



EFOP-3.4.3-16-2016-00009

A felsőfokú oktatás minőségének és hozzáférhetőségének
együttes javítása a Pannon Egyetemen

FPGA-BASED EMBEDDED SYSTEM DEVELOPMENT (VEMIVIB334BR)



Created by Zsolt Voroshazi, PhD

voroshazi.zsolt@mik.uni-pannon.hu

Updated: 18 Apr. 2024.

SZÉCHENYI 2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Strukturális
és Beruházási Alapok



BEFEKTETÉS A JÖVŐBE

7. VIVADO – EMBEDDED SYSTEM

Creating custom peripherals to BSB #3 (MyLED Peripheral)

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. Creating and adding custom (MyLED) 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.
- The directory path should be no longer than 256 characters!

XILINX VIVADO DESIGN SUITE

Creating custom IP core to the Embedded Base System

SZÉCHENYI 2020



MAGYARORSZÁG
KORMÁNYA

Európai Unió
Európai Strukturális
és Beruházási Alapok



BEFEKTETÉS A JÖVŐBE

Task

- Vivado – Block Designer
 - Create and add a **custom MyLED IP peripheral** to the block design (Embedded Base System) not in the IP Catalog,
 - Parameterize IP blocks, set connections, interfaces, address, and external ports (if needed),
- VITIS - SDK
 - Create SW driver
 - Customize **compiler** settings,
 - Creating a software application: LEDWrite ()

Main steps to solve the task

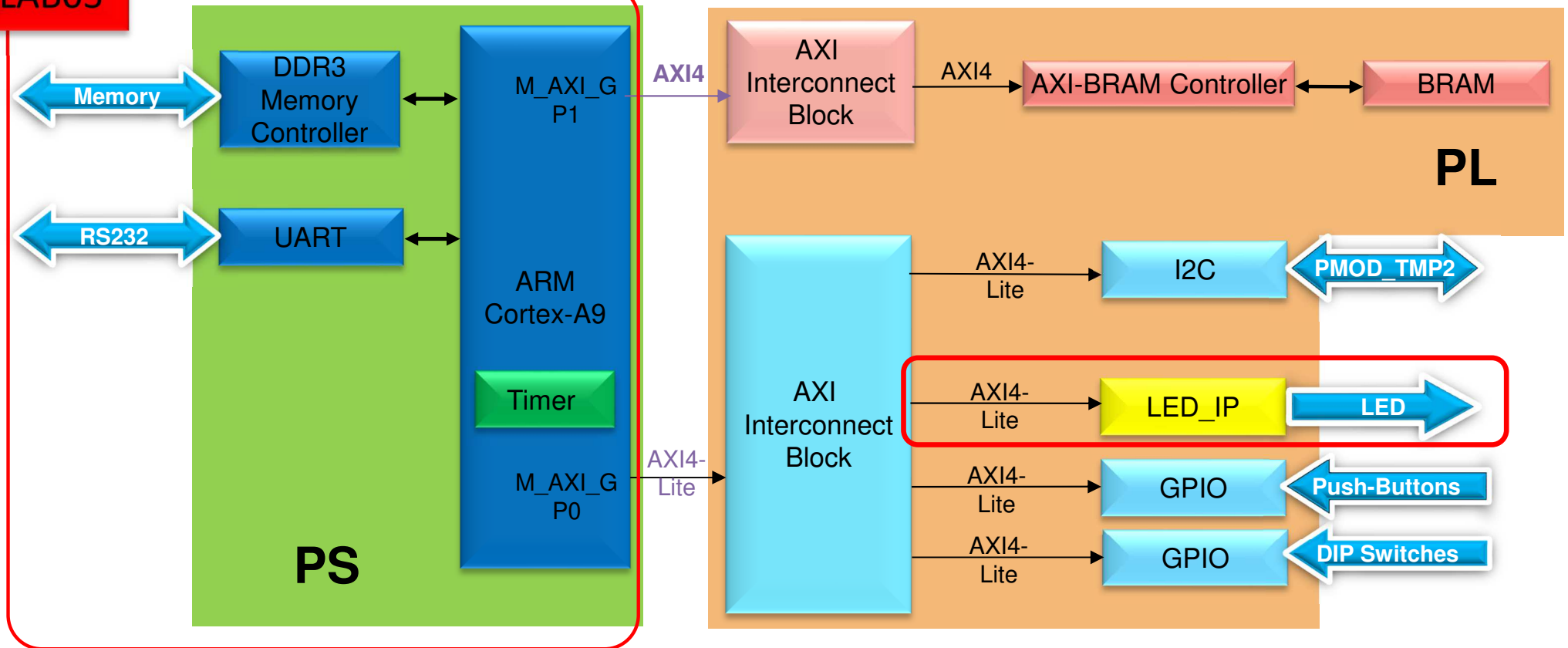
- Create a new project based on previous lab (**LAB02_A**) by using the **Xilinx Vivado (IPI)** embedded system designer,
 - LAB02_A project → Save as... → LAB03 !
- Create and generate custom IP Peripheral in Package IP Wizard,
- Select and add custom IP Peripheral to the base system,
- Parameterize 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 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**.

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... → LAB03**
 - (This will save the former project LAB02_A as LAB03)

Test system to be implemented

LAB03



PS side:

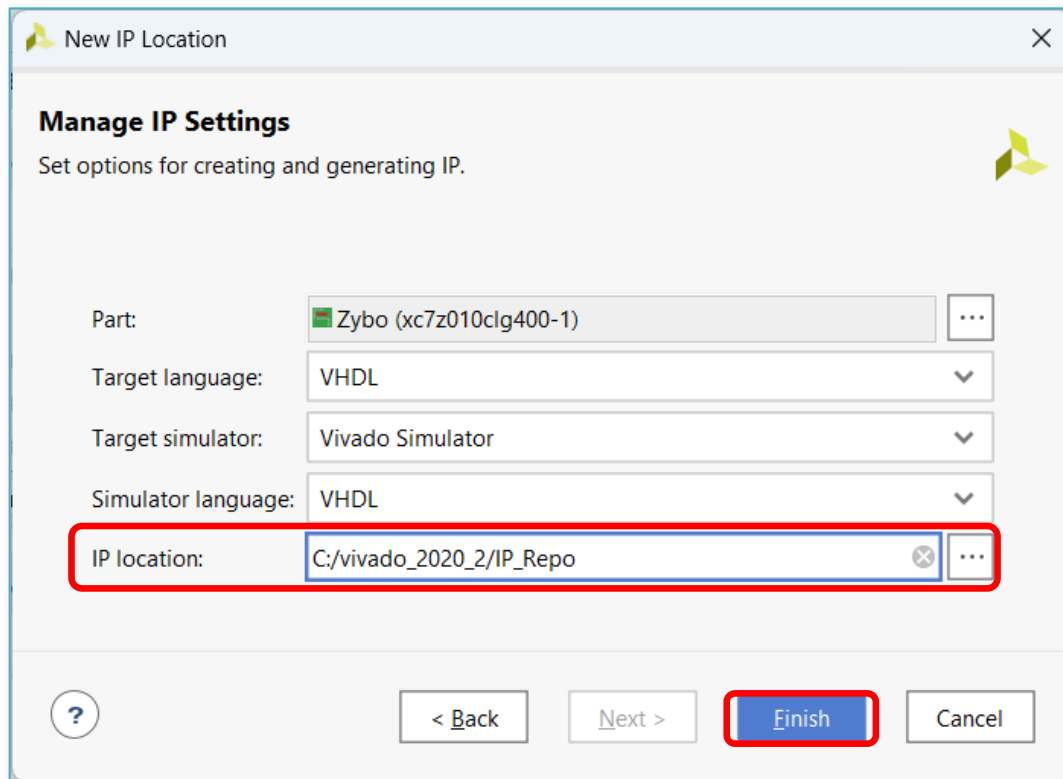
- ARM hard-processor (Core0)
- Internal OnChip-RAM controller
- UART1 (serial) interface
- External DDR3 memory controller

PL side (in FPGA logic)

- LAB03: custom MyLED IP

Add IP path

- File → IP → New Location... → Next

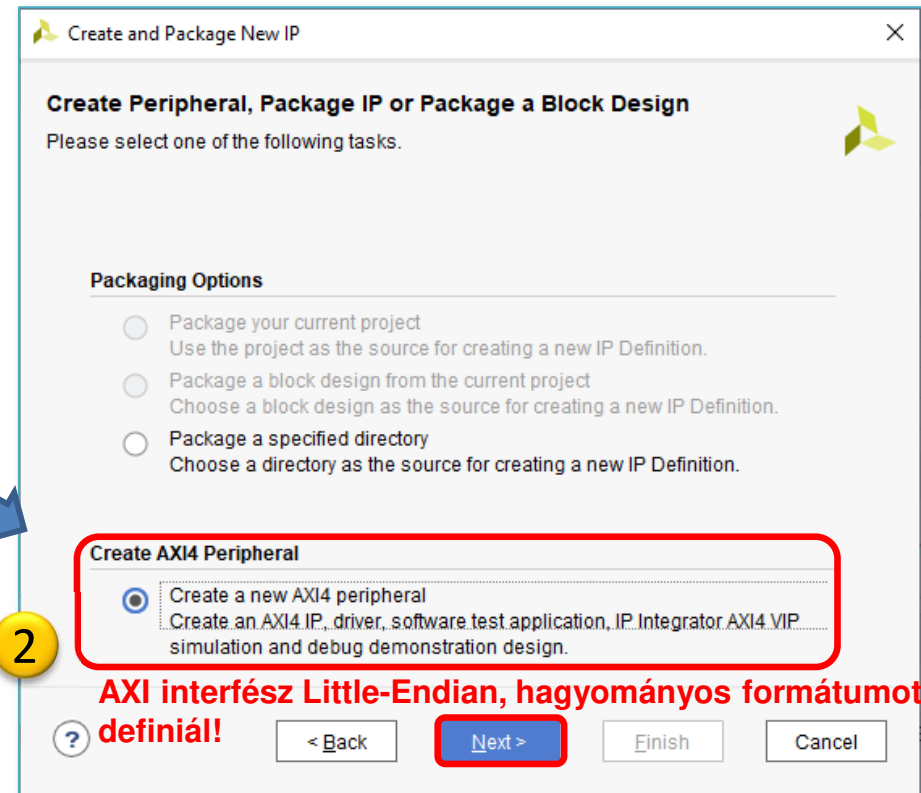
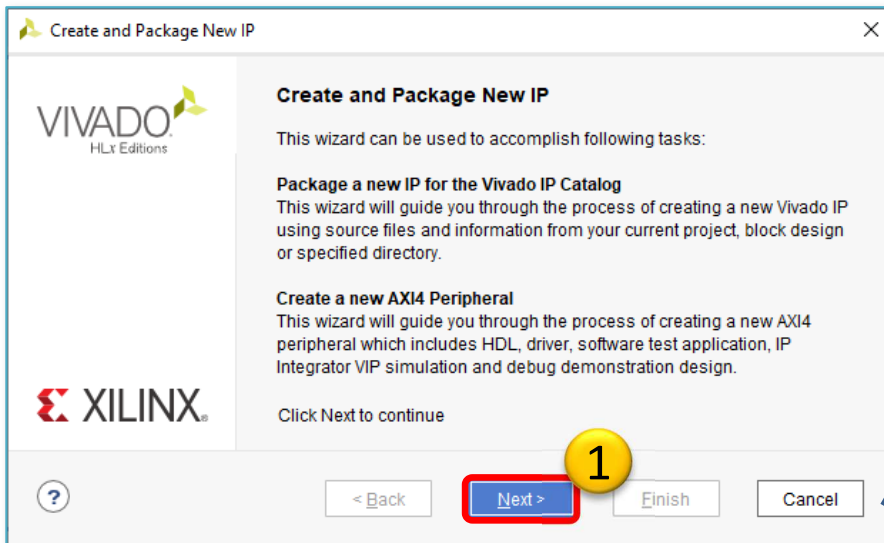


A new IP can be located at:
a.) locally into the actual project directory
or
b.) globally into the Vivado's **IP Catalog** (~global **repository**)
We want to use this latter now
add **\IP_Repo** at the end of path (where our previous projects located).

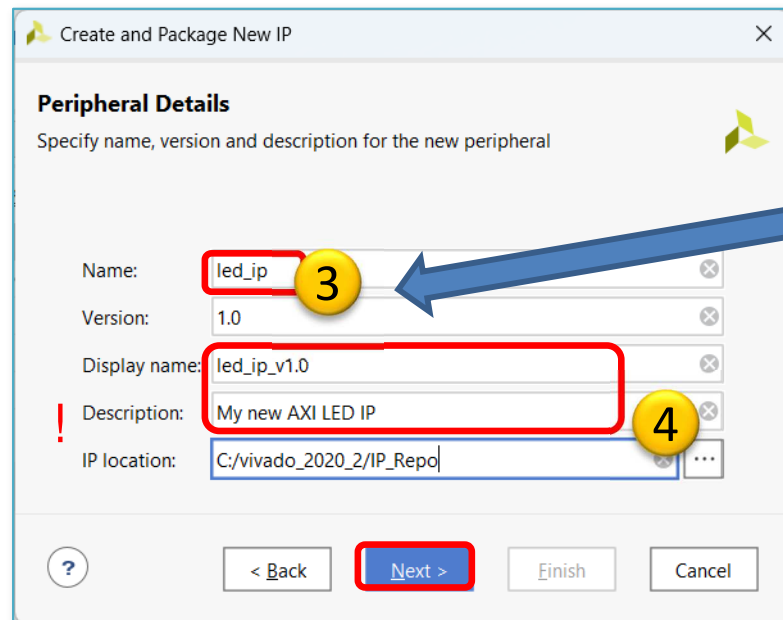
\IP_Repo\managed_ip_project
subdirectory created with an **.xpr** project file.

