Access Intel® FPGAs for Acceleration

Karl Qi

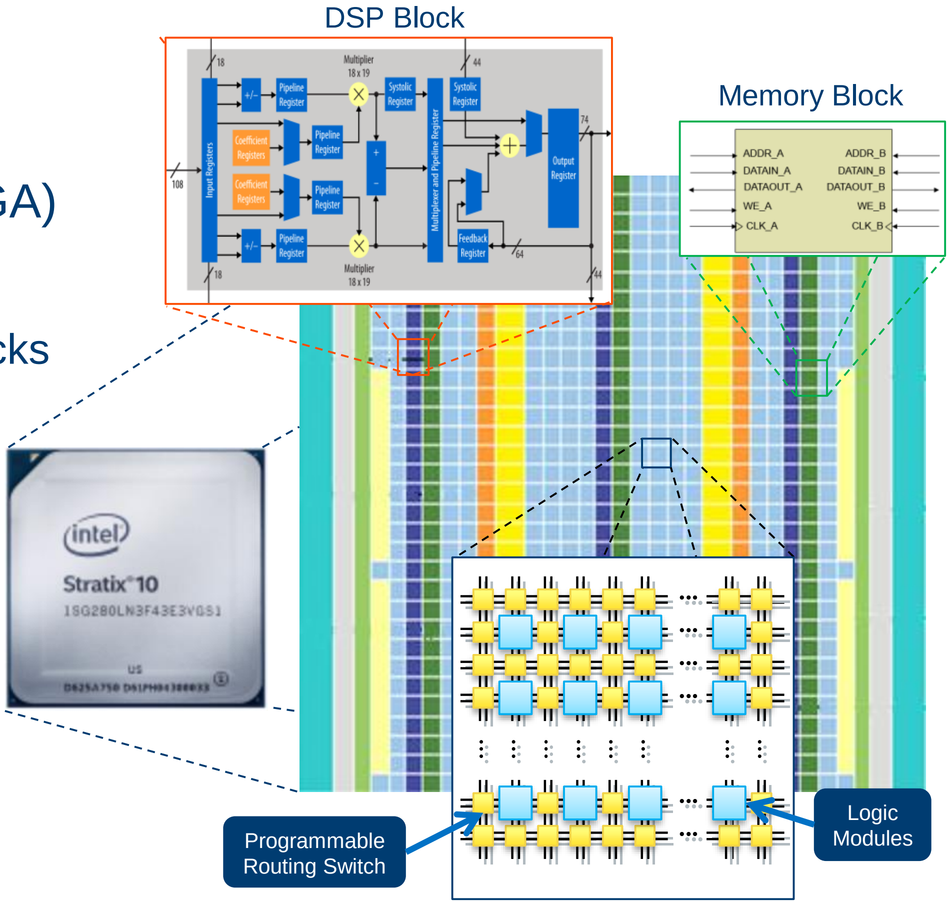
CERN 2018

Agenda

- **High-level synthesis with the Intel® HLS Compiler**
- Intel® FPGA SDK for OpenCL™
- Acceleration Stack for Intel® Xeon CPUs and FPGAs
- Deep Learning Inference on FPGAs

FPGA Overview

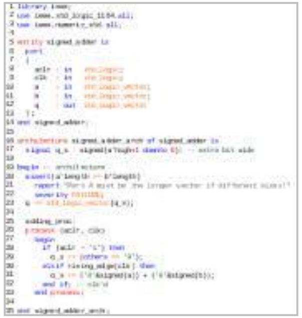
- Field Programmable Gate Array (FPGA)
  - Millions of logic elements
  - Thousands of embedded memory blocks
  - Thousands of DSP blocks
  - Programmable routing
  - High speed transceivers
  - Various built-in hardened IP
- Used to create **Custom Hardware!**



Traditional FPGA Design Process

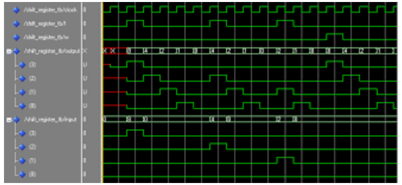
Potentially Time-Consuming Effort

HDL

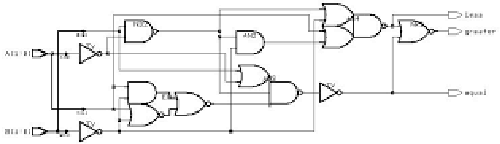


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_adder is
  port
  (
    aclr : in std_logic;
    clk  : in std_logic;
    a    : in std_logic_vector;
    b    : in std_logic_vector;
    q    : out std_logic_vector
  );
end signed_adder;

architecture signed_adder_arch of signed_adder is
  signal q_s : signed(a'high+1 downto 0); -- extra bit wide

begin -- architecture
  assert(a'length >= b'length)
    report "Port A must be the longer vector if different sizes!"
    severity FAILURE;
  q <= std_logic_vector(q_s);

  adding_proc:
  process (aclr, clk)
    begin
      if (aclr = '1') then
        q_s <= (others => '0');
      elsif rising_edge(clk) then
        q_s <= ('0'&signed(a)) + ('0'&signed(b));
      end if; -- clk'd
    end process;

end signed_adder_arch;
```

Behavioral Simulation



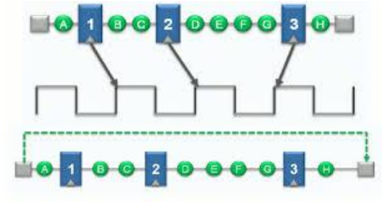
Synthesis



Place & Route / Timing Analysis / Timing Closure

Intel® Quartus® Prime

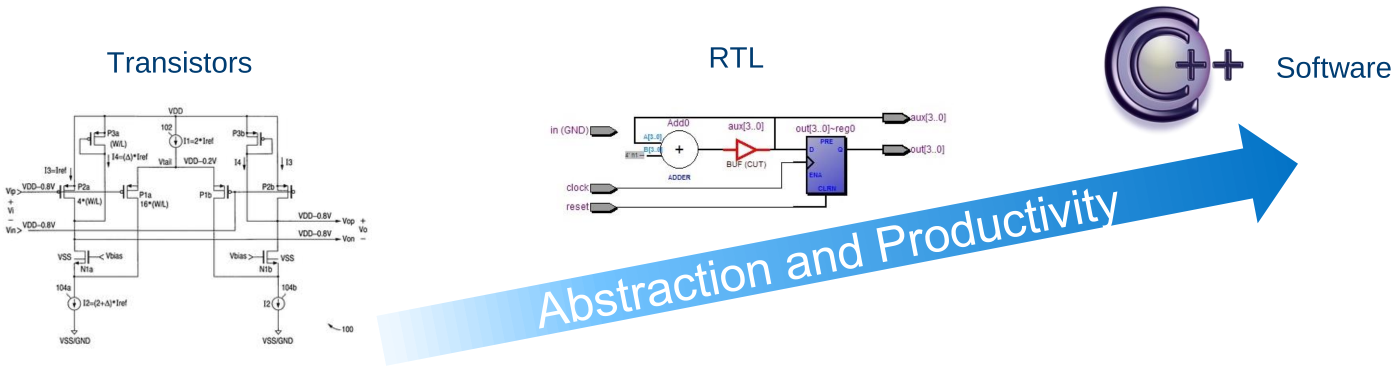
Design Software



Why HLS?

Designing IP at a higher level of abstraction = increase productivity

- Debugging software is much faster than hardware
- Easier to specify functions in software
- Productivity tool for RTL designers



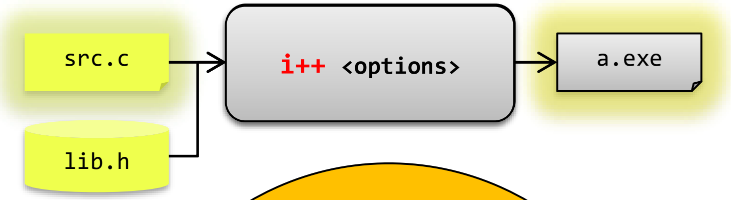
Transistors

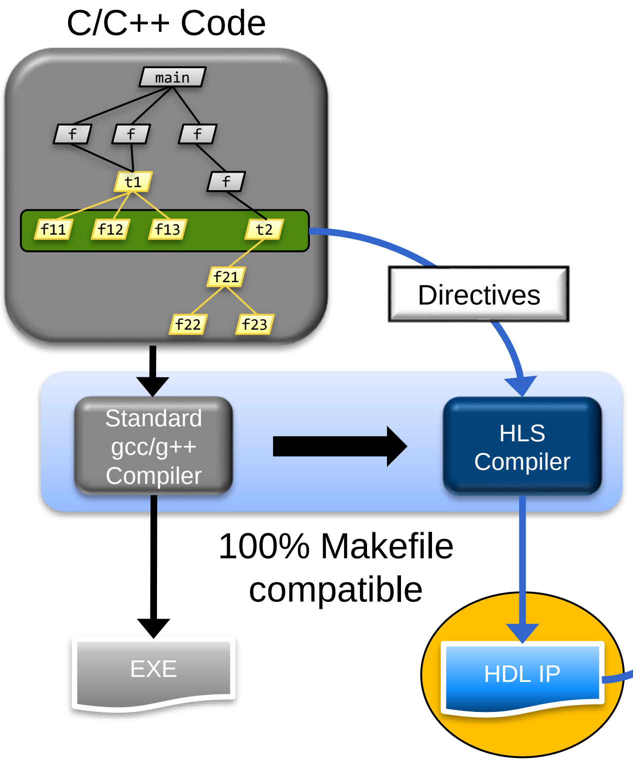
RTL

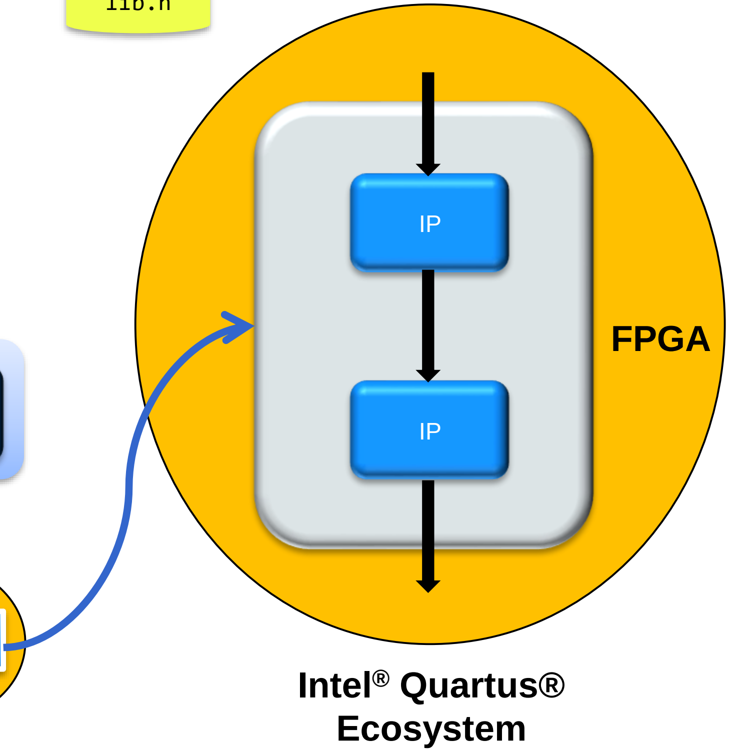
Software

Abstraction and Productivity

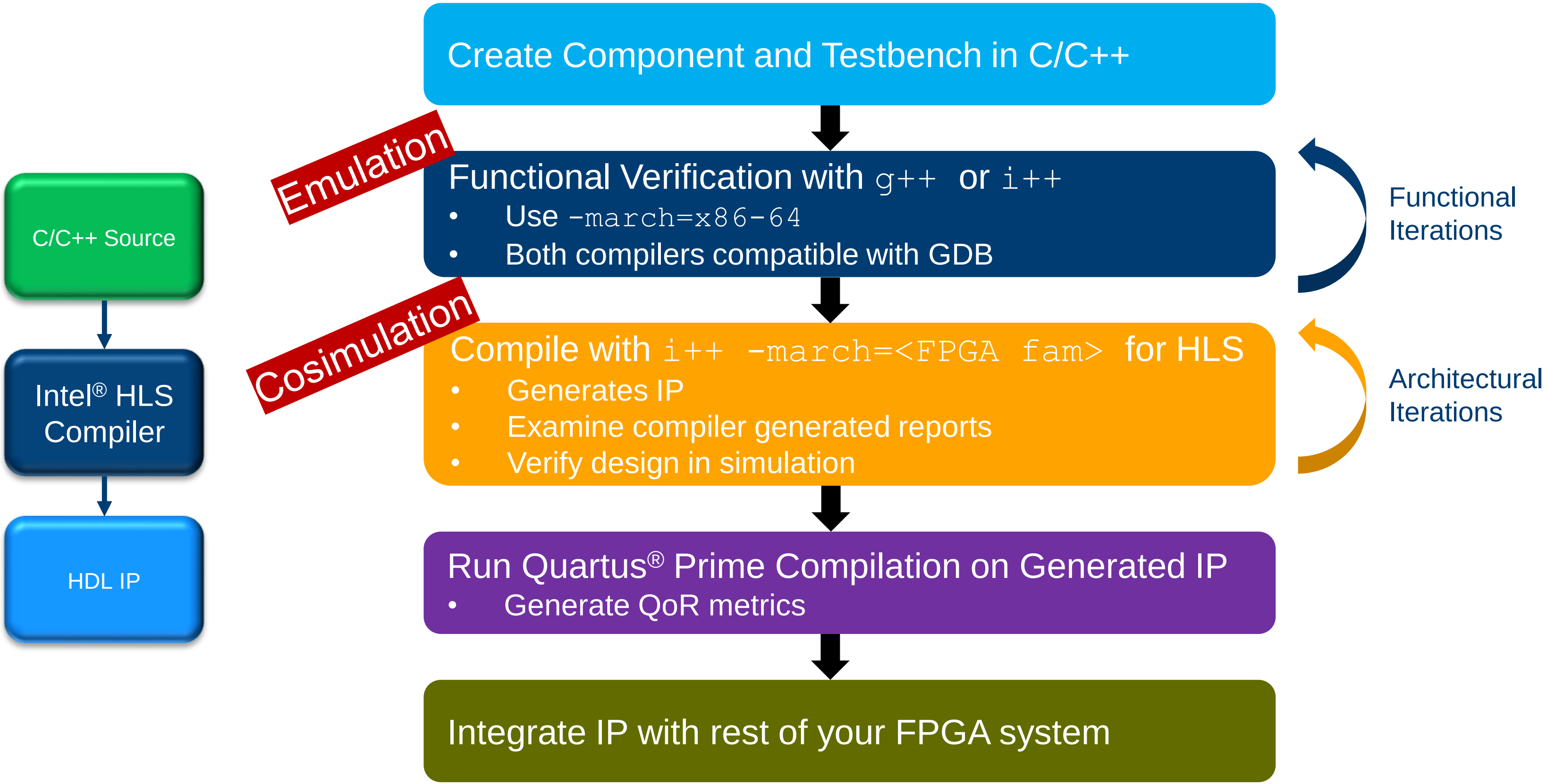
HLS Use Model







HLS Procedure



C/C++ Source

Intel® HLS Compiler

HDL IP

Create Component and Testbench in C/C++

Emulation

Functional Verification with `g++` or `i++`
- Use `-march=x86-64`
- Both compilers compatible with GDB

Functional Iterations

Cosimulation

Compile with `i++ -march=<FPGA fam>` for HLS
- Generates IP
- Examine compiler generated reports
- Verify design in simulation

Architectural Iterations

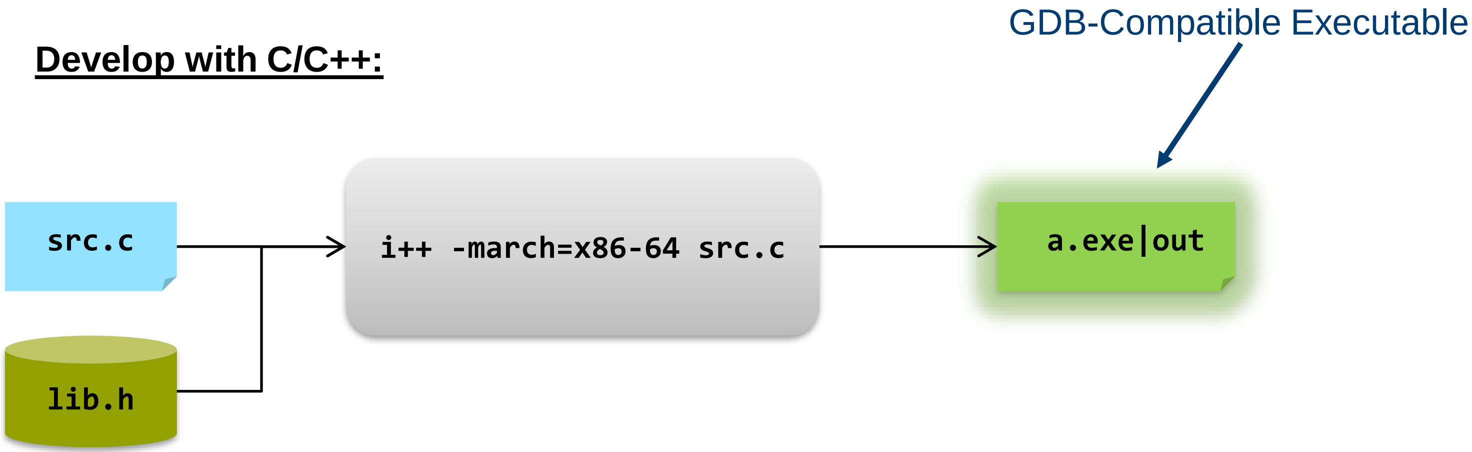
Run Quartus® Prime Compilation on Generated IP
- Generate QoR metrics

Integrate IP with rest of your FPGA system

Emulation Mode

- Just like any executing any other software
- Debug with
  - printf/cout
  - gdb
  - Valgrind

Develop with C/C++:



GDB-Compatible Executable

src.c

lib.h

```
i++ -march=x86-64 src.c
```

a.exe|out

Cosimulation: Synthesize Component Function into RTL

```
#include "HLS/hls.h"
#include "assert.h"
#include "HLS/stdio.h"
#include "stdlib.h"

component int accelerate(int a, int b) {
        return a+b;
}

int main() {
        srand(0);
        for (int i=0; i<10; ++i) {
                int x=rand() % 10;
                int y=rand() % 10;
                int z=accelerate(x, y);
                printf("%d + %d = %d\n", x, y, z);
                assert(z == x + y);
        }
        return 0;
}
```

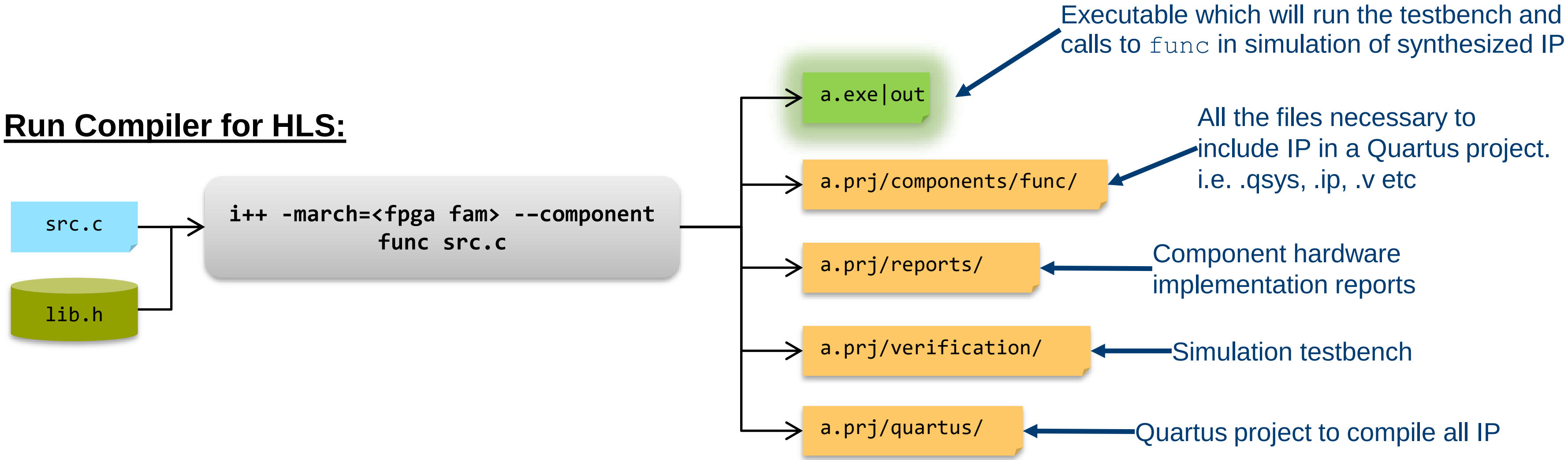
```
i++ -march=<fpga family> --component accelerate mysource.cpp
```

`accelerate()` becomes an FPGA component

- Use `--component` i++ argument or `component` attribute in source

`main()` becomes testbench for component `accelerate()`

The Cosimulation Flow



Run Compiler for HLS:

`src.c`

`lib.h`

