

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. szeptember 21.

Beágyazott rendszerek fejlesztése (FPGA)

Dr. Vörösházi Zsolt

- 3. Beágyazott alap tesztrendszer (BSB - Base System Builder)
összeállítása
(Xilinx Vivado – Embedded Development Kit / Software
Development Kit)**



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 alrendszerhez. Saját periféria hozzáadása az összeállított beágyazott alrendszerhez
5. Szoftver alkalmazások fejlesztése, tesztelése, hibakeresése (debug) Xilinx Vivado SDK (Software Development Kit) használatával

Mire figyeljetelek? (jótanácsok)



- Figyeljünk arra, hogy a létrehozandó Vivado projekt elérési útja **NE** tartalmazzon **ékezetet**, illetve ún. „white-space” karaktereket!
- Megfelelő jogosultságunk legyen azon a meghajtón amin dolgoztok:
 - Hálózati meghajtóra lehetőség szerint **NE** dolgozzunk!
- A projekt neve, és a források nevei **NE** kezdődjenek számmal, de tartalmazhatnak számot! (VHDL)
- Használjuk következetesen a kis-nagy betűket egy forrás fájlban, és a projekten belül!
- Lehetőség szerint a projekt könyvtárának neve, projekt neve és a forrás(ok) neve legyen eltérő, és utaljon azok funkciójára, a hibaüzenetekben szereplő könnyebb azonosítás végett!

- Ahhoz, hogy a **ZyBo HW** platform, kiválasztható legyen a *Vivado fejlesztő* környezetben, a következő lépéseket kell megtenni (csak első alkalommal):
- 1.) A kártya gyártójának megfelelő támogató csomagot letölteni:

Digilent ZYBO

- Vivado-hoz a támogató csomag(**):



Letölthető gyakorlati anyag

Digilent ZyBo Fejlesztőkártyákhoz (BSP, XDC):

XDC (FPGA lábkiosztás - GIT master):
[Zybo-Master.xdc](#)

Base System Pack (BSP):
[Vivado Board Files \(2015.2\)](#)

Digilent Zybo hivatalos weboldal:
[Zybo Zynq-7000 ARM/FPGA SoC Trainer Board](#)

Xilinx Vivado telepítési útmutató:
 [TM_Xilinx_Vivado_telepitési_utmutato_1.0](#)

ZYBO platform Xilinx Vivado alatt



- ** A gyártó oldaláról is letölthető:
 - <https://github.com/Digilent/vivado-boards/archive/master.zip>
- A csomagban lévő
`\vivado-zybo-master\new\board_files\zybo` könyvtár
átmásolása a Xilinx könyvtárba a következő helyre:
 - ÁTMÁSOLNI →
`C:\Xilinx\Vivado\2015.x\data\boards\board_files`
könyvtárba.

Xilinx Vivado Design Suite használata

BEÁGYAZOTT RENDSZER ÖSSZEÁLLÍTÁSA

- Egy új projekt létrehozása a *Xilinx Vivado-ban*
 - a **BSB** = *Base System Builder* - beágyazott alaprendszer összeállítása (*Block Designer*),
 - Egyszerű **ARM** / **AXI** alapú beágyazott alaptesztrendszer (BSB) létrehozása,
 - **AXI-Lite** "busz" interfész alapú Xilinx **IP**-k (Intellectual Property = gyártó szellemi termékeinek) hozzáadása.

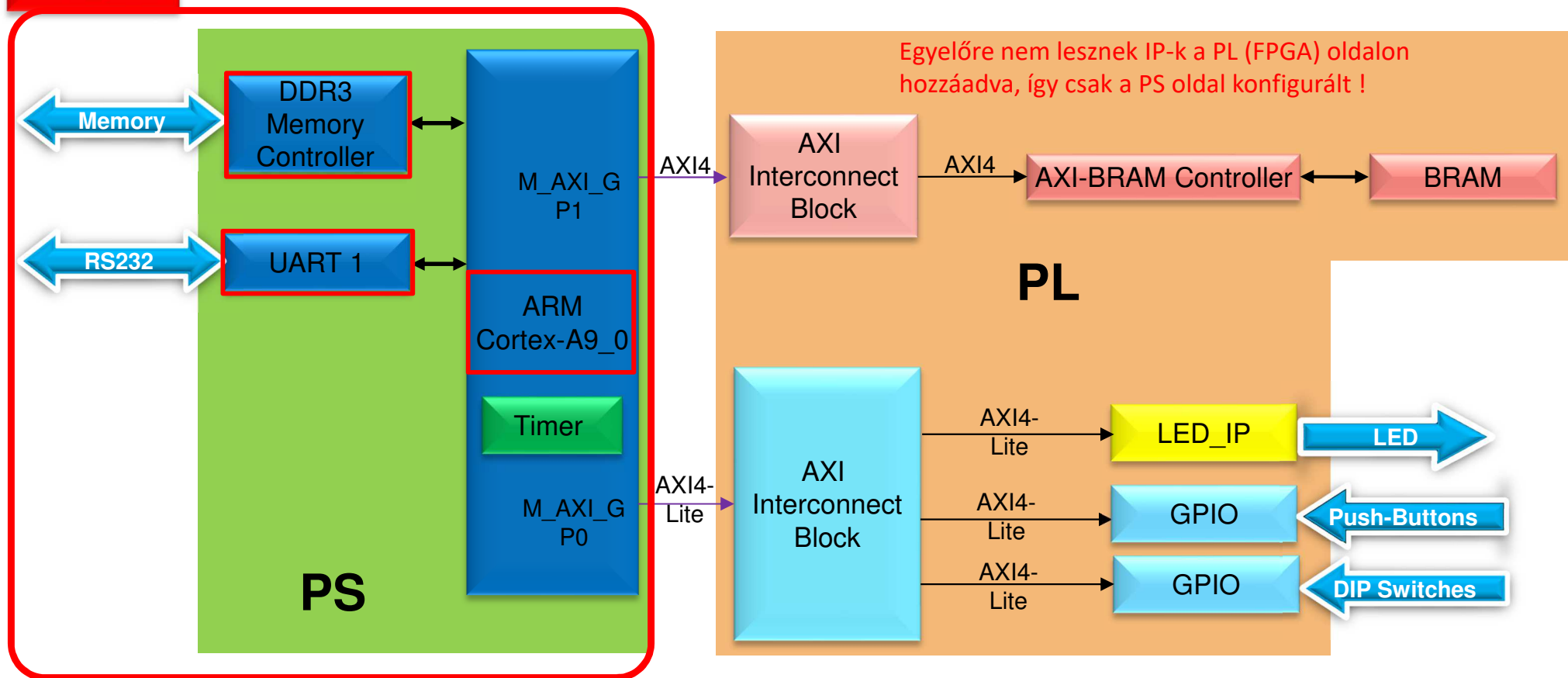
A feladat megoldásának lépései



- Új projekt létrehozása a **Xilinx Vivado** beágyazott rendszer tervezőjének segítségével,
- A létrehozott projekt áttekintése,
- *(Implementációs és Bitstream generálása, amennyiben PL oldal is konfigurálva van!)*
- *ARM-en futó Hello world! és Memória-teszt* alkalmazás készítése a **Xilinx Vivado Software Development Kit (SDK)** segítségével,
- Az elkészített beágyazott rendszer és szoftver alkalmazás teszt verifikálása **Digilent ZyBo kártyán**

A megvalósítandó tesztrendszer

LAB_01



PS oldal:

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

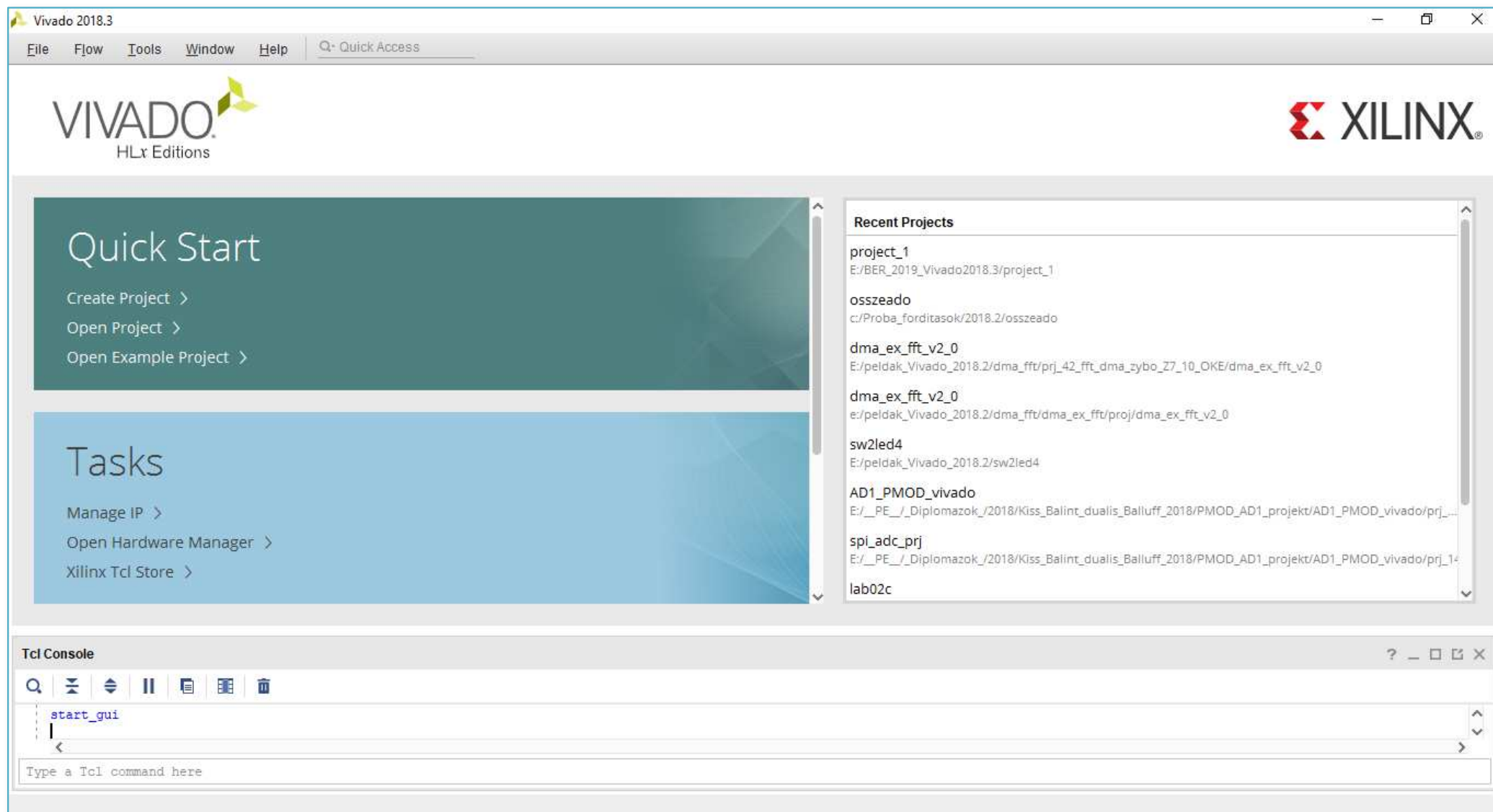
PL (FPGA) oldal most nem lesz konfigurálva: pl.

- GPIOk, LED_IP,
- AXI interconnection, stb.

Xilinx Vivado indítása

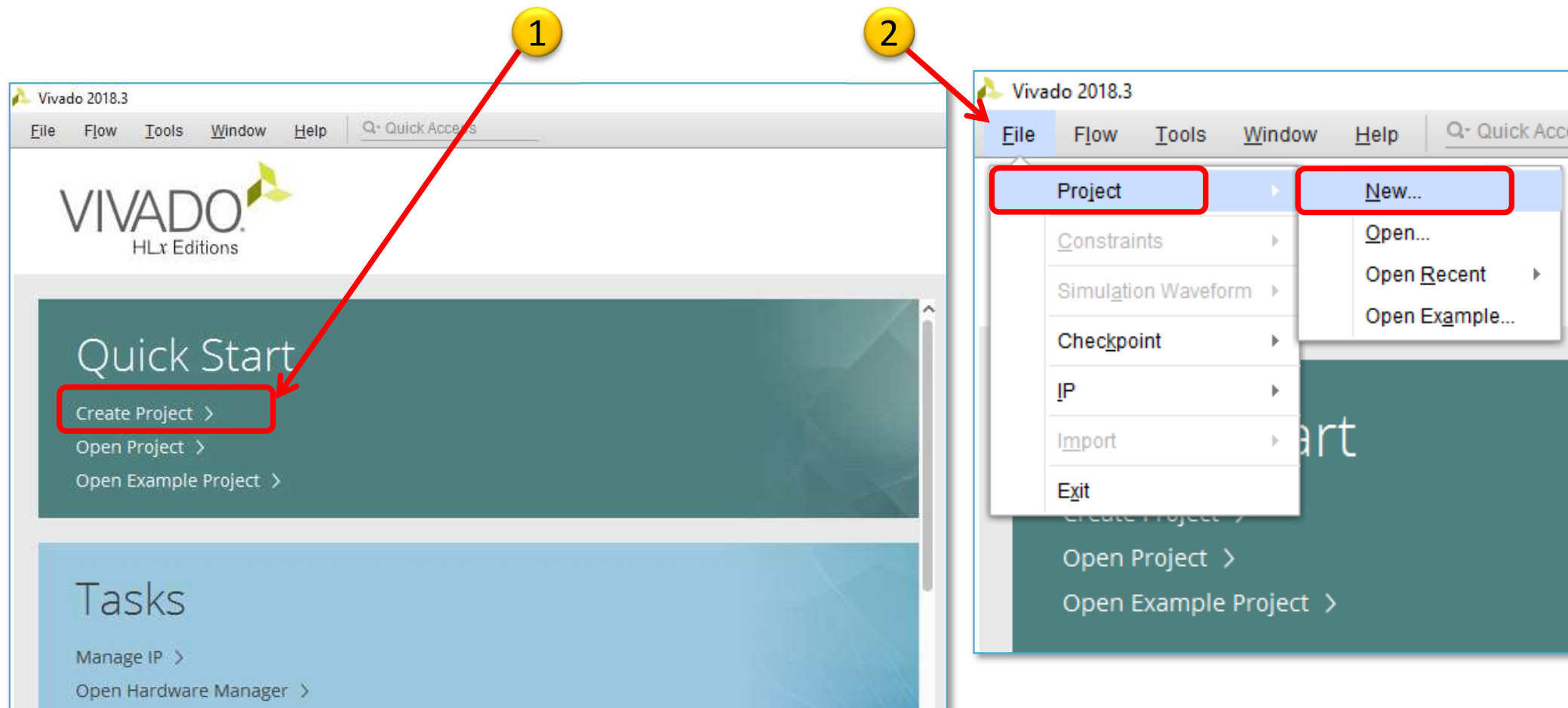


Start menü → Programok → Xilinx Design Tools → Vivado 2018.3



Új projekt létrehozása I.

Több lehetőség is van:



Új projekt készítése II.

New Project

Project Name
Enter a name for your project and specify a directory where the project data files will be stored.

