



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

***Lecture Notes and Practical Courses on System on Chip (SoC):  
Materials for Embedded Systems Subjects***

József VÁSÁRHELYI & Ahmed BOUZID

First Edition

**SZÉCHENYI** 2020



MAGYARORSZÁG  
KORMÁNYA

Európai Unió  
Európai Szociális  
Alap



**BEFECTETÉS A JÖVŐBE**



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

Authors: *Dr. habil. József VÁSÁRHELYI, Associate Professor*  
*Dr. Ahmed BOUZID, Adjunct Professor*  
Lecturer: *Dr. Rabab BENOTSMANE, Assistant Lecturer*  
Editor: *University of Miskolc*  
Keywords: *Embedded Systems, FPGA, SoC, Computational Technologies, HDL*

November 2021

All rights reserved. No part of this book may be reproduced, in any form or by any means without permission in writing from the publisher.

ISBN 978-963-358-238-1

**SZÉCHENYI** 2020



MAGYARORSZÁG  
KORMÁNYA

**Európai Unió**  
Európai Szociális  
Alap



**BEFEKTETÉS A JÖVŐBE**



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

ΛΥΟΛΣΣ ΛΙΜΚΥ ΑΥΘΗ ΔΞΗΤΙ

Translation: There is no rose without thorns.

Hungarian proverb

Q.ŁŁ ƆXXQ X.Θ Ł. §WM. †Φ%Ж.Э.О. Լ.Վ.ՃԻՃ ԻՕ §ՕԷՕ.Լ

Translation: Always aim for the moon; if you miss, you'll land among the stars.

Amazigh (Berber) proverb



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

# Contents

<b>1</b>	<b>Embedded System Architectures – System on Chip Anatomy</b>	<b>1</b>
1.1	Introduction to Embedded Systems . . . . .	1
1.2	System on Chip (SoC) . . . . .	1
1.2.1	Introduction to SoCs . . . . .	1
1.2.2	The Principle of SoCs . . . . .	3
<b>2</b>	<b>The Zynq device</b>	<b>9</b>
2.1	Introduction . . . . .	9
2.2	Zynq-7000 Overview . . . . .	10
2.3	The Zynq Processing System . . . . .	10
2.3.1	ARM processor . . . . .	10
2.3.2	Memory interfaces . . . . .	16
2.3.3	Processing System Interconnect . . . . .	16
2.3.4	Memory Map . . . . .	16
2.3.5	PS Boots first . . . . .	16
2.3.6	Clock resources . . . . .	17
2.4	Summary . . . . .	17
<b>3</b>	<b>VHDL hardware description language</b>	<b>19</b>
3.1	Introduction to Hardware Description Language . . . . .	19
3.2	Hardware Abstraction Levels . . . . .	20
3.3	Textual description of digital systems . . . . .	21
3.3.1	VHDL hardware description language . . . . .	22
3.3.2	Verilog hardware description language . . . . .	22
3.3.3	SystemC hardware description language . . . . .	23
3.4	VHDL basics . . . . .	23
3.4.1	Entity as the interface model . . . . .	25
3.4.2	Architecture as model functionality, behavior, structure . . . . .	25
3.4.3	Constants and Signals . . . . .	27
3.4.4	Packages, Libraries . . . . .	27
3.4.5	VHDL grammar and lexical elements . . . . .	28
3.4.6	VHDL Objects . . . . .	28
3.4.7	Data types and operations . . . . .	29





„FÖNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

3.5	Behavioral system description in VHDL . . . . .	30
3.5.1	Combinational Signal Assignments . . . . .	30
3.5.2	Sequential signal assignments . . . . .	31
3.5.3	Conditional Decisions in processes . . . . .	34
3.6	Structural description in VHDL . . . . .	36
<b>4</b>	<b>Hardware and software setup</b>	<b>42</b>
4.1	Creating an SSH tunnel . . . . .	42
4.1.1	Password change . . . . .	47
4.1.2	Login to the internal machine . . . . .	48
4.2	Installing Xilinx Vitis . . . . .	48
4.3	UART-USB driver install . . . . .	49
<b>5</b>	<b>Embedded system Design with Zynq, hardware and software design basics</b>	<b>51</b>
5.1	Lab 1 . . . . .	52
5.1.1	Lab 1 Part 1: Hello World . . . . .	52
5.1.2	Lab 1 Part 2: Led Blinking through MIO . . . . .	80
5.2	Lab 3: Create your own IP . . . . .	99
5.2.1	Create and Manage IP Project . . . . .	100
5.2.2	Create the Vivado Project with User Designed IP . . . . .	115
5.2.3	Launch Vitis and Generate the Test Application of the project . . . . .	131
<b>6</b>	<b>Directed Projects</b>	<b>142</b>
6.1	Project 1: GPS parsing using SoC for Position Determination. Simulation then HW/SW Implementation . . . . .	143
6.1.1	Phase 1: Coding and Simulation . . . . .	143
6.1.2	Phase 2: Hardware design, Implementation and Test . . . . .	143
6.2	Project 2: Remote DC Motor Control using SoC and Hall effect Sensor . . . . .	144
<b>7</b>	<b>Exercises</b>	<b>145</b>
7.1	Multiple Choice Quizzes . . . . .	145
7.2	Questions . . . . .	146
<b>A</b>	<b>Additional Content</b>	<b>151</b>
A.1	GPS Data . . . . .	151
A.1.1	10 seconds 1Hz GPS NMEA (correct data) . . . . .	151
A.1.2	10 seconds 1Hz GPS NMEA (incorrect data) . . . . .	152





## Chapter 1

# Embedded System Architectures – System on Chip Anatomy

### 1.1 Introduction to Embedded Systems

Is a computing system that has been designed to accomplish a single specific application. Embedded systems can be based on homogeneous and heterogeneous technologies. The most familiar ones are the digital, analog or mixed-signal integrated circuits. However, some other technologies are not excluded since they offer better performances for some applications, for instance Quantum computers, Integrated Fluidic Circuits, Photonic Integrated Circuits and many other Unconventional computing systems.

Microcontrollers and Microprocessor are basically not embedded systems before being programmed.

### 1.2 System on Chip (SoC)

#### 1.2.1 Introduction to SoCs

The advance of microelectronics allows the reduction of the size of transistors to get them embedded in increasingly small spaces. This ability to miniaturize components gives us the possibility of integrating different components in the same chip that were previously connected in a PCB (Printed Circuit Board) (See fig. 1.1). We are therefore witnessing the development of SoCs (System on Chip). Previously a digital computer was a single logic gate. Gathering those logic gates according to a specific architecture gives birth to processors. The packaging and miniaturization of these processors is the key point in the creation of microprocessors. What mainly differs between microprocessors and microcontrollers is the autonomy in terms of computation; in other words, microprocessors need other components, called peripherals (e.g. memory, I/O ...), for normal operation. That said, if in addition to peripherals





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

are added other computing units from other technologies (Graphical Processors, Sound Processors ...), gives a System-on-Chip.

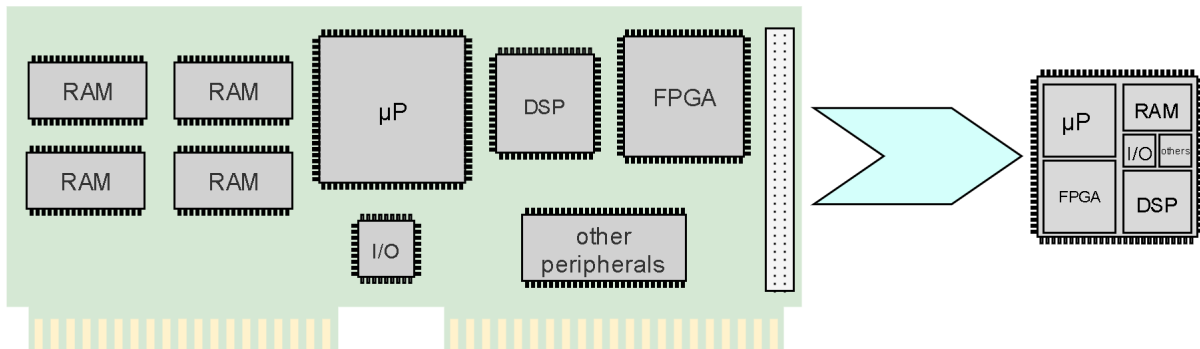


Figure 1.1: Miniaturization from a Printed Circuit Board to a System on Chip

The interest is to minimize the size, the energy consumption and above all to expand the fields of application. SoCs are a systematic result of the miniaturisation of processors, in consequence, they are useful for creating custom architectures that can accelerate microcontrollers or microprocessors for some tasks.

A hardware accelerator is a computational element that helps the main processing system to perform some tasks. The latter can be: matrix computations, floating point operations, image processing, digital signal processing... In addition to some tasks that cannot be performed by a digital processor such as: conditioning and analog signal processing.

SoCs allow to create other architectures and take benefits of design reuse by means of the emergence of IPs (Intellectual Properties). This leads to the ease of reuse, the reduction of time to market and therefore cheaper products.

### Moore's Law

According to Gordon Moore, in 1965 the number of transistors in a given area on a chip could double every 18-24 months [Moo98]. In other words, it is possible to double the efficiency of semiconductor technology, and therefore chips, every two years or so while keeping the same power consumption. This law has enabled manufacturers of microelectronic manufacturing to clearly define their objectives and avoid unfair competition. 40 years after Moore's prediction, i.e. in 2005, the latter himself mentioned the future of this law which will end around 2025. Indeed, if we follow the evolution curve of the size of the transistors (Fig. 1.2), we can see that by 2025 the size will be around 2nm which is roughly equivalent to the size of two glucose molecules.



„FÖNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

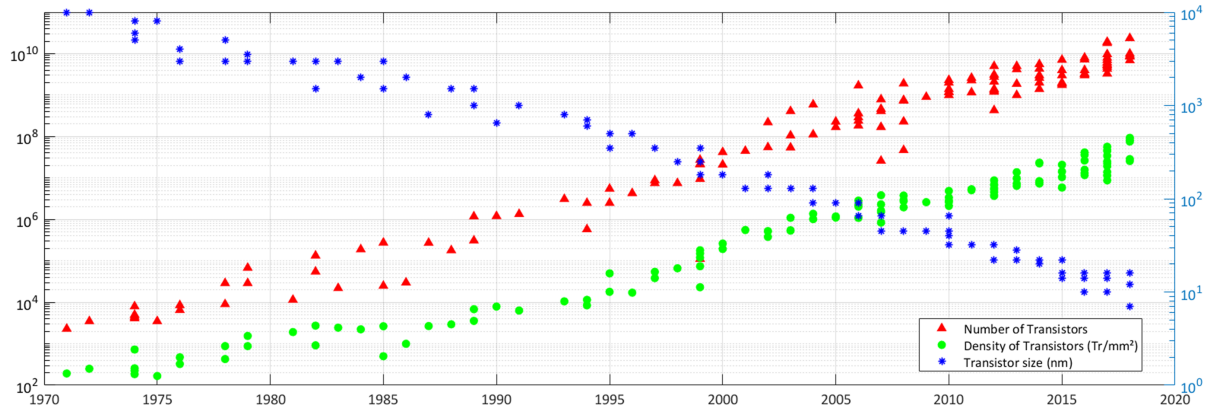


Figure 1.2: The evolution of the number of transistors, their density and size

## 1.2.2 The Principle of SoCs

A System-on-Chip (SoC) is a circuit that packages computing elements into a single die dedicated for an autonomous task. From this definition one could understand that previously the elements of calculations were separated in different chips. These computers can be simple transistors from their basic purposes/function (switch inverter, amplification), to more complex circuits (memories, processors...). Computers have more and more functionalities thanks to the ability to merge multiple heterogeneous technologies in the same chip such as mixed-signal integrated circuits.

An SoC can be defined by the integration of computational technologies into a single chip. From this definition SoC could be divided into 3 types, these types are chronologically ordered:

**Specialized SoCs**, which also belongs to the family of ASICs (Application-Specific Integrated Circuit) since they are dedicated to a specific application.

The first System-on-Chip was an Intel 5810A CMOS chip created in 1972 for Microma-Seiko digital watches. The IC has two functions: LCD driver and timer. [Cor76] [Jr.07]

The Microma liquid crystal display (LCD) digital watch is the first product to integrate a complete electronic system onto a single silicon chip. [CGL<sup>+</sup>20] [VSM01] [HKM08]





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015



Figure 1.3: The first wristwatch equipped with a dedicated SoC: The Seiko Microma in 1974 [Pre14]

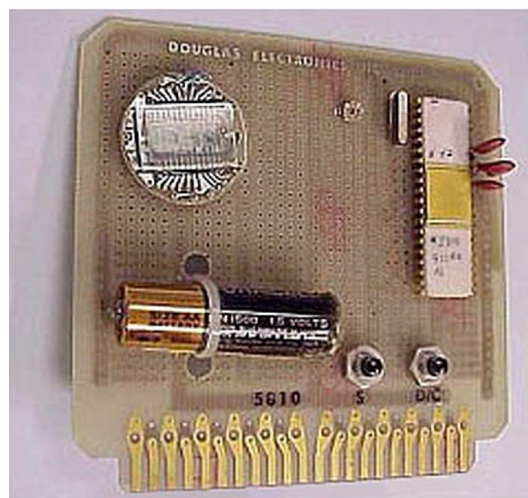


Figure 1.4: The prototype of the Intel 5810A based watch [VSM01]

**Programmable SoCs.** This category concerns chips which contain a microprocessor, peripherals, and a programmable logic area (or FPGA). Zynq is a family of Xilinx SoCs which will be used to implement the labs proposed in this teaching material (see fig. 1.6). Actual research works propose chips where a programmable analog area is also included in the same chip which allows flexibility for mixed-signal (analog and digital) designs such as the RASP 3.0 (Reconfigurable Analog Signal Processor) (see 1.8). Actual Programmable SoCs (such as ACAP from Xilinx) are mainly a combination of microprocessors, FPGA, DSPs, GPU, RF, and notably Intelligent Engines capable of implementing artificial intelligent



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

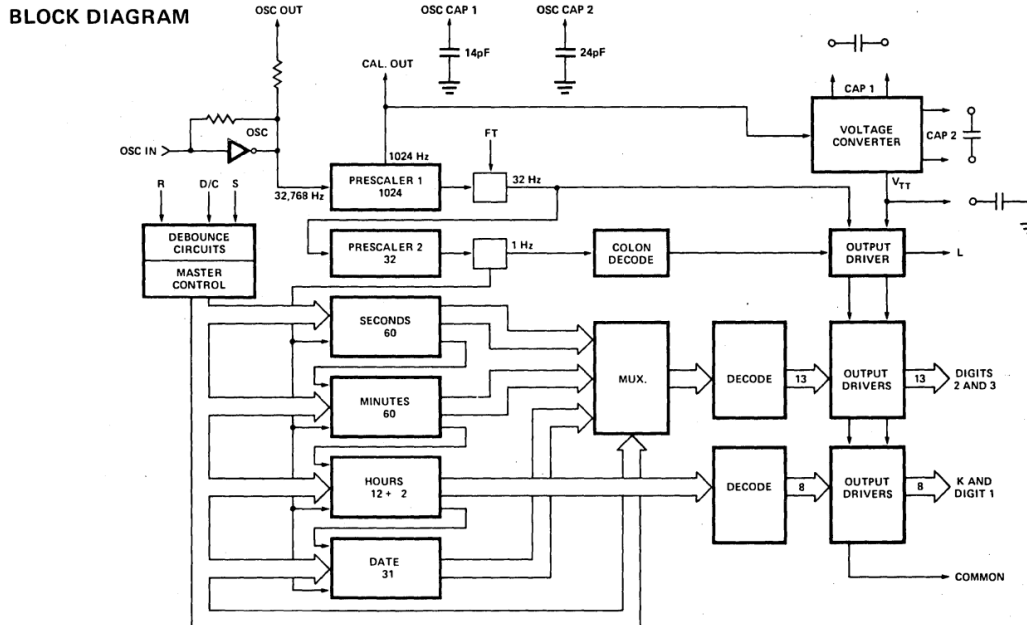


Figure 1.5: Block diagram of the first SoC: the Intel 5810A [Cor76]

algorithms (See fig 1.7). Unlike Programmable SoCs, the previous category has the disadvantage of its fixed architecture and its impossibility to upgrade.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

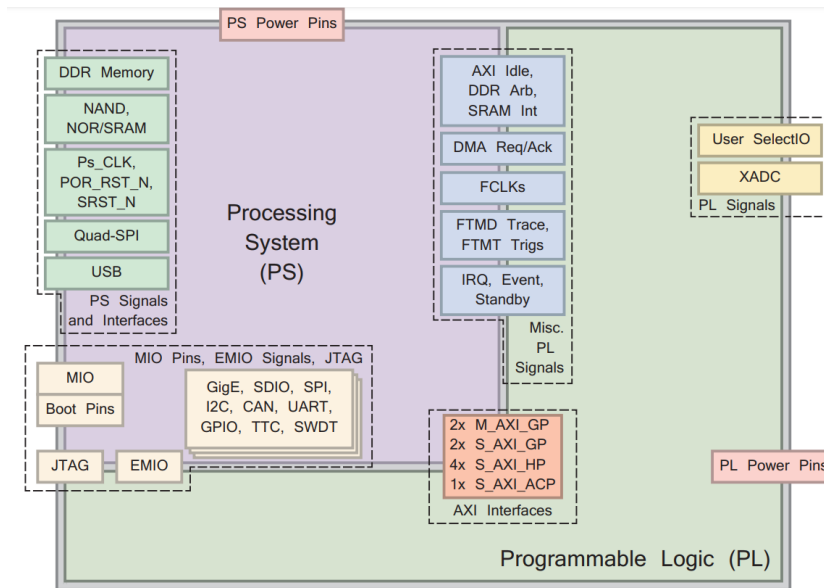


Figure 1.6: The architecture of Xilinx's Zynq Z-7010 SoC

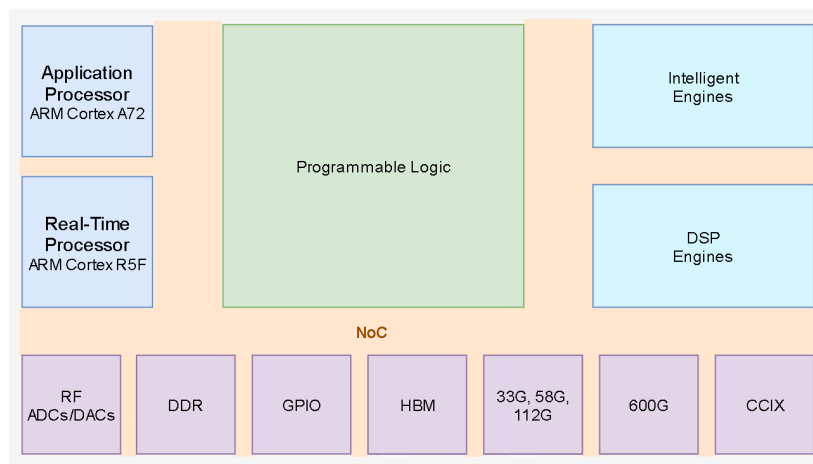


Figure 1.7: The architecture of Xilinx's ACAP (Adaptive Compute Acceleration Platform)





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

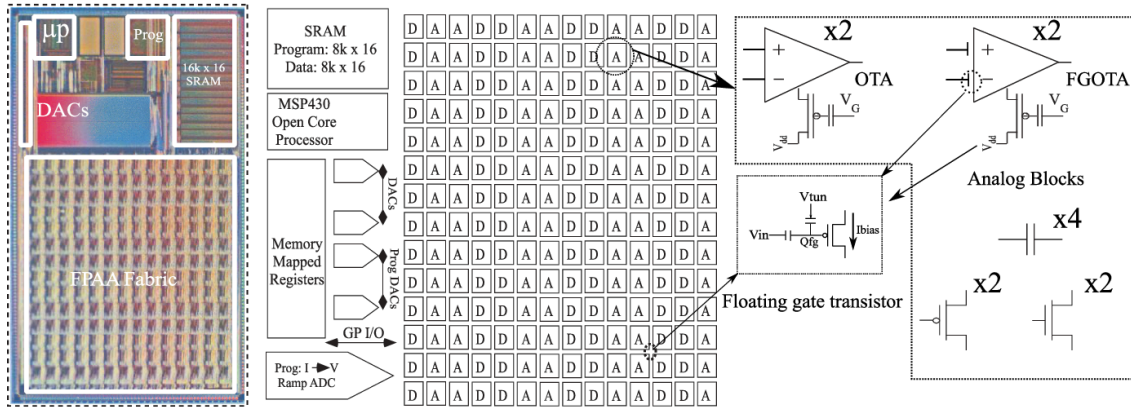


Figure 1.8: The architecture of the RASP 3.0 (Reconfigurable Analog Signal Processor) [STGH19]

**Microprocessor based SoCs.** This type constitute the main concern of the actual market of mobile phones, tablets, and some laptops. In addition to a multicore microprocessor, these chips mainly contain a GPU, DSPs, memory, and peripherals. Comparing to Programmable SoCs, this type of SoCs are not reconfigurable i.e., its architecture is fixed, however contain much higher performance processors.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

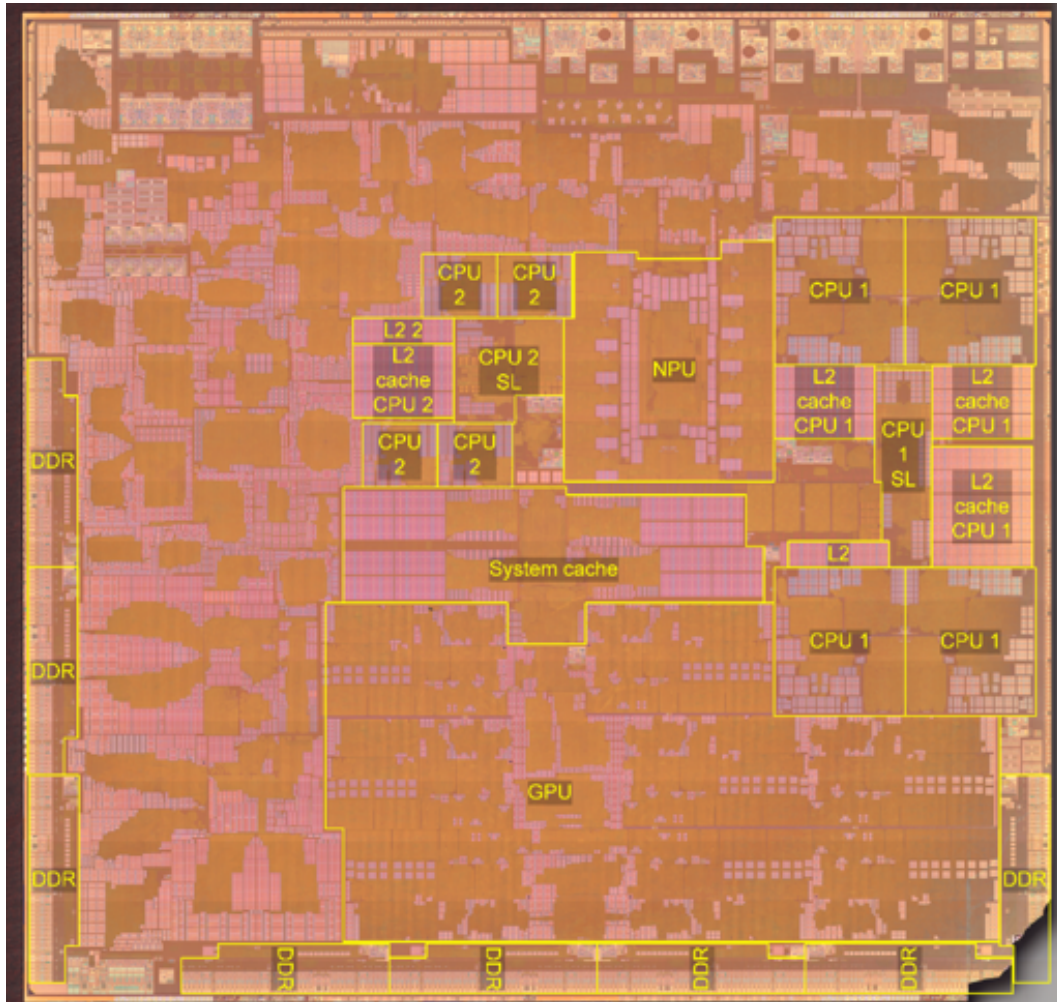


Figure 1.9: Detailed die photo of the Apple's M1 SoC (2020) [tec]





## Chapter 2

# The Zynq device

### 2.1 Introduction

In the previous chapter was mentioned the impact of Moore’s law on the evolution of integrated circuits. There were treated the SOC structures in general and was mentioned that the first system on chip was created by Intel. We have to clarify that this Intel’s SOC was not programmable by the user. In this sense this Intel 5810A was an Application Specific Integrated Circuit (ASIC). Before we introduce the Zynq device let us mention some milestones, which lead to the All Programmable System on Chip (AP-SOC) architecture, which is the Zynq device and also should be here mentioned the Adaptive Compute Acceleration Platform (ACAP), which is much more than a system on chip. First it should be mentioned Tsugio Makimoto’s paper [Mak02] in which the author detailed that evolution of integrated circuit (IC) technology every ten years oscillating between the standardization and customization depending on the “technology crisis”. Also the author predicted the second digital wave, which came after the PC world (representing the first digital wave). In this sense it was easy to predict that different user programmable circuits and fix function IC will converge in what we call today APSOC and ACAP. We can affirm that because of the “von Neumann centered” engineering world, since the appearance of the FPGA on the market, it became a challenge to implement in the FPGA a processor. VHDL (Very High Speed Integrated Circuit Hardware Description Language) offered the solution to integrate a so called *soft processor* in the FPGA. These processors (8051 and PIC) were not integrated physically in the chip but they were described in VHDL and embedded in the user design. One can realize that almost from beginning of the FPGA era appeared the need to have a processor together with the FPGA in the same chip.

The first *hard processor* physically integrated in a System on Chip (hard processor and FPGA) was the 8032 micro-controller realized by Triscend in 1998 and later this company integrated the ARM7 processor (Xilinx acquired Triscend in 2004). Since the department always used Xilinx FPGA and APSOC in research and education this chapter focus on the most current device used in the embedded systems, which is the Zynq device. The chapter will not give a detailed description of the device, since there is





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

a huge literature about the Zynq device [[ARMB], [CEES14], [Xil18b]] and other materials found on the Xilinx webpage.

## 2.2 Zynq-7000 Overview

Based on the data sheet [Xil18b] The Zynq-7000 device is first generation architecture based on the Xilinx SoC architecture. The chip processing system (PS) integrate in the device a dual core ARM® Cortex™-A9 processor and programmable logic (PL) based on the 28 nm technology. Strictly speaking the Zynq device PS shows micro controller structure with its on-chip memory and external memory interfaces with a rich set of peripheral interfaces such as DMA controller, Ethernet MAC, USB 2.0B OTG interfaces, CAN 2.0B controllers, SD/SDIO 2.0/MMC3.31 compliant controllers, full duplex SPI ports, high-speed UARTs and multiplexed IOs for peripheral pin assignment. The connectivity within the PS and between the PS and PL it is realized with the AMBA® AXI standard interface bus. The programmable logic contains Configurable Logic Blocks (CLB), 36 Kb Block RAM, Digital Signal Processing (DSP) blocks, Programmable I/O Blocks (IOB), JTAG Boundary Scan interface, PCI Express® Block and two 12 bit Analog to Digital (AD) converters. With this reach architecture the Zynq-7000 device target "cost sensitive as well as high performance" embedded applications. In the following pages we will present the PS and the PL system. We underline that for an effective SoC embedded system development it is very important to have a deep knowledge about the hardware structure.

## 2.3 The Zynq Processing System

The hard processor integrated in all of Zynq devices is the ARM Cortex 9. The processing system incorporates not just the the ARM processor but also resources that come to extend the processor to a complex dual core micro controller. The Application Processing Unit (APU) is a dual core processing system. Each ARM processor has instruction and data cache, memory management unit (MMU) and a floating point NEON media processing engine for SIMD (Simple Instruction Multiple Data) support. The Accelerator coherency port (ACP) interface enable a coherent access from PL to CPU memory space. The 8-channel DMA supports multiple transfer types. Axi interface enable high throughput DMA transfers. Also the DMA has 4 channels to the PL. Interrupt are handled by the General Interrupt Controller (Gic), program execution can be controlled by the Watchdog Timers. The APU also has two triple timers/counters.

### 2.3.1 ARM processor

The ARM core have RISC architecture. RISC architectures have simple and powerful instructions, which are executed in a single clock cycle. Basically the RISC architecture reduces the complexity of instructions performed by the hardware and provide greater flexibility in software. For the ARM processors the RISC philosophy is implemented with four major considerations as detailed in [SSW04].





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

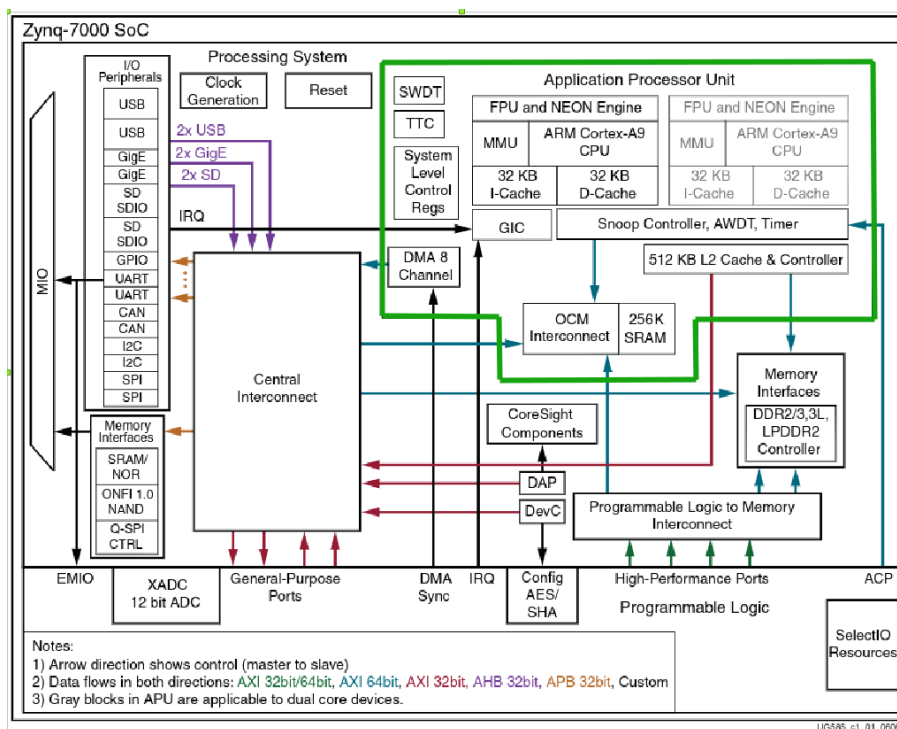


Figure 2.1: The Zynq Processing System [Xil18b]







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

1. RISC Instructions - processors have a reduced number of instructions. All instructions can be executed in single clock cycle and each have a fixed length to allow pipeline instruction execution.
2. Pipelines - Multiple instructions are executed in parallel, each instruction, which participate in the pipeline execute a separate phase of the instruction execution.
3. Registers - The processor have a large number of general purpose register set. Any register can store data or address.
4. Load-Store architecture - The processor process only data stored in registers. To transfer data between the register and external memory there are special load and store instructions. This comes from the fact that memory instructions are costly.

### ARM registers

The Cortex A9 processor implements the ARMv7-A architecture that include the following architecture extensions: SIMD architecture for integrated floating-point vector operations; vector floating point computation (IEEE 754 standard); security extension for enhanced security; multiprocessing extensions for multiprocessing functionality ([ARMb]). General purpose registers are available in user mode, which hold data or and address. The register bank contain sixteen general purpose registers plus the special purpose register bank. Figure 2.2 represent a general overview of the registers. In the figure the special registers can differ from the represented one.

Registers R0 - R12 are general purpose registers, which are divided in low registers (R0-R7) and high registers (R8-R12). The low registers can be accessed by any instructions, while the high registers cannot be accessed by some Thumb instructions.

Register R13 is the Stack Pointer records the current address of the stack. R13 it is used for saving the context of a program while switching between tasks. It has to stack pointer (SP) registers. On of SP is the Main stack pointer (MSP), used in applications that require privileged access such as operating system kernel, and exception handlers. The other stack pointer is the Process Stack Pointer (PSP), which is used in base-level application code (when not running an exception handler).

R14 is the Link Register (LR) used to store the return address of a subroutine or a function call.

R15 register is the Program Counter (PC) used to record the address of the current instruction code. It is automatically incremented by 4 at each operation (for 32-bit instruction code). Exceptions are the branching operations. A branching operation, such as a function calls, will change the PC content to a specific address, meanwhile save the current PC to the Link Register. The PC will load the value from LR after a function is finished.

Special register provide information about the program execution status, and provide the ALU (Arithmetical Logical Unit) flags. The xPSR register contains three registers, such as Application PSR (APSR) containing the ALU flags, the Interrupt PSR (IPSR), containing executing interrupt service routine number and the EPSR the Execution PSR (EPSR). For a detailed register description please consult [ARMb] Technical Reference Manual.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

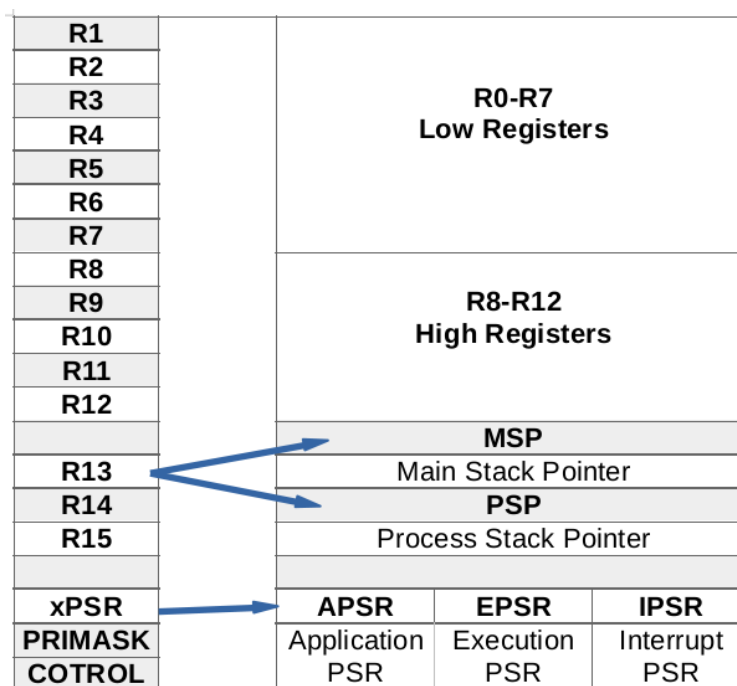


Figure 2.2: ARM registers available in user mode





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

## The Advanced Micro-controller Bus Architecture (AMBA)

For System on Chip architectures there are two bus types: external and internal bus.

Table 2.1: ARM AMBA bus types and examples

AMBA Family	Bus Protocol	Processor
AMBA5	CHI	Cortex-A57, A53
AMBA4	ACE	Cortex-A7, A15
	AXI4	
AMBA3	AXI	Cortex-A9, A8, R4, R5
	AHB	Cortex-M0, M3, M4
	APB	Cortex-M0, M3, M4
	ATB	
AMBA2	AHB, ATB	ARM7, ARM9
AMBA1	ASB, APB	

The external bus connects off-chip memory, and other external devices. The Zynq processing system has an external memory interface to connect external memories such as DDR2, DDR3 and other newer memory types. In this way the user do not have to care about external memory interfacing. The internal bus connects internal components inside the chip. This bus is known as system bus with internal bus characteristics, which depends only by the programmable interconnections. The ARM processor cores have standardized bus interface architecture called also AMBA bus. This specification is intended to implement the on-chip communication interface. The AMBA standard is an open standard on-chip specification. The internal bus is the standard interface that enables Intellectual Property (IP) re-use and facilitate the development of multi-processor designs. Table 2.1 defines the AMBA bus protocols. Since the SOC Zynq architecture use AXI bus protocol, here we just enumerate all the other AMBA bus protocols in order to get an idea about what their acronym means:

- The AMBA5 CHI (Coherent Hub Interface) specification defines the interfaces for the connection of fully coherent processors.
- AMBA4 ACE Coherency Extensions (ACE) extends AXI with additional signaling introducing system wide coherency. This system coherency allows multiple processors to share memory and enables technology like ARM’s big.LITTLE processing. The ACE-Lite protocol enables one-way aka IO coherency, for example a network interface that can read from the caches of a fully coherent ACE processor [KS13].





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- **AMB AXI Advanced** is part of the ARM Advanced Micro-controller Bus Architecture AXI3 and AXI4 specifications [ARMa] is a parallel high-performance, synchronous, high-frequency, multi-master, multi-slave communication interface, mainly designed for on-chip communication. AXI offers a wide spectrum of features, including:
  - separate address/control and data phases;
  - support for unaligned data accesses;
  - burst-based transfers, with a single transmission of the starting address;
  - separate and independent read and write channels;
  - support for outstanding transactions;
  - support for out-of-order transaction completion for transactions having different thread IDs on the same master port. Transactions on the same master port that have the same thread ID must be completed in order. Additionally, different master ports may be completed out of order with respect to each other.
  - support for atomic operations.
- **AHB Advanced High-performance Bus** - is a bus protocol introduced in Advanced Micro-controller Bus Architecture. A simple transaction on the AHB bus consists of an address phase and a subsequent data phase. Access to the target device is controlled through a non-three-state multiplexer, for this reason only one bus-master is allowed at a time. Allows development of embedded systems in FPGA.
- **APB Advanced Peripheral Bus** is a low cost interface optimized of low power consumption and reduced interface complexity. The APB interface is a simple, synchronous protocol. It is not a pipeline protocol.
- **ATB Advanced Trace Bus** – defines how a trace information transfers between components in a trace system. The ATB interface supports various features, including: stalling of data, using valid and ready responses, control signals that indicate the number of bytes valid in a cycle, identification of the originating component, by signaling an associated ID with each data packet, support for any trace protocol information, data information or data format requirements, identification of data from all originating components, flushing.

### **ARM Cortex-A9 Processor Micro-Architecture**

Based on the ARM documentation [ARMb] ARM Cortex A9 processor instructions has a speculative out-of-order speculative issue super-scalar execution 8-stage pipeline giving 2.50 DMIPS/MHz/core [ARMb] each core processing. The NEOM co-processor technology allows SIMD operation, and floating point unit accelerate the floating point operations with twice speed of the previously ARM FPU version. THUM2-2 instruction set increase code density without influencing the instruction execution speed





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

compared to the performance of ARM 32 bit instruction performance. The processor implements the ARMv7 Debug architecture that includes support for TrustZone and CoreSight. The Cortex-A9 processor implements Baseline CP14, Extended CP14 debug access, and memory mapped access to the debug registers. See [ARMB] for more detailed in formations.

### 2.3.2 Memory interfaces

The Xilinx Zynq SOC and 7 series FPGAs with a "Memory Interface Solution" core with its memory controller and physical layer core (PHY) optimize interfacing for user designs using the AMBA standard Advanced eXTensible Interface (AXI4) slave interfaces to DDR3 and DDR2 SDRAM devices and other. Using the Memory Interface Generator (MIG) IP one can easily interface external memories. One can setup the clock ratio, the power supply voltage for the respective FPGA IO banks, setup the memory type, memory part, data width and etc. Also the IP can create a custom memory controller part.

### 2.3.3 Processing System Interconnect

The processing system interconnect provide an interface between the PS (Processing System) and PL (Programmable Logic). The interconnect features are as follows: Enable/Disable I/O Peripherals (IOP), Enable/Disable AXI I/O ports, MIO Configuration, Extended Multiple Use I/Os (EMIO), Accelerator coherency port (ACP), Transaction checker (ATC), Interconnect logic for Vivado Design Suite IP – PS interface, PL Clocks and Interrupts, PS internal clocking, Generate PS configuration register.

### 2.3.4 Memory Map

The Cortex-A9 processor uses 32-bit addressing. PS peripherals and PL AXI slave interfaces are memory mapped to the Cortex-A9 processor cores. All AXI slave PL peripherals will be located between the addresses:

- 4000\_000 and 7FFF\_FFFF connected to AXI GP0 interface and
- 8000\_0000 and BFFF\_FFFF connected to AXI GP1 interface.

Note that the Zynq 7000 have different memory resources. First the primary memory is the On-chip memory (OCM), which is the RAM and Boot ROM. Second resource of memory is the off chip memory, which is connected via the MIG controller (DDR dynamic memory controller), which supports LPDDR2, DDR2, DDR3. Third resource is the Flash/static memory controller, which supports SRAM, QSPI, NAND/NOR FLASH.

### 2.3.5 PS Boots first

After Power-on-Reset (POR) the PS clock is enabled. Then, the PS begins executing the BootROM code in the on-chip ROM to boot the system. The POR resets the entire device with no previous state saved.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

The BootRom is the first software run in the application processor unit. The BootROM executes on CPU 0 and CPU 1 executes the wait-for-event (WFE) instruction. The boot device can also hold a bitstream to configure the PL and an embedded operating system, but these are not accessed by the BootROM code. The bitstream and the operating system are stored in the flash memory (external memory). In master boot mode the system boots from the flash device, while in JTAG boot mode, the BootROM code does minimal system configuration and enables a JTAG interface. After the BootROM executes, the First Stage Boot Loader (FSBL)/User code takes control of the PS and is able to further configure the device, including the PL. For further details and reading please see Xilinx User Guide UG821 and [Xil21].

### 2.3.6 Clock resources

The external clock source for the PS drives the PS clock subsystem, which are derived from one of three programmable PLLs: CPU, DDR and I/O. Each of these PLLs is loosely associated with the clocks in the CPU, DDR and peripheral subsystems (see figure 2.3). The PLLs generates the corresponding clock for the SPU, DDR and peripherals, also contains four clock generators for the PL. The clock generation paths include glitch-free multiplexers and glitch-free clock gates to support dynamic clock control. The 6 bit programmable dividers helps to generate the corresponding clock frequency together with the Clock Ratio Generator.

## 2.4 Summary

This chapter gave a summary about the Zynq processing system. It was presented the ARM processor in general and was summarized the AMBA bus standard and ARM Cortex A9 processor. There were presented the memory interface MIG, the processing system interconnect, memory map and the PS boot sequences.





„FÖNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

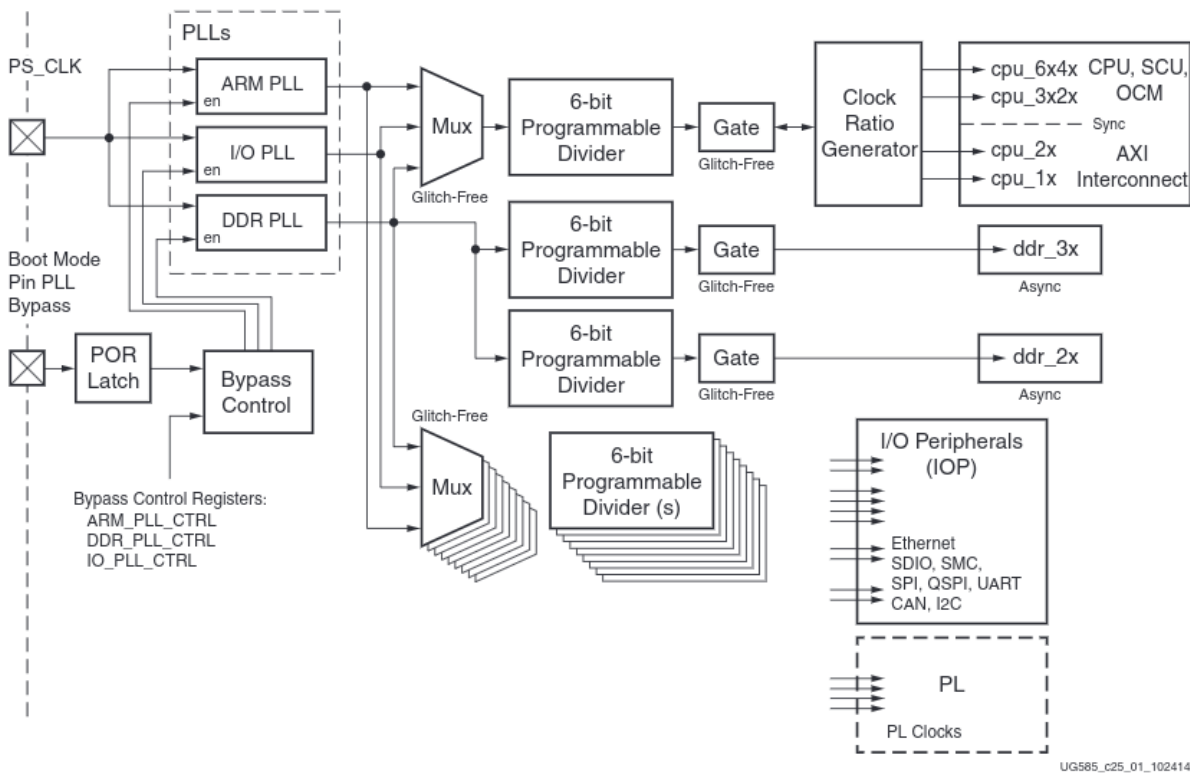


Figure 2.3: PS Clock subsystem block diagram





## Chapter 3

# VHDL hardware description language

### 3.1 Introduction to Hardware Description Language

Embedded system design is the process that begins with setting the requirements for the system and goes through several steps to the physical implementation of the system. With the development of integrated circuit technology, can be implemented extremely complex embedded systems. Dealing with complex designs has resulted in different trends in design methods:

- Behavioral method: It specifies the system in terms of its expected behavior. It is the closest to a natural language description of the circuit functionality, but also the most difficult to synthesize.
- Structural design style: The system is specified in terms of lower level components (in this case logic gates), functional blocks connected with internal signals. The translation of such a specification into a physical circuit is straightforward.
- - Data-flow method: This design style is similar to logical equations, although in general is not limited to logical values only. The specification is comprised of expressions made up of input signals and assigned to outputs. In most cases such an approach can be quite easily translated into structure and the implemented.

In the design process, simulations are complementary activities of the synthesis (see figure 3.1). In the case of application specific integrated circuits (ASIC) or embedded systems design; for example, simulations eliminate design errors and thus significantly reduce manufacturing costs (ex. production of integrated circuit masks or printed board circuit – PCB or Field Programmable Gate Array – FPGA – design). Extensive simulations can result in 82% of bug elimination and 9 hours saved by finding each defect, also increase productivity. shows the correspondence between design steps and simulation steps during the development process. The first step is to define the the system requirements. These requirements specify operating speed, delay times, connection points (interfaces), dissipated power, and other physical parameters.







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

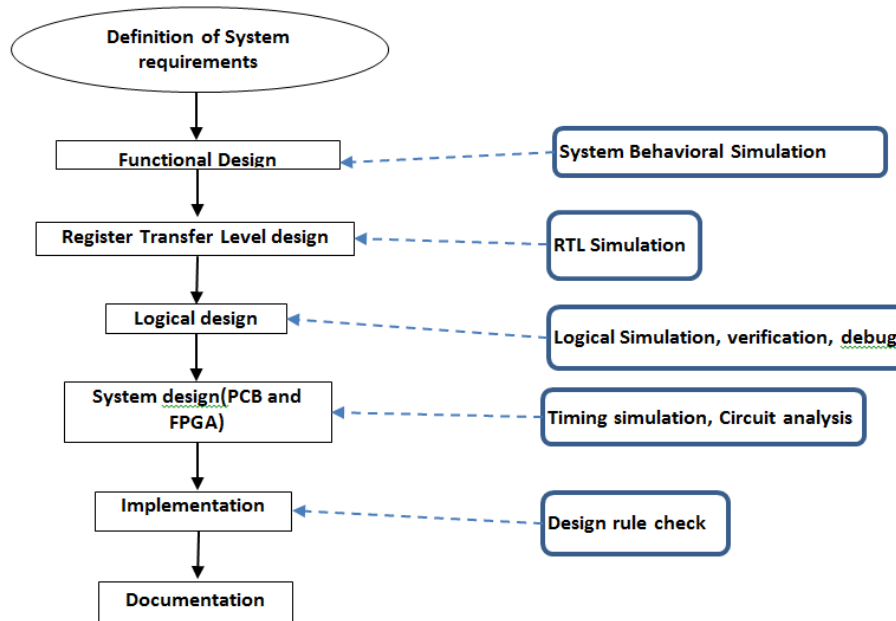


Figure 3.1: Design flow of digital systems

Based on the requirements mentioned above, the system is designed during the functional design and the functional requirements can be checked with the simulation used at this level. This design can be refined at the register transfer level. At this level, the tasks are taken over by registers, memories, arithmetic units, state machines. Logical design implements the elements defined at the register transfer level. The verification, debug simulation also models the design errors expected during production and the design errors induced by environmental influences. Finally, the physical implementation creates the digital circuit, which can even be a ready-to-manufacture integrated circuit or system.

### 3.2 Hardware Abstraction Levels

At each level of the design hierarchy, we describe the system design using circuit elements. Complex digital systems are described in different ways. However, these description methods are compatible with each other. Circuits are generally described in three domains: a behavioral domain, structural domain and a physical domain. These description methods are expressed in the Y diagram represented in figure 3.2.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

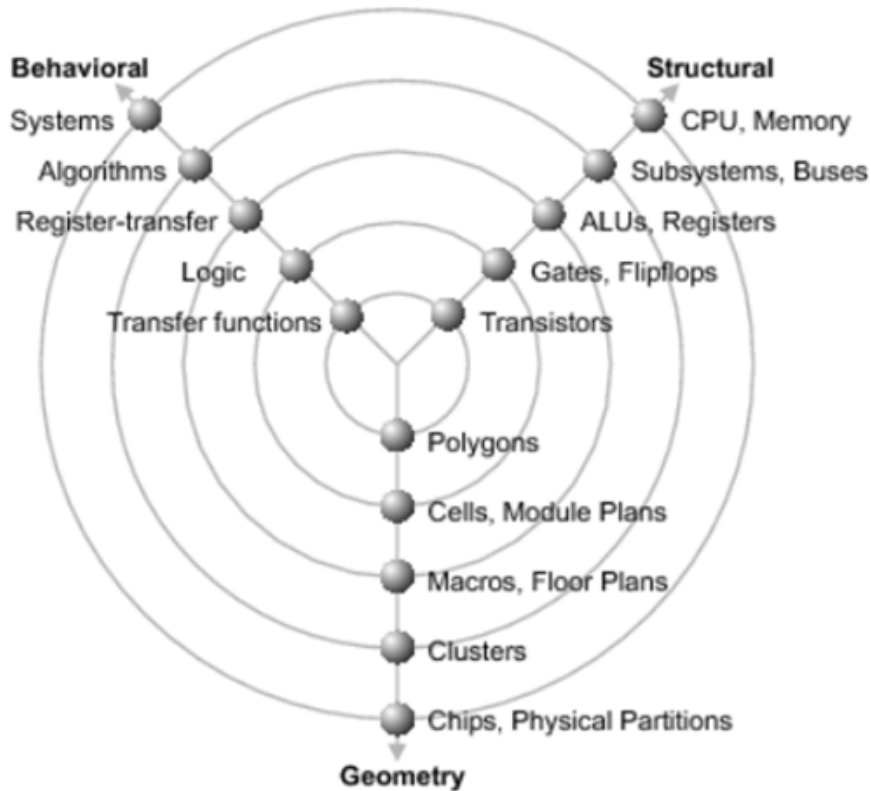


Figure 3.2: Y chart [GD99] p. 358

Gajski has developed the Y chart in VLSI (Very Large Scale Integrated Circuits) and Kuhn in the year 1983 to categorize the behavior of hardware designs based on the three different domains [[GK83]]. The three different domains are behavioral, structural and physical/geometry domain, which are on radial axis. Each of the corresponding domains can be further divided into levels of abstraction using concentric rings and each of the domains falling within the circle forms a group and keep going on in a top down fashion towards the center of the core.

### 3.3 Textual description of digital systems

Technology development and growing complexity of integrated circuits resulted in that conventional schematic design methods has become increasingly opaque and difficult-to-manage the design. Circuit technology has made it possible to manufacture so-called programmable logic circuits (as mentioned





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

before). However, designing with programmable logic devices required a design tool that allows the user easily describe the objective functionality of the design. First, this description method was elaborated in the description of the fuse map file resulting from the circuit technology. The fuse map defined the connections inside the circuit, similar to the net list used in schematic designs. This method was cumbersome and complex, so it became necessary to develop a general hardware modeling language capable of uniformly describing digital circuits at different levels of abstraction to design a variety of usable electronic systems using programmable circuits from different manufacturers and implemented with different technologies. The technology evolution resulted in different text based hardware description languages such as: PALASM, ABEL, VHDL, VERILOG, System C, etc. In our days the most known hardware languages are VHDL, System Verilog and C or C++ based hardware description languages (such as SystemC, Catalpult C and others). In the following are presented shortly the most known hardware description languages and the VHDL is presented in detail.

### 3.3.1 VHDL hardware description language

What means the double acronym VHDL? The meaning of the VHDL is: VHSIC-HDL – (Very High Speed Integrated Circuits Hardware Description Language). Originally the VHDL was developed by the USA DoD (DoD – Department of Defense). The development result was a hardware description language, which highly satisfied all expectation of system design requirements. The VHDL is a self-documenting, structured and understandable language. The source code in the meantime is also a kind of specification document. The most important part of HDL (hardware description language) is the parallelism and the concurrent process handling. In addition, HDL can handle complex and compact sequential circuit models. The standardization of VHDL was made in 1987 by the IEEE (Institute of Electrical and Electronics Engineers). The first official standard of VHDL was elaborated in 1993. The VHDL extension to analog and mixed signal systems modeling opened a new chapter in the electronic system design and modeling. VHDL-AMS (analogue mixed signal) is an extension of the VHDL. The extension is valid for systems model simulation, because the analog circuit synthesis is a very complex and many parameter problem, which is not solved yet.

### 3.3.2 Verilog hardware description language

The designers of Verilog HDL (Gateway Design Automation – 1984) intention was to create a C like hardware description language, since the C programming language is already spread in the micro processing development applications. Verilog is sensible to lowercase and uppercase letters. Similar to ANSI C/C++ has pre-processing, flow control instructions (if/else, for, while, case, etc.) and is compatible to precedence operators. Syntactical differences are in the variable typed declarations, process module separation and others. A design written in Verilog program has multiple hierarchical modules. The modules contain the design hierarchy. The connection between the modules give the ports (input, output, inout) A module can contain internal signal declarations (such as wire, reg, integer, etc.), sequential blocks, modules Verilog is a simultaneous and synchronous data-flow processing language.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

### 3.3.3 SystemC hardware description language

“SystemC is a set of C++ classes and macros which provide an event-driven simulation interface (see also discrete event simulation). These facilities enable a designer to simulate concurrent processes, each described using plain C++ syntax. SystemC processes can communicate in a simulated real-time environment, using signals of all the data types offered by C++, some additional ones offered by the SystemC library, as well as user defined. In certain respects, SystemC deliberately mimics the hardware description languages VHDL and Verilog, but is more aptly described as a system-level modeling language.” (Wikipedia) SystemC® is an extension of the ANSI C++ class library for hardware design. The purpose of SystemC to give a C++ based design service for hybrid systems, where both hardware and software elements can be found.

The language is described by the IEEE 1666-2005 standard. During a SystemC application one can use the possibility of C++, but with the constraints that are in the standard definitions.

### 3.4 VHDL basics

Let us analyze the structural and behavioral system description in more detail. A digital system is basically used to process signals. Signals can take binary values (0 or 1, X, Z, etc.). The elements of a digital system are components such as logic gates, flip-flops, counters, processors, and so on. The connections between the components are made by wires. The input signals are also converted by the components (by the logical functions) into output signals, but there are bidirectional or input-output signals also. A VHDL program from the point of view of its structure consists of design units. A feasible (synthesizable) VHDL program should contain two design units: one **entity** and one **architecture** unit. The **entity** is the interface to the block of hardware, while the **architecture** defines the system internal structure or the system behavior. An **entity** may have several alternative **architecture** (see figure 3.3).

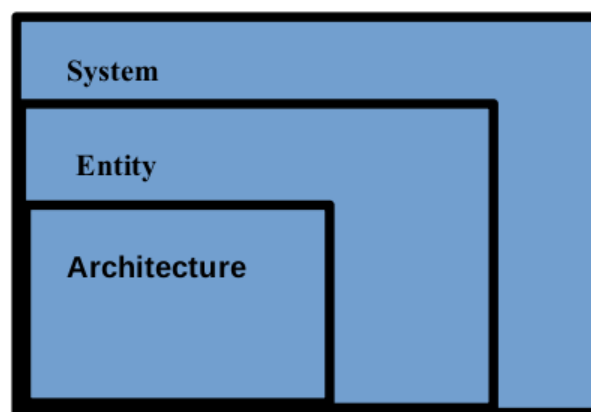


Figure 3.3: VHDL model



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

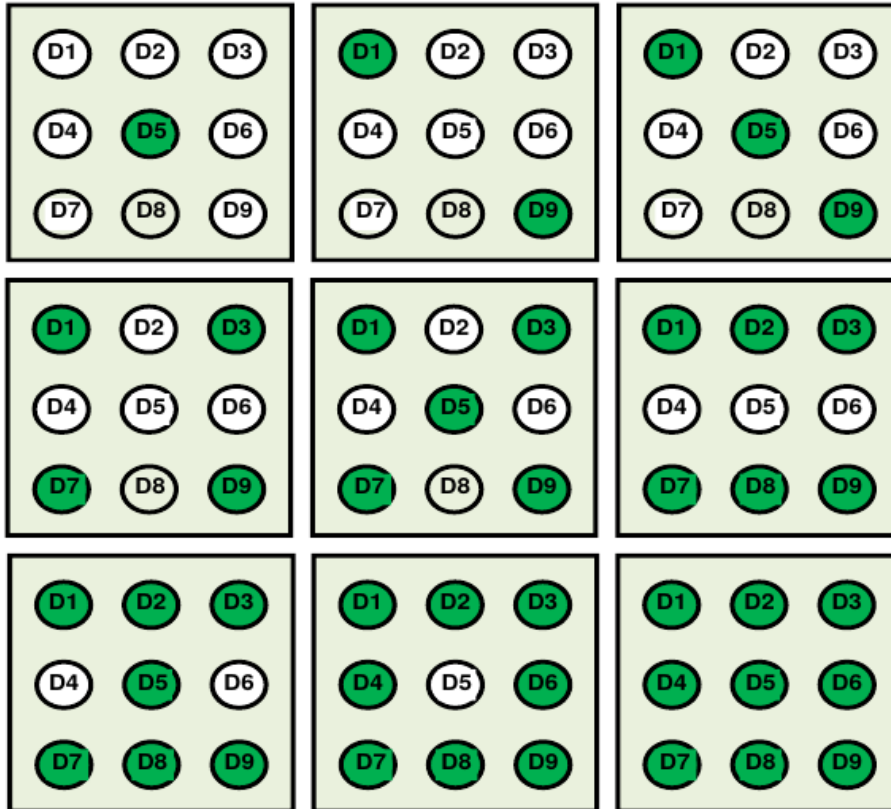


Figure 3.4: Domino code display

The structure of a VHDL program can best be illustrated through an example. Let us consider the example of the so called “domino code display” (see figure 3.4). The output of the domino code will be 9 LEDs organized as seen in figure 3.4. In the classical digital systems first one should write the truth table, then using Boolean algebra and Karnaugh maps, then minimize the logical equations and finally implement the circuit. The result of the Boolean minimization is given by:

$$D1 = D9 = D + C + B; D2 = D8 = D + C * B; D3 = D7 = D + C; D4 = D6 = D; D5 = A; \quad (3.1)$$

where the “+” is the logical **OR** and the “\*” symbolize the logical **AND** function, **D, C, B, A** are the input signals (variables) and **D9, D8, D7, D6, D5, D4, D3, D2, D1** are the output functions. See figure 3.4). The “domino coder” solution given in VHDL code is shown in figure 3.5.

Let us explain the domino VHDL code implementation presented above, and introduce the basic elements of a VHDL program (such as entity, architecture).





„FÖNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

```

19 ⊙ -----
20 :
21 :
22 : library IEEE;                -- open library IEEE
23 : use IEEE.STD_LOGIC_1164.ALL;  -- open standard logic package
24 :
25 ⊙ entity domino is           -- entity name: domono;
26 ⊙                               -- entity begins here;
27 ⊙     generic (nr_point_domino: integer := 10 );  -- generic is not used in this example
28 :     Port ( D : in STD_LOGIC;  -- input port definition D
29 :           C : in STD_LOGIC;  -- input C
30 :           B : in STD_LOGIC;  -- input B
31 :           A : in STD_LOGIC;  -- input A
32 :           domino : out STD_LOGIC_VECTOR (9 downto 1));  -- output port
33 :                               -- output port definition is vector of 9 bits
34 ⊙ end entity domino;         -- end entity domino
35 :
36 ⊙ architecture Behavioral of domino is  -- architecture description begins here
37 :
38 : begin
39 :     -- description of output functions
40 :     domino(1) <= D OR C OR B;  -- DOMINO led 1
41 :     domino(2) <= D OR (C AND B);  -- DOMINO led 2
42 :     domino(3) <= D OR C;  -- DOMINO led 3
43 :     domino(4) <= D;  -- DOMINO led 4
44 :     domino(5) <= A;  -- DOMINO led 5
45 :     domino(6) <= D;  -- DOMINO led 6
46 :     domino(7) <= D OR C;  -- DOMINO led 7
47 :     domino(8) <= D OR (C AND B);  -- DOMINO led 8
48 :     domino(9) <= D OR C OR B;  -- DOMINO led 9
49 :
50 : end Behavioral;  -- end architecture description
51 ⊙

```

Figure 3.5: Domino coder implemented in VHDL

### 3.4.1 Entity as the interface model

The **entity** defines the design name and the way that the described VHDL element how is connected to other elements, the circuit unit’s basic parameters, input and output pins. That is, with what static parameters (constants) and dynamic channels (signals) happens the information exchange between the **entity** and its environment. The **entity** allows defining any parameters that are passed into the model using hierarchy. The basic template for an entity is shown in figure 3.6.

Signals defined with the **port** section can be used to connect entities together. In the domino example 3.5 the entity contains four input signals (D, C, B, A) and a 9 bit bus output (D[9:0]). The schematic symbol is much more expressive (see figure 3.7). The **textbfport** declaration defines the type of connection, direction, and dimension. A **textbfport** direction can be *in* (only read), *out* (only write) and *inout* (read and write). Port type is *bit* (*std\_logic*) or *bit\_vector* (*std\_logic\_vector*). The type *bit\_vector* defines buses. If the model has parameters, then this are defined by using the keyword **generic**. Generic declaration is similar to that of constant. A **generic** definition should contain type (integer, time) and initial value defined after the symbol“:=”.

### 3.4.2 Architecture as model functionality, behavior, structure

There is no **architecture** without an interface. Its importance is so great that **architecture** is specified in VHDL as the architecture of entity (see domino implementation above figure 3.5). The general structure of the architecture is shown in 3.8. The first line of the architecture links the implementation to the



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```

1
2
3 entity <entity_name> is
4 generic (
5     <generic_name> : <type> := <value>;
6     <other generics>...
7 );
8 port (
9     <port_name> : <mode> <type>;
10    <other ports>...
11 );
12 end <entity_name>;
13
14

```

Figure 3.6: Entity template

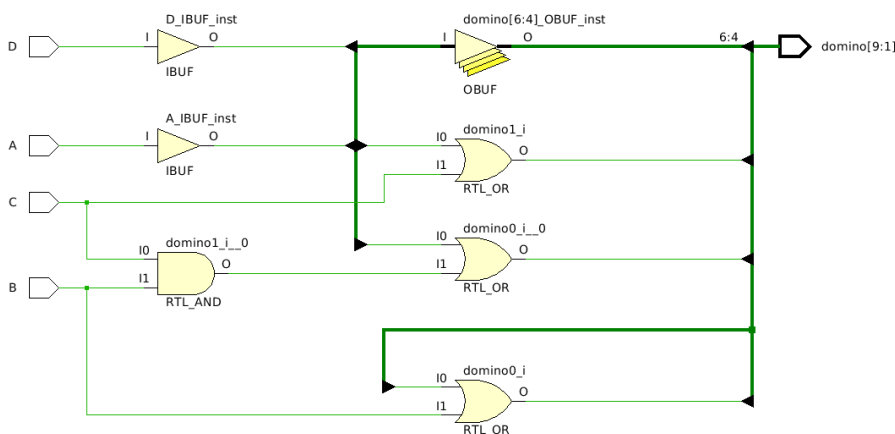


Figure 3.7: The domino coder schematic created from VHDL code

entity defined above (*architecture Behavioral of domino is*).

VHDL architecture can have a variety of structures to achieve different type of functionality (from simple combinatorial to complex sequential or structural description). Declarative items can be internal signals seen only in the architecture, or other components described by other VHDL models.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```
1 |  
2 |  
3 | architecture <arch_name> of <entity_name> is  
4 |   -- declarative_items (signal declarations, component declarations, etc.)  
5 | begin  
6 |   -- architecture body  
7 | end <arch_name>;  
8 |
```

Figure 3.8: Architecture structure

### 3.4.3 Constants and Signals

When a value needs to be static throughout a simulation, the type of element to use is a **constant**. A **constant** is used to initialize parameters, registers, or other. **Signals** are the “wires” inside the architecture connects processes or design elements (components) together. **Signals** have no direction they are read-and-write type. **Constants** and **signals** are declared in the architecture before the **begin** statement.

### 3.4.4 Packages, Libraries

Design units are the building blocks of the VHDL program. When compiling a VHDL program, the processing program distinguishes five separate program units. Each unit is analyzed separately. The five separate units are:

- entity,
- architecture,
- package definition,
- package body,
- configuration.

The entity and the architecture were presented previously. The *package definition* describes the elements implemented and usable in the package, while the *package description* includes equations describing the implementation of functions, components, VHDL program codes, and so on. One characteristic of VHDL is that multiple architectures can be assigned to an entity. The **configuration** specifies which architecture is assigned to the design entity under the conditions we defined. The VHDL **library** stores the design units that contain the most commonly used components and functions. To achieve rapid synthesis, the IEEE created various standardized VHDL **packages**, which were defined in standards 1164, 1076.3. (The actual standard is IEEE 1076-2008). Standard libraries are opened with the **library** keyword, while the library package you want to use is specified with the **use** keyword.

Opening a library and assign packages within a VHDL program is necessary because one can use in the given VHDL program different data types defined in the packages. For example: standard logic (std\_logic), standard bus system (std\_logic\_vector), which is part of the std\_logic\_1164 package.







„FÖNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```
library IEEE;           -- open library IEEE
use IEEE.STD_LOGIC_1164.ALL; -- open standard logic package
use IEEE.numeric_std.all; -- commonly used packages
use IEEE.std_logic_unsigned.all;
```

Figure 3.9: Open IEEE Library and use packages

### 3.4.5 VHDL grammar and lexical elements

Lexical elements are the syntax units that form the basis of the VHDL language. These include comments, identifiers, reserved words, numbers, characters, and strings. The **comment** is text after two minus signs (" - "). The comment is ignored when processing the program. The comments are mainly used to document the program. A well commented program documenting itself. **Identifier** is the name of a VHDL object. The following rules apply to identifiers:

- only Latin letters are allowed;
- decimal numbers and underscore can be used to name the identifier;
- the first character of the identifier is always a letter;
- the last character cannot be an underscore (" \_ ")

The last character cannot be an underscore (" \_ ") Correct examples for identifiers: A10, next\_state, NextFunction, etc. VHDL is not case sensitive; the following identifier names all refer to the same object: identifier, IdENTifier, IDENTIFIER. Therefore, the correct programming mode is to always refer to an object in the same way. Best practice for identifier names is to use expressive names as possible, but the use of reserved words is forbidden!

**Numbers, characters and strings:** In VHDL, the type of numbers can be integer, floating point, real. In the representation of numbers, the decimal number system (i.e. 23), the double precision system (i.e. 23 => 2 #10111#) or the hexadecimal system e.g. (23 => 16#17#) is used. For example, the number 123456 is the same as the number 123\_456, just as the binary number 2#10100101101# is the same as the number 2#1010\_0101\_1010#.

**Character** types are denoted by apostrophes, such as 'V', 'H', 'D', 'L'. Note that the 23 and '23' are different, the first is a number and the second is a character type. Character strings or strings are enclosed in double quotation marks, such as "VHDL". Note that the number 2 # 0101\_1010 # and the character lines "0101\_1010" also differ in terms of notation and content, what's more, the following two character lines are not the same: "0101\_1010", "01011010".

### 3.4.6 VHDL Objects

An object is a data type which contains a given value. Four object types are known:

- signal





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- variable
- constant
- file

The file type is not discussed in this chapter because it is not a synthesizable element.

A **Signal** must be defined at the beginning of the architecture (after “architecture”, before “begin”). The signals are only visible in the architecture which defined them. The **signal** definition is as follows:

```
signal <name> : <type> := <value>;  
signal a_signal_name: std_logic := '0';
```

The value of the signal is assigned by the following operator: <= Example:

```
a_signal_name <= d_in0 and din1;
```

**Variables** in the traditional sense play the same role like variables in programming languages. The use of variables is only allowed in **processes**. The variable can also be thought of as a “local symbolic storage element” to describe the abstract behavior of the system. The variable is defined as follows:

```
variable <name>: <type> := <value>; – variable declaration
```

The value of a variable is assigned by the following operator :=; Example:

```
variable name_of_var: std_logic_vector(2 downto 0) := "00"; – variable with init value "00"  
name_of_var := in1 and in2;
```

**Note:** Because the compiler does not assign any timing values to a variable, means that assigning a value to the variable has an immediate effect on it.

**Constant** The value of the constant cannot be changed in the VHDL code. The constant is defined as follows:

```
constant <name>: <type> := <value>;  
constant bus_dimension: integer := 32;
```

### 3.4.7 Data types and operations

In the VHDL language, we assign to each variable a data type (i.e. object or signal). The data type defines a set of values that variables can take and a set of operations that can be performed with the variables. Because in VHDL the values that can be assigned to signals and the operations that can be performed on them are strongly type-dependent, type conversion should be applied if other data type is assigned to a given object.

#### Standard VHDL data types

From the synthesis point of view, the most common data types are as follows:

- **integer** in VHDL the integer type has a 32 bit representation and can get values in the  $[-2^{31}, 2^{31} - 1]$ . VHDL defines two sub types such as normal type ( $N^*$ ) and positive (N) type;
- **Boole** type with *true*, *false* values;





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- **bit** type with '0', '1' values;
- **bit\_vector** type defined as a multi bit signal.

In the case of real digital design, a signal can be assigned with many signal levels not only with two-state signal levels, as a signal can be in a high-impedance state or even can have a signal level conflict in the case of wired logic. Therefore, in the VHDL, the **STD\_LOGIC** and **STD\_LOGIC\_VECTOR** types were introduced in the IEEE **std\_logic\_1164** package to solve this problem. This data types allows a more flexible signal handling.

### STD\_LOGIC, STD\_LOGIC\_VECTOR data types

The most commonly used two data types are contained in the **STD\_LOGIC\_1164** package. The **std\_logic** type is a sub type of the unsigned **std\_ulogic** contained in the **STD\_ULOGIC\_1164** package. For use of any data type, first the library and then the package must be opened:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

The **std\_logic** data type can have nine possible values: {'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'}. These values are explained as follows:

- '0' and '1' - denote the values “definite logic 0” and “definite logic 1”, respectively, which are taken by a signal driven by standard circuit;
- The high impedance value 'Z' can be taken by the signal if the drive circuit is a tri-state buffer;
- 'L' and 'H' - ‘weak logic 0’ and ‘weak logic 1’ can be set to a signal for wired logic circuits when the driver circuit provides a weak drive current;
- 'X' and 'W' - “indeterminate logic state” or “weak indeterminate logic state” value taken when the driver signal value cannot be interpreted as a logic 0 or a logic 1. This state occurs when two output drivers control the signal to a logic state of conflicting values (logic 0 and 1). This is happen especially in circuit simulation conditions indicate faulty circuit operation, especially in circuit simulation.
- 'U' - Status used in simulation. Indicates that a value has not yet been assigned to the signal or variable.
- '-' indicates that the signal is redundant.

## 3.5 Behavioral system description in VHDL

### 3.5.1 Combinational Signal Assignments

There is a strong relationship between the signal assignments and VHDL, as result there can be achieved very effective implementations. In the VHDL standard there are three type of simultaneous/parallel signal assignments (see example in 3.10):

- Simple signal assignment, which correspond to the Boolean algebraic equations;





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- Conditional signal assignment;
- Selected signal assignment.

Digital circuits are combinational and sequential type. Combinational circuits do not have internal storage elements (memory) and internal intermediate states. The output function of the combinational system has always the same value for a given variation of the input signals. In reality, short transients may appear due to the gate delay of the circuits. However, when the input variables are steady state, the output always take the same value. From the point of view of implementation (with programmable logic circuits) the combinational circuit is without memory or internal storage (latch or flip-flop) or without feedback.

– **simple signal assignment with Boolean functions:**

The signal assignment `""<=` means that the input signal value is assigned immediately to the output (see figure 3.10).

– **Conditional signal assignment**

The outputs get the input values based on the *when ... else* condition (see figure 3.10).

– **Selected signal assignment**

This type of signal assignment is similar to the truth table. The selector usually is a *std\_logic\_vector*, when the value what gets the selector is true then the given values is assigned to the output signal. The closing operator *when others* assign an output values for all other input signal variations, which are not enumerated (see figure 3.10).

### 3.5.2 Sequential signal assignments

The outputs of sequential circuits are a function of the state of the input variables and the actual value of the internal variables (state register output values). The description of sequential system can be done by concurrent assignments, but this is not the common method. The *process* is the sequential description mode used for description of sequential systems, state machines, etc. The process is the mechanism by which sequential statements can be executed in the correct sequence, and with more then on process, concurrently. A process consist of a sensitivity list, declarations and statements. The process is executed simultaneously with the parallel assignments, however, the program lines in the process take place sequentially after a signal from the sensitivity list activate the process (change of signal). In the process execution an important role plays the signal identities, conditional signal assignments ("if then else", "case"), and cycles. It is important to clarify the difference between sequential signal assignment and a sequential logic circuit. The first one describes the process internal state while the last one is the digital circuit with internal states. Note that a process can implement not only sequential circuit but also combinational one.

The signals from the sensitivity list (see figure 3.11; "all\_input\_signals,separated\_by\_commas,"clk" and "rst") activate the process. If at least one signal change its state the process is executed.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

```

34 entity sig_assign is
35     Port ( a : in STD_LOGIC;
36           b : in STD_LOGIC;
37           c : in STD_LOGIC;
38           d : in STD_LOGIC;
39           s : in STD_LOGIC_VECTOR (3 downto 0);
40           f0 : out STD_LOGIC;
41           f1 : out STD_LOGIC;
42           f2 : out STD_LOGIC_VECTOR(6 downto 0));
43 end sig_assign;
44 architecture Behavioral of sig_assign is
45     -- signal for input data bus definition
46     signal d_in: std_logic_vector (3 downto 0);
47     begin
48         -- data bus from input signals
49         -- d_in <= d & c & b & a;
50         -- simple signal assignment with Boolean functions:
51         f0 <= a and (b or c) and d; -- simple assignment with "<="
52         -----
53         -- conditional signal assignment
54         f1 <= a when s = "0000" else -- if s = 0 then f1 <= a else
55             b when s = "0001" else -- if s = 1 then f1 <= b else
56             c when s = "0010" else -- if s = 2 then f1 <= c else
57             d ; -- if s = any other value then f1 <= d
58         -----
59         -- selected signal assignment
60         -- the example is a seven segment decoder example
61         with s SElect
62         f2 <= "1111001" when "0001", --1
63             "0100100" when "0010", --2
64             "0110000" when "0011", --3
65             "0011001" when "0100", --4
66             "0010010" when "0101", --5
67             "0000010" when "0110", --6
68             "1111000" when "0111", --7
69             "0000000" when "1000", --8
70             "0010000" when "1001", --9
71             "0001000" when "1010", --A
72             "0000011" when "1011", --b
73             "1000110" when "1100", --C
74             "0100001" when "1101", --d
75             "0000110" when "1110", --E
76             "0001110" when "1111", --F
77             "1000000" when others; --0
78     end Behavioral;
79
    
```

Figure 3.10: Combinational Signal Assignment Methods

## Variables

Definition of internal **variables**, used by the process, which are local signals, objects can be seen only inside the process. As the process is activated the sequential signal assignments are executed one-by-one and the process transits to the next internal state. A process as part of a digital systems can be in active or suspended state. Inside the process, signal assignment is done line by line execution of the process body. Signal use inside the the process is done taking into account three important aspects:



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```

34 entity example_proc is
35     generic (dim: integer := 16);
36     Port ( all_input_signals: in STD_LOGIC_VECTOR (dim-1 downto 0);
37           separated_by_commas: in STD_LOGIC;
38           clk : in STD_LOGIC; -- clock
39           rst : in STD_LOGIC); -- reset
40 end example_proc;
41
42 architecture Behavioral of example_proc is
43
44     begin
45         -- combinational process example
46     process (all_input_signals,separated_by_commas)
47     begin
48         -- <statements>;
49     end process;
50     -- sequential process with negative edge clocked
51     -- and synrconous High reset
52     process (clk)
53     begin
54         if (clk'event and clk = '0') then
55             -- if there was a change in the clock signal and its value = 0
56             -- this define the negative edge
57             if rst = '1' then
58                 -- if reset is 1 then execute the reset statments
59                 -- <statements>;
60             else
61                 -- otherwise
62                 -- <statements>;
63             end if;
64         end if;
65     end process;
66     -- sequential process with positive edge clocked
67     -- and synrconous Low reset
68     process (clk)
69     begin
70         if (clk'event and clk = '1') then
71             if rst = '0' then
72                 -- <statements>;
73             else
74                 -- <statements>;
75             end if;
76         end if;
77     end process;
78 end Behavioral;

```

Figure 3.11: Example of different process types

- signal definition in process is forbidden;
- signals take the new value assigned in the process became valid at the process termination. Signals preserve their previous values until the process termination;
- in the case of multiple assignments for the same signal the last value is preserved at process termination.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

```

34 entity var_example is
35     Port ( a : in STD_LOGIC; -- input signals a, b, c
36           b : in STD_LOGIC;
37           c : in STD_LOGIC;
38           f_out : out STD_LOGIC); -- output function f_out
39 end var_example;
40
41 architecture Behavioral of var_example is
42
43 begin
44     process (A, B, C)
45         -- variable declarations var1, var2, var3
46         variable var1, var2, var3: std_logic;
47         begin
48             var1 := c; -- var1 = c see schetic figure
49             var2 := var1 and a; -- on schematic f_out2_i
50             var3 := var2 nand b; -- on schematic f_out3_i
51             f_out <= var2 or var3 or c; -- on schematic f_out1_i and f_out0_i
52         end process;
53 end Behavioral;

```

Figure 3.12: Variable assignment example

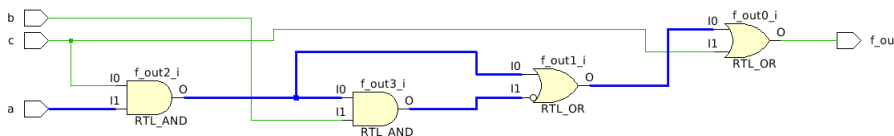


Figure 3.13: Schematic resulted from variable assignment example

Since signals preserve the last value assigned to them, for this reason they are not used to store temporal values. On the other hand value assignment is made at process end or when the process is suspended. To solve this problem the term of **variable** is introduced. **Variables** are defined inside a process can store temporal value, and they take the values assigned to them immediately after assignment. **Variable** value assignment is made with ":= " operator. The following example demonstrate variable usage (figure 3.12).

Figure 3.13 represents the schematic obtained by the example given in figure 3.12. The blue wires represents the variables. *Var1* assignment is useless in this form, since the and gate *f\_out3\_i* input is directly connected to signal *c*.

### 3.5.3 Conditional Decisions in processes

The following decisions can be used only in the process body: **IF\_THEN\_ELSE**, **CASE**, **for**. **IF\_THEN\_ELSE** general structure is represented in figure 3.14. All the possible variants of the **IF\_THEN\_ELSE** decisions are given in the example (figure 3.15). The figure illustrate by examples the decision possibilities from the basic syntax to the most complete syntax of the conditional assignment.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

```

1
2  if <condition> then
3      <statement>
4  elsif <condition> then
5      <statement>
6  else
7      <statement>
8  end if;
9
10
    
```

Figure 3.14: IF\_THEN\_ELSE structure

```

41  begin
42  process(a, b, c, d)
43  begin
44      -- simple IF THEN decision exmple
45  if (a=b) then
46      f1 <= a;
47  end if;
48      -- IF THEN ELSE example
49  if (a=b) then
50      f2 <= a;
51      -- statments;
52  else
53      f2 <= b;
54      -- more statments;
55  end if;
56      -- multiple if condition using genreal form example
57  if a=b then
58      f3 <= a;
59      -- statments;
60  elsif (a>b) then
61      f3 <= b;
62      -- more elsif conditions and statments;
63  else
64      f3 <= c;
65      -- other statments eventually;
66  end if;
67  end process;
68  end Behavioral;
    
```

Figure 3.15: IF\_THEN\_ELSE examples

### CASE

As demonstrated above with the if statement is simple to define multiple conditions but for complex







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

branching the solution is the case structure without having to use Boolean conditions in every case. This structure is useful for state machine description or any other special state transitions. An example of the **CASE** structure is presented in figure 3.16, where four possible choices are presented.

**Note** that for a complete description **CASE** structure the termination "**others**" should be included (as shown in figure 3.16)! Figure 3.17 is a case example where the selector line sel is a two bits vector and

```
1 |
2 | case (<2-bit select>) is
3 |     when "00" =>
4 |         <statement>;
5 |     when "01" =>
6 |         <statement>;
7 |     when "10" =>
8 |         <statement>;
9 |     when "11" =>
10 |         <statement>;
11 |     when others =>
12 |         <statement>;
13 | end case;
14 |
15 |
```

Figure 3.16: CASE structure

depending on sel value the output f1 is assigned with the values of input signals a, b, c and d respectively.

### 3.6 Structural description in VHDL

Structural description for a digital system means describing what components the system consists of and how these components are connected to each other. The structural description allows the application of multi-level (hierarchical) design in the VHDL program using the **component**. Components can be used in the subsystems of a system (in other words in the main VHDL program) that implement even simpler functions. At the lowest level, the VHDL program describes the component in terms of its behavior. The components are connected with signals in the VHDL program. As represented in figure 3.18 the



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```
70 ⊕ process (sel, a, b,c, d)
71 ⊕ -- process demonstrating case with 2 input select
72 ⊕ -- sel is a 2 bit std_logic_vector
73 ⊕ begin
74 ⊕ case (sel) is
75 ⊕   when "00" =>
76 ⊕     f1 <= a;
77 ⊕     -- <statement>;
78 ⊕   when "01" =>
79 ⊕     f1 <= b;
80 ⊕     --<statement>;
81 ⊕   when "10" =>
82 ⊕     f1 <= c;
83 ⊕     -- <statement>;
84 ⊕   when "11" =>
85 ⊕     f1 <= d;
86 ⊕     -- <statement>;
87 ⊕   when others =>
88 ⊕     f1 <= a and b and c and d;
89 ⊕     --<statement>;
90 ⊕ end case;
91 ⊕ end process;
```

Figure 3.17: CASE structure example

system is built from components, which are connected with signals (arrows at both end), while the input and output ports represented by unidirectional arrows. In order to connect a component first should be declared (figure 3.19) then instantiated (figure 3.20). A **component** needs two VHDL elements for the structural description. First the **component** is described in as a stand-alone entity - a VHDL program - that has a stand-alone entity and architecture. Secondly the **component** should be declared then instantiated. **Component** declaration makes the connection to the original VHDL module with the definition of its entity (generics and ports) while **Component** instantiating defines the relationship of the **component** ports and other signal from the architecture where the component was invoked. Steps to insert the part:

1. Component behavior design in VHDL;
2. Component declaration in the structural VHDL design;
3. Component instantiating.

### Structural description example

On the following we will present the structural design style by designing a three bit shift register. The example starts from the the lowest level presenting a D type flip-flop design, followed by D type component declaration in the shift register VHDL program. Since the VHDL program is well commented the figures are self self-explanatory. First the rising edge controlled D flip-flop is presented (figure 3.21)





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

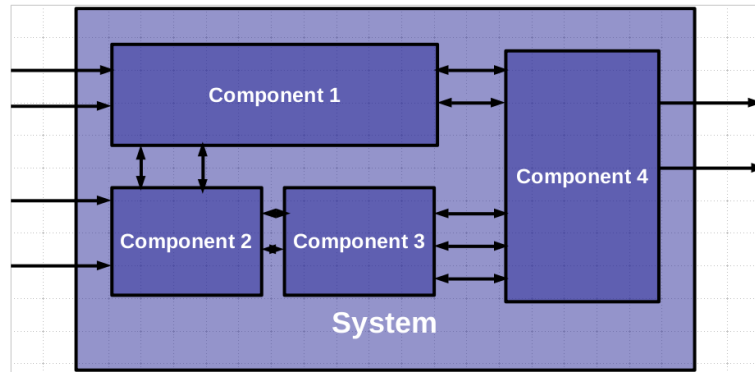


Figure 3.18: Structural system

```

1 |
2 |
3 | component <component_name>
4 | generic (
5 |   <generic_name> : <type> := <value>;
6 |   <other generics>...
7 | );
8 | port (
9 |   <port_name> : <mode> <type>;
10 |  <other ports>...
11 | );
12 | end component;
13 |
14 |
  
```

Figure 3.19: Component declaration





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```
1 |  
2 |  
3 | <instance_name> : <component_name>  
4 | generic map (  
5 |     <generic_name> => <value>,  
6 |     <other generics>...  
7 | )  
8 | port map (  
9 |     <port_name> => <signal_name>,  
10 |     <other ports>...  
11 | );  
12 |  
13 |
```

Figure 3.20: Component instantiating

then the hierarchical design of the shift register (figure 3.22) and finally the Schematic obtained from the VHDL code is presented.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```
19 ⊖ -----  
20 ⊖ library IEEE;  
21 ⊖ use IEEE.STD_LOGIC_1164.ALL;  
22 ⊖ -- fdd.vhd - is the file name  
23 ⊖ entity fdd is  
24 ⊖     port ( d      : in STD_LOGIC;      -- D flip-flop input  
25 ⊖           clk    : in STD_LOGIC;      -- CLK control clock  
26 ⊖           q      : out STD_LOGIC);    -- Q flip-flop output on rising edge clock  
27 ⊖ end fdd;  
28 ⊖  
29 ⊖ architecture Behavioral of fdd is  
30 ⊖     -- rising edge flip-flop example  
31 ⊖     begin  
32 ⊖         fddflipflop:  
33 ⊖         process (clk,d)  
34 ⊖             -- rising edge D flip-flop process  
35 ⊖             -- activating signals clk-clock and d-flip-flop input  
36 ⊖             -- if rising clock edge exist then =>  
37 ⊖             -- d input is written out to q;  
38 ⊖             begin  
39 ⊖                 if (clk'event and clk = '1') then  
40 ⊖                     -- other possibility for the rising edge description:  
41 ⊖                     -- if (rising_edge(clk))  
42 ⊖                         q <= d after 5 ns;  
43 ⊖                     --after 5 ns;  
44 ⊖                 end if;  
45 ⊖             end process; -- end process fdd flip-flop  
46 ⊖ end Behavioral;  
47 ⊖ -----
```

Figure 3.21: D flip-flop VHDL example





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 entity shift_reg_ex is
25     Port ( d_in : in STD_LOGIC; -- serial data input
26           clk  : in STD_LOGIC; -- clock rising edge is controlling the shift regiszter
27           rst  : in STD_LOGIC; -- reset when rst='1' then the output are q_out = '0'
28           q_out : out STD_LOGIC_VECTOR (2 downto 0)); -- register output
29 end shift_reg_ex;
30 architecture Behavioral of shift_reg_ex is
31     -- component declaration of D flip-flop
32     component fdd
33         port ( d      : in STD_LOGIC;      -- D flip-flop input
34               rst    : in STD_LOGIC;      -- RESET when rst='1' then q='0'
35               clk    : in STD_LOGIC;      -- CLK control clock
36               q      : out STD_LOGIC);    -- Q flip-flop output on rising edge clock
37     end component;
38     signal fqd : STD_LOGIC_VECTOR (2 downto 0);
39     begin
40         -- component instantiating bit 0, bit 1, bit2:
41         fd0 : fdd
42             port map ( d => d_in,
43                       clk => clk,
44                       rst => rst,
45                       q  => fqd(0) );
46         fd1 : fdd
47             port map ( d => fqd(0),
48                       clk => clk,
49                       rst => rst,
50                       q  => fqd(1));
51         fd2 : fdd
52             port map ( d => fqd(1),
53                       clk => clk,
54                       rst => rst,
55                       q  => fqd(2) );
56         q_out <= fqd;
57     end Behavioral;
```

Figure 3.22: Shift register example





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

## Chapter 4

# Hardware and software setup

hardware setup for zybo from digilent boardifels how to create a project on the vivado 2019.2  
preparation of the software and hardwer,

- SSH setup
  - IP cam setup
  - Vivado and ZedBoard/zybo setup
  - sbRIO - ZedBoard setup
- this • Extra components setup (DC motor, motor driver, hall effect sensor, GPS receiver)

### 4.1 Creating an SSH tunnel

Install PuTTY: <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

1. Start PuTTY as administrator.
2. In the Host Name. field, enter: mzsola.iit.uni-miskolc.hu





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

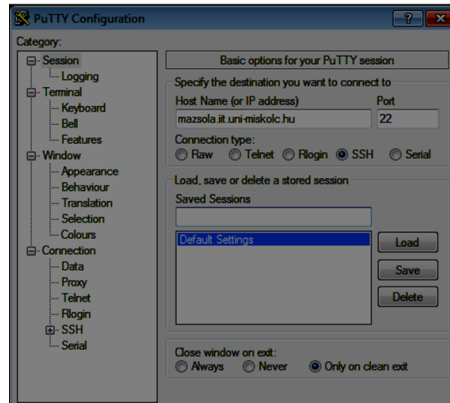


Figure 4.1

3. Open the Connection tab, in the Seconds between keepalives field, enter: 120

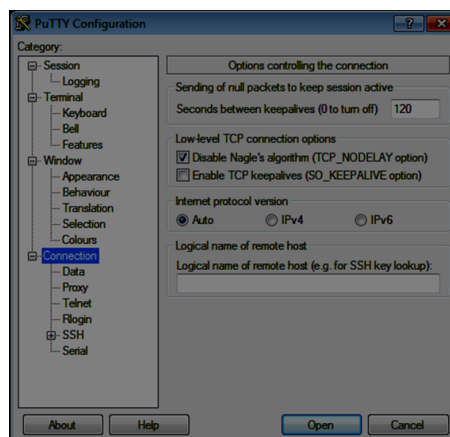


Figure 4.2

4. If you do not want to enter a user name every time you log in, on the Data tab, type in the Auto-login username field







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

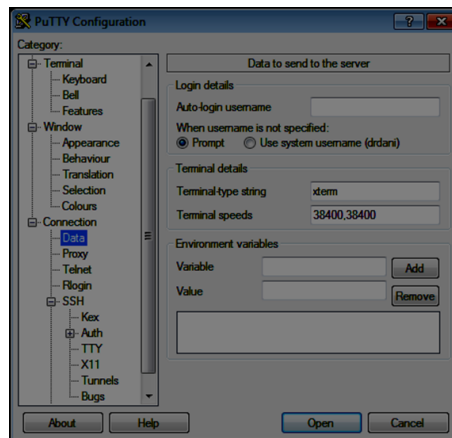


Figure 4.3

5. Open the Tunnels tab, then fill in the Source port and Destination fields as follows :  
Source port: 3331  
Destination: 192.168.201.21:3389
6. Press the Add button:
7. Go back to the Sessions tab and enter a name for the settings ( Save d Sessions field) and press Save :

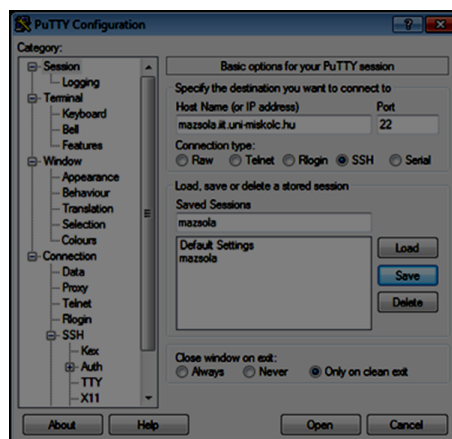


Figure 4.4

8. Then start the session by double-clicking on its name or selecting-Load-Open. To log in, enter the user name (if not saved in the session) and password. Keep the window open while entering



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

the internal machine.

