Online track reconstruction using Kalman filters on FPGAs

Connecting the Dots / Workshop on Intelligent Trackers 2017, LAL Orsay

Sioni Summers

CMS Compact Muon Solenoid

Imperial College London

MAXELER Technologies
MAXIMUM PERFORMANCE COMPUTING

Contents

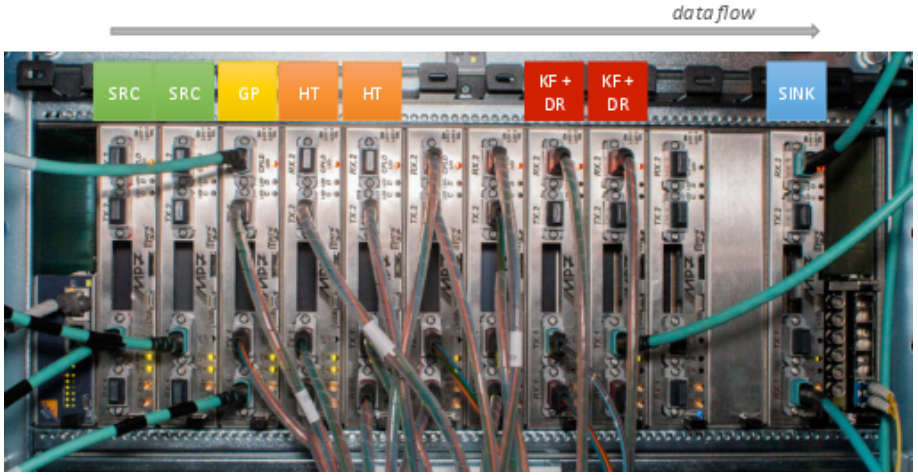
- An FPGA based track finder for the HL-LHC
- The Kalman Filter track fit
- FPGA Advantage
- A Level 1 Kalman Filter
- FPGA coprocessor

Context

- At HL-LHC will see up to 200 simultaneous pp collisions per bunch crossing at 40MHz
- Tracking at Level 1 necessary to maintain acceptable trigger rate at desirable particle momentum thresholds
- Tracking at High Level Trigger becomes harder (slower)
  - New algorithms, new technology, both?

FPGA Track Finder

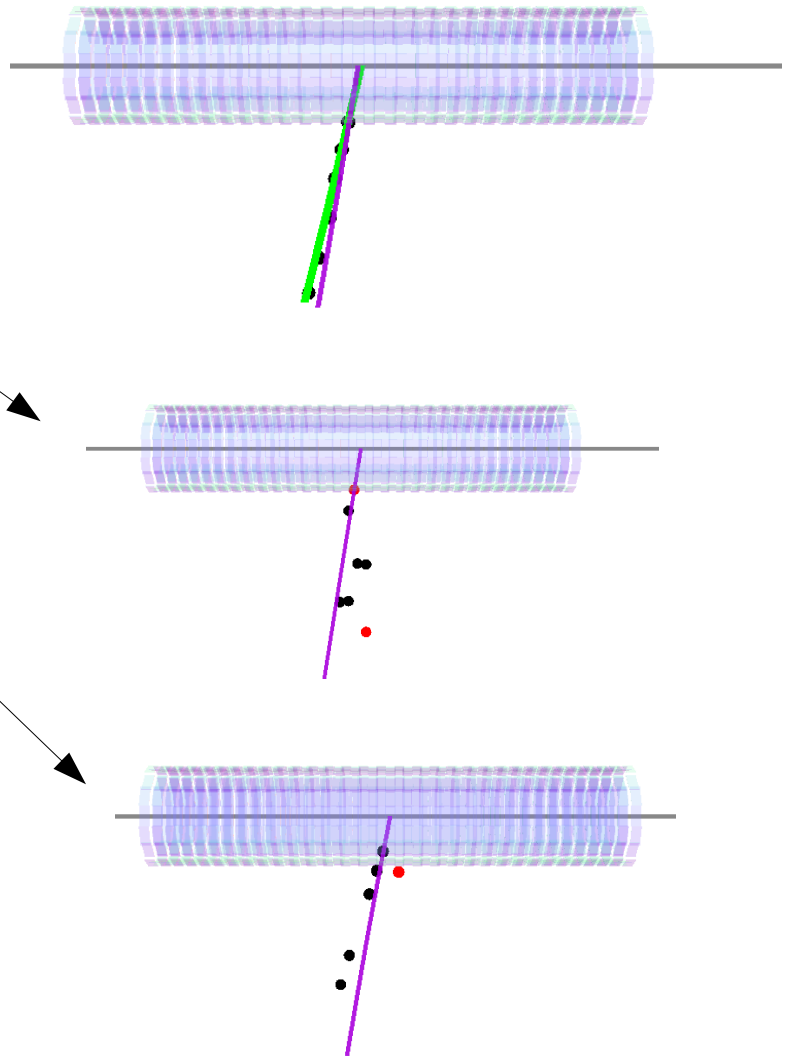
- Developed for CMS Level 1 Time Multiplexed Track Trigger Demonstrator
- Hough Transform (HT) finds tracks in r-ϕ plane
  - After detector segmentation in η & ϕ
- Implemented in MP7 boards (Virtex 7 690T FPGA, 1Tb/s max. throughput)
- Time multiplexing + detector segmentation + HT = 1.3μs
  - Limit 12.5μs total for L1



