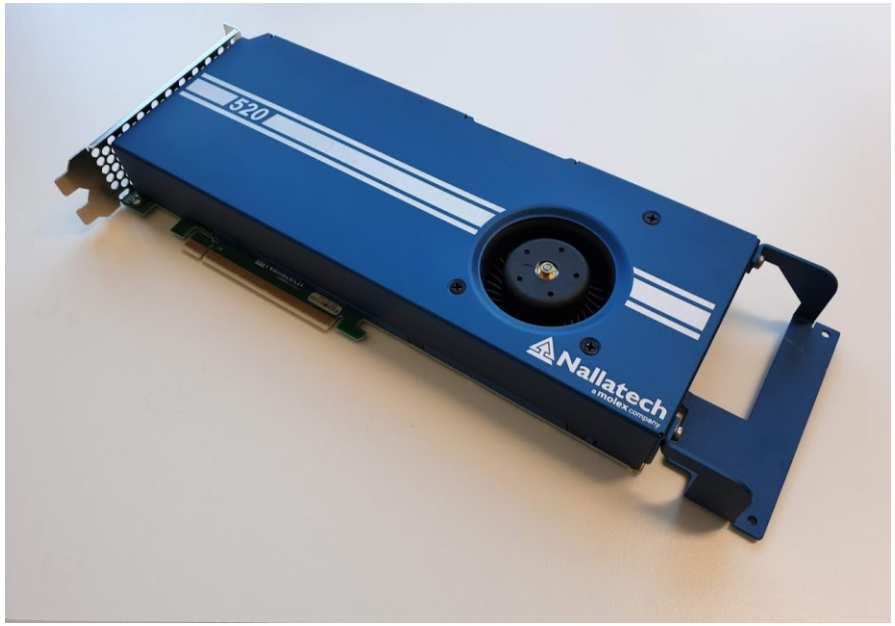
OpenCL on FPGAs



What are FPGAs?

- Reprogrammable hardware
- Integrate huge numbers of lookup tables (LUTs), registers, on-chip memories, and arithmetic hardware (e.g. DSP blocks)
- These on-chip resources are connected through a reconfigurable network
- Traditionally programmed through a very low-level hardware description language
  - VHDL or Verilog

