



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

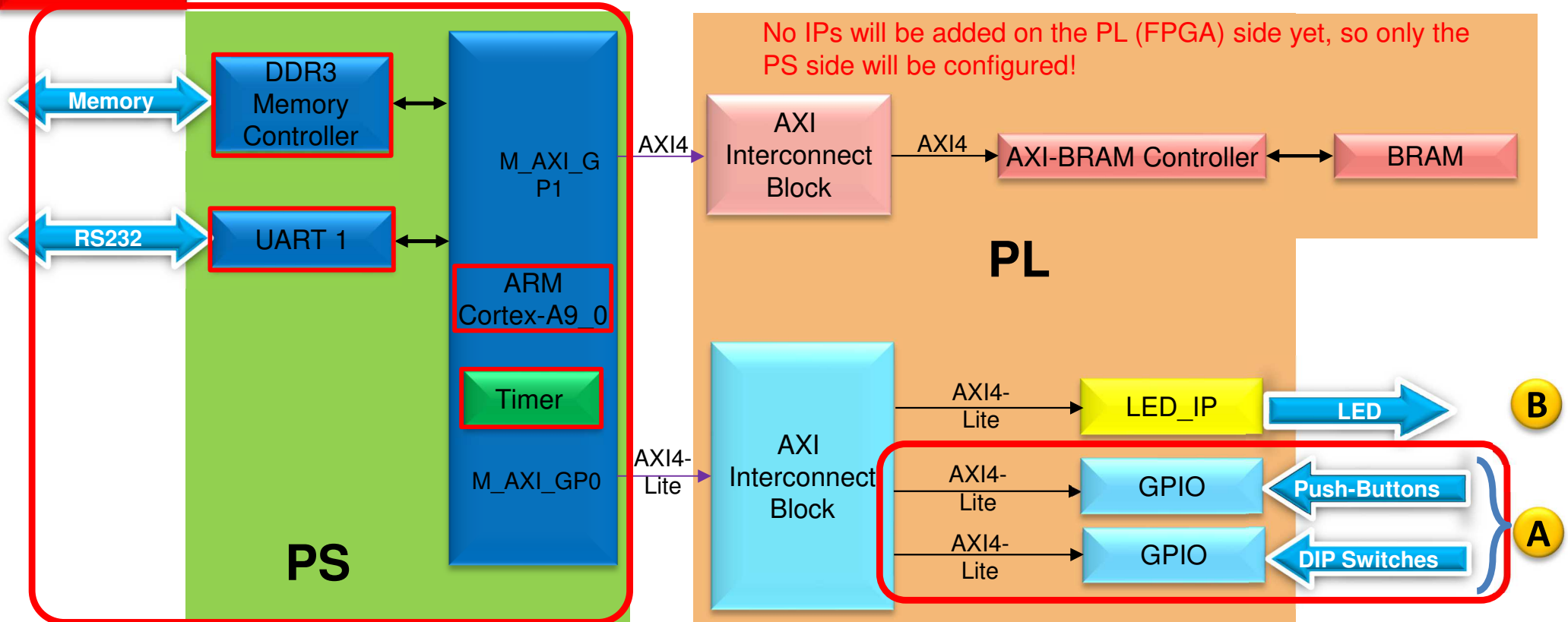


**BEFEKTETÉS A JÖVŐBE**



# 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)

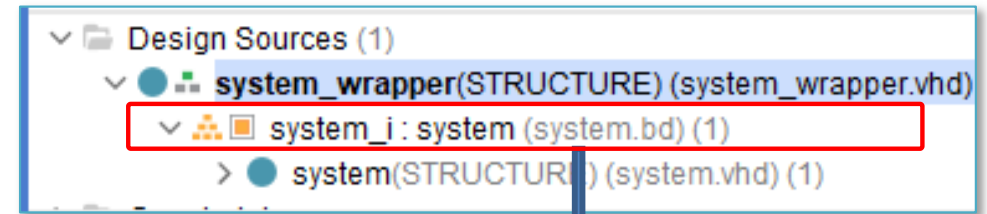
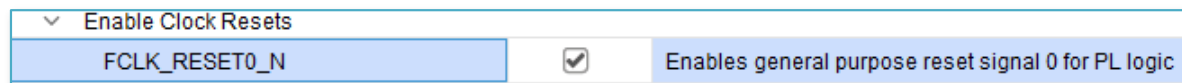
# Modify Zynq PS settings

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

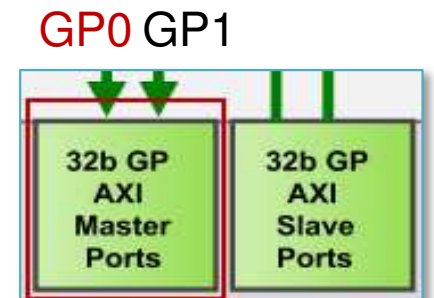
- Open Processing\_system7\_0

- PS-PL configuration:

- > General → Enable Clock Resets:  
enable **FCLK\_RESET0\_N**

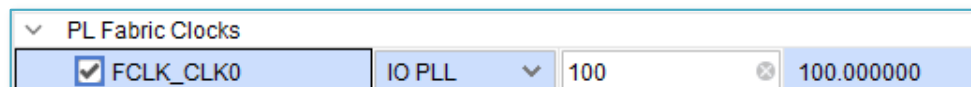


- > AXI Non Secure Enablement: GP Master AXI interface  
enable **M\_AXI\_GP0** port!



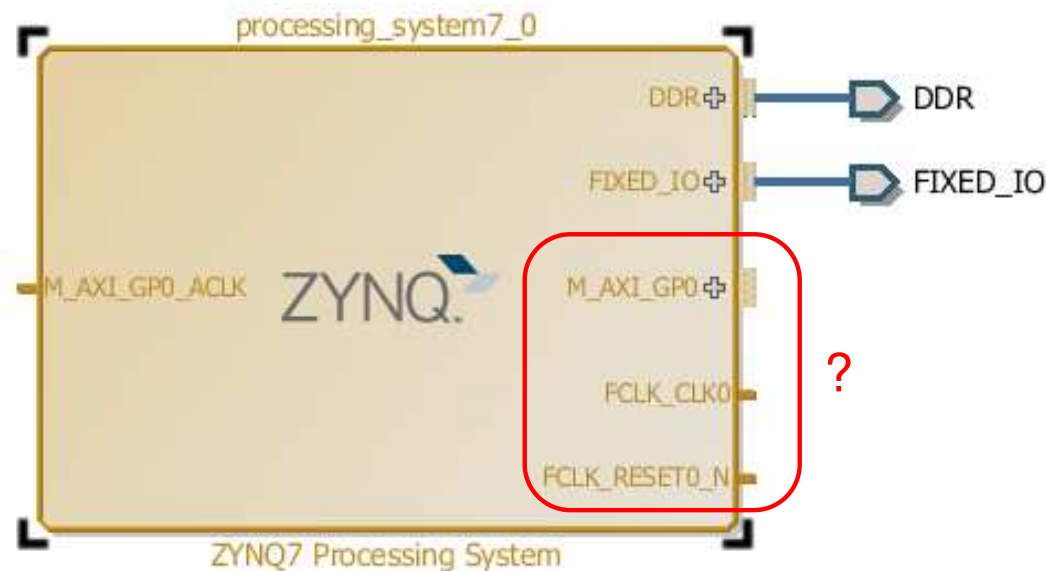
- Clock configuration:

- PL Fabric Clocks: enable **FCLK\_CLK0** ( 100 MHz IO\_PLL)



# Zynq PS – Block diagram



- Examine, that the previously enabled ports:
  - GP0 AXI Master interface,
  - FCLK\_CLK0 PL-side clock,
  - FCLK\_RESET0\_N reset portare visible now?

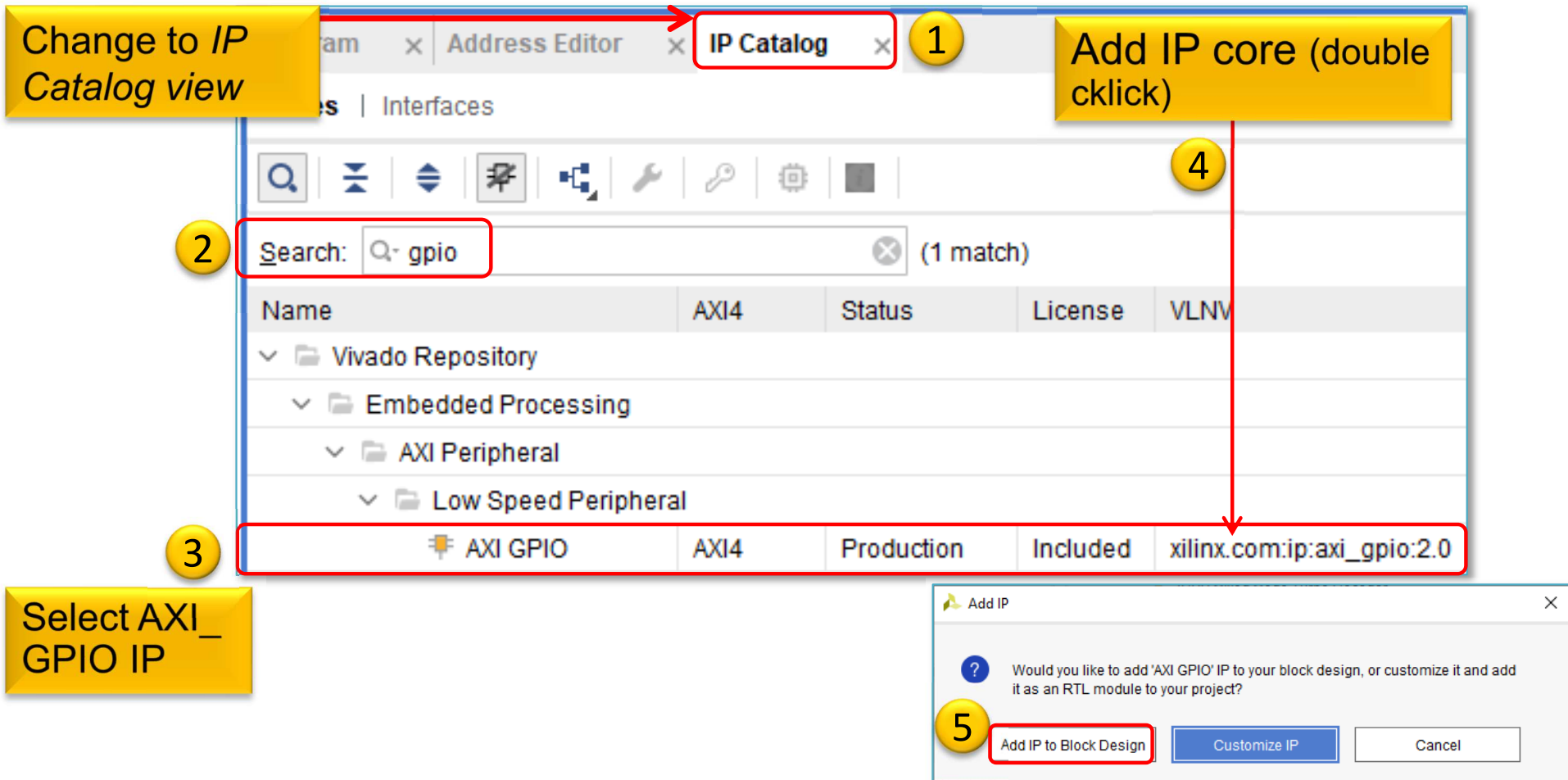


- What do you think about their functionality?

# 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 illustrates the process of adding an AXI\_GPIO IP core to a Vivado project. The IP Catalog window is open, showing a search for 'gpio' which yields one match: 'AXI GPIO'. The 'AXI GPIO' entry is selected, and the 'Add IP' dialog box is displayed, asking if the user wants to add the IP to the block design or customize it. The 'Add IP to Block Design' button is highlighted.

**Change to IP Catalog view**

**Add IP core (double click)**

**2** Search:  (1 match)

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

**3** Select AXI\_GPIO IP

**5** Add IP to Block Design

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?

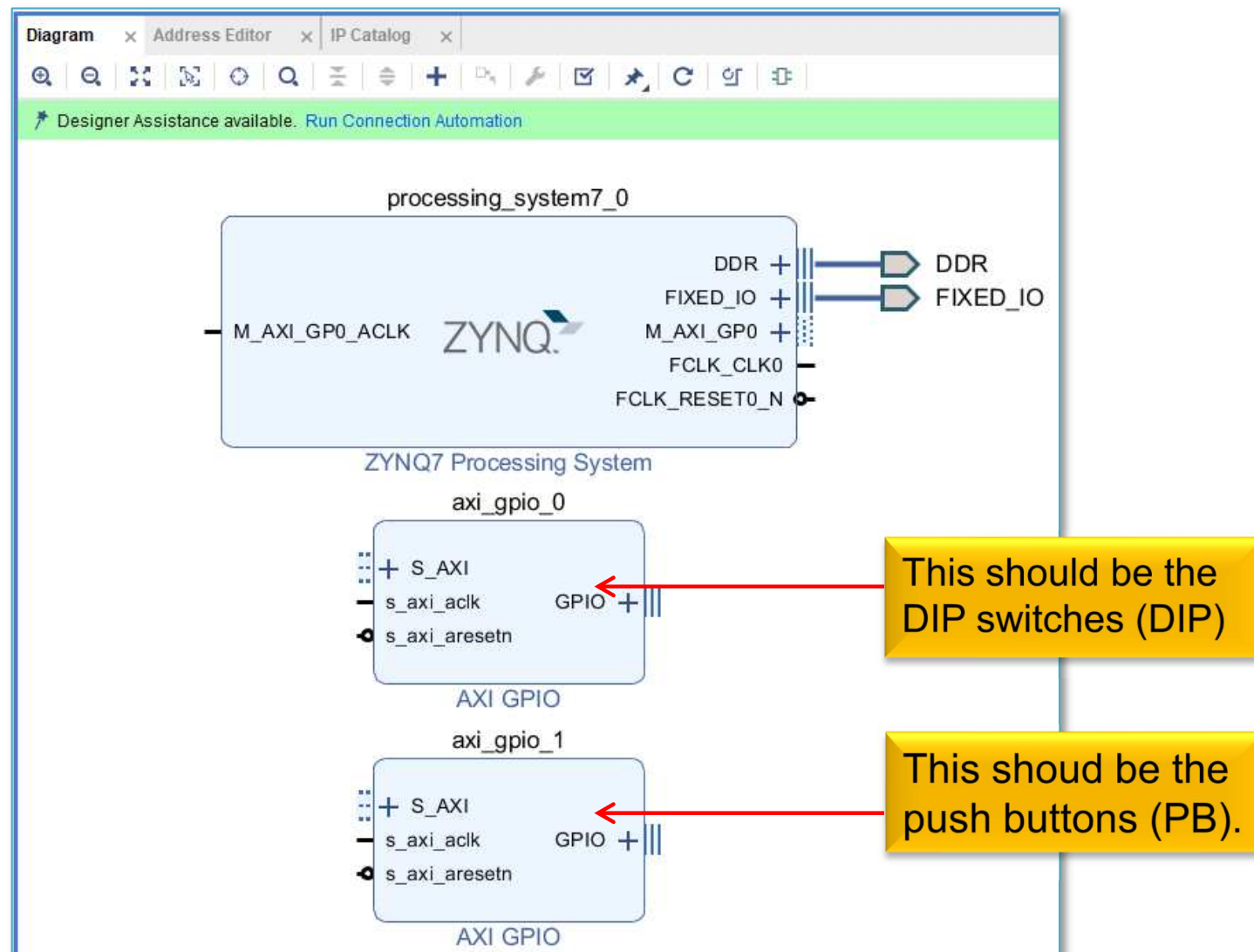
Buttons: Add IP to Block Design, Customize IP, Cancel

