FPGA implementation of the HL-LHC CMS Drift Tubes Level-1 Trigger Algorithm

A. Navarro-Tobar, J. León Holgado

Ciemat

Plan de Recuperación, Transformación y Resiliencia

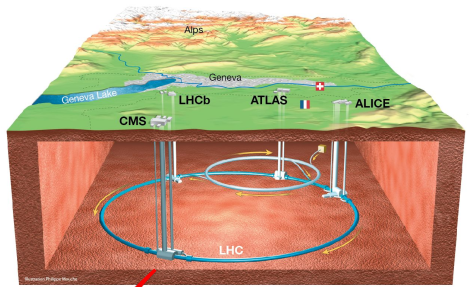
Funded by the European Union NextGenerationEU

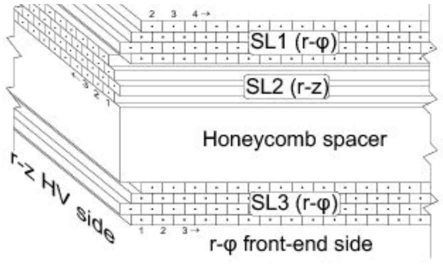
Proyecto PID2020-116262RB-C42 financiado por:

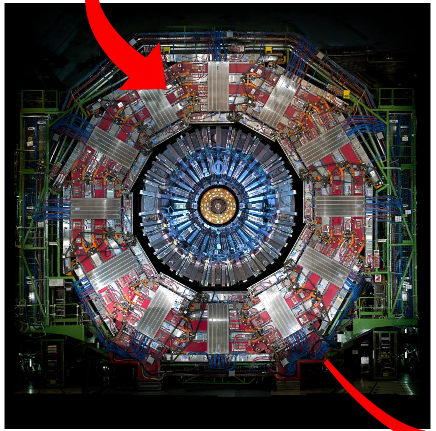
MINISTERIO DE CIENCIA E INNOVACIÓN

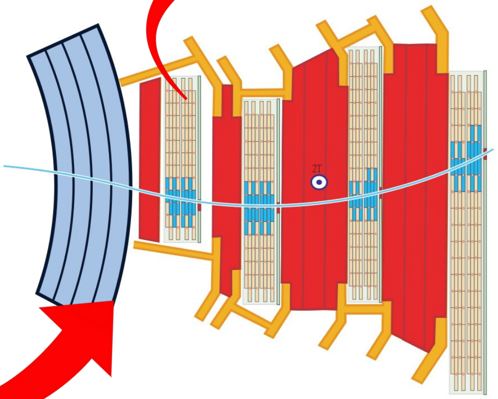
AGENCIA ESTATAL DE INVESTIGACIÓN

Introduction - Context



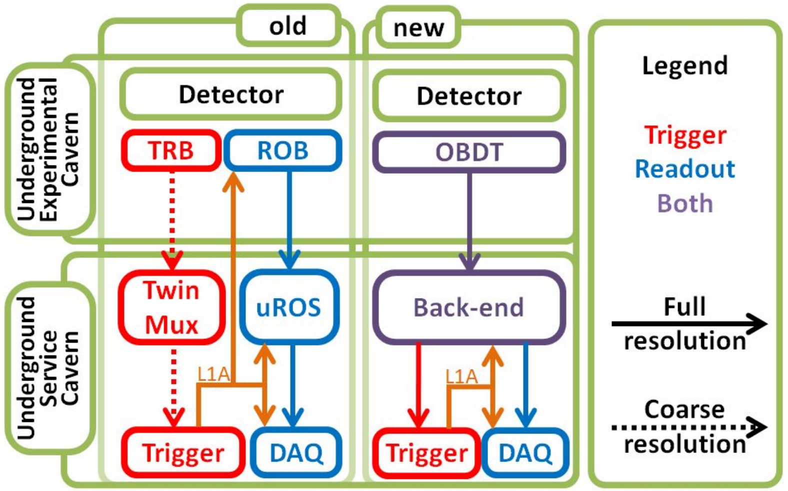






- Compact Muon Solenoid: general purpose detector at the LHC
- Muon Barrel: outermost part of the central wheels of CMS, identifies and tracks muons (minimum ionizing particles that can cross CMS iron yokes)
- Drift Tubes: 5 wheels x 12 sectors, 4 chambers, each contains 3 superlayers (4-layer blocks) provide information for phi (2x) and theta (1x) views
- DT subdetector consists on 172k Drift Cells. Drift time provides information on the position of muon (the farthest from wire, the later the hit arrives)

Introduction - Upgrade Phase 2



- Trigger/readout architecture changes
- Full data streaming to Underground Service Cavern (USC) of all DT data (no filtering)
- Trigger can done at USC (no radiation): bigger and faster FPGAs (before on-detector ASIC + USC V7)
- Achieve offline-grade (SW) performance at Level-1 Trigger (HW)

