Introduction to FPGA acceleration

Marco Barbone
m.barbone19@imperial.ac.uk

CC BY

Imperial College London

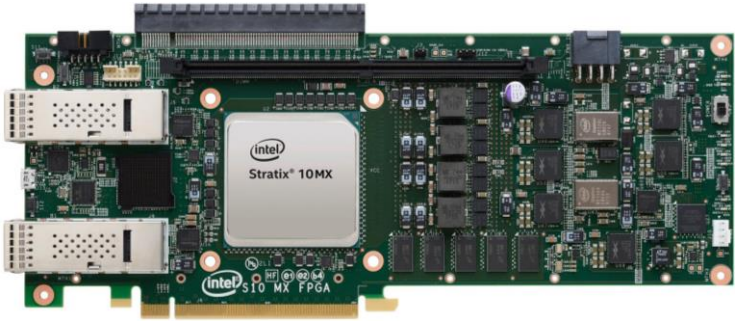
About me

- M.Sc. in Computer Science and Networking Scuola superiore Sant'Anna & Università di Pisa
- CERN OpenLab Summer student
- Research Software Engineer, Maxeler Technologies
- Ph.D. Student, Custom Computing research group, Imperial College London, Supervisor Prof. Wayne Luk
- Member of Imperial College CMS group
- Associate Ph.D. Student, The Institute of Cancer Research, London

Imperial College London

What is an FPGA?

A **field-programmable gate array** (**FPGA**) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence the term "field-programmable".



https://www.xilinx.com/content/dam/xilinx/imgs/kits/u200-hero-p.jpg https://www.intel.com/content/dam/altera-www/global/en_US/support/boards-kits/stratix10/mx_fpga/stratix10mxboardtop.jpg/_jcr_content/renditions/cq5dam.web.1920.1080.jpeg https://en.wikipedia.org/wiki/Field-programmable_gate_array

Imperial College London

Microsoft's Project Brainwave

"Microsoft's *Project Brainwave* is a **deep learning platform** for real-time AI inference in the cloud and on the edge. A soft Neural Processing Unit (NPU), based on a high-performance **field-programmable gate array (FPGA),** accelerates deep neural network (DNN) inferencing, with applications in computer vision and natural language processing. Project Brainwave is transforming computing by augmenting CPUs with an interconnected and configurable compute layer composed of programmable silicon."

https://www.microsoft.com/en-us/research/project/project-brainwave/
https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/

Imperial College London

Amazon EC2 F1

Amazon EC2 F1 instances use FPGAs to enable delivery of custom hardware accelerations. F1 instances are easy to program and come with everything you need to develop, simulate, debug, and compile your hardware acceleration code, including an FPGA Developer AMI and supporting hardware level development on the cloud. **Using F1 instances to deploy hardware accelerations can be useful in many applications to solve complex science, engineering, and business problems** that require high bandwidth, enhanced networking, and very high compute capabilities. **Examples of target applications that can benefit from F1 instance acceleration are genomics, search/analytics, image and video processing, network security, electronic design automation (EDA), image and file compression and big data analytics.**

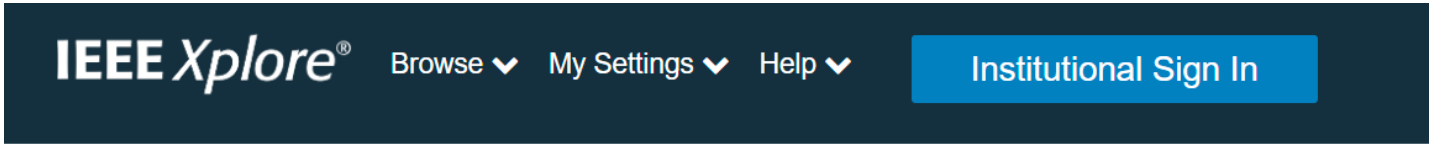
https://aws.amazon.com/ec2/instance-types/f1/

Imperial College London

DNA Sequencing: DRAGEN Platform

The DRAGEN Platform enables labs of all sizes and disciplines to do more with their genomic data. The **DRAGEN Platform uses highly reconfigurable field-programmable gate array technology (FPGA) to provide hardware-accelerated implementations of genomic analysis algorithms**, such as BCL conversion, mapping, alignment, sorting, duplicate marking, and haplotype variant calling.

https://www.illumina.com/products/by-type/informatics-products/dragen-bio-it-platform.html



Conferences > 2019 29th International Confe...

Dataflow Acceleration of Smith-Waterman with Traceback for High Throughput Next Generation Sequencing

Publisher: IEEE

Konstantina Koliogeorgi ; Nils Voss ; Sotiria Fytraki ; Sotirios Xydis ; Georgi Gaydadjiev ; Dimitrios S… **All Authors**

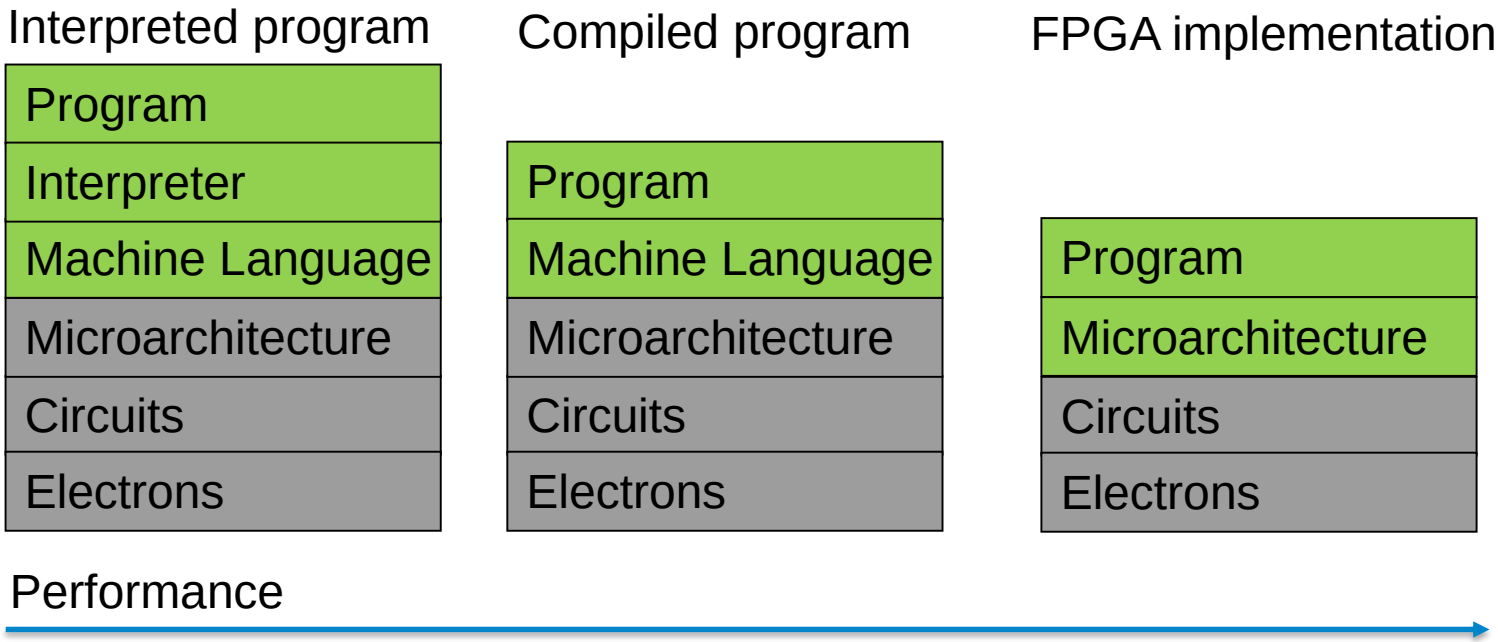
The dataflow architecture for Smith-Waterman Matrix-fill and Traceback alignment stages, to perform short-read alignment on NGS data that **delivers ×18 speedup** over the respective Bowtie2 standalone components, while our codesigned **Bowtie2 demonstrates a 35% boost in performance.**

https://ieeexplore.ieee.org/document/8892144

Imperial College London

Programmable | Fixed

FPGAs fill the gap between software and hardware



Imperial College London

Field Programmable Gate Array

- FPGA is a reconfigurable substrate
  - Reconfigurable functions, interconnection of functions, input/output
- FPGAs fill the gap between software and hardware
  - higher throughput than software → Not always true!
  - lower latency than software
  - more flexibility than hardware

Imperial College London

Latency

FPGAs achieve lower latency than software:

- latency in the order of microseconds
- latency **constant** and **predictable**

