FPGAtors

Introduction to Vitis

Lucas Aguirre • 10/2/2026 • LAR234



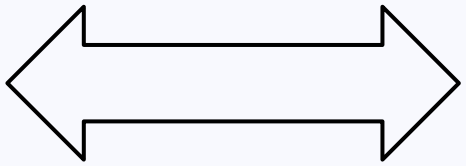
Today’s roadmap

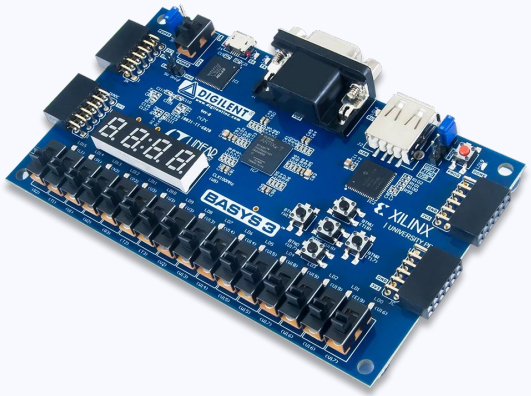
01 History	Brief background on Vitis
02 Why	Build justification for Vitis
03 What	Vitis and it’s functionality
04 How	A simple example

Before an integrated SoC

- A processor executes software and handles control-oriented tasks
- An FPGA implements configurable digital hardware and parallel data paths.
- Systems can connect a processor and FPGA as separate devices
- The two still need an interface, address space, and shared understanding of the hardware.
- Building the FPGA logic does not automatically teach processor software how to use it.



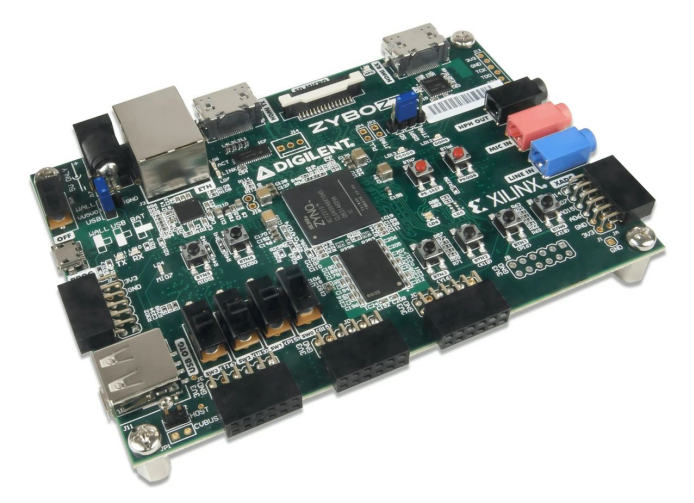






System-on-Chip

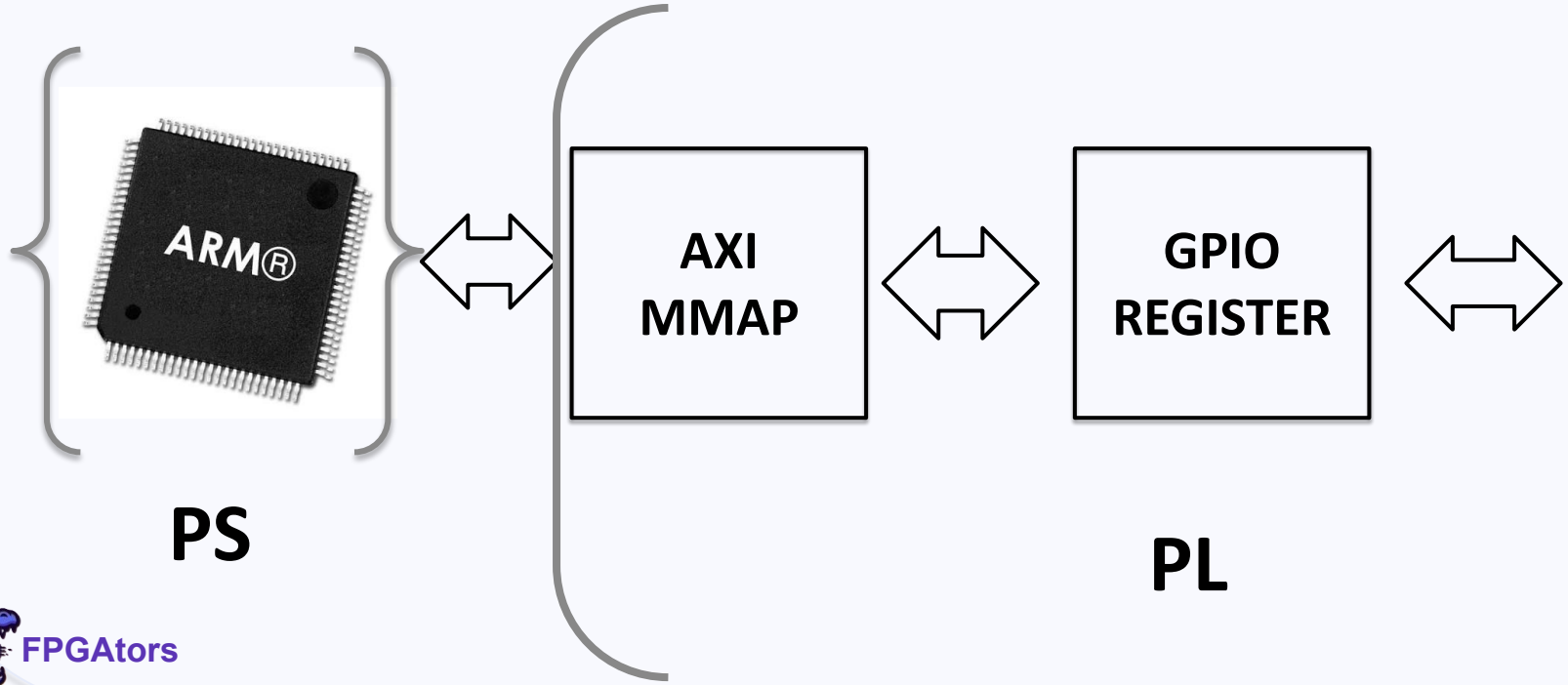
- A **system-on-chip (SoC)** integrates major system functions on one chip
- Zynq-7000 combines an ARM-based processing system (PS) and programmable logic (PL)
- The PS contains processors and fixed-functon peripherals
- The PL can implement custom peripherals and accelerators
- Integration shortens the physical distance; it does **not** remove the need for hardware/software coordination