85x

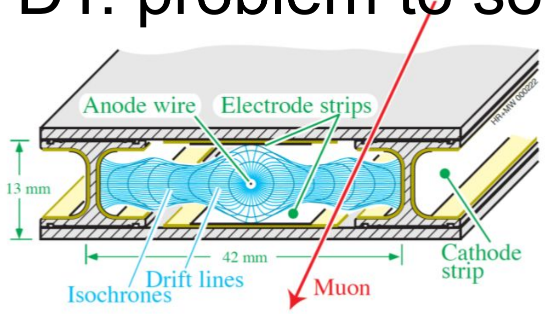
28M Logic Cells

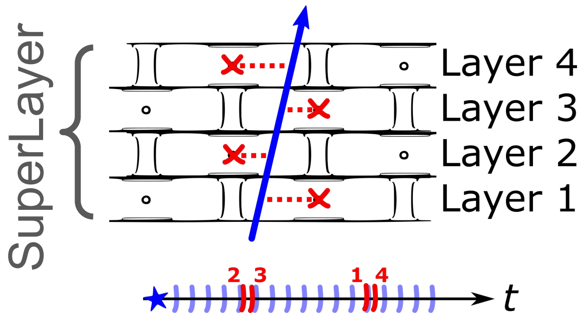
42x

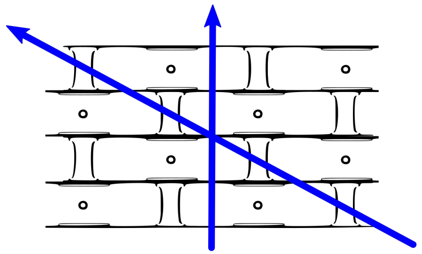
158M Logic Cells

OBDT poster at TWEPP by Ignacio Redondo: https://indico.cern.ch/event/1381495/contributions/6063106

