config_0	0xe0200000	0xe0200fff	-		register
ps7_slcr_0	0xf8000000	0xf8000fff	-		register
ps7_dma_s	0xf8003000	0xf8003fff	-		register
ps7_dma_ns	0xf8004000	0xf8004fff	-		register
ps7_ddrc_0	0xf8006000	0xf8006fff	-		register
ps7_dev_cfg_0	0xf8007000	0xf80070ff	-		register
ps7_xadc_0	0xf8007100	0xf8007120	-		register

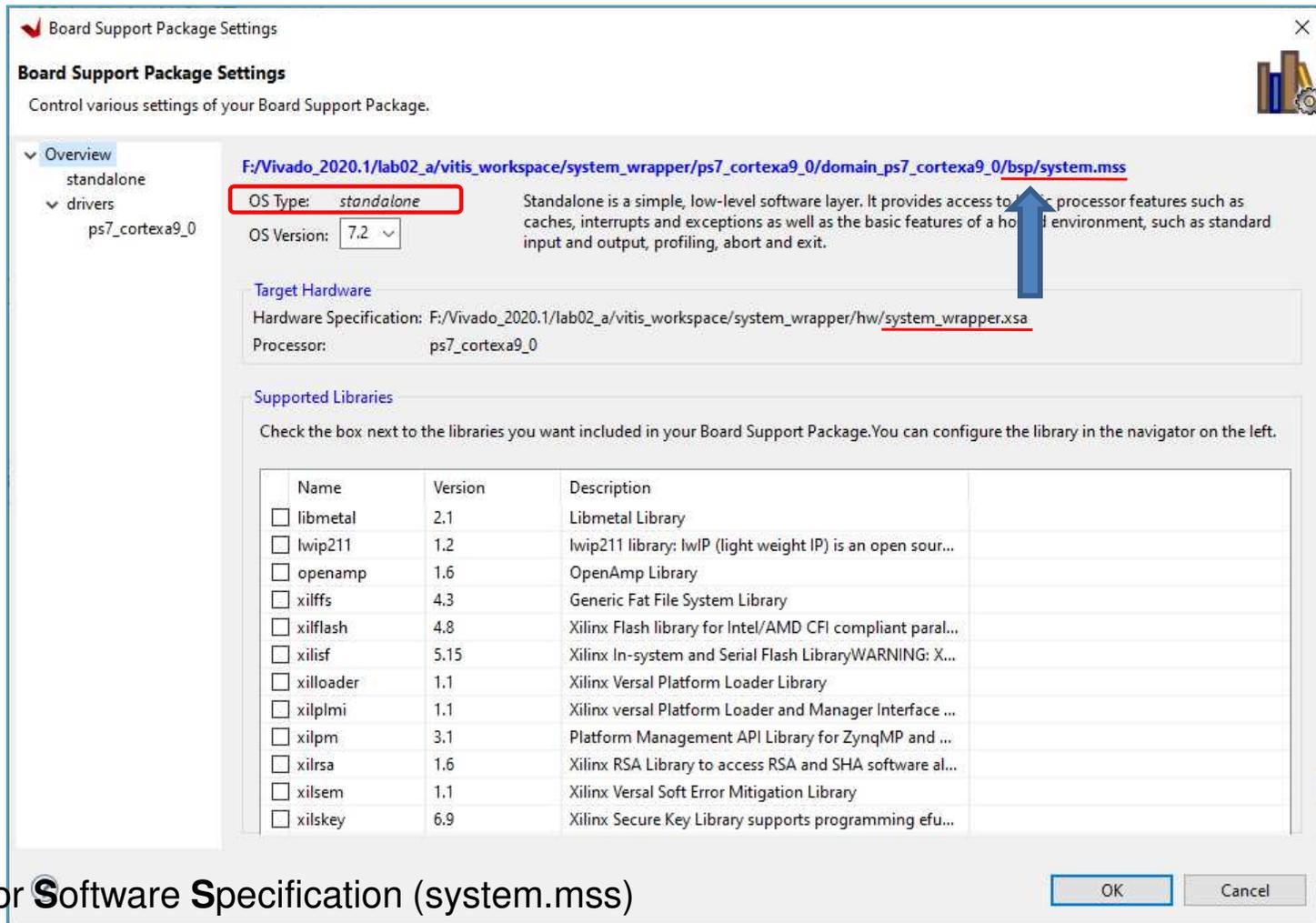
IP Instance

IP Instance	IP Type	IP Version	Register
dip	axi_gpio	2.0	Registers
pb	axi_gpio	2.0	Registers
processing_system7_0	processing_system7	5.5	-
ps7_0_axi_periph	axi_interconnect	2.1	-
ps7_afi_0	ps7_afi	1.00.a	-

VITIS – BSP Board Support Package

- **Software Platform Settings**

- Selected OS: *standalone* vs. *freertos_10* (or 3rd Party OS)
- Supported SW libraries (lib)



Board Support Package Settings

Control various settings of your Board Support Package.

Overview

- standalone
- drivers
 - ps7_cortexa9_0

F:/Vivado_2020.1/lab02_a/vitis_workspace/system_wrapper/ps7_cortexa9_0/domain_ps7_cortexa9_0/bsp/system.mss

OS Type: **standalone** Standalone is a simple, low-level software layer. It provides access to processor features such as caches, interrupts and exceptions as well as the basic features of a host environment, such as standard input and output, profiling, abort and exit.

OS Version: 7.2

Target Hardware

Hardware Specification: F:/Vivado_2020.1/lab02_a/vitis_workspace/system_wrapper/hw/system_wrapper.xsa

Processor: ps7_cortexa9_0

Supported Libraries

Check the box next to the libraries you want included in your Board Support Package. You can configure the library in the navigator on the left.

Name	Version	Description
<input type="checkbox"/> libmetal	2.1	Libmetal Library
<input type="checkbox"/> lwip211	1.2	Lwip211 library: lwIP (light weight IP) is an open sour...
<input type="checkbox"/> openamp	1.6	OpenAmp Library
<input type="checkbox"/> xilffs	4.3	Generic Fat File System Library
<input type="checkbox"/> xilflash	4.8	Xilinx Flash library for Intel/AMD CFI compliant paral...
<input type="checkbox"/> xilisf	5.15	Xilinx In-system and Serial Flash LibraryWARNING: X...
<input type="checkbox"/> xilloader	1.1	Xilinx Versal Platform Loader Library
<input type="checkbox"/> xilplmi	1.1	Xilinx versal Platform Loader and Manager Interface ...
<input type="checkbox"/> xilpm	3.1	Platform Management API Library for ZynqMP and ...
<input type="checkbox"/> xilrsa	1.6	Xilinx RSA Library to access RSA and SHA software al...
<input type="checkbox"/> xilsem	1.1	Xilinx Versal Soft Error Mitigation Library
<input type="checkbox"/> xilskey	6.9	Xilinx Secure Key Library supports programming efu...