Project name: 1

Project location: 2

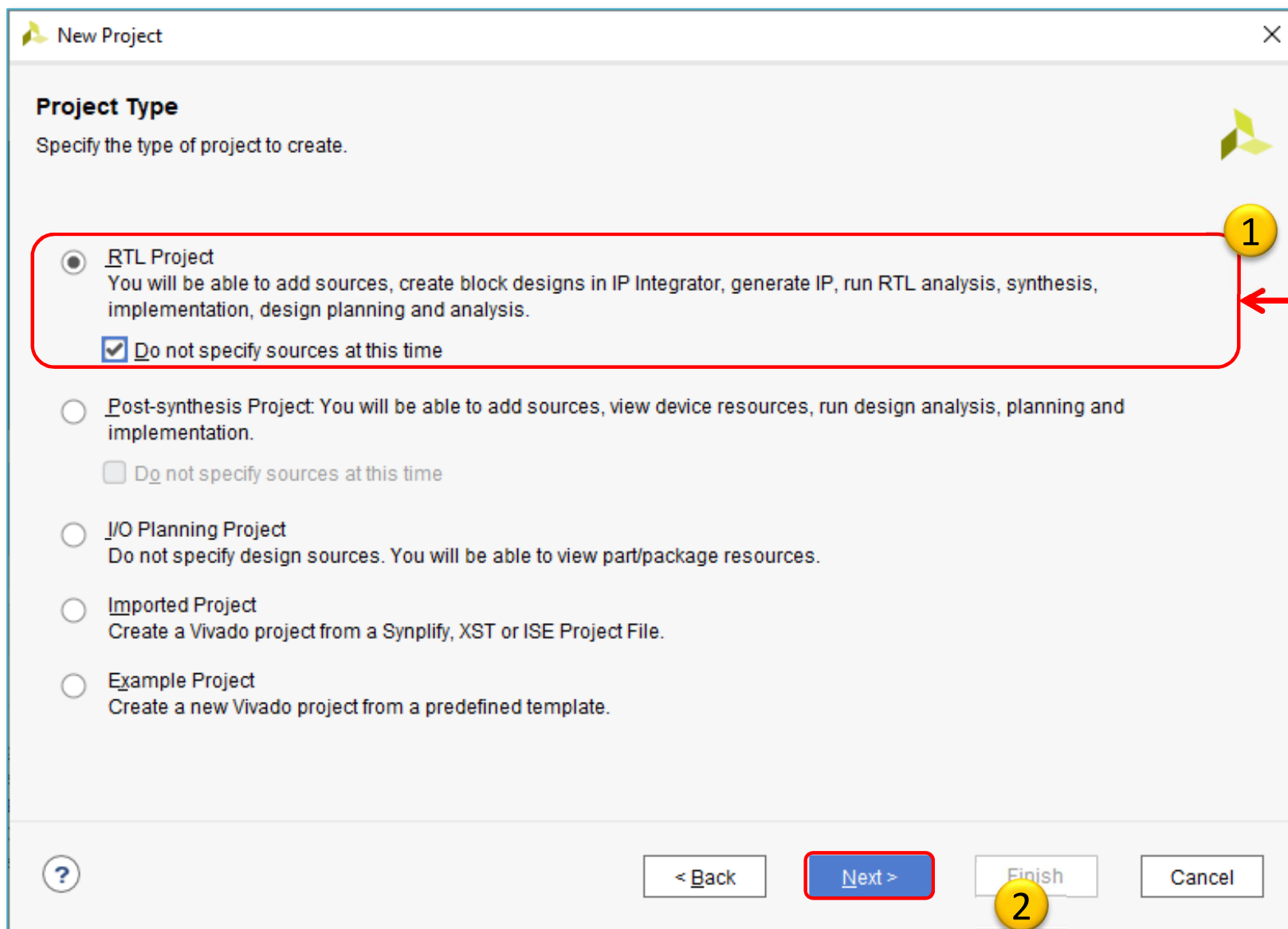
☒ Create project subdirectory

Project will be created at: E:/BER_2019_Vivado2018.3/lab01/project_1

? < Back Next > 3 Cancel

Új projekt elérési
útvonalának és
nevének beállítása
<nev.xmp>

Új projekt készítése III.



New Project

Project Type
Specify the type of project to create.

☒ **RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☒ Do not specify sources at this time

☐ **Post-synthesis Project**: You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

☐ **I/O Planning Project**
Do not specify design sources. You will be able to view part/package resources.

☐ **Imported Project**
Create a Vivado project from a Synplify, XST or ISE Project File.

☐ **Example Project**
Create a new Vivado project from a predefined template.

? < Back Next > Finish Cancel

RTL projekt kiválasztása (későbbi blokk design-hoz, és IP integrációhoz).
Ne adjunk hozzá más forrásokat (mivel nincsenek még:)

Fejlesztő platform kiválasztása

****** Csak azután választható ki a Zybo, miután telepítettük a támogató csomagját (lásd 5-6. dia)

New Project

Default Part
Choose a default Xilinx part or board for your project.

Parts | **Boards** 1

[Reset All Filters](#)

Vendor: All Name: Zybo Board Rev: Latest

Search: Q-

Display Name	Preview	Vendor	File Version	Part
Zybo		digilentinc.com	1.0	xc7z010clg400-1

****** 2

3

[?<](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

Projekt összegzés - ellenőrzés



New Project

VIVADO_{HLx Editions}

New Project Summary

1 **i** A new RTL project named 'project_1' will be created.

i The default part and product family for the new project:

1 Default Board: Zybo
Default Part: xc7z010clg400-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg400
Speed Grade: -1

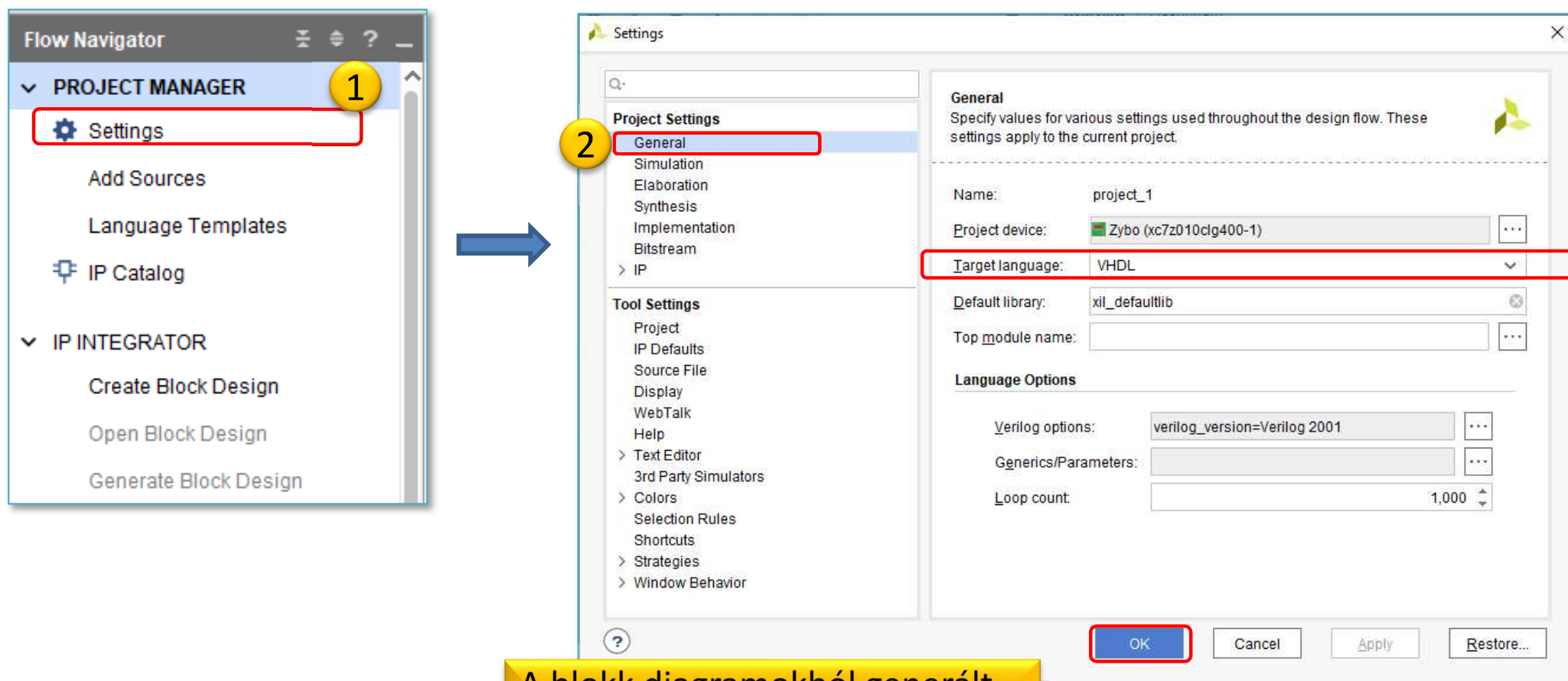
XILINX_®

To create the project, click Finish

2 **?** **< Back** **Next >** **Finish** **Cancel**

Projekt beállítások – VHDL

- Project Manager -> Settings -> General -> **VHDL** kiválasztása, majd OK.



The image shows the Project Manager and Settings dialog in Xilinx ISE. In the Project Manager, the 'Settings' option is highlighted with a red box and a yellow circle labeled '1'. A blue arrow points to the Settings dialog, where the 'General' tab is selected with a red box and a yellow circle labeled '2'. In the General tab, the 'Target language' dropdown is set to 'VHDL' and is highlighted with a red box. The 'OK' button is also highlighted with a red box.

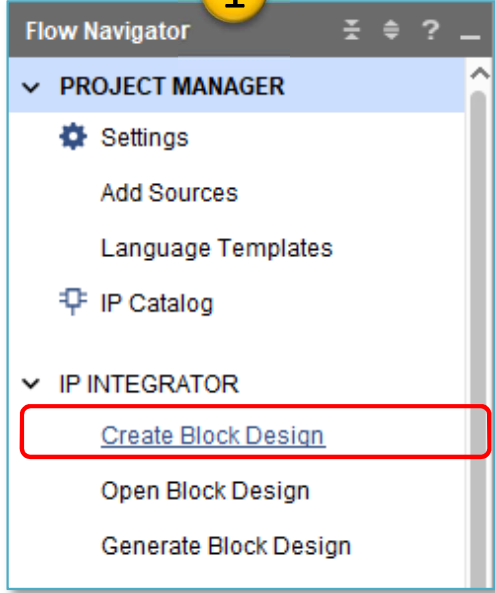
A blokk diagramokból generált top szintű „wrapper” erre a nyelvre (VHDL) generálódik le.

Beágyazott rendszer – IP integrátor



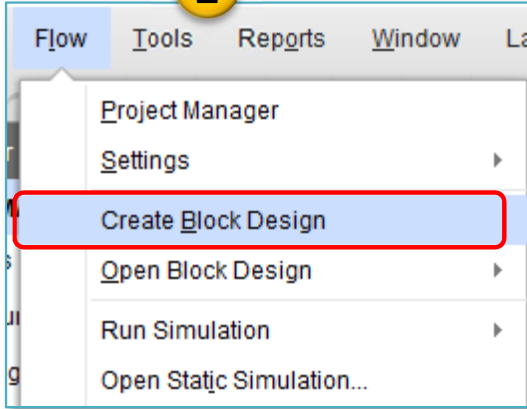
- **IP Integrátorral – új Block Design létrehozása**
 - 1. Create Block Design, vagy
 - 2. Flow menü -> Create Block Design,
 - 3. Legyen a neve „system” (3.). Végül OK.

1

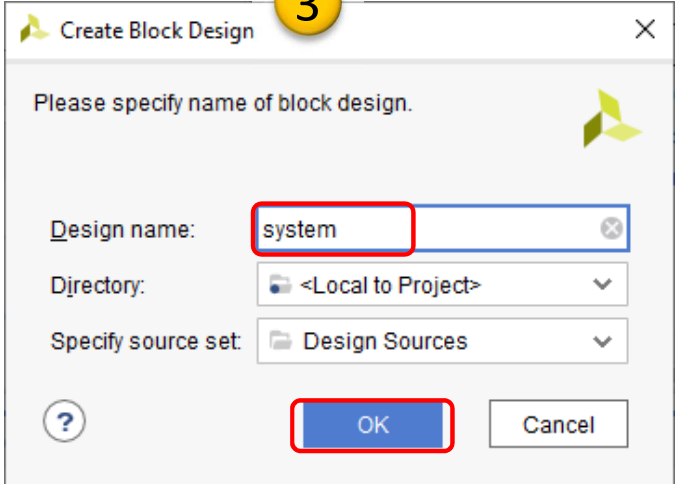


v.

2



3

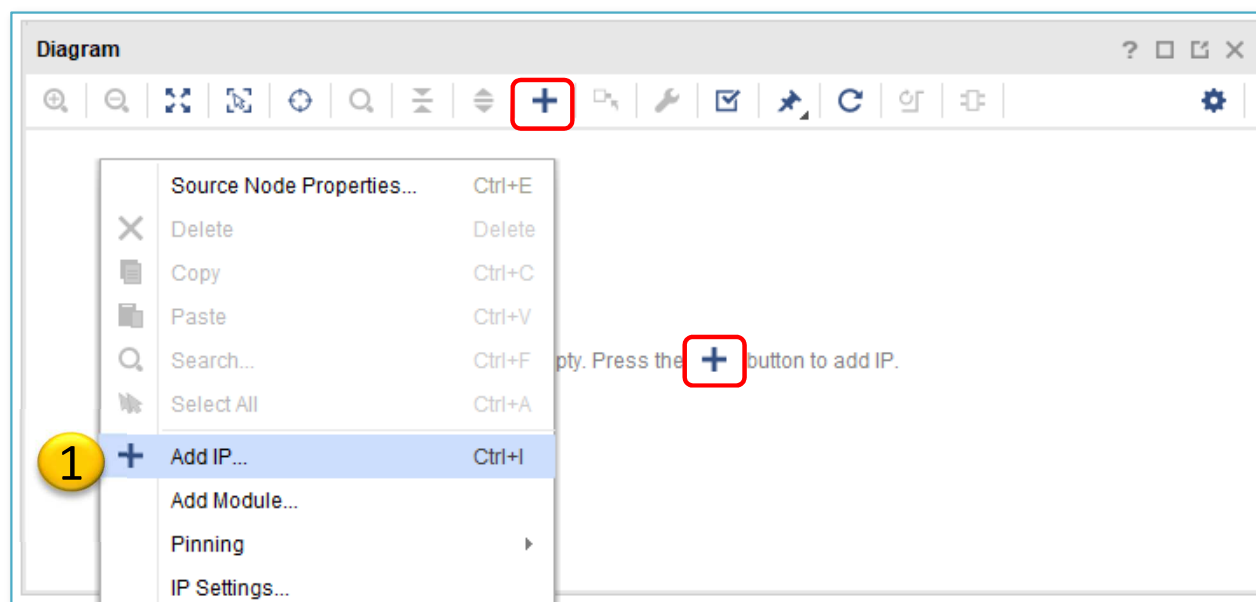
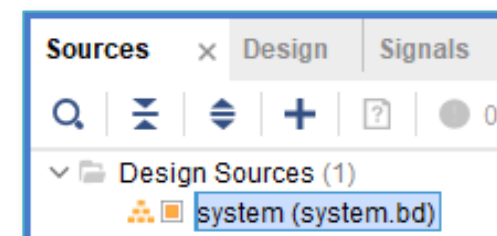


The diagram illustrates the steps to create a new Block Design in the IP Integrator. It shows three methods: 1. Using the Flow Navigator (IP INTEGRATOR > Create Block Design), 2. Using the Flow menu (Flow > Create Block Design), and 3. The resulting 'Create Block Design' dialog box where the design name is set to 'system'.

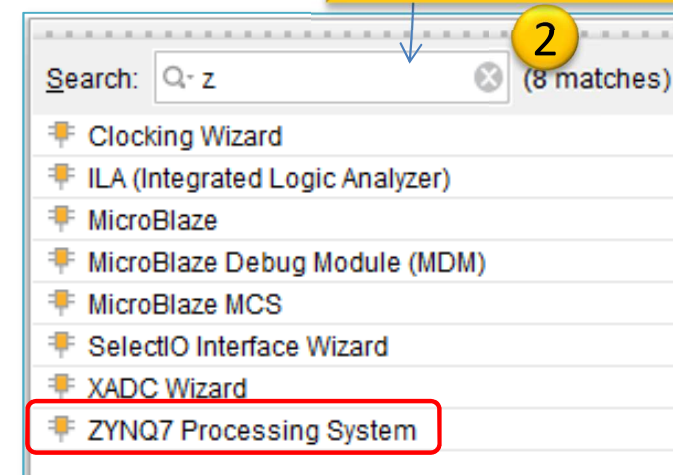
Beágyazott rendszer – Add IP

- Diagram kiválasztása (system.bd)
- Majd Add IP 

– „ZYNQ7 Processing System” kiválasztása (= PS)



Szűrés megadható.



IP katalógus: (adatbázis)

Nyissuk meg az IP katalógust.

