Using MaxCompiler for High Level Synthesis of Trigger Algorithms

TWEPP 2016

Sioni Summers

Contents

- What is MaxCompiler?
  - How could it be useful in triggering?
- Case study: CMS Level 1 Calorimeter Trigger
  - Benchmarking VHDL vs MaxJ
- Case study: CMS Level 1 Track Trigger
  - Developing new algorithms

What is MaxCompiler?

- A compiler to execute Java (MaxJ) and generate output synthesisable to FPGAs
- MaxJ is at a higher level of abstraction than HDL
- Simulate or run hardware seamlessly with CPU applications
- Normally targetting Maxeler hardware
  - Permitted to obtain the VHDL for this work, and use with the MP7 board
  - May become widely available, e.g. through OpenSPL

Why MaxCompiler?

- Higher level of abstraction than VHDL/Verilog could yield:
  - More physicists able to develop firmware
  - Better code maintainability
  - Shorter development times
- For CMS:
  - High granularity calorimeter
  - Track Trigger

Tool flow

- Write Java (MaxJ)

```
public myKernel extends Kernel(){
  DFEVar x = io.input(“xi”, dfeInt(32));
  // Do things with x
  io.output(“xo”, x, dfeInt(32));
}
```

- Run Java
  - Produce VHDL, Coregen (Xilinx)
  - Produces .max file

MyKernel.vhd:
    Synthesis/Modelsim
MyKernel.max:
    Proprietary Sim.

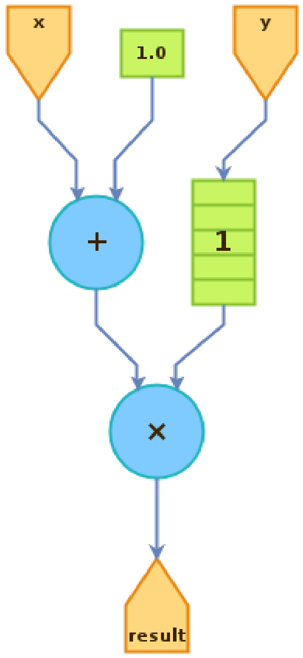
- Simulate / Synthesise

```
myResult = myDesign(myData);
```

MaxCompiler Features

- Pipelining (mostly) taken care of
  - Scheduling
  - Tool delays signals to accommodate latency of neighbouring paths
- Design graphs
  - Useful visualisation
  - Linked to source code

```
DFEVar x = io.input("x", dfeInt(16));
DFEVar y = io.input("y", dfeInt(16));
DFEVar result = (x + 1) * y;
io.output("result", result, dfeInt(16));
```



MaxCompiler Features

- Easy handling of numerics and bit-growth

```
DFEVar x = io.input('x', dfeFixMax(9, 3, SignMode.UNSIGNED)); // Fixed point
DFEVar y = io.input('y', dfeFixMax(9, 3, SignMode.UNSIGNED)); // Fixed point
optimization.pushEnableBitGrowth(true); // allow output to have more bits
DFEVar z0 = x + y;
DFEVar z1 = x * y;
optimization.pushFixOpMode(...);
DFEVar z2 = x * y;
```

- Choose strategies for selecting the output type
  - e.g. truncation/rounding, number of bits, fixed point location (avoid underflow, overflow or user defined)

Achieving low latency

- Default behaviour is to pipeline after every op.
- Desirable for high throughput, but can yield higher latency
- Programmer can instead remove registers where desired
- Use source annotations to evaluate
- Tool reports total latency and longest path

```
optimization.pushPipeliningFactor(0);
// Do operations with no pipelining
optimization.pipeline(var);
// Manually register var
optimization.popPipeliningFactor();
// Return to default
```