CPUs are not suitable as thread overhead is ≈ 10 microseconds

GPUs are even worse since PCIe bus sys-call requires milliseconds

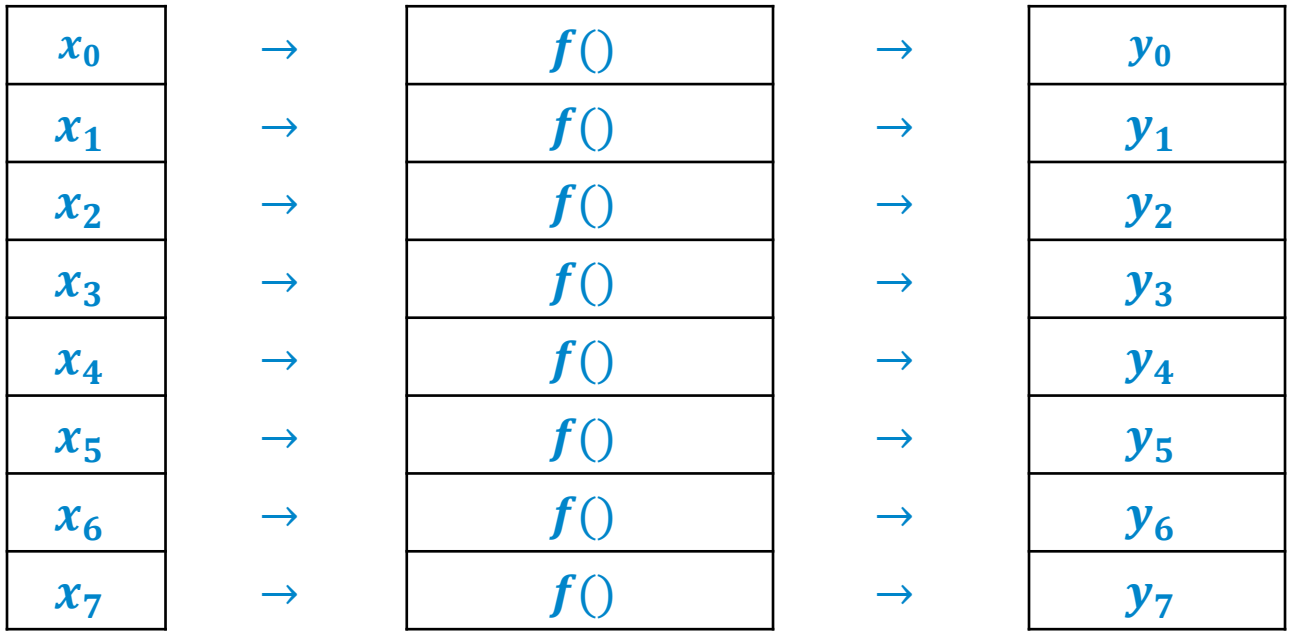
Examples:

- Ultra low latency trading
- CERN trigger system

https://www.velvetech.com/blog/fpga-in-high-frequency-trading/
https://indico.cern.ch/event/557251/contributions/2245644/attachments/1310074/2143323/thea_trg_intro_isotdaq2017.pdf

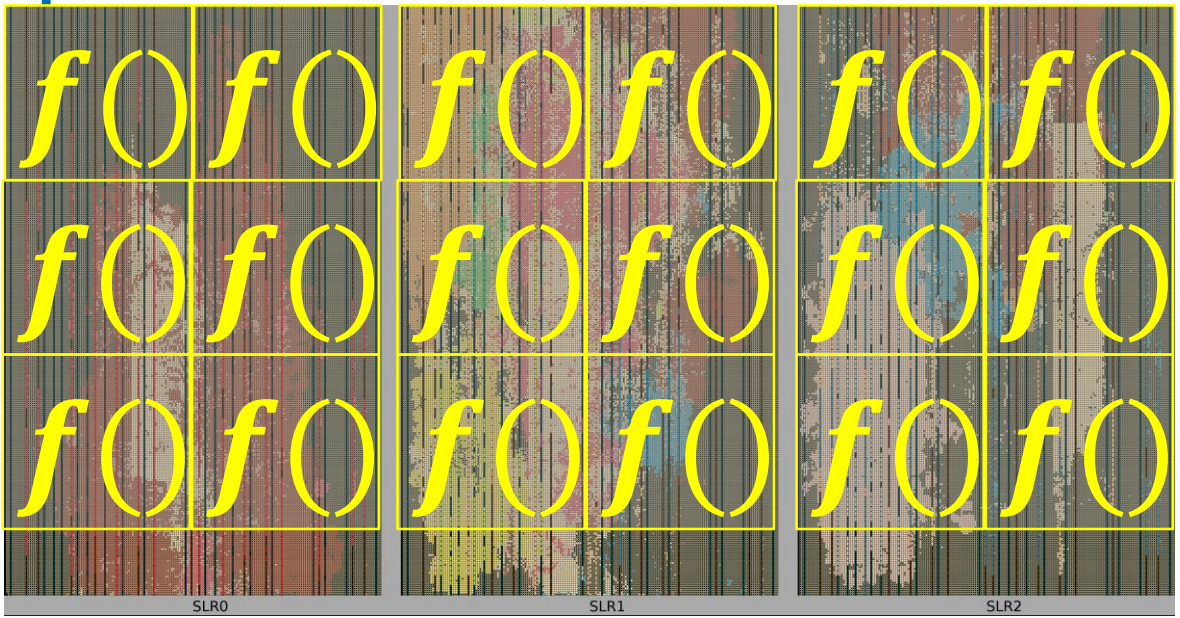
Imperial College London

Throughput



Imperial College London

Throughput



Imperial College London

FPGA Accelerator Card

XILINX

Enlarge

Images are for reference only
See Product Specifications

Mouser No:	217-A-U250-A64G-PQ-G
Mfr. No:	A-U250-A64G-PQ-G
Mfr.:	Xilinx
Customer No:	Customer No
Description:	Accelerator Cards Alveo U250 Data Center Accelerator Card - Active
Lifecycle:	**New Product:** New from this manufacturer.
Datasheet:	A-U250-A64G-PQ-G Datasheet (PDF)
More Information	Learn more about Xilinx A-U250-A64G-PQ-G

Stock:	5
Factory Lead Time:	10 Weeks
Enter Quantity:	Minimum: 1 Multip

Pricing (USD)

Qty.	Unit Price
1	$8,707.34

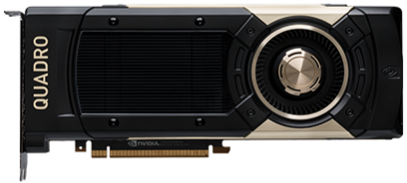
Price $8,707.34

https://eu.mouser.com/

Imperial College London

GP-GPU

QUADRO GV100



$8,999. 00

OVERVIEW SPECS INCLUDED

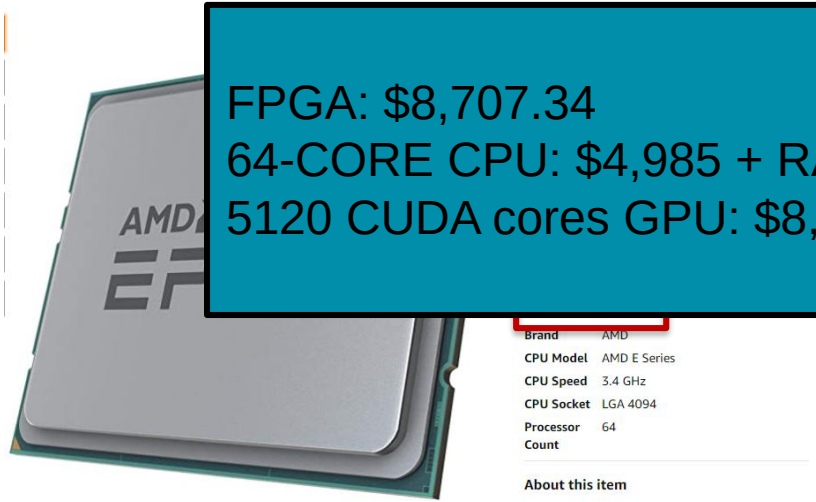
CUDA Parallel-Processing Cores	5120
NVIDIA Tensor Cores	640
GPU Memory	32 GB HBM2
FP64 Performance	7.4 TFLOPS
FP32 Performance	14.8 TFLOPS
FP16 Performance	29.6 TFLOPS
Tensor Performance	118.5 TFLOPS

5120 CUDA cores
Price $8,999

https://www.nvidia.com/en-us/design-visualization/store/

Imperial College London

CPU



FPGA: $8,707.34
64-CORE CPU: $4,985 + RAM
5120 CUDA cores GPU: $8,999

https://www.amazon.com/

Imperial College London

- FPGA have higher throughput than CPUs (and GPUs) if the workload:
  - Has many branch mispredictions
  - Has many cache faults

- CPU: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
- L1 latency = **2 cycles**
- L2 latency = **15 cycles**
- L3 latency = **32 cycles**

https://github.com/DiamonDinoia/cpp-learning/blob/master/cache-latency/cache_test.c

Imperial College London

Simulator example

```
1. void main_loop() {
2.     while (true) {
3.         for (x=0; x<ROW; x++) {
4.             for(y=0;y<COL;y++){
5.                 switch (cell_to_char(PLANET[x][y])) {
6.                     case 'S':
7.                         if(UPDATE[x][y]==0 && shark_rule2(pw, x, y, k, l)==ALIVE) {
8.                             shark_rule1 (pw, x, y, k, l);
9.                         }
10.                        break;
11.                    case 'F':
12.                        if(UPDATE[x][y]==0){
13.                            fish_rule4(pw, x, y, k, l);
14.                            fish_rule3(pw, x, y, k, l);
15.                        }
16.                        break;
17.                }
18.            }
19.        }
20.    }
21.}
```

Imperial College London

```
1. int shark_rule2 (wator_t* pw, int x, int y, int *k, int* l){
2.     int i=0;
3.     long rand=random();
4.     motion move[4];
5.     if (DEATH[x][y]==pw->sd) {
6.         DEATH[x][y]=0;
7.         PLANET[x][y]=char_to_cell('W');
8.         BIRTH[x][y]=0;
9.         return DEAD;
10.    }
11.    if(DEATH[x][y]<pw->sd){
12.        DEATH[x][y]++;
13.    }
14.    if (BIRTH[x][y]<pw->sb){
15.        BIRTH[x][y]++;
16.        return ALIVE;
17.    }
18.    if (BIRTH[x][y]==pw->sb) {
19.        BIRTH[x][y]=0;
20.        inizialize_motion(pw, x, y, move);
21.        i=partialsort(move, 'W');
22.        if(!i){
23.            return ALIVE;
24.        }
25.        new_coordinate(x, y, k, l, rand%i, move);
26.        return ALIVE;
27.    }
28.    return ERROR;
29.}
```

