

Dr. Vörösházi Zsolt:

Beágyazott rendszerek fejlesztése (FPGA)



**A felsőfokú informatikai oktatás
minőségének fejlesztése,
modernizációja**

TÁMOP-4.1.2.A/1-11/1-2011-0104



Főkedvezményezett:
Pannon Egyetem
8200 Veszprém
Egyetem u. 10.

Kedvezményezett:
Szegedi Tudományegyetem
6720 Szeged
Dugonics tér 13.



Frissítve: 2019. október 12.

Beágyazott rendszerek fejlesztése (FPGA)

Dr. Vörösházi Zsolt

4. Perifériák hozzáadása (IP adatbázisból) az összeállított beágyazott alapszisztemhez



1. Bevezetés – Beágyazott rendszerek, FPGA-k, Digilent ZYBO fejlesztő kártyák és eszközök
2. Beágyazott Rendszer fejlesztő szoftverkörnyezet (Xilinx Vivado Embedded Development) áttekintése
3. Beágyazott alap tesztrendszer (BSB - Base System Builder and Board Bring-Up) összeállítása
4. **Perifériák hozzáadása (IP adatbázisból) az összeállított beágyazott alaprendszerhez. Saját periféria hozzáadása az összeállított beágyazott alaprendszerhez**
5. Szoftver alkalmazások fejlesztése, tesztelése, hibakeresése (debug) Xilinx Vivado SDK (Software Development Kit) használatával

Xilinx Vivado használata

IP HOZZÁADÁSA A BEÁGYAZOTT RENDSZERHEZ

- Vivado – Block Design:
 - *Katalógusban lévő IP-k* (Intellectual Property) hozzáadása az elkészült hardver rendszerhez,
 - Az **.XDC** fájl módosítása: az IP magok külső portokhoz rendelése (lábak),
- Vivado SDK
 - fordító beállításainak testre szabása,
 - Teszt-alkalmazás generálása, és megírása a beépített SW template (sablon) alapján.

A feladat megoldásának lépései

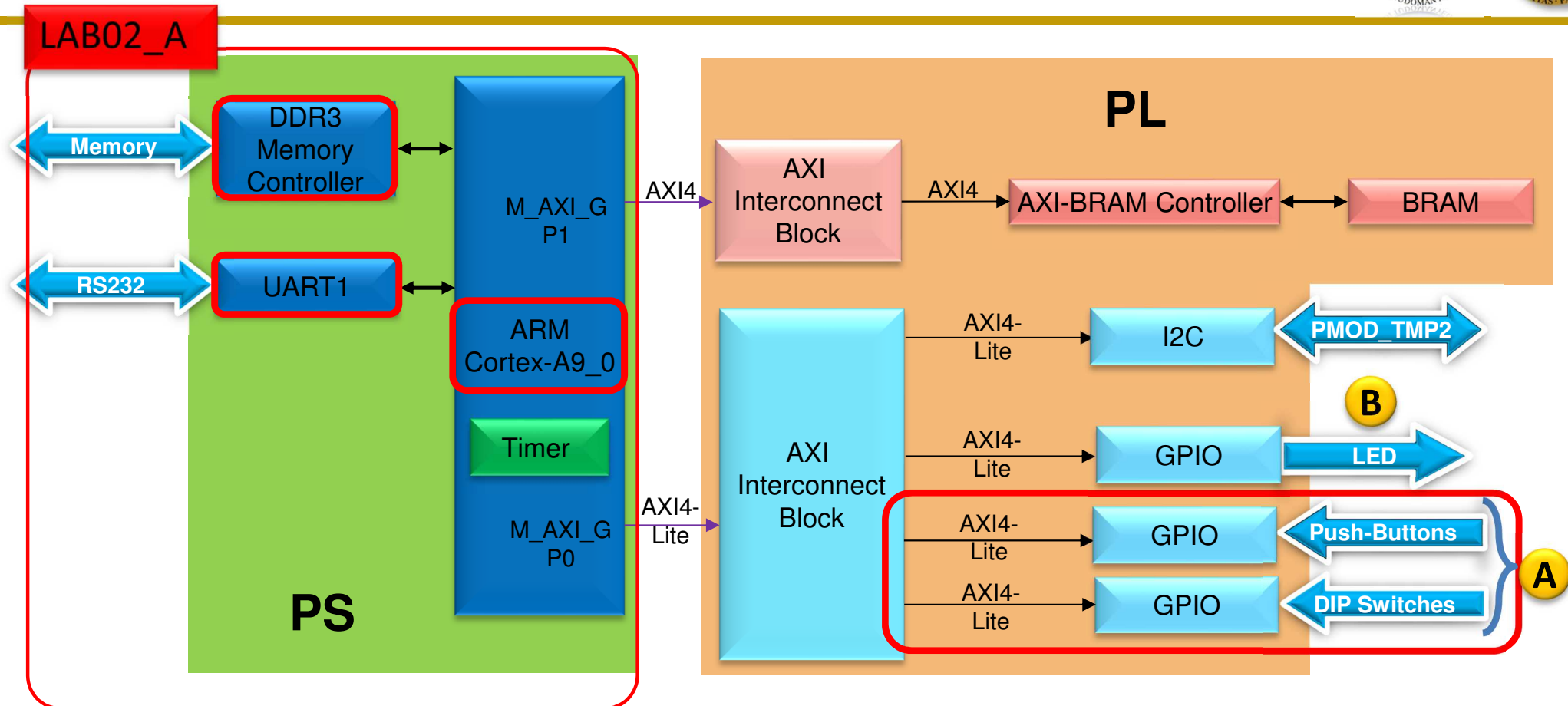


- Az előző (03. fólia) ismeretkör elsajátítása során létrehozott projekt archiválása (LAB01 → LAB02 néven), majd pedig a \LAB02 megnyitása Vivado-ban,
- Az IP katalógusból kiválasztott GPIO perifériák integrálása és összekötése az alaprendszerrel,
- Külső GPIO portok létrehozása (.XDC),
- Egy Periféria teszt-alkalmazás (TestApp) készítése az SDK-környezetben,
- A FW+SW tervek teszt verifikálása a Digilent ZyBo kártyán.

A. NYOMÓGOMBOK, DIP KAPCSOLÓK

BEÁGYAZOTT RENDSZER ÉS SZOFTVER ALKALMAZÁS ÖSSZEÁLLÍTÁSA

A bővíthető tesztrendszer



PS oldal:

- ARM hard-processzor mag (Core0)
- belső OnChip-RAM vezérlő
- UART1 - RS232 soros interfész
- külső DDR3 memória vezérlő
- (I2C vezérlő)

PL oldal:

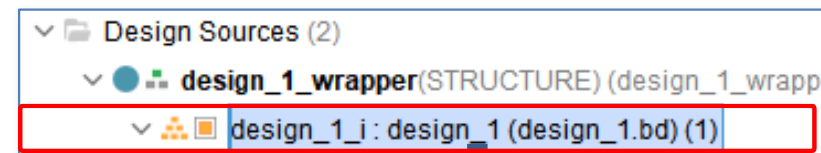
- **(A) LAB02_A: GPIO bemenetek**
 - **PBSs: Push Button (nyomógomb kezelő)**
 - **DIPs: Switches (kapcsoló kezelő)**
- **(B) LAB02_B: GPIO kimenet**
 - **LEDs: LED kijelző**

- Hozzunk létre egy új mappát, legyen a neve `\LAB02_A`
 - **File -> Project -> Save As... (LAB02_A) vagy**
 - Másoljuk át az előző ismeretkör elsajátításakor létrehozott projektet (azaz a `\LAB01` mappa tartalmát)
- Indítsuk el a Vivado szoftvert:
 - **Start → Programok →**
Xilinx Design Tools → Vivado 2018.3 → Vivado 2018.3
- **File → Open Project →**
`<projectdir>/LAB02_A/<system>.xpr →`
Open

Zynq – beállítások módosítása



Design Sources -> *design_1.bd* blokk design megnyitása



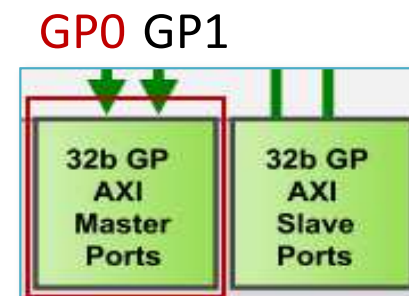
Processing_system7_0 megnyitása

- PS-PL configuration:

- General -> Enable Clock Resets:
FCLK_RESET0_N engedélyezése

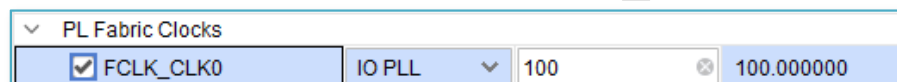


- AXI Non Secure Enablement: GP Master AXI interfészben a **M_AXI_GP0** port (híd) engedélyezése!



- Clock configuration:

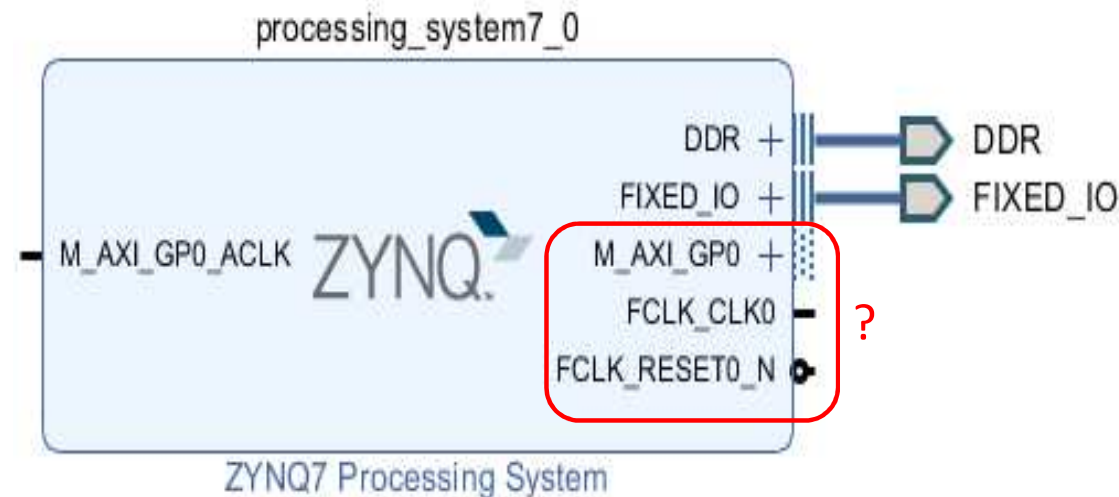
- PL Fabric Clocks: **FCLK_CLK0** (100 MHz IO_PLL) engedélyezése



Zynq blokk diagram



- Vizsgálja meg, hogy az engedélyezett
 - GP0 AXI Master port,
 - FCLK_CLK0 PL órajel port,
 - FCLK_RESET0_N reset portmegjelentek-e ?!



PL oldali AXI GPIO perifériák hozzáadása és összekötése az alaprendszerrel I.



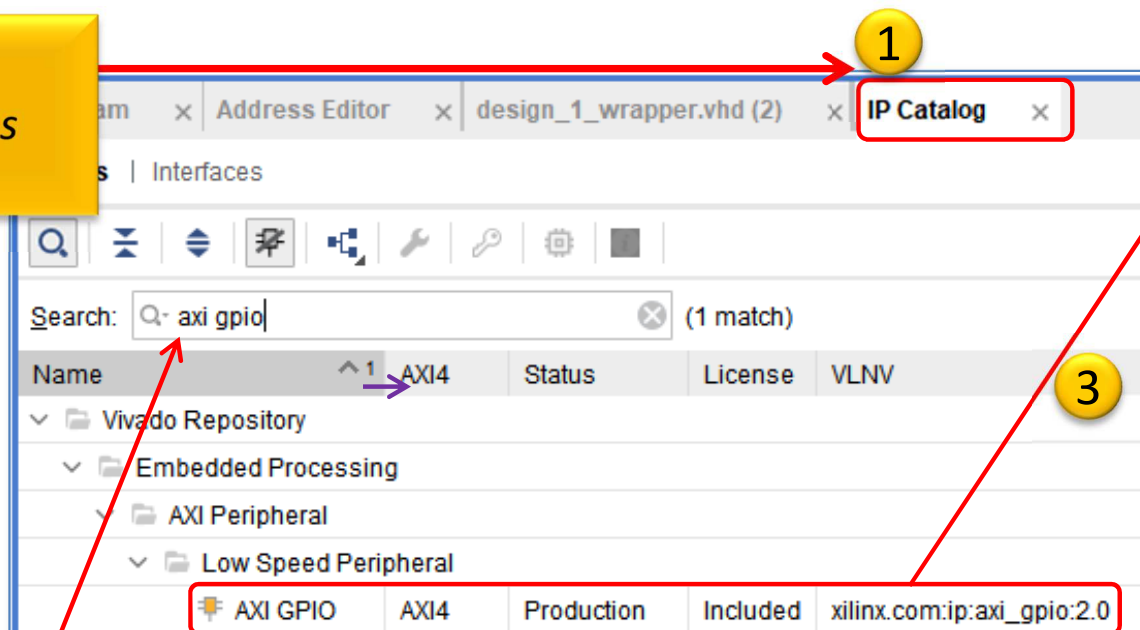
- Új IP mag hozzáadására több lehetőség kínálkozik a Vivado-ban:
 - a.) Block Diagram View -> Add IP  (vagy CTRL+I)
 - b.) IP Catalog megnyitása -> IP kiválasztása -> Dupla kattintással – Add IP to Block Design
- Adjunk a processzor rendszerhez két PL oldali AXI_GPIO perifériát

2x

1x DIP

1x PB

Váltás a
az IP katalógus
nézetre



IP mag hozzáadása
(dupla kattintás)