9. Press “Windows” + “I” to open settings and click on “Update & Security”.

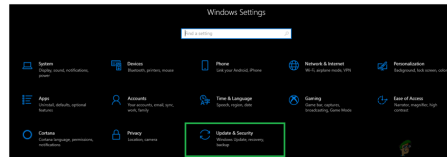


Figure 4.5

10. Select the “Windows Security” tab from the left pane and click on the “Firewall and Network Security” option.



Figure 4.6

11. Select the “Advanced Settings” button from the list.
12. A new window will open up, Click on the “Inbound Rules” option, and select “New Rule”.



**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

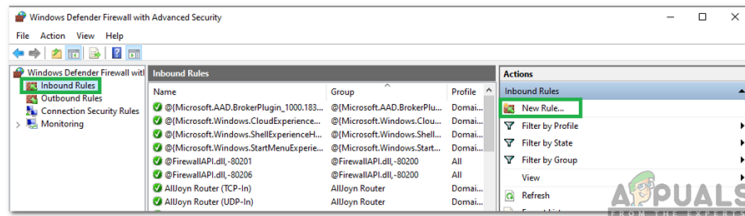


Figure 4.7

13. Select “Port” and click on “Next”.

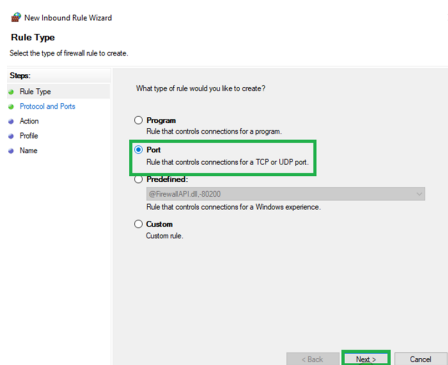


Figure 4.8

14. Click on “TCP” and select the “Specified Local Ports” option.

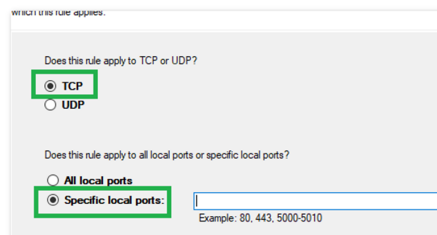


Figure 4.9

15. Enter in “3389” into the port number.

16. Click on “Next” and select “Allow the Connection”.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

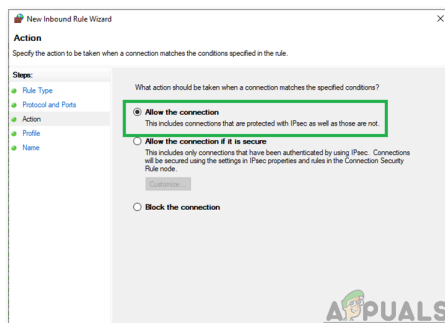


Figure 4.10

17. Select “Next” and make sure all three options are checked.

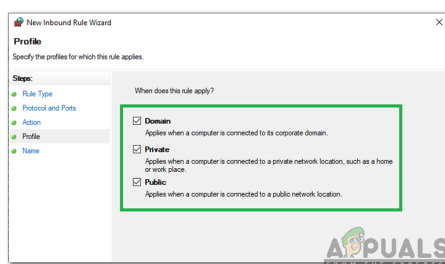


Figure 4.11

18. Again, click on “Next” and write a “Name” for the new rule.

19. Select “Next” after writing a name and click on “Finish”.

20. Similarly, go back to the 4th step that we have listed and select “Outbound Rules” this time and repeat the whole process to create an Outbound Rule for this process as well.

21. After creating both an inbound and an Outbound rule, check to see if the issue persists.

#### 4.1.1 Password change

1. Log in to mazzola.iit.uni-miskolc.hu as described before (e.g. by creating an ssh tunnel) or with a simple ssh connection.
2. To log in, use the username and password you received.
3. After logging in, issue the ppasswd command. Then enter the current password first and then the new password twice.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

## 4.1.2 Login to the internal machine

### On Linux

1. Install a Remote Desktop Protocol (RDP Client) program, e.g. Remmina, VirtualBox package also in-cludes one or the other.
2. Start and connect with RDP to localhost:3331 OR 127.0.0.1:3331

### On Windows

1. Find the Remote Desktop program and launch it.
2. Connect to localhost:3331 OR 127.0.0.1:3331

### Login to the internal machine

Windows often displays "others are logged in, continue?" message even if no one is logged on to the machine. Press TAB to switch to "Yes" and then enter.

To log in, use the username and the password provided by your instructor.

## 4.2 Installing Xilinx Vitis

This section gives the tutorial to install the Xilinx software. We recommend to use the latest Vitis software. The tutorials and the labs presented in this book were developed on Vitis 2019.2.

It is recommended for the online lab to install the Vitis 2020.2 software version in order to be compatible with the hardware server.

Vitis is the new name for the earlier *Software Development Kit (SDK)*, in addition integrates some other earlier tools such as SDAccel. Now, the whole suite is also called Vitis, which includes Vivado also. If Vitis is installed, then Vivado also gets installed. Vitis is used for the software development, while Vivado is used for hardware development. Bellow are the steps to be followed to install the software:

- You need a Xilinx account. If you don't, please create one.
- Xilinx unified web installer is downloadable at (login required):
- <https://www.xilinx.com/support/download.html>
- Download the Xilinx Unified Installer Windows/Linux Self Extracting Web Installer corresponding to your operating system.
- Run the downloaded file with system administrator privileges.
- Click *Next*, and enter your Xilinx account details. In the selection below of the installer choose 'Download and Install Now' then click *Next*.
- Select Vitis then click *Next*.
- Do the selection as given in the screenshot below (See 4.12).





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- Agree all the license agreements. Click *Next* and Select a suitable directory.

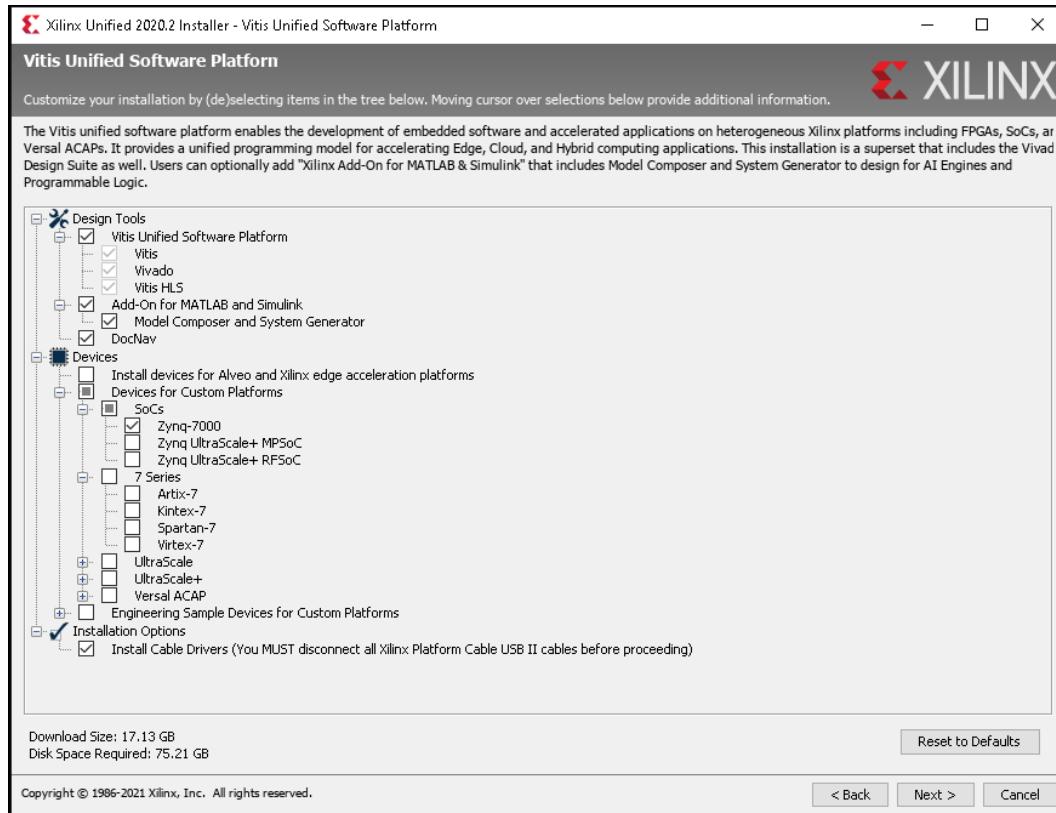


Figure 4.12: Vitis Unified Software Platform Install - Device selection screenshot

- Click *Next* once more and Install. If you are prompted to allow installation of some hardware, firewall warnings (Windows) about opening certain ports, simply click allow without changing the default options. This may also happen during the first run of certain tools after installation. When the installation has completed, click *Finish* to close the wizard.
- For the Vivado License Manager window, just cancel it, as the built-in Webpack license will suffice.

## 4.3 UART-USB driver install

### Windows

Installing UART-USB driver may be unnecessary, as Windows 10 should be able to get the driver from the internet. If you have issues, please follow the instructions given in the "Setup Guide" of the Zedboard:



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

[http://www.zedboard.org/sites/default/files/documentations/CY7C64225\\_Setup\\_Guide\\_1\\_3.pdf](http://www.zedboard.org/sites/default/files/documentations/CY7C64225_Setup_Guide_1_3.pdf). Download the driver (need an account to download from Cypress) and run the **setup.exe** after extracting the zip file.

## Linux

If you want to be able to access local hardware, do the following. If you are using FPGA remotely, this is not needed. Cable drivers are not installed by the installation wizard. Please install it from the script you can find at <install\_directory>/Xilinx/Vitis/2019.2/data/xicom/cable\_drivers/lin64/install\_script/install\_drivers (system administrator privileges are required).

## Installing Digilent Board Support Files

Zybo and Zybo Z7 are provided by Digilent. To use Zybo in the lab one need to install the board support files provided by Digilent. These files make it easy to select the correct part when creating a new project and allow for automated configuration of several complicated components used in many designs. For the Digilent board support files please refer to:

**<https://reference.digilentinc.com/vivado/installing-vivado/start>**

To work with **PMOD** the Digilent PMOD Library repository should be installed. To download and install this please refer to:

**<https://reference.digilentinc.com/learn/programmable-logic/tutorials/pmod-ips/start>**





## Chapter 5

# Embedded system Design with Zynq, hardware and software design basics

This chapter guide the student thru some basic laboratory examples, which give the necessary knowledge for further hardware-software development. In the chapter are used some of the workshop materials from Xilinx. We recommend to consult also the Xilinx documents and videos. For this open the **Docnav**, which was installed together with the software installation. These documents will familiarize you with the Xilinx Vivado Design Flow, which can be seen in figure 5.1. The Chapter is organized in three lab projects, which are as follows:

1. Lab 1 which is divided in two parts:
  - (a) Lab 1 Part 1 – basic processing system design;
  - (b) Lab 1 Part 2 – Processing system design with MIO.
2. Lab 2 Design with Xilinx IP library;
3. Lab 3 Create a user specific IP

The chapter concentrate on the Embedded Processor Design hardware and software development. We follow the methodology as it is in the workshop Xilinx material (Introduction, Objectives and Procedure). First an ARM Cortex A9 based processor system is designed and tested with the "Hello world" software application (see figure 5.1.2) The design includes a processor, external DDR and UART. Then LED blinking hardware test is demonstrated in two different ways. First using Processor system (PS) and MIO connected LED to create a software controlled project. This project allow the reader to explore the Zynq PS parametrizable resources, in the example using MIO for LED blinking project. Secondly the Programmable Logic will be used and adding to the PS system a PL interface using Xilinx IP library. In this case a general purpose port (GPIO) will be added to drive the board LEDs. The third lab use VHDL code to create a user specific peripheral, this time the board switches are connected to the AXI bus as a user created IP. The VHDL program will be integrated in the Block Design.







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

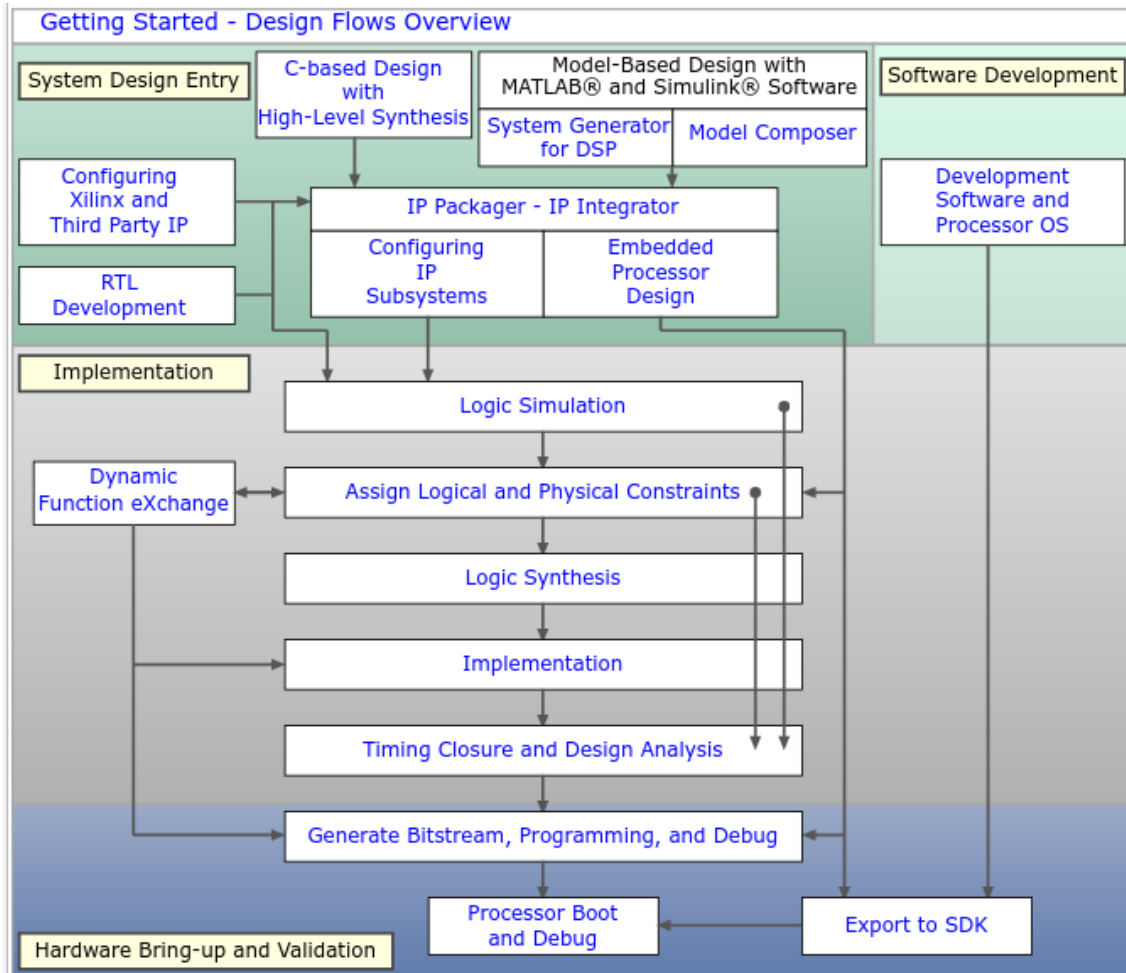


Figure 5.1: Xilinx UltraFast Design Methodology - System-Level Design Flow

## 5.1 Lab 1

### 5.1.1 Lab 1 Part 1: Hello World

#### Introduction

In this part of the lab we create a simple ARM Cortex A9 based processor design targeting the Zybo/Zed-Board board. Where the instructions refer to both boards, choose the board you are using. First the hardware system is created using Vivado, then using Vitis using an example application the hardware functionality is verified. **REDESSINER**





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

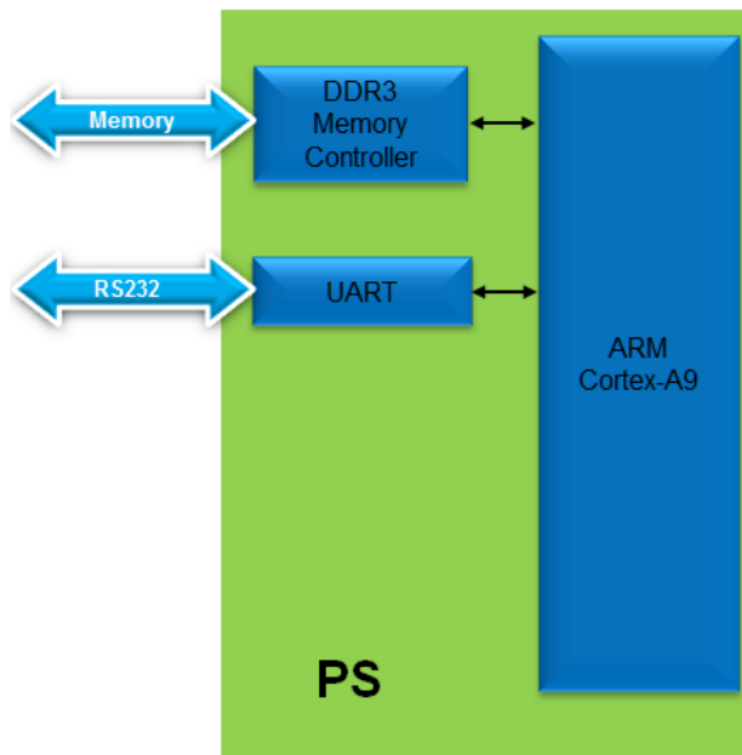


Figure 5.1.2: Processor design of Lab 1 [Xil18a] page 38.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

## Objectives

- Create a Vivado project for a Zynq system;
- Use IP Integrator to create a hardware system;
- Use Vitis to create a standard "Hello World" test project;
- Run the test application on the board.

## Procedure

In the first lab the design steps will be presented in detail to provide adequate instructions. Detailed instructions are given which must be followed in order to complete the lab. To complete the project, first the hardware design must be created. The PS system is defined, external DDR and UART is connected. Lab 1 "Hello World" hardware design follows exactly the steps to be completed as they are written in the Xilinx Embedded workshop material Lab1 [Xil18a].

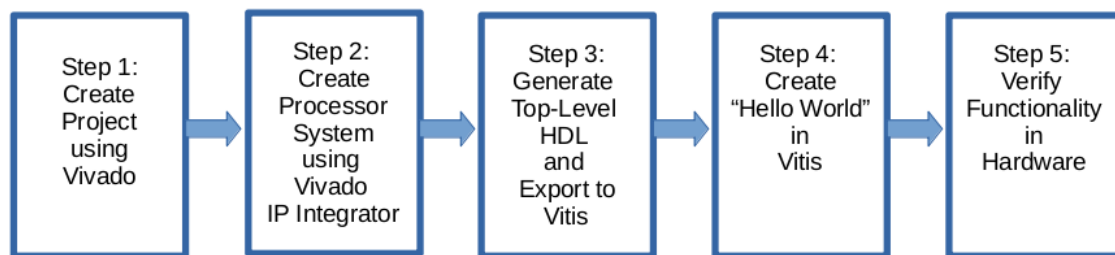


Figure 5.1.3: General Flow for Lab1 [Xil18a] page 2

## Create a Vivado Project and test the project in Vitis

1. **Launch Vivado and create an empty project targeting the Zybo/ZedBoard, using the VHDL language.**
  - (a) Open Vivado by selecting: **Start > All Programs > Xilinx Design Tools > Vivado 2020.2 > Vivado 2020.2**
  - (b) Open Vivado by starting the *xilinx.sh -v* script if you are logged in the lab. Select **Vivado 2020.2** from the list.
  - (c) Click **Create New Project** to start the wizard. You will see the *Create a New Vivado Project* dialog box. Click **Next**.
  - (d) Click in the field of the **Project Location** and type */embed/zybo/*. Enter **lab1** in the **Project Name** field. Make sure that the **Create project subdirectory** box is checked. Click **Next**. See figure 5.1.4.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- (e) In the **Project Type** form select **RTL Project**. Make sure that **Do not specify sources this time** box is checked and click **Next**. See figure 5.1.5.
- (f) For the **Default Part** window choose **Boards** and select **Zybo or Zedboard** click **Next**.
- (g) For choosing **Zybo** choose **Vendor digilentinc.com**, while for the **Zedboard** choose **em.avnet.com**, then select the board in the window or simply type in the search box the desired board and revision. The online Zedboard is rev D. See figure 5.1.6 and 5.1.7. Then click **Next** and finally click **Finish**.

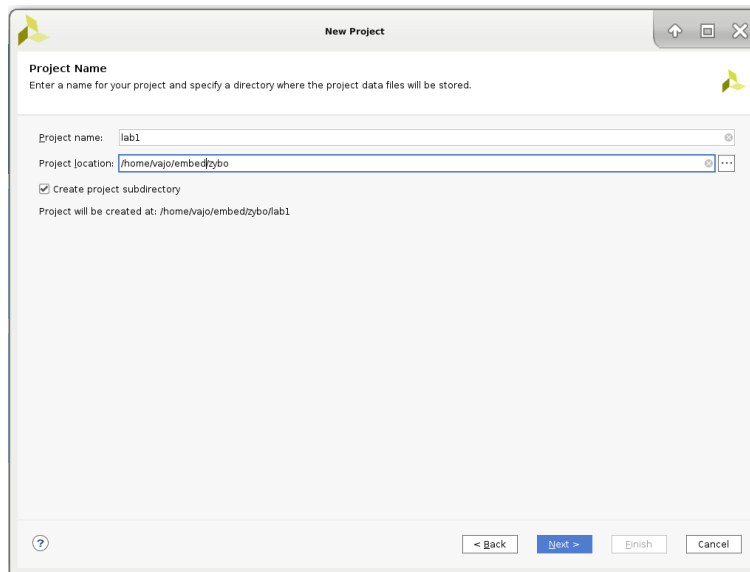


Figure 5.1.4: Project Name Entry





**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

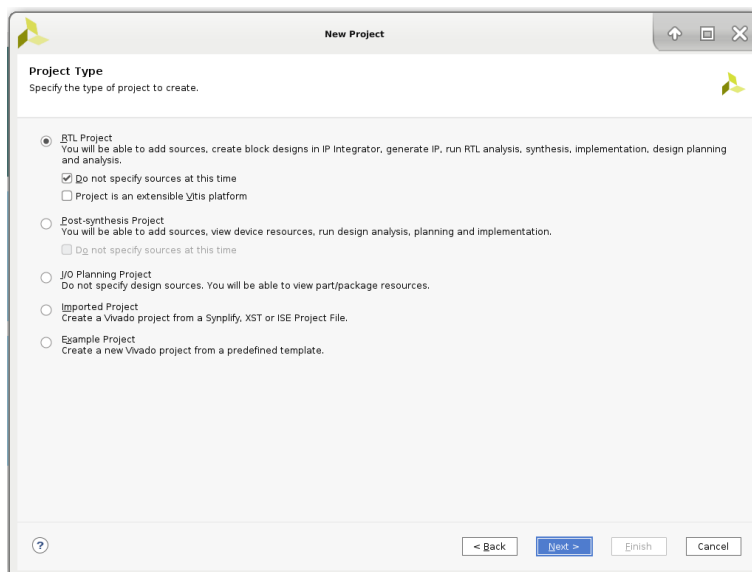


Figure 5.1.5: RTL Project selection

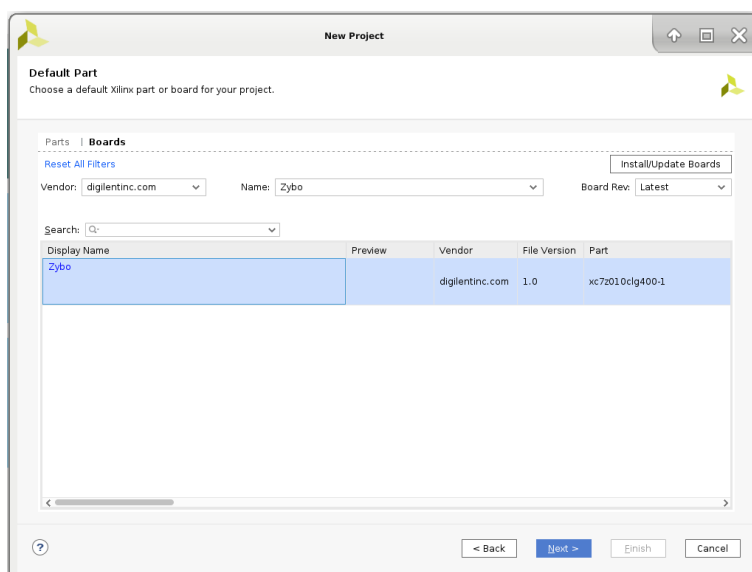


Figure 5.1.6: Board selection window for Zybo



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

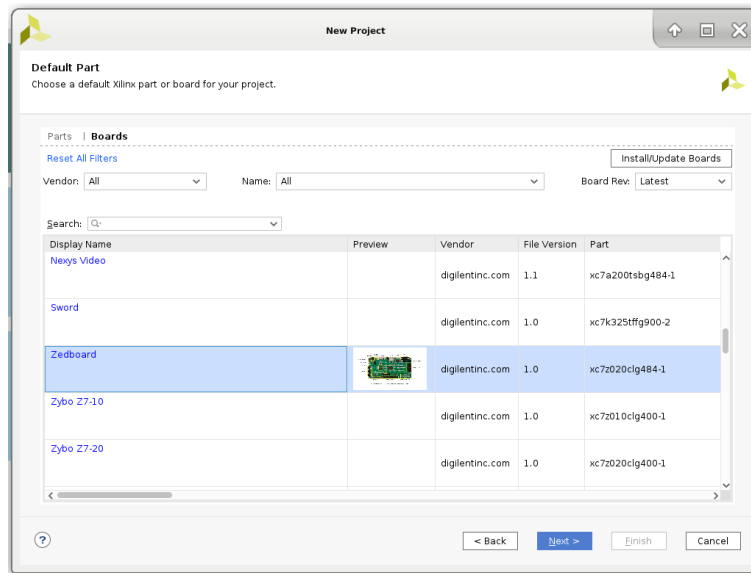


Figure 5.1.7: Board selection window for Zedboard

## 2. Creating the System Using the IP Integrator

- Use the IP Integrator to create a new Block Design, add the ZYNQ processing system block, and configure the processing system.
- In the Flow Navigator, click **Create Block Design** under **IP Integrator**. See figure 5.1.8
- Enter **system** for the design name and click **OK**. See figure 5.1.9.
- IP from the catalog can be added in different ways. Click + icon in the empty block diagram workspace or **Add IP** icon in the block diagram side bar or press **Ctrl + I**, or right-click anywhere in the Diagram workspace and select **Add IP**. See figure 5.1.10
- Once the IP Catalog is open, type “z” into the Search bar, find and double click on **ZYNQ7 Processing System** entry, or click on the entry and hit the Enter key to add it to the design (See 5.1.11) .
- Notice the message at the top of the Diagram window that *Designer Assistance Available*. Click **Run Block Automation** and select */processing\_system7\_0*. See figure 5.1.12
- In the *Run Block Automation* window, leave the default settings, including **Apply Board Preset** checked, and click **OK**. See figure 5.1.13. Once Block Automation has been complete, notice that ports have been automatically added for the DDR and Fixed IO, and some additional ports are now visible. The imported configuration for the Zynq related to the Zybo/Zedboard board has been applied which will now be modified. See figure 5.1.14





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- (h) Double-click on the added block to open its Customization window. Notice now the Customization window shows selected peripherals (with tick marks). This is the default configuration for the board applied by the block automation. See figure 5.1.15

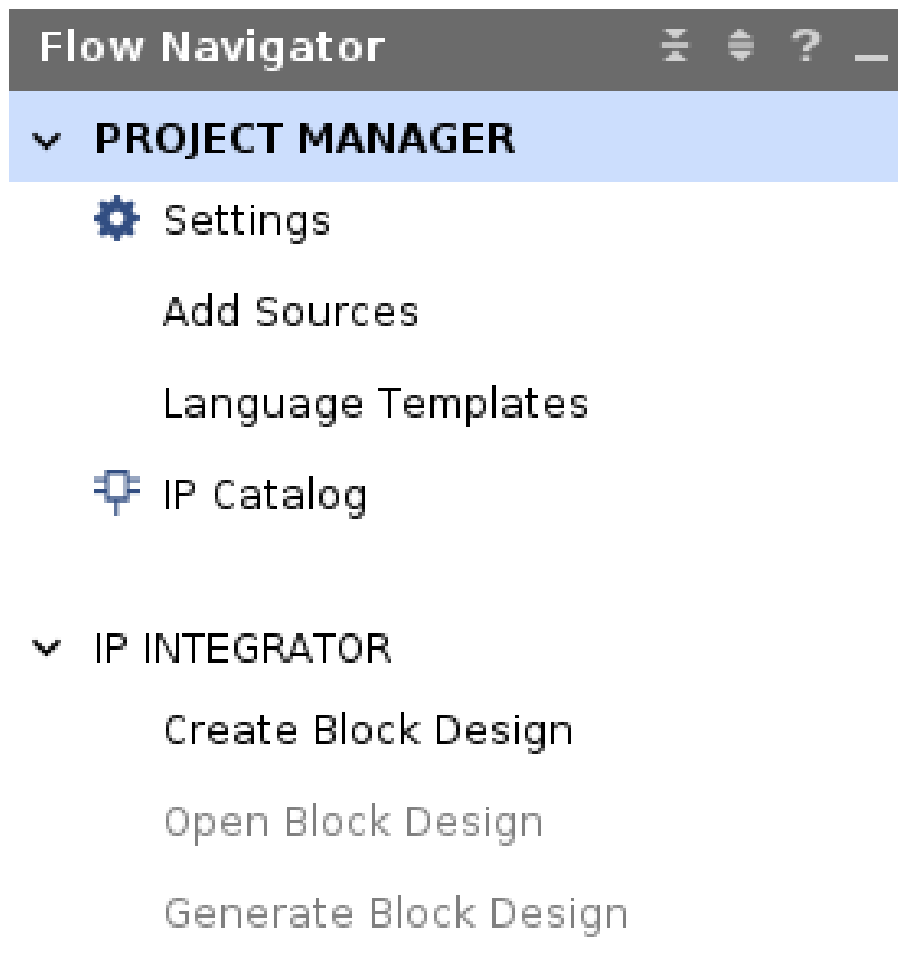


Figure 5.1.8: Create IP Integrator Block Diagram



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

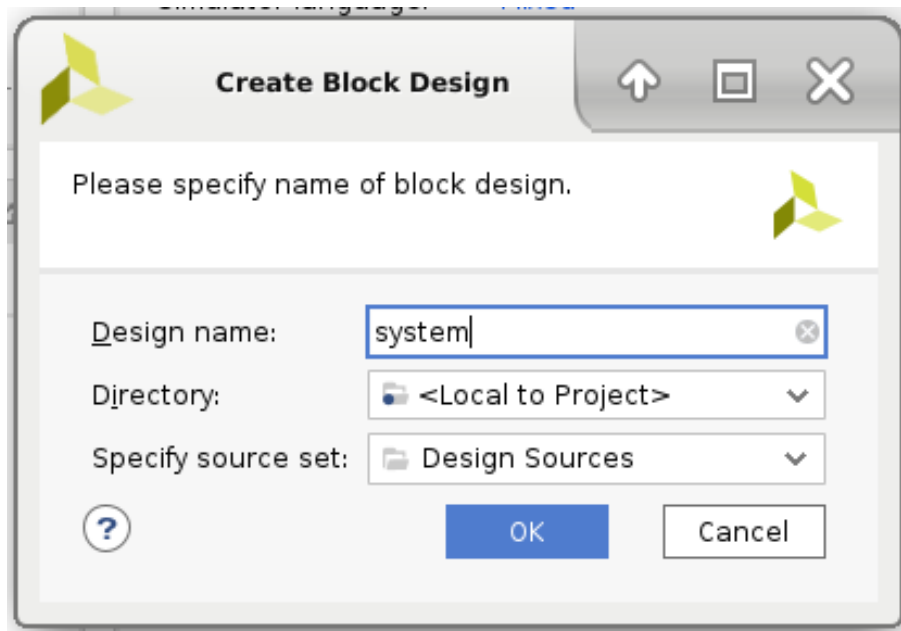


Figure 5.1.9: Create New Block Diagram

This design is empty. Press the **+** button to add IP.



Figure 5.1.10: Add IP to Block Diagram possibilities







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

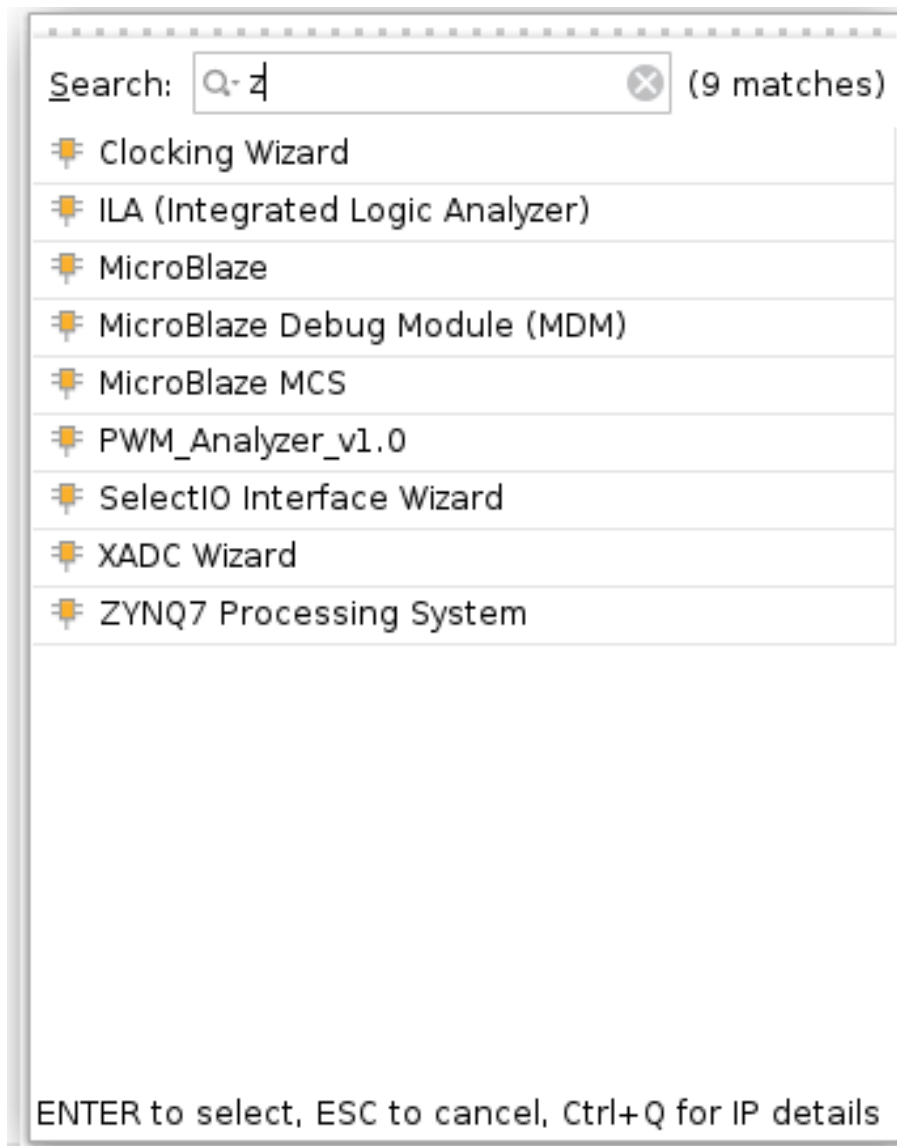


Figure 5.1.11: Add IP ZYNQ7 Processing System





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

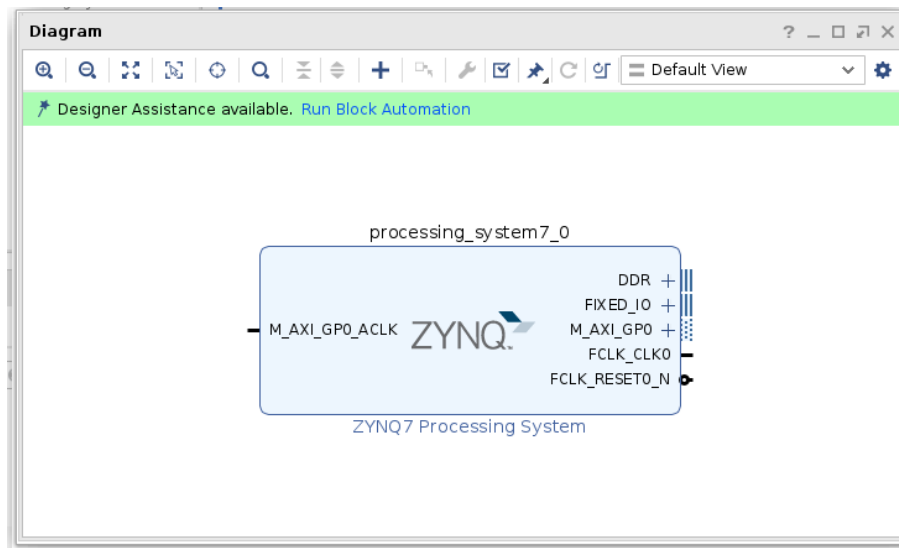


Figure 5.1.12: Run Block Automation Process

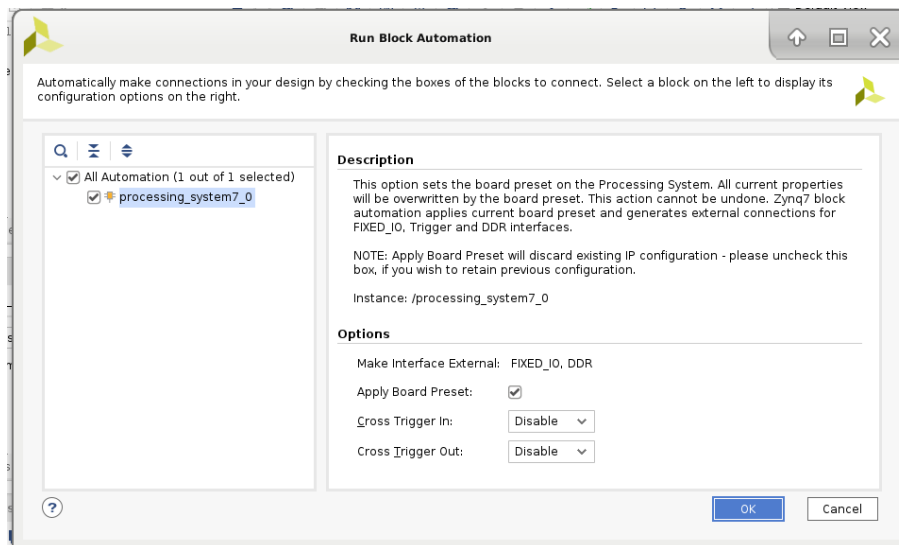


Figure 5.1.13: Selecting connections on Run Block Automation Process Settings



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

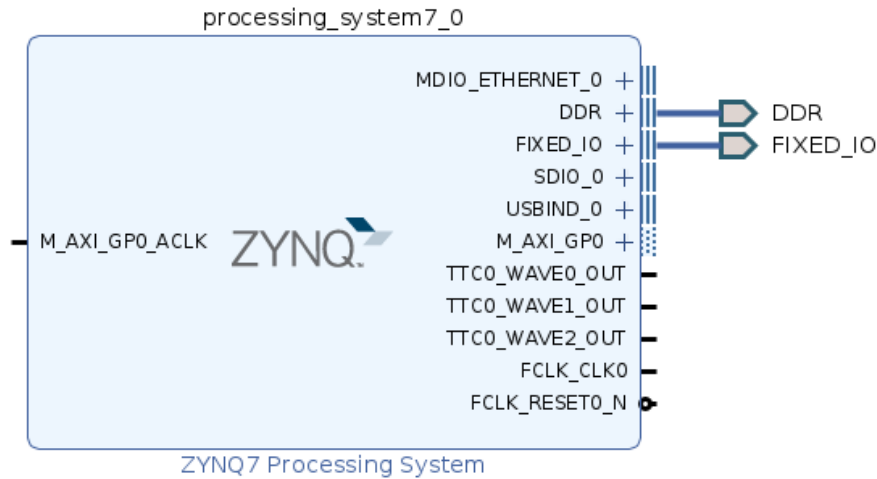


Figure 5.1.14: Schematic of the ZYNQ connected to DDR and FIXED\_IO

### 3. Configure the processing block with just UART 1 peripheral enabled.

- (a) A block diagram of the Zynq should now be open again (figure 5.1.15), showing various configurable blocks of the *Processing System*. At this stage, the designer can select or deselect various configurable blocks (highlighted in green) and change the system configuration.
- (b) Only the UART is required for this lab, so all other peripherals will be deselected.
- (c) Click on one of the peripherals (in green) in the *I/O Peripherals* block, or **select the MIO Configuration** tab on the left to open the configuration form. See figure 5.1.16
  - i. Expand **I/O peripherals** if necessary, and ensure all the following I/O peripherals are deselected except **UART 1** (figure 5.1.16).
  - ii. Remove **ENET 0**, **USB 0**, **SD 0** (figure 5.1.17)
  - iii. Expand **GPIO** to deselect **GPIO MIO**.
  - iv. Expand **Memory Interfaces** to deselect **Quad SPI Flash**.
  - v. Expand **Application Processor Unit** to disable **Timer 0**. See figure 5.1.17
  - vi. Select the **PS-PL Configuration** tab on the left. Expand **AXI Non Secure Enablement** ⇒ **GP Master AXI interface** and select **M AXI GPO interface** if not all ready selected(!). (figure 5.1.18).
  - vii. Expand **General** ⇒ **Enable Clock Resets** and deselect the **FCLK\_RESETO\_N** option. (figure 5.1)
  - viii. Select the **Clock Configuration** tab on the left. Expand the **PL Fabric Clocks** and select the **FCLK\_CLK0** option and click **OK**. See figure 5.1.19.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- ix. In the block design connect **FCLK\_CLK0** with **textbfM\_AXI\_GPO\_ACLK**. Hover your mouse over the connector port until the pencil button appears then connect the two signals (Figure 5.1.20).
- x. Click on the **Validate Design** button and make sure that there are no errors.