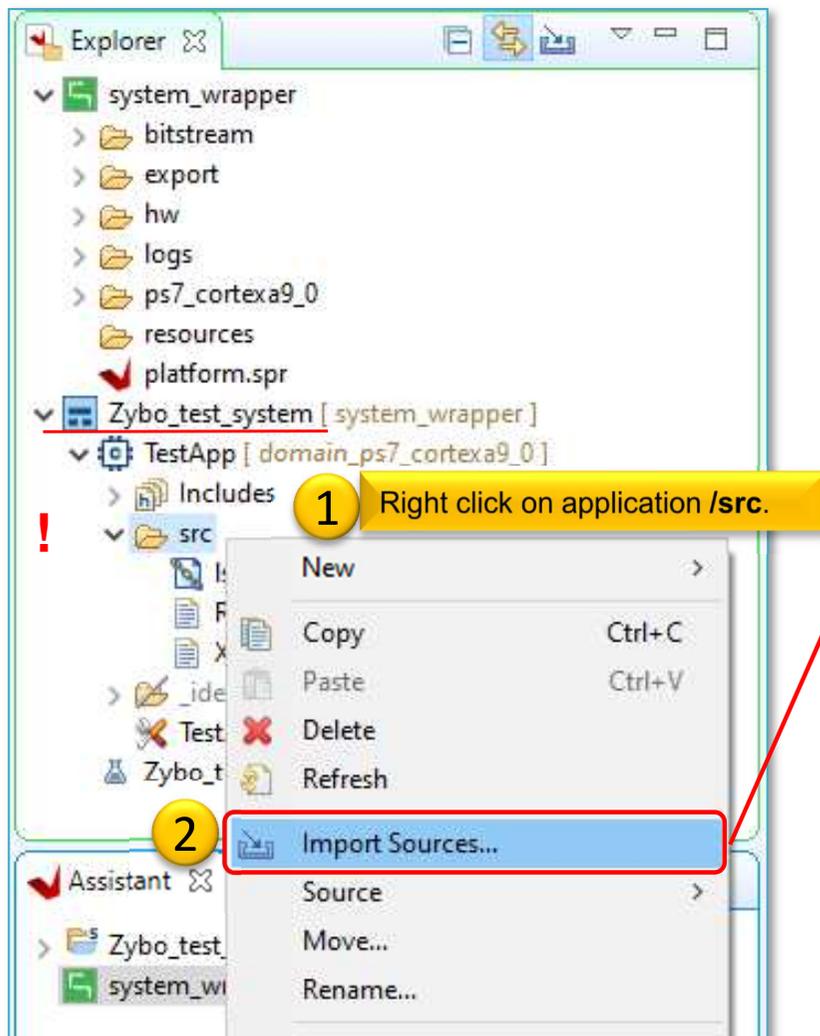
OK Cancel

Q & A 1.)

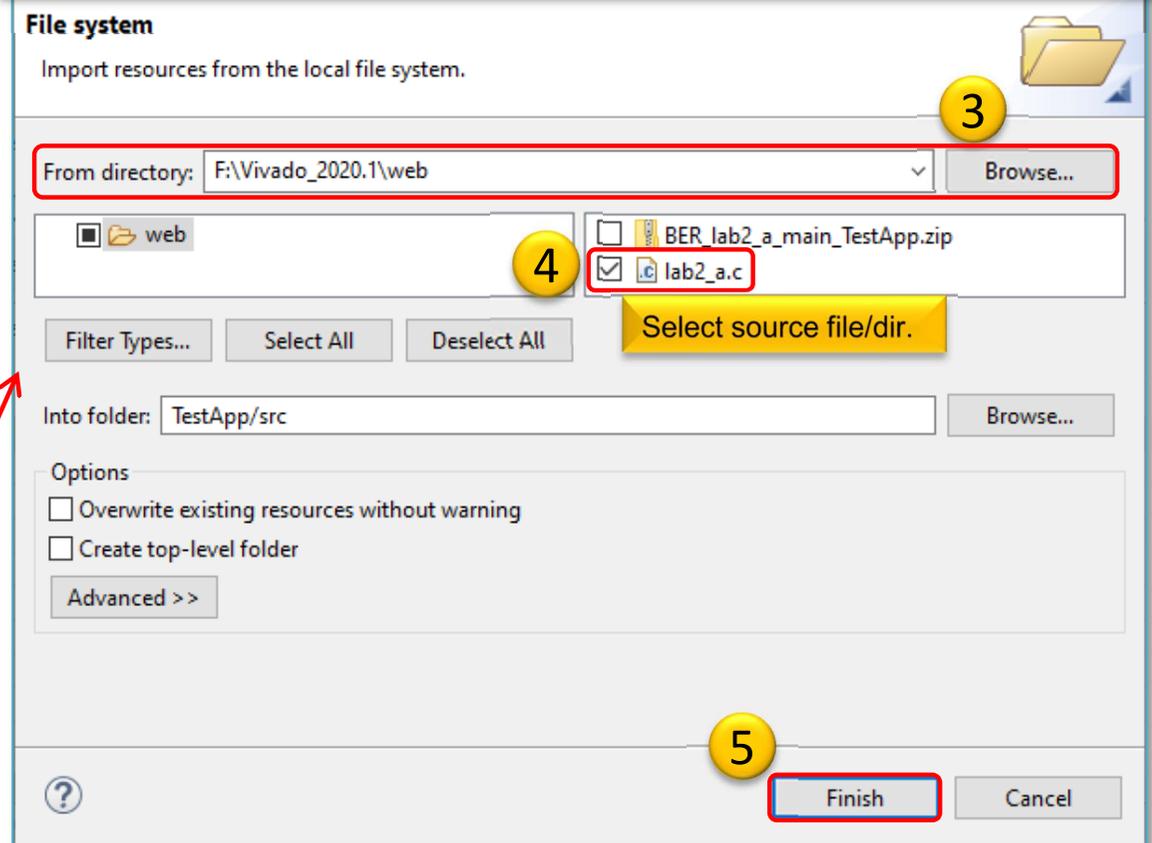
- **What is the *IP type* and *IP version* of „dip” and „pb” instances?**
 - axi_gpio,
 - v2.0
- **What is the driver name of them?**
 - gpio
- **Calculate what size they are?**
 - dip: 0x4120 0000-0x4120 ffff = 64 KByte
 - pb: 0x4121 0000-0x4121 ffff = 64 KByte

