ADIUVO

ENGINEERING AND TRAINING, LTD.

Session Four

How do FPGA do Math?

Why do we need to do math in FPGA

Programmable Logic enables accelerated applications

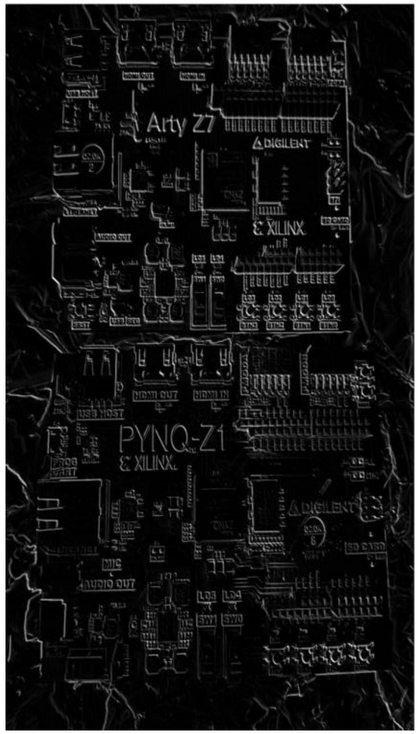
- Image Processing – Noise removal, edge detection, filtering
- Radar Processing – Signal generation, reply processing
- Signal Processing – Signal filtering & manipulation
- Robotics – End Effector positioning, Navigation
- Control Systems – Kalman Filtering, PID Loops
- Motor Control – Servo Motor, DC Motor Control

All these applications require the ability of FPGA to do Maths and implement algorithms.

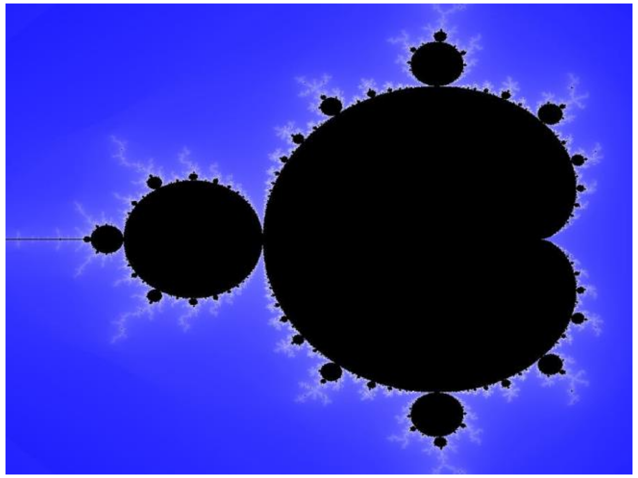
Example applications

Background Removal & Substitution

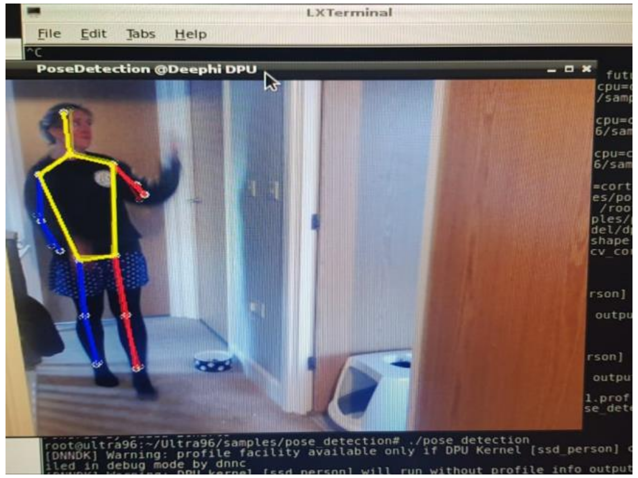
Edge Detection



Fractals

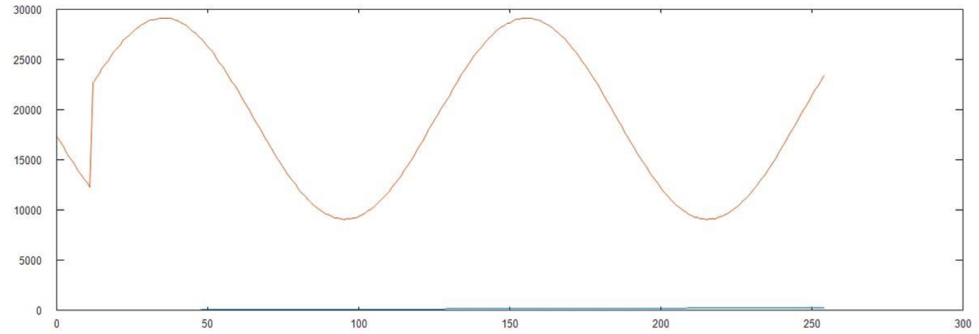


AI/ML



```
^C
root@ultra96:~/Ultra96/samples/pose_detection# ./pose_detection
[DNNDK] Warning: profile facility available only if DPU Kernel [ssd_person] c
iled in debug mode by dnnc
```

Signal Processing

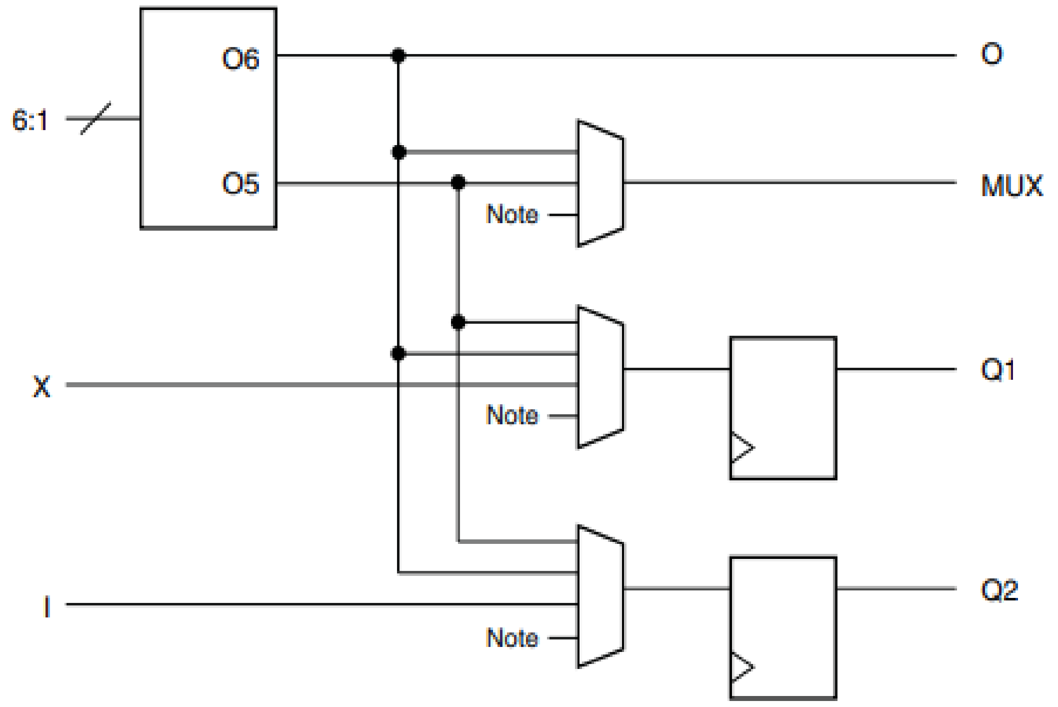


FPGA Architecture

FPGA are register and logic rich

- Configurable Logic Blocks contain
  - Registers
  - Look Up Table
  - Distributed RAM
  - Carry Mux

Logic resources are the basic building blocks of our algorithms. It is where we implement our mathematical algorithms



DSP Elements

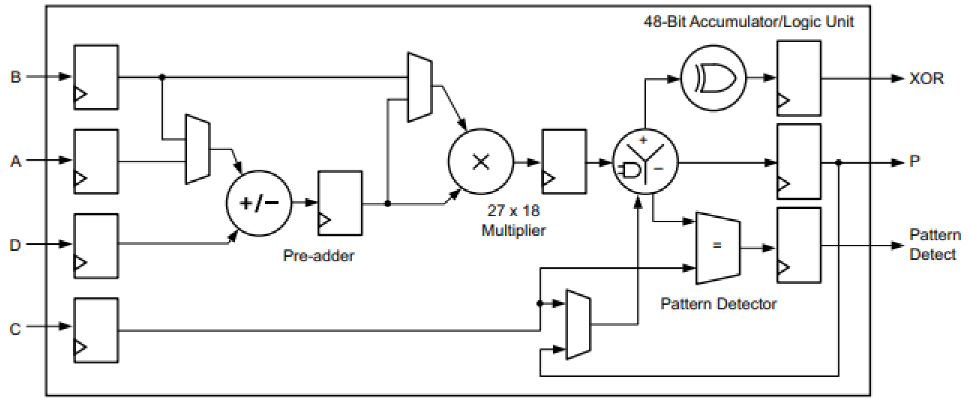
Implementing math directly in logic is costly

- Resources – increased resources
- Performance – reduced performance

Manufactures to address this include dedicated DSP elements e.g. DSP48

Capable of doing 48 bit Multiply accumulator

- Pre-Adder
- Multiply
- Accumulator
- Can do advance things SIMD – More later!



FPGA Math's

So far, we have looked at Logic & DSP elements, but they all have one thing in common ?

They are all ideal for the implementation of fixed-point solutions.

What is the difference between fixed point and floating-point numbers ?

Fixed Point Number

Fixed-point representation maintains the decimal point within a fixed position allowing for straight forward arithmetic operations.

The major drawback of fixed-point representation is that to represent larger numbers or to achieve a more accurate result with fractional numbers, a larger number of bits are required.

A fixed-point number consists of two parts called the integer and fractional parts.

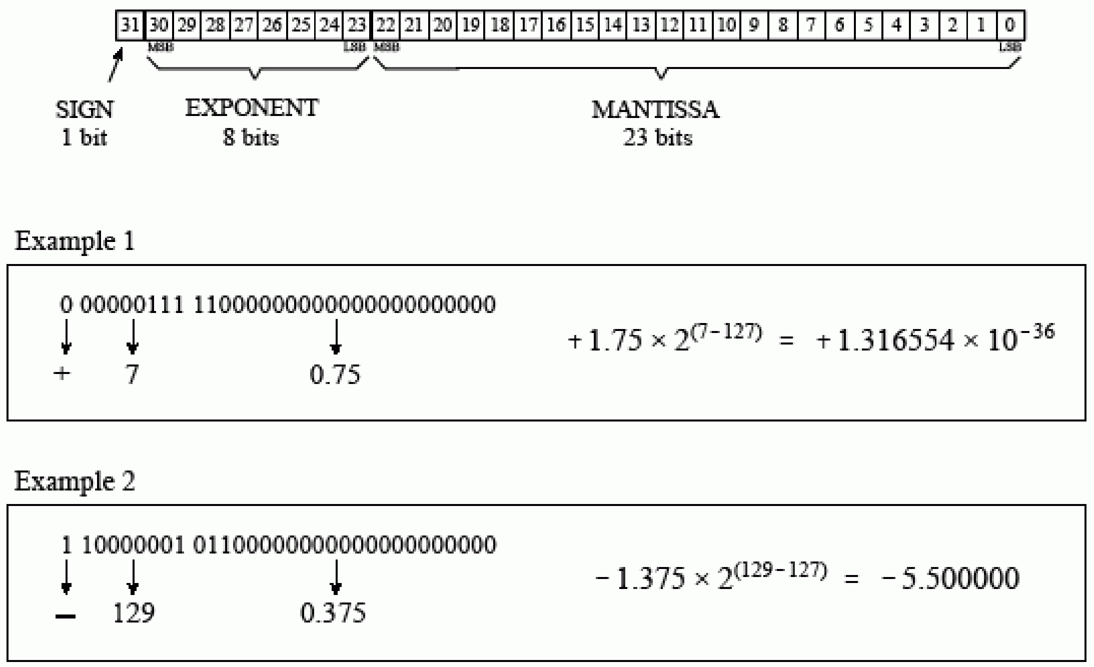
But in programmable logic Fixed Point Maths can be very fast

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	●	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}

Floating Point Number

Floating point representation allows the decimal point to float to different places within the number depending upon the magnitude.

The floating-point number is standardized by an IEEE / ANSI Standard 754-1985 the basic IEEE floating point number



$+1.75 \times 2^{(7-127)} = +1.316554 \times 10^{-36}$

$-1.375 \times 2^{(129-127)} = -5.500000$

FIGURE 4-2
Single precision floating point storage format. The 32 bits are broken into three separate parts, the sign bit, the exponent and the mantissa. Equations 4-1 and 4-2 shows how the represented number is found from these three parts. MSB and LSB refer to "most significant bit" and "least significant bit," respectively.

Why Fixed Point

- Less complex to implement in logic
- Enables a faster solution
- Can be more power efficient
- FPGA are register rich!
- No standard for implementation but, Q format is popular
    - Q15 – 15 fractional bits
    - M,N – M integer bits and N fractional bits

Number schemes

Fixed point numbers need to represent positive and negative numbers

- Sign and Magnitude - Utilises the left most bit to represent the sign of the number (0 = positive, 1 = negative) the remainder of the bits represent the magnitude. BUT! Positive and Negative Numbers

- Ones Complement – Same unsigned representation for positive numbers as Sign and Magnitude representation. However, for negative numbers the inversion (ones complement) of the positive number are used. Requires end around carry for subtraction – Added complexity.

