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: 2. May. 2024.

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 (Ultrasonic sensor – HC-SR04) 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!

# ULTRASONIC SENSOR

HC SR04 sensor board


VIVADO HLx Editions

SZÉCHENYI 2020

MAGYARORSZÁG KORMÁNYA

Európai Unió
Európai Strukturális és Beruházási Alapok

BEFEKTETÉS A JÖVŐBE

# References

- **HC-SR04: Ultrasonic Sensor tutorial**

📖 https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/

- **Jordy Achten - HacksterIO tutorial: Minized and VITIS motor control with HC-SR04 (2020)**

📖 https://www.hackster.io/jordy-achten/minized-and-vitis-for-motor-control-with-added-hc-sr04-0e82cb

- **Xilinx Vivado – Creating custom Ips (UG1118)**

📖 https://docs.amd.com/v/u/2019.2-English/ug1118-vivado-creating-packaging-custom-ip

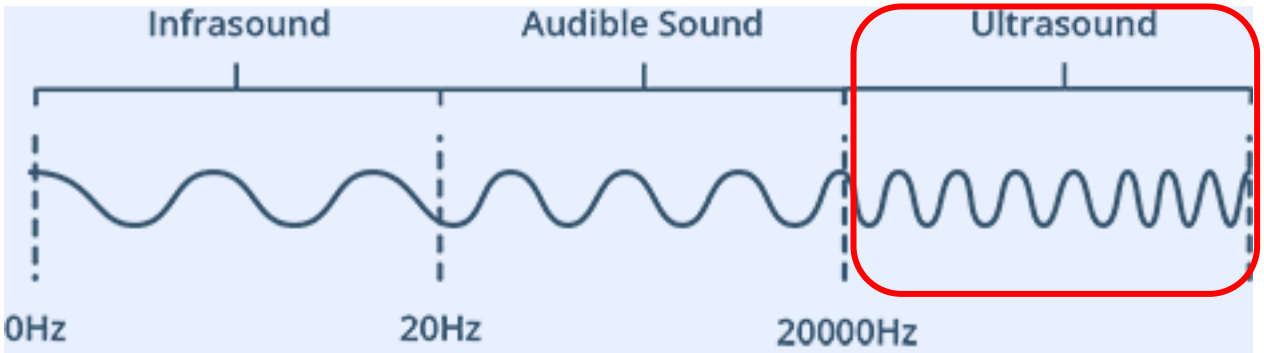
# HC-SR04 Ultrasonic sensor

**Sensor Specifications:**

| | |
|---|---|
| Operating Voltage | **DC 3V3** (and 5V tolerant) |
| Operating Current | 15 mA |
| Operating Frequency (T: transmitter) | 40 KHz |
| Max Range | 4 m |
| Min Range | 2 cm |
| Ranging Accuracy | ~3 mm |
| Measuring Angle | 15 degree |
| Trigger Input Signal | 10 µS TTL pulse |
| Dimension | 45 x 20 x 15mm |

# What is ultrasound?

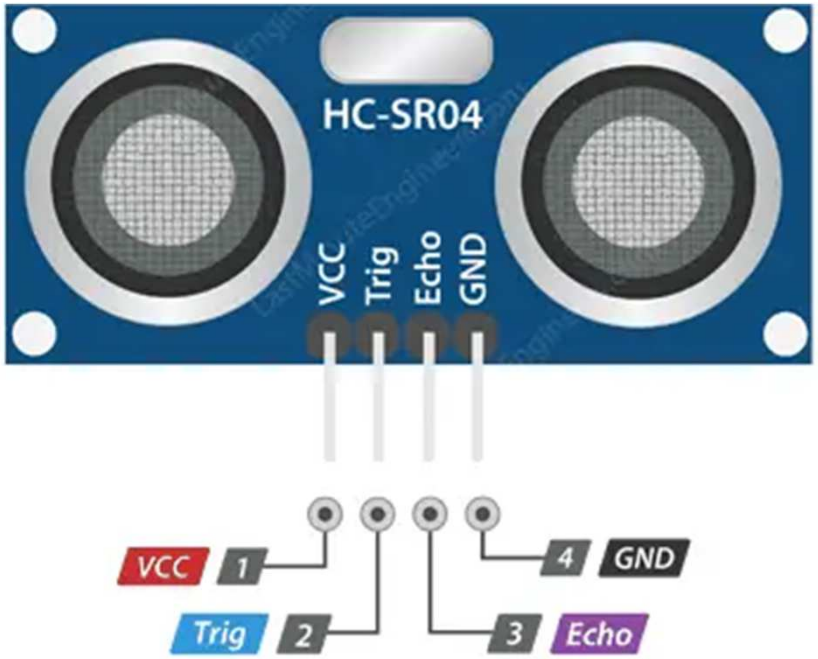
- Ultrasound is a high-pitched sound wave: it's frequency exceeds the audible range of human hearing.




- Humans can hear sound waves that vibrate in the range of about 20 times per sec (20 Hz, a deep rumbling noise) to 20,000 times a second (20 KHz, a high-pitched whistle).
- **Ultrasound has a frequency of more than 20 KHz and is therefore inaudible to humans**.

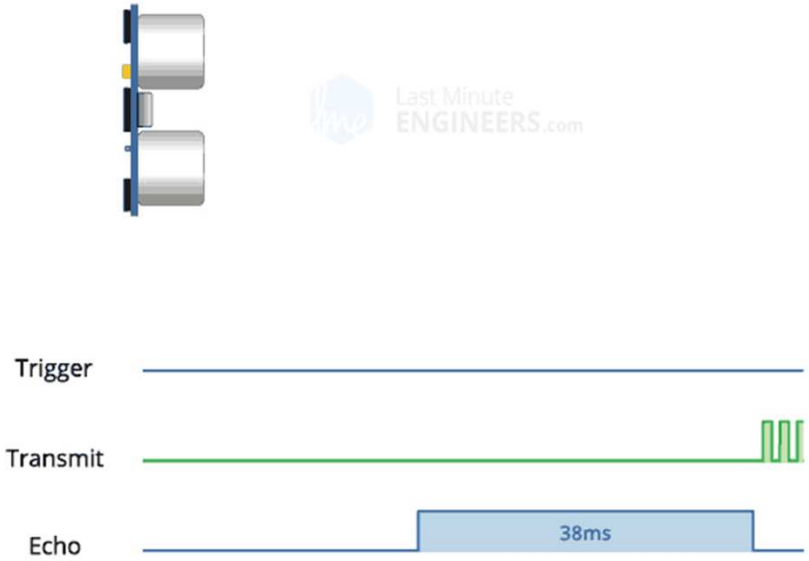
# HC-SR04 pinout

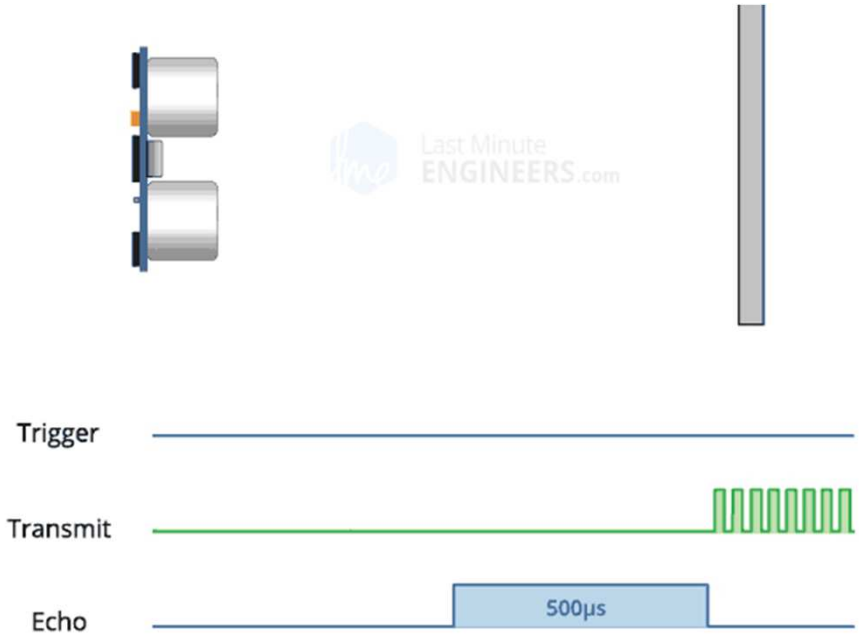
- 1. VCC supplies power to the HC-SR04 sensor. You can connect it to the 3.3V of PMOD connector.
- 2. Trig (Trigger) pin triggers ultrasonic sound pulses. By setting this pin to HIGH for **10µs**, the sensor initiates an ultrasonic burst.
- 3. Echo pin goes high when the ultrasonic burst is transmitted and remains high until the sensor receives an echo, after which it goes low. By measuring the time the Echo pin stays high, the **distance** can be calculated.
- **4. GND:** ground pin.



# HC-SR04 Ultrasonic Sensor

- **How does HC-SR04 Sensor work?**



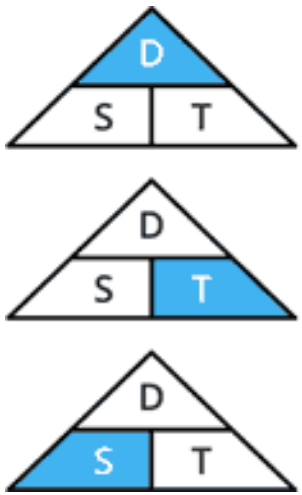




- Trigger pin is set HIGH for 10µs. In response, the sensor transmits an ultrasonic burst of eight pulses at 40 kHz. This is a 8-pulse pattern.
- Pulse pattern travel through the air. Meanwhile the echo pin goes HIGH to initiate the echo-back signal.

- A. If pulses are not reflected back, the echo signal times out and goes low after 38ms.
- B. If pulses are reflected back, this generates a pulse on the echo pin whose width varies from 150 µs to 25 ms depending on the **time taken to receive the signal => calculate the distance!**

# Calculate the distance

- The width of the received pulse is used to calculate the distance from the reflected object.




Distance = Speed x Time

$$\text{Time} = \frac{\text{Distance}}{\text{Speed}}$$

$$\text{Speed} = \frac{\text{Distance}}{\text{Time}}$$

Example: suppose that
time := 500 µs.
(The speed of sound is 340 m/s.)

**To calculate the distance,** we need to convert the speed of sound into cm/µs.
It is 0.034 cm/µs.

Distance = 0.034 cm/µs x 500 µs

Distance = (0.034 cm/µs x 500 µs) / 2

**Distance = 8.5 cm**

Last Minute ENGINEERS.com

# 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 Ultrasonic Sensor (HC-SR04) 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: `HC_SR04_IP_mReadReg()`

# Main steps to solve the task

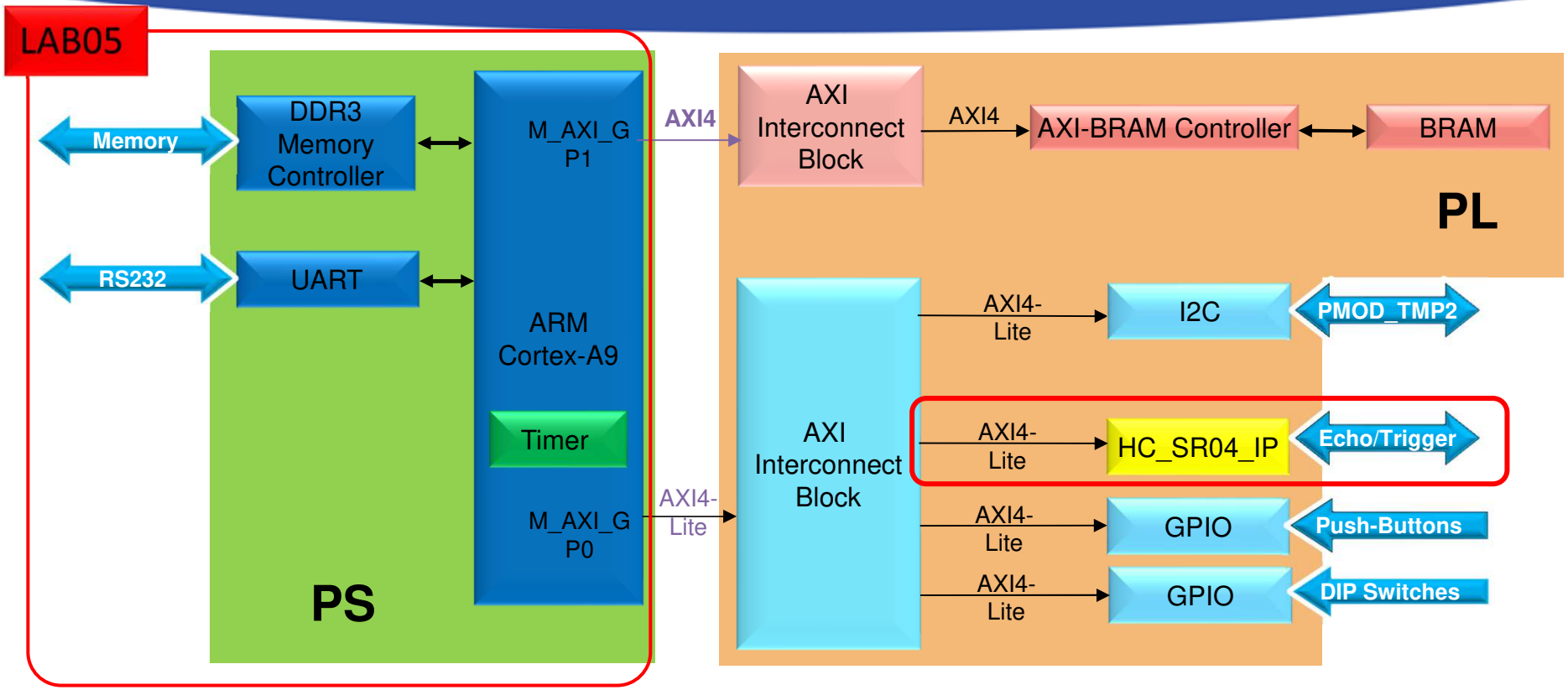
- Create a new project based on previous lab (LAB02_A) by using the **Xilinx Vivado (IPI)** embedded system designer,
  - `LAB02_A` project → Save as… → `LAB05` !
- Create and generate custom IP Peripheral in Package IP Wizard,
- Select and add custom IP Peripheral to the base system,
- Parameterize 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 2020.2
- Open the previous project! (`LAB02_A`)
  - File → Project → Open… / Open Recent…
  - `<projectdir>/LAB02_A/<system_name>.xpr` → **Open**

