Reco Status

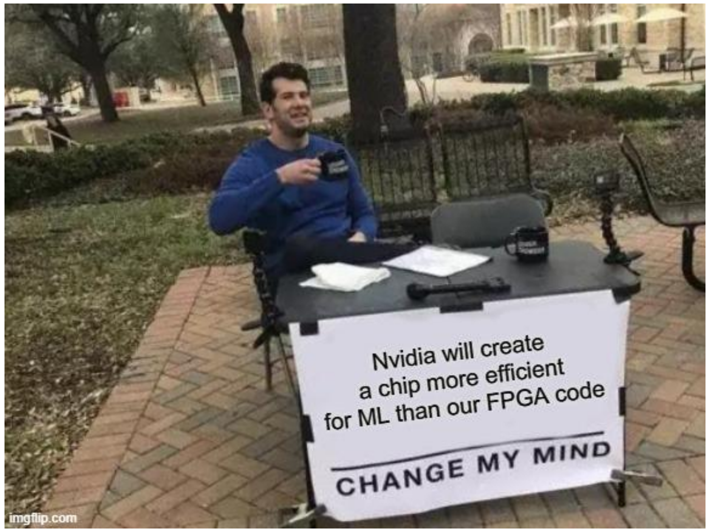
Alison Elliot (RAL), Sam Harper (RAL)

SwiftHEP Meeting

March 27th 2024

Introduction

- Aim is to investigate uses of FPGAs to accelerate “offline” reconstruction
  - HLT reconstruction and offline reconstruction are the same to me, particularly from a code point of view
- There are two ways you can go here
  - FPGA to accelerate targeted c++ functions
  - FPGA to accelerate ML inference
- Currently focusing on seeing if FPGAs can accelerate c++ functions
  - while ML is an approach, it does rather feel we’re competing with a multi billion dollar research effort from the major tech companies
  - depends on how cutting edge we are
    - fixed silicon may not be optimised for latest algos
    - also how many different ML algos we use


Nvidia will create a chip more efficient for ML than our FPGA code
CHANGE MY MIND
imgflip.com

Disclaimer

- Alison and myself are physicists not firmware experts
- part of the goal here is to see if physicists can write accleratable code
  - maintainability issue if only a handful of (difficult to retain) firmware engineers can write/understand the reco code
  - in the next step we will be talking to RAL firmware experts to better utilise the FPGA but again idea is to have this physicist understandable
- this talk represents our current understanding which is continuously evolving
  - please let us know if its incorrect or you have some suggestions for improvements

Xilinx (AMD) Tools

https://docs.amd.com/r/2021.1-English/ug1399-vitis-hls/Getting-Started-with-Vitis-HLS
https://github.com/Xilinx/Vitis-Tutorials

- **vitis**: builds the fpga binary
  - both command line and GUI options
  - latest GUI version is now vscode based! (old was eclipse ☹)
  - can be done on any machine
- **XRT**: runs the fpga binary
  - handles the calls in the host program
  - library host program links against, apache 2.0 licenced
    - therefore need version compiled with correct gcc & arch
    - will need multiple versions for athena / CMSSW / whatever
  - there has been a problem in that it was hardcoded to install at /opt/xilinx/xrt
    - worked around it by installing at /opt/xilinx/xrt_slc7_amd64_gcc11/opt/xilinx/xrt/
    - latest branch appears to have a fix for this and appears to be able to be installed anywhere
      - https://github.com/Xilinx/XRT/pull/7835
      - which is necessary for us!
  - supports opencl and proprietary xrt api:
    - currently using the xrt api , haven't got opencl working yet

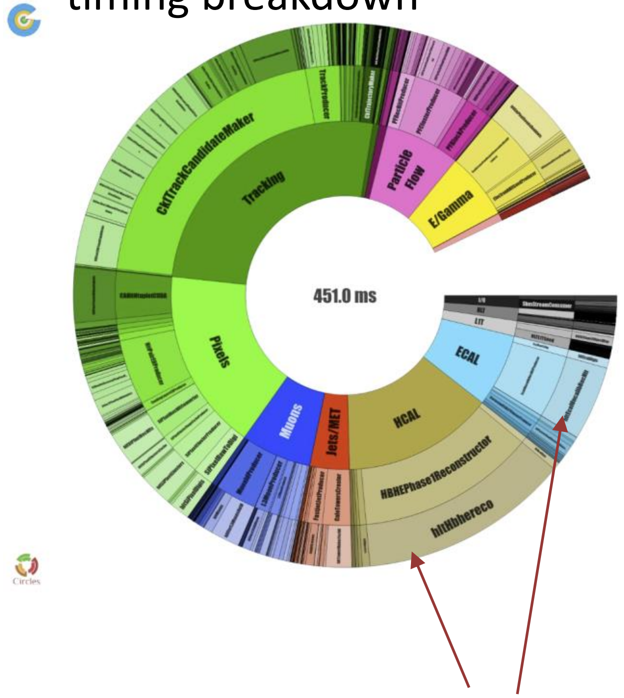
Setup

- Alveo U250 data center card
  - https://www.xilinx.com/products/boards-and-kits/alveo/u250.html
  - note: algos presented in this talk typically take ~1-2% of the FPGA
  - power is ~25-30W (basically idle), max is 225W
- server details:
  - 2x Xeon Gold 6242R (20 cores 40 threads each) at 3.1GHz
  - 192 GB ram
  - centos 7 , using gcc 11
- XRT: 2022.2
  - compiled with gcc 11 to enable linking with CMSSW
  - custom compilation is a little fiddly but not too bad
    - mainly getting cmake to pick up gtest, protobuf , boost libraries also compiled with gcc11
    - cmake experienced users may have an easier time

Strategy

- reproduce the CMS GPU effort
  - in particular focus on the ECAL reco algos
  - contrary to general GridPP perception, CMS GPUs efforts are highly advanced and are an everyday thing now
- two methods:
  - **weights**: simple algo which sums observed amplitudes with given weights to estimate the energy in the BX
    - simple multiplication
    - aim to gain some basic experience in FPGA algos
  - **multifit:** fits the observed amplitudes to predefined signal templates to estimate the energy in each bunch crossing
    - standard least squares fit
    - takes significant part of the HLT processing time

CMS HLT Run3 projection timing breakdown



ECAL and HCAL multifit

Workflow Cycle

two main steps: building and linking

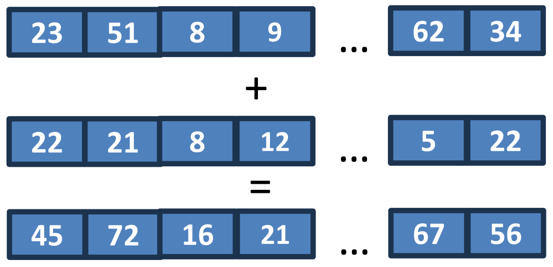
- building:
  - compiles the HLS code into a xilinx object file (.xo)
  - for simple functions, takes ~few mins (both hw and hw_emu)
  - gives a nice summary of how long the algorithm is expected to take
    - eg initiation interval, latency estimates etc etc
    - also gives feedback on things to improve to get faster
- linking:
  - makes a binary to run on the FPGA (.xclbin)
  - takes ~10mins for hw emulation
  - takes ~2hrs for hw build
    - so vitally important you test it with the hw_emu run that it does what you want first

General Observations

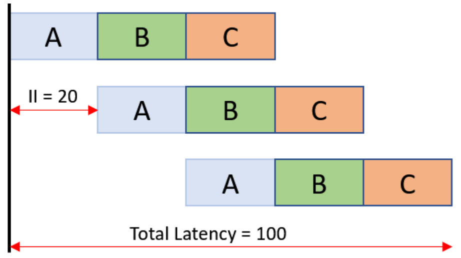
- simple to get code to synthesis to the FPGA
  - as long as it doesn’t do anything to crazy and has no dynamic memory allocation
- to be performant, HLS is still fairly low level, yes its c++ code but you have to spend a lot of time thinking about loop unrolling, loop pipelining, DSPs to gain efficiency
  - none of this is very straightforward
  - while emulation is useful for testing and getting an idea of speed up, to actually get a measurement you need a 2hr+ build which is annoying
- error messages can be unhelpful and sometimes buried
  - simulation output was being buried in a hidden directory we missed at first

Architectures

- GPU:
  - massively parallel, single instruction , multiple data
  - needs careful alignment of the memory
    - structure of arrays (SoA)
  - more about organising the problem so the exact same operation can operate an memory block



- FPGA:
  - flow of data through a circuit, everything happens at once
  - takes a given latency to pass through the circuit
  - achieves parallelization through pipelining
    - ie how many events you can send through the circuit at once
    - the number of cycles before new data can enter the pipeline is known as the **initiation interval (II)**
    - best is 1 cycle, so if it takes 150 cycles to get a result for 1 event, it'll take 151 cycles for two events, 152 cycles for three events etc etc
    - keeping that pipeline full is key to performance
  - can also easily create N instances of a kernel
    - assuming have enough resources
    - while useful, not the best way to achieve parallelization in an FPGA algo as it is inefficient in resource usage

