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: 27 Nov. 2020.

SZÉCHENYI 2020

MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Strukturális és Beruházási Alapok

BEFEKTETÉS A JÖVŐBE

6. VIVADO – EMBEDDED SYSTEM

Adding peripherals to BSB from IP Catalog #2 (PMOD, GPIO)

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 Peripherals (from IP database) to BSB
7. **Adding Custom (I2C IP and XADC) 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 / USB 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 in source file and 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 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)

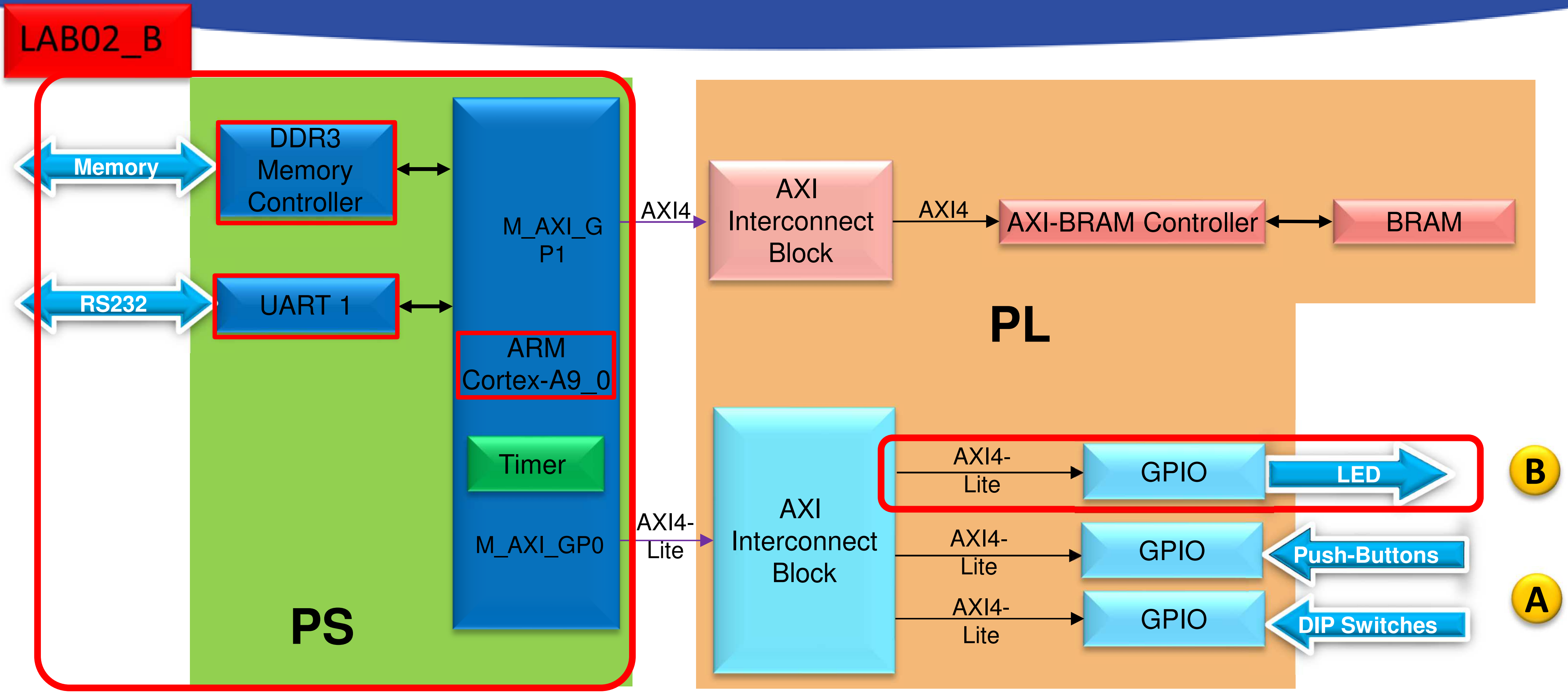
Main steps to solve the task

- Create a new project based on previous laboratory (slide 05.) by using the **Xilinx Vivado (IPI)** embedded system designer,
  - `LAB02_A` project → Save as… → `LAB02_B`
- Select and add GPIO (LED) 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. LED controller (GPIOs)

Test system to be implemented



PS side:

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

PL (FPGA)

- (A) LAB02_A: GPIO inputs
  - **PBSs: Push Button**
  - **DIPs: Switches**
- **(B) LAB02_B: GPIO outputs**
  - **LED controller**

Project – Open / Save as…

- Start Vivado
  - Start menu → Programs → Xilinx Design Tools → Vivado 2020.1
- Open the previous project! (`LAB02_A`)
  - File → Project → Open… / Open Recent…
  - `<projectdir>/LAB02_A/<system_name>.xpr` → **Open**
- File → Project → Save As… → LAB02_B
  - (This will save the former project LAB02_A as LAB02_B)

Task 1.) Adding LED controller

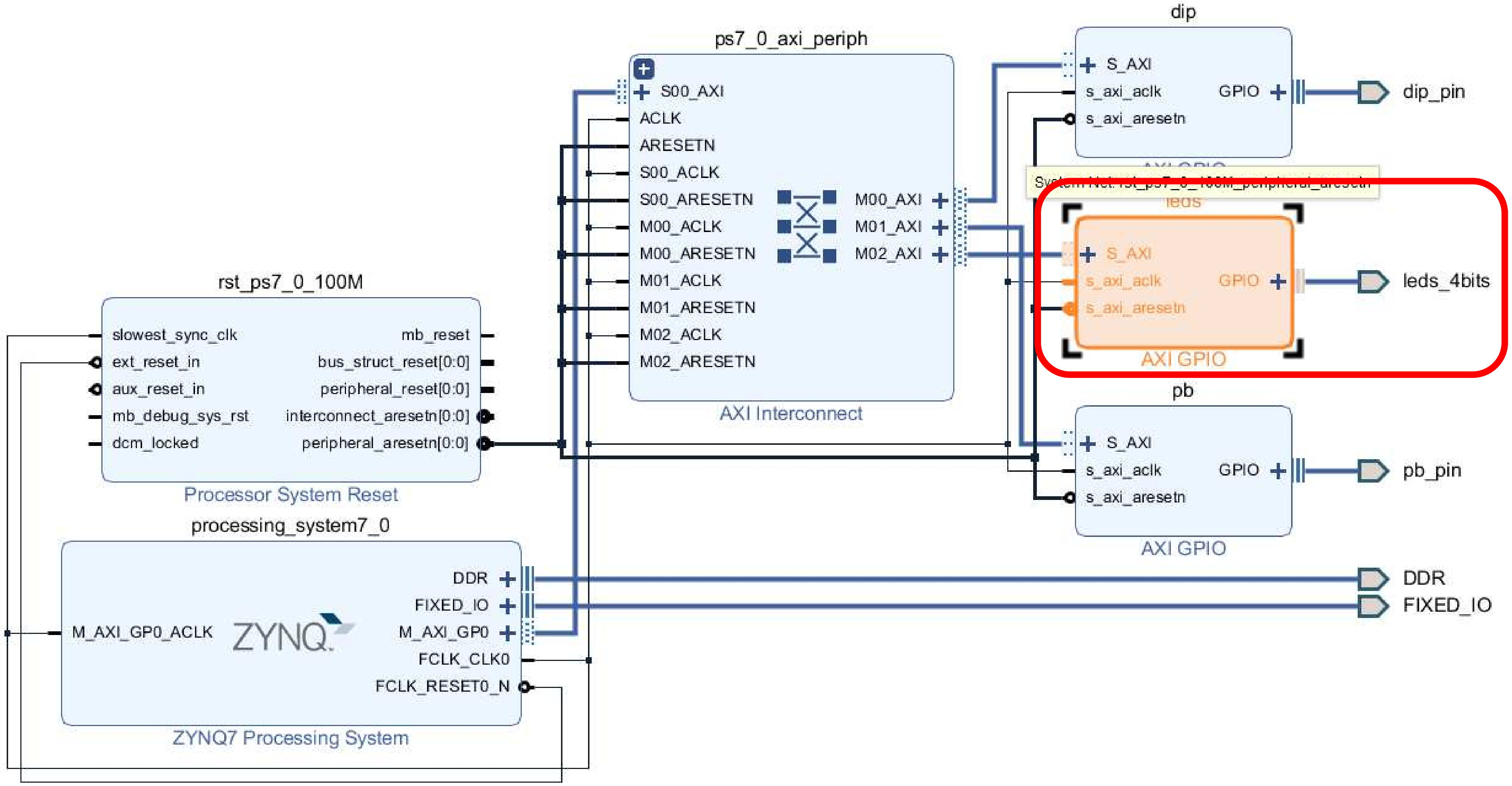
Vivado:

- LED controller: integrate and connect `AXI_GPIO` peripherals selected from the Vivado IP catalog to the base system (4 pieces of LEDs),
- GPIO IP instance should be named as "`leds`".
- GPIO Board interface: select "`leds_4bits`".
- Base Address: `0x4123_0000` (size: 64 K)
- Assign external GPIO LED port to FPGA pins (`led_pin`),
- Examine Block Design and Generate Bitstream

VITIS (~SDK):

- Create a Peripheral test application (PeriphTest) in the VITIS SDK environment (based on the former app `lab02.c` !),
- Test verification of FW-SW plans on the ZyBo platform.

Vivado – Completed design



Task 2.) LED counter(s)

- Modify the Peripheral Test SW application (`lab02_a.c`) in order to flash the LEDs by increasing the value of a 4-bit counter.
  - Apply the PeripheralTest template (see GpioOutputExample () function)
- Help:
  - Use the built-in data types (e.g. `u8`)
  - Since `sys_clk = 100 MHz`, delay the LEDs up / down (by using a `for()` cycle) so that the flash time can be percebtible: ~1 sec.
  - Using macros from the `xparameters.h` file if a possible build error(s) occurs
    - set `LED_DELAY, GPIO_BITWIDTH`!
- BER_lab2b_led8bit_count.zip

Task 3.) LED strings

- Modify the application of the Peripheral Test SW in the former Task 2.) so that the value of the LEDs is always shifted by one position to the left (as an ascending binary weight counter)
- **Help:** BER_lab2b_led8bit_shift.zip

Task 4.) „Knight Rider” LED strings

- Modify the application of the Peripheral Test SW in the former Task e 3.) so that the value of the LEDs is always shifted by one position to the left and then when it reaches the end it moves backwards to the right (using an increasing and then decreasing binary weight counter).
- **Help:** BER_lab2b_led8bit_knightrider.zip

XILINX VIVADO DESIGN SUITE

LAB2_C. PMOD_TMP temperature measurement (AXI_I2C controller / PS_I2C controller).

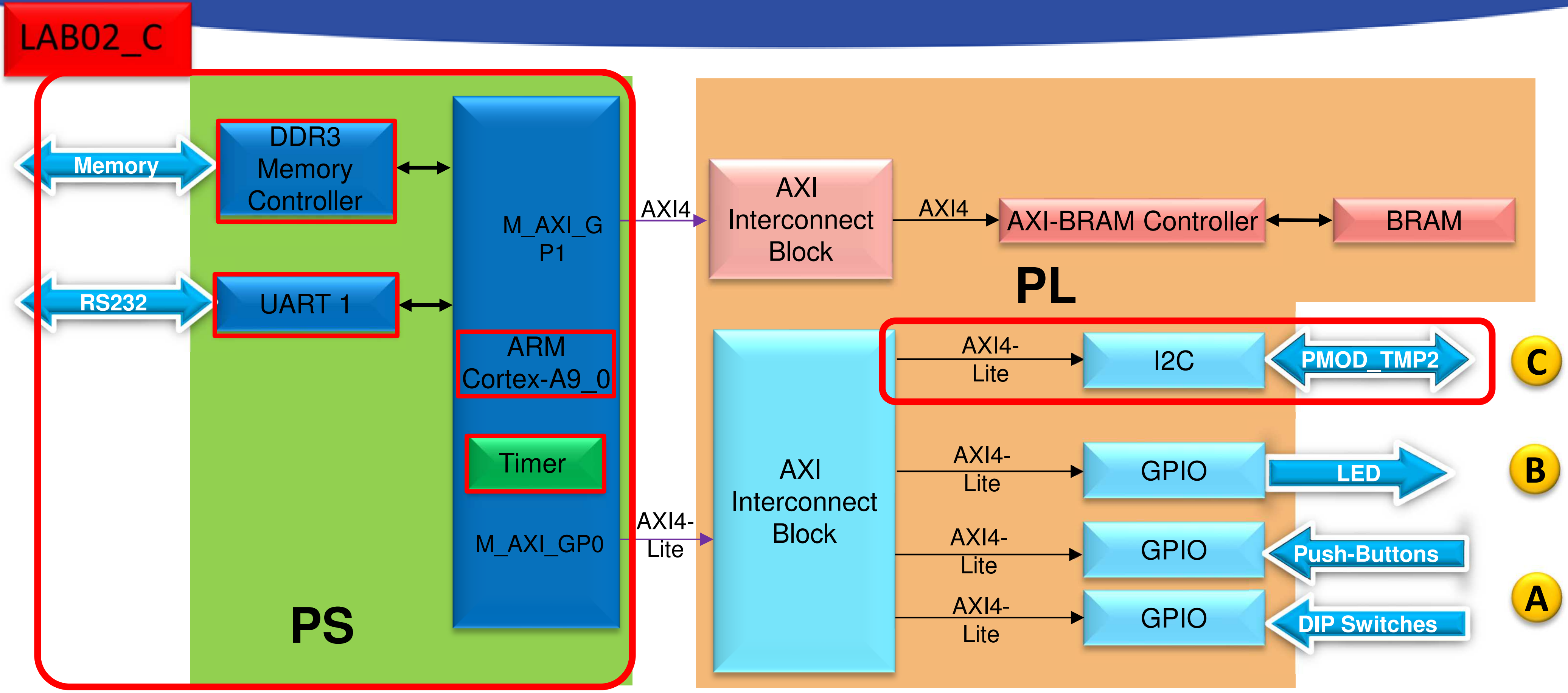
SZÉCHENYI 2020

Magyarország Kormánya

Európai Unió
Európai Strukturális és Beruházási Alapok

BEFEKTETÉS A JÖVŐBE

Test system to be implemented



PS side:

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

PL (FPGA)

- (A) LAB02_A: GPIO inputs
  - PBSs: Push Button (nyomógomb kezelő)
  - DIPs: Switches (kapcsoló kezelő)
- (B) LAB02_B: GPIO outputs
  - LED controller
- **(C) LAB03_C: I2C component**

Main steps to solve the task

- Create a new project based on previous laboratory (slide 05.) by using the **Xilinx Vivado (IPI)** embedded system designer,
  - `LAB02_A` project → Save as… → `LAB02_C`
- Select and add AXI_I2C (or PS_I2C) peripheral controller 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.

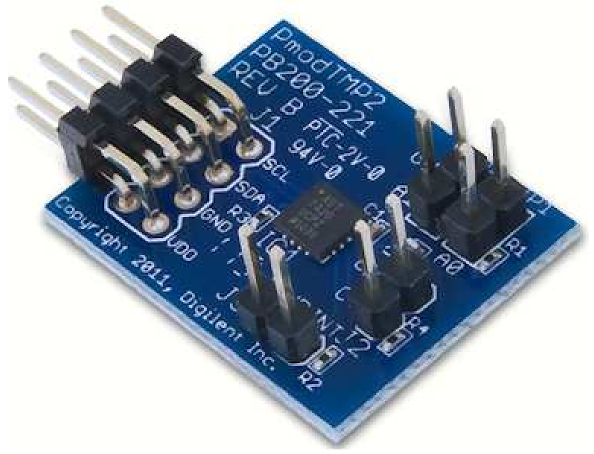
References

- Digilent PMOD TMP2 temperature sensor modul (I^2C):
  - 🕮 https://store.digilentinc.com/pmod-tmp2-temperature-sensor/
  - 🕮 https://reference.digilentinc.com/reference/pmod/pmodtmp2/reference-manual
- Analog Devices ADT7420 sensor IC:
  - 🕮 http://www.analog.com/media/en/technical-documentation/data-sheets/ADT7420.pdf
- Analog Devices – Digilent Wiki page:
  - 🕮 http://wiki.analog.com/resources/alliances/digilent

  Reference design for FPGA (inc. SW drivers)
  - 🕮 http://wiki.analog.com/resources/fpga/xilinx/pmod/adt7420
- I^2C standard:
  - 🕮 Dr. Fodor Attila, Dr. Vörösházi Zsolt: Beágyazott rendszerek, TÁMOP 4.1.2 (PE MIK, Villamosmérnöki és Információs Rendszerek Tanszék) 2011. – in hungarian - http://tananyagfejlesztes.mik.uni-pannon.hu/
  - 🕮 I2C Wikipedia: https://en.wikipedia.org/wiki/I%C2%B2C

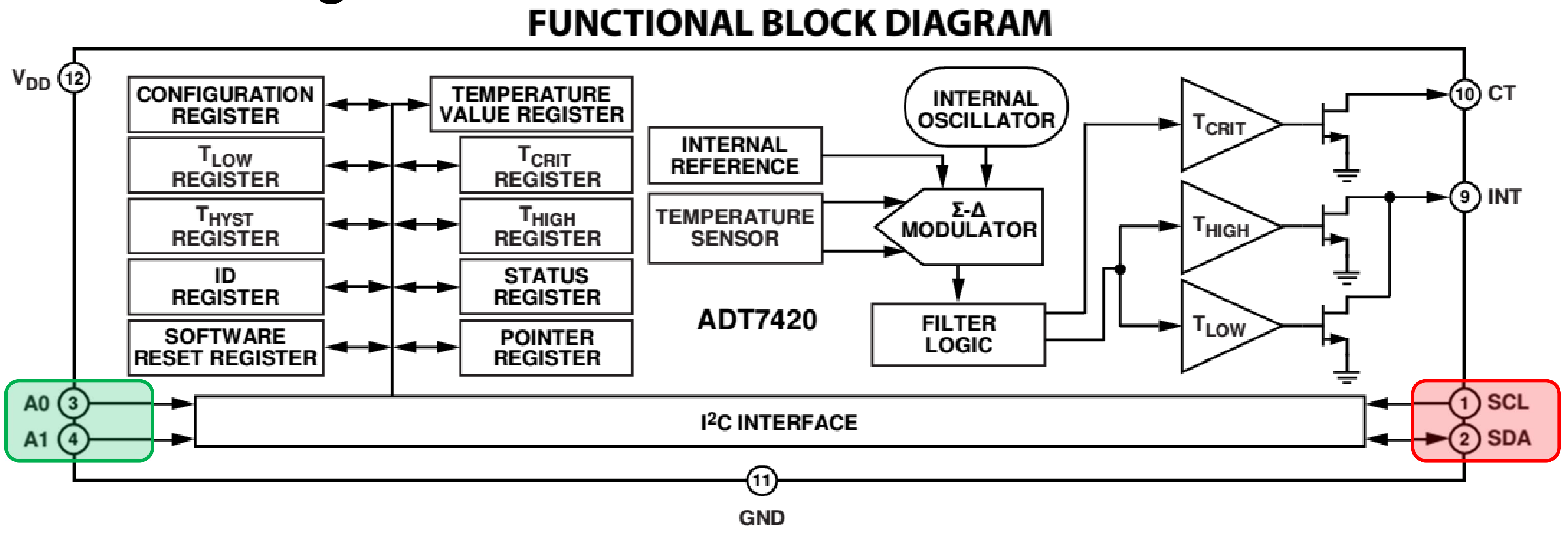
Digilent PMOD_TMP2

- I^2C alapú based temperature sensor and temperature controller peripheral module
  - T_A= −40°C … +150°C,
  - max. scalable to 16-bit resolution,
  - An average accuracy better than 0.25 °C,
    - 13 = 9+4 bit mode: $1/2^4$ = 0.0625 °C,
    - 16 = 9+7 bit mode: $1/2^7$ = 0.0078 °C,
  - I^2C interface, 4 selectable (jumper) with I^2C address (A_1-A_0)
    - „Daisy Chain” option (7/10 bit addressing)
  - Continuous conversion in every 240ms,
  - Programmable treshold (max/min - CT), external pins as threshold (INT),
  - **3.3**V or 5V interface support,
  - No calibration required!