Add C/C++ source(s)

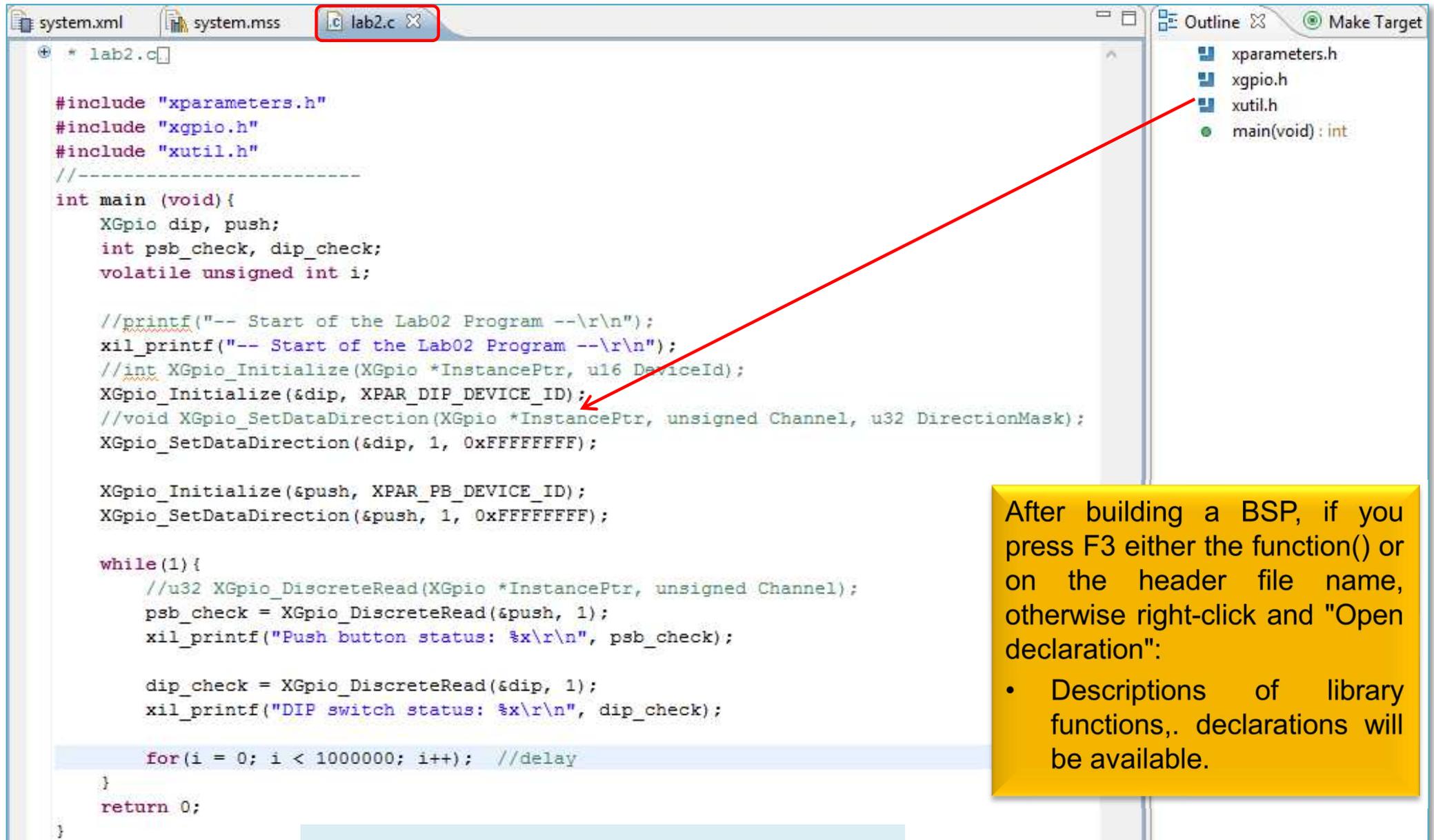
- Add/Create new source (`lab2_a.c`) to the application project



Download and unpack the archive from laboratory website:
[BER lab2 a main TestApp.zip](#)



TestApp – source code



```
system.xml | system.mss | lab2.c X
+ * lab2.c
#include "xparameters.h"
#include "xgpio.h"
#include "xutil.h"
//-----
int main (void){
    XGpio dip, push;
    int psb_check, dip_check;
    volatile unsigned int i;

    //printf("-- Start of the Lab02 Program --\r\n");
    xil_printf("-- Start of the Lab02 Program --\r\n");
    //int XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId);
    XGpio_Initialize(&dip, XPAR_DIP_DEVICE_ID);
    //void XGpio_SetDataDirection(XGpio *InstancePtr, unsigned Channel, u32 DirectionMask);
    XGpio_SetDataDirection(&dip, 1, 0xFFFFFFFF);

    XGpio_Initialize(&push, XPAR_PB_DEVICE_ID);
    XGpio_SetDataDirection(&push, 1, 0xFFFFFFFF);

    while(1){
        //u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
        psb_check = XGpio_DiscreteRead(&push, 1);
        xil_printf("Push button status: %x\r\n", psb_check);

        dip_check = XGpio_DiscreteRead(&dip, 1);
        xil_printf("DIP switch status: %x\r\n", dip_check);

        for(i = 0; i < 1000000; i++); //delay
    }
    return 0;
}
```

