

MicroBlaze Core Interfaces

Creation of the Hardware Platform

Examples of IP Cores

Software Development

LECTURE 6

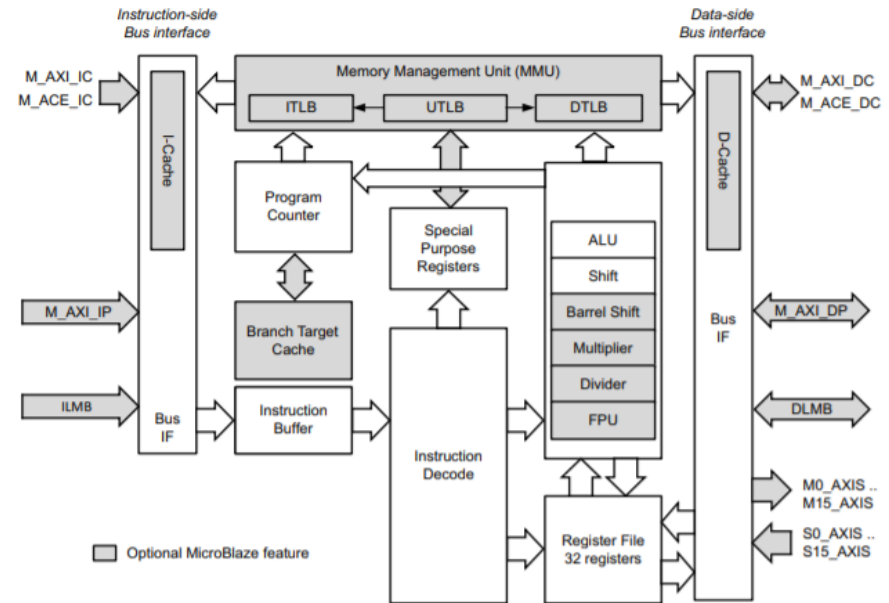
PEDRO SANTOS (SLIDES BY IOULIIA SKLIAROVA)

MicroBlaze Interfaces

- MicroBlaze does not separate data accesses to I/O and memory (it uses **memory-mapped I/O**).
- The processor has up to three interfaces for memory accesses:
 - Local Memory Bus (LMB) providing single-cycle access to on-chip dual-port block RAM.
 - **Advanced eXtensible Interface (AXI4)** for connection to both **on-chip and off-chip peripherals and memory**.
 - Advanced eXtensible Interface (AXI4) or AXI Coherency Extension (ACE) **for cache coherent connections to memory**.
- MicroBlaze also supports up to 16 AXI4-Stream interface ports, each with one master and one slave interface.
- The interfaces on MicroBlaze are 32 bits wide.

MicroBlaze Core Interfaces

- **M_AXI_DP**: Peripheral Data Interface, AXI4-Lite or AXI4 interface
- **DLMB**: Data interface, Local Memory Bus (BRAM only)
- **M_AXI_IP**: Peripheral Instruction interface, AXI4-Lite interface
- **ILMB**: Instruction interface, Local Memory Bus (BRAM only)
- **M0_AXIS..M15_AXIS**: AXI4-Stream interface master direct connection interfaces
- **S0_AXIS..S15_AXIS**: AXI4-Stream interface slave direct connection interfaces
- **M_AXI_DC**: Data-side cache AXI4 interface
- **M_ACE_DC**: Data-side cache AXI Coherency Extension (ACE) interface
- **M_AXI_IC**: Instruction-side cache AXI4 interface
- **M_ACE_IC**: Instruction-side cache AXI Coherency Extension (ACE) interface
- **Core**: Miscellaneous signals for: clock, reset, interrupt, debug, trace