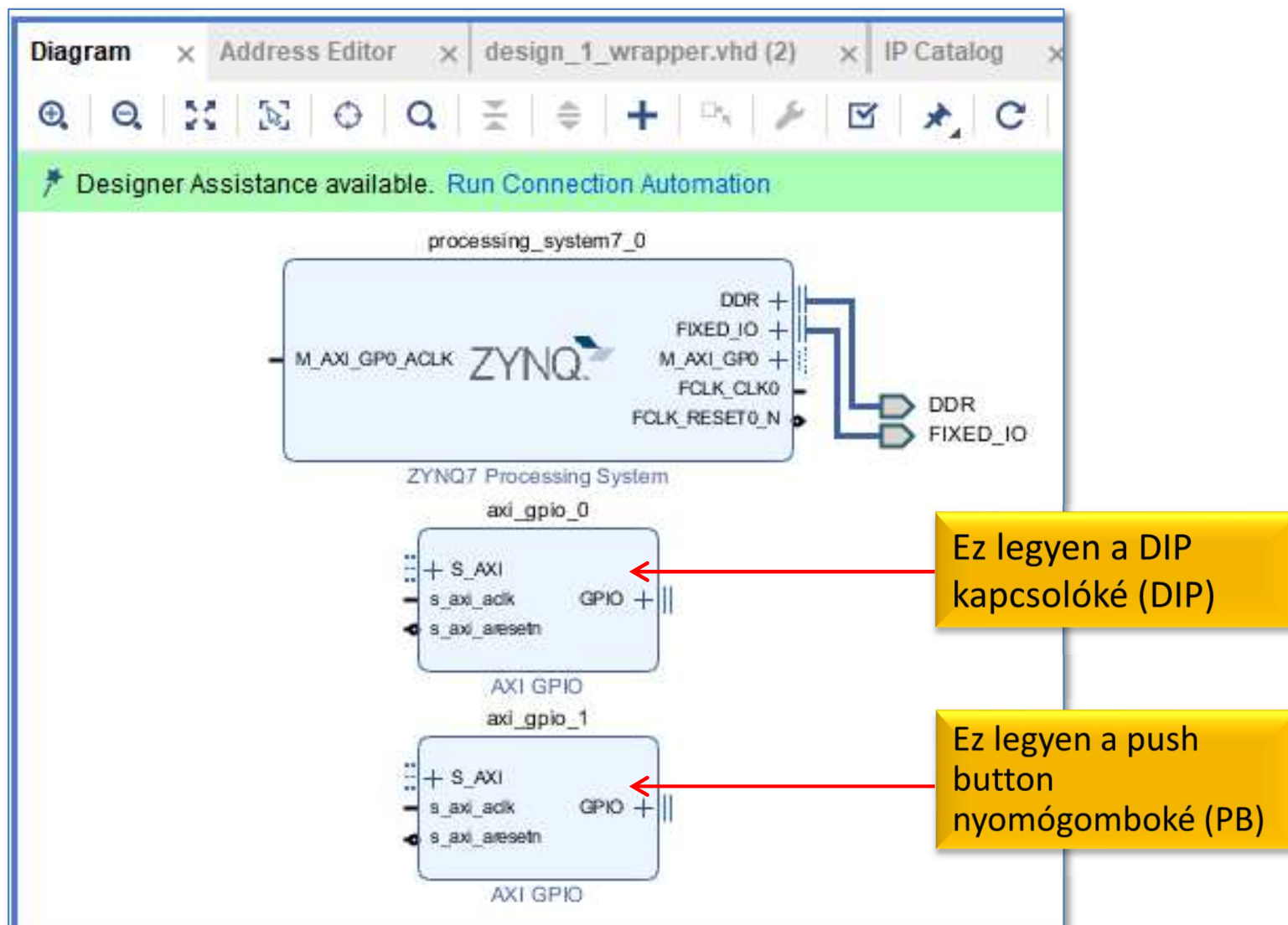
AXI GPIO
szűrés



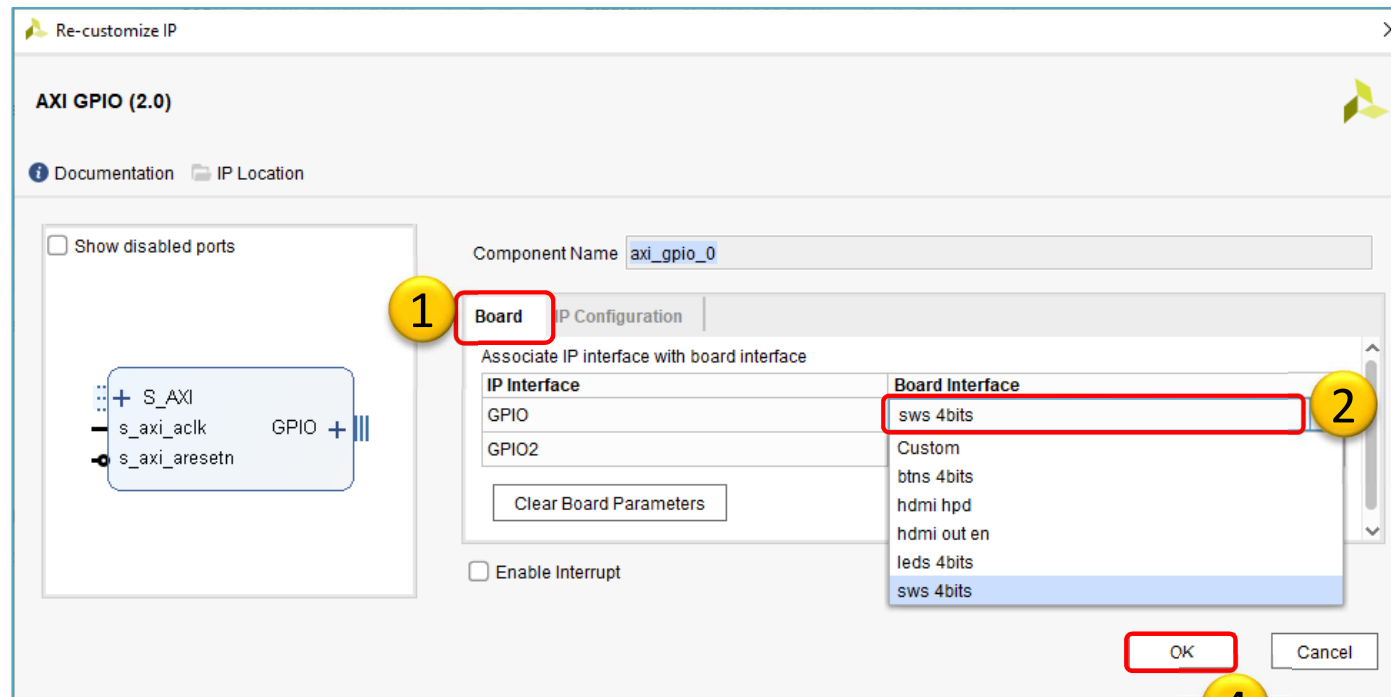
- Ezek után egy-egy IP modulhoz (pl. **AXI_GPIO**) a Vivado-ban következőket szükséges beállítani (lehet manuális / automatikus is!):
 - a.) az ***interfész kapcsolatot*** az IP modul és buszrendszer (AXI) között
 - b.) az IP modul ***címtartományhoz*** rendelését (Base-High Addresses)
 - c.) az IP modulok I/O portjainak ***külső*** (external) ***port***-okhoz rendelése,
 - d.) Végül a külső portok ***fizikai*** FPGA ***lábakhoz*** rendelését (.XDC szerkesztése) – IO planning.

Block Diagram

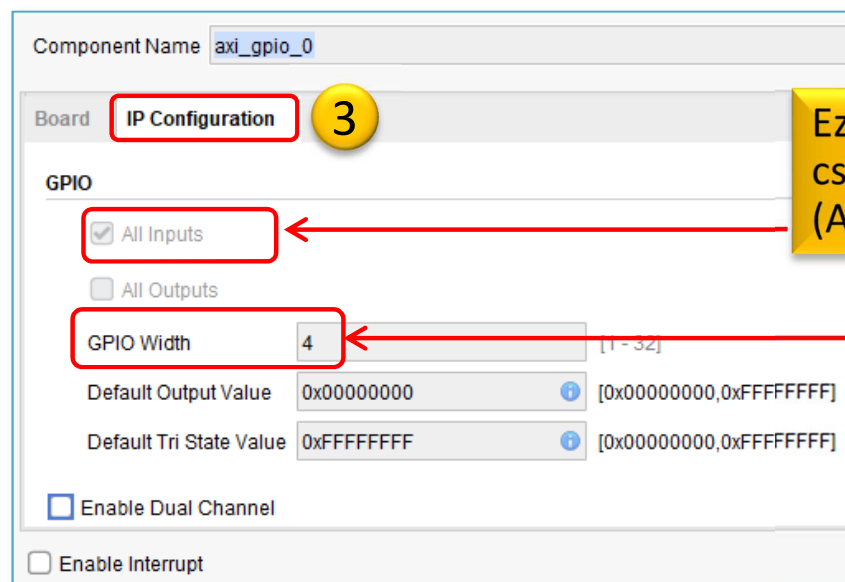
Dupla kattintással állítsuk be egyrészt a AXI GPIO_0 (DIP kapcsolóknak),
másrészt az AXI_GPIO_1-et (PB nyomógomboknak).



AXI GPIO perifériák hozzáadása az alaprendszerhez II. – DIP kapcsoló sor



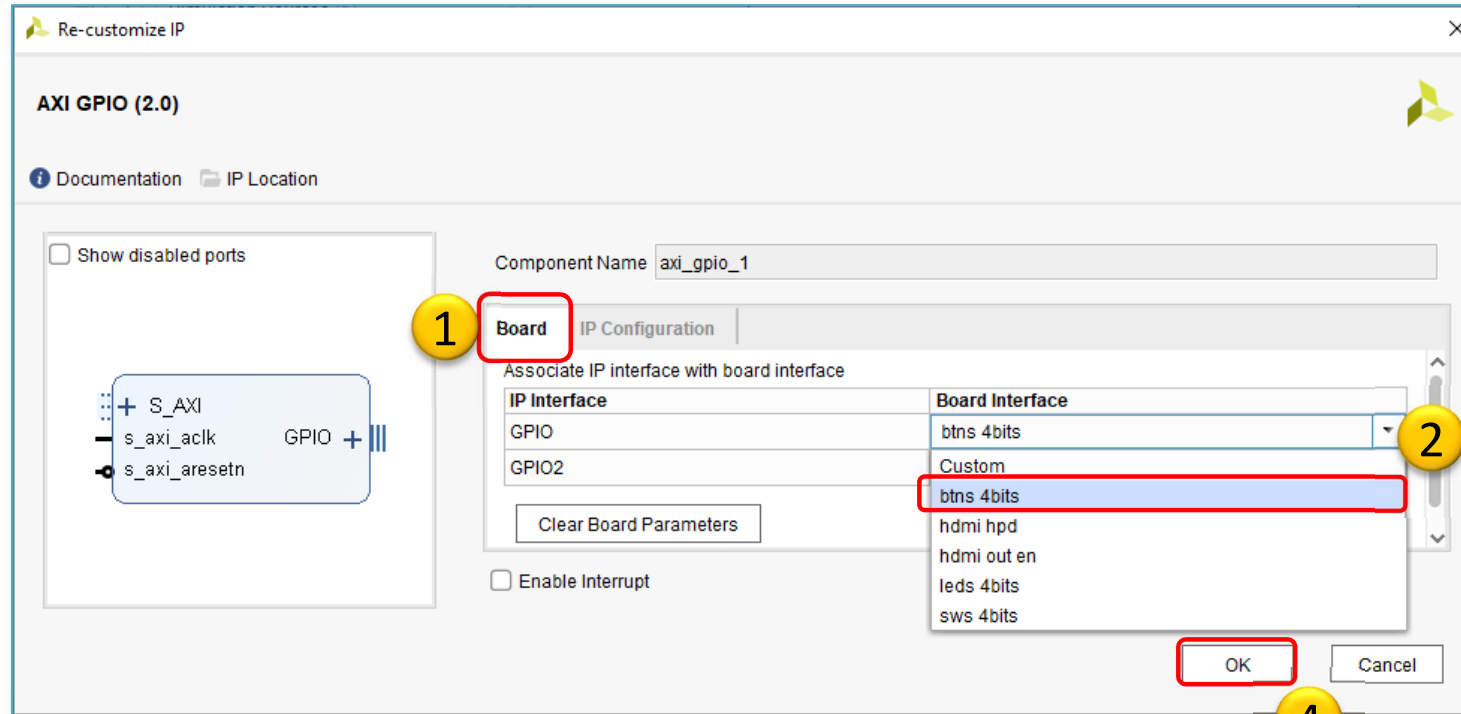
Board interface:
„sws_4bits”
kiválasztása.



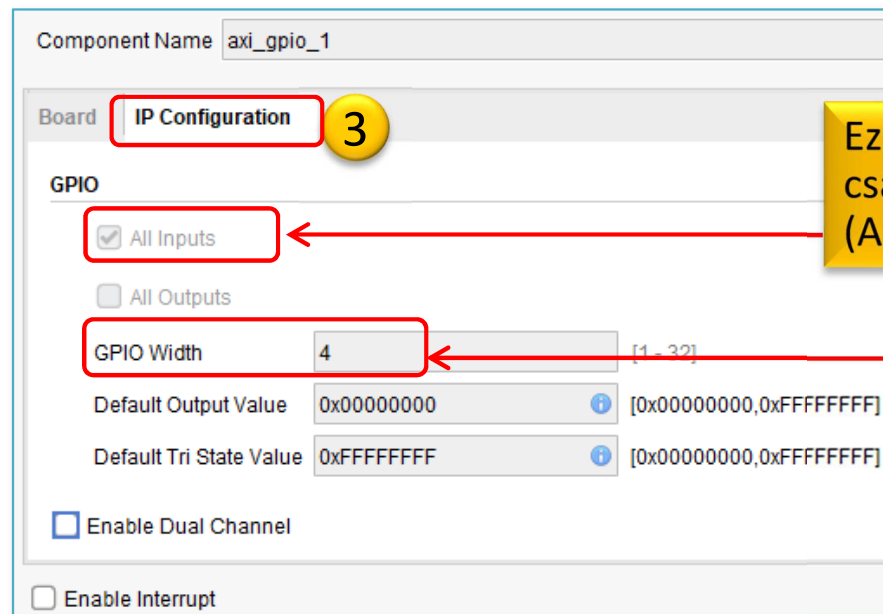
Ez a GPIO csatorna
csak bemenet lehet
(All inputs)

GPIO csatorna
szélessége = 4
(mivel 4 DIP
kapcsoló van a
ZyBo kártyán)

AXI GPIO perifériák hozzáadása az alaprendszerhez III. – PB nyomógombok



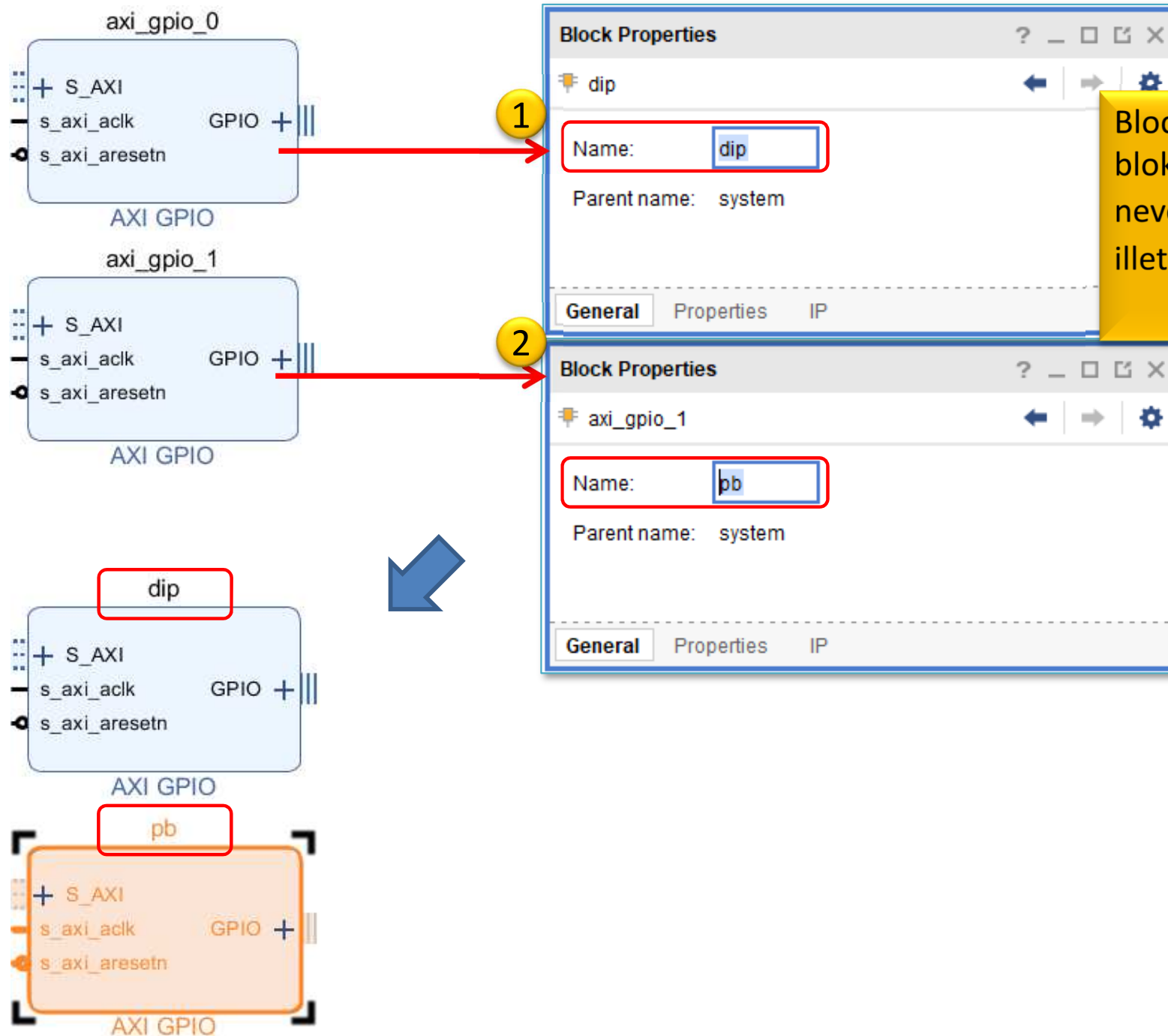
Board interface:
„btns_4bits”
kiválasztása.



