Part 1: Board Setup & GPIO

Erwin

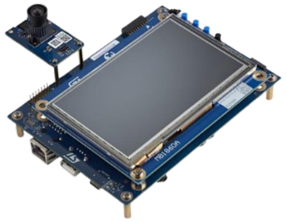
Introduction


AI is everywhere
Warung Makan Indomie
WARMINDO
WARKOP AI
Minuman

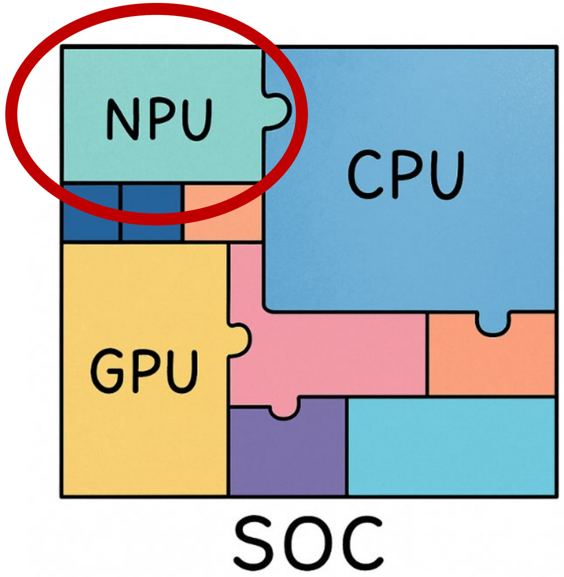
Neural Processing Unit (NPU)

Analog Devices MAX78000

STM32N6



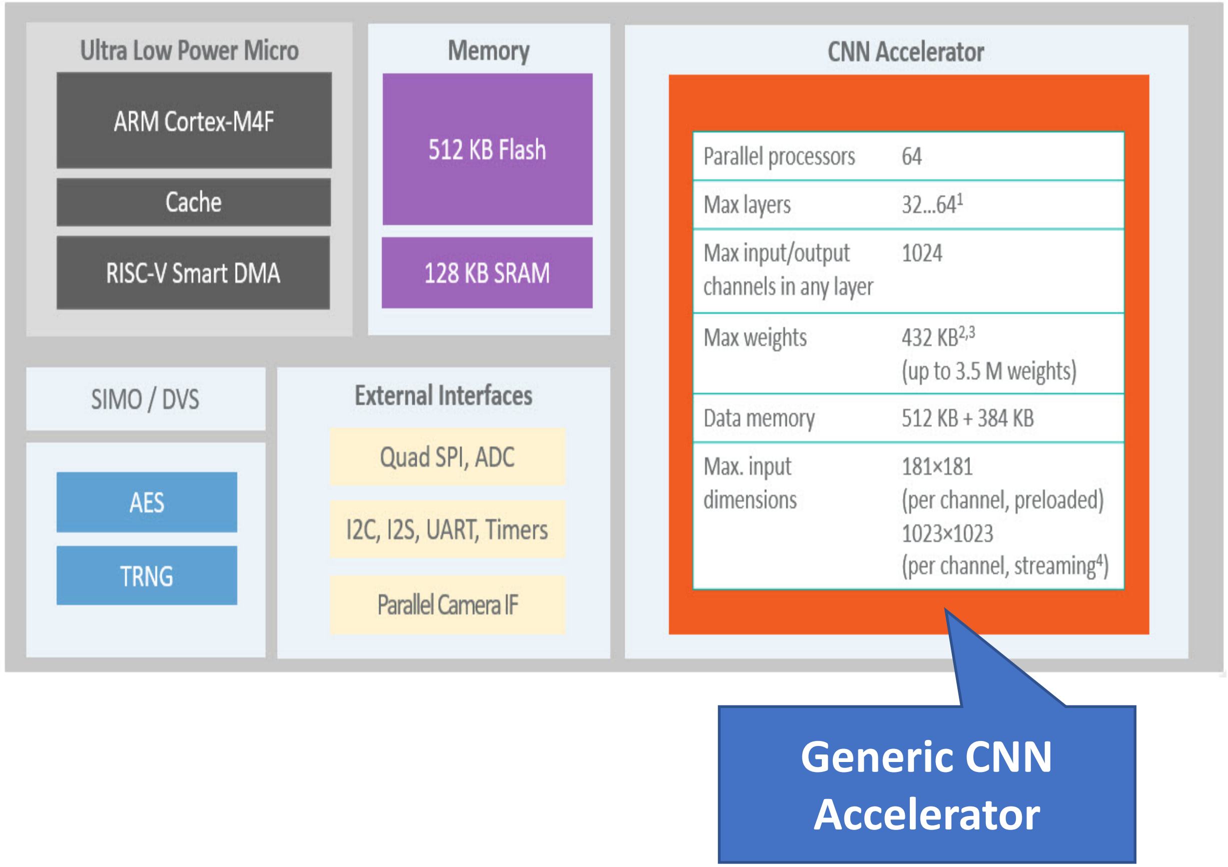
Intel Core Ultra

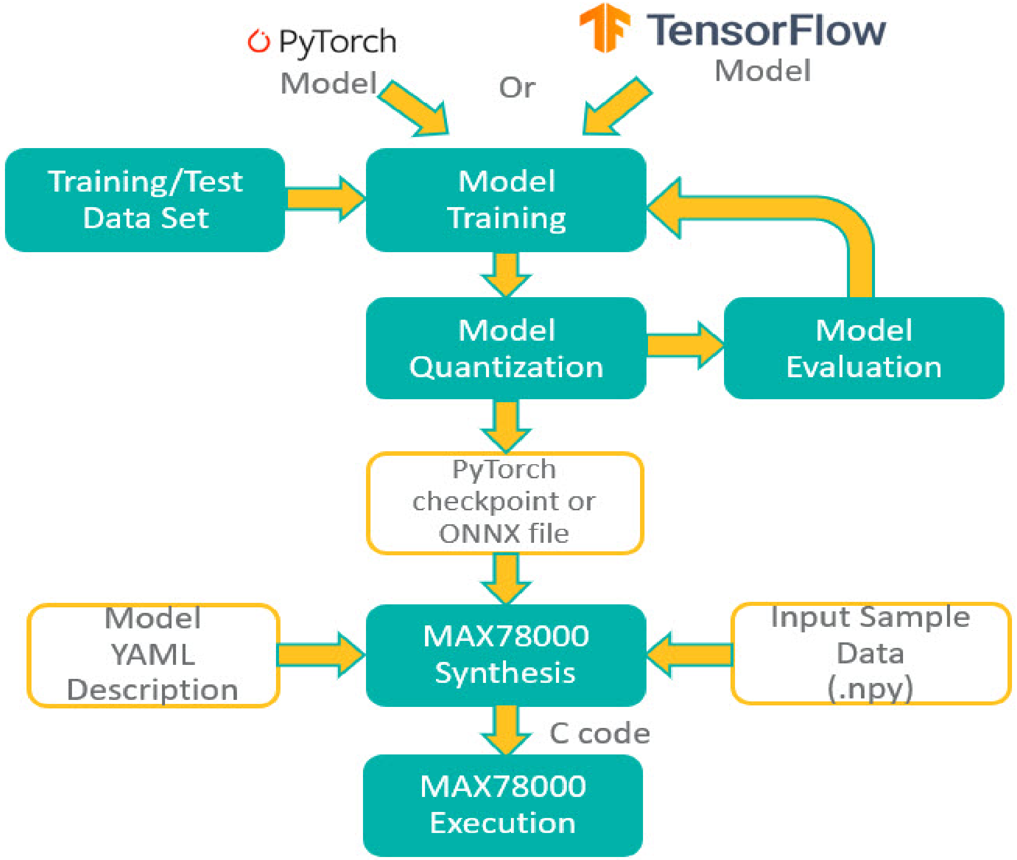


Top Edge AI Chip Players (2025)

Company	Unique Approach & Status	Primary Use Cases	Current Products
Apple	Neural Engine in A/M series SoCs	iPhones, iPads, Macs, Apple Watch	A18 Pro, M4 Neural Engine
NVIDIA	Jetson edge systems; expanding into telecom and embedded AI	Robotics, drones, smart infrastructure	Jetson Orin NX/Nano, AGX Orin
Qualcomm	Hexagon AI Engine; edge infrastructure box	Smartphones, XR, automotive, smart cities	Snapdragon 8 Gen 4, Ride Flex
Tesla	In-house AI chip design; Samsung as foundry	Autonomous driving, edge AI in EVs	AI6 chip
Sima.AI	MLSoC with software-centric edge AI stack	Robotics, industrial automation	Cheetah SoC
Axelera AI	In-memory compute + RISC-V control; EU-backed	Vision systems, retail AI, robotics	Metis AI platform
Mythic	Analog compute-in-memory	Surveillance, drones, industrial vision	M1076 AMP
MediaTek	Affordable, AI-optimized SoCs for edge devices	Smartphones, infotainment, IoT	Dimensity 9300+, Genio 1200
Intel	Tiber SoCs; modular edge platform	Smart cities, retail, industrial edge	Tiber Edge platform
Kneron	Custom NPUs optimized for face/gesture AI	Smart home, surveillance	KL720, KL730
Hailo	Ultra-efficient dataflow-based AI accelerator	Edge cameras, robotics, AIoT	Hailo-8, Hailo-15
Huawei / HiSilicon	CloudMatrix platform; secure China-centric edge deployments	Telecom, industrial, surveillance	Atlas 300/500, Kunlun II
Ambarella	Vision-focused SoCs for ADAS and edge vision processing	Autonomous vehicles, robotics, surveillance	CV3, CV5
Analog Devices	CNN-capable ultra-low-power MCUs	Medical devices, industrial sensors	MAX78000/02
NXP	Embedded NPUs with TinyML and secure AI compute	Automotive, industrial control, smart sensors	i.MX 8M Plus, i.MX 9 series
Synaptics / Eta Compute	Voice/vision AI on ultra-low-power platforms	Smart home, wearables, always-on devices	Katana SoC; Astra developer kits
BrainChip	Neuromorphic Akida chip (event-based)	Low-power edge vision and event detection	Akida AKD1000

MAX78000 Example





https://www.analog.com/en/resources/design-notes/keywords-spotting-using-the-max78000.html