- File → Project → Save As… → LAB05
  - (This will save the former project LAB02_A as LAB05)

# Test system to be implemented




**PS side:**

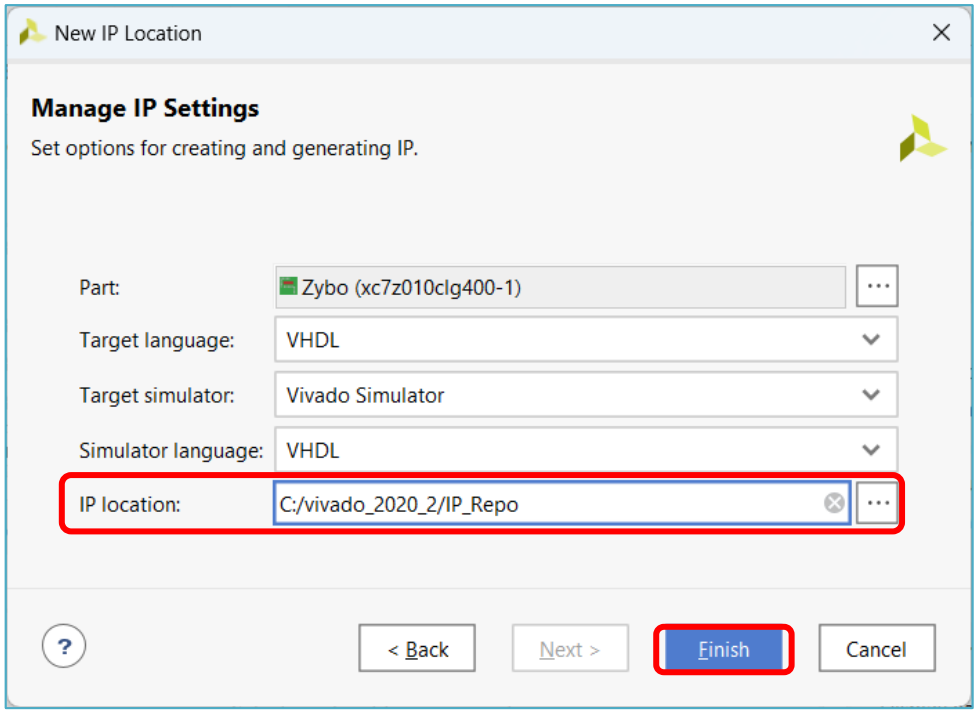
- **ARM hard-processor (Core0)**
- **Internal OnChip-RAM controller**
- **UART1 (serial) interface**
- **External DDR3 memory controller**

PL side (in FPGA logic)

- 2 GPIOs for Push Button and Dip Switches
- LAB05: custom HC_SR04 Ultrasonic IP

# Add IP path (similar to LAB03)

- File → IP → New Location... → Next

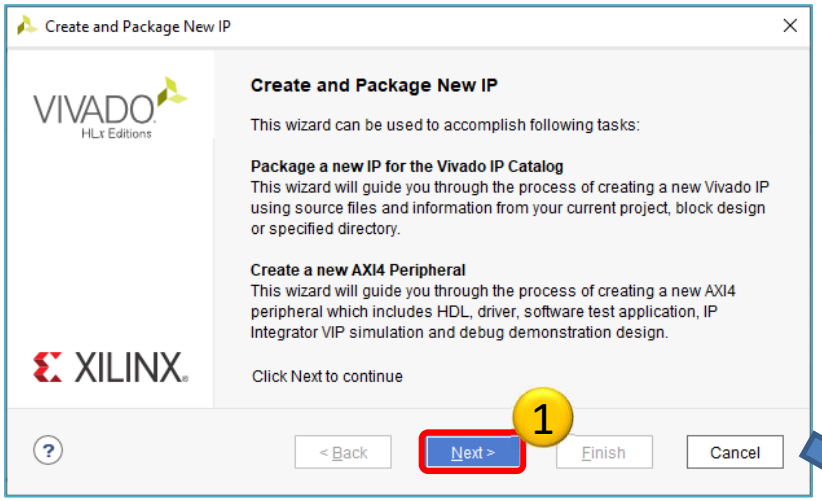


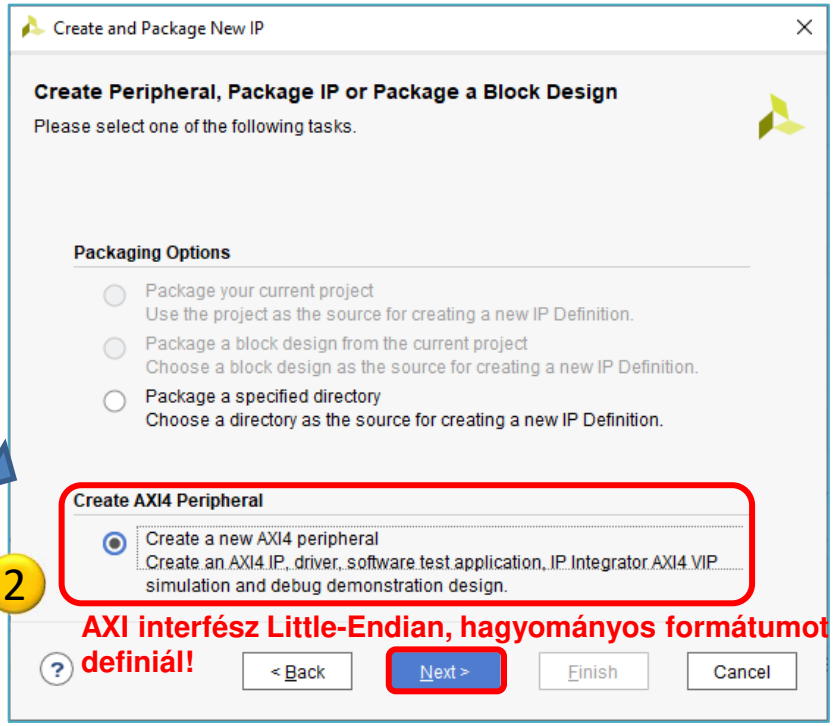

A new IP can be located at:
a.) locally into the actual project directory **or**
b.) globally into the Vivado's **IP Catalog** (~global **repository**)
We want to use this latter now
add `\IP_Repo` at the and of path (where our previous projects located).

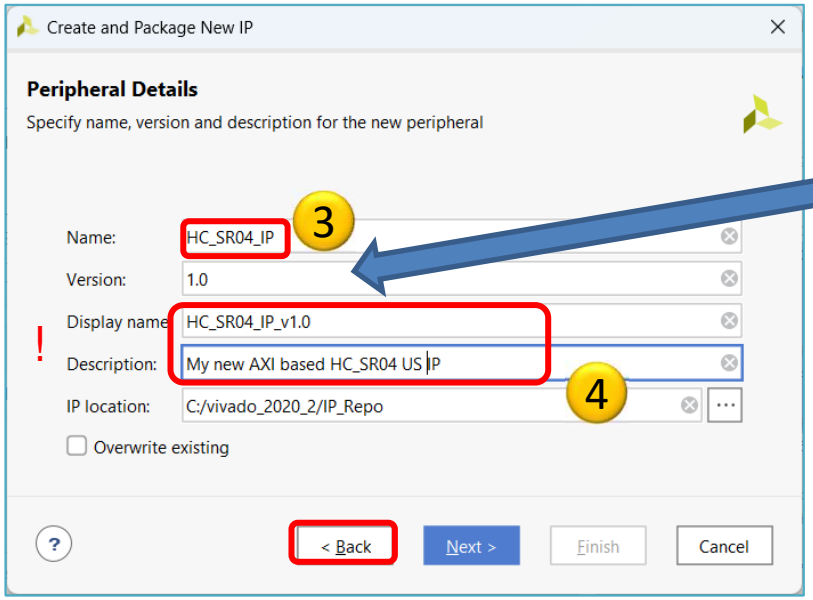
`\IP_Repo\managed_ip_project` subdirectory created with an **.xpr** project file.

# IP Wizard – LED IP peripheral (I.)

- Tools → Create and Package New IP… → Next



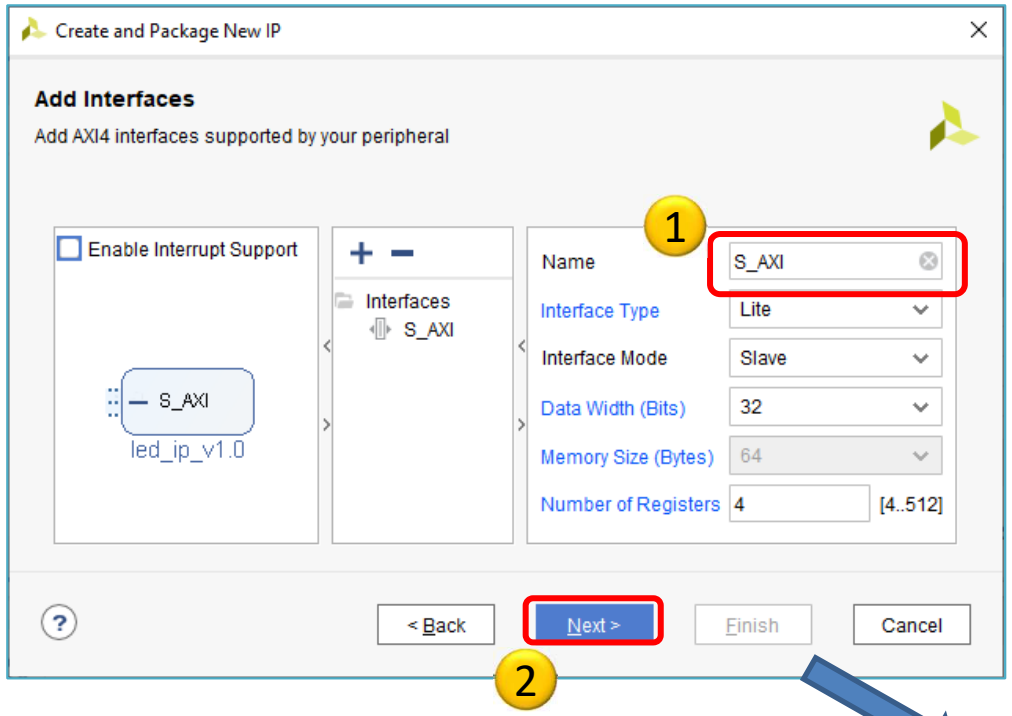

New IP name: „*HC_SR04_IP*"
Version: 1.00.a (default)
Description: as you want
`\HC_SR04_IP`-t tegyük be a `\IP_Repo` -alá. NEXT >>

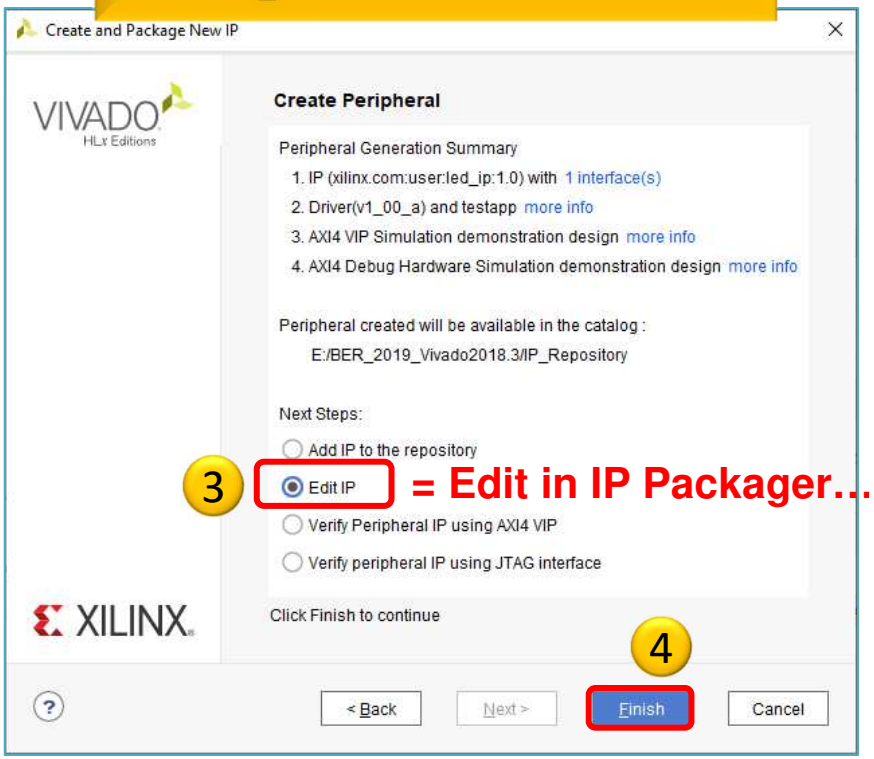
# IP Wizard – LED IP peripheral (II.)

