

INTRO TO FPGA FLOWS OVERVIEW

Bill Jenkins

Intel Programmable Solutions Group

Agenda

- 9:00 am Welcome
- 9:15 am Introduction to FPGAs
- 9:45 am FPGA Programming models: RTL
- 10:15 am FPGA Programming models: HLS
- 11:00 am Lab 1 HLS Flow
- 11:45 am Lunch
- 12:30 pm FPGA Programming models: OpenCL
- 1:00 pm High Performance Data Flow Concepts
- 1:30 pm Lab 2 OpenCL Flow
- 2:15 pm Introduction to DSP Builder
- 3:00 pm Introduction to Acceleration Stack
- 4:00 pm Lab 3 Acceleration Stack
- 4:30 pm Curriculum & University Program Coordination



INTRODUCTION

Intel Proprietary
for LRZ

The Problem: Flood of Data

BY 2020



The average internet user will generate
~1.5 GB OF TRAFFIC PER DAY



Smart hospitals will be generating over
3 TB PER DAY



Self driving cars will be generating over
4 TB PER DAY... EACH



A connected plane will be generating over
40 TB PER DAY



A connected factory will be generating over
1 PB PER DAY



RADAR ~10-100 KB PER SECOND

SONAR ~10-100 KB PER SECOND

GPS ~50 KB PER SECOND

LIDAR ~10-70 MB PER SECOND

CAMERAS ~20-40 MB PER SECOND

1 CAR 5 EXAFLOPS PER HOUR

All numbers are approximated
<http://www.asco.com/c/en/us/solutions/service-provider/vni-network-traffic-forecast/infographic.html>
http://www.asco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html
<https://datafloq.com/read/self-driving-cars-create-2-petabytes-data-annually/172>
http://www.asco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html
http://www.asco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html

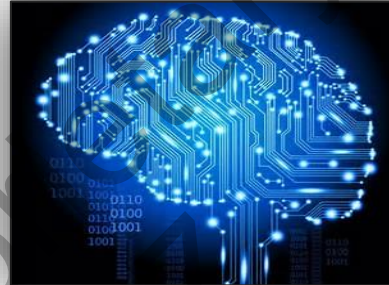
Typical HPC Workloads



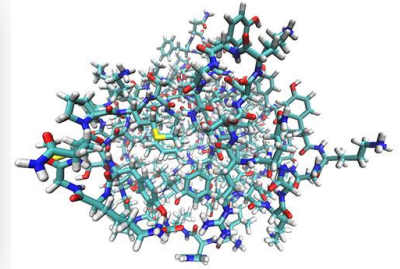
ASTROPHYSICS



GENOMICS / BIO-INFORMATICS



ARTIFICIAL INTELLIGENCE



MOLECULAR DYNAMICS*



BIG DATA ANALYTICS



FINANCIAL



WEATHER & CLIMATE



CYBER SECURITY

* Source: <https://comp-physics-lincoln.org/2013/01/17/molecular-dynamics-simulations-of-amphiphilic-macromolecules-at-interfaces/>

Fast Evolution of Technology

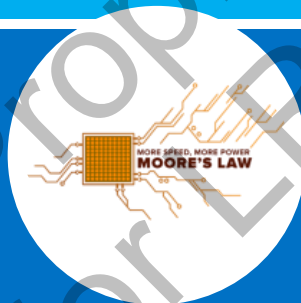
We now have the compute to solve these problems today in near real-time

Bigger Data



Image: 50 MB / picture
Audio: 5 MB / song
Video: 47 GB / movie

Better Hardware



Transistor density doubles
every 18 months
Cost / GB in 1995: \$1000.00
Cost / GB in 2015: \$0.03

Smarter Algorithms



Advances in neural
networks leading to better
accuracy in training models



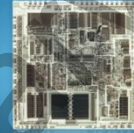
1974

Intel® 8080 processor
Transistors: 4,500
6 micron



1978

Intel® 8086 processor
Transistors: 29,000
3 micron



1982

Intel® 286™ processor
Transistors: 134,000
1.5 micron



1985

Intel® 386™ processor
Transistors: 275,000
1.5 microns

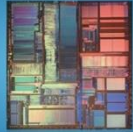


INTEL SOFTWARE



1989

Intel® 486™ processor
Transistors: 1.2 million
1 micron



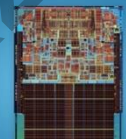
1993

Intel® Pentium® processor
Transistors: 3.1 million
0.8 micron



2000

Intel® Pentium® 4 processor
Transistors: 42 million
1.8 micron



2006

Intel® Core™2 Duo processor
Transistors: 291 million
65nm



2015

Intel® Xeon D
Transistors: 4.3 billion
14 nm



50+ Years of Moore's Law Computing has Changed...



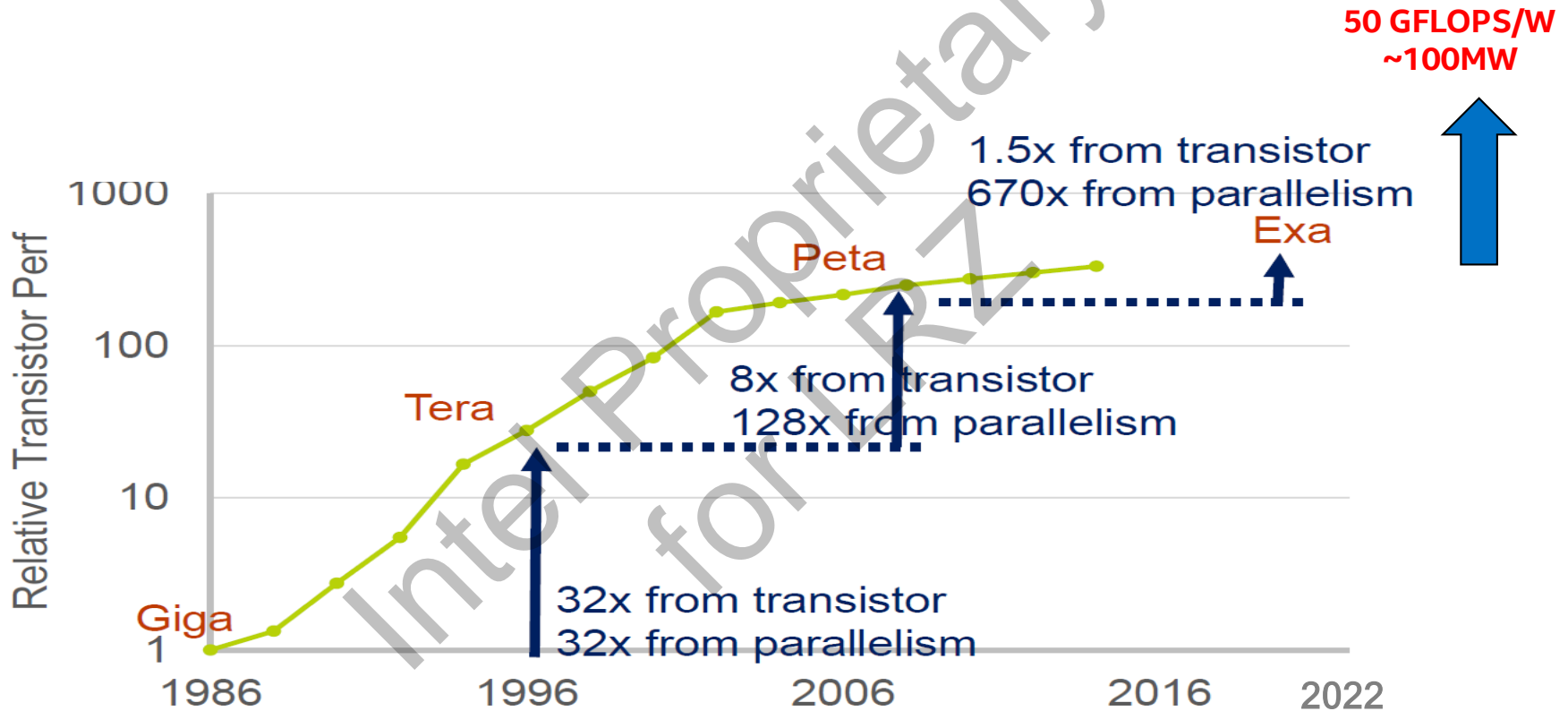
The Urgency of Parallel Computing

If engineers keep building processors the way we do now, CPUs will get even faster but they'll require so much power that they won't be usable.

—Patrick Gelsinger,
former Intel Chief Technology Officer,
February 7, 2001

Source: <http://www.cnn.com/2001/tech/ptech/02/07/hot.chips.idg/>

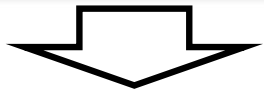
Implications to High Performance Computing



Challenges Scaling Systems to Higher Performance

CPU Intensive

Result:
Excessive power requirements



Memory Intensive

Result: Slow Performance

Memory



Bottleneck

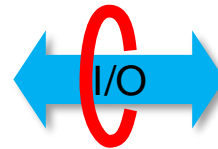
System

Bottleneck



I/O

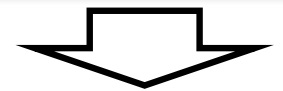
Bottleneck



I/O

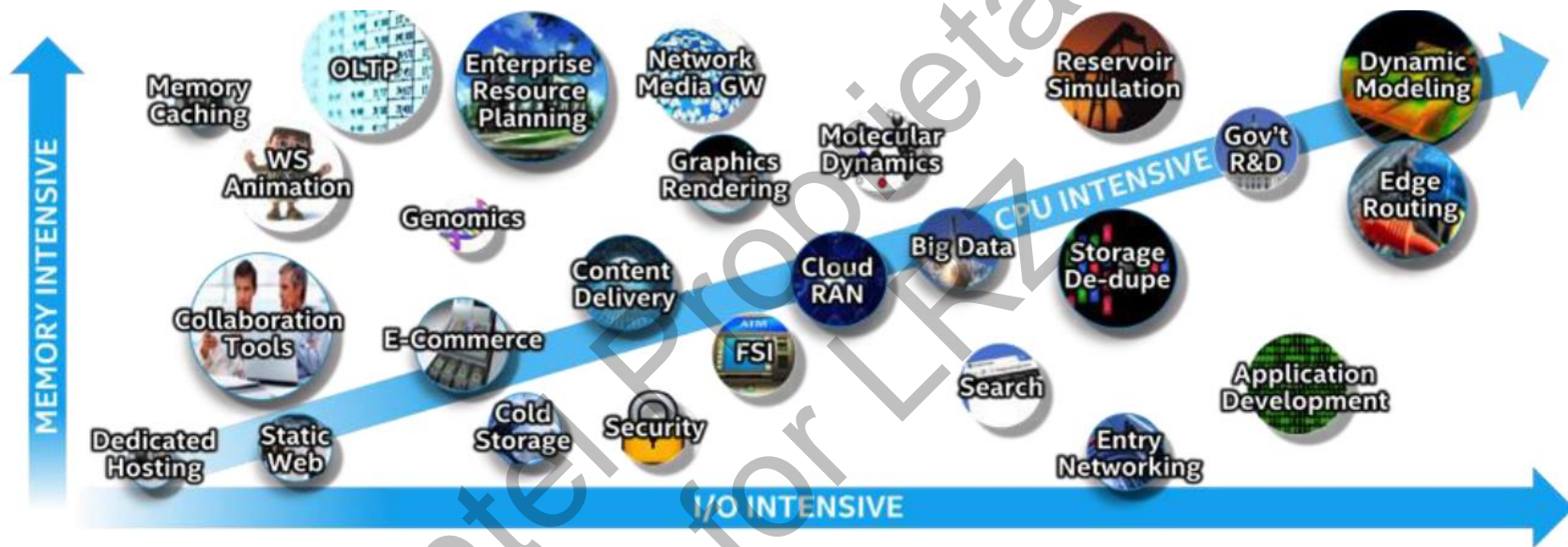
IO Intensive

Result:
Slow Performance
(high latency)



Need to think about Compute Offload as well as Ingress/Egress Processing

Diverse Application Demands



Accelerators can increase Performance at lower TCO for targeted workloads

Intel estimates; bubble size is relative CPU intensity

The Intel Vision

Heterogeneous Systems:

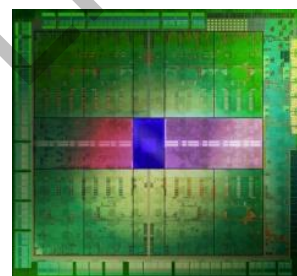
- Span from CPU to GPU to FPGA to dedicated devices with consistent programming models, languages, and tools



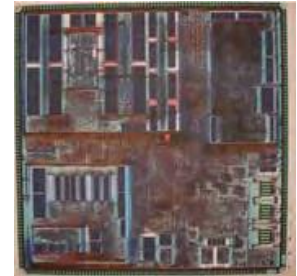
CPUs



GPUs



FPGAs



ASSP

Heterogeneous Computing Systems

Modern systems contain more than one kind of processor

- Applications exhibit different behaviors:
 - Control intensive (Searching, parsing, etc...)
 - Data intensive (Image processing, data mining, etc...)
 - Compute intensive (Iterative methods, financial modeling, etc...)
- Gain performance by using specialized capabilities of different types of processors

Separation of Concerns

Two groups of developers:

- Domain experts concerned with getting a result
 - Host application developers leverage optimized libraries
- Tuning experts concerned with performance
 - Typical FPGA developers that create optimized libraries

Intel® Math Kernel Library a simple example of raising the level of abstraction to the math operations

- Domain experts focus on formulating their problems
- Tuning experts focus on vectorization and parallelization



INTRODUCTION TO FPGAS

Intel Proprietary
for RZ

FPGA Enabled Performance and Agility

Workload Optimization: ensure Xeon cores serve their highest value processing

Efficient Performance: improve performance/watt

Real-Time: high bandwidth connectivity and low-latency parallel processing

Developer Advantage: code re-use across Intel FPGA data center products

Workload 1

Workload 2

Workload N



Milliseconds

Acceleration Stack for Intel® Xeon® CPU with FPGAs

Intel Environment

Code re-use

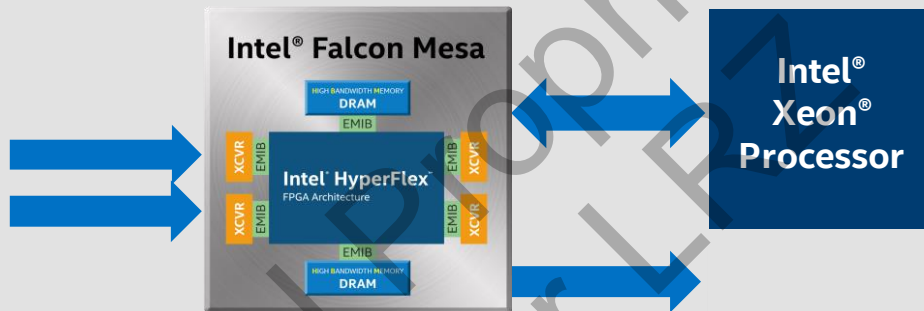
IP Libraries

FPGAs enhance CPU-based processing by *accelerating algorithms* and *minimizing bottlenecks*

FPGAs Provide Flexibility to Control the Data path

Compute Acceleration/Offload

- Workload agnostic compute
- FPGAaaS
- Virtualization



Inline Data Flow Processing

- Machine learning
- Object detection and recognition
- Advanced driver assistance system (ADAS)
- Gesture recognition
- Face detection

Storage Acceleration

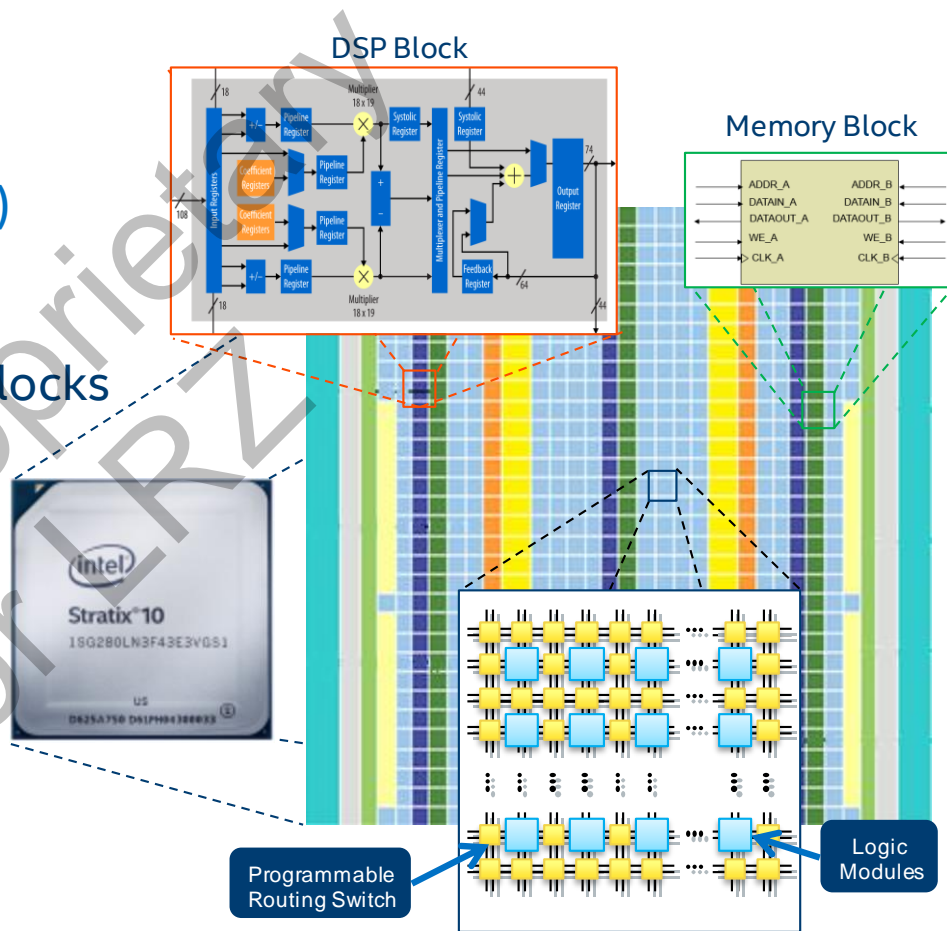
- Machine learning
- Cryptography
- Compression
- Indexing

FPGA Architecture

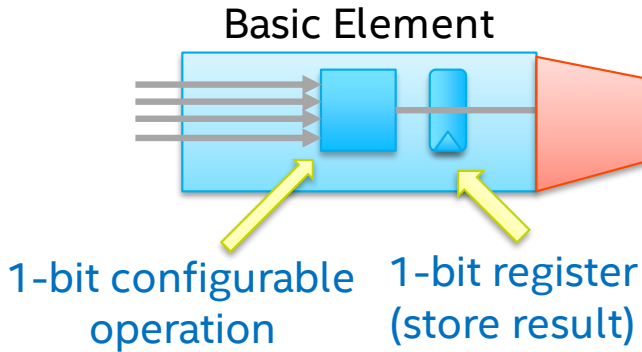
Field Programmable Gate Array (FPGA)

- Millions of logic elements
- Thousands of embedded memory blocks
- Thousands of DSP blocks
- Programmable interconnect
- High speed transceivers
- Various built-in hardened IP

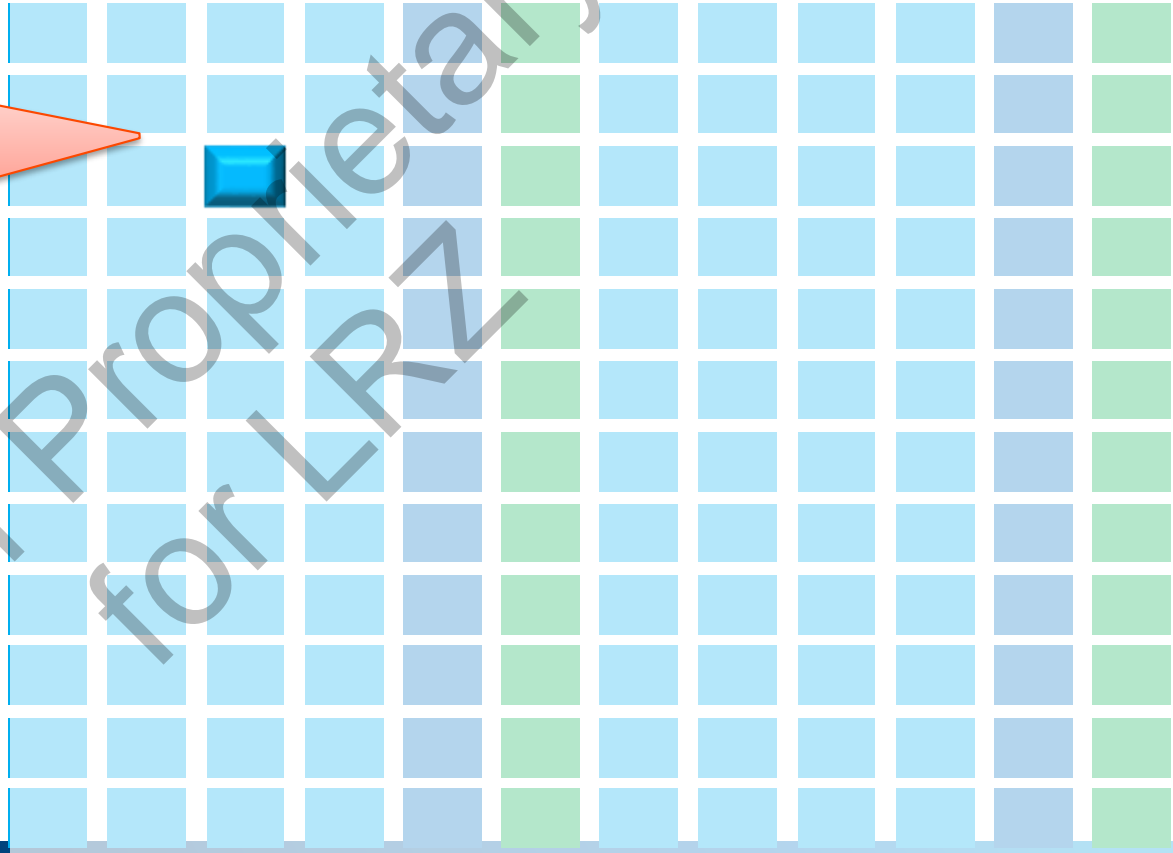
Used to create **Custom Hardware!**



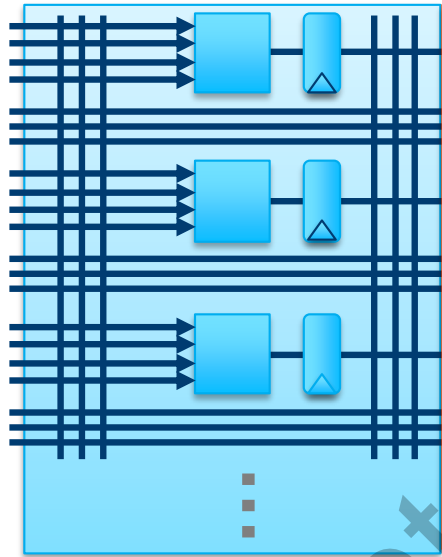
FPGA Architecture: Basic Elements



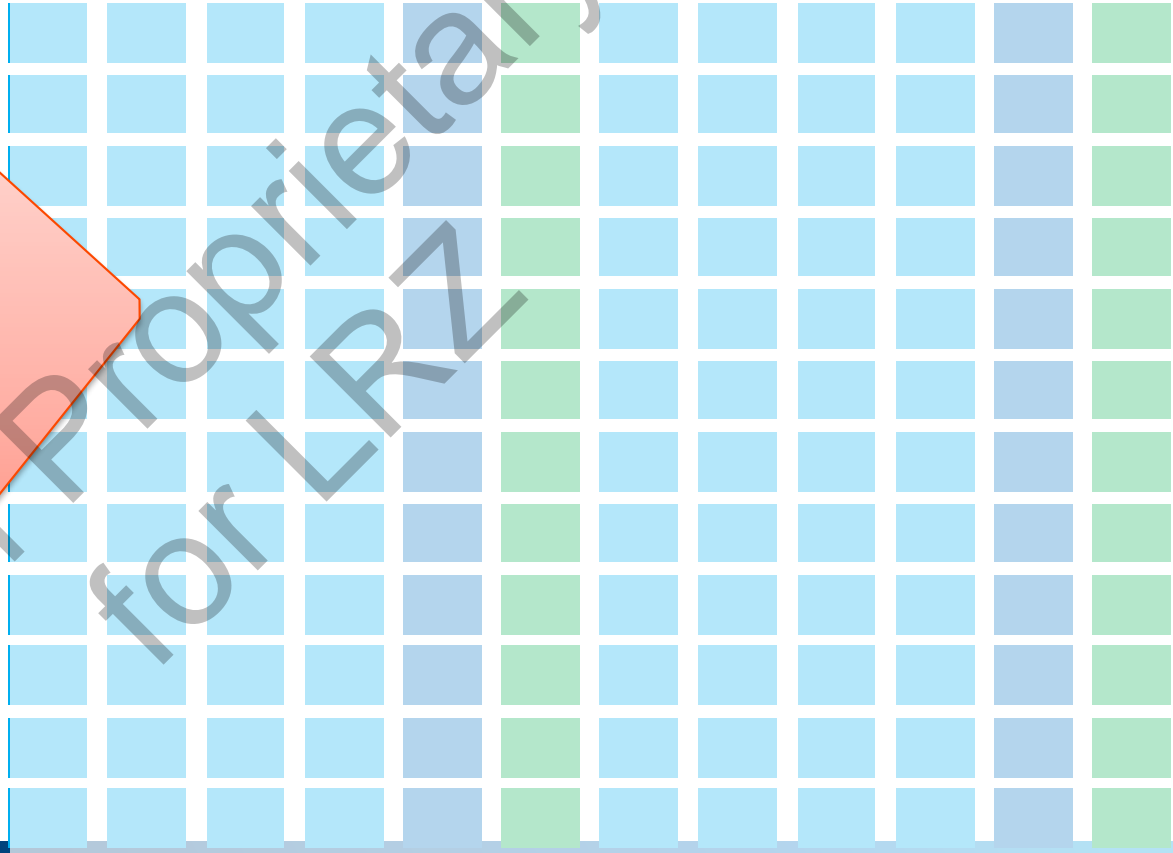
Configured to perform any
1-bit operation:
AND, OR, NOT, ADD, SUB



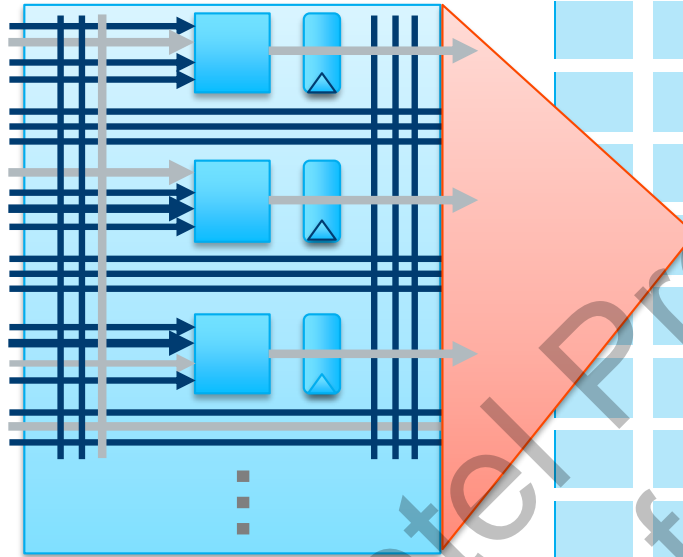
FPGA Architecture: Flexible Interconnect



Basic Elements are surrounded with a flexible interconnect

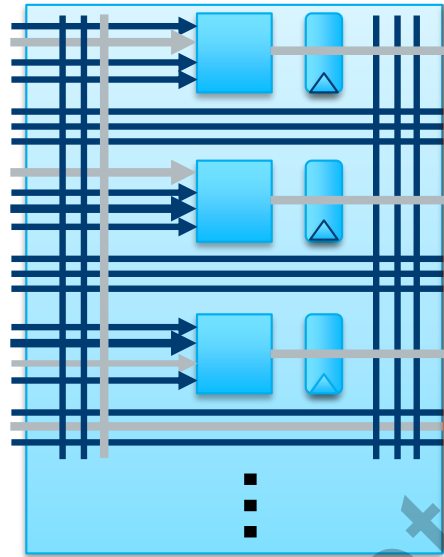


FPGA Architecture: Flexible Interconnect

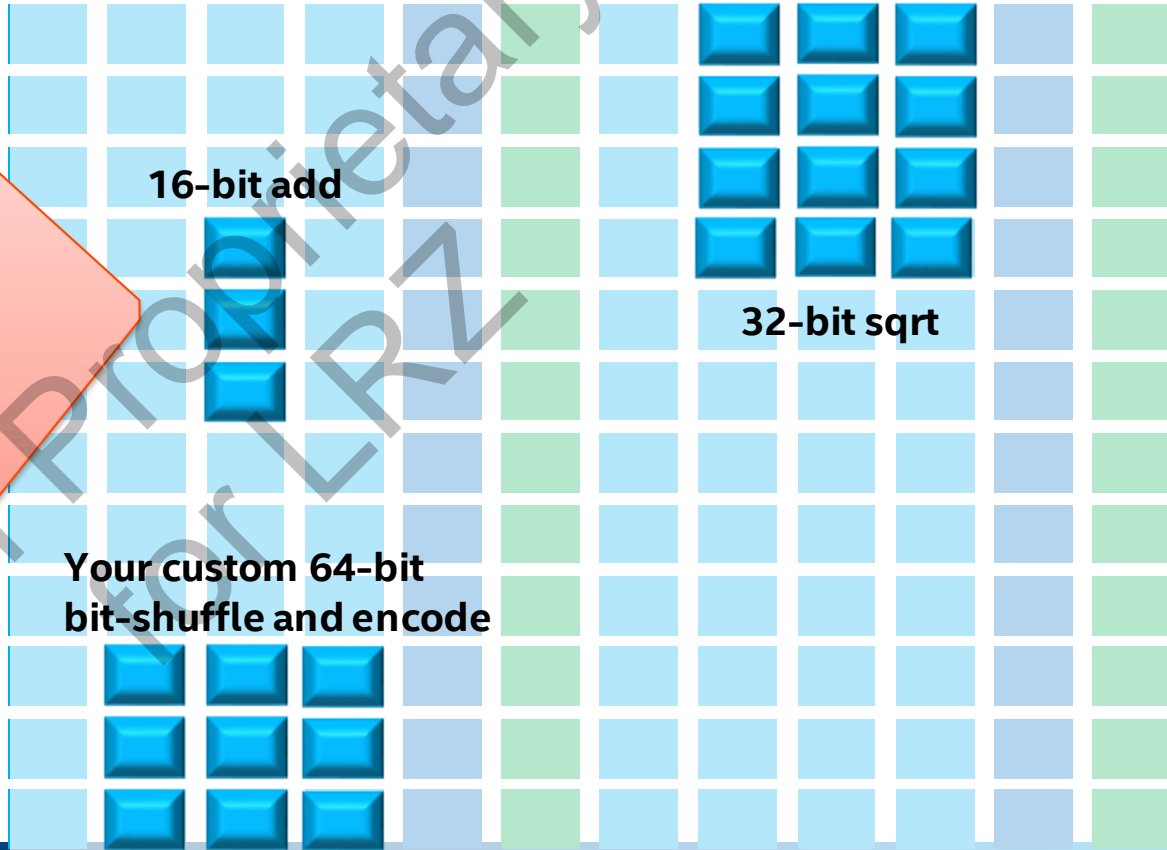


Wider *custom* operations are implemented by configuring and interconnecting Basic Elements

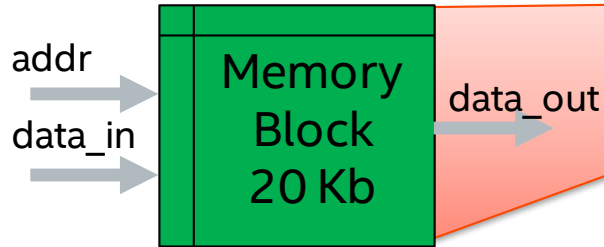
FPGA Architecture: Custom Operations Using Basic Elements



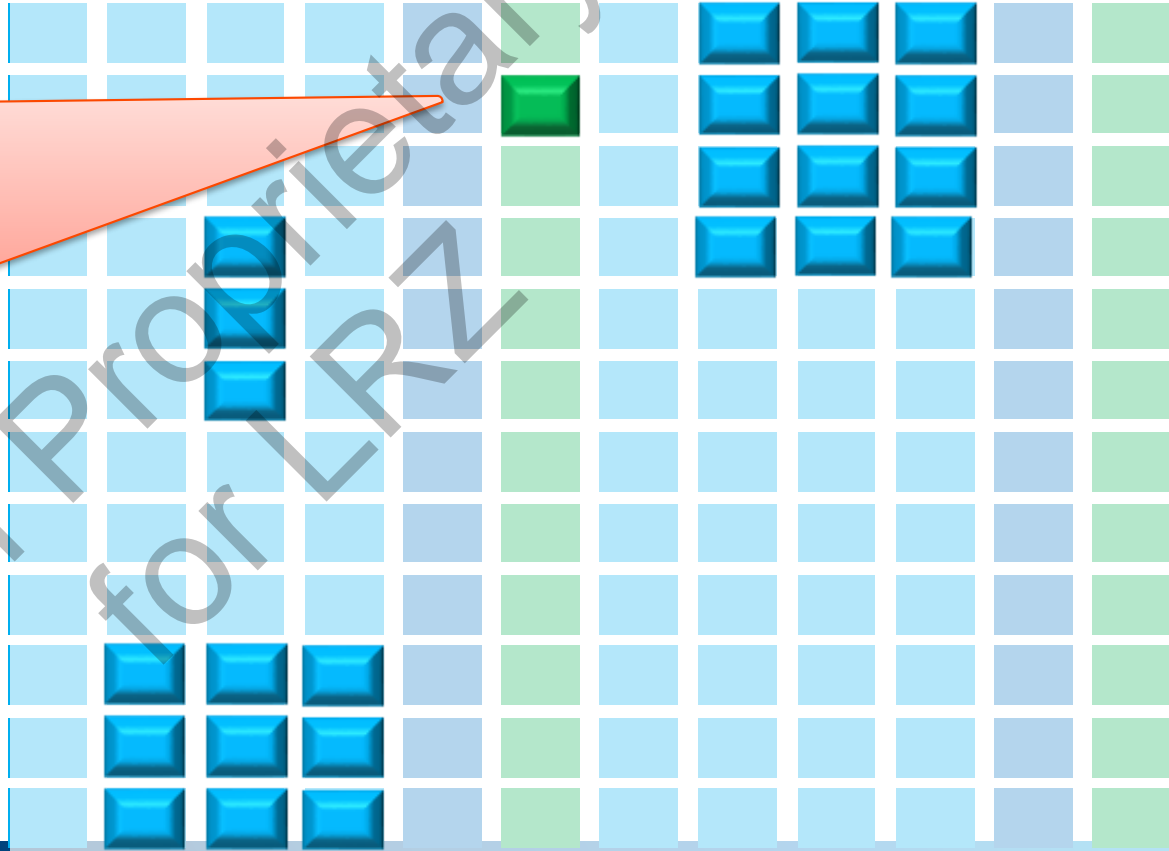
Wider *custom* operations are implemented by configuring and interconnecting Basic Elements



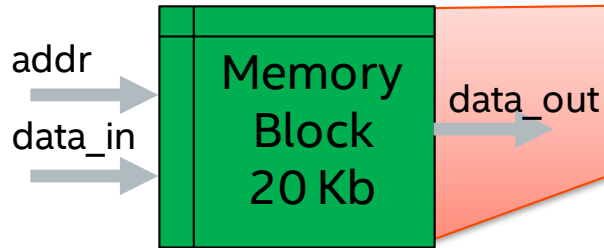
FPGA Architecture: Memory Blocks



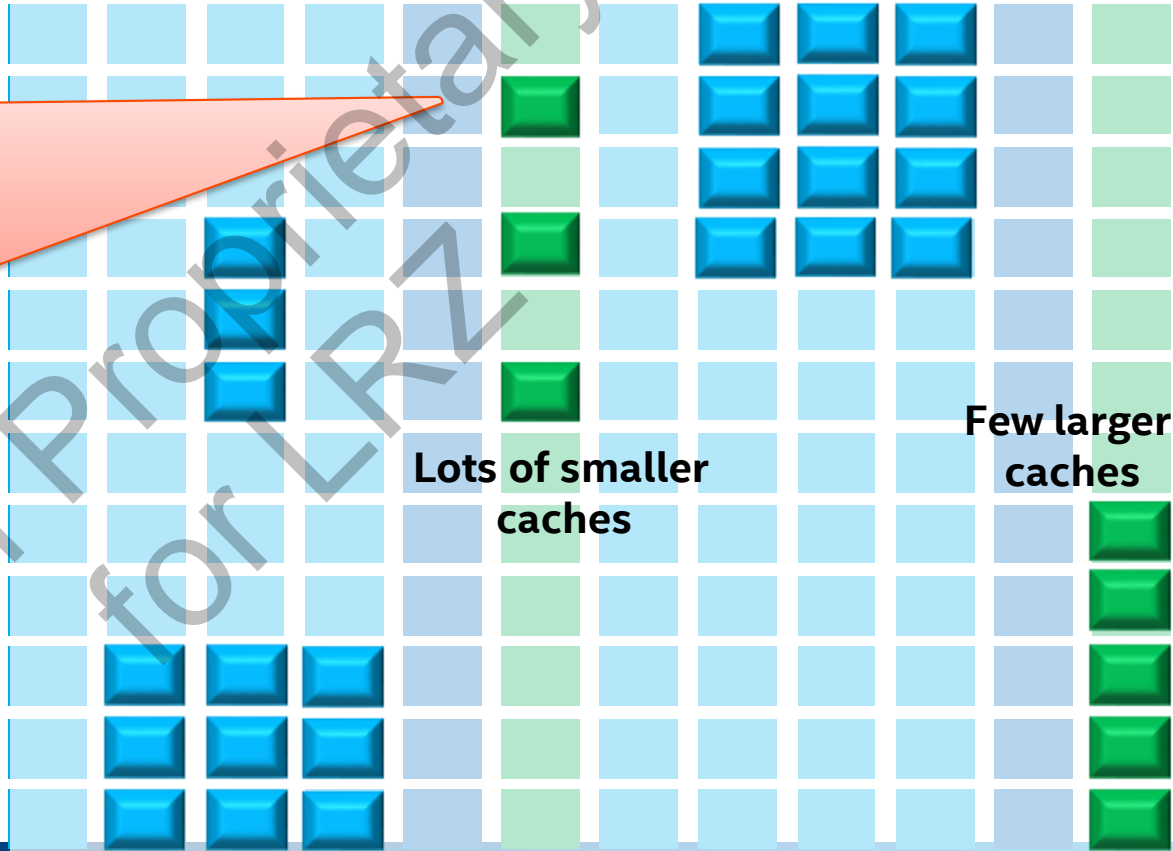
Can be configured and grouped using the interconnect to create various cache architectures



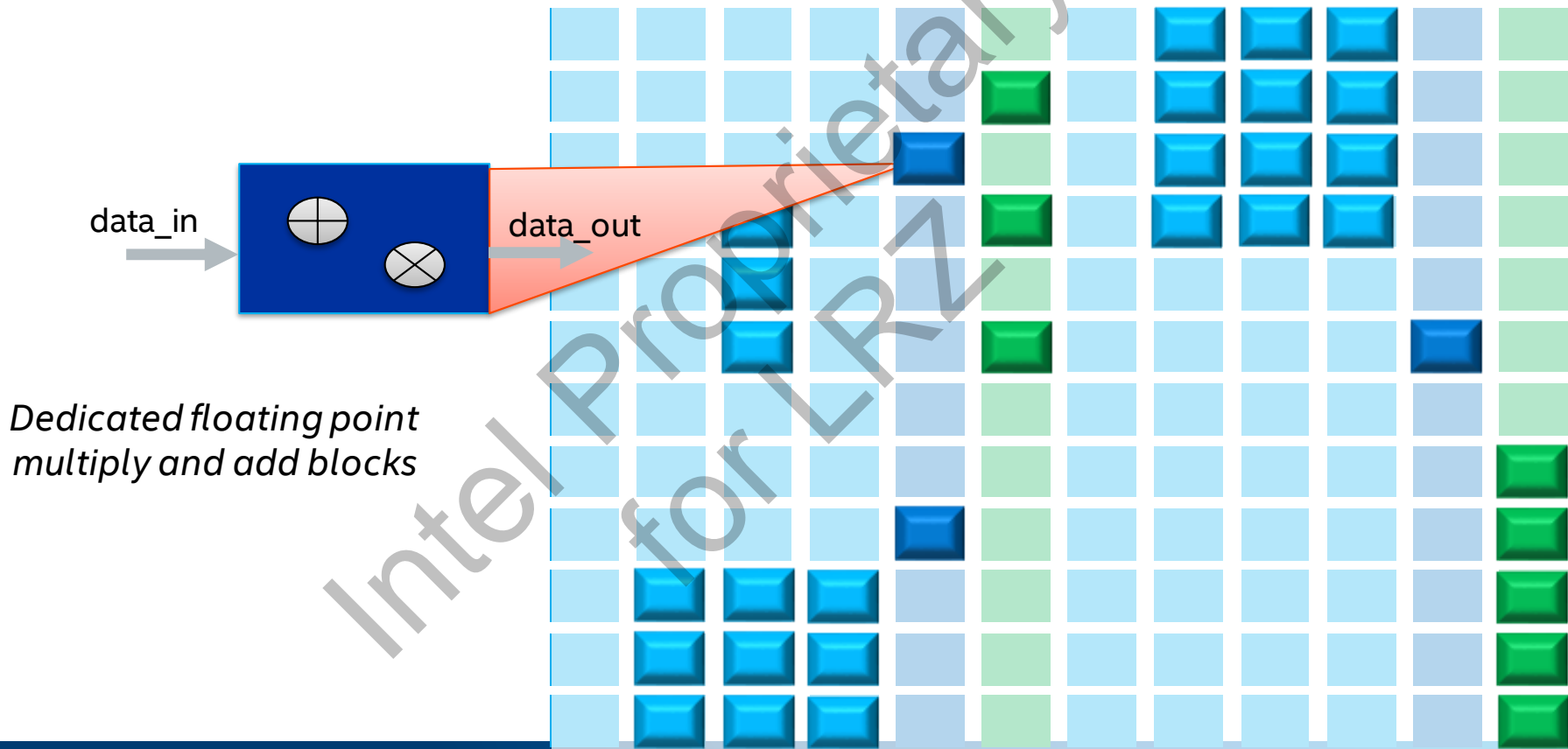
FPGA Architecture: Memory Blocks



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



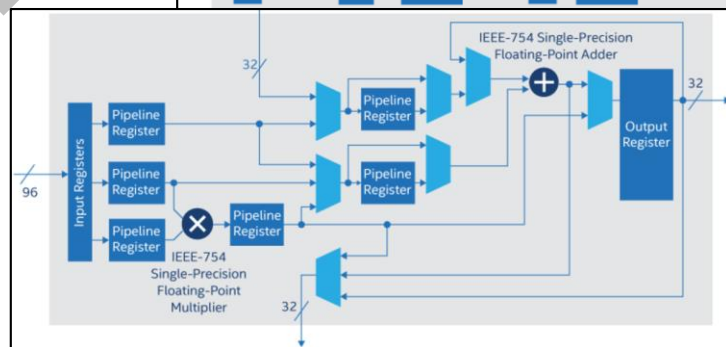
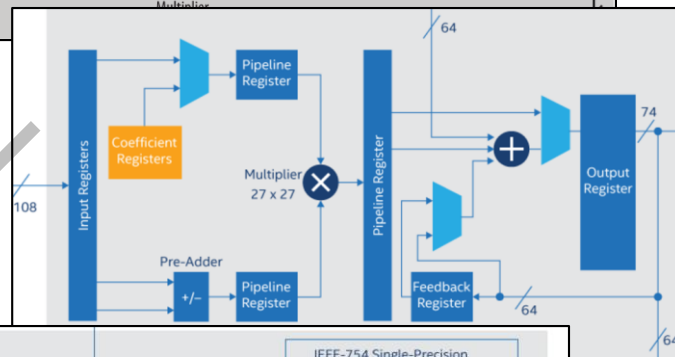
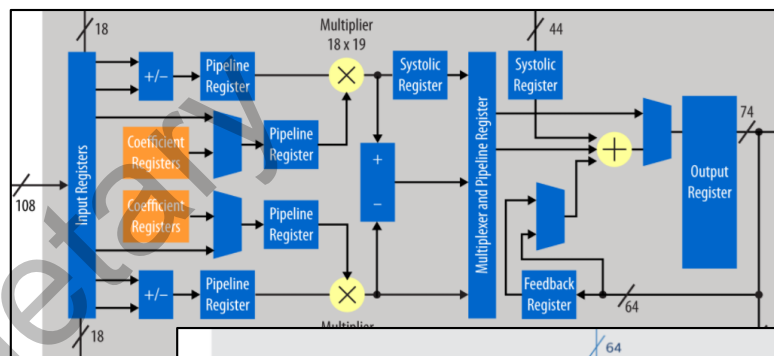
FPGA Architecture: Floating Point Multiplier/Adder Blocks



DSP Blocks

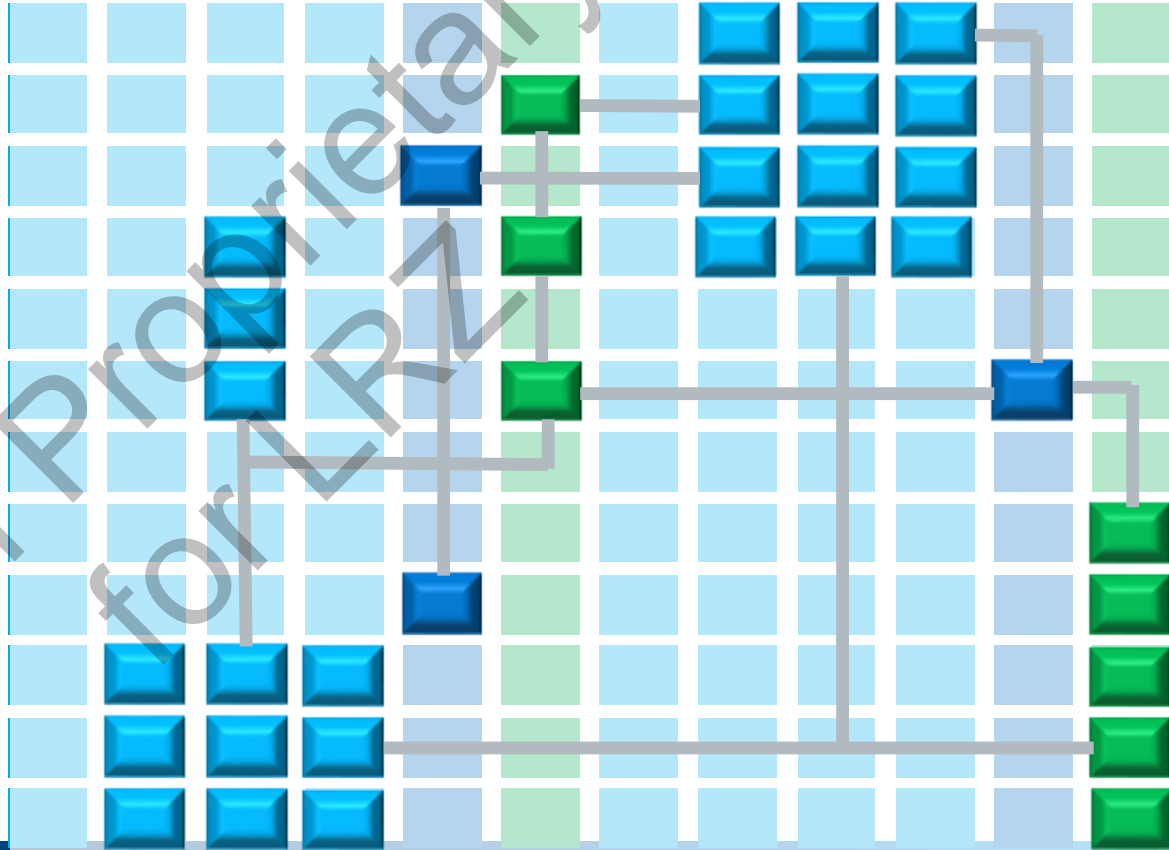
Thousands DSP Blocks in Modern FPGAs

- Configurable to support multiple features
 - Variable precision fixed-point multipliers
 - Adders with accumulation register
 - Internal coefficient register bank
 - Rounding
 - Pre-adder to form tap-delay line for filters
 - Single precision floating point multiplication, addition, accumulation



FPGA Architecture: Configurable Routing

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



Intel Proprietary

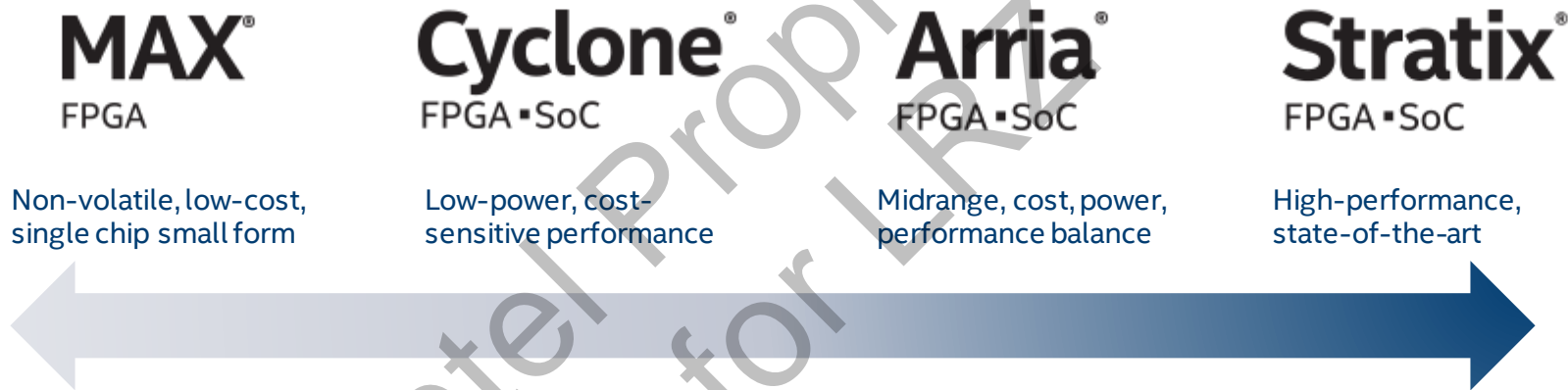
FPGA I/Os and Interfaces

FPGAs have flexible IO features to support many IO and interface standards

- Hardened Memory Controllers
 - Available interfaces to off-chip memory such as HBM, HMC, DDR SDRAM, QDR SRAM, etc.
- High-Speed Transceivers
- PCIe* Hard IP
- Phase Lock Loops

Intel® FPGA Product Portfolio

Wide range of FPGA products for a wide range of applications



- Products features differs across families
 - Logic density, embedded memory, DSP blocks, transceiver speeds, IP features, process technology, etc.