LSI Contest 2018-2025

- Design challenge : "Neural Network(Backpropagation)"
- Design Specification
- Who can join： the team of 1-3 University or college students.
- The final report deadline ： ~~Wednesday, 31st January, 2018~~ → Friday, 9th February, 2018
- Report
- In this home page, we have used Synphony Model Compiler as a development tool;howver, applicants can freely use any type of architecutre as well as any EDA tools.
- Any Q&A: support@LSI-contest.com

- Design challenge : "Deep Learning(Backpropagation)"
  【Changes】
- Design Specification
- Who can join： the team of 1-3 University or college students.
- The final report deadline ： Thursday, 31st January, 2019
- Report
- Any Q&A: support@LSI-contest.com

- Design challenge : "CNN(Convolution Neural Network)"
  【Changes】
- Design Specification
- Who can join： the team of 1-3 University or college students.
- The final report deadline ： Friday, 31st January, 2020
- Report
- Any Q&A: support@LSI-contest.com

- Design challenge : "Reinforcement Learning"
  【Changes】
- Design Specification
- Who can join： the team of 1-3 University or college students.
- The final report deadline ： Friday, 29th January, 2021
- Report
- Any Q&A: support@LSI-contest.com

- Design theme: Deep Q-Network : DQN
- Report Deadline: Friday, 28th January 2022.
- Presentation: Friday, 4th March 2022

- Design challenge : "Autoencoder"
- Design Specification
- Who can join： the team of 1-3 University or college students.
- The final report deadline ： Wednesday, 31st January 2024
- Report
- Any Q&A: support@LSI-contest.com

- Design challenge : "Variational Autoencoder"
- Design Specification
- Who can join： the team of 1-3 University or college students.
- The final report deadline ： Tuesday, 31st January 2025
- Submit or any Q&A: support@LSI-contest.com

General vs. Specific AI Accelerator

⚙️ Two Types of AI/ML Accelerators

Type	Description	Example
1. General / Programmable Accelerator	Runs AI models compiled from frameworks (TensorFlow, PyTorch, ONNX). Uses a **compiler or synthesis tool** to map operations onto general compute blocks (MAC arrays, SIMD units, etc).	NVIDIA GPU, Google TPU, Intel NNP, or custom instruction set extensions on RISC-V
2. Specific / Fixed-Function Accelerator	Hardware is **custom-tailored** to a specific model or layer type (e.g., CNN, transformer, LSTM). Usually has **hardwired datapaths** for specific operations.	Mobile NPUs, ASIC for YOLO or BERT, FPGA pipeline for a known CNN model

General vs. Specific AI Accelerator

🧠 Hardware Design Perspective: Which Is Easier?

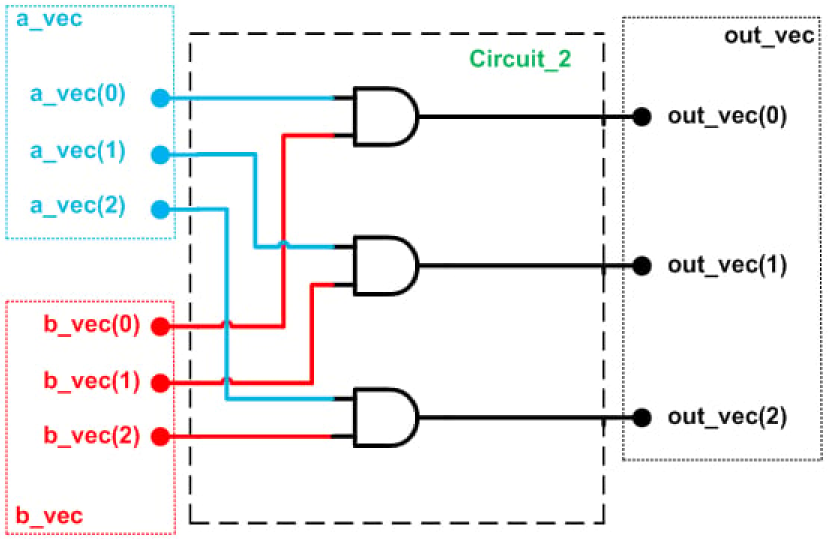
Let's compare in detail:

Aspect	General Accelerator	Specific Accelerator
Design complexity	❌ **More complex** — needs programmable datapath, instruction decoder, scheduling, memory management, etc.	✅ **Simpler** — only build fixed datapath for known model structure.
Verification	❌ Harder — must test many instruction combinations and model types.	✅ Easier — limited dataflow, fewer corner cases.
Flexibility	✅ High — can run many models via software tools.	❌ Low — must redesign/re-synthesize for new model.
Performance efficiency	⚖️ Moderate — some overhead for programmability.	✅ Very high — dataflow and memory tailored exactly to model.
Toolchain requirement	❌ Needs compiler backend, ISA mapping, runtime libraries.	✅ Often runs standalone with simple controller.
Time to design (HW)	⏱️ Long (months–years)	⏱️ Short (weeks–months)
Use case	Research chips, startups aiming for flexibility.	Edge devices, IoT, vision systems with fixed workloads.

FPGA

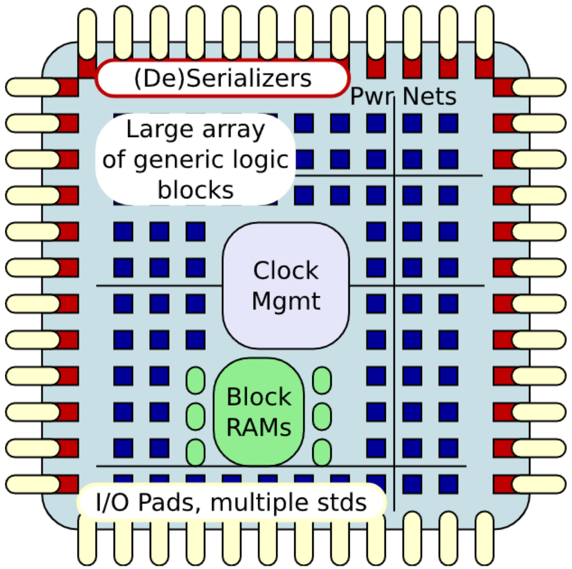
Design Flow

Verilog Design



Synthesis,
Implementation

FPGA

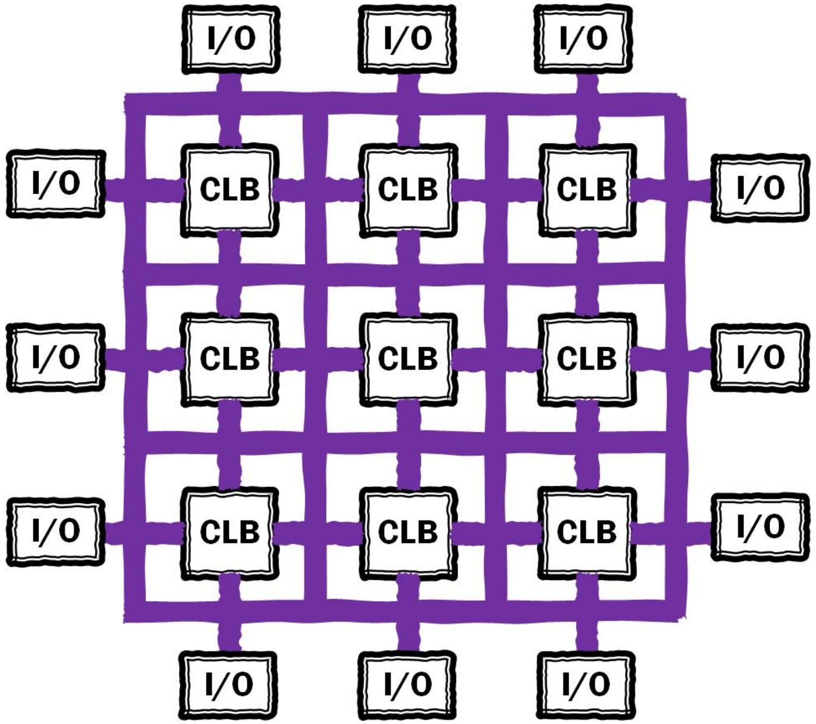


LUT, Flip-flop, RAM, clock manager

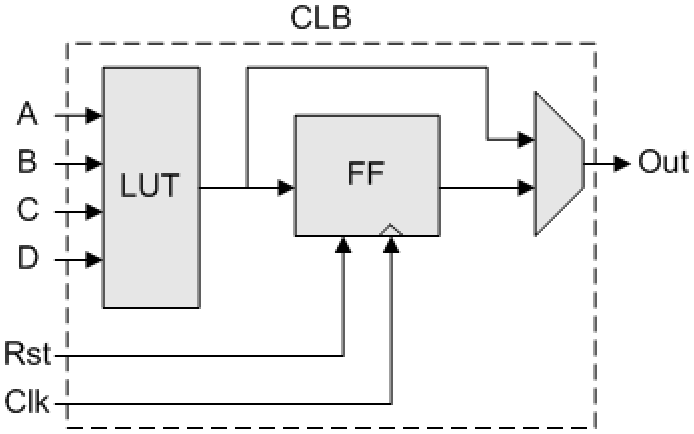
ASIC

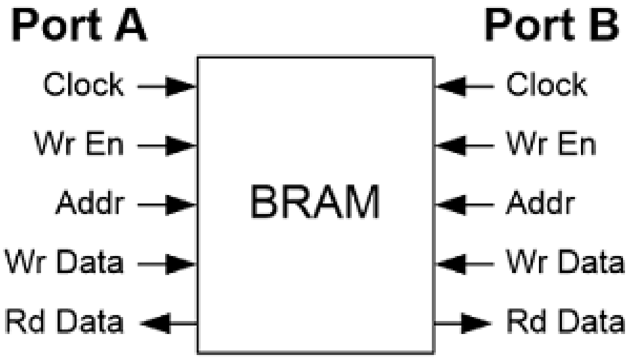
SRAM block, gate level, standartd cell transistor

Xilinx FPGA (Simplified Block)