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

HDL:

- DATA_WIDTH
- MEMORY_SIZE
- NUM_REG




= Edit in IP Packager…

# Project Manager – Package IP template

New menu option: Edit Packaged IP

Open the top-level HDL : `HC_SR04_IP_v1_0.vhd`

HC_SR04_IP_v1_0_project - [c:/vivado_2020_2/lab05/lab05.tmp/hc_sr04_ip_v1_0_project/HC_SR04_IP_v1_0_project.xpr] - Vivado 2020.2

File Edit Flow Tools Reports Window Layout View Help Quick Access

Flow Navigator

PROJECT MANAGER
- Settings
- Add Sources
- Language Templates
- IP Catalog
- Edit Packaged IP (1)

IP INTEGRATOR
- Create Block Design
- Open Block Design
- Generate Block Design

SIMULATION
- Run Simulation

PROJECT MANAGER - HC_SR04_IP_v1_0_project

Sources (2)
- Design Sources (2)
  - HC_SR04_IP_v1_0(arch_imp) (HC_SR04_IP_v1_0.vhd) (1)
    - HC_SR04_IP_v1_0_S_AXI_inst : HC_SR04_IP_v1_0_S_AXI(ai
      - axi_HC_SR04 : HC_SR04(Behavioral) (HC_SR04.vhd)
- IP-XACT (1)
- Constraints

Hierarchy Libraries Compile Order

Properties

Project Summary | Package IP - HC_SR04_IP

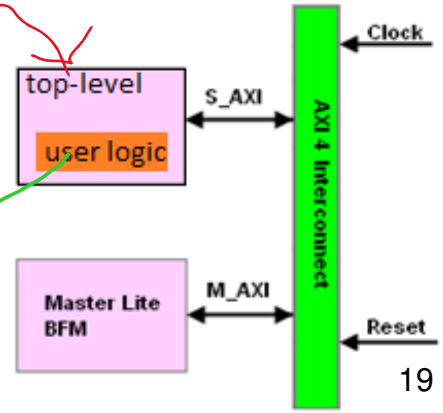
Packaging Steps (3)
- Identification
- Compatibility
- File Groups
- Customization Parameters
- Ports and Interfaces
- Addressing and Memory
- Customization GUI
- Review and Package

Identification

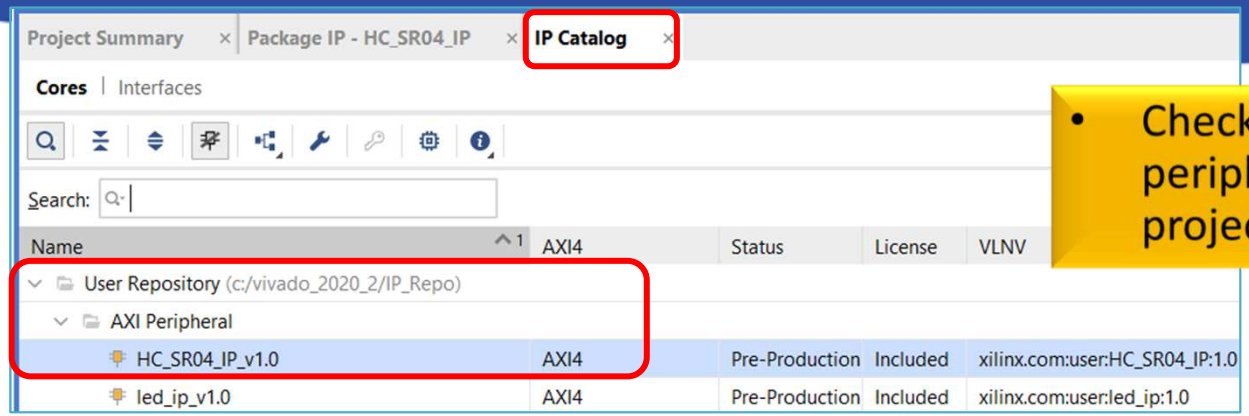
| | |
|---|---|
| Vendor: | xilinx.com |
| Library: | user |
| Name: | HC_SR04_IP |
| Version: | 1.0 |
| Display name: | HC_SR04_IP_v1.0 |
| Description: | My new AXI based HC_SR04 US IP |
| Vendor display name: | |
| Company url: | |
| Root directory: | c:/vivado_2020_2/IP_Repo/HC_SR04_IP_1.0 |
| Xml file name: | c:/vivado_2020_2/IP_Repo/HC_SR04_IP_1.0/component.xml |

Hierarchy (design sources):

- `HC_SR04_IP_v1_0.vhd` (top-level wrapper = „interface logic" template)
  - `HC_SR04_IP_v1_0_S_AXI.vhd` (user-logic =„R/W register template")



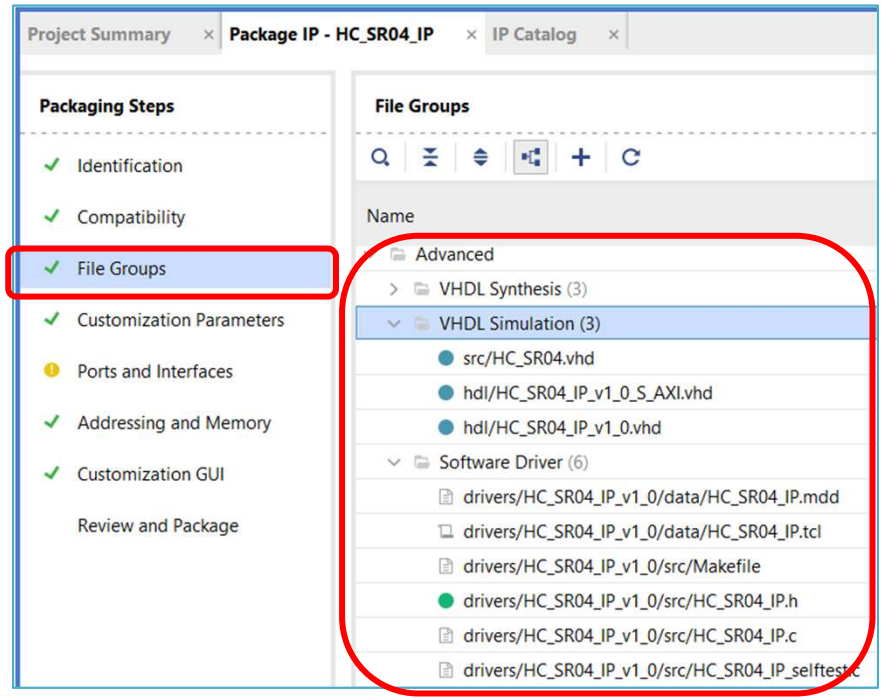
# Generate IP peripheral – IP Catalog




- Check! Has your own *HC_SR04_IP* peripheral been created in your project?

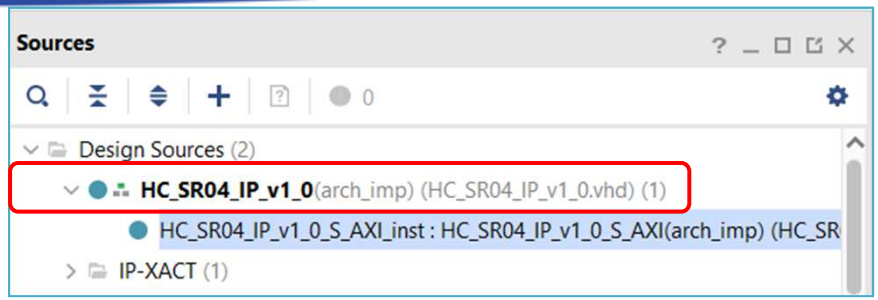
NOTE: IP-XACT is a standard **xml-based descriptor** (component.xml) that contains definitions, macros, descriptors of custom, reusable, pluggable IPs that can be integrated into an electronic circuit system - in our case an embedded system.

- *HC_SR04_IP* file/directory structure



# Modify peripheral template I. - HDLs

- Open the **„top-level"** **HC_SR04_v1_0.vhd**




Add the following lines to the file:

```
17      port (
18          -- Users to add ports here
19          clk          : in STD_LOGIC;
20          echo         : in STD_LOGIC;
21          trigger      : out STD_LOGIC;
22          -- User ports ends
```

1

Finally (CTRL+S or Save)

**19-21. lines:** extend entity's PORT list

*Add these ports to the entity*

***Note**: CLK requires 100 MHz signal from PL-side!!*

```
59          port (
60          clk          : in STD_LOGIC;
61          echo         : in STD_LOGIC;
62          trigger      : out STD_LOGIC;
63          S_AXI_ACLK   : in std_logic;
```

2

**60-62. lines**: add ports to the component list (user_logic)

```
95      port map (
96          clk => clk,
97          echo => echo,
98          trigger => trigger,
99          S_AXI_ACLK  => s_axi_aclk,
```

3

**96-98. line:** map ports to user_logic

# Modify peripheral template II. - HDLs

- Open **„sub-level”** **HC_SR04_v1_0_S_AXI.vhd-t** **(as „sub-modul”)**
- Add the following lines to the file: :

Finally (CTRL+S or Save)

```
17      port (
18          -- Users to add ports here
19          clk          : in STD_LOGIC;
20          echo         : in STD_LOGIC;
21          trigger      : out STD_LOGIC;
22          -- User ports ends
```

1

**19-21. lines:** extend entity's PORT list

*Add these ports to the entity*

```
102     -- Example-specific design signals
103     signal sonar_out_internal     : STD_LOGIC_VECTOR(C_S_AXI_DATA_WIDTH-1 downto 0);
104     -- local parameter for addressing 32 bit / 64 bit C_S_AXI_DATA_W
```

2

**103. line**: add internal signal *sonar_out_internal* (it is a 32-bit, LE)

```
124     component HC_SR04
125     Port (clk        : in STD_LOGIC;
126         echo       : in STD_LOGIC;
127         trigger    : out STD_LOGIC;
128         sonar_out  : out STD_LOGIC_VECTOR(15 downto 0));
129     end component;
130
131 begin
```

3

**124-129. lines**: add component's port list

```
358
359     process (sonar_out_internal, slv_reg0, slv_reg1, slv_reg2, slv_reg3, axi_araddr,
360     variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
361     begin
362         -- Address decoding for reading registers
363         loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto A
364         case loc_addr is
365           when b"00" =>
366             reg_data_out <= sonar_out_internal;
```

4

**359. and 366 lines**: extend process with our `sonar_out_internal` signal and assign it to the `reg_data_out`!

# Modify peripheral template III. (cont.) - HDLs

- Still open **„sub-level”** **HC_SR04_v1_0_S_AXI.vhd-t** **(as „sub-modul”)**
- Add the following lines to the file:

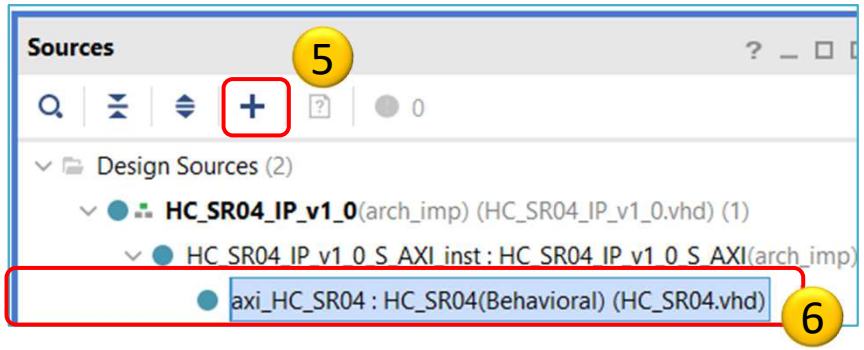
4

```
397     -- Add user logic here
398     axi_HC_SR04 : HC_SR04
399         Port map (  clk         => clk,
400                     trigger     => trigger,
401                     echo        => echo,
402                     sonar_out   => sonar_out_internal(15 downto 0));
403     -- User logic ends
```

398-402. lines: own VHDL code soure here: mapping the HC_SR04 component ports to the user logic.