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

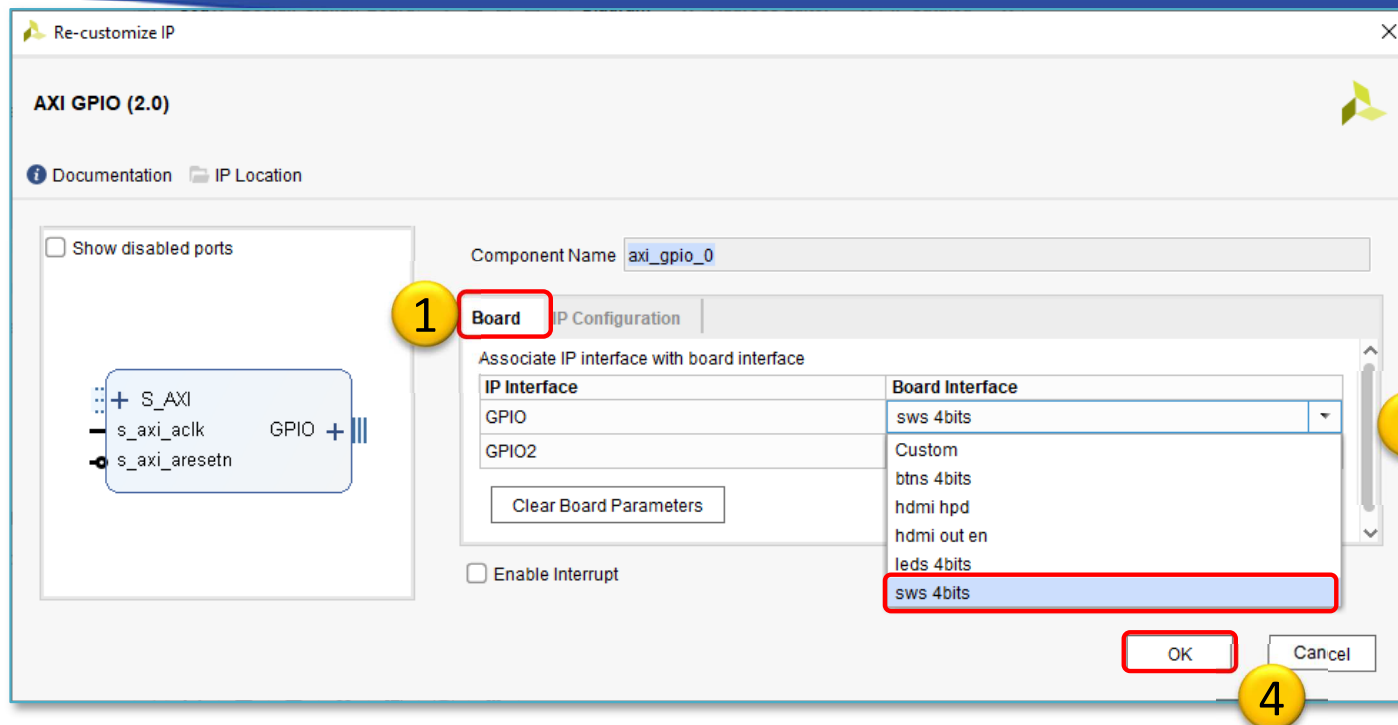
- For each IP modul (e.g. **AXI\_GPIO**) the following should be set:
  - a.) ***interface connection*** between the IP modul and bus system (AXI),
  - b.) **mapping** IP modul to the PS **address space** (Base-High Addresses),
  - c.) assigning I/O ports of IP modules **to external ports**,
  - d.)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

# Adding GPIOs – Block Diagram view

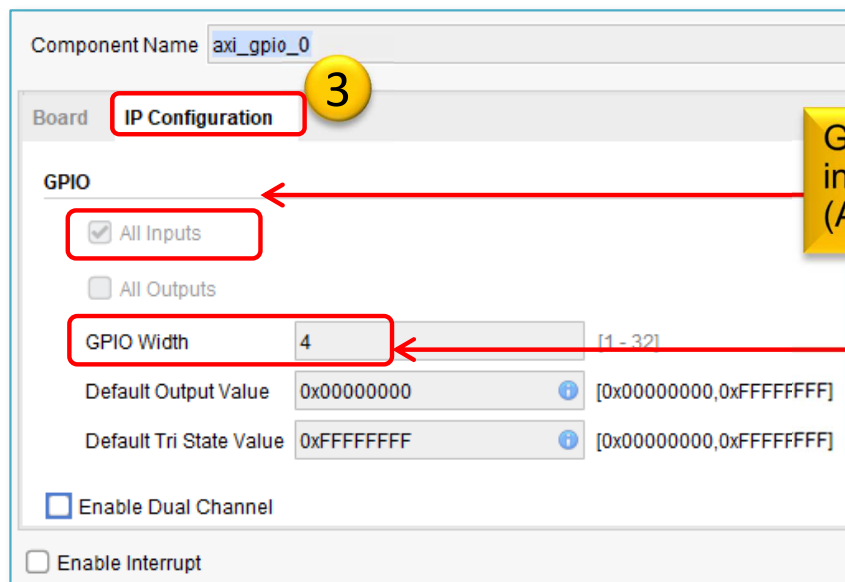
AXI\_GPIO\_0 as DIP switches (DIP) and AXI\_GPIO\_1 as push buttons (PB).



# Set AXI\_GPIO peripheral - Dip switches



Board interface:  
select „sws\_4bits”.

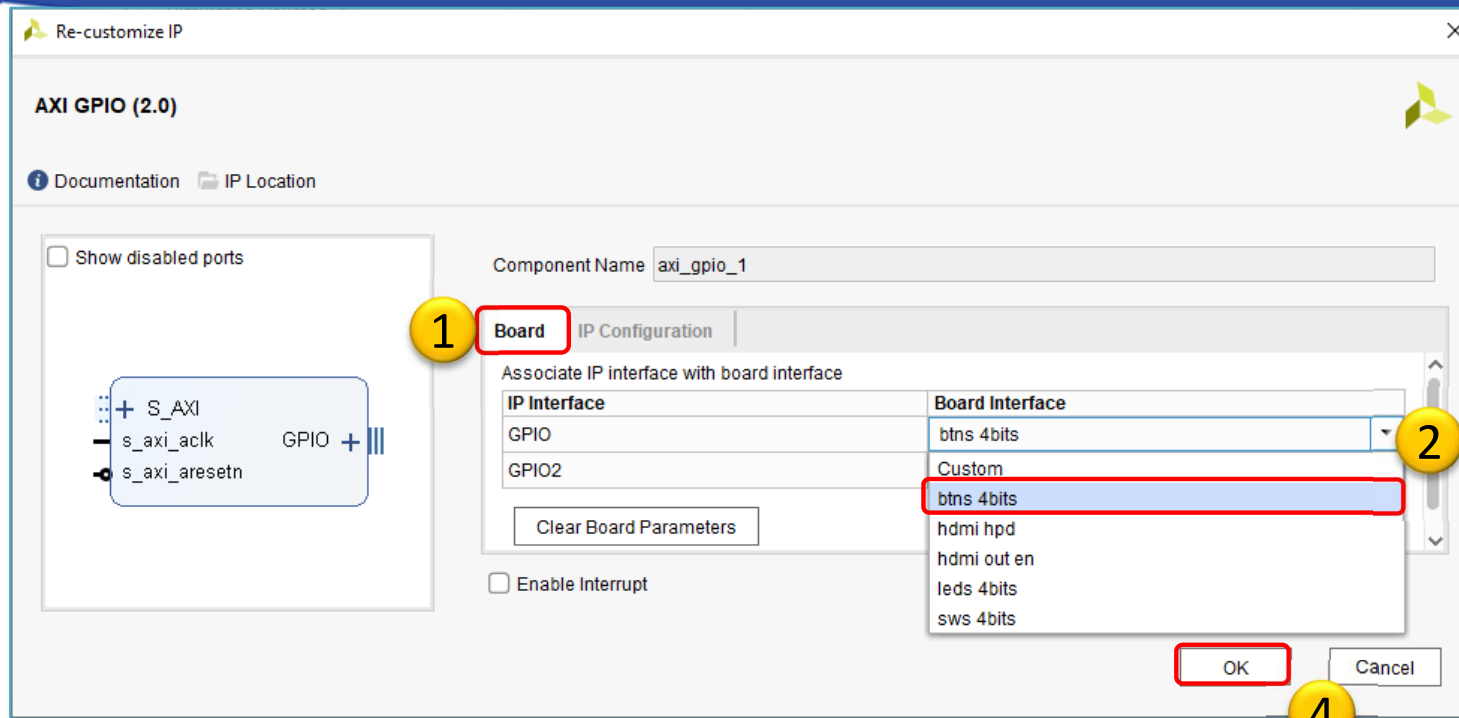


GPIO channel only  
inputs for dip switches  
(All inputs)

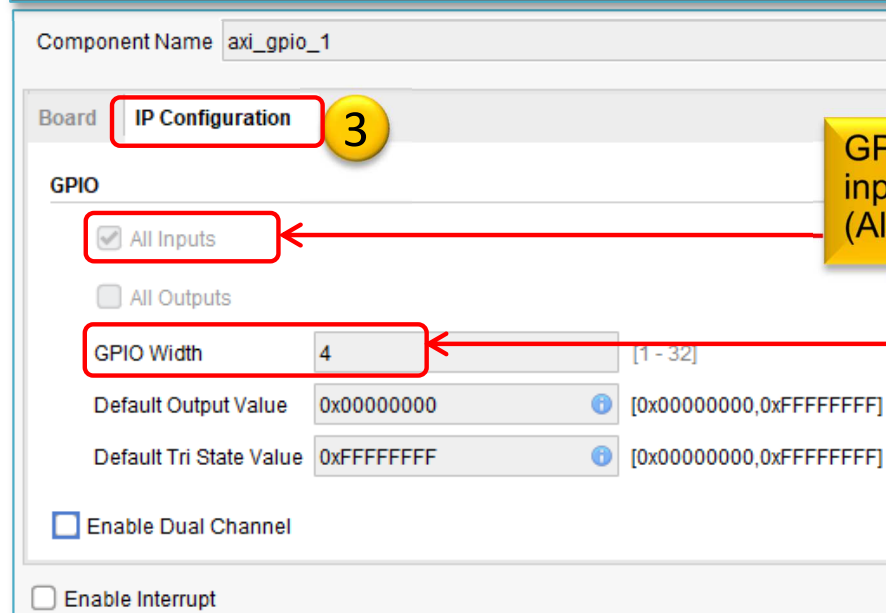
Check GPIO channel width = 4  
(because ZyBo has 4 dip  
switches)



# Set AXI\_GPIO peripheral - Push buttons



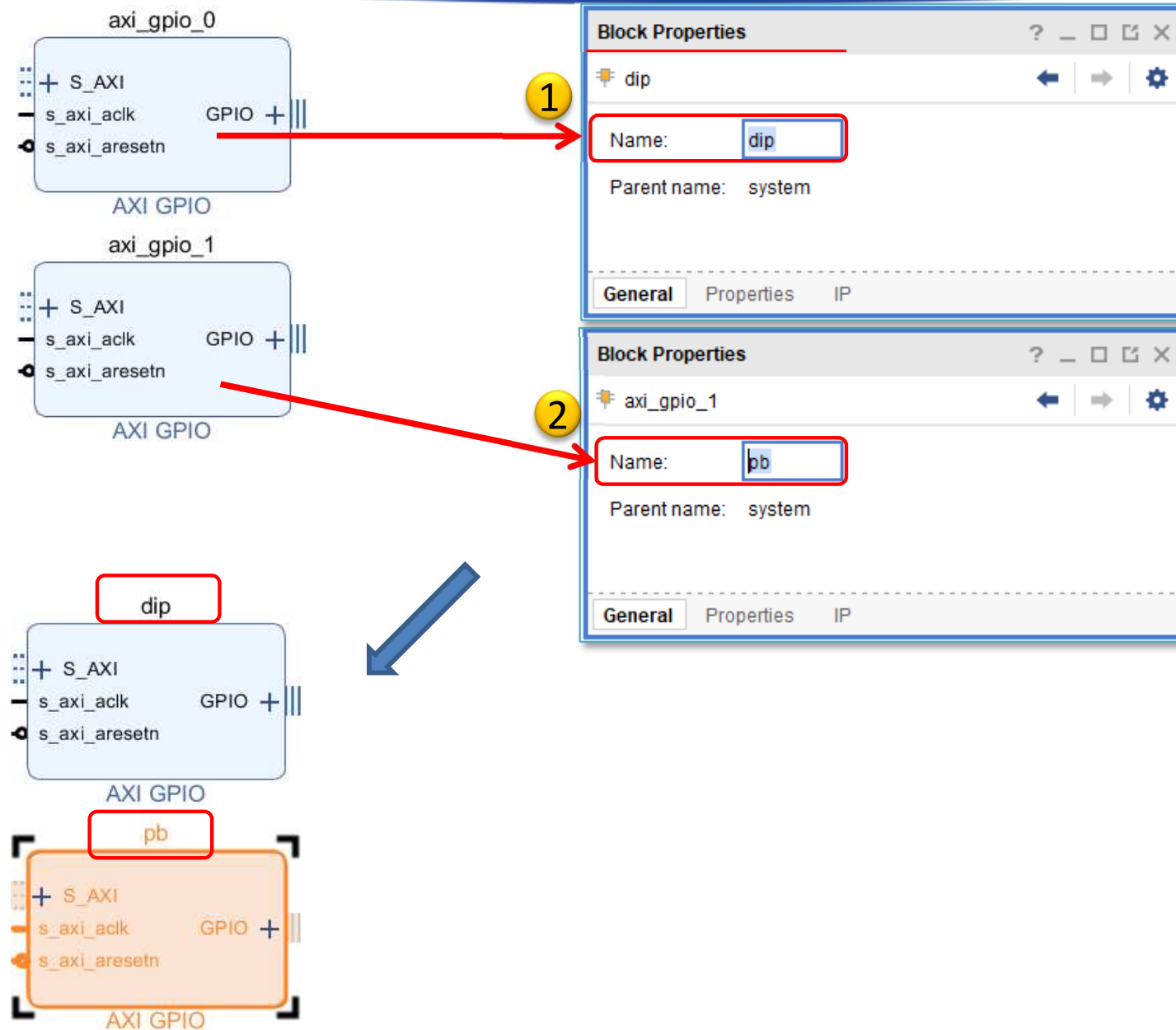
Board interface:  
select „btns\_4bits”.