AXI

- **Advanced eXtensible Interface** is a point to point interconnect that is designed for high performance, high speed microcontroller systems.
- The **AXI** protocol is based on a **point-to-point** interconnect to avoid bus sharing and therefore allow higher bandwidth and lower latency.
- There are three types of AXI4 interfaces:
 - **AXI4-Lite**—for simple, low-throughput memory-mapped communication, providing a register-like structure with reduced features and complexity (1 transfer per transaction)
 - **AXI4** —for high-performance memory-mapped requirements (up to 256 data transfers)
 - **AXI4-Stream**—for high-speed streaming data (unlimited amount of data)
- The **AXI specifications describe an interface between a single AXI master and a single AXI slave.**
- **AXI interconnect** allow multiple masters and/or multiple slaves to interface with each other.
 - In reality, interconnects contain slave interfaces that connect to AXI masters and master interfaces that connect to AXI slaves.
 - What goes on in an interconnect—i.e., how different masters communicate to different slaves—depends on the implementation.
 - Interconnects can allow a shared address bus, shared data bus, both shared, or neither shared.

AXI4 and AXI4-Lite Channels

- There are five independent channels between an AXI master and slave.
- The **address channels** are used to send address and control information while performing a basic handshake between master and slave.
- A master reads data from and writes data to a slave. Read response information is placed on the **read data channel**, while **write response** information has a **dedicated channel**. This way the master can verify a write transaction has been completed.
- Every exchange of data is called a **transaction**. A transaction includes the address and control information, the data sent, as well as any response information. The actual data is sent in bursts which contain multiple transfers.



Example MicroBlaze Project

MicroBlaze-based Project Example

- MicroBlaze processor
- Local instruction and data memory
- Clock and reset units
- General purpose I/O ports (connected to Nexys-4 LEDs, switches, buttons and 7-segment displays)
- RS232 UART
- AXI interconnect (crossbar for microprocessor and peripherals interconnect)
- Interrupt controller

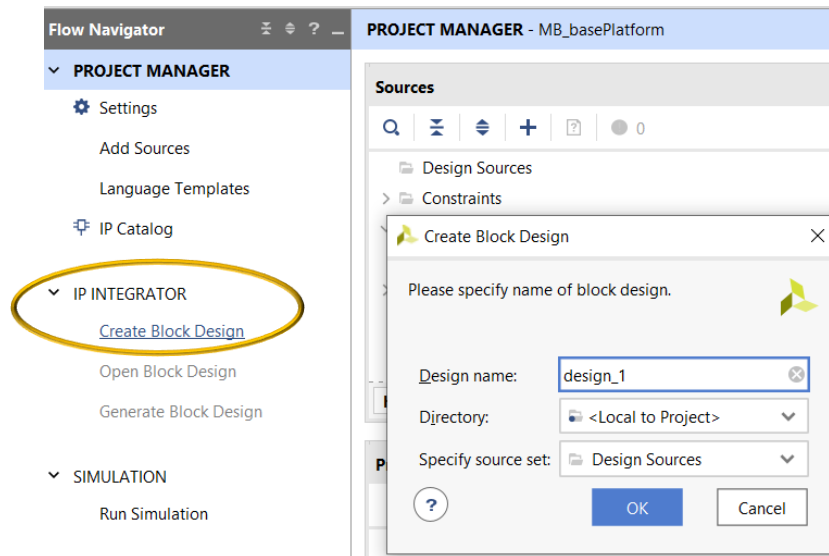
Board File Installation

Configure the board:

- Do not use Install/Update feature when creating a new project
- Download the board files from [Ficheiro "Nexys4.rar" \(board support files da Nexys-4 para Xilinx Vivado\)](#) on eLearning
- Copy nexys4 folder to Vivado installation folder (`C:\Xilinx\Vivado\2022.2\data\boards\board_files`)
 - By default this folder contains XML files for different FPGA boards manufactured by Xilinx
 - XML files define various interfaces on the board, such as slide switches, push buttons, LEDs, USB-UART, memory, Ethernet etc.
- Restart Vivado
- You are now ready to start a new IP Integrator based Vivado project for the Nexys-4 board
- If not working, execute TCL:
 - `set_param board.repoPaths ""`
 - `set_param board.repoPaths {C:/Xilinx/Vivado/2022.2/data/boards/board_files/nexys4}`

