

Introduction to MicroBlaze

Creation of the Hardware Platform

Examples of IP Cores

Software Development

IOULIIA SKLIAROVA

UNIVERSITY OF AVEIRO (PORTUGAL)

MicroBlaze Processor

Soft processor

- ~1 900-7 000 logic cells (~1 200-4 500 LUTs) - estimates
- 63 400 LUTs available in Artix-7 XC7A100T

RISC architecture

- utilizes a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures

32/64-bit architecture

- Thirty-two 32-bit or 64-bit general purpose registers

In production since 2002

Supported in

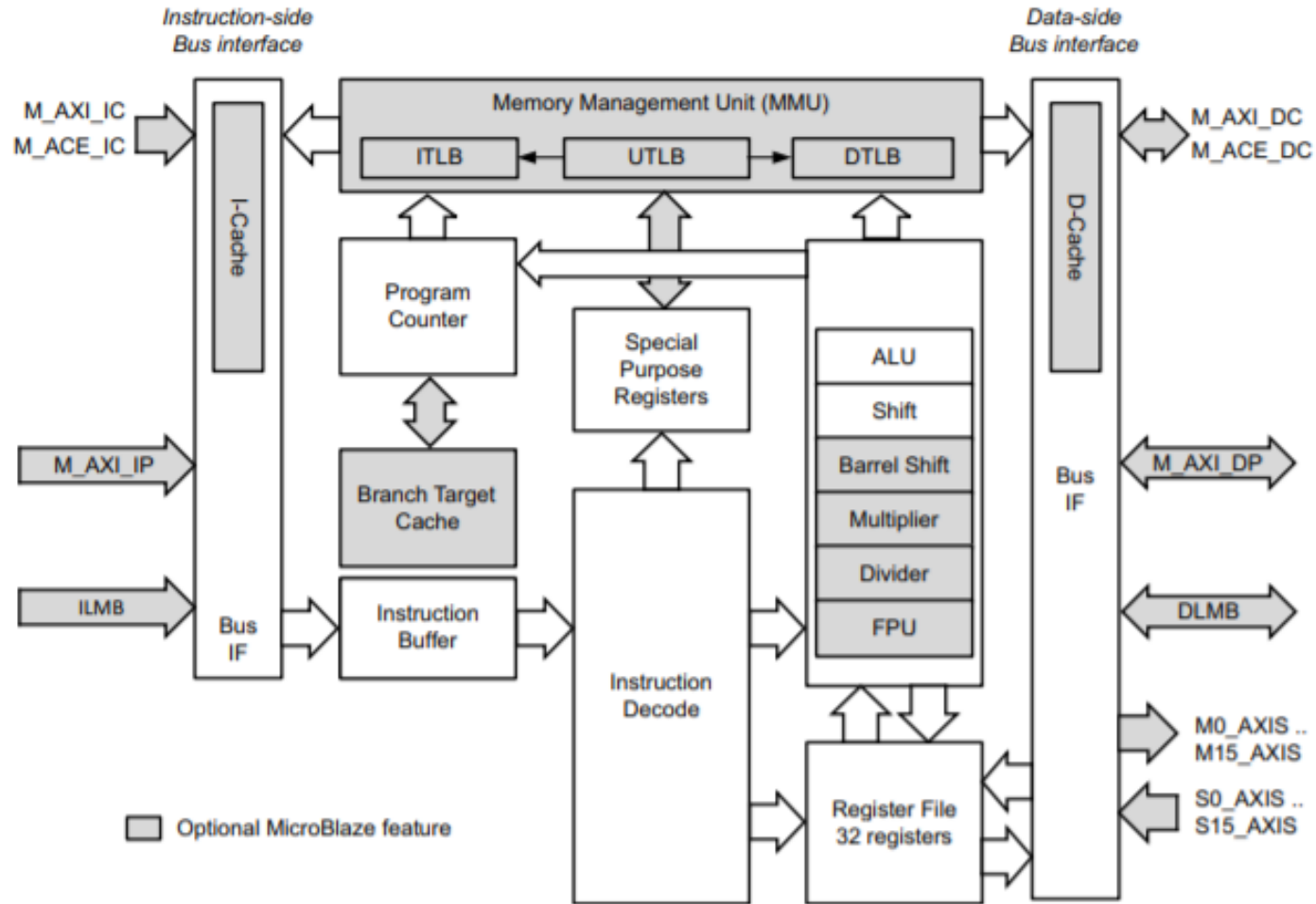
- 7-series/UltraScale/UltraScale+ devices

Three preset configurations:

- a simple microcontroller running bare-metal applications (~200 MHz in Artix-7);
- a real-time processor running FreeRTOS (~170 MHz);
- an application processor with a memory management unit running Linux (~140 MHz)

MicroBlaze Overview

MicroBlaze Processor Reference Guide - UG984



MicroBlaze Features

The fixed feature set of the processor includes:

- Thirty-two 32-bit or 64-bit general purpose registers
- 32-bit instruction word with three operands and two addressing modes
- Default 32-bit address bus, extensible to 64 bits
- Single issue pipeline

Configurable features

- Processor pipeline depth
- Floating-point unit (FPU)
- Hardware divider
- Area or speed optimized
- 64-bit mode
- ...

