



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](mailto:voroshazi.zsolt@mik.uni-pannon.hu)

Updated: 10 Apr. 2024.

**SZÉCHENYI** 2020



MAGYARORSZÁG  
KORMÁNYA

**Európai Unió**  
Európai Strukturális  
és Beruházási Alapok



**BEFEKTETÉS A JÖVŐBE**

# 8. VIVADO – EMBEDDED SYSTEM

Adding peripherals to BSB from IP Catalog #3 (XADC)

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.
- The directory path should be no longer than 256 characters!

# XILINX VIVADO DESIGN SUITE

Adding IP cores to the Embedded Base System (XADC)

**SZÉCHENYI** 2020



MAGYARORSZÁG  
KORMÁNYA

**Európai Unió**  
Európai Strukturális  
és Beruházási Alapok



**BEFEKTETÉS A JÖVŐBE**

# Task


- Vivado – Block Designer
  - Add **XADC IP peripheral** to the formerly elaborated block design (Embedded Base System) from the IP Catalog,
  - Parameterize IP blocks, set connections, interfaces, address, and external ports (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 ([slides in BER\\_05\\_LAB02A\\_GPIO.pdf](#)) by using the **Xilinx Vivado (IPI)** embedded system designer,
  - LAB02\_A project → Save as... → LAB04 !
- Select and add **XADC (Xilinx ADC)** peripheral as a System monitor 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 „TestXADC” 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**.

# Further references

- XADC - Xilinx Analog-Digital Converter:

 [https://docs.amd.com/r/en-US/ug480\\_7Series\\_XADC/7-Series-FPGAs-and-Zynq-7000-SoC-XADC-Dual-12-Bit-1-MSPS-Analog-to-Digital-Converter-User-Guide-UG480](https://docs.amd.com/r/en-US/ug480_7Series_XADC/7-Series-FPGAs-and-Zynq-7000-SoC-XADC-Dual-12-Bit-1-MSPS-Analog-to-Digital-Converter-User-Guide-UG480)

- Vivado – AXI XADC IP manual :

 [https://www.xilinx.com/products/intellectual-property/axi\\_xadc.html](https://www.xilinx.com/products/intellectual-property/axi_xadc.html)


- **XADC Wizard** (parameters):


 <https://docs.amd.com/v/u/en-US/pg091-xadc-wiz>

- XADC Use-cases:

 <https://docs.amd.com/v/u/en-US/xapp795-driving-xadc>

- **Adam Taylor – MicroZed Chronicles Part #7-8:**

 Part-7: <https://forums.xilinx.com/t5/Xcell-Daily-Blog/Getting-the-XADC-Running-on-the-MicroZed-Adam-Taylor-s-MicroZed/ba-p/380989>

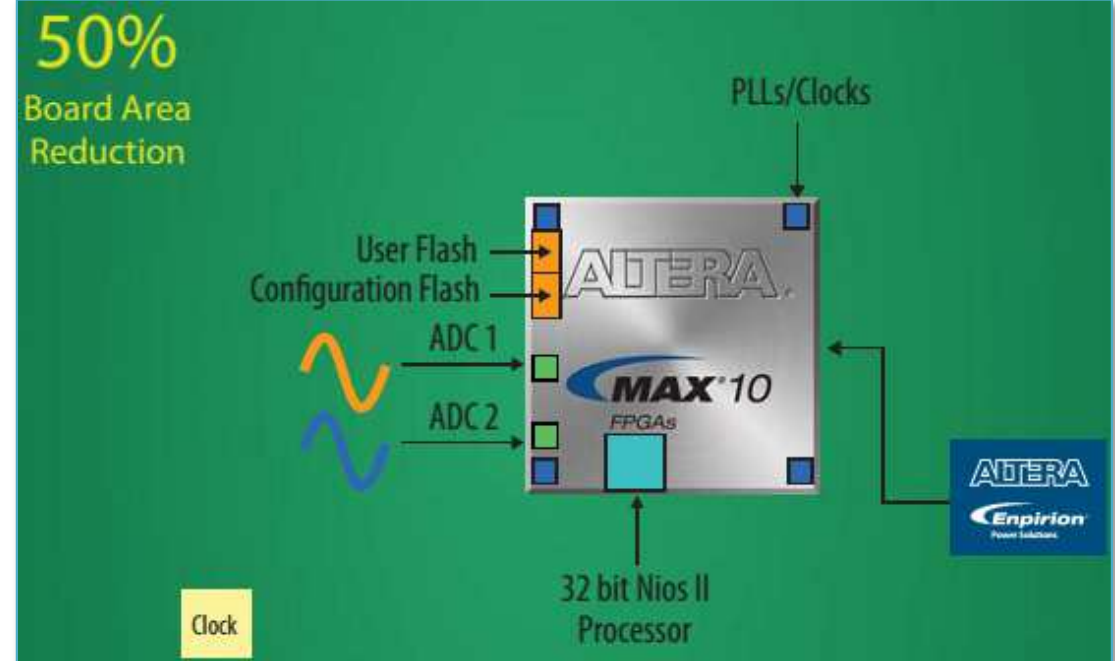
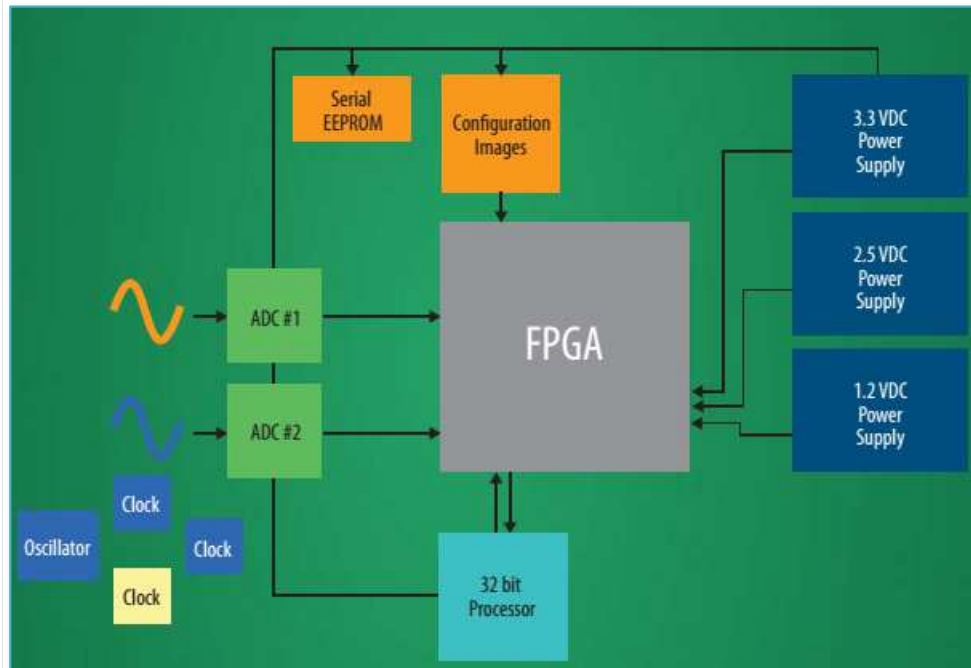
 Part-8: <https://forums.xilinx.com/t5/Xcell-Daily-Blog/MicroZed-XADC-Software-Adam-Taylor-s-MicroZed-Chronicles-Part-8/ba-p/383861>



# ADC – Analog/Digital Converters

- Possible methods to process analog input signals:
  - **External** ADC (outside of FPGA platform):  
**traditionally** this function is on an external expansion card
    - Feasible in all FPGAs
  - **Internal-integrated** ADC (within the FPGA, an „on-chip” function)
    - Only a certain FPGAs, or CPLDs
      - Pl. Xilinx XADC IP core (insided the Xilinx 7 series)

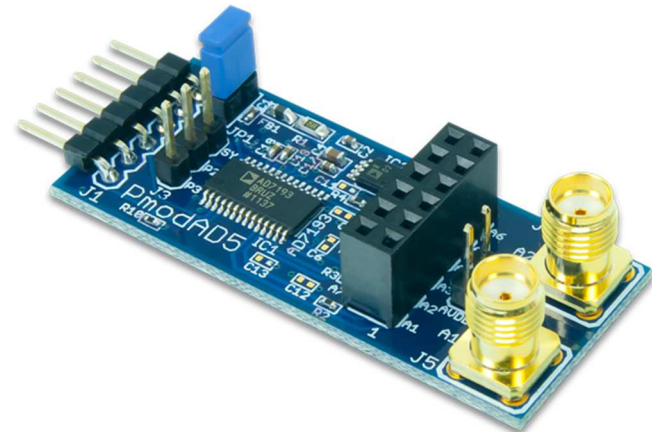
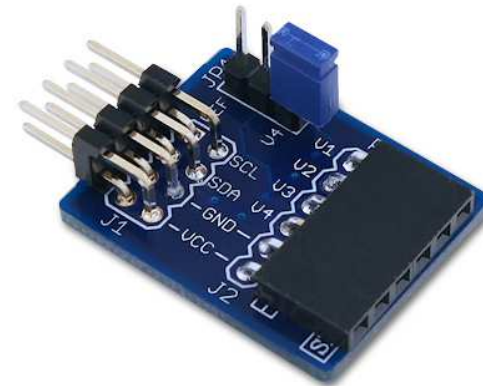
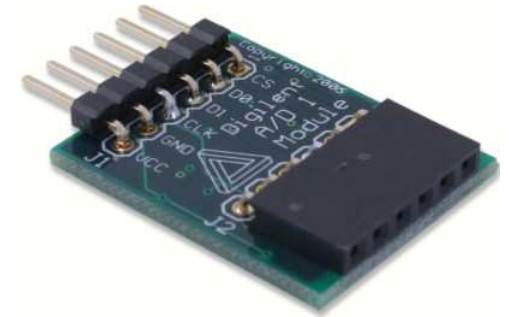
# Traditional vs. Integrated solution



