Ecole Informatique 2016
IN2P3, Ecole Polytechnique, 27 Mai 2016

OpenCL On FPGA

Marc Gaucheron
INTEL Programmable Solution Group

Agenda

- FPGA architecture overview
- Conventional way of developing with FPGA
- OpenCL: abstracting FPGA away
- ALTERA BSP: abstracting FPGA development
- Examples

A bit of Marketing ?

Efficiency via Specialization

Industry Trends

- Increas
- Smaller
- Shorter
- Limited

Parallel Computing

Need for Parallel Computing

"A form
operat
ones,

Inf

Realiz

Cor

Challenges in Parallel Programming

- Findir
  - Wh
- Data
  - Wh
    tim
- Applic
  - Co
  - Da
  - Co
- Powe

Programmers Dilemma

"The way the processor industry is going, is to add more and more cores, but nobody knows how to program those things. I mean, two yeah; four not really; eight, forget it."

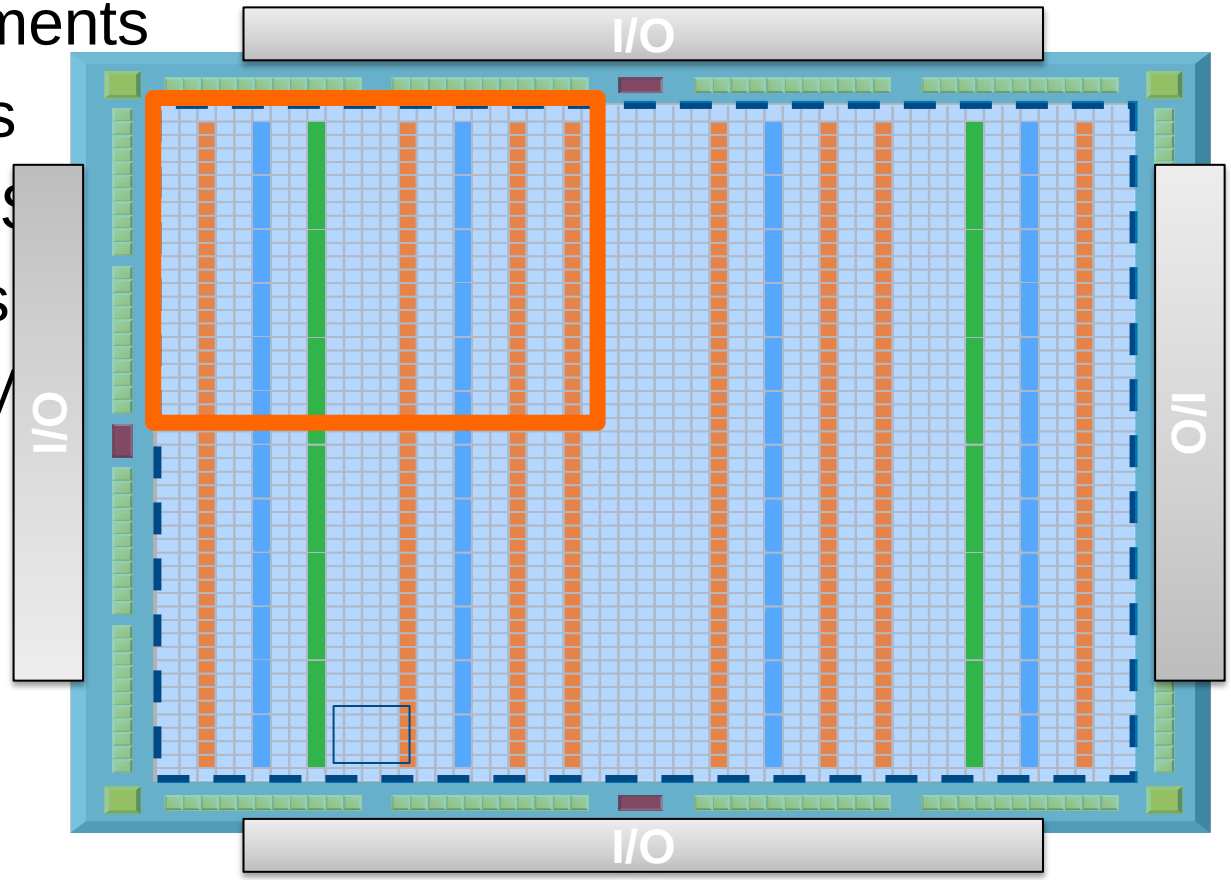
~ Steve Jobs, 1955-2011

FPGA architecture overview

FPGA Architecture: Fine-grained Massively Parallel

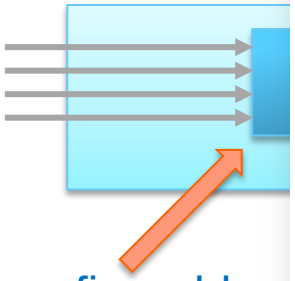
- Millions of reconfigurable logic elements
- Thousands of 20Kb memory blocks
- Thousands of Variable Precision DS
- Dozens of High-speed transceivers
- Multiple High Speed configurable M
- Multiple ARM© Cores