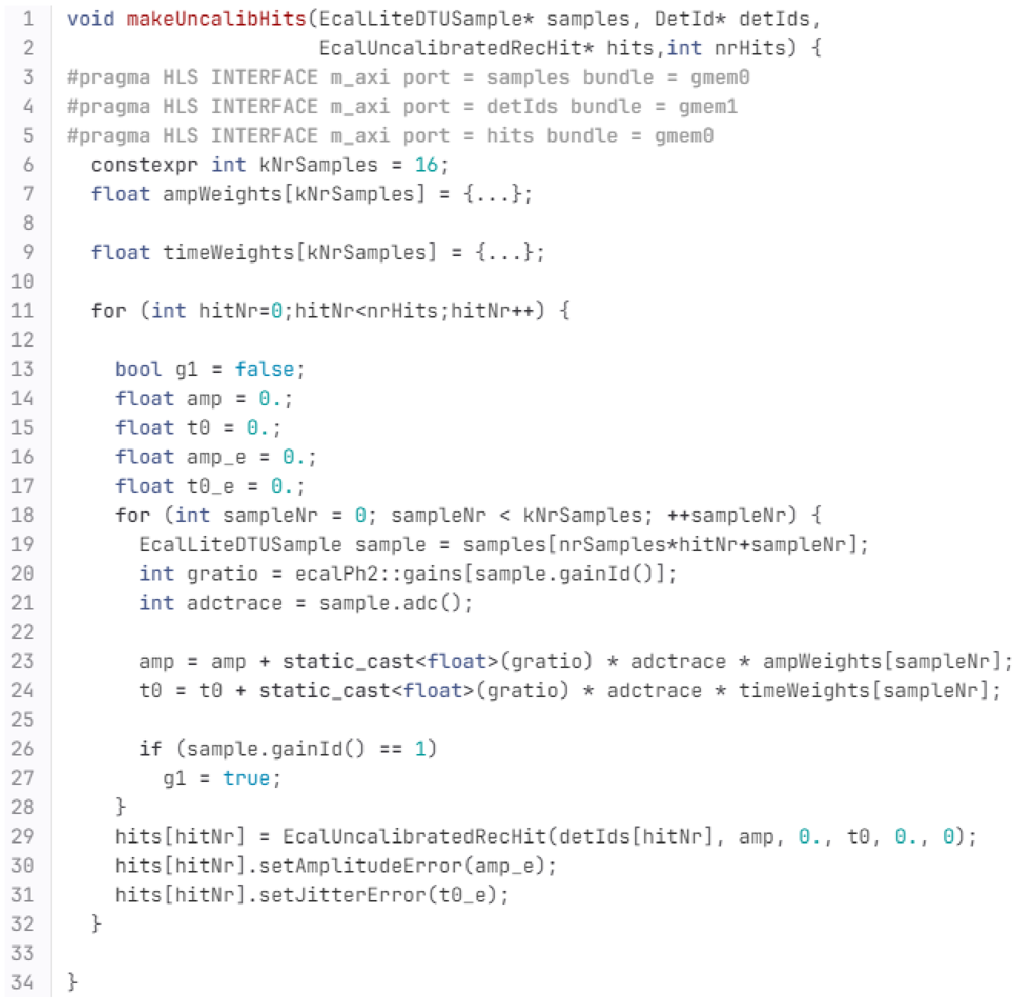


Native CMSSW Datatypes

- HLS can synthesis native CMSSW dataformats
  - as long as they are not too complex
  - pointers is the main stumbling block, typically don't use those in CMSSW dataformats
    - ironically except the low level hardware IO formats which are other probably the easiest for a FPGA to deal with
- Ecal Weights algo updated to write the legacy CMSSW dataformat directly
  - no need for AoS -> SoA -> AoS conversions like GPU
  - well maybe no need, see how we do performance wise, a definitive statement there when we have good performance

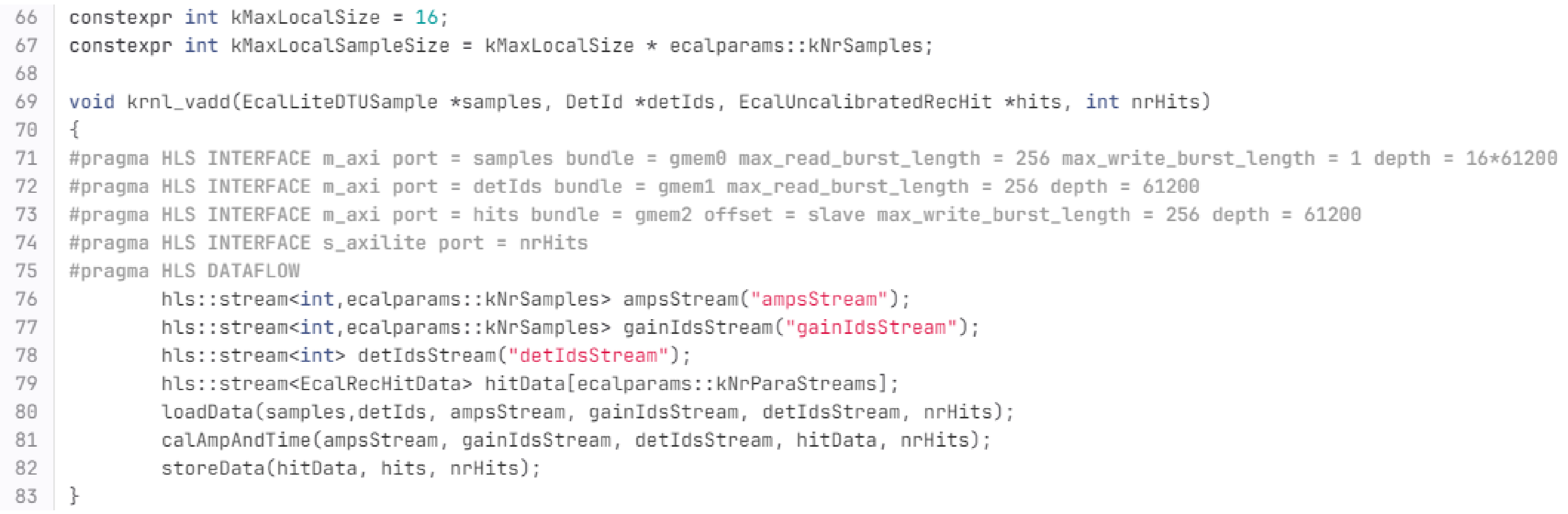
ECAL weights

- In November had a simple algo on the FPGA
  - it sums the 16 samples of the ADC with their appropriate weights to determine hit energy
  - it ran, producing rec-hits on the FPGA
- speed:
  - slow…., it does one event at a time



```
void makeUncalibHits(EcalLiteDTUSample* samples, DetId* detIds,
                     EcalUncalibratedRecHit* hits,int nrHits) {
#pragma HLS INTERFACE m_axi port = samples bundle = gmem0
#pragma HLS INTERFACE m_axi port = detIds bundle = gmem1
#pragma HLS INTERFACE m_axi port = hits bundle = gmem0
  constexpr int kNrSamples = 16;
  float ampWeights[kNrSamples] = {...};

  float timeWeights[kNrSamples] = {...};

  for (int hitNr=0;hitNr<nrHits;hitNr++) {

    bool g1 = false;
    float amp = 0.;
    float t0 = 0.;
    float amp_e = 0.;
    float t0_e = 0.;
    for (int sampleNr = 0; sampleNr < kNrSamples; ++sampleNr) {
      EcalLiteDTUSample sample = samples[nrSamples*hitNr+sampleNr];
      int gratio = ecalPh2::gains[sample.gainId()];
      int adctrace = sample.adc();

      amp = amp + static_cast<float>(gratio) * adctrace * ampWeights[sampleNr];
      t0 = t0 + static_cast<float>(gratio) * adctrace * timeWeights[sampleNr];

      if (sample.gainId() == 1)
        g1 = true;
    }
    hits[hitNr] = EcalUncalibratedRecHit(detIds[hitNr], amp, 0., t0, 0., 0);
    hits[hitNr].setAmplitudeError(amp_e);
    hits[hitNr].setJitterError(t0_e);
  }

}
```