- Project Manager -> IP Catalog

- **IP integráció támogatása**

-  Ingyenes = „included” vs.
 -  fizetős = „purchase”
(licenzszelhető
– próbaidő 90 nap)

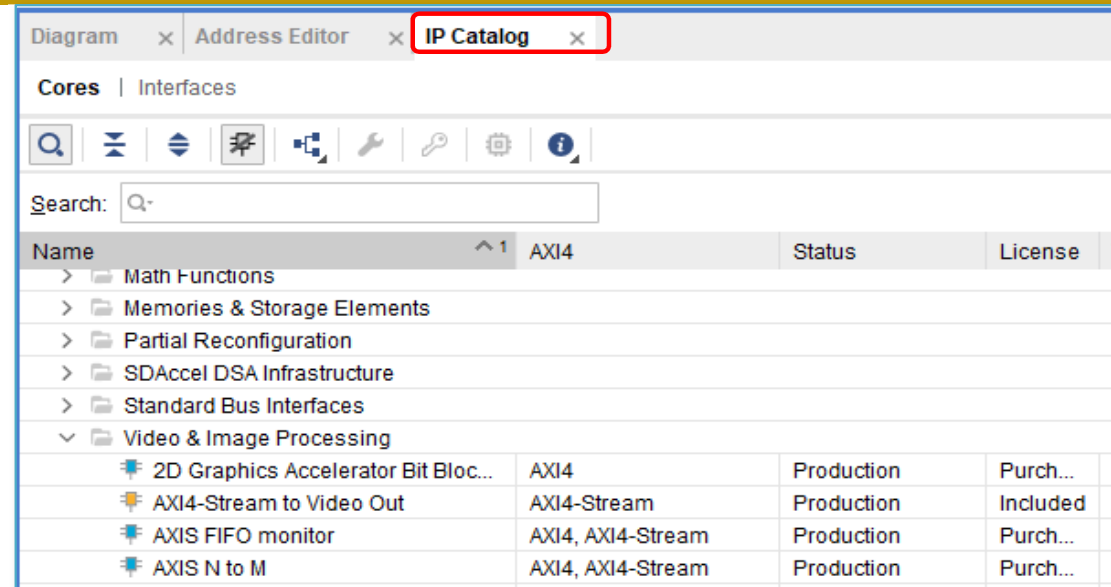
- gyors IP paraméterezés 

- Vivado IP GUI:

- kompatibilitás 
 - adatlapok elérése 
 - changelog, webpage

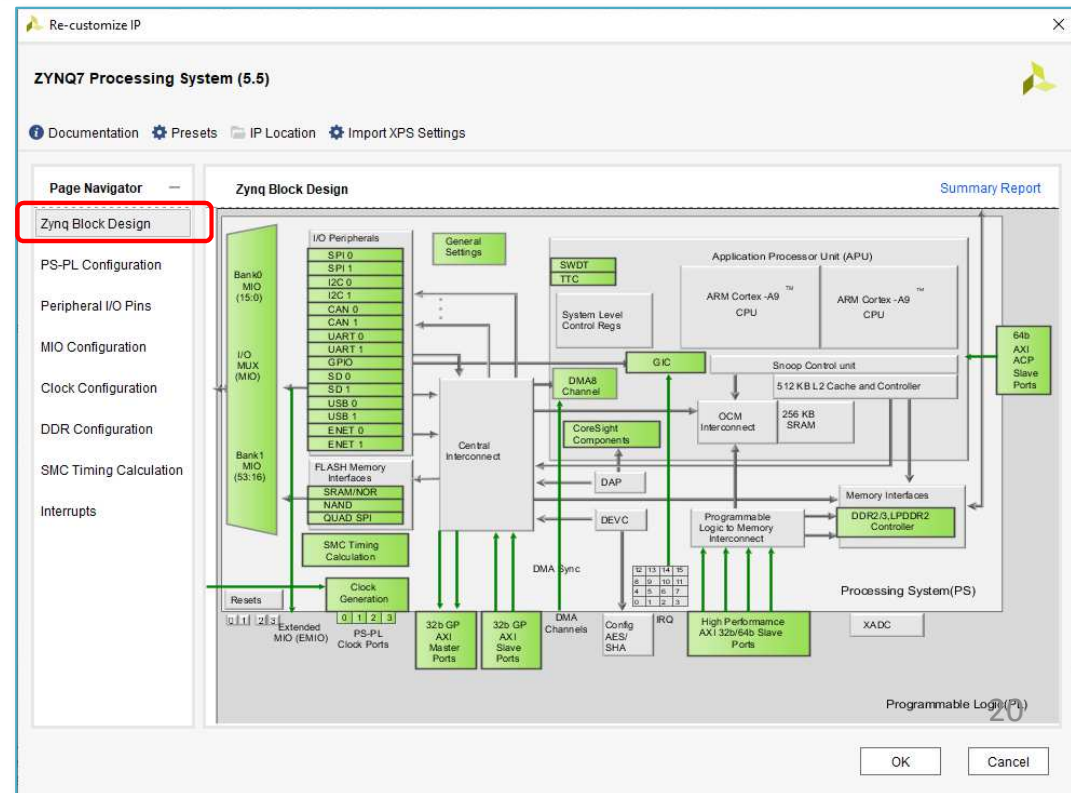
- Vivado „design flow”: Szintézis és Implementáció támogatása

- .Tcl támogatás



The screenshot shows the Vivado IP Catalog window with the 'IP Catalog' tab selected. The 'Cores' and 'Interfaces' tabs are visible. A search bar is present. The table below lists various IP cores and their details.

Name	AXI4	Status	License
Math Functions			
Memories & Storage Elements			
Partial Reconfiguration			
SDAccel DSA Infrastructure			
Standard Bus Interfaces			
Video & Image Processing			
2D Graphics Accelerator Bit Bloc...	AXI4	Production	Purch...
AXI4-Stream to Video Out	AXI4-Stream	Production	Included
AXIS FIFO monitor	AXI4, AXI4-Stream	Production	Purch...
AXIS N to M	AXI4, AXI4-Stream	Production	Purch...



Milyen IP perifériák vannak?



- Bus and bridge controllers
 - AXI to AXI connector
 - Local Memory Bus (LMB)
 - AXI Chip to Chip
 - AHB-Lite to AXI
 - AXI4-Lite to APB
 - AXI4 to AHB-Lite...
- Debug cores
 - Integrated Logic Analyzer
- DMA and Timers
 - Watchdog, fixed interval
- Inter-processor communication
 - Mailbox, Mutex
 -
- External peripheral controller
 - Memory and memory controller
- High-speed and low-speed communication peripherals
 - AXI 10/100 Ethernet MAC controller
 - Hard-core tri-mode Ethernet MAC
 - AXI IIC
 - AXI SPI
 - AXI UART...
- Other cores
 - System monitor
 - Xilinx Analog-to-Digital Converter (XADC)
 - Clock generator, System reset module
 - interrupt controller
 - Traffic Generator, Performance monitor
 -

IP repository (könyvtár szerkezet)

- IP Core könyvtára (két lehetőség: a projekt könyvtárában vagy a központi könyvtárban = Repository - CVS/GIT)

- { component } .xml alapú leíró

- /MyProcessorIPLib directory (user defined)

- Repo alkönyvtárak listázása:

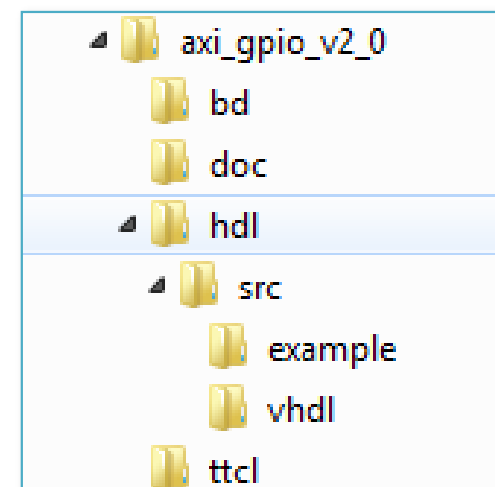
Project Manager →

Settings →

IP Defaults / Repository tab

- %XILINX_INSTALL%\Vivado\2018.X\data\ip

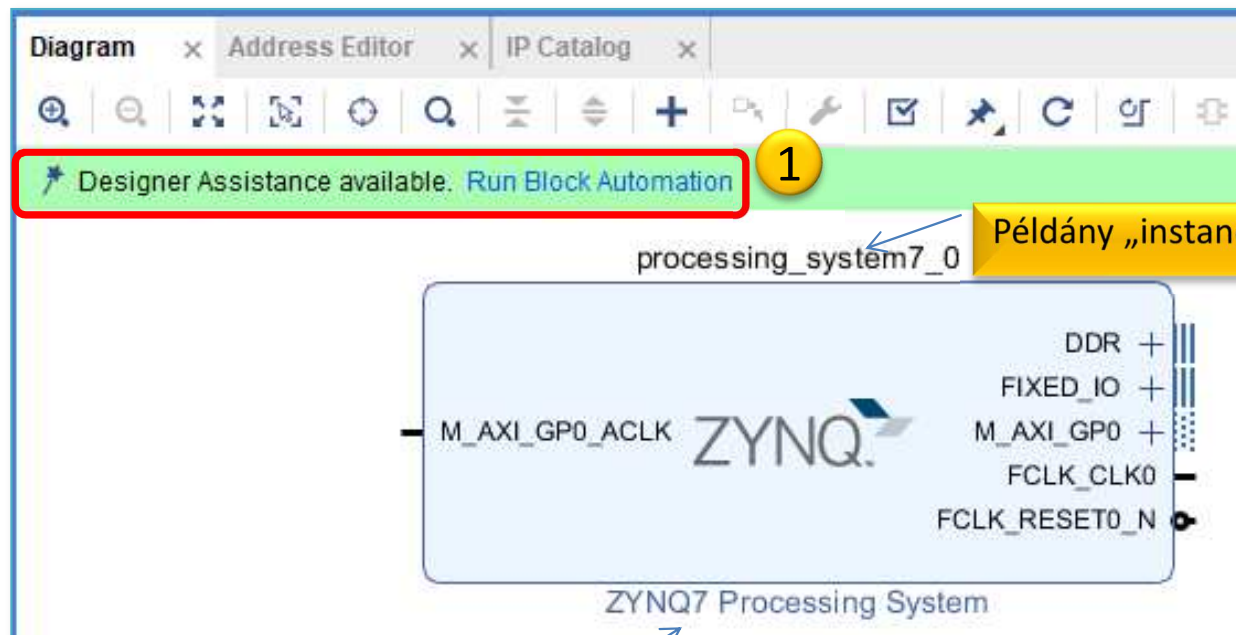
PI:



PI:

```
<?xml:version="1.0".encoding="UTF-8"?><spirit:component.xmlns:xilinx="http://www.xilinx.com".xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009".xml:
..<spirit:vendor>xilinx.com</spirit:vendor>
..<spirit:library>ip</spirit:library>
..<spirit:name>axi_gpio</spirit:name>
..<spirit:version>2.0</spirit:version>
..<spirit:busInterfaces>
...<spirit:busInterface>
.....<spirit:name>S_AXI</spirit:name>
.....<spirit:displayName>S_AXI</spirit:displayName>
.....<spirit:busType>spirit:vendor="xilinx.com".spirit:library="interface".spirit:name="aximm".spirit:version="1.0"/>
.....<spirit:abstractionType>spirit:vendor="xilinx.com".spirit:library="interface".spirit:name="aximm_rtl".spirit:version="1.0"/>
.....<spirit:slave/>
.....<spirit:portMaps>
.....<spirit:portMap>
.....<spirit:logicalPort>
.....<spirit:name>ARADDR</spirit:name>
.....</spirit:logicalPort>
```

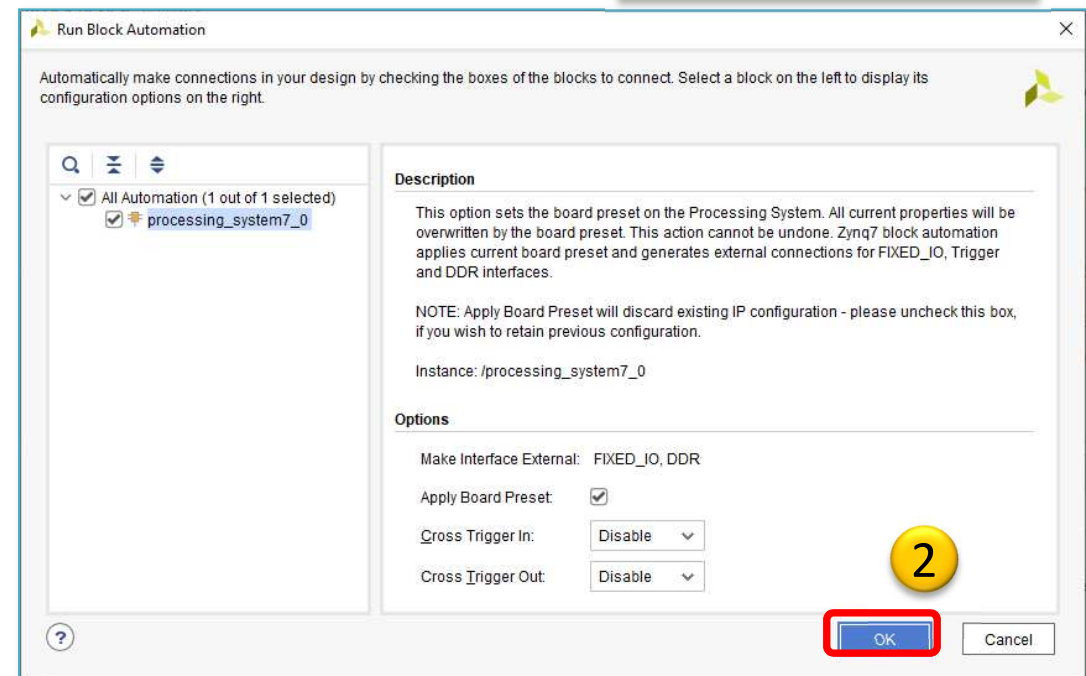
Diagram – Run Block Automation



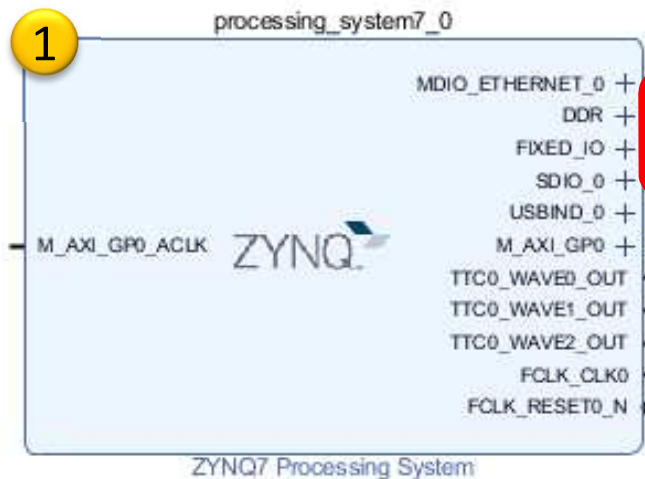
Példány „instance” név (tetszőleges)

IP katalógus név (fix)

Minden maradjon az alap beállításon. OK.

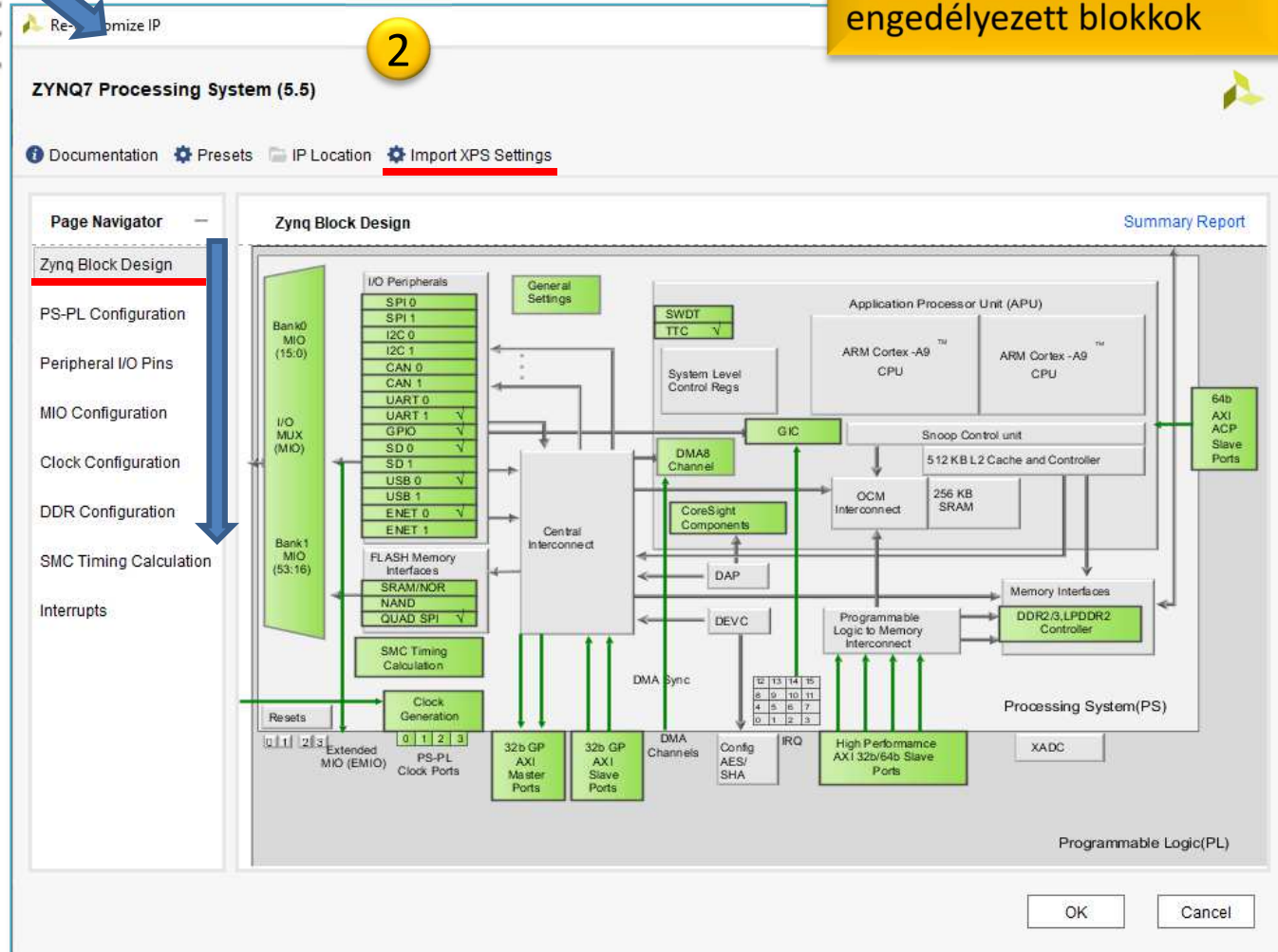


Zynq Blokk Design (DDR, IO portok)



Láthatóvá válnak a
külső DDR és IO
portok

Zöld: konfigurálható,
paraméterezhető blokkok
✓: használatra
engedélyezett blokkok



- *Import XPS settings:*
Lehetőség van a
beállítások importálására
(gyári default preset .xml
file-től eltérően)!