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

1 Run Connection Automation

processing\_system7\_0

M\_AXI\_GP0\_ACLK ZYNQ

DDR + DDR

FIXED\_IO + FIXED\_IO

M\_AXI\_GP0 +

FCLK\_CLK0

FCLK\_RESET0\_N

2 3

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.

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

4

Clock source for driving Interconnect IP Auto

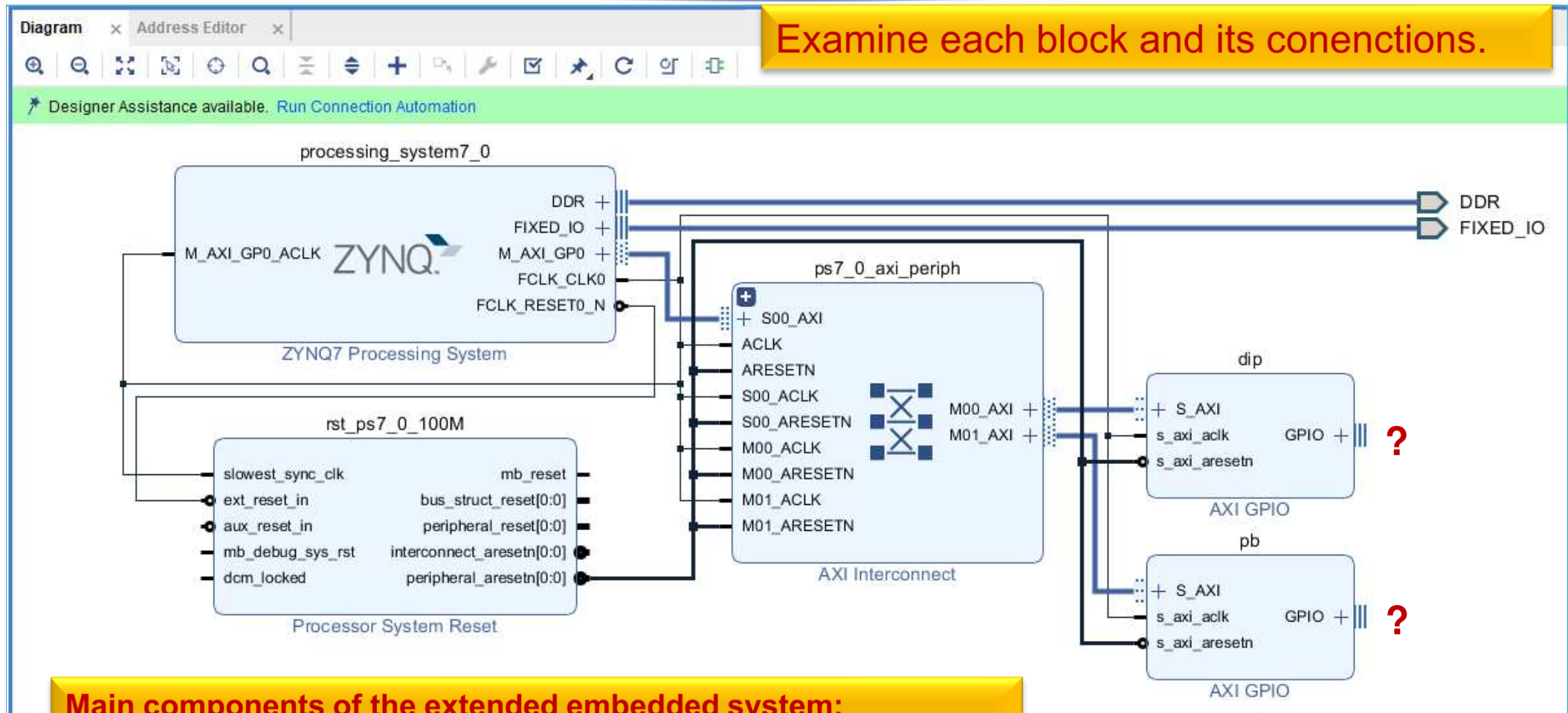
Clock source for Master interface Auto

Clock source for Slave interface Auto

OK Cancel

DIP → enable S\_AXI interface,  
PB → enable S\_AXI interface

# Block Design – Analyzis



## Main components of the extended embedded system:

- Processing\_system7: (PS ARM processor system)
- Processing\_system7\_axi\_periph: AXI bus interface
- Rst\_processing\_system7\_100M: PS/PL reset generator
- dip: AXI GPIO for DIP switches (PL side)
- pb: AXI GPIO for PB push buttons (PL side)

# 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 Integrator. When the endpoint IPs are attached in width, clock or protocol. If a converter IP is inserted, it configures the converter to the appropriate conversion. To see which conversion is used, click the 'expand hierarchy' buttons.

NOTE: Addressing information for the slave and master interfaces is defined in the component's configuration file.

☐ Enable Advanced Configuration

**2** **Slave Interfaces**

Slave Interface	Enable Register Slice	Enable Data FIFO
S00_AXI	None	None

**3** **Master Interfaces**

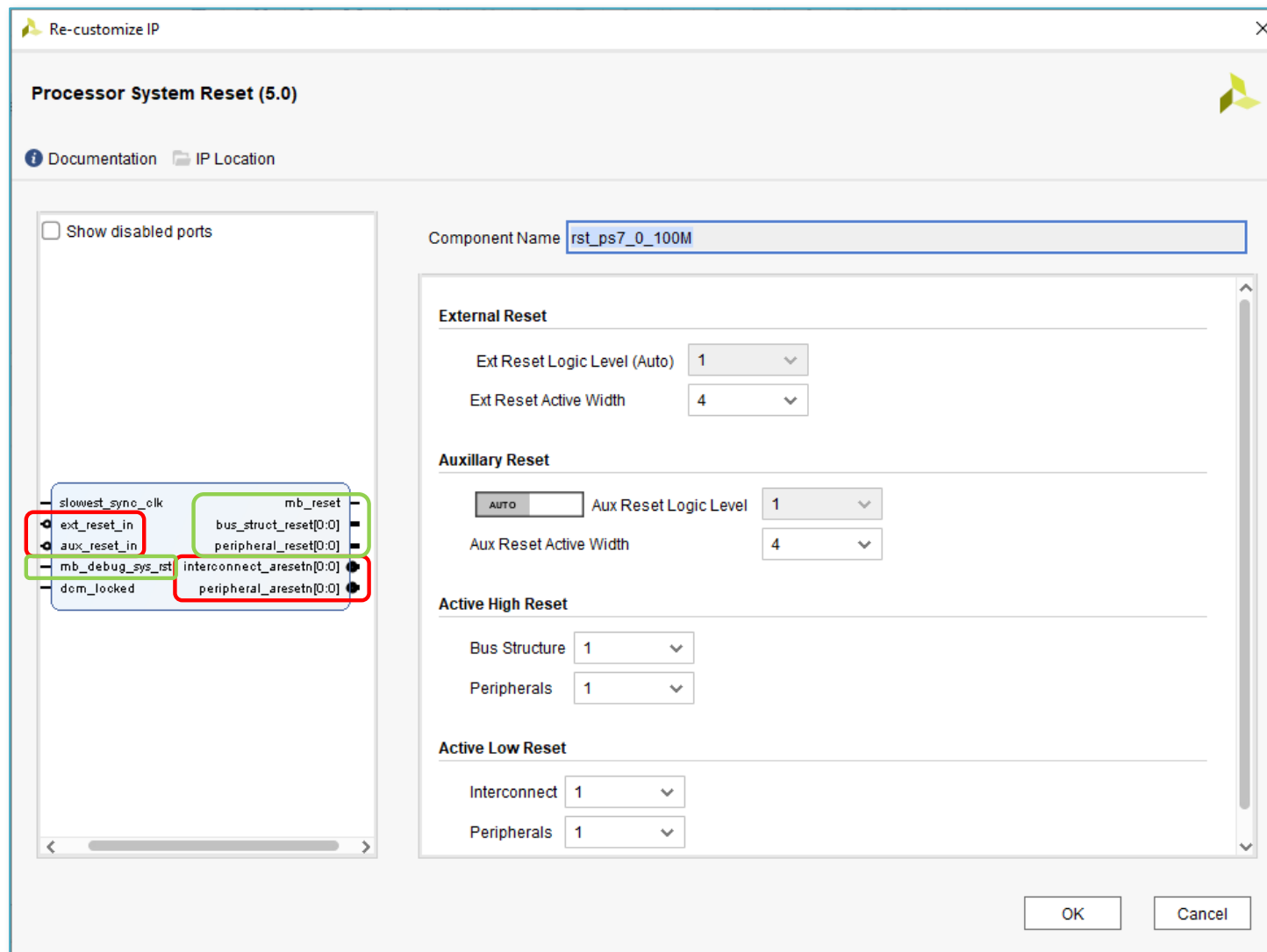
Master Interface	Enable Register Slice	Enable Data FIFO
M00_AXI	None	None
M01_AXI	None	None

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?





# 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)

b.) Base address manual set \*  
pb: 0x4121\_0000 (64K)  
dip: 0x4120\_0000 (64K)

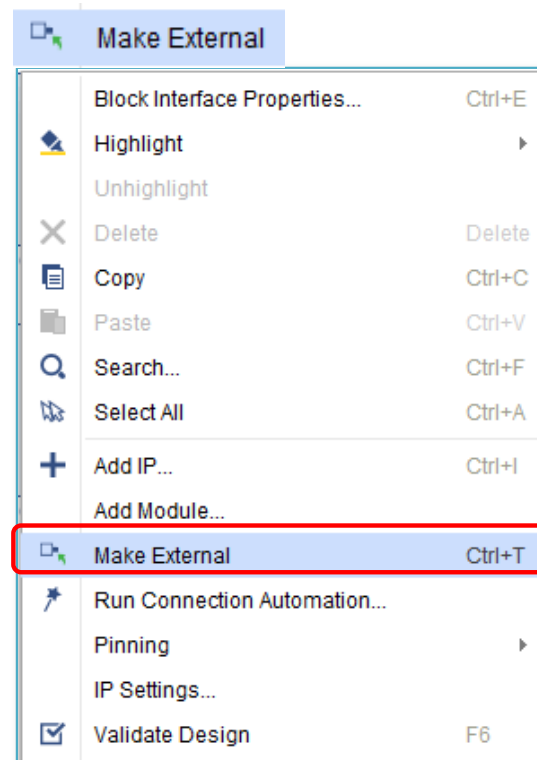
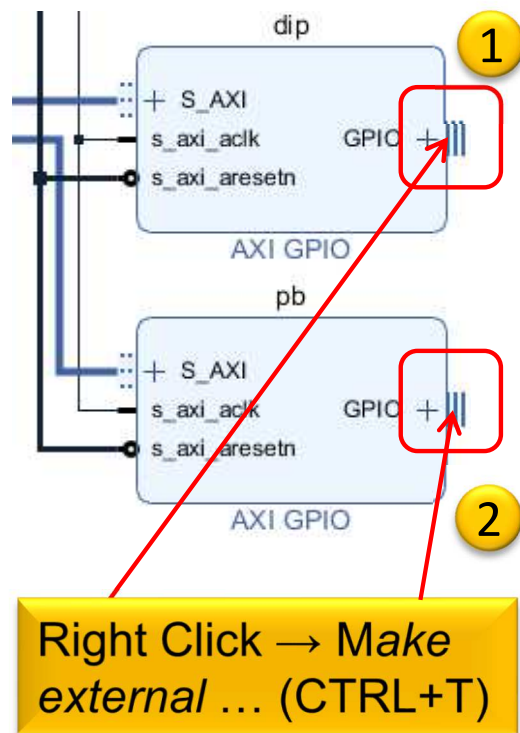
Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/processing_system7_0					
/processing_system7_0/Data (32 address bits : 0x40000000 [ 1G ])					
/dip	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
/pb	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF

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

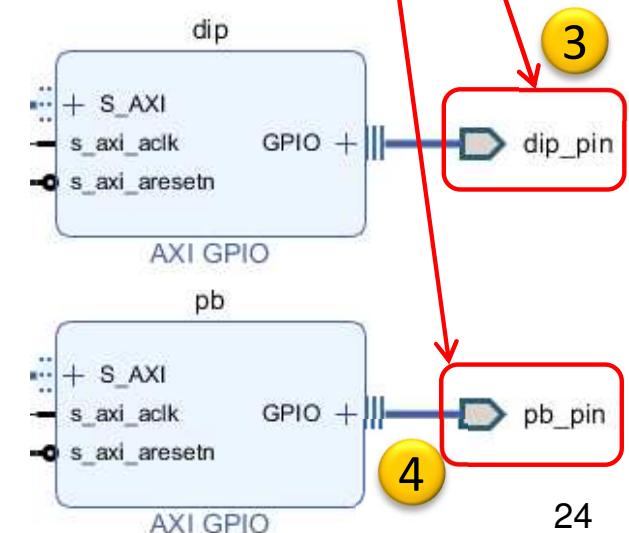
# Making external ports – AXI GPIO

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.



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

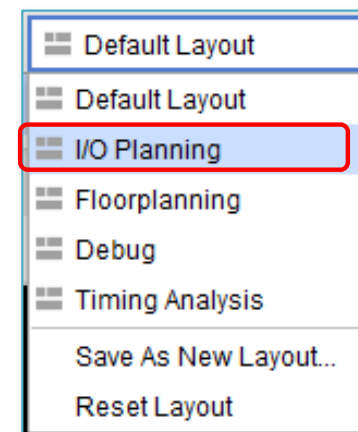


Run Synthesis

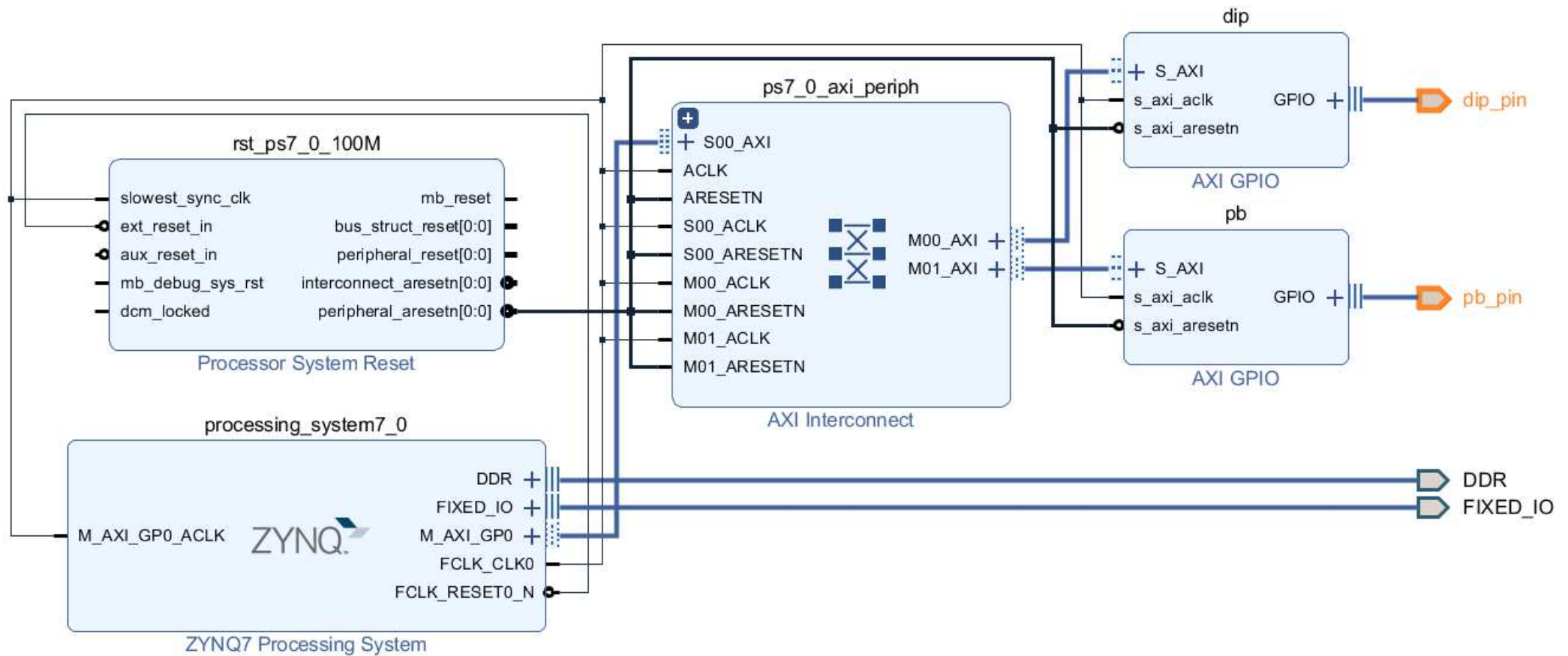
- Open Synthesized Design, OK

- 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.

