PYNQ™
Introduction to Overlays

XILINX

> Overlay Concept

> External devices support

> base Overlay

> Python programmer's view

XILINX.

FPGA overlays – hardware libraries

> **Overlays are generic FPGA designs that target multiple users with new design abstractions and tools**

> **Overlay characteristics**

- Post-bitstream programmable via software APIs

- Typically optimized for given application <u>domains</u>

- Encourages the use of open source tools & fast compilation

- Enables productivity by re-using pre-optimized designs

- Makes benefits of FPGAs accessible to new users
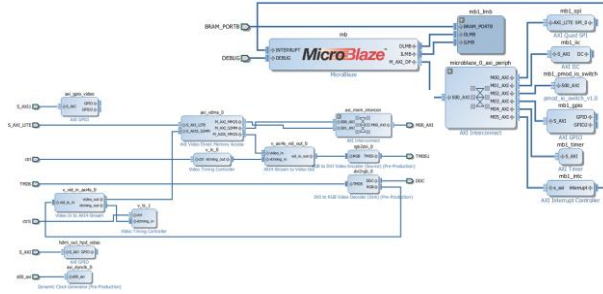
XILINX.

Anatomy of an overlay IP subsystem

> **Designed to be immediately reused by anyone**

>> or re-purposed elsewhere by "person skilled in the art" (PSITA)

> **Comprises**

>> Programmable FPGA IP core

>> FPGA bitstream

>> C code to expose programmable functionality

>> Python-to-C bindings

>> Python library with API

>> Protocol

>> Jupyter notebook examples

XILINX.

FPGA overlays – hardware libraries

Step 1:
Create an FPGA design for a <u>class of related applications</u>

Step 2:
Export the bitstream and a C API for programming the design

Step 3:
Wrap the C API to create a Python library

Step 4:
Import the bitstream and the library in your Python scripts and program

XILINX.

External interfacing with the Base Overlay

Low-cost PYNQ boards: Pmod, Rpi, Arduino Interfaces
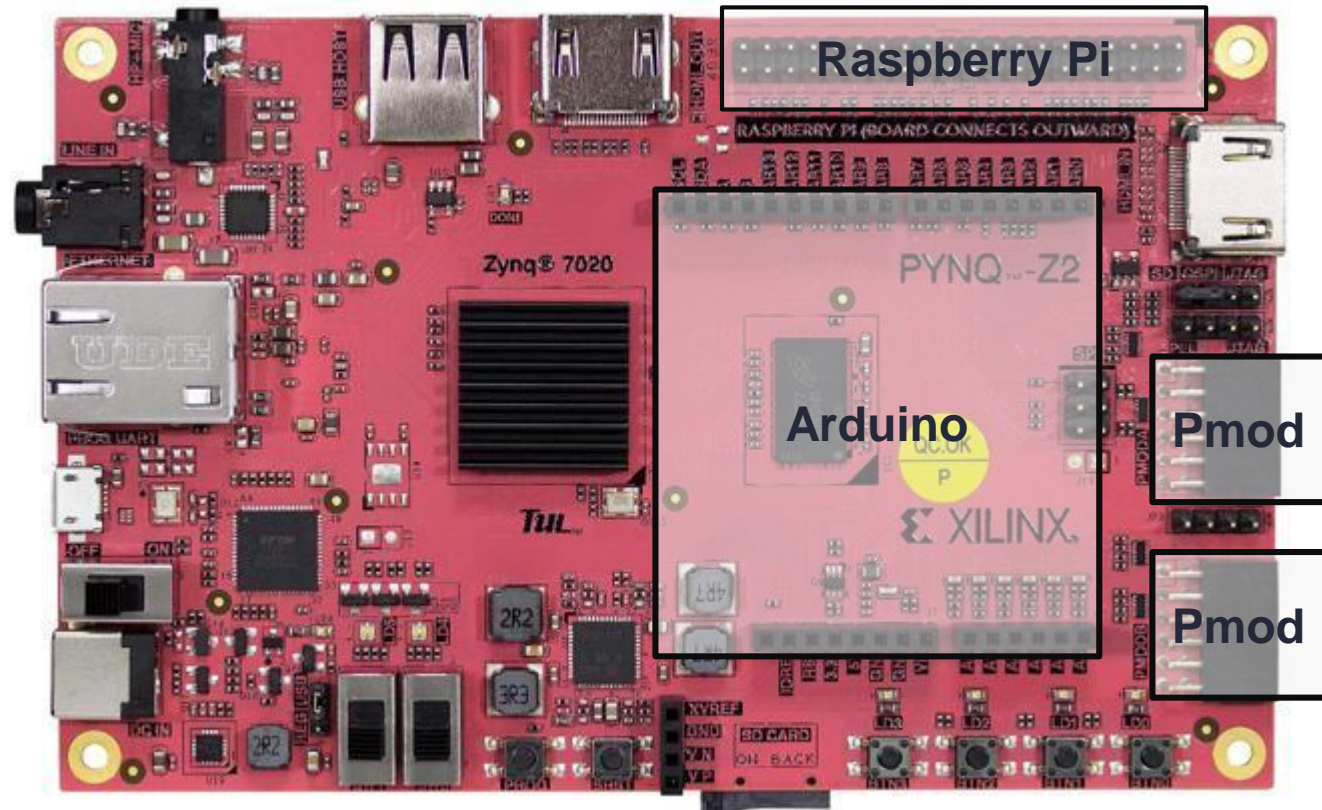
Typically every new Pmod, Raspberry Pi, or Arduino module requires a new design/seperate bitstream

© Copyright 2019 Xilinx

Pmod: many physical & electrical instances

Pmod

Pin	Signal	Description
1 & 5	SCL	Serial Clock
2 & 6	SDA	Serial Data
3 & 7	GND	Power Supply Ground
4 & 8	VCC	Power Supply (3.3V/5V)

ADC

Connector J1

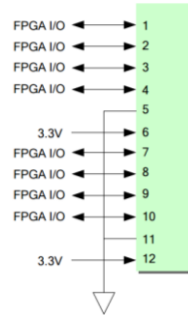| Pin	Signal	Description
1	CS	SPI Chip Select (Slave Select)
2	SDIN	SPI Data In (MOSI)
3	None	Unused Pin
4	SCLK	SPI Clock
7	D/C	Data/Command Control
8	RES	Power Reset
9	VBATC	V_{BAT} Battery Voltage Control
10	VDDC	V_{DD} Logic Voltage Control
5, 11	GND	Power Supply Ground
6, 12	VCC	Power Supply

OLED

Pin	Signal	Description
1	~CS	Chip Select
2	MOSI	Master-Out-Slave-In
3	(NC)	Not Connected
4	SCLK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V/5V)

DAC

VCC GND   8 signals

Pin 6   Pin 1

Pin 12
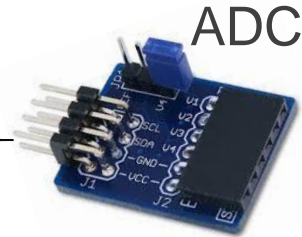
What if we could handle all Pmod instances with a single bitstream?

