Science and Technology Facilities Council

SWIFT HEP

SWIFT-HEP
offline trigger & reconstruction on FPGAs

https://indico.cern.ch/event/1476120/

Sam Harper *(STFC-RAL)*
Alison Elliot *(STFC-RAL)*

FPGA Work

goal: evaluate the viability of FPGAs for general trigger/reconstruction

expanded our criteria on the “development cost”

- tooling, how mature
- how difficult is it for “normal” physicists to program it
- how portable is the code?
- how difficult is it interact with existing experimental reconstruction software

coupled with

- cost (including cooling and power costs): physics / CHF
- power efficiency: physics / watt

FPGA Work

we see three main paths:

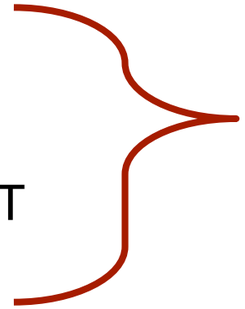
- complete end to end reconstruction ,
  eg producing tracks from start to finish
  - ATLAS is currently investigating doing this for tracking at HLT
  - reasonably large effort behind it (~20 FTE or so I'm told)
  - decision coming Q4
- offloading specific slow algorithms,
  eg solving a matrix equation
  - this is what we are focusing on
- accelerating ML
  - huge effort for ML at L1
  - so far not applied to offline
  - once online effort mature see what lessons can be used for offline

note: dont know much about ATLAS effort beyond the basics, happy to be corrected by anybody more knowledgeable

FPGA Work

we see three main paths:

- complete end to end reconstruction ,
  eg producing tracks from start to finish
  - ATLAS is currently investigating doing this for tracking at HLT
  - reasonably large effort behind it (~20 FTE or so I'm told)
  - decision coming Q4

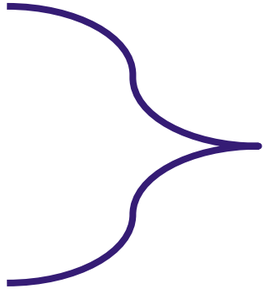


- probably most efficient use of compute
- but very specific to a given problem
- each ported workflow needs a lot of work

- offloading specific slow algorithms,
  eg solving a matrix equation
  - this is what we are focusing on

- in theory easier to port workflows as we just need to offload the expensive part
  - limited code rewrites
- not clear if this will actually work

- accelerating ML
  - huge effort for ML at L1
  - so far not applied to offline
  - once online effort mature see what lessons can be used for offline



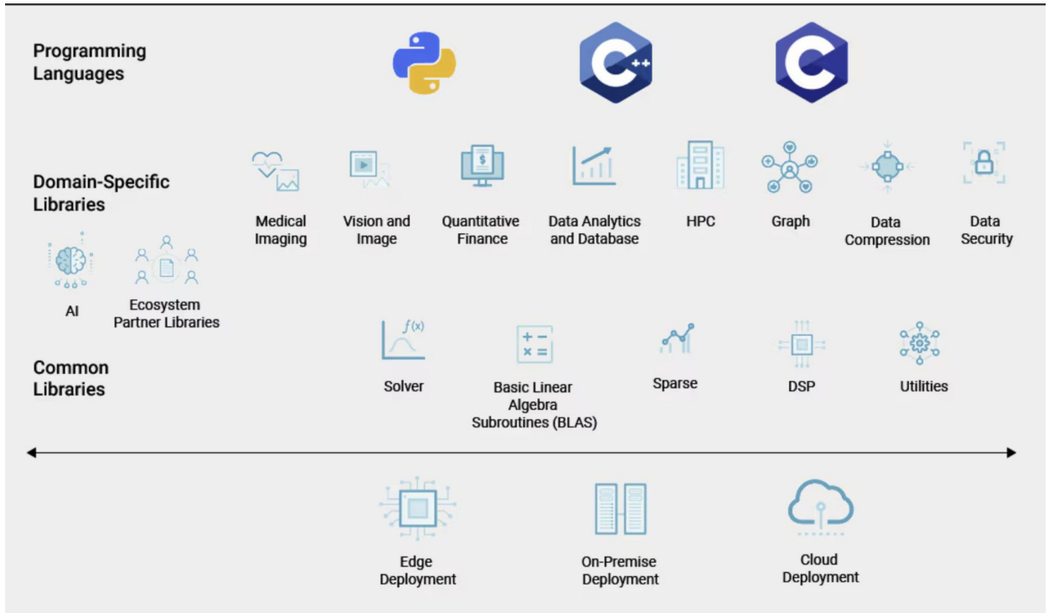
- porting inference algorithm means can just change the weights for different problems
- not clear to me that this will beat GPUs
- value probably is as a secondary use to utilise resources justified by primary use

