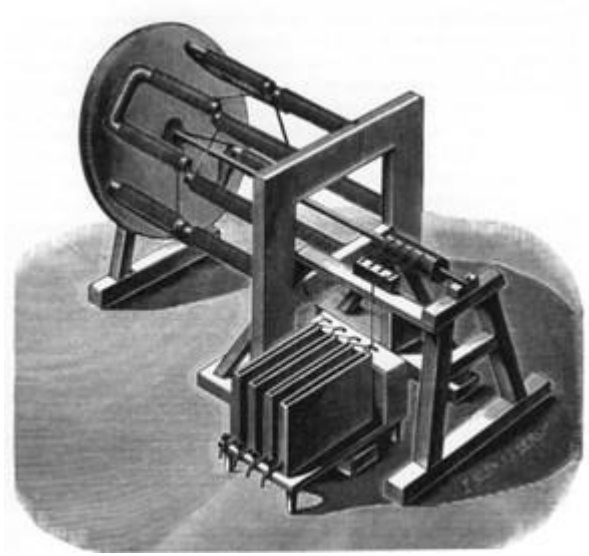
Python Powered Edge Analytics & Machine Learning for Electric Drives

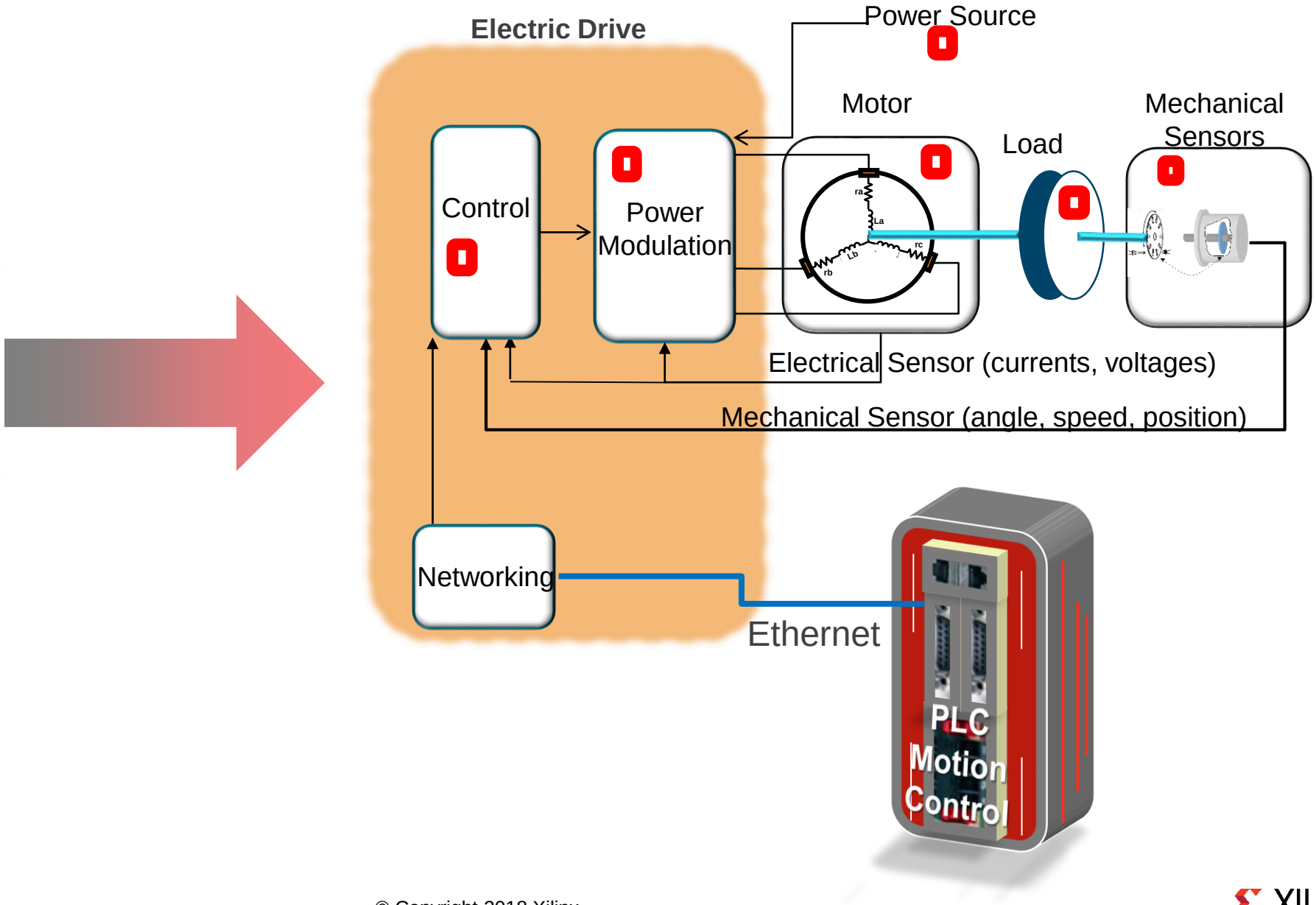
June 20, 2018

XILINX.