# Traditional solution: ADC card with FPGA

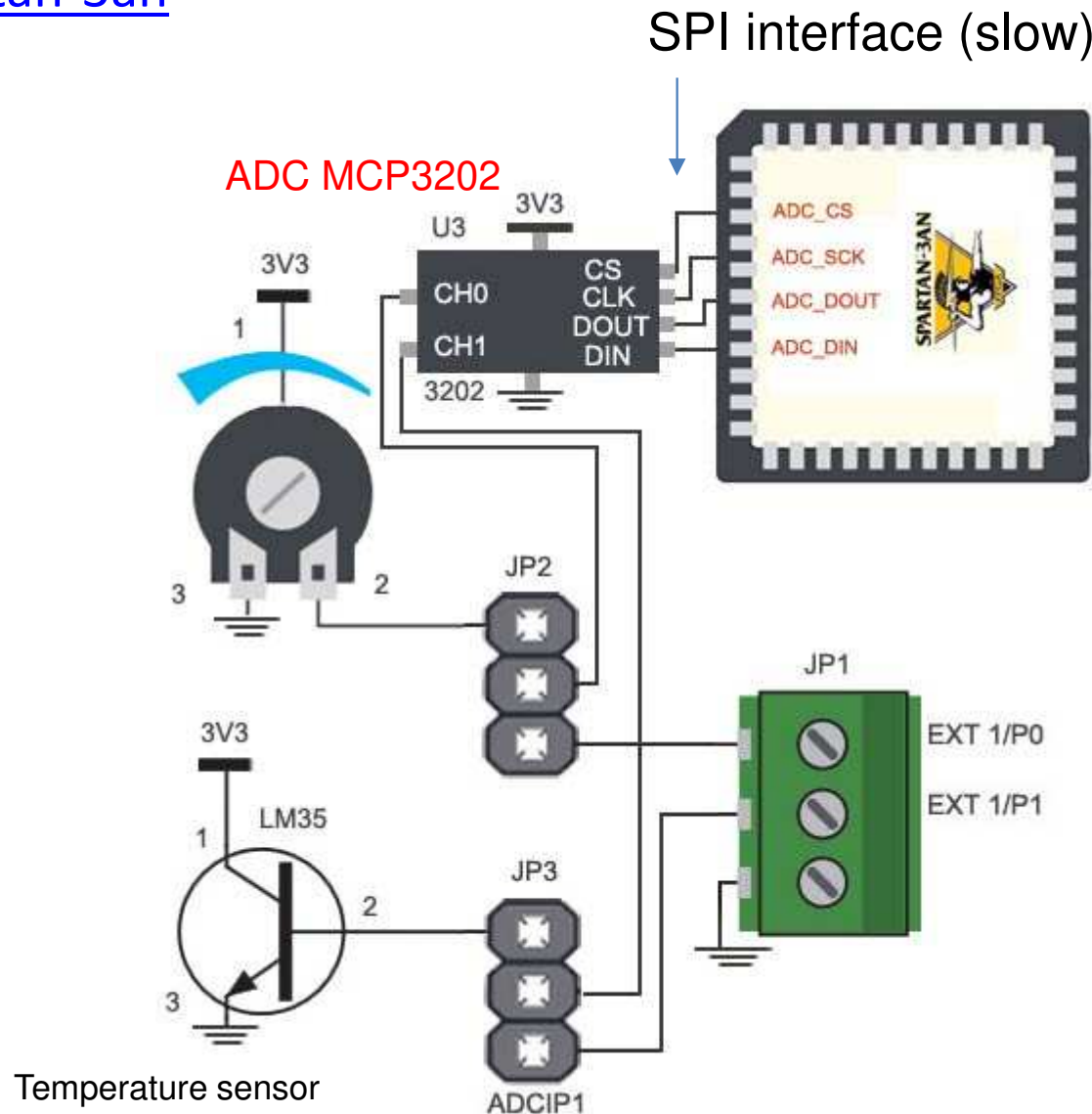
e.g. Digilent PMOD ADCs

- **PDMOD AD1:** 2-channels [Analog Devices AD7476](#) 12-Bit, 1 MSPS, Low-Power A/D Converter
- **PDMOD AD2:** 2 pieces of 4-channels [Analog Devices AD7476](#) 12-Bit, 1 MSPS, Low-Power A/D Converter (+I2C)
- **PDMOD AD5:** [Analog Devices AD7991](#) 4-channels, 24-Bit ADC, 1MSPS

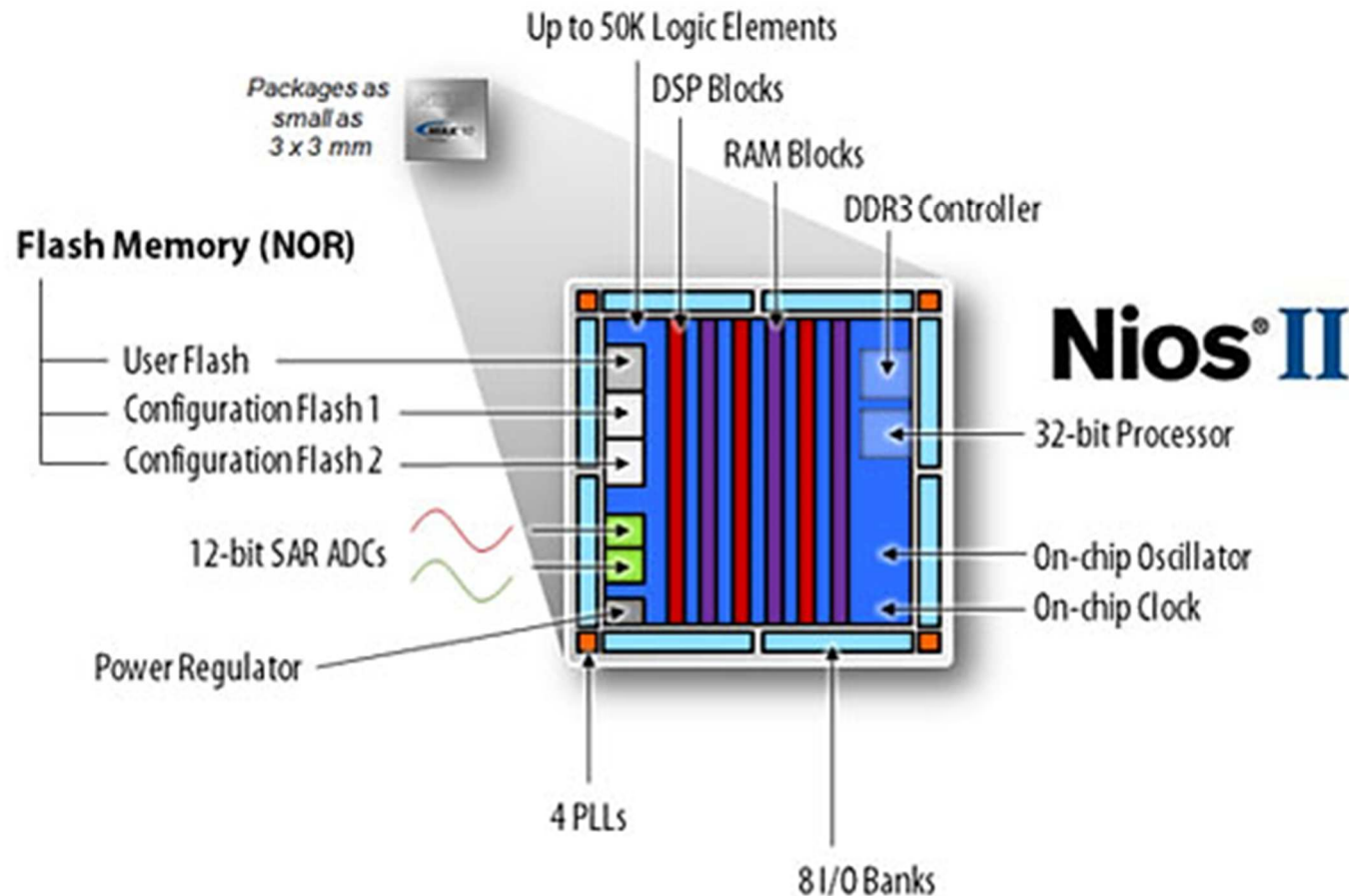


# SPI ADC – Xilinx Spartan3-AN

- 📖 <https://www.pantechsolutions.net/project-kits/how-to-interface-spi-adc-with-spartan-3an>

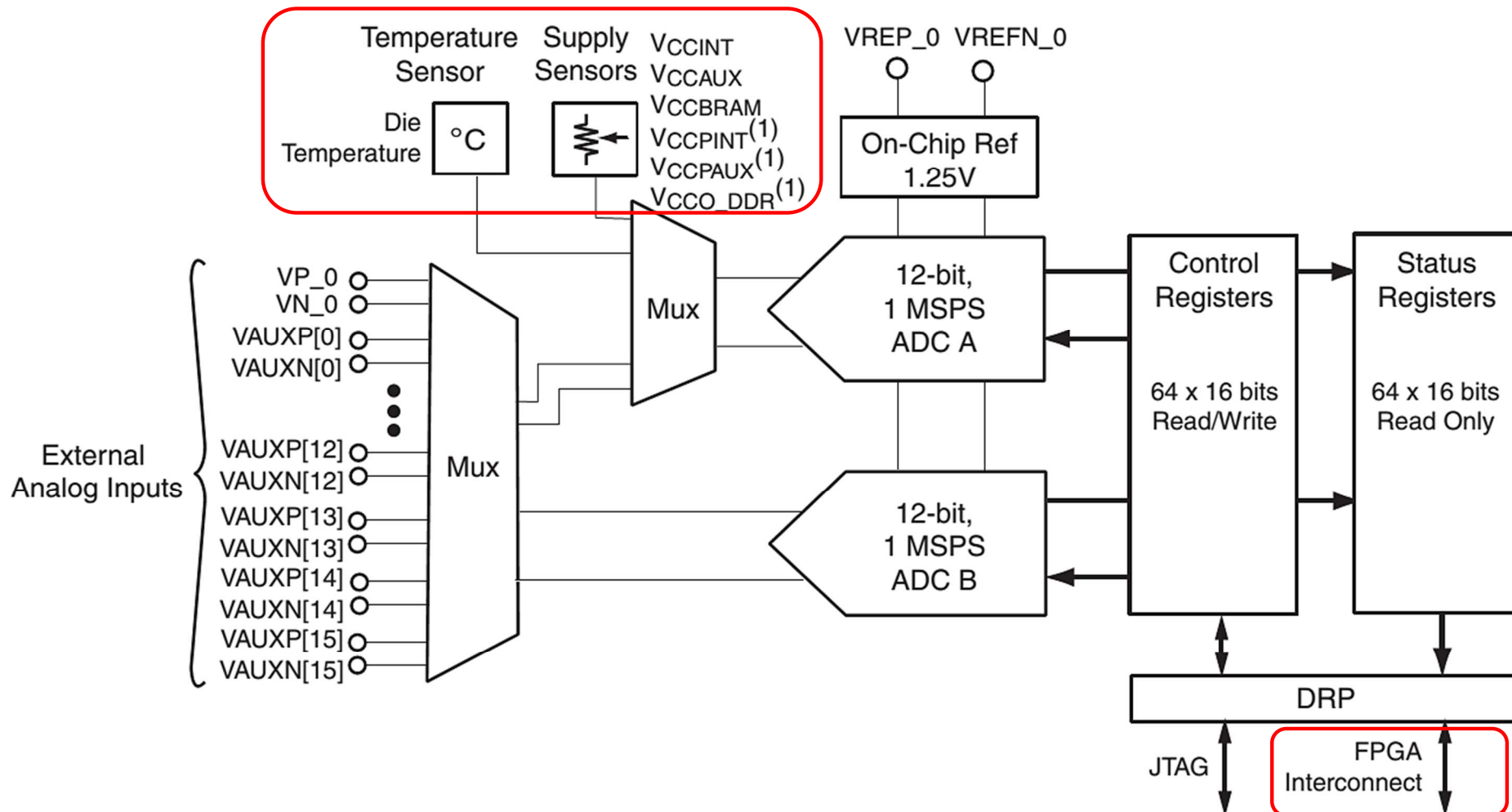


# Intel/Altera MAX 10 „FPGA”



- 2 pieces of embedded SAR ADCs – 12 bit, 1 MSPs
  - Max 18 analog input channels
  - Temperature sensor

# ADC – Analog/Digital Converters



- 1 XADC = 2 pieces of embedded SAR ADCs – 12 bit, 1 MSPs (1%)
  - Max 17 analog input channels
  - Temperature sensor

(Artix-7, Kintex-7, Virtex-7, Zynq 7000)

# XILINX VIVADO DESIGN SUITE

LAB04. Adding Integrated Xilinx ADC (XADC) to the BSB

SZÉCHENYI 2020



MAGYARORSZÁG  
KORMÁNYA

**Európai Unió**  
Európai Strukturális  
és Beruházási Alapok



**BEFEKTETÉS A JÖVŐBE**

# What is an XADC?

- **Xilinx Analog to Digital Converter Block**
  - FPGA / APSoC temperature, voltage level measurement
  - Alarm functions - alarms at adjustable intervals
  - **AXI4-Lite** / Stream interface v. user interface
  - Easy to configure (Wizard ~ GUI)
  - Select / use multiple channels



