

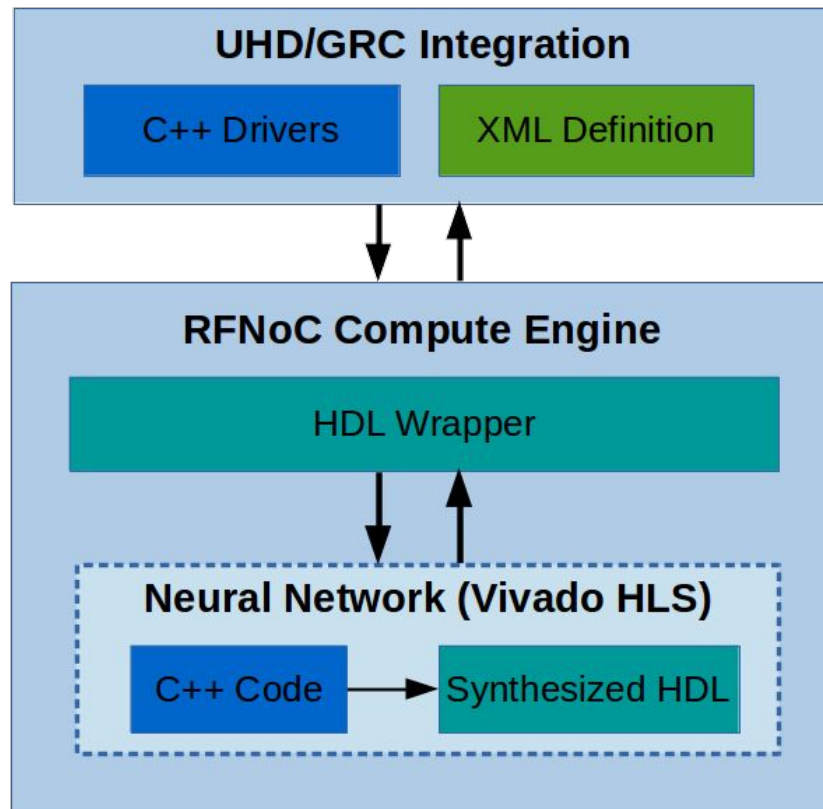
# RFNoC Neural-Network Library using Vivado HLS (rfnoc-hls-neuralnet)



EJ Kreinar  
Team “E to the J Omega”

