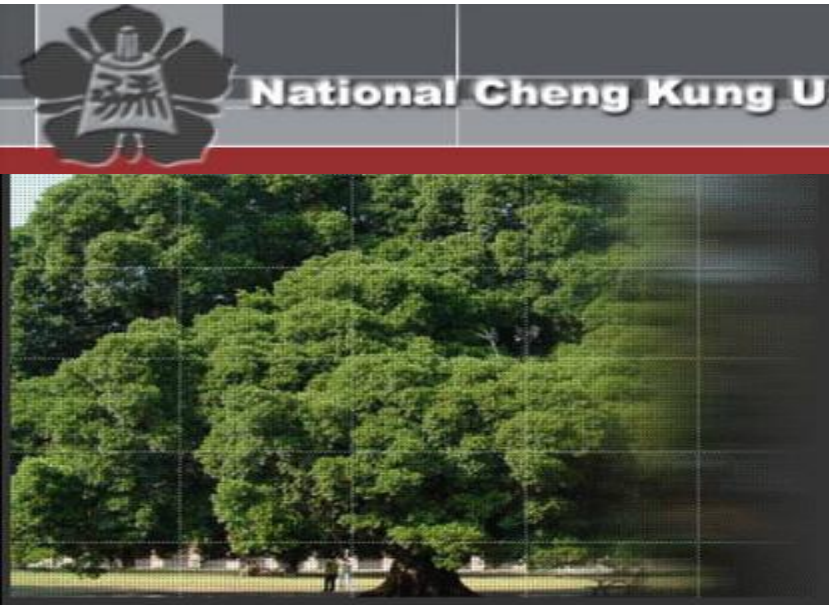


RISC-V Microcontroller

Developed by Wei-Feng Huang, DICLAB, NCKU

Microcontroller Selection

- E203
  - RV32IM ISA with interrupt and exception support
  - Developed by Nuclei System Technology (China)
  - FreeRTOS support available
- MicroBlaze V
  - Xilinx
  - Configurable ISA with interrupt and exception support
  - JTAG support available
  - FreeRTOS support planned for 2026
  - Strong toolchain support (Vivado/Vitis integration, beginner-friendly)
- DICLAB's CPU (developing)
  - RV32IMFB ISA with interrupt and exception support
  - FreeRTOS support targeted