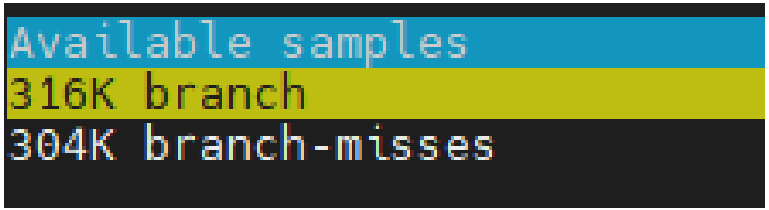
Total of 9 branches in 50 lines of code

Imperial College London

Metrics

```
$    perf record -e branch,branch-misses `./wator
[ perf record: Woken up 108 times to write data ]
[ perf record: Captured and wrote 28.441 MB perf.data (620737 samples) ]
$     perf report -n --symbols=main_loop
$
```

Imperial College
London



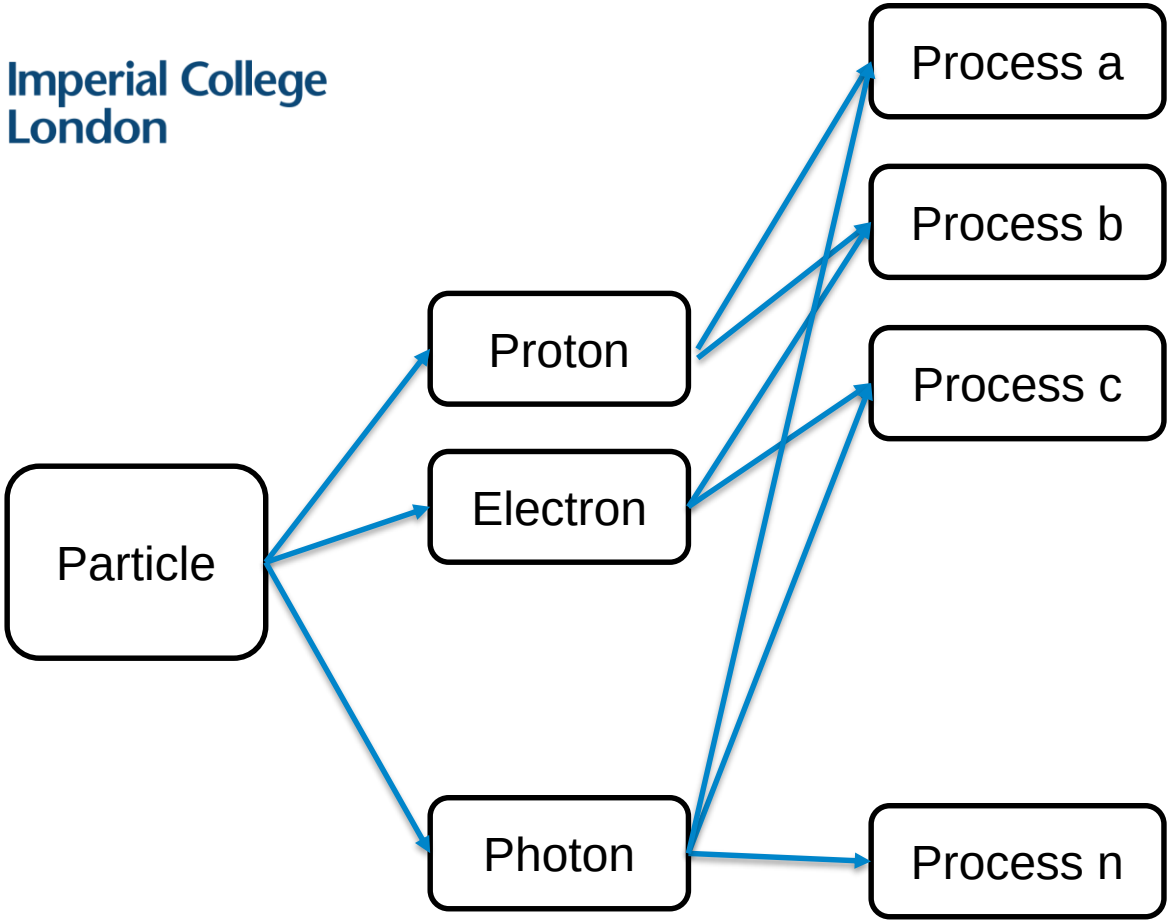
316K branches
304K branch-misses

Imperial College London

Workload Examples

FPGA	GPU
Sparse Matrix	Dense Matrix
Sparse Graph	Dense Graphs
Graph Neural Network	Convolutional Neural Network
Decision tree	
(Monte-Carlo) Simulation	
Genomics computation	

Imperial College London



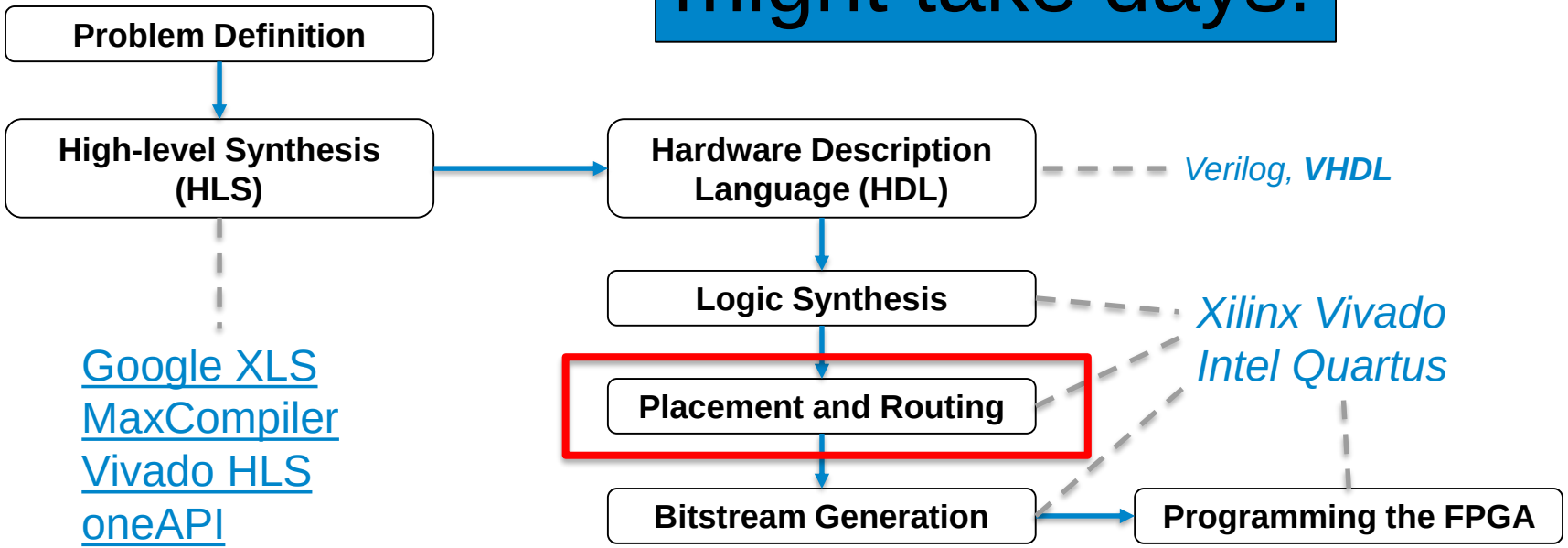
Imperial College London

FPGA Programming

Imperial College London

FPGA Design Flow

Place & Route might take days!



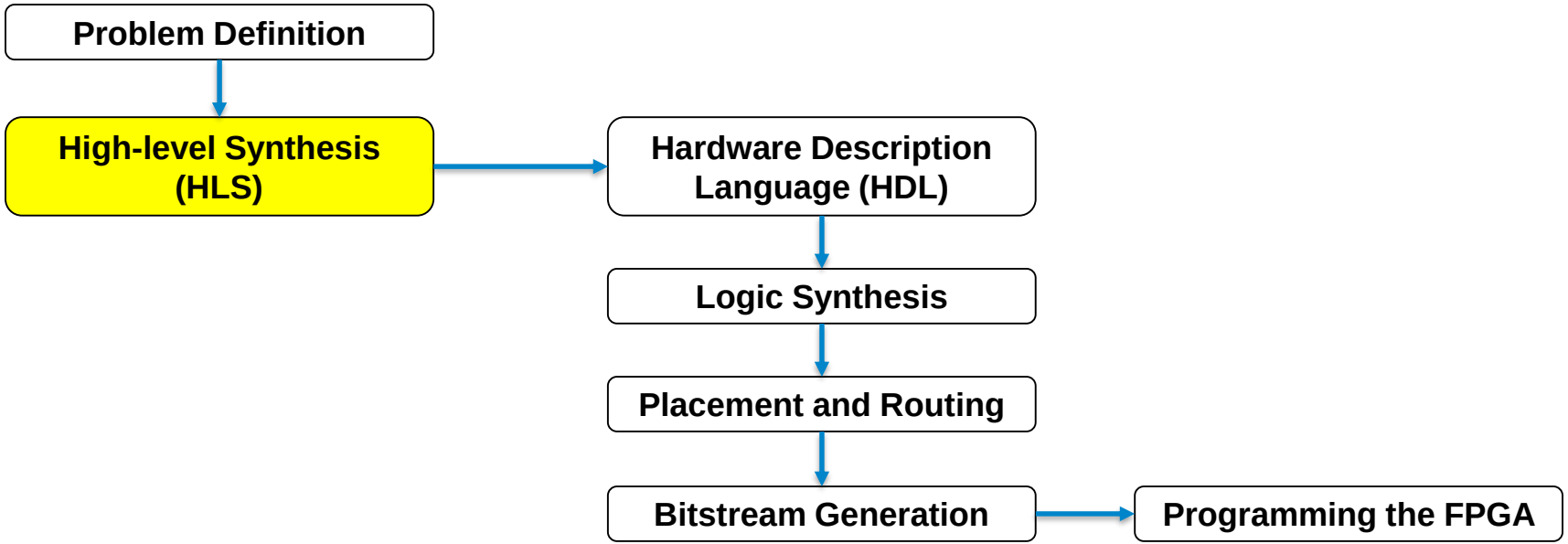
	CPU	GPU	FPGA
Engineering time	short	medium	very long
Compilation time	short	short	very long
Debugging	easy	medium	hard
Maintainability	easy	medium	hard