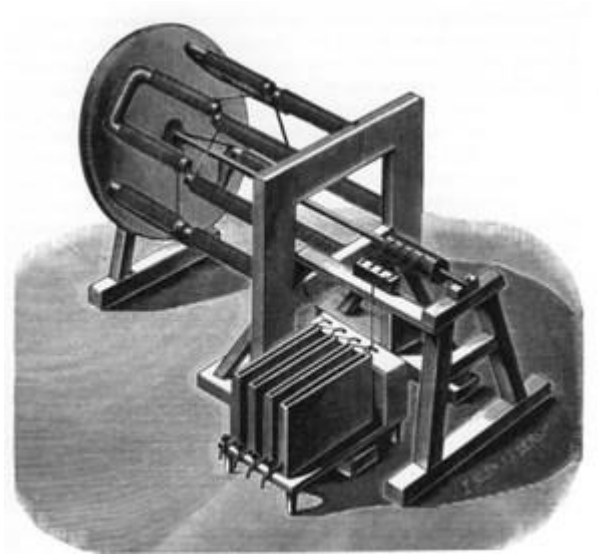
Introduction to Modern Electric Drives

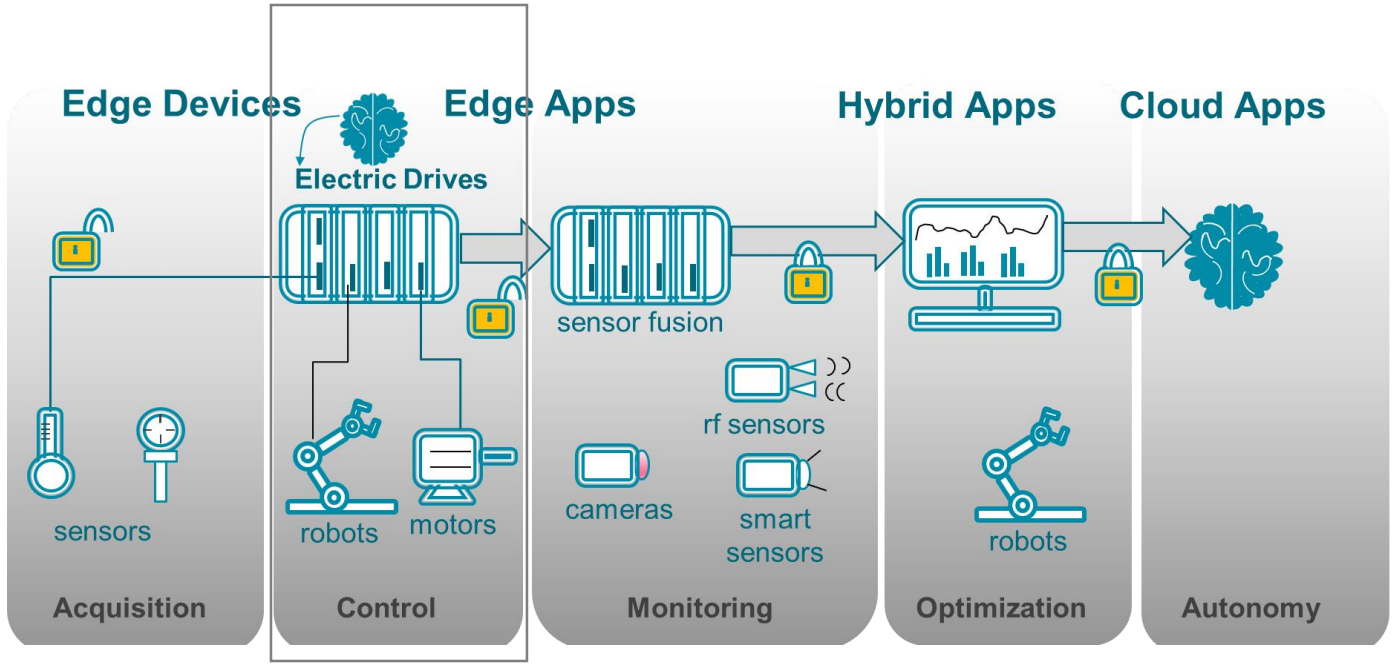
What is an Electric Drive





Industrie 4.0 and IIoT: Electric Drives are Edge Devices



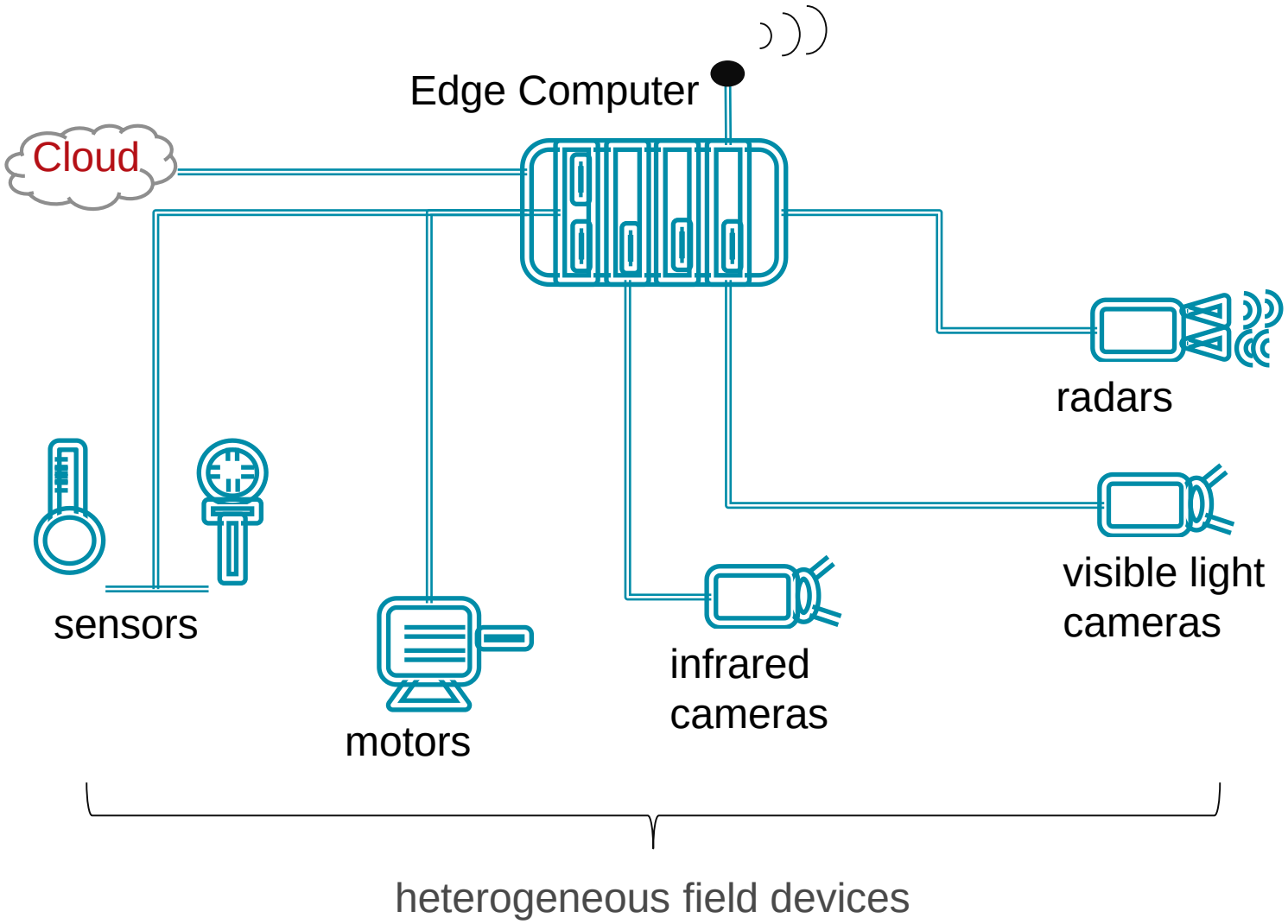


Today's Distributed Industrial Architectures
Industrie 4.0 / Industrial IoT

Electric Drives have evolved—
They are expected to do a lot more!

XILINX.

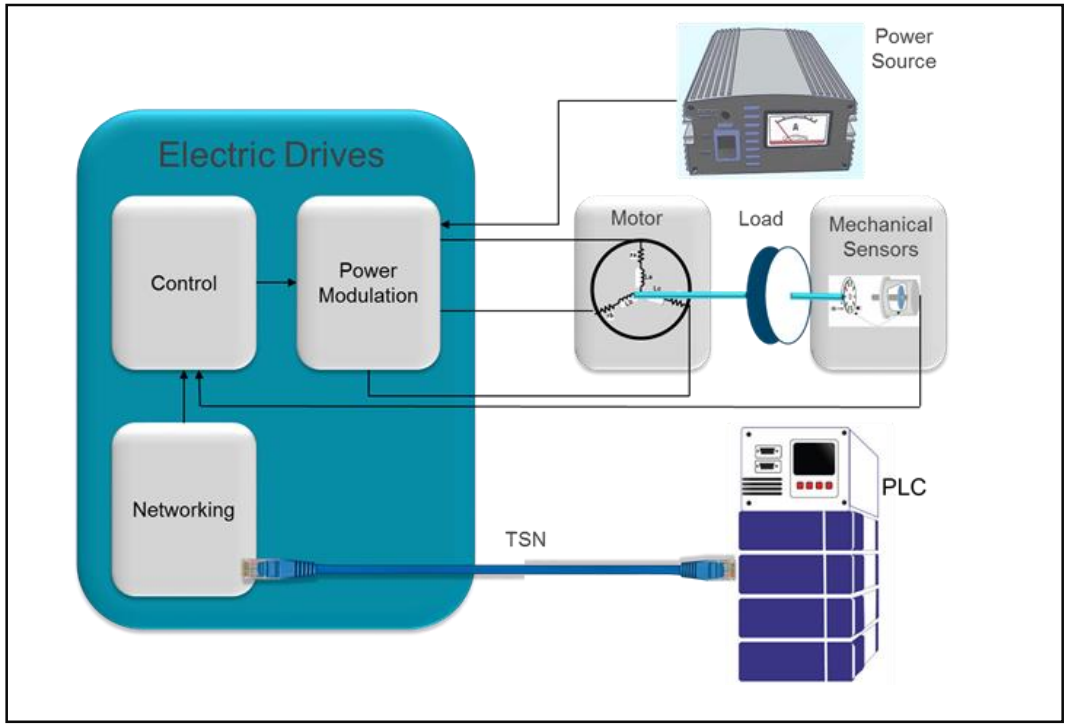
Edge computing solving some of today's challenges



- Minimize Connectivity cost
- Heterogeneous field devices
- Data collection
- Data ingestion
- Data aggregation
- In situ digital signal processing
- In situ data formatting
- Interaction with control
- Interaction with cloud
- Scalability

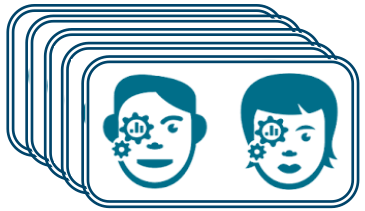
Common Features of IIoT-enabled Electric Drives

- Support IT-OT convergence without compromising OT responsiveness
- Support for higher number of sensors
- Networking over Industrial Standards
- Cybersecurity
- Functional Safety
- Improved diagnostic, logging, resilience
- Fast development time through platforms
- Sometime HMI, many more
  - Watch Introduction to Modern Electric Drives



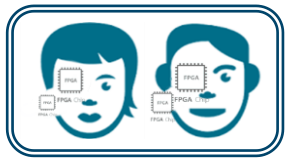
Modern Electric Drive System

Impact of User Personas and Industry Trends



Domain Experts

Emb. software Engineers



Hardware Engineers

Most new graduates specializing in machine learning or analytics are familiar with python programming & ecosystems around it

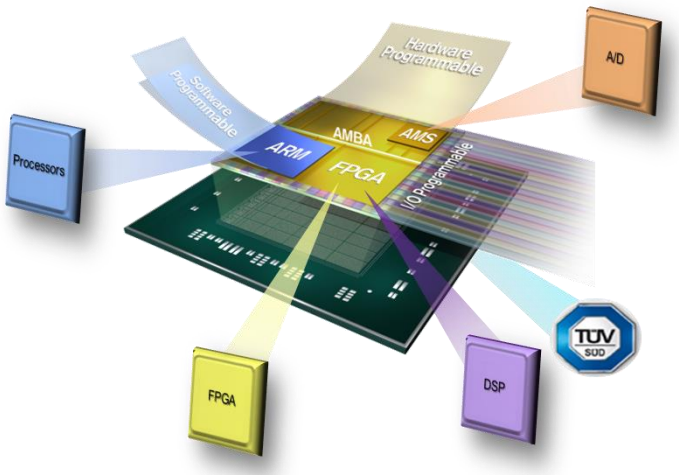
Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C		99.7
3. Java		99.4
4. C++		97.2
5. C#		88.6
6. R		88.1
7. JavaScript		85.5
8. PHP		81.4
9. Go		76.1
10. Swift		75.3

https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages

Xilinx response to current trends

- **ZYNQ SoC Portfolio:** Arm processing system with varying size of programmable logic
- **Design Tools & Methodology:** SDSoC and Vivado HLS
- **IIoT Solution Stack:** Xilinx, Ecosystem building blocks and solutions used across Industrial IoT platforms
- **EDDP:** Platform to provide methodology for creating motor control (or any) solution from C/C++ code
- **PYNQ:** A hardware-software framework to bring python productivity to ZYNQ

Zynq SoCs



Xilinx Tools

SDSoC Environment

Vivado™ HLS

XILINX IIoT SOLUTION STACK

- Application
- Analytics & Machine Learning
- OS/RTOS & Hypervisors/Containers
- Any-to-Any Connectivity
  Smarter Control
  Embedded Vision
- Functional Safety & Cybersecurity
- Silicon Architecture

Xilinx Kits

Electric Drives Demonstration Platform

PYNQ™

XILINX

Case Study from SPS IPC Drives 2017: Kollmorgen

- ***"The tremendous integration enabled by Zynq boosts control performance and encompasses safety, multiple communication buses, a display for easy setup and diagnosis and all relevant encoder types."***

- **Value of Zynq and Zynq UltraScale+ SoCs**
  - Lowest latency between IT and OT tasks through integration
  - Functional Safety
  - Networking over Industrial Standards
  - HMI
  - Improved diagnostic, logging, resilience
  - Fast development time through platforms
  - OT functions isolated from IT functions

Additional Case Study: STÖBER

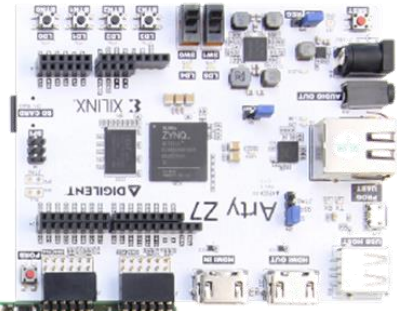
- **Xilinx Zynq is the processor for Stober's 6th generation of Drive Controllers SI6**
- ***"4 axes, 16 or 97? A single SI6 drive controller can control up to two axes. Thanks to the modular system, the number of motors or axes to be controlled can be freely scaled. The SI6 drive controller is the most compact solution on the market."***

- **Value of Zynq and Zynq UltraScale+ SoCs**
  - Compact and economical solution for multi axis control
  - EtherCAT or PROFINET, programmable through firmware
  - HIPERFACE DSL One Cable
  - PLe (Cat 4) / SIL 3 Safety following EN 13849-1
  - Safety over EtherCAT (FSoE)
  - OT functions isolated from IT functions

EDDP: Electric Drives Demonstration Platform

- Design Methodology Predicated on Open Source & Ease of Use
- EDDP takes complete advantages of **Xilinx Zynq SoCs**
- Platform offers two different flows to build motor control solutions
  - Xilinx SDSoC Design Flow
  - Xilinx Vivado HLS Design Flow

