融入Python生态的Zynq软硬件设计框架

刘伟 Xilinx大学计划合作部

XILINX

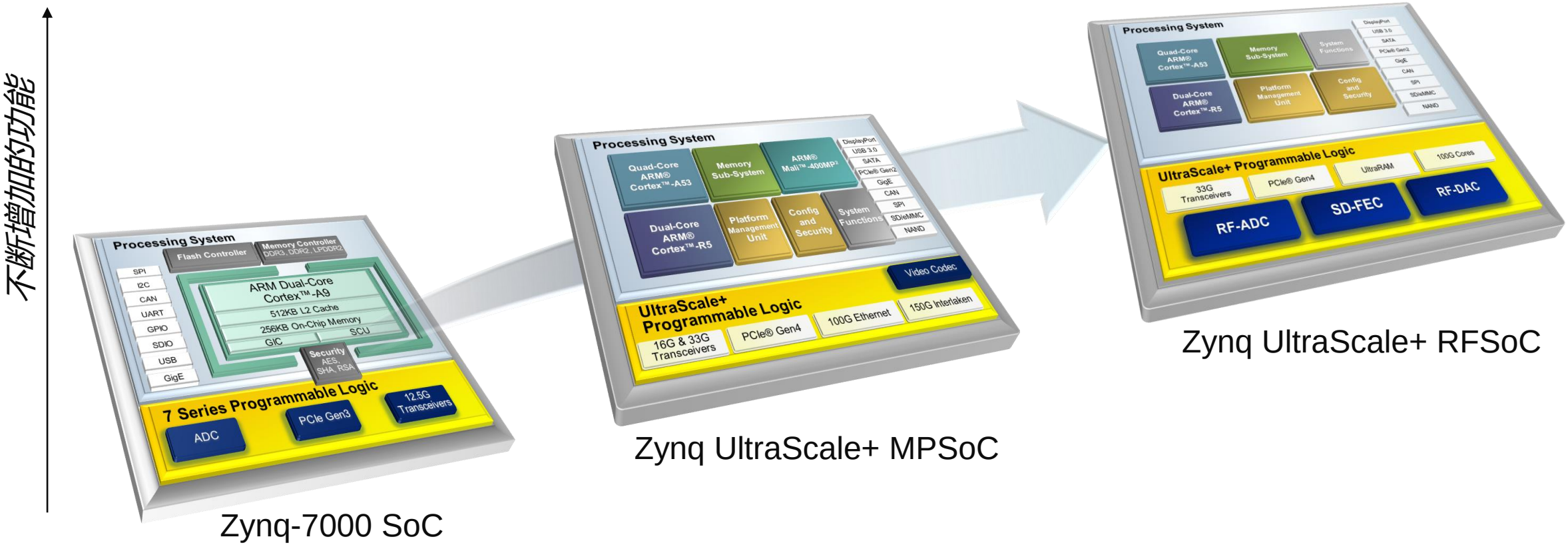
ZYNQ SoC介绍

XILINX

丰富的 SoC 平台系列

面向自主嵌入式系统的异构处理器

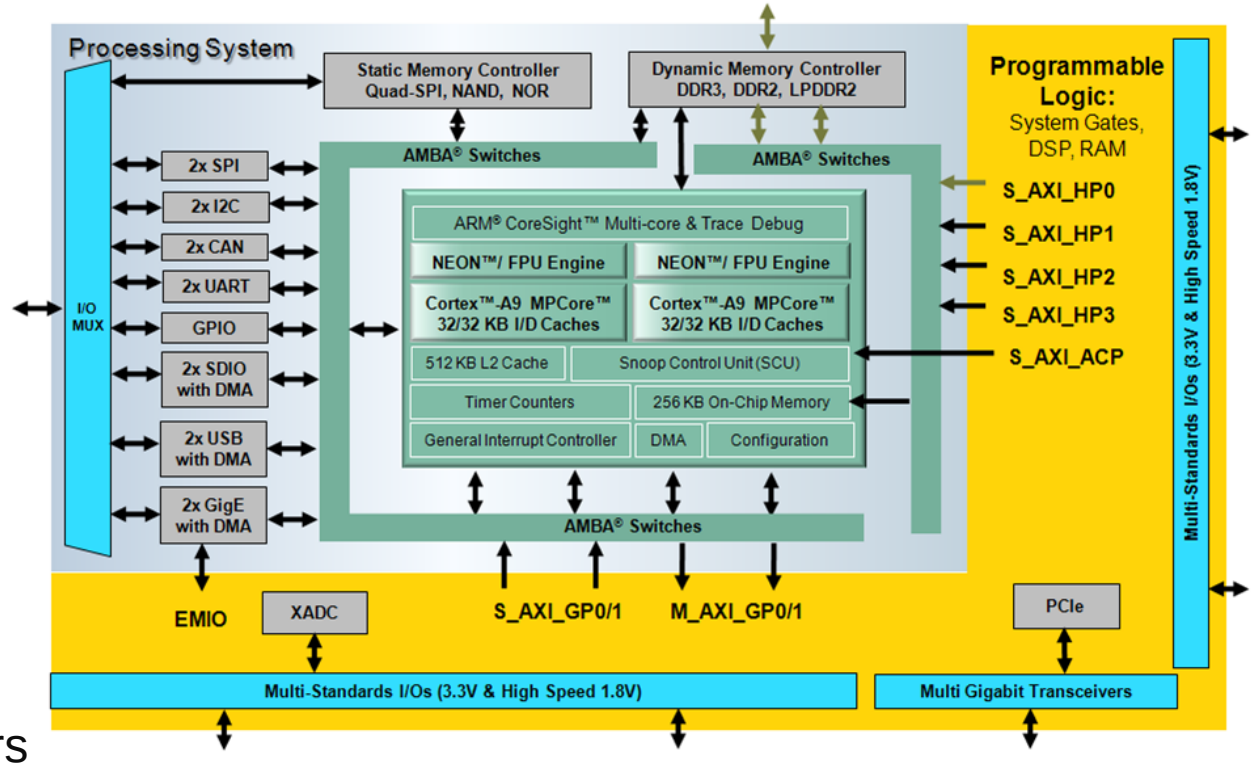
嵌入式视觉　汽车　工业　边缘计算　多级的安全性和保密性



XILINX.

Zynq-7000 Architecture

- **Complete ARM®-based processing system**
  - Application Processor Unit (APU)
    - Dual ARM Cortex™-A9 processors
    - Caches and support blocks
  - Fully integrated memory controllers
  - I/O peripherals
- **Tightly integrated programmable logic**
  - Used to extend the processing system
  - Scalable density and performance
- **Flexible array of I/O**
  - Wide range of external multi-standard I/O
  - High-performance integrated serial transceivers
  - Analog-to-digital converter inputs



XILINX.

> The Zynq-7000 SoC architecture consists of two major sections

- >> PS: Processing system
  - – Single/Dual ARM Cortex-A9 processor based
  - – Multiple peripherals
  - – Hard silicon core
- >> PL: Programmable logic
  - – Shares the same 7-series programmable logic as
    - ▪ Artix™-based devices: Z-7007S, Z-7012S, Z-7014S, Z-7010, Z-7015 and Z-7020
    - ▪ Kintex™-based devices: Z-7030, Z-7035, Z-7045 and Z-7100

Features	Zynq-7000S	Zynq-7000	
Devices	Z-7007S, Z-7012S, Z-7014S	Z-7010, Z-7015, Z-7020	Z-7030, Z-7035, Z-7045, Z-7100
Processor Core	Single-core ARM® Cortex™-A9 MPCore™	Dual-core ARM Cortex-A9 MPCore	
Maximum Frequency	Up to 766MHz	Up to 866 MHz	Up to 1GHz
External Memory Support	DDR3, DDR3L, DDR2, LPDDR2		
Key Peripherals	USB 2.0, Gigabit Ethernet, SD/SDIO		
Dedicated Peripheral Pins	Up to 128	Up to 128	128

ZYNQ SoC开发常用工具链

XILINX

ZYNQ工具链 – 满足各种应用场景

应用程序

Linux

Drivers

Libraries

Xilinx工具链

VIVADO 硬件设计

HLx Editions 硬件IP设计

SDK 嵌入式开发

Petalinux

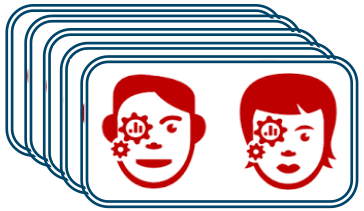
SDSoC Environment

ZYNQ平台

XILINX

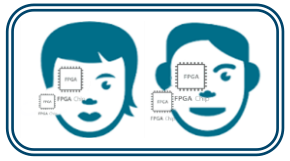
PYNQ软件框架的设计理念

PYNQ Python Productivity for Zynq



Domain Experts

Embedded software Engineers



Hardware Engineers

Targeting the data center artificial intelligence, machine learning, data science

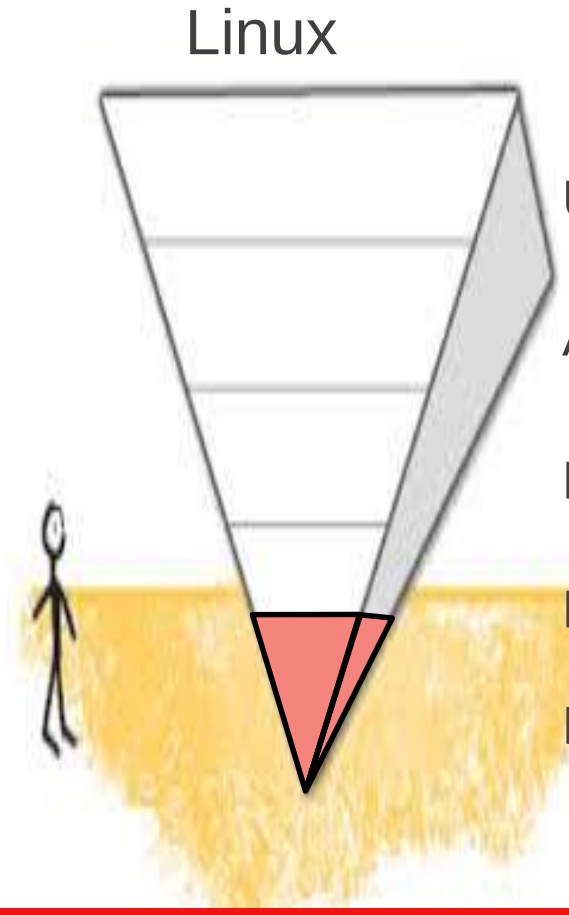
New users are not hardware designers, or embedded systems designers

PYNQ

Enable more people to program Xilinx processing platforms, more productively

XILINX

Hundreds of People maintain the Linux Kernel, Tens of Millions of People use it!

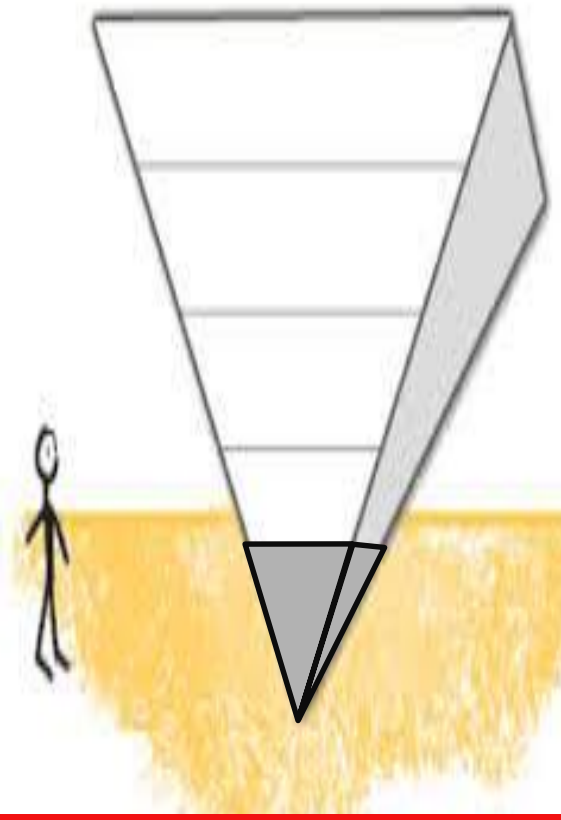


Users:	Tens of millions
Apps Programmers:	Hundreds of thousands
Programmers(embedded):	Tens of thousands
Device driver writers:	Thousands
Kernel developers:	Hundreds

How can we support hundreds of thousands of Zynq programmers with hundreds of bitstream developers?

Productivity Languages & Hardware Overlays

Zynq / Zynq UltraScale+



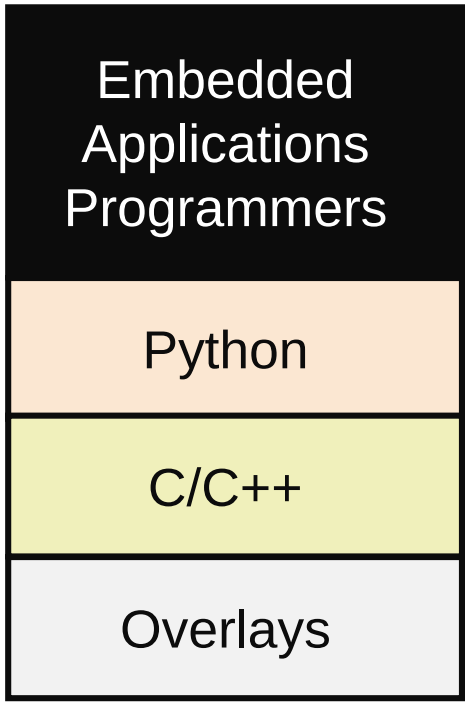
Users:

Apps Programmers:

Programmers(embedded):

Device driver writers:

Kernel developers:



Small group of experts create SoC overlays and C API/drivers

Many more users build applications in C/Python

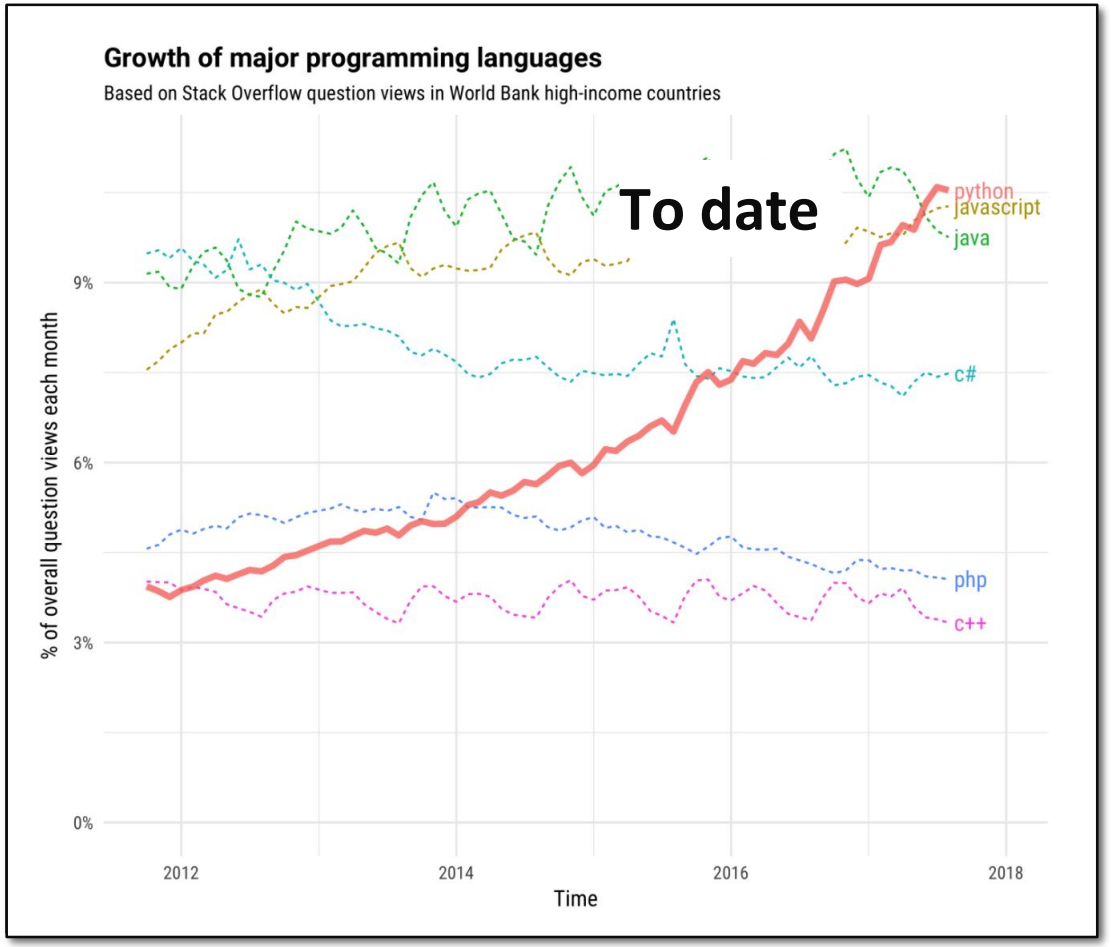
Python is increasingly the Language of Choice

Top Programming Languages,
IEEE Spectrum, July'18

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		98.4
3. C		98.2
4. Java		97.5
5. C#		89.8
6. PHP		85.4
7. R		83.3
8. JavaScript		82.8
9. Go		76.7
10. Assembly		74.5

Python is listed as an embedded language for the first time

https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages



https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Python is the fastest growing language: driven by data science, AI, ML and academia

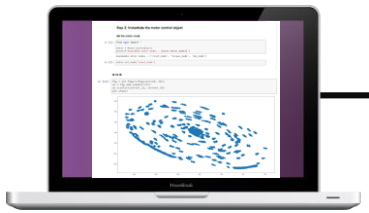
XILINX

PYNQ框架介绍

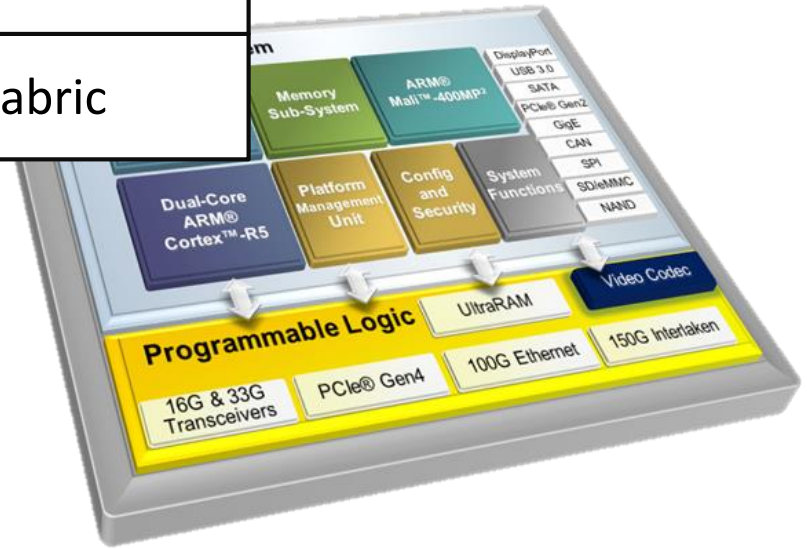
Python productivity for Zynq

Jupyter notebooks/lab, browser-based interface

PYNQ enables Jupyter on Zynq and ZU+



Jupyter web server	
IPython kernel	**Hardware drivers wrapped in Python libraries**
Ubuntu-based Linux	Overlays/designs
PS: ARM	**PL**: Fabric



