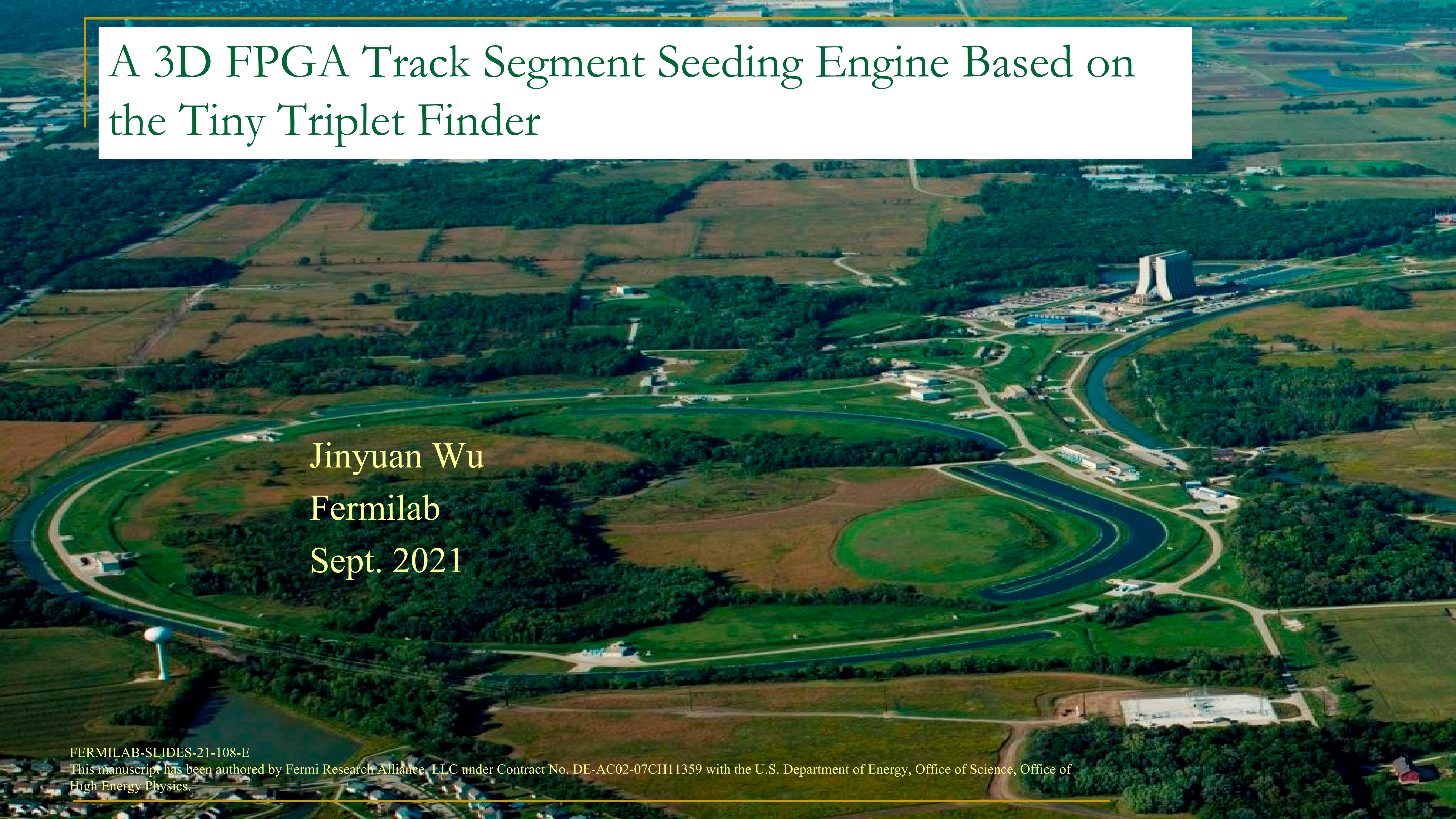


A 3D FPGA Track Segment Seeding Engine Based on the Tiny Triplet Finder

Jinyuan Wu

Fermilab

Sept. 2021

FERMILAB-SLIDES-21-108-E

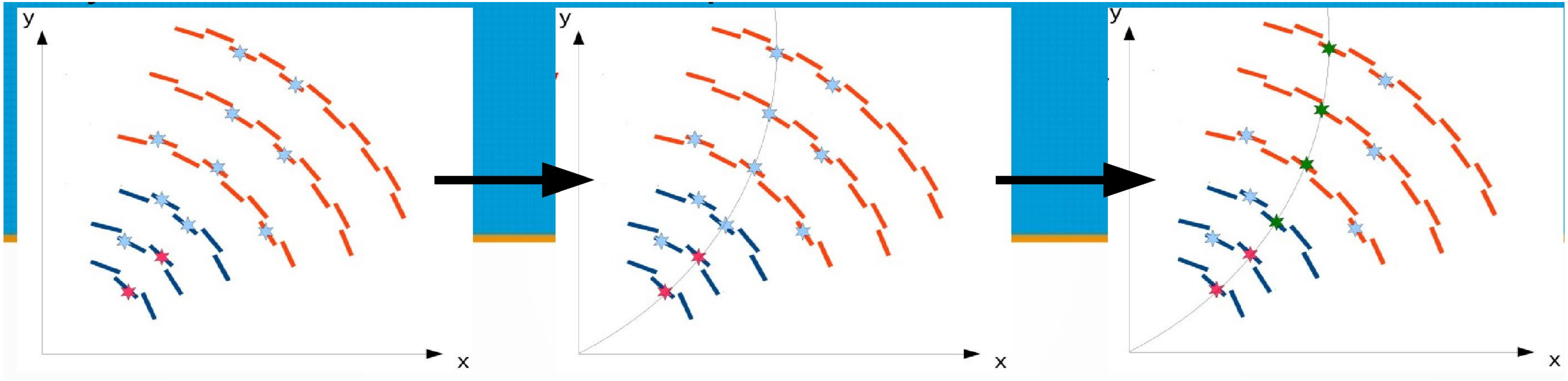
This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

Introduction

- An exercise of implementing a 3-hit 3-D track segment seeding engine based on the Tiny Triplet Finder is presented.
- It is intended to group hits into track segments and to feed into later Kalman filtering stage.
- The silicon resource consumption of the seeding engine is small, and it can be put into a low-cost FPGA.

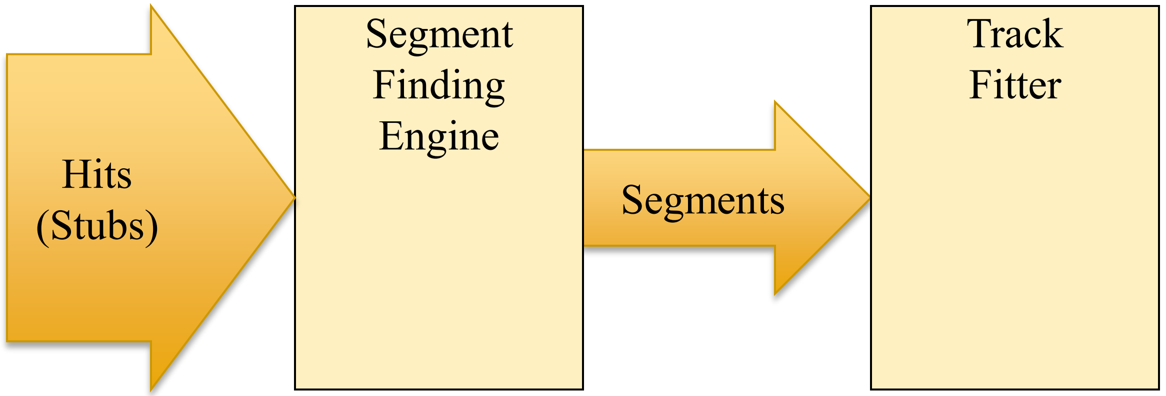
- *In our work, we use CMS L1 tracking Hybrid Algorithm as the example because it is challenging. But this work is not in official CMS project*

Tracking in Multilayer Detectors

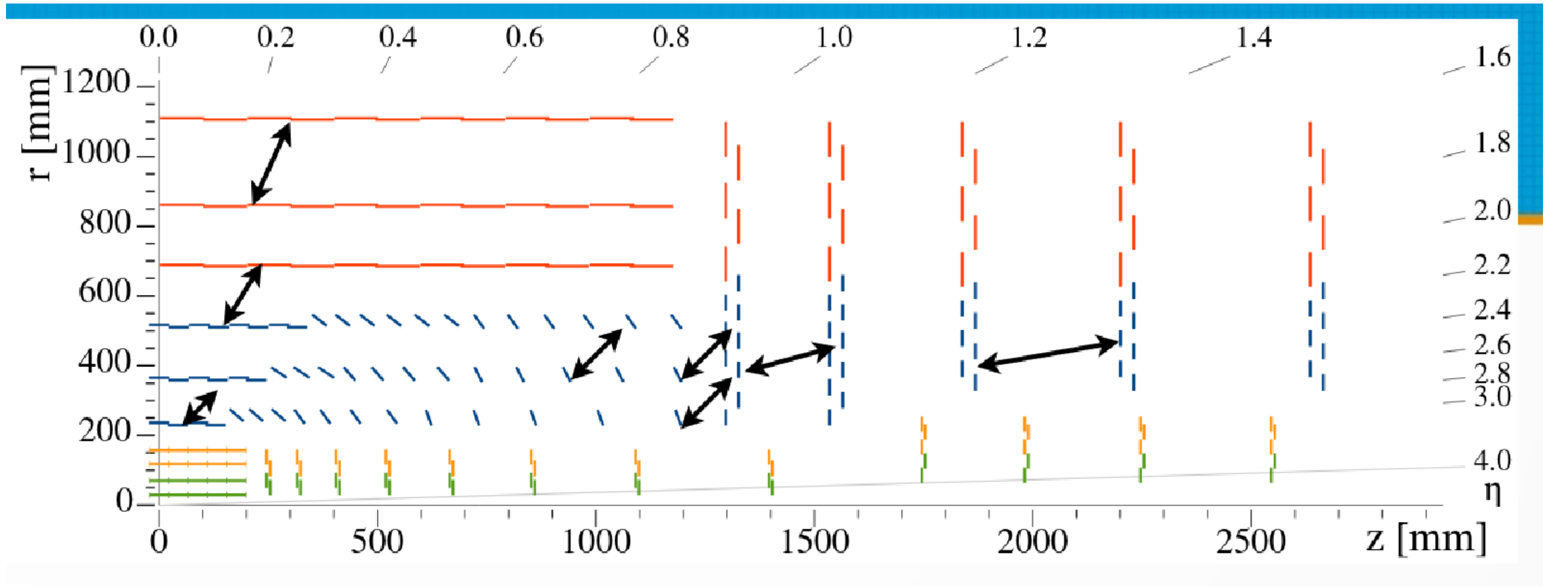


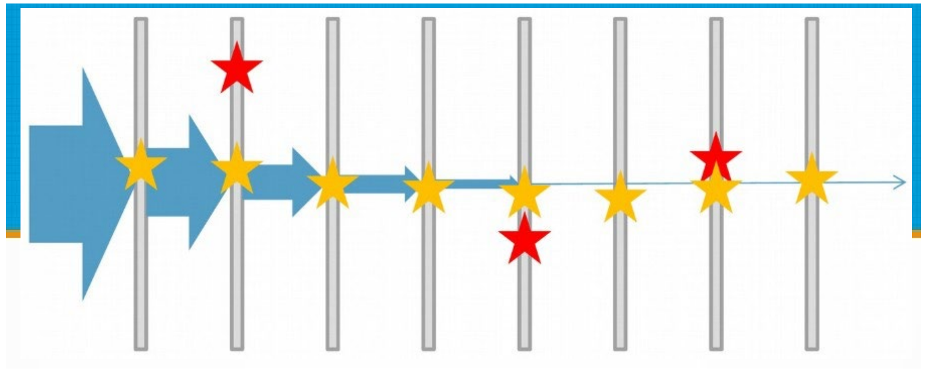
Andrew Hart et. al. 2020

- Well accepted tracking approaches take the following stages:
  - Hits or stubs are first grouped into track segments.
  - Track segments are fitted into tracks.



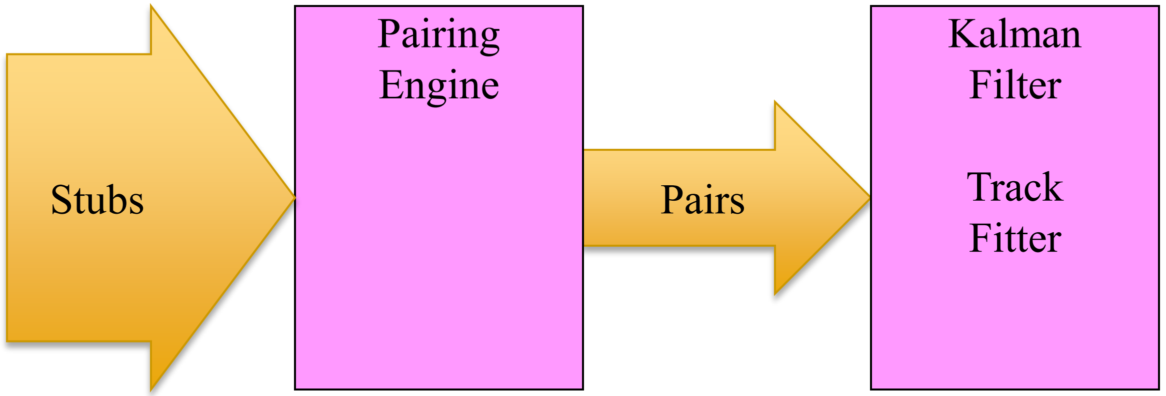
The CMS L1 Tracking Hybrid Algorithm





Andrew Hart et. al. 2020

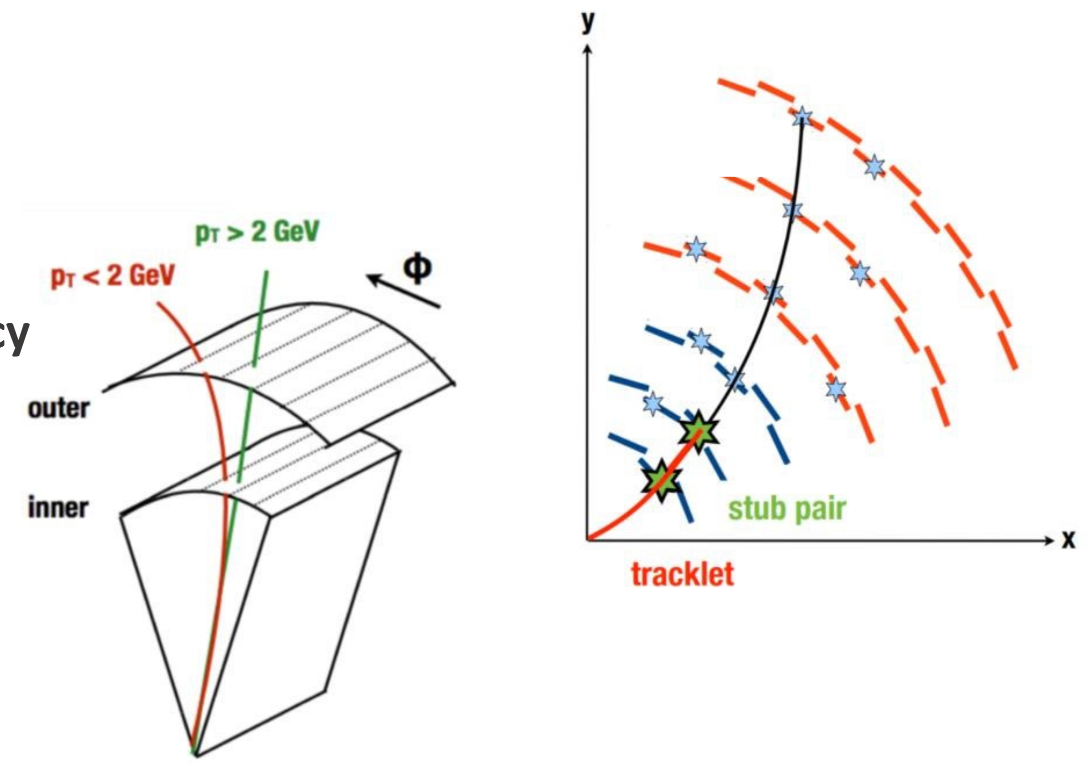
- Hits from different layers are first paired
- Track segments with paired hits are extended.
- Hits are fitted into tracks.