- Twos Complement – Positive are represented in the same manner as an unsigned numbers. While negative numbers are represented as the binary number you add to a positive number of the same magnitude to get zero

Twos Compliment

A negative twos complement number is calculated by first taking the ones complement (inversion) of the positive number and then adding one to it. The twos complement number system allows subtraction of one number form another by performing an addition of the two numbers. The range a twos complement number can represent is given by

- (2n-1) to + (2n-1 – 1)

One method we can use to convert a number to its twos complement format is to work right to left leaving the number the same until the first one is encountered, after this each bit is inverted.

ADIUVO
ENGINEERING AND TRAINING, LTD.

Fixed Point

How many bits do we need to represent my value ?

$$\text{Integer Bits Required} = Ceil\left(\frac{LOG_{10}\,\text{Integer\_Maximum}}{LOG_{10}\,2}\right)$$

For example, the number of integer bits required to represent a value between 0.0 and 423.0

$$9 = Ceil\left(\frac{\text{LOG}_{10}\,423}{\text{LOG}_{10}\,2}\right)$$

Fixed Point

How do we work out fractional bit ? Trade off between bit length and accuracy

To store the number 1.45309806319x10-4

Multiply by 2^16 1.45309806319x10-4 * 65536 = 9.523023

Can only store 9 in the FPGA registers.

9/65536 = 1.37329101563x10-4

Significant loss of accuracy, how can we address this?

Fixed Point

We can obtain a more accurate result by scaling the number up by a factor of 2 that produces a result of between 32768 and 65535 therefore still allowing storage in a 16-bit number

268435456 * 1.45309806319x10-4 = 39006.3041205

Stored number therefore 1.45308673382x10-4 (39006/ 268435456)

Number is formatted as Q28 or 1,28

Fixed Point Rules

Fixed Point Arithmetic does have some rules which must be followed.

- Addition - Decimal points must be aligned
- Subtraction – Decimal points must be aligned
- Division – Decimal Points must be aligned
- Multiplication – Decimal points do not need to be aligned

Fixed Point Result Sizes

Operation	
A + B	Max(A’left, B’left) + 1 downto Min (A’right, B’right)
A - B	Max(A’left, B’left) + 1 downto Min (A’right, B’right)
A * B	(A’left + B’left) + 1 downto A’right + B’right
A / B - Unsigned	A’left - B’left downto (A’right + B’right) -1
A / B - Signed	(A’left - B’left) +1 downto A’right + B’right

Implementing in VHDL

Two options

- Numeric Standard (pre VHDL 2008)
    - Unsigned
    - Signed
    - Need to keep track of decimal point – Range of number from X downto 0
    - Quantisation needs to be performed by developer
    - No in-built checking, requires design to check correct sizing / overflow etc

- Fixed Point (VHDL 2008)
    - Ufixed
    - Sfixed
    - Decimal point located between the 0 and -1 bit
    - Inbuilt checking to ensure correct sizing of results

Fixed Package

Integer bits are represented in the range MSB down to 0

Fractional bits are represented in the range -1 down to LSB

SIGNAL example : ufixed(3 DOWNTO -3);

Which represents the vector of 000.000 allowing for a range of 0.0 to 7.875

To help initialise signals, variables and constants in our algorithm we can use the to_ufixed and to_sfixed, these can be used with integers, real, ufixed, sfixed and std_logic_vectors.

What about more complex math

How would I implement the following functions

- Sine / Cosine / ArcTan
- SineH / CosH / ArcTanH
- Square Root
- Exponential
- Ln

Taylor / Maclaurin Series? Look Up Table ?

But how do we achieve performance ?

CORDIC Algorithm

CORDIC (COordinate Rotation DIgital Computer) algorithm invented by Jack Volder for B58 Program

Deployed in first scientific calculator HP35

Shift and Add algorithm which can be used to implement transcendental functions.

No dedicated Multiplier required.

CORDIC Algorithm

Three configurations - Linear / Hyperbolic / Circular

Each mode has two modes – Rotation / Vectoring

Enable a range of complex math's functions to be implemented.

Configuration	Rotation	Vectoring
Linear	Op Y = X * Y	Op Z = X / Y
Hyperbolic	Op X = CosH(X) Op Y = SinH(Y)	Op Z = ArcTanH
Circular	Op X = Cos(X) Op Y = Sin(Y)	Op Z = ArcTan(Y) Op X = SQR($X^2 + Y^2$)

Additional Functions

Tan = Sin / Cos

TanH = Sinh / Cosh

Exponential = Sinh + Cosh

Natural Logarithm = 2 * ArcTanH note 1

SQR = $(X^2 – Y^2)^{0.5}$

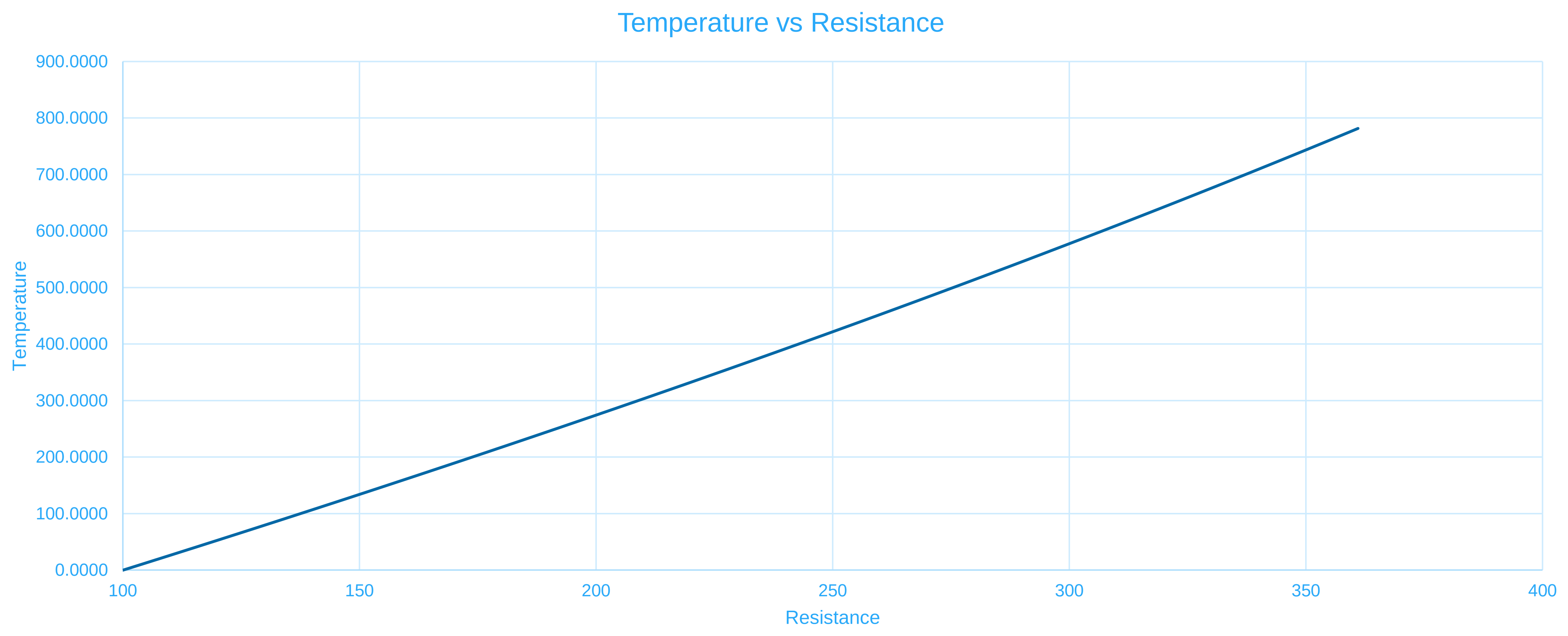
More Complex Algorithm

How would you implement the following algorithm

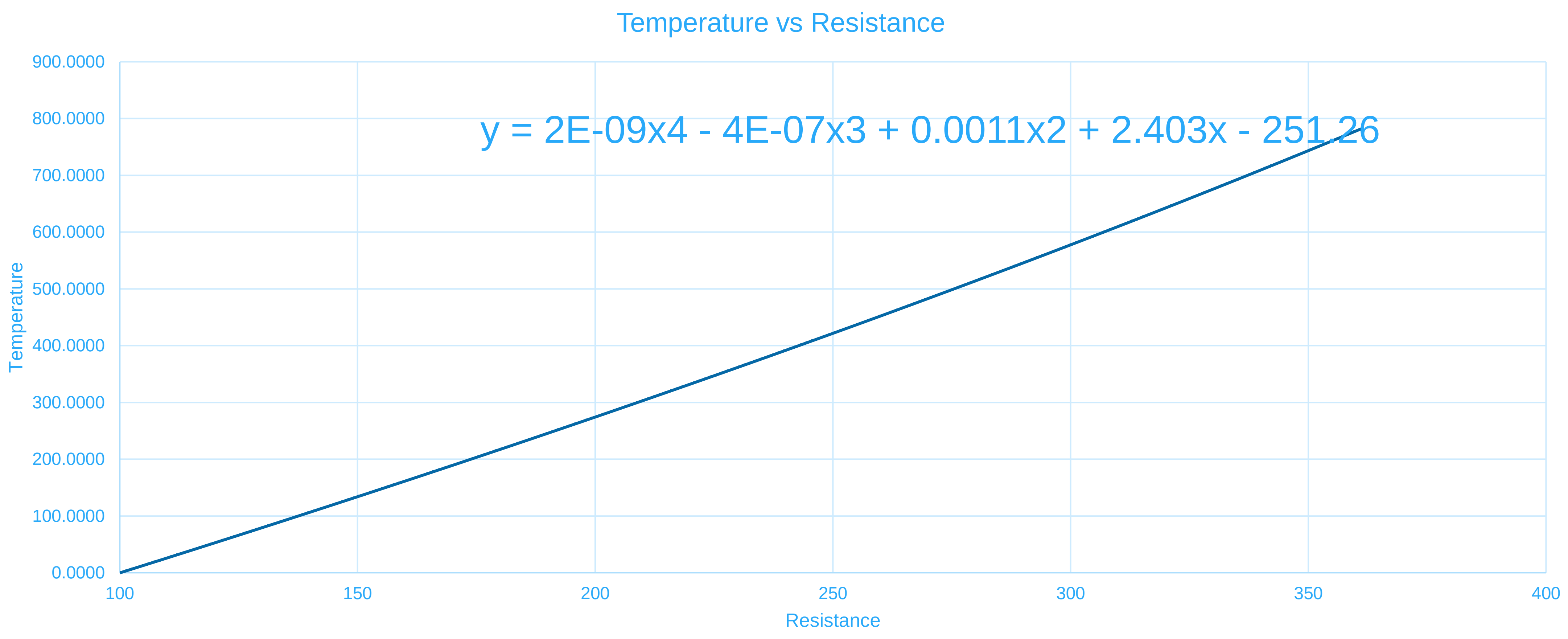
$$t = \frac{-R_0 \times a + \sqrt{R_0^2 \times a^2 - 4 \times R_0 \times b \times (R_0 - R)}}{2 \times R_0 \times b}$$

Typical Platinum Resistance Thermometer conversion equation used in industrial applications

More Complex Algorithm



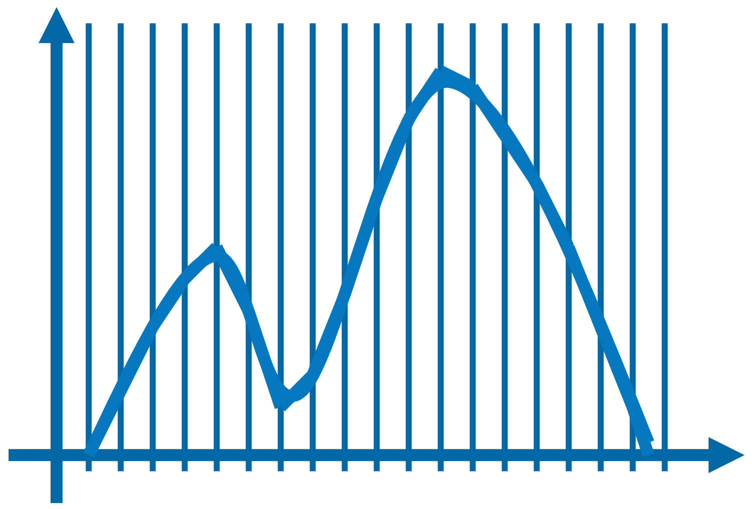
Polynomial Approximation



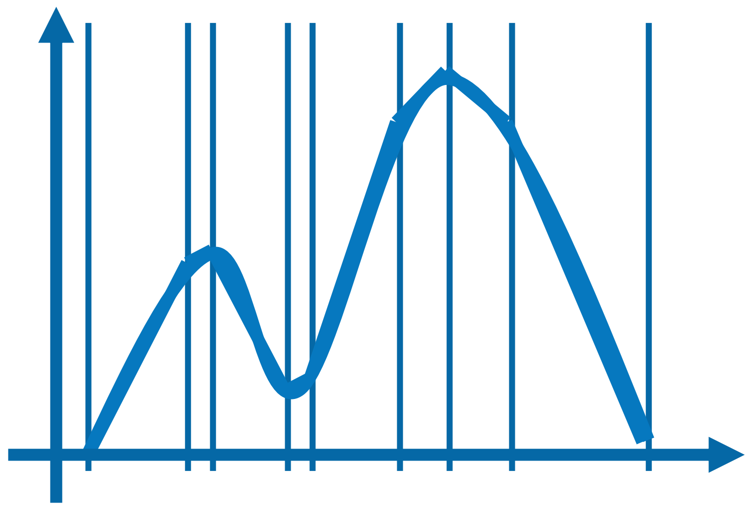
Polynomial Approximation

Leverage FPGA DSP rich environment – Addition and Multiplication easy to do in FPGA especially as we now know how!