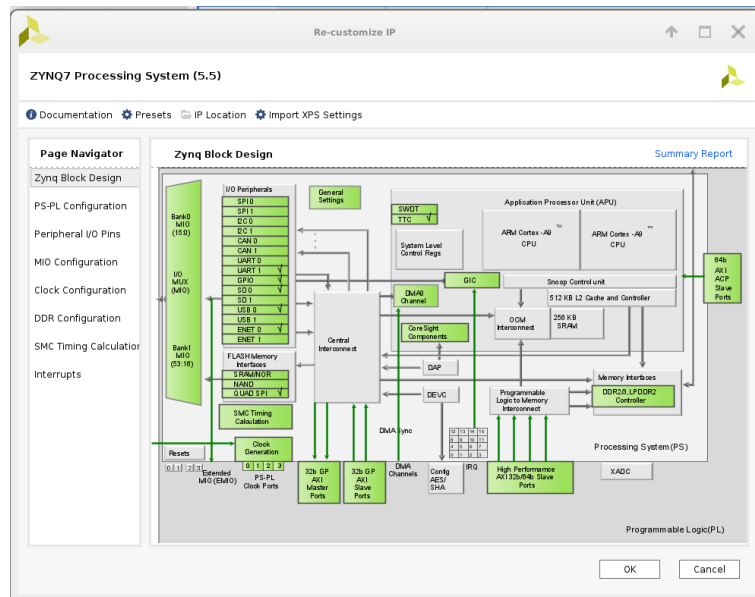


Figure 5.1.15: ZYNQ 7 IP Processing System Settings





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

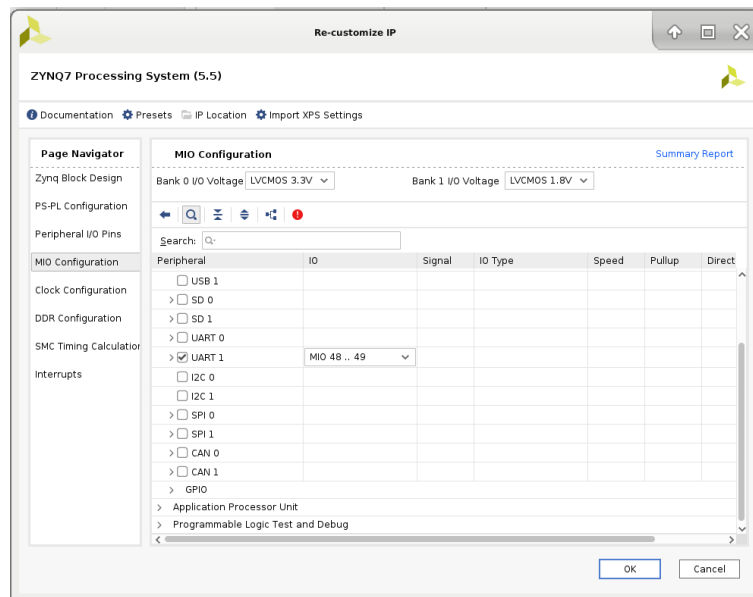


Figure 5.1.16: ZYNQ 7 IP Processing System Peripheral Settings





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

Peripheral	IO
<input checked="" type="checkbox"/> I/O Peripherals	
> <input type="checkbox"/> ENET 0	
> <input type="checkbox"/> ENET 1	
<input type="checkbox"/> USB 0	
<input type="checkbox"/> USB 1	
> <input type="checkbox"/> SD 0	
> <input type="checkbox"/> SD 1	
> <input type="checkbox"/> UART 0	
> <input checked="" type="checkbox"/> UART 1	MIO 48 .. 49
<input type="checkbox"/> I2C 0	
<input type="checkbox"/> I2C 1	
> <input type="checkbox"/> SPI 0	
> <input type="checkbox"/> SPI 1	
> <input type="checkbox"/> CAN 0	
> <input type="checkbox"/> CAN 1	
<input checked="" type="checkbox"/> GPIO	
<input type="checkbox"/> GPIO MIO	
<input type="checkbox"/> EMIO GPIO (Width)	
> <input type="checkbox"/> ENET Reset	
> <input type="checkbox"/> USB Reset	
> <input type="checkbox"/> I2C Reset	
<input checked="" type="checkbox"/> Application Processor Unit	
<input type="checkbox"/> Timer 0	
<input type="checkbox"/> Timer 1	
<input type="checkbox"/> Watchdog	

Figure 5.1.17: ZYNQ 7 IP configuration: deselect ENET 0, USB 0, SD 0, GPIO MIO, Timer 0





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

> Enable Clock Triggers		
▼ Enable Clock Resets		
FCLK_RESET0_N	<input type="checkbox"/>	Enables general purpose reset signal 0 for PL logic
FCLK_RESET1_N	<input type="checkbox"/>	Enables general purpose reset signal 1 for PL logic
FCLK_RESET2_N	<input type="checkbox"/>	Enables general purpose reset signal 2 for PL logic
FCLK_RESET3_N	<input type="checkbox"/>	Enables general purpose reset signal 3 for PL logic
▼ AXI Non Secure Enablement	0	Enable AXI Non Secure Transaction
▼ GP Master AXI Interface		
> M AXI GP0 interface	<input type="checkbox"/>	Enables General purpose AXI master interface 0
> M AXI GP1 interface	<input type="checkbox"/>	Enables General purpose AXI master interface 1

Figure 5.1.18: ZYNQ 7 IP configuration: deselect AXI GP0 Interface

Component	Clock Source	Requested Fre...	Actual Freque...	Range(MHz)
> Processor/Memory Clocks				
> IO Peripheral Clocks				
▼ PL Fabric Clocks				
<input type="checkbox"/> FCLK_CLK0	IO PLL	100	10.000000	0.100000 : 250.000000
<input type="checkbox"/> FCLK_CLK1	IO PLL	50	10.000000	0.100000 : 250.000000
<input type="checkbox"/> FCLK_CLK2	IO PLL	50	10.000000	0.100000 : 250.000000
<input type="checkbox"/> FCLK_CLK3	IO PLL	50	10.000000	0.100000 : 250.000000
> System Debug Clocks				
> Timers				

Figure 5.1.19: ZYNQ 7 IP deselect FCLK\_CLK0

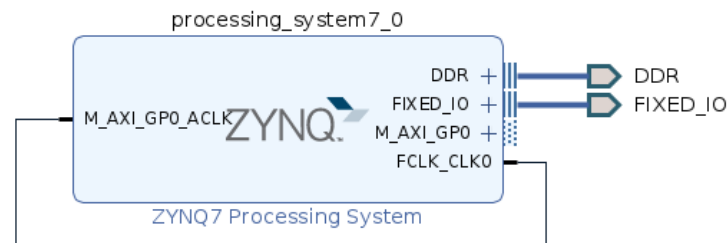


Figure 5.1.20: Updated schematic block design

#### 4. Generate IP Integrator Outputs, the top-level HDL, export the hardware to Vitis.

- Right-click on **system.bd**, and select **Create HDL Wrapper** to generate the top-level VHDL model. Leave the *Let Vivado manager wrapper and auto-update* option selected, and click **OK**. The **system\_wrapper.vhd** file will be created and added to the project (Figure 5.1.21).
- In the sources panel, right-click again on **system.bd**, and select **Generate Output Products** and click **Generate** to generate the Implementation, Simulation and Synthesis files for the



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

design. Select **Global** You can also click on **Generate Block Design** in the *Flow Navigator pane* to do the same. See figure 5.1.22

**Note:** If the synthesis option is **Global**, only wrapper files are generated during the block design generation phase, and the design will be synthesized as a whole at the synthesis stage. If the synthesis option is **Out of context per IP** or **Out of context per Block design**, the wrapper of the IP or block design will be generated and synthesized during block design generation, and the generated netlists will be combined together at the synthesis stage. See figure 5.1.23. Double-click on the file to see the content in the Auxiliary pane.

- (c) Notice that the VHDL file is already Set As the Top module in the design, indicated by the icon in front of the **system\_wrapper.vhd**.
- (d) In the main menu Select **File > Export > Hardware** and click **Next**. See figure 5.1.24.  
**Note:** Since we do not have any hardware in Programmable Logic (PL) there is no bitstream to generate, hence the *Include bitstream* option is not necessary at this time (figure 5.1.25). Click **Next**.
- (e) As in figure 5.1.26 name the XSA file to **system\_wrapper** and specify the folder name where the project is exported **Export to: /embed/lab1**. Click **Next** and then click **Finish**.

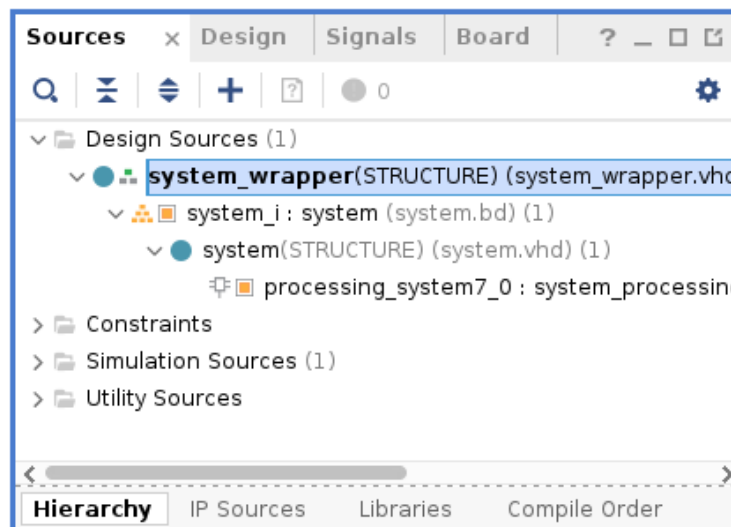


Figure 5.1.21: System Wrapper File Created





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

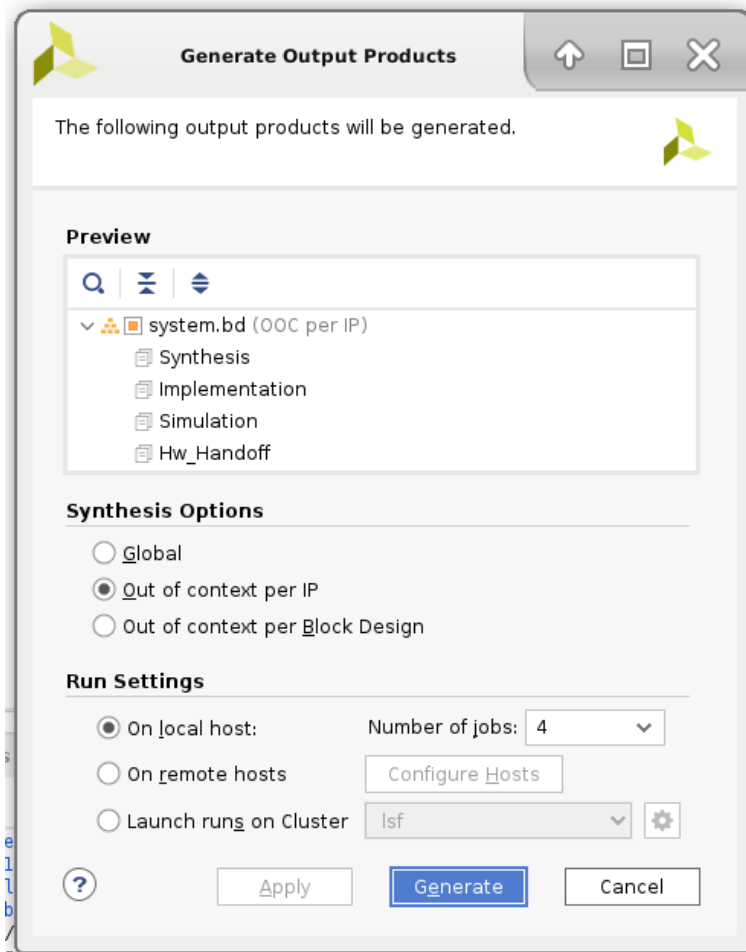


Figure 5.1.22: Generate Output Products





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

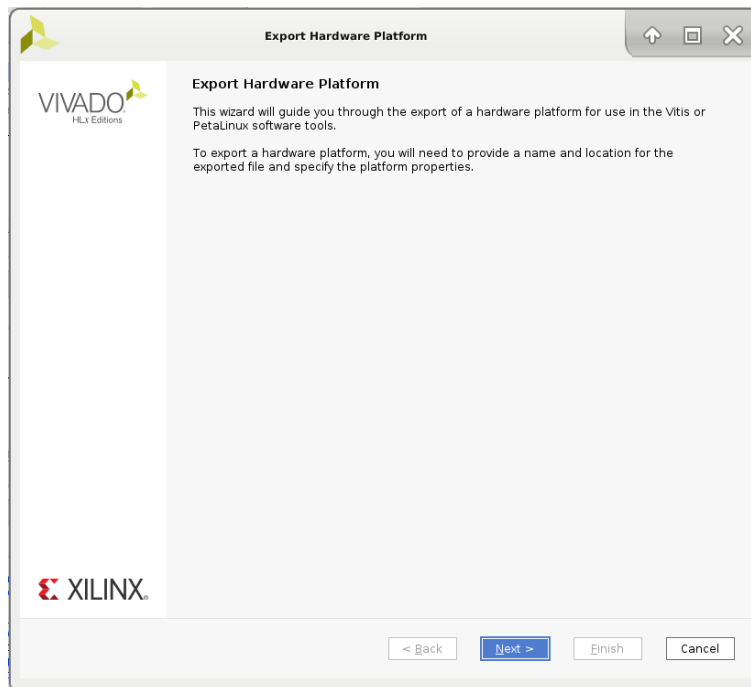


Figure 5.1.23: Export Hardware Platform





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

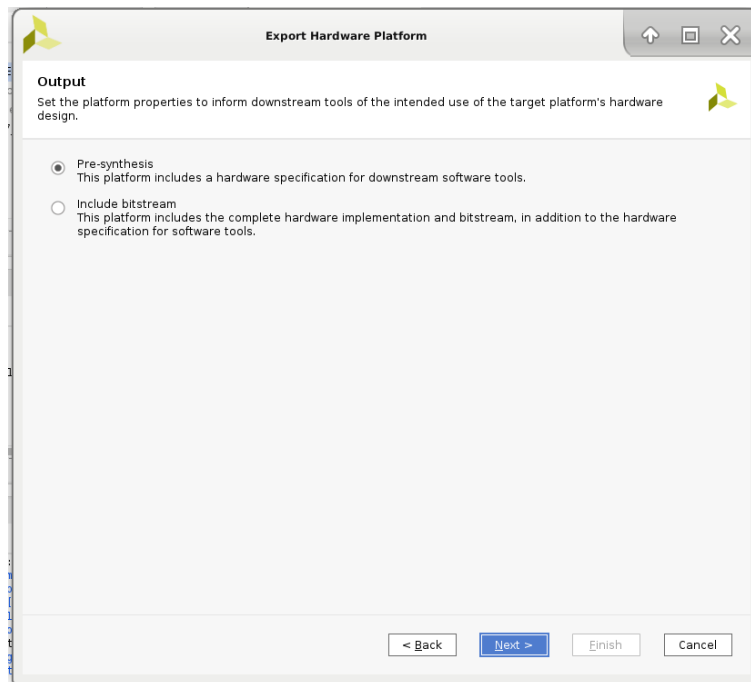


Figure 5.1.24: Export Hardware Platform

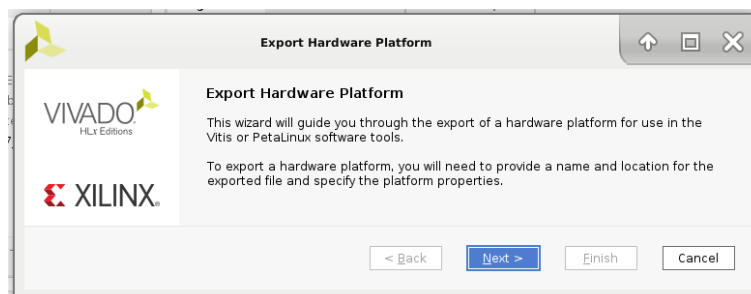


Figure 5.1.25: Export Hardware Platform Window



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

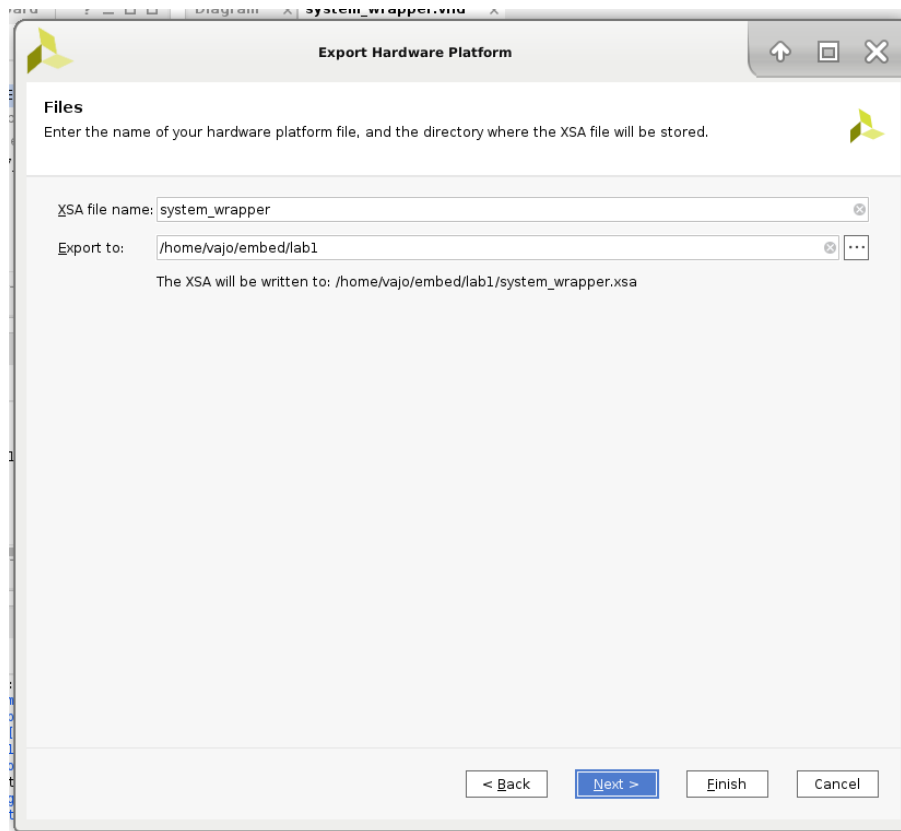


Figure 5.1.26: Export Hardware - Specify XSA file name and Export to Directory

The XSA (Xilinx Support Archive) is a container file that contains all the information needed to build a platform for a users target device. One of the files here is the Hardware Hand-off File (HWH). This HWH file is created when the output products is ran on a Block Design (BD). Only the information in the Block Design (BD) will be contained in the HWH. The HWH file is used by the software tools to abstract all the information needed to build a targeted application to a users device; such as the CPU (or CPUs), Buses, IP and the ports and pins used in the system such as interrupts.

#### 5. Launch Vitis. Generate Hello World Application in Vitis

- In the main menu select **Tools** ⇒ **Launch Vitis IDE**;
- Specify the workspace; **embed/workspace/lab1** and click **Launch**
- On the Vitis welcome screen click **Create Platform Project** See figure 5.1.27;



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- (d) In the **New Platform Project** specify the *Platform project name*: **zybo\_system** or **zed\_system** depending on which board you are using. Click **Next** as shown in 5.1.28. Click **Next**.
- (e) In the **Create a new platform from hardware XSA**. Click **Browse** (5.1.29) to specify the XSA file. Xilinx hardware designs are created with the Vivado The XSA (Xilinx Support Archive) file was created when the project was exported from Vivado to Vitis. The XSA proprietary file format is used by the Vitis software platform to support the project in Vitis. Click **Next**
- (f) The Vitis IDE open with the platform project created (5.1.30). In the menu select **File** ⇒ **New** ⇒ **Application Project** for creating the application "Hello World". This will open the application project creation wizard (5.1.31). Click **Next**.
- (g) The next window is the domain selection window. for this first project we create standalone\_domain. Click **Next**. See 5.1.34.
- (h) In the new Application project examples select "Hello World" and click **Finish**.
- (i) In the file Explorer window right click on the **zed\_project**. Also expand the *hello\_world\_system* => hello\_world => src and double click on the **helloworld.c** .C file to opened in the editor window. Right click on the *hello\_word\_system* and select **Build project**.
- (j) If you worked with Zybo connect the board to your PC and power on.
- (k) If you worked with the Zedboard start **putty.exe** and login to "mazsola" to open a tunnel for the hardware server, where the Zedboard is connected. Also login to the computer "nyolcas".
- (l) For Zybo/Zed open RealTerm serial terminal. Select the UART port *CypresVirtualCom0* and setup to *Baud: 115200, Parity: none, Data Bits: 8, Stop Bits: 1* then connect the terminal (Figure 5.1.39).
- (m) In the Main menu select **Xilinx** ⇒ **XSCT Console**. Then in the console window type **connect** press *Enter*, then type **reset** and press *Enter*. In the XSCT window should be displayed something similar like in 5.1.38.
- (n) In the Explorer window right click on the application project *hello\_world\_system* select **Run as** ⇒ **Run Configurations**. Once the *Run Configurations* windows opens, twice click on **System Project Debug** to create an application debug **SystemDebugger\_hello\_world\_system** (Figure 5.1.39). Then click on **Run**
- (o) "Hello World" appears on the RealTerm Terminal, as shown in the figure (5.1.40).





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

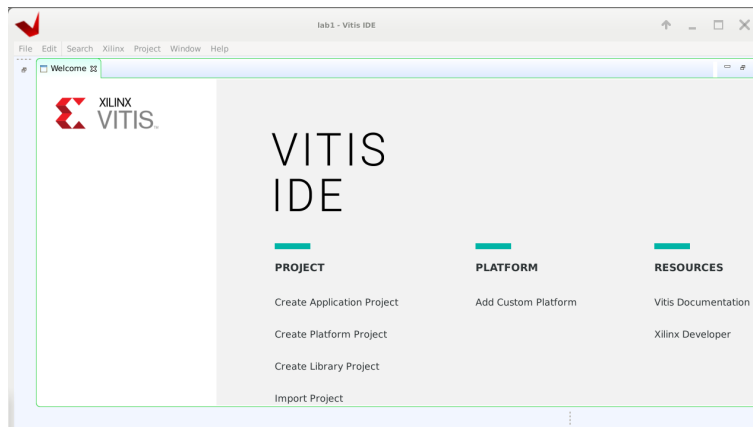


Figure 5.1.27: Vitis IDE start window

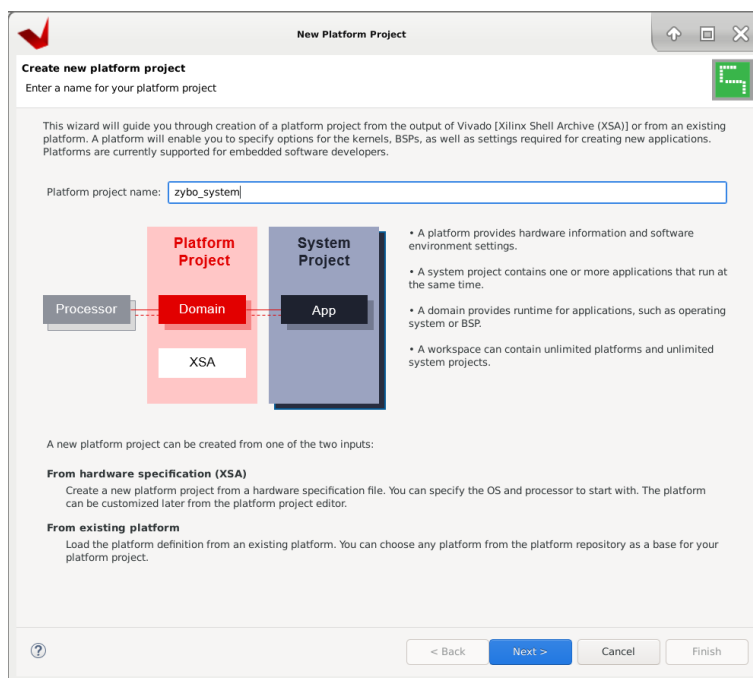


Figure 5.1.28: Create New Platform Project





**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

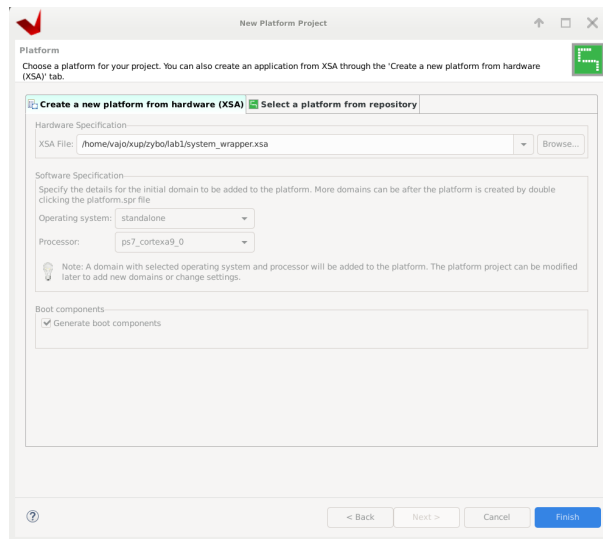


Figure 5.1.29: Create a new platform from hardware (XSA)

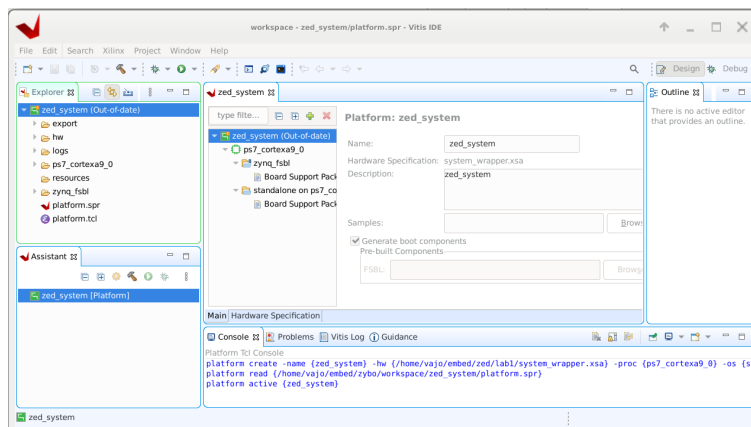


Figure 5.1.30: Vitis IDE with the platform project created



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

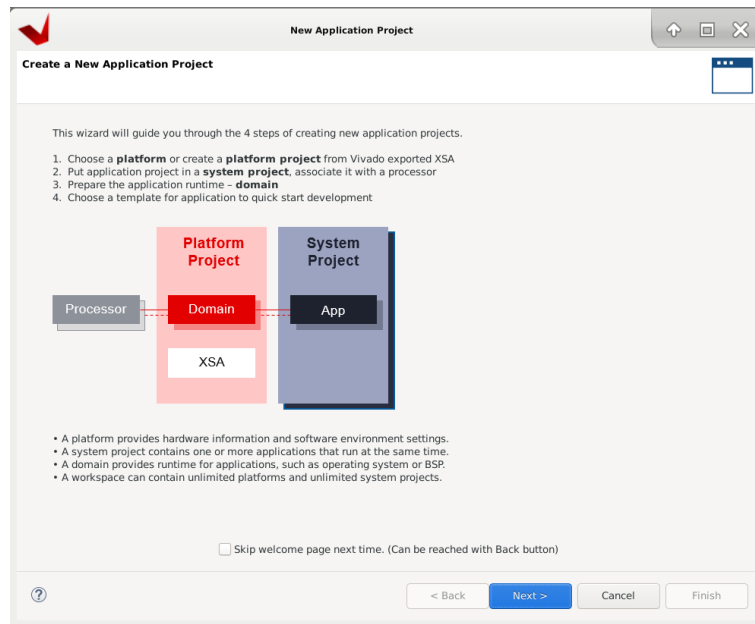


Figure 5.1.31: New Application Project Wizard

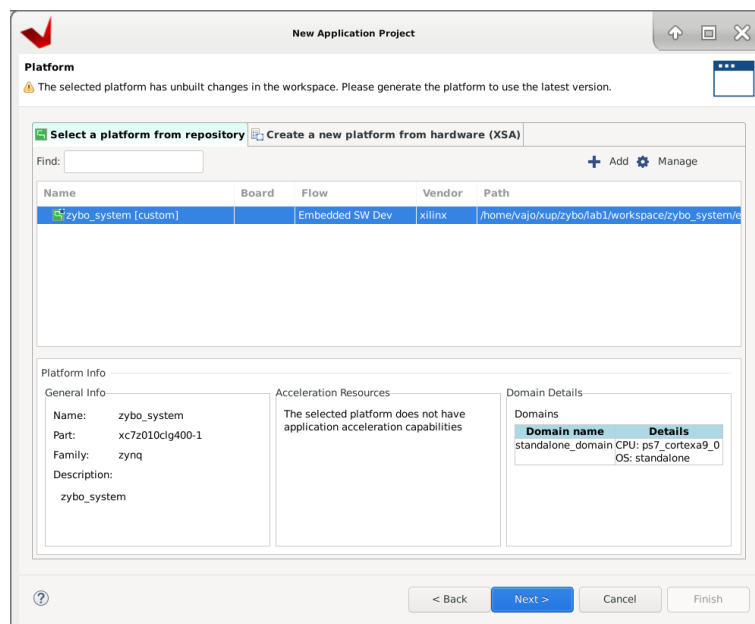


Figure 5.1.32: Select Platform





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

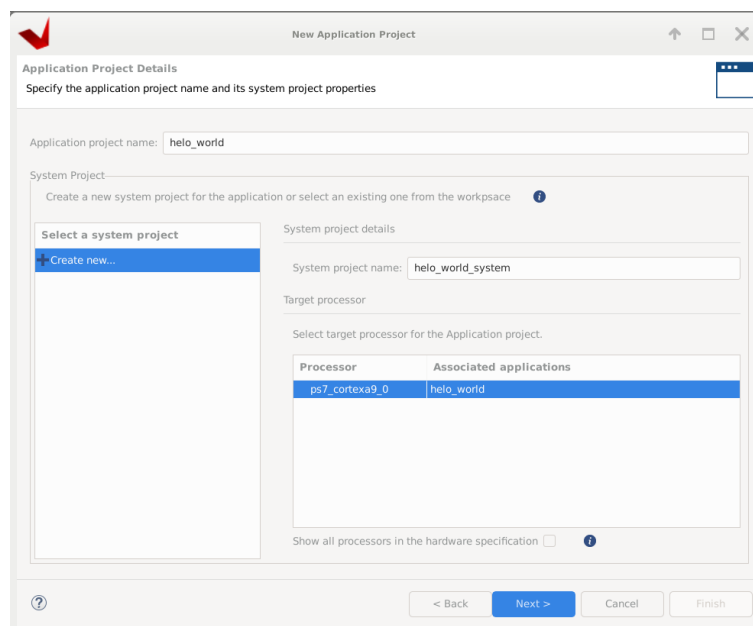


Figure 5.1.33: Specify the Application Project Name





**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

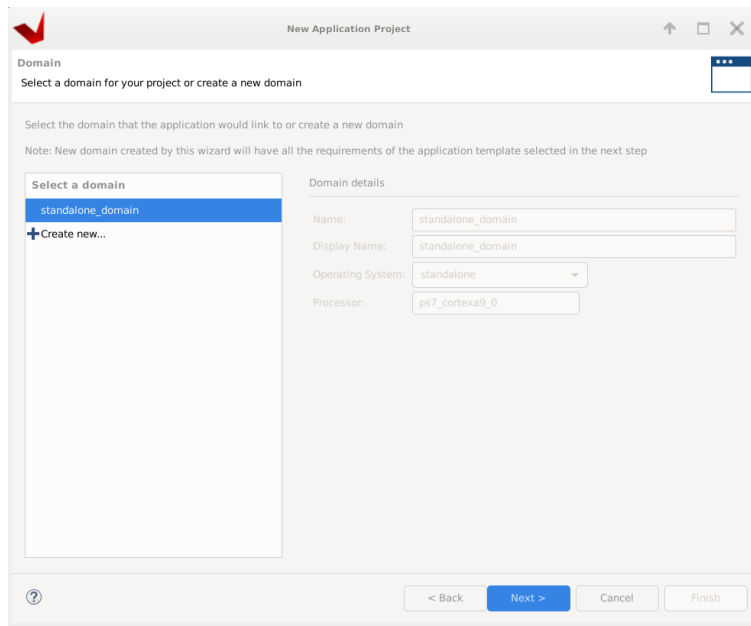


Figure 5.1.34: Select domain as standalone\_platform

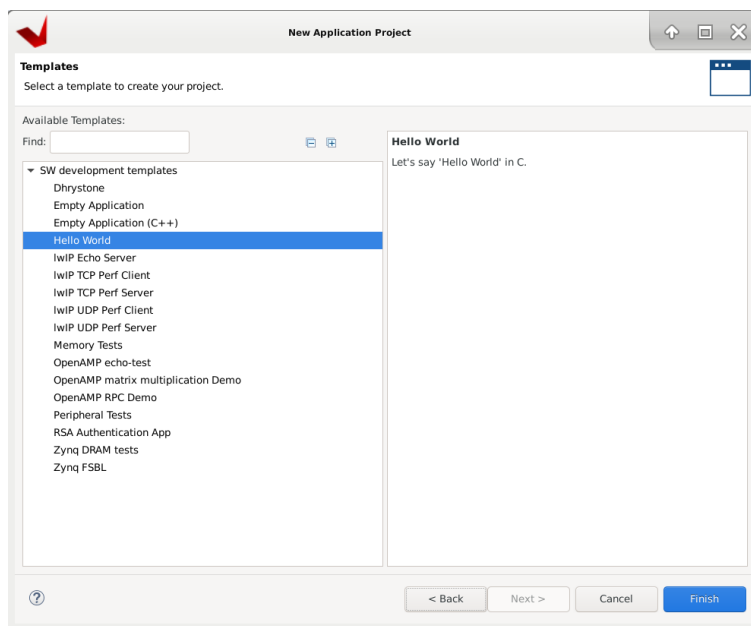


Figure 5.1.35: Select "Hello World" example project





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

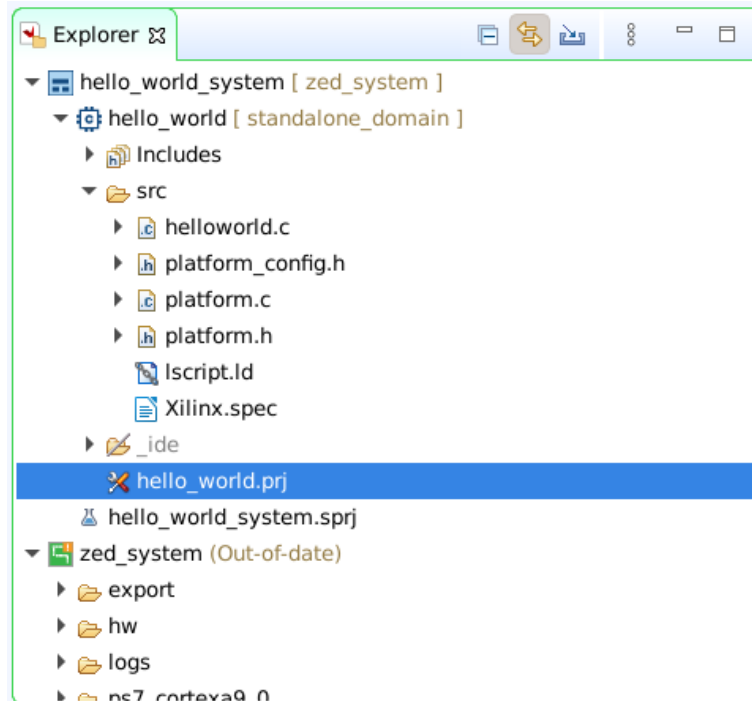


Figure 5.1.36: Application project "hello\_world" Explorer view

```
47
48 #include <stdio.h>
49 #include "platform.h"
50 #include "xil_printf.h"
51
52
53 int main()
54 {
55     init_platform();
56
57     print("Hello World\n\r");
58     print("Successfully ran Hello World application");
59     cleanup_platform();
60     return 0;
61 }
62
```

Figure 5.1.37: The "Hello World" program





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

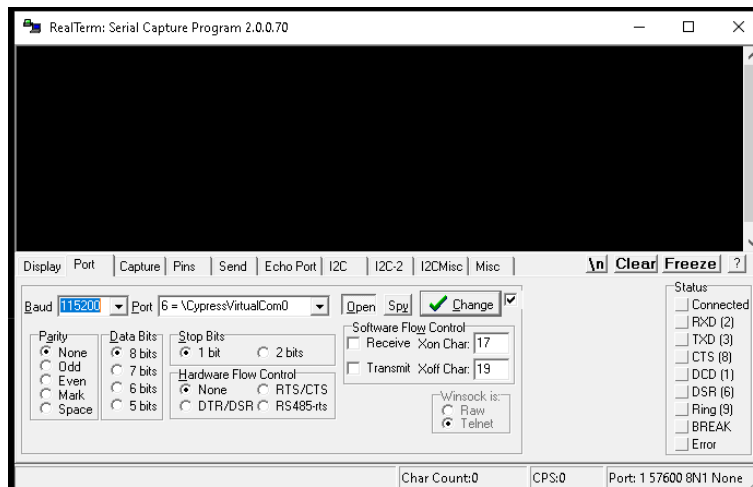


Figure 5.1.38: RealTerm terminal setup

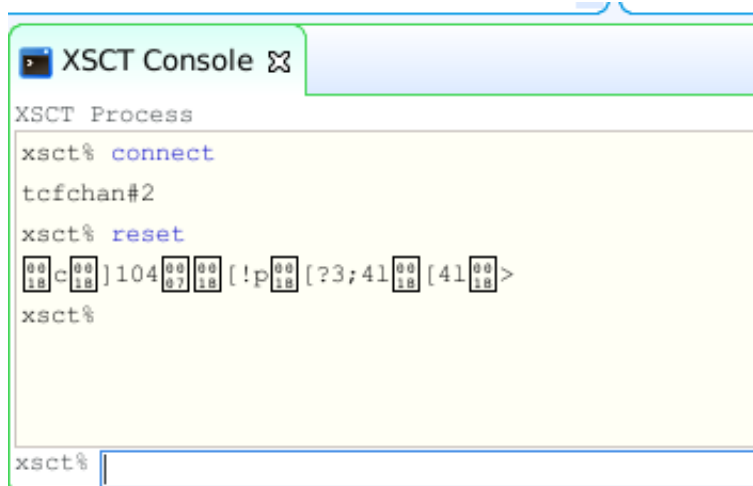


Figure 5.1.39: XSCt console connection view



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

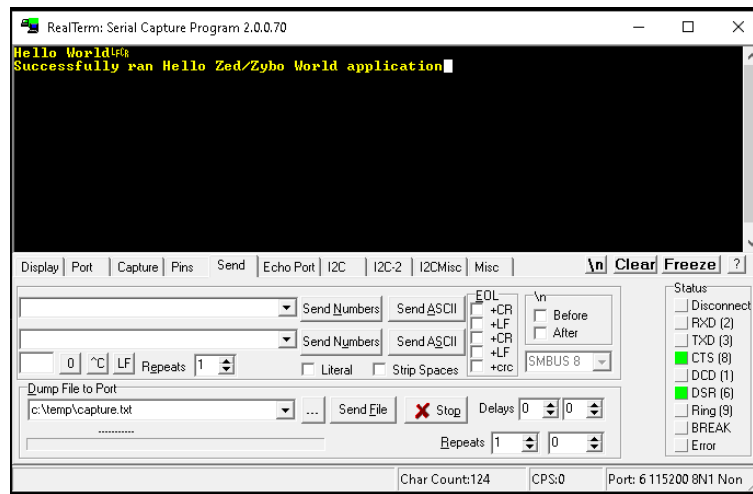


Figure 5.1.40: Caption

In the part 1 of the laboratory work 1 (called Lab1) we passed thru the steps, which have to be done for each embedded system project using Zynq architecture. True there were not explained the steps. Detailed information about the development process and “what to do and why to do” are explained in the Xilinx documentations (search for User Guides ug940, ug1165). For other information, tutorials and video tutorials start the DocNav (Document Navigator), installed together with the Vitis installation. Some of previously presented “Hello World” project material are based on Xilinx XUP (Xilinx University Program) and other tutorials such as ug1165 material [Xil18a].

## 5.1.2 Lab 1 Part 2: Led Blinking through MIO

### Introduction

This part of the lab consists of creating a software application to blink a LED connected to PS. The LED is connected to the multiplexed IO (MIO), which is directly accessible by the ARM A9 processors.