Mapping a Simple Program to an FPGA

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



CPU instructions

R0 ← **Load** Mem[100]

R1 ← **Load** Mem[101]

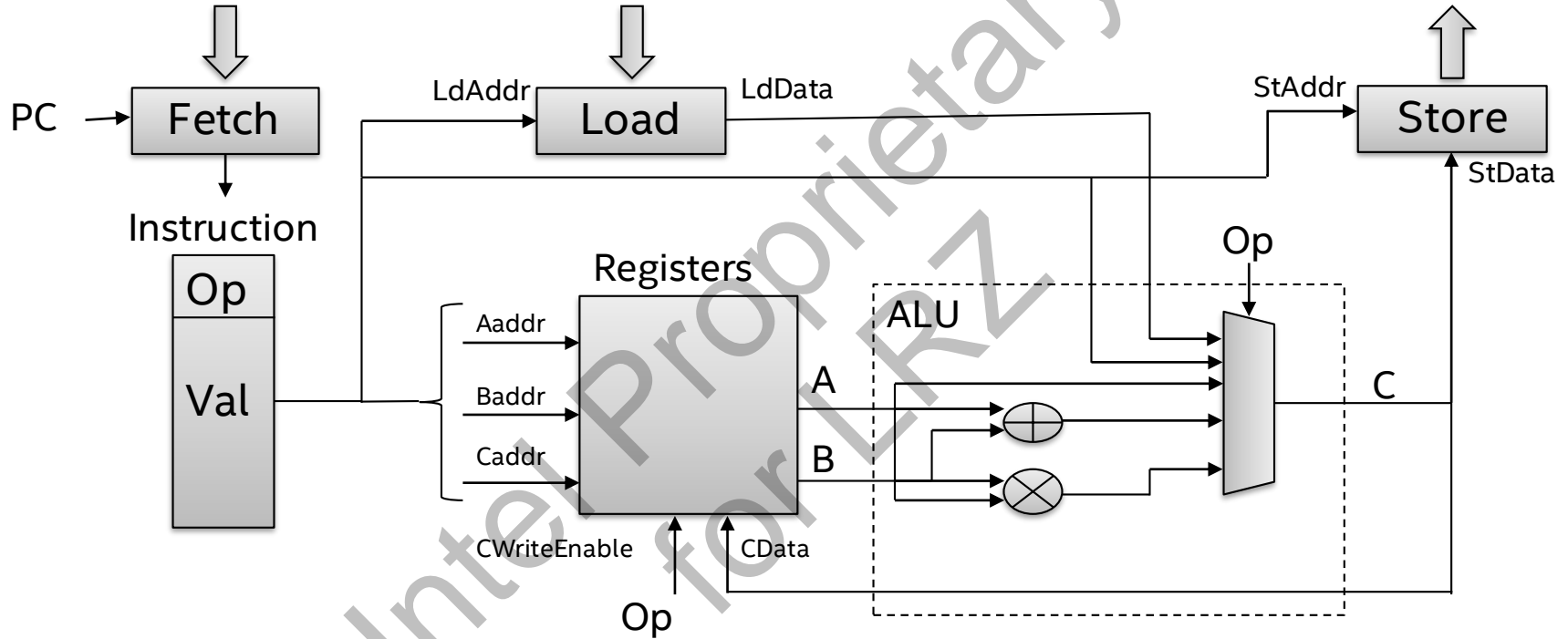
R2 ← **Load** #42

R2 ← **Mul** R1, R2

R0 ← **Add** R2, R0

Store R0 → Mem[100]

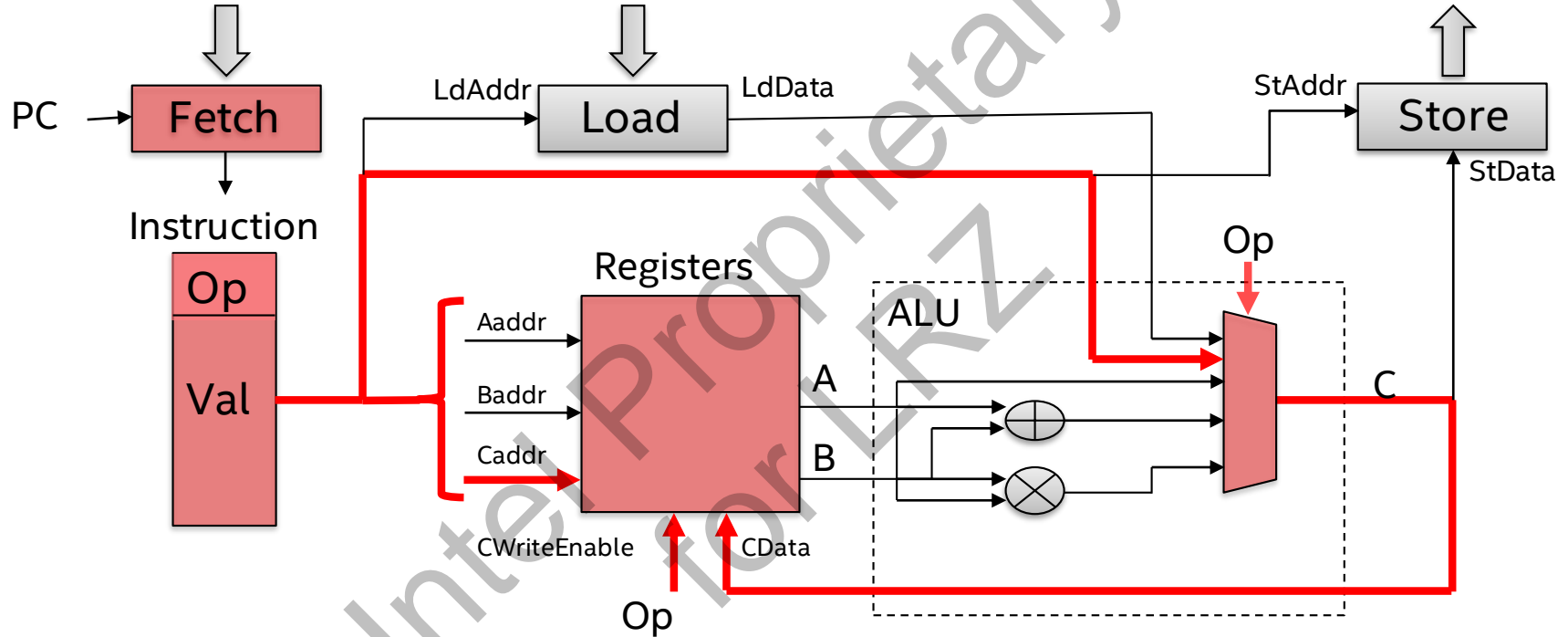
First let's take a look at execution on a simple CPU



Fixed and general architecture:

- General "cover-all-cases" data-paths
- Fixed data-widths
- Fixed operations

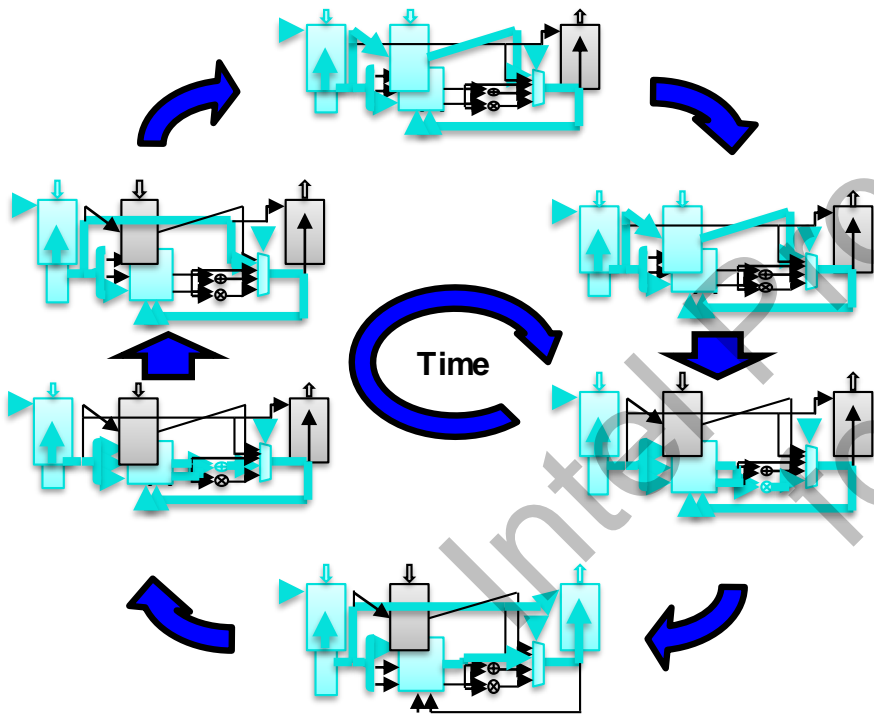
Looking at a Single Instruction



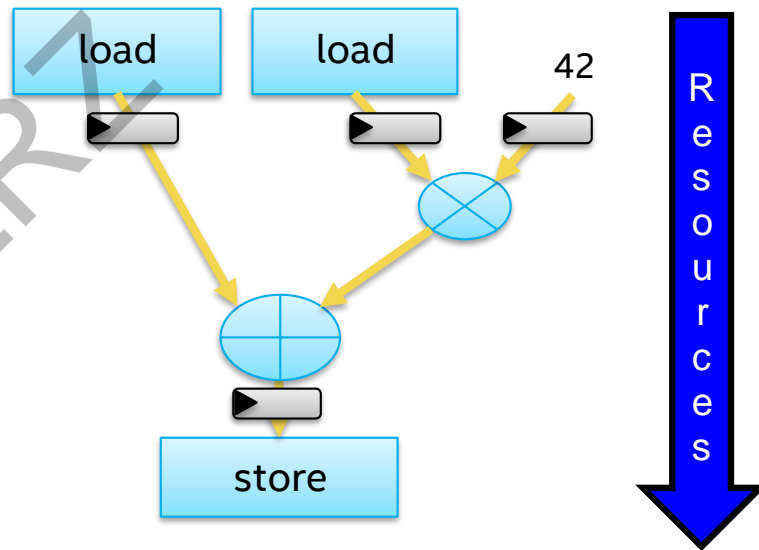
Very inefficient use of hardware!

Sequential Architecture vs. Dataflow Architecture

Sequential CPU Architecture



FPGA Dataflow Architecture

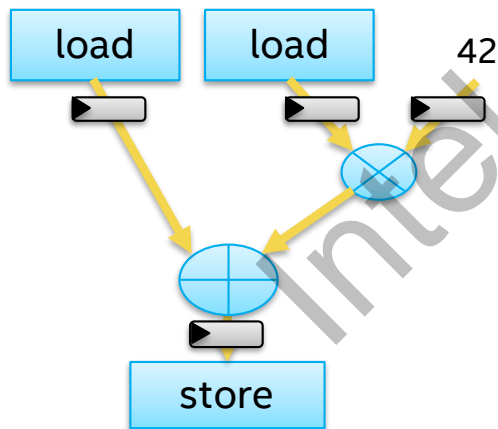


Custom Data-Path on the FPGA Matches Your Algorithm!

High-level code

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

Custom data-path



Build exactly what you need:

Operations

Data widths

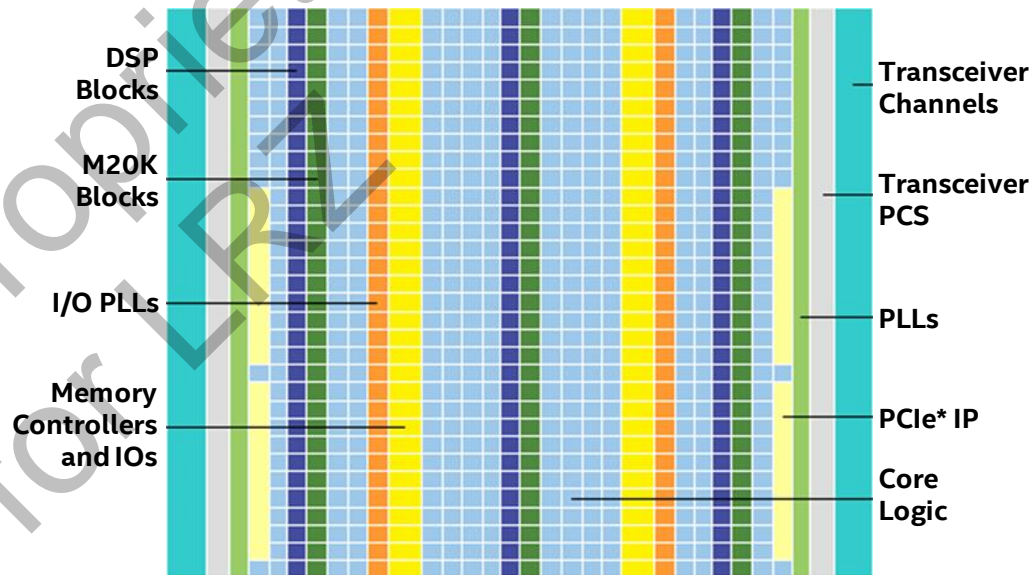
Memory size & configuration

Efficiency:

Throughput / Latency / Power

Advantages of Custom Hardware with FPGAs

- **Custom hardware!**
- Efficient processing
- Fine-grained parallelism
- Low power
- Flexible silicon
- Ability to reconfigure
- Fast time-to-market
- Many available I/O standards



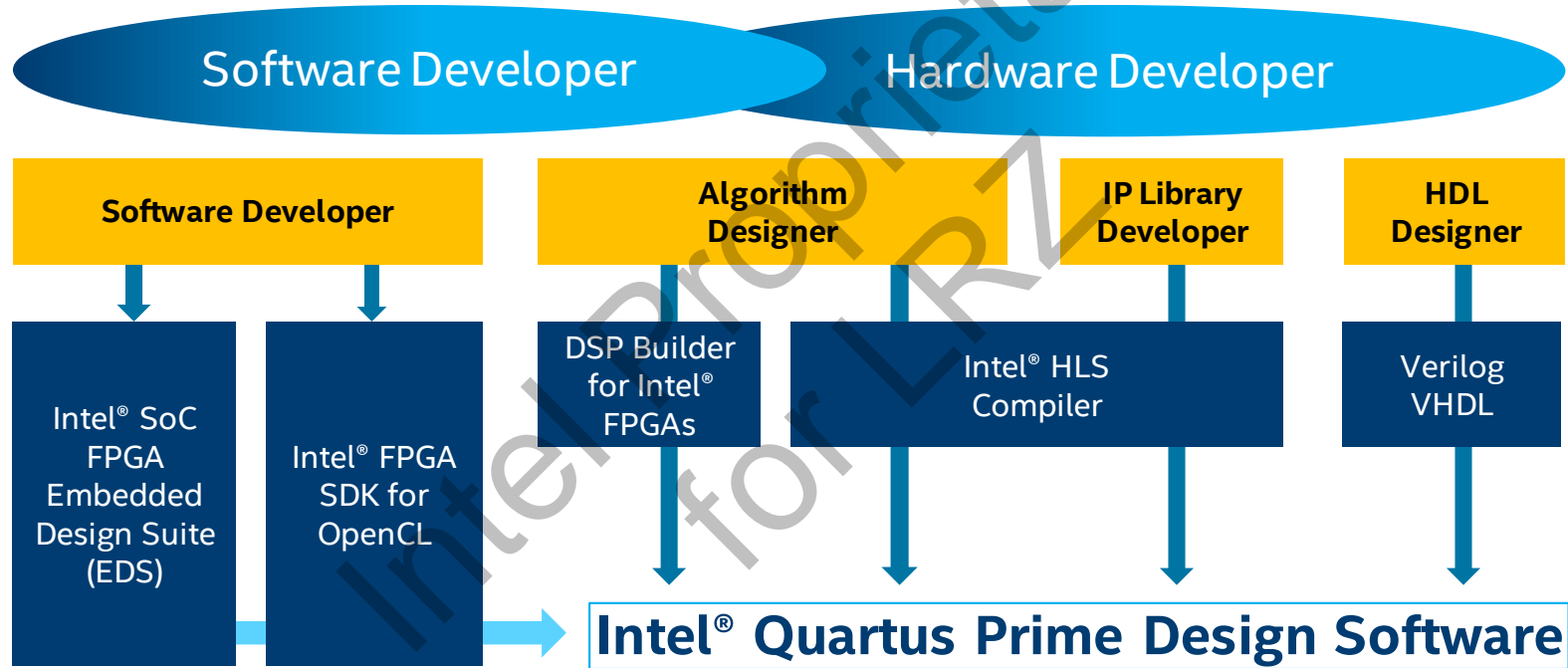


FPGA PROGRAMMING MODEL

RTL

Intel Proprietary
FOU/RZ

FPGA Development and Programming Tools



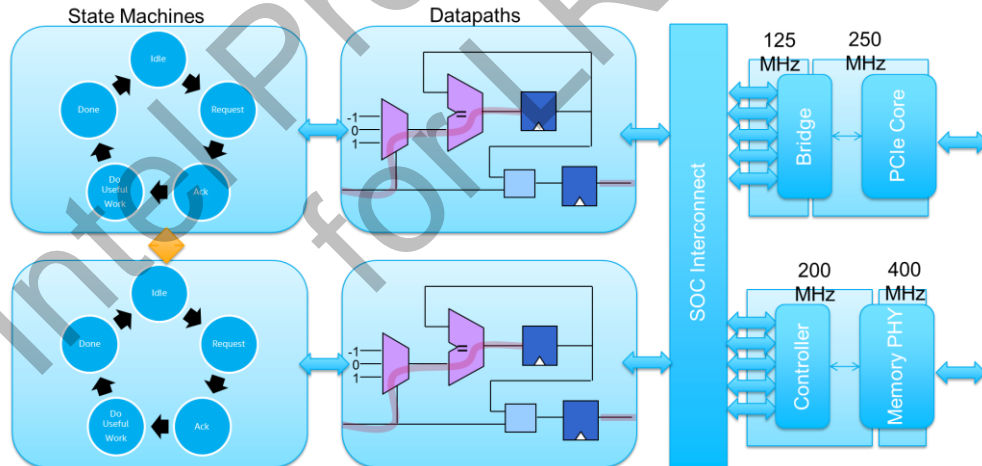
Verilog, VHDL and the Intel® FPGA SDK for OpenCL are currently supported by the Acceleration Stack. High Level Synthesis can be used manually by following app note

Traditional FPGA Design Entry

Circuits described using Hardware Description Languages (HDL) such as VHDL or Verilog
A designer must describe the behavior of the algorithm to create a low-level digital circuit

- Logic, Registers, Memories, State Machines, etc.

Design times range from several months to even years!



Intel® Quartus® Prime Design Software

Default Operating Environment

Project Navigator

Tasks window

The screenshot displays the Intel Quartus Prime Design Software interface. The main window is titled "Quartus Prime Pro Edition - C:/altera_trn/Quartus_Prime_Foundation_17_1_v1/QP17_1/VHDL/pipemult - pipemult_lc". The interface is divided into several panes:

- Project Navigator:** Located on the left, it shows a tree view of the project structure. The "Instance" column lists "mult_inst" and "ram_inst", and the "Entity" column lists "mult" and "ram".
- Tasks window:** Located below the Project Navigator, it shows a list of tasks such as "Revisions...", "Project Files", "Assignments", and "Compilation".
- Tool View window:** The central pane displays the VHDL code for the "pipemult.vhd" file. The code includes a library declaration for "ieee" and "std_logic_1164", an entity declaration for "pipemult", and an architecture block for "bdf_type".
- IP Catalog:** Located on the right, it shows a list of installed IP blocks, including "Basic Functions", "DSP", "Interface Protocols", "Low Power", "Memory Interfaces and Controllers", "Processors and Peripherals", and "University Program".
- Messages window:** Located at the bottom, it shows the output of the compilation process, including the command "quartus_asm --read_settings_files=off --write_settings_files=off pipemult -c pipemult_lc" and the message "Loading final database".

```
-- All information provided herein is provided on an "as is" basis,
-- without warranty of any kind.
-- Module Name: pipemult
-- Module Function: This file contains the top level module for the p
-- project.
-- REVISION HISTORY:
-- Revision 1.0 12/07/2009 - Initial release
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY pipemult IS
    port (
        clk1      : IN    STD_LOGIC;
        wr_en     : IN    STD_LOGIC;
        dataa     : IN    STD_LOGIC_VECTOR( 15 downto 0);
        datab    : IN    STD_LOGIC_VECTOR( 15 downto 0);
        rdaddress : IN    STD_LOGIC_VECTOR( 5  downto 0);
        wraddress : IN    STD_LOGIC_VECTOR( 5  downto 0);
        q         : OUT   STD_LOGIC_VECTOR(31 downto 0)
    );
END pipemult;

ARCHITECTURE bdf_type OF pipemult IS
    component mult is
        port (
            dataa : in  std_logic_vector(15 downto 0) := (others => 'X';
            datab : in  std_logic_vector(15 downto 0) := (others => 'X';
            clock : in  std_logic
            result : out std_logic_vector(31 downto 0)
```

Messages window

100% 00:05:26

Intel® Quartus® Prime Design Software Projects

Description

- Collection of related design files & libraries
- Must have a designated top-level entity
- Target a single device
- Store settings in the software settings file (**.qsf**)
- Compiled netlist information stored in **qdb** folder in project directory

Create new projects with **New Project Wizard**

- Can be created using Tcl scripts

Intel® FPGA Design Store

Login

Download complete example design templates for specific development kits

Design examples include design files, device programming files, and software code as required

Install **.par** files and select as template in New Project Wizard

<https://cloud.altera.com/devstore/platform/>

The screenshot shows the Intel FPGA Design Store interface. At the top, there is a navigation bar with the Intel FPGA logo and an 'Admin Login' link. Below the navigation bar, there is a breadcrumb trail: 'Dashboard > Design Store > Design Examples'. The main heading is 'Design Store', with a 'Take a tour' button. There are two tabs: 'Design Examples' (selected) and 'Development Kits'. A yellow banner contains the text: 'Looking for more design examples? Find them here.' and 'Interested in contributing content to the design store? Click here.' Below the banner, there are several filter dropdowns: 'Family: Any', 'Category: Any', 'Quartus II Version: 16.0', 'Development Kit: Any', and 'IP Core: Any'. A search bar is located at the bottom right of the filter section. The main content area displays a table of design examples.

	Name	Category	Development Kit	Family	Quartus II Version	Vendor	Downloads	
➔	JPEG Decoder Design Example (OpenCL)	Design Example \ Outside Design Store	Non kit specific: Stratix V Design Examples	Stratix V	16.0.0	Altera	0	🔍
➔	100Gbps Ethernet PHY only Testbench	Design Example \ Outside Design Store	Non kit specific: Stratix V Design Examples	Stratix V	16.0.2	Altera	0	🔍
📄	Accelerated FIR with Built-In Direct Memory Access Example	Design Example	Cyclone V E FPGA Development Kit	Cyclone V	16.0.0	Altera	81	🔍
📄	Adapting Digilent PmodCLP LCD to DE10 Lite Development Kit Arduino Shield Header	Design Example	MAX 10 DE10 - Lite	MAX 10	16.0.0	Altera	49	🔍

Device Selection

Choose device family
& family category
(transceiver options,
SoC options, etc.)

Family, Device & Board Settings

Device **Board**

Select the family and device you want to target for compilation. You can install additional device support with the Install Devices command on the Tools menu. To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: Arria 10 (GX/SX/GT)

Device: Arria 10 GX

Show in 'Available devices' list

Package: FBGA

Pin count: 1932

Core speed grade: 1

Transceiver speed grade: 2

Name filter:

Show advanced devices

Target device

Specific device selected in 'Available devices' list

Other: n/a

Available devices:

Name	Core Voltage	ALM	Total I/Os	GPIOs	HSSI Channels	PCIe Hard IP Blocks	memory bits	M20K	
10AX115S2F45I1SG	0.9V or 0.95V	427200	960	624	72	4	55562240	2713	1
10AX115S2F45I1SG...	0.95V	427200	960	624	72	4	55562240	2713	1
10AX115U2F45E1SG	0.9V or 0.95V	427200	928	480	96	4	55562240	2713	1
10AX115U2F45I1SG	0.9V or 0.95V	427200	928	480	96	4	55562240	2713	1

Filter device list

Choose specific part from list

```
Tcl: set_global_assignment -name FAMILY "device family name"  
Tcl: set_global_assignment -name DEVICE <part_number>
```

Chip Planner

Graphical view of

- Layout of device resources
- Routing channels between device resources
- Global clock regions

Uses

- View placement of design logic
- View connectivity between resources used in design
- Make placement assignments
- Debugging placement-related issues

Chip Planner



Tools menu or toolbar

Tasks window

The screenshot shows the Intel Chip Planner interface with several windows and annotations:

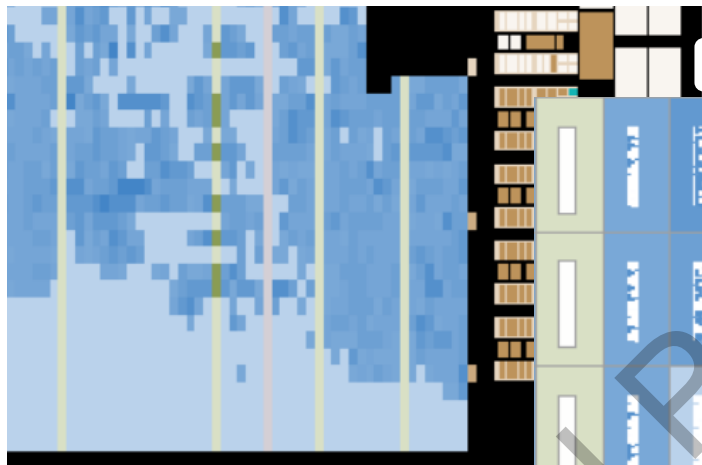
- Report window:** Located at the top left, displaying "Report not available".
- Tasks window:** Located at the bottom left, listing tasks such as "Toggle Background C...", "Report Resources...", "Report Compilation M...", "Mark Selection", "Core Reports", "Report High-Speed...", "Report Routing Ut...", "Clock Reports", and "Report Used Clock...".
- Device floorplan aka Chip View:** The central workspace showing a floorplan with a blue vertical bar representing a memory block.
- Memory block in use:** A blue arrow points to the blue vertical bar in the floorplan.
- Unused LAB:** A blue arrow points to a light blue area in the floorplan.
- Selected Node Properties:** A window on the right showing properties for the selected element "ram_block5a0".
- Layers Settings:** A blue arrow points to the "Layers Settings" tab in the "Selected Node Properties" window.

Properties/Modes	
Full Name	dp_core_0 vip vfb vfb pkt_trans_r
Full Name with entity	N/A
Coordinate	(84, 155)

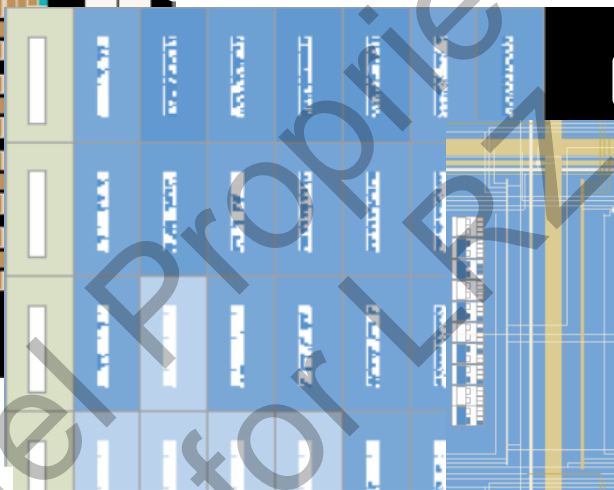
RAM	
roperties	Fan-in Fan-out
je Properties	Layers Settings Color Legend

Floorplan Views

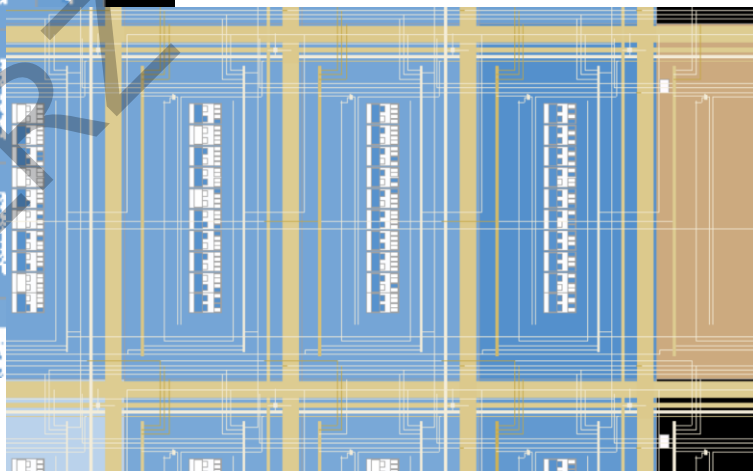
Overall device resource usage



Lower level block usage



Lowest level routing detail



Zoom in for detailed logic implementation & routing usage

Pin Planner



Interactive graphical tool for assigning pins

- Drag & drop pin assignments
- Set pin I/O standards
- Reserve future I/O locations

Default window panes

- Package View
- All Pins list
- Groups list
- Tasks window
- Report window

Assignments menu → **Pin Planner**,
toolbar, or **Tasks** window

Pin Planner Window

Toolbar

The screenshot shows the Pin Planner window for an Arria 10 - 10AX115S2F4511SG chip. The interface includes a toolbar on the left, a Groups list, a Tasks pane, a Package View, a Pin Legend, and an All Pins list.

Groups list

Node Name	Direction
dataa[15..0]	Input Group
dataa[14..0]	Input Group
q[31..0]	Outp..roup
rdaddr...[5..0]	Input Group
wraddr...[5..0]	Input Group
<<new group>>	

Tasks pane

- Early Pin Planning
 - Early Pin Planning...
 - Run I/O Assignment
 - Export Pin Assignmen
 - Pin Finder...
- Clock Pins
 - Clock

Package View

Top View - Flip Chip
Arria 10 - 10AX115S2F4511SG

Pin Legend

Symbol	Pin Type
○	User I/O
●	User assign...
●	Fitter assign...
○	Unbonded ...
●	Reserved pin
○	DQ
○	DQS
○	DQSB
○	CLK_n
○	CLK_p
○	GX_X*n
○	GX_X*p
○	TEMPDIODE
○	VSIG
○	Other PLL
○	MSELO
○	MSEL1
○	MSEL2
○	CONF_DONE
○	DCLK
○	nCE
○	nCONFIG

All Pins list

Node Name	Direction	Location	I/O Bank	Fitter Locator	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
clk1	Input			PIN_G26	1.8 V		12mA ...ault)		
dataa[15]	Input			PIN_G26	1.8 V		12mA ...ault)		
dataa[14]	Input						12mA ...ault)		
dataa[13]	Input						12mA ...ault)		
dataa[12]	Input						12mA ...ault)		
dataa[11]	Input			PIN_AV35	1.8 V		12mA ...ault)		
dataa[10]	Input			PIN_AV33	1.8 V		12mA ...ault)		
dataa[9]	Input			PIN_AU35	1.8 V		12mA ...ault)		

The Programmer



Tools menu → Programmer

Hardware Setup... USB-Blaster [USB-0] JTAG Progress: []

Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/Configure	Verify	Blank-Check	Examine	Security Bit	Erase
output_files/bleinking_led.sof	10AX115S2F45	307FED2F	307FED2F	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<none>	5M2210Z	00000000	<none>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

TDI → [10AX115S2F45] → [5M2210Z] → TDO

State Machine Editor

File menu → New or Tasks window
Select *State Machine File (.smf)*

Create state machines in GUI

- Manually by adding individual states, transitions, and output actions
- Automatically with State Machine Wizard (Tools menu & toolbar)

Generate state machine HDL code (required)

- VHDL
- Verilog
- SystemVerilog

The screenshot displays the State Machine Editor interface. On the left, there are panels for 'Input Table' and 'Output Table'. The 'Input Table' lists 'clock', 'reset', 'input1', and 'input2'. The 'Output Table' lists 'output1'. The main workspace shows a state transition diagram with three states: 'state1' (blue), 'state2' (purple), and 'state3' (green). Transitions are labeled with conditions like 'input1 & ~input2', 'input1 | input2', and 'OTHERS'. A blue arrow points to the 'OTHERS' transition label with the text: 'Double-click states & transitions to edit properties: name, equations, actions'. At the bottom, a 'State Table' lists transitions with columns for 'state', 'Destination State', and 'Transition (In Verilog or VHDL OTHERS)'. The table contains six entries:

state	Destination State	Transition (In Verilog or VHDL OTHERS)
1	state2	input1 & ~input2
2	state1	OTHERS
3	state3	input1 input2
4	state1	OTHERS
5	state1	OTHERS
6	state3	input1 & input2

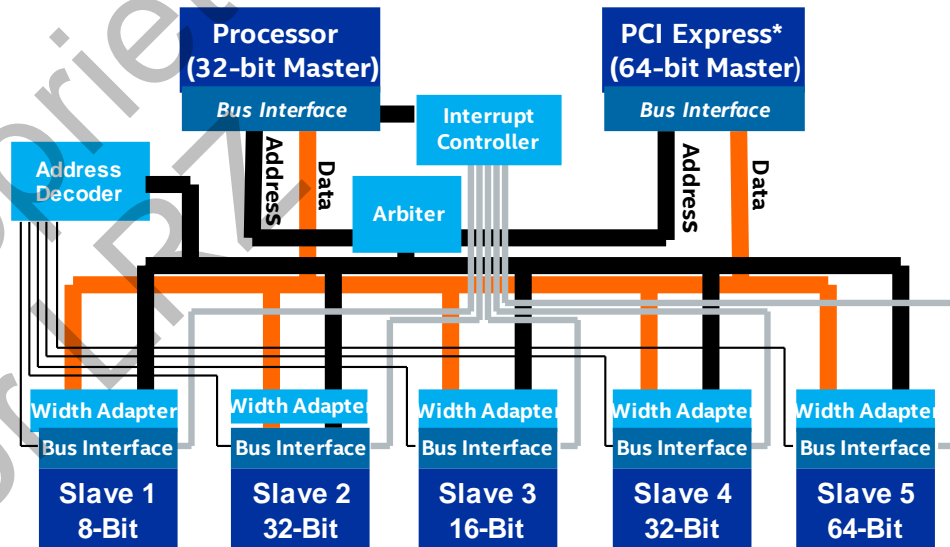
Navigation tabs at the bottom include 'General', 'Inputs', 'Outputs', 'States', 'Transitions', and 'Actions'.

Platform Designer

Components in system use different interfaces to communicate (some standard, some non-standard)

Typical system requires significant engineering work to design custom interface logic

Integrating design blocks and intellectual property (IP) is tedious and error-prone



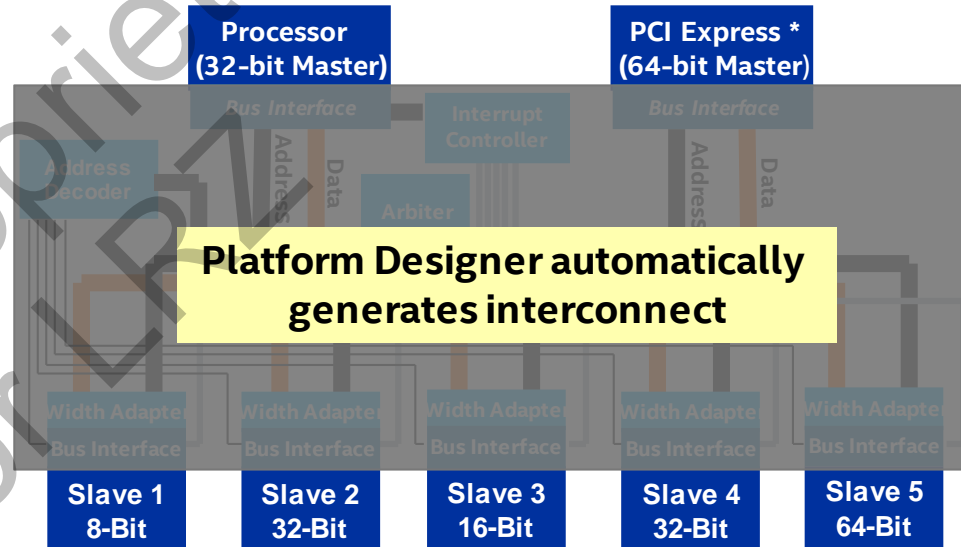
Automatic Interconnect Generation

Avoids error-prone integration

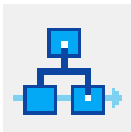
Saves development time with automatic logic & HDL generation

Enables you to focus on value-add blocks

Platform Designer improves productivity by automatically generating the system interconnect logic



The Platform Designer GUI



Access in Tools menu, toolbar, or Tasks window

The screenshot displays the Platform Designer GUI with several key components:

- System Contents:** A central window showing a circuit diagram and a table of components. A blue arrow points to the tabs at the top of this window, labeled "Draggable, detachable tabs".
- Hierarchy:** A left-hand pane showing a tree view of the project structure.
- Messages:** A bottom pane showing a list of messages, including warnings and errors.

Component	Description	Export
clk_clk_in	Clock Source	clk_clk_in
clk_clk_in_reset	Clock Input	clk_clk_in_reset
clk	Clock Output	clk
clk_reset	Reset Output	clk_reset
pll	Altera PLL	pll
refclk	Clock Input	refclk
reset	Reset Input	reset
outclk0	Clock Output	outclk0
outclk1	Clock Output	outclk1
external_connection	Conduit	external_connection
push_button_reg	Avalon Push Button RW Register	pushbutton_switches
clock	Clock Input	clock
reset	Reset Input	reset
buttonreg	Avalon Memory Mapped Slave	buttonreg
buttonregm	Avalon Memory Mapped Master	buttonregm
av_sm_master	Avalon State Machine Master	av_sm_master
clock	Clock Input	clock
reset	Reset Input	reset
avalon_master	Avalon Memory Mapped Master	avalon_master
led_out	LED Flasher	led_out
clock	Clock Input	clock
reset	Reset Input	reset
st_sink	Avalon Streaming Sink	st_sink
debug	Avalon Memory Mapped Slave	debug

System Contents

Hierarchy

Messages

0 Errors, 5 Warnings

Generate HDL... Finish

Draggable,
detachable tabs

RTL FLOW

Lab 1

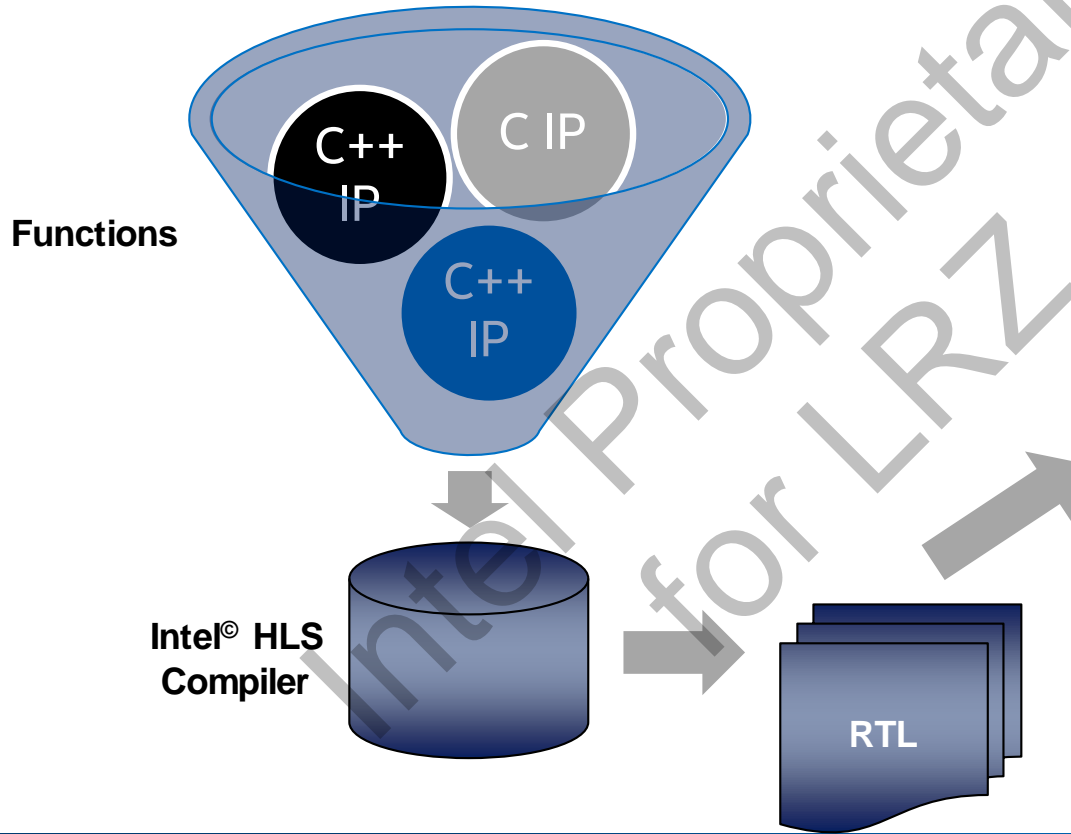
Intel Proprietary
for LRZ

FPGA PROGRAMMING MODEL:

High Level Synthesis

Intel Proprietary
for PERZ

Can Also Be Wrapped With Higher Level Flows



Use	Connections	Name	Description
<input checked="" type="checkbox"/>		clk clk_in clk_in_reset clk_reset clk_output	Clock Source Clock Input Reset Input Clock Output Reset Output
<input checked="" type="checkbox"/>		master_0 clk clk_reset master master_reset	JTAG to Avalon Master Bridge Clock Input Reset Input Avalon Memory Mapped Master Reset Output
<input checked="" type="checkbox"/>		sys_0 nios_sys clk reset onchip_memory2_0 onchip_memory2_0 onchip_memory2_0 mm_bridge_0_m0	Reset Input Clock Input Reset Input Reset Input Clock Input Avalon Memory Mapped Slave Avalon Memory Mapped Master
<input checked="" type="checkbox"/>		address_span_exten... clock reset windowed_slave expanded_master cntl	Address Span Extender Clock Input Reset Input Avalon Memory Mapped Slave Avalon Memory Mapped Master Avalon Memory Mapped Slave
<input checked="" type="checkbox"/>		address_span_exten... clock reset windowed_slave expanded_master cntl	Address Span Extender Clock Input Reset Input Avalon Memory Mapped Slave Avalon Memory Mapped Master Avalon Memory Mapped Slave
<input checked="" type="checkbox"/>		hps_sys h2f_reset h2f_user0_clock f2h_axi_clock f2h_axi_slave f2h_axi_slave h2f_axi_clock h2f_axi_master f2h_sdram_0_data f2h_sdram_0_clock memory hps_io	Hard Processor System Reset Output Clock Output Clock Input AXI Slave AXI Slave Clock Input AXI Master AXI Slave Clock Input Conduit Conduit

Platform Designer

The Software Programmer's View



Programmers develop in mature software environments

- Ideas can easily be expressed in languages such as 'C'
- Typically start with simple sequential program
- Use parallel APIs / language extensions to exploit multi core for additional performance
- Compilation times are almost instantaneous
- Immediate feedback
- Rich debugging tools

High Level Design is the Bridge Between HW & SW

100x More Software Engineers than Hardware Engineers

Key to wide-spread adoption of FPGA in Datacenter

Debugging software is much faster than hardware

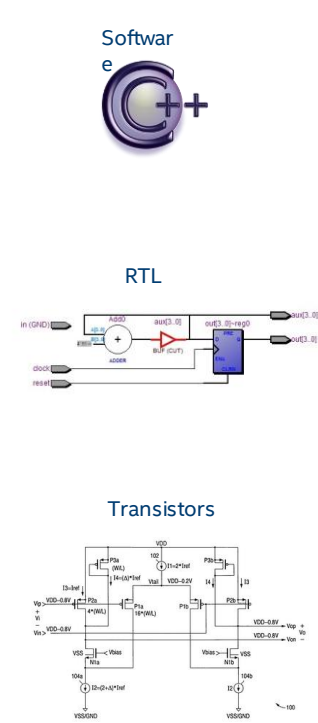
Many functions are easier to specify in software than RTL

Simulation of RTL takes thousands times longer than software

Design Exploration is much easier and faster in software

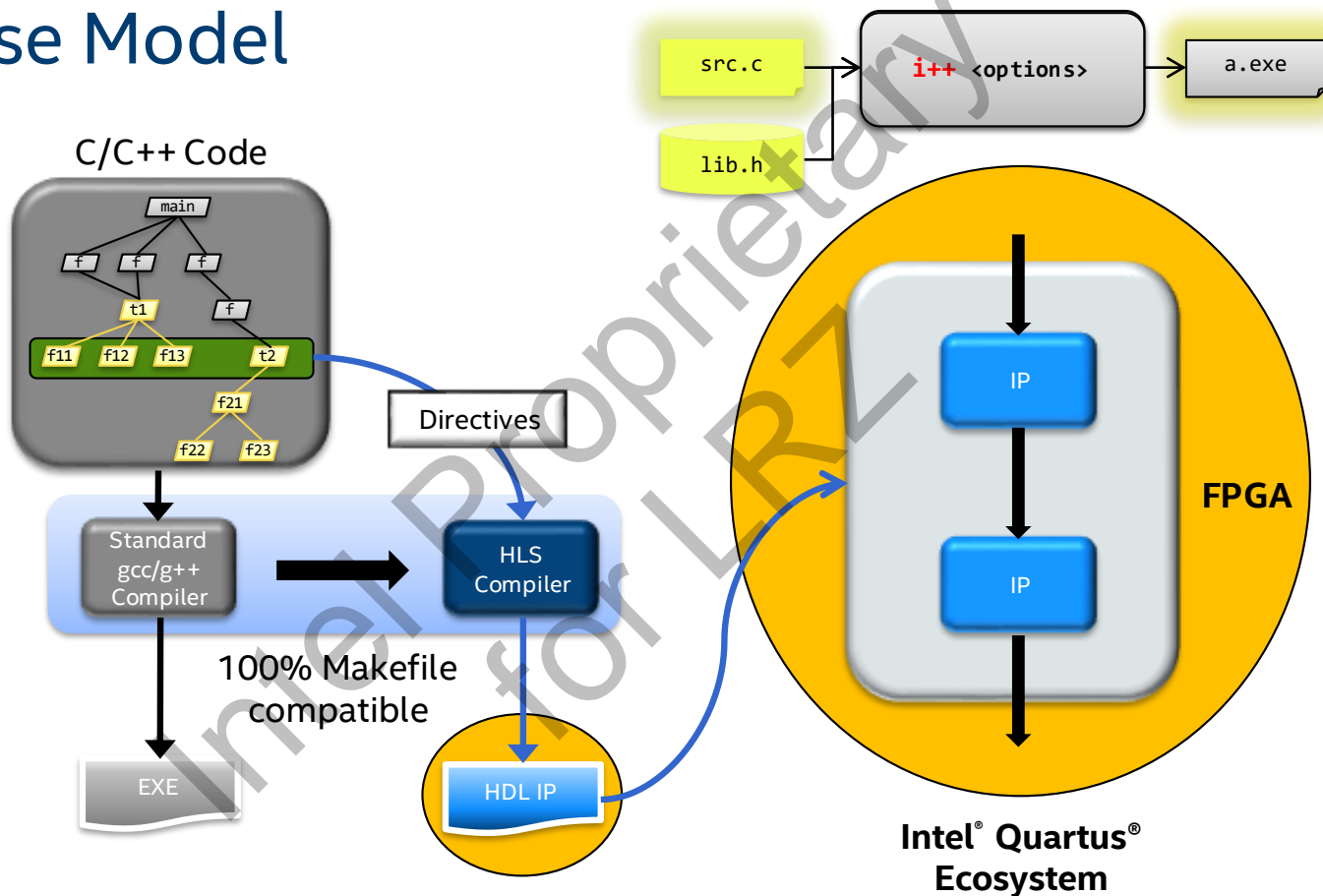
We Need to Raise the Level of Abstraction

- Similar to what assembly programmers did with C over 30 years ago
 - (Today) Abstract away FPGA Design with Higher Level Languages
 - (Today) Abstract away FPGA Hardware behind Platforms
 - (Tomorrow) Leverage Pre-Compiled Libraries as Software Services



Abstraction and Productivity

HLS Use Model



Intel® HLS Compiler

Targets Intel® FPGAs

Command-line executable: `i++`

Builds an IP block

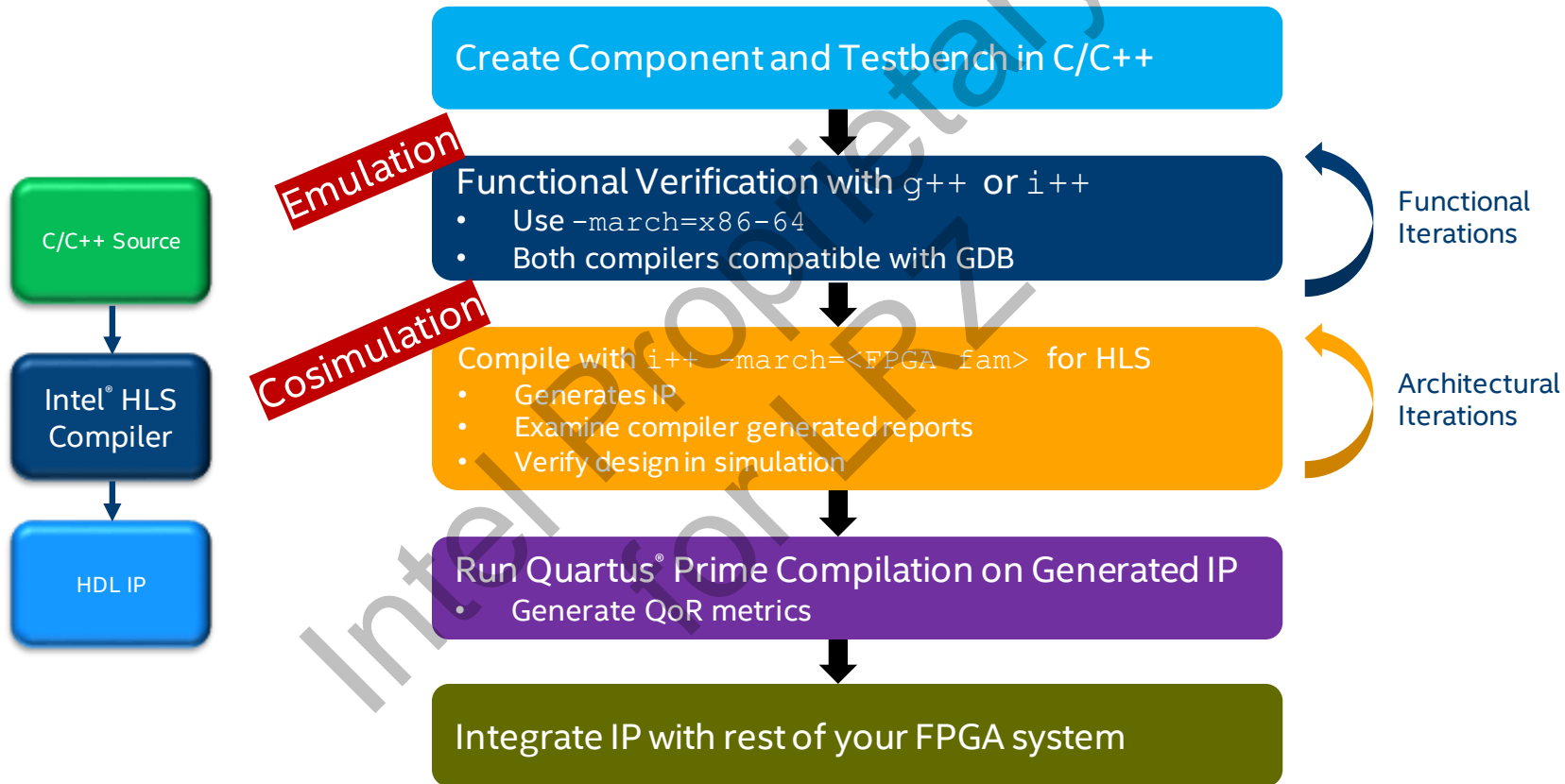
- To be integrated into a traditional FPGA design using FPGA tools



Leverages standard C/C++ development environment

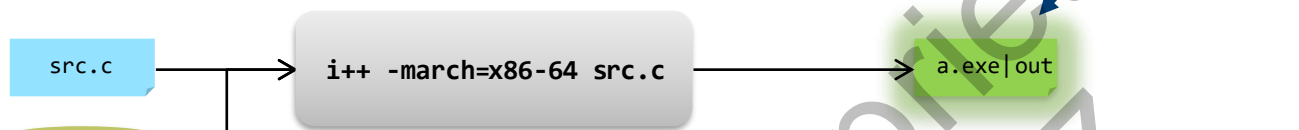
Goal: Same performance as hand-coded RTL with 10-15% more resources

HLS Procedure

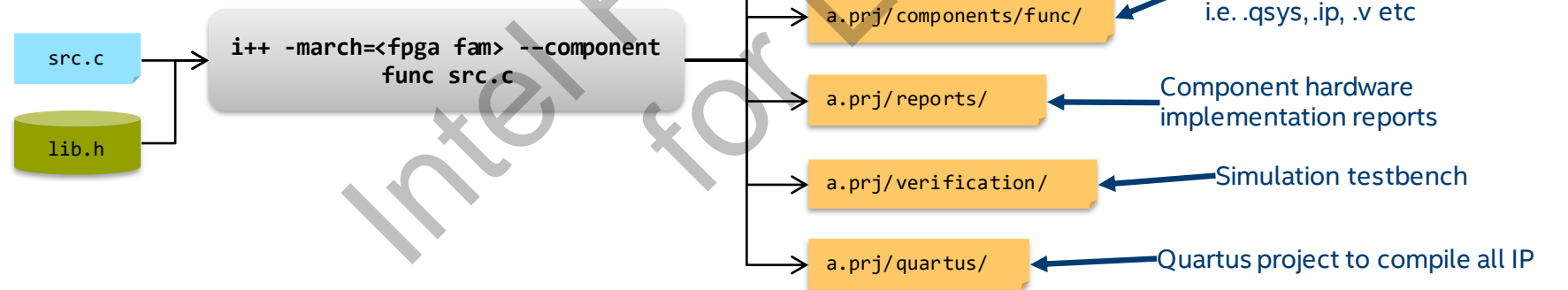


Intel® HLS Compiler Usage and Output

Develop with C/C++:

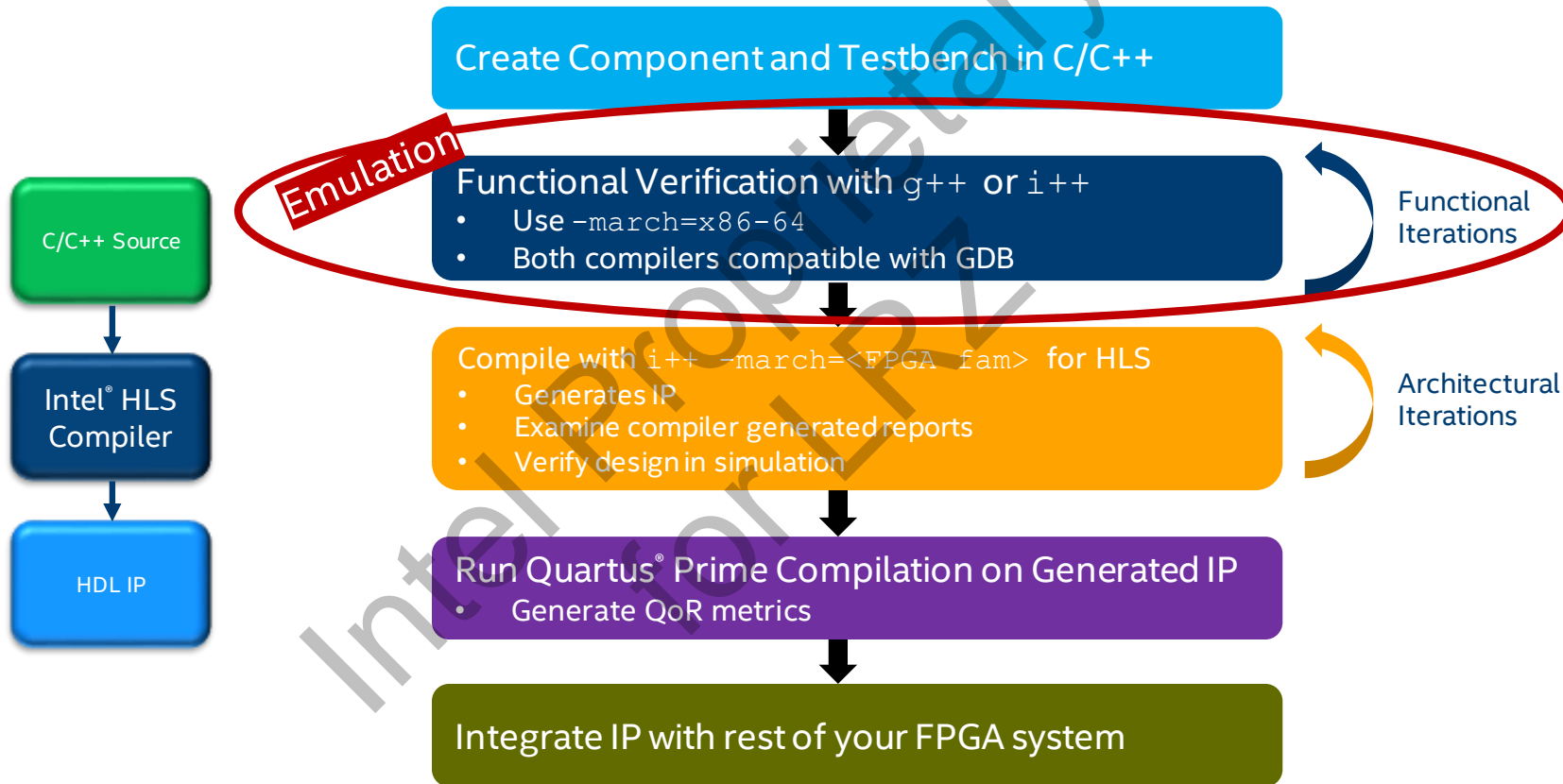


Run Compiler for HLS:



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

HLS Procedure: x86 Emulation



Simple Example Program: `i++` and `g++` flow

Example Program

```
// test.cpp
#include <stdio.h>

int main() {
    printf("Hello world\n");
    return 0;
}
```

Terminal Commands and Outputs

```
$ g++ test.cpp
$ ./a.out
Hello world
$
```

```
$ i++ test.cpp
$ ./a.out
Hello world
$
```

Using the default `-march=x86-64`

g++ Compatibility

Intel HLS Compiler is command line compatible with g++

- Similar command-line flags, x86 behavior, and compilation flow
- Changing “g++” to “i++” should just work
 - `g++ <flags> <src>`
 - `i++ <flags> <src>`
- x86 behavior should match g++
 - Except for integer promotion (discussed later)
- No source modifications required (for x86 mode)
- Support for GNU Makefiles

i++ Options : g++ Compatible Options

Option	Description
-h	Display help information
-o <name>	Specify a non-default output name
-c	Instructs compiler generate the object files and not the executable
-march=<arch>	Compile for architecture x86-64 (Default) or <FPGA Family>
-v	Verbose mode
-g	Generate debug information (default)
-g0	Do not generate debug information
-I<dir>	Add to include path
-D<macro> [=<val>]	Define <macro> with <val> or 1
-L<dir> -l<library>	Library search directory and library name when linking

Example: `i++ -march=x86-64 myfile.cpp -o myexe`

i++ Options: FPGA Related Options

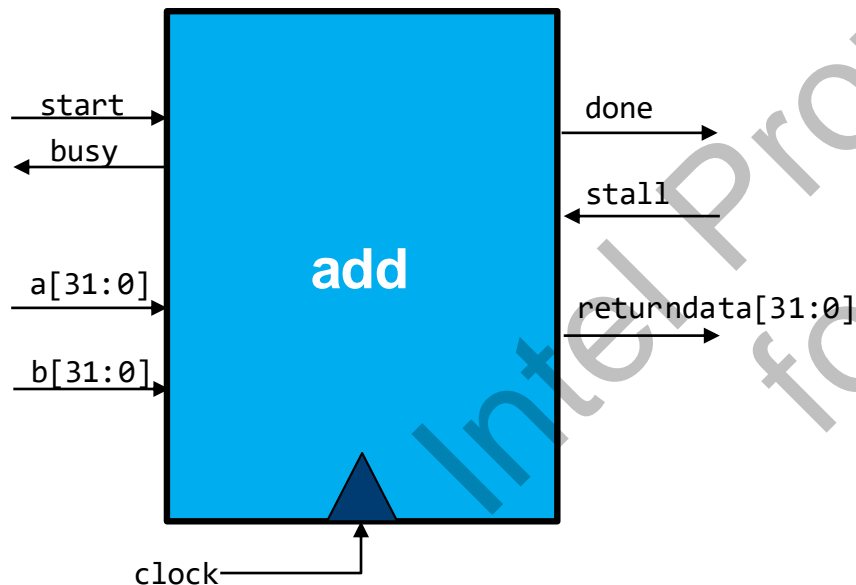
Option	Description
<code>--component <components></code>	Specify a comma-separated list of function names to be synthesized to RTL
<code>--clock <clock_spec></code>	Optimizes the RTL for the specified clock frequency or period
<code>-ghdl</code>	Enable full debug visibility and logging of all signals when verification executable is run
<code>--quartus-compile</code>	Compiles the resulting HDL files using the Intel® Quartus® Prime software
<code>--simulator <simulator></code>	Specify the simulator used for verification, "none" to skip testbench generation
<code>--x86-only</code>	Only create the executable for testbench, no RTL or cosim support
<code>--fpga-only</code>	Create FPGA component project, RTL and cosim support, no testbench binary

Example: `i++ -march=<fpga fam> --component mycomp --clock 400Mhz myfile.cpp`

There are many other optimization options available please see the *Intel HLS Compiler Reference Manual*

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

Example Makefile

```
FILE      := myapp
DEVICE    := Arria10

all:

gpp: $(FILE).cpp
    g++ $(GCFLAGS) $(FILE).cpp -o $(FILE).out

emu: $(FILE).cpp
    i++ $(GCFLAGS) $(FILE).cpp -o $(FILE)_emu.out

fpga: $(FILE).cpp
    i++ $(GCFLAGS) $(FILE).cpp -o $(FILE)_fpga.out -march=$(DEVICE)
```

x86 Debugging Tools

printf/cout
gdb
Valgrind



Using `printf()`

Requires “HLS/stdio.h”

- Maps to `<stdio.h>` when appropriate

Can be included in the testbench or the component

- Used with no limitations in the x86 emulation flow

`printf` statements inside the **component** ignored for HDL generation

- Ignored in the cosimulation flow with an HDL simulator

Using printf(): Example

Example Program

```
// test.cpp
#include "HLS/stdio.h"

void say_hello() {
    printf("Hello from the component\n");
}

int main() {
    printf("Hello from the testbench\n");
    say_hello();
    return 0;
}
```

Terminal Commands and output

```
$ i++ test.cpp
$ ./a.out
Hello from the testbench
Hello from the component
$
```

```
$ i++ test.cpp -march=Arria10 \
    --component say_hello
$ ./a.out
Hello from the testbench
$
```

Debugging Using gdb

i++ integrates well with GNU gdb

- Debug data is generated by default
 - Unlike g++, -g enabled by default, use -g0 to turn off debug data

`-march=x86-64` flow:

- Can step through any part of the code (including the component)

`-march=<fpga family>` flow:

- Can step through testbench code
- gdb does not see the component side execution (that runs in an HDL simulator)

gdb Example

Example Program

```
// test.cpp
#include "HLS/hls.h"
#include "HLS/stdio.h"

component void say_hello() {
    printf("Hello from the component\n");
}

int main() {
    printf("Hello from the testbench\n");
    say_hello();
    return 0;
}
```

Terminal Commands and output

```
$ i++ test.cpp -march=x86-64 -o test-x86
$ gdb ./test-x86
.....
<GDB Command Prompt>
(gdb)
```

```
$ i++ test.cpp -march=Arria10 -o test-fpga
$ gdb ./test-fpga
.....
<GDB Command Prompt>
(gdb)
```

Debugging with Valgrind

“Valgrind is an instrumentation framework for building dynamic analysis tools.”