MicroBlaze V high-end Peripheral

General Purpose

- Multichannel DMA
- Streaming FIFO
- Timer / Watchdog
- Mutex / Mailbox

I/O

- UART
- USB 2.0
- SPI
- GPIO
- PWM

Video

- HDMI Camera/Display Interface
- MIPI-CSI, MIPI-DSI
- Video DMA

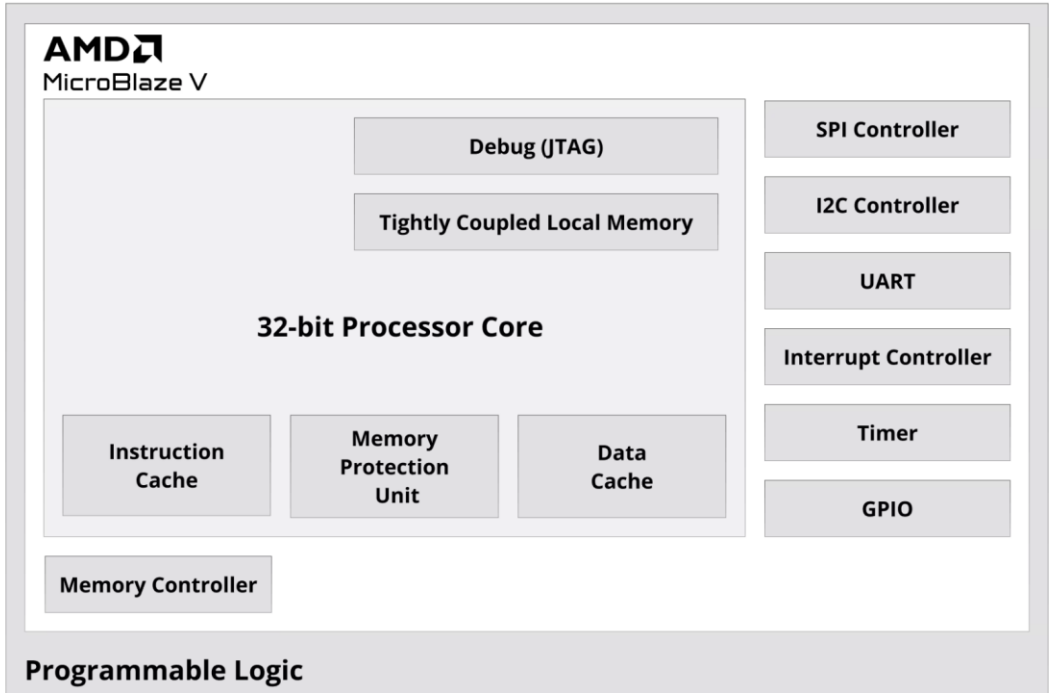
Memory

- DDR
- Quad SPI
- SDRAM

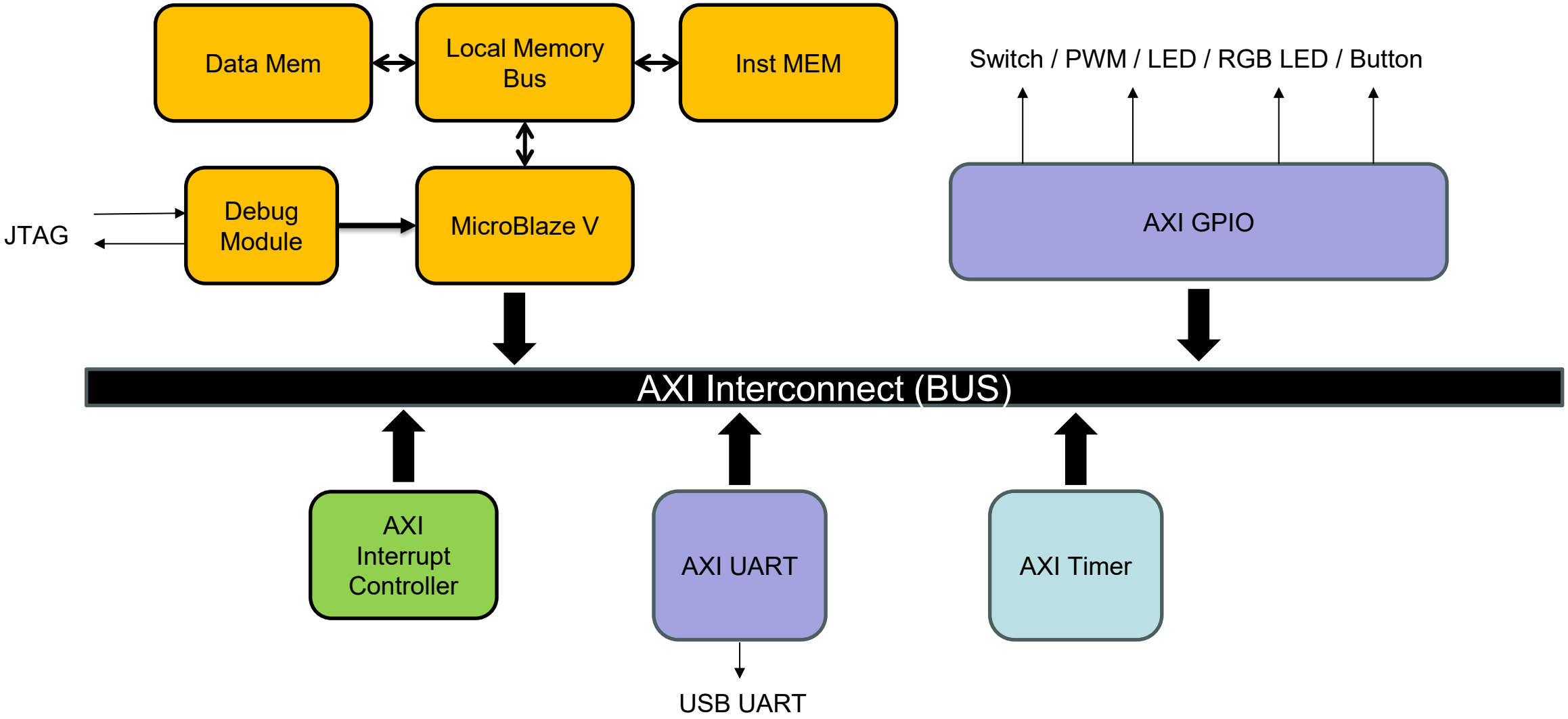
Networking

- Ethernet Subsystem
- Controller Area Network