ADT 7420

- Block diagram:



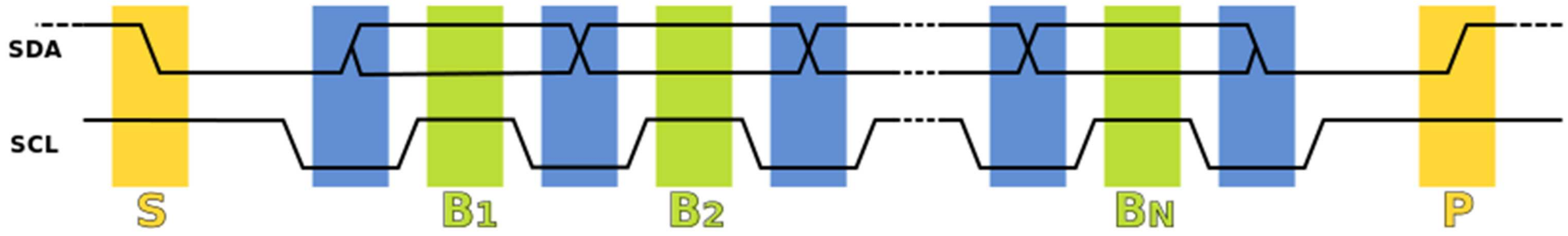
- PMOD_TMP2 signals / I^{2}C addresses:

Connector J1 – I2C Communications		
Pin	**Signal**	**Description**
1, 2	SCL	I2C Clock
3, 4	SDA	I2C Data
5, 6	GND	Power Supply Ground
7, 8	VCC	Power Supply (3.3V/5V)

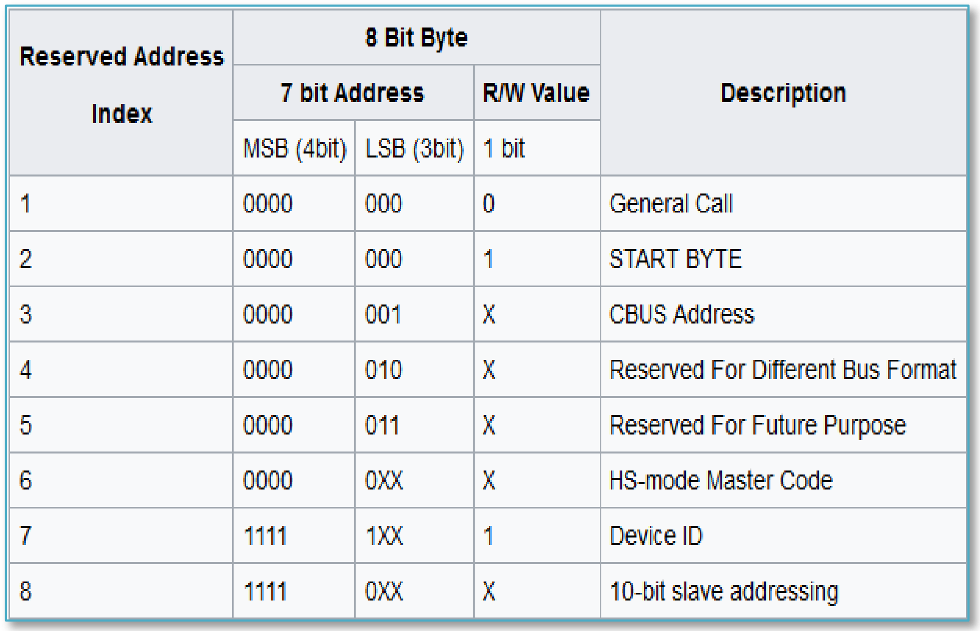
Addresses		
JP2	**JP1**	**Address**
Open	Open	0x4B (0b1001011)
Open	Shorted	0x4A (0b1001010)
Shorted	Open	0x49 (0b1001001)
Shorted	Shorted	0x48 (0b1001000)

Philips I^{2}C standard (1982)

- Web: https://en.wikipedia.org/wiki/I%C2%B2C



1. Data Transfer is initiated with a *START bit* (S) signaled by **SDA** being pulled **low** while SCL stays high (pull-down).
2. SDA sets the 1st data bit level while keeping **SCL low** (during **blue** bar time) .
3.The data is sampled (received) when **SCL rises** (**green**) for the first bit (B1).
4. This process repeats, SDA transitioning until **SCL is low** again, and the data being read while **SCL is high** (B2, Bn).
5. A *STOP bit* (P) is signaled when SDA is pulled high while **SCL** is **high**.

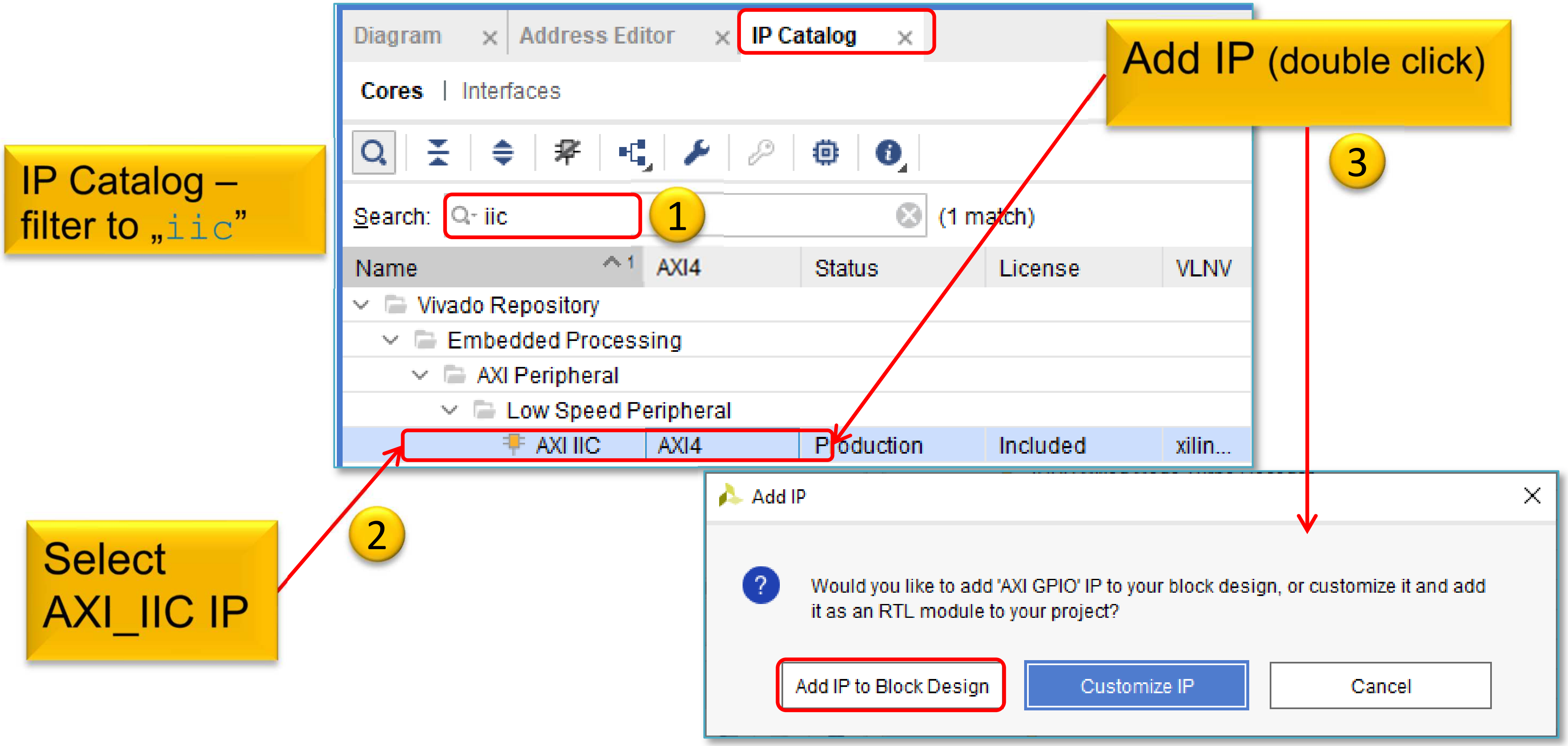


Reserved Address Index	8 Bit Byte			Description
	7 bit Address		R/W Value	
	MSB (4bit)	LSB (3bit)	1 bit	
1	0000	000	0	General Call
2	0000	000	1	START BYTE
3	0000	001	X	CBUS Address
4	0000	010	X	Reserved For Different Bus Format
5	0000	011	X	Reserved For Future Purpose
6	0000	0XX	X	HS-mode Master Code
7	1111	1XX	1	Device ID
8	1111	0XX	X	10-bit slave addressing

„Master write(0) to / read(1) from the Slave"

Adding I²C controller to the Base System I.

- Add I²C controller – two possible ways by
  - a.) adding **PL-side AXI_I2C IP core,** OR
  - b.) enabling PS-side (PS_I2C0/1) IIC controller in Zynq PS7
- Add an `AXI_IIC` controller to the Block Diagram (IP Catalog)

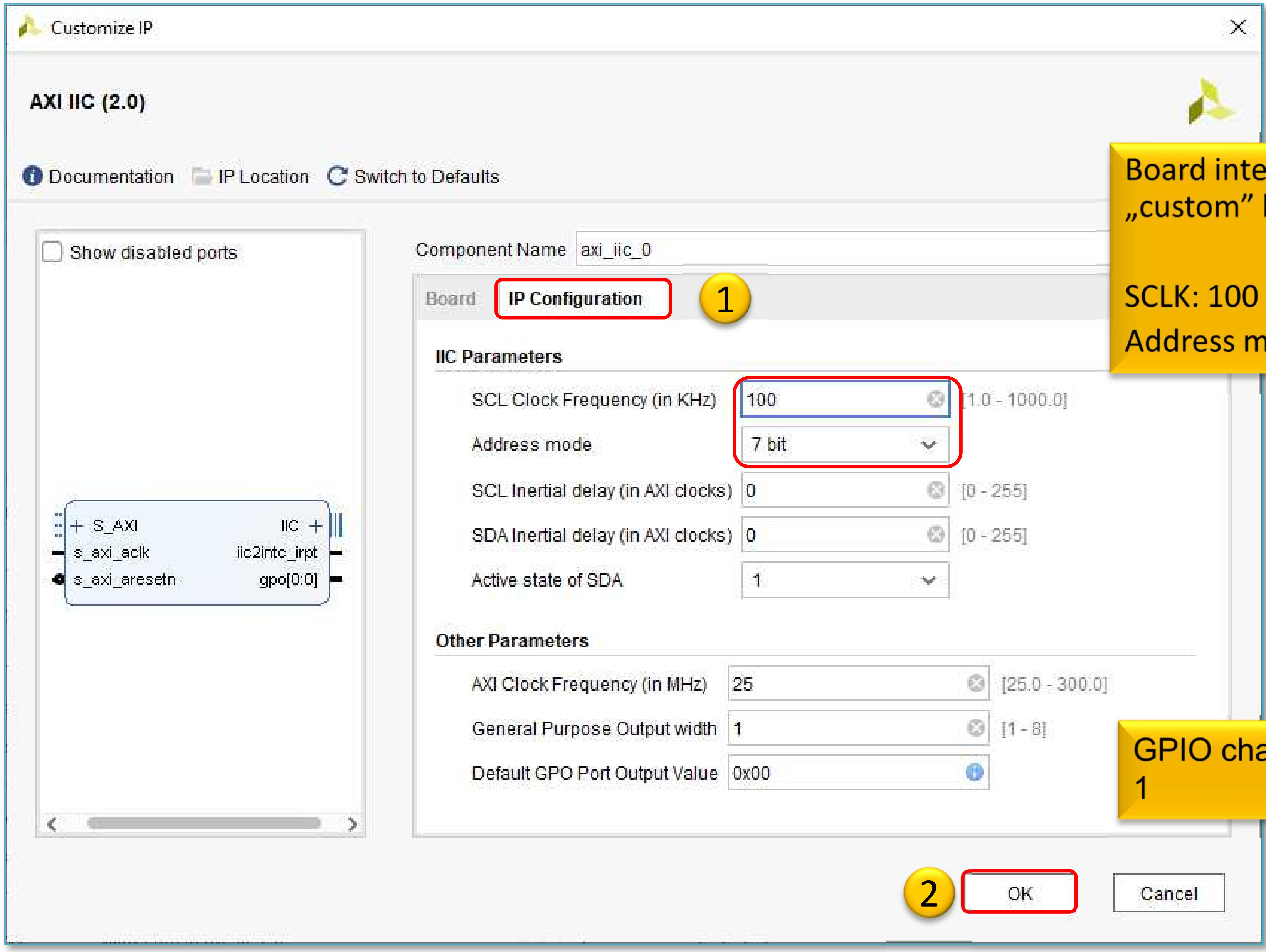


Name	AXI4	Status	License	VLNV
Vivado Repository				
Embedded Processing				
AXI Peripheral				
Low Speed Peripheral				
AXI IIC	AXI4	Production	Included	xilin...

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?

Adding AXI IIC peripheral interface to the Base System II.



Customize IP

AXI IIC (2.0)

Documentation   IP Location   Switch to Defaults

Show disabled ports

Component Name: axi_iic_0

Board | IP Configuration

1

IIC Parameters

SCL Clock Frequency (in KHz)	100	[1.0 - 1000.0]
Address mode	7 bit	
SCL Inertial delay (in AXI clocks)	0	[0 - 255]
SDA Inertial delay (in AXI clocks)	0	[0 - 255]
Active state of SDA	1	

Other Parameters

AXI Clock Frequency (in MHz)	25	[25.0 - 300.0]
General Purpose Output width	1	[1 - 8]
Default GPO Port Output Value	0x00	

2 OK   Cancel

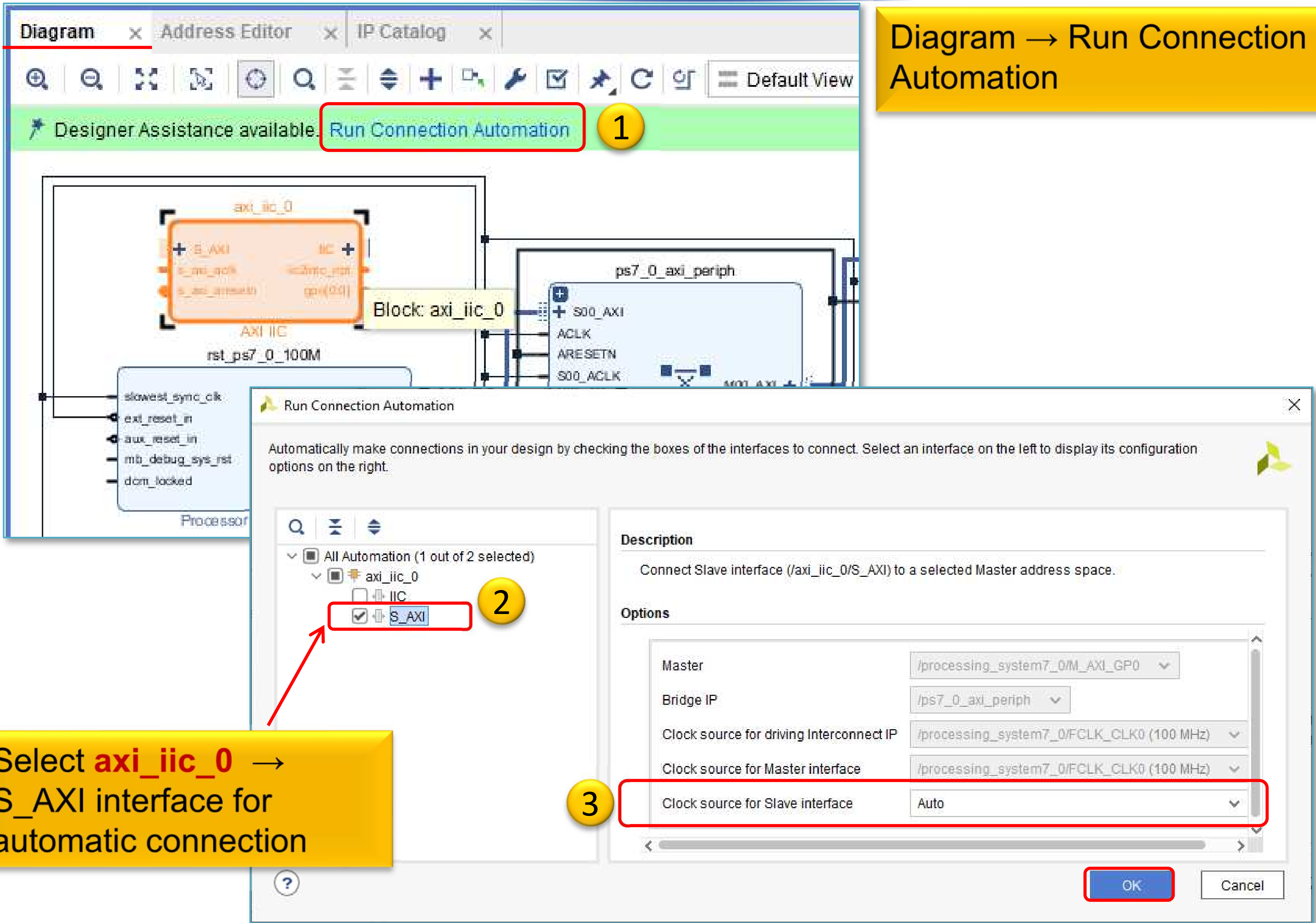
Board interface: „custom" by default.

SCLK: 100 KHz

Address mode: 7 bit

GPIO channel width : 1

Run Autorouter (AXI_IIC)



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 (1 out of 2 selected)
- axi_iic_0
  - [ ] IIC
  - [✓] S_AXI

Description

Connect Slave interface (/axi_iic_0/S_AXI) to a selected Master address space.