# XADC – as System Monitor

## What does it measure?

*"Everything about status of FPGA / APSoC."*

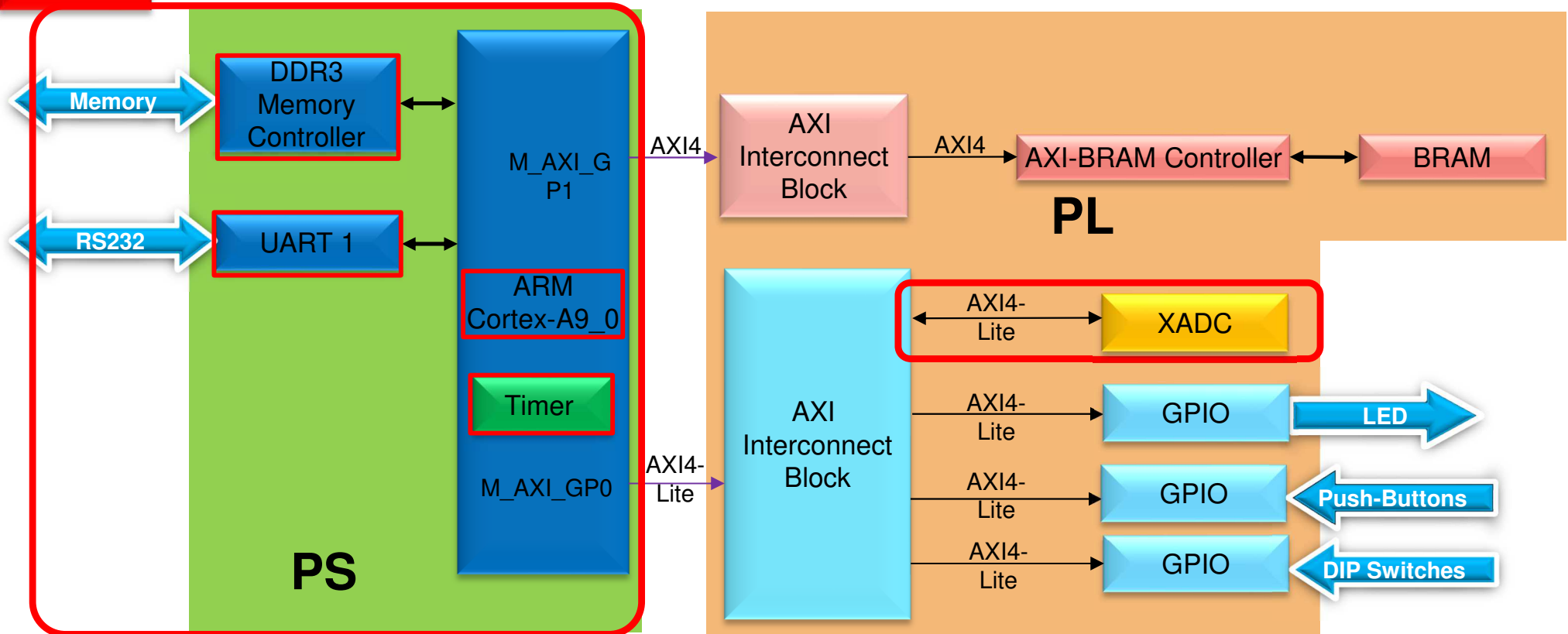
- Temperature: On-chip temperature sensor
- VCC<sub>Int</sub>: The internal PL core voltage
- VCC<sub>Aux</sub>: The auxiliary PL voltage
- VRef<sub>P</sub>: The XADC positive reference voltage
- VRef<sub>N</sub>: The XADC negative reference voltage
- VCC<sub>Bram</sub>: The PL BRAM voltage
- VCCP<sub>Int</sub>: The PS internal core voltage
- VCCP<sub>Aux</sub>: The PS auxiliary voltage
- VCC<sub>Ddr</sub>: The operating voltage of the DDR RAM connected to the PS

# 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... → LAB04**
  - (This will save the former project LAB02\_A as LAB04)

# Test system to be implemented

## LAB04



### PS side:

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

### PL (FPGA)

- LAB04: XADC IP

# Task - Adding XADC

## **Vivado:**

- XADC: integrate and connect it selected from the Vivado IP catalog to the base system ,
- XADC IP instance should be named
- Base Address: 0x4200\_0000 (size: 64 K)
- Examine Block Design and Generate Bitstream

## **VITIS (~SDK):**

- Create a test application (XADCTest) in the VITIS SDK environment,
- Test verification of FW-SW plans on the ZyBo platform.

# Adding XADC to the Base System I.

- Add XADC controller to the Block Diagram (IP Catalog)
  - adding **PL-side** "XADC Wizard"s IP core

IP Catalog – filter to „xadc”

1

2

3

4

Add IP (double click)

Select XADC Wizard

Add IP to Block Design

Customize IP

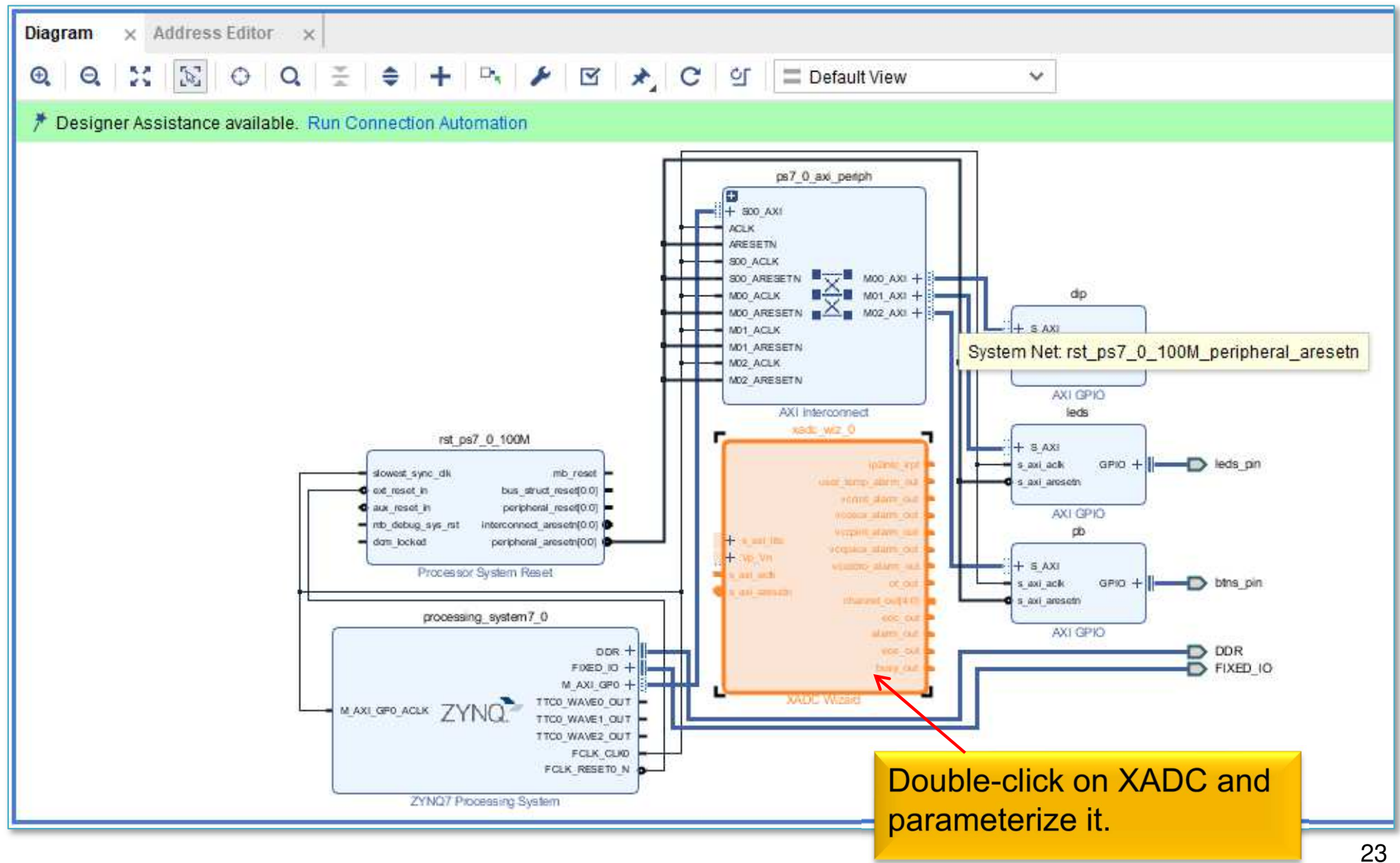
Cancel

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

# Adding XADC to the Base System II.

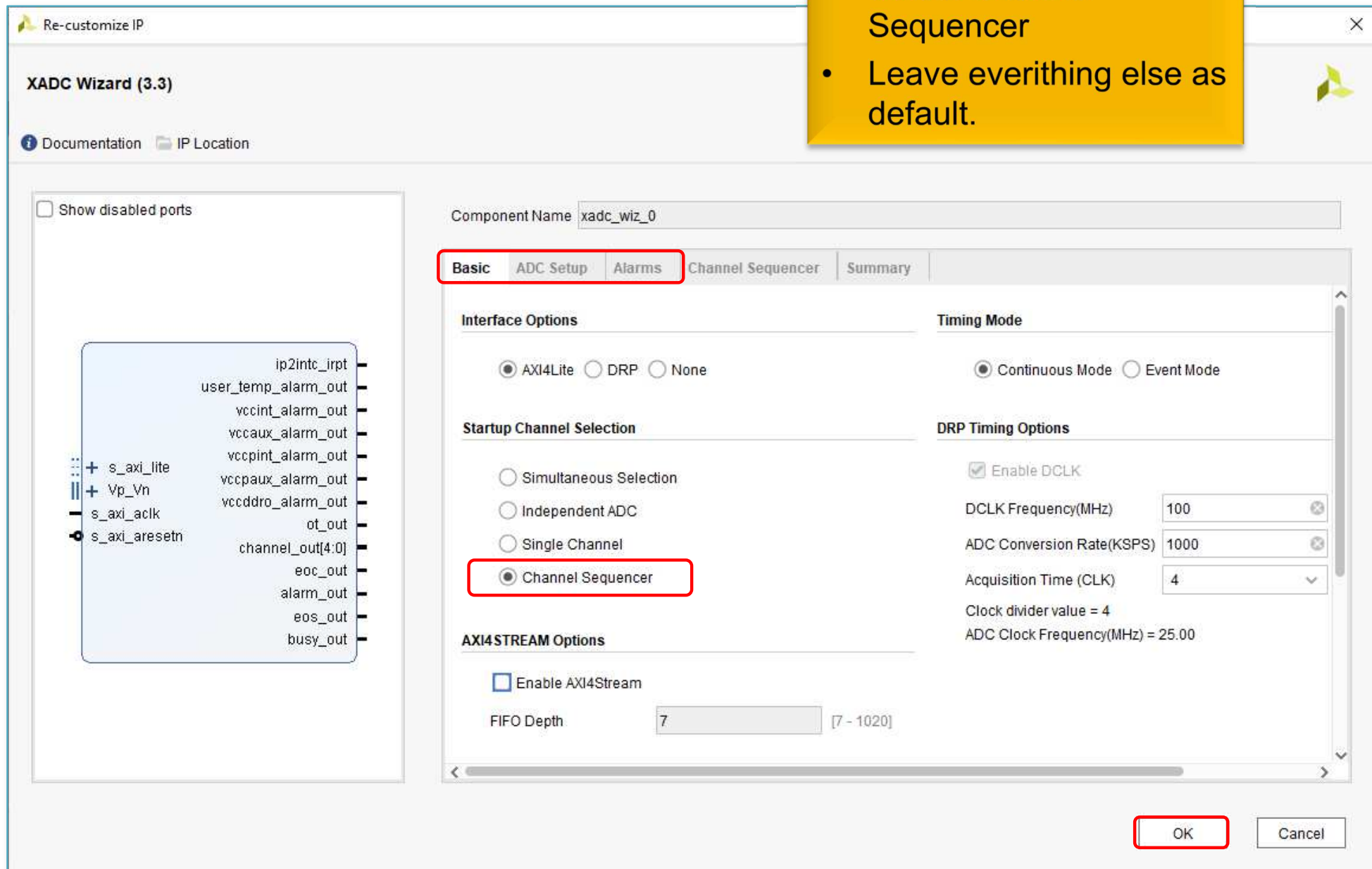
- The following must be set for the XADC IP module in Vivado (it can also be manual / automatic!):
  - a.) interface connection between the IP module and the bus system (AXI Lite),
  - b.) Assigning an IP module to an address range (Base-High Addresses),

# Block Diagram



# XADC Wizard - parameters

- Select Channel Sequencer
- Leave everything else as default.