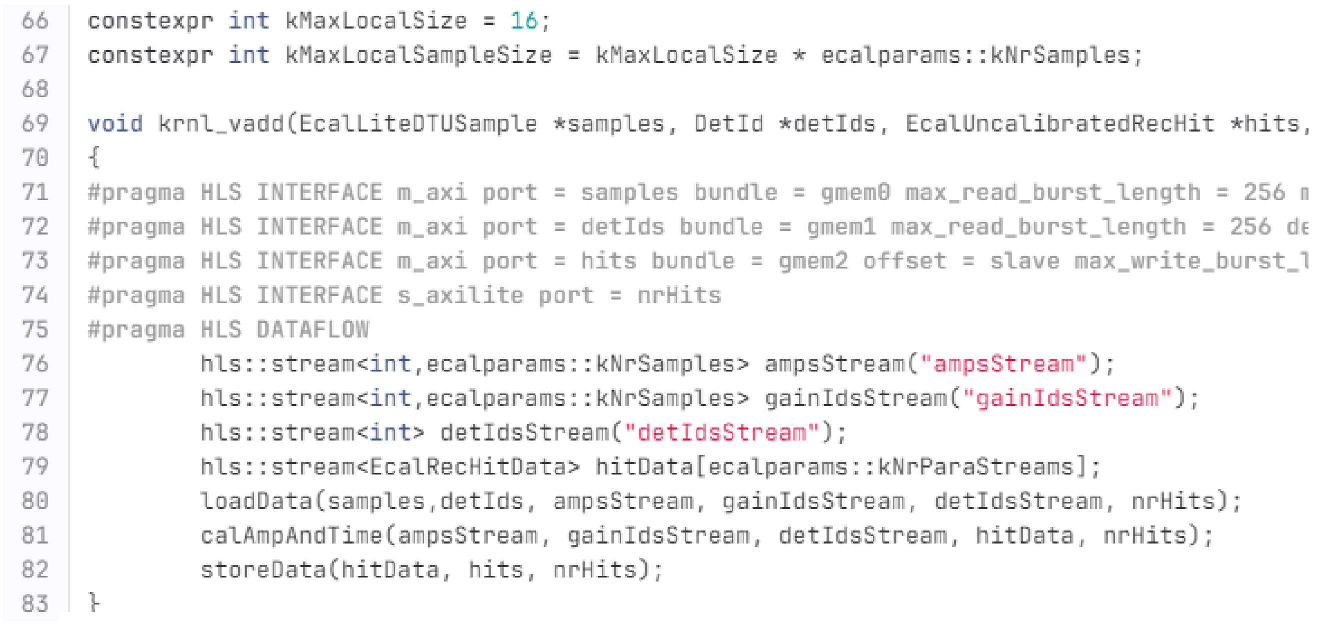
ECAL weights



```
constexpr int kMaxLocalSize = 16;
constexpr int kMaxLocalSampleSize = kMaxLocalSize * ecalparams::kNrSamples;

void krnl_vadd(EcalLiteDTUSample *samples, DetId *detIds, EcalUncalibratedRecHit *hits, int nrHits)
{
#pragma HLS INTERFACE m_axi port = samples bundle = gmem0 max_read_burst_length = 256 max_write_burst_length = 1 depth = 16*61200
#pragma HLS INTERFACE m_axi port = detIds bundle = gmem1 max_read_burst_length = 256 depth = 61200
#pragma HLS INTERFACE m_axi port = hits bundle = gmem2 offset = slave max_write_burst_length = 256 depth = 61200
#pragma HLS INTERFACE s_axilite port = nrHits
#pragma HLS DATAFLOW
        hls::stream<int,ecalparams::kNrSamples> ampsStream("ampsStream");
        hls::stream<int,ecalparams::kNrSamples> gainIdsStream("gainIdsStream");
        hls::stream<int> detIdsStream("detIdsStream");
        hls::stream<EcalRecHitData> hitData[ecalparams::kNrParaStreams];
        loadData(samples,detIds, ampsStream, gainIdsStream, detIdsStream, nrHits);
        calAmpAndTime(ampsStream, gainIdsStream, detIdsStream, hitData, nrHits);
        storeData(hitData, hits, nrHits);
}
```

- now we've pipelined it: ~16 times as fast
  - was hoping for more but will take it
- key gains:
  - adopt load -> compute - > store model
  - use hls::streams to internally buffer from global memory

More on pipelining

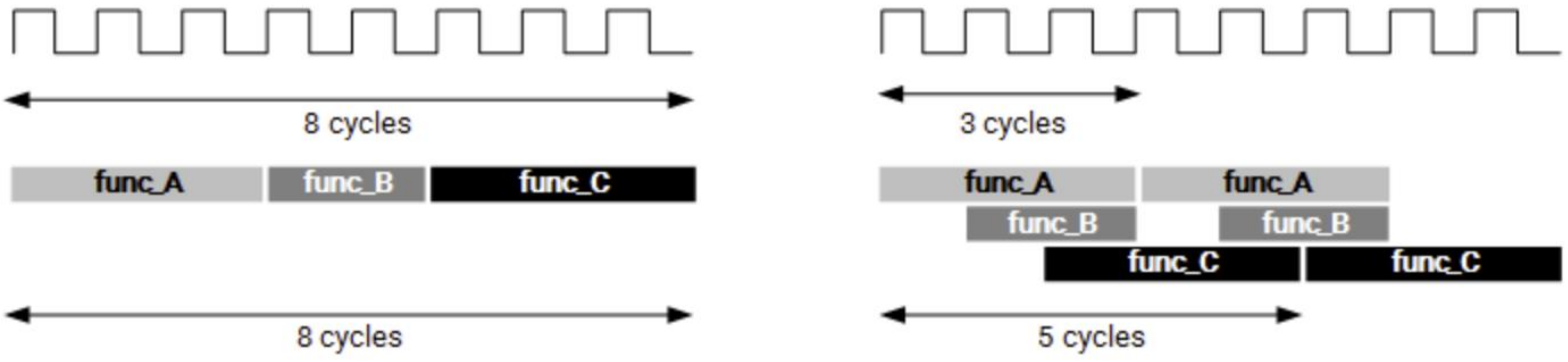


```
constexpr int kMaxLocalSize = 16;
constexpr int kMaxLocalSampleSize = kMaxLocalSize * ecalparams::kNrSamples;

void krnl_vadd(EcalLiteDTUSample *samples, DetId *detIds, EcalUncalibratedRecHit *hits,
{
#pragma HLS INTERFACE m_axi port = samples bundle = gmem0 max_read_burst_length = 256 m
#pragma HLS INTERFACE m_axi port = detIds bundle = gmem1 max_read_burst_length = 256 de
#pragma HLS INTERFACE m_axi port = hits bundle = gmem2 offset = slave max_write_burst_l
#pragma HLS INTERFACE s_axilite port = nrHits
#pragma HLS DATAFLOW
        hls::stream<int,ecalparams::kNrSamples> ampsStream("ampsStream");
        hls::stream<int,ecalparams::kNrSamples> gainIdsStream("gainIdsStream");
        hls::stream<int> detIdsStream("detIdsStream");
        hls::stream<EcalRecHitData> hitData[ecalparams::kNrParaStreams];
        loadData(samples,detIds, ampsStream, gainIdsStream, detIdsStream, nrHits);
        calAmpAndTime(ampsStream, gainIdsStream, detIdsStream, hitData, nrHits);
        storeData(hitData, hits, nrHits);
}
```

#DATAFLOW causes functions to be pipelined

as soon as the detId is used, the next one is loaded

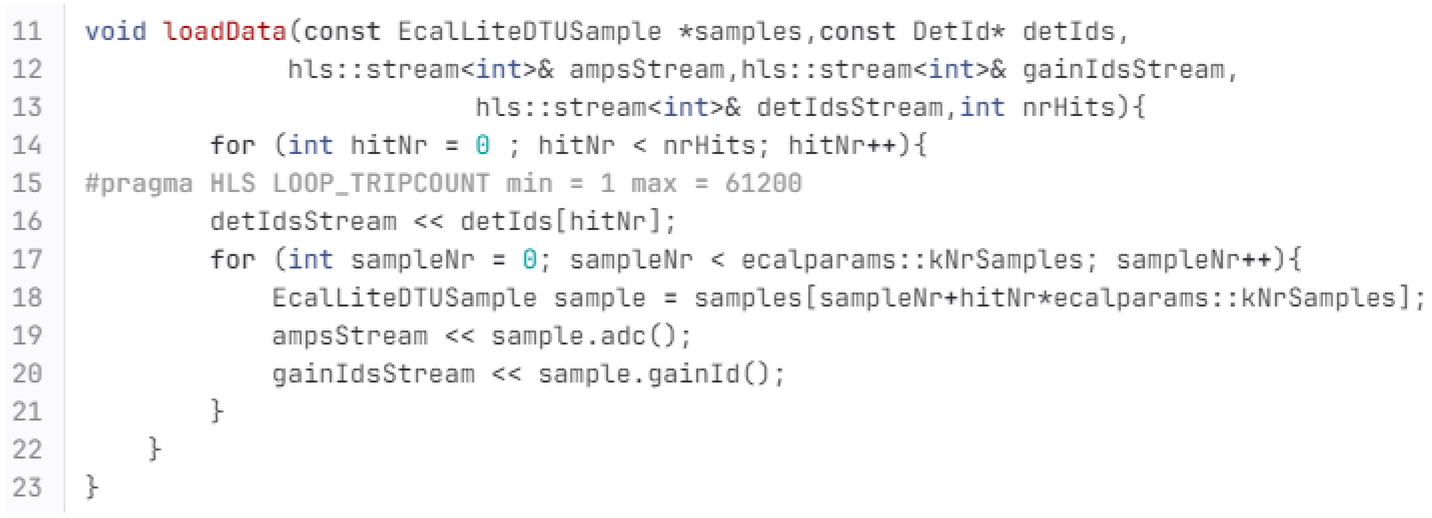
while the hit is being written, the next one is being loaded