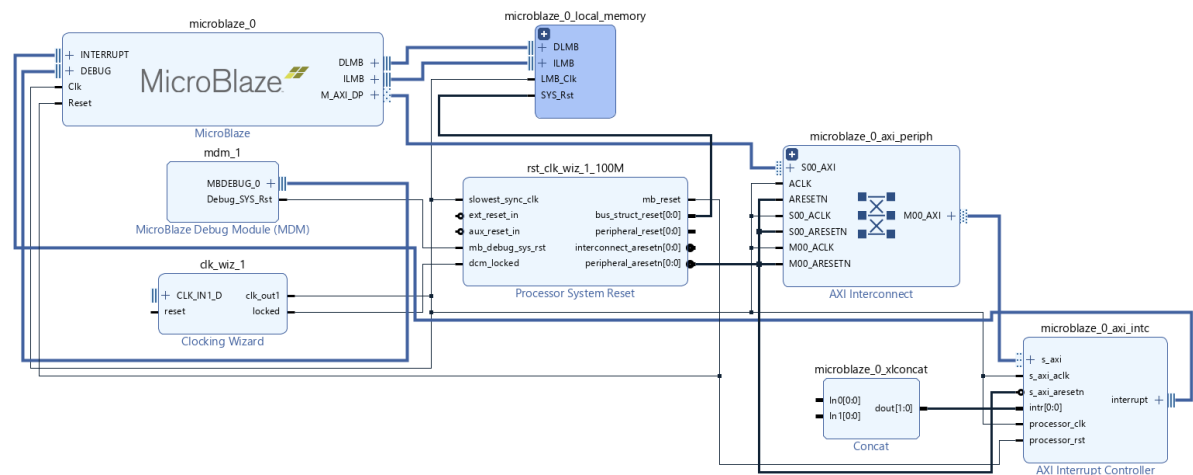
IP Integrator

- As FPGAs become larger and more complex, and as design schedules become shorter, use of third-party IP and design reuse is becoming mandatory.
- **IP Integrator** aims to aid designers with IP design and reuse issues.
- The Vivado IP integrator lets you create complex system designs by instantiating and interconnecting IP from the Vivado IP catalog on a design canvas.