# Connect XADC (autorouter)

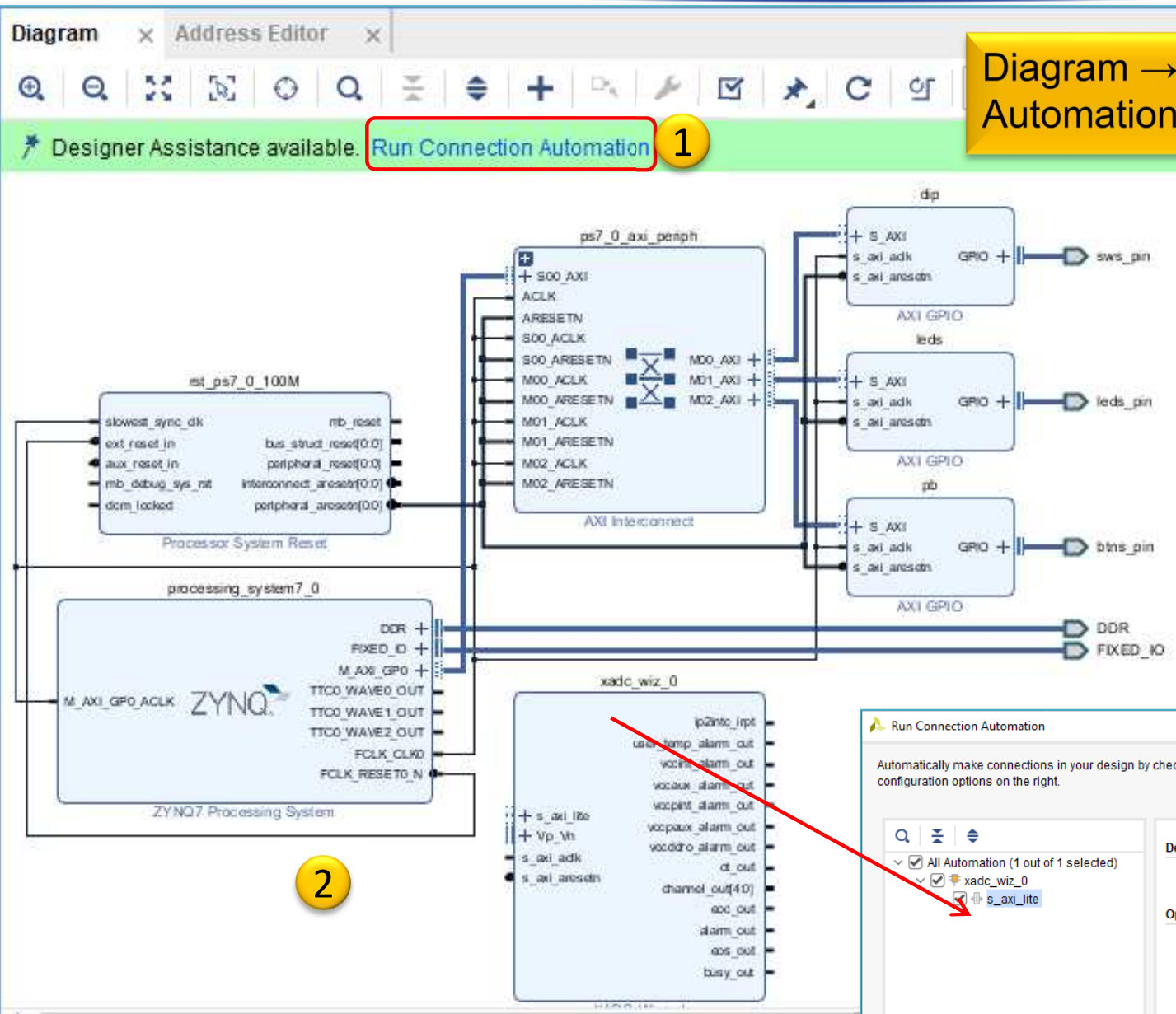
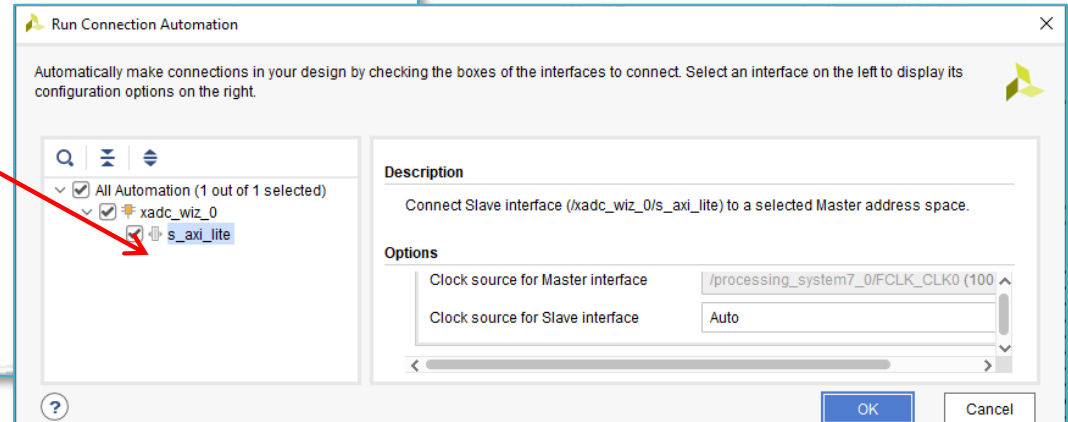


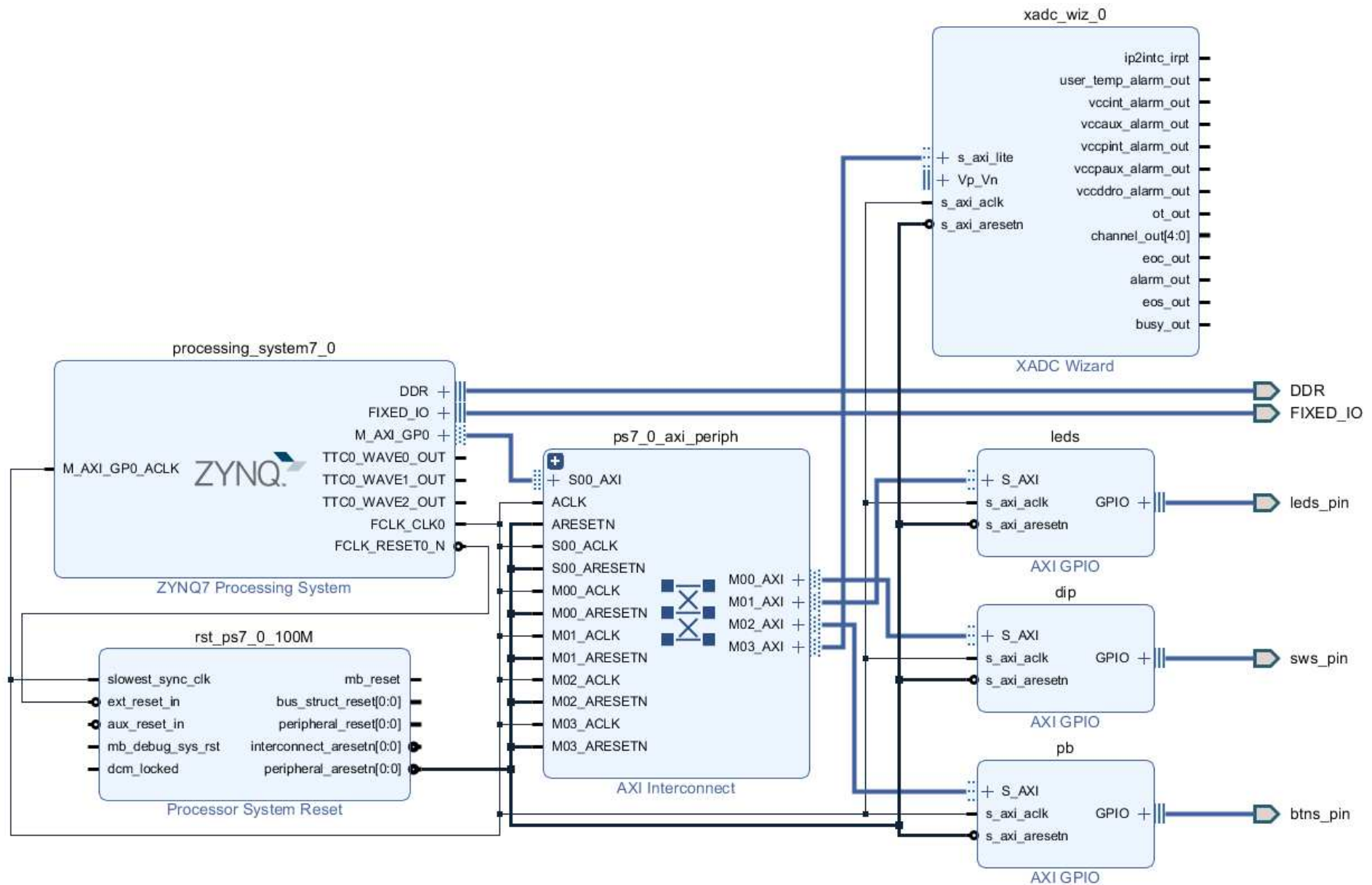
Diagram → Run Connection Automation

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

**xadc\_wiz\_0** →  
S\_AXI\_Lite interface:  
Select S\_AXI\_Lite



# Vivado – Completed design



# XADC – 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

The screenshot shows the 'Address Editor' window in Xilinx IDE. The 'Assigned (4)' tab is active. A tree view on the left shows the hierarchy: Network 0 > /processing\_system7\_0 > /processing\_system7\_0/Data (32 address bits : 0x40000000 [ 1G ]). The main table lists assigned peripherals. The row for '/xadc\_wiz\_0' is highlighted with a red box and a yellow callout '2'. A yellow callout '1' points to the 'Address Editor' tab. Another yellow callout explains that '0X4000\_0000' is assigned to GP0 PS-PL port. A third yellow callout describes automatic address generation.