XILINX

base Overlay MicroBlaze IO Processor (IOP)

The A9 controls the MicroBlaze processors

The A9 delivers the binary code to be executed via dual-ported Instruction BRAMs

The A9 exchange data and control (commands/status) with the MicroBlazes via dual-ported Data BRAMs

Onboard (Fixed) Peripherals

Switches

LEDs

Pushbuttons

AXI-Lite

ARM A9

ARM A9

MB-PMOD Switch — Control Signals, Shared IBRAM, Shared DBRAM — PMOD IO1, PMOD IO2, PMOD IO8

PmodA

PmodB

MB-Arduino Switch — Control Signals, Shared IBRAM, Shared DBRAM

Arduino

XILINX

Configure IO Processor for PmodA

© Copyright 2019 Xilinx

© Copyright 2019 Xilinx

Arduino IO Processor

© Copyright 2019 Xilinx
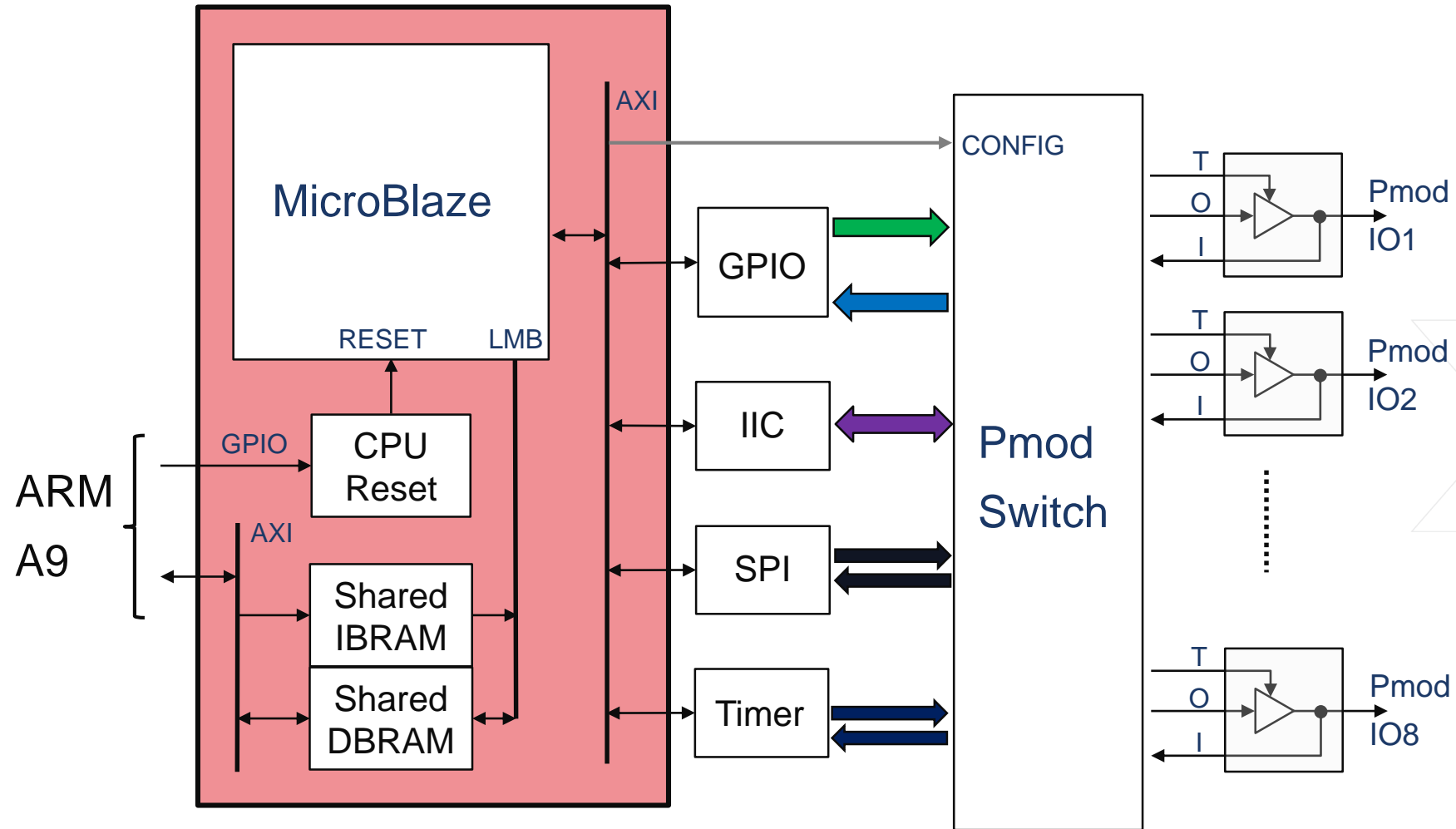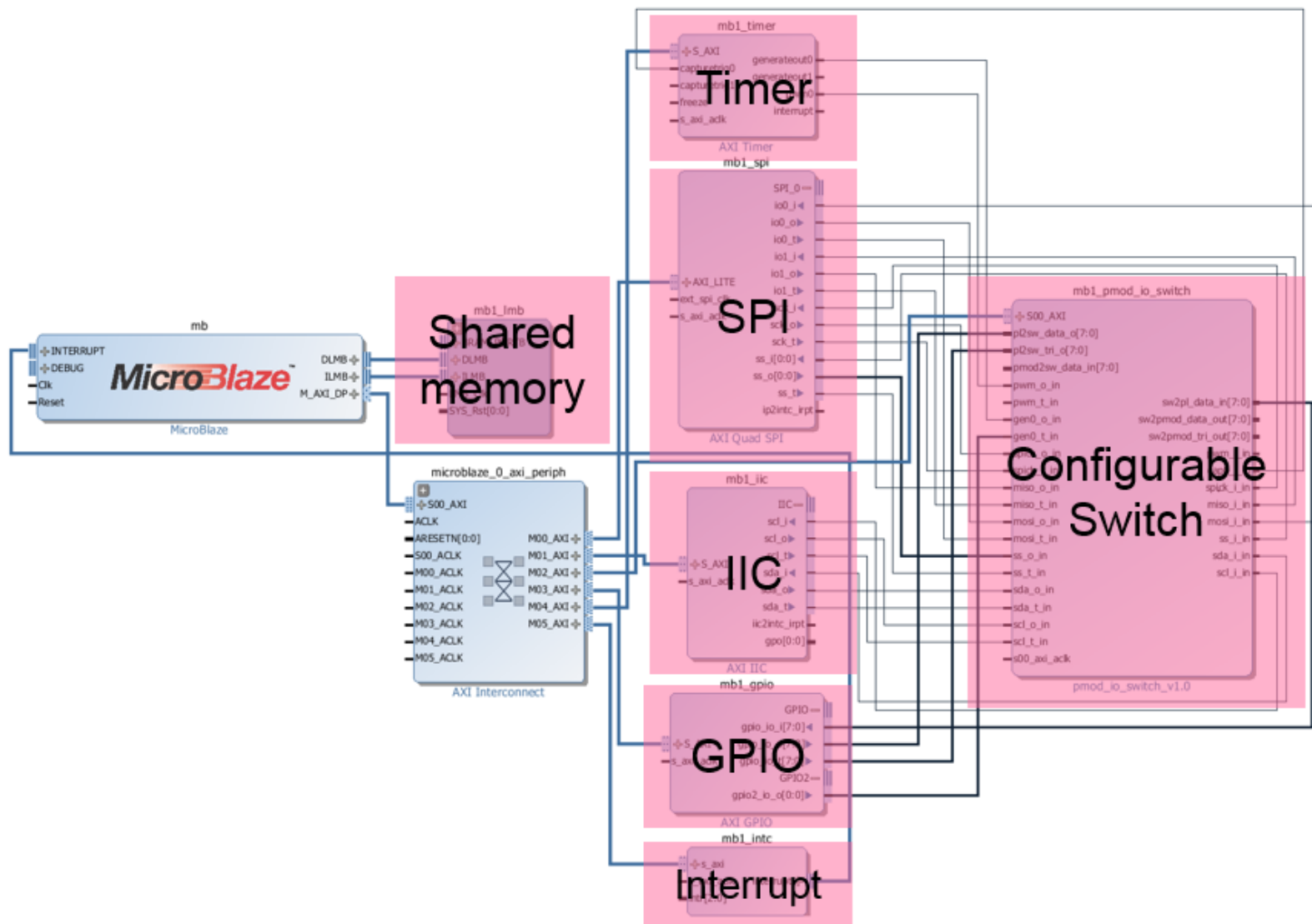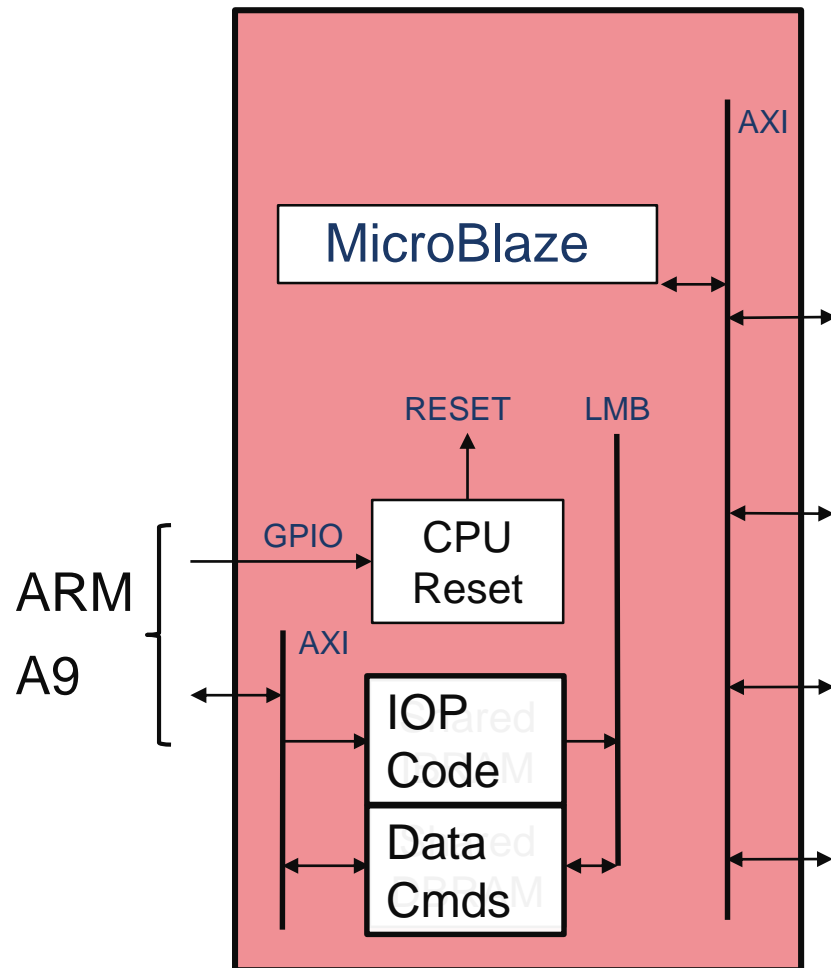
Arduino IOP
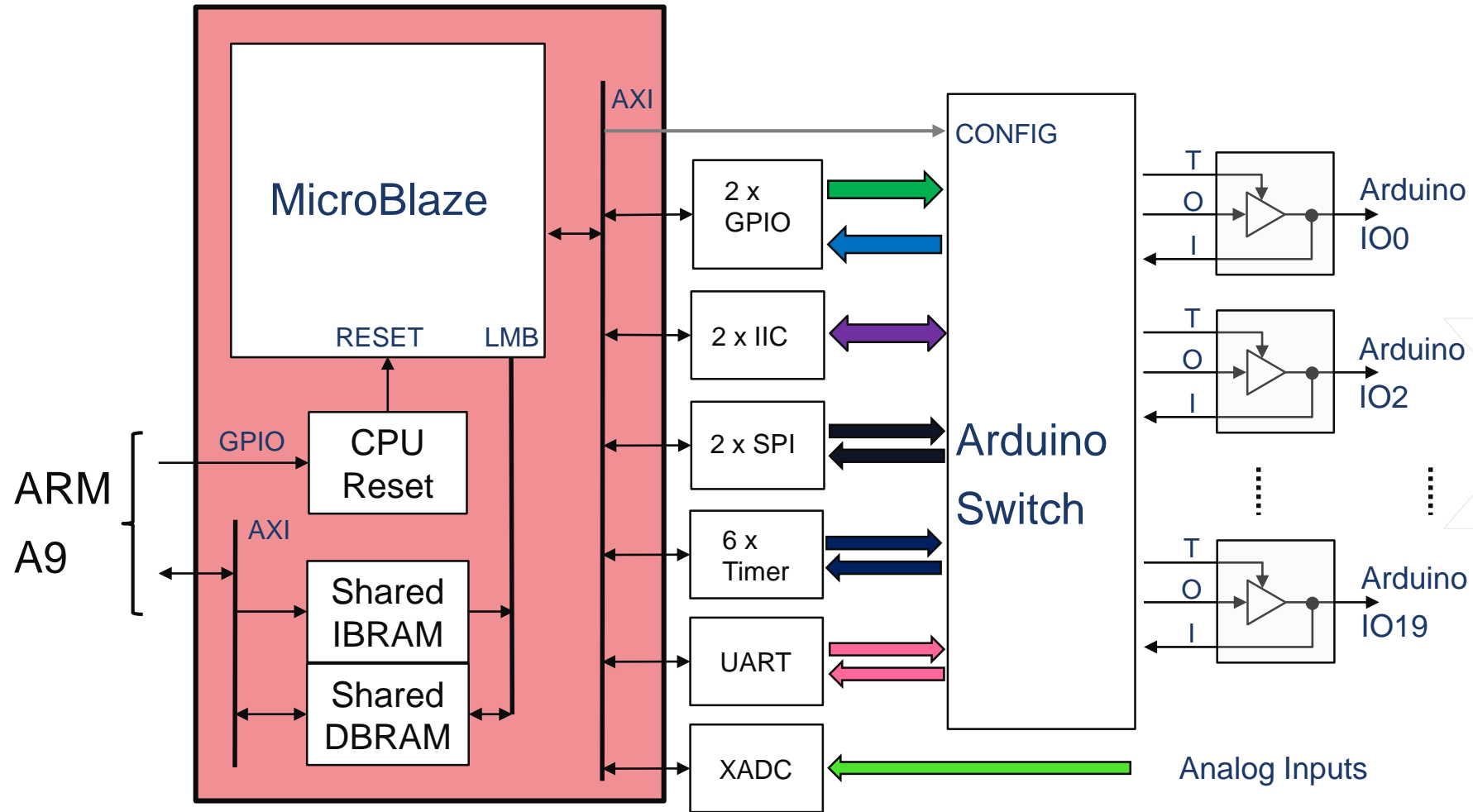
Soft Processor Subsystem (SPS)