**I/O Ports**

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	LVCMOS33*	3.300				NONE	NONE
pb_dip_tri_i (4)	IN					<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300				NONE	NONE
pb_dip_tri_i[3]	IN	btns_4bits_tr...			Y16	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300				NONE	NONE
pb_dip_tri_i[2]	IN	btns_4bits_tr...			V16	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300				NONE	NONE
pb_dip_tri_i[1]	IN	btns_4bits_tr...			P16	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300				NONE	NONE
pb_dip_tri_i[0]	IN	btns_4bits_tr...			R18	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300				NONE	NONE
Scalar ports (0)														
GPIO_51240 (4)	IN					<input checked="" type="checkbox"/>	(Multiple)	LVCMOS33*	3.300				NONE	NONE
dip_pin_tri_i (4)	IN					<input checked="" type="checkbox"/>	(Multiple)	LVCMOS33*	3.300				NONE	NONE
dip_pin_tri_i[3]	IN	sws_4bits_tr...			T16	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300				NONE	NONE
dip_pin_tri_i[2]	IN	sws_4bits_tr...			W13	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300				NONE	NONE
dip_pin_tri_i[1]	IN	sws_4bits_tr...			P15	<input checked="" type="checkbox"/>	34	LVCMOS33*	3.300				NONE	NONE
dip_pin_tri_i[0]	IN	sws_4bits_tr...			G15	<input checked="" type="checkbox"/>	35	LVCMOS33*	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)

1

2

3

Physical Constraints

Device

Netlist

Project Summary

Device

system\_wrapper.vhd

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

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

system\_wrapper

Nets (146)

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

Power

Methodology

I/O Ports

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.

General Information

Timer Settings

Design Timing Summary

Clock Summary (1)

Check Timing (8)

Intra-Clock Paths

Inter-Clock Paths

Other Path Groups

User Ignored Paths

# XILINX VIVADO DESIGN SUITE

LAB02\_A. ANALYZING BLOCK DIAGRAM AND REPORTS

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?

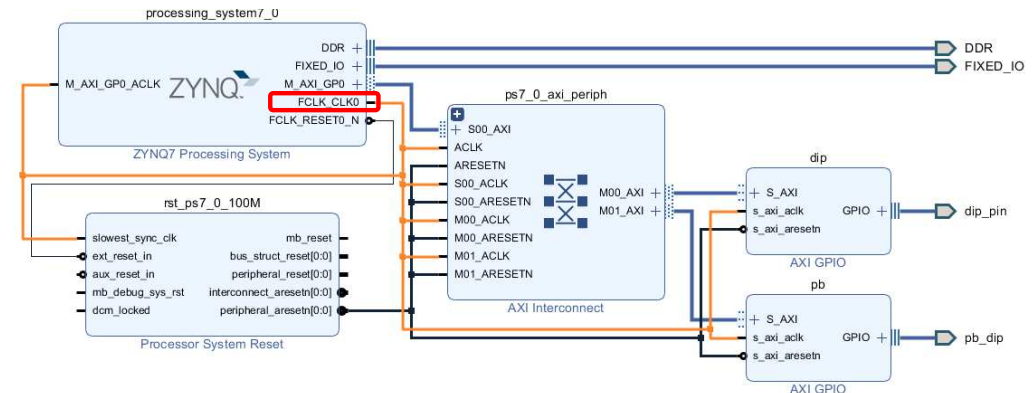


# Analyzing Block Diagram (cont.)

## • 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`

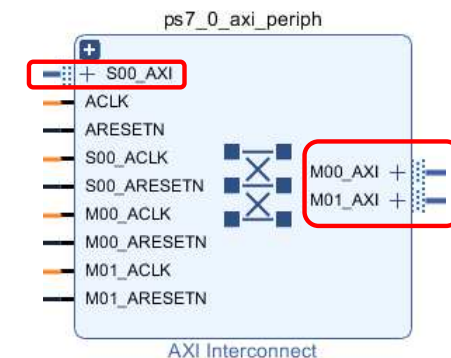


– Clock `processing_system7_0_FCLK_CLK0` is:

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

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

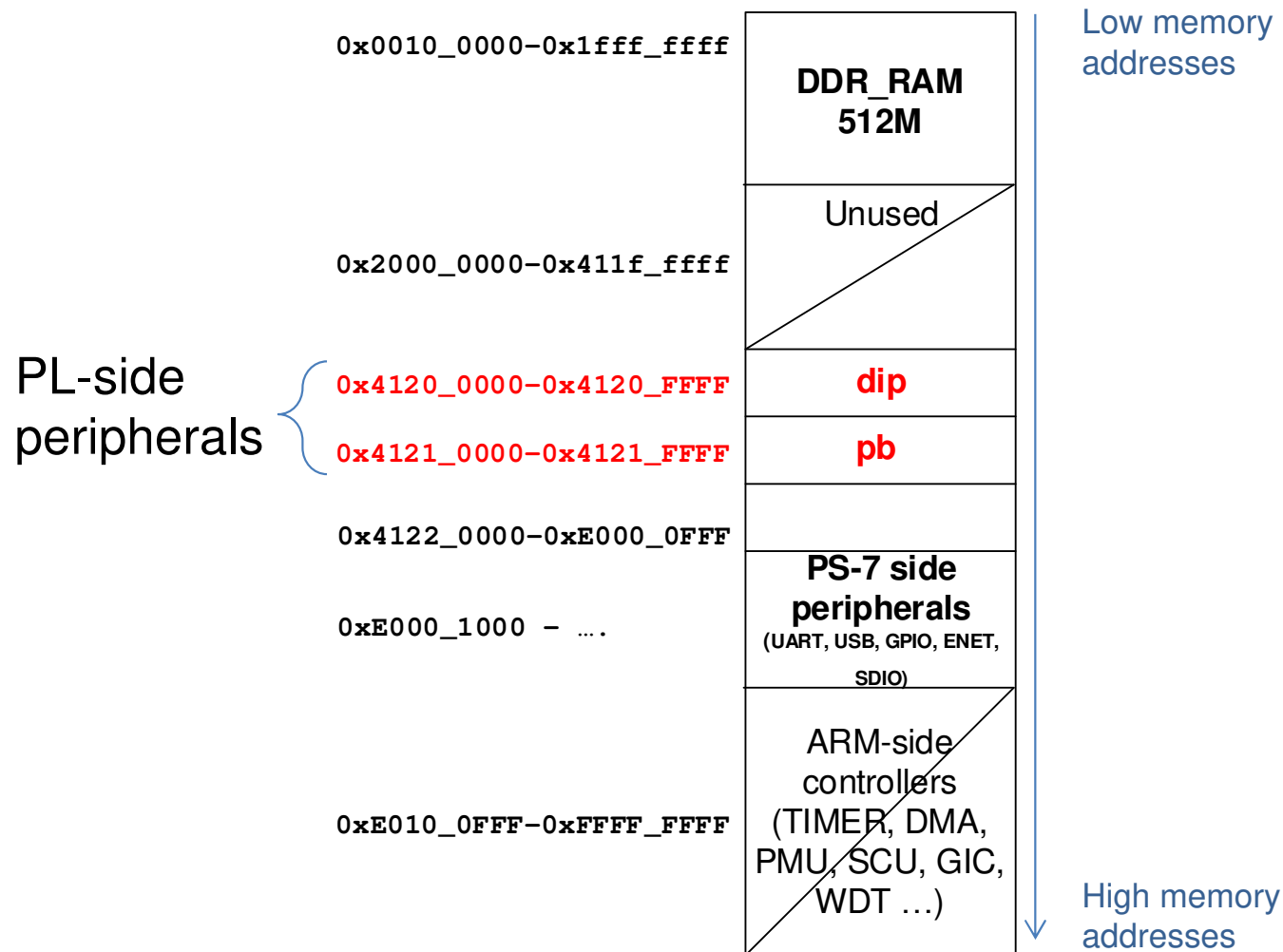
- `processing_system7`
- `dip`
- `pb`



# 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?

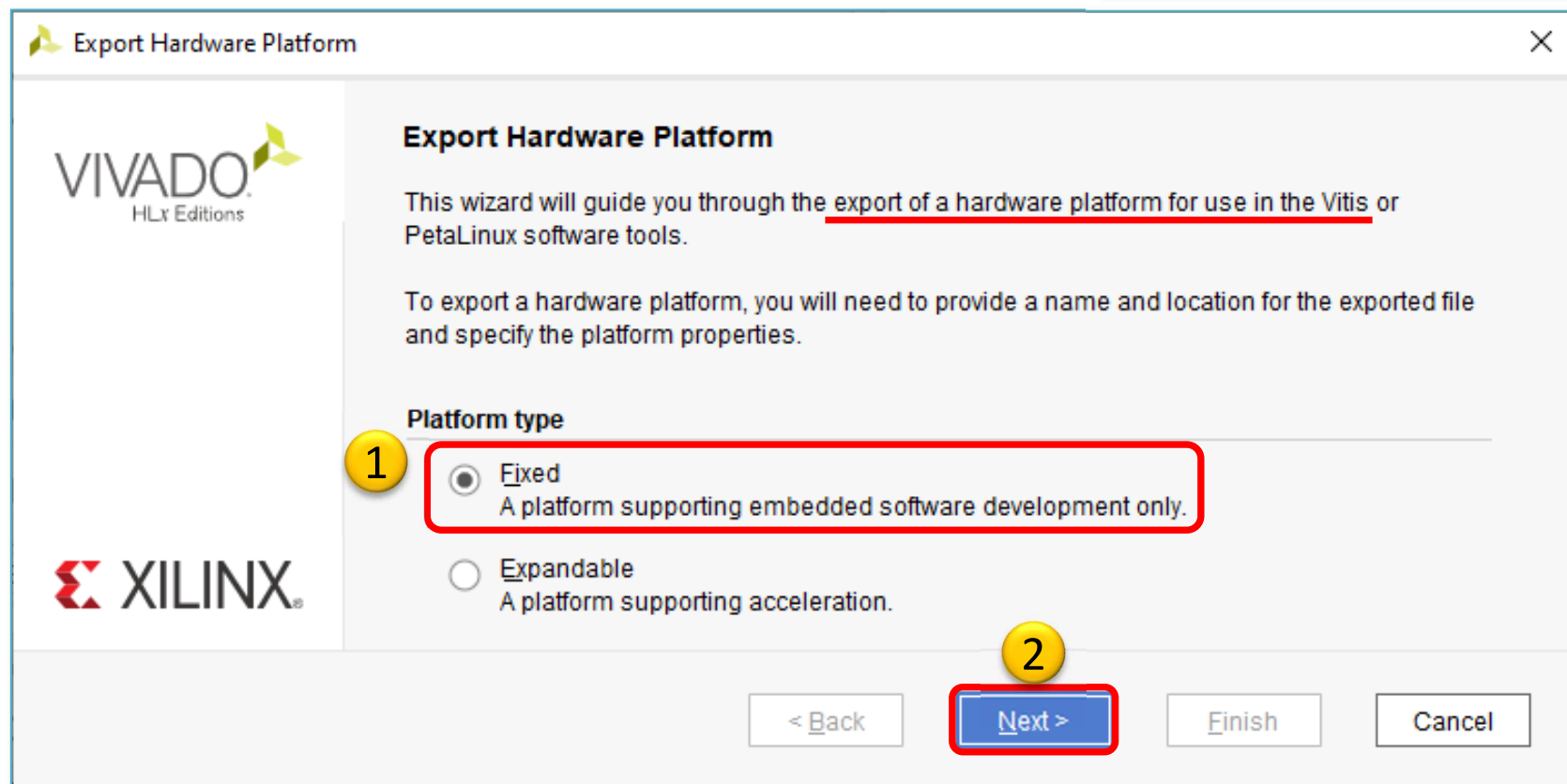
- Reports tab → Report Utilization (vagy Project Summary )

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)