- Valgrind tools can detect:
 - Memory leaks
 - Invalid pointer uses
 - Use of uninitialized values
 - Mismatched use of malloc/new vs free/delete
 - Doubly freed memory
- Use to debug component and testbench in the x86 emulation flow



Simple Valgrind Example

Example Program:

```
// test.cpp
1 #include "hls/stdio.h"
2 #include <stdlib.h>
3
4 int bin_count (int *bins, int a) {
5     return ++bins[a];
6 }
7
8 int main() {
9     int *bins = (int *) malloc(16 * sizeof(int));
10    srand(0);
11    for (int i = 0; i < 256; i++) {
12        int x = rand();
13        int res = bin_count (bins, x);
14        printf("Count val: %d\n", res);
15    }
16    return 0;
17 }
```

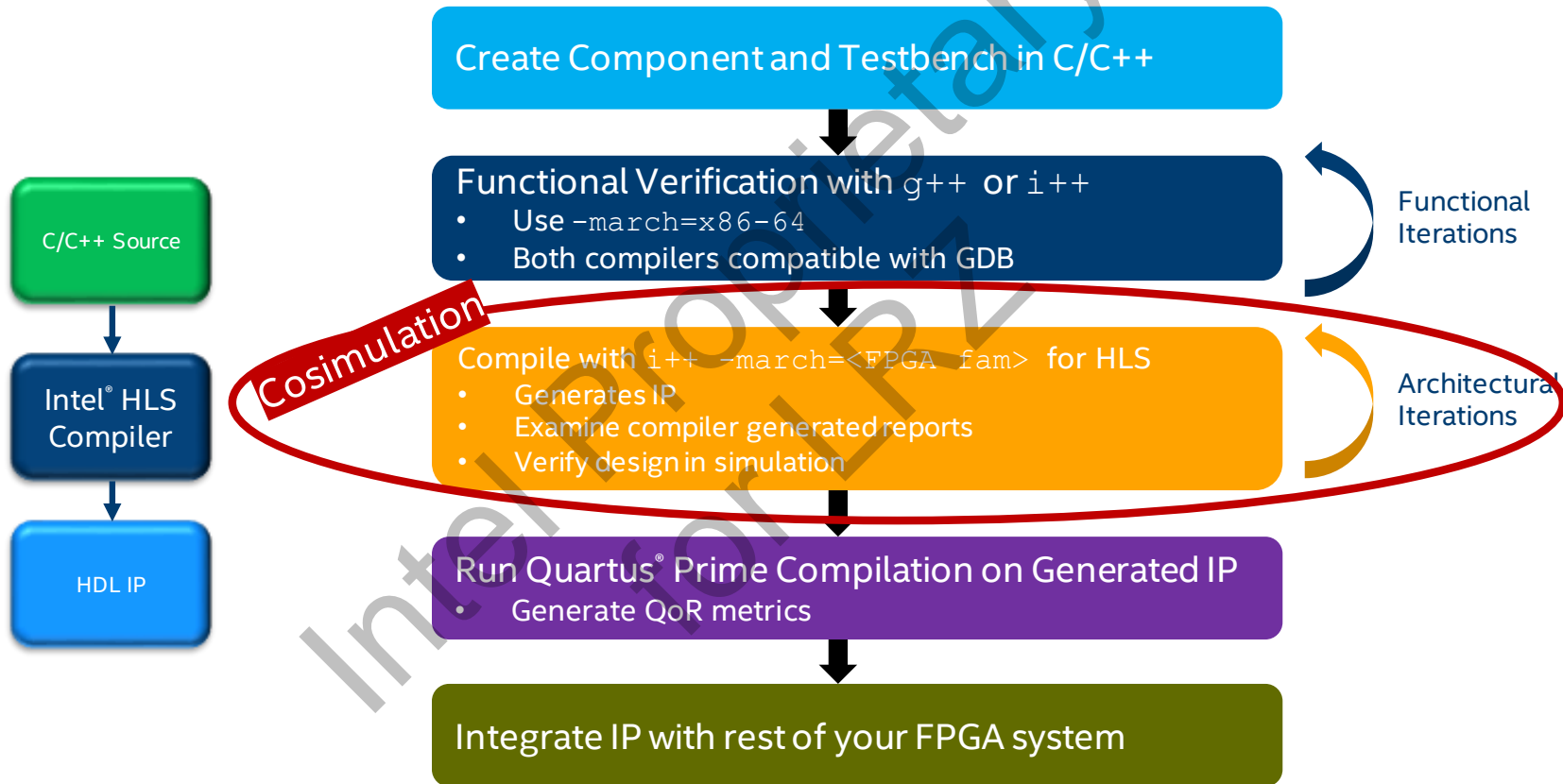
Terminal Commands and output:

```
$ i++ test.cpp
$ ./a.out
Segmentation Fault
$ valgrind --leak-check=full --show-reachable=yes ./a.out
=====
==9744== Invalid read of size 4
==9744==    at 0x4006B3: bin_count(int*, int) (test.cpp:5)
==9744==    by 0x400723: main (test.cpp:13)
==9744==   Address 0x1b31075dc is not stack'd, malloc'd or
(recently) free'd
==9744== Process terminating with default action of signal
11 (SIGSEGV)
==9744== Access not within mapped region at address
0x1B31075DC
==9744==    at 0x4006B3: bin_count(int*, int) (test.cpp:5)
==9744==    by 0x400723: main (test.cpp:13)
=====
==9744== 64 bytes in 1 blocks are still reachable in loss
record 1 of 1
==9744==    at 0x4A06A2E: malloc (vg_replace_malloc.c:270)
==9744==    by 0x4006ED: main (test.cpp:9)
=====
Segmentation fault
```

Valgrind: Segmentation Fault Fixed

```
int bin_count (int *bins, int a) {  
    return ++bins[a % 16];  
}  
  
int main() {  
    int *bins = (int *) malloc(16 * sizeof(int));  
    srand(0);  
    for (int i = 0; i < 256; i++) {  
        int x = rand();  
        int res = bin_count(bins, x);  
        printf("Count val: %d\n", res);  
    }  
    free (bins);  
    return 0;  
}
```

HLS Procedure: Cosimulation



Example Component/Testbench Source

```
#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()

Translation from C function API to HDL module

All component functions are synthesized to HDL

- Each synthesized component is an independent HDL module

Component functions can be declared:

- Using component keyword in source
- Specifying “--component <component_name>” in the command-line

Intel Proprietary
for LRZ

Cosimulation

Combines x86 testbench with RTL simulation

HDL code for the component runs in an RTL Simulator

- Verilog
- RTL testbench automatically created from software

`main()` and everything else called from `main` runs on x86 as the testbench

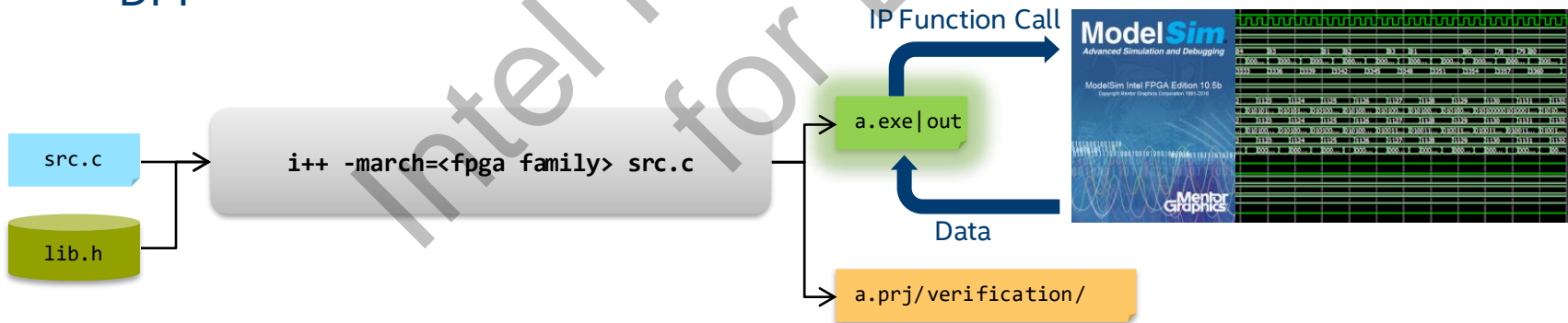
Communication using SystemVerilog Direct Programming Interface (DPI)

- Allows C/C++ to interface SystemVerilog
- Inter-process communication (IPC) library used to pass testbench input data to RTL simulator, and returns the data back to the x86 testbench

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



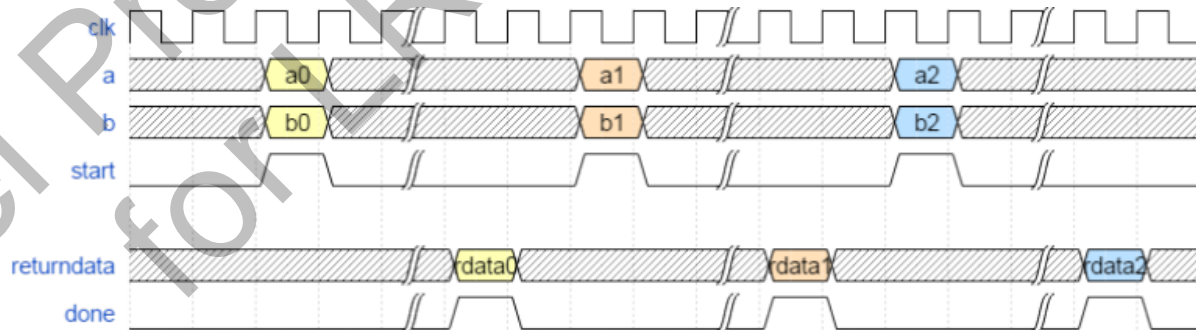
Default Simulation Behavior

Function calls to the simulator are sequential by default

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

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

int main() {
    int x1, x2, x3;
    x1=acc(1, 2);
    x2=acc(3, 4);
    x3=acc(5, 6);
    ...
}
```



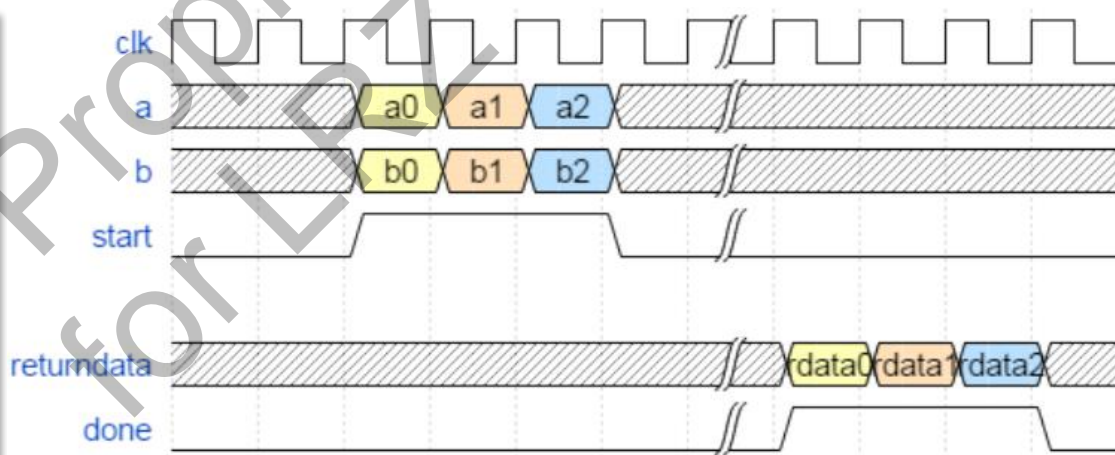
Streaming Simulation Behavior

Use enqueue function calls to stream data into the component

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

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

int main() {
    int x1, x2, x3;
    altera_hls_enqueue(&x1, &acc, 1, 2);
    altera_hls_enqueue(&x2, &acc, 3, 4);
    altera_hls_enqueue(&x3, &acc, 5, 6);
    altera_hls_component_run_all("acc");
    ...
}
```



Viewing Component Waveforms

- Compile design with `i++ -ghdl` flag
 - Enable full visibility and logging of all HDL signals in simulation
- After cosimulation execution, waveform available at `a.prj/verification/vsim.wlf`
- Examine with the ModelSim GUI:
 - `vsim a.prj/verification/vsim.wlf`

Viewing Waveforms in Modelsim

ModelSim - Intel FPGA Edition 10.4

File Edit View Compile Simulate Add Objects Tools Layout Bookmarks Window Help

vsim - Default

Objects

- tb
 - clock_reset_inst
 - component_dpi_control...
 - concatenate_component...
 - my mult_inst
 - my mult_internal_inst
 - split_component_start_in...

Wave - Default

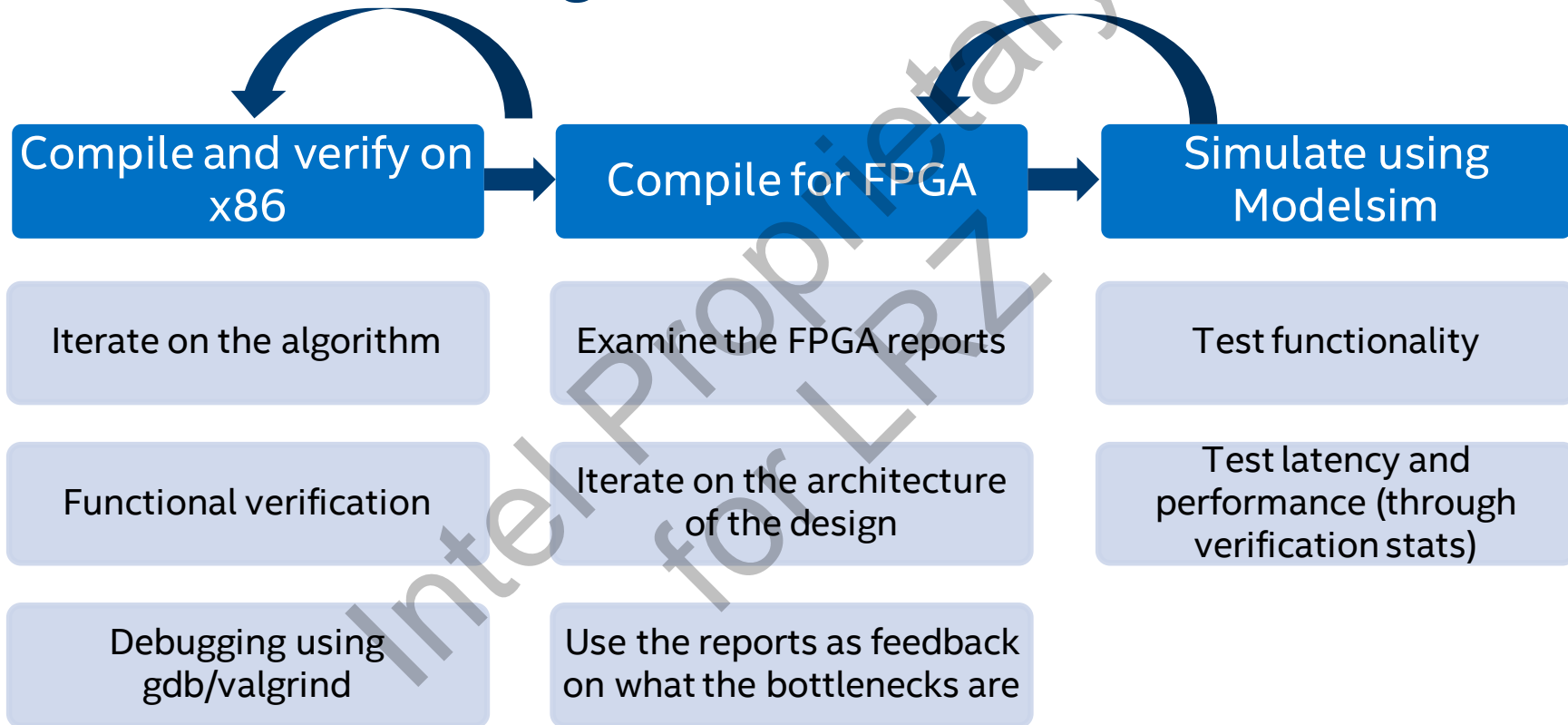
Signal	Msgs
/tb/my mult_inst/clock	St0
/tb/my mult_inst/resetn	St1
/tb/my mult_inst/start	St0
/tb/my mult_inst/busy	St0
/tb/my mult_inst/a	0
/tb/my mult_inst/b	0
/tb/my mult_inst/return...	0
/tb/my mult_inst/done	St0
/tb/my mult_inst/stall	St0

Waveform showing signals: /tb/my mult_inst/clock, /tb/my mult_inst/resetn, /tb/my mult_inst/start, /tb/my mult_inst/busy, /tb/my mult_inst/a, /tb/my mult_inst/b, /tb/my mult_inst/return..., /tb/my mult_inst/done, /tb/my mult_inst/stall.

Locate Component

Add Signals to Waveform

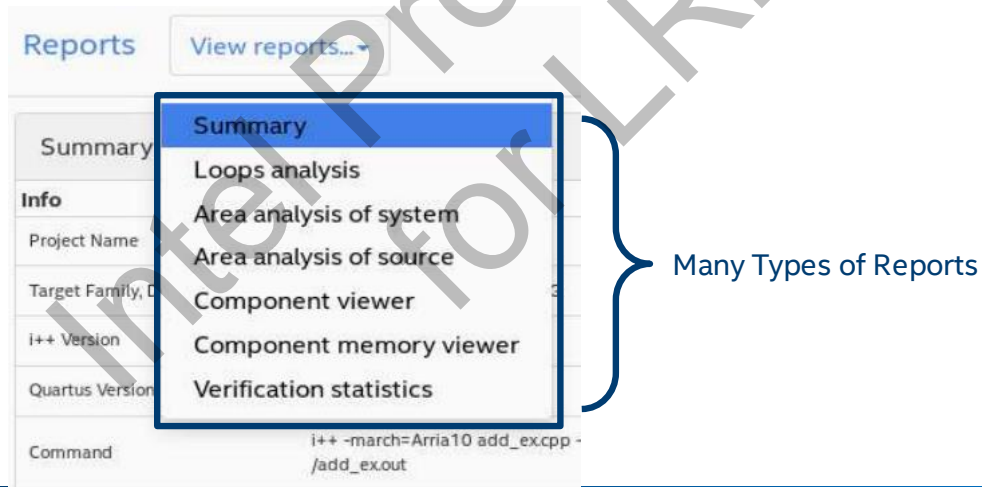
Cosimulation Design Process



Main HTML Report

The Intel® HLS Compiler automatically generates HTML report that analyzes various aspects of your function including area, loop structure, memory usage, and system data flow

- Located at a `.prj/reports/report.html`



HTML Report: Summary

Overall compile statics

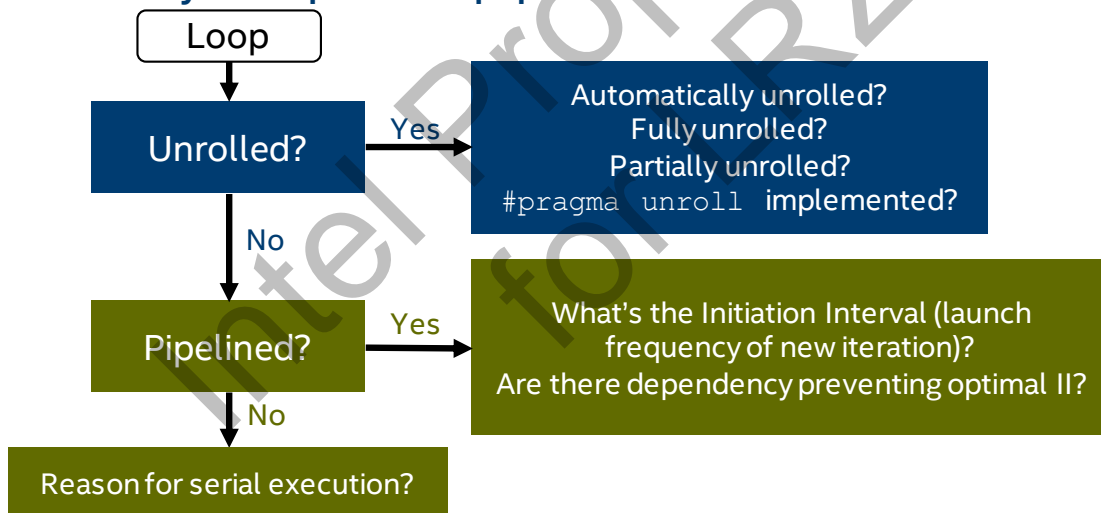
- FPGA Resource Utilization
- Compile Warnings
- Quartus® fitter results
 - Available after Quartus compilation
- etc.

Summary				
Project Name	./fpga/add_ex			
Target Family, Device	Arria 10, 10AX115U1F45I15G			
i++ Version	17.1.0 Build 240			
Quartus Version	17.1.0 Build 240			
Command	i++ -march=Arria10 --component add add_ex.cpp -o ./fpga/add_ex.out			
Reports Generated At	Tue Oct 31 10:18:13 2017			
Quartus Fit Clock Summary				
	1x clock fmax			
Frequency (MHz)	612.75			
Quartus Fit Resource Utilization Summary				
	ALMs	FFs	RAMs	DSPs
add	18	3	0	0
Estimated Resource Usage				
Component Name	ALUTs	FFs	RAMs	DSPs
add	38	2	0	0
Total	38 (0%)	2 (0%)	0 (0%)	0 (0%)
Available	854400	1708800	2713	1518
Compile Warnings				

HTML Report: Loops

Serial loop execution hinders function dataflow circuit performance

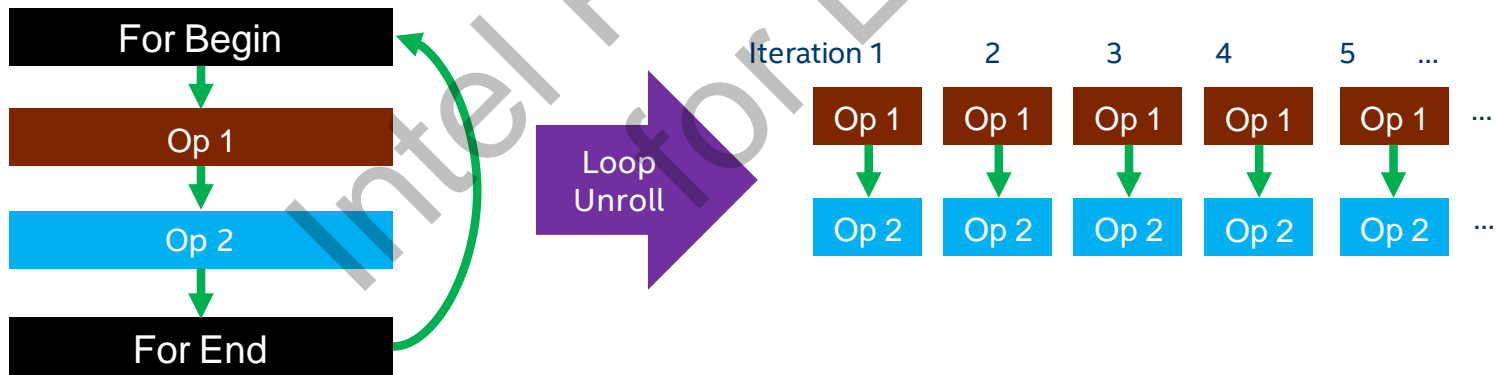
- Use Loop Analysis report to see if and how each loop is optimized
 - Helps identify component pipeline bottlenecks



Loop Unrolling

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

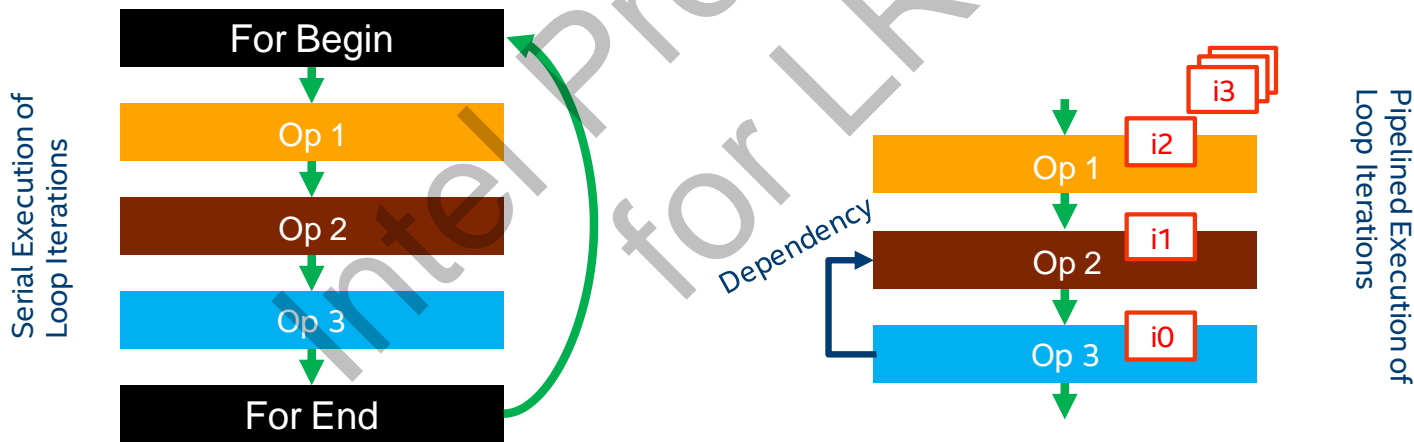
- Simple loops unrolled by the compiler automatically
- User may use `#pragma unroll` to control loop unrolling
- Loop must not have dependency from iteration to iteration



Loop Pipelining

Loop pipelining: Launch loop iterations as soon as dependency is resolved

- Initiation interval(II): launch frequency (in cycles) of a new loop iteration
 - II=1 is optimally pipelined
 - No dependency or dependencies can be resolved in 1 cycle



HTML Report: Loop Analysis

Loop analysis shows how loops are implemented

- Ability to correlate with source code

Loops analysis				
		<input checked="" type="checkbox"/> Show fully unrolled loops		
	Pipelined	II	Bottleneck	Details
whiletrue.entry (Implicit infinite loop)	Yes	1	n/a	Serial exe: Memory dependency
for.body (example.cpp:14)	Yes	1	n/a	
for.cond13.preheader (example.cpp:18)	Yes	2	II	Memory dependency
Fully unrolled loop (example.cpp:21)	n/a	n/a	n/a	Unrolled by #pragma unroll
for.body34 (example.cpp:25)	Yes	1	n/a	

Compiler-added loop, not in the code, implicit infinitely loop allowing the component to run continuously in pipelined fashion

Pipelined loop, II=1

Pipelined loop, II=2 due to memory dependency

Fully unrolled loop, due to user #pragma unroll

HTML Report: Area Analysis

View detailed estimated resource consumption by system or source line

- Analyze data control overhead
- View memory implementation
- Shows resource usage
 - ALUTs
 - FFs
 - RAMs
 - DSPs
- Identifies inefficient uses

Area report (source view)
Area utilization values are estimated
Notation file: X > file: Y indicates a function call on line X was inlined using code on line Y.

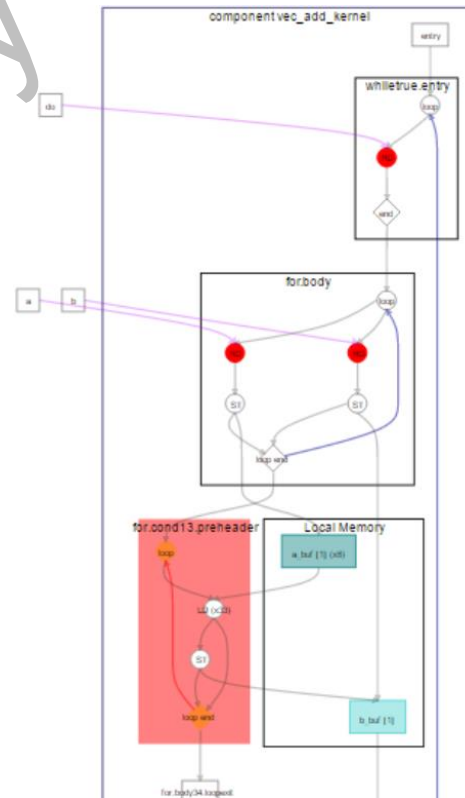
	ALUTs	FFs	RAMs	DSPs	Details
Var: isiget = 'j' (example.cpp:11)	85	133	0	0	• Implemente...
example.cpp:12 (a_buf)	33	1152	16	0	• Memory sys...
example.cpp:13 (b_buf)	0	0	64	0	• Memory sys...
▶ No Source Line	853	1168	0	0	
▶ example.cpp:14	37	51	1	0	
▼ example.cpp:15	94	111	0	0	
State	68	87	0	0	
Store	34	24	0	0	
▶ example.cpp:16	94	111	0	0	
▶ example.cpp:22	1038	794	0	0	
▶ example.cpp:26	14	28	0	0	
▶ example.cpp:25	34	111	0	0	

```
example.cpp  hls.h  hls_internal.h
1 #include "hls/hls.h"
2 #include "stdio.h"
3 #include "stdlib.h"
4
5 typedef altera::stream_in<int> my_operand;
6 typedef altera::stream_out<int> my_result;
7
8 component void vec_add_kernel(my_operand &a, my_operand &b, my_result
9 &c)
10 {
11     int i;
12     int j;
13     int a_buf[32][32];
14     int b_buf[32][32];
15     for (i = 0; i < 32; i++) {
16         a_buf[i / 32][i % 32] = a.read();
17     }
18     for (j = 0; j < 1024 / 32; j++) {
19
20         #pragma unroll
21         for (i = 0; i < 32; i++) {
22             b_buf[j % 32][i] += a_buf[i][j % 32];
23         }
24     }
25     for (i = 0; i < 32 * 32; i++) {
26         c.write(b_buf[i / 32][i % 32]);
27     }
28 }
29
30 int main() {
31     my_operand a, b;
32     my_result c, d;
33
34     unsigned long long start = altera_hls_get_sim_time();
35 }
```

HTML Report: Component Viewer

Displays abstracted netlist of the HW implementation

- View data flow pipeline
 - See loads and stores
 - Interfaces including stream reads and writes
 - Memory structure
 - Loop structure
 - Possible performance bottlenecks
 - Unpipelined loops are colored light red
 - Stallable points are red



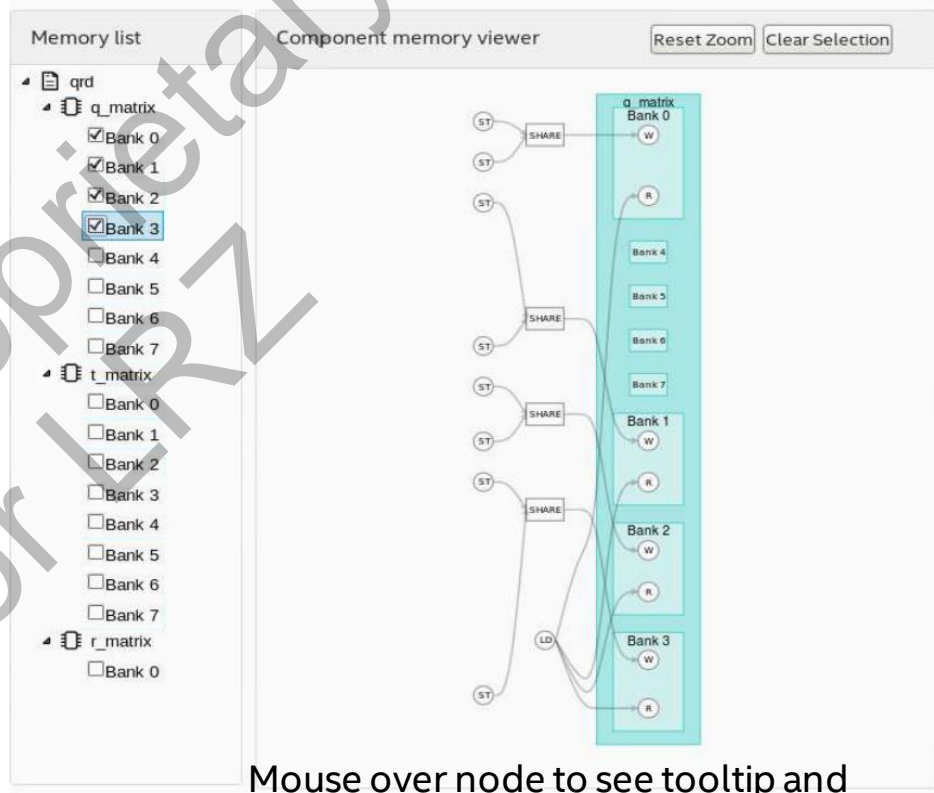
Mouse over node to see tooltip and details.

Correlates with source code.

HTML Report: Memory Viewer

Displays local memory implementation and accesses

- Visualize memory architecture
 - Banks, widths, replication, etc
- Visualize load-store units (LSUs)
 - Stall-free?
 - Arbitration
 - Red indicates stallable



Mouse over node to see tooltip and details.

Correlates with source code.

HTML Report: Verification Statistics

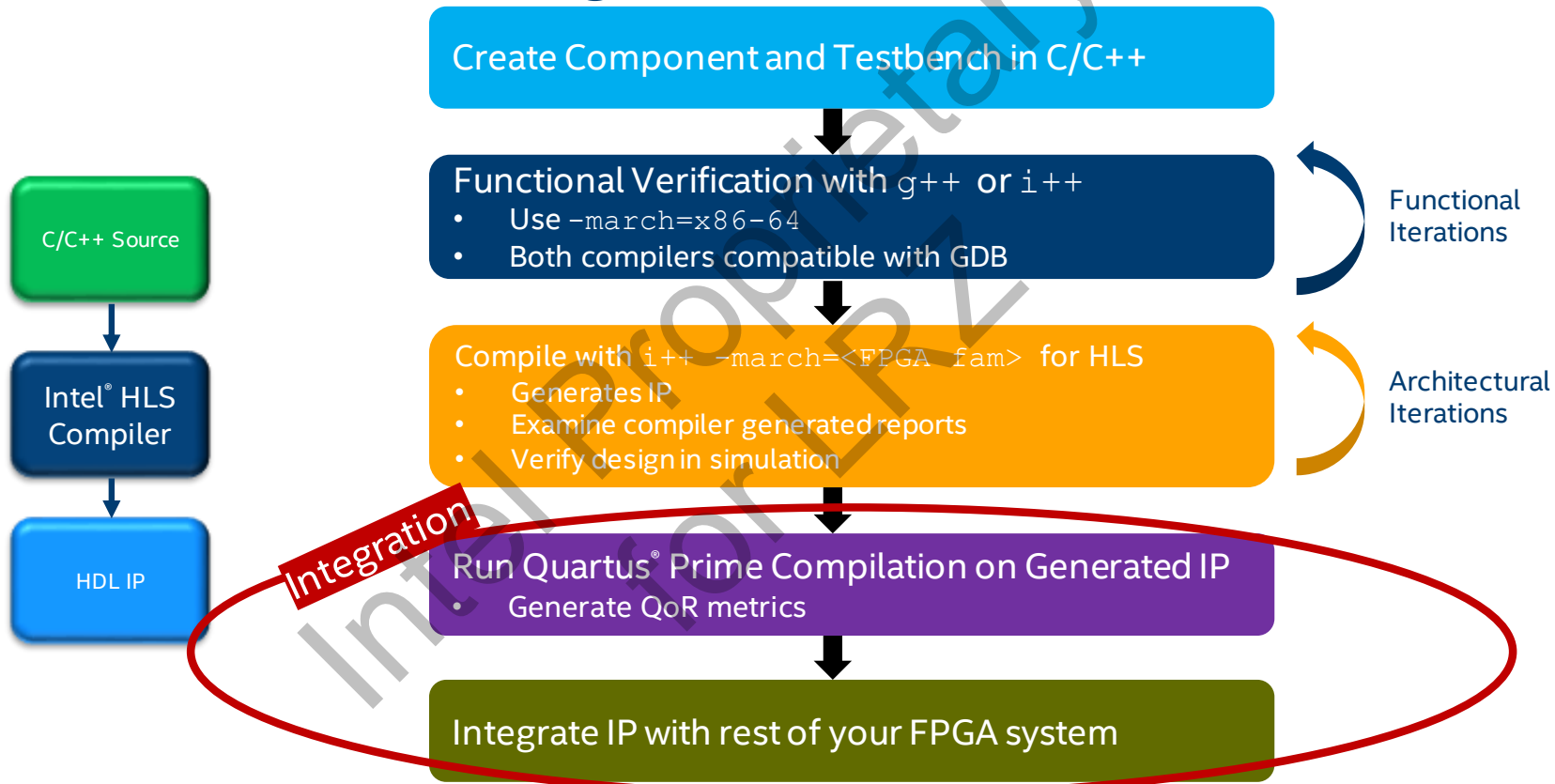
Reports execution statics from testbench execution, available after component is simulated (testbench executable ran)

- Number and type of component invocation
- Latency of component
- Dynamic Initiation interval of Component
- Data rates of streams

Measurements based on latest execution of testbench

Verification Statistics				
	Invocations	Latency (min,max,avg)	II (min,max,avg)	Details
dut (Unknown location)	101	4,4,4	1,1,1	Click for details
Explicit component invocations (Unknown location)	1	4,4,4	n/a,n/a,n/a	
Enqueued component invocations (Unknown location)	100	4,4,4	1,1,1	

HLS Procedure: Integration



Quartus® Generated QoR Metrics for IP

Use Intel® Quartus® Prime software to generate quality-of-result reports

- `i++` creates the Quartus project in a `.prj/quartus`
- To generate QoR data (final resource utilization, fmax)
 - Run `quartus_sh --flow compile quartus_compile`
 - Or use `i++ --quartus-compile opti`
- Report part of the HTML report
 - `a.prj/reports/report.html`
 - Summary page



The screenshot shows a web browser view of a Quartus QoR report. It contains two tables: 'Quartus Fit Clock Summary' and 'Quartus Fit Resource Utilization Summary'.

Quartus Fit Clock Summary				
1x clock fmax				
Frequency (MHz)	612.75			

Quartus Fit Resource Utilization Summary				
	ALMs	FFs	RAMs	DSPs
mycomp	18	3	0	0

Estimated Resource Usage				
Component Name	ALUTs	FFs	RAMs	DSPs
mycomp	38	2	0	0
Total	38 (0%)	2 (0%)	0 (0%)	0 (0%)

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 desired

2 use scenarios

1. Instantiate in HDL
2. Adding IP to a Platform Designer system

HDL Instantiation

Add Components to Intel® Quartus Project

- `<component>.qsys` to Standard Edition
- `<component>.ip` to Pro Edition

Instantiate component module in your design

- Use template

`a.prj/components/<component>/<component>_inst.v`

```
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  
);
```

Platform Designer System Integration Tool



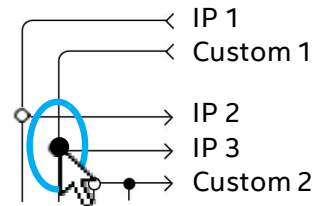
Catalog of available IP

- Interface protocols
- Memory
- DSP
- Embedded
- Bridges
- PLL
- Custom Components
- Custom Systems

Accelerate development

Use	Connections	Name	Description	Export	Clock
<input checked="" type="checkbox"/>		clk	Clock Source	clk	exported
<input checked="" type="checkbox"/>		clk_in	Clock Input	reset	
<input checked="" type="checkbox"/>		clk_in_reset	Reset Input	Double-click to export	clk
<input checked="" type="checkbox"/>		clk	Clock Output	Double-click to export	
<input checked="" type="checkbox"/>		clk_reset	Reset Output	Double-click to export	
<input checked="" type="checkbox"/>		pll	Altera PLL	Double-click to export	clk
<input checked="" type="checkbox"/>		refclk	Clock Input	Double-click to export	
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	sys_clk
<input checked="" type="checkbox"/>		outclk0	Clock Output	Double-click to export	ssram_clk
<input checked="" type="checkbox"/>		outclk1	Clock Output	Double-click to export	
<input checked="" type="checkbox"/>		logged	Conduit	Double-click to export	
<input checked="" type="checkbox"/>		reset_debounce	Reset Button Debounce	Double-click to export	sys_clk
<input checked="" type="checkbox"/>		clk	Clock	Double-click to export	[dock]
<input checked="" type="checkbox"/>		button_debounce_in	Reset Input	Double-click to export	[dock]
<input checked="" type="checkbox"/>		button_qual	Reset Output	Double-click to export	[dock]
<input checked="" type="checkbox"/>		button_qual_n	Reset Output	Double-click to export	[dock]
<input checked="" type="checkbox"/>		button_switch	Button Switch PIO	Double-click to export	sys_clk
<input checked="" type="checkbox"/>		dock	Clock Input	Double-click to export	[dock]
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	[dock]
<input checked="" type="checkbox"/>		buttonreg	Avalon Memory Mapped Slave	Double-click to export	[dock]
<input checked="" type="checkbox"/>		button_conduit	Conduit	Double-click to export	[dock]
<input checked="" type="checkbox"/>		switch_output	Conduit	Double-click to export	[dock]
<input checked="" type="checkbox"/>		av_sm_master	Avalon State Machine Master	Double-click to export	sys_clk
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	

Connect custom IP and systems



Simplify integration

Automate integration tasks

HDL

Platform Designer Integration

Platform Designer component generated for each component:

- For PD Standard – `a.prj/components/<component>/<component>.qsys`
- For Platform Designer – `a.prj/components/<component>/<component>.ip`

In Platform Designer, instantiate component from the IP Catalog in the HLS project directory

- Add IP directory to IP Catalog Search Locations
 - May use `a.prj/components/**/*`
- Can be stitched with other user IP or Intel® Quartus® IP with compatible interfaces

See tutorials under `tutorials/usability`

Platform Designer HLS Component Example

Example

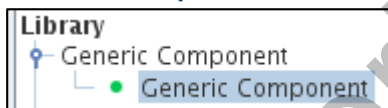
- Cascaded low-pass filter and high-pass filter

HLS Components

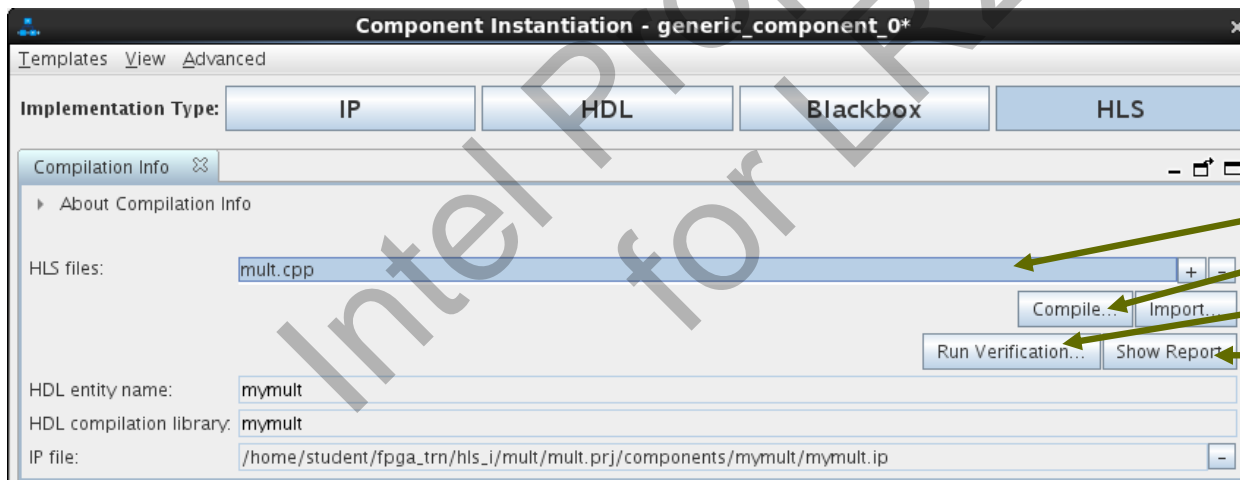
Use	Connections	Name	Description	Export	Clock
<input checked="" type="checkbox"/>		clock_in	Clock Bridge		
		in_clk	Clock Input	clk	exported
		out_clk	Clock Output	Double-click to export	clock_in_o...
<input checked="" type="checkbox"/>		reset_in	Reset Bridge		
		clk	Clock Input	Double-click to export	clock_in_...
		in_reset	Reset Input	Double-click to export	[clk]
		out_reset	Reset Output	Double-click to export	[clk]
<input checked="" type="checkbox"/>		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]
<input checked="" type="checkbox"/>		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]

HLS-Backed Components

- Generic component can be used in place of actual IP core



- Choose Implementation Type: HLS



- Specify HLS source files
- Compile Component
- Run Cosim
- Display HTML report

HLS FLOW

Lab 2

Intel Proprietary
for LRZ

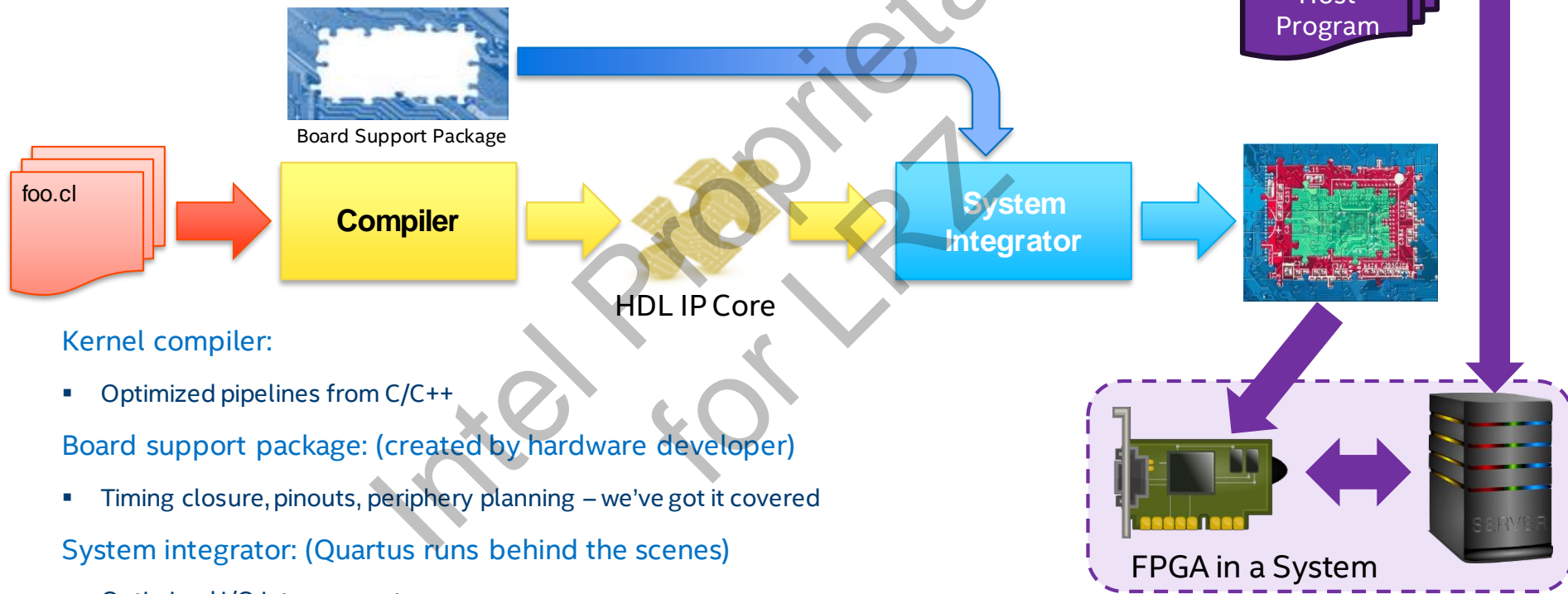
FPGA PROGRAMMING MODEL:

OpenCL

Intel Proprietary
for ERZ

Intel FPGA SDK for OpenCL™ Flow

A system level view:



Kernel compiler:

- Optimized pipelines from C/C++

Board support package: (created by hardware developer)

- Timing closure, pinouts, peripheral planning – we've got it covered

System integrator: (Quartus runs behind the scenes)

- Optimized I/O interconnects

OpenCL

Hardware Agnostic Compute Language

Invented by Apple

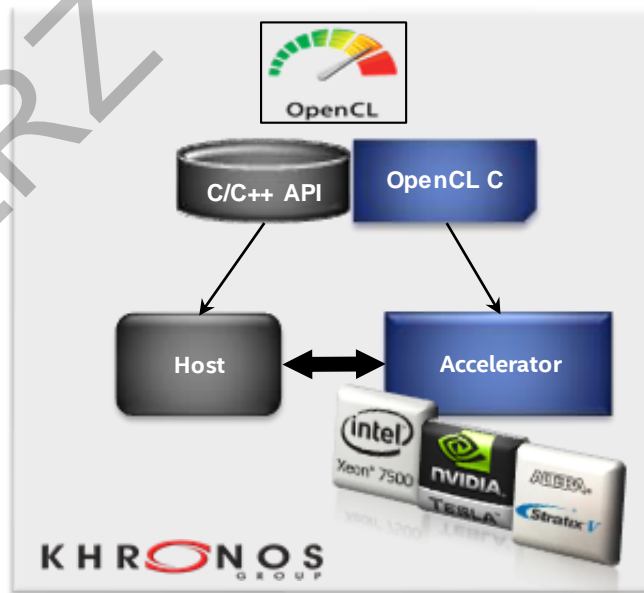
- 2008 Specification donated to Khronos Group
- Now managed by Intel

OpenCL C and C++

What does OpenCL™ give us?

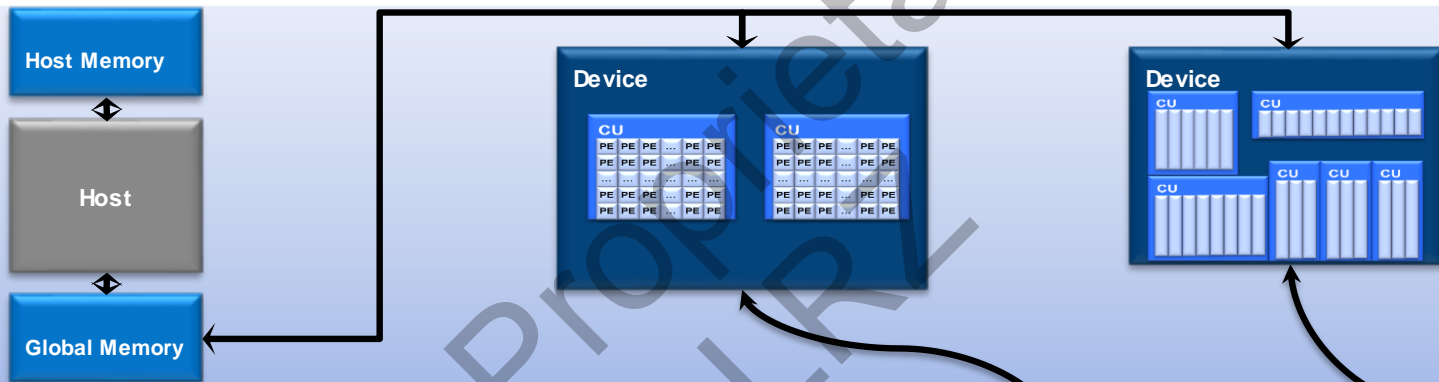
- Industry standard programming model
- Functional portability across platforms
- Well thought out specification

KHRONOS
GROUP



Heterogeneous Platform Model

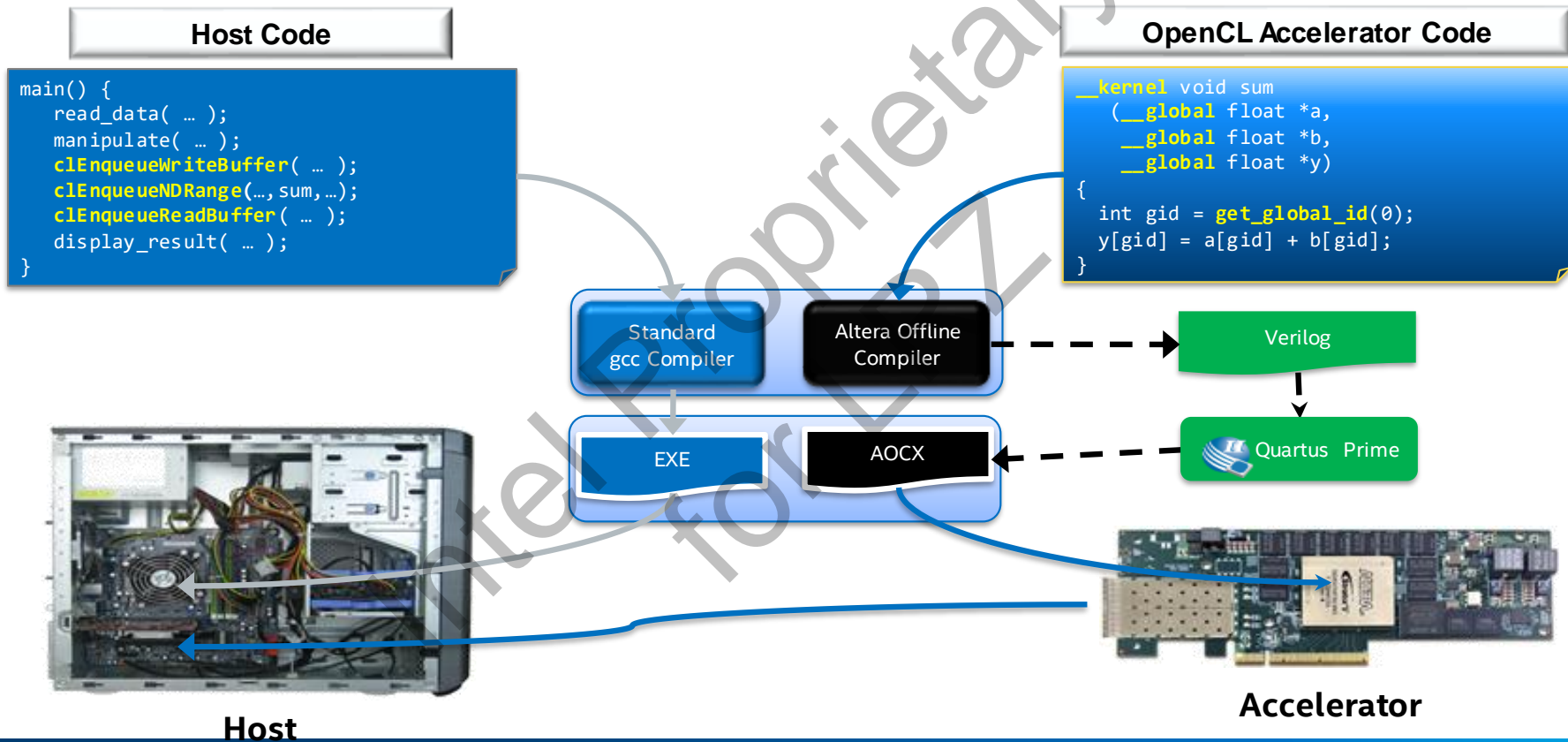
OpenCL
Platform
Model



Example
Platform



OpenCL Use Model: Abstracting the FPGA away



OpenCL Host Program

Pure software written in standard C/C++ languages

Communicates with the accelerator devices via an API which abstracts the communication between the host processor and the kernels

Copy data from Host to
FPGA

Tell the FPGA to run a
particular kernel

Copy data from FPGA to
Host