### Objectives

- Create a Vivado project for a Zynq system;
- Use IP Integrator to create a hardware system;
- Enable MIO in the Zynq IP;
- Use Vitis to create a standard “Hello World” test project;
- Edit the “Hello World” program to blink the MIO LED;





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- Run the test application on the board.

## Procedure

In the first lab the design steps will be presented in detail to provide adequate instructions. Detailed instructions are given which must be followed in order to complete the lab.

This part of the lab consists of creating a software application to blink an LED connected to PS. No special hardware is required for this part but a minimal configuration is required (invoking and parameterizing the Zynq IP). Figure 5.1.41 resumes the steps to follow.

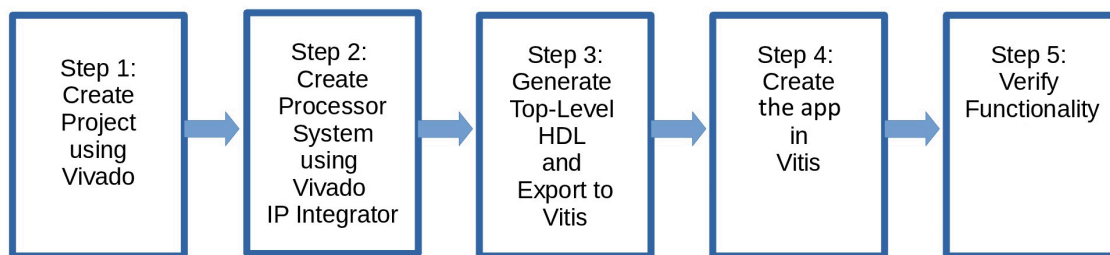


Figure 5.1.41: General Flow for Lab1 Part2

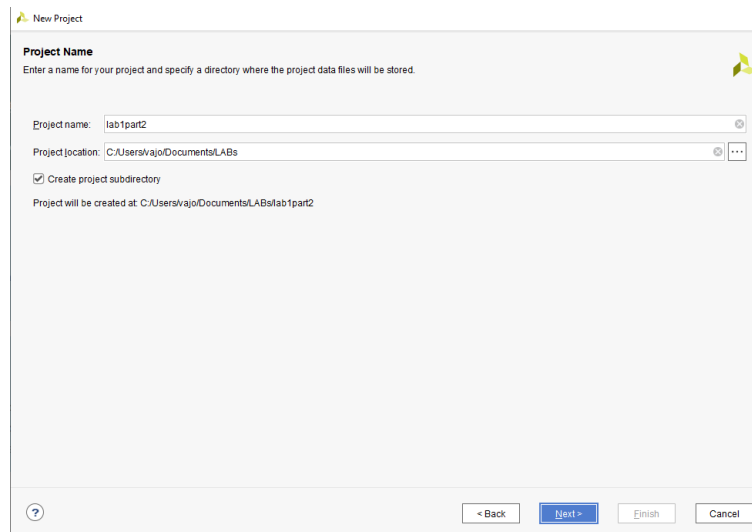
## Create a Vivado Project

1. Launch Vivado and create an empty project targeting the ZedBoard, using the VHDL language.
  - (a) Open Vivado by selecting: **Start > All Programs > Xilinx Design Tools > Vivado 2020.2 > Vivado 2020.2**
  - (b) Open Vivado by starting the *xilinx.sh -v* script if you are logged in the lab. Select **Vivado 2020.2** from the list.
  - (c) Click **Create New Project** to start the wizard. You will see the *Create a New Vivado Project* dialog box. Click **Next**.
  - (d) Click in the field of the **Project Location** and type */embed/zed/*. Enter *lab1part2* in the **Project Name** field. Make sure that the *Create project subdirectory* box is checked. Click **Next**. See figure 5.1.42.
  - (e) In the **Project Type** form select **RTL Project**. Make sure that *Do not specify sources this time* box is checked and click **Next**. See figure 5.1.43.
  - (f) For the **Default Part** window choose **Boards** and select *Zybo or Zedboard* click **Next**.
  - (g) For *Zedboard* choose *em.avnet.com*, then select the board in the window or simply type in the search box the desired board and revision. The online Zedboard is rev D. See figure 5.1.44. Then click **Next** and finally click **Finish**.





**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015



**New Project**

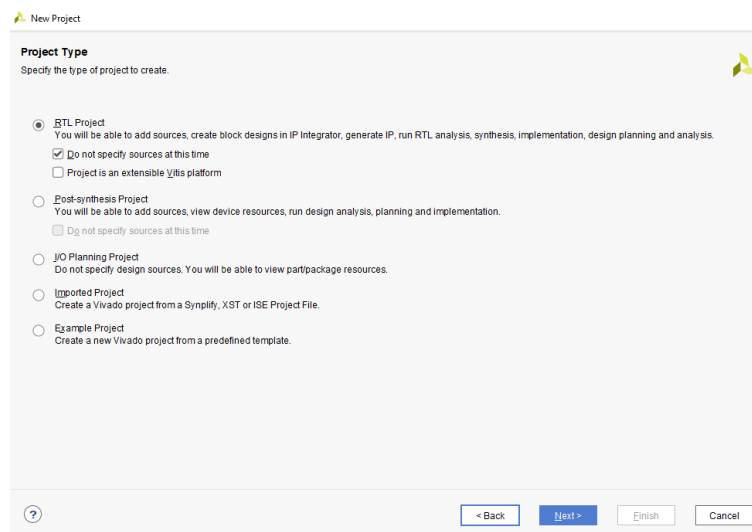
**Project Name**  
Enter a name for your project and specify a directory where the project data files will be stored.

Project name:

Project location:

Create project subdirectory  
Project will be created at: C:/Users/vajo/Documents/LABS/lab1part2

Figure 5.1.42: Project Name Entry



**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  
 Project is an extensible Vitis platform
- 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
- IO 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.

Figure 5.1.43: RTL Project selection



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

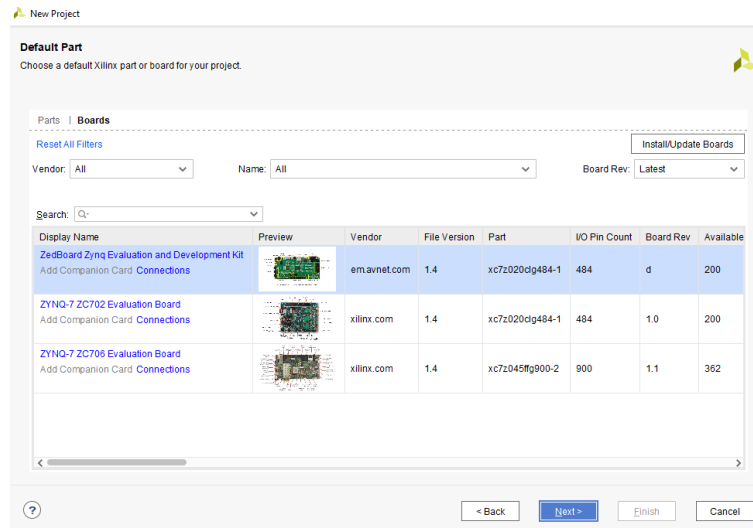


Figure 5.1.44: Board selection window for Zedboard

## 2. Creating the System Using the IP Integrator

- Use the IP Integrator to create a new Block Design, add the ZYNQ processing system block, and configure the processing system.
- In the Flow Navigator, click **Create Block Design** under **IP Integrator**.
- Enter **system** for the design name and click **OK**. See figure 5.1.45.
- IP from the catalog can be added in different ways. Click + icon in the empty block diagram workspace or **Add IP** icon in the block diagram side bar or press **Ctrl + I**, or right-click anywhere in the Diagram workspace and select **Add IP**. See figure 5.1.46
- Once the IP Catalog is open, type “z” into the Search bar, find and double click on **ZYNQ7 Processing System** entry, or click on the entry and hit the Enter key to add it to the design (See 5.1.47) .
- Notice the message at the top of the Diagram window that *Designer Assistance Available*. Click **Run Block Automation** and select `/processing_system7_0`. See figure 5.1.48
- In the *Run Block Automation* window, leave the default settings, including **Apply Board Preset** checked, and click **OK**. See figure 5.1.49. Once Block Automation has been complete, notice that ports have been automatically added for the DDR and Fixed IO, and some additional ports are now visible. The imported configuration for the Zynq related to the Zynq/Zedboard board has been applied which will now be modified. See figure 5.1.50





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- (h) Double-click on the added block to open its Customization window. Notice now the Customization window shows selected peripherals (with tick marks). This is the default configuration for the board applied by the block automation. See figure 5.1.51

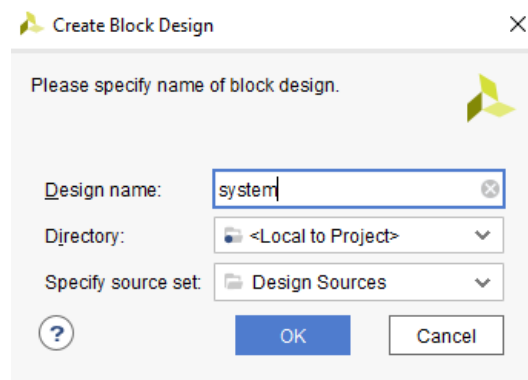


Figure 5.1.45: Create New Block Diagram

This design is empty. Press the **+** button to add IP.



Figure 5.1.46: Add IP to Block Diagram possibilities

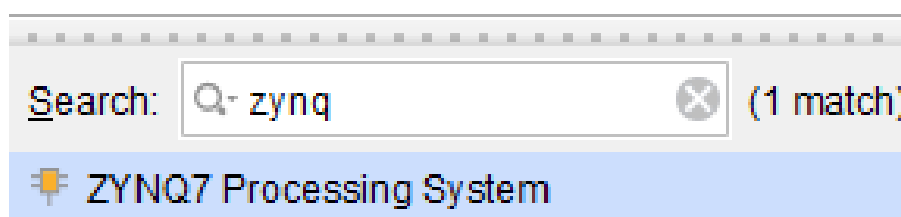


Figure 5.1.47: Add IP ZYNQ7 Processing System



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

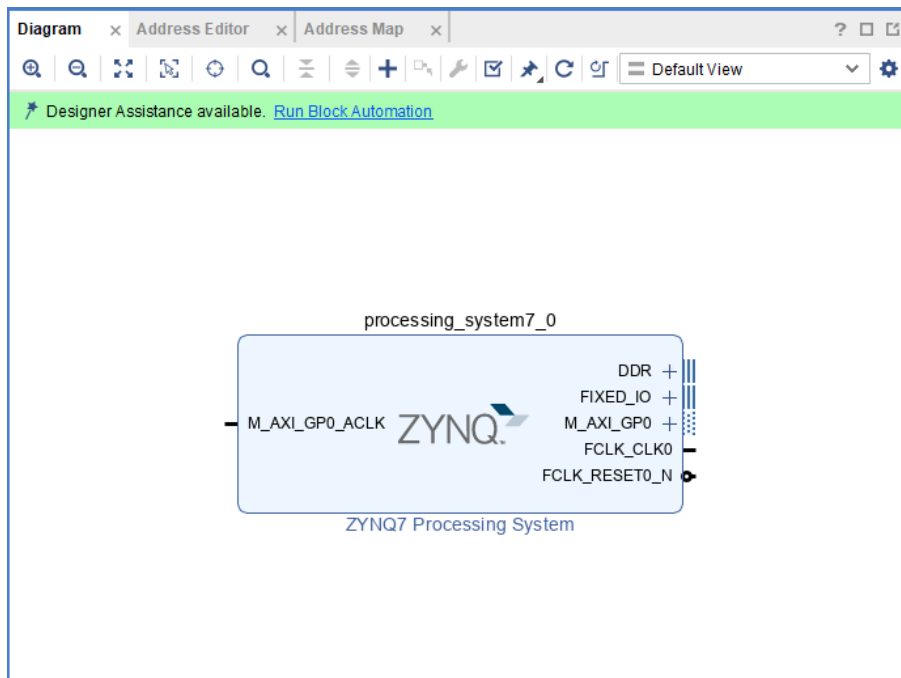


Figure 5.1.48: Run Block Automation Process





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

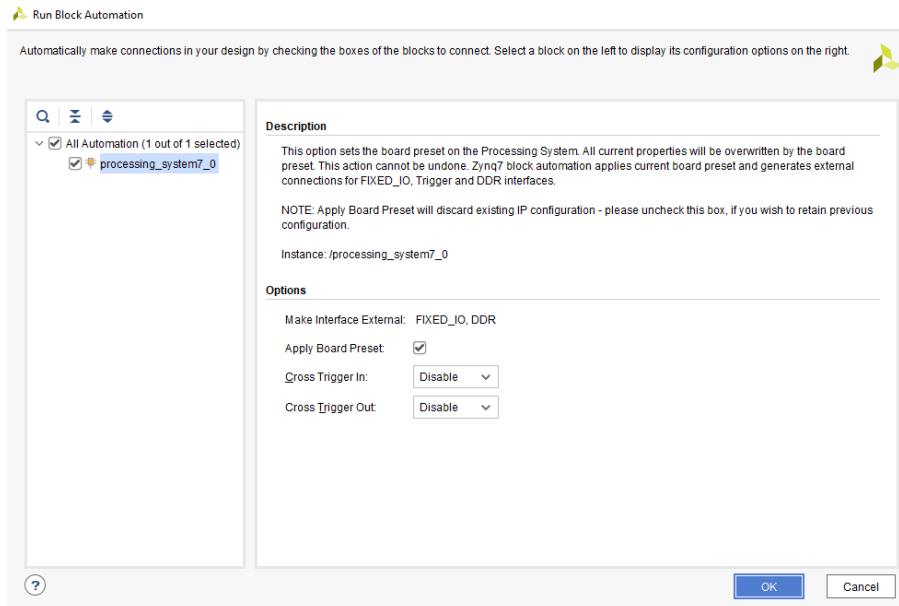


Figure 5.1.49: Selecting connections on Run Block Automation Process Settings

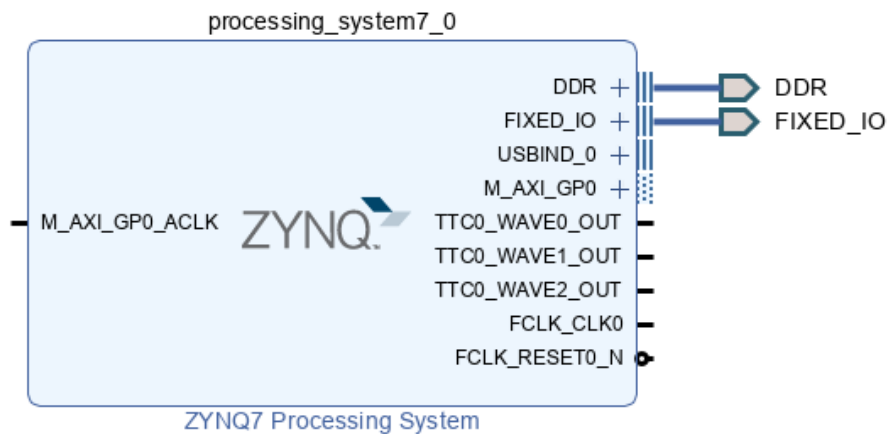


Figure 5.1.50: Schematic of the ZYNQ connected to DDR and FIXED\_IO

3. Configure the processing block with just GPIO MIO peripheral enabled.

(a) A block diagram of the Zynq should now be open again, showing various configurable blocks



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

of the *Processing System*. At this stage, the designer can select or deselect various configurable blocks (highlighted in green) and change the system configuration.

- (b) Only the GPIO MIO is required for this lab, so all other peripherals will be deselected.
- (c) Click on one of the peripherals (in green) in the *I/O Peripherals* block, or **select the MIO Configuration** tab on the left to open the configuration form. The procedures are resumed in Figure 5.1.51 and Figure 5.1.52
  - i. Expand ***I/O peripherals*** if necessary, and ensure all the following I/O peripherals are deselected except **GPIO MIO**
  - ii. Expand ***Memory Interfaces*** to deselect ***Quad SPI Flash***.
  - iii. Expand ***Application Processor Unit*** to disable ***Timer 0***.
  - iv. Select the ***PS-PL Configuration*** tab on the left. Expand ***AXI Non Secure Enablement > GP Master AXI interface*** and select ***M AXI GP0 interface*** if not all ready selected(!).
  - v. Expand ***General > Enable Clock Resets*** and deselect the ***FCLK\_RESETO\_N*** option.
  - vi. Select the ***Clock Configuration*** tab on the left. Expand the ***PL Fabric Clocks*** and select the ***FCLK\_CLK0*** option and click **OK**
  - vii. In the block design connect ***FCLK\_CLK0*** with ***textbfM\_AXI\_GP0\_ACLK***. Hover your mouse over the connector port until the pencil button appears then connect the two signals
  - viii. Click on the **Validate Design** button and make sure that there are no errors.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

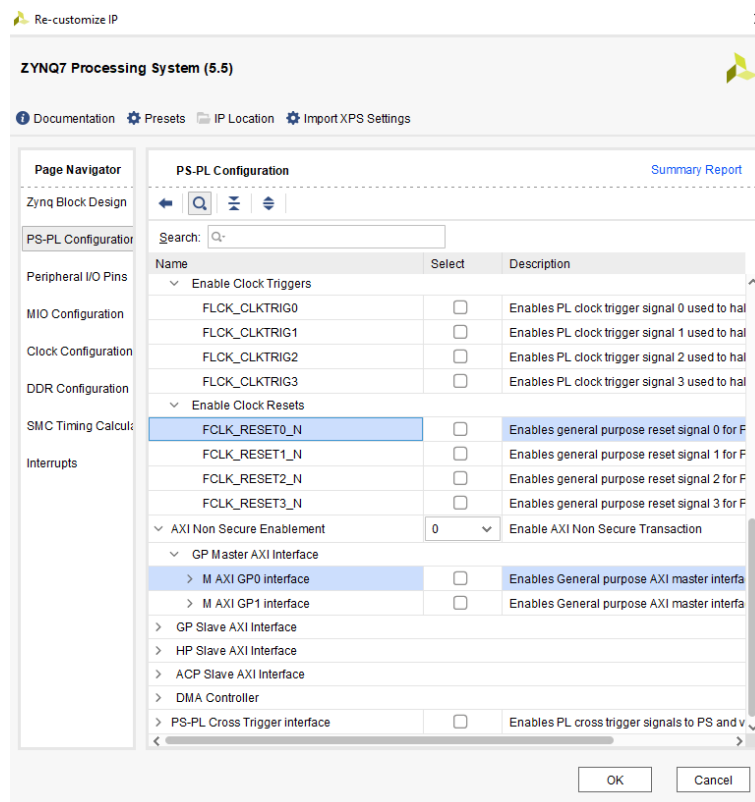


Figure 5.1.51: ZYNQ 7 IP Processing System Settings





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

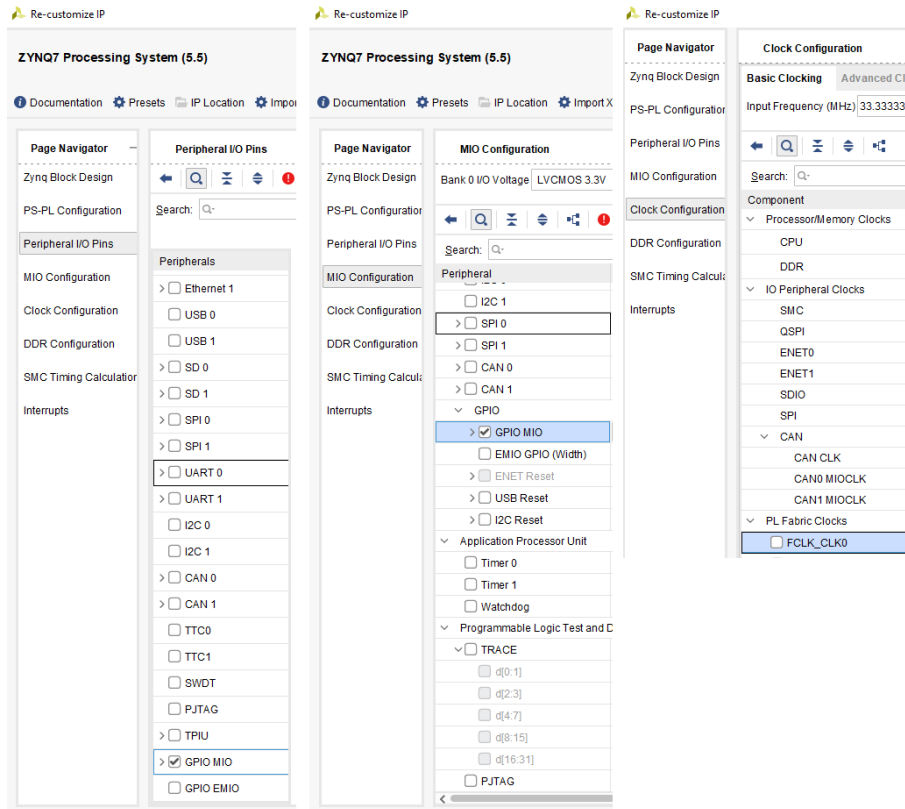


Figure 5.1.52: ZYNQ 7 IP Processing System Peripheral, MIO and Clock Settings

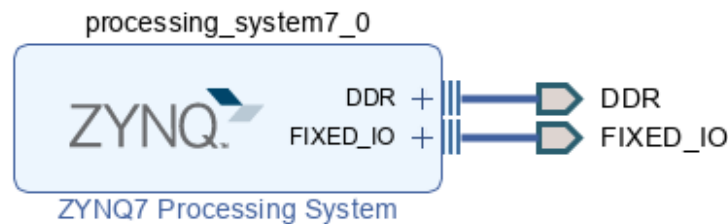


Figure 5.1.53: Updated schematic block design

4. Generate IP Integrator Outputs, the top-level HDL, export the hardware to Vitis and start Vitis.

- (a) Right-click on **system.bd**, and select **Create HDL Wrapper** to generate the top-level VHDL model. Leave the *Let Vivado manager wrapper and auto-update* option selected, and click



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

**OK** The **system\_wrapper.vhd** file will be created and added to the project (Figure 5.1.54).

- (b) In the sources panel, right-click again on **system.bd**, and select **Generate Output Products** and click **Generate** to generate the Implementation, Simulation and Synthesis files for the design. Select **Global** You can also click on **Generate Block Design** in the **Flow Navigator pane** to do the same.

**Note:** If the synthesis option is **Global**, only wrapper files are generated during the block design generation phase, and the design will be synthesized as a whole at the synthesis stage. If the synthesis option is **Out of context per IP** or **Out of context per Block design**, the wrapper of the IP or block design will be generated and synthesized during block design generation, and the generated netlists will be combined together at the synthesis stage. Double-click on the file to see the content in the Auxiliary pane.

- (c) Notice that the VHDL file is already Set As the Top module in the design, indicated by the icon in front of the **system\_wrapper.vhd**.
- (d) In the main menu Select **File > Export > Hardware** and click **Next**  
**Note:** Since we do not have any hardware in Programmable Logic (PL) there is no bitstream to generate, hence the **Include bitstream** option is not necessary at this time. Click **Next**.
- (e) Name the XSA file to **system\_wrapper** and specify the folder name where the project is created.

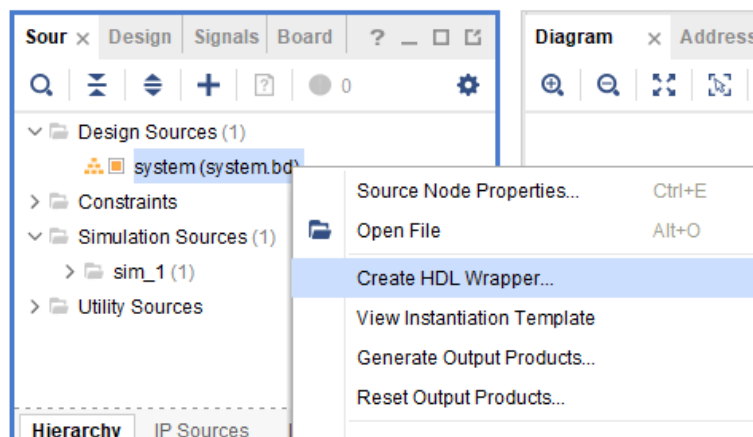


Figure 5.1.54: System Wrapper File Created

The XSA (Xilinx Support Archive) is a container file that contains all the information needed to build a platform for a users target device. One of the files here is the Hardware Hand-off File (HWH). This HWH file is created when the output products is ran on a Block Design (BD). Only the information in the Block Design (BD) will be contained in the HWH. The HWH



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

file is used by the software tools to abstract all the information needed to build a targeted application to a users device; such as the CPU (or CPUs), Buses, IP and the ports and pins used in the system such as interrupts.

## 5. Generate Hello World Application in Vitis

- (a) In the main menu select **Tools →Launch Vitis IDE**;
- (b) Specify the workspace and click **Launch** (Figure 5.1.55)
- (c) On the Vitis welcome screen click **Create Platform Project**
- (d) In the **New Platform Project** specify the *Platform project name*: **zed\_system**. Figure 5.1.57
- (e) In the **Create a new platform from hardware XSA**. Click **Browse** (5.1.29) to specify the XSA file. Xilinx hardware designs are created with the Vivado The XSA (Xilinx Support Archive) file was created when the project was exported from Vivado to Vitis. The XSA proprietary file format is used by the Vitis software platform to support the project in Vitis. Click **Next**
- (f) The Vitis IDE open with the platform project created (5.1.30). In the menu select **File -> New -> Application Project** for creating the application "Hello World". This will open the application project creation wizard (5.1.31). Click **Next**.
- (g) The next window is the domain selection window. for this first project we create standalone\_domain. Click **Next**. See 5.1.34.
- (h) In the new Application project examples select "Hello World" and click **Finish**.
- (i) In the file Explorer window right click on the **zed\_project**. Also expand the *hello\_world\_system* -> *hello\_world* -> *src* and double click on the **helloworld.c** .C file to opened in the editor window. Right click on the *hello\_word\_system* and select **Build project**.
- (j) If you worked with Zybo connect the board to your PC and power on.
- (k) If you worked with the Zedboard start **putty.exe** and login to "mazsola" to open a tunnel for the hardware server, where the Zedboard is connected. Also login to the computer "nyolcas".
- (l) For Zybo/Zed open RelTerm serial terminal. Select the UART port *CypresVirtualCom0* and setup to *Baud: 115200, Parity: none, Data Bits: 8, Stop Bits: 1* then connect the terminal (Figure 5.1.39).
- (m) In the Main menu select **Xilinx →XSCT Console**. Then in the console window type **connect** press **Enter**, then type **reset** and press **Enter**. In the XSCT window should be displayed something similar like in 5.1.38.
- (n) In the Explorer window right click on the application project *hello\_world\_system* select **Run as →Run Configurations**. Once the *Run Configurations* windows opens, twice click on **System Project Debug** to create an application debug **SystemDebugger\_hello\_world\_system**(Figure 5.1.66). Then click on **Run**







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

(o) “Hello World” appears on the RealTerm Terminal, as shown in the figure (5.1.67).

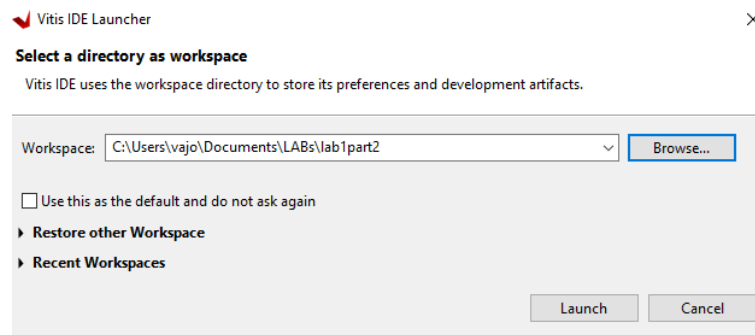


Figure 5.1.55: Lunch Vitis

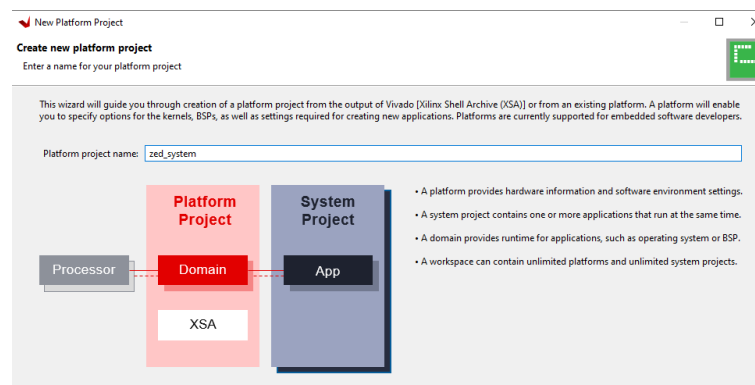


Figure 5.1.56: Create New Platform Project



**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

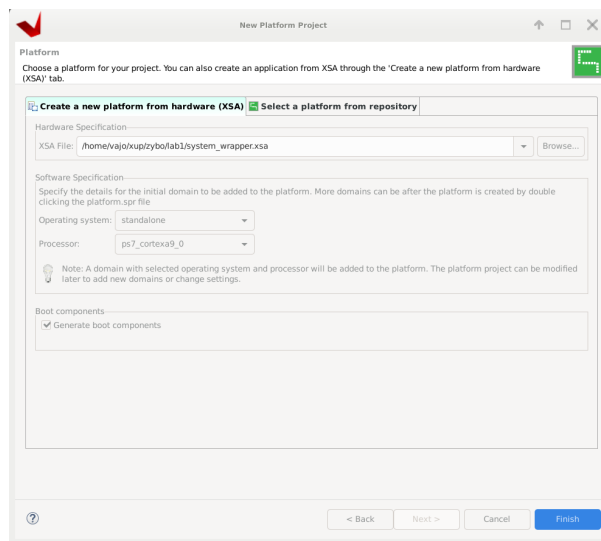


Figure 5.1.57: Create a new platform from hardware (XSA)

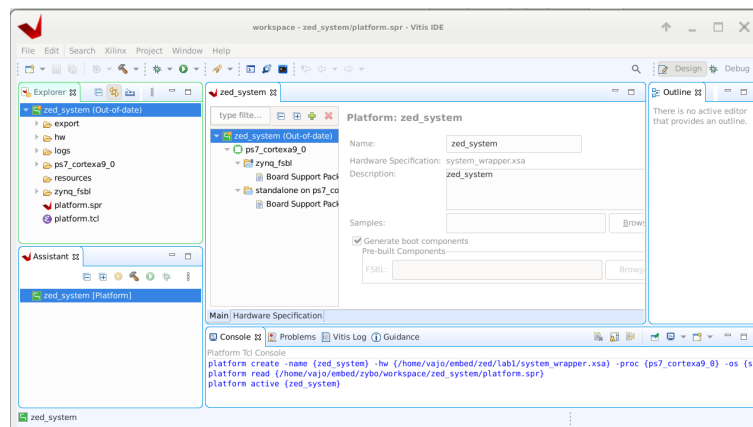


Figure 5.1.58: Vitis IDE with the platform project created





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

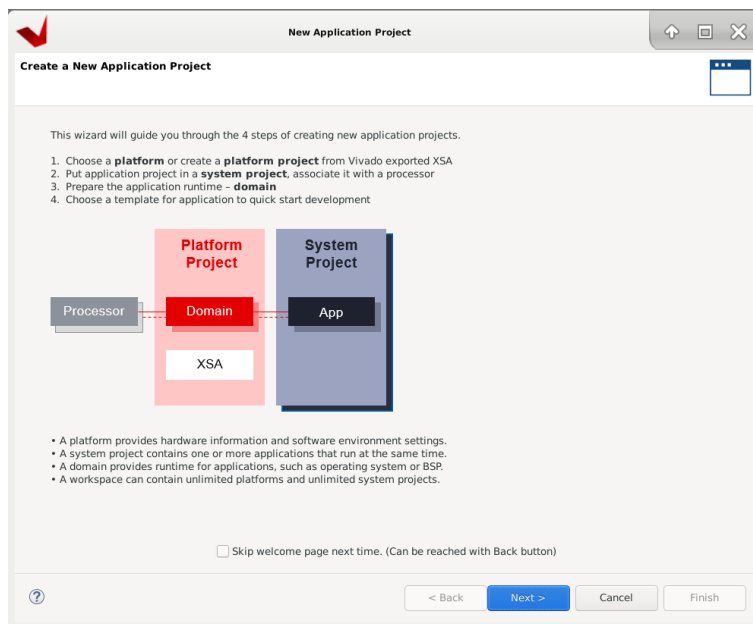


Figure 5.1.59: New Application Project Wizard

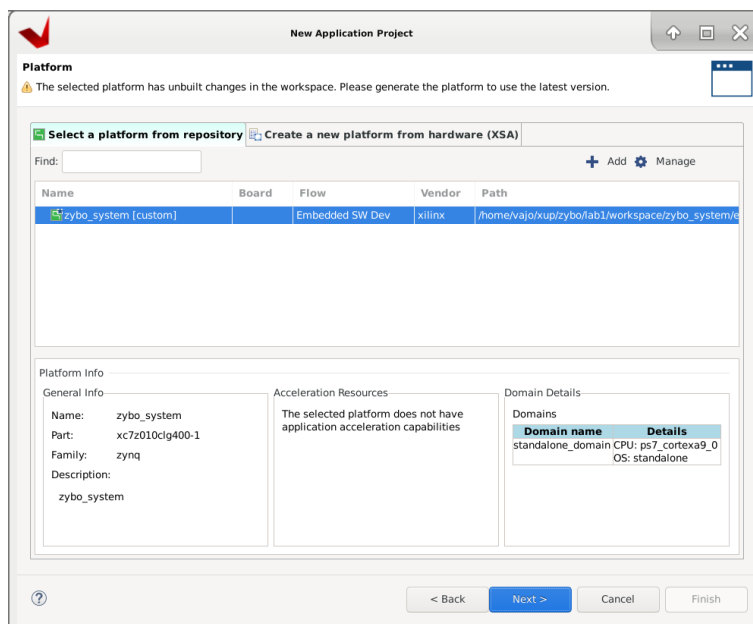


Figure 5.1.60: Select Platform



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

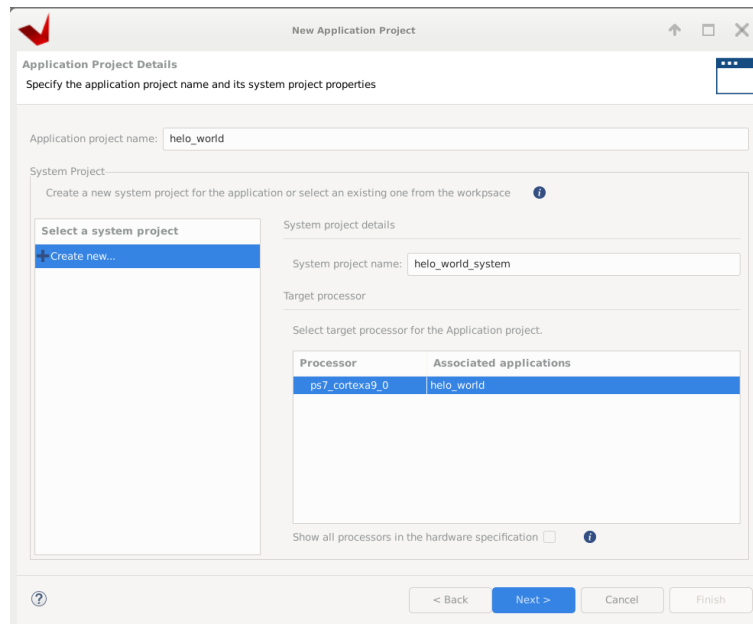


Figure 5.1.61: Specify the Application Project Name





**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

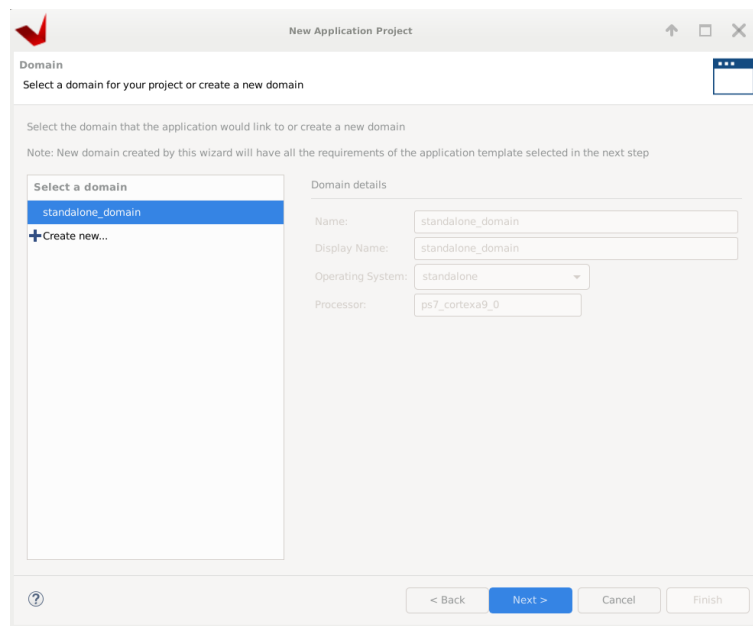


Figure 5.1.62: Select domain as standalone\_platform

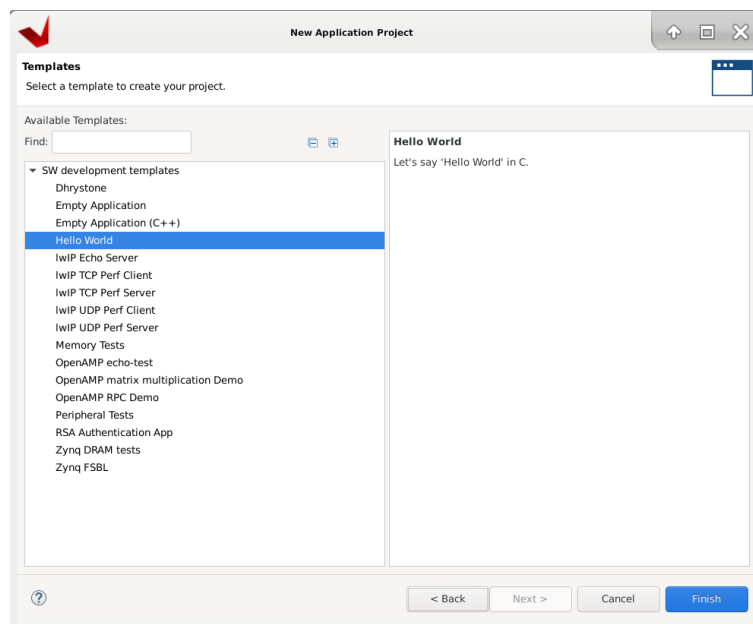


Figure 5.1.63: Select "Hello World" example project



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

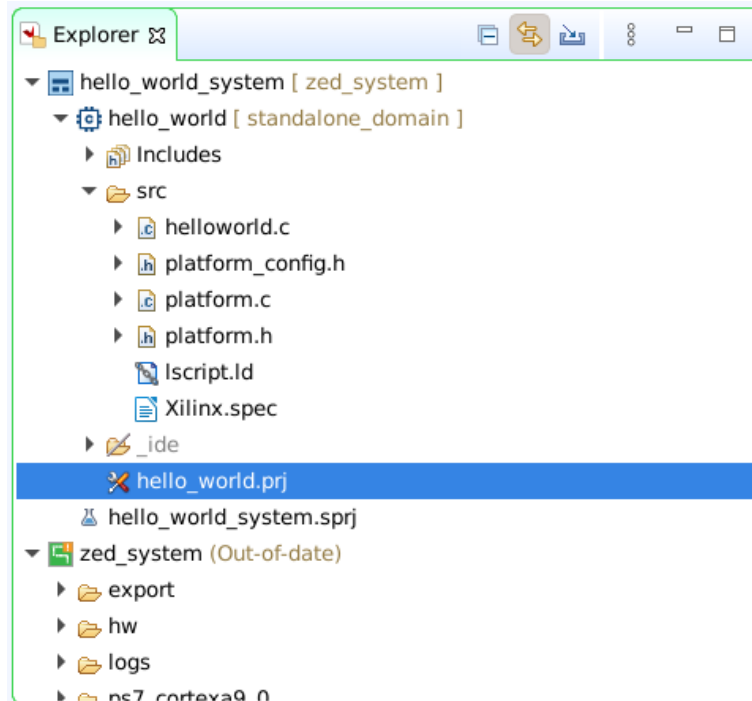


Figure 5.1.64: Application project "hello\_world" Explorer view

```

47
48 #include <stdio.h>
49 #include "platform.h"
50 #include "xil_printf.h"
51
52
53 int main()
54 {
55     init_platform();
56
57     print("Hello World\n\r");
58     print("Successfully ran Hello World application");
59     cleanup_platform();
60     return 0;
61 }
62

```

Figure 5.1.65: The "Hello World" program



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

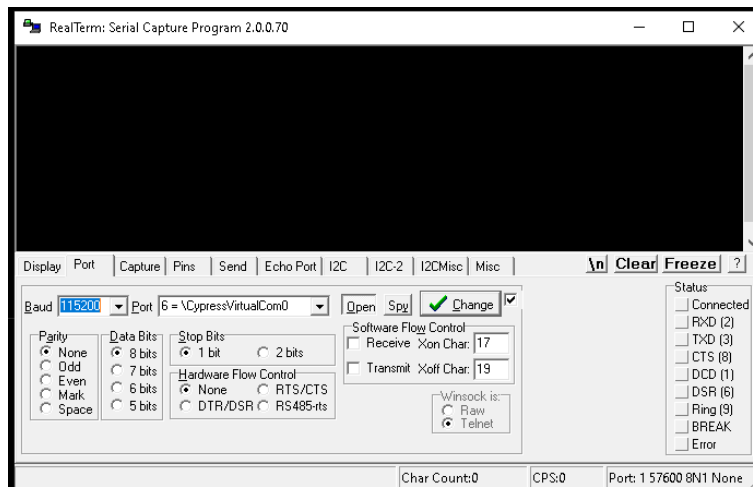


Figure 5.1.66: RealTerm terminal setup

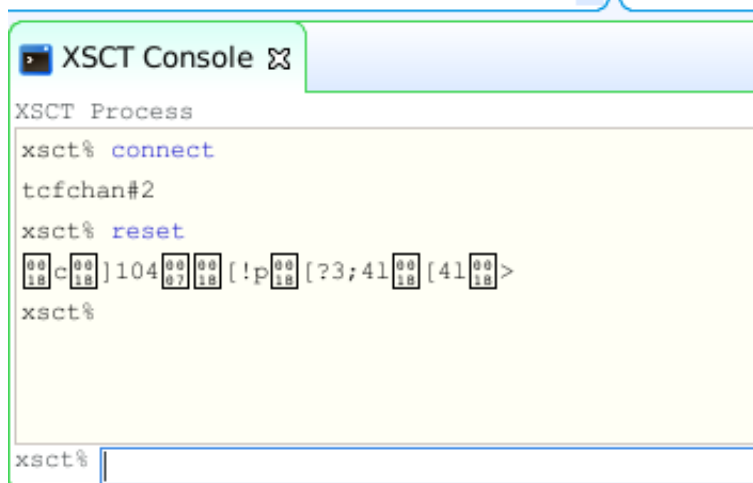


Figure 5.1.67: XSCt console connection view



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

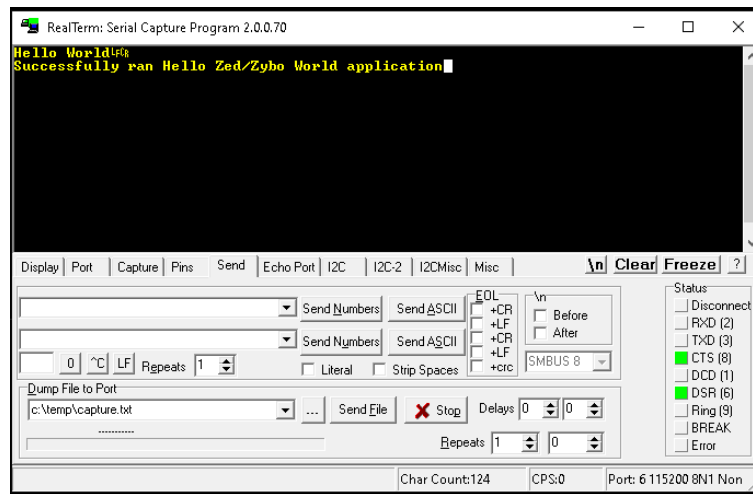


Figure 5.1.68: Caption

In the part 1 of the laboratory work 1 (called Lab1) we passed thru the steps, which have to be done for each embedded system project using Zynq architecture. True there were not explained the steps. Detailed information about the development process and “what to do and why to do” are explained in the Xilinx documentations (search for User Guides ug940, ug1165). For other information, tutorials and video tutorials start the DocNav (Document Navigator), installed together with the Vitis installation. Some of previously presented “Hello World” project material are based on Xilinx XUP (Xilinx University Program) and other tutorials such as ug1165 material [Xil18a].