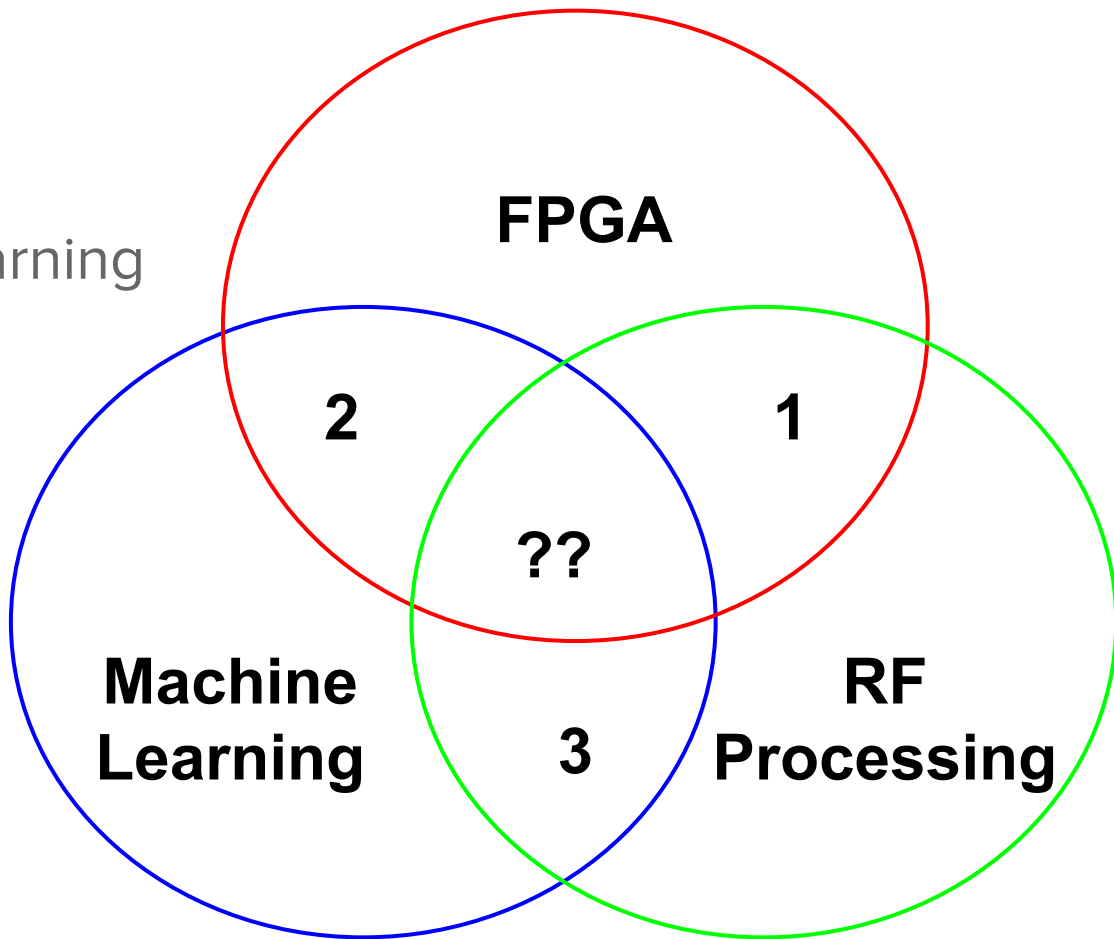
# Overview

An RFNoC out-of-tree module that can be used to simulate, synthesize, and run a neural network on an FPGA



# Why?

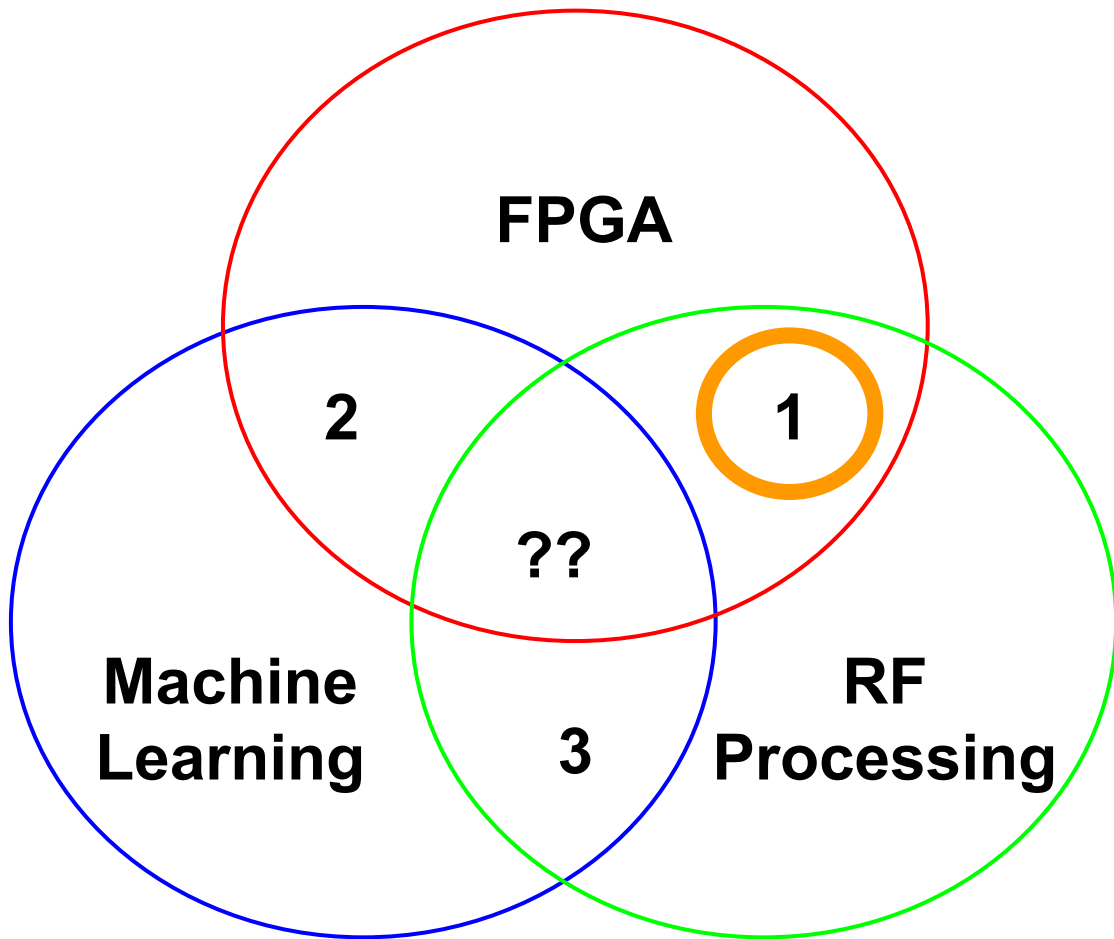
RF + FPGA + Machine Learning  
“venn diagram”