Let's zoom in



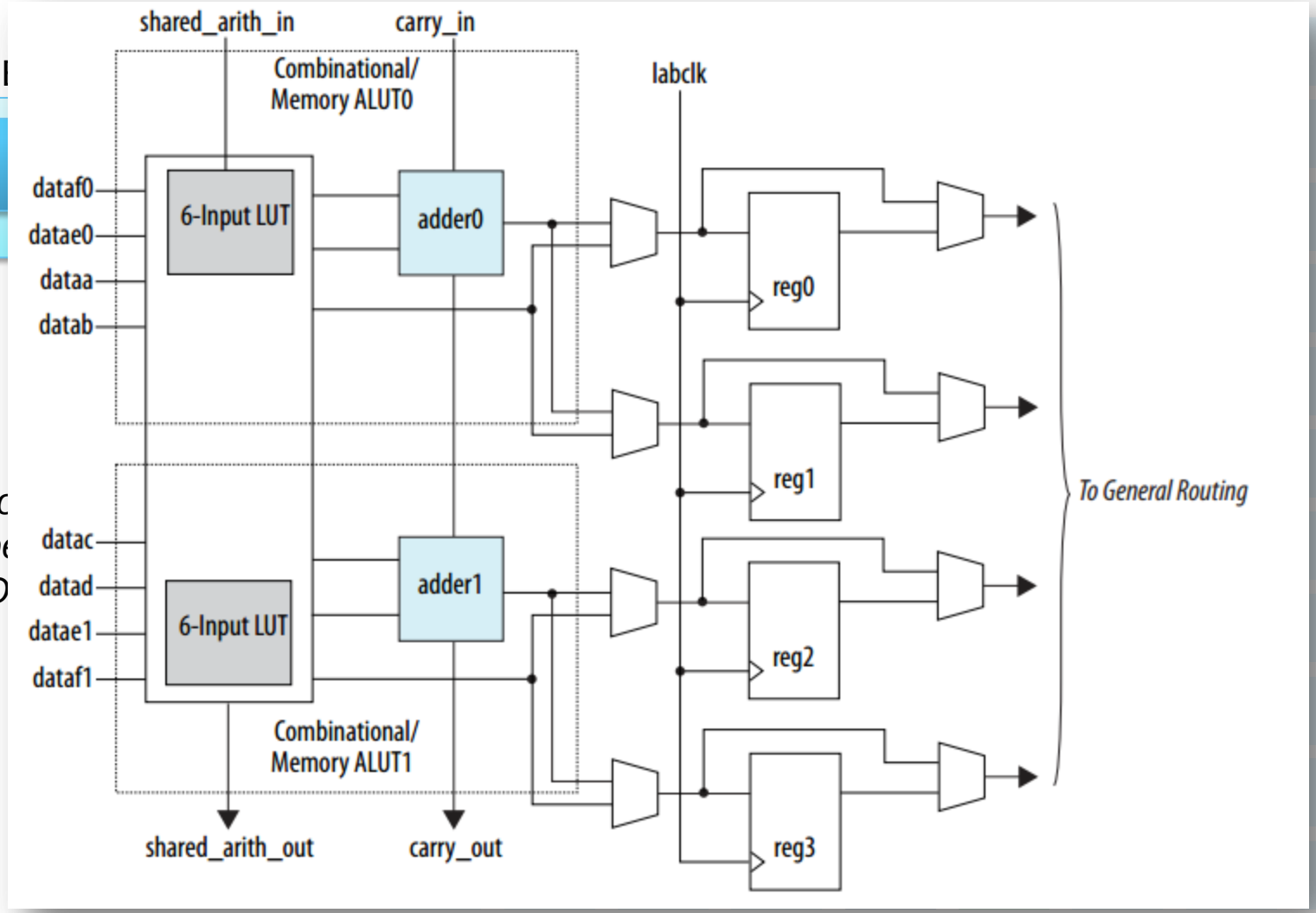
FPGA Architecture: Basic Elements

Basic

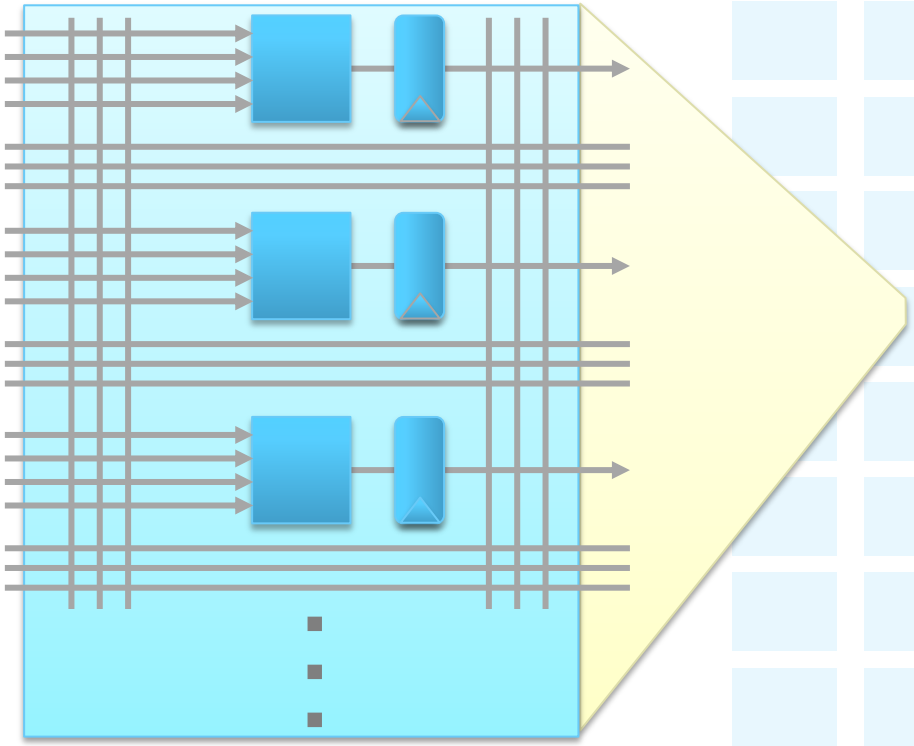


1-bit configurable operation

Configured *to*
1-bit ope
AND, OR, NO

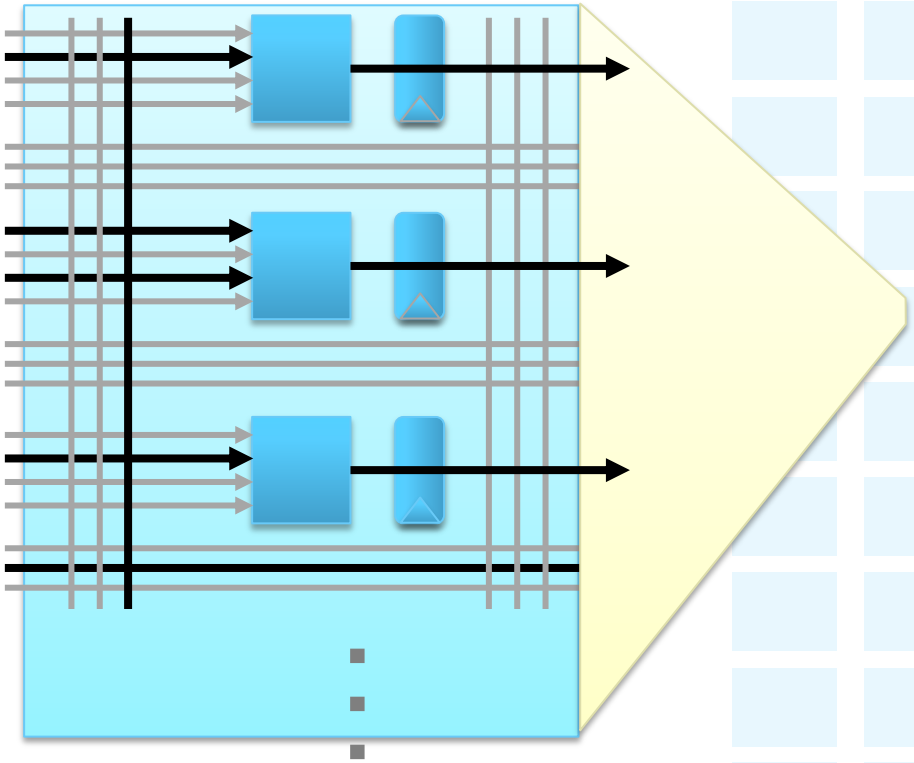


FPGA Architecture: Flexible Interconnect



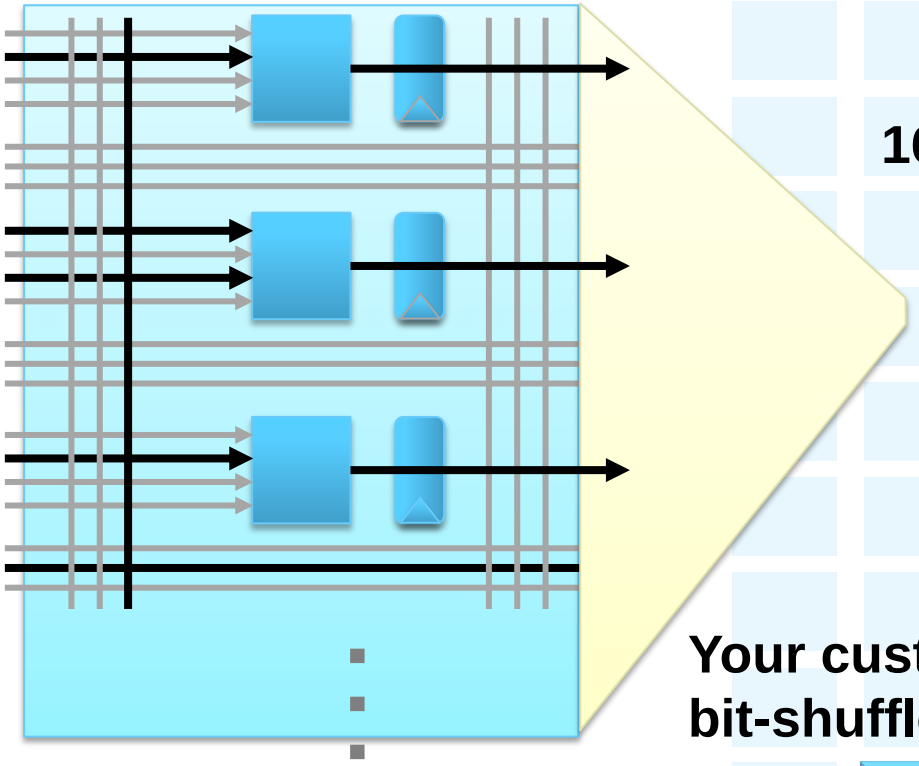
Basic Elements are surrounded with a flexible interconnect

FPGA Architecture: Flexible Interconnect



Wider *custom* operations are implemented by configuring and interconnecting Basic Elements

FPGA Architecture: Custom Operations Using Basic Elements



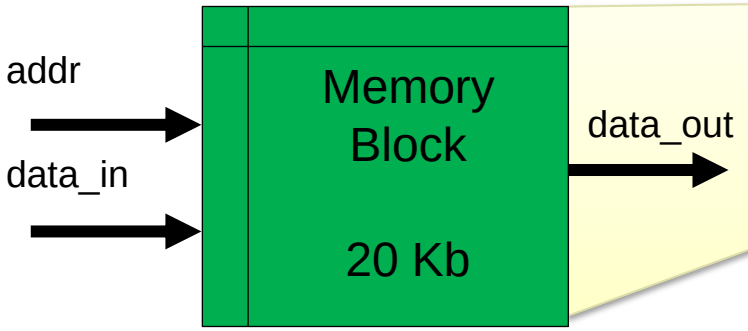
16-bit add

32-bit sqrt

Your custom 64-bit
bit-shuffle and encode

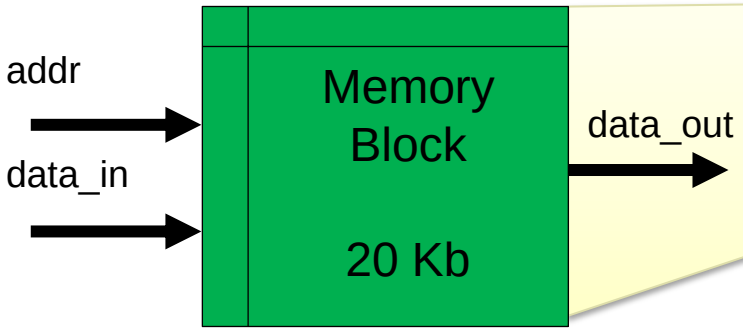
Wider *custom* operations are implemented by configuring and interconnecting Basic Elements

FPGA Architecture: Memory Blocks



Can be configured and grouped using the interconnect to create various cache architectures

FPGA Architecture: Memory Blocks



Can be configured and grouped using the interconnect to create various cache architectures

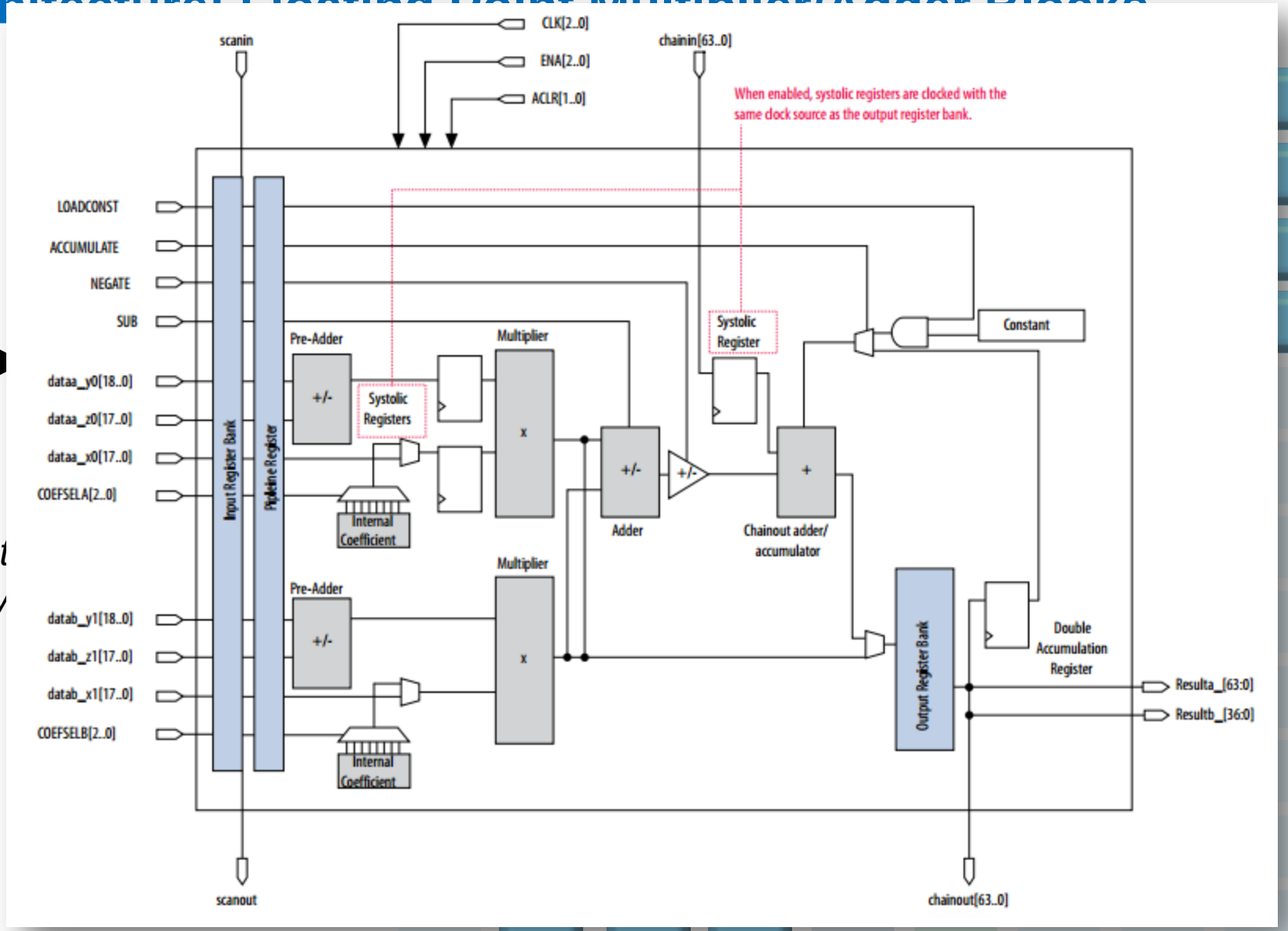
Lots of smaller caches

Few larger caches

FPGA Architecture: Floating Point Multiplier/Adder Blocks

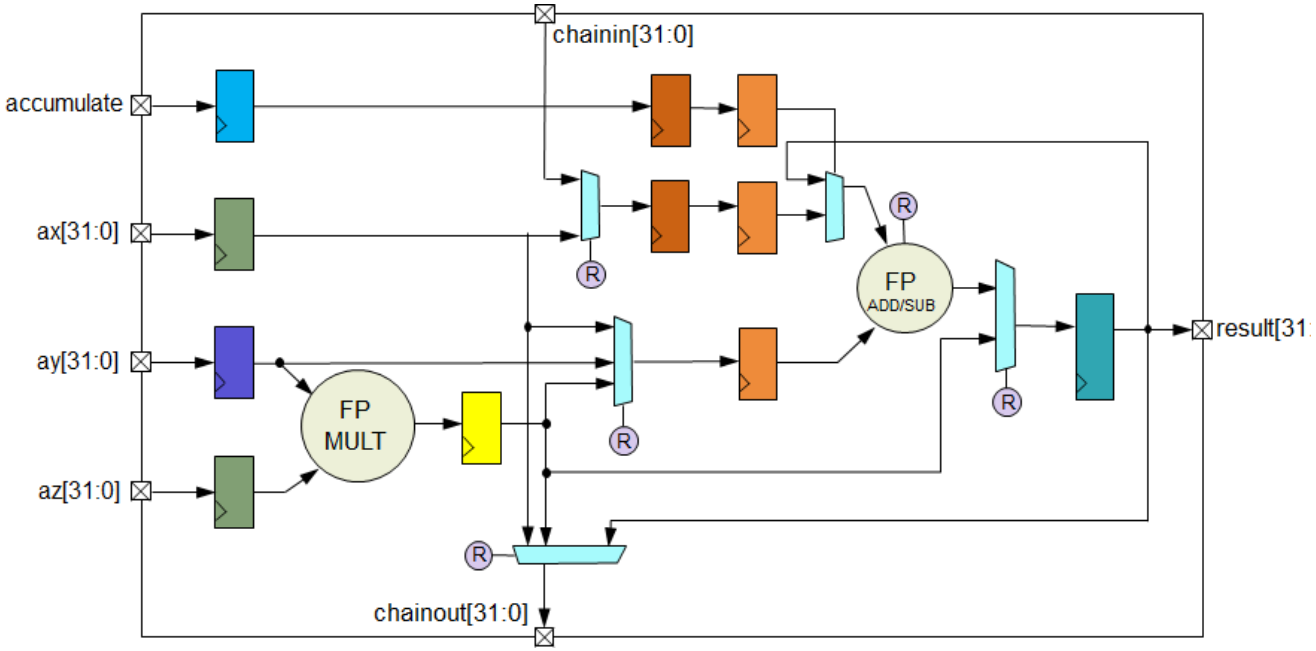
data_in

Dedicat
multiply

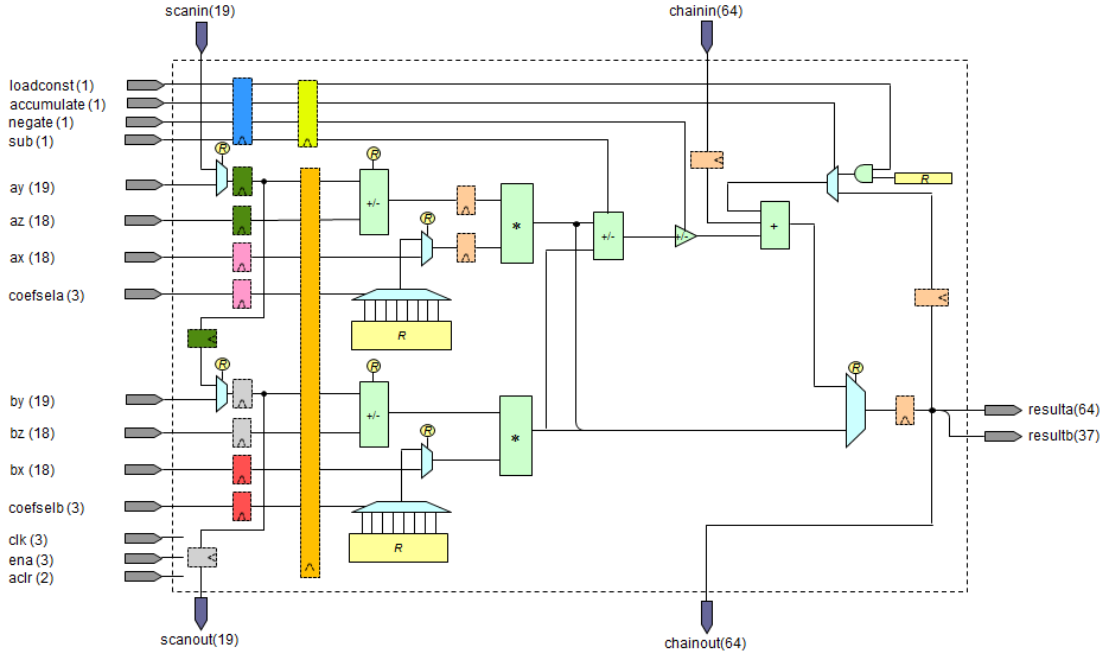


DSP block architecture

Floating-Point DSP

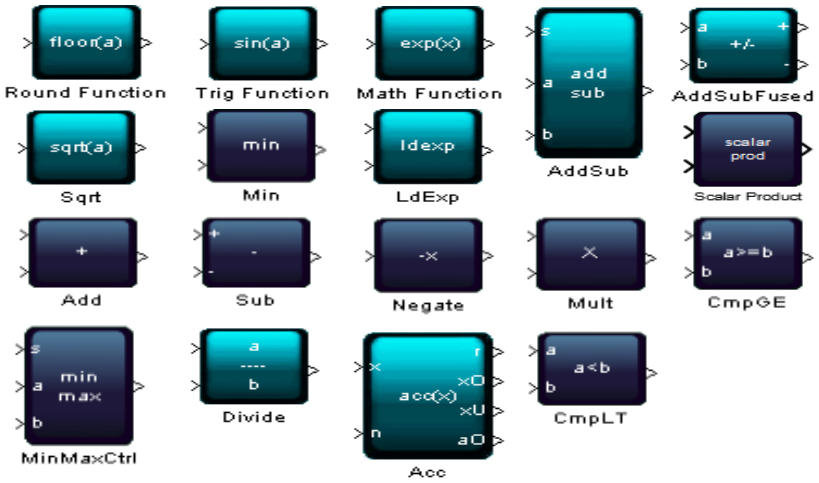


Fixed-Point DSP



Elementary math functions supporting floating point

- Coverage of ~70 elementary math functions
- Patented & published efficient mapping to FPGA hardware
  - Polynomial approximation, Horner's method, truncated multipliers, …
- Compliant to OpenCL & IEEE754 accuracy standards
- Rounding mode options for fundamental operators
- Half- to Double-precision