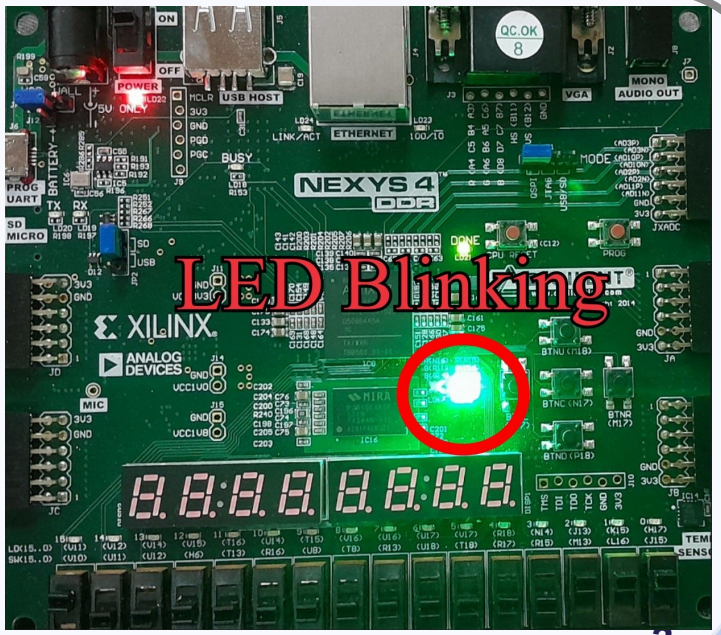




How does PS talk to PL?

- PS and PL exchange control and data through defined interfaces.
- **AXI** is a major interface family used on Zynq
- A processor can access PL through **AXI Memory-Mapped** registers
- The address map determines *where* software reads or writes
- Larger data transfers may use different interfaces like AXI Stream or DMA

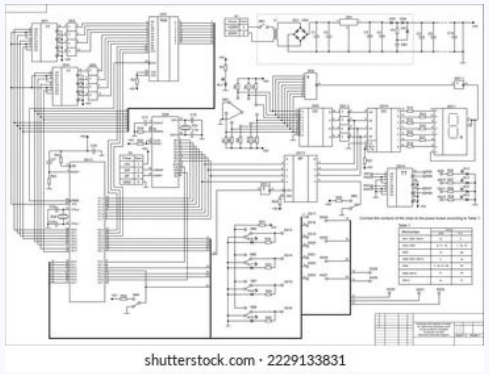




The missing agreement

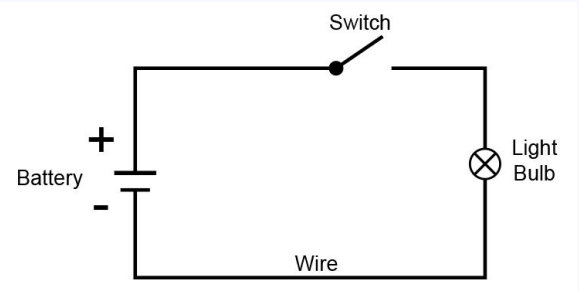
- Vivado defined *which* hardware blocks exist
- Vivado assigns addresses and connects clocks, resets, and interfaces
- Software needs matching peripheral information
- Software also needs to targe the correct processor and execution environment
- If hardware changes but software's view doesn't, the system can build and still fail!