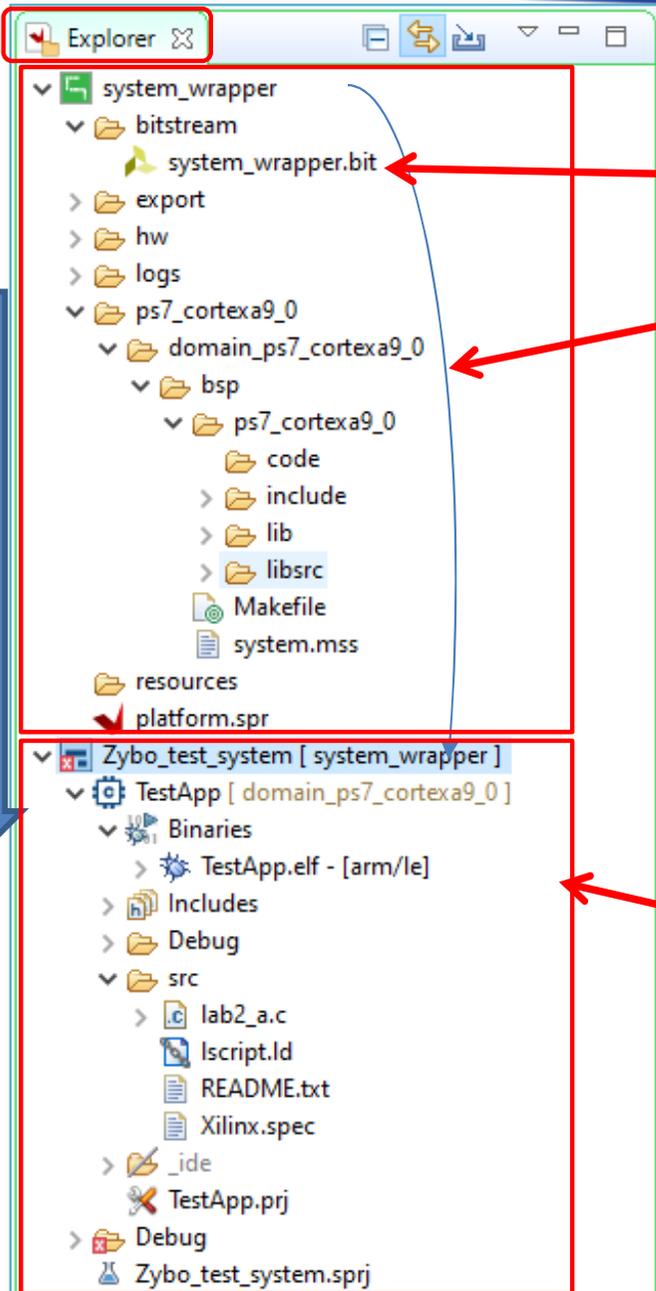
Outline

- xparameters.h
- xgpio.h
- xutil.h
- main(void) : int

After building a BSP, if you press F3 either the function() or on the header file name, otherwise right-click and "Open declaration":

- Descriptions of library functions, declarations will be available.

VITIS – Project Explorer / Hierarchy



 `system_wrapper` as **HW** platform was exported from Vivado (.xsa, .bit, etc.)

– **System_wrapper.bit** (FPGA configuration = bitstream)
contains:

– **BSP:** (OS routines, device drivers, etc.)

- **MSS: Microprocessor software/driver descriptor** (`system.mss`)
- **`/includes/xparameters.h` !!!** (all related `#define` and address ranges are defined here)

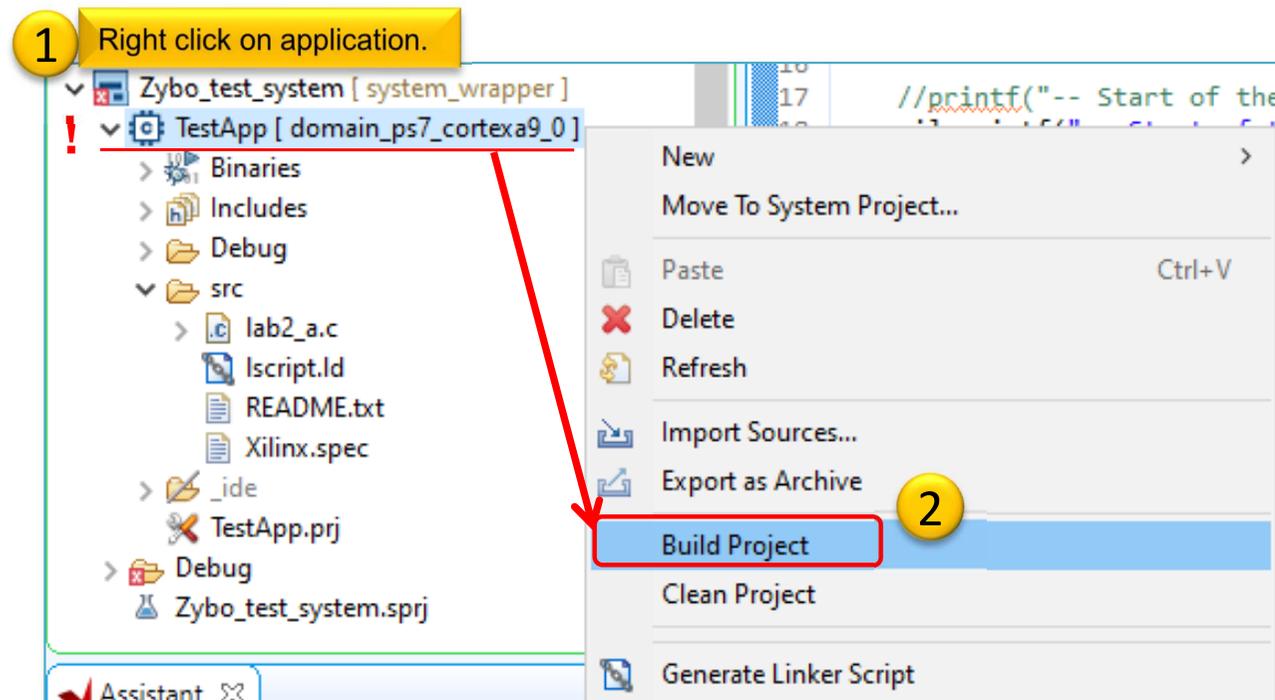
 `Zybo_test_system` as **system_project** contains

 **SW:** TestApp (SW application)

- `\Binaries` (**executable load file as .elf** object file)
- `\Includes` (factory default headers)
- `\Debug`
- `\Src` = collection of `.h`, `.c`, `.cpp` sources (e.g. `lab2_a.c`)
- **.ld = linker script!**
- **Main()** entry point in the `helloworld.c` file.