Pairing

Stub Organisation & Seeding

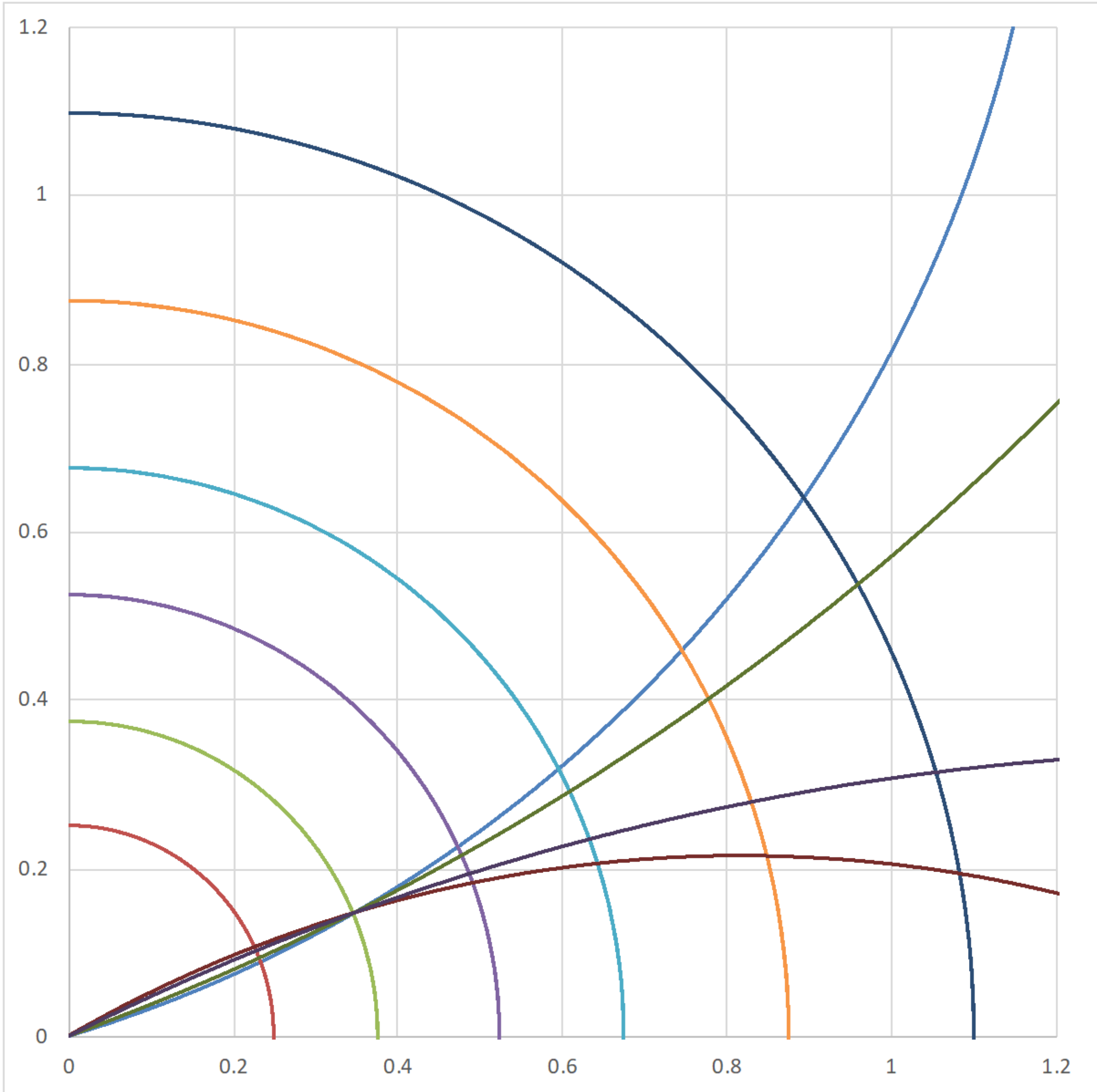
- Seeds (tracklets) are generated from **pairs of layers**
- Layer pairs selected to give full coverage in η, including **redundancy**
- Seeding step **massively parallelised** by internal geometrical partitioning & stub organisation (Virtual Modules - VMs)

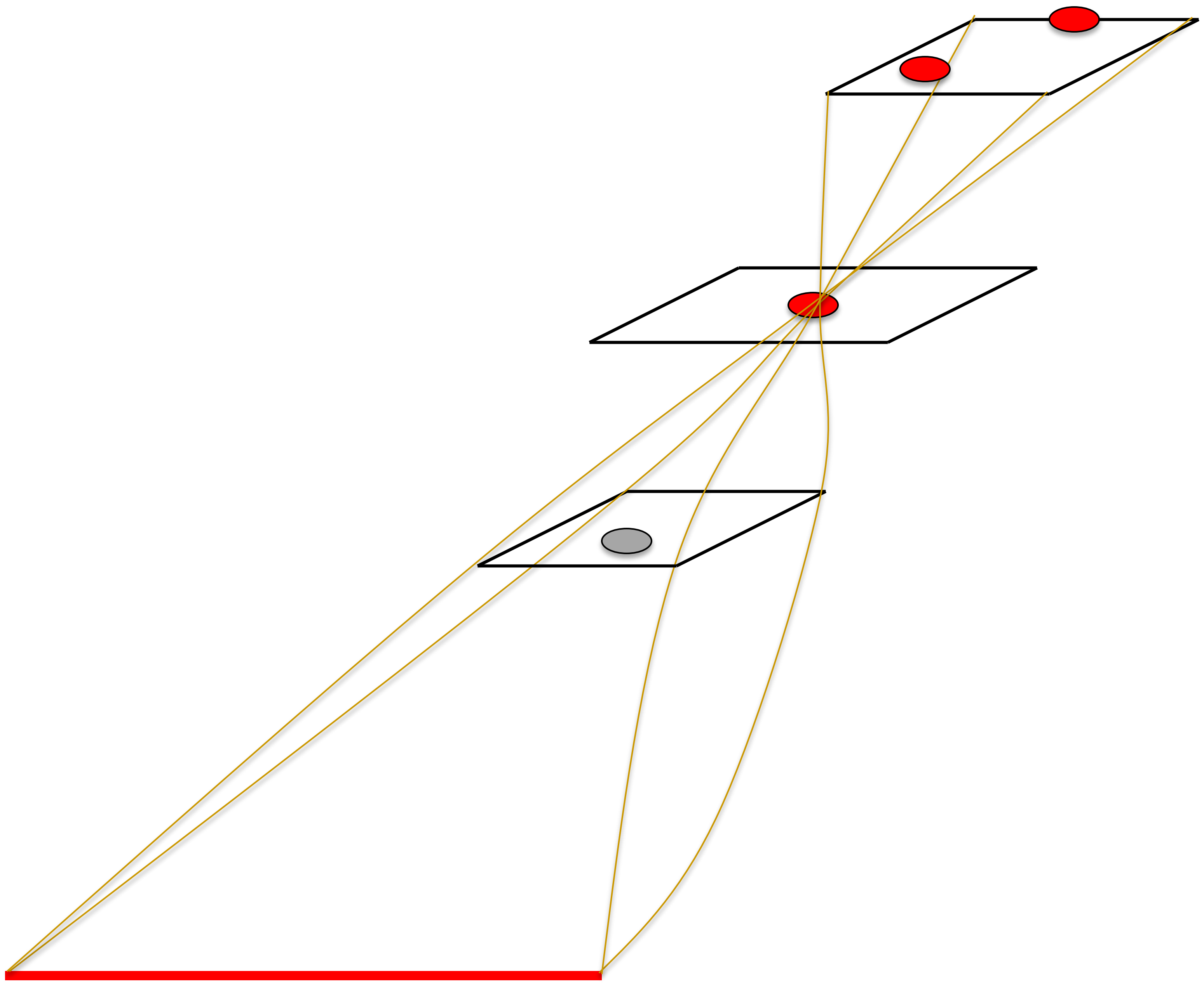


Mark Pesaresi et. al. ACES 2020

- The CMS L1 Tracking Hybrid Algorithm: find a **pair** => find third => fit and find others

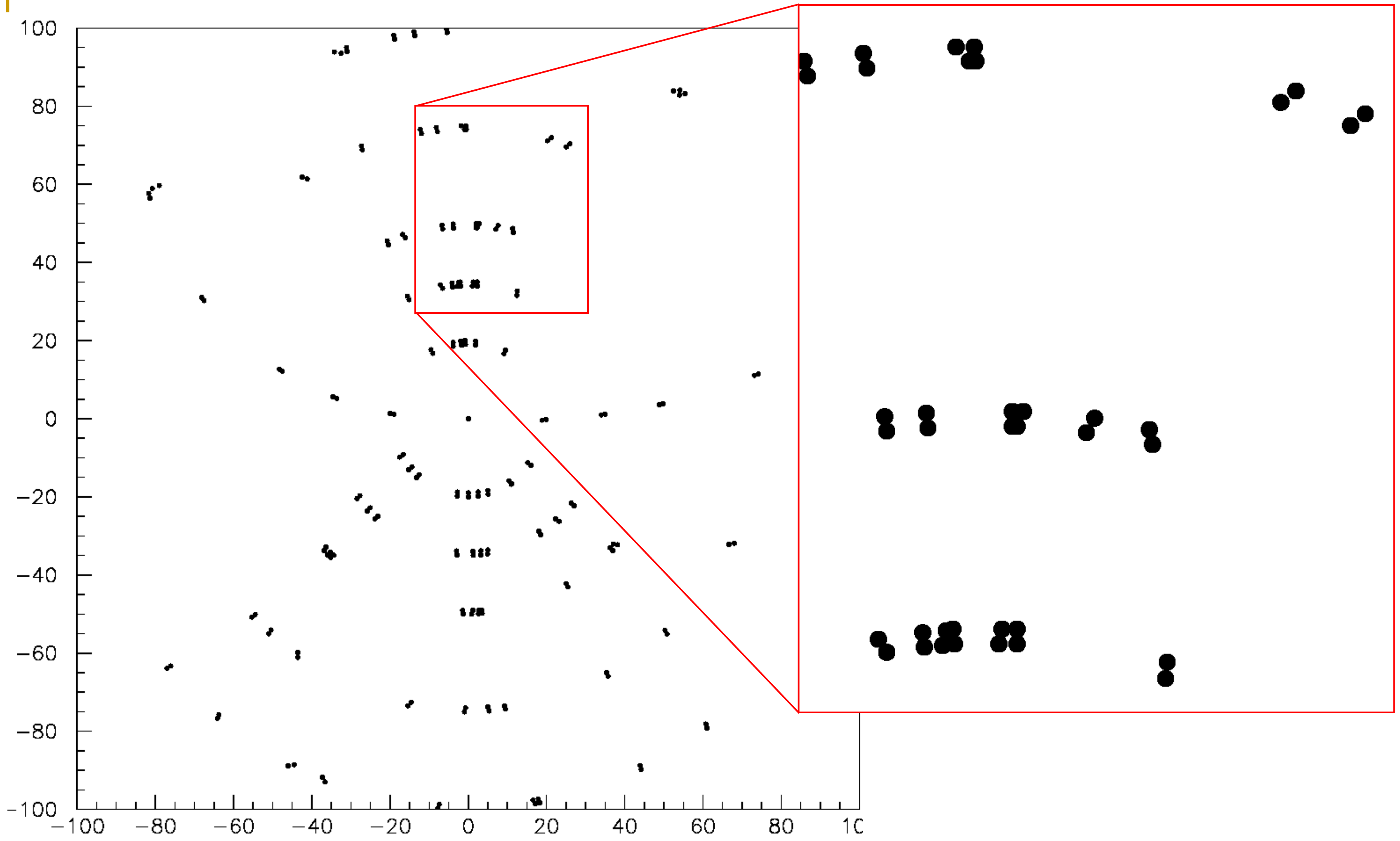
Pairing Stubs





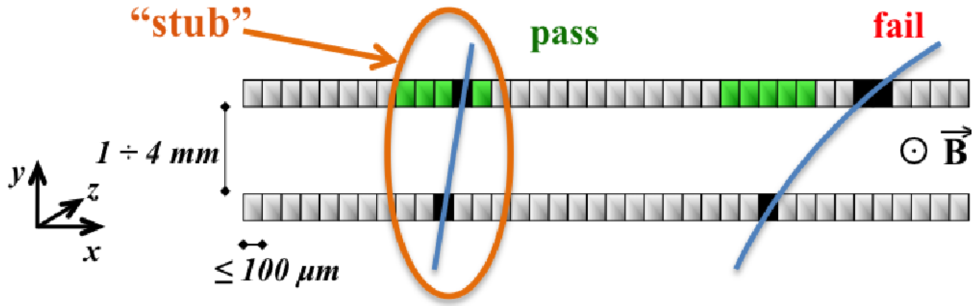
- For any stub in Layer 2, a range in phi and z must be searched in Layer 1 or/and Layer 3.
- Fake pairs may occur.

Pairing or Triplet Finding



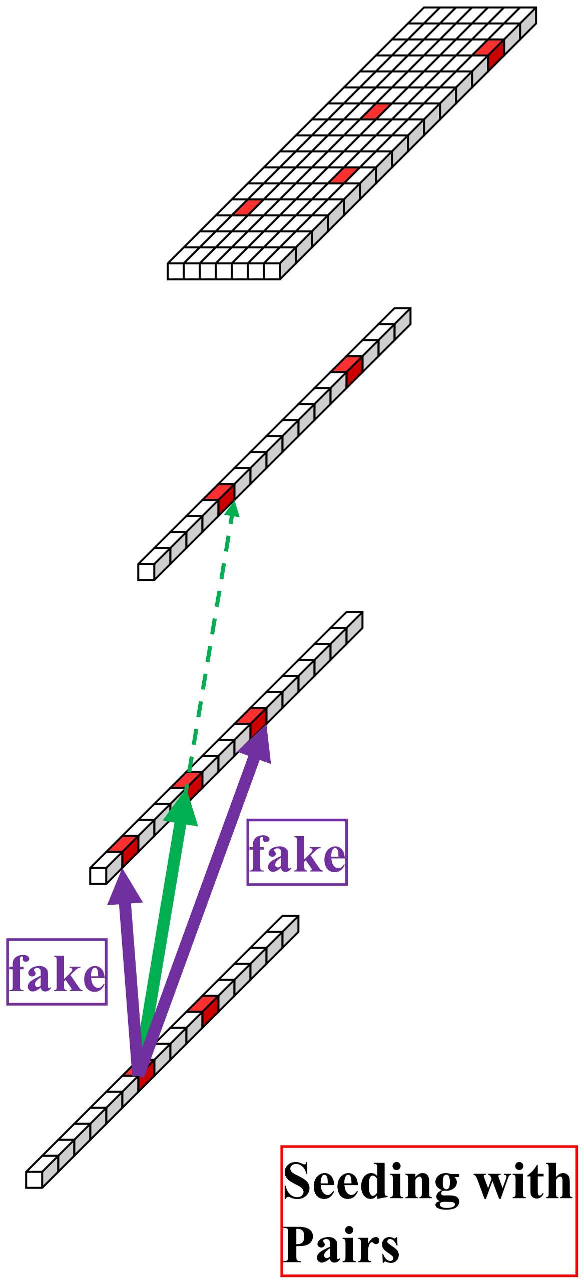
Wu et. al. 2004
https://indico.cern.ch/event/418639/contributions/1018451/attachments/868833/1216631/lowpt_lecc2004p.pdf

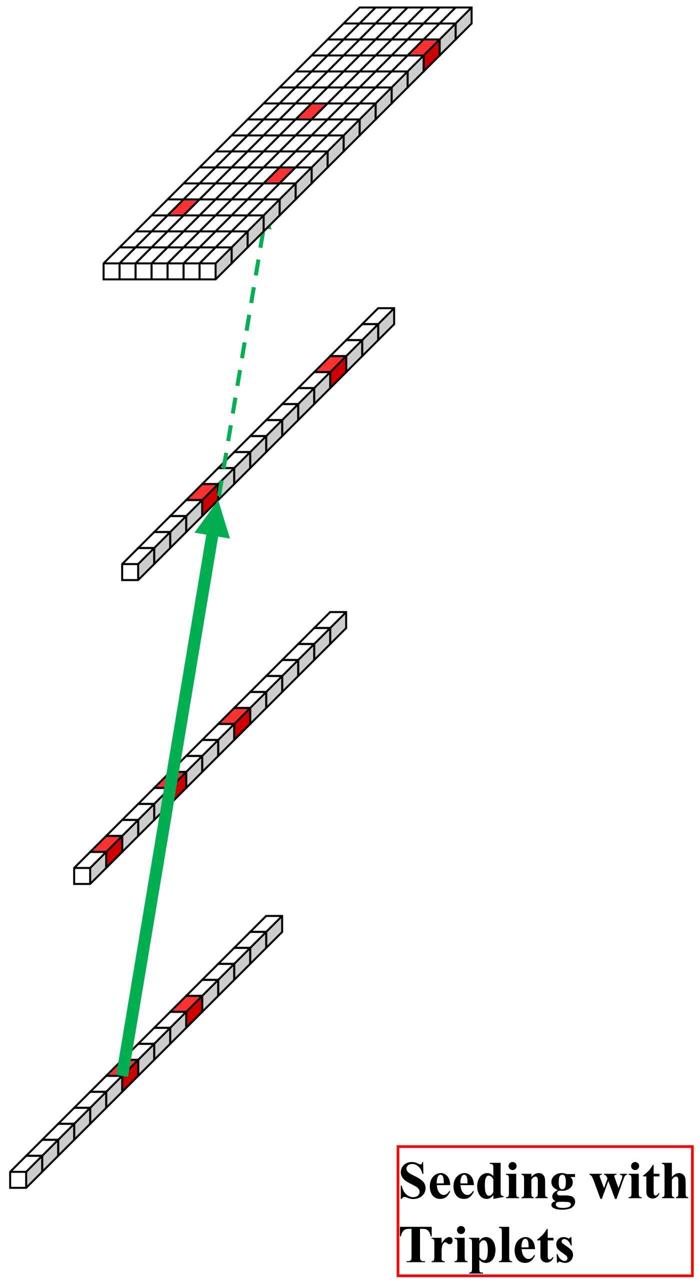
- If the separation of silicon sensor layers were large (~ 10 mm), then stub **pairing** would be sufficient.
- Given the separation of ~ 4 mm in PS and 2S module, correlating **triplets** of stubs becomes necessary.