Ez a GPIO csatorna
csak bemenet lehet
(All inputs)

GPIO csatorna
szélessége = 4
(mivel 4 PB
nyomógomb van a
ZyBo kártyán)

AXI GPIO perifériák átnevezése



Block Properties -> IP blokkok példány neve legyen **dip**, illetve **pb**.

AXI GPIO-k bekötése (autorouter)

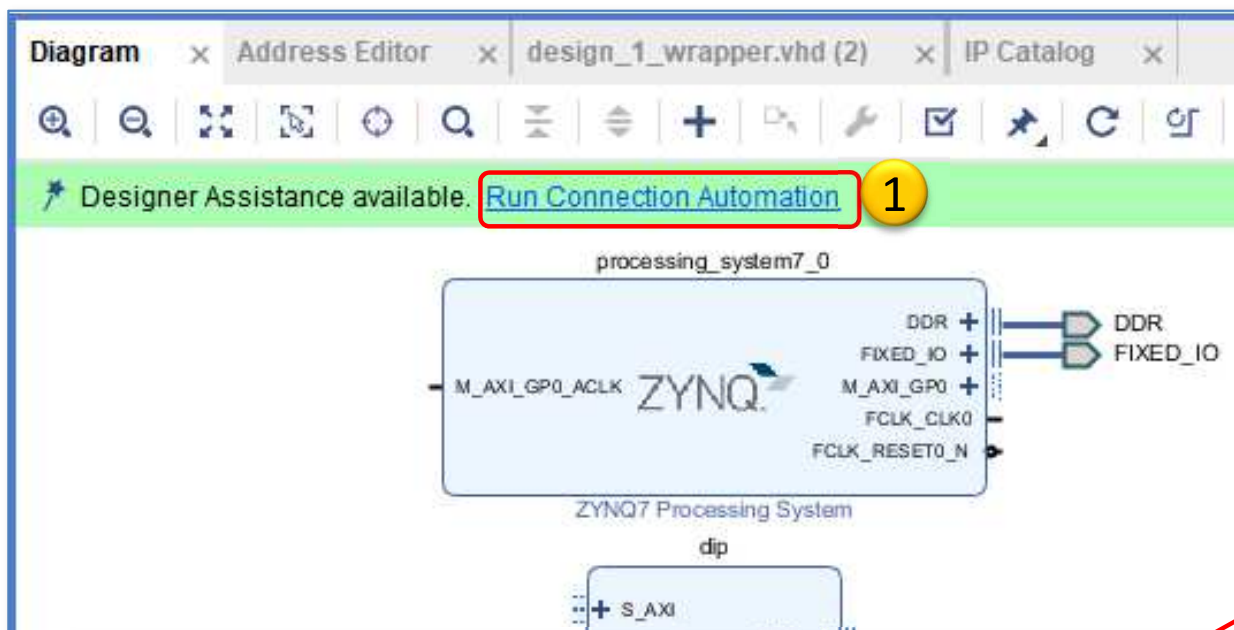


Diagram -> Run
Connection Automation
indítása

DIP -> S_AXI interfész
kiválasztása, ill.
PB -> S_AXI interfész
kiválasztása az
automata routoláshoz

Minden más Auto /
Default

Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface configuration options on the right.

Search | Filter | Sort

▼ ☒ All Automation (2 out of 4 selected)

▼ ☒ dip

☐ GPIO

☒ S_AXI

▼ ☒ pb

☐ GPIO

☒ S_AXI

Description

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

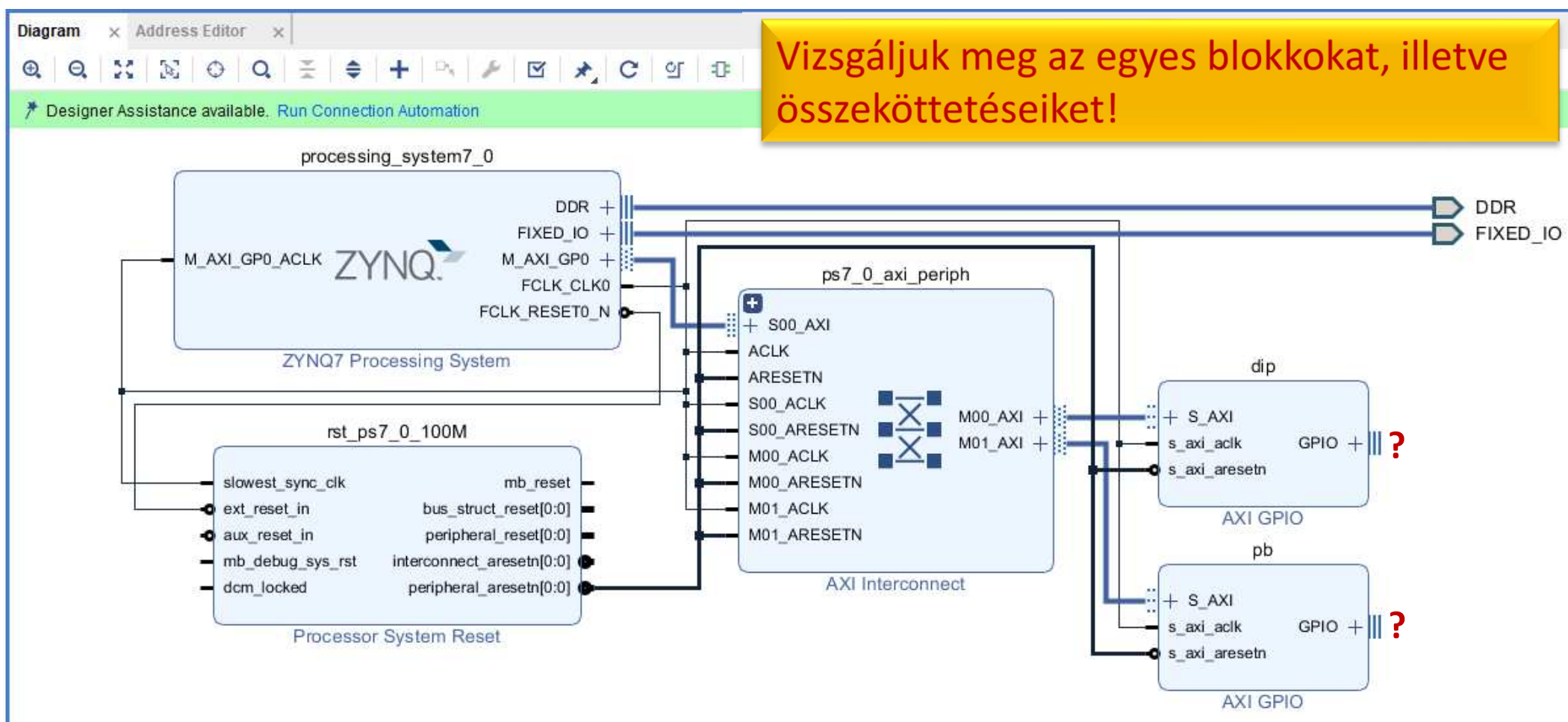
Options

Master: /processing_system7_0/M_AXI_GP0

Bridge IP: New AXI Interconnect

OK Cancel

Block design – teljes nézet



Főbb részei:

- Processing_system7: (PS ARM rendszer)
- Processing_system7_axi_periph: AXI interfész
- Rst_processing_system7_100M: PS/PL reset generátor
- dip: AXI GPIO a DIP kapcsolóknak (PL oldali)
- pb: AXI GPIO a PB nyomógomboknak (PL oldali)

Processing_system7_axi_periph



Analizáljuk az AXI periféria interfész paramétereit!

- Mennyi slave, illetve master oldali interfésze van, és miért?

1 Top Level Settings

Component Name: ps7_0_axi_periph

Number of Slave Interfaces: 1

Number of Master Interfaces: 2

Interconnect Optimization Strategy: Custom

AXI Interconnect includes IP Integrator automatic converter insertion and configuration.

When the endpoint IPs attached to the interconnect differ in width, clock or protocol, a converter IP is inserted. If a converter IP is inserted, IP integrator configures the converter to match the endpoint. To see which conversion IPs have been inserted, click the 'expand hierarchy' buttons to explore the interconnect hierarchy.

NOTE: Addressing information for AXI Interconnect is available in the IP User Guide.

☐ Enable Advanced Configuration Options

2 Slave Interfaces

Slave Interface	Enable Register Slice	Enable Data FIFO
S00_AXI	None	None

3 Master Interfaces

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

OK Cancel

Rst_Processing_system7 (Reset)



Analizáljuk az AXI reset generátor paramétereit, és jeleit!

- Mennyi magas / alacsony aktív *reset* jelet generál ?
- Mely blokkok kapják meg ezeket a *reset* jeleket?

Re-customize IP

Processor System Reset (5.0)

Documentation IP Location

☐ Show disabled ports

Component Name

External Reset

Ext Reset Logic Level (Auto)

Ext Reset Active Width

Auxiliary Reset

Aux Reset Logic Level

Aux Reset Active Width

Active High Reset

Bus Structure

Peripherals

Active Low Reset

Interconnect

Peripherals

☐ slowest_sync_clk mb_reset

☒ ext_reset_in bus_struct_reset[0:0]

☒ aux_reset_in peripheral_reset[0:0]

☒ mb_debug_sys_rst interconnect_aresetn[0:0]

☒ dcm_locked peripheral_aresetn[0:0]

OK Cancel

AXI GPIO - Memória címtartományok beállítása



- Block Design -> „Address Editor” nézet kiválasztása
- „Map”-eletlen IP perifériák memória-címtartományhoz rendelése:
 - a.) automatikusan - címgenerálással vs. b.) manuálisan

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

a.) Automatikus címgenerálás
(jobb gomb -> Auto Assign Address)

b.) Base address kézi beállítása*

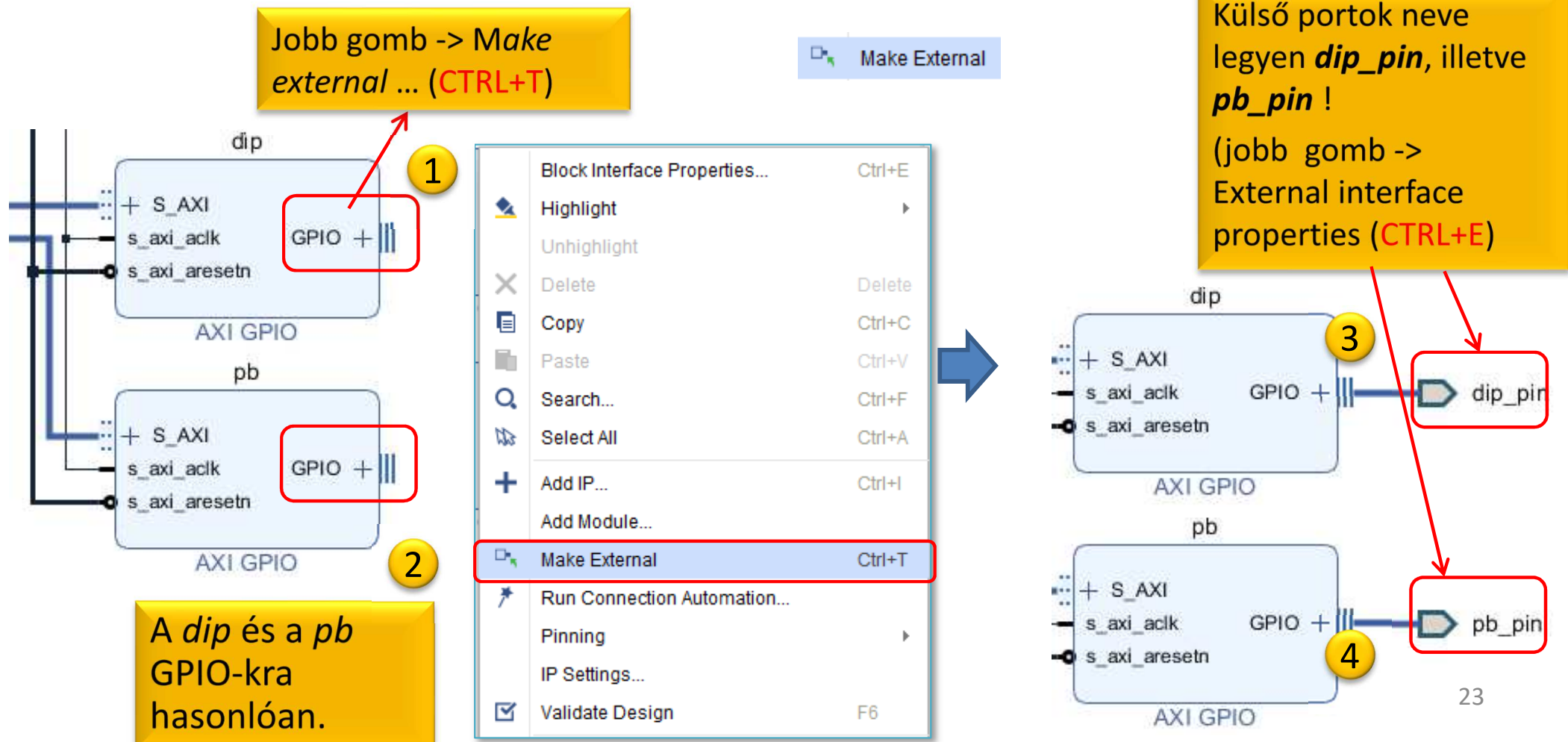
pb: 0x4121_0000 (64K)
dip: 0x4120_0000 (64K)

*Címtartományoknak 2^n méretűeknek kell lennie és nem lapolódhatnak át!