Zynq – PS-PL configuration



- *General [+]* ->
 - *UART1 Baud rate: 115.200 ?*
 - *Enable Clock Resets* : törölni a **FCLK_RESET0_N** –t.
- *AXI Non Secure Enablement [+]* ->
 - *GP Master AXI interface*: törölni a **M AXI GP0** bridge-t .

1

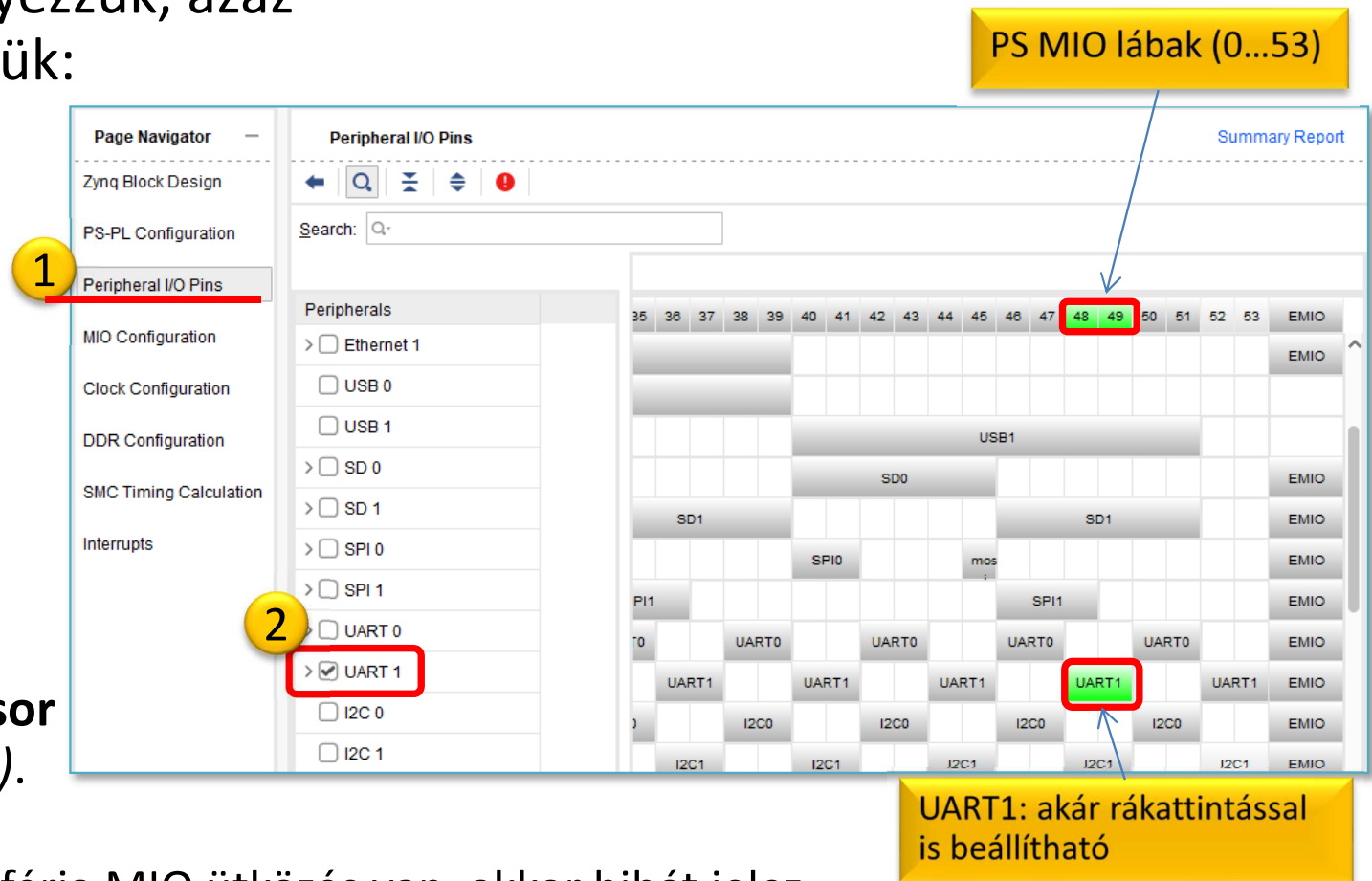
The screenshot shows the 'PS-PL Configuration' window. On the left, the 'Page Navigator' lists various configuration pages, with 'PS-PL Configuration' highlighted and marked with a yellow circle containing the number '1'. The main area displays a table of configuration options under the 'AXI Non Secure Enablement' section.

Name	Select	Description
> General		
AXI Non Secure Enablement	0	Enable AXI Non Secure Transaction
> GP Master AXI Interface		
> M AXI GP0 interface	☐	Enables General purpose AXI master interface 0
> M AXI GP1 interface	☐	Enables General purpose AXI master interface 1
> GP Slave AXI Interface		
> HP Slave AXI Interface		
> ACP Slave AXI Interface		
> DMA Controller		
> PS-PL Cross Trigger interface	☐	Enables PL cross trigger signals to PS and vice-versa

Zynq PS – Peripheral IO pins

- Állítsuk be (✓) az **UART1**-et a soros log-olás miatt.
- A többi alapértelmezett blokkot egyelőre NE engedélyezzük, azaz vegyük ki a pipát előlük:

- *ENET 0*
- *USB 0*
- *SD 0*
- **GPIO - GPIO MIO**
- **Memory Interfaces**
– *Quad SPI Flash*
- **Application Processor Unit - Timer 0 (TTC0).**



PS MIO lábak (0...53)

Summary Report

Peripheral I/O Pins

Search: Q

Peripherals

- > ☐ Ethernet 1
- ☐ USB 0
- ☐ USB 1
- > ☐ SD 0
- > ☐ SD 1
- > ☐ SPI 0
- > ☐ SPI 1
- > ☐ UART 0
- > ☒ **UART 1**
- ☐ I2C 0
- ☐ I2C 1

Pin assignment grid (Pins 35-53):

Pin	Function
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	UART1
49	
50	
51	
52	
53	

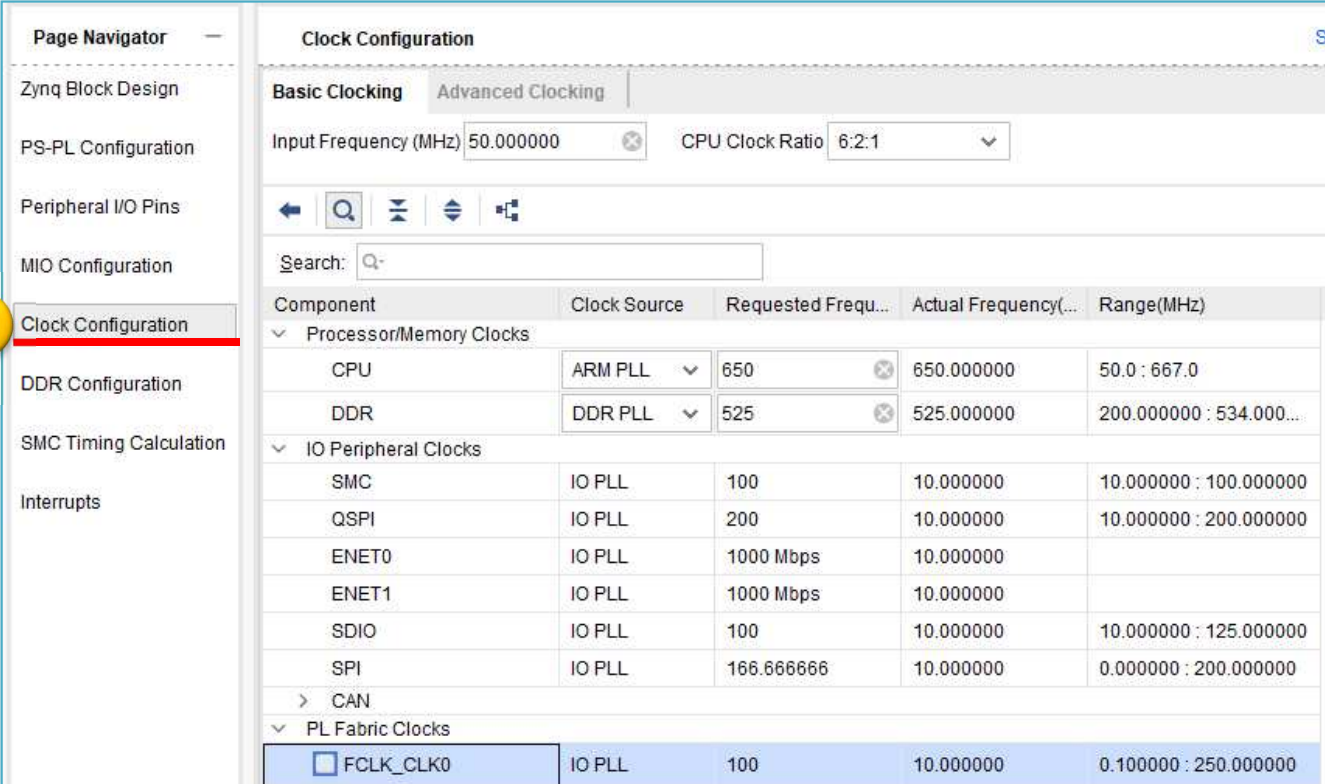
UART1: akár rákattintással is beállítható

! Conflict View: ha periféria MIO ütközés van, akkor hibát jelez.

Zynq – Clock configuration

- PL Fabric Clocks (FPGA) ->
 - FCLK_CLK0 törlése

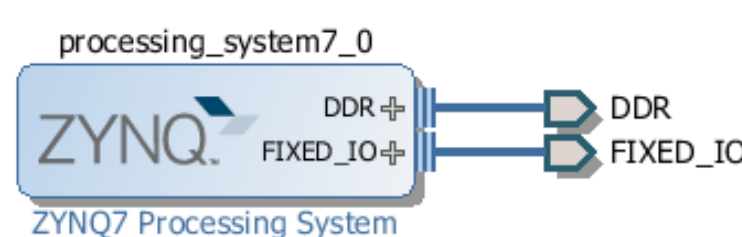
1



Component	Clock Source	Requested Frequ...	Actual Frequency(...)	Range(MHz)
Processor/Memory Clocks				
CPU	ARM PLL	650	650.000000	50.0 : 667.0
DDR	DDR PLL	525	525.000000	200.000000 : 534.000...
IO Peripheral Clocks				
SMC	IO PLL	100	10.000000	10.000000 : 100.000000
QSPI	IO PLL	200	10.000000	10.000000 : 200.000000
ENET0	IO PLL	1000 Mbps	10.000000	
ENET1	IO PLL	1000 Mbps	10.000000	
SDIO	IO PLL	100	10.000000	10.000000 : 125.000000
SPI	IO PLL	166.666666	10.000000	0.000000 : 200.000000
> CAN				
PL Fabric Clocks				
☐ FCLK_CLK0	IO PLL	100	10.000000	0.100000 : 250.000000

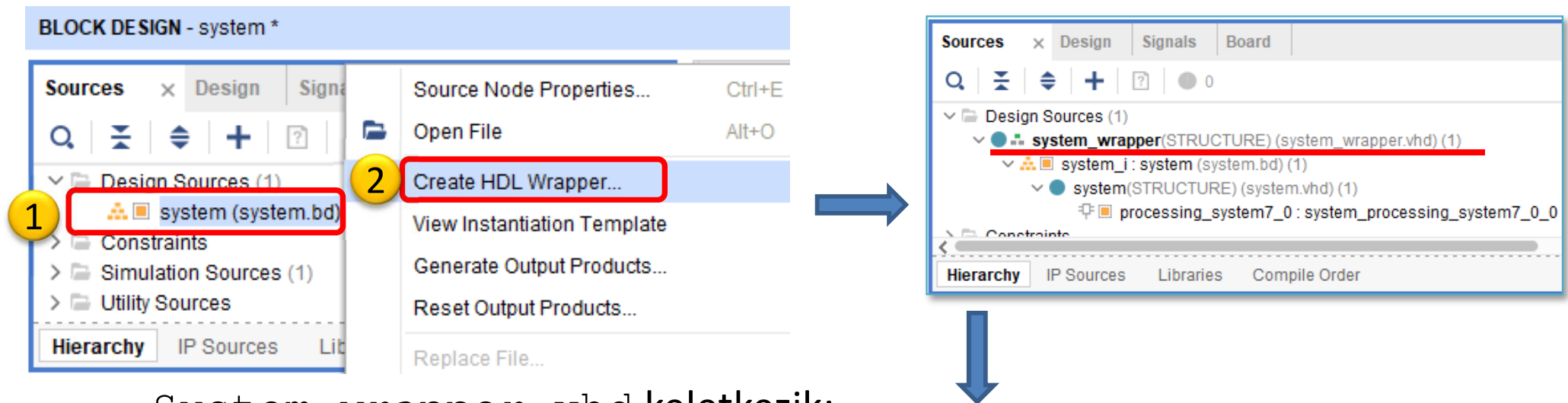
- Végül: klikk OK.

- Regenerate Layout
- Validate Design (DRC)



Generate top-level HDL

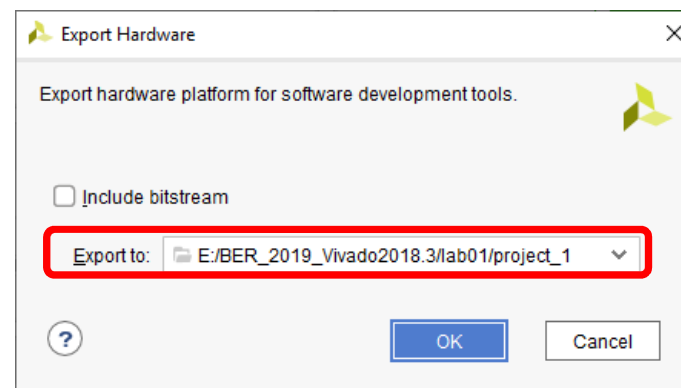
- Design Sources -> Create HDL wrapper ->
 - *Let Vivado manager wrapper and auto-update*
- Megj: top-level  wrapper nélkül nem futna majd az implementáció



```
Address Editor x IP Catalog x system_wrapper.vhd x
2018.3/lab01/project_1/project_1.srcs/sources_1/bd/system/hdl/system_wrapper.vhd x
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 library UNISIM;
13 use UNISIM.VCOMPONENTS.ALL;
14 entity system_wrapper is
15 port (
16     DDR_addr : inout STD_LOGIC_VECTOR ( 14 downto 0 );
```

Export HW -> Vivado SDK

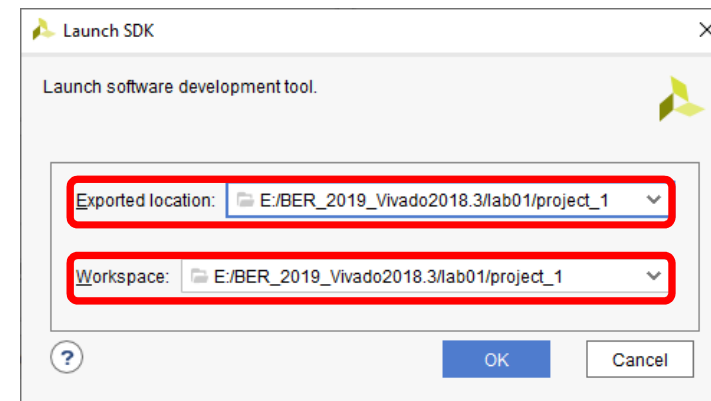
- RTL Analysis -> Open Elaborated Design
- File -> Export -> Export hardware



2018.3: legalább elaborated design kell, hogy lehessen HW exportálni!

