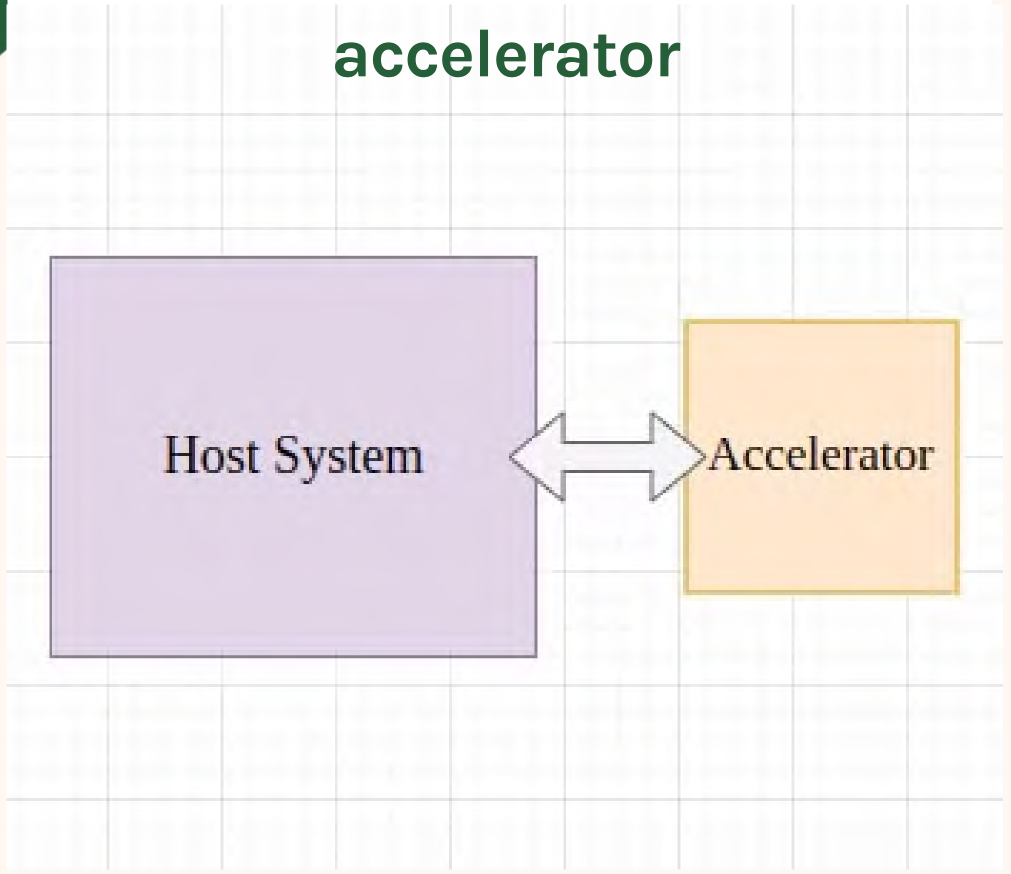
INTRODUCTION TO PYNQ

FPGA Accelerators

Traditional FPGA accelerator



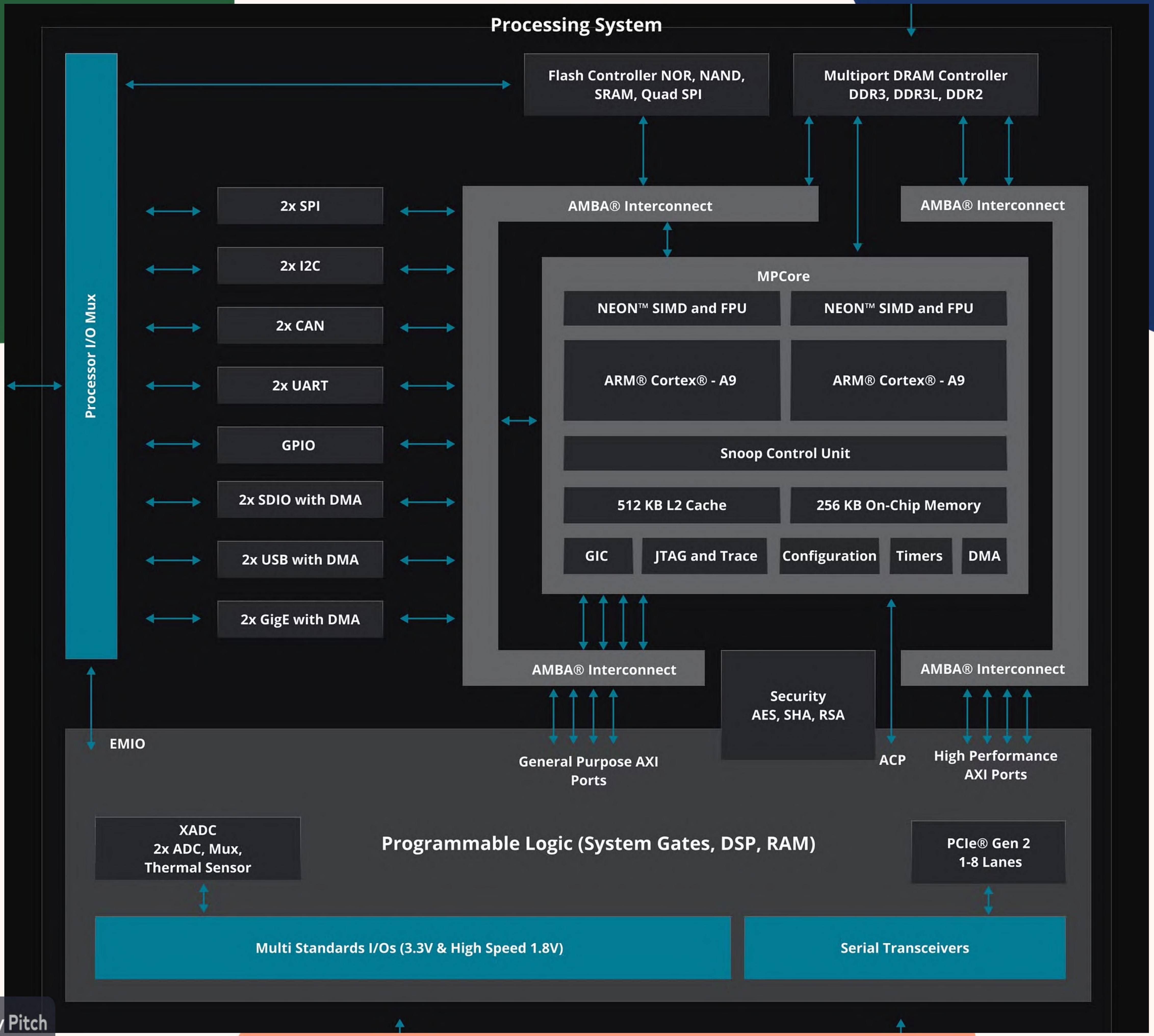
Host system properties

- Is able to execute instructions
- Can ask for 'help' from accel