IP Wizard – LED IP peripheral (I.)

- Tools → Create and Package New IP... → Next



AXI interfész Little-Endian, hagyományos formátumot definiál!

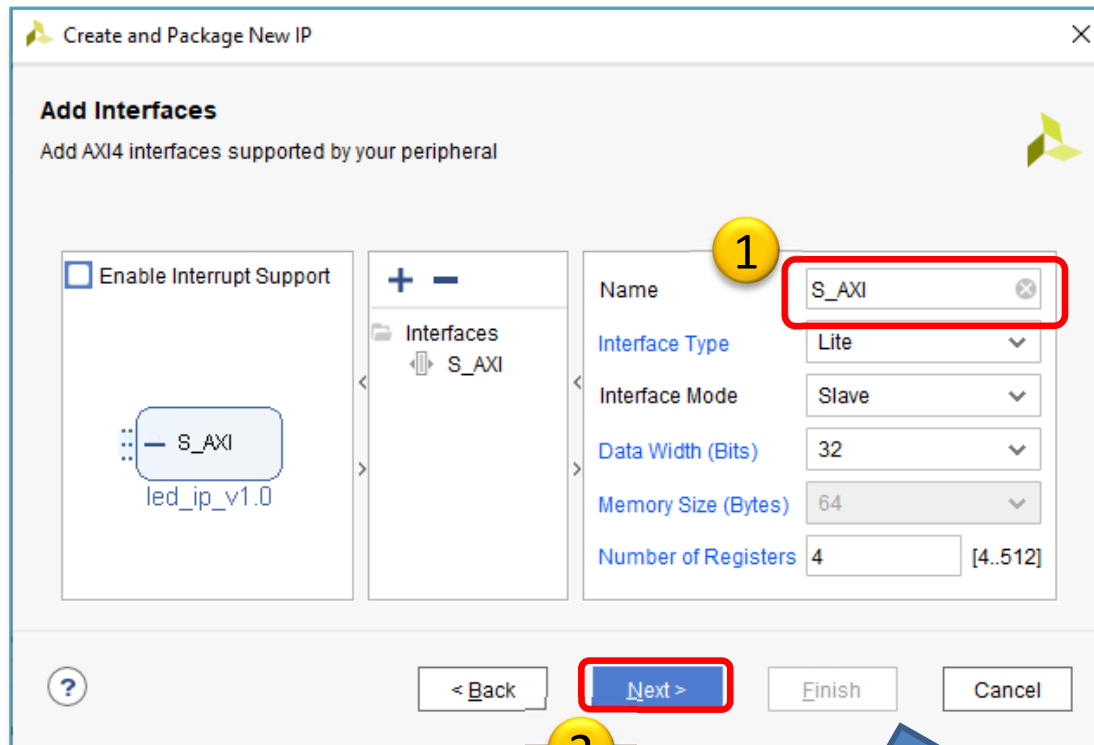


New IP name: „**led_ip**” *
Version: 1.00.a (default)
Description: as you want

led_ip -t tegyük be a
ip_repo -alá. NEXT >>

* IP neve: csak kisbetűs legyen!

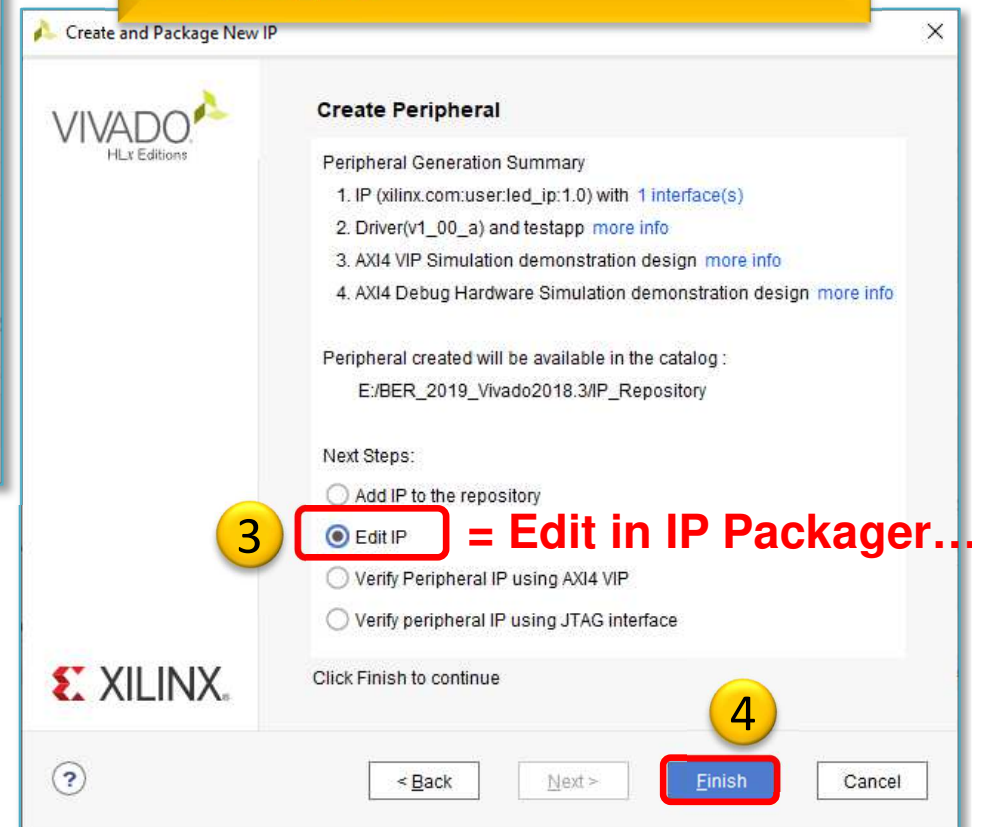
IP Wizard – LED IP peripheral (II.)



- Interrupt not enabled
- **S_AXI** (and not S00_AXI !)
- Lite (AXI Lite if.)
- Slave mode

HDL:

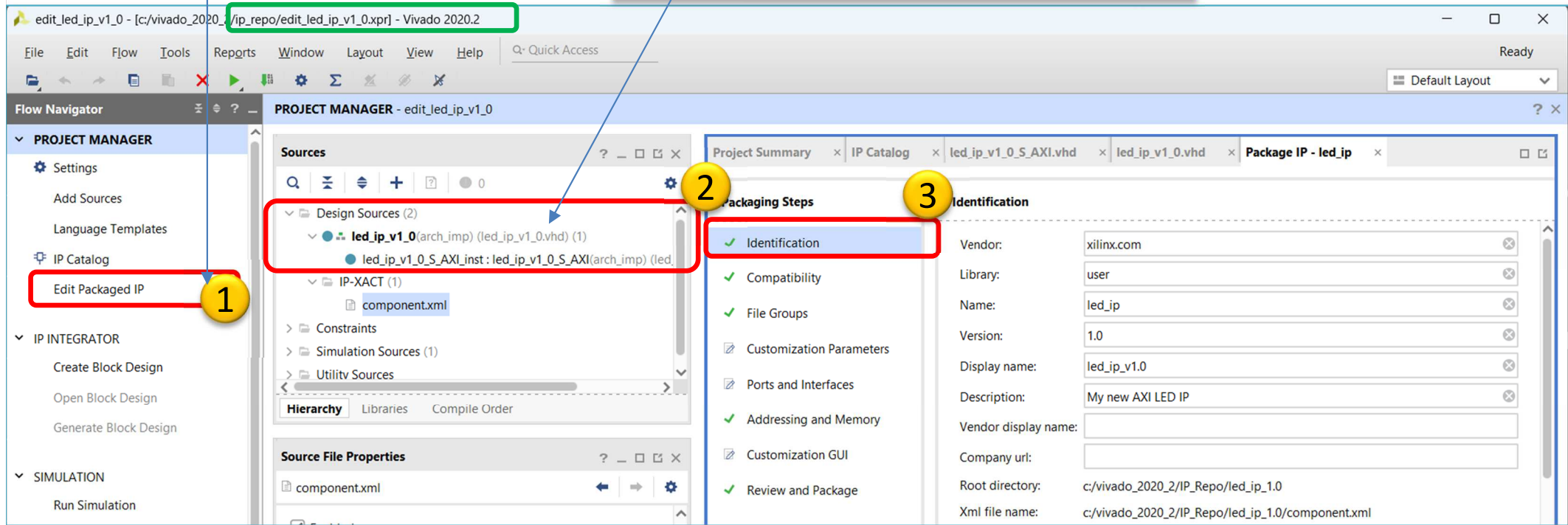
- DATA_WIDTH
- MEMORY_SIZE
- NUM_REG



Project Manager – Package IP template

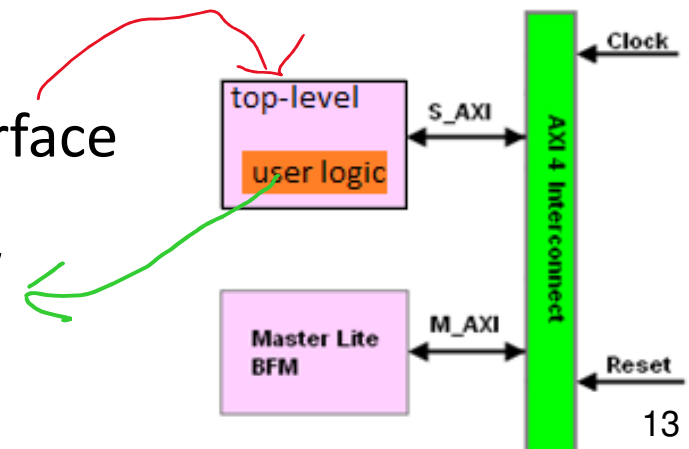
New menu option: Edit Packaged IP

Open the top-level HDL :
led_ip_v1_0.vhd



Hierarchy (design sources):

- led_ip_v1_0.vhd (top-level wrapper = „ interface logic” template)
 - led_ip_v1_0_S_AXI.vhd (user-logic = „R/W register template”)



Generated components of IP peripheral

XPR: the generated IP peripheral can be opened as a separate Vivado project (**edit_led_ip_v1_0.xpr**)

IP-XACT: component.xml descriptor

- **HDL source** -

- `<ip_proj_dir>/ip_repository/ ip_repo/led_ip_1.0/hdl`
 - top entity : led_ip_v1_0.vhd
 - user logic : led_ip_v1_0_S_AXI.vhd

- **BD – Block Diagram** -

- `<ip_proj_dir>/ip_repository / ip_repo/ led_ip_1.0/ bd`
 - bd.tcl

- **XGUI**

- `<ip_proj_dir>/ip_repository / ip_repo/ led_ip_1.0/ xgui`

- **Example design**

- `<ip_proj_dir>/ip_repository / ip_repo/ led_ip_1.0/ example designs`
 - /Bfm_design : Bus Functional Simulation sources
 - debug_hw_design :

- **Driver**

- `<ip_proj_dir>/ip_repository / ip_repo/ led_ip_1.0/ drivers / led_ip_v1_0/src`
 - makefile : Makefile
 - header : led_ip.h
 - source : led_ip.c
 - selftest : led_ip_selftest.c