Name	Interface	Slave	Base Address	Size	Range
<b>/processing_system7_0/Data (32 address bits : 0x40000000 [ 1G ])</b>					
/dip	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
/leds	S_AXI	Reg	0x4122_0000	64K	0x4122_FFFF
/pb	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF
/xadc_wiz_0	s_axi_lite	Reg	0x4200_0000	64K	0x4200_FFFF

**1** Address Editor

**2** 0X4000\_0000 because GP0 PS-PL port has been configured

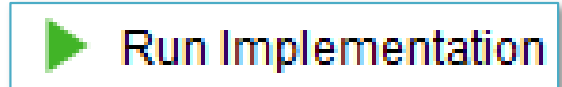
a.) Automatic address generation (right click → Auto Assign Address)

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

b.) Base address manual set\*  
XADC\_wiz\_0: 0x4200\_0000 (64K)

# Implementation and Bitstream generation

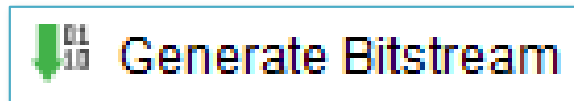
- Flow Navigator menu → **Run Implementation**



- 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**

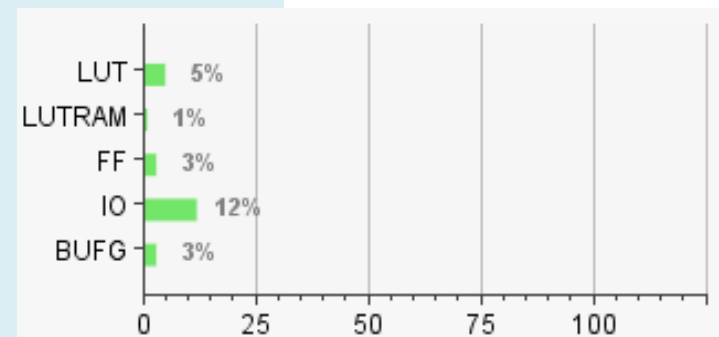


# Q&A 1.) Reports

- How many resources are occupied on PL-side?

Reports → Report Utilization (or Project Summary )

Site Type	Used	Fixed	Available	Util%
Slice LUTs	811	0	17600	4.61
LUT as Logic	749	0	17600	4.26
LUT as Memory	62	0	6000	1.03
LUT as Distributed RAM	0	0		
LUT as Shift Register	62	0		
Slice Registers	1112	0	35200	3.16
Register as Flip Flop	1112	0	35200	3.16
Register as Latch	0	0	35200	0.00
F7 Muxes	0	0	8800	0.00
F8 Muxes	0	0	4400	0.00



...

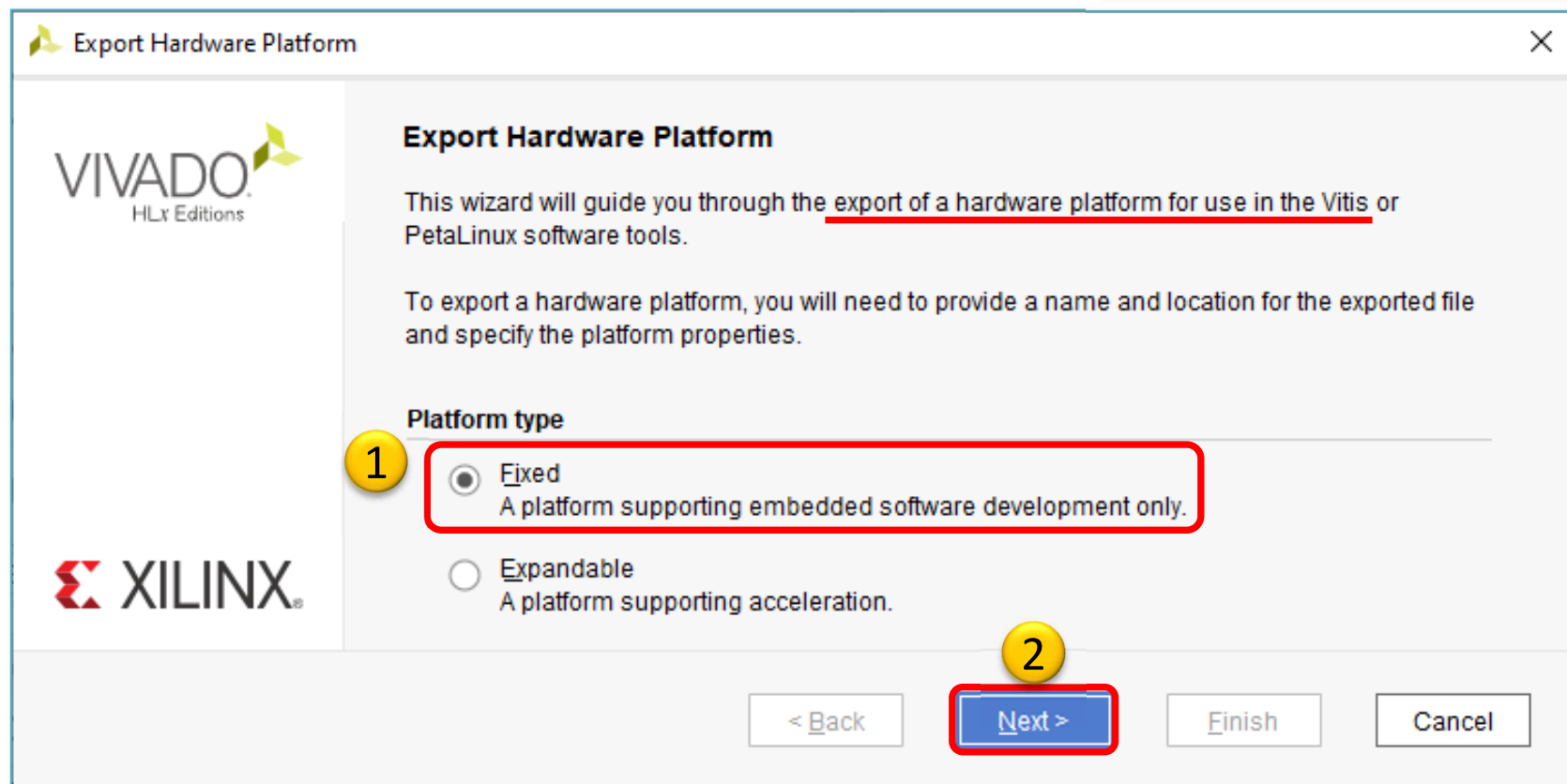
## 7. Specific Feature

Site Type	Used	Fixed	Available	Util%
XADC	1	0	1	100.00

# VIVADO Export HW → VITIS (~SDK)

- File → Export → Export Hardware...

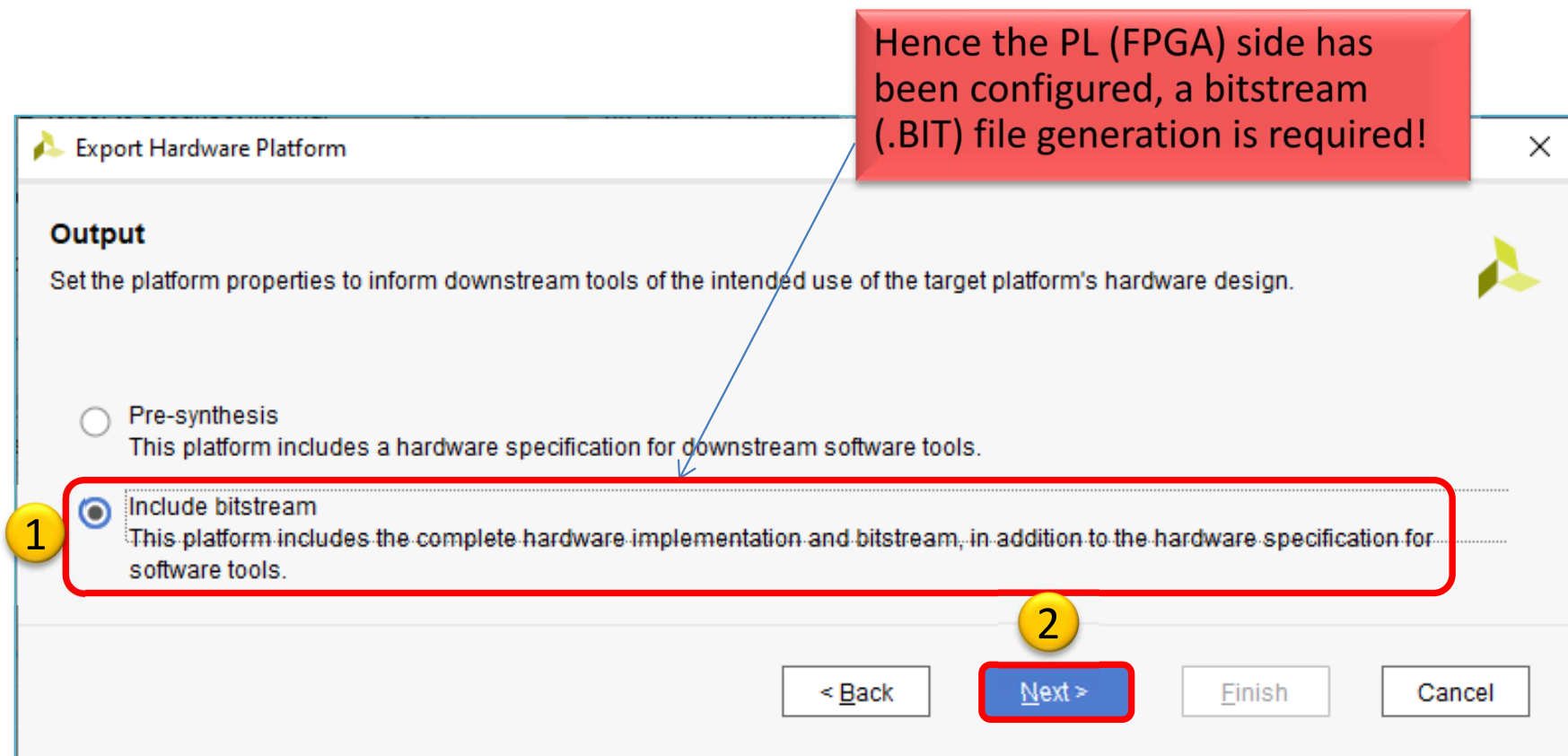
2020.x: at least an Elaborated Design must be able to be exported to HW!





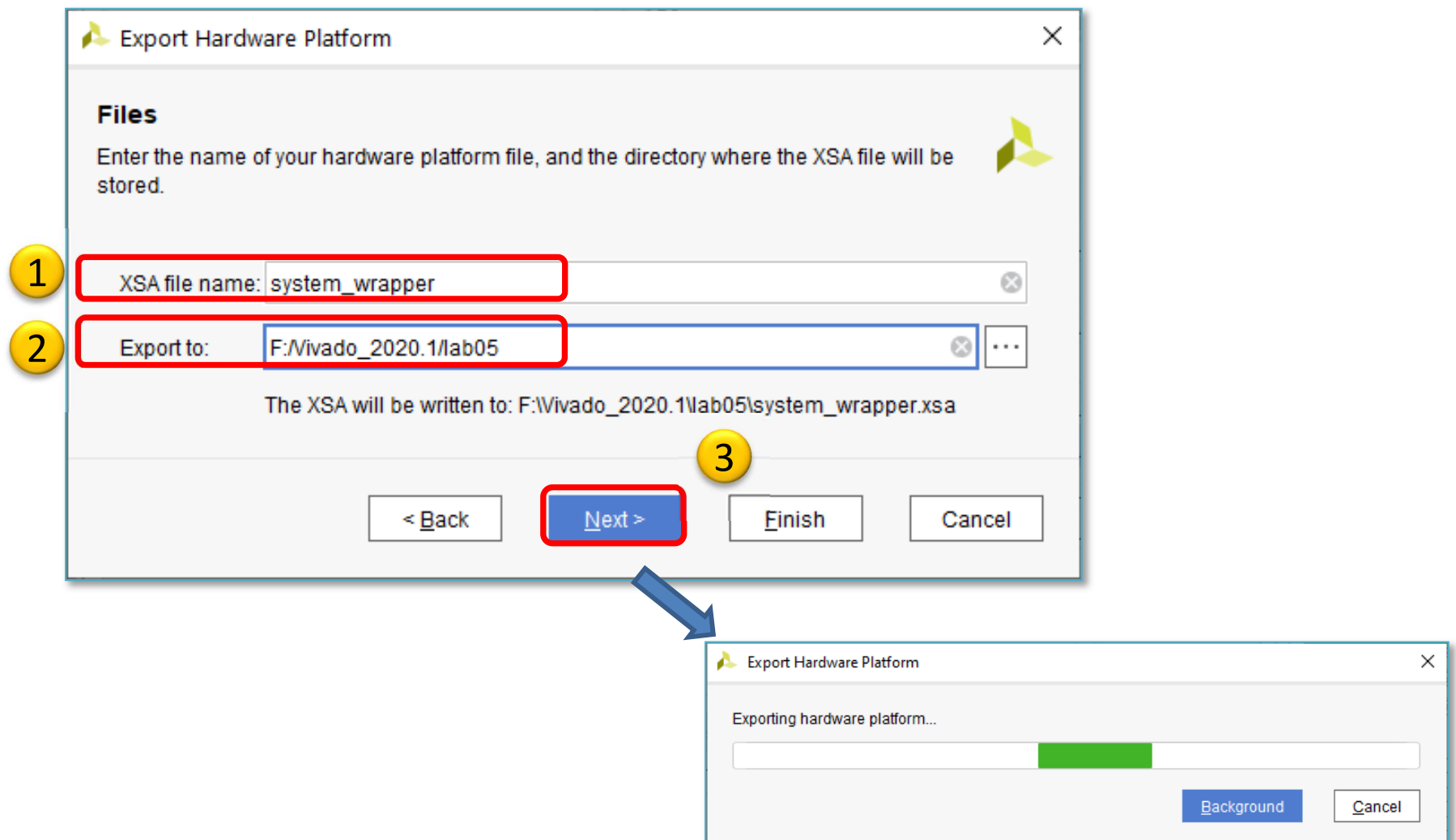
# VIVADO Export HW → VITIS (cont.)