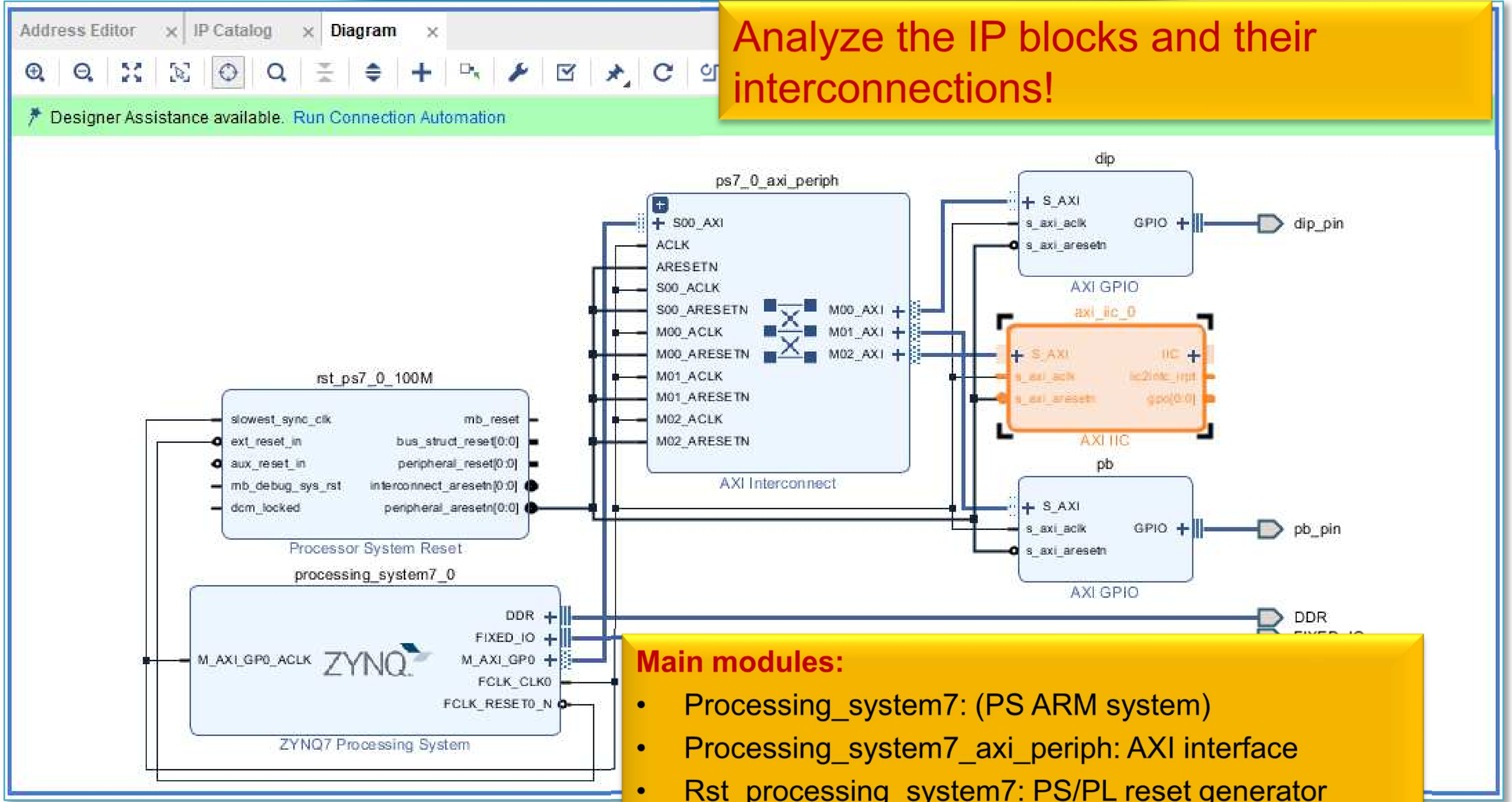
Options

Master	/processing_system7_0/M_AXI_GP0
Bridge IP	/ps7_0_axi_periph
Clock source for driving Interconnect IP	/processing_system7_0/FCLK_CLK0 (100 MHz)
Clock source for Master interface	/processing_system7_0/FCLK_CLK0 (100 MHz)
Clock source for Slave interface	Auto

Diagram → Run Connection Automation

Select **axi_iic_0** → S_AXI interface for automatic connection

Block Design – Analyze



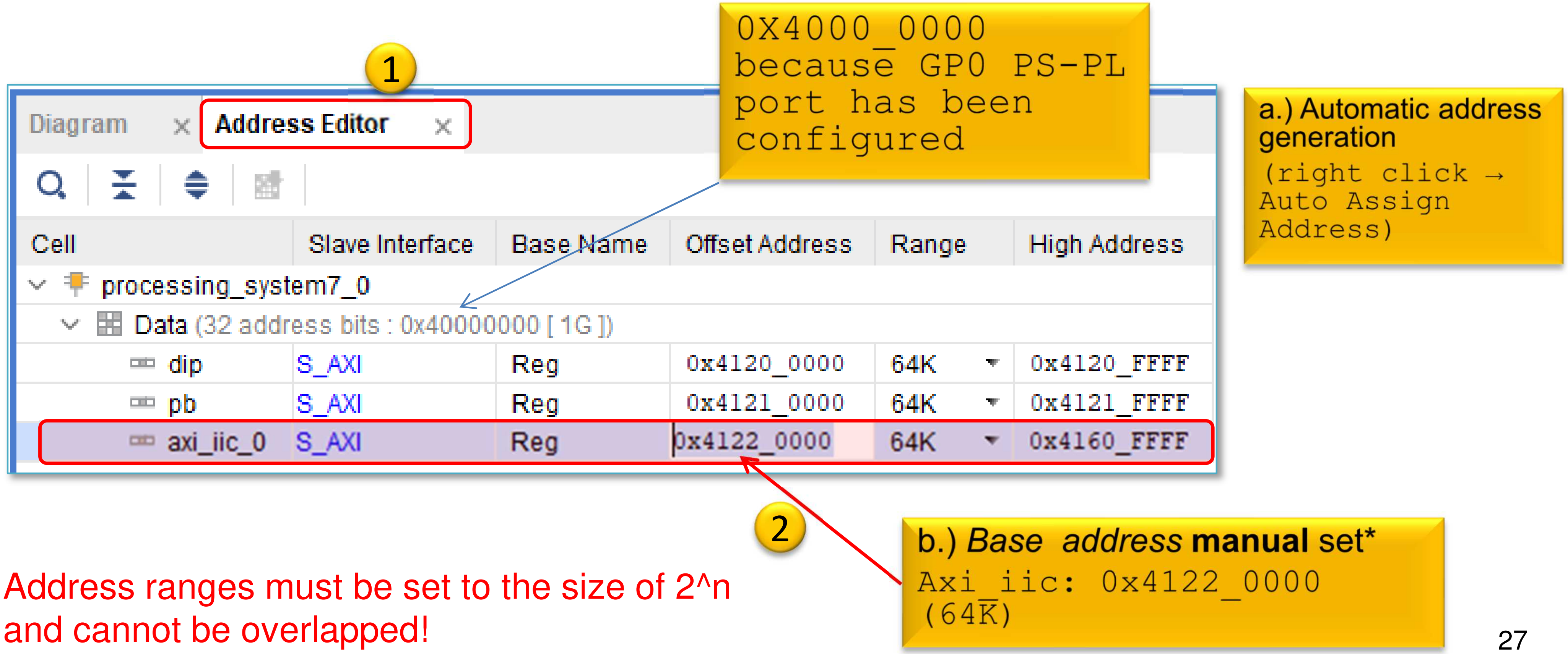
Analyze the IP blocks and their interconnections!

Main modules:

- Processing_system7: (PS ARM system)
- Processing_system7_axi_periph: AXI interface
- Rst_processing_system7: PS/PL reset generator
- dip: AXI GPIO for DIP switches (PL side)
- Pb: AXI GPIO for PB pushbuttons (PL side)
- **Axi_iic_0: AXI_IIC peripheral controller**

AXI IIC – Set memory address

- Block Design → select „Address Editor” view
- Assign „*UnMapped*” IP peripherals to the ARM’s address range:
  - a.) automatic - **vs.** b.) manual address generation


1
Diagram
Address Editor
0X4000_0000 because GP0 PS-PL port has been configured
a.) Automatic address generation
(right click → Auto Assign Address)
2
b.) *Base address* **manual** set*
Axi_iic: 0x4122_0000 (64K)

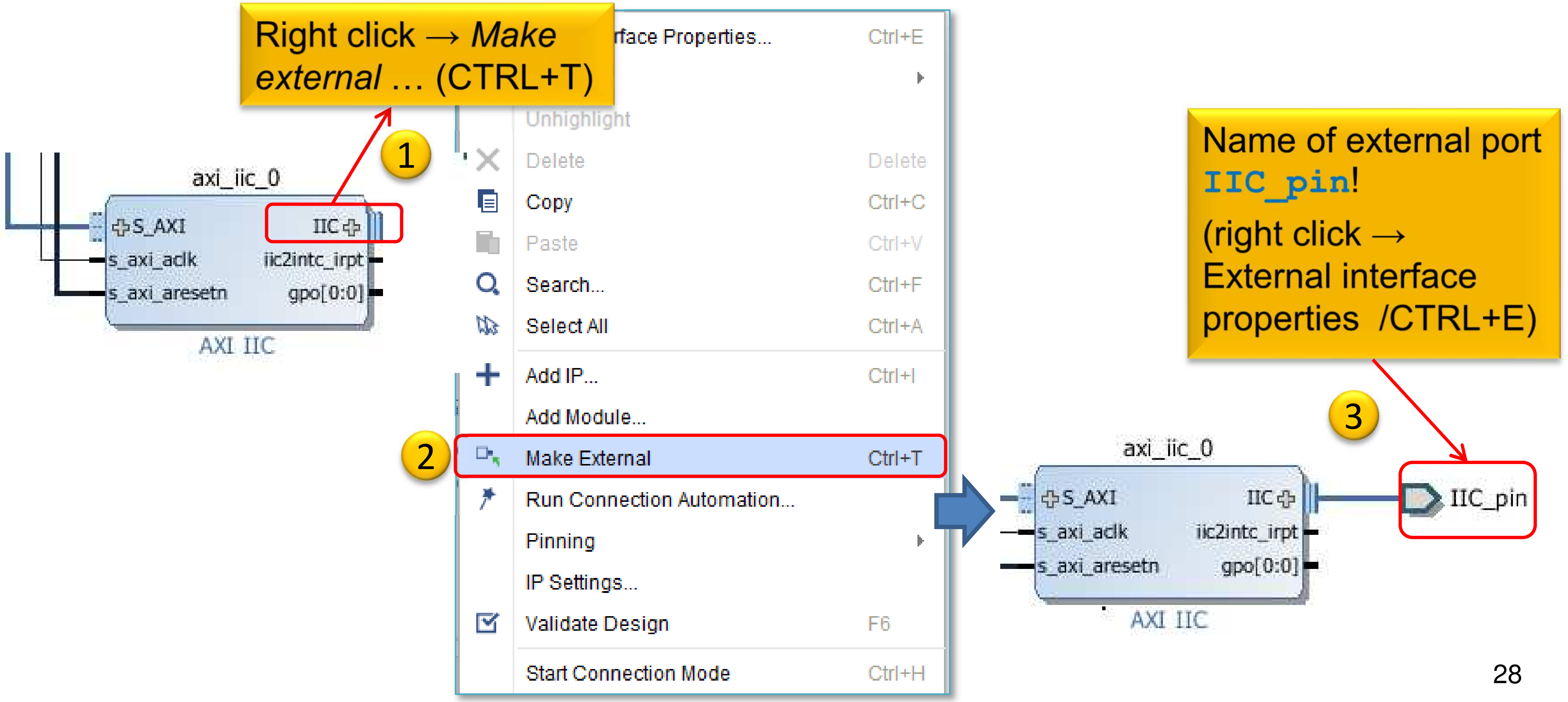
Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
dip	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
pb	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF
axi_iic_0	S_AXI	Reg	0x4122_0000	64K	0x4160_FFFF

Address ranges must be set to the size of 2^n and cannot be overlapped!

AXI IIC – Assign ports to external pins

The **AXI_IIC** instance must be connected to the FPGA (PL-side) pins on the ZyBo card (`PMOD` **JE** `3-4` signals):

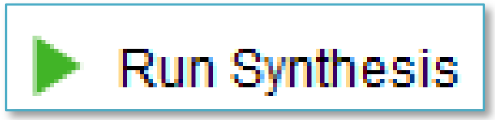
1. The AXI_IIC ports must also be connected to the external physical FPGA pins,
2. If necessary, we also define the names of the external ports (ending in _pin), then
3. In the `<system> .XDC` file you will need to specify constraints (proper FPGA pins).



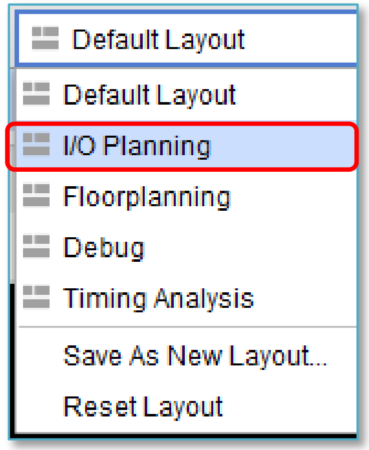
Block Design – Layout synthesis

- Block Design must be updated:
  - Regenerate Layout
  - Validate Design (DRC)

  - Flow Navigator → Run Synthesis
    - Then **- Open Synthesized Design** , OK

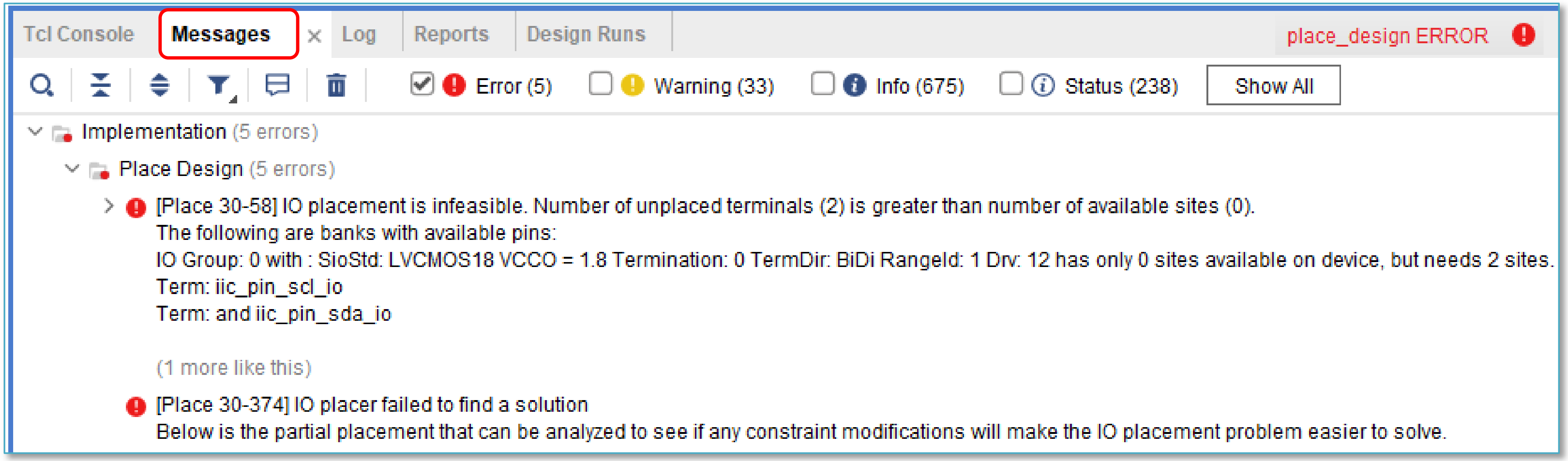


- Finally, two FPGA I/O pins must also be assigned to the external ports (`IIC_pin`)!
  - Layout menu → „IO Planning" layout view



IO Planning - Incorrect pin assignment

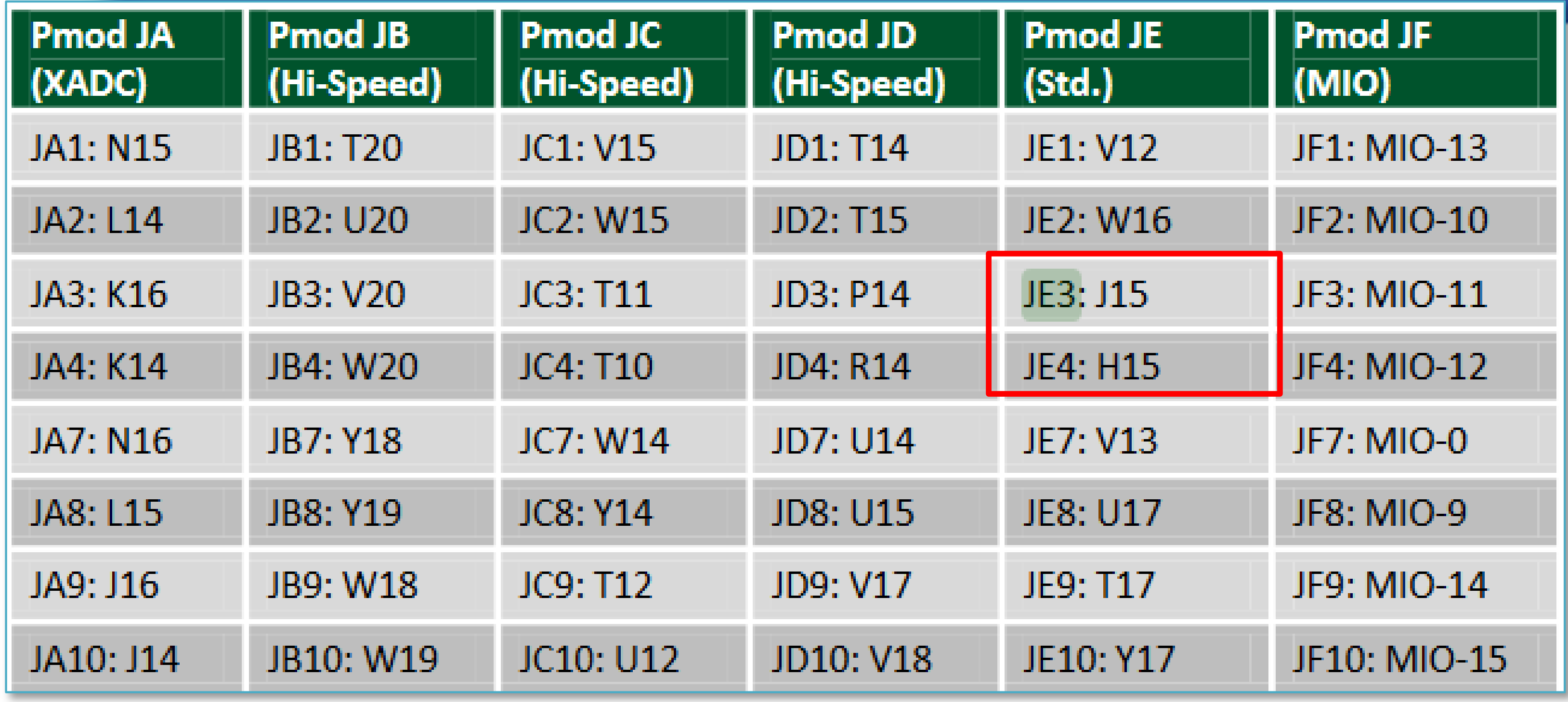
- Note: If you immediately start the „Run Implementation” without constraining pins, you will receive the following error message:



```
Implementation (5 errors)
  Place Design (5 errors)
    [Place 30-58] IO placement is infeasible. Number of unplaced terminals (2) is greater than number of available sites (0).
    The following are banks with available pins:
    IO Group: 0 with : SioStd: LVCMOS18 VCCO = 1.8 Termination: 0 TermDir: BiDi RangeId: 1 Drv: 12 has only 0 sites available on device, but needs 2 sites.
    Term: iic_pin_scl_io
    Term: and iic_pin_sda_io

    (1 more like this)
    [Place 30-374] IO placer failed to find a solution
    Below is the partial placement that can be analyzed to see if any constraint modifications will make the IO placement problem easier to solve.
```

- Cause: Vivado tryed to assign the I2C signals
  - But assigned bad pin location and IO Standard (LVCMOS 1.8V)