Mivel PL (FPGA) oldal nincs konfigurálva (mindent töröltünk), ezért most nem kell bitstream.

- File -> Launch SDK, Útvonalak beállítása, OK.
 - Vivado SDK indítása a keretrendszerből (ezt kívülről indítva is meg lehetne tenni)



Xilinx Vivado SDK használata

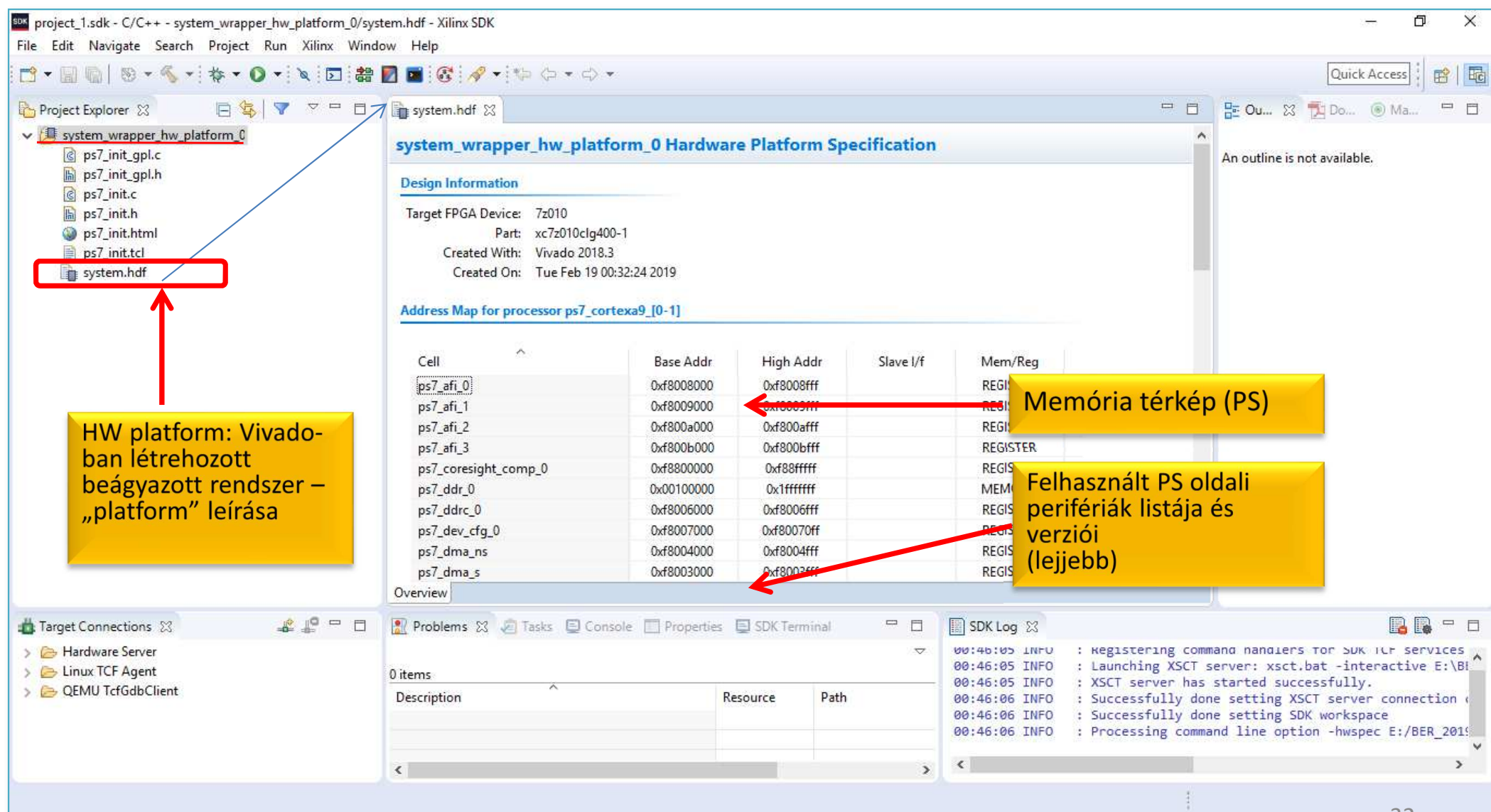
SZOFTVER TESZT ALKALMAZÁS ÖSSZEÁLLÍTÁSA

- 
1. Vivado projekt létrehozása, majd Export → **SDK**,
 2. Új üres, vagy C/C++ template-ből generálható alkalmazás létrehozása (legyen egy Hello World, ill. egy Memória-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, ha van PL oldal)*
 6. *„debug” – hardveres hibakeresés beállítása*
 7. Debug (breakpoint-ok beszúrása, léptetés, stb.)

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.



HW platform: Vivado-ban létrehozott beágyazott rendszer – „platform” leírása

Memória térkép (PS)

Felhasznált PS oldali perifériák listája és verziói (lejjebb)

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
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	0xf800c000	0xf800cfff		REGISTER
ps7_ddr_0	0x00100000	0x1fffffff		MEMORY
ps7_ddrc_0	0xf8006000	0xf8006fff		REGISTER
ps7_dev_cfg_0	0xf8007000	0xf80070ff		REGISTER
ps7_dma_ns	0xf8004000	0xf8004fff		REGISTER
ps7_dma_s	0xf8003000	0xf8003fff		REGISTER

0 items

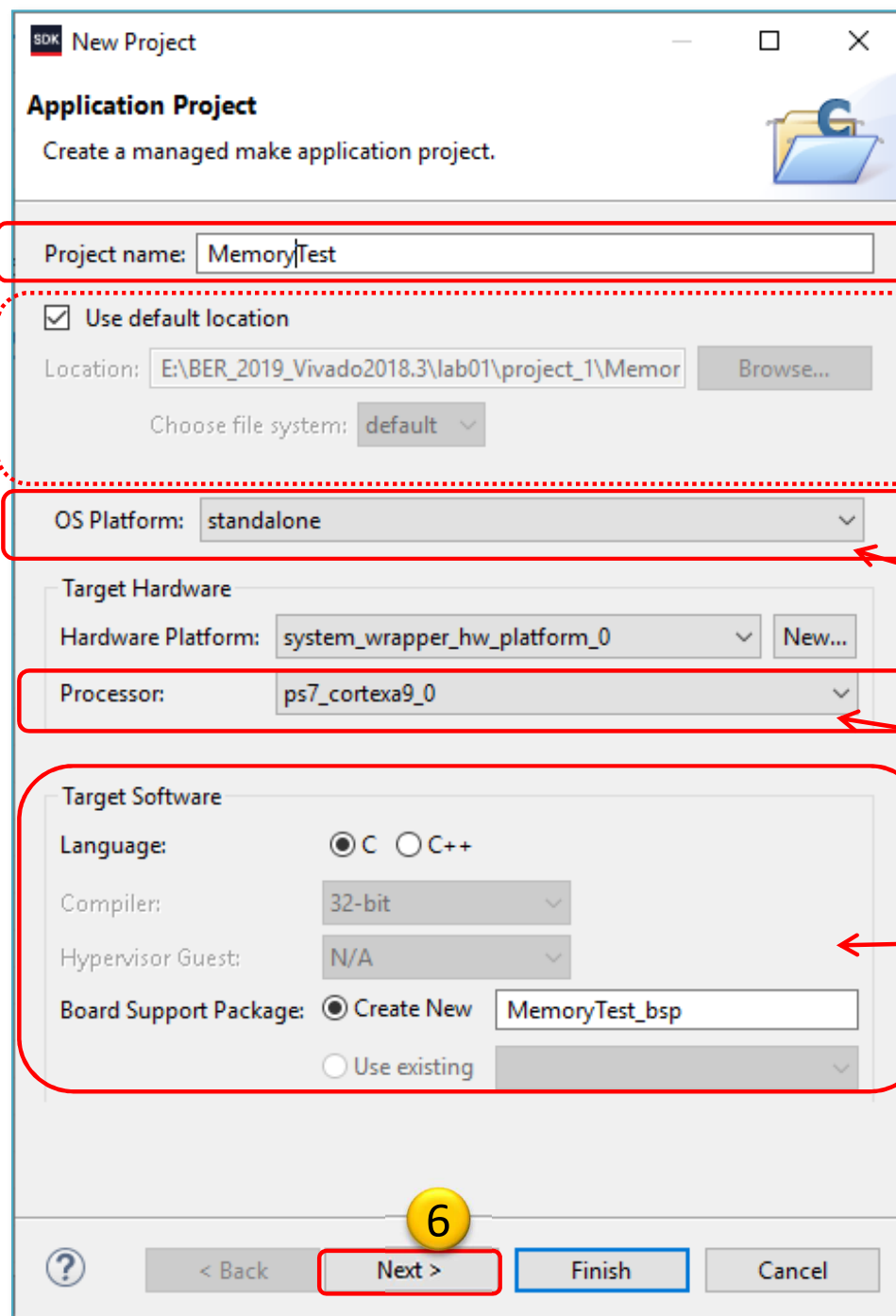
Description	Resource	Path
-------------	----------	------

SDK Log

```

00:46:05 INFO : registering command handlers for SDK ICF services
00:46:05 INFO : Launching XSCT server: xsct.bat -interactive E:\B...
00:46:05 INFO : XSCT server has started successfully.
00:46:06 INFO : Successfully done setting XSCT server connection
00:46:06 INFO : Successfully done setting SDK workspace
00:46:06 INFO : Processing command line option -hwspec E:\BER_2019
  
```

Teszt alkalmazás készítése I.



SDK menü: File → New → Application Project

1 Adjuk meg a projekt nevét

2 Adjuk meg a projekt elérési útvonalát (alapértelmezett az Vivado-ból generált könyvtár)

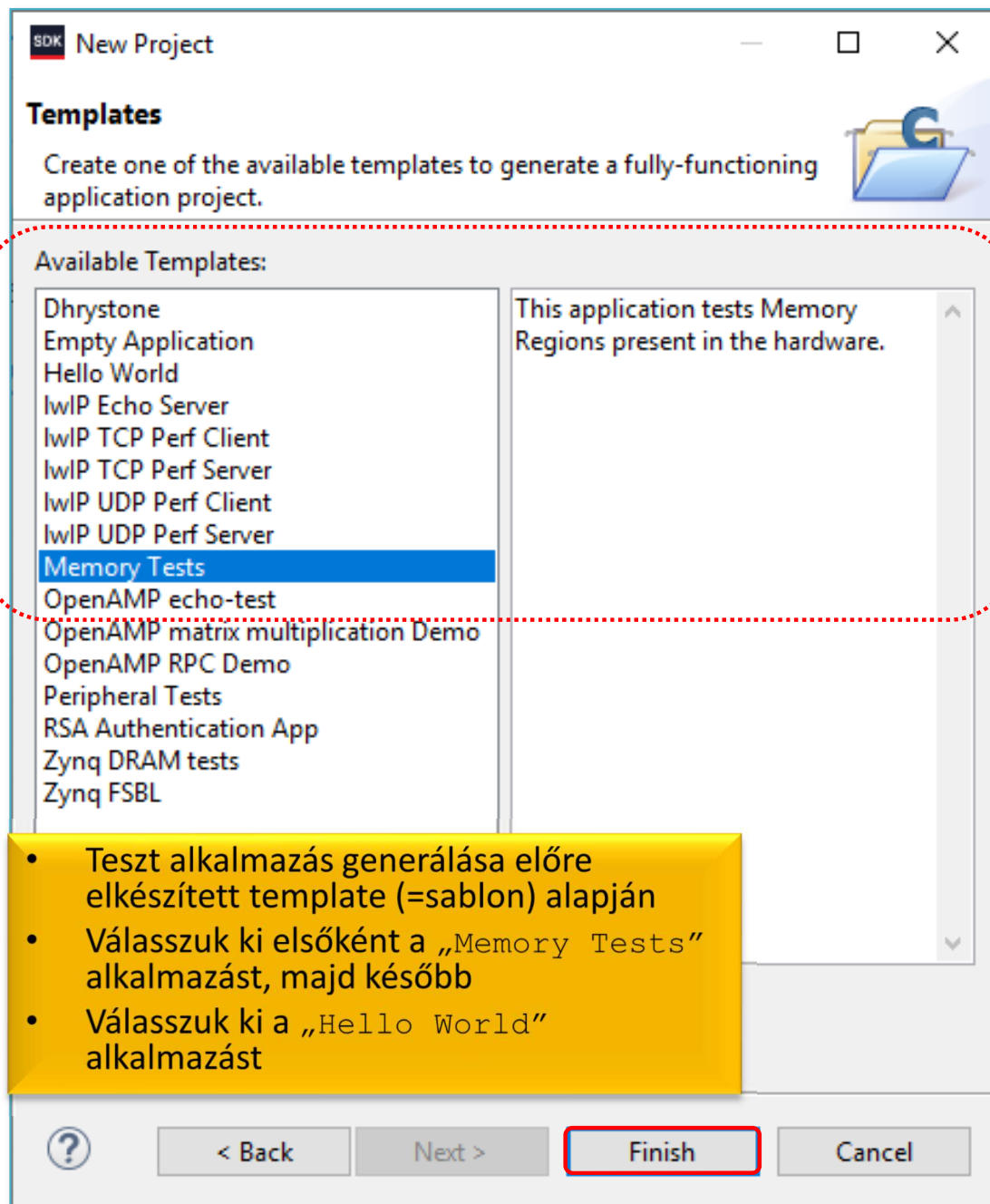
3 OS, non-OS környezet beállítása: **Standalone** vs. (xilkernel) vs. Linux

4 Válasszuk ki: ps7_cortexa9_0 (= ARM Core 0)

5 Új, vagy meglévő BSP létrehozása

NEXT >>

Teszt alkalmazás készítése II.



Az alapértelmezett Board Support Package (BSP) létrehozása

- BSP tartalmazza a legalacsonyabb szintű könyvtári függvényeket, drivereket, eszközkészítőket, mely az alkalmazásunk szoftver stack-jének legalacsonyabb rétege és az összeállított BSB-re épül
- Az szoftver alkalmazásunknak az adott BSP felett kell futnia, API segítségével:
- HW → BSP → SW egymásra épülése
- BSP: új ①, vagy korábban már létrehozott ② is használható (alapértelmezett hely \SDK)

BSP – system.mss (grafikus nézet)



.MSS: Microprocessor Software Specification (system.mss)

The screenshot shows an IDE interface with two main panes. On the left, the 'Project Explorer' pane (labeled 1) displays a project tree. The 'system.mss' file (labeled 2) is selected under the 'system_wrapper_hw_platform_0' folder. On the right, the 'system.mss' file (labeled 3) is open, showing the 'MemoryTest_bsp Board Support Package' configuration. The configuration includes buttons for 'Modify this BSP's Settings' and 'Re-generate BSP Sources'. The 'Target Information' section states: 'This Board Support Package is compiled to run on the following t', 'Hardware Specification: E:\BER_2019_Vivado2018.3\lab01\project', and 'Target Processor: ps7_cortexa9_0'. The 'Operating System' section shows 'Board Support Package OS.' with details: 'Name: standalone', 'Version: 6.8', and 'Description: Standalone is a simple, low-level software layer environment, such as standard input and output'. The 'Documentation' link is 'standalone v6 8'. The 'Peripheral Drivers' section lists 'ps7_afi_0 generic'.

1 Project Explorer

2 system.mss

3 system.mss

System.mss (jobb gomb) ->
Open with MSS Viewer

MemoryTest_bsp Board Support Package

Modify this BSP's Settings Re-generate BSP Sources

Target Information

This Board Support Package is compiled to run on the following t

Hardware Specification: E:\BER_2019_Vivado2018.3\lab01\project

Target Processor: ps7_cortexa9_0

Operating System

Board Support Package OS.