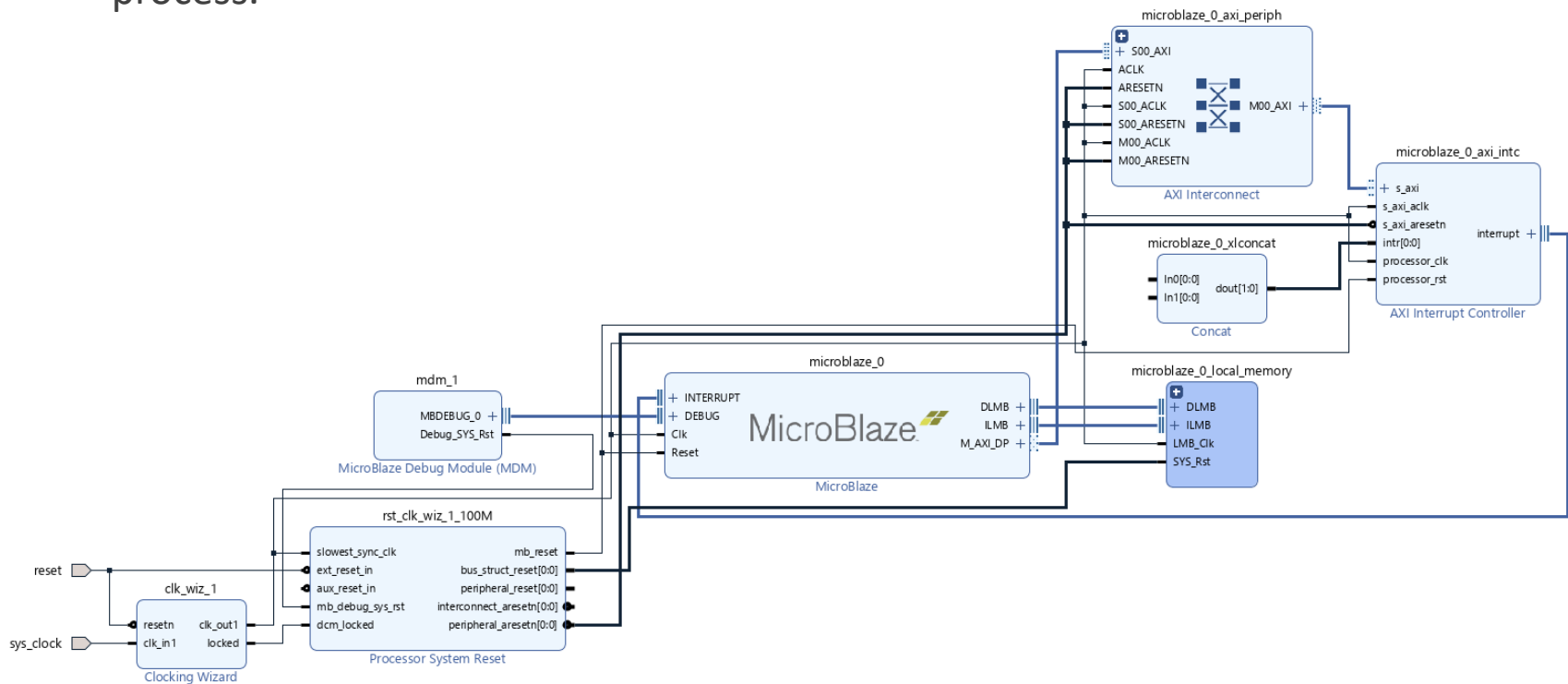
Block Automation

- Add MicroBlaze IP
- **Block Automation** assists you in putting together a basic MicroBlaze system consisting of the following:
 - A MicroBlaze debug module
 - A hierarchical block called the `microblaze_0_local_memory`
 - A clocking wizard
 - A reset module
 - An AXI interconnect
 - An AXI interrupt controller



Connection Automation

- **Connection Automation** assists you in making internal connections between different blocks and connections to external interfaces.
- Several adjustments have to be done to guide and accomplish the connection process.