(A) Without Dataflow Pipelining

(B) With Dataflow Pipelining

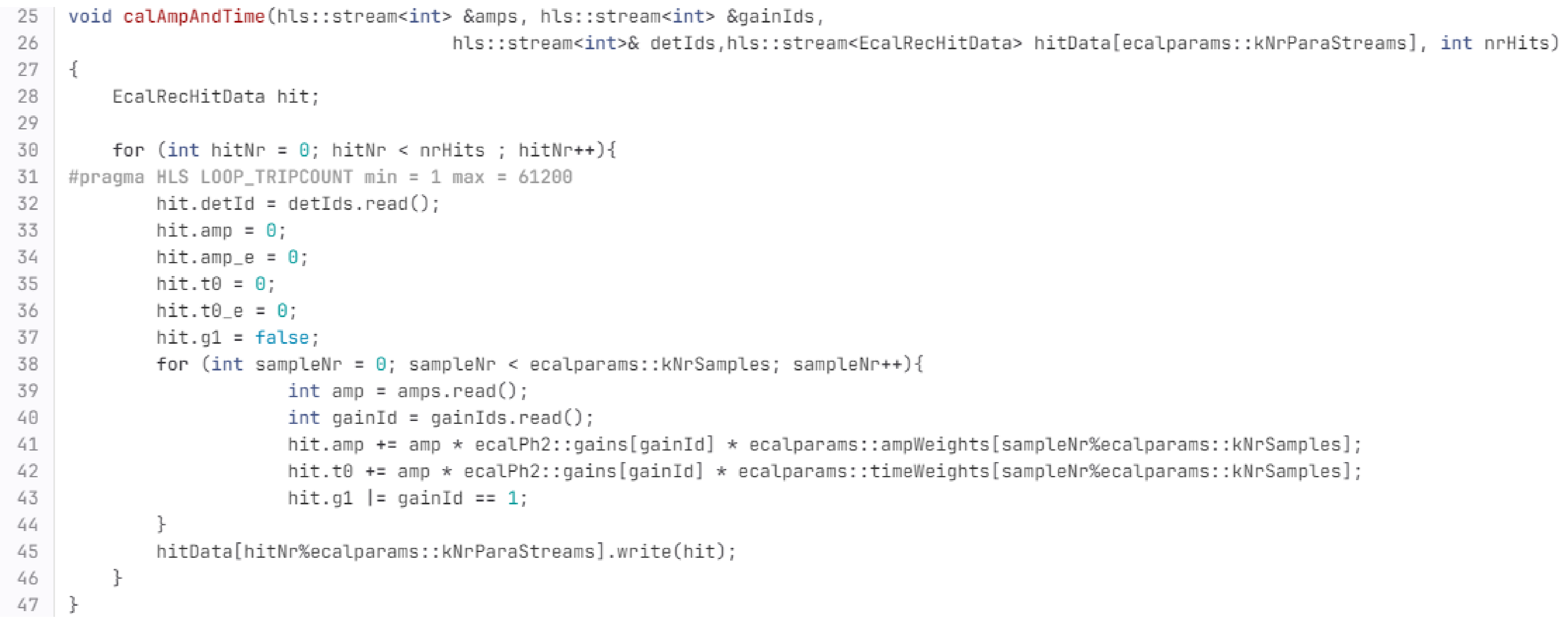
load



```
void loadData(const EcalLiteDTUSample *samples,const DetId* detIds,
                hls::stream<int>& ampsStream,hls::stream<int>& gainIdsStream,
                            hls::stream<int>& detIdsStream,int nrHits){
        for (int hitNr = 0 ; hitNr < nrHits; hitNr++){
#pragma HLS LOOP_TRIPCOUNT min = 1 max = 61200
        detIdsStream << detIds[hitNr];
        for (int sampleNr = 0; sampleNr < ecalparams::kNrSamples; sampleNr++){
            EcalLiteDTUSample sample = samples[sampleNr+hitNr*ecalparams::kNrSamples];
            ampsStream << sample.adc();
            gainIdsStream << sample.gainId();
        }
    }
}
```

- loads the data from global memory
- hls::streams are 1 and 16 (#samples) depth fifos
  - this is the model suggested by Xilinx
  - using hls::streams rather than direct memory access helps improve pipelining

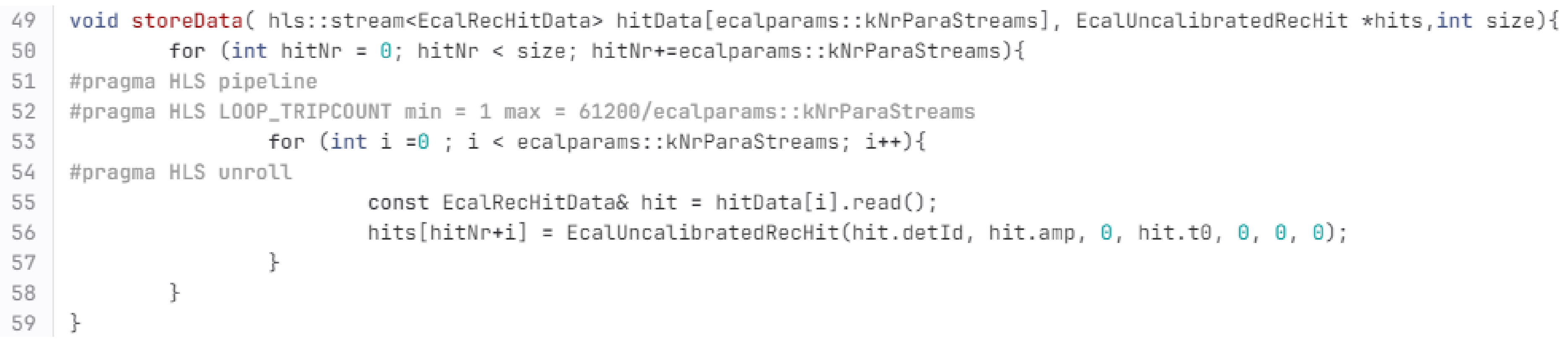
compute



```
void calAmpAndTime(hls::stream<int> &amps, hls::stream<int> &gainIds,
                                   hls::stream<int>& detIds,hls::stream<EcalRecHitData> hitData[ecalparams::kNrParaStreams], int nrHits)
{
    EcalRecHitData hit;

    for (int hitNr = 0; hitNr < nrHits ; hitNr++){
#pragma HLS LOOP_TRIPCOUNT min = 1 max = 61200
        hit.detId = detIds.read();
        hit.amp = 0;
        hit.amp_e = 0;
        hit.t0 = 0;
        hit.t0_e = 0;
        hit.g1 = false;
        for (int sampleNr = 0; sampleNr < ecalparams::kNrSamples; sampleNr++){
                    int amp = amps.read();
                    int gainId = gainIds.read();
                    hit.amp += amp * ecalPh2::gains[gainId] * ecalparams::ampWeights[sampleNr%ecalparams::kNrSamples];
                    hit.t0 += amp * ecalPh2::gains[gainId] * ecalparams::timeWeights[sampleNr%ecalparams::kNrSamples];
                    hit.g1 |= gainId == 1;
        }
        hitData[hitNr%ecalparams::kNrParaStreams].write(hit);
    }
}
```

- computes the amplitudes

write



```
void storeData( hls::stream<EcalRecHitData> hitData[ecalparams::kNrParaStreams], EcalUncalibratedRecHit *hits,int size){
        for (int hitNr = 0; hitNr < size; hitNr+=ecalparams::kNrParaStreams){
#pragma HLS pipeline
#pragma HLS LOOP_TRIPCOUNT min = 1 max = 61200/ecalparams::kNrParaStreams
                for (int i =0 ; i < ecalparams::kNrParaStreams; i++){
#pragma HLS unroll
                        const EcalRecHitData& hit = hitData[i].read();
                        hits[hitNr+i] = EcalUncalibratedRecHit(hit.detId, hit.amp, 0, hit.t0, 0, 0, 0);
                }
        }
}
```

- writes its to the global memory

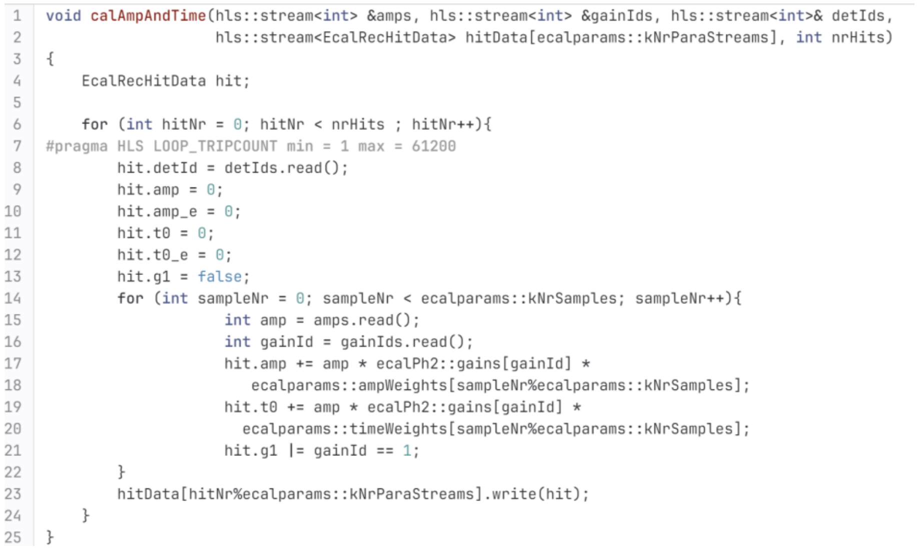
Benchmark numbers

#Threads	No Pipeline (ev/s)	Pipeline (ev/s)	CPU (ev/s)
2	1.19 ± 0.01	15.6± 0.7	15.4 ± 0.2
1	1.11 ± 0.01	10.66 ± 0.03	7.1± 0.3