Optimized for developer productivity

- All the Linux libraries and drivers you expect
- Pre-built SD image
- Ubuntu/Debian ecosystem & community
  - >> 110,000,000 Google hits

XILINX

‘Ubuntu-based’ Linux versus embedded Linux

Ubuntu-based Linux

Optimized for developer productivity

- All the Linux libraries and drivers you expect
- Pre-built SD card image
- Ubuntu/Debian ecosystem & community
  - >>145,000,000 Google hits

3 orders of magnitude difference

Embedded Linux

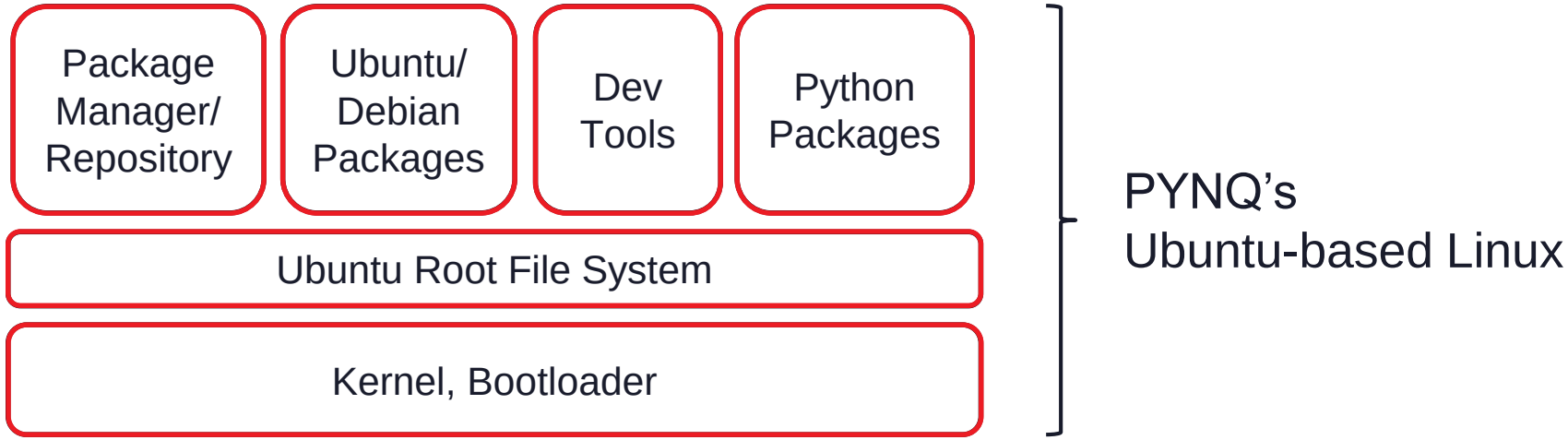
Optimized for deployment efficiency

- Selective Linux libraries and drivers
- Commonly delivered in flash memory on board
- PetaLinux ecosystem:
  - >> 143,000 Google hits

PYNQ's Ubuntu-based Linux

PYNQ uses Ubuntu's:
- Root file system (RFS)
- Package manager (*apt-get*)
- Repositories

PYNQ bundles :
- Development tools
  - Cross-compilers
- Latest Python packages

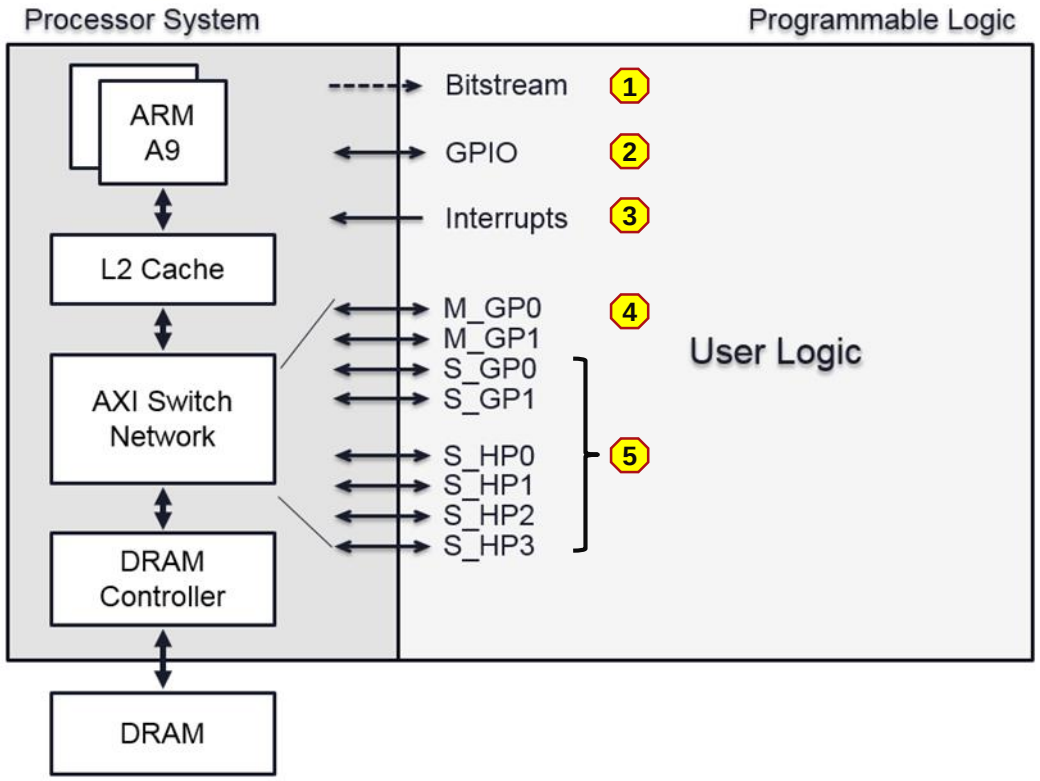


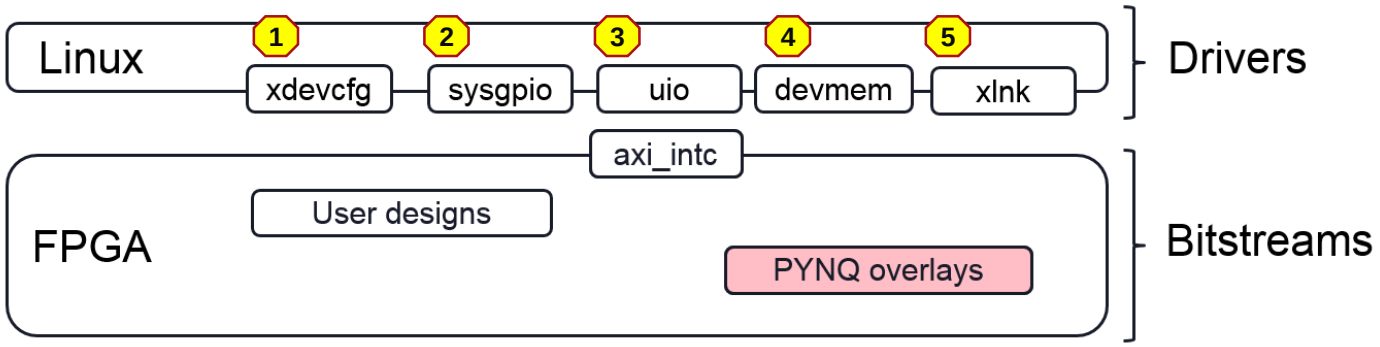
PYNQ uses the PetaLinux build flow and board support package:
- Access to all Xilinx kernel patches
- Works with any Xilinx supported board
- Configured with additional drivers for PS-PL interfaces

PYNQ provides Linux Drivers for PS-PL Interfaces …

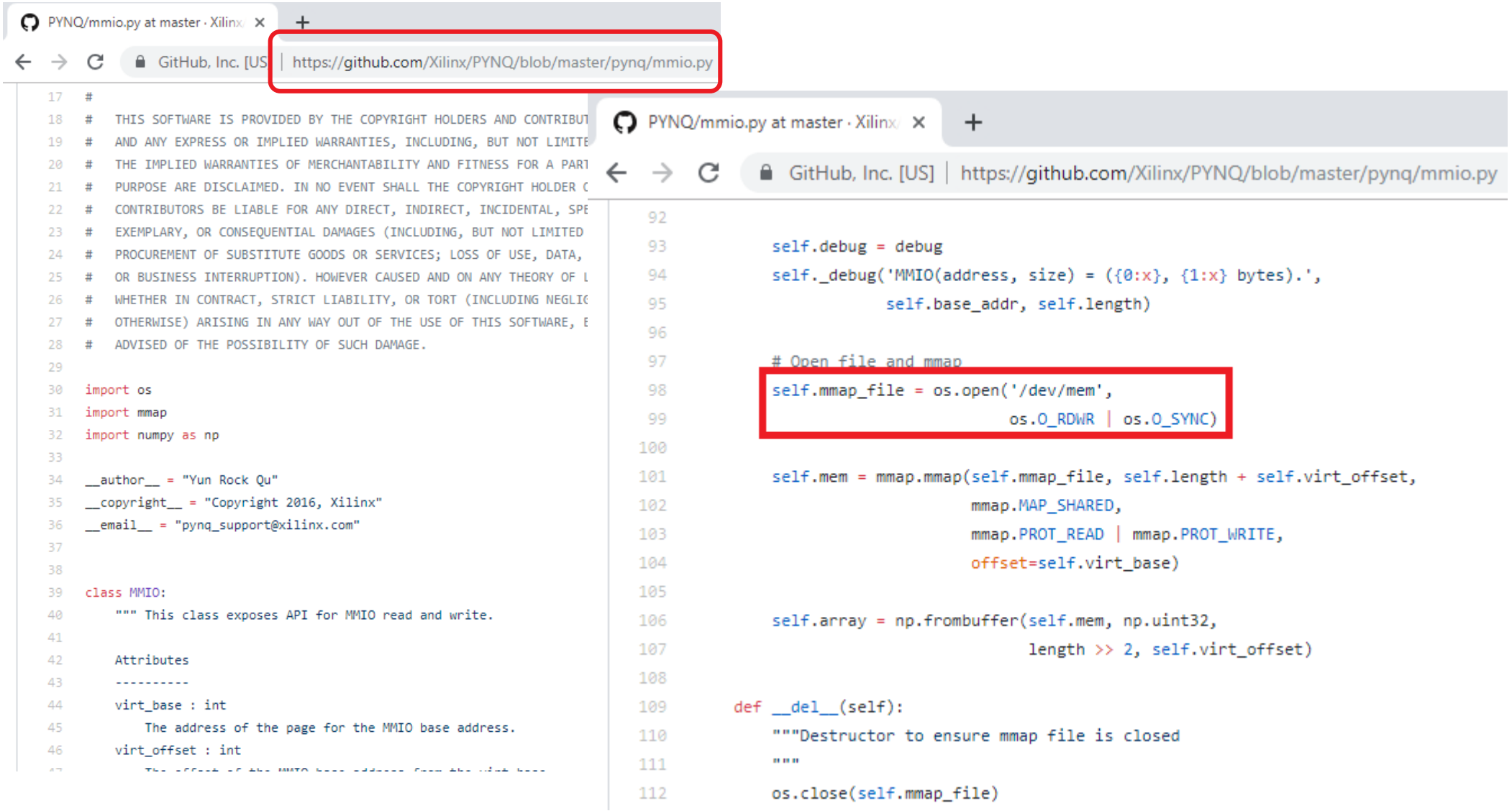
wrapped in Python Libraries

Zynq

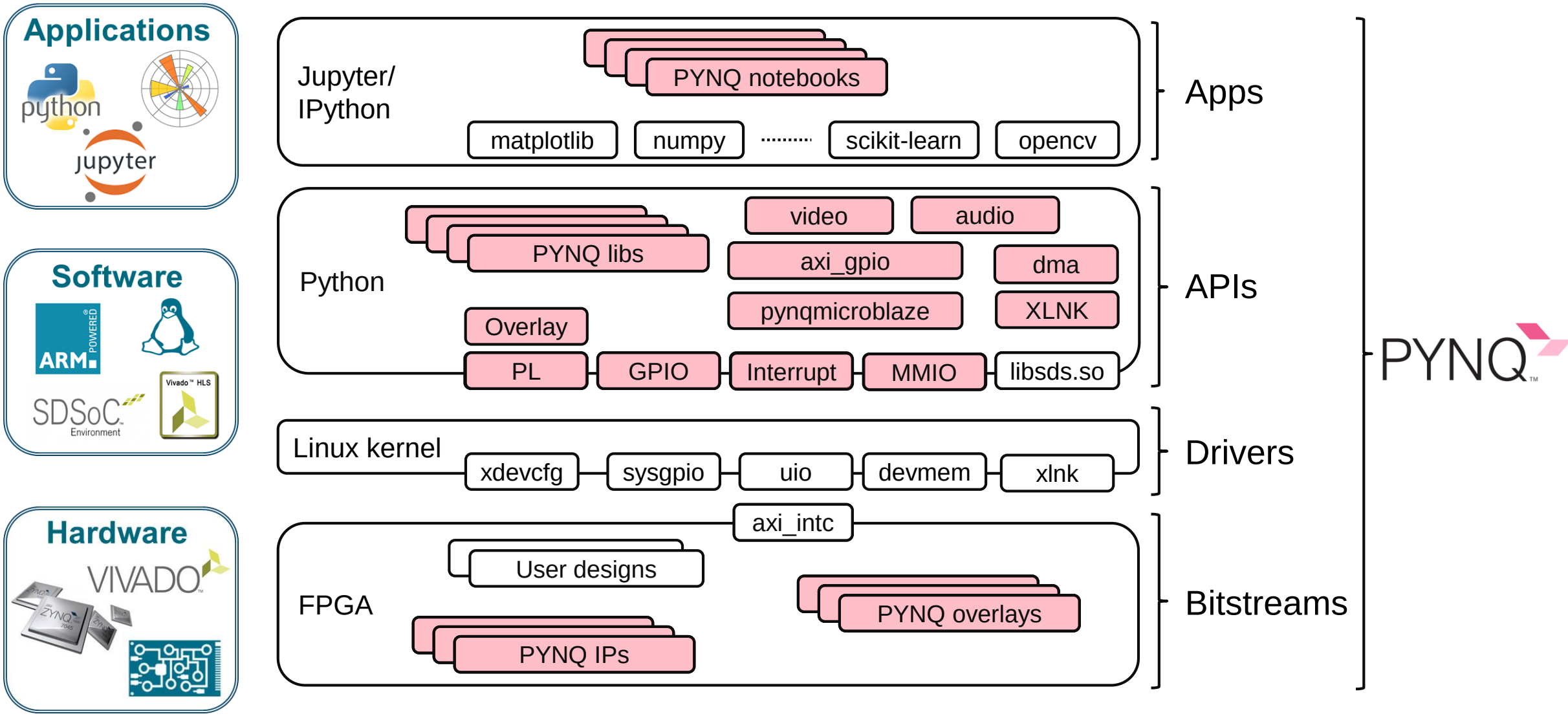




How MMIO works



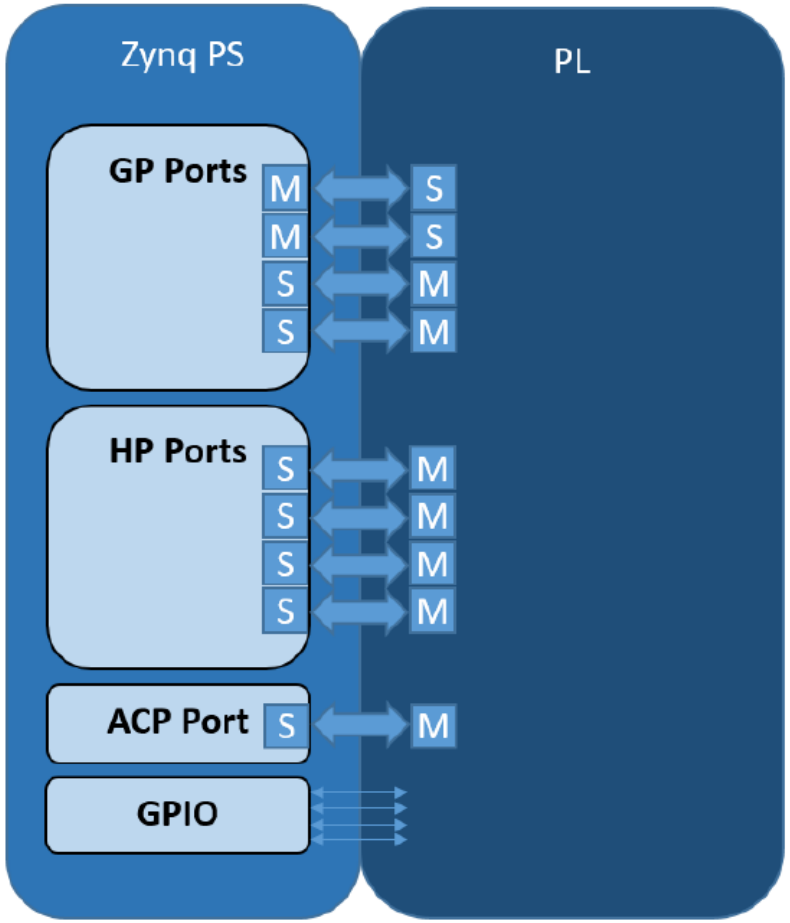
PYNQ is a Framework
- The key of productivity is the Unified