MicroBlaze Data Types

The MicroBlaze processor uses Big-Endian or Little-Endian (default) format to represent data, depending on the selected endianness.

The hardware supported data types for 32-bit MicroBlaze are word, half word, and byte. With 64-bit MicroBlaze the data types long and double are also available in hardware.

Big-Endian Byte Address	n	n+1	n+2	n+3
Big-Endian Byte Significance	MSByte			LSByte
Big-Endian Byte Order	n	n+1	n+2	n+3
Big-Endian Byte-Reversed Order	n+3	n+2	n+1	n
Little-Endian Byte Address	n+3	n+2	n+1	n
Little-Endian Byte Significance	MSByte			LSByte
Little-Endian Byte Order	n+3	n+2	n+1	n
Little-Endian Byte-Reversed Order	n	n+1	n+2	n+3
Bit Label	0			31
Bit Significance	MSBit			LSBit

MicroBlaze Instruction Summary

All MicroBlaze instructions are 32 bits and are defined as either Type A or Type B.

Type A instructions have up to two source register operands and one destination register operand.

Type B instructions have one source register and a 16-bit immediate operand. Type B instructions have a single destination register operand.

Table 2-7: MicroBlaze Instruction Set Summary

Type A	0-5	6-10	11-15	16-20	21-31	Semantics
Type B	0-5	6-10	11-15	16-31		
ADD Rd,Ra,Rb	000000	Rd	Ra	Rb	00L000000000	$Rd := Rb + Ra$
RSUB Rd,Ra,Rb	000001	Rd	Ra	Rb	00L000000000	$Rd := Rb + \overline{Ra} + 1$
ADDC Rd,Ra,Rb	000010	Rd	Ra	Rb	00L000000000	$Rd := Rb + Ra + C$
RSUBC Rd,Ra,Rb	000011	Rd	Ra	Rb	00L000000000	$Rd := Rb + \overline{Ra} + C$
ADDK Rd,Ra,Rb	000100	Rd	Ra	Rb	00L000000000	$Rd := Rb + Ra$
RSUBK Rd,Ra,Rb	000101	Rd	Ra	Rb	00L000000000	$Rd := Rb + \overline{Ra} + 1$
CMP Rd,Ra,Rb	000101	Rd	Ra	Rb	00L000000001	$Rd := Rb + \overline{Ra} + 1$ Rd[0] := 0 if (Rb >= Ra) else Rd[0] := 1
CMPU Rd,Ra,Rb	000101	Rd	Ra	Rb	00L000000011	$Rd := Rb + \overline{Ra} + 1$ (unsigned) Rd[0] := 0 if (Rb >= Ra, unsigned) else Rd[0] := 1

MicroBlaze Pipeline

MicroBlaze instruction execution is pipelined.

For most instructions, each stage takes one clock cycle to complete.

Consequently, the number of clock cycles necessary for a specific instruction to complete is equal to the number of pipeline stages, and one instruction is completed on every cycle in the absence of data, control or structural hazards.

- A data hazard occurs when the result of an instruction is needed by a subsequent instruction. This can result in stalling the pipeline, unless the result can be forwarded to the subsequent instruction. The MicroBlaze GNU Compiler attempts to avoid data hazards by reordering instructions during optimization.
- A control hazard occurs when a branch is taken, and the next instruction is not immediately available. This results in stalling the pipeline. MicroBlaze provides delay slot branches and the optional branch target cache to reduce the number of stall cycles.
- A structural hazard occurs for a few instructions that require multiple clock cycles in the execute stage or a later stage to complete. This is achieved by stalling the pipeline.

Three Stage Pipeline

With the MicroBlaze is optimized for **area**, the pipeline is divided into three stages to minimize hardware cost: Fetch, Decode, and Execute.

The three stage pipeline does not have any data hazards.