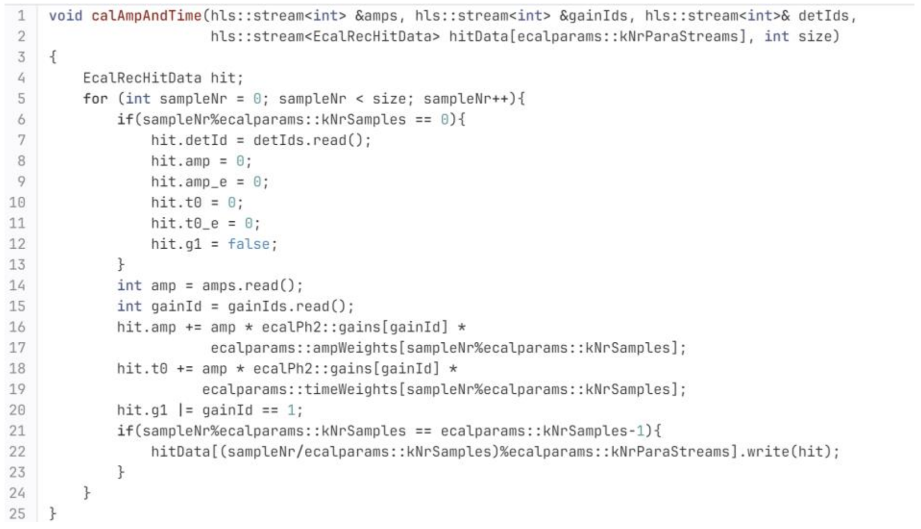
- issue that algo is so fast that the read IO of the digis is bottleneck
- make 8 copies of the producer so makes the hits 8 times such that I/O is not a bottle neck
- ran either one or two threads for the reconstruction
  - 2 threads is probably now saturating the FPGA kernel
- message: we are roughly comparable to CPU now
  - some caveats that the FPGA doesn't do all the CPU does but don't think it will effect the general message

Aside

- two implementations of the algo



```
void calAmpAndTime(hls::stream<int> &amps, hls::stream<int> &gainIds, hls::stream<int>& detIds,
                   hls::stream<EcalRecHitData> hitData[ecalparams::kNrParaStreams], int nrHits)
{
    EcalRecHitData hit;

    for (int hitNr = 0; hitNr < nrHits ; hitNr++){
#pragma HLS LOOP_TRIPCOUNT min = 1 max = 61200
        hit.detId = detIds.read();
        hit.amp = 0;
        hit.amp_e = 0;
        hit.t0 = 0;
        hit.t0_e = 0;
        hit.g1 = false;
        for (int sampleNr = 0; sampleNr < ecalparams::kNrSamples; sampleNr++){
                    int amp = amps.read();
                    int gainId = gainIds.read();
                    hit.amp += amp * ecalPh2::gains[gainId] *
                       ecalparams::ampWeights[sampleNr%ecalparams::kNrSamples];
                    hit.t0 += amp * ecalPh2::gains[gainId] *
                      ecalparams::timeWeights[sampleNr%ecalparams::kNrSamples];
                    hit.g1 |= gainId == 1;
        }
        hitData[hitNr%ecalparams::kNrParaStreams].write(hit);
    }
}
```



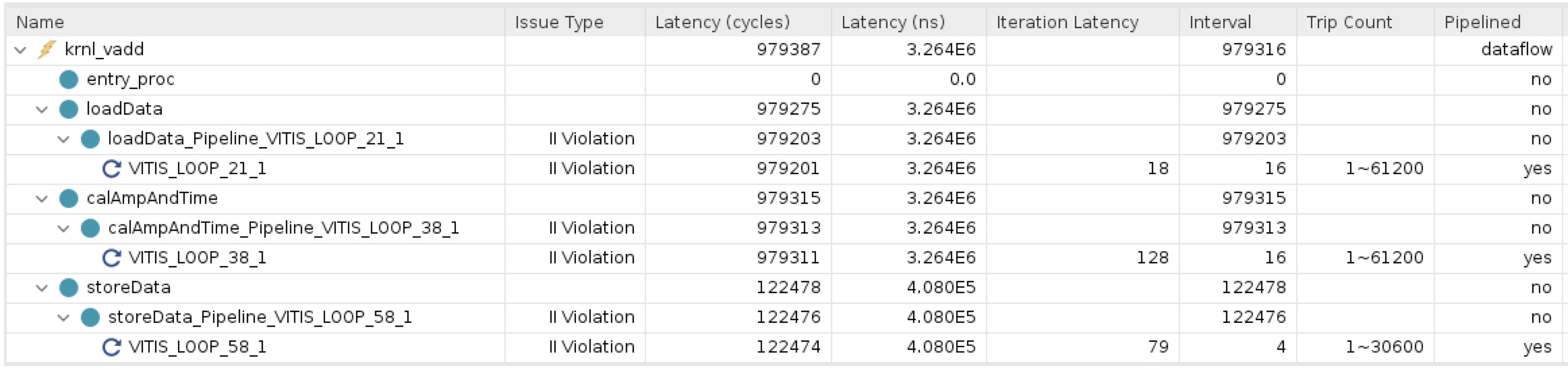
```
void calAmpAndTime(hls::stream<int> &amps, hls::stream<int> &gainIds, hls::stream<int>& detIds,
                   hls::stream<EcalRecHitData> hitData[ecalparams::kNrParaStreams], int size)
{
    EcalRecHitData hit;
    for (int sampleNr = 0; sampleNr < size; sampleNr++){
        if(sampleNr%ecalparams::kNrSamples == 0){
            hit.detId = detIds.read();
            hit.amp = 0;
            hit.amp_e = 0;
            hit.t0 = 0;
            hit.t0_e = 0;
            hit.g1 = false;
        }
        int amp = amps.read();
        int gainId = gainIds.read();
        hit.amp += amp * ecalPh2::gains[gainId] *
                   ecalparams::ampWeights[sampleNr%ecalparams::kNrSamples];
        hit.t0 += amp * ecalPh2::gains[gainId] *
                  ecalparams::timeWeights[sampleNr%ecalparams::kNrSamples];
        hit.g1 |= gainId == 1;
        if(sampleNr%ecalparams::kNrSamples == ecalparams::kNrSamples-1){
            hitData[(sampleNr/ecalparams::kNrSamples)%ecalparams::kNrParaStreams].write(hit);
        }
    }
}
```

- left runs 4 times faster then the right…
  - 15.6± 0.7 vs 4.79 ± 0.01 ev/s
- admittedly the right is a little weirdly implemented but didn’t think would be that slow
  - looking at it now, it is possible vitis didn’t pipeline the right loop

Vitis Reports

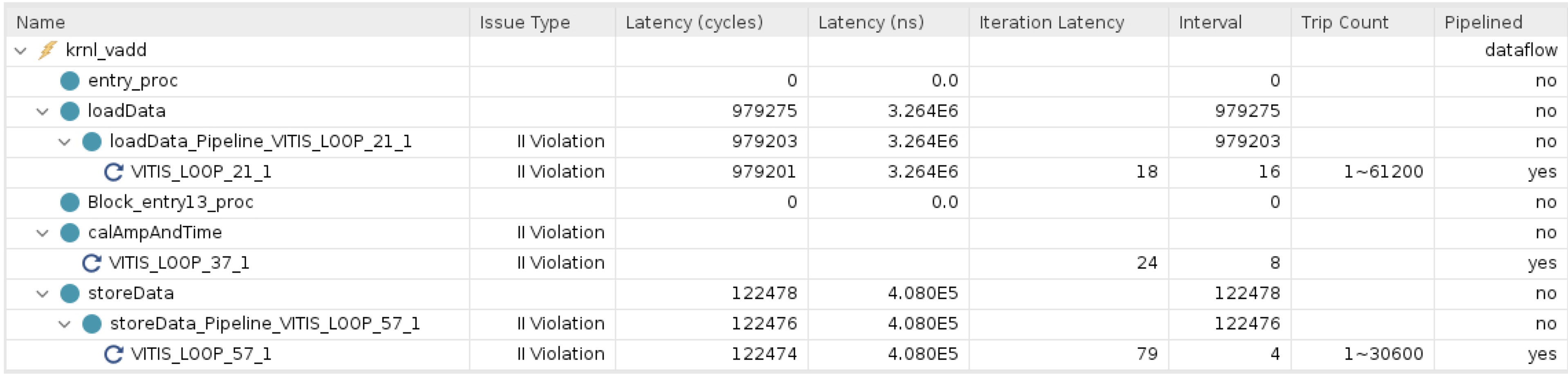
- still trying to understand these
- from the numbers I would have thought the speed was the other way around