If Accuracy is difficult with one overall polynomial equation – Segment it to several elements



Uniform segmentation

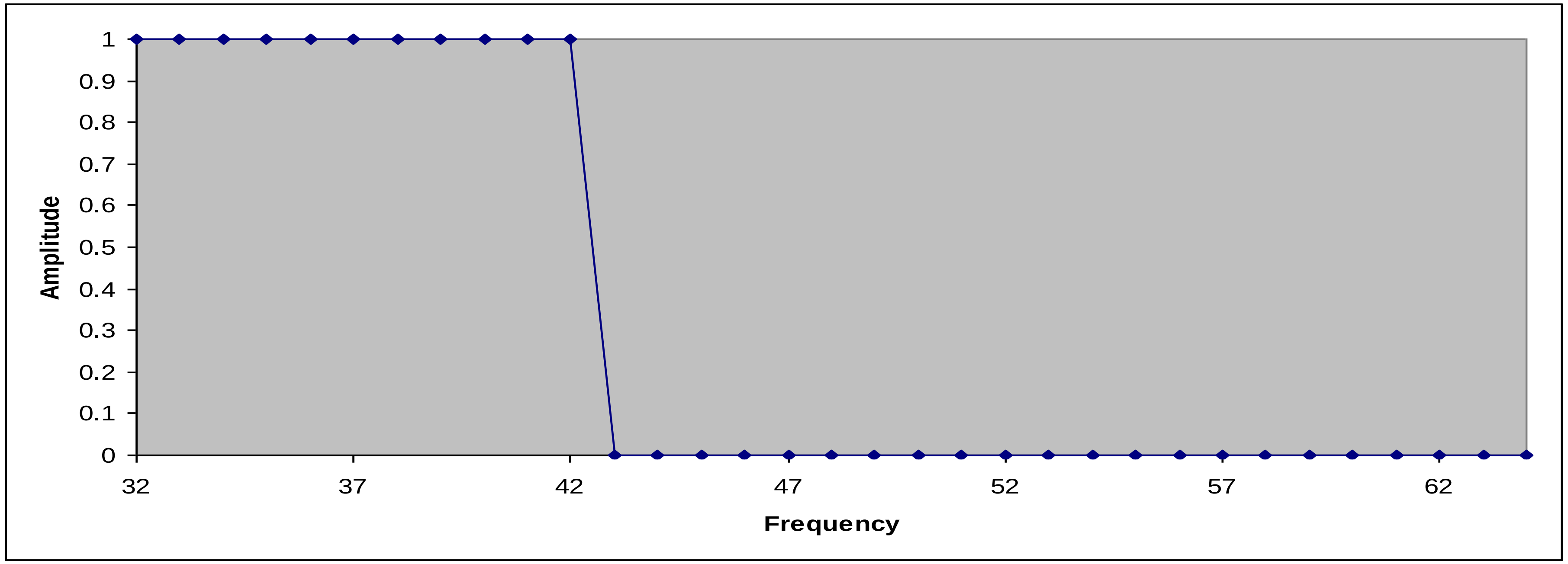


Non-uniform segmentation

What about signal processing

Finite Impulse Response Filters – Leverage the Multiple Accumulate Capability

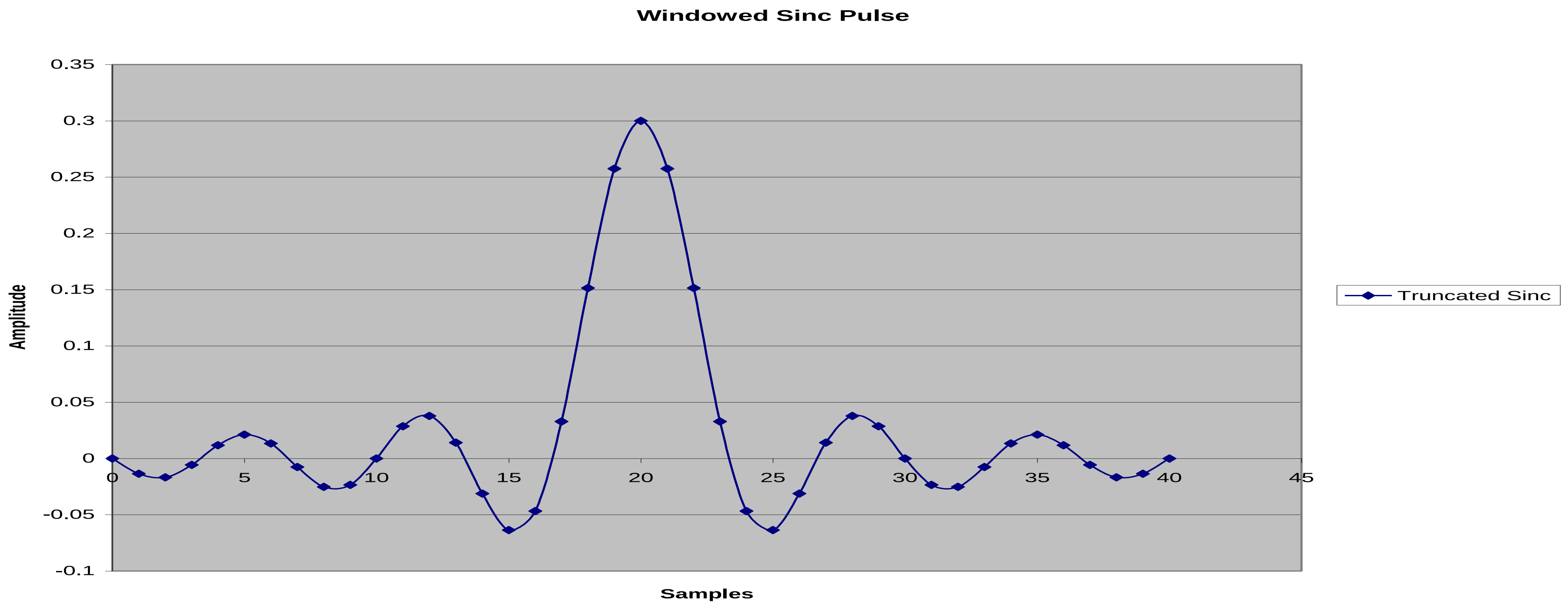
Assume an ideal filter in the frequency domain – Brick Wall



FIR Filter

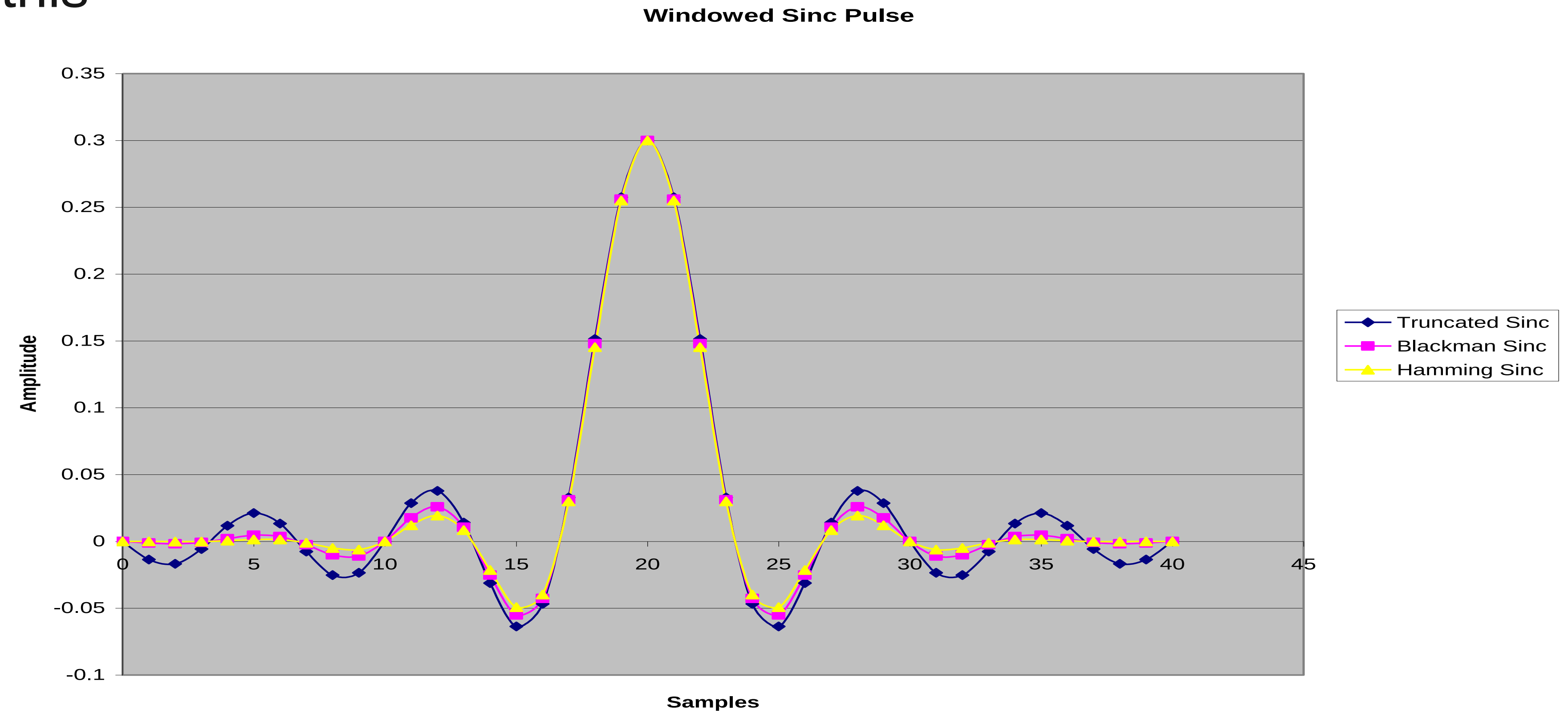
IFFT of the brick wall filter gives us the Windowed Sync Pulse

The ripples extend to infinity and never settle to zero.