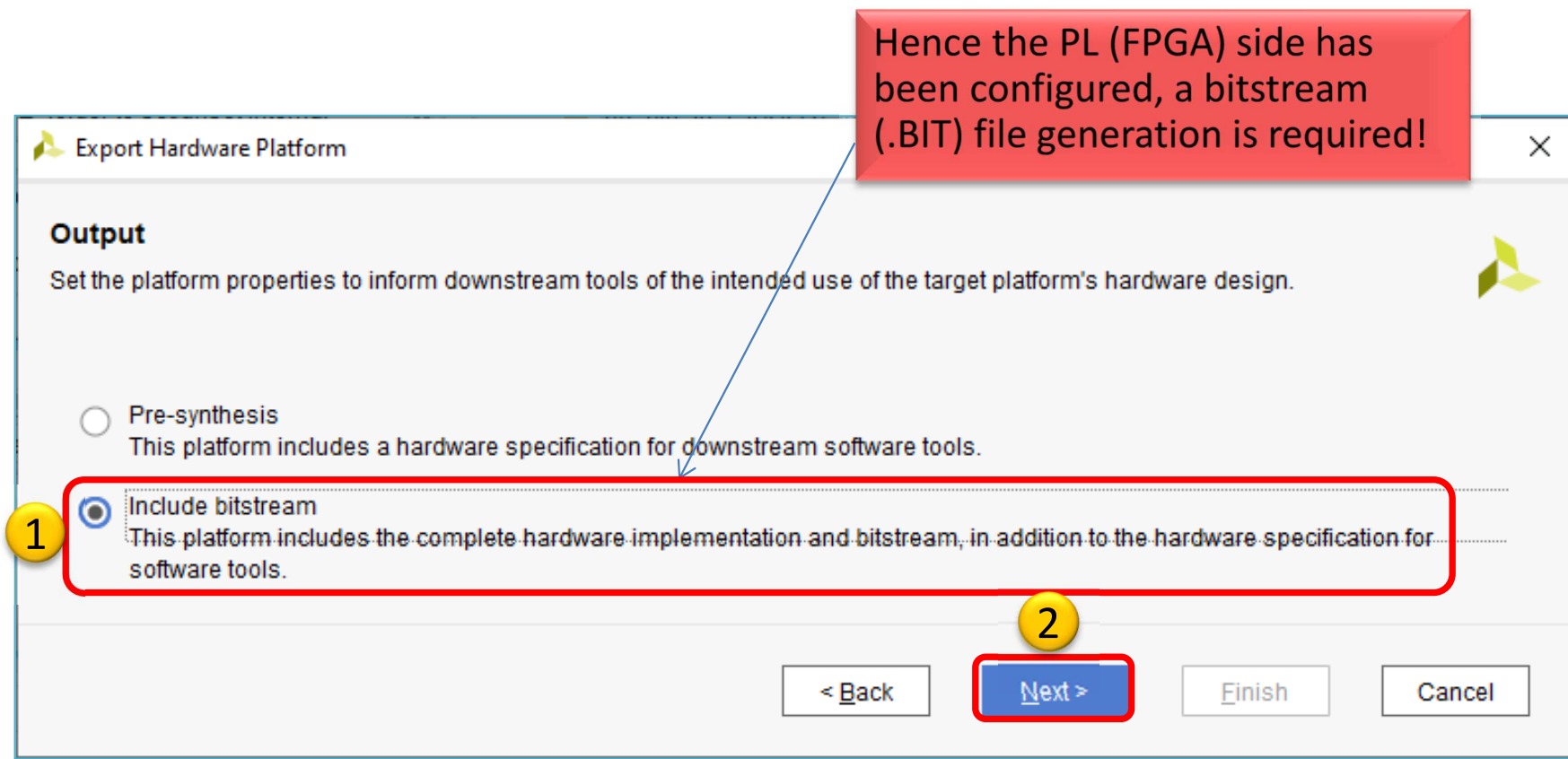
- **File → Export → Export Hardware...**

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



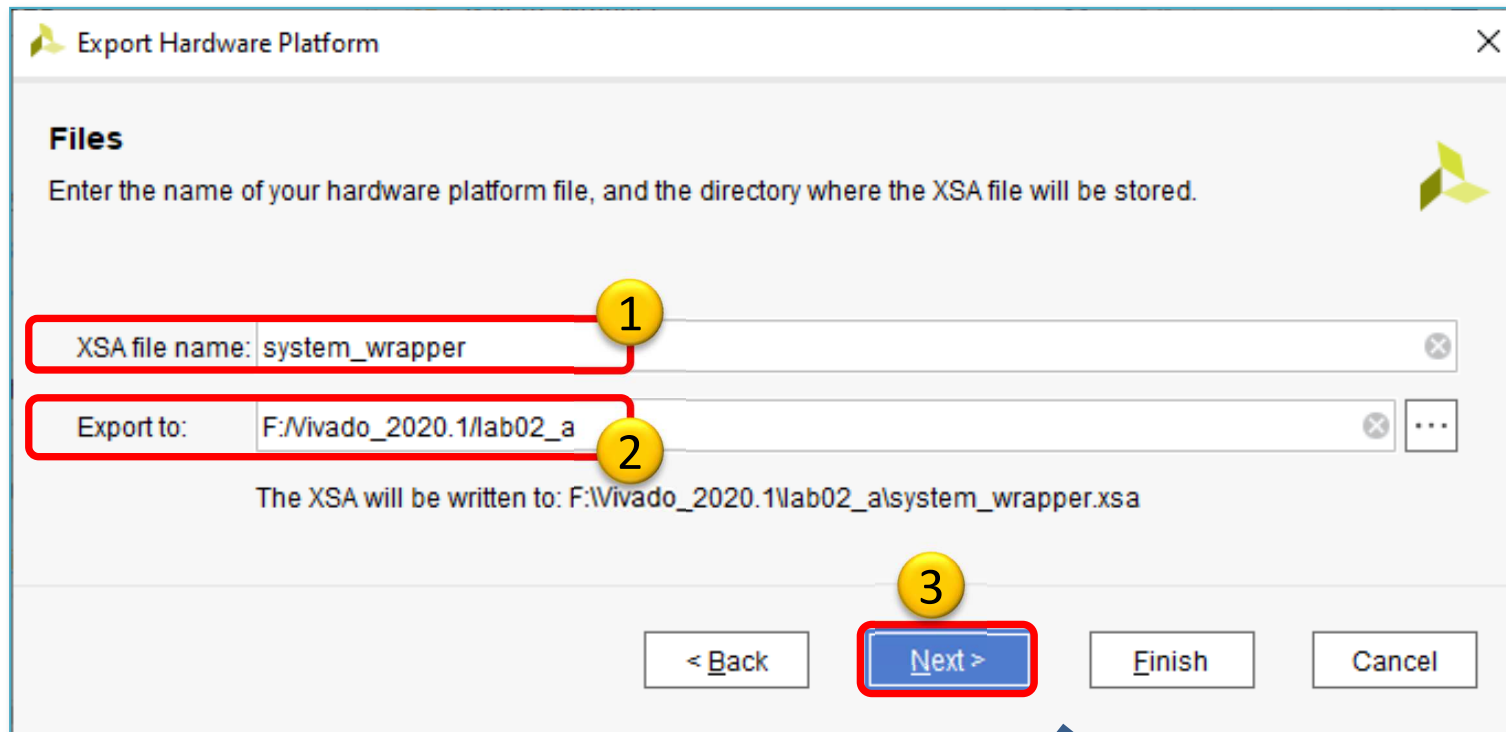
# VIVADO Export HW → VITIS (cont.)

Select „Include bitstream” option as output:

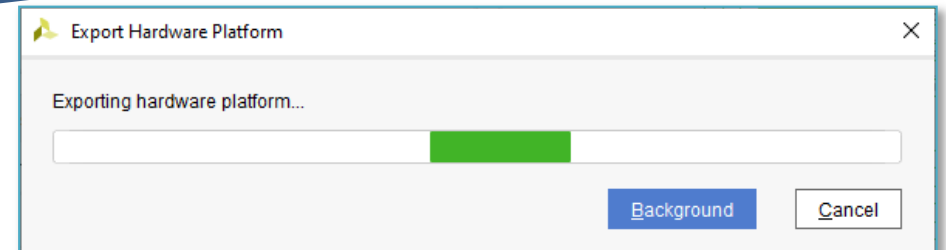


# Export HW → VITIS (cont.)

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



The dialog box titled "Export Hardware Platform" contains a "Files" section with the instruction: "Enter the name of your hardware platform file, and the directory where the XSA file will be stored." It features two input fields: "XSA file name:" with the value "system\_wrapper" (highlighted by a red box and a yellow circle with the number 1) and "Export to:" with the value "F:\Vivado\_2020.1\lab02\_a" (highlighted by a red box and a yellow circle with the number 2). Below these fields, a summary line states: "The XSA will be written to: F:\Vivado\_2020.1\lab02\_a\system\_wrapper.xsa". At the bottom, there are four buttons: "< Back", "Next >" (highlighted by a red box and a yellow circle with the number 3), "Finish", and "Cancel". A blue arrow points from the "Next >" button to the next dialog box.



The second dialog box, also titled "Export Hardware Platform", shows the progress of the export. It contains the text "Exporting hardware platform..." and a progress bar that is approximately 25% full (green). At the bottom right, there are two buttons: "Background" and "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



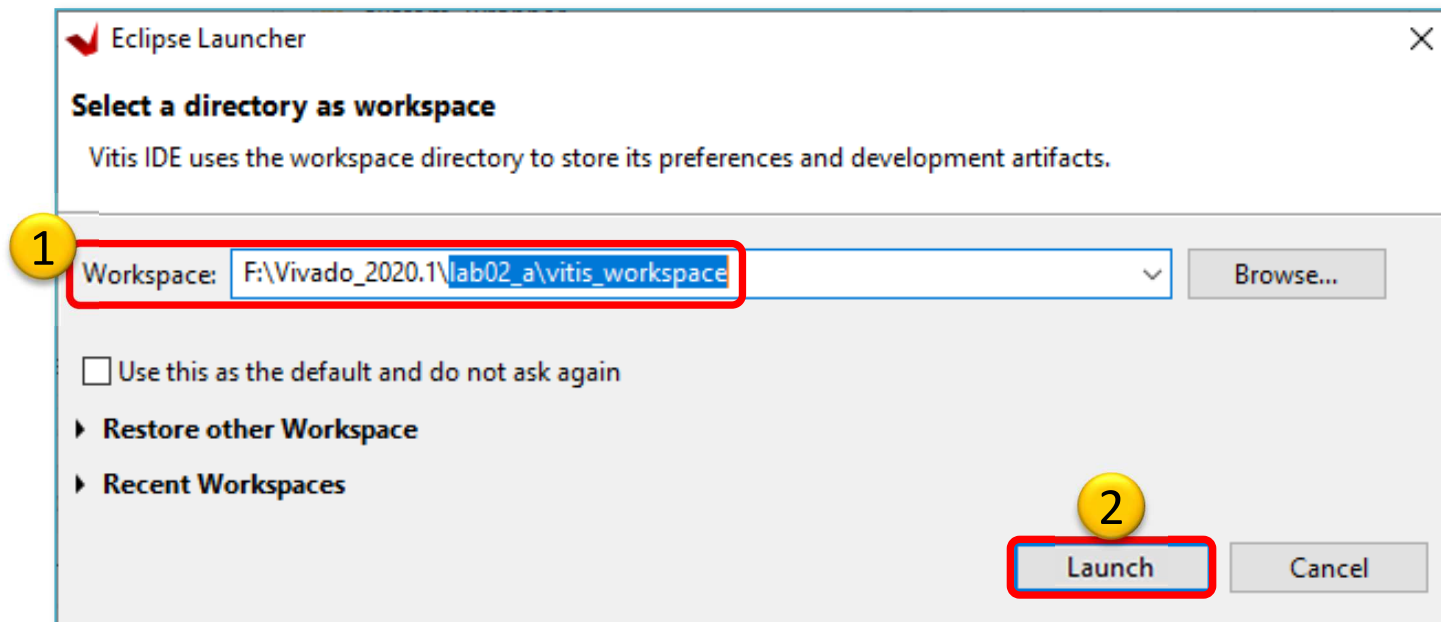
From Vivado: Tools menu → Launch VITIS IDE

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: **TestApp**

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

Select a system project  
+ Create new...

System project details  
2  
System project name: **Zybo\_test\_system**

Target processor  
Select target processor for the Application project.

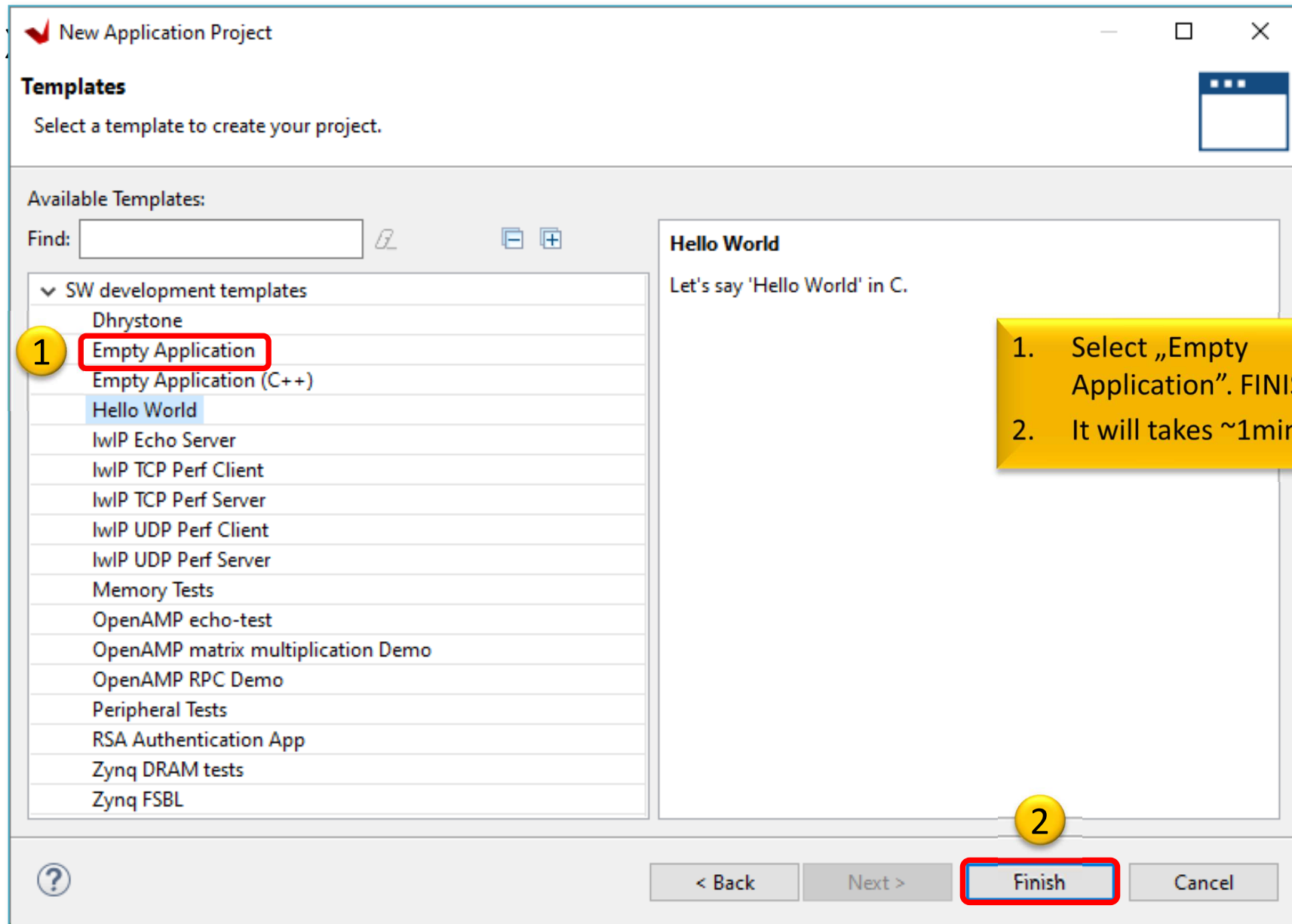
Processor	Associated applications
3 <b>ps7_cortexa9_0</b>	<b>TestApp</b>
ps7_cortexa9_1	
ps7_cortexa9 SMP	