Actual Hardware



vs.

Software Assumption





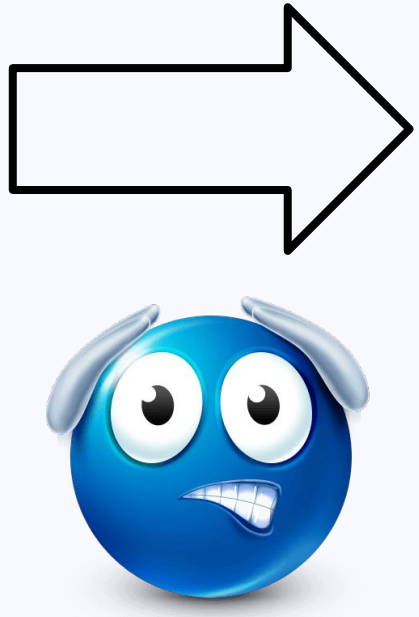
Is Vitis Necessary?

- No, technically we can manually use processor toolchains, linker scripts, startup code, and register definitions ourselves :D
  - And then after obtain peripheral addresses and write or integrate drivers
    - And then configure and load the PL separately
      - And then load/debug processor code with appropriate target tools
        - And then keep all those assumptions synchronized when hardware changes :D
- Is this possible? Yes! And it's a testament to an electrical/computer engineer's technical prowess
- The real question is, can you as the engineer maintain all of this?



What would happen if we didn't set it up perfectly?

- Wrong processor or software environment?
- Wrong peripheral base address?
- Missing or stale hardware handoff?
- PL not configured?
- Wrong output path?



- Unsuitable build or startup
- Writes go to the wrong places
- Software does not match the Vivado design
- A PL peripheral cannot respond as intended
- Watching the wrong console!

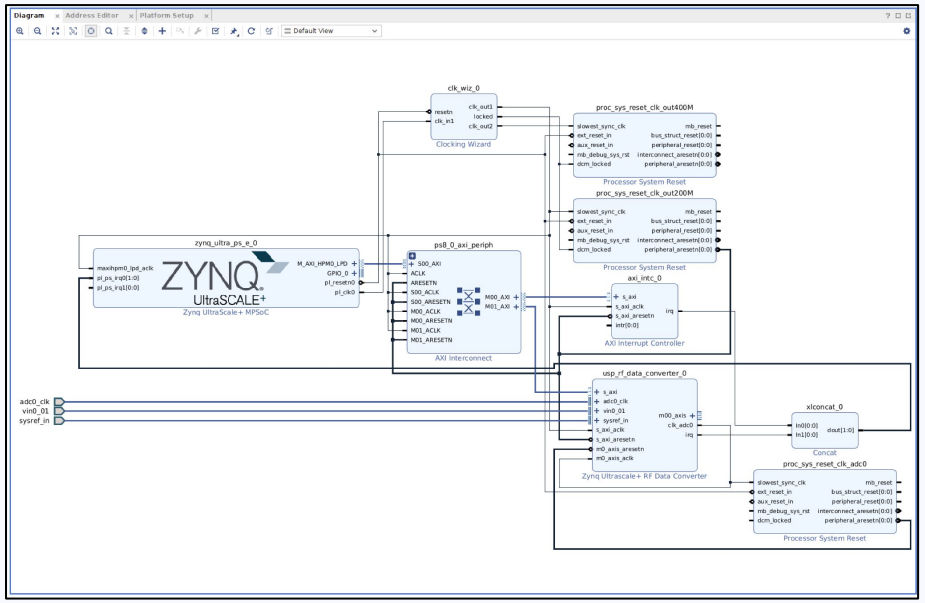
Vitis, the solution!

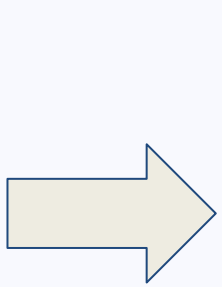
- AMD's **Vitis software platform** is a tool ecosystem for applications that target and take advantage of its adaptive hardware.
- It supports **embedded software development** for processors on SoCs
- It also supports **application acceleration** using hardware kernels.
- The **Unified IDE** is one way to use the tools; however scripted workflows exist (and are usually preferred)
- Vitis works *with* Vivado, it **DOES NOT REPLACE** hardware design or implementation.

