

# 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
- ◀ Live Demo
- ◀ Developing a Custom OpenCL BSP

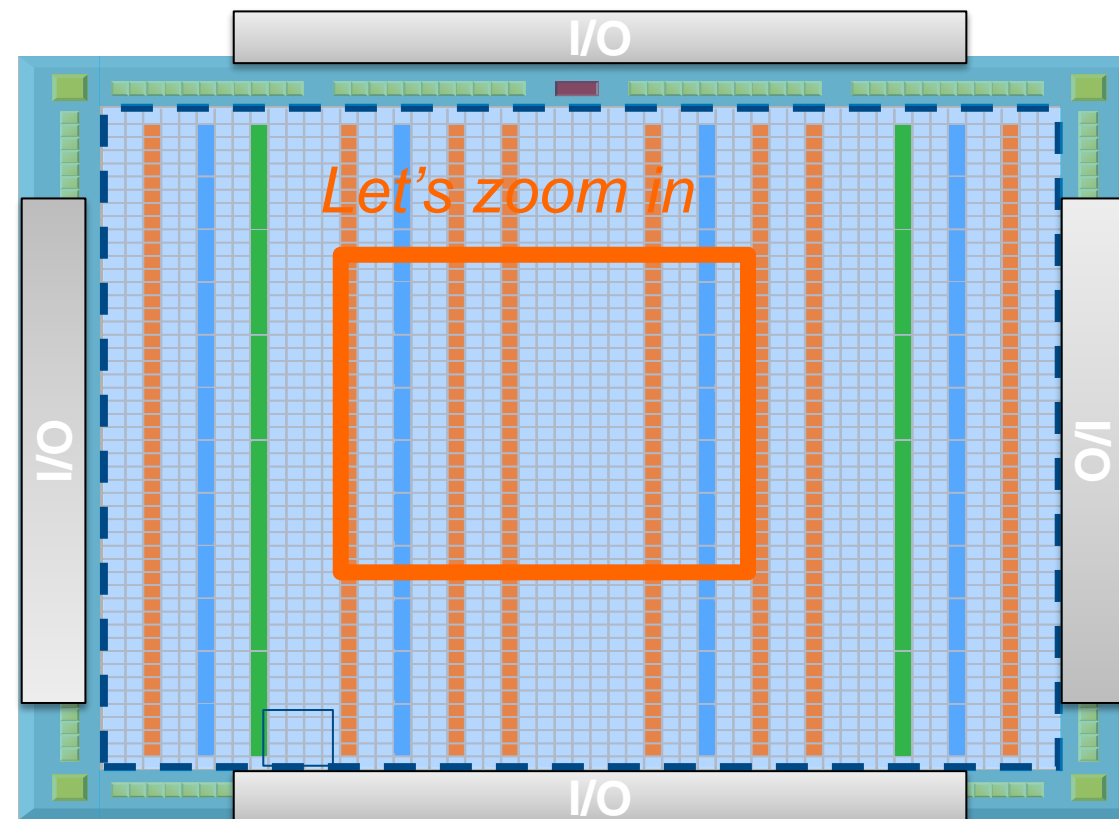
# FPGA architecture overview



**ALTERA**  
now part of Intel

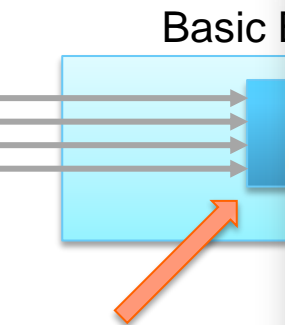
## FPGA Architecture: Fine-grained Massively Parallel

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



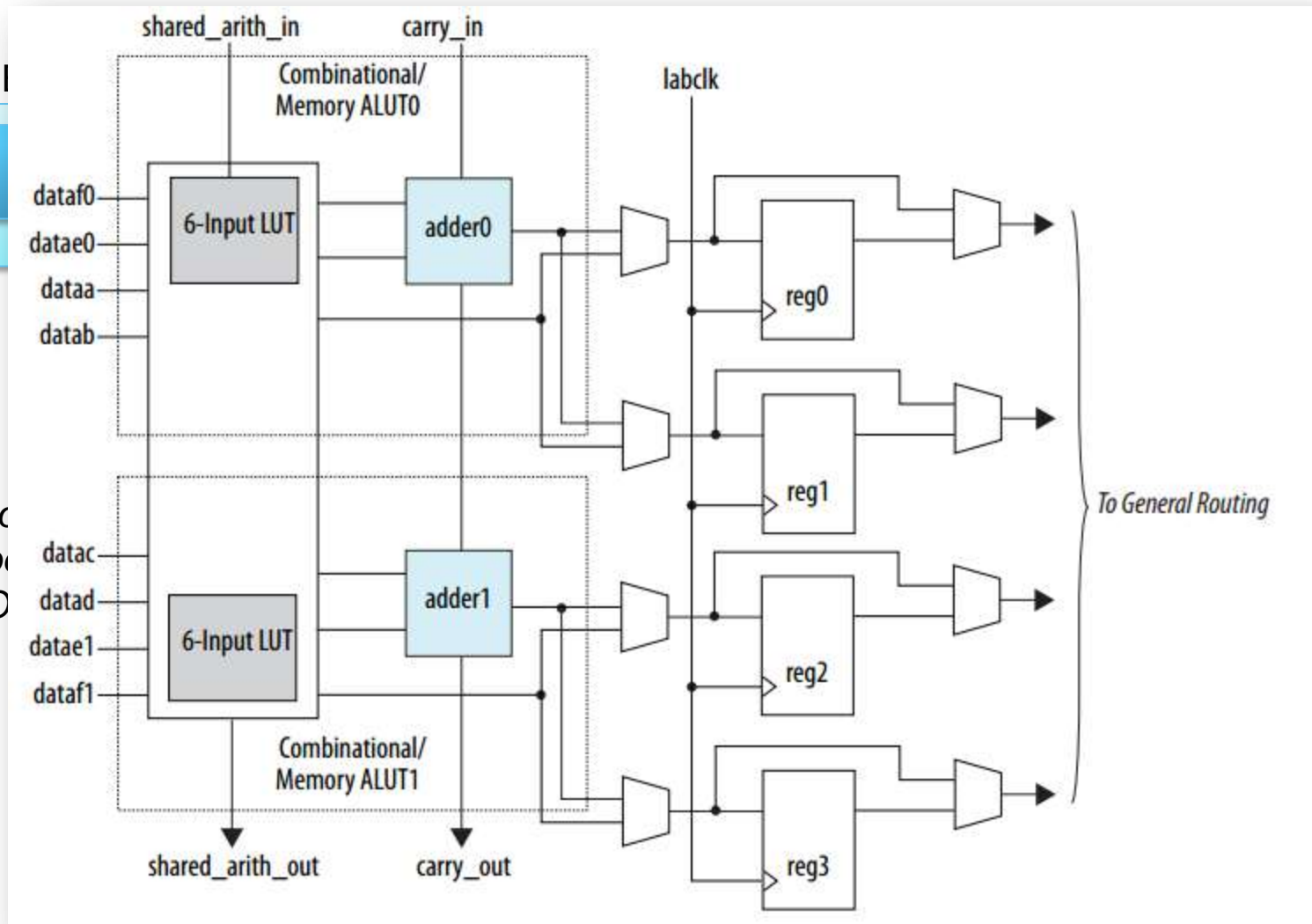
# FPGA Architecture: Basic Elements

Basic I/O

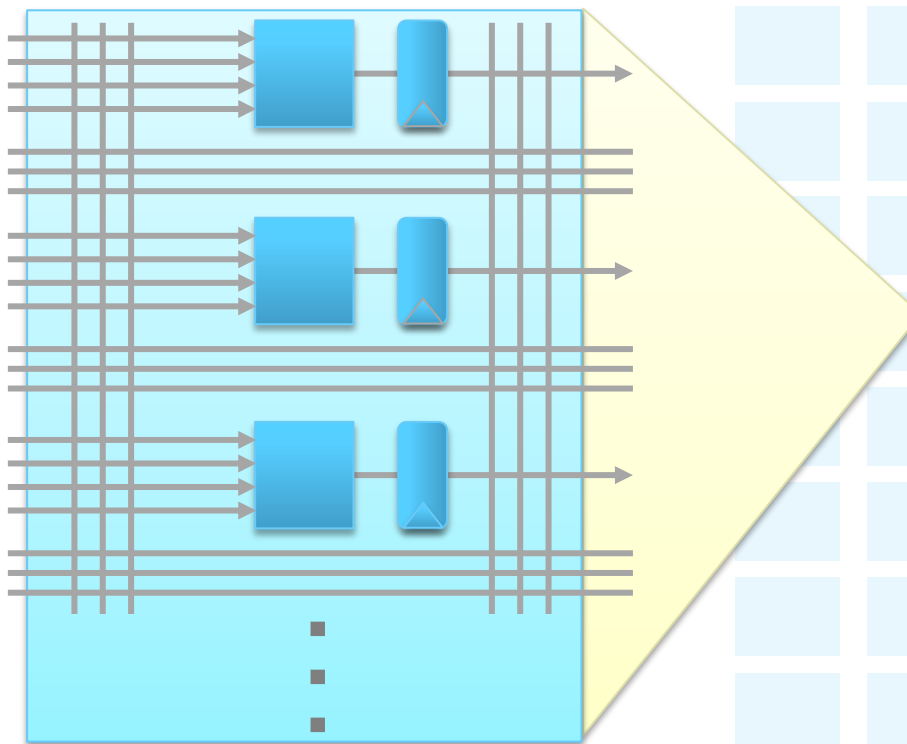


1-bit configurable operation

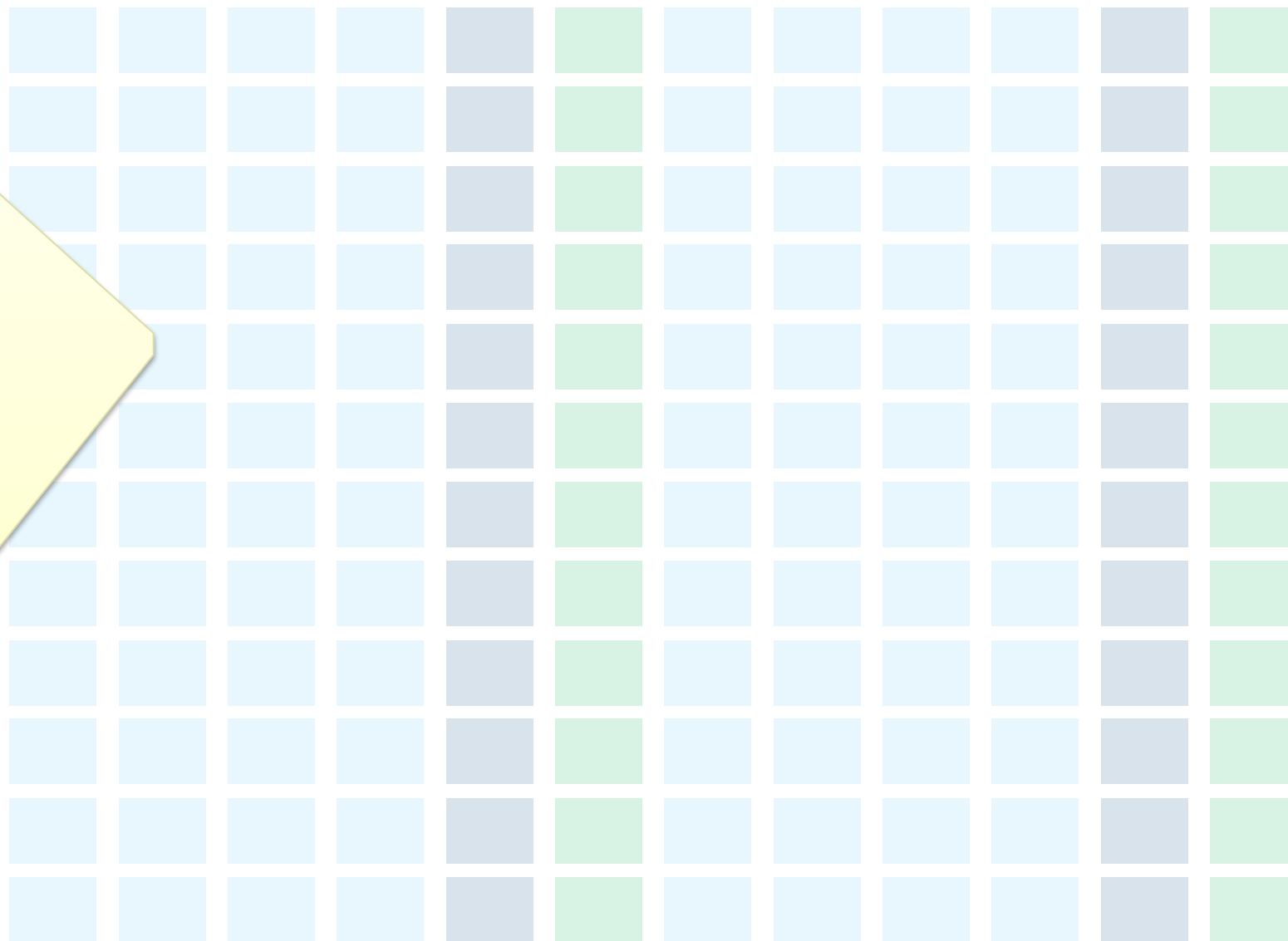
Configured to  
1-bit op  
AND, OR, NO



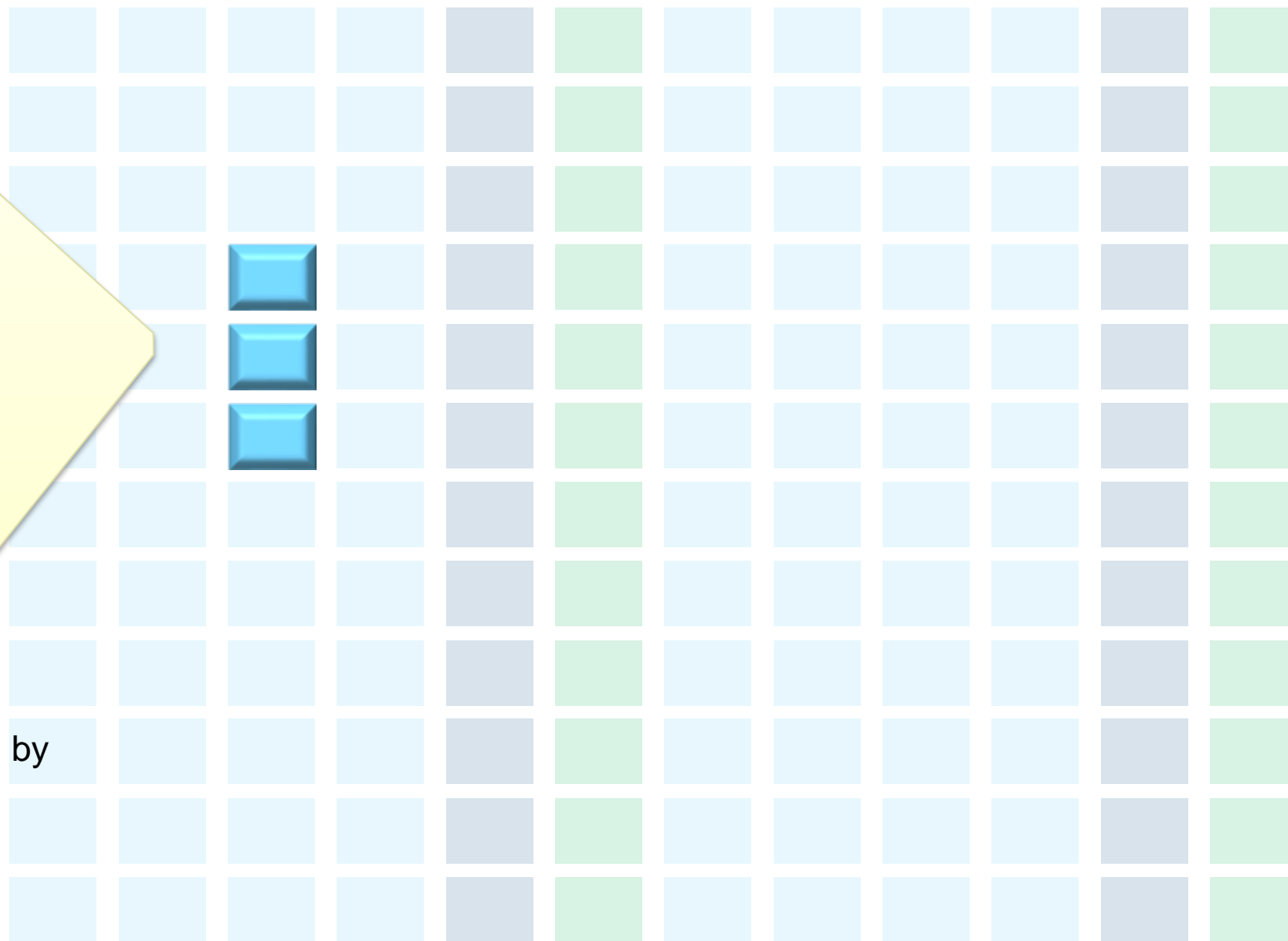
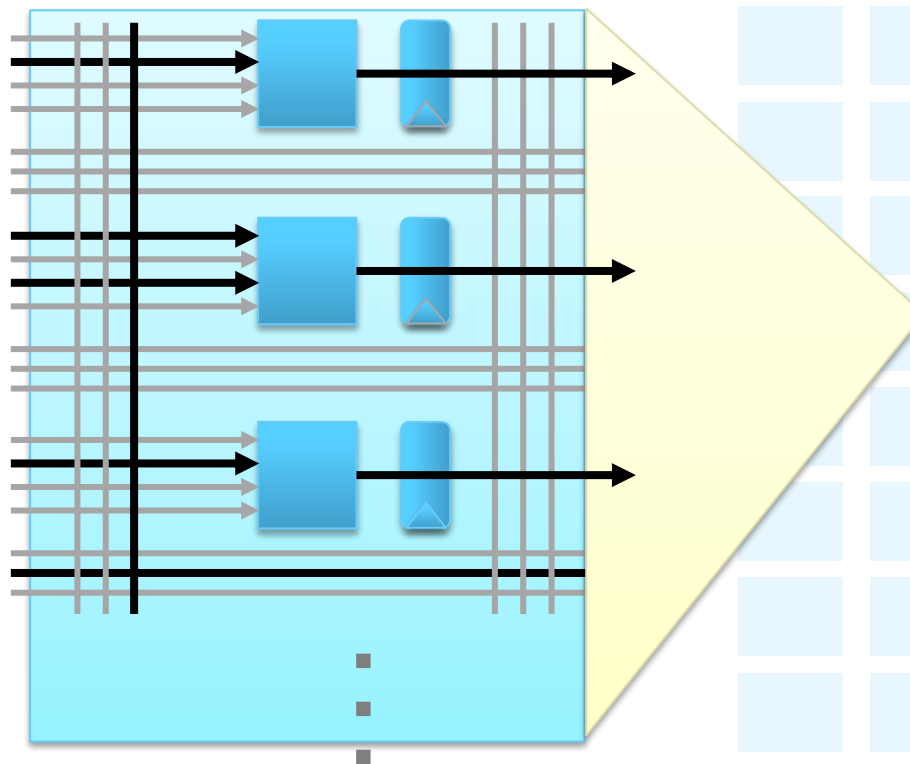
# FPGA Architecture: Flexible Interconnect