- **Driver interface**

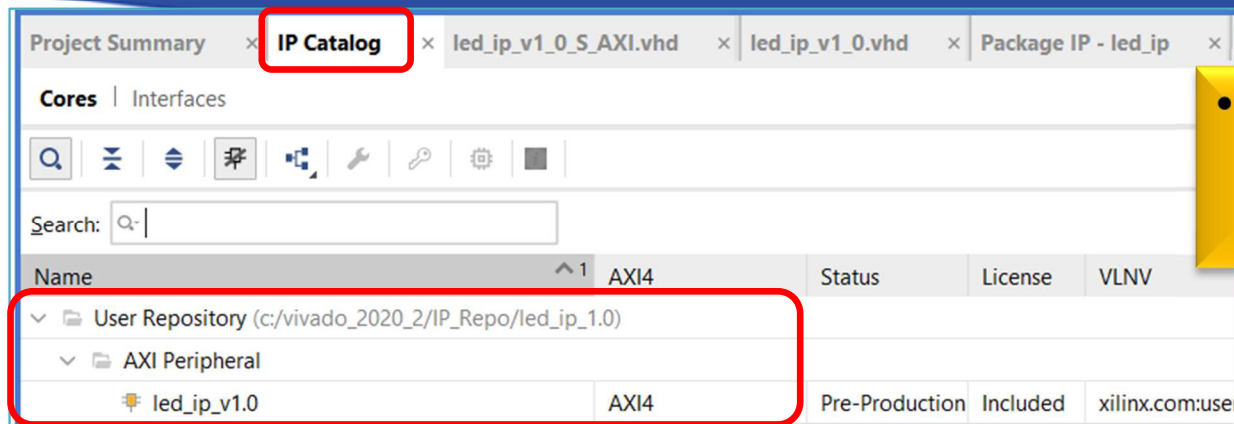
- `<ip_proj_dir>/ip_repository / ip_repo/ led_ip_1.0/ drivers / led_ip_v1_0/ data`
 - mdd : led_ip.mdd
 - tcl : led_ip_v2_1_0.tcl

led_ip_v1_0

FW sources

SW source,
drivers

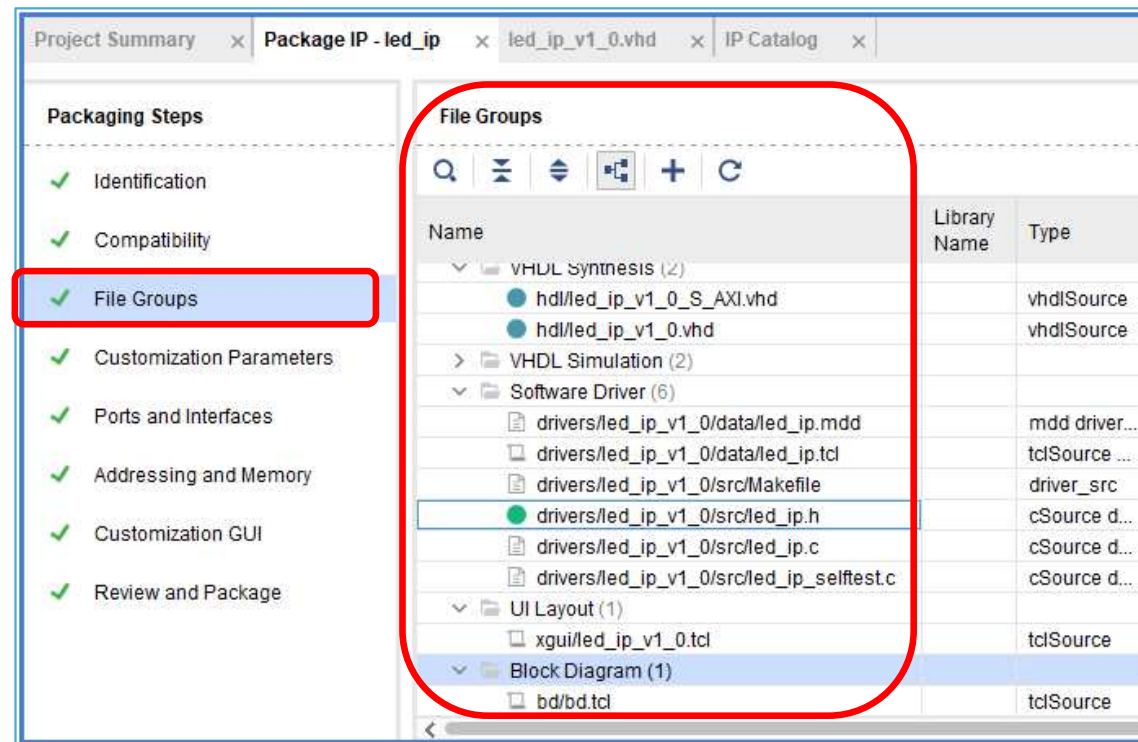
Generate IP peripheral – IP Catalog



- Check! Has your own **LED_IP** peripheral been created in your project?

NOTE: IP-XACT is a standard **xml-based descriptor** (component.xml) that contains definitions, macros, descriptors of custom, reusable, pluggable IPs that can be integrated into an electronic circuit system - in our case an embedded system.


- LED_IP** file/directory structure



Modify peripheral template I. - HDLs

- Open the „top-level“ **led_ip_v1_0.vhd**

- Add the following lines to the file:

Finally (CTRL+S or Save) 

```
5 entity led_ip_v1_0 is
6   generic (
7     -- Users to add parameters here
8     LED_WIDTH : integer := 4;
9     -- User parameters ends
```

1

```
17  port (
18    -- Users to add ports here
19    LED : out std_logic_vector(LED_WIDTH-1 downto 0);
20    -- User ports ends
21    -- Do not modify the ports beyond this line
```

2

```
51  -- component declaration
52  component led_ip_v1_0_S_AXI is
53    generic (
54      LED_WIDTH      : integer := 4;
55      C_S_AXI_DATA_WIDTH : integer := 32;
56      C_S_AXI_ADDR_WIDTH : integer := 4
```

3

```
58  port (
59    LED      : out std_logic_vector(LED_WIDTH-1 downto 0);
60    S_AXI_ACLK : in std_logic;
```

4

```
86 -- Instantiation of Axi Bus Interface S_AXI
87 led_ip_v1_0_S_AXI_inst : led_ip_v1_0_S_AXI
88   generic map (
89     LED_WIDTH      => LED_WIDTH,
90     C_S_AXI_DATA_WIDTH => C_S_AXI_DATA_WIDTH,
91     C_S_AXI_ADDR_WIDTH => C_S_AXI_ADDR_WIDTH
92   )
93   port map (
94     LED      => LED,
95     S_AXI_ACLK => s_axi_aclk,
```

5

6

8. line: integer type generic constant – 4 (LED bit width)

19. Line : extend entity's PORT list
Out direction, 4-bit, Little Endian
std_logic_vector type, name: LED

54. line : add an integer type generic constant to a component (user logic)

59. line : add a LED port to a component (user logic)

89. line: map LED_WIDTH generic to a user_logic

94. line: map LED port to user_logic

Modify peripheral template II. - HDLs

- Open „sub-level” **led_ip_v1_0_S_AXI.vhd-t** (in the „user-logic”)
- Add the following lines to the file :

```
5 entity led_ip_v1_0_S_AXI is
6   generic (
7     -- Users to add parameters here
8     LED_WIDTH : integer := 4;
9     -- User parameters ends
```

1

```
17  port (
18    -- Users to add ports here
19    LED : out std_logic_vector(LED_WIDTH-1 downto 0);
20    -- User ports ends
21    -- Do not modify the ports beyond this line
```

2


8. line: integer type generic constant – 4 (bit width of the LED)

19. line: Expansion of entity's PORT
Out direction, 4-bit, Little Endian
std_logic_vector type, name: LED

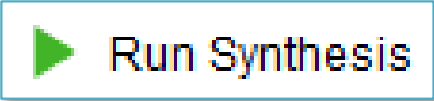
```
380
381 -- Add user logic here
382 LED <= slv_reg0(LED_WIDTH-1 downto 0);
383 -- User logic ends
384
385 end arch_imp;
```

3

388. line: own VHDL code source here:
mapping slv_reg0 lower 4 bits
to the LED (Little endian AXI Lite)!

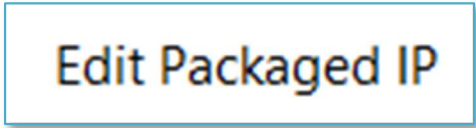
Finally (CTRL+S or Save) 

Synthesis – Package IP

- Flow Navigator menu → **Run Synthesis** (*Save before!)
 - Open Synthesized IP peripheral design, OK 
 - Warning messages are allowed (the design can be implemented),
 - (Here you can simulate the behaviour of your IP periphery).



Project Manager → Edit Package IP:



- Open **led_ip**

Package IP – Customization Parameters

Package x Device x Package IP - led_ip x led_ip_v1_0_S_AXI.vhd x Schematic x Project Summary x

Packaging Steps

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

Customization Parameters

2 Merge changes from Customization Parameters Wizard

Q [] + C

Name	Description	Display Name	Value
Customization Parameters			
✚ C_S_AXI_DATA_WIDTH	Width of S_AXI data bus	C S AXI DATA WIDTH	32
✚ C_S_AXI_ADDR_WIDTH	Width of S_AXI address bus	C S AXI ADDR WIDTH	4
✚ C_S_AXI_BASEADDR		C S AXI BASEADDR	0xFFFFFFFF
✚ C_S_AXI_HIGHADDR		C S AXI HIGHADDR	0x00000000

↓

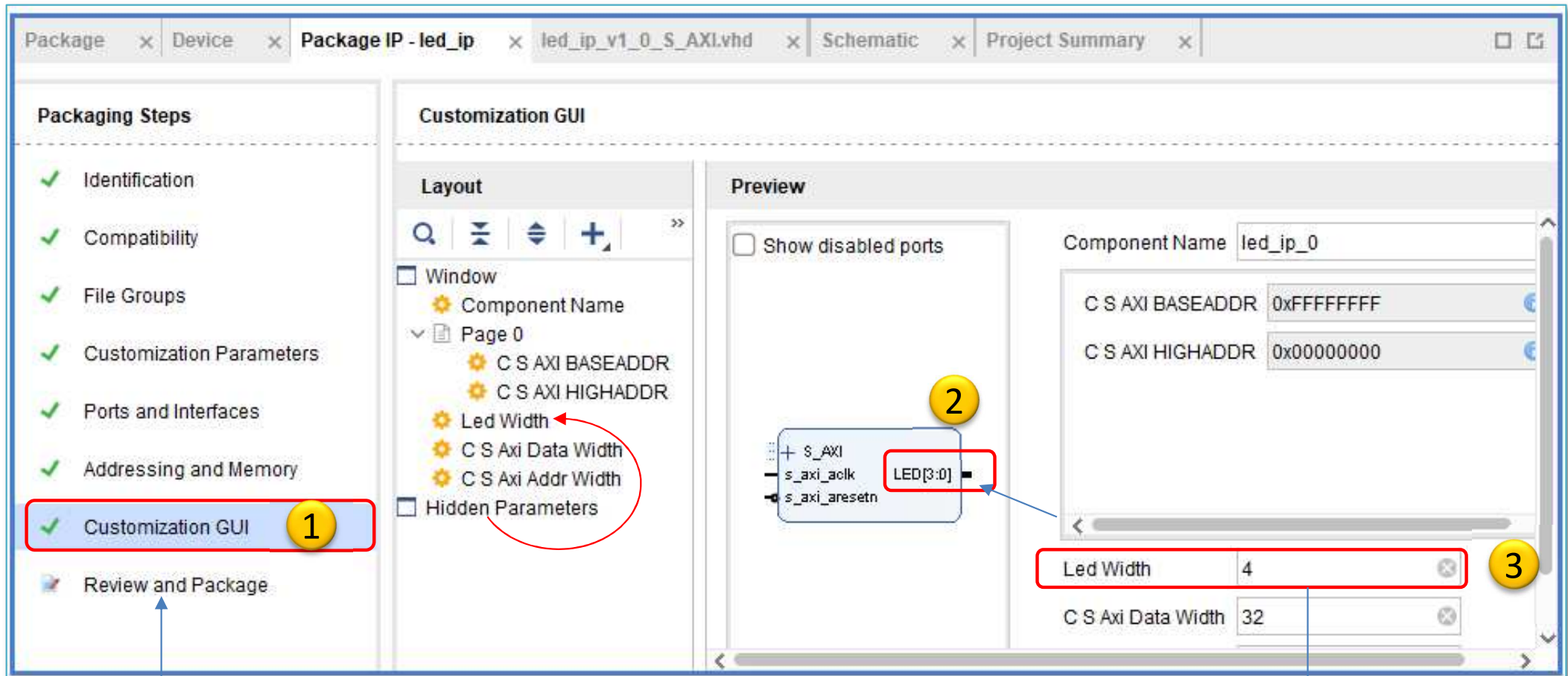
Name	Description	Display Name	Value
Customization Parameters			
✚ C_S_AXI_DATA_WIDTH	Width of S_AXI data bus	C S AXI DATA WIDTH	32
✚ C_S_AXI_ADDR_WIDTH	Width of S_AXI address bus	C S AXI ADDR WIDTH	4
✚ C_S_AXI_BASEADDR		C S AXI BASEADDR	0xFFFFFFFF
✚ C_S_AXI_HIGHADDR		C S AXI HIGHADDR	0x00000000
Hidden Parameters			
✚ LED_WIDTH		LED Width	4

3

Edit Parameter...
Add Parameter...
Remove Parameter
4 Import IP Parameters...
Refresh Table

Import IP parameters: hidden parameter is visible: LED_WIDTH parameter (generic)

Package IP – Customization GUI



4

Finally: Click on „Review and Package”

Package IP – Review and Package

The screenshot shows the 'Review and Package' step of the IP Packaging Wizard. The left sidebar lists the packaging steps: Identification, Compatibility, File Groups, Customization Parameters, Ports and Interfaces, Addressing and Memory, Customization GUI, and Review and Package. The 'Review and Package' step is highlighted with a red box and a yellow circle labeled '1'. The main area displays the 'Summary' and 'After Packaging' sections. The 'Summary' section shows the display name 'led_ip_v1.0', description 'My new AXI LED IP', and root directory 'c:/vivado_2020_2/IP_Repo/led_ip_1.0'. The 'After Packaging' section states that an archive will not be generated and provides a link to 'Edit packaging settings'. A red box and yellow circle labeled '2' highlight the 'Edit packaging settings' link. A red text annotation says 'Remember where the "led_ip" project was generated'. A blue arrow points from the 'Edit packaging settings' link to the 'Re-Package IP' button, which is highlighted with a yellow circle labeled '3'. The 'Re-Package IP' button is located at the bottom right of the wizard. Below the wizard, there is a list of project settings, including 'Project', 'IP Defaults', 'File', 'WebTalk', 'Help', 'Text Editor', '3rd Party Simulators', 'Colors', 'Selection Rules', 'Shortcuts', 'Strategies', and 'Window Behavior'. A yellow circle labeled '4' highlights the 'After Packaging' section of the 'Automatic Behavior' settings, which includes checkboxes for 'Create archive of IP', 'Add IP to the IP Catalog of the current project', 'Close IP Packager window', and 'Include Source project archive'. A yellow circle labeled '5' highlights the 'OK' button at the bottom right of the settings window. A red box highlights the 'OK' button. A red text annotation says '5.) OK. Finally Re-Package IP → YES (IP project will automatically close)'.