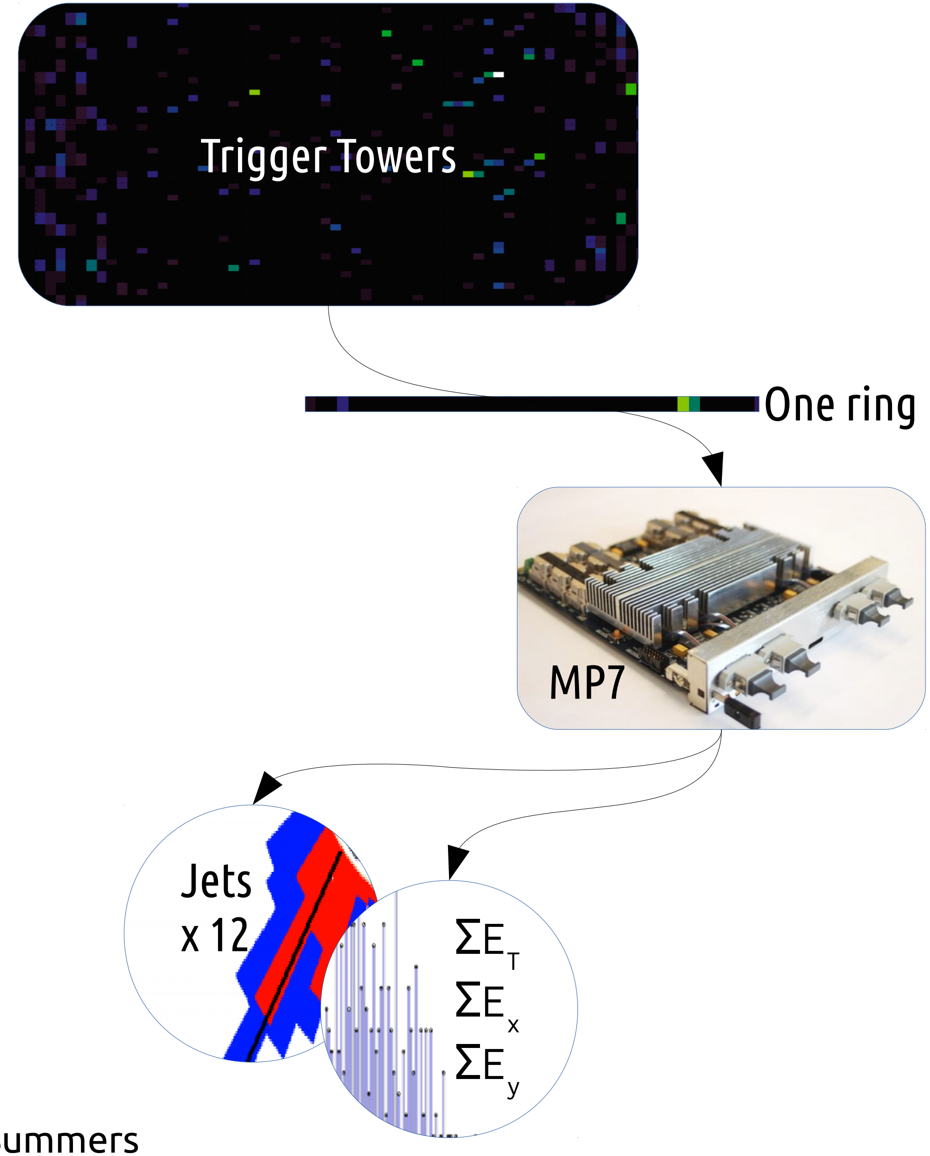
```
Col 1 - Longest path through line.
Col 2 - Sum of all node output latencies for
line.

5 | 5 | DFEVar x = io.input("x", dfeInt(16));
5 | 5 | DFEVar y = io.input("y", dfeInt(16));
4 | 4 | DFEVar result = (x + 1) * y;
```

CMS Level 1 Calorimeter Trigger

- 'Benchmarking' MaxCompiler
- Existing energy sum and jet algorithms re-implemented with MaxJ
  - Using alternative design patterns where appropriate
- Compare with original handwritten VHDL
  - Resources
  - Latency
  - Lines of code
  - Output equivalence