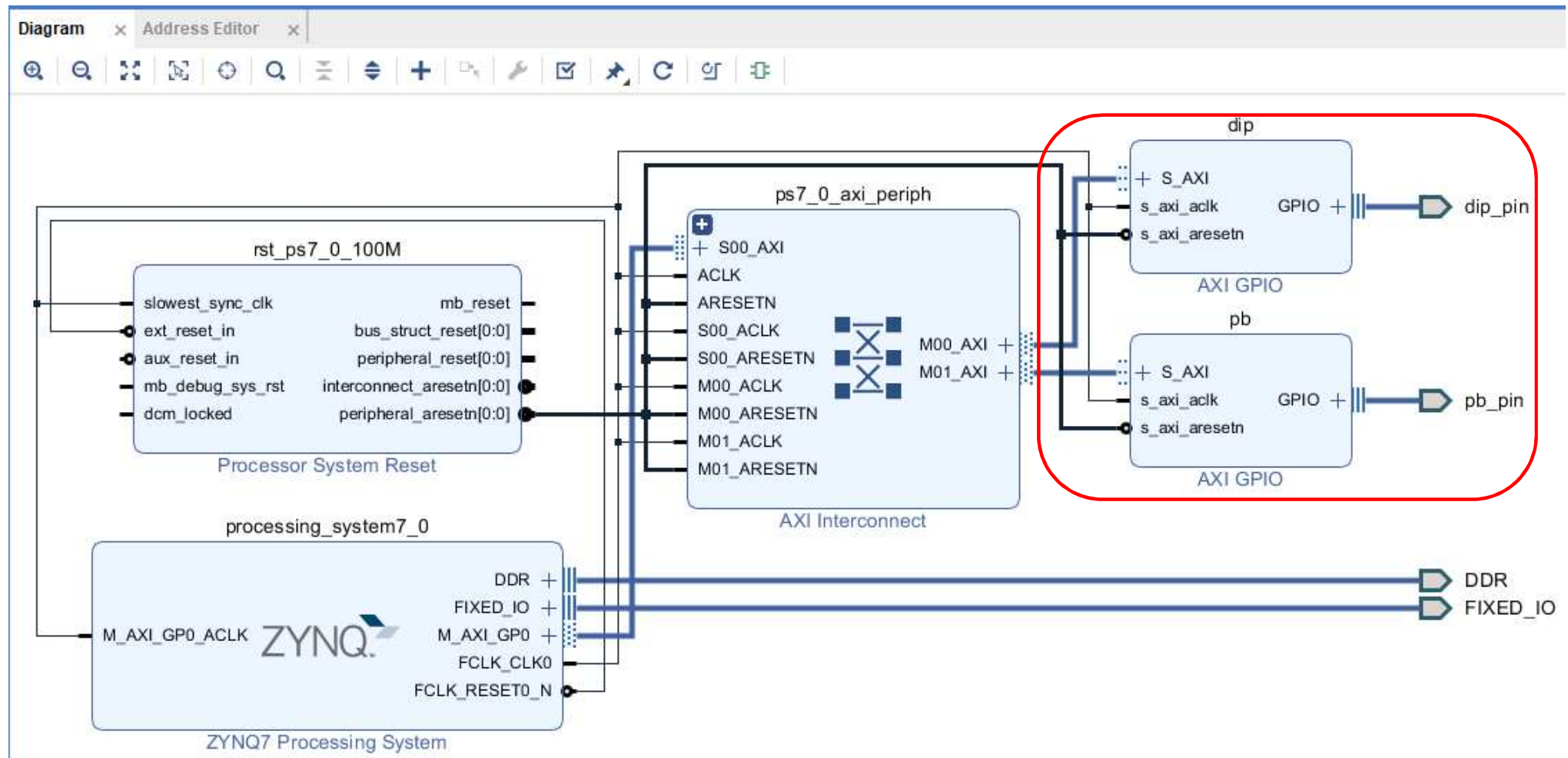
AXI GPIO - Külső portok hozzárendelése

A *dip* és *pb* nevű GPIO példányokat hozzá kell kapcsolni a ZyBo kártyán található (dip) kapcsolókhoz és (pb) nyomógombokhoz tartozó FPGA lábakhoz:

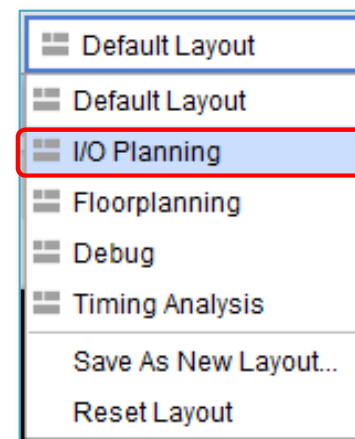
- 1.) A GPIO példányok adat port-jait a külső fizikai FPGA lábakra (pin) is kell kötni,
- 2.) Ha kell, a külső portok neveit is definiáljuk (pl: `_pin` végződésűre), majd
- 3.) Az `<system>.XDC` fájl-ban meg kell adni az adott FPGA pin azonosítóját.



Összeállított rendszer



- Ezek után egy-egy a Block Design-t frissíteni kell:
 - Regenerate Layout 
 - Validate Design (DRC) 
 - Flow Navigator -> Run Synthesis  Run Synthesis
 - Majd - **Open Synthesized Design** , OK
- Utolsó lépésként még a két külső porthoz (`dip_pin`, illetve `pb_pin`) hozzá kell rendelni az FPGA IO lábakat is!
 - Layout menü -> I/O Planning layout nézet



Zybo_master.xdc alapján ellenőrizhető a helyes (automatikus) lábkiosztás.

[illegible]

- Flow Navigator menü → Run Implementation



- Kiszűri az esetleges elkötéseket, hibákat,
- Figyelmeztető (warning) jellegű üzenetek megengedettek (implementálható a terv),
- Legtöbb lebegő (floating) vezetékekkel sem kell foglalkozni (pl. Peripheral Reset, stb).
- **Miközben a Vivado dolgozik érdemes megnézni a fordítási riportokat!**

Ezután indítható el a Bitstream generálás.

- Flow Navigator → Generate Bitstream.



Layout Editor - Floorplanning



IMPLEMENTED DESIGN - xc7z010clg400-1 (active)

1

2

3

Physical Constraints

Device

Internal VREF

- 0.6V
- 0.675V
- 0.75V
- 0.9V
- NONE (2)
 - I/O Bank 34
 - I/O Bank 35

Drop I/O banks on voltages or the "NONE" folder to set/unset Internal VREF.

Cell Properties

Cell Prop

Clock Regi

dip_pin_tri_i_IBUF[0]_inst

Name: dip_pin_tri_i_IBUF[0]_inst

Reference name: IBUF

Type: IO

BEL: INBUF_EN

Site: G15

General Properties Nets

Netlist

system_wrapper

- Nets (146)
- Leaf Cells (8)
 - dip_pin_tri_i_IBUF[0]_inst (IBUF)
 - dip_pin_tri_i_IBUF[1]_inst (IBUF)
 - dip_pin_tri_i_IBUF[2]_inst (IBUF)
 - dip_pin_tri_i_IBUF[3]_inst (IBUF)
 - pb_dip_tri_i_IBUF[0]_inst (IBUF)
 - pb_dip_tri_i_IBUF[1]_inst (IBUF)
 - pb_dip_tri_i_IBUF[2]_inst (IBUF)
 - pb_dip_tri_i_IBUF[3]_inst (IBUF)
- system_i (system)

Project Summary

Device

system_wrapper.vhd

Timing

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 2,712 ns	Worst Hold Slack (WHS): 0,071 ns	Worst Pulse Width Slack (WPWS): 4,020 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 1701	Total Number of Endpoints: 1701	Total Number of Endpoints: 885

All user specified timing constraints are met.

Timing Summary: impl_4 (saved)

Xilinx Vivado használata

BLOKK DIAGRAM/RIPORTOK VIZSGÁLATA

- 1.) Kérdés (buszok, belső jelek)
 - Mely IP periféria példányok kapcsolódnak a `processing_system7_0_FCLK_CLK0` nevű órajelhez?
 - Mekkora ez az órajel?
 - Mely IP periféria példányok kapcsolódnak az AXI Lite busz interfészhez?
- 2.) Kérdés (címek)
 - Vázolja fel a rendszer teljes memória térképét a példánynevek megadásával!
- 3.) Kérdés (erőforrás foglалás)
 - Mennyi erőforrást foglalunk el PL oldalon?

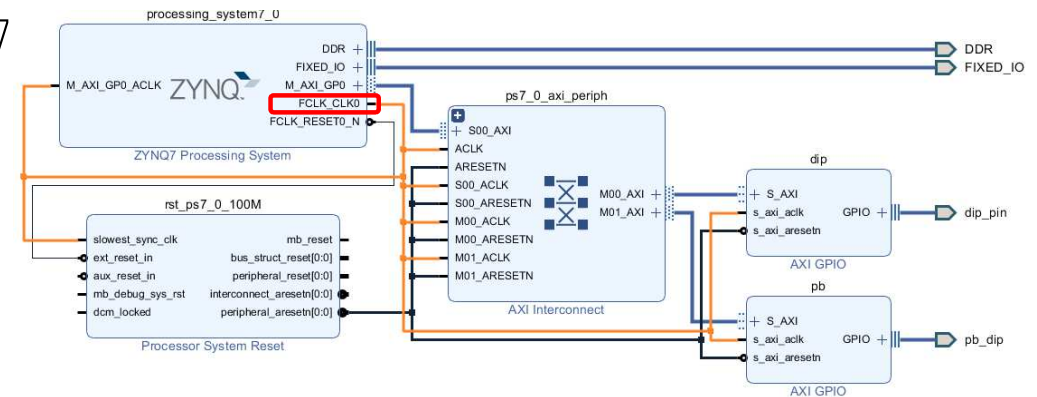
Blokk Diagram áttekintése



• 1.) Kérdés – Megoldás

- Mely IP periféria példányokhoz kapcsolódik a processing_system7_0_FCLK_CLK0 órajel

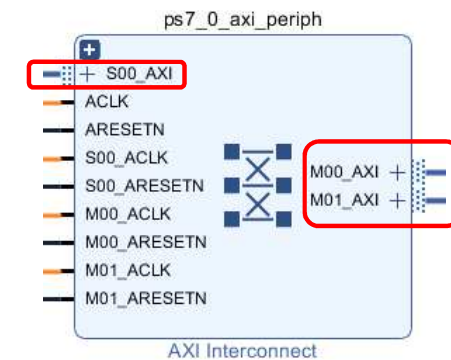
- processing_system7 (visszacsatolás)
- rst_processing_system7
- dip
- pb
- ps7_axi_periph



- processing_system7_0_FCLK_CLK0 órajel 100 MHz!
(ZynqPS -> Clock Configuration -> PL Fabric Clock)

- Mely IP periféria példányok kapcsolódnak az ps7_axi_periph AXI interfész M/S_AXI portjaihoz:

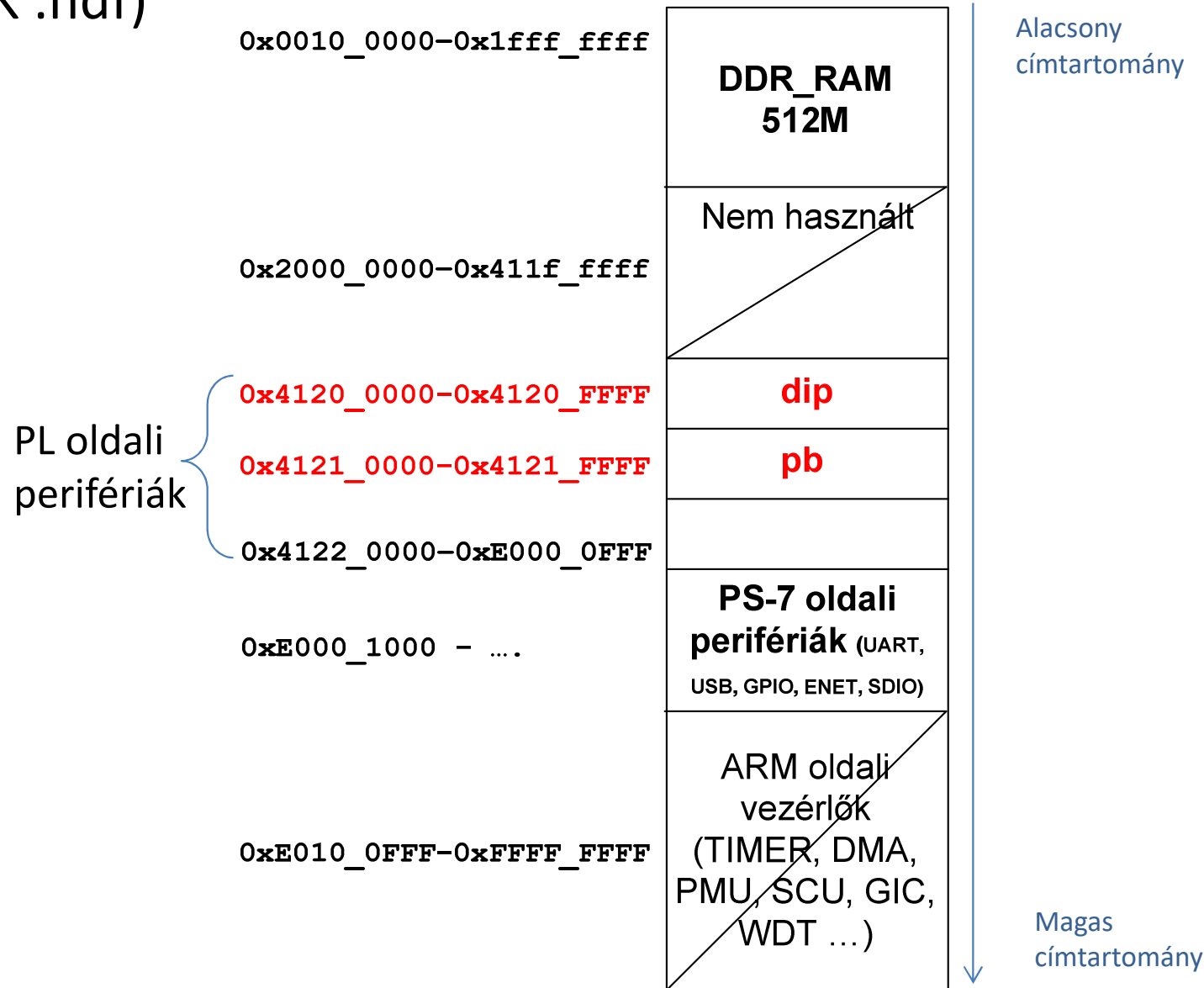
- processing_system7
- dip
- pb



Blokk Diagram áttekintése



- 2.) Kérdés - Megoldás (Block Diagram → Address Editor /vagy SDK .hdf)



Reports (implementáció után)

- 3.) Kérdés – Megoldás: Mennyi erőforrást foglalunk el PL oldalon?
 - Reports -> Report Utilization (vagy Project Summary Σ)

Site Type	Used	Fixed	Available	Util%
Slice LUTs	596	0	17600	3.39
LUT as Logic	534	0	17600	3.03
LUT as Memory	62	0	6000	1.03
LUT as Distributed RAM	0	0		
LUT as Shift Register	62	0		
Slice Registers	819	0	35200	2.33
Register as Flip Flop	819	0	35200	2.33
Register as Latch	0	0	35200	0.00
F7 Muxes	0	0	8800	0.00
F8 Muxes	0	0	4400	0.00

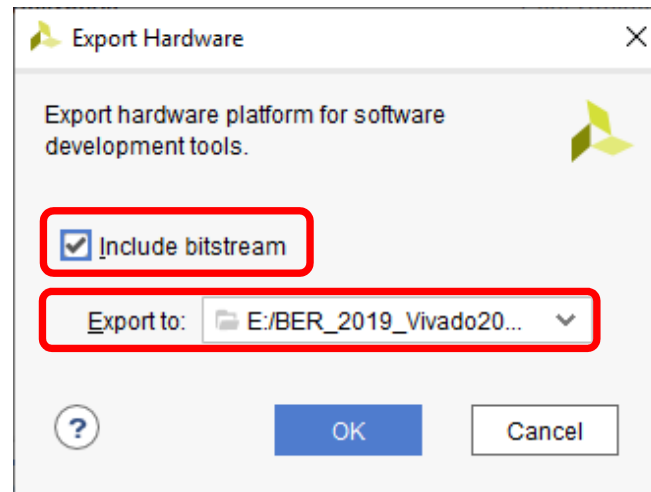
Xilinx Vivado SDK használata

SZOFTVER TESZT ALKALMAZÁS ÖSSZEÁLLÍTÁSA

- 
- A large blue arrow pointing downwards, indicating the sequence of steps.
1. Vivado projekt létrehozása, majd Export → **SDK**,
 2. **Új**, vagy C/C++ template-ből generálható alkalmazás létrehozása (**TestApp** periféria-teszt alkalmazás) :
 - a) **BSP** (Board Support Package) generálása és fordítása,
 - b) **Linker Script** generálása (memória szekciók megadása, .ld),
 - c) **SW** alkalmazáskód megírása/generálása, és fordítása.
 3. Soros terminál beállítása (USB-soros port beállítása),
 4. JTAG-USB programozó csatlakoztatása és beállítása,
 5. *FPGA konfigurálása! (.bit: PL oldal konfigurációja!)*
 6. *'debug' – hardveres hibakeresés beállítása*
 7. Debug (breakpoint-ok beszúrása, léptetés, stb.)