From Pairing to Triplet Seeding

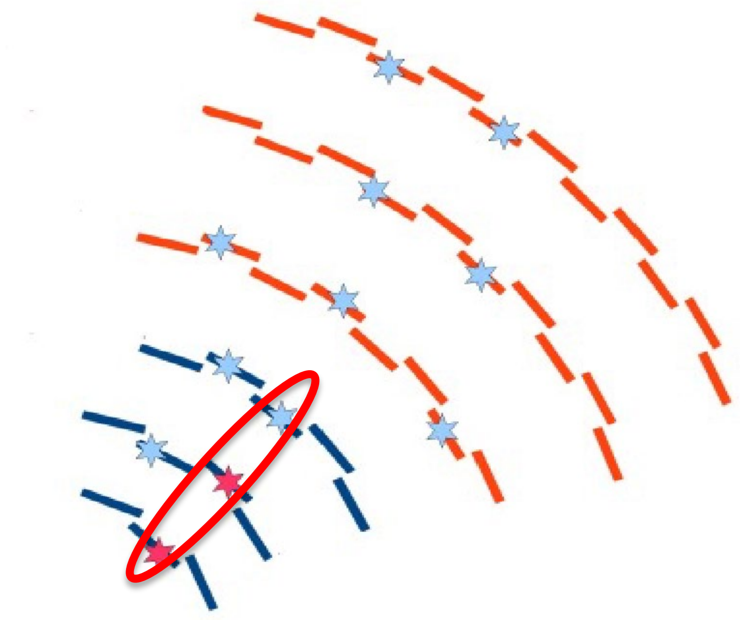
- Simple pairing will creates more fake hit pairs, which must be eliminated in later stages.
- When three or more hits (stubs) are picked up together in triplet seeding stage, less fake segments are expected.

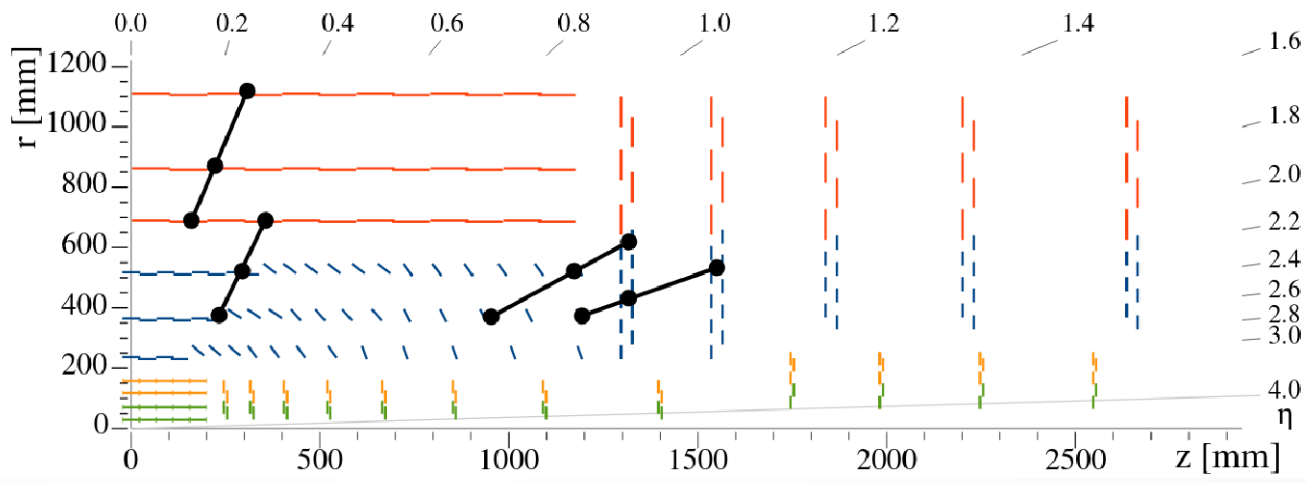




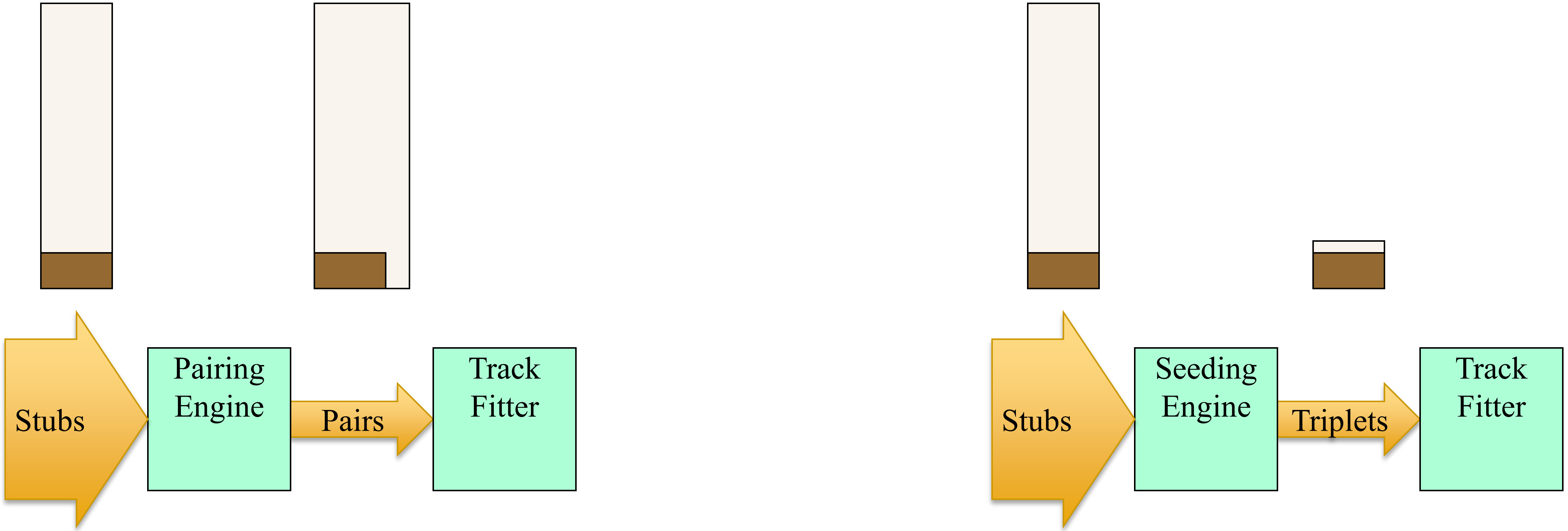
Triplet Finding

- If three hits can be correlated together in a single action, would that be useful in the Hybrid Algorithm?
- It seems that the triplet finding is more natural for:
  - Displaced track finding
  - Beam-spot constraint tracking





Triplet Seeding

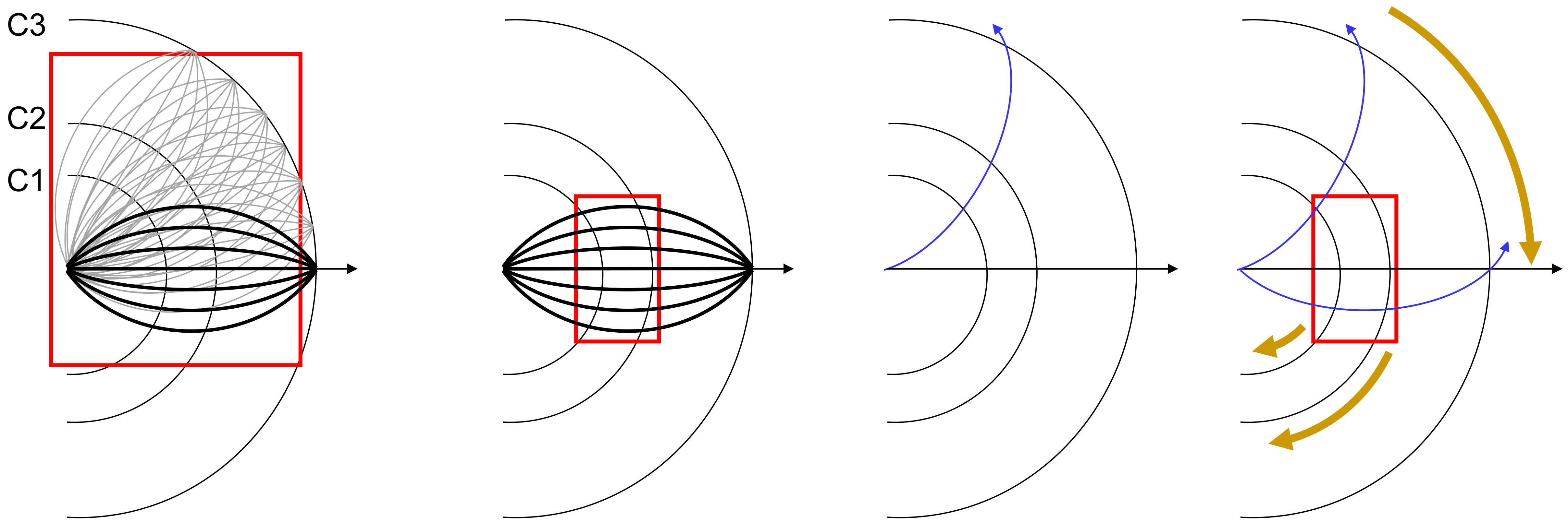


- Stub pairing won’t eliminate stubs not associated with a high Pt track.
- Using three layers or more to most eliminate fake stubs in the seeding engine.

Triplet seeding is good.
The algorithm is straightforward.
The challenge is how to fit it into FPGA.

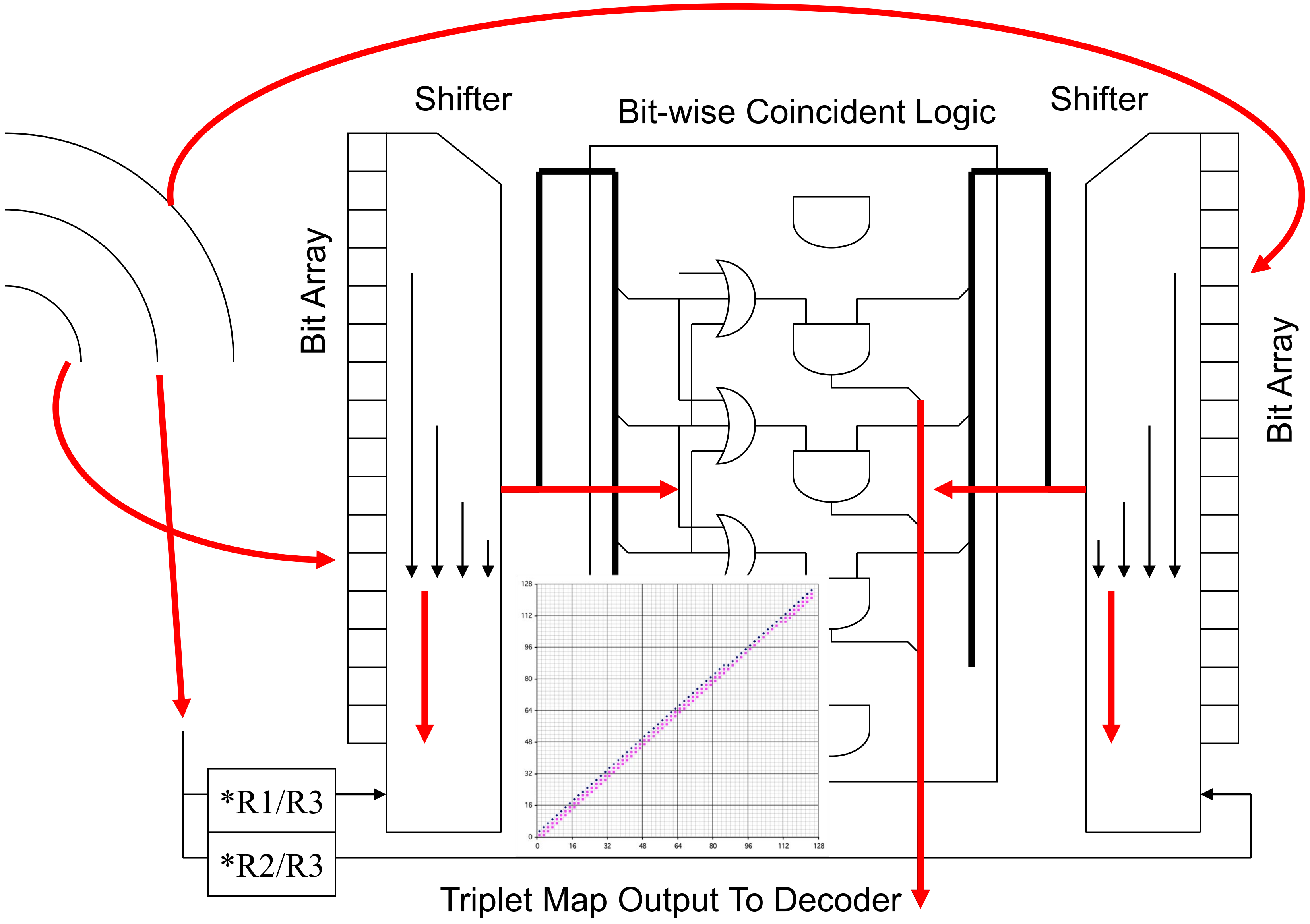
Tiny Triplet Finder:
-- A low resource usage scheme

The Tiny Triplet Finder for Cylindrical Detector Geometry



- The Tiny Triplet Finer is a scheme to reuse a single set of track road coincidence logic for whole detector.
- It uses shift (or rotate) to bring bit arrays to feed the track road coincidence logic.