```
main()
{
    read_data_from_file( ... );
    manipulate_data( ... );
    clEnqueueWriteBuffer( ... );
    clEnqueueNDRange(..., sum, ...);
    clEnqueueReadBuffer( ... );
    display_result ( ... );
}
```

OpenCL Kernels

Kernel: Data-parallel function

- Defines many parallel threads
- Each thread has an identifier specified by "get_global_id"
- Contains keyword extensions to specify parallelism and memory hierarchy

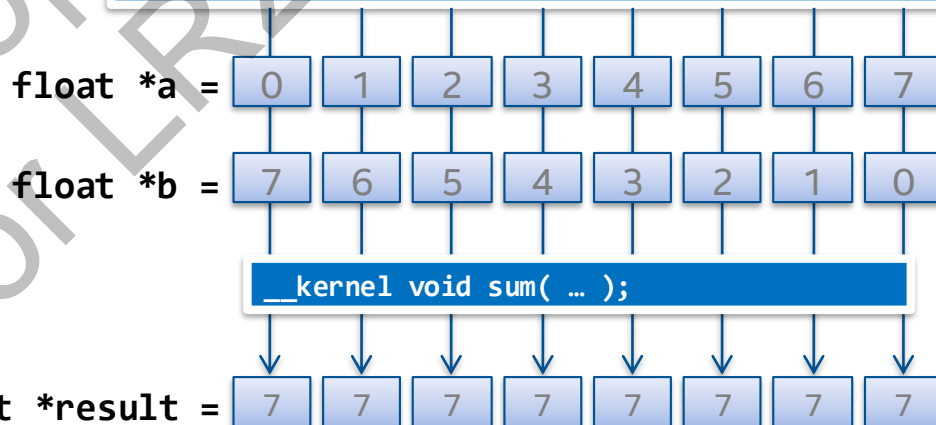
Executed by an OpenCL device

- CPU, GPU, FPGA

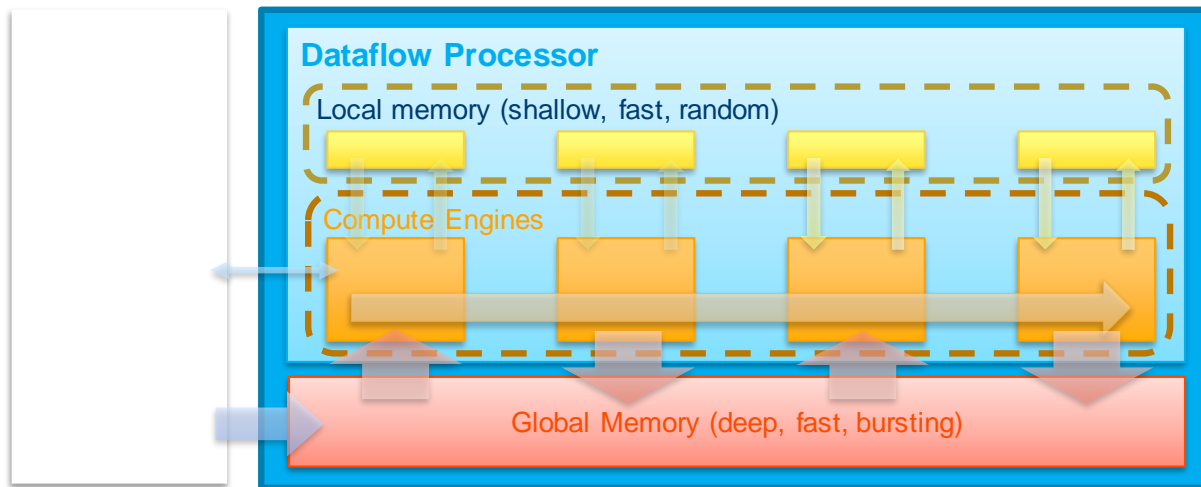
Code portable NOT performance portable

- Between FPGAs it is!

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



Software Engineer's View of an OpenCL System



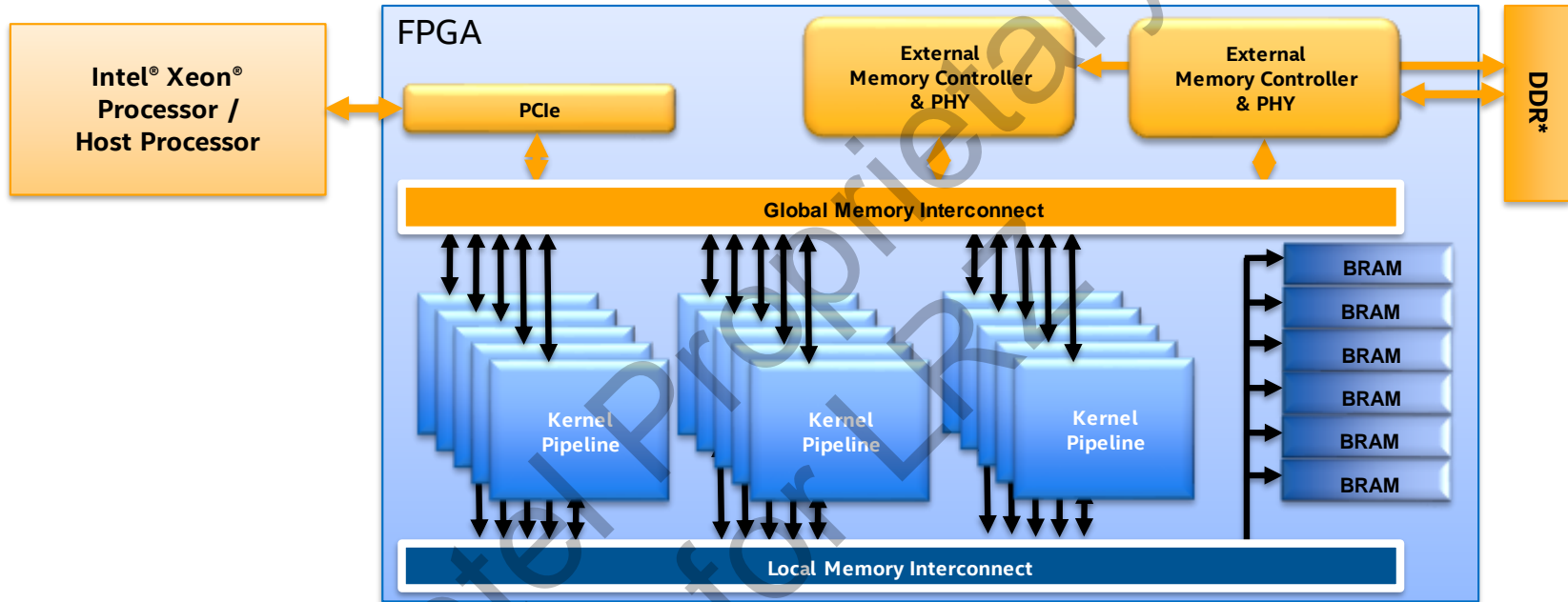
Device contains compute engines that run the kernel

Host talks to global memory through OpenCL routines

Global memory is large, fast, and likes to burst

Local memory is small, fast, and supports random access

FPGA OpenCL Architecture



Modest external memory bandwidth

Extremely high internal memory bandwidth

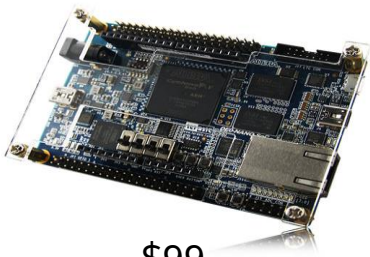
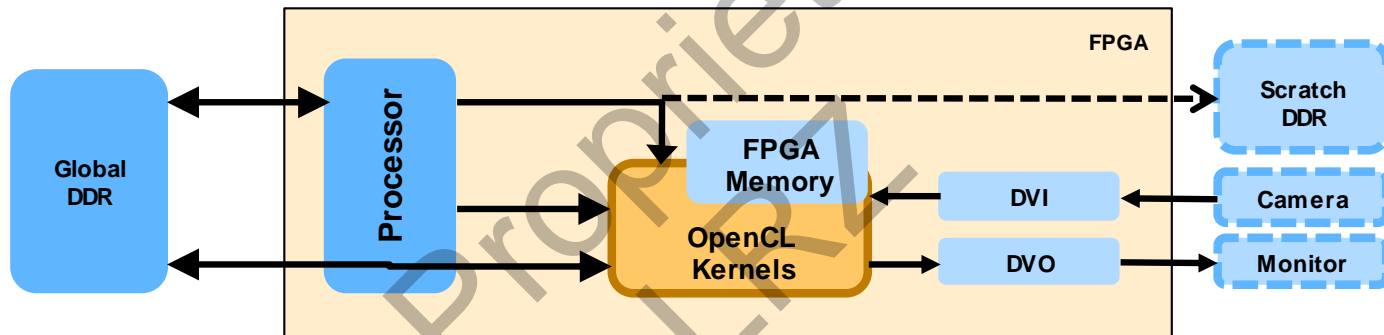
Highly customizable compute cores

Start with a Reference Platform (1/2)

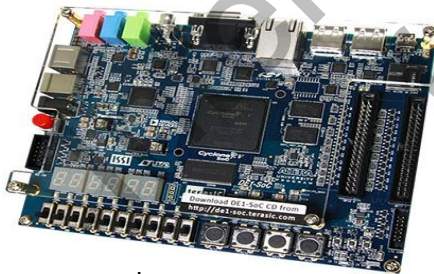
	Network Enabled	High Performance Computing (HPC)
Requirement	Low Latency	Compute Power/ Memory Bandwidth
Architecture	<p>Stratix V FPGA</p> <p>OpenCL API HAL UMD KMD</p> <p>CPLD FLASH</p> <p>CPLD Bridge</p> <p>DMA</p> <p>DDR3</p> <p>DDR3</p> <p>10G UDP</p> <p>10G UDP</p> <p>(OpenCL Kernels)</p> <p>PCle</p>	<p>Stratix V FPGA</p> <p>OpenCL API HAL UMD KMD</p> <p>DMA</p> <p>DDR3</p> <p>DDR3</p> <p>(OpenCL Kernels)</p> <p>PCle</p>
Global Memory	DDR and QDR II+	Large amount of DDR
IO Channels	2x10GbE (MAC/UOE)	None (Minimize IP overhead)

Start with a Reference Platform (2/2)

Host and accelerator in same package: SoC



\$99



\$175

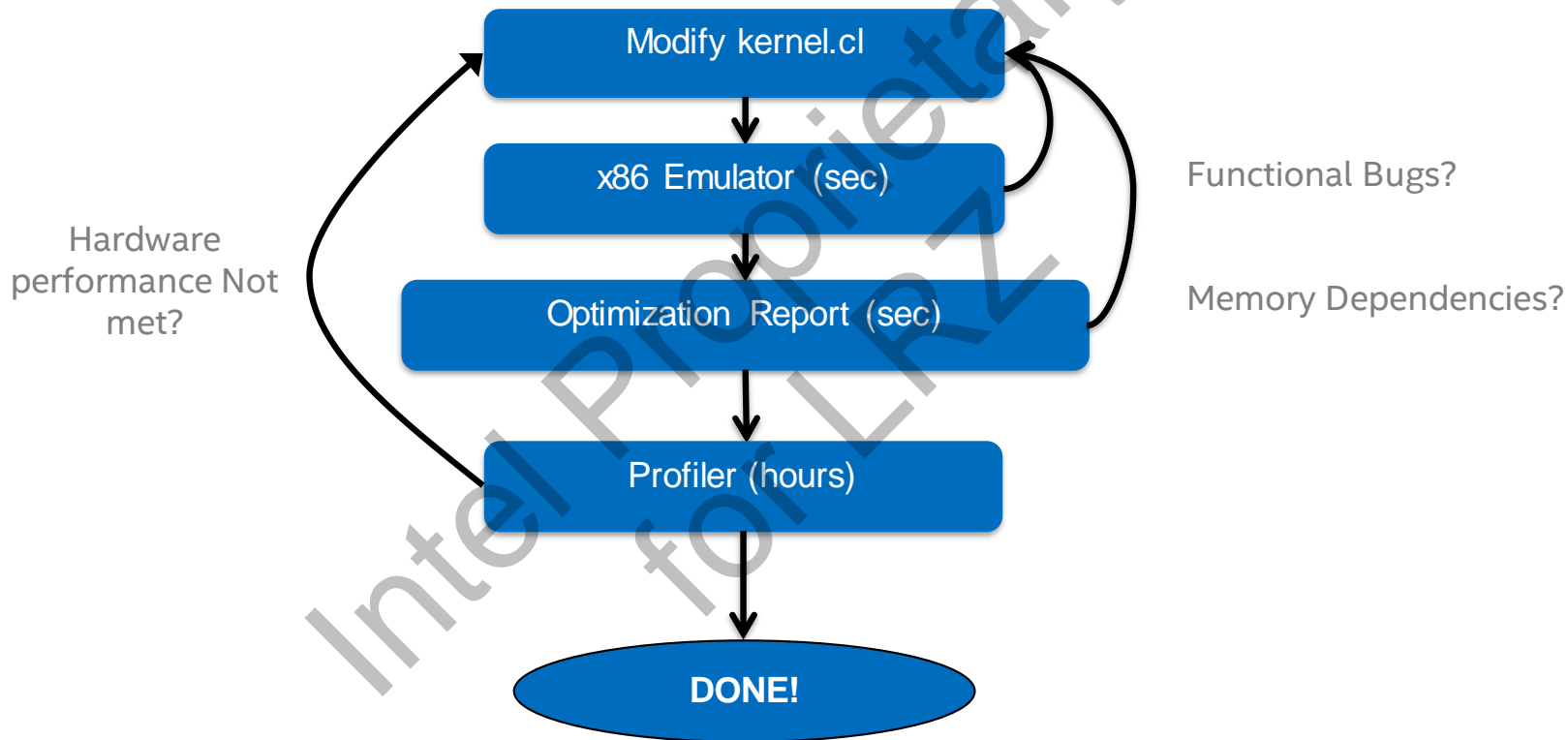


\$250



>\$1000

Development Flow using SDK



Compiling Kernel

Run the Altera Offline Compiler in command prompt

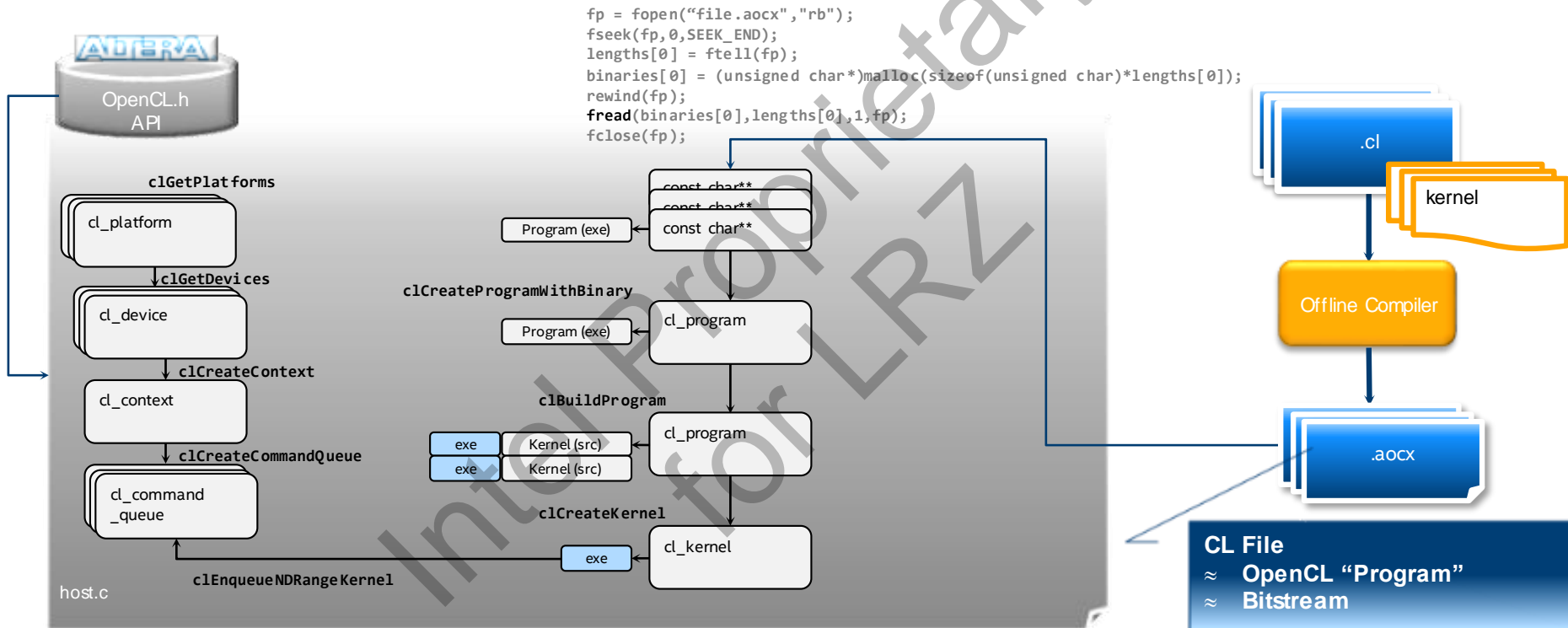
- `aoc --board <board> <Kernel.cl>`
- Run `aoc --list-boards` to see all available boards

AOC performs system integration to generate the kernel hardware system and the Quartus Prime software to compile the design

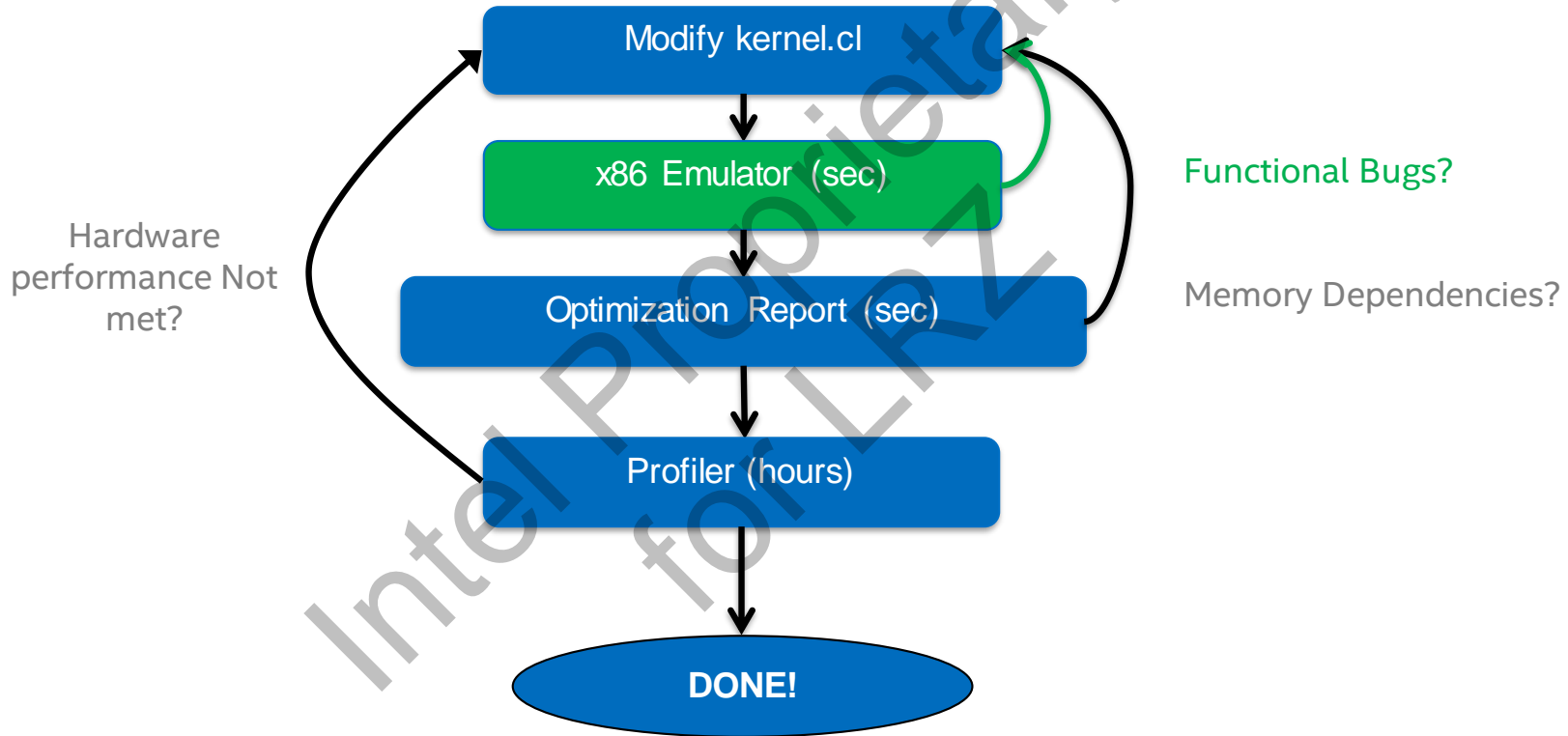
```
/mydesigns/matrixMult$ aoc matrixMul.cl  
aoc: Selected target board bittware_s5pciehq
```

```
+-----+  
; Estimated Resource Usage Summary ;  
+-----+  
; Resource + Usage ;  
+-----+  
; Logic utilization ; 52% ;  
; Dedicated logic registers ; 23% ;  
; Memory blocks ; 31% ;  
; DSP blocks ; 54% ;  
+-----+
```


Executing the kernel: clCreateProgramWithBinary



Development Flow using SDK

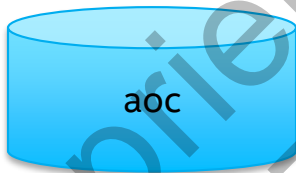


Emulator – The Flow

Generate emulation aocx

```
kernel void convolution(  
    global int * filter_coef,  
    global int * input_image,  
    global int * output_image  
) {  
    int grid = get_group_id(0);  
    ...  
}
```

conv.cl



aoc **-march=emulator** conv.cl



conv.aocx

Run host program with emulator aocx

- Host compile does not change
- set `CL_CONTEXT_EMULATOR_DEVICE_ALTERA=<number_of_boards>`

```
C:\>  
c:\openc1>aoc -march=emulator conv.cl  
c:\openc1>dir  
host.exe  conv.cl  conv.aocx  
c:\openc1>host.exe  
running..  
done!
```

Printf

Can use printf within kernel on FPGA

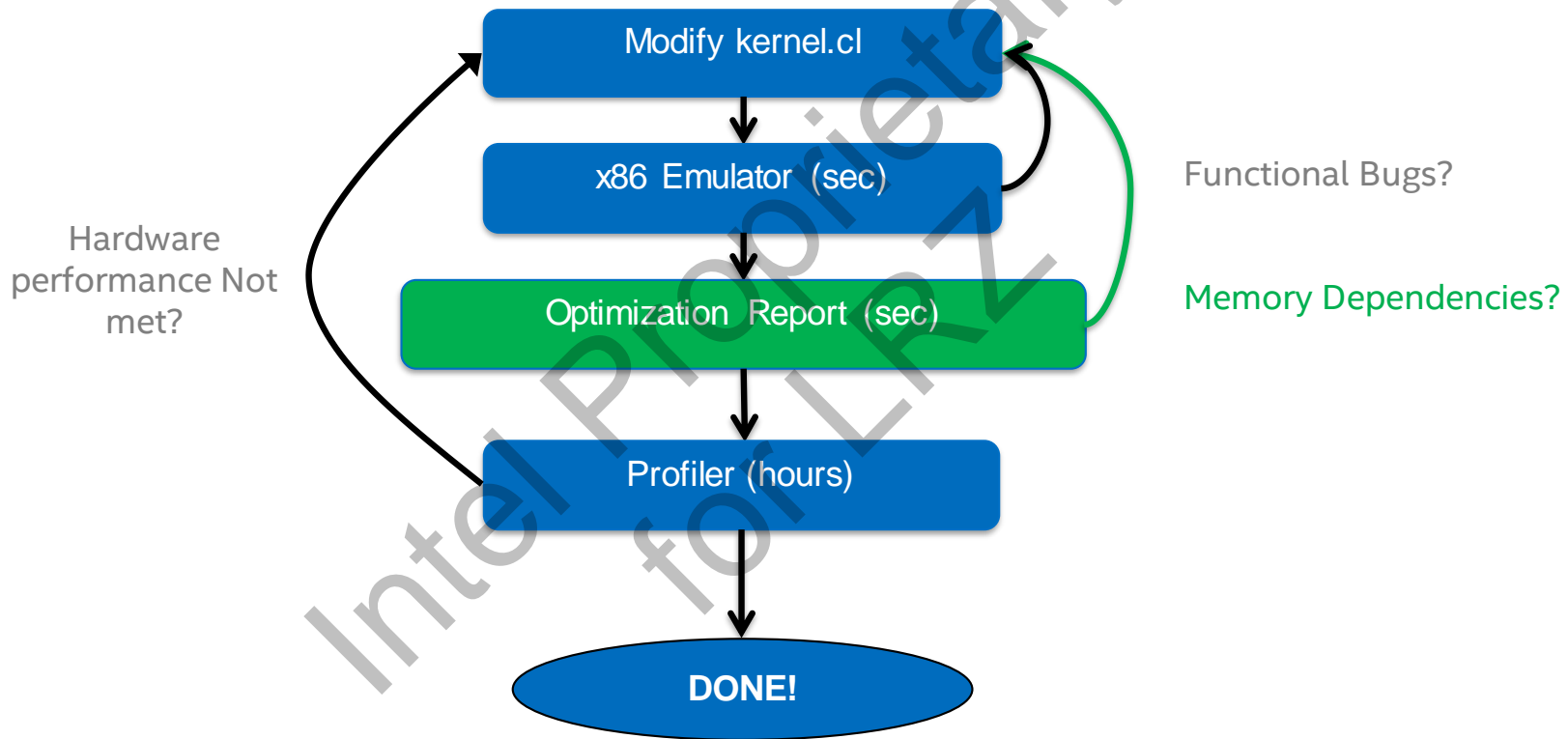
- Adds some memory traffic overhead

In the emulator, printf runs on IA

- Useful for fast debug iterations

Intel Proprietary
for LRZ

Development Flow using SDK



Optimization Report

Intel FPGA SDK for OpenCL provides a static report to identify performance bottlenecks when writing single-threaded kernels

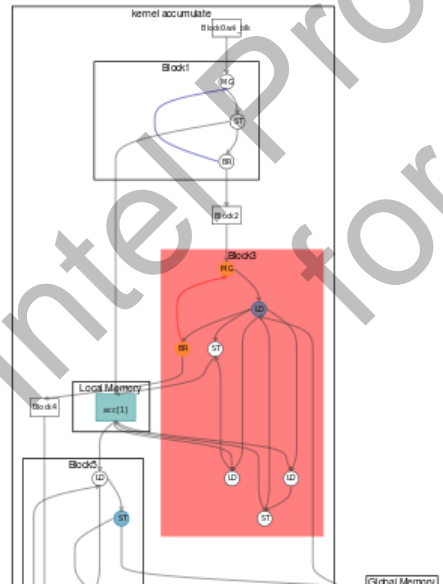
Use `-c` to stop after generating the reports

- `aoc -c <kernel.cl>`
- Report is in: `<kernel>/reports/report.html`

Intel Proprietary
for LRZ

Pipelined	II	Bottleneck	Details
Yes	1	n/a	
Yes	11	II	Memory dependency and...
Yes	1	n/a	

```
ict data_in, global float* restrict data
```



Details

Block3:
 II bottleneck due to memory dependency between:

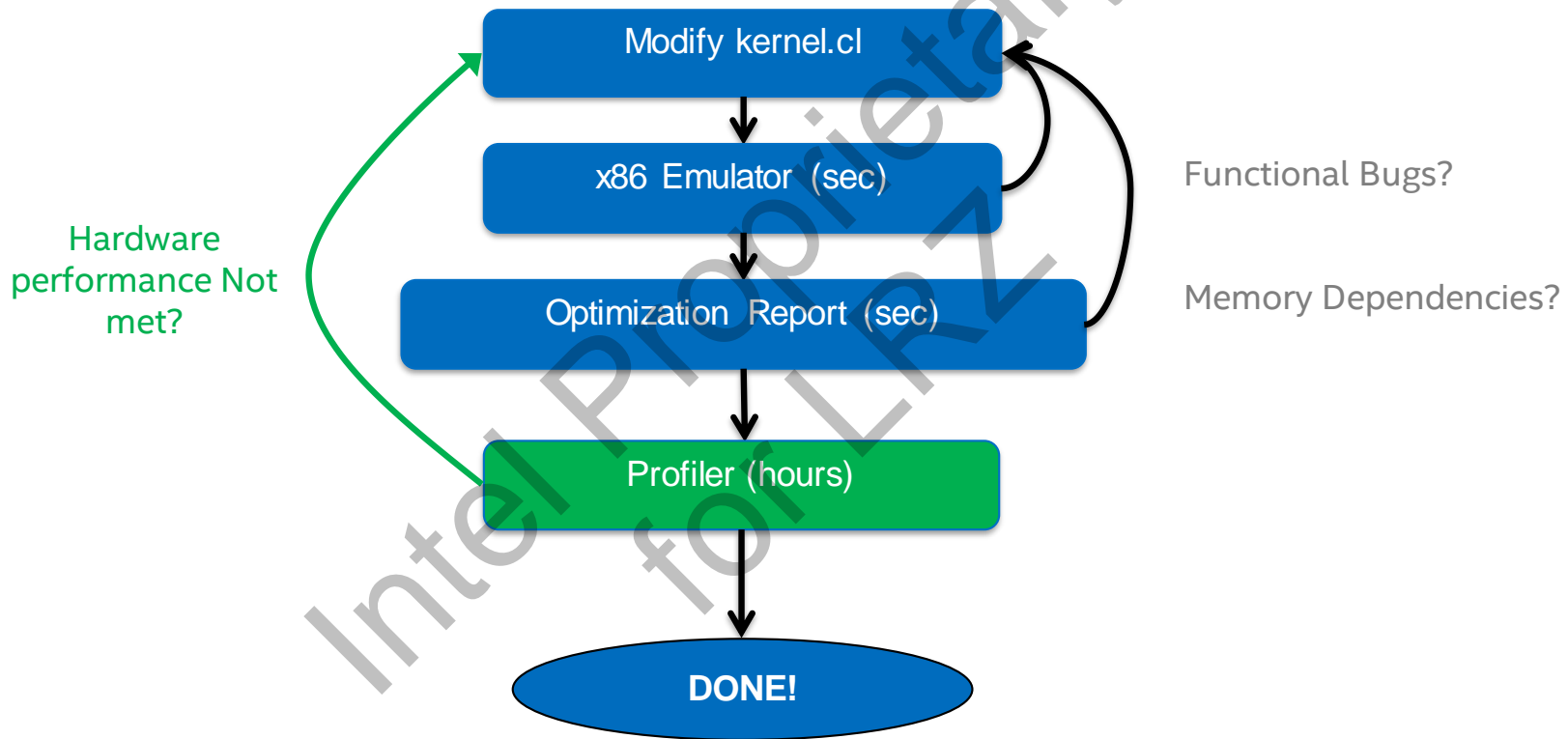
- Load Operation ([acc.cl:12](#))
- Store Operation ([acc.cl:13](#))

Largest critical path contributor(s):

- 31%: Load Operation ([acc.cl:12](#))
- 27%: Load Operation ([acc.cl:13](#))
- 18%: Hardened Floating-Point Multiply-Add Operation ([acc.cl:12](#))
- 14%: Store Operation ([acc.cl:12](#))



Development Flow using SDK

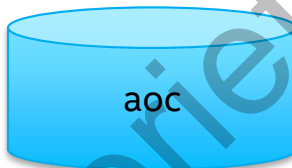


Profiler – the flow

1. Generate program bitstream with profiling enabled

```
kernel void convolution(  
    global int * filter_coef,  
    global int * input_image,  
    global int * output_image  
) {  
    int grid = get_group_id(0);  
}
```

conv.cl



aoc --profile conv.cl



conv.aocx

2. Run host program with instrumented aocx

```
c:\>  
c:\openc1>dir  
host.exe  conv.aocx  
c:\openc1>host.exe  
running...  
done!  
c:\openc1>dir  
host.exe  conv.aocx  profile.mon
```

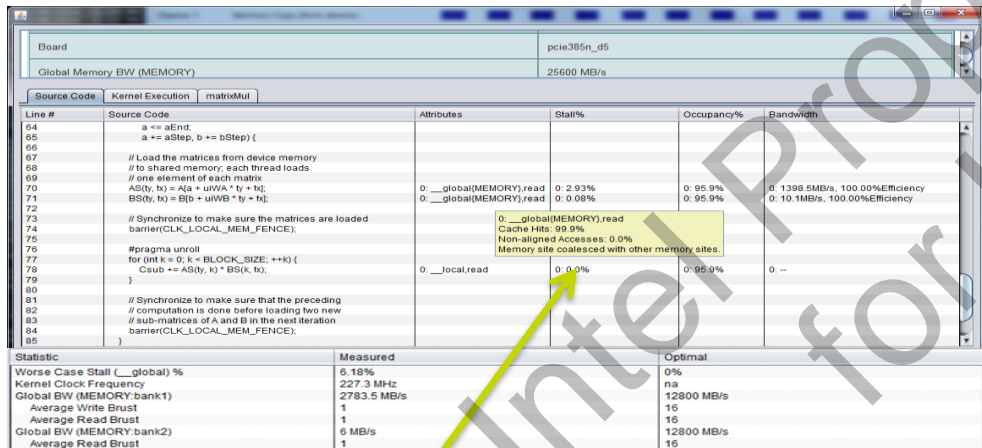
3. Run the profiler GUI:

aocl report <aocx> <profile.mon>

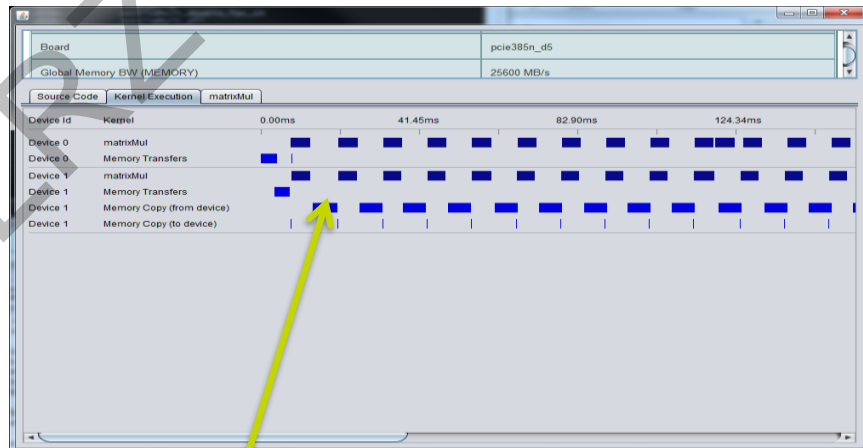
Dynamic Profiler

Intel FPGA SDK for OpenCL enables users to get runtime information about their kernel performance

Bottlenecks, bandwidth, saturation, pipeline occupancy



Performance Stats



Execution Times



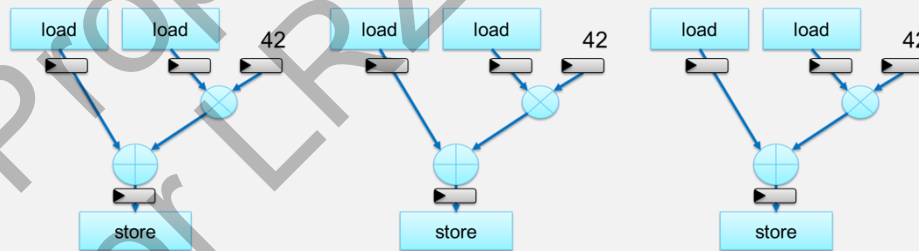
HIGH PERFORMANCE DATA FLOW

Intel Proprietary
f011RZ

Execution of Threads on FPGA – Naïve Approach

Thread execution can be executed on replicated pipelines in the FPGA

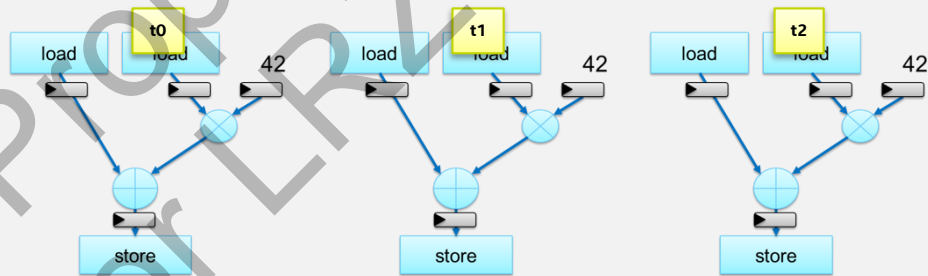
```
kernel void  
add( global int* Mem ) {  
    ...  
    Mem[100] += 42*Mem[101];  
}
```



Execution of Threads on FPGA – Naïve Approach

Thread execution can be executed on *replicated* pipelines in the FPGA

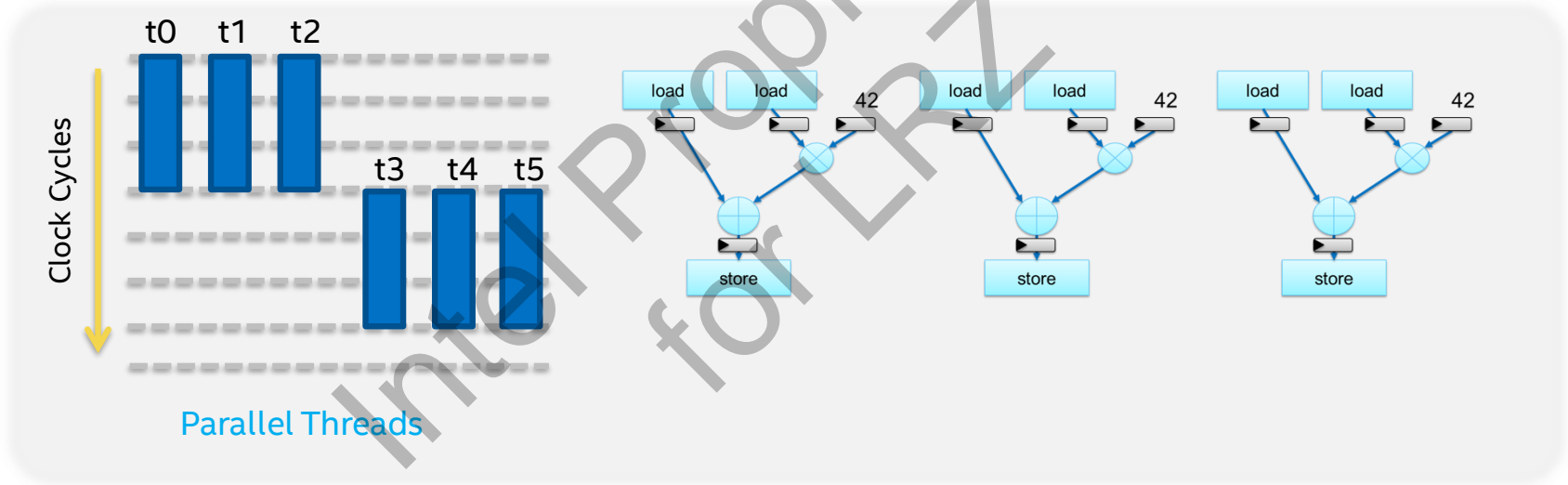
```
kernel void  
add( global int* Mem ) {  
    ...  
    Mem[100] += 42*Mem[101];  
}
```



Execution of Threads on FPGA – Naïve Approach

Thread execution can be executed on *replicated* pipelines in the FPGA

- **Throughput = 1 thread per cycle**
- Area inefficient

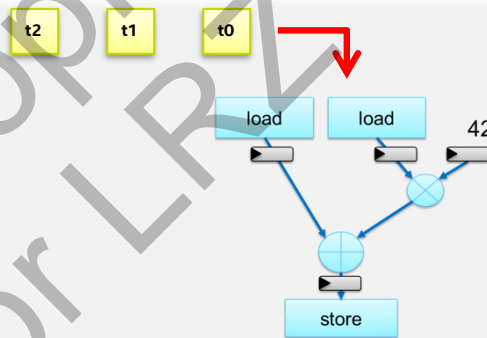


Execution of Threads on FPGA

Better method involves taking advantage of *pipeline parallelism*

- Attempt to create a deeply pipelined implementation of kernel
- On each clock cycle, we attempt to send in new thread

```
kernel void  
add( global int* Mem ) {  
    ...  
    Mem[100] += 42*Mem[101];  
}
```

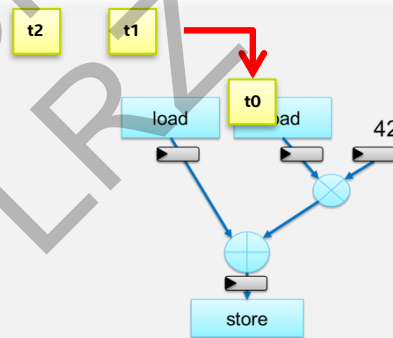


Execution of Threads on FPGA

Better method involves taking advantage of *pipeline parallelism*

- Attempt to create a deeply pipelined implementation of kernel
- On each clock cycle, we attempt to send in new thread

```
kernel void  
add( global int* Mem ) {  
    ...  
    Mem[100] += 42*Mem[101];  
}
```

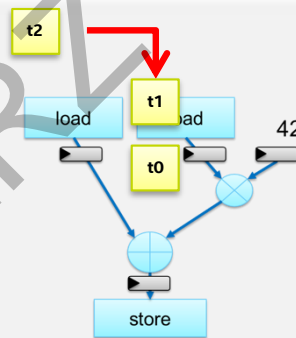


Execution of Threads on FPGA

Better method involves taking advantage of *pipeline parallelism*

- Attempt to create a deeply pipelined implementation of kernel
- On each clock cycle, we attempt to send in new thread

```
kernel void
add( global int* Mem ) {
    ...
    Mem[100] += 42*Mem[101];
}
```

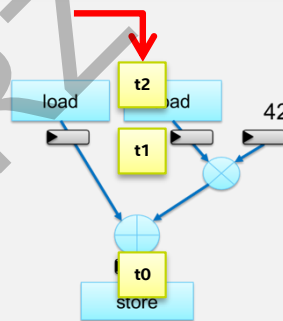


Execution of Threads on FPGA

Better method involves taking advantage of *pipeline parallelism*

- Attempt to create a deeply pipelined implementation of kernel
- On each clock cycle, we attempt to send in new thread

```
kernel void
add( global int* Mem ) {
    ...
    Mem[100] += 42*Mem[101];
}
```

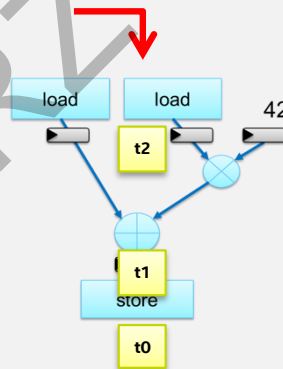


Execution of Threads on FPGA

Better method involves taking advantage of *pipeline parallelism*

- Attempt to create a deeply pipelined implementation of kernel
- On each clock cycle, we attempt to send in new thread

```
kernel void
add( global int* Mem ) {
    ...
    Mem[100] += 42*Mem[101];
}
```

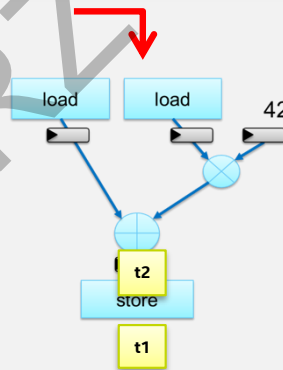


Execution of Threads on FPGA

Better method involves taking advantage of *pipeline parallelism*

- Attempt to create a deeply pipelined implementation of kernel
- On each clock cycle, we attempt to send in new thread

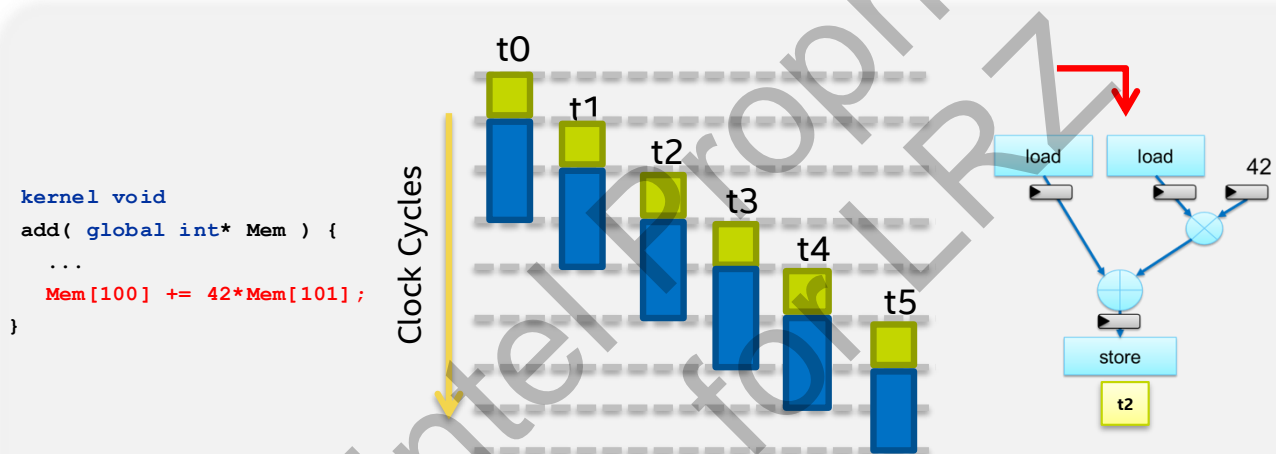
```
kernel void
add( global int* Mem ) {
    ...
    Mem[100] += 42*Mem[101];
}
```



Execution of Threads on FPGA

Better method involves taking advantage of *pipeline parallelism*

- *Throughput = 1 thread per cycle*





SINGLE THREADED OPTIMIZATIONS

Intel Proprietary
for RZ

OpenCL on Intel FPGAs

Main assumptions made in previous OpenCL programming model

- Data level parallelism exists in the kernel program

Not all applications well suited for this assumption

- Some applications do not map well to data-parallel paradigms

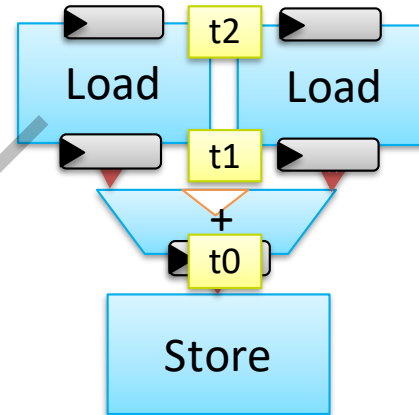
These are the only workloads that GPUs support

Intel Proprietary for LRV

Data-Parallel Execution

On the FPGA, we use the idea of pipeline parallelism to achieve acceleration