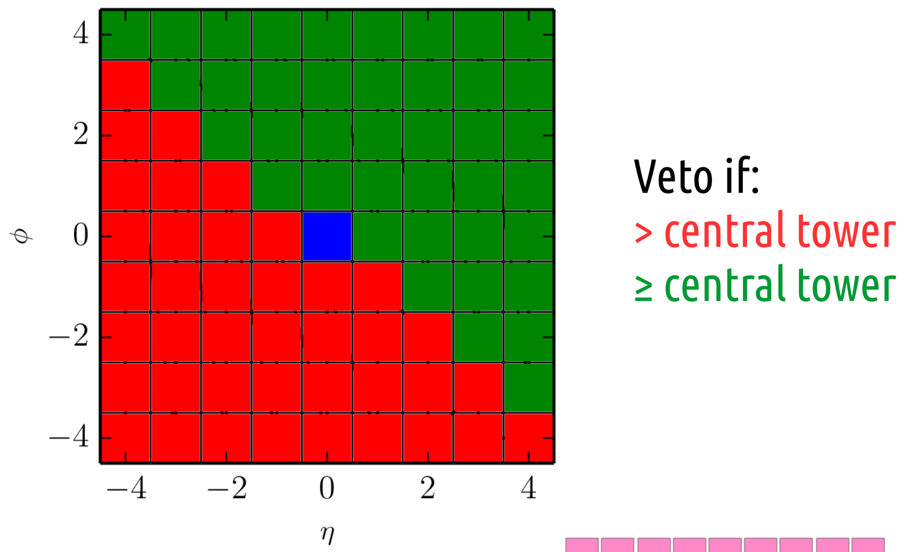
CMS Level 1 Calorimeter Trigger

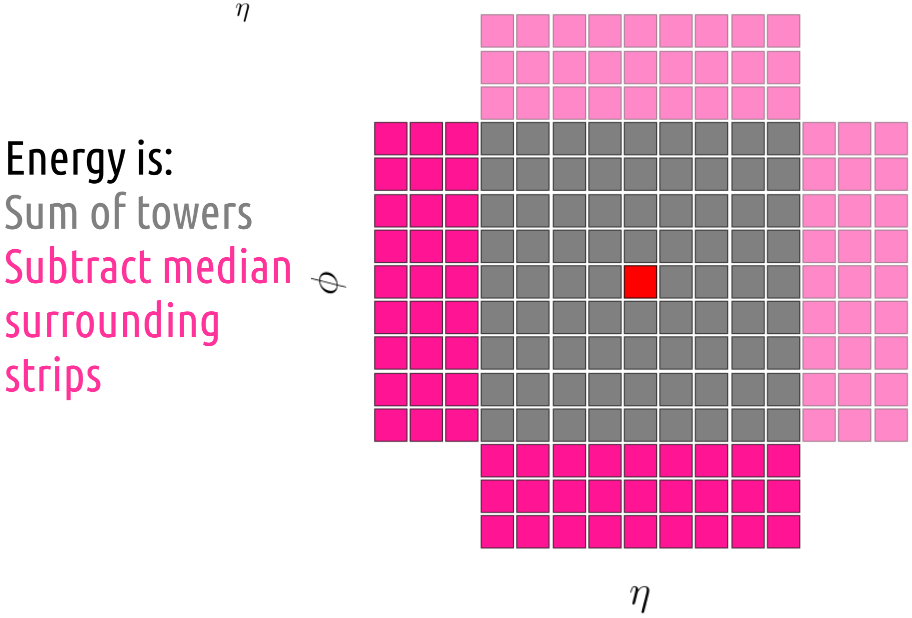
- Time Multiplexed Trigger
- Main Processor (MP) receives one ring (72 trigger towers) from each half per clock cycle
- Determine total event energy
  - Scalar & Vector
- Find objects: jets, e/γ, τ
  - With pileup subtraction



Jet Algorithm

- 'Sliding window'
- Sum the energy in a fixed sized grid
- Subtract local pileup estimate
- Jet centre is tower with highest energy in window
- Output 12 highest energy jets





Energy Sums: efficient computation

- Sliding window → looking across inputs and long pipeline
- Neighbouring candidate towers can share sums
- Write 'memory intensive' code
- Reuse (or fanout) of a computation must be written explicitly
- In this sense MaxJ is close to a HDL

Do:

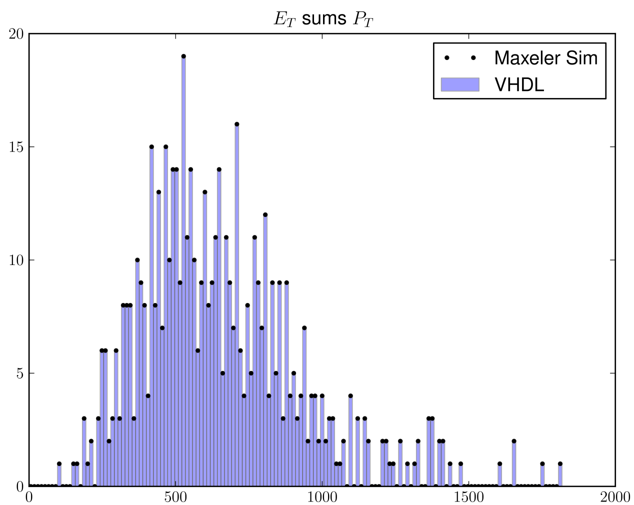
```
sum3x1[i] = tower[i-1] +
            tower[i]   +
            tower[i+1];

sum9x1[i] = sum3x1[i-3] +
            sum3x1[i]   +
            sum3x1[i+3];
```

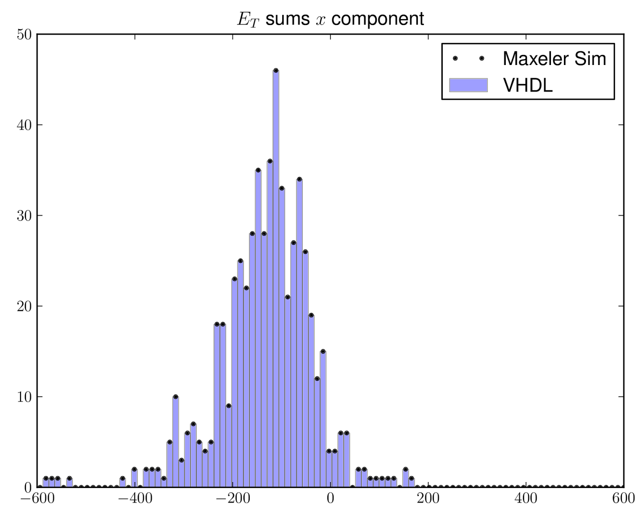
Not:

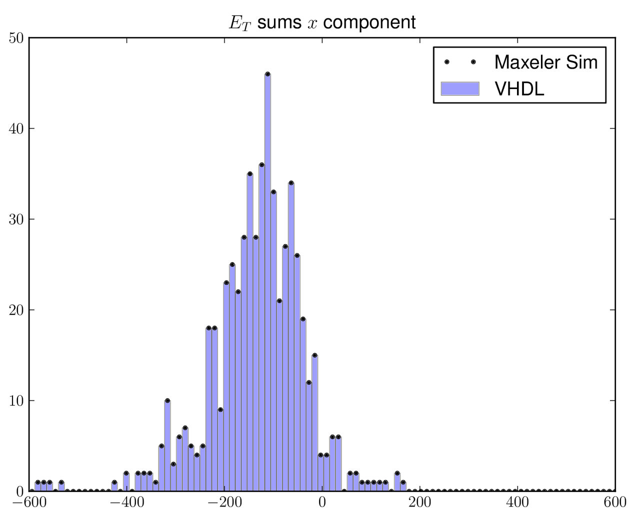
```
sum9x1[i] = tower[i-4] +
            tower[i-3] +
            …
            tower[i+3] +
            tower[i+4];
```

Version 2 uses 10x resources

Design Equivalence – E_T Sums

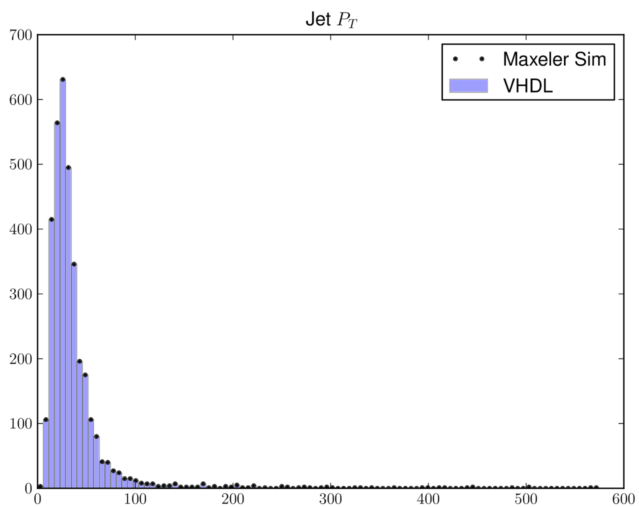


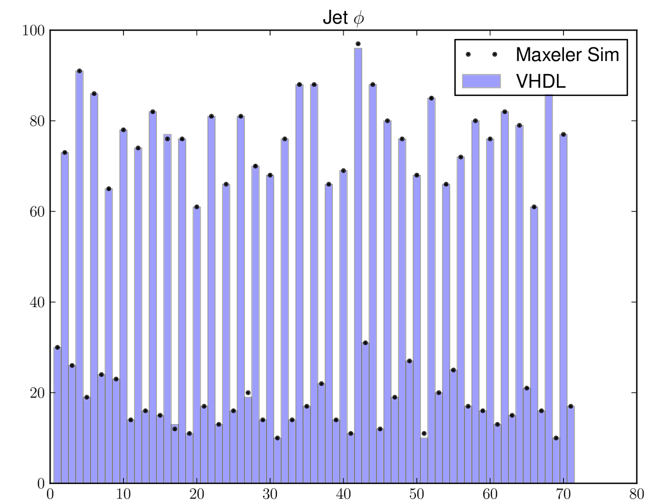


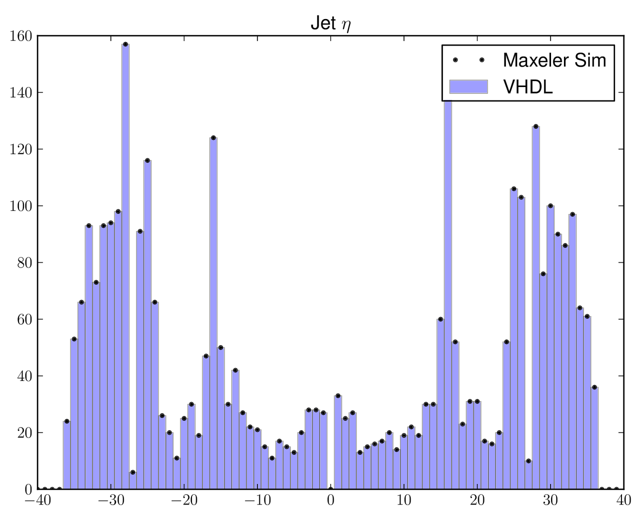


- E_T sums in 'trigger units' of bits at output
- Output shows exact match between VHDL and MaxJ implementations

Design Equivalence - Jets







- Comparisons of jet output: p_T and η, ϕ position, in 'trigger units'
- p_T and η are identical
- ϕ discrepancy: jets of the same p_T and η can be ordered differently after sort in ϕ
  - Source code for sort is very different between implementations!