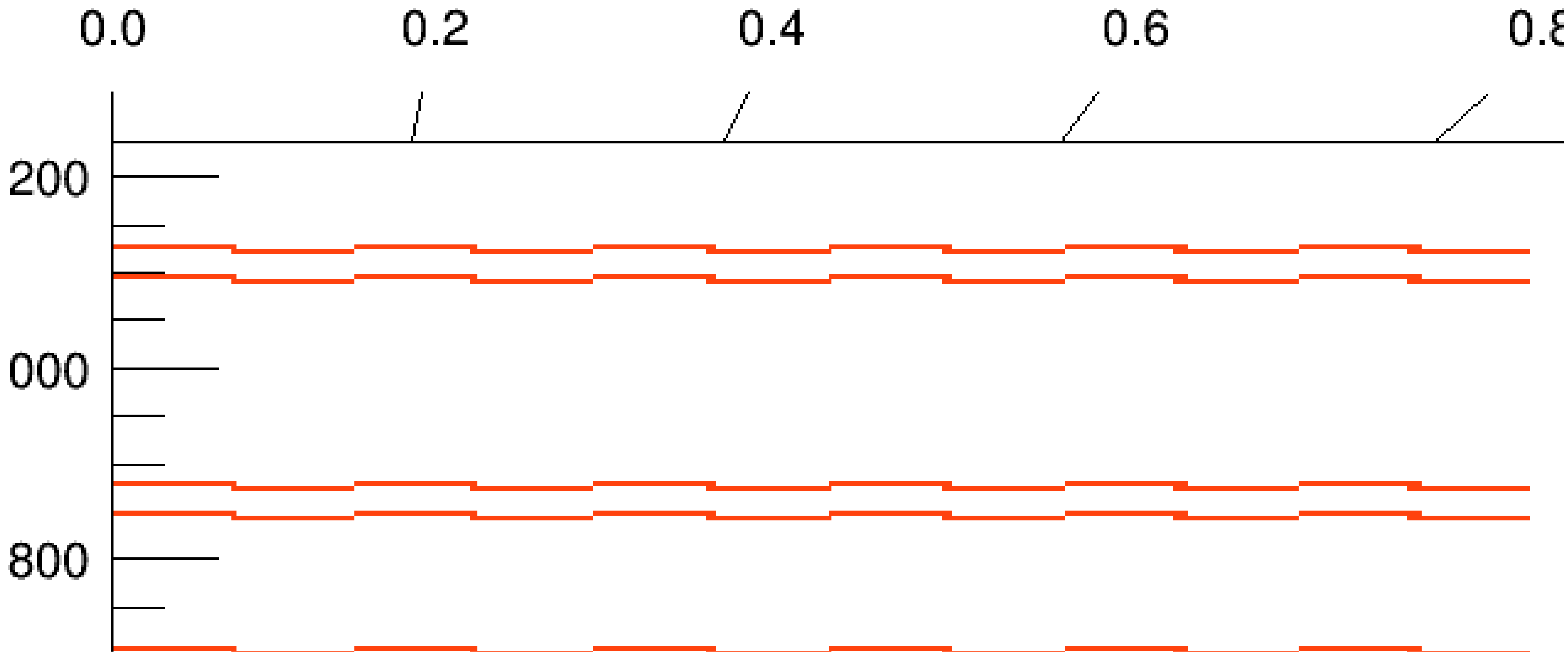
General Structure of the Tiny Triplet Finder



- A typical Tiny Triplet Finder includes:
  - Bit arrays for different layers of the detector.
  - Shifters
  - Bitwise coincidence logic.

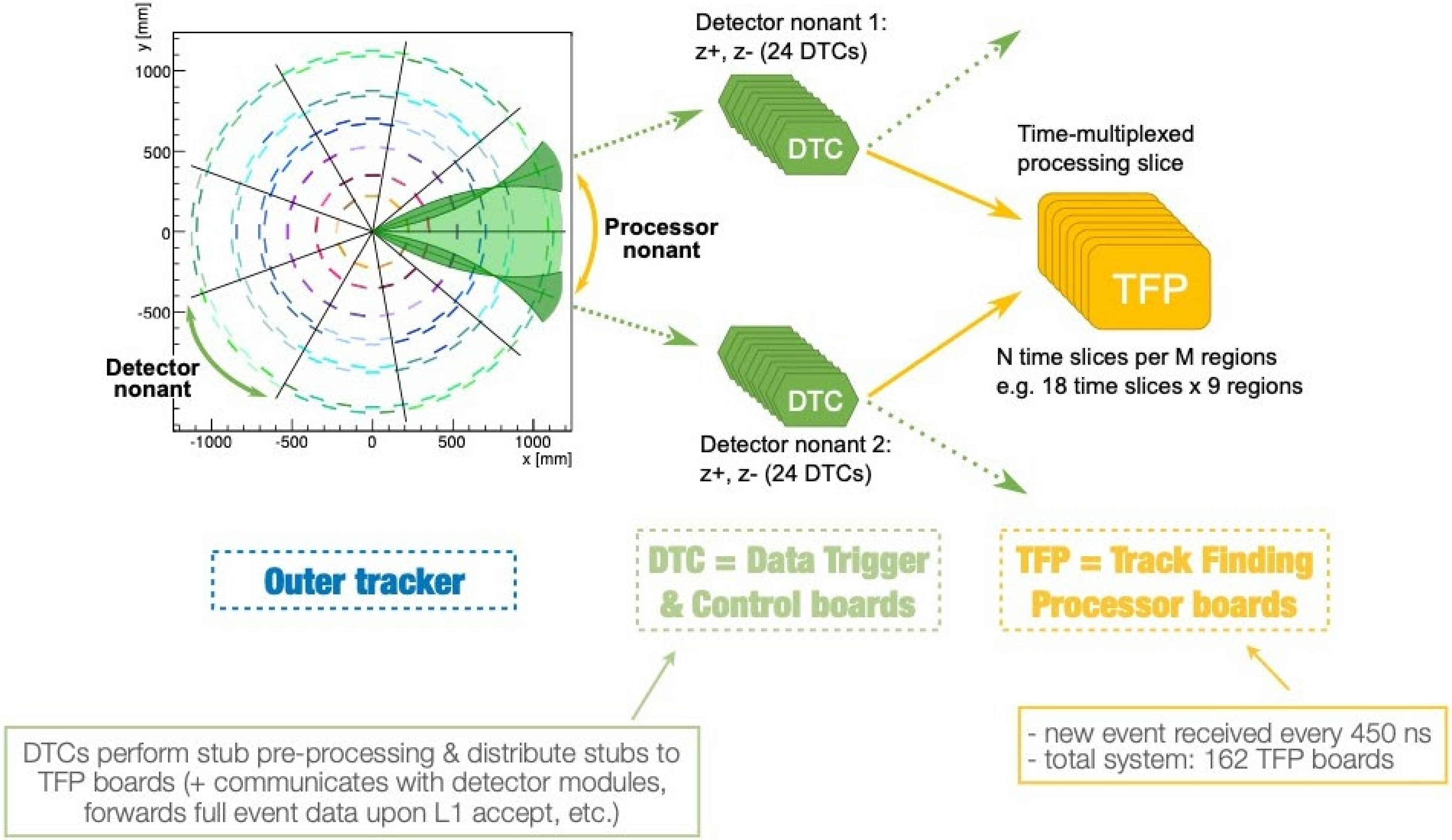
An Implementation Exercise

An Implementation Exercise Using CMS L1 Tracking as Example



- The detector in the exercise have the same radii and lengths as CMS Outer Tracker

System Division

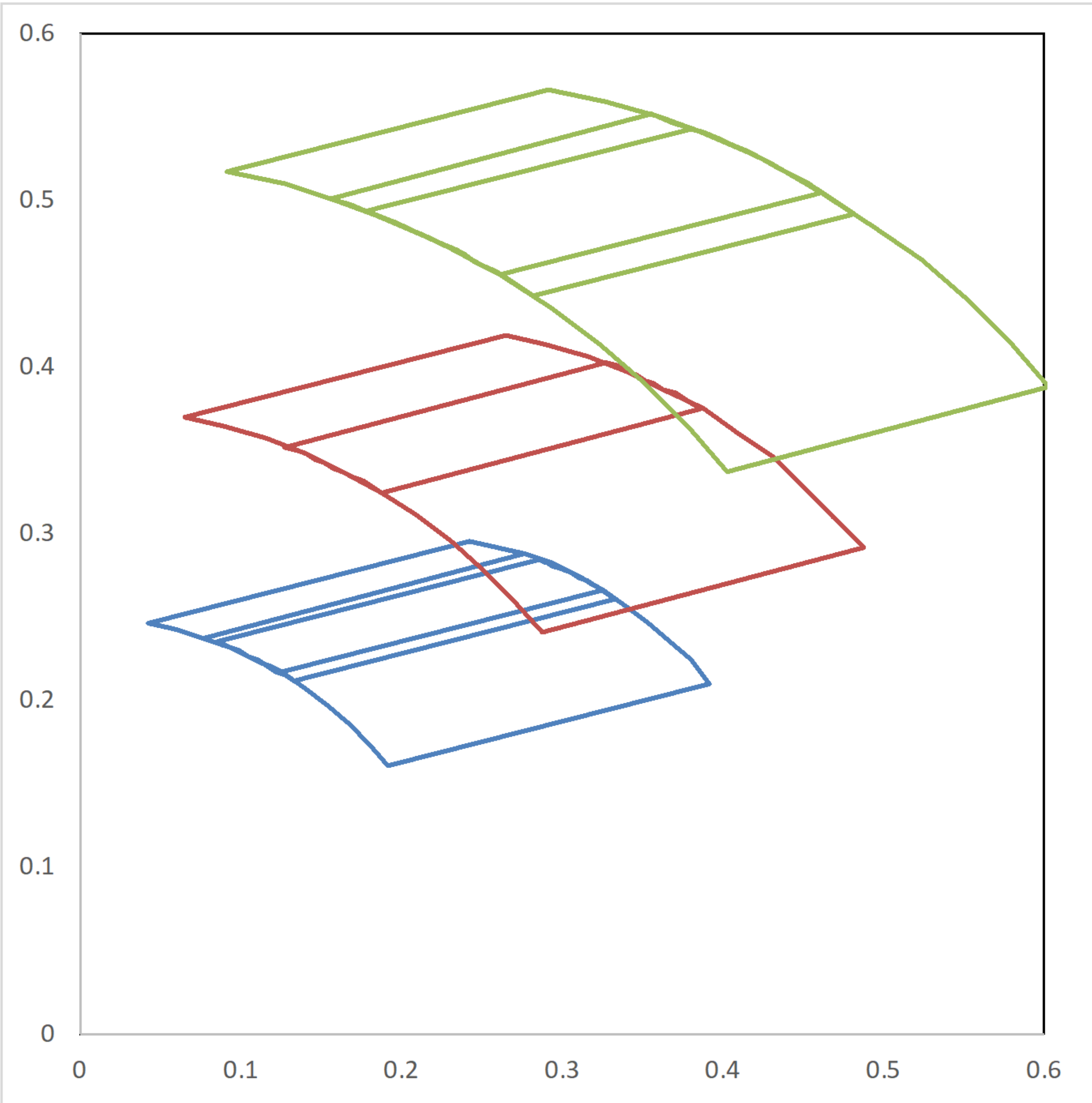


Louise Skinnari et. al. 2020

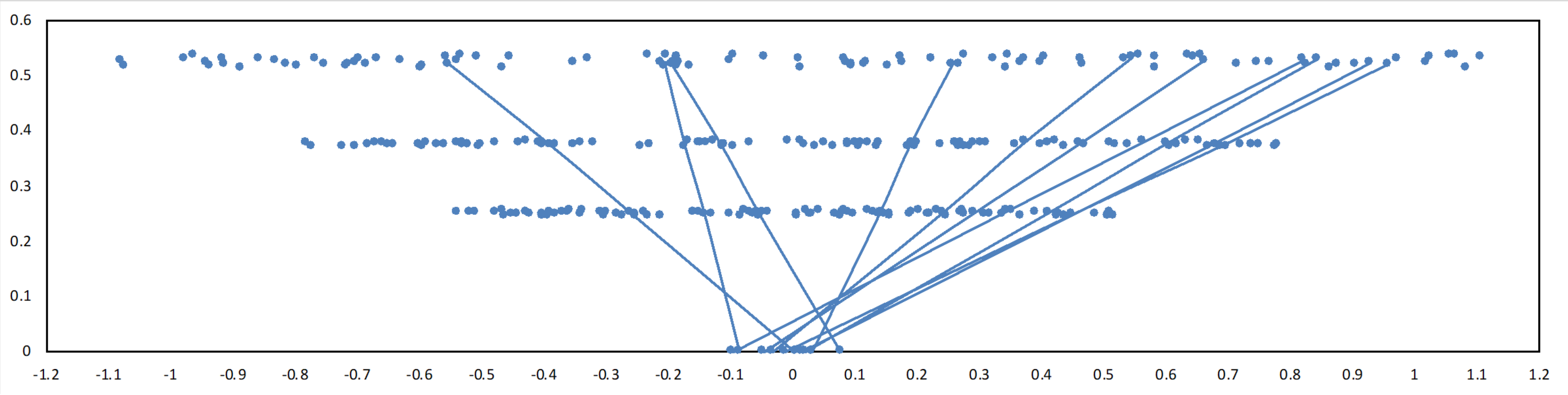
- System Division:
  - 18 time slices
  - 9 regions
- Each TFP serves:
  - 40 degrees.
  - 25 ns x 18 = 450 ns or 225 clock cycles @500 MHz

Serving Range of Seeding Engine

- In each TFP, 4 seeding engines are implemented, each serving 2.4 m (full length) x 10 degrees plus overlap.
- Each engine sees <100 hits/layer per event.
- The layer 2 is divided with exact 10-degree boundaries.
- Overlapping in:
  - Layer 1: +- 2.16 degrees, 14.3 degrees total
  - Layer 3: +- 2.62 degrees, 15.2 degrees total