XILINX

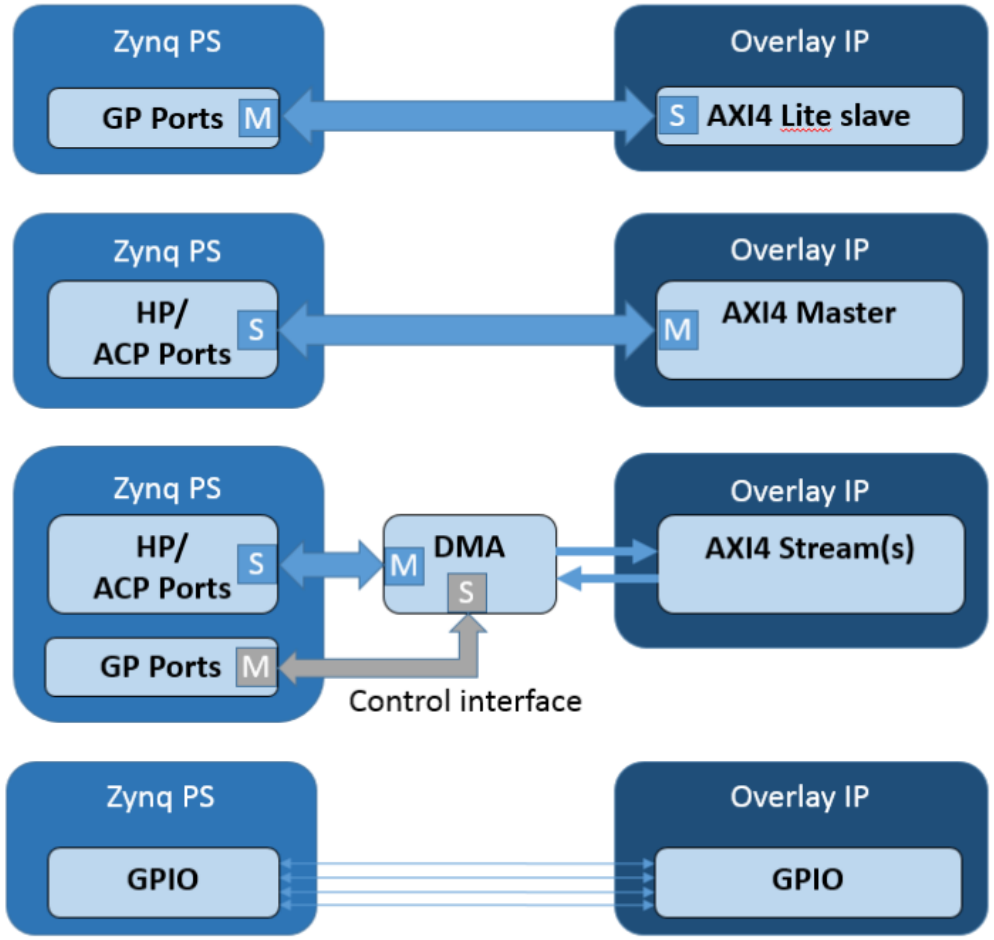
ZYNQ PL – PS 接口介绍

ZYNQ PL与PS接口



- **AXI General Purpose ports**
  - 2x Master (32-bit)
  - 2x Slave (32-bit)
- **AXI High Performance**
  - 4x Slave (64-bit)
    - 1K FIFOs
- **ACP**
  - 1x Slave (64-bit)
  - Cache access
- **GPIO (EMIO)**
  - Wires (64)

常用IP接口

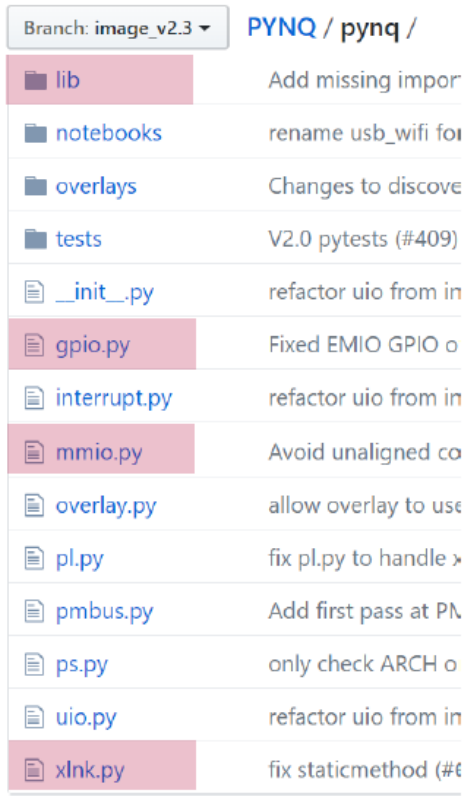


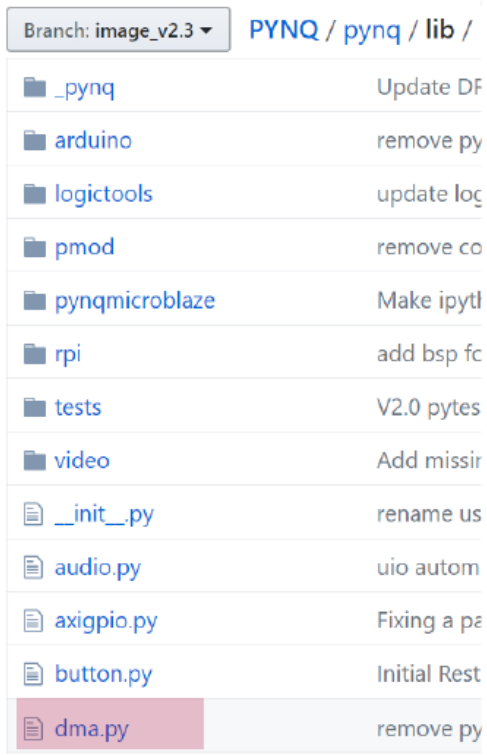
- **AXI (Lite) Slave IP**
  - General Purpose ports
  - Typically lower performance IP
- **AXI Master**
  - AXI HP/ACP
  - Typically higher performance IP
- **AXI Stream**
  - Via DMA
  - HP/ACP ports for data path
  - GP slave for control
- **GPIO**
  - Control type data

PYNQ接口类介绍

PYNQ - PL与PS接口类

- **MMIO (pynq.mmio)**
  - Memory Mapped Input Output
  - Register based memory mapped transactions
- **Xlnk (pynq.xlnk)**
  - Memory allocation (used in DMA or for AXI Master peripheral)
- **DMA (pynq.lib.dma)**
  - Direct Memory Access
  - Offload memory transfers from main CPU
- **GPIO (pynq.gpio)**
  - Read/Write GPIO wires





XILINX.

PYNQ - MMIO类

- **Import MMIO**
- **Define memory mapped region**
  - BASE_ADDRESS: starting location
  - ARRAY_SIZE: length of accessible memory (optional, default 4 bytes)
- **Read and Write 32-bit values**
  - ADDRESS_OFFSET: Offset from BASE_ADDRESS, should be multiples of 4
  - Need to ensure the memory can be read/written

```
from pynq import MMIO

BASE_ADDRESS = 0x40000000
ARRAY_SIZE = 1024

mmio = MMIO(BASE_ADDRESS, ARRAY_SIZE)

data = 0x12345678
ADDRESS_OFFSET = 0x10

mmio.write(ADDRESS_OFFSET, data)

result = mmio.read(ADDRESS_OFFSET)

print(hex(result))
> 0x12345678
```

PYNQ - Xlnk类

- **Memory managed by Linux**
  - Virtual
- **Memory must be allocated before PL Master can access it**
  - PL needs the physical address of a memory buffer
- **Xlnk can allocate (contiguous) memory buffers (using NumPy)**
  - Maps virtual and physical addresses
  - Contiguous memory is more efficient/allows simpler DMA logic
  - Array can be specified as NumPy data type, and size/shape
- **Once buffer is allocated a DMA can be used to transfer data between PS/PL**
- **DMA class uses Xlnk for memory allocation**

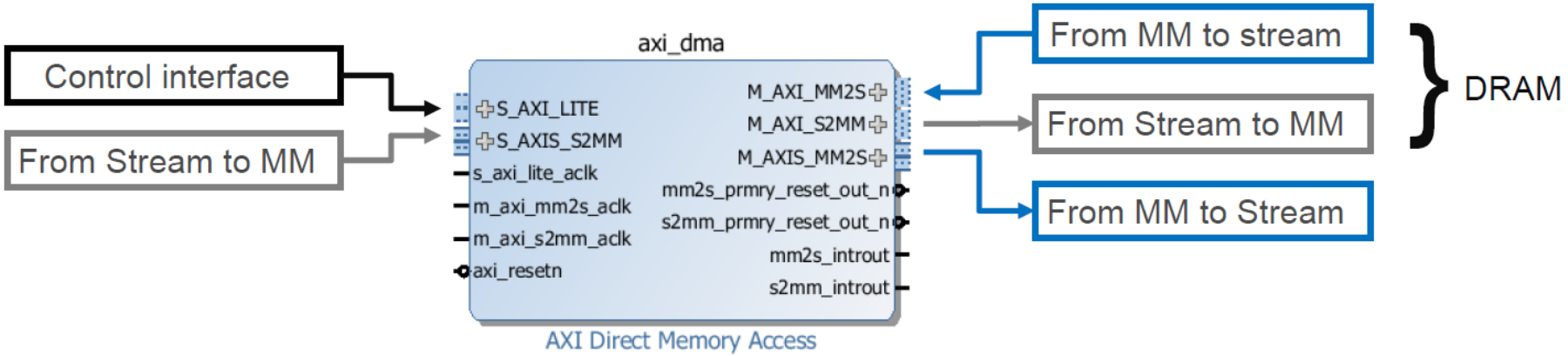
PYNQ - Xlnk类案例

- **Import Xlnk**
- **Allocate contiguous buffer**
  - cma_array()
  - Returns Linux virtual address
- **Get Physical Address**
  - `.physical_address`
  - Can be used by PL to access DDR (Linux) memory
- **Read/write buffer**

```
MEMORY_SIZE = 10
from pynq import Xlnk
import numpy as np
xlnk = Xlnk()

input_buffer = xlnk.cma_array(shape=(10,),
dtype=np.float32)
phy_addr = input_buffer.physical_address

for i in range(10):
    input_buffer[i] = i
print(input_buffer)
```

PYNQ - AXI DMA IP

- AXI Lite control interface (AXI GP port)
- Memory mapped interface (AXI interface, HP/ACP ports)
- AXI Stream interface (AXI stream accelerator)
- Transfer between streams and memory mapped locations
  - Paths from PL to DRAM and DRAM to PL
  - Memory Mapped to Stream (MM2S)/Stream to Memory Mapped (S2MM)



PYNQ - AXI DMA IP

- **Direct memory access**
  - Transfer data between memories directly
    - PS - PL
  - Bypasses CPU
    - Doesn't waste CPU cycles on data transfer
  - Speed up memory transfers with burst transactions
- **Xilinx *AXI Direct Memory Access* IP block supported in PYNQ**
  - Read and Write Paths from PL to DDR and DDR to PL
  - Memory Mapped to Stream
  - Stream to Memory Mapped
- **Needs to stream to/from an allocated memory buffer**
  - PYNQ DMA class inherits from xlnk for memory allocation

PYNQ - DMA类使用案例

- Setup DMAs
- Allocate memory buffers
- Start DMA transactions

Create DMA instances

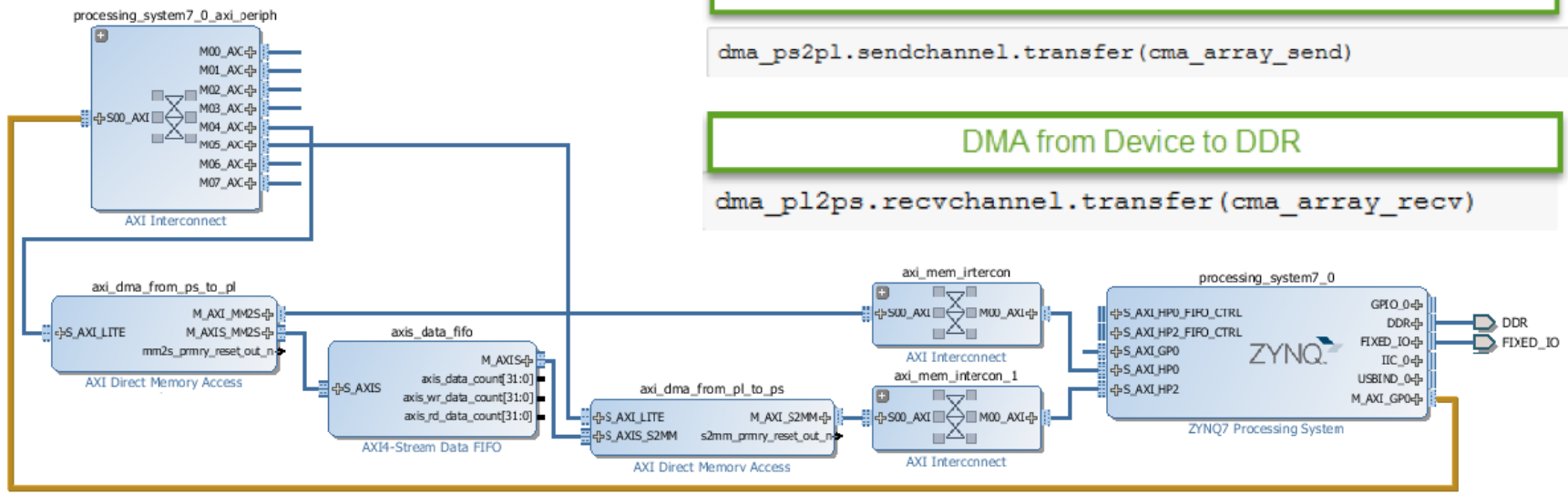
```
from pynq.lib import DMA

dma_ps2pl_description = overlay.ip_dict['axi_dma_from_ps_to_pl']
dma_ps2pl = DMA(dma_ps2pl_description)

dma_pl2ps_description = overlay.ip_dict['axi_dma_from_pl_to_ps']
dma_pl2ps = DMA(dma_pl2ps_description)
```

DMA from DDR to Device

```
dma_ps2pl.sendchannel.transfer(cma_array_send)
```

DMA from Device to DDR

```
dma_pl2ps.recvchannel.transfer(cma_array_recv)
```



XILINX.

PYNQ - GPIO类

- **Up to 64 GPIO wires from PS**
  - Tri-state
- **Accessed from Linux**
- **Setup GPIO instance**
- **Map to GPIO pin**
  - Translated pin numbers to Linux GPIO number
  - get_gpio_pin()
- **Read/Write**
- **Most appropriate for simple control**
  - Set, reset, start, done …

```
from pynq import GPIO

btn_gpio = GPIO.get_gpio_pin(0)
led_gpio = GPIO.get_gpio_pin(1)

ps_btn = GPIO(btn_gpio, 'in')
ps_led = GPIO(led_gpio, 'out')

ps_btn.read()

ps_led.write()
```

PYNQ封装类

- Standard Python code
- Interface classes used
  - MMIO, Xlnk, DMA, GPIO
- Check PYNQ repository for examples

```
class LED(object):
    """This class controls the onboard leds.

    Attributes
    ----------
    _impl : object
        An object with appropriate LED methods
    """
    def __init__(self, device):
        """Create a new LED object.
        Parameters
        ----------
        device : object
            An object with appropriate LED methods:
            on, off, toggle and len
        """
        methods = ['on', 'off', 'toggle']
        if all(m in dir(device) for m in methods):
            self._impl = device
        else:
            raise TypeError("'device' must contain LED methods: " +
                            str(methods))
    def toggle(self):
        """Toggle led on/off."""
        self._impl.toggle()
    def on(self):
        """Turn on led."""
         self._impl.on()
     def off(self):
         """Turn off led."""
         self._impl.off()
```

PYNQ Overlay API

Python Overlay API

- **PYNQ Overlay class supports basic overlay functionality**
  - Requires overlay Tcl
  - Allows download, and overlay discovery (ip_dict)
  - Assigns default drivers to IP
    - Read() and write() access to IP address space

```
from pynq import Overlay
overlay = Overlay("pynqtutorial.bit")
```

```
help(overlay)
```

```
class Overlay(pynq.pl.Bitstream)
 |  Default documentation for overlay pynqtutorial.bit.
 |  attributes are available on this overlay:
 |
 |  IP Blocks
 |  ----------
 |  axi_dma_from_pl_to_ps : pynq.lib.dma.DMA
 |  axi_dma_from_ps_to_pl : pynq.lib.dma.DMA
 |  btns_gpio             : pynq.lib.axigpio.AxiGPIO
 |  mb_bram_ctrl_1        : pynq.overlay.DefaultIP
 |  mb_bram_ctrl_2        : pynq.overlay.DefaultIP
 |  rgbleds_gpio          : pynq.lib.axigpio.AxiGPIO
 |  swsleds_gpio          : pynq.lib.axigpio.AxiGPIO
 |  system_interrupts     : pynq.overlay.DefaultIP
```

```
overlay.rgbleds_gpio.write(0, 3)
overlay.swsleds_gpio.read()
```

定制化Overlay API

- **Custom Overlay API**
  - Allows automatic assignment of custom drivers
    - Overwrite default drivers
  - No need to import additional drivers
  - Easier for software developers to use overlays
- **Overlay “attributes” available from overlay instance**
  - E.g. base.PMODA
- **help()**
  - Discover information about the overlay

```
from pynq.overlays.base import
BaseOverlay

base = BaseOverlay("base.bit")
```

```
help(base)
```

```
Help on BaseOverlay in module pynq.overlays.base.base ob

class BaseOverlay(pynq.overlay.Overlay)
 |  The Base overlay for the Pynq-Z1
 |
 |  This overlay is designed to interact with all of the
 |  and external interfaces of the Pynq-Z1 board. It exp
 |  attributes:
 |
 |  Attributes
 |  ----------
 |  iop1 : IOP
 |       IO processor connected to the PMODA interface
 |  iop2 : IOP
 |       IO processor connected to the PMODB interface
 |  iop3 : IOP
 |       IO processor connected to the Arduino/ChipKit i
 |  trace_pmoda : pynq.logictools.TraceAnalyzer
 |      Trace analyzer block on PMODA interface, control
```


XILINX.