```
kernel void
sum(global const float *a,
    global const float *b,
    global float *c)
{
    int xid = get_global_id(0);
    c[xid] = a[xid] + b[xid];
}
```

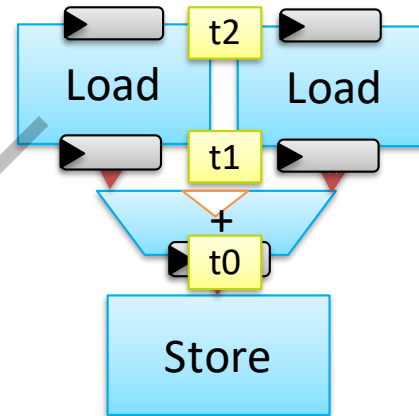


Threads can execute in an embarrassingly parallel manner

Data-Parallel Execution - Drawbacks

Difficult to express programs which have partial dependencies during execution

```
kernel void
sum(global const float *a,
    global const float *b,
    global float *c)
{
    int xid = get_global_id(0);
    c[xid] = c[xid-1] + b[xid];
}
```



Would require complicated hardware and new language semantics to describe the desired behavior

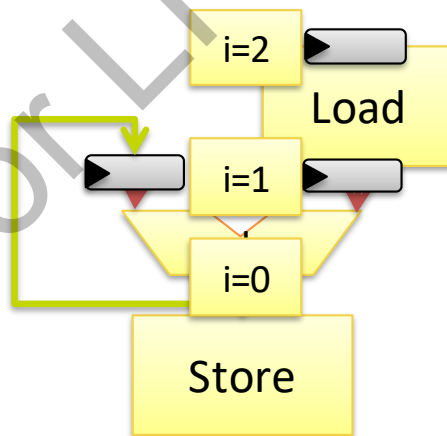
Solution: Tasks and Loop-Pipelining

Allow users to express programs as a single-thread

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

Pipeline parallelism still leveraged to efficiently execute loops in Intel's FPGA OpenCL

- **Parallel execution inferred by compiler**
- **Loop Pipelining**



Loop Carried Dependencies

Loop-carried dependencies are dependencies where one iteration of the loop depends upon the results of another iteration of the loop

```
kernel void state_machine(ulong n)
{
    t_state_vector state = initial_state();
    for (ulong i=0; i<n; i++) {
        state = next_state( state );
        unit y = process( state );
        write_channel_altera(OUTPUT, y);
    }
}
```

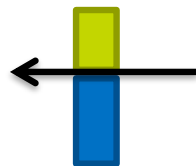
The variable `state` in iteration 1 depends on the value from iteration 0. Similarly, iteration 2 depends on the value from iteration 1, etc.

Loop Carried Dependencies

To achieve acceleration, we can *pipeline* each iteration of a loop containing loop carried dependencies

- Analyze any dependencies between iterations
- Schedule these operations
- Launch the next iteration as soon as possible

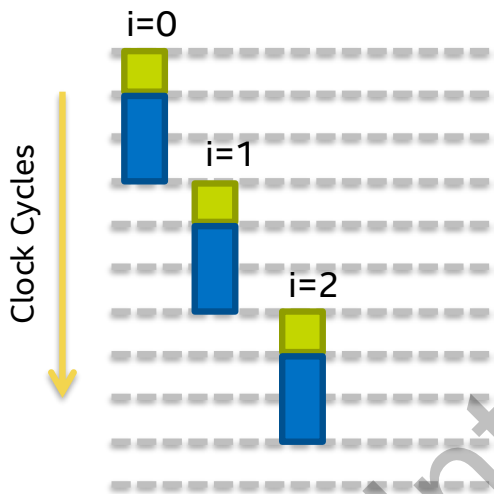
```
kernel void state_machine(ulong n)
{
    t_state_vector state = initial_state();
    for (ulong i=0; i<n; i++) {
        state = next_state( state );
        unit y = process( state );
        write_channel_altera(OUTPUT, y);
    }
}
```



At this point, we can launch the next iteration

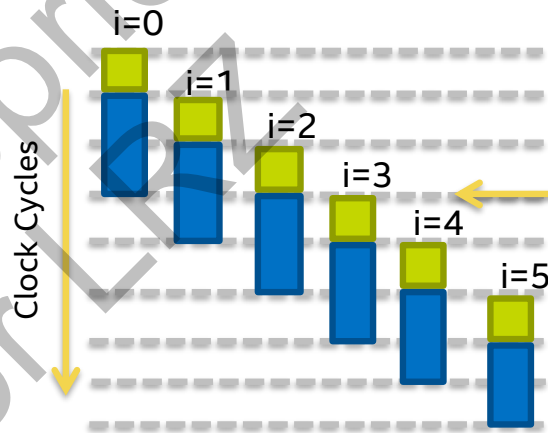
Loop Pipelining Example

No Loop Pipelining



No Overlap of Iterations!

With Loop Pipelining

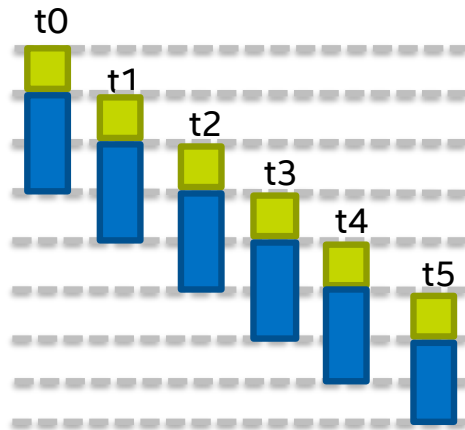


Looks almost like multi-threaded execution!

Finishes Faster because Iterations Are Overlapped

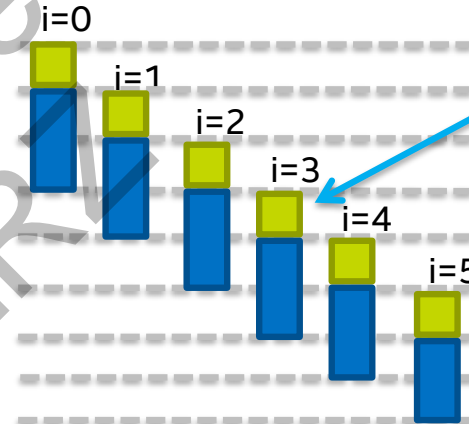
Parallel Threads vs. Loop Pipelining

So what's the difference?



Parallel Threads

Parallel threads launch 1 thread per clock cycle in pipelined fashion



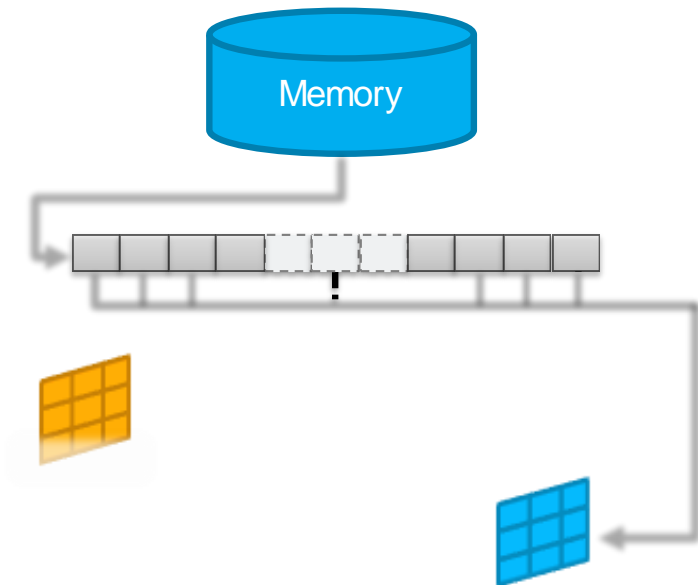
Loop Pipelining

Loop dependencies may not be resolved in 1 clock cycle

Loop Pipelining enables Pipeline Parallelism *AND* the communication of state information between iterations.

Image Filter

```
// Note that no thread identifiers anywhere in the kernel - single threaded code!  
kernel void sobel ( global char * restrict data_in, global char * restrict data_out,  
    unsigned iterations, int threshold ) {  
  
    const int filterH[3][3] = { {-1,0,1}, {-2,0,2}, {-1,0,1} };  
    const int filterV[3][3] = { {-1,-2,-1}, {0,0,0}, {1,2,1} };  
  
    char rows[2 * WIDTH + 3]; // Pixel buffer of 2 rows and 3 extra pixels  
  
    int count = 0;  
    while (count != iterations) {  
        // Each cycle, shift a new pixel into the buffer.  
        // Unrolling this loop allows the compiler to infer a shift register.  
        #pragma unroll  
        for (int i = WIDTH * 2 + 2; i > 0; --i) {  
            rows[i] = rows[i - 1];  
        }  
        rows[0] = data_in[count]; // Shift image data (from DDR) into one end  
  
        int accumH=0, accumV=0;  
        for (unsigned y=0; y<TILE_SIZE; y++) {  
            for (unsigned x=0; x<TILE_SIZE; x++) {  
                unsigned int pixel = rows[y * WIDTH + x];  
  
                accumH += pixel * filterH[y][x];  
                accumV += pixel * filterV[y][x];  
            }  
        }  
        int accum = accumH*accumH + accumV*accumV;  
        char out_val = (accum > (threshold * threshold)) ? 255 : 0;  
        data_out[count++] = out_val; //output pixel (to DDR)  
    }  
}
```





CHANNELS

Harnessing Dataflow to Reduce Memory Bandwidth

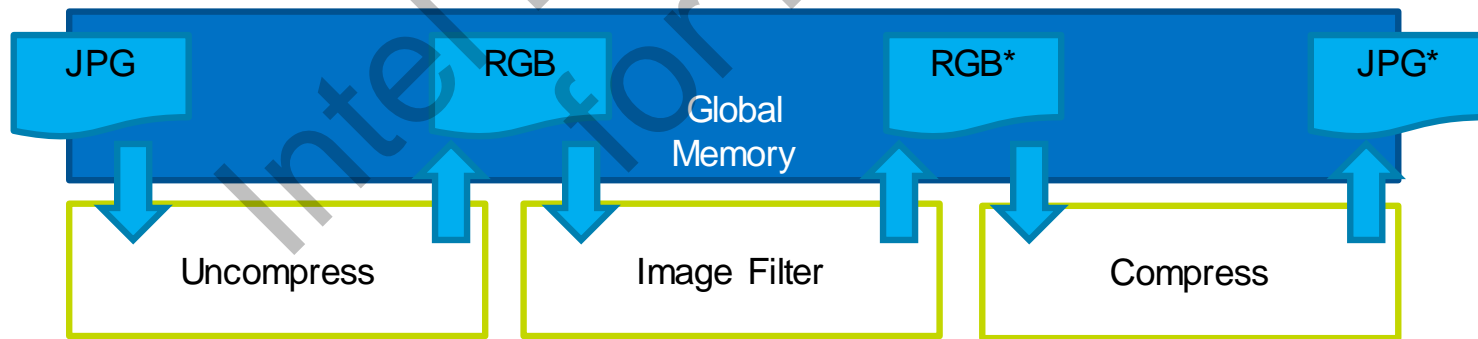
Intel Proprietary
for LRZ

Data Movement in GPUs

Data is moved from host over PCIe/express

Instructions and data is constantly sent back and forth between host cache and memory and GPU memory

- Requires buffering larger data sets before passing to GPU to be processed
- Significant latency penalty
- Requires high memory and host bandwidth
- Requires sequential execution of kernels



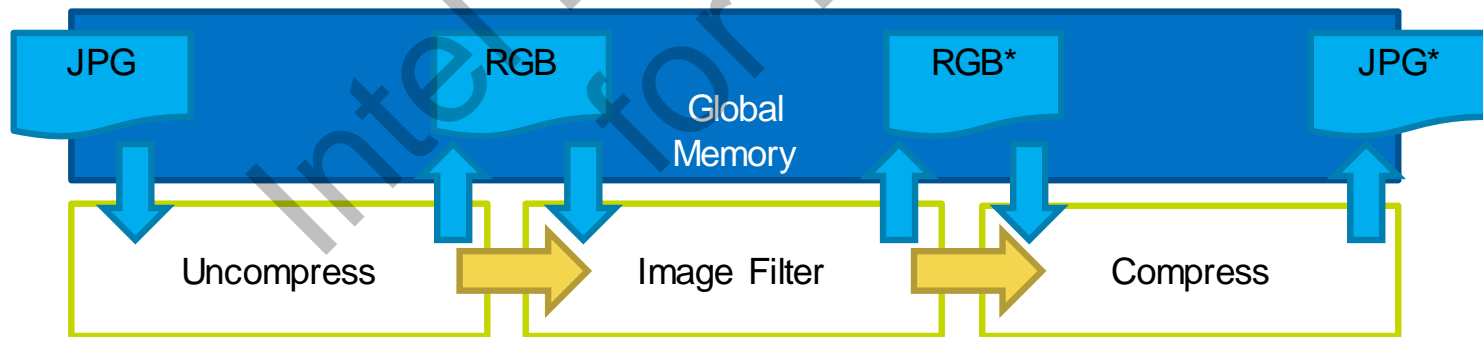
Altera_Channels Extension

An FPGA has programmable routing

Can't we just send data across wires between kernels?

Advantages:

- Reduce memory bandwidth
- Lower latency through fine-grained synchronization between kernels
- Reduce complexity (wires are trivial compared to memory access)
 - o Lower cost, lower area, higher performances
- Enable modular dataflow design through small kernels exchanging data
- Different workgroup sizes and degrees of parallelism in connected modules



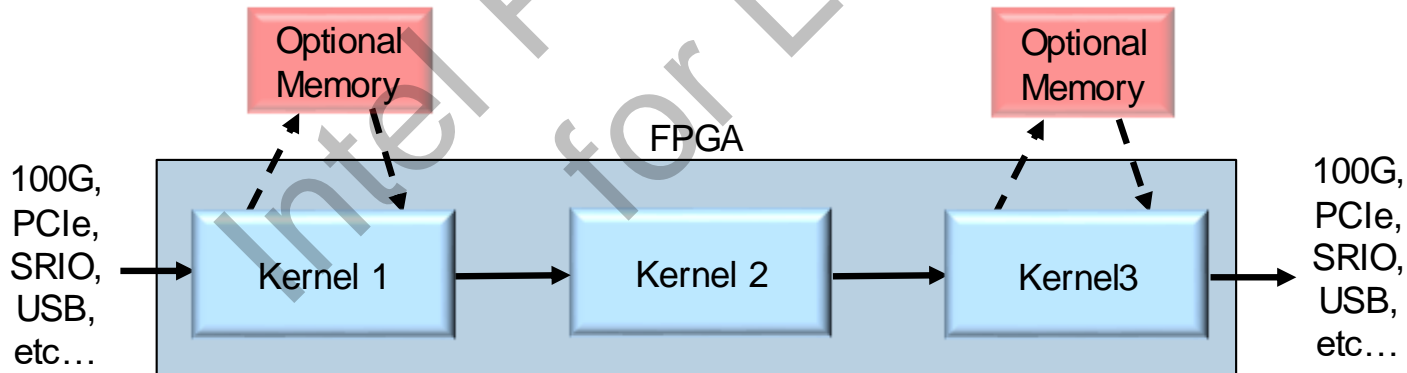
Data Movement in FPGAs

FPGA allows for result reuse between instructions

Ingress/Egress to custom functions 100% flexible

Multiple memory banks of various types directly off FPGA

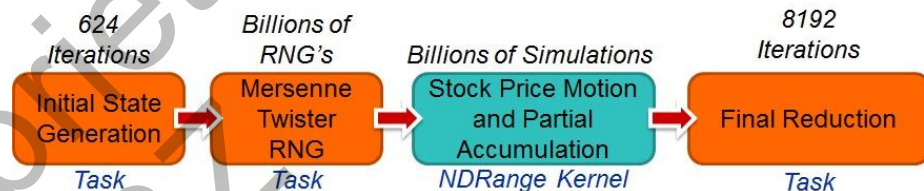
- Algorithms can be architected to minimize buffering to external memory or host memory
- Multiple optional memory banks can be used to allow simultaneous access



Example: Multi-Stage Pipeline

An algorithm may be divided into multiple kernels:

- Modular design patterns
- Partition the algorithm into kernels with different sizes and dimensions
- Algorithm may naturally split into both single-threaded and NDRange kernels



Generating random data for a Monte Carlo simulation:

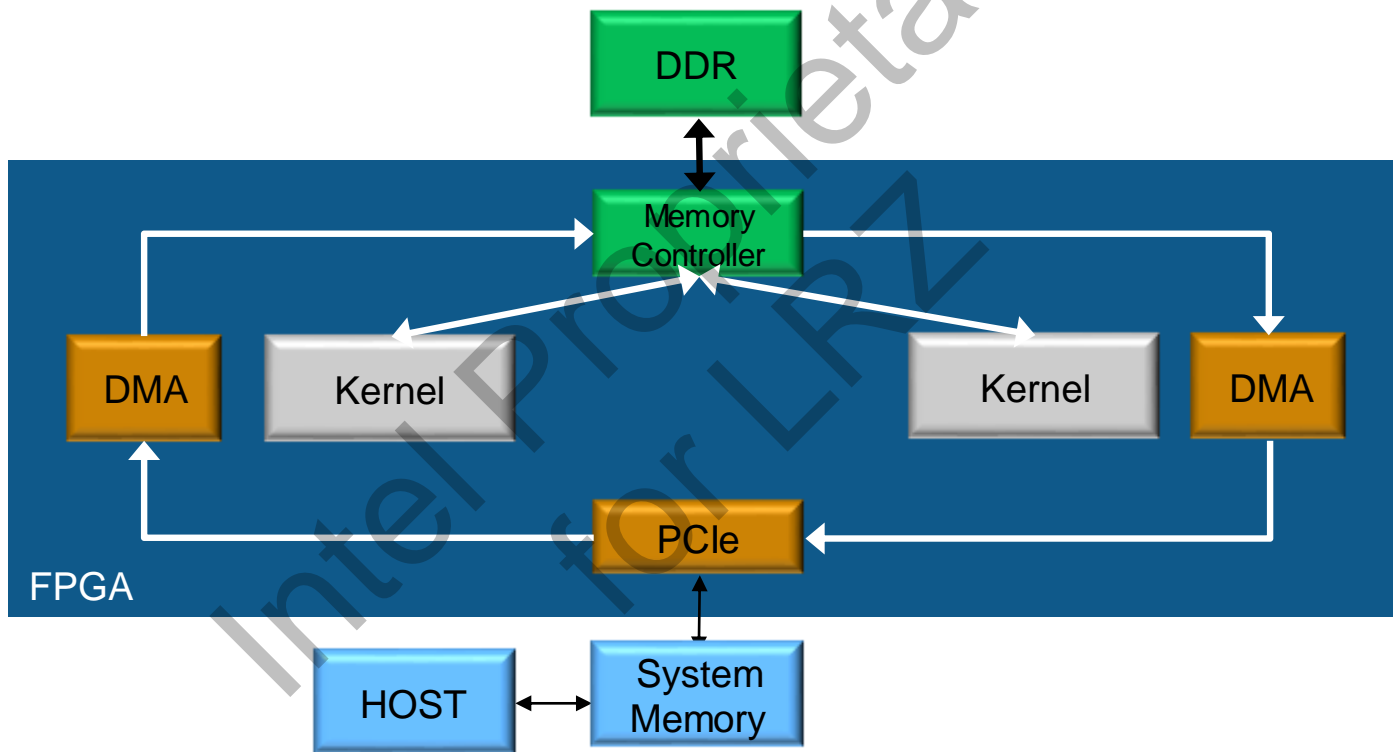
```
kernel void rng(int seed) {  
    int r = seed;  
    while(true) {  
        r = rand(r);  
        write_channel_altera(  
            RAND, r);  
    }  
}
```

Single-Threaded

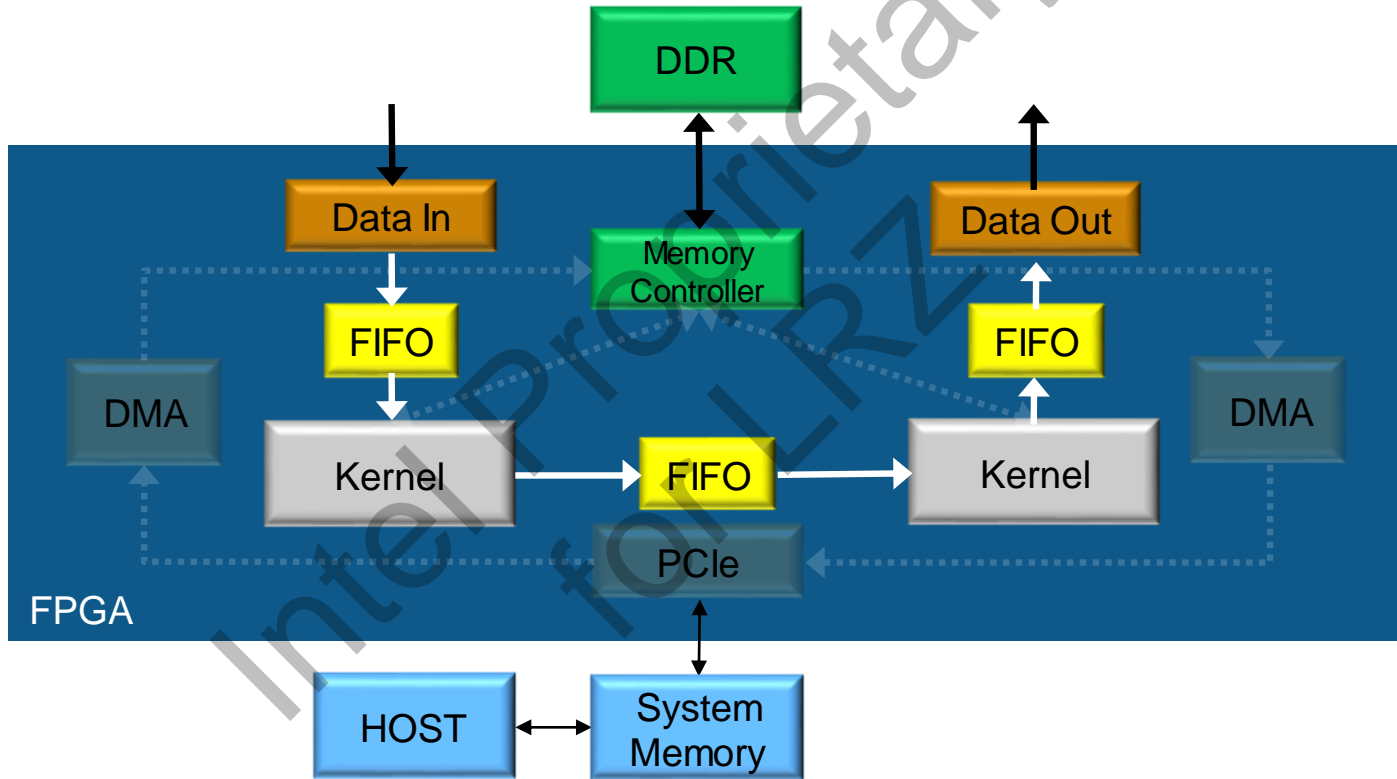
```
kernel void sim(...) {  
    int gid = get_global_id(0);  
    int rnd = read_channel_altera(  
        RAND);  
    out[gid] = do_sim(data, rnd);  
}
```

NDRange

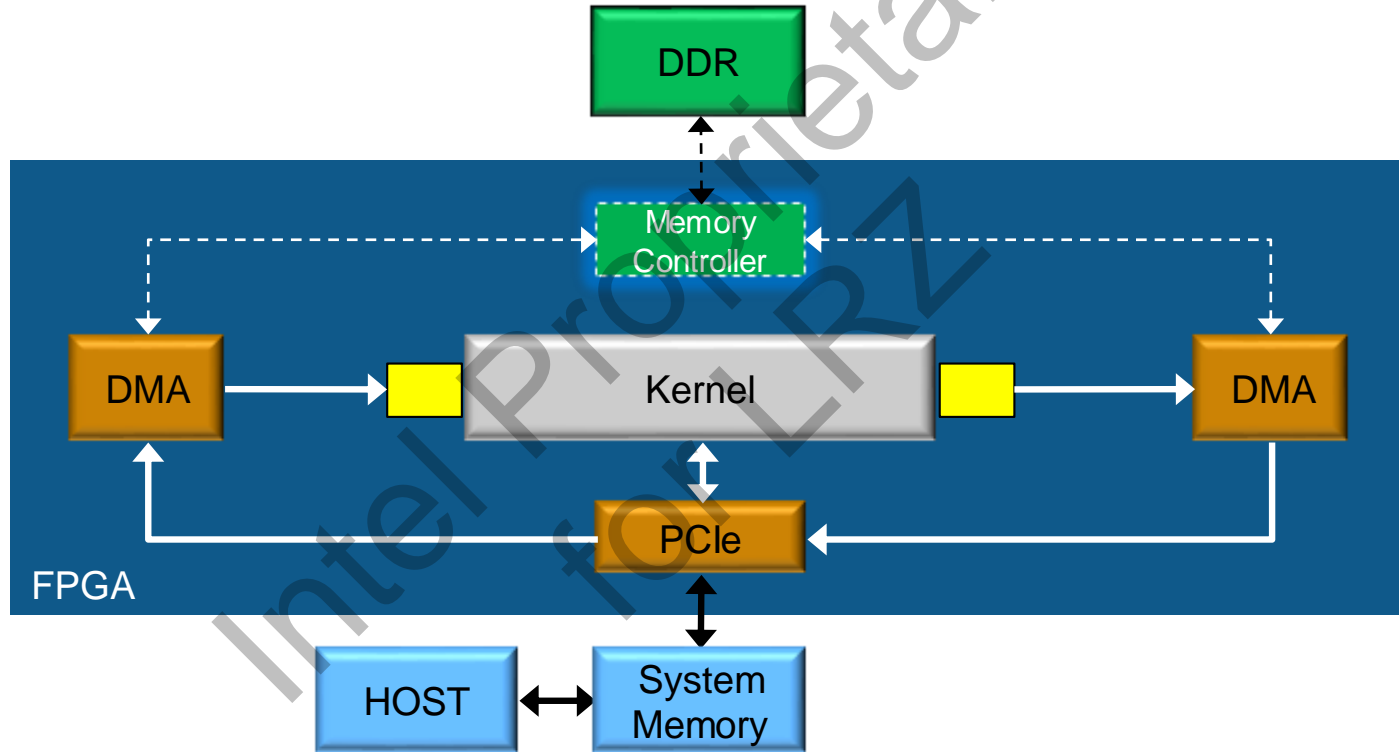
Traditional Data Movement Without Channels



Data Movement Using Channels



Data Movement Using Host Channels



An Even Closer Look: FPGA Custom Architectures

Kernel Replication with `num_compute_units` using OpenCL

- Step #1: Design an efficient kernel
- **Step #2: How can we scale it up?**

```
kernel void PE() {  
  
    ...  
}
```

PE

Processing element
(task-based kernel)

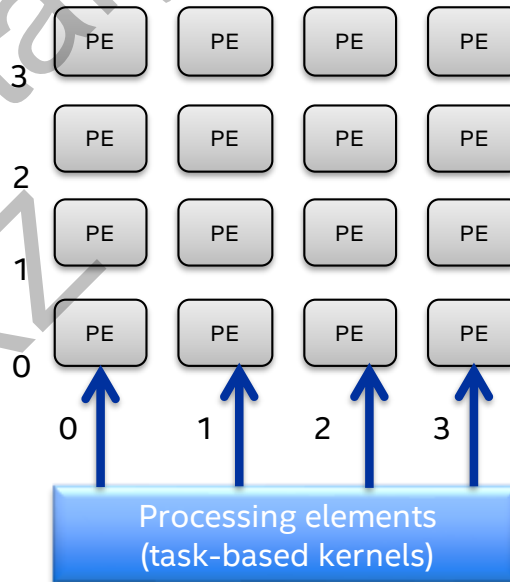
Kernel Replication With Intel® FPGA SDK for OpenCL

Attribute to specify 1-dim or 2-dim array of kernels

Add API to identify kernel in the array

```
__attribute__((num_compute_units(4,4)))  
kernel void PE() {  
  
    row = get_compute_id(0);  
    col = get_compute_id(1);  
  
    ...  
}
```

Compile-time constants
allows compiler to specialize each PE



Kernel Replication With Intel® FPGA SDK for OpenCL

Topology can be expressed with software constructs

- Channel connections specified through compute IDs

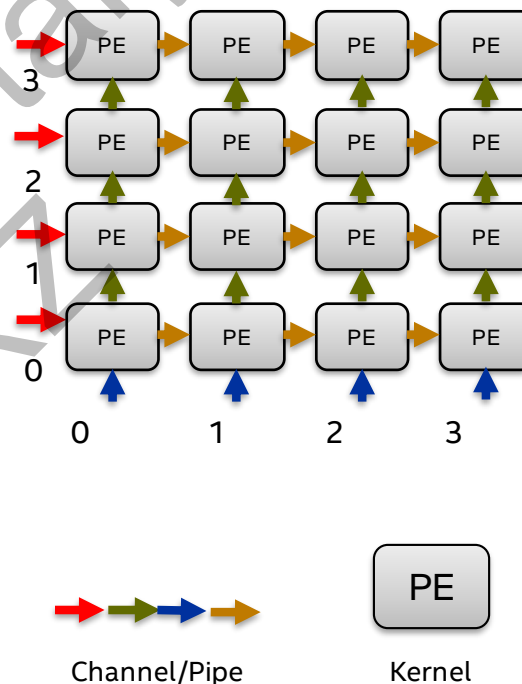
```
channel float4 ch_PE_row[4][4];
channel float4 ch_PE_col[4][4];
channel float4 ch_PE_row_side[4];
channel float4 ch_PE_col_side[4];

__attribute__((num_compute_units(4,4)))
kernel void PE() {
    row = get_compute_id(0);
    col = get_compute_id(1);

    float4 a,b;

    if (row==0)
        a = read_channel(ch_PE_col_side[col]);
    else
        a = read_channel(ch_PE_col[row-1][col]);