- UK TMTT demonstrator: reconstructing 1/8 detector, 1/36 events in today's hardware
- See “An FPGA based track finder at Level 1 for CMS at the High Luminosity LHC”, Alexander Morton, this morning

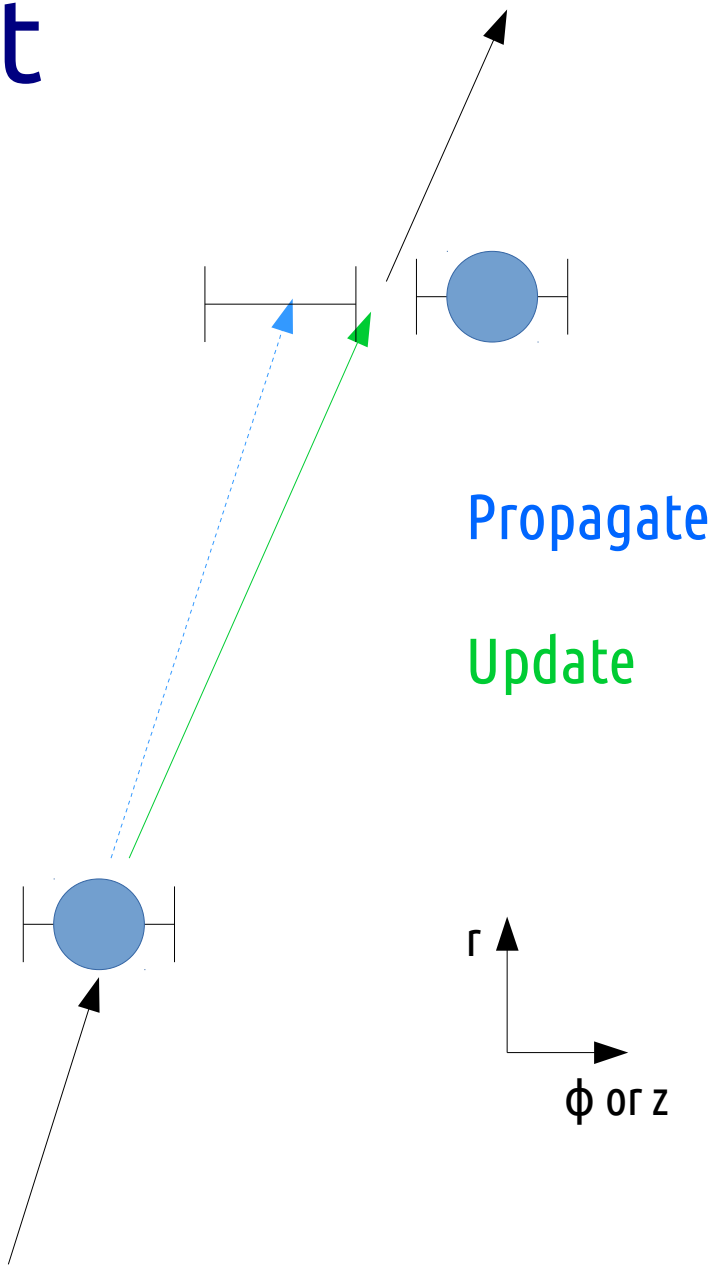
Fitting and filtering environment

- HT finds 96% of genuine tracks
- 28% of found tracks are fake
- 56% of genuine tracks contain extra fake stubs
  - → cleaning of tracks important
- Average 30 candidates per HT board per event
  - New event every 900ns
- Motivates a Kalman Filter with separate propagation when multiple stubs found