```
i++ -march=<fpga fam> --component
func src.c
```

`a.exe|out` — Executable which will run the testbench and calls to `func` in simulation of synthesized IP

`a.prj/components/func/` — All the files necessary to include IP in a Quartus project. i.e. .qsys, .ip, .v etc

`a.prj/reports/` — Component hardware implementation reports

`a.prj/verification/` — Simulation testbench

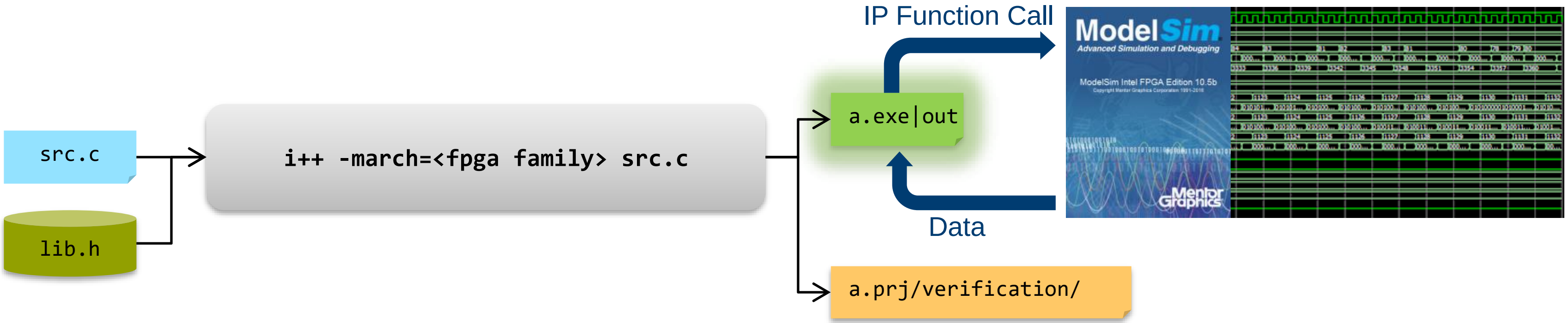
`a.prj/quartus/` — Quartus project to compile all IP

a is the default output name, -o option can be used to specify a non-default output name

Cosimulation Verifying HLS IP

The Intel® HLS compiler automatically compiles and links C++ testbench with an instance of the component running in an RTL simulator

- To verify RTL behavior of IP, just run the executable generated by the HLS compiler targeting the FPGA architecture
  - Any calls to the component function becomes calls the simulator through DPI



src.c

lib.h

`i++ -march=<fpga family> src.c`

a.exe|out

IP Function Call

Data

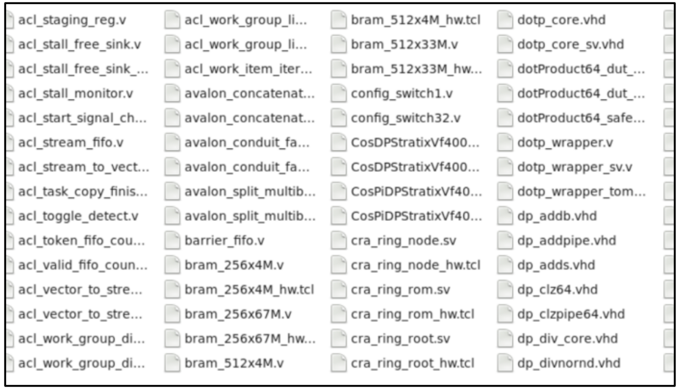
a.prj/verification/

ModelSim
Advanced Simulation and Debugging
ModelSim Intel FPGA Edition 10.5b
Mentor Graphics

C/C++ Functions to Dataflow Circuits

Each component function is converted into custom dataflow hardware

- Gain the benefits of Intel® FPGAs without the length design process
- Implement C/C++ operators as circuits
  - HDL code located in <HLS Installation>\ip
  - Load Store units to read/write memory
  - Arithmetic units to perform calculations
  - Flow control units
  - Connect circuits according to data flow in the function



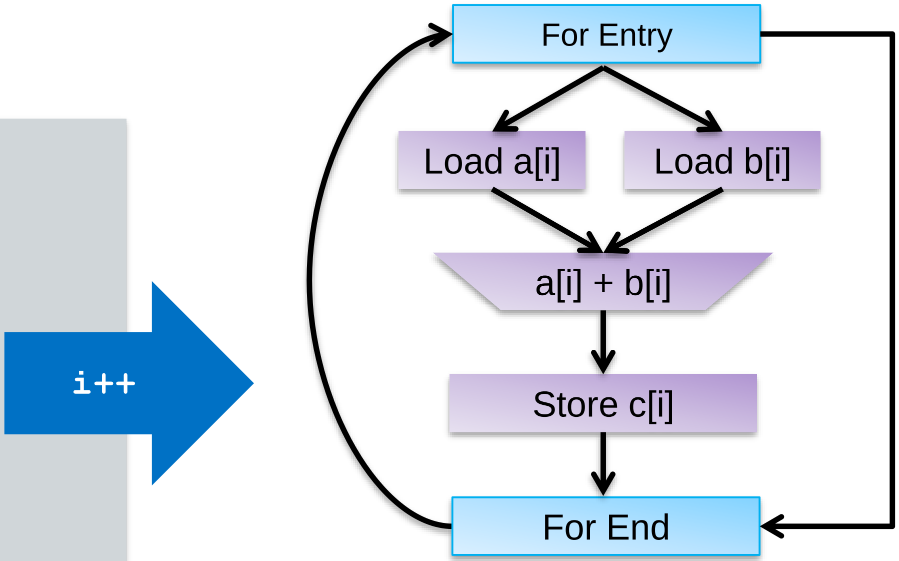
```
acl_staging_reg.v
acl_stall_free_sink.v
acl_stall_free_sink_...
acl_stall_monitor.v
acl_start_signal_ch...
acl_stream_fifo.v
acl_stream_to_vect...
acl_task_copy_finis...
acl_toggle_detect.v
acl_token_fifo_cou...
acl_valid_fifo_coun...
acl_vector_to_stre...
acl_vector_to_stre...
acl_work_group_di...
acl_work_group_di...
acl_work_group_li...
acl_work_group_li...
acl_work_item_iter...
avalon_concatenat...
avalon_concatenat...
avalon_conduit_fa...
avalon_conduit_fa...
avalon_split_multib...
avalon_split_multib...
barrier_fifo.v
bram_256x4M.v
bram_256x4M_hw.tcl
bram_256x67M.v
bram_256x67M_hw...
bram_512x4M.v
bram_512x4M_hw.tcl
bram_512x33M.v
bram_512x33M_hw...
config_switch1.v
config_switch32.v
CosDPStratixVf400...
CosDPStratixVf400...
CosPiDPStratixVf40...
CosPiDPStratixVf40...
cra_ring_node.sv
cra_ring_node_hw.tcl
cra_ring_rom.sv
cra_ring_rom_hw.tcl
cra_ring_root.sv
cra_ring_root_hw.tcl
dotp_core.vhd
dotp_core_sv.vhd
dotProduct64_dut_...
dotProduct64_dut_...
dotProduct64_safe...
dotp_wrapper.v
dotp_wrapper_sv.v
dotp_wrapper_tom...
dp_addb.vhd
dp_addpipe.vhd
dp_adds.vhd
dp_clz64.vhd
dp_clzpipe64.vhd
dp_div_core.vhd
dp_divnornd.vhd
```

Compilation Example

Software compiled into dataflow circuit with flow control

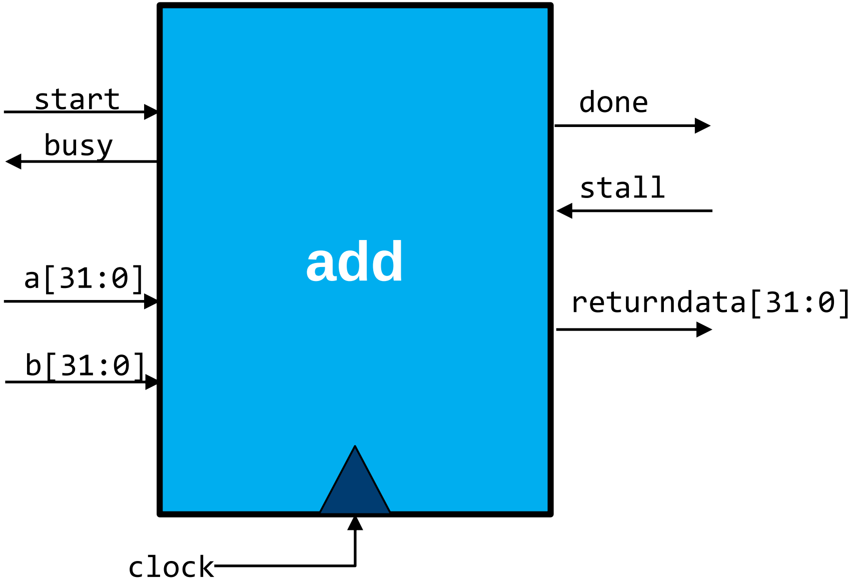
- Include branch and merge units

```
void my_component(    int *a,
                      int *b,
                      int *c,
                      int N)
{
    int i;
    for (i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```



The Default Interfaces

```
component int add(int a, int b) {
        return a+b;
}
```



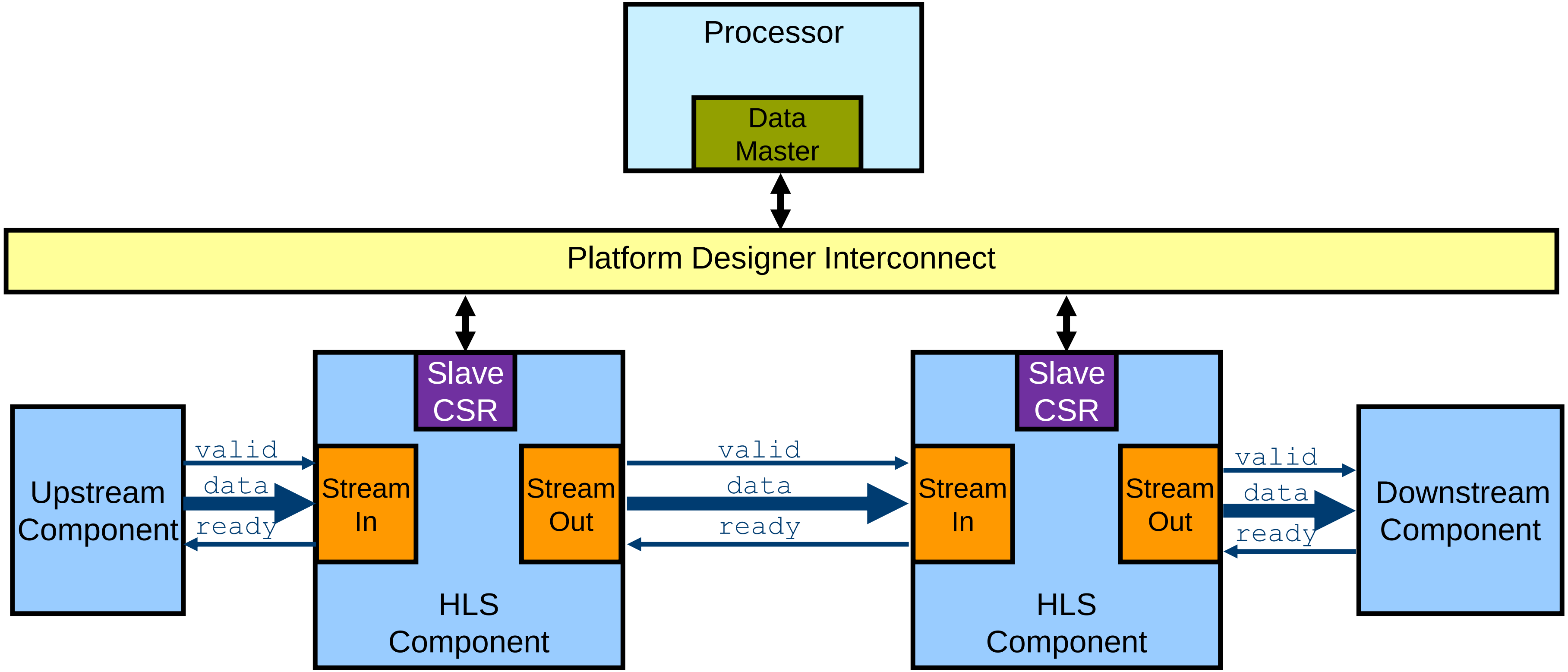
C++ Construct	HDL Interface
Scalar arguments	Conduits associated with the default start/busy interface
Pointer arguments	Avalon memory master interface
Global scalars and arrays	Avalon memory master interface

Note: more on interfaces later

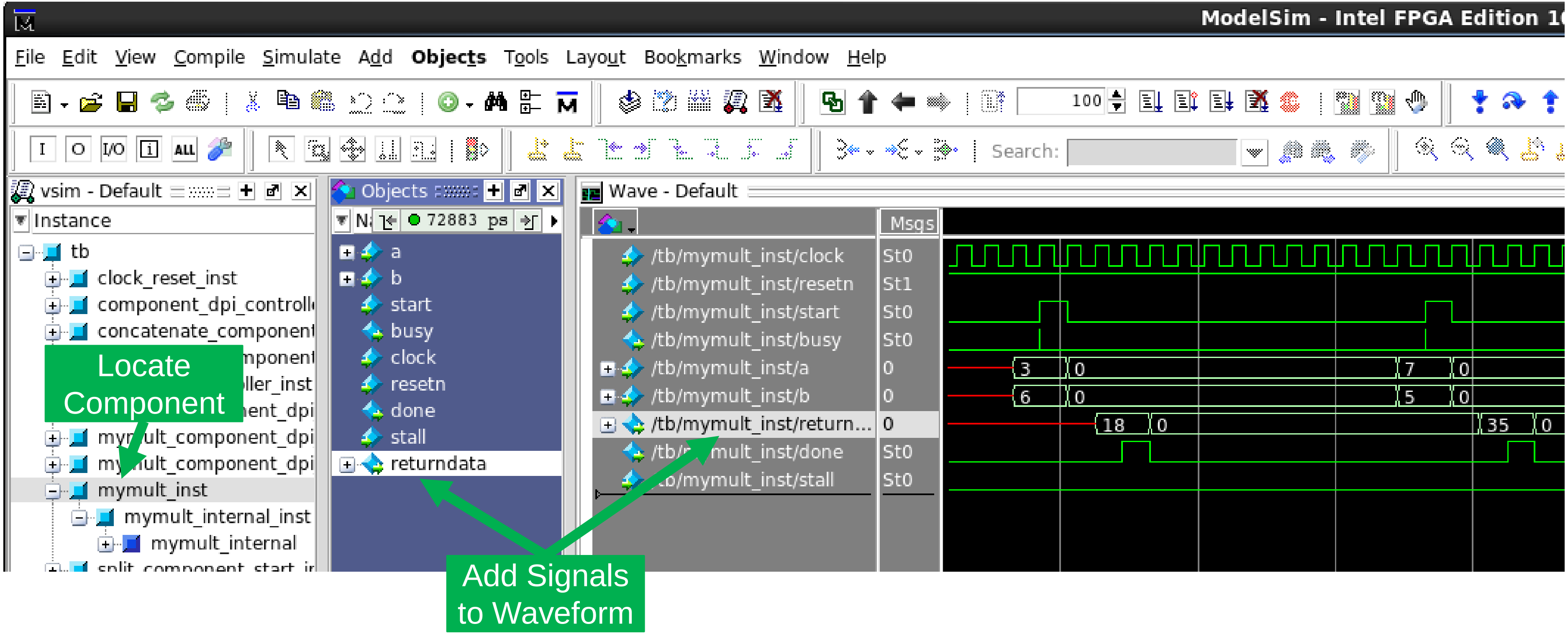
Other Custom Interfaces

- Customizable Avalon Streaming Interfaces
  - Explicit ready/valid signals for each data argument
- Explicit Memory-Mapped Master interfaces
  - Create a number of master interfaces with customizable features
- Slave Registers
  - Slave port for scalar values
- Slave Memory
- Slave Control
  - Call/Return interface done through register

MM HLS Component with Streaming Interfaces

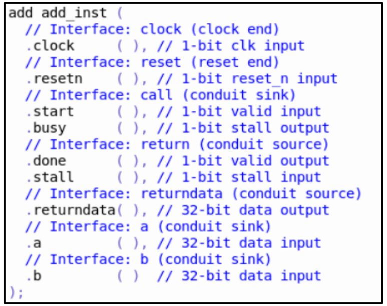


Viewing Waveforms in Modelsim

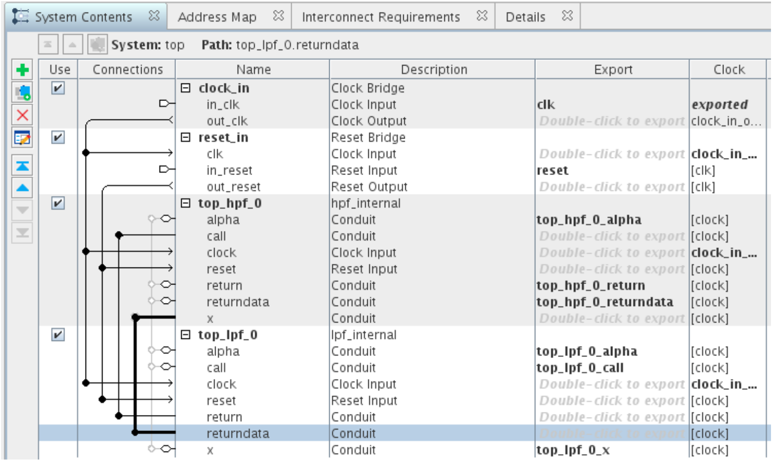


Intel® Quartus® Software Integration

- `a.prj/components` directory contains all the files to integrate
  - One subdirectory for each component
    - Portable, can be moved to a different location if desire
- 2 use scenarios
  1. Instantiate in HDL
  2. Adding IP to a Platform Designer system integration tool system



```
add add_inst (
  // Interface: clock (clock end)
  .clock      ( ), // 1-bit clk input
  // Interface: reset (reset end)
  .resetn     ( ), // 1-bit reset_n input
  // Interface: call (conduit sink)
  .start      ( ), // 1-bit valid input
  .busy       ( ), // 1-bit stall output
  // Interface: return (conduit source)
  .done       ( ), // 1-bit valid output
  .stall      ( ), // 1-bit stall input
  // Interface: returndata (conduit source)
  .returndata( ), // 32-bit data output
  // Interface: a (conduit sink)
  .a          ( ), // 32-bit data input
  // Interface: b (conduit sink)
  .b          ( )  // 32-bit data input
);
```

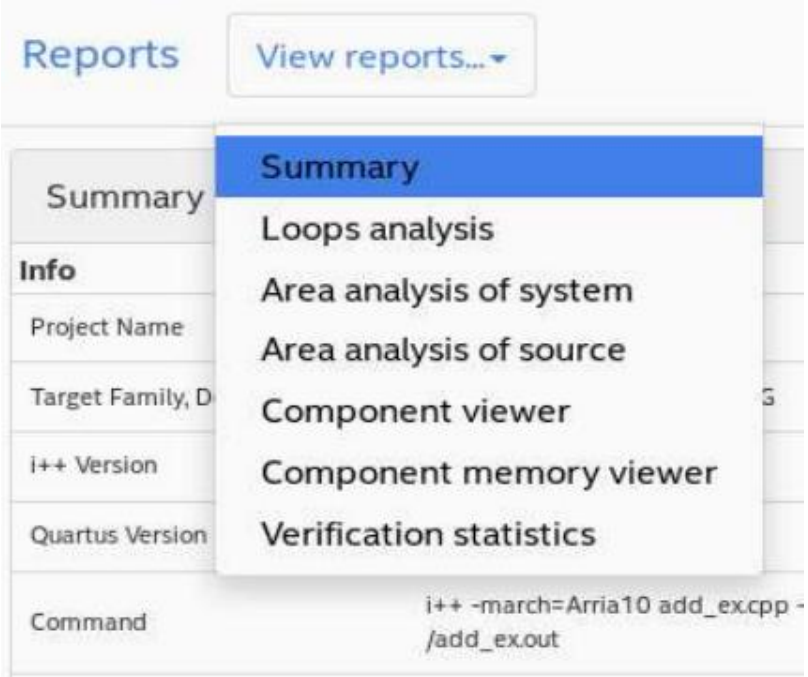


System: top **Path:** top_lpf_0.returndata

Use	Connections	Name	Description	Export	Clock
☑		⊟ **clock_in**	Clock Bridge		
		in_clk	Clock Input	**clk**	***exported***
		out_clk	Clock Output	*Double-click to export*	clock_in_o...
☑		⊟ **reset_in**	Reset Bridge		
		clk	Clock Input	*Double-click to export*	**clock_in_...**
		in_reset	Reset Input	**reset**	[clk]
		out_reset	Reset Output	*Double-click to export*	[clk]
☑		⊟ **top_hpf_0**	hpf_internal		
		alpha	Conduit	**top_hpf_0_alpha**	[clock]
		call	Conduit	*Double-click to export*	[clock]
		clock	Clock Input	*Double-click to export*	**clock_in_...**
		reset	Reset Input	*Double-click to export*	[clock]
		return	Conduit	**top_hpf_0_return**	[clock]
		returndata	Conduit	**top_hpf_0_returndata**	[clock]
		x	Conduit	*Double-click to export*	[clock]
☑		⊟ **top_lpf_0**	lpf_internal		
		alpha	Conduit	**top_lpf_0_alpha**	[clock]
		call	Conduit	**top_lpf_0_call**	[clock]
		clock	Clock Input	*Double-click to export*	**clock_in_...**
		reset	Reset Input	*Double-click to export*	[clock]
		return	Conduit	*Double-click to export*	[clock]
		returndata	Conduit	*Double-click to export*	[clock]
		x	Conduit	**top_lpf_0_x**	[clock]

Main HTML Optimization Report

Fast generation of optimization report

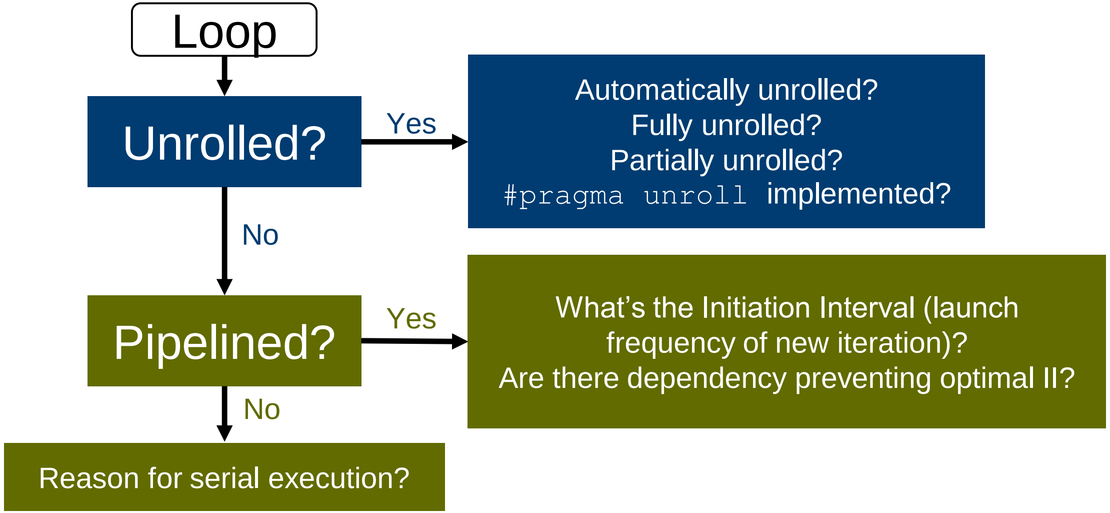


Summary	
Info	
Project Name	
Target Family, D	G
i++ Version	
Quartus Version	
Command	i++ -march=Arria10 add_ex.cpp - /add_ex.out

Loops

Serial loop execution hinders function dataflow circuit performance

- Use Loop Analysis report to see if and how each loop is optimized
  - Helps identify component pipeline bottlenecks

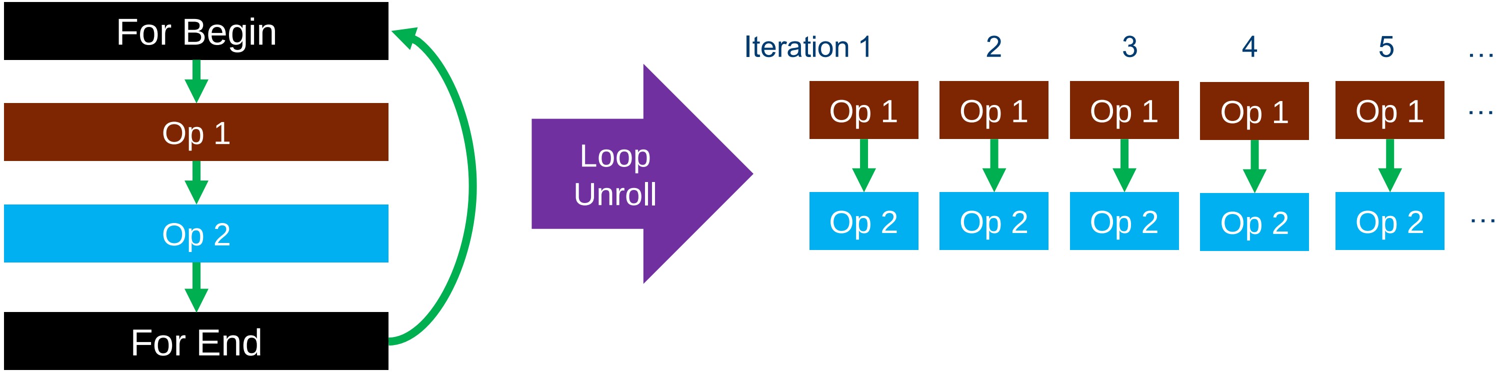


Loop
Unrolled?
Yes
Automatically unrolled?
Fully unrolled?
Partially unrolled?
#pragma unroll implemented?
No
Pipelined?
Yes
What's the Initiation Interval (launch frequency of new iteration)?
Are there dependency preventing optimal II?
No
Reason for serial execution?