- Avoid running Place and Route
  - Simulate VHDL, Verilog using ModelSim

- ModelSim is slow!
  - It is possible to execute very small testbenches in a reasonable amount of time

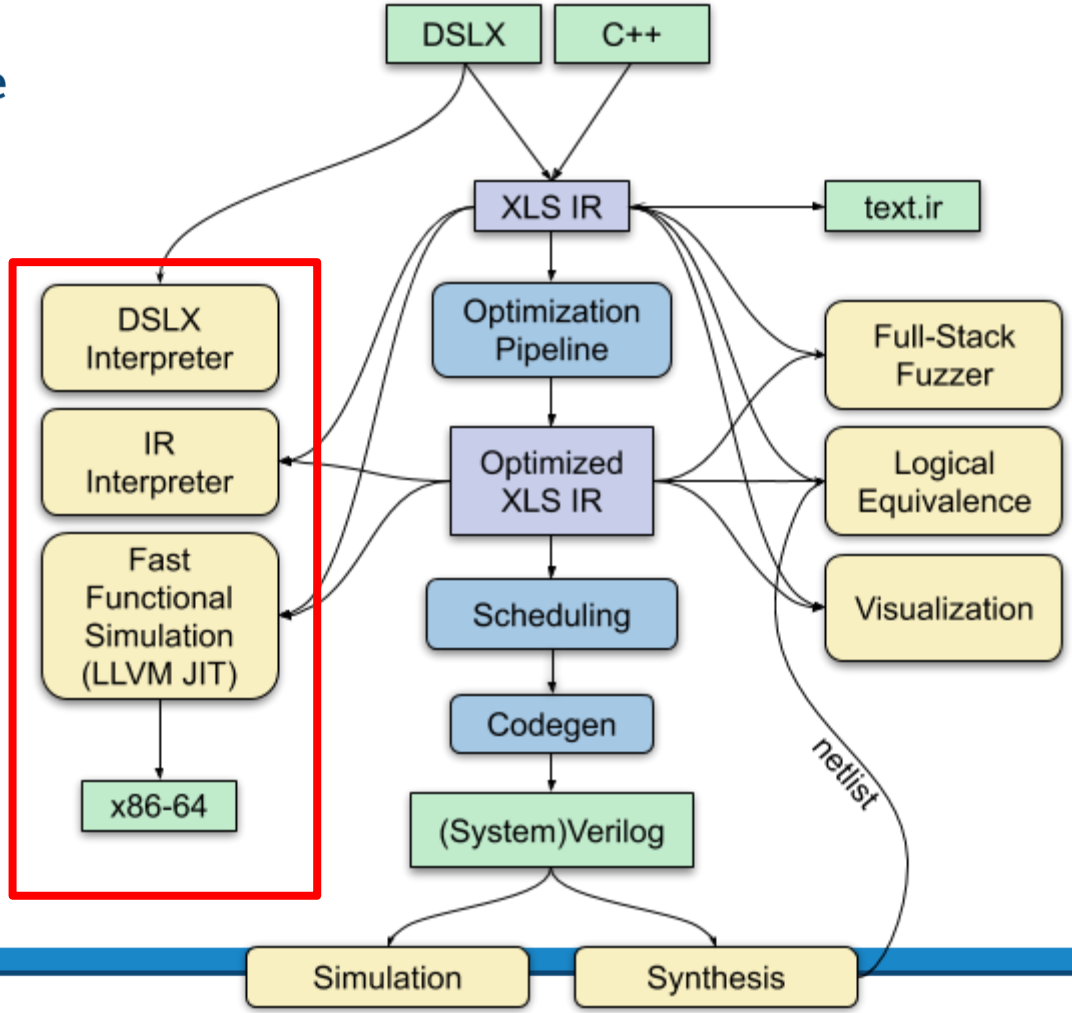
Imperial College London

FPGA Design Flow



Imperial College London

It generates a X86-64 executable from source



Imperial College London

High-level Synthesis (HLS)

- Using HLS toolchains makes unit testing possible in a reasonable amount of time
- It is easy to interface with a reference software implementation to compare the results
- It makes FPGA programming much less cumbersome

Imperial College London

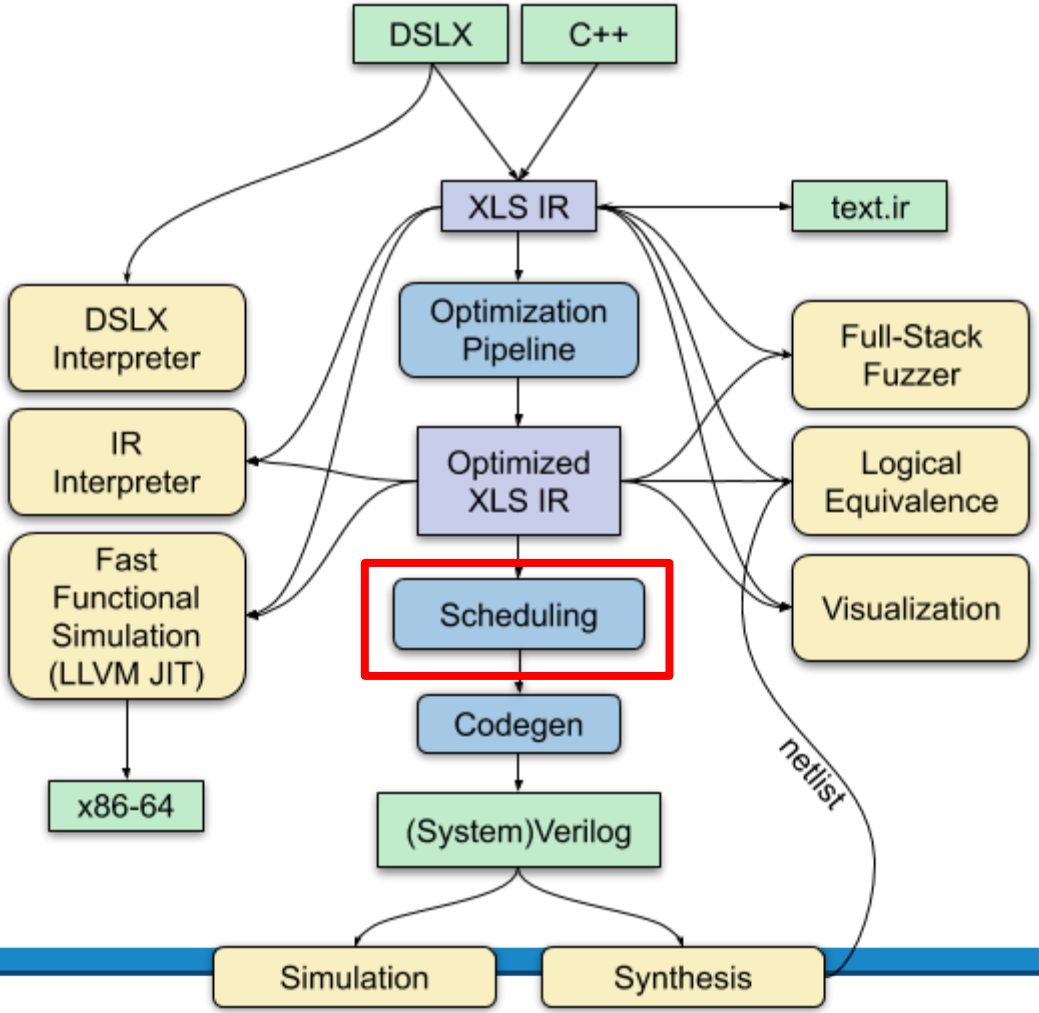
Double floating-point multiplication in Verilog

```
1.  double_multiplier multiplier_39759952(
2.      .clk(clk),
3.      .rst(rst),
4.      .input_a(wire_39069600),
5.      .input_a_stb(wire_39069600_stb),
6.      .input_a_ack(wire_39069600_ack),
7.      .input_b(wire_39795024),
8.      .input_b_stb(wire_39795024_stb),
9.      .input_b_ack(wire_39795024_ack),
10.     .output_z(wire_39795168),
11.     .output_z_stb(wire_39795168_stb),
12.     .output_z_ack(wire_39795168_ack));
```

Using a library function!