# Why?

- Conventional approach
- RFNoC

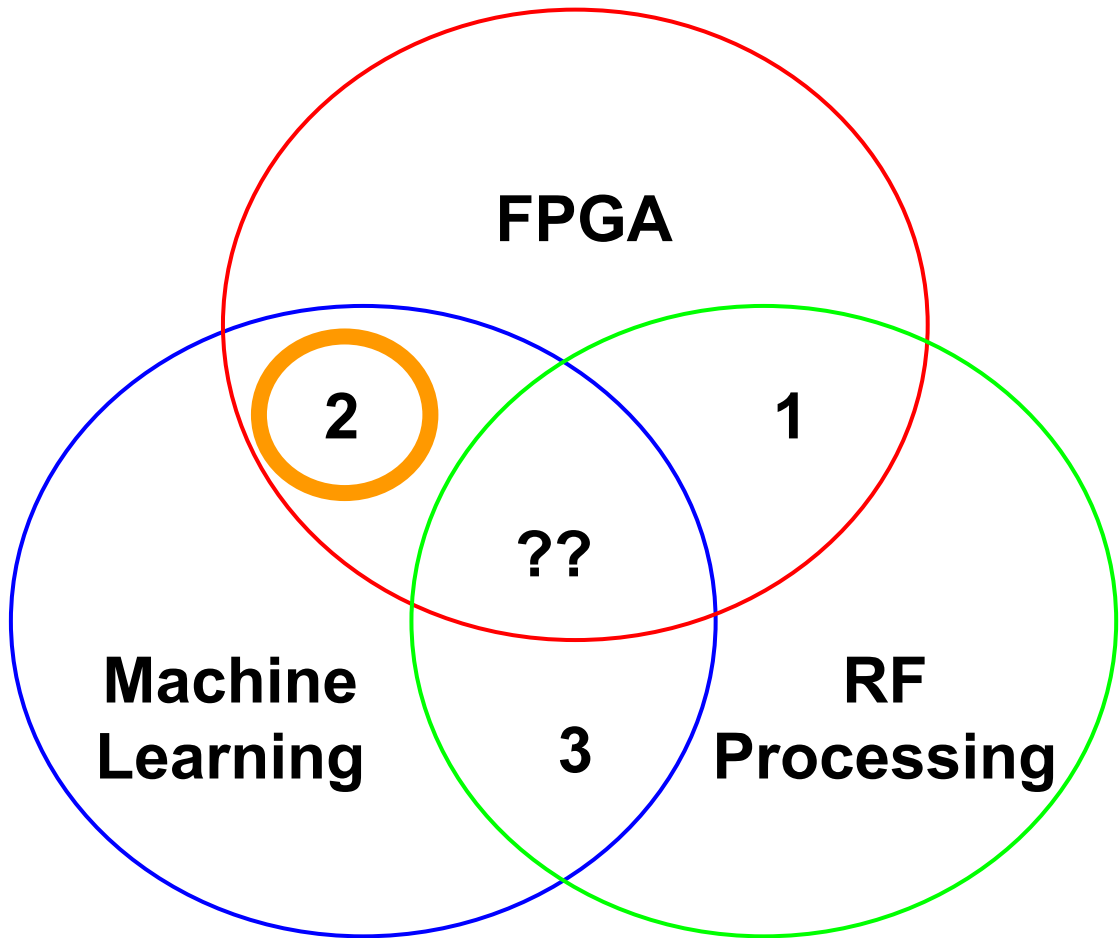
**Nothing New**



# Why?

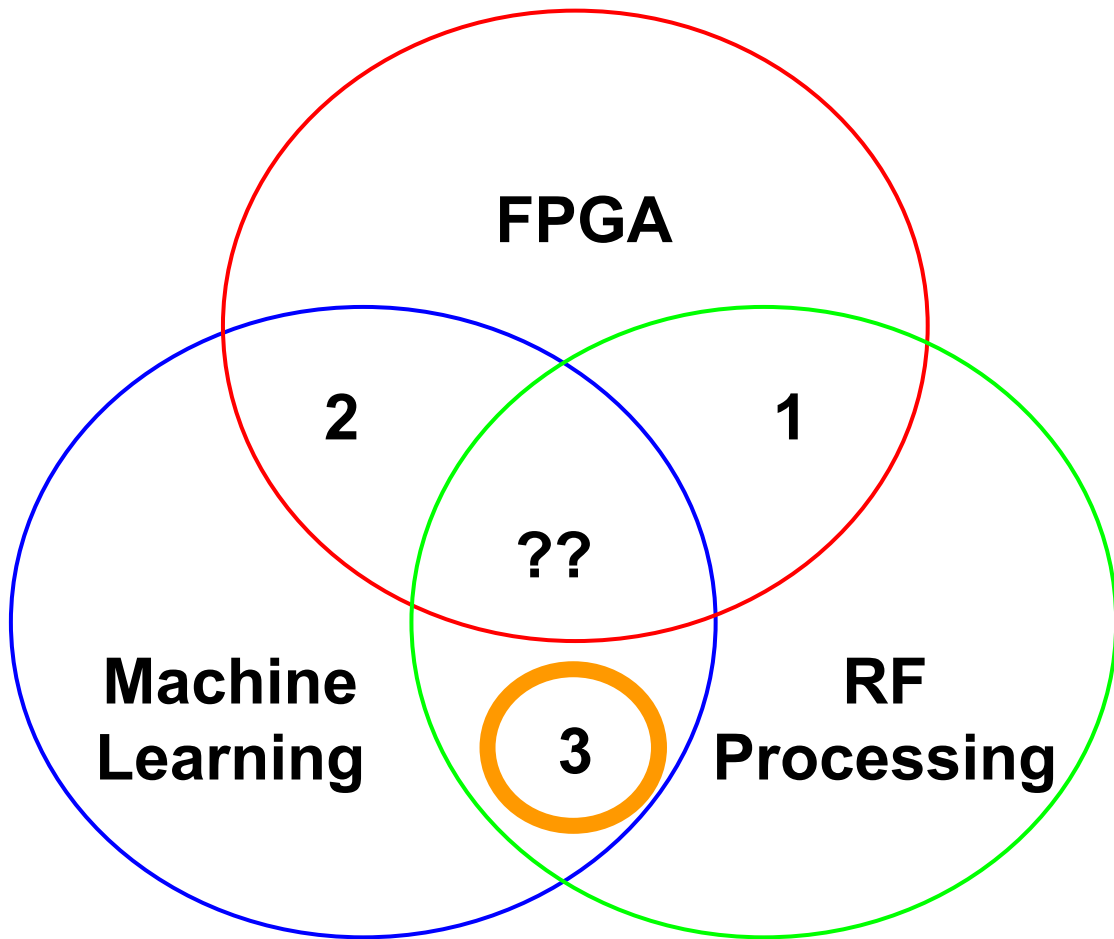
- Becoming very popular...
- AWS “F1”
  - Zynq Ultrascale
- Intel / Microsoft
  - Altera FPGA SoC
- Lots of research
  - Grad schools