Loop Unrolling

Loop unrolling: Replicate hardware to execute multiple loop iterations at once

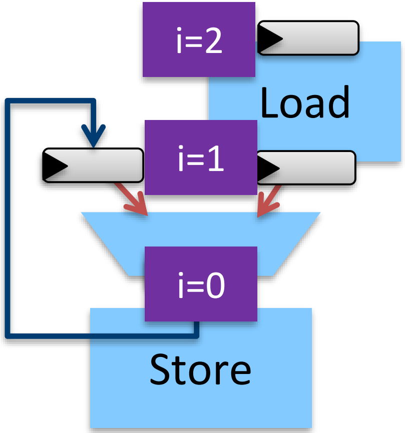
- Simple loops unrolled by the compiler automatically
- User may use `#pragma unroll` to control loop unrolling
- Dependencies resolved through scheduling of operations



Loop-Pipelining and Dependencies

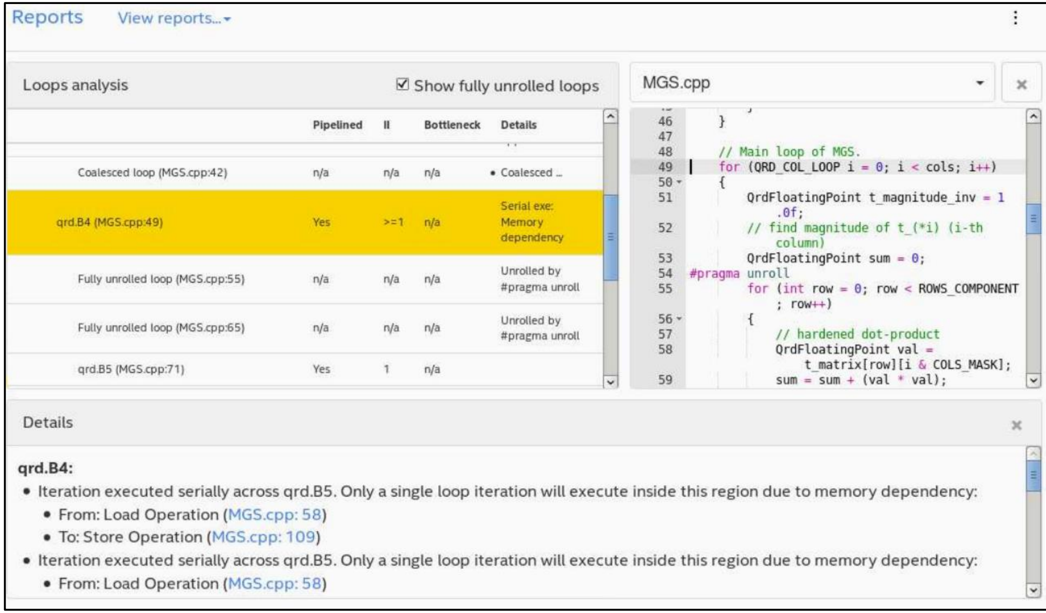
```
for (int i=1; i < n; i++) {
     c[i] = c[i-1] + b[i];
}
```

- Execute next iteration as soon as possible
- Dependencies can resolved by the compiler
    - Values transferred between loop iterations with FPGA resources



Loop Pipeline Analysis

- Automatically Generated
- Reports status of loop pipelining
- Displays dependency information



☑ Show fully unrolled loops

	Pipelined	II	Bottleneck	Details
Coalesced loop (MGS.cpp:42)	n/a	n/a	n/a	• Coalesced ...
qrd.B4 (MGS.cpp:49)	Yes	>=1	n/a	Serial exe: Memory dependency
Fully unrolled loop (MGS.cpp:55)	n/a	n/a	n/a	Unrolled by #pragma unroll
Fully unrolled loop (MGS.cpp:65)	n/a	n/a	n/a	Unrolled by #pragma unroll
qrd.B5 (MGS.cpp:71)	Yes	1	n/a	

```
    }

    // Main loop of MGS.
    for (QRD_COL_LOOP i = 0; i < cols; i++)
    {
        QrdFloatingPoint t_magnitude_inv = 1
            .0f;
        // find magnitude of t_(*i) (i-th
            column)
        QrdFloatingPoint sum = 0;
#pragma unroll
        for (int row = 0; row < ROWS_COMPONENT
            ; row++)
        {
            // hardened dot-product
            QrdFloatingPoint val =
                t_matrix[row][i & COLS_MASK];
            sum = sum + (val * val);
```

qrd.B4:

- Iteration executed serially across qrd.B5. Only a single loop iteration will execute inside this region due to memory dependency:
  - From: Load Operation (MGS.cpp: 58)
  - To: Store Operation (MGS.cpp: 109)
- Iteration executed serially across qrd.B5. Only a single loop iteration will execute inside this region due to memory dependency:
  - From: Load Operation (MGS.cpp: 58)

- Part of HTML Report
  - `<.prj folder>\reports\report.html`

Loop Pipelining Optimization Report

Reports shows pipeline status of each loop

- Minimizing II is the key to loop pipelining optimization
- Report shows
  - If loops are pipelined
    - Reason given if loop not pipelined
  - Initiation interval of pipelined loops
    - If II>1, shows operations that contributes to loop-carried dependency
      - Data computation or memory dependencies
      - Dependencies increases II

Arbitrary Precision Datatypes

- Algorithmic C (AC) datatypes
  - From Mentor Graphics under the Apache License
  - User Guide shipped with the HLS tool
    - `<path_to_HLS_installation>/include/ref/ac_datatypes_ref.pdf`
- Templated classes that allows instantiation of **arbitrary sized integers** and **arbitrary precision fixed-point** datatypes
- `ac_int` and `ac_fixed` are supported by the Intel® HLS Compiler
- Two implementations shipped with the Intel® HLS Compiler
  - `ref/ac_int.h, ac_fixed.h`: Mentor Graphics reference implementation
  - `HLS/ac_int.h, ac_fixed.h`: Intel-optimized implementation for HLS

Local Component Memories

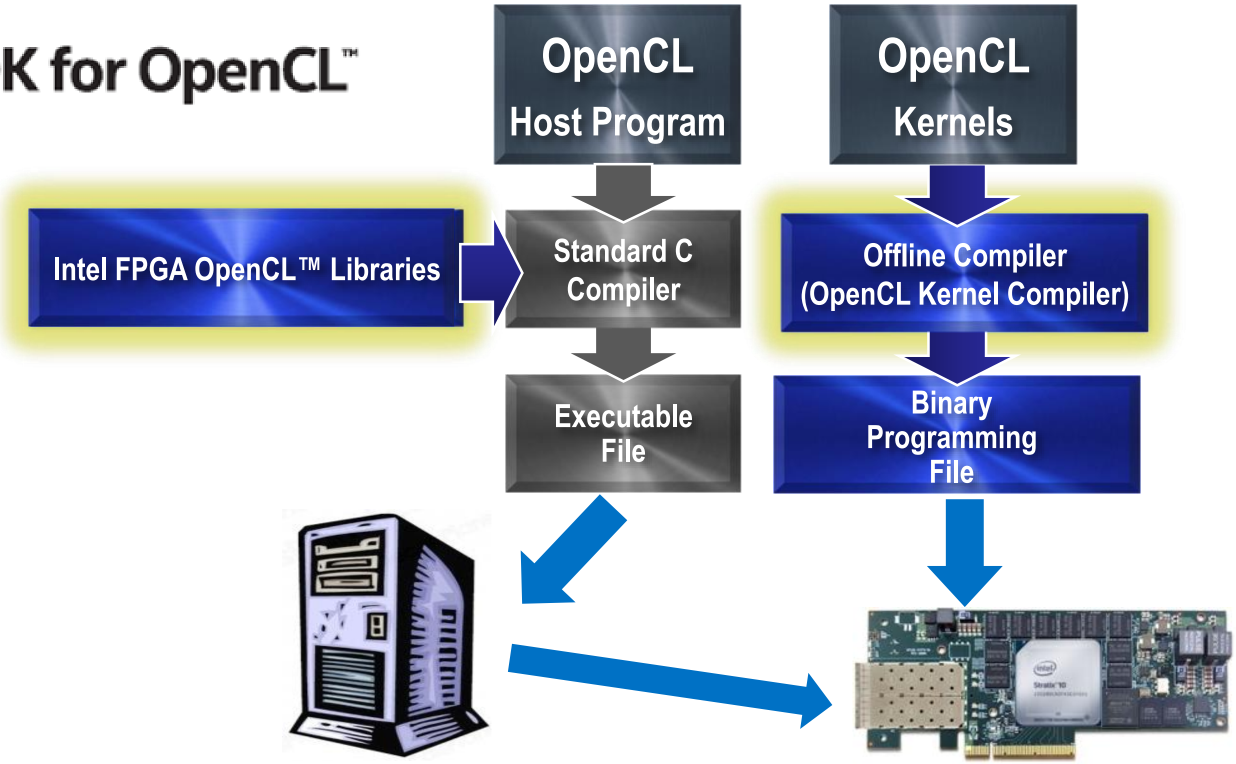
- Local component memories implemented with on-chip RAM resources
  - Much better performance than off-chip system memories
- Local memory system is customized to your application at compile time
  - Dependent on data type and usage
  - Banking configuration (number of banks, width), and interconnect customized to minimize contention
  - Big advantage over fixed-architecture accelerators
- Note: local memory cannot be dynamically allocated inside the component

Agenda

- High-level synthesis with the Intel® HLS Compiler
- **Intel® FPGA SDK for OpenCL™**
- Acceleration Stack for Intel® Xeon CPUs and FPGAs
- Deep Learning Inference on FPGAs

Intel® FPGA SDK for OpenCL™ Usage

Intel® FPGA SDK for OpenCL™



OpenCL Host Program

OpenCL Kernels

Intel FPGA OpenCL™ Libraries

Standard C Compiler

Offline Compiler (OpenCL Kernel Compiler)

Executable File

Binary Programming File

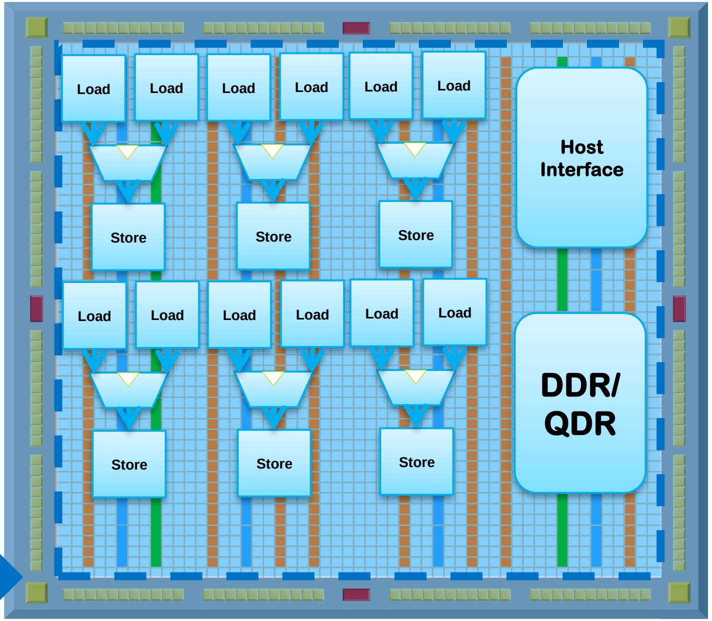
*OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission of Khronos

Compiling OpenCL™ Kernel to Intel® FPGA

- Using similar concepts and optimization techniques as HLS

```
__kernel void increment ( __global float *a, float
c, int N)
{
      int i;
      for (i = 0; i < N; i++)
            a[i] = a[i] + c;
}
```

```
aoc –board=a10_ref
```

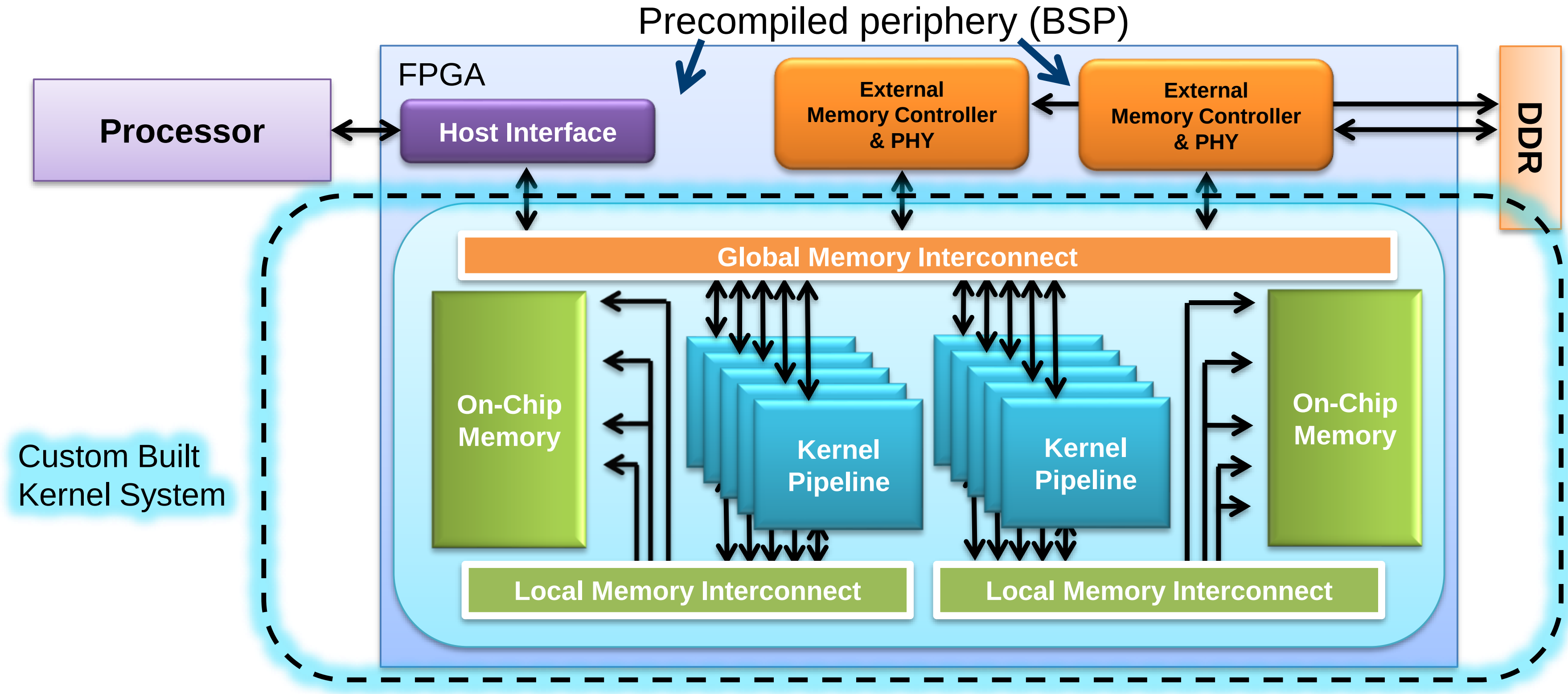


*OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission of Khronos

Benefits of OpenCL™ on FPGAs

- For software developers
- Faster software-centric development flow
  - C-based design leads to shorter architectural exploration and development time
- Obtain performance and power advantages of an FPGA
- Portability between different HW accelerators (CPU, GPGPU, FPGA, etc)

FPGA Architecture for OpenCL™ Implementation


Precompiled periphery (BSP)
FPGA
Processor
Host Interface
External Memory Controller & PHY
External Memory Controller & PHY
DDR
Global Memory Interconnect
On-Chip Memory
Kernel Pipeline
Kernel Pipeline
On-Chip Memory
Local Memory Interconnect
Local Memory Interconnect
Custom Built Kernel System

*OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission of Khronos

aoc Output Files

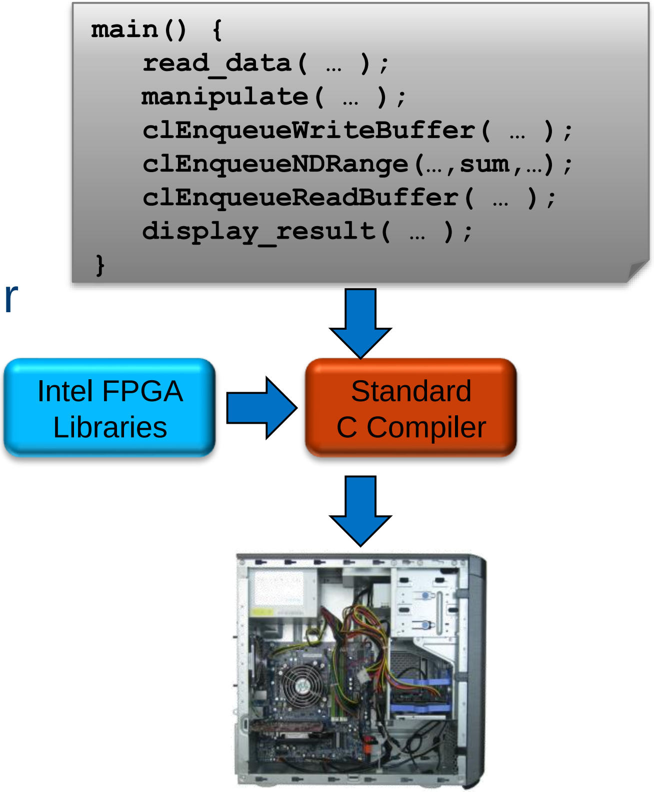
- <kernel file>.aoco
  - Intermediate object file representing the created hardware system
- <kernel file>.aocx
  - Kernel executable file used to program FPGA
- Inside <kernel file> folder
  - <kernel file folder>\reports\report.html
    - Interactive HTML report
    - Static report showing optimization, detailed area, and architectural information
  - <kernel file>.log compilation log
  - Intel® Quartus® Prime software generated source and report files

Example Host Program

```
void main()
{    ...
   // 1. Create then build program
   c::Program myprogram = (… mybinaries_of_aocx…);
   err = myprogram.build(mydevlist);

   // 2. Create kernels from the program
   cl::Kernel mykernel(myprogram, ”increment”, &err);

   // 3. Tansfer buffers on/to device
   err=myqueue.enqueueWriteBuffer(a_device, CL_FALSE, 0, size, a_host);
   …
   // 4. Set up the kernel argument list
   err = mykernel.setArg(0, buffer);
   // 5. Launch the kernel
   err = myqueue.enqueueTask(mykernel);

   // 6. Transfer result buffer back
   err = myqueue.enqueueReadBuffer(a_device, CL_TRUE, 0, NUM_ELEMENTS*sizeof(cl_float), a_host);
}
```

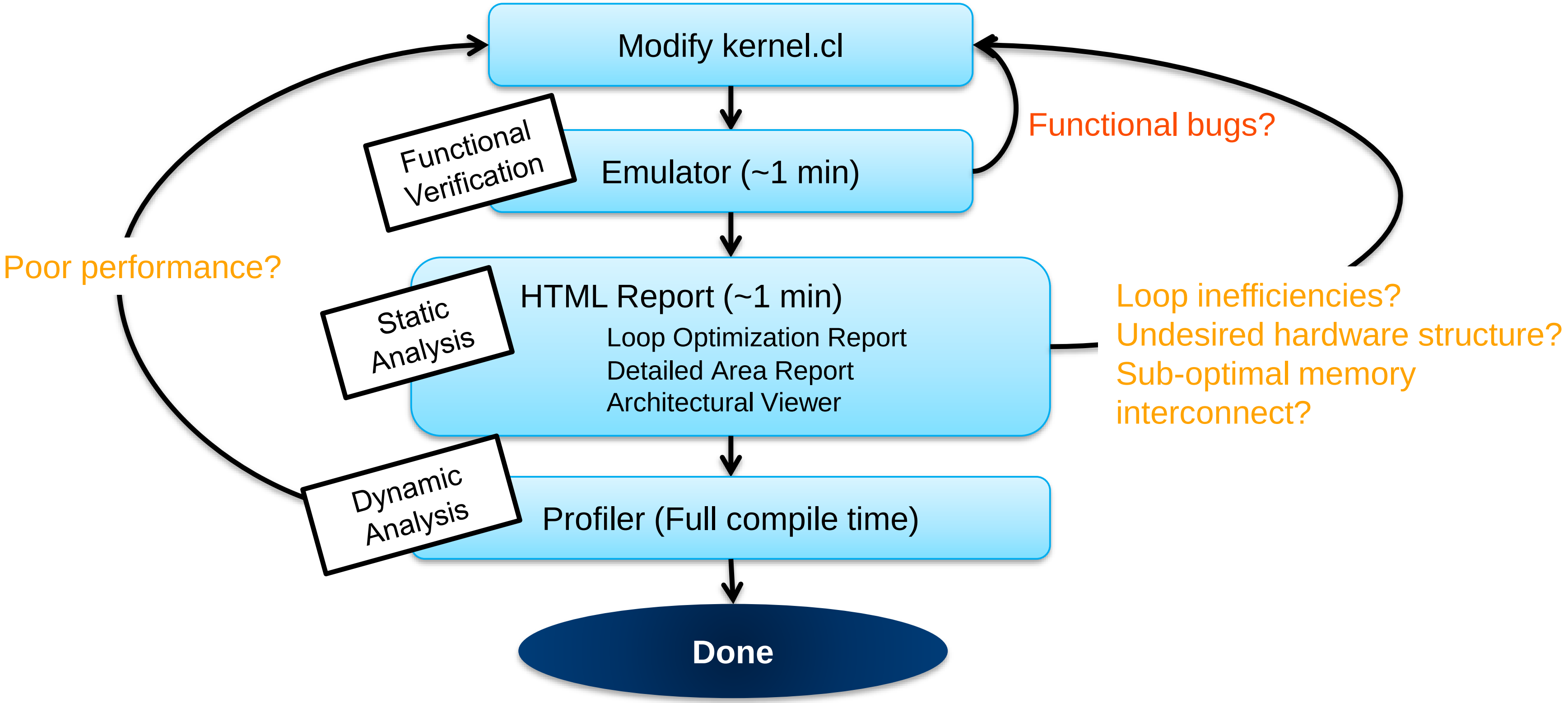
Compiling the Host Program

- Include `CL/opencl.h` or `CL/cl.hpp`
- Use a conventional C compiler (Visual Studio*/GCC)
- Add `$INTELFPGASDKROOT/host/include` to your file search path
  - Recommended to use `aocl compile-config`
- Link to Intel® FPGA OpenCL™ libraries
  - Link to libraries located in the `$INTELFPGASDKROOT/host/<OS>/lib` directory
    - Recommended to use `aocl link-config`



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

Kernel Development Flow and Tools



Modify kernel.cl

Emulator (~1 min)

Functional Verification

Functional bugs?