The A9 controls the MicroBlaze processors

The A9 delivers the binary code to be executed via dual-ported Instruction BRAMs

The A9 exchange data and control (commands/status) with the MicroBlazes via dual-ported Data BRAMs

Multiple SPS units can be used to control subsystems in the PL fabric: e.g. IO interfaces, internal interfaces, data-path units and instrumentation

SPS can also be used for distributed processing

XILINX.
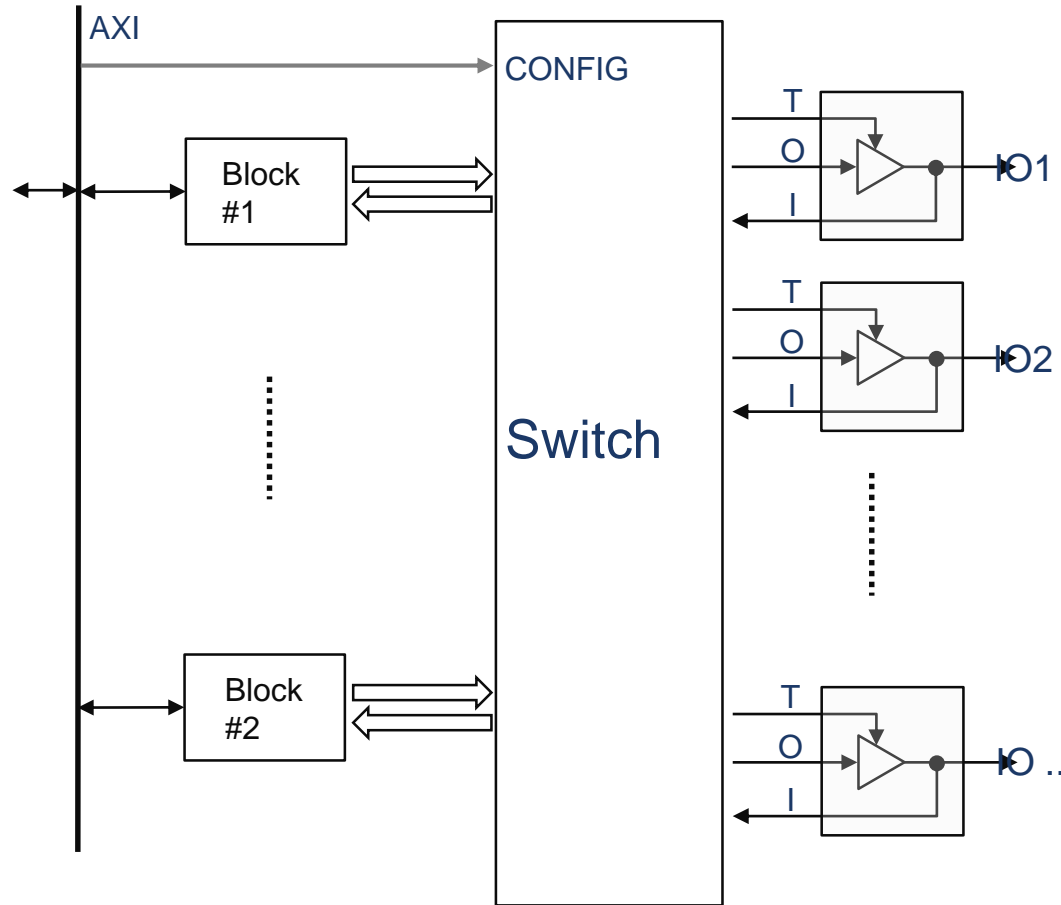
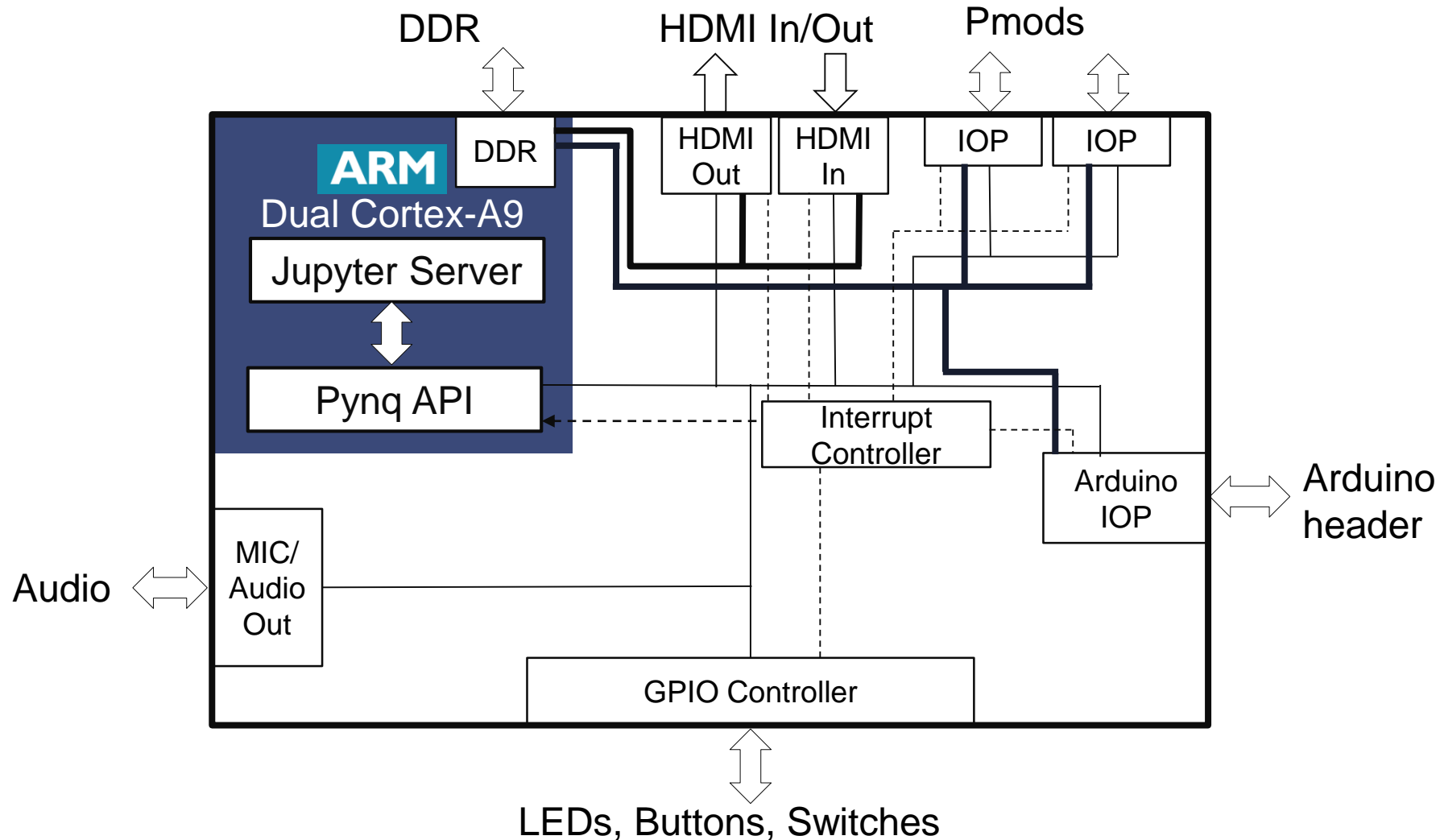# IO Switch (IOS)