Fixed and floating point

Floating point only

Trigonometrics of pi*x

FPSinPiX
FPCosPiX
FPTanPiX
FPCotPiX

Exp, Log and Power

FPLn
FPLn1px
FPLog10
FPLog2
FPExp
FPExpFPC
FPExpM1
FPExp2
FPExp10
FPPowr

Trig with argument reduction

FPSinX
FPCosX
FPSinCosX
FPTanX
FPCotX

Inverse trigonometric functions

FPArcsinX\
FPArcsinPi
FPArccosX
FPArccosPi
FPArctanX
FPArctanPi
FPArctan2

Conversion

FXPToFP
FPToFXP
FPToFXPExpert
FPToFXPFused
FPToFP

Macro Operators

FPFusedHorner
FPFusedHornerExpert
FPFusedHornerMulti
FPFusedMultiFunction

Trigonometrics misc

FPHypot
FPRangeReduction

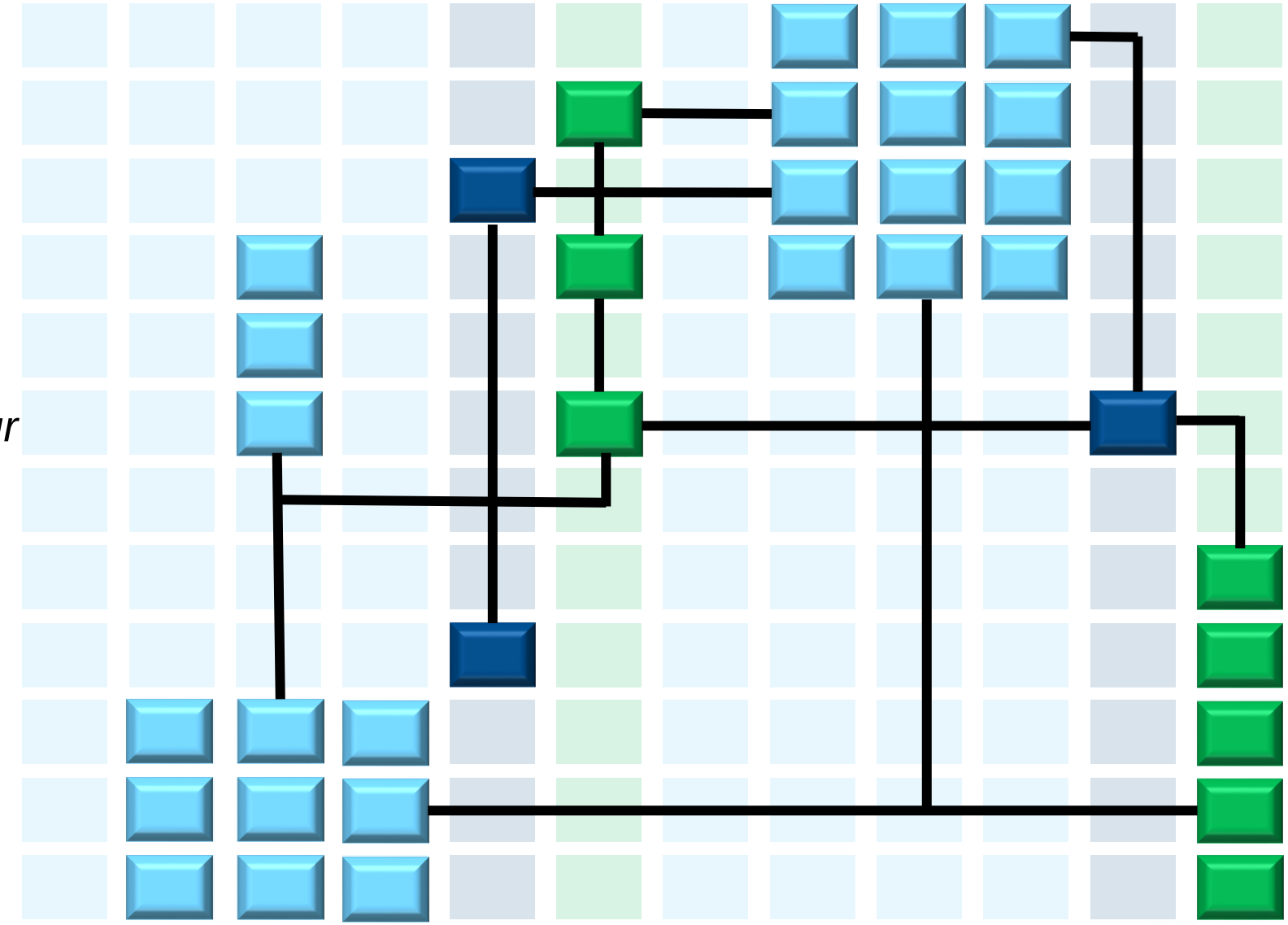
Basic Floating Point

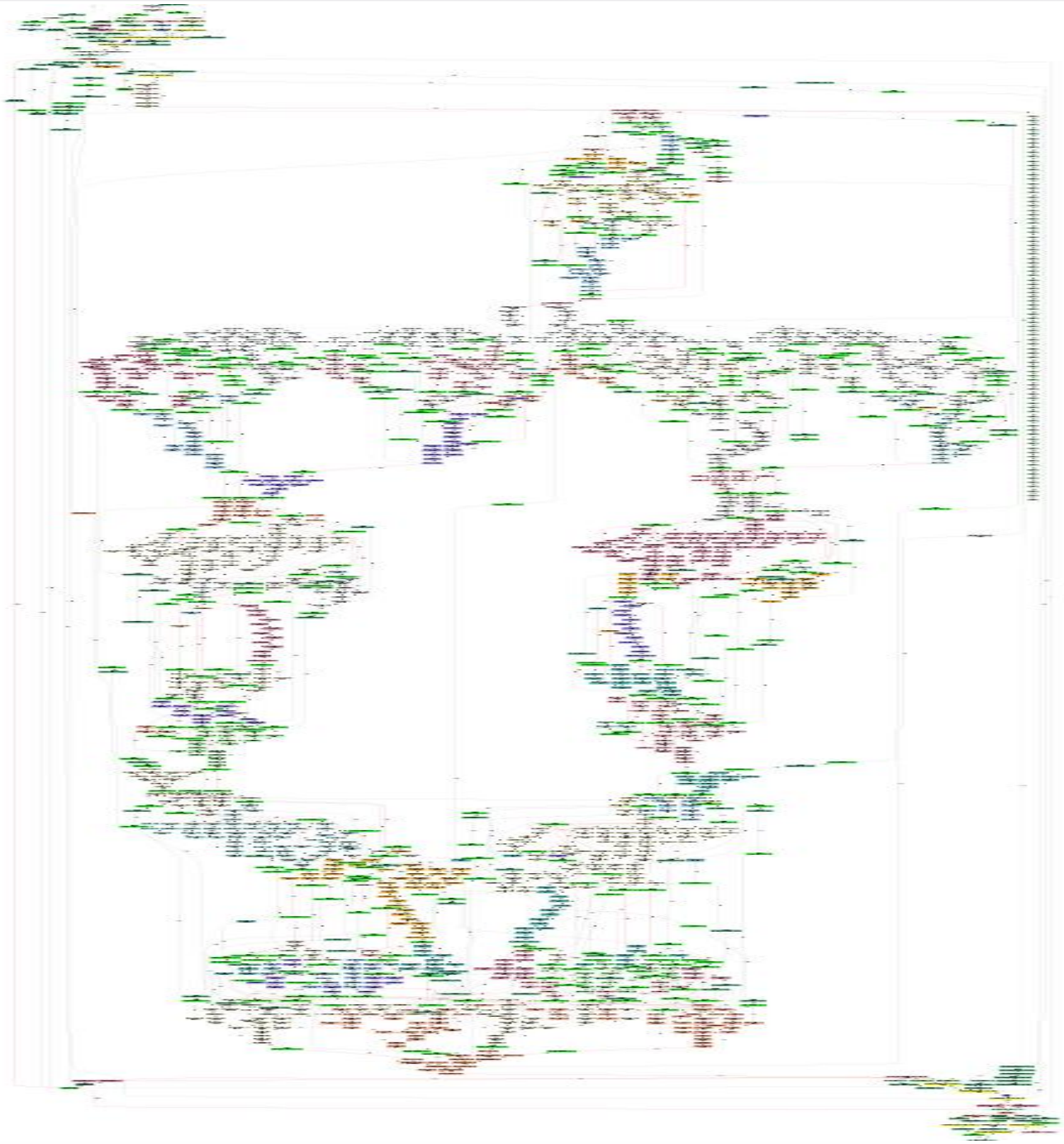
FPAdd
FPAddExpert
FPAddN
FPSubExpert
FPAddSub \
FPAddSubExpert
FPFusedAddSub
FPMul
FPMulExpert
FPConstMul
FPAcc
FPSqrt
FPDivSqrt
FPRecipSqrt
FPCbrt
FPDiv
FPInverse
FPFloor
FPCeil
FPRound
FPRint
FPFrac
FPMod
FPDim
FPAbs
FPMin
FPMax
FPMinAbs
FPMaxAbs
FPMinMaxFused
FPMinMaxAbsFused
FPCompare
FPCompareFused

FPGA Architecture: Configurable Connectivity = Efficiency

*Blocks are connected into a **custom data-path** that matches your application.*

Streaming data-path more efficient than copying to/from global memory





1GHz
Core Performance

5.5M
Logic Elements

Up to 70%
Lower Power

Heterogeneous up to
1TB/s
3D SIP Integration

Up to 10
TFLOPS

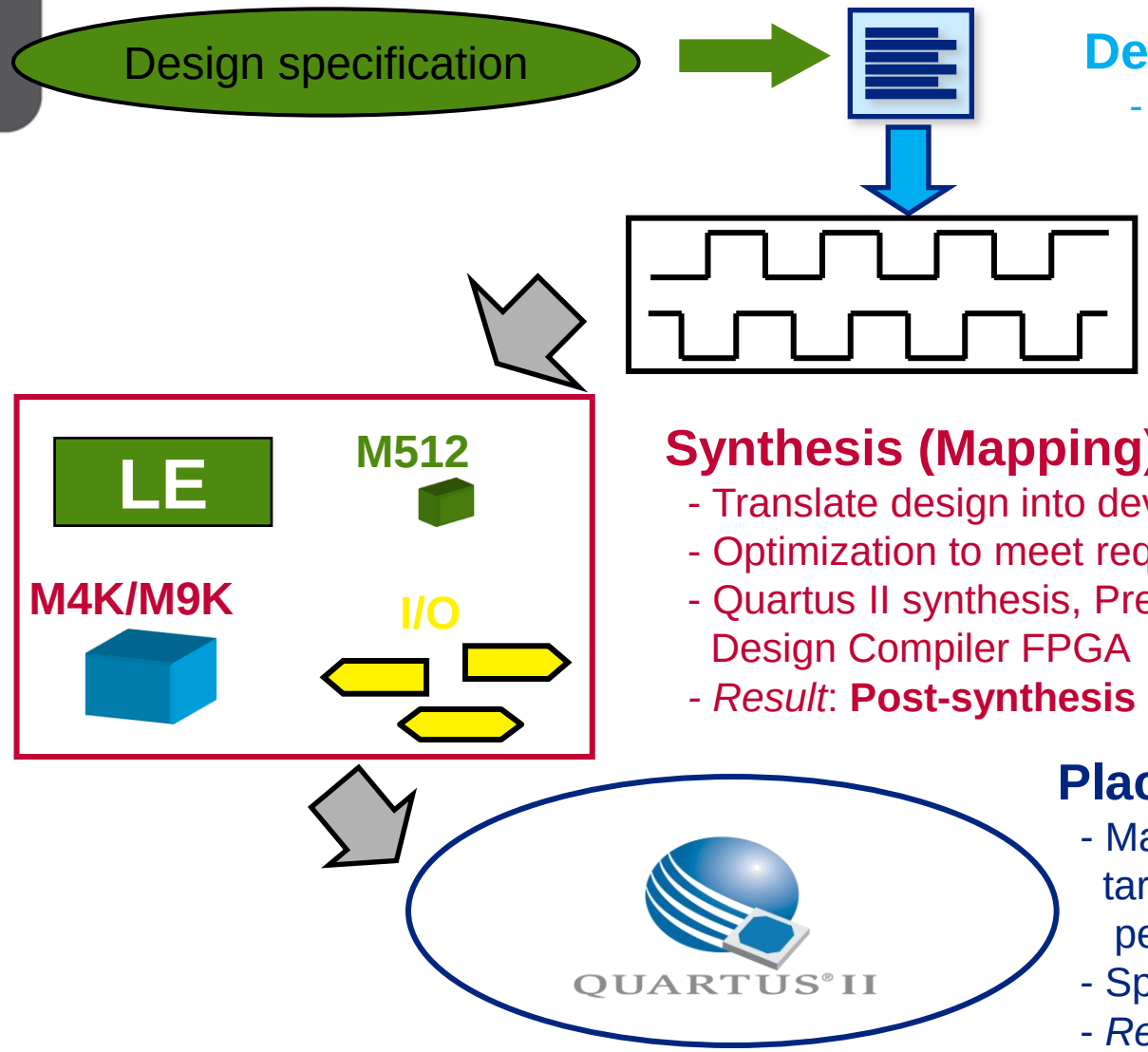
Intel 14 nm
Tri-Gate

Most
Comprehensive
Security

Quad-Core
Cortex-A53
ARM Processor

Developing with FPGA

Typical Programmable Logic Design Flow



Design entry/RTL coding

- Behavioral or structural description of design

RTL simulation

- Functional simulation
  (Mentor Graphics ModelSim® or other 3rd-party simulators)
- Verify logic model & data flow
  (no timing delays)