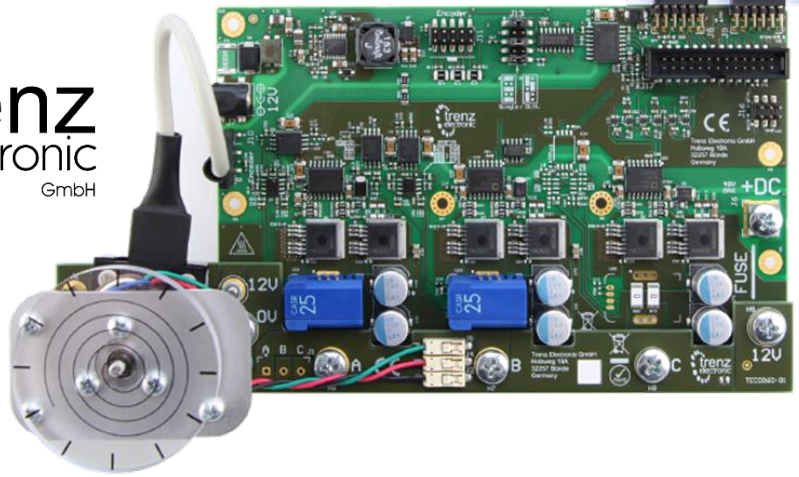
EDDP KIT



2

3

1

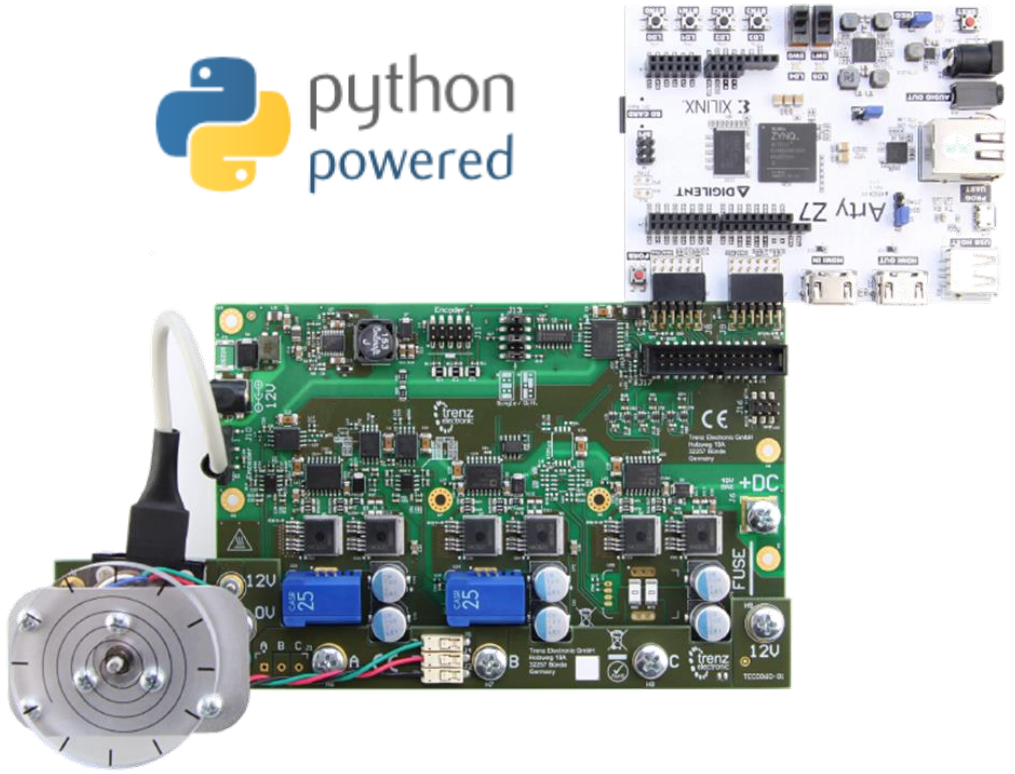


Three simple steps
to get started!
EDDP Demo Video

XILINX

Python Powered Electric Drives

- **Why?**
  - Unlock the data hidden in plain sight to make the drive intelligent and more capable without impacting performance
- **How?**
  - Testament to Xilinx flexibility—not a planned feature
  - Incorporated into the PYNQ Framework
- **What is the PYNQ Framework?**



Introduction to the PYNQ Framework

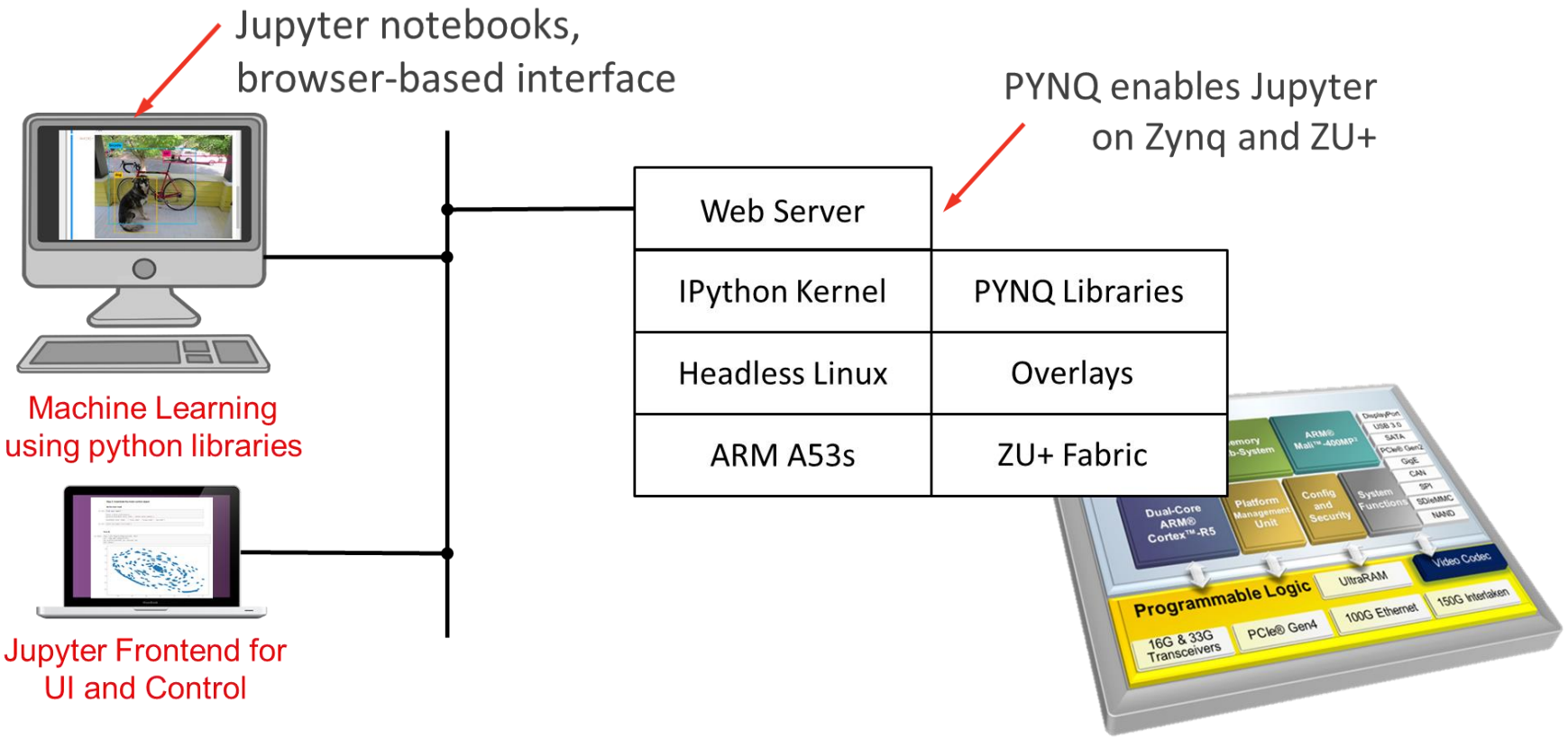
PYNQ: Python productivity for Zynq

What is PYNQ?

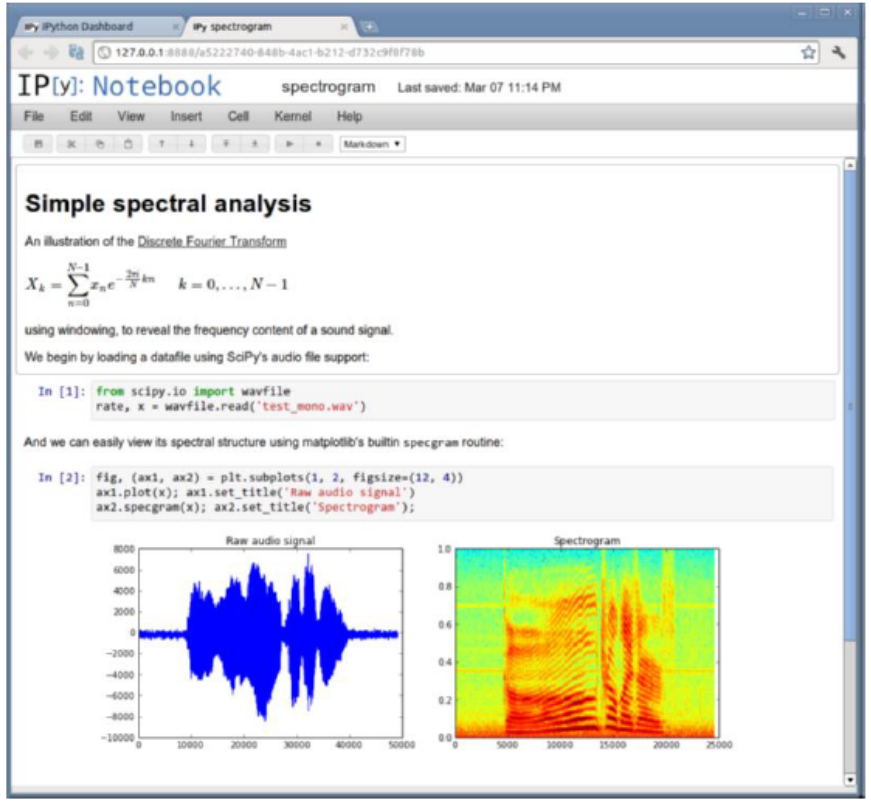
- PYNQ is a Software-Hardware framework
- PYNQ is an Open Source Project from Xilinx
- PYNQ includes software, libraries & overlays

Read the White Paper

PYNQ Basics



Jupyter Notebooks: browser-based development … with rich, multi-media support



github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks

- Designed for
  - Interactive, exploratory computing
  - Reproducible results
- Ideal for
  - Teaching and learning
  - Projects and research
- Rapid adoption
  - Across multiple disciplines

Part of a broader trend towards new, web-based IDEs

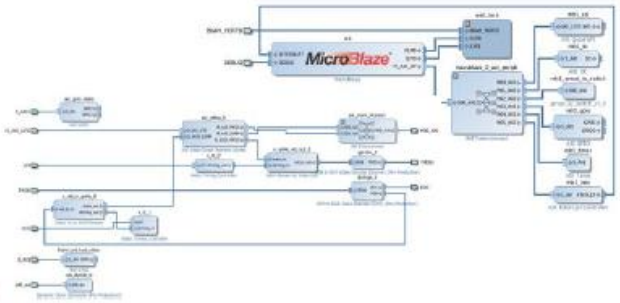
Introduction to Overlays (key differentiation for YOU)

What is an Overlay?