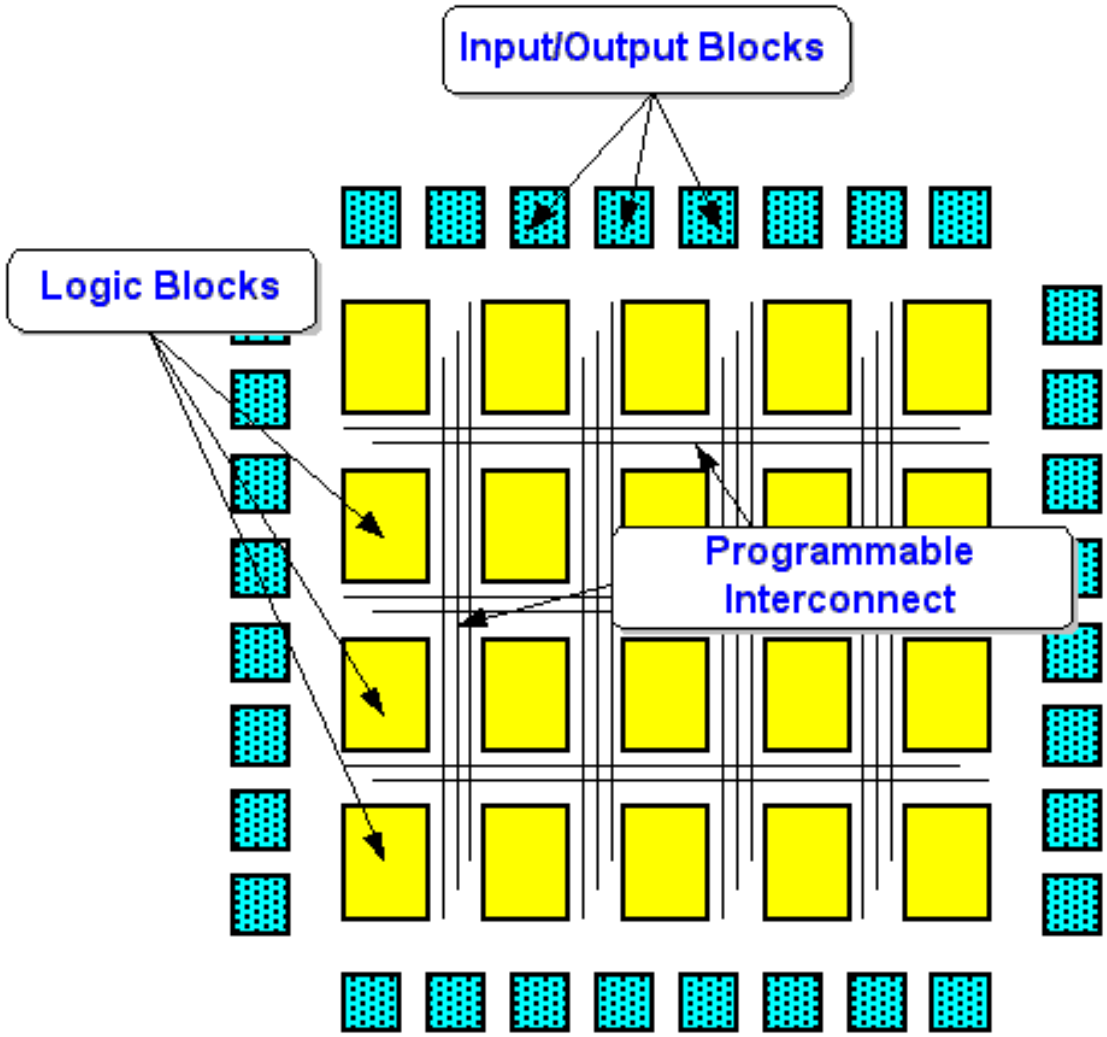
Why FPGAs?

- Prototyping hardware designs
  - Application-Specific Integrated Circuit (ASIC): customized circuit for a specialized application e.g. aerospace microcontroller, Bitcoin miner
  - Application-Specific Standard Product (ASSP): customized for application market e.g. automotive microcontrollers, smart phone chips
- Production systems
  - Reconfigurable = can modify electronics in situ
  - As cheap and power efficient as ASICs (except for very large volumes)

OpenCL on FPGAs

- FPGA architectures are very different from GPUs and CPUs
- Requires a completely different approach to achieve good performance
- On CPUs/GPUs, you want lots of parallelism: i.e. lots of work-items and work-groups
- For FPGAs, you want just a few work-items, each representing a long pipeline
- Base-level for programming FPGAs is hardware definition language (HDL): Verilog or VHDL
  - Detailed; low-level; highly-specialized
- OpenCL makes programming FPGAs more accessible