  - Commercial
- Not much open source?

**Designed for Image  
Processing**



# Why?

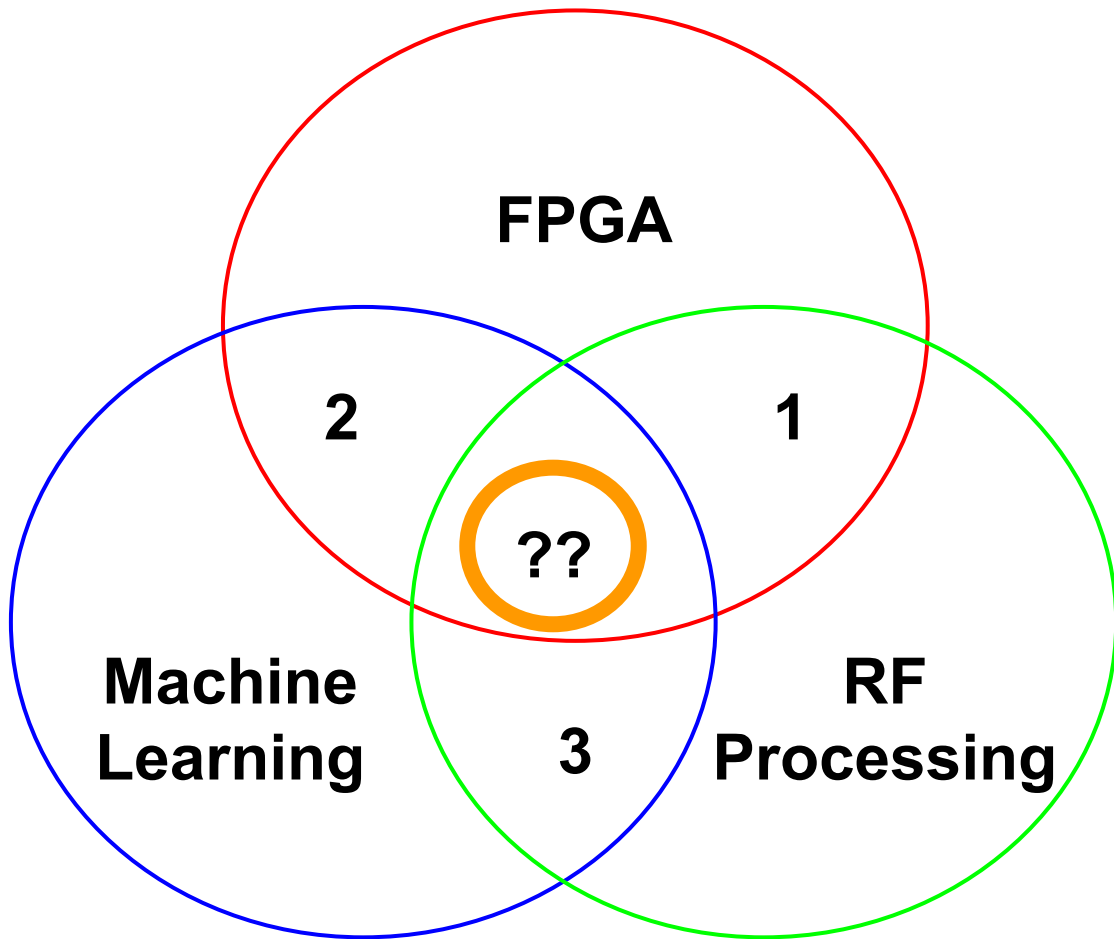
- “Cognitive radios”  
(not really machine learning)
- Neural network success  
in other fields
- GRCon 2016
  - Demonstrated  
interest & results
- DARPA BAA 2017