## 5.2 Lab 3: Create your own IP

### Introduction

In the previous ?? it was demonstrated how to use the Xilinx IP library for embedded system design. This lab consists of extending the processing system with a custom peripheral (own IP).

### Objectives

Some times the user want to design its own peripheral for this is necessary to use the “IP wizard”. The AXI4Lite interface peripheral is created with the Vivado IP Packer. The reader became familiar with the IP design steps, which are demonstrated within a simple IP design. The IP read the switches from the board, and send it to the processing system. Figure 5.2.1 show the block schematic of the design.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

## Procedure

In the first step after Vivado was started the user will create its own Custom IP and export it to the IP library. Next in step 4 the Vivado project is created and the IP is imported. Finally the project is tested in Vitis. Figure 5.2.1 resumes the steps to follow.

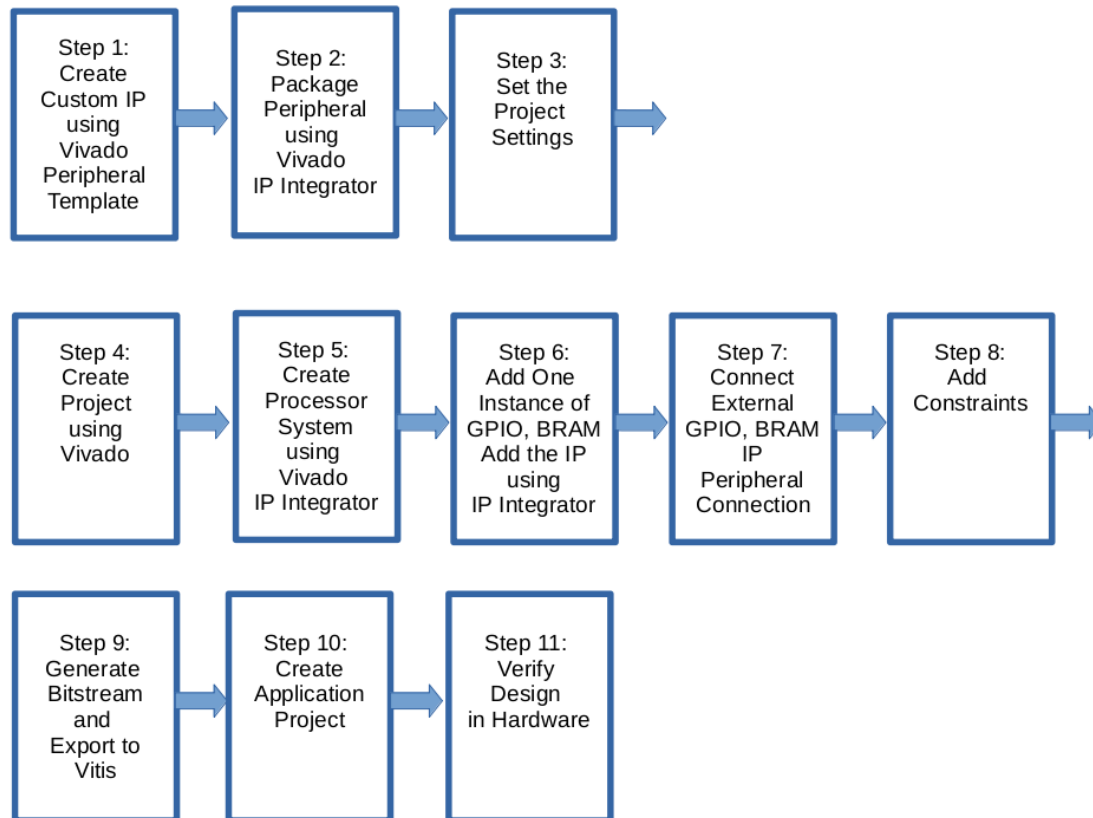


Figure 5.2.1: General flow for Lab 3

The lab is divided in two parts. The first part creates and pack the IP and the second part of the lab create the processing system extended with the GPIO, BRAM and IP.

### 5.2.1 Create and Manage IP Project

1. Launch Vivado and create a Custom IP using the Create and Package IP Wizard





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- Click **Manage IP** (See figure 5.2.2) and select *New IP Location* and click **Next** in the *New IP Location* window. See for details in figure 5.2.3
- Select part **xc7z010clg400-1** Zynq device for this project, but later compatibility for other devices will be added to packaged IP.
- Select VHDL as the *Target Language*, **Mixed** as the *Simulator Language* and for *IP location*, type **embed\sw\_ip** and click **Finish** (leave other settings as defaults and click **OK** if prompted to create the directory) (figure 5.2.3)

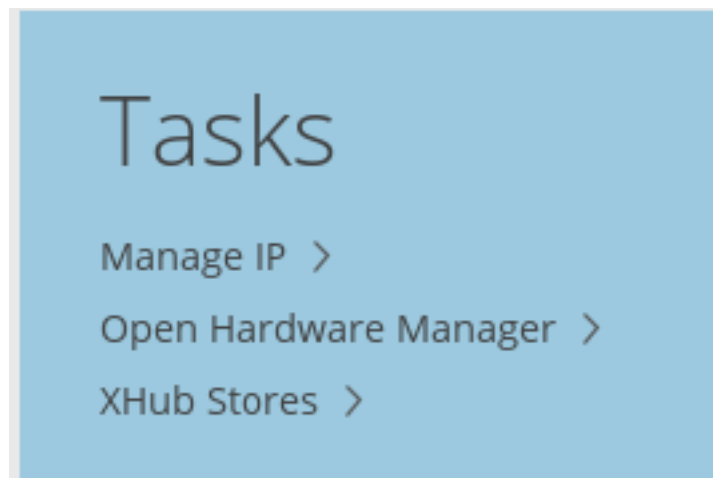


Figure 5.2.2: Manage IP

„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

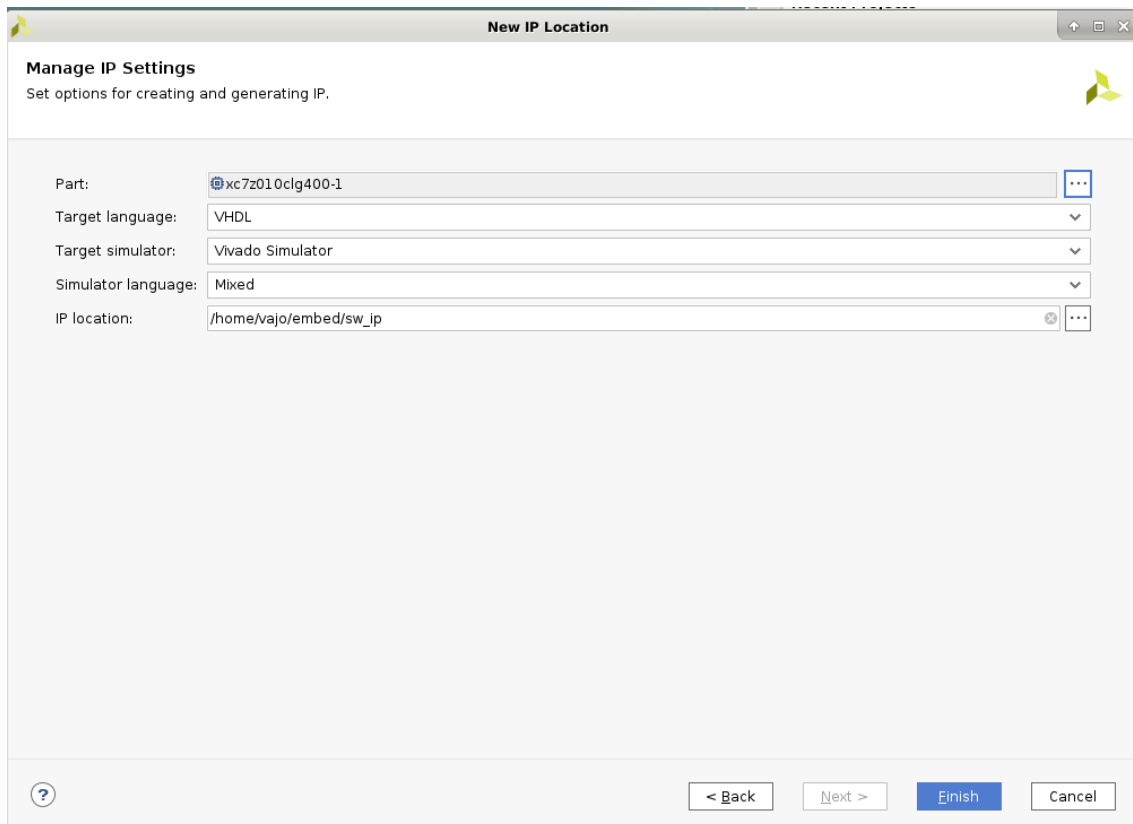


Figure 5.2.3: New IP location form

## 2. Run the Create and Package IP Wizard

- (a) Select **Tools => Create and Package IP**
- (b) In the window, click **Next**
- (c) Select **Create a new AXI4 peripheral**, and click **Next**
- (d) Fill in the details for the IP: **Name: sw\_ip; Display Name: sw\_ip\_v1\_0**; Fill in a *Description, Vendor Name and URL*. Click **Next**. (See figure 5.2.4) Can be observed the Peripheral AXI interface details. Leave it as it is. For this lab is important that the peripheral will have 4 registers. Click **Next**.
- (e) Select **Edit IP**, overview of the peripheral generation summary (figure 5.2.6) . Click **Finish**



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

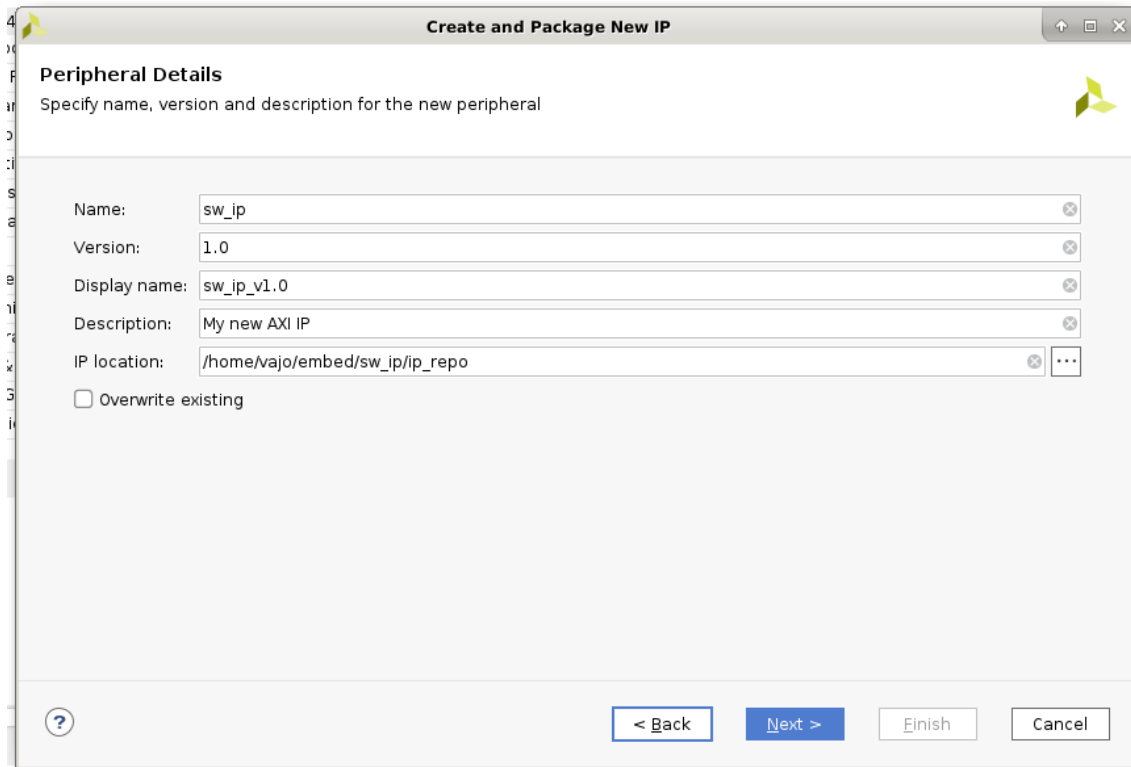


Figure 5.2.4: Create new IP

- (f) In figure 5.2.4 the new IP parameters can be edited. Such as the IP name (for example you can change it to S\_Axi) the interface type which now is AXI Lite but can be change to AXI Full or AXI Stream bus, which are different variants of the AXI bus as mentioned in 2. The interface should be a slave interface since the master will be the processing system. Data width is 32 bits as the PS is also 32 bits, and the number of user registers at this time is 4 (in this lab we will use only one register). Click **Next**.
- (g) As shown in 5.2.6 choose edit IP and Click **Finish**.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

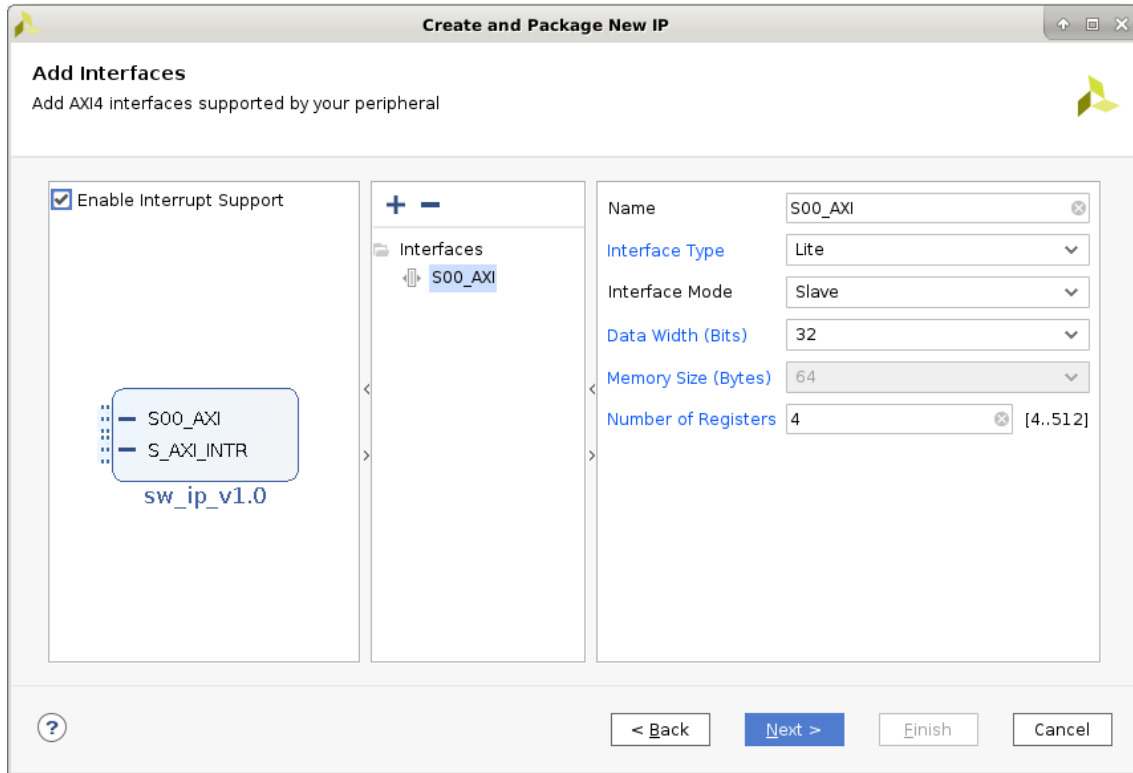


Figure 5.2.5: Add Interfaces screen





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

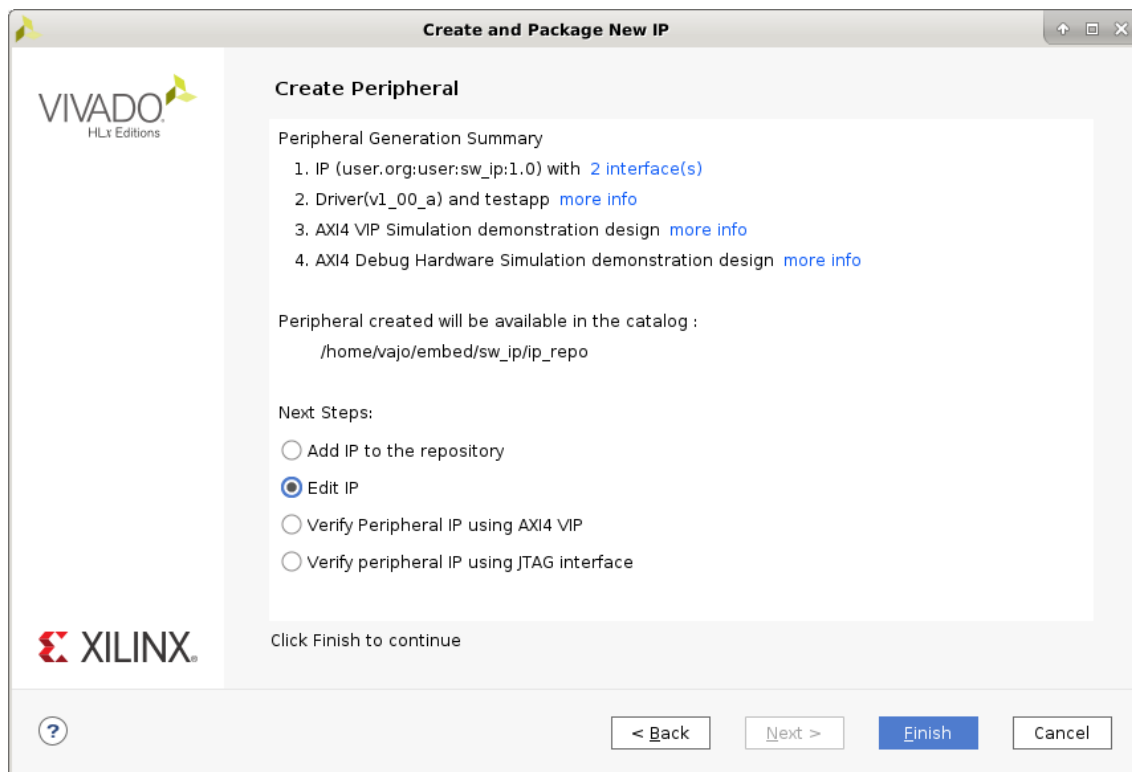


Figure 5.2.6: Create peripheral overview





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

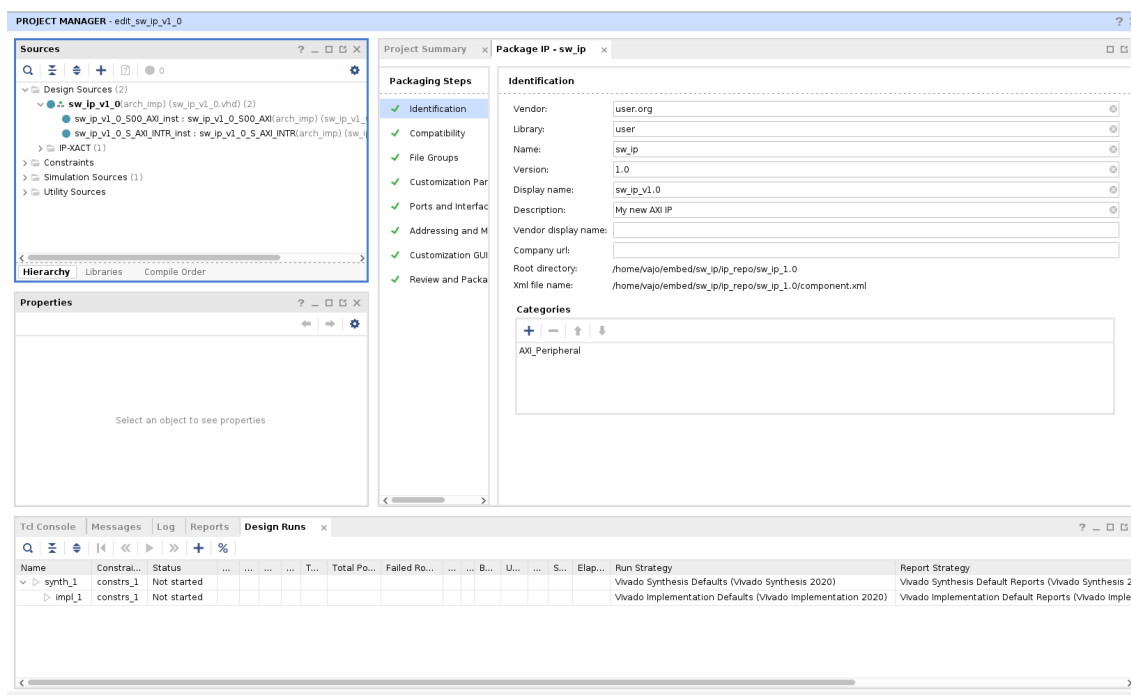


Figure 5.2.7: Edit IP general view

### 3. Create the interface to the Switches

- In the sources panel, double-click the **sw\_ip\_v1\_0.vhd** file. (See 5.2.7)  
This file contains the VHDL code for the interface(s) selected above. The top level file contains a module which implements the AXI Lite interfacing logic, and an example design to write to and read from the number of registers specified above. This template is used as a basis for creating custom IP.
- In the top level design the new input port **sw** – Switch will be created. The AXI read data in the sub-module will be connected to the external **sw** port. Scroll down to line 7 where the user parameters space is provided in the *generic* part of the *entity*.
- Add the parameter of the **sw\_width**, which is different for *Zybo* and *Zed*. See figure 5.2.8. On the *Zybo* board there are 4 switches, while on the *Zedboard* there are 8 switches.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

```

4 | :
5 | entity sw_ip_v1_0 is
6 |     generic (
7 |         -- Users to add parameters here
8 |         sw_width      : integer      := 4;
9 |         -- User parameters ends
10 |        -- Do not modify the parameters beyond this line
11 |
12 |
13 |        -- Parameters of Axi Slave Bus Interface S00_AXI
14 |        C_S00_AXI_DATA_WIDTH  : integer := 32;
15 |        C_S00_AXI_ADDR_WIDTH  : integer := 4;
16 |
17 |        -- Parameters of Axi Slave Bus Interface S_AXI_INTR
18 |        C_S_AXI_INTR_DATA_WIDTH : integer := 32;
19 |        C_S_AXI_INTR_ADDR_WIDTH : integer := 5;
20 |        C_NUM_OF_INTR          : integer := 1;
21 |        C_INTR_SENSITIVITY     : std_logic_vector := x"FFFFFFFF";
22 |        C_INTR_ACTIVE_STATE    : std_logic_vector := x"FFFFFFFF";
23 |        C_IRQ_SENSITIVITY      : integer := 1;
24 |        C_IRQ_ACTIVE_STATE     : integer := 1;
25 |     );
26 |     port (
27 |         -- Users to add ports here
28 |         sw      : in std_logic_vector(sw_width-1 downto 0);
29 |         -- User ports ends
30 |         -- Do not modify the ports beyond this line
    
```

Figure 5.2.8: Declaring users port in the lower-level module for the Zybo

- (d) Scroll down the VHDL code to line 54 where the `sw_ip_v1_0_S00_AXI` component is declared and introduce the same generic (`sw_width`) and port `sw` in the component declaration section. See figure 5.2.9







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

```

50 |
51 | architecture arch_imp of sw_ip_v1_0 is
52 |
53 |     -- component declaration
54 |     component sw_ip_v1_0_S00_AXI is
55 |         generic (
56 |             -- sw zybo = 4 zed = 8
57 |             sw_width : integer := 4;
58 |             C_S_AXI_DATA_WIDTH : integer := 32;
59 |             C_S_AXI_ADDR_WIDTH : integer := 4
60 |         );
61 |         port (
62 |             sw : in std_logic_vector (sw_width-1 downto 0);
63 |             S_AXI_ACLK : in std_logic;
    
```

Figure 5.2.9: Adding user generics and port to the component declaration section

- (e) Scroll down the VHDL code to line 89 where the *AXI BUS Interface S00\_AXI* component is instantiated. and instantiate the generic for *sw\_width* and connect the *sw* to the *sw* port of the *sw\_ip\_v1\_0\_S00\_AXI* VHDL component (figure 5.2.10).

```

89 | -- Instantiation of Axi Bus Interface S00_AXI
90 | sw_ip_v1_0_S00_AXI_inst : sw_ip_v1_0_S00_AXI
91 |     generic map (
92 |         sw_width => sw_width,
93 |         C_S_AXI_DATA_WIDTH => C_S00_AXI_DATA_WIDTH,
94 |         C_S_AXI_ADDR_WIDTH => C_S00_AXI_ADDR_WIDTH
95 |     )
96 |     port map (
97 |         sw => sw,
98 |         S_AXI_ACLK => s00_axi_aclk,
    
```

Figure 5.2.10: Completing the component instantiate part.

- (f) Double click on the *sw\_ip\_v1\_0\_S00\_AXI.vhd* file to implement the IP. First add in the entity section the generic and the port as previously, such as in (figure 5.2.12) line 5-21.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```
4 :
5 ⊖ entity sw_ip_v1_0_S00_AXI is
6   generic (
7     -- Users to add parameters here
8     sw_width : integer := 4;
9 ⊖     -- User parameters ends
10 :    -- Do not modify the parameters beyond this line
11 :
12 ⊖     -- Width of S_AXI data bus
13 :    C_S_AXI_DATA_WIDTH : integer := 32;
14 :    -- Width of S_AXI address bus
15 :    C_S_AXI_ADDR_WIDTH : integer := 4
16 :  );
17 :  port (
18 ⊖     -- Users to add ports here
19 ⊖     -- sw is input
20 :    sw : in std_logic_vector (sw_width-1 downto 0);
21 ⊖     -- User ports ends
22 :    -- Do not modify the ports beyond this line
```

Figure 5.2.11: Edit entity `sw_ip_v1_0_S00_AXI` generic and port declaration

- (g) One can analyze the IP template for write and read process. At the file end can be edited the *user logic part* of the IP. But since in this lab we only read the switches this can be done in the slave register read section of the IP as in figure 5.2.12 line 357, where the switches are read directly in the `reg_data_out(sw_width-1 downto 0)`, which are the lower bits of the AXI read data register.