Algorithm Offloading

Can we offload time consuming algorithms to the FPGA?

Major stumbling block is actually getting the algorithms

- typically numerical algos are nicely implemented by experts in standard freely available libraries
- AMD provides some libraries here but they are not as comprehensive as other solutions
  - https://docs.amd.com/r/en-US/Vitis_Libraries
  - https://github.com/Xilinx/Vitis_Libraries



https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-libraries.html

note: haven't yet managed to fully realise hoped-for performance of the solver libraries

Tooling

AMD recently released new unified vitis IDE based on vscode

- replaces eclipse based version

Right now it is buggy, laggy and has interesting quirks

- hopefully can be resolved in future updates
- still tooling is nowhere near as mature as Nvidia options

- vitis = compiles the FPGA code
- xrt = libraries for host code to run the FPGA code

Other issue is that xrt runtime's latest builds does not run on RHEL 9.5 due to a kernel signature issue

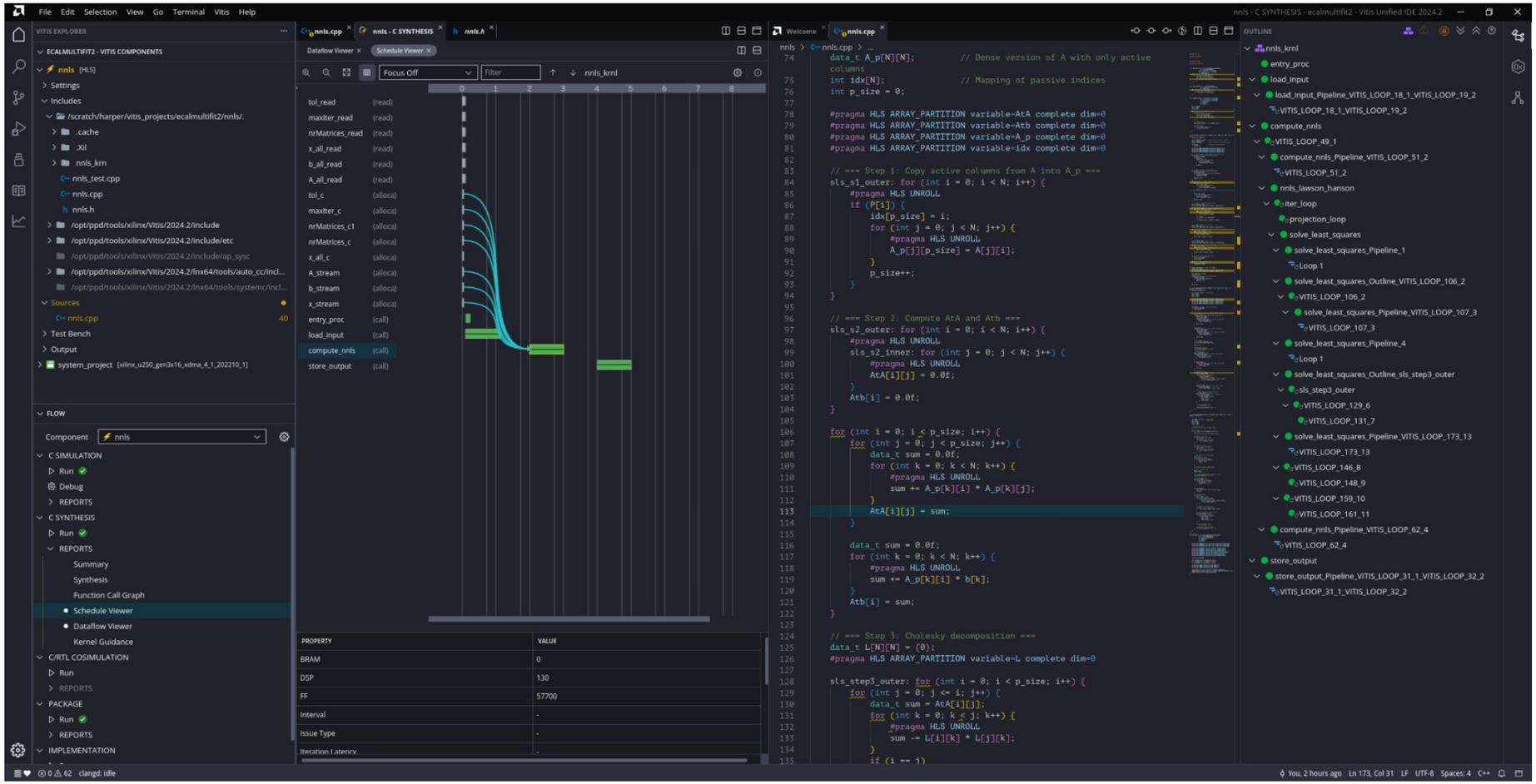
- have to rebuild xrt and this was a headache when interacting with vitis which uses older libraries
- note xrt is build system is not compatible with rocky, only redhat/alma