- Can reply back to accel
- Can have multiple peripherals/accels

Accelerator properties

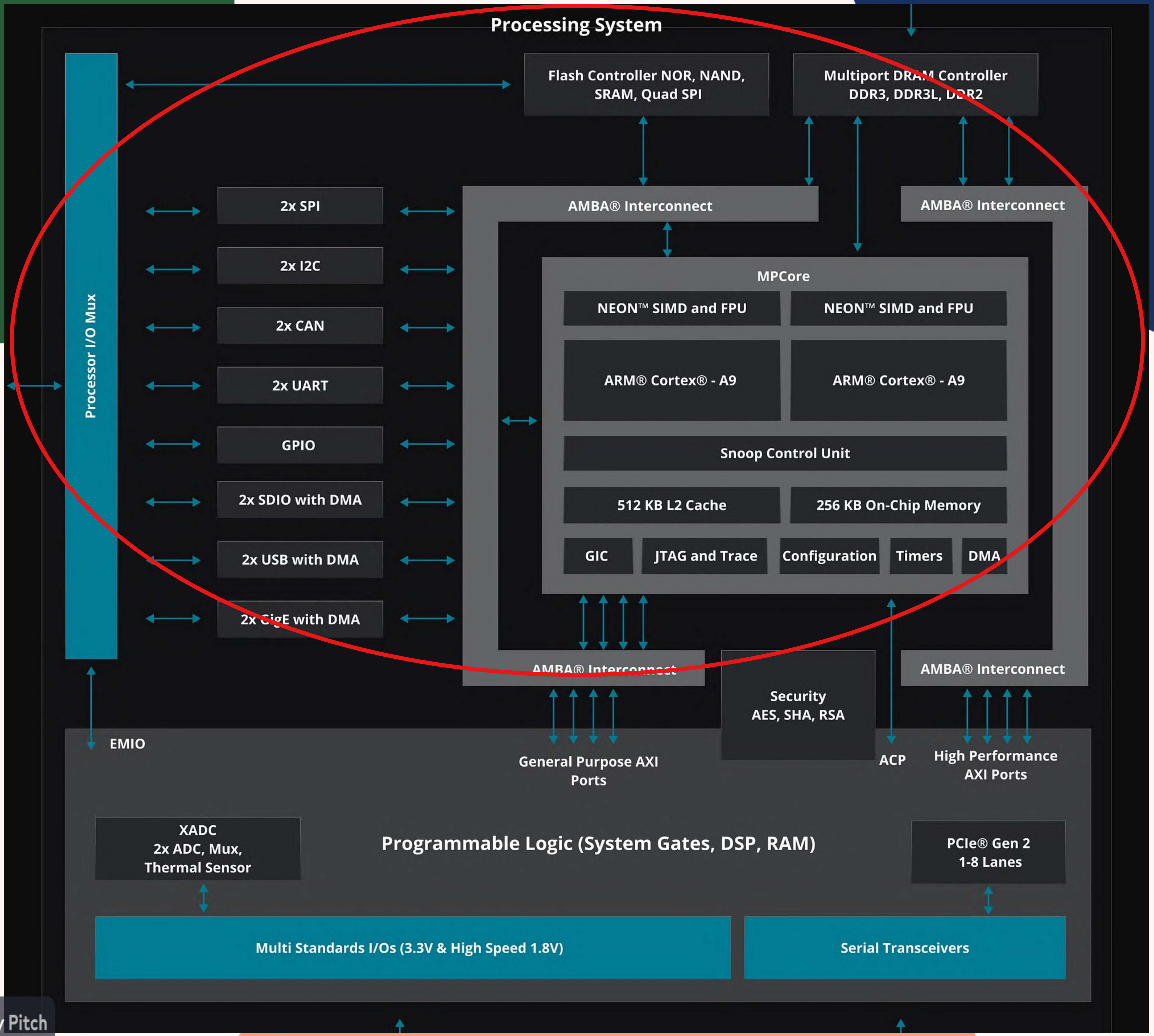
- Not a complete system
- Is good at one type of workload
- Can talk to host system
- Can be reconfigured to do something else