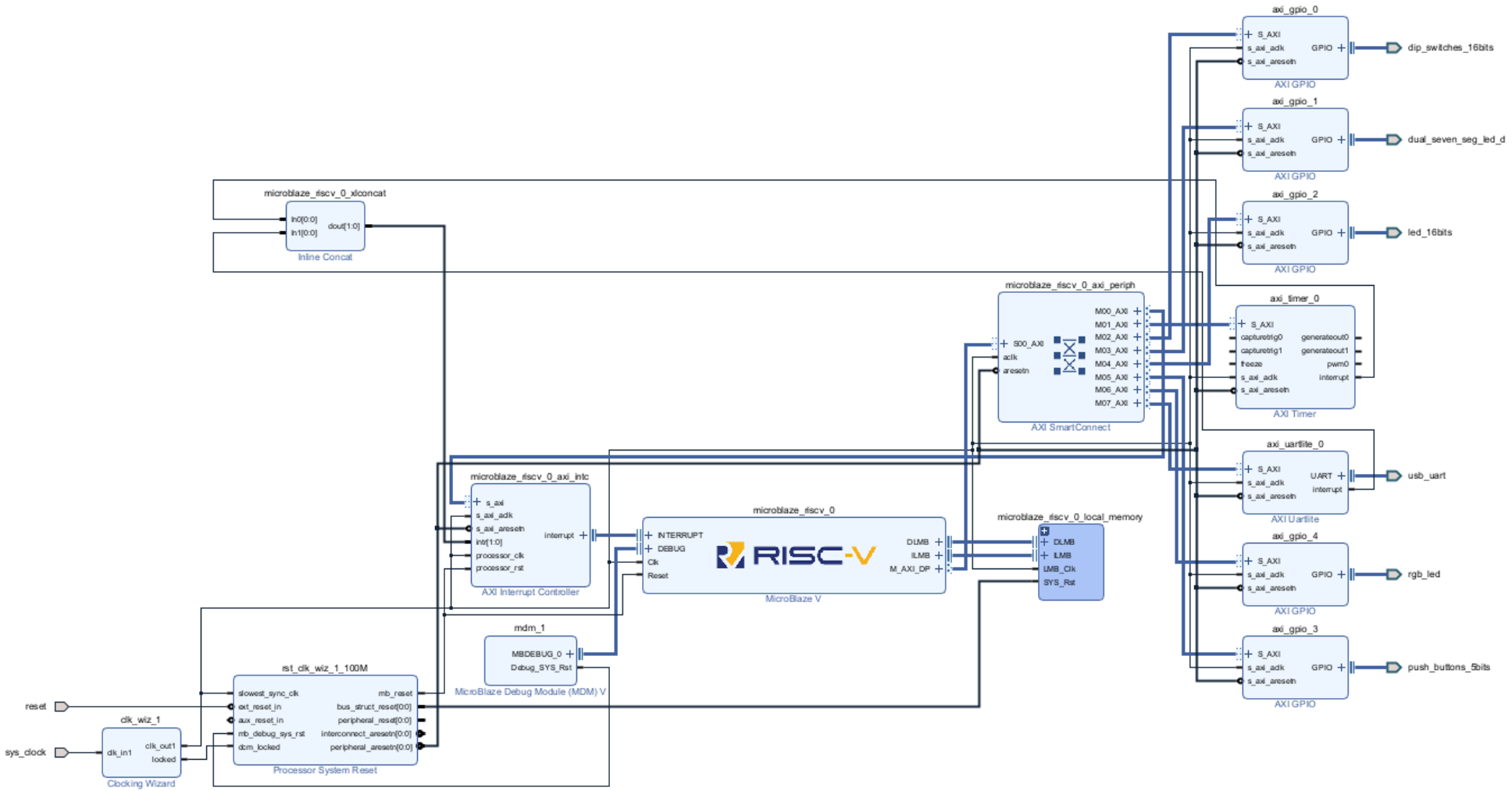
- External I/O is FPGA-configurable via GPIO ports.
- Onboard switches/LEDs/buttons are also supported.
- The right figure shows a Xilinx MicroBlaze V microcontroller reference design example.