    if (col==0)
        ...
}
```



Matrix Multiply in OpenCL

Every PE / feeder is a kernel

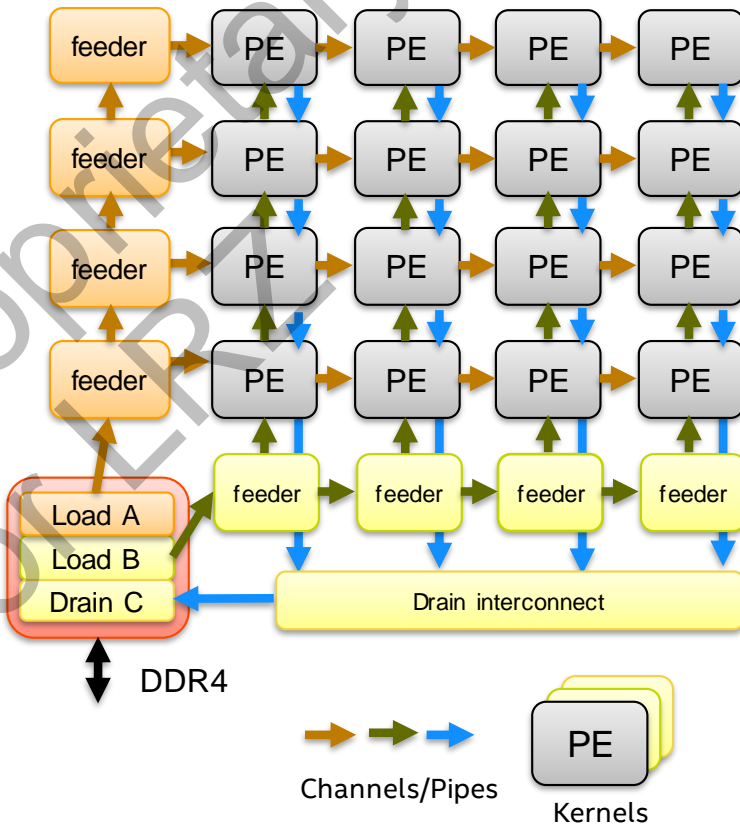
Communication via OpenCL channels

Data-flow network model

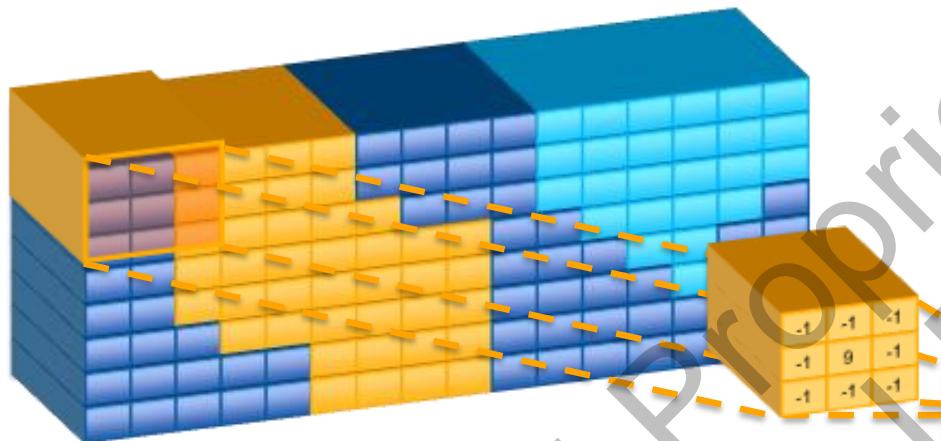
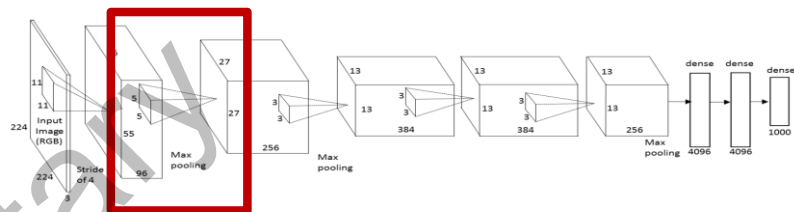
Software control:

- Compute unit granularity
- Spatial Locality
- Interconnect topology
- Data movement
- Caching
- Banking

Performance: ~1 TFLOPs



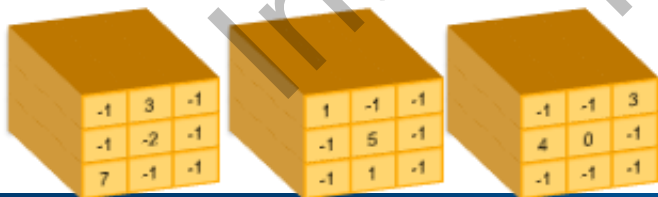
Traditional CNN



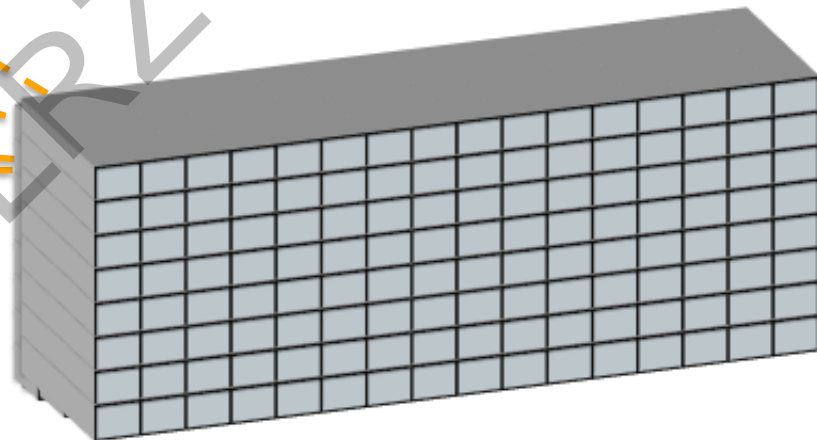
Input Feature Map
(Set of 2D Images)

Filter
(3D Space)

Output Feature Map



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



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

CNN On FPGA

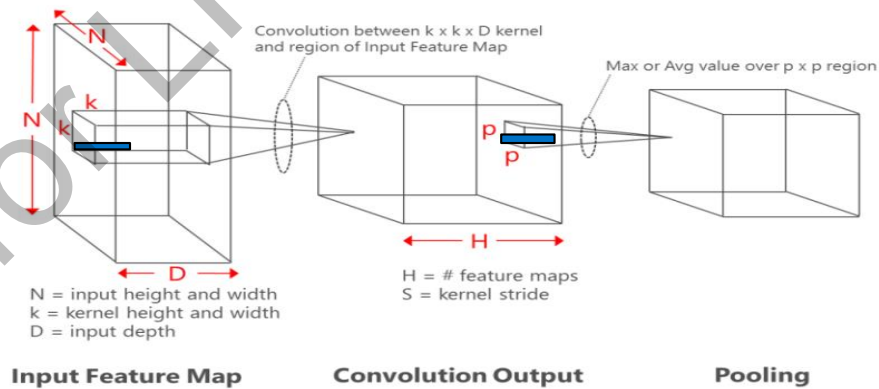
Want to minimize accessing external memory

Want to keep resulting data between layers on the device and between computations

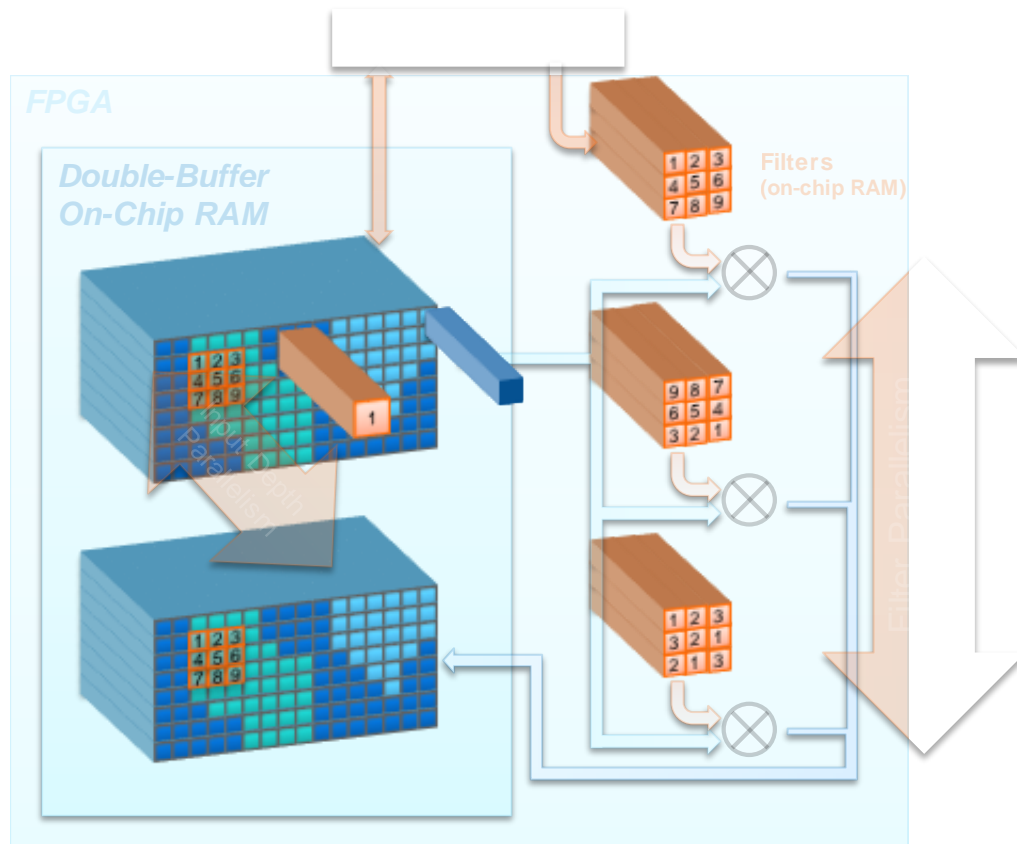
Want to leverage reuse of the hardware between computations

Parallelism in the depth of the kernel window and across output features.
Defer complex spatial math to random access memory.

Re-use hardware to compute multiple layers.



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

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

OPENCL FLOW

Lab 3

Intel Proprietary
for LRZ

FPGA PROGRAMMING MODEL:

DSP Builder Advanced Blockset

Intel Proprietary
for ERZ

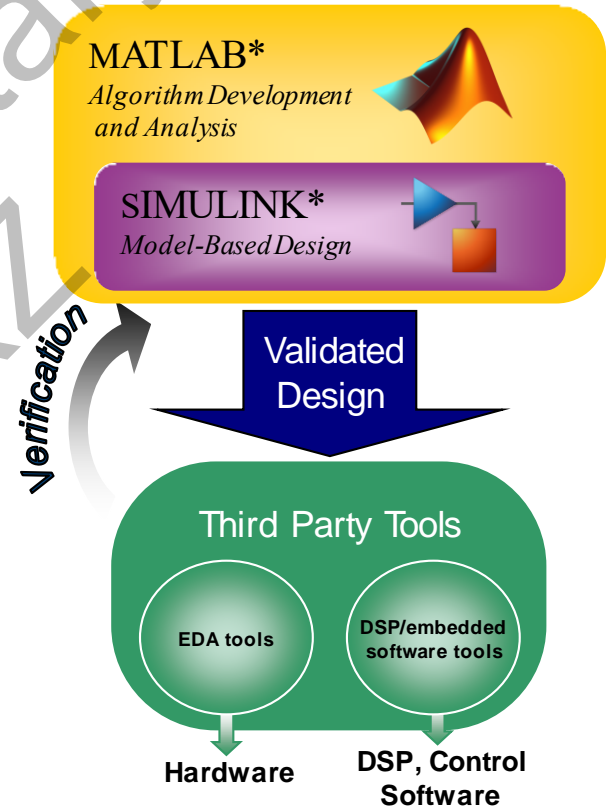
The Mathworks* Design Environment

■ Matlab*

- High-level technical computing language
 - Simple C like language
 - Efficient with vectors and matrices
 - Built-in mathematical functions
- Interactive environment for algorithm development
 - 2D/3D graphing tool for data visualization

■ Simulink*

- Hierarchical block diagram design & simulation tool
- Digital, analog/mixed signal & event driven
- Visualize signals
- Integrated with MATLAB*



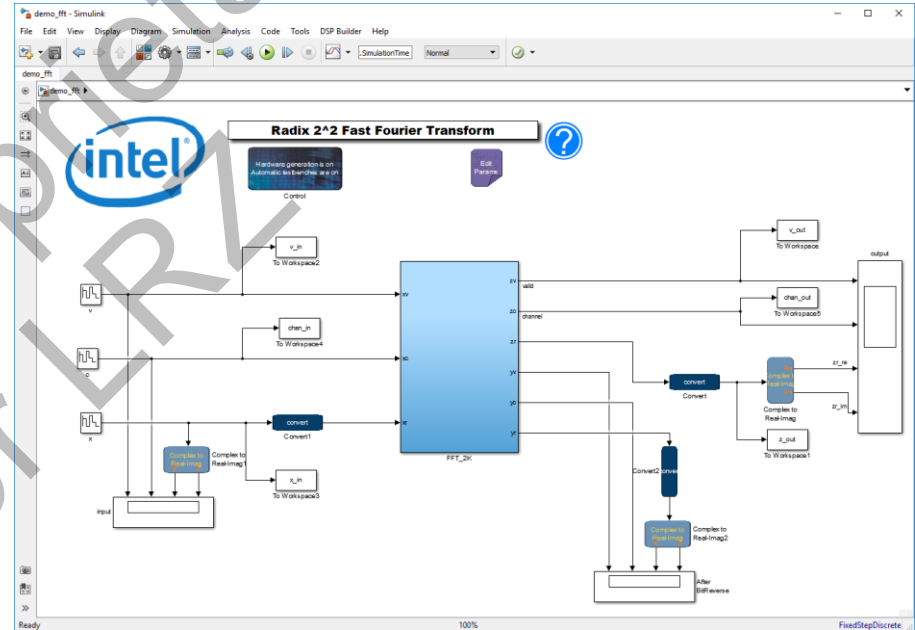
DSP Builder for Intel® FPGAs

Enables MathWorks* Simulink for Intel FPGA design

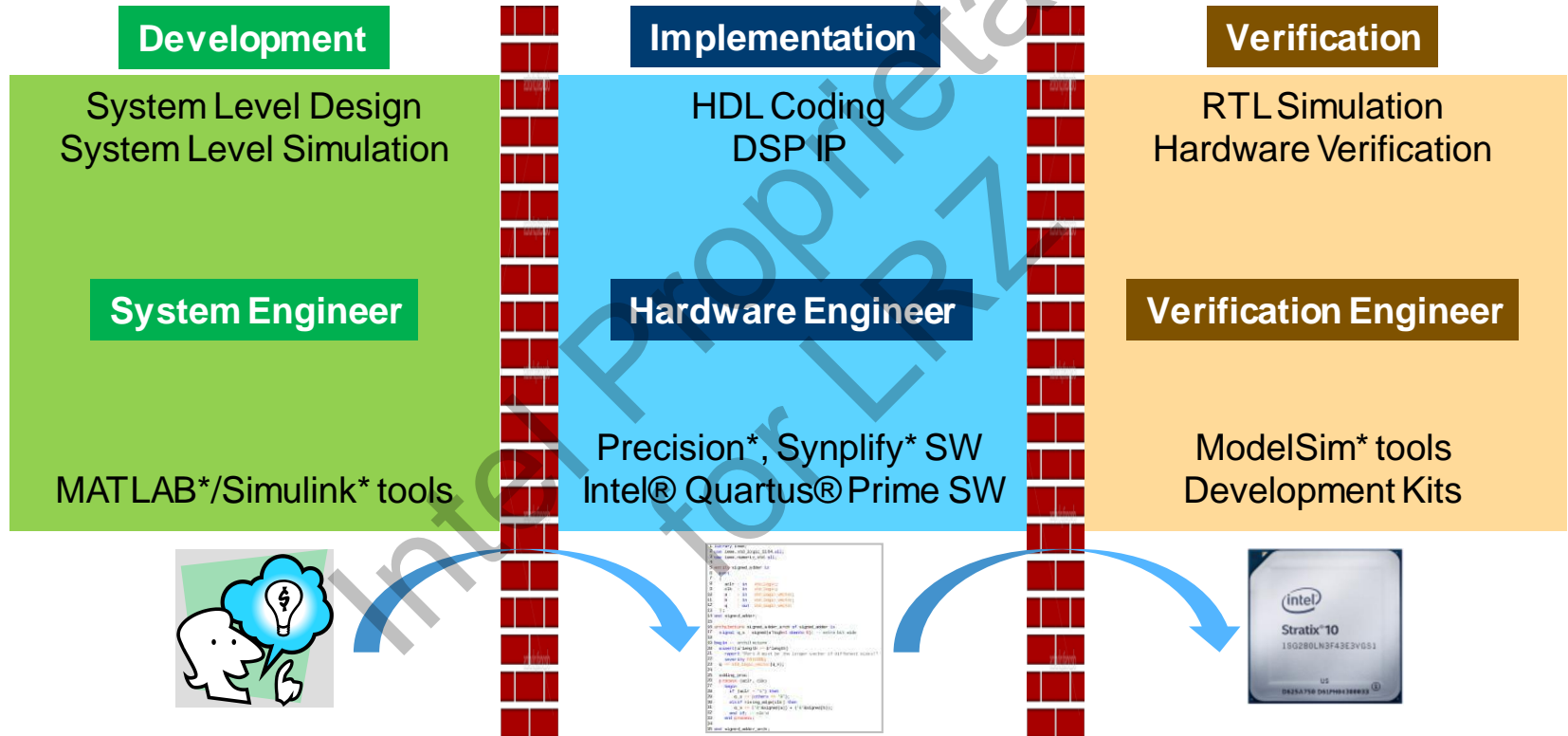
Device optimized Simulink* DSP Blockset

- Key Features:

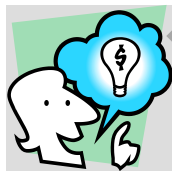
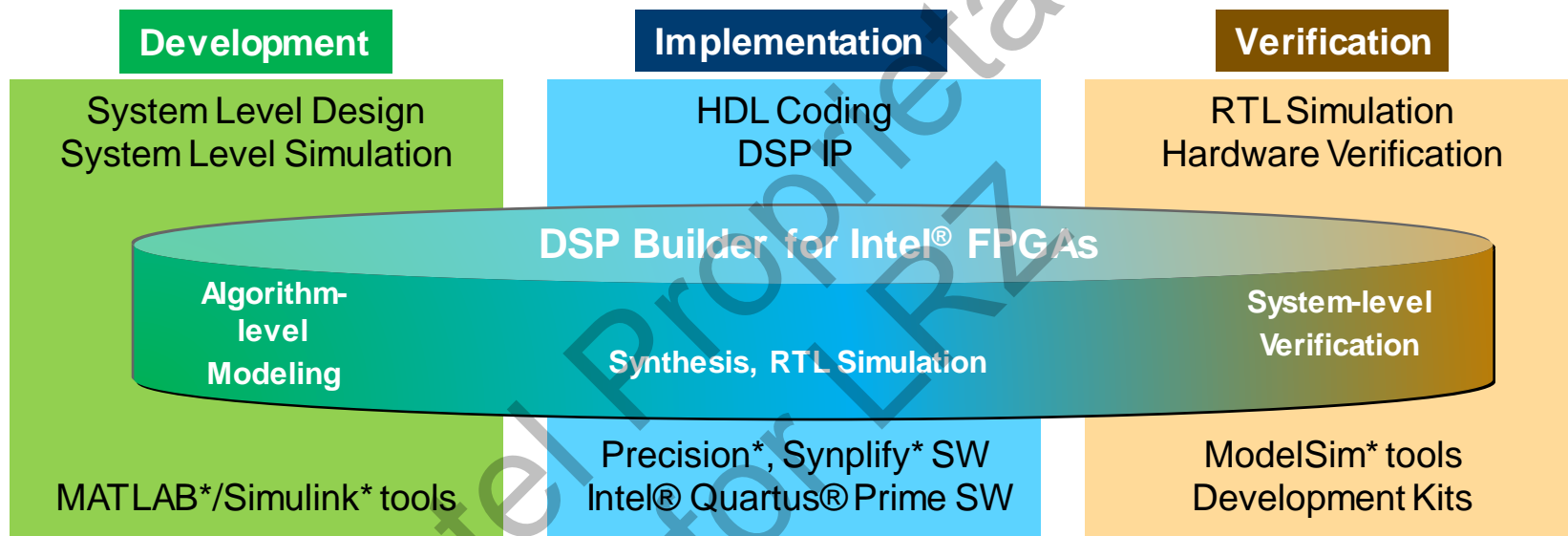
- High-Level Design Exploration
- HW-in-the-Loop verification
- IP Generation for Intel® Quartus SW / Platform Designer



FPGA Design Flow - Traditional



FPGA Design Flow – DSP Builder for Intel® FPGAs



**Single Simulink*
Representation**



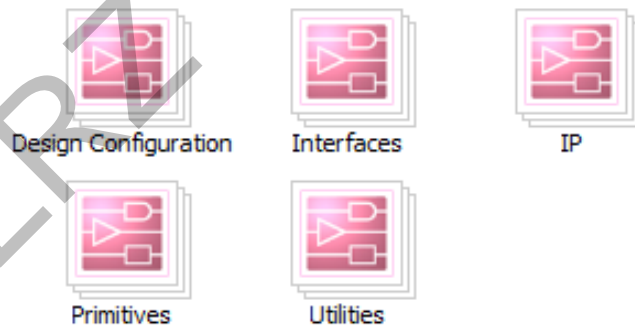
Core Technologies

- IP (ready made) library
 - Multi-rate, multi-channel filters
 - Waveform synthesis (NCO/DDS/Mixers)
- Custom IP creation using primitive library
 - Vectorization
 - Zero latency
 - Scheduled
 - Aligned RTL generation
- System integration
 - Platform Designer
 - Processor Integration
- Automatic pipelining
- Automatic folding and resource sharing
- Multichannel designs with automatic vectorization
- Avalon® Memory-Mapped and Streaming Interfaces
- Design exploration across device families
- High-performance floating-point designs
- System-in-the-Loop accelerated simulation

Advanced Blockset - High Performance DSP IP

Over 150 device optimized DSP building blocks for Intel® FPGAs

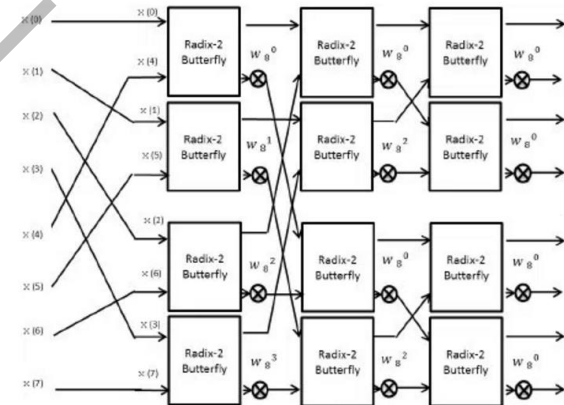
- DSP building blocks
- Interfaces
- IP library blocks
- Primitives library blocks
 - Math and Basic blocks
- Vector and Complex data types



Build Custom FFTs from FFT Element Library

- Quickly build DSP designs using Complete FFT IP Functions from the FFT Library
- Build custom radix-2² FFTs using blocks from the FFT Element Library

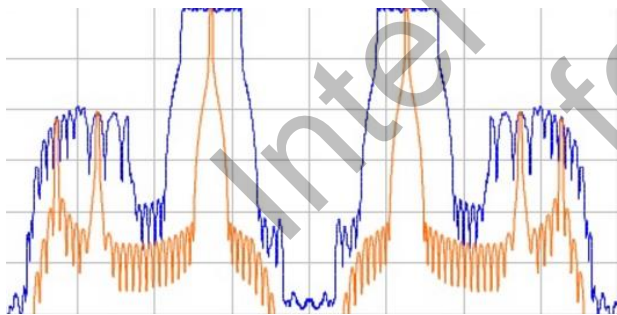
FFT IP Library	FFT Element Library
FFT	Pruning and Twiddle
FFT_float	Bit vector combine
VFFT	Butterfly Unit
VFFT_float	Choose Bits
BitReverseCoreC	Dual Twiddle Memory
VariableBitReverse	Edge Detect
	Floating-Point Twiddle Gen
	Crossover Switch



Filter and Waveform Synthesis Library

DSP Builder includes a comprehensive waveform IP library

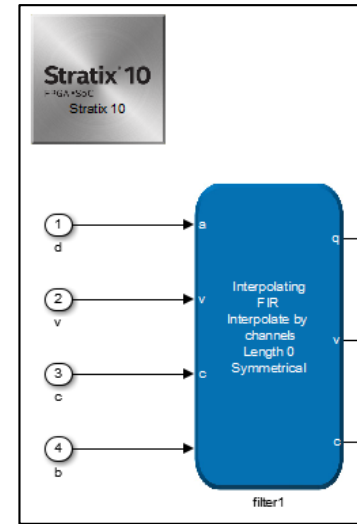
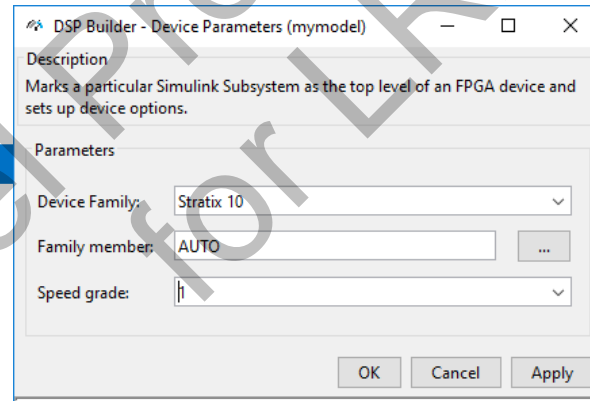
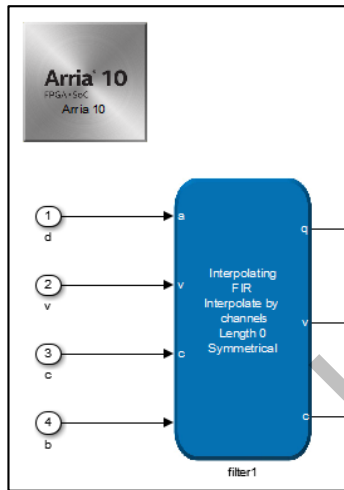
- Automatic resource sharing based on sample rate
- Support for super sample rate architectures



IP	Implementations
FIR	<ul style="list-style-type: none">• Half-band• L-Band• Symmetric• Decimating• Fractional Rate• Interpolation• Single-Rate• Super Sample Rate
CIC	<ul style="list-style-type: none">• Decimating• Interpolating• Super Sample Rate
Mixer	<ul style="list-style-type: none">• Complex• Real• Super Sample Rate
NCO	<ul style="list-style-type: none">• Super Sample Rate• Multi-bank

Library is Technology Independent

- Target device using a **Device** block
- Same model generates optimized RTL for each FPGA and speed grade

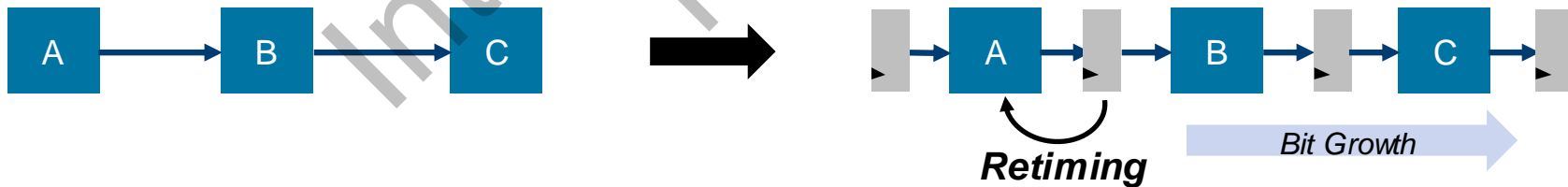
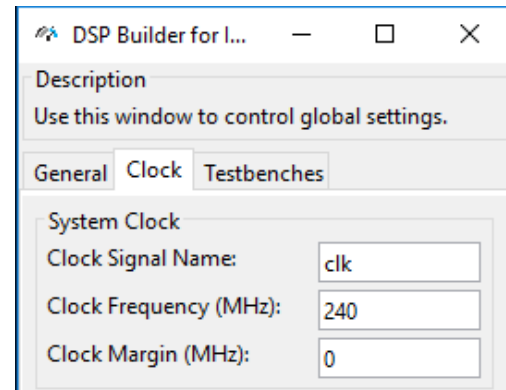


Datapath Optimization for Performance

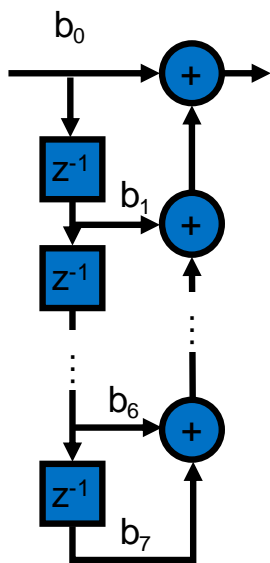
Automatic Timing Driven Synthesis of Model

- Based on specified device and clock frequency

Optimization	Description
Pipelining	Inserts registers to improve Fmax
Algorithmic Retiming	Moves registers to balance pipelining
Bit Growth Management	Manages bit growth for fixed-point designs
Multi-rate Optimizations	Optimizes hardware based on sample rate

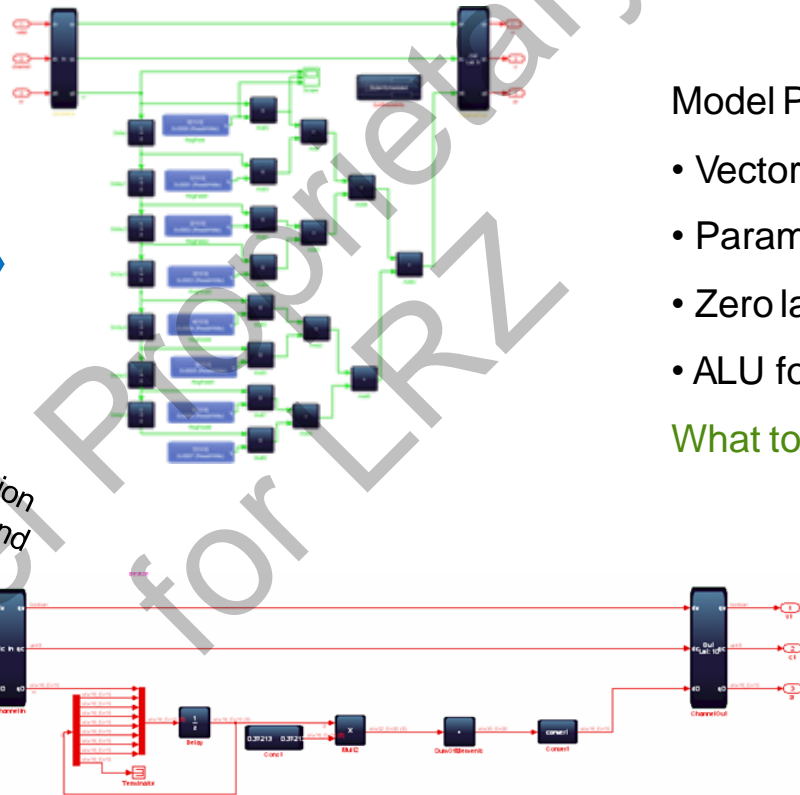


Custom IP Generation



Textbook based design entry

Schematic simplification using vector support and zero latency blocks



Model Primitive Features

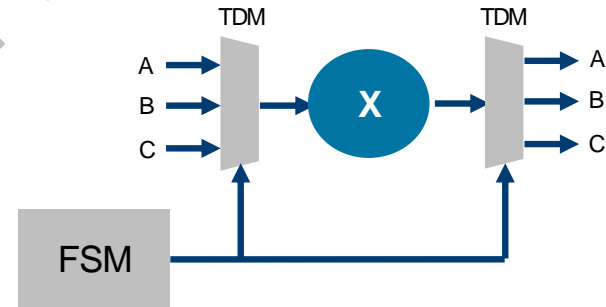
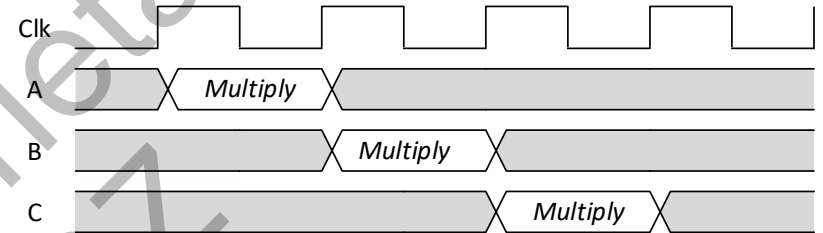
- Vector support
- Parameterizable
- Zero latency block
- ALU folding

What to do not when to do it

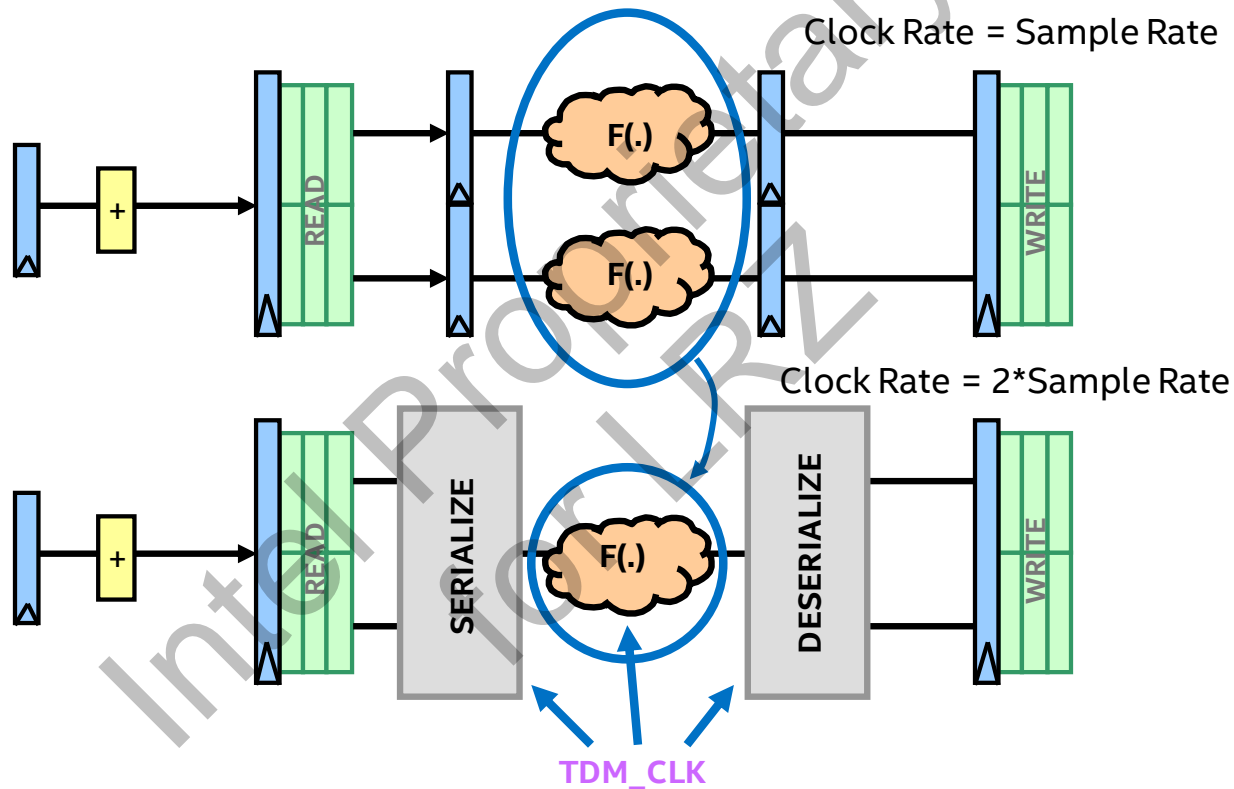
ALU Design Folding Improves Area Efficiency

Optimizes hardware usage for low-throughput designs

- Arranges one of each resources in a central arithmetic logic unit (ALU) fashion
- Folding factor = clock rate / data rate
- Performed when Folding factor > 500



TDM Resource Sharing



TDM Design: Trade-Off Example

49-tap Symmetric Single Rate FIR Filter				
Stratix 10	Resources			
	LUT4s	Mults	Memory bits	TDM Factor
Clock Rate = 72 MHz Sample Rate = 72 MSPS	898	26	0	1
Clock Rate = 144 MHz Sample Rate = 72 MSPS	1082	14	0	2
Clock Rate = 288 MHz Sample Rate = 72 MSPS	741	8	0	4
Clock Rate = 72 MHz Sample Rate = 36 MSPS	1082	14	0	2

2 Antenna DUC Reference Design

Clock Rate = 179.2MHz

ChannelFIR:

ChanCount = 4

Output Sample Rate = 11.2 MSPS

Output Period = 16

Output Seq. = I1, I2, Q1, Q2, zeros(1, 16-4)

Interpolate4FIR:

ChanCount = 4

Output Sample Rate = 89.6 MSPS

Output Period = 2

ChanWireCount = ceil(4/2) = 2

ChanCycleCount = ceil(4/2) = 2

Output Seq. = I1, I2

Q1, Q2

ComplexMixer:

ChanCount = 2 (complex channel)

Sample Rate = 89.6 MSPS

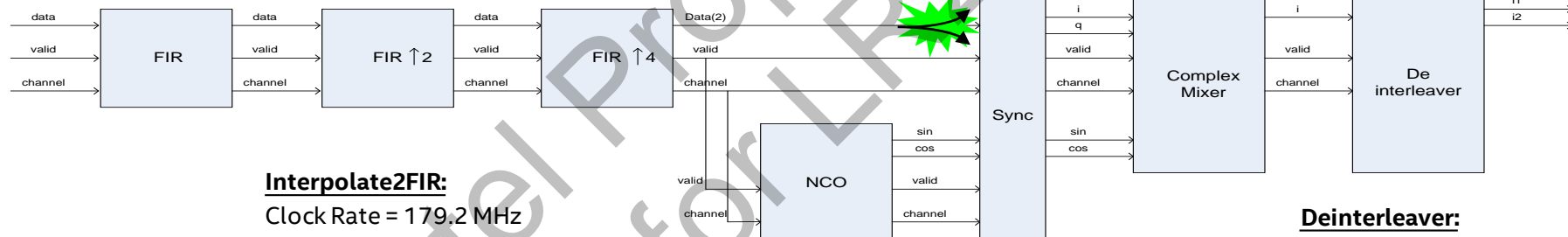
Period = 2

$I' = I * \cos - Q * \sin$

$Q' = I * \sin + Q * \cos$

Output i Seq. = I1, I2

Output q Seq. = Q1, Q2 (Terminated)



Interpolate2FIR:

Clock Rate = 179.2 MHz

ChanCount = 4

Output Sample Rate = 22.4 MSPS

Output Period = 8

Output Seq. = I1, I2, Q1, Q2, zeros(1, 8-4)

NCO:

ChanCount = 2 (complex channel)

Sample Rate = 89.6 MSPS

Period = 2

Sine Seq. = sinA1, sinA2

Cosine Seq. = cosA1, cosA2

Deinterleaver:

Sample Rate = 89.6 MSPS

Period = 2

Input I Seq. = I1, I2

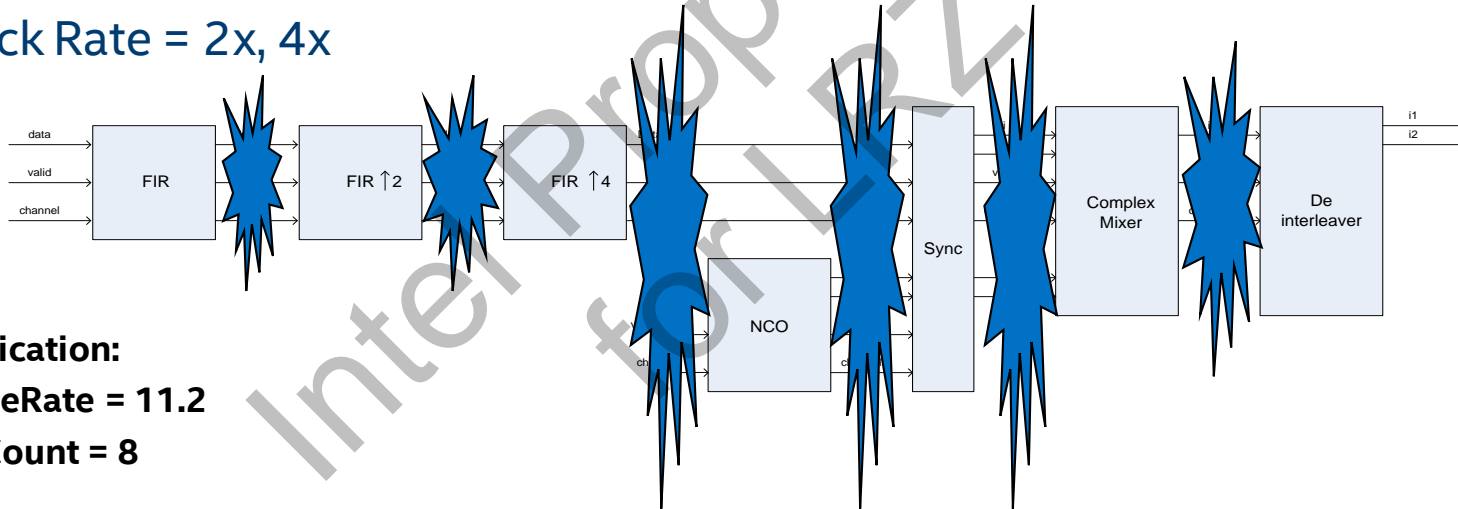
Antenna 1 Seq. = I1, -

Antenna 2 Seq. = I2, -

Reference Design Included with DSP Builder

Changing the Design without DSP Builder

- Tedious and time consuming
- Channel Count = 8, 16, 32
- Clock Rate = 2x, 4x



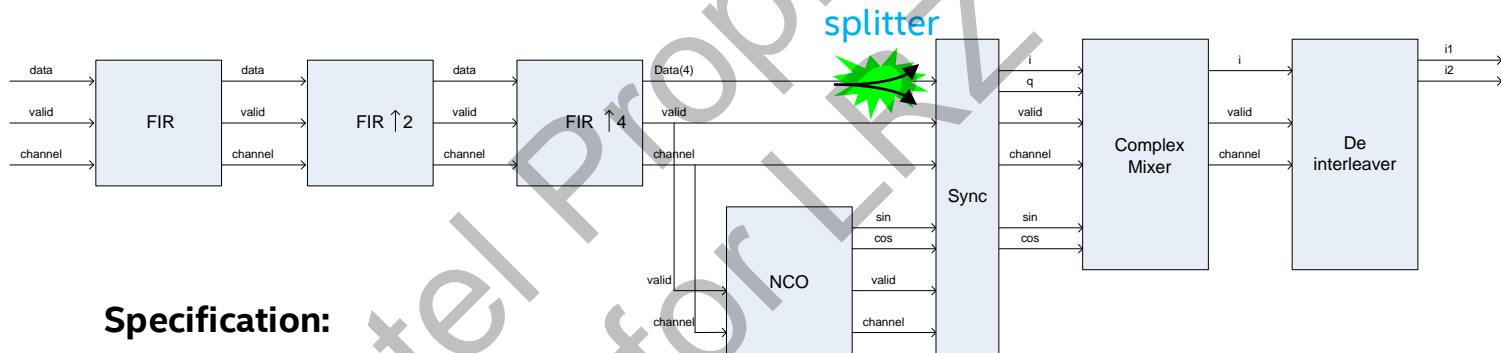
Specification:

SampleRate = 11.2

ChanCount = 8

Changing the Design with DSP Builder

- Modifications done in minutes
- Design still looks the same



Specification:

SampleRate = 11.2

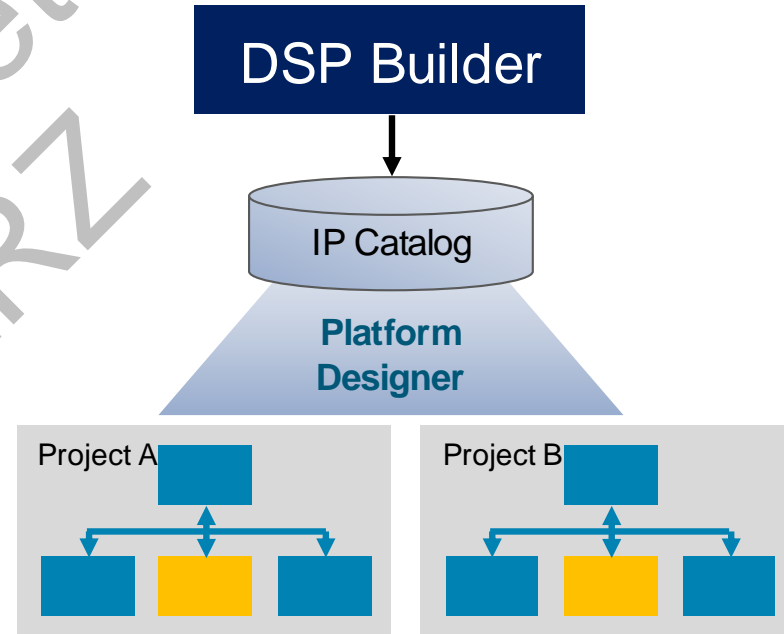
ChanCount = 8

Five Designs Iterations < 1 Hour

	Arria® 10 6 channel	Arria 10 6 channel	Arria 10 12 channel	Stratix® 10 6 channel	Stratix 10 12 channel
Requested Clock (MHz)	250	450	450	450	450
Actual Fmax (slow model, 85C)	351	458	458	524	484.5
Multiplier Count (18x18)	10	6	10	6	10
Logic Resources (registers)	686	465	818	1267	1863
Block Memory Resources (kbits)	0	0	0	0	25.8

Generates Reusable IP for Platform Designer

- Platform Designer is the System Integration Environment for Intel® FPGAs
- DSP Builder designs fully compatible with Platform Designer
- Integrate with other FPGA IPs
 - Processors
 - State machines
 - Streaming interfaces
- Design reuse fully supported



Typical Design Flow

Identify system architecture, design filters and choose desired Fmax and device

Set the top level system parameters in the MATLAB® software using the 'params' file - number of channels, performance, etc.

Build the system using the Advanced Blockset tool

Simulate the design using Simulink® and ModelSim® tools

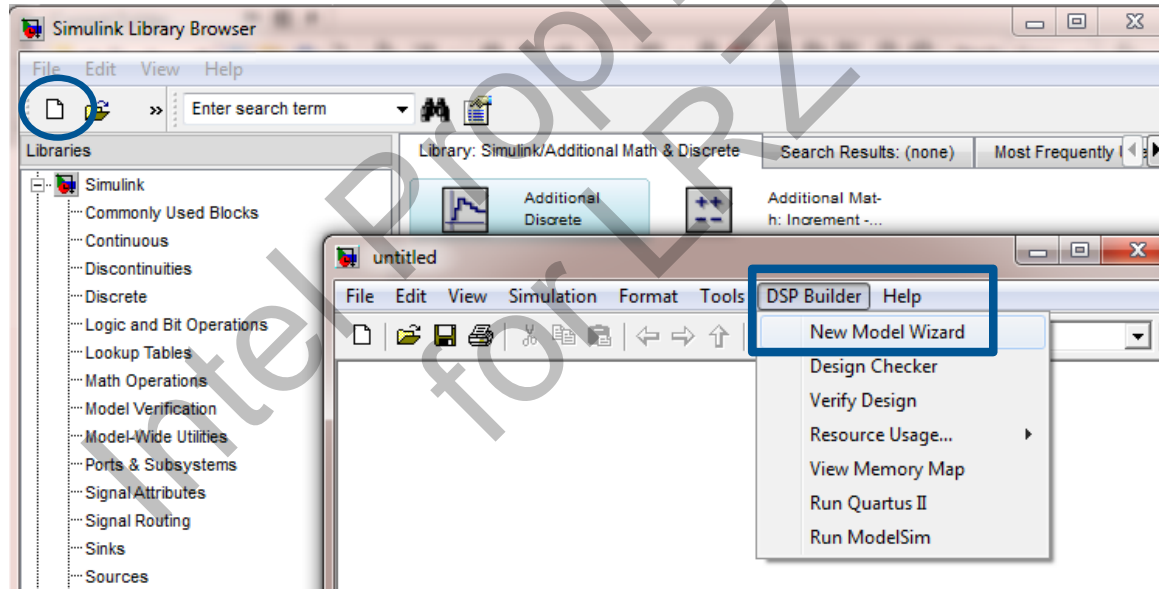
Target the right FPGA family and compile

As system design specs changes, edit the 'params' file and repeat

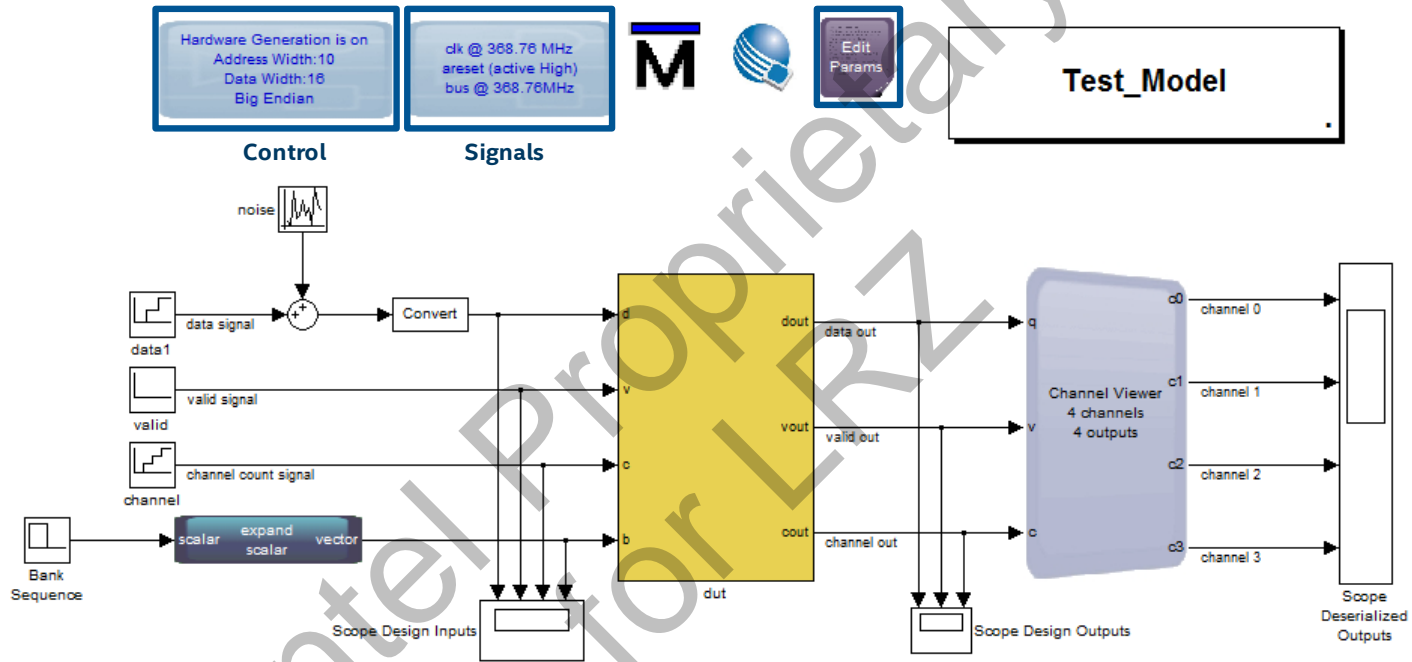
Design Flow – Create Model

Create a new blank model

Select New Model Wizard from DSP Builder menu

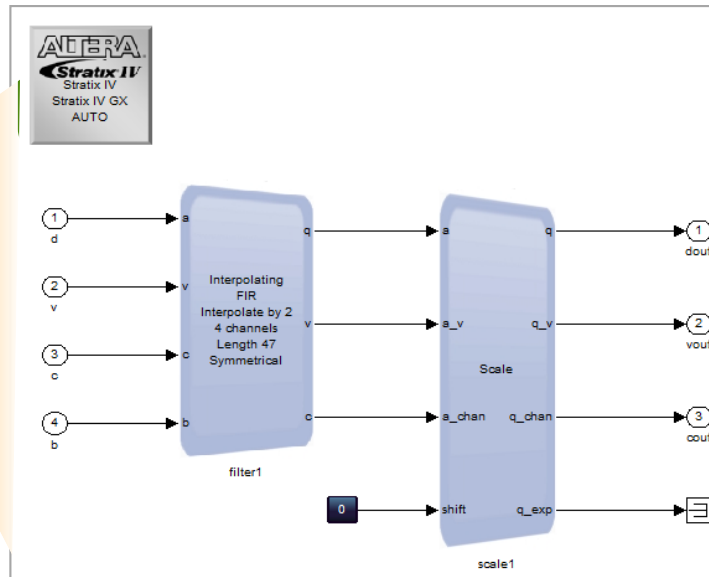
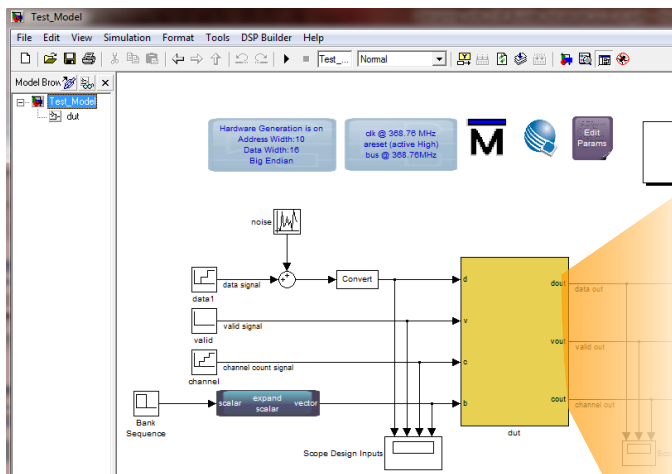


Top Level Testbench



Top-level of a DSPB-AB design is a testbench
Must include Control and Signals blocks

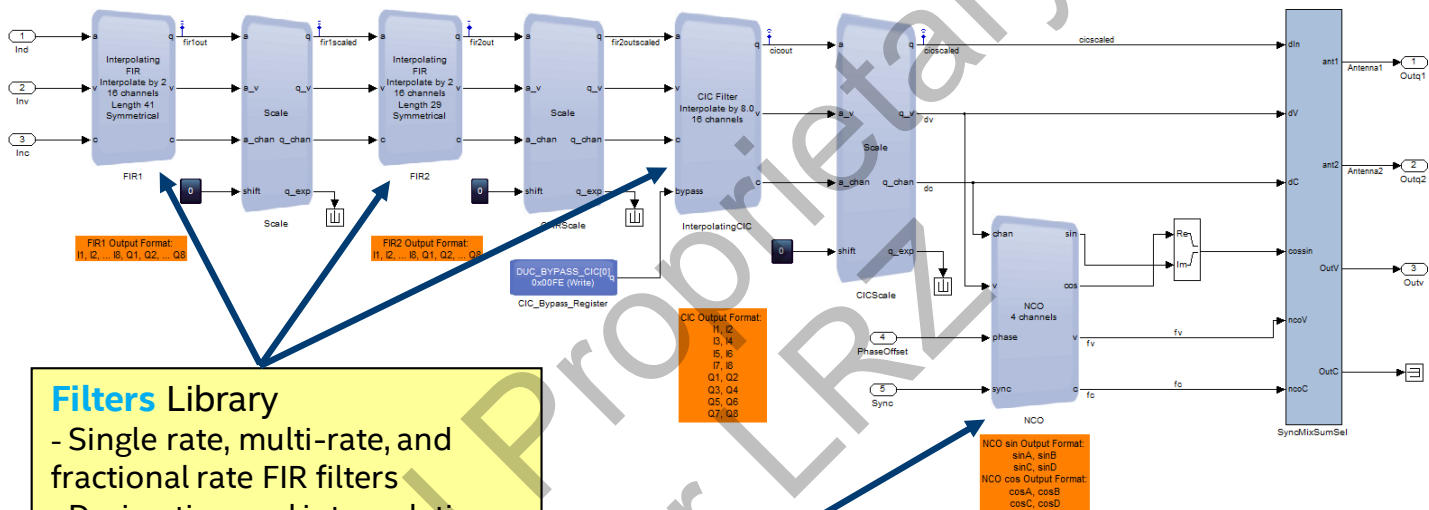
Design Flow - Synthesizable Model



Enter the design in the subsystem

Device block marks the top level of the FPGA

Design Flow – ModelIP Blocks



Filters Library

- Single rate, multi-rate, and fractional rate FIR filters
- Decimating and interpolating cascaded integrator comb (CIC) filters

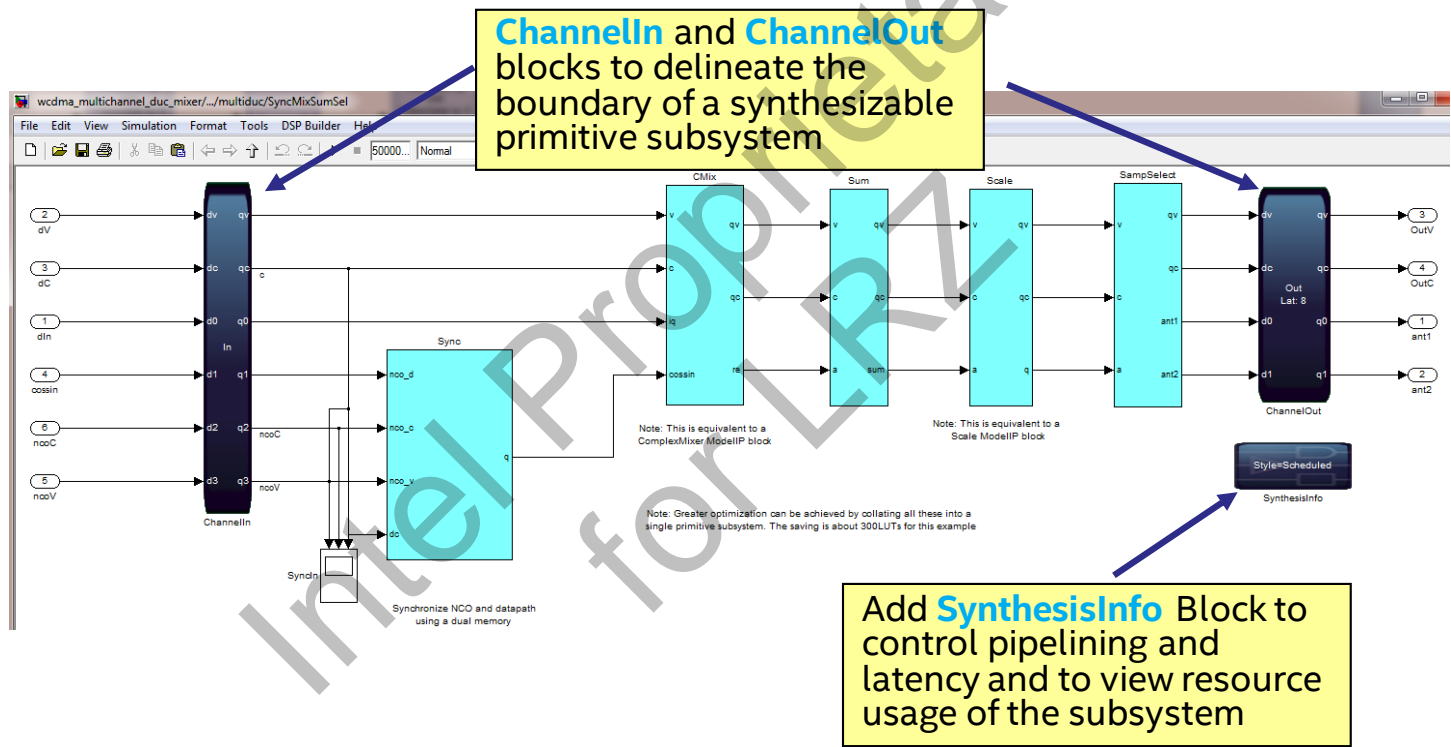
Note: Supports super-sample rate (data rate > system clock freq) interpolation by 2 filters.

Waveform Synthesis Library

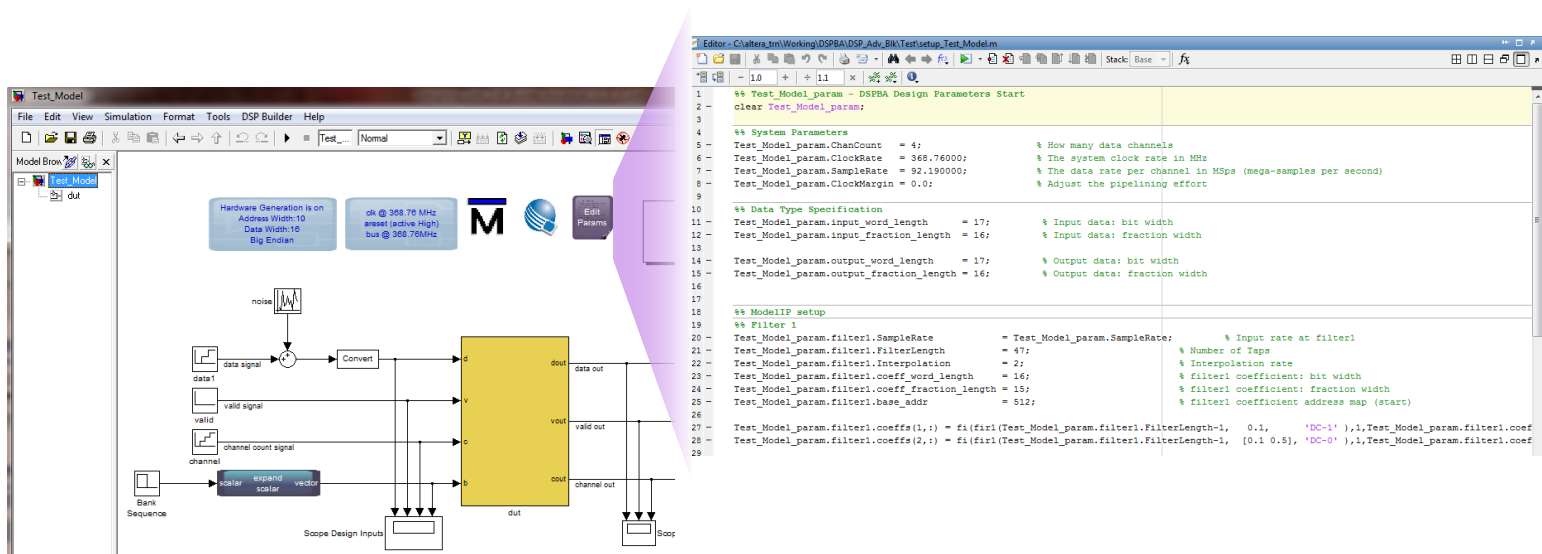
- Real and complex mixer
- Numerically controlled oscillator (NCO)

Note: The NCO block supports frequency hopping (each channel can hop to different frequency from a pool of frequencies)

Design Flow – ModelPrim Blocks



Design Flow – Parameterize the Design



The image displays a Simulink model titled 'Test_Model' and its corresponding C structure-like parameter template. The Simulink model on the left shows a block diagram with inputs like 'data signal', 'valid signal', and 'channel count signal', and outputs like 'data out', 'valid out', and 'channel out'. A 'Scope Design Inputs' block is also visible. The C template on the right, titled 'Test_Model_param - DSPBA Design Parameters Start', defines various parameters for the design, including system parameters, data type specifications, and filter setup.