DT: problem to solve

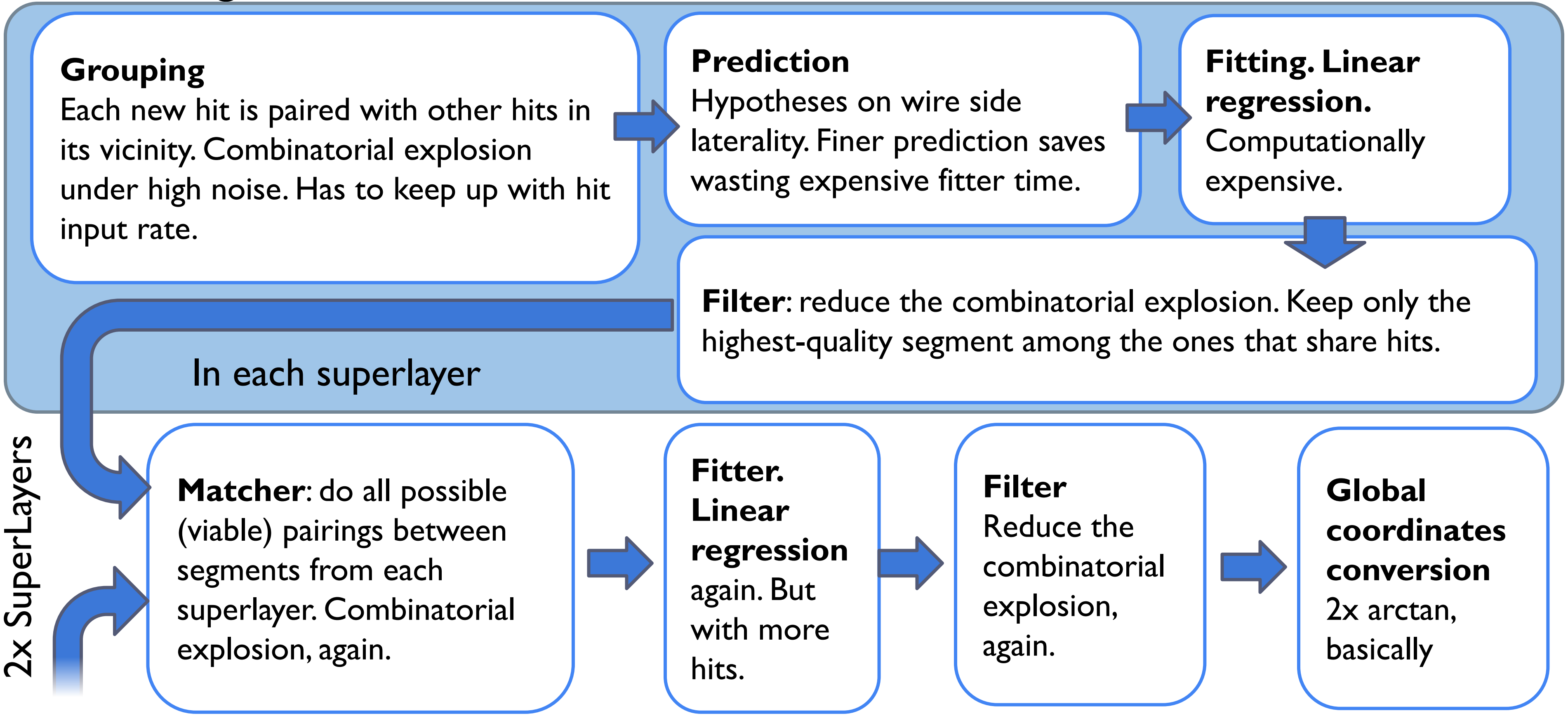






- Drift Time is up to 16 BXs (LHC Bunch Crossing 25ns)
- Muon hits can mix with hits from other muons, even from other events
- Laterality: we don't know a priori if a muon track passed to the right or left of the wire
  - Tracks with same hits but different laterality are called **ghosts**
- Sliding window, continuous processing: we cannot do event-based processing. A hit can form a group with hits 32 BX apart
- When pileup/noise increase → combinatorial explosion (number of possible pairings of hits grows very fast)
- This is for one superlayer: then we match each segment with the ones produced in the other 2 superlayers (new combinatorial explosion, new linear regression)

Our algorithm



Grouping
Each new hit is paired with other hits in its vicinity. Combinatorial explosion under high noise. Has to keep up with hit input rate.

Prediction
Hypotheses on wire side laterality. Finer prediction saves wasting expensive fitter time.

Fitting. Linear regression.
Computationally expensive.

Filter: reduce the combinatorial explosion. Keep only the highest-quality segment among the ones that share hits.

In each superlayer

2x SuperLayers

Matcher: do all possible (viable) pairings between segments from each superlayer. Combinatorial explosion, again.

Fitter. Linear regression again. But with more hits.

Filter
Reduce the combinatorial explosion, again.

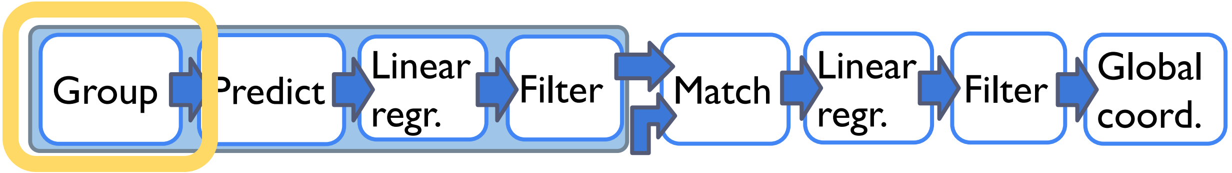
Global coordinates conversion
2x arctan, basically