Kalman Filter Track Fit

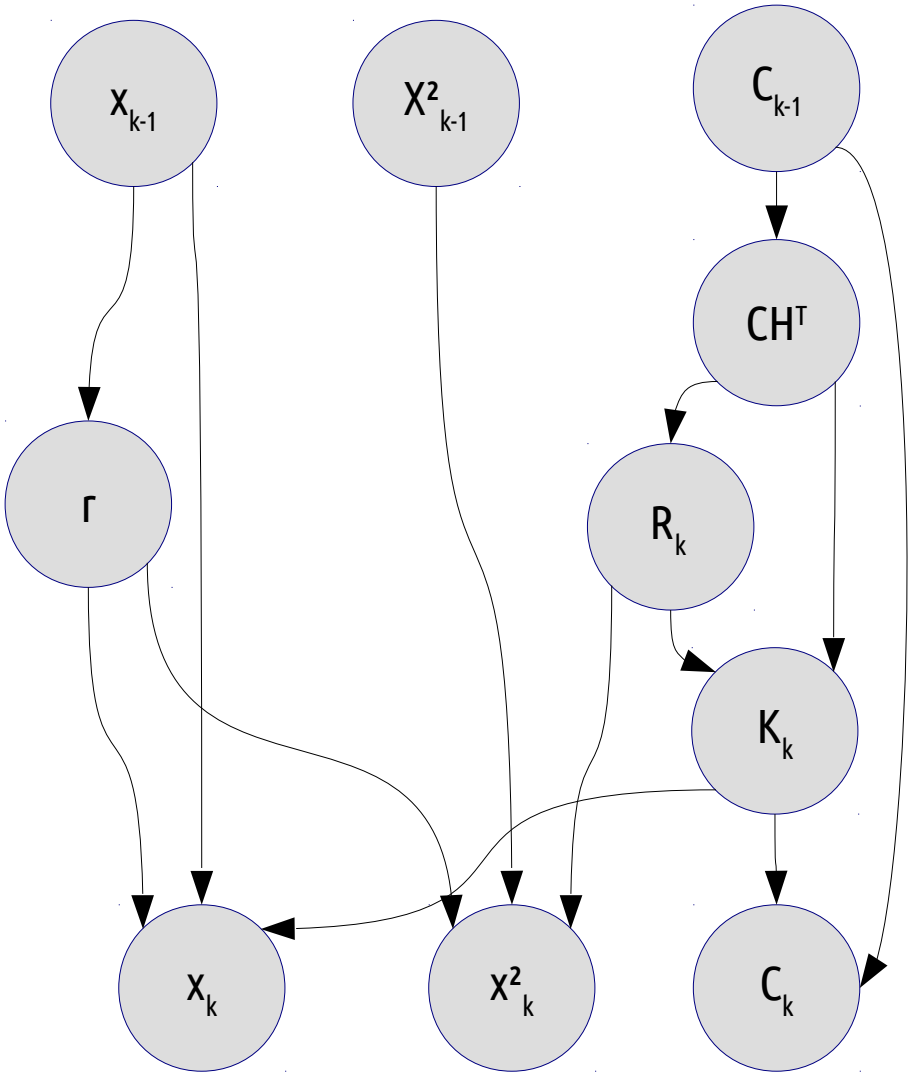
- Fit and filter candidates found by the HT
  - Improve q/p_T, ϕ_0 resolution, obtain vertex, pseudo-rapidity
- Propagate track parameters inside-out
- Use stubs to update track parameters
- Ignore hits with large residuals
- Split candidate into independent candidates when multiple stubs on a layers
  - r-ϕ only finding adds incorrect hits to tracks
  - Don't allow bad hits to contribute
  - Improves resolution & purity



Kalman Filter - equations

$$
\begin{aligned}
x_k^{k-1} &= \mathbf{F}_{k-1} x_{k-1} \\
\mathbf{C}_k^{k-1} &= \mathbf{F}_{k-1} \mathbf{C}_{k-1} \mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \\
r_k^{k-1} &= m_k - \mathbf{H}_k x_k^{k-1} \\
\mathbf{R}_k^{k-1} &= \mathbf{V}_k + \mathbf{H}_k \mathbf{C}_k^{k-1} \mathbf{H}_k^T \\
\mathbf{K}_k &= \mathbf{C}_k^{k-1} \mathbf{H}_k^T \left(\mathbf{R}_k^{k-1}\right)^{-1} \\
x_k &= x_k^{k-1} + \mathbf{K}_k r_k^{k-1} \\
\mathbf{C}_k &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{C}_k^{k-1} \\
\chi_+^2 &= r_k^{k-1T} \left(\mathbf{R}_k^{k-1}\right)^{-1} r_k^{k-1} \\
\chi_k^2 &= \chi_{k-1}^2 + \chi_+^2
\end{aligned}
$$