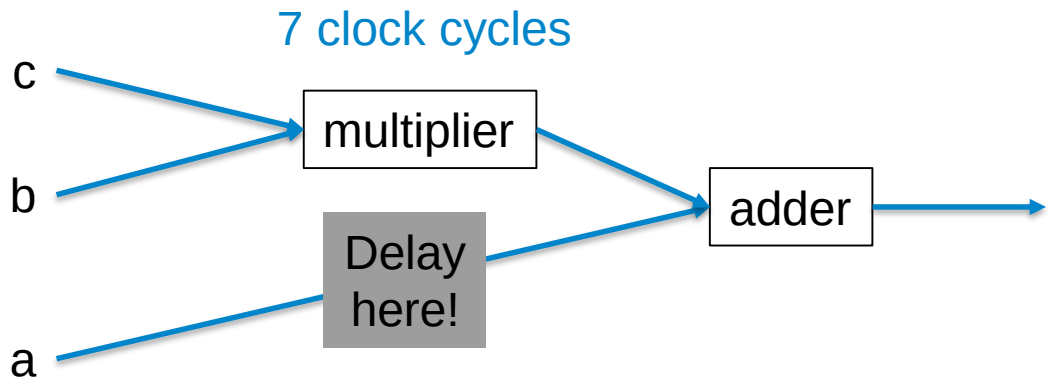
https://github.com/dawsonjon/fpu/blob/master/double_multiplier/test_bench.v

Imperial College London



Imperial College London

Scheduling



Imperial College London

FPGA Programming with HLS

	CPU	GPU	FPGA Sim	FPGA Hw
Engineering time	short	medium	medium	medium
Compilation time	short	short	short	Very long
Debugging	easy	medium	medium	medium
Maintainability	easy	medium	medium	medium

Imperial College London

HLS

- Advantages:
  - simplify programming
  - reduces engineering time
  - Allows “rapid prototyping”
- Disadvantages:
  - Nowadays HLS still achieves less performance than hand-written optimized Verilog/VHDL

Imperial College London

Performance

- HLS still requires significant engineering effort depending on the size of the problem
- Hardware builds still take significant amount of time
- What if the FPGA does not perform as hoped?
- Reengineer the problem and start from scratch?
- Optimize the code a build several times?

Imperial College London

Performance Modelling

Imperial College London

Objective

To understand the major factors that dictate performance when using compute co-processors

- Bandwidths and latencies of the various interconnects
- The capacities of different memories
- Impact of data movements
- Accelerator specific properties, e.g., for FPGA
  - Hardware/arithmetic operations space
  - Clock frequency
  - Fine-grain tunable numeric operations and their impact
  - Other custom computing approaches, e.g., compression

Accelerator compute capabilities

Imperial College London

FPGA Performance

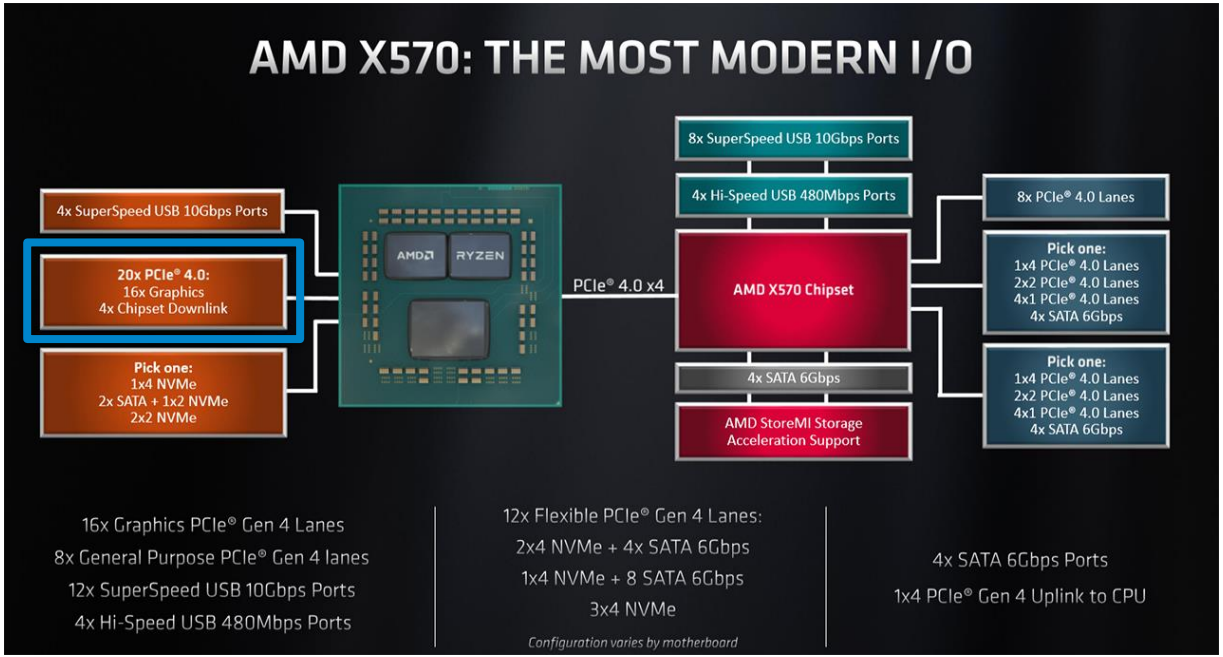
- FPGA Performance is predictable
- There is no context switch, garbage collector or any background process
- The bitstream will be executed the same number of clock cycles every time
- The number of clock cycles needed can be computed easily

Further read: Nils Voss et al. (2021), On Predictable Reconfigurable System Design

https://doi.org/10.1145/3436995

Imperial College London

Modern AMD PC Architecture



https://hexus.net/tech/features/mainboard/131789-amd-ryzen-3000-supporting-x570-chipset-examined/

