Conversation with Xilinx Research Labs

CTO Organization
October 2018
Xilinx Research Mission

Drive technology innovation to increase growth and value of Xilinx

➤ Enable New Users
➤ Open New Markets
➤ Create New Value
➤ Win Mindshare of Innovators
Research: Pathfinding & Differentiation

Core

Context

Risk Avoidance

Mission Critical

Non-Critical

Engineering
Execute

Innovate

Outsource

3rd Party

Research Labs

ACADEMIA
STARTUPS
Xilinx University Program  Mission

Empower academic teaching, research, and entrepreneurship with Xilinx technologies

> Provide support for teaching, training professors, workshops, hackathons
> Give students access to our latest technology
> Enable new business and technical opportunities through research partnerships
Our Corporate Mission

Building the Adaptable Intelligent World
Our Strategy

Focus on Data Center Opportunity

Accelerate Core Markets Growth

Build Adaptive Computing Platform
Adaptive Computing

© Copyright 2018 Xilinx
Data Center

Devices

Production Boards
Compute, Networking, Storage

FPGA as a Service (FaaS)
Enabling Software Developers

Open Frameworks

- TensorFlow™
- gatk
- FFmpeg

Accelerated Libraries

- Machine learning
- HEVC
- Database analytics
- OpenCV

Development Environment

SDAccel™ Environment

Software Application Developers

Development Stack

System Developers

Development Boards
Discussion Topics

> PYNQ : Python Productivity on Zynq

> FINN : Software framework for reduced precision Neural Networks

> Programming SmartNIC using the P4 language and NetFPGA

> RapidWright : A framework for fast and efficient implementation of modular design

> Engaging with Xilinx University Program
Python Productivity for Zynq

Presented By

Patrick Lysaght
Senior Director
1st Oct 2018
Overview

- More productivity
- Enabling technologies
- Open source
- PYNQ
- Next steps
- Research opportunities
New users are not always hardware designers, or embedded systems designers.

Enable more people to program Xilinx processing platforms, more productively.

AND

Offers more rapid development for h/w designers and embedded s/w engineers.
Python is increasingly the language of choice

Top Programming Languages, IEEE Spectrum, July’18

<table>
<thead>
<tr>
<th>Language Rank</th>
<th>Types</th>
<th>Spectrum Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Python</td>
<td></td>
<td>100.0</td>
</tr>
<tr>
<td>2. C++</td>
<td></td>
<td>98.4</td>
</tr>
<tr>
<td>3. C</td>
<td></td>
<td>98.2</td>
</tr>
<tr>
<td>4. Java</td>
<td></td>
<td>97.5</td>
</tr>
<tr>
<td>5. C#</td>
<td></td>
<td>96.8</td>
</tr>
<tr>
<td>6. PHP</td>
<td></td>
<td>95.4</td>
</tr>
<tr>
<td>7. R</td>
<td></td>
<td>93.3</td>
</tr>
<tr>
<td>8. JavaScript</td>
<td></td>
<td>92.8</td>
</tr>
<tr>
<td>9. Go</td>
<td></td>
<td>76.7</td>
</tr>
<tr>
<td>10. Assembly</td>
<td></td>
<td>74.5</td>
</tr>
</tbody>
</table>

Python is listed as an embedded language for the first time

Standard Python comes with comprehensive libraries but also has a huge external ecosystem

152,480 projects 1,079,748 releases 1,477,547 files 263,910 users

The Python Package Index (PyPI) is a repository of software for the Python programming language.
PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.
Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.


Python is the fastest growing language: driven by data science, AI, ML and academia
Jupyter Notebooks to JupyterLab IDE

Code editor

Terminal

Jupyter notebooks

Visualization

2017 ACM Software System Award

Jupyter ... Julia, Python, R

Default engine of data science

2+ million GitHub notebooks

Taught to 1,000+ Berkeley students every semester

Next-gen browser IDE

Includes Jupyter Notebooks

© Copyright 2018 Xilinx
Python productivity for Zynq

Jupyter notebooks, browser-based interface

PYNQ enables JupyterLab on Zynq and ZU+

Jupyter web server

IPython kernel

Ubuntu-based Linux

ARM A53s

Overlays/designs

ZU+ Fabric

Most competitive, open and advanced GUI

Runs natively on ARM processors
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, eg for PS-PL interfaces
PYNQ provides Linux drivers for PS-PL interfaces ... wrapped in Python libraries
PYNQ is a Framework

- **Apps**
  - Jupyter/IPython
    - PYNQ notebooks
    - matplotlib
    - numpy
    - scikit-learn
    - opencv
  - Python
    - PYNQ packages
    - dma
    - XLNK
  - Overlay
    - PL
    - GPIO
    - Interrupt
    - MMIO
    - libsdso.so
  - Linux kernel
    - xdevcfg
    - sysgpio
    - uio
    - devmem
    - xlnk
  - FPGA
    - User designs
    - PYNQ CPUs
    - PYNQ overlays