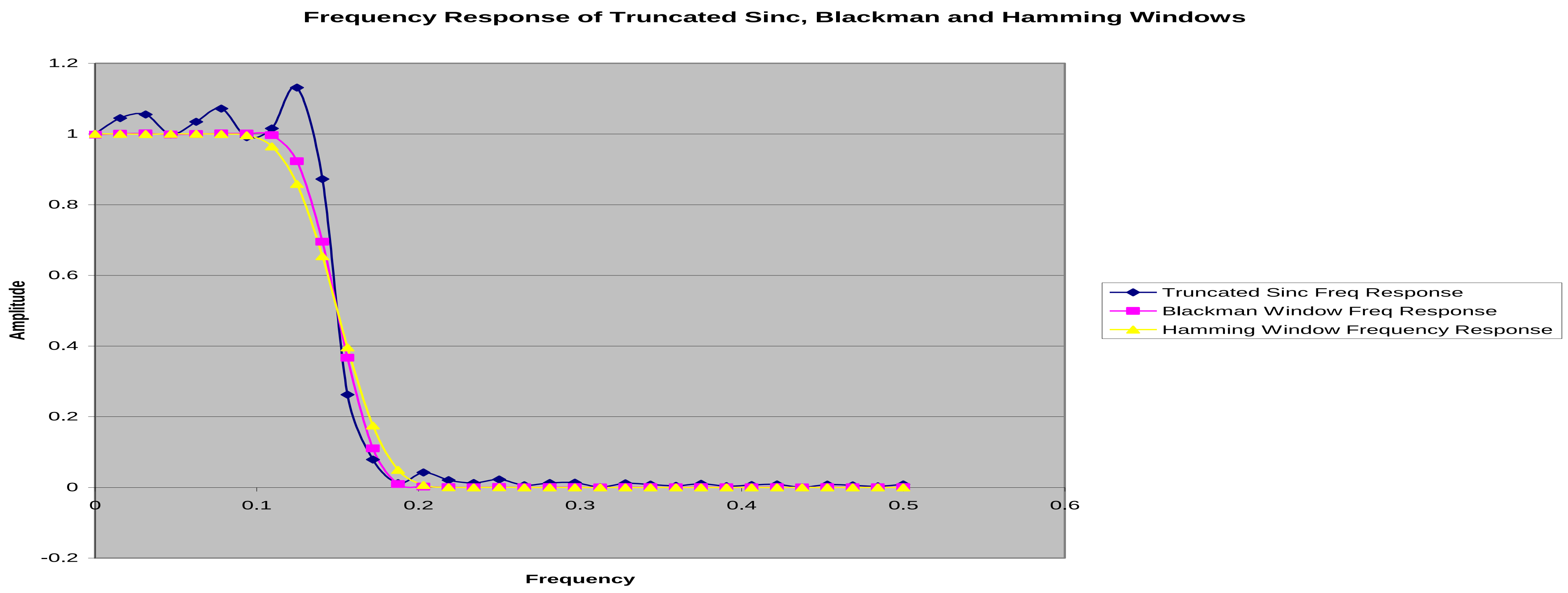
FIR Filter

Truncating the impulse response gives us ripples – Windowing helps address this



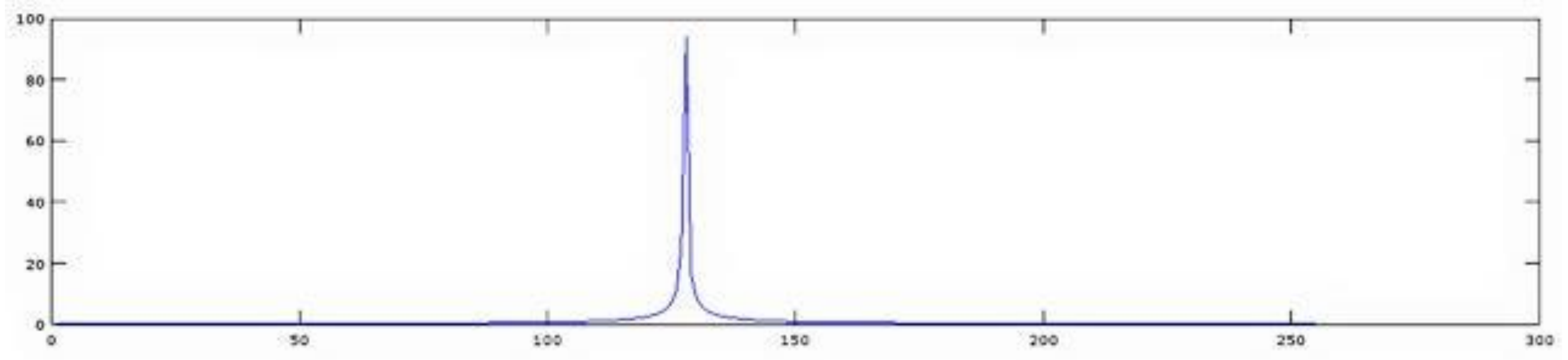
FIR Filter

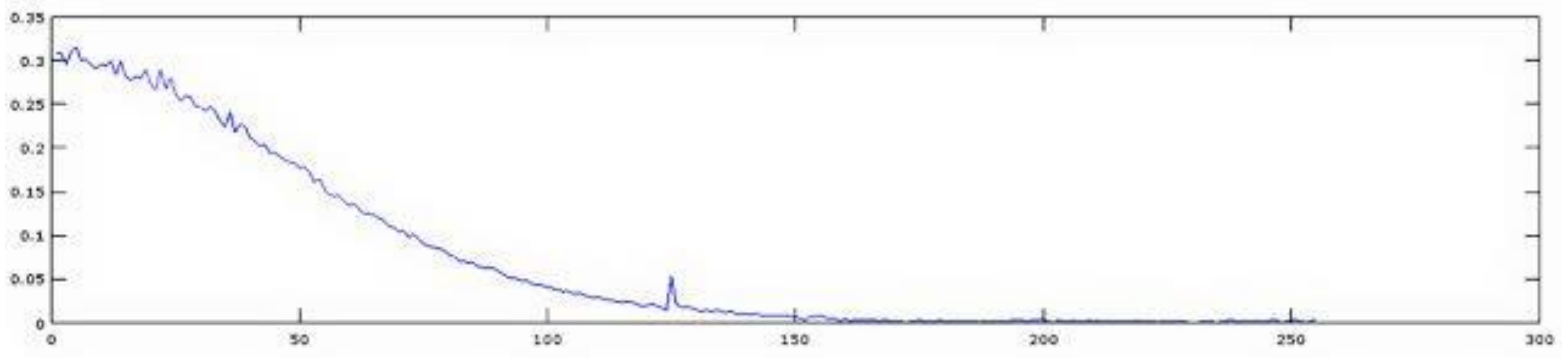
Filter Response is improved with window



FIR Filter

When Implemented (Input top / Output bottom)



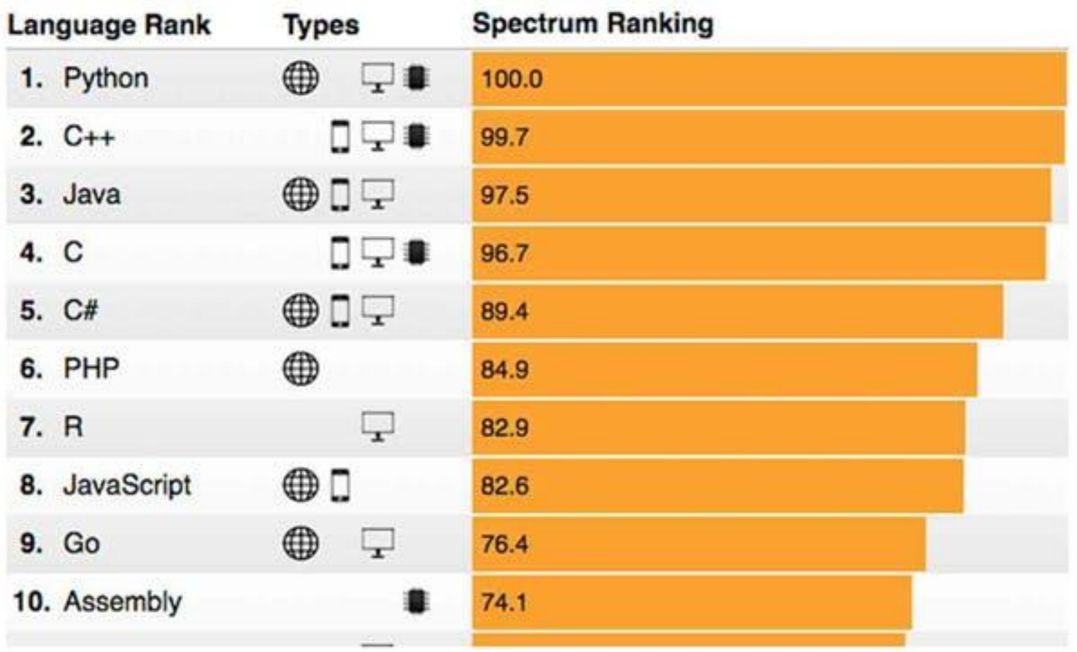


What is PYNQ – Introduction to PYNQ framework

What is PYNQ?

PYNQ is an open source project started by Xilinx, which fuses the productivity of Python with the acceleration provided by programmable logic within the Zynq / Zynq MPSoC

Hosted at PYNQ.io



Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥 ▪	100.0
2. C++	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	97.5
4. C	📱 🖥 ▪	96.7
5. C#	🌐 📱 🖥	89.4
6. PHP	🌐	84.9
7. R	🖥	82.9
8. JavaScript	🌐 📱	82.6
9. Go	🌐 🖥	76.4
10. Assembly	▪	74.1

Why should I learn PYNQ ?

Tight coupling of processing system (PS) and programmable logic (PL) in the Zynq / Zynq MPSoC creates a system which is

» Responsive – Leverage the parallel processing capability provided by the PL

» Deterministic – Creates processing pipeline competing for fewer shared resources

» Power Efficient – Less off chip transactions to or from DDR memory, dedicated hardware implementation is more efficient than

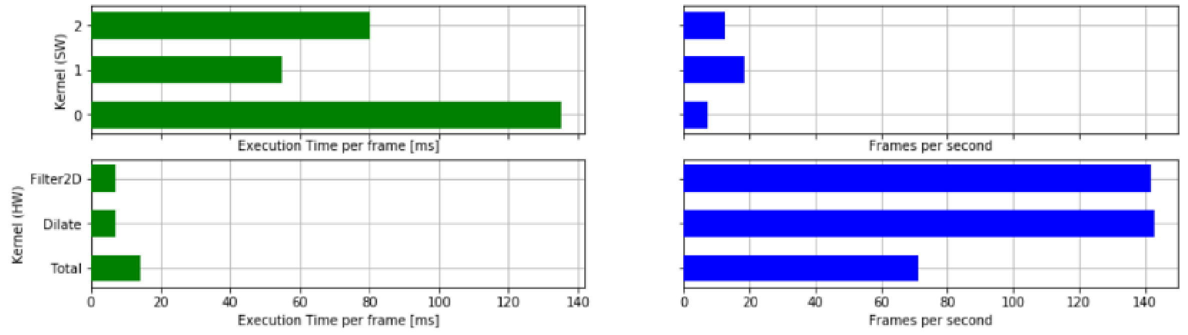
PYNQ frees Python programmers from the sequential software world and opens up the acceleration of programmable logic without the need to be a digital designer.

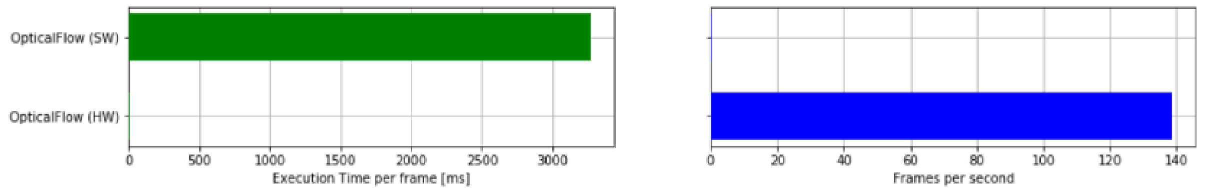
Why should I learn PYNQ ?

Simple example of PYNQ in a image processing application

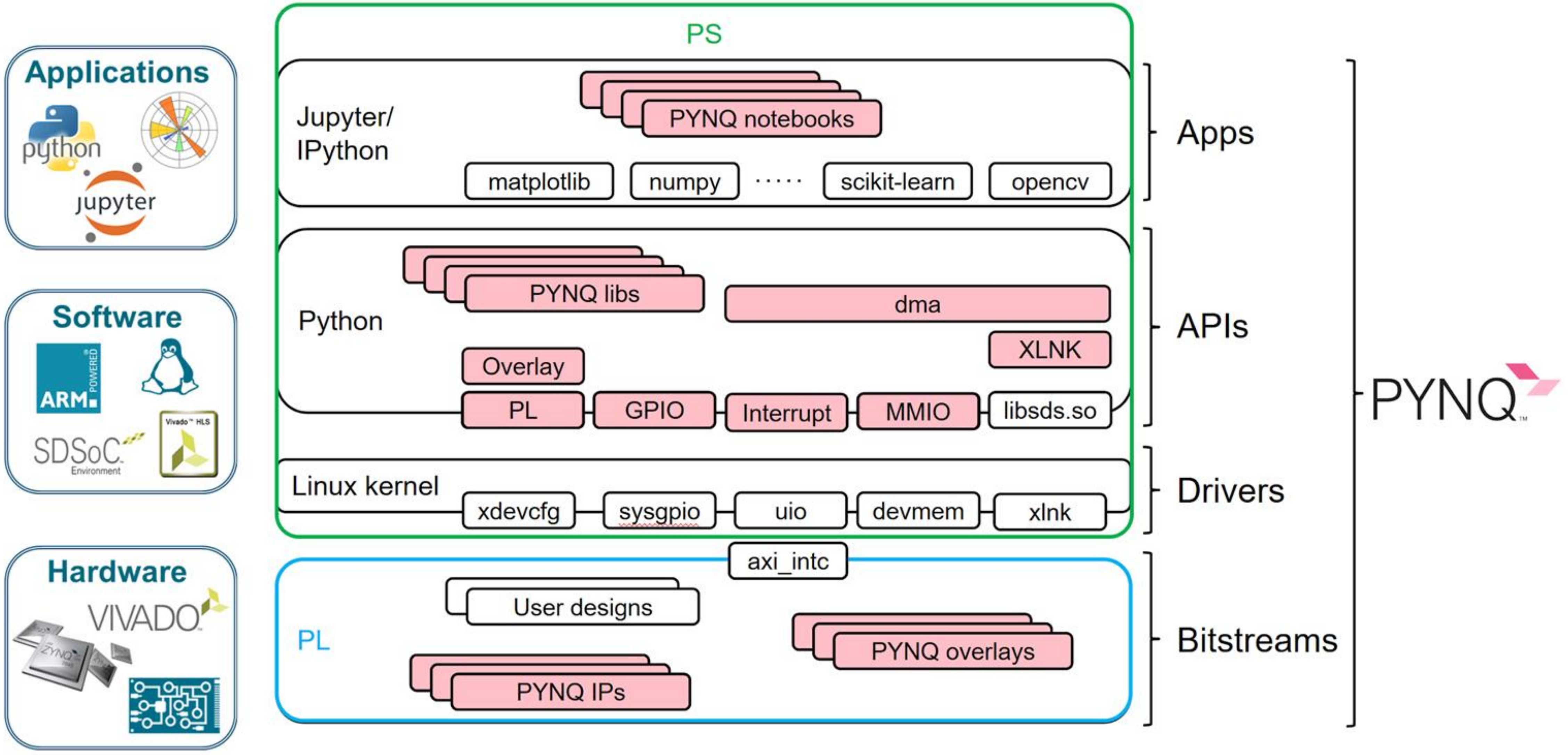
» Image Filtering
- SW < 20 Frames per Second
- HW > 60 Frames per Second