Our Microcontroller (temporary)



Block Design



Memory-Mapped I/O vs SFR

- Memory-Mapped I/O (MMIO):
  - Peripherals are mapped into the memory address space
  - Accessed using normal load/store instructions
  - Easy to integrate and extend (add new devices by assigning new addresses)

- SFR (Special Function Registers) in traditional MCUs (e.g., PIC):
  - Peripheral control uses dedicated special registers
  - Often requires bank selection / bit-level control
  - Less flexible when scaling to more peripherals

Memory Map in our microcontroller

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0 (/microblaze_riscv_0/Data)					
/microblaze_riscv_0					
/microblaze_riscv_0/Data (32 address bits : 4G)					
/axi_gpio_0/S_AXI	S_AXI	Reg	0x4000_0000	64K	0x4000_FFFF
/axi_gpio_1/S_AXI	S_AXI	Reg	0x4001_0000	64K	0x4001_FFFF
/axi_gpio_2/S_AXI	S_AXI	Reg	0x4002_0000	64K	0x4002_FFFF
/axi_gpio_3/S_AXI	S_AXI	Reg	0x4003_0000	64K	0x4003_FFFF
/axi_gpio_4/S_AXI	S_AXI	Reg	0x4004_0000	64K	0x4004_FFFF
/axi_timer_0/S_AXI	S_AXI	Reg	0x41C0_0000	64K	0x41C0_FFFF
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x4060_0000	64K	0x4060_FFFF
/microblaze_riscv_0_axi_intc/S_AXI	s_axi	Reg	0x4120_0000	64K	0x4120_FFFF
/microblaze_riscv_0_local_memory/dlmb_bram_if_cntlr/S	SLMB	Mem	0x0	64K	0xFFFF
Network 1 (/microblaze_riscv_0/Instruction)					
/microblaze_riscv_0					
/microblaze_riscv_0/Instruction (32 address bits : 4G)					
/microblaze_riscv_0_local_memory/ilmb_bram_if_cntlr/S	SLMB	Mem	0x0	64K	0xFFFF

Registers

There are four internal registers in the AXI GPIO design as shown in Table 4. The memory map of the AXI GPIO design is determined by setting the C_BASEADDR parameter. The internal registers of the AXI GPIO are at a fixed offset from the base address and are byte accessible.