Basic Elements are surrounded with a flexible interconnect

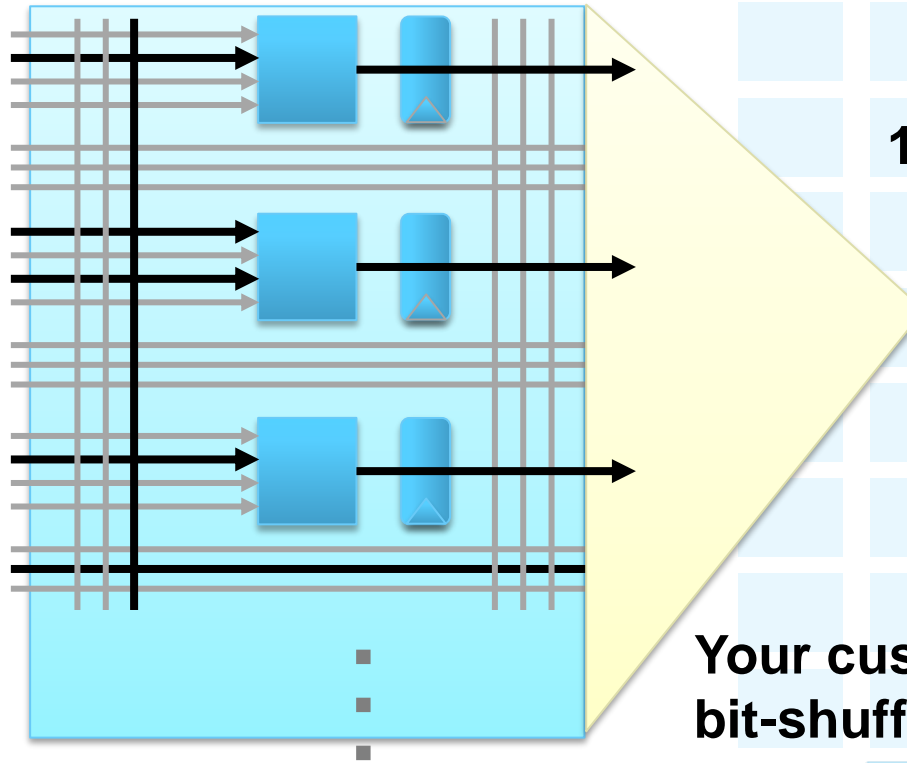


# FPGA Architecture: Flexible Interconnect



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

# FPGA Architecture: Custom Operations Using Basic Elements



16-bit add

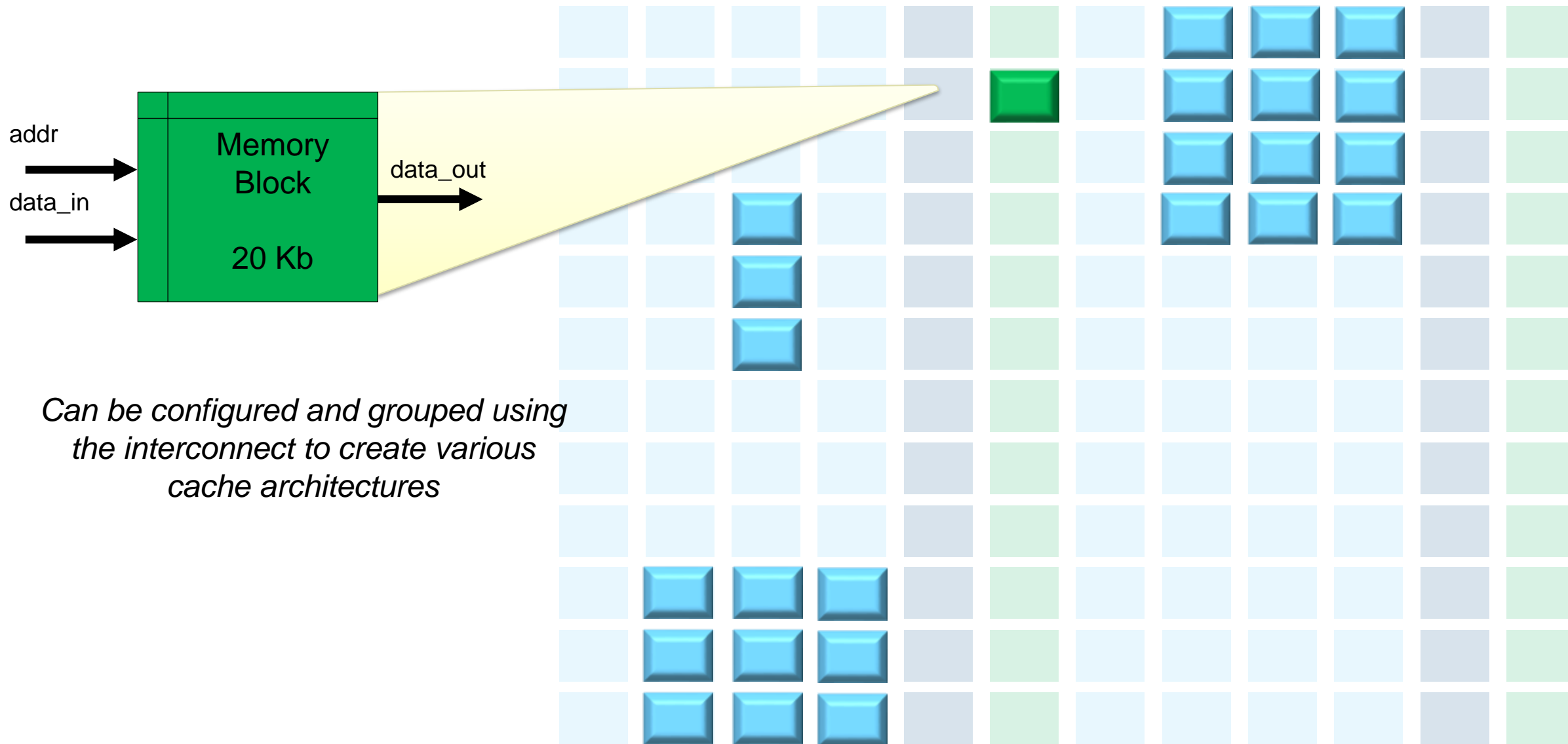
32-bit sqrt

Your custom 64-bit  
bit-shuffle and encode

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

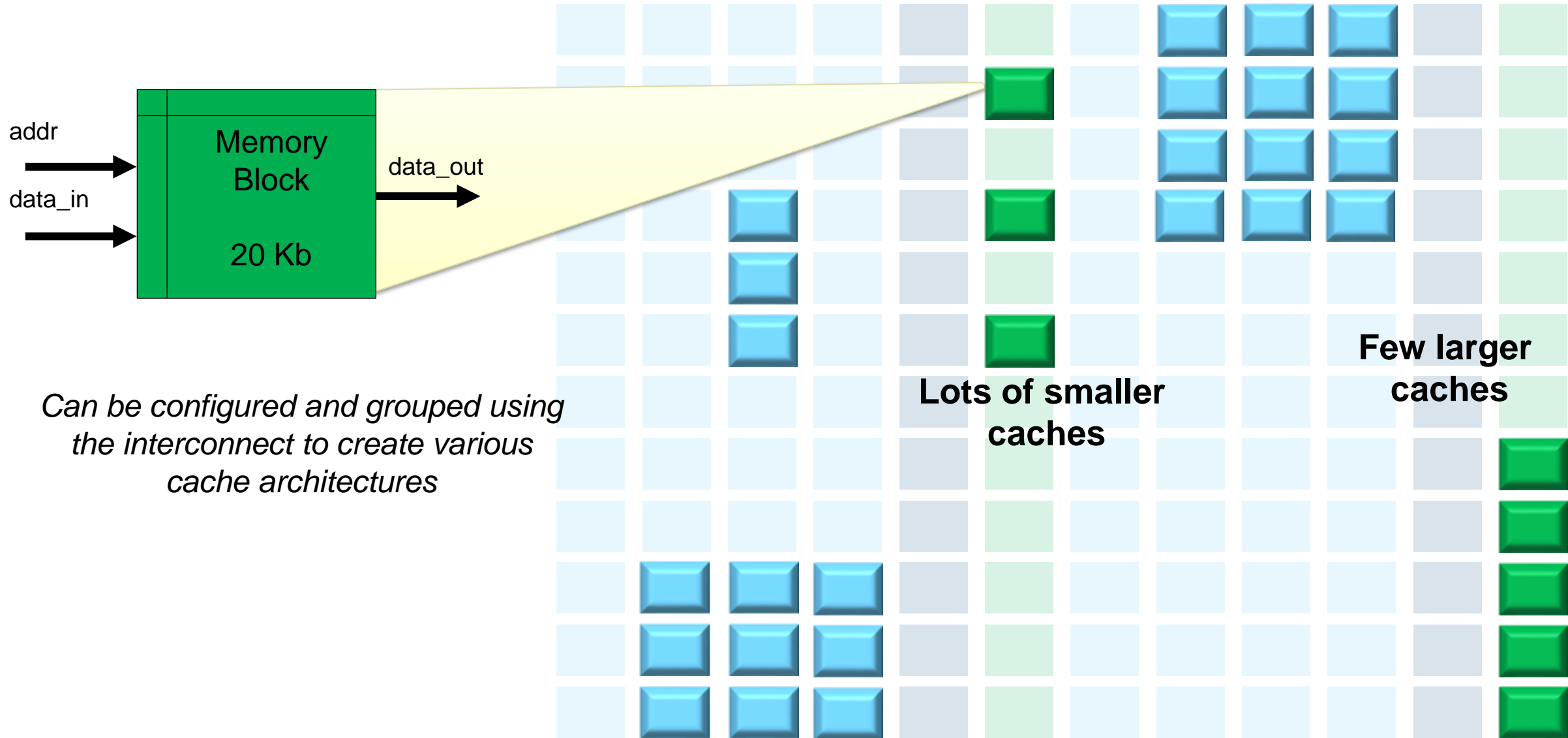


# FPGA Architecture: Memory Blocks




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

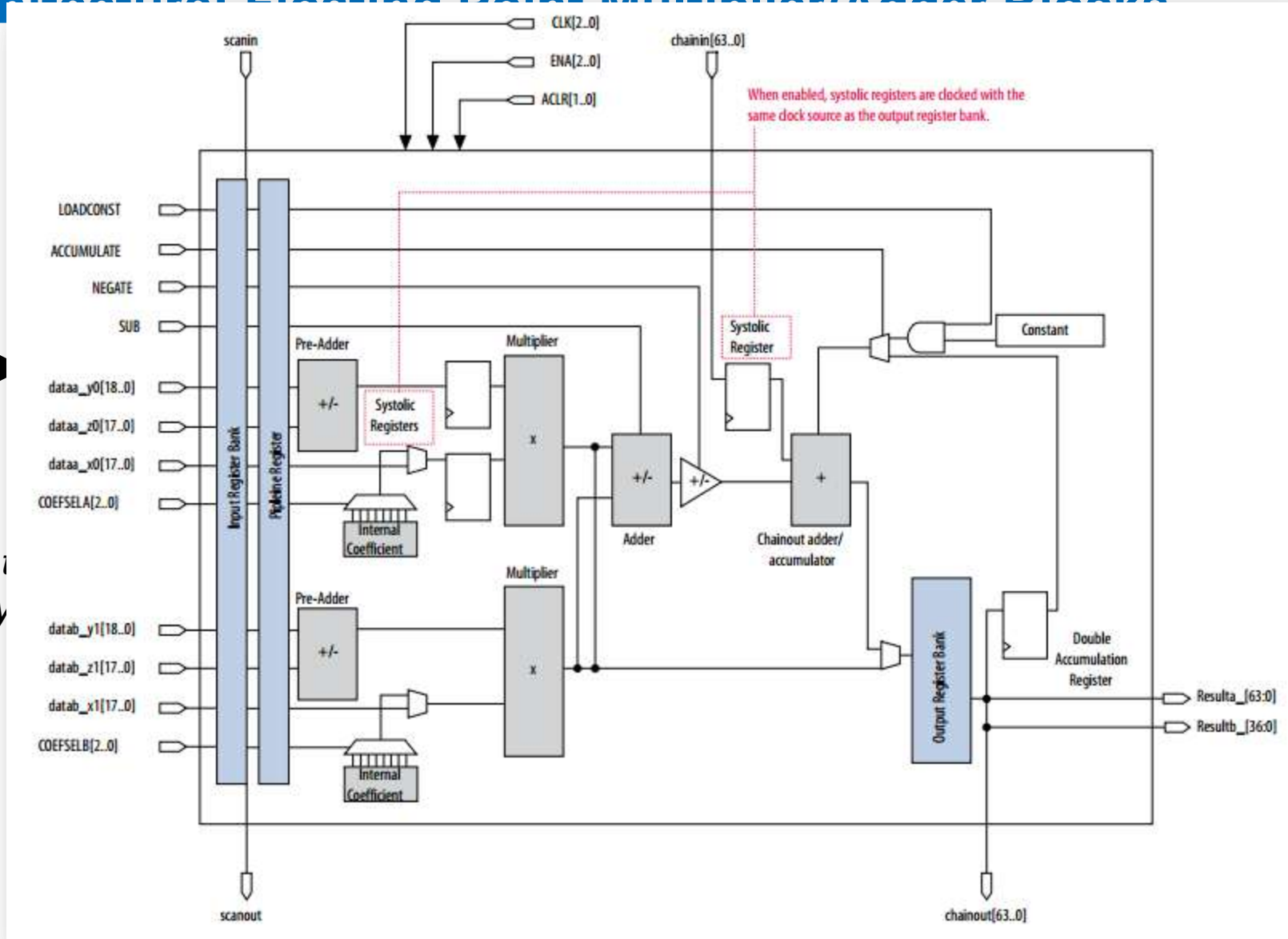
# FPGA Architecture: Memory Blocks



# FPGA Architecture: Floating Point Multiplier/Adder Blocks

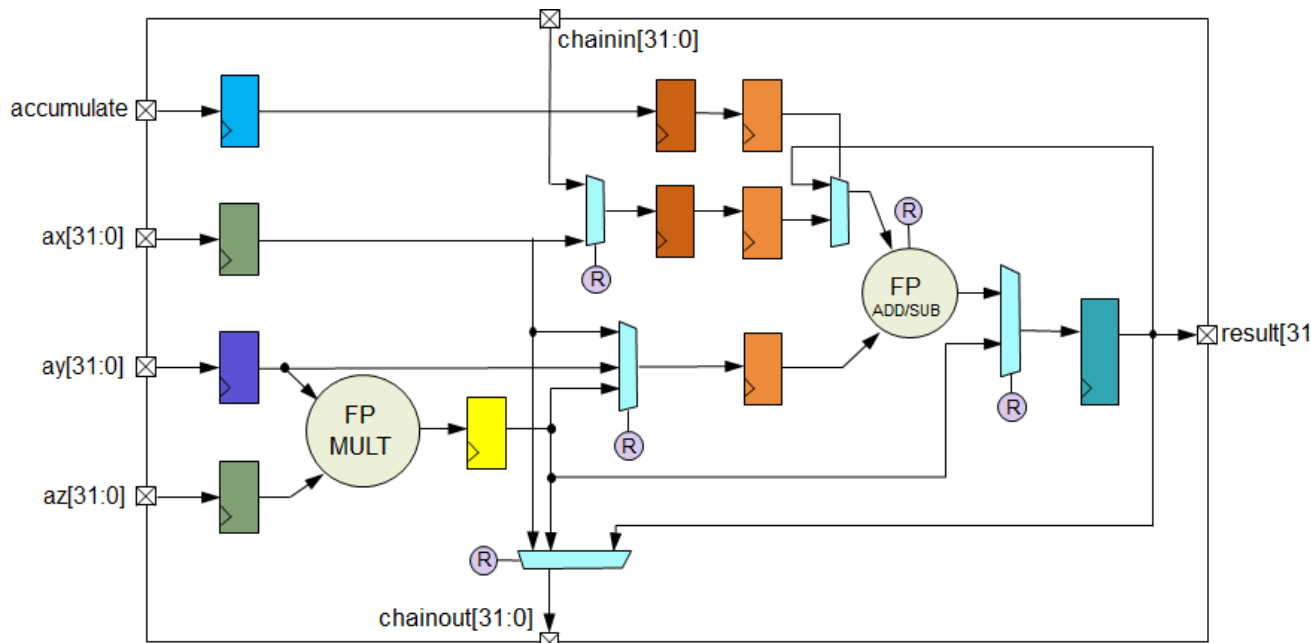
data\_in  


*Dedicated multiplier*

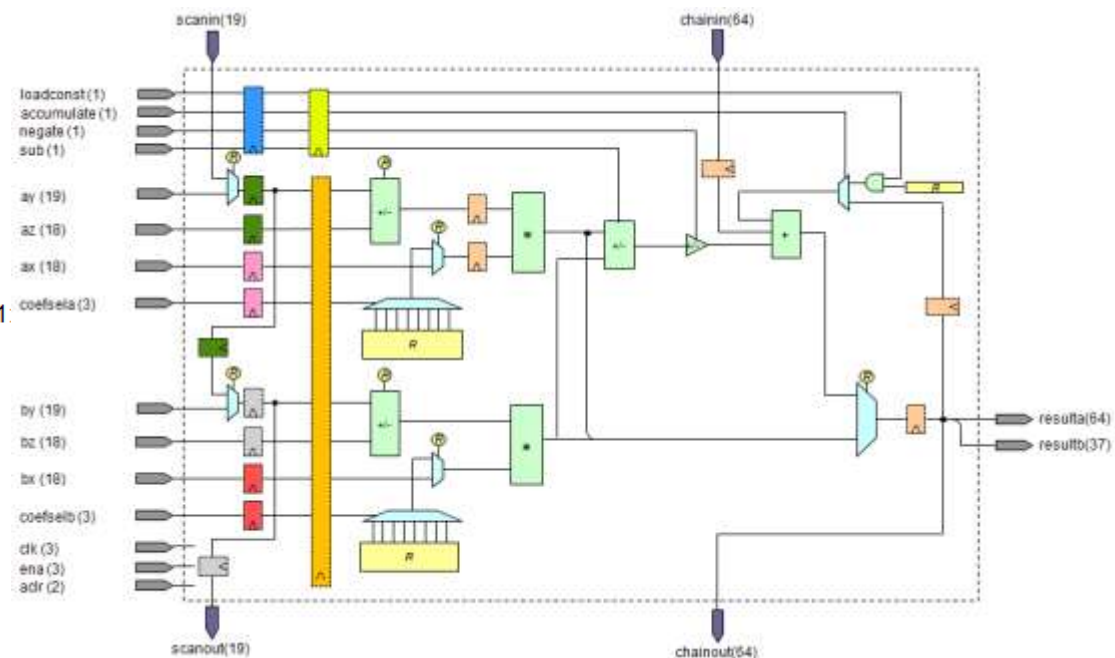


# DSP block architecture

## Floating-Point DSP

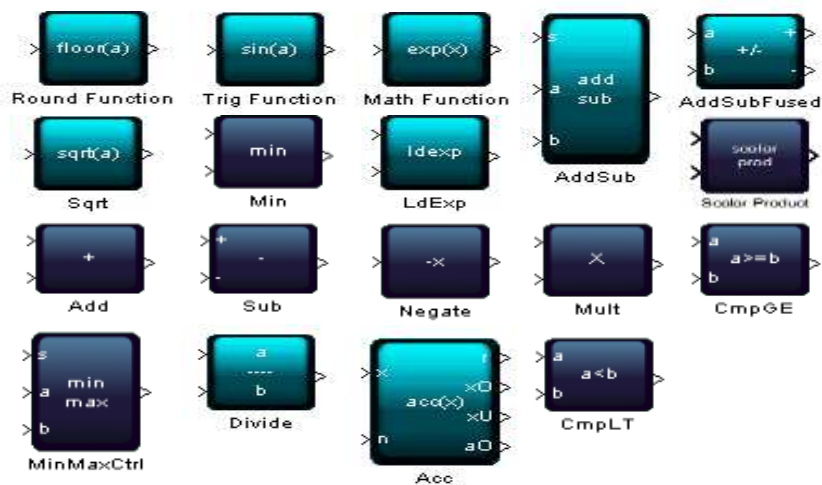




## Fixed-Point DSP



# 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

- FPArctanX
- FPArctanPi
- FPArccosX
- FPArccosPi
- FPArctanX
- FPArctanPi
- FPArctan2

## Conversion

- FXPToFP
- FPToFXP
- FPToFXPExpert
- FPToFXPFused
- FPToFP

## Macro Operators

- FPFusedHorner
- FPFusedHornerExpert
- FPFusedHornerMulti
- FPFusedMultiFunction

## Trigonometrics misc

- FPHypot
- FPRangeReduction

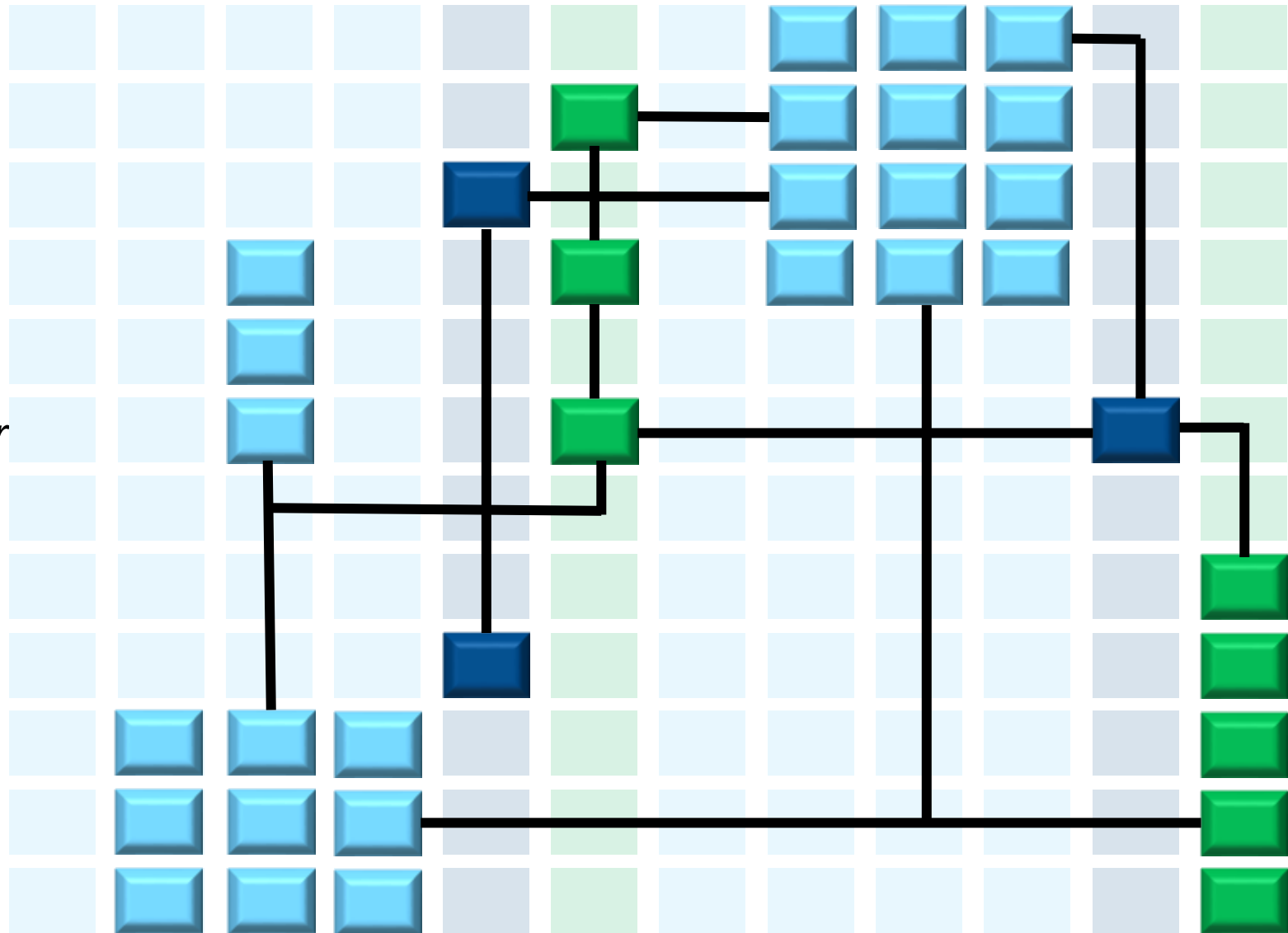
## Basic Floating Point

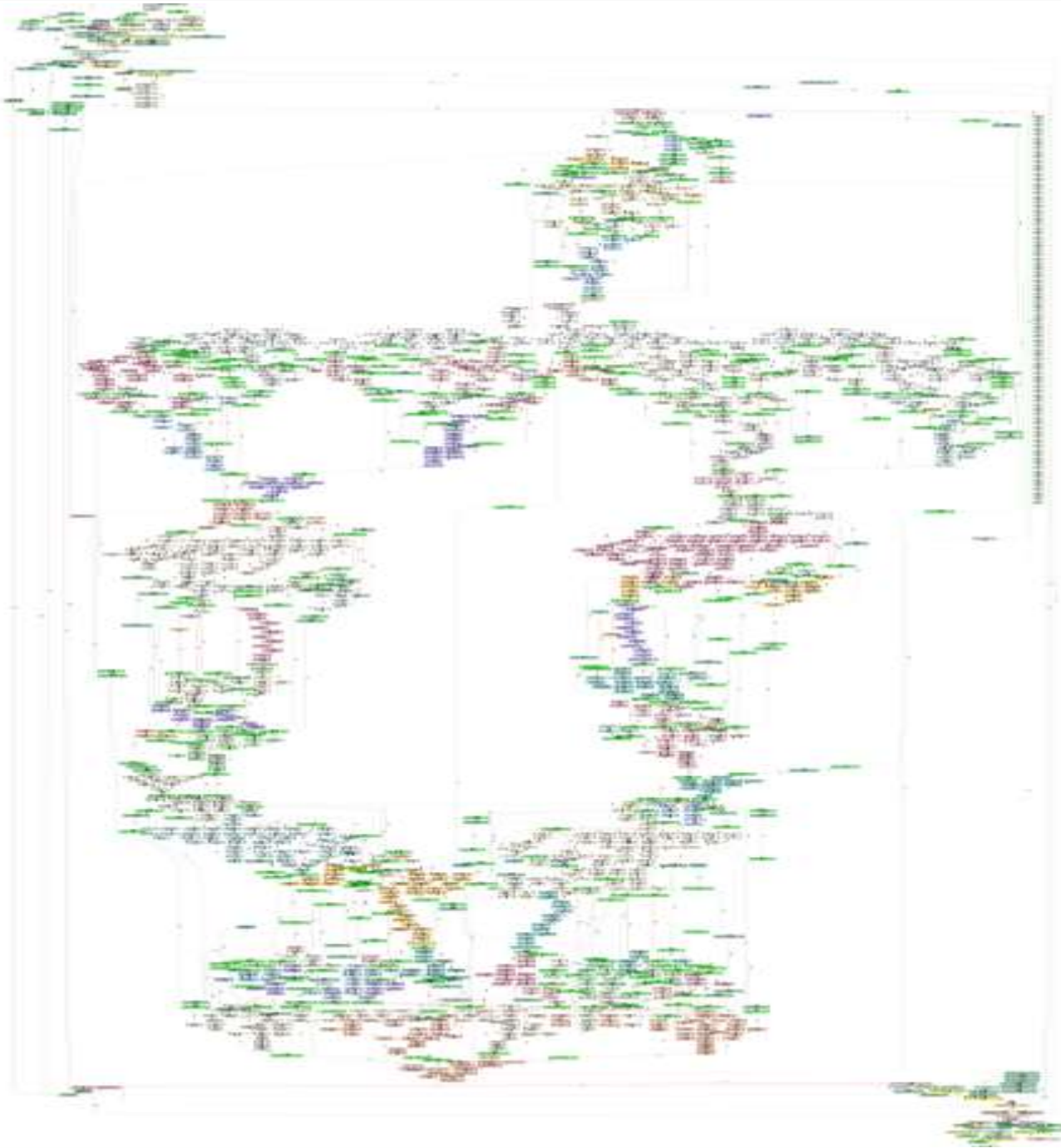
- FPAdd
- FPAddExpert
- FPAddN
- FPSubExpert
- FPAddSub \
- FPAddSubExpert
- FPFusedAddSub
- FPMul
- FPMulExpert
- FPConstMul
- FPAcc
- FPSqrt
- FPDivSqrt
- FPRecipSqrt
- FPCbrt
- FPDiv
- FPInverse
- FPFloor
- FPCEil
- FPRound
- FPRint
- FPFrac
- FPMOD
- FPDim
- FPAbs
- FPMIn
- FPMMax
- FPMInAbs
- FPMMaxAbs
- 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

Up to **10**  
TFLOPS

Most  
Comprehensive  
**Security**



Heterogeneous up to  
**1TB/s**  
3D SIP Integration

Intel **14 nm**  
Tri-Gate

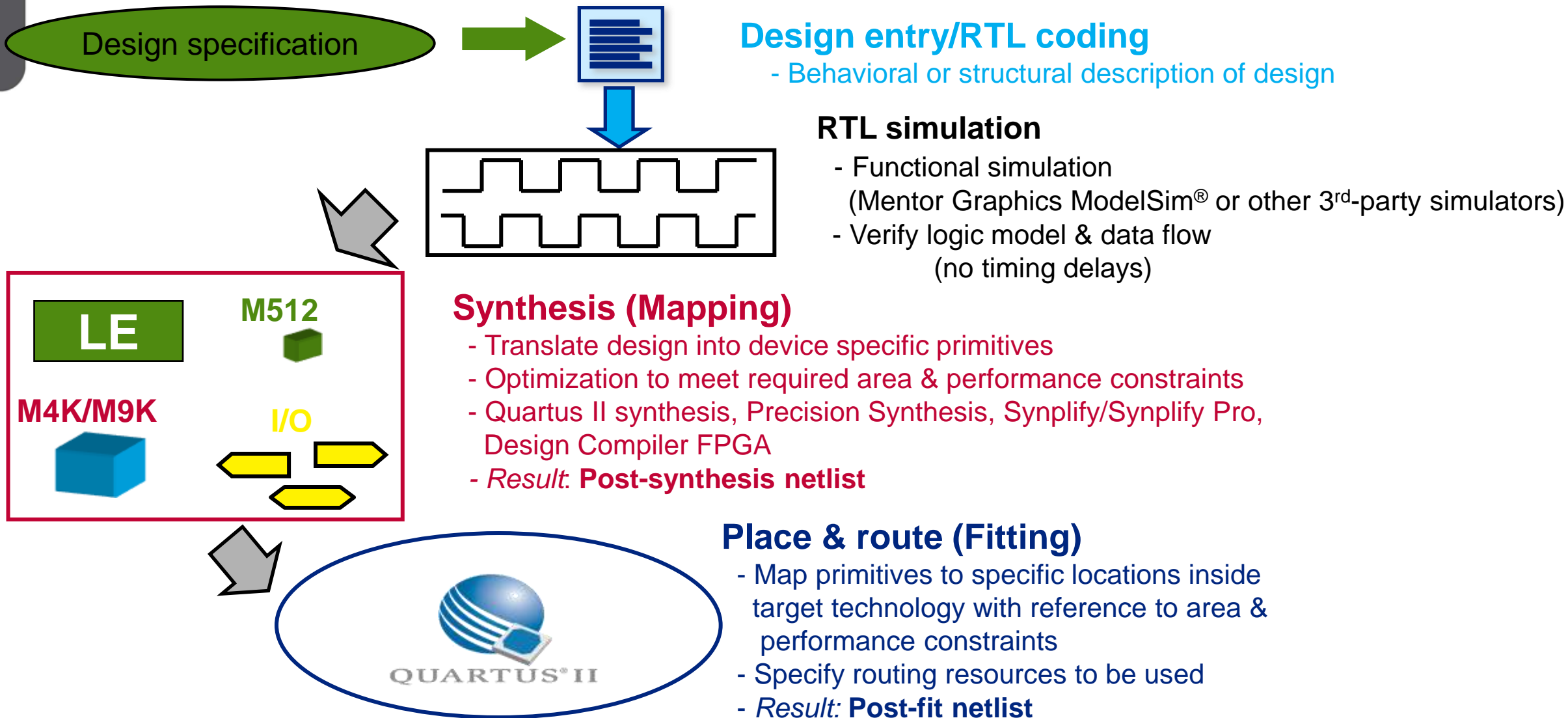
Quad-Core  
**Cortex-A53**  
ARM Processor



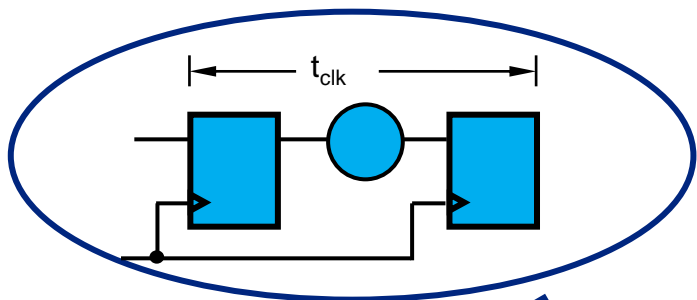
# Developing with FPGA



# Typical Programmable Logic Design Flow

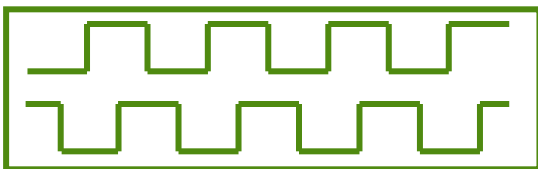


# Typical Programmable Logic Design Flow



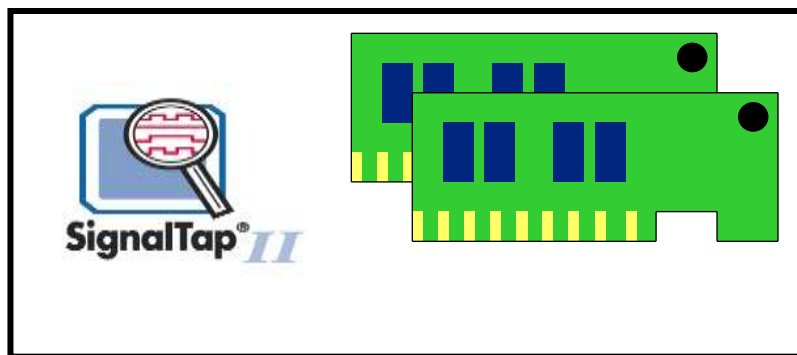
## Timing analysis

- Verify performance specifications were met
- Static timing analysis



## Gate level simulation (optional)

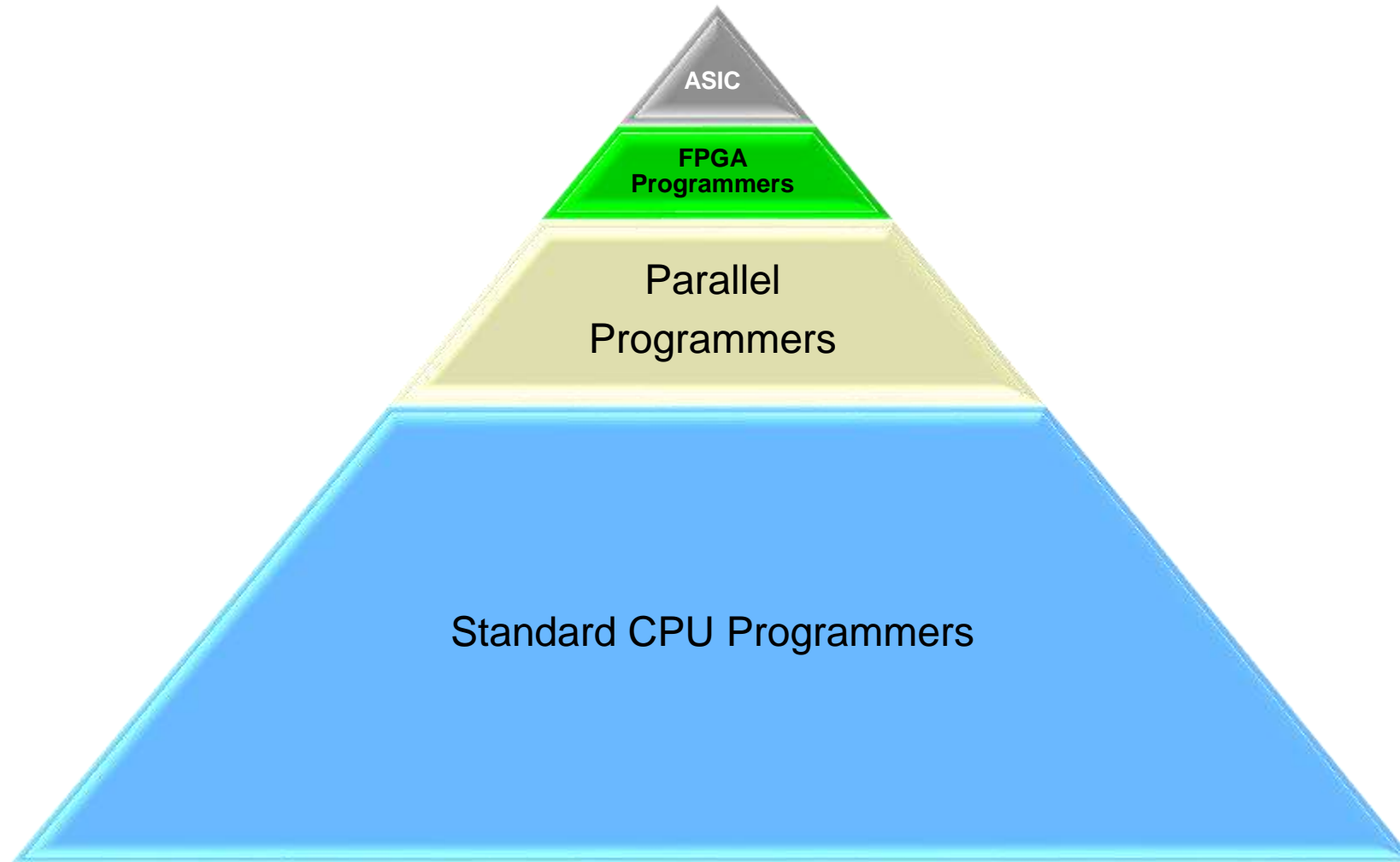
- Timing simulation
- Verify design will work in target technology