- **APIs**
  - PYNQ IPs

- **Drivers**
  - xlnk

- **Bitstreams**
  - User designs
  - PYNQ overlays

© Copyright 2018 Xilinx
Software-style packaging & distribution of designs
Enabled by new *hybrid packages*

Download a design from GitHub with a single Python command:

```
pip install git+https://github.com/Xilinx/pynqDL.git
```
Load the downloaded resizer design into Zynq

Zynq

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

PYNQ automatically configures many design parameters based on data parsed from hybrid package
Software only re-sizing

Hardware accelerated re-sizing
Activity Snapshots

DAC Contest

2018 DAC System Design Contest on Low Power Object Detection

61 registered teams

PYYNQ Z1 Zynq 7020 Performance

<table>
<thead>
<tr>
<th>Team Name</th>
<th>Idle Power (mW)</th>
<th>FPS</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGIF</td>
<td>0.623798</td>
<td>4200</td>
<td>11.9556</td>
</tr>
<tr>
<td>SystemsIETI</td>
<td>0.491326</td>
<td>2456</td>
<td>20.8678</td>
</tr>
<tr>
<td>Umit5</td>
<td>0.5743</td>
<td>2590</td>
<td>7.4686</td>
</tr>
</tbody>
</table>

Nvidia TX2 Pascal Performance

<table>
<thead>
<tr>
<th>Team Name</th>
<th>Idle Power (mW)</th>
<th>FPS</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICT-CAS</td>
<td>0.6878</td>
<td>12077</td>
<td>24.55</td>
</tr>
<tr>
<td>Dec2</td>
<td>0.6961</td>
<td>13271</td>
<td>35.30</td>
</tr>
<tr>
<td>SDU-legend</td>
<td>0.6847</td>
<td>3008</td>
<td>23.64</td>
</tr>
</tbody>
</table>
Next steps: scaling across Platforms and Domains
New PYNQ-Z2 Board available now

- New PYNQ reference platform
- New stereo audio with on-board codec
- New Raspberry Pi connector
- Open source design
- Manufactured by TUL in Taiwan
- Distributed by Newark & Newegg
- Academic discounts & donations available

$119 to everyone in US
New Research Opportunities: RFSoC and JupyterLab

- State-of-the-art BIG DATA analysis
- State-of-the-art BIG DATA interactive visualization
- Opportunities: ML and SDR
- Cognitive & agile radios
**Edge-to-cloud SDR with Machine Learning**

**Inner loop/s:**
- Real-time control
- Localized ML
- Local communication between nodes

**Outer loop:**
- Heavy duty ML
- Aggregated across edge nodes
- Longer timescale ML

‘No future SDR will be complete without machine learning’
Edge-to-cloud Co-design Opportunities

Common JupyterLab tools 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

© Copyright 2018 Xilinx
pynq.io

pynq.readthedocs.org

github.com/Xilinx/PYNQ

tul.com.tw/ProductsPYNQ-Z2.html

pynq.io/support
What is PYNQ?

PYNQ is an open-source project from Xilinx® that makes it easy to design embedded systems with Xilinx Zynq® Systems on Chip (SoCs). Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors in Zynq to build more capable and exciting embedded systems. PYNQ users can now create high performance embedded applications with:

- parallel hardware execution
- high frame-rate video processing
- hardware accelerated algorithms
- real-time signal processing
- high bandwidth I/O
- low latency control

Who is PYNQ for?

PYNQ is intended to be used by a wide range of designers and developers in:

- Software developers who want to take advantage of the capabilities of their Zynq design.
- System architects who want an easy software interface and framework of their Zynq design.
- Hardware designers who want their designs to be used by the widest possible audience.

PYNQ Introduction

Xilinx® makes Zynq® and Zynq Ultrascale+™ devices, a class of programmable System-on-Chip (SoC) which integrates a multi-core processor (Dual-core ARM® Cortex®-A9 or Quad-core ARM® Cortex®-A53) and a Field Programmable Gate Array (FPGA) into a single integrated circuit. FPGAs, or programmable logic, and microprocessors are complementary technologies for embedded systems. Each meets distinct requirements for embedded systems that the other cannot perform as well.

Project Goals

The main goal of PYNQ, Python Productivity for Zynq, is to make it easier for designers of embedded systems to exploit the unique benefits of Xilinx devices in their applications. Specifically, PYNQ enables architects, engineers and programmers who design embedded systems to use Zynq devices, without having to use ASIC-style design tools to design programmable logic circuits.