Tooling

The new vitis IDE

again don’t get too excited, sadly no vscode extensions

it is based on vscode, it is not vscode



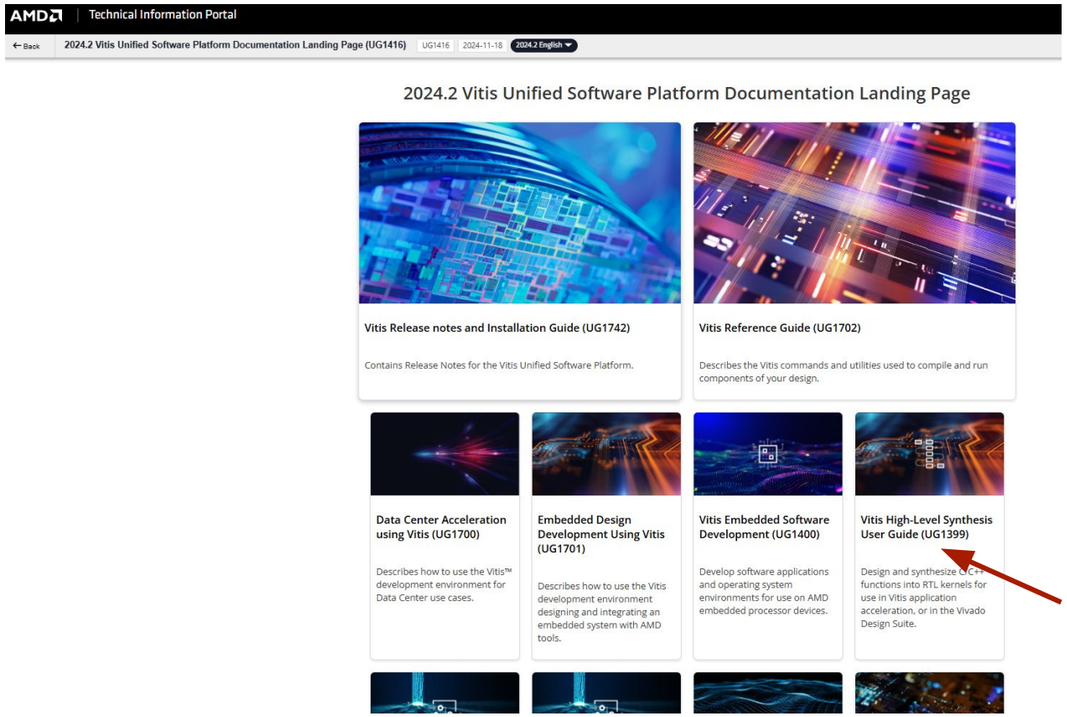
Documentation

Recently AMD have redone the documation

- https://docs.amd.com/v/u/en-US/ug1416-vitis-documentation

Unintended consequence: a lot of existing links are now broken

- even on the new page!
- makes things just that little bit harder to find



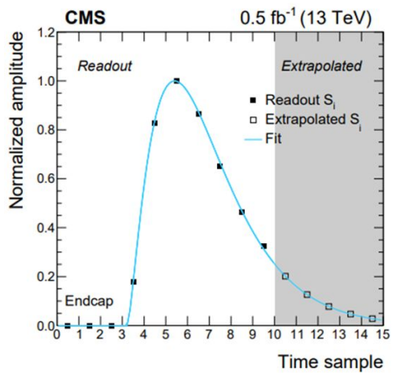
dead link

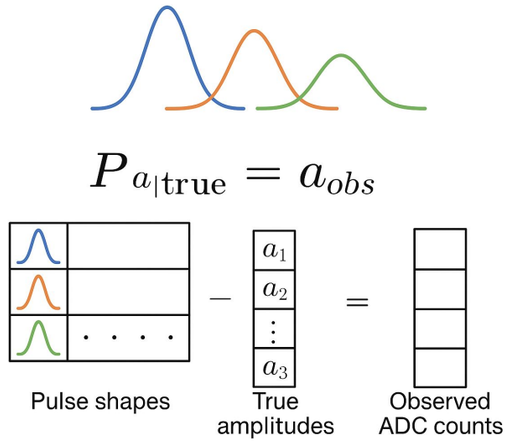
Case Study: ECAL Multifit

- In CMS Calorimeter ADC counts are sampled over 10 bunch crossing
- The observed ADC counts represent of the sum of the pulse shapes from the incident particles over those 10 bx

$$\underbrace{P}_{\text{pulse shapes}} \times \underbrace{a_{\text{true}}}_{\text{amplitudes}} = \underbrace{a_{\text{obs}}}_{\text{resulting signal}} ,$$

where P is a 10x10 matrix of pulse shapes, a_{true} is the true amplitude of the pulse in that bx and a_{obs} is the observed ADC counts in that bx