HTML Report (~1 min)

- Loop Optimization Report
- Detailed Area Report
- Architectural Viewer

Static Analysis

Loop inefficiencies?
Undesired hardware structure?
Sub-optimal memory interconnect?

Poor performance?

Profiler (Full compile time)

Dynamic Analysis

Done

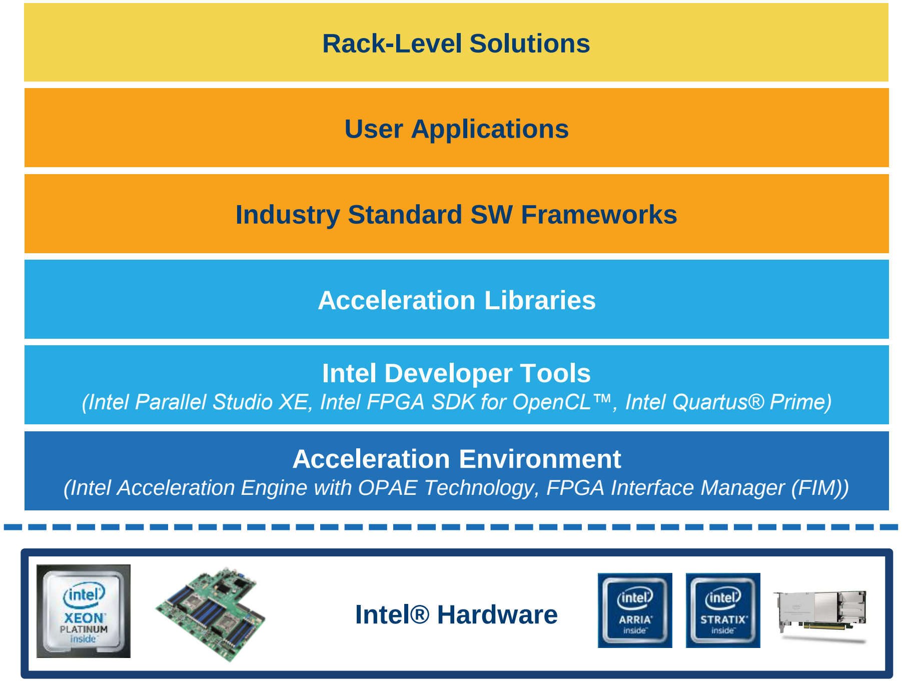
Intel® FPGA-Specific Features

- Single Work-Item Execution
- Channels
- Controlling Hardware Generation with Attributes
  - Autorun Kernels, Vectorization Factor, Compute Unit replication, etc…
- Libraries (Calling custom RTL)
- SoC Platforms
- Shared Virtual Memory
- Custom Boards

Agenda

- High-level synthesis with the Intel® HLS Compiler
- Intel® FPGA SDK for OpenCL™
- **Acceleration Stack for Intel® Xeon CPUs and FPGAs**
- Deep Learning Inference on FPGAs

Acceleration Stack for Intel® Xeon® CPU with FPGAs



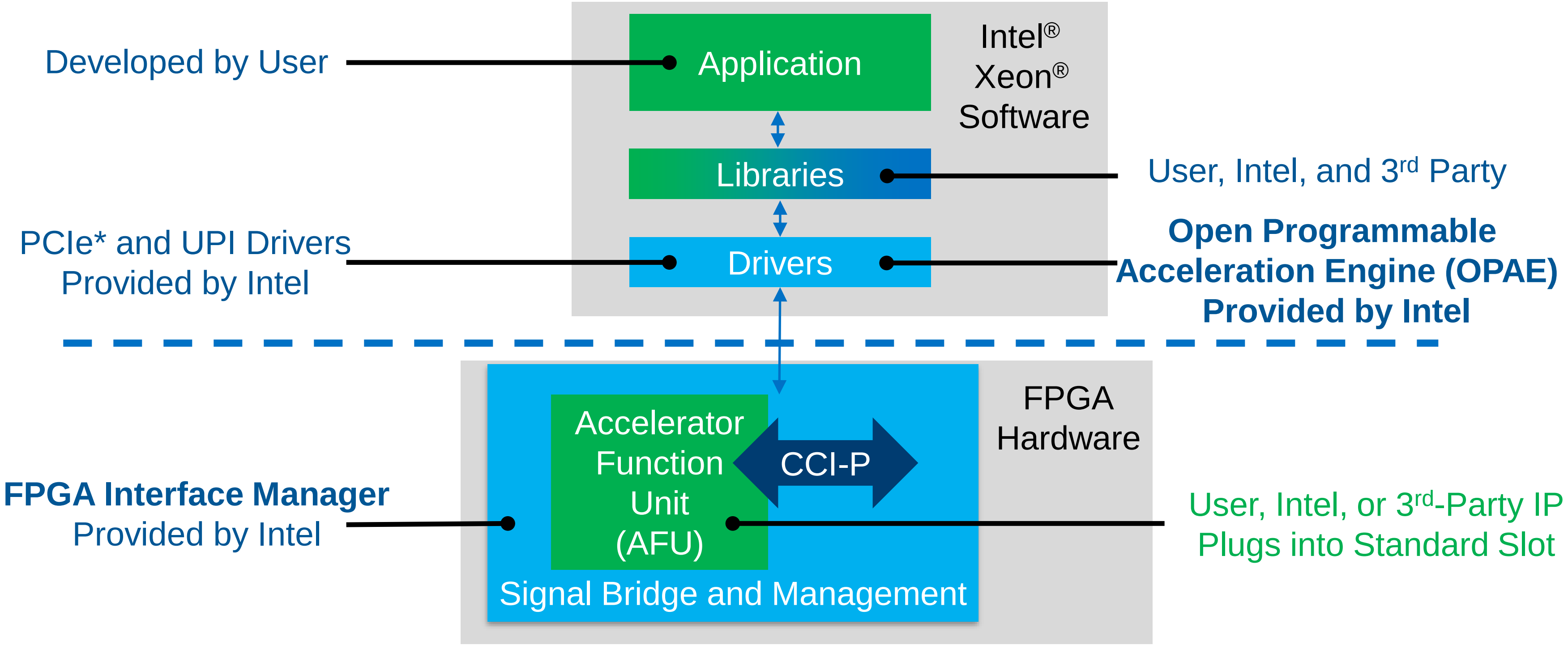
Dynamically Allocate Intel® FPGAs for Workload Optimization	**Rack-Level Solutions**	intel Rack Scale Design openstack
Simplified Application Development	**User Applications**	Deep Learning, Networking, Genomics, etc.
Leverage Common Frameworks	**Industry Standard SW Frameworks**	Caffe theano gatk
Fast-Track Your Performance	**Acceleration Libraries**	LZ4, Snappy, etc. zlib DPDK
Workload Optimization with Less Effort	**Intel Developer Tools** *(Intel Parallel Studio XE, Intel FPGA SDK for OpenCL™, Intel Quartus® Prime)*	PARALLEL STUDIO XE OpenCL Quartus Prime Design Suite
Common Developer Interface for Intel FPGA Data Center Products	**Acceleration Environment** *(Intel Acceleration Engine with OPAE Technology, FPGA Interface Manager (FIM))*	

Intel® Hardware

Intel® delivers a system-optimized solution stack for your data center workloads

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos
Some names pending final approval and may change in the future.
Logos and names provided for illustrative purposes only. Current availability may be different.

Intel® Xeon® with FPGA **Virtualization Framework**



Developed by User

Application

Libraries

Drivers

Intel® Xeon® Software

User, Intel, and 3rd Party

PCIe* and UPI Drivers
Provided by Intel

Open Programmable Acceleration Engine (OPAE)
Provided by Intel

FPGA Hardware

Accelerator Function Unit (AFU)

CCI-P

Signal Bridge and Management

FPGA Interface Manager
Provided by Intel

User, Intel, or 3rd-Party IP
Plugs into Standard Slot

Simplifies the use of FPGAs in virtualized cloud environments

Intel® Programmable Acceleration Card with Intel Arria® 10 GX FPGA

Intel’s 1st versatile FPGA PCIe acceleration card
that offers inline & look-aside acceleration for workloads requiring up to 45W

1st acceleration card to offer the Acceleration Stack for Intel Xeon CPU with FPGAs enabling broader FPGA adoption in data center

[Intel Programmable Acceleration Card with Intel Arria 10 GX FPGA]()

Open Programmable Acceleration Engine (OPAE)

Consistent API across product generations and platforms

- Abstraction for hardware specific FPGA resource details

Designed for minimal software overhead and latency

- Lightweight user-space library *(libfpga)*

Open ecosystem for industry and developer community

- License: FPGA API (BSD), FPGA driver (GPLv2)

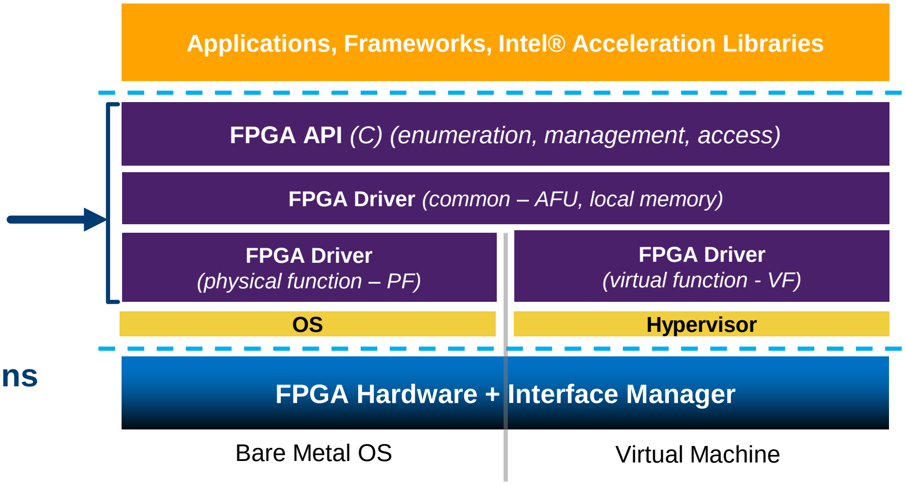
FPGA driver being upstreamed into Linux kernel

Supports both virtual machines and bare metal platforms

Faster development and debugging of Accelerator Functions with the included AFU Simulation Environment (ASE)

Includes guides, command-line utilities and sample code

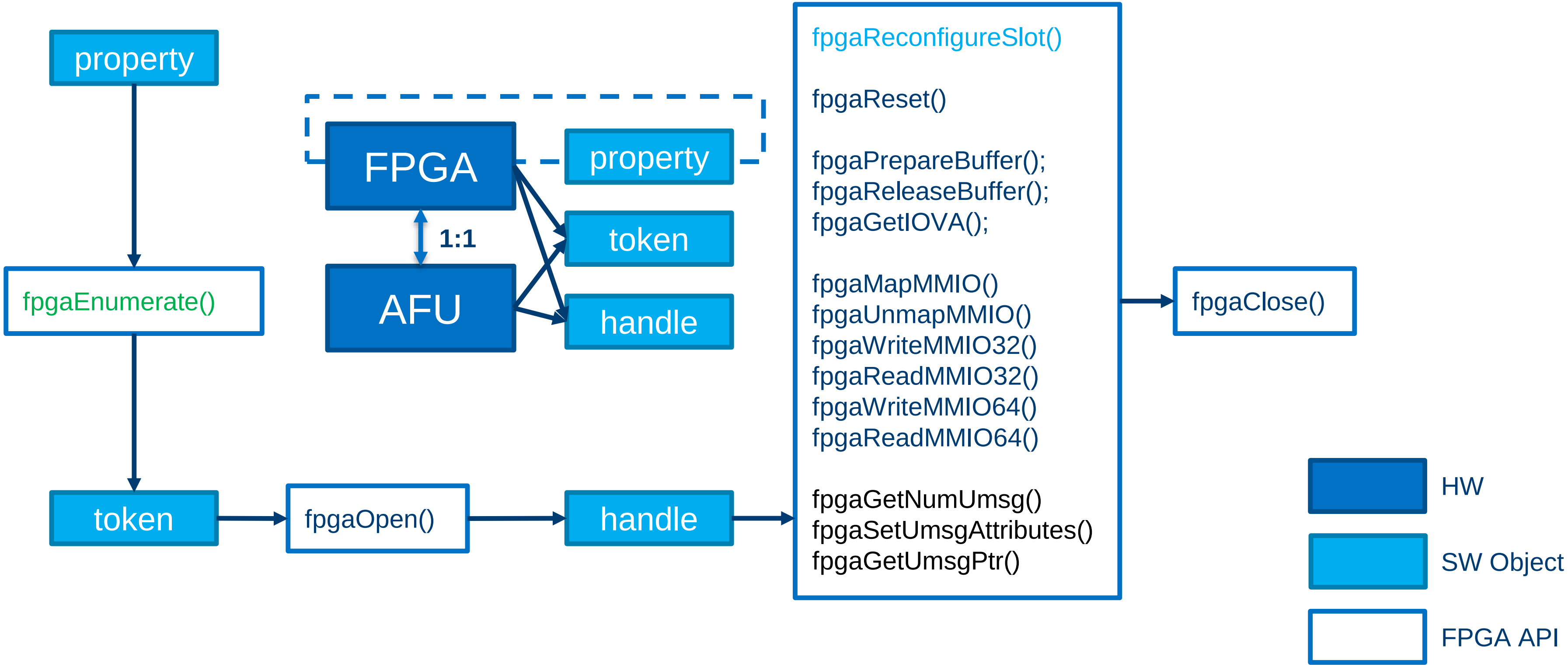
Simplified FPGA Programming Model
for Application Developers



Applications, Frameworks, Intel® Acceleration Libraries
FPGA API (C) (enumeration, management, access)
FPGA Driver (common – AFU, local memory)
FPGA Driver (physical function – PF)
FPGA Driver (virtual function - VF)
OS
Hypervisor
FPGA Hardware + Interface Manager
Bare Metal OS
Virtual Machine

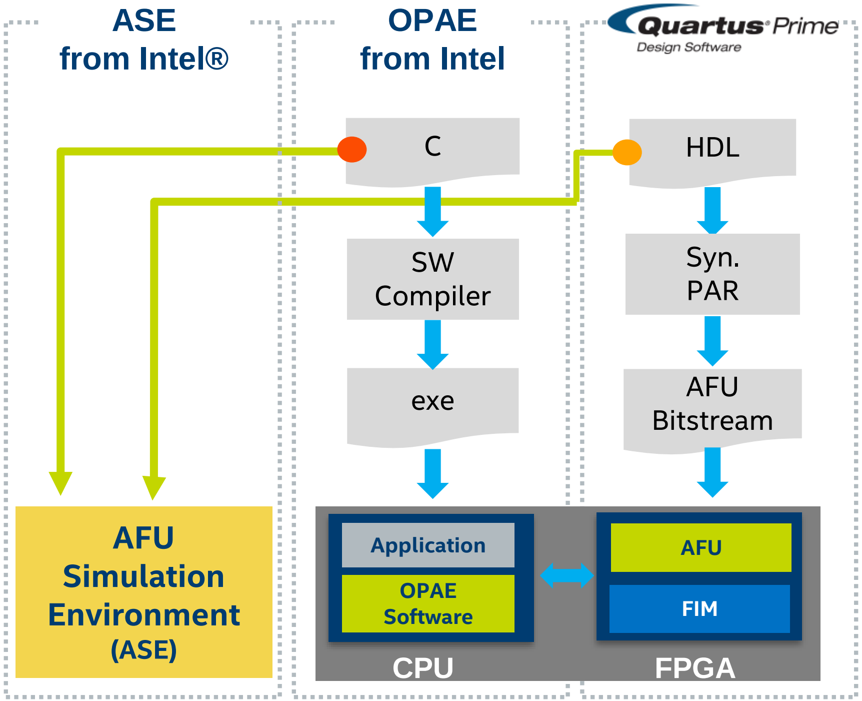
Start developing for Intel FPGAs with OPAE today: http://01.org/OPAE

OPAE FPGA API – Enumerate, Manage & Access

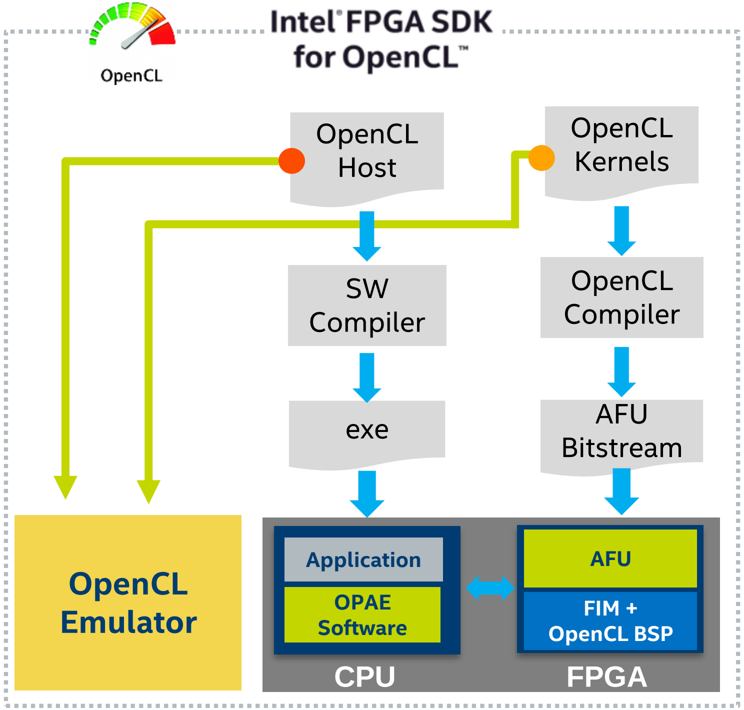


Two Development Approaches

HDL Programming



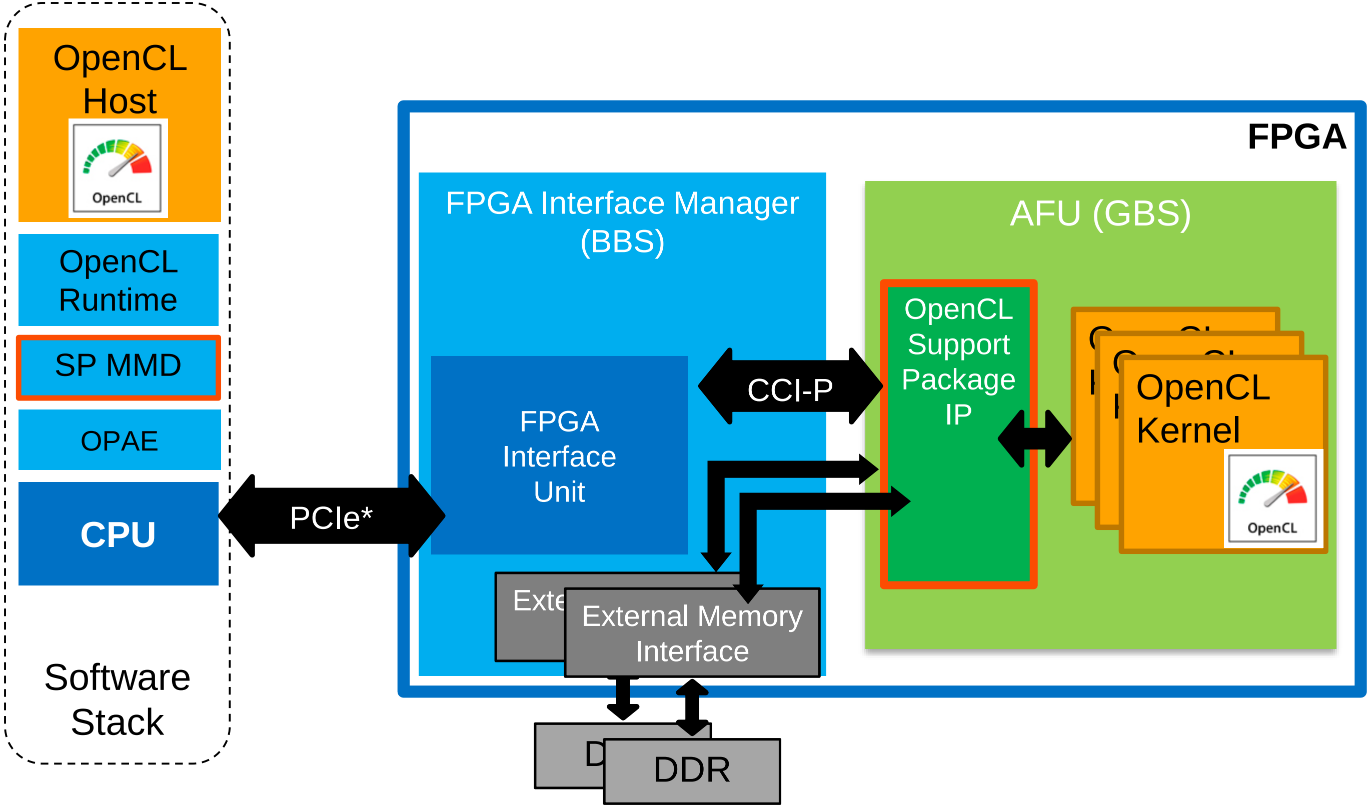
OpenCL* Programming



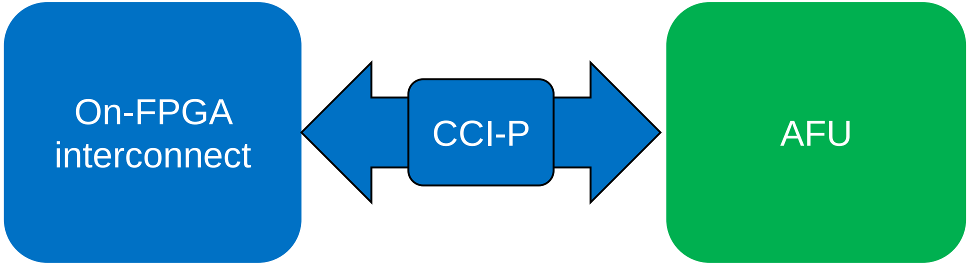
OpenCL™ Flow

- Usage no different from traditional OpenCL™ flow
  - C based development and optimization flow to create AFUs and Host Application
  - Standard OpenCL FPGA application using the Intel® FPGA SDK for OpenCL
    - FPGA OpenCL debug and profiling tools supported
- The Acceleration Stack abstracted away from user
  - OPAE part of the Host Run-Time