```
345 ⊖ -- Implement memory mapped register select and read logic generation
346 : -- Slave register read enable is asserted when valid address is available
347 ⊖ -- and the slave is ready to accept the read address.
348 : slv_reg_rden <= axi_arready and S_AXI_ARVALID and (not axi_rvalid) ;
349 :
350 ⊖ process (slv_reg0, slv_reg1, slv_reg2, slv_reg3, axi_araddr, S_AXI_ARESETN, slv_reg_rden)
351 : variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
352 : begin
353 :   -- Address decoding for reading registers
354 :   loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
355 ⊖   case loc_addr is
356 ⊖     when b"00" =>
357 ⊖     reg_data_out <= slv_reg0(C_S_AXI_DATA_WIDTH-1 downto sw_width) & sw(sw_width-1 downto 0);
358 ⊖     when b"01" =>
359 ⊖     reg_data_out <= slv_reg1;
360 ⊖     when b"10" =>
361 ⊖     reg_data_out <= slv_reg2;
362 ⊖     when b"11" =>
363 ⊖     reg_data_out <= slv_reg3;
364 ⊖     when others =>
365 ⊖     reg_data_out <= (others => '0');
366 ⊖   end case;
367 ⊖ end process;
```

Figure 5.2.12: Peripheral read data from Switches process

- (h) Click **Run Synthesis** and **Save** if prompted. (This is to check the design synthesizes correctly)





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

before packaging the IP. If this was your own design, you would simulate it and verify functionality before proceeding) Check the Messages tab for any errors and correct if necessary before moving to the next step When Synthesis completes successfully, click **Cancel**.

#### 4. Package the IP

The package IP steps can be followed in the figures from 5.2.13 – to 5.2.22

- (a) First in the *Identification* window you can edit the IP Identification details.
- (b) Such as in the [Xil] documents we change the in the IP Catalog the `sw_ip` category, where the ip will be included. "For the IP to appear in the IP catalog in particular categories, the IP must be configured to be part of those categories. To change which categories the IP will appear in the IP catalog click the browse box on the *Categories* line. This opens the Choose IP Categories window. For the purpose of this exercise, uncheck the **AXI Peripheral** box and check the **Basic Elements** and click **OK**."
- (c) In the *Compatibility* window shows the different Xilinx families for which the IP was created. You can add other Xilinx devices. In this case the *Zynq* devices are already set.
- (d) In the **File Groups** window you can see that some changes are not merged. Click on **Merge changes from the File Groups Wizard** to merge the changes. This will update the IP Packager with the changes that were made to the IP.
- (e) In the **IP Customization Parameters** window click again **Merge changes from IP Customization Parameters Wizard**. Notice that in the *IP Ports and Interfaces* window now shows the user created `sw` port.
- (f) Select **Review and Package** window, notice the path where the IP will be created. Click **Re-Package IP**. The project is packed and closed.
- (g) In the original Vivado window click **File Menu => Close Project**





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

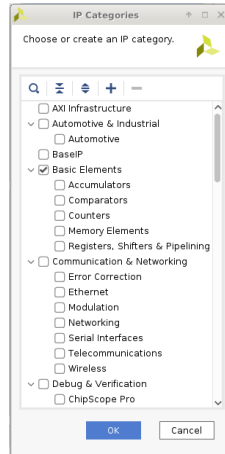


Figure 5.2.14: Choose or Create an IP category window

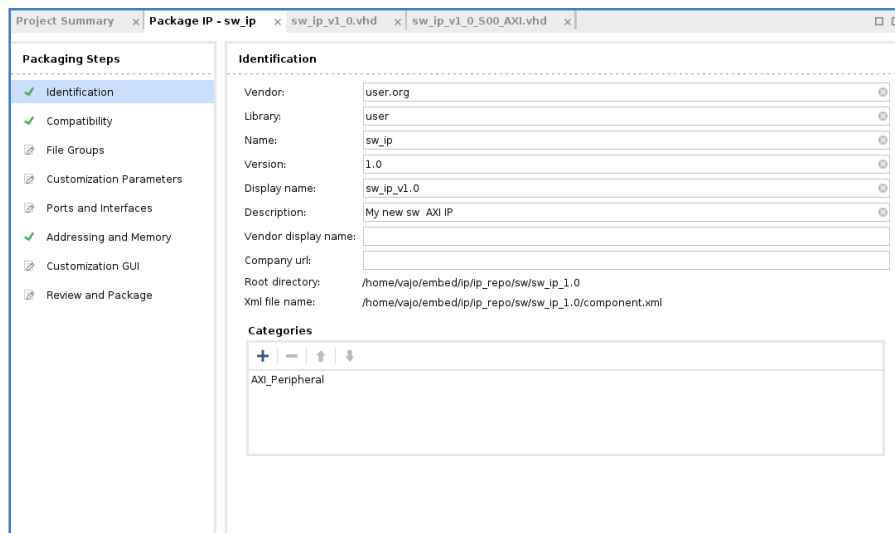


Figure 5.2.13: Pack IP – sw\_ip window



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

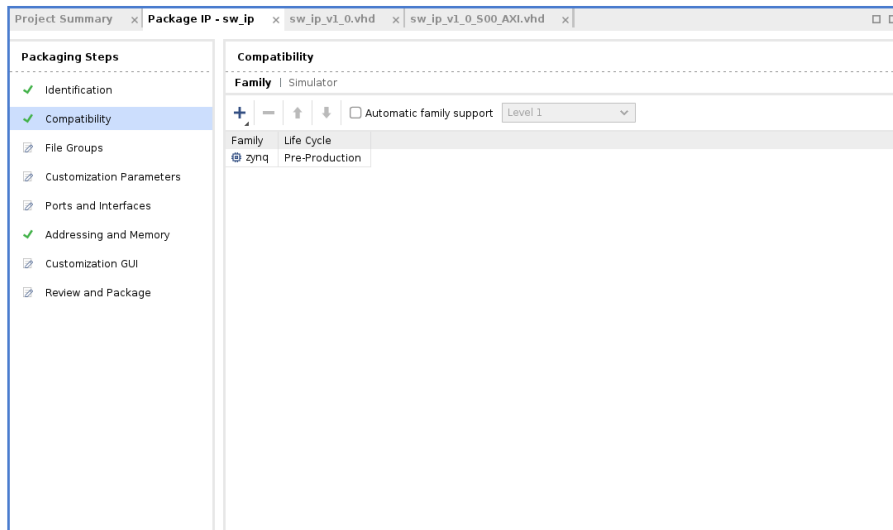


Figure 5.2.15: Compatibility window

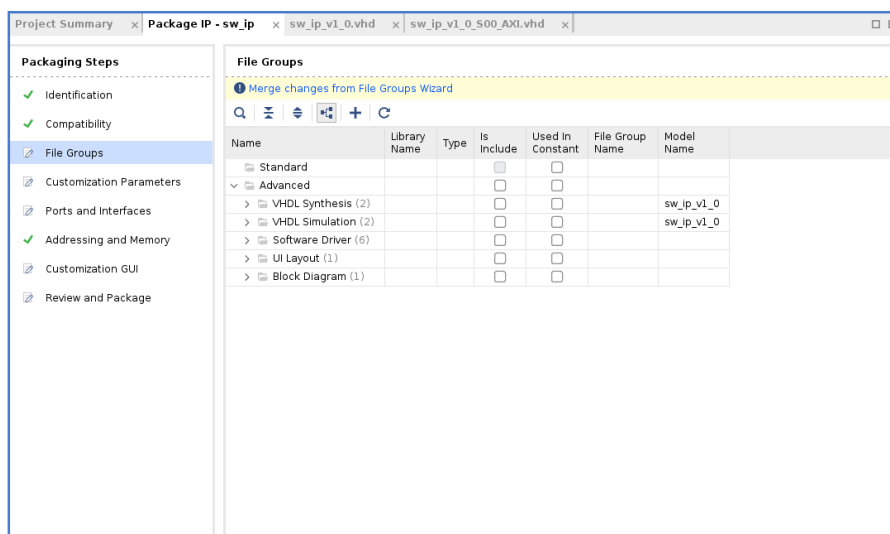


Figure 5.2.16: Merge changes from the File Groups Wizard Window



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

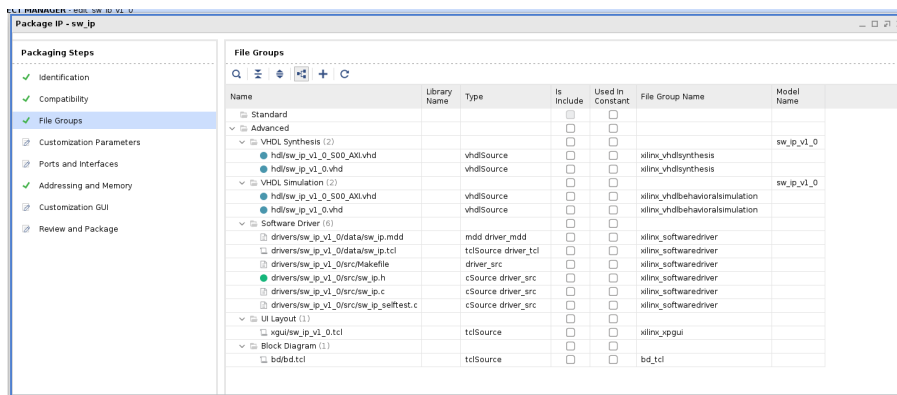


Figure 5.2.17: File Groups Window

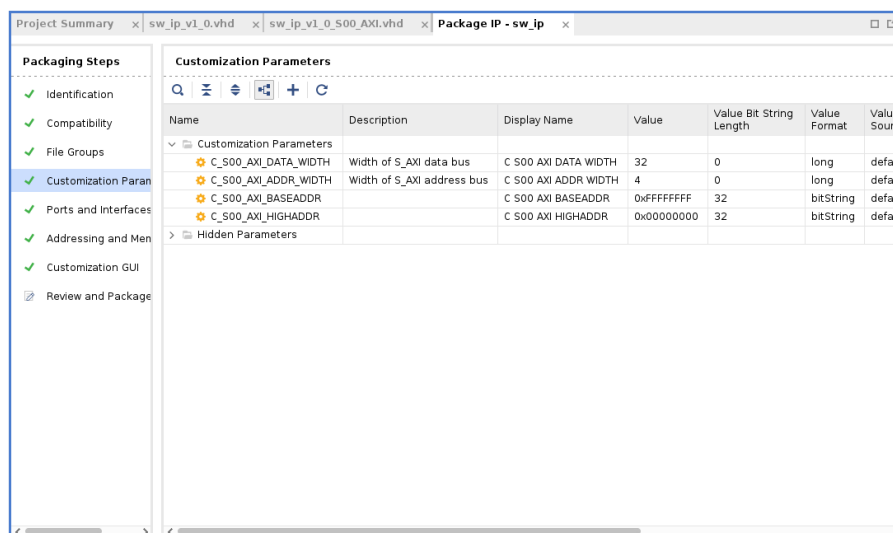


Figure 5.2.18: Customization Parameters Window



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

Ports and Interfaces							
Name	Interface Mode	Enablement Dependency	Direction	Driver Value	Size Left	Size Right	Size Left Dependency
> S00_AXI	slave						
> Clock and Reset Signals							
sw			in		3	0	(sw_width - 1)

Figure 5.2.20: User Port imported

Customization Parameters							
Name	Description	Display Name	Value	Value Bit String Length	Value Format	Value Source	
C_S00_AXI_DATA_WIDTH	Width of S_AXI data bus	C S00 AXI DATA WIDTH	32	0	long	default	
C_S00_AXI_ADDR_WIDTH	Width of S_AXI address bus	C S00 AXI ADDR WIDTH	4	0	long	default	
C_S00_AXI_BASEADDR		C S00 AXI BASEADDR	0xFFFFFFFF	32	bitString	default	
C_S00_AXI_HIGHADDR		C S00 AXI HIGHADDR	0x00000000	32	bitString	default	
sw_width		Sw Width	4	0	long	default	

Figure 5.2.19: User parameter imported

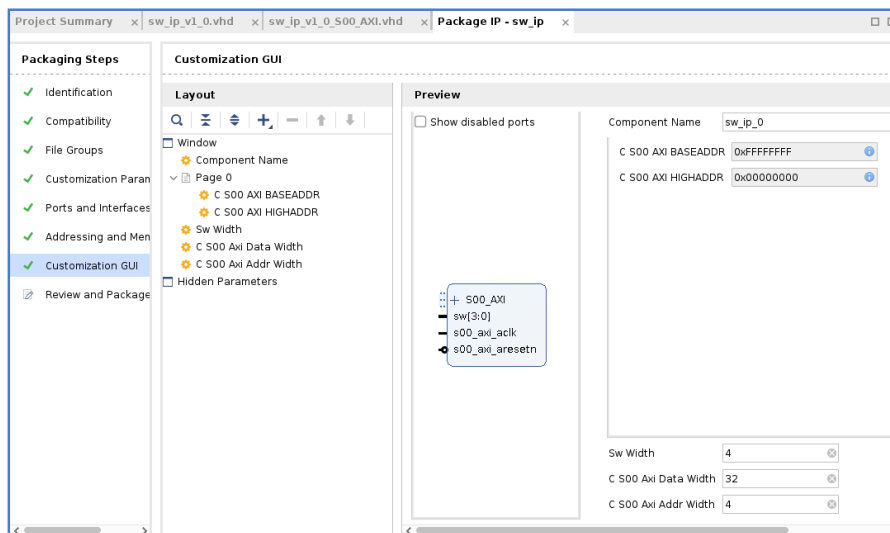


Figure 5.2.21: Customization Graphical User Interface GUI Window



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

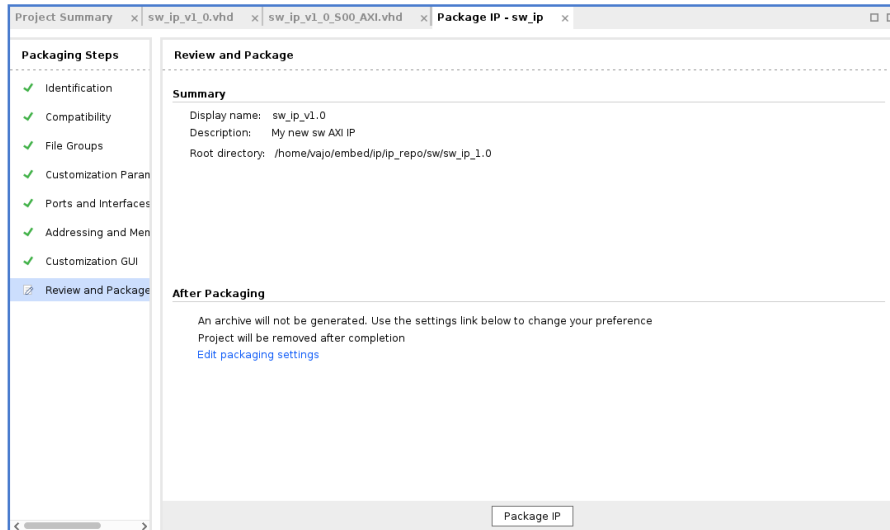


Figure 5.2.22: Review and Package Window

## 5.2.2 Create the Vivado Project with User Designed IP

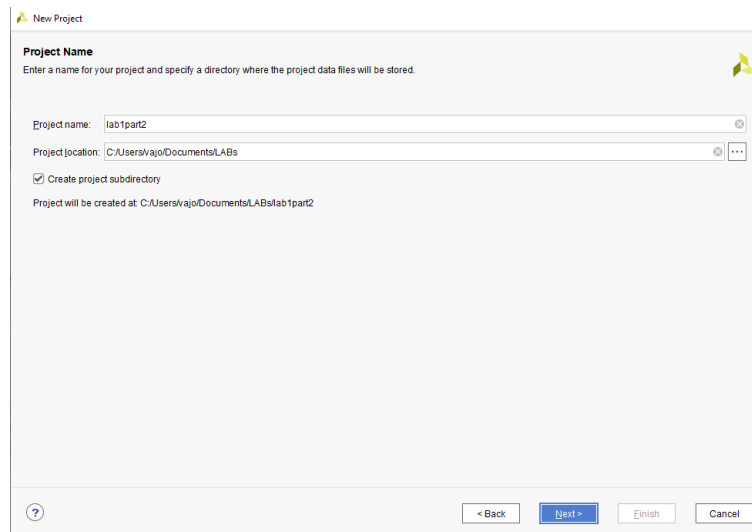
### Create New Project or continue with lab2 project

1. Launch Vivado and create an empty project targeting the ZedBoard, using the VHDL language or use the project lab2 presented in ?? and save the project as lab3.
  - (a) Open Vivado by selecting: **Start => All Programs => Xilinx Design Tools => Vivado 2020.2 => Vivado 2020.2**
  - (b) Open Vivado by starting the *xilinx.sh -v* script if you are logged in the lab. Select **Vivado 2020.2** from the list.
  - (c) Click **Create New Project** to start the wizard. You will see the *Create a New Vivado Project* dialog box. Click **Next**.
  - (d) Click in the field of the **Project Location** and type */embed/zed/*. Enter **lab3** in the **Project Name** field. Make sure that the **Create project subdirectory** box is checked. Click **Next**. See figure 5.2.23.
  - (e) In the **Project Type** form select **RTL Project**. Make sure that **Do not specify sources this time** box is checked and click **Next**. See figure 5.2.24.
  - (f) For the **Default Part** window choose **Boards** and select **Zybo or Zedboard** click **Next**.
  - (g) For **Zedboard** choose *em.avnet.com*, then select the board in the window or simply type in the search box the desired board and revision. The online Zedboard is rev D. See figure 5.2.25. Then click **Next** and finally click **Finish**.





**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

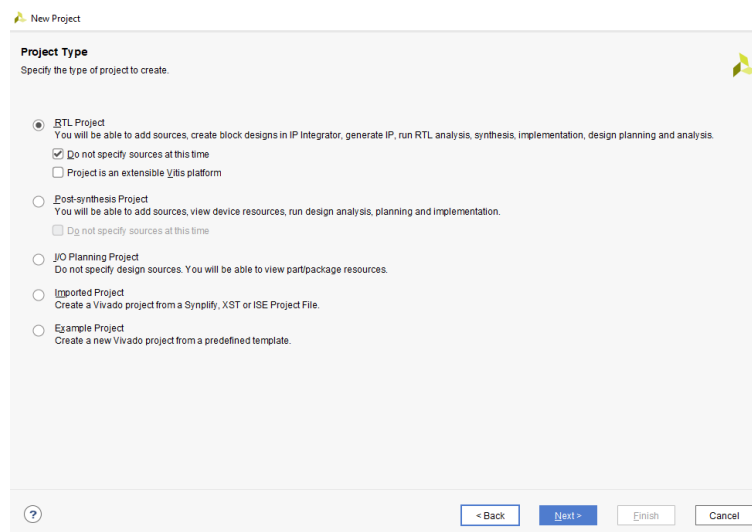


The screenshot shows a 'New Project' dialog box with the following fields and options:

- Project Name:** lab1part2
- Project location:** C:/Users/vajo/Documents/LABS
- Create project subdirectory
- Project will be created at: C:/Users/vajo/Documents/LABS/lab1part2

Buttons at the bottom: < Back, Next >, Finish, Cancel.

Figure 5.2.23: Project Name Entry



The screenshot shows a 'New Project' dialog box with the following options:

- 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  
 Project is an extensible Vitis platform
- 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
- IO 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.

Buttons at the bottom: < Back, Next >, Finish, Cancel.

Figure 5.2.24: RTL Project selection



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

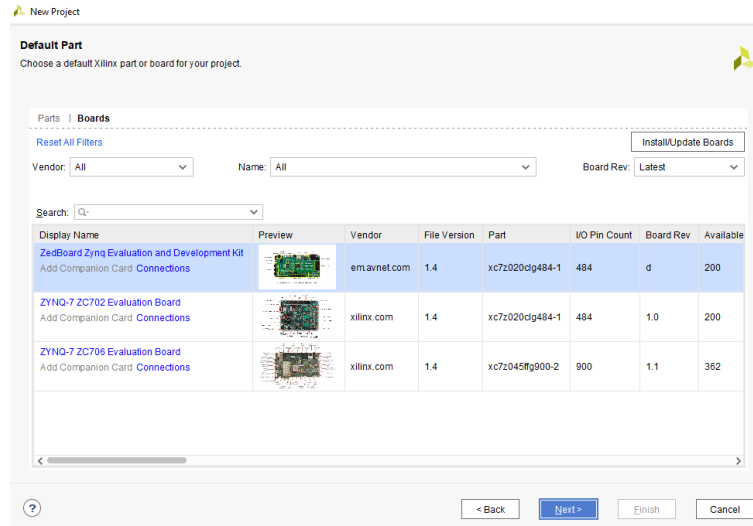


Figure 5.2.25: Board selection window for Zedboard

### Modify Project settings

1. Click **Project Settings** in the Flow Navigator pane.
2. **Select IP** in the left pane of the Project Settings form. **Click** to open.
3. Select **Repository** and in the **IP Repositories** windows click **+** to add the user IP repository where the **sw\_ip** was created. Select the directory where the sw IP was created namely: **/embed/ip/ip\_repo/sw**. See figure 5.2.26.

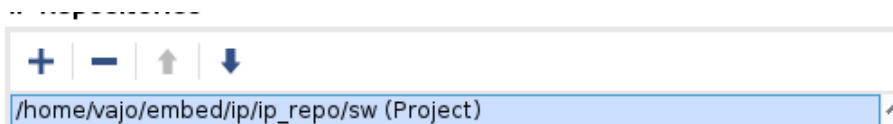


Figure 5.2.26: Set up IP repository

### Create the System Block Design Using IP Integrator

1. Use the IP Integrator to create a new Block Design, add the ZYNQ processing system block, and configure the processing system.
2. In the Flow Navigator, click **Create Block Design** under **IP Integrator**.
3. Enter **system** for the design name and click **OK**. See figure 5.2.27.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

4. IP from the catalog can be added in different ways. Click + icon in the empty block diagram workspace or **Add IP** icon in the block diagram side bar or press **Ctrl + I**, or right-click anywhere in the Diagram workspace and select **Add IP**. See figure 5.2.28
5. Once the IP Catalog is open, type “z” into the Search bar, find and double click on **ZYNQ7 Processing System** entry, or click on the entry and hit the Enter key to add it to the design (See 5.2.29)
6. Notice the message at the top of the Diagram window that *Designer Assistance Available*. Click **Run Block Automation** and select `/processing_system7_0`. See figure 5.2.30
7. In the *Run Block Automation* window, leave the default settings, including **Apply Board Preset** checked, and click **OK**. See figure 5.2.31. Once Block Automation has been complete, notice that ports have been automatically added for the DDR and Fixed IO, and some additional ports are now visible. The imported configuration for the Zynq related to the Zybo/Zedboard board has been applied which will now be modified. See figure 5.2.32
8. Double-click on the added block to open its Customization window. Notice now the Customization window shows selected peripherals (with tick marks). This is the default configuration for the board applied by the block automation. See figure 5.2.33
9. A block diagram of the Zynq should now be open again, showing various configurable blocks of the *Processing System*. At this stage, the designer can select or deselect various configurable blocks (highlighted in green) and change the system configuration.
10. Configure the processing block with UART0 and GPIO MIO peripheral enabled.
  - (a) Only the UART0 and GPIO MIO is required for this lab, so all other peripherals will be deselected.
  - (b) Click on one of the peripherals (in green) in the *I/O Peripherals* block, or **select the MIO Configuration** tab on the left to open the configuration form. The procedures are resumed in Figure 5.2.33 and Figure 5.2.34
    - i. Expand **I/O peripherals** if necessary, and ensure all the following I/O peripherals are deselected except **GPIO MIO and UART0**
    - ii. Expand **Memory Interfaces** to deselect **Quad SPI Flash**.
    - iii. Expand **Application Processor Unit** to disable **Timer 0**.
    - iv. Select the **PS-PL Configuration** tab on the left. Expand **AXI Non Secure Enablement => GP Master AXI interface** and select **M AXI GP0 interface** if not already selected(!).
    - v. Expand **General => Enable Clock Resets** and deselect the **FCLK\_RESET0\_N** option.
    - vi. Select the **Clock Configuration** tab on the left. Expand the **PL Fabric Clocks** and select the **FCLK\_CLK0** option and click **OK**
    - vii. In the block design connect **FCLK\_CLK0** with `textbfM_AXI_GP0_ACLK`. Hover your mouse over the connector port until the pencil button appears then connect the two signals





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- viii. Click on the **Validate Design** button and make sure that there are no errors.
- (c) Enable **AXI\_M\_GP0** interface, **FCLK\_RESETO\_N**, and **FCLK\_CLK0** ports if already not enabled [Xil]. Click **OK** to close the Zynq Processing System configuration window.

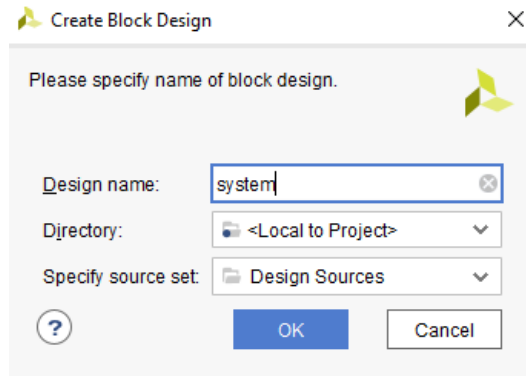


Figure 5.2.27: Create New Block Diagram

This design is empty. Press the **+** button to add IP.



Figure 5.2.28: Add IP to Block Diagram possibilities

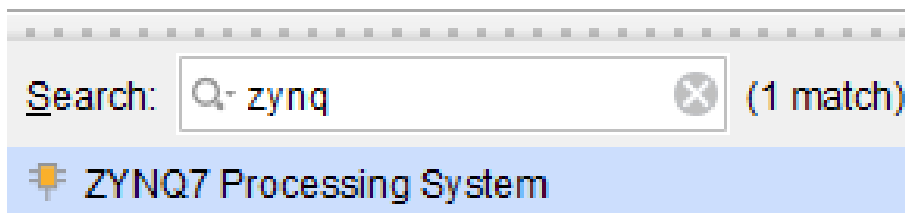


Figure 5.2.29: Add IP ZYNQ7 Processing System



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

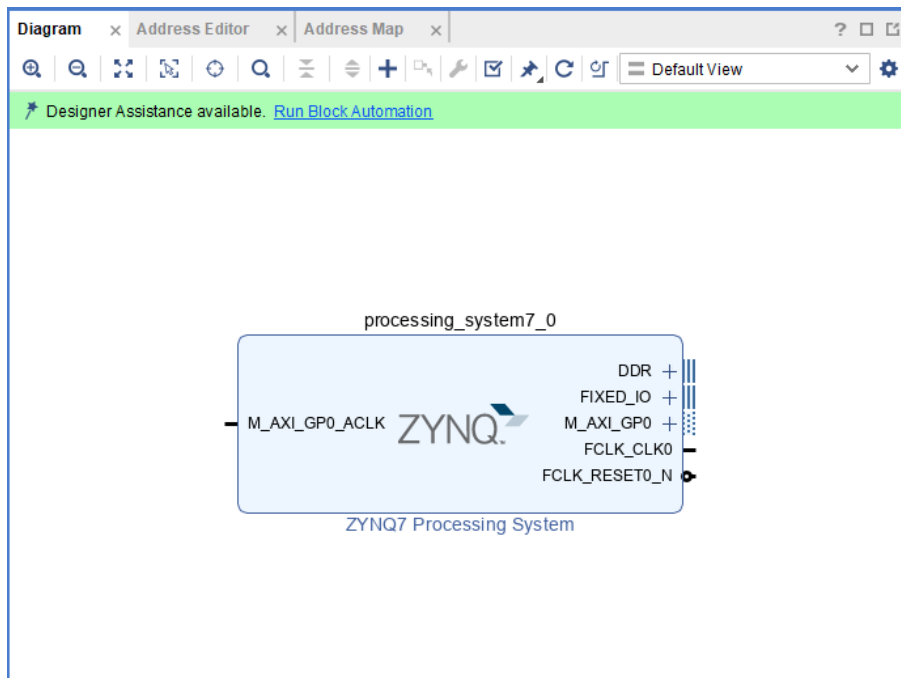


Figure 5.2.30: Run Block Automation Process





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

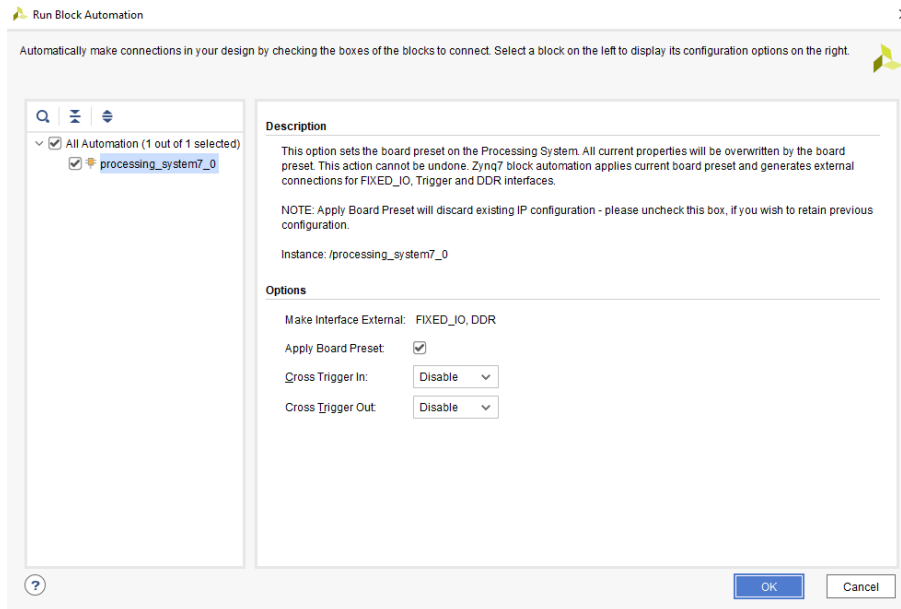


Figure 5.2.31: Selecting connections on Run Block Automation Process Settings

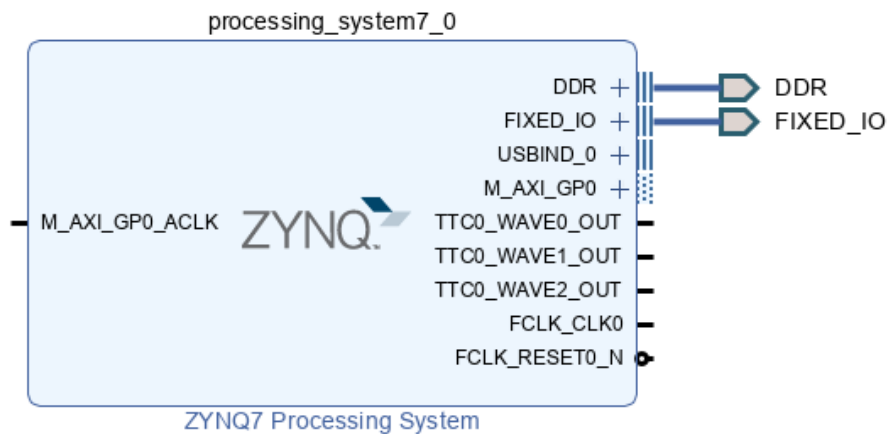


Figure 5.2.32: Schematic of the ZYNQ connected to DDR and FIXED\_IO



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

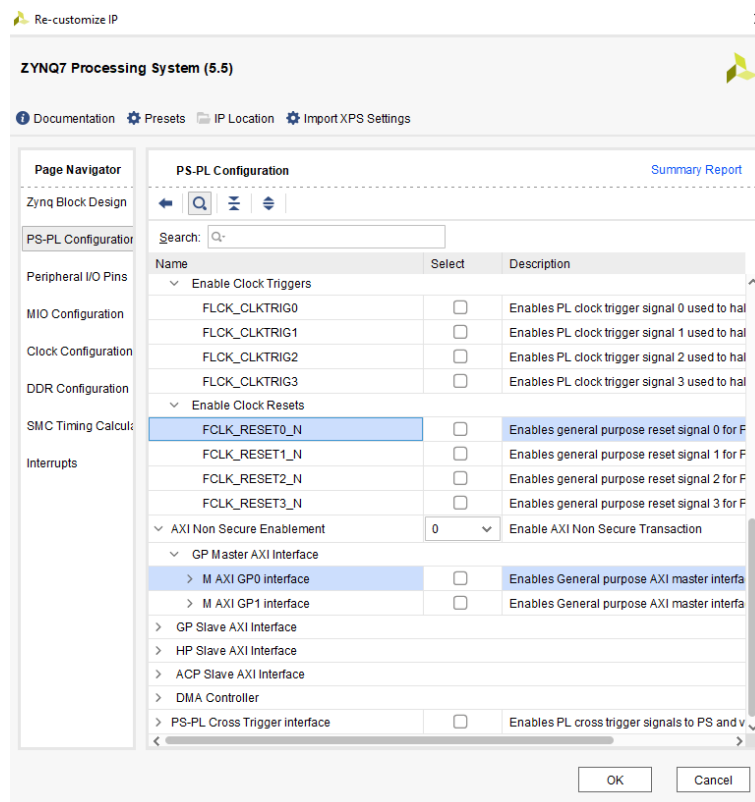


Figure 5.2.33: ZYNQ 7 IP Processing System Settings





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

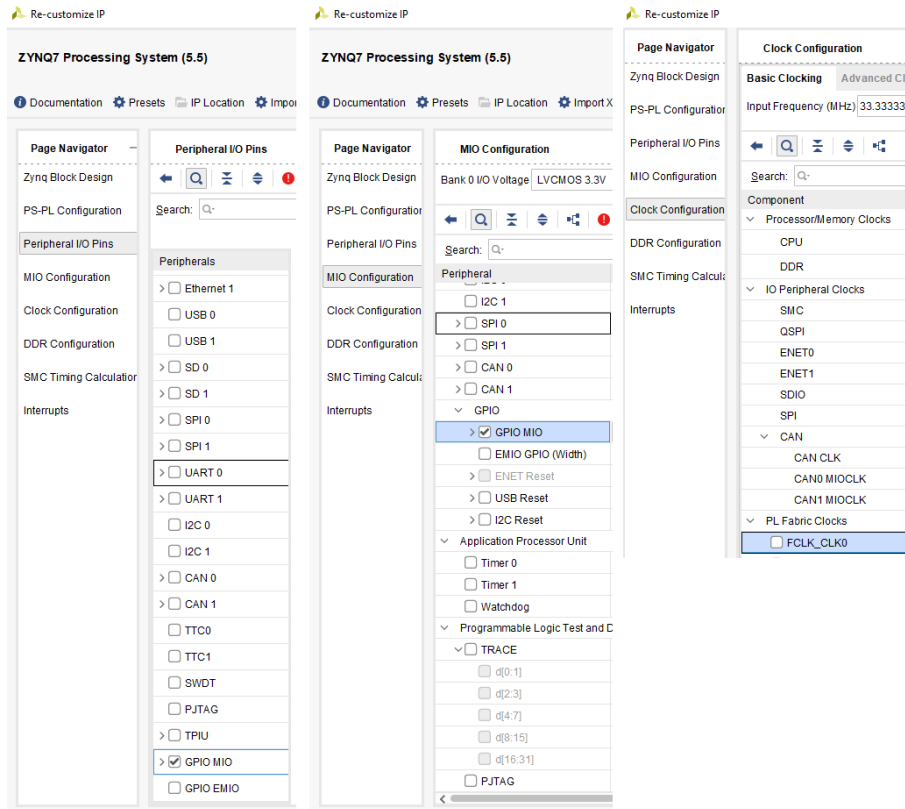


Figure 5.2.34: ZYNQ 7 IP Processing System Peripheral, MIO and Clock Settings



Figure 5.2.35: Updated schematic block design







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

### Add GPIO peripherals.

1. In the Sources panel choose Board (see figure 5.2.36), double click on the LED IP to add at the block diagram.
2. Do the same for the Push Buttons but at this time choose in the pop-up window **Connect Board Component** => **Create new IP** => **GPIO** (Figure 5.2.37).
3. From the IP Catalog select and add the user created IP **sw\_ip** (see figure 5.2.38).
4. From the IP Catalog add **Block Ram Controller** and **Block Memory Generator** – figure 5.2.40.
5. Run Connection Automation.
6. Rename the AXI GPIOs connected to the LEDs and Push Buttons to **leds** and **buttons** respectively – click on each AXI GPIO and rename them on the *Block Properties window*.
7. Right click on the **sw\_ip sw** port and select **Make external**. Since is connected rename the external port to **sw** by clicking on the **sw\_0** port and in the Properties window change the name.
8. Double Click on the **sw\_ip** to re-customize the IP. Edit the number of switches i.e. *Sw width* corresponding to the Zed board (8) or Zybo (4) as seen in figure 5.2.42
9. Validate the design. The design should look like in figure 5.2.41
10. In the design **Sources** right click on the *system.bd* design and **Create HDL Wrapper**, select *Let Vivado manage wrapper and auto update* option.
11. In the **Flow Navigator** window select **IP Integrator** => **Generate Block Design**.
12. Run Synthesis





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

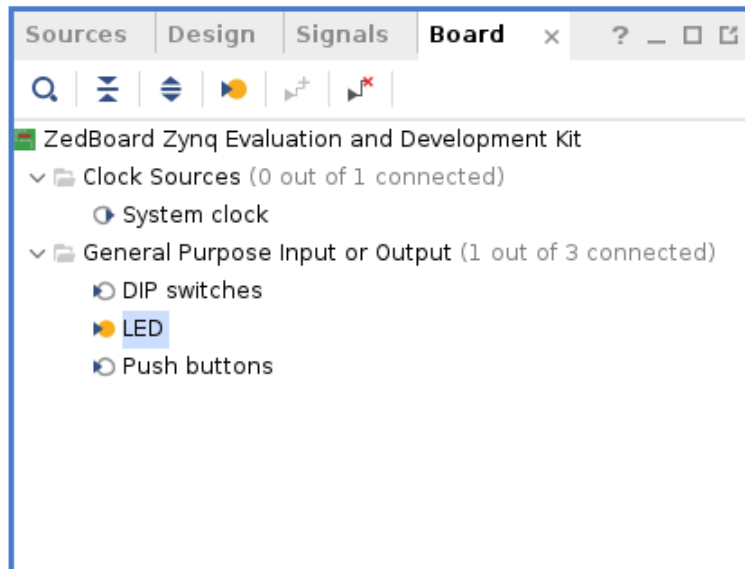


Figure 5.2.36: Adding Board IP LEDs

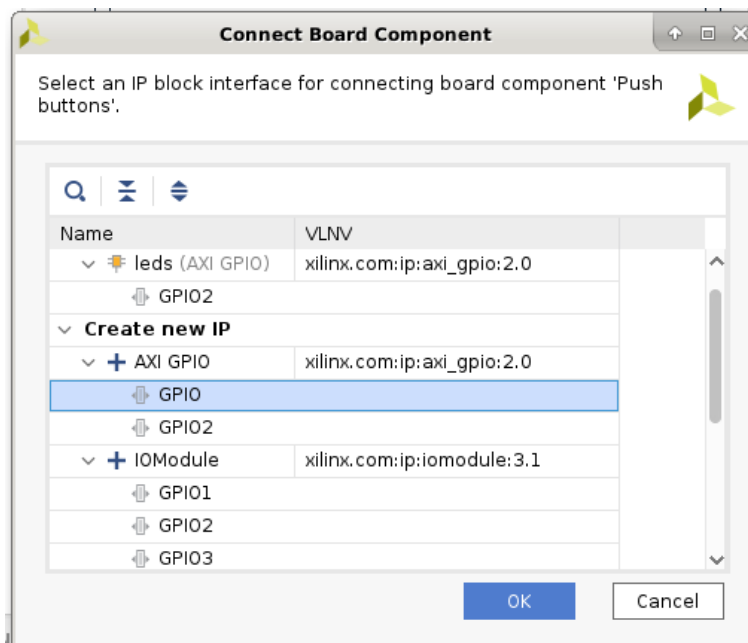


Figure 5.2.37: New AXI GPIO selected for the Push Buttons



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

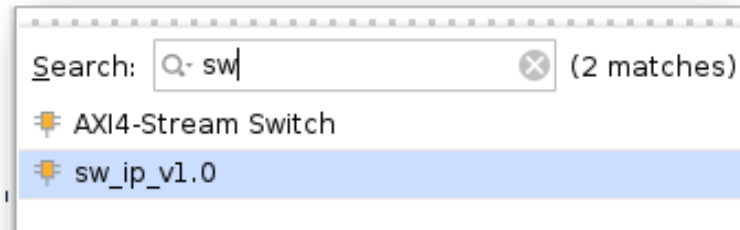


Figure 5.2.38: labsys3

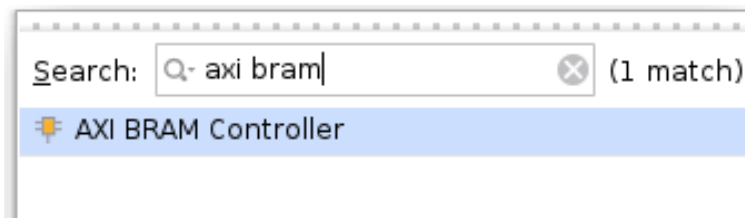


Figure 5.2.39: labsys4

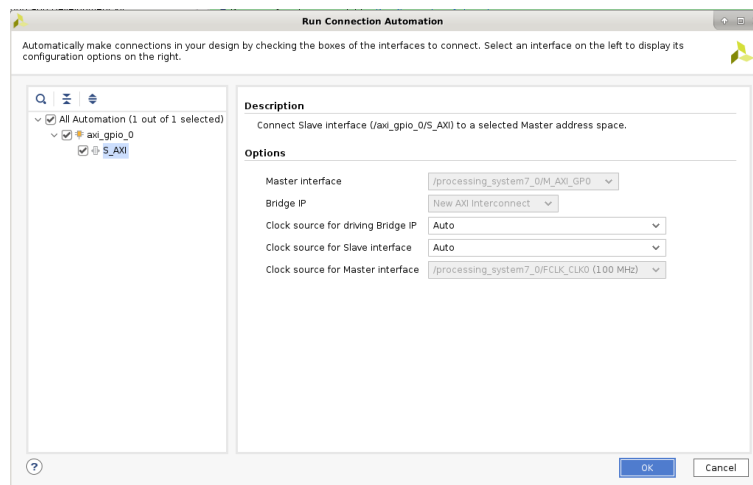


Figure 5.2.40: labsys2 run connection automation





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

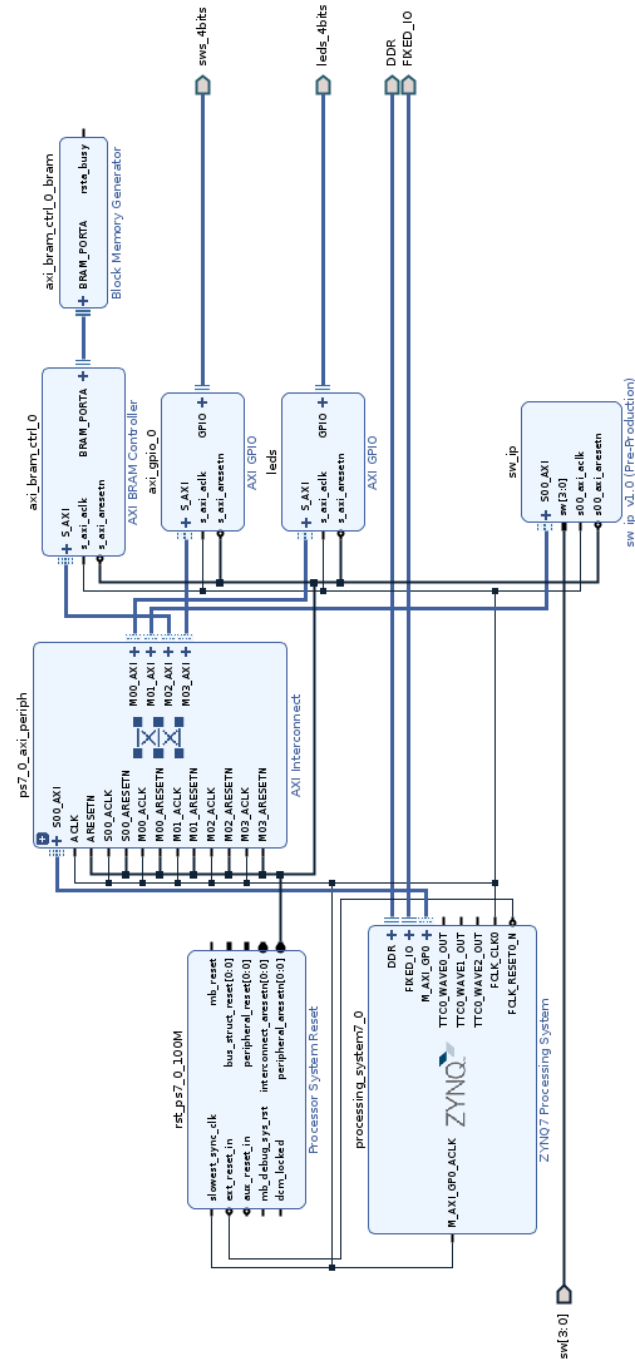


Figure 5.2.41: Design schematic of lab3



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

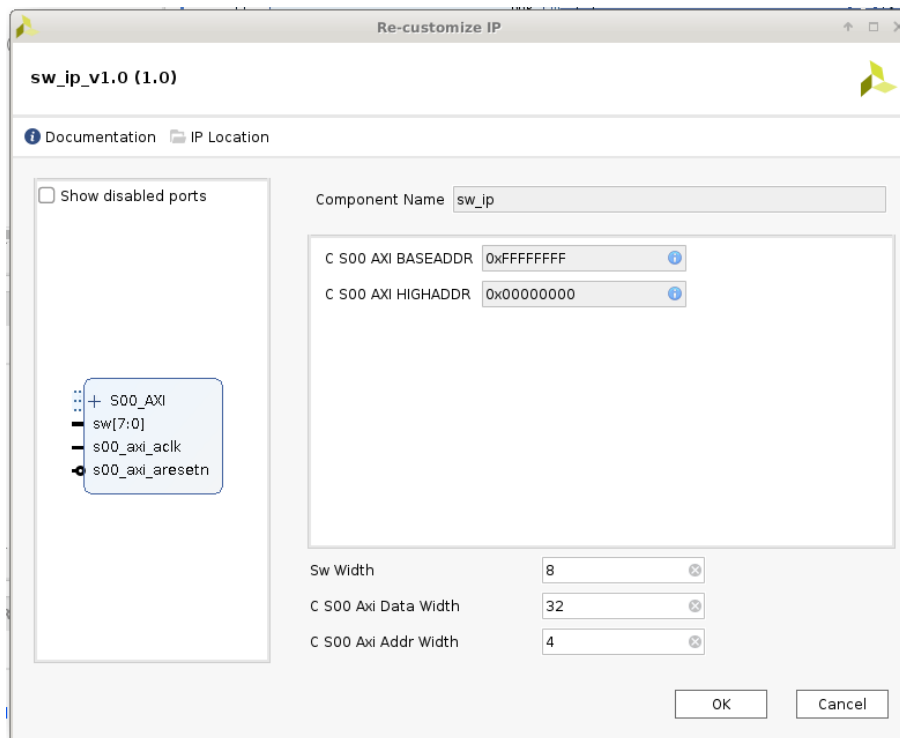


Figure 5.2.42: Re-customize IP window

### Open Synthesized Design and edit constraints

The `sw_ip` created has access to the Zedboard/Zybo switches. So in the constraints file the location and technology type must be introduced the. One of the possibilities is to open the synthesized design to edit the pin locations and use **IO Planning** layout.

1. Open Synthesized design and switch layout (in the right top corner of the Vivado to **I/O Planning**).
2. Click the **sw(8)** pin to open the sw IO ports loc (figure 5.2.43).
3. Edit the pin locations and technology for the corresponding board. Conform table edit the pad locations and set-up technology to LVCMOS33. The figure (5.2.44) shows the situation after editing the pin locations and technology for the Zed board.
4. Another way to add pin constraints is to download from the DigilentInc homepage the corresponding constrains file (.xdc) and add to the design.
  - (a) This can be done in the *Flow Navigator* pane click **Add Sources** => **Add or Create Constraints** and click **Next**.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

- (b) Click **Add Files** browse to the directory where the *zed.xdc* or *zybo.xdc* was downloaded. Select the file and click **Finish** to add the file.
- (c) Edit the corresponding .xdc file and comment the lines which are not needed with the character "#", then save the file.

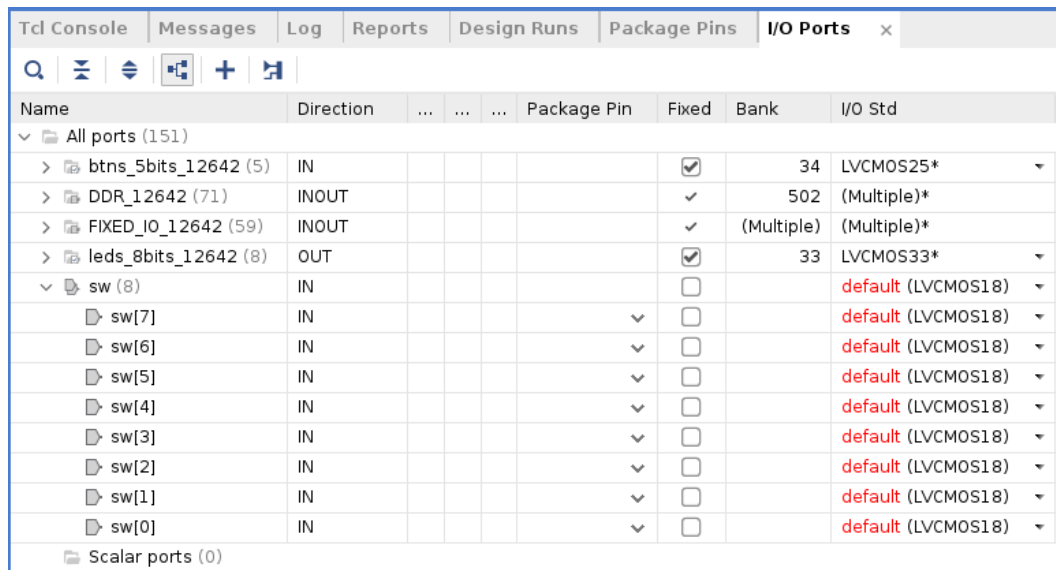


Figure 5.2.43: Edit sw\_ip constraints

Table 5.1: Zed Board DIP Switch Connections [Dig21a]

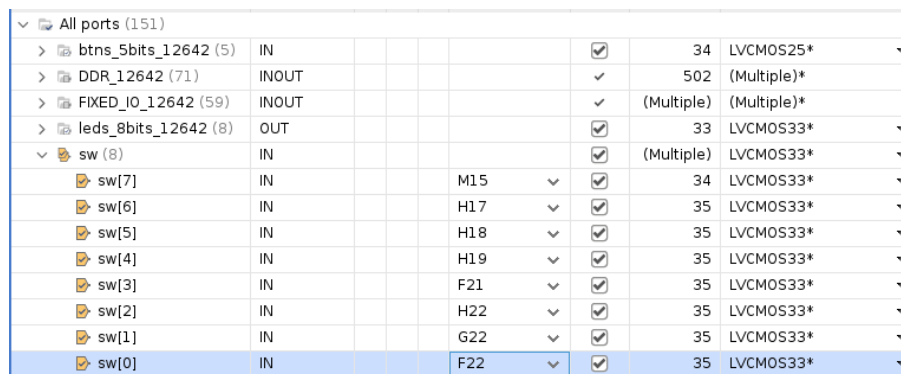
Signal Name	Zynq EPP pin
SW0	F22
SW1	G22
SW2	H22
SW3	F21
SW4	H19
SW5	H18
SW6	H17
SW7	M15



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
 EFOP 3.4.3-16-2016-00015

Table 5.2: Zybo Board DIP Switch Connections [Dig21b]

Signal Name	Zynq EPP pin
SW0	G15
SW1	P15
SW2	W13
SW3	T16



Component	Direction	Pin	Mode	Strength	IO Standard
sw[7]	IN	M15	✓	34	LVCMOS33*
sw[6]	IN	H17	✓	35	LVCMOS33*
sw[5]	IN	H18	✓	35	LVCMOS33*
sw[4]	IN	H19	✓	35	LVCMOS33*
sw[3]	IN	F21	✓	35	LVCMOS33*
sw[2]	IN	H22	✓	35	LVCMOS33*
sw[1]	IN	G22	✓	35	LVCMOS33*
sw[0]	IN	F22	✓	35	LVCMOS33*

Figure 5.2.44: Constraints for the sw pins

### Generate IP Integrator Outputs, the top-level HDL, export the hardware to Vitis and start Vitis.

1. Right-click on **system.bd**, and select **Create HDL Wrapper** to generate the top-level VHDL model. Leave the *Let Vivado manager wrapper and auto-update* option selected, and click **OK**. The **system\_wrapper.vhd** file will be created and added to the project (Figure 5.1.54).
2. In the sources panel, right-click again on **system.bd**, and select **Generate Output Products** and click **Generate** to generate the Implementation, Simulation and Synthesis files for the design. Select **Global**. You can also click on **Generate Block Design** in the *Flow Navigator pane* to do the same.

**Note:** If the synthesis option is **Global**, only wrapper files are generated during the block design generation phase, and the design will be synthesized as a whole at the synthesis stage. If the synthesis option is **Out of context per IP** or **Out of context per Block design**, the wrapper of the IP or block design will be generated and synthesized during block design generation, and the generated net lists will be combined together at the synthesis stage. Double-click on the file to see the content in the Auxiliary pane.

3. Notice that the VHDL file is already Set As the Top module in the design, indicated by the icon in front of the **system\_wrapper.vhd**.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

4. In the main menu Select **File => Export => Hardware** and click **Next**

**Note:** Since we do not have any hardware in Programmable Logic (PL) there is no bitstream to generate, hence the **Include bitstream** option is not necessary at this time. Click **Next**.

5. Name the XSA file to **system\_wrapper** and specify the folder name where the project is created.

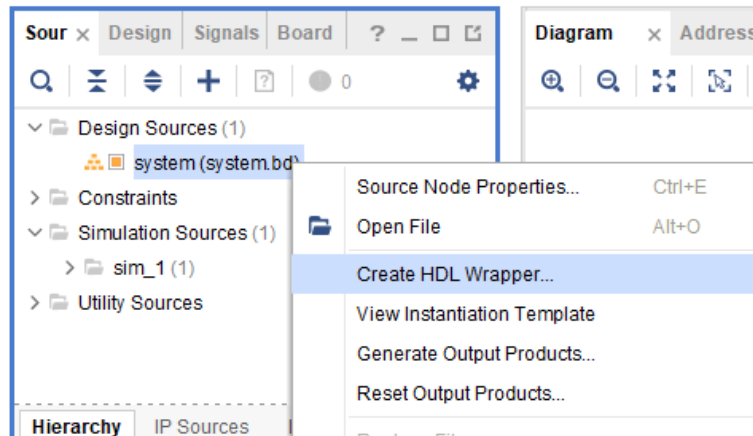


Figure 5.2.45: System Wrapper File Created

### Generate Bitstream and Export the bitstream to Vitis

1. In the *Flow Navigator* pane **Run Synthesis** if Necessary (Vivado reports *Synthesis Design is Out of Date*).
2. In the *Flow Navigator* pane **Run Implementation**
3. In the *Flow Navigator* pane run **Generate Bitstream**

used in the system such as interrupts.

### 5.2.3 Launch Vitis and Generate the Test Application of the project

1. In the main menu select **Tools => Launch Vitis IDE**;
2. Specify the workspace and click **Launch** (Figure 5.2.46)
3. On the Vitis welcome screen click **Create Platform Project**
4. In the **New Platform Project** specify the *Platform project name*: **zed\_system**. Figure 5.2.47
5. In the **Create a new platform from hardware XSA**. Click **Browse** (5.2.48) to specify the XSA (Xilinx Support Archive) file for the hardware specification. Click **Next**





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

6. The Vitis IDE open with the platform project created (5.2.49). In the menu select **File => New => Application Project** for creating the application "Hello World". This will open the application project creation wizard (5.2.50). Click **Next**.
7. The next window is the domain selection window. For this first project we create standalone\_domain (without operating system). Click **Next**. See 5.2.51.
8. In the new Application project examples select "Hello World" and click **Finish**.
9. In the file Explorer window right click on the **zed\_project**. Also expand the *hello\_world\_system => hello\_world=> src* and double click on the **helloworld.c** .C file to open it in the editor window.
10. Edit the **helloworld.c** program and add the program lines, which handle the LEDs and the switch IP. The program can be seen in the figure 5.2.56 and 5.2.57. Then save the program. Right click on the *hello\_word\_system* and select **Build project**.
11. If you worked with Zybo connect the board to your PC and power on.
12. If you worked with the Zedboard start **putty.exe** and login to "mazsola" to open a tunnel for the hardware server, where the Zedboard is connected. Also login to the computer "nyolcas".
13. For Zybo/Zed open RelTerm serial terminal. Select the UART port *CyprusVirtualCom0* and setup to *Baud: 115200, Parity: none, Data Bits: 8, Stop Bits: 1* then connect the terminal (Figure 5.1.39).
14. In the Main menu select **Xilinx →XSCT Console**. Then in the console window type **connect** press *Enter*, then type **reset** and press *Enter*. In the XSCT window should be displayed something similar like in 5.1.38.
15. In the Explorer window right click on the application project *hello\_world\_system* select **Run as →Run Configurations**. Once the *Run Configurations* windows opens, twice click on **System Project Debug** to create an application debug **SystemDebugger\_hello\_world\_system**(Figure 5.1.39). Then click on **Run**
16. "Hello World" appears on the RealTerm Terminal, as shown in the figure (5.1.40).

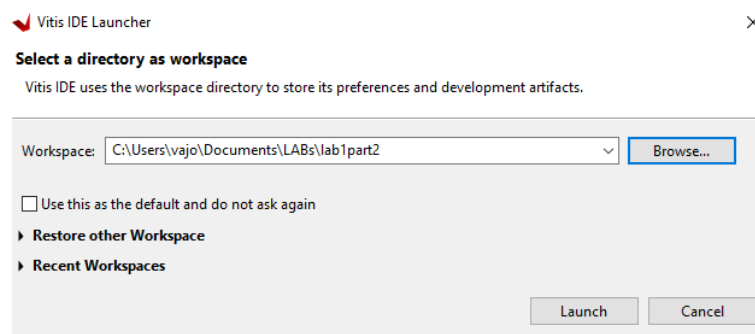


Figure 5.2.46: Lunch Vitis



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

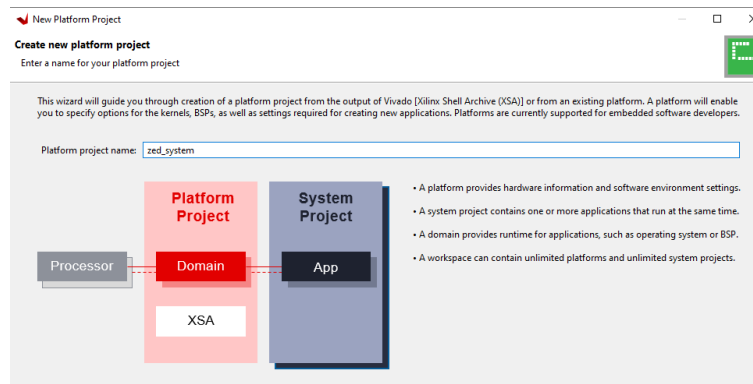


Figure 5.2.47: Create New Platform Project

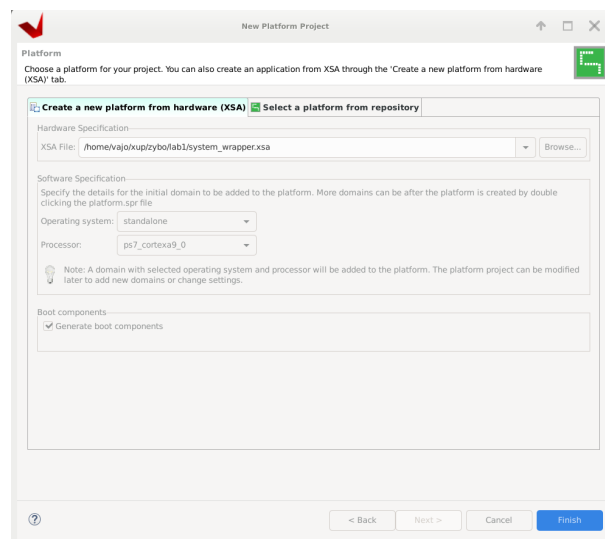


Figure 5.2.48: Create a new platform from hardware (XSA)





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

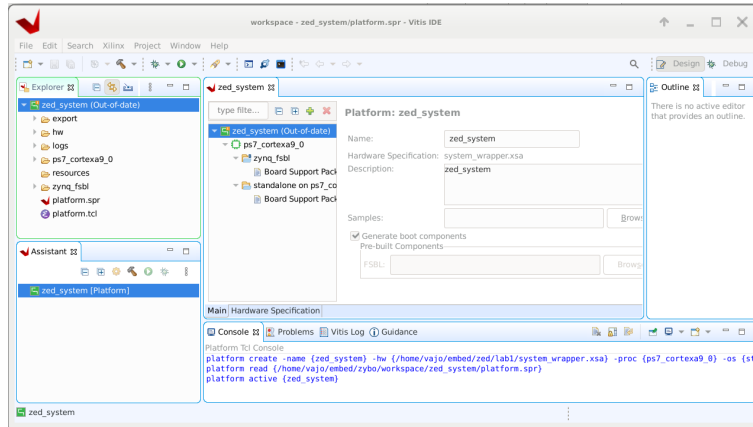


Figure 5.2.49: Vitis IDE with the platform project created

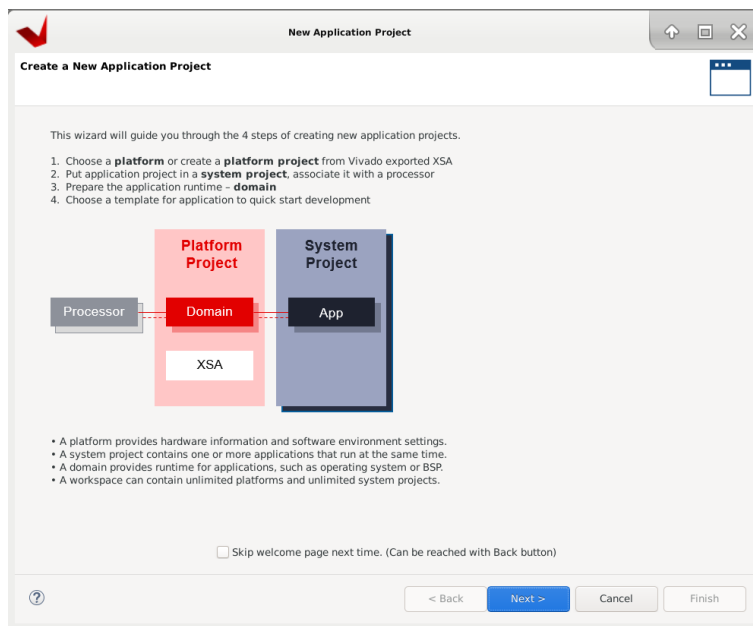


Figure 5.2.50: New Application Project Wizard





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

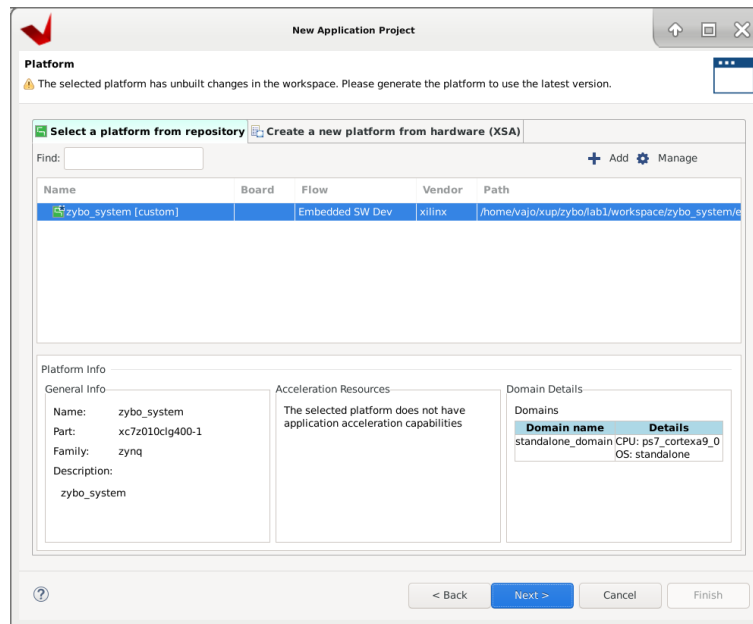


Figure 5.2.51: Select Platform





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

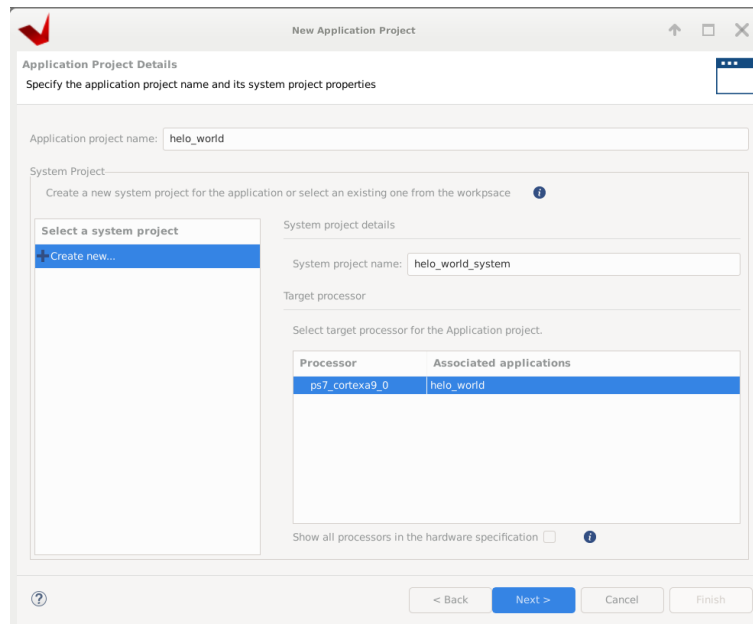


Figure 5.2.52: Specify the Application Project Name





**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

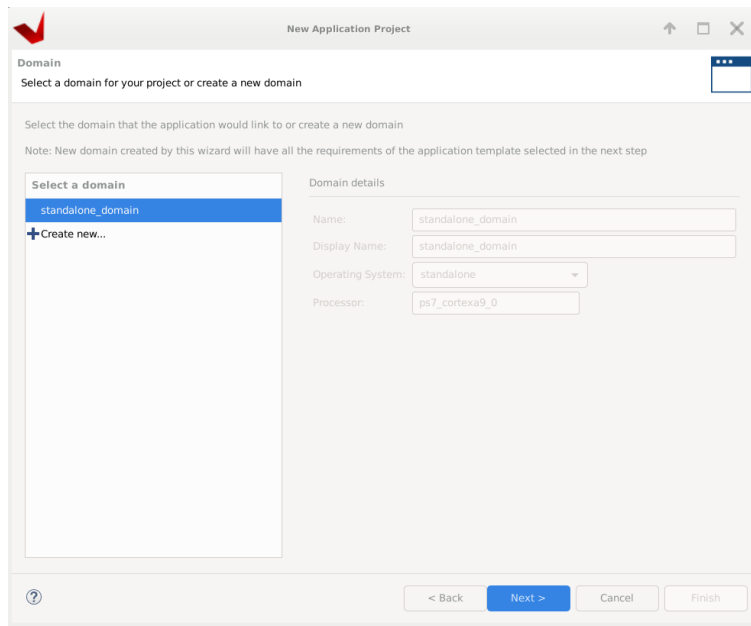


Figure 5.2.53: Select domain as standalone\_platform

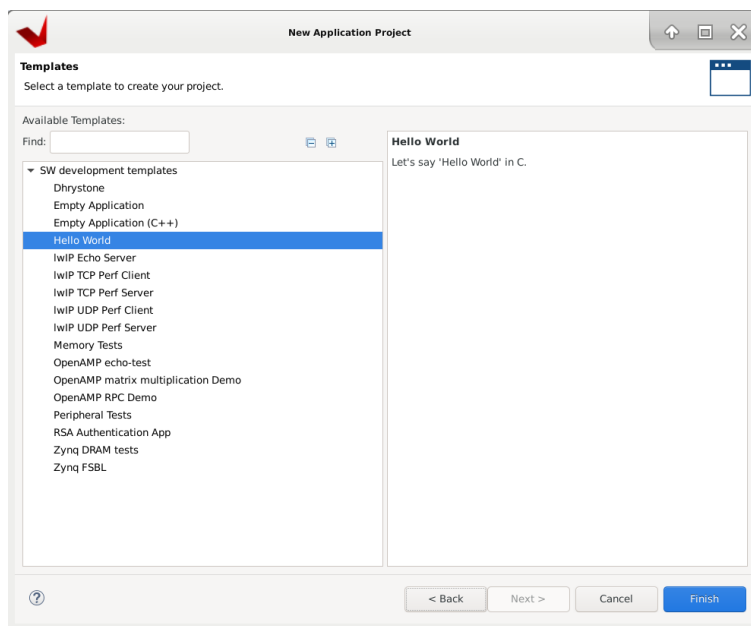


Figure 5.2.54: Select "Hello World" example project





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

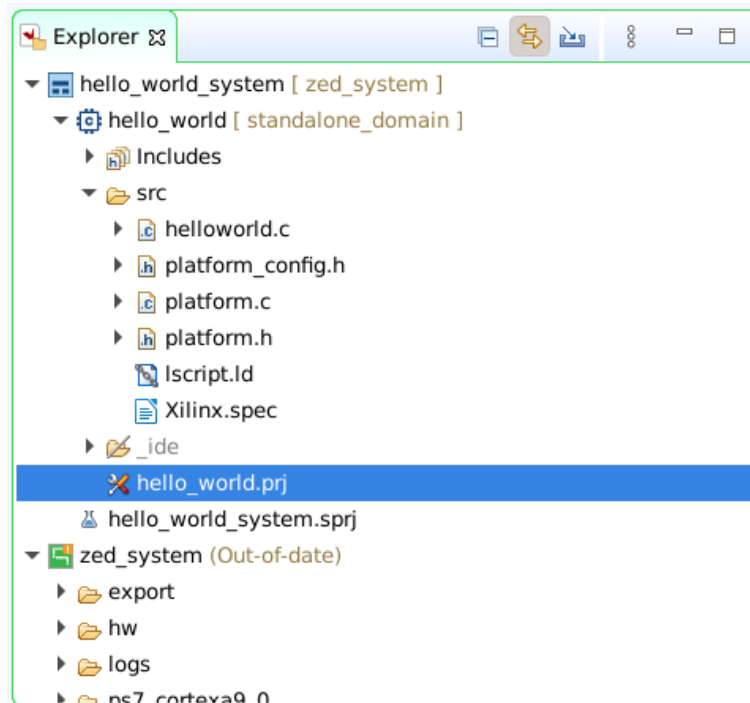


Figure 5.2.55: Application project "hello\_world" Explorer view