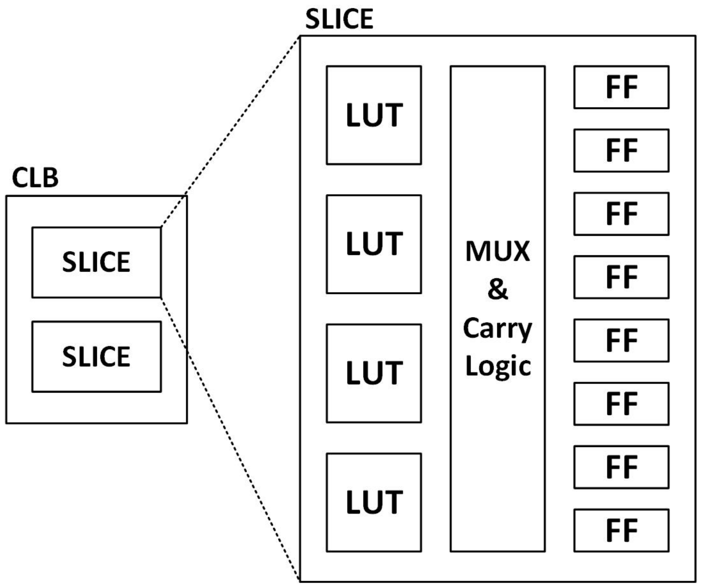


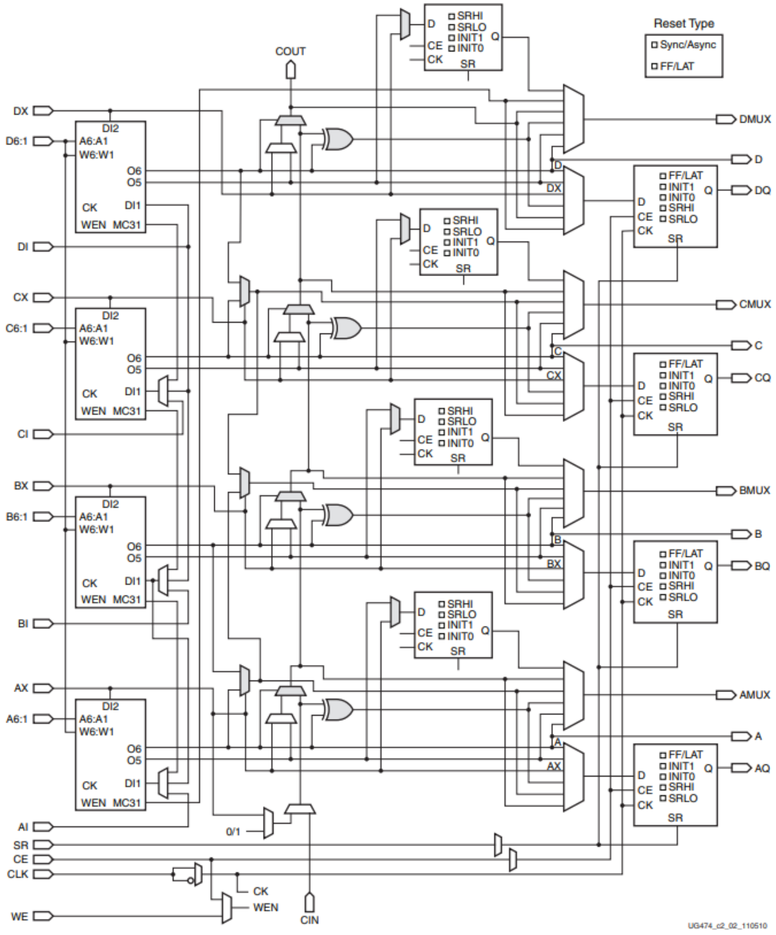
FPGA Architecture

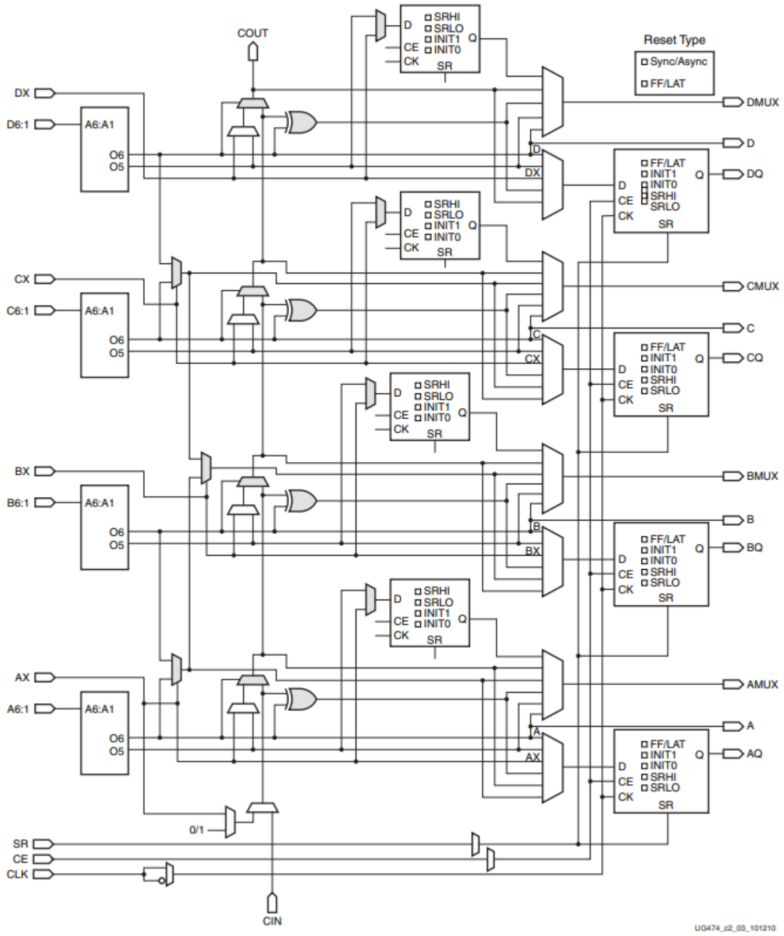




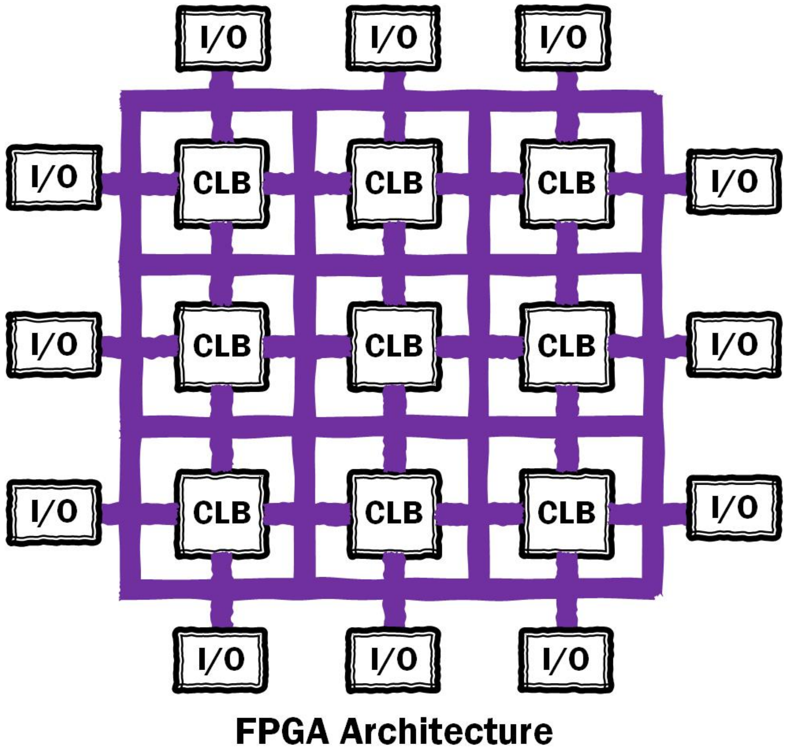
Xilinx FPGA (Detailed)