OpenCL™ Support Package for Intel® PAC

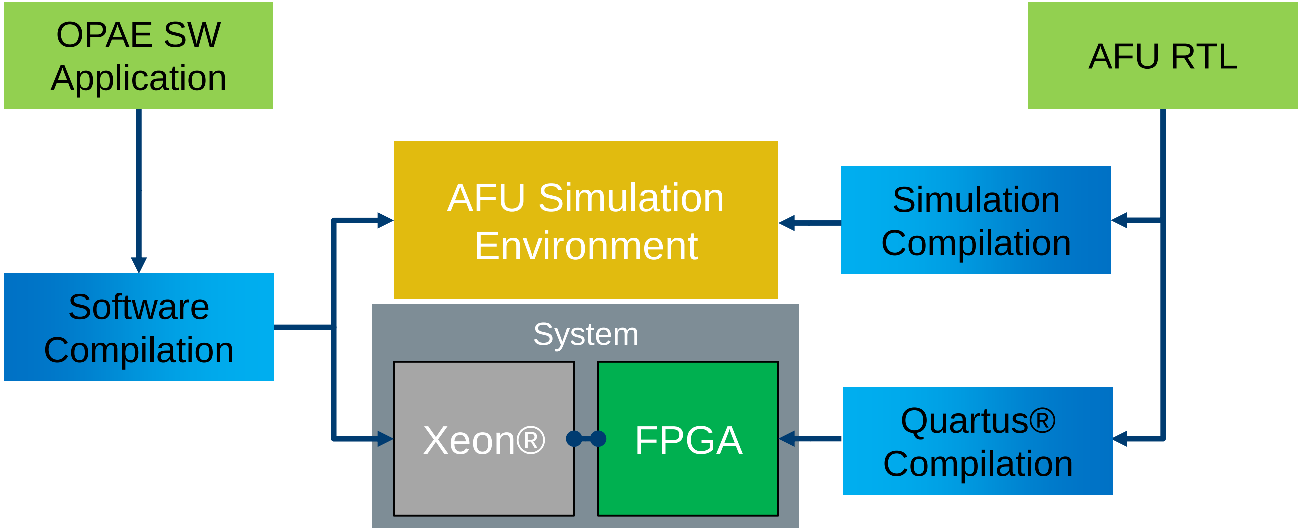


RTL AFU



- Develop RTL AFU with standard FPGA development tools
- Interface with the acceleration stack through Core Cache Interconnect (CCI-P)
  - Provides a base platform memory interface
    - Simple request/response interface (Simple Read/Write)
    - Physical addresses
    - No order guarantees
  - These minimal requirements satisfy major classes of algorithms, e.g.:
    - Double buffered kernels that read from and write to different buffers
    - Streaming kernels that read from one memory-mapped FIFO and write to another

RTL Flow

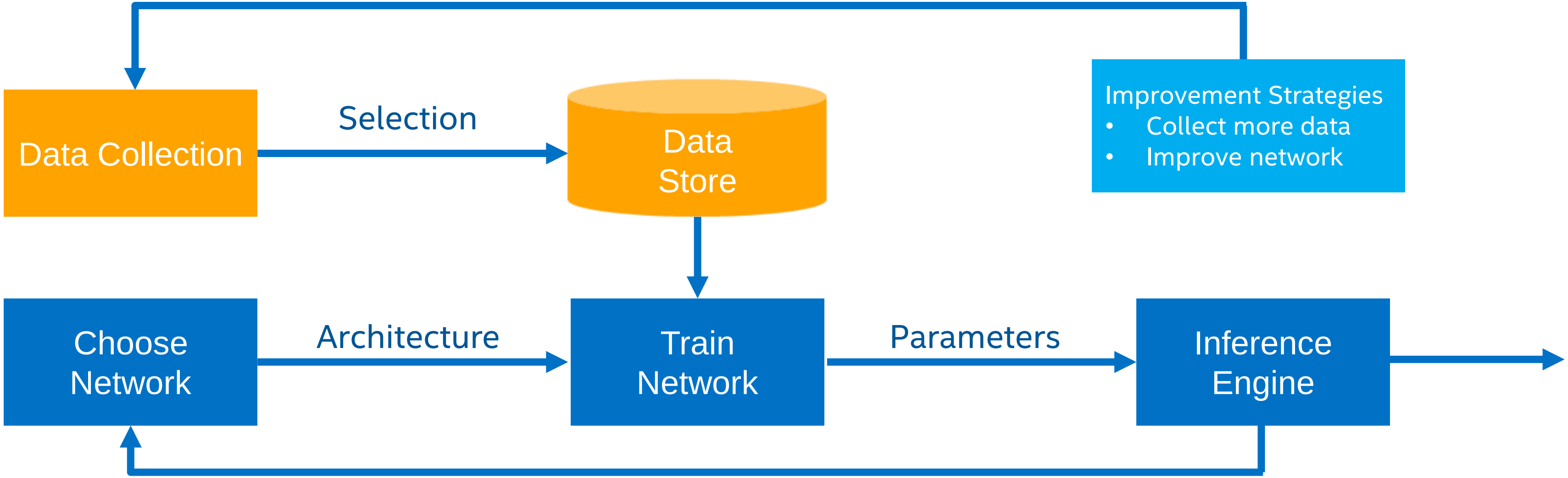


- AFU Simulation Environment (ASE) enables seamless portability to real HW
  - Allows fast verification of OPAE software together with AFU RTL without HW
    - SW Application loads ASE library and connects to RTL simulation
  - For execution on HW, application loads Runtime library and RTL is compiled by Intel® Quartus into FPGA bitstream

Agenda

- High-level synthesis with the Intel® HLS Compiler
- Intel® FPGA SDK for OpenCL™
- Acceleration Stack for Intel® Xeon CPUs and FPGAs
- **Deep Learning Inference on FPGAs**

Design Flow with Machine Learning



Choose Network topology

- Use framework (e.g. Caffe, Tensor Flow)

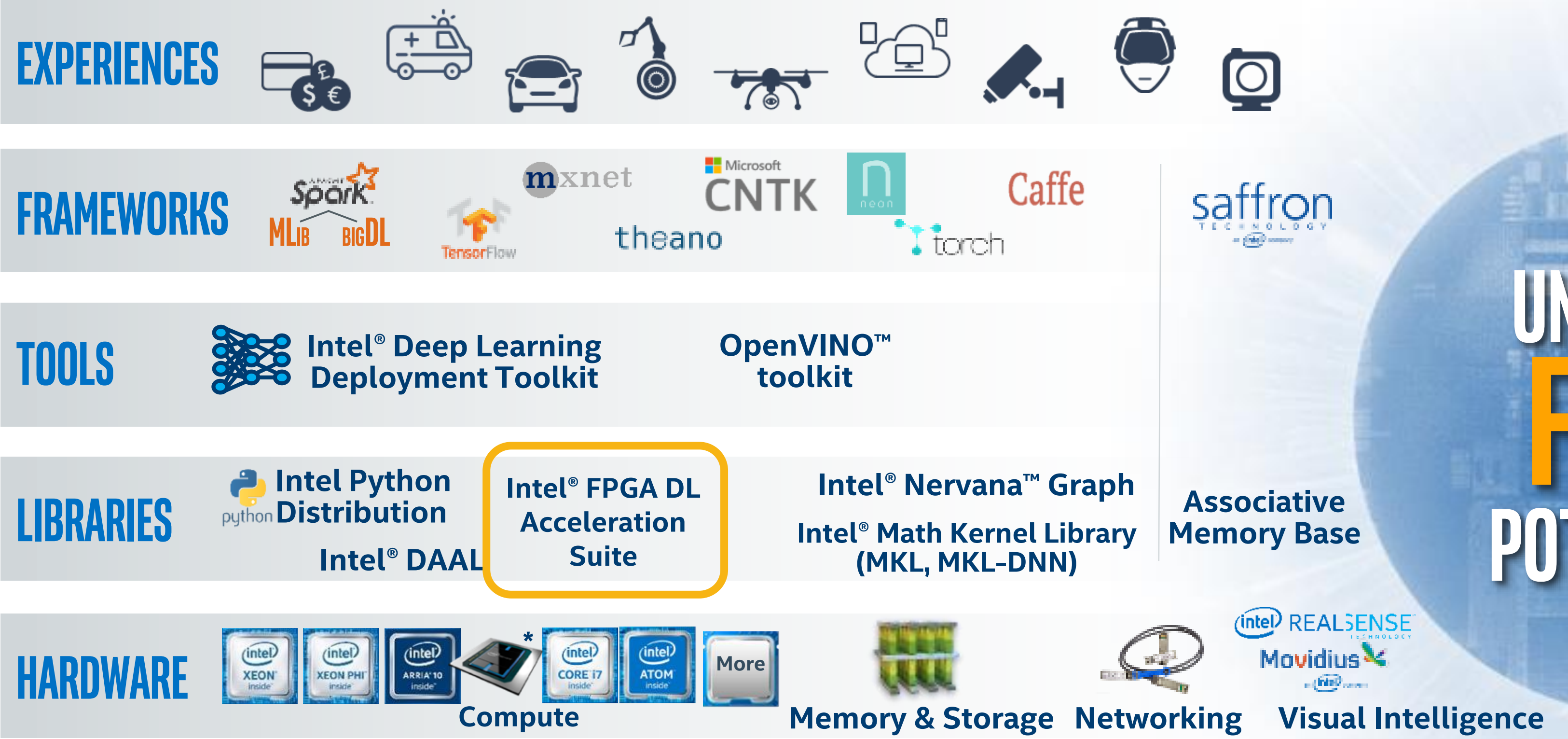
Train Network

- A high-performance computing (HPC) workload from large dataset
- Weeks to months process

Inference Engine (FPGA Focus)

- Implementation of the neural network performing real-time inferencing

INTEL® AI PORTFOLIO



EXPERIENCES

FRAMEWORKS

TOOLS

Intel® Deep Learning Deployment Toolkit

OpenVINO™ toolkit

LIBRARIES

Intel Python Distribution

Intel® DAAL

Intel® FPGA DL Acceleration Suite

Intel® Nervana™ Graph

Intel® Math Kernel Library (MKL, MKL-DNN)

Associative Memory Base

HARDWARE

Compute

Memory & Storage

Networking

Visual Intelligence

UNLEASH FULL POTENTIAL

Solving Machine Learning Challenges with FPGA

EASE-OF-USE

SOFTWARE ABSTRACTION,
PLATFORMS & LIBRARIES

Intel FPGA solutions enable software-defined programming of customized machine learning accelerator libraries.

REAL-TIME

DETERMINISTIC
LOW LATENCY

Intel FPGA hardware implements a deterministic low latency data path unlike any other competing compute device.

FLEXIBILITY

CUSTOMIZABLE HARDWARE
FOR NEXT GEN DNN ARCHITECTURES

Intel FPGAs can be customized to enable advances in machine learning algorithms.

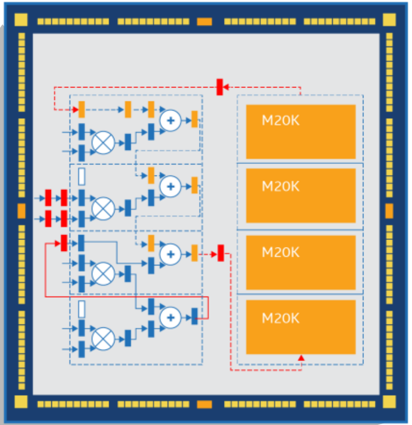
Why Intel® FPGAs for Machine Learning?

Convolutional Neural Networks are Compute Intensive

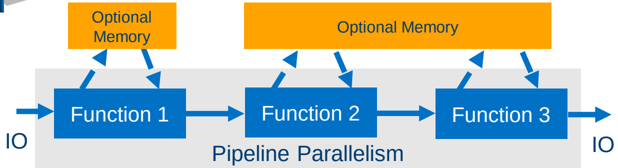
SOLUTION BRIEF

Intel® FPGAs

Accelerating Real-Time Video Analytics with Artificial Intelligence



Fine-grained & low latency between compute and memory



Optional Memory

Optional Memory

IO

Function 1

Function 2

Function 3

IO

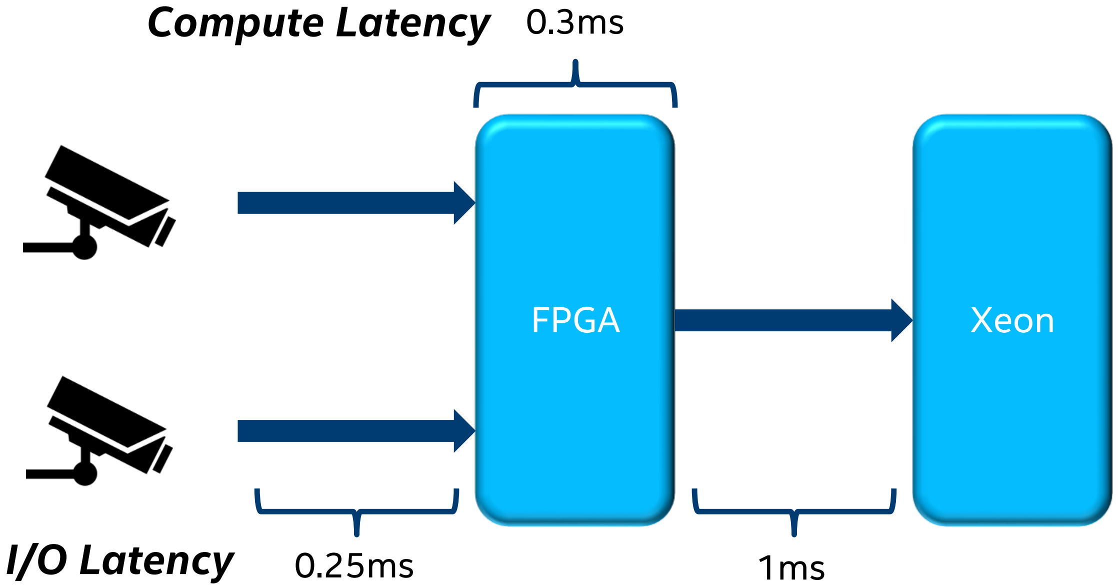
Pipeline Parallelism

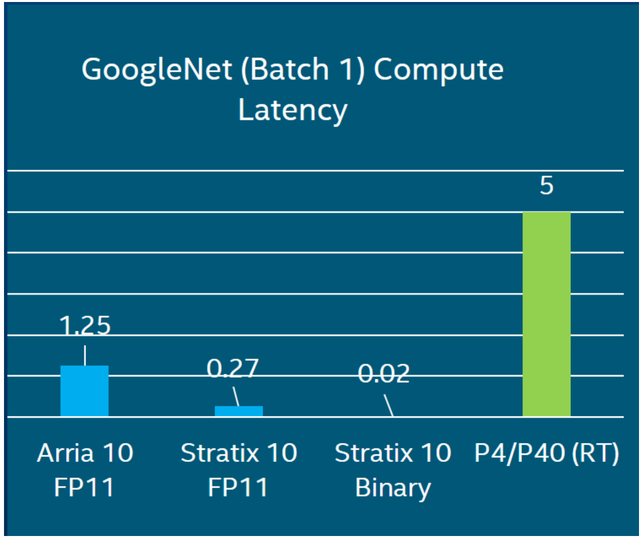
Feature	Benefit
Highly parallel architecture	Facilitates efficient low-batch video stream processing and reduces latency
Configurable Distributed Floating Point DSP Blocks	FP32 9Tflops, FP16, FP11 Accelerates computation by tuning compute performance
Tightly coupled high-bandwidth memory	>50TB/s on chip SRAM bandwidth, random access, reduces latency, minimizes external memory access
Programmable Data Path	Reduces unnecessary data movement, improving latency and efficiency
Configurability	Support for variable precision (trade-off throughput and accuracy). Future proof designs, and system connectivity

FPGAs Provide Deterministic System Latency

FPGAs can leveraging parallelism across the entire chip reducing the compute time to a fraction

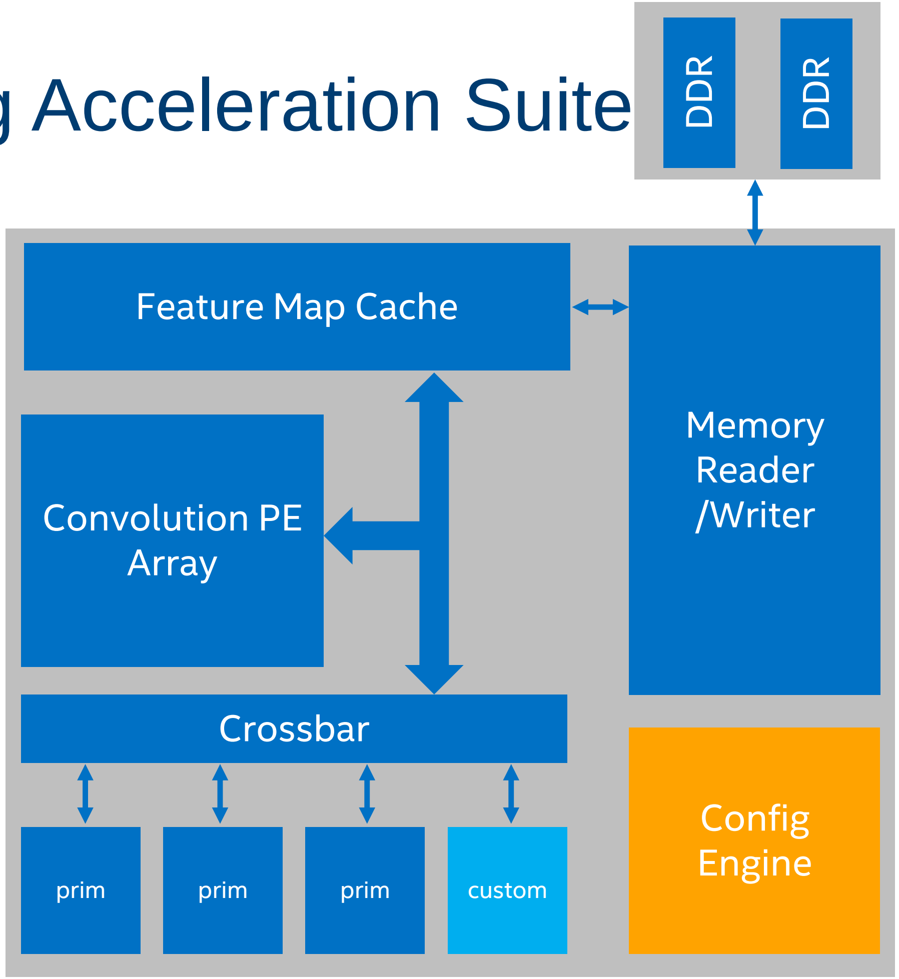
System Latency = I/O Latency + Compute Latency



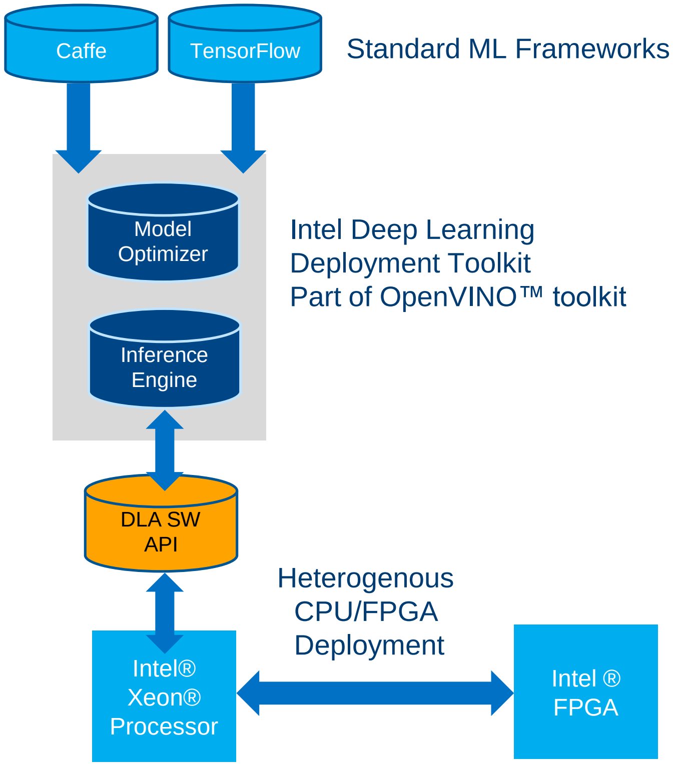


Intel® FPGA Deep Learning Acceleration Suite

- CNN acceleration engine for common topologies executed in a graph loop architecture
  - AlexNet, GoogleNet, LeNet, SqueezeNet, VGG16, ResNet, Yolo, SSD, LSTM…
- Software Deployment
  - No FPGA compile required
  - Run-time reconfigurable
- Customized Hardware Development
  - Custom architecture creation w/ parameters
  - Custom primitives using OpenCL™ flow