```
1  %% Test_Model_param - DSPBA Design Parameters Start
2  clear Test_Model_param;
3
4  %% System Parameters
5  Test_Model_param.ChanCount = 4;           % How many data channels
6  Test_Model_param.ClockRate = 368.76000; % The system clock rate in MHz
7  Test_Model_param.SampleRate = 92.190000; % The data rate per channel in MSPs (mega-samples per second)
8  Test_Model_param.ClockMargin = 0.0;     % Adjust the pipelining effort
9
10 %% Data Type Specification
11 Test_Model_param.input_word_length = 17; % Input data: bit width
12 Test_Model_param.input_fraction_length = 16; % Input data: fraction width
13
14 Test_Model_param.output_word_length = 17; % Output data: bit width
15 Test_Model_param.output_fraction_length = 16; % Output data: fraction width
16
17
18 %% ModelIP setup
19 %% Filter 1
20 Test_Model_param.filter1.SampleRate = Test_Model_param.SampleRate; % Input rate at filter1
21 Test_Model_param.filter1.FilterLength = 47; % Number of Taps
22 Test_Model_param.filter1.Interpolation = 2; % Interpolation rate
23 Test_Model_param.filter1.coeff_word_length = 16; % filter1 coefficient: bit width
24 Test_Model_param.filter1.coeff_fraction_length = 15; % filter1 coefficient: fraction width
25 Test_Model_param.filter1.base_addr = 512; % filter1 coefficient: address map (start)
26
27 Test_Model_param.filter1.coeffs(1,1) = f1(fir1(Test_Model_param.filter1.FilterLength-1, 0.1, 'DC-1'),1,Test_Model_param.filter1.coeff
28 Test_Model_param.filter1.coeffs(2,1) = f1(fir1(Test_Model_param.filter1.FilterLength-1, [0.1 0.5], 'DC-0'),1,Test_Model_param.filter1.coeff
29
```

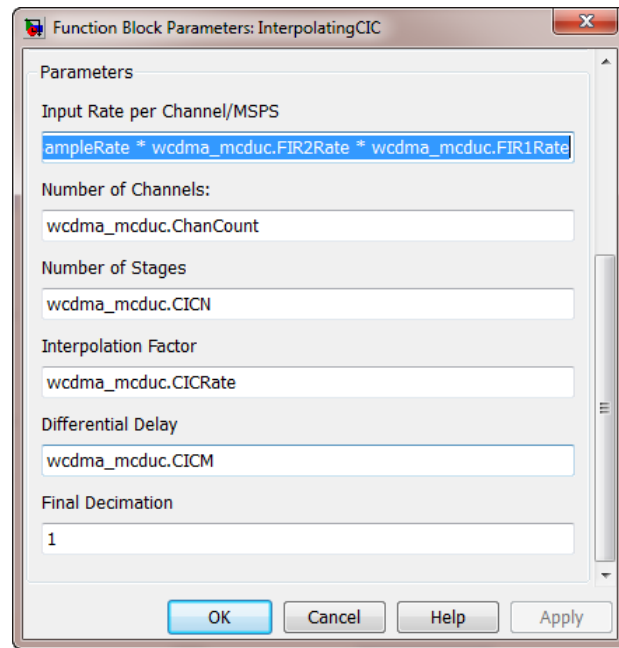
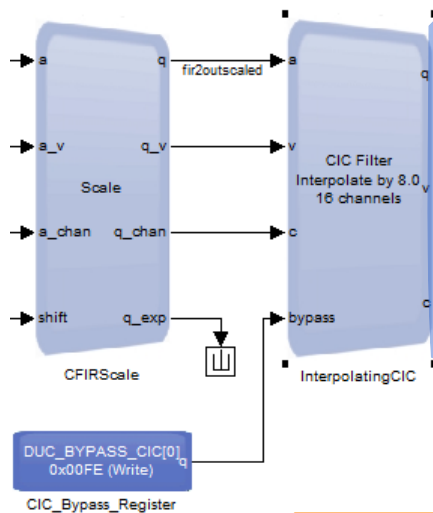
C structure like template

Runs when model is opened or simulation is run

Design Flow – Processor Interface

Drop memory and registers in the design

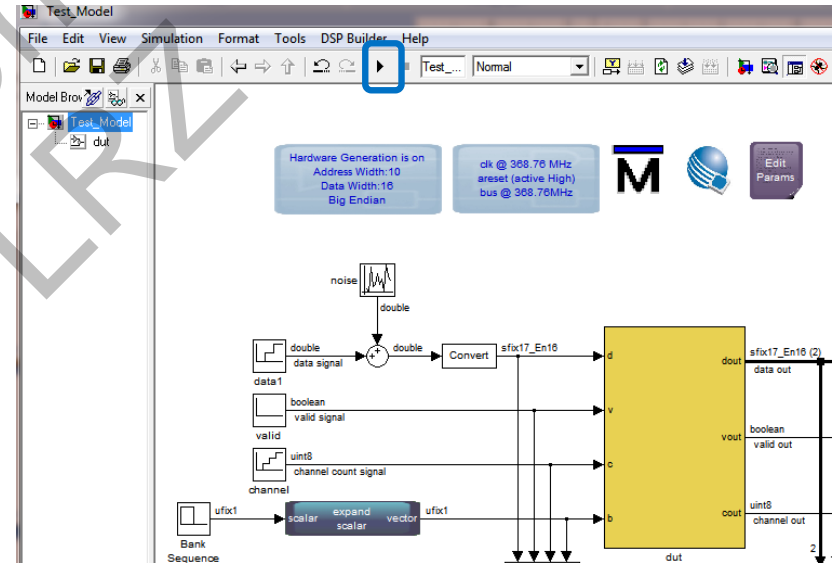
ModelIPs have built in memory mapped interface to control registers, coefficient registers



Design Flow - Running Simulink Simulation

Creates files in location specified by Control block

- VHDL Code
- Timing constraints file (.sdc)
- DSPB-AB subsystem Quartus® IP file

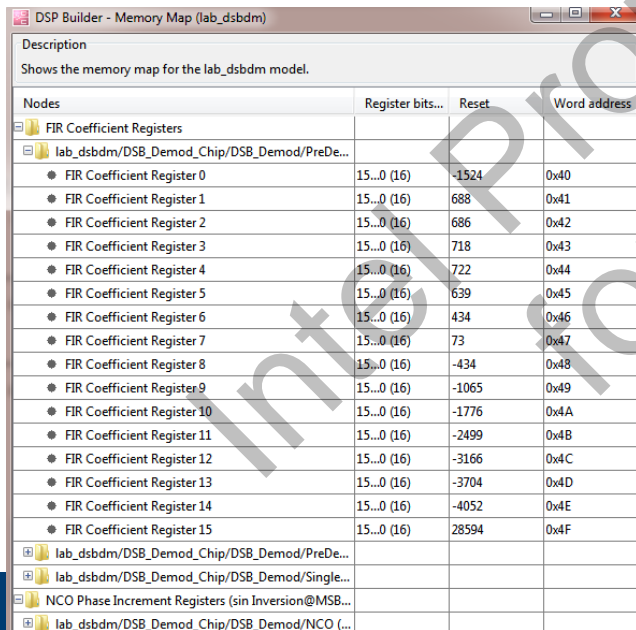


Design Flow - Documentation Generation

Get accurate resource utilization of all modules right after simulation, without place & route

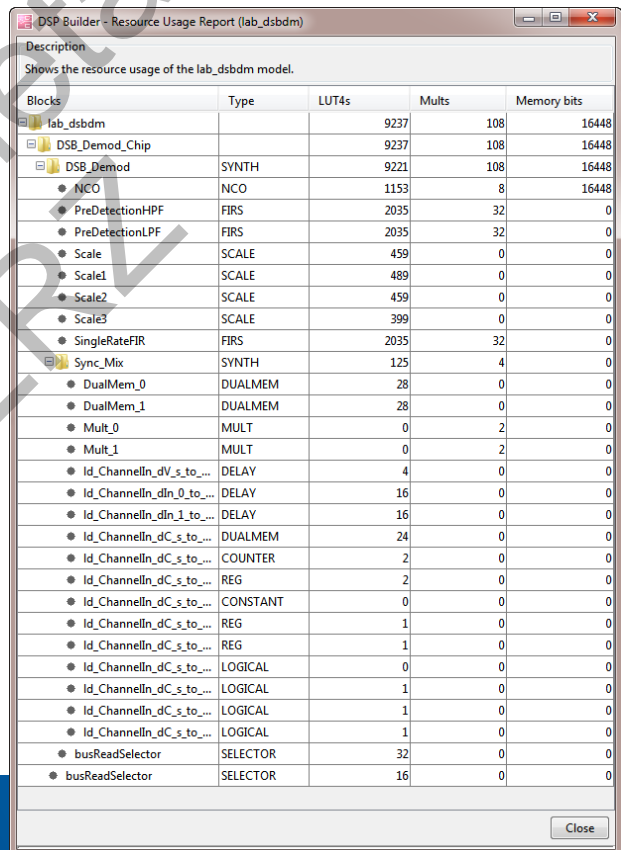
DSP Builder > Resource Usage

DSP Builder > View Address Map



Description
Shows the memory map for the lab_dsbdm model.

Nodes	Register bits...	Reset	Word address
lab_dsbdm/DSB_Demod_Chip/DSB_Demod/PreDe...			
● FIR Coefficient Register 0	15...0 (16)	-1524	0x40
● FIR Coefficient Register 1	15...0 (16)	688	0x41
● FIR Coefficient Register 2	15...0 (16)	686	0x42
● FIR Coefficient Register 3	15...0 (16)	718	0x43
● FIR Coefficient Register 4	15...0 (16)	722	0x44
● FIR Coefficient Register 5	15...0 (16)	639	0x45
● FIR Coefficient Register 6	15...0 (16)	434	0x46
● FIR Coefficient Register 7	15...0 (16)	73	0x47
● FIR Coefficient Register 8	15...0 (16)	-434	0x48
● FIR Coefficient Register 9	15...0 (16)	-1065	0x49
● FIR Coefficient Register 10	15...0 (16)	-1776	0x4A
● FIR Coefficient Register 11	15...0 (16)	-2499	0x4B
● FIR Coefficient Register 12	15...0 (16)	-3166	0x4C
● FIR Coefficient Register 13	15...0 (16)	-3704	0x4D
● FIR Coefficient Register 14	15...0 (16)	-4052	0x4E
● FIR Coefficient Register 15	15...0 (16)	28594	0x4F
lab_dsbdm/DSB_Demod_Chip/DSB_Demod/PreDe...			
lab_dsbdm/DSB_Demod_Chip/DSB_Demod/Single...			
NCO Phase Increment Registers (sin Inversion@MSB...			
lab_dsbdm/DSB_Demod_Chip/DSB_Demod/NCO (...)			



Description
Shows the resource usage of the lab_dsbdm model.

Blocks	Type	LUT4s	Mults	Memory bits
lab_dsbdm		9237	108	16448
DSB_Demod_Chip		9237	108	16448
DSB_Demod	SYNTH	9221	108	16448
● NCO	NCO	1153	8	16448
● PreDetectionHPF	FIRS	2035	32	0
● PreDetectionLPF	FIRS	2035	32	0
● Scale	SCALE	459	0	0
● Scale1	SCALE	489	0	0
● Scale2	SCALE	459	0	0
● Scale3	SCALE	399	0	0
● SingleRateFIR	FIRS	2035	32	0
Sync_Mix	SYNTH	125	4	0
● DualMem_0	DUALMEM	28	0	0
● DualMem_1	DUALMEM	28	0	0
● Mult_0	MULT	0	2	0
● Mult_1	MULT	0	2	0
● Id_ChannelIn_dV_s_to_...	DELAY	4	0	0
● Id_ChannelIn_dIn_0_to_...	DELAY	16	0	0
● Id_ChannelIn_dIn_1_to_...	DELAY	16	0	0
● Id_ChannelIn_dC_s_to_...	DUALMEM	24	0	0
● Id_ChannelIn_dC_s_to_...	COUNTER	2	0	0
● Id_ChannelIn_dC_s_to_...	REG	2	0	0
● Id_ChannelIn_dC_s_to_...	CONSTANT	0	0	0
● Id_ChannelIn_dC_s_to_...	REG	1	0	0
● Id_ChannelIn_dC_s_to_...	REG	1	0	0
● Id_ChannelIn_dC_s_to_...	LOGICAL	0	0	0
● Id_ChannelIn_dC_s_to_...	LOGICAL	1	0	0
● Id_ChannelIn_dC_s_to_...	LOGICAL	1	0	0
● Id_ChannelIn_dC_s_to_...	LOGICAL	1	0	0
● Id_ChannelIn_dC_s_to_...	LOGICAL	1	0	0
● busReadSelector	SELECTOR	32	0	0
● busReadSelector	SELECTOR	16	0	0

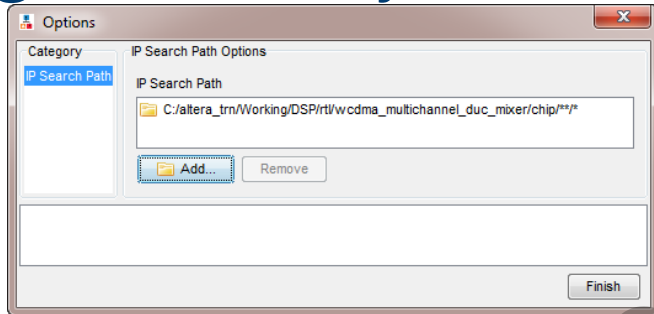
Design Verification

The image displays the Quartus II software interface. The top window, titled 'Test_Model', shows the 'DSP Builder' menu with 'Run ModelSim' highlighted. A callout box points to this menu item with the text: 'Run ModelSim block loads the design into the ModelSim simulator'. Below this, the ModelSim 'wave - default' window is shown in 'RTL Simulation' mode. The waveform viewer displays several signals over time, with a time scale of 200 ns. The signals include a clock signal, a system reset signal, and various data buses. The simulation is connected to a block diagram of the 'dut' (Design Under Test), which includes a 'Convert' block and a 'dout' block. The 'dout' block has multiple outputs: 'sfix17_En16 (2) data out', 'boolean valid out', and 'uint8 channel out (2)'. The 'Convert' block has inputs for 'double' and 'noise', and an output for 'sfix17_En16'. The 'dout' block has inputs for 'd', 'v', 'c', and 'b', and an output for 'cout'. The 'Expand scalar' block is also visible, showing a 'vector' output.

Run ModelSim block loads the design into the ModelSim simulator

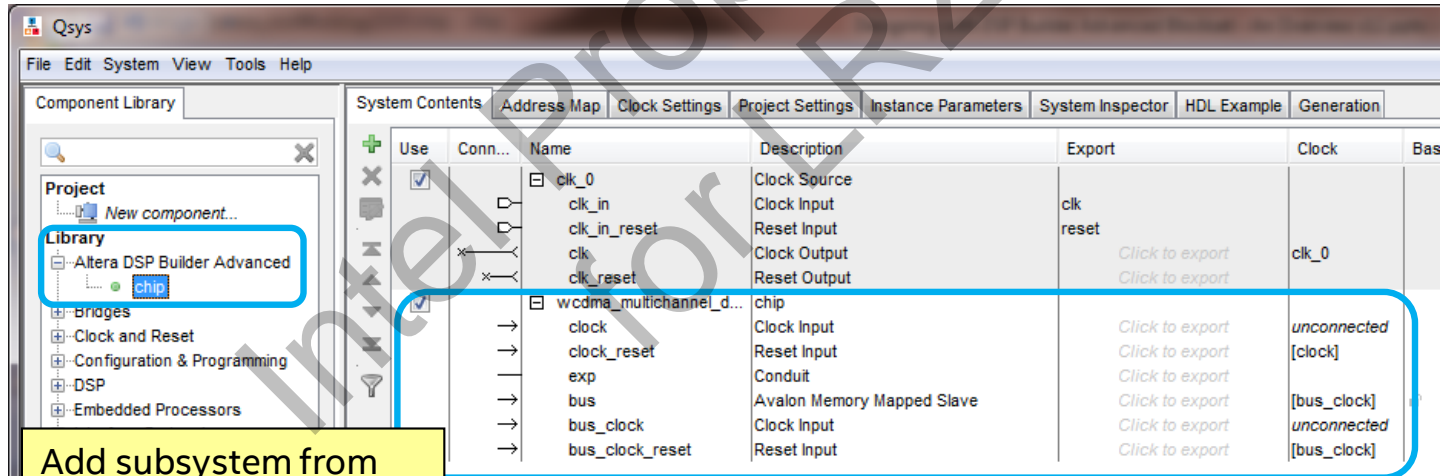
RTL Simulation

Design Flow – System Integration

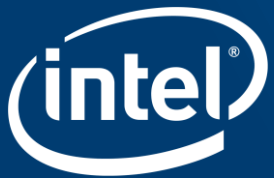


Add <subsystem>_hw.tcl directory to Qsys IP Search Path

Qsys-> Tools -> Options -> IP Search Path



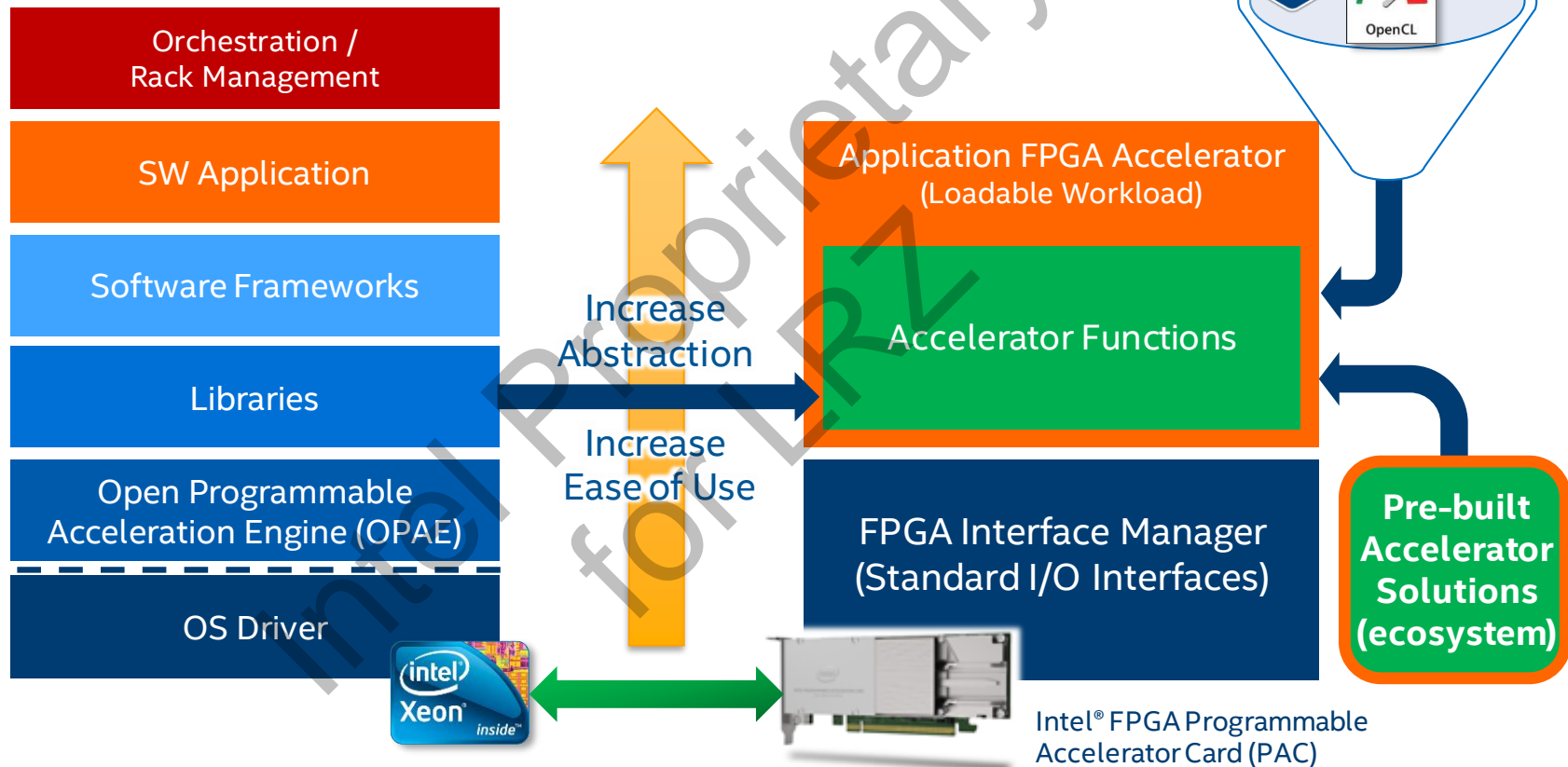
Add subsystem from the Component pick list



ACCELERATION STACK FOR XEON WITH FPGA

Intel Proprietary
for RZ

Using FPGAs Just Got Easier



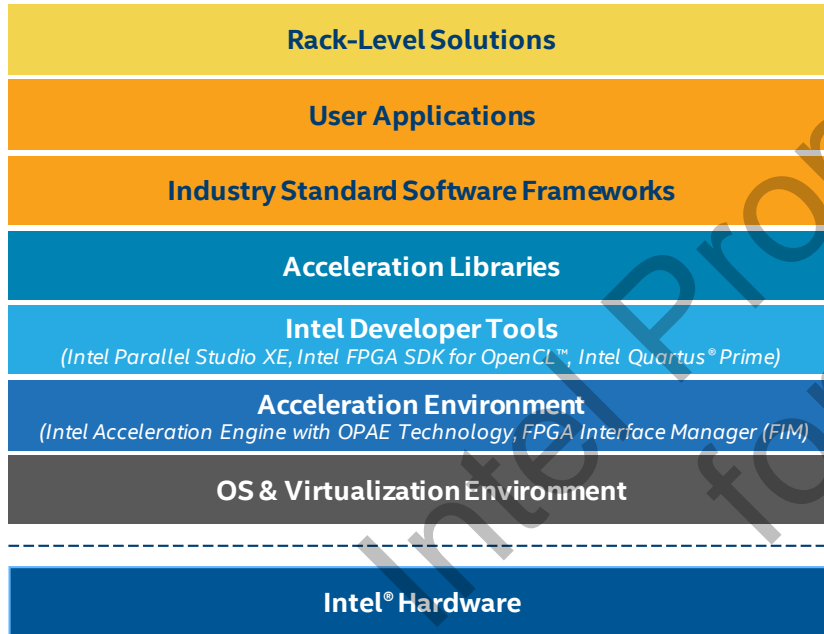
* Other names and brands may be claimed as the property of others.

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



ACCELERATION STACK FOR INTEL® XEON® CPU WITH FPGAS

COMPREHENSIVE ARCHITECTURE FOR DATA CENTER DEPLOYMENTS



Faster Time to Revenue

- Fully validated Intel® board
- Standardized frameworks and high-level compilers
- Partner-developed workload accelerators

Simplified Management

- Supported in VMware vSphere* 6.7 Update 1*
- Rack management and orchestration framework integration

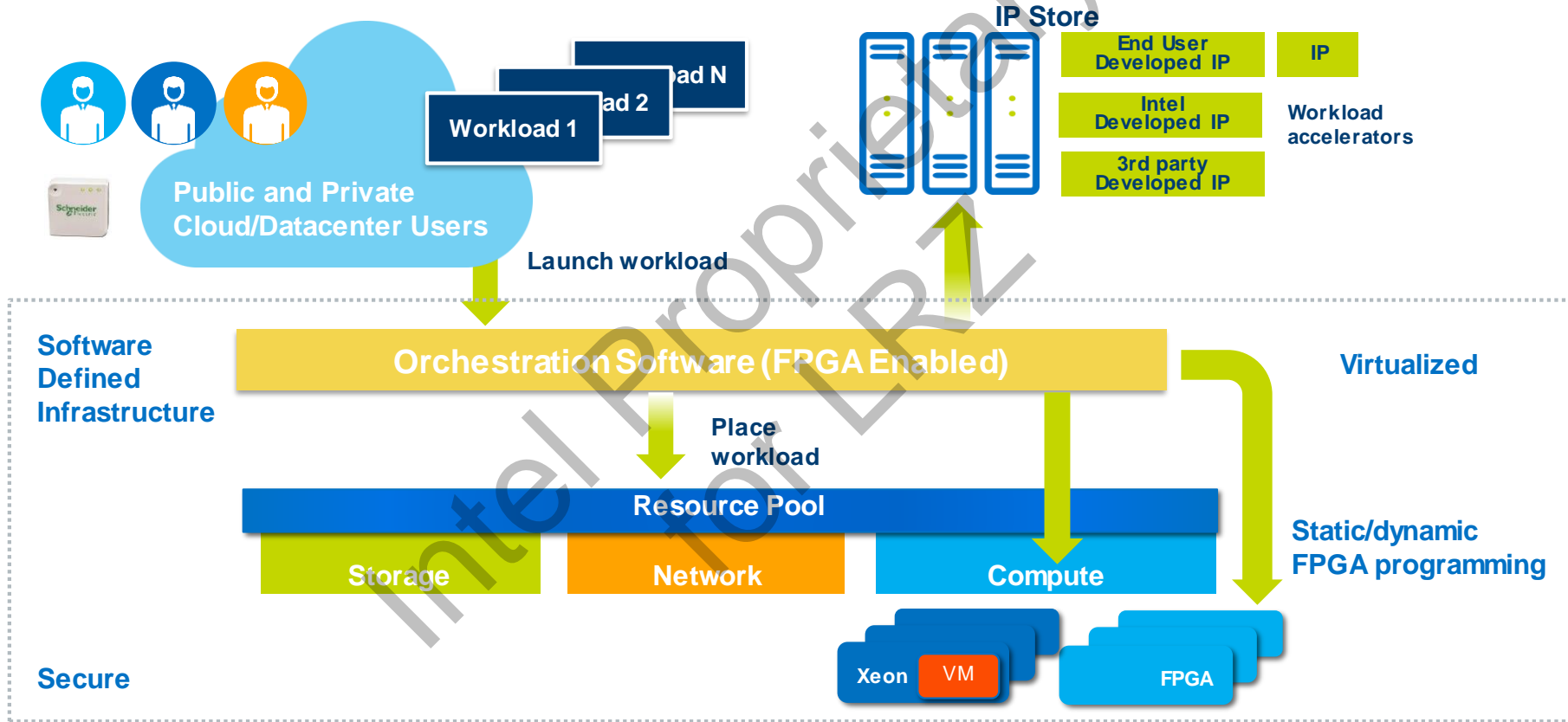
Broad Ecosystem Support

- Upstreaming FPGA drivers to Linux* kernel
- Qualified by industry-leading server OEMs
- Partnering with IP partners, OSVs, ISVs, SIs, and VARs

* Demonstrated at VMWorld Las Vegas - August 28-30, 2018

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

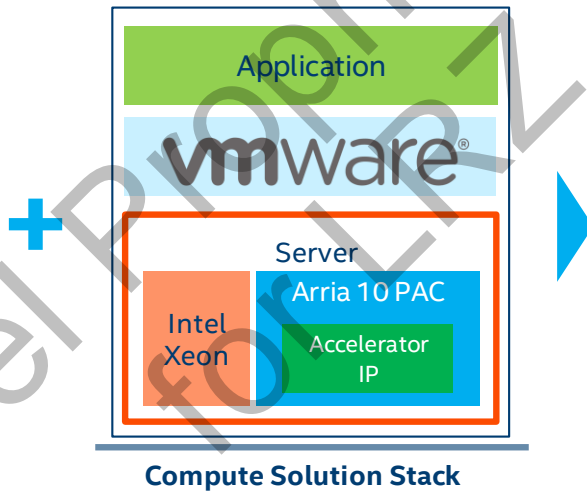
Acceleration Stack Provides FPGA Orchestration in Cloud/Data Center



Server Virtualization for the Acceleration Stack with VMware



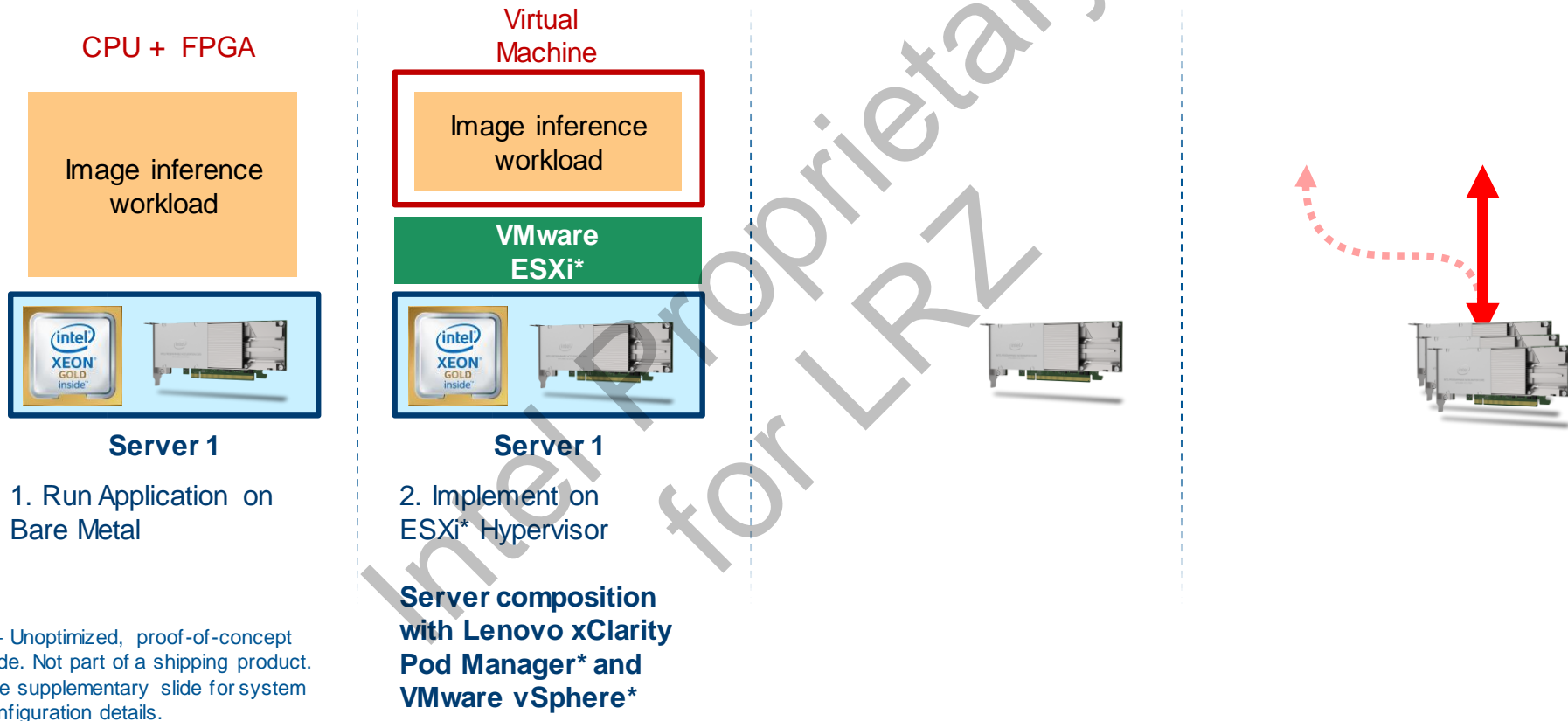
Intel Arria 10 Programmable Acceleration Card with Acceleration Stack



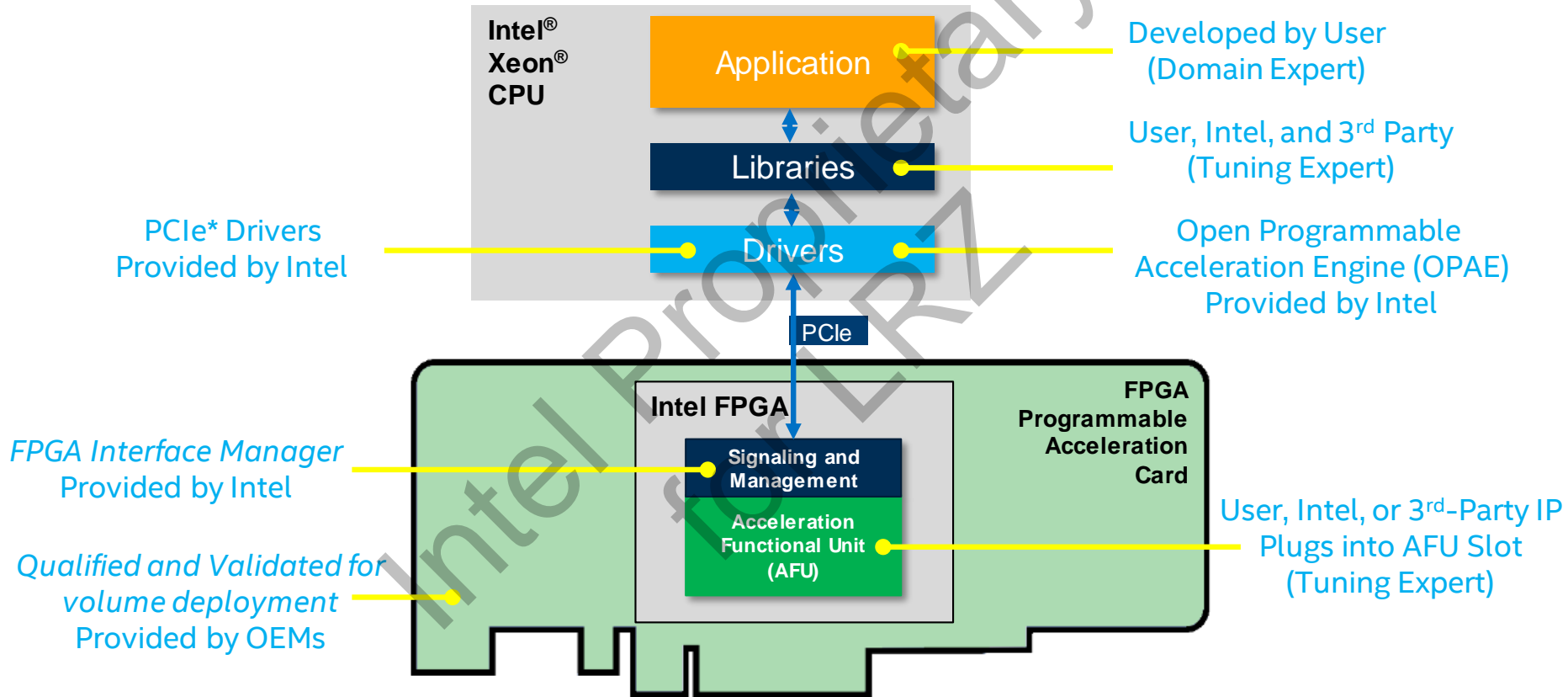
Out-of-the-box support from VMWare for Intel Arria 10 PAC and Acceleration Stack in **upcoming vSphere 6.7 U1**

Server virtualization enables customers to deploy FPGA workload acceleration with **lower total cost of ownership**

Migrating FPGA-Accelerated Workload with vMotion*



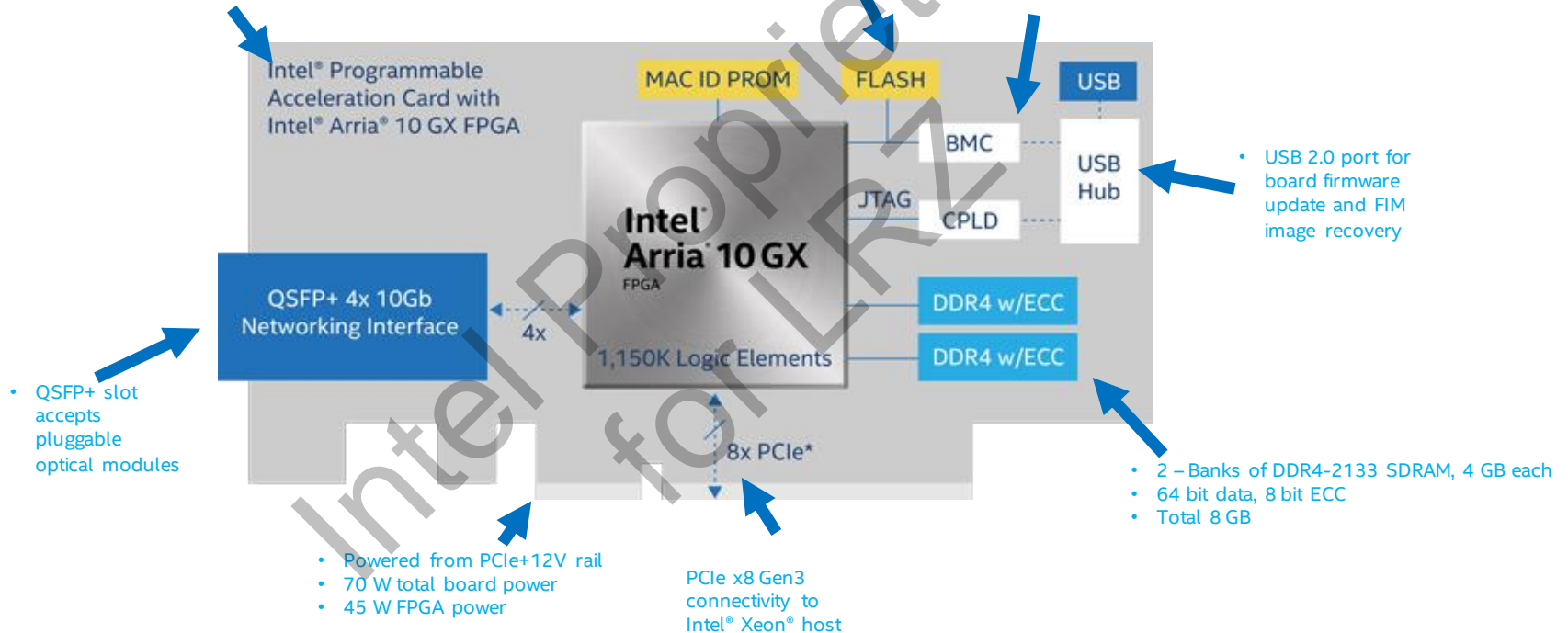
Components of Acceleration Stack: Overview



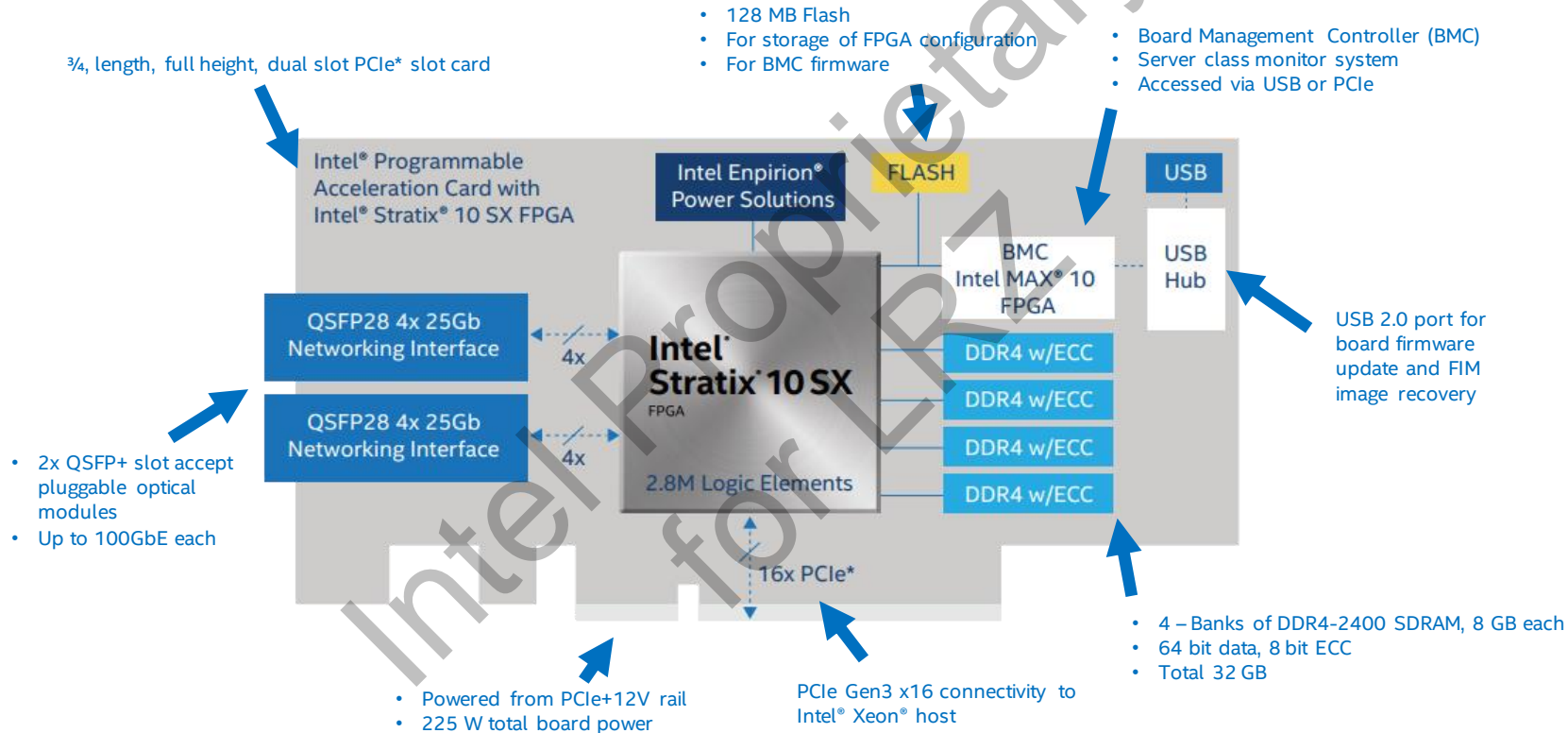
PAC with Intel® Arria® 10 FPGA

- Low-profile (half-length, half height) PCIe* slot card
- 168 mm × 56 mm
- Maximum component height: 14.47 mm
- PCIe × 16 mechanical

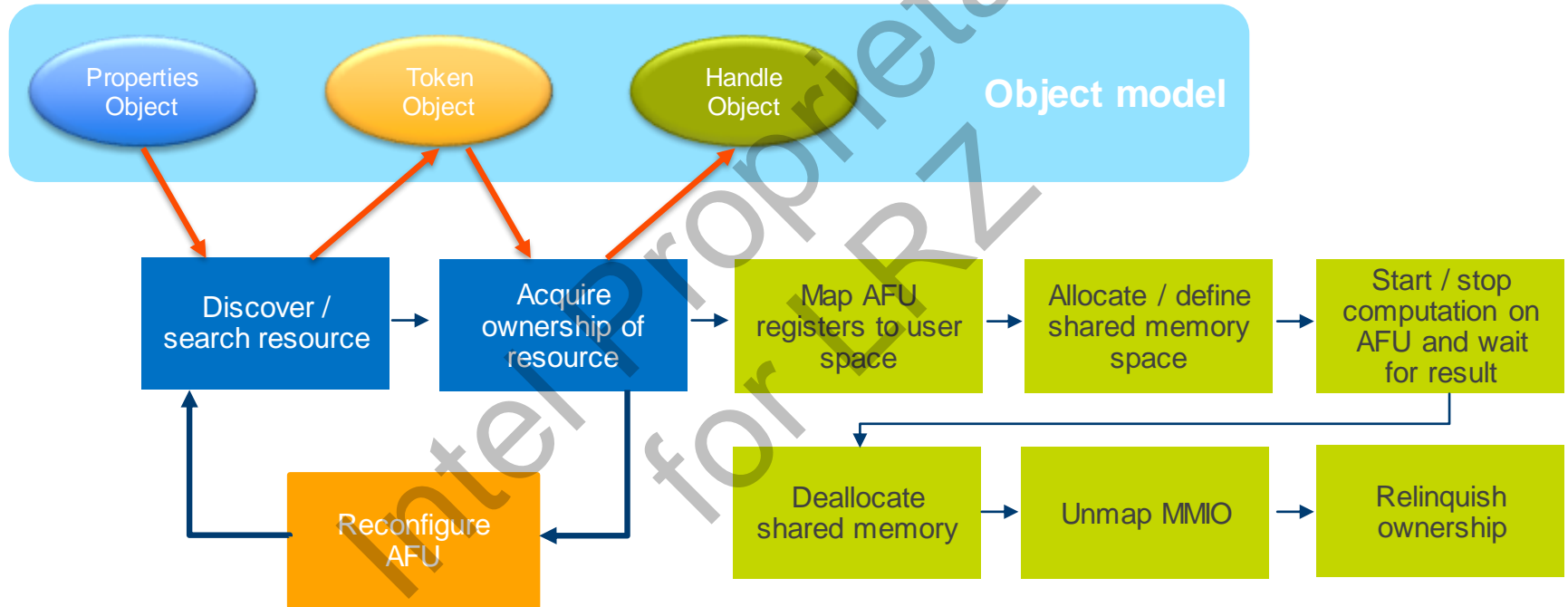
- 128 MB Flash
- For storage of FPGA configuration
- Board Management Controller (BMC)
- Server class monitor system
- Accessed via USB or PCIe



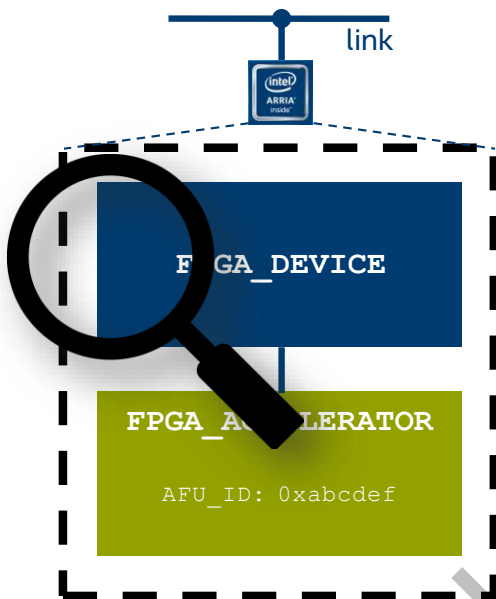
PAC with Intel® Stratix® 10 FPGA



Nearly Transparent Software Application Use Model



Enumeration and Discovery



fpga_properties prop

objtype: FPGA_ACCELERATOR
guid: 0xabcdef

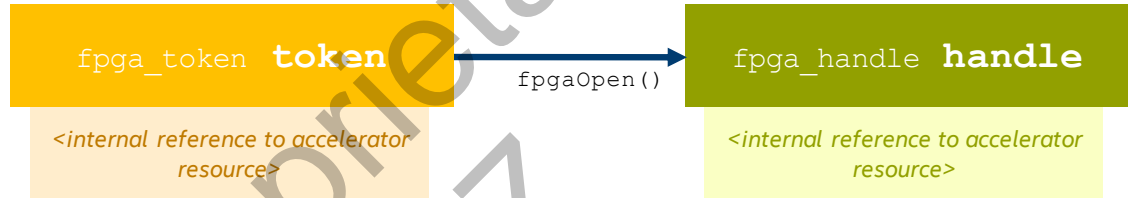
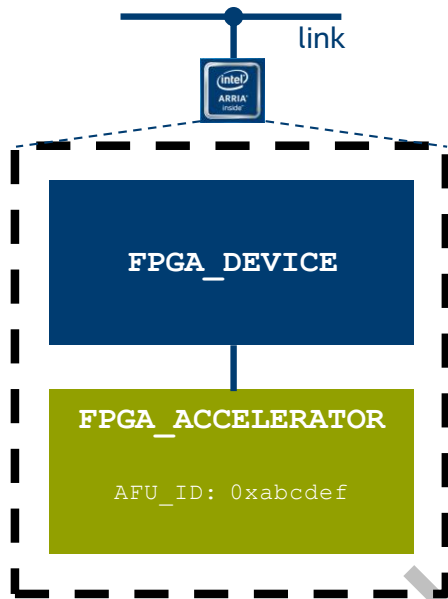
fpgaEnumerate()

fpga_token token

<internal reference to accelerator
resource>

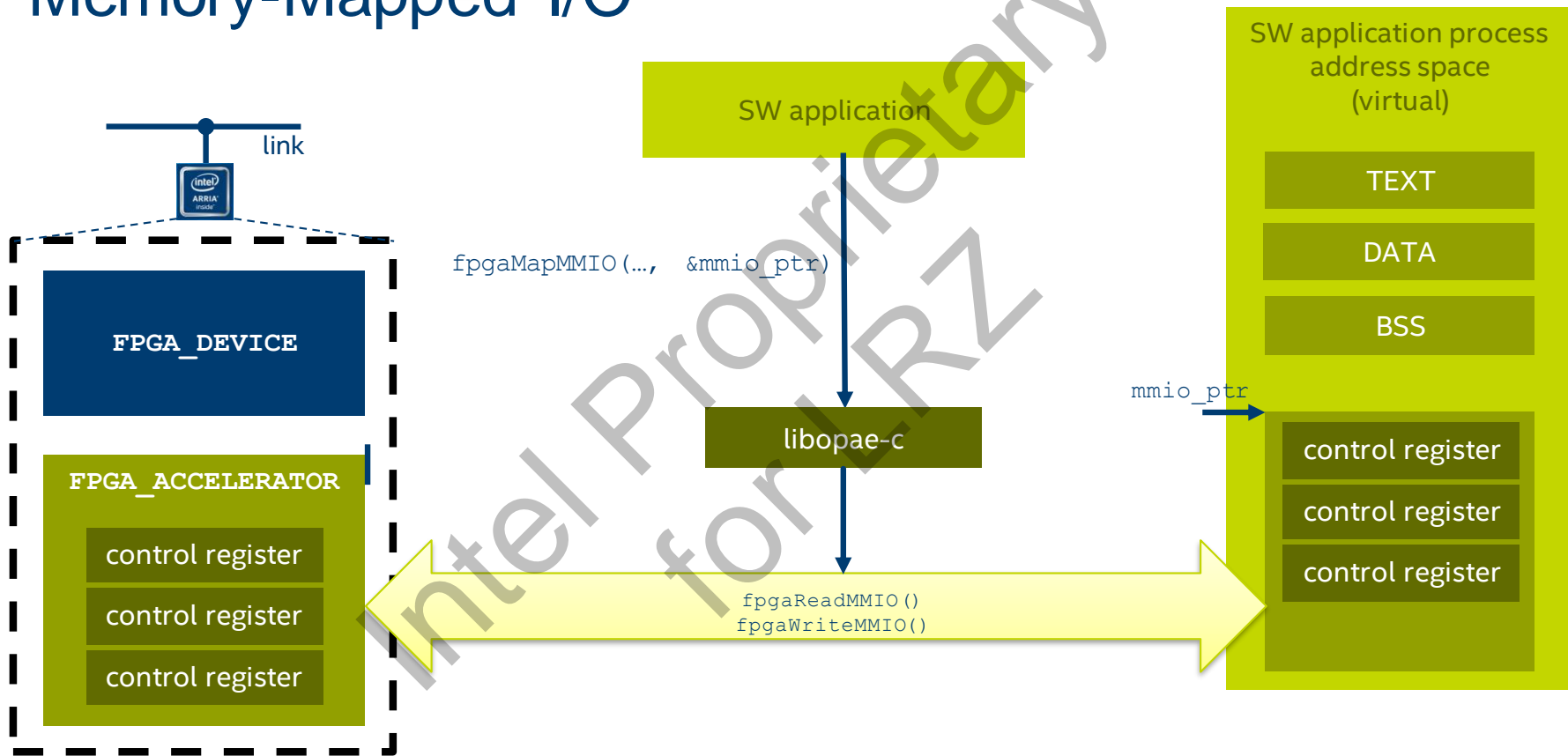
```
fpga_properties prop;  
fpga_token token;  
fpga_guid myguid; /* 0xabcdef */  
  
fpgaGetProperties (NULL, &prop);  
  
fpgaPropertiesSetObjectType(prop, FPGA_ACCELERATOR);  
fpgaPropertiesSetGUID(prop, myguid);  
  
fpgaEnumerate(&prop, 1, &token, 1, &n);  
  
fpgaDestroyProperties (&prop);
```

Acquire and Release Accelerator Resource

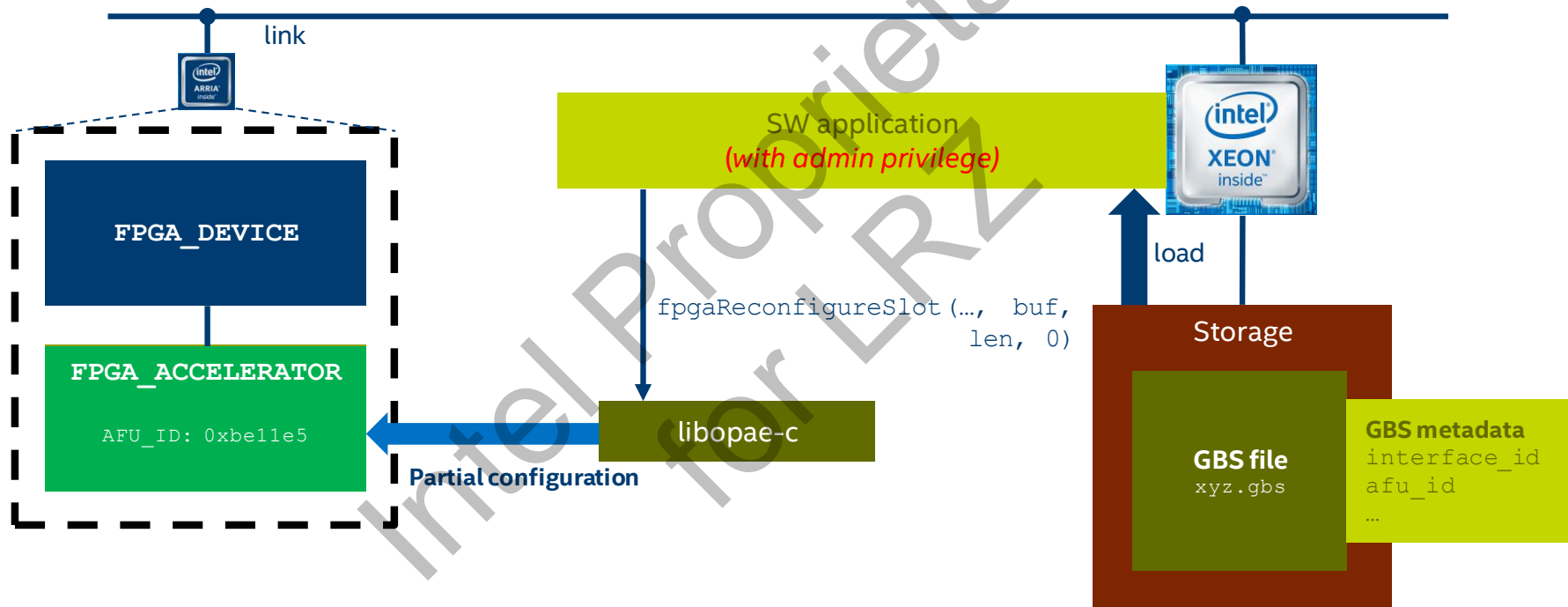


```
fpga_token token;  
// ... enumeration ...  
fpga_handle handle;  
  
fpgaOpen(token, &handle, 0);  
.  
.  
.  
fpgaClose(handle);
```

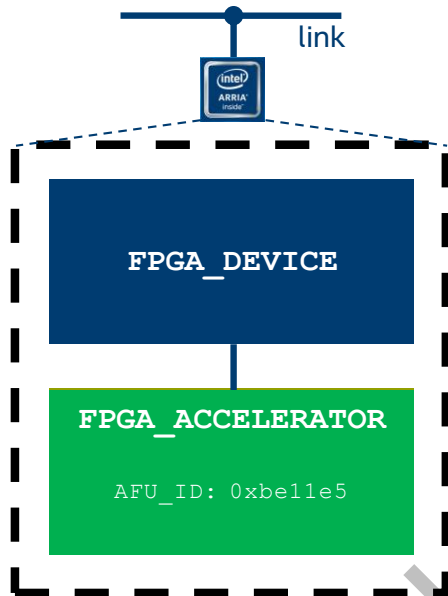
Memory-Mapped I/O



Management and Reconfiguration



Management and Reconfiguration



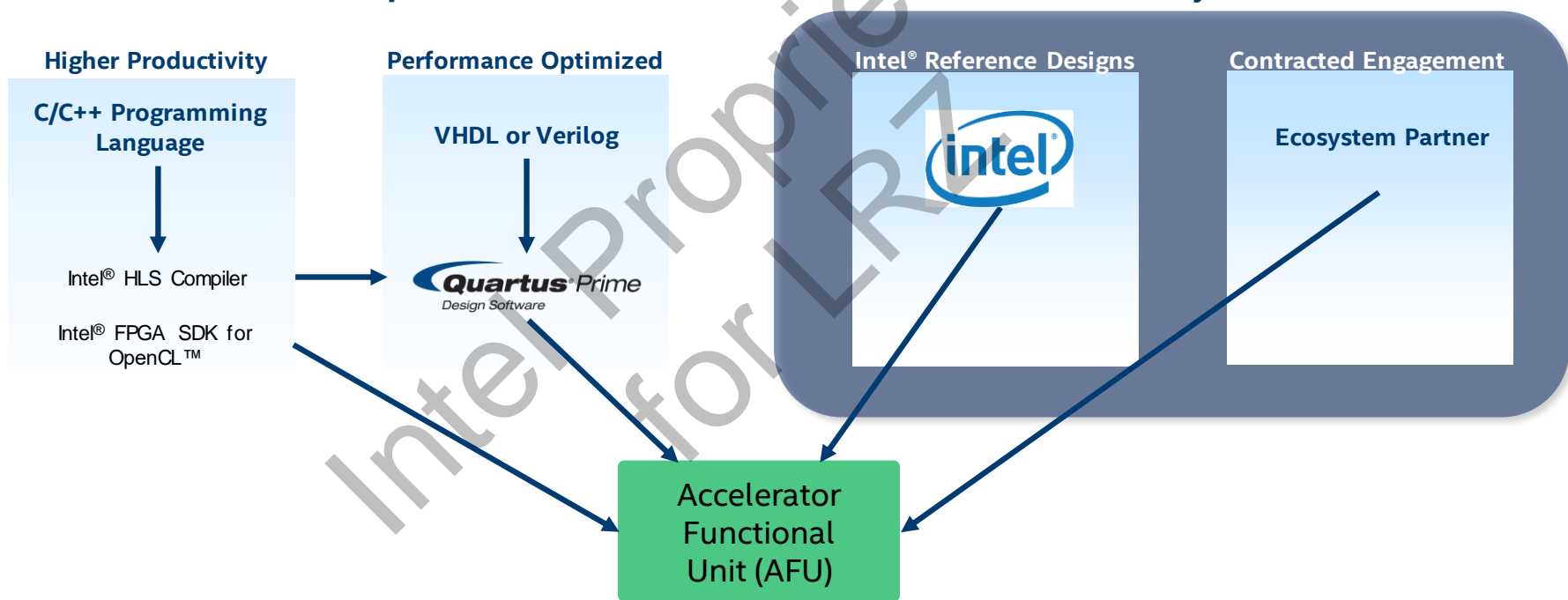
```
fpga_handle handle;          /* handle to device */
FILE *gbs_file;
void *gbs_ptr;
size_t gbs_size;

/* Read bitstream file */
gbs_ptr = malloc(gbs_size);
fread(gbs_ptr, 1, gbs_len, gbs_file);

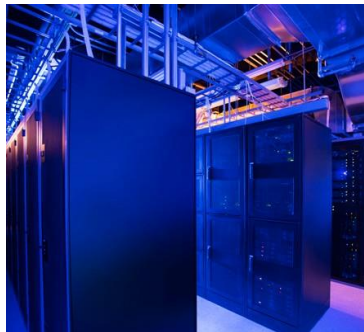
/* Program GBS to FPGA */
fpgaReconfigureSlot(handle, 0, gbs_ptr, gbs_size, 0);
/* ... */
```

Self-Developed

Externally-Sourced



Growing the Xeon+FPGA Ecosystem



IP AND SOLUTIONS

Portfolio of Accelerator Solutions developed by Intel and third-party technologists to expedite application development and deployment



DEVELOPER COMMUNITY

Enabling software developers access via:

- Intel Builder programs
- AI Academy
- Intel Developer Zone (IDZ)
- Rocketboards.org



UNIVERSITIES

Reaching over 200,000 students per year with FPGA publications, workshops and hands-on research labs

Committed to Open Source vision



ISV PARTNERS

Expanding the reach for system vendors with platforms and ready-to-use application workloads.

Growing List of Accelerator Solution Partners

Easing Development and Data Center Deployment of Intel FPGAs For Workload Optimization



- Data Analytics
- Finance
- Genomics
- Media Transcoding
- Cyber Security
- AI

Accelize ALGO-LOGIC ADAPTIVE MICRO-WARE bigstream
b.com
CAST CTACCEL falcon COMPUTING iAbra
Levyx MEGH COMPUTING myrtle.ai NAGASE
napatech
ENIAC SOC enyx swarm64

INTEL PAC TOP SOLUTIONS FOR DATA CENTER ACCELERATION

						
Cassandra	PostgreSQL	Genomics GATK	JPEG2Lepton JPEG2Webp	Big Data Streaming Analytics	Financial Black Scholes	Network Security/ Monitoring
96% latency reduction	1/2 TCO	2.5X performance	3-4X performance	5X performance	8X performance	3x performance

CASE STUDY: 8X SPARK SQL / KAFKA PERFORMANCE INCREASE



Customer Application: Big Data Applications running on Spark/Kafka Platforms

Current solution: Run Spark/SQL on a cluster of CPUs

Challenge: For many applications in the FinServ/Genomics/Intelligence Agencies/etc. Spark performance does not meet customers SLA requirements, especially for delay sensitive streaming workloads

**Solution
Value
Proposition**

Performance - Accelerate Spark SQL/Kafka by 8x
Ease of Use – Zero Code Change
Scalability - Hardware Agnostic
Lower TCO

CASE STUDY: 5X RISK ANALYTICS PERFORMANCE INCREASE



Customer Application: Risk Management acceleration framework (financial back-testing)

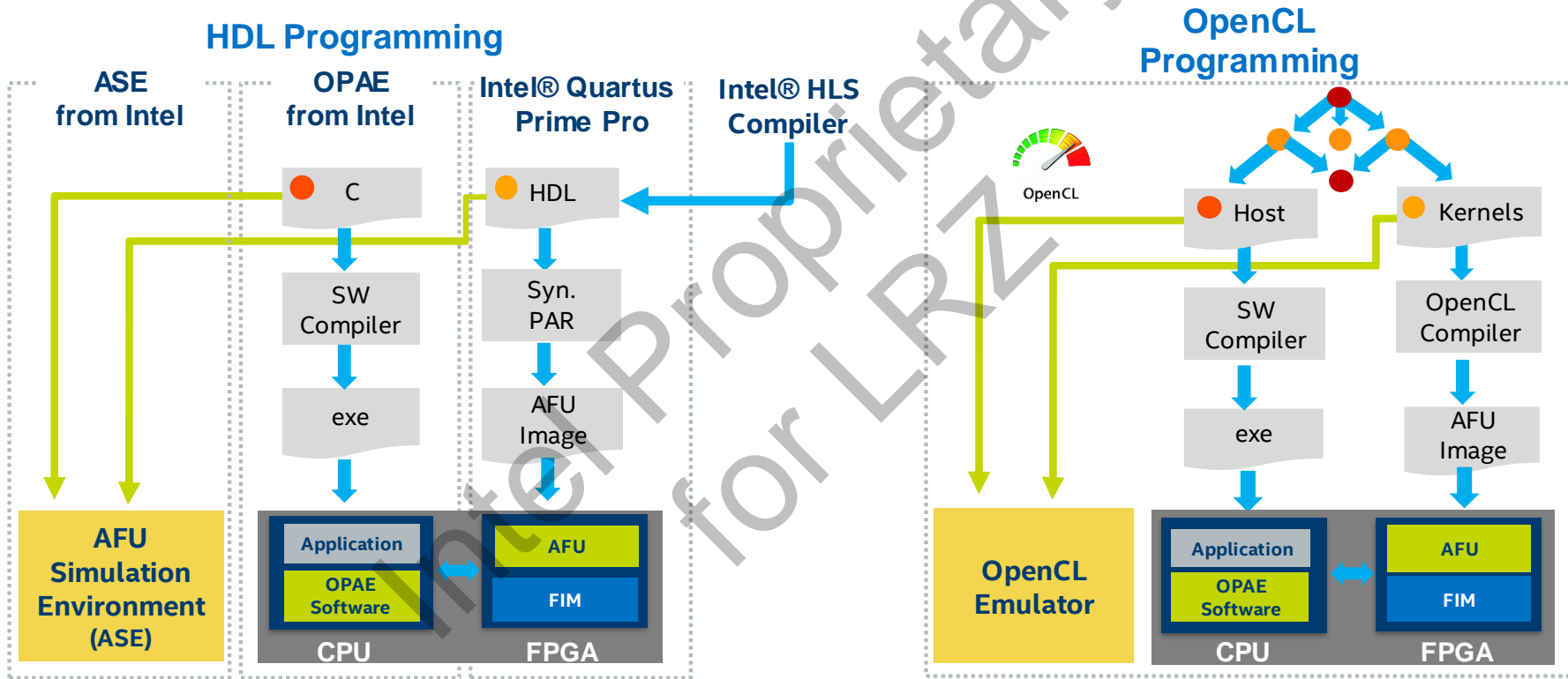
Current solution: Deploy a cluster of CPUs or GPUs with complex data access

Challenge: Traditional risk management methods are compute intensive, time consuming applications - > 10+ hours for financial back-testing

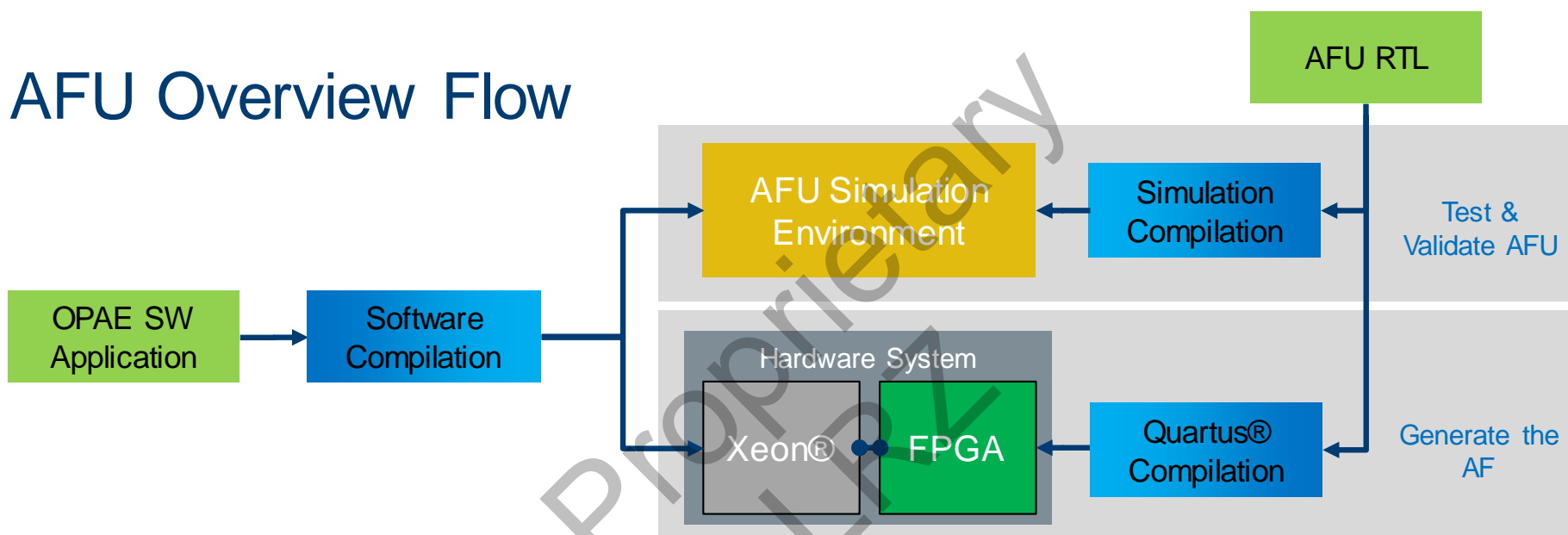
**Solution
Value
Proposition**

>5x Performance Improvement
Perform Risk and Pricing Calculations Simultaneously
Abstraction - Integrated Solution
with Apache Spark, SSD Access and FPGA Implementation

Leverage FPGA Developers and Build Your Own



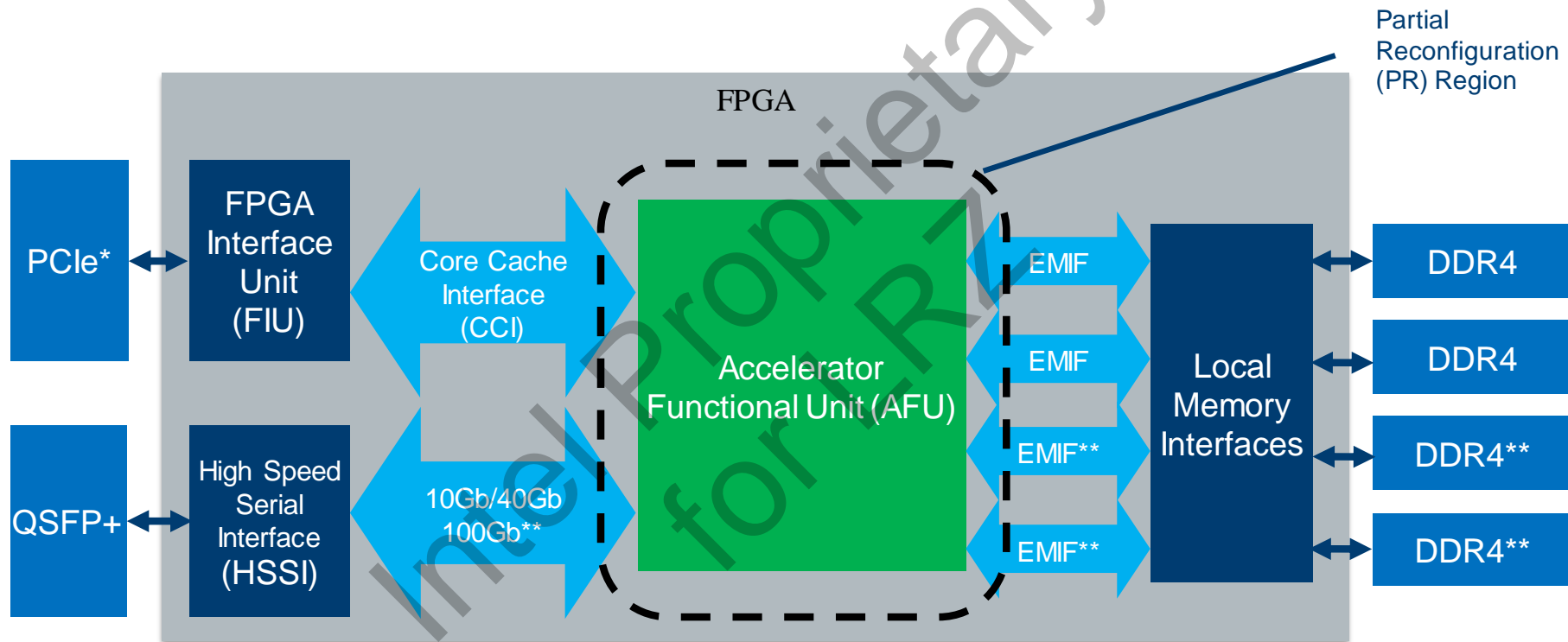
AFU Overview Flow



AF Simulation Environment (ASE) enables seamless portability to real HW

- Allows fast verification of OPAE software together with AF 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

FPGA Components of Acceleration Stack



* Could be other interfaces in the future (e.g. UPI)

** Stratix 10 PAC Card

AFU Development Flow Using OPAE SDK

AFU requests the `ccip_std_afu` top level interface classes

- `$OPAE_PLATFORM_ROOT/hw/samples/hello_afu/hw/rtl/hello_afu.json`

AFU RTL files implementing accelerated function

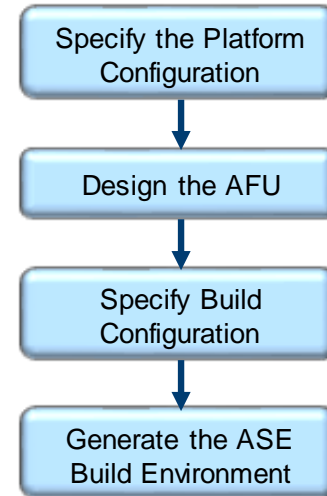
- `$OPAE_PLATFORM_ROOT/hw/samples/hello_afu/hw/rtl/afu.sv`

List all source files and platform configuration file

- `$OPAE_PLATFORM_ROOT/hw/samples/hello_afu/hw/rtl/filelist.txt`

In terminal window, enter these commands:

- `cd $OPAE_PLATFORM_ROOT/hw/samples/hello_afu`
- `afu_sim_setup --source hw/rtl/filelist.txt build_sim`



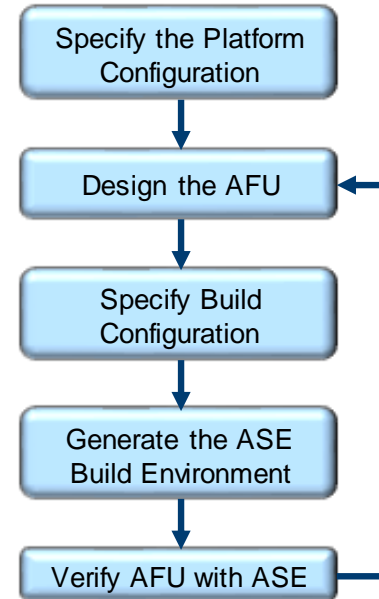
AFU Development Flow Using OPAE SDK

Compile AFU and platform simulation models and start simulation server process

- `cd build_sim`
- `make`
- `make sim`

In 2nd terminal window compile the host application and start the client process

- `Export ASE_WORKDIR= $OPAE_PLATFORM_ROOT/hw/samples/hello_afu/build_sim/work`
- `cd $OPAE_PLATFORM_ROOT/hw/samples/hello_afu/sw`
- `make clean`
- `make USE_ASE=1`
- `./hello_afu`



AFU Simulation Environment (ASE)

Hardware software co-simulation environment for the Intel Xeon FPGA development

Uses simulator Direct Programming Interface (DPI) for HW/SW connectivity

- Not cycle accurate (used for functional correctness)
- Converts SW API to CCI transactions

Provides transactional model for the Core Cache Interface (CCI-P) protocol and memory model for the FPGA-attached local memory

Validates compliance to

- CCI-P protocol specification
- Avalon® Memory Mapped (Avalon-MM) Interface Specification
- Open Programmable Acceleration Engine

Simulation Complete