» Optical Flow
- SW < 1 Frame per Second
- HW > 120 Frames per Second





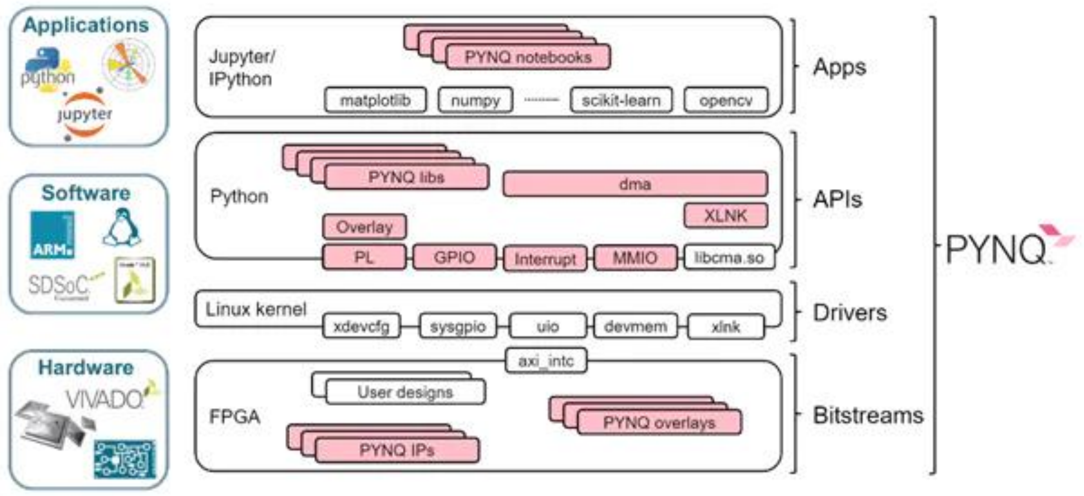
PYNQ Framework: interfacing Python with Xilinx SoC



PYNQ Components

To achieve performance PYNQ is comprised of

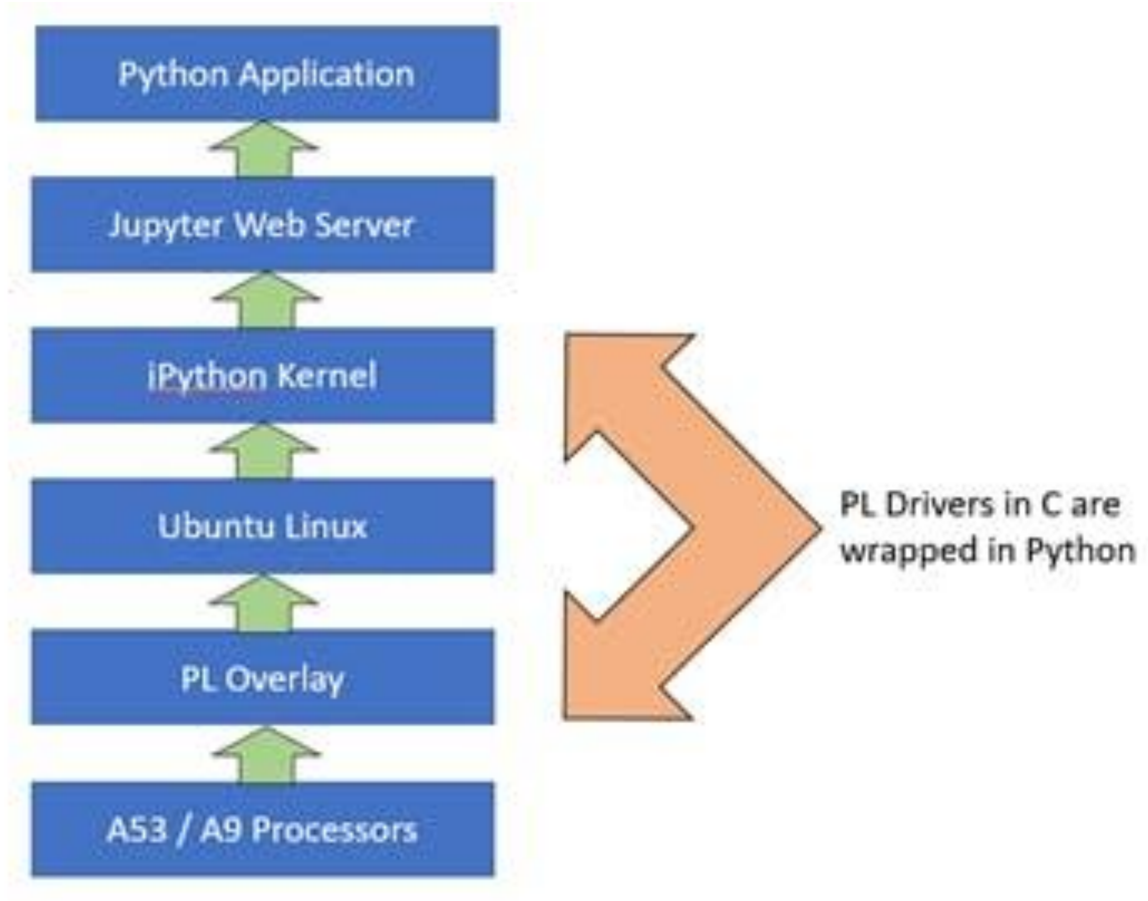
- » Jupyter Notebooks
- » PYNQ Package
- » PYNQ Libs
- » PYNQ Classes
- » PYNQ IP
- » PYNQ Overlays



PYNQ Architecture

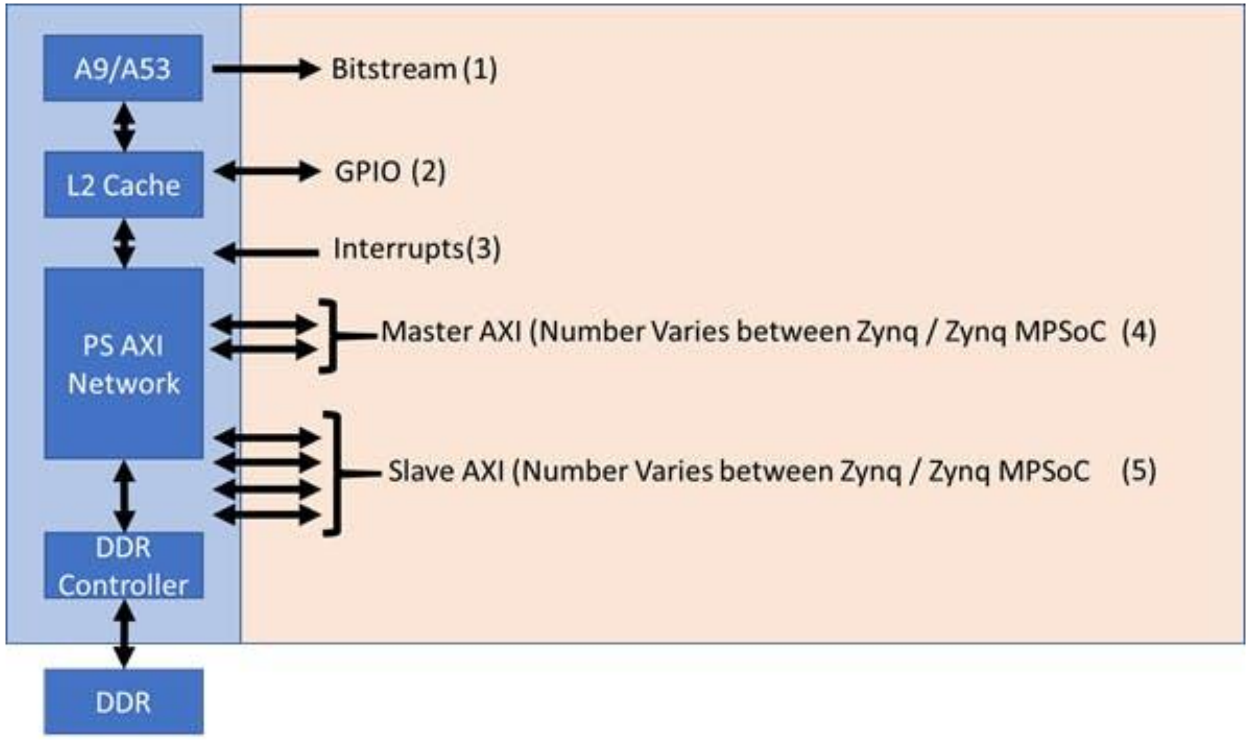
Pynq is built upon Xilinx Petalinux flow

Standard way to create PYNQ for a custo
we will see



Working with the PL

1. Bitstream — This configures the programmable logic for the desired application. In the PYNQ framework, the **xdevcfg** driver is used.
2. GPIO — This provides simple IO in both directions. In the PYNQ framework, this is supported by the **sysgpio** driver.
3. Interrupts — Support interrupt generation from the programmable logic to the processing system. In the PYNQ framework, this is supported by the **Userspace IO** driver.
4. Master AXI Interfaces — These are used to transfer data between the PS to the PL when the PS is the initiator of the transaction. The PYNQ framework uses **devmem** when employing master AXI interface.
5. Slave AXI Interfaces — These are used to transfer data between the PS and PL when the PL is the initiator of the transaction. The PYNQ framework uses **xlnk** to enable these transfers.


A9/A53
L2 Cache
PS AXI Network
DDR Controller
DDR
Bitstream (1)
GPIO (2)
Interrupts(3)
Master AXI (Number Varies between Zynq / Zynq MPSoC (4)
Slave AXI (Number Varies between Zynq / Zynq MPSoC (5)

Available PYNQ boards

Several PYNQ builds for existing boards:

- » Pynq Z1 — Zynq SoC 7020 & Arty Z7-20
- » Pynq Z2 — Zynq SoC 7020
- » ZCU104 — Zynq MPSoC XCZU7EV
- » ZCU111 — Zynq RFSoC XCZU28DR
- » Ultra96 — Zynq MPSoC ZU3EG

Overlays

Overlays are the design loaded into the programmable logic

Can be custom created or accessed via the PYNQ.IO community

Range of Overlays in the community including

- Machine Learning
- Image Processing
- RISC-V
- Kalman filter

Base Overlay is the initial overlay which is created with the PYNQ Image

Download and work with new overlays as required

Of course you can also create you own – As we will see

PYNQ Libraries

PYNQ provides several libraries which provide support for management of the processor and allow access to the low level hardware including

IP Cores – Audio, AXI GPIO, AXI IIC, DMA, Logic Tools, Video

IOP – Arduino, Grove, RPI, PMOD

PYNQ MicroBlaze – MicroBlaze Subsystem RPC and Library

PS / PL Interface – Interrupt, MMIO, PS GPIO, Xlnk

PS Control – PMBus

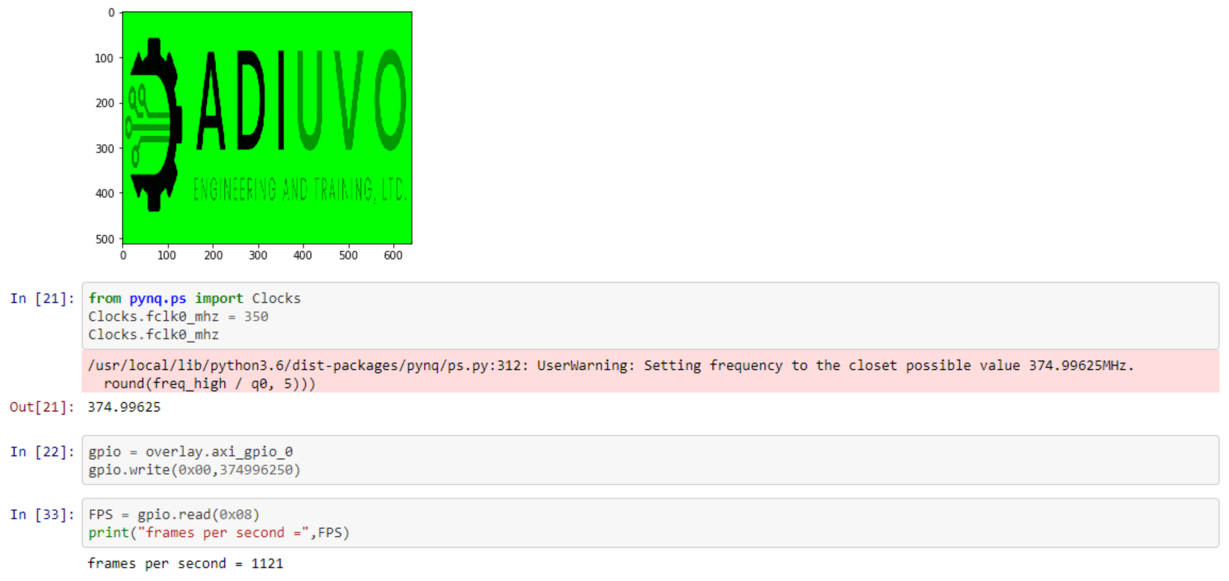
PL Control – Overlay, PL and Bitstream Classes

PYNQ In Industry

Working with THALES – Major Defense Client – High Speed Image processing (3000 > 10,000 FPS)

ZCU102 Development board

“The use of Pynq to rapidly test and evaluate design patterns for image processing has been invaluable. Not only has it sped up design, but it also reduces the necessary team size to a manageable level.”