Package IP - led_ip

Packaging Steps

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

Review and Package

Summary

Display name: led_ip_v1.0
Description: My new AXI LED IP
Root directory: c:/vivado_2020_2/IP_Repo/led_ip_1.0

After Packaging

An archive will not be generated. Use the settings link below to change your preference
IP will be made available in the catalog using the repository -
c:/vivado_2020_2/IP_Repo/led_ip_1.0
[Edit packaging settings](#)

Re-Package IP

Automatic Behavior

After Packaging

- ☒ Create archive of IP
- ☒ Add IP to the IP Catalog of the current project
- ☒ Close IP Packager window
- ☒ Include Source project archive

Edit IP in IP Packager

- ☒ Delete project after packaging

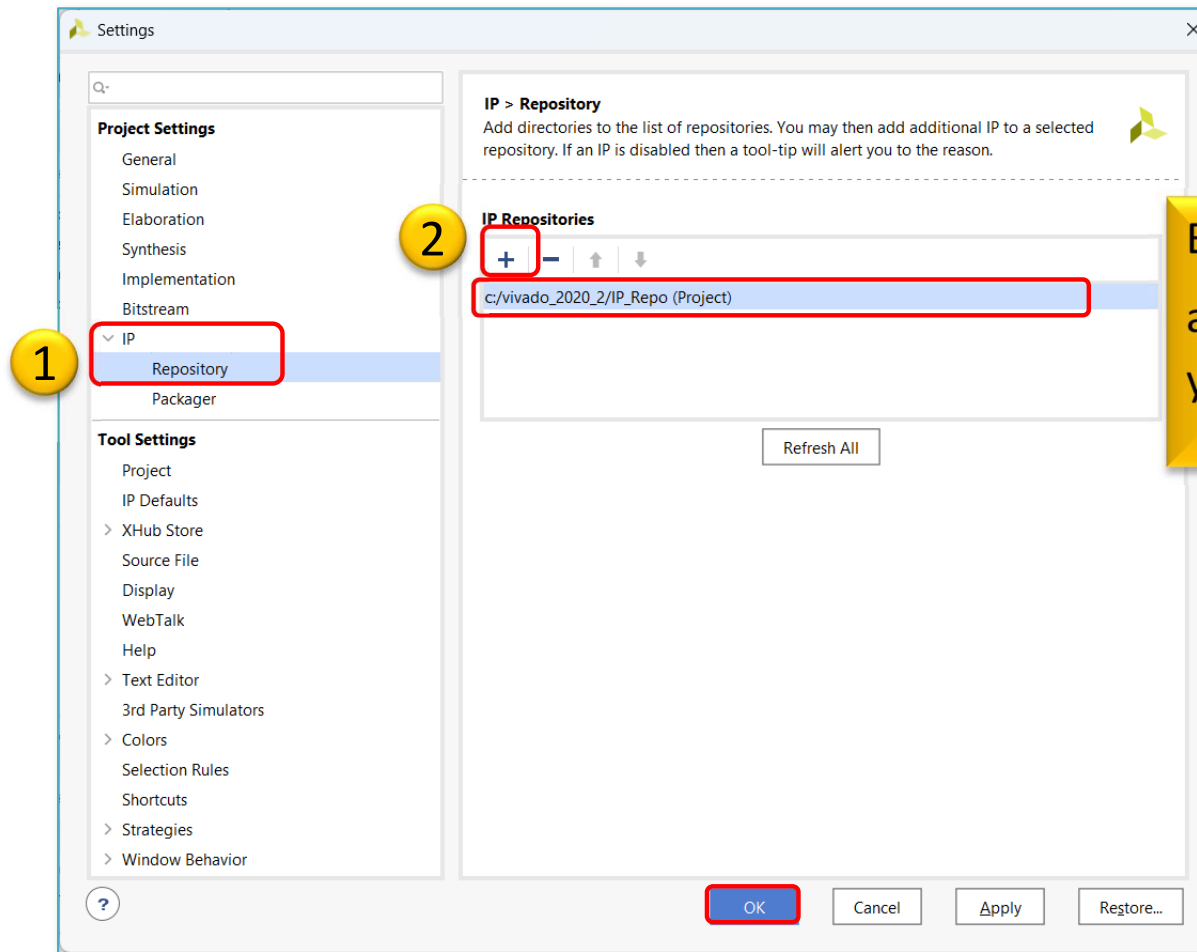
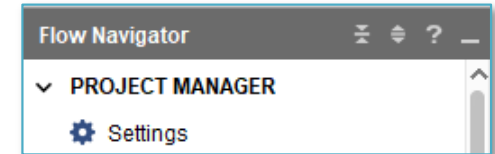
File Extensions to Filter on Add Directory

Create a list of file extensions that will be automatically filtered when adding a directory to a File Group.

5.) OK.
Finally **Re-Package IP** → **YES**
(IP project will automatically close)

Return to LAB03

- Open project → Choose „LAB03”
 - Project Manager → Settings
 - Select IP → + → Add IP path



Browse for IP repository,
and add + „IP_repo” to
your project

Adding and connecting PL side LED_IP to the base system I.

New IP core can be added in Vivado (two options):

- a.) Block Diagram View → Add IP
- b.) Open IP Catalog -> Select IP → Double-click → Add IP to Block Design

Add your own LED_IP peripheral on the PL side to the BSB

The screenshot shows the Vivado IP Catalog window. A yellow box labeled 'Change to IP Catalog view' points to the 'IP Catalog' tab (1). The search bar contains 'led' (3), showing one match. Under the 'AXI Peripheral' folder, 'led_ip_v1.0' is selected (2). A yellow box labeled 'Select LED_IP' points to this entry. The details for 'led_ip_v1.0' are shown below the table. A yellow box labeled 'Add IP (double click, or +)' points to the 'led_ip_v1.0' entry. A dialog box 'Add IP' (4) is open, asking 'Would you like to add 'led_ip_v1.0' IP to your block design, or customize it and add it as an RTL module to your project?'. The 'Add IP to Block Design' button is highlighted.

Change to IP Catalog view

1

2

3

4

Select LED_IP

Add IP (double click, or +)

Add IP

Would you like to add 'led_ip_v1.0' IP to your block design, or customize it and add it as an RTL module to your project?

Add IP to Block Design

Customize IP

Cancel

Name	AXI4	Status	License	VLNV
User Repository (e:/BER_2019_Vivado2018.3/IP_Repository)				
AXI Peripheral				
led_ip_v1.0	AXI4	Pre-...	Included	xilin...

Details

Version: 1.0 (Rev. 2)

Interfaces: AXI4

Description: Sajat AXI LED IP Periferia

Status: Pre-Production

License: Included

Vendor: Xilinx, Inc.

VLNV: xilinx.com:user:led_ip:1.0

Repository: e:/BER_2019_Vivado2018.3/IP_Repository

Adding and connecting PL side LED_IP to the base system II.

Now, for your own IP module (LED_IP) you need to configure the following in Vivado (can be manual / automatic!):

- a.) **interface connection** between IP module and bus system (AXI),
- b.) assignment of the IP module to an **address** range (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) - IO planning.

Double-click on **led_ip_0** and examine its parameters.



Parameterising of LED_IP

The screenshot shows the 'Re-customize IP' window for the 'led_ip_v1.0 (1.0)' component. On the left, a port diagram shows 'S_AXI' with sub-ports 's_axi_aclk' and 's_axi_aresetn', and a signal 'LED[3:0]' highlighted with a red box. The main area contains configuration fields: 'Component Name' is 'led_ip_0'; 'C S AXI BASEADDR' is '0xFFFFFFFF'; 'C S AXI HIGHADDR' is '0x00000000'; 'Led Width' is '4' (highlighted with a red box); 'C S Axi Data Width' is '32'; and 'C S Axi Addr Width' is '4'. At the bottom are 'OK' and 'Cancel' buttons, with 'OK' highlighted by a red box. A red arrow points from a yellow callout box to the 'Led Width' field.

Parameter	Value
Component Name	led_ip_0
C S AXI BASEADDR	0xFFFFFFFF
C S AXI HIGHADDR	0x00000000
Led Width	4
C S Axi Data Width	32
C S Axi Addr Width	4

Check
LED_WIDTH:=4
(Zybo has 4 LEDs)

Other address values
remain default.