Table 4: **Registers**

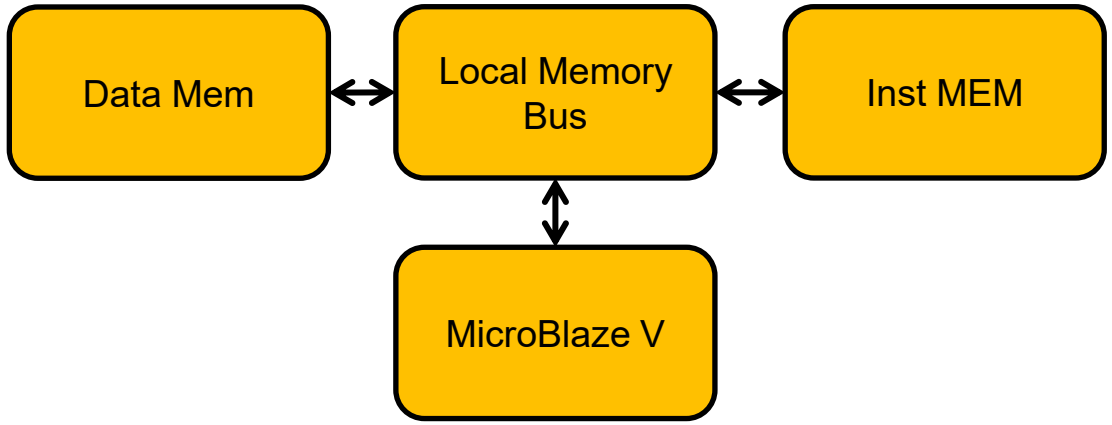
Base Address + Offset (hex)	Register Name	Access Type	Default Value (hex)	Description
C_BASEADDR + 0x00	GPIO_DATA	Read/Write	0x0	Channel 1 AXI GPIO Data Register
C_BASEADDR + 0x04	GPIO_TRI	Read/Write	0x0	Channel 1 AXI GPIO 3-state Register
C_BASEADDR + 0x08	GPIO2_DATA	Read/Write	0x0	Channel 2 AXI GPIO Data Register
C_BASEADDR + 0x0C	GPIO2_TRI	Read/Write	0x0	Channel 2 AXI GPIO 3-state Register

- Each mapped IP provides its own datasheet / product guide.
- The document specifies the internal register map (offsets from the base address) and explains how to use each register for configuration and I/O control.

Memory Architecture

CPU internal (MicroBlaze core):

MicroBlaze uses a Harvard-style internal structure, with separate instruction and data paths.



Modified Harvard Architecture

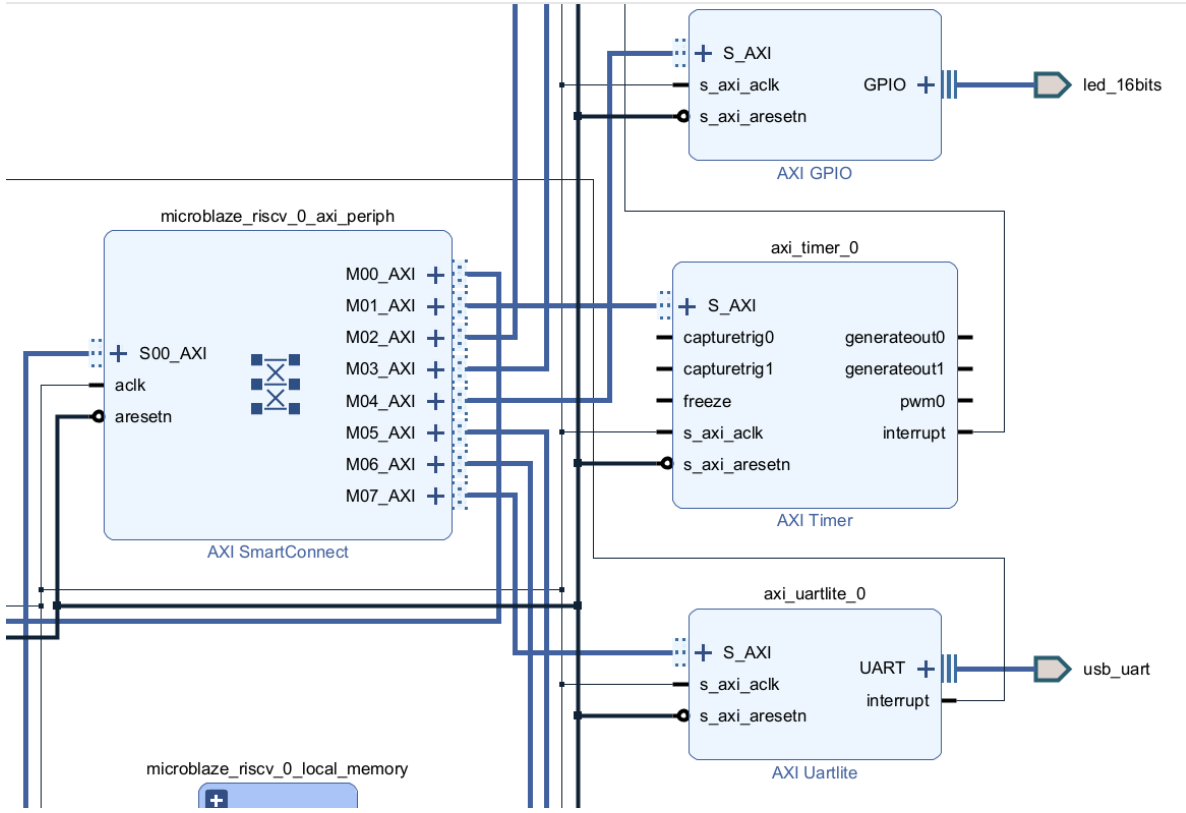
SOC Level:

The overall system looks more like a **von Neumann model**, because both instructions and data are placed in a **single unified memory address space** when accessing external memory and memory-mapped peripherals.

AXI Mapping

- Although each peripheral is mapped to a memory address range, this address region does not correspond to physical RAM.

- Instead, AXI address decoding and routing logic forwards these transactions to the target peripheral's register interface (MMIO).



Resource Utilization

Resource	Utilization	Available	Utilization %
LUT	4117	63400	6.49
LUTRAM	139	19000	0.73
FF	3621	126800	2.86
BRAM	16	135	11.85
DSP	4	240	1.67
IO	55	210	26.19
BUFG	6	32	18.75
MMCM	1	6	16.67

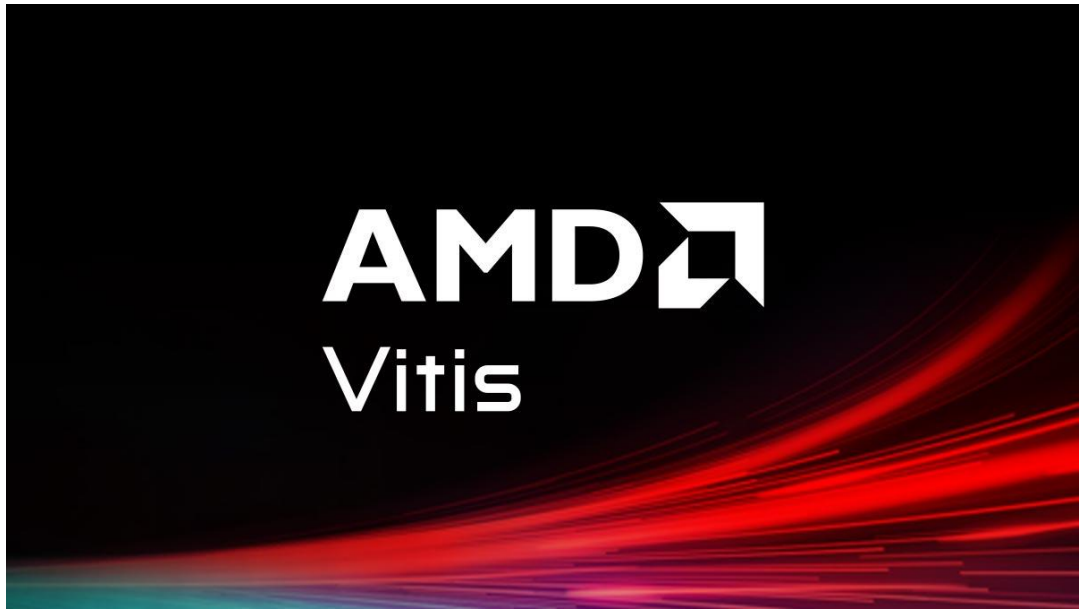
Spartan-7 FPGAs

		I/O Optimization at the Lowest Cost and Highest Performance-per-Watt (1.0V, 0.95V)					
	Part Number	**XC7S6**	**XC7S15**	**XC7S25**	**XC7S50**	**XC7S75**	**XC7S100**
Logic Resources	Logic Cells	6,000	12,800	23,360	52,160	76,800	102,400
	Slices	938	2,000	3,650	8,150	12,000	16,000
	CLB Flip-Flops	7,500	16,000	29,200	65,200	96,000	128,000
Memory Resources	Max. Distributed RAM (Kb)	70	150	313	600	832	1,100
	Block RAM/FIFO w/ ECC (36 Kb each)	5	10	45	75	90	120
	Total Block RAM (Kb)	180	360	1,620	2,700	3,240	4,320
Clock Resources	Clock Mgmt Tiles (1 MMCM + 1 PLL)	2	2	3	5	8	8
I/O Resources	Max. Single-Ended I/O Pins	100	100	150	250	400	400
	Max. Differential I/O Pairs	48	48	72	120	192	192
Embedded Hard IP Resources	DSP Slices	10	20	80	120	140	160
	Analog Mixed Signal (AMS) / XADC	0	0	1	1	1	1
	Configuration AES / HMAC Blocks	0	0	1	1	1	1
Speed Grades	Commercial Temp (C)	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2
	Industrial Temp (I)	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L
	Expanded Temp (Q)	-1	-1	-1	-1	-1	-1