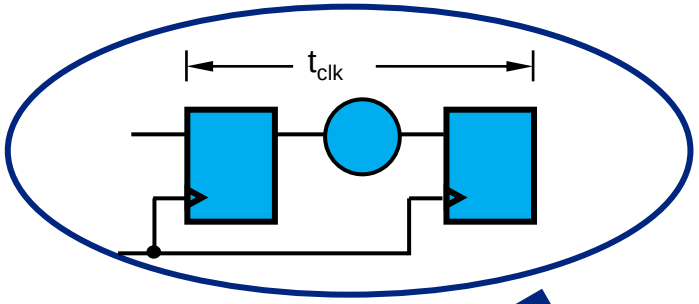
Synthesis (Mapping)

- Translate design into device specific primitives
- Optimization to meet required area & performance constraints
- Quartus II synthesis, Precision Synthesis, Synplify/Synplify Pro, Design Compiler FPGA
- *Result*: **Post-synthesis netlist**

Place & route (Fitting)

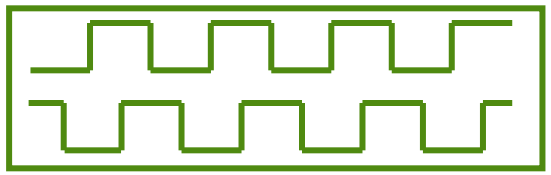
- Map primitives to specific locations inside target technology with reference to area & performance constraints
- Specify routing resources to be used
- *Result:* **Post-fit netlist**

Typical Programmable Logic Design Flow



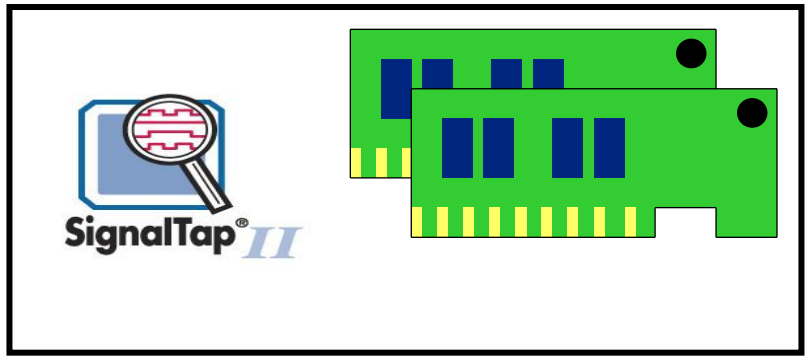
Timing analysis

- Verify performance specifications were met
- Static timing analysis



Gate level simulation (optional)

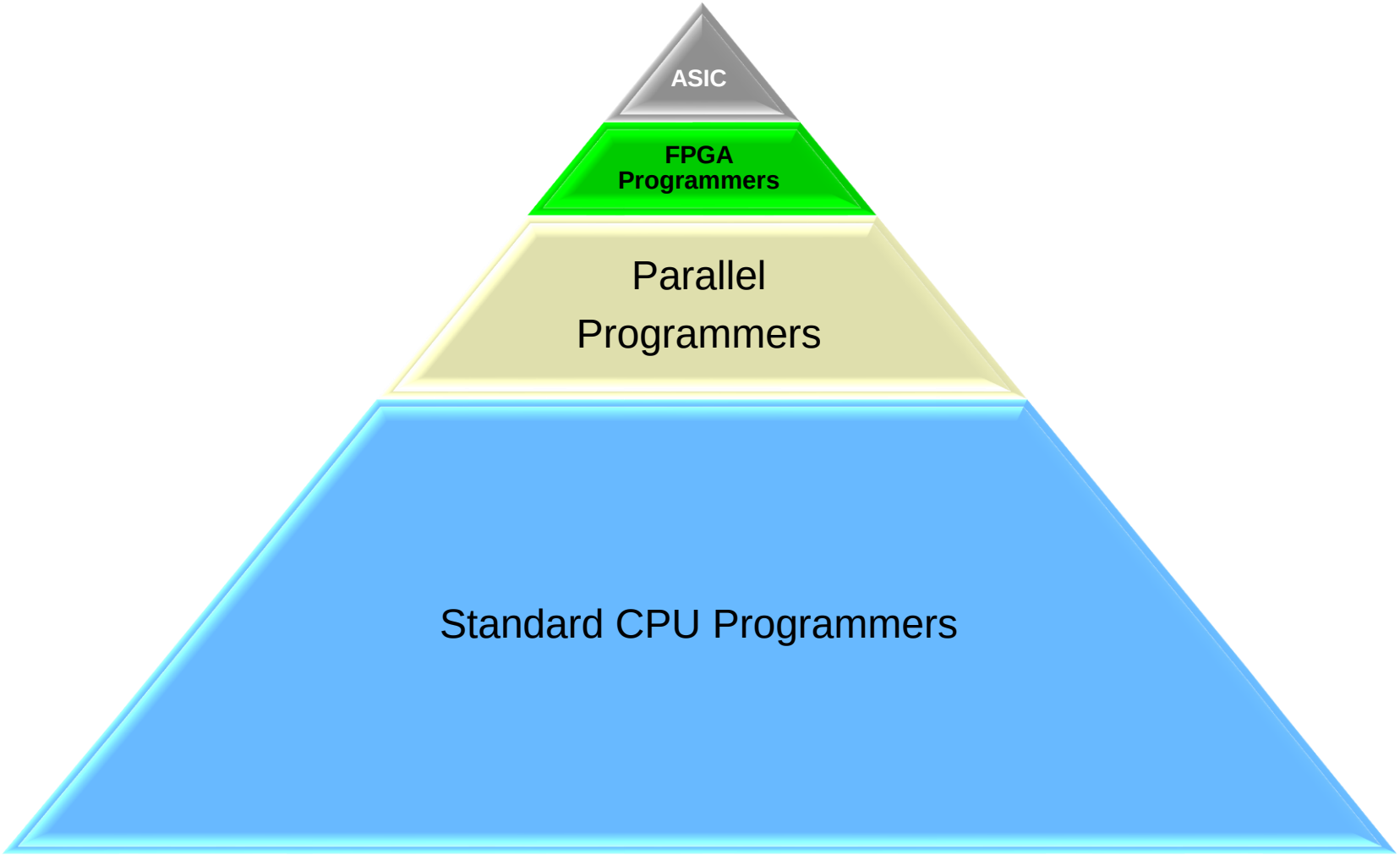
- Timing simulation
- Verify design will work in target technology



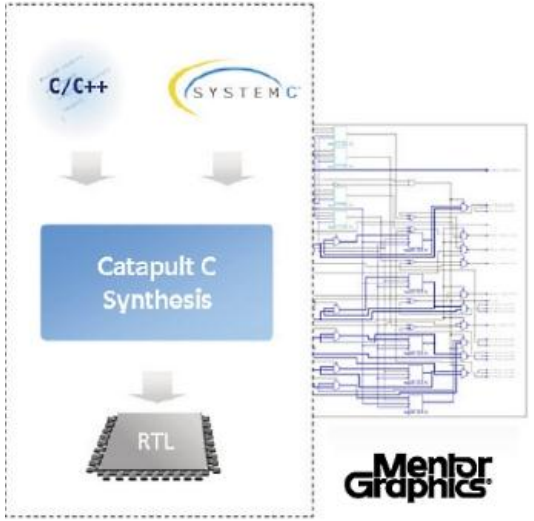
PC board simulation & test

- Simulate board design
- Program & test device on board
- Use on-chip tools for debugging

Application Development Paradigm



The magic trick ?



MathWorks®

HDL Coder

Celoxica

impulse
accelerated technologies

FpgaC

criticalblue
Accelerating Embedded Software

OpenACC to FPGA

May 24, 2016

ORNL Researchers Create Framework for Easier, Effective FPGA Programming

John Russell

Programmability and portability problems have long inhibited broader use of FPGA technology. FPGAs are already widely and effectively used in many dedicated applications (accelerated packet processing, for example), but generally not in situations that require 'reconfiguring' the FPGA to accommodate different applications. A group of researchers from Oak Ridge National Laboratory is hoping to change that.

Presenting at the 30th IEEE International Parallel & Distributed Processing Symposium (IPDPS) being held this week in Chicago, the group will discuss their work which is described in a new paper – *OpenACC to FPGA: A Framework for Directive-based High-Performance Reconfigurable Computing*. The authors include Seyong Lee, Jungwon Kin and Jeffrey S. Vetter, all of ORNL.

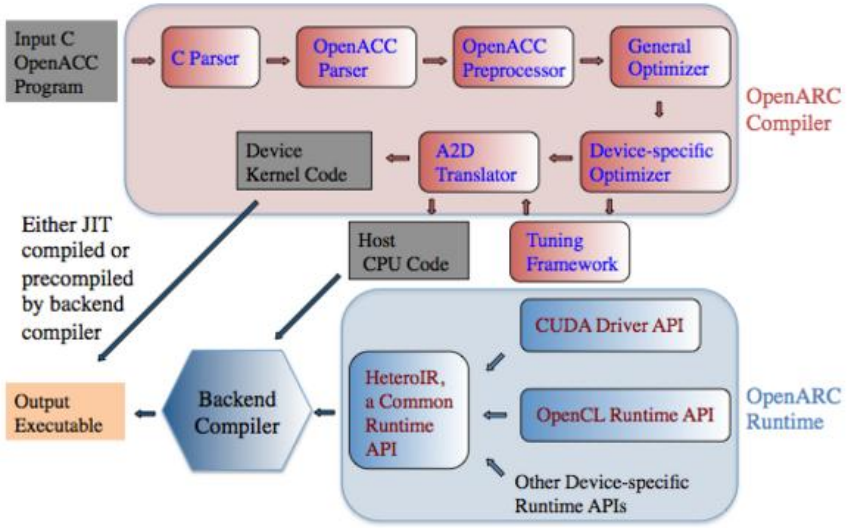


Figure 1: OpenARC System Architecture

OpenCL: abstracting FPGA away

Altera OpenCL Program Overview

- 2010 research project
  - Toronto Technology Center
- 2011 Development started
  - Proof of concept
  - 9 customer evaluations
- 2012 Early Access Program
  - Demo's at Supercomputing '12
  - Over 60 customer evaluations
- 2013 First public release
  - Publically available May 2013
  - Passed Conformance Testing
    - >8500 programs run properly
- Public release 13.1 (Nov 2013)
  - Channels (Streaming IO)
  - Example Designs
  - SoC Support
- Release 14.0 (June 2014)
  - Platforms
  - Emulator/Profiler
  - Rapid Prototyping
- Release 14.1 (Nov 2014)
  - Arria 10 support
  - Shared Virtual Memory (PoC)
- Release 15.1 (Nov 2015)
  - Kernel Update
  - Library support

OpenCL Use Model: Abstracting the FPGA away

Host Code

```
main() {
   read_data( … );
   manipulate( … );
   clEnqueueWriteBuffer( … );
   clEnqueueNDRange(…,sum,…);
   clEnqueueReadBuffer( … );
   display_result( … );
}
```

OpenCL Accelerator Code

```
__kernel void sum
   (__global float *a,
    __global float *b,
    __global float *y)
{
  int gid = get_global_id(0);
  y[gid] = a[gid] + b[gid];
}
```