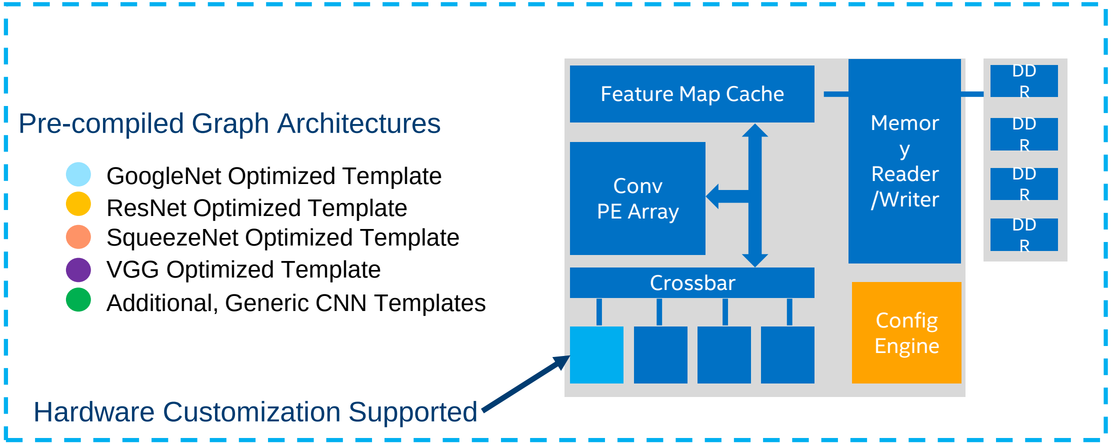


Intel® FPGA DLA Suite Usage



- Supports common software frameworks (Caffe, Tensorflow)
- Intel DL software stack provides graph optimizations
- Intel FPGA Deep Learning Acceleration Suite provides turn-key or customized CNN acceleration for common topologies

Optimized Acceleration Engine


Pre-compiled Graph Architectures

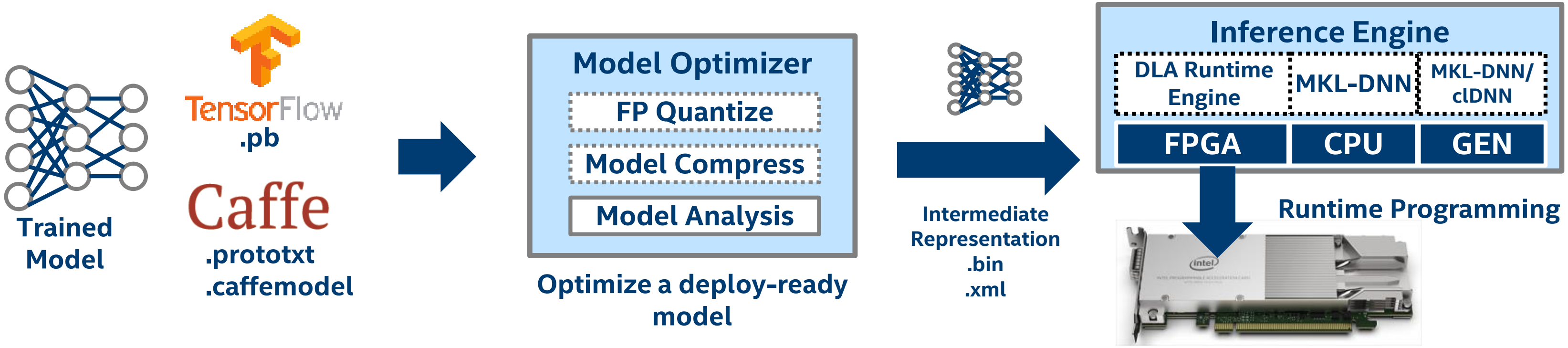
- ● GoogleNet Optimized Template
- ● ResNet Optimized Template
- ● SqueezeNet Optimized Template
- ● VGG Optimized Template
- ● Additional, Generic CNN Templates

Hardware Customization Supported

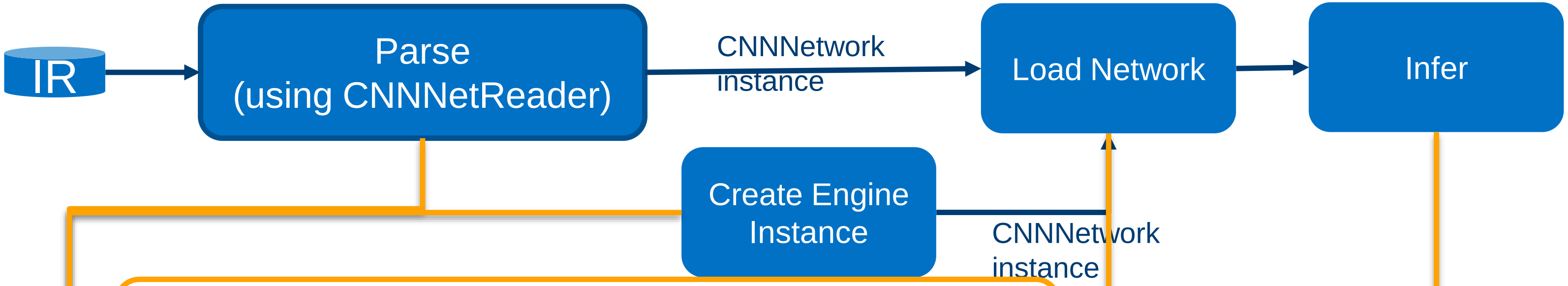
Mapping a Topology to the Architecture in FPGA

Using the Intel® DL Deployment Toolkit component of the OpenVINO™ toolkit to enable deployment of trained model on all Intel® architectures

- CPU, GPU, FPGA, …
- Optimize for best execution
- Enable users to validate and tune
- Easy-to-use runtime API across all devices

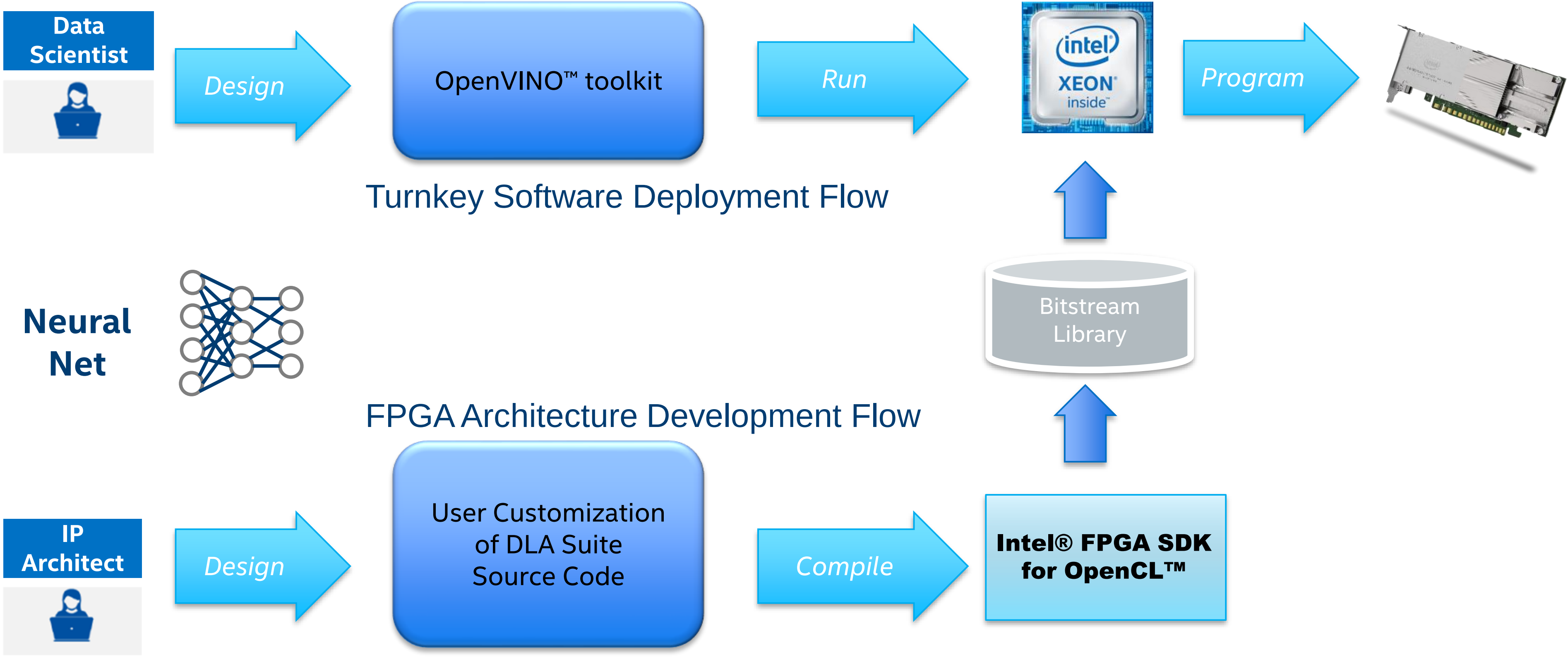


Using the Inference Engine API



```
auto netBuilder = new InferenceEngine::CNNNetReader();
netBuilder->ReadNetwork(“Model.xml”);
netBuilder->ReadWeights(“Model.bin")

auto enginePtr = new InferenceEngine::InferenceEnginePluginPtr(getSuitablePlugin(eFPGA));

enginePtr->LoadNetwork(*netBuilder->network, &resp);

InferenceEngine::TBlob<float> output;
InferenceEngine::SizeVector inputDims;
netBuilder->getInputDimentions(inputDims);
InferenceEngine::TBlob<short> input(inputDims);
input.allocate();
enginePtr->Infer(input, output, &resp);
```

User Flow



Data Scientist

Design

OpenVINO™ toolkit

Run

Program

Turnkey Software Deployment Flow

Bitstream Library

Neural Net

FPGA Architecture Development Flow

IP Architect

Design

User Customization of DLA Suite Source Code

Compile

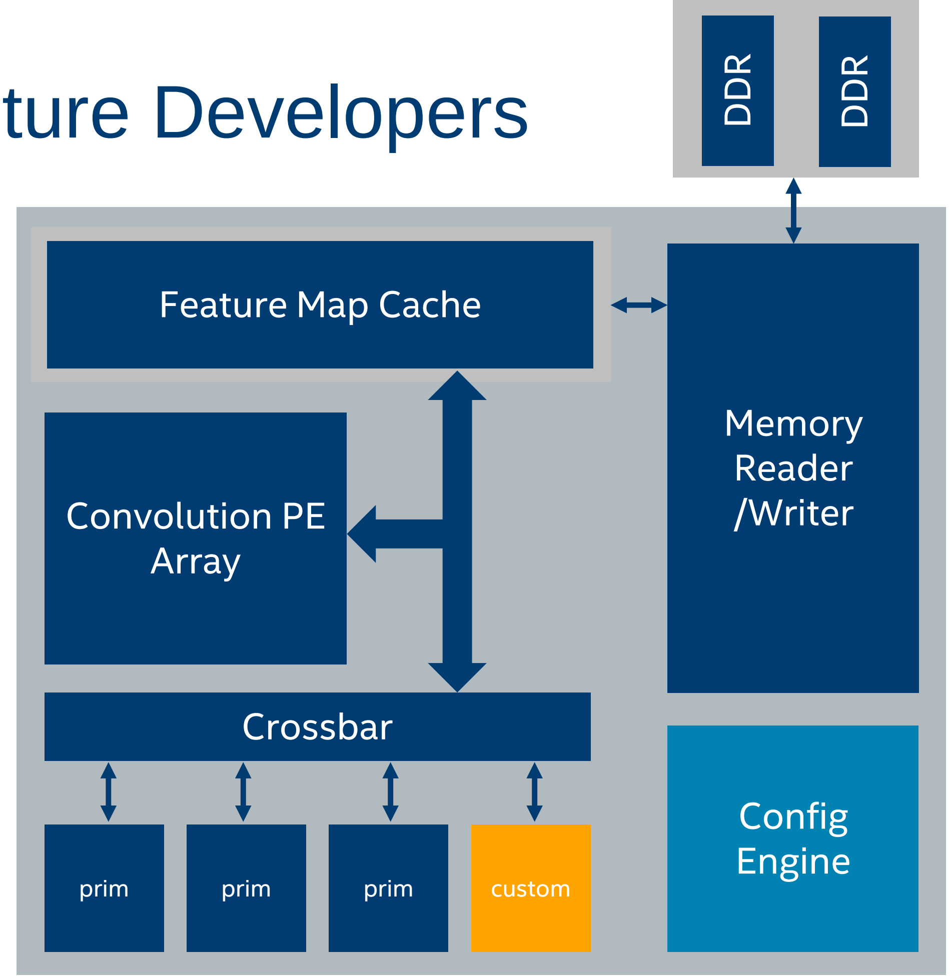
Intel® FPGA SDK for OpenCL™

Customization for Architecture Developers

Add a custom primitive into crossbar

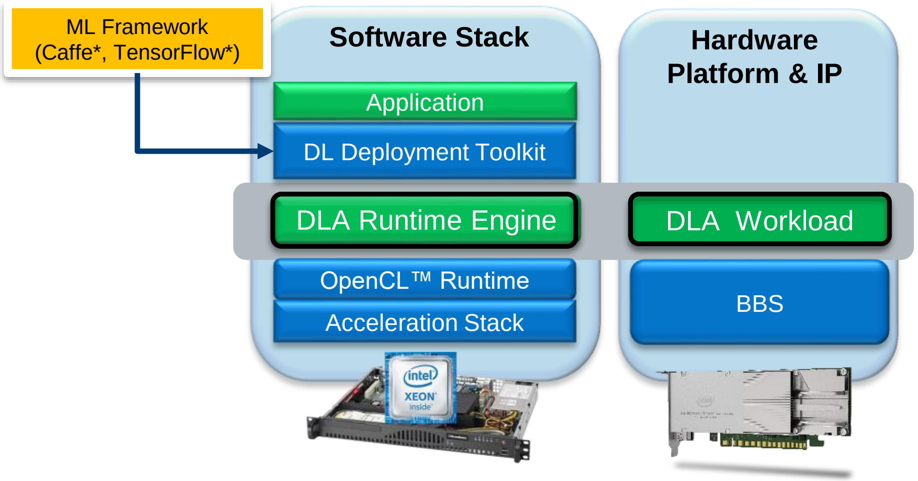
- Three primitive types supported:
  - Unary (ReLU, Tanh)
  - Binary (Eltwise Add, Mult)
  - Window (Pool, LRN, Norm)
  - Unary w/ coefficients
    - Scale/Dropout (a couple of coefficients per layer: coefficients loaded via layer config)
    - BatchNorm (dozen or more coefficients per layer: coefficients loaded via DDR)

custom



Machine Learning on Intel® FPGA Platform

Acceleration Stack Platform Solution



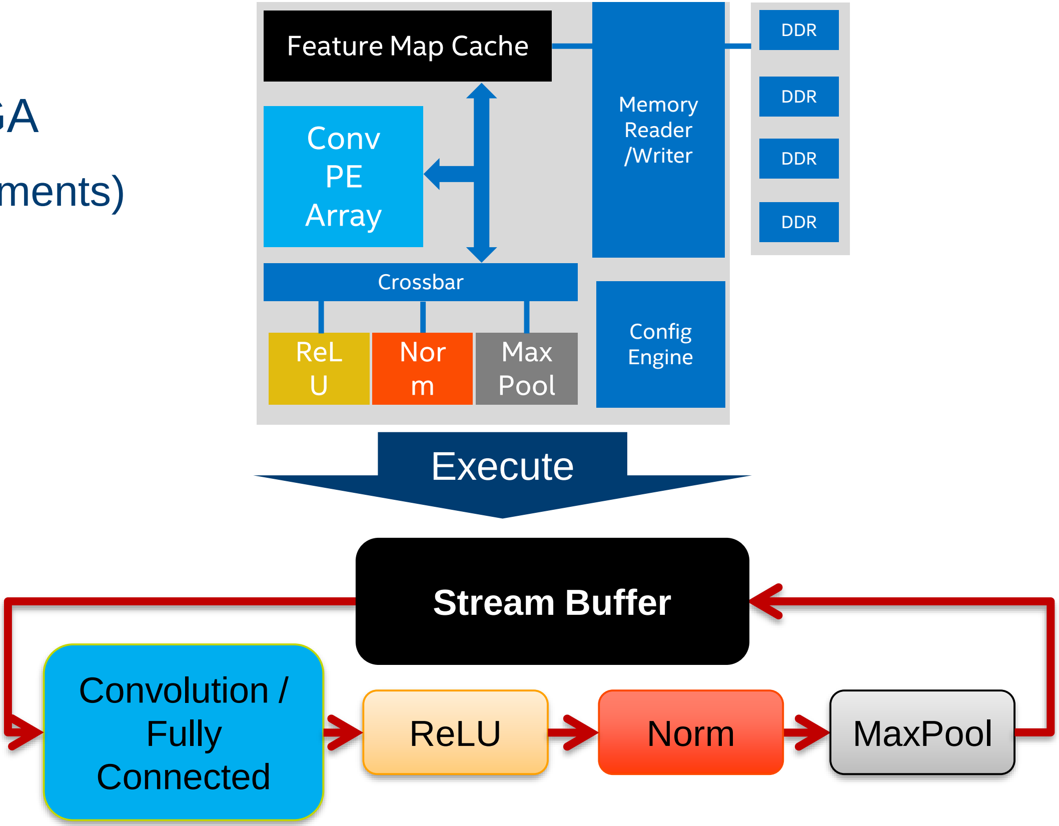
Intel® Xeon CPU

PAC Family Boards

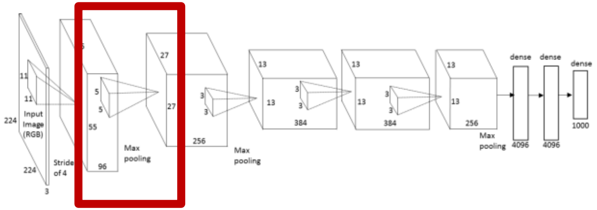
For more information on the Acceleration Stack for Intel® Xeon® CPU with FPGAs on the Intel® Programmable Acceleration Card, visit the Intel® FPGA Acceleration Hub

DLA Architecture: Built for Performance

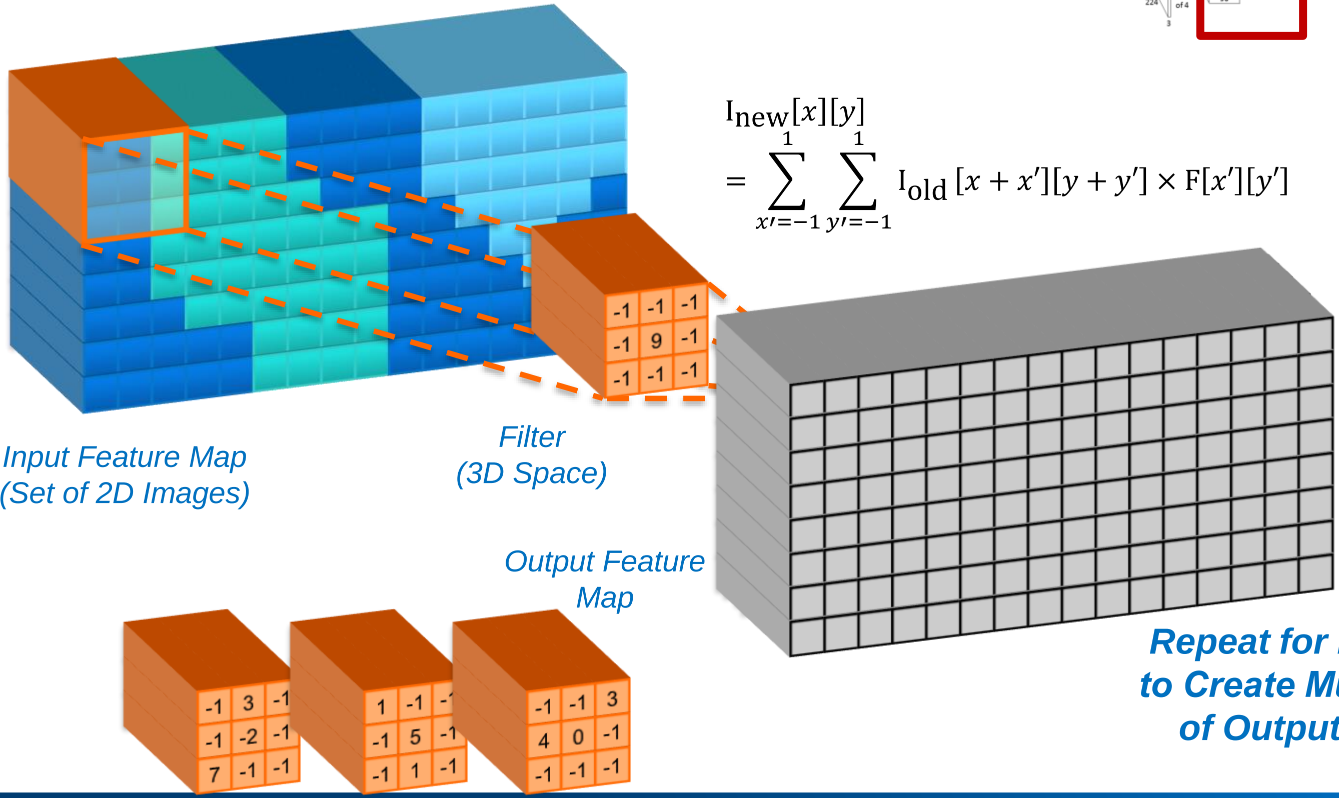
- Maximize Parallelism on the FPGA
  - Filter Parallelism (Processing Elements)
  - Input-Depth Parallelism
  - Winograd Transformation
  - Batching
  - Feature Stream Buffer
  - Filter Cache
- Choosing FPGA Bitstream
  - Data Type / Design Exploration
  - Primitive Support



CNN Computation in One Slide



$$I_{\text{new}}[x][y] = \sum_{x'=-1}^{1} \sum_{y'=-1}^{1} I_{\text{old}}[x+x'][y+y'] \times F[x'][y']$$



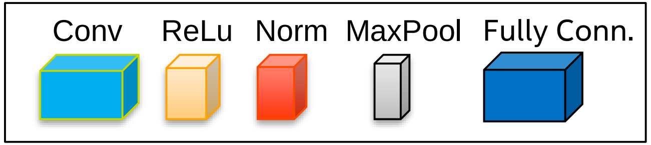
Input Feature Map
(Set of 2D Images)

Filter
(3D Space)