PYNQ achieves this goal in three ways:

- Programmable logic circuits are presented as hardware libraries called overlays. These overlays are analogous to software libraries. A software engineer can select the overlay that best matches their application. The overlay can be accessed through an application programming interface (API). Creating a new overlay still requires engineers with expertise in designing programmable logic circuits. The key difference however, is the build once, re-use many times paradigm. Overlays, like software libraries, are designed to be configurable and re-used as often as possible in many different applications.
Machine Learning: FINN

Presented By

Kees Vissers
Fellow
October 1, 2018
Increasing Range of Applications use Machine Learning

- **Image Classification**
- **Object Detection**
- **Semantic Segmentation**

**Computer Vision**

CNNs

- **Speech Recognition**
- **Speaker Diarization**
- **Speech Recognition**

RNNs, LSTMs

**Natural Language Processing**

- **Translation**
- **Sentiment Analysis**

Sequence to sequence

- **Others**
- **Recommender**
- **GamePlay**

© Copyright 2018 Xilinx
Popular Neural Networks

Computer Vision
- CNNs
- Image Classification
  - ResNet50, VGG, AlexNet, InceptionV3
- Object Detection
  - Faster R-CNN, Yolo9000, YoloV2
- Semantic Segmentation
  - Sedan
  - Motorcycle: 0.005
  - Truck: 0.005
- Speech Recognition
  - RNNs, LSTMs
  - DeepSpeech2
- Natural Language Processing
  - Sequence to sequence
    - Seq2Seq, Transformer
  - Translation
  - Sentiment Analysis
    - Seq-CNN
- Others
  - Recommender
    - NCF
  - GamePlay
    - MiniGo, DeepQ, A3C

© Copyright 2018 Xilinx
The use of Frameworks for inference on FPGAs

- Datasets (e.g. Imagenet)
- Neural Networks (Algorithms) (e.g. Resnet50)
- Neural Networks Inference
  - implementation on FPGA with Reduced Precision
- Neural Networks Training
  - Training for Reduced Precision
Architectures for Deep Learning

DPU: Deep Learning Processing Unit

CPUs

GPUs

Soft DPUs (FPGA)

Hard DPUs (ASIC)

Vector-based SIMD processors becoming increasingly customized for Deep Learning (Tensor Cores, Reduced Precision, …)
Range of architectural solutions for Inference on FPGA

Systolic array processor architecture
- XDNN
- Deephi DPU

Dataflow
- Direct synthesis of the Neural Network
- FINN, HLS based research solution

Intermediate variations

Weights, Thresholds

Layer buffers

Optimal resource use
- Compiler + processor
- IP is labor intensive

Different Systolic architectures
- e.g. Deephi for CNN and RNN, LSTM

Weights, Thresholds

Scalable for devices
- Flexible for bit-precisions
- On-chip memory limits
DPU Specialization

**Xilinx DPU**
- XDNN (Cloud/CNN)
- Aristotle (Edge/CNN)
- Descartes (Edge/Cloud LSTM)

[https://github.com/Xilinx/ml-suite](https://github.com/Xilinx/ml-suite)
[http://deephi.com](http://deephi.com)

**Spectrum of Options**

**Xilinx FINN**
[https://github.com/Xilinx/FINN](https://github.com/Xilinx/FINN)
[https://github.com/Xilinx/BNN-PYNQ](https://github.com/Xilinx/BNN-PYNQ)
[https://github.com/xilinx/LSTM-PYNQ](https://github.com/xilinx/LSTM-PYNQ)

Using Pynq see
[http://pynq.io/ml](http://pynq.io/ml)

Focus on direct Implementation
*With reduced Precision*

Open Source
FINN: Reduced Precision and retraining

Reduce Precision

Retraining

Notation: 3b/5b: 3 bit weights/ 5 bit activation
FINN: Design Space Exploration

**IMAGENET CLASSIFICATION TOP5% VS COMPUTE COST F(LUT,DSP)**

- 1b weights
- 2b weights
- 5bit weights
- 8bit weights
- FP weights
- minifloat
- ResNet-50
- Syq

**Resnet18**
8b/8b
Compute Cost 286
Error 10.68%

**Resnet50**
2b/8b
Compute Cost 127
Error 9.86%

Pareto-optimal solutions
FINN Concepts dataflow implementation of Neural Networks

• Each layer custom parameters
  • Number of Bits
  • Scaling factor
  • Folding factor
One hardware layer per BNN layer, parameters built into bitstream,

Design for balanced throughput
  ➤ Allocate compute resources according to FPS and network requirements

Streaming: Maximize throughput, minimize latency
  ➤ Overlapping computation and communication, batch size = 1, sliding windows between the layers
FINN: for binarized neural networks, training in Caffe

Ingest Neural Network Topology and Weights

Streamline Network Topology and Scale Hardware Performance Model

Generate and Synthesize Hardware bit file

Deploy to target Platform

Target Platform Descriptor File

Caffe

```
layer {
  name: "conv1"
  type: "BinaryConvolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 20.0009
    decay_mult: 1
  }
  convolution_param {
    num_output: 64
    pad: 0
  }
}
```