Pipeline stalls are caused by control hazards, structural hazards due to multi-cycle instructions, memory accesses using slower memory, instruction fetch from slower memory, or stream accesses.

	cycle1	cycle2	cycle3	cycle4	cycle5	cycle6	cycle7
instruction 1	Fetch	Decode	Execute				
instruction 2		Fetch	Decode	Execute	Execute	Execute	
instruction 3			Fetch	Decode	Stall	Stall	Execute

Five Stage Pipeline

With the MicroBlaze is optimized for **performance**, the pipeline is divided into five stages to maximize performance: Fetch (IF), Decode (OF), Execute (EX), Access Memory (MEM), and Writeback (WB).

Pipeline stalls are caused by data hazards, control hazards, structural hazards due to multicycle instructions, memory accesses using slower memory, instruction fetch from slower memory, or stream accesses.

	cycle1	cycle2	cycle3	cycle4	cycle5	cycle6	cycle7	cycle8	cycle9
instruction 1	IF	OF	EX	MEM	WB				
instruction 2		IF	OF	EX	MEM	MEM	MEM	WB	
instruction 3			IF	OF	EX	Stall	Stall	MEM	WB

Eight Stage Pipeline

With the MicroBlaze is optimized for **frequency**, the pipeline is divided into eight stages to maximize possible frequency: Fetch (IF), Decode (OF), Execute (EX), Access Memory 0 (M0), Access Memory 1 (M1), Access Memory 2 (M2), Access Memory 3 (M3) and Writeback (WB).

Pipeline stalls are caused by data hazards, control hazards, structural hazards, memory accesses using slower memory, instruction fetch from slower memory, or stream accesses.

	cycle1	cycle2	cycle3	cycle4	cycle5	cycle6	cycle7	cycle8	cycle9	cycle10	cycle11
instruction 1	IF	OF	EX	M0	M1	M2	M3	WB			
instruction 2		IF	OF	EX	M0	M0	M1	M2	M3	WB	
instruction 3			IF	OF	EX	Stall	M0	M1	M2	M3	WB

Memory Architecture

MicroBlaze is implemented with a **Harvard memory architecture**; instruction and data accesses are done in separate address spaces.

- The **instruction address space** has a 32-bit virtual address range with 32-bit MicroBlaze (that is, handles up to 4GB of instructions), and can be extended up to a 64-bit physical address range.
- The **data address space** has a default 32-bit range, and can be extended up to a 64-bit range (that is, handles from 4GB to 16EB of data).

The instruction and data memory ranges can be made to overlap by mapping them both to the same physical memory. The latter is necessary for software debugging.

Both instruction and data interfaces of MicroBlaze are default 32 bits wide and use big endian or little endian, bit-reversed format, depending on the selected endianness. MicroBlaze supports word, halfword, and byte accesses to data memory.

Data accesses must be aligned (word accesses must be on word boundaries, halfword on halfword boundaries), unless the processor is configured to support unaligned exceptions. All instruction accesses must be word aligned.

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.

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.

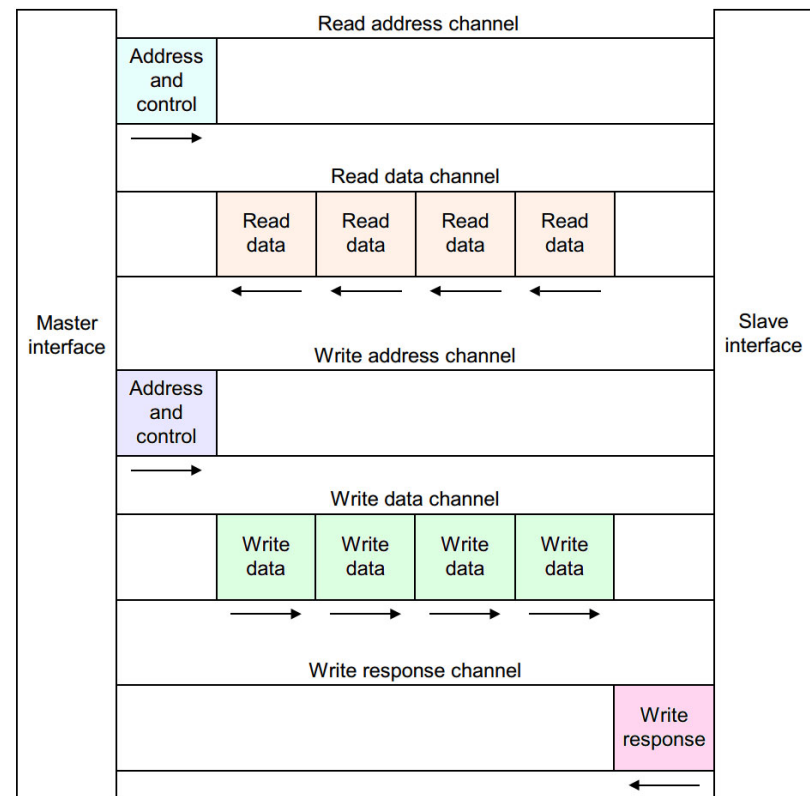
AXI4 and AXI4-Lite Channels

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.