Show all processors in the hardware specification ☒

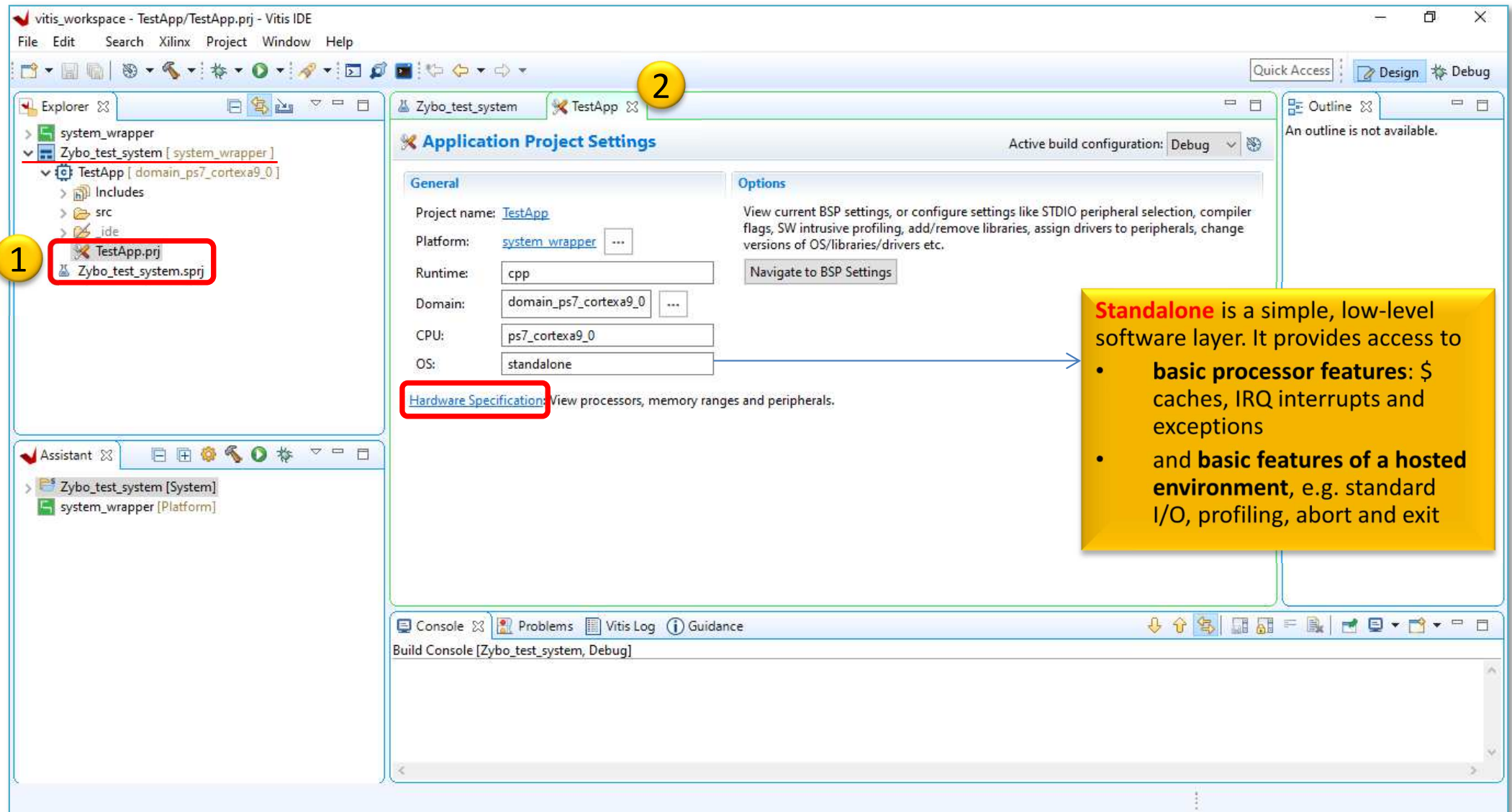
4  
< Back **Next >** Finish Cancel

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



# VITIS GUI – Main window



# VITIS – HW platform

**1** platform.spr

**2** system\_wrapper

**3** Build Console [Zybo\_test\_system, Debug]

**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!

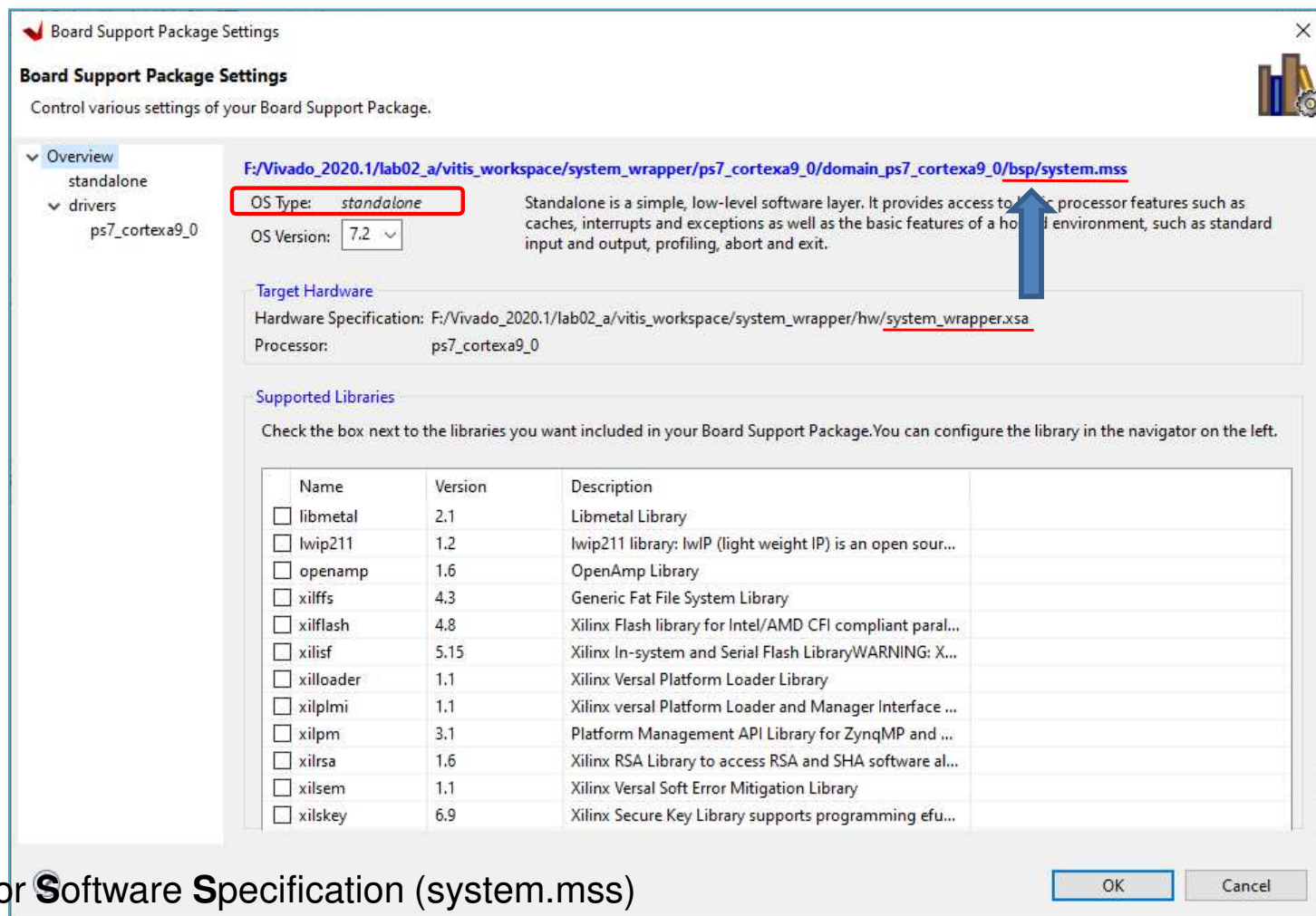
List and versions of used PS peripherals (below)

IP Instance	IP Type	IP Version	Register
dip	axi_gpio	2.0	<a href="#">Registers</a>
pb	axi_gpio	2.0	<a href="#">Registers</a>
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)



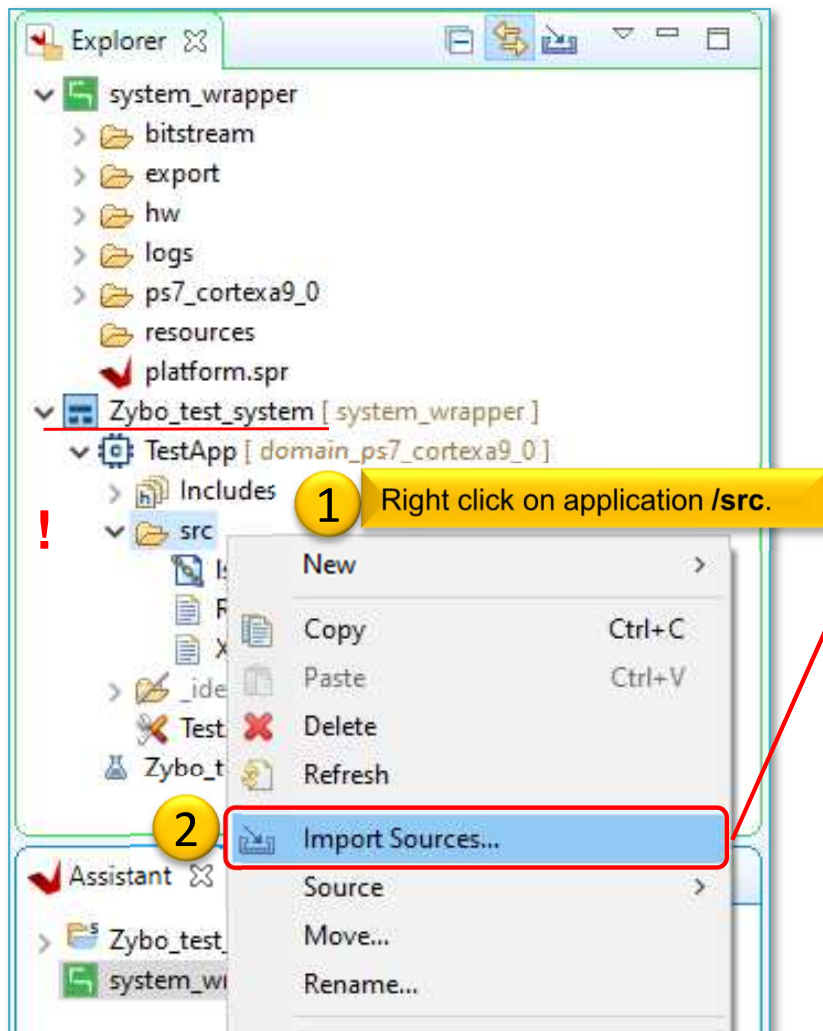
# Q & A 1.)