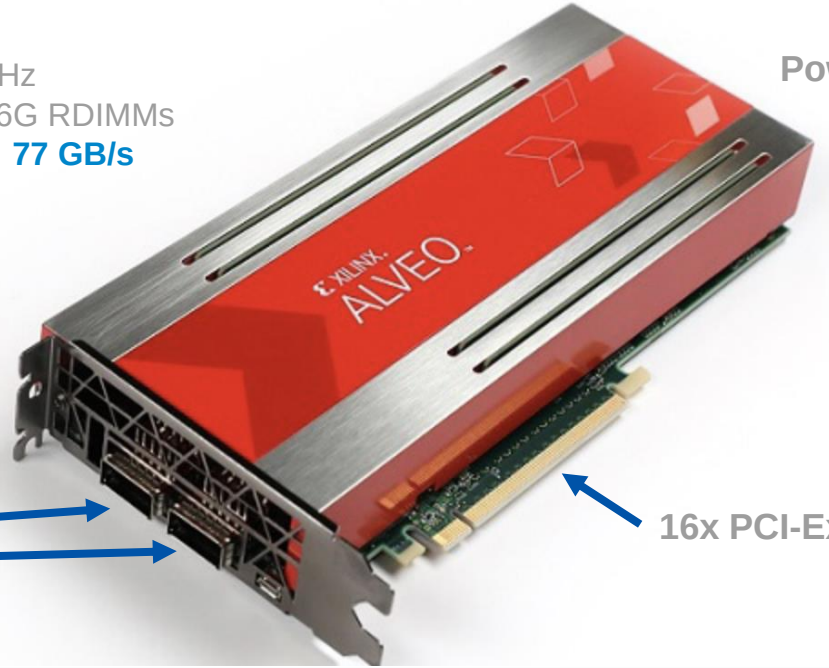
Imperial College London

Xilinx Alveo U250 Data Center Accelerator Card

64 GB DDR4 @600MHz
4 channels of 64-bit 16G RDIMMs
available bandwidth: 77 GB/s
or 2,400 MT/s

Power = 225W

54MB On-chip Memory
available at 38 TB/s

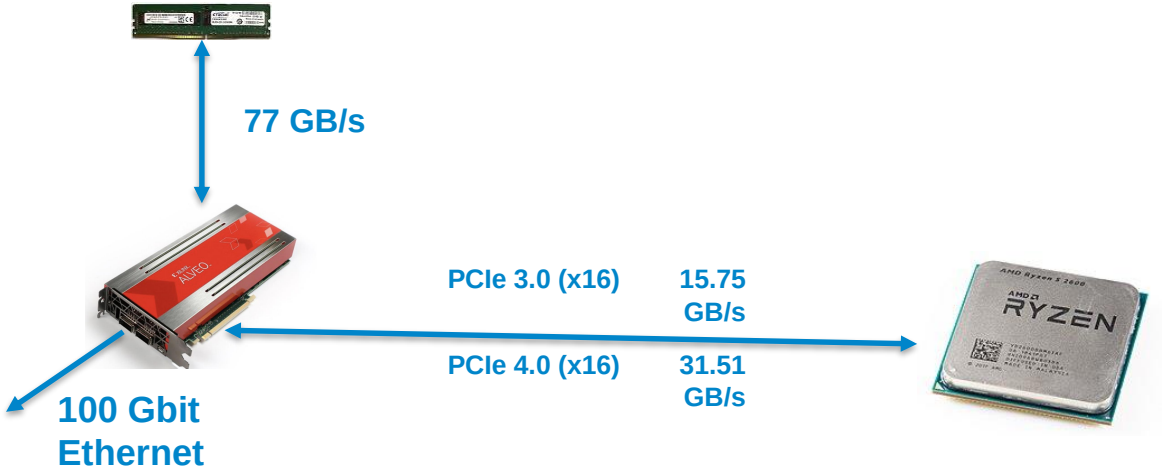


100 Gbit Eth
(QSFP28) x 2

16x PCI-Express

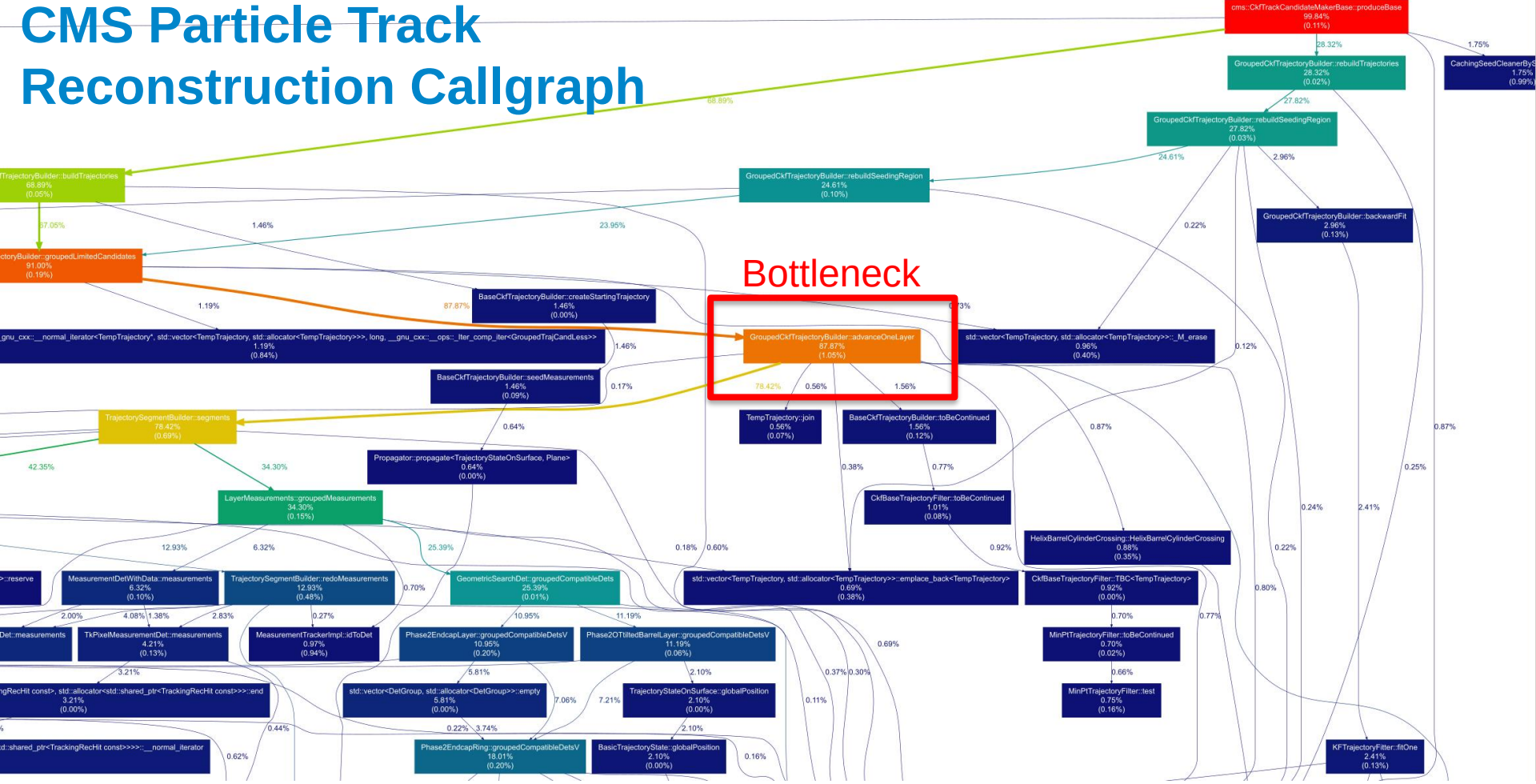
Imperial College London

Modern System Architecture



https://en.wikipedia.org/wiki/DDR4_SDRAM https://en.wikipedia.org/wiki/List_of_AMD_Ryzen_processors https://www.xilinx.com/products/boards-and-kits/alveo/u250.html

CMS Particle Track Reconstruction Callgraph



Imperial College London

1. If compute bound -> optimize the algorithm, increase clock frequency
2. If I/O bound, find a strategy to reduce the I/O (caching, compression accelerate other portion of the code)
3. Repeat until theoretical limit reached

Imperial College London

Performance Model example

Data Types		
Integer	32	bits
Floating Point (24bit mantissa)	32	bits
Float Type	32	bits
Complex Float	64	bits
Fixed Point (27b)	32	bits
Complex Fixed Point	64	bits

		No. DSPs for Operation			
Type	Multiply	Divide	Exponential	Square-root	
Integer	2	0	1	0	
Double (D)	2	0	15	0	
Complex<D>	8	0	60	0	
Fixed Point (FP)	2	0	1	0	
Complex<FP>	8	0	4	0	

Board Parameters

	Board	
Field	VU9P	VU13P
FMEM size (MB)	43.2375	56.8125
DSPs	6840	12228
DSPs (80% utilisation)	5472	9782
LMEM to FMEM data rate (MB/s)	45000	45000
Clock frequency (Hz)	2.00E+08	200000000
Clock cycle time (s)	5.00E-09	0.000000005

CPU Parameters

CPU Type	i7-8700 @ 3.2GHz

NUFFT Benchmarks					
Type	x Dimensions	y Dimensions	b Dimensions	F Dimensions	Time
1	415*103	415*103	415*103	320*320	0.00258800
2	415*103	415*103	320*320	415*103	0.00247200

Imperial College London

Performance Model example

Algorithm Parameters

Assumptions:

1. Single value variables in methods (e.g. loop counter) are ignored (might add in later)
2. Negligible time for the CPU orchestrating tasks performed on the DFE

	Matrix	Data-Type:Dimensions	# Elements	Memory (MB)	Notes	# FMEM Memory Size	Time to Load into FMEM
Inputs	kc	complex:[nz, ntviews, nc]	977256	7.818048	Central profiles of stack-of-stars k-space	1	1.74E-04
Inputs	k	complex:[nx, ntviews]	344865	2.75892	k-space trajectory	1	6.13E-05
Inputs	w	double:[nx, ntviews]	344865	1.37946	Density compenation weights	1	3.07E-05
Inputs	kdata	complex:[nx, ntviews, nc]	2758920	22.07136	k-data	1	4.90E-04
Inputs	b1	complex:[ncs, ncs, nc]	819200	6.5536	Coil sensitivity maps	1	1.46E-04
Calculated	zip	double:[ntviews, ntviews, nc]	5524488	22.097952	Z-Index projection profiles, calculated by generateZProjectionProfs	1	4.91E-04
Calculated	index	int:[ntviews]	831	0.003324	Tells us the order of spokes, calculated by respMotionDetection	1	7.39E-08
Calculated	ku	complex:[nx, nline, ntres]	341960	2.73568	Sorted k using index	1	6.08E-05
Calculated	wu	double:[nx, nline, ntres]	342016	1.368064	Sorted w using index	1	3.04E-05
Calculated	kdatau	complex:[nx, nline, nc, ntres]	2736128	21.889024	Sorted kdata using index	1	4.86E-04
Calculated	recon_cs	complex:[ntres, ncs, ncs]	819200	6.5536	Output image data	1	1.46E-04

Imperial College London

Performance Model example

Time Estimate

The following time estimates are computed by modelling the individual steps and calculating a time for each **(see 'Algorithms' tab)**
The time estimate corresponds to the reconstruction of one slice only and **no** preprocessing steps are considered
Timing is measured/computed for the worst case performance of the algorithm (no early stopping for backtracking line search or the conjugate gradient method)

Number of NUFFT calls

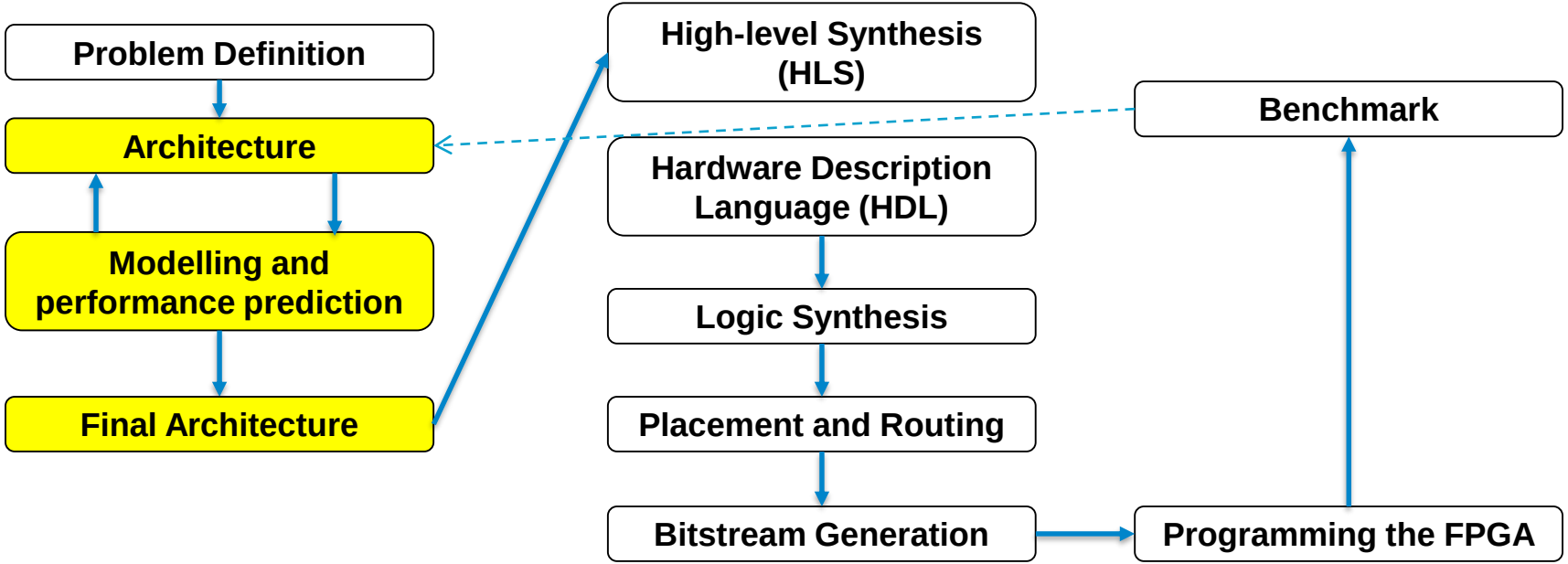
	Type 1	Type 2
mtimesAdj	64	0
grad	64	64
objective	0	64
TOTAL	704	5760
Ratio	11	90