Debug Overview

MicroBlaze features a debug interface to support JTAG based software debugging tools.

The debug interface is designed to be connected to the Xilinx Microprocessor Debug Module (MDM) core, which interfaces with the JTAG port of Xilinx FPGAs.

To be able to download programs, set software breakpoints and disassemble code, the instruction and data memory ranges must overlap, and use the same physical memory.

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)

Board File Installation

Configure the board:

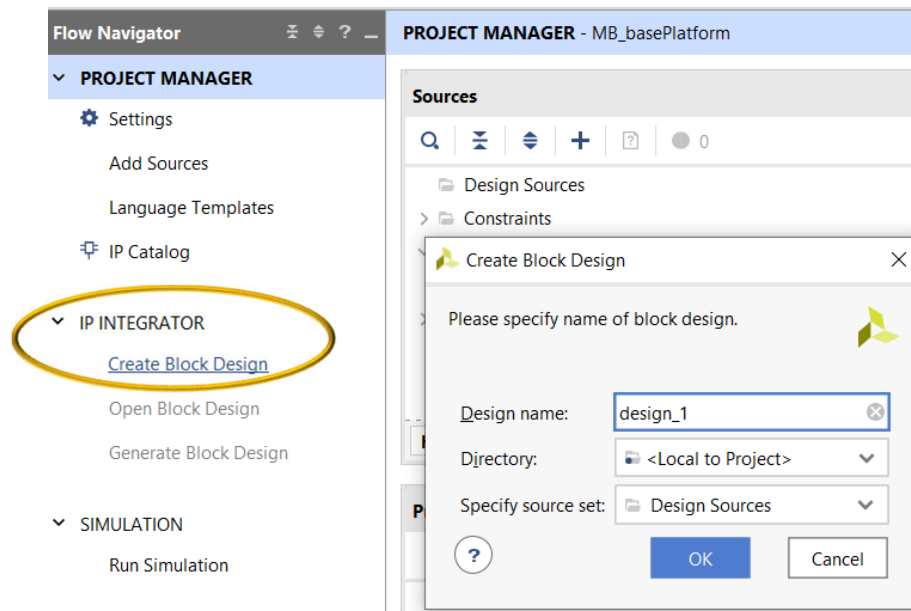
- Do not use Install/Update feature when creating a new project
- Download the board files from https://sweet.ua.pt/iouliia/Courses/MB_TUT/nexys4_ddr.rar
- Copy nexys4_ddr folder to Vivado installation folder (C:\Xilinx\Vivado\2022.2\data\boards\board_files)
 - By default this folder contains XML files 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-4DDR board

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.