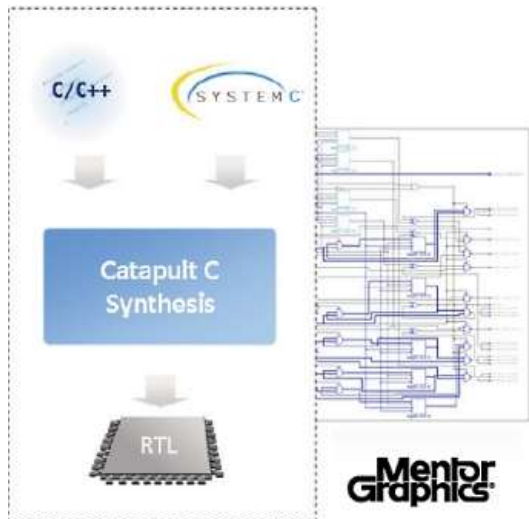
## PC board simulation & test

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

# Application Development Paradigm



# The magic trick ?



FpgaC

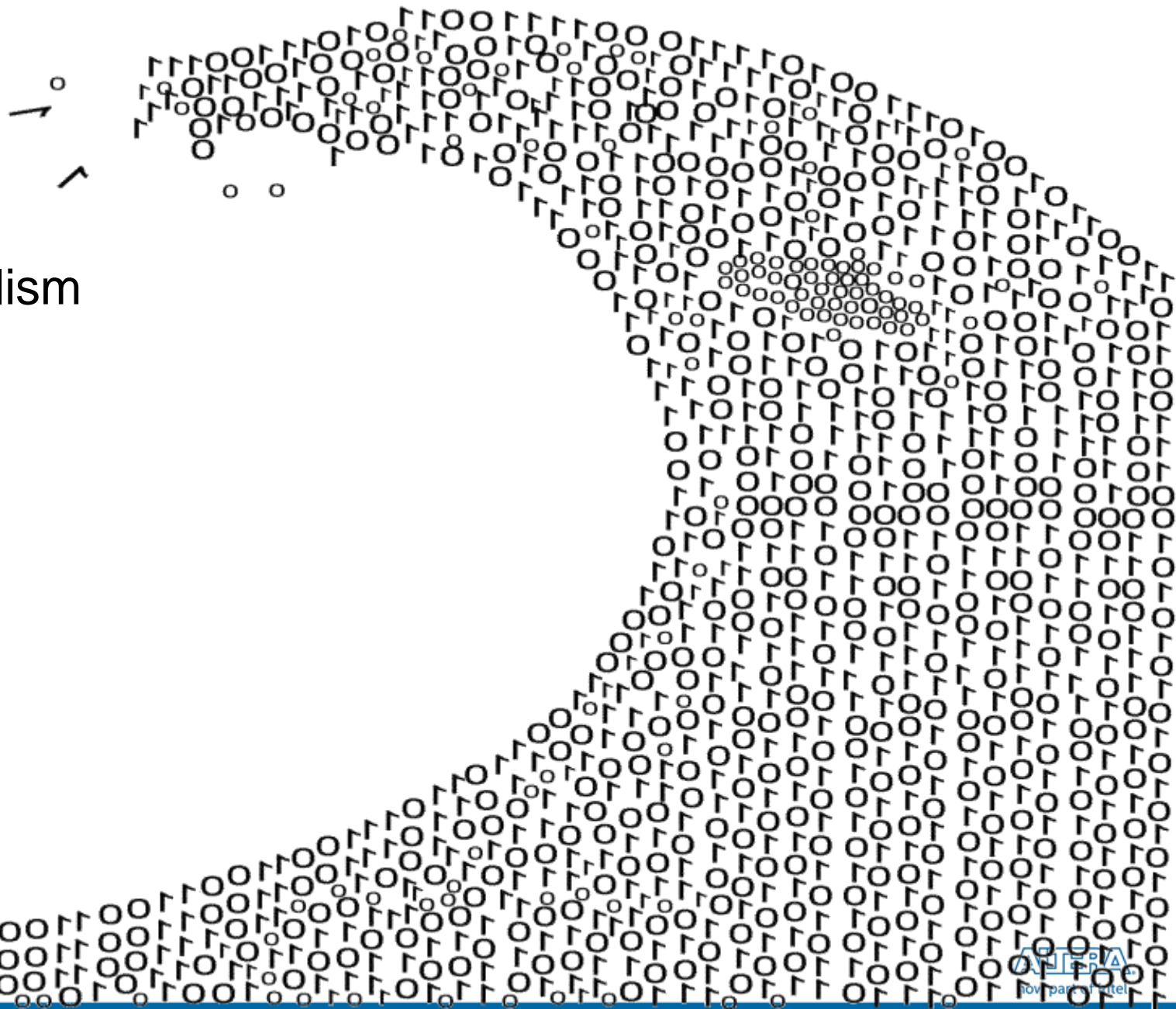


# OpenCL Concepts



## Setting the right expectations

- ◀ We have to think data parallelism
- ◀ Algorithms have to be rethink at the mathematics level.



# OpenCL C Language

## ◀ Derived from ISO C99

- No standard C99 headers, function pointers, recursion, variable length arrays, and bit fields

## ◀ Additions to the language for parallelism

- Work-items and workgroups
- Vector types
- Synchronization

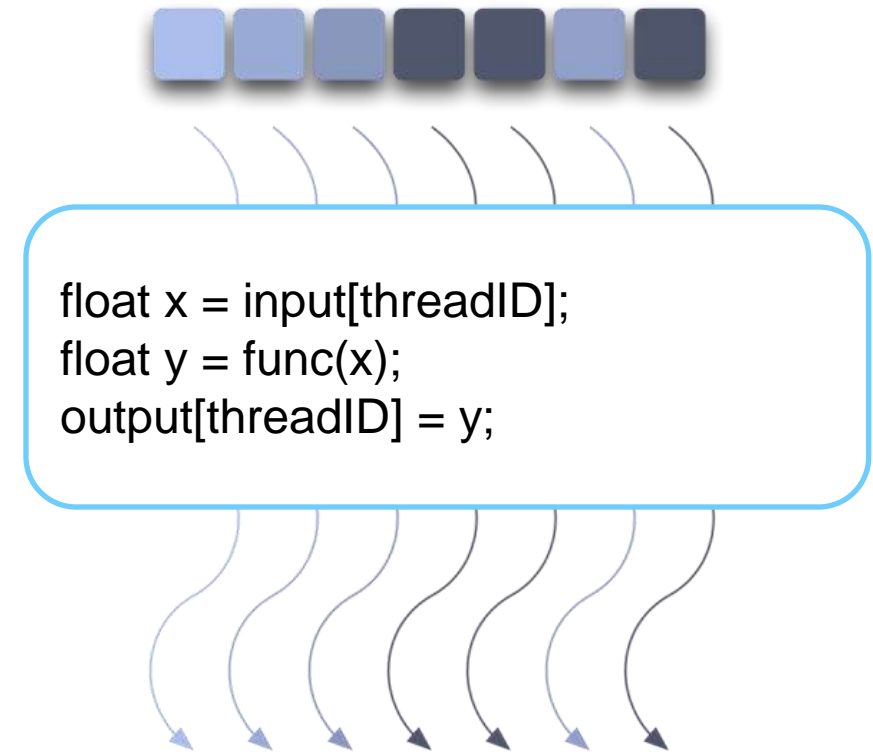
## ◀ Address space qualifiers

## ◀ Built-in functions

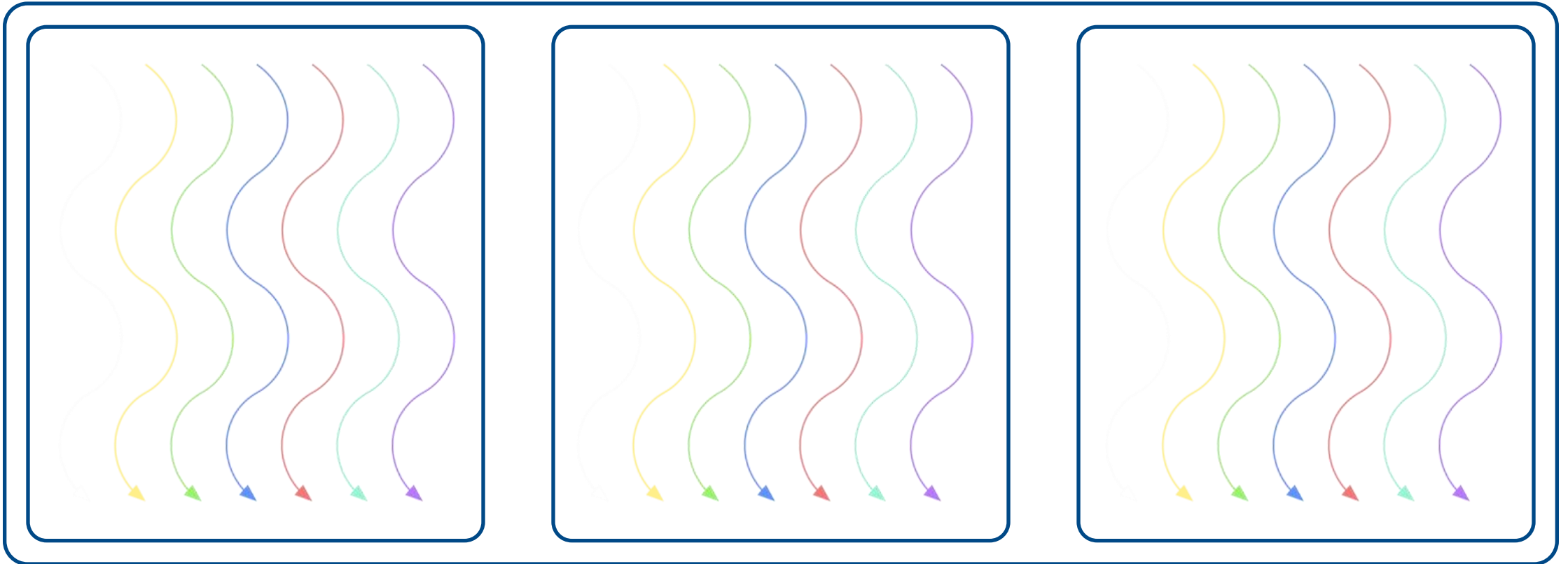


## OpenCL Kernels: Parallel Threads

- ▶ A **kernel** is a function executed on an Accelerator device
  - Array of threads, in parallel
- ▶ All threads (or **work-items**) execute the same code, can take different paths
- ▶ Each thread has an ID
  - Select input/output data
  - Control decisions

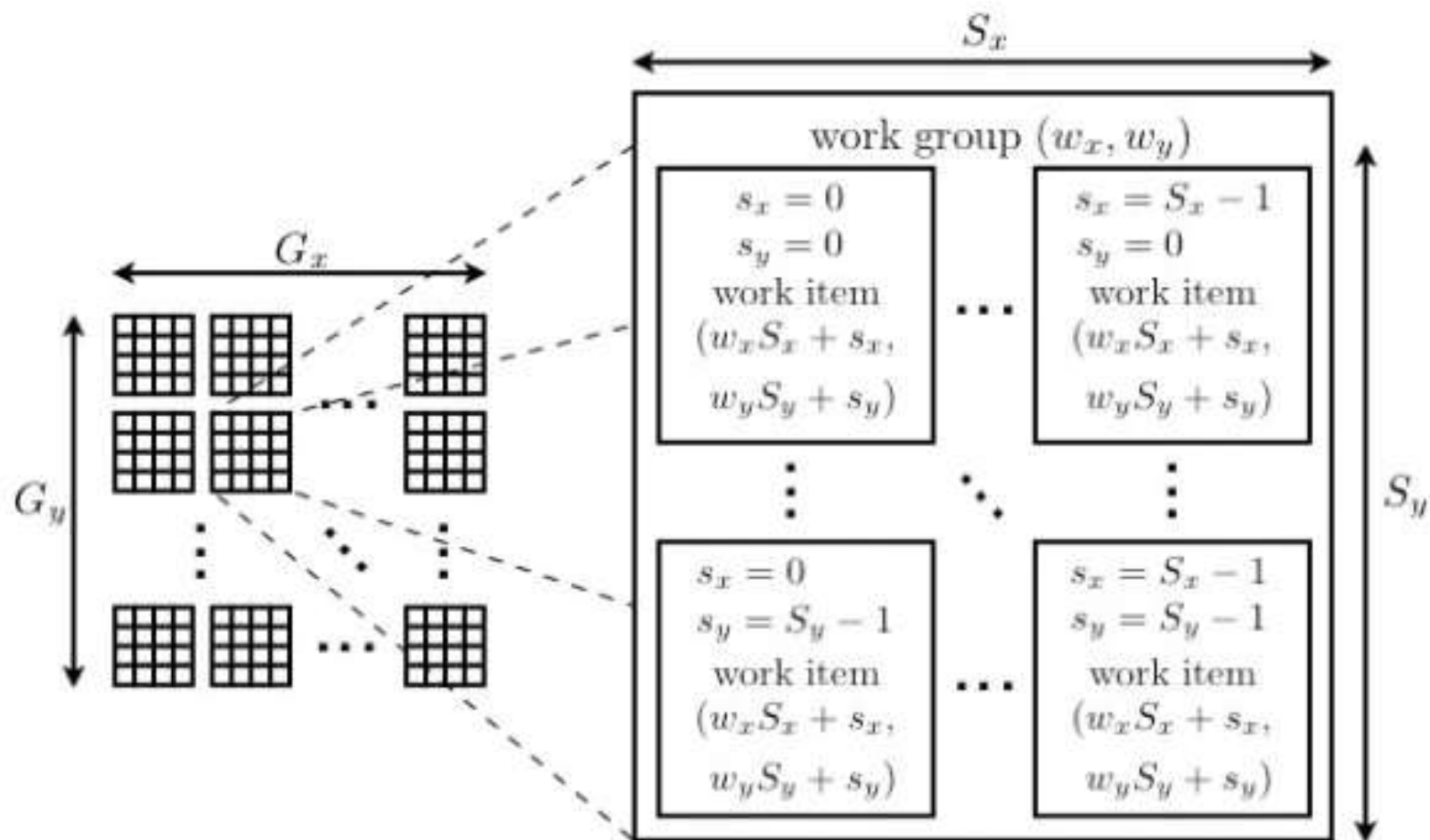


## OpenCL Kernels: Divide into Workgroups



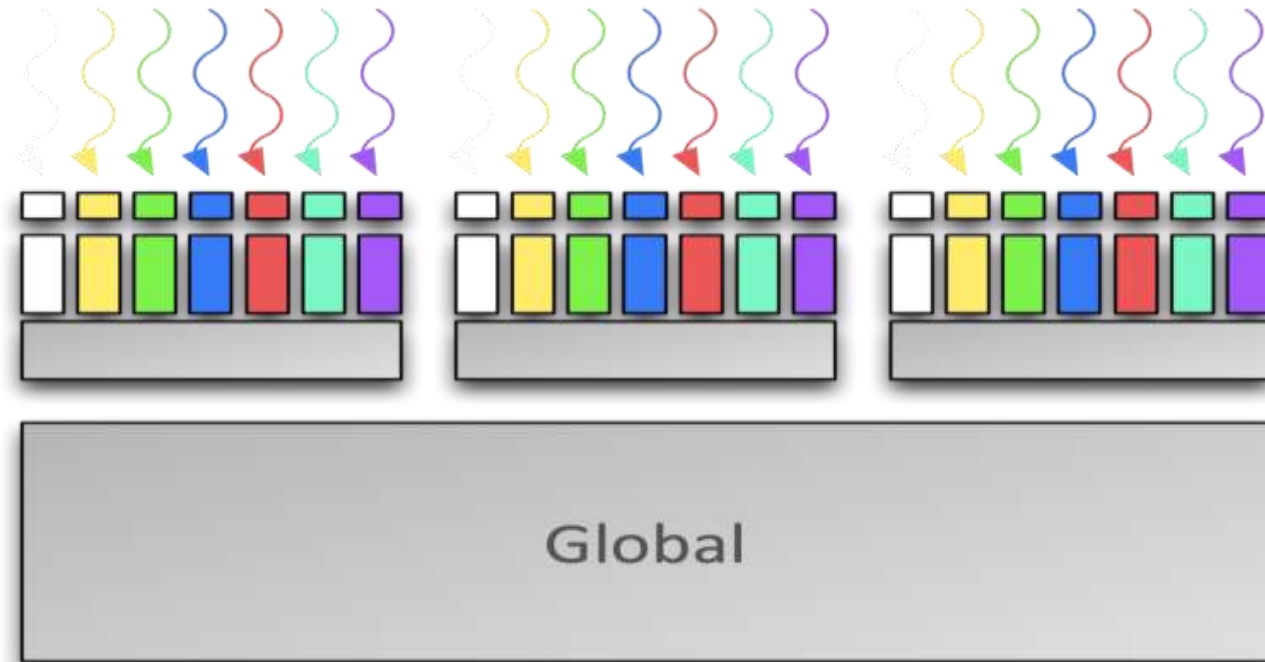
- Threads in **workgroups** can cooperate with each through fast local (on-chip) memory

# Data Organization



# Memory hierarchy

- Thread:
  - Registers
- Thread:
  - Private memory
- Workgroups:
  - Local or Shared memory
- All Workgroups:
  - Global memory



# OpenCL: abstracting FPGA away

The Altera logo consists of the word "ALTERA" in a bold, blue, sans-serif font. The letters are outlined, giving it a three-dimensional appearance. A registered trademark symbol (®) is located to the upper right of the letter "A".

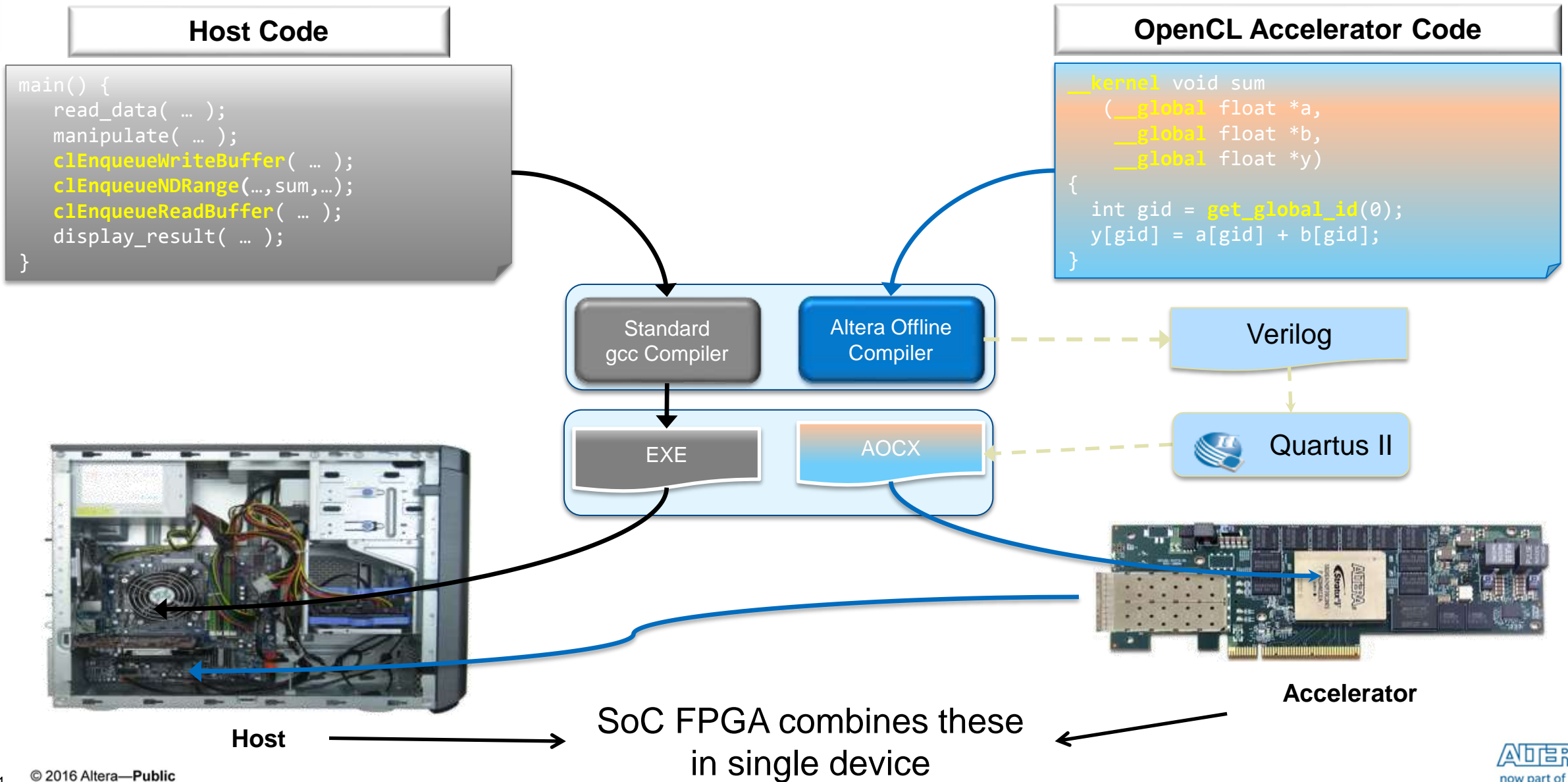
**ALTERA**®

now part of Intel

# 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

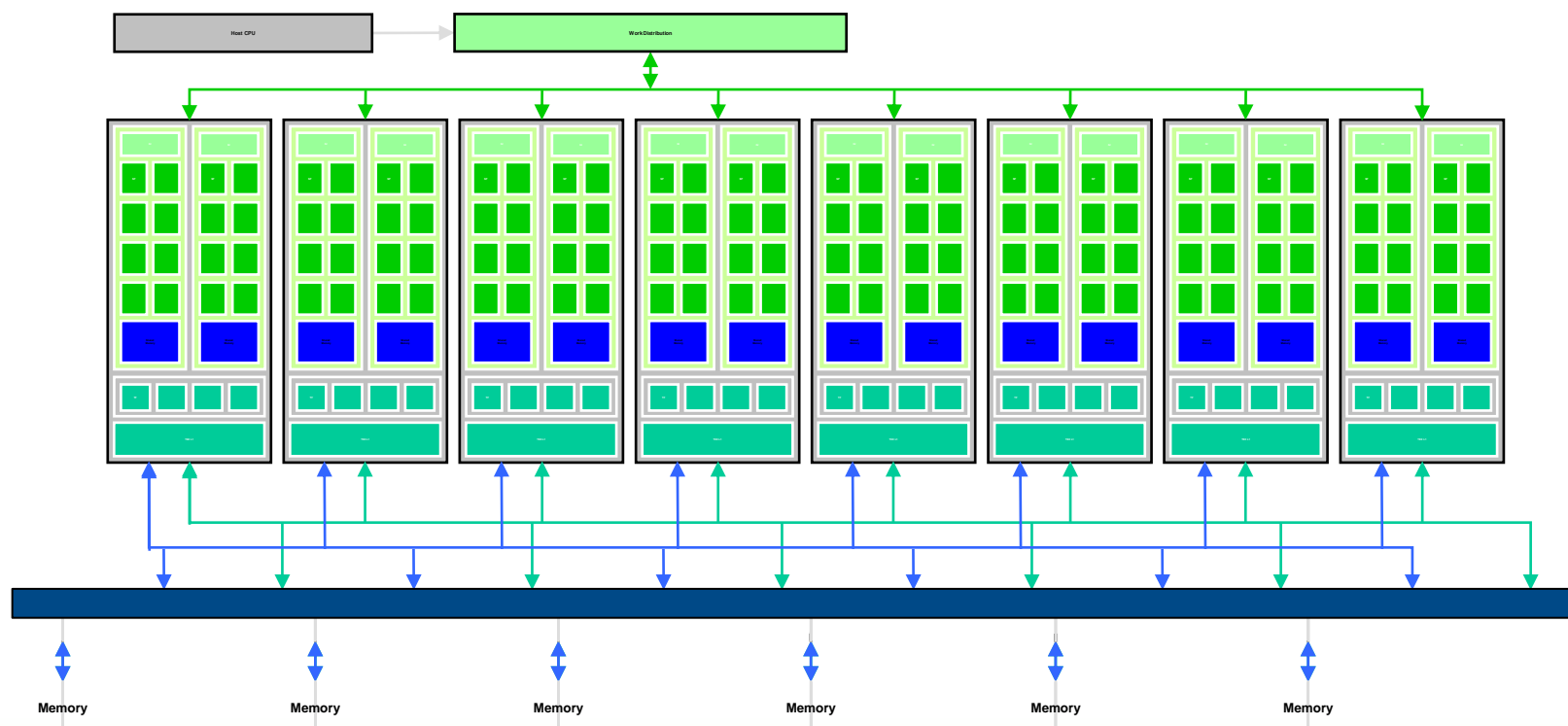


# OpenCL on GPU/Multi-Core CPU Architectures

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



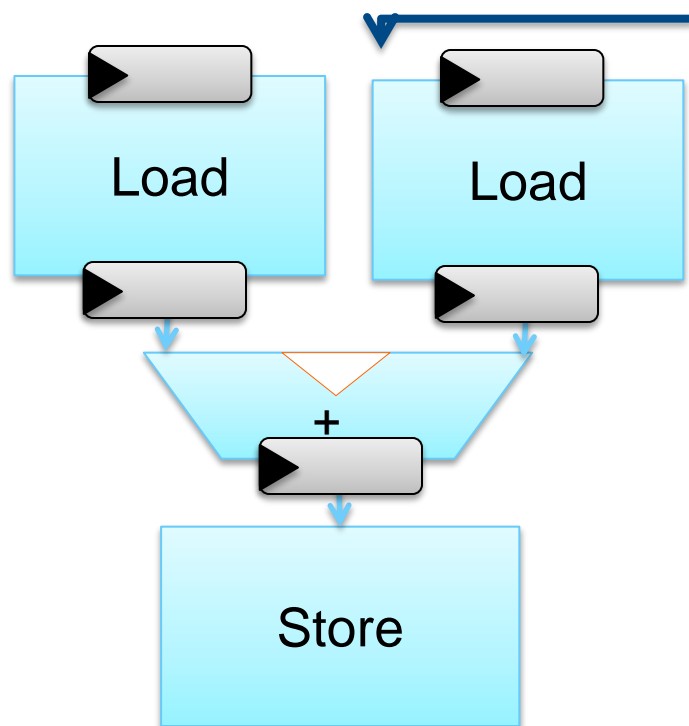


## OpenCL on FPGA

- ◀ 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*

## Example Pipeline for Vector Add



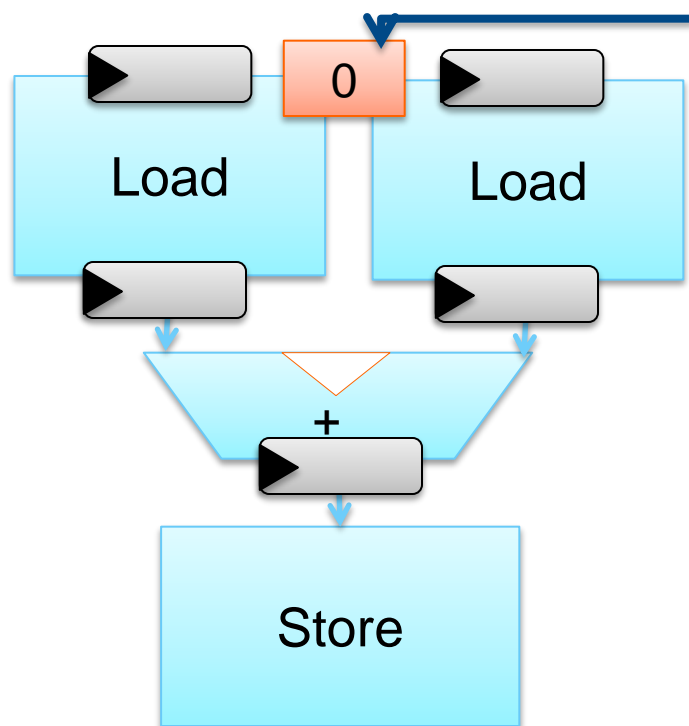
8 threads for vector add example



*Thread IDs*

- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

## Example Pipeline for Vector Add



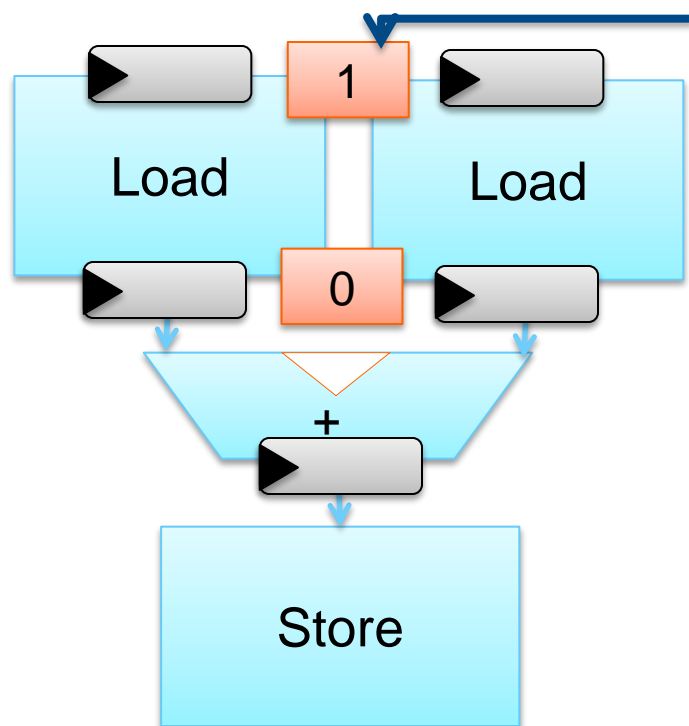
8 threads for vector add example



*Thread IDs*

- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

## Example Pipeline for Vector Add



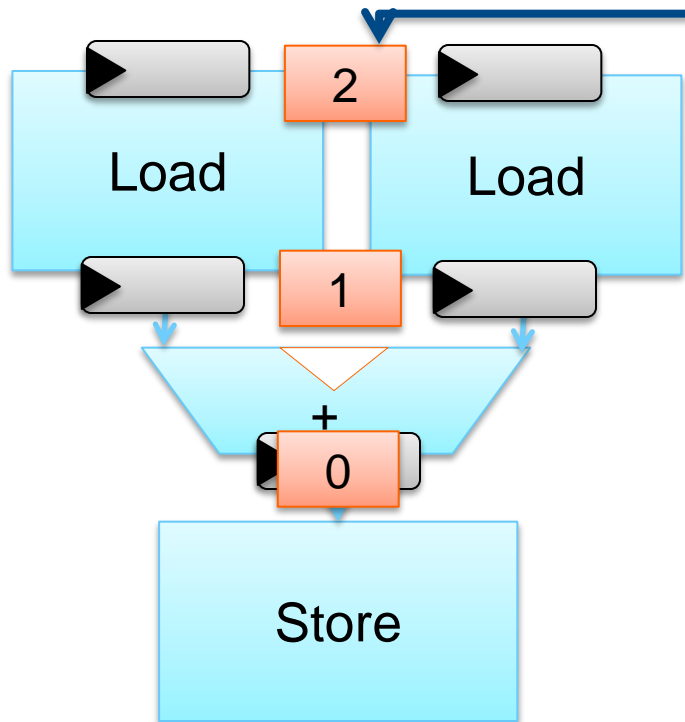
8 threads for vector add example



*Thread IDs*

- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

## Example Pipeline for Vector Add



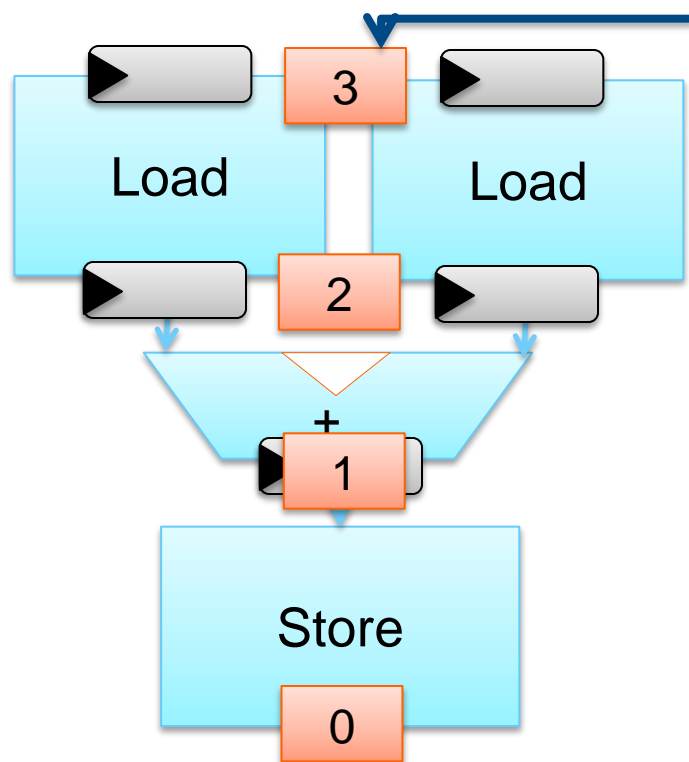
8 threads for vector add example



*Thread IDs*

- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

## Example Pipeline for Vector Add



8 threads for vector add example



*Thread IDs*

- On each cycle the portions of the pipeline are processing different threads
- While thread 2 is being loaded, thread 1 is being added, and thread 0 is being stored