Select „Include bitstream” option as output:



# Export HW → VITIS (cont.)

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







# USING XILINX VITIS

LAB04. Creating a software test application for XADC

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

# Starting VITIS



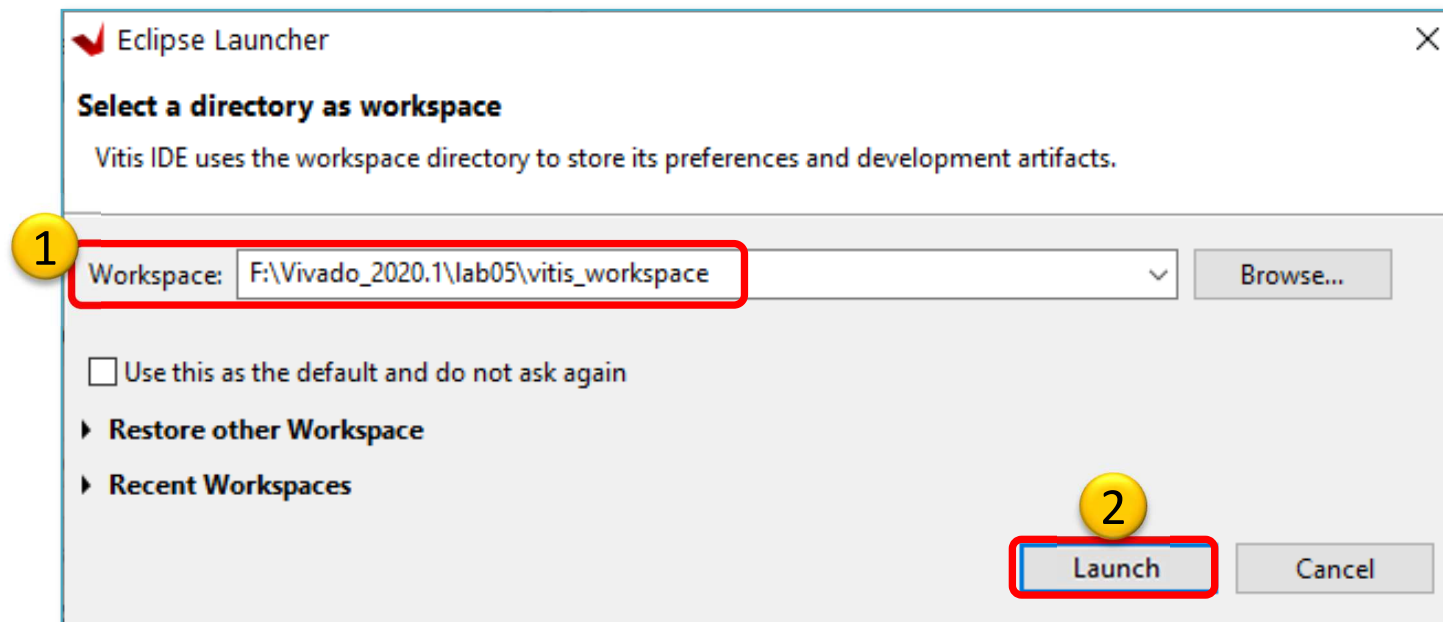
From Vivado: Tools menu → Launch VITIS IDE

OR externally

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

Do Not run Xilinx VITIS HLS 2020.2 !

- Set workspace directory properly (*lab04*):
  - Recommended to use *vitis\_workspace* as a subdirectory in your lab folder. Then Launch...



# Xilinx Vitis – Create Application

Recall the steps of the former LAB01/LAB02\_A!

## 1. Create a new application project

- File → New → Application Project...

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

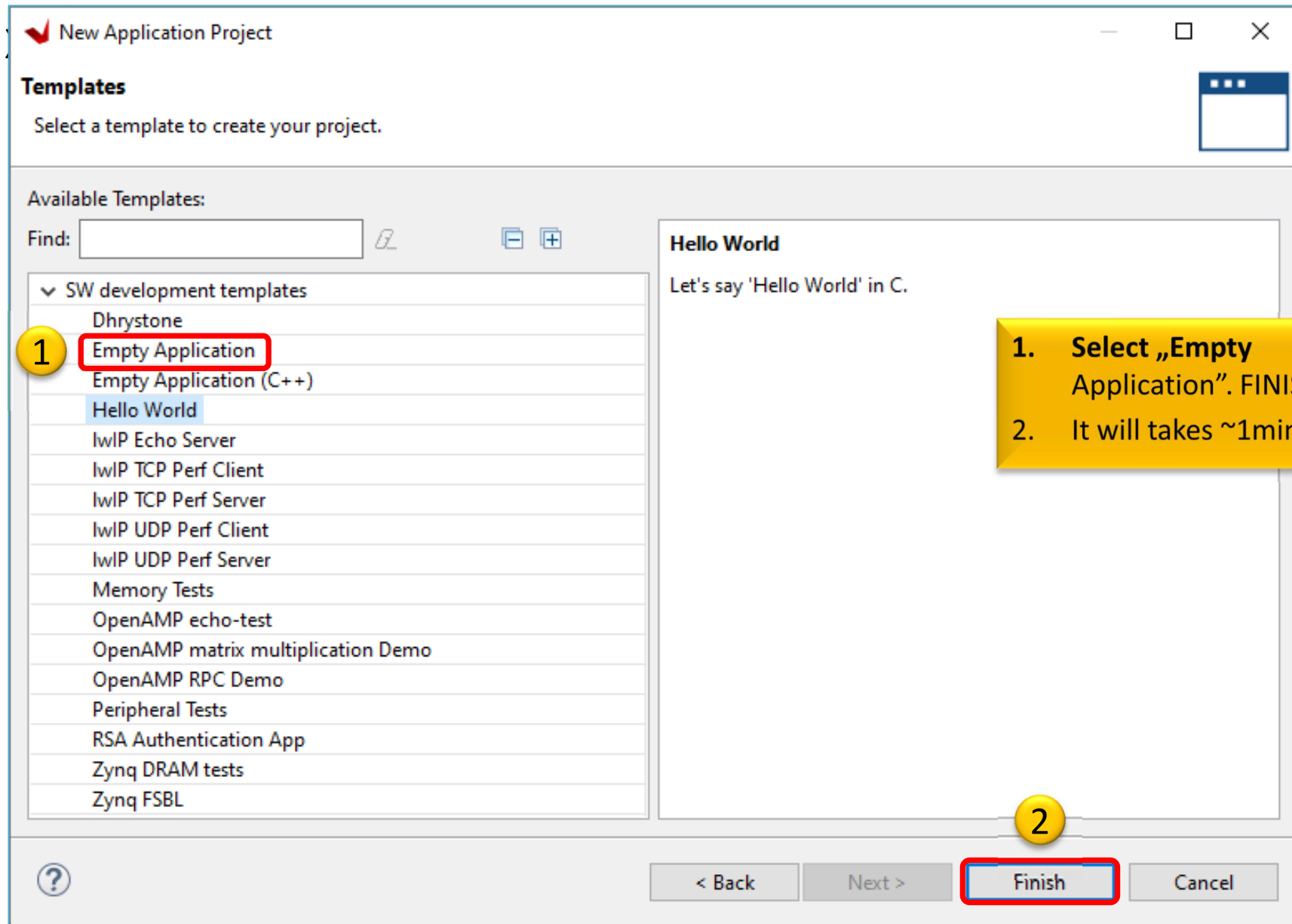
- Browse... for LAB04 `system_wrapper.xsa`. Open it.
- Do not select the „*Generate boot components*”

## 3. Application project details

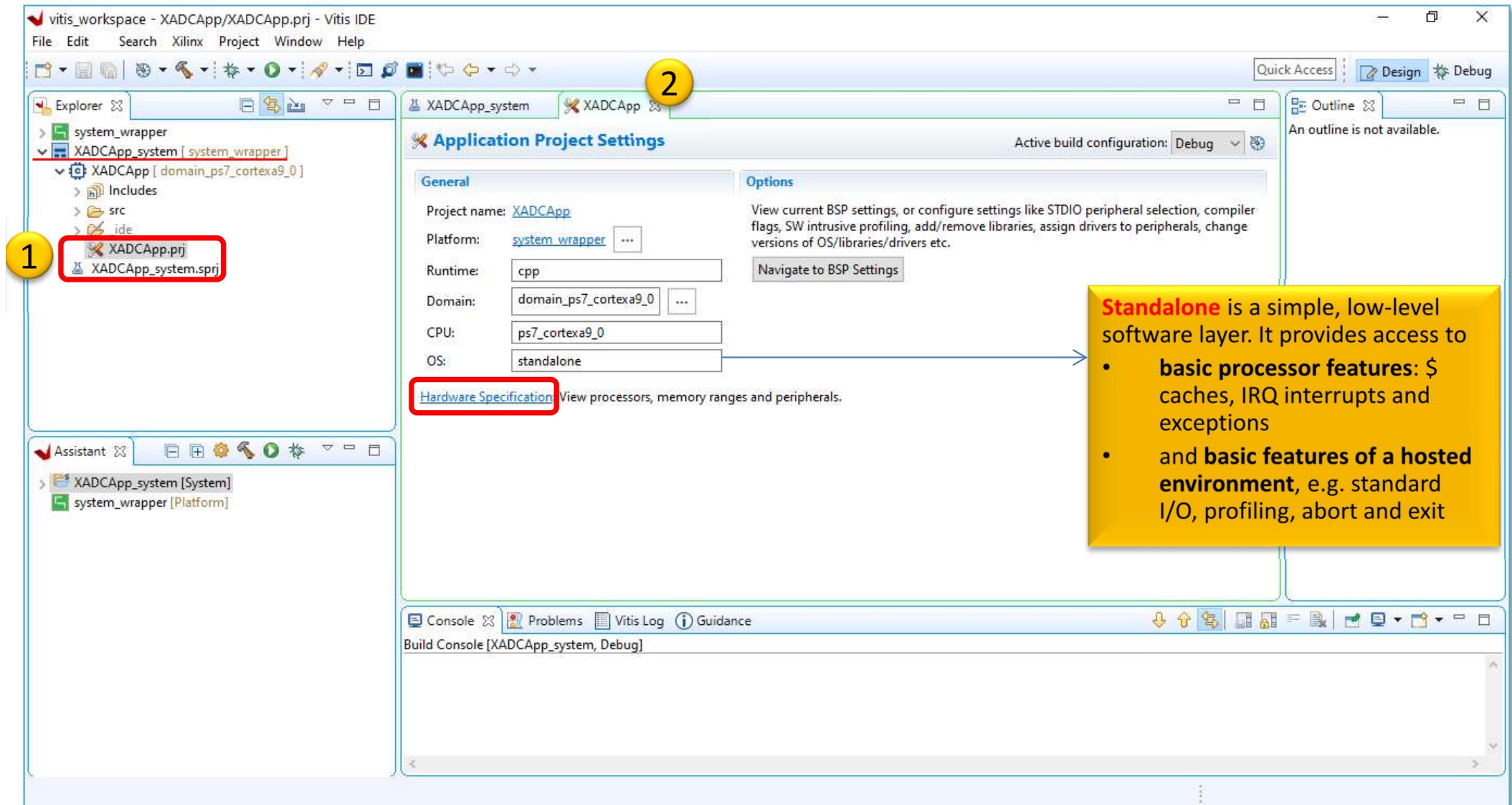
- Type „`XADCApp`” as project name
- Type „`TestApp_system`” as system project name
- Select `ps7_cortexa9_0` as target ARM core 0