Name: standalone

Version: 6.8

Description: Standalone is a simple, low-level software layer environment, such as standard input and output

Documentation: [standalone v6 8](#)

Peripheral Drivers

Drivers present in the Board Support Package.

ps7_afi_0 generic

BSP – system.mss (szöveges nézet)

.MSS: Microprocessor Software Specification (system.mss)

1

2

3

Project Explorer

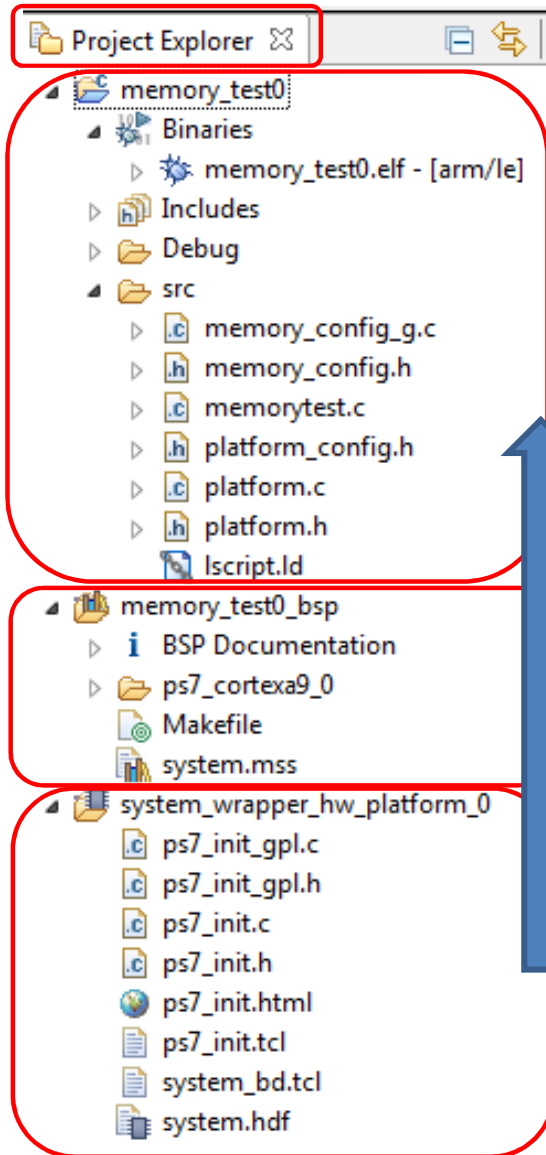
- MemoryTest
- MemoryTest_bsp
 - BSP Documentation
 - ps7_cortexa9_0
 - Makefile
 - system.mss
 - system_wrapper_hw_platform_0
 - ps7_init_gpl.c
 - ps7_init_gpl.h
 - ps7_init.c
 - ps7_init.h
 - ps7_init.html
 - ps7_init.tcl
 - system.hdf

system.hdf system.mss

```
BEGIN OS # alap beágyazott OS -  
    „standalone”  
    PARAMETER OS_NAME = standalone  
    PARAMETER OS_VER = 6.8  
    PARAMETER PROC_INSTANCE = ps7_cortexa9_0  
    #példány név  
    PARAMETER STDIN = ps7_uart_1  
    PARAMETER STDOUT = ps7_uart_1 #soros  
    portra log-ol  
END  
.....  
... # az Uart1 IP modulhoz tartozó Xilinx  
    driver  
BEGIN DRIVER  
    PARAMETER DRIVER_NAME = uartps #driver  
    neve fix  
    PARAMETER DRIVER_VER = 3.7  
    PARAMETER HW_INSTANCE = ps7_uart_1  
END  
...
```

System.mss (jobb gomb) ->
Open with Text Editor

Teszt alkalmazás készítése III.



- **SW:** memory_test0 (SW alkalmazás)
 - \Binaries (**futtatható .elf** object fájl)
 - \Includes (gyári header-ek)
 - \Debug
 - \Src = **.h, .c, .cpp** források gyűjteménye
 - **.ld = linker script!**
 - **Main() a memorytest.c-ben**

- **BSP:** memory_test0_bsp (SW BSP)
 - **MSS: Microprocessor software/driver** leíró fájl (**system.mss**)
 - **/includes/xparameters.h** (minden kapcsolódó **#define** és címtartomány itt definiálódik)

HW platform (Vivado-ból exportált)
(.bit, .hdf)

- **xparameters.h** használata (define-ok, címek)
- `init_platform();`
 - `enable_caches();`
- **test_memory_range** (**struct** `memory_range_s` *range)
- `test_memory_range(&memory_ranges[i])`
 - `n_memory_ranges = 2` (`memory_config.h`-ban definiálva)
- `cleanup_platform();`
 - `disable_caches();` // \$ ARM alapból engedélyezné

Xil_TestMemN() függvény



```
status = Xil_TestMem32((u32*)range->base, 1024, 0xAAAA5555,  
XIL_TESTMEM_ALLMEMTESTS);  
    print("          32-bit test: "); print(status ==  
XST_SUCCESS? "PASSED!":"FAILED!"); print("\n\r");
```

```
status = Xil_TestMem16((u16*)range->base, 2048, 0xAA55,  
XIL_TESTMEM_ALLMEMTESTS);  
    print("          16-bit test: "); print(status ==  
XST_SUCCESS? "PASSED!":"FAILED!"); print("\n\r");
```

```
status = Xil_TestMem8((u8*)range->base, 4096,  
0xA5, XIL_TESTMEM_ALLMEMTESTS);  
    print("          8-bit test: "); print(status ==  
XST_SUCCESS? "PASSED!":"FAILED!"); print("\n\r");
```

Xil_TestMemN() függvény deklarációja a következő helyen van:

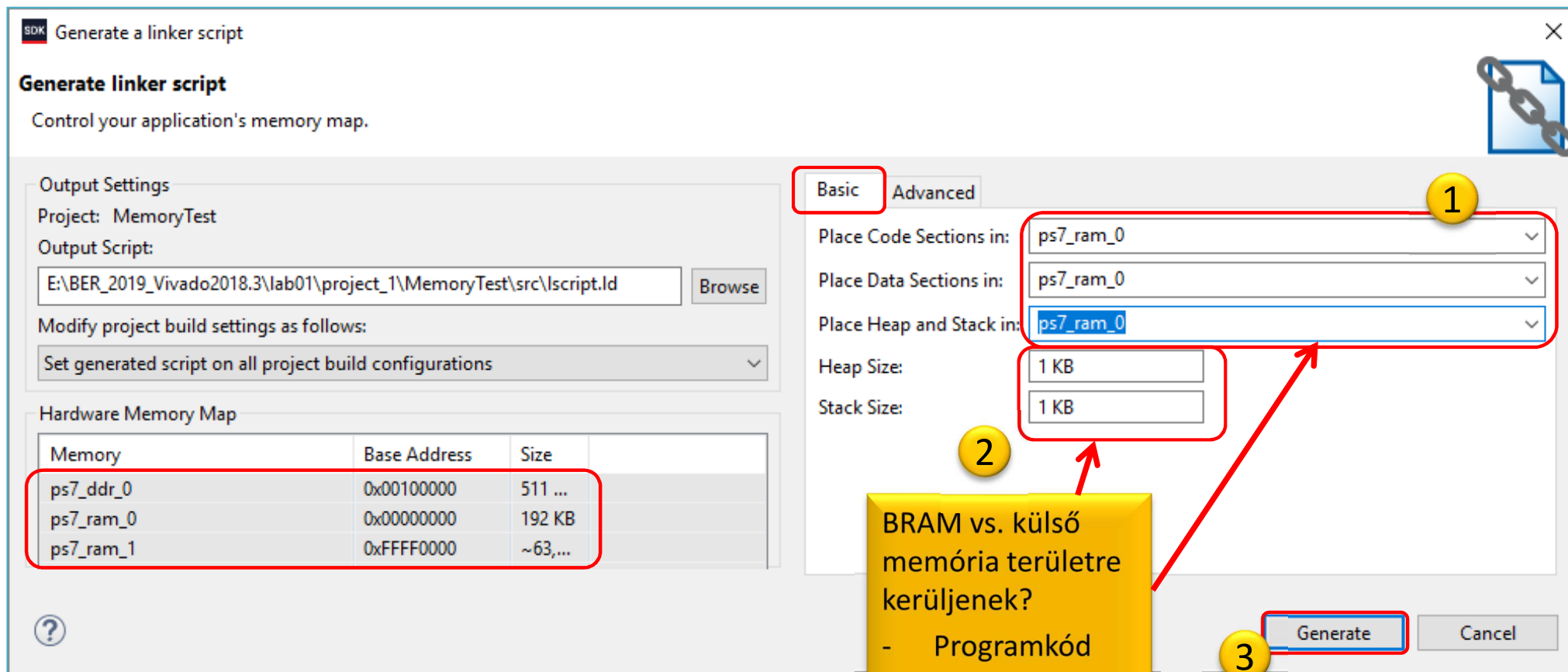
```
<xilinx_install_dir>\SDK\2015.2\data\embeddedsd\lib\bsp\standalone_v5  
_1\src\common\xil_testmem.c
```

s32 Xil_TestMem32(u32 *Addr, u32 Words, u32 Pattern, u8 Subtest);

Teszt alkalmazás készítése IV./a.

Linker script generálása (`lscript.ld`)

- Xilinx Tools menü → Generate Linker Script (**Basic** nézet)
- Állítsuk át a memória részeket (ddr_0-ról -> ram_0-ra)



Generate a linker script

Generate linker script

Control your application's memory map.

Output Settings

Project: MemoryTest

Output Script:

E:\BER_2019_Vivado2018.3\lab01\project_1\MemoryTest\src\lscript.ld

Browse

Modify project build settings as follows:

Set generated script on all project build configurations

Hardware Memory Map

Memory	Base Address	Size
ps7_dds_0	0x00100000	511 ...
ps7_ram_0	0x00000000	192 KB
ps7_ram_1	0xFFFF0000	~63,...

Basic Advanced

Place Code Sections in: ps7_ram_0

Place Data Sections in: ps7_ram_0

Place Heap and Stack in: ps7_ram_0

Heap Size: 1 KB

Stack Size: 1 KB

Generate Cancel

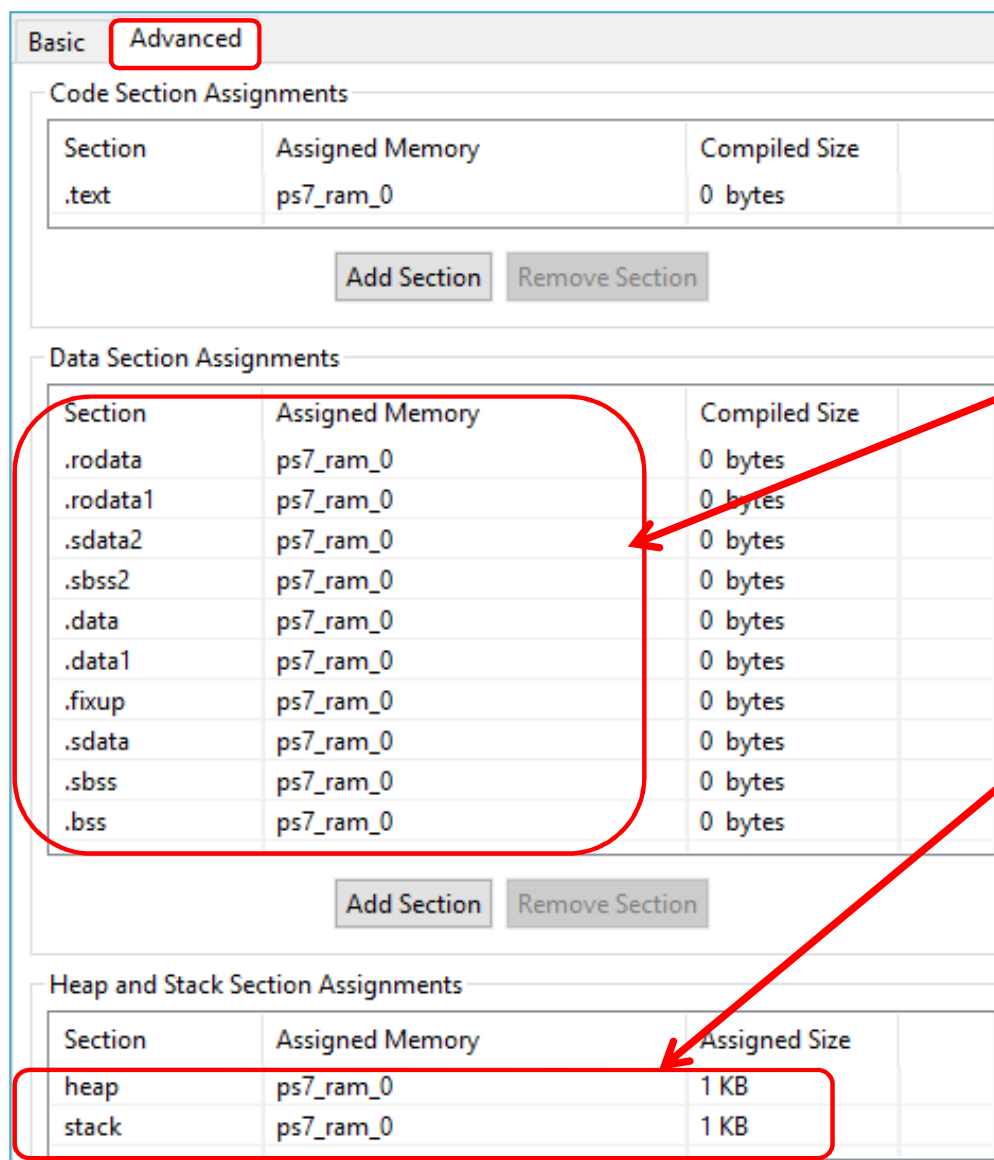
BRAM vs. külső memória területre kerüljenek?

- Programkód
- Változók
- Heap / Stack (méretük megadása)

Teszt alkalmazás készítése IV./b.

Linker script generálása (`lscript.ld`)

- Xilinx Tools menü → Generate Linker Script (**Advanced** nézet)



Basic **Advanced**

Code Section Assignments

Section	Assigned Memory	Compiled Size
.text	ps7_ram_0	0 bytes

Add Section Remove Section

Data Section Assignments

Section	Assigned Memory	Compiled Size
.rodata	ps7_ram_0	0 bytes
.rodata1	ps7_ram_0	0 bytes
.sdata2	ps7_ram_0	0 bytes
.sbss2	ps7_ram_0	0 bytes
.data	ps7_ram_0	0 bytes
.data1	ps7_ram_0	0 bytes
.fixup	ps7_ram_0	0 bytes
.sdata	ps7_ram_0	0 bytes
.sbss	ps7_ram_0	0 bytes
.bss	ps7_ram_0	0 bytes

Add Section Remove Section

Heap and Stack Section Assignments

Section	Assigned Memory	Assigned Size
heap	ps7_ram_0	1 KB
stack	ps7_ram_0	1 KB

- .text** — a végrehajtható kód/utasítás
- .rodata** — programkód végrehajtása során használt bármely „csak-olvasható” adat
- .data** — amelyben a „írható-olvasható” változók és „pointerek” tárolódnak
- .bss** — adat szegmens (ds) egy olyan része, amelyben a „statikusan allokalható” változók vannak
- .heap** — ahol a „dinamikusan allokalált” memória helyezkedik el
- .stack** — ahol a függvény hívás (CALL) paraméterei és más ideiglenes adatok tárolódnak

Fordítás (build) fontosabb lépései:

- 1.) Build Project → BSP
- 2.) Build Project → SW alkalmazás

Megjegyzés: alapértelmezettként az automatikus fordítás be van kapcsolva!

(Érdemes kikapcsolni:

Project menü → Build Automatically – nincs pipa)

Memória-teszt alkalmazás készítése VI.



SW alkalmazás fordításának eredménye:

```
'Building target: MemoryTest.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 -
L.../..../MemoryTest_bsp/ps7_cortexa9_0/lib -o "MemoryTest.elf"
./src/memory_config_g.o ./src/memorytest.o ./src/platform.o -Wl,--
start-group,-lxil,-lgcc,-lc,--end-group
'Finished building target: MemoryTest.elf'
'
'Invoking: ARM v7 Print Size'
arm-none-eabi-size MemoryTest.elf |tee "MemoryTest.elf.size"
   text    data    bss     dec     hex filename
 28368    1184    8248   37800   93a8 MemoryTest.elf
'Finished building: MemoryTest.elf.size'
```

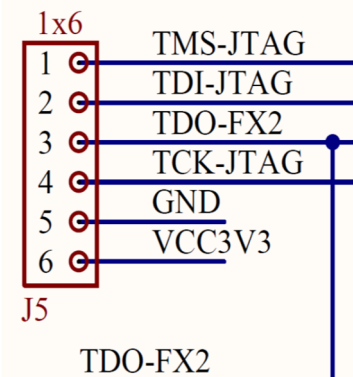
Decimal: 37800 byte ~38 KByte . Azaz a program el tud férni mind a belső on-chip RAM 0/1, mind pedig a külső DDR RAM memóriában. PL/FPGA-oldalon viszont ekkora BRAM memóriát kellene lefoglalni. Ezért generálódott a futtatható .elf fájl is sikeresen.

Beágyazott rendszer és szoftver teszt-verifikációja I.