Build project

- 1. Select Application project (e.g. `TestApp`)
- 2. Project menu → Build Project... in two steps:
 - Build BSP (`system_wrapper`)
 - Build software application (`lab2_a.c`)



Build project – Result (Console)

```
'Building target: TestApp.elf'  
'Invoking: ARM v7 gcc linker'  
arm-none-eabi-gcc -mcpu=cortex-a9 -mfloat-abi=hard -Wl,-  
build-id=none -specs=Xilinx.spec -Wl,-T -Wl,../src/lscript.ld -  
LF:/Vivado_2020.1/lab02_a/vitis_workspace/system_wrapper/export/syste  
m_wrapper/sw/system_wrapper/domain_ps7_cortexa9_0/bsplib/lib -o  
"TestApp.elf" ./src/lab2_a.o -Wl,--start-group,-lxil,-lgcc,-lc,--  
end-group  
'Finished building target: TestApp.elf'  
' '
```

```
'Invoking: ARM v7 Print Size'  
arm-none-eabi-size TestApp.elf |tee "TestApp.elf.size"  
text data bss dec hex filename  
22840 1176 22584 46600 b608 TestApp.elf  
'Finished building: TestApp.elf.size'
```

Decimal size: 46600 byte ~46 KByte . The entire program can be placed both the internal on-chip RAM 0/1 and the external DDR RAM. (On the PL / FPGA-side, however, this amount of BRAM memory should be reserved). Therefore, the executable `.elf` file was also generated successfully.

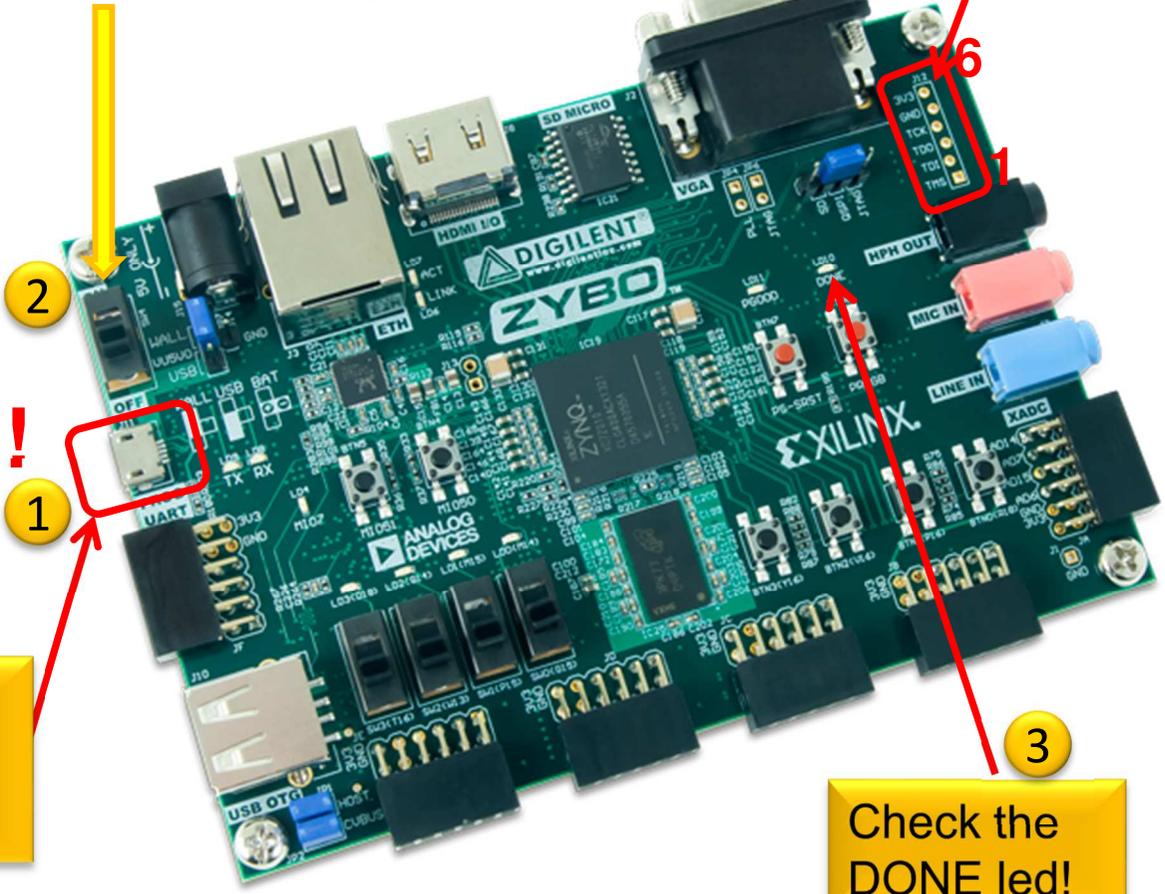
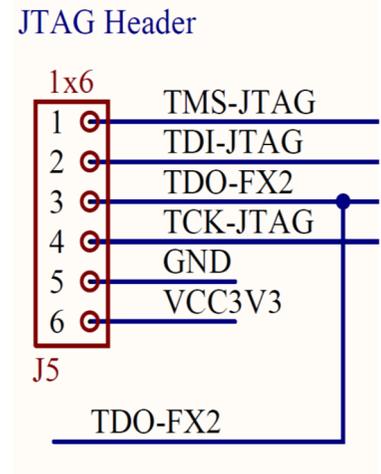
Embedded system and software test verification

1. **Connect** the USB-serial cable (power+programmer functionality). Please check:

- JP7 jumper = USB power!
- JP5 jumper = JTAG mode!

2. Now **Power ON** the ZyBo platform

JTAG programming port (optional, but we don't use it!)



We use the USB-serial connector.

Check the DONE led!

ZyBo – Xilinx USB programming cable

- VCC3V3 – red VREF (6)
- GND – black GND (5)
- TCK-JTAG – yellow TCK (4)
- TDO-FX2 – lilac – TDO (3)
- TDI-JTAG – white TDI (2)
- TMS-JTAG – green TMS (1)

Creating Debug Configuration

- **Select the application** (TestApp) in the Project Explorer

The screenshot shows an IDE interface with the Project Explorer on the left. A yellow circle with the number '1' is next to the 'TestApp' folder, which is selected. A yellow callout box says 'Right click on application.' A red arrow points from the 'TestApp' folder to a context menu. In the context menu, 'Debug As' is highlighted with a red box and a yellow circle with the number '2'. A red arrow points from 'Debug As' to a sub-menu. In the sub-menu, 'Debug Configurations...' is highlighted with a red box and a yellow circle with the number '3'. The background shows a code editor with C code and a console window with build output.

```
Lab02 Program --\r\n");
the Lab02 Program --\r\n");
pio *InstancePtr, u16 DeviceId);
AR_DIP_DEVICE_ID);
tion(XGpio *InstancePtr, unsigned Channel, u32 Direction,
ip, 1, 0xFFFFFFFF);
```

```
building file: ../src/lab2_a.c
'Invoking: ARM v7 gcc compiler'
arm-none-eabi-gcc -Wall -O0 -g3 -
'finished building: /src/lab2
```

Create a new GDB configuration

- Select „Single Application Debug (GDB)” option
 - New configuration 



Debug Configurations

Create, manage, and run configurations

Debug a program using Application Debugger (GDB).