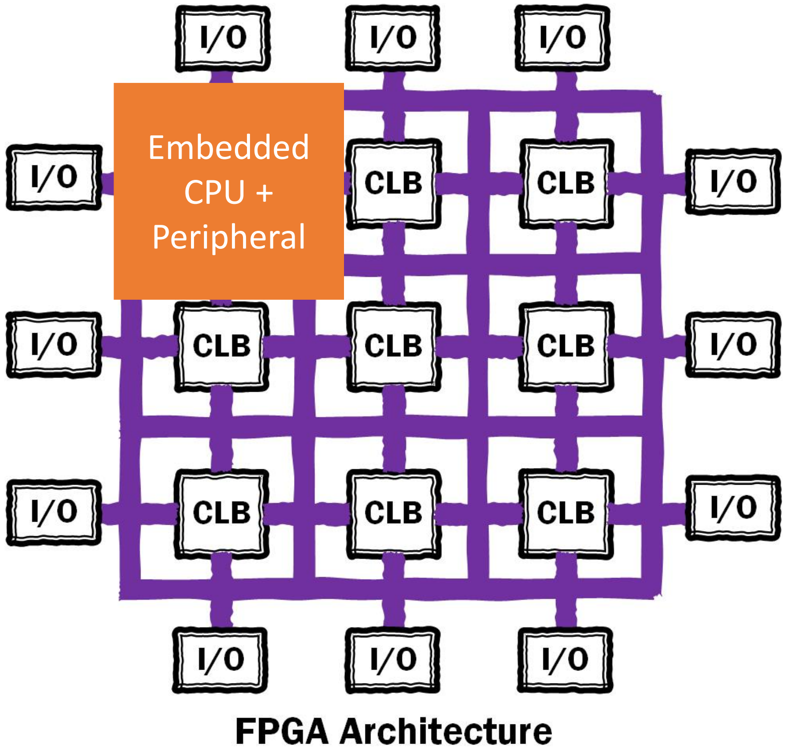




Traditional FPGA vs. Modern SoC FPGA

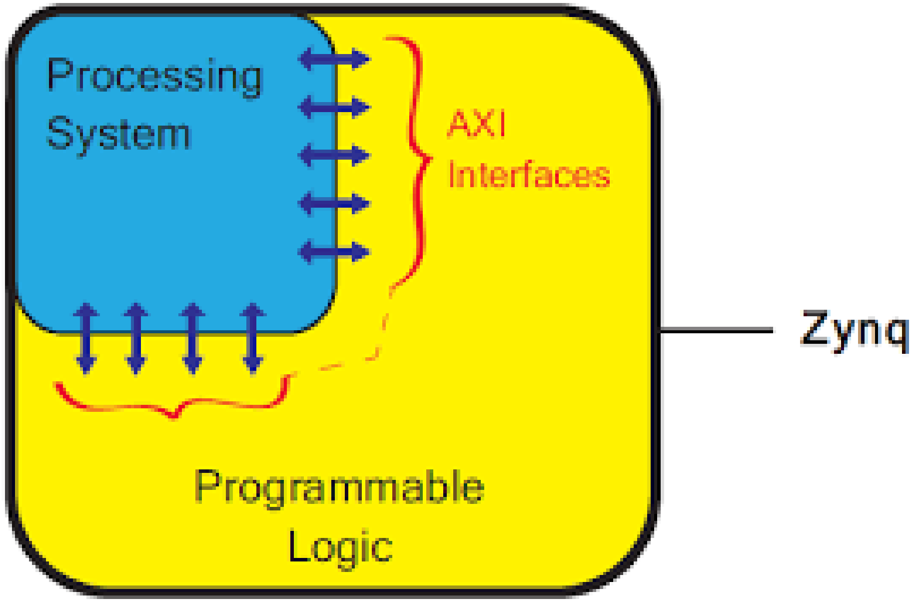


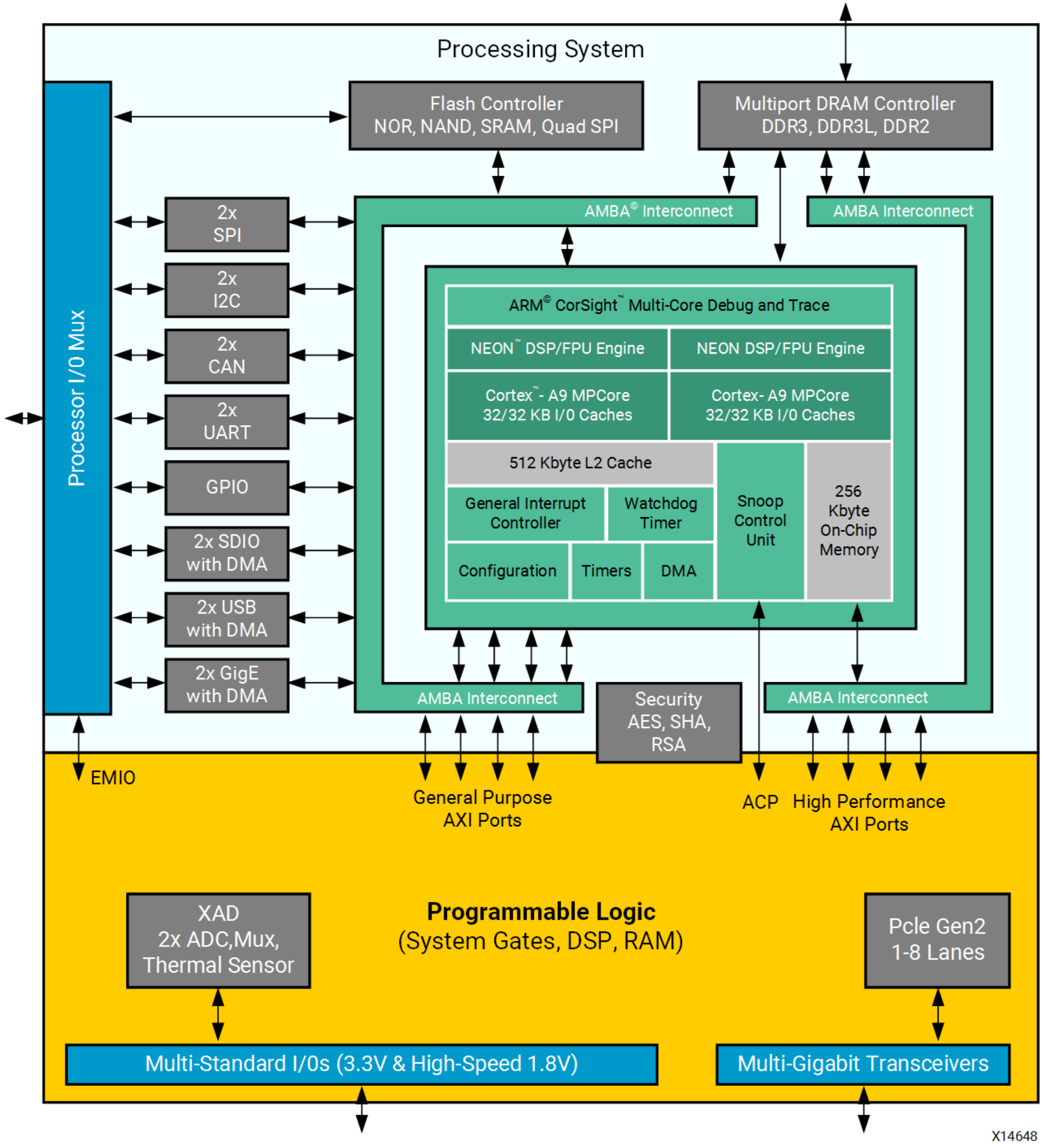
Traditional FPGA



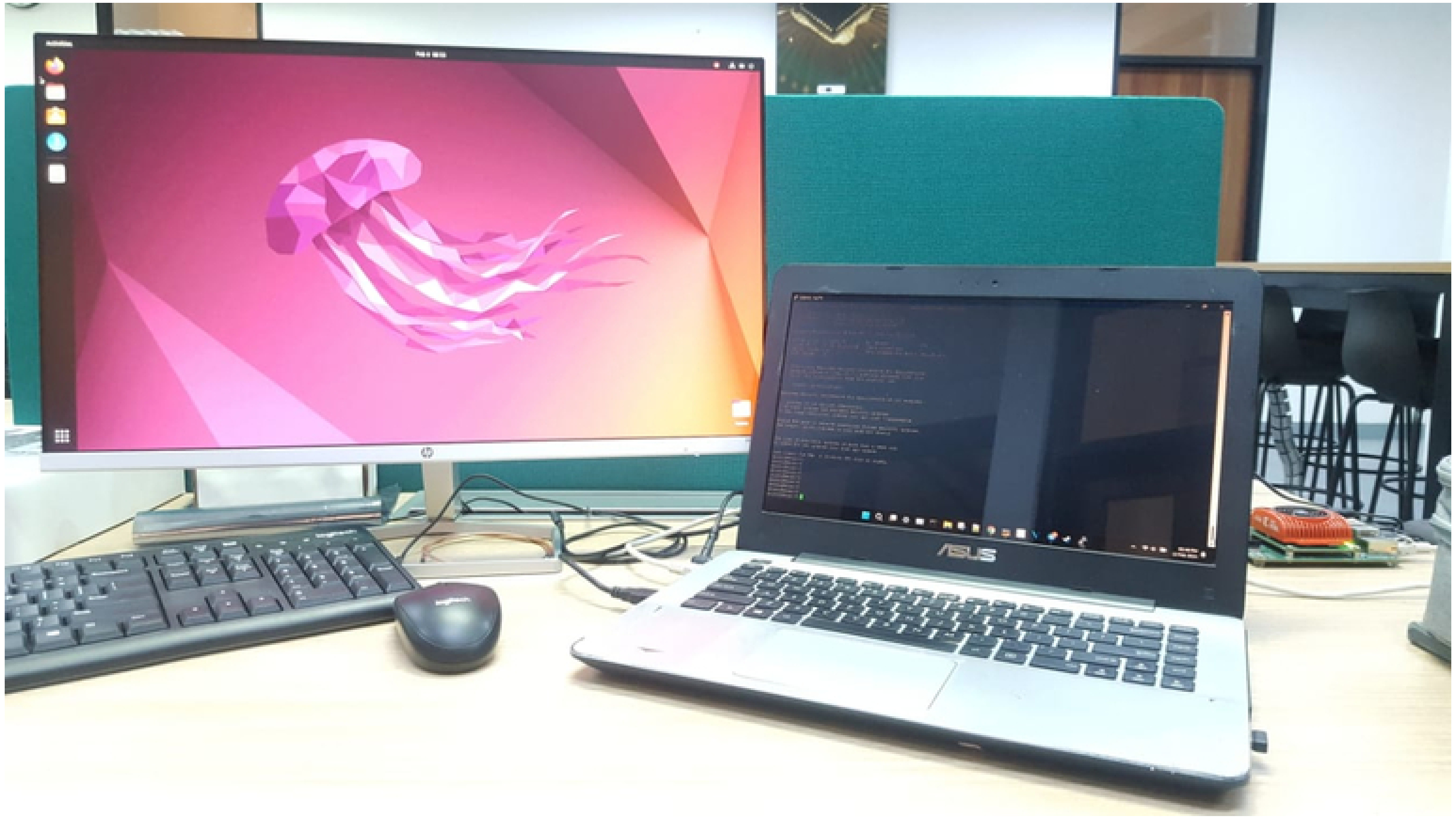
Modern SoC FPGA

Xilinx Zynq SoC FPGA

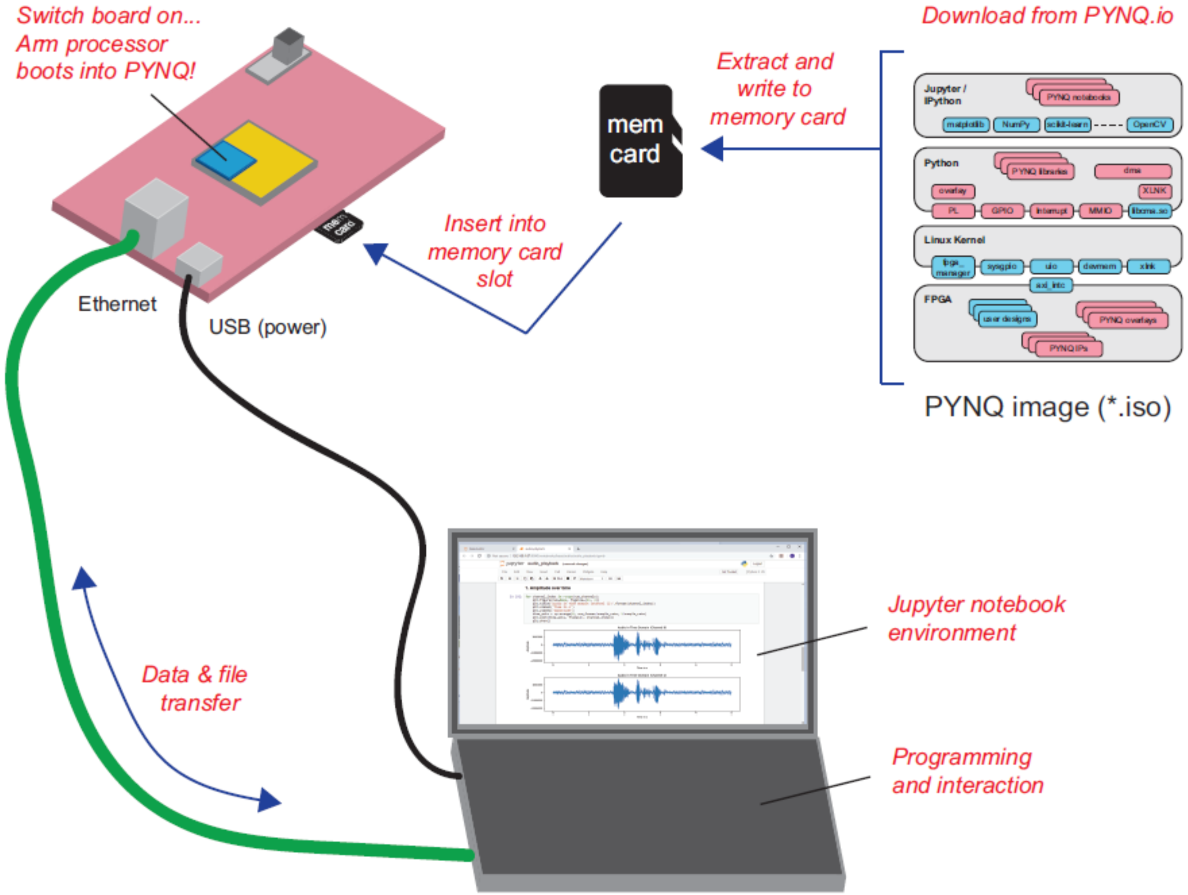


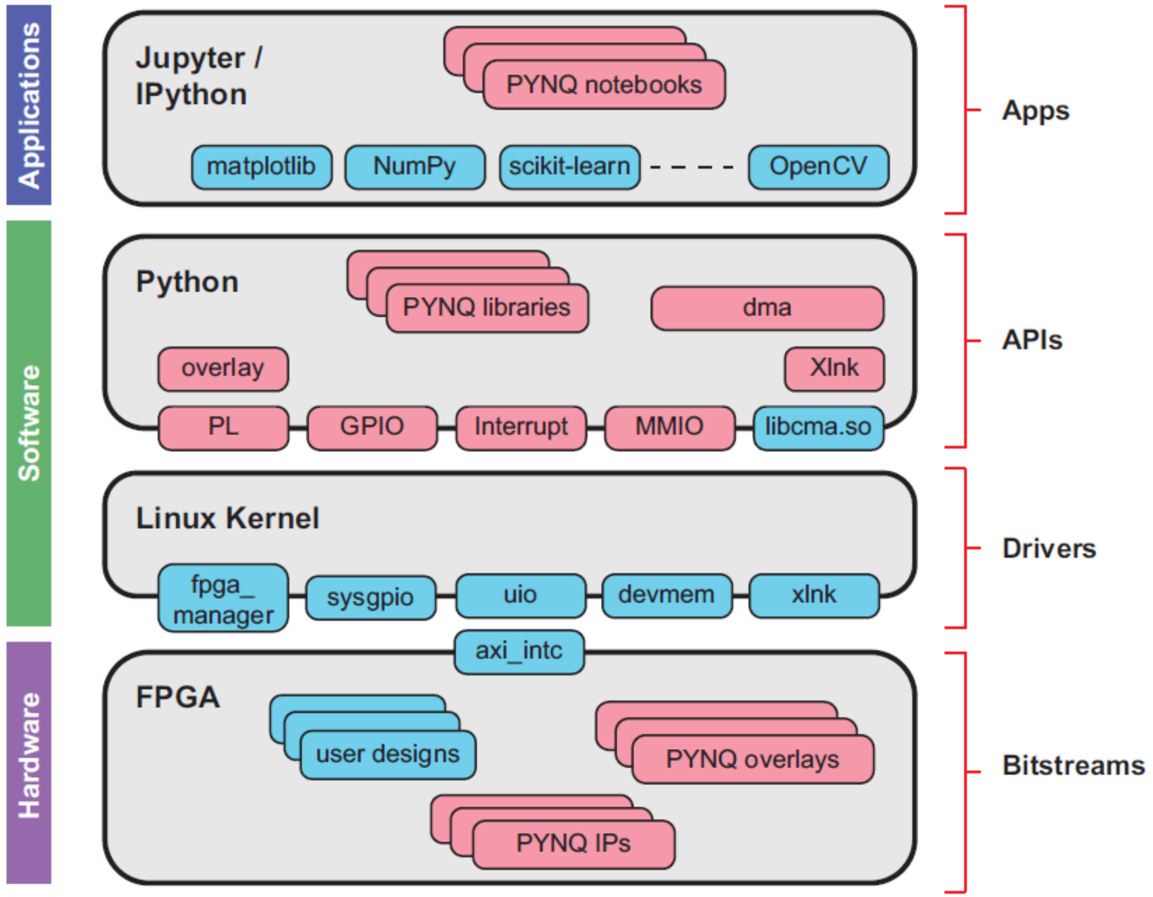


Xilinx Zynq SoC FPGA Runs Linux



Linux + PYNQ (Python for Zynq) Framework





Linux + PYNQ (Python for Zynq) Framework

- PYNQ is used for software development that supports your accelerator. It doesn't replace Verilog — you still need Verilog to design the hardware.

🧠 **Think of it like this**

- **Verilog** → builds the "engine" (hardware accelerator)
- **PYNQ** → builds the "driver's dashboard" (software interface to run it easily)

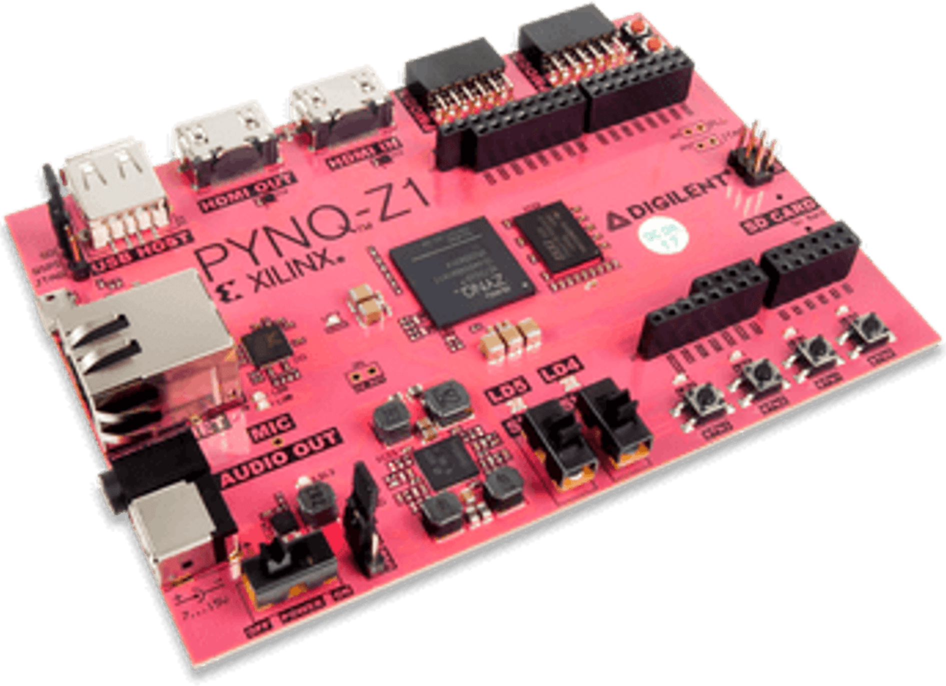
Why Use SoC FPGA + PYNQ?