**Early adopter interest**

# Why?

- No current solution that combines all 3 technologies
- Lots of computing required
- Potentially requires running on embedded devices with low SWAP and high sample rate



**rfnoc-hls-neuralnet**

# Fundamental Problems

- Neural network performance is HIGHLY dependent on architecture
- FPGA development requires many iteration cycles of resource vs throughput vs latency tradeoffs
- ... plus glue code
- ... plus interaction with a processor

**A good solution will allow developers to *efficiently* iterate & change neural network architectures on the FPGA**

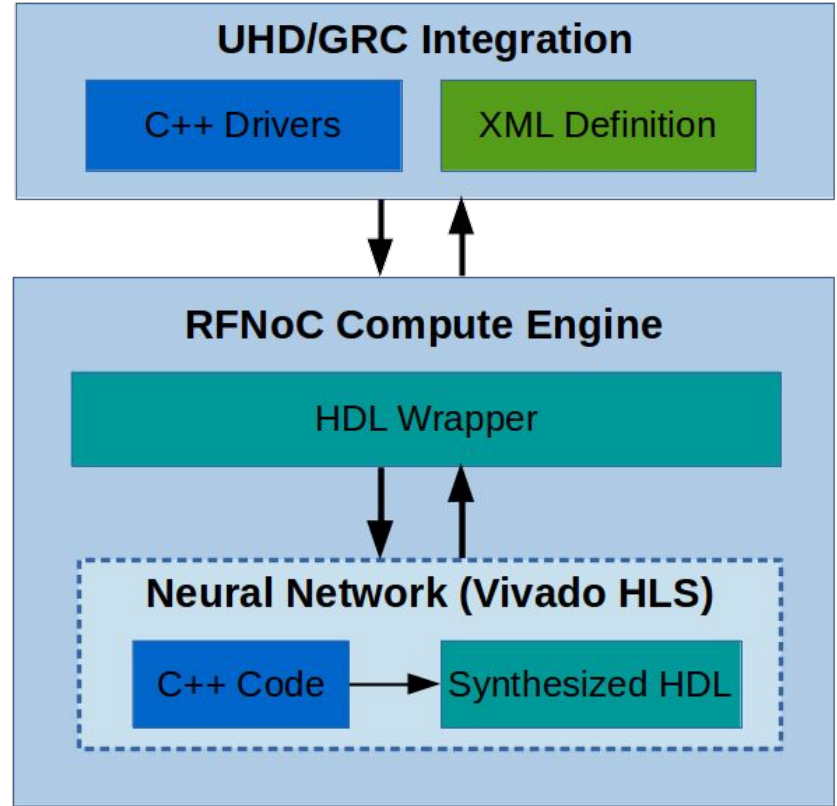




# HLS + RFNoC

1. **C++** neural network implementation
2. **Verilog** RFNoC compute engine
3. **XML** gnuradio-companion block interface