Try Pitch



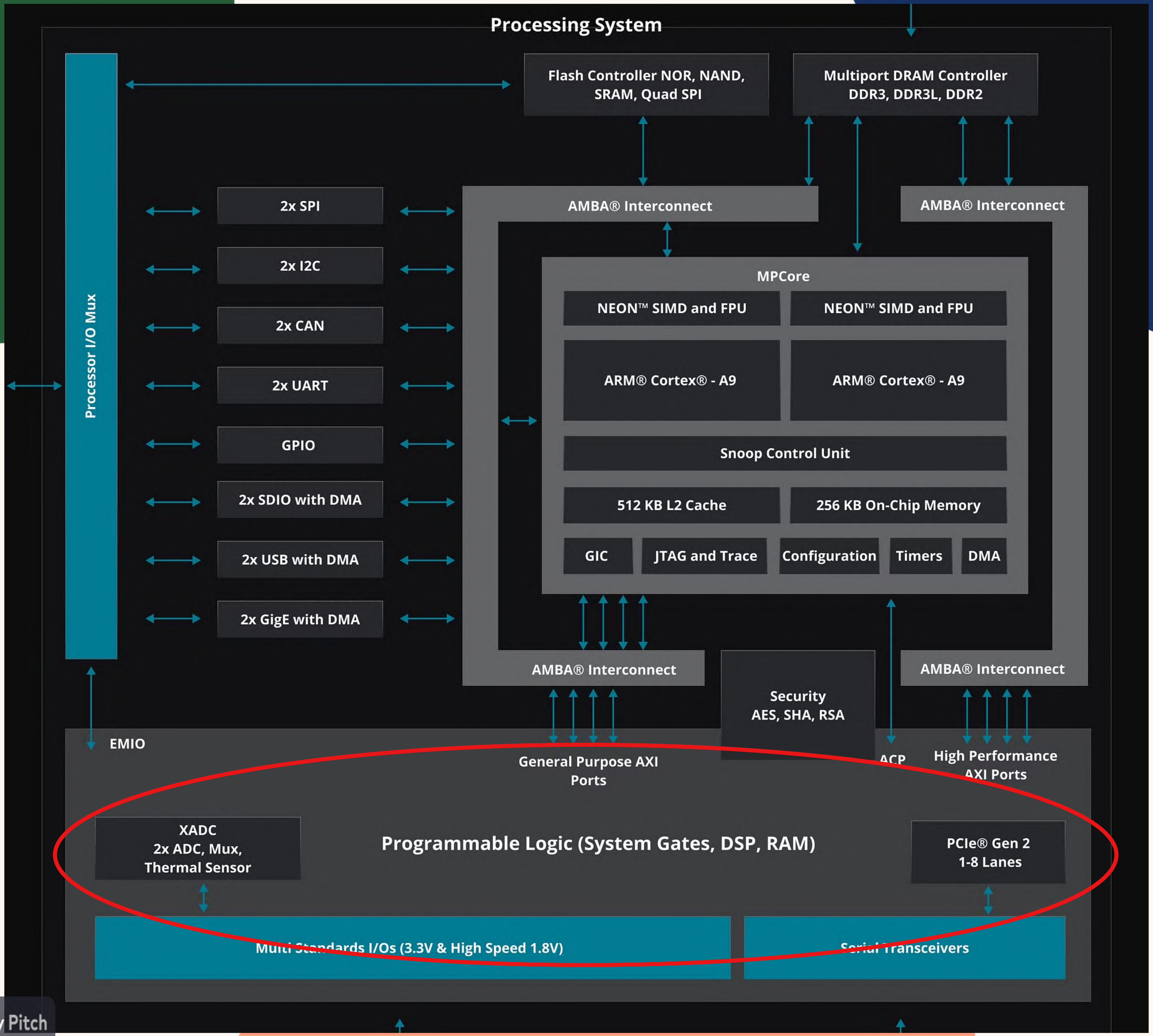
AMD PS/PL

- Host - PS - Processing System
- Accel - PL - Programmable Logic



Processing system

- ARM-based processing system
- CPU cores, memory IO, etc
- Hosts the operating system, executes software