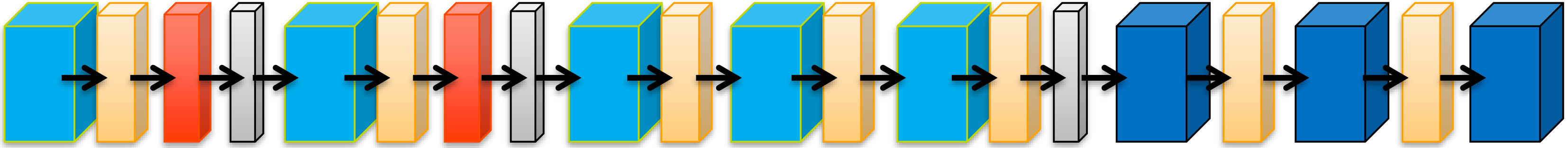
Output Feature Map

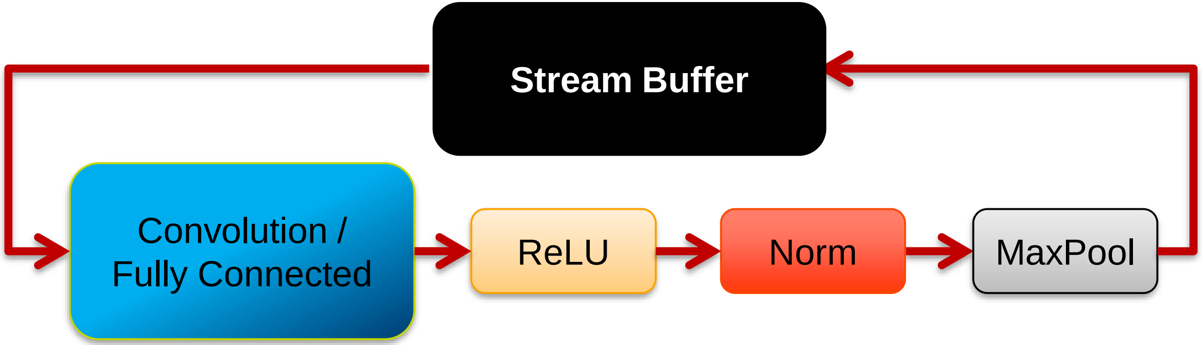
Repeat for Multiple Filters to Create Multiple "Layers" of Output Feature Map

Mapping Graphs in DLA



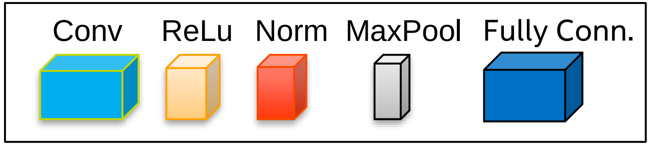
AlexNet Graph



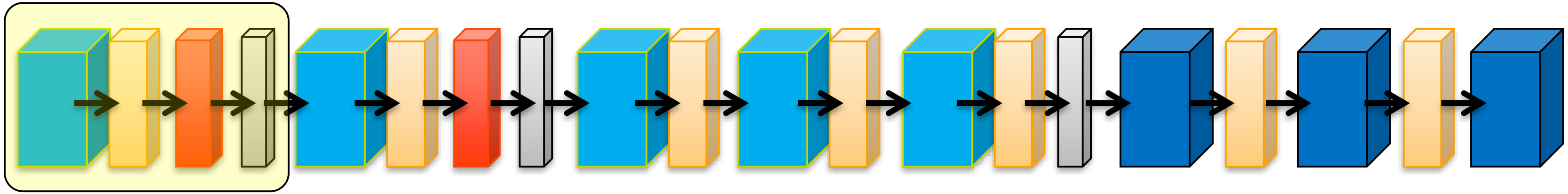


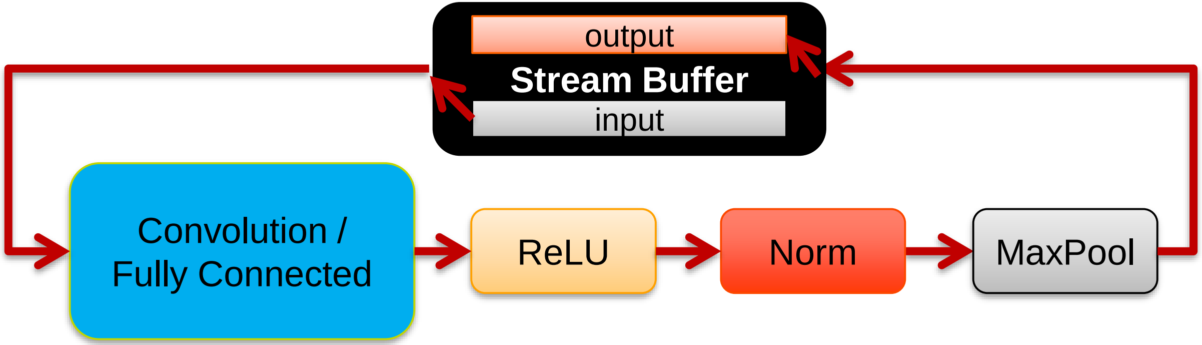
Blocks are run-time reconfigurable and bypassable

Mapping Graphs in DLA



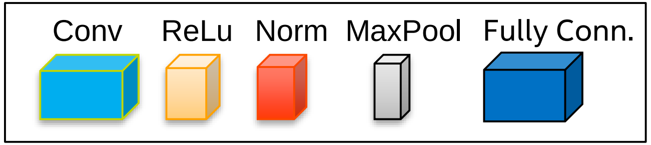
AlexNet Graph



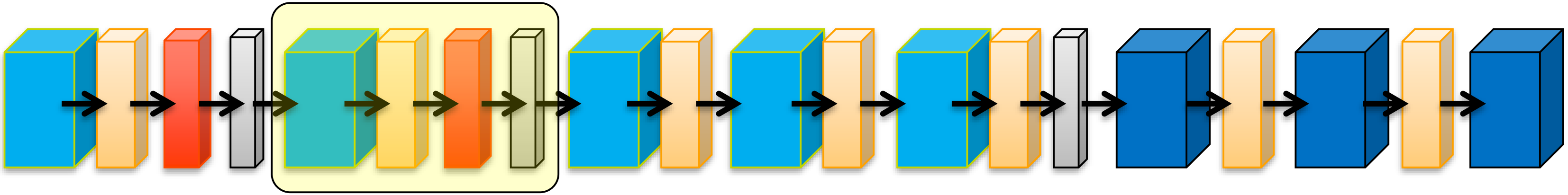


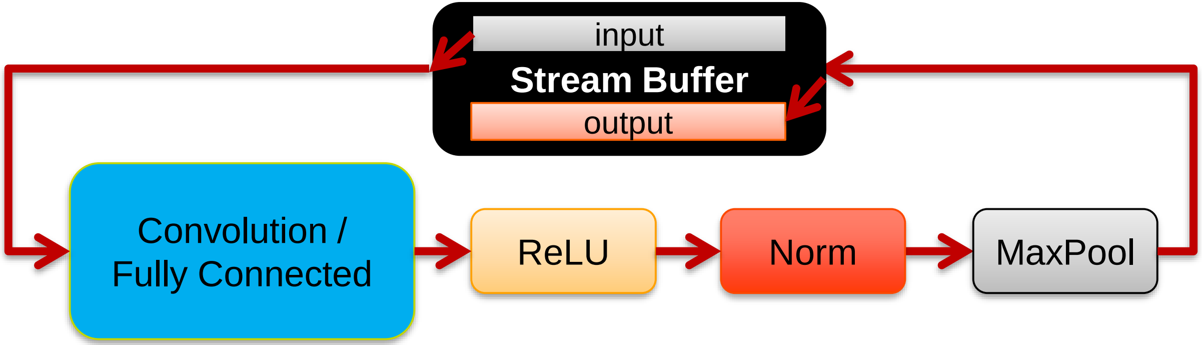
Blocks are run-time reconfigurable and bypassable

Mapping Graphs in DLA



AlexNet Graph





input

Stream Buffer

output

Convolution / Fully Connected

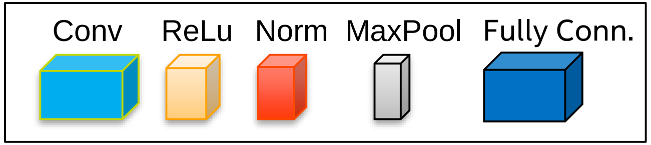
ReLU

Norm

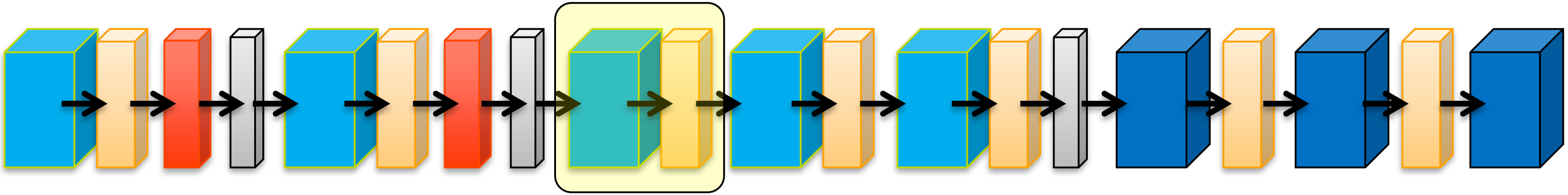
MaxPool

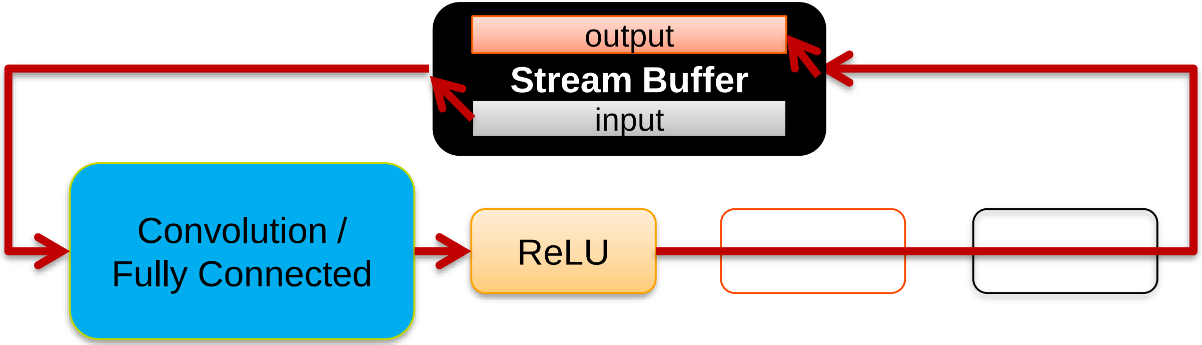
Blocks are run-time reconfigurable and bypassable

Mapping Graphs in DLA



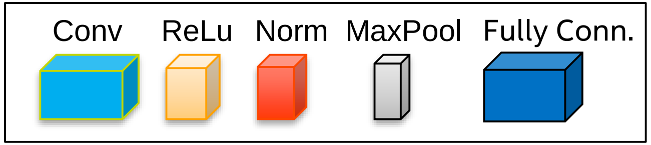
AlexNet Graph



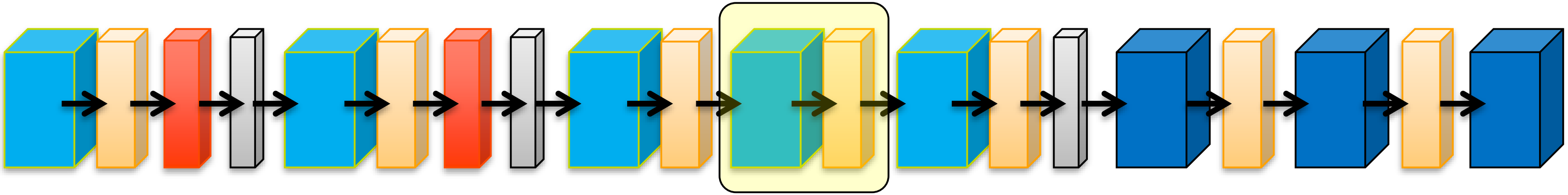


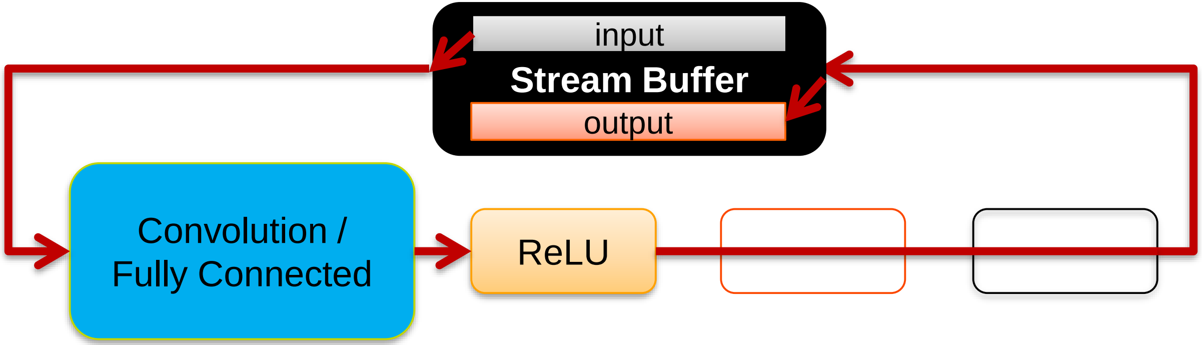
Blocks are run-time reconfigurable and bypassable

Mapping Graphs in DLA



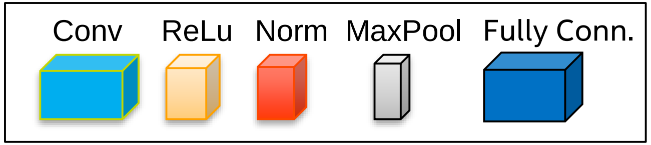
AlexNet Graph



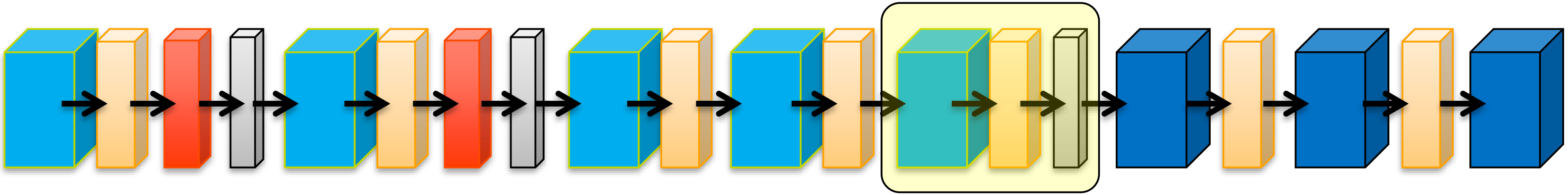


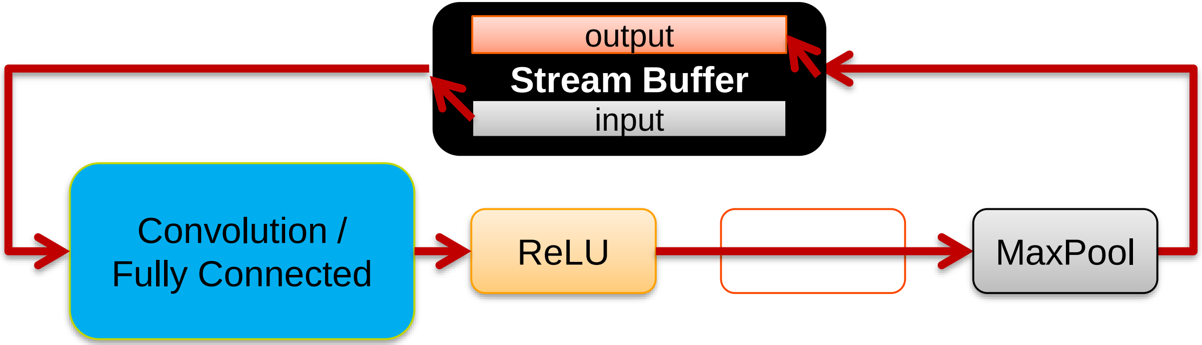
Blocks are run-time reconfigurable and bypassable

Mapping Graphs in DLA



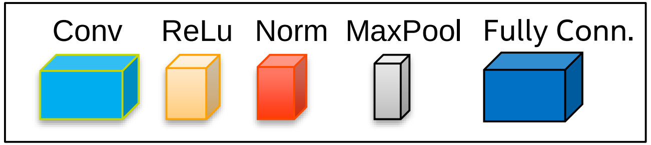
AlexNet Graph



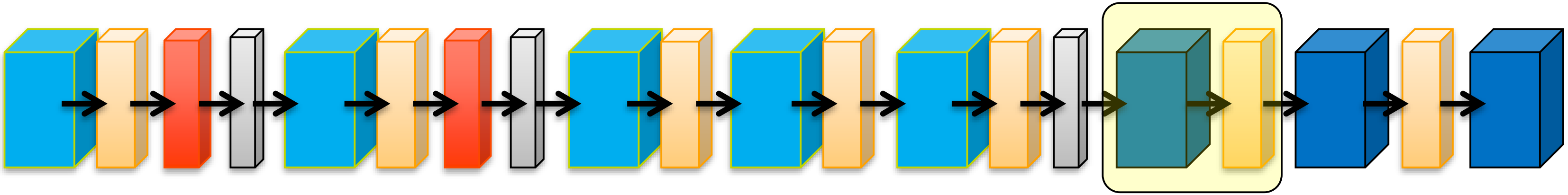


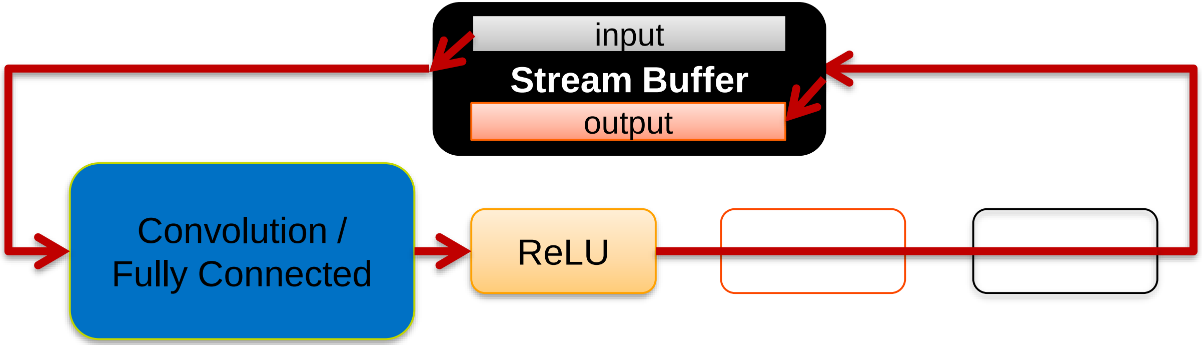
Blocks are run-time reconfigurable and bypassable

Mapping Graphs in DLA



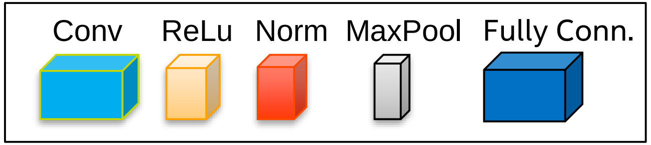
AlexNet Graph



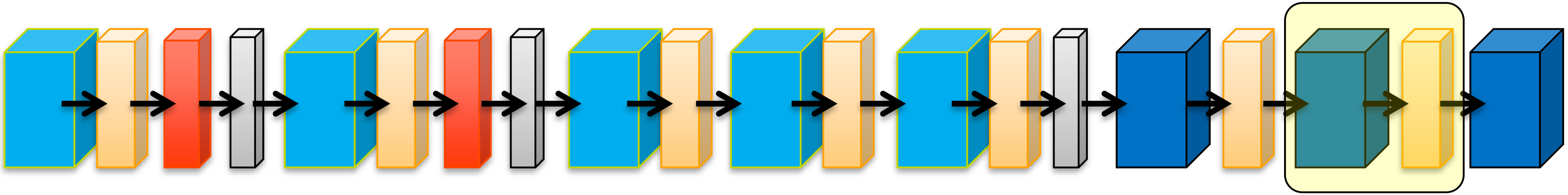


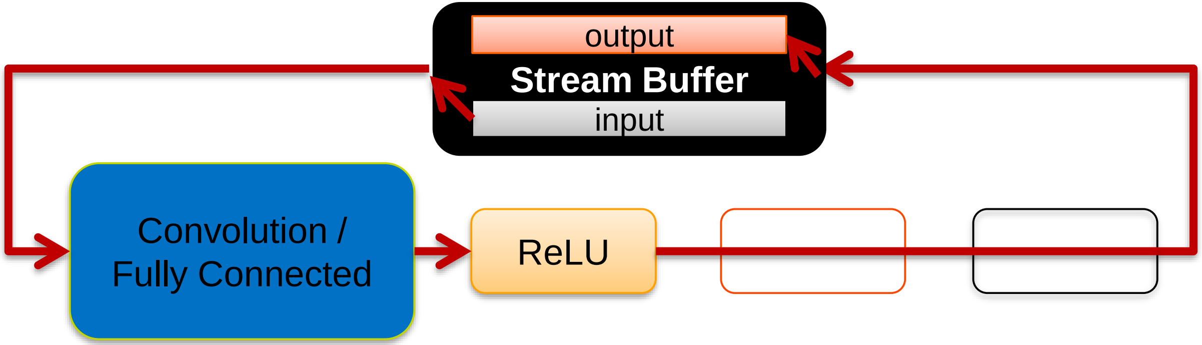
Blocks are run-time reconfigurable and bypassable

Mapping Graphs in DLA



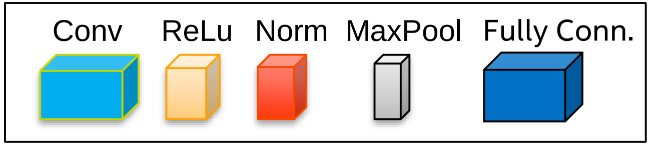
AlexNet Graph



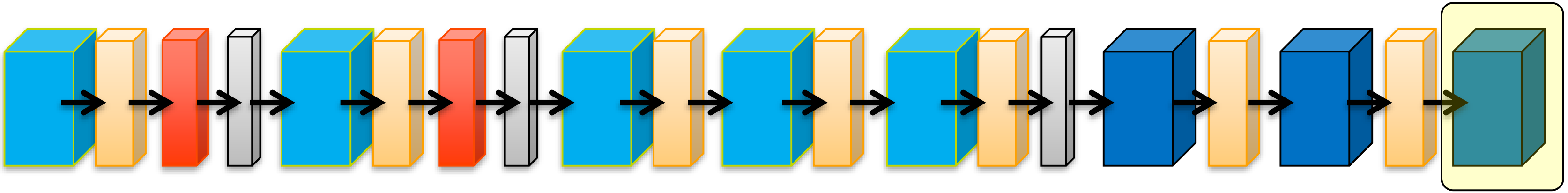


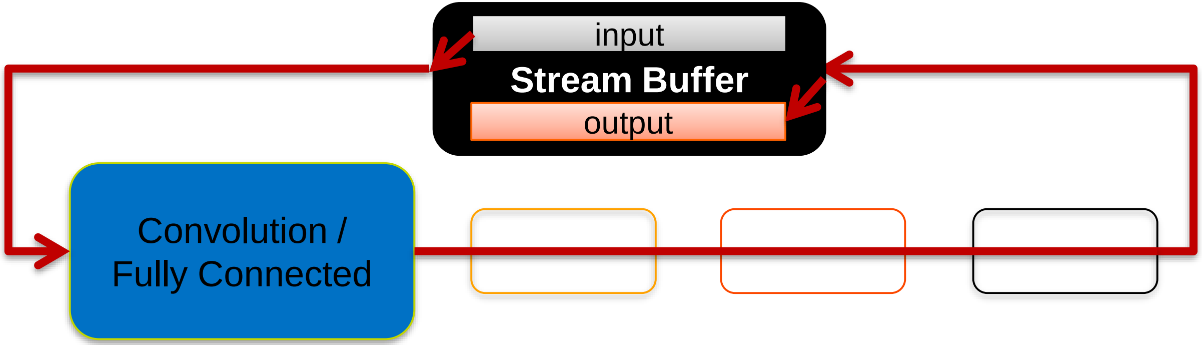
Blocks are run-time reconfigurable and bypassable