FPGA Architecture

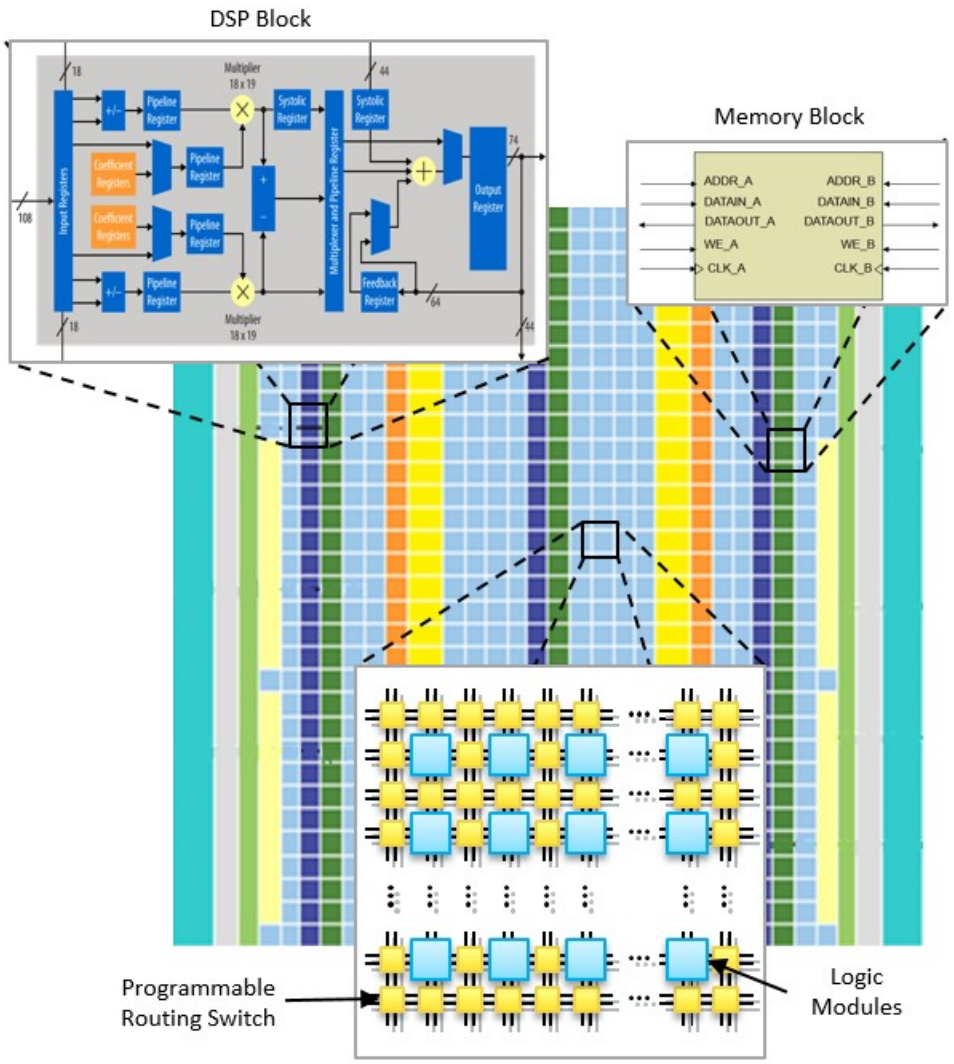


Source: http://www.fpga-site.com/faq.html

FPGA Hard Blocks

Most FPGA packages include blocks of predefined hardware (hard blocks) to implement commonly required functions

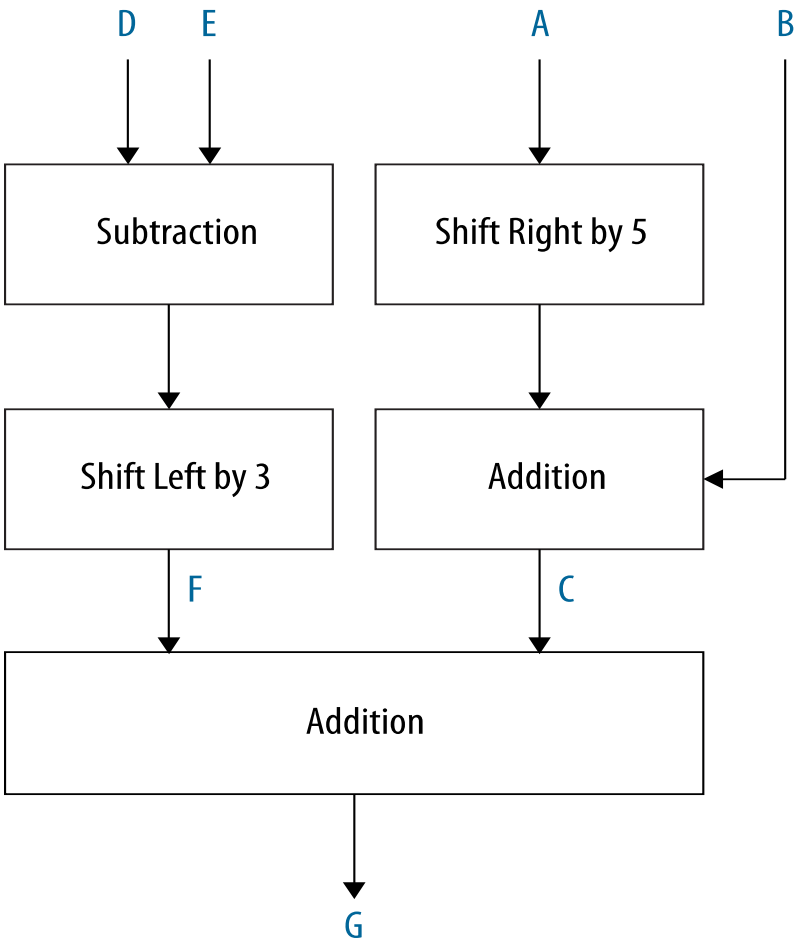
- Digital signal processor (DSP)
- Arithmetic units
- I/O logic
- Memory blocks