External Interfaces & GPIO Core

External Interface

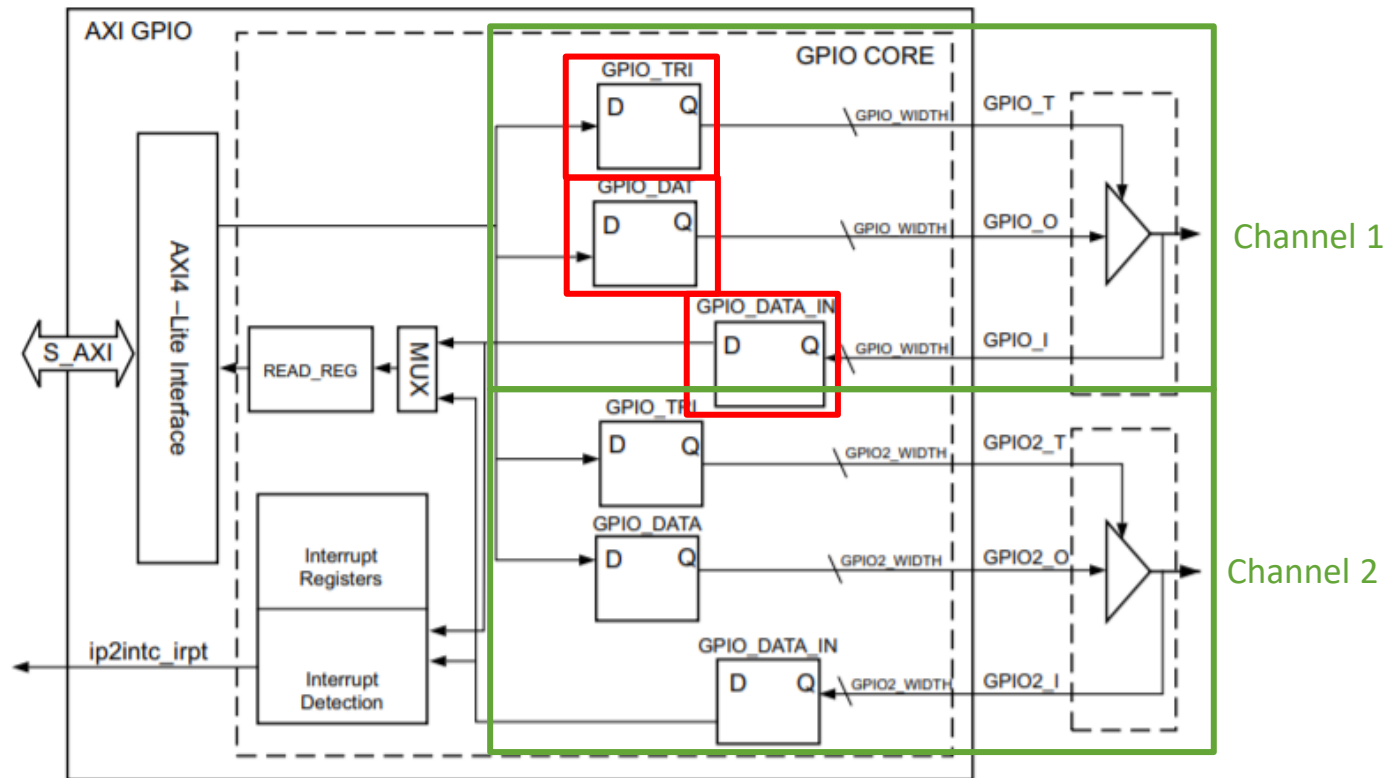
GPIO interface can be tied to one of the several interfaces present on the target board.

GPIO core provides a general purpose input/output interface to the AXI interface. This 32-bit soft IP core supports:

- the AXI4-Lite interface specification
- configurable single or dual GPIO channel(s)
- configurable channel width for GPIO pins from 1 to 32 bits
- dynamic programming of each GPIO bit as input or output
- individual configuration of each channel
- independent reset values for each bit of all registers
- optional interrupt request generation

AXI GPIO

The GPIO core consists of registers and multiplexers for reading and writing the AXI GPIO channel registers. It also includes the necessary logic to identify an interrupt event when the channel input changes.



GPIO Registers

The AXI GPIO data register is used to read the general purpose input ports and write to the general purpose output ports. When a port is configured as input, writing to the AXI GPIO data register has no effect.

The AXI GPIO 3-state control register is used to configure the ports dynamically as input or output.

- When a **bit** within this register is **set**, the corresponding I/O port is configured as an **input port**.
- When a **bit** is **cleared**, the corresponding I/O port is configured as an **output port**.

Address Space Offset ⁽³⁾	Register Name	Access Type	Default Value	Description
0x0000	GPIO_DATA	R/W	0x0	Channel 1 AXI GPIO Data Register.
0x0004	GPIO_TRI	R/W	0x0	Channel 1 AXI GPIO 3-state Control Register.
0x0008	GPIO2_DATA	R/W	0x0	Channel 2 AXI GPIO Data Register.
0x000C	GPIO2_TRI	R/W	0x0	Channel 2 AXI GPIO 3-state Control.
0x011C	GIER ⁽¹⁾	R/W	0x0	Global Interrupt Enable Register.
0x0128	IP IER ⁽¹⁾	R/W	0x0	IP Interrupt Enable Register (IP IER).
0x0120	IP ISR ⁽¹⁾	R/TOW ⁽²⁾	0x0	IP Interrupt Status Register.