Overlay API示例

- **Custom Python wrapper provided with overlay**
- **Inherits from pynq.Overlay class**
- **Define custom drivers, interface types, IOP types, etc.**
- **If no driver is specified, a *DefaultIP* driver is assigned**
  - Read(), write() to IP address space (base on MMIO)

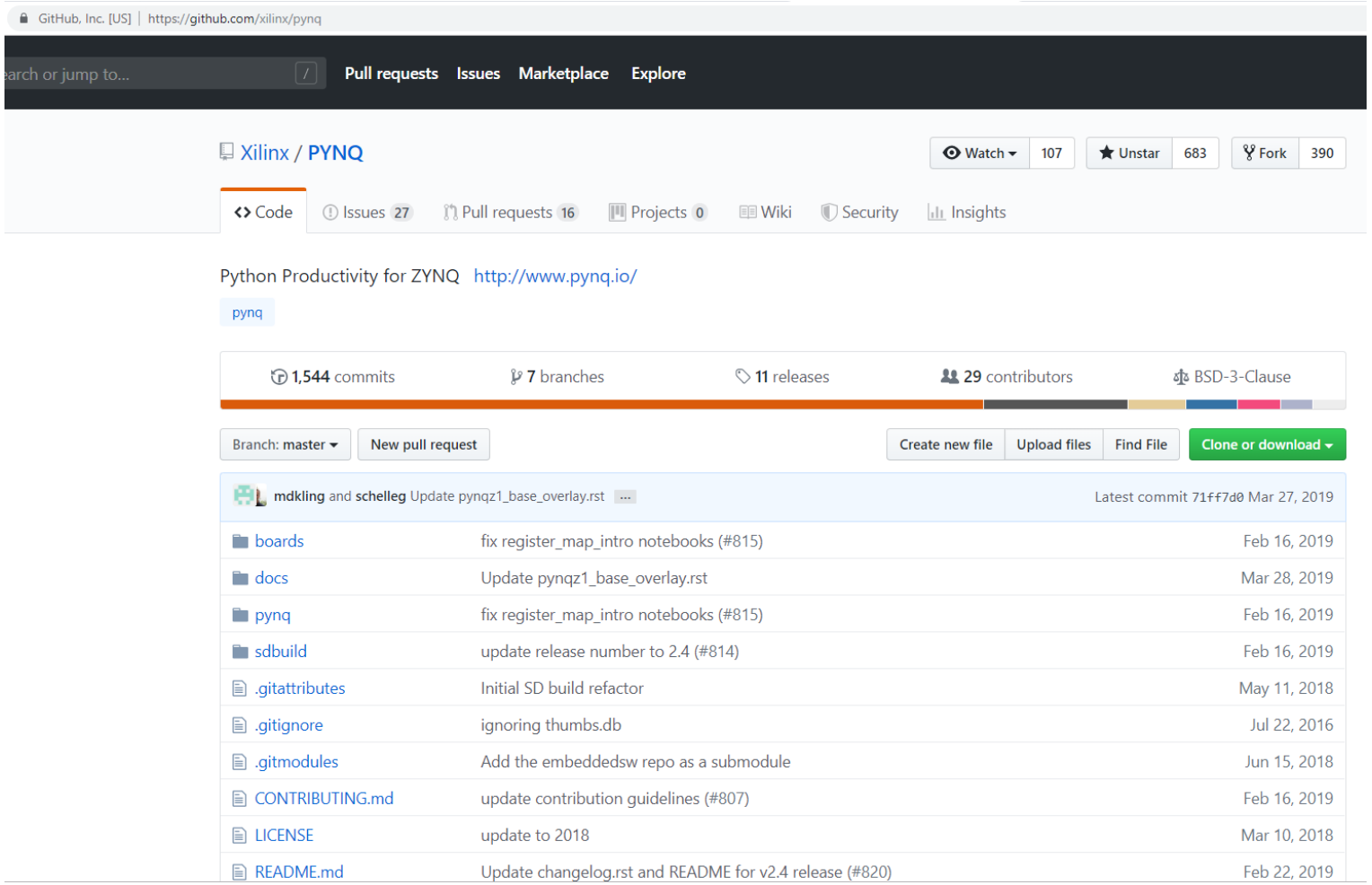
```
class BaseOverlay(pynq.Overlay):

    self.iop_pmoda.mbtype = "Pmod"
    self.iop_arduino.mbtype = "Arduino"
    self.iop_rpi.mbtype = "Rpi"


    self.leds =
         self.leds_gpio.channel1
    self.switches =
         self.switches_gpio.channel1

    self.leds.setlength(4)
    self.switches.setlength(2)

    self.leds.setdirection("out")
    self.switches.setdirection("in")
```

xilinx\pynq\boards\Pynq-Z2\base\base.py

PYNQ Python Package

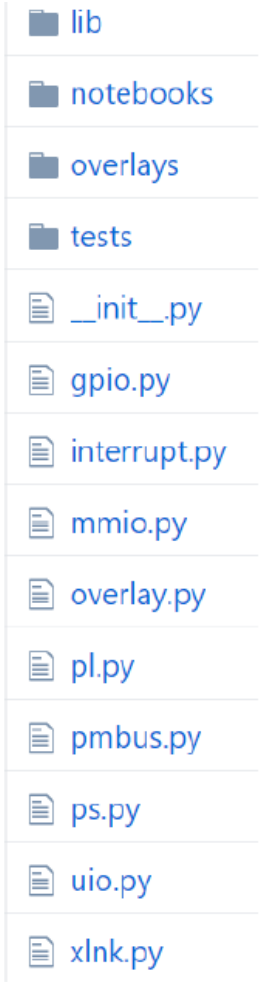
PYNQ源代码

> https://github.com/xilinx/pynq



PYNQ Python Package

- **pynq/**
  - Interface classes
  - Overlays
  - Tests
- **pynq/lib**
  - Pmod, Grove, Arduino, RPi driver classes
  - *logictools* class
  - Video, Audio, DMA, Tracebuffer, WiFi
- **pynq/overlays**
  - Application domains
- **pynq/tests**
  - Pytests for basic modules



添加自定义模块

- **__init__.py**
  - Exists in each directory
  - Include filename and class to be imported
- **Hierarchical**
  - Top level can include lower level directory

```
from .gpio import GPIO
from .mmio import MMIO
from .uio import UioController
from .ps import Register
from .ps import Clocks
from .pl import PL
from .pl import PL_SERVER_FILE
from .pl import Bitstream
from .ps import Register
from .ps import Clocks
from .interrupt import Interrupt
from .xlnk import Xlnk
from .overlay import Overlay
from .overlay import DefaultHierarchy
from .overlay import DefaultIP
from .pmbus import get_rails
from .pmbus import DataRecorder
from . import lib
```

pynq__init__.py

```
from .pynqmicroblaze import PynqMicroblaze
from .pynqmicroblaze import MicroblazeRPC
from .pynqmicroblaze import MicroblazeLibrary
from .axigpio import AxiGPIO
from .dma import DMA
from .led import LED
from .rgbled import RGBLED
from .switch import Switch
from .button import Button
from .iic import AxiIIC
from .wifi import Wifi
from .rpi import Rpi
from .arduino import Arduino
from .arduino import Arduino_DevMode
from .arduino import Arduino_IO
from .arduino import Arduino_Analog
from .arduino import Arduino_LCD18
from .pmod import Pmod
from .pmod import Pmod_DevMode
from .pmod import Pmod_ADC
from .pmod import Pmod_DAC
```

pynq\lib\pmod__init__.py

PYNQ开发初探

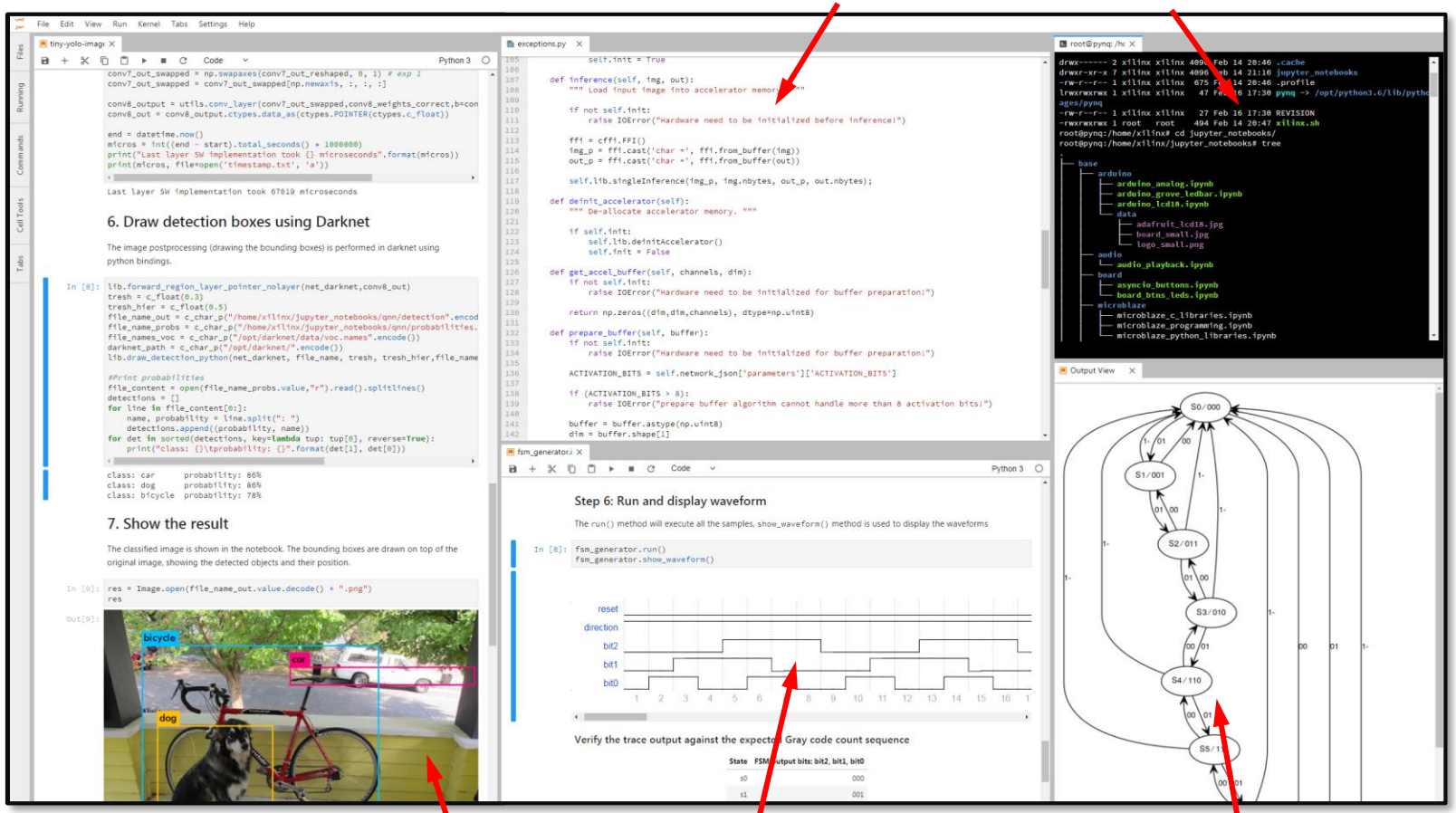
XILINX

Jupyter Notebooks to JupyterLab IDE

2017 ACM
Software System Award

Code editor

Terminal



Jupyter notebooks

Visualization

Jupyter … Julia, Python, R

Default engine of data science

Taught to 1,000+ Berkeley students every semester

2+ million notebooks on GitHub

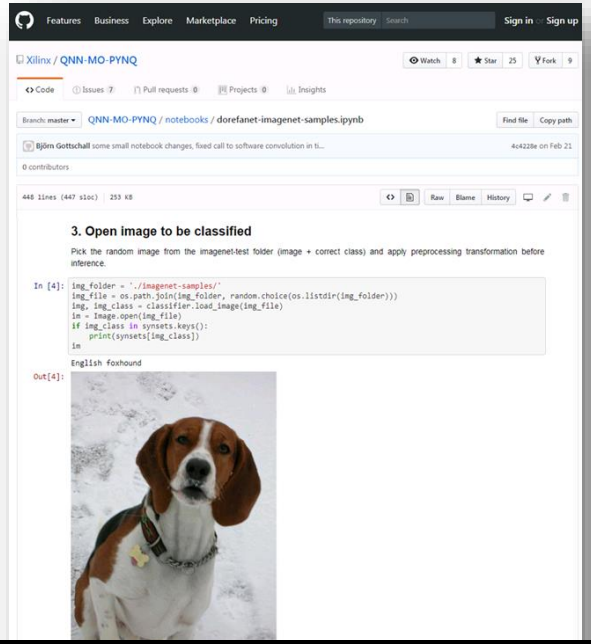
Next-gen browser IDE

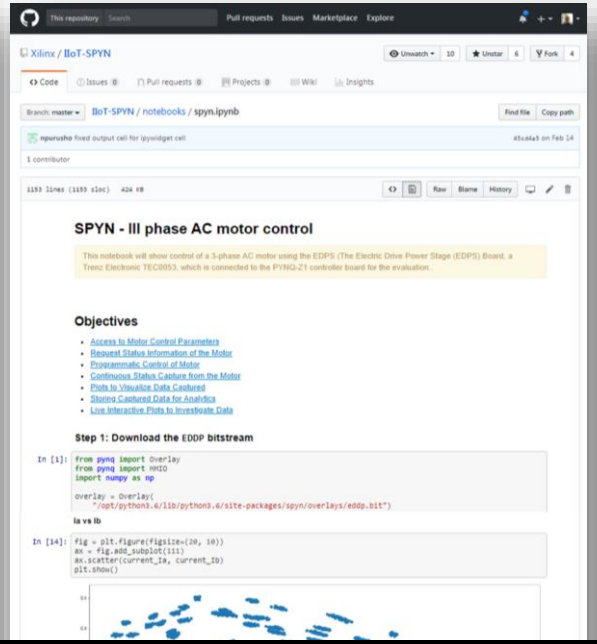
Includes Jupyter Notebooks

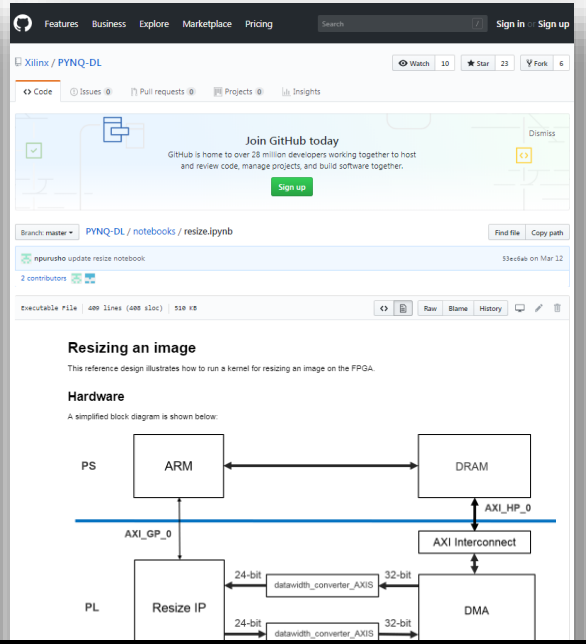
XILINX

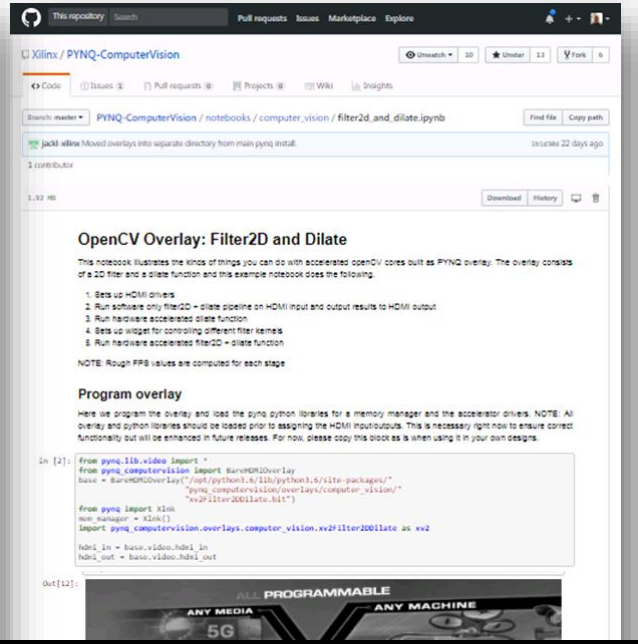
Software-style packaging & distribution of designs

Enabled by new *hybrid libraries*





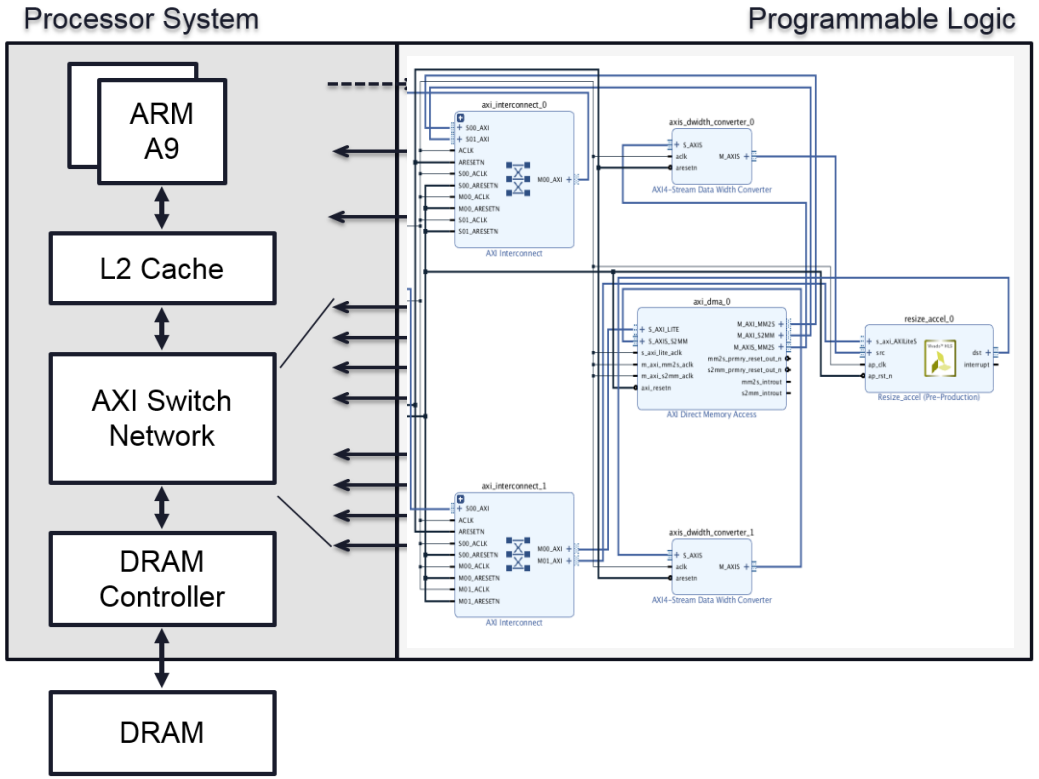




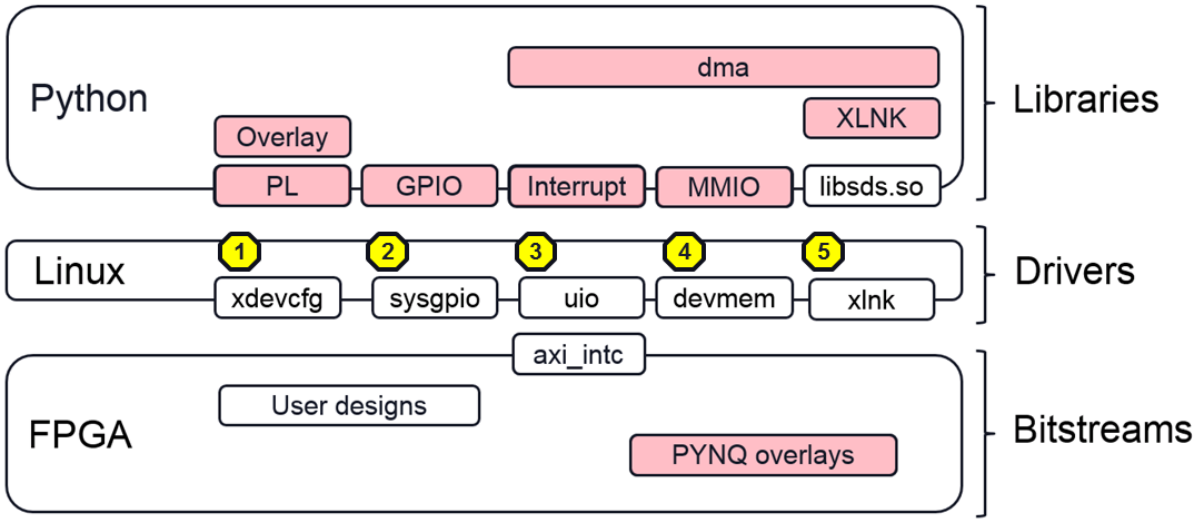

```
Download a design from GitHub with a single Python command:

pip install git+https://github.com/Xilinx/pynqDL.git
```


XILINX.