ZyBo - PMOD connectors

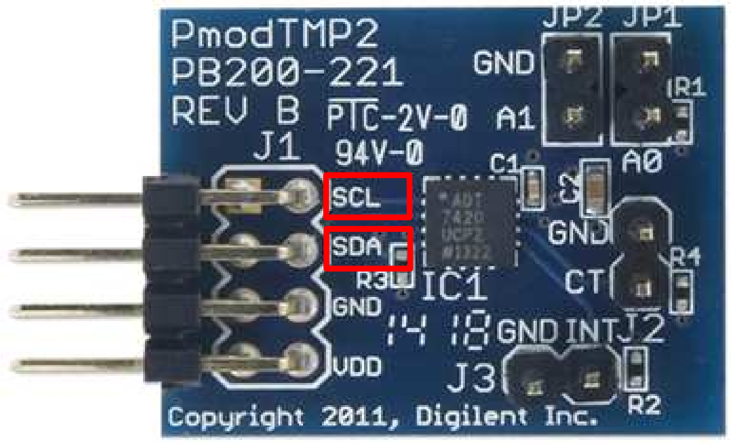


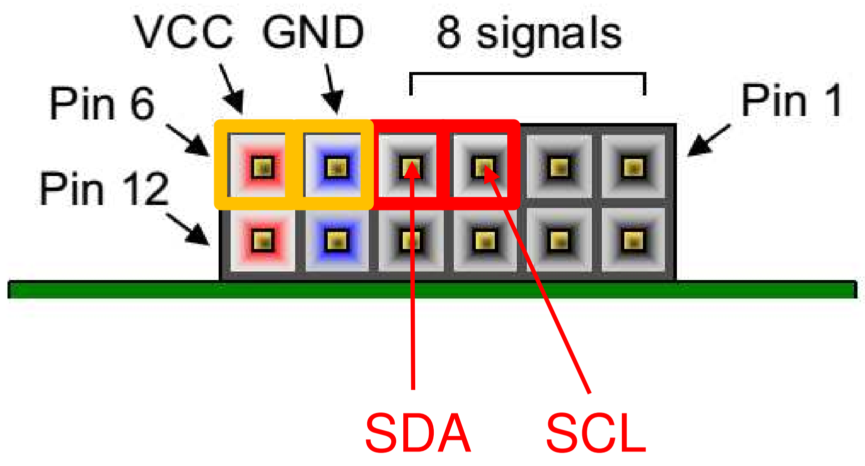
Pmod JA (XADC)	Pmod JB (Hi-Speed)	Pmod JC (Hi-Speed)	Pmod JD (Hi-Speed)	Pmod JE (Std.)	Pmod JF (MIO)
JA1: N15	JB1: T20	JC1: V15	JD1: T14	JE1: V12	JF1: MIO-13
JA2: L14	JB2: U20	JC2: W15	JD2: T15	JE2: W16	JF2: MIO-10
JA3: K16	JB3: V20	JC3: T11	JD3: P14	JE3: J15	JF3: MIO-11
JA4: K14	JB4: W20	JC4: T10	JD4: R14	JE4: H15	JF4: MIO-12
JA7: N16	JB7: Y18	JC7: W14	JD7: U14	JE7: V13	JF7: MIO-0
JA8: L15	JB8: Y19	JC8: Y14	JD8: U15	JE8: U17	JF8: MIO-9
JA9: J16	JB9: W18	JC9: T12	JD9: V17	JE9: T17	JF9: MIO-14
JA10: J14	JB10: W19	JC10: U12	JD10: V18	JE10: Y17	JF10: MIO-15

Now we use the standard PMOD **JE** 3-4 connector pins:

i2c_scl : `J15`

i2c_sda: `H15`



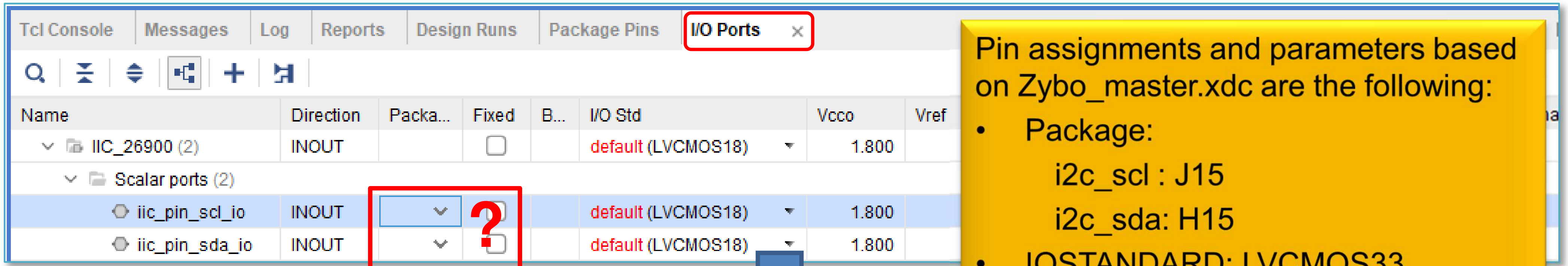


https://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPZYBO/documentation/ZYBO_RM_B_V6.pdf

I/O Planning - Correct pin assignment I.

1. One option is to use Vivado IO Planning (GUI)

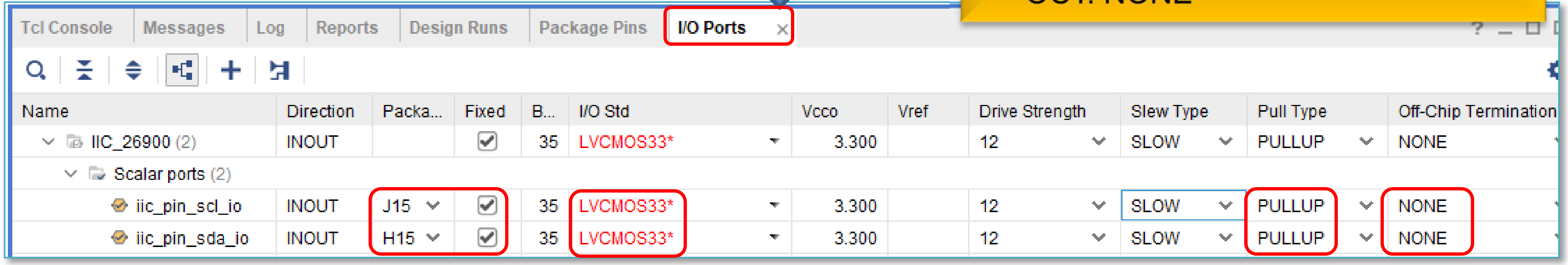
   SYNTHESIS → ***Open Synthesized Design → I/O Planning***



Name	Direction	Packa...	Fixed	B...	I/O Std	Vcco	Vref
IIC_26900 (2)	INOUT		☐		default (LVCMOS18)	1.800	
Scalar ports (2)							
iic_pin_scl_io	INOUT		☐		default (LVCMOS18)	1.800	
iic_pin_sda_io	INOUT		☐		default (LVCMOS18)	1.800	

Pin assignments and parameters based on Zybo_master.xdc are the following:

- Package:
  - i2c_scl : J15
  - i2c_sda: H15
- IOSTANDARD: LVCMOS33
- Pull type: PULLUP
- OCT: NONE



Name	Direction	Packa...	Fixed	B...	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination
IIC_26900 (2)	INOUT		✓	35	LVCMOS33*	3.300		12	SLOW	PULLUP	NONE
Scalar ports (2)											
iic_pin_scl_io	INOUT	J15	✓	35	LVCMOS33*	3.300		12	SLOW	PULLUP	NONE
iic_pin_sda_io	INOUT	H15	✓	35	LVCMOS33*	3.300		12	SLOW	PULLUP	NONE

Finally ***File → Save Constraints*** or **CTRL+S**.

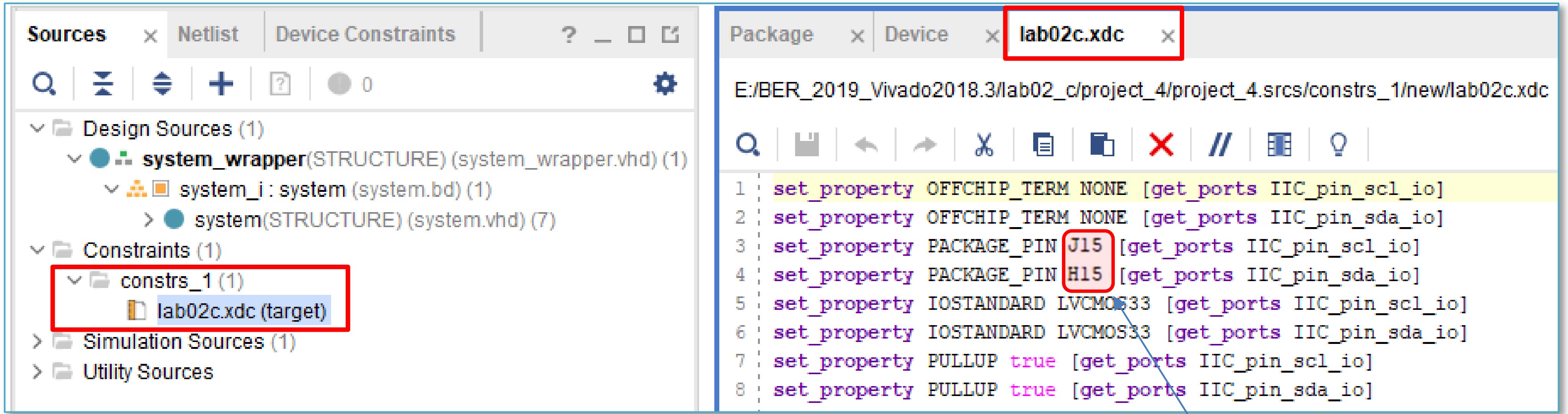
- Name the file: „`lab02c.xdc`"

IO Planning - Correct pin assignment II.

2. Another option is to edit .XDC constraints

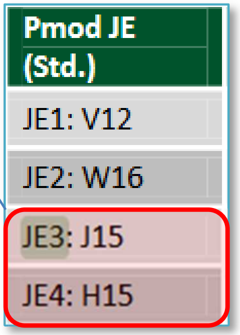
File → Add Sources → Add or Create constraints

– Create File, then giva a name: „`lab02c.xdc`”



E:/BER_2019_Vivado2018.3/lab02_c/project_4/project_4.srcs/constrs_1/new/lab02c.xdc

```
set_property OFFCHIP_TERM NONE [get_ports IIC_pin_scl_io]
set_property OFFCHIP_TERM NONE [get_ports IIC_pin_sda_io]
set_property PACKAGE_PIN J15 [get_ports IIC_pin_scl_io]
set_property PACKAGE_PIN H15 [get_ports IIC_pin_sda_io]
set_property IOSTANDARD LVCMOS33 [get_ports IIC_pin_scl_io]
set_property IOSTANDARD LVCMOS33 [get_ports IIC_pin_sda_io]
set_property PULLUP true [get_ports IIC_pin_scl_io]
set_property PULLUP true [get_ports IIC_pin_sda_io]
```

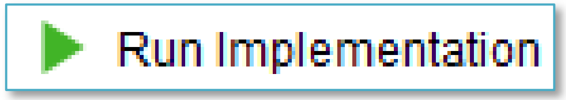


Pmod JE (Std.)
JE1: V12
JE2: W16
JE3: J15
JE4: H15

Finally ***File*** → ***Save Constraints*** or **CTRL+S**.
Name the file: „`lab02c.xdc`”

Implementation and Bitstream generation

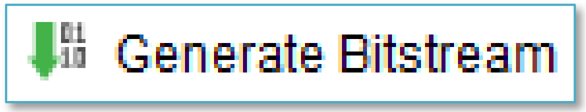
- Flow Navigator menu → **Run Implementation**



  - 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**

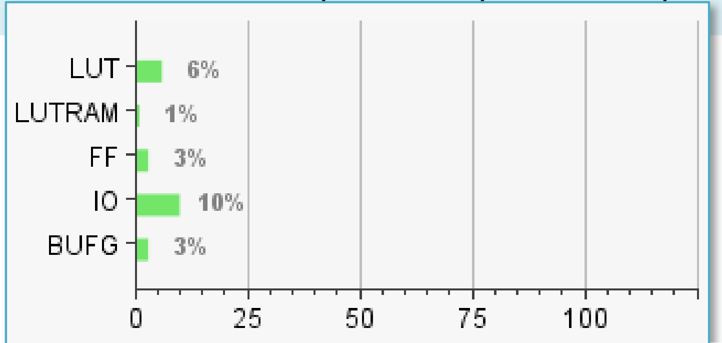


Q&A 1.) Reports

- How many resources are occupied on PL-side?

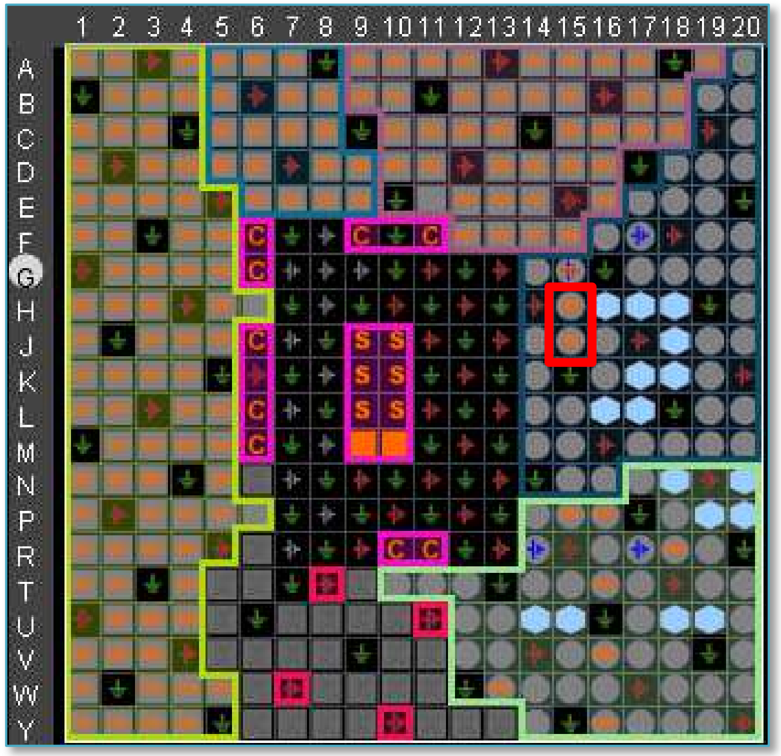
  Reports → Report Utilization (or Project Summary Σ)

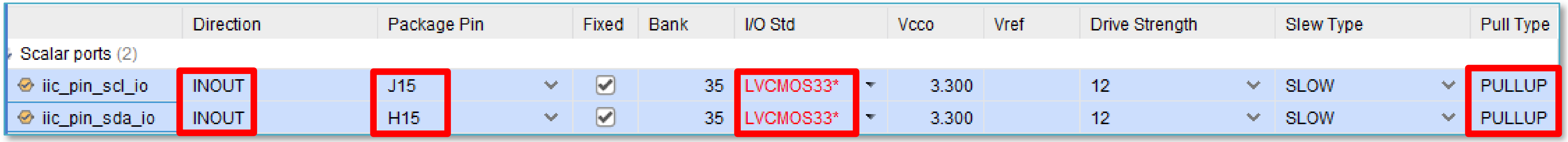
```
+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs                 |  971 |     0 |     17600 |  5.52 |
|   LUT as Logic             |  899 |     0 |     17600 |  5.11 |
|   LUT as Memory            |   72 |     0 |      6000 |  1.20 |
|     LUT as Distributed RAM |    0 |     0 |           |       |
|     LUT as Shift Register  |   72 |     0 |           |       |
| Slice Registers            | 1196 |     0 |     35200 |  3.40 |
|   Register as Flip Flop    | 1196 |     0 |     35200 |  3.40 |
|   Register as Latch        |    0 |     0 |     35200 |  0.00 |
| F7 Muxes                   |    8 |     0 |      8800 |  0.09 |
| F8 Muxes                   |    4 |     0 |      4400 |  0.09 |
+----------------------------+------+-------+-----------+-------+
```



Q&A 2.) I/O Planning

- Check where the I2C pins are located on the FPGA?
  - Package Pins?
  - Direction?
  - I/O Standard?
  - Pull type?



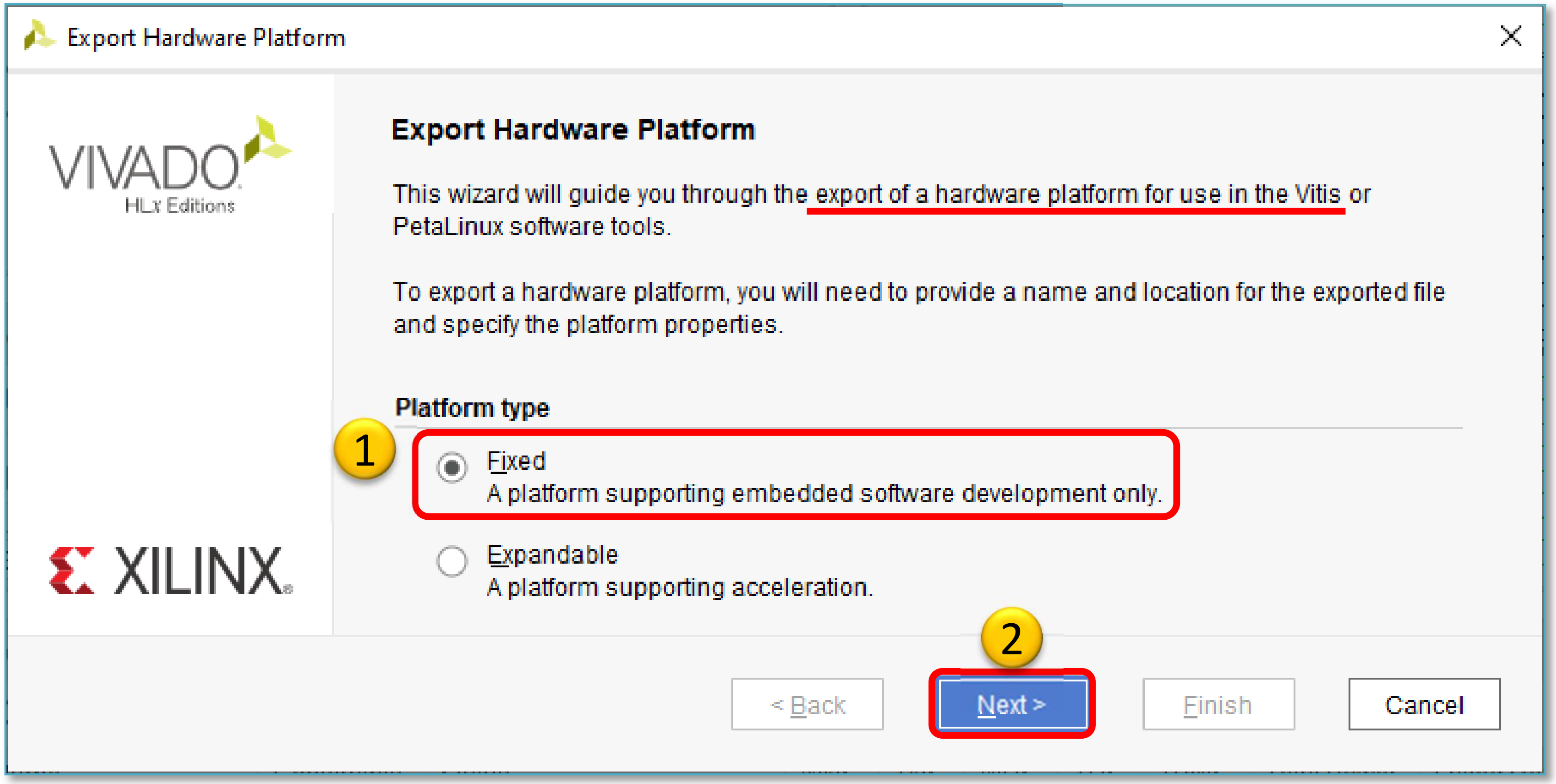


	Direction	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
Scalar ports (2)										
iic_pin_scl_io	INOUT	J15	✓	35	LVCMOS33*	3.300		12	SLOW	PULLUP
iic_pin_sda_io	INOUT	H15	✓	35	LVCMOS33*	3.300		12	SLOW	PULLUP

VIVADO Export HW → VITIS (~SDK)

- **File → Export → Export Hardware…**

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


Export Hardware Platform

Export Hardware Platform

This wizard will guide you through the export of a hardware platform for use in the Vitis or PetaLinux software tools.

To export a hardware platform, you will need to provide a name and location for the exported file and specify the platform properties.