Benefit	Description
Hardware acceleration	Run heavy layers (Conv, MatMul, etc.) on FPGA for speed-up
Rapid prototyping	Use PYNQ Jupyter Notebooks to test ideas without writing HDL testbenches
Easy integration	Combine Python AI frameworks (TensorFlow, PyTorch, NumPy) with FPGA kernels

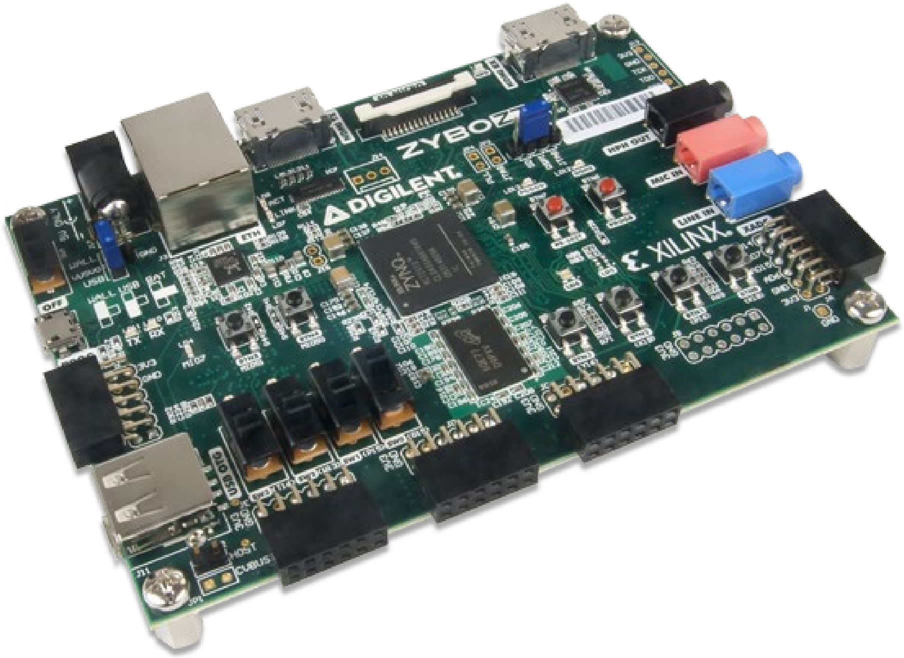
Board Setup

Tutorial: https://hackmd.io/@ween168/SyNPIkUb1x

Zynq FPGA Board

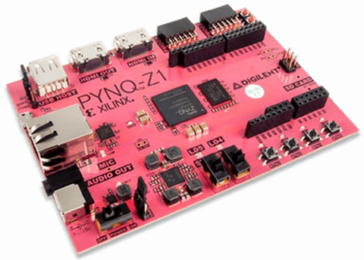


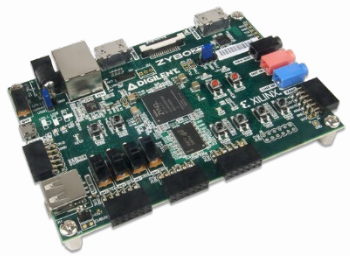
Pynq Z1



Zybo

Required Hardware





- A ZYNQ FPGA board that supports PYNQ
- A USB micro cable
- An Ethernet cable (also a USB-to-Ethernet adapter if your laptop doesn't have Ethernet)
- A MicroSD card with a capacity of 16 GB; **do not use** one that is 64 GB or larger (not supported)
- A MicroSD card reader (if your laptop doesn't have one) only needed once to flash the micro SD