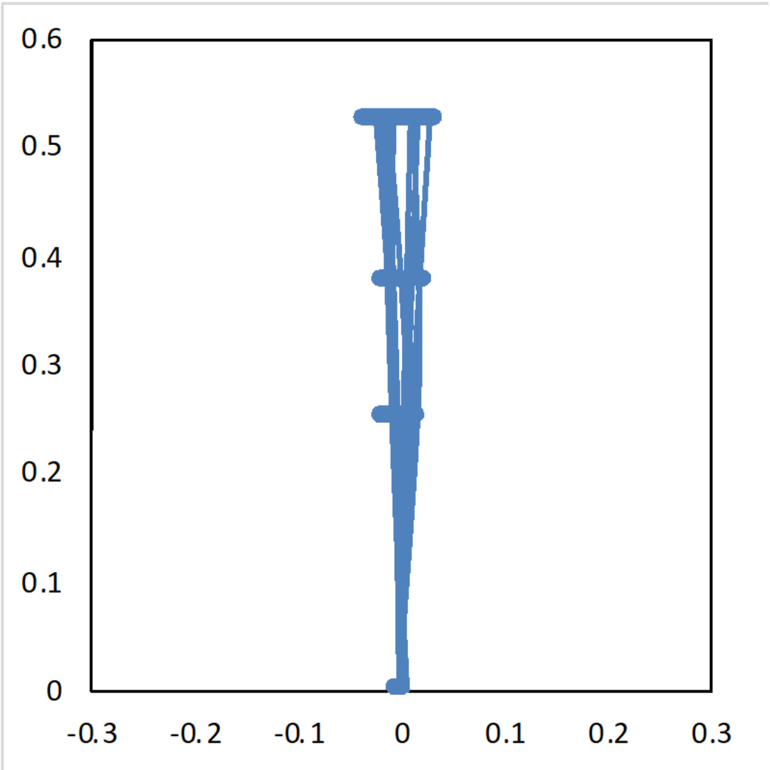


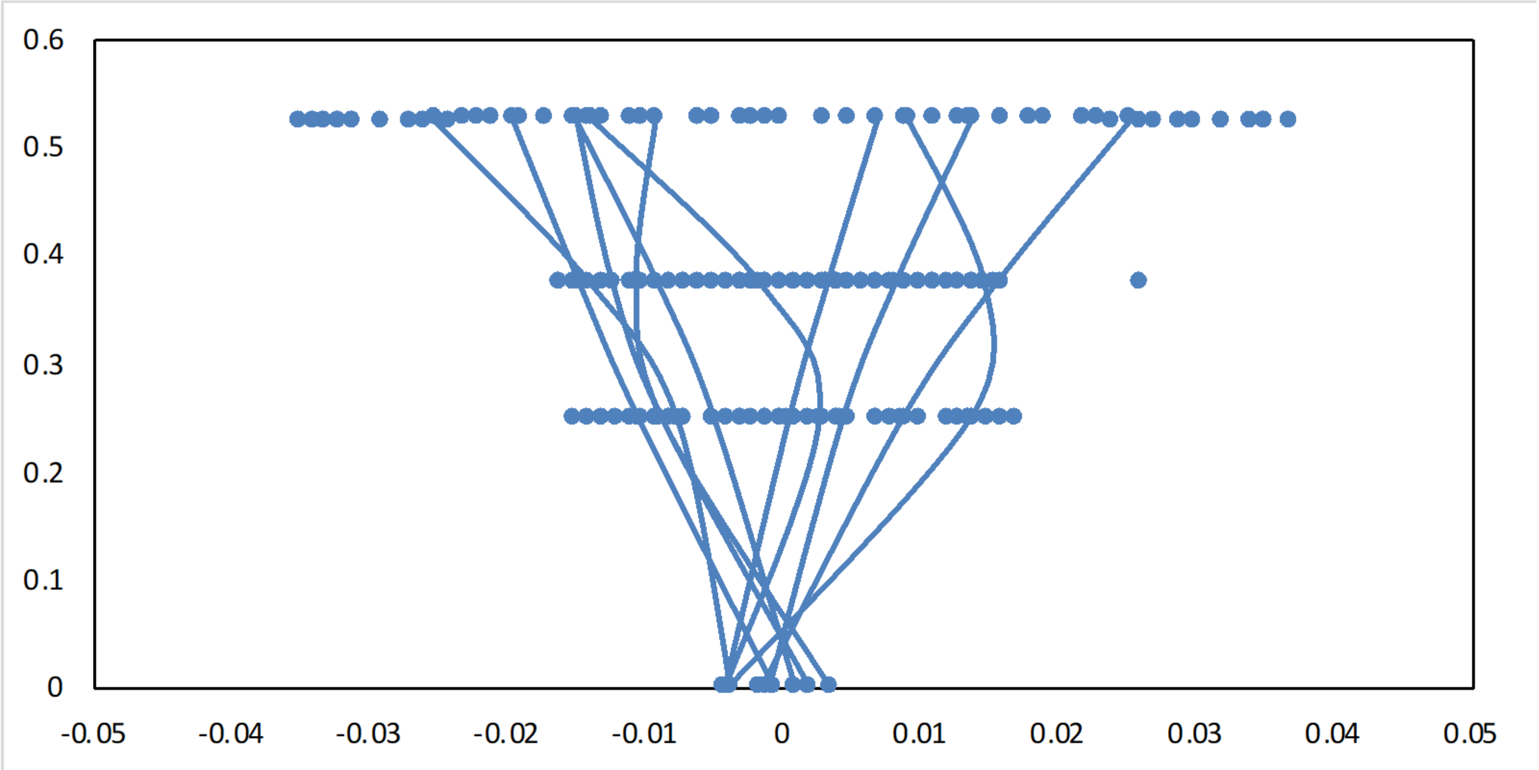
Generated Events: r-z View



- In 240 cm (z) x 10 degrees (phi) range, up to 112 hits per layer are generated, out of which, 10 are from good tracks.
- The z0 of the vertices spread in +- 10 cm along beam axis.

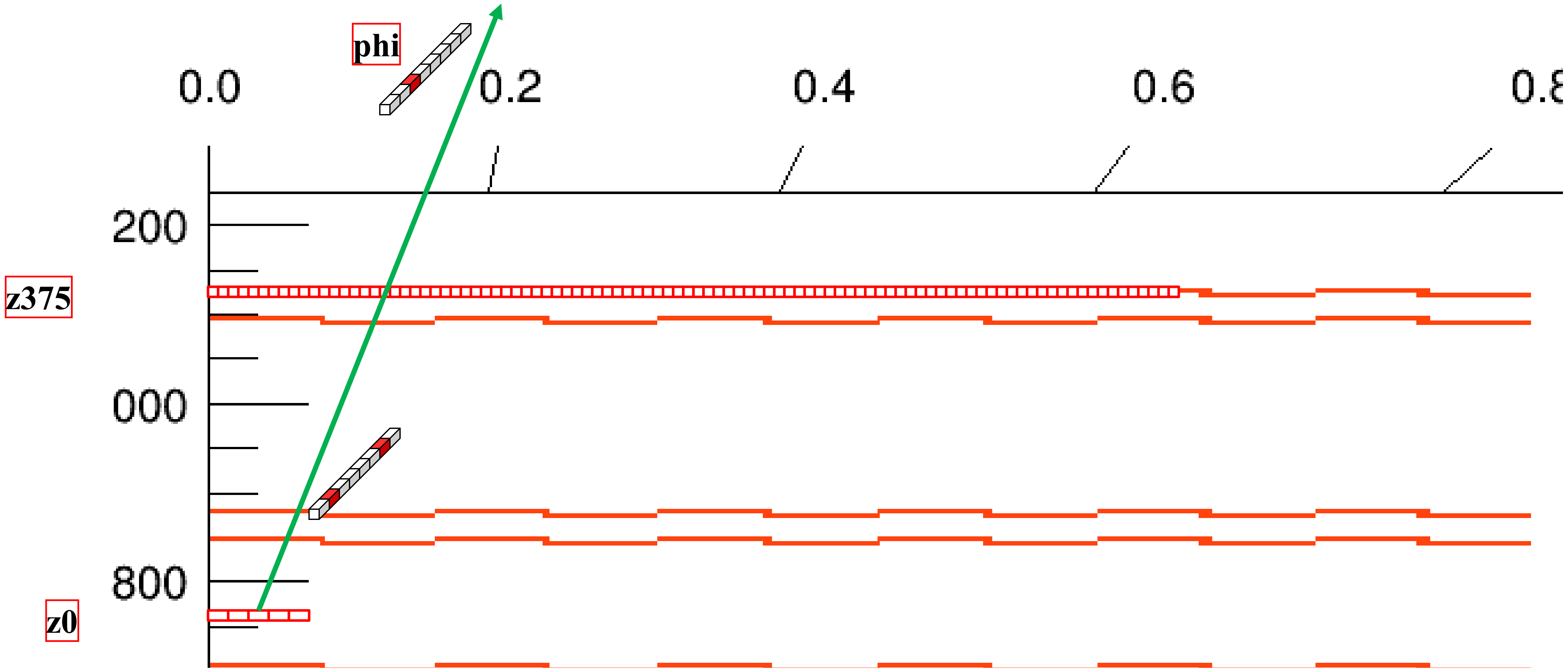
The Events: r-phi View





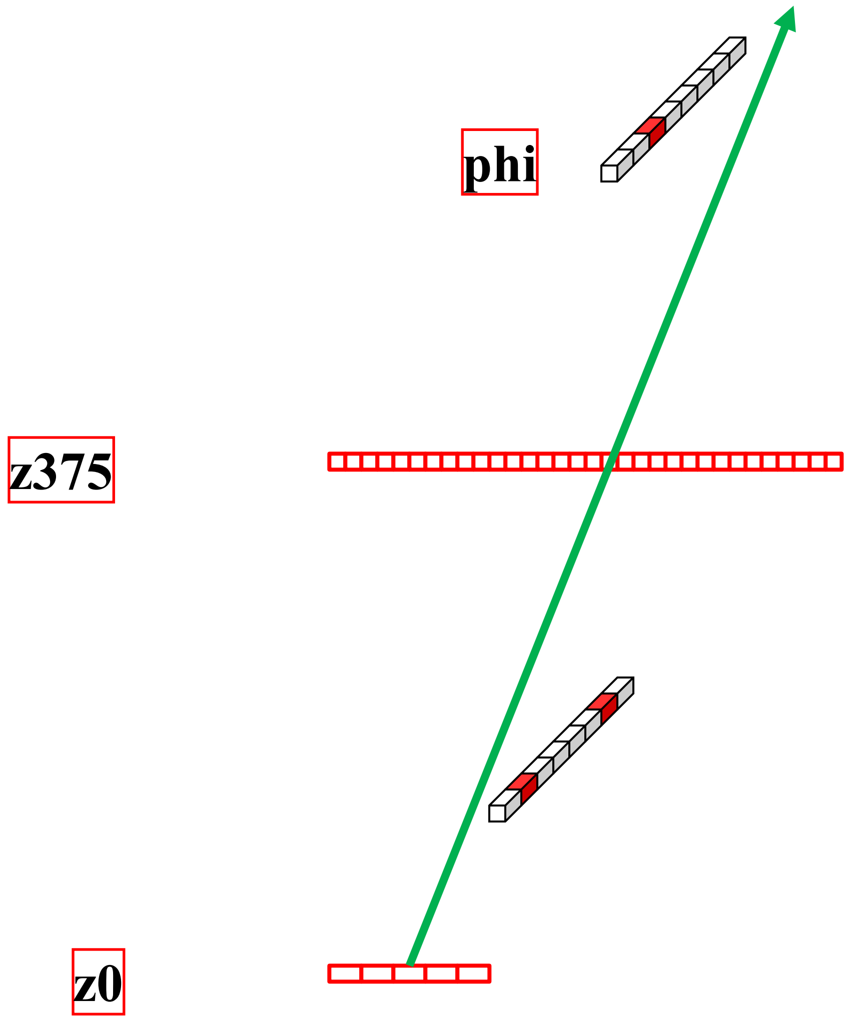
- Minimum PT: 2GeV/c.
- The hits on Layer 2 is limited to 10 degrees, but hits on Layer 1 and 3 spread up to 16 degrees.
- The impact parameters of the tracks: +- 2 mm.

The 3D Seeding Engine for SP Layers



- The parameters chosen in r-z plane are z0 and z375 (z375 close to z of Layer 2).
- For any stub in Layer 2, the phi pattern in Layer 1 and Layer 2 with matching (z0, z375) are checked.
- The triplets matching not only in r-z, but also in r-phi are reported for further track fitting.

Bin Numbers

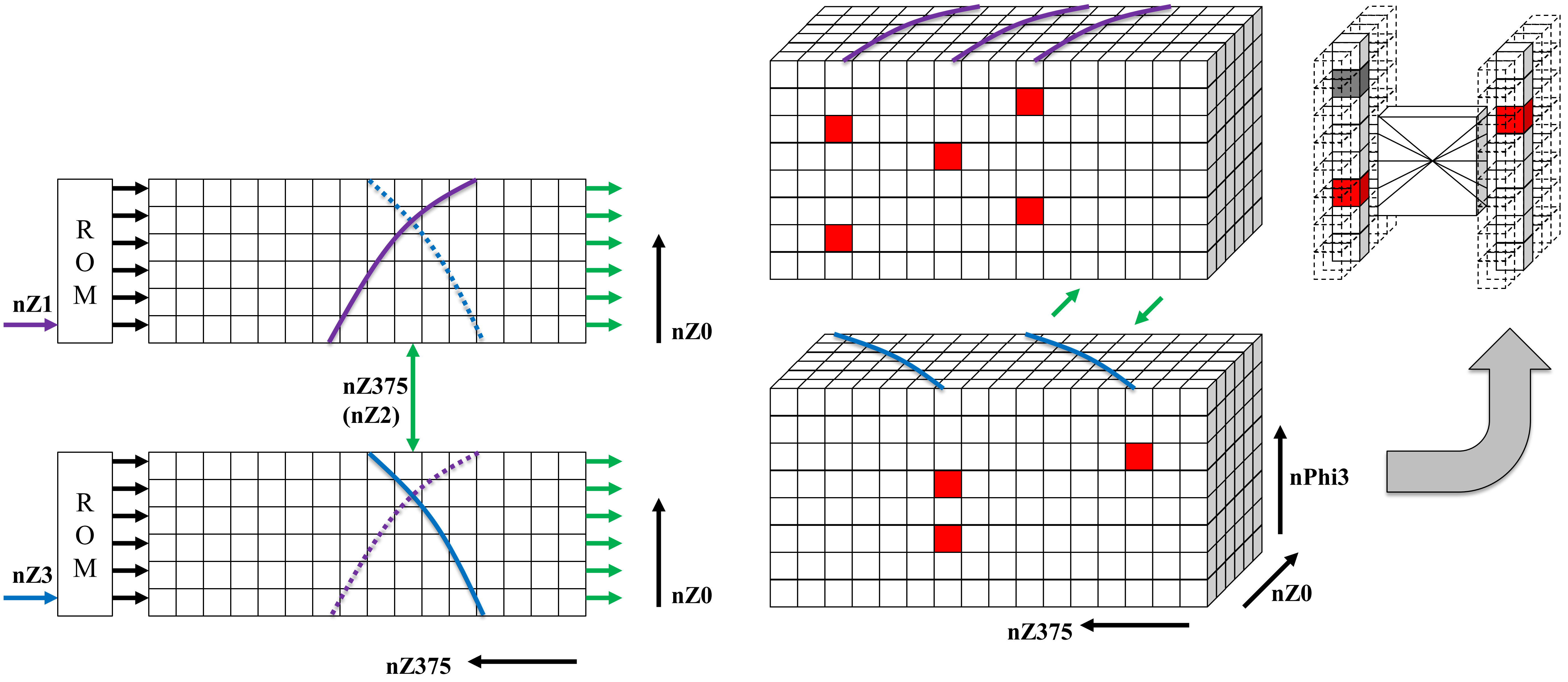


	Width	Number
z0	2 cm	10
z375	1 cm	240
phi	0.125 degrees (0.55 mm at 25 cm)	128 (/16 degrees)

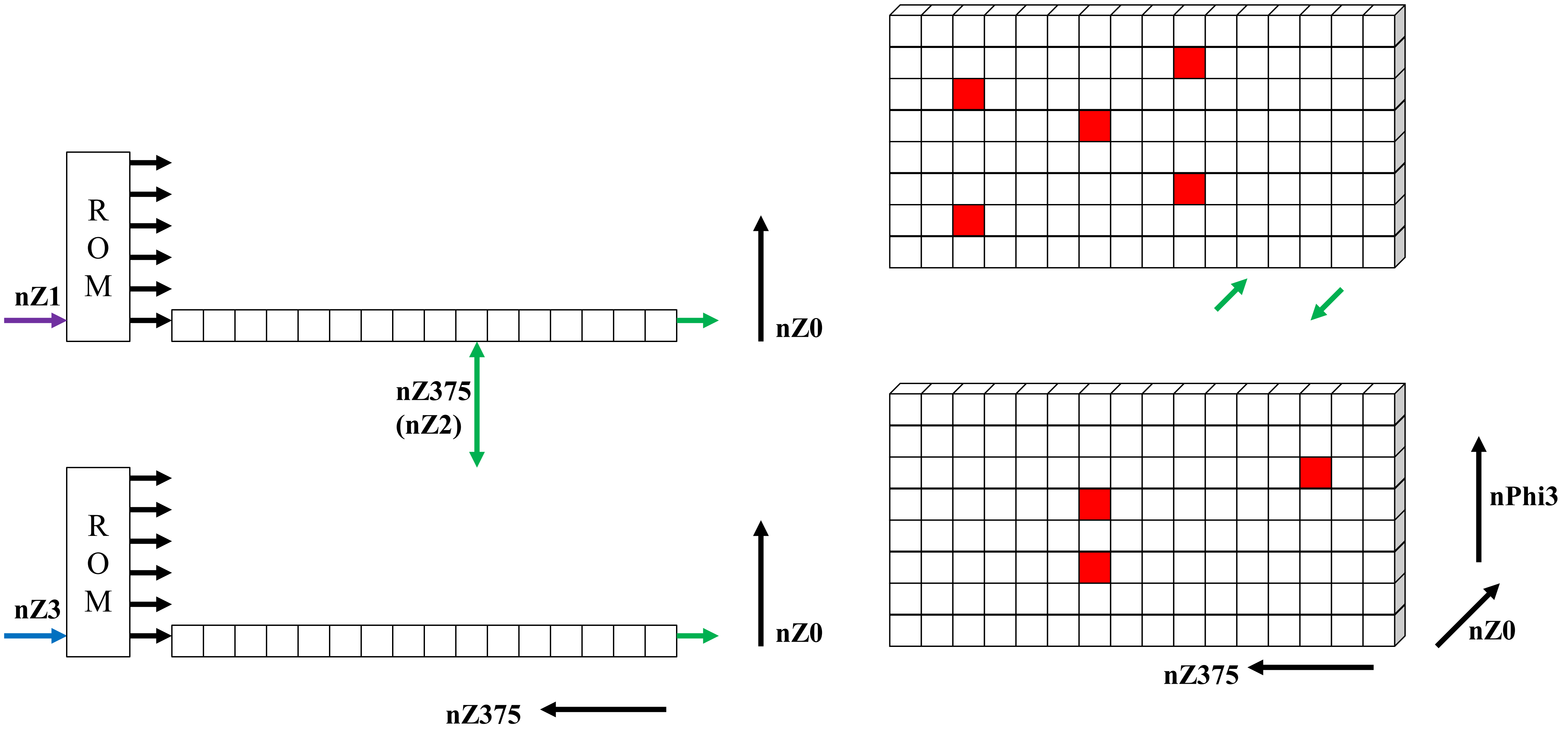
- Neither phi bins nor z bins can be too fine.
- Just checking r-phi or r-z would have too high fake rate.
- The 3-D track seeding (i.e., checking both r-phi and r-z) should provide sufficient fake stub rejection.