GPIO Configuration

For input ports when the Interrupt is not enabled, use the following steps:

- Configure the port as input by writing the corresponding bit in **GPIOx_TRI register with the value of 1.**
- Read the corresponding bit in GPIOx_DATA register.

For output ports, use the following steps:

- Configure the port as output by writing the corresponding bit in **GPIOx_TRI register with a value of 0.**
- Write the corresponding bit in GPIOx_DATA register.

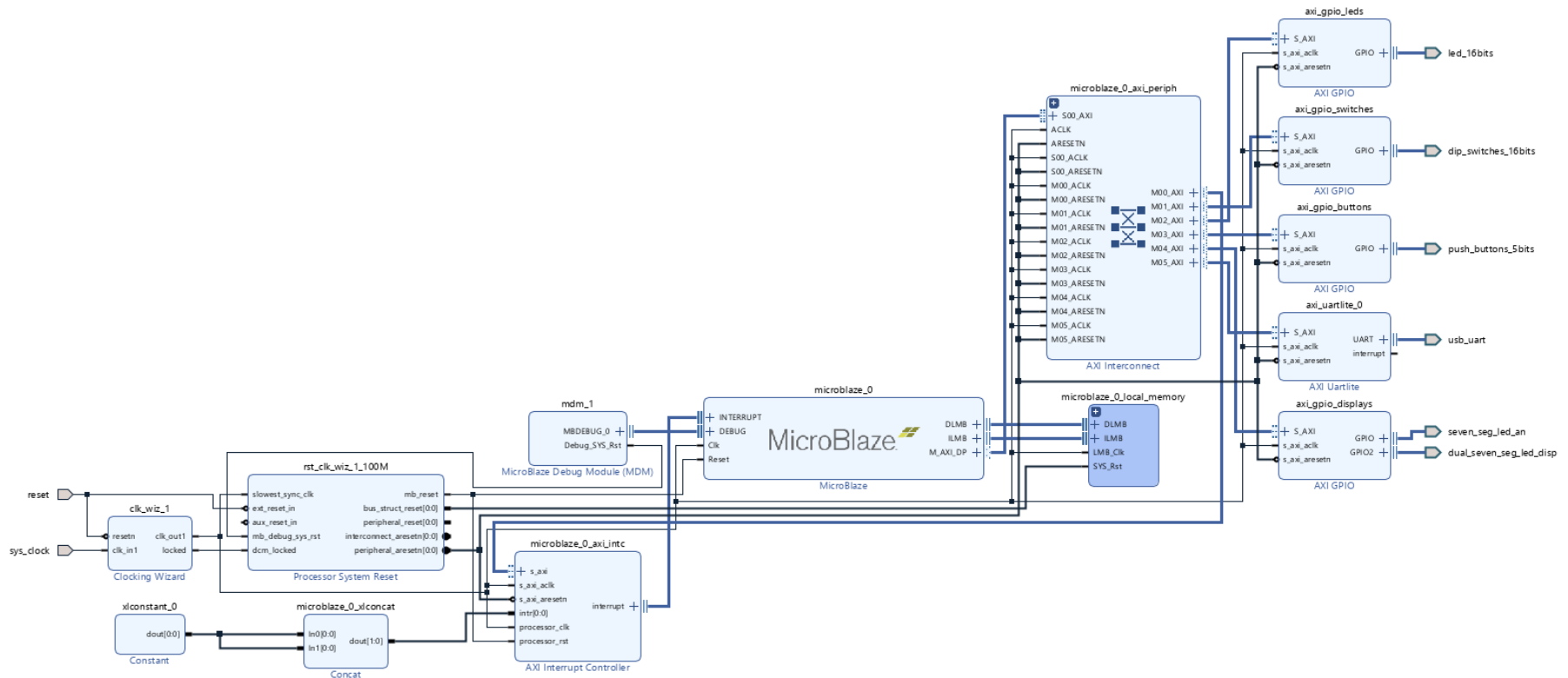
Address Editor

The address editor is a tree-table view that lists all address paths.