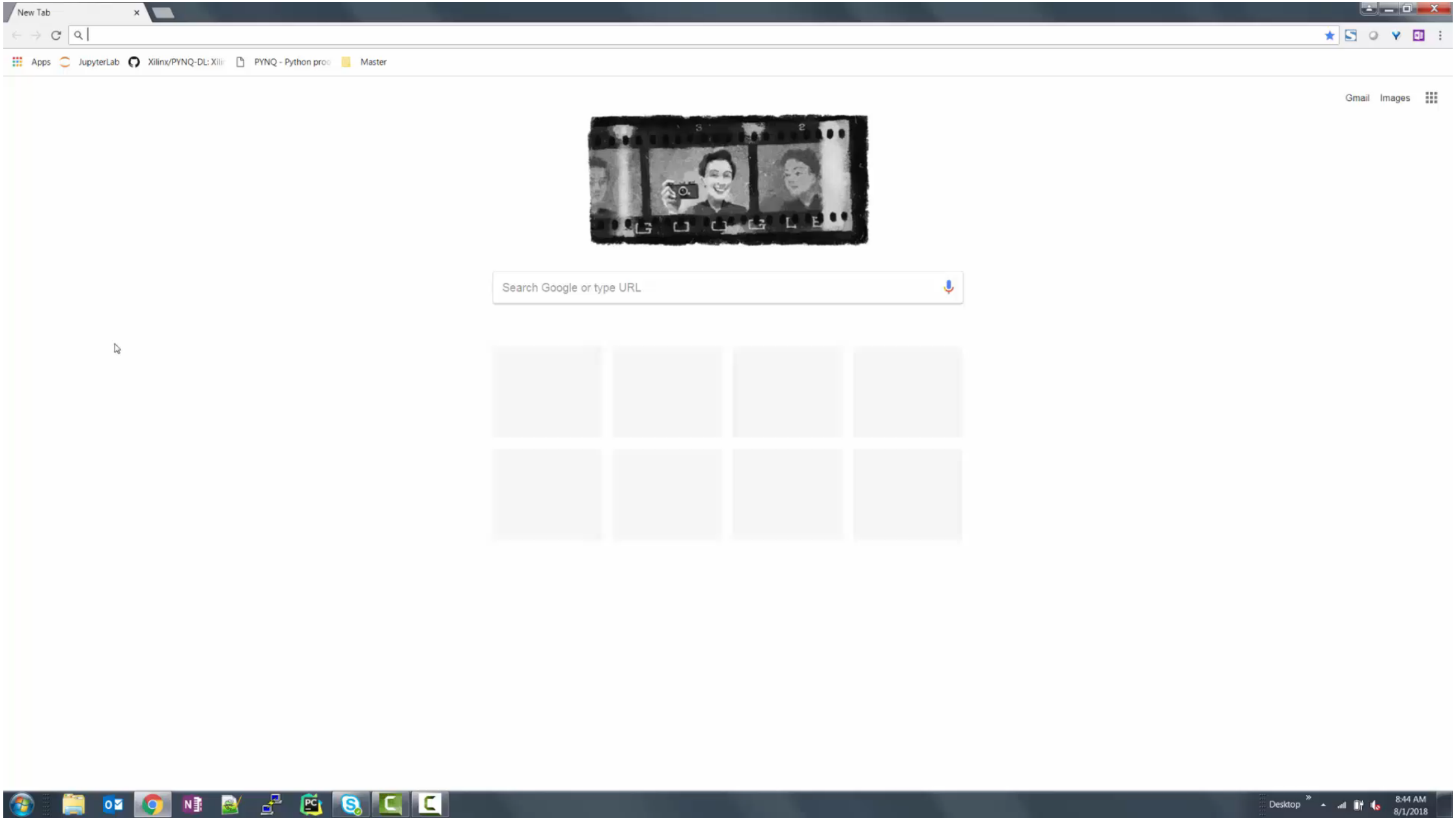
Loading a design into Zynq using PYNQ




```
from pynq import Overlay
resizer = Overlay('./resizer.bit')
```



PYNQ automatically configures many design parameters based on data parsed from hybrid library



How Python helps, really a lot..

Ecosystem Advantage: there's a Library for that…

Standard Python comes with comprehensive libraries for common operations (web, regex, os, etc)
In addition to this 'batteries included' strategy, there is a massive external ecosystem …

CPython is written in C … and most popular C/C++ frameworks have Python libraries

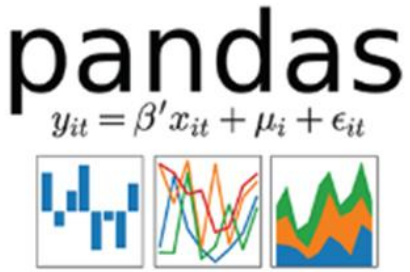
Base Python libraries used for all Use Case

NumPy

A Matlab™ like framework for ***numerical computing***.

Matplotlib

2D plotting library for ***static and interactive data visualizations***



pandas

Data wrangling for easy-to-use data ***ingestion, transformation, and export functions***

Acquire, Transform, Organize, Display

What is NumPy

- **NumPy is the fundamental package for scientific computing with Python. It contains among other things:**
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform

```
In [1]: import numpy as np

        #Array multiplication...
        a = np.array([2,3,4,5])  # in Matlab a = [2 3 4 5]

        b = np.array([6,7,8,9])  # in Matlab b = [6 7 8 9]

        c = a * b  # as in Matlab
        c
Out[1]: array([12, 21, 32, 45])

In [2]: #Matrix multiplication...
        m1 = np.array([[1, 2, 3, 4.0], [5.1, 6.2, 7.3, 8.4], [9, 10, 11, 12]])
        m2 = np.array([[1.3, 2.5, 3.7, 4.9], [5.21, 6.22, 7.33, 8.44], [12, 10, 11, 10]])

        m3 = m1 * m2
        m3
Out[2]: array([[  1.3  ,   5.   ,  11.1  ,  19.6  ],
               [ 26.571,  38.564,  53.509,  70.896],
               [108.   , 100.   , 121.   , 120.   ]])
```

- **NumPy can also be used as:**
  - An efficient multi-dimensional container of generic data.
  - Arbitrary data-types can be defined.
  - This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

How Numpy interacts with Programmable Logic?

NumPy

Array, Matrix

```
In [7]: import numpy as np
        import pynq

        def get_pynq_buffer(shape, dtype):
            """ Simple function to call PYNQ's memory allocator with numpy attributes

            """
            return pynq.Xlnk().cma_array(shape, dtype)
```

Virtual memory

PYNQ

Xlnk

Physical memory

Contiguous Array in DDR Memory
Shared Object

Physical memory

Hard Real Time
Offload CPU and OS
Dedicated “Accelerator”

A X I H P

VIVADO HLx Editions

SDSoC Environment

PL

XILINX

Provide a Numpy Array (or Arrays) with collected data Or Pandas structure

This notebook makes the Exploratory Data Analysis:

We aquire the Data

We display the Data

We process the Data

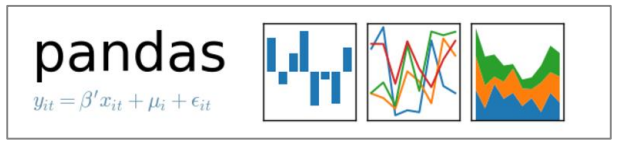
```
In [3]: PLdata #it is the Numpy Array
```

Out[3]:

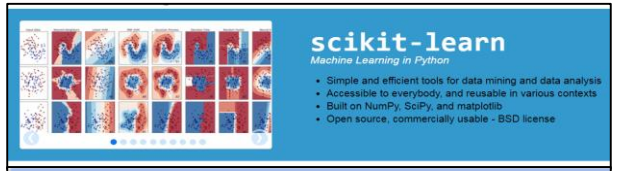
	Mot-ID	cycle	Setup1	Setup2	Setup3	Ia_mA	Ib_mA	Total_Current	s4	s5	s6	s7	s8	s9	Acc	Freq	ttf
0	1	1	-0.0007	-0.0004	100	518.67	641.82	1589.70	1400.60	14.62	21.61	554.36	2388.06	9046.19	1.3	47.47	191
1	1	2	0.0019	-0.0003	100	518.67	642.15	1591.82	1403.14	14.62	21.61	553.75	2388.04	9044.07	1.3	47.49	190
2	1	3	-0.0043	0.0003	100	518.67	642.35	1587.99	1404.20	14.62	21.61	554.26	2388.08	9052.94	1.3	47.27	189
3	1	4	0.0007	0.0000	100	518.67	642.35	1582.79	1401.87	14.62	21.61	554.45	2388.11	9049.48	1.3	47.13	188
4	1	5	-0.0019	-0.0002	100	518.67	642.37	1582.85	1406.22	14.62	21.61	554.00	2388.06	9055.15	1.3	47.28	187
5	1	6	-0.0043	-0.0001	100	518.67	642.10	1584.47	1398.37	14.62	21.61	554.67	2388.02	9049.68	1.3	47.16	186
6	1	7	0.0010	0.0001	100	518.67	642.48	1592.32	1397.77	14.62	21.61	554.34	2388.02	9059.13	1.3	47.36	185
7	1	8	-0.0034	0.0003	100	518.67	642.56	1582.96	1400.97	14.62	21.61	553.85	2388.00	9040.80	1.3	47.24	184
8	1	9	0.0008	0.0001	100	518.67	642.12	1590.98	1394.80	14.62	21.61	553.69	2388.05	9046.46	1.3	47.29	183
9	1	10	-0.0033	0.0001	100	518.67	641.71	1591.24	1400.46	14.62	21.61	553.59	2388.05	9051.70	1.3	47.03	182
10	1	11	0.0018	-0.0003	100	518.67	642.28	1581.75	1400.64	14.62	21.61	554.54	2388.05	9049.61	1.3	47.15	181
11	1	12	0.0016	0.0002	100	518.67	642.06	1583.41	1400.15	14.62	21.61	554.52	2388.09	9049.37	1.3	47.18	180
12	1	13	-0.0019	0.0004	100	518.67	643.07	1582.19	1400.83	14.62	21.61	553.44	2388.12	9046.82	1.3	47.38	179

Your Job is done
Customer can take from it

Connect other Python Libraries -

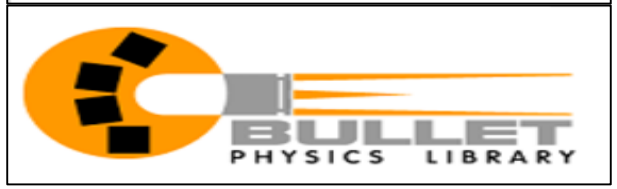


data analysis



Statistical machine Learning

data analysis



sensors

robots

discrete event simulation

Quick Prototyping

python

PYNQ™

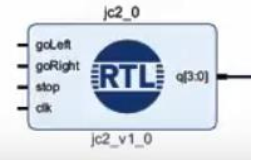
Embedded Acceleration

C++

SDSoC™ Environment

VIVADO HL*x* Editions

RTL Acceleration

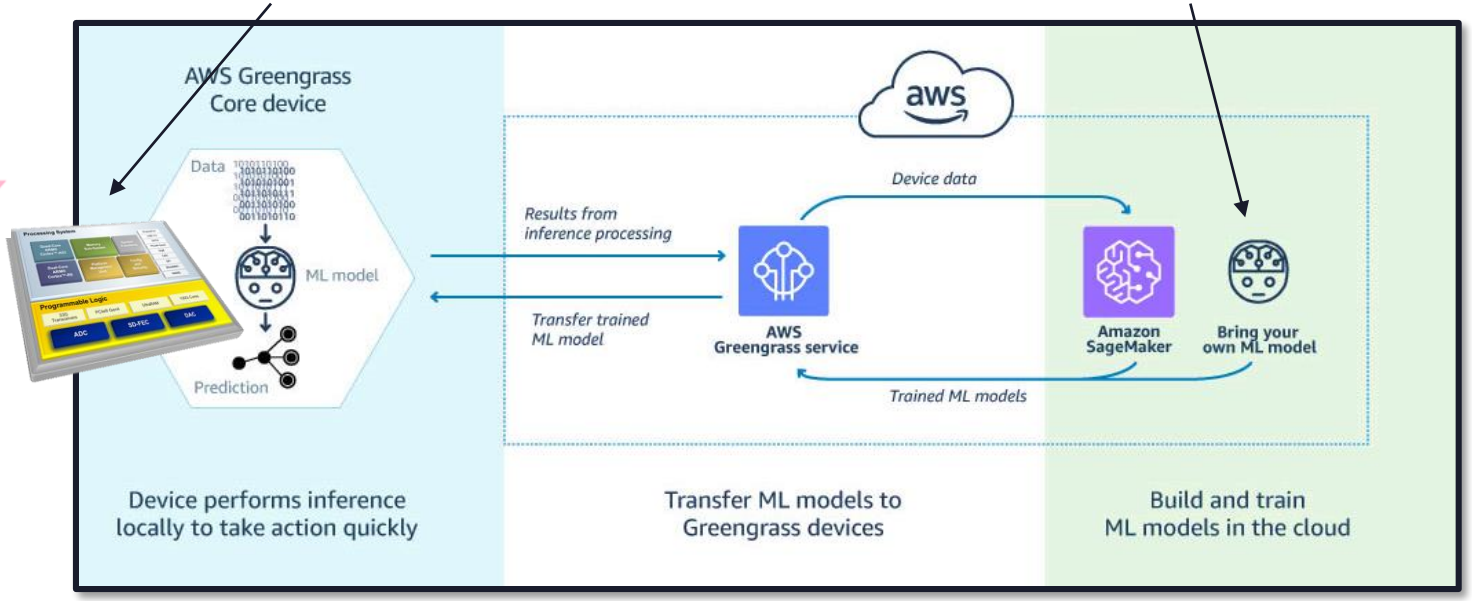


OVERLAYS

XILINX.