- k-1, k: previous layer, current layer
- x: track helix parameters
- **C**: their covariance matrix
- m: a measurement (stub)
- **F**: forecast matrix
- **H**: measurement matrix
- **K**: the Kalman gain

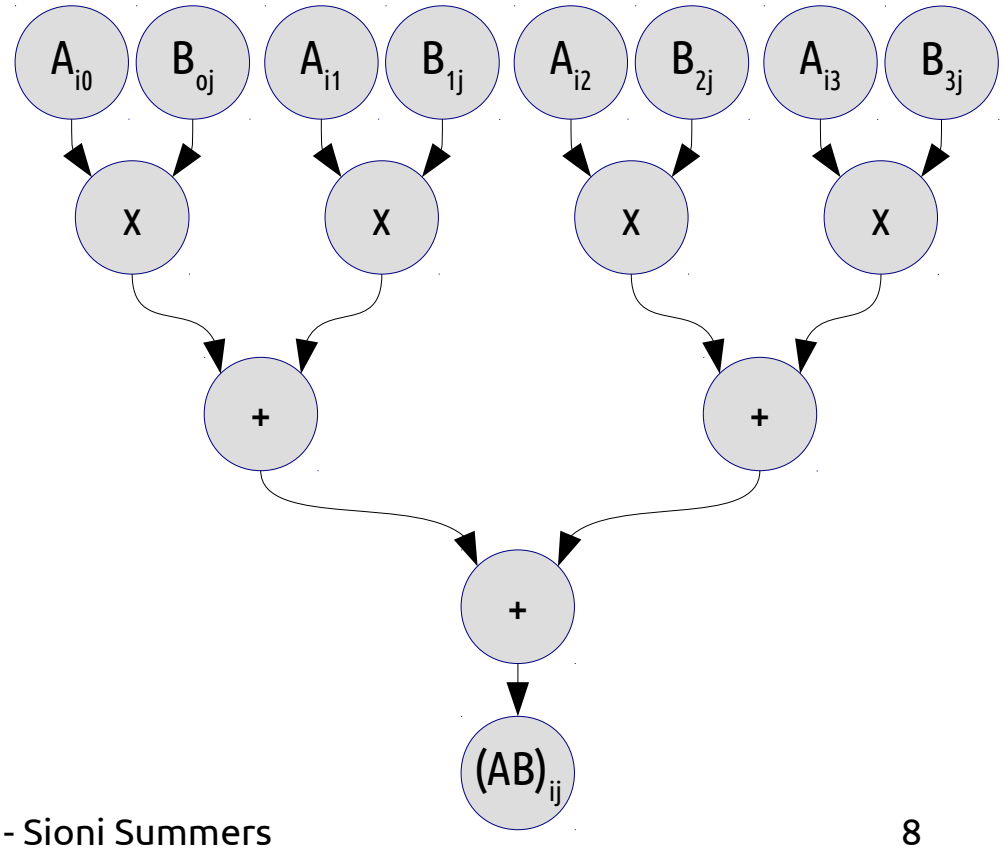


FPGA advantage

- FPGAs are a programmable digital circuit
- 'Programming in space'
  - Low level parallelism of operations
- Pipelining
  - Assembly line of instructions
  - Accept new data on every clock cycle
- Large I/O bandwidth capacity
  - 1 Tb/s for MP7

Matrix multiplication using unique resources for each operation

```
for(k = 0; k < 4; k++)
    AB_[i][j][k] =  A[i][k] * B[k][j];
AB[i][j] = (AB_[i][j][0] + AB_[i][j][1]) +
(...);
```