# Mapping a simple program to an FPGA

High-level code

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

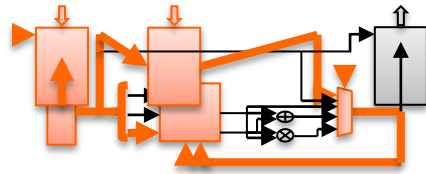


CPU instructions

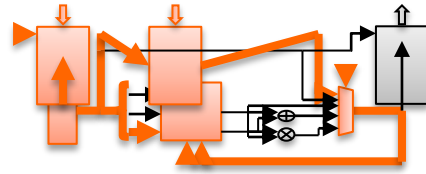
```
R0 ← Load Mem[100]  
R1 ← Load Mem[101]  
R2 ← Load #42  
R2 ← Mul R1, R2  
R0 ← Add R2, R0  
Store R0 → Mem[100]
```

# CPU activity, step by step

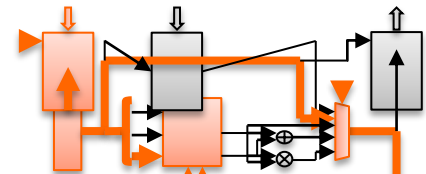
R0 ← Load Mem[100]



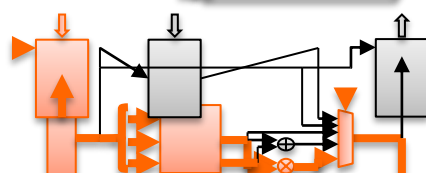
R1 ← Load Mem[101]



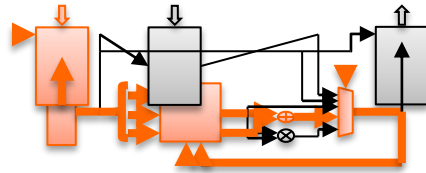
R2 ← Load #42



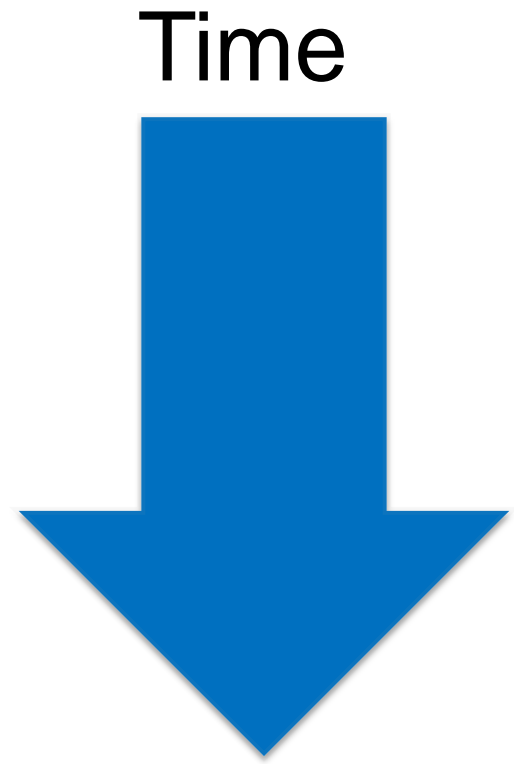
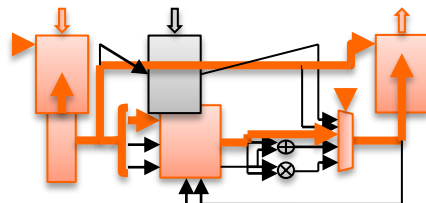
R2 ← Mul R1, R2



R0 ← Add R2, R0



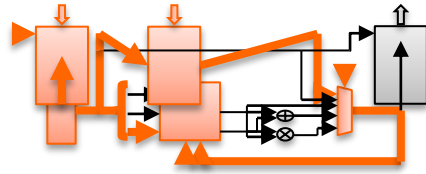
Store R0 → Mem[100]



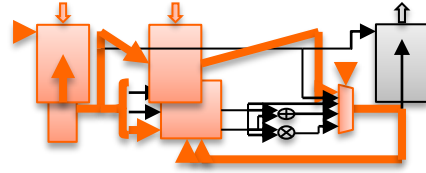


# On the FPGA we unroll the CPU hardware...

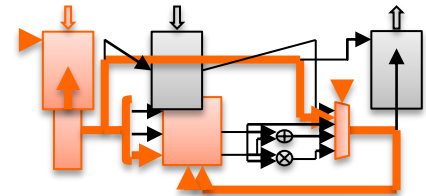
R0 ← Load Mem[100]



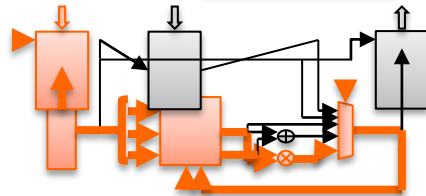
R1 ← Load Mem[101]



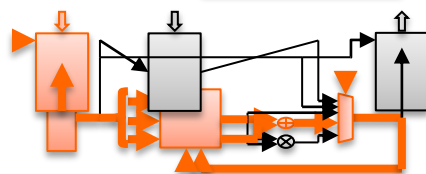
R2 ← Load #42



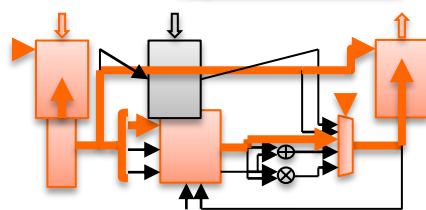
R2 ← Mul R1, R2



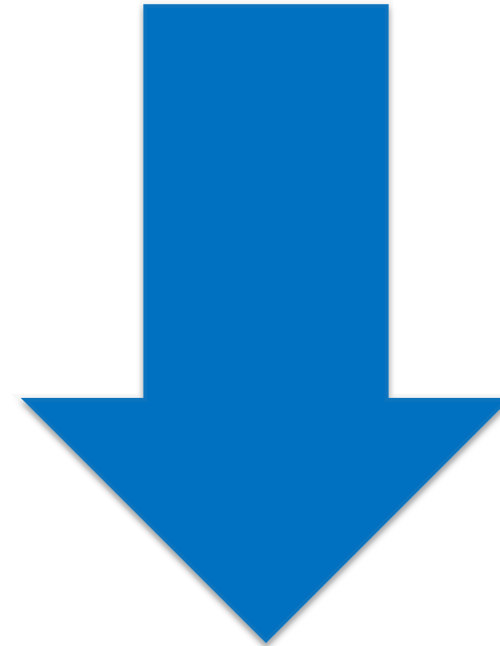
R0 ← Add R2, R0



Store R0 → Mem[100]

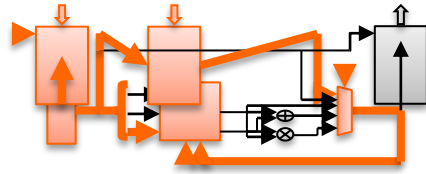


Space

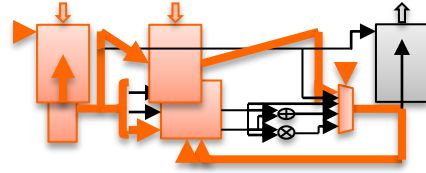


## ... and specialize by position

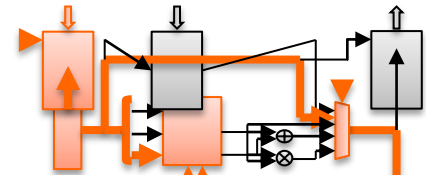
R0 ← Load Mem[100]



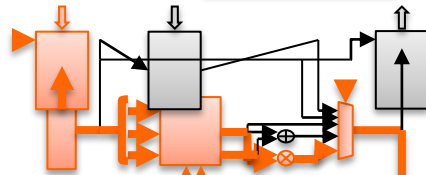
R1 ← Load Mem[101]



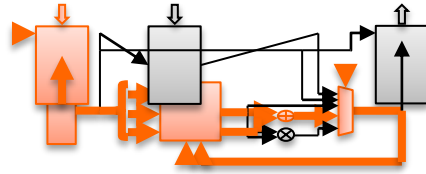
R2 ← Load #42



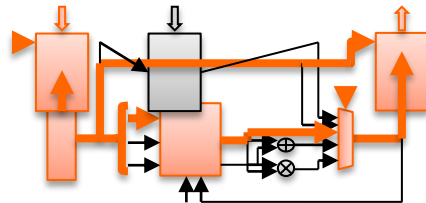
R2 ← Mul R1, R2



R0 ← Add R2, R0



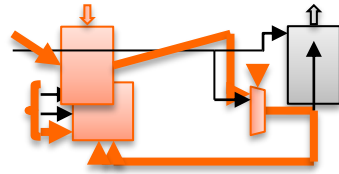
Store R0 → Mem[100]



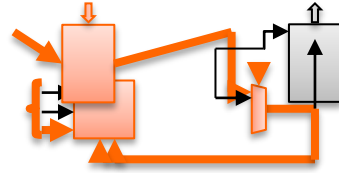
1. Instructions are fixed. Remove “Fetch”

## ... and specialize

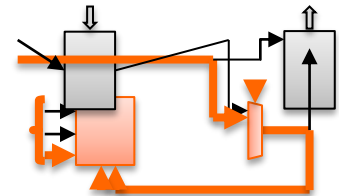
R0 ← Load Mem[100]



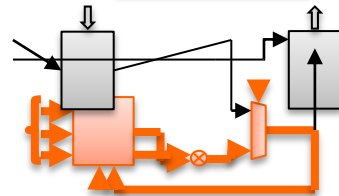
R1 ← Load Mem[101]



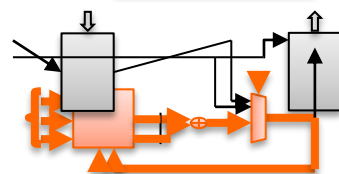
R2 ← Load #42



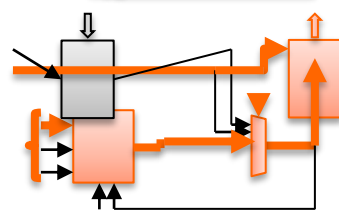
R2 ← Mul R1, R2



R0 ← Add R2, R0



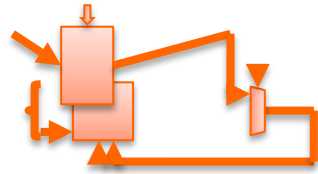
Store R0 → Mem[100]



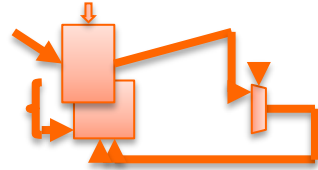
1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops

## ... and specialize

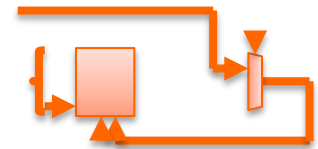
R0 ← Load Mem[100]



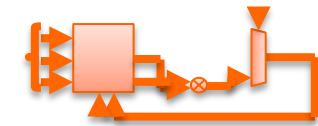
R1 ← Load Mem[101]



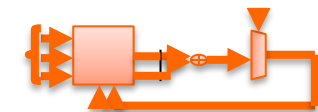
R2 ← Load #42



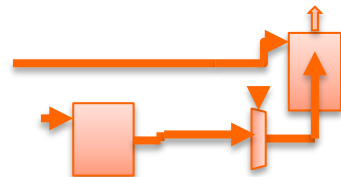
R2 ← Mul R1, R2



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]

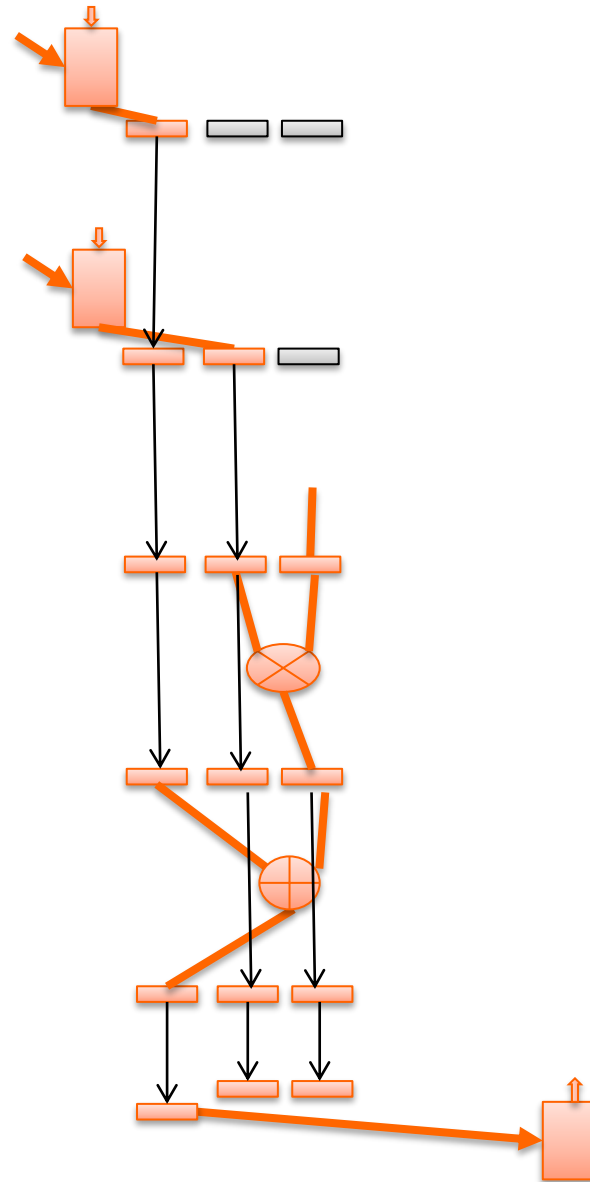
R1 ← Load Mem[101]

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]

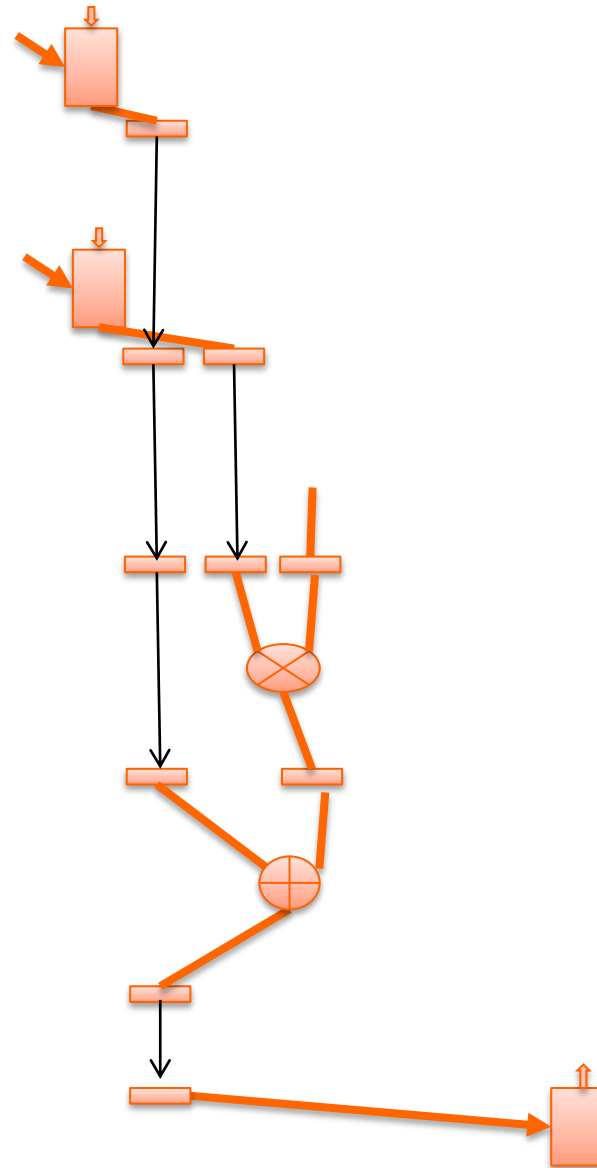
R1 ← Load Mem[101]

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]

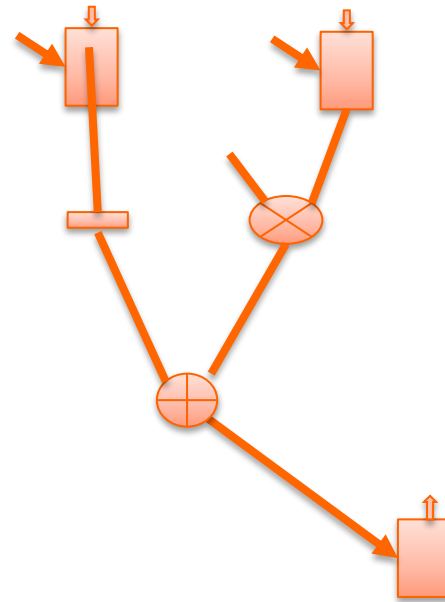
R1 ← Load Mem[101]

R2 ← Load #42

R2 ← Mul R1, R2

R0 ← Add R2, R0

Store R0 → Mem[100]



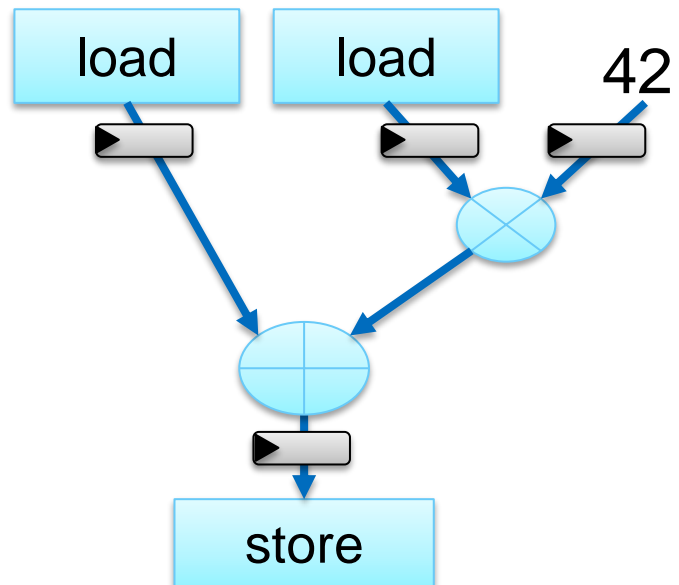
1. Instructions are fixed. Remove “Fetch”
2. Remove unused ALU ops
3. Remove unused Load / Store
4. Wire up registers properly! And propagate state.
5. Remove dead data.
6. Reschedule!