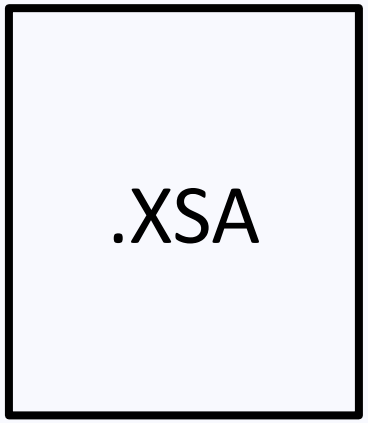


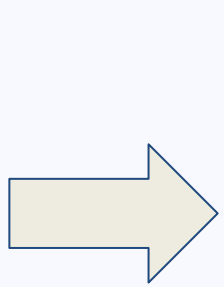
What does Vitis need?

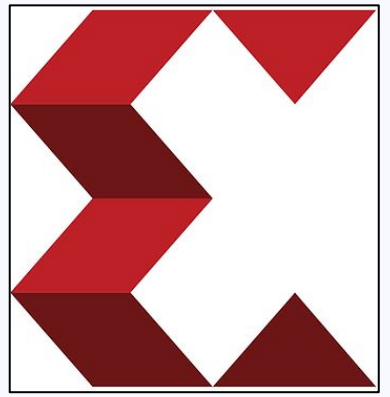
- Vivado exports a hardware handoff: an **XSA**
- The XSA conveys the hardware specification Vitis targets
- For our embedded design, the hardware is completed in Vivado
- The handoff may include the deice image needed to program the PL
- Changing relevant Vivado hardware means updating the software-side platform (XSA)
  - An example of this is [Dr. Wong's platform for EEE4514C]