Finally (CTRL+S or Save)

- Add *HC_SR04.vhd VHDL source* as **„user_logic”** file:

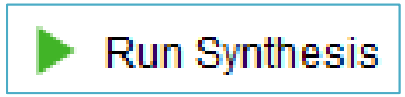



This VHDL source implements the ultrasonic sensor.

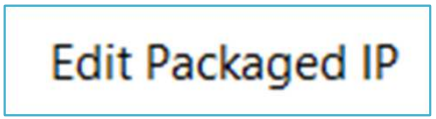
BER_09_LAB05_HC_SR04.zip

# 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).

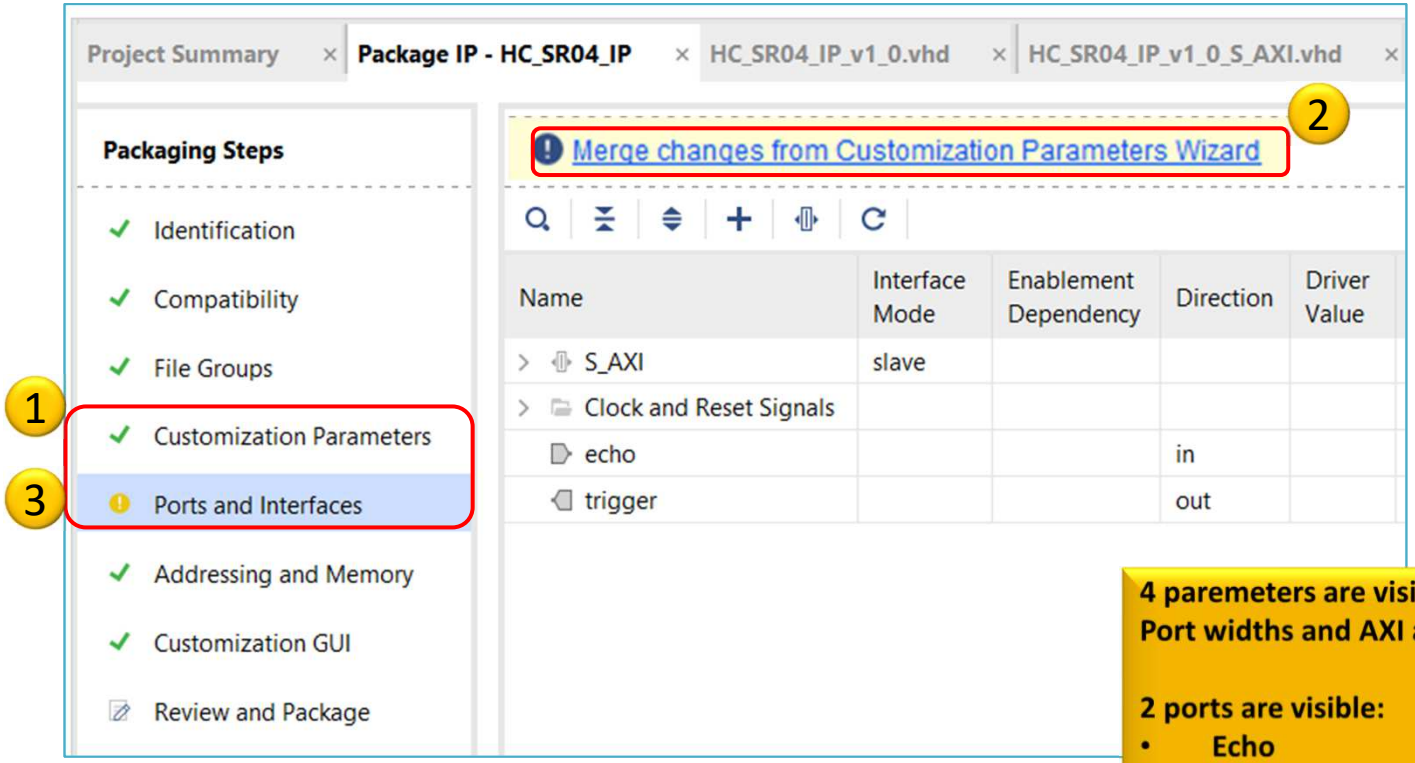



Project Manager → Edit Package IP:




- Open **HC_SR04_IP**

# Package IP – Customization Parameters




**4 paremeters are visible**
**Port widths and AXI addresses**

**2 ports are visible:**

- **Echo**
- **Trigger**

If yellow circle on the page which means that there are warnings. These are not critical, just ignore it.

# Package IP – Review and Package

Package IP - HC_SR04_IP | IP Catalog | HC_SR04_IP_v1_0_S_AXI.vhd | HC_SR04_IP_v1_0.vhd | HC_SR04.vhd

**Packaging Steps**

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

**Review and Package**

1 warning 2 info messages

Remember, where the „HC_SR04_IP" project was generated

**Summary**

Display name: HC_SR04_IP_v1.0

Description: My new AXI based HC_SR04 US IP

Root directory: c:/vivado_2020_2/IP_Repo/HC_SR04_IP_1.0

**After Packaging**

Create archive of IP - c:/vivado_2020_2/IP_Repo/HC_SR04_IP_1.0/xilinx.com_user_HC_SR04_IP_1.0.zip

edit

Project will be removed after completion

Edit packaging settings

Package IP

1 2 3

Project
IP Defaults
Text Editor
3rd Party Simulators
Colors
Selection Rules
Shortcuts
Strategies
Window Behavior

Packager
cify settings related to IP Packager.

ult Values

The following values will be automatically applied after finishing the IP Packager Wizard.

Vendor: xilinx.com
Library: user
Category: /UserIP
IP location: ../ip_repo

**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.

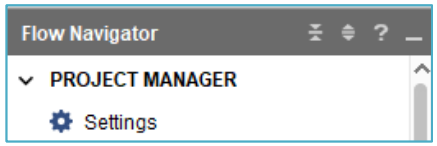
4 5

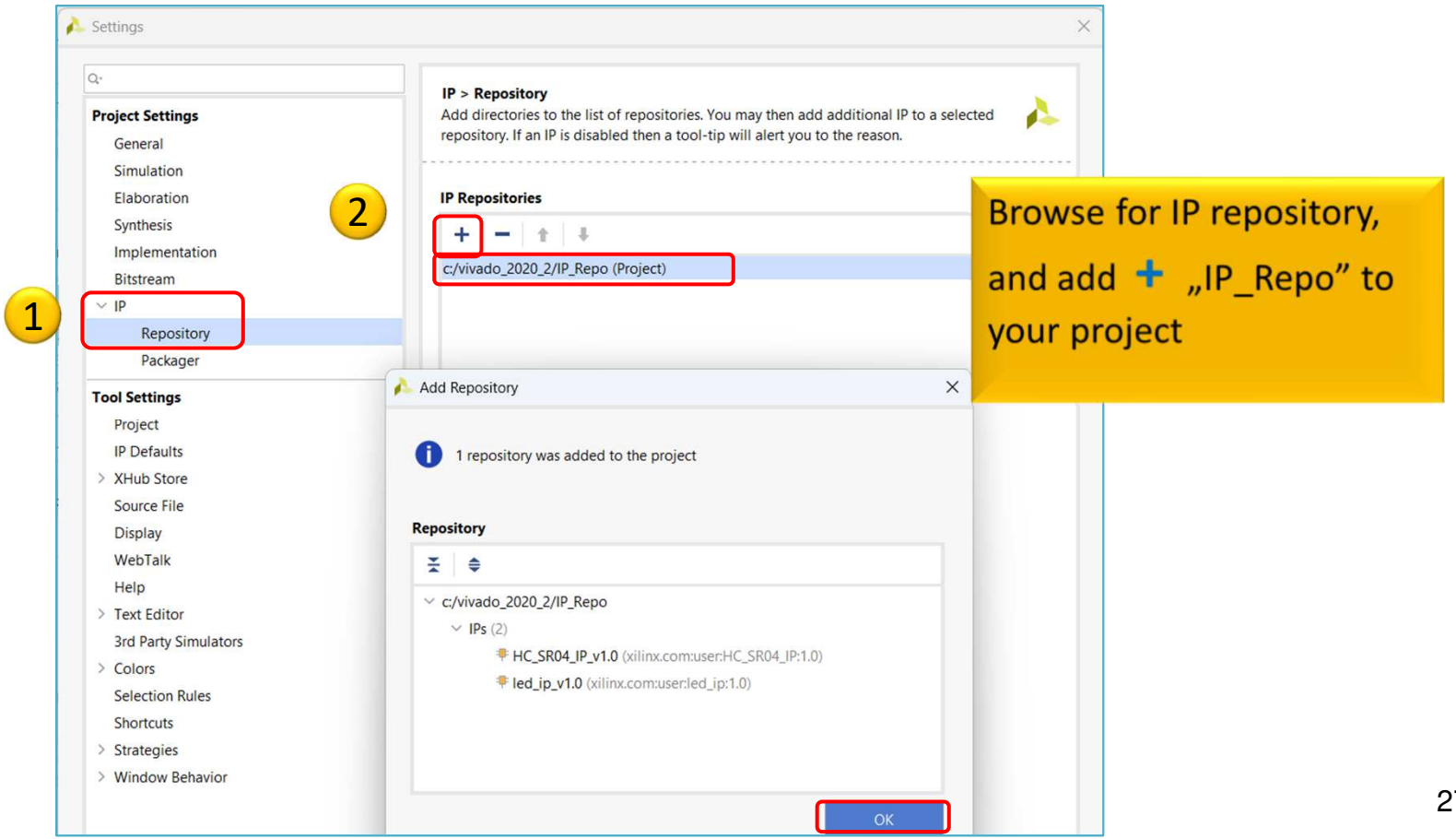
OK | Cancel | Apply | Restore

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

# Return to LAB05

- Open project → Choose „LAB05”
  - Project Manager → Settings
  - Select IP → + → Add IP path



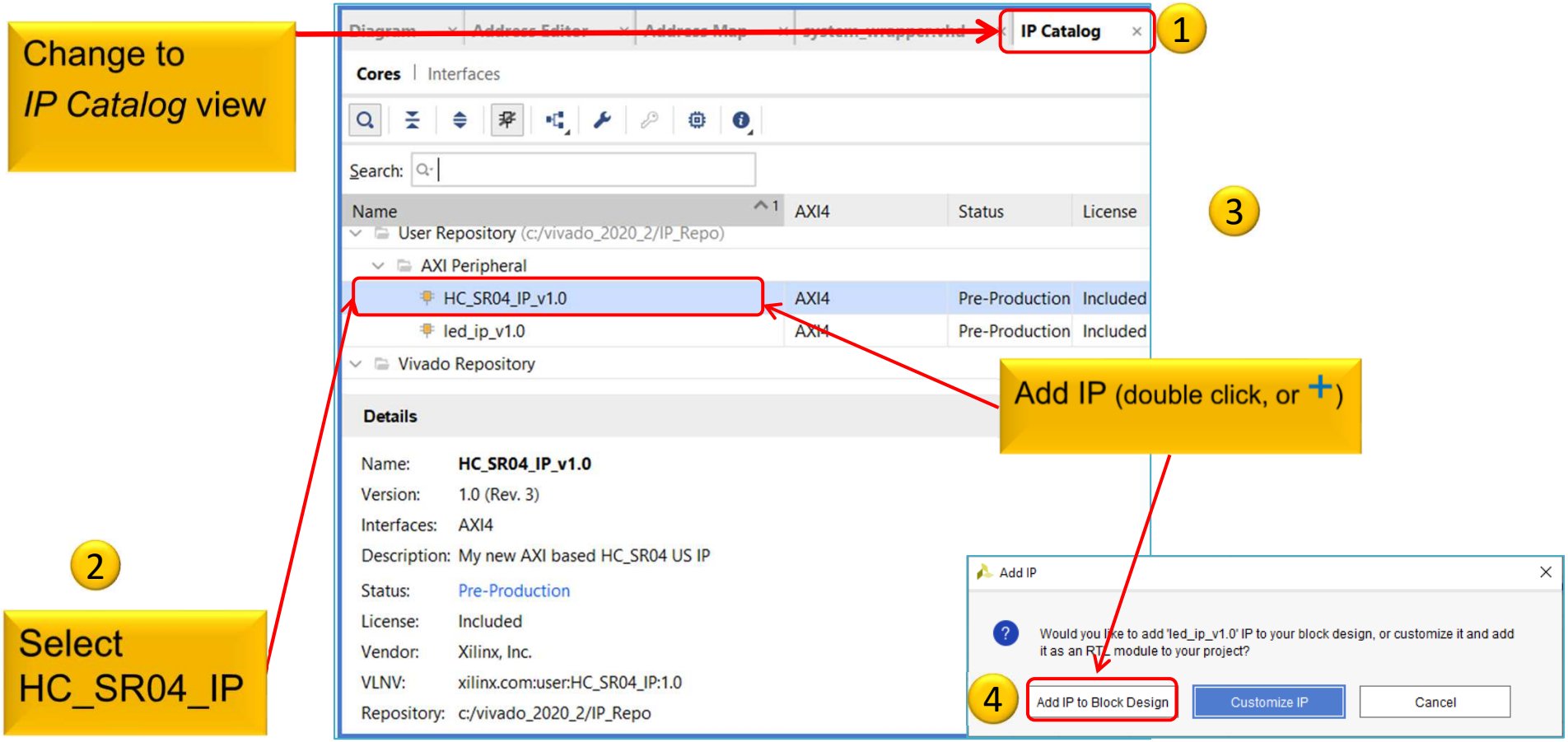
# Adding and connecting PL side HC_SR04_IP to the base system I.