	Time				Time (Observed) (FPGA runs only)	
Setup	Total	NUFFT	Non-NUFFT	Prop'n in NUFFT	Total	
DFE + NUFFT on CPU	16.07058	16.06067	0.00991	99.94%		
All on CPU (C++)	24.72233	16.06067	8.66165	64.96%		580.2363981
All on CPU (MATLAB)	65.118		N/A			220.2892267
						2.633975373
CPU + NUFFT on GPU	18.43535	9.773696	8.66165	53.02%		
DFE + NUFFT on GPU	9.78360	9.77370	0.00991	99.90%		4.052000805

Setup	Total time (s)
DFE + NUFFT on CPU	16.07058
All on CPU (C++)	24.72233
All on CPU (MATLAB)	65.118
CPU + NUFFT on GPU	18.43535
DFE + NUFFT on GPU	9.78360

Imperial College London

FPGA Design Flow (simplified)



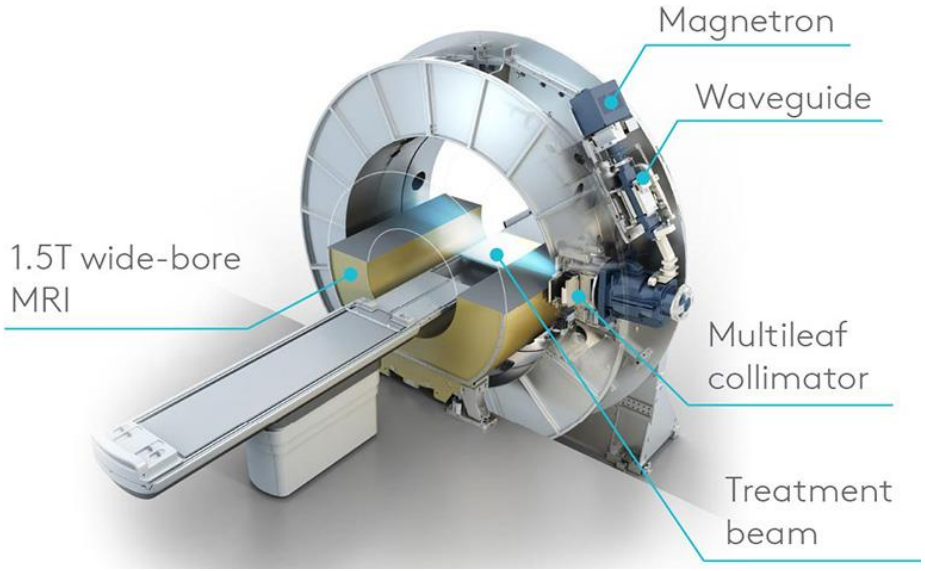
Imperial College London

Current projects

Imperial College London

MR-Linac

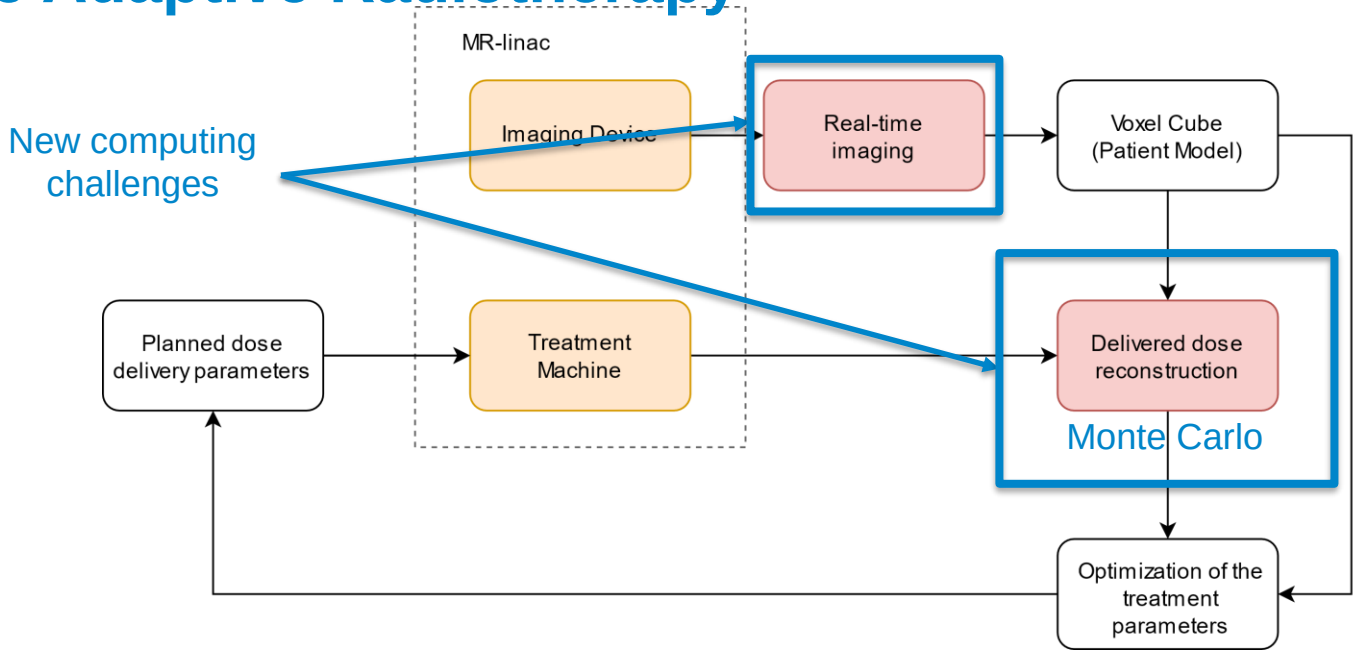
combines an MRI scanner and a linear particle accelerator



Source: https://www.icr.ac.uk/news-features/mr-linac

ICR The Institute of Cancer Research | Imperial College London

Real Time Adaptive Radiotherapy



Imperial College London

Expensive

10^8 particles needs to be simulated to achieve statistical significance

The stochastic behaviour breaks the pipeline of modern CPUs

Not suitable for GPU due to the lack of branch-predictors

Random memory accesses due to many lookup tables

Modern CPU and GPU implementations take more than 1 minute (Ziegenhein2015)

Other cloud-based approaches have reliability concerns (Ziegenhein2017)

Source: https://iopscience.iop.org/article/10.1088/0031-9155/60/15/6097/meta https://iopscience.iop.org/article/10.1088/1361-6560/aa5d4e/meta

Imperial College London

Worst-case for branch predictors

Assuming rng uniformly distributed:

```
if(rng.getRandom() < 0.5) then f(x) else g(y)
```

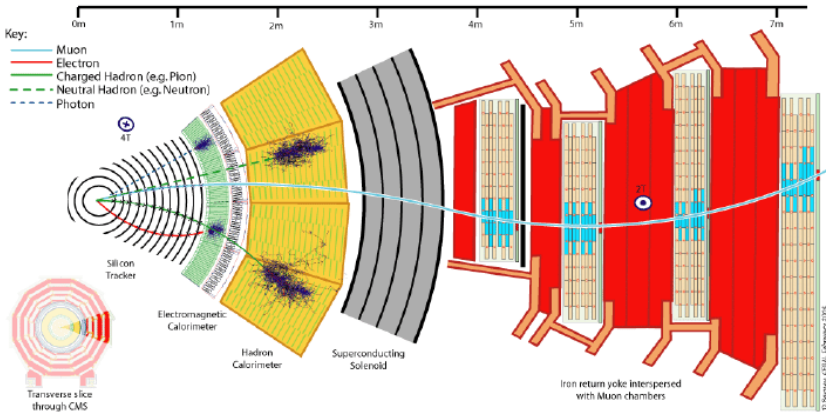
In the GPU case branches can significantly affect the instruction throughput by causing threads of the same warp to diverge.

The diverging branches are de-scheduled!