Connect LED_IP

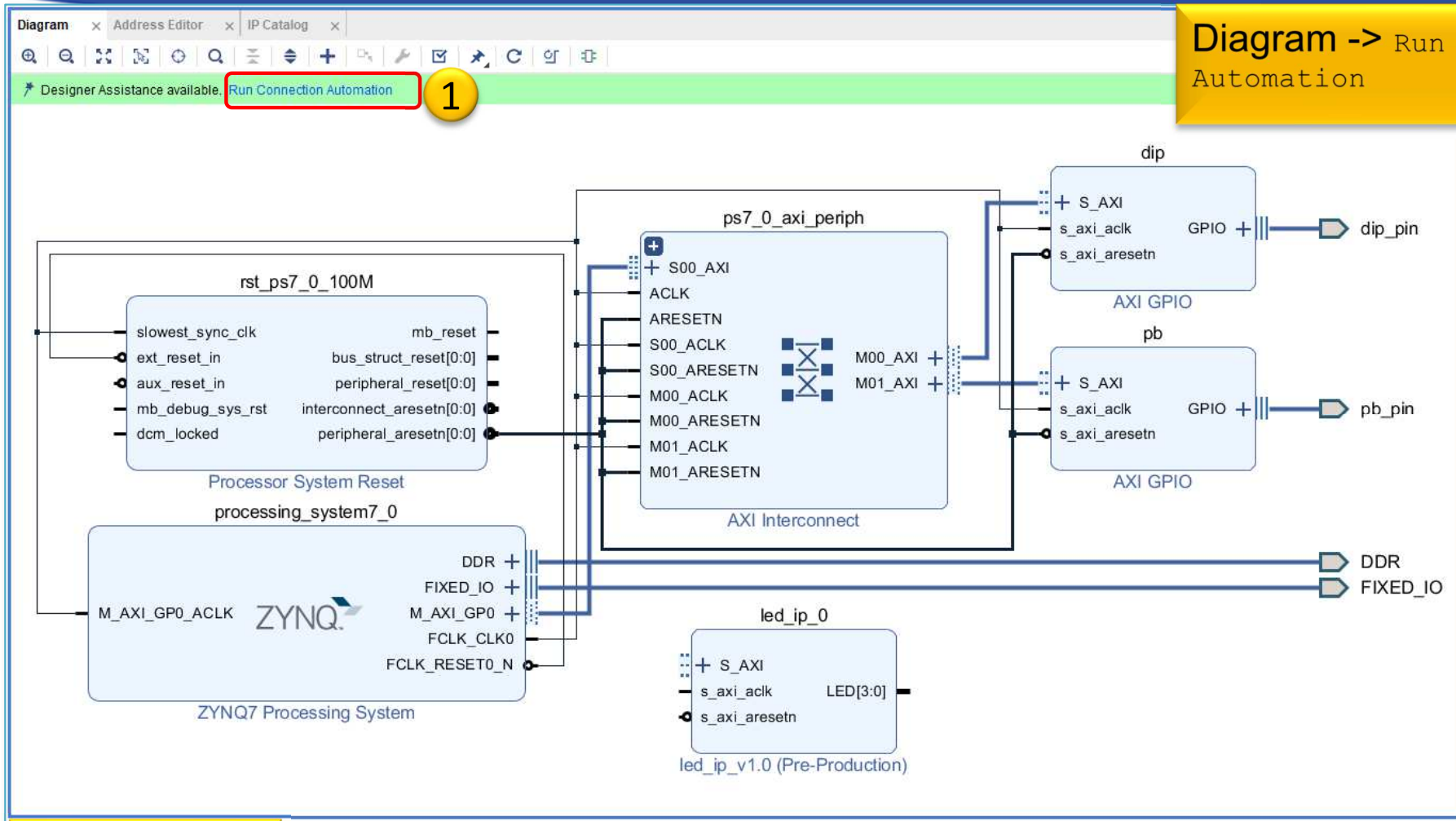
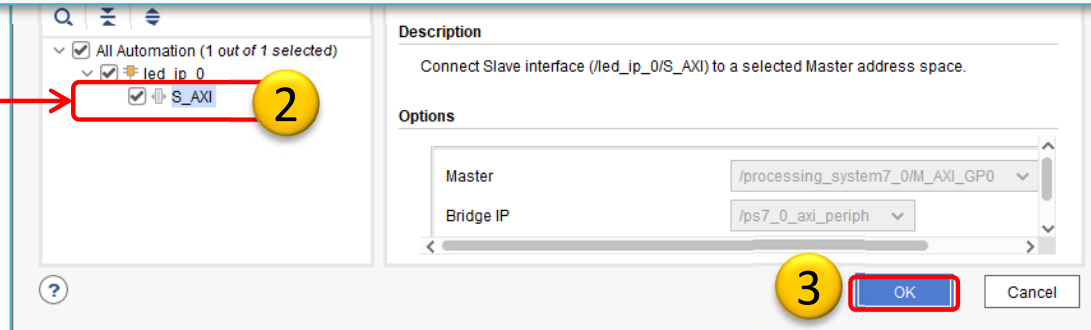
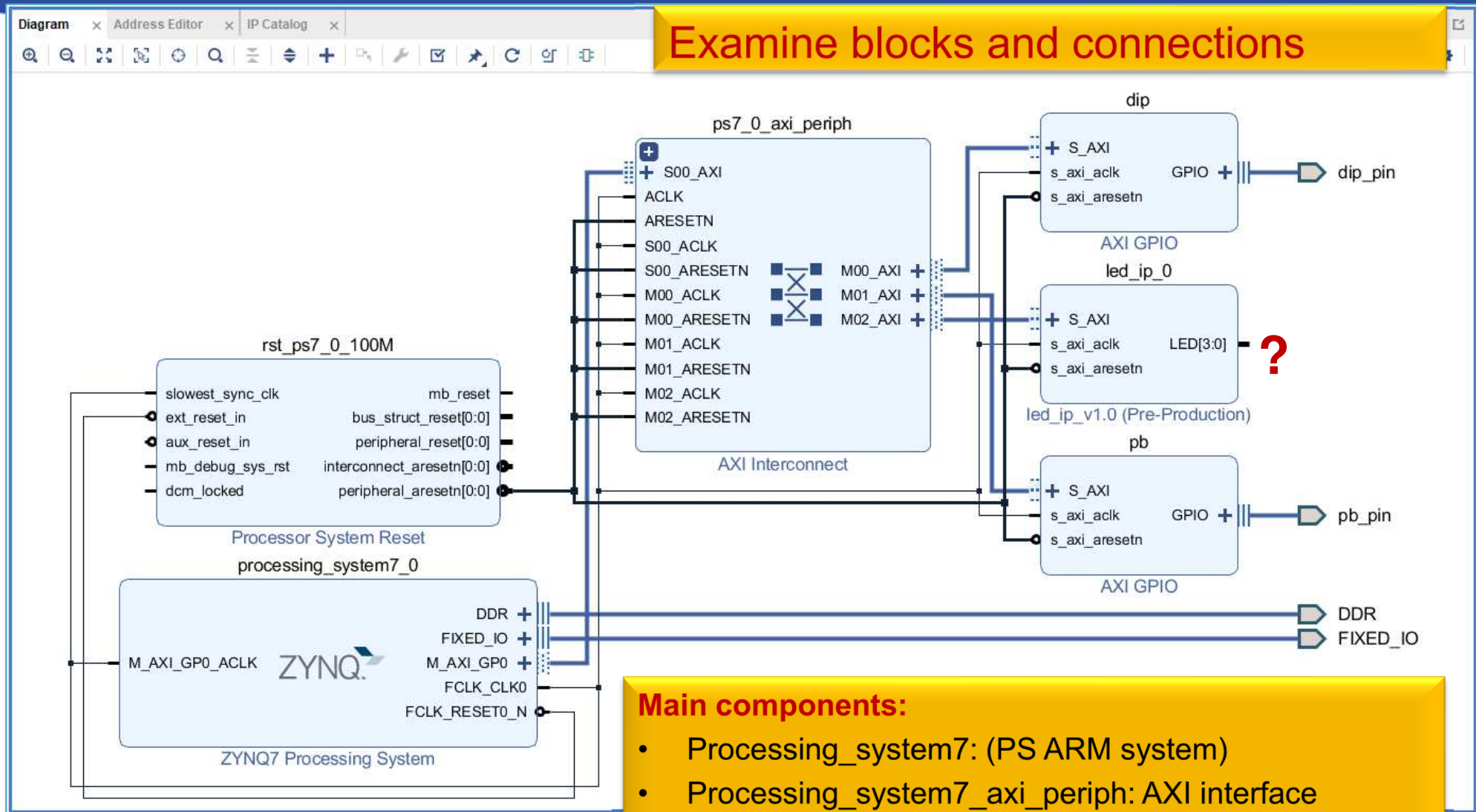


Diagram -> Run Connection Automation

LED_IP
select **S_AXI**
interface for
automatic
routing



Completed block design



Main components:

- Processing_system7: (PS ARM system)
- Processing_system7_axi_periph: AXI interface
- Rst_processing_system7: PS/PL reset generator
- dip: AXI GPIO - DIP switches (PL side)
- pb: AXI GPIO - PB push buttons (PL side)
- led_ip_0: own LED_IP peripheral (PL side)

LED_IP – configure memory address

- Block Design → Select „Address Editor”
- Assign the unmapped IP peripheral into the memory address:
 - a.) automatically – address generation vs. b.) manually (now)

Diagram x Address Editor x Catalog x

0x4000_0000
(Note: GP0 port was set)!

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
led_ip_0	S_AXI	S_AXI_reg	0x4122_0000	64K	0x4122_FFFF

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

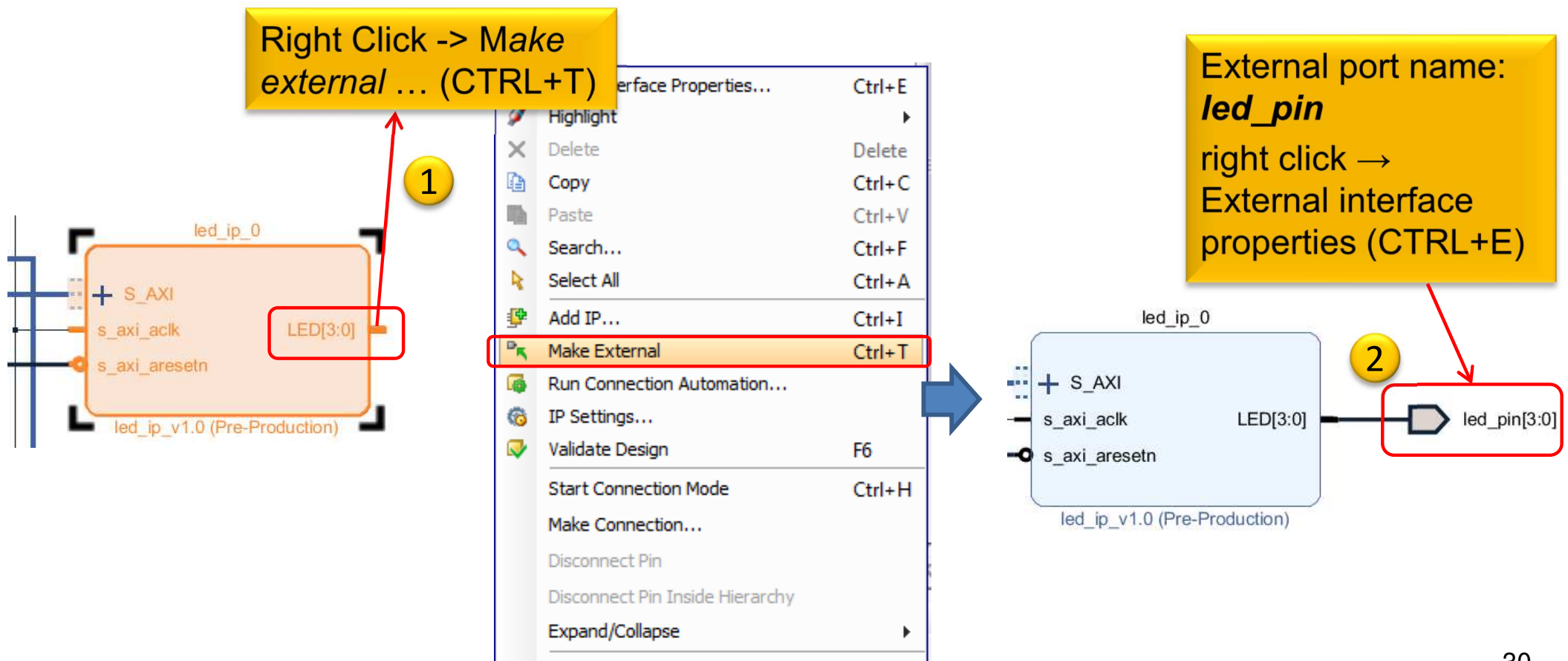
b.) Base address manual set*
Led ip: 0x4122_0000
(64K)

* Address ranges must be aligned into 2^n size and cannot be overlapped!




LED_IP – Assign external ports

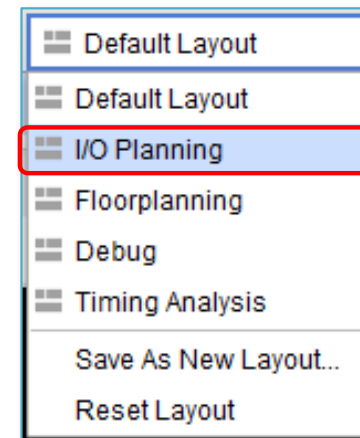
led_ip_0 must be connected to the FPGA pins on the ZyBo card:

- 1.) The data ports of the LED_IP instance must be connected to the external physical FPGA pins,
- 2.) If necessary, define the names of the external ports (e.g. *led_pin*), then
- 3.) In the <system>.XDC file, the pin of the FPGA must be specified.



Block Design – Layout synthesis

- Refresh the Block Design:
 - Regenerate Layout 
 - Validate Design (DRC) 
 - Flow Navigator → Run Synthesis  Run Synthesis
 - Then - **Open Synthesized Design** , OK
- Final step, assign `led_pin<3:0>` to FPGA IO pins!
 - Layout menu -> IO planning layout view

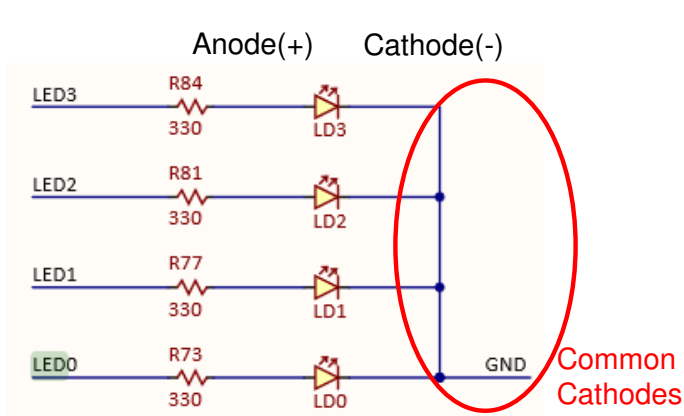


IO planning – pin assignments

We use now I/O planning (GUI) for pin assignments!

		Site		IO Std					OCT	
led_pin (4)	OUT			default (LVCMOS18)		1.800	12	SLOW	NONE	FP_VTT_50
led_pin[3]	OUT			default (LVCMOS18)		1.800	12	SLOW	NONE	FP_VTT_50
led_pin[2]	OUT	?		default (LVCMOS18)	?	1.800	12	SLOW	NONE	FP_VTT_50
led_pin[1]	OUT			default (LVCMOS18)		1.800	12	SLOW	NONE	FP_VTT_50
led_pin[0]	OUT			default (LVCMOS18)		1.800	12	SLOW	NONE	FP_VTT_50
Scalar ports (0)										

Tcl Console Messages Log Reports Design Runs Package Pins I/O Ports



proper pins assignments based on *Zybo_master.xdc*:

- Package Pin:
 - led_pin[0]: M14
 - led_pin[1]: M15
 - led_pin[2]: G14
 - led_pin[3]: D18
- IOSTANDARD: LVCMOS33
- OffChipTermination (OCT): NONE

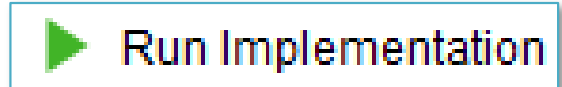
		1 Package Pin		I/O Std					OCT	
led_pin (4)	OUT			35 LVCMOS33*		3.300	12	SLOW	NONE	NONE*
led_pin[0]	OUT	M14		35 LVCMOS33*		3.300	12	SLOW	NONE	NONE*
led_pin[1]	OUT	M15		35 LVCMOS33*		3.300	12	SLOW	NONE	NONE*
led_pin[2]	OUT	G14		35 LVCMOS33*		3.300	12	SLOW	NONE	NONE*
led_pin[3]	OUT	D18		35 LVCMOS33*		3.300	12	SLOW	NONE	NONE*
pb_pin_tri_j (4)	IN			34 LVCMOS33*		3.300			NONE	NONE
Scalar ports (14)										

Tcl Console Messages Log Reports Design Runs Package Pins I/O Ports

File → **Save Constraints** or **CTRL+S**. Then, save the XDC file as: „lab03.xdc”