Overlays are FPGA designs, using the design pattern concept, that extend user application from processing system to programmable logic, efficiently pre-processing the data prior to interaction by Arm Processor

- **The .bit and .tcl files are generated by using Xilinx Tools**
- **Files are stored in the PYNQ overlay library**
- **PYNQ parses the files & recognizes the custom overlay**
- **Write custom Jupyter notebooks or scripts to interact**

Overlays (aka Hardware Libraries)



Step 1:
Create an FPGA design for a class of related applications

```
pmod_init(0,1);
while(1){
    while((MAILBOX_CMD_ADDR & 0x01
    cmd=MAILBOX_CMD_ADDR;

    count = (cmd & 0x0000ff00) >>
    if((count==0) || (count>255)){
        // clear bit[0] to indicate
        // set rest to 1s to indic
        MAILBOX_CMD_ADDR = 0xffffff
        return -1;
    }
    for(i=0; i<count; i++) {
        if (cmd & 0x08) // Python
        {
            switch ((cmd & 0x06) >> 1) { // use bit[2:1]
                case 0 : MAILBOX_DATA(i) = *(u8 *) MAILBOX_ADDR; break;
                case 1 : MAILBOX_DATA(i) = *(u16 *) MAILBOX_ADDR; break;
                case 2 : break;
                case 3 : MAILBOX_DATA(i) = *(u32 *) MAILBOX_ADDR; break;
            }
```

```
6c78 3963 7367 3232 3500 6300 0b32
332f 3039 2f33 3000 6400 0931 323a
3a31 3900 6500 0532 7cff ffff ffff
ffff ffff ffff ffff ffaa 9955 6630
0720 0031 a103 8031 413d 0831 6109
c204 0010 9330 e100 cf30 c100 8120
0020 0020 0020 0020 0020 0020 0020
0020 0020 0020 0020 0020 0020 0020
813c c831 8108 8134 2100 0032 0100
e1ff ff33 2100 0533 4100 0433 0101
6100 0032 8100 0032 a100 0032 c100
```

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

```
void setNormalDisplay(){
    sendCommand(OLED_Normal_Display_Cmd);
}

void setInverseDisplay(){
    sendCommand(OLED_Inverse_Display_Cmd);
}

int main(void)
{
    int cmd;
    int Row, Column;

    arduino_init(0,0,0,0);
    config_arduino_switch(A_GPIO, A_GPIO, A_GPIO,
                          A_GPIO, A_SDA, A_SCL,
                          D_GPIO, D_GPIO, D_GPIO, D_GPIO, D_GPIO,
                          D_GPIO, D_GPIO, D_GPIO, D_GPIO,
                          D_GPIO, D_GPIO, D_GPIO, D_GPIO);
    // Initialization
    oled_init();
```

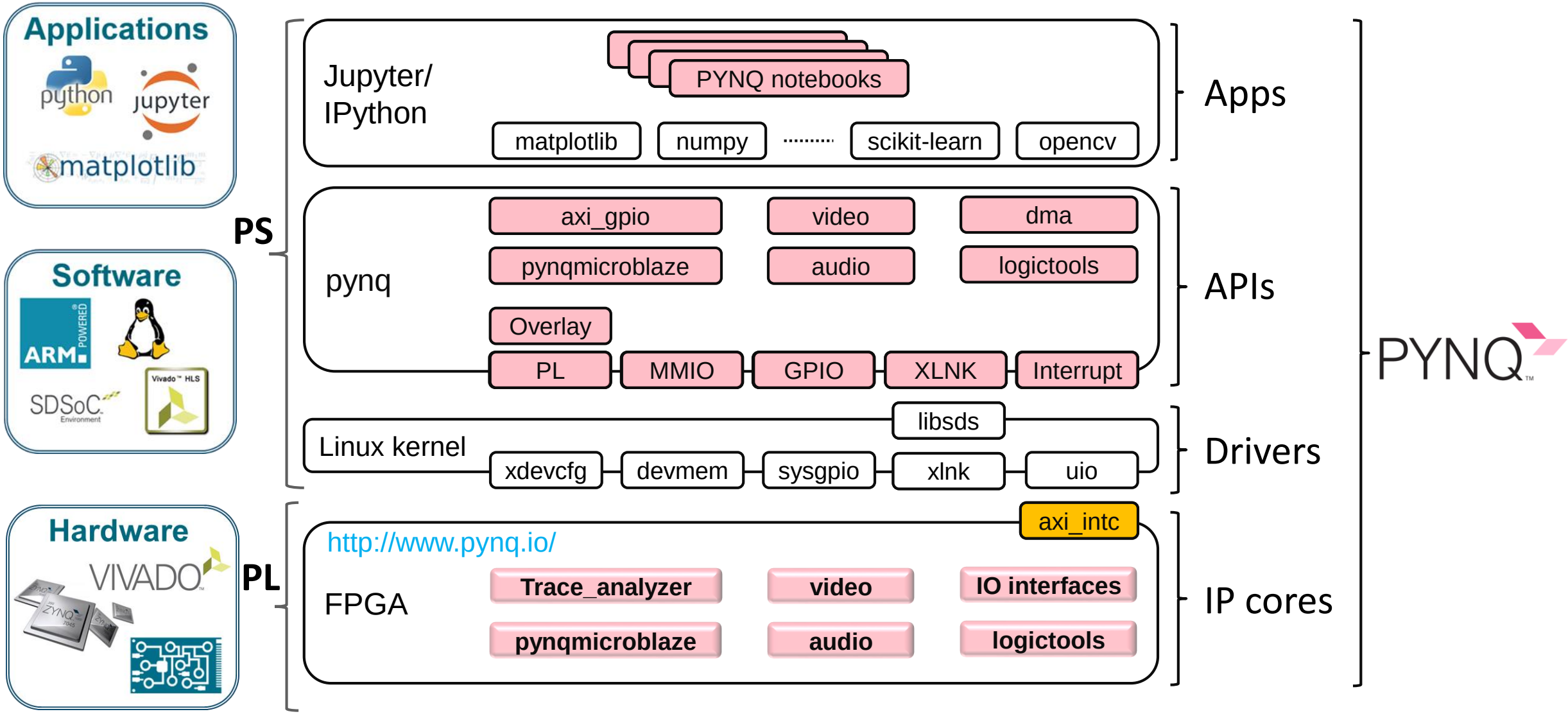
Step 3:
Wrap the C API to create a Python library

```
from time import sleep
from pynq import Overlay
from pynq.iop import PMOD_ADC, PMOD_DAC

ol = Overlay("base.bit")
ol.download()
# Writing values from 0.0V to 2.0V with step 0.1V.
dac_id = int(input("Type in the PMOD ID of the DAC (1 ~ 2): "))
adc_id = int(input("Type in the PMOD ID of the ADC (1 ~ 2): "))

dac = PMOD_DAC(dac_id)
adc = PMOD_ADC(adc_id)

for j in range(20):
    value = 0.1 * j
    dac.write(value)
    sleep(0.5)
#    readings=adc.read(1,0,0)
#    x1=readings[0]
    print("Voltage read by DAC is: {:.4f} Volts".format(adc.read(1,0,0)[0]))
```

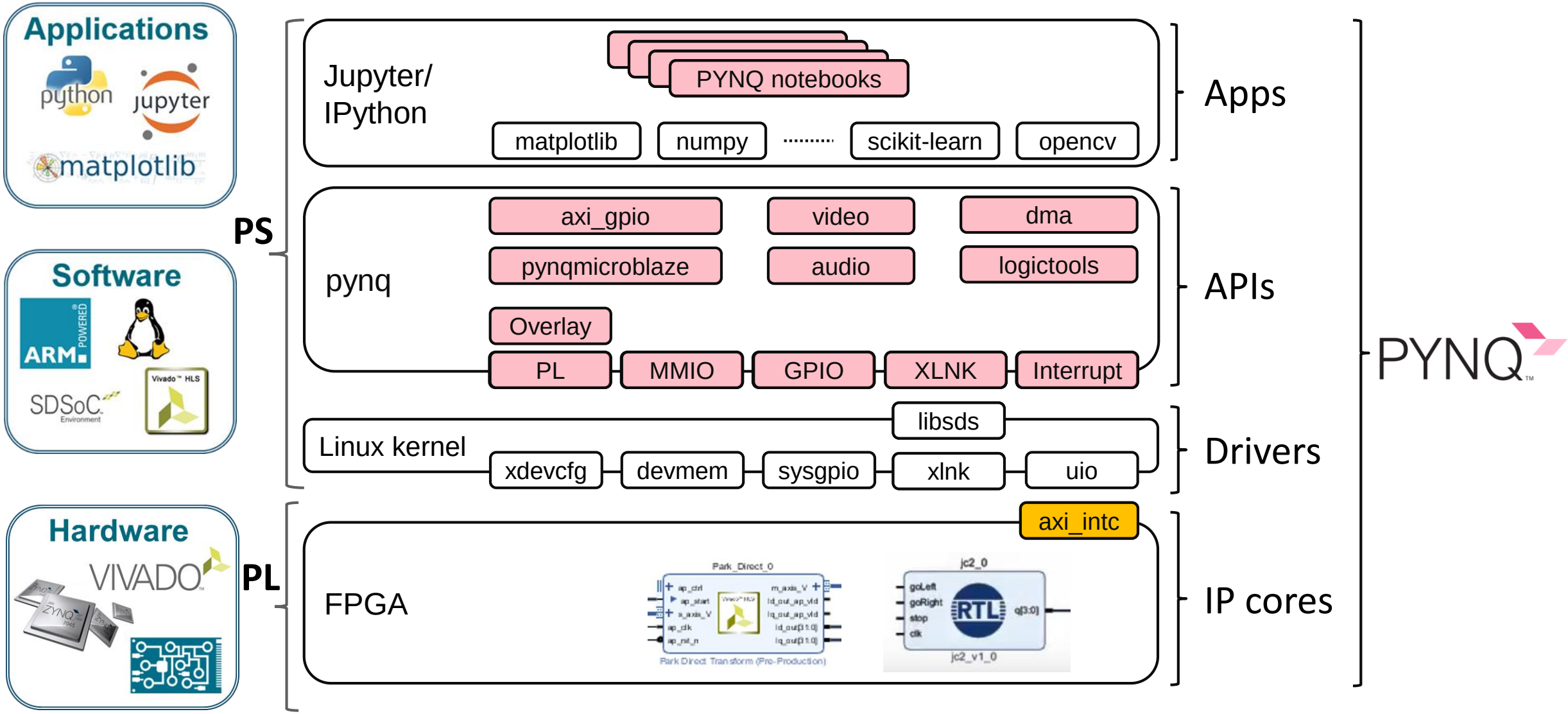
Step 4:
Import the bitstream and the library in your Python scripts and program

XILINX.