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 HC_SR04_IP peripheral on the PL side to the BSB



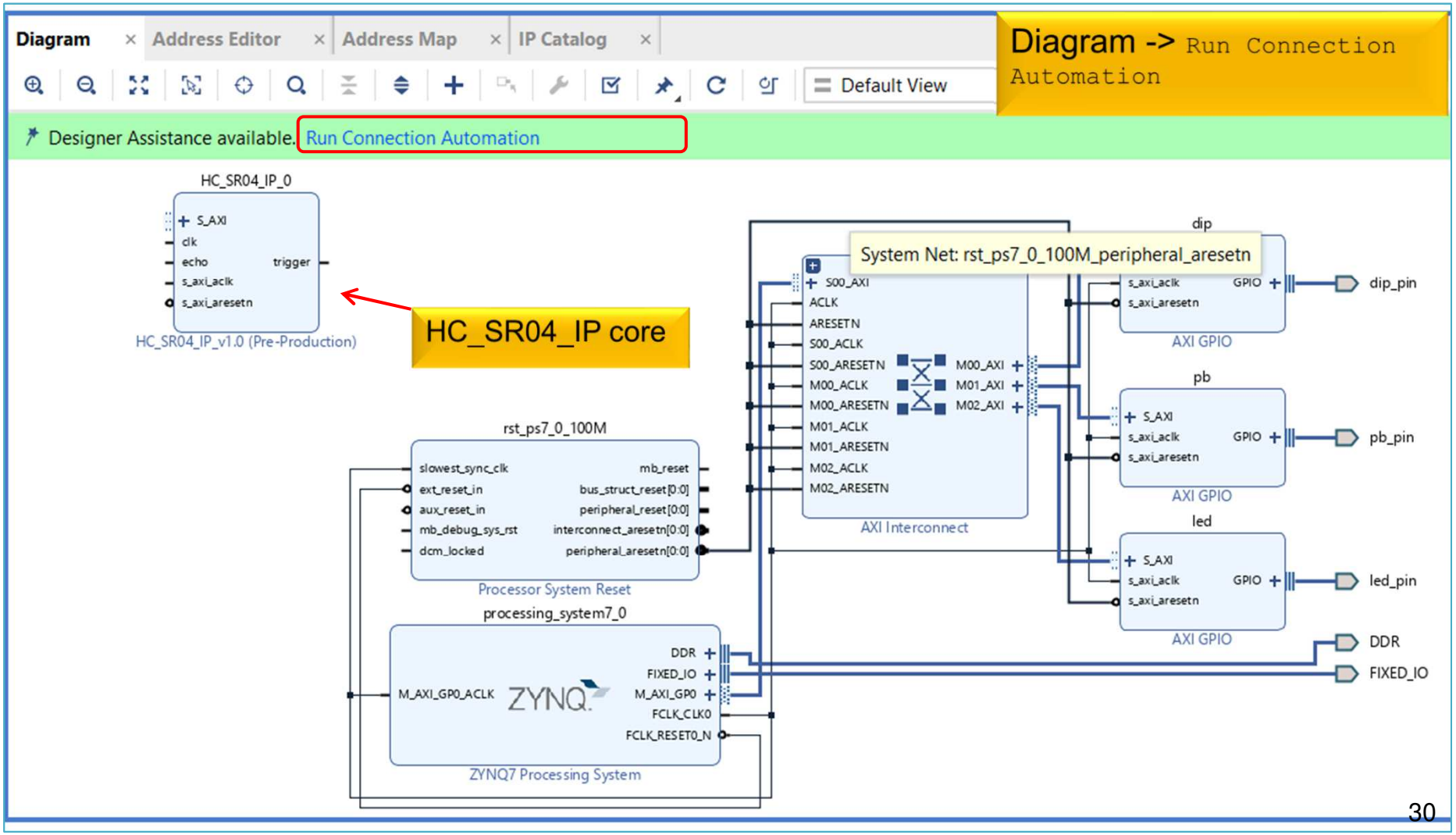
# Adding and connecting PL side HC_SR04_IP to the base system II.

Now, for your own IP module (HC_SR04_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.

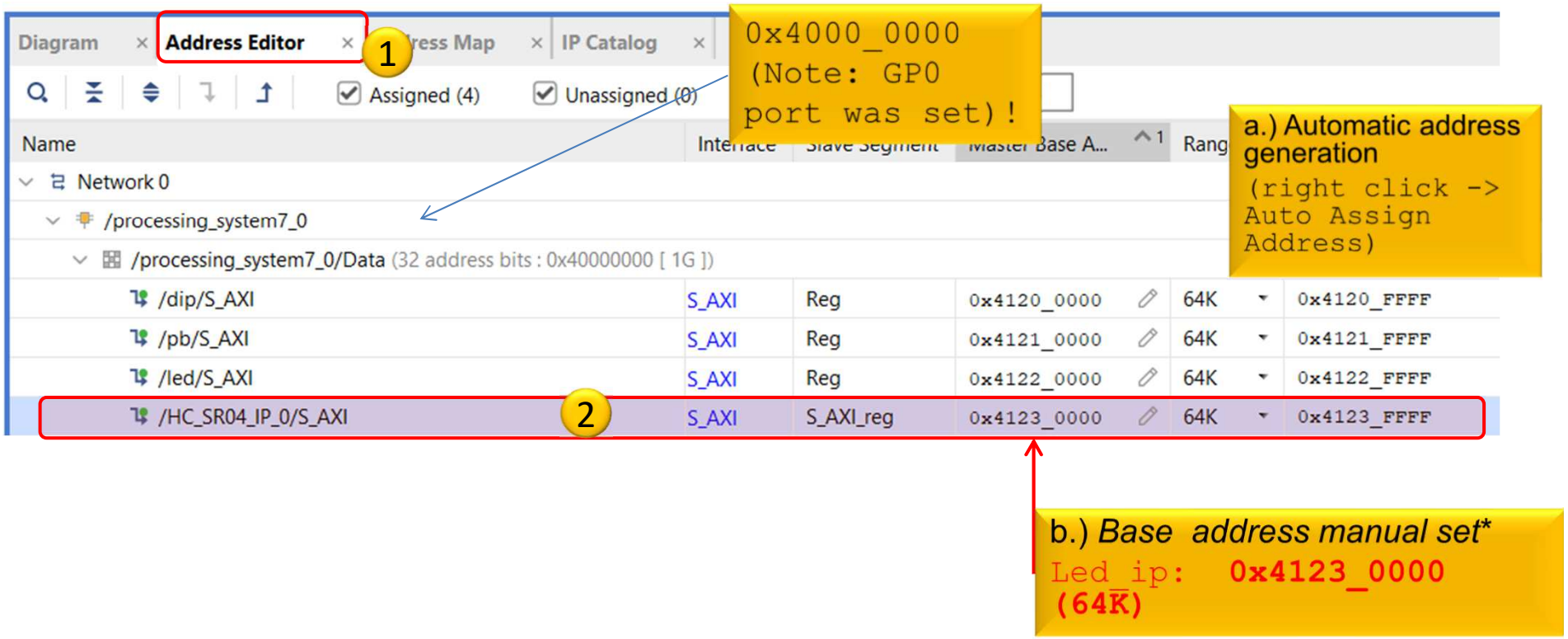
# Block diagram

Double-click on **HC_SR04_0** and examine its parameters.



# HCSR04_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)**




*Address ranges must be aligned into 2^n size and cannet be overlapped!

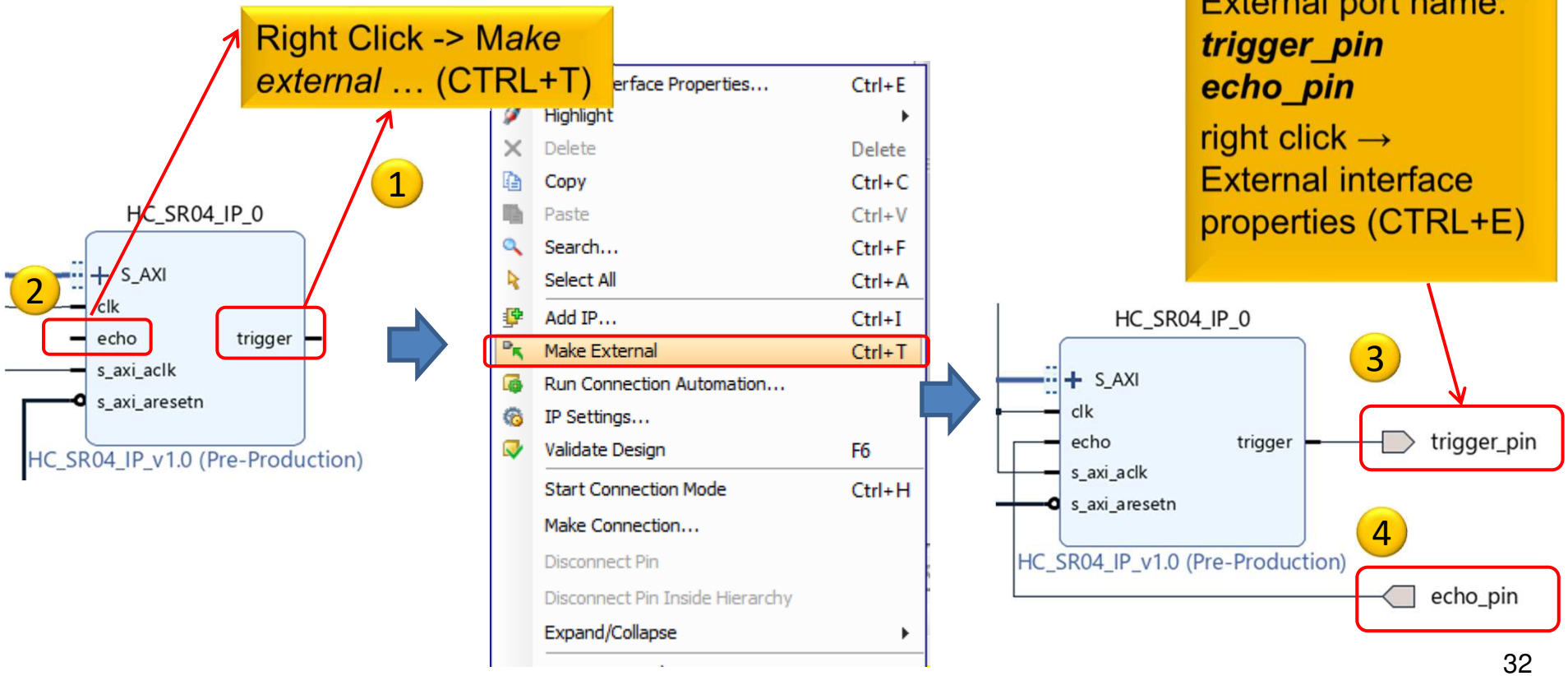
# HC_SR04_IP – Assign external ports

*HC_SR04_IP_0* must be connected to the FPGA pins on the ZyBo card:

1.) The data ports of the HC_SR04_IP instance must be connected to external physical FPGA pins,

2.) If necessary, define the names of the external ports (e.g. **`trigger_pin`**, and **`echo_pin`**), then

3.) In the <system>.XDC file, the pin of the FPGA must be specified.