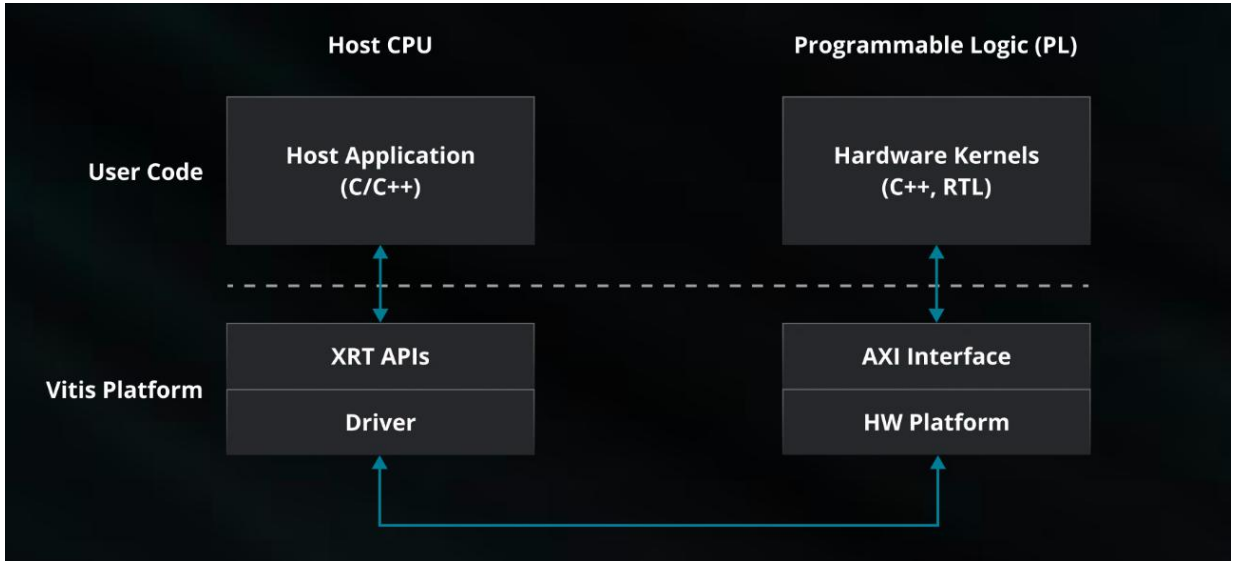
→ =1.6 * LUT

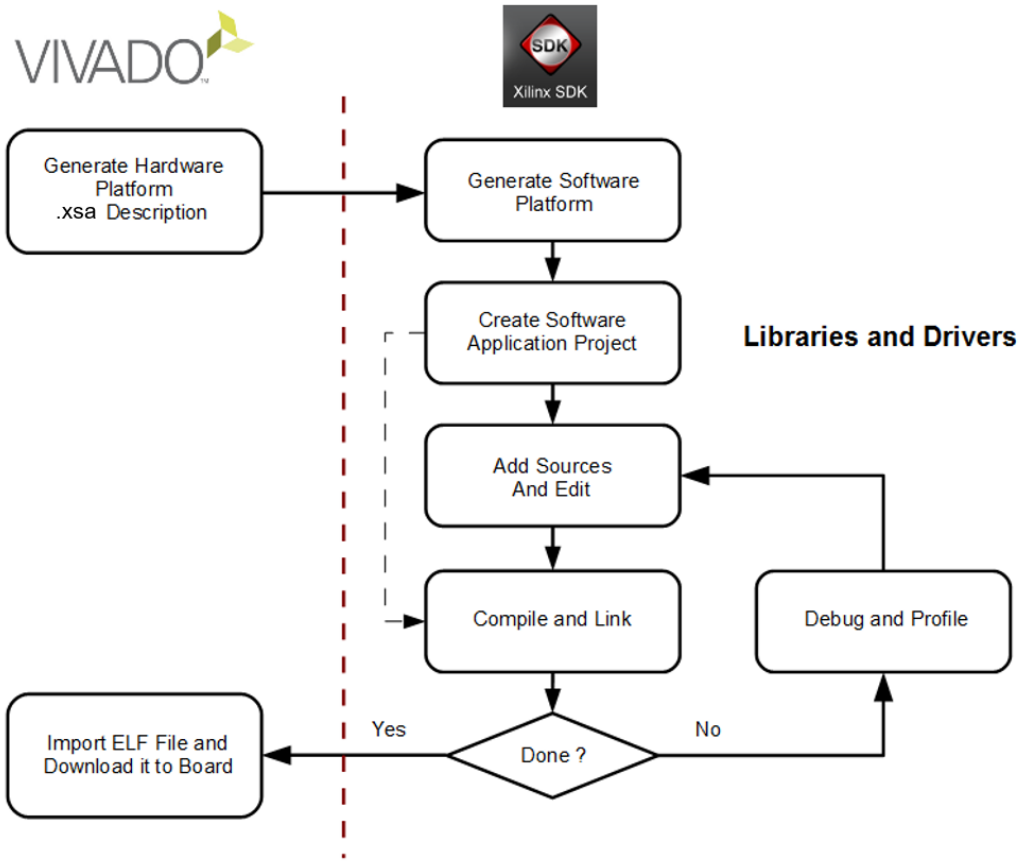
Tool Chain

- Vivado for FPGA Design (RTL → Bitstream)
- Vitis for Software Development (C/C++ → ELF)



Design Flow





Program to FPGA and Debug - XSDB (via JTAG)

- XSDB (Xilinx System Debugger) communicates with the target board through JTAG (IEEE 1149.1), a standard interface for device programming and hardware debugging.
- It uses the MicroBlaze Debug Module (MDM) to halt/reset the CPU and access system memory.
- XSDB then downloads the .elf sections (e.g. .text, .data) into on-chip BRAM according to the ELF memory map.
- Finally, XSDB sets the program entry point (PC) and starts CPU execution.