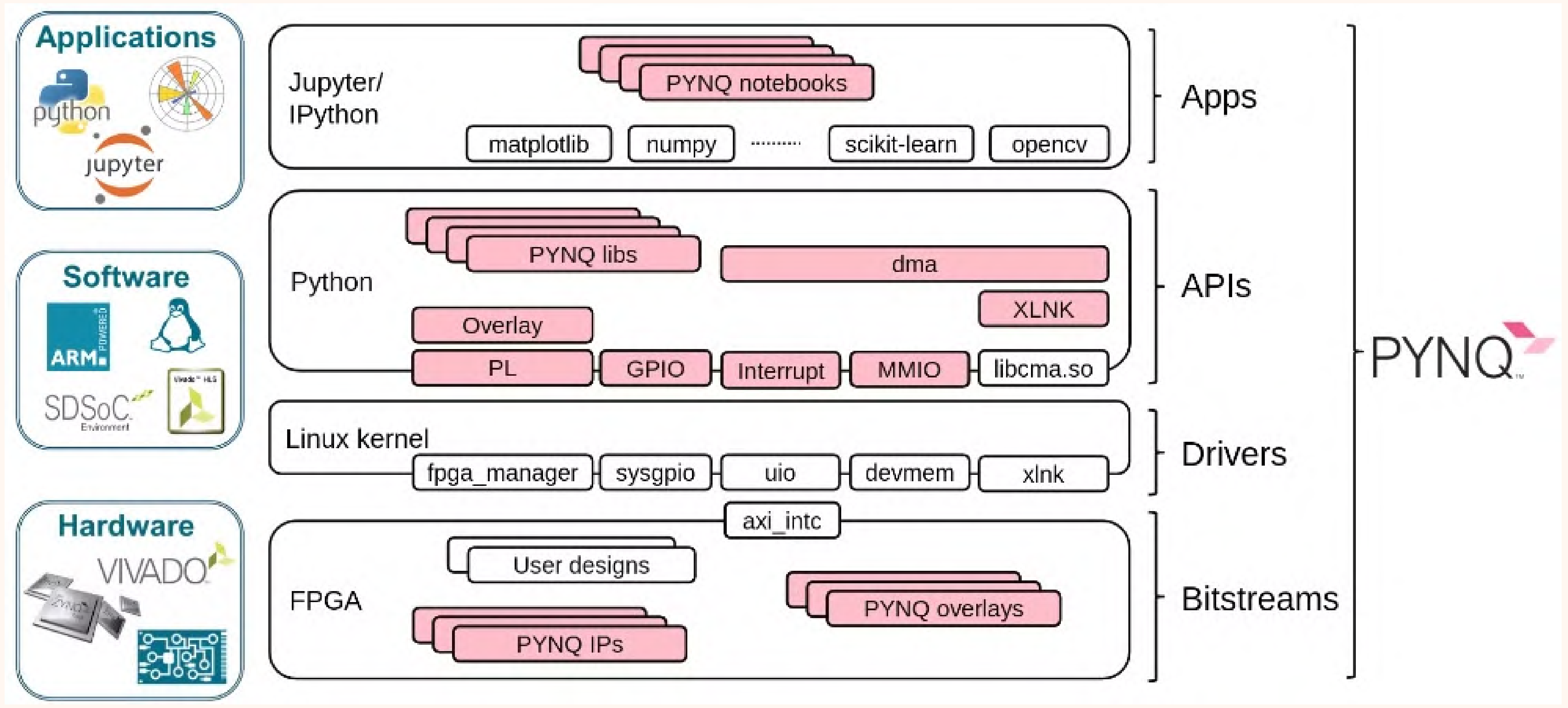
Try Pitch

Programmable Logic

- The FPGA fabric part
- Used to implement custom hardware accelerators
- Made up of LUTs, BRAMs, DSP blocks

What is PYNQ?