The Track Seeding Engine

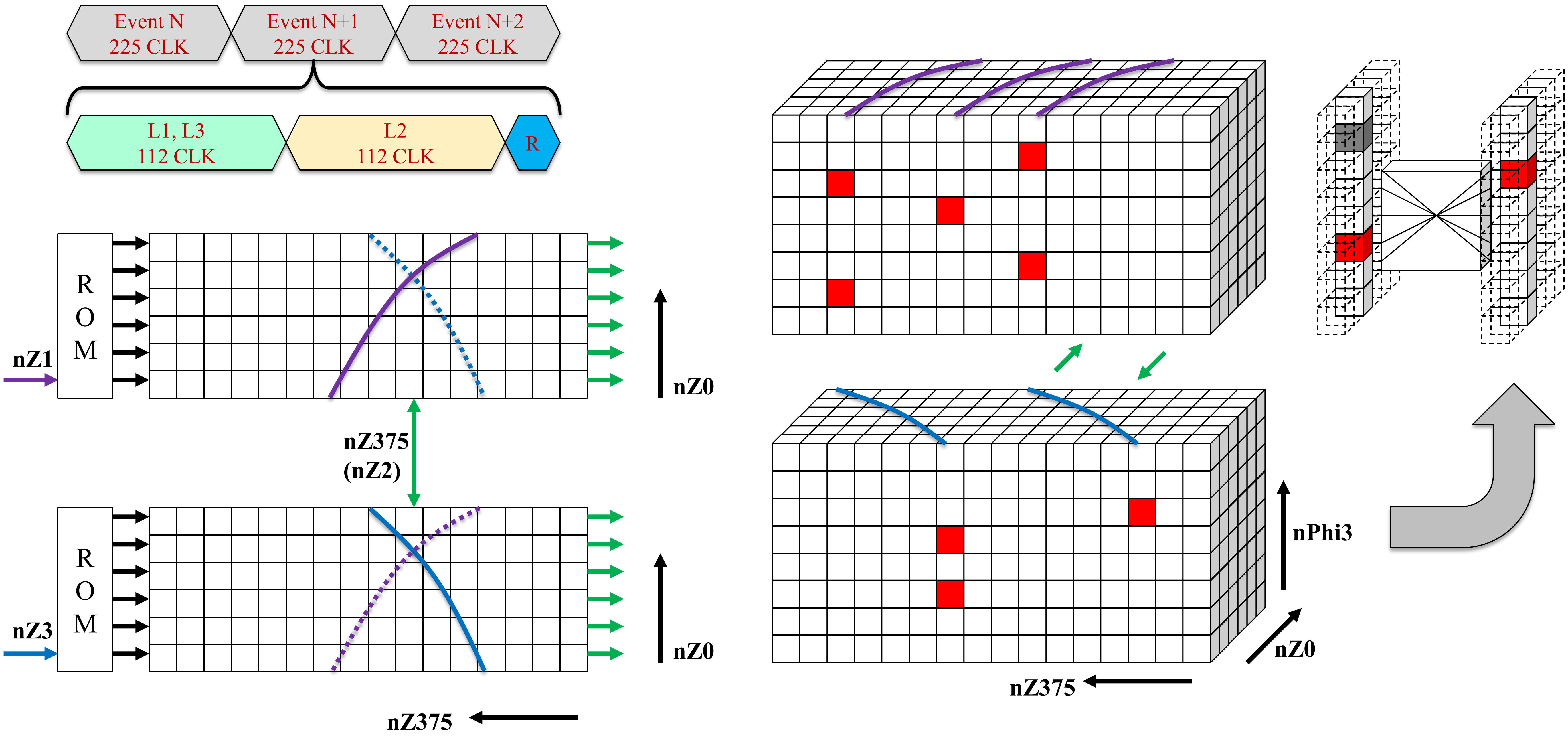
Structure of the Seeding Engine: Two RAM Banks



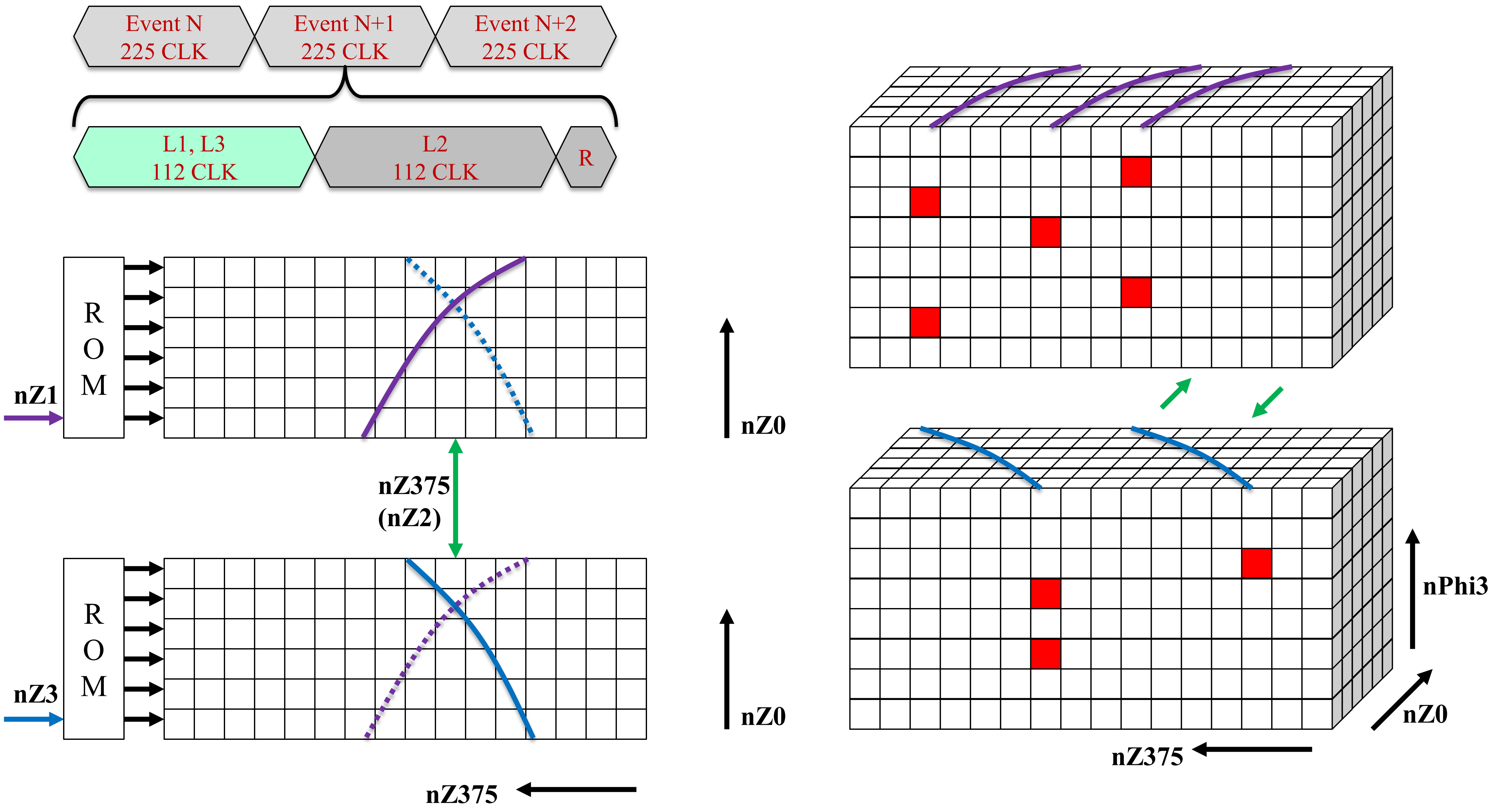
Structure of the Seeding Engine: Single Physical RAM View