Export HW -> Vivado SDK

- **File -> Export -> Export hardware**



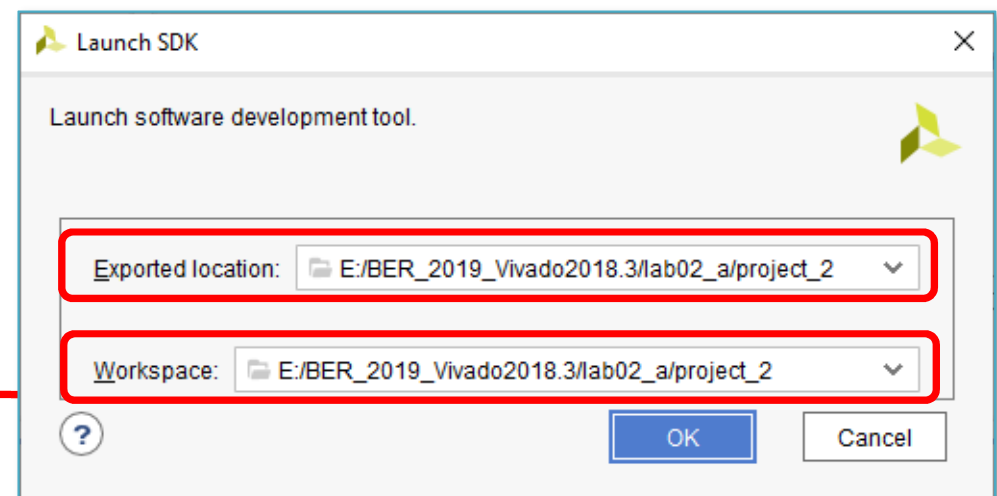
Mivel PL (FPGA) oldal is konfigurálva lett, ezért most már kell a bitstream is az SDK-hoz.

- Include bitstream kiválasztása
- Export to: elérési út kiválasztása (**Choose location**)
Export to: lab02_A

- **File -> Launch SDK, OK.**

– Vivado SDK indítása a keretrendszerből (ezt kívülről indítva is meg lehetne tenni)

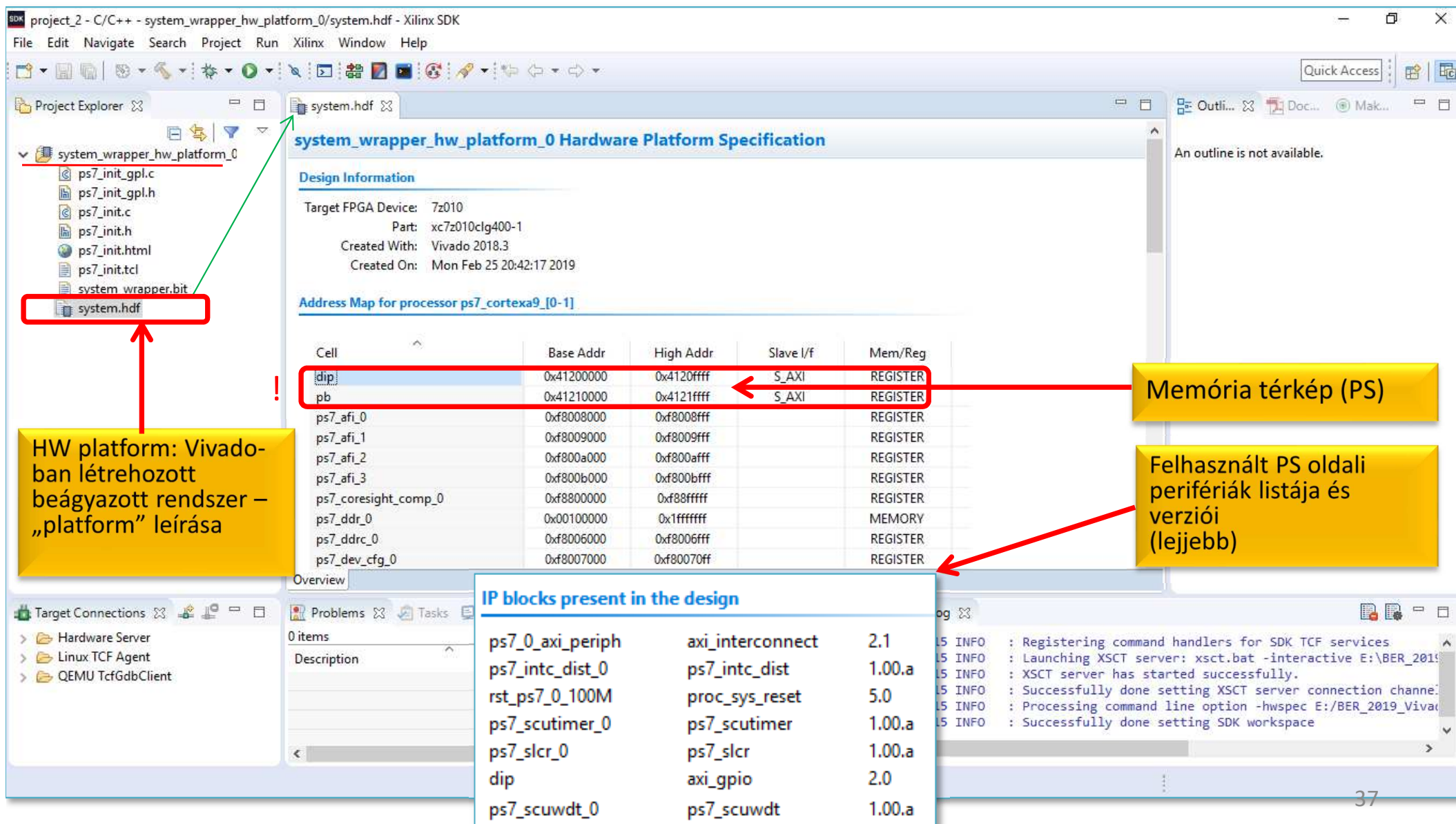
Fontos, hogy a workspace-t lehetőleg az aktuális Vivado projektbe hozza létre - > **Choose location...** – és mostani **lab02_a** kiválasztása. **OK.**



Vivado SDK

.hdf = Hardware description file = hardver konfigurációs csomag file.

(Valójában egy zip, amely két leíró xml-t tartalmaz!) Ebből olvassa ki az SDK a beállításokat.



The screenshot shows the Vivado SDK interface with the following components:

- Project Explorer:** Shows the project structure for `system_wrapper_hw_platform_0`. The `system.hdf` file is highlighted with a red box and a red arrow pointing to it.
- system_wrapper_hw_platform_0 Hardware Platform Specification:**
 - Design Information:**
 - Target FPGA Device: 7z010
 - Part: xc7z010clg400-1
 - Created With: Vivado 2018.3
 - Created On: Mon Feb 25 20:42:17 2019
 - Address Map for processor ps7_cortexa9 [0-1]:**

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
dip	0x41200000	0x4120ffff	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	0xf8800000	0xf880ffff		REGISTER
ps7_ddr_0	0x00100000	0x1fffffff		MEMORY
ps7_ddrc_0	0xf8006000	0xf8006fff		REGISTER
ps7_dev_cfg_0	0xf8007000	0xf80070ff		REGISTER
- IP blocks present in the design:**

Block Name	Component	Version
ps7_0_axi_periph	axi_interconnect	2.1
ps7_intc_dist_0	ps7_intc_dist	1.00.a
rst_ps7_0_100M	proc_sys_reset	5.0
ps7_scutimer_0	ps7_scutimer	1.00.a
ps7_slcr_0	ps7_slcr	1.00.a
dip	axi_gpio	2.0
ps7_scuwdt_0	ps7_scuwdt	1.00.a

Annotations:

- HW platform: Vivado-ban létrehozott beágyazott rendszer – „platform” leírása** (points to `system.hdf`)
- Memória térkép (PS)** (points to the Address Map table)
- Felhasznált PS oldali perifériák listája és verziói (lejjebb)** (points to the IP blocks table)

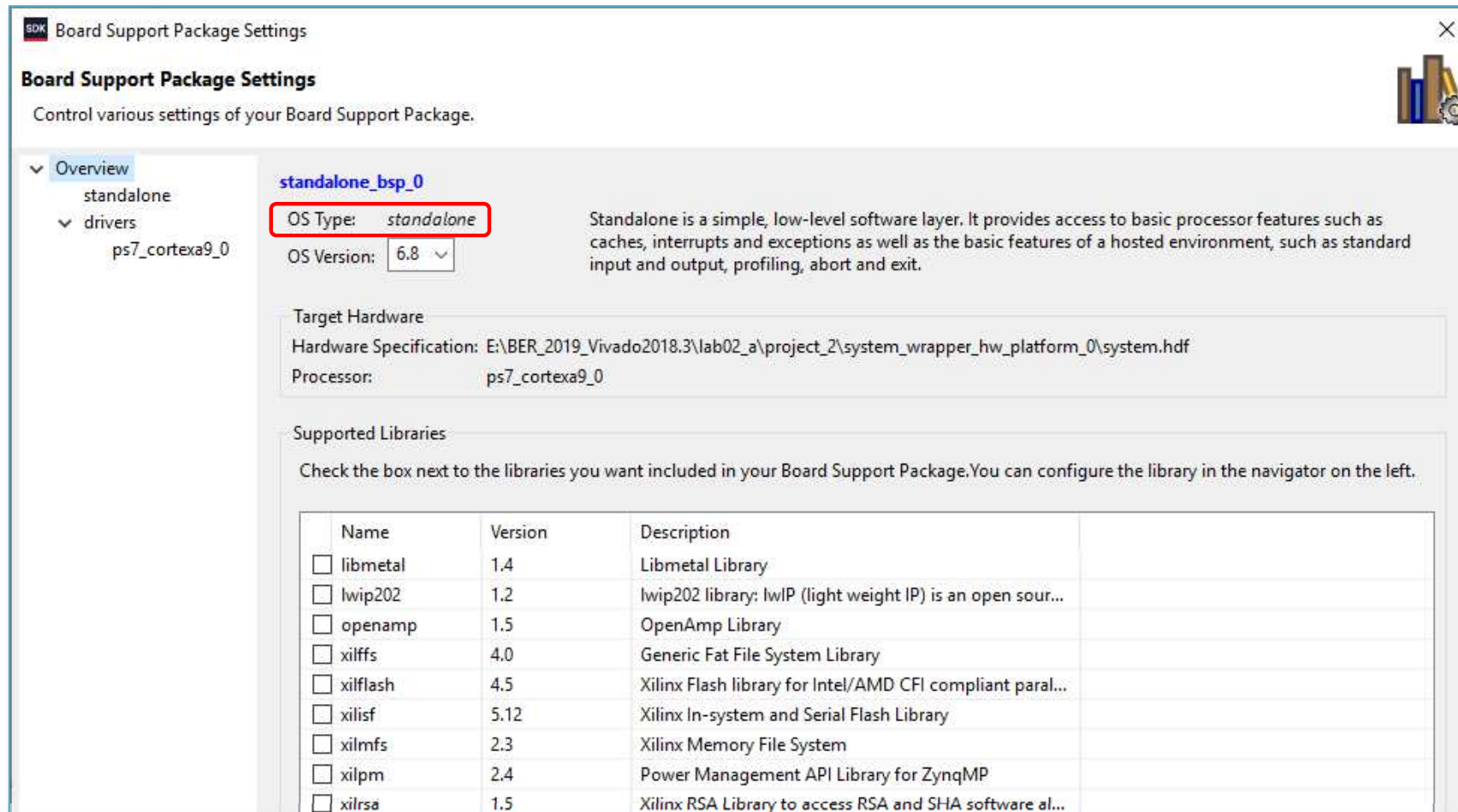
- 1.) A Launch SDK - **Choose Location...** opciójával válasszuk ki a `<project_dir>/LAB02_A/` → majd kattintsunk az **OK**-ra.
- 2.) Az SDK-ban: **File** → **New** → **Xilinx Board Support Package** → kattintsunk a **Finish**-re.
 - Új BSP neve `standalone_bsp_0` is lehet akár,
 - Hagyjunk mindent alapértéken.

Tesztalkalmazás (TestApp) készítése II.



- **Software Platform Settings**

- Operációs rendszer kiválasztása: *standalone* vs. *freertos_10* (esetleg 3rd Party OS)
- Rendelkezésre álló könyvtári függvények (lib) kiválasztása

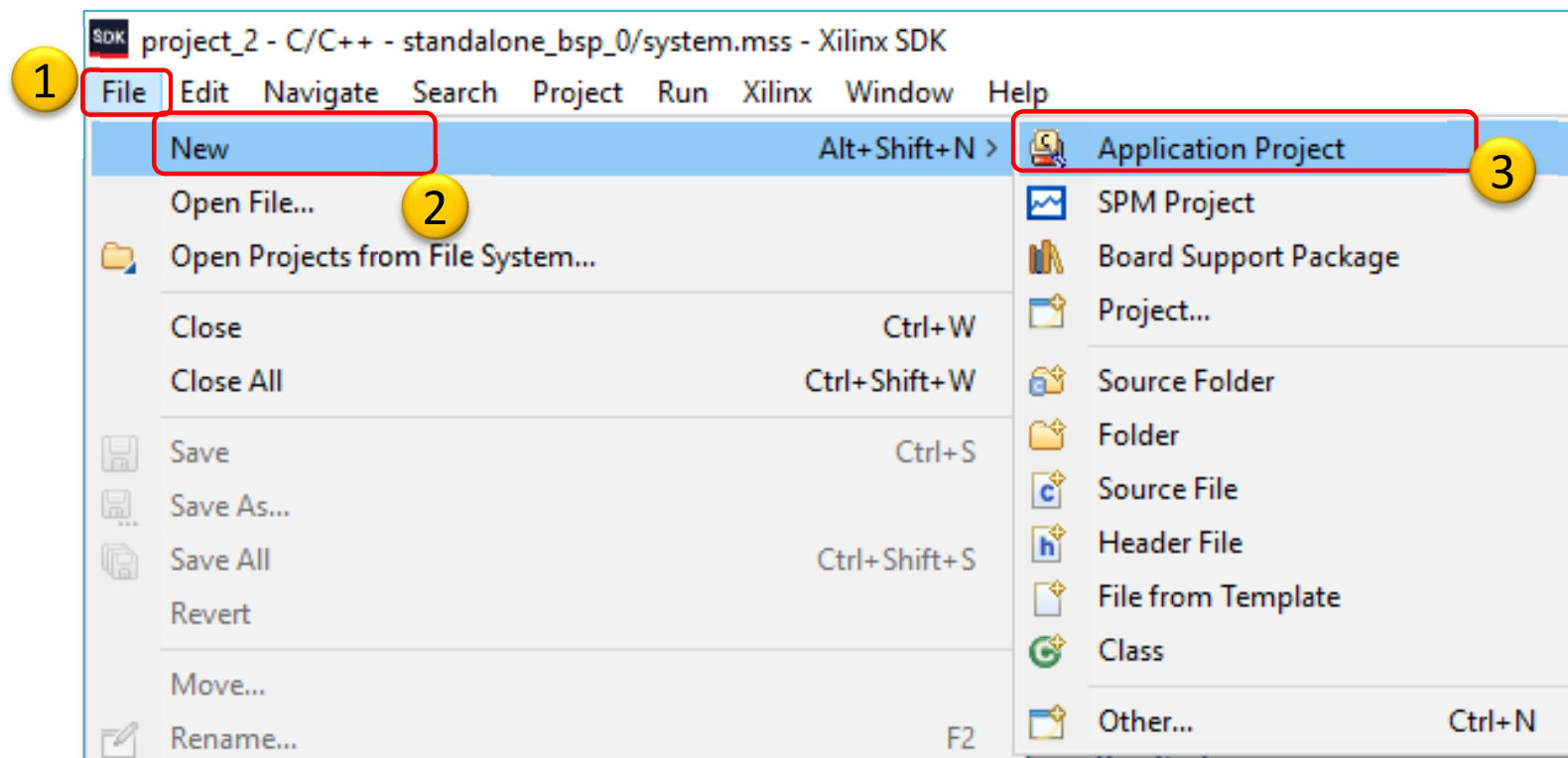


OK -> Legenerálódik az **xparameters.h** és további mappák.