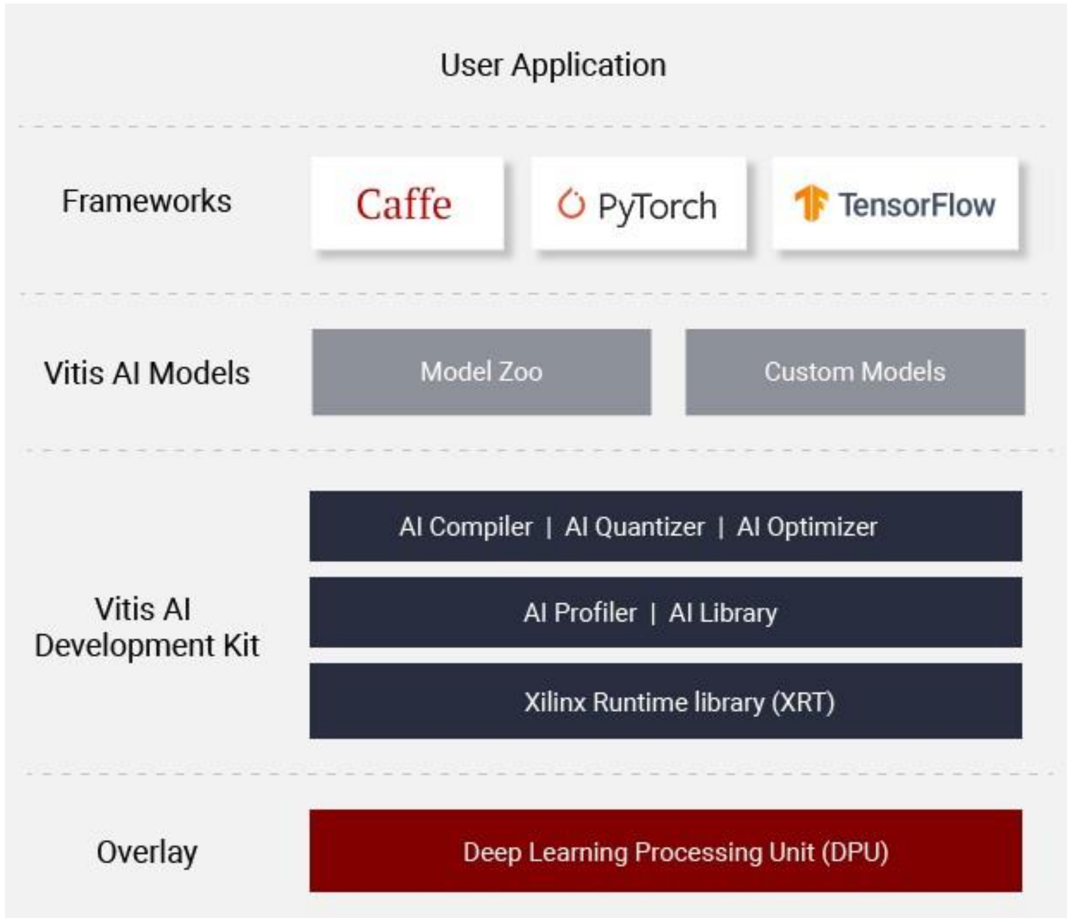


```
In [21]: from pynq.ps import Clocks
         Clocks.fclk0_mhz = 350
         Clocks.fclk0_mhz

         /usr/local/lib/python3.6/dist-packages/pynq/ps.py:312: UserWarning: Setting frequency to the closet possible value 374.99625MHz.
           round(freq_high / q0, 5)))

Out[21]: 374.99625

In [22]: gpio = overlay.axi_gpio_0
         gpio.write(0x00,374996250)

In [33]: FPS = gpio.read(0x08)
         print("frames per second =",FPS)

         frames per second = 1121
```

Vitis AI

Vitis AI Development Environment

- Vitis AI enables acceleration of AI inference at edge & Cloud
- Supports leading frameworks such as TensorFlow, Caffe and Pytorch
- Works with fixed point representation and Xilinx Deep Learning Processor Unit

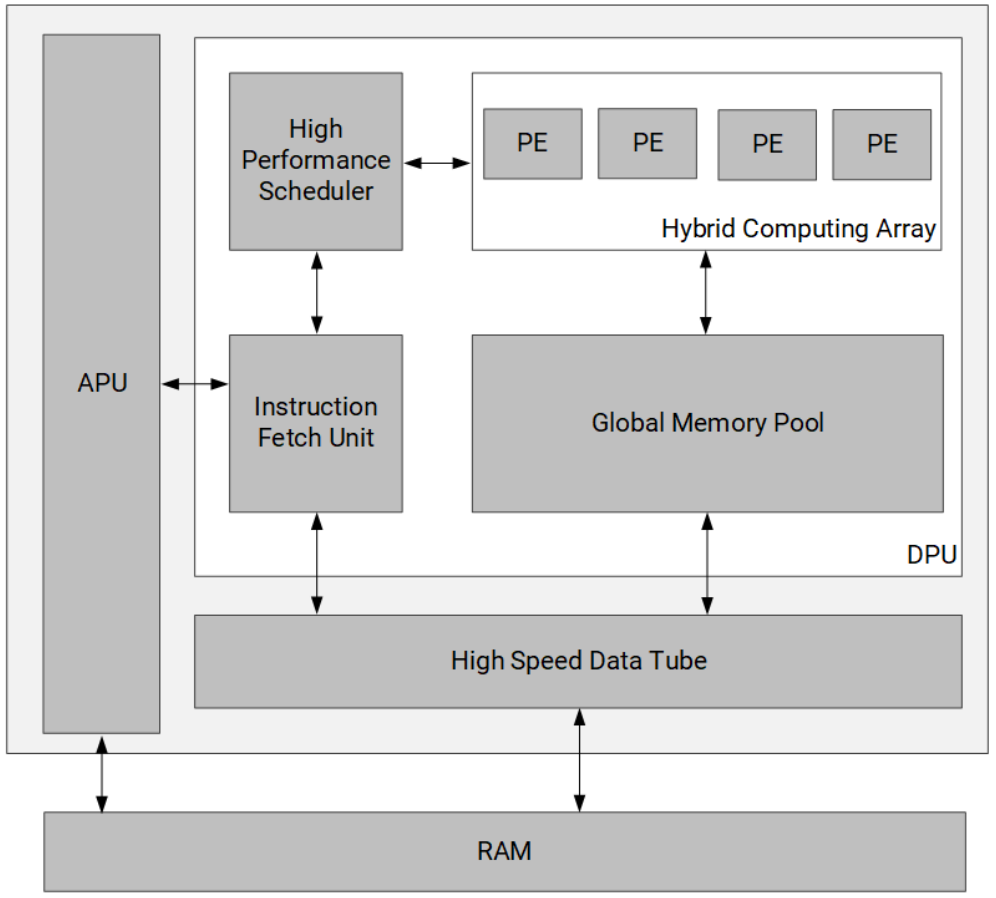


Deep Learning Processing Unit

Deep Learning Processing Unit (DPU) is a programmable engine optimized for convolutional neural networks

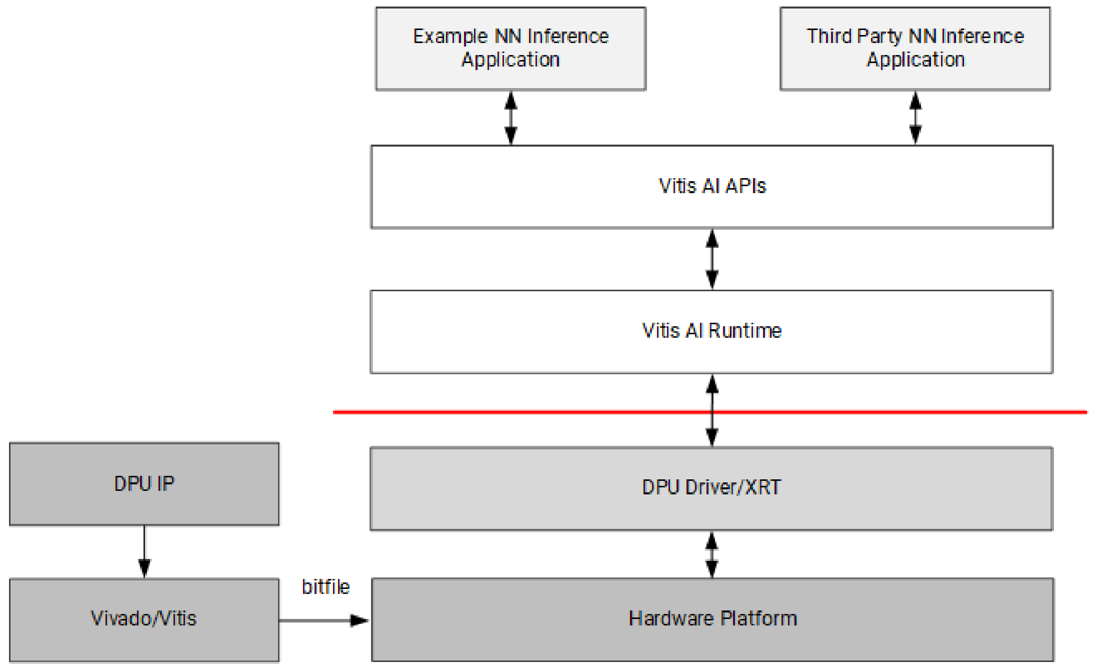
Can be used to implement

VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, and FPN



DPU Development flow

- Use Vitis / Vivado to generate bit stream
- Vitis AI to generate the neural network.
- Can come from Xilinx Model Zoo

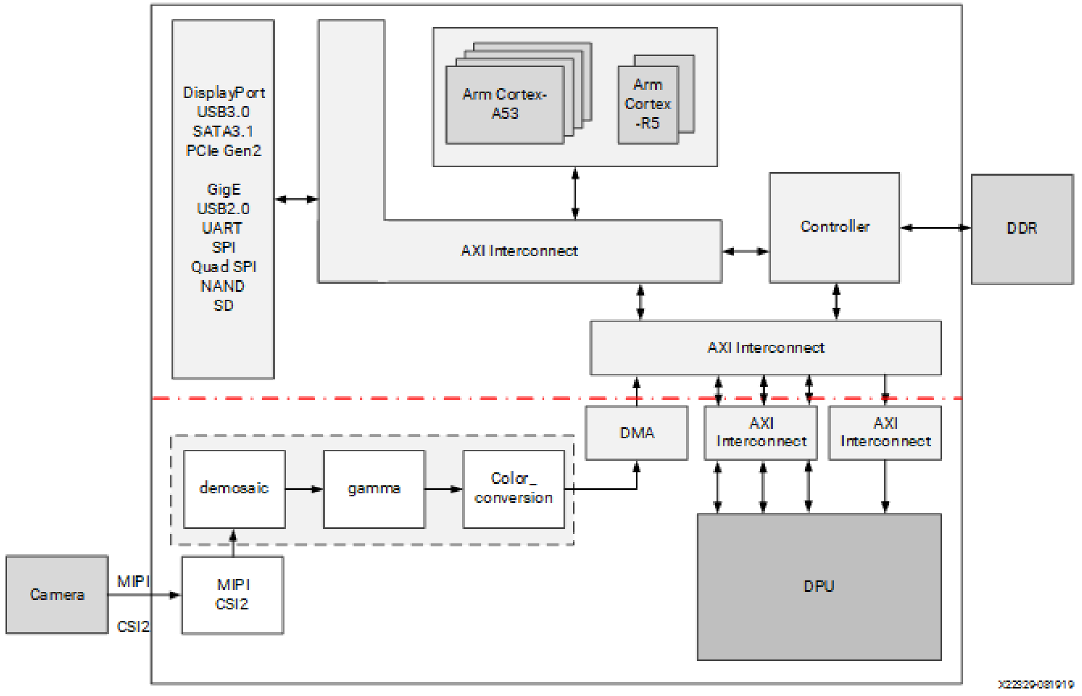


Example Application

AI is only a small part (but important part) of the solution

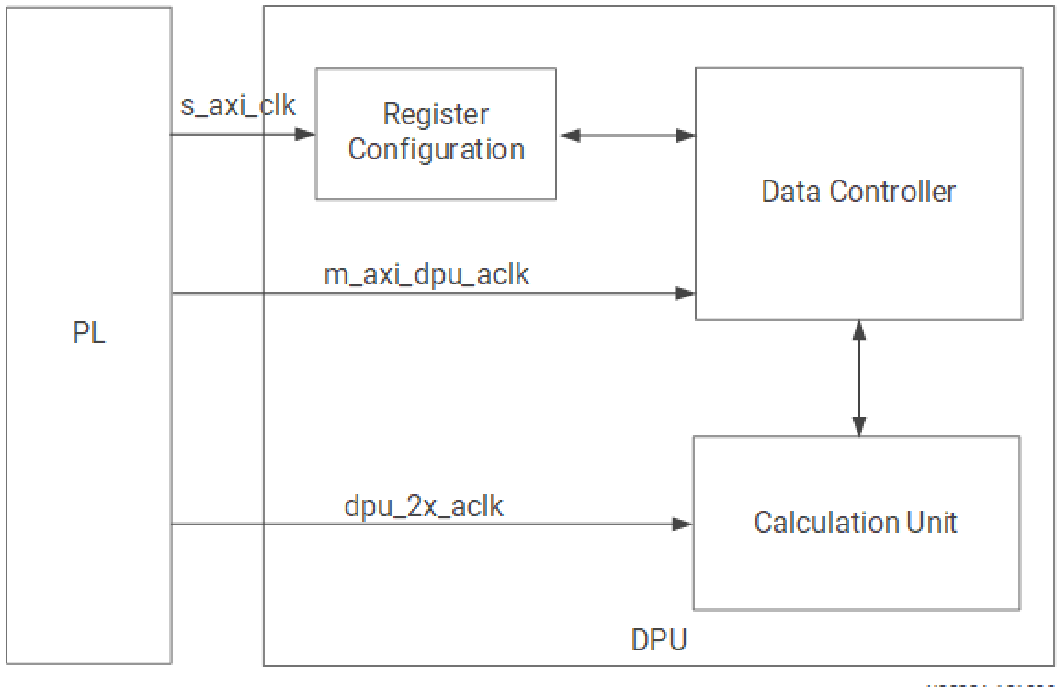
Need to be able to get image into or out of the system.