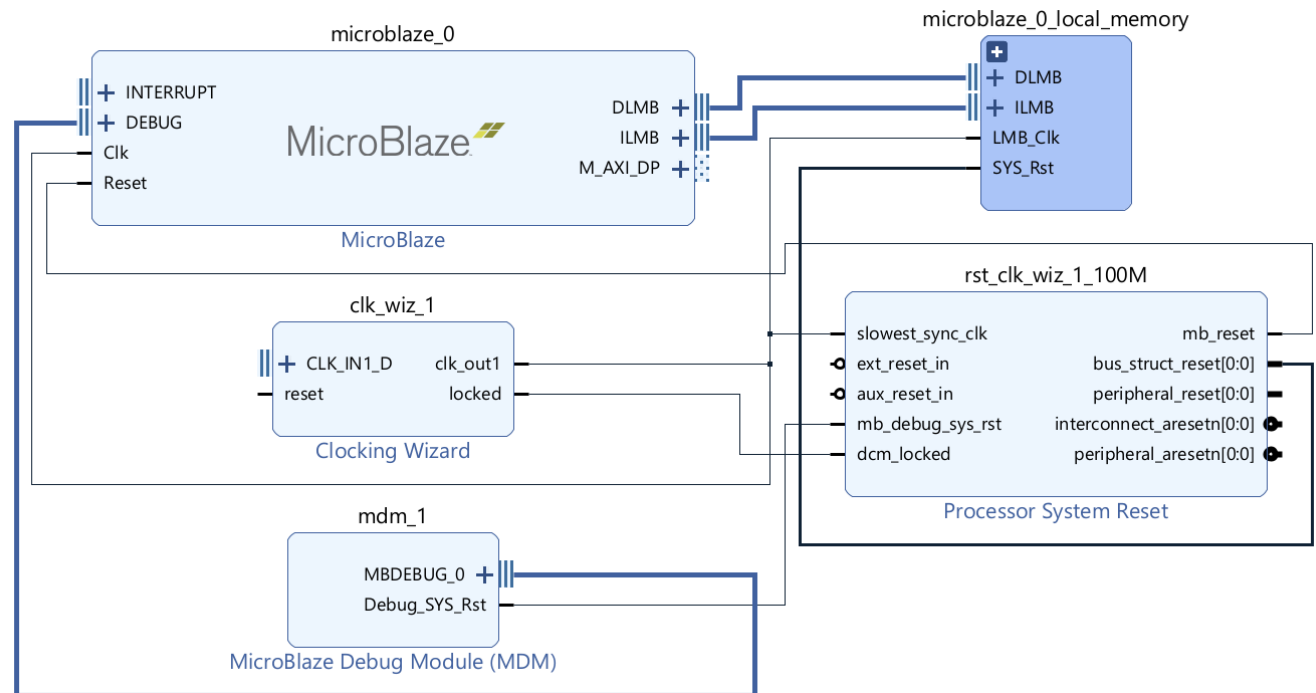


Block Automation

Add MicroBlaze IP

Block Automation assists you in putting together a basic MicroBlaze system consisting of the following:

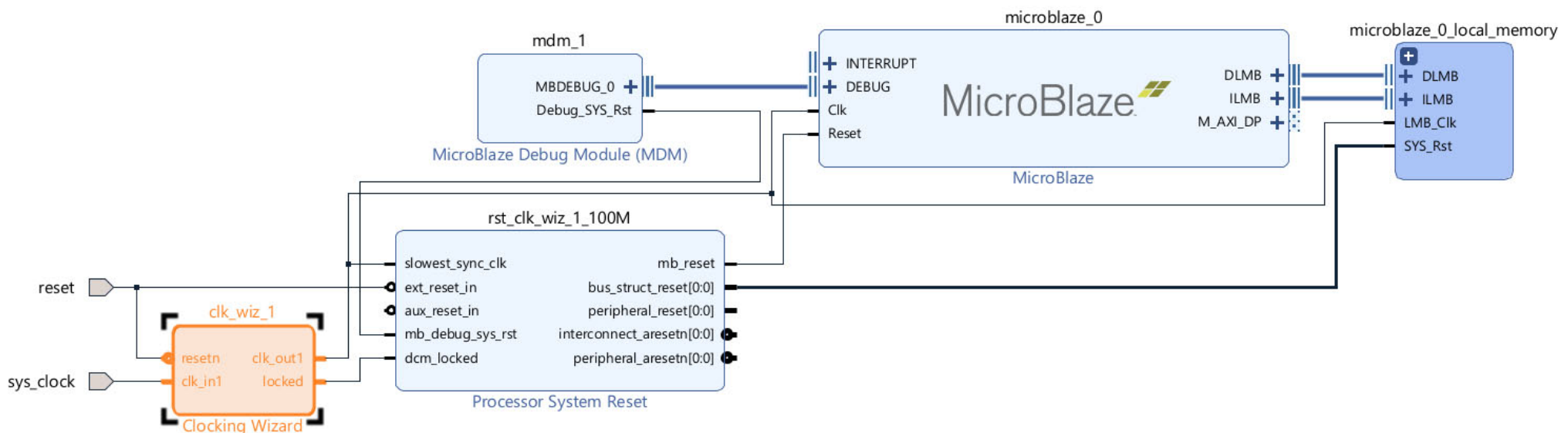
- A MicroBlaze debug module
- A hierarchical block called the `microblaze_0_local_memory`
- A clocking wizard
- A reset module