Name: Debugger_TestApp-GDB

Check all GDB settings.

Hardware Platform: Search... Browse...

Bitstream File: Search... Browse... Generate...

Use FSBL flow for initialization

Initialization File: Search... Browse...

Summary:

Following operations will be performed before launching the debugger.

1. Resets entire system. Clears the FPGA fabric (PL).
2. Program FPGA fabric (PL).
3. Runs ps7_init to initialize PS.
4. Runs ps7_post_config. Enables level shifters from PL to PS. (Recommended to use this option only after system reset or board power ON).
5. All processors in the system will be suspended, and Applications will be downloaded to the following processors as specified in the Applications tab.
 - 1) ps7_cortexa9_0
(F:\Vivado_2020.1\lab02_a\vitis_workspace\TestApp\Debug\TestApp.elf)

Reset entire system

Program FPGA

Skip Revision Check

Run ps7_init

Run ps7_post_config

Enable Cross-Triggering

Filter matched 4 of 4 items

Revert Apply

Debug Close

Lunching Debugger

1

2

Debugger step into this entry point = first instruction of the source code. (lab2_a.c)

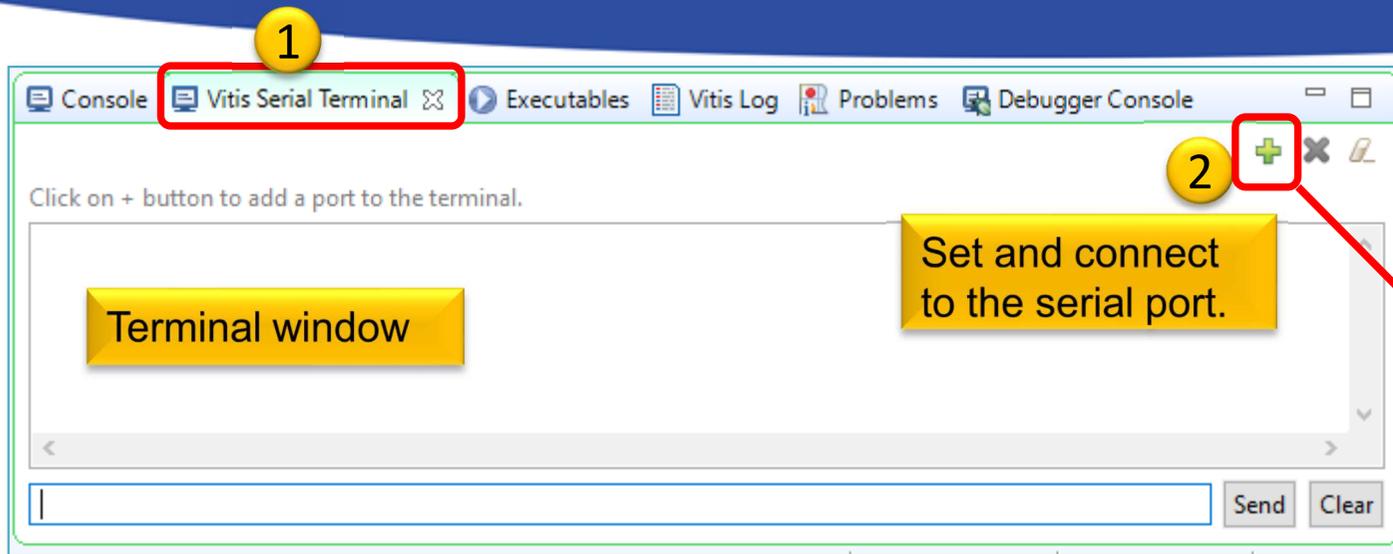
```
20 * lab2.c
7
8 #include "xparameters.h"
9 #include "xgpio.h"
10 // #include "xutil.h"
11 //-----
12 int main (void){
13     XGpio dip, push;
14     int psb_check, dip_check;
15     volatile unsigned int i;
16
17     //printf("-- Start of the Lab02 Program --\r\n");
18     xil_printf("-- Start of the Lab02 Program --\r\n");
19     //int XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId);
20     XGpio_Initialize(&dip, XPAR_DIP_DEVICE_ID);
21     //void XGpio_SetDataDirection(XGpio *InstancePtr, unsigned Channel,
22     XGpio_SetDataDirection(&dip, 1, 0xFFFFFFFF);
23
24     XGpio_Initialize(&push, XPAR_PB_DEVICE_ID);
25     XGpio_SetDataDirection(&push, 1, 0xFFFFFFFF);
26
27     while(1){
28         //u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
29         psb_check = XGpio_DiscreteRead(&push, 1);
30         xil_printf("Push button status: %x\r\n", psb_check);
31
32         dip_check = XGpio_DiscreteRead(&dip, 1);
33         xil_printf("DIP switch status: %x\r\n", dip_check);
34
35         for(i = 0; i < 1000000; i++); //delay
36     }
37     return 0;
38 }
39
```

3

Console Vitis Serial Termi... Executables Vitis Log Problems Debugger

Launching Debugger_Zy...t-GDB: (98%)