For example, MIPI camera interface



Clocking the DPU

- s_axi_aclk is used for the register configuration module. This module receives the DPU configuration though the S_AXI interface. The S_AXI clock can be configured as common with the M-AXI clock or as an independent clock.
- The primary function of the data controller module is to schedule the data flow in the DPU IP. The data controller module works with m_axi_dpu_aclk. The data transfer between the DPU and external memory happens in the data controller clock domain, so m_axi_dpu_aclk is also the AXI clock for the AXI_MM master interface in the DPU IP
- The DSP slices in the computation unit module are in the dpu_2x_clk domain, which runs at twice the clock frequency of the data controller module. The two related clocks must be edge-aligned.

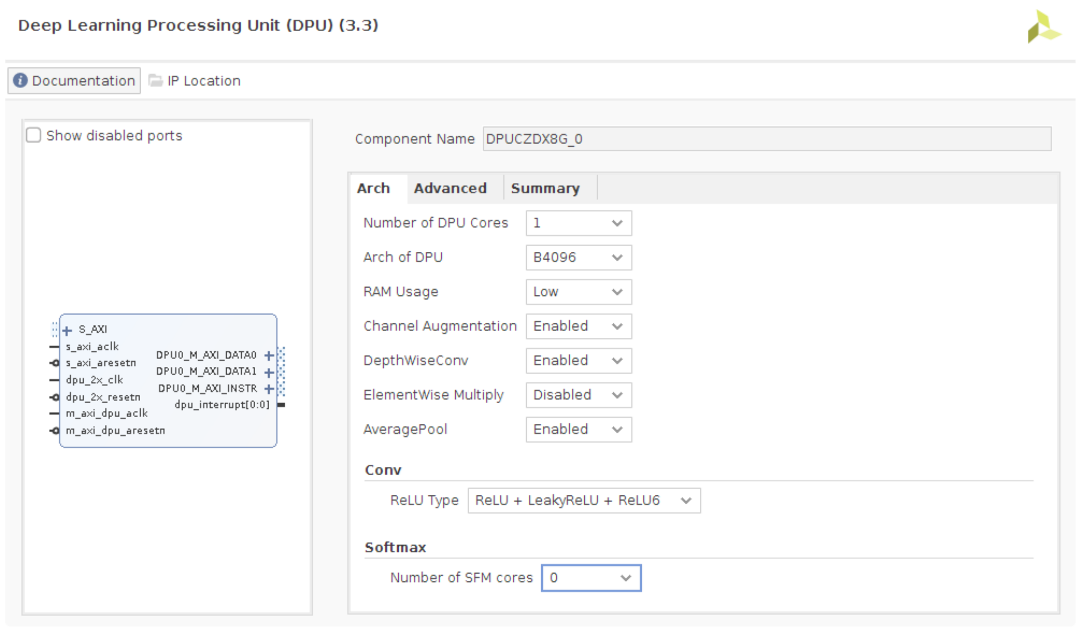


DPU is configurable

- Depending on flow either IP integrator or in script (Vitis)

- Can control performance and resource usage

- Can be deployed in



Deep Learning Processing Unit (DPU) (3.3)

Documentation | IP Location

☐ Show disabled ports

Component Name DPUCZDX8G_0

Arch | Advanced | Summary

Number of DPU Cores	1
Arch of DPU	B4096
RAM Usage	Low
Channel Augmentation	Enabled
DepthWiseConv	Enabled
ElementWise Multiply	Disabled
AveragePool	Enabled

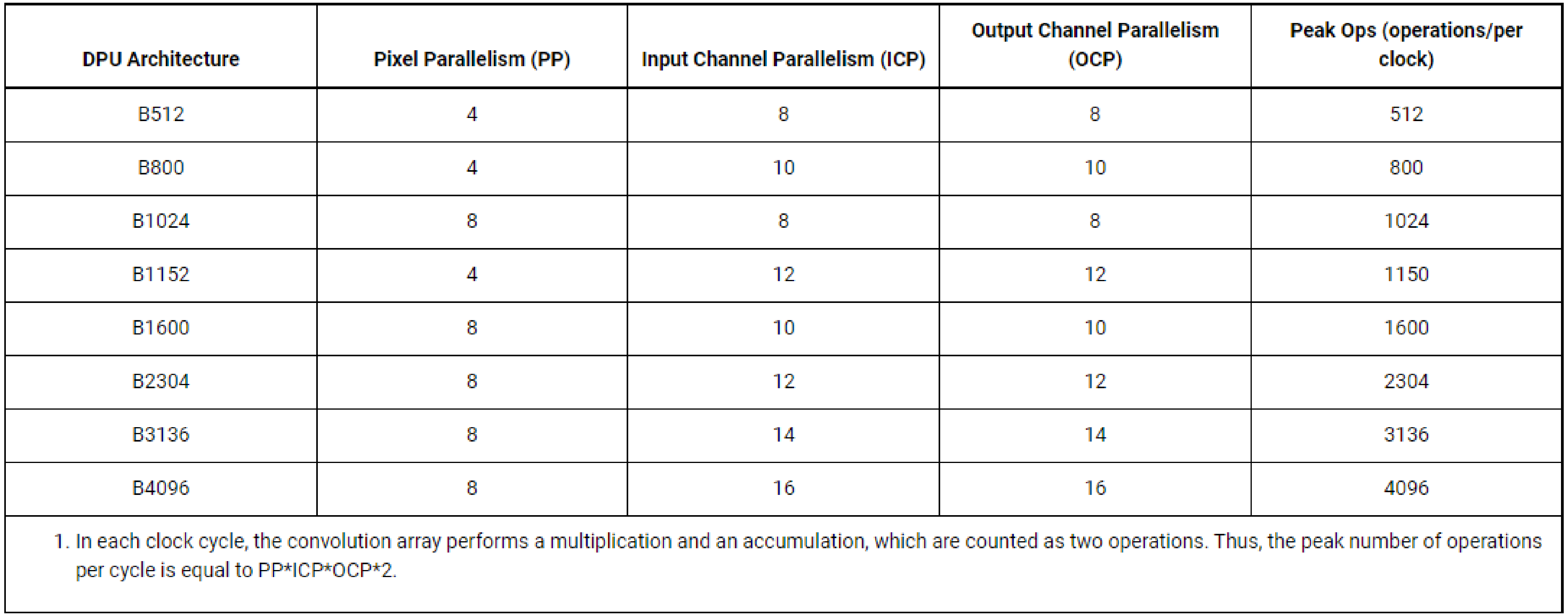
Conv

ReLU Type: ReLU + LeakyReLU + ReLU6

Softmax

Number of SFM cores: 0

Performance Optimisation



DPU Architecture	Pixel Parallelism (PP)	Input Channel Parallelism (ICP)	Output Channel Parallelism (OCP)	Peak Ops (operations/per clock)
B512	4	8	8	512
B800	4	10	10	800
B1024	8	8	8	1024
B1152	4	12	12	1150
B1600	8	10	10	1600
B2304	8	12	12	2304
B3136	8	14	14	3136
B4096	8	16	16	4096

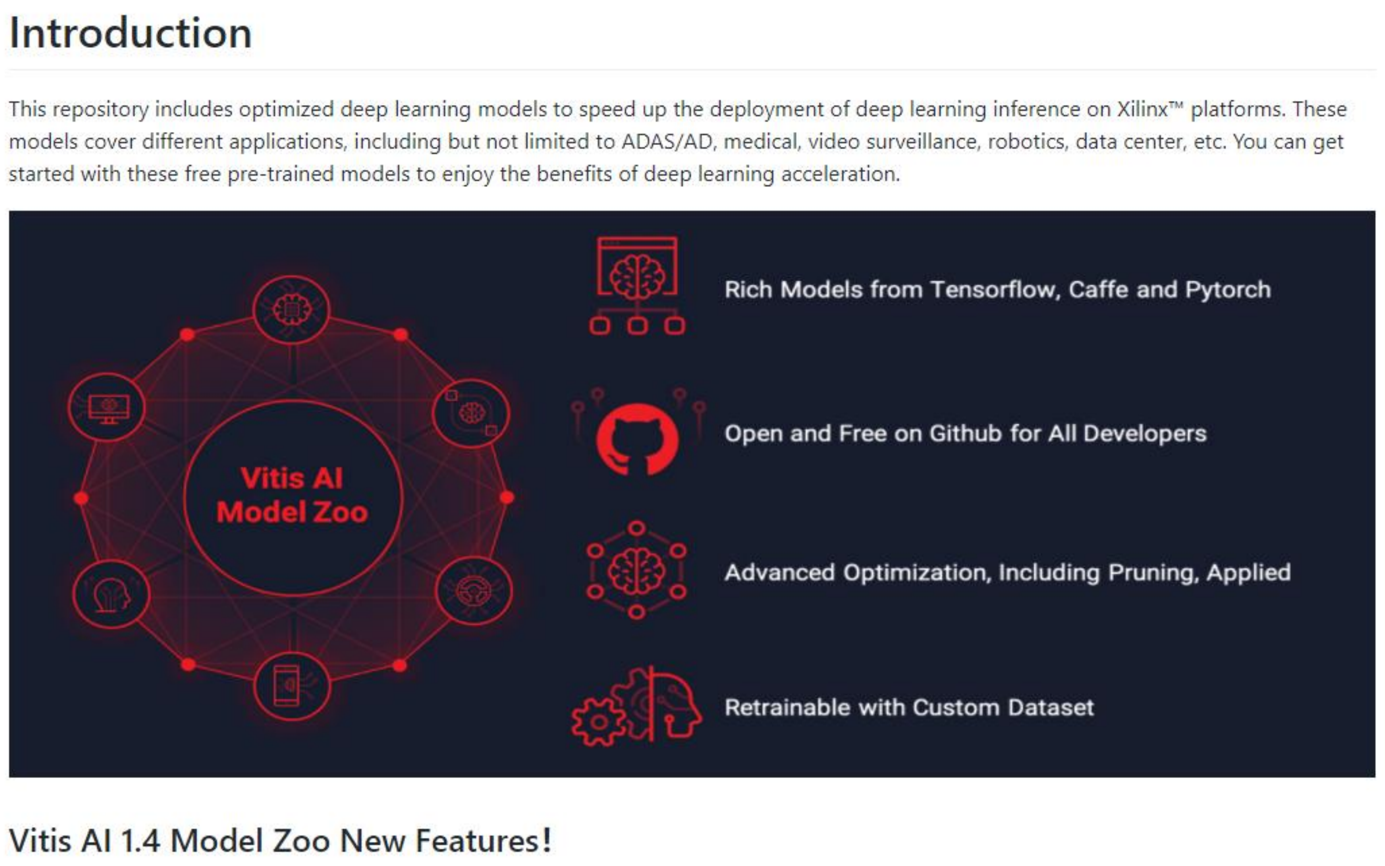
1. In each clock cycle, the convolution array performs a multiplication and an accumulation, which are counted as two operations. Thus, the peak number of operations per cycle is equal to PP*ICP*OCP*2.