Standard gcc Compiler

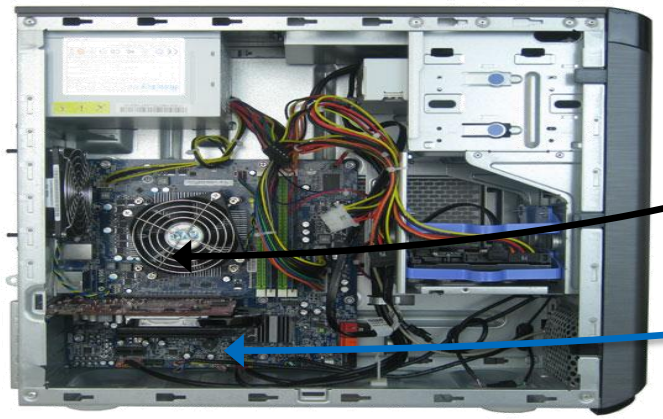
Altera Offline Compiler

Verilog

Quartus II

EXE

AOCX

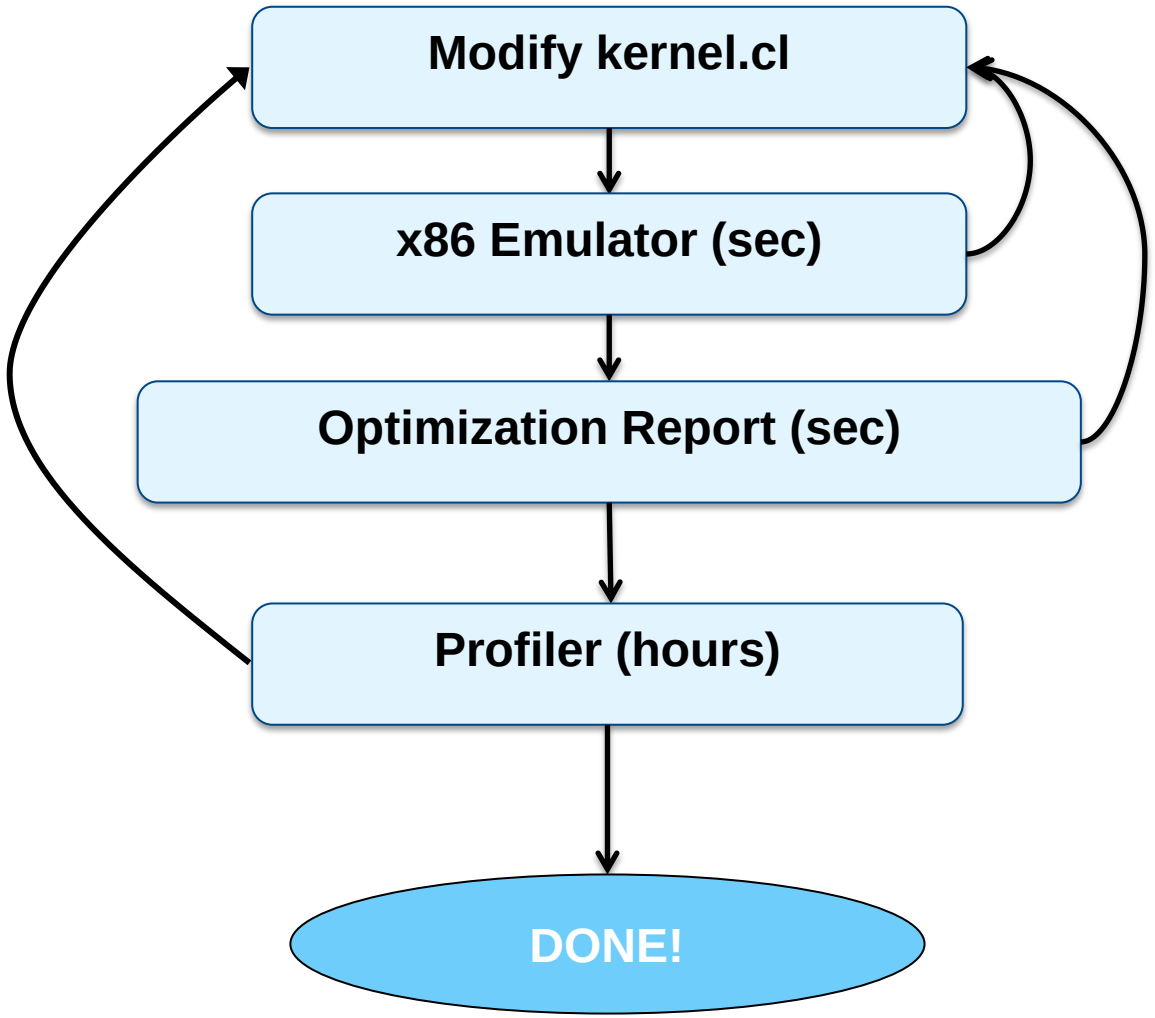


Host

Accelerator

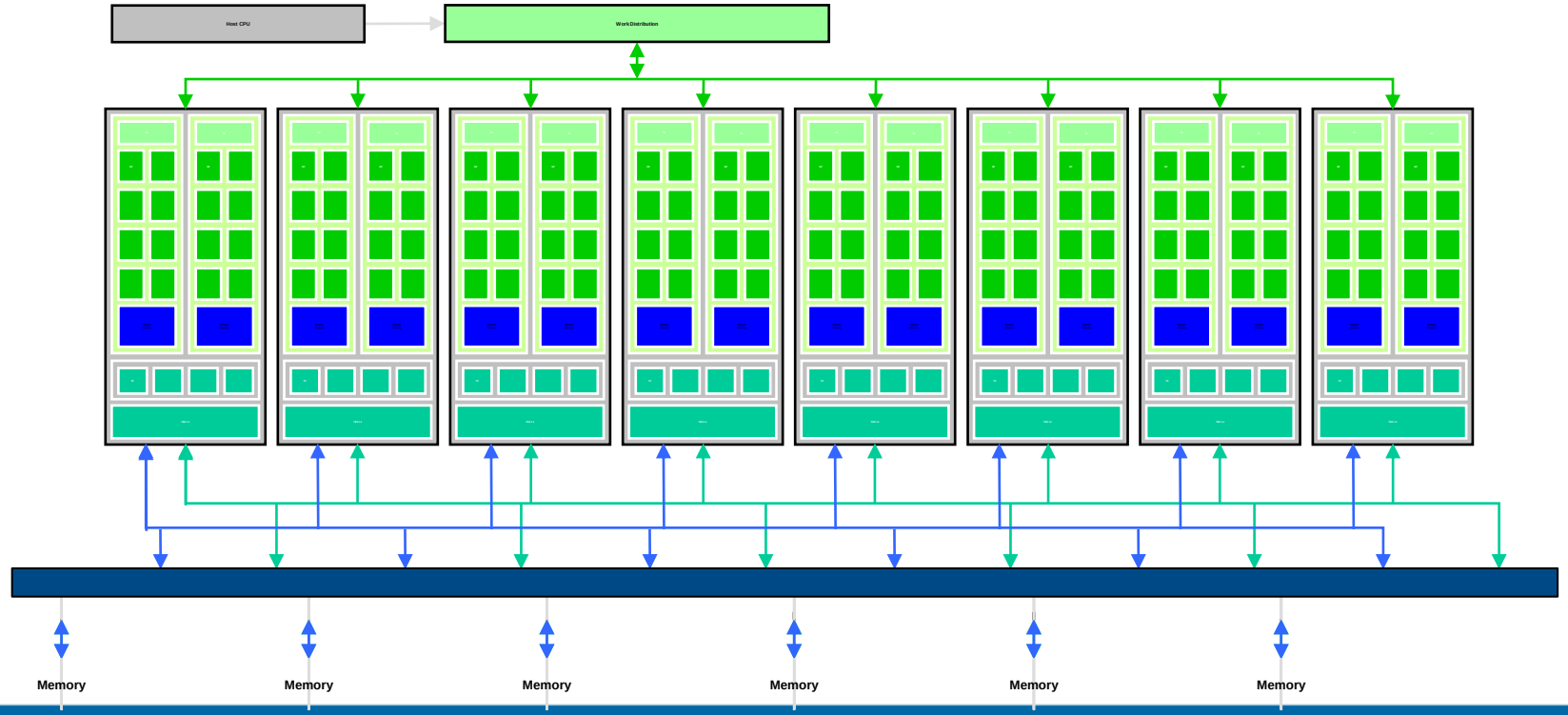
SoC FPGA combines these in single device

Software Development Flow



OpenCL on GPU/Multi-Core CPU Architectures

- Conceptually many parallel threads
- Simplified View
  - Each thread runs sequentially on a different processing element (PE)
  - Fixed #s of Functional Units, Registers available on each PE
  - Many processing elements are available to provide significant parallel speedup



OpenCL on FPGA

- OpenCL kernels are translated into a highly parallel circuit
  - A unique functional unit is created for every operation in the kernel
    - Memory loads / stores, computational operations, registers
  - Functional units are only connected when there is some data dependence dictated by the kernel
- Pipeline the resulting circuit with a new thread on each clock cycle to keep functional units busy

Amount of parallelism is dictated by the number of pipelined computing operations in the generated hardware

Mapping a simple program to an FPGA

High-level code

```
Mem[100] += 42 * Mem[101]
```

CPU instructions

```
R0 ← Load Mem[100]
R1 ← Load Mem[101]
R2 ← Load #42
R2 ← Mul R1, R2
R0 ← Add R2, R0
Store R0 → Mem[100]
```

CPU activity, step by step

R0 ← Load Mem[100]

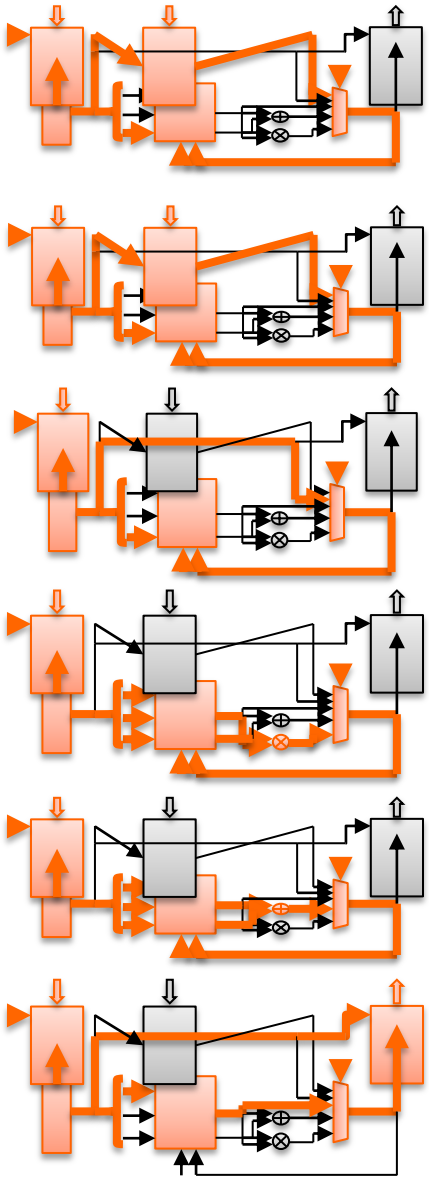
R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]



Time

On the FPGA we unroll the CPU hardware…

R0 ← Load Mem[100]

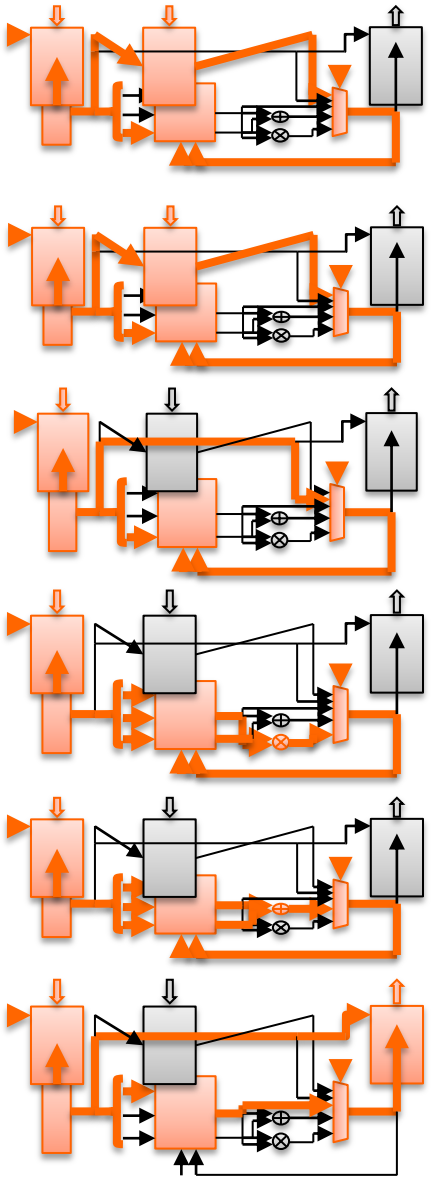
R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]



Space

… and specialize by position

R0 ← Load Mem[100]