## 4. Domain: leave settings as default (standalone)

# Example I.) Creating TestApp application



# VITIS GUI – Main window



# VITIS – HW platform

**1** platform.spr

**2** system\_wrapper

**3** Hardware Specification

**4** Address Map for processor ps7\_cortexa9[0-1]

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

Cell	Base Address	High Address	Slave Interface	Addr Range Type
ps7_ram_0	0x00000000	0x0002ffff	-	memory
ps7_ddr_0	0x00100000	0x1ffffff	-	memory
dip	0x41200000	0x4120ffff	S_AXI	register
pb	0x41210000	0x4121ffff	S_AXI	register
leds	0x41220000	0x4122ffff	S_AXI	register
xadc_wiz_0	0x42000000	0x4200ffff	s_axi_lite	register
ps7_uart_1	0xe0001000	0xe0001fff	-	register

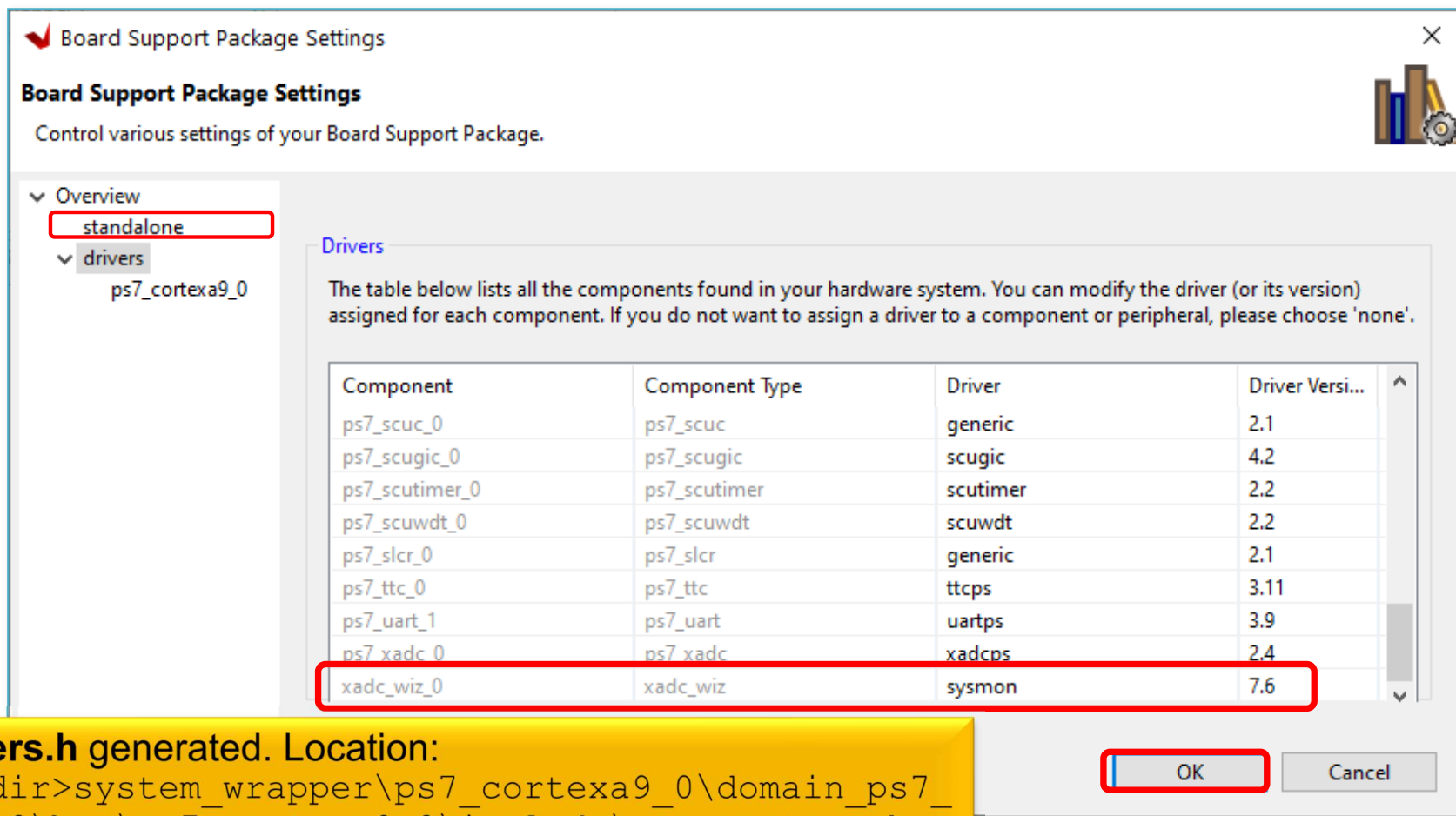
IP Instance	IP Type	IP Version	Register
rst_ps7_0_100M	proc_sys_reset	5.0	-
xadc_wiz_0	xadc_wiz	3.3	-
ps7_0_axi_periph	axi_interconnect	2.1	-
dip	axi_gpio	2.0	<a href="#">Registers</a>
pb	axi_gpio	2.0	<a href="#">Registers</a>
leds	axi_gpio	2.0	<a href="#">Registers</a>
ps7_intc_dist_0	ps7_intc_dist	1.00.a	-



# VITIS – BSP Board Support Package

- **Software Platform Settings**

- Selected OS: *standalone*
- Check the supported SW drivers (and its version): „*sysmon*” v.7.6



xparameters.h generated. Location:

<projectdir>system\_wrapper\ps7\_cortexa9\_0\domain\_ps7\_cortexa9\_0\bsp\ps7\_cortexa9\_0\include\xparameters.h



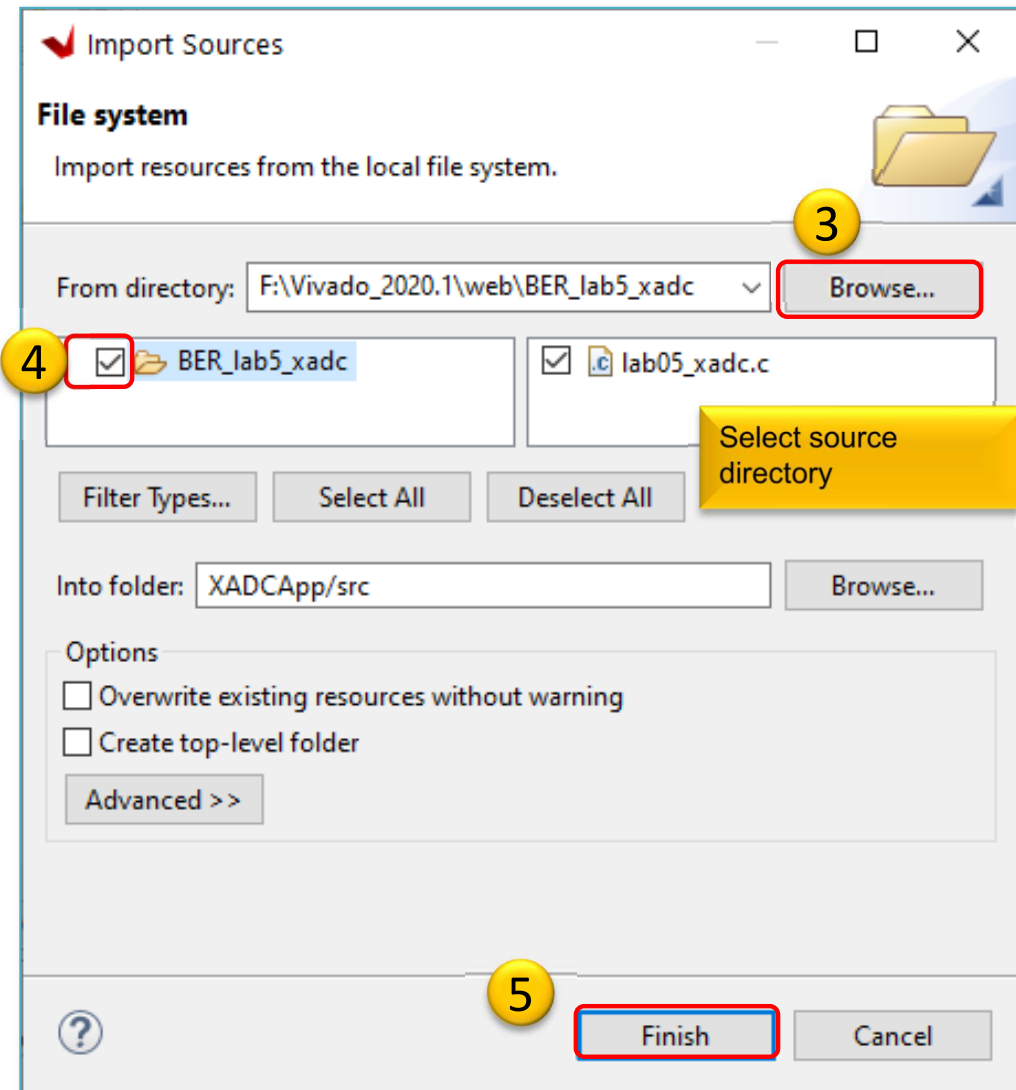
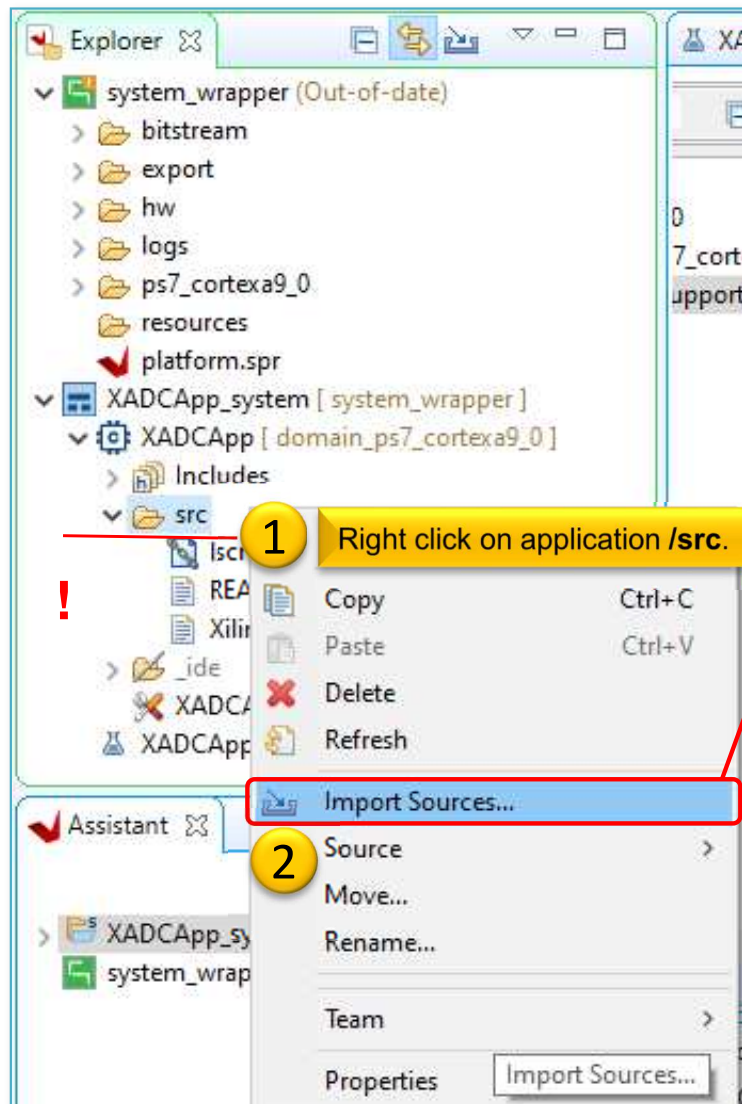
# Q & A 1.)