Diagram × Address Editor × Address Map ×						
<div><div><div>🔍</div><div>⌵</div><div>⌴</div><div>⬇</div><div>⬆</div></div><div><input checked="" type="checkbox"/> Assigned (8) <input checked="" type="checkbox"/> Unassigned (0) <input checked="" type="checkbox"/> Excluded (0) <div>Hide All</div></div></div>						
Name	Interface	Slave Segment	Master Base Address	Range	Master High Address	
Network 0						
/microblaze_0						
/microblaze_0/Data (32 address bits : 4G)						
/axi_gpio_buttons/S_AXI	S_AXI	Reg	0x4002_0000	64K ▾	0x4002_FFFF	
/axi_gpio_displays/S_AXI	S_AXI	Reg	0x4003_0000	64K ▾	0x4003_FFFF	
/axi_gpio_leds/S_AXI	S_AXI	Reg	0x4001_0000	64K ▾	0x4001_FFFF	
/axi_gpio_switches/S_AXI	S_AXI	Reg	0x4000_0000	64K ▾	0x4000_FFFF	
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K ▾	0x4060_FFFF	
/microblaze_0_axi_intc/S_AXI	s_axi	Reg	0x4120_0000	64K ▾	0x4120_FFFF	
/microblaze_0_local_memory/dlmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	64K ▾	0x0000_FFFF	
Network 1						
/microblaze_0						
/microblaze_0/Instruction (32 address bits : 4G)						
/microblaze_0_local_memory/ilmb_bram_if_cntlr/SLMB	SLMB	Mem	0x0000_0000	64K ▾	0x0000_FFFF	

End Result

Block Design (BD)



Implementing Hardware

1. Validate design

- you can run a comprehensive design check on the design.

2. Generate output products (Global)

- After the BD is complete and the design is validated, you must generate output products for synthesis and simulation, to integrate the BD into a top-level RTL design. The source files and the appropriate constraints for all the IP are generated and made available in the Vivado® Design Suite (IDE) Sources window.
- Generating the output products generates the top-level netlist of the BD. The netlist is generated in the HDL language specified by the Settings → General → Target Language for the project.

3. Create HDL Wrapper

- This command generates a top-level HDL file with an instantiation template for the IP integrator BD.

4. Generate Bitstream

Problems and Results

Open the synthesized design and execute the following TCL commands:

- `set_property CONFIG_VOLTAGE 3.3 [get_designs synth_1]`
- `set_property CFGBVS VCCO [get_designs synth_1]`