Helye: <projectdir>/lab02/standalone_bsp_0/ps7_cortexa9_0/include mappa

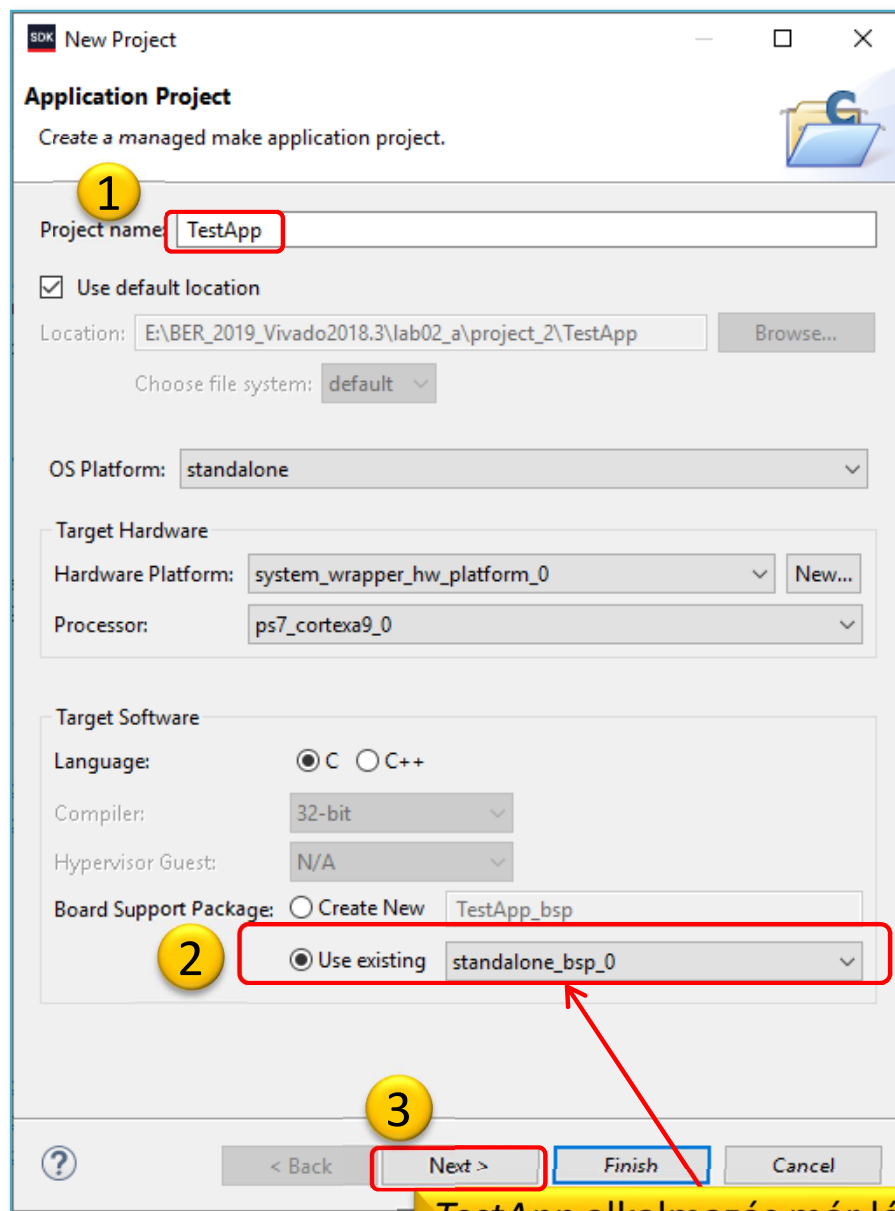
Tesztalkalmazás (TestApp) készítése III.

- Új projekt alkalmazás létrehozása
(File → New → Application Project)



BSP: *standalone_bsp_0*
(alapértelmezett)

Tesztalkalmazás (TestApp) készítése IV.



SDK New Project

Application Project

Create a managed make application project.

1 Project name: **TestApp**

☒ Use default location

Location: E:\BER_2019_Vivado2018.3\lab02_a\project_2\TestApp

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: system_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: ☒ C ☐ C++

Compiler: 32-bit

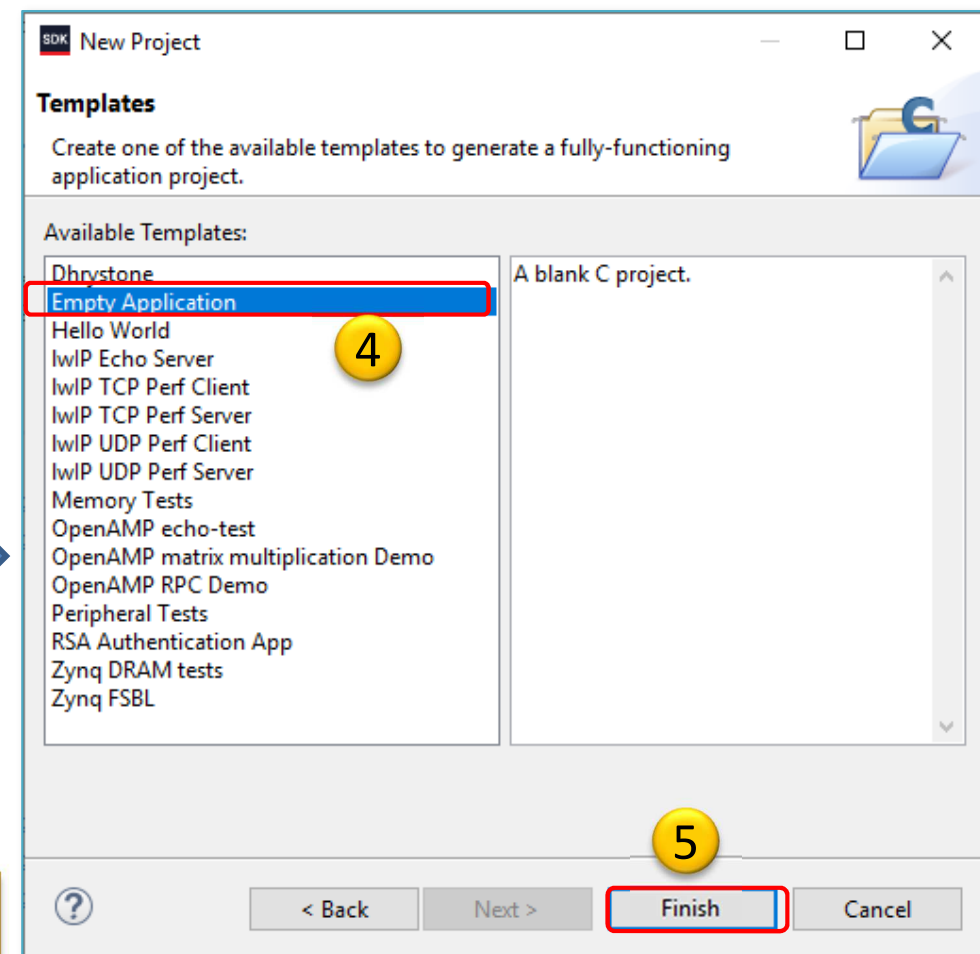
Hypervisor Guest: N/A

Board Support Package: ☐ Create New TestApp_bsp

2 ☒ Use existing standalone_bsp_0

3 Next >

Projekt neve: *TestApp*



SDK New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Dhrystone
- 4 Empty Application
- Hello World
- IwIP Echo Server
- IwIP TCP Perf Client
- IwIP TCP Perf Server
- IwIP UDP Perf Client
- IwIP UDP Perf Server
- Memory Tests
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

A blank C project.

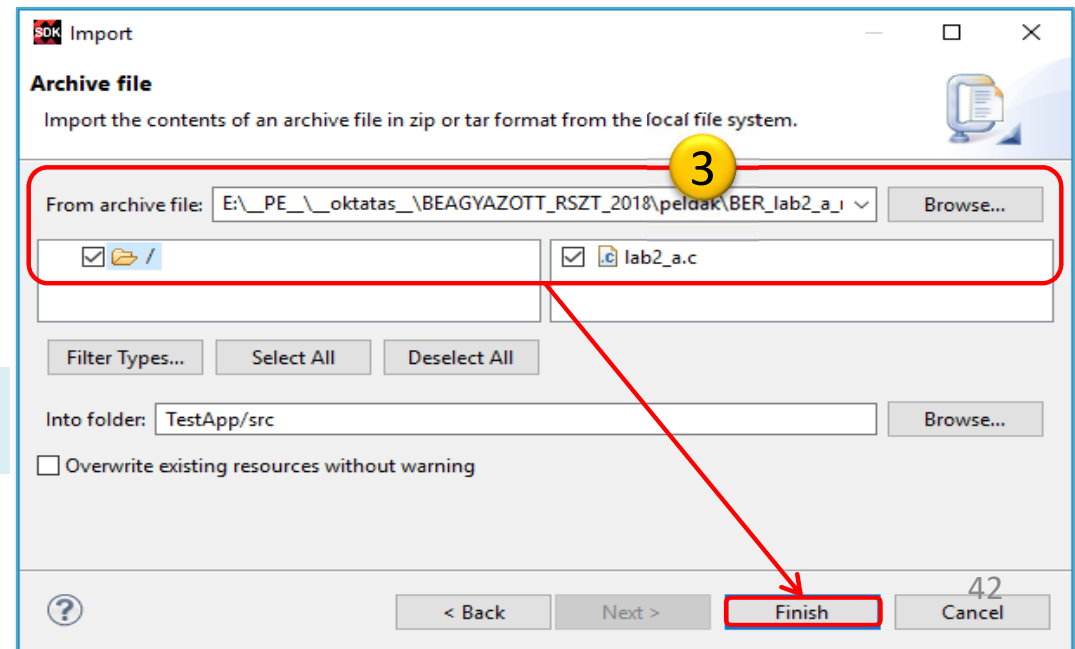
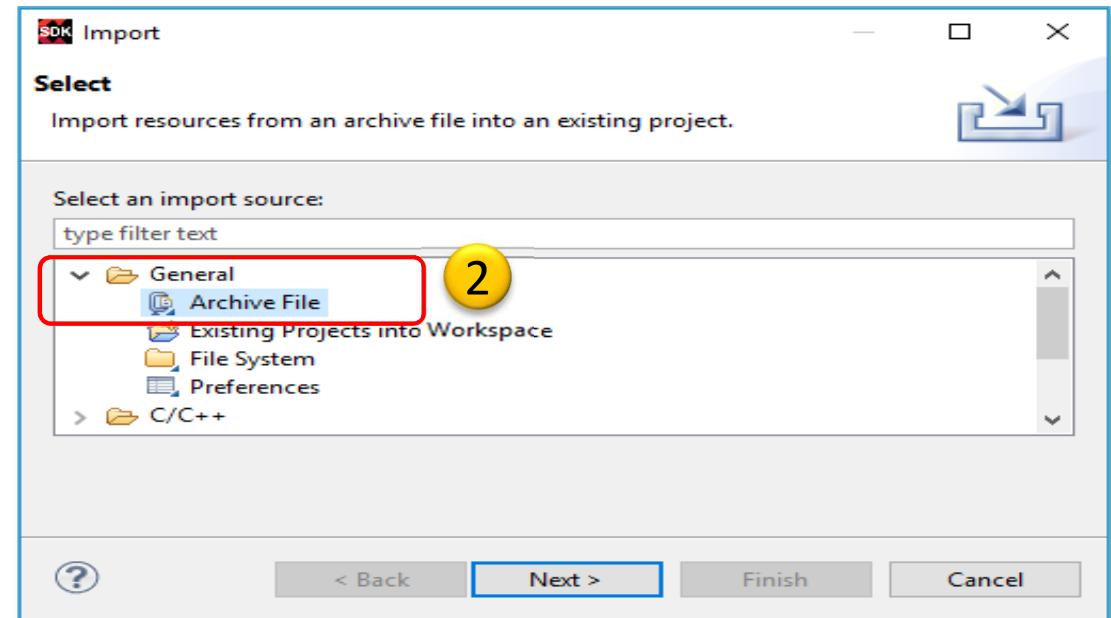
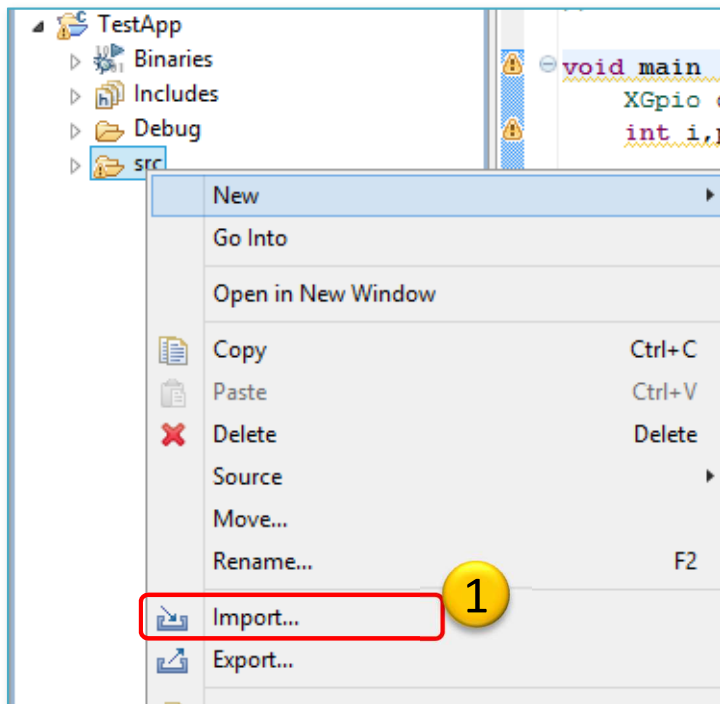
5 Finish

TestApp alkalmazás már létező, korábban létrehozott BSP-hez rendelése (*standalone_bsp*).
NEXT >>

Tesztalkalmazás (TestApp) készítése V.



- Új C forrás fájl létrehozása/**hozzáadása** (lab2_a.c):



TestApp -> Src -> (jobb gomb)
Import... -> General -> Archive file...
Végül Browse...

[BER lab2 a main TestApp.zip](#)

Tesztalkalmazás (TestApp) készítése VI.