Mapping Graphs in DLA



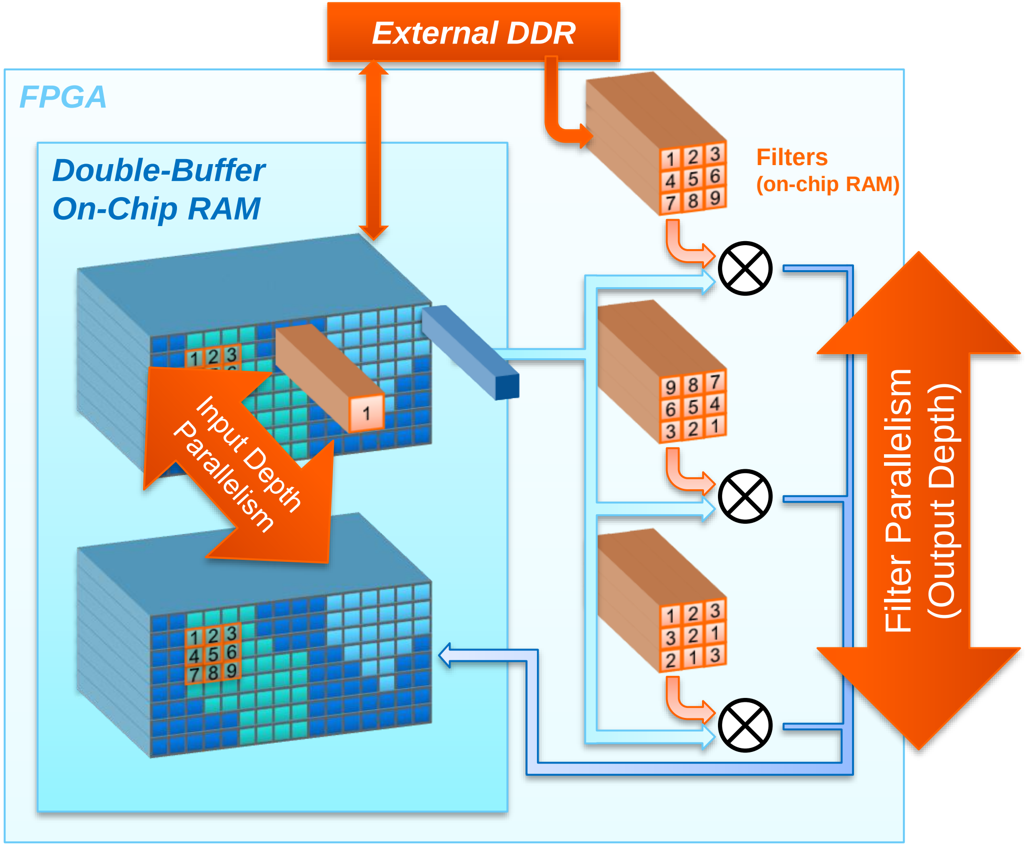
AlexNet Graph





Blocks are run-time reconfigurable and bypassable

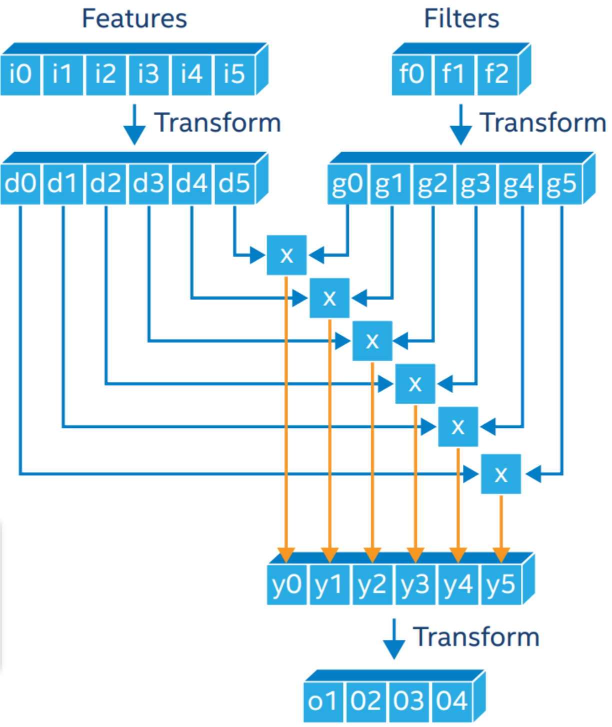
Efficient Parallel Execution of Convolutions

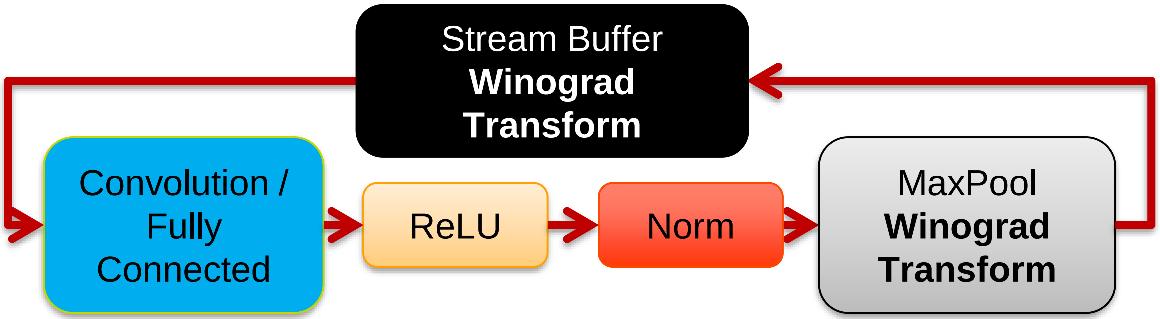


- Parallel Convolutions
  - Different filters of the same convolution layer processed in parallel in different processing elements (PEs)
- Vectored Operations
  - Across the depth of feature map
- PE Array geometry can be customized to hyperparameters of given topology

Winograd Transformation

- Perform convolutions with fewer multiplication
  - Allows more convolutions to be done on FPGA
- Take 6 input features elements and 3 filter elements
  - Standard convolution requires 12 multiplies
  - Transformed convolution requires just 6 multiplies




Stream Buffer **Winograd Transform**

Convolution / Fully Connected

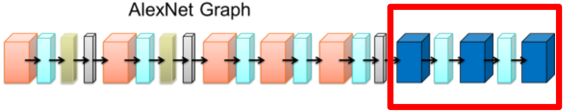
ReLU

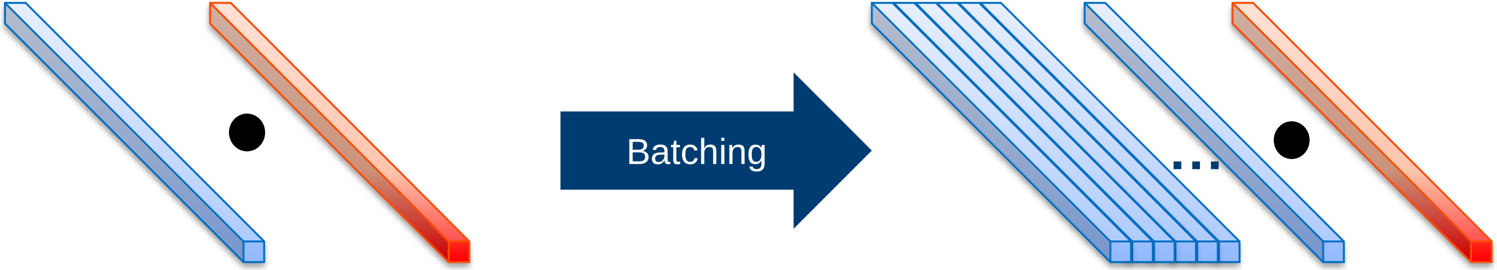
Norm

MaxPool **Winograd Transform**

Fully Connected Computation and Batching

- Fully Connected Layer computation does not allow for data reuse of weights
  - Different from convolutions
  - Very memory bandwidth intensive
- Solution: Batch up images
  - Weights reused across multiple images





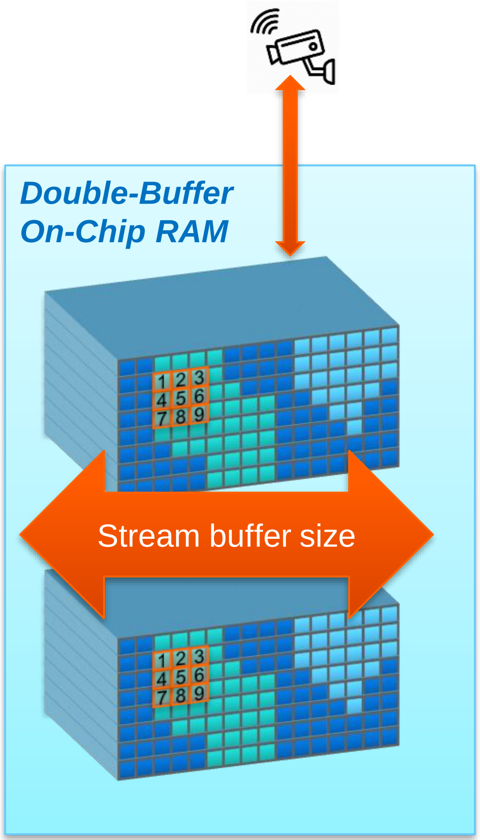
$$o = I_{vec} * W_{vec}$$

$$O_{vec} = I_{mat} * W_{vec}$$

Feature Cache

Feature data cached on-chip

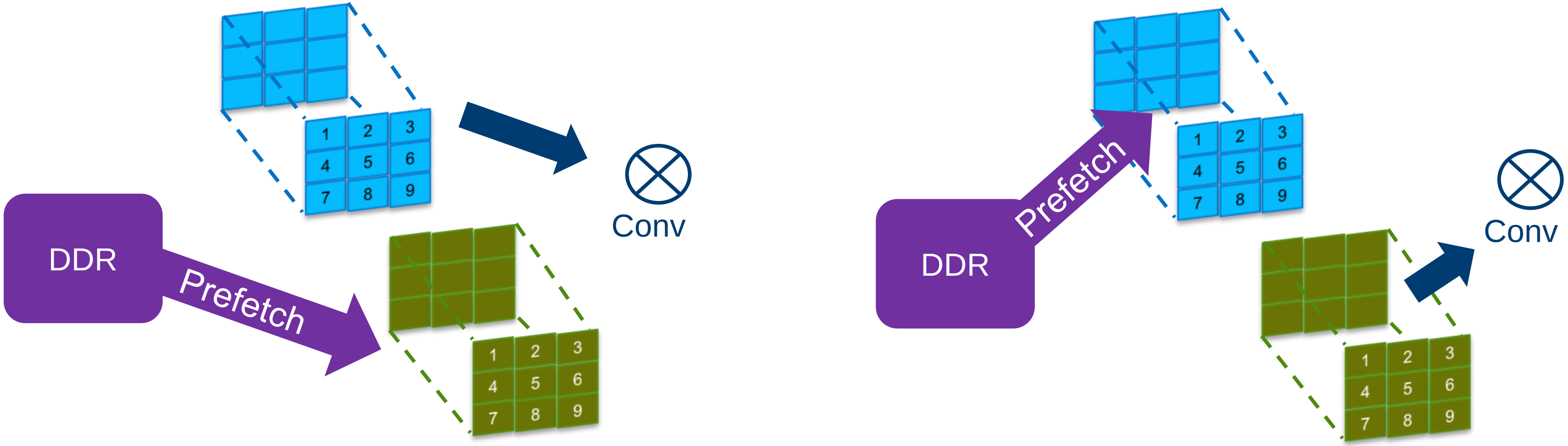
- Streamed to a daisy chain of parallel processing elements
- Double buffered
  - Overlap convolution with cache updates
  - Output of one subgraph becomes input of another
  - Eliminates unnecessary external memory accesses



Filter Cache

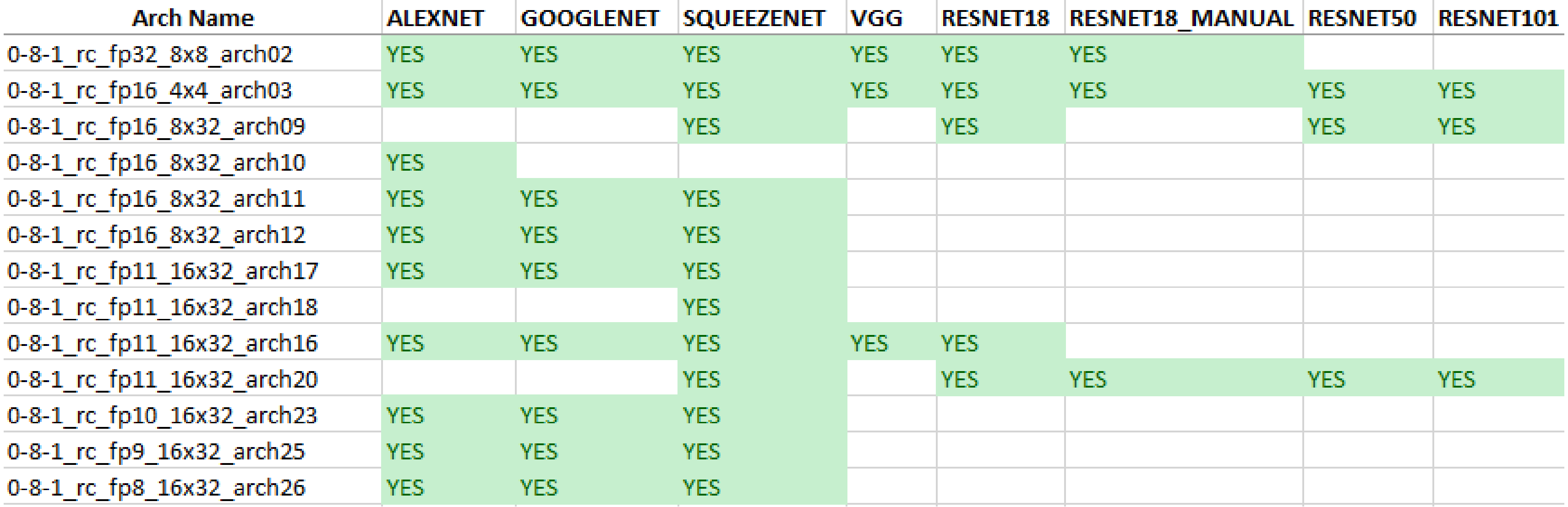
Filter weights cached in each processing element

- Double buffered in order to support prefetching
  - While one set is used to calculate output feature maps, another set is prefetched



DLA Architecture Selection

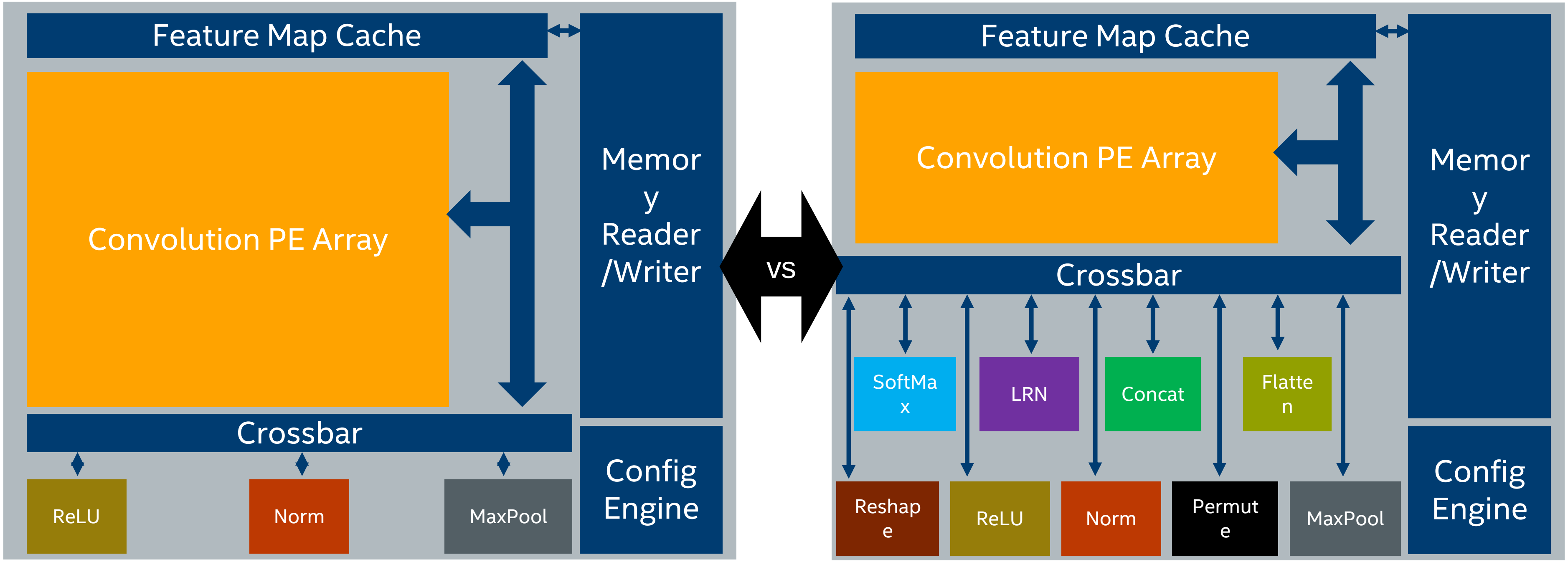
- Find ideal FPGA image that meets your needs
- Create custom FPGA image based on need



Arch Name	ALEXNET	GOOGLENET	SQUEEZENET	VGG	RESNET18	RESNET18_MANUAL	RESNET50	RESNET101
0-8-1_rc_fp32_8x8_arch02	YES	YES	YES	YES	YES	YES		
0-8-1_rc_fp16_4x4_arch03	YES	YES	YES	YES	YES	YES	YES	YES
0-8-1_rc_fp16_8x32_arch09			YES		YES		YES	YES
0-8-1_rc_fp16_8x32_arch10	YES							
0-8-1_rc_fp16_8x32_arch11	YES	YES	YES					
0-8-1_rc_fp16_8x32_arch12	YES	YES	YES					
0-8-1_rc_fp11_16x32_arch17	YES	YES	YES					
0-8-1_rc_fp11_16x32_arch18			YES					
0-8-1_rc_fp11_16x32_arch16	YES	YES	YES	YES	YES			
0-8-1_rc_fp11_16x32_arch20			YES		YES	YES	YES	YES
0-8-1_rc_fp10_16x32_arch23	YES	YES	YES					
0-8-1_rc_fp9_16x32_arch25	YES	YES	YES					
0-8-1_rc_fp8_16x32_arch26	YES	YES	YES					

Support for Different Topologies

Tradeoff between features and performance



Supported Primitives and Topologies

Primitives

- ✓ batch norm
- ✓ max pool
- ✓ average pool
- ✓ inner product
- ✓ reshape
- ✓ priorbox
- ✓ concat
- ✓ relu, leaky relu
- ✓ scale
- ✓ permute
- ✓ detection output
- ✓ fully connected
- ✓ flatten
- ✓ lrn normalization
- ✓ softmax
- ✓ prelu
- ✓ conv
- ✓ eltwise

- bias
- local conv
- power
- slice
- group conv
- sigmoid
- crop
- depthwise conv
- dilated conv
- depthwise conv
- elu
- proporal
- roi pooling

- ✓ tanh
- ✓ deconv

Topologies

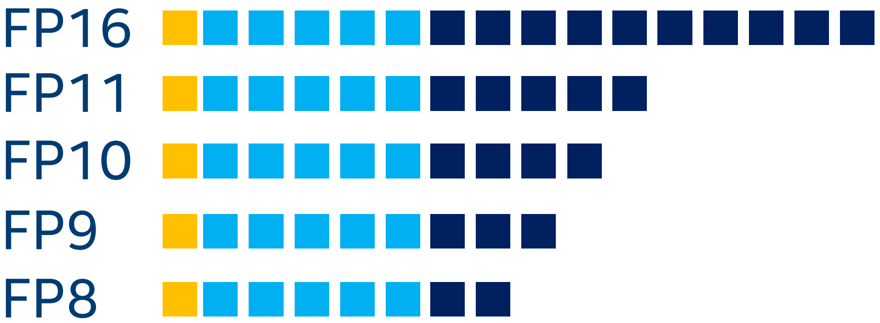
✓ AlexNet	
✓ GoogleNet v1	✓ SSD
✓ ResNet18	✓ SSD
✓ ResNet50	
✓ ResNet101	
✓ SqueezeNet	✓ SSD
✓ VGG16	
✓ Tiny Yolo	
✓ LeNet	

✓ Supported
✓ Upon Request
✓ Future

Design Exploration with Reduced Precision

Tradeoff between performance and accuracy

- Reduced precision allows more processing to be done in parallel
- Using smaller Floating Point format does not require retraining of network
- FP11 benefit over using INT8/9
  - No need to retrain, better performance, less accuracy loss



FP16 Sign, 5-bit exponent, 10-bit mantissa

FP11 Sign, 5-bit exponent, 5-bit mantissa

FP10 Sign, 5-bit exponent, 4-bit mantissa

FP9 Sign, 5-bit exponent, 3-bit mantissa

FP8 Sign, 5-bit exponent, 2-bit mantissa

Summary

- Use HLS Compiler to generate excelleration IP for HW Developers
- Use OpenCL to accelerate for software developers
  - May use over Acceleration Stack
- Acceleration stack enables data center acceleration
  - Supports RTL and OpenCL flow, HLS in the future
- Use Deep Learning Acceleration Suite to easily deploy inference tasks on the FPGA
  - Supported for the Acceleration Stack
  - In the future will support custom platforms

Legal Disclaimers/Acknowledgements

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at www.intel.com.

Intel, the Intel logo, Intel Inside, the Intel Inside logo, MAX, Stratix, Cyclone, Arria, Quartus, HyperFlex, Intel Atom, Intel Xeon and Enpirion are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

OpenCL is the trademark of Apple Inc. used by permission by Khronos

*Other names and brands may be claimed as the property of others intel®

experience
what's inside™