Inside PYNQ – *the "base overlay"*



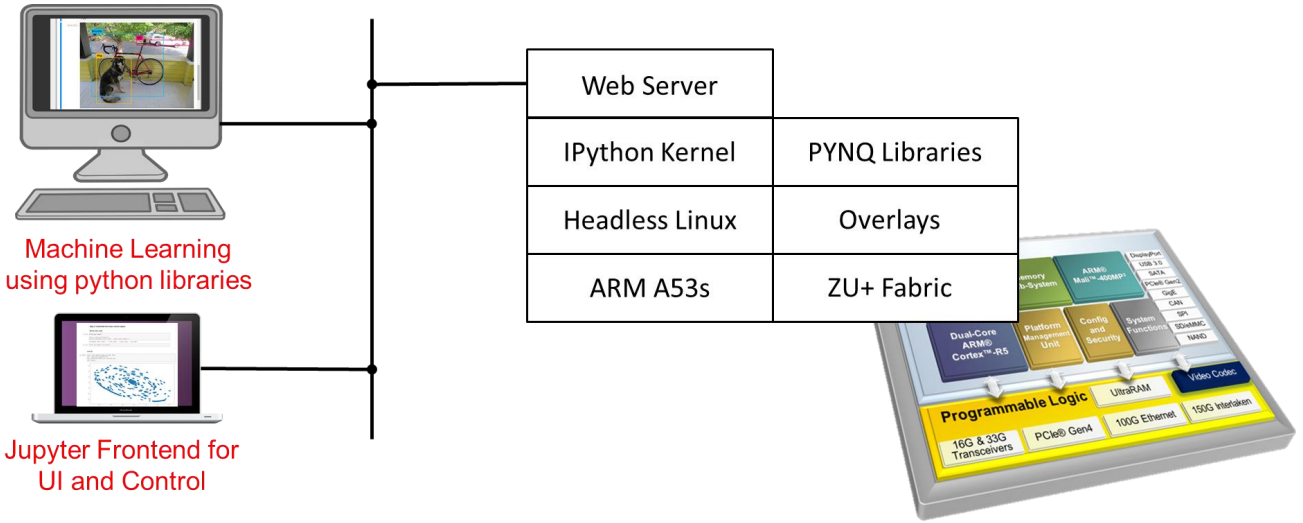
Inside PYNQ – *Your overlay*



PYNQ Summary

- PYNQ is a **Productivity Framework**
- PYNQ includes software, libraries & overlays
- Overlays are FPGA designs that extends user application from processing system to programmable logic
- Jupyter notebooks & python programming for quick exploration
- Developers can create custom overlays & package the project making available **an hardware library to a wider audience**

PYNQ™



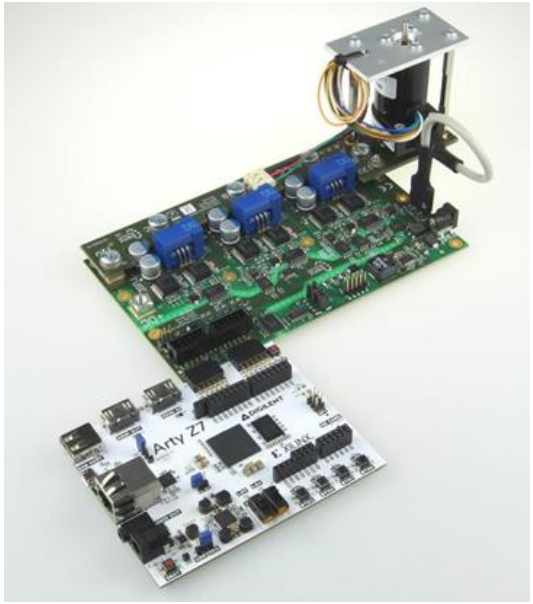
Read the White Paper

XILINX.

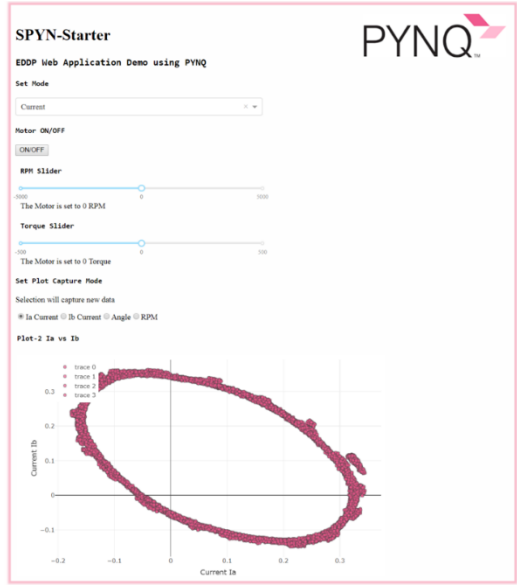
Introduction to the SPYN Open Source Design

XILINX

SPYN: Bridging the two worlds



+ PYNQ™ =



EDDP
Electric Drives Demo Platform

PYNQ
Python Productivity for Zynq

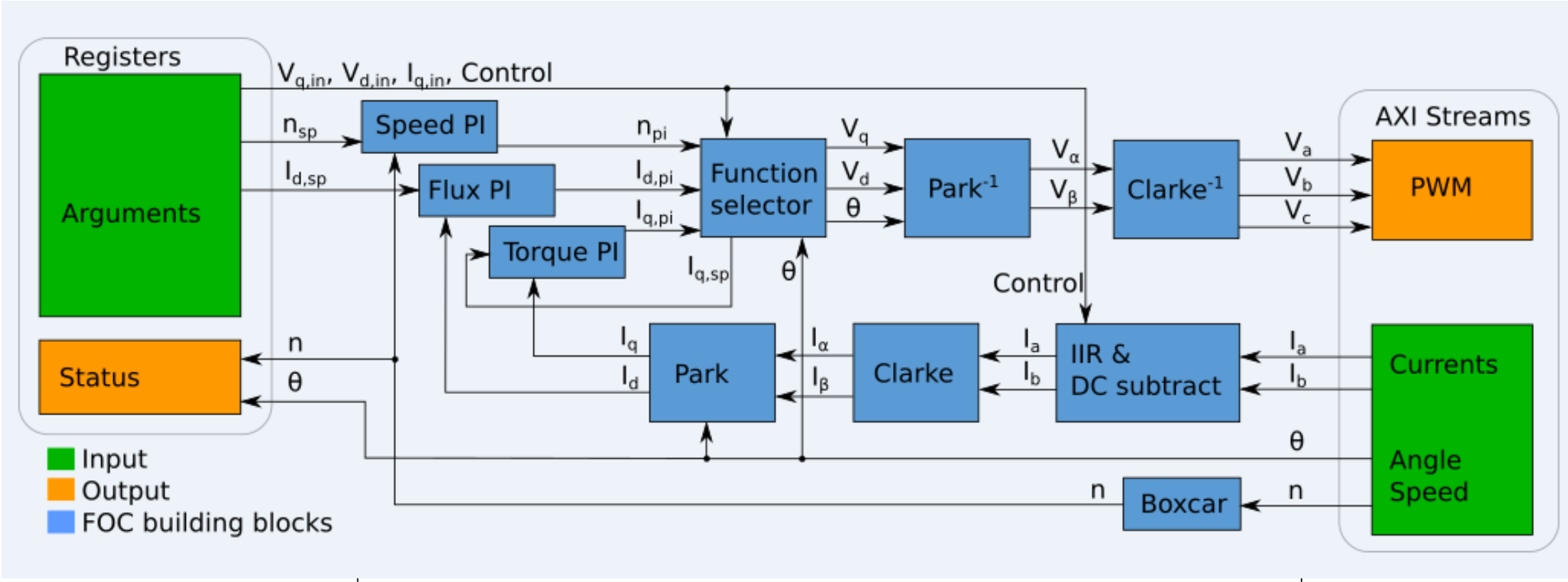
SPYN
Extreme Edge Analytics for Motor Control

- **SPYN takes advantage of both EDDP & the PYNQ framework**
- **The EDDP kit can also be used to test, modify & build the SPYN project at no additional charge!**
- **The solution enables python powered machine learning & edge analytics for motor control**
- **Python libraries are leveraged to provide UI for control, data manipulation, analytics & visualization**

XILINX.

Overlay Algorithm of Motor Field Oriented Control

AXI_Lite



1 Processor interface

7 algorithmic components

2 I/O components

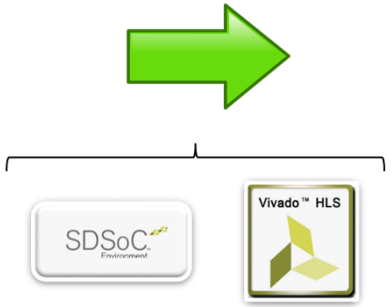
XILINX.