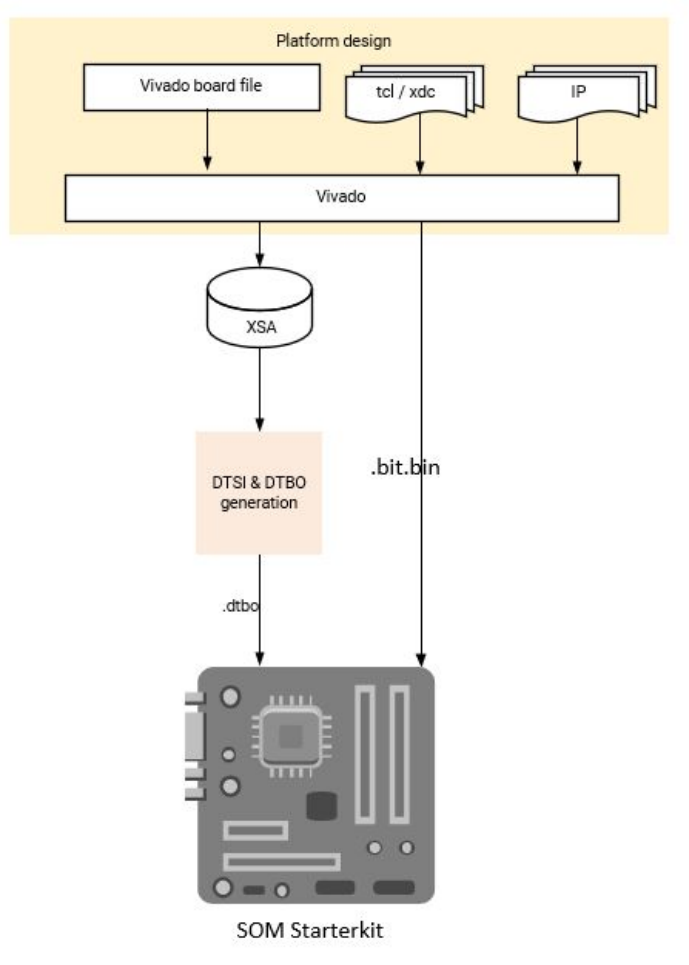


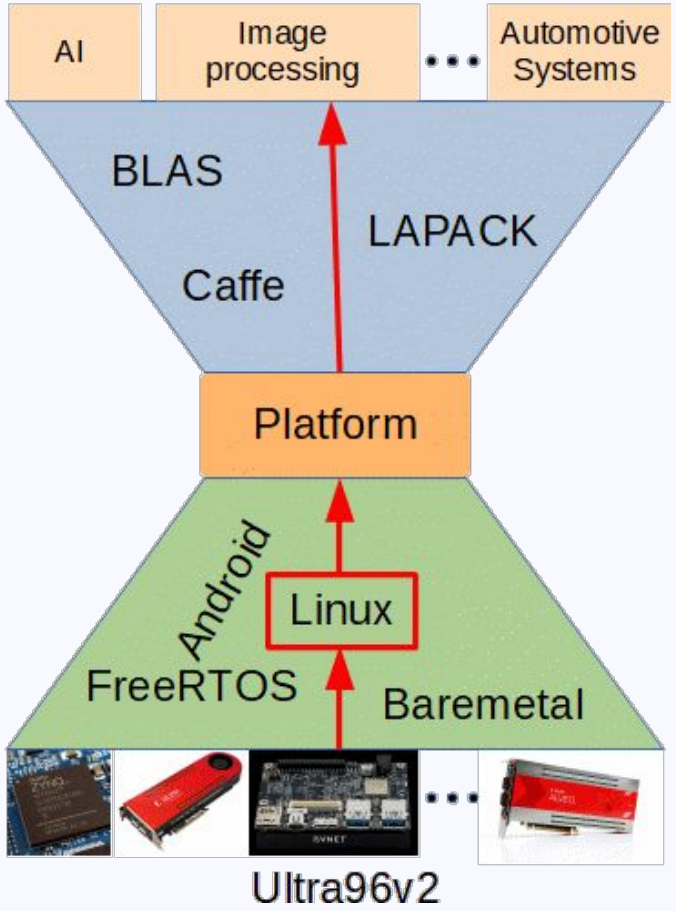




What is a Vitis Platform?

- A platform combines hardware specification and software environment
- It identifies available processors and relevant hardware
- A **domain** specifies the processor and software environment for an application
- Standalone domains include Board Support Packages (BSP) and drivers
- Multiple applications can target a reusable platform
- A platform is **not** the application itself





So What Does Vitis *Actually* Do?

- Creates and builds platforms from exported hardware
- Creates applications for a selected processor/domain
- Builds software using the matching drivers and libraries
- Loads, runs, and debugs applications on a target
- Helps package boot images for the chosen deployment math
- In other flows: develops and integrates accelerator components



Essential Vitis Artifacts

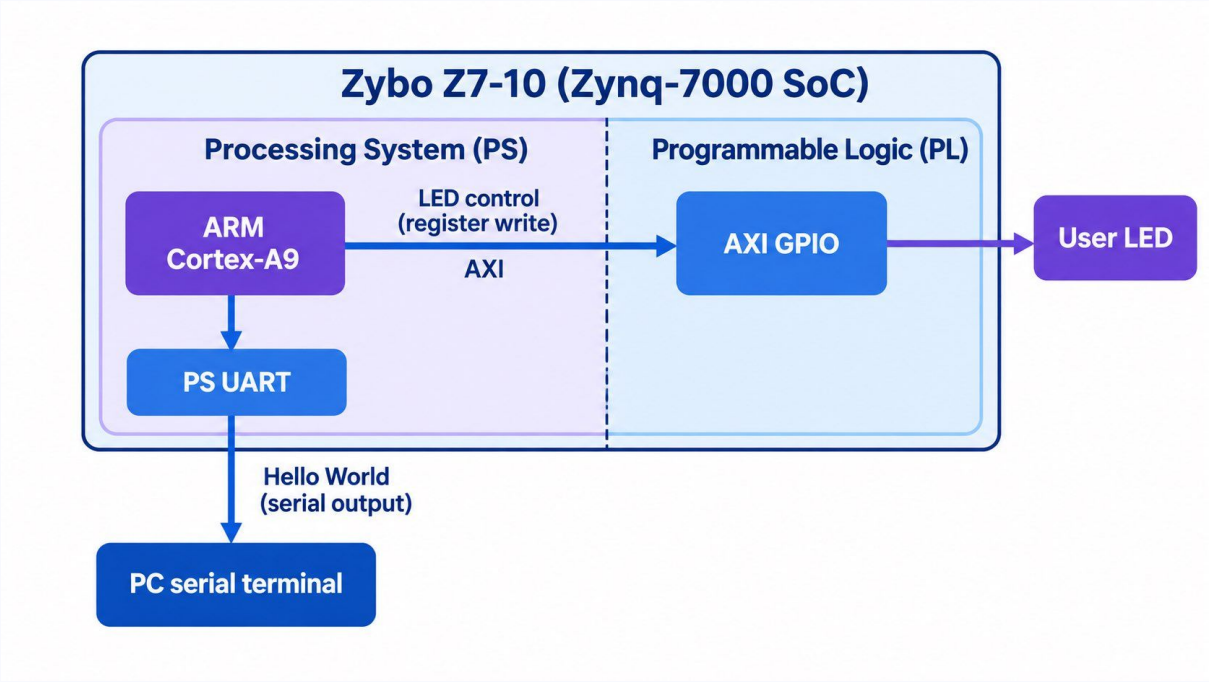
Artifact	What it represents	When we need to think about it
.bit/ device image	PL configuration	When the application uses PL hardware
.xsa	VIvado hardware handoff	Creating or refreshing the platform
Vitis platform	Hardware + software target definition	Building one or more applications
.elf	Compiled processor application	Loading/debugging software
Boot image	Packaged boot component	Starting without an IDE-driven launch