fast version



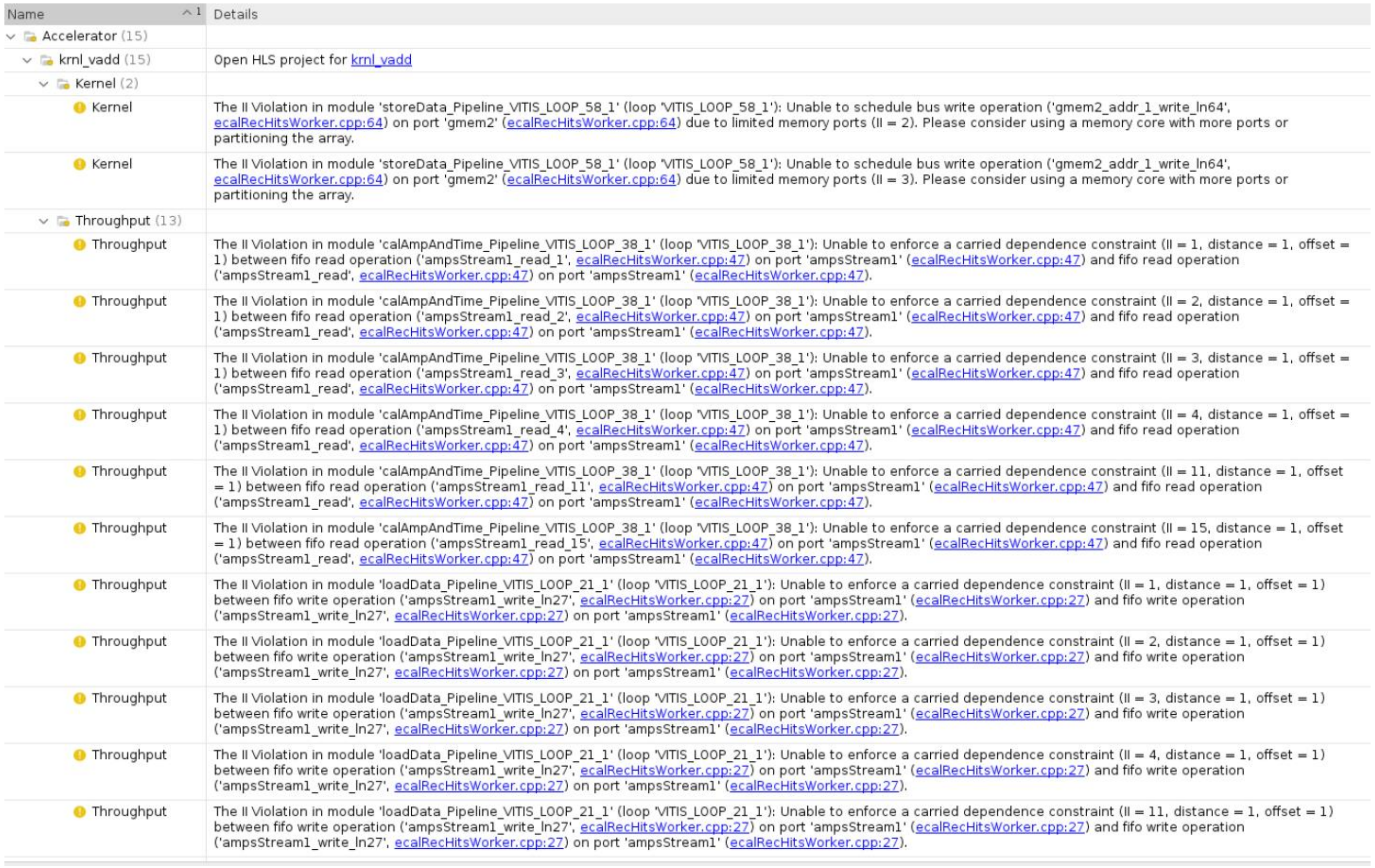
Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined
krnl_vadd		979387	3.264E6		979316		dataflow
entry_proc		0	0.0		0		no
loadData		979275	3.264E6		979275		no
loadData_Pipeline_VITIS_LOOP_21_1	II Violation	979203	3.264E6		979203		no
VITIS_LOOP_21_1	II Violation	979201	3.264E6	18	16	1~61200	yes
calAmpAndTime		979315	3.264E6		979315		no
calAmpAndTime_Pipeline_VITIS_LOOP_38_1	II Violation	979313	3.264E6		979313		no
VITIS_LOOP_38_1	II Violation	979311	3.264E6	128	16	1~61200	yes
storeData		122478	4.080E5		122478		no
storeData_Pipeline_VITIS_LOOP_58_1	II Violation	122476	4.080E5		122476		no
VITIS_LOOP_58_1	II Violation	122474	4.080E5	79	4	1~30600	yes

slow version



Name	Issue Type	Latency (cycles)	Latency (ns)	Iteration Latency	Interval	Trip Count	Pipelined
krnl_vadd							dataflow
entry_proc		0	0.0		0		no
loadData		979275	3.264E6		979275		no
loadData_Pipeline_VITIS_LOOP_21_1	II Violation	979203	3.264E6		979203		no
VITIS_LOOP_21_1	II Violation	979201	3.264E6	18	16	1~61200	yes
Block_entry13_proc		0	0.0		0		no
calAmpAndTime	II Violation						no
VITIS_LOOP_37_1	II Violation			24	8		yes
storeData		122478	4.080E5		122478		no
storeData_Pipeline_VITIS_LOOP_57_1	II Violation	122476	4.080E5		122476		no
VITIS_LOOP_57_1	II Violation	122474	4.080E5	79	4	1~30600	yes

Still Room to speed it up



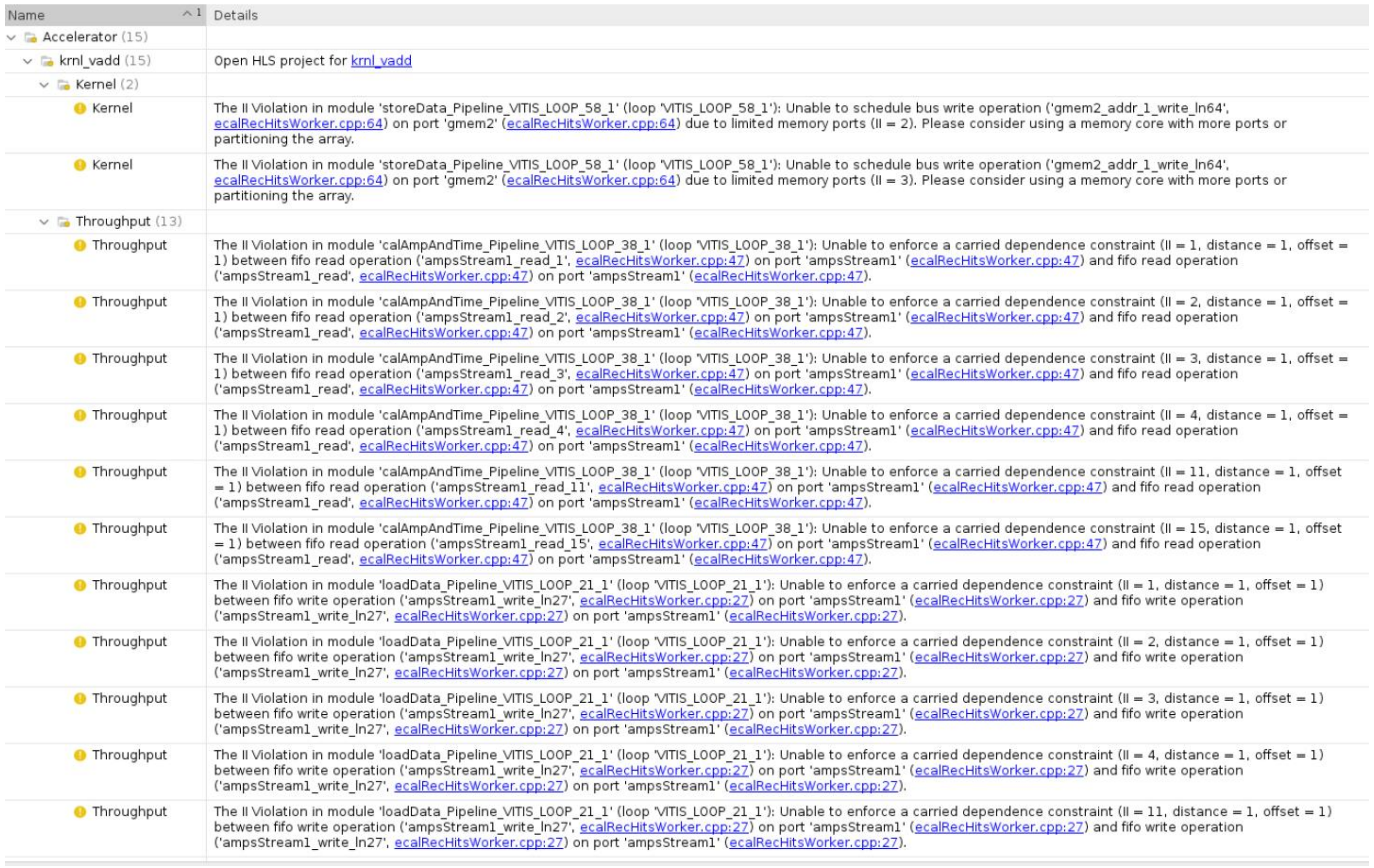
Name	Details
Accelerator (15)	
krnl_vadd (15)	Open HLS project for krnl_vadd
Kernel (2)	
Kernel	The II Violation in module 'storeData_Pipeline_VITIS_LOOP_58_1' (loop 'VITIS_LOOP_58_1'): Unable to schedule bus write operation ('gmem2_addr_1_write_ln64', ecalRecHitsWorker.cpp:64) on port 'gmem2' (ecalRecHitsWorker.cpp:64) due to limited memory ports (II = 2). Please consider using a memory core with more ports or partitioning the array.
Kernel	The II Violation in module 'storeData_Pipeline_VITIS_LOOP_58_1' (loop 'VITIS_LOOP_58_1'): Unable to schedule bus write operation ('gmem2_addr_1_write_ln64', ecalRecHitsWorker.cpp:64) on port 'gmem2' (ecalRecHitsWorker.cpp:64) due to limited memory ports (II = 3). Please consider using a memory core with more ports or partitioning the array.
Throughput (13)	
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 1, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_1', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 2, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_2', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 3, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_3', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 4, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_4', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 11, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_11', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 15, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_15', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 1, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 2, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 3, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 4, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 11, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).