Algorithm translates in C code and compiled in RTL

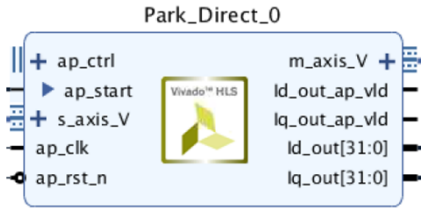
Algorithm in C/C++

```
// See the header file for the documentation.
void Park_Direct(hls::stream<int64_t> &s_axis, hls::stream<int64_t> &m_axis, int32_t *Id_out, int32_t *Iq_out){

#pragma HLS interface axis port=m_axis
#pragma HLS interface axis port=s_axis
    int64_t in_data, res;
    int16_t Ialpha, Ibeta, Theta, RPM;
    int32_t Id, Iq;
    int32_t cos_theta, sin_theta;
    int32_t Ia_cos, Ib_sin, Ib_cos, Ia_sin;

    // Decode Input stream
    in_data = s_axis.read();                    // Read one value from AXI4-Stream
    Ialpha = int16_t(in_data & 0xFFFF);            // Extract Ialpha - bits[15..0] from input stream
    Ibeta = int16_t((in_data >> 16) & 0xFFFF);     // Extract Ibeta - bits[32..16] from input stream
    RPM = int16_t((in_data >> 32) & 0xFFFF);       // Extract RPM - bits[47..32] from input stream
    Theta = int16_t((in_data >> 48) & 0xFFFF);     // Extract Angle - bits[63..48] from input stream

    // Process data
    cos_theta = (int32_t)cos_table[Theta];
    sin_theta = (int32_t)sin_table[Theta];
    Ia_cos = (int32_t)Ialpha * cos_theta;
    Ib_sin = (int32_t)Ibeta * sin_theta;
    Ib_cos = (int32_t)Ibeta * cos_theta;
    Ia_sin = (int32_t)Ialpha * sin_theta;
    Id = (Ia_cos + Ib_sin) >> 15;
    Iq = (Ib_cos - Ia_sin) >> 15;
    Id = (Id > MAX_LIM) ? MAX_LIM : Id;            // Clip max
    Id = (Id < MIN_LIM) ? MIN_LIM : Id;            // Clip min
    Iq = (Iq > MAX_LIM) ? MAX_LIM : Iq;            // Clip max
    Iq = (Iq < MIN_LIM) ? MIN_LIM : Iq;            // Clip min

    *Id_out = Id;
    *Iq_out = Iq;
    // Write output stream
    res =   (((int64_t)Theta << 48) & 0xFFFF000000000000) | // Put Angle bits[63:48]
            (((int64_t)RPM << 32)   & 0x0000FFFF00000000) | // Put RPM bits[47:32]
            (((int64_t)Iq << 16)    & 0x00000000FFFF0000) | // Put Iq bits[31:16]
            ( (int64_t)Id           & 0x000000000000FFFF);  // Put Id bits[15:0]
    m_axis.write(res);                          // Write result to the output stream
}
```

Xilinx Tools



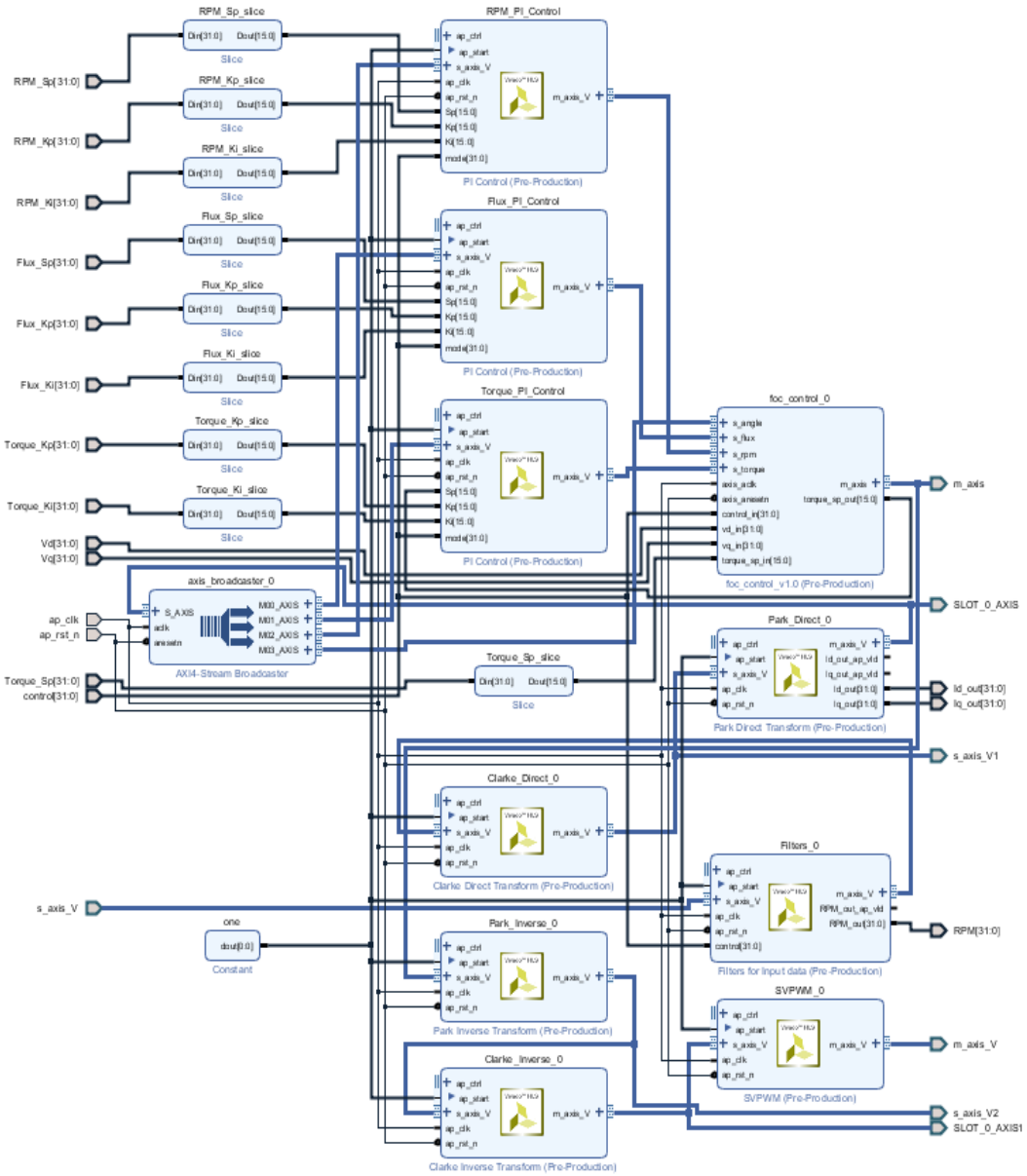
RTL Design

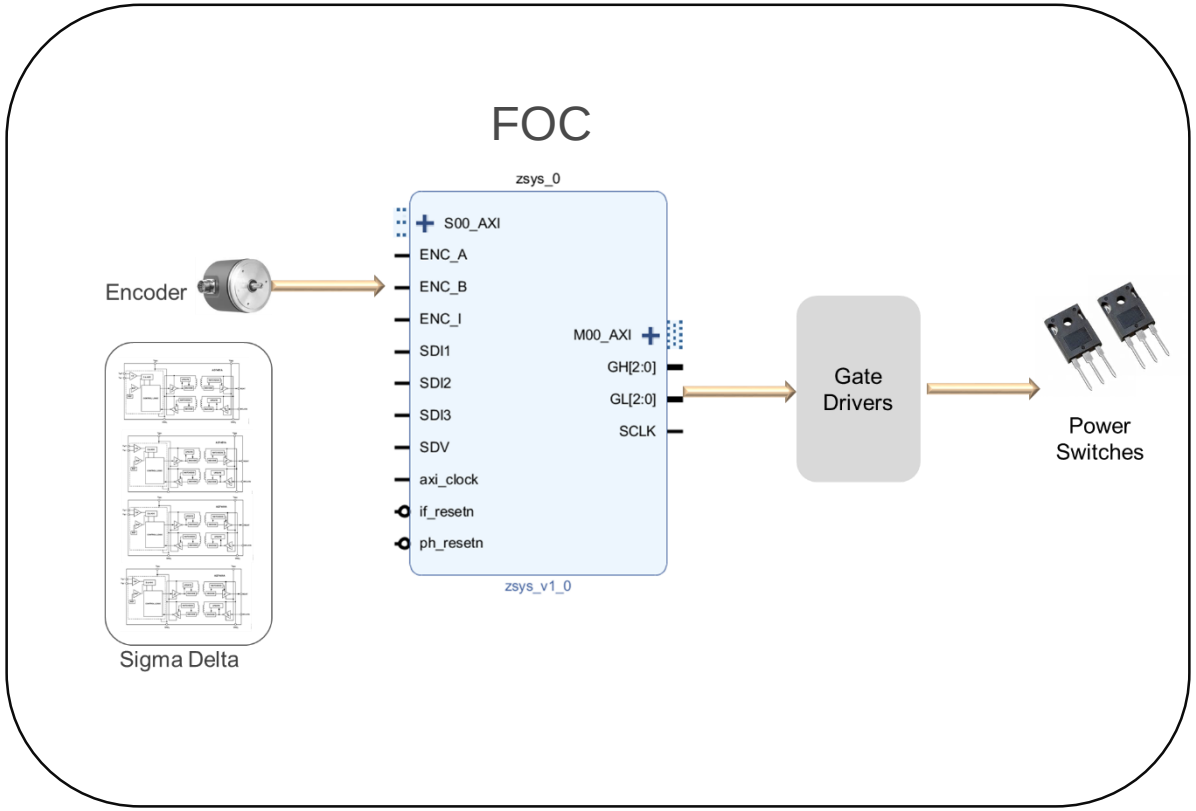


- **Tools:** SDSoC and Vivado HLS enabled motor control algorithm written in c code to be made into hardware implementation

XILINX.