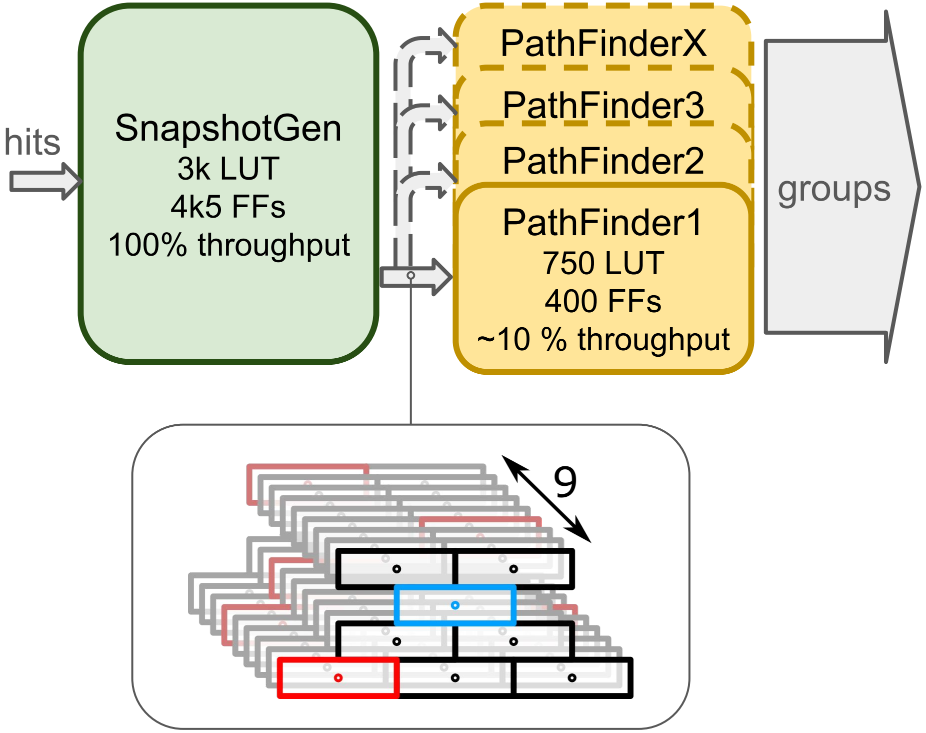
Constraints

- In each FPGA (VU13P)
  - Infrastructure (gigabit links take up much space)
  - 8 chambers (algorithm in previous slide)
  - Other algorithms to come
    - Showers: see Javier Prado's talk (next talk): https://indico.cern.ch/event/1381495/contributions/5988789
    - Theta and theta matching
    - RPC matching
- Maximum latency ~1 µs (40 BXs)
- We're very tight in resources, we're very tight in latency

⇒ We must maximize operation frequency to increase computing power. We aim for ~2 ns clock period (480 MHz = 12x LHC bunch frequency)

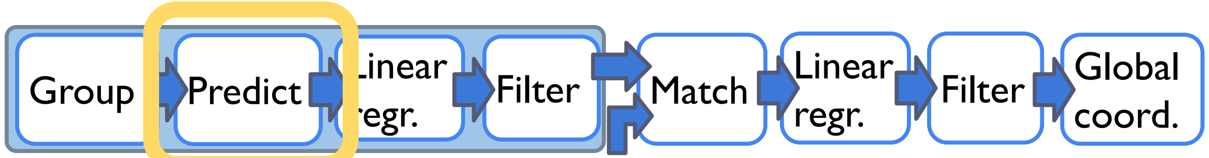
Grouping

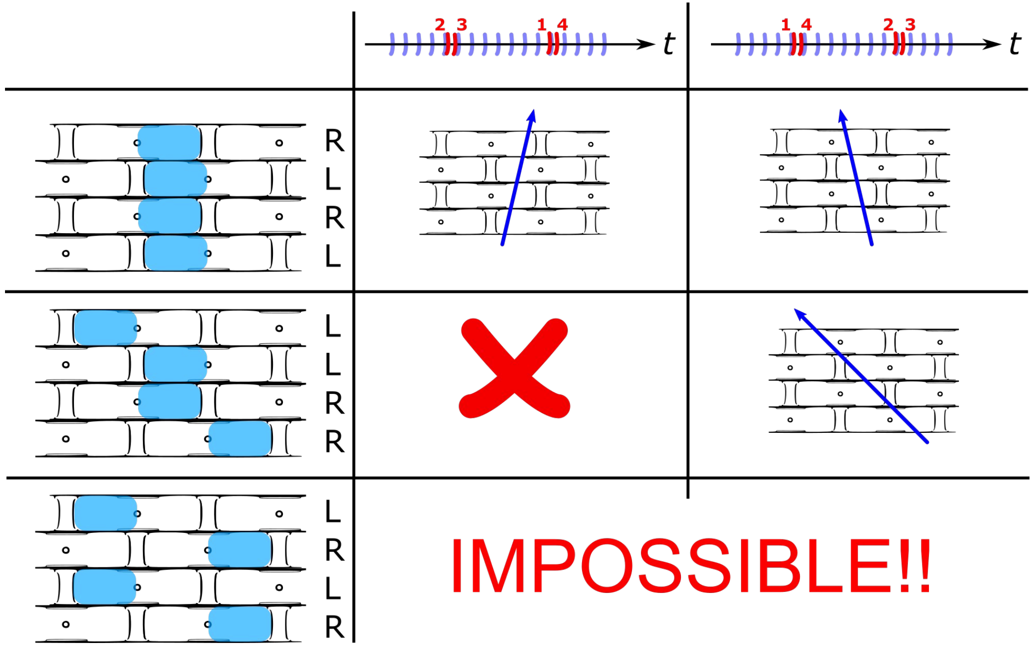




- Receives 1 hit/2ns, keeps history, delivers 1 group/2ns, grouped by time and spatial proximity
- Divide and conquer:
  - SnapshotGen: for each hit, delivers a snapshot, a collection of “photos” of the hits received in its vicinity in the past 16 BXs. Keeps up the pace with the input
  - PathFinder: for each snapshot, delivers all possible combinations of past hits with the new hit. Can take many clks to process one snapshot
- PathFinder doesn’t keep up with the input rate, several can be instantiated in parallel
- Processing stops when newly-generated groups would be out of maximum latency