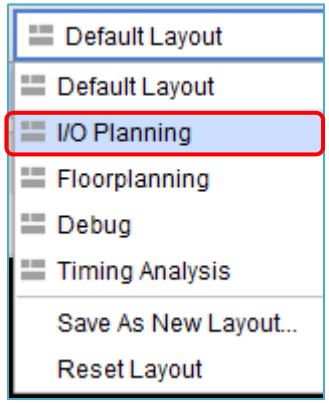


# Block Design – Layout synthesis

- Refresh the Block Design:
  - Regenerate Layout
  - Validate Design (DRC)

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

- Final step, assign `trigger_pin, echo_pin` to FPGA IO pins!
  - Layout menu ⟶ IO planning layout view



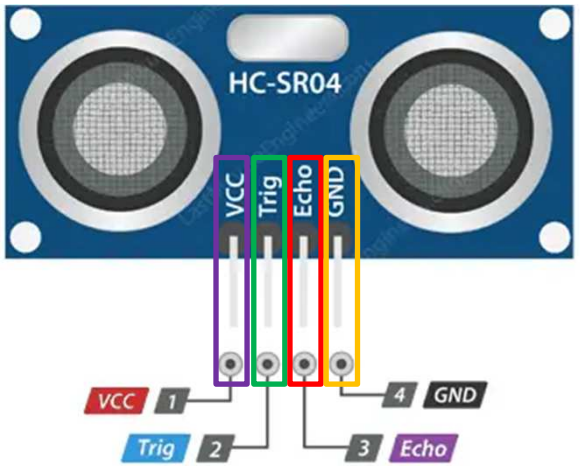
# ZyBo - PMOD connectors

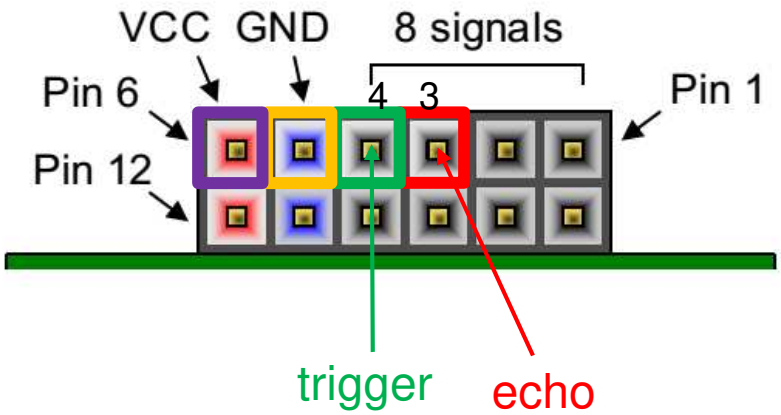
| Pmod JA (XADC) | Pmod JB (Hi-Speed) | Pmod JC (Hi-Speed) | Pmod JD (Hi-Speed) | Pmod JE (Std.) | Pmod JF (MIO) |
|---|---|---|---|---|---|
| JA1: N15 | JB1: T20 | JC1: V15 | JD1: T14 | JE1: V12 | JF1: MIO-13 |
| JA2: L14 | JB2: U20 | JC2: W15 | JD2: T15 | JE2: W16 | JF2: MIO-10 |
| JA3: K16 | JB3: 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:

echo: **J15**

trigger: **H15**




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

# IO planning – pin assignments

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

proper pins assignments based on *Zybo_master.xdc:*

- Package Pin:
  - echo_pin[0]: J15 (as PMOD JE[3])
  - trigger_pin[1]: H15 (as PMOD JE[4])
- IOSTANDARD: LVCMOS33
- OffChipTermination (OCT): NONE

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

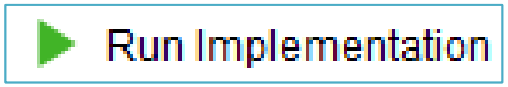
| Name | Direction | Package Pin | Fixed | Bank | I/O Std | Vcco | Vref | Drive Strength | Slew Type | Pull Type | Off-Chip Termina |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DDR_12642 (71) | INOUT | | ✓ | 502 | (Multiple)* | 1.500 | (Multiple) | | (Multiple) | NONE | FP_VTT_50 |
| dip_pin_12642 (4) | IN | | ☑ | (Multiple) | LVCMOS33* | 3.300 | | | | NONE | NONE |
| FIXED_IO_12642 (59) | INOUT | | ✓ | (Multiple) | (Multiple)* | (Multiple) | (Multiple) | (Multiple) | (Multiple) | NONE | (Multiple) |
| pb_pin_12642 (4) | IN | | ☑ | 34 | LVCMOS33* | 3.300 | | | | NONE | NONE |
| Scalar ports (2) | | | | | | | | | | | |
| echo_pin | IN | J15 | ☑ | 35 | LVCMOS33* | 3.300 | | | | NONE | NONE |
| trigger_pin | OUT | H15 | ☑ | 35 | LVCMOS33* | 3.300 | | 12 | | NONE | NONE |

1 Package Pin  I/O Std 2  OCT 3

***File*** → ***Save Constraints*** or **CTRL+S**. Then, save the XDC file as: „`lab05.xdc`"

# Implementation and Bitstream generation

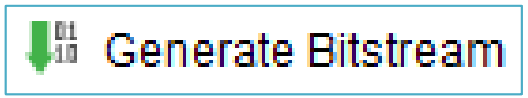
- Flow Navigator menu → **Run Implementation**




  - 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

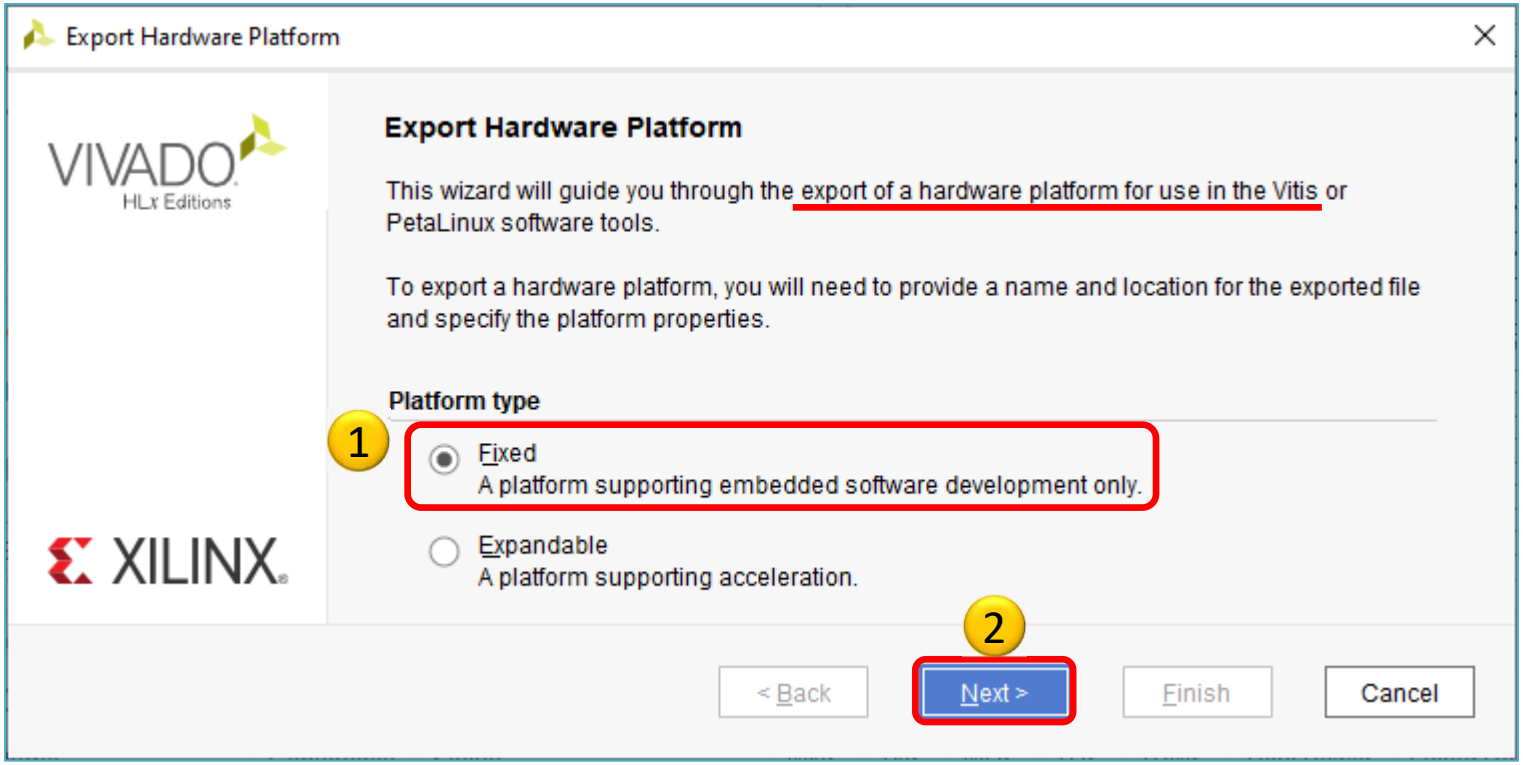
- Question-1.) how many resources are occupied on PL?
- Solution: Reports → Report Utilization (or Project Summary Σ )

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| Slice LUTs | 691 | 0 | 17600 | 3.93 |
| LUT as Logic | 629 | 0 | 17600 | 3.57 |
| LUT as Memory | 62 | 0 | 6000 | 1.03 |
| LUT as Distributed RAM | 0 | 0 | | |
| LUT as Shift Register | 62 | 0 | | |
| Slice Registers | 1010 | 0 | 35200 | 2.87 |
| Register as Flip Flop | 1010 | 0 | 35200 | 2.87 |

# VIVADO Export HW → VITIS (~SDK)

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

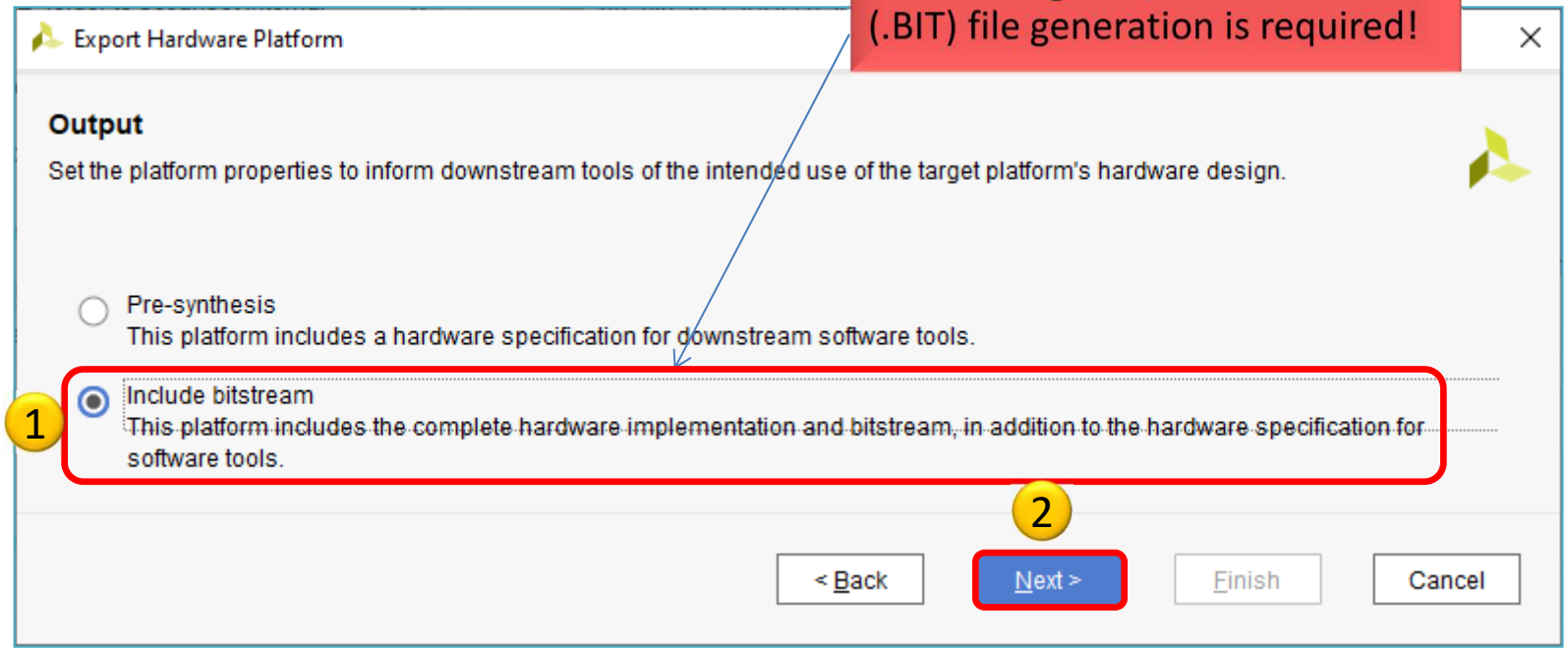
2020.x: at least an Implemented Design must be able to be exported to HW!