PYNQ - Python on ZYNQ

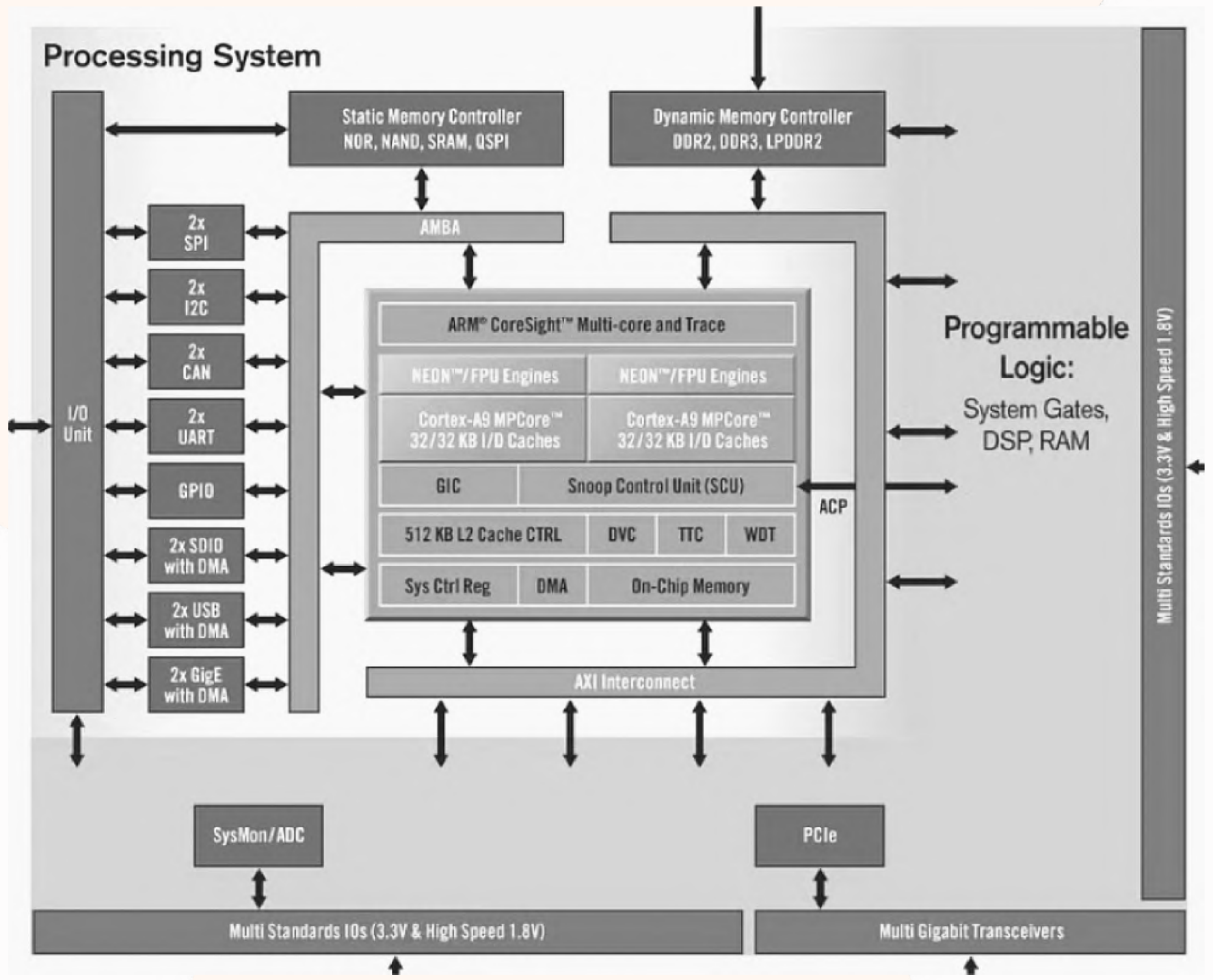


Layers

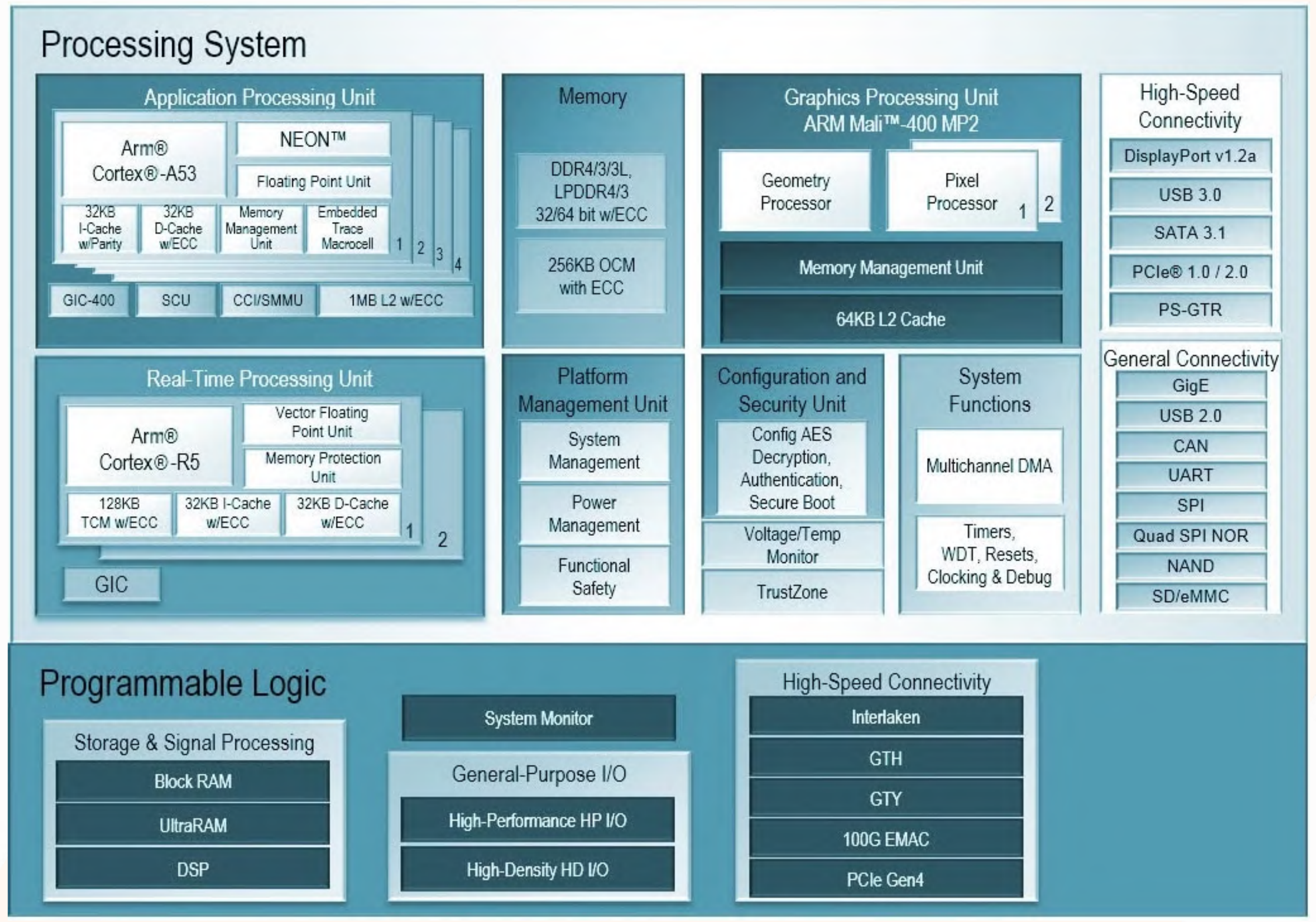
1. Jupyter Notebooks
2. IPython Kernel
3. Ubuntu-Based Linux
4. ARM A9 core <-> PL