- **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

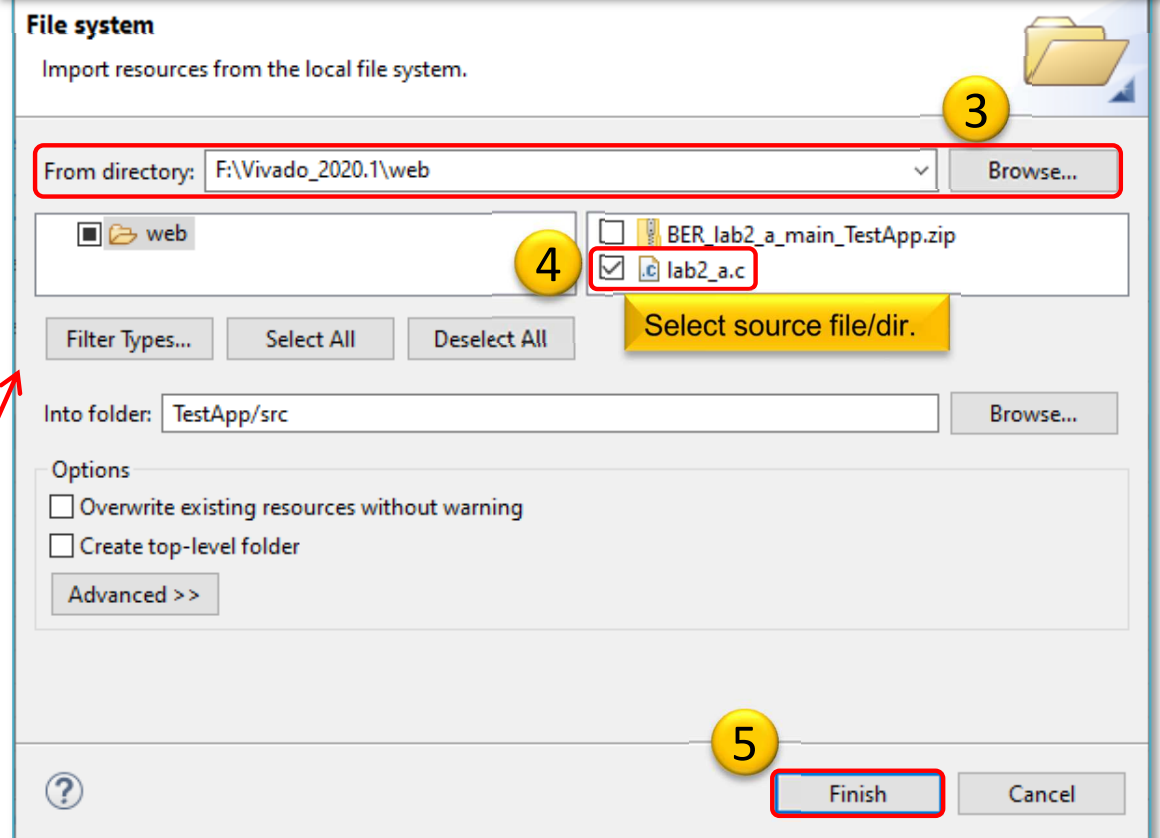


# Add C/C++ source(s)

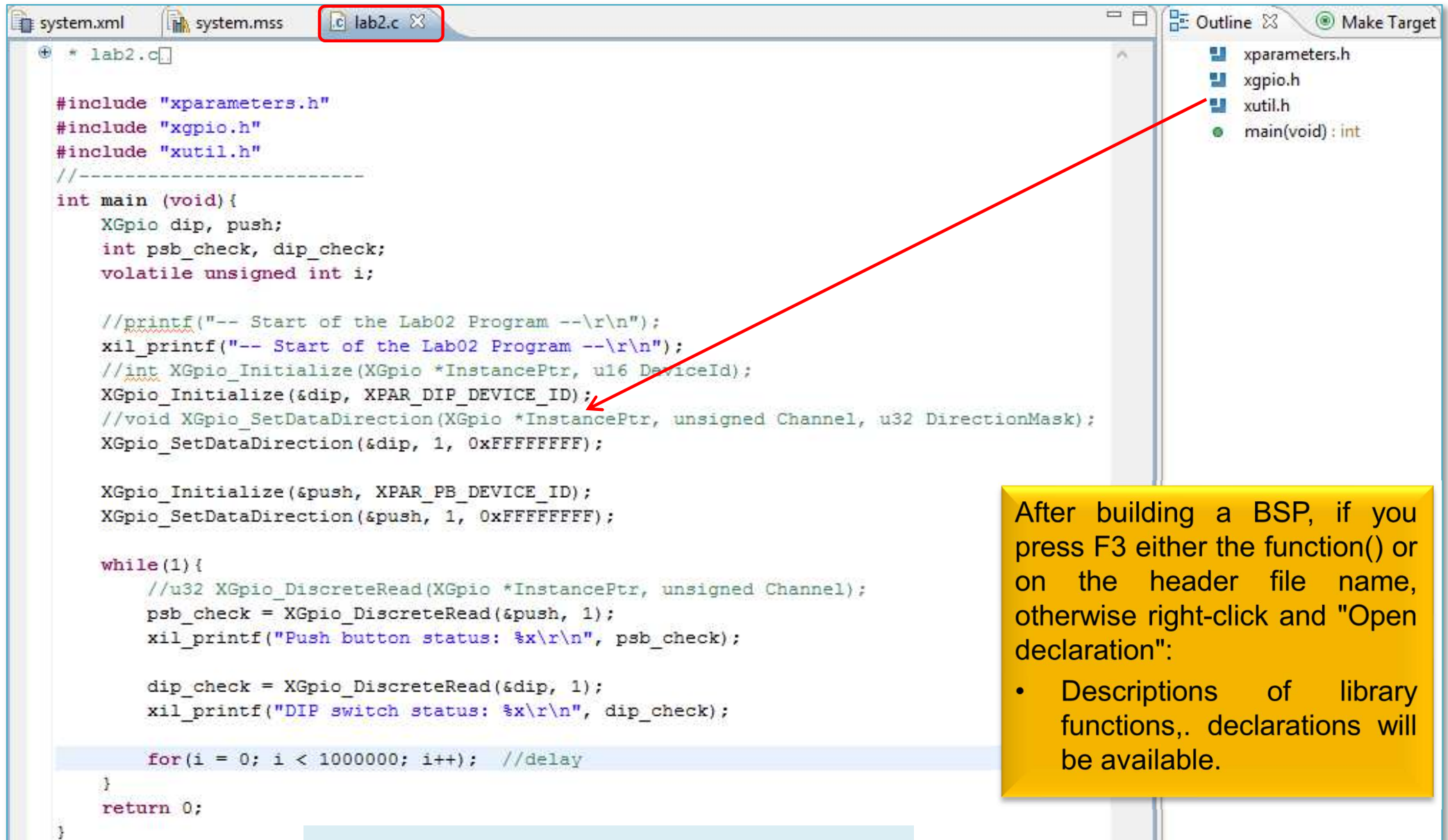
- Add/Create new source (`lab2_a.c`) to the application project



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



# TestApp – source code



```
#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

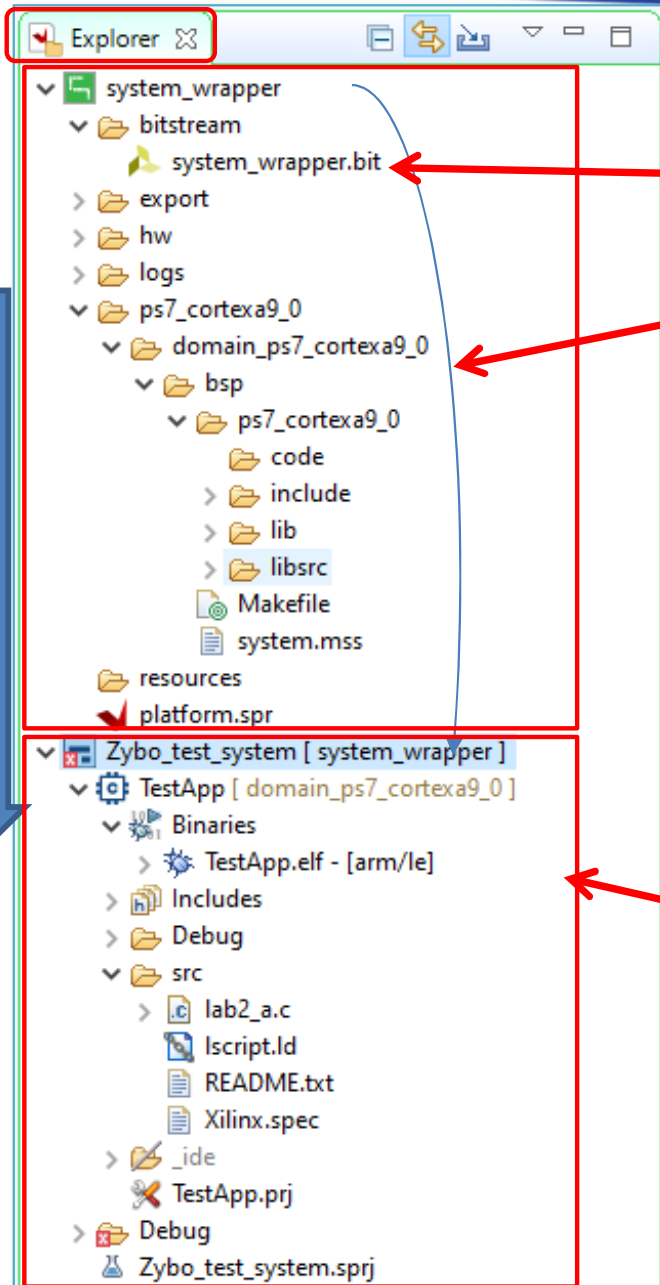
- xparameters.h
- xgpio.h
- xutil.h
- main(void) : int


Make Target

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)

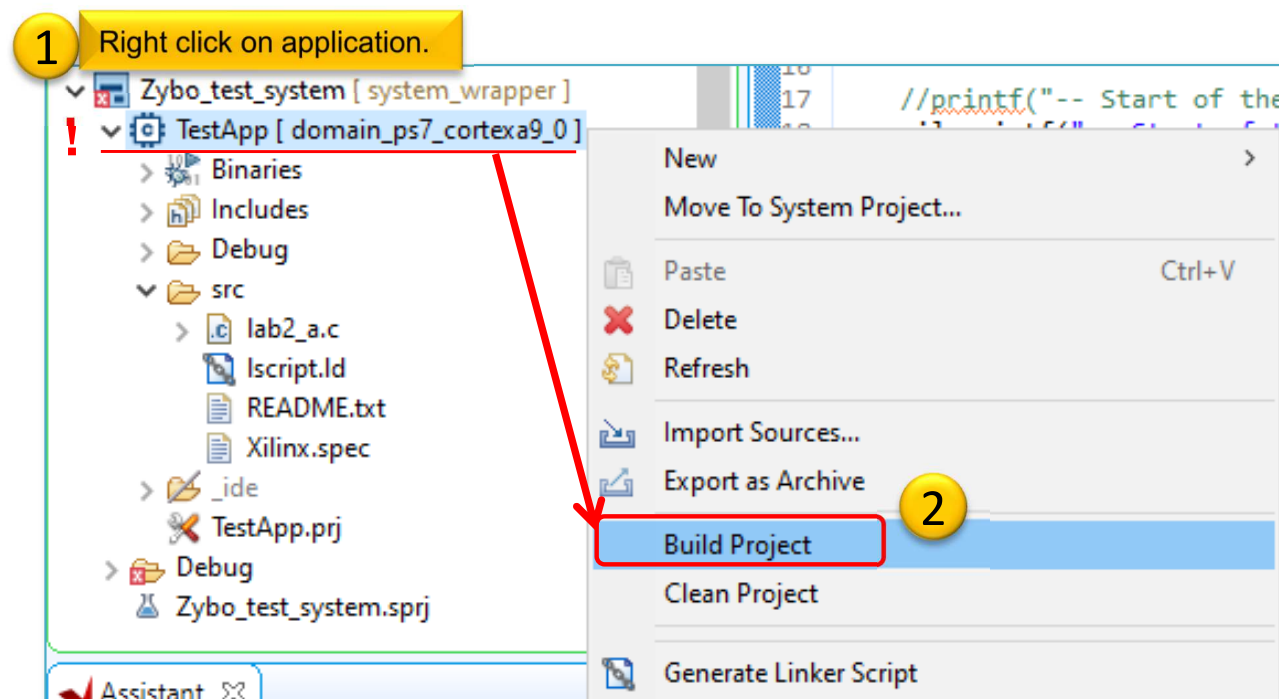
 **Zybo\_test\_system** as **system\_project** contains

 **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

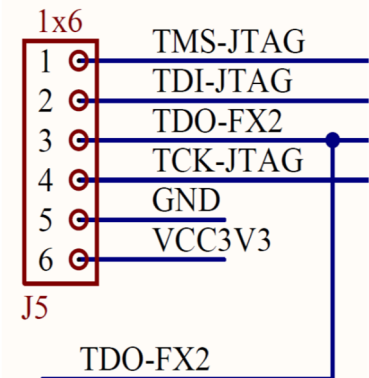
1. **Connect** the USB-serial cable (power+programmer functionality). Please check:

- 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!)

JTAG Header



ZyBo – Xilinx USB programming cable

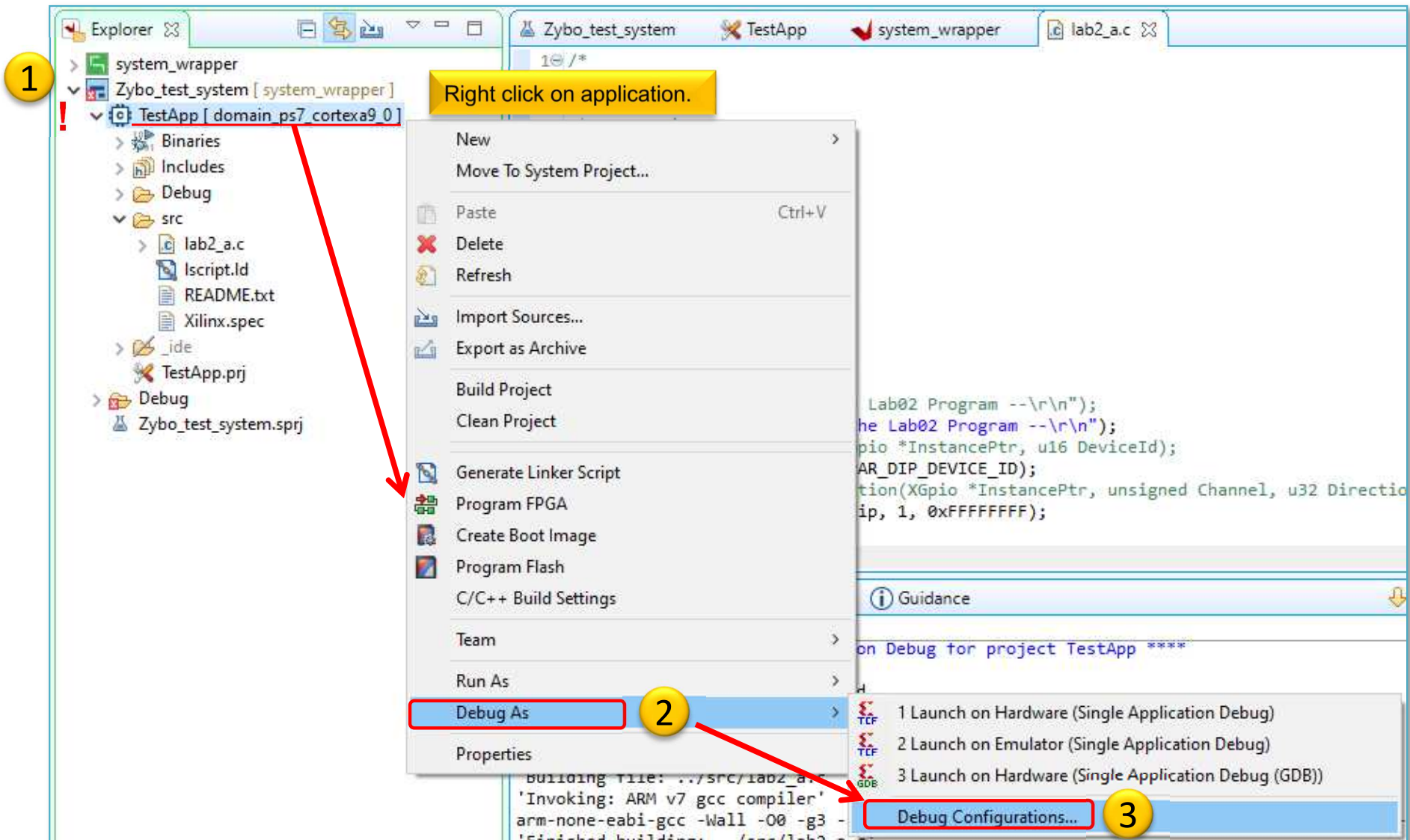
**VCC3V3** – red VREF (6)  
**GND** – black GND (5)  
**TCK-JTAG** – yellow TCK (4)  
**TDO-FX2** – lilac – TDO (3)  
**TDI-JTAG** – white TDI (2)  
**TMS-JTAG** – green TMS (1)

We use the USB-serial connector.

Check the DONE led!

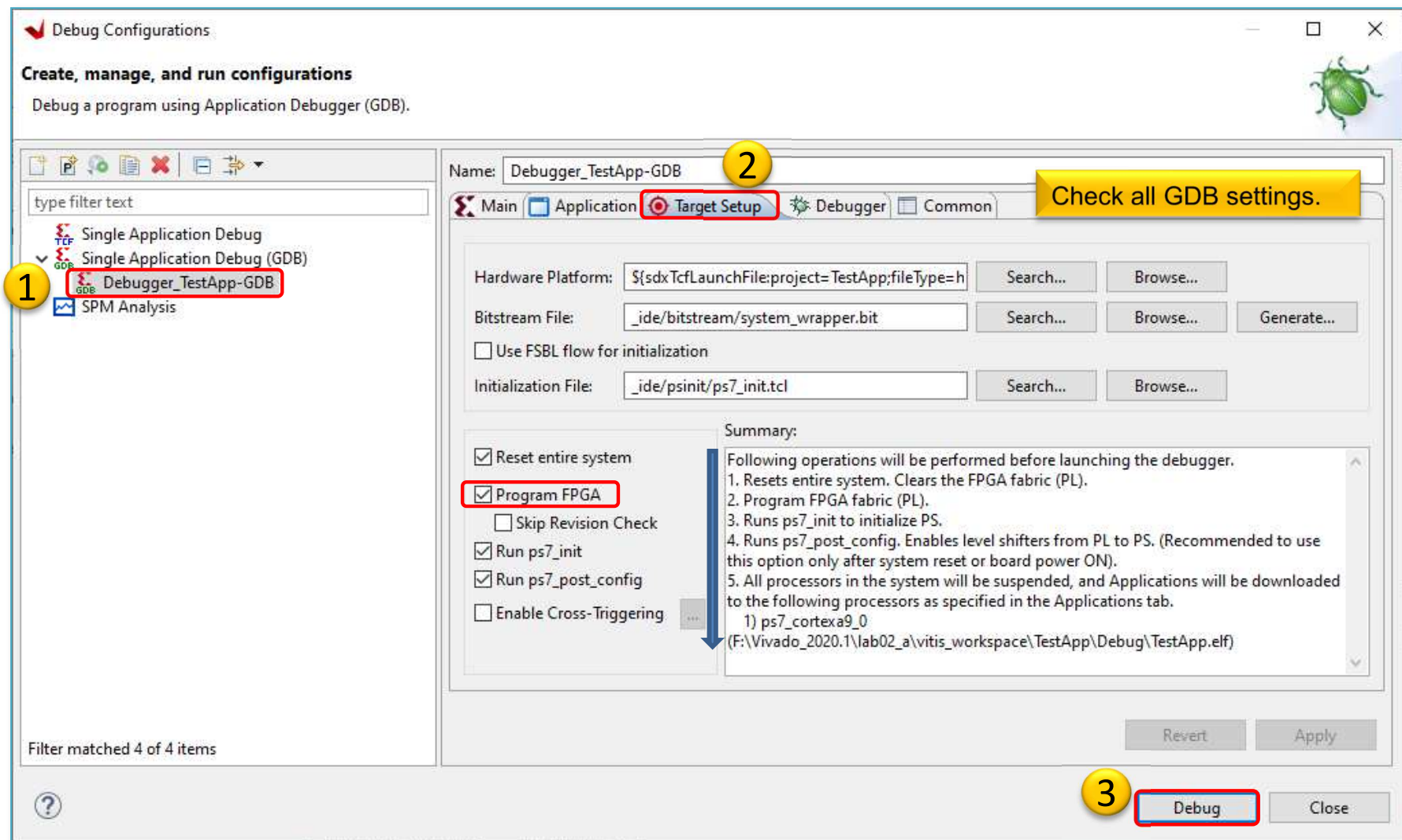
# Creating Debug Configuration

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



# Create a new GDB configuration

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





# Lunching Debugger

1

2

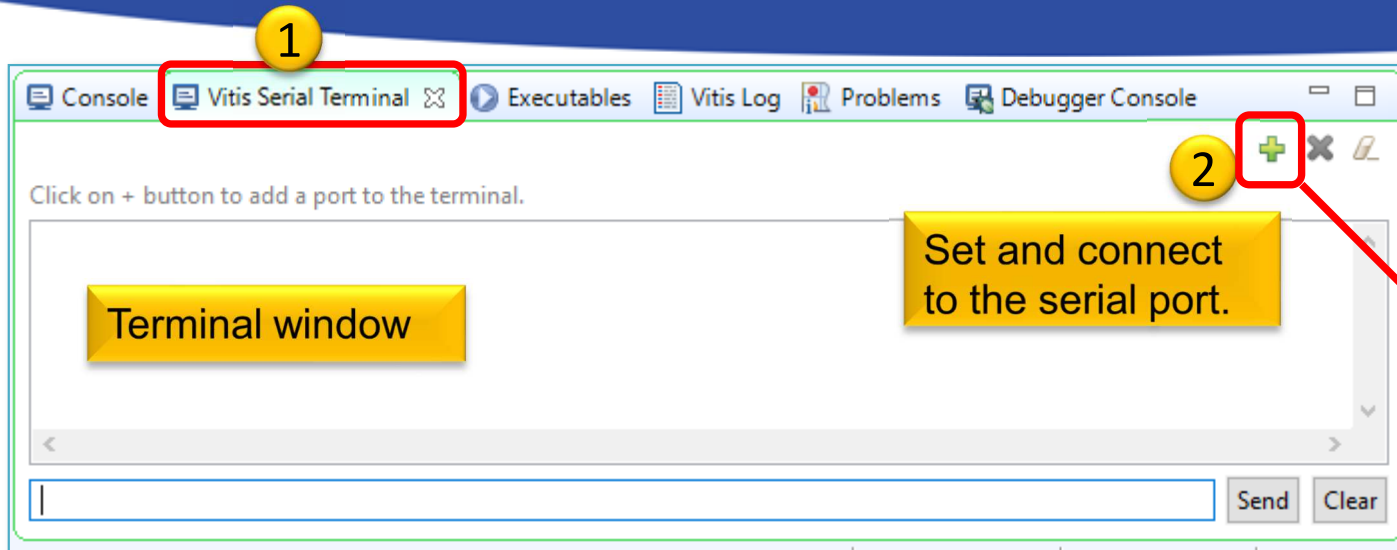
3

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
```