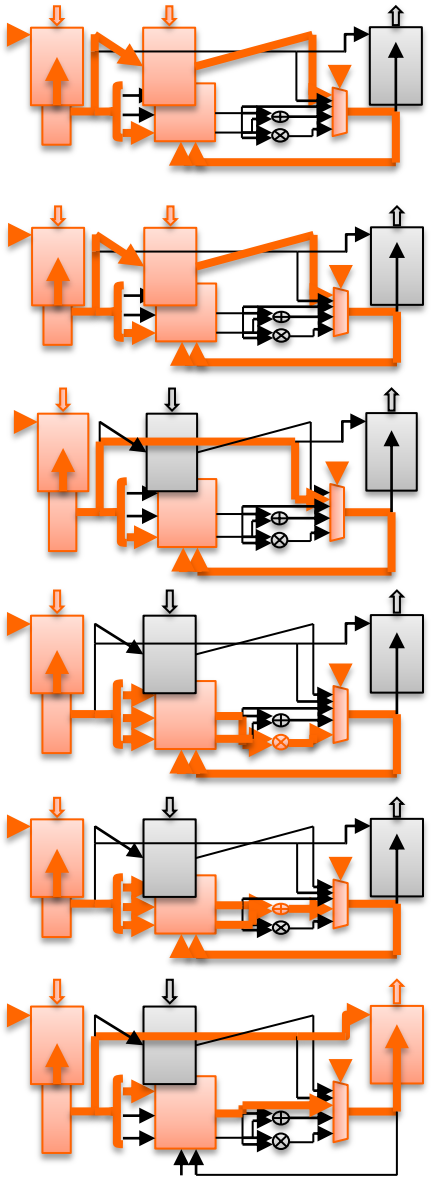
R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]



1. Instructions are fixed. Remove “Fetch”

… and specialize

R0 ← Load Mem[100]

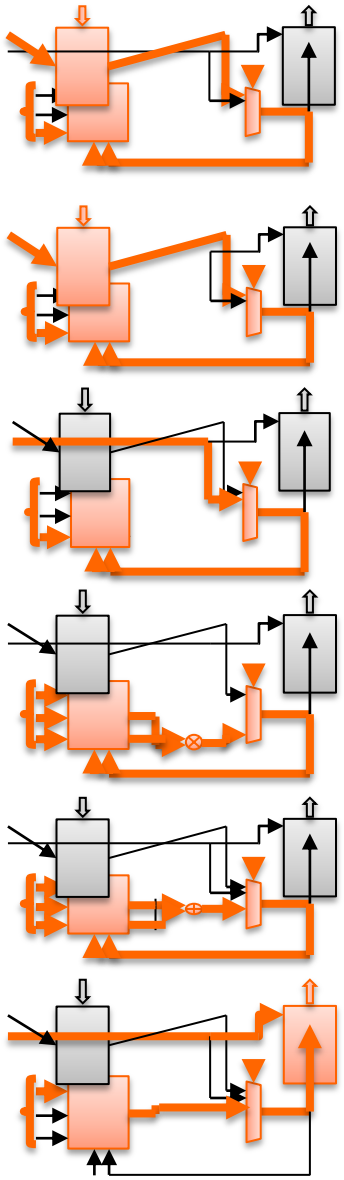
R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]



1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops

… and specialize

R0 ← Load Mem[100]

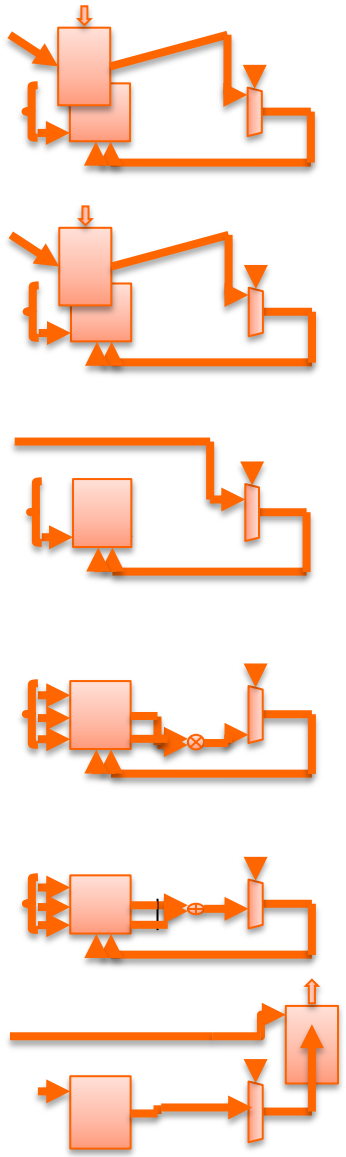
R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]



1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store

… and specialize

R0 ← Load Mem[100]

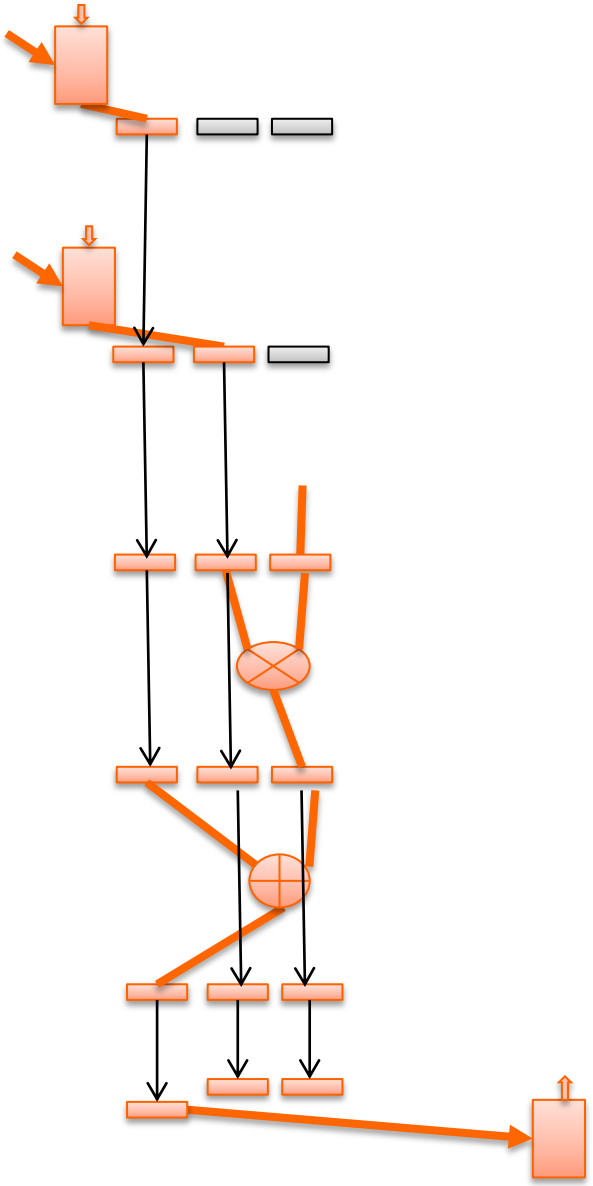
R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]



1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly! And propagate state.

… and specialize

R0 ← Load Mem[100]

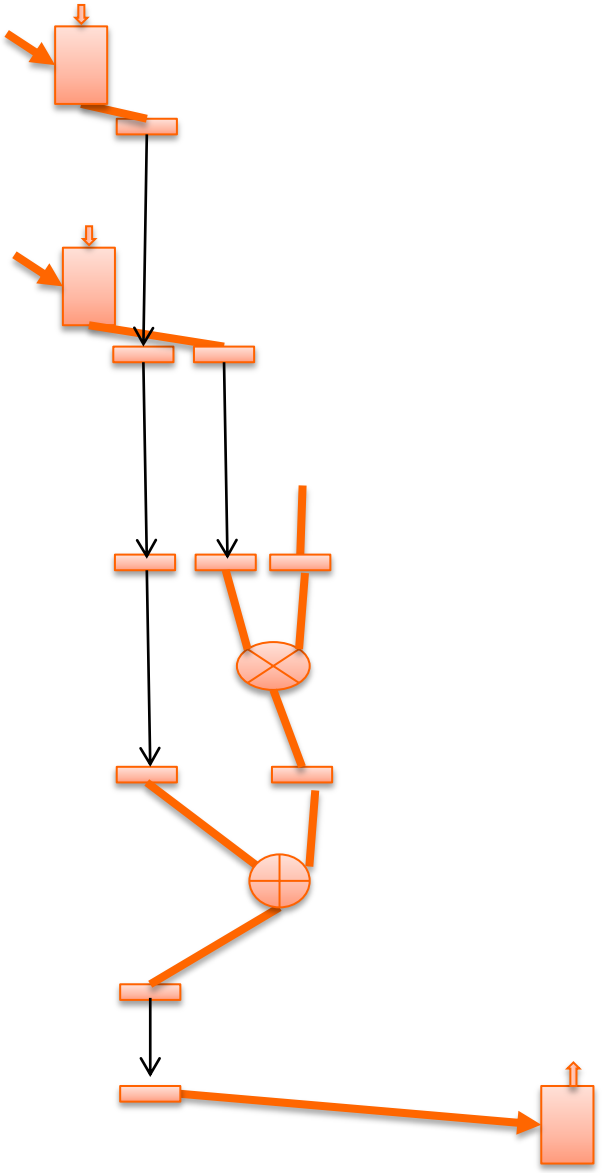
R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]



1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly! And propagate state.
5. Remove dead data.

… and specialize

R0 ← Load Mem[100]

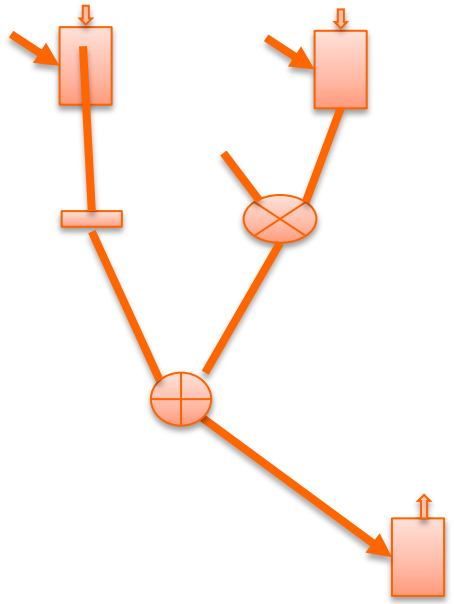
R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]



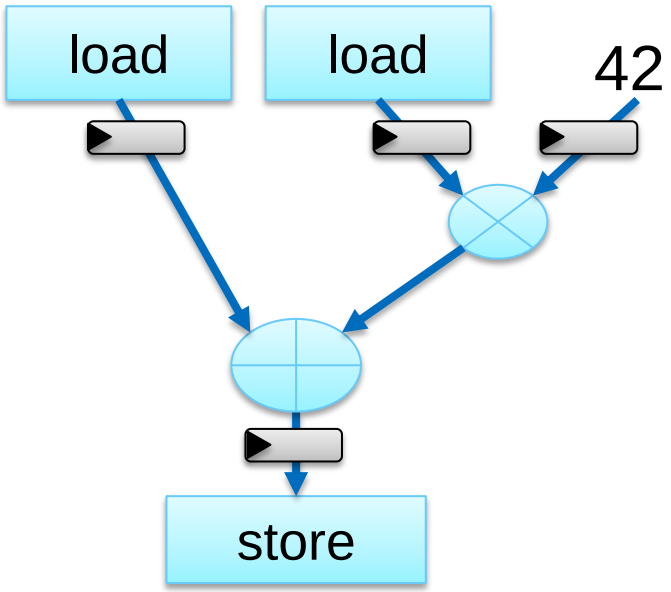
1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly! And propagate state.
5. Remove dead data.
6. Reschedule!

Custom data-path on the FPGA matches your algorithm!

High-level code

```
Mem[100] += 42 * Mem[101]
```

Custom data-path



Build exactly what you need:

Operations

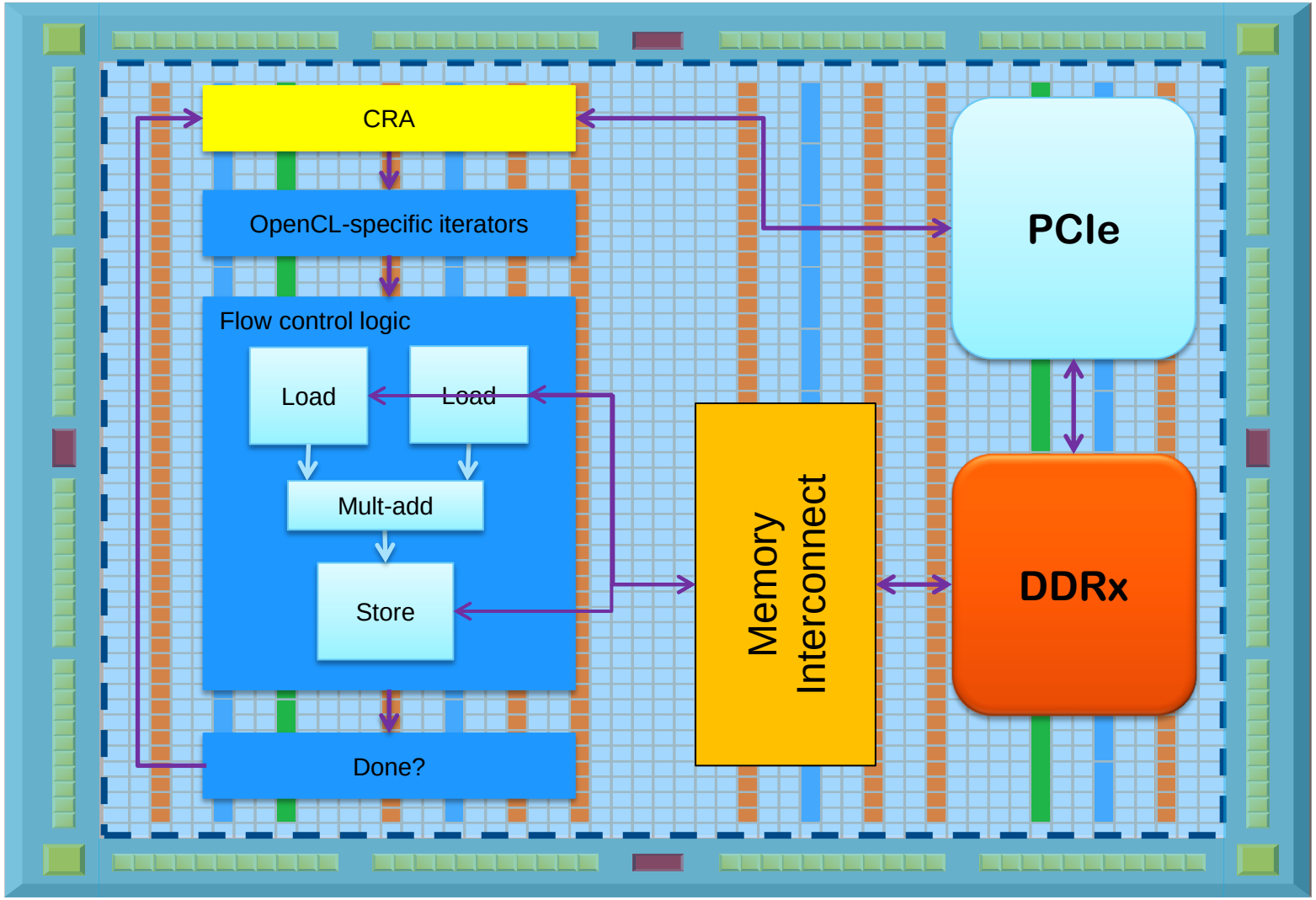
Data widths

Memory size & configuration

Efficiency:

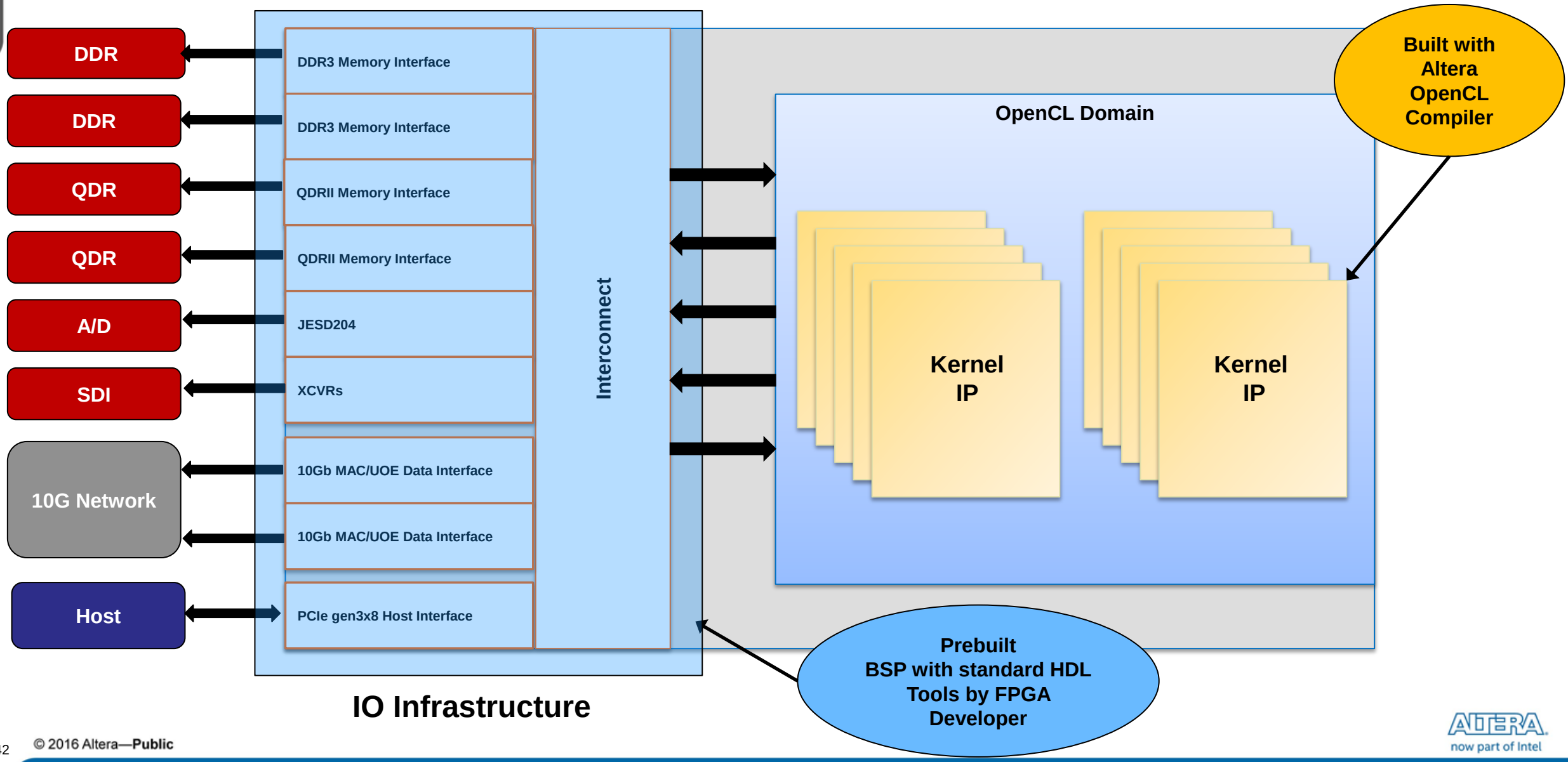
Throughput / Latency / Power

What Hardware do we produce?



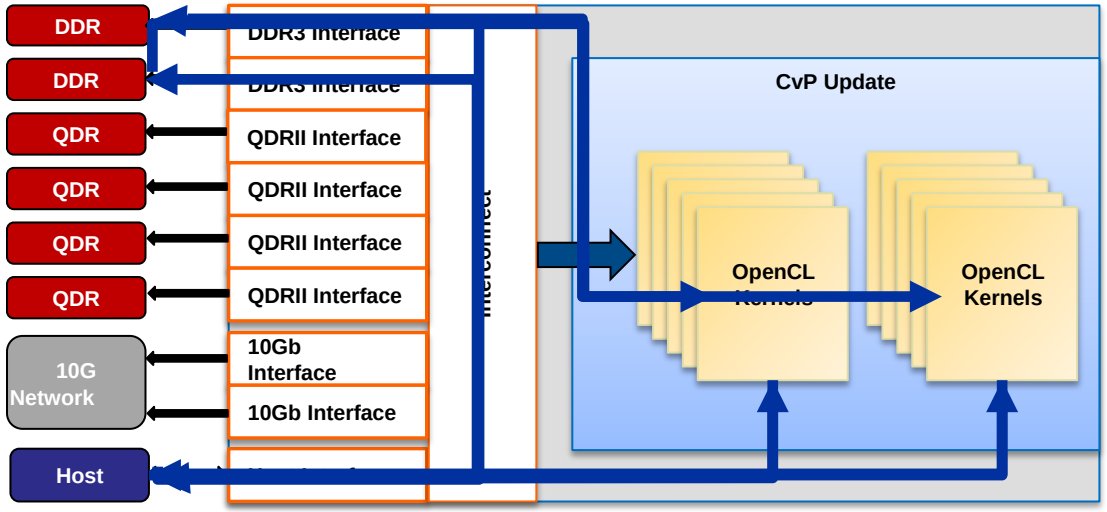
ALTERA BSP: abstracting FPGA development

An adaptable Board Support Package



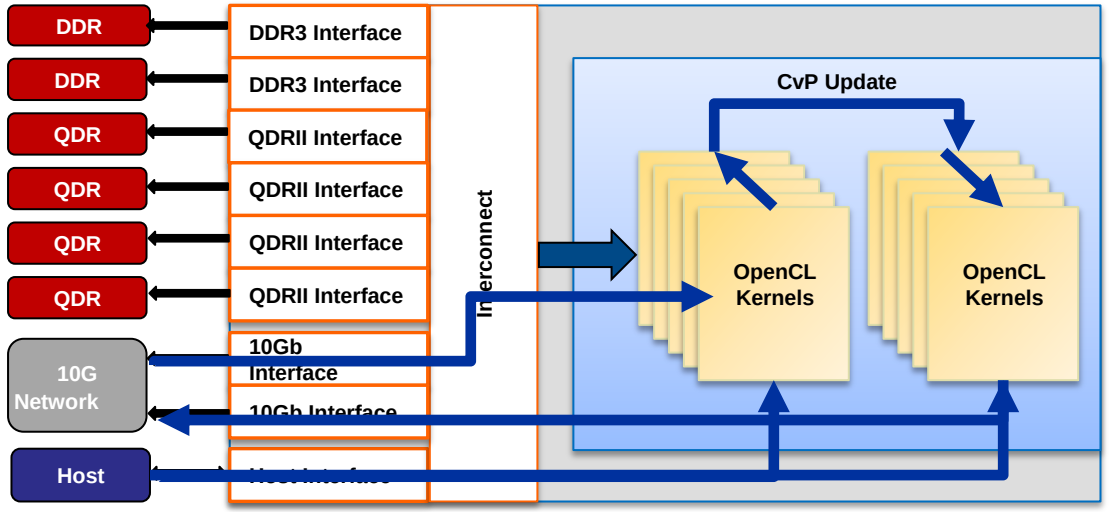
Channels Advantage

Standard OpenCL



Altera Vendor Extension

IO and Kernel Channels




```
channel int DataChannel;

kernel producer(...) {
  write_channel_altera(DataChannel, value);
}

kernel consumer(...) {
  value = read_channel_altera(DataChannel);
}
```