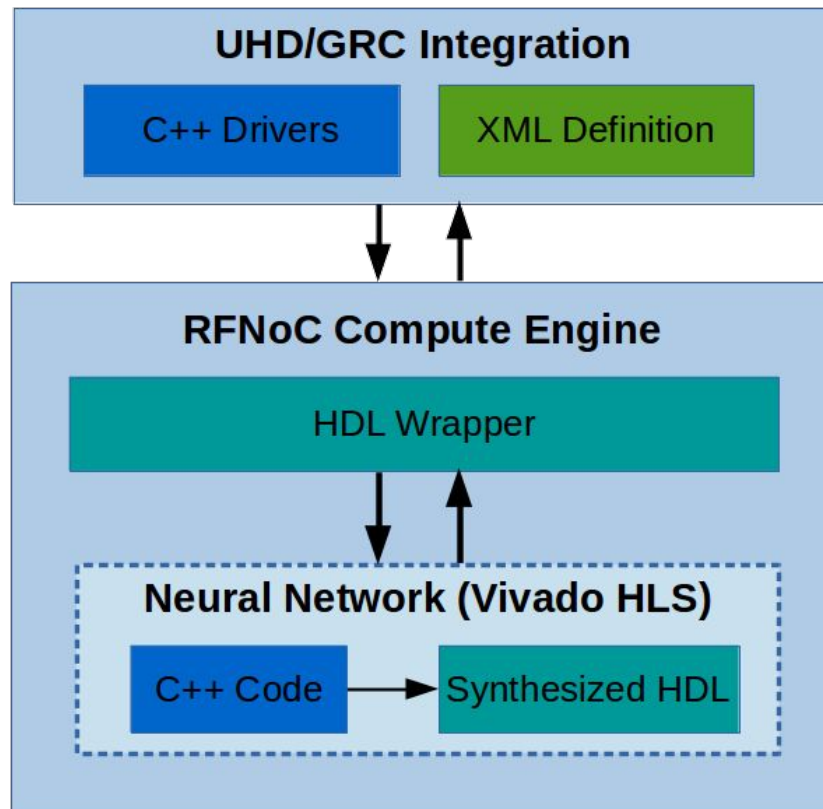
Designed as a tool to efficiently implement FPGA neural networks



# How To Use: C++

Neural net library provides pre-optimized “building blocks” that can instantiate a network similar to TensorFlow

- Fully connected layer
- IQ convolution layer
- One-dimensional convolution layer
- Activation functions:
  - relu
  - sigmoid
  - tanh
- Maxpool operation (size 2, stride 2)