Required Software

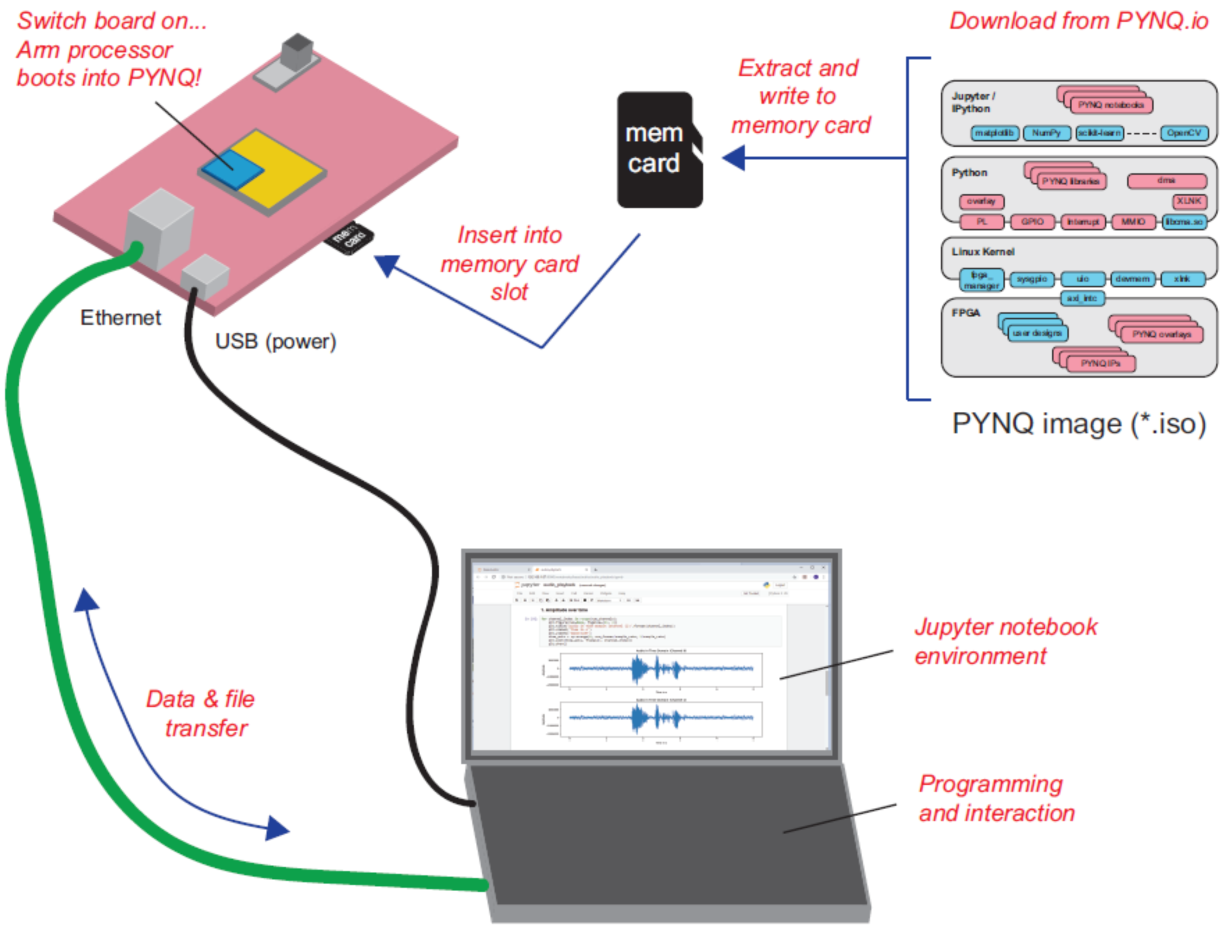
Download and install the following software tools:

- Vivado with board files installed
    - Retired Zybo, Zybo-Z7-10, Zybo-Z7-20: https://github.com/Digilent/vivado-boards/tree/master
    - PYNQ Z1, PYNQ Z2: https://pynq.readthedocs.io/en/v2.6.1/overlay_design_methodology/board_settings.html
- Win32DiskImager (https://win32diskimager.org/)
- PuTTY (https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html)
- WinSCP (https://winscp.net/eng/index.php?)

Download the PYNQ Linux OS image file depending on your FPGA board. Use the image v3.0.1.

- Retired Zybo, Zybo-Z7-10, Zybo-Z7-20: https://github.com/nick-petrovsky/PYNQ-ZYBO
- PYNQ Z1, PYNQ Z2, and others: https://www.pynq.io/boards.html

Step-by-Step Flow



Proper Shutdown of The Board

- **[IMPORTANT]** How to turn off the board? To prevent MicroSD card corruption, when turning off the board, perform a shutdown process with the command:

```
sudo shutdown -h now
```

- Wait until the FPGA board is completely shut down before turning off the board's power supply.

⚙️ Why You Need Proper Shutdown

The **PYNQ board** runs a **Linux-based operating system** from the SD card.
When Linux is running, it continuously **reads and writes files** — logs, caches, temporary files, configurations, etc.

If you **suddenly cut the power** (flip the switch) while it's writing something:

- The SD card write may be **interrupted mid-process**
- File allocation tables or metadata can become **inconsistent or corrupted**
- The OS might **fail to boot** next time or lose some files

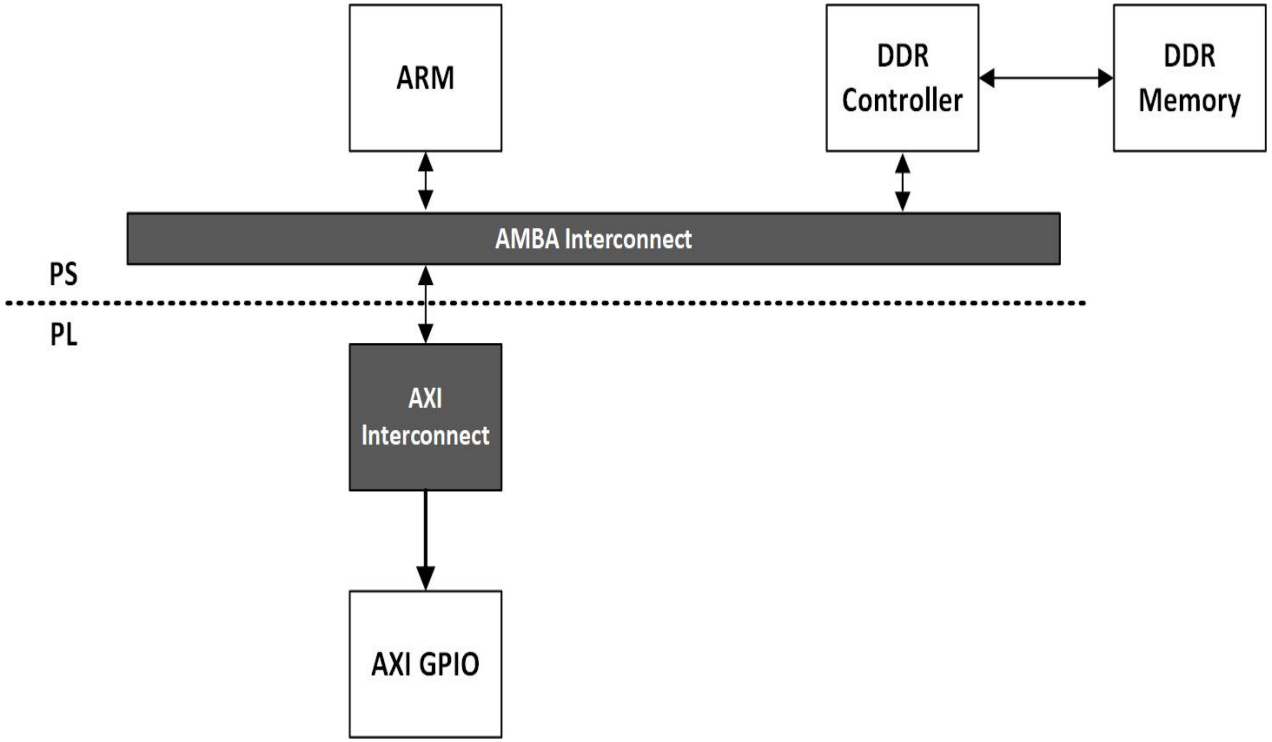
🧠 Think of It Like This

Turning off power abruptly is like **unplugging your PC** without shutting down Windows or Linux first.
→ The filesystem (ext4, FAT32, etc.) doesn't get a chance to "close" properly.