- Forráskód hozzáadása:

```
#include "xparameters.h"
#include "xgpio.h"
#include "xutil.h"
//-----
int main (void){
    XGpio dip, push;
    int psb_check, dip_check;
    volatile unsigned int i;

    //printf("-- Start of the Lab02 Program --\r\n");
    xil_printf("-- Start of the Lab02 Program --\r\n");
    //int XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId);
    XGpio_Initialize(&dip, XPAR_DIP_DEVICE_ID);
    //void XGpio_SetDataDirection(XGpio *InstancePtr, unsigned Channel, u32 DirectionMask);
    XGpio_SetDataDirection(&dip, 1, 0xFFFFFFFF);

    XGpio_Initialize(&push, XPAR_PB_DEVICE_ID);
    XGpio_SetDataDirection(&push, 1, 0xFFFFFFFF);

    while(1){
        //u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
        psb_check = XGpio_DiscreteRead(&push, 1);
        xil_printf("Push button status: %x\r\n", psb_check);

        dip_check = XGpio_DiscreteRead(&dip, 1);
        xil_printf("DIP switch status: %x\r\n", dip_check);

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

BSP fordítása (build) után, ha a függvény() nevén, vagy header fájlok nevén (F3-at) vagy jobb gomb-ot nyomunk, majd „Open declaration”, akkor:

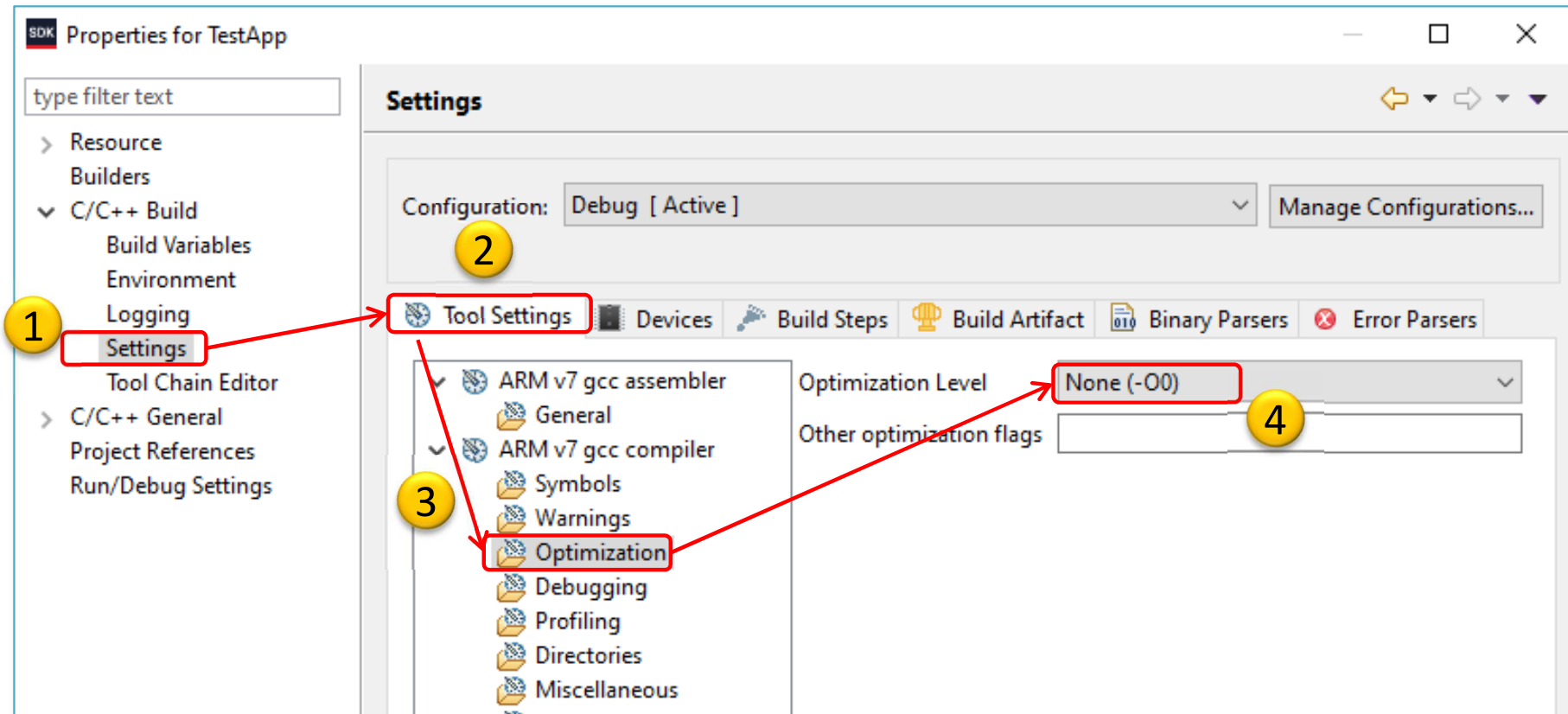
- Könyvtári függvények leírásai, ill. deklarációi elérhetőek lesznek.

[BER lab2 a main TestApp.zip](#)

Tesztalkalmazás (TestApp) készítése VII.



- Fordító beállítása
 - Jobb kattintás a **TestApp**-ra → **C/C++ Build Settings**

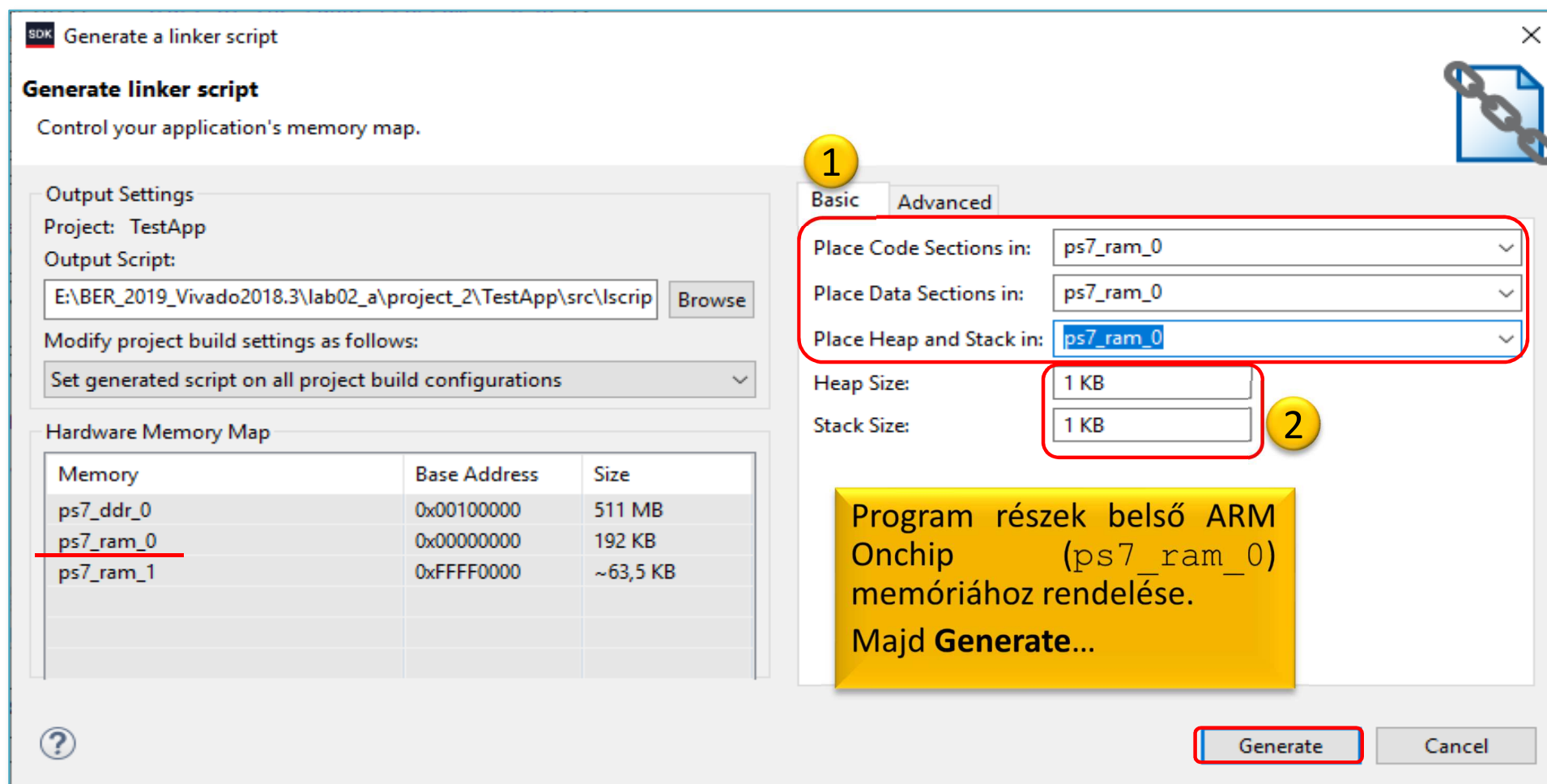


- Fontos: Mivel a programunkban van egy **for()** ciklus is, mely a késleltetésért felel, és nem szeretnénk, hogy a fordító „kioptimalizálja”, ezért állítsuk az optimalizálást **(-O0) = None** szintűre.

Tesztalkalmazás (TestApp) készítése VIII.



- Linker Script létrehozása
 - Jobb kattintás a **TestApp**-ra → **Generate Linker Script**



Tesztalkalmazás (TestApp.elf)



- 1.) Kérdés: Build-elés során mekkora lett az TestApp.elf file mérete ? Milyen a mértékegysége?

– Megoldás: 32684 byte (ha optimization –O0)

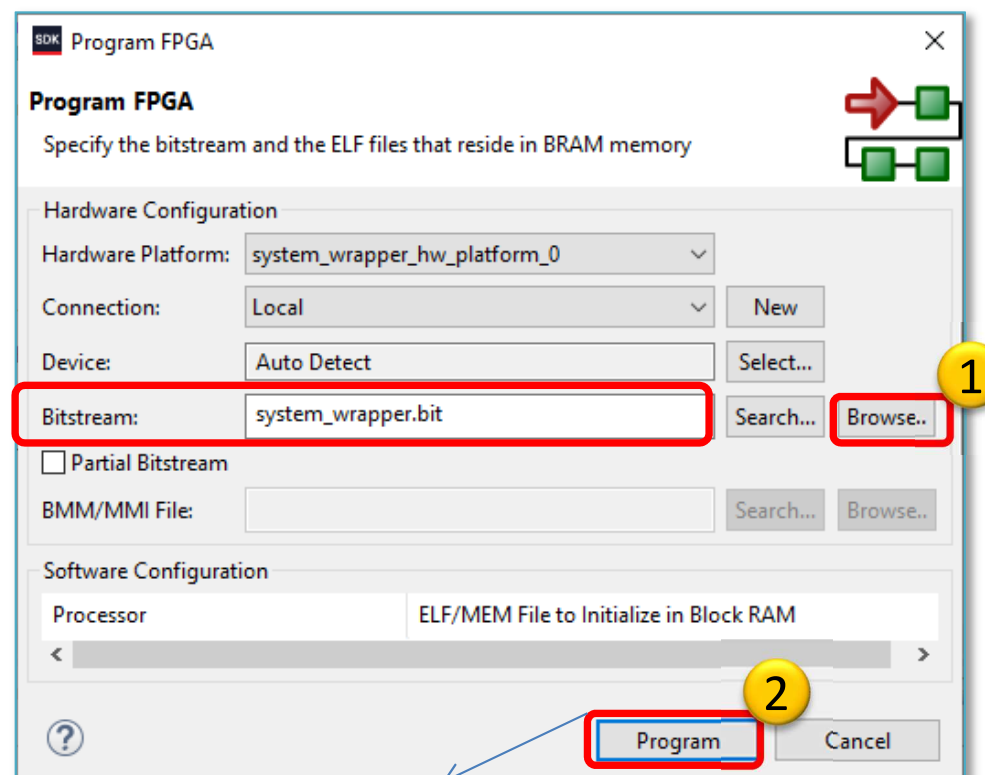
```
'Finished building target: TestApp.elf'
'
'Invoking: ARM v7 Print Size'
arm-none-eabi-size TestApp.elf |tee "TestApp.elf.size"
  text      data      bss      dec      hex filename
 23432     1184     8248    32864    8060 TestApp.elf
'Finished building: TestApp.elf.size'
```

Megjegyzés: érdemes kikapcsolni az automatikus buildelést is, mivel minden forráskód módosítás és mentés után újragenerálja az alkalmazás .elf –et.

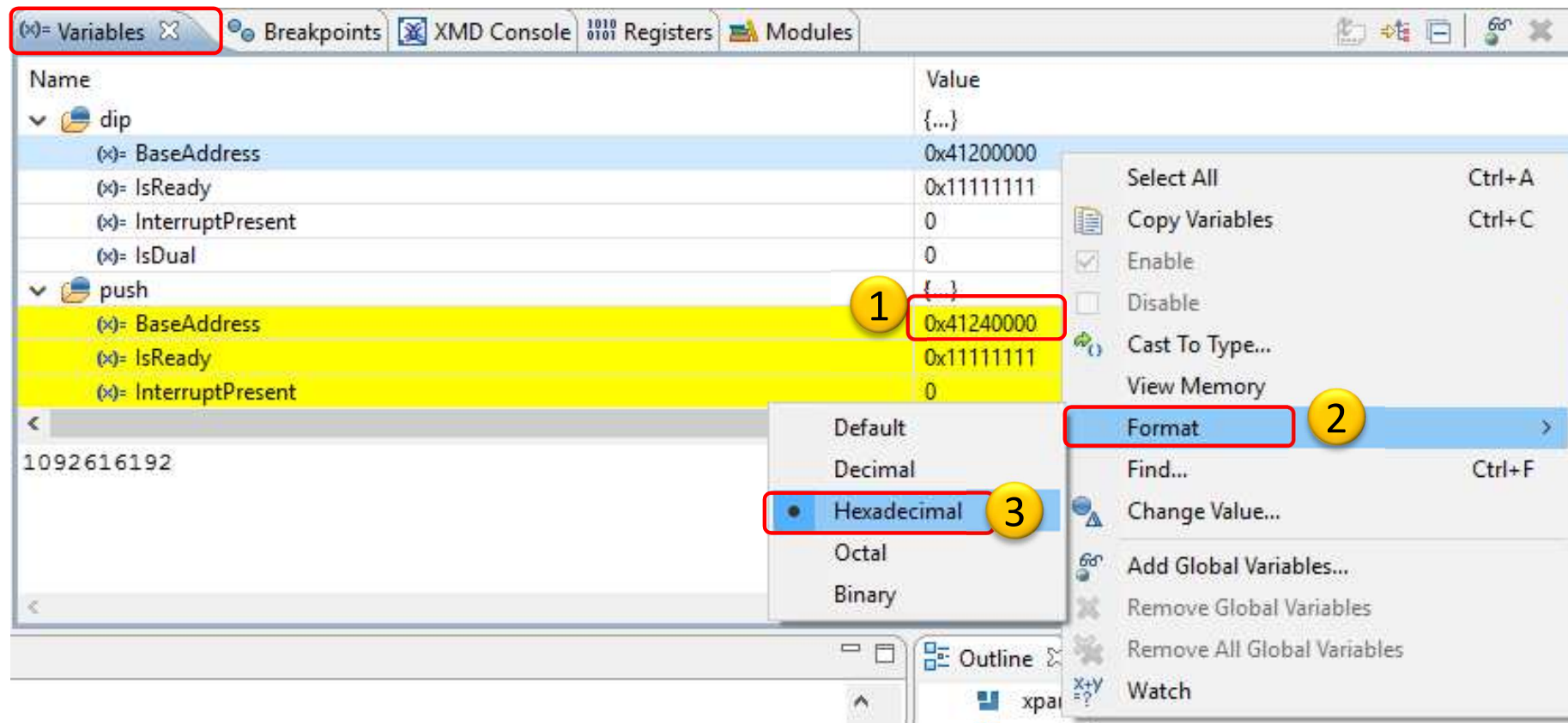
Lépés: Project Menü -> Build Automatically kikapcsolása

Program FPGA – bitstream letöltés

- A JTAG-USB programozó (USB-soros) kábel csatlakoztatása a számítógéphez és a kártyához,
- Debug konfiguráció létrehozása (lásd lab01): a SDK Terminal-on az USB soros port beállítása (COMx), majd Csatlakozás...
- Digilent ZyBo kártya bekapcsolása, FPGA (PL) bitstream letöltése 
 - Xilinx Tools → Program FPGA (.bit fájl kiválasztása)



INFO: FPGA configured successfully with bitstream
"D:/__PE__/BER_2018_Vivado/lab02/project_1/system_wrapper_hw_platform_0/system_wrapper.bit"



Debug során a Variables ablakban állítsuk a pb / push változók BaseAddress/IsReady paramétereit Hexadecimális értékűre!

- Kapcsolók (*dip*), és nyomógombok (*push*) működésének verifikációja (Peripheral_test template alapján készült)

Tegyünk be brake-point-okat!