Try Pitch

PYNQ Z2 vs PYNQ Ultra-scale

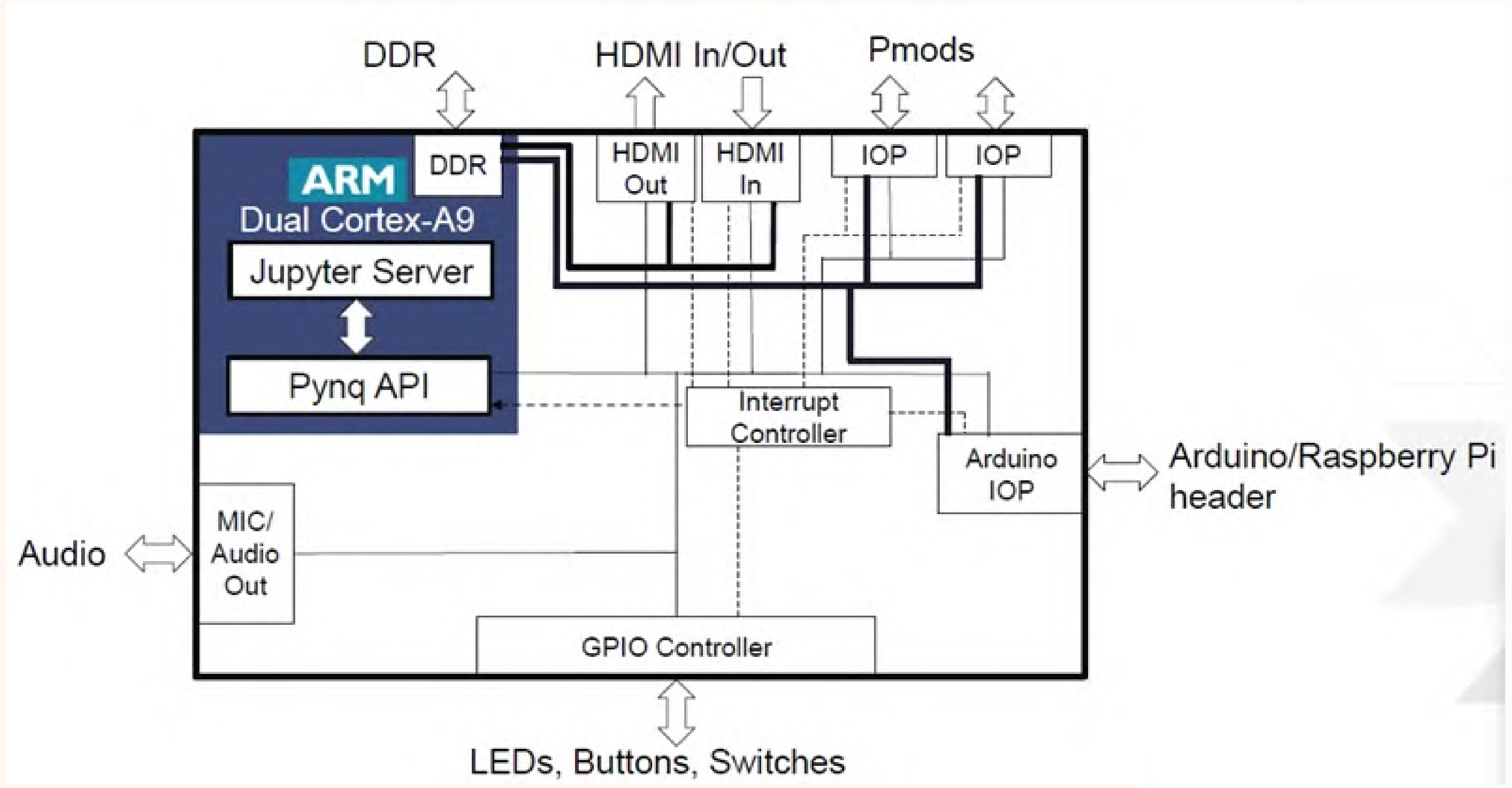

Processing System
Static Memory Controller NOR, NAND, SRAM, QSPI
Dynamic Memory Controller DDR2, DDR3, LPDDR2
AMBA
I/O Unit
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
ARM® CoreSight™ Multi-core and Trace
NEON™/FPU Engines
NEON™/FPU Engines
Cortex-A9 MPCore™ 32/32 KB I/D Caches
Cortex-A9 MPCore™ 32/32 KB I/D Caches
GIC
Snoop Control Unit (SCU)
512 KB L2 Cache CTRL
DVC
TTC
WDT
Sys Ctrl Reg
DMA
On-Chip Memory
ACP
AXI Interconnect
Programmable Logic:
System Gates, DSP, RAM
Multi Standards IOs (3.3V & High Speed 1.8V)
SysMon/ADC
PCIe
Multi Standards IOs (3.3V & High Speed 1.8V)
Multi Gigabit Transceivers

- ZNYQ Z-7020 SoC
- Dual-core Arm Cortex-A9
- 85k Logic cells
- 4.9 Mb Bram



- ZYNQ Ultrascale+ (XCZU5EG-1SFVC784)
- Quad-core Arm Cortex-A53
- Graphics Unit - Arm Mali-400
- 256k Logic cells
- 26.6 Mb RAM

Base overlay bitstream



- Loaded into the PL when the PYNQ-Z2 boots up
- Access to its onboard and external peripheral interfaces via Python
- **Onboard peripherals** - controlled by a fixed GPIO controller
- **External peripherals** - controlled using programmable MicroBlaze IO Processors

- **Limited functionality**

Try Pitch

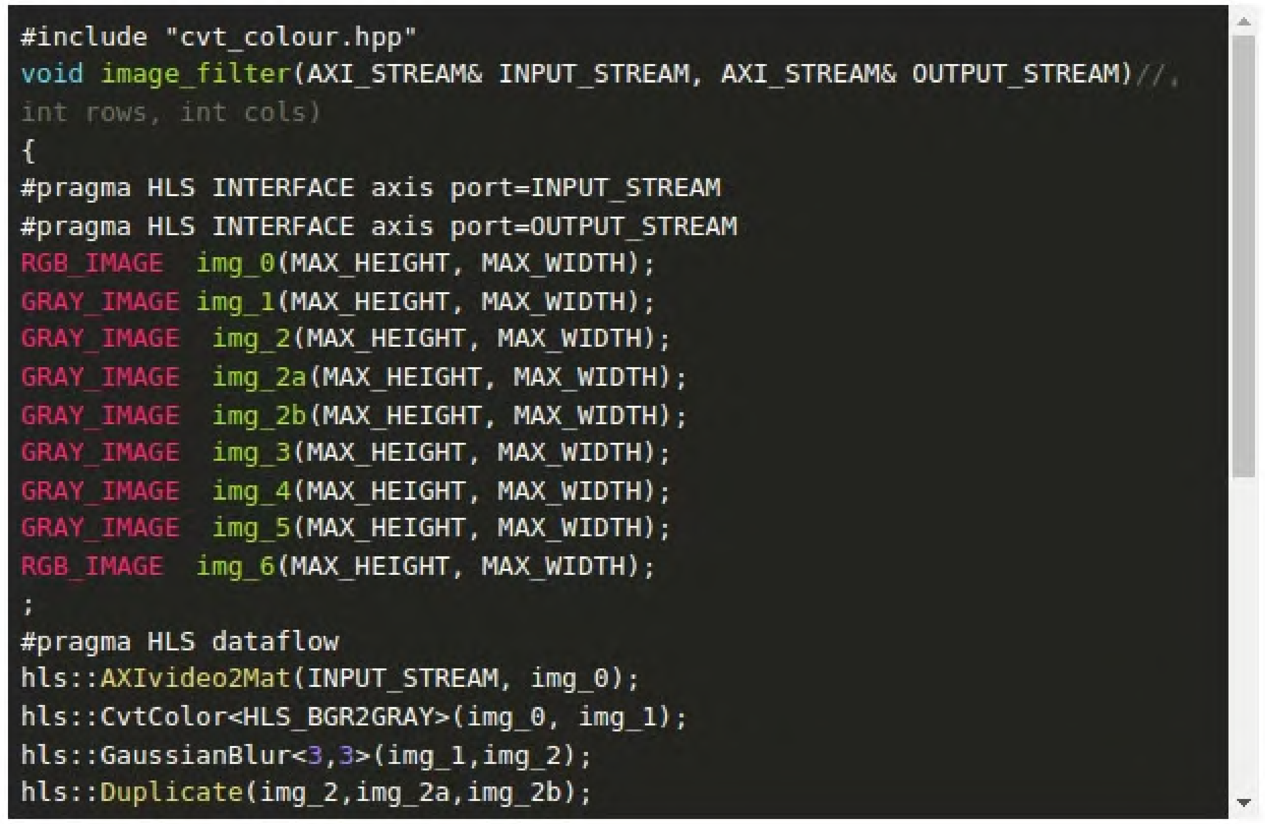
Exercise 1

Write a python program that implements a boolean gate. Use the switches or buttons as inputs, and LEDs as outputs.