several advisories impacting initiation interval

issue with bandwidth writing on gmem

- tried to have multiple streams for writing the hits , didn't work
- now trying to widen the port width

Still Room to speed it up



Name	Details
Accelerator (15)	
krnl_vadd (15)	Open HLS project for krnl_vadd
Kernel (2)	
Kernel	The II Violation in module 'storeData_Pipeline_VITIS_LOOP_58_1' (loop 'VITIS_LOOP_58_1'): Unable to schedule bus write operation ('gmem2_addr_1_write_ln64', ecalRecHitsWorker.cpp:64) on port 'gmem2' (ecalRecHitsWorker.cpp:64) due to limited memory ports (II = 2). Please consider using a memory core with more ports or partitioning the array.
Kernel	The II Violation in module 'storeData_Pipeline_VITIS_LOOP_58_1' (loop 'VITIS_LOOP_58_1'): Unable to schedule bus write operation ('gmem2_addr_1_write_ln64', ecalRecHitsWorker.cpp:64) on port 'gmem2' (ecalRecHitsWorker.cpp:64) due to limited memory ports (II = 3). Please consider using a memory core with more ports or partitioning the array.
Throughput (13)	
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 1, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_1', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 2, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_2', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 3, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_3', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 4, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_4', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 11, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_11', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'calAmpAndTime_Pipeline_VITIS_LOOP_38_1' (loop 'VITIS_LOOP_38_1'): Unable to enforce a carried dependence constraint (II = 15, distance = 1, offset = 1) between fifo read operation ('ampsStream1_read_15', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47) and fifo read operation ('ampsStream1_read', ecalRecHitsWorker.cpp:47) on port 'ampsStream1' (ecalRecHitsWorker.cpp:47).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 1, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 2, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 3, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 4, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).
Throughput	The II Violation in module 'loadData_Pipeline_VITIS_LOOP_21_1' (loop 'VITIS_LOOP_21_1'): Unable to enforce a carried dependence constraint (II = 11, distance = 1, offset = 1) between fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27) and fifo write operation ('ampsStream1_write_ln27', ecalRecHitsWorker.cpp:27) on port 'ampsStream1' (ecalRecHitsWorker.cpp:27).

several advisories impacting initiation interval

issue with the 16 samples being read in the adder causing a delay of 16 cycles

- modified it with a stream for each sample -> advisories went away but FIFO deadlocked -> still trying to understand

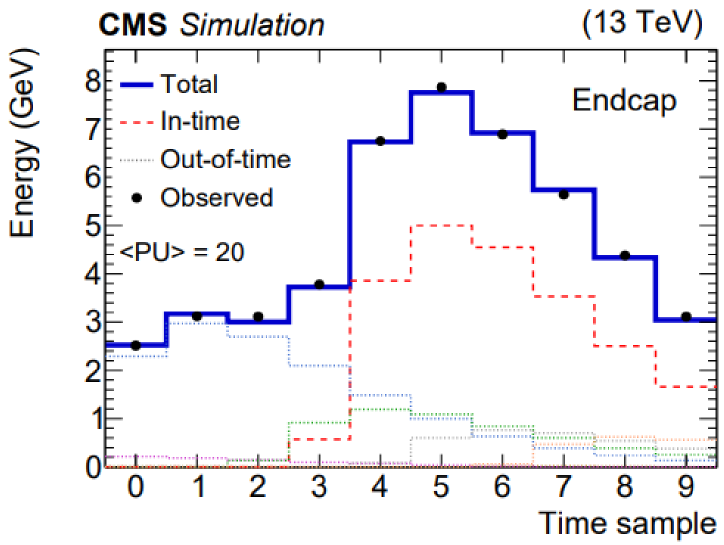
Aside: Deadlocking

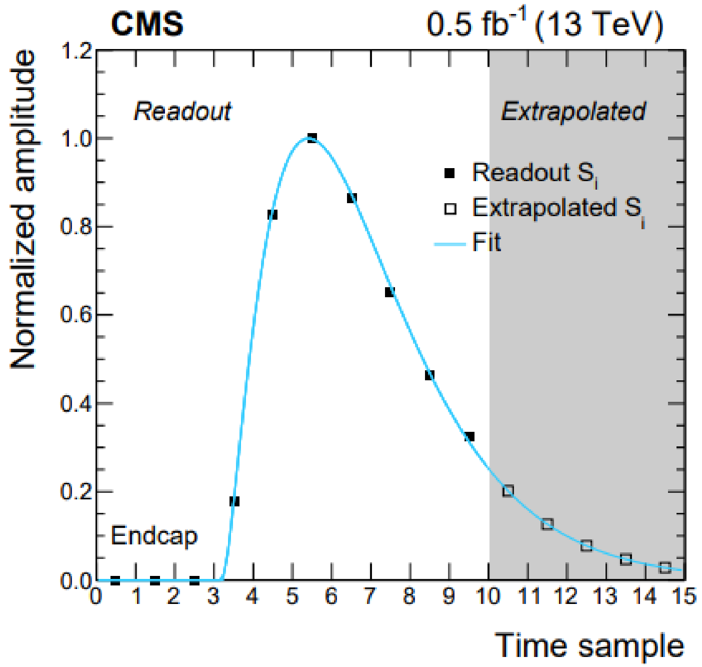
- doing this approach it was really easy to deadlock the FPGA
  - original design naively had two load functions
    - 1) loaded all detIds
    - 2) loaded all amplitudes
  - this caused the FPGA to deadlock and honestly the detection of the tools to prevent this is subpar
    - deadlock detection on the emulator works sometimes
    - and when it did, the error message was buried in a non obvious log file
      - Emulation-HW/.run/151479/hw_em/device0/binary_0/behav_waveform/xsim/simulate.log
      - yes it was in a hidden directory…
    - wasted quite a bit of time with this

ECAL Multifit

JINST **15** P10002

- both ECAL and HCAL local calo reconstruction work on the same principle
  - fit to each amplitude in each bunch crossing
- has CUDA and alpaka based algos implemented in CMSSW
  - large fraction of time spent in the HLT
  - probably the single biggest thing to port
- standard linear algebra solve:
  - *P a = b*
  - P is the contribution of BX_i deposit to the counts in BX_j
  - a is the amplitudes of each deposit in each BX
  - b observe counts in each BX





Vitis Libraries

- vitis has several Apache 2.0 licensed libraries
  - https://www.xilinx.com/products/design-tools/vitis/vitis-libraries.html
  - https://github.com/Xilinx/Vitis_Libraries
- relevant to the multifit problem are
  - **blas** : basic linear algebra subroutines
  - **solver**: collection of matrix decomposition operations, linear solvers and eigenvalue solvers
    - was attempting to write a NNLS algo myself till I noticed their implementation although not non negative constrained but good enough
- header only libraries, easy to include
  - for some definition of easy, the documentation is merely *okay...*
  - not much guidance on making it fast

Vitis BLAS

- aimed to test out its matrix methods for multiplication
  - use Eigen as a CPU reference
- simple matrix multiplication as a starting point, 1000 4x4 matrix x vector multiplications
  - FPGA duration: 875 us
  - FPGA mem duration:120 us
  - CPU duration: 43us
- not great but then this is something the CPU does very well
  - I also suspect my usage not well pipelined or making efficient use of FPGA