Designing a fast Kalman Filter

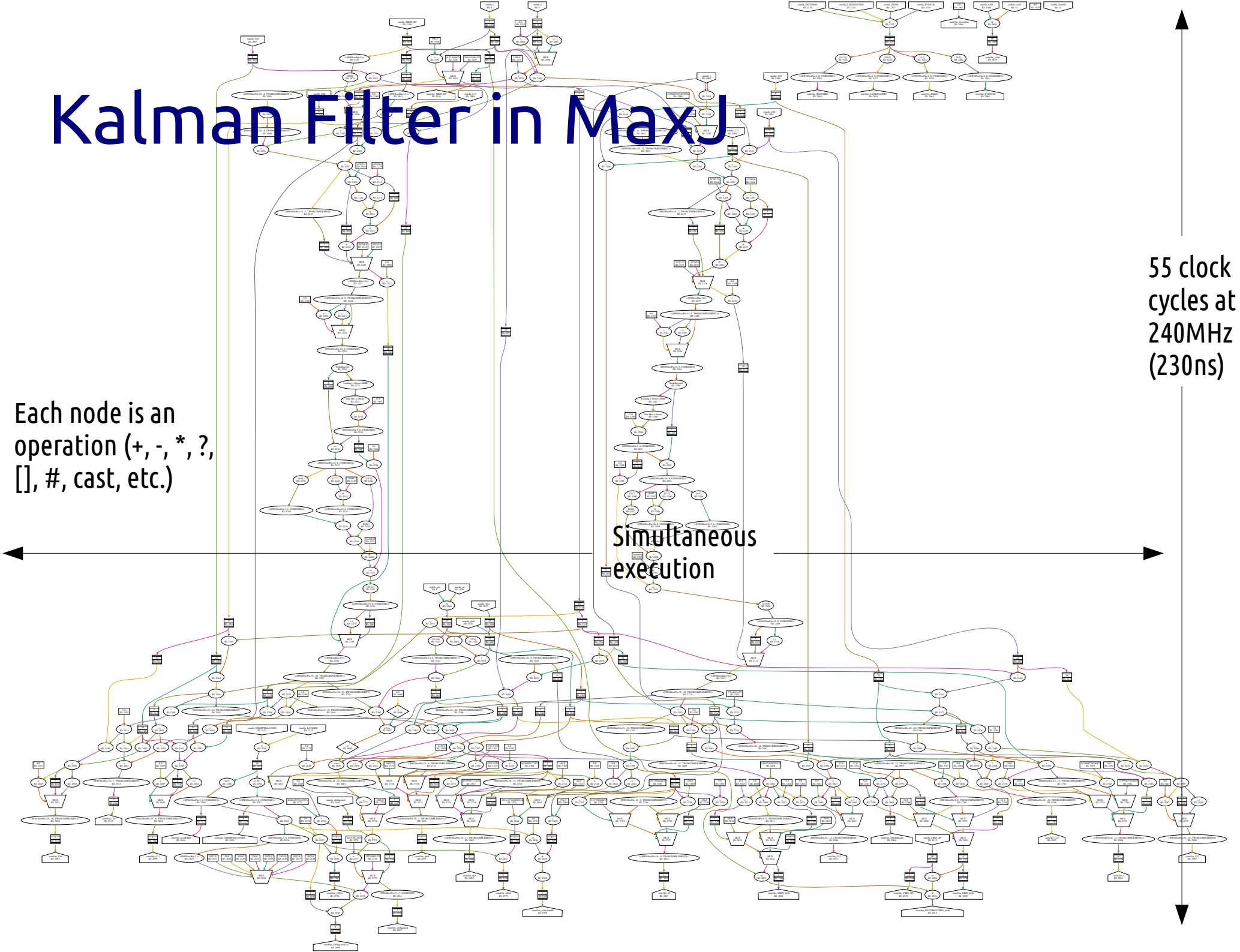
- Lots of tricks required for μs latency:
- Simplified track equations
  - 4 parameters, straight lines in r-ϕ and r-z
- No multiple scattering
- Homogeneous magnetic field
- Limit to 4 iterations (6 layer outer tracker)
- Fixed-point arithmetic
  - e.g. $5.375_{10} = 0101.0110_2$ c.f. 1010110×2^{100} floating point
  - Uses integer operations → faster, lower resource
- Make good use of pipelining

Matrix Maths Implementation

- Maths can be tricky in the FPGA...
  - Low-level
  - Latency, resource constrained
- Fixed-point arithmetic
  - For speed, resource saving
  - Rounding errors, limited range
- Custom division
- 18, 25, 35 bit variables
  - To efficiently use DSP resources
- Fully pipelined
  - Can take a new state & stub on *every clock cycle*