Resource Optimization

DPU Architecture	LUT	Register	Block RAM	DSP
B512 (4x8x8)	27893	35435	73.5	78
B800 (4x10x10)	30468	42773	91.5	117
B1024 (8x8x8)	34471	50763	105.5	154
B1152 (4x12x12)	33238	49040	123	164
B1600 (8x10x10)	38716	63033	127.5	232
B2304 (8x12x12)	42842	73326	167	326
B3136 (8x14x14)	47667	85778	210	436
B4096 (8x16x16)	53540	105008	257	562

Model Zoo

Networks and many others are often made available for different frameworks in a Model Zoo



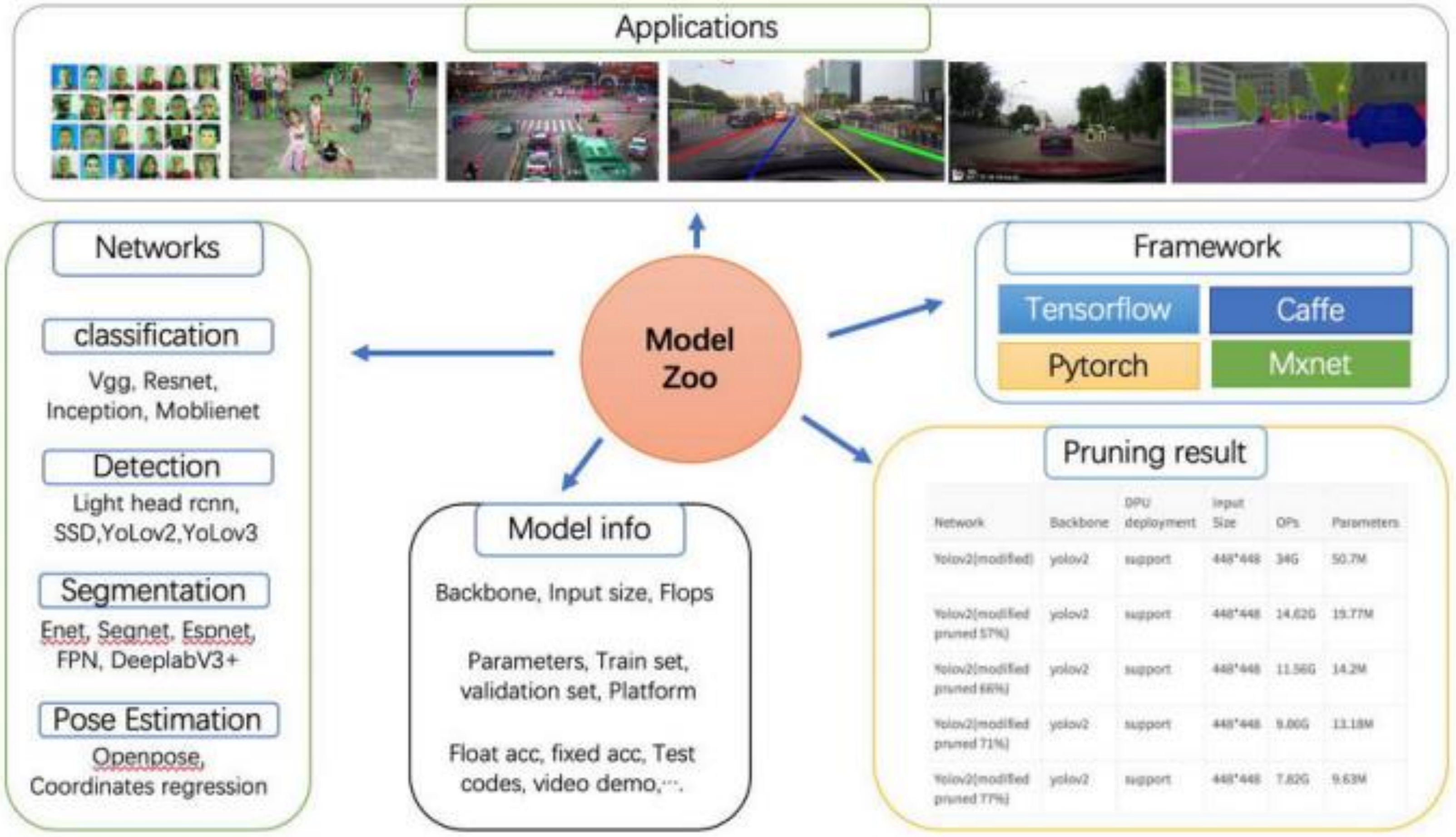
Introduction

This repository includes optimized deep learning models to speed up the deployment of deep learning inference on Xilinx™ platforms. These models cover different applications, including but not limited to ADAS/AD, medical, video surveillance, robotics, data center, etc. You can get started with these free pre-trained models to enjoy the benefits of deep learning acceleration.

Vitis AI Model Zoo
Rich Models from Tensorflow, Caffe and Pytorch
Open and Free on Github for All Developers
Advanced Optimization, Including Pruning, Applied
Retrainable with Custom Dataset

Vitis AI 1.4 Model Zoo New Features!

Model Zoo


Applications
Networks
classification
Vgg, Resnet, Inception, Moblienet
Detection
Light head rcnn, SSD,YoLov2,YoLov3
Segmentation
Enet, Segnet, Espnet, FPN, DeeplabV3+
Pose Estimation
Openpose, Coordinates regression
Model Zoo
Model info
Backbone, Input size, Flops
Parameters, Train set, validation set, Platform
Float acc, fixed acc, Test codes, video demo,….
Framework
Tensorflow
Caffe
Pytorch
Mxnet
Pruning result

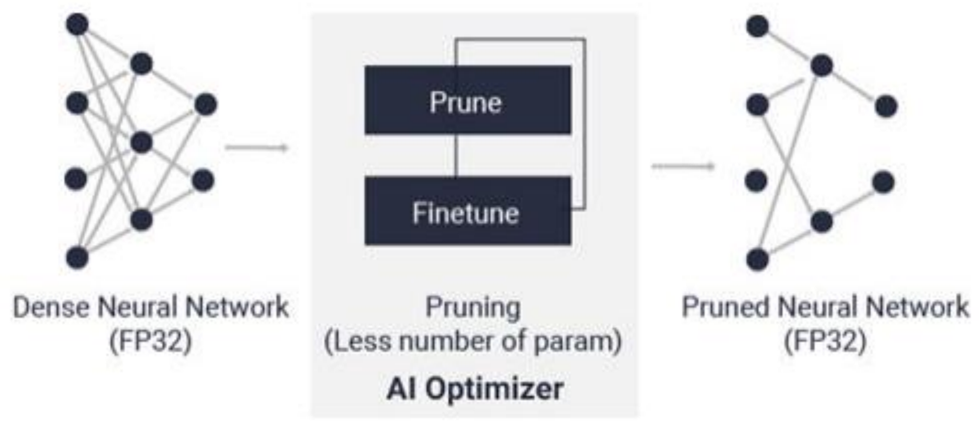
Network	Backbone	DPU deployment	Input Size	OPs	Parameters
Yolov2(modified)	yolov2	support	448*448	34G	50.7M
Yolov2(modified pruned 57%)	yolov2	support	448*448	14.62G	19.77M
Yolov2(modified pruned 66%)	yolov2	support	448*448	11.56G	14.2M
Yolov2(modified pruned 71%)	yolov2	support	448*448	9.00G	13.18M
Yolov2(modified pruned 77%)	yolov2	support	448*448	7.82G	9.63M

Vitis AI Optimizer

Optimizes the network

Reduce model complexity by 5x to 50x with minimal accuracy degradation

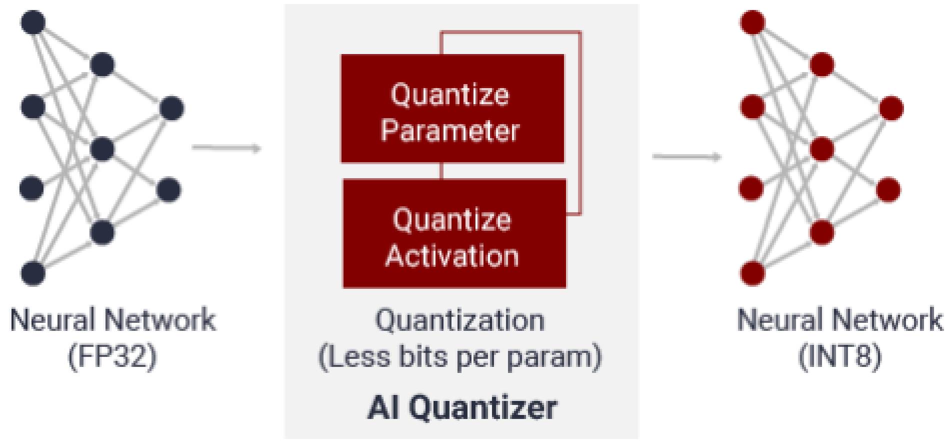
Requires a commercial license


Prune
Finetune
Dense Neural Network (FP32)
Pruning (Less number of param)
AI Optimizer
Pruned Neural Network (FP32)

Vitis AI Quantizer

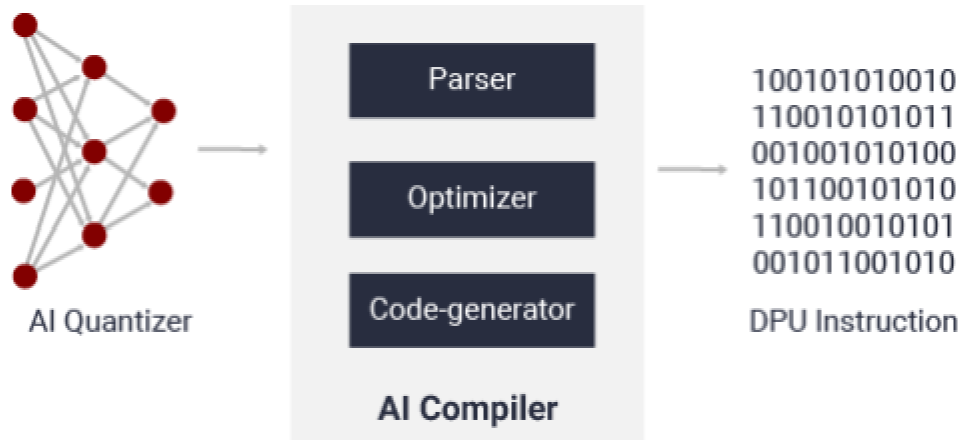
- Converts the 32-bit floating-point weights and activations to fixed-point e.g. INT8

- Fixedpoint network model requires less memory bandwidth, thus providing faster speed and higher power efficiency than the floating-point mode

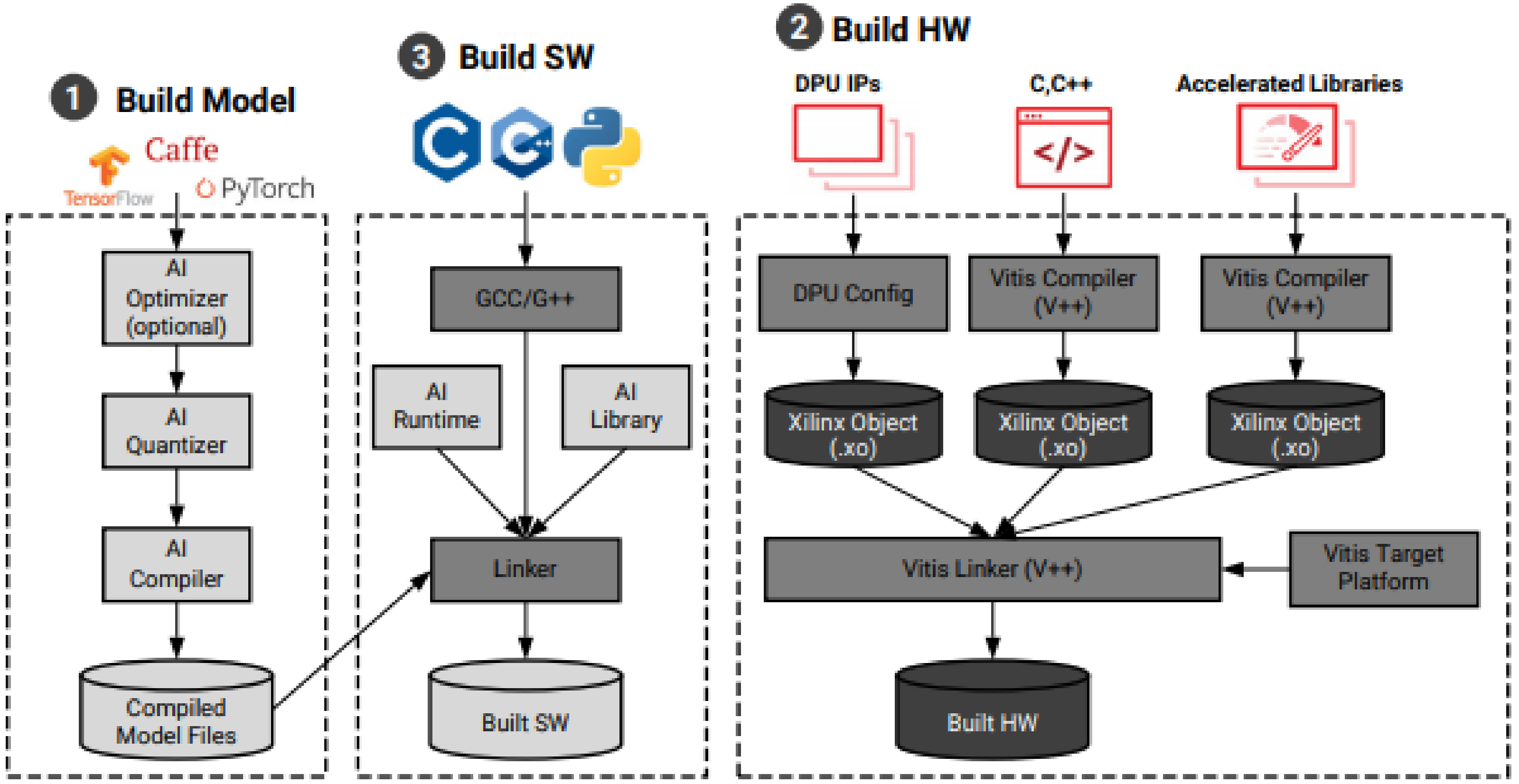


Vitis AI Compiler

- Maps the AI model to the instruction set and dataflow model
- Performs optimizations such as
  - layer fusion,
  - instruction scheduling
  - reuses on-chip memory.



Putting it all Together



ADIUVO

ENGINEERING AND TRAINING, LTD.

www.adiuvoengineering.com

adam@adiuvoengineering.com