- A USB-soros kábel (táp+programozó) csatlakoztatása
 - JP7 jumper = USB power!
 - JP5 jumper = JTAG mód!
- Az ZyBo kártya bekapcsolása

JTAG Programozó port
(opcionális, **nem ezt használjuk!**)

JTAG Header



ZyBo – Xilinx USB programming cable

VCC3V3 – piros VREF (6)
GND – fekete GND (5)
TCK-JTAG – sárga TCK (4)
TDO-FX2 – lilá – TDO (3)
TDI-JTAG – fehér TDI (2)
TMS-JTAG – zöld TMS (1)

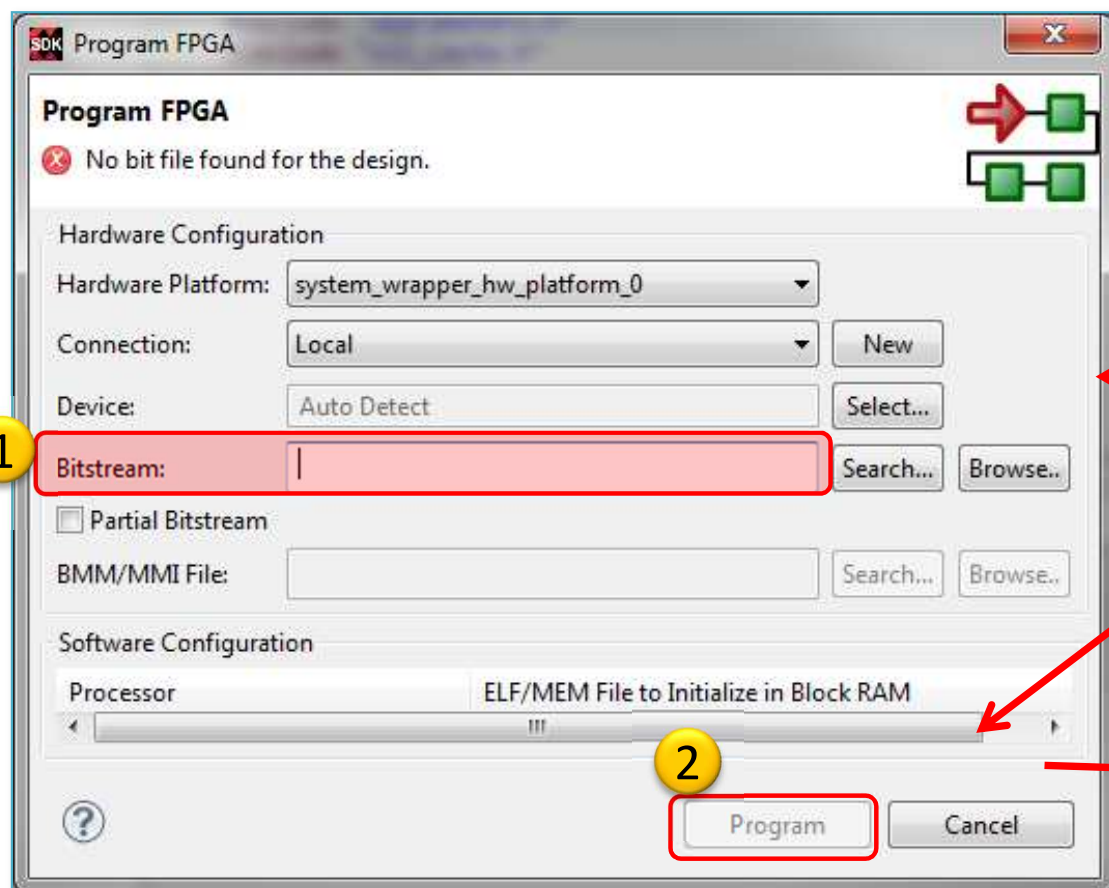
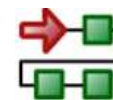
USB-Soros
csatlakozó (táp is
egyben)

Beágyazott rendszer és szoftver teszt-verifikációja I.

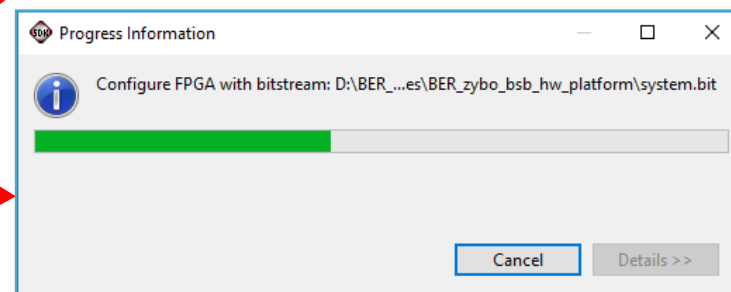


Amennyiben Vivado-ban konfigurálva lett a PL oldal is:

- FPGA felprogramozása: Xilinx Tools → Program FPGA
- MemoryTest.elf fájl kiválasztása → Program
 - A MemoryTest.elf és a system.bit, (system.bmm) fájlok összekombinálása (merge) → egyetlen download.bit fájlba.
 - Végül download.bit fájl letöltése az PL/FPGA-ra*



.bit (.bmm) fájlok,
valamint a
futtatható .elf fájl
megadása
→
download.bit
(FW+SW = Program)



HW-es hibakeresés konfigurálása

- **Jelöljük ki** a Project Explorer-ben az alkalmazást (MemoryTest)
- Run → **Debug Configurations...** (vagy jobb gomb Debug As...)

Új GDB Debug feladat létrehozása

1

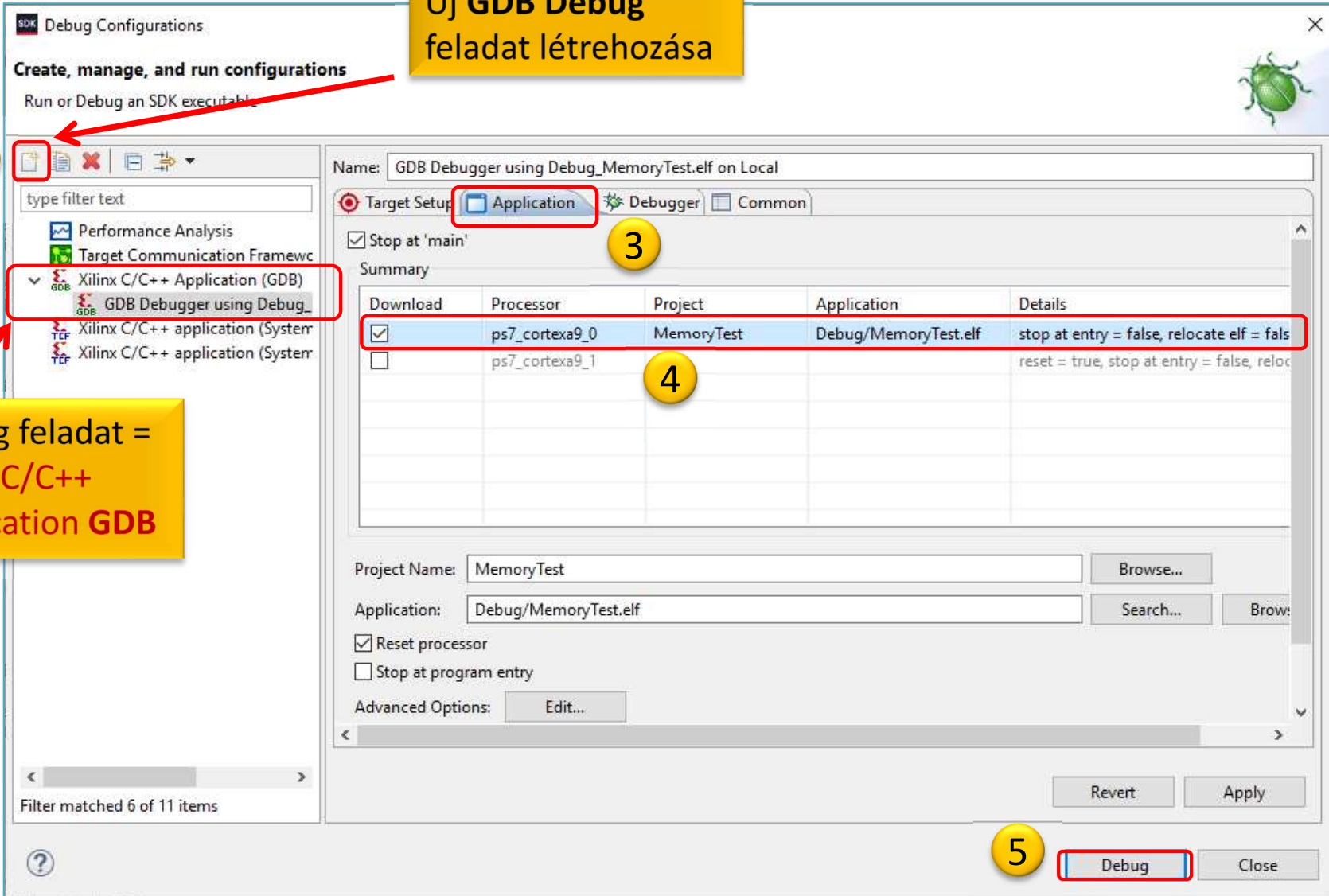
2

Debug feladat = Xilinx C/C++ application **GDB**

3

4

5



Create, manage, and run configurations
Run or Debug an SDK executable

type filter text

- Performance Analysis
- Target Communication Framework
- Xilinx C/C++ Application (GDB)
- GDB Debugger using Debug_MemoryTest.elf on Local
- Xilinx C/C++ application (System)
- Xilinx C/C++ application (System)

Name: GDB Debugger using Debug_MemoryTest.elf on Local

Target Setup **Application** Debugger Common

☒ Stop at 'main'

Summary

Download	Processor	Project	Application	Details
☒	ps7_cortexa9_0	MemoryTest	Debug/MemoryTest.elf	stop at entry = false, relocate elf = false
☐	ps7_cortexa9_1			reset = true, stop at entry = false, relocate elf = false

Project Name: MemoryTest Browse...

Application: Debug/MemoryTest.elf Search... Browse...

☒ Reset processor

☐ Stop at program entry

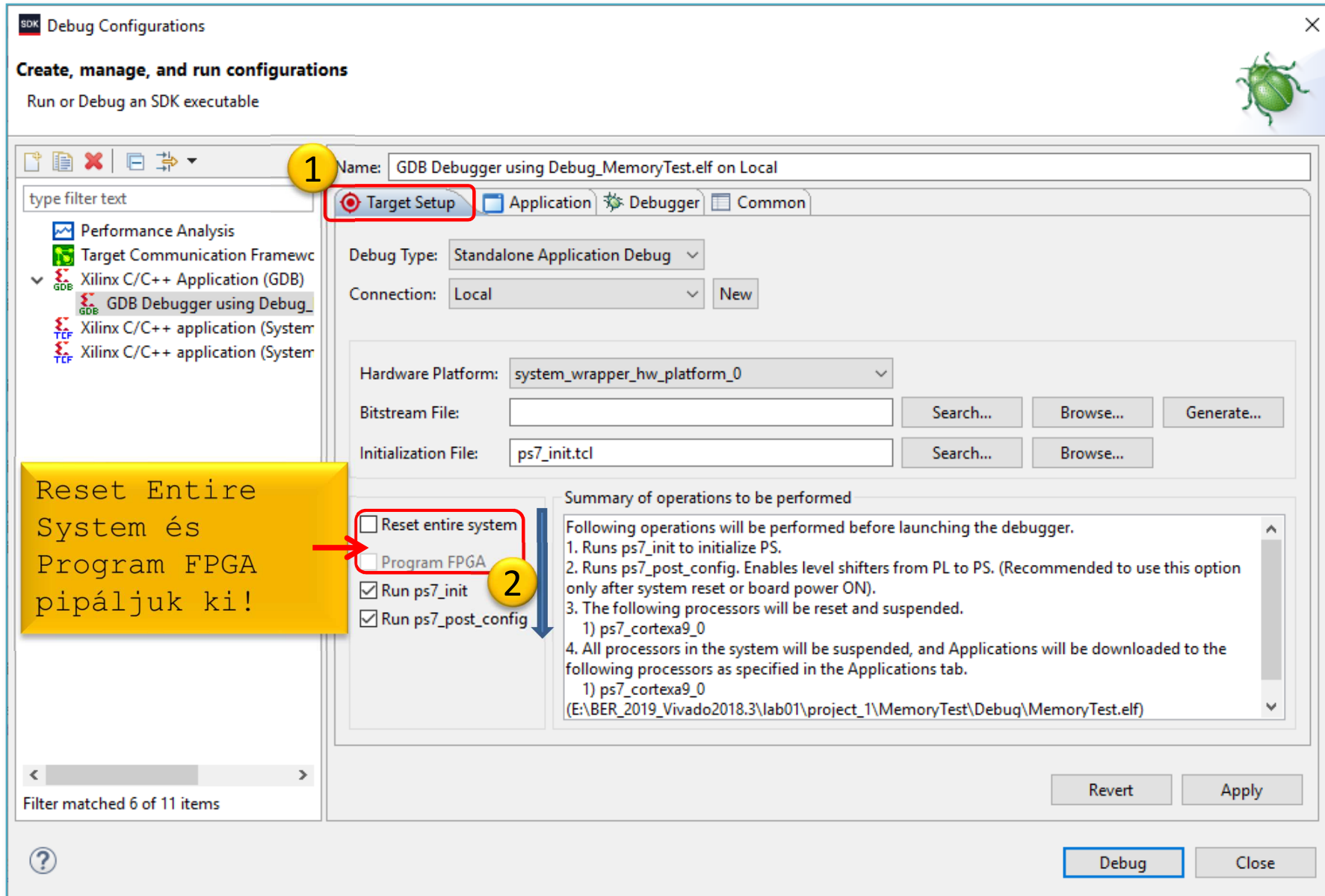
Advanced Options: Edit...

Revert Apply

Debug Close

HW-es hibakeresés konfigurálása

- Debug Configurations -> Target Setup



SDK Debug Configurations

Create, manage, and run configurations

Run or Debug an SDK executable

1 Name: GDB Debugger using Debug_MemoryTest.elf on Local

Target Setup Application Debugger Common

Debug Type: Standalone Application Debug

Connection: Local New

Hardware Platform: system_wrapper_hw_platform_0

Bitstream File: Search... Browse... Generate...

Initialization File: ps7_init.tcl Search... Browse...

Reset Entire System és Program FPGA pipáljuk ki!

☐ Reset entire system

☐ Program FPGA

☒ Run ps7_init

☒ Run ps7_post_config

2

Summary of operations to be performed

Following operations will be performed before launching the debugger.

1. Runs ps7_init to initialize PS.
2. Runs ps7_post_config. Enables level shifters from PL to PS. (Recommended to use this option only after system reset or board power ON).
3. The following processors will be reset and suspended.
 - 1) ps7_cortexa9_0
4. All processors in the system will be suspended, and Applications will be downloaded to the following processors as specified in the Applications tab.
 - 1) ps7_cortexa9_0(E:\BER_2019_Vivado2018.3\lab01\project_1\MemoryTest\Debug\MemoryTest.elf)

Revert Apply

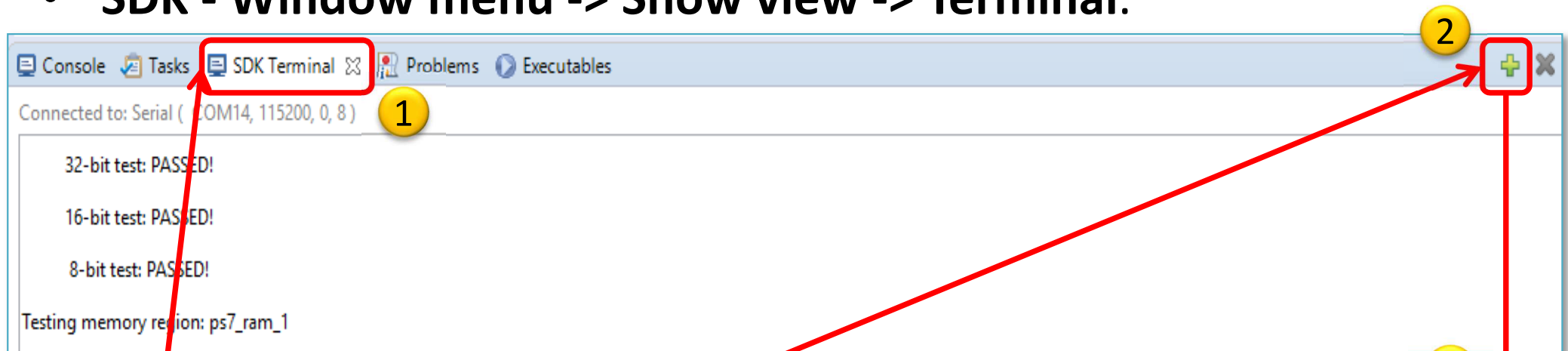
Debug Close

Filter matched 6 of 11 items

Debug soros port beállítása (SDK Terminal)



- SDK - Window menü -> Show view -> Terminal.