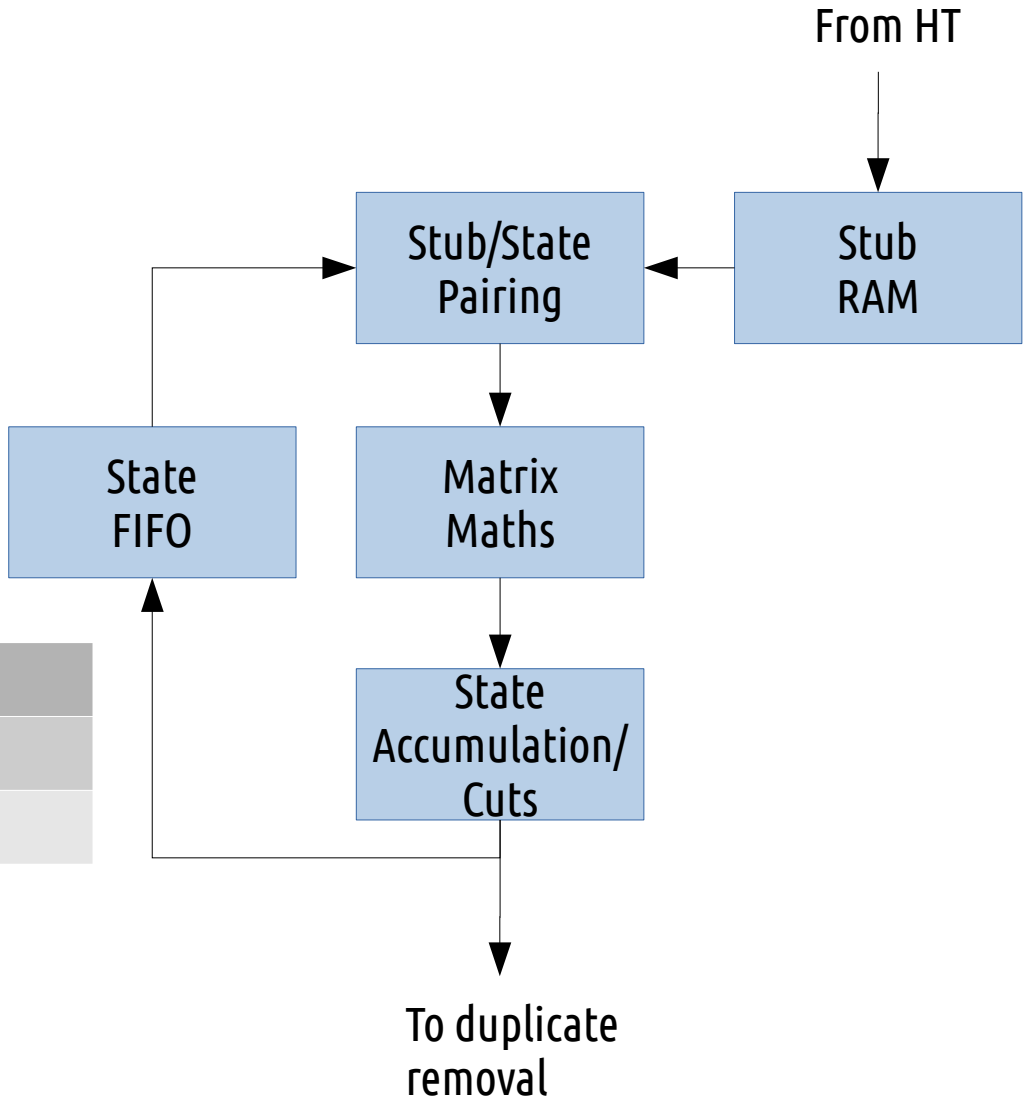
- Used MaxJ higher level language
  - Automatic pipelining and scheduling
  - Fixed-point support
  - Optimisation of constant operations
- Easier to design a complicated algorithm with such a tool
- Good integration with software running on CPU

Kalman Filter in MaxJ



Worker node

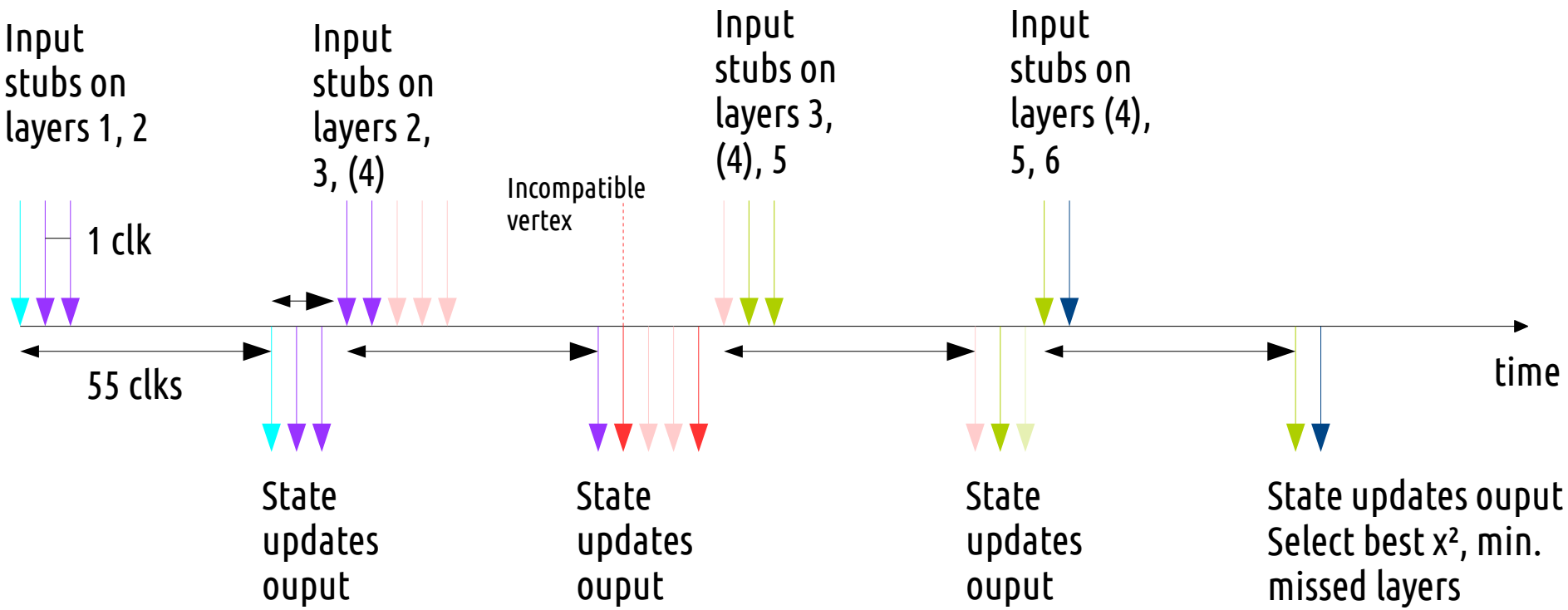
- Manage states/stubs into filter
- Updated states written into a FIFO
  - After x^2 ordering, cuts (vertex, min. p_T)
- Address book of IDs finds the next stub(s)
- Multiple candidates simultaneously processed