Implementation and Bitstream generation

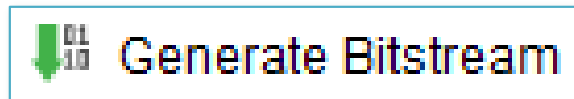
- Flow Navigator menu → **Run Implementation**



- 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**



Implementation reports

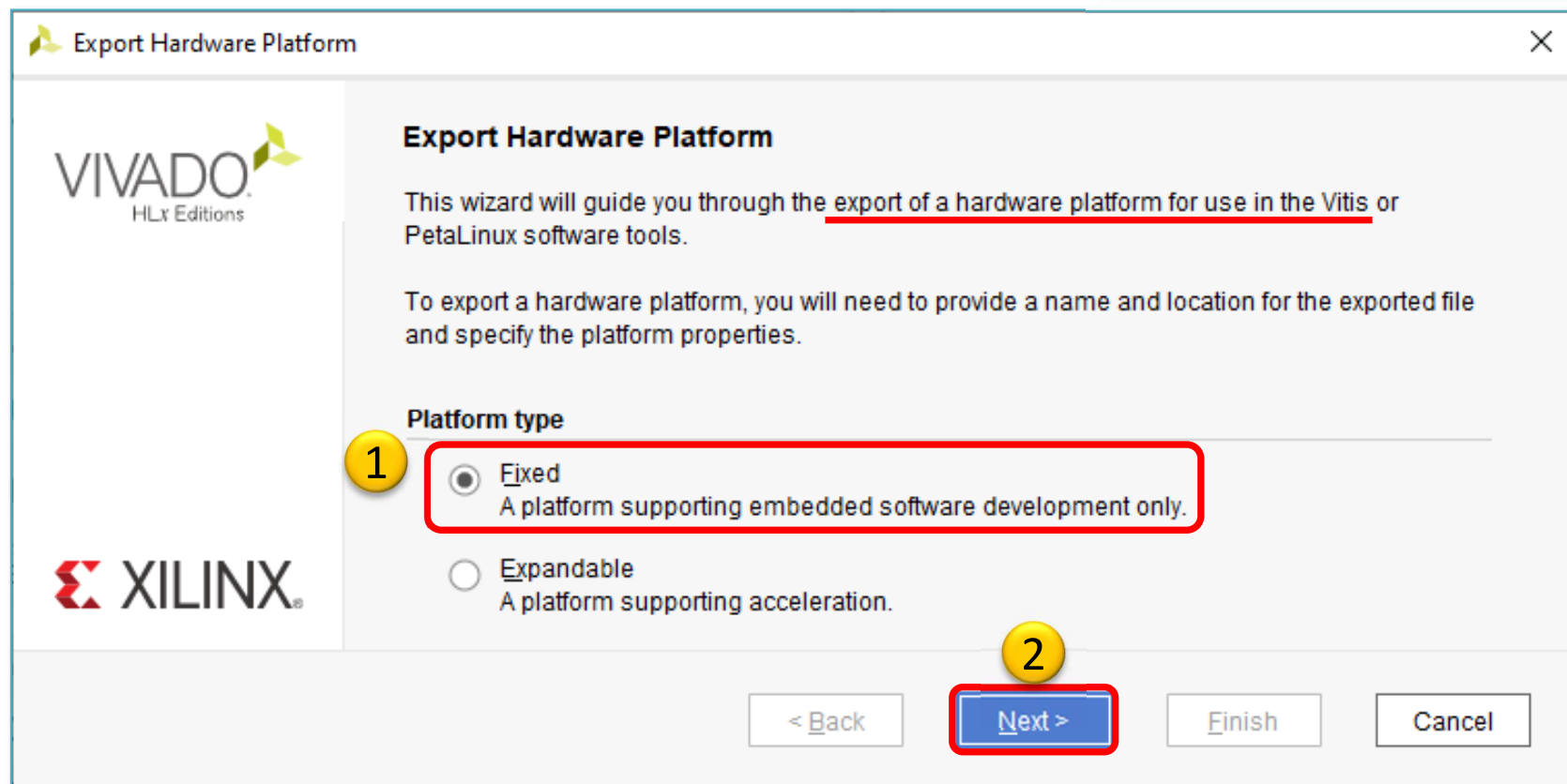
- Question-1.) how many resources are occupied on PL?
- Solution: Reports → Report Utilization (or Project Summary Σ)

Site Type	Used	Fixed	Available	Util%
Slice LUTs	673	0	17600	3.82
LUT as Logic	611	0	17600	3.47
LUT as Memory	62	0	6000	1.03
LUT as Shift Register	62	0		
Slice Registers	994	0	35200	2.82
Register as Flip Flop	994	0	35200	2.82

VIVADO Export HW → VITIS (~SDK)

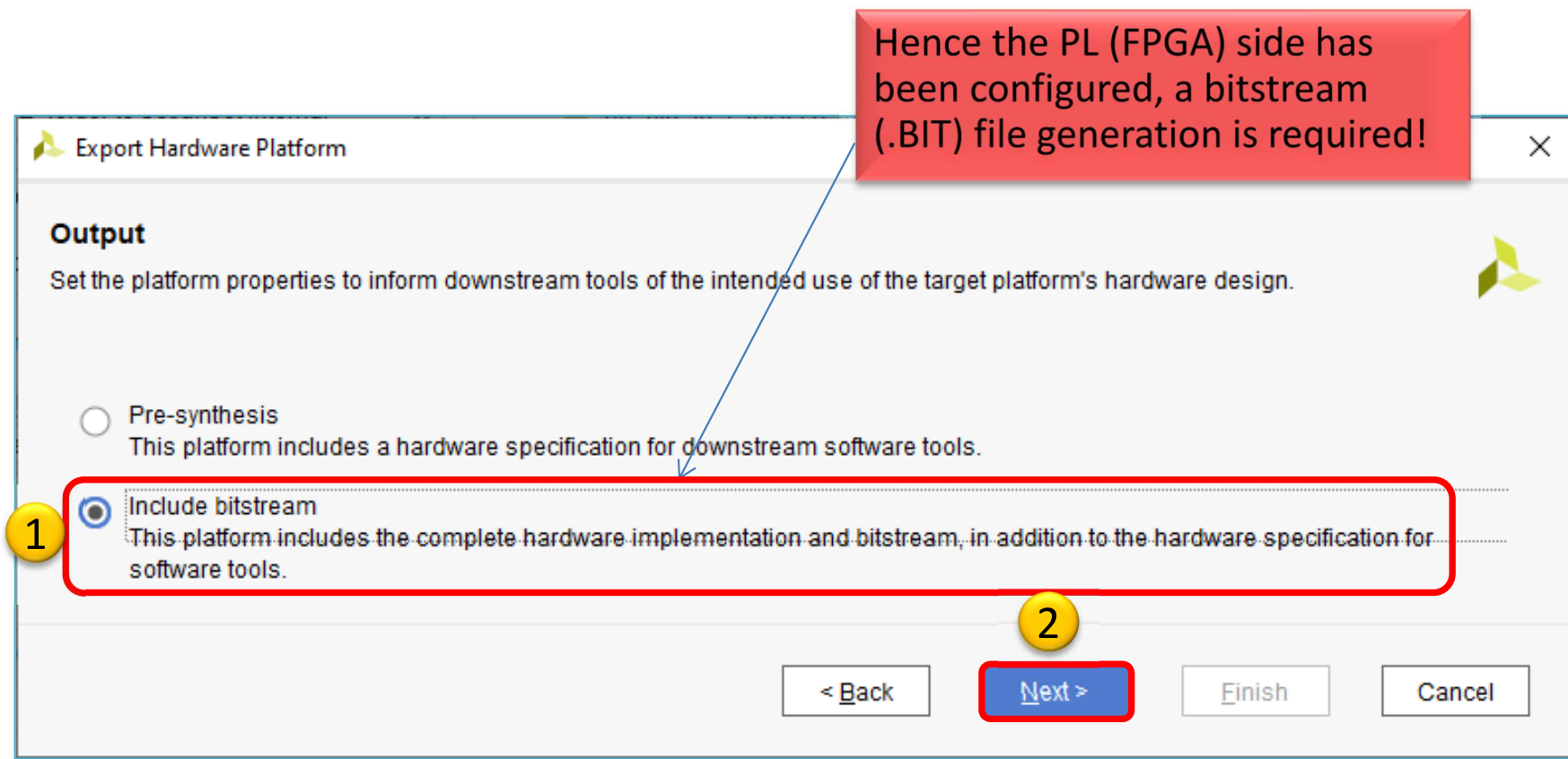
- File → Export → Export Hardware...

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



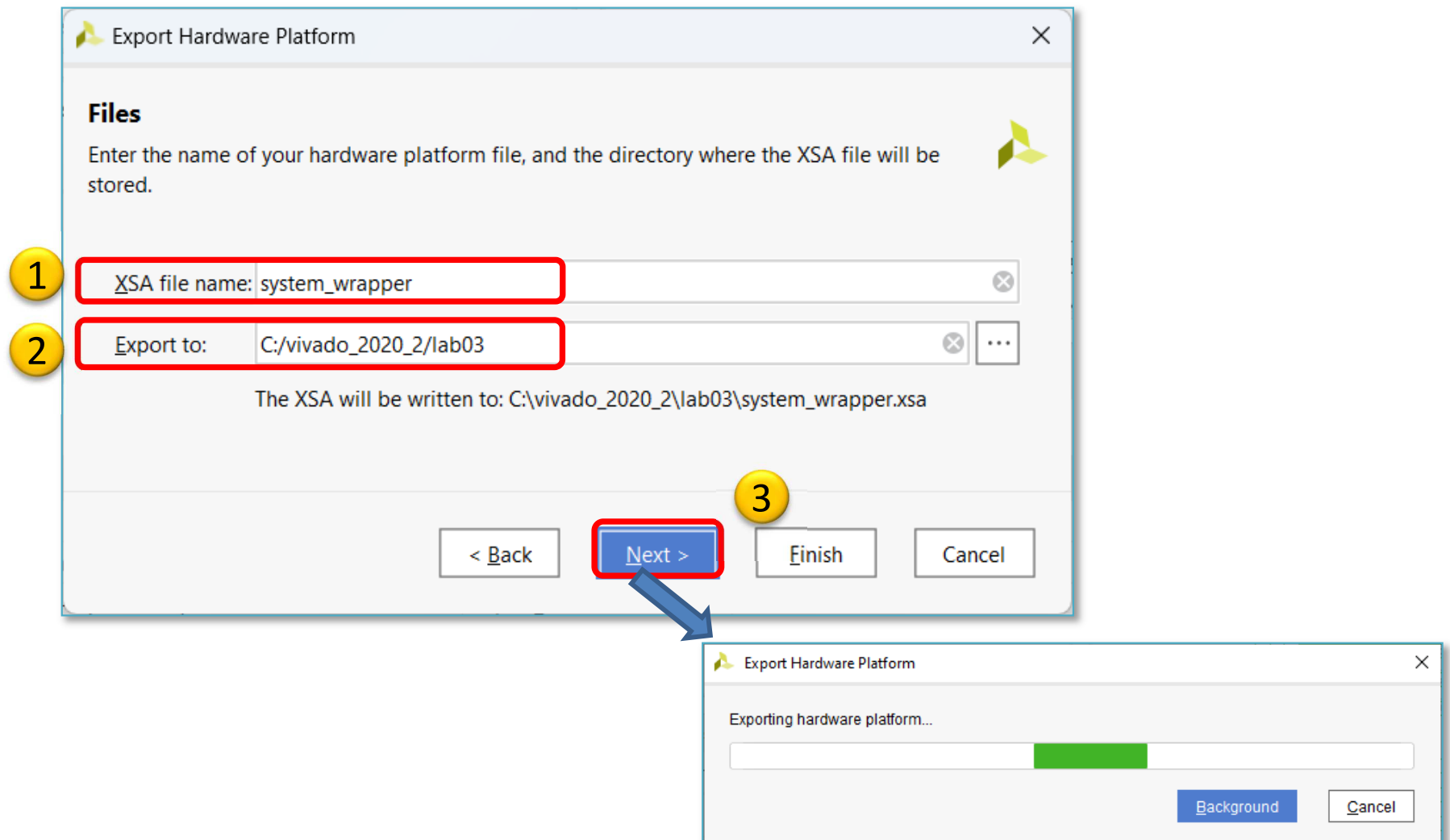
VIVADO Export HW → VITIS (cont.)

Select „Include bitstream” option as output:



Export HW → VITIS (cont.)

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





USING XILINX VITIS

LAB03. Creating a software test application for MyLED IP

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. *MyLEDApp* as system monitor 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. Creating a 'Debug Configuration' for hardware debugging
 4. Connecting and setup a JTAG-USB programmer,
 - Configuring the FPGA (**.BIT** hence PL-side was set)
 5. Setup a Serial terminal/Console (USB-serial port),
 6. Debug (insert breakpoints, stepping, run, etc.)