```
47 |
48 | #include <stdio.h>
49 | #include "platform.h"
50 | #include "xil_printf.h"
51 | #include "xparameters.h"
52 | #include "sw_ip.h"
53 | #include "xgpio.h"
54 | #define LEDs 0x5
55 | #define LED_CHANNEL 1
56 | // #define led_device_id XPAR_LEDS_DEVICE_ID
57 | int status_LED_GPIO, status_sw;
   | co
```

Figure 5.2.56: The "Hello World" program include section.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

```
60 int main()
61 {
62     //char led = LEDs;
63     XGpio dip, Gpio;
64     char led = LEDs;
65     char dip_check;
66
67     init_platform();
68     // sw initialize
69     //XGpio_Initialize(&dip, XPAR_SW_IP_DEVICE_ID);
70     //XGpio_SetDataDirection(&dip, 1, 0xffffffff);
71     // leds init
72     // LED PL
73     /* GPIO driver initialisation */
74     status_LED_GPIO = XGpio_Initialize(&Gpio, XPAR_SW_IP_DEVICE_ID);
75     if (status_LED_GPIO != XST_SUCCESS) {
76         xil_printf("GPIO output to the LEDs failed!\r\n");
77     }
78     /*Set the direction for the LEDs to output. */
79     xil_printf ("GPIO leds %x\r\n", status_LED_GPIO);
80     XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x00);
81
82     print("Hello World\r\n");
83     print("Successfully ran Hello World application\r\n");
84     while (1)
85     {
86         //SW_IP_mReadReg(&dip, unsigned RegOffset)
87         //dip_check = SW_IP_mReadReg(&dip, 0);
88         //xil_printf("Switch Status %x\r\n", dip_check);
89         XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, led);
90         xil_printf("led %1x\r\n", 0x0f&led);
91         led = ~led;
92         dip_check = SW_IP_mReadReg(XPAR_SW_IP_S00_AXI_BASEADDR, 0);
93         xil_printf("switch %1x\r\n", dip_check);
94         sleep(1);
95     }
96
97     cleanup_platform();
98     return 0;
99 }
```

Figure 5.2.57: The application program for lab3







„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

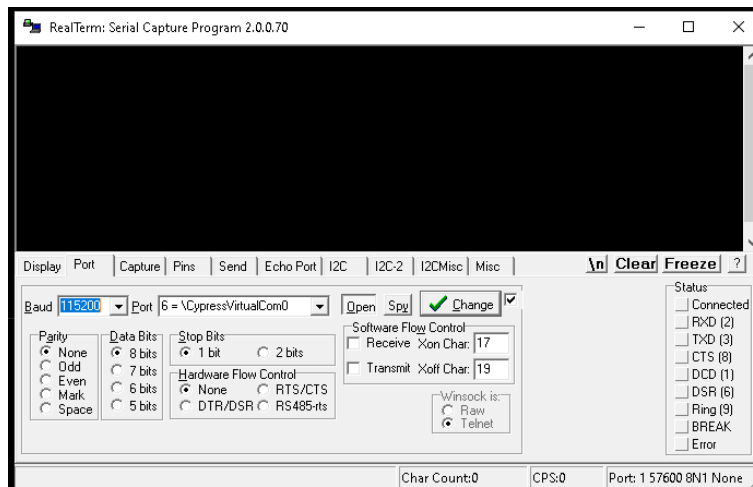


Figure 5.2.58: RealTerm terminal setup



Figure 5.2.59: Lab 3 program run result terminal widow.

In the part 1 of the laboratory work 1 (called Lab1) we passed through the steps, which have to be done for each embedded system project using Zynq architecture. True there were not explained the steps. Detailed information about the development process and "what to do and why to do" are explained in the Xilinx documentations (search for User Guides ug940, ug1165). For other information,



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

tutorials and video tutorials start the DocNav (Document Navigator), installed together with the Vitis installation. In Lab 2 there was presented the hardware design with Vivado IP library and lab3 presented the steps how to create a user specified IP Some of previously presented "Hello World" project material are based on Xilinx XUP (Xilinx University Program) and other tutorials such as ug1165 material [Xil18a].





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

## Chapter 6

### Directed Projects

For both projects presented in this chapter, a laboratory configuration has been set and is showed in Figure 6.0.1

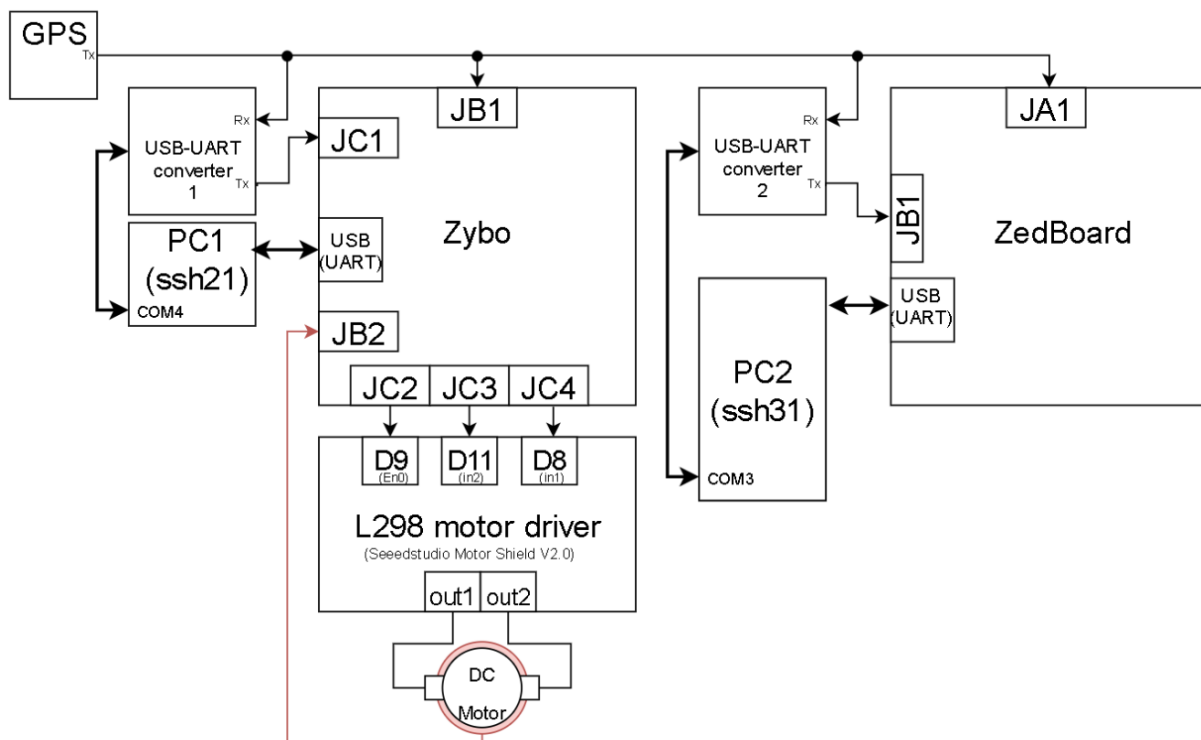


Figure 6.0.1





„FÖNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

## 6.1 Project 1: GPS parsing using SoC for Position Determination. Simulation then HW/SW Implementation

The goal of this work is to create an embedded system based on SoC and a GPS module to extract and display GPS coordinates.

### Software Requirements:

Vivado  
Serial Terminal (eg Realterm, TeraTerm, PuTTY...)  
com0com (free virtual serial port)  
Any IDE for C/C++ simulation

### Hardware Requirements:

ZedBoard  
GPS module 1Hz

### 6.1.1 Phase 1: Coding and Simulation

Taking into account that the final goal is to reach the implementation in ZYNQ SoC, it is advisable to code in C/C++ in order that it will be easier to implement on hardware. Other languages can be used but it is on the risk of the students since it is harder to find open resources and support. It is sad to see that the simulation is working on a language then getting trouble to find the useful tools for the hardware implementation. For ZYNQ the most used high level languages are C/C++ and MATLAB, then comes Python recently adopted.

The first checkpoint is to show me the simulation, if validated, you can start the hardware implementation phase

(please carefully read and understand the phase 2 too)

Data coming from GPS module are logged (see appendices), then 2 files are generated for you: one contains a 10 seconds text sent by the GPS module in case of a stable status of the module with full information, and a second text was sent by the same module in unstable situation because of some artifacts (e.g. number of satellites is not enough, noise...). Your design should consider both situations. Consider that the coordinates should be displayed in the following format:

**(Latitude: -18.836228, Longitude: -159.782770)**

### 6.1.2 Phase 2: Hardware design, Implementation and Test

During this step you should design the hardware of the solution, adapt your C/C++ program to the xilinx SDK syntax (including the necessary and compatible libraries/drivers).

It is allowed to only use PS for this purpose. But I give extra points for those who use PL as a hardware accelerator to compute GPS data after collection through UART.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

## 6.2 Project 2: Remote DC Motor Control using SoC and Hall effect Sensor

The aim of this project is to precisely control a DC motor using Zybo and hall effect sensor.

### Statement

The shaft of the DC motor has to perform a complete rotation than changes the direction to perform another complete rotation then change again the direction (loop). The rotation speed should be low (around half rotation per second). (Figure 6.0.1 shows the required setup for the project)

The motor driver (Motor Shield V2.0 based on L298 double H bridge chip) is connected to PMOD C. The PWM signal should pass through the port JC2. JC3 and JC4 are for setting the direction of the motor.

The hall effect sensor provides a square signal whose frequency is proportional to the speed of rotation of the motor. The sensor is connected to JB2 port. For one rotation of the shaft, there should be 700 impulses sent to the zybo.





## Chapter 7

### Exercises

#### 7.1 Multiple Choice Quizzes

1) DAC is used for

- converting analog signals to digital signals
- generating periodic signals
- generating sine wave signals

2) ADC converts

- a continuous-time signal to discrete-time signal
- a continuous-amplitude signal to discrete-amplitude signal

3) Conversion time is:

- measured in samples per second
- the inverse of the conversion rate
- the sum of sampling time and quantization time and coding time

4) Conversion rate is

- measured in samples per second
- the inverse of the sampling time

5) Suppose  $T_{s1}$  and  $T_{s2}$  are sampling periods of respectively ADC1 and ADC2 where  $T_{s1} > T_{s2}$ .

- ADC1 performs better conversions than ADC2
- ADC2 performs better conversions than ADC1





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- The change of sampling period does not affect the quality of the conversion

6) A comparator is a

- 1bit ADC
- 2bit ADC
- multiplexer
- Analog component

7) (Justify your answer(s)) SOC is a

- Software on Chip
- System on Chip
- Single Chip Component
- Integrated Circuit
- Embedded System

8) (Justify your answer(s)) Design productivity gap

- means: design productivity outpaces chip complexity
- means: chip complexity outpaces design productivity
- means: the number of transistors per chip increases faster than the number of transistors per staff-month
- could be reduced by integrating IPs

## 7.2 Questions

- 1) In order to have an embedded system based on two noisy sensors:
  - a) what can you suggest to put between ADC and sensors? (Suggest three elements)
  - b) explain your choice
  - c) draw a qualitative schematic of the solution
- 2) What can be the difference between a sensor and an actuator?
- 3) Give examples of 2 actuators and define the input and the output of each one.
- 4) Give 3 types of analog-to-digital converters. Mention the fastest ADC you know.
- 5) Give an alternative term to “voltage resolution”
- 6) What is the voltage resolution of a 3bit ADC if  $V_{ref} = 4V$  ?
- 7) Why an ADC is required for most of embedded systems based on analog sensors ? (in a short sentence)
- 8) Here is a signal (figure 7.2.1):



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

- is it a time domain or frequency domain representation ?
- what type of signal (waveform) is ?
- give the period and the frequency of this signal.
- what are the maximal and minimal amplitudes of this signal ?

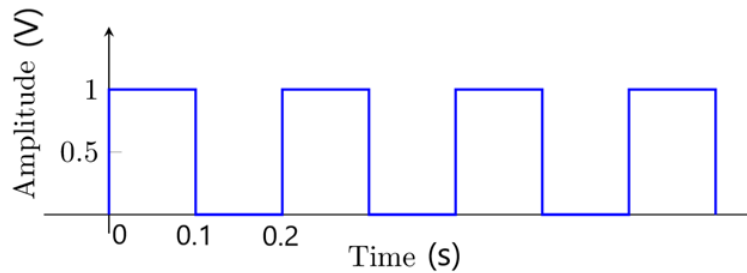


Figure 7.2.1

9) Suggest a schematic of a typical data acquisition/processing/displaying system (from physical quantity capture to display)

10) Consider a LED brightness controller (as shown in figure 7.2.2) as an embedded computer system based on rotational potentiometer.

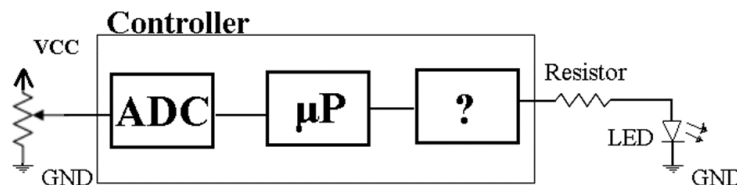


Figure 7.2.2

- What are the inputs of the controller?
- What are the inputs and outputs of the whole system?
- By describing the missing component, suggest two solutions to change the brightness of the LED
- Then, what will be the output of the controller for every solution you suggest?
- Suggest a completely different configuration to set the brightness of the LED. (by schematic)

11) Consider an Analog-to-Digital Converter having the following properties: 16-bit resolution and  $V_{ref} = 2.5V$ . A temperature sensor is connected the ADC and delivers  $900mV$  for an unknown temperature.





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

Assuming that at  $20^{\circ}C$  the sensor delivers  $700mV$  and the sensitivity is  $40mV/^{\circ}C$ , what are the unknown temperature and the output of the ADC ?

- 12) Give 3 common points and 3 differences between  $\mu C$ ,  $\mu P$  and  $SoC$
- 13) What is the resolution of an ADC if the  $V_{ref} = 4V$  and  $V_{LSB} = 250mV$  ?
- 14) Give a title to the figure 7.2.3

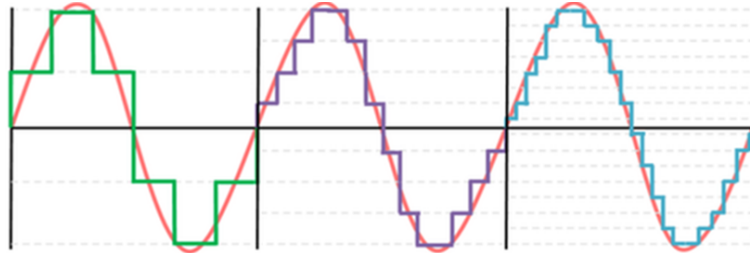


Figure 7.2.3

- 15) Make a qualitative drawing of a 2 bits digitalized sine wave.
- 16) What is the minimal wiring connection required for a full-duplex communication on UART? (illustrate by schematic)
- 17) Let be the figure 7.2.4

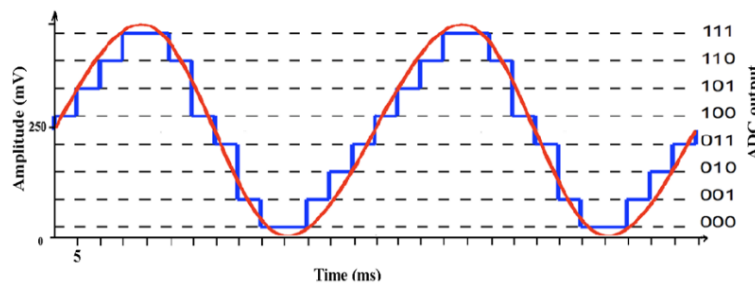


Figure 7.2.4

- a) what type of waveform is the continuous signal represented in the figure?
  - b) give the Period, the Frequency and the Peak-to-Peak voltage of the continuous signal
  - c) what represents the discrete signal on the figure?
  - d) give its Sampling Period, Sampling Frequency, Resolution and Quantum
  - e) in this case, are the frequency components of this signal aliased? (Justify your answer)
- 18) For the purpose of measuring the depth of a diver, you need to create a low-cost, low power and small size embedded system which has the purpose of indicating to the diver if the depth is less or more





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

than 3m. Two LEDs are the indicators such that if the level is less than 3m, the red LED should light up and the green LED lights off, and vice versa... You open your locker, (ignoring the packaging parts) you find:

- a red LED and a green LED,
- 2 analog pressure sensors; sensorA and sensorB (their behavior is described on figure 7.2.5)
- $\mu C$
- $\mu P$
- Some ICs (ADC, PWM, Comparator, Logic Inverter, Variable Reference Voltage)
- LCD
- Battery

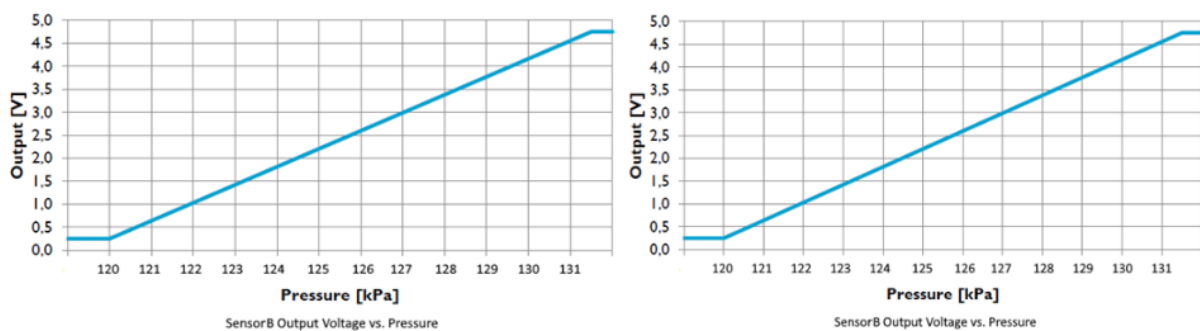


Figure 7.2.5

Assuming that the depth can be expressed with the figure 7.2.6

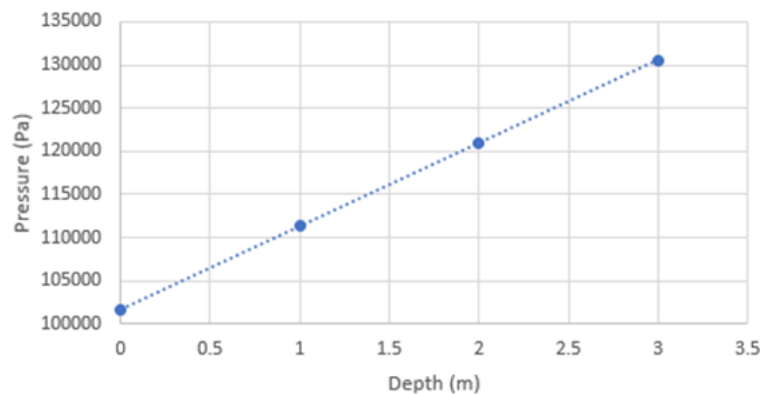


Figure 7.2.6

a) Propose two different solutions for the same purpose by drawing bloc diagrams justifying every choice of the components.



„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

b) Give a detailed explanation of the process. (A code is also allowed (to support your answer) if is followed by detailed comments).

19) Assuming you have an infinite budget, propose a completely different solution for the same purpose by means of the same LEDs (red and green).

20) We are trying to make a communication between two devices via UART in order to transmit the character “b”. The UART setup is 115200/8/1/N

- a) Draw the equivalent frame to perform the communication
- b) Define every parameter of the abbreviation 115200/8/1/N
- c) What is the necessary time to send the entire frame?
- d) Draw the frame if the communication is based on RS232 instead of UART





„FÖNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

## Appendix A

### Additional Content

#### A.1 GPS Data

##### A.1.1 10 seconds 1Hz GPS NMEA (correct data)

\$GPGGA,084356.00,4804.98959,N,02046.07578,E,1,06,1.90,139.3,M,38.1,M,,\*5D  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.34\*0B  
\$GPGSV,2,1,08,01,74,155,25,04,33,206,29,08,01,184,07,09,00,218,\*7C  
\$GPGSV,2,2,08,11,35,173,27,21,43,145,33,31,28,088,33,32,18,047,24\*76  
\$GPGLL,4804.98959,N,02046.07578,E,084356.00,A,A\*64  
\$GPRMC,084357.00,A,4804.98967,N,02046.07574,E,0.028,041120,,A\*71  
\$GPVTG,T,,M,0.028,N,0.052,K,A\*2E  
\$GPGGA,084357.00,4804.98967,N,02046.07574,E,1,06,1.90,139.6,M,38.1,M,,\*58  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.34\*0B  
\$GPGSV,2,1,08,01,74,155,25,04,33,206,29,08,01,184,07,09,00,218,\*7C  
\$GPGSV,2,2,08,11,35,173,27,21,43,145,32,31,28,088,33,32,18,047,24\*77  
\$GPGLL,4804.98967,N,02046.07574,E,084357.00,A,A\*64  
\$GPRMC,084358.00,A,4804.98969,N,02046.07574,E,0.057,041120,,A\*78  
\$GPVTG,T,,M,0.057,N,0.106,K,A\*26  
\$GPGGA,084358.00,4804.98969,N,02046.07574,E,1,06,1.90,140.0,M,38.1,M,,\*51  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.34\*0B  
\$GPGSV,2,1,08,01,74,155,24,04,33,206,29,08,01,184,09,00,218,\*7A  
\$GPGSV,2,2,08,11,35,173,27,21,43,145,32,31,28,088,33,32,18,047,24\*77  
\$GPGLL,4804.98969,N,02046.07574,E,084358.00,A,A\*65  
\$GPRMC,084359.00,A,4804.98974,N,02046.07573,E,0.107,041120,,A\*76  
\$GPVTG,T,,M,0.107,N,0.199,K,A\*24  
\$GPGGA,084359.00,4804.98974,N,02046.07573,E,1,06,1.90,140.5,M,38.1,M,,\*5E  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.35\*0A  
\$GPGSV,2,1,08,01,74,155,23,04,33,206,29,08,01,184,09,00,218,\*7D  
\$GPGSV,2,2,08,11,35,173,27,21,43,145,32,31,28,088,33,32,18,047,24\*77  
\$GPGLL,4804.98974,N,02046.07573,E,084359.00,A,A\*6F  
\$GPRMC,084400.00,A,4804.98975,N,02046.07577,E,0.330,041120,,A\*7E  
\$GPVTG,T,,M,0.330,N,0.611,K,A\*25  
\$GPGGA,084400.00,4804.98975,N,02046.07577,E,1,06,1.90,141.0,M,38.1,M,,\*54  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.35\*0A  
\$GPGSV,2,1,08,01,74,155,22,04,33,206,28,08,01,184,09,00,218,\*7D





**„FÖNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

\$GPGSV,2,2,08,11,35,173,27,21,43,145,32,31,28,088,33,32,18,047,25\*76  
\$GPGLL,4804.98975,N,02046.07577,E,084400.00,A,A\*61  
\$GPRMC,084401.00,A,4804.98986,N,02046.07569,E,0.213,,041120,,A\*7C  
\$GPVTG,,T,,M,0.213,N,0.394,K,A\*2D  
\$GPGGA,084401.00,4804.98986,N,02046.07569,E,1,06,1.90,141.5,M,38.1,M,,\*53  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.35\*0A  
\$GPGSV,2,1,08,01,74,155,21,04,33,206,28,08,01,184,,09,00,218,\*7E  
\$GPGSV,2,2,08,11,35,173,26,21,43,145,32,31,28,088,32,32,18,047,25\*76  
\$GPGLL,4804.98986,N,02046.07569,E,084401.00,A,A\*63  
\$GPRMC,084402.00,A,4804.98996,N,02046.07563,E,0.062,,041120,,A\*70  
\$GPVTG,,T,,M,0.062,N,0.115,K,A\*22  
\$GPGGA,084402.00,4804.98996,N,02046.07563,E,1,06,1.90,142.1,M,38.1,M,,\*5C  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.35\*0A  
\$GPGSV,2,1,08,01,74,155,22,04,33,206,28,08,01,184,,09,00,218,\*7D  
\$GPGSV,2,2,08,11,35,173,26,21,43,145,32,31,28,088,32,32,18,047,25\*76  
\$GPGLL,4804.98996,N,02046.07563,E,084402.00,A,A\*6B  
\$GPRMC,084403.00,A,4804.99008,N,02046.07557,E,0.107,,041120,,A\*7B  
\$GPVTG,,T,,M,0.107,N,0.197,K,A\*2A  
\$GPGGA,084403.00,4804.99008,N,02046.07557,E,1,06,1.90,142.7,M,38.1,M,,\*53  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.35\*0A  
\$GPGSV,2,1,08,01,74,155,21,04,33,206,28,08,01,184,,09,00,218,\*7E  
\$GPGSV,2,2,08,11,35,173,26,21,43,145,31,31,28,088,32,32,18,047,25\*75  
\$GPGLL,4804.99008,N,02046.07557,E,084403.00,A,A\*62  
\$GPRMC,084404.00,A,4804.99013,N,02046.07555,E,0.017,,041120,,A\*74  
\$GPVTG,,T,,M,0.017,N,0.031,K,A\*27  
\$GPGGA,084404.00,4804.99013,N,02046.07555,E,1,06,1.90,143.0,M,38.1,M,,\*5A  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.35\*0A  
\$GPGSV,2,1,08,01,74,155,21,04,33,206,28,08,01,184,,09,00,218,\*7E  
\$GPGSV,2,2,08,11,35,173,26,21,43,145,31,31,28,088,32,32,18,047,25\*75  
\$GPGLL,4804.99013,N,02046.07555,E,084404.00,A,A\*6D  
\$GPRMC,084405.00,A,4804.99021,N,02046.07544,E,0.043,,041120,,A\*75  
\$GPVTG,,T,,M,0.043,N,0.079,K,A\*2A  
\$GPGGA,084405.00,4804.99021,N,02046.07544,E,1,06,1.90,143.3,M,38.1,M,,\*59  
\$GPGSA,A,3,31,01,04,21,11,32,,,,,3.02,1.90,2.35\*0A  
\$GPGSV,2,1,08,01,74,155,22,04,33,206,28,08,01,184,,09,00,218,\*7D  
\$GPGSV,2,2,08,11,35,173,26,21,43,145,31,31,28,088,32,32,18,047,25\*75  
\$GPGLL,4804.99021,N,02046.07544,E,084405.00,A,A\*6D  
\$GPRMC,084406.00,A,4804.99032,N,02046.07518,E,0.277,,041120,,A\*78  
\$GPVTG,,T,,M,0.277,N,0.514,K,A\*21

### A.1.2 10 seconds 1Hz GPS NMEA (incorrect data)

\$GPGGA,,,,,0,00,99.99,,,,\*48  
\$GPGSA,A,1,,,,,,,,,99.99,99.99,99.99\*30  
\$GPGSV,1,1,00\*79  
\$GPGLL,,,,,V,N\*64  
\$GPRMC,,V,,,,,,,,,N\*53  
\$GPVTG,,,,,,N\*30  
\$GPGGA,,,,,0,00,99.99,,,,\*48  
\$GPGSA,A,1,,,,,,,,,99.99,99.99,99.99\*30  
\$GPGSV,1,1,00\*79  
\$GPGLL,,,,,V,N\*64  
\$GPTXT,01,01,02,ANTSTATUS=INIT\*25  
\$GPRMC,,V,,,,,,,,,N\*53





Európai Unió

„FÖNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

\$GPVTG,,,,,N\*30  
\$GPGGA,,,,,0,00,99.99,,,, \*48  
\$GPGSA,A,1,,,,,99.99,99.99,99.99\*30  
\$GPGSV,1,1,00\*79  
\$GPGLL,,,,,V,N\*64  
\$GPTXT,01,01,02,ANTSTATUS=OK\*3B  
\$GPRMC,,V,,,,,N\*53  
\$GPVTG,,,,,N\*30  
\$GPGGA,,,,,0,00,99.99,,,, \*48  
\$GPGSA,A,1,,,,,99.99,99.99,99.99\*30  
\$GPGSV,1,1,00\*79  
\$GPGLL,,,,,V,N\*64  
\$GPRMC,,V,,,,,N\*53  
\$GPVTG,,,,,N\*30  
\$GPGGA,,,,,0,00,99.99,,,, \*48  
\$GPGSA,A,1,,,,,99.99,99.99,99.99\*30  
\$GPGSV,1,1,00\*79  
\$GPGLL,,,,,V,N\*64  
\$GPRMC,,V,,,,,N\*53  
\$GPVTG,,,,,N\*30  
\$GPGGA,,,,,0,00,99.99,,,, \*48  
\$GPGSA,A,1,,,,,99.99,99.99,99.99\*30  
\$GPGSV,1,1,00\*79  
\$GPGLL,,,,,V,N\*64  
\$GPRMC,,V,,,,,N\*53  
\$GPVTG,,,,,N\*30  
\$GPGGA,,,,,0,00,99.99,,,, \*48  
\$GPGSA,A,1,,,,,99.99,99.99,99.99\*30  
\$GPGSV,1,1,00\*79  
\$GPGLL,,,,,V,N\*64  
\$GPRMC,,V,,,,,N\*53  
\$GPVTG,,,,,N\*30  
\$GPGGA,,,,,0,00,99.99,,,, \*48  
\$GPGSA,A,1,,,,,99.99,99.99,99.99\*30  
\$GPGSV,1,1,00\*79  
\$GPGLL,,,,,V,N\*64  
\$GPRMC,,V,,,,,N\*53  
\$GPVTG,,,,,N\*30  
\$GPGGA,,,,,0,00,99.99,,,, \*48  
\$GPGSA,A,1,,,,,99.99,99.99,99.99\*30  
\$GPGSV,1,1,00\*79  
\$GPGLL,,,,,V,N\*64  
\$GPRMC,,V,,,,,N\*53  
\$GPVTG,,,,,N\*30





„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében  
EFOP 3.4.3-16-2016-00015

## Bibliography

- [ARMa] ARM. Amba documentation, <https://developer.arm.com/architectures/system-architectures/amba>.
- [ARMb] ARM. Cortex - a9 revision: r4p1 technical reference manual, <https://developer.arm.com/documentation/ddi0407/g>.
- [CEES14] Louise Helen Crockett, Ross Elliot, Martin Enderwitz, and Robert Stewart. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, 2014.
- [CGL<sup>+</sup>20] Marcello Coppola, Miltos D Grammatikakis, Riccardo Locatelli, Giuseppe Maruccia, and Lorenzo Pieralisi. *Design of cost-efficient interconnect processing units: Spidergon STNoC*. CRC press, 2020.
- [Cor76] Intel Corp. *Data Catalog*. 1976.
- [Dig21a] Digilent. Zedboard, hardware user's guide, 2021.
- [Dig21b] Digilent. Zybo z7 reference manual, 2021.
- [GD99] Göran Herrman Gerd Dost. *Entwurf und Technologie von Mikroprozessoren*. Fachbuchverlag Leipzig, 1999.
- [GK83] Daniel D Gajski and Robert H Kuhn. New vlsi tools. *Computer*, 16(12):11–14, 1983.
- [HKM08] Takuji Hara, Norio Kambayashi, and Noboru Matsushima. *Industrial innovation in Japan*. Routledge, 2008.
- [Jr.07] George M. Phillips Jr. *The Collector's guide to Vintage Intel Microchips*. 2007.
- [KS13] Abderahman Kriouile and Wendelin Serwe. Formal analysis of the ace specification for cache coherent systems-on-chip. In *International Workshop on Formal Methods for Industrial Critical Systems*, pages 108–122. Springer, 2013.
- [Mak02] Tsugio Makimoto. The hot decade of field programmable technologies. In *2002 IEEE International Conference on Field-Programmable Technology, 2002.(FPT). Proceedings.*, pages 3–6. IEEE, 2002.
- [Moo98] Gordon E Moore. Cramping more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.
- [Pre14] Intel Free Press. Apple watch revealed: The original, <https://newsroom.intel.com/editorials/apple-watch-revealed-iwatch-intel-microroma>, Sep 2014.
- [SSW04] Andrew Sloss, Dominic Symes, and Chris Wright. *ARM system developer's guide: designing and optimizing system software*. Elsevier, 2004.
- [STGH19] Sahil Shah, Hakan Töreyn, Cihan Berk Güngör, and Jennifer Hasler. A real-time vital-sign monitoring in the physical domain on a mixed-signal reconfigurable platform. *IEEE transactions on biomedical circuits and systems*, 13(6):1690–1699, 2019.
- [tec] Two new apple socs, two market events: Apple a14 and m1, <https://www.techinsights.com/ja/node/33562>.
- [VSM01] Andrew M Volk, Peter A Stoll, and Paul Metrovich. Recollections of early chip development at intel. *Intel Technology journal*, 2001.
- [Xil] Embedded system design flow on zynq using vivado, <https://www.xilinx.com/support/university/vivado/vivado-workshops/vivado-embedded-design-flow-zynq.html>.
- [Xil18a] Xilinx. Lab workbook, use vivado to build an embedded system, <https://www.xilinx.com/support/university/vivado/vivado-workshops/vivado-embedded-design-flow-zynq.html>, 2018.





**„FŐNIX ME” – Megújuló Egyetem felsőoktatási intézményi fejlesztések a felsőfokú oktatás minőségének és hozzáférhetőségének együttes javítása érdekében**  
EFOP 3.4.3-16-2016-00015

[Xil18b] Xilinx. Zynq-7000 soc data sheet overview, v1.11.1, [https://www.xilinx.com/support/documentation/data\\_sheets/ds190-zynq-7000-overview.pdf](https://www.xilinx.com/support/documentation/data_sheets/ds190-zynq-7000-overview.pdf), 2018.

[Xil21] Xilinx. Zynq-7000 soc technical reference manual, [https://www.xilinx.com/support/documentation/user\\_guides/ug585-zynq-7000-trm.pdf](https://www.xilinx.com/support/documentation/user_guides/ug585-zynq-7000-trm.pdf), 2021.