© Copyright 2018 Xilinx
BNN- PYNQ: reduced precision networks + training

Theano and BinaryNet
- Training Scripts available for network examples
- How-to set-up the training environment
- Support for arbitrary precision on Theano

Network parameters exporting
- Python scripts to export weights and generate thresholds @ target precision
- Loaded at run-time

Network description in VHLS (C++)
- Relying on a validated HLS library
- Including all most common layers
- Examples available for multiple networks and multiple precisions

Bitstream generation
- Scripts targeting multiple supported platforms (PYNQ-Z1, PYNQ-Z2, Ultra96)

Runtime for PYNQ platforms
- Stack of SW, wrapped in a python class easy to use in jupyter environment
LSTM – PYNQ: environment for LSTM with training

PyTorch
- Pytorch-ocr
  • Small training library for OCR with Pytorch using a LSTM-based network
  • Can be extended to different OCR datasets
  • Export of a trained quantized network’s description and weights
- Pytorch-quantization
  • Support for training of quantized LSTMs and FC at arbitrary multi-bit precision
  • Low-level export API of quantized layers
- Network description in VHLS (C++)
  • HLS library for BiLSTM acceleration
  • Takes in network’s file generated by Pytorch
- Bitstream generation
  • Scripts targeting PYNQ-Z2
  • Baked in per-network weights and config
- Runtime for PYNQ platforms
  • Stack of SW, wrapped in a python class easy to use in jupyter environment
Repositories

- https://github.com/Xilinx/FINN
- https://github.com/Xilinx/BNN-PYNQ
- https://github.com/xilinx/LSTM-PYNQ
- https://github.com/Xilinx/pytorch-quantization
- https://github.com/Xilinx/pytorch-ocr
- https://github.com/Xilinx/QNN-MO-PYNQ
- http://www.pynq.io
- http://www.pynq.io/ml
- https://github.com/Xilinx/ml-suite
- http://www.ultra96.org
- http://deephi.com
Publications

- FPL’18: **FINN-L**: Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs
- FPL’18: **BISMO**: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing
- FPL’18: **Customizing Low-Precision Deep Neural Networks For FPGAs**
- ACM TRETS, Special Issue on Deep Learning: **FINN-R**: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks
- ARC’18: **Accuracy to Throughput Trade-Offs for Reduced Precision Neural Networks on Reconfigurable Logic**
- CVPR’18: **SYQ**: Learning Symmetric Quantization For Efficient Deep Neural Networks
- DATE’18: **Inference of quantized neural networks on heterogeneous all-programmable devices**
- ICONIP’17: **Compressing Low Precision Deep Neural Networks Using Sparsity-Induced Regularization in Ternary Networks**
- ICCD’17: **Scaling Neural Network Performance through Customized Hardware Architectures on Reconfigurable Logic**
- PARMA-DITAM’17: **Scaling Binarized Neural Networks on Reconfigurable Logic**
- FPGA’17: **FINN**: A Framework for Fast, Scalable Binarized Neural Network Inference
- H2RC’16: **A C++ Library for Rapid Exploration of Binary Neural Networks on Reconfigurable Logic**
Conversation with Xilinx Research Labs: P4 and NetFPGA

Presented By
Gordon Brebner
Distinguished Engineer
1 October 2018
Open source programmable networking on FPGA

P4 programming language for packet processing

NetFPGA platform for line-rate packet processing

Automated workflow for running P4 on NetFPGA
Benefits of Programmable Networking (… or FPGA in fact)

- Control and Customization. Make switch or SmartNIC behave exactly as you want
- Reliability. Reduce risk by removing unused features
- Efficiency. Reduce energy consumption and expand scale by doing only what you need
- Add new features on your schedule
- Telemetry. Be able to see inside the network
- Exclusivity and Differentiation. Add secret sauce to vendor offerings
P4 Programming Protocol-independent Packet Processors

- Language first appeared in paper published in July 2014

- Three goals:
  - Reconfigurability in the field – reprogramming of networking equipment
  - Protocol independence – not tied to any specific networking protocols
  - Target independence – not tied to any specific networking hardware

- P4 consortium (P4.org) set up in 2015 – now an open source Linux Foundation project
  - Xilinx was a founding member of P4.org
  - Now has >100 members

- P4 has emerged as the *de facto* standard language for packet processing
P4 language elements