Comparison

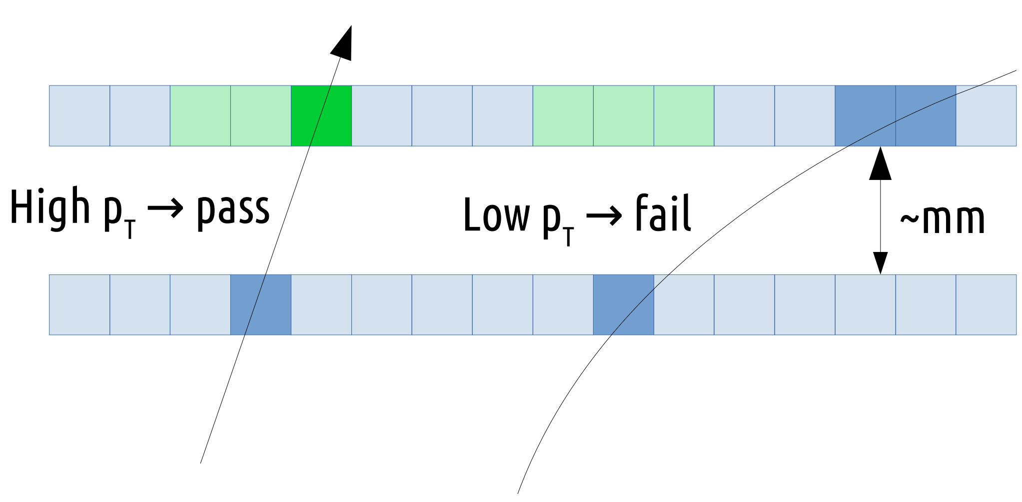
- 7% more LUT slices than the handwritten VHDL
  - All MaxCompiler cores have logic for asynchronous running
- DSP usage identical
- Half the lines of code
  - Suggests better maintainability

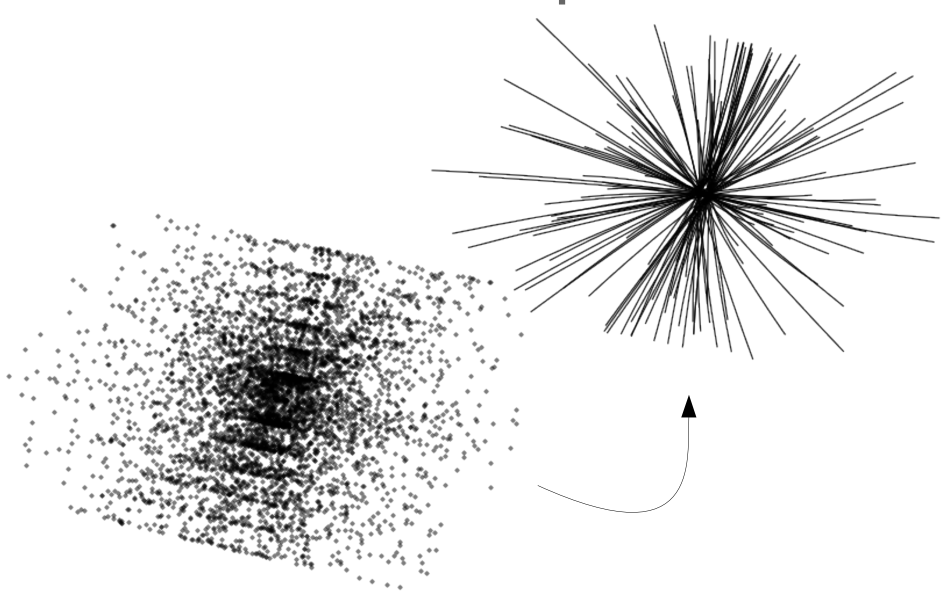
Item	Number (VHDL)	Number (MaxJ)
LUT Slices	95235	102508
Slice Regs.	153198	130072
DSPs	288	288
BRAM tiles	0	0
Lines of Code	3,000	1,500

- Built for a Virtex-7 690t using Vivado 2015.4, with MP7 core infrastructure

Case Study II: CMS Level 1 Track Trigger

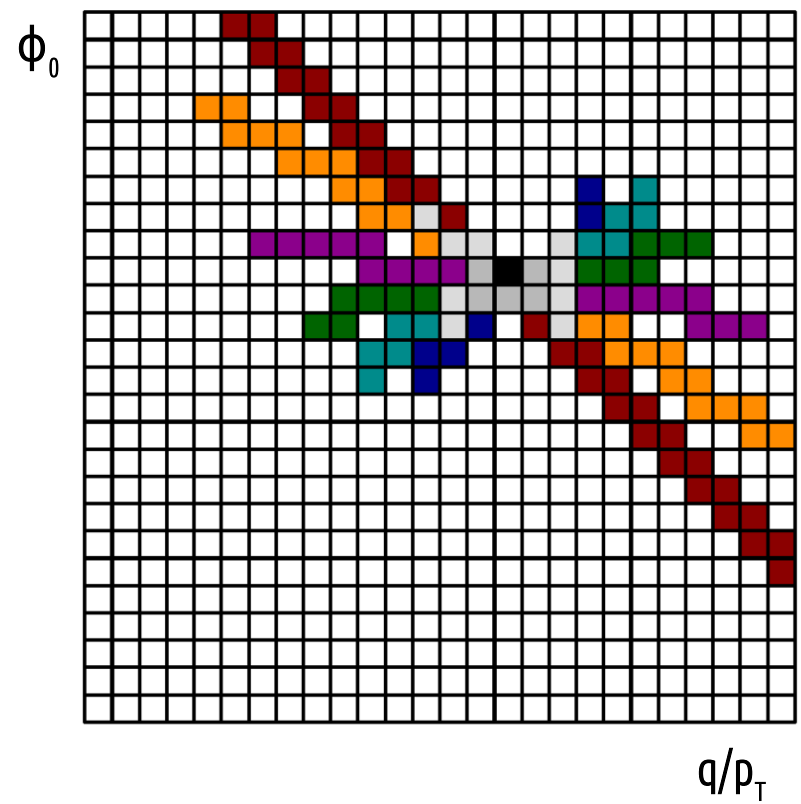
- At the High-Lumi LHC CMS will use tracking at Level 1 to cope with 140 pp interactions per bunch crossing
- 'p$_T$ module' will only read-out stubs above a threshold
  - 2 to 3 GeV under investigation
- Track trigger will reconstruct stubs to tracks in 10μs