Vitis Solver

- using `xf::solver::gelinearsolver`
  - solves $Ax = b$ where A is a general *mxm* matrix, x is the vector to solve and B is the observed vector
  - A = pulse shape matrix, x = bx amplitudes, b = observed ADC counts in each timeslice
- benchmark:
  - 10x10 matrices of doubles (ECAL pulse shape matrix size)
  - matrix is random numbers in [0,1], A is random ints from [0,1023]
  - CPU reference is Eigen:
    - https://eigen.tuxfamily.org/dox/group__LeastSquares.html
      - also tried Eigen::NNLS
    - best method seemed to be: “Using normal equations”
  - gcc flags: -O3 -march=cascadelake

Solver Benchmark results

- 100 matrixes per event (sent at once)
  - 1st event:
    - FPGA duration: 4399µs
    - FPGA mem duration: 677 µs
    - CPU duration: 1004 µs
  - subsequent events:
    - FPGA duration: 3855 ± 5 µs
    - FPGA mem duration:253 ± 3 µs
    - CPU duration: 347 ± 2.5 µs

its not clear to me why the CPU speeds up on subsequent events

possibly caches or something

- FPGA offload takes roughly 4-10 times as CPU
  - I had hoped this would be faster, particularly as this is a stock algo
  - need to understand if there is something we can do to speed this up or if we're not using it correctly or something
  - major focus here to understand this
- note: this is using ~2% of the FPGA
  - but is also only using 1 core of the CPU

Aside on Alpaka

- RAL CMS through Thomas Reis is involved in porting CMS's CUDA based code to Alpaka
  - specifically the ECAL multifit code
  - merged in CMSSW, undergoing validation
  - goal is to be in production this year
- CMSSW source code:
  - https://github.com/cms-sw/cmssw/tree/master/RecoLocalCalo/EcalRecProducers/plugins/alpaka
- observations:
  - originally thought that it would be simple to port from CUDA
    - its heavily inspired by it
  - turns out not so much:
    - devil in the details
    - shared memory an issue
  - CMSSW code is an example implementation now but it is heavily CMSSW framework based

Summary

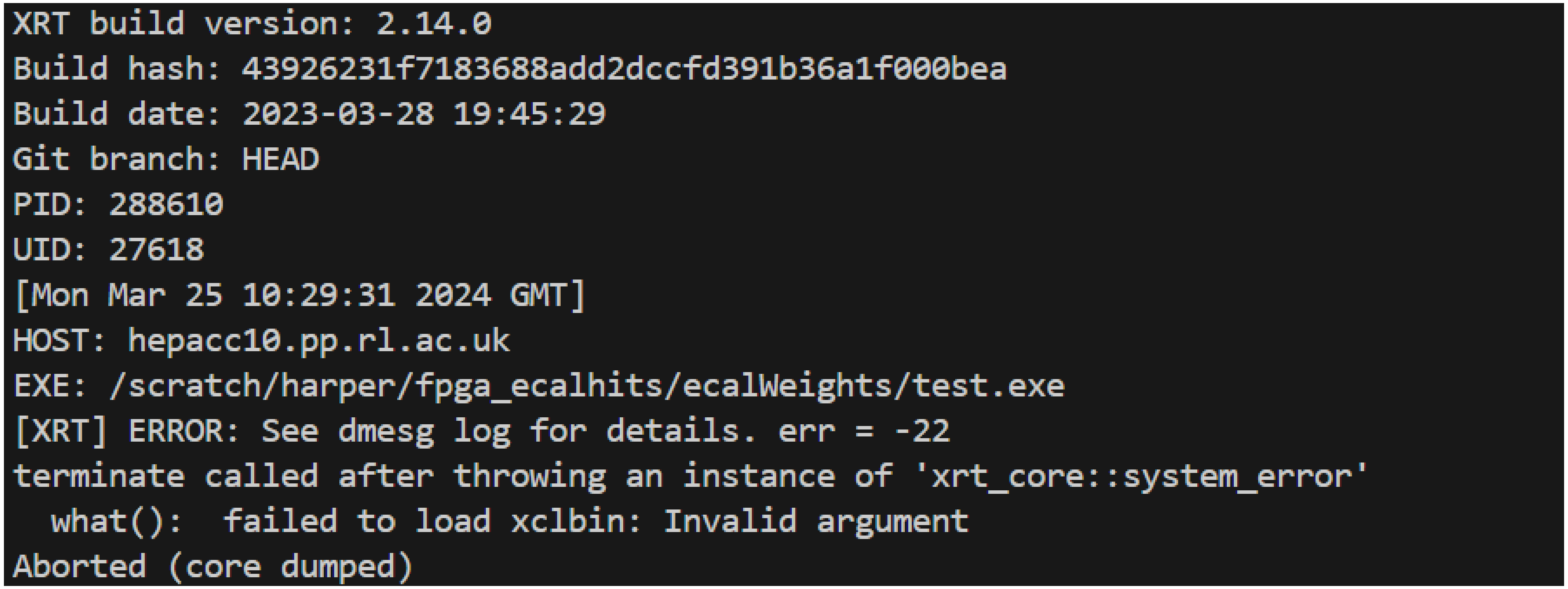
- HLS has some nice features:
  - can produce CMSSW data types directly
  - integrates with CMSSW external work framework nicely
- can now produce ECAL uncalibrated rechits on FPGA in same time as CPU
  - also likely room for further improvement, lots of advanced features to try
  - still need to verify and test as the setup is a little artificial and may introduce biases
- it is tricky to get performance from
  - still fairly slow, tools are a little lacking
    - new vitis gui released, to be tested
  - development cycles are also a little painful (although now setup to work more efficiently)
- still want to try a few things to improve algo performance
  - focus in on trying to get the multifit algorithm to work and be performant
    - takes a huge part of CMS HLT reco time
- aim to document various performance improvements
  - still trying to understand why some algos are fast and some algos are slo
- after that, probably should do some ML inference studies
  - again though I'm wary of dedicated chips being better for that

Spares

Useful Info Learnt

- the following is more a reference of issues we encountered
- it was not presented but could be a useful reference for folks trying to reproduce our efforts

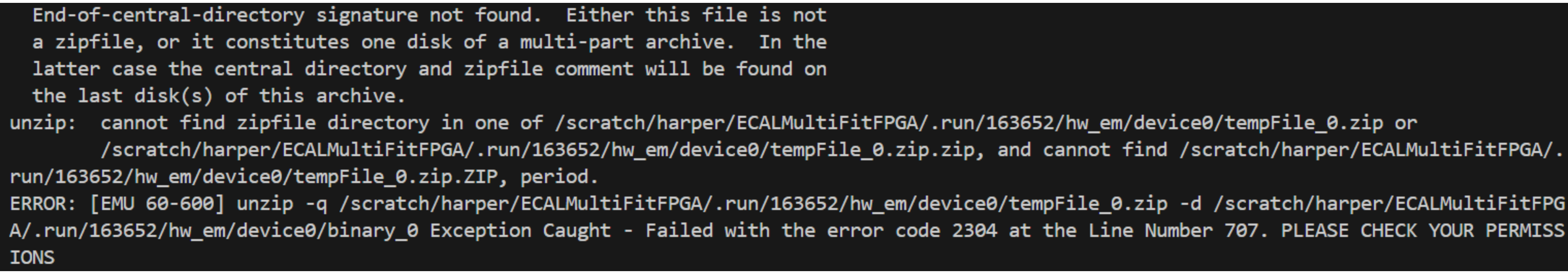
Running Emulated Binaries: Crash



```
XRT build version: 2.14.0
Build hash: 43926231f7183688add2dccfd391b36a1f000bea
Build date: 2023-03-28 19:45:29
Git branch: HEAD
PID: 288610
UID: 27618
[Mon Mar 25 10:29:31 2024 GMT]
HOST: hepacc10.pp.rl.ac.uk
EXE: /scratch/harper/fpga_ecalhits/ecalWeights/test.exe
[XRT] ERROR: See dmesg log for details. err = -22
terminate called after throwing an instance of 'xrt_core::system_error'
  what():  failed to load xclbin: Invalid argument
Aborted (core dumped)
```

- loading an hw_emu binary without setting
  - export XCL_EMULATION_MODE=hw_emu

Running HW Binaries: Crash



```
  End-of-central-directory signature not found.  Either this file is not
  a zipfile, or it constitutes one disk of a multi-part archive.  In the
  latter case the central directory and zipfile comment will be found on
  the last disk(s) of this archive.
unzip:  cannot find zipfile directory in one of /scratch/harper/ECALMultiFitFPGA/.run/163652/hw_em/device0/tempFile_0.zip or
        /scratch/harper/ECALMultiFitFPGA/.run/163652/hw_em/device0/tempFile_0.zip.zip, and cannot find /scratch/harper/ECALMultiFitFPGA/.
run/163652/hw_em/device0/tempFile_0.zip.ZIP, period.
ERROR: [EMU 60-600] unzip -q /scratch/harper/ECALMultiFitFPGA/.run/163652/hw_em/device0/tempFile_0.zip -d /scratch/harper/ECALMultiFitFPG
A/.run/163652/hw_em/device0/binary_0 Exception Caught - Failed with the error code 2304 at the Line Number 707. PLEASE CHECK YOUR PERMISS
IONS
```

- loading a hw binary with
  - XCL_EMULATION_MODE=hw_emu