# VIVADO Export HW → VITIS (cont.)

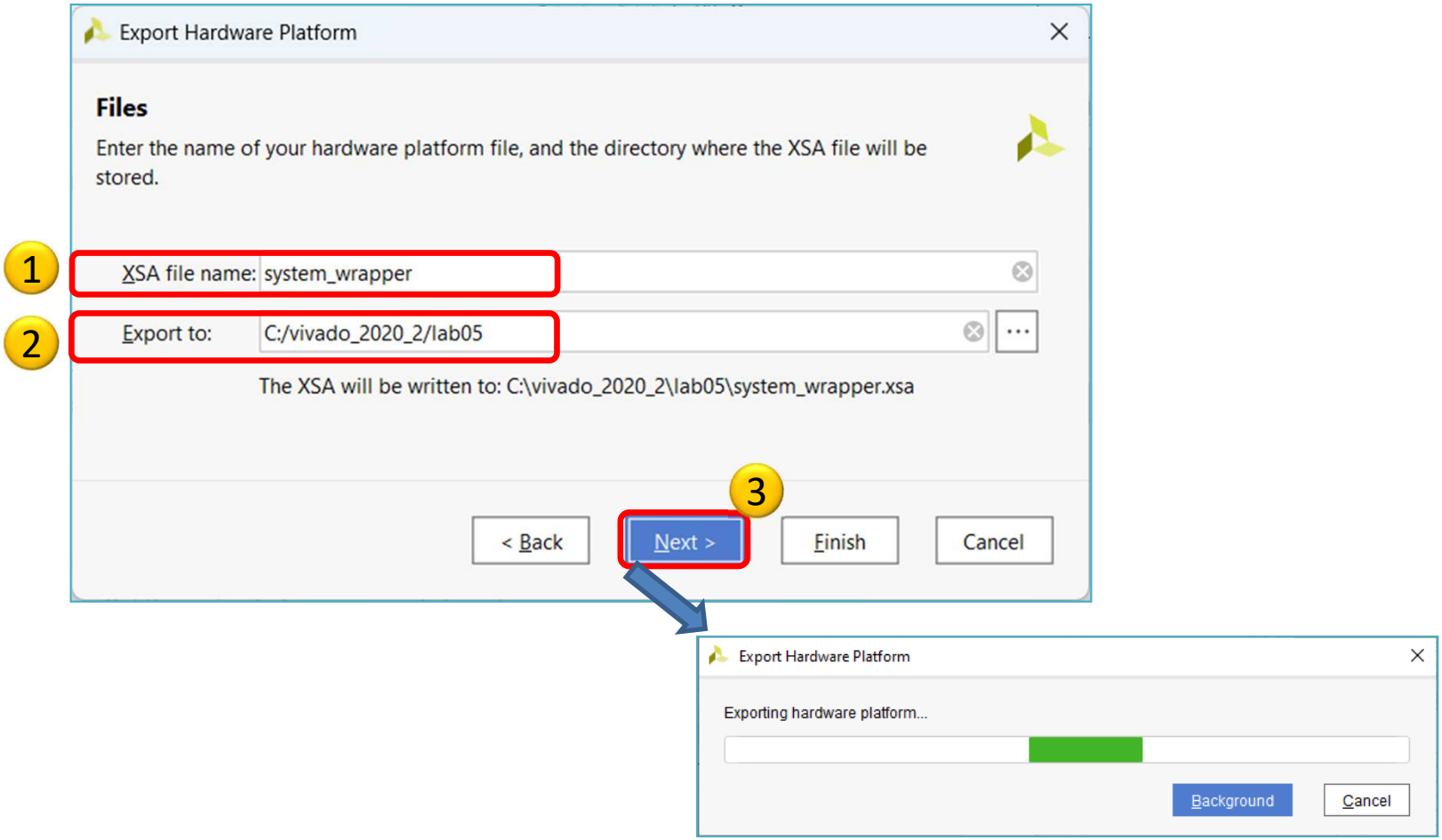
## Select „Include bitstream" option as output:

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



# Export HW → VITIS (cont.)

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




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

# USING XILINX VITIS

LAB05. Creating a software test application for HC_SR04 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. *TestSonar* as SW application):
   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

Xilinx Vitis 2020.1

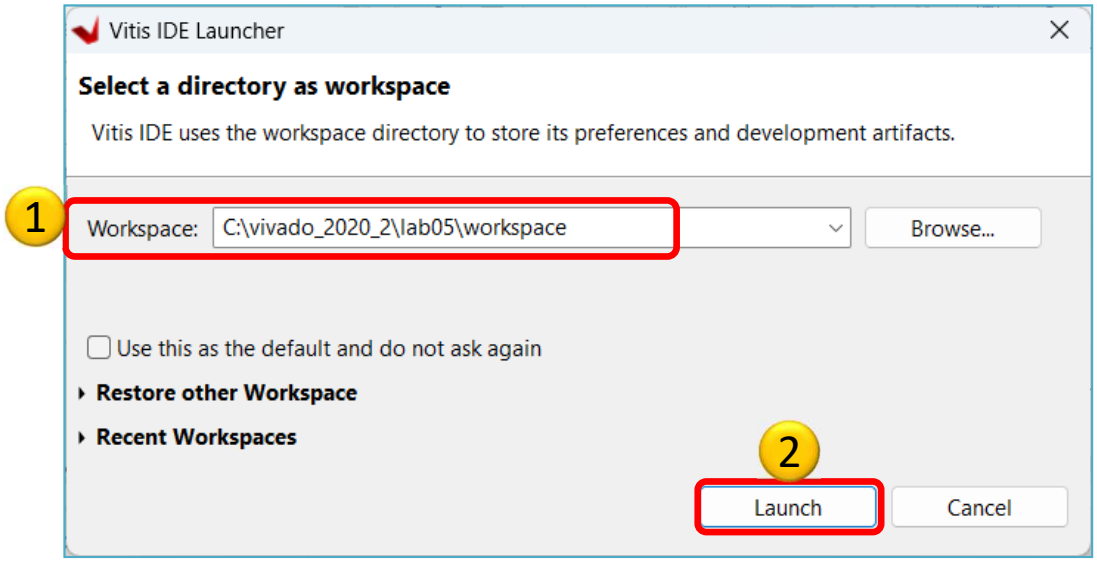
**From Vivado: Tools menu → Launch VITIS IDE**

OR externally

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

Do Not run Xilinx VITIS HLS 2020.2 !

- Set workspace directory properly (`lab05`):
  - Recommended to use `vitis_workspace` as a subdirectory in your lab folder. Launch it...

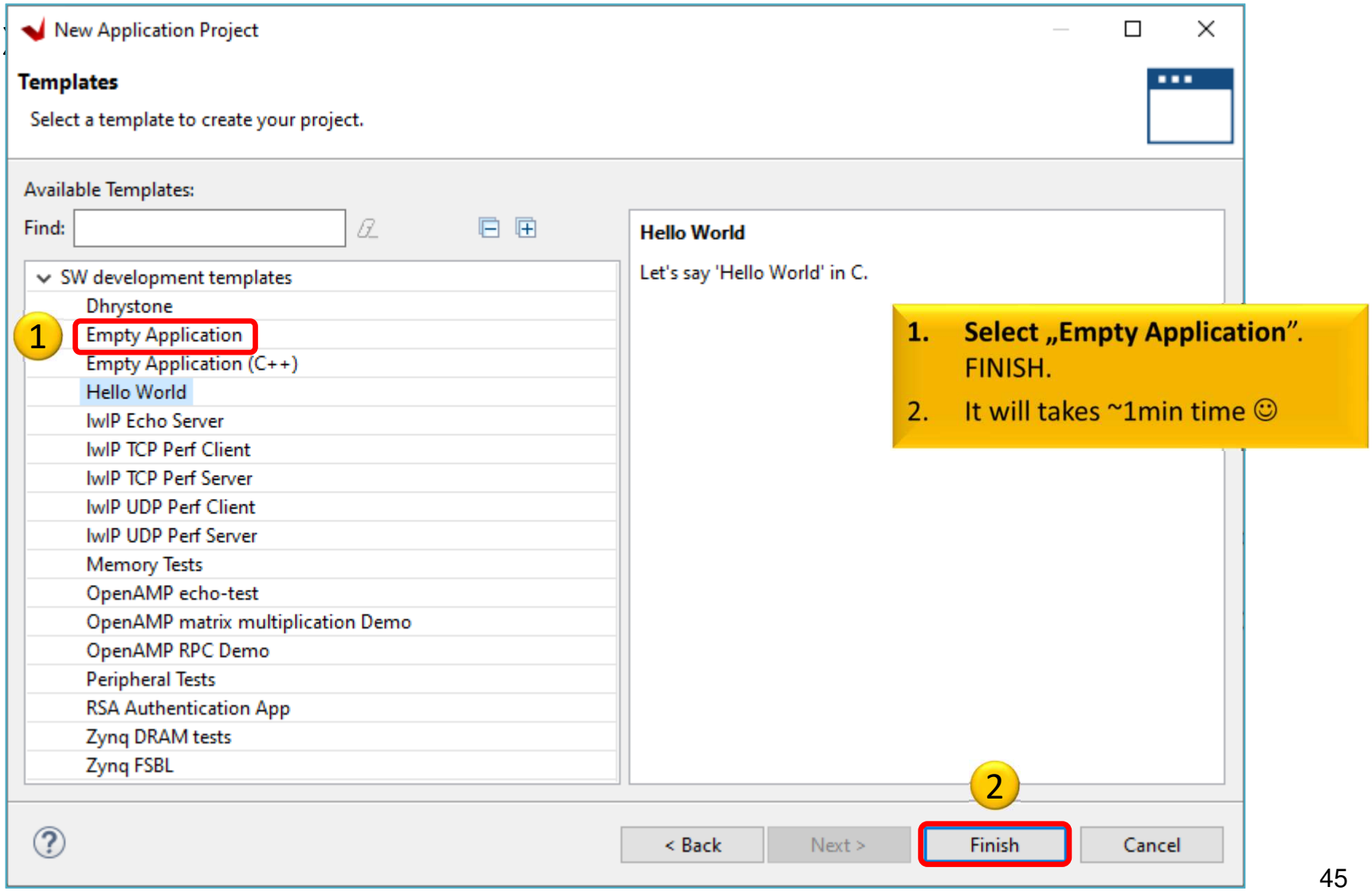


# Xilinx VITIS – Create Application

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 LAB05 `system_wrapper.xsa`. Open it.
   - ! Do not select the „*Generate boot components*"
3. **Application project details**
   - Type „`TestSonar`" as project name
   - Type „ `TestSonar_system`" as system project name
   - Select `ps7_cortexa9_0` as target ARM core 0
4. **Domain:** leave settings as default (standalone)

# Example I.) Creating TestSonar as empty application

- 

# VITIS GUI – Main window (HW)

workspace - system_wrapper/platform.spr - Vitis IDE

File Edit Search Xilinx Project Window Help

Explorer

- system_wrapper
  - export
  - hw
  - logs
  - ps7_cortexa9_0
  - resources
  - platform.spr
  - platform.tcl
- Test_Sonar_system [ system_wrapper ]
  - Test_Sonar [ standalone_ps7_cortexa9_0 ]
    - Binaries
    - Includes
    - Debug
    - src
    - _ide
    - Test_Sonar.prj
  - Test_Sonar_system.sprj

1

Assistant

- Test_Sonar_system [System]
  - Test_Sonar [Application]
    - Debug
    - Release
  - Debug
  - Release
- system_wrapper [Platform]

lab05.c | HC_SR04_IP.h | system_wrapper | Test_Sonar

## Hardware Platform Specification

### Design Information

Target FPGA Device: 7z010
Part: xc7z010clg400-1
Created With: Vivado 2020.2
Created On: Wed Apr 24 00:26:54 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 |
|---|---|---|---|---|
| ps7_ram_0 | 0x00000000 | 0x0002ffff | - | memory |
| ps7_ddr_0 | 0x00100000 | 0x1fffffff | - | memory |
| dip | 0x41200000 | 0x4120ffff | S_AXI | register |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| HC_SR04_IP_0 | 0x41230000 | 0x4123ffff | 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 |

2

**HC_SR04_0:** address map and specification of your custom IP core

Main | Hardware Specification

Outline

There is no active editor that provides an outline.

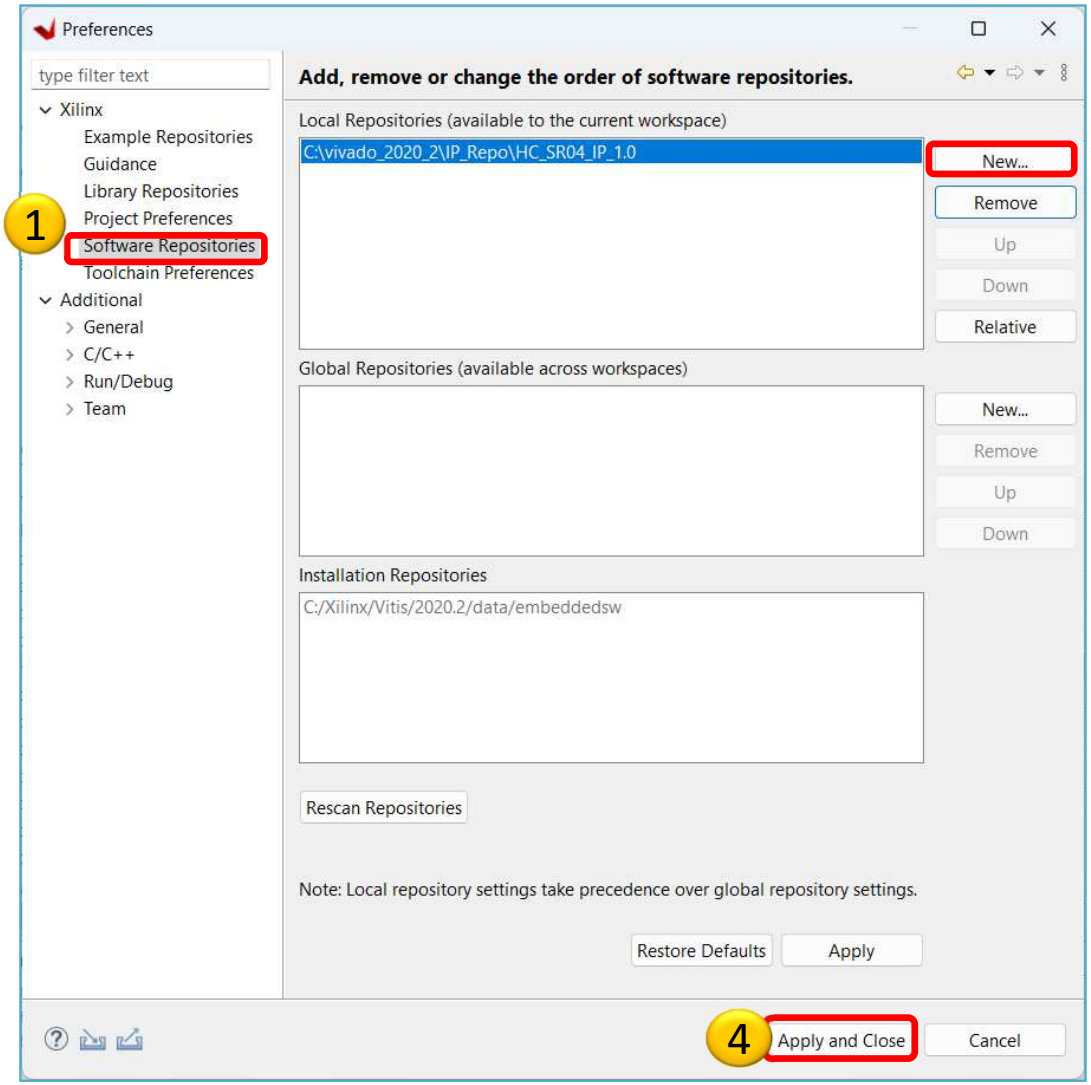
Console | Problems | Vitis Log | Guidance