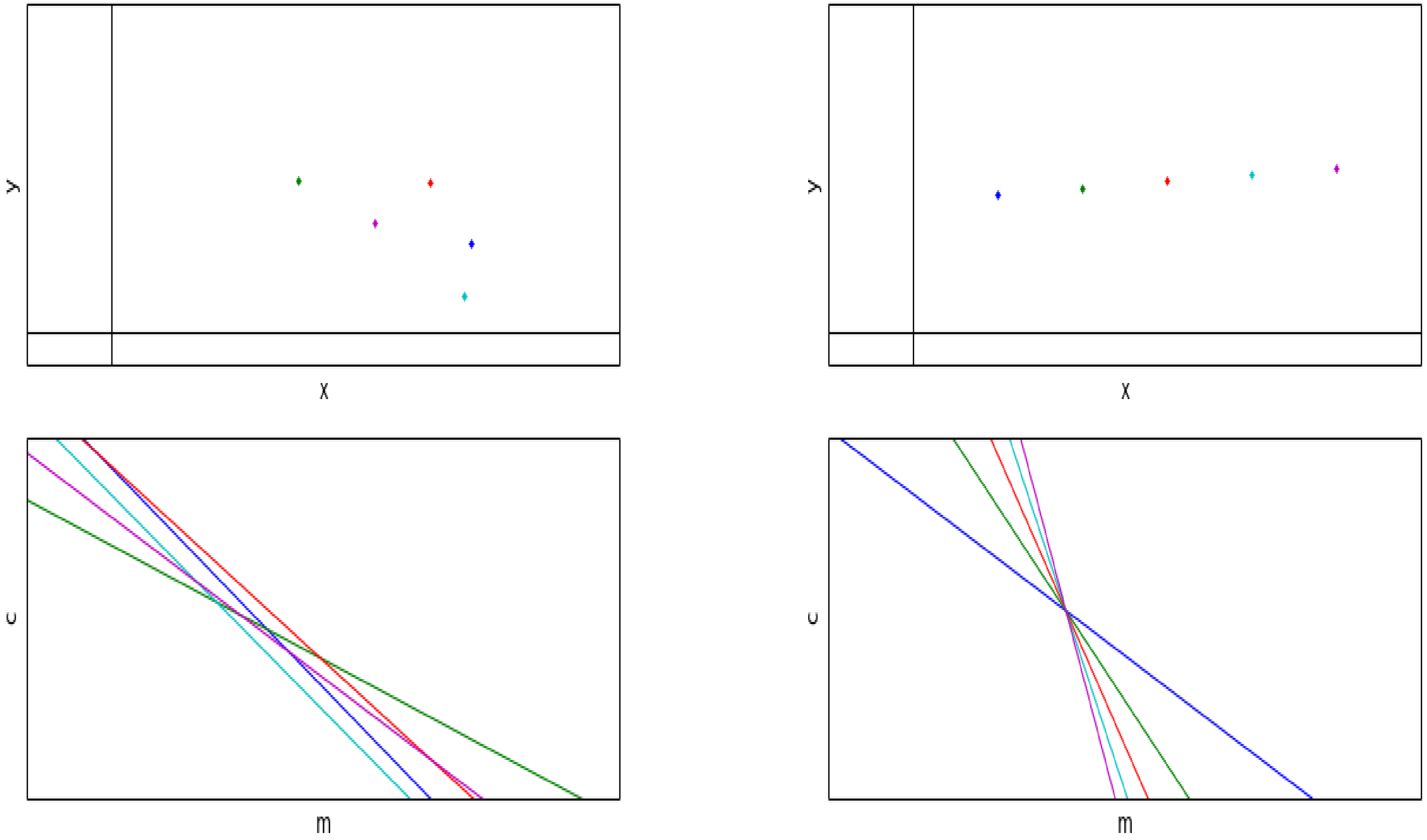
Time Multiplexed Track Trigger – Hough Transform (HT)

- Tracks above p_T threshold are approximately straight in r-ϕ plane
  - $\phi = (q/p_T)r + \phi_0$
- Bins are in (q/p_T) and ϕ_0
- Use stub bend to limit q/p_T range
- Do not use the z information
  - Keeps the histogram 2D
  - Leads to extra fakes
- Use the bin parameters as an estimate of trajectory for fit