Starting VITIS



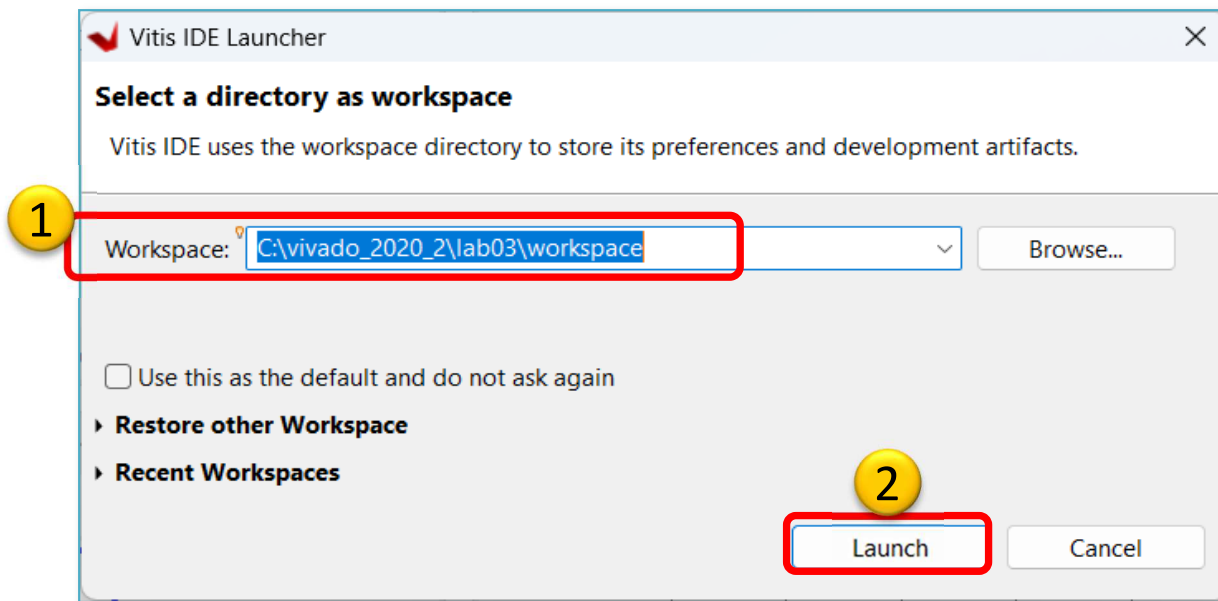
From Vivado: Tools menu → Launch VITIS IDE

OR externally

Start menu → Programs → Xilinx Design Tools → Xilinx VITIS 2020.2

Do Not run Xilinx VITIS HLS 2020.2 !

- Set workspace directory properly (*lab03*):
 - Recommended to use *vitis_workspace* as a subdirectory in your lab folder. Launch it...



Xilinx VITIS – Create Application

Recall the steps of the former LAB01/LAB02 ...

1. Create a new application project

- File → New → Application Project...

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

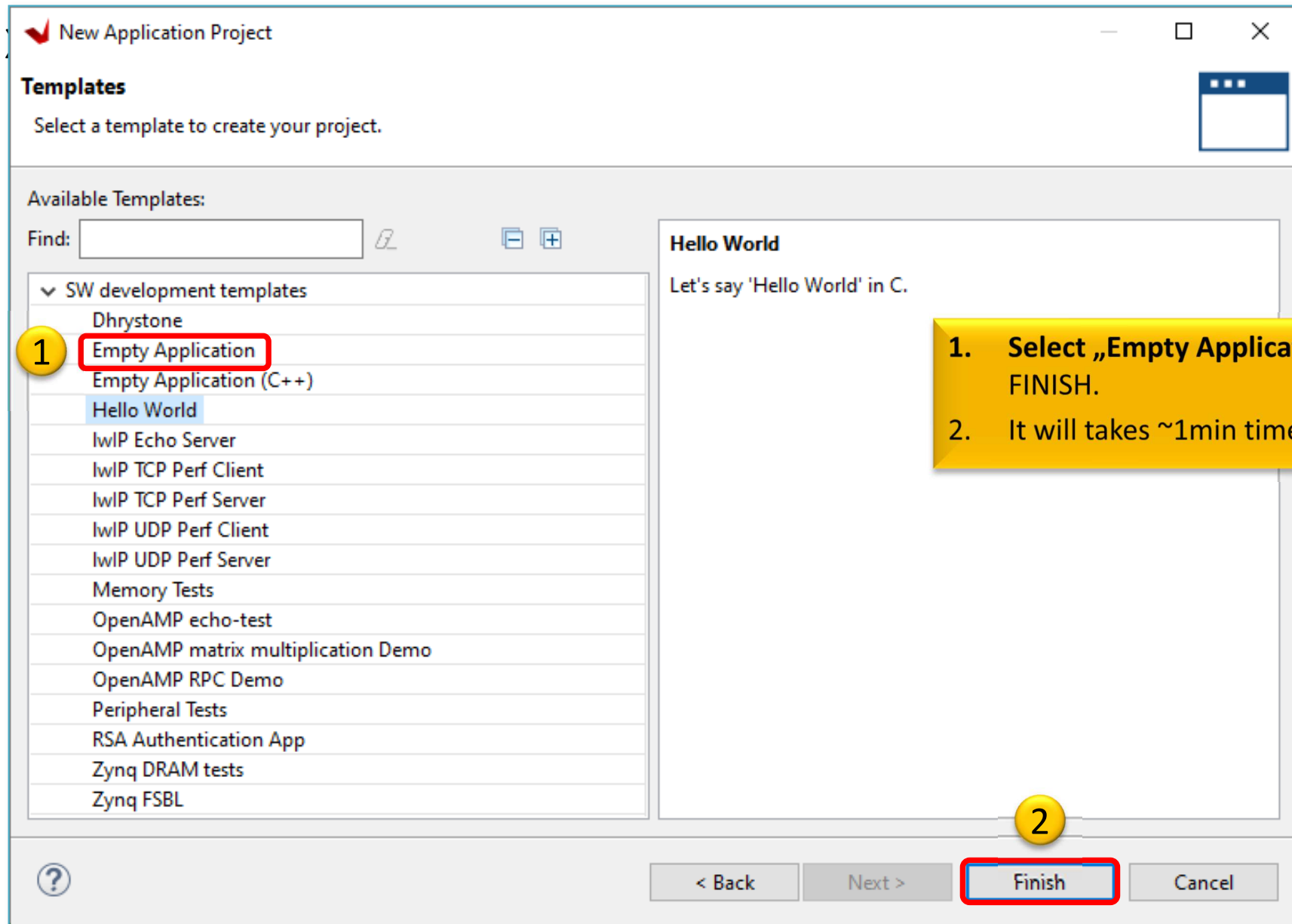
- Browse... for LAB03 `system_wrapper.xsa`. Open it.
- ! Do not select the „*Generate boot components*”

3. Application project details

- Type „`MyLEDApp`” as project name
- Type „`MyLEDApp_system`” as system project name
- Select `ps7_cortexa9_0` as target ARM core 0

4. Domain: leave settings as default (standalone)

Example I.) Creating MyLEDApp as empty application



VITIS GUI – Main window (HW)

workspace - system_wrapper/platform.spr - Vitis IDE

File Edit Search Xilinx Project Window Help

MyLEDApp_system MyLEDApp system_wrapper

Hardware Platform Specification

Design Information

Target FPGA Device: 7z010
Part: xc7z010clg400-1
Created With: Vivado 2020.2
Created On: Tue Apr 16 23:05:29 2024

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]

Filter: Search: 30 Loaded - 30 Shown - 1 Selected - [Custom: -- Table Default --]

Cell	Base Address	High Address	Slave Interface	Addr Range Type
dip	0x41200000	0x4120ffff	S_AXI	register
led_ip_0	0x41220000	0x4122ffff	S_AXI	register
pb	0x41210000	0x4121ffff	S_AXI	register
ps7_afi_0	0xf8008000	0xf8008fff	-	register
ps7_afi_1	0xf8009000	0xf8009fff	-	register
ps7_afi_2	0xf800a000	0xf800afff	-	register
ps7_afi_3	0xf800b000	0xf800bfff	-	register
ps7_coresight_comp_0	0xf8000000	0xf88fffff	-	register
ps7_ddr_0	0x00100000	0x1fffffff	-	memory
ps7_ddrc_0	0xf8006000	0xf8006fff	-	register
ps7_dev_cfg_0	0xf8007000	0xf80070ff	-	register

1

2

Led_IP_0: address map and specification of your custom IP core

Assistant

MyLEDApp_system [System]
MyLEDApp [Application]
Debug
Release
Debug
Release
system_wrapper [Platform]

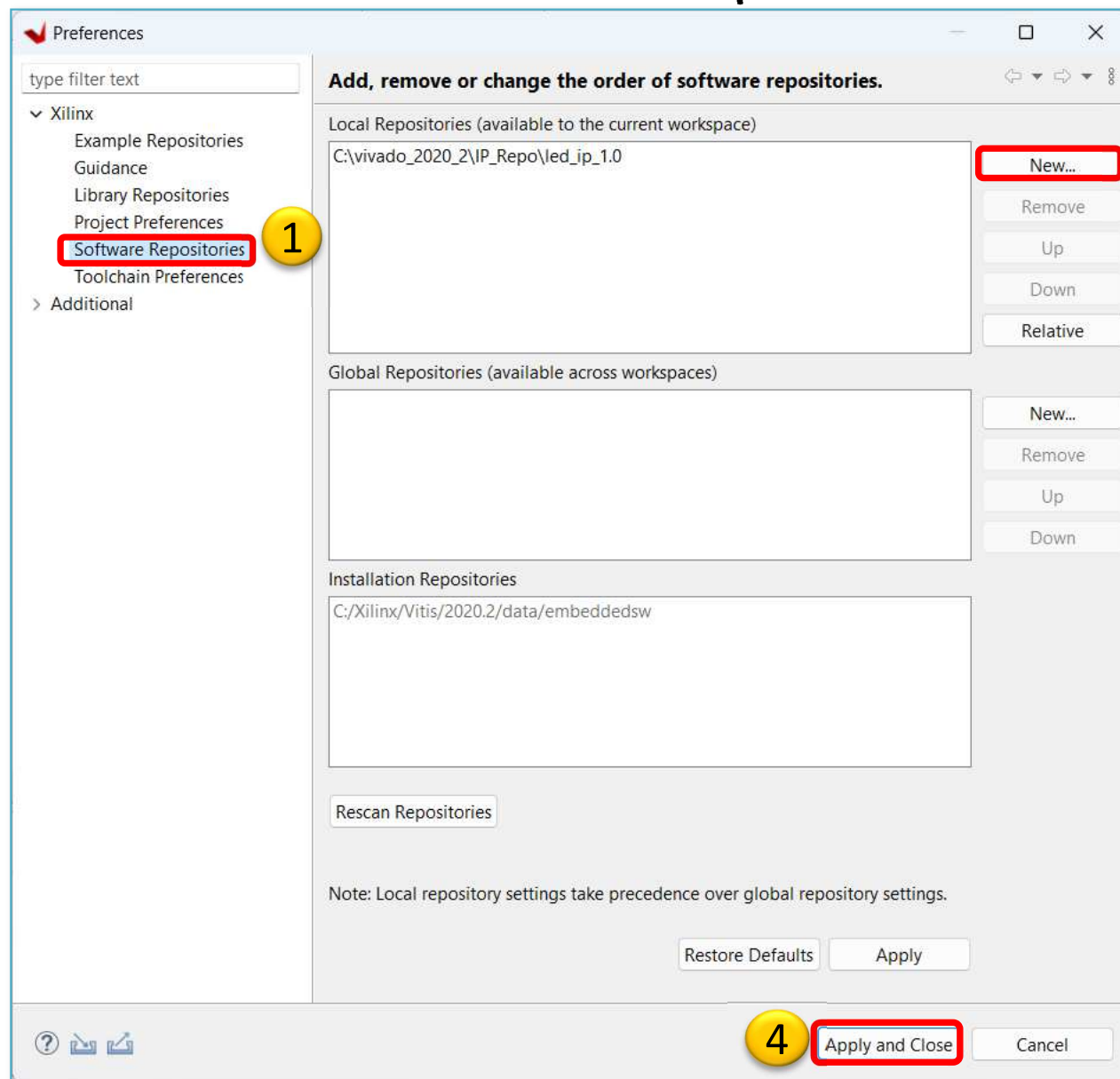
Main Hardware Specification

Console Problems Vitis Log Guidance

Build Console [MyLEDApp, Debug]

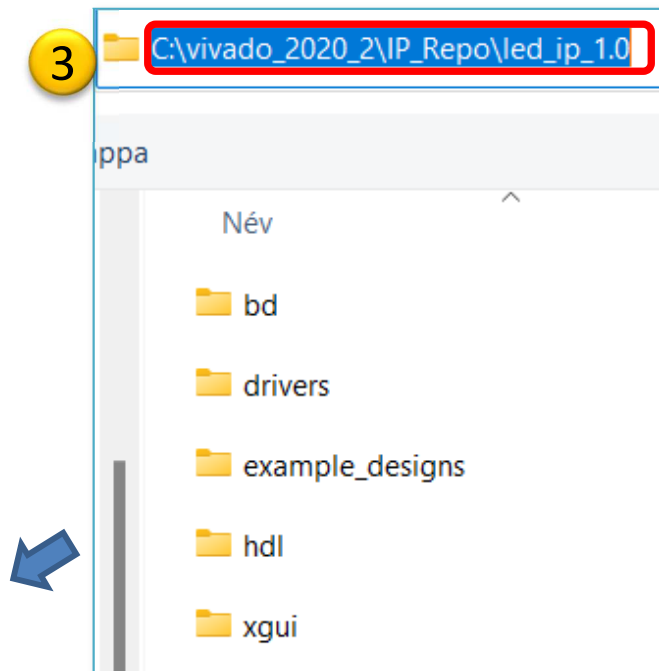
VITIS – Add Driver Repository

Xilinx menu → SW Repositories



Note:

New → global location where you created your IP with the previous LED_IP Package manager (add the directory level where your drivers are located `<dir>\led_ip_1.0`).