Source: Intel FPGA for OpenCL SDK Pro Edition: Best Practices Guide

Compiling OpenCL into Hardware

```
size_t index = get_global_id(0);
C[index] = (A[index] >> 5) + B[index];
F[index] = (D[index] – E[index]) << 3;
G[index] = C[index] + F[index];
```

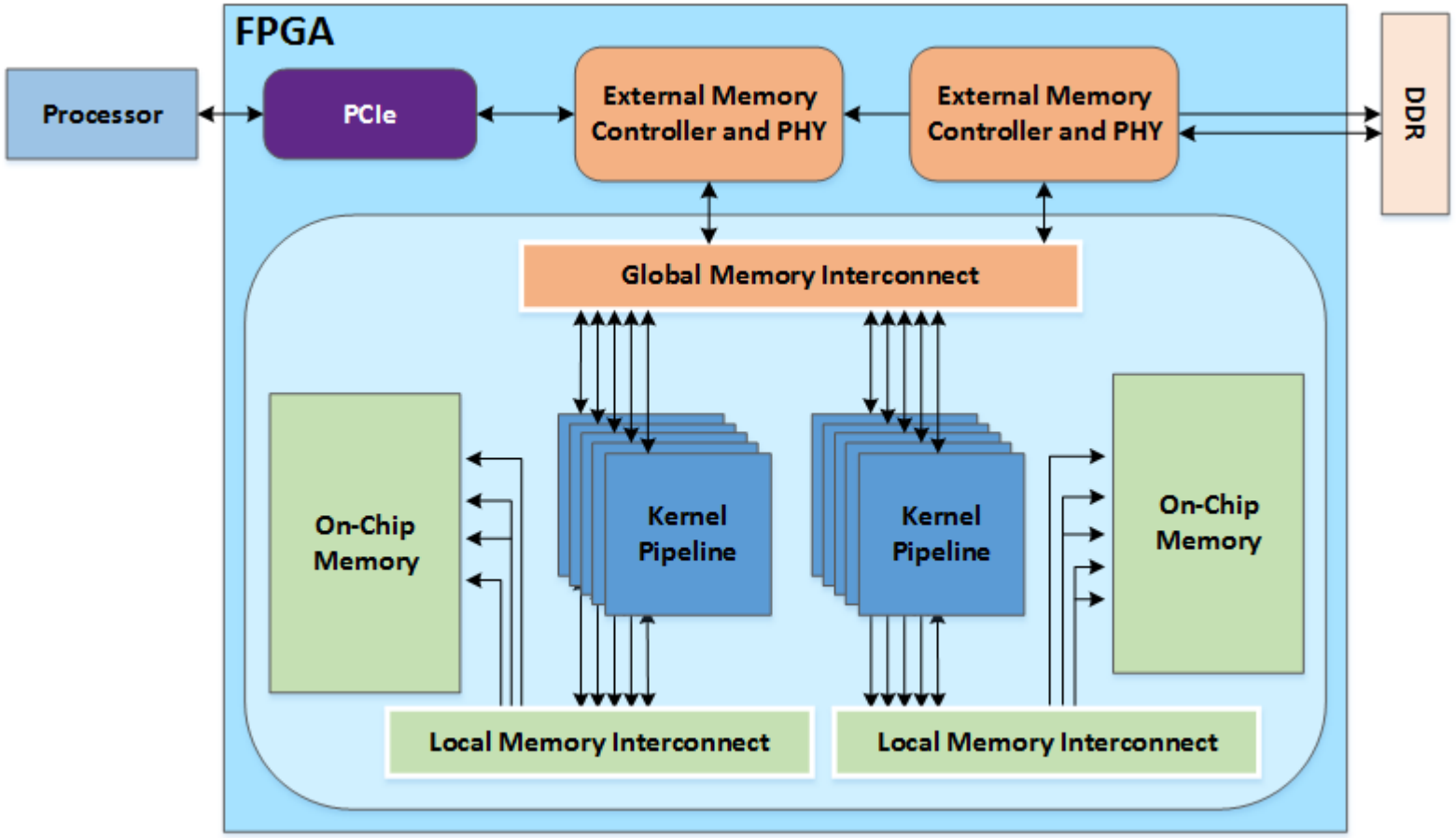


The Intel FPGA SDK for OpenCL Offline Compiler provides a custom pipeline structure that speeds up computation by allowing operations within a large number of work-items to occur concurrently. The offline compiler can create a custom pipeline that calculates the values for variables C, F and G every clock cycle, as shown below. After a ramp-up phase, the pipeline sustains a throughput of one work-item per cycle.

Source: Intel FPGA for OpenCL SDK Pro Edition: Best Practices Guide

OpenCL Design Components