Example Setup Flow

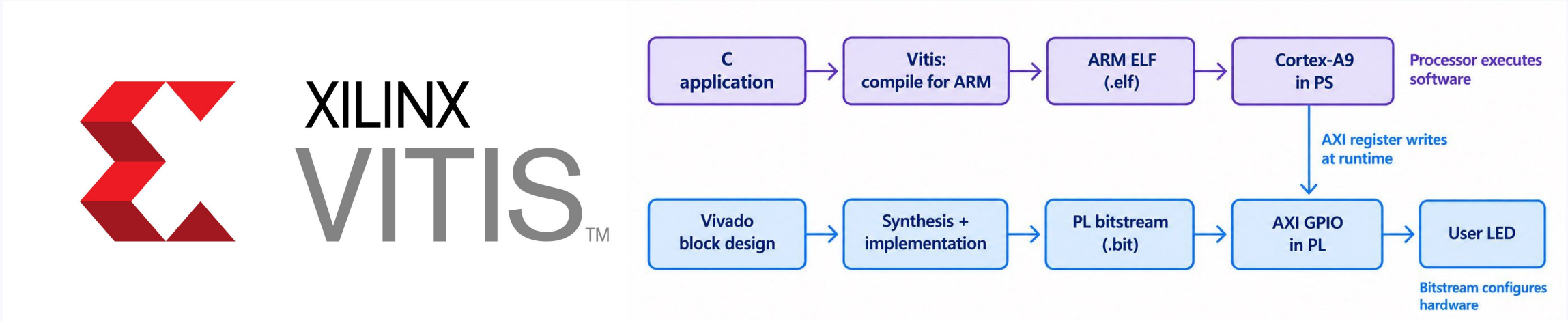
- Start in Vivado
  - Choose the Zybo Z7-10's Zynq device/board configuration
  - Configure the PS and its required peripherals
  - Add AXI GPIO in PL and connect it to the board LED
  - Validate te design, assign addresses, implement hardware
  - Generate bitstream and export the XSA





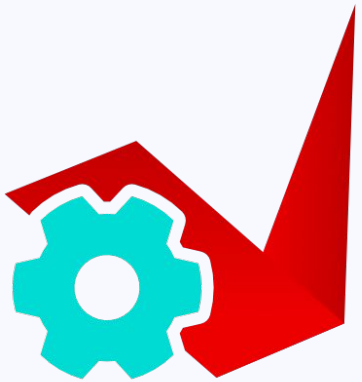
Example Software Development Flow

- In Vitis
  - Create an application using the XSA file
  - Compile C for the ARM processor, producing an ELF
  - Initialize the AXI GPIO driver
  - Write GPIO values to switch the LED on and off
  - Print a startup message through the configured UART (JTAG)
  - Rebuild software after changing C; rebuild hardware (in vivado) after making a hardware change.



Example Debugging Flow

- In Vitis as well
  - Use JTAG to load/run or debug during development
  - Watch the LED for evidence that the PL GPIO path works
  - Watch a serial terminal for UART text
  - Set a breakpoint at the GPIO write to inspect software execution
  - If software steps correctly but LED stays dark, investigate hardware
  - Debugging the CPU and observing internal PL signals are different tasks