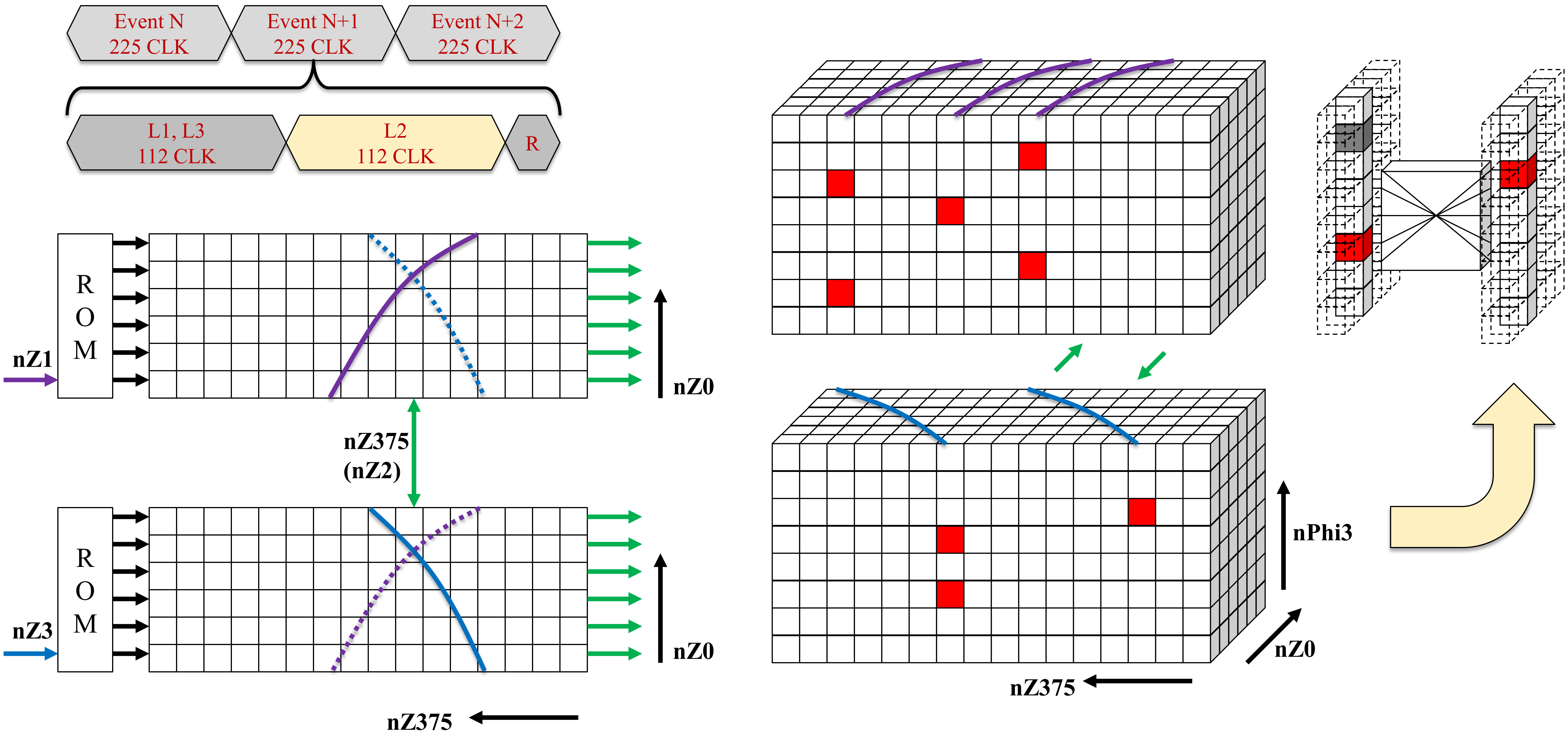
General Timing



Operation: (1) Filling Hits

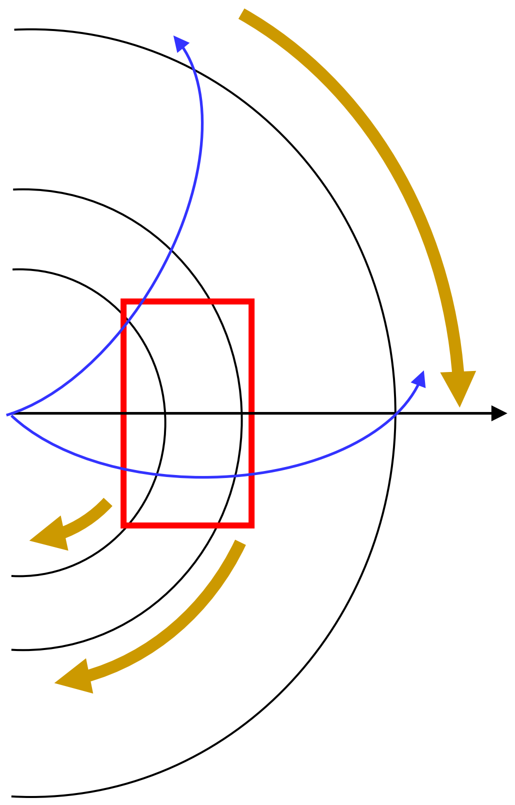


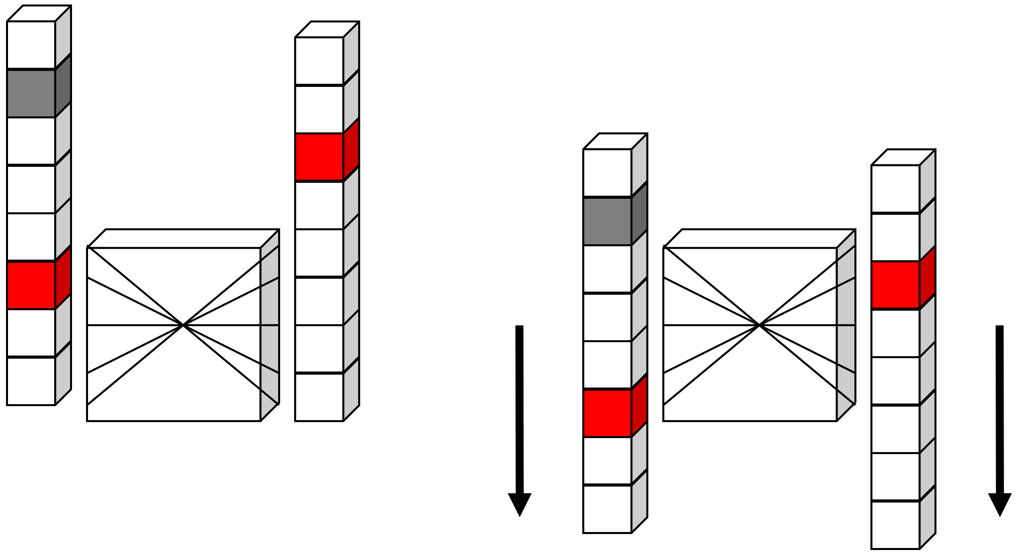
Operation: (2) Checking Coincidence



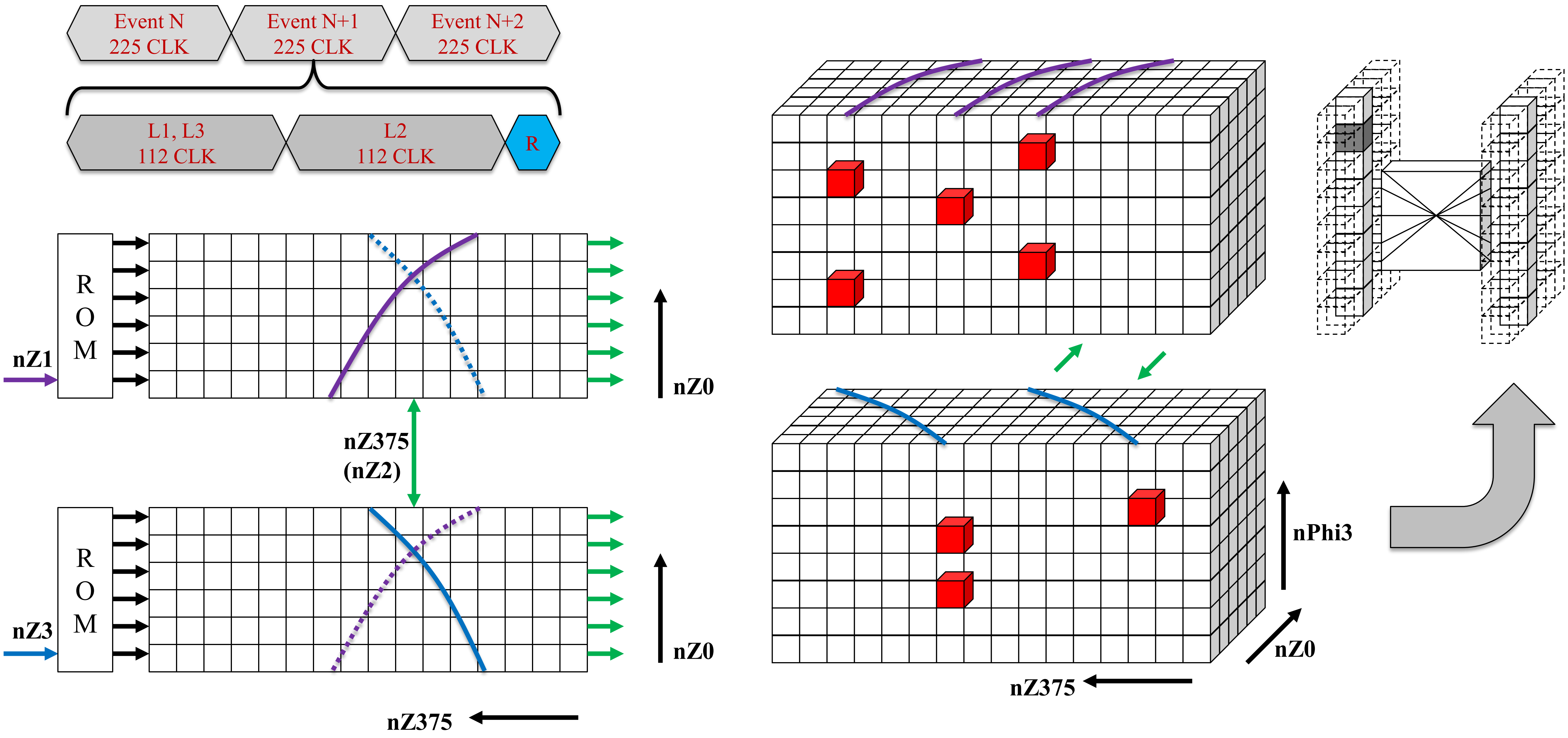
Pattern Shifting and checking

- The bit patterns with the same z0 from Layer 1 and Layer 3 are shifted to align with the coincidence road array.
- The amount of shift is determined by phi2.
- Matching bits are output for further process.

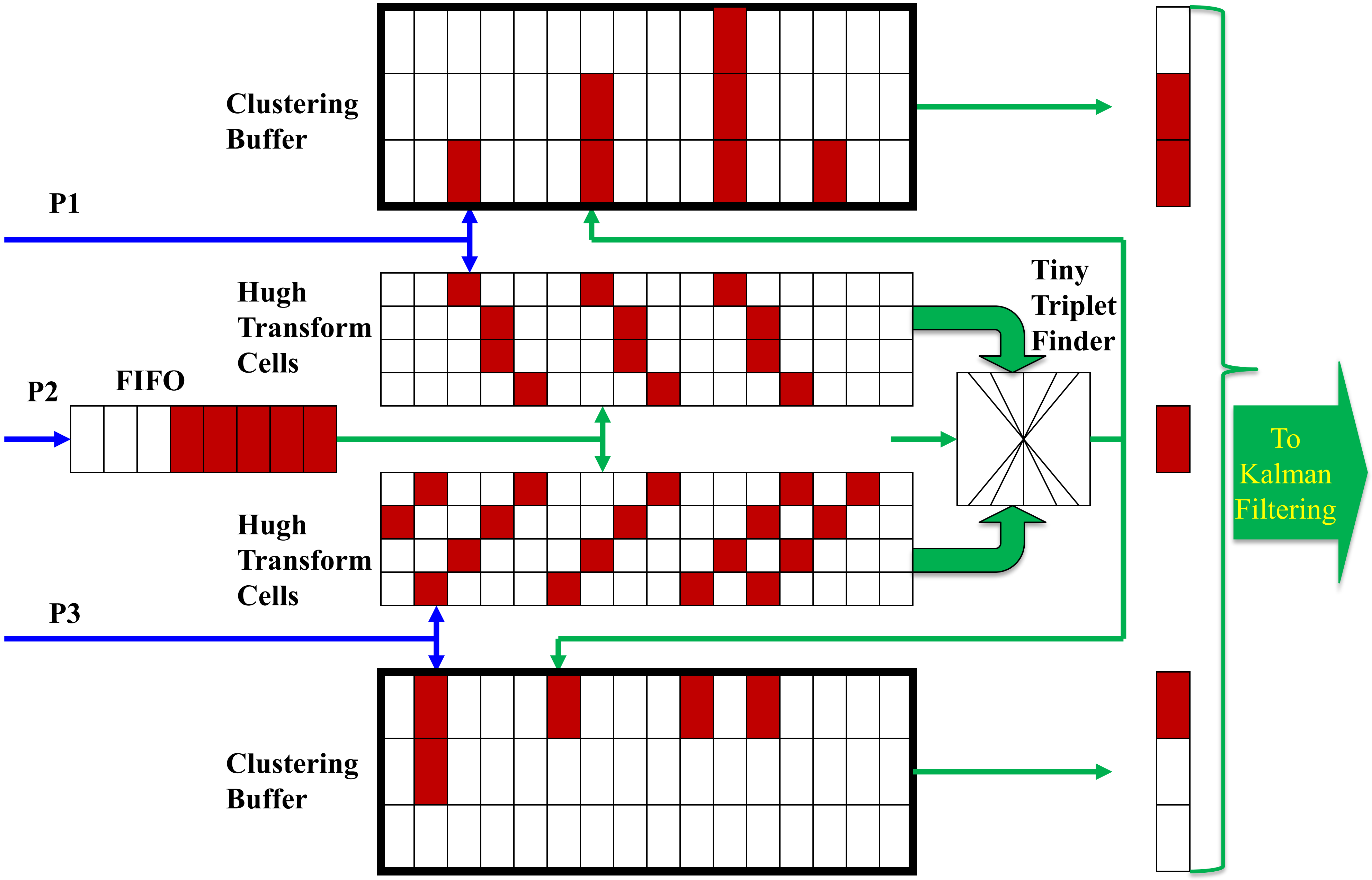




Operation: (3) Refresh Cycle



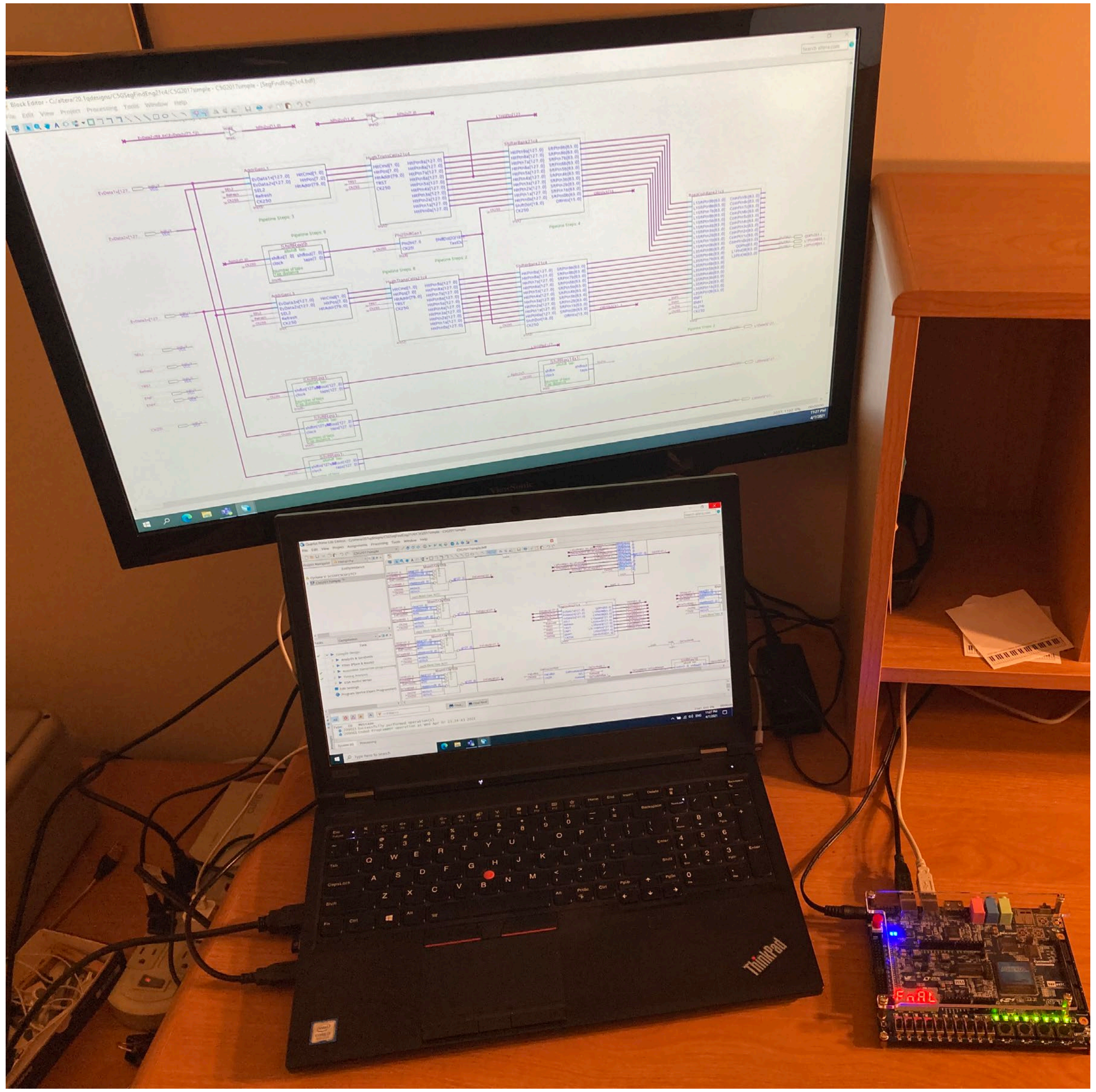
The Seeding Engine & Clustering Buffer

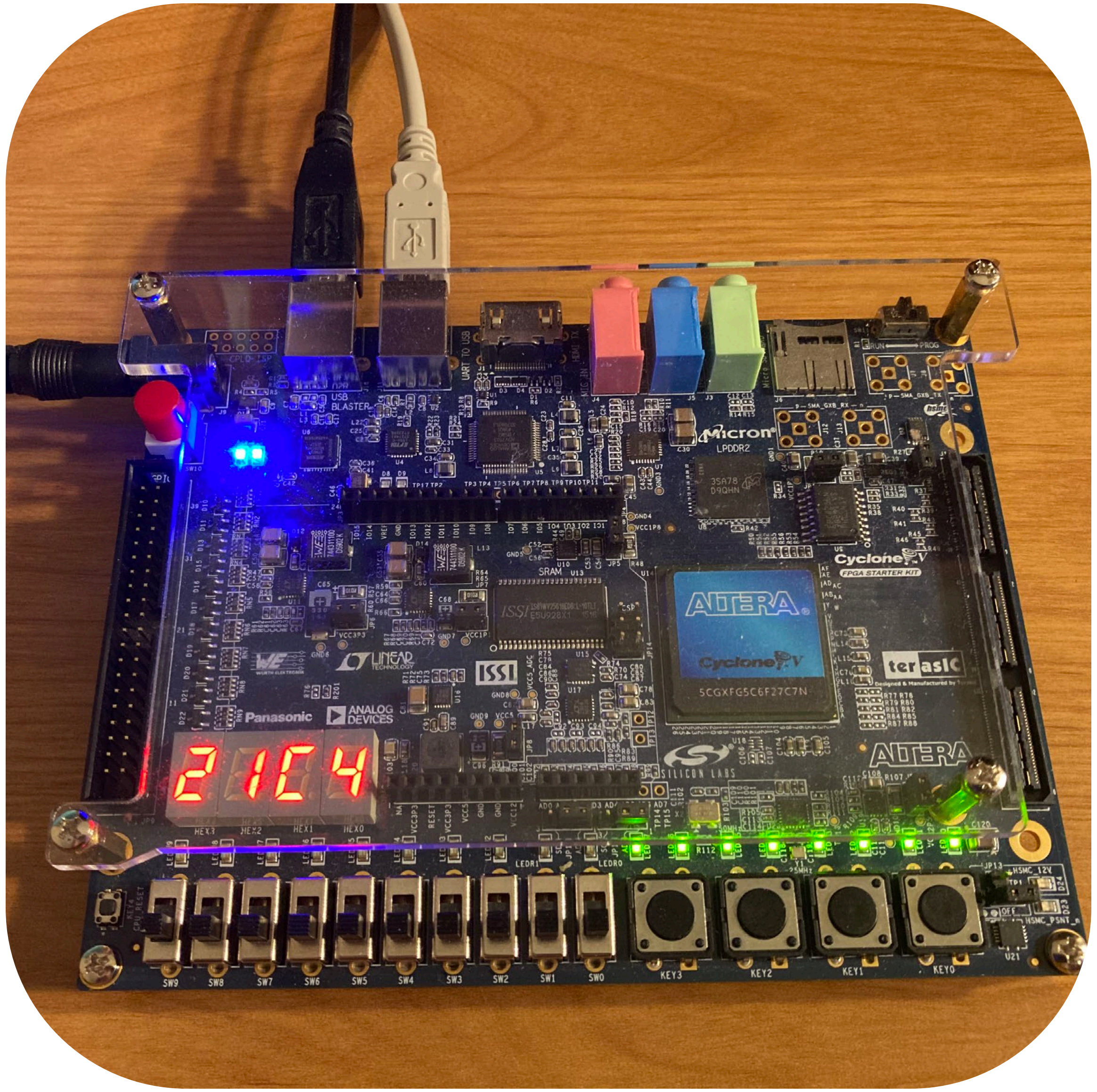


- Full data of hits are stored in the clustering buffer.
- After triplet finding, the full data are sent to the Kalman filter stage for further track fitting.