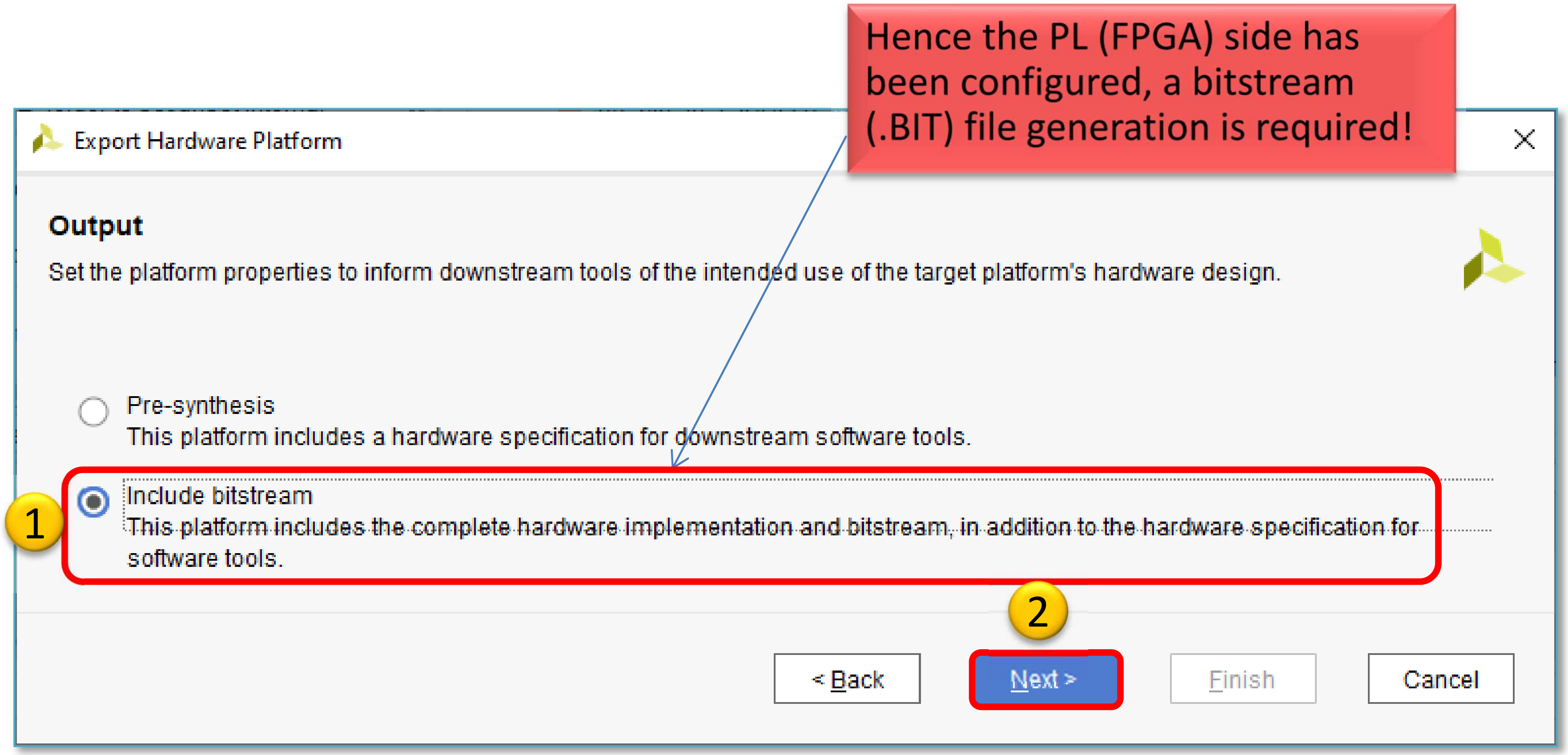
Platform type

[✓ Fixed
A platform supporting embedded software development only.]

Expandable
A platform supporting acceleration.

VIVADO Export HW → VITIS (cont.)

Select „Include bitstream” option as output:


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

Export Hardware Platform

Output

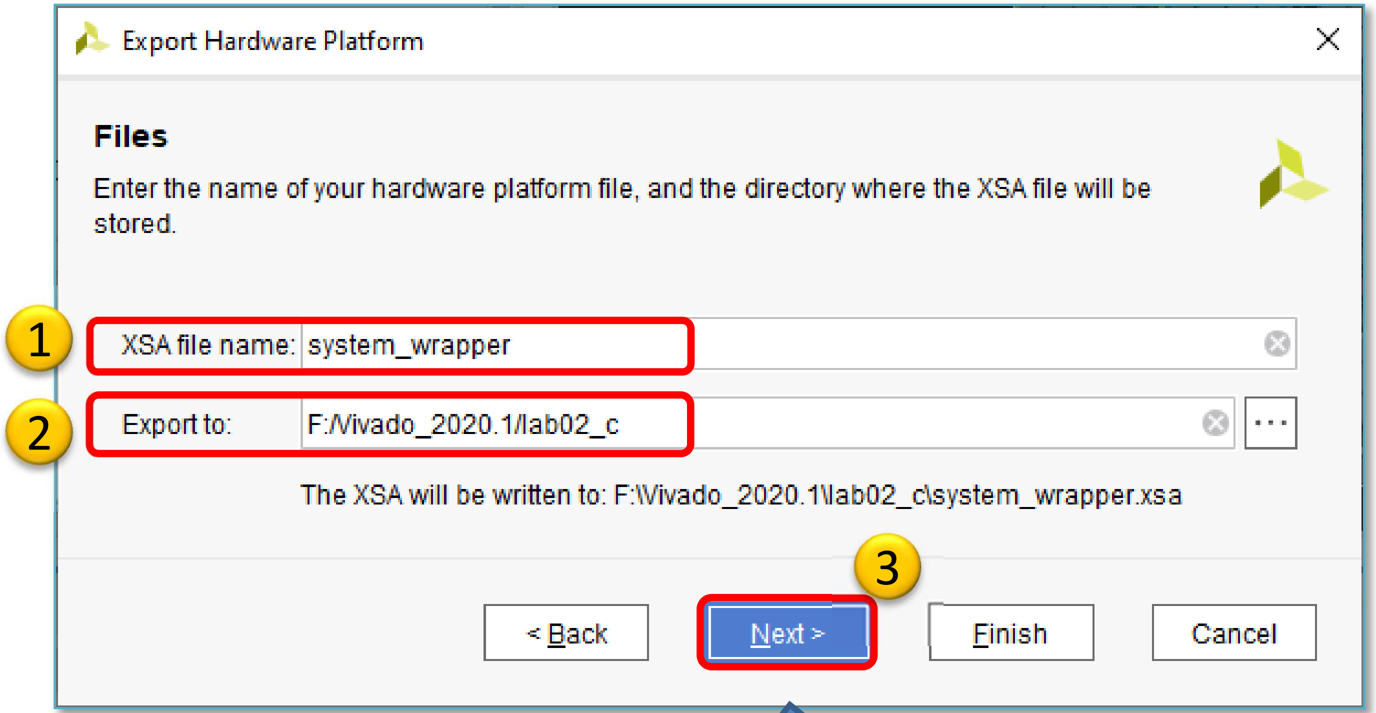
Set the platform properties to inform downstream tools of the intended use of the target platform's hardware design.

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

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

Export HW → VITIS (cont.)

Set XSA* file name and export directory path:



Export Hardware Platform

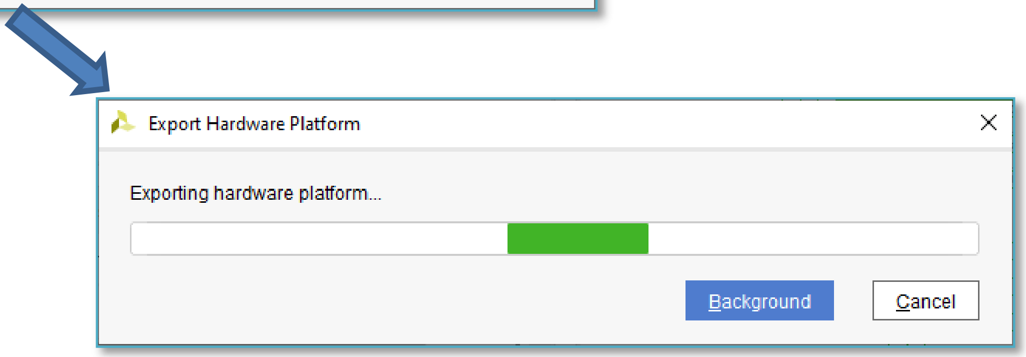
Files

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_c

The XSA will be written to: F:\Vivado_2020.1\lab02_c\system_wrapper.xsa



Export Hardware Platform

Exporting hardware platform...

*Xilinx® Support Archive: new hw descriptor format since VITIS 2020.x (see the next slide)

USING XILINX VITIS

LAB02_C. Creating a software test application

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

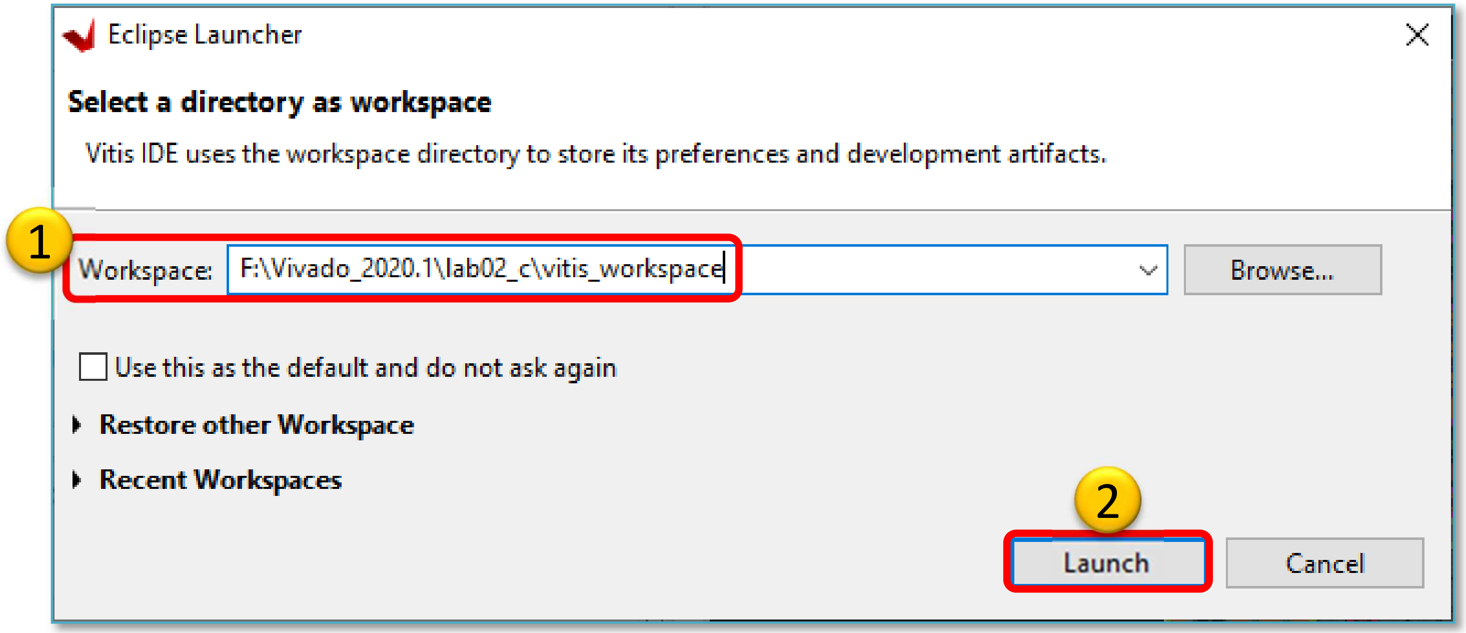
From Vivado: Tools menu → Launch VITIS IDE

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_c`):
  - Recommended to use `vitis_workspace` as a subdirectory in your lab folder. Then Launch...



Eclipse Launcher

Select a directory as workspace

Vitis IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: F:\Vivado_2020.1\lab02_c\vitis_workspace

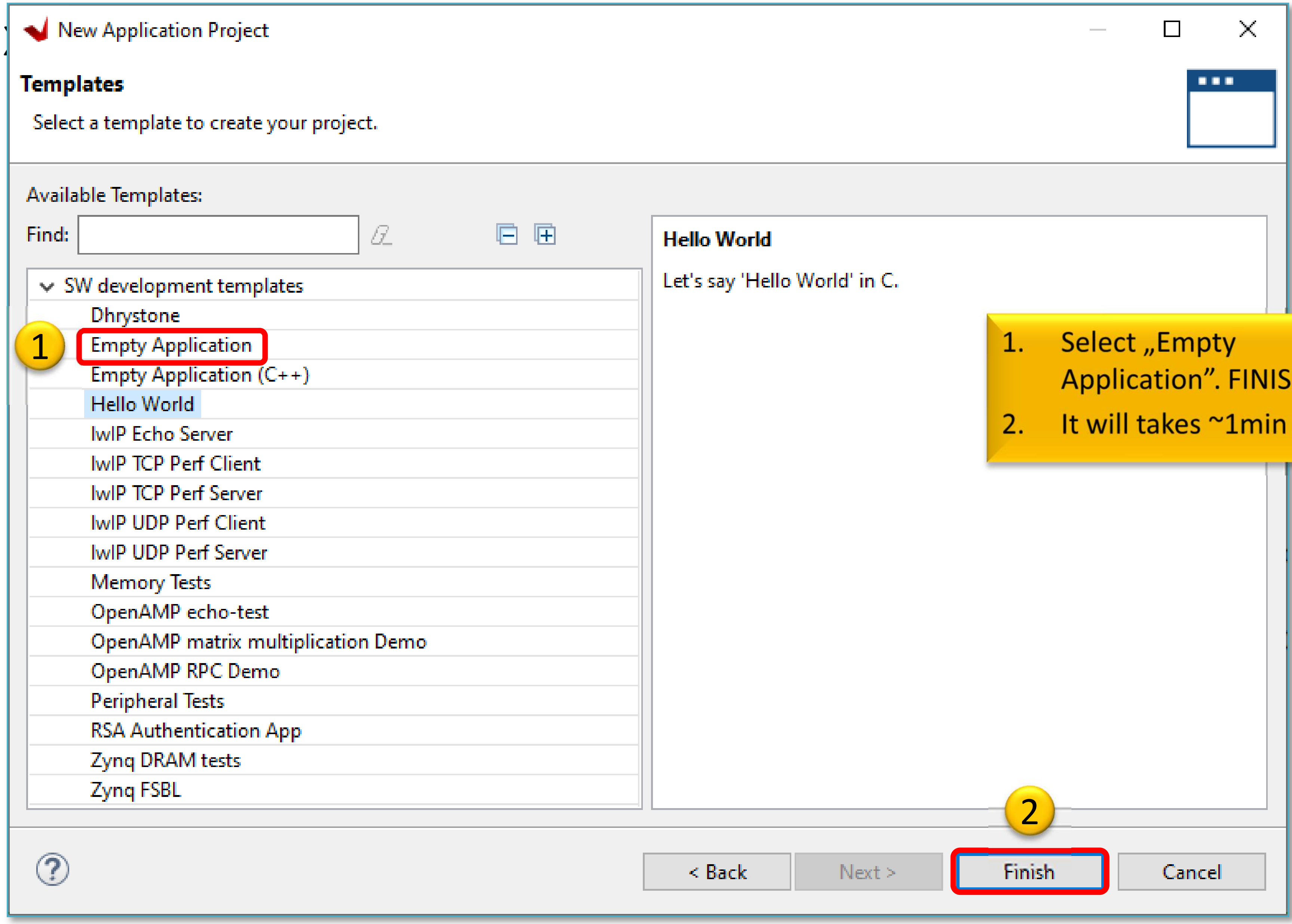
☐ Use this as the default and do not ask again

Xilinx VITIS – Create Application

Recall the steps of the former LAB01/LAB02_A!

1. **Create a new application project**
   - File → New → Application Project…
2. **Platform – Create a new platform from HW (XSA)**
   - Browse… for LAB02_C `system_wrapper.xsa`. Open it.
   - Do not select the „*Generate boot components*”
3. **Application project details**
   - Type „`TestApp`” as project name
   - Type „`TestApp_system`” as system project name
   - Select `ps7_cortexa9_0` as target ARM core 0
4. **Domain:** leave settings as default (standalone)

Example I.) Creating TestApp application



New Application Project

Templates

Select a template to create your project.

Available Templates:

Find:

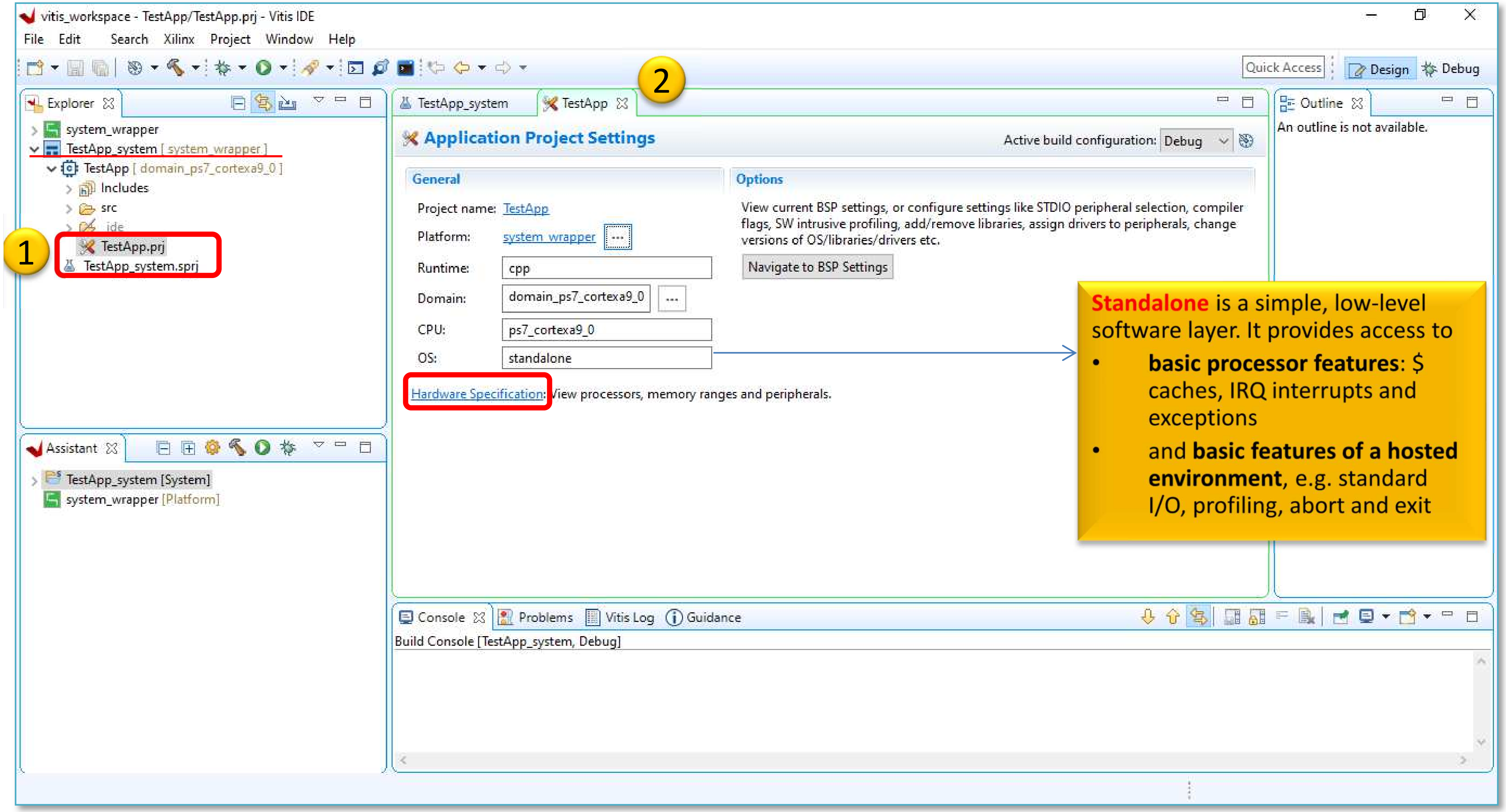
- SW development templates
  - Dhrystone
  - Empty Application
  - Empty Application (C++)
  - Hello World
  - lwIP Echo Server
  - lwIP TCP Perf Client
  - lwIP TCP Perf Server
  - lwIP UDP Perf Client
  - lwIP UDP Perf Server
  - Memory Tests
  - OpenAMP echo-test
  - OpenAMP matrix multiplication 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 ☺

VITIS GUI – Main window



vitis_workspace - TestApp/TestApp.prj - Vitis IDE

Application Project Settings

Active build configuration: Debug

General

Project name: TestApp

Platform: system_wrapper

Runtime: cpp

Domain: domain_ps7_cortexa9_0

CPU: ps7_cortexa9_0

OS: standalone

Hardware Specification: View processors, memory ranges and peripherals.