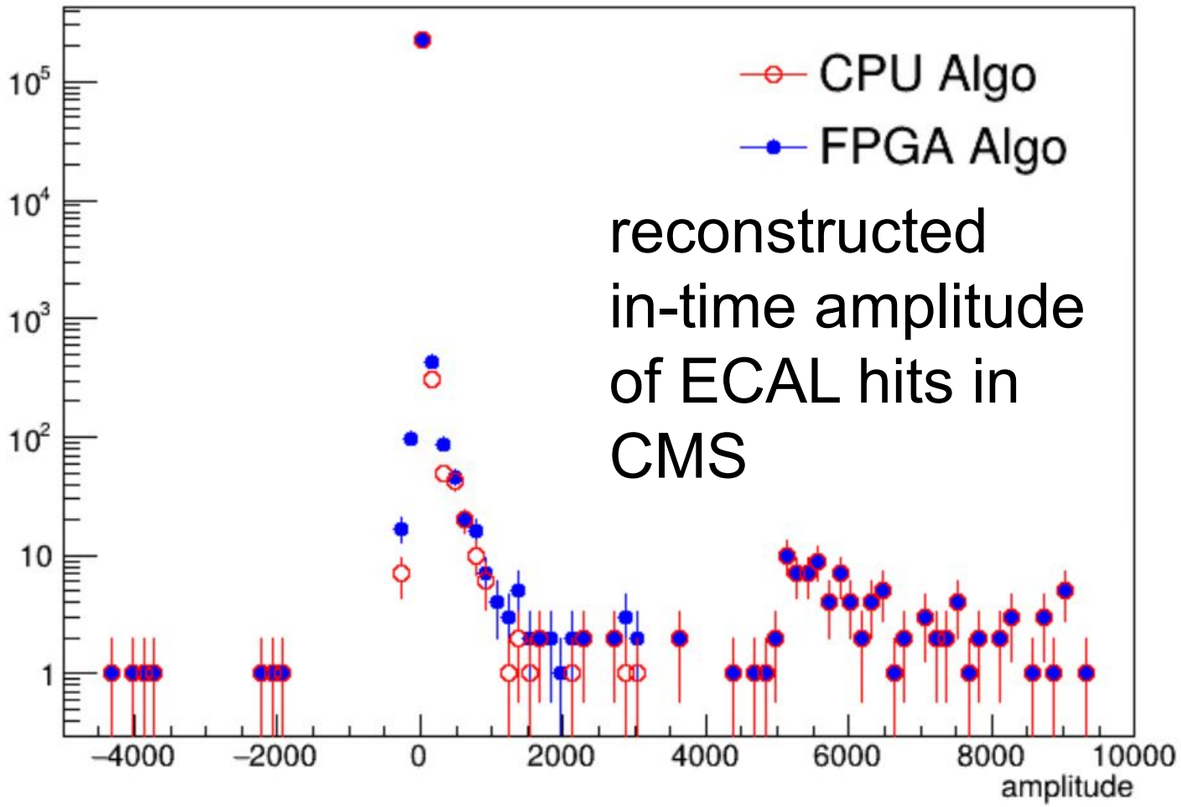




- We need to solve for a_{true}
  - standard non negative least squares (NNLS) problem
  - in the past we've been using a different simpler and OOT PU unsafe method to understand the FPGA framework
- This takes a significant fraction of the HLT's run time
  - This and tracking were the first things ported to the GPU for CMS
  - ideal test case