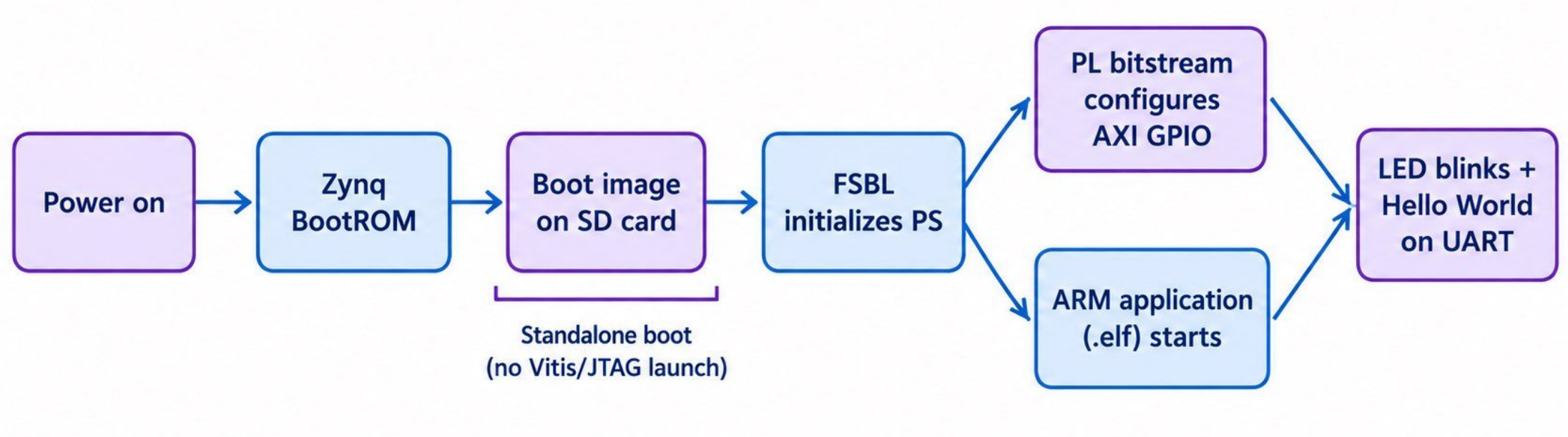






Run from Power-Up

- A JTAG launch is a development workflow
- Autonomous startup needs the board's supported boot path
- Boott components must initialize the system and load the application
- If the design needs PL hardware, that configuration must also be handled
- Build and test the boot image separately from the JTAG demo



Vitis So Far

- Processors and PL offer different capabilities
- Software must match *specific hardware* we built
- Vivado exports that hardware description and Vitis builds a platform around it
- Applications run on a selected processor and use the available hardware
- Build, debug, and standalone boot are separate parts and milestones for a working Vitis project



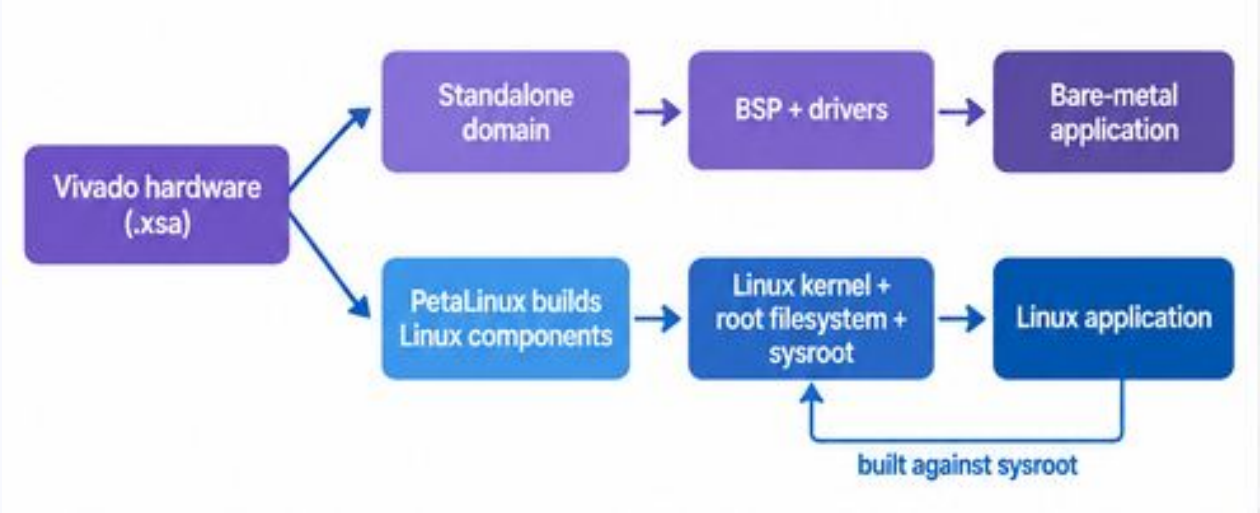






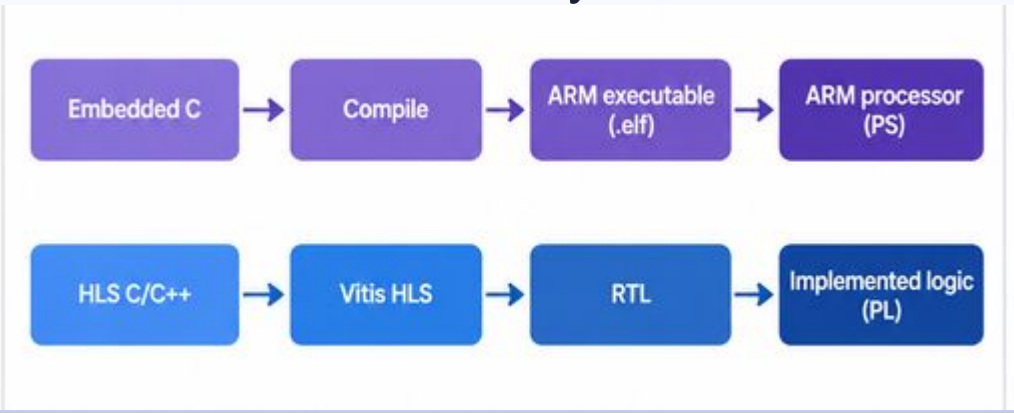
Some Final Things to Mention

- Petalinux
  - Way too much to cover today but the basic idea is:
- What if the SoC runs Linux?
  - **Standalone**: one application runs without a general purpose OS
  - **Embedded Linux:** the processor boots a kernel and runs user-space programs
  - Linux needs a kernel, root filesystem, boot components, and hardware description
  - **PetaLinux** is an AMD toolset for building/customizing those Linux components
  - Vitis can build applications against a Linux target and it's **sysroot**