- **Parsers**
  - State Machines, bit-field extraction

- **Controls**
  - Match-Action Tables, control flow statements

- **Expressions**
  - Basic operations and operators

- **Data Types**
  - Bit-strings, headers, structures, arrays

- **Architecture**
  - Programmable blocks and their interfaces

- **Extern Libraries**
  - Support for specialized components

Packet processing pipeline
P4 “Hello World” (networking style) example

```c
#include <core.p4>
#include <XilinxSwitch.p4>

struct user_meta_t {}
struct headers {}

parser MyParser(packet_in packet, out headers hdr, inout user_meta_t meta, inout std_meta_t std_meta) {
    state start { transition accept; }
}

control MyPipe(inout headers hdr, inout user_meta_t meta, in std_meta_t std_meta) {
    action set_egress_port(bit<9> port) {
        std_meta.egress_port = port;
    }

    table forward {
        key = { std_meta.ingress_port: exact; }
        actions = {
            set_egress_port;
            NoAction;
        }
        size = 1024;
        default_action = NoAction();
    }

    apply { forward.apply(); }
}

control MyDeparser(packet_out packet, in headers hdr) {
    apply {} 
}

XilinxSwitch( MyParser(), MyPipe(), MyDeparser() ) main;
```

<table>
<thead>
<tr>
<th>Key</th>
<th>Action Name</th>
<th>Action Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>set_egress_port</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>set_egress_port</td>
<td>1</td>
</tr>
</tbody>
</table>

© Copyright 2018 Xilinx
P4 ecosystem

User-developed

- Application
- Application
- Application
- Application

Data plane: P4 and Control plane: C, Python, etc.

Community-developed

- P4 Language
- P4 Core Library

Vendor-supplied

- Architecture Definition
- Extern Libraries
- P4 Compiler
Programming and operating a P4 platform

- **P4 Program**
- **P4 Architecture Model**
- **P4 Compiler**
- **Target-specific configuration binary**
- **Load**
- **User supplied**
- **Control Plane**
  - Add/remove table entries
  - Extern control
  - Packet-in/out
- **Data Plane**
  - Tables
  - Extern objects
  - CPU port
- **Vendor supplied**

Platform
Xilinx P4-SDNet product (www.xilinx.com/sdnet)

Xilinx Labs prototype (May 2017):
• First-ever P4-2016 compiler
• 100G line rate
Production version (Dec 2018):
• 50% less latency and resources

Xilinx P4 Compiler

% sdnet example.p4

Verification Environment

Top level Verilog wrapper
Verilog components
System Verilog testbench
High level C++ testbench
Run time drivers

Example target: Xilinx P4-Smart NIC card

© Copyright 2018 Xilinx
P4-SDNet research community today:
60 institutions in 22 countries

Bosnia: 1
France: 2
Germany: 4
Ireland: 1
Italy: 3
Poland: 1
Romania: 1
Russia: 1
Serbia: 1
Spain: 3
Sweden: 1
Switzerland: 2
UK: 4

Canada: 2
USA: 13

Brazil: 3

China: 7
India: 1
Israel: 1
Japan: 1
South Korea: 2
Taiwan: 5
NetFPGA (= Networked FPGA)

- Line-rate, flexible, open networking platform for teaching and research
- Community began with Stanford and Xilinx Labs, now anchored at Cambridge
- NetFPGA systems deployed at over 150 institutions in over 40 countries

Four elements:
- Community: NetFPGA.org
- Low-cost board family
- Tools and reference designs
- Contributed projects
NetFPGA SUME reference switch design

- 4x10G Ethernet switch, with CPU slow path as 5th port

- Five-stage pipeline:
  - Input ports
  - Input arbitration
  - Forwarding decision and packet modification
  - Output queuing
  - Output ports

- Standard module interfaces
- Modules can be customized or substituted
P4 → NetFPGA workflow overview

NetFPGA SUME reference switch design

Drop-in substitute

© Copyright 2018 Xilinx
P4→NetFPGA workflow steps

1. Write P4 program
2. Write additional externs (if required)
3. Write python test packet script
4. Compile to Verilog / Generate API & CLI tools
5. Run simulations to test/debug
6. Build bitstream
7. Check implementation reports
8. Test on the hardware

All of the user effort goes here – not on the FPGA detail

Iterate
Implement
P4→NetFPGA community

> 150 active members from academia and industry around the world, and growing

> Members of the community highly encouraged to contribute in ways such as:
  
  >> New P4 projects
  >> Extra extern functions
  >> Performance analysis tools
  >> Verification tools

> Used for research, and for teaching networking concepts on real hardware
  
  >> No hardware design experience needed

> Some current projects
  
  >> Distributed congestion control
  >> In-band Network Telemetry
  >> In-network compression; In-network key-value cache
  >> Network-accelerated sorting; Network-accelerated consensus

> Getting started:
  
  >> Public documentation: https://github.com/NetFPGA/P4-NetFPGA-public/wiki
Research directions

> Language: Extend coverage of P4
  >> Programmable Traffic Management (MIT + NYU + Stanford + Xilinx Labs + P4.org)
  >> Programmable Target Architectures (Cornell + Stanford + Xilinx Labs)