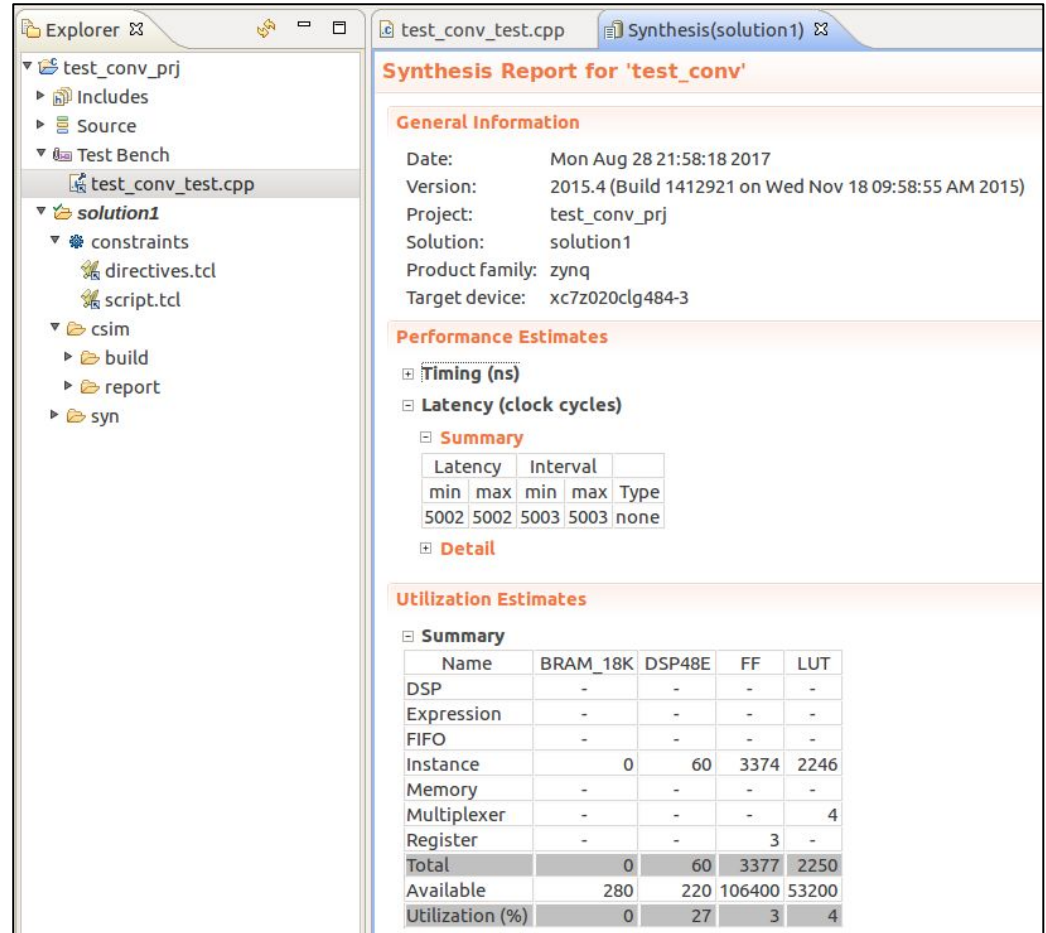


# How To Use: C++

Simulate the neural network using both floating point and fixed point data types

Synthesize into HDL code using Vivado HLS

Easy prototyping & simulation of neural network in C++



The screenshot displays the Vivado HLS interface. On the left, the Explorer pane shows a project named 'test\_conv\_prj' with subfolders for 'Includes', 'Source', and 'Test Bench'. The 'Test Bench' folder contains 'test\_conv\_test.cpp'. A 'solution1' folder is also visible, containing 'constraints', 'directives.tcl', 'script.tcl', 'csim', 'build', 'report', and 'syn'.

The right pane shows the 'Synthesis Report for 'test\_conv''. It is divided into three sections:

- General Information**
  - Date: Mon Aug 28 21:58:18 2017
  - Version: 2015.4 (Build 1412921 on Wed Nov 18 09:58:55 AM 2015)
  - Project: test\_conv\_prj
  - Solution: solution1
  - Product family: zynq
  - Target device: xc7z020clg484-3
- Performance Estimates**
  - Timing (ns)**
  - Latency (clock cycles)**
    - Summary**

| Latency | Interval |      |      |      |
|---------|----------|------|------|------|
| min     | max      | min  | max  | Type |
| 5002    | 5002     | 5003 | 5003 | none |
    - Detail**
- Utilization Estimates**
  - Summary**

| Name            | BRAM_18K | DSP48E    | FF          | LUT         |
|-----------------|----------|-----------|-------------|-------------|
| DSP             | -        | -         | -           | -           |
| Expression      | -        | -         | -           | -           |
| FIFO            | -        | -         | -           | -           |
| Instance        | 0        | 60        | 3374        | 2246        |
| Memory          | -        | -         | -           | -           |
| Multiplexer     | -        | -         | -           | 4           |
| Register        | -        | -         | 3           | -           |
| <b>Total</b>    | <b>0</b> | <b>60</b> | <b>3377</b> | <b>2250</b> |
| Available       | 280      | 220       | 106400      | 53200       |
| Utilization (%) | 0        | 27        | 3           | 4           |

# Selected Resource Estimates

| Component Name               | Size             | BRAM18     | DSP48     | FF           | LUT          |
|------------------------------|------------------|------------|-----------|--------------|--------------|
| Fully Connected Layer        | 40 x 40          | 8          | 8         | 592          | 1128         |
| <b>Fully Connected Layer</b> | <b>784 x 256</b> | <b>193</b> | 8         | 647          | 1222         |
| IQ Convolution               | 2 x 8 x 128      | 0          | <b>64</b> | <b>13440</b> | <b>37542</b> |
| Sigmoid                      | 100              | <b>1</b>   | <b>0</b>  | <b>56</b>    | <b>182</b>   |
| Maxpool (size 2, stride 2)   | 128 x 64         | 2          | 0         | 77           | 252          |

## Key Notes:

1. Dense fully connected layers require too much BRAM for E310/E312
2. Large IQ convolution runs a high on FFT/LUT. Could be optimized more
3. Sigmoid/Maxpool essentially free