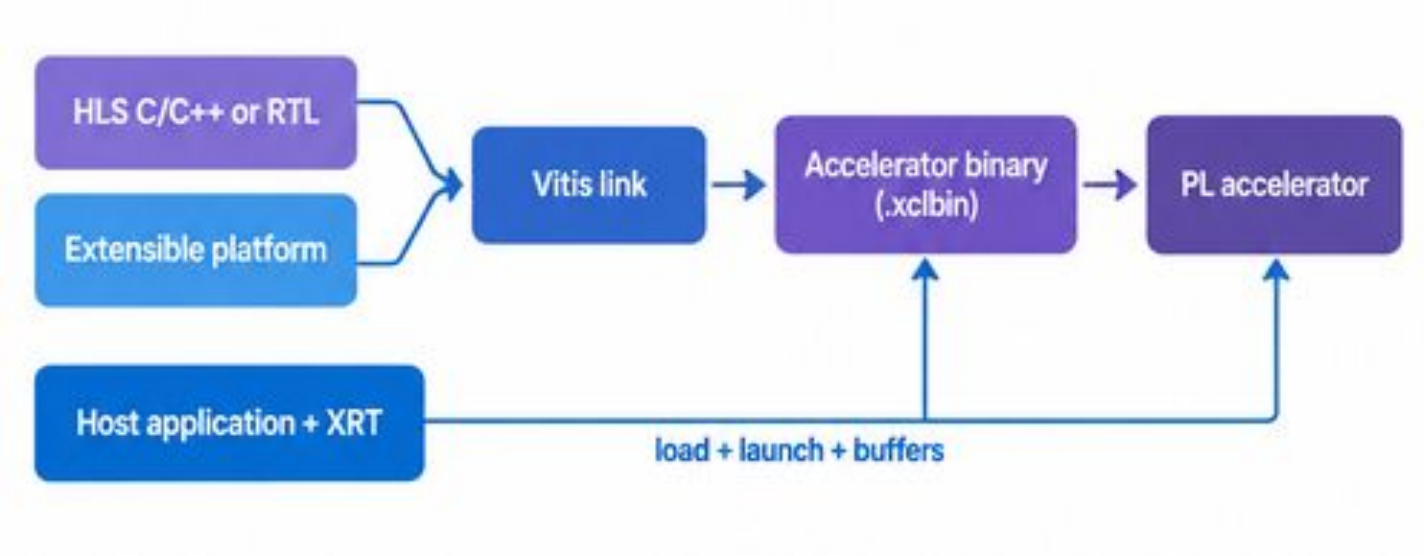
Some Final Things to Mention

- Vitis HLS
  - Again, way too much to cover now, but useful to know for Guitar Tuner
- RTL sucks to rewrite every single time, it's basically hardware assembly
  - What if there was a faster way?
- High-Level Synthesis (HLS)!!
  - Ordinary embedded C compiles into instructions for a processor
  - **Vitis HLS** synthesizes suitable C/C++ functions into RTL for PL
  - Directives help shape pipelines, parallelism, memory, and interfaces
  - C simulation checks the source; C/RTL Co-sim checks generated hardware
  - HLS output can become Vivado IP or a Vitis HW Kernel
  - HLS does **not** remove the need to verify hardware or meet timing requirements



Some Final Things to Mention

- XRT and Hardware Kernels
  - A hardware kernel is a PL compute function built for Vitis acceleration flow
  - Kernels can originate from HLS or packaged RTL
  - An extensible platform supplies the infrastructure into which kernels are integrated
  - The linked device binary, the .xclbin, carries the accelerator design.
  - A host application can use XRT to load the binary, manage buffers, launch kernels, and retrieve results



The Big Vitis Picture

- Our Example Zybo project: **embedded software on a fixed hardware platform**
- Acceleration projects: **extensible hardware platforms** accept linked kernels
- **Vitis HLS** can generate RTL code from suitable C/C++
- RTL kernels can also participate in acceleration flows
- XRT is relevant to certain host-controlled accelerator systems.
- None of these acceleration pieces is required, but all deserve a purpose





FPGAtors

Questions?



1