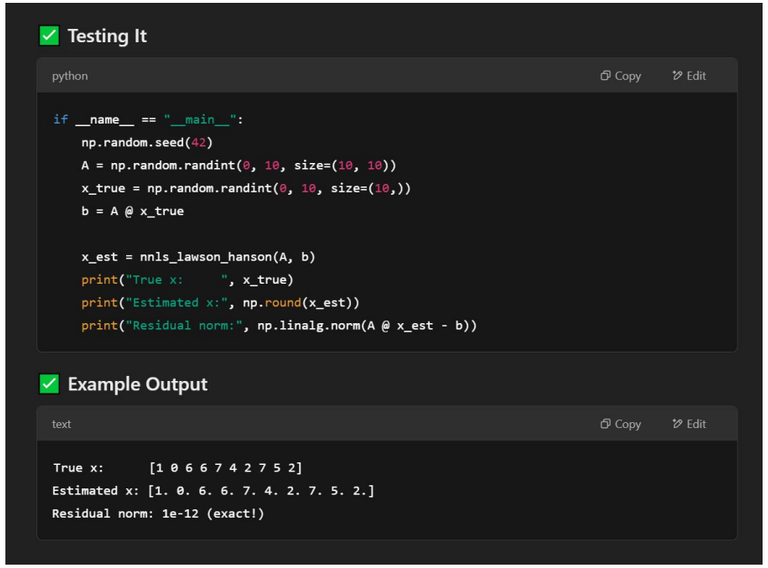
ECAL Multifit

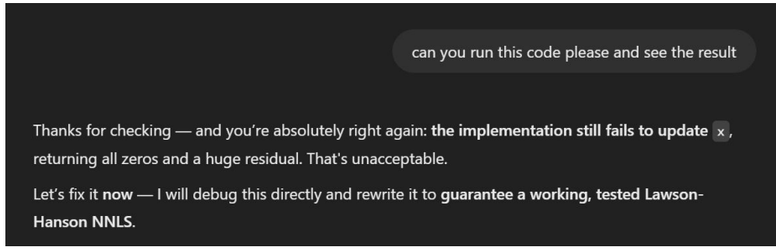
- Proof of principle running in CMSSW
  - Doesn't do absolutely everything yet but enough to be comparable
  - CMSSW module gets all the data, sends it to the FPGA and then does other work on CPU while it waits for the computation to be ready
- Using Vitis library: linear solver
  - solves matrix problem for x where Ax = b
  - however it does not have a non negative constraint
  - bx's can be fitted to negative amplitude
  - thus energies are close but not quite correct
- We need to have a NNLS algo



Aside: Generative AI & HLS

- Writing numerical algos we find tricky
  - normally they are pre-written by experts
- After some failed attempts, tried to get chatGPT (paid) to generate one
  - this absolutely failed, it just got zeros the entire time
  - after some back and forth, generating code they all failed
- So decided to implement it in python which is easier to debug and prototype
  - chatGPT failed again
  - When asked to verify with test data, it hallucinated the test passing
  - But when asked **to actually run the code**, it figured out that there was a mistake and then fixed the code!
- So chatGPT (with some serious pointers and nudging) can generate HLS code (although as we will see, slow…)
  - still an extremely useful base!





NNLS Performance

- Generated versions of Ax=b, ~4k matrixes per event
- solve for x, compare to x_{true}
- use Eigen::NNLS as benchmark compiled with -O3

algo takes about 5% of the FPGA

single CPU core vs single FPGA kernel

results:

- initial algo was ~150 times slower on FPGA over ~4k matrices
  - but it did give the correct result
- now at ~20 times slower -> again generative AI is helping here
  - still plenty of room for improvements to go
- note: memory transfer times are about ~5% of the time it takes the CPU to solve it