The IO Switch
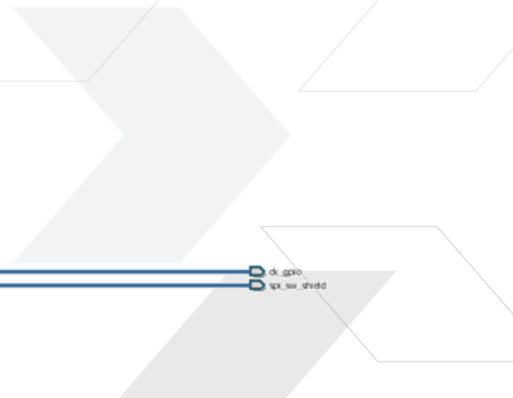
can be re-used to control

any external interface

© Copyright 2019 Xilinx

The rest of the *base* Overlay

Complete *base* Overlay (base.bit)

XILINX

base Overlay – Vivado Interface View

> **PL design of the base Overlay**

> **Standard FPGA design flow used**

> **Vivado IPI**

> **Interface to Python**

> **Memory Map**

> **Open source**

> **Components are re-useable**

Video

> ## HDMI_in, HDMI_out

>> Stream from HDMI_in to DRAM; stream from DRAM to HDMI_out

>> 3 separate DRAM framebuffers available

> ## Image processing on ARM A9s

>> E.g. OpenCV
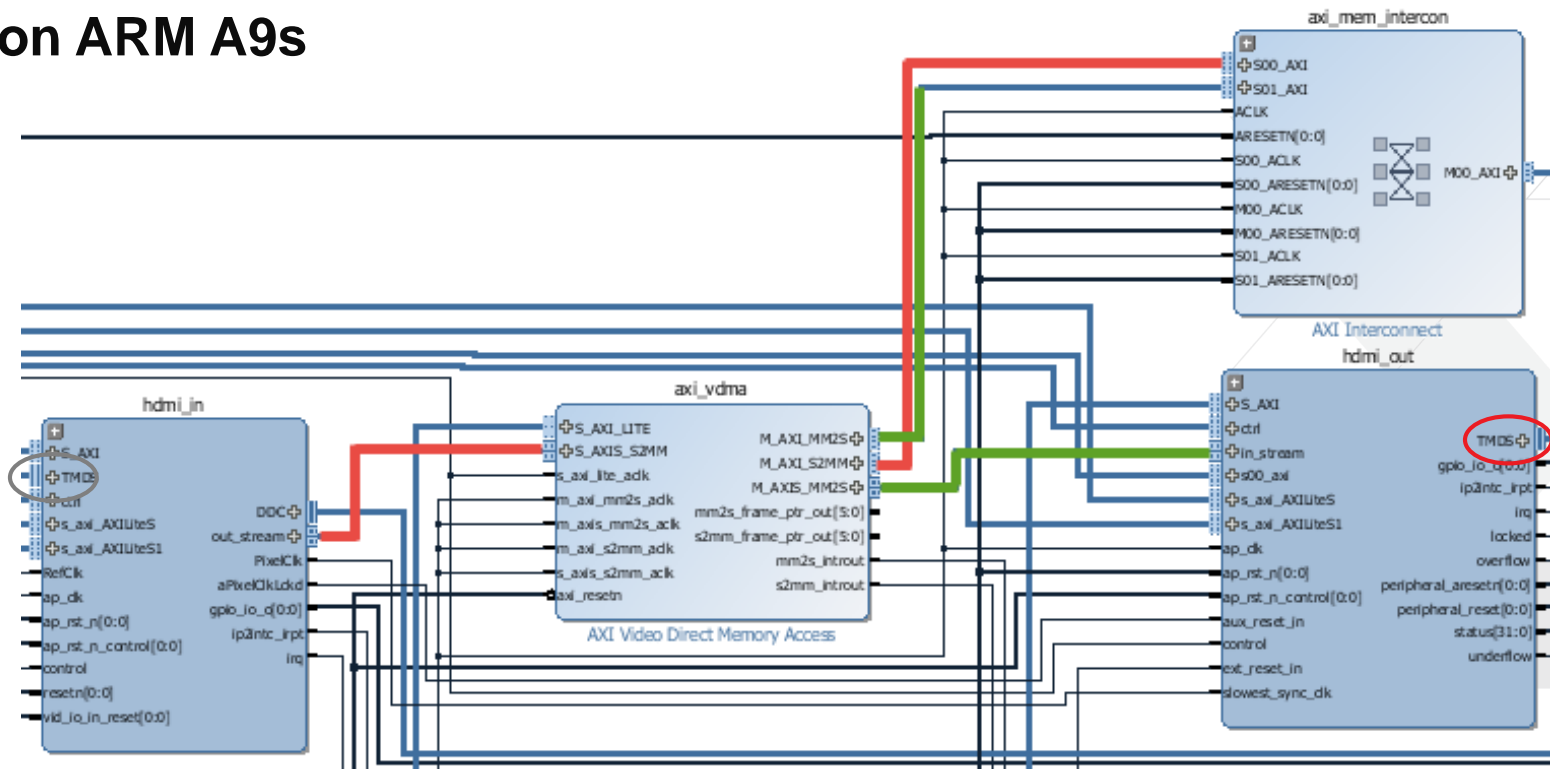
Video In path

> **HDMI_in (TMDS) -> frontend -> color_convert -> axis_register_slice -> pixel_clock -> axis_vdma -> HP0 (PS7)**