```
[APP]
[APP] Issuing Soft Reset...
[APP] MMIO Read : tid = 0x002, offset = 0x0
[APP] MMIO Read Resp : tid = 0x002, data = 1000010000000000
AFU DFH REG = 1000010000000000
[APP] MMIO Read : tid = 0x003, offset = 0x8
[APP] MMIO Read Resp : tid = 0x003, data = 9722d43375b61c66
AFU ID LO = 9722d43375b61c66
[APP] MMIO Read : tid = 0x004, offset = 0x10
[APP] MMIO Read Resp : tid = 0x004, data = 850adcc26ceb4b22
AFU ID HI = 850adcc26ceb4b22
[APP] MMIO Read : tid = 0x005, offset = 0x18
[APP] MMIO Read Resp : tid = 0x005, data = 0
AFU NEXT = 00000000
[APP] MMIO Read : tid = 0x006, offset = 0x20
[APP] MMIO Read Resp : tid = 0x006, data = 0
AFU RESERVED = 00000000
[APP] MMIO Read : tid = 0x007, offset = 0x80
[APP] MMIO Read Resp : tid = 0x007, data = 0
Reading Scratch Register (Byte Offset=00000080) = 00000000
MMIO Write to Scratch Register (Byte Offset=00000080) = 123456789abcdef
[APP] MMIO Write : tid = 0x008, offset = 0x80, data = 0x123456789abcdef
[APP] MMIO Read : tid = 0x009, offset = 0x80
[APP] MMIO Read Resp : tid = 0x009, data = 123456789abcdef
Reading Scratch Register (Byte Offset=00000080) = 123456789abcdef
Setting Scratch Register (Byte Offset=00000080) = 00000000
[APP] MMIO Write : tid = 0x00a, offset = 0x80, data = 0x0
[APP] MMIO Read : tid = 0x00b, offset = 0x80
[APP] MMIO Read Resp : tid = 0x00b, data = 0
Reading Scratch Register (Byte Offset=00000080) = 00000000
Done Running Test
[APP] Deinitializing simulation session
[APP] Closing Watcher threads
[APP] Deallocating UMAS
[APP] Deallocating memory /umas.187070359034322 ...
[APP] SUCCESS
[APP] Deallocating MMIO map
[APP] Deallocating memory /mmio.187070359034322 ...
[APP] SUCCESS
[APP] Deallocate all buffers ...
[APP] Took 583,231,696 nsec
[APP] Session ended

# [SIM] 1 ADDED /umas.187070359034322
# [SIM] Request to deallocate "/umas.187070359034322" ...
# [SIM] 1 REMOVED /umas.187070359034322
# [SIM] Request to deallocate "/mmio.187070359034322" ...
# [SIM] 0 REMOVED /mmio.187070359034322
# [SIM] ASE recognized a SW simkill (see ase.cfg)... Simulator will EXIT
# [SIM] SIM-C : Exiting event socket server/tmp/ase_event_server_187070359034322...
# [SIM] Closing message queue and unlinking...
# [SIM] Untinking Shared memory regions....
# [SIM] Session code file removed
# [SIM] Removing message queues and buffer handles ...
# [SIM] Cleaning session files...
# [SIM] Simulation generated log files
# [SIM] Transactions file | $ASE_WORKDIR/ccip_transactions.tsv
# [SIM] Workspaces info | $ASE_WORKDIR/workspace_info.log
# [SIM] ASE seed | $ASE_WORKDIR/ase_seed.txt
# [SIM] Tests run ==> 1
# [SIM] Sending kill command...
# [SIM] Simulation kill command received...
#
# Transaction count | VA VL0 VH0 VH1 | MCL-1 MCL-2 MCL-4
# -----|-----|-----|-----|-----|-----|-----|
# MMIOwrReq 2 |
# MMIOrdReq 10 |
# MMIOrdRsp 10 |
# IntrReq 0 |
# IntrResp 0 |
# RdReq 0 | 0 0 0 0 | 0 0 0
# RdResp 0 | 0 0 0 0 | 0 0 0
# WrReq 0 | 0 0 0 0 | 0 0 0
# WrResp 0 | 0 0 0 0 | 0 0 0
# WrFence 0 | 0 0 0 0 | 0 0 0
# WrFenRsp 0 | 0 0 0 0 | 0 0 0
#
# ** Note: $finish : /home/student/fpga_trn/AccelStack_Workshop/hello_afu/build_sim/rtl/cc
4)
# Time: 21620047500 ps Iteration: 2 Instance: /ase top/ccip_emulator
# End time: 12:34:40 on Aug 21,2018, Elapsed time: 0:28:57
# Errors: 0, Warnings: 3
/home/student/fpga_trn/AccelStack_Workshop/hello_afu/build_sim
```

AFU Simulator Window (server)

Application SW Window (client)

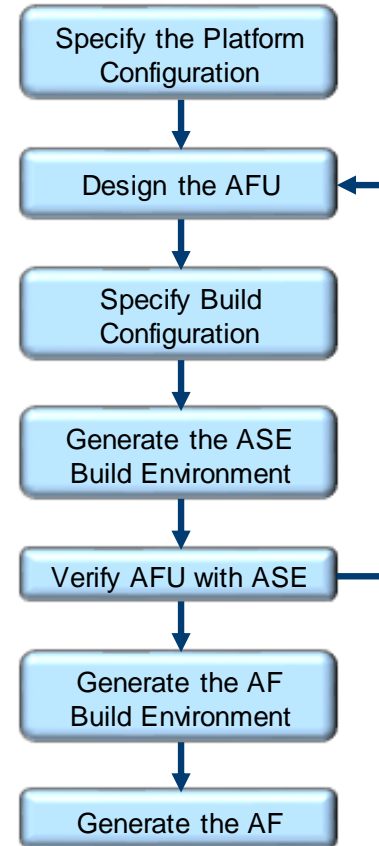
AFU Development Flow Using OPAE SDK

Generate the AF build environment:

- `cd $OPAE_PLATFORM_ROOT/hw/samples/hello_afu`
- `afu_synth_setup --source hw/rtl/filelist.txt build_synth`

Generate the AF

- `cd build_synth`
- `$OPAE_PLATFORM_ROOT/bin/run.sh`



Using the Quartus GUI

Compiling the AFU uses a command line-driven PR compilation flow

- Builds PR region AF as a .gbs file to be loaded into OPAAE hardware platform

Can use the Quartus GUI for the following types of work:

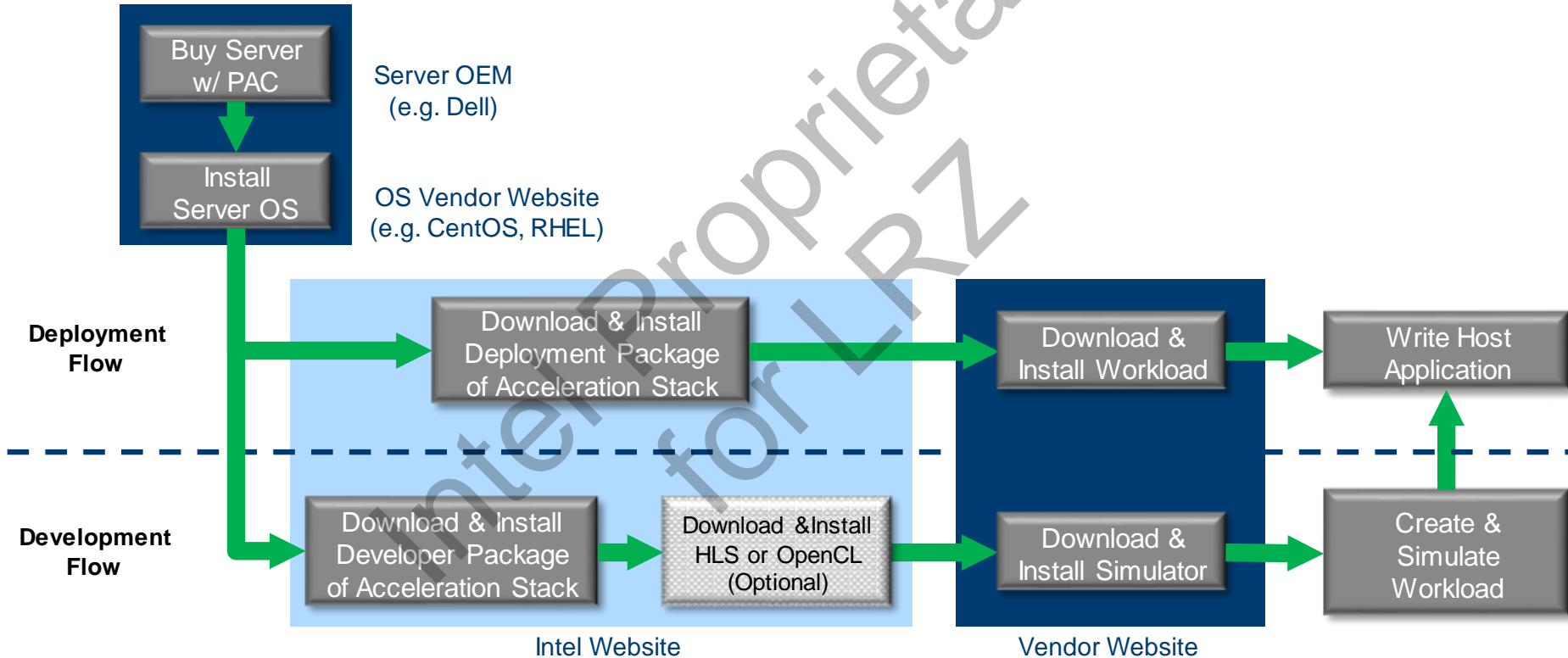
- Viewing compilation reports
- Interactive Timing Analysis
- Adding SignalTap instances and nodes

Acceleration Stack Demo

Lab 3

Intel Proprietary
for IRZ

Getting Started with Acceleration



Getting Qualified Hardware is Step 1

DELLEMC

FUJITSU

Hewlett-Packard
Enterprise

QCTTM

inspur

Now:

Dell PowerEdge*
R640, R740,
R740xd, R840,
R940xa

Now:

PRIMERGY*
RX2540 M4

Available soon:

HPE ProLiant*
DL360, DL380

And more coming

Programmable Acceleration Cards (PAC)

Intel® Arria® 10 Accelerator Card



Broadest Deployment at Lowest Power

40G, PCIe* Gen3 x8

½ length, ½ height, single-slot PCIe card

Lowest power 66W TDP

Intel Stratix® 10 Accelerator Card



Highest Performance and Throughput

2x 100G, PCIe Gen3 x16

¾ length, full height, dual-slot PCIe card

Up to 225 W maximum

INTEL® FPGA ACCELERATION HUB

A new collection of software, firmware, and tools that allows all developers to leverage the power of Intel® FPGAs.

Intel® portal for all things related to FPGA acceleration

- **Acceleration Stack** for Intel® Xeon® with FPGAs
- **FPGA Acceleration Platforms**
- **Acceleration Solutions & Ecosystem**
- **Knowledge Center**
- **FPGA as a Service**
- **01.org***

* 01.org is an open source community site



NEXT STEPS

Intel Proprietary
for LRZ

Follow-On Courses

<https://www.intel.com/content/www/us/en/programmable/support/training/overview.html>

[Introduction to Cloud Computing](#)

[Introduction to High Performance Computing \(HPC\)](#)

[Introduction to Apache™ Hadoop](#)

[Introduction to Apache Spark™](#)

[Introduction to Kafka™](#)

[Introduction to Intel® FPGAs for Software Developers](#)

[Introduction to the Acceleration Stack for Intel® Xeon® CPU with FPGA](#)

[Application Development on the Acceleration Stack for Intel® Xeon® CPU with FPGAs](#)

[Building RTL Workloads for the Acceleration Stack for Intel® Xeon® CPU with FPGAs](#)

[OpenCL™ Development with the Acceleration Stack for Intel® Xeon® CPU with FPGA](#)

[Intel FPGA OpenCL Trainings](#) and [HLS Trainings](#)

Teaching Resources

University-focused content & curriculum

- Semester-long laboratory exercises for hands-on learning with solutions
- Tutorials and online workshops for self-study on key use cases
- Free library of IP common for student projects
- Example designs and sample projects

Easy-to-use, powerful software tools

- Quartus Prime CAD Environment
- ModelSim
- Intel FPGA Monitor Program for assembly & C development
- Intel® SDK for OpenCL™ Applications
- Intel OpenVINO™ toolkit (Visual Inference & Neural Network Optimization)

Teaching Resources (cont.)

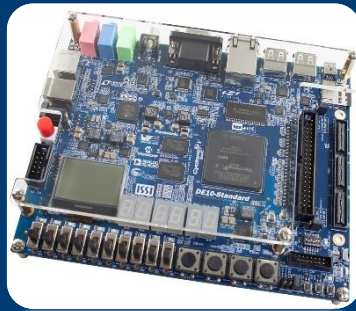
Hardware designed for education

- 4 different FPGA kits with a variety of peripherals to match project needs
- Compact designs with robust shielding to provide longevity
- Reduced academic prices (range: \$55-\$275)
- Donations available in some circumstances

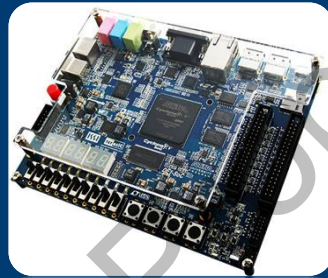
Support

- Total access to all developer resources
 - Documentation
 - Design examples
 - Support forum
 - Virtual or on-demand trainings

DE-Series Development Boards



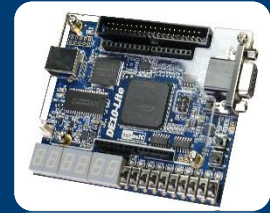
DE10-Standard
Cyclone V FPGA + SoC
\$259



DE1-SOC
Cyclone V FPGA + SoC
\$175



DE10-Nano
Cyclone V FPGA + SoC
\$99



DE10-Lite
Max 10 FPGA
\$55

Visit our [website](#) for full specs on these boards
See the full catalog of Intel FPGA boards & kits at www.terasic.com

	Beginner FPGA Dev Kit	FPGA+SoC Academic Dev Kit		Full-Featured Academic Dev Kit
Dev Kit	INTEL DE10-LITE	INTEL DE10-NANO	INTEL DE1-SOC	INTEL DE10-STANDARD
Academic Price	\$55	\$99	\$175	\$259
FPGA	Max [®] 10	Cyclone [®] V	Cyclone [®] V	Cyclone [®] V
Logic Elements	50,000	110,000	85,000	110,000
ARM Cortex-A9 Dual-Core System-on-Chip (SoC)	✗	800 MHz	925 MHz	925 MHz
Memory	64 MB SDRAM	1 GB DDR3 SDRAM (HPS)	1 GB DDR3 SDRAM (HPS), 64 MB SDRAM (FPGA)	1 GB DDR3 SDRAM (HPS), 64 MB SDRAM (FPGA)
PLLs	4	9	9	9
GPIO Count	500	469	469	469
7 Segment Displays	6	✗	6	6
Switches	10	4	10	10
Buttons	2	2	4	4
LEDs	10	8	10	10
Clocks	(2x) 50 MHz	(3x) 50 MHz	(4x) 50 MHz	(4x) 50 MHz
GPIO Count	40-pin header	(2x) 40-pin header	(2x) 40-pin header	40-pin header
Video Out	VGA 12-bit DAC	HDMI	VGA 24-bit DAC	VGA 24-bit DAC
ADC Channels	✗	8	8 + programmable voltage range	8 + programmable voltage range
Video In	✗	✗	NTSC, PAL, Multi-format	NTSC, PAL, Multi-format
Audio In/Out	✗	✗	Line In/Out, Microphone In (24 bit Audio CODEC)	Line In/Out, Microphone In (24 bit Audio CODEC)
Ethernet	✗	Gigabit	10/100/1000 Ethernet (x1)	10/100/1000 Ethernet (x1)
USB OTG	✗	1x USB OTG	2x USB 2.0 (Type A)	2x USB 2.0 (Type A)
LCD	✗	✗	✗	128x64 backlit
Micro SD Card Support	✗	✓	✓	✓
Accelerometer	✓	✓	✓	✓
PS/2 Mouse/Keyboard Port	✗	✗	✓	✓
Infrared	✗	✗	✓	✓
HSMC Header	✗	✗	✗	✓
Arduino Header	✓	✓	✗	✗

Undergrad Lab Exercise Suites: Digital Logic

First digital hardware course in EE, CompEng or CS curriculum

Traditionally introduced sophomore year

Offered in VHDL or Verilog

Lab 1 - Switches, Lights, and Multiplexers

Lab 7 - Finite State Machines

Lab 2 - Numbers and Displays

Lab 8 - Memory Blocks

Lab 3 - Latches, Flip-flops, and Registers

Lab 9 - A Simple Processor

Lab 4 - Counters

Lab 10 - An Enhanced Processor

Lab 5 - Timers and Real-Time Clock

Lab 11 - Implementing Algorithms in Hardware

Lab 6 - Adders, Subtractors, and Multipliers

Lab 12 - Basic Digital Signal Processing

Undergrad Lab Exercise Suites: Comp Organization

Typically second hardware course in EE, CompEng or CS curriculum

Introduction to microprocessors & assembly language program

Use ARM processor (on SOC kits) or NIOS II soft processor

Intel FPGA Monitor Program for compiling & debugging assembly & C code

Lab 1 - Using an ARM Cortex-A9 System or NIOS II System

Lab 2 - Using Logic Instructions with the ARM Processor

Lab 3 - Subroutines and Stacks

Lab 4 - Input/Output in an Embedded System

Lab 5 - Using Interrupts with Assembly Code

Lab 6 - Using C code with the ARM Processor

Lab 7 - Using Interrupts with C code

Lab 8 - Introduction to Graphics and Animation

Intel FPGA MONITOR PROGRAM

Design environment used to compile, assemble, download & debug programs for ARM* Cortex* A9 processor in Intel's Cyclone® V SoC FPGA devices

- Compile programs, specified in assembly language or C, and download the resulting machine code into the hardware system
- Display the machine code stored in memory
- Run the ARM processor, either continuously or by single-stepping instructions
- Modify the contents of processor registers
- Modify the contents of memory, as well as memory-mapped registers in I/O devices
- Set breakpoints that stop the execution of a program at a specified address, or when certain conditions are met

Clean and simple UX

Tutorials at fpgauniversity.intel.com

Download independently or as part of University Program Installer (always free!)

Undergrad Lab Exercise Suites: Embedded Systems

Typically third hardware course in EE, CompEng or CS curriculum

Combines hardware and software

Introduction to embedded Linux

Lab 1 - Getting Started with Linux

Lab 2 - Developing Linux Programs that Communicate with the FPGA

Lab 3 - Character Device Drivers

Lab 4 - Using Character Device Drivers

Lab 5 - Using ASCII Graphics for Animation

Lab 6 - Introduction to Graphics and Animation

Lab 7 - Using the ADXL345 Accelerometer

Lab 8 - Audio and an Introduction to Multithreaded Applications

Lab Exercise Suites: Machine Learning Basics

Machine Learning on FPGAs

Senior or grad-level course in EE, CompEng, CS or data science curriculum

Teaches how to use the Intel® SDK for OpenCL™ Applications with FPGAs

Basic understanding of AI fundamentals recommended*

Lab 1 – Introduction to OpenCL

Lab 2 – Image Processing

Lab 3 – Lane Detection for Autonomous Driving

Lab 4 – Linear Classifier for Handwritten Digits

Lab 5 – Neural Networks

Lab 6 – Using the Deep Learning Accelerator Library

Lab 7 – Integration OpenCL Accelerators into Existing Software

**For foundational AI & Machine Learning curriculums, visit our partner program [Intel AI Academy](#)*

AI Academy Course Outline

Runs in Cloud on Arria 10 PAC card

Contains Slides, Lab exercises, and recordings for each class

<https://software.intel.com/en-us/ai-academy/students/kits/dl-inference-fpga>

Class 1 - Introduction to FPGAs for deep learning inferencing

Class 2 - Building a deep learning computer vision application w/ Acceleration

Class 3 - Introduction to the OpenVINO™ toolkit

Class 4 - Introduction to the Deep Learning Accelerator Suite for Intel FPGAs

Class 5 - Introduction to the Acceleration Stack for Intel Xeon CPU with FPGAs

Lab 1 - Deploy an application on an Intel CPU using DL framework

Lab 2 - Deploy an application on an Intel CPU using the OpenVINO toolkit

Lab 3 - Accelerate the application on an Intel FPGA

In-Person Workshops

Throughout the year our technical outreach team visits universities and industry conferences around the world to conduct hands-on workshops that train professors and students on how to use Intel FPGAs for education and research.

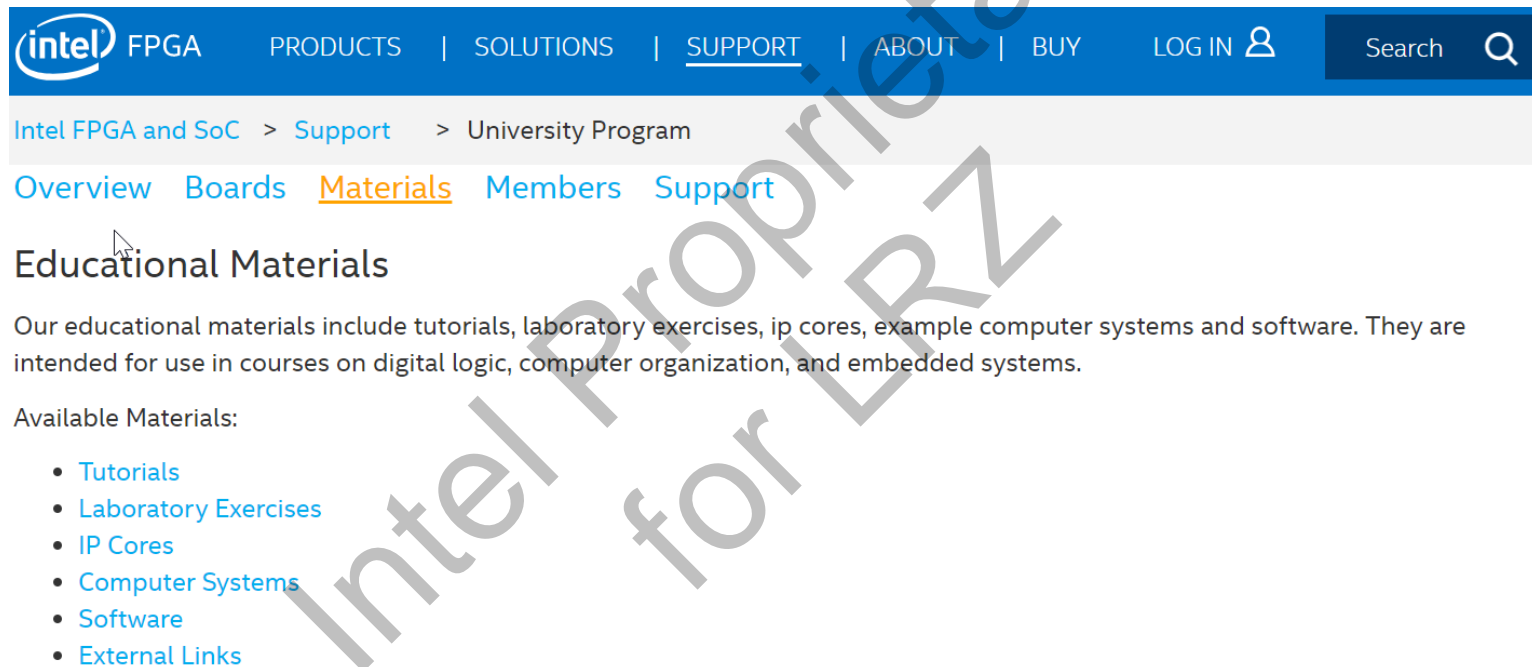
Topics:

Intro to FPGAs and Quartus (4 hrs.)	Embedded Design using Nios II (4 hrs.)
High-Speed IO (4 hrs.)	High-level Synthesis (4 hrs.)
Static Timing Analysis of Digital Circuits (4 hrs.)	Machine Learning Acceleration (4 hrs.)
Simulation & Debug (4 hrs.)	Modern Applications of FPGAs (1 hr.)
Embedded Linux (4 hrs.)	How to Get Hired in the Tech Industry (1 hr.)



Contact us at FPGAUniversity@intel.com to inquire about scheduling a workshop

Find Materials:

FPGAUniversity.INTEL.com



The screenshot shows the Intel FPGA University website. The top navigation bar is blue with the Intel logo and links for FPGA, PRODUCTS, SOLUTIONS, SUPPORT (underlined), ABOUT, BUY, and LOG IN. A search bar is on the right. Below the navigation bar, the breadcrumb trail reads 'Intel FPGA and SoC > Support > University Program'. A secondary navigation bar contains links for Overview, Boards, Materials (highlighted in orange), Members, and Support. The main heading is 'Educational Materials'. The text below states: 'Our educational materials include tutorials, laboratory exercises, ip cores, example computer systems and software. They are intended for use in courses on digital logic, computer organization, and embedded systems.' Under 'Available Materials:', there is a bulleted list: Tutorials, Laboratory Exercises, IP Cores, Computer Systems, Software, and External Links.

intel FPGA | PRODUCTS | SOLUTIONS | SUPPORT | ABOUT | BUY | LOG IN  Search 

Intel FPGA and SoC > Support > University Program

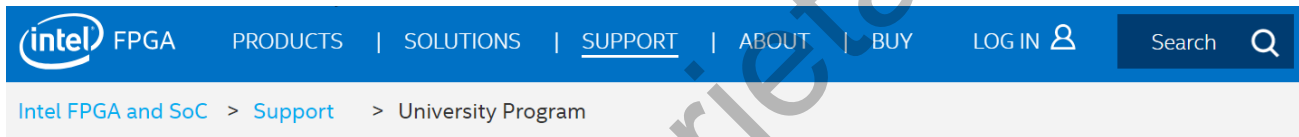
Overview Boards **Materials** Members Support

Educational Materials

Our educational materials include tutorials, laboratory exercises, ip cores, example computer systems and software. They are intended for use in courses on digital logic, computer organization, and embedded systems.

Available Materials:

- Tutorials
- Laboratory Exercises
- IP Cores
- Computer Systems
- Software
- External Links



[Overview](#) [Boards](#) [Materials](#) [Members](#) [Support](#)

Membership Overview

The Intel® FPGA University Program offers donations of licenses for software and intellectual property (IP), and donations of FPGA hardware. To submit a donation request, you must be registered as a member of the Intel FPGA University Program. Membership is available to faculty and staff of universities and colleges. To enroll in the program or to make requests, click on the links to the online forms below.

Students are ineligible to be members. But students can download and use [Quartus® Prime Lite Edition software](#) free of charge, and can purchase boards at the academic price from [Terasic* Technologies](#).

Online forms:

- [Enrollment Request](#): Become a member
- [License Request](#): Free academic licenses for software and IP
- [Hardware Request](#): Board and device donations
- [Purchase Request](#): Boards and devices at academic prices
- [My Account](#): View your profile and request history

Contact the University Team



Rebecca Nevin
Outreach Manager
Intel FPGA University Program
rebecca.l.nevin@intel.com



Larry Landis
Senior Manager
New User Experience Group
lawrence.landis@intel.com



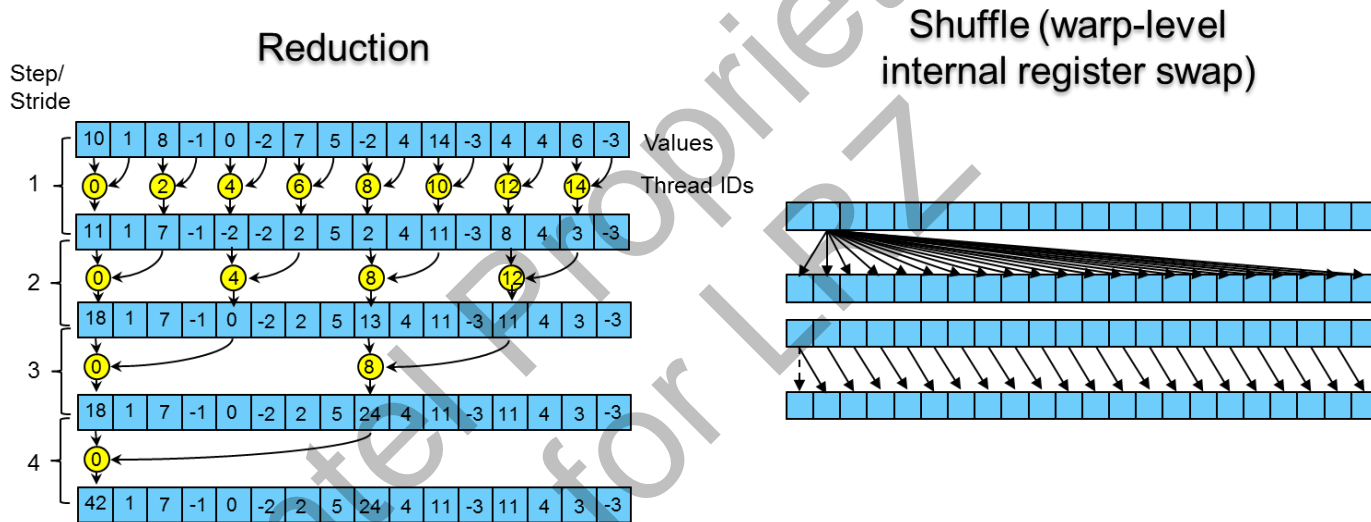
BACKUP

GPU Comparison

Intel Proprietary
for LRZ

How do GPUs Deal With Fine Grained Data Sharing?

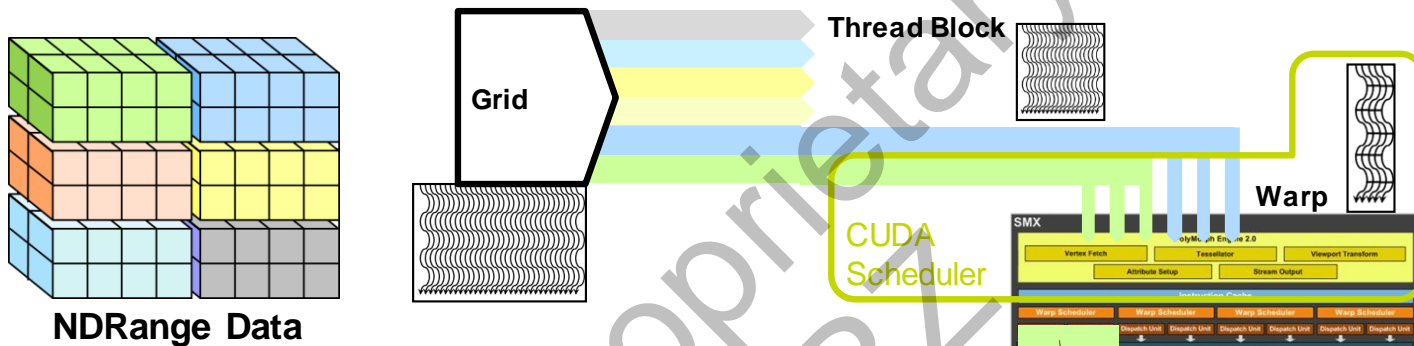
Some GPU techniques involve implicit SIMT synchronization



FPGA threads aren't warp-locked, so implicit sync doesn't make sense

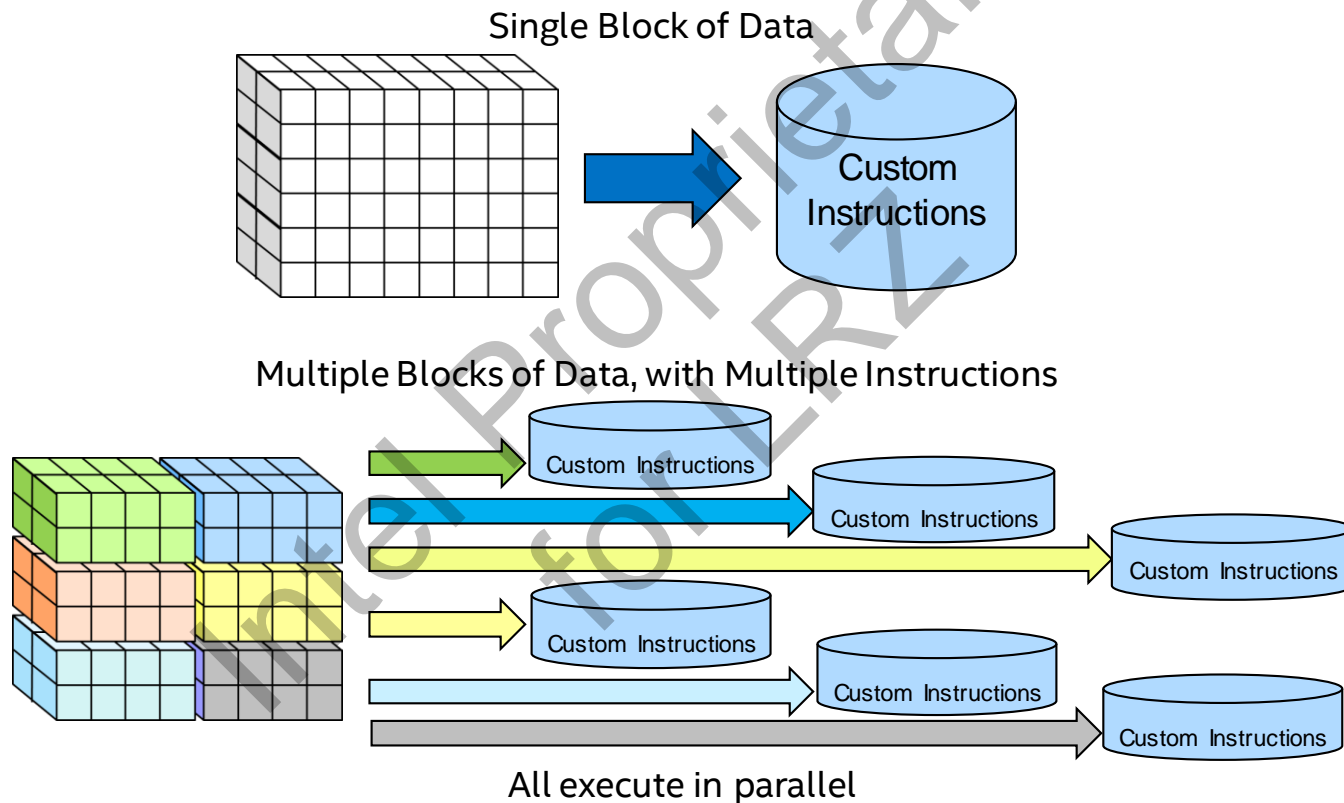
- FPGAs do exactly what you ask them to do the way you code it

An Even Closer Look: CUDA Execution Model



	FERMI GF100 SM	FERMI GF104 SM	KEPLER GK104 SMX	KEPLER GK110 SMX	MAXWELL GM107 SMM
Compute Capability	2.0	2.1	3.0	3.5	5.0
Shared Memory/SM	48KB	48KB	48KB	48KB	64KB
32-bit Registers/SM	32768	32768	64K	64K	64K
Max Threads/Thread Block	1024	1024	1024	1024	1024
Max Thread Blocks/SM	8	8	16	16	32
Max Threads/SM	1536	1536	2048	2048	2048
Threads/Warp	32	32	32	32	32
Max Warps/SM	48	48	64	64	64
Max Registers/Thread	63	63	63	255	255

FPGA Execution Model



Divergent Control Flow on GPU

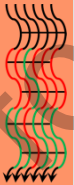
Single instruction

- Thread-locked work items running through different branches
- Serialized
- Major performance factor

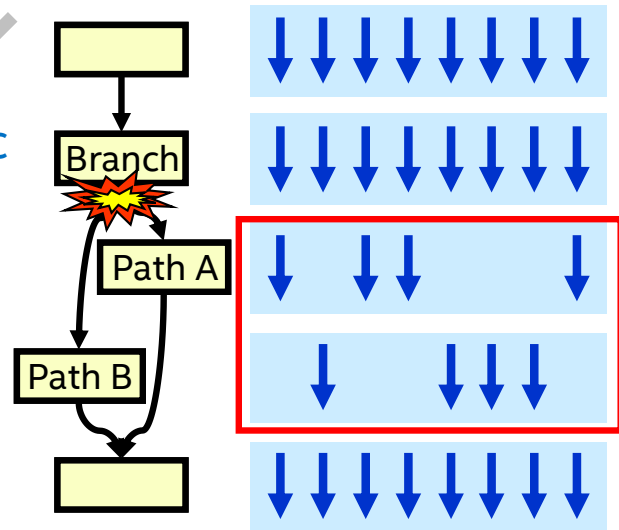
```
for (i=0;i<N;i++)  
  if (x[i]<y[i])  
    foo() else bar();
```

GPU uses SIMT pipeline to save area on control logic

```
mask = (x[i]<y[i])  
if mask foo()  
mask = ~mask  
if mask bar();
```



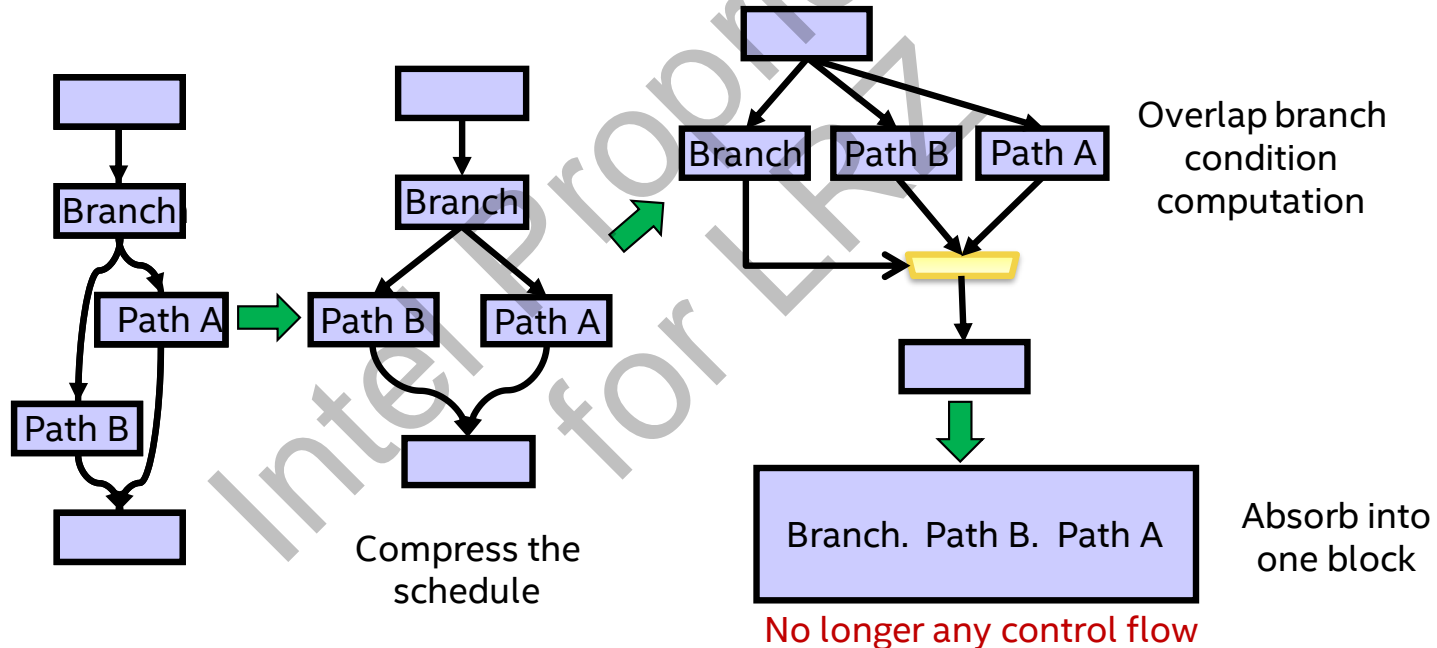
CPUs offer branch prediction



Divergent Control Flow: Just Fine for FPGA

FPGA data path already has all operations in silicon

- Speculatively execute



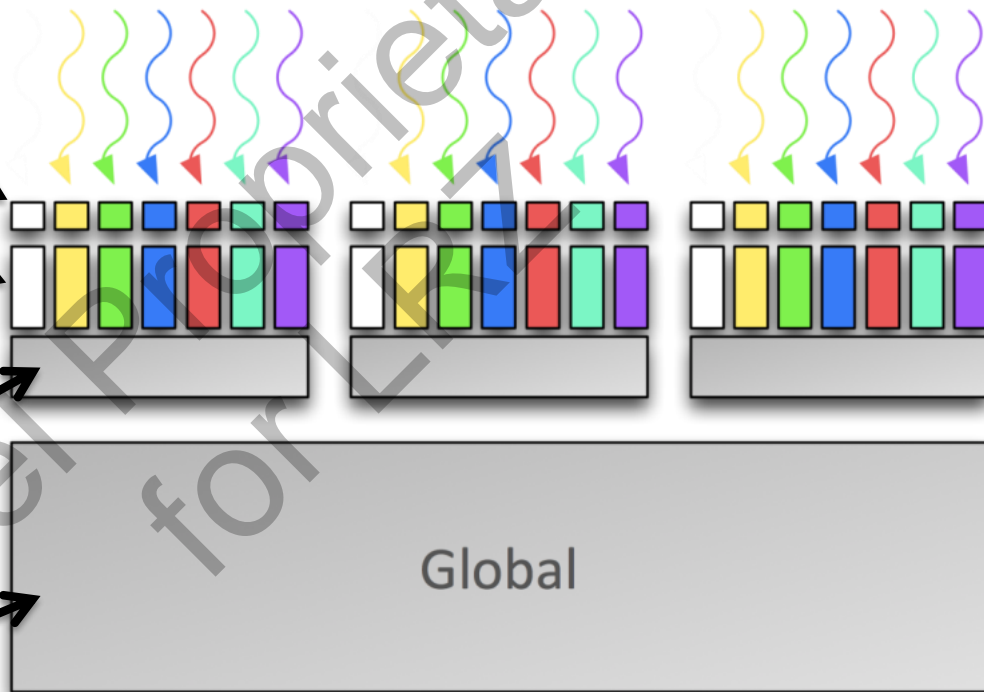
Memory Hierarchy

1. Register data:
Registers in FPGA
fabric

2. Private data:
Registers in FPGA
fabric

3. Local memory:
On-chip RAMs

4. Global memory:
Off-chip external
memory

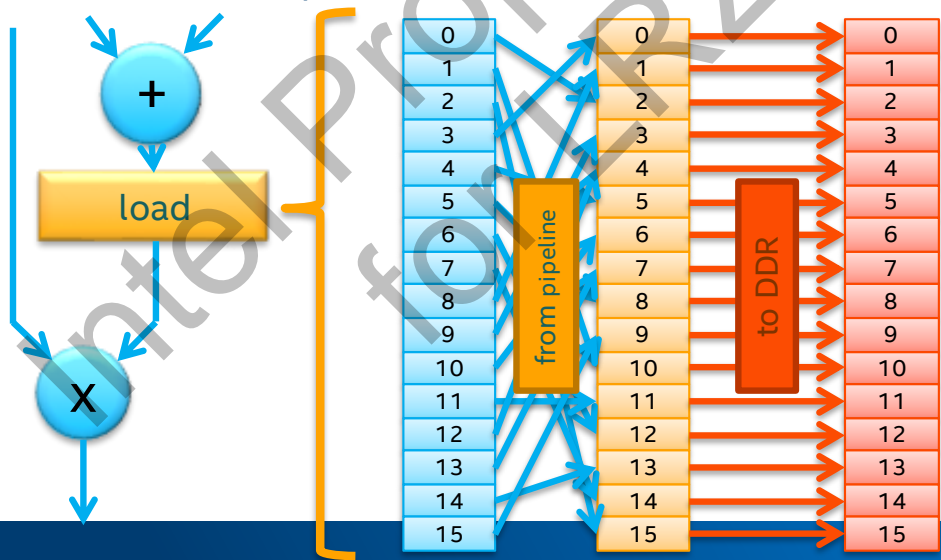


External Memory Dynamic Coalescing

For CPU/GPU the cache and memory controller handle

For FPGA, we create dynamic coalescing hardware matched to specific memory characteristics connected to

- Re-order memory accesses at runtime to exploit data locality
- DDR is extremely inefficient at random access
- Access with row bursts whenever possible



On-chip FPGA Memory

“Local” memory uses on-chip block RAM resources

- Very high bandwidth, 8TB/s,
- Random access in 2 cycles
- Limited capacity

The memory system is customized to your application

- Huge value proposition over fixed-architecture accelerators

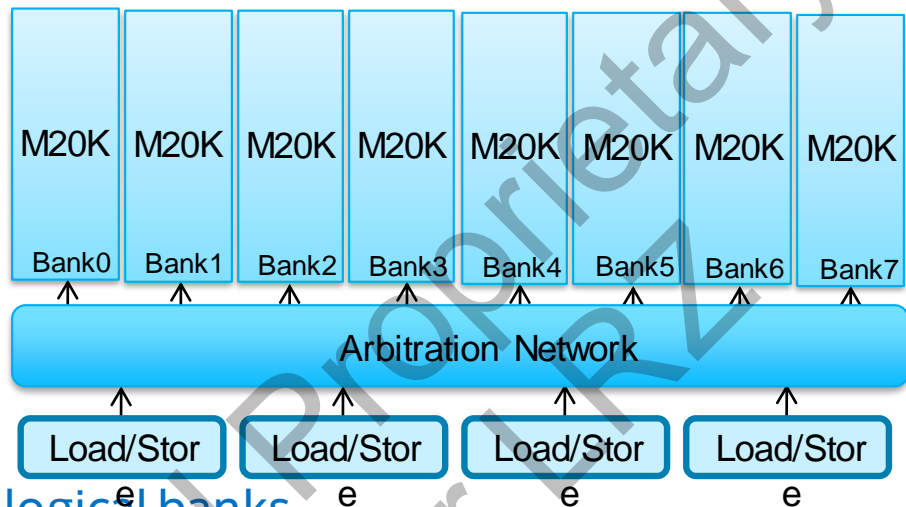
Banking configuration (number of banks, width), and interconnect all customized for your kernel

- Automatically optimized to eliminate or minimize access contention

Key idea: Let the compiler minimize bank contention

- If your code is optimized for another architecture (e.g. `array[tid + 1]` to avoid bank collisions), undo the fixed-architecture workarounds
- Can prevent optimal structure from being inferred

FPGA Local Memory



Split memory into logical banks

- An N-bank configuration can handle N-requests per clock cycle as long as each request addresses a different bank
- Manipulate memory addresses so that parallel threads likely to access different banks – reduce collisions

Local Memory Attributes

Annotations added to local memory variables to improve throughput or reduce area

Banking control:

- numbanks
- bankwidth

Port control:

- numreadports/numwriteports
- singlepump/doublepump

numbanks(N) and bandwidth(N) memory attribute

What does it do?

Specifies the banking geometry for your local memory system

A bank = single independent memory system

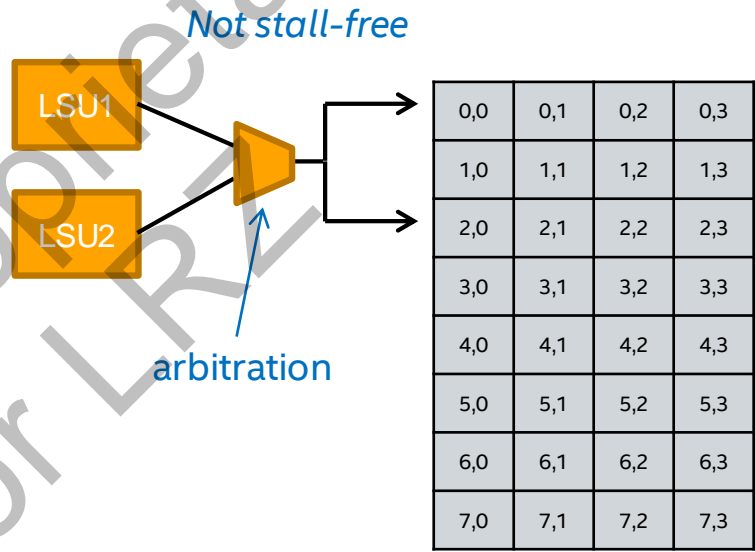
What is it for?

Can be used to optimize LSU-to-memory connectivity in an effort to boost performance

Banking should be set up to maximize “stall-free” accesses

numbanks(N) and bankwidth(N) memory attribute

```
local int lmem[8][4];  
  
#pragma unroll  
for(int i = 0; i<4; i+=2)  
{  
    lmem[i][x] = ...;  
}
```



```
local int lmem[8][4]
```


numbanks(N) and bankwidth(N) memory attribute

```
local int
__attribute__((numbanks(8),
               bankwidth(16)))
lmem[8][4];

#pragma unroll
for(int i = 0; i<4; i+=2)
{
    lmem[i][x & 0x3] = ...;
}
```

Mask access to tell compiler
no out-of-bounds accesses

Stall-free

LSU1

LSU2

0,0	0,1	0,2	0,3	Bank 0
1,0	1,1	1,2	1,3	Bank 1
2,0	2,1	2,2	2,3	Bank 2
3,0	3,1	3,2	3,3	Bank 3
4,0	4,1	4,2	4,3	Bank 4
5,0	5,1	5,2	5,3	Bank 5
6,0	6,1	6,2	6,3	Bank 6
7,0	7,1	7,2	7,3	Bank 7

local int lmem[8][4]

numreadports/numwriteports and
singlepump/doublepump memory attribute

What does it do?

num<read/write>ports: specifies the number of read/write ports in the local memory system

<single/double>pump: specifies the pumping of the local memory system (1x/2x clock)

What is it for?

Controls the number of memory blocks used to implement the local memory system

numreadports/numwriteports and singlepump/doublepump memory attribute

```
local int  
__attribute__((singlepump,  
              numreadports(3),  
              numwriteports(1)))  
lmem[16];
```

```
local int  
__attribute__((doublepump,  
              numreadports(3),  
              numwriteports(1)))  
lmem[16];
```