Launching Debugger\_Zy...t-GDB: (98%)

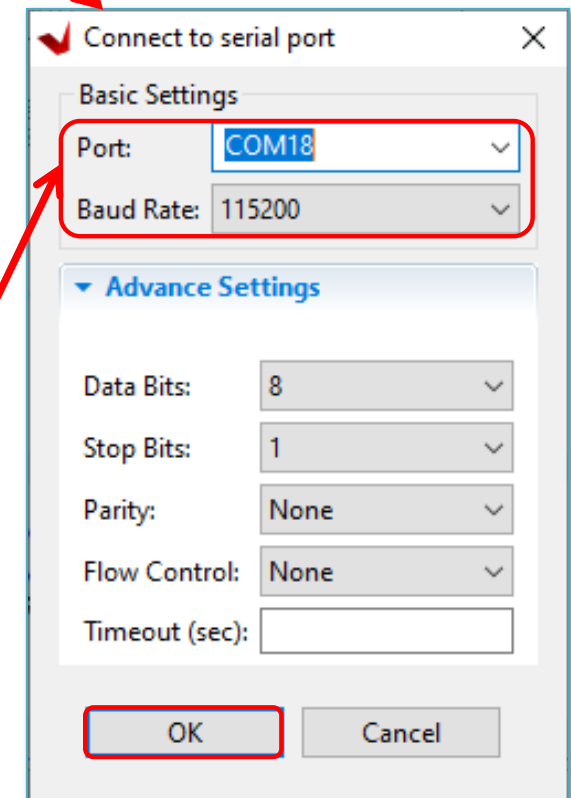
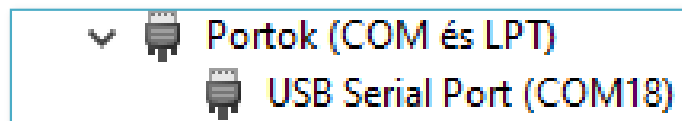
# Set Debug-serial port (VITIS terminal)



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
```

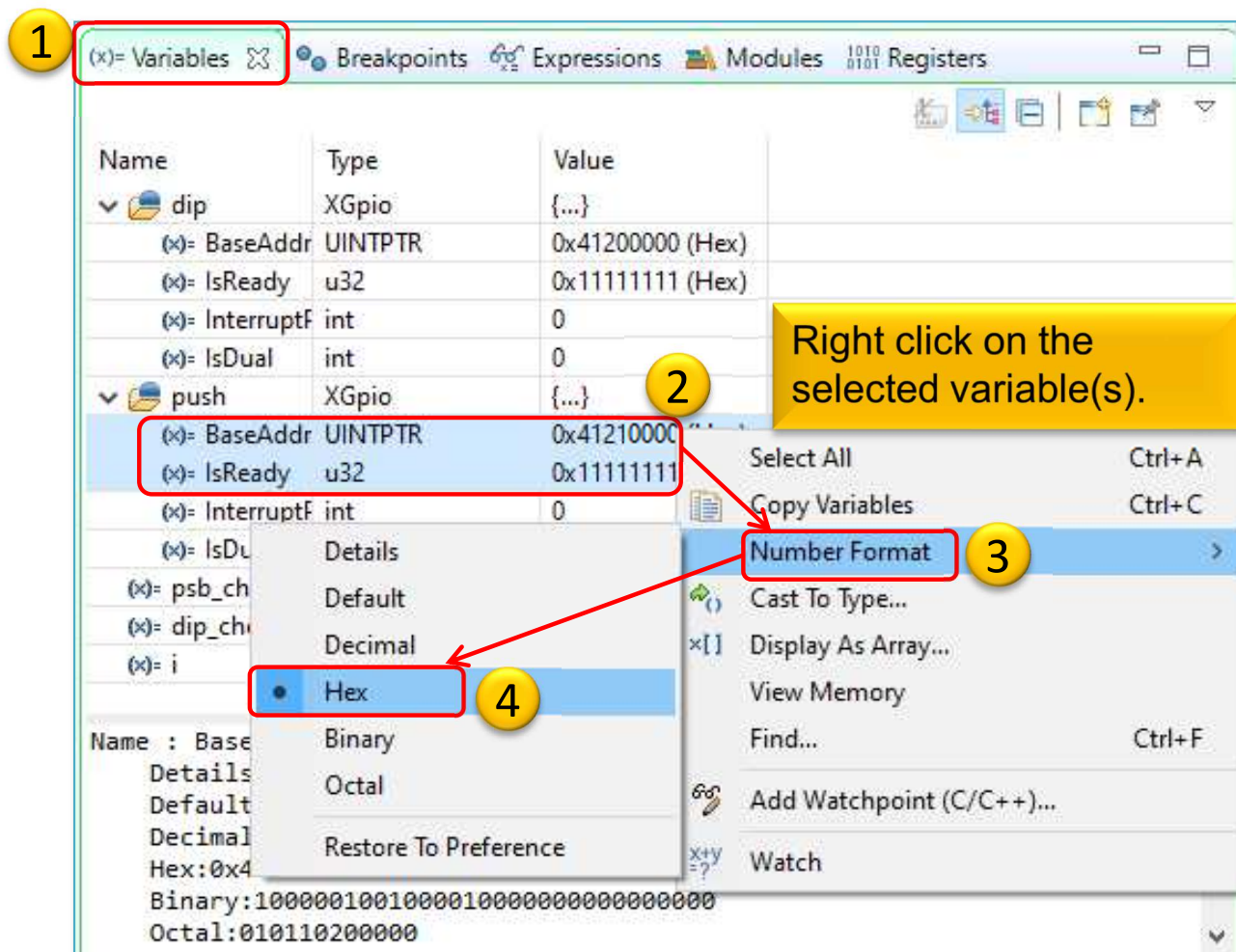
Console Vitis Serial Termi... Executables Vitis Log Problems Debugger Cons...

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!

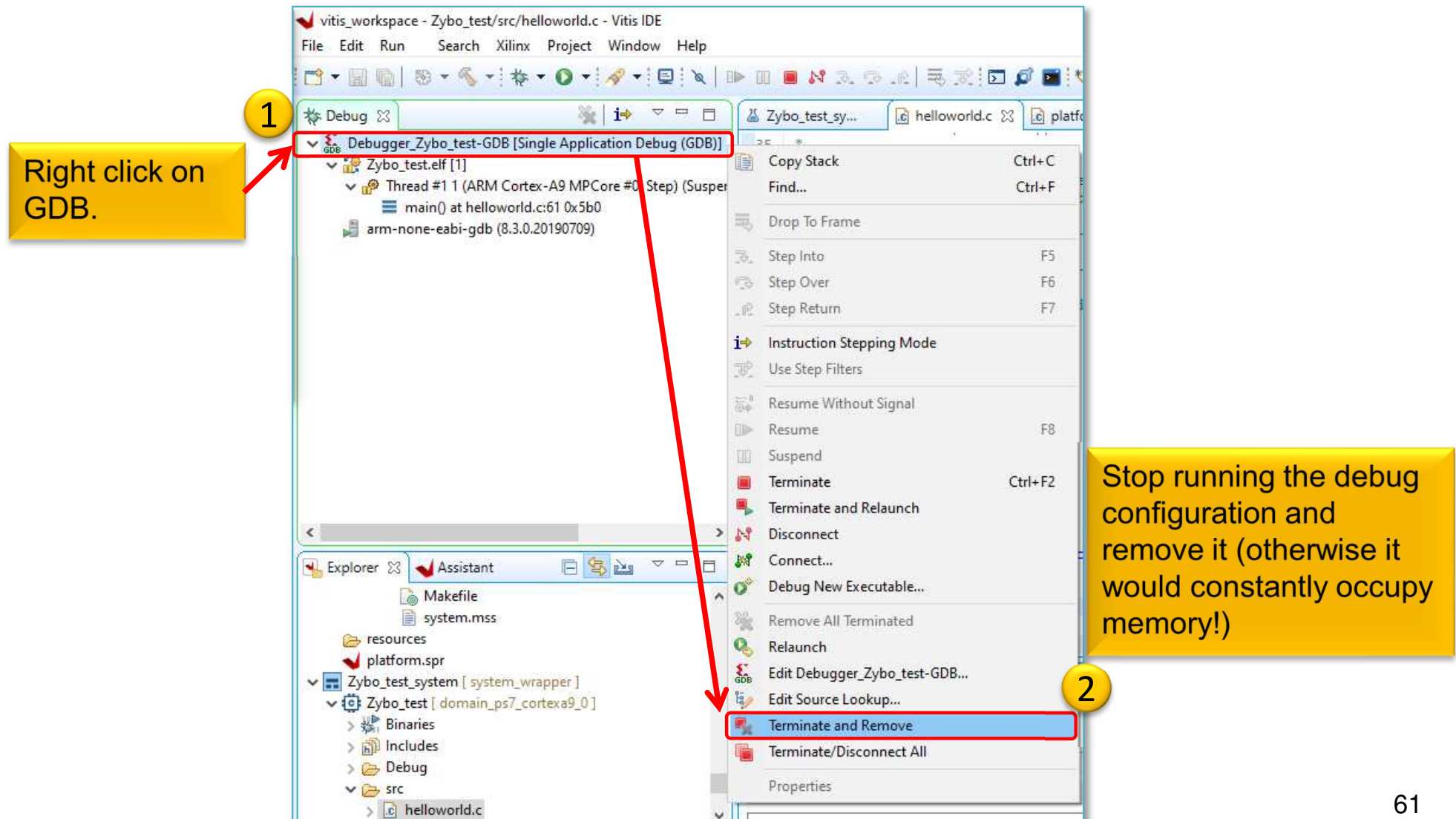
# TestApp – Verification result



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*!





# Compiler settings

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

Properties for TestApp

type filter text

1

2

3

Settings

- ARM v7 gcc assembler
  - General
- ARM v7 gcc compiler
  - Symbols
  - Warnings
  - Optimization
  - Debugging
  - Profiling
  - Directories
  - Miscellaneous
  - Inferred Options
- ARM v7 gcc linker
  - General
  - Libraries
  - Miscellaneous
  - Linker Script
  - Inferred Options
- ARM v7 Print Size

Optimization Level: None (-O0)

Other optimization flags:

Important: Since our program also has a `for()` loop, which is responsible for the delay, and we do not want the compiler to "optimize" it, select the optimization to **(-O0)** = None.

Apply and Close Cancel

# 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

**Generate a linker script**

Control your application's memory map.

**Generate linker script**

Output Settings

Project: Zybo\_test

Output Script: F:\Vivado\_2020.1\lab01\vitis\_workspace\Zybo\_test\src\lscript.ld

Modify project build settings as follows:

Set generated script on all project build configurations

**Hardware Memory Map**

Memory	Base Address	Size
ps7_ddr_0	0x00100000	511 MB
ps7_ram_0	0x00000000	192 KB
ps7_ram_1	0xFFFF0000	~63,5 KB


Fixed Section Assignments

# 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. Lunch 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