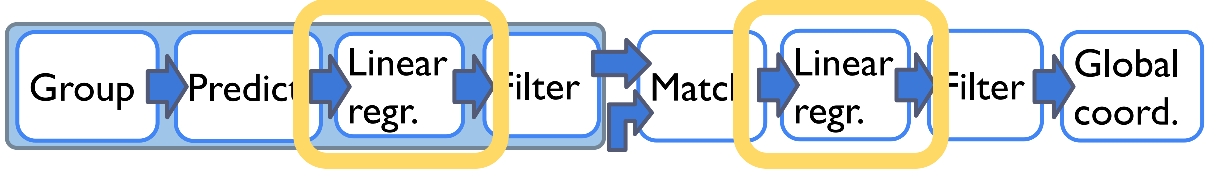
Prediction



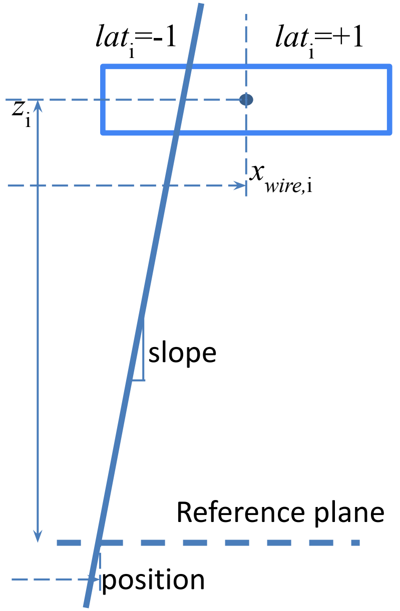


- Hit groups do not have laterality assignments → new explosion of workload
- Linear regression is expensive. We don't want to waste it with unviable candidates
- Pre-calculated Look-Up Table gives likely acceptable laterality combinations for a group of hits
- Input to the LUT is cell layout (geometry of cells involved) plus *coarsified* time (9 → 2 bits) of each hit
  - *Coarsification* is not just truncation or rounding: optimal thresholds result of bayesian optimizer
- Takes 600 LUT + 500 FFs
- Reduces average number of laterality combinations per group from 2.78 to 2.16

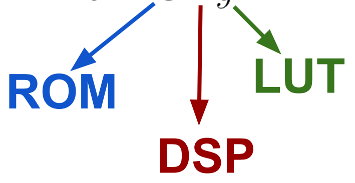
Linear regression



$$y \equiv \begin{bmatrix} x_{wire,1} + lat_1 \cdot t_1 \\ x_{wire,2} + lat_2 \cdot t_2 \\ \dots \\ x_{wire,8} + lat_8 \cdot t_8 \end{bmatrix}, X \equiv \begin{bmatrix} lat_1 & 1 & z_1 \\ lat_2 & 1 & z_2 \\ & \dots & \\ lat_8 & 1 & z_8 \end{bmatrix}, b \equiv \begin{bmatrix} t_0 \\ pos \\ slope \end{bmatrix}$$

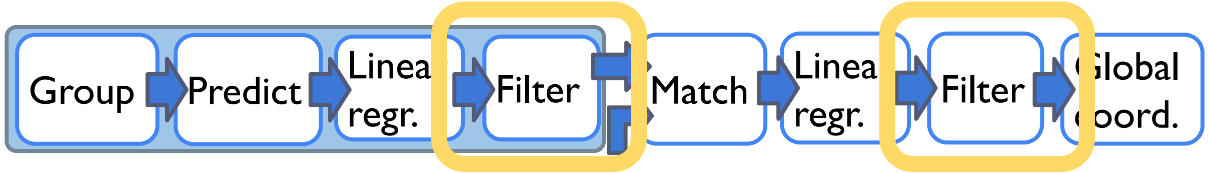


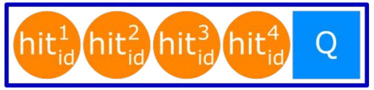
$$y = X \cdot b + \epsilon$$
$$b = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$
$$C \equiv (X^T \cdot X)^{-1} \cdot X^T$$
$$b = C \cdot y$$