> Infrastructure: Open source hardware reference platform for P4
  >> Complement existing software reference platform
  >> Cover NIC-style architectures as well as switch-style architectures

> Applications
  >> Programmable networking offload and acceleration
  >> Congestion control, in-band network telemetry
  >> In-network computing
  >> … your ideas here
RapidWright\(^1\): Modular pre-implemented methodology

Presented By

Alireza Kaviani, Ph.D.
Distinguished Engineer,
Xilinx Research Labs
Oct 1, 2018

\(^1\) Wright = maker or builder
RapidWright value proposition

- **IMPLEMENTATION COMPILE TIME**
  - CPUs
  - FPGAs + Pre-implemented blocks
  - FPGAs + Shells
  - FPGAs
  - ASICs
  - SDx

© Copyright 2018 Xilinx
Focus on emerging applications

> Module-based approach to implementation
  > Lock-in performance with reusable modules
  > Fewer inter-block timing closure issues

> Advantages
  > > 10X reduction in compile time
  > > Near-spec performance
  > > Predictable timing closure
Vision: Rapid accelerator assembly

<table>
<thead>
<tr>
<th></th>
<th>RAPID WRIGHT</th>
<th>VIVADO</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation Time</td>
<td>~ 6 mins</td>
<td>~1-3Hours</td>
<td>10-30X</td>
</tr>
<tr>
<td>Accelerator Fmax</td>
<td>~700MHz</td>
<td>~450MHz</td>
<td>~1.5X</td>
</tr>
</tbody>
</table>

© Copyright 2018 Xilinx
RapidWright overview

> Enables targeted solutions
  >> Reuse & relocate pre-implemented modules
  >> Just-in-time implementations
  >> Create shells & overlays

> Companion framework for Vivado
  >> Communicates through Design CheckPoints¹ (DCPs)
  >> Fast, light-weight, open source
  >> Java, Python coding

> Power user ecosystem
  >> Academic algorithm validation
  >> Rapid prototyping of CAD concepts

¹: DCP contains netlist + P&R info + constraints
A Modular pre-implemented methodology

**DOMAIN DESIGN TASKS**

1. Design selection attributes:
   - Modular
   - Latency tolerant
   - Prefers replication

2. Placement planning

**IMPLEMENTATION TOOL TASKS**

3. P&R modules cached:
   - Relocatable
   - Reusable
   - Timing predictable

4. Run implementation
Building relocatable domain-specific shells

Fact

- Advances in silicon have created QoR opportunity

Community role

- Domain-specific shell design or overlays

RapidWright value proposition

- Achieve near-spec performance
RapidWright pre-implemented module flow

User Design

Design Parser

Block Assembler

Block Cache

Block Placer

Route Design

Final Impl.

Fully Placed, Partially Routed Implementation
# Design performance results

<table>
<thead>
<tr>
<th>Design</th>
<th>Target Device</th>
<th>Baseline (initial design)</th>
<th>RapidWright(^1) Flow</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seismic</td>
<td>KU040</td>
<td>270MHz</td>
<td>390MHz</td>
<td>41%</td>
</tr>
<tr>
<td>FMA</td>
<td>KU115</td>
<td>270MHz</td>
<td>417MHz</td>
<td>54%</td>
</tr>
<tr>
<td>GEMM</td>
<td>KU115</td>
<td>391MHz</td>
<td>462MHz</td>
<td>16%</td>
</tr>
<tr>
<td>ML overlay</td>
<td>ZU9EG</td>
<td>368MHz</td>
<td>541MHz</td>
<td>50%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design</th>
<th>LUT</th>
<th>FF</th>
<th>DSP</th>
<th>BRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seismic</td>
<td>93%</td>
<td>5%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FMA (HPC design)</td>
<td>25%</td>
<td>50%</td>
<td>97%</td>
<td>6%</td>
</tr>
<tr>
<td>GEMM</td>
<td>19%</td>
<td>20%</td>
<td>87%</td>
<td>-</td>
</tr>
<tr>
<td>ML overlay</td>
<td>46%</td>
<td>29%</td>
<td>42%</td>
<td>96%</td>
</tr>
</tbody>
</table>

\(^1\): RapidWright: Enabling Custom Crafted Implementations for FPGAs, FCCM 2018
Fully Connected Network (FCN) accelerator

> Fully Connected Network Accelerator (FCN)
  >> GEMM + ReLU (activation function)
  >> BRAM and DSP Utilization higher than 80%
  >> Goal: fit four compute kernels on F1

> Regular Host Interconnect
  >> 2 compute Kernels (@ 200 MHz) fit
    – Three kernels does not route, due to overhead of data movement

> LinkBlaze\(^1\) Host Interconnect
  >> Three kernels (@ 200 MHz) fully placed & routed