Edge-to-cloud co-design

Common JupyterLab tooling at edge and cloud



PYNQ enables ML experts and radio engineers to focus on their ‘value-add’

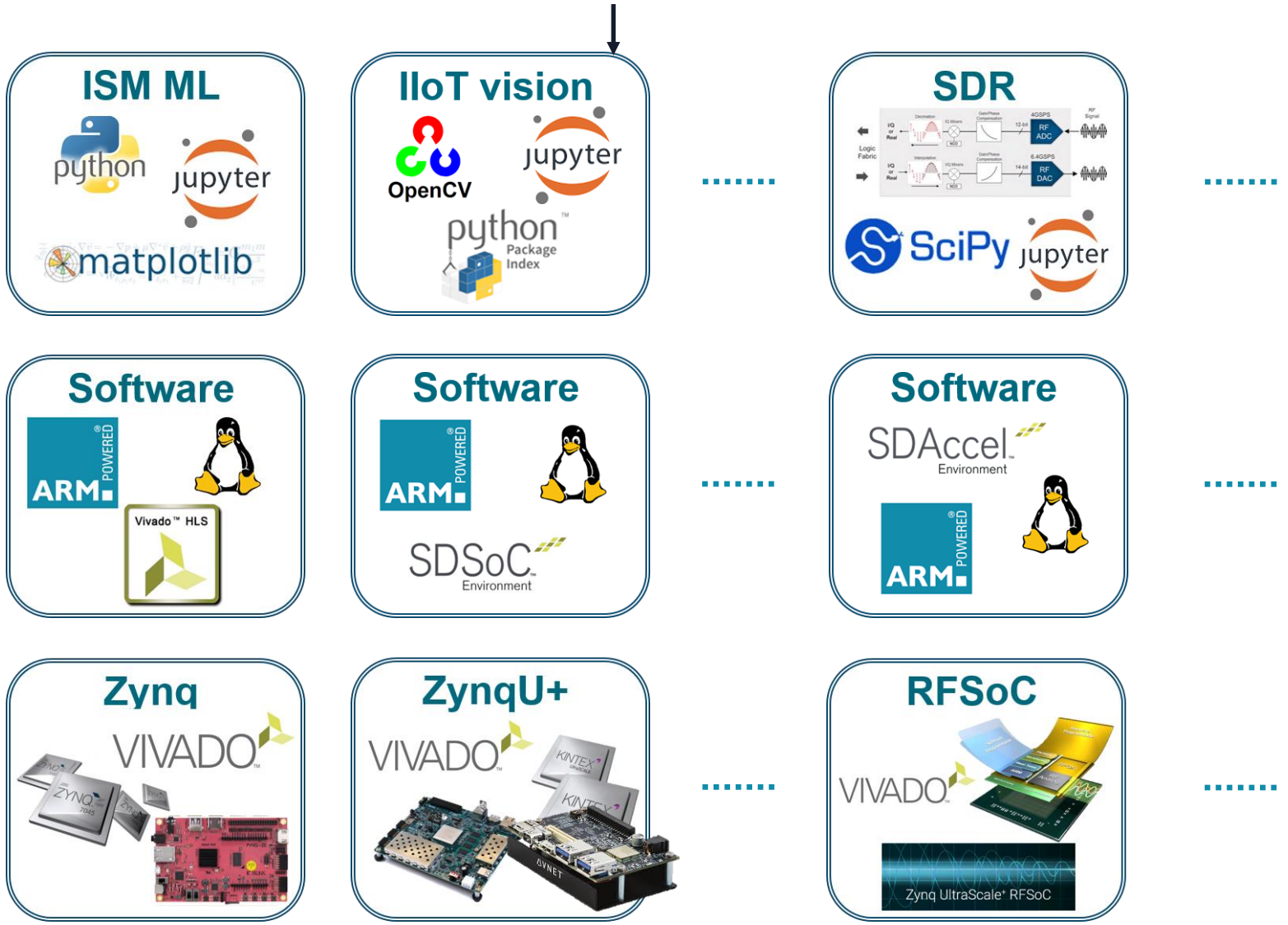
Edge-to-cloud co-design trade-offs:

- Maximize on-chip processing
- Minimize edge-to-cloud data exchange
- Exploit scalability of cloud processing
- Aggregate intelligence between and across multiple edge nodes
- Co-optimize the above for best system performance

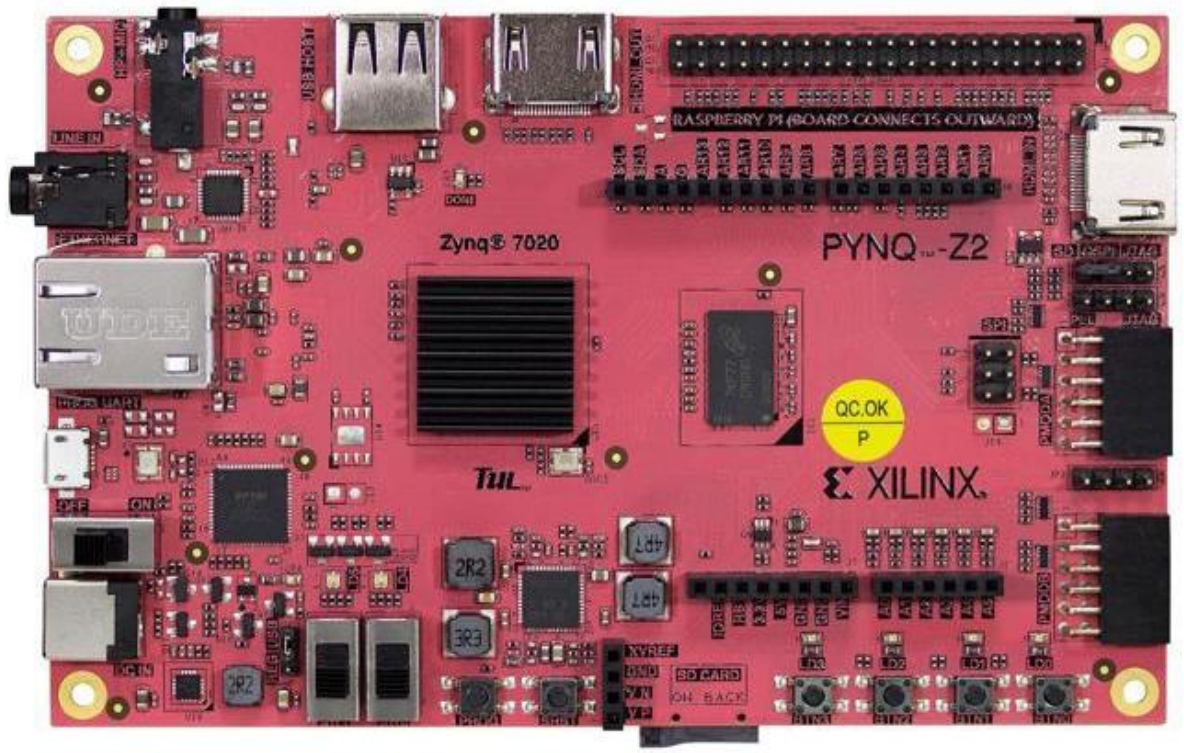
PYNQ Boards

XILINX®

Boards scaling across platforms and domains



PYNQ-Z2 Board
-design for starter



$119 to everyone in US

- New PYNQ reference platform
- New stereo audio with on-board codec
- New Raspberry Pi connector
- Open source design
- Z2 manufactured in Taiwan by TUL
- Distributed globally by Premier Farnell
- Also Newegg in US
- Academic discounts & donations available

Efficient porting PYNQ to any Zynq-based platform

3 Target-specific PYNQ components

Board-specific porting
Common interfaces available for re-use

2 PYNQ Software Core (board independent)

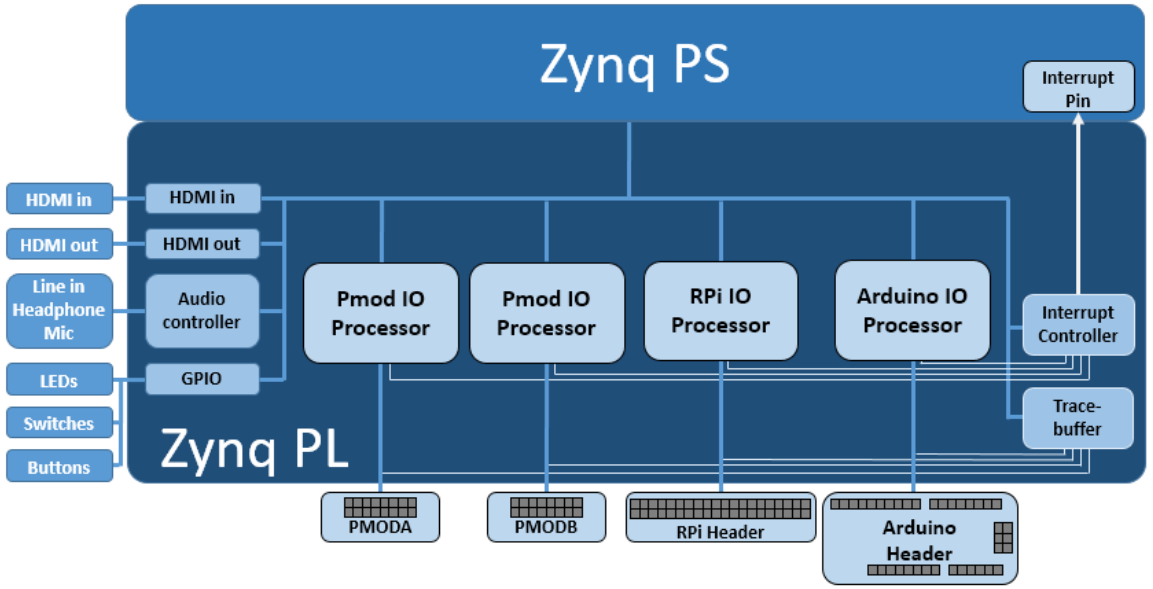
Pre-built images available
Expected to take less than 1 day to port

1 PetaLinux build tools/BSP

Available for any Xilinx-supported board;
Instructions for building for custom boards available

PYNQ补充 – Base Overlay介绍

PYNQ-Z2 Base Overlay（默认Overlay）



The base overlay on PYNQ-Z2:

- HDMI (Input and Output)
- Audio codec
- User LEDs, Switches, Pushbuttons
- 2x Pmod PYNQ MicroBlaze
- Arduino PYNQ MicroBlaze
- RPi (Raspberry Pi) PYNQ MicroBlaze
- 4x Trace Analyzer

加载 Overlay

- **通过overlay class加载**

```
In [1]: from pynq import Overlay
        overlay = Overlay("base.bit")
```

- **Base overlay具有专用的BaseOverlay class**
  - 可通过读写对象属性的方法来访问Overlay内的IP
  - 仅适用于Base Overlay，不可用于其它可下载的或自定义Overlay

```
In [1]: from pynq.overlays.base import BaseOverlay
        base = BaseOverlay("base.bit")
```

浏览Overlay内硬件库

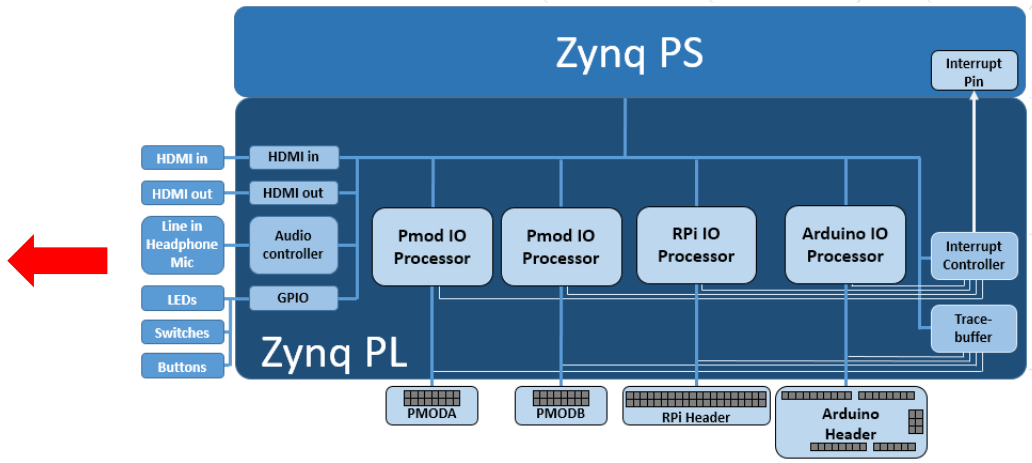
help命令查看硬件库资源

```
In [3]: help(base)
```

```
Help on BaseOverlay in module pynq.overlays.base.base object:

class BaseOverlay(pynq.overlay.Overlay)
 |  The Base overlay for the Pynq-Z2
 |
 |  This overlay is designed to interact with all of the on board peripherals
 |  and external interfaces of the Pynq-Z2 board. It exposes the following
 |  attributes:
 |
 |  Attributes
 |  ----------
 |  iop_pmoda : IOP
 |      IO processor connected to the PMODA interface
 |  iop_pmodb : IOP
 |      IO processor connected to the PMODB interface
 |  iop_arduino : IOP
 |      IO processor connected to the Arduino interface
 |  iop_rpi : IOP
 |      IO processor connected to the RPi interface
```

```
In [3]: help(base)
```

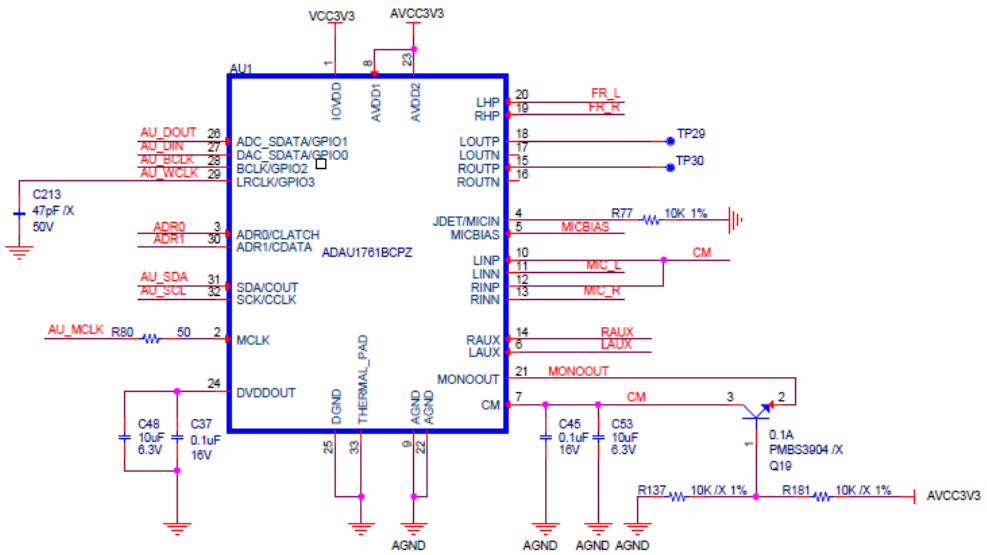
```
 |      Trace analyzer block on PMODA interface, controlled by PS.
 |  trace_pmodb : pynq.logictools.TraceAnalyzer
 |      Trace analyzer block on PMODB interface, controlled by PS.
 |  leds : AxiGPIO
 |      4-bit output GPIO for interacting with the green LEDs LD0-3
 |  buttons : AxiGPIO
 |      4-bit input GPIO for interacting with the buttons BTN0-3
 |  switches : AxiGPIO
 |      2-bit input GPIO for interacting with the switches SW0 and SW1
 |  rgbleds : [pynq.board.RGBLED]
 |      Wrapper for GPIO for LD4 and LD5 multicolour LEDs
 |  video : pynq.lib.video.HDMIWrapper
 |      HDMI input and output interfaces
 |  audio : pynq.lib.audio.Audio
 |      Headphone jack and on-board microphone
 |  pin_select : GPIO
 |      The pin selection between PMODA (0) and RPI header (1).
 |
 |  Method resolution order:
```



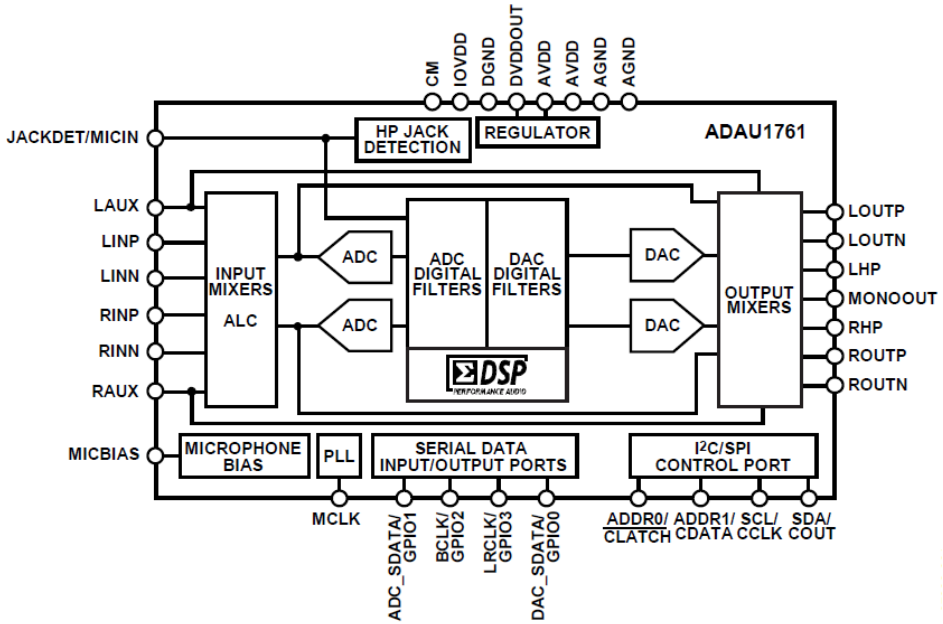
XILINX.

Audio – 硬件

> 原理图



> ADAU1761 - SigmaDSP Stereo, up to 96 kHz sample rate, 24-Bit Audio Codec with Integrated PLL



XILINX.

Audio

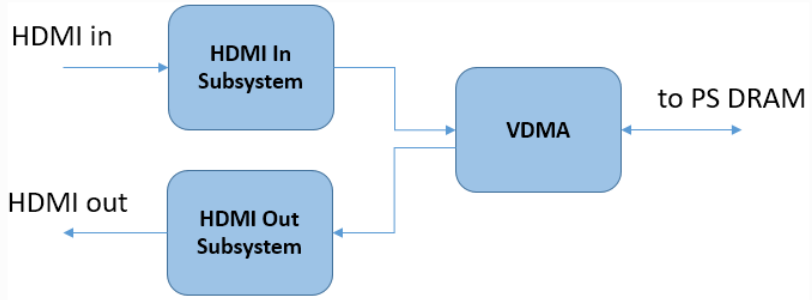
- help查看提供的方法与属性
- 允许audio的示例
- 更多内容请参考： <Jupyter Home>/base/audio/audio_playback.ipynb

```
In [2]: help(base.audio)

    play(self)
        Play audio buffer via audio jack.

        Since both channels are sampled, the buffer size has to be twice
        the sample length.

        Returns
        -------
        None

    record(self, seconds)
        Record data from audio controller to audio buffer.

        The sample rate for both channels is 48000Hz. Note that the
        `sample_len` will only be changed when the buffer is modified.
        Since both channels are sampled, the buffer size has to be twice
        the sample length.

        Parameters
```

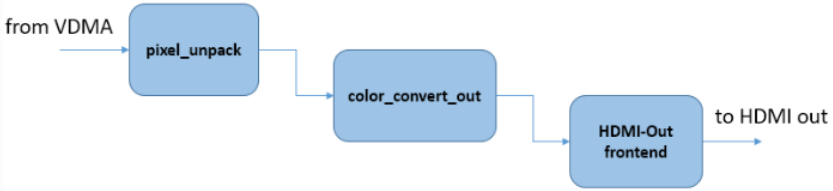
```
In [3]: from pynq.overlays.base import BaseOverlay

In [4]: base=BaseOverlay("base.bit")

In [5]: help(base.audio)

        Help on AudioADAU1761 in module pynq.lib.audio object:

        class AudioADAU1761(pynq.overlay.DefaultIP)
         |  Class to interact with audio codec controller.
         |
         |  Each raw audio sample is a 24 bits, padded to 32 bits.
         |  The audio controller supports both mono and stereo modes, and I2S format
         |  of data.
         |
         |  Attributes
         |  ----------
         |  buffer : numpy.ndarray
         |      The numpy array to store the audio.
         |  sample_rate: int
         |      Sample rate of the codec.
         |  sample_len: int
         |      Sample length of the current buffer content.
         |  iic_index : int
         |      The index of the IIC instance in /dev.

In [6]: base.audio.load("/home/xilinx/jupyter_notebooks/base/audio/recording_0.wav")

In [8]: base.audio.play()
```

Video

Block Diagram

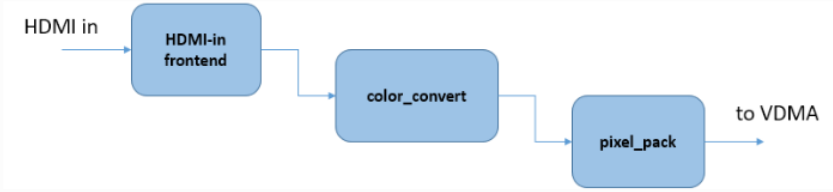


HDMI-In



The Pixel Unpack and the Color Convert block allow conversion between different color spaces at runtime.