>> The frontend module wraps all of the clock and timing logic
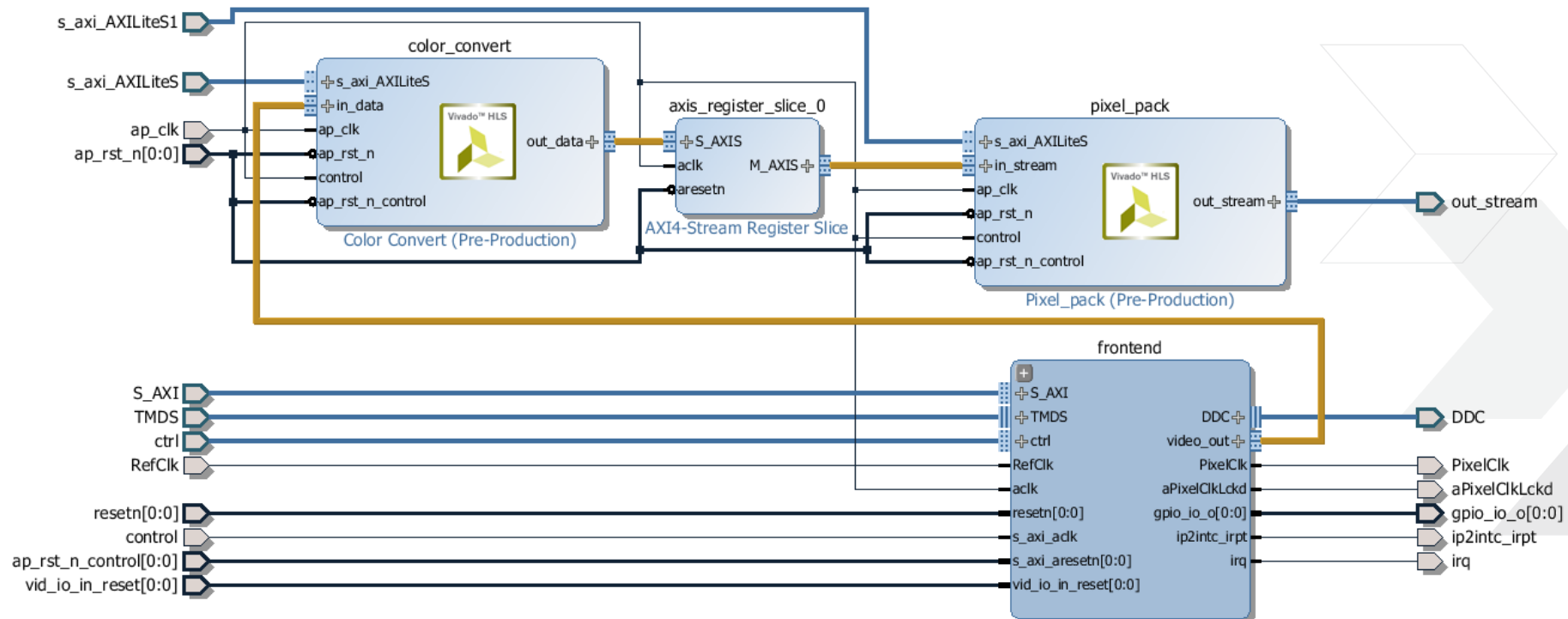
Color conversion and pixel packing

> **color_convert**
>> Transform the input signal into different color spaces

> **pixel_pack**
>> Convert between 8/24/32-bit

> **Default: BGR (24-bit)**
>> RGB (24-bit)
>> RGBA (32-bit)
>> BGR (24-bit)
>> YCbCr (24-bit)
>> Grayscale (8-bit)

> **HLS source available**

color_convert

Color Convert (Pre-Production)

pixel_pack

Pixel_pack (Pre-Production)
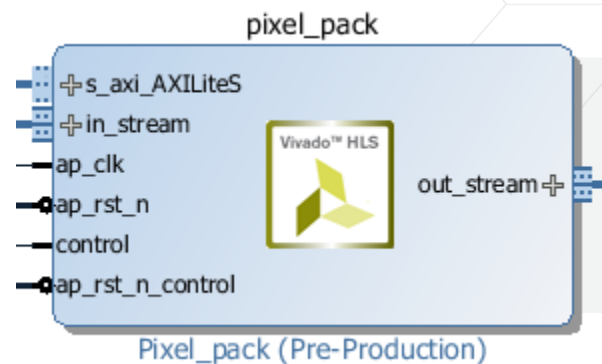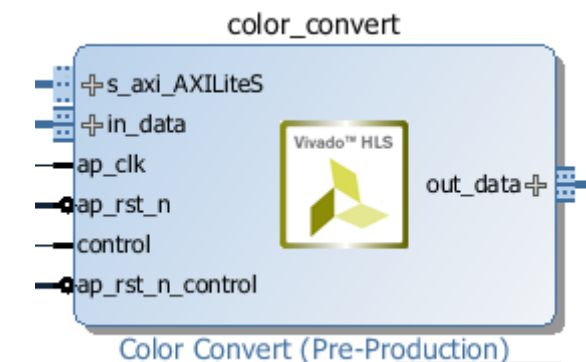
Video Out path

> **HP0 (PS7) -> axi_vdma -> pixel_unpack -> axis_register_slice -> color_convert -> frontend -> HDMI_out (TMDS)**
>> All sub-modules perform reverse operations of the HDMI IN block