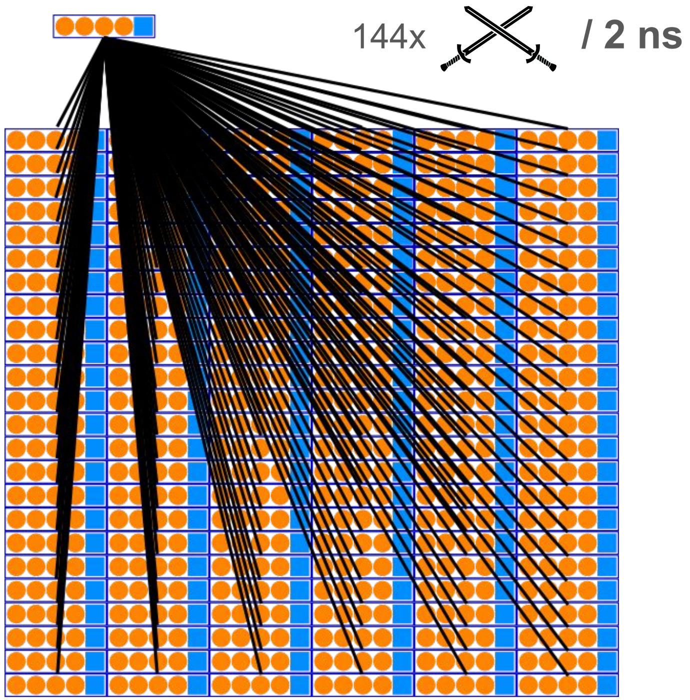
- Inputs: horizontal position of the wires, TDC value, laterality hypothesis
- Outputs: track parameters (T0, position, slope) and chi squared
- Ordinary Least Squares, matrix solution:
  - C matrix super expensive computationally
  - y matrix: much simpler, unavoidable (hit timestamps)
- We compute C matrix offline, store it in ROM
  - For 1 SL, ~50 rows, 200 bits each → distributed RAM
  - For 2 SL, ~1500 rows, 360 bits each → 20 Block RAM (UltraRAM doesn't allow ROM initialization)
- Computational load reduced to bare minimum
- Latency: 15 clock cycles
- 8 layers: 3k LUT, 4.5k FFs, 20 BRAM, 25 DSP

Filtering





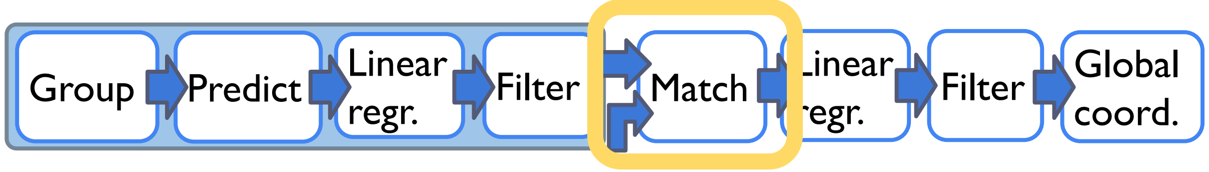
1 segment = 35 bits (*)

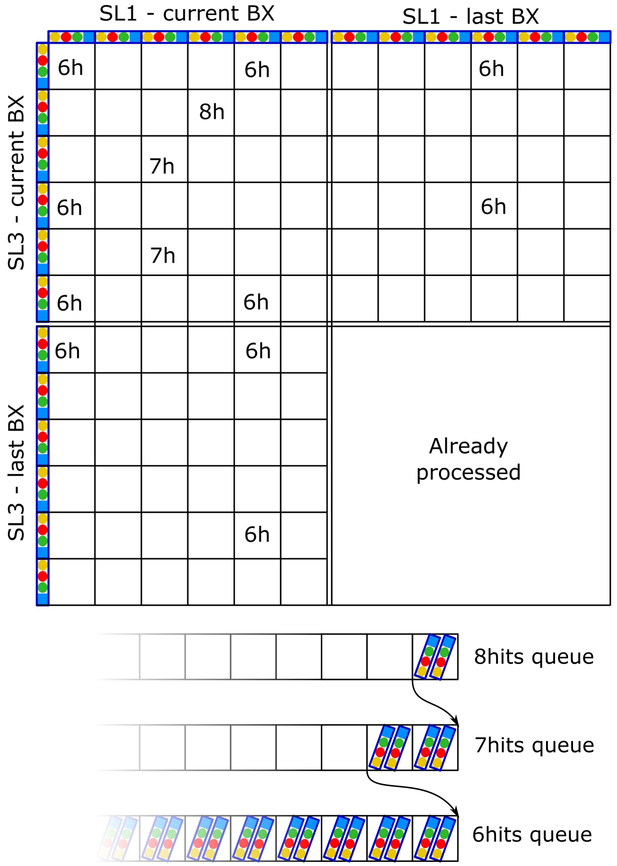


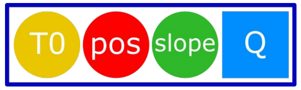
- Hits must be used once, in the best-quality segment
- Arriving segments are compared with 144 previous segments (24-BX deep history x 6 segments width):
  - May duel any (only if they share any hit)
  - May be killed by any
  - If it survives, it may kill any
- All must happen before the next segment arrives (2 ns!)
- Pipeline the input stage: before being written, each incoming segment spends 4 cycles calculating its "duel" result with all 144 pre-stored segments
- **But…** when a hit in the pipeline reaches the table, the table may have been changed already… Incoming hits "shoot bullets" at each other, these "bullets" also traverse the pipeline and reach its target in the required moment
- 6k LUT, 7k FFs, 4 BRAM