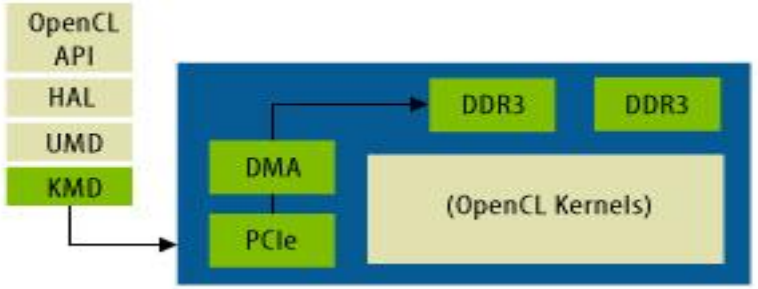
Start with OpenCL ready platforms 1/2

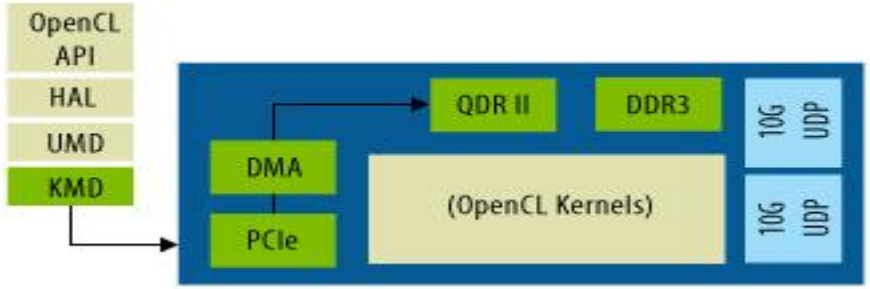
HPC Applications

Stratix® FPGA • SoC

Arria® FPGA • SoC

Network Applications





BittWare

REFLEX CES Custom Embedded Systems

Nallatech

terasIC www.terasic.com

picocomputing

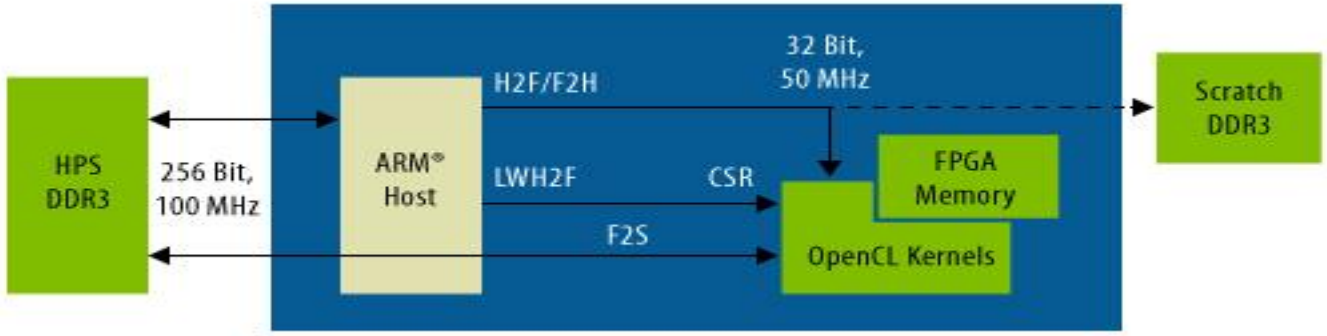
GIDEL

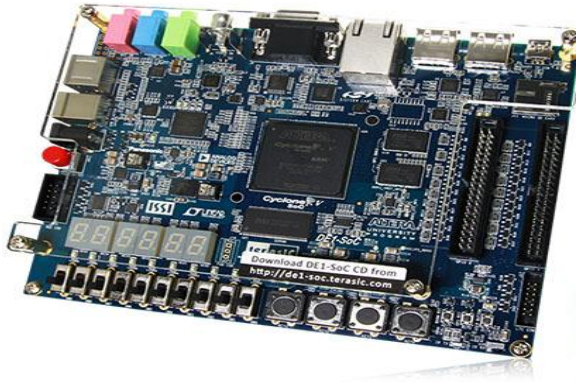
BittWare

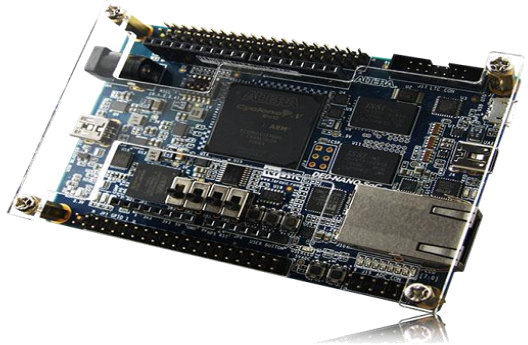
Nallatech

terasIC www.terasic.com

Start with OpenCL ready platforms 2/2

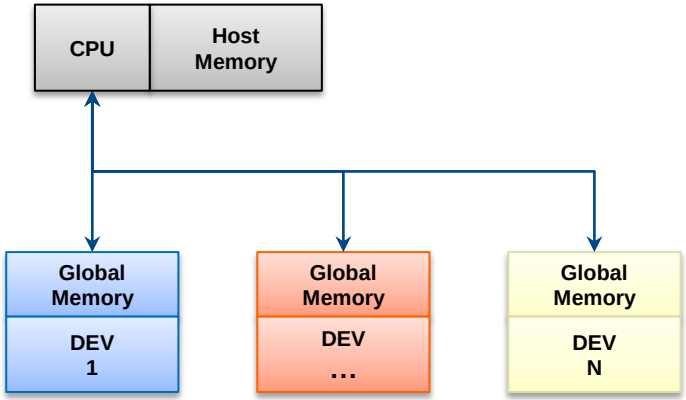


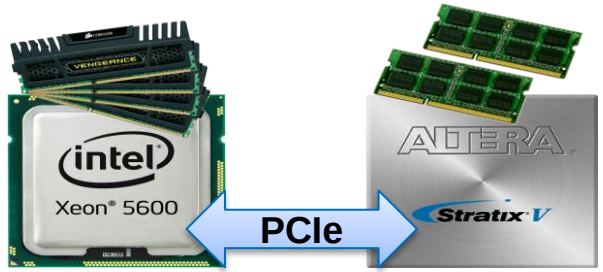




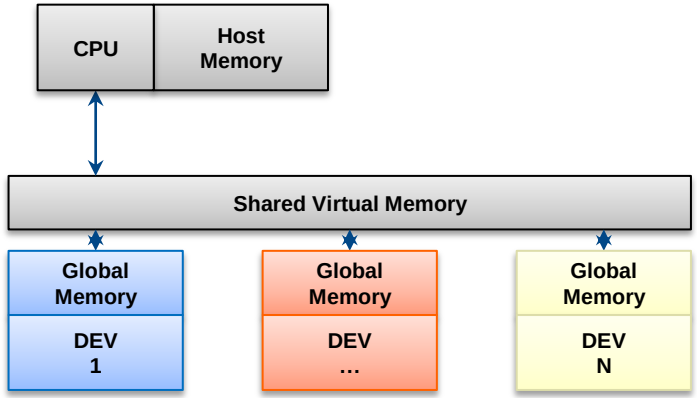
Shared Virtual Memory (SVM) Platform Model

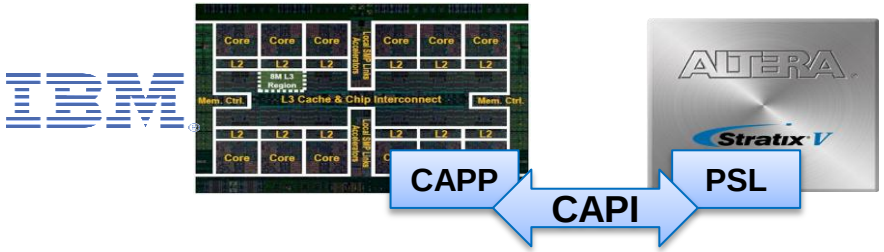
- OpenCL 1.2
  - Traditional Hosted Heterogeneous Platform

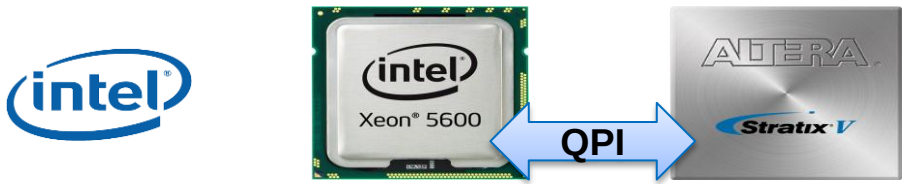




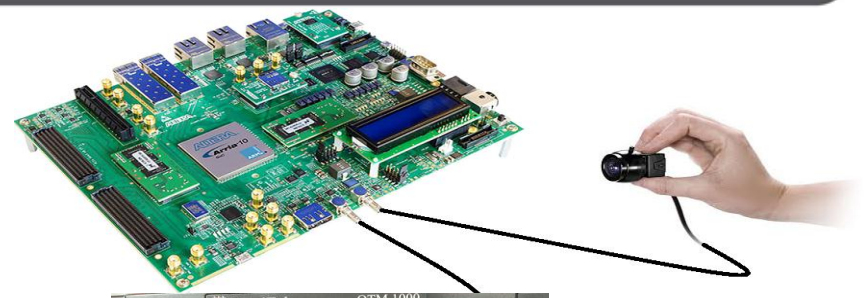
- OpenCL 2.0
  - New Hosted Heterogeneous Platform with SVM







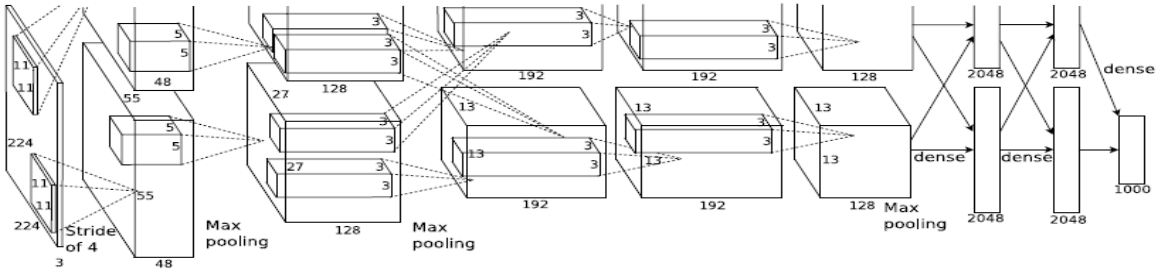
VIP based BSP Customization



Example

Case Study: Image Classification

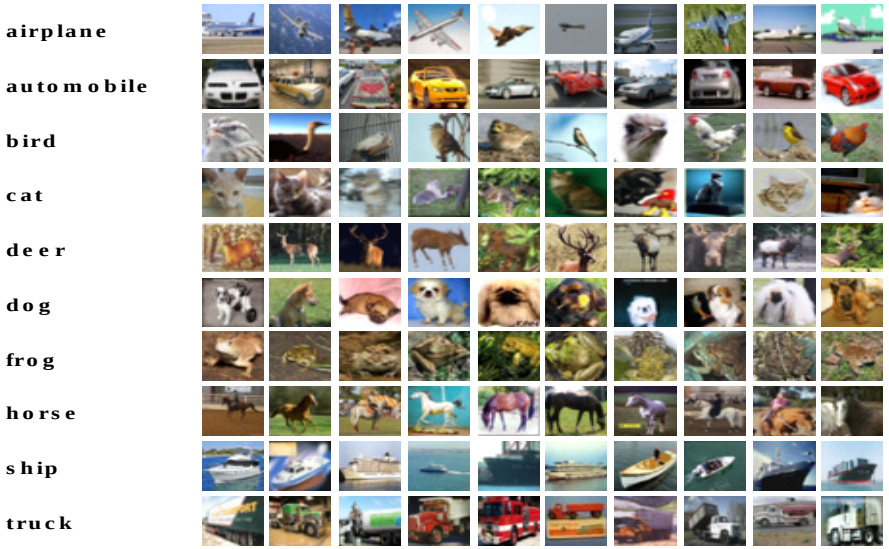
- Deep Learning Algorithm
  - Convolutional Neural Networking
  - Based on Hinton's CNN
- Early Results on Stratix V
  - 2X Perf./Power vs. gpgpu
    - despite soft floating point
    - 400 images/s
  - 8+ simultaneous kernels
    - vs. 2 on gpgpu
  - Exploiting OpenCL channels
    - between kernels
- A10 results
  - Hard floating point uses all DSPs
    - Better density and frequency
    - ~ 4X performance/watt v SV
  - 6800 images/s
  - No code change required



Hinton's CNN Algorithm

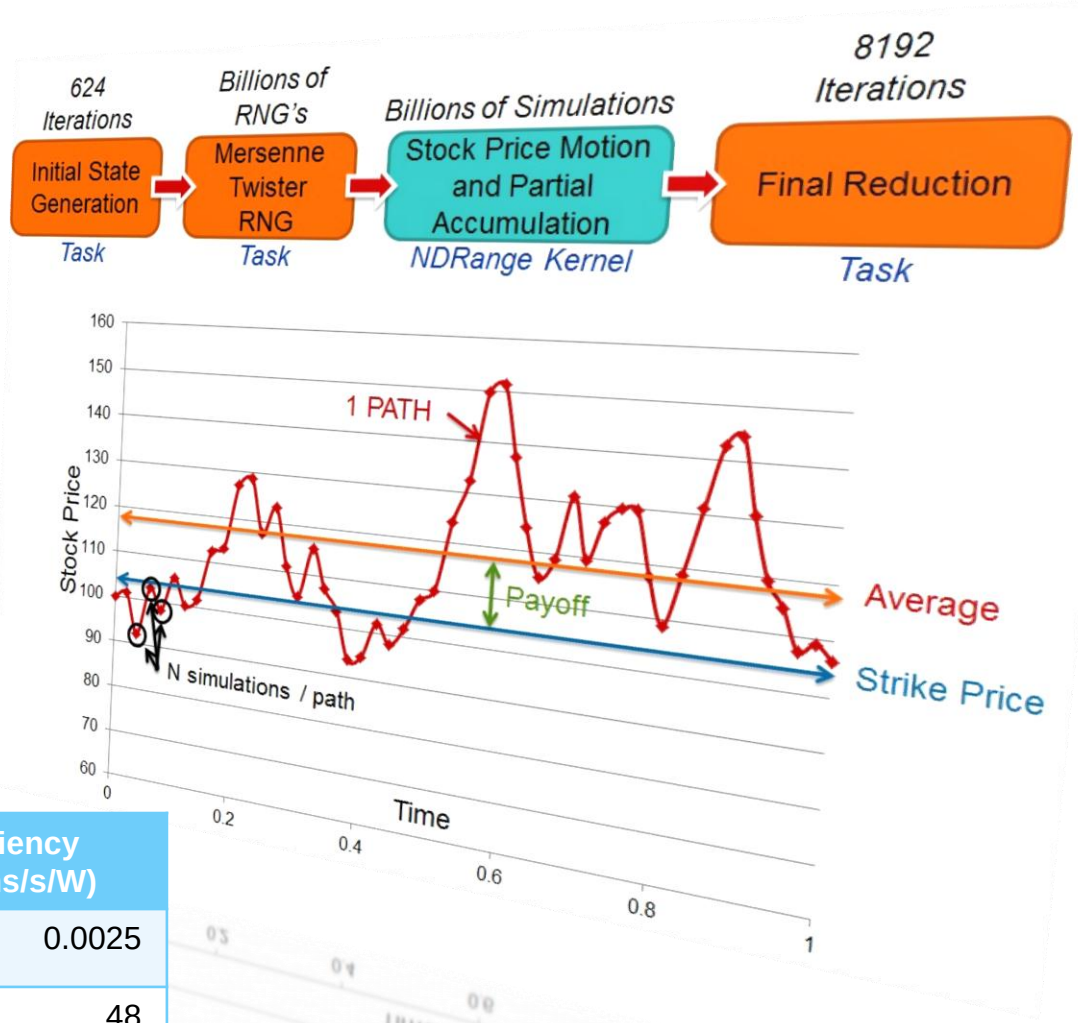
The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Here are the classes in the dataset, as well as 10 random images from each:



Multi-Asset Barrier Option Pricing

- Monte-Carlo simulation
  - No closed form solution possible
  - High quality random number generator required
  - Billions of simulations required
- Used GPU vendors example code
- Advantage FPGA
  - Complex Control Flow
- Optimizations
  - Channels, loop pipelining



Platform	Power (W)	Performance (Bsims/s)	Efficiency (Msims/s/W)
W3690 Xeon Processor	130	.032	0.0025
nVidia Kepler20	212	10.1	48
Bittware S5-PCIe-HQ	45	12.0	266

Summary

OpenCL + FPGA Key Benefits

- Faster development vs. traditional FPGA design flow
  - Puts the FPGA in the software developers hands
  - Familiar C-based development flow
- Heterogeneous IO interface
  - Multiple 10G Ethernet
  - SDI, HMDI, A/D Interface
- Higher performance/watt vs. CPU/GPGPU
  - Implement exactly what you need
  - Pipeline parallel structures
  - Custom interconnect converging with data processing cores
- Portability & Obsolescence free
  - Code can transfer between different HW accelerators (CPU, GPGPU, FPGA, etc)
  - Code ports seamlessly to new generations of the FPGA
  - FPGA life cycle considerably longer than CPUs or GPGPUs

Q & A