HDMI-Out



XILINX.

Video

Set up an instance of the HDMI-in, and HDMI-out.

```
from pynq import Overlay
from pynq.lib.video import *

base = Overlay('base.bit')
hdmi_in = base.video.hdmi_in
hdmi_out = base.video.hdmi_out
```

Configuration

The HDMI-in interface is enabled using the `configure` function which can optionally take a colorspace parameter. If no colorspace is specified then 24-bit BGR is used by default. The HDMI-in *mode* can be used to configure the HDMI-out block. This specifies the output color space and resolution.

```
hdmi_in.configure()
hdmi_out.configure(hdmi_in.mode)
```

Execution

Once the HDMI controllers have been configured, they can be started:

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

To connect a simple stream from HDMI-in to HDMI-out, the two streams can be tied together.

```
hdmi_in.tie(hdmi_out)
```

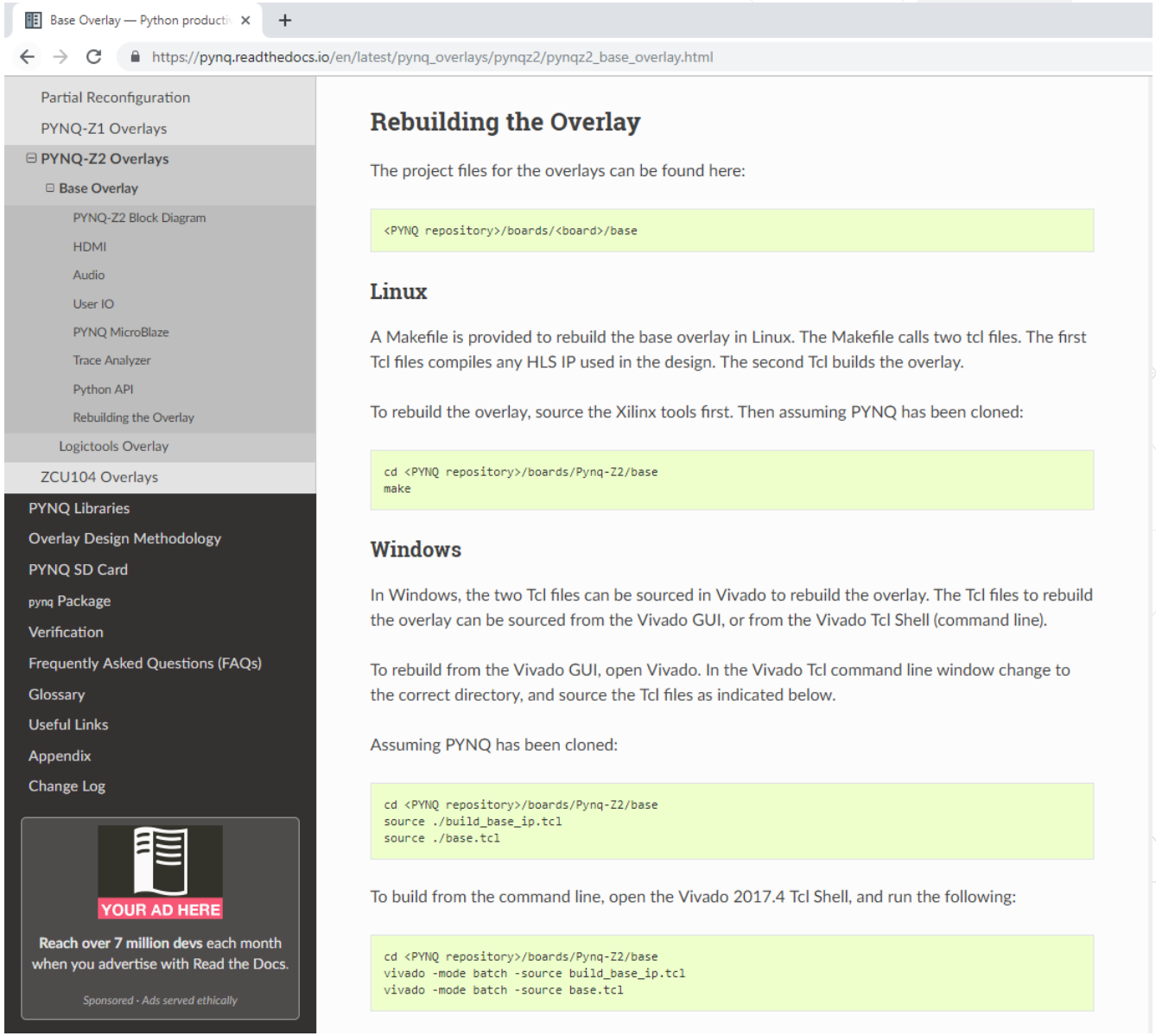
This takes the unmodified input stream and passes it directly to the output. While the input and output are tied frames can still be read from the input but any call to `hdmi_out.writeframe` will end the tie.

```
frame = hdmi_in.readframe()
...
hdmi_out.writeframe(frame)
```

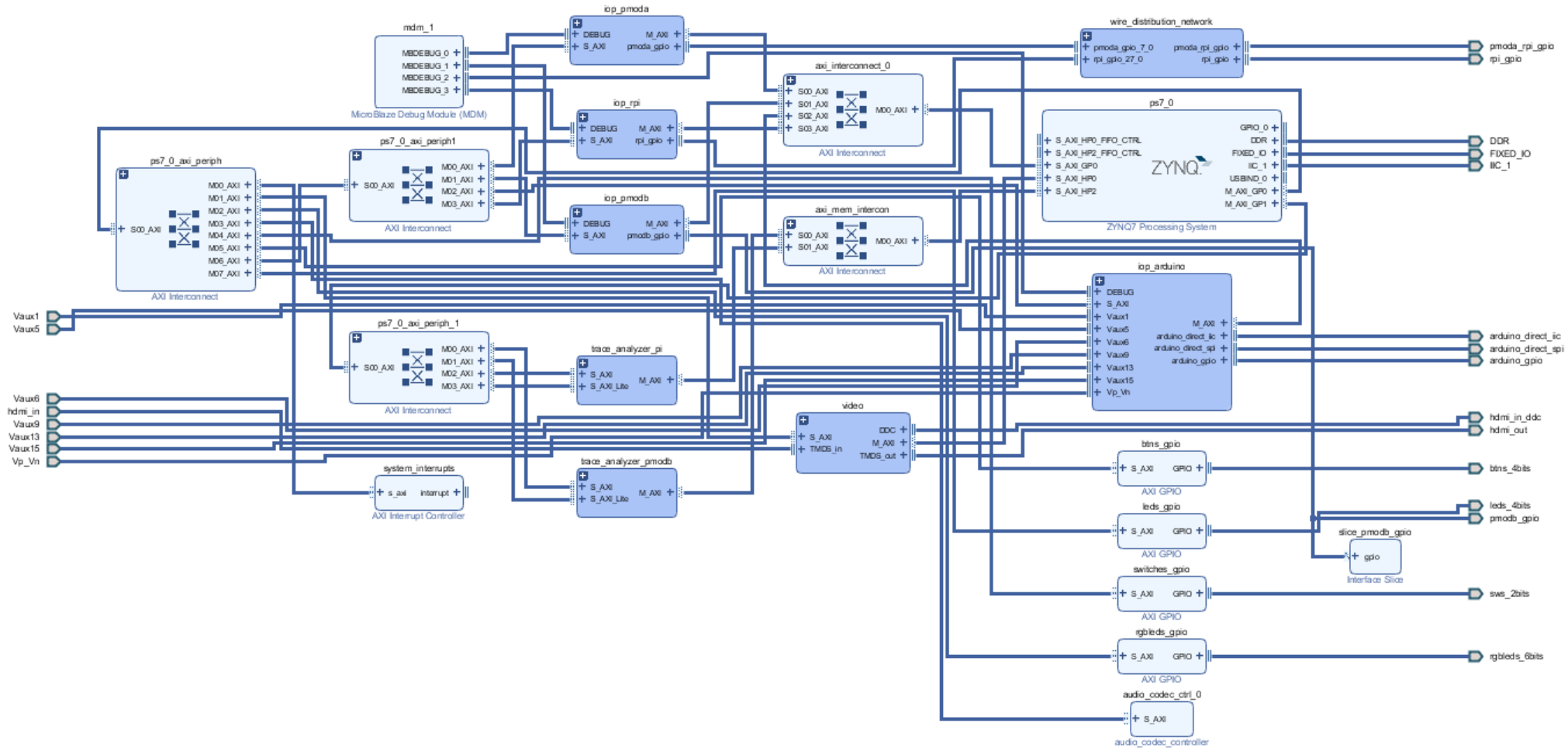
重新build Base Overlay Vivado工程

1. 下载Pynq 源代码到本地；
   https://github.com/Xilinx/PYNQ

1. 打开Vivado软件，在TCL
   cd <PYNQ repository>/boards/Pynq-Z2/base
   source ./build_base_ip.tcl
   source ./base.tcl

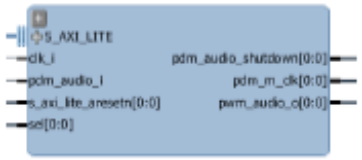


Base Overlay Block Design

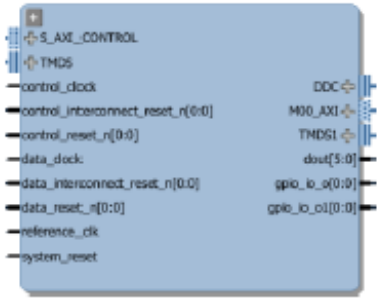


XILINX.

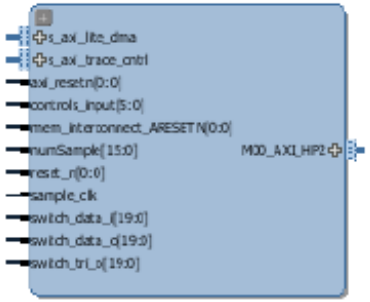
Base Overlay 可重用IP



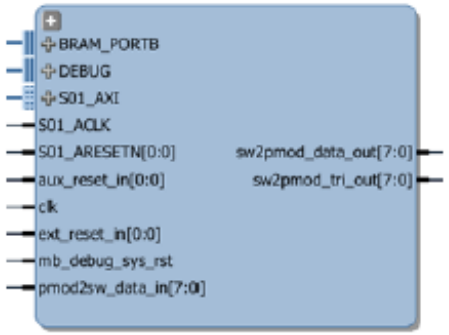
Audio



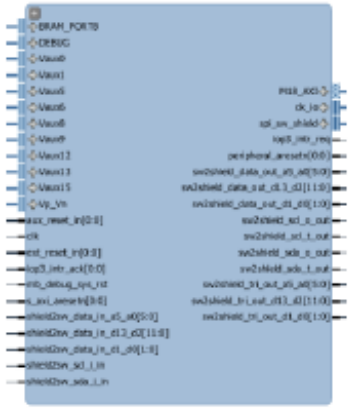
Video



Tracebuffer



Pmod IOP



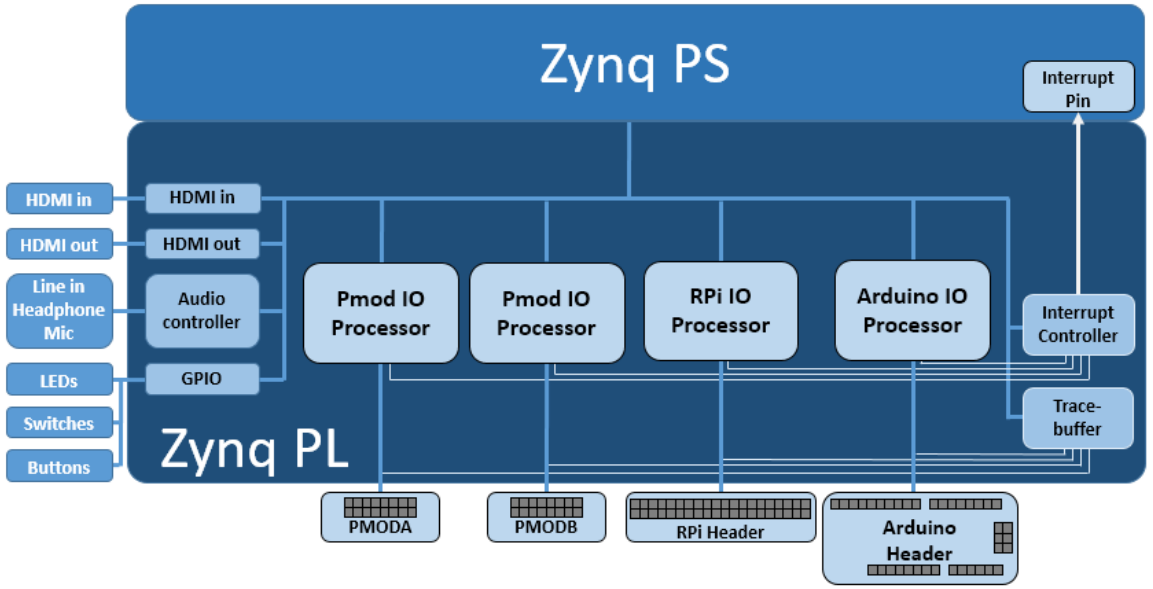
Arduino IOP

XILINX.

PYNQ补充 - Slave processor

XILINX®

PYNQ-Z2 Base Overlay（默认Overlay）

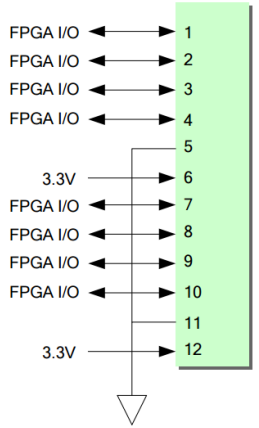


The base overlay on PYNQ-Z2:

- HDMI (Input and Output)
- Audio codec
- User LEDs, Switches, Pushbuttons
- 2x Pmod PYNQ MicroBlaze
- Arduino PYNQ MicroBlaze
- RPi (Raspberry Pi) PYNQ MicroBlaze
- 4x Trace Analyzer

Typical Pmod IO physical and electrical Interfaces

Pmod



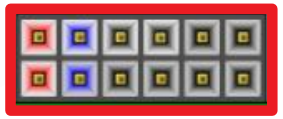
Pin	Signal	Description
1	~CS	Chip Select
2	MOSI	Master-Out-Slave-In
3	(NC)	Not Connected
4	SCLK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V/5V)

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

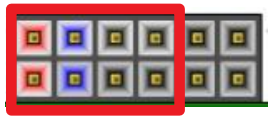
Pin	Signal	Description
1 & 5	SCL	Serial Clock
2 & 6	SDA	Serial Data
3 & 7	GND	Power Supply Ground
4 & 8	VCC	Power Supply (3.3V/5V)

DAC

OLED

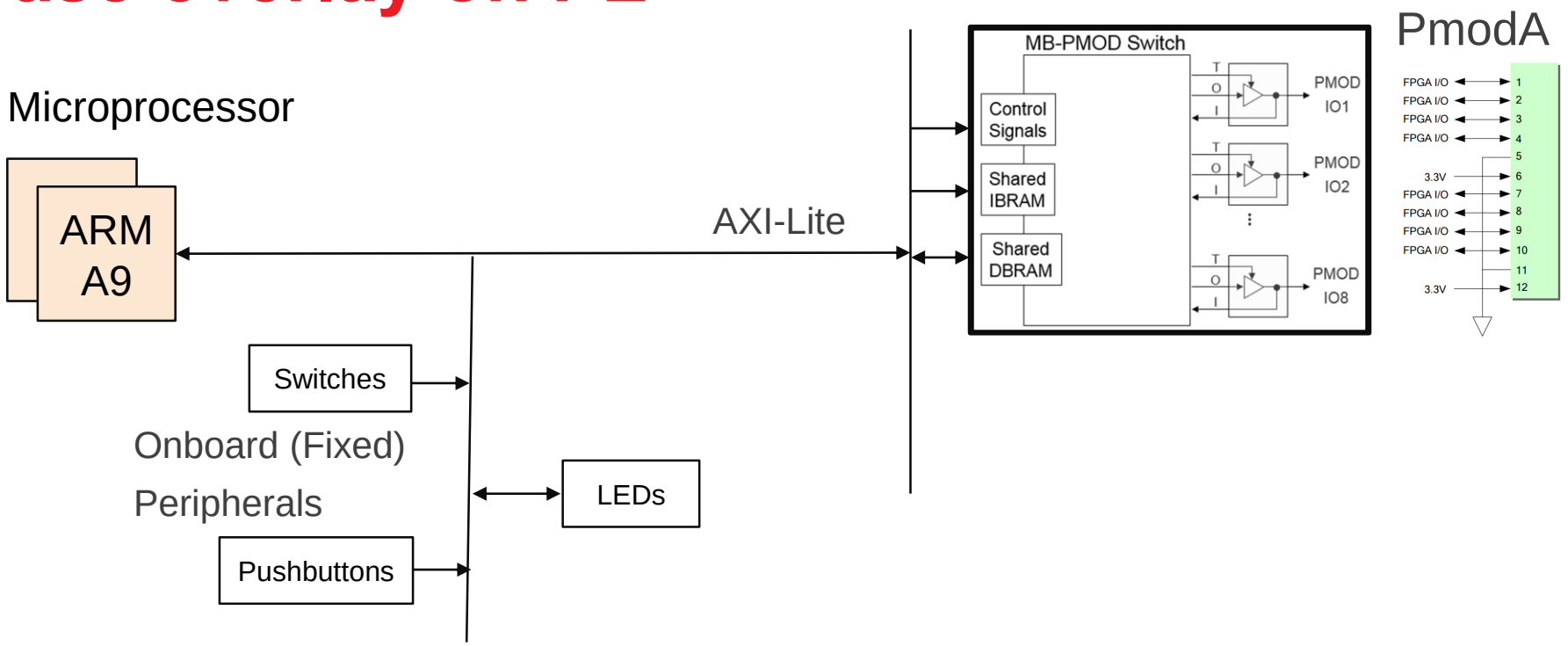


ADC



XILINX.