# 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**

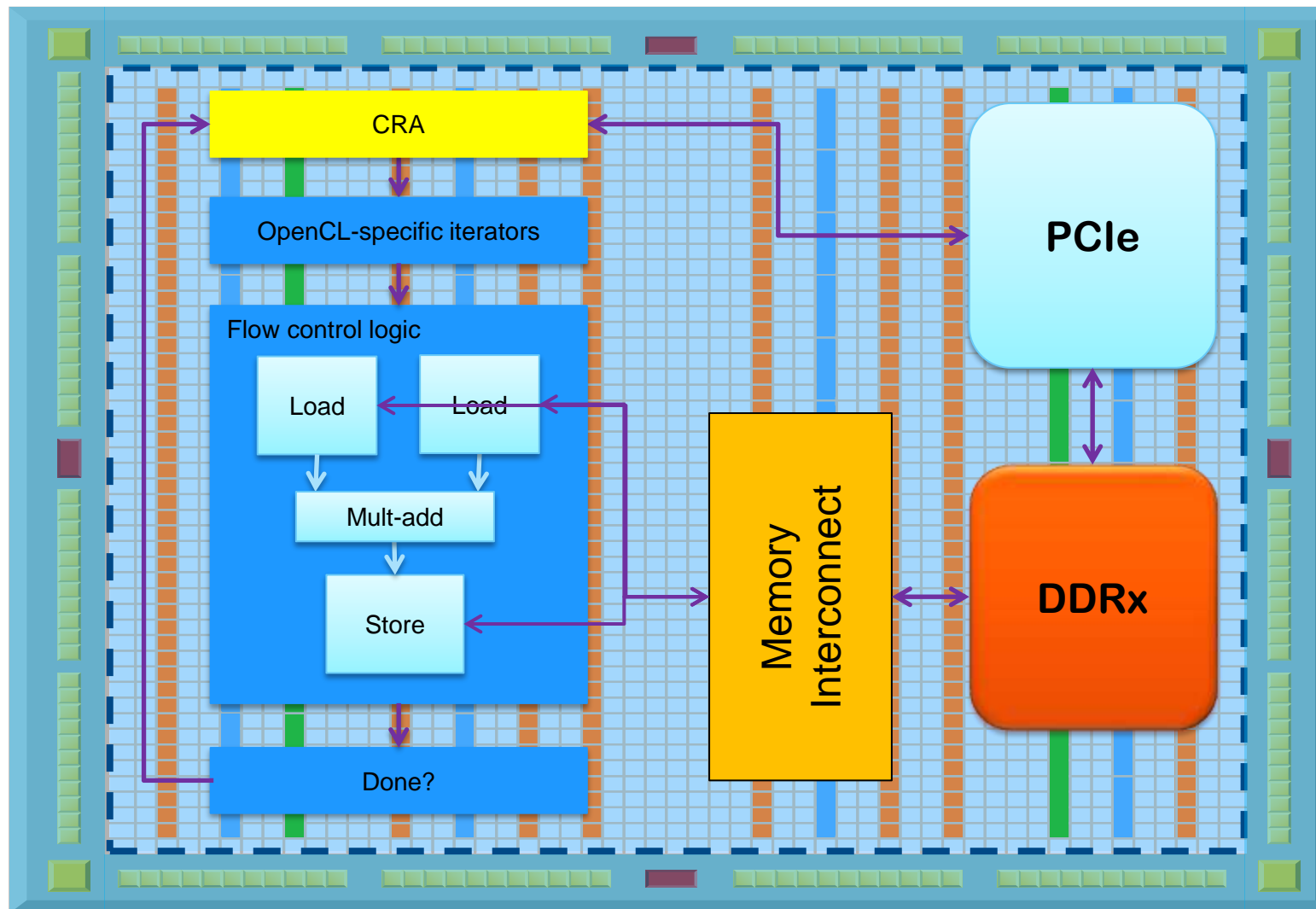
**Memory size & configuration**

**Efficiency:**

**Throughput / Latency / Power**



# What Hardware do we produce?

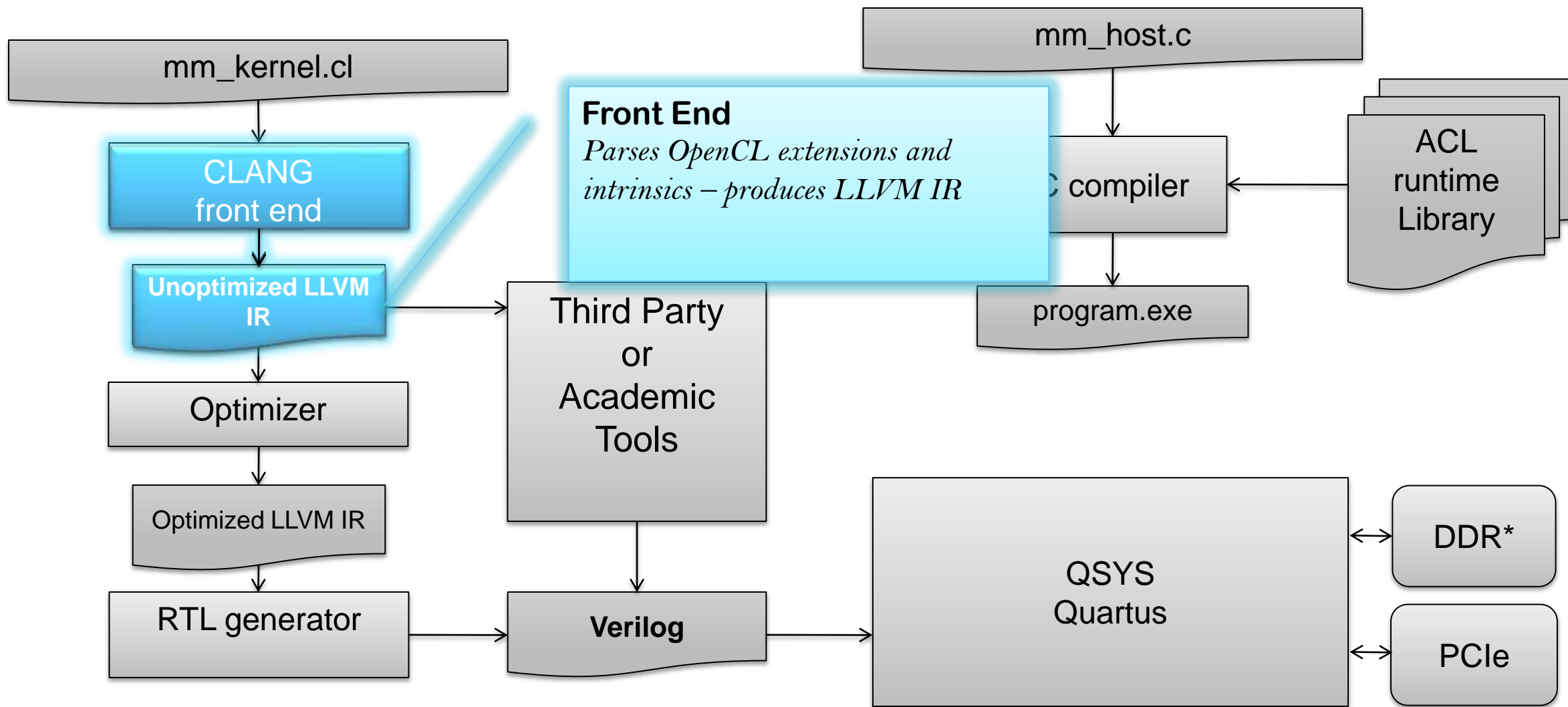


# ALTERA SDK for OpenCL

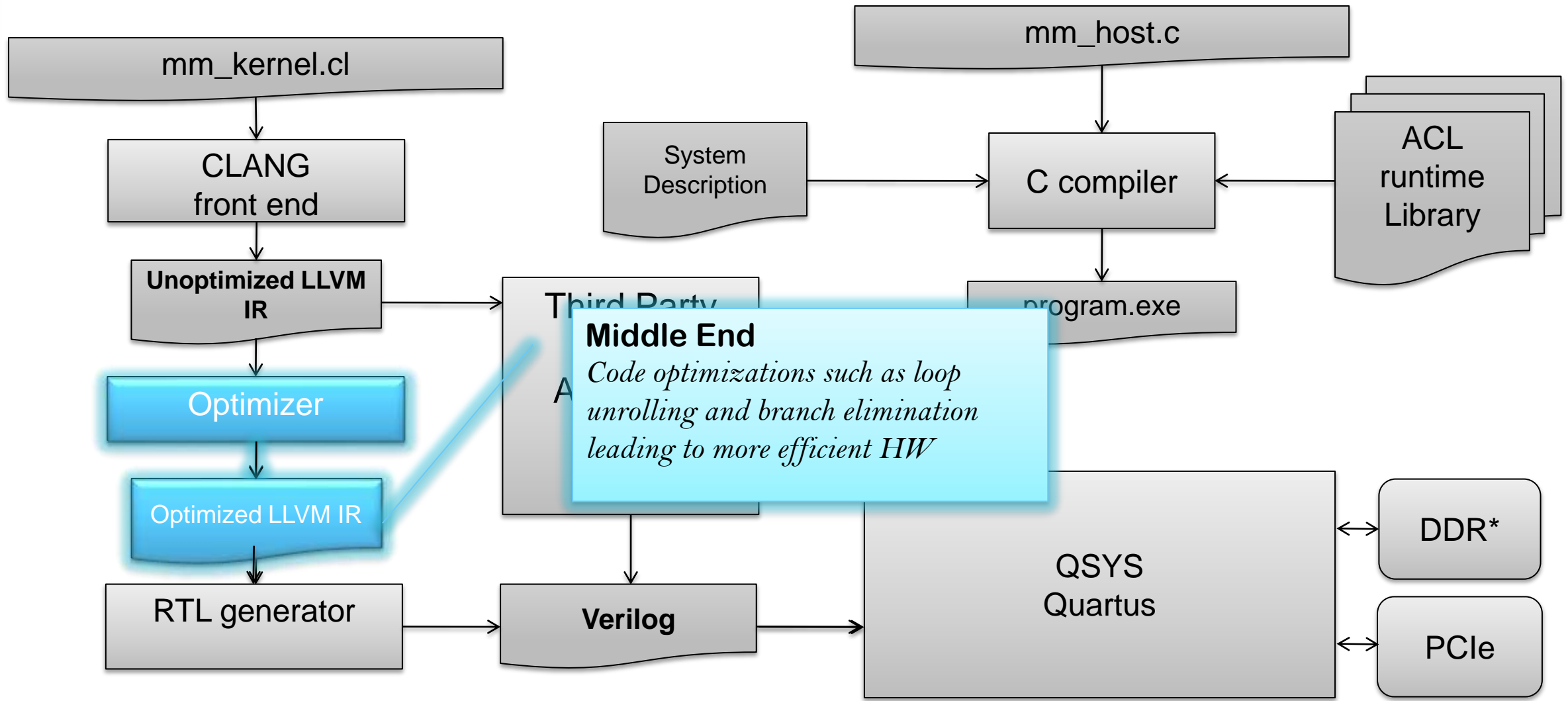
Development Flow & Features



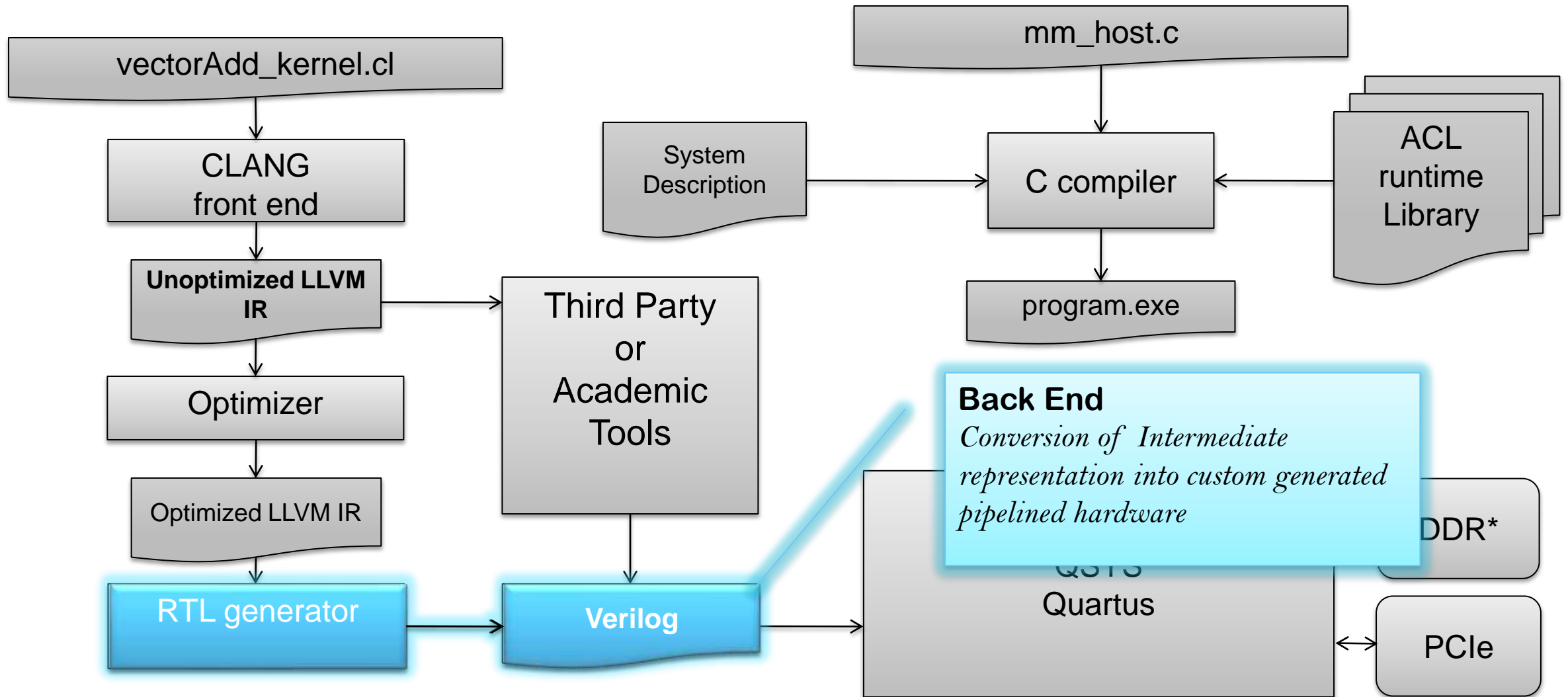
# OpenCL CAD Flow



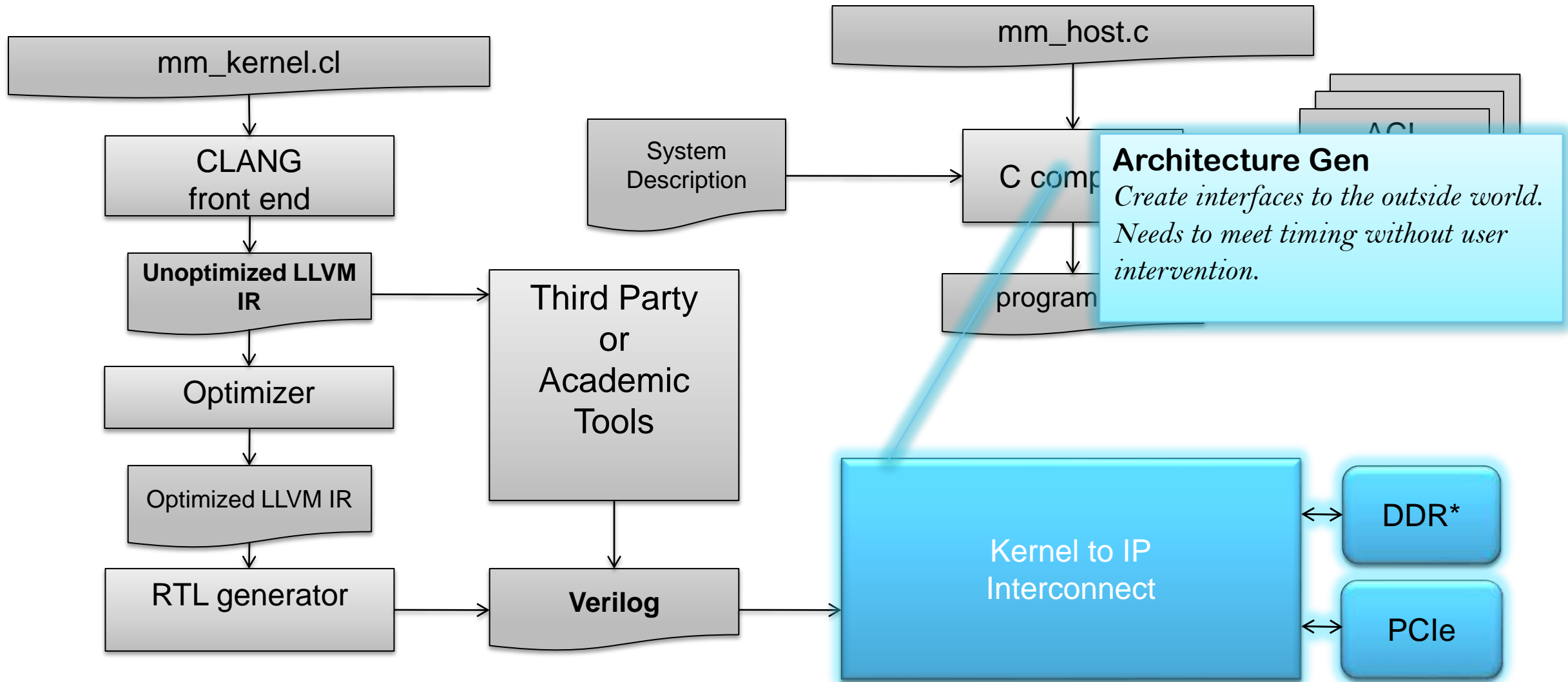
# OpenCL CAD Flow



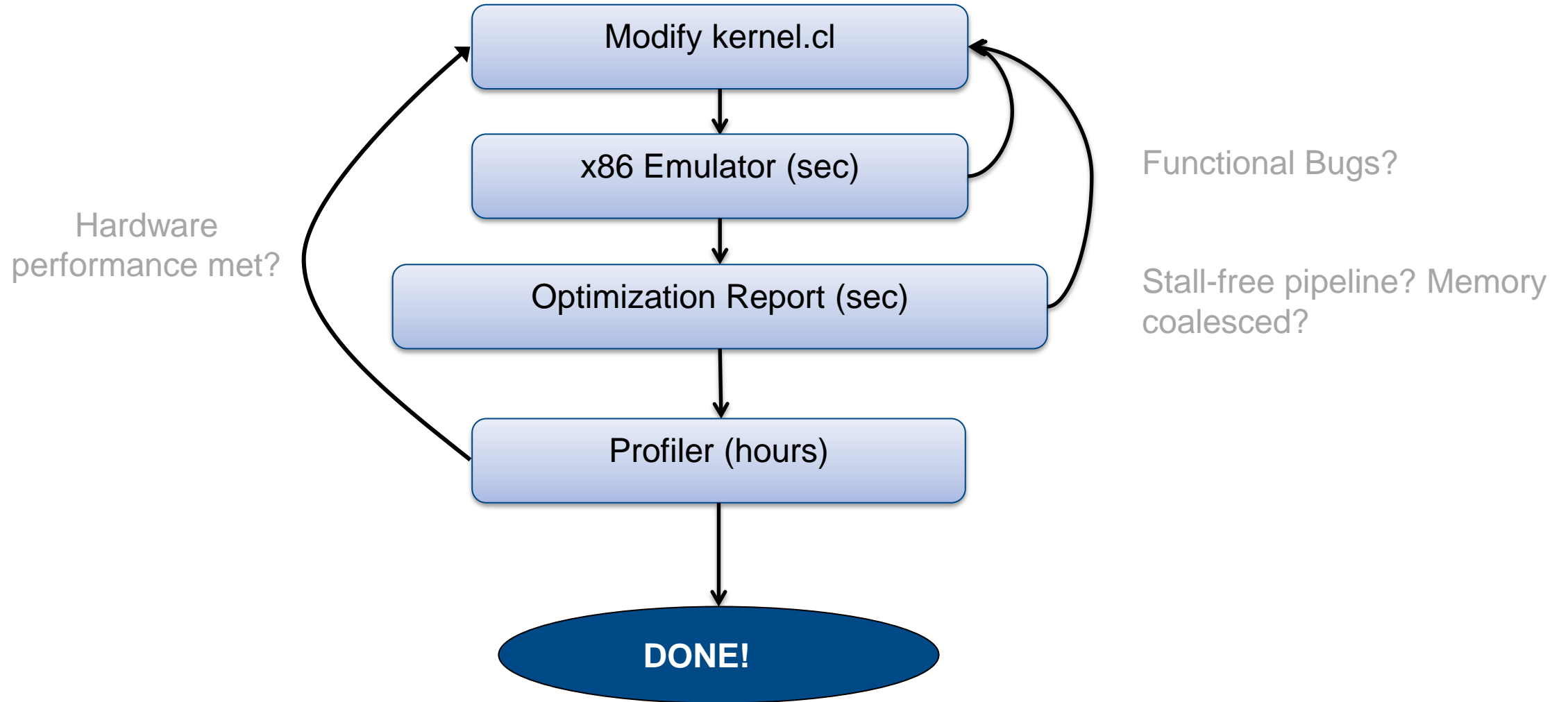
# OpenCL CAD Flow



# OpenCL CAD Flow



# OpenCL Kernel Development Flow



# x86 emulator

- Enable functional debug on x86 system of kernel code
  - Prototype support to allow users run kernels on x86 platform
  - Debug support for Altera vendor specific debug support such as channels

```
kernel void accel(...) {  
    ...  
    gid = get_global_id(0);  
    out[gid] =  
        proc(data[gid]);  
    ...  
}
```



## Supports

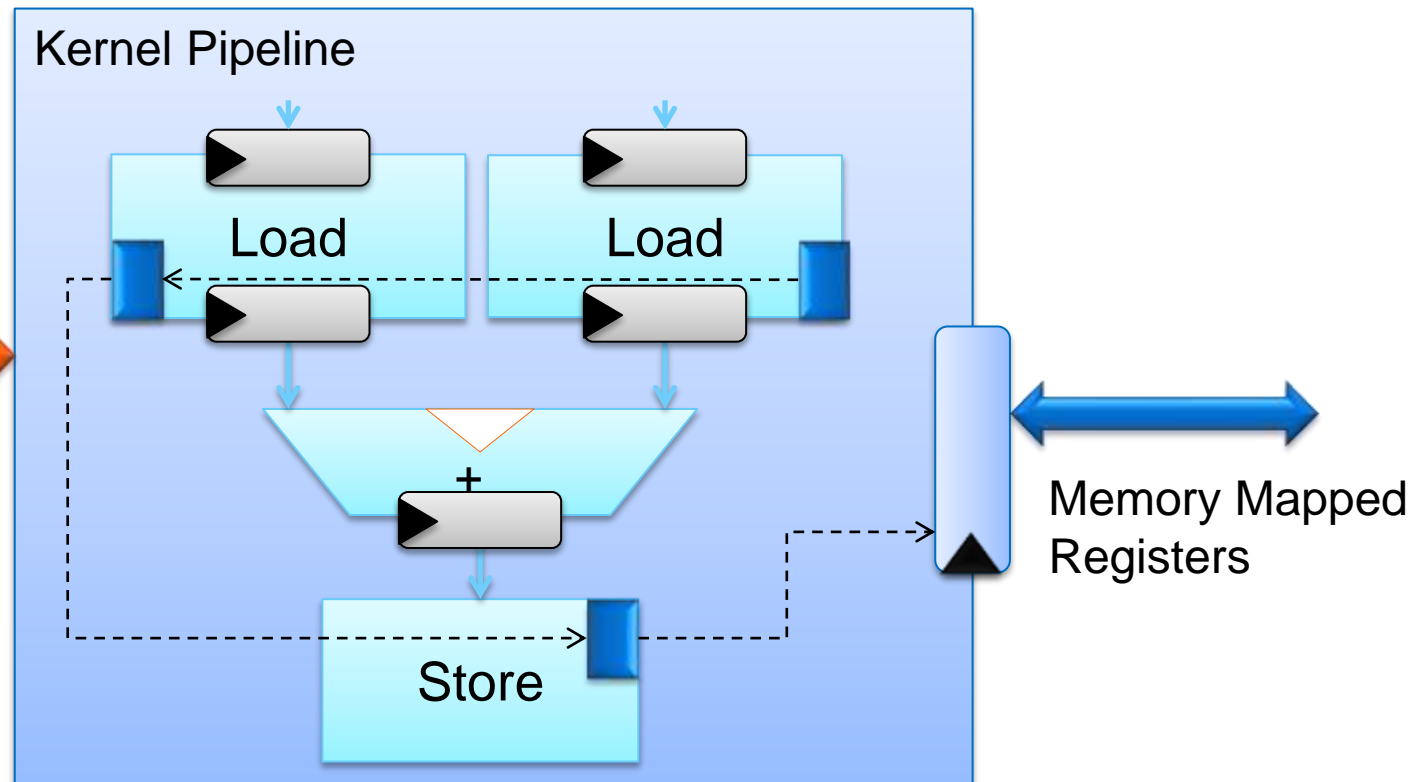
- OpenCL syntax
- Channels
- Printf



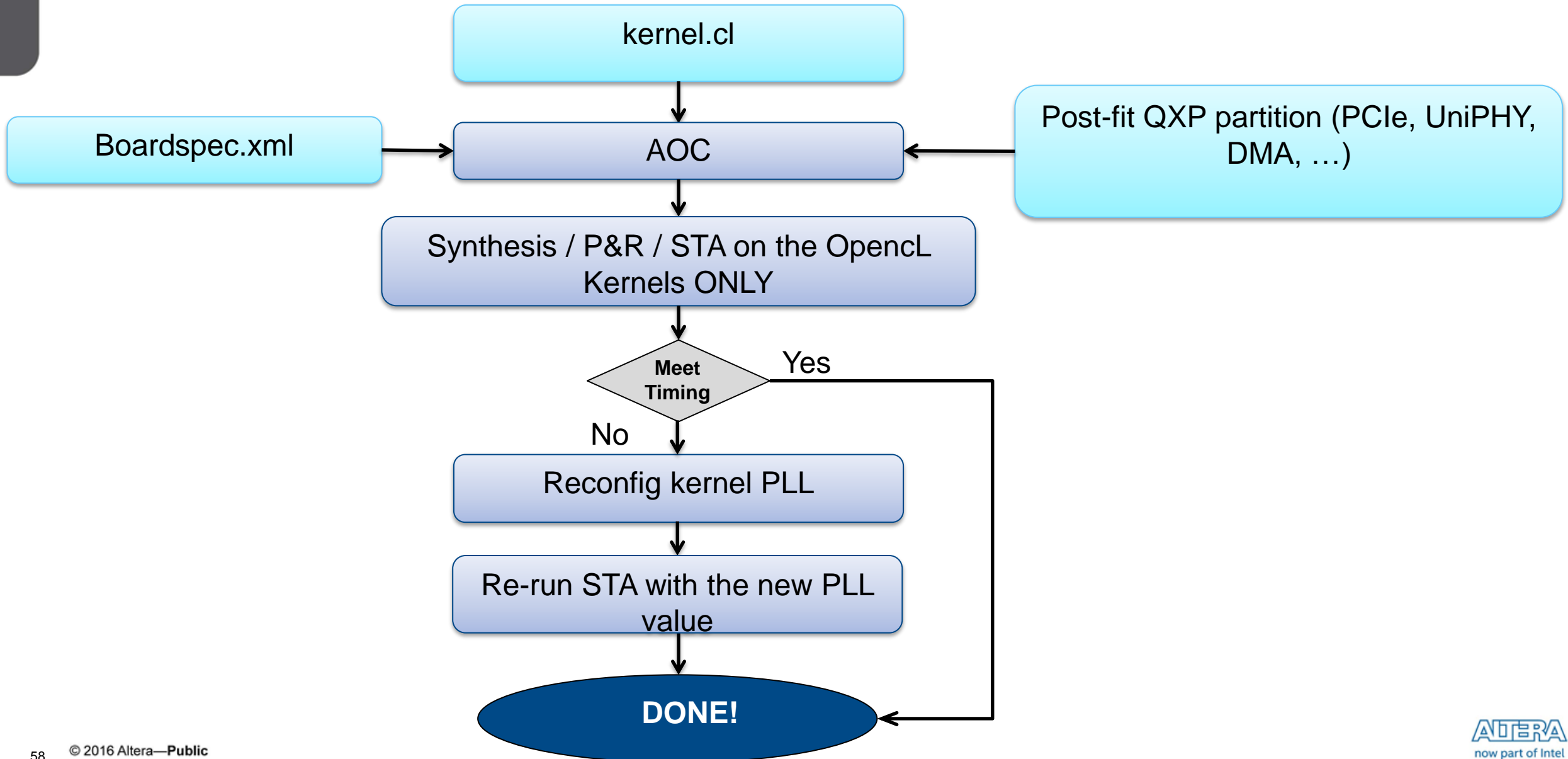
# Profiler

- Instrument the pipeline with performance counters and profiling logic
- Transfer the profiling information to the host via PCIe link

```
kernel void accel(...) {  
    ...  
    gid = get_global_id(0);  
    out[gid] = a[gid]+b[gid];  
    ...  
}
```



# Guaranteed Timing Flow



# Optimization Report Example: Load to Store dependency

```
1 kernel void prefixsum( global int* restrict A, unsigned N ) {  
2   for ( unsigned i = 1 ; i < N ; i++ ) {  
3     int a = A[i-1];  
4     A[i] += a;  
5   }  
6 }
```

```
=====  
|                               *** Optimization Report ***                               |  
=====  
| Kernel: prefixsum
```

Relative cost of global memory  
to local computation

```
| Loop for.body
```

```
| 2.25
```

```
|   Pipelined execution inferred.
```

```
|   Successive iterations launched every 321 cycles due to:
```

```
|     Memory dependency on Load Operation from:
```

```
| 3.21
```

```
|       Store Operation
```

```
|     Largest Critical Path Contributors:
```

```
|       49%: Load Operation
```

```
|       49%: Store Operation
```

True fix requires restructuring  
the code

```
| 4.7
```

# Optimization Report Example: Accumulating a value

```
1 kernel void test( global float* restrict input,  
2                  global float* restrict output, unsigned N )  
3 {  
4     float mul = 1.0f;  
5     for ( unsigned i = 0; i < N; i++ ) {  
6         mul *= input[ i ];  
7     }  
8     *output = mul;  
9 }
```

```
=====  
| *** Optimization Report *** |  
=====  
| Kernel: test | Ln.Col |  
=====  
| Loop for.body | 5.24 | | |
| Pipelined execution inferred. | |  
| Successive iterations launched every 3 cycles due to: | |  
| | | | |  
| Data dependency on variable mul | 4.10 | |  
| Largest Critical Path Contributor: | | |  
| 100%: Fmul Operation | 6.7 | |  
=====
```

# Architecture Visualizer (Hidden)

Hierarchical  
and interactive

The screenshot shows the Architecture Visualizer (vis) interface. On the left, a code editor displays the source code for `node-details.cl`. The code includes a kernel named `meminst` and another kernel named `coalesced`. The `meminst` kernel contains a loop that iterates over an array `A` of size `N` (defined as 256) and updates its elements. The `coalesced` kernel is also shown, with its function signature and a loop body. On the right, the 'Stall Points Graph' tab is active, showing a hierarchical view of the `meminst` kernel. It consists of three basic blocks: 'Basic Block 0', 'Basic Block 1', and 'Basic Block 2'. 'Basic Block 1' is highlighted in grey and has a blue arrow pointing to a 'Global Memory' block, indicating a stall point. The 'Area Report' and 'Details' tabs are also visible but currently empty.