GPIO Example

Tutorial: https://hackmd.io/@ween168/SyNPIkUb1x

GPIO Examples



In [1]:
```python
from pynq import Overlay
from pynq import MMIO

# Program bitstream to FPGA
overlay = Overlay('/home/xilinx/design_1.bit')

# Access to memory map of the AXI GPIO
ADDR_BASE = 0x41200000
ADDR_RANGE = 0x2000
gpio_obj = MMIO(ADDR_BASE, ADDR_RANGE)
```

In [2]:
```python
# Write data to AXI GPIO
gpio_obj.write(0x0, 168)
```

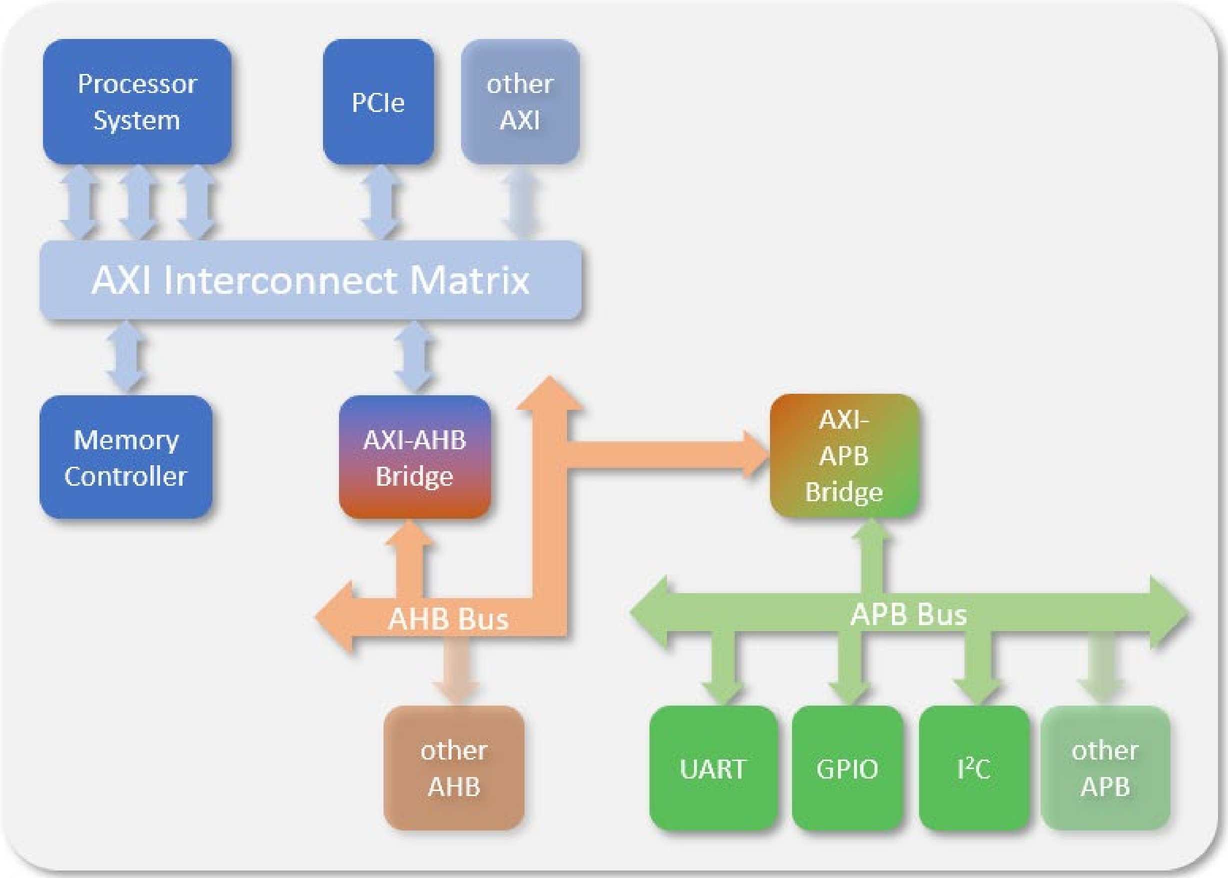
In [3]:
```python
# Read data from AXI GPIO
gpio_obj.read(0x0)
```

Out[3]: 168

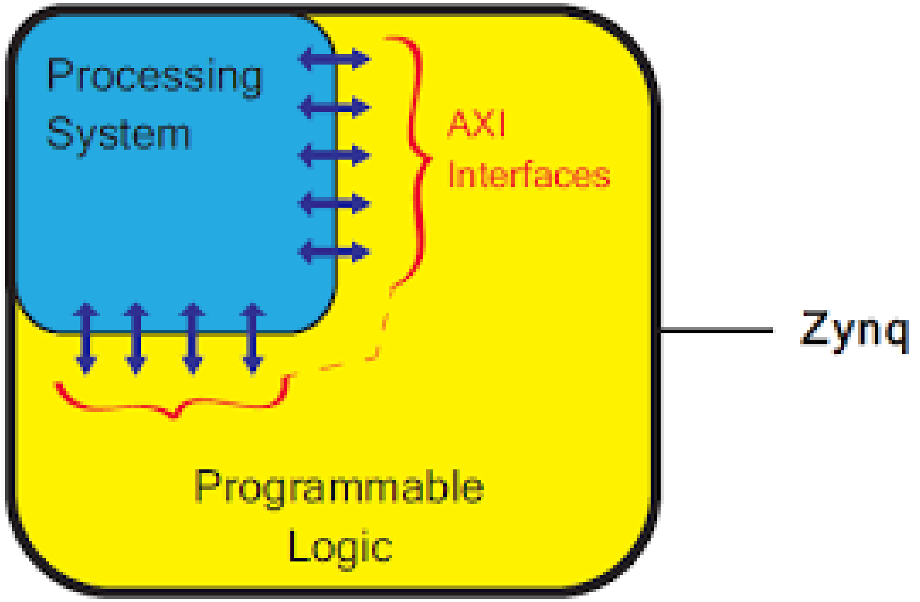
In [ ]:

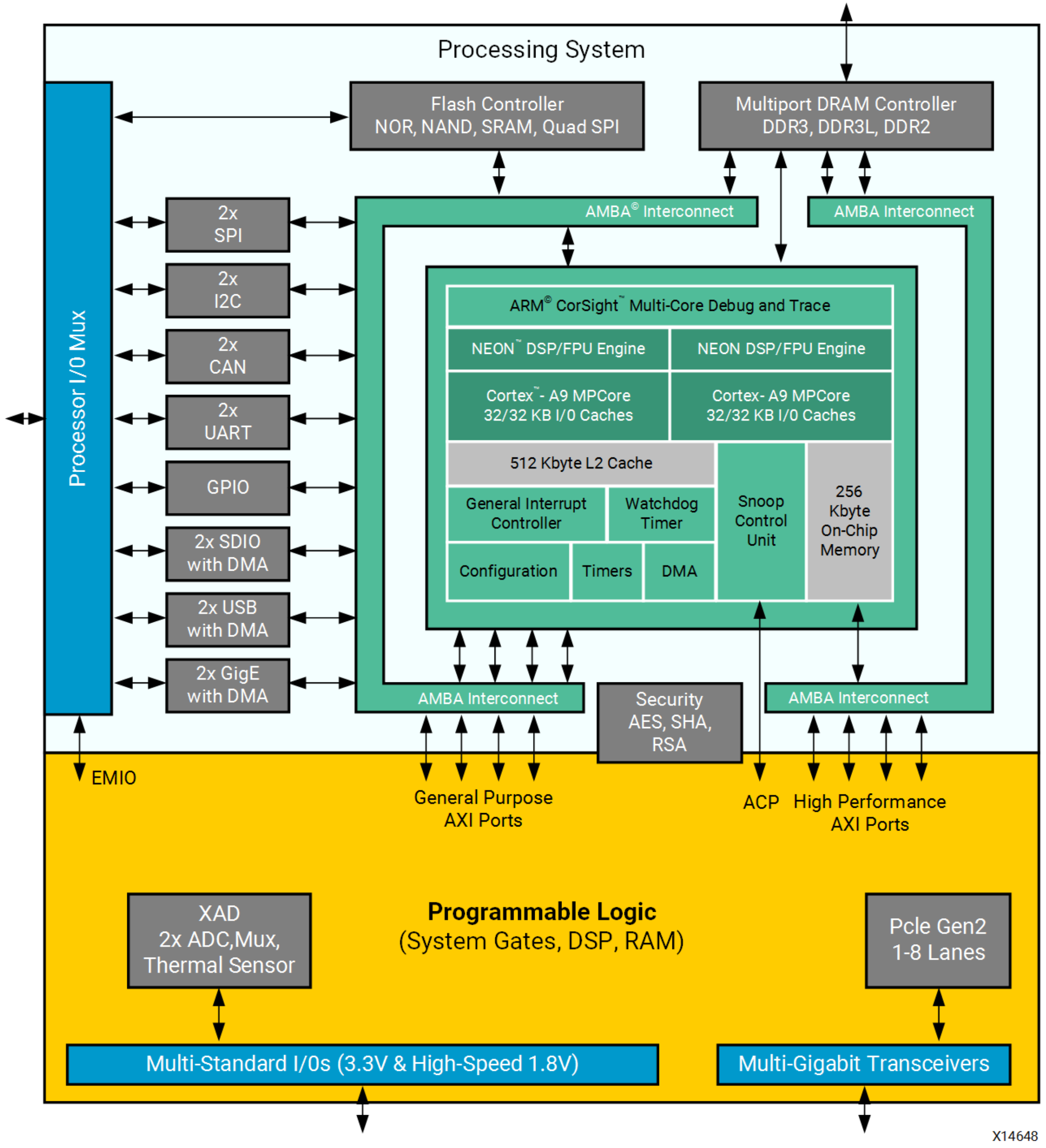
System Bus Interconnect (General)