An OpenCL system design provides kernels with access to local and global memory (just like in a regular OpenCL program)



Source: Intel FPGA for OpenCL SDK Pro Edition: Best Practices Guide

FPGA Optimisation Tips

- Create single work-item kernels if:
  - You cannot break down an algorithm into separate work-items easily because of data dependencies that arise when multiple work-items are in flight.
  - You organize your OpenCL application in multiple kernels, you use channels to transfer data among the kernels, and the data processing sequence is critical to your application.
  - Equivalent to an NDRange size of (1, 1, 1)

Single Work-Item Kernels

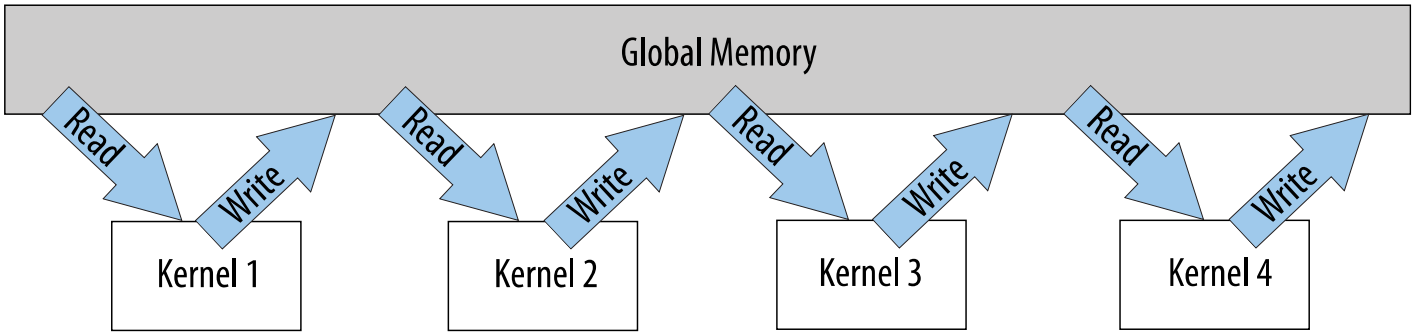
- In this approach, the FPGA OpenCL compiler will attempt to pipeline the work-item
- Special care needed to ensure the compiler can pipeline loops