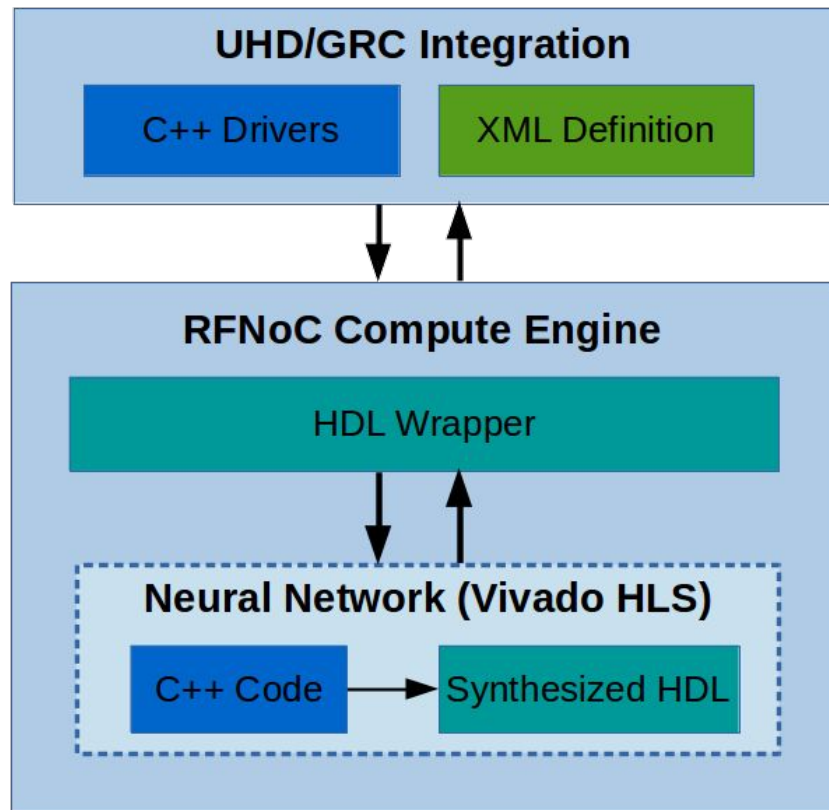
# How To Use: HDL

Insert synthesized verilog module into an RFNoC Compute Engine (CE)

Convenient verilog wrapper interacts with synthesized HDL to format RFNoC packet sizes correctly:

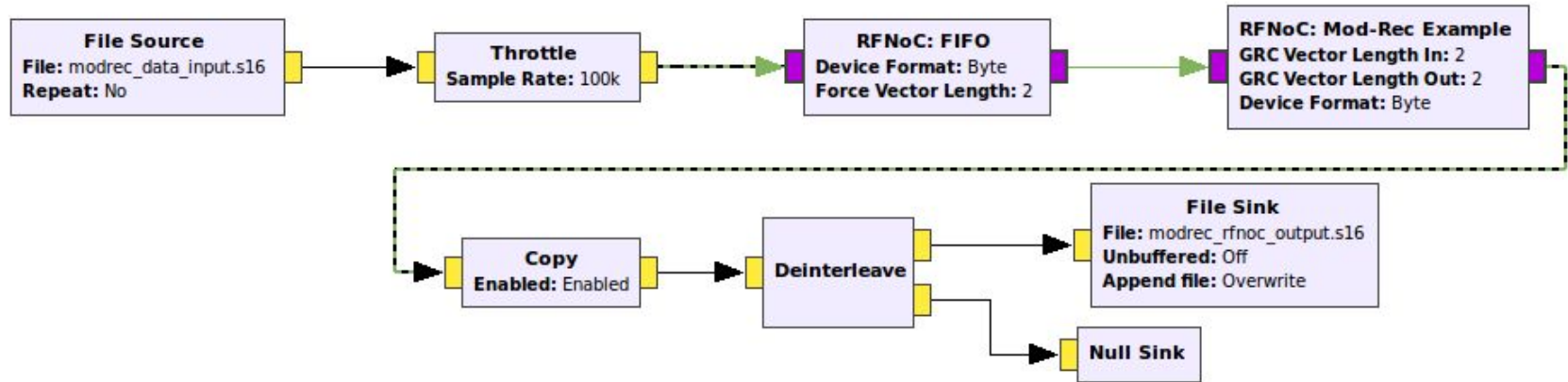
“nnet\_vector\_wrapper.v”

**Run RFNoC testbench to simulate HDL CE**



# How To Use: Gnuradio Companion

RFNoC CEs designed to use the “default” gnuradio ettus block definition. No custom C++ driver code required. Some data type wrangling needed



**Gnuradio + UHD manages software interface for X300 and E310 FPGAs**

# Examples

Four examples provided as inspiration:

1. One-layer image classification based on Udacity's machine learning course
  - a. HLS implementation
  - b. RFNoC HDL and GRC
2. Two layer image classification based on Udacity's machine learning course
  - a. HLS implementation ONLY
3. RF modulation recognition using a set of expert features
  - a. HLS implementation
  - b. RFNoC HDL and GRC
4. Convolutional network for modulation recognition of streaming IQ data
  - a. HLS implementation ONLY

# Future (Potential) Improvements

- Additional neural network types
  - 2D convolution, recurrent networks
- Programmable weights
  - Currently weights may only be hardcoded into HLS C++ code
- Improve weight storage
  - Use larger FPGA BRAMs
  - External/off chip RAM?
- Optimize existing neural network blocks for wider applications
- Alternate neural network architectures (binarized, etc)

**Will prioritize based on feedback...**



# Live Demo

Modulation recognition on expert features

**Thank You!**