© Copyright 2019 Xilinx
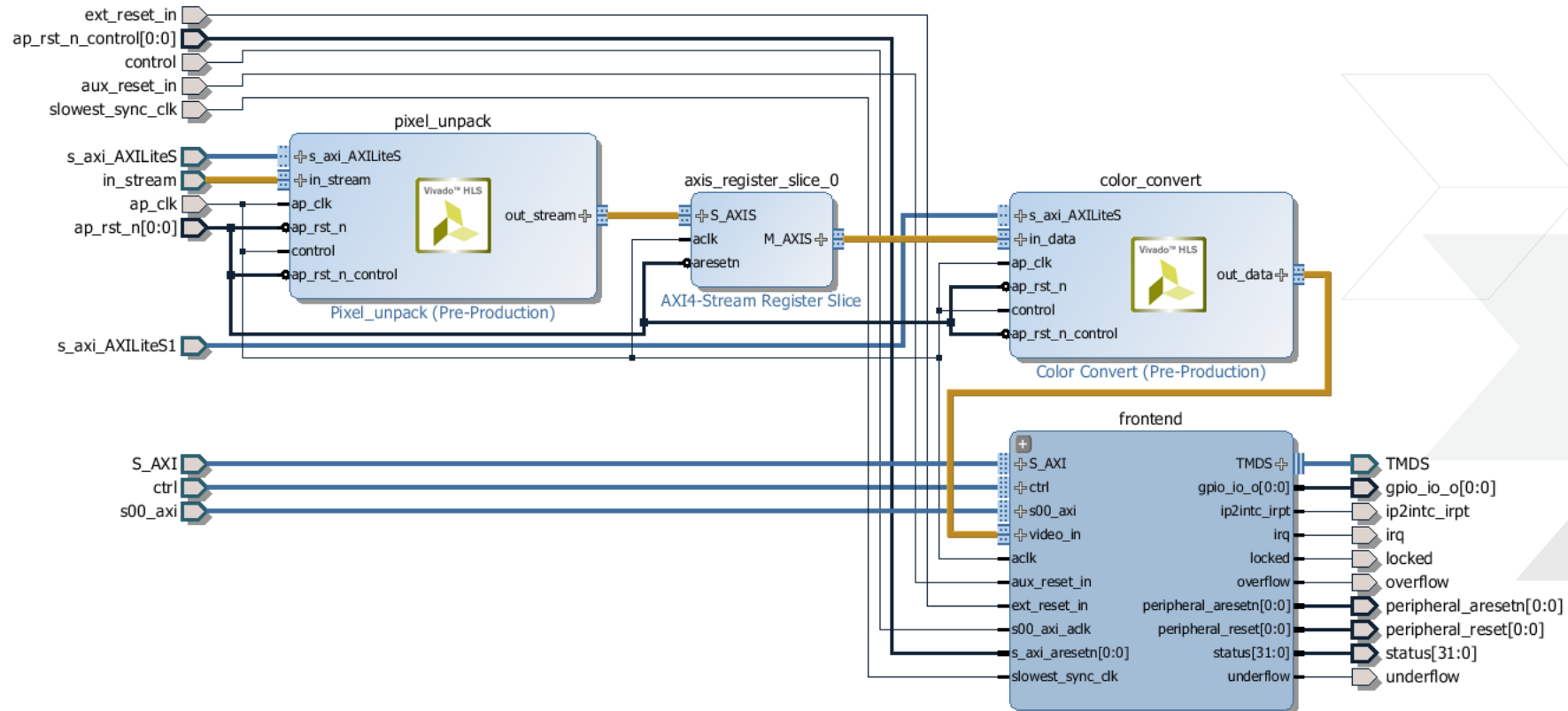
Video example

```
In [1]: from pynq.overlays.base import BaseOverlay
        from pynq.lib.video import *

        base = BaseOverlay("base.bit")
        hdmi_in = base.video.hdmi_in
        hdmi_out = base.video.hdmi_out
```

Python imports,
HDMI instances

```
In [4]: import time

        numframes = 600
        start = time.time()

        for _ in range(numframes):
            f = hdmi_in.readframe()
            hdmi_out.writeframe(f)

        end = time.time()
        print("Frames per second:  " + str(numframes / (end - start)))

Frames per second:  60.08865801337141
```

readframe()

writeframe()

Configure() – HDMI in/out
resolution/colorspace

```
In [2]: hdmi_in.configure()
        hdmi_out.configure(hdmi_in.mode)

        hdmi_in.start()
        hdmi_out.start()
```

```
In [3]: hdmi_in.tie(hdmi_out)
```

Connect HDMI in
to out

https://github.com/Xilinx/PYNQ/blob/master/boards/Pynq-Z1/base/notebooks/video/hdmi_introduction.ipynb

XILINX.

Base Overlay Resource Utilization

➤ Z-7020, LUTs resource utilization ~50%

>> IOP (Pmod) ~ 6 - 10%   IOP (Arduino) ~ 12%
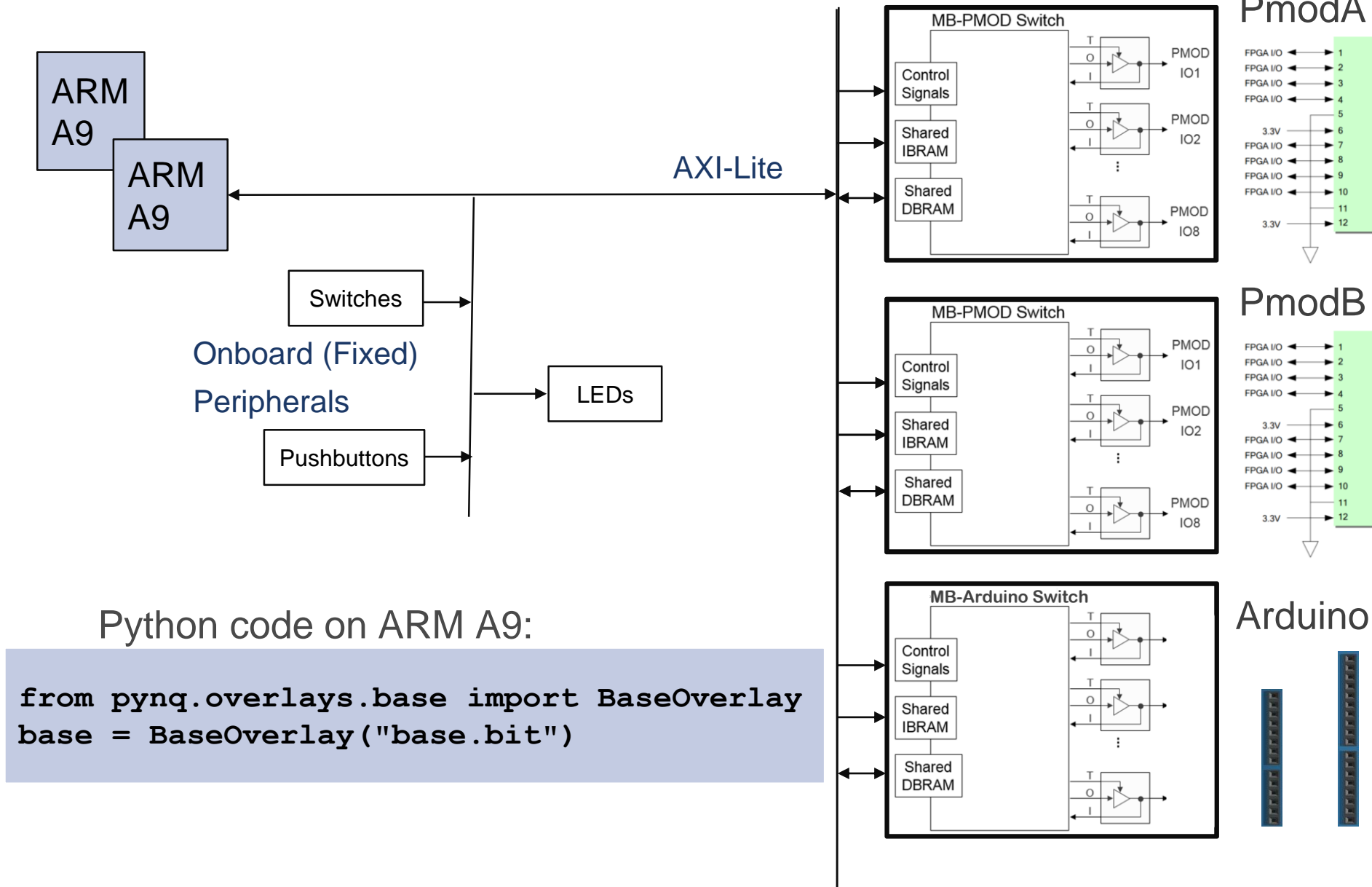
>> Video ~ 15%   Audio ~ 2%