# Detailed Area Report (aocl analyze-area)

## Per-line area break-down.

- Very useful, as single careless line of code can burn many FPGA resources.

```
fft2d.temp.cl  fft_8.cl  twld_radix4_8.cl
128  a1210 >>= (LOGN / 2);
129  a75 <<= (LOGN / 2);
130  where = (x & ~mask) | a1210 | a75;
131  where_global = mangle_bits(where);
132  } else {
133  where = x;
134  where_global = where;
135  }
136
137  uint buf_base_addr = where & ((1 <<
(LOGN + LOGPOINTS)) - 1);
138  buf[buf_base_addr] = src[where_global];
139  buf[buf_base_addr + 1] = src[where_glob
al + 1];
140  buf[buf_base_addr + 2] = src[where_glob
al + 2];
141  buf[buf_base_addr + 3] = src[where_glob
al + 3];
142  buf[buf_base_addr + 4] = src[where_glob
al + 4];
143  buf[buf_base_addr + 5] = src[where_glob
al + 5];
144  buf[buf_base_addr + 6] = src[where_glob
al + 6];
145  buf[buf_base_addr + 7] = src[where_glob
al + 7];
146
147  barrier(CLK_LOCAL_MEM_FENCE);
148
149  int row = get_local_id(0) >> (LOGN -
LOGPOINTS);
150  int col = get_local_id(0) & (N / POINTS
- 1);
151
```

Stall Points Graph Area Report

entry	Resources			
	LEs	FFs	RAMs	DSPs
Operations				
fft2d.temp.cl:114	0	0	0	0
fft2d.temp.cl:128	0	0	0	0
fft2d.temp.cl:129	0	0	0	0
fft2d.temp.cl:130	0	0	0	0
fft2d.temp.cl:137	46	123	1	0
fft2d.temp.cl:138	507	2082	13	0
fft2d.temp.cl:147	110	73	0	0
fft2d.temp.cl:150	0	0	0	0
fft2d.temp.cl:154	20	29	0	0
fft2d.temp.cl:155	19	29	0	0

# Additional Altera OpenCL Collateral

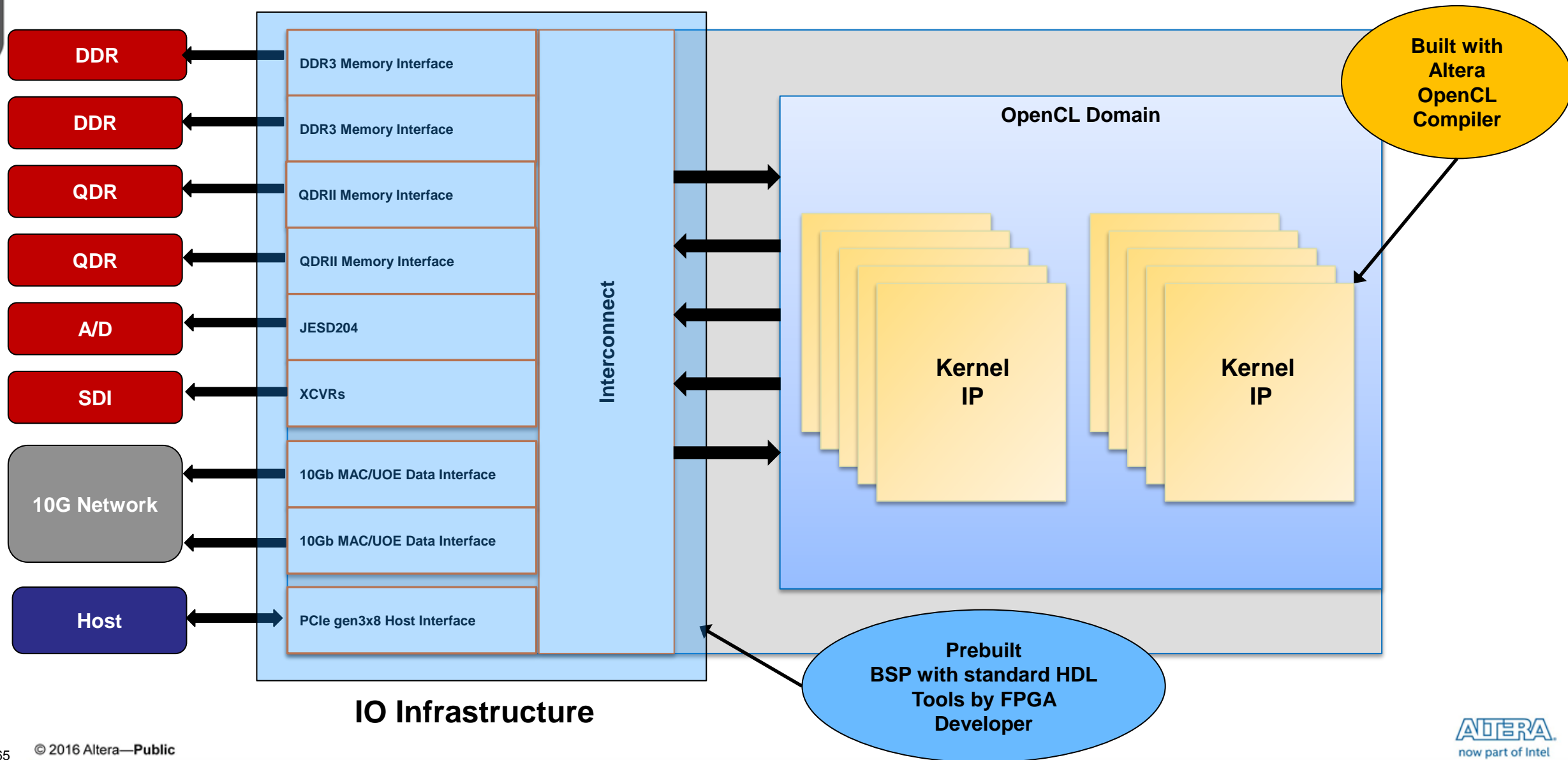
- ◀ [White papers on OpenCL](#)
- ◀ [OpenCL online demos](#)
- ◀ [OpenCL design examples](#)
- ◀ [Instructor-Led training](#)
  - [Parallel Computing with OpenCL Workshop by Altera – \(1 Day\)](#)
  - [Optimization of OpenCL for Altera FPGAs Training by Altera – \(1 Day\)](#)
- ◀ [Online training](#)
  - [Introduction to Parallel Computing with OpenCL](#)
  - [Writing OpenCL Programs for Altera FPGAs](#)
  - [Running OpenCL on Altera FPGAs](#)
  - [Single-Threaded vs. Multi-Threaded Kernels](#)
  - [Building Custom Platforms for Altera SDK for OpenCL](#)
- ◀ [OpenCL board partners page](#)

# ALTERA BSP: abstracting FPGA development



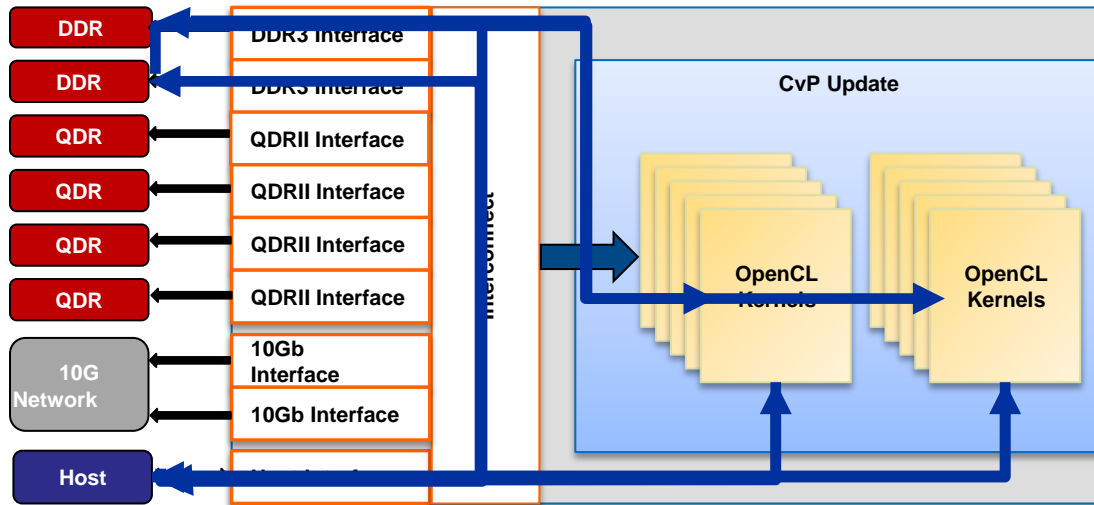


# An adaptable Board Support Package

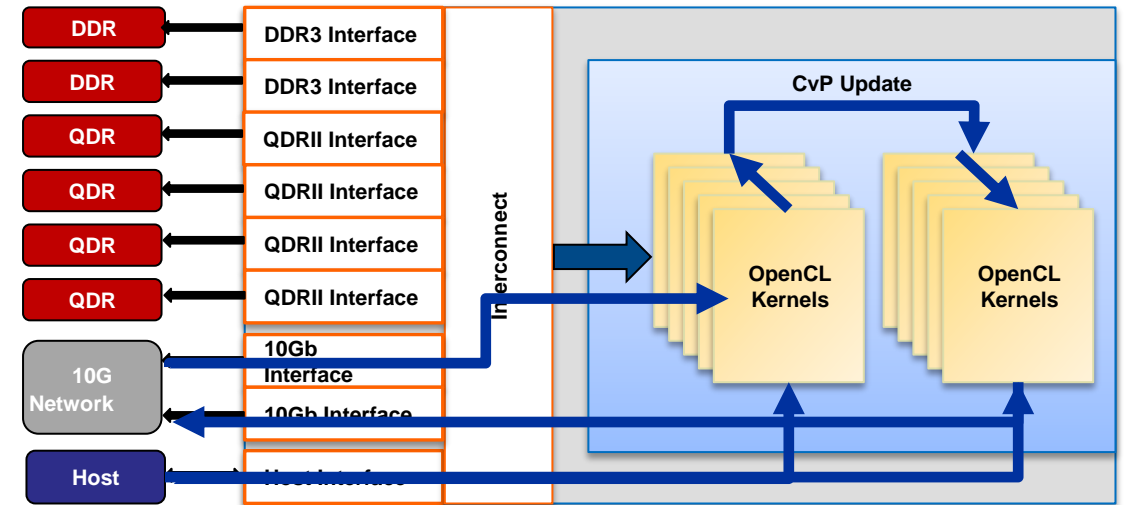


# Channels Advantage

## Standard OpenCL



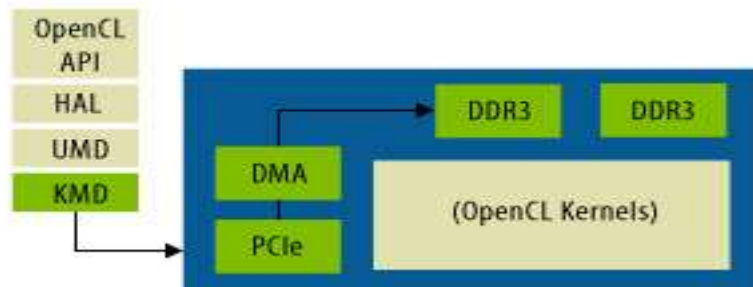
## Altera Vendor Extension IO and Kernel Channels



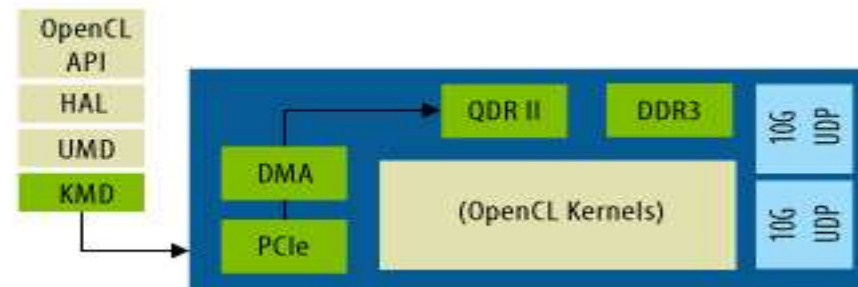
```
channel int DataChannel;  
  
kernel producer(...) {  
    write_channel_altera(DataChannel, value);  
}  
  
kernel consumer(...) {  
    value = read_channel_altera(DataChannel);  
}
```

# Start with OpenCL ready platforms 1/2

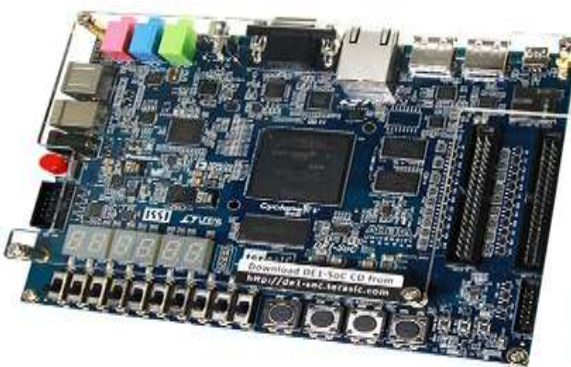
## HPC Applications



## Network Applications



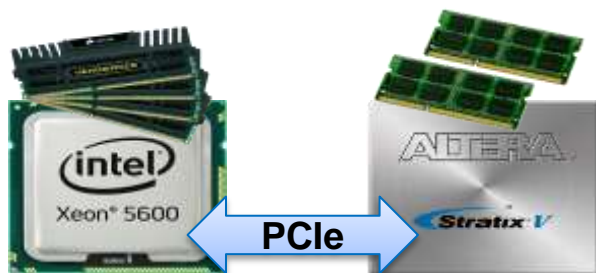
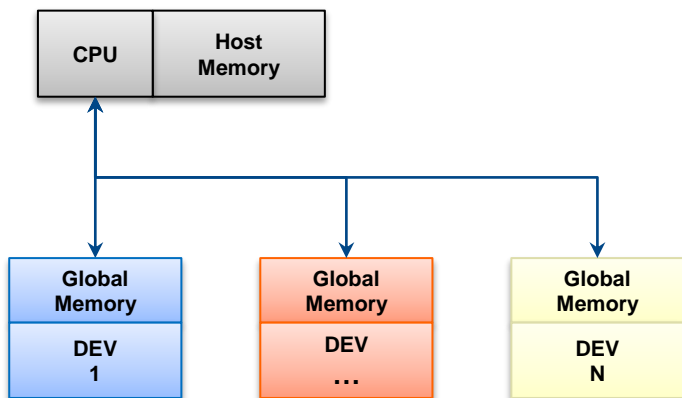
# Start with OpenCL ready platforms 2/2



# Shared Virtual Memory (SVM) Platform Model

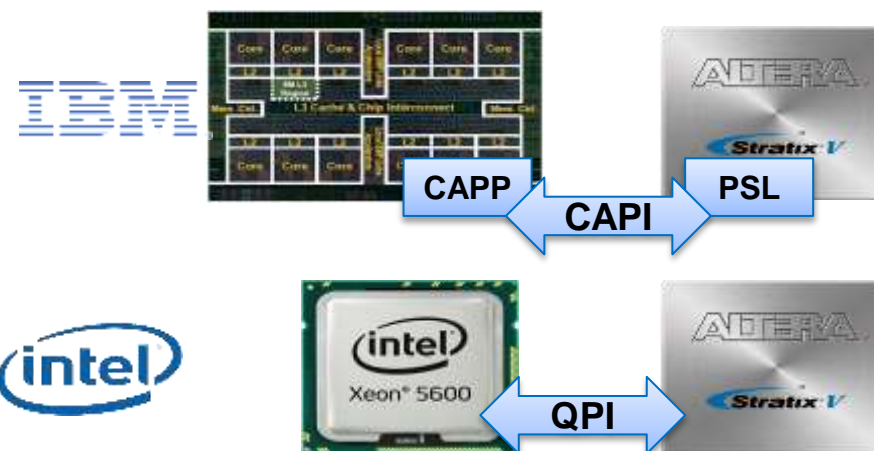
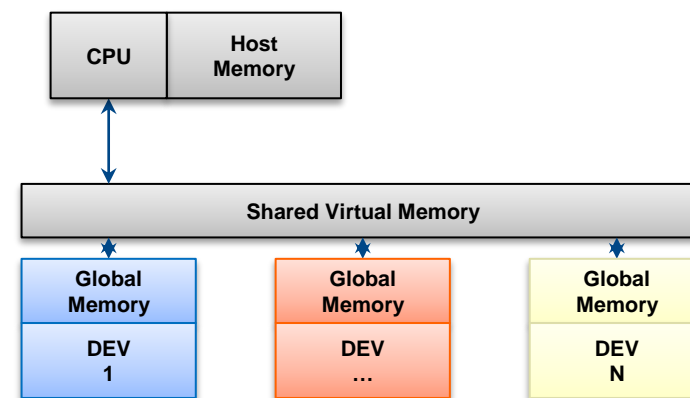
## OpenCL 1.2

- Traditional Hosted Heterogeneous Platform

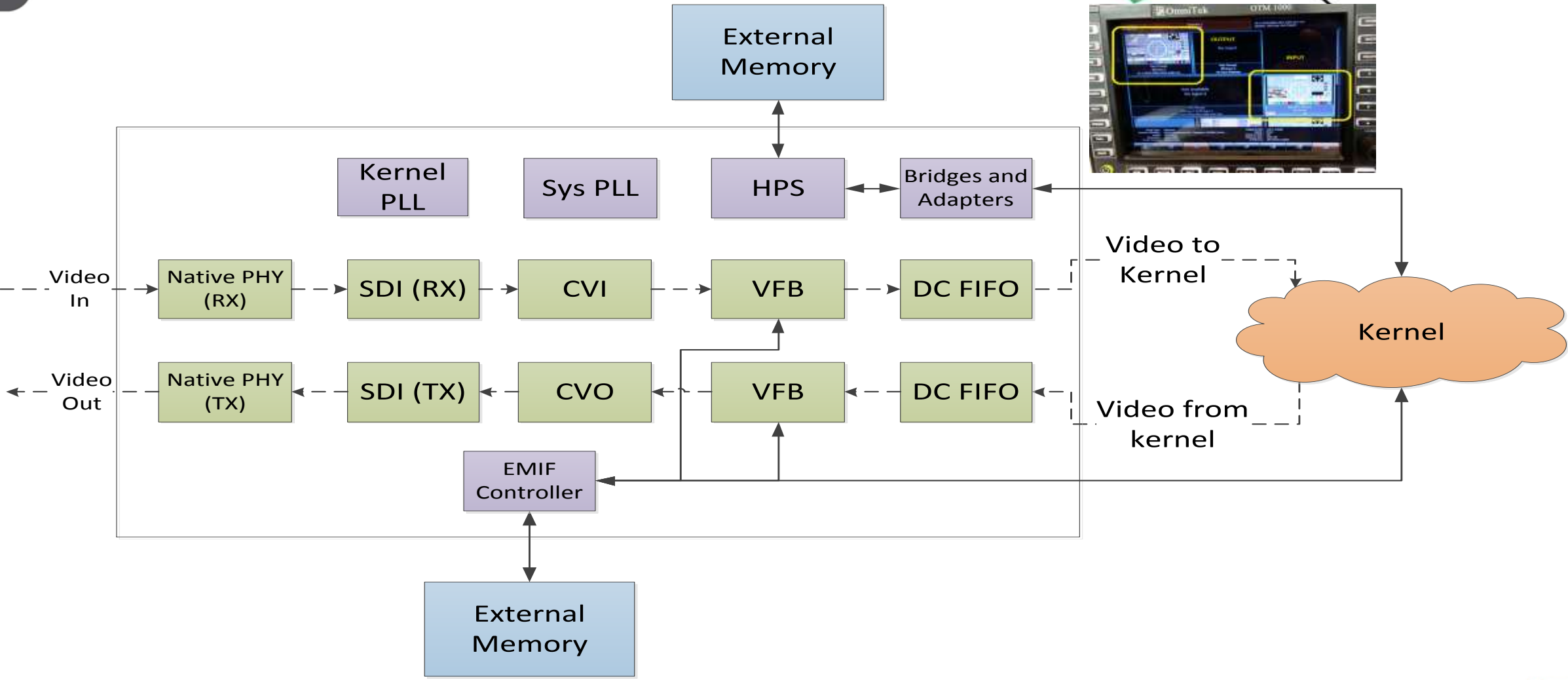
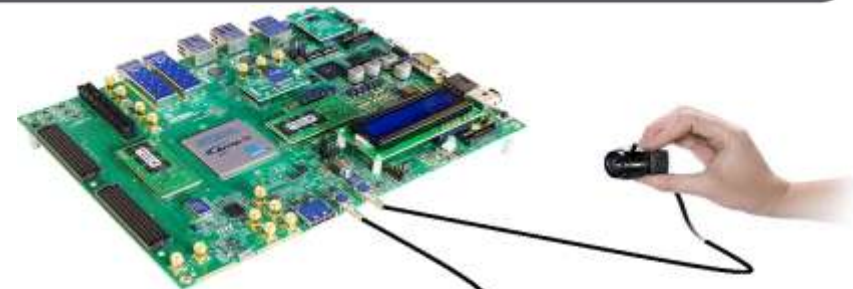


## OpenCL 2.0

- New Hosted Heterogeneous Platform with SVM



# VIP based BSP Customization





# Live Demo



**ALTERA**  
now part of Intel

# Developing a Custom OpenCL BSP

Deep Dive





# Recommended Hardware

## Development system

- Available PCIe slot (if using PCIe-based accelerator card)
- x86 based development system
- Altera device documentation defines minimum recommended system RAM

## FPGA accelerator card

- PCIe interface or SoC
- DDR3 or DDR4 External Memory
- Embedded USB blaster or JTAG header

# Software Requirements

## ◀ Operating system: 64-bit<sup>1</sup>

- Microsoft 64-bit Windows 7 on the x86-64 architecture
- Red Hat Enterprise 64-bit Linux (RHEL) 6.0 on the x86-64 architecture

## ◀ Quartus Prime

- Accelerator devices installed
- Quartus Prime license

## ◀ Altera SDK for OpenCL

- Must match Quartus Prime version
- Altera SDK for OpenCL License

## ◀ C compiler for host code

- E.g. Microsoft® Visual Studio or GCC
- Needed to compile the host program
- Able to compile and link **64-bit** code
  - ◀ Except when targeting a SoC host

1. CentOS, Ubuntu and Windows 8 supported in a future version of the Quartus software

# SDK Components

## ◀ AOCL Utility

- Perform various tasks related to the board, drivers, and compile process

## ◀ Altera Offline Compiler (AOC)

- Translates your OpenCL C kernel source file into an FPGA hardware image ready to be loaded onto the Altera FPGA

## ◀ Host Libraries

- Provides the OpenCL host Platform API and Runtime API to be used by OpenCL host applications
- Libraries for the host program to link to

## Altera OpenCL (AOCL) Utility

- ◀ Custom Platforms must support a set of **aocl** utilities
  - Executables delivered in a subdirectory within the Custom Platform files
- ◀ AOCL looks for corresponding executables when respective **aocl** calls are made
- ◀ **install** (`aocl install`)
  - Installs driver into the host operating system
- ◀ **uninstall** (`aocl install`)
  - Removes driver from the host operating system
- ◀ **program** (`aocl program <device> <kernel file>.aocx`)
  - Programs the FPGA using the provided aocx file
- ◀ **flash** (`aocl flash <device> <kernel_file>.aocx`)
  - Programs base programming image into Flash
- ◀ **diagnose** (`aocl diagnose [<device_name>]`)
  - Confirms board functionality

## aoc Output Files

### ◀ <kernel file>.aoco

- Intermediate object file representing the created hardware system

### ◀ <kernel file>.aocx

- Kernel executable file used to program FPGA

### ◀ <kernel file> folder

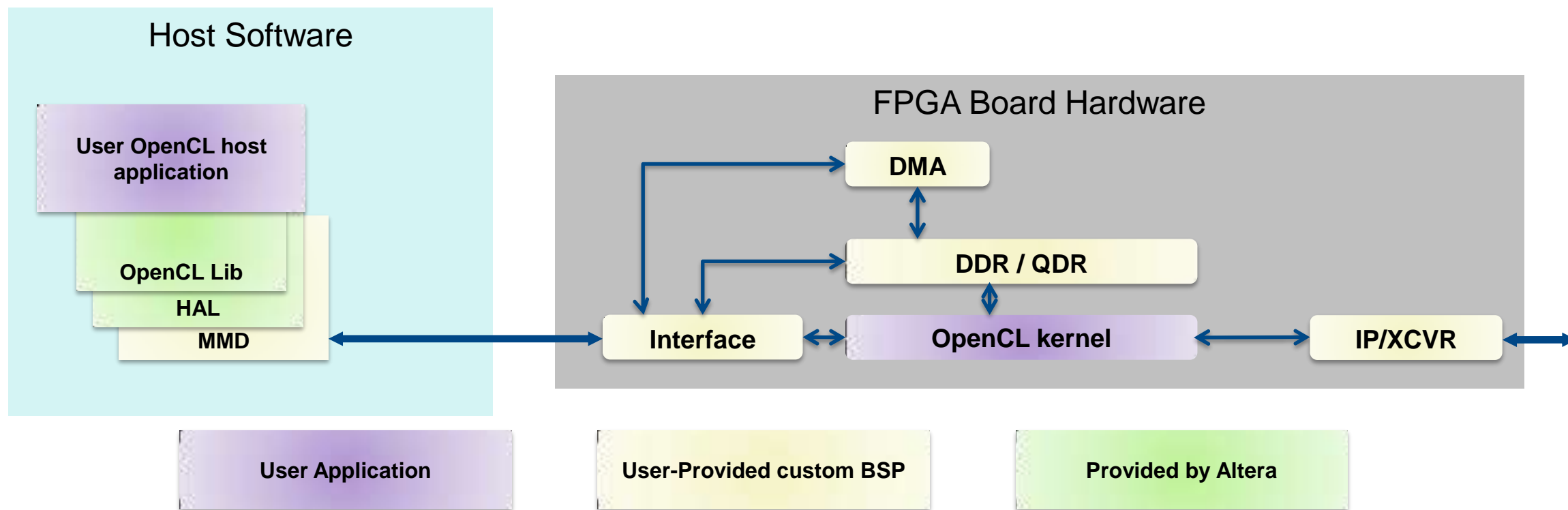
- <kernel file>.log
  - ◀ Main compile log including estimated resource usage, optimization report, and compile messages
- Quartus project
  - ◀ Project files
  - ◀ Source files
  - ◀ Timing reports
  - ◀ quartus\_sh\_compile.log
    - Output from the Quartus software compile
    - Useful for error checking

## Compiling the Host Program

- ◀ Use a conventional C compiler (Visual Studio/GCC)
- ◀ Add `%ALTERAOCLSDKROOT%/host/include` to your file search path
  - Recommended to use `aocl compile-config`
- ◀ Include `CL/opencl.h` in your source code
- ◀ Link to Altera OpenCL libraries
  - Link to libraries located in the `%ALTERAOCLSDKROOT%/host/<OS>/lib` directory
    - ◀ Recommended to use `aocl link-config`

# Custom Platforms

- Framework of host software and FPGA interface design to enable the use of OpenCL on a custom board



## Custom vs. Preferred Platform

Description	Custom	Preferred
Flexibility	High	Low
Supports Custom Boards	Yes	No
Supports Custom Interfaces	Yes	No
Design skills required	Many	Fewer
HDL coding skills required	Yes	No
High-speed interface skills required	Yes	No
Software coding skills required	More	Some
Development Time	Higher	Lower
Qsys system design effort required	Yes	No
Floorplanning effort required	Yes	No



# Custom Platform BSP Overview

## ◀ Goals

- Allow Altera® SDK for OpenCL™ to automatically create FPGA images from OpenCL kernel C code for custom boards
- Allow the compilation of OpenCL host code to easily run kernels on the FPGA board

## ◀ Tools

- Custom Platform Toolkit
- Use one of the reference platforms as a starting point
  - ◀ Network Reference Platform
  - ◀ High Performance Computing (HPC) Reference Platform
  - ◀ FPGA design, software, and board bring up skills required

## ◀ Not required if using an Altera Preferred Board for OpenCL

- Download BSP from the board vendor

# Reference Platforms

- ◀ Stratix V Network Reference Platform
  - [User Guide](#)
  - Download from OpenCL board [platforms landing page](#)
- ◀ Stratix V Reference Platform
  - Ships with the Altera SDK for OpenCL
- ◀ Cyclone V SoC Reference Platform
  - [User Guide](#)
  - Ships with the Altera SDK for OpenCL
- ◀ Template project - Custom BSP toolkit deliverable
  - Skeleton design
- ◀ Arria 10 GX Reference Platform
  - *Contact your Altera representative for the latest version*

# Custom Platform Development Support

## ◀ Custom Platform developer page

- <https://www.altera.com/products/design-software/embedded-software-developers/openc1/developer-zone.html#Custom>

## ◀ Custom Platform Toolkit

- <https://www.altera.com/products/design-software/embedded-software-developers/openc1/developer-zone.html#Custom>

## ◀ Custom Platform Toolkit User Guide

- [https://www.altera.com/content/dam/altera-www/global/en\\_US/pdfs/literature/hb/openc1-sdk/ug\\_aocl\\_custom\\_platform\\_toolkit.pdf](https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/openc1-sdk/ug_aocl_custom_platform_toolkit.pdf)

# Developing a Custom OpenCL BSP

## Important New Quartus Software Features<sup>1</sup>

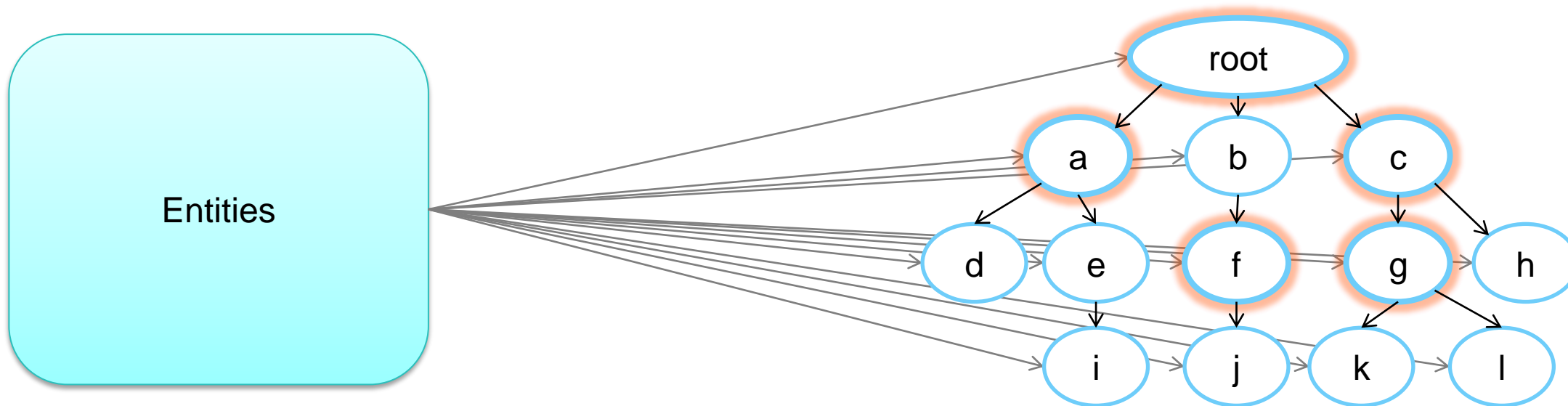
1. These features apply to Quartus Prime Pro which only supports Arria 10 devices and newer



# Partitions and blocks

- ◀ Partitions/blocks and like instances/entities
  - Except that most of the time, they're one-to-one
  - In PR, one partition can have multiple blocks
  
- ◀ In fact, a partition is *always* an instance
  - The partition name is the instance name
  - A partition is simply an instance that cannot be dissolved
  
- ◀ A block is the implementation of a partition
  - Local assignments
  - A netlist
  - Placement and routing information

# Setting Partitions<sup>1</sup>

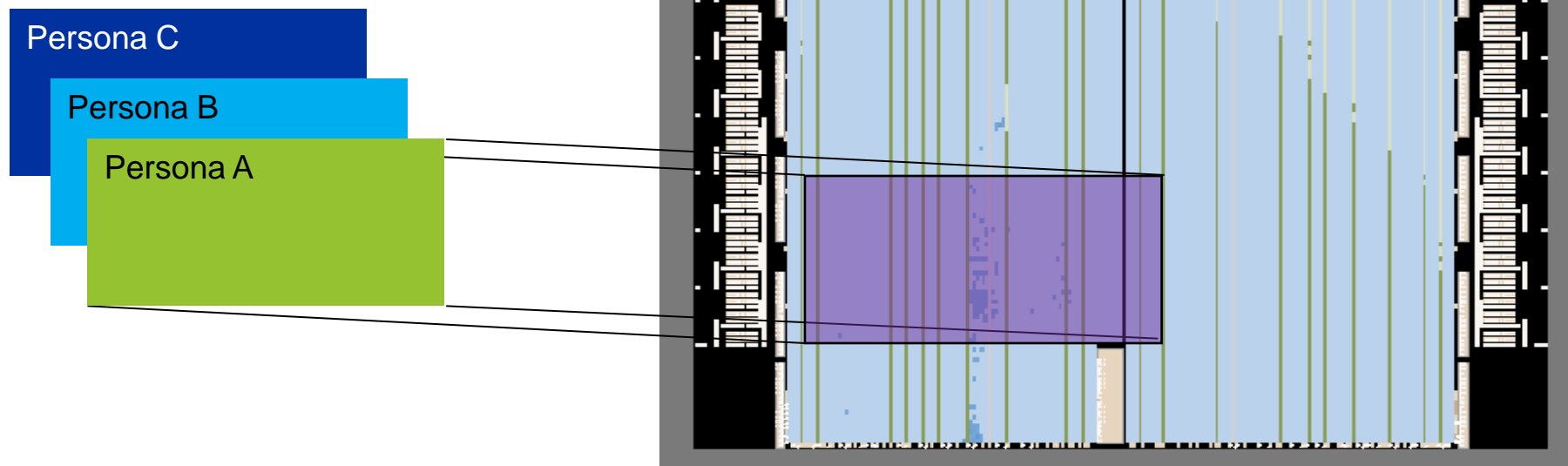


```
set_instance_assignment -name PARTITION a_block -to a
set_instance_assignment -name PARTITION c_block -to c
set_instance_assignment -name PARTITION bf_block -to b|f
set_instance_assignment -name PARTITION cg_block -to c|g
```

1. Partitions must be set in the QSF file at this time. Partitions will be supported in the GUI in a future version of Quartus software.