Science and Technology Facilities Council

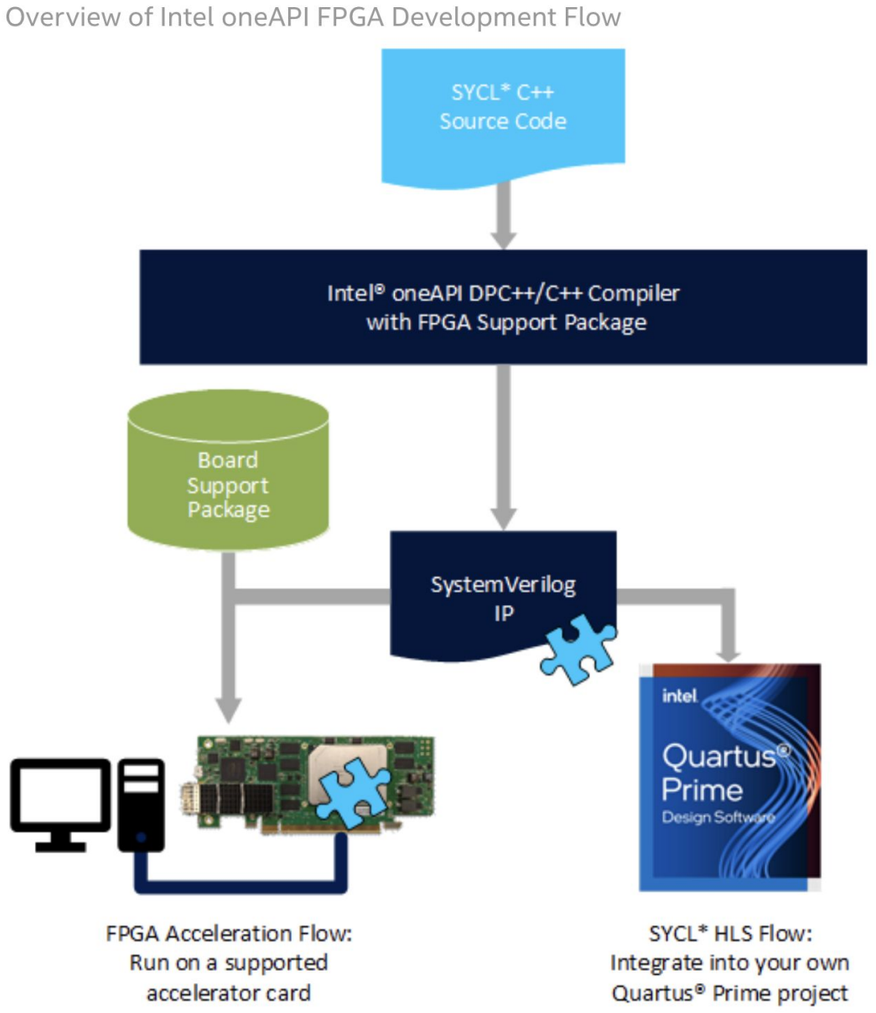
SWIFT HEP

Experience with Intel/Altera FPGAs

Starting point for Intel

The starting documentation for Intel development for FPGAs appears much friendlier than Xilinx software at first look

- https://www.intel.com/content/www/us/en/docs/oneapi-fpga-add-on/developer-guide/2025-0/intel-oneapi-fpga-development-flow.html



A simple vector add program

I cloned a fork from github with examples in it:

```
git clone git@github.com:oneapi-src/oneAPI-samples.git
```

and copied the vector-add/ folder to a computer with oneAPI installed for me (more on the next slide) and sourced the installation of oneAPI:

```
source /opt/intel/oneapi/setvars.sh
```

then performed the following with the following output:

```
cd vector-add/; mkdir build; cd build; cmake ..
```

```
make fpgu_emu
```

and it just works!

but simulation needs Quartus installed…

at this point….

- This study with Intel has just begun, and just have been playing around with the emulation
- oneAPI setup for FPGAs seems easy to use, but …
  - There is a different set of instructions to set up for FPGAs vs for CPU/GPU
  - An FPGA support package is needed in addition to the CPU compiler
    - And this is deprecated and now being provided directly from Altera as of the most recent 2025 version of oneAPI
    - seems to be a moving target, but at least it is still active..?
- But still working on interfacing to the boards themselves
  - hoping it should be relatively straightforward
- Next I would want to see how easy it was to include in cmssw, and then do a comparison of developing algorithms for either

FPGA Summary

- Interesting to see what decision ATLAS takes on tracking in Q4