(*) Actually close to 300 bits, but unused are stored to RAM

Matching



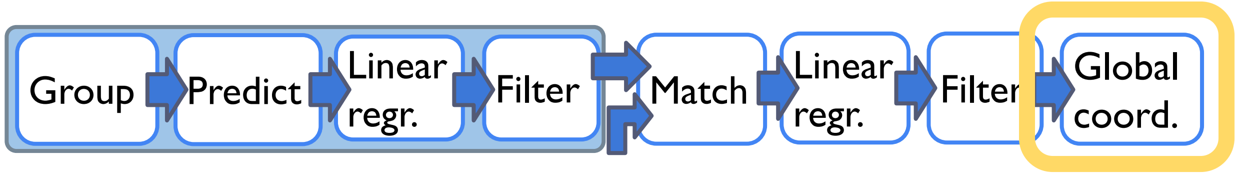


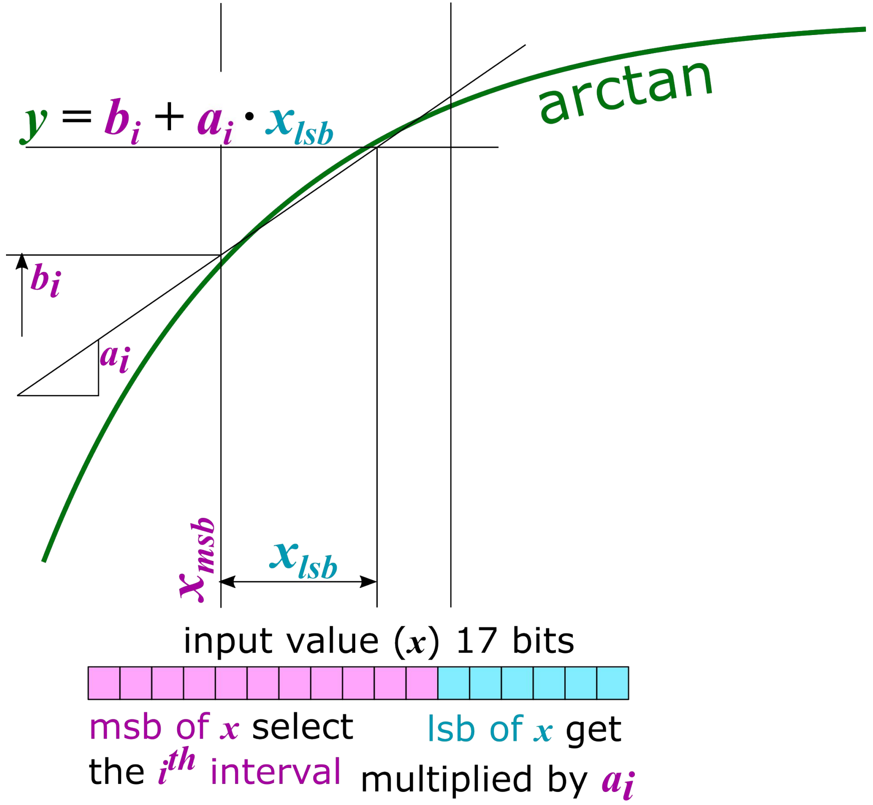


1 SuperLayer segment = 20 bits
(Actually close to 300 bits, but unused bits are stored to RAM)

- For ϕ view, pairs are made with segments from SL 1 and 3, from present and last BX
- 108 possible combinations: assess compatibility in t0, position, slope
- Only the best (linear regr. is expensive!) are selected for re-fitting
- Perfectly sorting data within each quality would be too expensive (and seldom make a difference)
- We classify the candidates in N queues according to predefined criteria (categories)
- Candidates are delivered starting from the highest-priority queue
- Currently 3 queues based only on number of hits of the combined segment
- Resources: 1k8 LUT, 1k2 FF (pairings), ~500 LUT/FF (each queue)