# Partial Reconfiguration (PR)

- ◀ The ability to reconfigure (reprogram) part of the device, while the rest of the device is running
- ◀ Used by the OpenCL Runtime to program kernels without disturbing the periphery
- ◀ PR for Arria 10 devices is Early Access<sup>1</sup> only in 15.1



1. PR for OpenCL as shown is not available for previous devices. Previous devices used Configuration via Protocol (CvP) for OpenCL.

# Commonly Used PR Terms

## ◀ Static region

- Remains constant across all PR personas
- Part(s) of the design not changed by PR
- Essentially the BSP

## ◀ PR partition

- A design partition targeted for PR

## ◀ PR region

- A physical location assigned to a PR partition
- Contains the kernels generated by the aoc compiler

## ◀ Persona

- One of the variations in functionality that a PR region can take
- A PR region may have more than 1 persona

## ◀ Freeze Wrapper

- discussed later



# Developing a Custom OpenCL BSP

Hardware Development



## Hardware Procedure - Setup Environment

1. Copy the Arria 10 GX Reference Platform
2. Rename the directories of the platform
3. Modify the XML files
4. Modify the environment variables
5. Conduct a base compile using boardtest.cl
6. Verify timing
7. Copy the base\_qhd.qar file to the custom BSP directory<sup>1</sup>
8. Conduct an import compile with a simple kernel
9. Verify error free compile

## Modify board\_env.xml file<sup>1</sup>

- Modify the `<your_custom_platform>/board_env.xml` file to match the names of your platform and board directories

```
<?xml version="1.0"?>
<board_env version="15.1" name="custom_platform">
  <hardware dir="hardware" default="my_board"></hardware>
  <platform name="linux64">
    <mmdlib>%b/linux64/lib/libaltera_a10_ref_mmd.so</mmdlib>
    <linkflags>-L%b/linux64/lib</linkflags>
    <linklibs>-laltera_a10_ref_mmd</linklibs>
    <utilbindir>%b/linux64/libexec</utilbindir>
  </platform>

  <platform name="windows64">
    <mmdlib>%b/windows64/bin/altera_a10_ref_mmd.dll</mmdlib>
    <linkflags>/libpath:%b/windows64/lib</linkflags>
    <linklibs>altera_a10_ref_mmd.lib</linklibs>
    <utilbindir>%b/windows64/libexec</utilbindir>
  </platform>
</board_env>
```

1. The board\_env.xml file will be explained in more detail in a later section.  
%b references your board installation directory

## Modify board\_spec.xml file<sup>1</sup>

- ◀ Modify the **<your\_custom\_platform>/hardware/<board variant>/board\_spec.xml** file to match the name of your board directory

```
<?xml version="1.0"?>
<board version="15.1" name="my_board">

  <compile project="top" revision="top" qsys_file="system.qsys" generic_kernel="1">
    <generate cmd="echo"/>
    <synthesize cmd="quartus_sh -t import_compile.tcl"/>
    <auto_migrate platform_type="a10_ref" >
      <include fixes=""/>
      <exclude fixes=""/>
    </auto_migrate>
  </compile>

  .
  .
  .
```

1. The board\_spec.xml file will be discussed in more detail in a later section.

# Features of the Arria 10 Reference Platform

## OpenCL Host

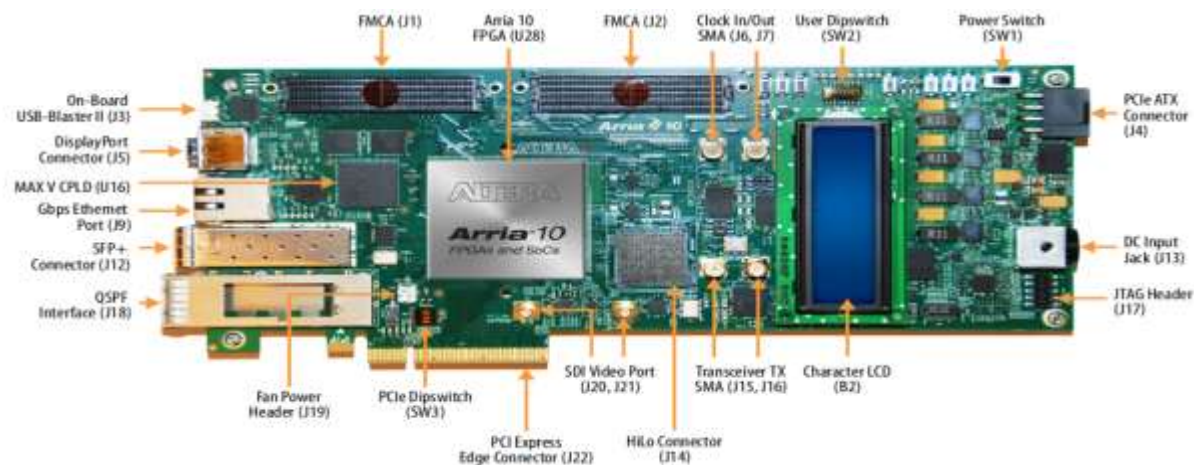
- PCIe-based host that connects to the Arria 10 PCIe Gen3 x8 Hard IP core

## OpenCL Global Memory

- One 2-gigabyte (GB) DDR4 SDRAM daughter card

## FPGA Programming via one of the following methods:

- Partial Reconfiguration (PR) over PCIe
- External cable and the Arria 10 GX FPGA Development Kit's on-board USB-Blaster® II interface
- On-board FLASH



# Contents of the Arria 10 Reference Platform

## ◀ \hardware

- Contains the Quartus Prime project templates for three board variants
- Each board variant implements the entire OpenCL hardware system on a given kit

## ◀ \windows64 /linux64

- Contains the MMD library, kernel mode driver, and executable files of the AOCL utilities (that is, install, uninstall, flash, program, diagnose) for the OS

## ◀ \source\_windows64

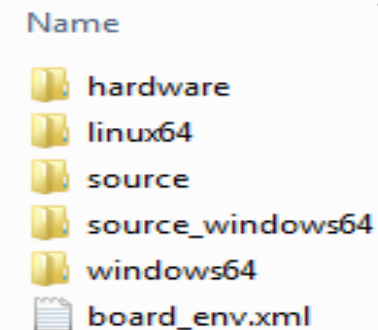
- Contains source codes for the MMD library and AOCL utilities
- The MMD library and the AOCL utilities are in the windows64 folder

## ◀ /source

- Contains source codes for the MMD library and AOCL utilities
- The MMD library and the AOCL utilities are in the linux64 directory

## ◀ board\_env.xml

- eXtensible Markup Language (XML) file that describes the Reference Platform to the Altera SDK for OpenCL

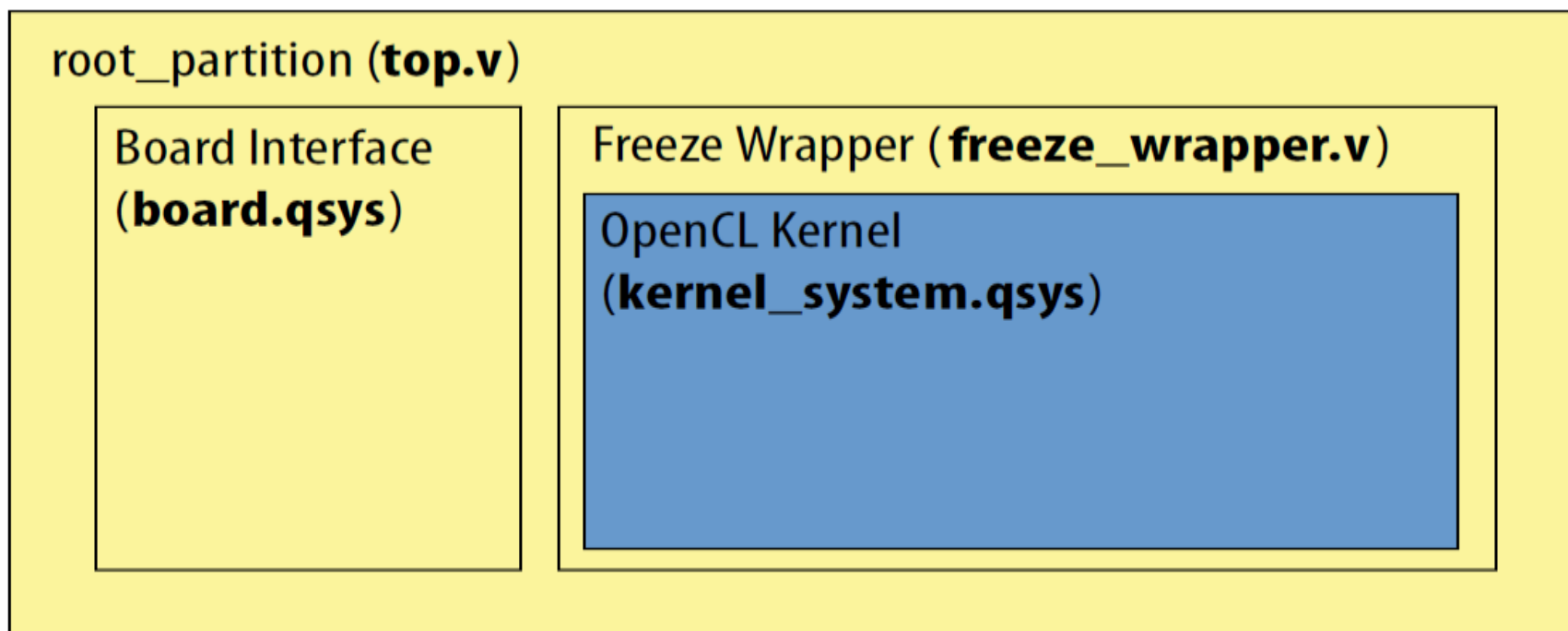


# Contents of Each Board Variant Directory

Option	Description
<b>quartus.ini</b>	Contains any special Quartus Prime software options that you need to compile OpenCL kernels for the Reference Platform.
<b>system.qsys</b>	Legacy file that you must update with interfaces, to match those defined in the <b>board spec.xml</b> file, for the compilation flow to work properly. The compilation process does not include the <b>system.qsys</b> file into the OpenCL hardware system.
<b>board.qsys</b>	Qsys system that implements the board interfaces (that is, the static region) of the OpenCL hardware system.
<b>top.qpf</b>	Quartus Prime Project File for the OpenCL hardware system.
<b>top.qsf</b>	Quartus Prime Settings File for the AOCL-user compilation flow.
<b>top.sdc</b>	Synopsys Design Constraints File that contains board-specific timing constraints.
<b>top.v</b>	Top-level Verilog Design File for the OpenCL hardware system.
<b>top_post.sdc</b>	Qsys and AOCL IP-specific timing constraints.
<b>top_synth.qsf</b>	Quartus Prime Settings File for the Quartus Prime revision in which the OpenCL kernel system is synthesized.
<b>base.qsf</b>	Quartus Prime Settings File for the base project revision. Use this revision when porting the Arria 10 Reference Platform to your own custom BSP. The Quartus Prime Pro Edition software compiles this base project revision from source code.

**Do not try to compile the BSP project in the Quartus Prime software!**

# Hardware System Overview





# Altera SDK for OpenCL-Specific Qsys Components

## ◀ Required

- OpenCL Clock Generator
- OpenCL Kernel Interface
- OpenCL Bank Divider

## ◀ Altera Interface IP

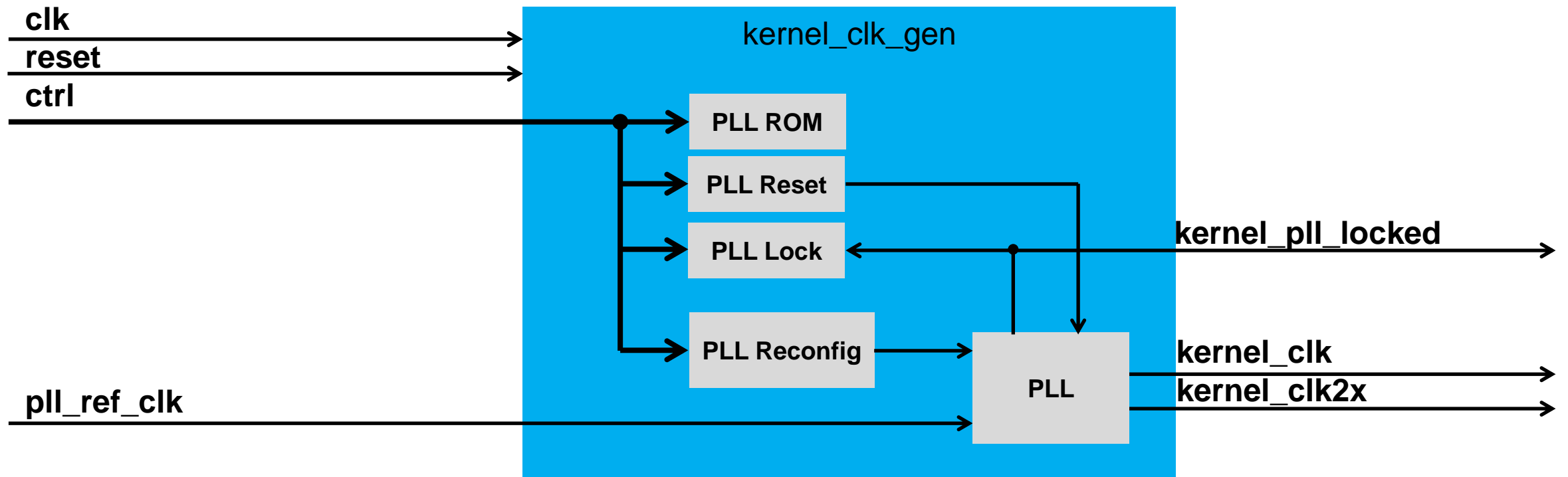
- PCI Express Hard IP
- DDR Controller
- QDR Controller

## ◀ Altera Supporting IP

- Avalon-MM Pipeline Bridge
- Scatter Gather DMA
- Uniphy Status Component
- ACL Version ID
- Reset Components

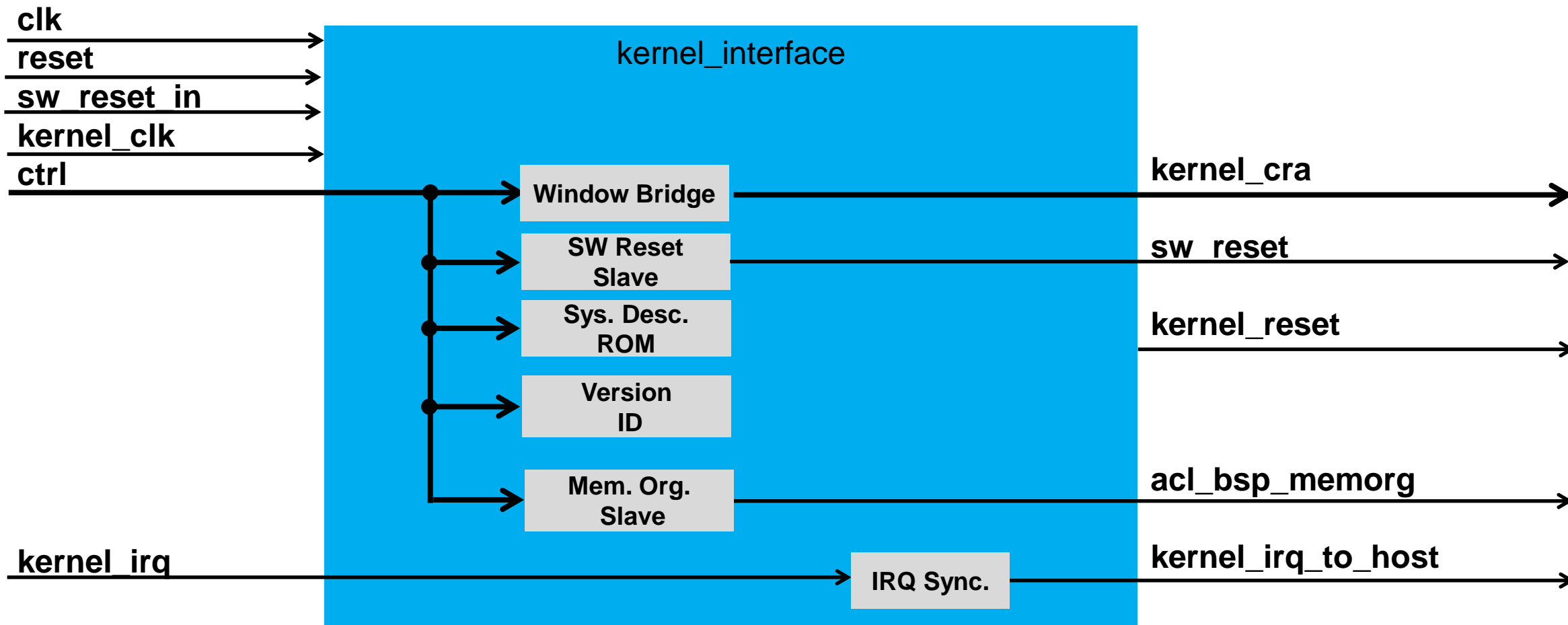
# OpenCL Clock Generator

- Programmable PLL to adjust kernel clock rate
- Status interfaces allow software to observe the PLL



# OpenCL Kernel Interface

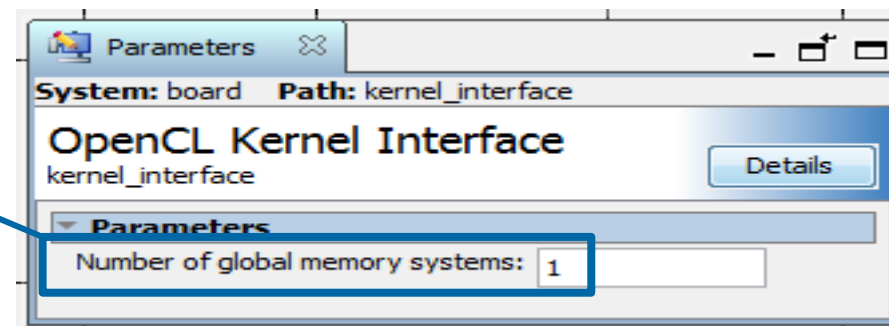
Allows the host to control the kernel compute units



# OpenCL Kernel Interface

◀ Interface added for each global memory system

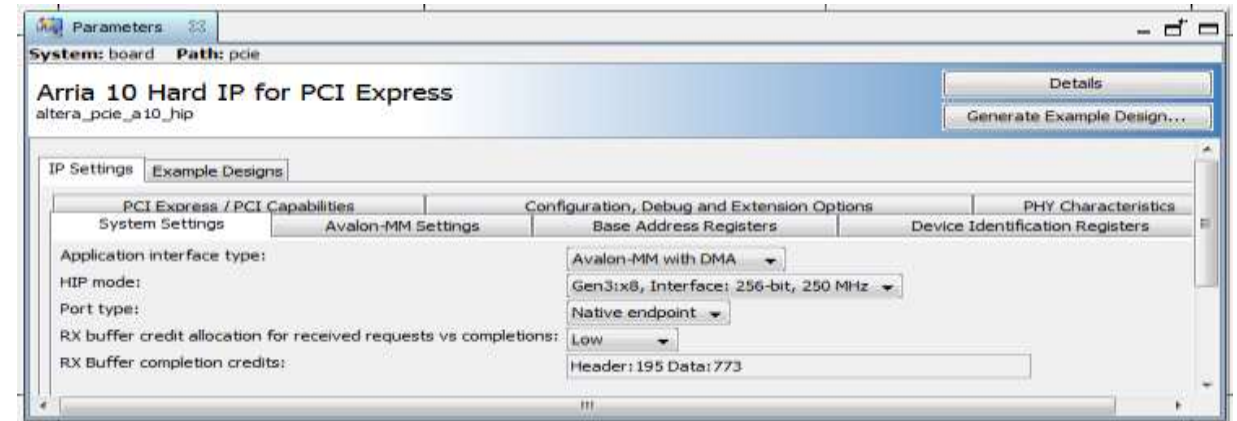
Name	Description	Export	Clock
<b>kernel_interface</b>	OpenCL Kernel Interface		
clk	Clock Input	<i>Double-click to export</i>	<b>pcie_corec...</b>
reset	Reset Input	<i>Double-click to export</i>	[clk]
ctrl	Avalon Memory Mapped Slave	<i>Double-click to export</i>	[clk]
kernel_clk	Clock Input	<i>Double-click to export</i>	<b>kernel_clk...</b>
kernel_cra	Avalon Memory Mapped Master	<b>kernel_cra</b>	[kernel_clk]
sw_reset_in	Reset Input	<i>Double-click to export</i>	[clk]
kernel_reset	Reset Output	<i>Double-click to export</i>	[kernel_clk]
sw_reset_export	Reset Output	<i>Double-click to export</i>	[clk]
ad_bsp_memorg_hos...	Conduit	<i>Double-click to export</i>	
kernel_irq_from_kernel	Interrupt Receiver	<b>kernel_irq</b>	[kernel_clk]
kernel_irq_to_host	Interrupt Sender	<i>Double-click to export</i>	[kernel_clk]



# Hard IP for PCI Express

◀ PCIe Hard IP handles host-to-device communication

Name	Description	Export	Clock
<b>pcie</b>	Arria 10 Hard IP for PCI Express		
coreclkout_bip	Clock Output	<i>Double-click to export</i>	pcie_coreclk...
<b>refclk</b>	Clock Input	<i>Double-click to export</i>	<b>pcie_refclk...</b>
npwr	Conduit	<b>pcie_npwr</b>	
app_nreset_status	Reset Output	<i>Double-click to export</i>	pcie_coreclk...
hip_ctrl	Conduit	<i>Double-click to export</i>	
hip_pipe	Conduit	<i>Double-click to export</i>	
hip_serial	Conduit	<b>pcie_hip_serial</b>	
msi_intf	Conduit	<i>Double-click to export</i>	
msi_control	Conduit	<i>Double-click to export</i>	
msix_intf	Conduit	<i>Double-click to export</i>	
intx_intf	Conduit	<i>Double-click to export</i>	
txs	Avalon Memory Mapped Slave	<i>Double-click to export</i>	pcie_coreclk...
rxm_bar4	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
dma_rd_master	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
dma_wr_master	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
rd_dts_slave	Avalon Memory Mapped Slave	<i>Double-click to export</i>	pcie_coreclk...
wr_dts_slave	Avalon Memory Mapped Slave	<i>Double-click to export</i>	pcie_coreclk...
rd_dcm_master	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
wr_dcm_master	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
skpdetect	Conduit	<i>Double-click to export</i>	



`create_clock -period 100MHz [get_ports pcie_refclk]`

◀ Modify the top.sdc file if the refclk frequency changes

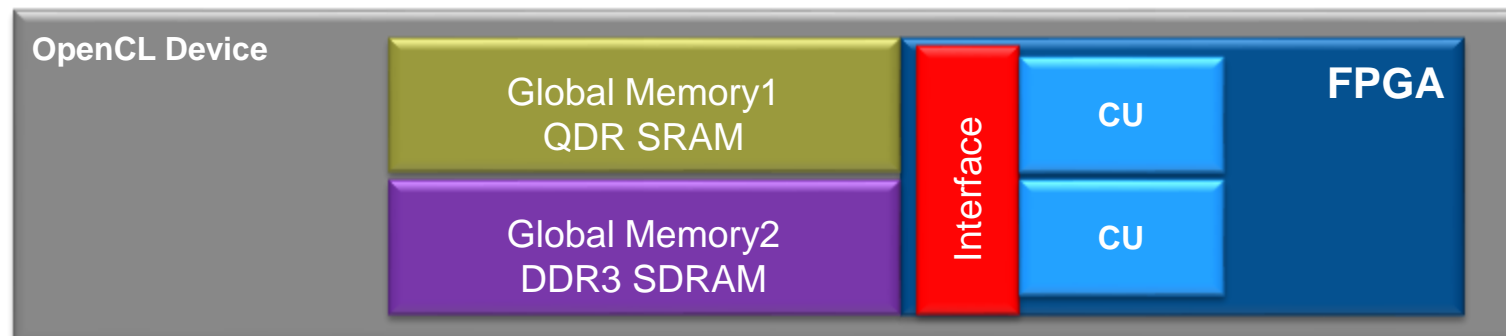
See [28nm PCIe online training](#) and [PCIe instructor-led training](#) & device-specific [Hard IP for PCIe User Guides](#)

## External Memory Buffers

- ◀ A BSP may support different memory device types
  - Take advantage of memory device characteristics

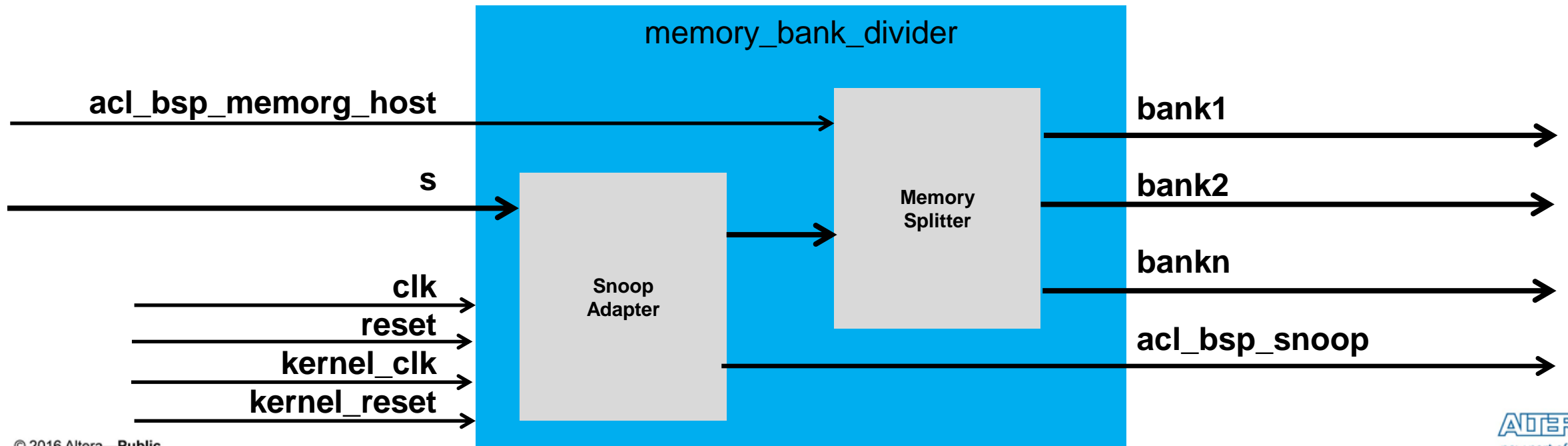
	Latency	Density	Cost	Usage
DDR SDRAM	high	high	low	Ideal for sequential access applications such as input/output data
QDR SRAM	low	low	high	Better suited for random access applications such as look-up tables

- ◀ External memory information is specified in `board_spec.xml` (discussed later)



# OpenCL Memory Bank Divider

- ◀ Interface host to kernel memory
- ◀ Multiple banks support interleaving memory
- ◀ Provide at least one Memory Bank Divider for each memory type
- ◀ Number of banks and memory type must be entered into the board\_spec.xml file (*discussed later*)




# OpenCL SGDMA Controller

## Controlled from the Host

- PCIe Tx BAR0 Master

## Connect to

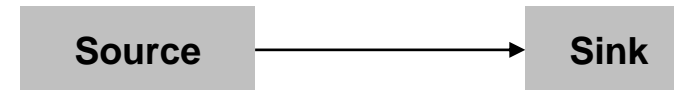
- Host PCIe Rx Slave
- All global memories
  - Through Memory Bank Divider if used

Name	Description	Export	Clock
 <b>pcie</b>	Arria 10 Hard IP for PCI Express		
coredkout_hip	Clock Output	<i>Double-click to export</i>	pcie_coreclk...
refclk	Clock Input	<i>Double-click to export</i>	<b>pcie_refclk...</b>
npwr	Conduit		
app_nreset_status	Reset Output	<b>pcie_npwr</b>	
hip_ctrl	Conduit	<i>Double-click to export</i>	pcie_coreclk...
hip_pipe	Conduit	<i>Double-click to export</i>	
hip_serial	Conduit	<i>Double-click to export</i>	
msi_intf	Conduit	<b>pcie_hip_serial</b>	
msi_control	Conduit	<i>Double-click to export</i>	
msix_intf	Conduit	<i>Double-click to export</i>	
intx_intf	Conduit	<i>Double-click to export</i>	
txs	Avalon Memory Mapped Slave	<i>Double-click to export</i>	pcie_coreclk...
rxm_bar4	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
<b>dma_rd_master</b>	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
<b>dma_wr_master</b>	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
rd_dts_slave	Avalon Memory Mapped Slave	<i>Double-click to export</i>	pcie_coreclk...
wr_dts_slave	Avalon Memory Mapped Slave	<i>Double-click to export</i>	pcie_coreclk...
rd_dcm_master	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
wr_dcm_master	Avalon Memory Mapped Master	<i>Double-click to export</i>	pcie_coreclk...
skpdetect	Conduit	<i>Double-click to export</i>	pcie_coreclk...