More Tips for FPGA Optimisation

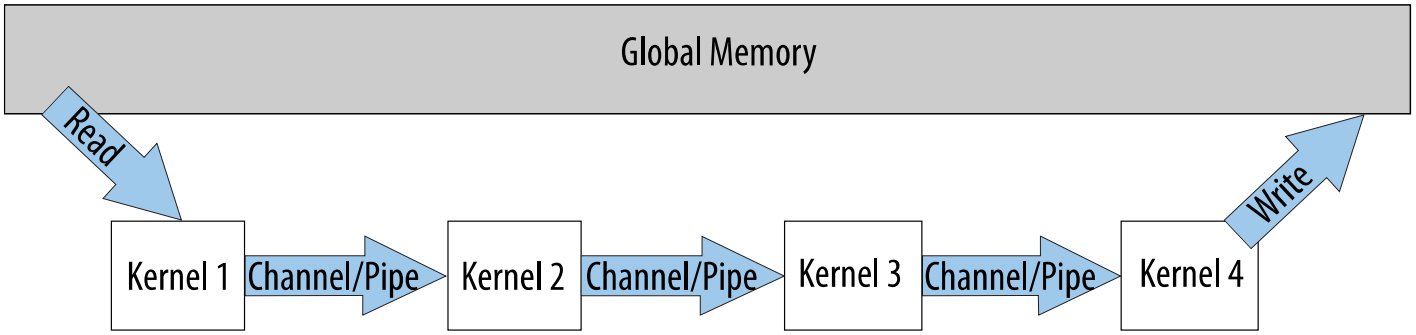
- Optimize each kernel to target a single compute unit first
- Then scale the number of compute units up until you've filled the FPGA
  - Compiling with fewer compute units takes much less FPGA compilation time
- Consider moving data between kernels using OpenCL pipes or vendor extensions such as channels
- Unrolling loops can help FPGA OCL compilers
  - e.g. `#pragma unroll 8`
- Optimise floating point operations
- Avoid expensive operations
- Allocate memory aligned to at least 64 bytes
- Use restrict to avoid pointer aliasing
- Avoid work-item ID-dependent backward branching

Using Pipes and Channels

Global Memory Access Pattern Before AOCL Channels or Pipes Implementation



Global Memory Access Pattern After AOCL Channels or Pipes Implementation



Source: Intel FPGA for OpenCL SDK Pro Edition: Best Practices Guide

Optimising Floating Point

- Giving the FPGA OpenCL compiler more freedom regarding IEEE compliance can make a huge difference in performance
- Key compiler flags include:
  - --fp-relaxed : compiler can change order of operations
  - --fpc : minimise type conversions and combine multiple rounding operations into one. Results in use of fused multiple-accumulate (FMAC) instructions
- Fixed point even better than floating point on FPGAs, can pack in more execution units
  - OpenCL supports 8, 16, 32 and 64-bit fixed point

Operation costs on FPGAs

- Expensive operations include:
  - Integer division and modulo (remainder) operators
  - Most floating-point operators except addition, multiplication, absolute value, and comparison
  - Atomic functions
- In contrast, cheap operations include:
  - Binary logic operations such as AND, NAND, OR, NOR, XOR, and XNOR
  - Logical operations with one constant argument
  - Shift by constant
  - Integer multiplication and division by a constant that is a power of two

Source: Intel FPGA for OpenCL SDK Pro Edition: Best Practices Guide

Other FPGA Kernel Tips

- Use well-formed loops
  - These have an exit condition that compares against an integer bound, and have a simple induction increment of one per iteration
- Avoid pointer arithmetic, use simple array indexing instead
- Avoid complex loop exit conditions
- Convert nested loops into a single loop
- Declare variables in the deepest scope possible

OpenCL on FPGA Summary

- You'll probably need completely different kernels for optimal performance on an FPGA
- Still uses the same overall OpenCL host infrastructure though
- In theory, OpenCL supports using CPUs, GPUs, DSPs and FPGAs all at the same time…

Useful Resources

Intel (formerly Altera):

- https://www.intel.com/content/www/us/en/programmable/support/support-resources.html
- https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf

Xilinx:

- http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html#documentation