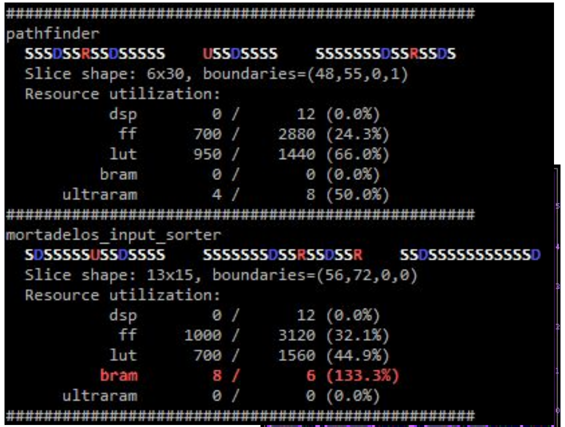
Global coordinates



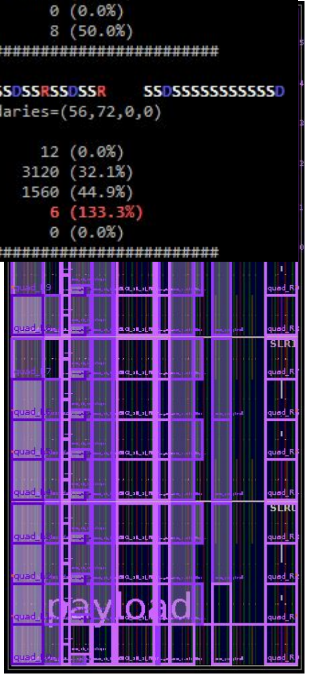

$y = b_i + a_i \cdot x_{lsb}$
arctan
b_i
a_i
x_{msb}
x_{lsb}
input value (x) 17 bits
msb of x select the i^{th} interval
lsb of x get multiplied by a_i

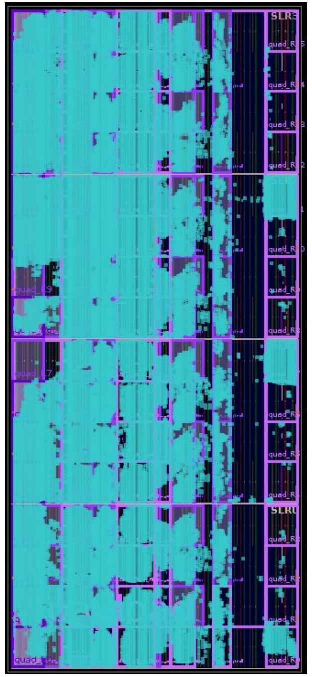
- Essentially 2 arctangent operations: one for position, one for slope
- High throughput, low latency, low resources…
- Piecewise linear approximation
- a and b coefficients stored in RAM (loaded at configuration, different for each chamber)
- ≤1 LSB approximation error
- Latency 4 clock cycles
- 500 LUT, 500 FF, 3 BRAM, 3 DSP

Timing closure



```
##################################################
pathfinder
  SSSDSSRSSDSSSSS    USSDSSSS    SSSSSSSDSSRSSDS
  Slice shape: 6x30, boundaries=(48,55,0,1)
  Resource utilization:
           dsp       0 /      12 (0.0%)
            ff     700 /    2880 (24.3%)
           lut     950 /    1440 (66.0%)
          bram       0 /       0 (0.0%)
      ultraram       4 /       8 (50.0%)
##################################################
mortadelos_input_sorter
  SDSSSSSUSSDSSSS    SSSSSSSDSSRSSDSSR    SSDSSSSSSSSSSSD
  Slice shape: 13x15, boundaries=(56,72,0,0)
  Resource utilization:
           dsp       0 /      12 (0.0%)
            ff    1000 /    3120 (32.1%)
           lut     700 /    1560 (44.9%)
          bram       8 /       6 (133.3%)
      ultraram       0 /       0 (0.0%)
##################################################
```





- Big design, high clk freq → challenge
- Initial naive approach, out-of-context each module with big margin (1.7 ns), failed:
  - OOC hides interconnection issues
  - Vivado placer does poor job on big designs (more random choices more likely to make bad ones)
- Regular placement constraining is cumbersome with our design → developed python library and scripts to auto-generate pblocks
  - More user-friendly, way easier to maintain, with the design still in development
- Helped identifying the netlist problems between modules (high-fanout, insufficient piping…) and gave vivado the boost it needed to get me those last 150 ps
- Presented at 1st FPGA Developers Forum (June’24)
  - https://indico.cern.ch/event/1381060/contributions/5923235
  - See Filiberto Bonini’s talk tomorrow 17:30
- Available at gitlab.cern.ch
  - https://gitlab.cern.ch/anavarro/ant_placer
  - Let me know if interested!

Final remarks

- DT/Muon Barrel algorithm in good shape, still long way until Run 4
- Around 15000 lines of VHDL code (and counting!) plus many python helpers, ROM generators, etc.
- Resources around 350k LUTs, 500k FFs, 1000 BRAM, 150 URAM, 850 DSP
- Current latency 20 BX (0.5 μs)
  - 12 BX processing
  - 8 BX configurable delay at the SL filter: margin for computational peaks + improve filtering
- Many different interesting/challenging problems had to be addressed, many optimizations made
  - Varied problems: computational, dataflow modules…
- Clock frequency 480 MHz
  - Very challenging, placement was key