Load ‘base’overlay on PL

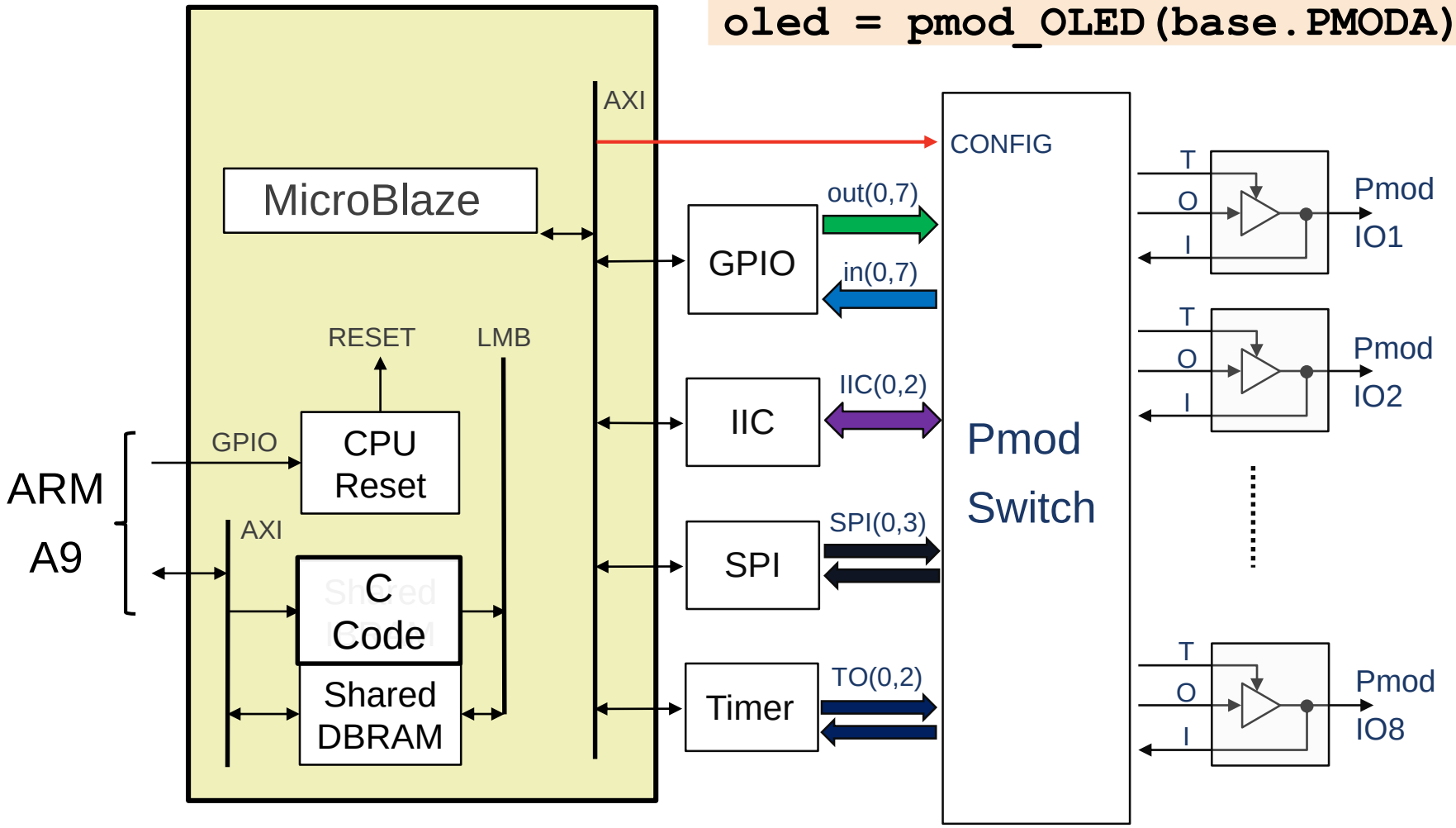


Python code on ARM A9:

```
from pynq.overlays.base import BaseOverlay
from pynq.lib import Pmod_OLED

base = BaseOverlay("base.bit")
```

Configure IO Processor (IOP) for PmodA

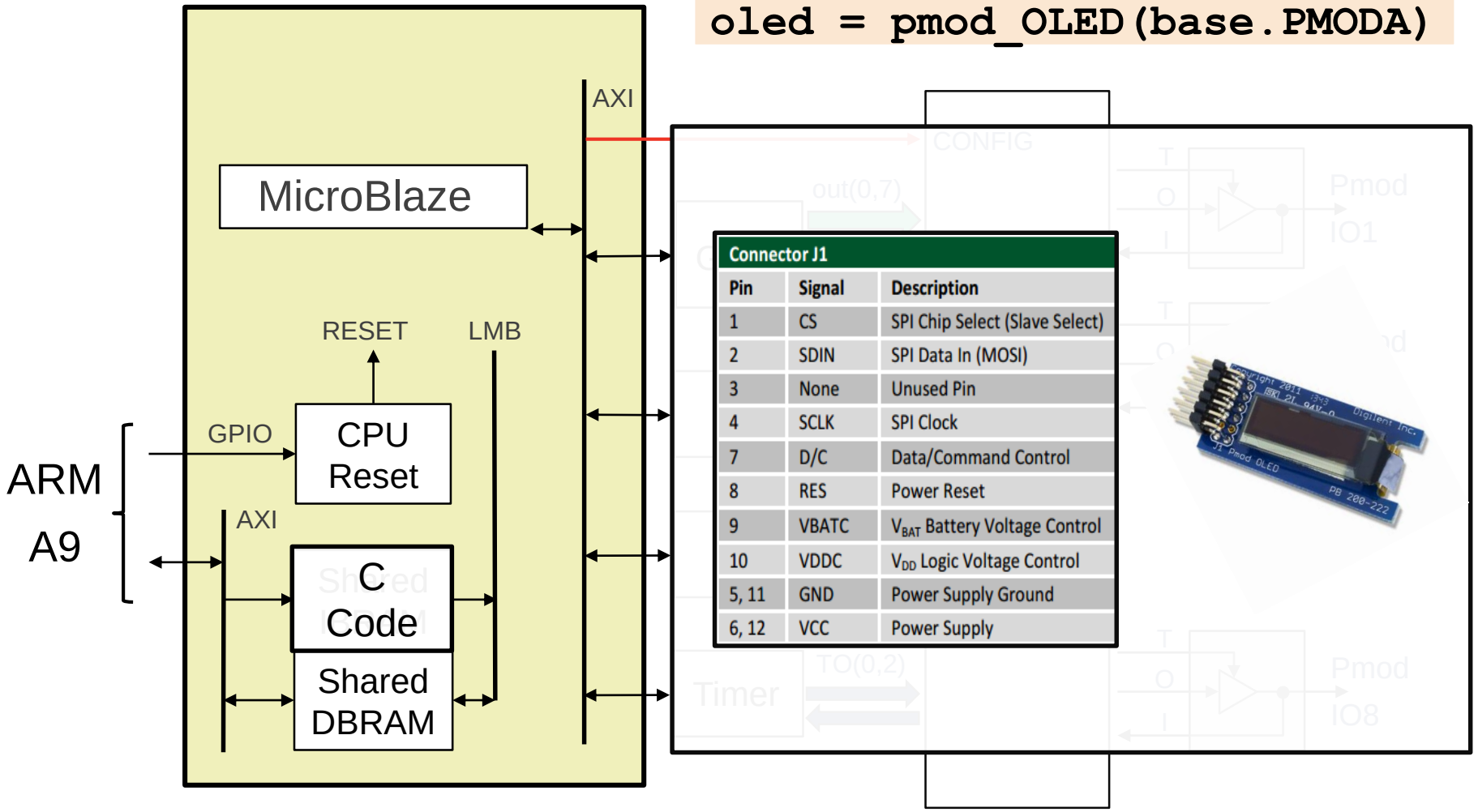
```
oled = pmod_OLED(base.PMODA)
```




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

Configure IO Processor (IOP) for PmodA

```
oled = pmod_OLED(base.PMODA)
```



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

Jupyter Notebook

In [1]:

```
1 from pynq.overlays.base import BaseOverlay
2 from pynq.lib import Pmod_OLED
3
4
5 base = BaseOverlay("base.bit")
6 oled = Pmod_OLED(base.PMODA)
7 oled.write('1 2 3 4 5 6')
```

A total of 5 lines of user code

… thanks to Python, FPGA overlays, abstraction & re-use

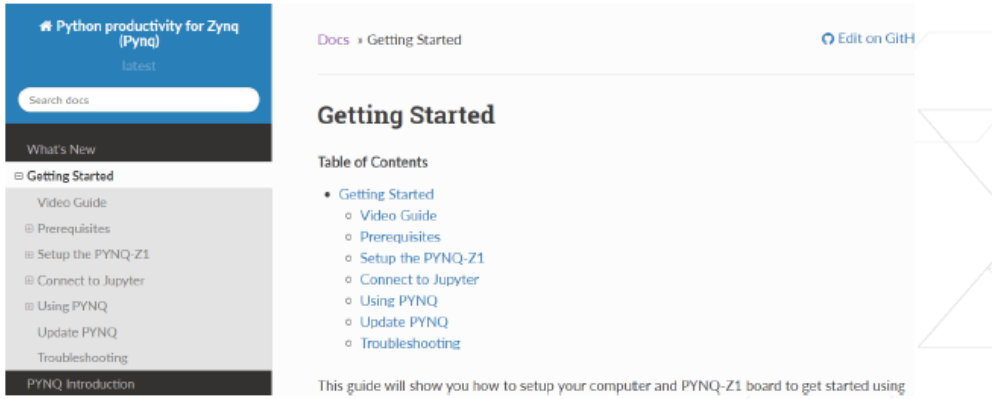
自定义Overlay封装与分发

Overlay分发

- **“Overlay” (optionally) consists of**
  - Bitstream (.bit)
  - Tcl (.tcl)
  - Python (.py)
  - Notebook (.ipynb)
  - Documentation
  - C library (.so), source files, header files…
- **Pip package management can be used to install PYNQ packaged**
  - `pip install` (pip3 install in current image)
  - Include setup.py for pip

Overlay封装目录案例

```
|- LICENSE
|- README.md
|- setup.py
|- project
|   |-- __init__.py
|   |-- *.py
|   |-- test
|       |-- *.py
|       |-- test_project.py
|- docs
|   |-- conf.py
|   |-- index.rst
|   |-- *.rst
|- requirements.txt
|- TODO.md
```



- **Read The Docs**
  - Automatically generate documentation from GitHub repository
- **Include**
  - conf.py – configuration file
  - *.rst (Markdown files)
  - Jupyter notebooks

Pip install example setup.py (BNN)

- **setuptools**
  - Used by pip
- **Package data references all files to be installed**
  - Will be delivered to package directory
- **data_files can be used to install outside the package directory**

```
from setuptools import setup, find_packages
import pynq_proj

setup(
    name = "pynq_proj",
    version = pynq_proj.__version__,
    url = 'https://github.com/pynq_proj',
    license = 'Apache Software License',
    author = "Cathal McCabe",
    author_email = "cmccabe@xilinx.com",
    packages = ['ws2812'],
    package_data = {
    '' : ['*.bit','*.tcl','*. py'],
    },
    description = "PYNQ Project ..."
)
```

Overlay封装分发方针

- **Distribute standalone Python package**
  - Not a fork of PYNQ
- **GitHub recommended**
  - Include readme.md - displays on GitHub home page
  - Bug tracking, feature request, version management
  - Jupyter Notebooks render on GitHub – easy to share and demonstrate work
- **Specify License**
  - BSD3 preferred (compatible with other open source licenses)
- **Verification**
  - py.test for regression should ship with package
- **Pip installable**

https://jeffknupp.com/blog/2013/08/16/open-sourcing-a-python-project-the-right-way/

Actions

PYNQ™

pynq.io

pynq.readthedocs.org

github.com/Xilinx/PYNQ

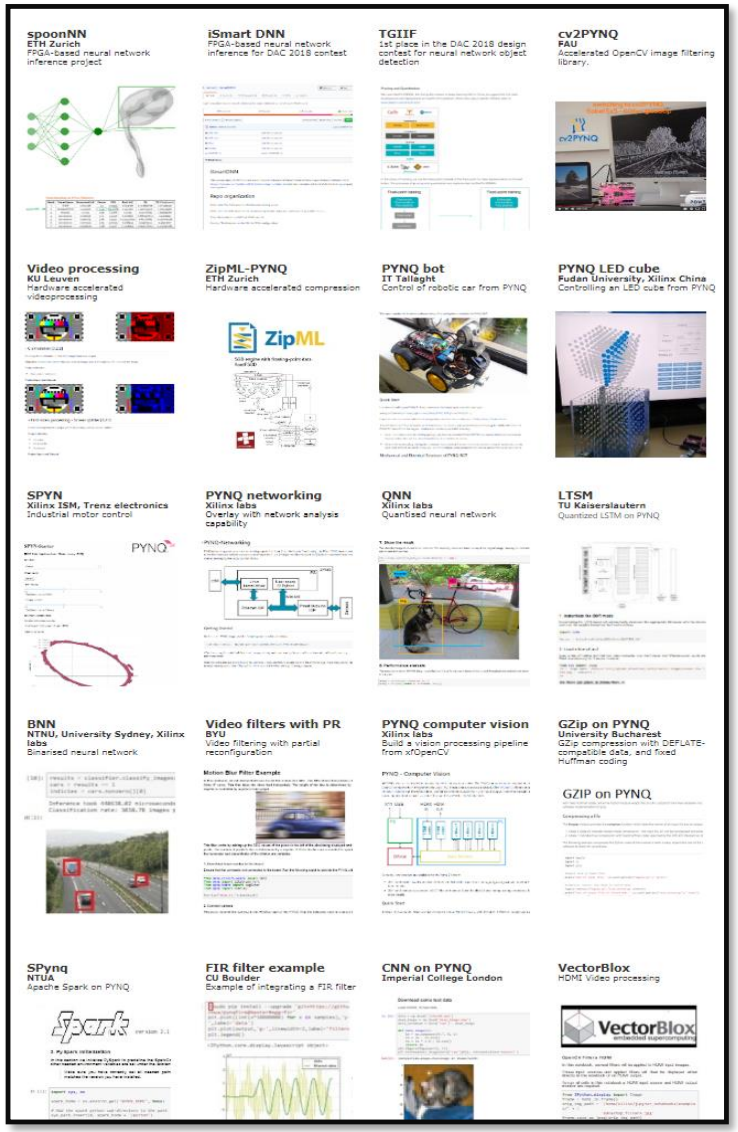
tul.com.tw/ProductsPYNQ-Z2.html

pynq.io/support

XILINX.

http://www.pynq.io/community.html

- 浏览已有Overlay

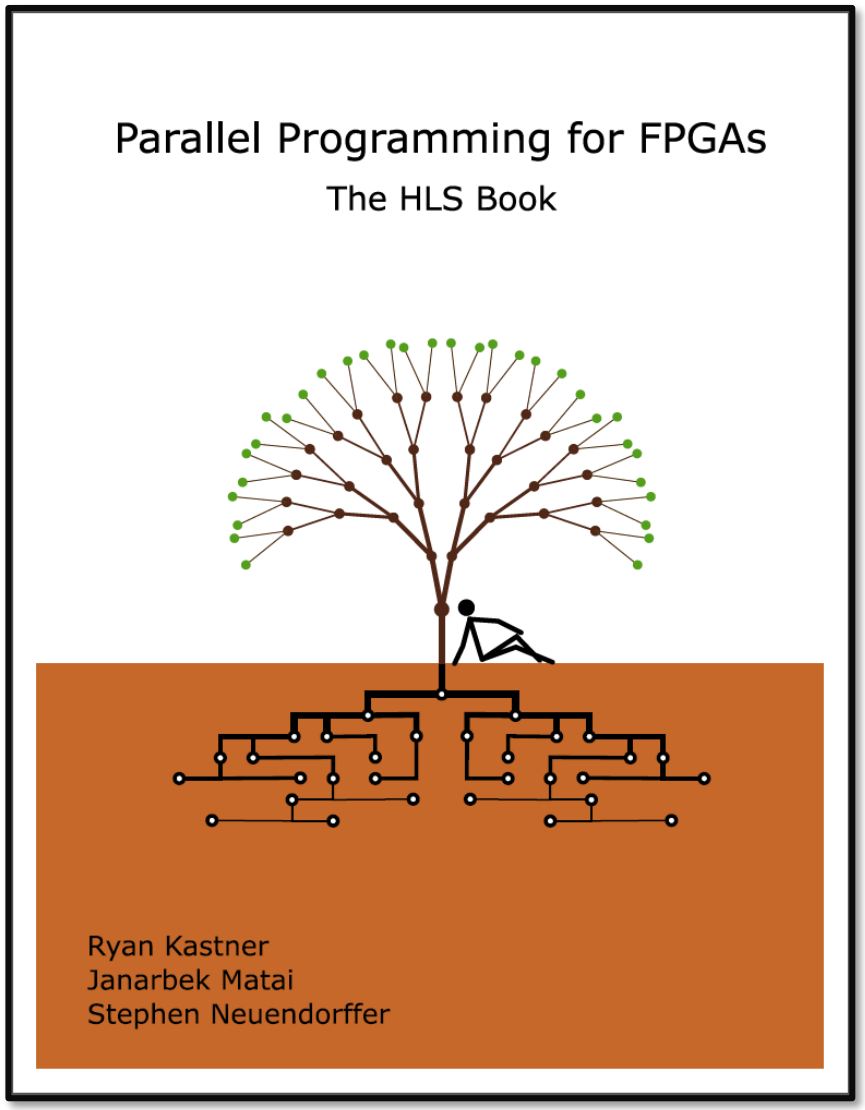


PYNQ框架学习路径

- **仅仅想使用已有的Overlay快速开发应用**
  - Python编程
  - PYNQ API接口

- **想掌握PYNQ完整开发路径**
  - Vivado开发流程
  - HLS
  - Python编程

Open source HLS book



Parallel Programming for FPGAs is an open-source book aimed at teaching hardware and software developers how to efficiently program FPGAs using high-level synthesis

PYNQ

公众号回复 pp4fpgas

XILINX.

思考：PYNQ框架到底有什么用？

- 利用已有Overlay快速开发app
- 提供了一种软硬件协同设计的简单实现方式
- 提供一种简单的验证FPGA算法的方法
- 学习Python的好选择

Adaptable.
Intelligent.

XILINX