Options

View current BSP settings, or configure settings like STDIO peripheral selection, compiler flags, SW intrusive profiling, add/remove libraries, assign drivers to peripherals, change versions of OS/libraries/drivers etc.

Navigate to BSP Settings

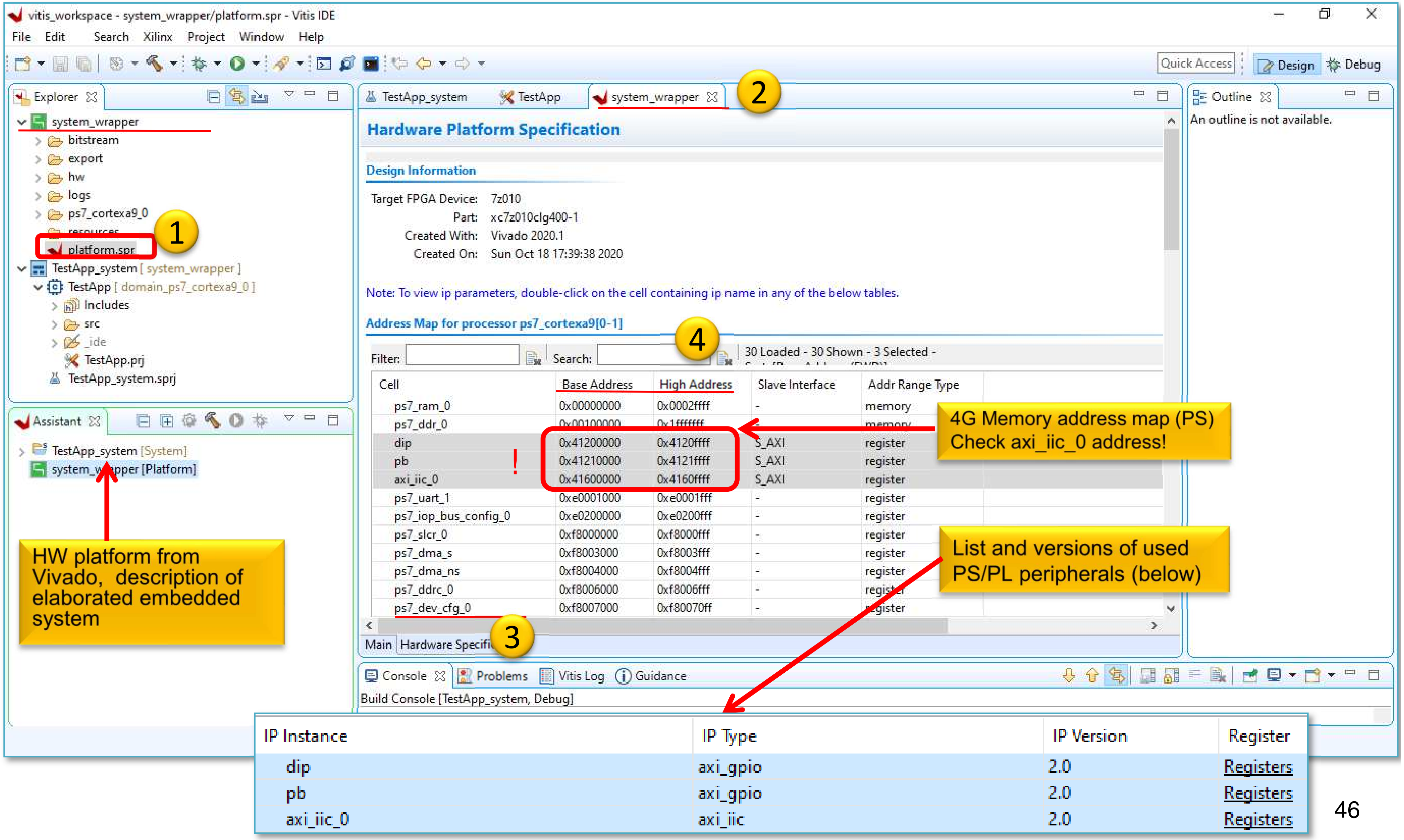
An outline is not available.

Build Console [TestApp_system, Debug]

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



vitis_workspace - system_wrapper/platform.spr - Vitis IDE

Hardware Platform Specification

Design Information

Target FPGA Device:	7z010
Part:	xc7z010clg400-1
Created With:	Vivado 2020.1
Created On:	Sun Oct 18 17:39:38 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]

30 Loaded - 30 Shown - 3 Selected -

Cell	Base Address	High Address	Slave Interface	Addr Range Type
ps7_ram_0	0x00000000	0x0002ffff	-	memory
ps7_ddr_0	0x00100000	0x1fffffff	-	memory
dip	0x41200000	0x4120ffff	S_AXI	register
pb	0x41210000	0x4121ffff	S_AXI	register
axi_iic_0	0x41600000	0x4160ffff	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

!

4G Memory address map (PS)
Check axi_iic_0 address!

List and versions of used PS/PL peripherals (below)

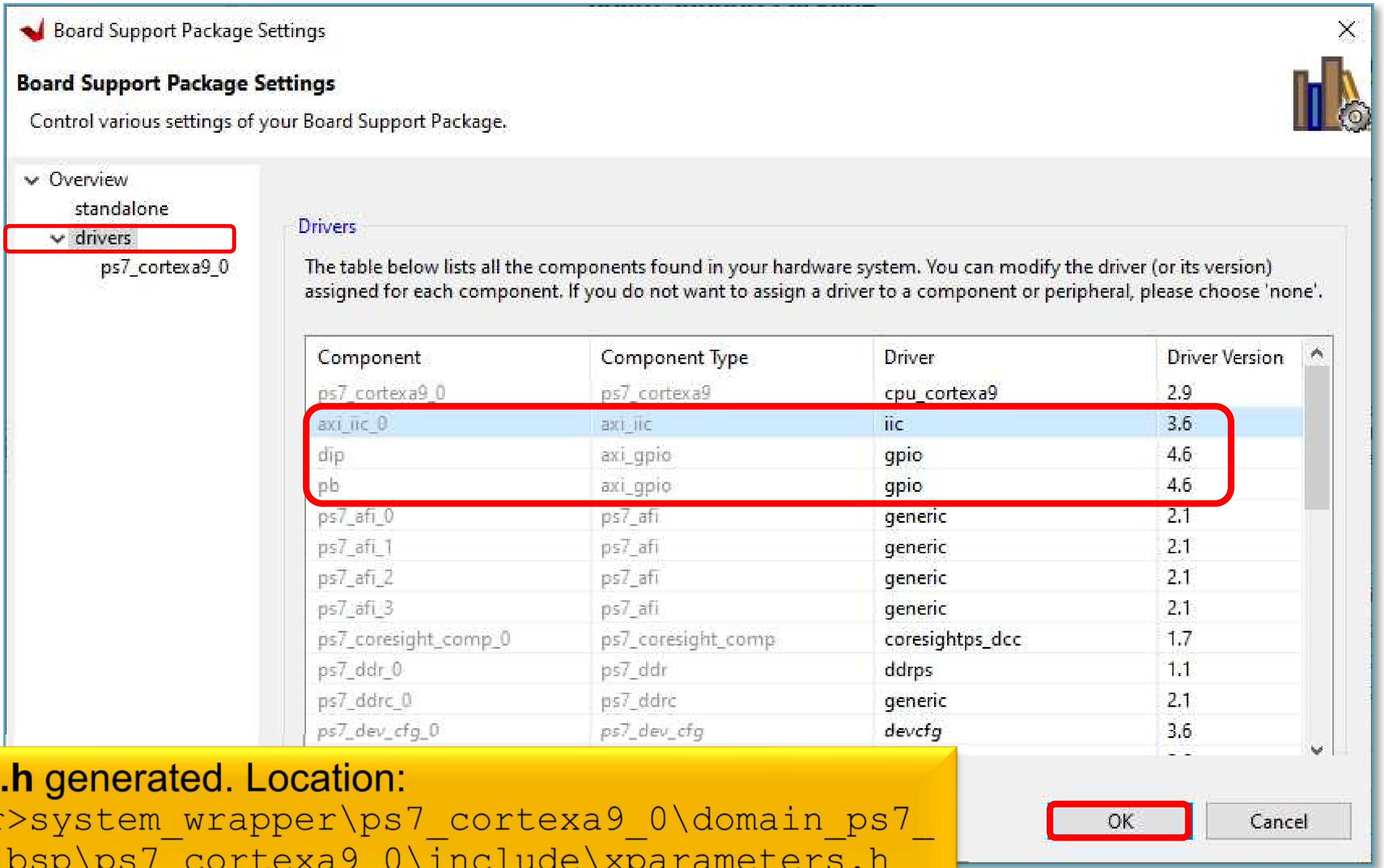
HW platform from Vivado, description of elaborated embedded system

IP Instance	IP Type	IP Version	Register
dip	axi_gpio	2.0	Registers
pb	axi_gpio	2.0	Registers
axi_iic_0	axi_iic	2.0	Registers

Build Console [TestApp_system, Debug]

VITIS – BSP Board Support Package

- **Software Platform Settings**
  - Selected OS: *standalone*
  - Check the supported SW drivers (and its version): „`axi_iic`”



Board Support Package Settings

Control various settings of your Board Support Package.

Drivers

The table below lists all the components found in your hardware system. You can modify the driver (or its version) assigned for each component. If you do not want to assign a driver to a component or peripheral, please choose 'none'.

Component	Component Type	Driver	Driver Version
ps7_cortexa9_0	ps7_cortexa9	cpu_cortexa9	2.9
axi_iic_0	axi_iic	iic	3.6
dip	axi_gpio	gpio	4.6
pb	axi_gpio	gpio	4.6
ps7_afi_0	ps7_afi	generic	2.1
ps7_afi_1	ps7_afi	generic	2.1
ps7_afi_2	ps7_afi	generic	2.1
ps7_afi_3	ps7_afi	generic	2.1
ps7_coresight_comp_0	ps7_coresight_comp	coresightps_dcc	1.7
ps7_ddr_0	ps7_ddr	ddrps	1.1
ps7_ddrc_0	ps7_ddrc	generic	2.1
ps7_dev_cfg_0	ps7_dev_cfg	devcfg	3.6

xparameters.h generated. Location:
`<projectdir>system_wrapper\ps7_cortexa9_0\domain_ps7_cortexa9_0\bsp\ps7_cortexa9_0\include\xparameters.h`

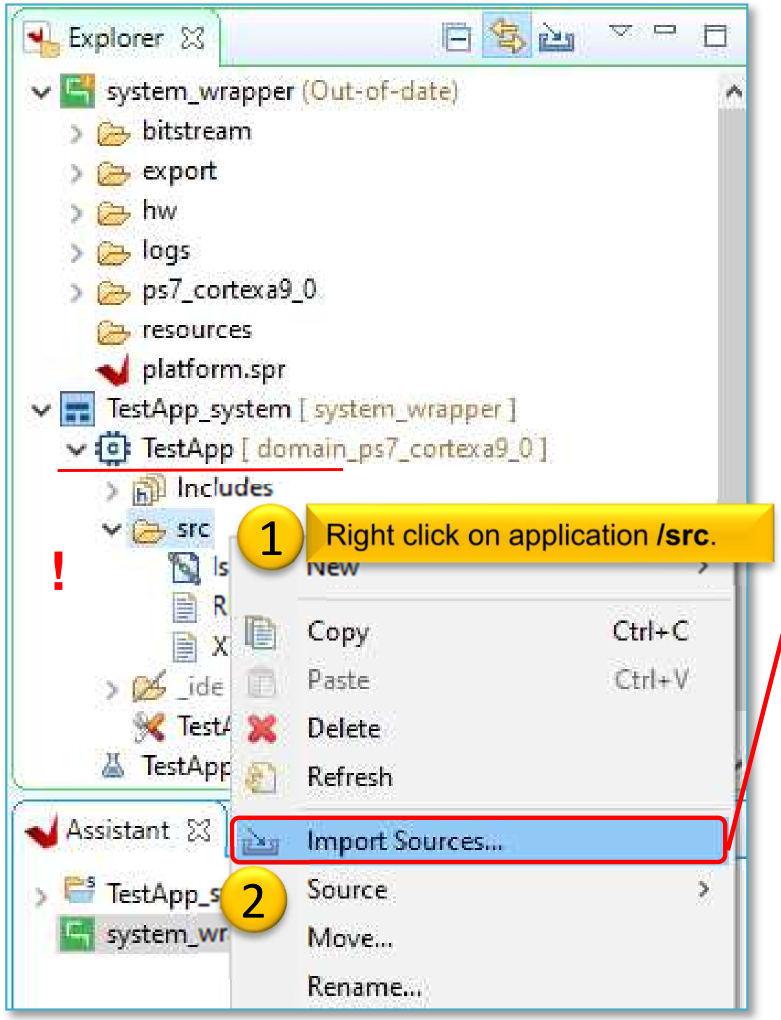
Q & A 1.)

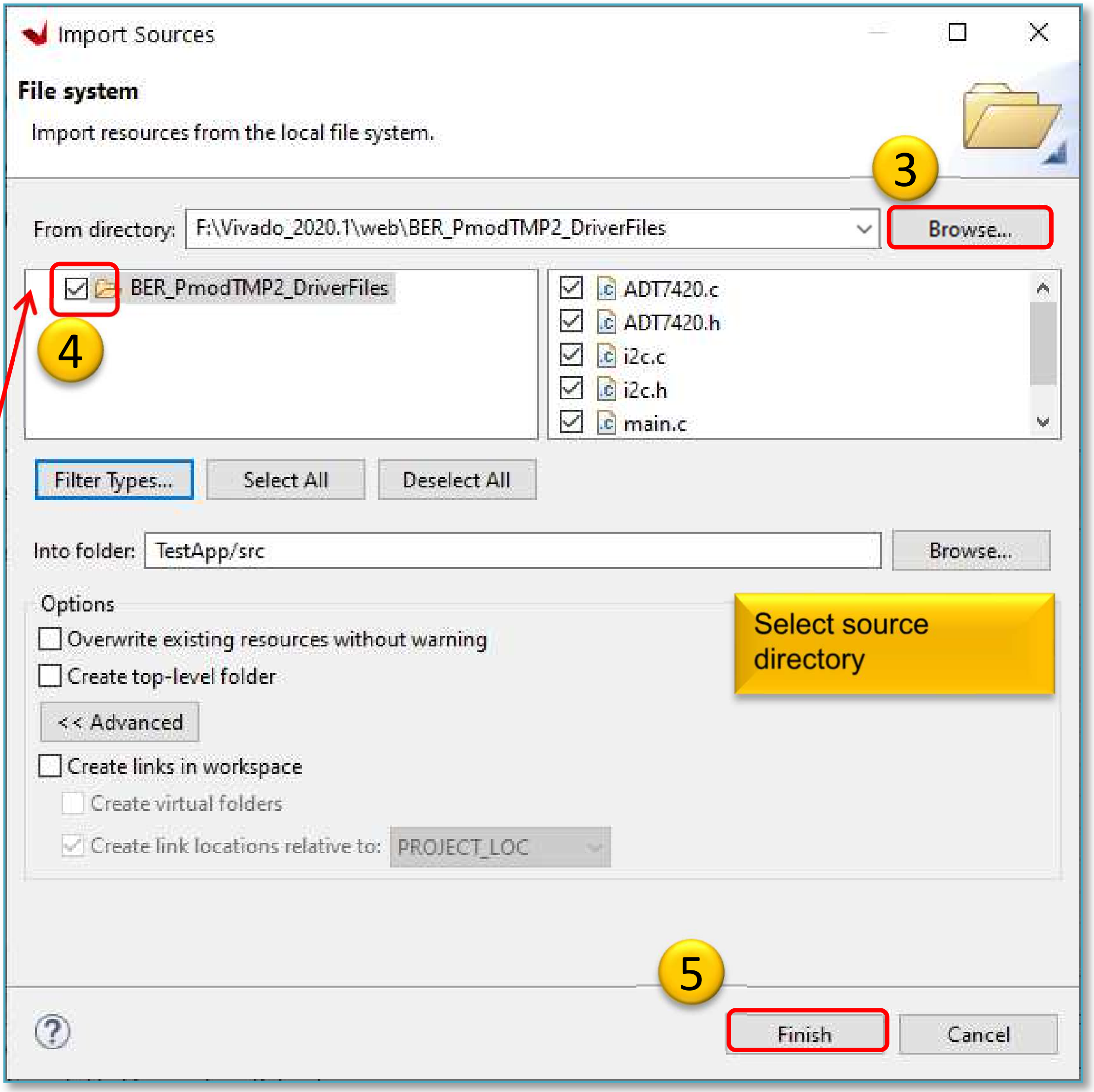
- **What is the *IP type* and *IP version* of „axi_iic_0” instance?**
  - axi_iic,
  - v2.0
- **What is the driver name and version of it?**
  - iic,
  - v3.6
- **Calculate what size they are?**
  - axi_iic_0: `0x4160 0000-0x4160 ffff` = 64 KByte

Import C/C++ source(s)

Download the archive from lab's website: BER_PmodTMP2_DriverFiles.zip

1. Download and unpack the .zip archive
2. Import all sources to the application project





From directory: F:\Vivado_2020.1\web\BER_PmodTMP2_DriverFiles

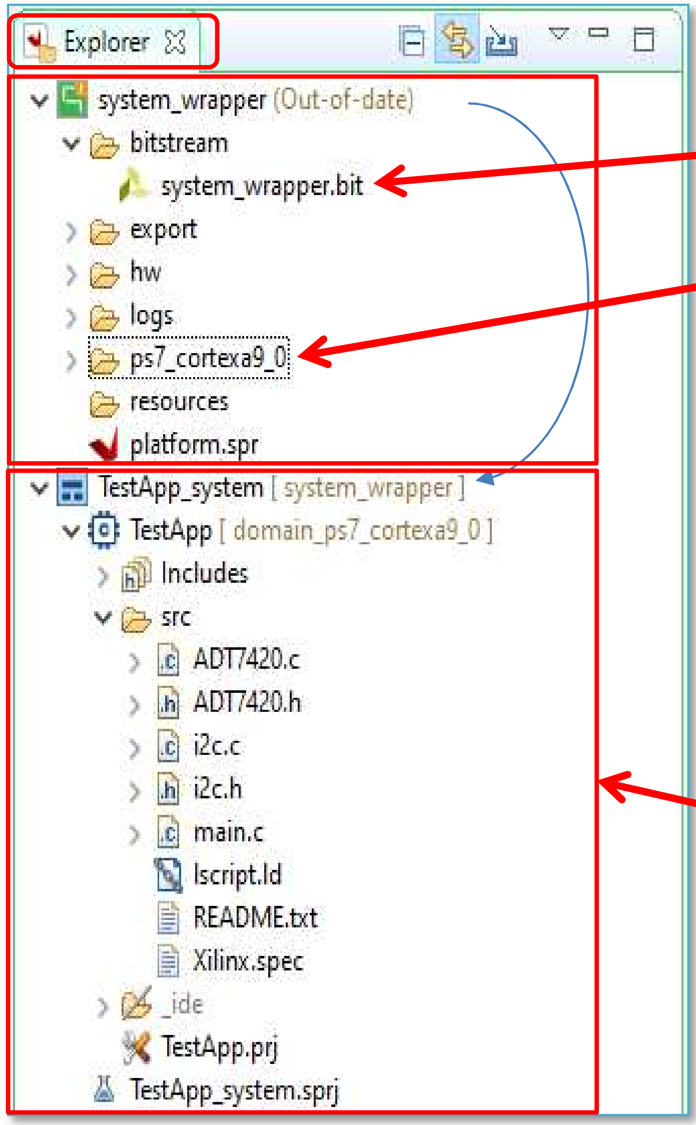
☑ BER_PmodTMP2_DriverFiles

☑ ADT7420.c
☑ ADT7420.h
☑ i2c.c
☑ i2c.h
☑ main.c

Into folder: TestApp/src

Options
☐ Overwrite existing resources without warning
☐ Create top-level folder
☐ Create links in workspace
☐ Create virtual folders
☑ Create link locations relative to: PROJECT_LOC