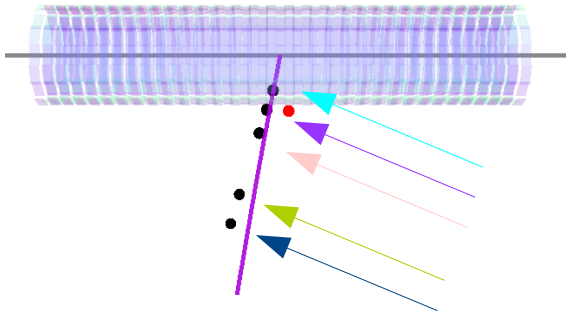
Item	DSPs	LUTs	BRAMs
Number	71	5520	24.5
% Virtex 7	2%	1%	1.7%

- Low resource usage → 36 parallel nodes per MP7

Control Flow – filling the pipeline



- 'Combinatorial explosion' mitigated thanks to pipelining
  - Filter-accumulate steps
- Total latency depends on number of stubs
  - Adds single clock cycle latency, not single iteration latency
- Space to fit other candidates simultaneously!

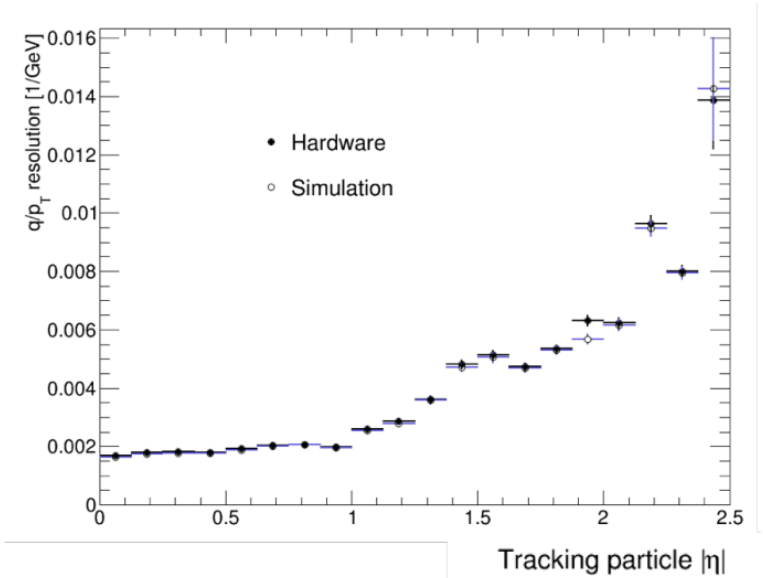


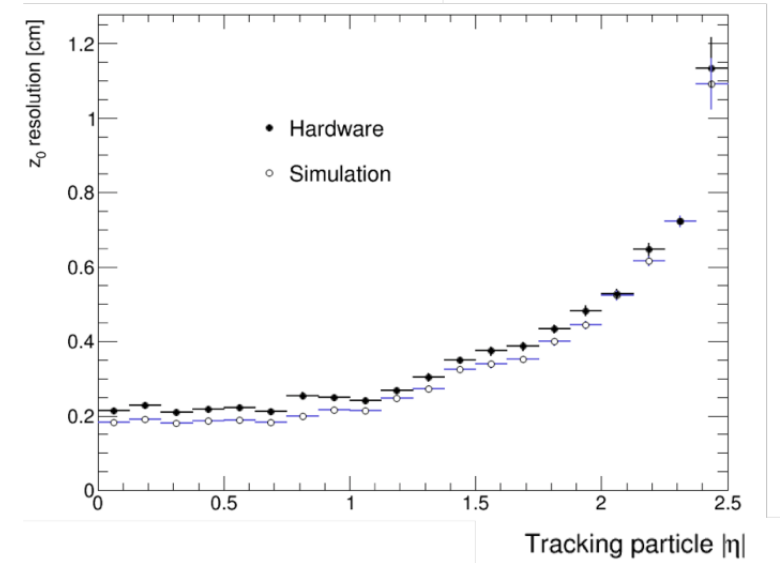
Performance

- Using $t\bar{t}$+200PU Monte Carlo:
- % $t\bar{t}$ tracks found:

HT	97.1%
KF	95.1% (97.9% of HT tracks)

- 1.62μs processing cut-off chosen
  - <0.1% lost efficiency due to time-out $t\bar{t}$+200PU