- **What is the *IP type* and *IP version* of „xadc\_wiz\_0” instance?**
  - xadc\_wiz,
  - v3.3
- **What is the driver name and version of it?**
  - sysmon,
  - v7.6
- **Calculate what size they are?**
  - xadc\_wiz\_0: 0x4200 0000–0x4200 ffff = 64 KByte

# Import C/C++ source(s)

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

Download the archive from lab's website:  
[BER\\_lab4\\_xadc](#)



# Analyzing C source code I.

```
xil_printf("-- PL XADC_WIZARD demonstration on ZyBo -- \n\r");
xil_printf("XADC configuration started...\n\r");
//XADC initialization
ConfigPtr = XAdcPs_LookupConfig(XPAR_AXI_XADC_0_DEVICE_ID);
if (ConfigPtr == NULL) {
    return XST_FAILURE;
}

Status_ADC = XAdcPs_CfgInitialize(XADCInstPtr, ConfigPtr, ConfigPtr->BaseAddress);
if (XST_SUCCESS != Status_ADC) {
    print("ADC INIT FAILED\n\r");
    return XST_FAILURE;
}

//self test
Status_ADC = XAdcPs_SelfTest(XADCInstPtr);
if (Status_ADC != XST_SUCCESS) {
    return XST_FAILURE;
}

//stop sequencer
XAdcPs_SetSequencerMode(XADCInstPtr, XADCPS_SEQ_MODE_SINGCHAN);

//disable alarms
XAdcPs_SetAlarmEnables(XADCInstPtr, 0x0);

//configure sequencer to just sample internal on chip parameters
XAdcPs_SetSeqInputMode(XADCInstPtr, XADCPS_SEQ_MODE_SAFE);

//configure the channel enables we want to monitor
XAdcPs_SetSeqChEnables(XADCInstPtr, XADCPS_CH_TEMP|XADCPS_CH_VCCINT|XADCPS_CH_VCCAUX|
XADCPS_CH_VBRAM|XADCPS_CH_VCCPINT|
XADCPS_CH_VCCPAUX|XADCPS_CH_VCCPDRO);
```

Functions Declared  
in xadcps.c  
and xadcps.h.

## XADC Wizard (3.3)

Basic ADC Setup Alarms Char

### Interface Options

☒ AXI4Lite ☐ DRP ☐ None

### Startup Channel Selection

☐ Simultaneous Selection

☐ Independent ADC

☐ Single Channel

☒ Channel Sequencer

# Analyzing C source code II.

```
while(1){
    xil_printf("\n\r Next iteration - read out XADC registers ...\n\r");

    TempRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_TEMP);
    TempData = XAdcPs_RawToTemperature(TempRawData);
    printf("Raw Temp %lu -> Real Temp %f [C°]\n\r", TempRawData, TempData);

    VccIntRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCINT);
    VccIntData = XAdcPs_RawToVoltage(VccIntRawData);
    printf("Raw VccInt %lu -> Real VccInt %f [V]\n\r", VccIntRawData, VccIntData);

    VccAuxRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCAUX);
    VccAuxData = XAdcPs_RawToVoltage(VccAuxRawData);
    printf("Raw VccAux %lu -> Real VccAux %f [V]\n\r", VccAuxRawData, VccAuxData);




    VBramRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VBRAM);
    VBramData = XAdcPs_RawToVoltage(VBramRawData);
    printf("Raw VccBram %lu -> Real VccBram %f [V]\n\r", VBramRawData, VBramData);

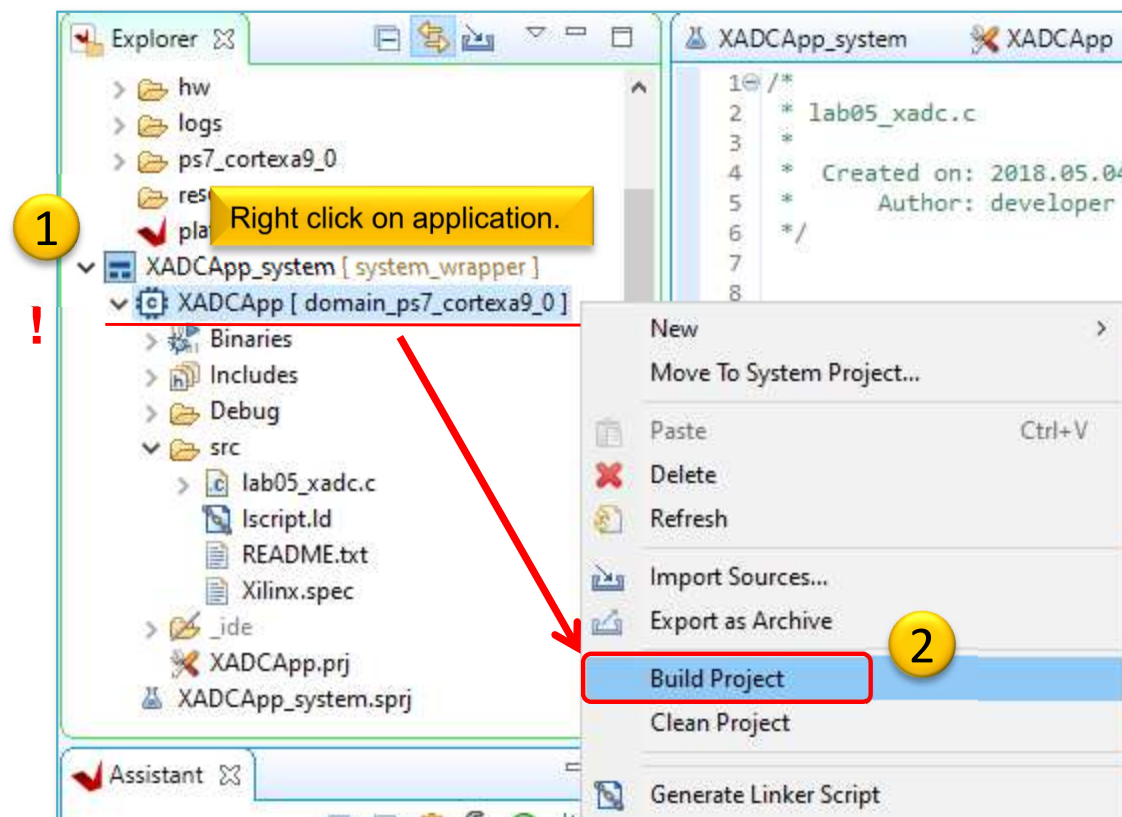
    VccPIntRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCPINT);
    VccPIntData = XAdcPs_RawToVoltage(VccPIntRawData);
    printf("Raw VccPInt %lu -> Real VccPInt %f [V]\n\r", VccPIntRawData, VccPIntData);

    VccPAuxRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCPAUX);
    VccPAuxData = XAdcPs_RawToVoltage(VccPAuxRawData);
    printf("Raw VccPAux %lu -> Real VccPAux %f [V]\n\r", VccPAuxRawData, VccPAuxData);

    VDDRRawData = XAdcPs_GetAdcData(XADCInstPtr, XADCPS_CH_VCCPDRO);
    VDDRData = XAdcPs_RawToVoltage(VDDRRawData);
    printf("Raw VccDDR %lu -> Real VccDDR %f [V]\n\r", VDDRRawData, VDDRData);
}
```

# Build project

- 1. Select Application project (e.g. [XADCApp](#))
- 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: XADCApp.elf'
'Invoking: ARM v7 gcc linker'
arm-none-eabi-gcc -mcpu=cortex-a9 -mfloat-abi=hard -Wl,-
build-id=none -specs=Xilinx.spec -Wl,-T -Wl,../src/lscript.ld -
LF:/Vivado_2020.2/lab04/vitis_workspace/system_wrapper/export/system_
wrapper/sw/system_wrapper/domain_ps7_cortexa9_0/bsplib/lib -o
"XADCApp.elf" ./src/lab04_xadc.o -Wl,--start-group,-lxil,-lgcc,-
lc,--end-group
'Finished building target: XADCApp.elf'
', '
```

```
'Invoking: ARM v7 Print Size'
arm-none-eabi-size XADCApp.elf |tee "XADCApp.elf.size"
  text    data    bss    dec    hex filename
 51335   2556  22656  76547  12b03 XADCApp.elf
'Finished building: XADCApp.elf.size'
```

**Decimal size: 76547 byte** ~76 Kbyte!!! The entire program can be placed only in the internal on-chip RAM0 or 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.

# Build

- Generate Linker Script to the internal 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 XADCAApp.elf |tee "XADCAApp.elf.size"  
   text    data     bss      dec     hex filename  
  51335    2556   10368   64259   fb03 XADCAApp.elf  
'Finished building: XADCAApp.elf.size'
```



# Embedded system and software test verification

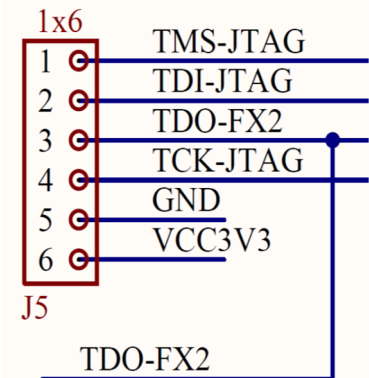
1. **Connect** the USB-serial cable (power+programmer functionality). Please check:

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

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

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

JTAG Header



ZyBo – Xilinx USB programming cable

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

We use the USB-serial connector.

Check the DONE led!



# TestApp – Verification result

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

```
-- PL XADC_WIZARD demonstration on ZyBo --  
XADC configuration started...
```

```
    Next iteration - read out XADC registers ...  
Raw Temp 40520 -> Real Temp 38.452148 [C°]  
Raw VccInt 21689 -> Real VccInt 0.992844 [V]  
Raw VccAux 39154 -> Real VccAux 1.792328 [V]  
Raw VccBram 21694 -> Real VccBram 0.993073 [V]  
Raw VccPInt 21654 -> Real VccPInt 0.991241 [V]  
Raw VccPAux 39142 -> Real VccPAux 1.791779 [V]  
Raw VccDDR 32598 -> Real VccDDR 1.492218 [V]  
...
```

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

# LAB04 XADC – Summary

- To the ARM-AXI based system created in the former (5. – LAB02\_A), here we added new PL-side **XADC** system monitor from the **Vivado** IP catalog.
- Peripheral were properly configured 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 (**XADCApp**) 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**