1: LinkBlaze: Efficient global data movement for FPGAs., Reconfig 2017
Fully Connected Network (FCN) accelerator

> Fully Connected Network Accelerator (FCN)
  - GEMM + ReLU (activation function)
  - BRAM and DSP Utilization higher than 80%
  - Goal: fit four compute kernels on F1

> Regular Host Interconnect
  - 2 compute Kernels (@ 200 MHz) fit
    - Three kernels does not route, due to overhead of data movement

> LinkBlaze\(^1\) Host Interconnect
  - Three kernels (@ 200 MHz) fully placed & routed
  - 4x Kernels with relocatable modular design will fit

Enabling 33% More Compute

1: LinkBlaze: Efficient global data movement for FPGAs., Reconf 2017
Pre-implimented data movement shell

> Goals
  >> Minimize overhead of compute (and overlays)
  >> Prove shell assembly model

> Build-to-order LinkBlaze shell
  >> 512 bit, bi-directional
  >> RapidWright Pre-implimented modules

<table>
<thead>
<tr>
<th>Vivado</th>
<th>RapidWright</th>
</tr>
</thead>
<tbody>
<tr>
<td>516MHz</td>
<td>620MHz (+20%)</td>
</tr>
</tbody>
</table>
Open Source Community Call for Action
Proposed domain-specific tool flows

- **Developers Driving Design**
  - Data scientists, application architects
  - Academic and industrial community
  - Xilinx and open source community

- **Tools and Frameworks**
  - High abstraction Domain-specific language
  - Domain-specific data flow graph (LLVM) compiler
  - Relocate pre-implemented operators and functions

- **Front-end Compiler**
  - Front-end compiler

- **Back-end Compiler**
  - Back-end compiler

- **Design Entry**

- **Application in Domain 1**
- **Application in Domain 2**
- **Application in Domain 3**

- **VIVADO**

- **XILINX DEVICE**

© Copyright 2018 Xilinx
Domain tool flow example

> Fact
  >> Emerging domains such as surveillance or vision have high replication

> Community role
  >> Identify and extract operators and functions in the domain

> RapidWright value proposition
  >> Assemble relocatable pre-implemented domain operators
  >> Deliver the best inference/watt
Beyond a pre-implemented methodology

- RapidWright probe router enables higher productivity
  - 21X more debug turns per day
  - Highest level of routing preservation possible
  - Future innovation:
    - iteration with extra probe inputs
    - Automatic insertion of pipeline flops to manage timing

<table>
<thead>
<tr>
<th>Vivado</th>
<th>RapidWright</th>
<th>Δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>modify_debug_probes</td>
<td>ProbeRouter</td>
<td></td>
</tr>
<tr>
<td>130 mins</td>
<td>6.3 mins</td>
<td>21X</td>
</tr>
</tbody>
</table>

Original

RapidWright Probes Rerouted

© Copyright 2018 Xilinx
Vision: Pre-implemented modules

Parameterizable Circuit Generators

Algorithmic Engines (SAT Solvers, ILP,...)

Vivado-optimized OOC Solutions

www.rapidwright.io

© Copyright 2018 Xilinx
Presented By

Hugo Andrade
Director, Xilinx University Program
What we do

Empower academic teaching, research, and entrepreneurship with Xilinx technologies
Who we are

➢ Global university engagement
  ➢ Americas/RoW, EMEA, APAC

➢ Americas/RoW and world-wide contacts

Dr. Parimal Patel  Naveen Purushotham  Hugo A. Andrade  Patrick Lysaght
How we can help

Access to tool and IP licenses, academic boards and chips

- Vivado
- HLS
- SDx: SDSoc, SDAccel
- Zynq
- MPSoC, RFSoc
- Ultrascale+

- Research enablement
- Teaching Material
- Reference designs
- Technical Support

- Conferences
- Workshops
- Summer Schools

- Design Contests
- Hackathon
- Startup program
- Cloud access
New Zynq Ultrascale+ MPSoC book with ML

Exploring Zynq® MPSoC
A Multi-Processor System-on-Chip
Featuring ARM® Applications & Real-Time Processors and FPGA Programmable Logic

Coming in 2019
New open source HLS book

Parallel Programming for FPGAs
The HLS Book

Ryan Kastner
Janarbek Matai
Stephen Neuendorffer

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.

http://kastner.ucsd.edu/hlsbook/
Global engagement and collaboration

Democratizing Customizable Computing via Automated Accelerator Generation

Yuze Chi¹, Jason Cong¹,², Jie Wang¹, Peng Wei¹ and Cody Hao Yu¹,²
University of California Los Angeles¹, Falcon Computing Solutions²