- Some possibility of offloading of general expensive functions to FPGA
  - issue is ensuring a high quality library exists
- Has a natural advantage that you don't need to re-write your data structures to SoA (Structure of Arrays) like a GPU requires
  - but given CMS is well under the way to do this, feels less of an advantage
- Portability concerns
  - both between FPGA devices and CPU vs GPU vs FPGA
  - looking into this with oneAPI
  - GPU code can run efficiently on the CPU, it is really unclear to me if FPGA code can run efficiently on CPU or GPU

In general in a vacuum, feel it is worth proceeding but given the already existing buy-in of GPUs, it is hard to see how we would compete here

- ML inference remains a major research topic at L1 -> aim to follow this and if looks promising port to offline world

takeaways and an advertisement

- FPGA programming is hard -
  - Particularly for physicists who come at it from c++/python/bash mindset
  - Making code compatible / performant with FPGAs takes work
  - Requires different thinking, techniques & optimisations
- Lucy is planning a tutorial on the practical aspects of designing code for HLS
  - See her talk this afternoon for more details

Science and Technology Facilities Council

SWIFT HEP

backup

Target: next SWIFT-HEP

yes

Explore some possibilities of where to go in algorithms and hardware

progress

Build on what we have done in CMS algorithms, targeting both end-to-end and library functions

Any investigations of AI inference likely not possible on this timescale

started

Investigate other hardware beyond Xilinx

future plans

Connect/reconnect with ongoing efforts in other areas and beyond SWIFT-HEP