VITIS – Main window (SW-driver)

The screenshot shows the Vitis IDE interface with the following components:

- Explorer:** Displays the project structure. The file `platform.spr` is highlighted with a red box and a yellow circle with the number 1.
- Board Support Package:** Shows the current BSP settings for the `standalone` board. It includes a description of the BSP and a table of drivers.
- Drivers Table:** A table with columns: Name, Driver, Documentation, and Examples. The row for `led_ip_0` is highlighted with a red box and a yellow circle with the number 2.
- Assistant:** Shows the project configuration for `MyLEDAApp [Application]` and `system_wrapper [Platform]`.
- Console:** Displays the build console output for `MyLEDAApp_system, Debug`.

Board Support Package

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.

[Modify BSP Settings...](#) [Reset BSP Sources](#)

A BSP settings file is generated with the user options selected in the settings dialog. To use existing settings, click the below link. This operation clears any existing modifications done. All the subsequent changes are applied on top of the loaded settings.

[Load BSP settings from file](#)

Operating System

Name: standalone
Version: 7.3

Description: caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.

Documentation: [standalone v7.3](#)

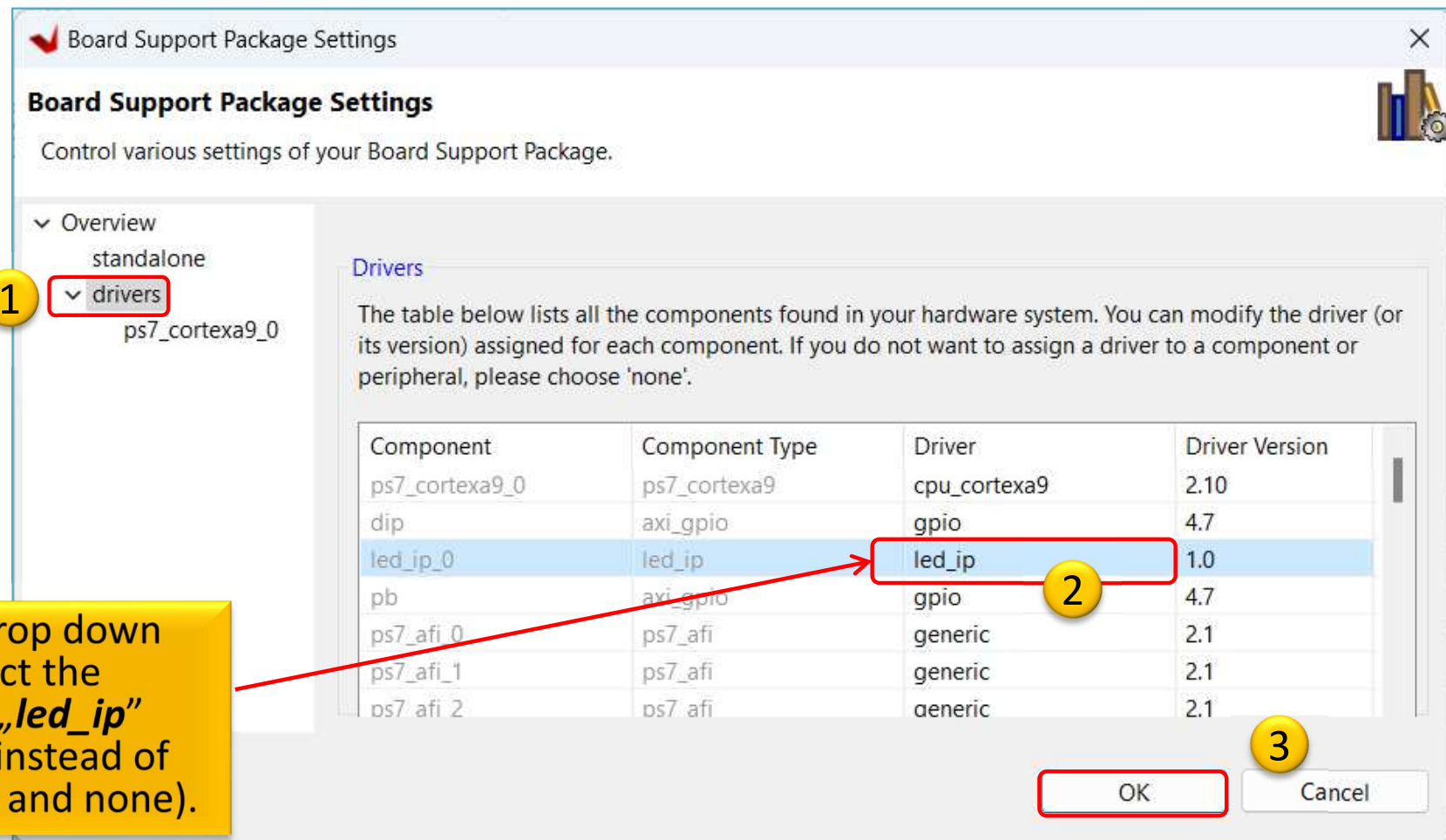
Name	Driver	Documentation	Examples
dip	gpio	-	Import Examples
led_ip_0	led_ip	-	Import Examples
pb	gpio	-	-
ps7_afi_0	generic	-	-
ps7_afi_1	generic	-	-
ps7_afi_2	generic	-	-
ps7_afi_3	generic	-	-
ps7_coresight_comp_0	coresightps_dcc	Documentation Link	-
ps7_ddr_0	ddr	Documentation Link	-

Led_IP_0: check the driver (led_ip)

If you see "generic", then the led_ip is not yet assigned to the correct driver!

VITIS – Set LED_IP driver

- Project Explorer → Right Click MyLEDAApp's → Board Support Package Settings



VITIS – SW project

- **Project Explorer** → double click on **lab3_led_ip.c** → Open the **Outline** → double click on **xparameters.h**
(This important header file can be generated after BSP compiled, and parameter values derived from Vivado settings)
- `#define XPAR_LED_IP_0_DEVICE_ID 0`
This macro defines our „**LED_IP**” custom peripheral
- This `#define` can be used to write to LEDs

LED_IP drivers

- Path :
 - <lab03_project>\system_wrapper\hw\drivers\led_ip_v1_0\src
- Investigate the content of .c, and .h source files (generated from Vivado tool)!
- Writing to the LED:

```
#define LED_IP_mWriteReg(BaseAddress, RegOffset, Data) \  
    Xil_Out32((BaseAddress) + (RegOffset), (u32)(Data))
```

Analyzing LED_IP application

- 1.) Read the actual state of **dip** switches (in an infinite loop)
- 2.) Write the value of dip switches on our LED_IP

```
#include "xparameters.h"
#include "xgpio.h"
#include "led_ip.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 Lab03 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);

        /*LED_IP_mWriteReg(BaseAddress, RegOffset, Data) \
        Xil_Out32((BaseAddress) + (RegOffset), (u32)(Data))*/
        LED_IP_mWriteReg(XPAR_LED_IP_0_S_AXI_BASEADDR, 0, dip_check);

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

Led_ip header file

Initialization of GPIO direction

Read actual state of DIP switches and print out.

Then write this value to the LEDs.

<Vivado project directory>
\\LAB_03_A_LED_IP\
drivers\\led_ip_v1_00_a\\src\\led_ip.h

Important Remark* - Makefile

*There is a build problem with VITIS 2020.x when creating a custom AXI-lite based IP. Makefile generation did not work properly (build error).

1. Open `system_wrapper\ps7_cortexa9_0\standalone_ps7_cortexa9_0\bsp\ps7_cortexa9_0\libsrc\led_ip_v1_0\src\Makefile`

2. Modify Makefile

```
COMPILER=
ARCHIVER=
CP=cp
COMPILER_FLAGS=
EXTRA_COMPILER_FLAGS=
LIB=libxil.a

RELEASEDIR=../.././lib
INCLUDEDIR=../.././include
INCLUDES=-I./ -I${INCLUDEDIR}

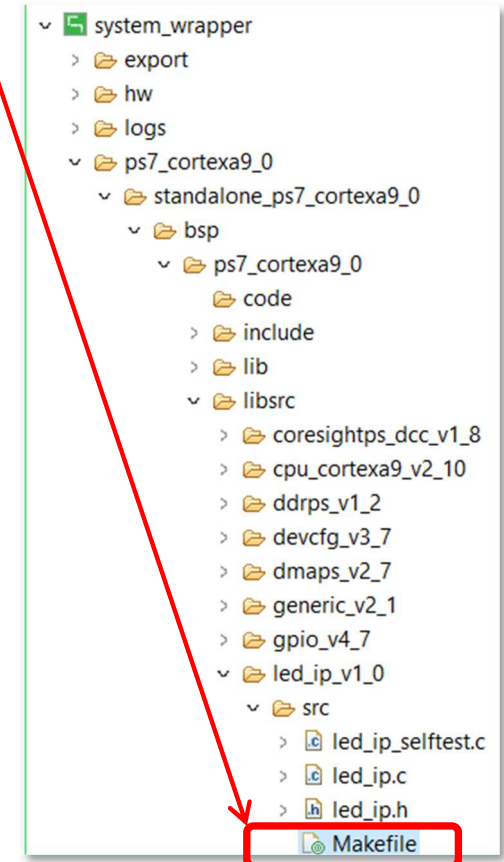
INCLUDEFILES=*.h
#LIBSOURCES=*.c
LIBSOURCES=$(wildcard *.c)
#OUTS = *.o
OUTS = $(addsuffix .o, $(basename $(wildcard *.c)))

libs:
echo "Compiling led_ip..."
$(COMPILER) $(COMPILER_FLAGS) $(EXTRA_COMPILER_FLAGS) $(INCLUDES) $(LIBSOURCES)
$(ARCHIVER) -r ${RELEASEDIR}/${LIB} ${OUTS}
make clean

include:
${CP} $(INCLUDEFILES) $(INCLUDEDIR)

clean:
rm -rf ${OUTS}
```

Modify OUTS parameter according to this line!



Generate Linker Script & Build

- Generate Linker Script to the internal on-chip PS7 **RAM0**

- Set the Heap / Stack size to **1KB!**



- Now rebuild the MyLEDApp again



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

```
'Invoking: ARM v7 Print Size'  
arm-none-eabi-size MyLEDApp.elf |tee "MyLEDApp.elf.size"  
   text    data     bss     dec     hexfilename  
 23368   1176   8248  32792  8018 MyLEDApp.elf  
'Finished building: MyLEDApp.elf.size'
```

MyLEDAApp – Verification result

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

```
Connected to COMX at 115200
-- Start of the Lab03 LedIP Program --
Dip Switches initialized successfully!
Push Buttons initialized successfully!
State of Dip switches 15!
State of Dip switches 0!
State of Dip switches 3!
State of Dip switches 0!
State of Dip switches 3!
State of Dip switches 4!
```

.....

LAB03 – Summary

- To the ARM-AXI based system created in the previous (5. – LAB02_A), here we designed and added a new PL-side **custom LED IP peripheral**, which is not part of the **Vivado** IP Catalog.
- Peripheral were properly configured to the BSB and connected to the external I/O pins of the FPGA.
- We examined both the Block Diagram and the report files.
- Finally, we verified the completed embedded system (HW+FW) and the correct operation of a SW application (**MyLEDApp**) in **VITIS** unified environment.

Task – Calculator test

- Create a **Calculator** SW application project (*CalcTest*)
- Modify the previous MyLEDApp SW application to implement a calculator capable of 4 basic operations.
 - Two operands (A,B) will be 2-2 bits, each: A[1:0], B[1:0], which are the values of the **dip switches (dip)**.
 - The following operations can be performed using **pushbuttons (pb)**:
 - `pb[2:0] = "000"` : addition,
 - `pb[2:0] = "001"` : subtraction,
 - `pb[2:0] = "010"` : multiplication,
 - `pb[2:0] = "011"` : division.
 - `pb[2:0] = "100"` : exit
 - Display the results of these operations - with only the integer part of the division - on MyLED[3:0] (also with `xprintf()`)

CalcTest – Verification result

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

```
Calculator Program --
Dip Switches initialized successfully!
Push Buttons initialized successfully!
Note: Addition      [+]: PB[2:0]=000
      Subtraction   [-]: PB[2:0]=001
      Multiplication [*]: PB[2:0]=010
      Division      [/]: PB[2:0]=011
      Exit          : PB[2:0]=100

-----
Initial value of result = 0
Partial result after addition 3 + 3 = 6
Partial result after subtraction 3 - 3 = 0
Partial result after multiplication 3 * 3 = 9
Partial result after division 3 / 3 = 1
Partial result after addition 2 + 3 = 5
Partial result after subtraction 2 - 3 = -1
Partial result after multiplication 2 * 3 = 6
Partial result after division 2 / 3 = 0
Exiting from Lab3a calculator!
...
```



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