- AXI
- AHB
- APB

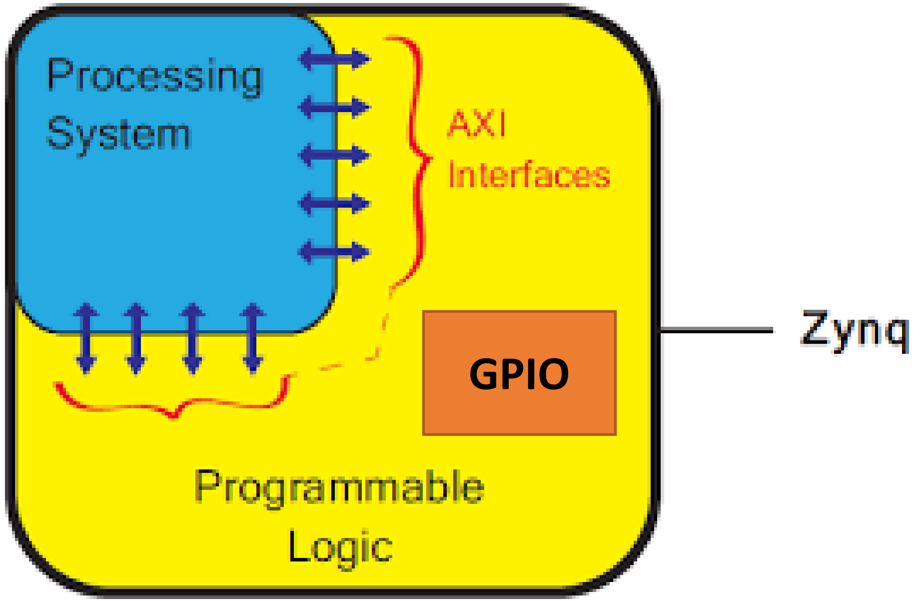


Xilinx Zynq SoC FPGA

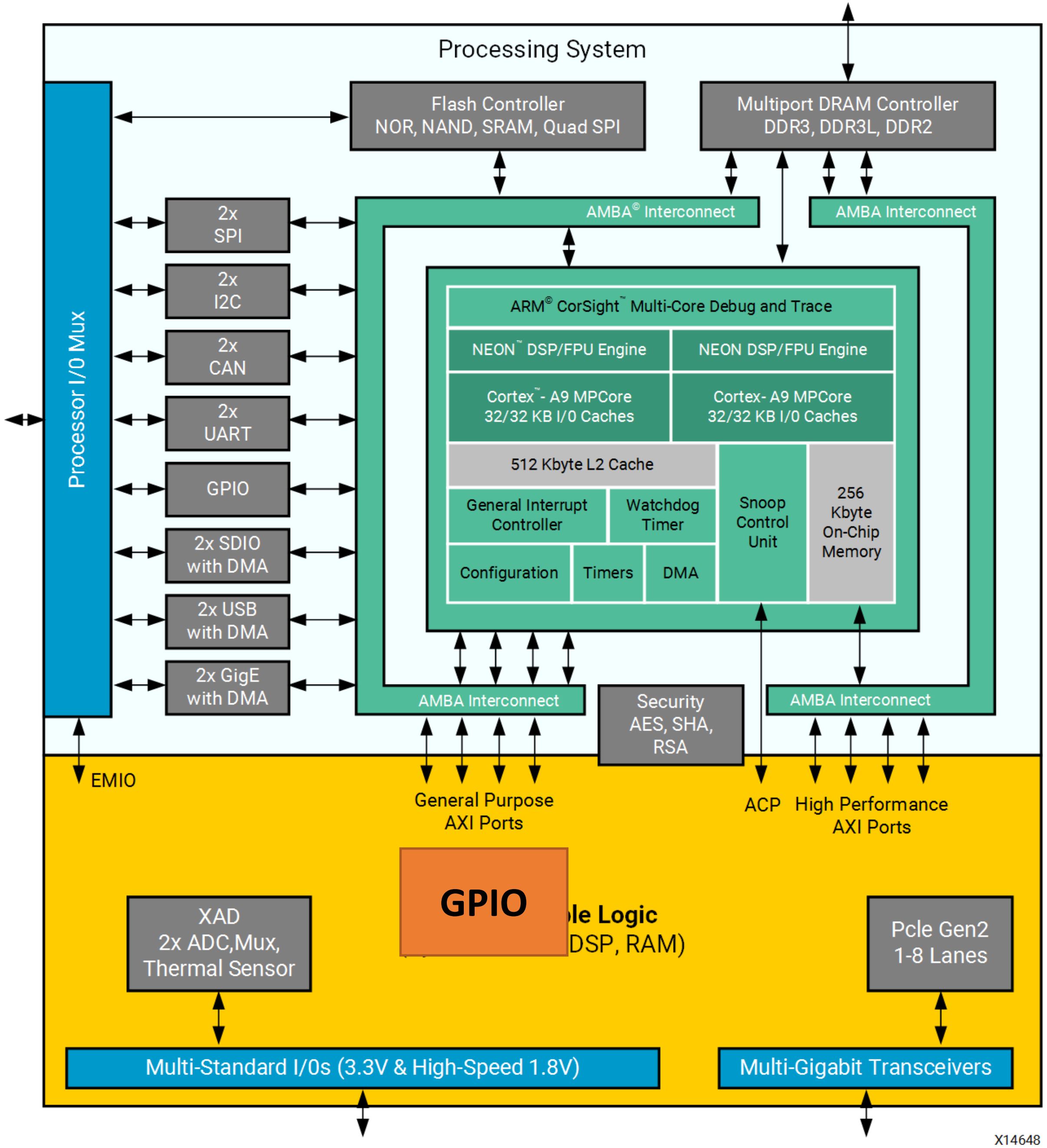




GPIO Design

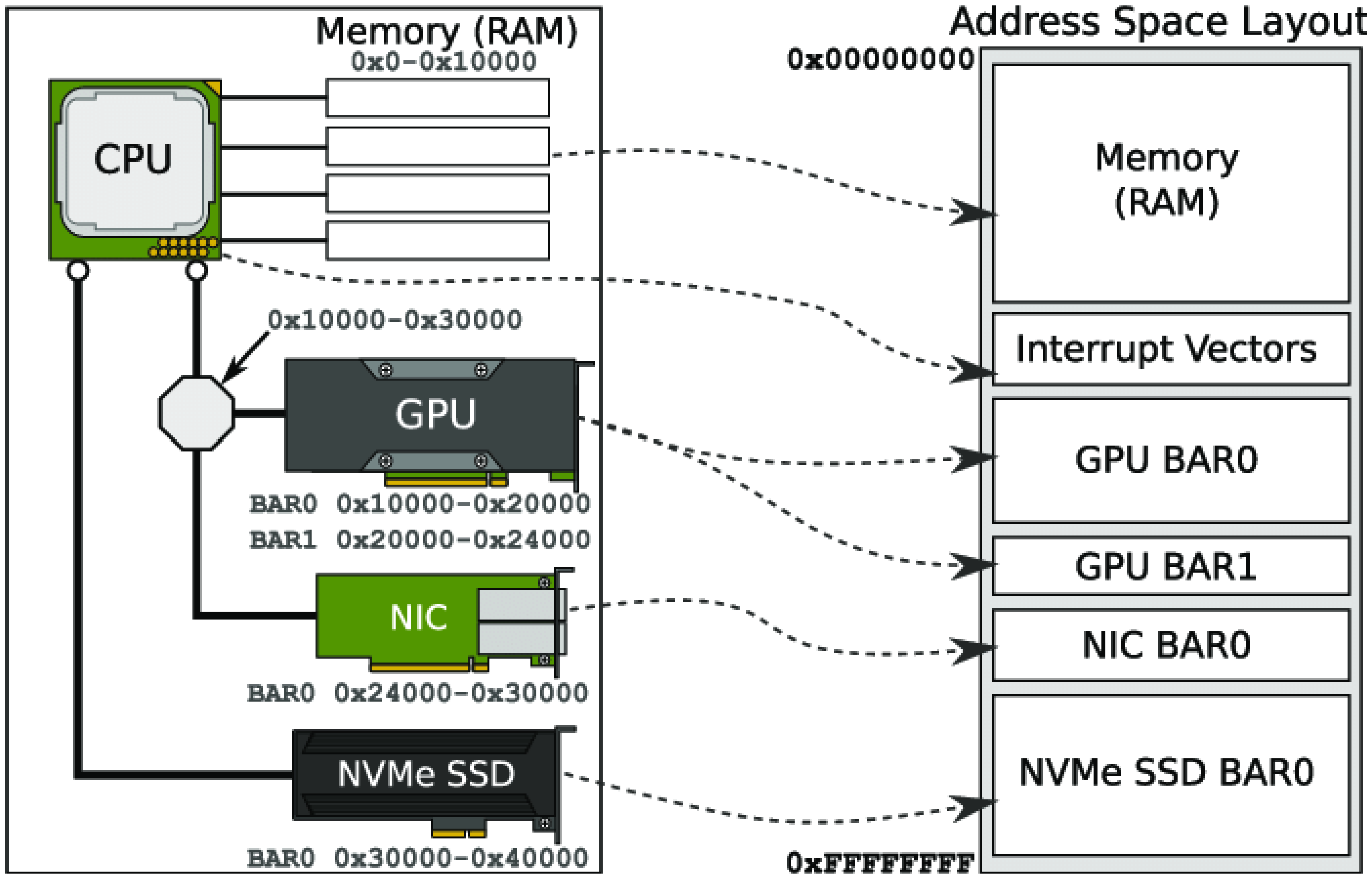


Processing System
AXI Interfaces
GPIO
Programmable Logic
Zynq



Processing System
Processor I/O Mux
Flash Controller
NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller
DDR3, DDR3L, DDR2
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA Interconnect
ARM® CorSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore
32/32 KB I/0 Caches
Cortex- A9 MPCore
32/32 KB I/0 Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Snoop Control Unit
256 Kbyte On-Chip Memory
Configuration
Timers
DMA
AMBA Interconnect
Security
AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XAD
2x ADC,Mux, Thermal Sensor
GPIO
le Logic
DSP, RAM)
Pcle Gen2
1-8 Lanes
Multi-Standard I/0s (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers
X14648

Memory Mapped IO



GPIO

GPIO Adress

In [1]:
```
from pynq import Overlay
from pynq import MMIO

# Program bitstream to FPGA
overlay = Overlay('/home/xilinx/design_1.bit')

# Access to memory map of the AXI GPIO
ADDR_BASE = 0x41200000
ADDR_RANGE = 0x2000
gpio_obj = MMIO(ADDR_BASE, ADDR_RANGE)
```

In [2]:
```
# Write data to AXI GPIO
gpio_obj.write(0x0, 168)
```

In [3]:
```
# Read data from AXI GPIO
gpio_obj.read(0x0)
```

Out[3]: 168

In [ ]:

Conclusion

- FPGA Design Flow.
- Install a Linux OS on a MicroSD card for the ZYNQ FPGA.
- Create a Hello World project for ZYNQ FPGA.