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 Interface

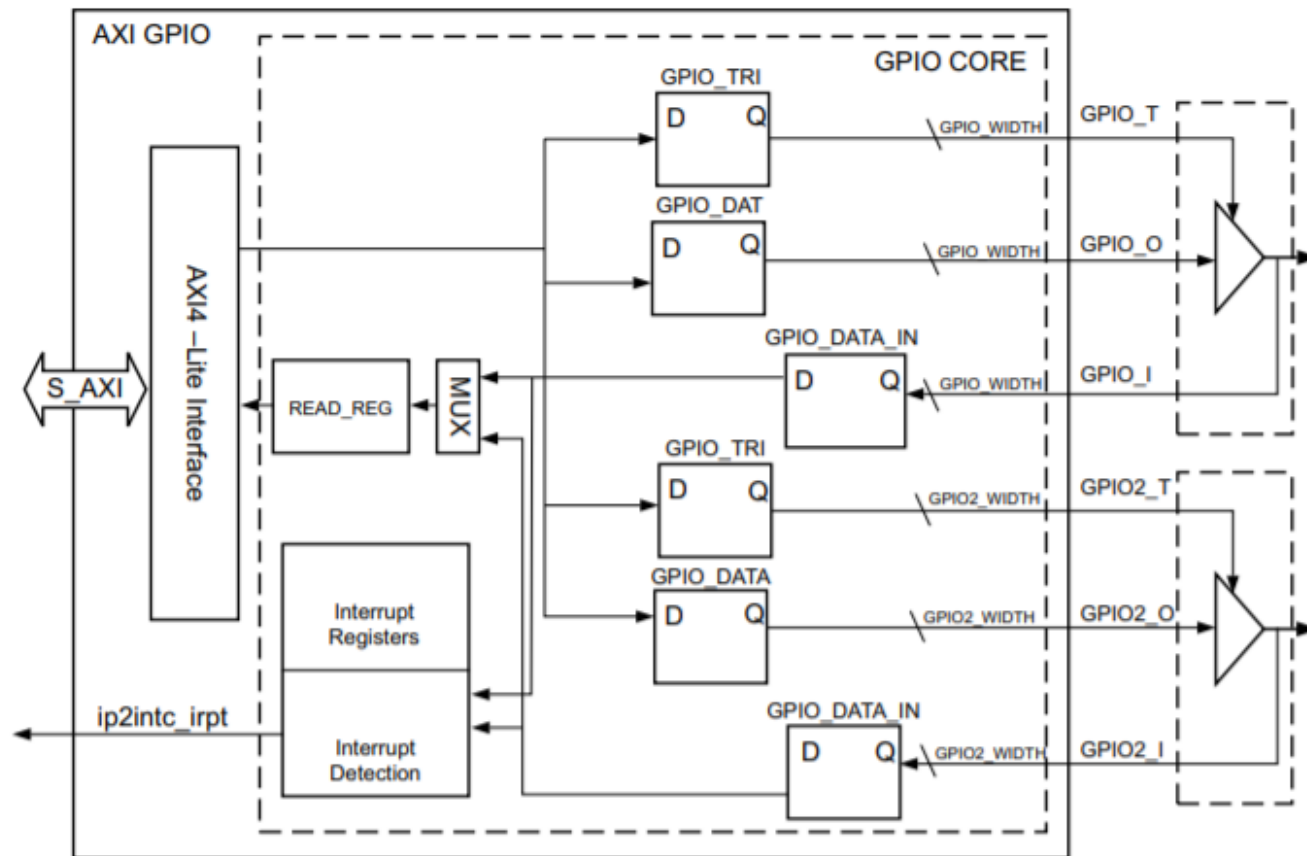
GPIO interface can be tied to one of the several interfaces present on the target board.

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:

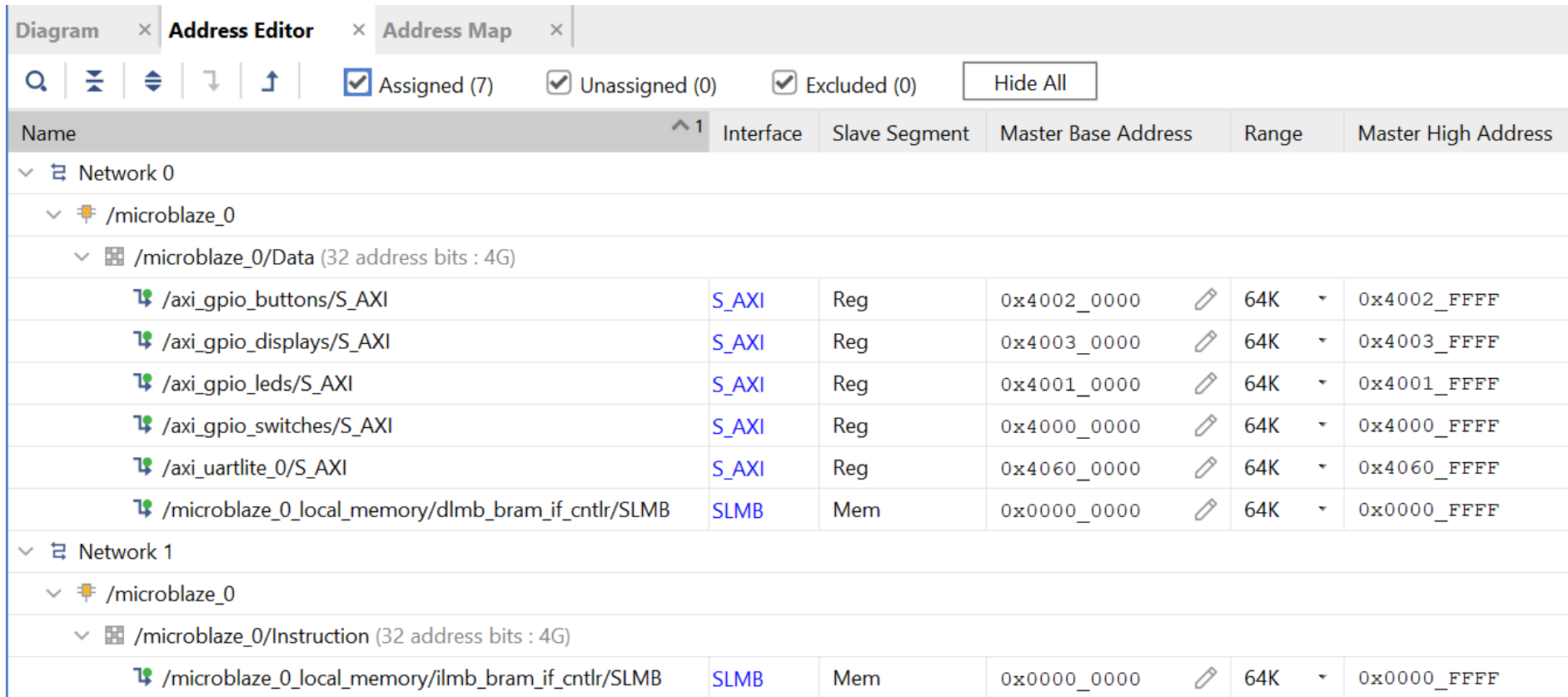
- Configure the port as input by writing the corresponding bit in GPIOx_TRI register with the value of 1.
- Read the corresponding bit in GPIOx_DATA register.

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

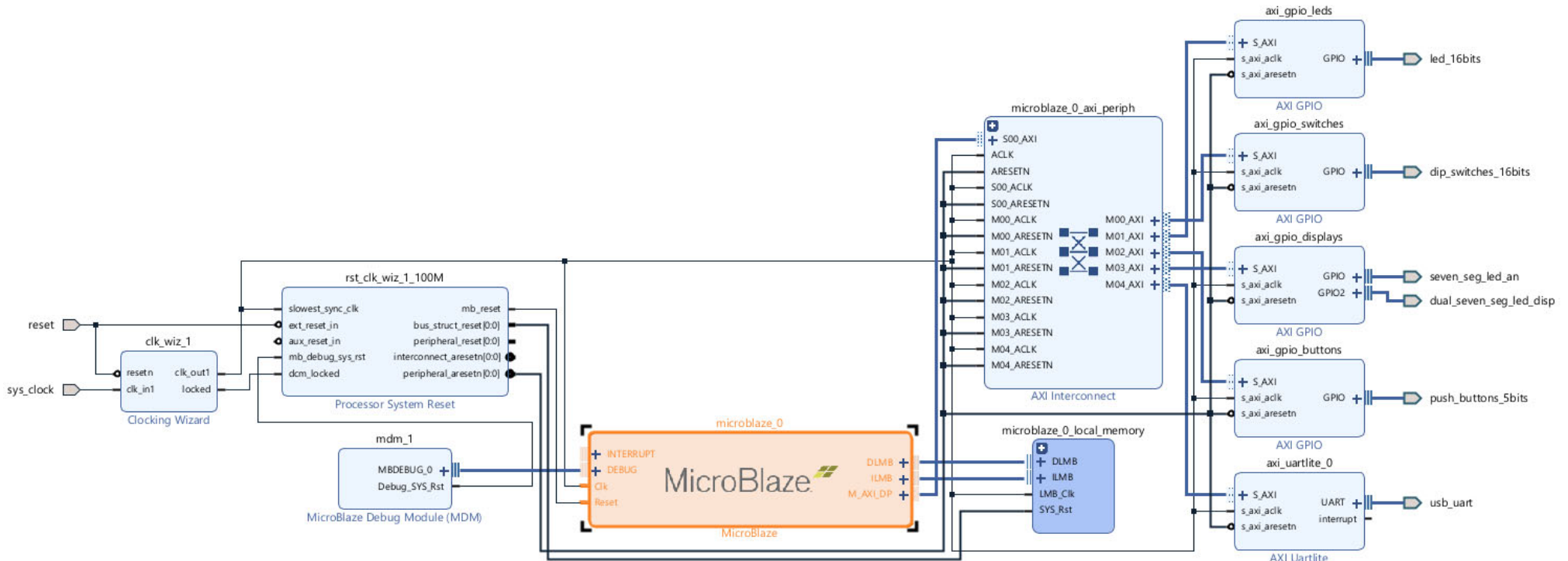
The address editor is a tree-table view that lists all address paths.