VITIS – Project Explorer / Hierarchy



```
Explorer
v system_wrapper (Out-of-date)
  v bitstream
      system_wrapper.bit
  > export
  > hw
  > logs
  > ps7_cortexa9_0
    resources
    platform.spr
v TestApp_system [ system_wrapper ]
  v TestApp [ domain_ps7_cortexa9_0 ]
    > Includes
    v src
      > ADT7420.c
      > ADT7420.h
      > i2c.c
      > i2c.h
      > main.c
        lscript.ld
        README.txt
        Xilinx.spec
    > _ide
      TestApp.prj
    TestApp_system.sprj
```

`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)**

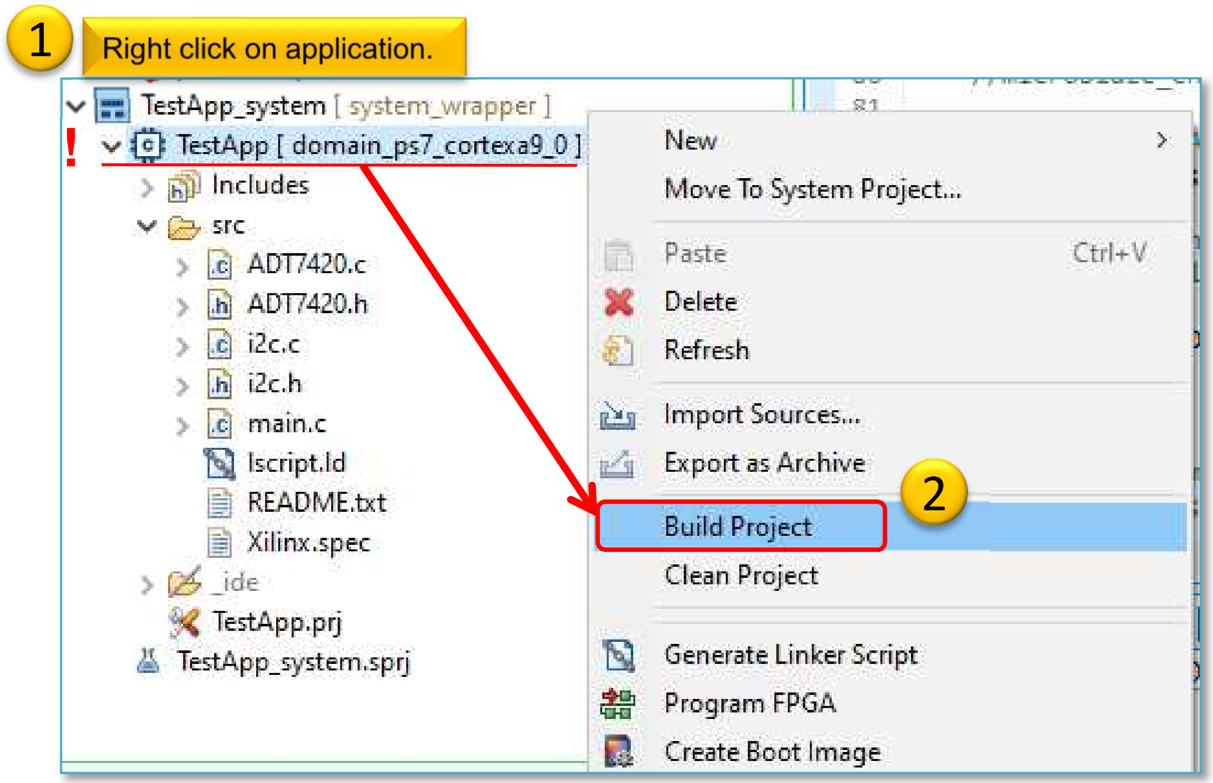
`TestApp_system` **as system_project** contains

SW: `TestApp` (SW application)

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

Build project

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



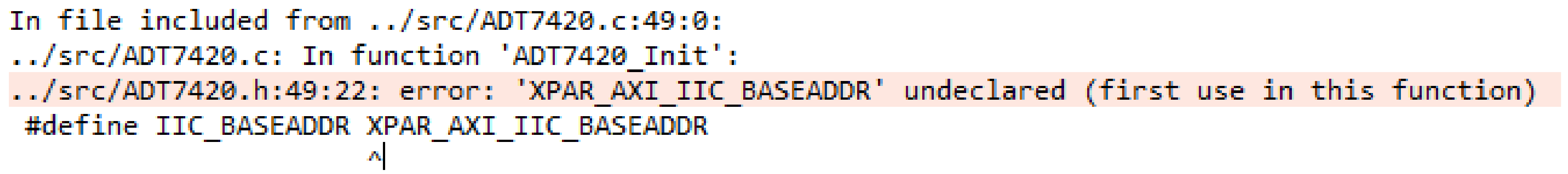
Build project – Result (Console)

```
'Building target: TestApp.elf'
'Invoking: ARM v7 gcc linker'
arm-none-eabi-gcc -mcpu=cortex-a9 -mfpu=vfpv3 -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.

Correct TestApp

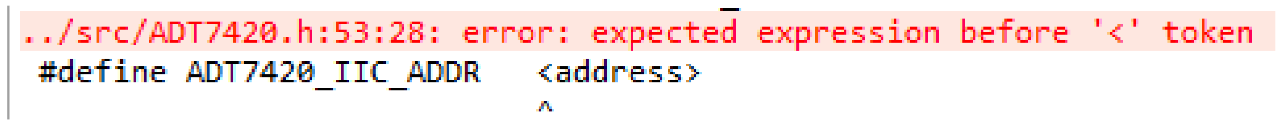
1.)



```
In file included from ../src/ADT7420.c:49:0:
../src/ADT7420.c: In function 'ADT7420_Init':
../src/ADT7420.h:49:22: error: 'XPAR_AXI_IIC_BASEADDR' undeclared (first use in this function)
 #define IIC_BASEADDR XPAR_AXI_IIC_BASEADDR
                      ^
```

– Solution: check and correct the XPAR_AXI_IIC_BASEADDR define in the ADT7420.h based on `xparameters.h` file

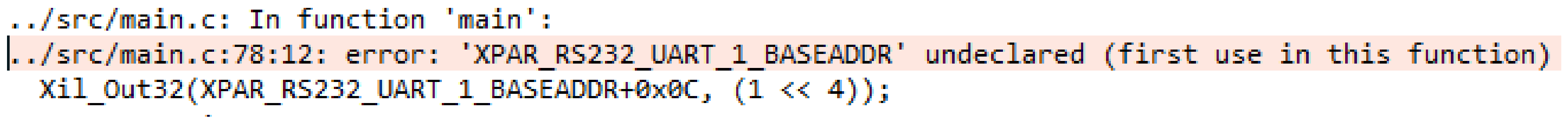
2.)



```
../src/ADT7420.h:53:28: error: expected expression before '<' token
 #define ADT7420_IIC_ADDR    <address>
                             ^
```

– Solution: examine the ADT7420_IIC_ADDR for correct addressing (see PMOD TMP2 datasheet and former slide 20-21 pp)

3.)



```
../src/main.c: In function 'main':
../src/main.c:78:12: error: 'XPAR_RS232_UART_1_BASEADDR' undeclared (first use in this function)
  Xil_Out32(XPAR_RS232_UART_1_BASEADDR+0x0C, (1 << 4));
```

– Solution: find and check the define in `ADT7240.h` file

If they has been successfully corrected uncomment lines 90-94 (while) in the main.cpp.

Build

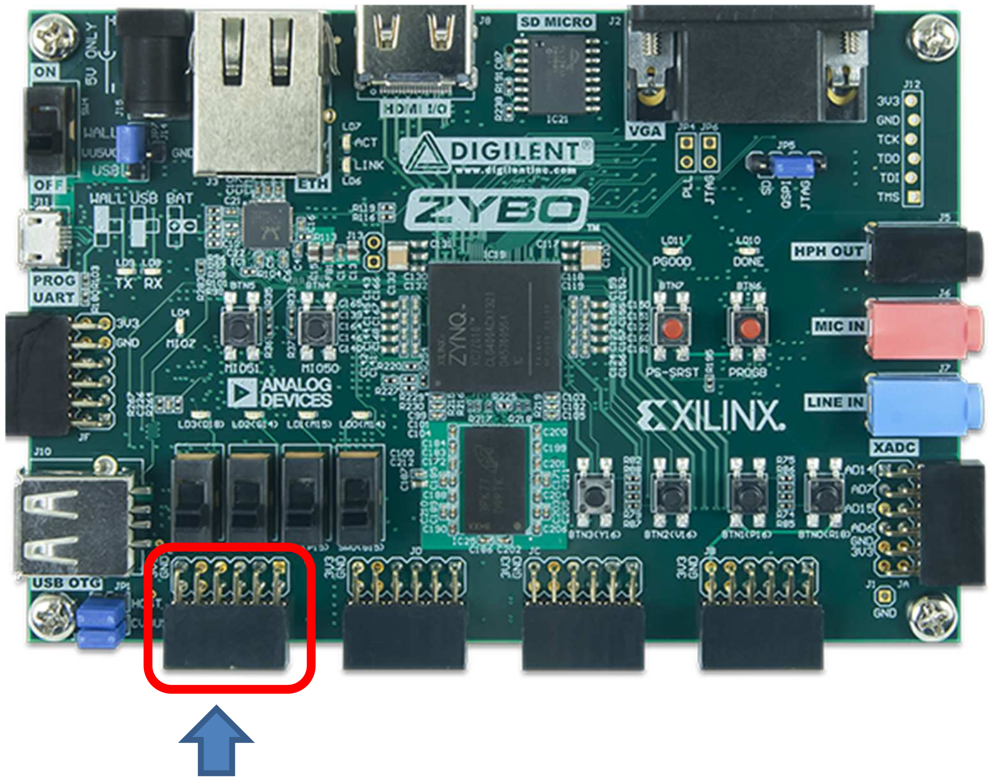
- Generate Linker Script to the interanal on-chip PS7 RAM0
  - Set the Heap / Stack size to **2KB**!
- Now rebuild the TestApp again.

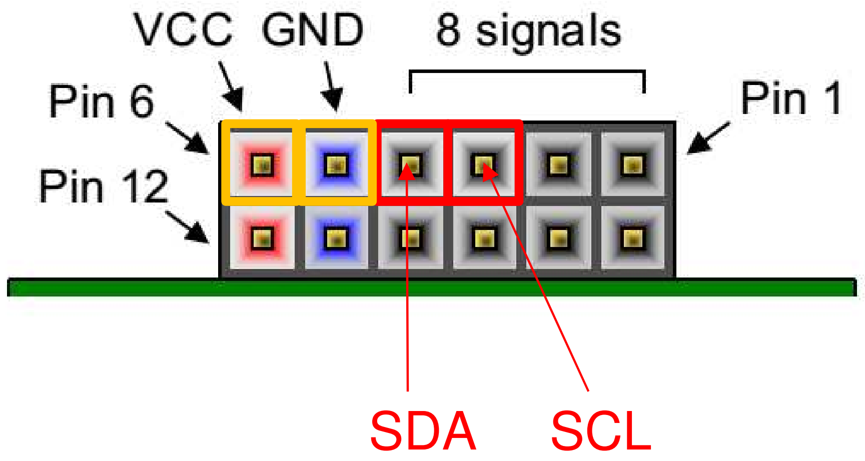
Q: What is the size of TestApp.elf binary?

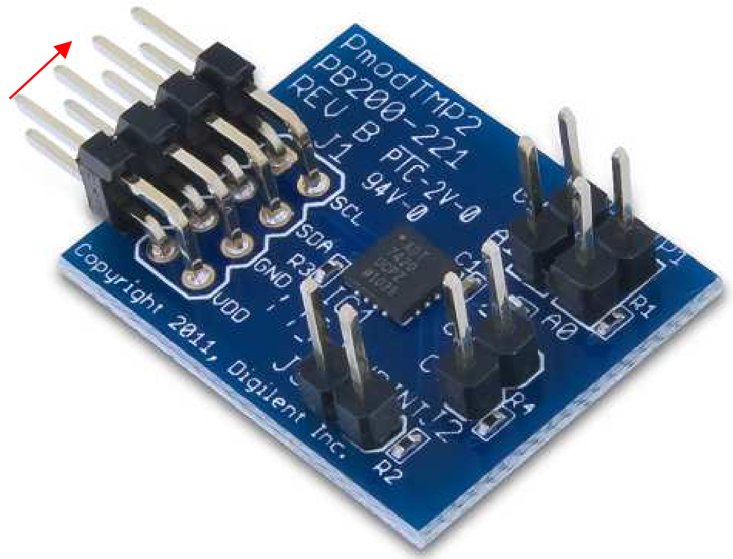
```
'Invoking: ARM v7 Print Size'
arm-none-eabi-size TestApp.elf  |tee "TestApp.elf.size"
   text    data     bss     dec     hexfilename
  27468    1144   10296   38908    97fcTestApp.elf
'Finished building: TestApp.elf.size'
```

Connect PMOD TMP2 to ZyBo

- Be sure the proper connection for PMOD JE conn.



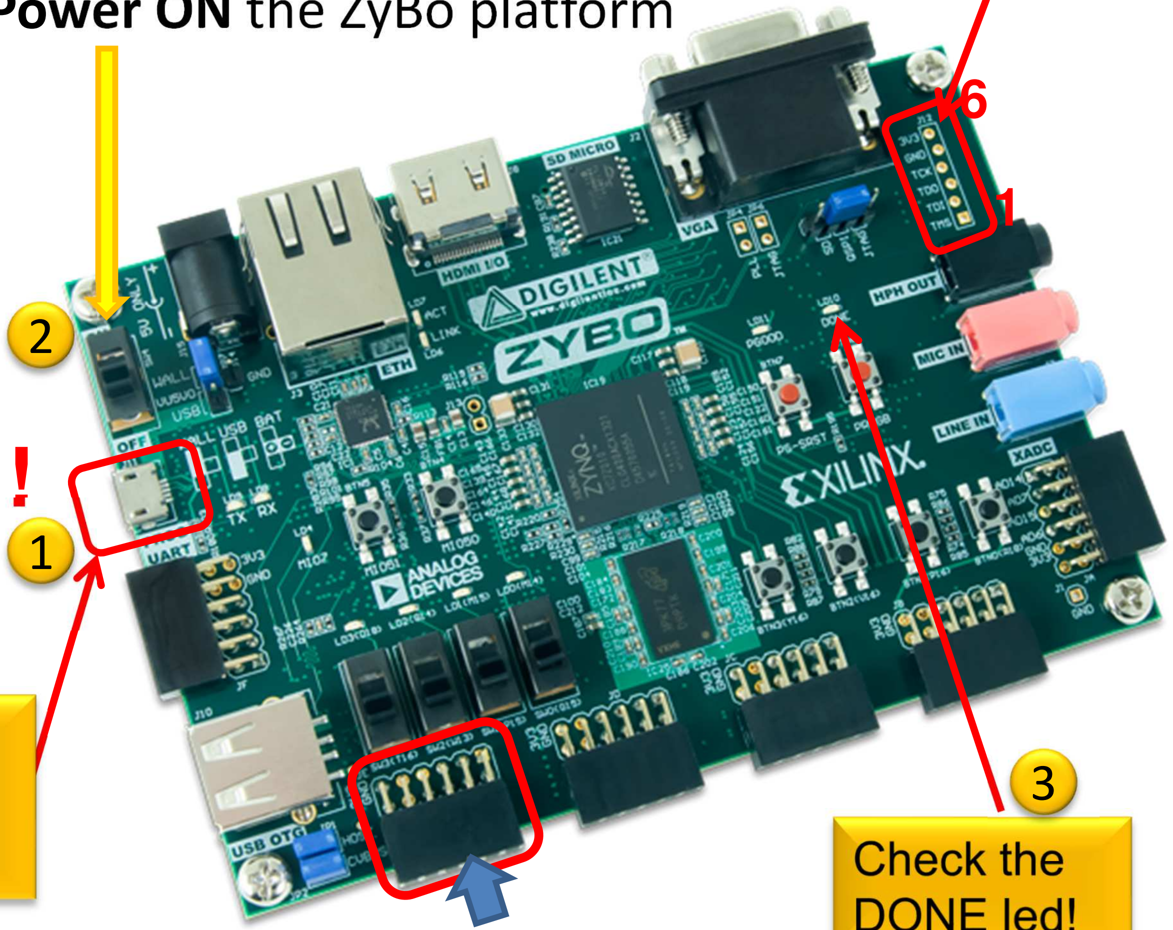




Embedded system and software test verification

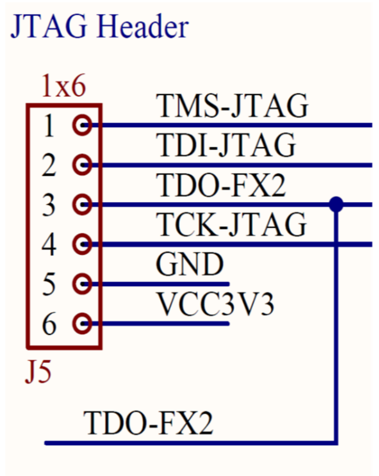
1. **Connect** the USB-serial cable (power+programmer functionality). Please check:
   - JP7 jumper = USB power!
   - JP5 jumper = JTAG mode!
2. Now **Power ON** the ZyBo platform

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



We use the USB-serial connector.

Check the DONE led!

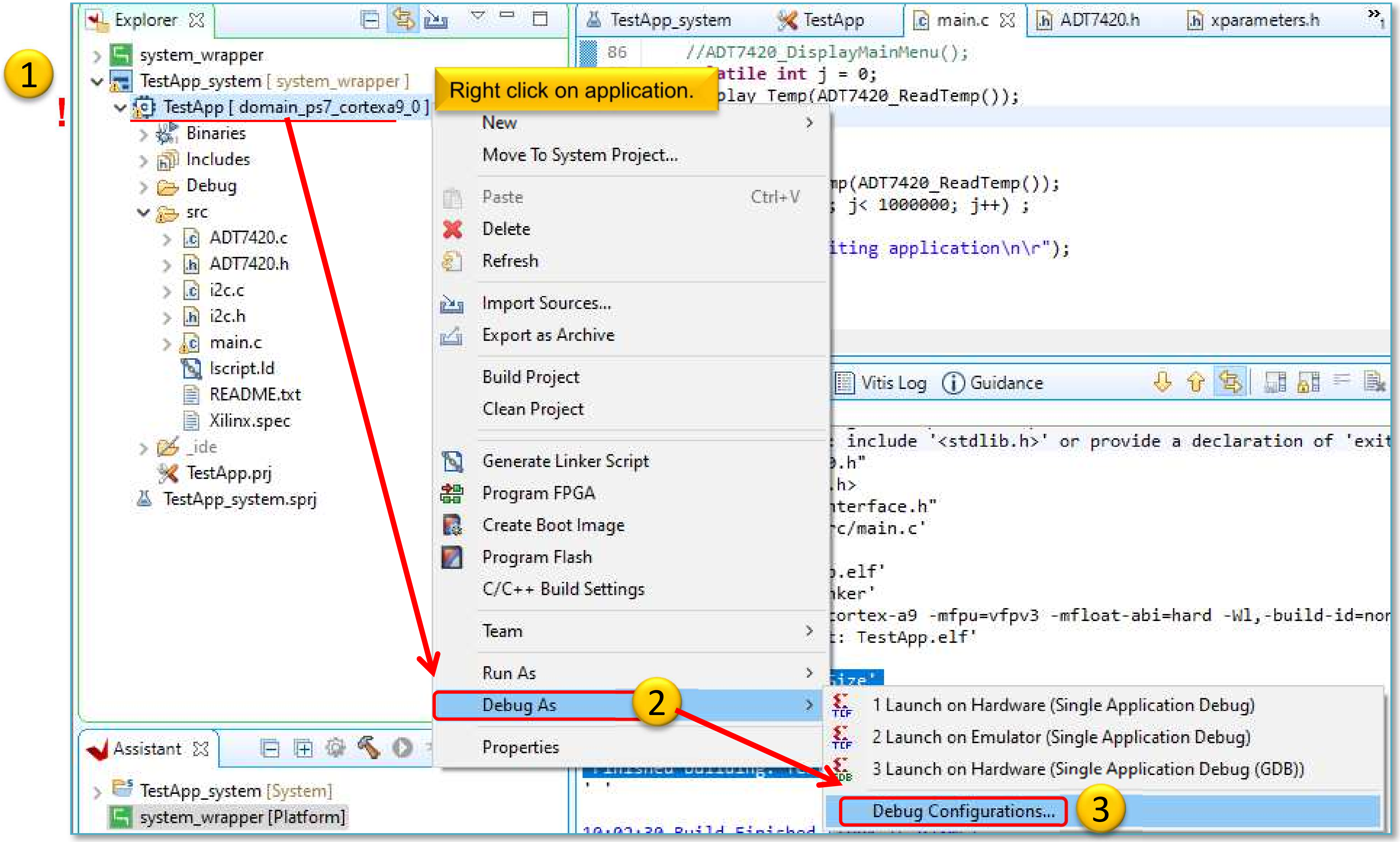


ZyBo – Xilinx USB programming cable

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

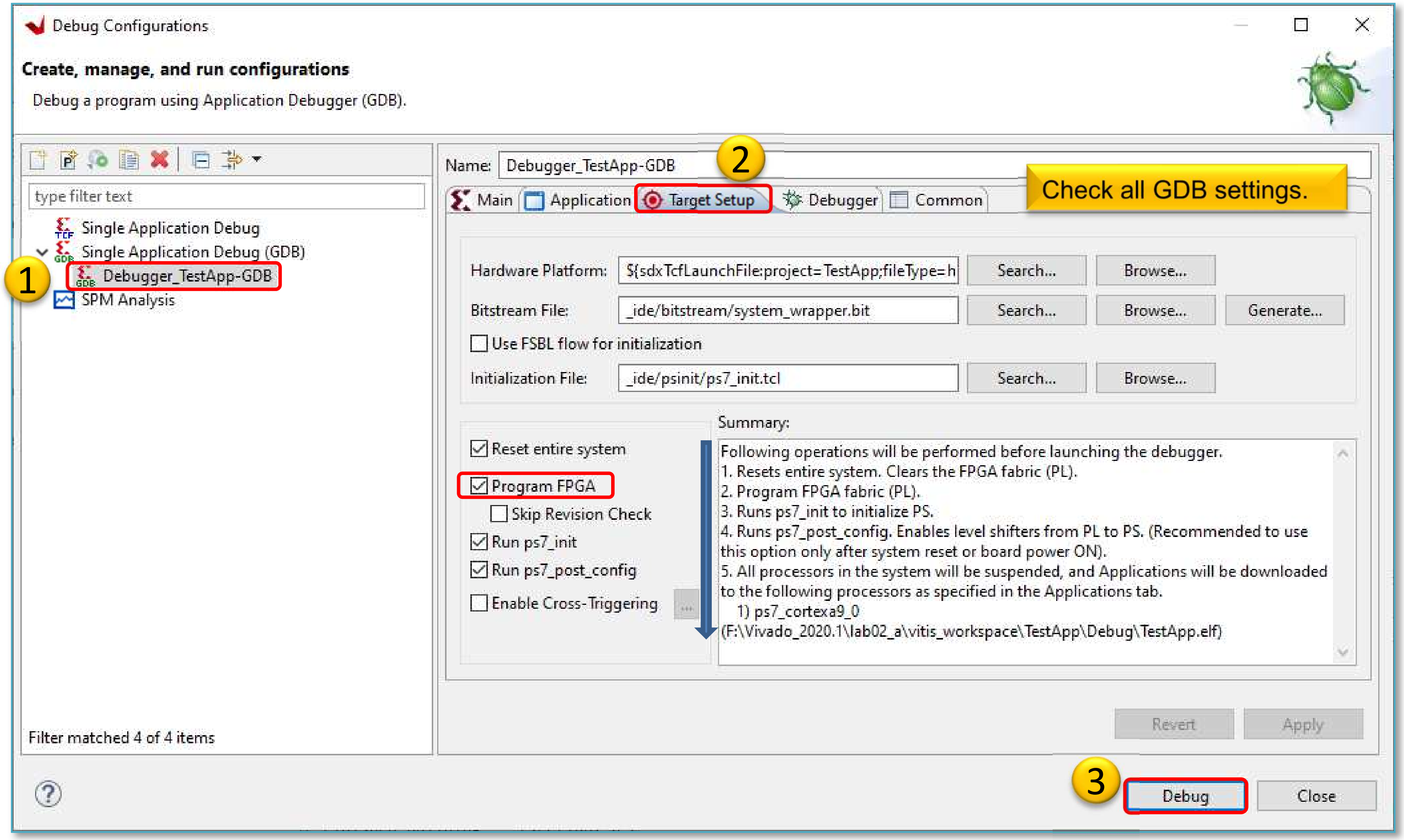
- **Select the application** (`TestApp`) in the Project Explorer