```
while(1){  
    //u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);  
    psb_check = XGpio_DiscreteRead(&push, 1);  
    xil_printf("Push button status: %x\r\n", psb_check);  
  
    dip_check = XGpio_DiscreteRead(&dip, 1);  
    xil_printf("DIP switch status: %x\r\n", dip_check);  
  
    for(i = 0; i < 1000000; i++); //delay  
}  
return 0;
```

Console

Tasks

Terminal

Problems

Executables

Memory

TestApp Debug [Xilinx C/C++ application (GDB)] D:\BER_zybo_bsb\LAB02\workspaces\TestApp\Debug\TestApp.elf (29/03/201

-- Start of the Lab02 Program --

Push button status: 0

DIP switch status: F

Push button status: 4

DIP switch status: F

Push button status: 4

DIP switch status: E

Push button status: 4

DIP switch status: C

* A példában while(1) ciklus van használva a folyamatos lekérdezéshez!

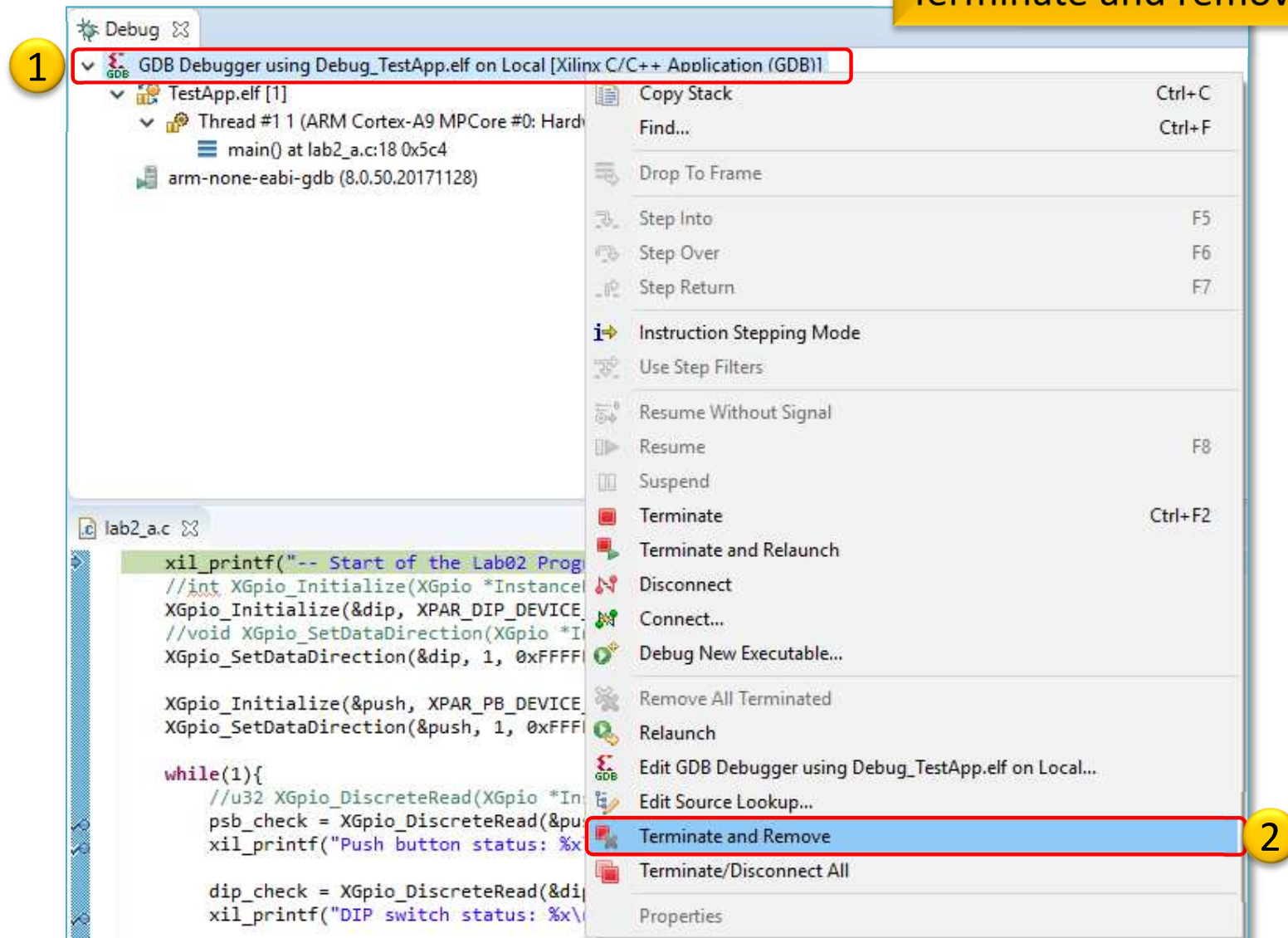
Tapasztalat?

LITTLE ENDIAN!

Jótanács: Debug terminálása

- Debuggolás végén: terminate

Debug → Jobb gomb →
Terminate and remove !!



Feladat 2: általános Periféria Teszt

- A beágyazott rendszeren futtassuk le a template-ből generálható *Peripheral Test-et*. (ha benne van, kommentezzük ki az Ethernet részeket!)
- Készítsük el annak Debug konfigurációját is, illetve Debug-oljuk az alkalmazást.

```
---Entering main---

Running ScuGicSelfTestExample() for ps7_scugic_0...
ScuGicSelfTestExample PASSED
ScuGic Interrupt Setup PASSED

Running GpioInputExample() for pb...
GpioInputExample PASSED. Read data:0x0

Running DcfgSelfTestExample() for ps7_dev_cfg_0...
DcfgSelfTestExample PASSED

Running XDmaPs_Example_W_Intr() for ps7_dma_s...
Test round 0
XDmaPs_Example_W_Intr PASSED

Running GpioInputExample() for dip...
GpioInputExample PASSED. Read data:0xC

Running ScuTimerPolledExample() for ps7_scutimer_0...
ScuTimerPolledExample PASSED

Running Interrupt Test for ps7_scutimer_0...
ScuTimerIntrExample PASSED

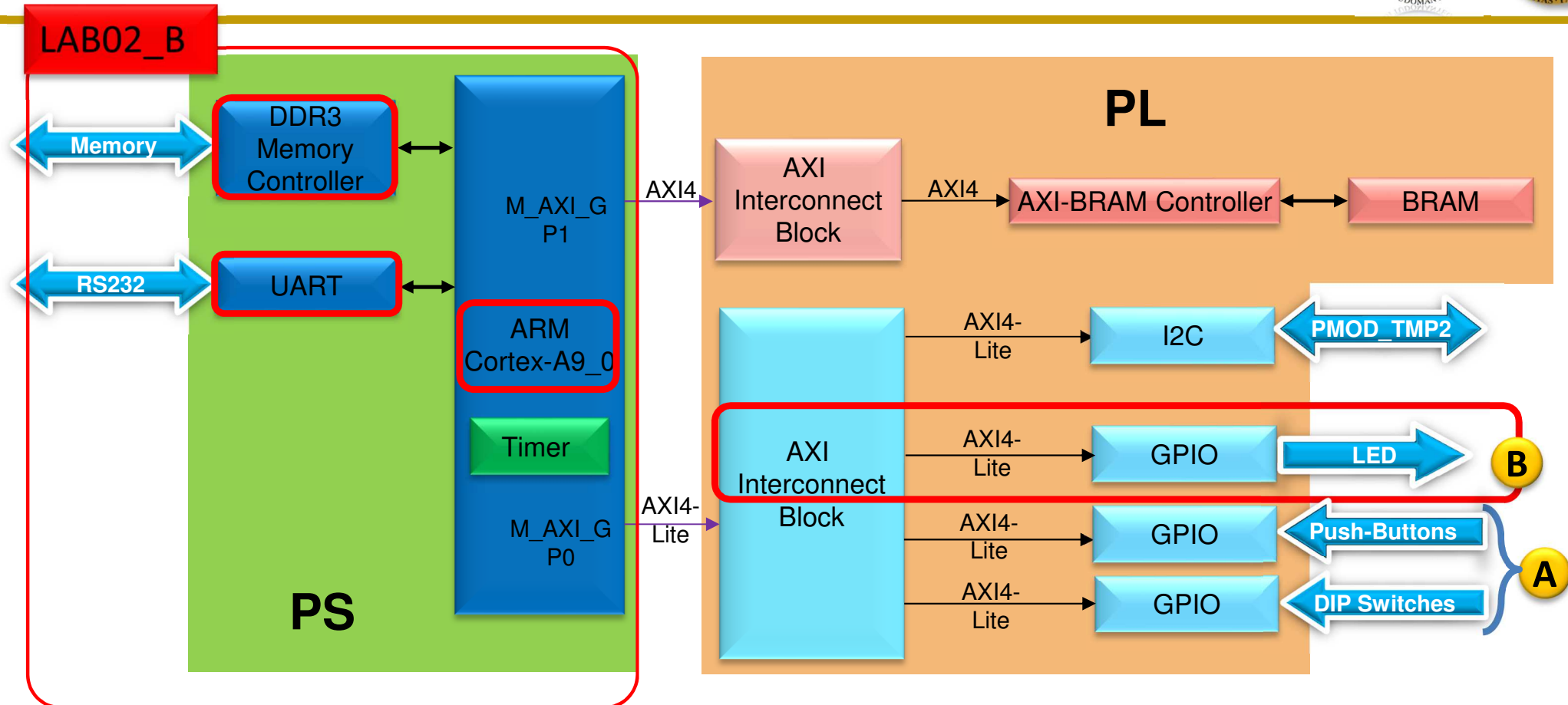
Running Interrupt Test for ps7_scuwdt_0...
ScuWdtIntrExample PASSED
---Exiting main---
```

- Az előző (3.) laboron létrehozott ARM-AXI alaprendszerhez hozzáadtunk két új PL oldali AXI *GPIO* perifériát a Vivado IP katalógusából.
- A perifériákat a megfelelő módon konfiguráltuk, és hozzákapcsoltuk az FPGA külső lábaihoz (pin).
- Ezután *megvizsgáltuk* a Blokk Diagramm-ot és a riport fileokat.
- Az lábkiosztásokhoz hozzárendeltük a ZyBo kártyán lévő DIP kapcsolókat (4), ill. PB nyomógombokat (4).
- Végül verifikáltuk az elkészült beágyazott rendszert és a SW alkalmazás működését (Periféria Teszt).

LAB02_B. LED vezérlő

BEÁGYAZOTT RENDSZER ÉS SZOFTVER ALKALMAZÁS ÖSSZEÁLLÍTÁSA

A bővíthető tesztrendszer



PS oldal:

- ARM hard-processzor mag
- belső OnChip-RAM vezérlő
- RS232 soros interfész
- külső DDR3 memória vezérlő
- I2C vezérlő

PL oldal:

- (A) LAB02_A: GPIO bemenetek
 - PBSs: Push Button (nyomógomb kezelő)
 - DIPs: Switches (kapcsoló kezelő)
- **(B) LAB02_B: GPIO kimenet**
 - **LEDs: LED kijelző**

Példa 1: LED vezérlő hozzáadása



Lépések:

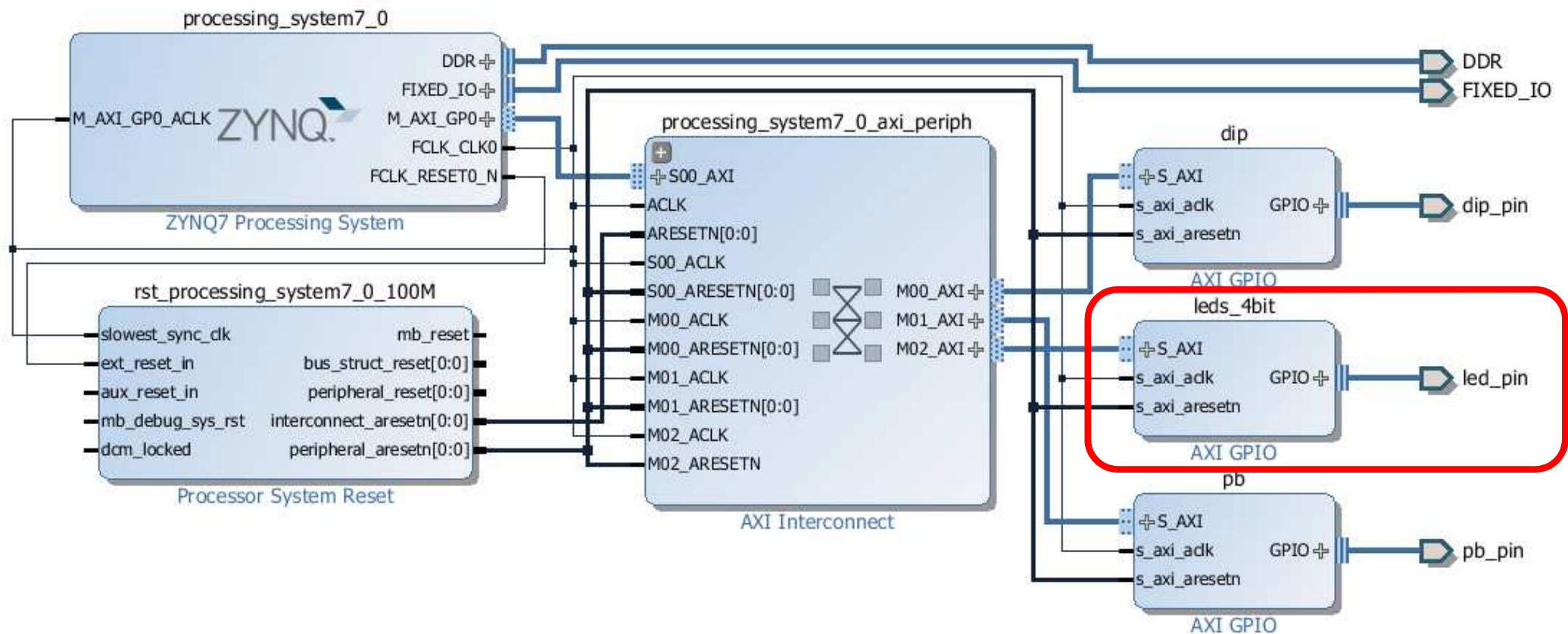
- A LAB02_A ismeretkör elsajátítása során létrehozott projekt archiválása,
 - átmásolása (LAB02_B néven), majd megnyitása a Xilinx Vivado-ban.
 - Alternatív megoldásként a File -> Save Project As... használható.
- LED vezérlő: a Vivado **IP katalógusából kiválasztott AXI GPIO periféria** integrálása és összekötése az alaprendszerrel (4-bites, mivel 4 LED van),
 - Base Addr: 0x4123_0000 (méret: 64 K)
 - IP példány neve legyen „leds_4bit”. Output irányú - 4 bites.
- Külső GPIO LED portok FPGA lábakhoz rendelése,
 - Külső láb neve legyen: led_pin
- Block Design vizsgálata, DRC, majd Bitstream generálás!

SDK: (HW Export után)

- Egy Periféria teszt-alkalmazás (PeriphTest) készítése a Xilinx SDK-környezetben (template-ből generálható),
- Debug Configuráció: a FW-SW tervek teszt verifikálása a ZyBo kártyán.

LED – GPIO megoldás

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
leds_4bit	S_AXI	Reg	0x4123_0000	64K	0x4123_FFFF



Példa 2.) LED-es számláló

- Módosítsa az előző Példa 1.) szereplő *Periféria Teszt SW* alkalmazását úgy, hogy a LED-eket egy **4-bites bináris számláló** értékének növelésével egymás után villantsa fel.
- **Segítség:** **BER_lab2b_led4bit_count.zip**
 - Használja a beépített pl. `u8` adattípust
 - Mivel `sys_clk = 100 MHz`, késleltesse a LED-ek fel-/le-villanását (pl. `for()` ciklussal), úgy hogy a felvillanások ideje érzékelhető legyen.
 - `xparameters.h` fileból a makrók használata, fordítási hiba esetén (pl. re-defining)
 - `LED_DELAY,` `GPIO_BITWIDTH` beállítása!
(`xgpio_tapp_example.c`)

Példa 3.) LED-es fényfüzér



- Módosítsa az előző Példa 2.) szereplő *Periféria Teszt SW* alkalmazását úgy, hogy a LED-ek értékét mindig egy pozícióval shifteli balra (növekvő bináris súlyú számláló)

Segítség: **BER_lab2b_led4bit_shift.zip**

Példa 4.) „Knight Rider”-es fényfüzér



- Módosítsa az előző Példa 3.) szereplő *Periféria Teszt SW* alkalmazását úgy, hogy a LED-ek értékét mindig egy pozícióval shift-eli balra, majd amikor eléri a végértéket visszafelé, jobbra shift-eli (növekvő, illetve csökkenő bináris súlyú számláló segítségével)

Segítség: **[BER_lab2b_led4bit_knightrider.zip](#)**

Példa 5.) elemi „Calculator”

- Módosítsa a korábbi példát úgy, hogy egy 4-alapműveletre képes számológépet kapjunk. Két operandus (A,B) egyenként 2-2 bites: $A[1:0]$, $B[1:0]$, melyek a **dip** kapcsolók értékeit jelentik.
- Nyomógombok (**pb**) segítségével legyenek a következő műveletek:
 - $Pb[0] = '1'$: összeadás, $Pb[1] = '1'$: kivonás, $Pb[2] = '1'$: szorzás, $Pb[3] = '1'$: osztás.
 - Ezek eredményeit - az osztás kivételével - jelenítse meg a **LED**[3:0]-n (osztás esetén `xprintf()`-el)