▲ 1 Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	LUT Flip Flop Pairs (53200)	Block RAM Tile (140)	DSPs (220)
⊟ 🅽 top	26821	33503	797	121	10603	24628	2193	12697	58	18
⊟ ▣ system_i (system)	26821	33503	797	121	10603	24628	2193	12697	58	18
⊞ ▣ audio (audio_imp_P...	1238	639	64	32	440	534	704	255	0	0
▣ audio_path_sel (syst...	0	0	0	0	0	0	0	0	0	0
⊞ ▣ axi_interconnect_0 (...	177	134	0	0	78	177	0	90	0	0
⊞ ▣ btns_gpio (system_b...	75	99	0	0	30	75	0	60	0	0
▣ concat_interrupts (s...	0	0	0	0	0	0	0	0	0	0
⊞ ▣ iop1 (iop1_imp_2R...	3252	2996	123	2	1084	3095	157	1344	16	0
⊞ ▣ iop2 (iop2_imp_GME...	3255	2996	123	2	1157	3098	157	1336	16	0
⊞ ▣ iop3 (iop3_imp_17Q...	6389	6247	134	4	2346	6195	194	3103	16	0
⊞ ▣ video (video_imp_1E...	8322	15086	292	20	4157	7757	565	4468	10	18

The cost of handling the interfaces is modest

🄴 XILINX.

The Python programmer's view

XILINX®

PYNQ

Load *base* overlay on PL

Python code on ARM A9:

```
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")
```
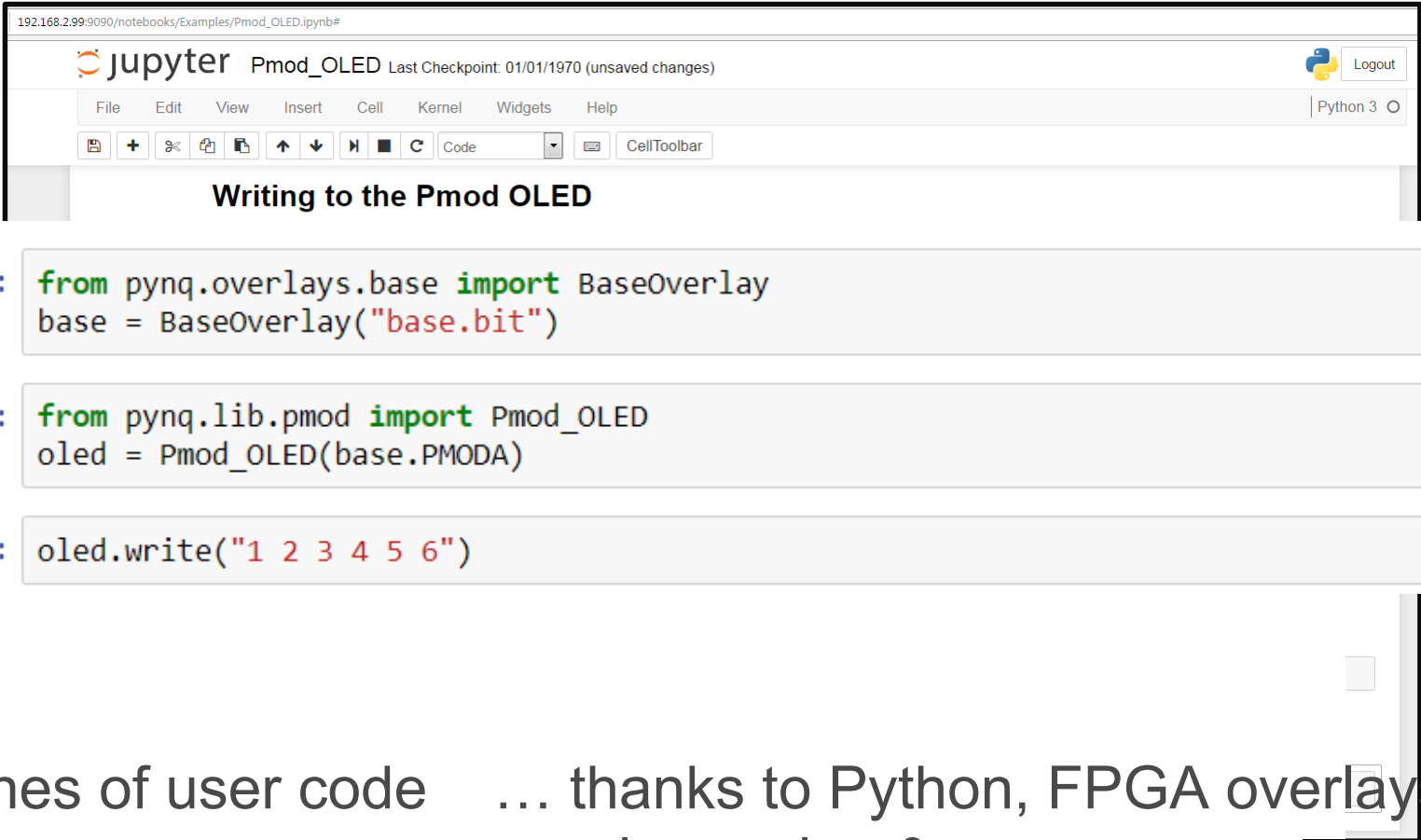
Configure IO processor

```
from pynq.lib.pmod import Pmod_OLED
oled = Pmod_OLED(base.PMODA)
```

MicroBlaze

AXI

RESET

LMB

CPU
Reset

GPIO

ARM
A9

AXI

IOP
Code

Data
Cmds

GPIO

out(0,7)

in(0,7)

CONFIG

PMOD
IO1

Connector J1		
Pin	**Signal**	**Description**
1	CS	SPI Chip Select (Slave Select)
2	SDIN	SPI Data In (MOSI)
3	None	Unused Pin
4	SCLK	SPI Clock
7	D/C	Data/Command Control
8	RES	Power Reset
9	VBATC	V_{BAT} Battery Voltage Control
10	VDDC	V_{DD} Logic Voltage Control
5, 11	GND	Power Supply Ground
6, 12	VCC	Power Supply

TO (0,3)

Timer

PMOD
IO8

```
oled.write("1 2 3 4 5 6")
```

XILINX

Jupyter Notebook

5 lines of user code … thanks to Python, FPGA overlays, abstraction & re-use

Summary

- Overlay Concept

- *base* Overlay

- IOPs

- Using Overlays

- Labs
  - >> Grove temperature sensor
  - >> Pmod OLED
  - >> Grove LEDBar (optional)
  - >> Grove light sensor (optional)
  - >> MicroBlaze programming

XILINX.

Questions?

XILINX

Adaptable.
Intelligent.

XILINX

PYNQ