Then close the synthesized design and save an XDC file with any name

Software Development

Development Tools

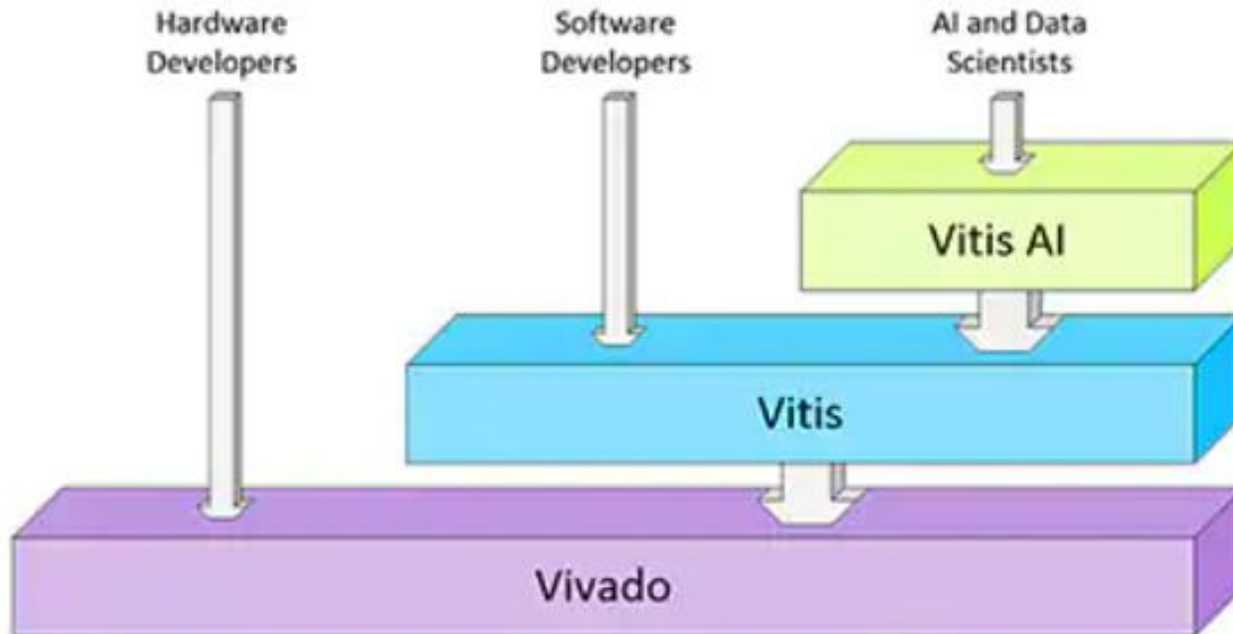


Figure 6: A high-level view of the Xilinx Vivado and Vitis design tool stack reflects how users can work with the tools at the most appropriate levels of abstraction. Hardware designers work with Vivado, software developers work with Vitis, and AI and data scientists work with Vitis AI. (Image source: Max Maxfield)

Software Development

1. Export hardware
2. Launch Vitis
 - Create a Vitis workspace
3. Create new application project
 - Create a new platform from hardware (XSA)
 - Create a “helloworld” project
4. Program device
5. Launch Vitis Serial Terminal (or any other similar application)
6. Build the project
7. Execute the software application

C code to configure GPIOs

- Command to write to GPIOs:

```
XGpio_WriteReg(<base address of GPIO device>,  
               <offset from base register to write to>,  
               <value to write>);
```

- Examples

```
XGpio_WriteReg(XPAR_AXI_GPIO_SWITCHES_BASEADDR,  
               XGPIO_TRI_OFFSET,  
               0xFFFFFFFF);
```

```
XGpio_WriteReg(XPAR_AXI_GPIO_BUTTONS_BASEADDR,  
               XGPIO_TRI_OFFSET,  
               0xFFFFFFFF);
```

- Macros exist for conveniently accessing the register addresses

Problem and Solutions

Issue: Vitis in Windows: a COM (Serial) Port is not assigned to Nexys 4.
This is observed:

- In Vitis Serial Terminal: no COM port shows up when configuring comm.
- In Windows Device Manager: no devices listed under “Ports”

Solution:

1. Open Device Manager (Gestor de Dispositivos)
2. Expand the list under “Universal Serial Bus”; look for “USB Serial Converter A” and “B”
3. If you have them, do: "**USB Serial Converter B**" > "**Properties**" > "**Advanced**" > check the "**Load VCP**" box > "OK"

Final Remarks

At the end of this lecture you should be able to:

- create a hardware platform with the MicroBlaze and GPIOs
- create and execute a simple Vitis project

To do:

- Construct the considered hardware platform
- Test the given applications in Vitis