- 'Simulation' resolutions for floating point simulation



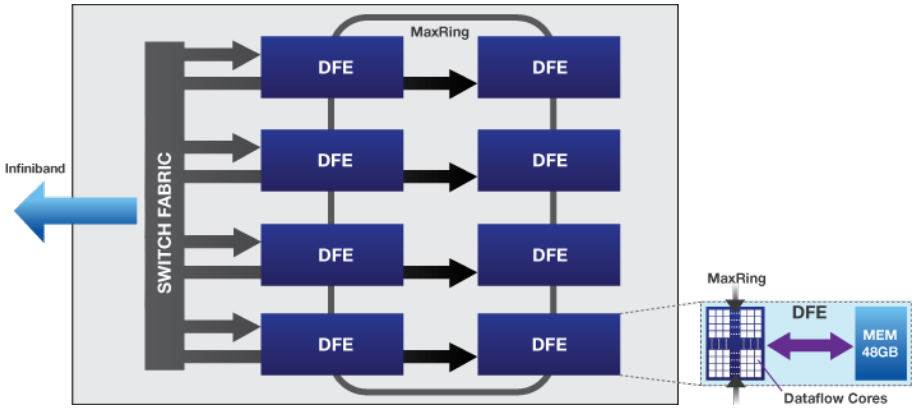


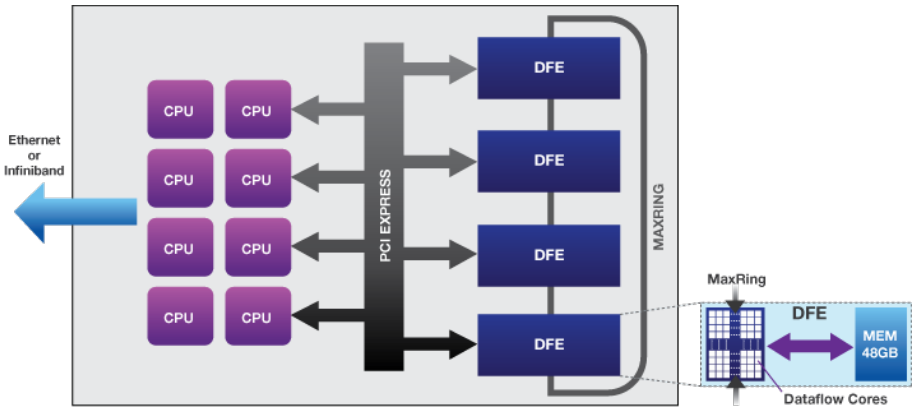
NB can be improved with extra bits

HLT - FPGA coprocessor concept

- Offload compute heavy tasks to FPGA
- Current devices contain 1000s of multipliers
- Can we use this Kalman Filter core to accelerate HLT tracking?
- Same (or similar) code for L1 and HLT?
- e.g. FPGA coprocessing available with Amazon cloud F1 instance

Some configurations from Maxeler





(DFE = 'Data flow engine' = board with FPGA)

Conclusions

- FPGAs offer massive parallelism
  - Many resources for low latency
  - Pipelining for high throughput
- Demonstrated a 1.6μs latency Kalman Filter track fit & clean
  - Part of CMS UK Time Multiplexed Track Trigger demonstrator