Platform Tcl Console

```
platform read {C:\vivado_2020_2\lab05\workspace\system_wrapper\platform.spr}
platform active {system_wrapper}
```

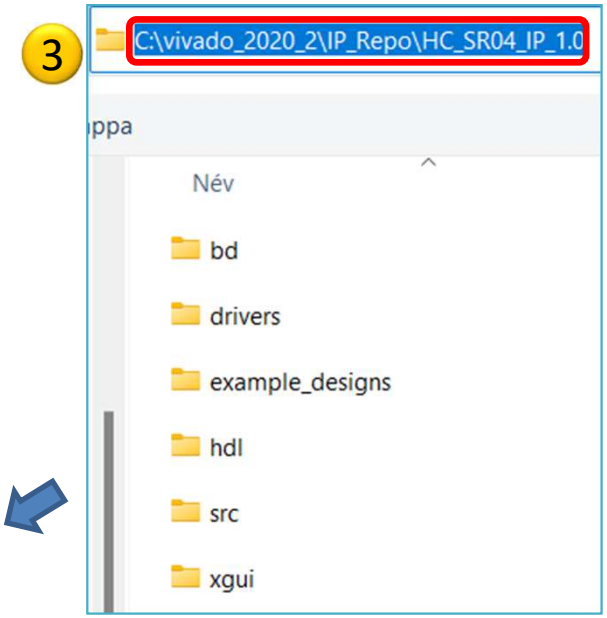
# VITIS – Add Driver Repository

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>\HC_SR04_IP_1.0`).



# VITIS – Main window (SW-driver)

workspace - system_wrapper/platform.spr - Vitis IDE

File Edit Search Xilinx Project Window Help

Explorer

- system_wrapper (Out-of-date)
  - export
  - hw
  - logs
  - ps7_cortexa9_0
  - platform.spr
  - platform.tcl
- Test_Sonar_system [ system_wrapper ]
  - Test_Sonar [ standalone_ps7_cortexa9_0 ]
    - Binaries
    - Includes
    - Debug
    - src
      - lab05.c
      - lscript.ld
      - README.txt
      - Xilinx.spec

1

Assistant

- Test_Sonar_system [System]
  - Test_Sonar [Application]
    - Debug
    - Release
  - Debug
  - Release
- system_wrapper [Platform]

system_wrapper

type filter text

- system_wrapper (Out-of-date)
  - ps7_cortexa9_0
    - standalone_ps7_cortexa9_0
      - Board Support Package

## 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 exising 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: Standalone is a simple, low-level software layer. It provides access to basic processor features such as 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

Drivers Libraries

| Name | Driver | Documentation | Examples |
|---|---|---|---|
| HC_SR04_IP_0 | HC_SR04_IP | - | - |
| dip | gpio | - | Import Examples |
| pb | gpio | | Import Examples |
| ps7_afi_0 | generic | | |
| ps7_afi_1 | generic | | |
| ps7_afi_2 | generic | - | |
| ps7_afi_3 | generic | - | |

2

Main Hardware Specification

Outline

There is no active editor that provides an outline.

Console Problems Vitis Log Guidance

Platform Tcl Console

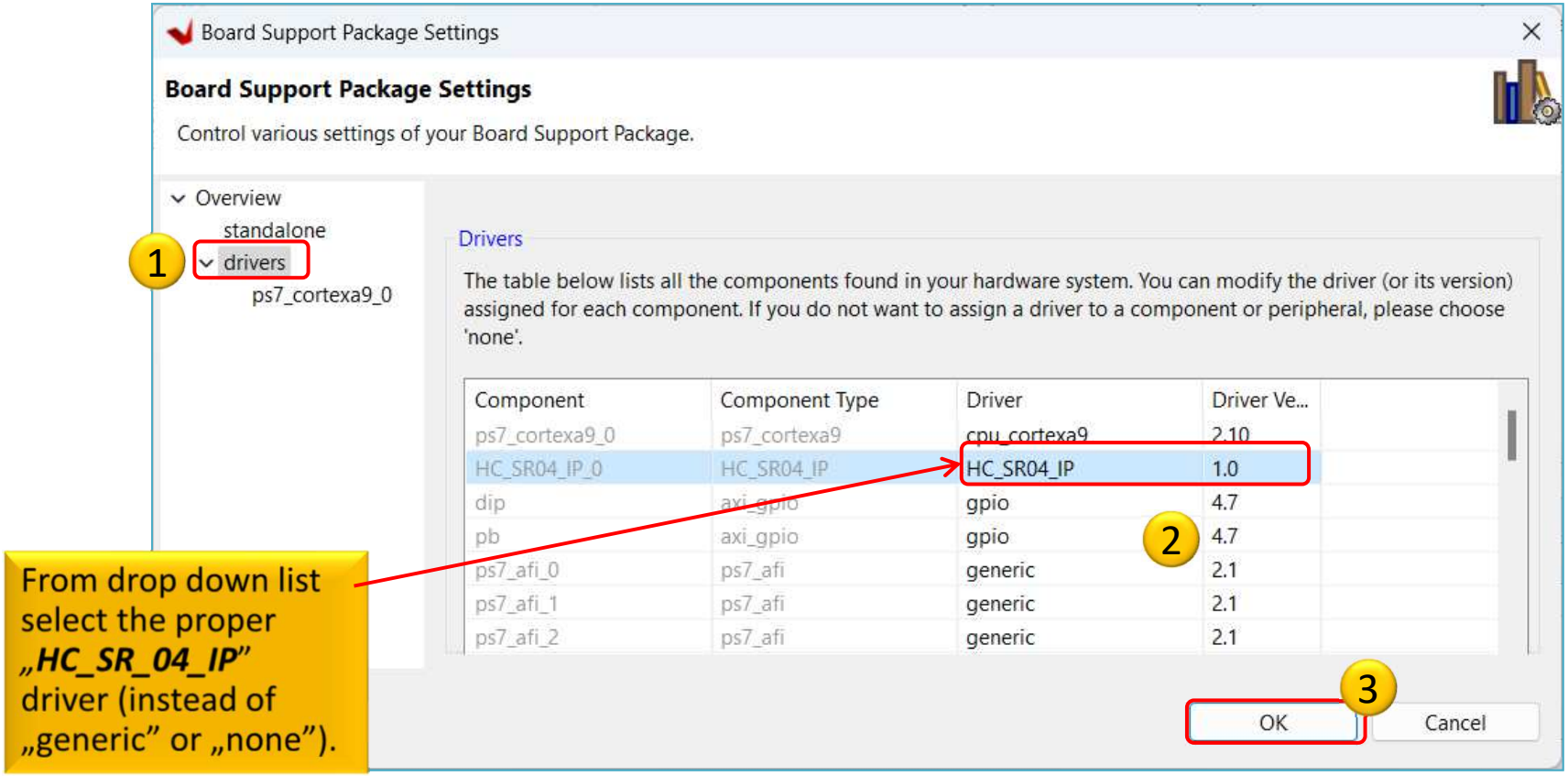
```
platform read {C:\vivado_2020_2\lab05\workspace\system_wrapper\platform.spr}
::scw::get_hw_path
::scw::regenerate_psinit C:/vivado_2020_2/lab05/workspace/system_wrapper/hw/system_wrapper.xsa
```

**CH_SR04_IP_0:** check the driver name (`HC_SR04_IP`)

**If you see "generic„ or „none" , then the HP_SR04_IP is not yet assigned to the correct driver!**

# VITIS – Set LED_IP driver

- **Project Explorer → Right Click** `TestSonar's` **→ Board Support Package Settings**



# VITIS – SW project

- **Project Explorer** → double click on **`lab5.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_HC_SR04_IP_0_S_AXI_BASEADDR`
- This macro defines our „**HC_SR04_IP**” custom peripheral
- This #define can be used to read from Ultrasonic sensor

# HC_SR04_IP drivers

- Path *:*
  - `<lab05_project>\system_wrapper\hw\drivers\HC_SR04_IP_v1_0\src`
- Investigate the content of .c, and .h source files (generated from Vivado tool)!
- Reading from the Ultrasonic Sensor:

```
#define HC_SR04_IP_mReadReg(BaseAddress, RegOffset) \
    Xil_In32((BaseAddress) + (RegOffset))
```

# Analyzing LED_IP application

- 1.) Read the distance from ultrasonic sensor (in an infinte loop)

```c
#include <stdio.h>
#include "xil_printf.h"
#include "xil_io.h"
#include "xparameters.h"
#include "HC_SR04_IP.h"

int main()
{
    u32 distance;
    volatile unsigned int i;
    xil_printf("lab05 - HC_SR04 Ultrasonic sensor program started...\r\n");
    while(1) {
        //#define HC_SR04_IP_mReadReg(BaseAddress, RegOffset) \
        Xil_In32((BaseAddress) + (RegOffset))
        distance = HC_SR04_IP_mReadReg(XPAR_HC_SR04_IP_0_S_AXI_BASEADDR, HC_SR04_IP_S_AXI_SLV_REG0_OFFSET);

        xil_printf("Distance: %lu [cm]\r\n", distance);
        for(i = 0; i < 100000000; i++); //delay
    }

    return 0;
}
```

IP header file

Read distance from the ultrasonic sensor.

# Important Remark* - Makefile

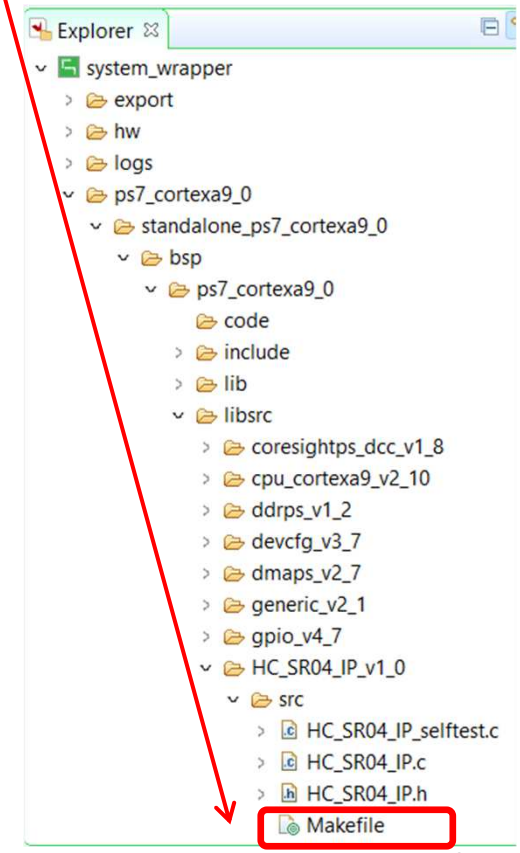
*There is a build problem with VITIS 2020.x when creating a custom AXI-lite based IP. Makefile generation did not work properly (build error).

1. Open `system_wrapper\ps7_cortexa9_0\standalone_ps7_cortexa9_0\bsp\ps7_cortexa9_0\libsrc\HC_SR04_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!




https://support.xilinx.com/s/question/0D52E00006hpOx5SAE/drivers-and-makefiles-problems-in-vitis-20202

# Generate Linker Script & Build

- Generate Linker Script to the internal on-chip PS7 **RAM0**
  - Set the Heap / Stack size to **1KB**! 
  - Now rebuild the TestSonar again! 

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

```
'Invoking: ARM v7 Print Size'
arm-none-eabi-size Test_Sonar.elf  |tee "Test_Sonar.elf.size"
   text    data     bss     dec     hexfilename
  21780    1144    8232   31156    79b4 Test_Sonar.elf
'Finished building: Test_Sonar.elf.size'
```

# TestSonar – Verification result

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

```
lab05 - HC_SR04 Ultrasonic sensor program
started...
Distance: 191 [cm]
Distance: 191 [cm]
Distance: 191 [cm]
Distance: 2 [cm]
Distance: 6 [cm]
Distance: 2 [cm]
Distance: 2 [cm]
Distance: 4 [cm]
Distance: 5 [cm]
Distance: 9 [cm]
Distance: 9 [cm]
Distance: 11 [cm]……
```