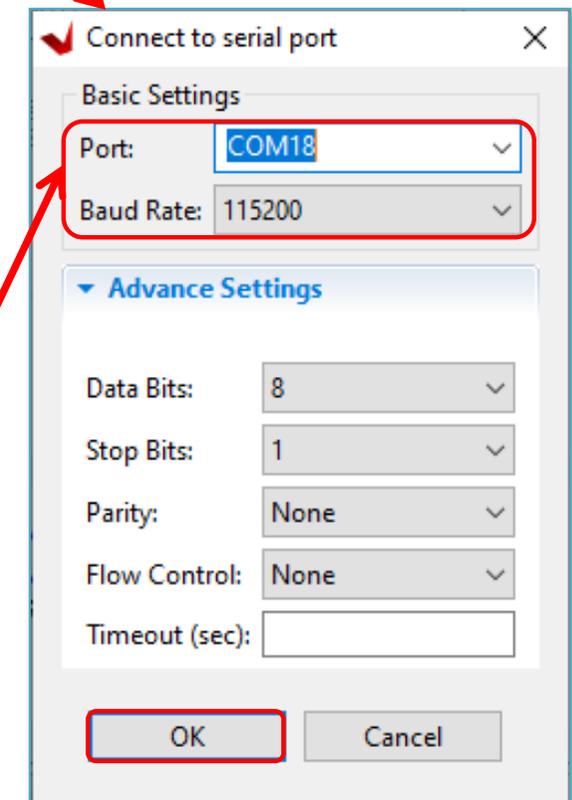
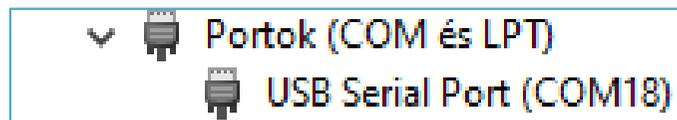
Set Debug-serial port (VITIS terminal)



Possible ways to login via serial port:

1. **VITIS Serial Terminal: integrated** or
2. using external program: (HyperTerminal, Putty etc.)

- Terminal: BaudRate / Data bits according to the settings of **PS UART** or / **AXI_UART IP modul!**
- Port: COM[XY] – setting according to WINDOWS → „Device Manager” → Ports (COM & LPT)



TestApp – Verification result

Set a brake-point!

```
24 XGpio_Initialize(&push, XPAR_PB_DEVICE_ID);
25 XGpio_SetDataDirection(&push, 1, 0xFFFFFFFF);
26
27 while(1){
28     //u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
29     psb_check = XGpio_DiscreteRead(&push, 1);
30     xil_printf("Push button status: %x\r\n", psb_check);
31
32     dip_check = XGpio_DiscreteRead(&dip, 1);
33     xil_printf("DIP switch status: %x\r\n", dip_check);
34
35     for(i = 0; i < 1000000; i++); //delay
36 }
37 return 0;
38 }
39
```

Console | Vitis Serial Termi... | Executables | Vitis Log | Problems | Debugger Cons...

Connected to: Serial (COM18, 115200, 0, 8)

```
-- Start of the Lab02 Program --
Push button status: 0
DIP switch status: 6
Push button status: 0
DIP switch status: 6
Push button status: 0
DIP switch status: F
Push button status: 0
DIP switch status: F
Push button status: C
```

What do you experience?
LITTLE ENDIAN!

TestApp – Verification result

1

Name	Type	Value
▼ dip	XGpio	{...}
(x)- BaseAddr	UINTPTR	0x41200000 (Hex)
(x)- IsReady	u32	0x11111111 (Hex)
(x)- InterruptF	int	0
(x)- IsDual	int	0
▼ push	XGpio	{...}
(x)- BaseAddr	UINTPTR	0x41210000
(x)- IsReady	u32	0x11111111
(x)- InterruptF	int	0
(x)- IsDual	int	0
(x)- psb_ch		
(x)- dip_ch		
(x)- i		

2 Right click on the selected variable(s).

3 Number Format

4 Hex

Name : Base
Details
Default
Decimal
Hex:0x4
Binary:10000010010000100000000000000000
Octal:010110200000

During active debug process set in the Variables window both the *BaseAddress* / *IsReady* parameters of the pb / push variables to Hexadecimal format.

Terminate Debug process

- **IMPORTANT!** At the end of the HW debug, the running debug configuration must be *Terminated and Removed!*

The screenshot shows the Vitis IDE interface. The top toolbar contains various icons for running and debugging. The 'Debug' window is open, showing a tree view of the debug configuration: 'Debugger_Zybo_test-GDB [Single Application Debug (GDB)]' is expanded to show 'Zybo_test.elf [1]', 'Thread #1 1 (ARM Cortex-A9 MPCore #0 Step) (Suspended)', and 'main() at helloworld.c:61 0x5b0'. A red box highlights the 'Debugger_Zybo_test-GDB [Single Application Debug (GDB)]' entry, with a yellow callout '1' and an arrow pointing to it. A yellow box on the left contains the text 'Right click on GDB.' Below the Debug window, the Explorer view shows the project structure, including 'Zybo_test_system [system_wrapper]' and 'Zybo_test [domain_ps7_cortexa9_0]'. A context menu is open over the 'Debugger_Zybo_test-GDB' entry, with a red box highlighting the 'Terminate and Remove' option, and a yellow callout '2' and an arrow pointing to it. A yellow box on the right contains the text 'Stop running the debug configuration and remove it (otherwise it would constantly occupy memory!)'. The IDE title bar reads 'vitis_workspace - Zybo_test/src/helloworld.c - Vitis IDE'.

Right click on GDB.