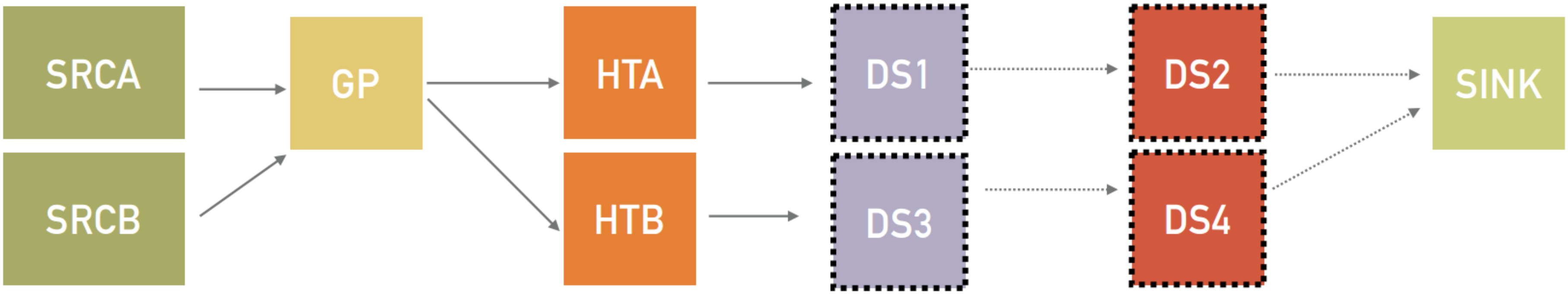


Hough Transform Example

$y = mx + c$



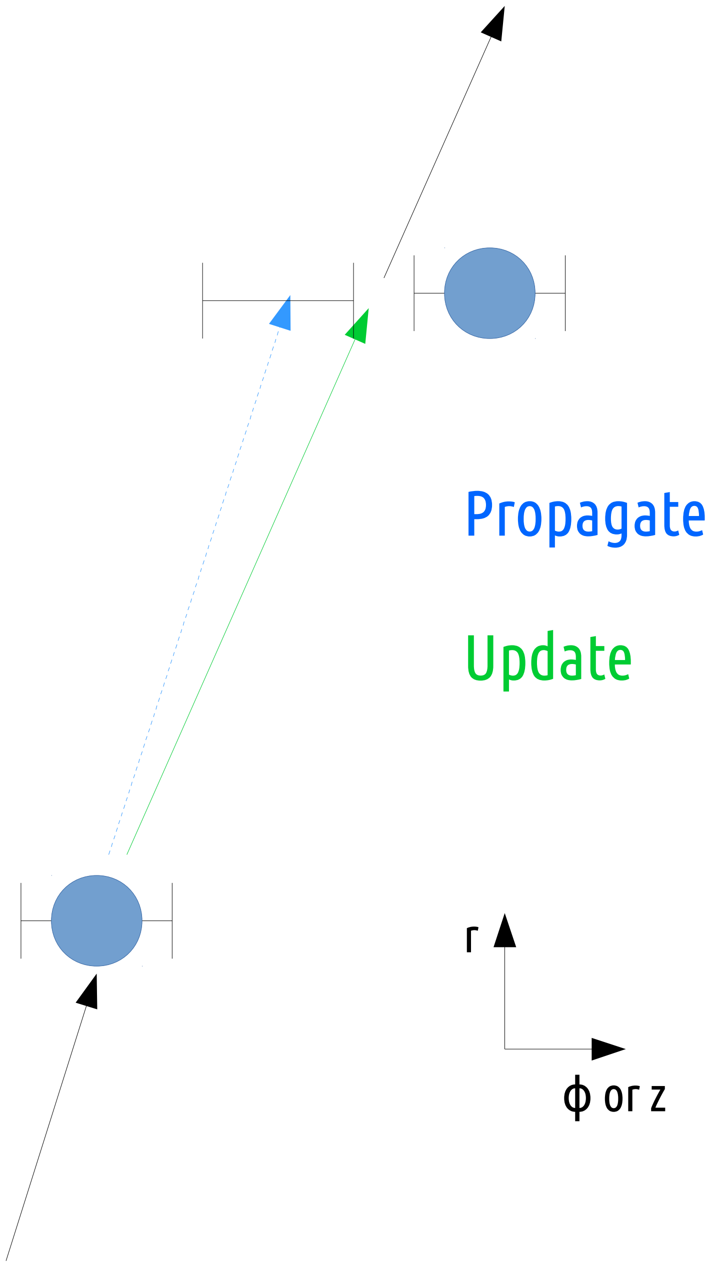
Time Multiplexed Track Trigger – CERN demonstrator



- Each box uses one MP7
- MaxCompiler used in GP (Geometric Processor) and TF (Track Fit) code
- GP and HT exist in hardware, 'DS' (downstream) under development