Abstract
High-performance & ease-of-programming: we want both
- Designing high-performance accelerators demands a great deal of programming effort
- Transforming software programs directly into FPGA circuits becomes feasible via HLS, but the QoR is horrible without heavy reconstruction of the software code
From C to high-quality HLS-C: dealing with large design space
- A large number of pragmas in many legal insertion points
- Complex performance-resource trade-offs
- Long evaluation time (tens of minutes per design point)
Automated accelerator generation: automatically producing a high-performance design in a reasonable amount of time
- Using machine learning to decrease the number of design points that need to be evaluated
- Using microarchitecture templates to confine the design choices, and, more importantly, enable analytical model-based fast design space exploration

PolySA: Polyhedral-Based Systolic Array Auto-Compilation

Example code of NV1

AutoCompilaton

C/C++

Example code of NV1

[ICCAD '19]

Polyhedral code generator

Virtual System Area

Front-End

Back-End

SODA: Stencil with Optimized Dataflow Arch

- Optimized dataflow arch.
- Accurate performance and resource modeling
- Automatic transformation from SODA DSL, all the way to bitstreams

AutoAccel: Automated Accelerator Generation w/ CPP Microarchitecture

Composable, Parallel, Pipelined [CPP]

- Optimized off-chip communication
- Explicit data caching
- Parallel & pipeline loop scheduling
- Scrapsbased Reorganization

Architecture

Reduce design space
Enable analytical model
Analytical Model
Quantize design traversal
Facilitate fast DSE
Code transformation
From nanosecond to machine time

Pros and Cons of Approaches

Learning-based approach
- Not being constrained to a specific microarchitecture
- Demanding a great deal of time to achieve the optimal solution due to time-consuming design point evaluation

Microarchitecture-based approach
- Coming up with a high-quality design in much shorter time if the input program fits into the microarch well
- May not work well for all kinds of compute domains

References

© Copyright 2018 Xilinx
Key Initiatives

- FPGA-based accelerators in the cloud
- PYNQ: Python productivity for Zynq
Promote & support AWS EC2 F1 in academic community

> An AWS EC2 compute instance with Xilinx FPGAs which can be programmed to create custom hardware accelerated applications

> F1 instances are easy to program and come with everything needed to develop, simulate, debug, and compile hardware accelerators

> Can be registered as an Amazon FPGA Image (AFI) and marketed
Latest technology training at top conferences

FPGA18, Monterey

Expanded conference categories covered

> FPGA
> Computer Architecture and HPC
> Applications
Promote & support AWS EC2 F1 based courses

> UC Berkeley, Prof. Krste Asanović
  >> Computer Architecture & Engineering
  >> “An important part is lab assignments using real microprocessor designs implemented in Chisel, running as simulators and FPGA emulators in the Amazon cloud as F1 instances.”

> Cornell, Prof. Zhiru Zhang
  >> High-level Digital Design Automation

> UCLA, Prof. Jason Cong
  >> Customizable Computing for Big Applications
  >> Parallel and Distributed Computing

> Coursera, Politecnico di Milano, Prof. Marco Domenico Santambrogio
  >> FPGA-accelerated Cloud Applications with SDaccel
Let’s chat further about:

• Opportunities in teaching, research and entrepreneurship using AWS EC2 F1
• How to get AWS credit vouchers
• Hand-on training
Advanced Embedded System Design on Zynq using Vivado

Course Description
This workshop provides professor the necessary skills to develop complex embedded systems using Vivado design suite: understand and utilize advanced development techniques of embedded systems design for architecting a complex system in the Zynq® System on a Chip (SoC).

Level
Intermediate

Duration
2 Days

Who should attend?
Professors who are familiar with embedded system design using Vivado and want to explore advanced design techniques using Xilinx SoC in Zynq.

Pre-requisites
- Digital logic and FPGA design experience
- Experience with Vivado software
- Experience with Embedded System design
- Have attended XUP Embedded System Design workshop or has an equivalent experience

Skills Gained
After completing this workshop, you will be able to:
- Assemble an advanced embedded system
- Explore various features of Zynq SoC for hardware-software co-design
- Design and integrate peripherals using interrupts
- Analyze system performance
- Utilize hardware debugging technique
- Design a bootable system ready for deployment in field

Course Overview

Quick Links
- Workshops Schedule
- Vivado Design Suite
- Vivado-Based Workshops
- ISE Design Suite
- ISE-Based Workshops

2018x Workshop Material

Common to PYNQ-Z1 and PYNQ-Z2
- Labdocs (PDF)
- Lab Source File
- Labdocs and Presentation (docx and pptx)*

PYNQ-Z1
- README
- Board Files (required to do the labs)
- Labsolution*

PYNQ-Z2
- README
- Board Files (required to do the labs)
- Labsolution*
Promote and support PYNQ hackathons

30 hours

45 participants

Industry and academia

Reproducible Hacking with Jupyter Notebooks
Reaching XUP

www.xilinx.com/xup
xup@xilinx.com