The screenshot shows the Address Editor interface with a tree-table view. The interface includes a search bar, filter buttons for Assigned (7), Unassigned (0), and Excluded (0), and a Hide All button. The table lists address paths for two networks, Network 0 and Network 1, under the /microblaze_0 directory. The table columns are Name, Interface, Slave Segment, Master Base Address, Range, and Master High Address.

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_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

Block Design (BD)



Implementing Hardware

Validate design

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

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.

Create HDL Wrapper

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

Synthesize design

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 VCC0 [get_designs synth_1]`

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

Generate bitstream

Development Tools

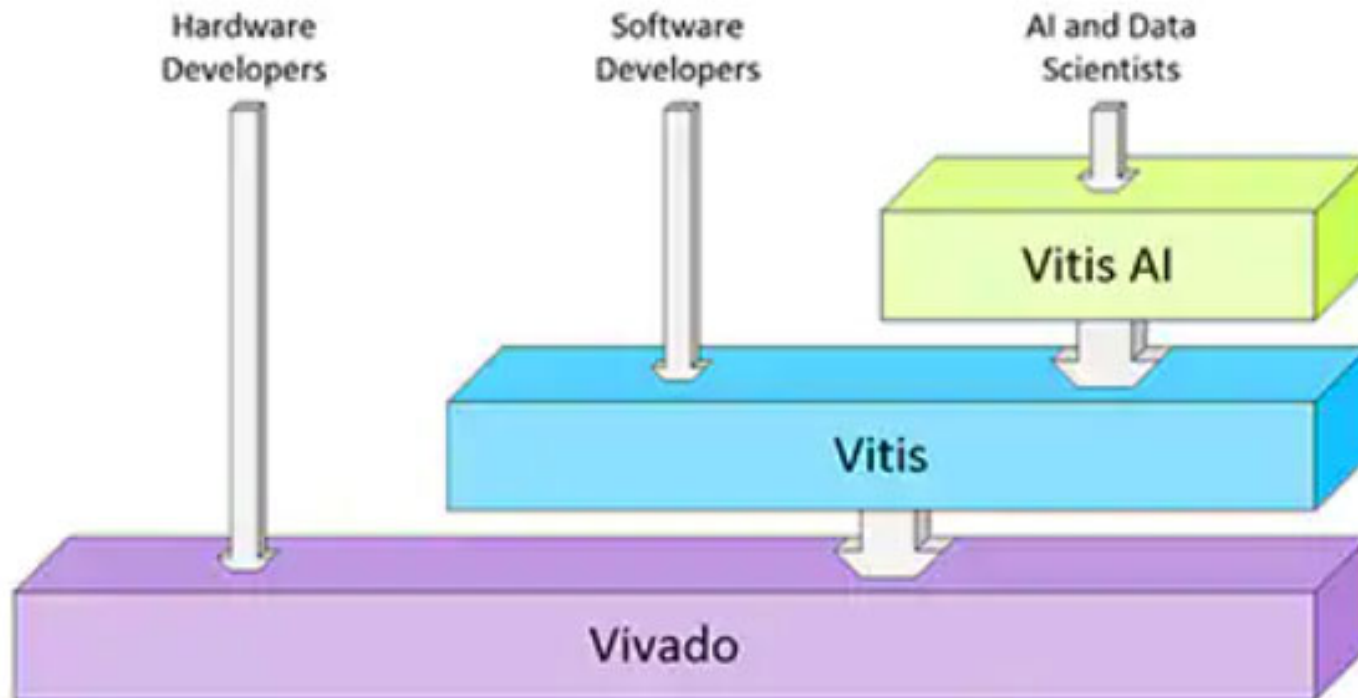


Figure 6: A high-level view of the Xilinx Vivado and Vitis design tool stack reflects how users can work with the tools at the most appropriate levels of abstraction. Hardware designers work with Vivado, software developers work with Vitis, and AI and data scientists work with Vitis AI. (Image source: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

Software Development

Export hardware

Launch Vitis

- Create a Vitis workspace

Create new application project

- Create a new platform from hardware (XSA)
- Create a “helloworld” project

Program device

Launch Vitis serial terminal (or any other similar application)

Build the project

Execute the software application

Final Remarks

At the end of this lecture and lab 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