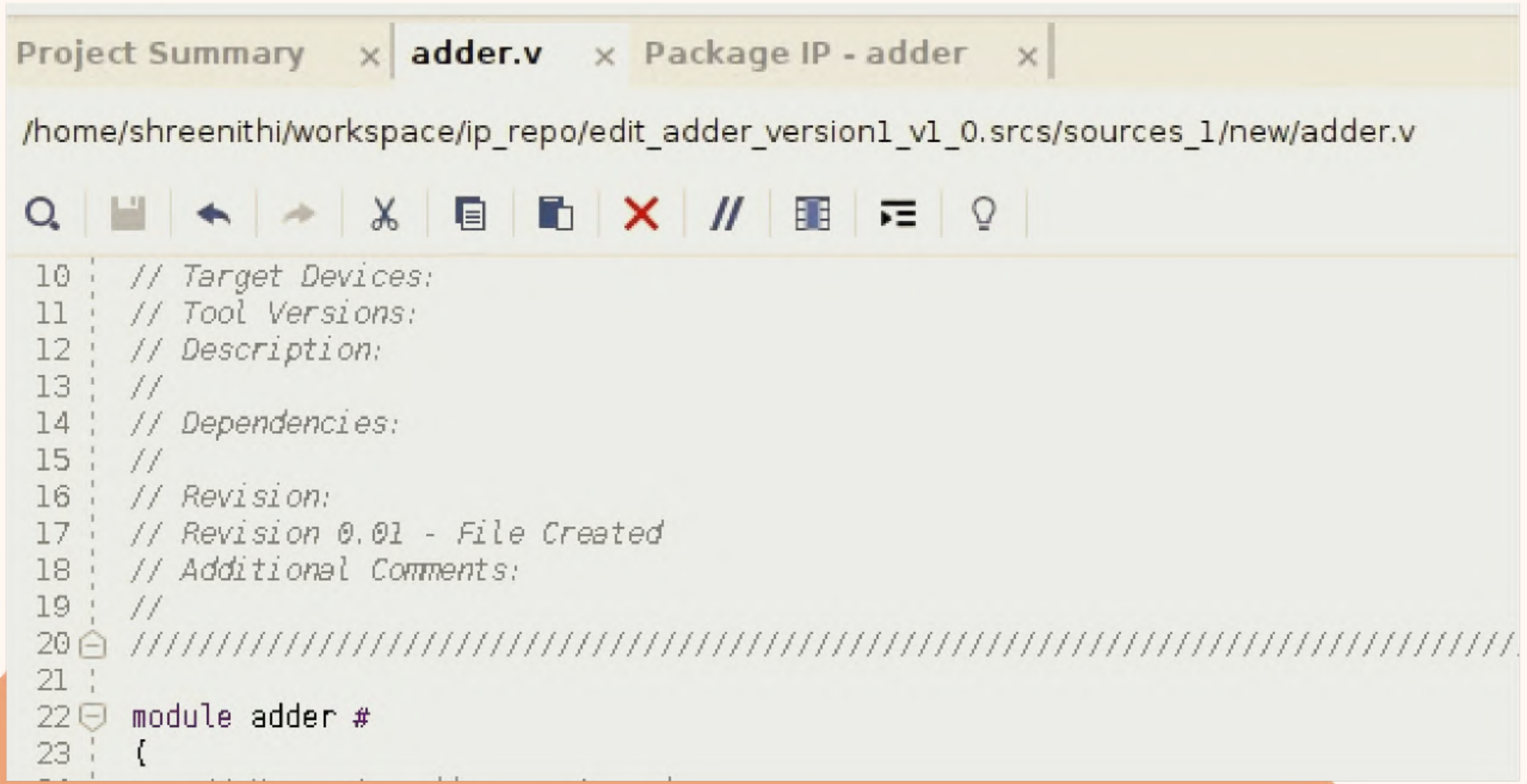
Hint: Reference notebook at link

Custom Overlays?

1 Create or Import Custom IP



```
#include "cvt_colour.hpp"
void image_filter(AXI_STREAM& INPUT_STREAM, AXI_STREAM& OUTPUT_STREAM)//,
int rows, int cols)
{
#pragma HLS INTERFACE axis port=INPUT_STREAM
#pragma HLS INTERFACE axis port=OUTPUT_STREAM
RGB_IMAGE  img_0(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE img_1(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_2(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_2a(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_2b(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_3(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_4(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_5(MAX_HEIGHT, MAX_WIDTH);
RGB_IMAGE  img_6(MAX_HEIGHT, MAX_WIDTH);
;
#pragma HLS dataflow
hls::AXIvideo2Mat(INPUT_STREAM, img_0);
hls::CvtColor<HLS_BGR2GRAY>(img_0, img_1);
hls::GaussianBlur<3,3>(img_1,img_2);
hls::Duplicate(img_2,img_2a,img_2b);
```