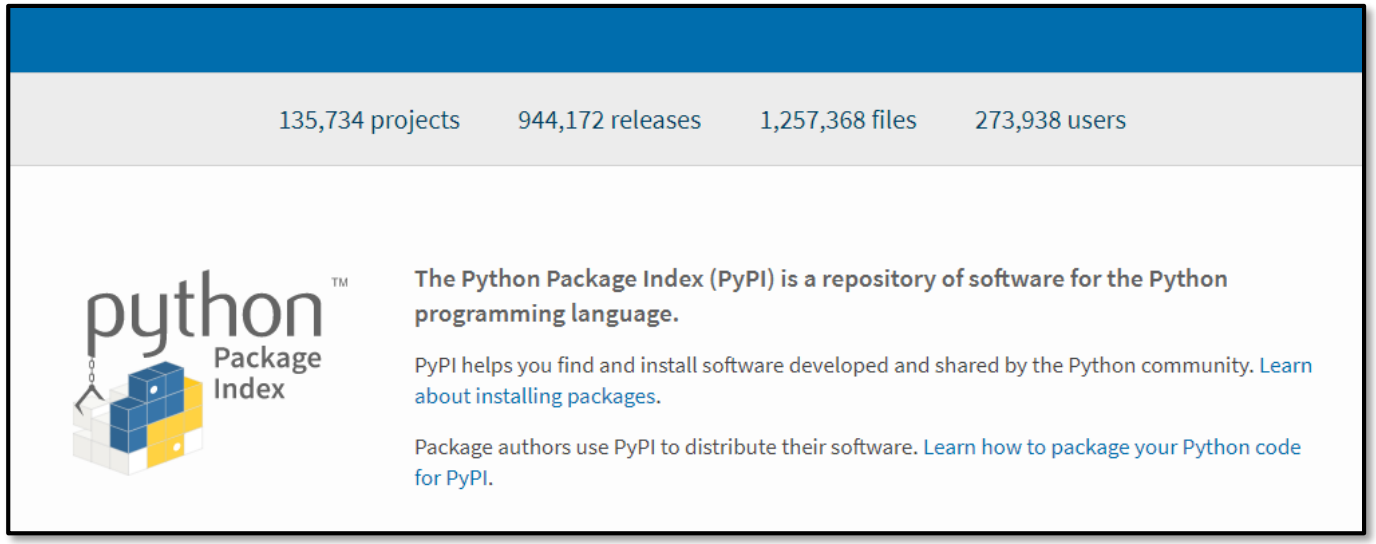
Whole FOC IP – is then an Overlay library component





XILINX.

Packaging the whole project as *pip*



- **Python Packaging:**
  - Python projects/libraries can be easily packaged
  - Packages are installable with single command

```
$ sudo pip3.6 install --upgrade git+https://github.com/Xilinx/IIoT-SPYN.git
```

  - Enable wider distribution & upgradability

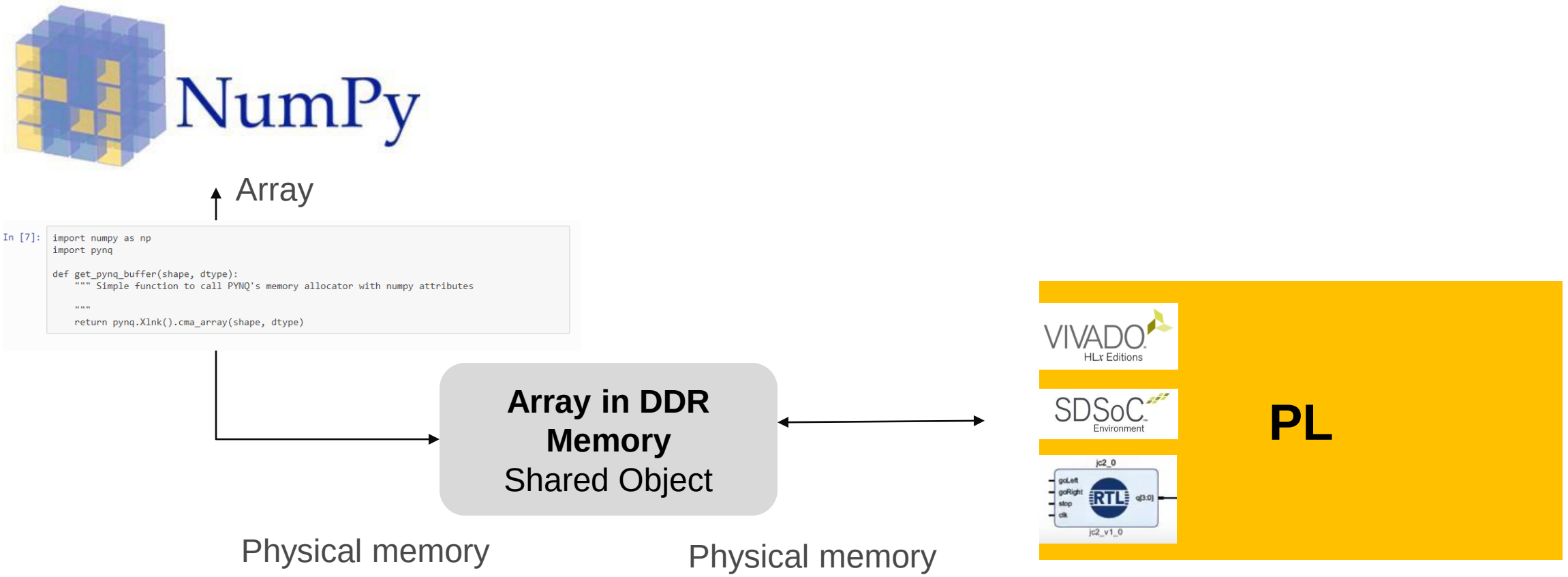
- **Ecosystem Advantages:**
  - Open Source
  - 135K+ installable packages publicly available

What is NumPy

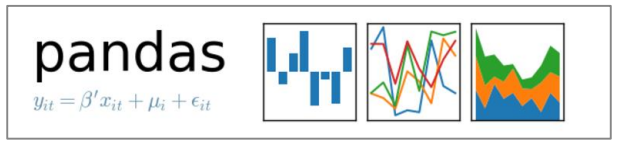
- NumPy is the fundamental package for scientific computing with Python. It contains among other things:
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number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

- 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.

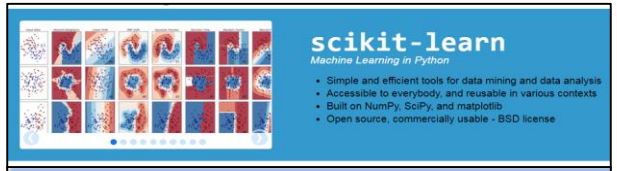
NumPy and interaction with programmable logic



Connect other Python Libraries -

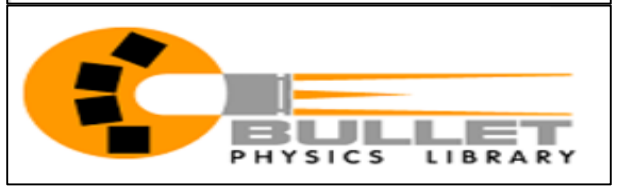


data analysis



Statistical machine Learning

data analysis



sensors

robots

discrete event simulation

Quick Prototyping

python

PYNQ™

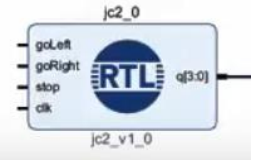
Embedded Acceleration

C++

SDSoC™ Environment

VIVADO HL*x* Editions

RTL Acceleration



OVERLAYS

XILINX.

The design pattern MCVD

model – control – view - deployment

XILINX

SPYN Design Enhanced Capabilities (1 of 3)

Monitoring:

```
motor_status = [(motor._read_controlreg(i + ANGLE.offset)) for i in
                range(0, 16, 4)]
high_sp, low_sp = bytesplit(motor_status[1])
high_id, low_id = bytesplit(motor_status[2])
high_iq, low_iq = bytesplit(motor_status[3])

print(f'Angle in degrees : {motor_status[0] * 0.36}')
print(f'Angle in steps per thousand: {(motor_status[0])}')
print(f'Id : {np.int16(low_id) * 0.00039} Amp')
print(f'Iq : {np.int16(low_iq) * 0.00039} Amp')
print(f'Speed in RPM : {-(np.int16(low_sp))}')
```

```
Angle in degrees : 104.39999999999999
Angle in steps per thousand: 290
Id : -0.062009999999999996 Amp
Iq : 0.31785 Amp
Speed in RPM : 983
```

Monitor the status parameters of the motor remotely anytime

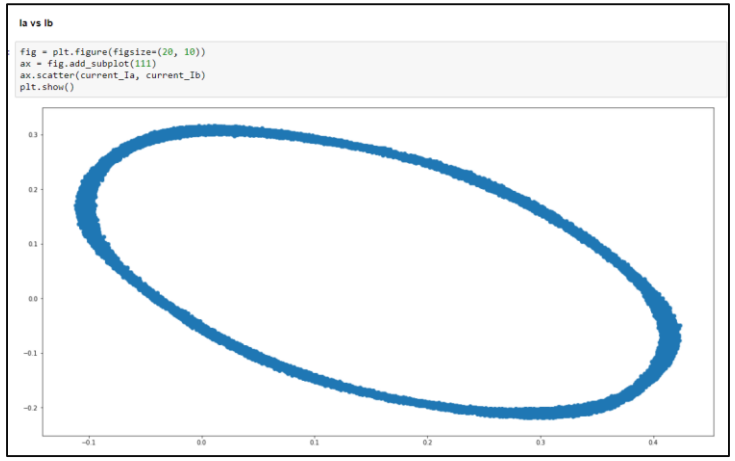
Control:

```
import time
motor.set_mode('rpm_mode')
for i in range(2):
        motor.set_rpm(1000)
        time.sleep(1)
        motor.set_rpm(0)
        time.sleep(2)
        motor.set_rpm(-50)
        time.sleep(2)
        motor.set_rpm(0)
        time.sleep(2)
motor.stop()
```

Simple python code routine to control motor

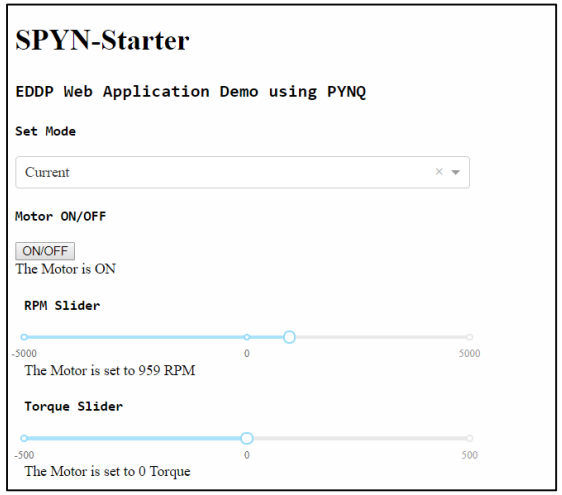
SPYN Design Enhanced Capabilities (2 of 3)

Visualization:



Leverage matplotlib library to draw plots to visualize motor data

Custom Interactive UI:



Custom UI to control motor & generate plots using dash by plotly

SPYN Design Enhanced Capabilities (3 of 3)

Analytics & ML Ready Data Format:

```
import pandas as pd

data = {'Ia' : current_Ia,
        'Ib' : current_Ib,
       'angle':cap_list[3],
       'rpm':  cap_list[2]}

df = pd.DataFrame(data, columns = ['Ia', 'Ib', 'angle', 'rpm'])
df
```

	Ia	Ib	angle	rpm
0	0.09204	0.09009	846	3541
1	0.09048	0.08853	845	3541
2	0.08619	0.09126	845	3527
3	0.08424	0.09321	845	3527
4	0.08307	0.09945	844	3527
5	0.08502	0.09789	844	3527
6	0.08697	0.09711	843	3527
7	0.09126	0.09594	843	3527
8	0.09360	0.09204	843	3527
9	0.08892	0.09282	842	3527
10	0.08463	0.09204	842	3527

Acquired Motor Data is stored as pandas data frames providing training data for ML & Analytics

Installable & Upgradable packaging :

```
root@kv_pynq:/home/xilinx# sudo pip3.6 install --upgrade git+https://github.com/Xilinx/IIoT-SPYN.git
Collecting git+https://github.com/Xilinx/IIoT-SPYN.git
  Cloning https://github.com/Xilinx/IIoT-SPYN.git to /tmp/pip-ge20fs95-build
Installing collected packages: spyn
  Found existing installation: spyn 1.0
    Uninstalling spyn-1.0:
      Successfully uninstalled spyn-1.0
  Running setup.py install for spyn ... done
Successfully installed spyn-1.0
root@kv_pynq:/home/xilinx#
```

SPYN is made into a installable package & enables remote upgradability

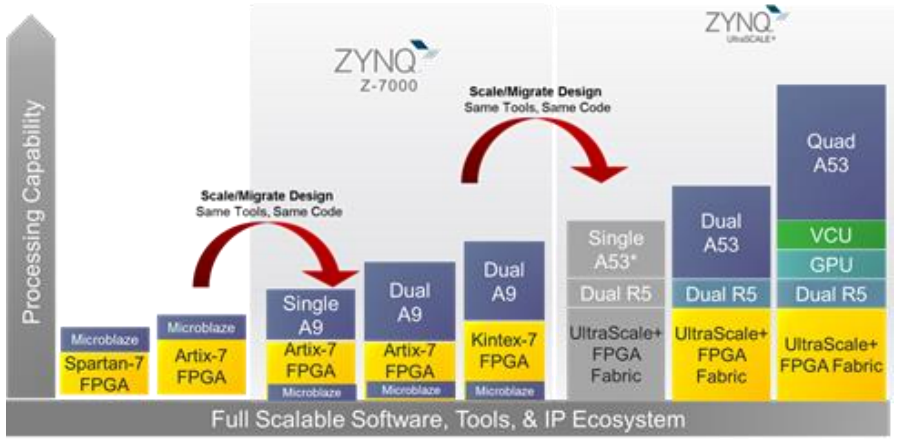
Beyond Prototyping into Production

- **Real-time:** Control implemented in programmable logic**,** 30-40x higher performance than alternatives, determinism with non-interference

- **Scalability:** Common processing systems and tool flows with varying amounts of programmable logic for custom accelerators

- **Built for Real World:** 15+ years Lifecycle, World Class Quality

- **Cost Savings through Integration:** Networking, Security, Safety, Intelligence, Sensor Connectivity, Custom Functions in single chip

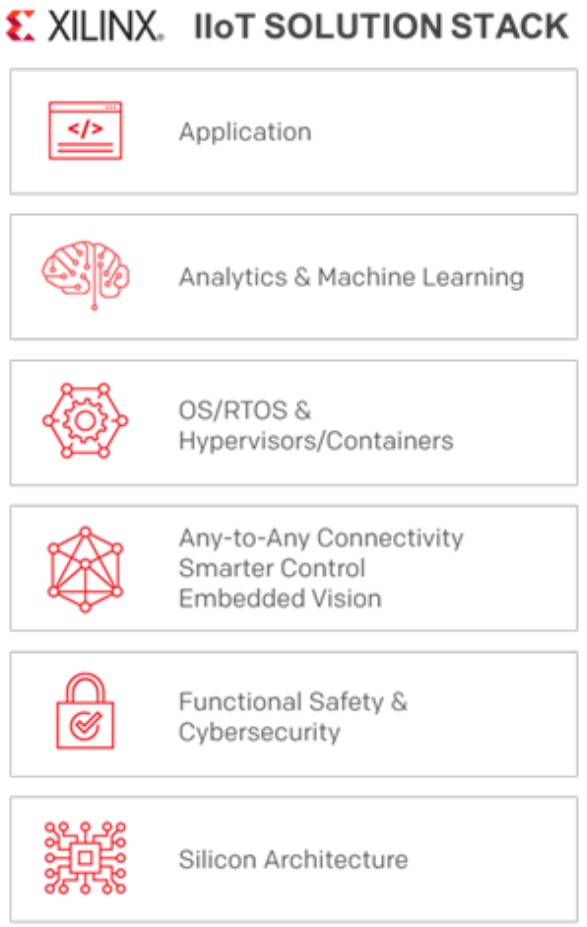


XILINX.

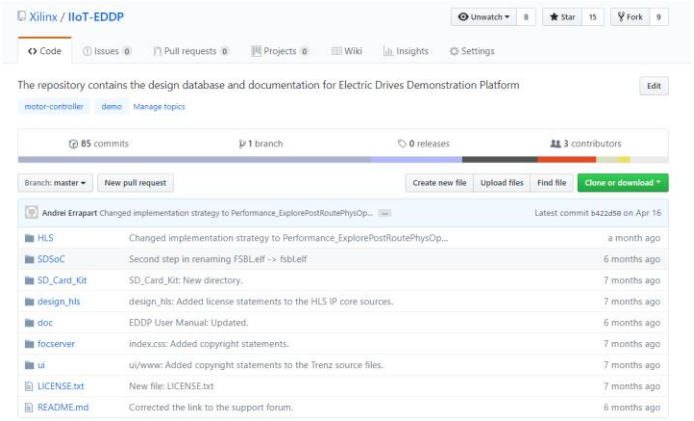
SPYN Summary

Summary

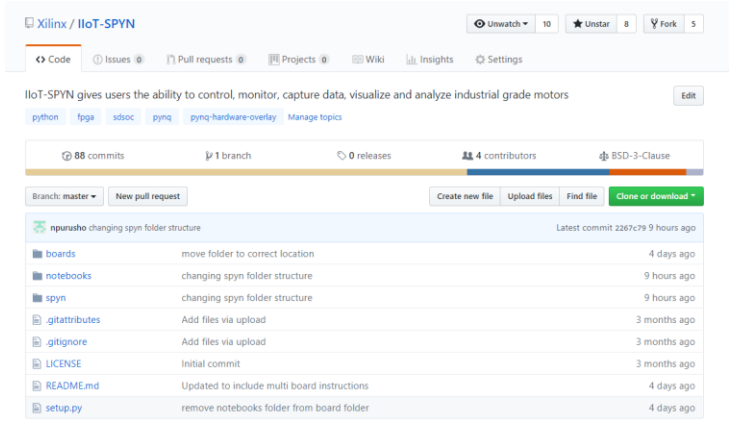
- **In Industrie 4.0 / IIoT, Electric Drives are expected to do more**
- **Xilinx IIoT Solution Stack:** Xilinx and Ecosystem building blocks and solutions used across Industrial IoT platforms
- **EDDP:** A learning tool for implementing Xilinx designs with C/C++
- **PYNQ:** A hardware-software framework to bring python productivity to ZYNQ and enabling edge analytics for Industrial IoT
- **SPYN:** Example design highlighting Python Powered Control, Edge Analytics & Machine Learning for Electric Drives



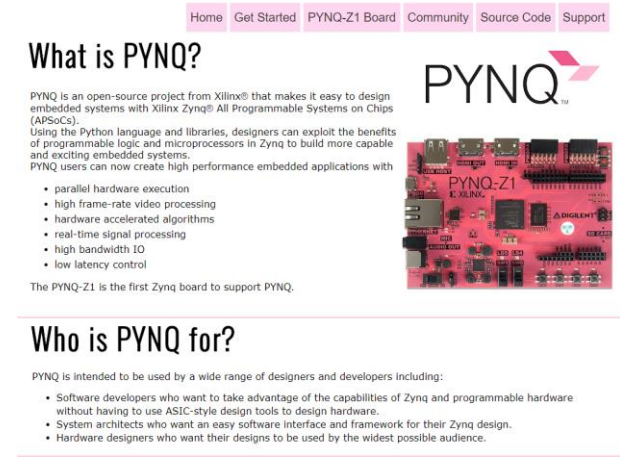
Next Steps: Explore Resources



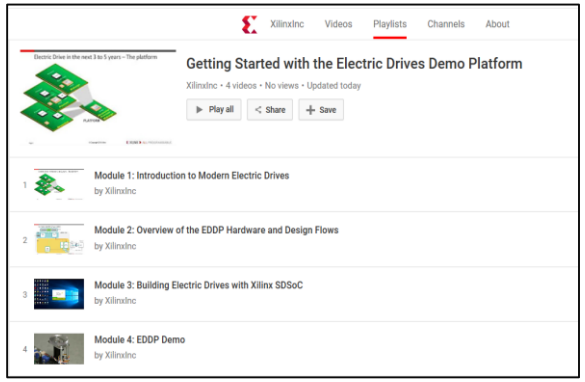
EDDP GitHub
https://github.com/Xilinx/IIoT-EDDP



SPYN GitHub
https://github.com/Xilinx/IIoT-SPYN



PYNQ GitHub
https://github.com/Xilinx/pynq

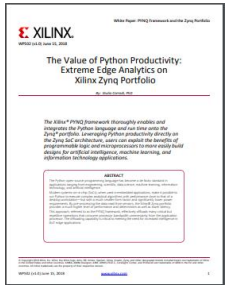


YouTube Videos:
Getting Started with the Electric Drives Demo
Xilinx.com Videos:
Available in English (xilinx.com)
Chinese (china.xilinx.com)
Japanese (japan.xilinx.com)

NEW! SPYN Video on YouTube
NEW! SPYN Video on Xilinx.com

Hardware Kit for EDDP and SPYN

trenz electronic



NEW! White Paper

XILINX

Adaptable.
Intelligent.

XILINX