Imperial College London

CMS Particle Track Reconstruction

On-going project that aims to accelerate track reconstruction using FPGA accelerators (Våge, Liv)



https://www.researchgate.net/publication/260003686_Empirical_Bayes_unfolding_of_elementary_particle_spectra_at_the_Large_Hadron_Collider

Imperial College London

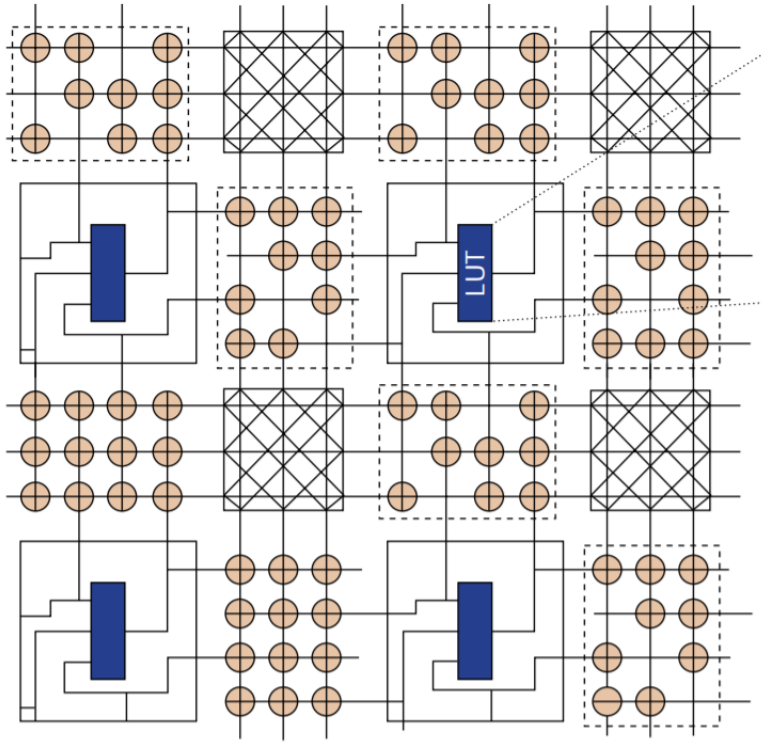
Summary

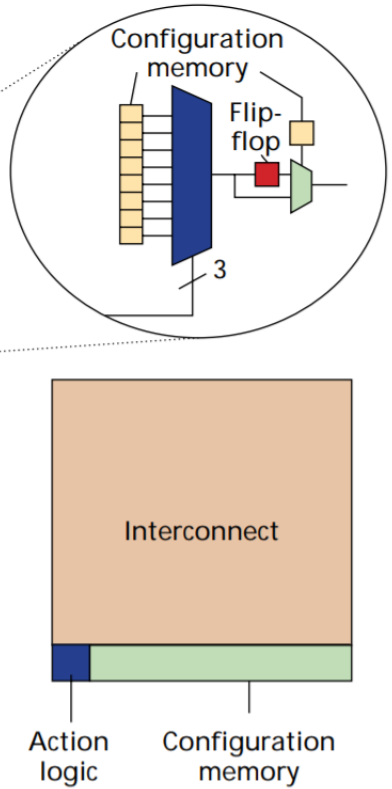
- FPGAs are becoming widely used in industry as accelerators for suitable challenges (ultra low latency inference, graph processing and genomics)
- FPGAs widely used in HEP close to detector (trigger systems)
- Recent toolchain developments making programming FPGAs more accessible
- Potential applications in HEP starting to be studied

Imperial College London

FPGA Architecture (Backup)

Look-Up Tables (LUT) and Switches

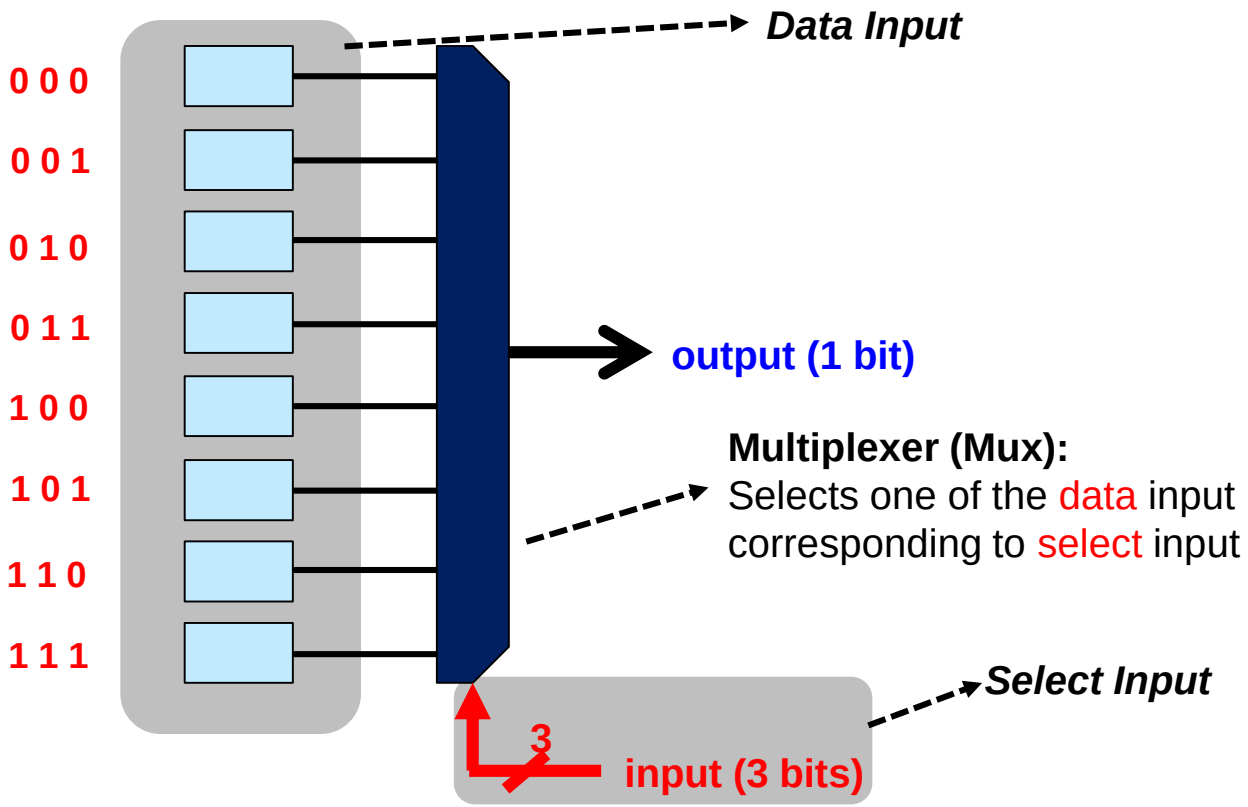




Imperial College London

3-bit input LUT (3-LUT)

3-LUT can implement **any** 3-bit input function



Imperial College London

DSP block

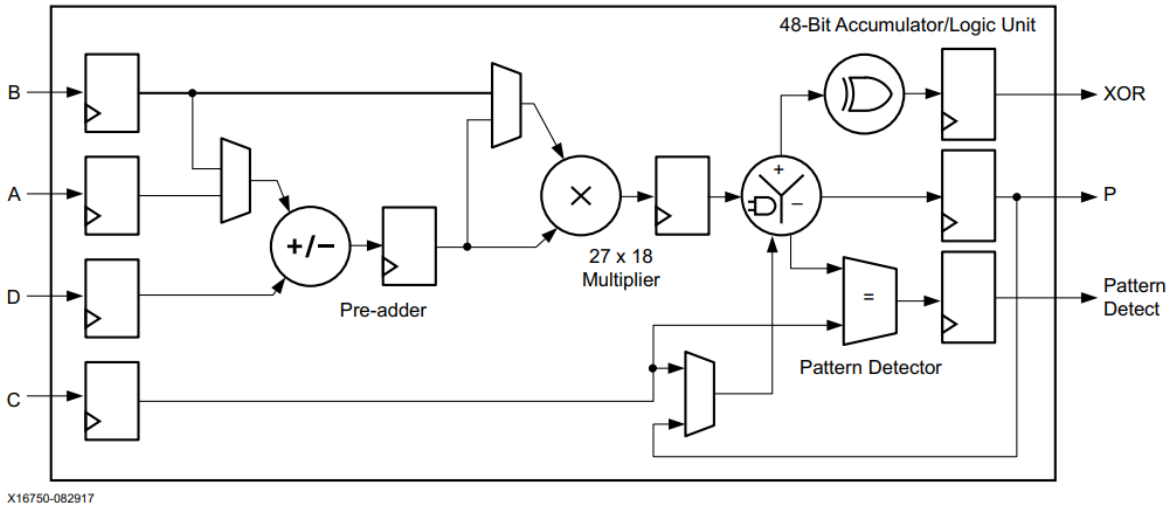


Figure 1-1: **Basic DSP48E2 Functionality**

Imperial College London

Example

$f(g())$

Imperial College London

ALGORITHM 1: Iterative refinement using the methodology (Numbers and letters in brackets do not describe specific transition but only refer to the different parts and steps described above).

```
1  begin
2    perform initial analysis of original application (1)
3    create initial partitioning in hw and sw (a)
4    create first initial software model, performance model and architecture (2, 3, 4)
5    while bottleneck exists
6      identify bottlenecks using performance model (e, 3)
7      resolve bottlenecks with new architecture or hw/sw partitioning (4, a)
8      verify new architecture in updated sw model (f)
9      perform further analysis as necessary (b, c)
10     refine performance model based on updated analysis and architecture (d, c)
11   end
12   start implementation (g)
13 end
```