Kalman Filter

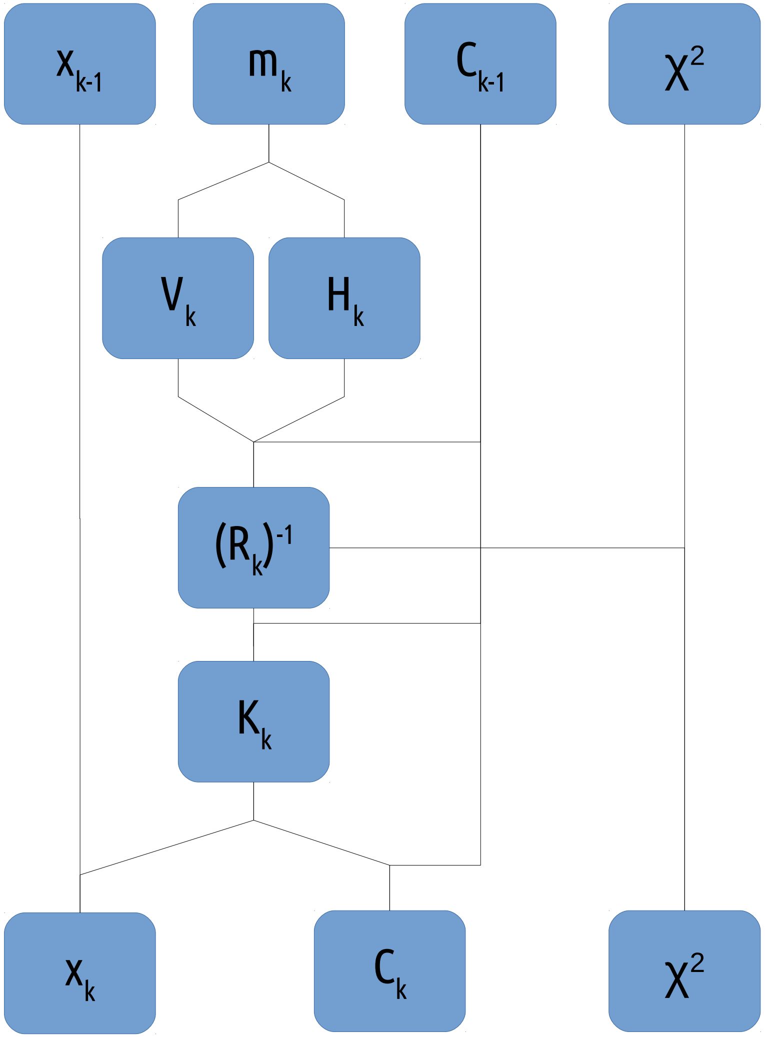
- Fit candidates found by the HT
  - Improve q/p_T, ϕ_0 resolution, obtain vertex, pseudo-rapidity
- Propagate track parameters inside-out
- Use stubs to update track parameters
- Ignore hits with high residuals
- Split multiple stubs into independent candidates
  - Improves resolution & purity
  - r-ϕ only tracking adds incorrect hits to tracks



Kalman Filter - equations

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

- 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



Kalman Filter in MaxJ

- Utilises the numerical manipulation features
  - Easy truncation
  - Fixed point arithmetic
- Allow scheduler to parallelise matrix maths
  - Optimise constant ops.

- Resource usage for one isntance of a 4 parameter state updater
- Chain together for a full fit
  - 6 for L1 track trigger

Item	Number (%)*
DSPs	44 (1.08)
BRAM Tiles	1 (0.07)
LUT Slices	3302 (0.76)
FF Slices	4131 (0.48)
Clock Cycles	40

* Percentage of a Virtex-7 690t

Conclusion

- MaxCompiler is a high level synthesis tool for designing complex FPGA firmware
- Results show minimal overhead in resource usage compared to VHDL implementation
  - CMS Calorimeter Trigger Study
- MaxCompiler in use for development of new firmware for a CMS Track Trigger demonstrator
  - Utilising numerical manipulation features

References

- The CMS Collaboration (S. Chatrchyan et al.), The CMS experiment at the CERN LHC, 2008 JINST 3 S08004, DOI: 10.1088/1748-0221/3/08/S08004
- Maxeler Technologies, *MaxCompiler white paper, url:* https://www.maxeler.com/media/documents/MaxelerWhitePaperMaxCompiler.pdf
- Kreis, B. et al, *Run 2 Upgrades to the CMS Level-1 Calorimeter Trigger*, Journal of Instrumentation, 11 (2016), url:http://stacks.iop.org/1748-0221/11/i=01/a=C01051
- G. Apollinari, I. Bejar Alonso, O. Bruning, M. Lamont, L. Rossi *High Luminosity Large Hadron Collider (HL-LHC) : Preliminary Design Report*, CERN, Geneva, 2015, DOI: 10.5170/CERN-2015-005
- Cieri, D. et al, *L1 track finding for a time multiplexed trigger*, Nuclear Instruments and Methods in Physics Research A 824 (2016) 268–269 DOI:http://dx.doi.org/10.1016/j.nima.2015.09.117