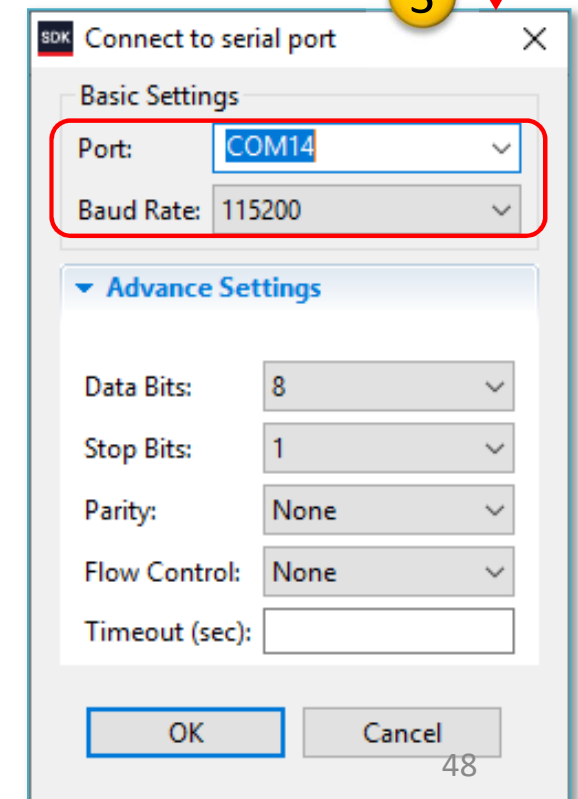
Terminál ablak

Soros port beállítása,
csatlakozás

Lehetőségek - Soros porti üzenetküldés használatára:

1. SDK Terminal: beépített vagy
2. külső program használata: (HyperTerminal, Putty stb.)

- Terminal: BaudRate / Adatbitek beállításai a **PS UART** vagy / **AXI_UART IP modulban** adottak szerint!
- Port: COM[XY] – beállítását a „Vezérlő pult → Hardver eszközök” beállításai szerint adjuk meg



Debug – Hibakeresés



Debug
eszközsor

Nézőke -
Változók

The screenshot shows the Xilinx IDE interface during a debug session. The top toolbar contains various icons for debugging, with a red box highlighting the 'Run' and 'Breakpoint' icons. The 'Debug' window on the left shows the GDB Debugger using 'Debug_MemoryTest.elf' on a local Xilinx C/C++ application. The 'Variables' window on the right shows a table with one variable:

Name	Type	Value
(*) i	int	30

The 'Breakpoint' window shows a breakpoint set at line 97 of 'memorytest.c'. The code editor displays the source code of 'main()'. A red box highlights the line: `print("NOTE: This application runs with D-Cache disabled.");`. The 'Outline' window on the right lists the project files. The 'Console' window at the bottom shows the output of the application, with a red box highlighting the text: `--Starting Memory Test Application--`.

Breakpoint

ARM/MicroBlaze
üzenet ablaka
(SDK terminal)

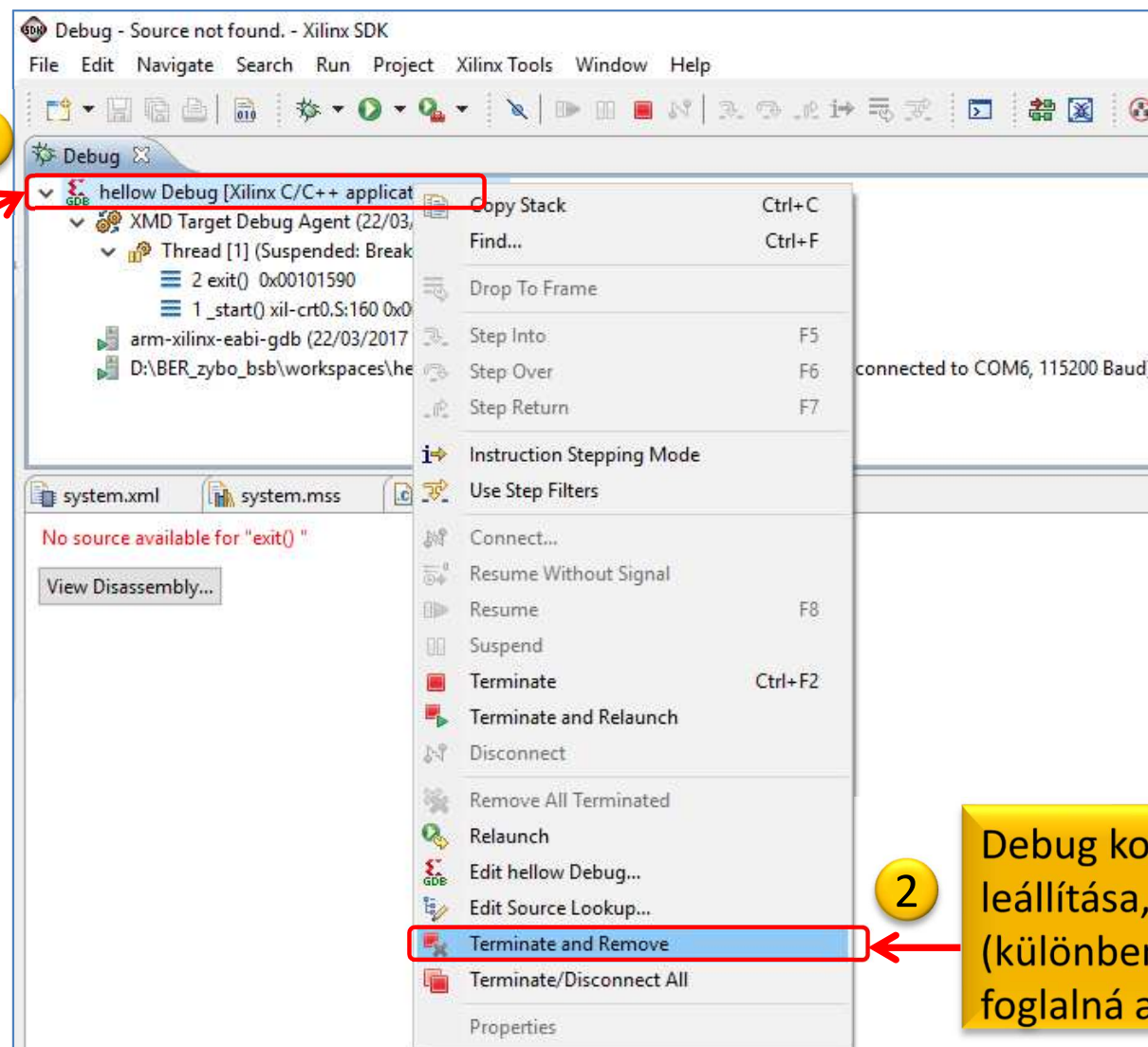
- Az SDK Terminal ablakban látható a memória-teszt alkalmazás futásának eredménye
- Ha az összeállított rendszer összeállítása helyes volt, akkor a tesztnek sikeresen le kell futnia (PASSED!), minden adatbit szélességre 😊

```
memory_test Debug [Xilinx C/C++ application (GDB)] D:\BER_zybo_bsb\workspaces\memory_test\Debug\memory_test.elf (22/03/2017 23:11) [Console connected to COM6, 115200 Baud]
--Starting Memory Test Application--
NOTE: This application runs with D-Cache disabled.As a result, cacheline requests will not be generated
Testing memory region: ps7_dds_0
Memory Controller: ps7_dds
Base Address: 0x00100000
Size: 0x1ff00000 bytes
32-bit test: PASSED!
16-bit test: PASSED!
8-bit test: PASSED!
Testing memory region: ps7_ram_1
Memory Controller: ps7_ram
Base Address: 0xffff0000
Size: 0x0000fe00 bytes
32-bit test: PASSED!
16-bit test: PASSED!
8-bit test: PASSED!
--Memory Test Application Complete--
```

ARM/PS oldal külső
512MB DDR3 RAM
(ps7_dds) illetve belső
On-chip 64K
memóriája (ram1)

Debug folyamat terminálása

- A HW-es hibakeresés végén nagyon fontos terminálni és eltávolítani a futtatott debug konfigurációt!



1
Debug ablak
– jobb gomb

2
Debug konfigurálásnak
leállítása, és eltávolítása
(különben folyamatosan
foglalná a memóriát!)

Példa 1.) Hello World

- MemoryTest-hez hasonlóan hozzon létre egy template-ből generálható **HelloWorld** alkalmazást (lásd köv oldal)
- (hozzá tartozó saját BSP-vel).
 - Mekkora a Hello_world.elf file mérete? (~28.7 - 37 Kbyte).

- A main függvényben a következő a módosítás (printf):

```
{  
    init_platform();  
    unsigned int i = 0;  
    do{  
        //printf("Hello World (%d)\n\r",i); //printf() = nagy  
        mem. foglalása van  
        xil_printf("Hello World (%d)\n\r",i); //xil_print()  
        vagy print() kis mem. foglalás. FPGA-ra optimalizált.  
        i++;  
    }while(1);  
    return 0;  
}
```

- Kérdés: Mekkora lett a printf()-el fordított .elf file mérete?

Példa 2.) Kód analízis

- Vizsgálja meg a **MemoryTest** SW mintaalkalmazás forráskódjait:
- **\src**
 - `memory_config.h` (struktúra definíció)
 - `memory_range_s`
 - `memory_config_g.c` (struktúra *ddr* vs. *ram1*)
 - *memorytest.c* (main függvény)
 - `platform_config.h`
 - `platform.c`
- **\<*_bsp>\ps7_cortexa9_x\include**
 - `xparameters.h` (`#defines` **.hdf/.xml** szerint)

Példa 3.)

- Módosítsa úgy a memória teszt alkalmazást úgy, hogy az a különböző *8-/16-/32-bites* adatszélességeknél a jelenleginél **nagyobb** külső *ddr0* memória tartományt (1024 szó helyett ... akár 8-32...512 MByte!) teszteli le!
- Segítség: 511 MB = 0x1FF0_0000 = range->size
 - TestMem32(): (range->size) / 64 (?), esetleg
 - TestMem16(): (range->size) / 32,
 - TestMem8(): (range->size) / 16.

Példa 3.) folytatás

- „memset()” függvénnnyel nullázzuk ki a tesztelendő régiókat és adjuk a `memorytest.c` alkalmazás kódhoz:

Name <https://linux.die.net/man/3/memset>

memset - fill memory with a constant byte

Synopsis

```
#include <string.h> void *memset(void *s, int c, size_t n);
```

Description

The `memset()` function fills the first *n* bytes of the memory area pointed to by *s* with the constant byte *c*.



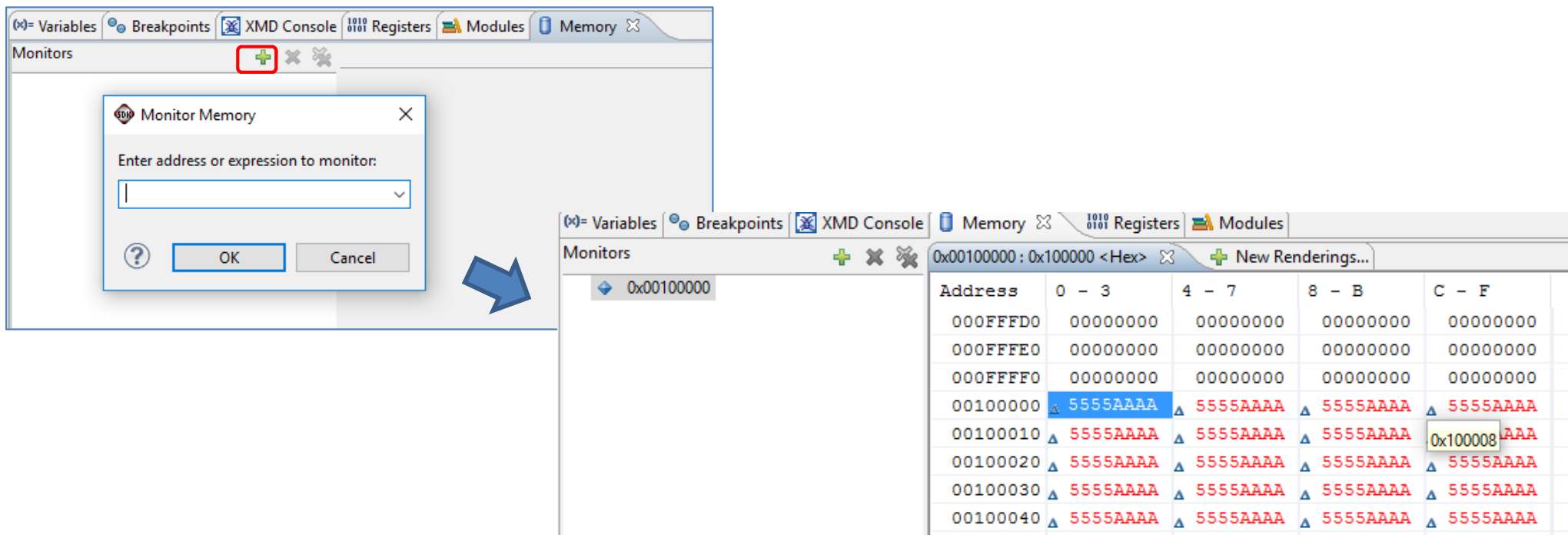
```
#include "string.h"
...
memset((u32*)range->base, 0x0, 1024);
status = Xil_TestMem32((u32*)range->base, 1024, 0xAAAA5555, XIL_TESTMEM_INCREMENT);
print("          32-bit test: "); print(status == XST_SUCCESS? "PASSED!":"FAILED!");
print("\n\r");

memset((u16*)range->base, 0x0, 2048);
status = Xil_TestMem16((u16*)range->base, 2048, 0xAA55, XIL_TESTMEM_INCREMENT);
print("          16-bit test: "); print(status == XST_SUCCESS? "PASSED!":"FAILED!");
print("\n\r");

memset((u8*)range->base, 0x0, 4096);
status = Xil_TestMem8((u8*)range->base, 4096, 0xA5, XIL_TESTMEM_INCREMENT);
print("          8-bit test: "); print(status == XST_SUCCESS? "PASSED!":"FAILED!");
print("\n\r");
```

Példa 4.)

- Állítsuk be a HW-es hibakaresés során a Memory ablakot:
 - Window → Show View → Memory
 - [+] Adjuk meg: **0x100000** (vagy **0xffff0000**)



The screenshot shows the Xilinx IDE interface. On the left, the 'Monitors' window is open, and a 'Monitor Memory' dialog box is displayed. The dialog box has a text field labeled 'Enter address or expression to monitor:' and buttons for 'OK' and 'Cancel'. A red box highlights the '+' button in the 'Monitors' window. A blue arrow points from the 'OK' button in the dialog box to the 'Monitors' window.

On the right, the 'Memory' view is shown, displaying a table of memory addresses and their values. The address 0x00100000 is selected, and the value is 5555AAAA. The table has columns for Address, 0 - 3, 4 - 7, 8 - B, and C - F.

Address	0 - 3	4 - 7	8 - B	C - F
000FFFD0	00000000	00000000	00000000	00000000
000FFFE0	00000000	00000000	00000000	00000000
000FFFF0	00000000	00000000	00000000	00000000
00100000	5555AAAA	5555AAAA	5555AAAA	5555AAAA
00100010	5555AAAA	5555AAAA	5555AAAA	5555AAAA
00100020	5555AAAA	5555AAAA	5555AAAA	5555AAAA
00100030	5555AAAA	5555AAAA	5555AAAA	5555AAAA
00100040	5555AAAA	5555AAAA	5555AAAA	5555AAAA

A HW-es hibakaresés előtt érdemes kikapcsolni az FPGA-s kártyát (random memória bejegyzések miatt) !

- A **Block Designer** segítségével könnyen és egyszerűen lehet létrehozni ARM/MicroBlaze processzor alapú beágyazott rendszereket a Xilinx Vivado-ban
- A folyamat részeként számos implementációs fájl is létrejön köztük a rendszert leíró **.HDF** (Hardver Definition File) fájl.
- A Vivado IP Integrator különböző nézetei segítségével beállíthatjuk az összeállított beágyazott rendszer komponenseit, és különböző paramétereit.
- A rendszer összeállítása után generálhatjuk a konfigurációs állományt (bitstream, amennyiben van PL tartalom!).
- A létrehozott **BSP-re** épülően a Software Development Kit (**SDK**) segítségével készíthetjük el a szoftver alkalmazást, melynél számos template (sablon) áll a rendelkezésünkre.
- A HW/FW és a SW helyes működése verifikálható a bitstream **.BIT**+futtatható **.ELF** FPGA-ra történő letöltésével.