# Avalon-ST Interface

- Used by Altera OpenCL channels or OpenCL 2.0 pipes
- Standard, flexible, and modular protocol for transfer of data
  - Unidirectional
  - Point-to-point connections
  - Fully synchronous
  - Supports simple and complex interface requirements
- Source interface
  - Launches data on rising edges of associated clock
- Sink interface
  - Latches data on rising edges of associated clock
- Data format/definition controlled by application or component



# Avalon-ST Interface Signals

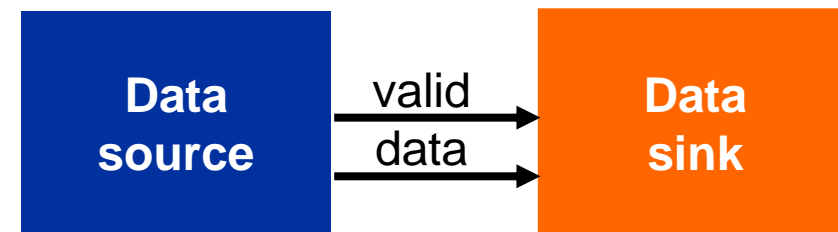
Signal type	Width	Direction	Description
<b>Fundamental signals</b>			
ready	1	Sink → Source	Indicates the sink can accept data (backpressure control)
valid	1	Source → Sink	Qualifies all source to sink signals
data	1-4096	Source → Sink	Payload of the information being transmitted
channel	1-128	Source → Sink	Channel number for data being transferred (if multiple channels supported)
error	1-256	Source → Sink	Bit mask marks errors affecting the data being transferred
<b>Packet transfer signals</b>			
startofpacket	1	Source → Sink	Marks the beginning of the packet
endofpacket	1	Source → Sink	Marks the end of the packet
empty	1-8	Source → Sink	Indicates the number of symbols that are empty during cycles that contain the end of a packet

Grayed out signals are not supported by OpenCL channels

# Simple Streaming Examples

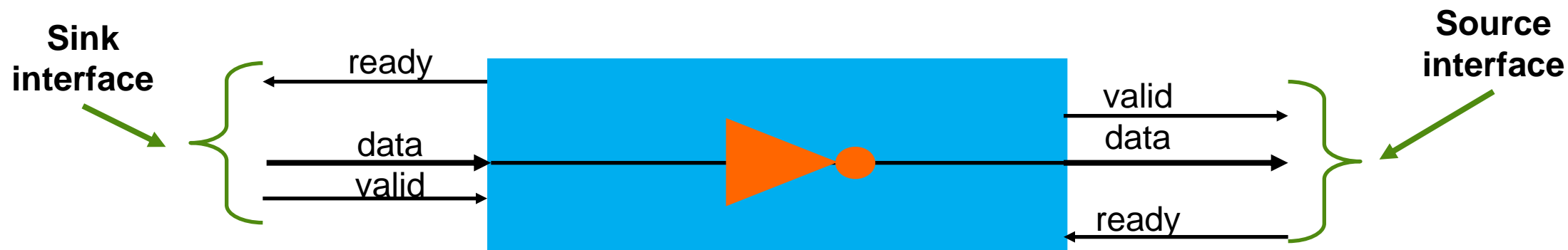
## Simple example

- **data** (presents information)
- **valid** (indicates data is valid)
- Both signals propagate from source to sink
- Sink cannot backpressure or stall transfer if valid is asserted



## Another example

- 32b inverter block in datapath in Qsys system
- **ready** used to “throttle” the transfer



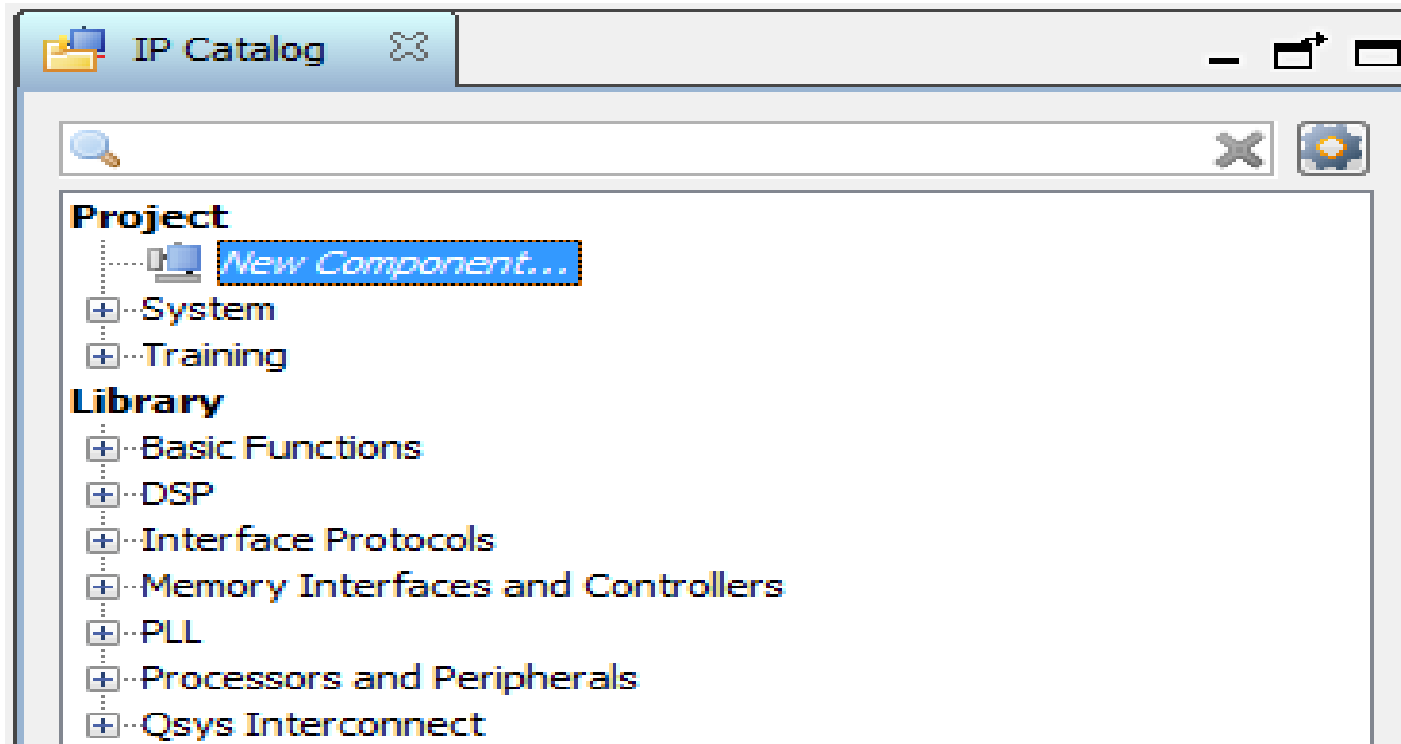
## Hardware Procedure – Modify the Platform

1. Open Quartus software project in the \boardtest\boardtest directory
2. Add or remove components in the board.qsys file
3. Add or remove signals in the system.qsys and top.v files
4. Add or remove SDC constraints in the top.sdc and top\_post.sdc files
5. Add or remove LogicLock Plus regions in the base.qsf file
6. Propagate all global assignments from base.qsf to the top.qsf and top\_synth.qsf files
7. Copy any files modified above into your custom BSP directory<sup>1</sup>
8. Conduct a base compile with boardtest.cl using several seeds
9. Verify timing
10. Copy the base\_qhd.qar file to your custom BSP directory<sup>1</sup>
11. Conduct an import compile with a simple kernel
12. Verify error free compile

1. These steps are easy to forget.

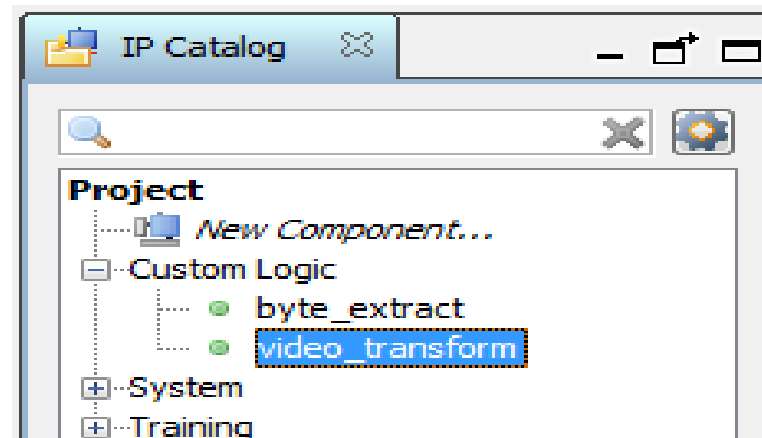
## Component Editor

- Used to import components into Qsys system
- Launch from Qsys IP Catalog or File menu → New Component



## \_hw.tcl File

- ◀ Only file generated by Component Editor
- ◀ Describes all component settings
- ◀ Portability: \_hw.tcl plus HDL code all that is needed to import a component into other projects
- ◀ Makes component look and feel like any other component in IP Catalog
- ◀ Tcl syntax discussed in Advanced Qsys Design Methodologies class

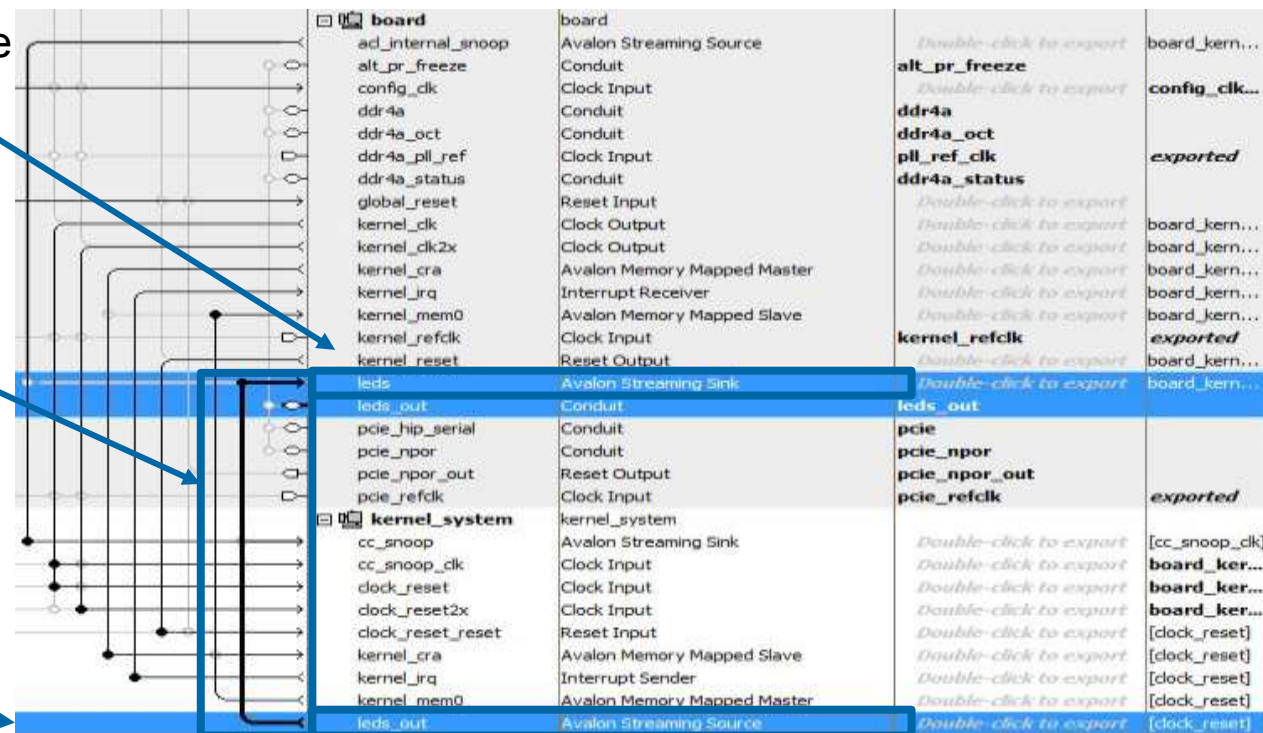


# If New Component is an IO Channel

◀ Add the channel the board\_spec.xml file

```
<interface name="board" port="leds" type="streamsink" width="16" chan_id="leds_out"/>
```

- ◀ type="streamsink" width="16"
  - Directs the compiler to make the "leds\_out" interface a streaming source that is 16 bits wide
- ◀ name="board" port="leds"
  - Directs the compiler to connect the "leds\_out" interface on the kernel.qsys system to the "leds" interface on the board.qsys system
- ◀ chan\_id="leds\_out"
  - Directs the compiler to add an exported interface to the kernel.qsys called "leds\_out".



system.qsys file

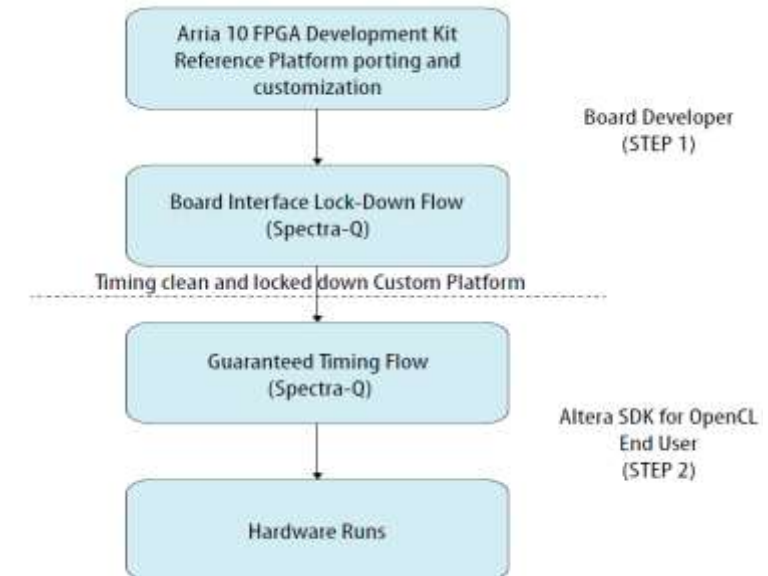
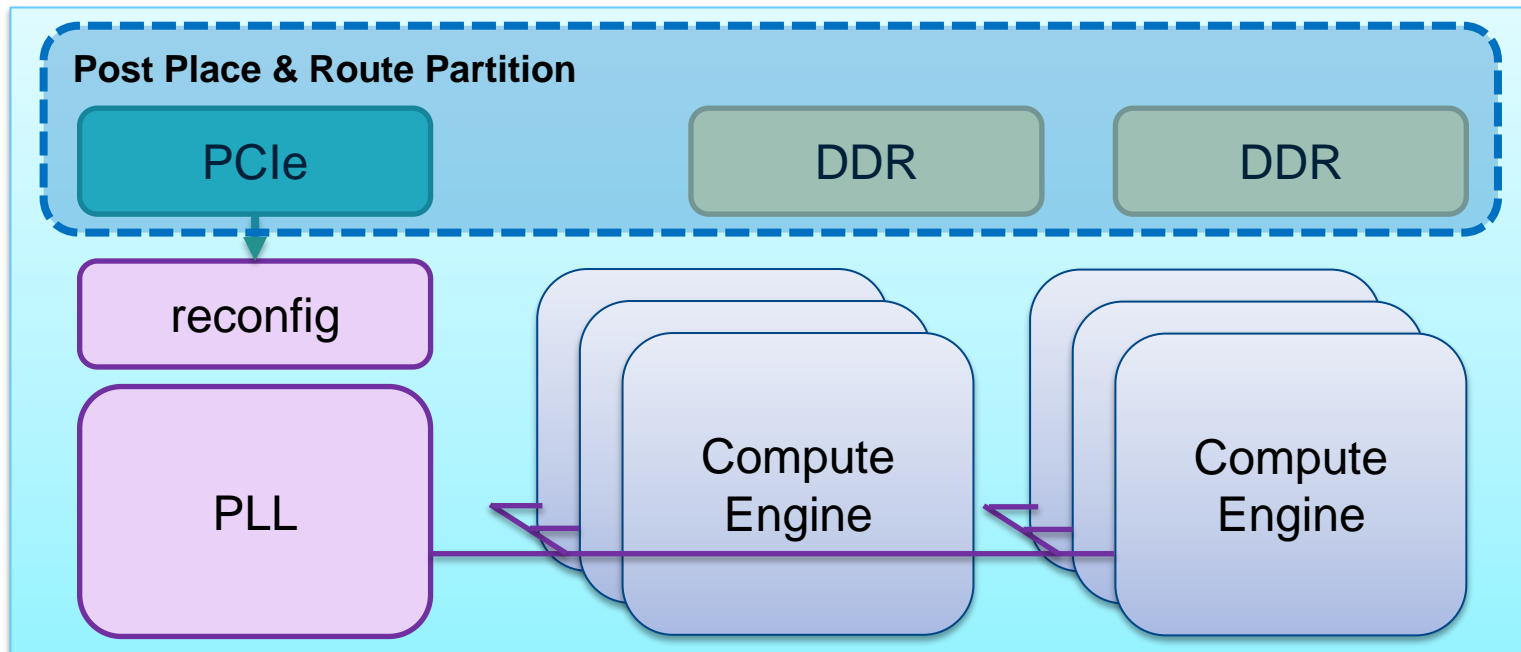
# Guaranteed Timing Closure

Some interfaces have required clock frequencies

- PCIe 125 MHz / 250 MHz
- DDR3-1600 800 MHz
- Kernel ??



The custom board developer is responsible for delivering a locked down, timing clean netlist for the custom platform





# Developing a Custom OpenCL BSP

Software Development



# Board XML Files Overview

- ◀ Platforms must include XML files
  - Describes your platform to the Altera SDK for OpenCL
  
- ◀ board\_env.xml
  - Describes the properties of your platform
    - ◀ e.g. library location, utility directory
  
- ◀ board\_spec.xml
  - Contains metadata describing your hardware system
    - ◀ e.g. memory properties, device resources used, interfaces, etc

## Board Environment XML

- ◀ AOCL\_BOARD\_PACKAGE\_ROOT points to directory where the `board_env.xml` is located
- ◀ Sets up board installation enabling AOC to target specific boards
- ◀ Template available in the `/board_package` directory of the Custom Platform Toolkit
  
- ◀ Top level elements
  - `hardware` element
  - One `platform` element for each supported OS
- ◀ Each `platform` element contains
  - `mmdlib`, `linkflags`, `linklibs`, `utilbindir`

## Board Description File – board\_env.xml

```
<?xml version="1.0"?>
<board_env version="15.1" name="MyPlatformName">
  <hardware dir="hardware" default="MyBoard"></hardware>

  <platform name="linux64">
    <mmdlib>%b/linux64/lib/libaltera_a10_ref_mmd.so</mmdlib>
    <linkflags>-L%b/linux64/lib</linkflags>
    <linklibs>-laltera_a10_ref_mmd </linklibs>
    <utilbindir>%b/linux64/libexec</utilbindir>
  </platform>

  <platform name="windows64">
    <mmdlib>%b/windows64/bin/altera_a10_ref_mmd.dll</mmdlib>
    <linkflags>/libpath:%b/windows64/lib</linkflags>
    <linklibs>altera_a10_ref_mmd.lib</linklibs>
    <utilbindir>%b/windows64/libexec</utilbindir>
  </platform>

</board_env>
```

%a references the AOCL installation directory (e.g. c:\altera\15.1\hld)

%b references your BSP installation directory (e.g. c:\altera\15.1\hld\board\MyPlatform)

## board\_env.xml Elements and Attributes

Element	Attributes and Descriptions
board_env	<code>version</code> : AOCL version used to develop the platform <code>name</code> : Name of Custom Platform board directory
hardware	<code>dir</code> : Subdirectory containing board variants <code>default</code> : Default board variant
platform	<code>name</code> : Name of OS
mmdlib	Path to the dynamic MMD libraries of the Custom Platform
linkflags	Linker flags necessary to statically link with the MMD layer
linklibs	Libraries the AOCL must statically link against
utilbindir	Directory where AOCL utility executables are located ( <code>install</code> , <code>uninstall</code> , <code>program</code> , <code>diagnose</code> , and <code>flash</code> )

## Testing board\_env.xml

1. Set AOCL\_BOARD\_PACKAGE\_ROOT to location of the Custom Platform
2. Run `aocl board-xml-test`

```
C:\Users\kqi>aocl board-xml-test
board-path      = C:\altera\15.1\hld\board\MyPlatformName
board-version   = 15.1
board-name      = MyPlatformName
board-default   = MyBoard
board-hw-path   = C:\altera\15.1\hld\board\MyPlatformName\hardware\MyBoard
board-link-flags = /libpath:C:\altera\15.1\hld\board\MyPlatformName\windows64\lib
board-libs      = altera_a10_ref_mmd.lib
board-util-bin  = C:\altera\15.1\hld\board\MyPlatformName\windows64\libexec
board-mmdlib    = C:\altera\15.1\hld\board\MyPlatformName\windows64\bin\altera_a10_ref_mmd.dll
```

3. F

```
C:\>aoc --list-boards
Board list:
a10gx
a10gx_es2
a10gx_es3
```

# Board Spec XML File (1)

```
<?xml version="1.0"?>
<board version="0.9" name="MyBoard">

  <device device_model="10ax115s2f45i2sges_dm.xml">
    <used_resources>
      <alms num="45000"/>
      <ffs num="117500"/>
      <dsps num="0"/>
      <rams num="583"/>
    </used_resources>
  </device>

  <!-- DDR4-2400 -->
  <global_mem name="DDR" max_bandwidth="19200" interleaved_bytes="1024" config_addr="0x18">
    <interface name="board" port="kernel_mem0" type="slave" width="512" maxburst="16"
      address="0x00000000" size="0x80000000" latency="240" addpipe="1" />
  </interface/>
</global_mem>

<channels>
  <interface name="udp_0" port="udp0_out" type="streamsource" width="256" chan_id="eth0_in"/>
  <interface name="udp_0" port="udp0_in" type="streamsink" width="256" chan_id="eth0_out"/>
</channels>
```

## Board Spec XML File (2)

```
<host>
  <kernel_config start="0x00000000" size="0x0100000"/>
</host>

<interfaces>
  <interface name="board" port="kernel_cra" type="master" width="64" misc="0"/>
  <interface name="board" port="kernel_irq" type="irq" width="1"/>
  <interface name="board" port="acl_internal_snoop" type="streamsource" enable="SNOOPENABLE"
    width="33" clock="board.kernel_clk"/>
  <kernel_clk_reset clk="board.kernel_clk" clk2x="board.kernel_clk2x" reset="board.kernel_reset"/>
</interfaces>

<compile project="top" revision="top" qsys_file="system.qsys" generic_kernel="1">
  <generate cmd="echo"/>
  <synthesize cmd="quartus_sh -t import_compile.tcl"/>
  <auto_migrate platform_type="MyPlatform" >
    <include fixes=""/>
    <exclude fixes=""/>
  </auto_migrate>
</compile>

</board>
```



## board\_spec.xml Elements and Attributes

Element	Attributes and Descriptions
board	<code>version</code> : AOCL version used to develop the platform <code>name</code> : Name of the current board directory
device	<code>device_model</code> : Device model file describing FPGA resources <code>used_resources</code> : FPGA resource used by the BSP hardware
global_mem	<code>name</code> , <code>max_bandwidth</code> , <code>interleaved_bytes</code> , <code>config_addr</code> , <code>interface</code> : global memory properties
host	<code>kernel_config</code> : Address offset where the kernel hardware resides
[channels]	<code>interface</code> : Characteristics of each channel interface for direct kernel-to-I/O accesses
interfaces	<code>interface</code> , <code>kernel_clk_reset</code> : Description of kernel interfaces connection to and controlling the kernel hardware
compile	<code>project</code> , <code>qsys_files</code> , <code>generate_cmd</code> , etc... : Controls Quartus Prime compilation

# Board XML Files Review

## ◀ board\_env.xml

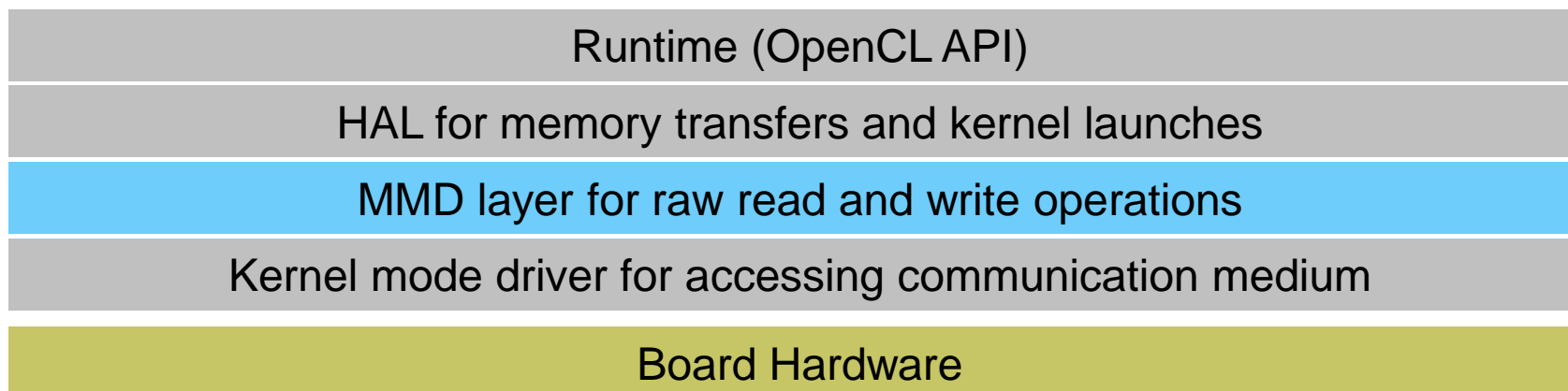
- One needed for each platform
- Describes the properties of your platform
  - ◀ e.g. library location, utility directory

## ◀ board\_spec.xml

- One needed for each board within the platform
- Contains metadata describing your hardware system
  - ◀ Memory properties
  - ◀ Channel properties
  - ◀ Device resources used
  - ◀ Control interfaces
  - ◀ Compile properties
  - ◀ etc.

## Memory-Mapped Devices (MMD) Software Layer

- Software layer for communicating with board
  - Over any medium
- Used by host programs and board utilities
- File I/O like interface needs to be implemented
  - Read/write/open/close etc.
- To be linked to by the host program
  - Statically and dynamically



# MMD API

<code>get_offline_info</code>	<code>get_info</code>
<code>set_status_handler</code>	<code>set_interrupt_handler</code>
<code>open</code>	<code>close</code>
<code>read</code>	<code>write</code>
<code>copy</code>	<code>yield</code>
<code>shared_mem_alloc</code>	<code>shared_mem_free</code>
<code>reprogram</code>	

## AOCL Utilities

- ◀ Custom Platforms must support a set of **aocl** utilities
  - Executables delivered in a subdirectory within the Custom Platform files
- ◀ AOCL looks for corresponding executables when respective **aocl** calls are made
- ◀ **install** (`aocl install`)
  - Installs driver into the host operating system
- ◀ **uninstall** (`aocl install`)
  - Removes driver from the host operating system
- ◀ **program** (`aocl program <device> <kernel file>.aocx`)
  - Programs the FPGA using the provided aocx file
- ◀ **flash** (`aocl flash <device> <kernel_file>.aocx`)
  - Programs base programming image into Flash
- ◀ **diagnose** (`aocl diagnose [<device_name>]`)
  - Confirms board functionality

# Summary



**ALTERA**  
now part of Intel

## 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, HDMI, 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



**ALTERA**  
now part of Intel