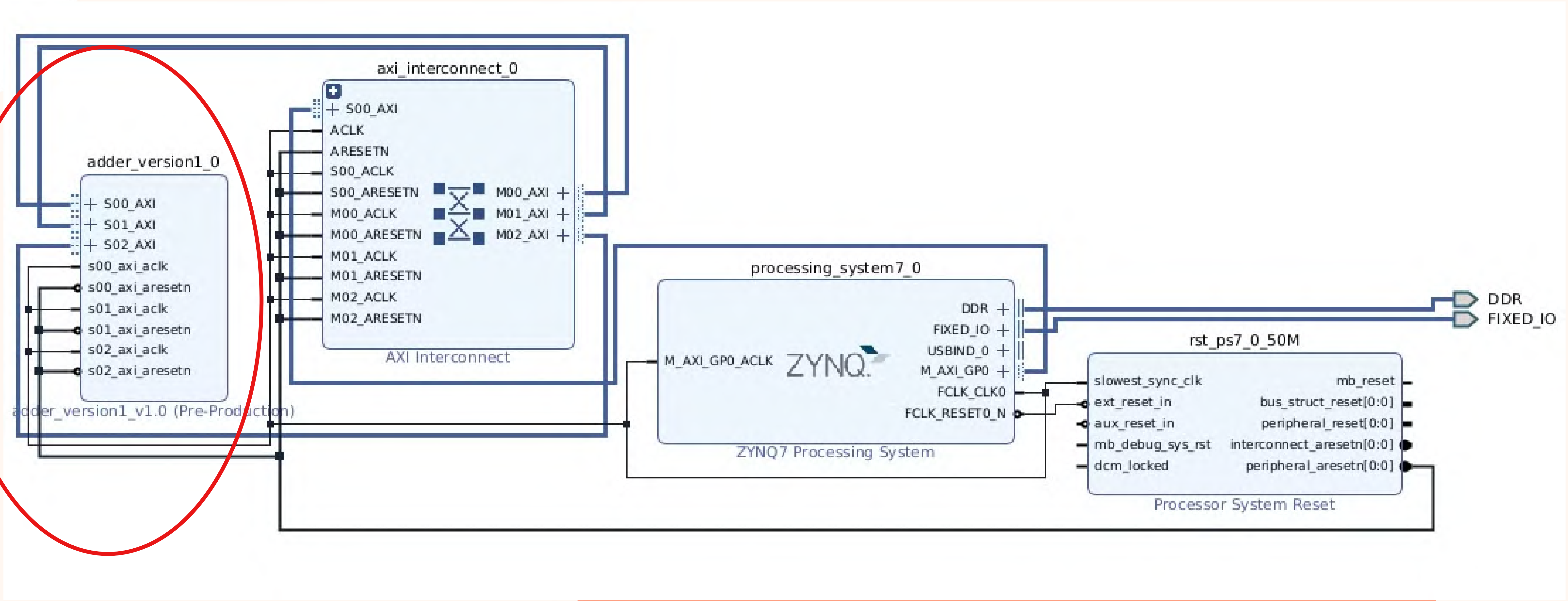


/home/shreenithi/workspace/ip_repo/edit_adder_version1_v1_0.srcs/sources_1/new/adder.v

```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////////////////////

module adder #
(
```

2 Integrate Custom IP with PS

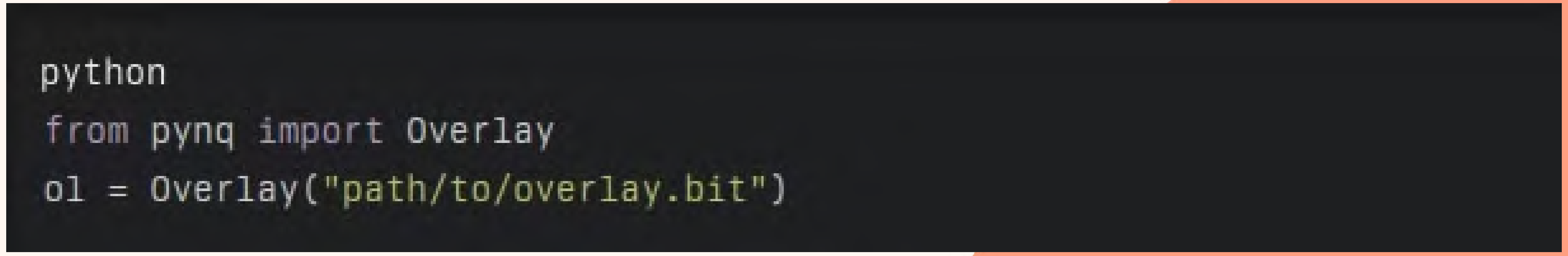


Try Pitch

3 Generate bitstream and hwh files

1. Bitstream (.bit) - binary file generated by Vivado, configures the PL
2. Hardware Hand-off (.hwh) file - Zynq system config, memory map, clk sync, etc

4 Load bitstream using python



```
python
from pynq import Overlay
ol = Overlay("path/to/overlay.bit")
```

Exercise 2

A. Implement (on the PS) edge detection or RGB to grayscale using OpenCV or another python library.

B. Create a custom IP for the same application, load it on the PL, and observe performance benefits.

References

1. Intro: https://github.com/Xilinx/PYNQ_Workshop/blob/master/01_PYNQ_Workshop_introduction.pdf
2. Block diagram: https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html
3. Block diagram: https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-mpsoc.html#tabs-f9ea639ee2-item-bc1aae72dd-tab
4. Base.bit: https://community.element14.com/products/roadtest/b/blog/posts/pynq-z2-dev-kit---working-with-base-overlays
5. Slides: https://github.com/Xilinx/PYNQ_Workshop
6. Cmds: https://github.com/Xilinx/PYNQ/blob/master/pynq/notebooks/common/overlay_download.ipynb
7. Edge detection IP: https://www.hackster.io/adam-taylor/fpga-based-edge-detection-using-hls-192ad2
8. Custom overlay example: https://github.com/wbrueckner/cv2pynq/tree/master?tab=readme-ov-file
9. Start-up guide: https://blog.umer-farooq.com/a-pynq-z2-guide-for-absolute-dummies-part-i-fun-with-leds-and-switches-47dd76abf9a9
10. Official docs: https://pynq.readthedocs.io/en/latest/overlay_design_methodology/overlay_tutorial.html
11. Image proc: https://github.com/ADG4050/Image-processing-PYNQ

Thank You!

Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It’s free)