```
//ADT7420_DisplayMainMenu();
```

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).

type filter text

- Single Application Debug
- Single Application Debug (GDB)
  - Debugger_TestApp-GDB
- SPM Analysis

Filter matched 4 of 4 items

Name: Debugger_TestApp-GDB

Main | Application | Target Setup | Debugger | Common

Check all GDB settings.

Hardware Platform: ${sdxTcfLaunchFile:project=TestApp;fileType=h

Bitstream File: _ide/bitstream/system_wrapper.bit

☐ Use FSBL flow for initialization

Initialization File: _ide/psinit/ps7_init.tcl

☑ Reset entire system

☑ Program FPGA

☐ Skip Revision Check

☑ Run ps7_init

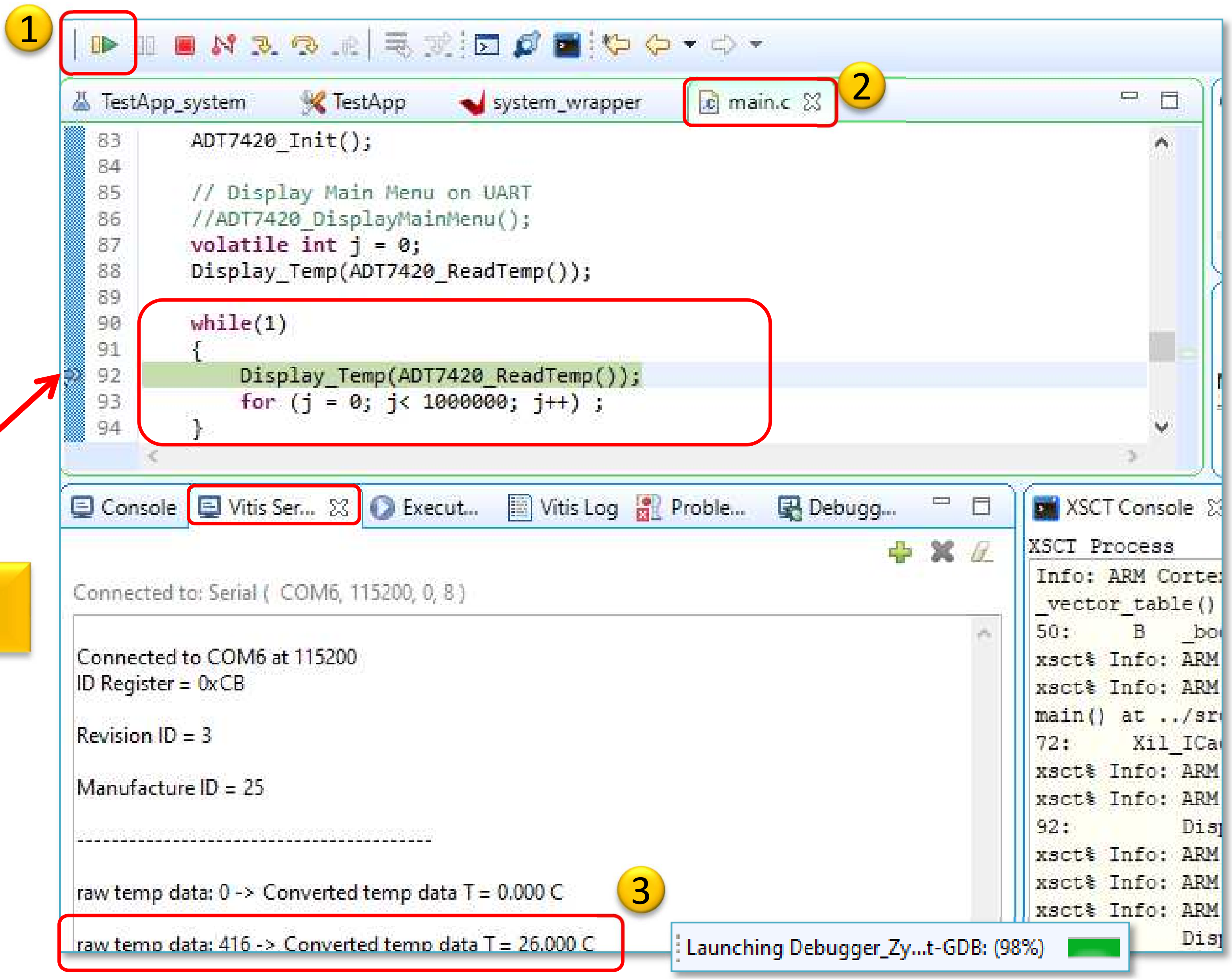
☑ Run ps7_post_config

☐ Enable Cross-Triggering

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)
```

Launching Debugger



```
    ADT7420_Init();

    // Display Main Menu on UART
    //ADT7420_DisplayMainMenu();
    volatile int j = 0;
    Display_Temp(ADT7420_ReadTemp());

    while(1)
    {
        Display_Temp(ADT7420_ReadTemp());
        for (j = 0; j< 1000000; j++) ;
    }
```

Connected to: Serial (COM6, 115200, 0, 8)

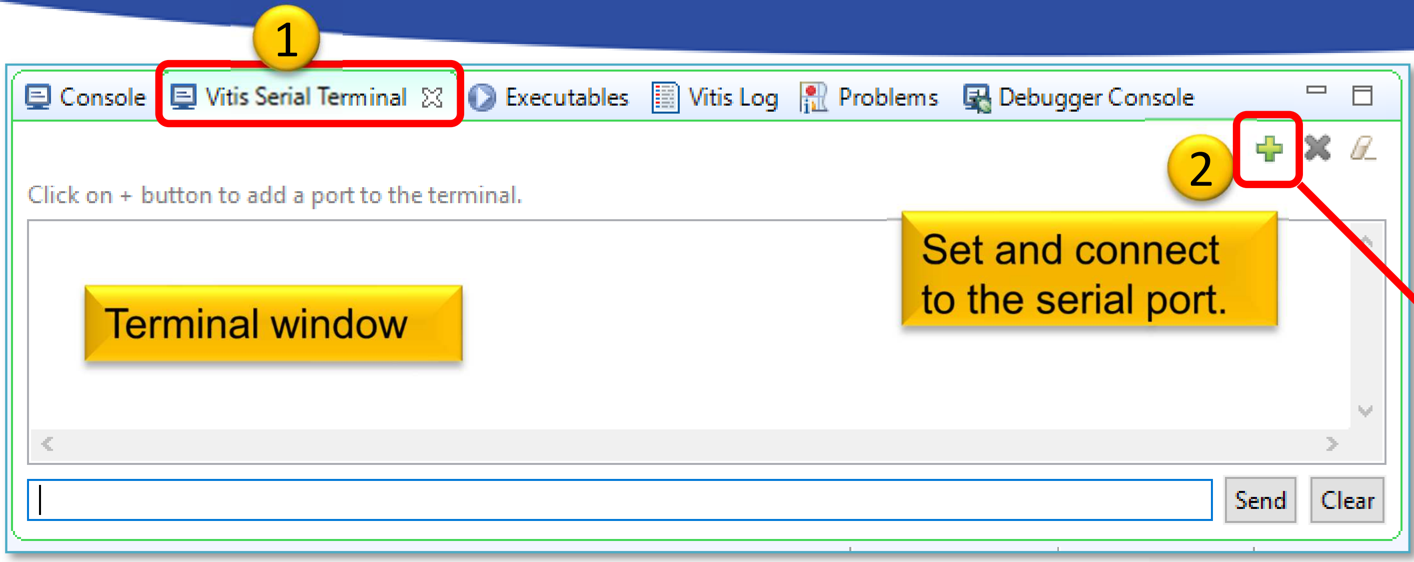
```
Connected to COM6 at 115200
ID Register = 0xCB

Revision ID = 3

Manufacture ID = 25

------------------------------------------

raw temp data: 0 -> Converted temp data T = 0.000 C

raw temp data: 416 -> Converted temp data T = 26.000 C
```

```
XSCT Process
Info: ARM Corte
_vector_table()
50:      B    _bo
xsct% Info: ARM
xsct% Info: ARM
main() at ../sr
72:      Xil_ICa
xsct% Info: ARM
xsct% Info: ARM
92:           Dis
xsct% Info: ARM
xsct% Info: ARM
xsct% Info: ARM
              Dis
```

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

Debugger step into this brakepoint

Set Debug-serial port (VITIS terminal)

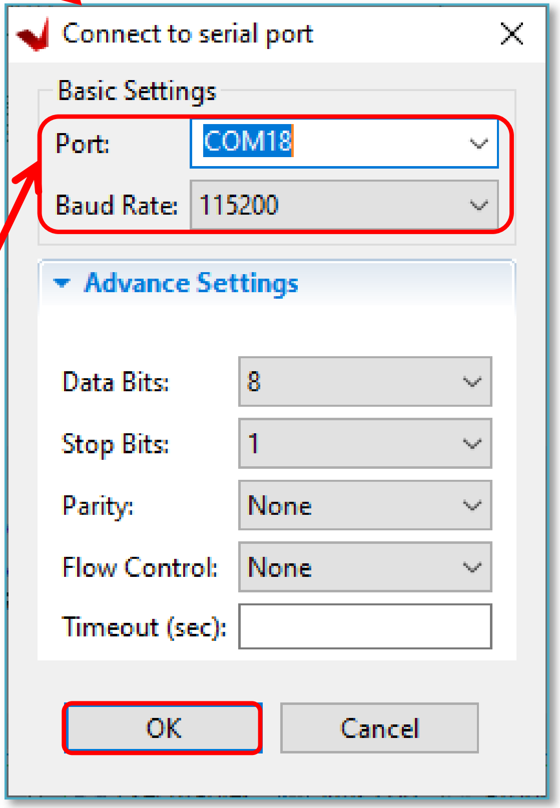


Click on + button to add a port to the terminal.

Terminal window

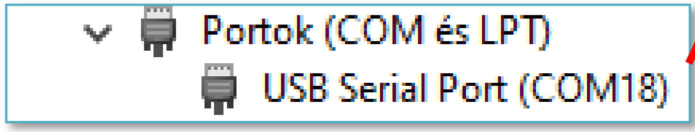
Set and connect to the serial port.

Possible ways to loggin via serial port:
1. **VITIS Serial Terminal: integrated** or
2. using external program: (HyperTerminal, Putty etc.)



Port: COM18
Baud Rate: 115200
Data Bits: 8
Stop Bits: 1
Parity: None
Flow Control: None
Timeout (sec):

- 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)*



Portok (COM és LPT)
USB Serial Port (COM18)

TestApp – Verification result

- Check debug output on VITIS terminal. What did you experience?

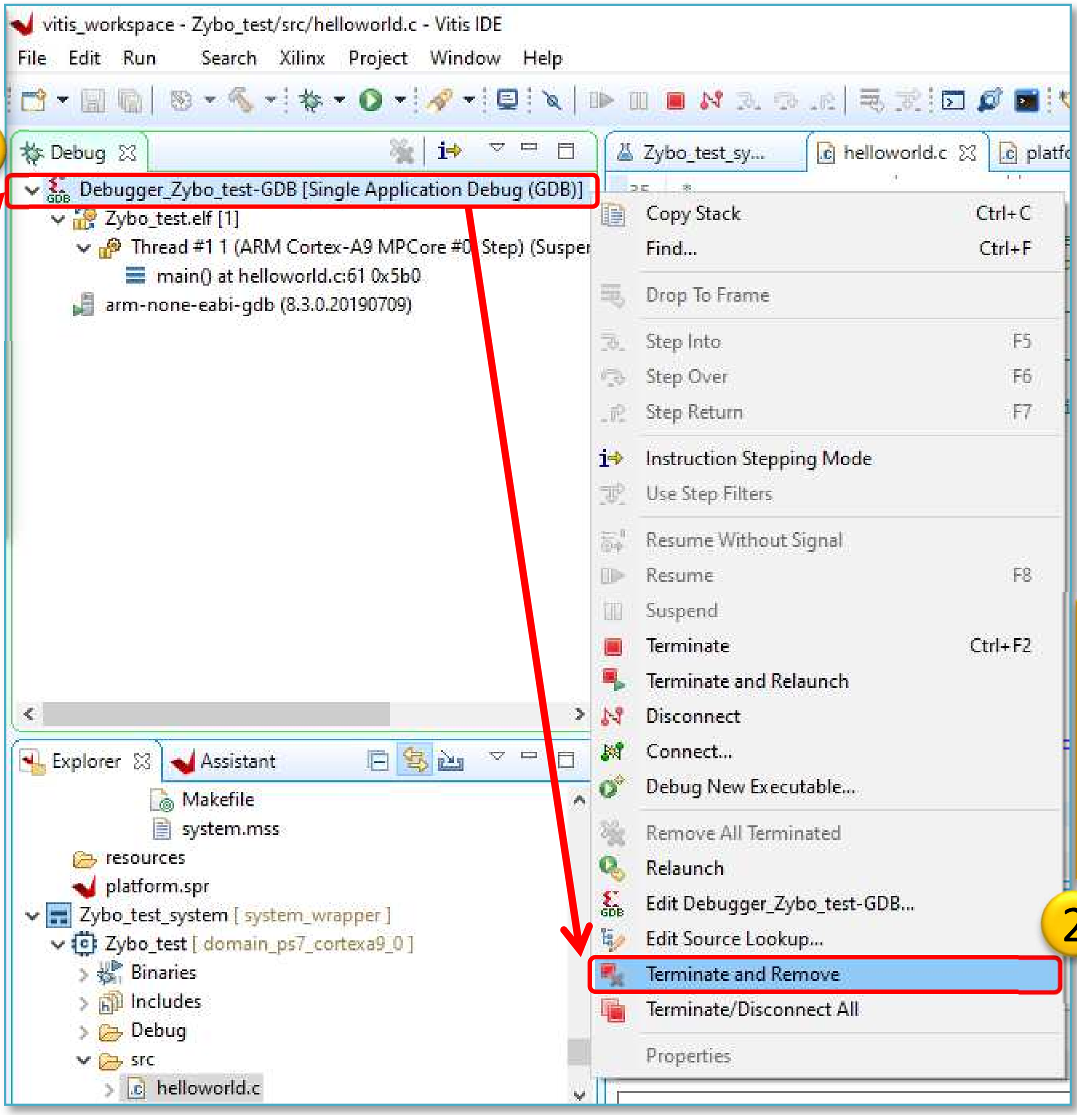
```
ID Register = 0xCB
Revision ID = 3
Manufacture ID = 25

-----------------------------------------

raw temp data: 0 -> Converted temp data T = 0.000 C

raw temp data: 416 -> Converted temp data T = 26.000 C
raw temp data: 416 -> Converted temp data T = 26.000 C
raw temp data: 419 -> Converted temp data T = 26.187 C
raw temp data: 419 -> Converted temp data T = 26.187 C
raw temp data: 419 -> Converted temp data T = 26.187 C
…
```

- Analyze the source code! What is the difference between raw data vs. converted temperature data?

Terminate Debug process

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

Right click on GDB.



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

LAB02_B and C – Summary

- To the ARM-AXI based system created in the previous (5. – LAB02_A), here we added new PL-side **AXI GPIO and AXI IIC** 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.
- **LED displays** (4) and **IIC interfaces** (2) 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 various SW applications (**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