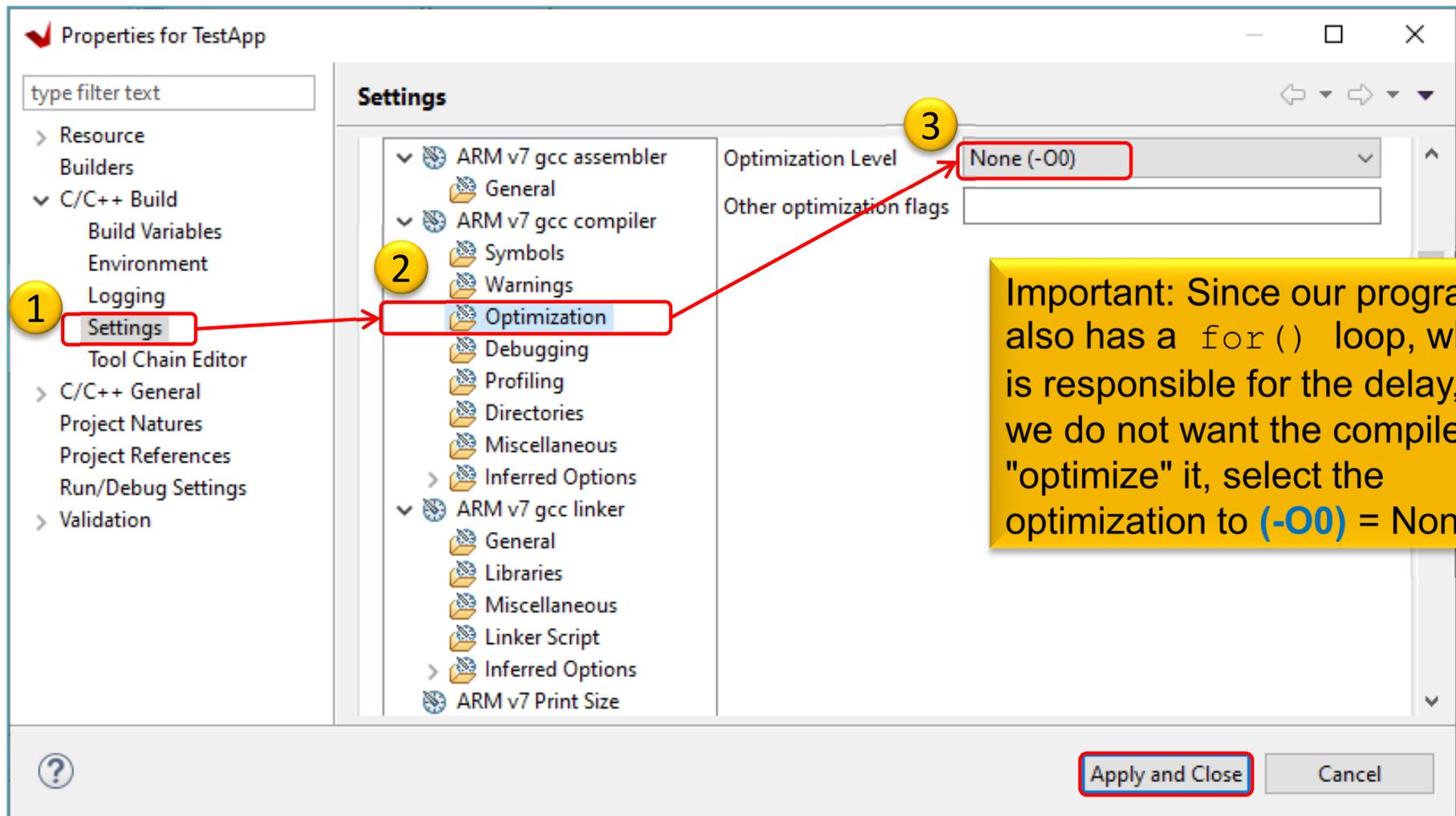
1

2

Stop running the debug configuration and remove it (otherwise it would constantly occupy memory!)

Compiler settings

- Check compiler settings:
 - Right click on **TestApp** → **C/C++ Build Settings**



Linker Script generation (Basic)

- Xilinx menu → Generate Linker Script (`lscript.ld`) → **RAM0**

1 Right click on application.

2 Build Project

Choose between Internal RAM0/1 vs. External memory space

- Instructions / Program codes
- Data / Variables
- Heap / Stack sizes

3

Basic Advanced

Place Code Sections in: ps7_ram_0

Place Data Sections in: ps7_ram_0

Place Heap and Stack in: ps7_ram_0

Heap Size: 1 KB 0x00000400

Stack Size: 1 KB 0x00000400

Place each sections into RAM0 instead of external DDR memory! Then Generate.

4 Generate Cancel

Memory	Base Address	Size
ps7_dds_0	0x00100000	511 MB
ps7_ram_0	0x00000000	192 KB
ps7_ram_1	0xFFFF0000	~63,5 KB

Example II.) Peripheral Test

1. File → New → Application Project ...
2. Select `system_wrapper.xsa` as platform
3. Add „Peripheral_test” as application project name
 - + Create a new system project (leave Peripheral Test_system by default)
 - Select `ps7_cortexa9_0` ARM core-0
4. Leave domain settings as default
5. Templates: select „Peripheral Test”. FINISH.

Example II.) Build and Debug

- 
1. Build the built-in *Peripheral Test* application (.elf)
 2. Launch the proper Debug configuration (GDB) for hardware debugging
 3. Setup the VITIS serial terminal/Console (USB-serial port),
 4. Connecting and setup a JTAG-USB programmer,
 - Configuring the FPGA (.BIT if PL-side existing)
 5. Debug procedure (insert breakpoints, stepping, run, etc.)
 - **Watching variables and examine memory monitor !**
 6. At the end of debug procedure do not forget to **Terminate and Remove** the actual Debug configuration (GDB)!
 7. That's all :D

Example II.) Questions & Answers

- What is the size of the Peripheral_test application?
 - ~82 Kbyte

```
'Invoking: ARM v7 Print Size'  
arm-none-eabi-size Peripheral_test.elf |tee  
"Peripheral_test.elf.size"  
   text    data     bss     dec     hexfilename  
 46796   1992   33440   82228  
14134Peripheral_test.elf  
'Finished building: Peripheral_test.elf.size'
```

- Generate the linker script to RAM1 (ps7_ram1) address space! What do you experience?
 - **Linking ERROR. Why?**

HW debugging steps – Peripheral_test

- 
1. Create a new Debug Configuration (GDB) for Peripheral_test
 2. Lunch Debugger
 3. Set-up Debug-serial port (VITIS terminal)
 4. HW debug – Peripheral_test
 5. Examine results – serial logs
 6. Terminate and remove debug process!

Serial log in VITIS terminal.

```
Connected to COM18 at 115200
---Entering main---
  Running ScuGicSelfTestExample() for ps7_scugic_0...
ScuGicSelfTestExample PASSED
ScuGic Interrupt Setup PASSED

Running GpioInputExample() for dip...
GpioInputExample PASSED. Read data:0xD

Running GpioInputExample() for pb...
GpioInputExample PASSED. Read data:0x4

  Running DcfgSelfTestExample() for ps7_dev_cfg_0...
DcfgSelfTestExample PASSED
.....
---Exiting main---
```

LAB02 – Summary

- To the ARM-AXI base system created in the previous (4. – LAB01), we added two PL-side **AXI GPIO** peripherals from the **Vivado** IP catalog.
- Peripherals were properly configured and connected to the external I/O pins of the FPGA.
- We examined both the Block Diagram and the report files.
- **The DIP switches (4) and PB pushbuttons (4)** on the ZyBo card have been assigned to the pin assignments.
- Finally, we verified the completed embedded system (HW+FW) and the correct operation of the SW application (**TestApp**, and **Peripheral Test**) in **VITIS** unified environment.



EFOP-3.4.3-16-2016-00009

A felsőfokú oktatás minőségének és hozzáférhetőségének
együttes javítása a Pannon Egyetemen

THANK YOU FOR YOUR KIND ATTENTION!

SZÉCHENYI  2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Strukturális
és Beruházási Alapok



BEFEKTETÉS A JÖVŐBE

S

- X