Implementation in Cyclone 5 FPGA

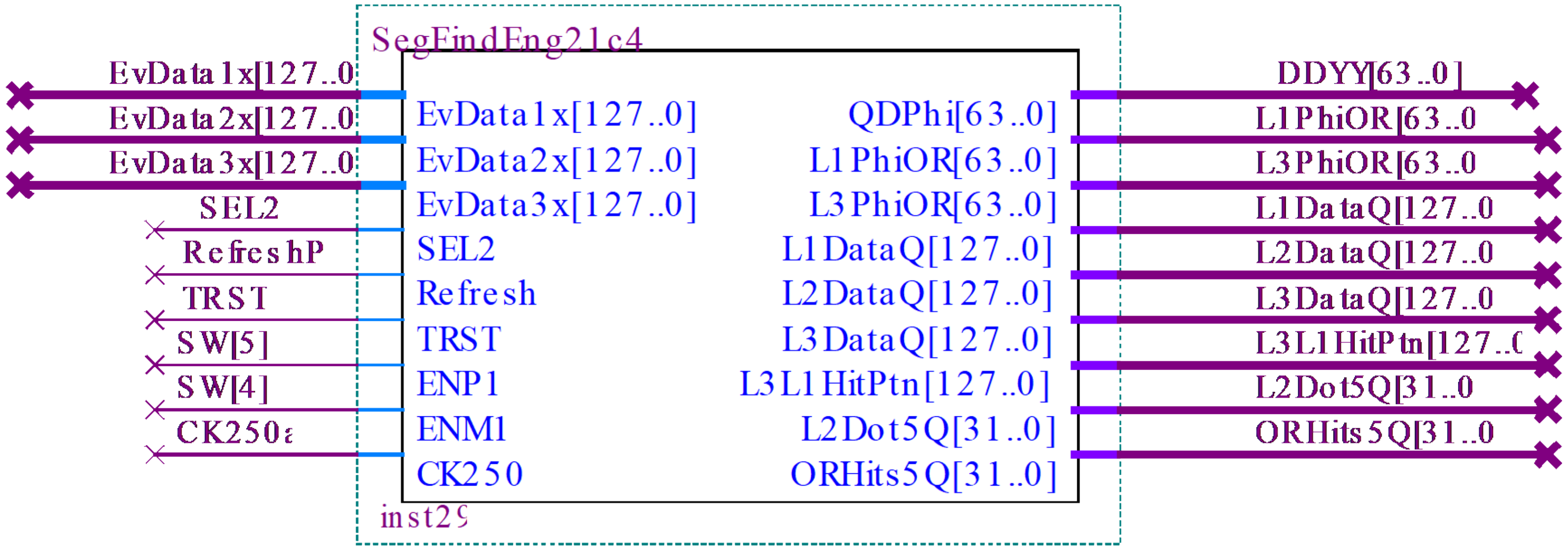
Home Test Stand and the Evaluation Module





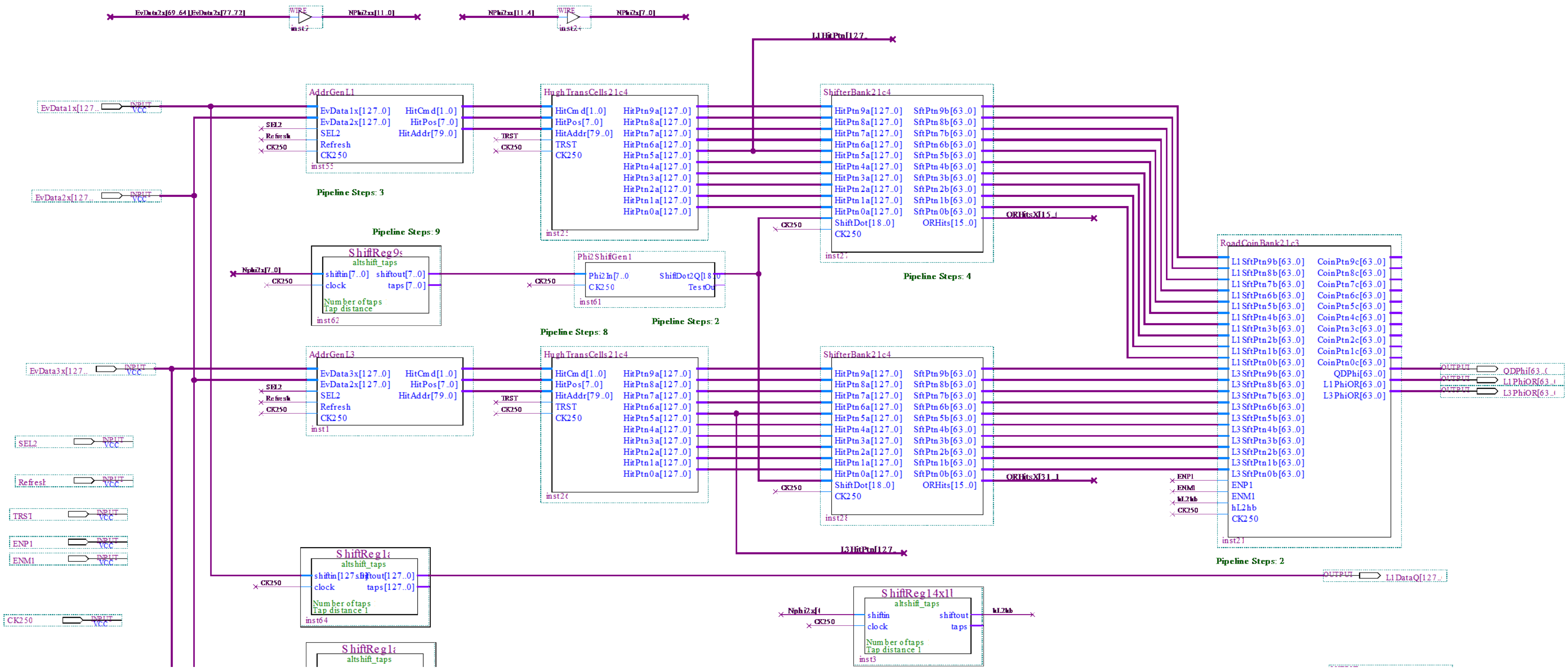
- The seeding engine core firmware is implemented in Altera C5G evaluation module (179 USD ea.).
- The FPGA is Cyclone V GX 5CGXFC5C6F27C7N (171 USD ea.).

The Seeding Engine Block I/O



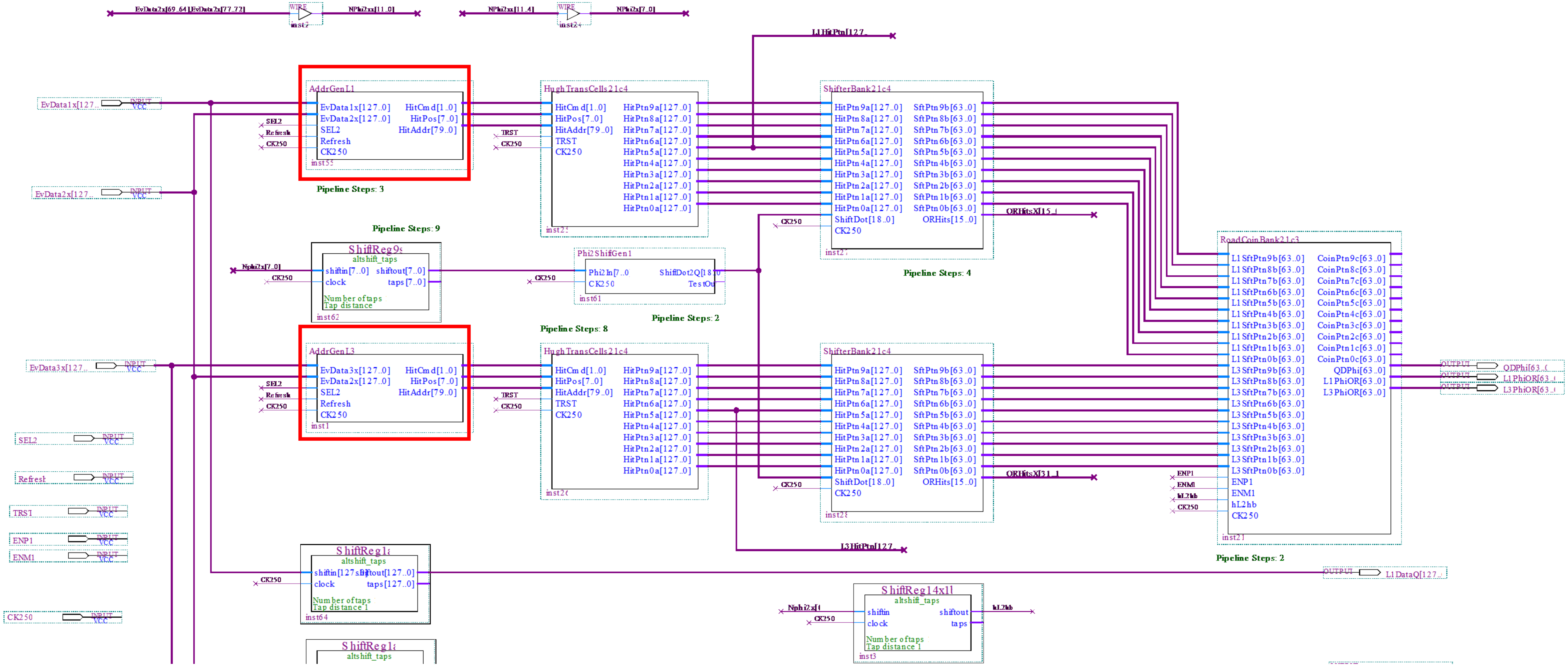
- The hit data from Layer 1, 2 and 3 are fed into the engine, one hit per clock cycle.
- The primary coincidence outputs are in port DDYY[63..0] which represents up to 64 possible 1/PT values of the track segment.

Structure of the seeding engine

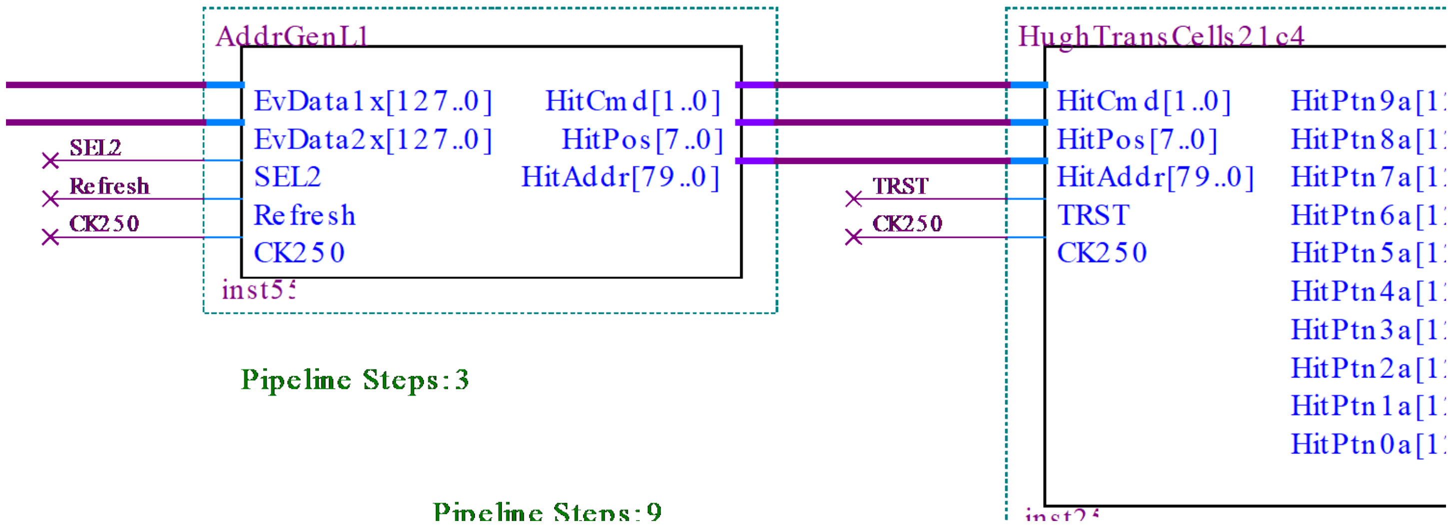


- The entire engine is organized as a pipeline so that it can run at high speed (250 MHz in Cyclone V).
- Blocks: Address Generation; Hugh Trans. Cells; Shifter Bank; Road Coincidence

The Seeding Engine



Address Conversion for Hugh Transform Cells



- For Layer 1 or Layer 3 Hugh Transform Cells have 10 RAM blocks corresponding 10 bins in z0.
- The z-coordinate z1 or z3 is converted to z375 for each z0 by checking the lookup table ROM.
- The ROM contents depend on detector geometry and are generated using simulation data.

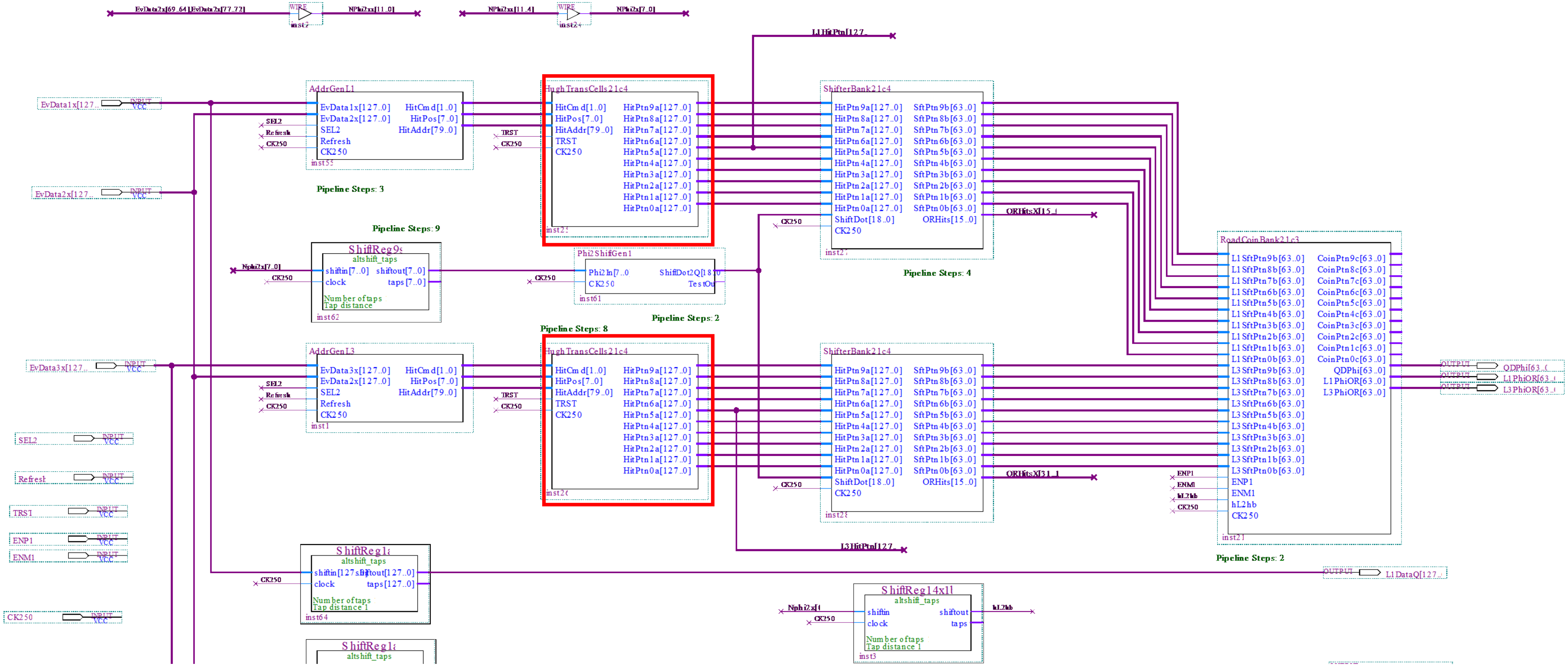
MemInit1.txt

```
--Format: Quartus .mif

WIDTH=80;
DEPTH=256;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
00 : 00000000000000000000;
01 : 00000000000000000000;
02 : 00000000000000000000;
03 : 00000000000000000000;
04 : 00000000000000000000;
05 : 00000000000000000000;
06 : 00000000000000000000;
07 : 00000000000000000000;
08 : 00000000000000000000;
09 : 00000000000000000000;
0a : 00000000000000000000;
0b : 00000000000000000000;
0c : 00000000000000000000;
0d : 00000000000000000000;
0e : 00000000000000000000;
0f : 00000000000000000000;
10 : 00000000000000000000;
11 : 00000000000000000000;
12 : 00000000000000000000;
13 : 00000000000000000000;
14 : 00000000000000000000;
15 : 00000000000000000000;
16 : 00000000000000000000;
17 : 00000000000000000000;
18 : 00000000000000000000;
19 : 00000000000000000000;
1a : 00000000000000000000;
1b : 00000000000000000000;
1c : 00000000000000000000;
1d : 00000000000000000000;
1e : 00000000000000000000;
1f : 00000000000000000000;
20 : 00000000000000000000;
21 : 00000000000000000000;
22 : 00000000000000000000;
23 : 00000000000000000000;
24 : 00000000000000000000;
25 : 00000000000000000000;
26 : 00000000000000000000;
27 : 00000000000000000000;
28 : 01000000000000000000;
29 : 02010000000000000000;
2a : 04030201000000000000;
2b : 05040302010000000000;
2c : 07060504030201000000;
2d : 08070605040302010000;
2e : 0a090807060504030201;
2f : 0b0a0908070605040302;
30 : 0d0c0b0a090807060504;
31 : 0e0d0c0b0a0908070605;
32 : 100f0e0d0c0b0a090807;
33 : 11100f0e0d0c0b0a0908;
```

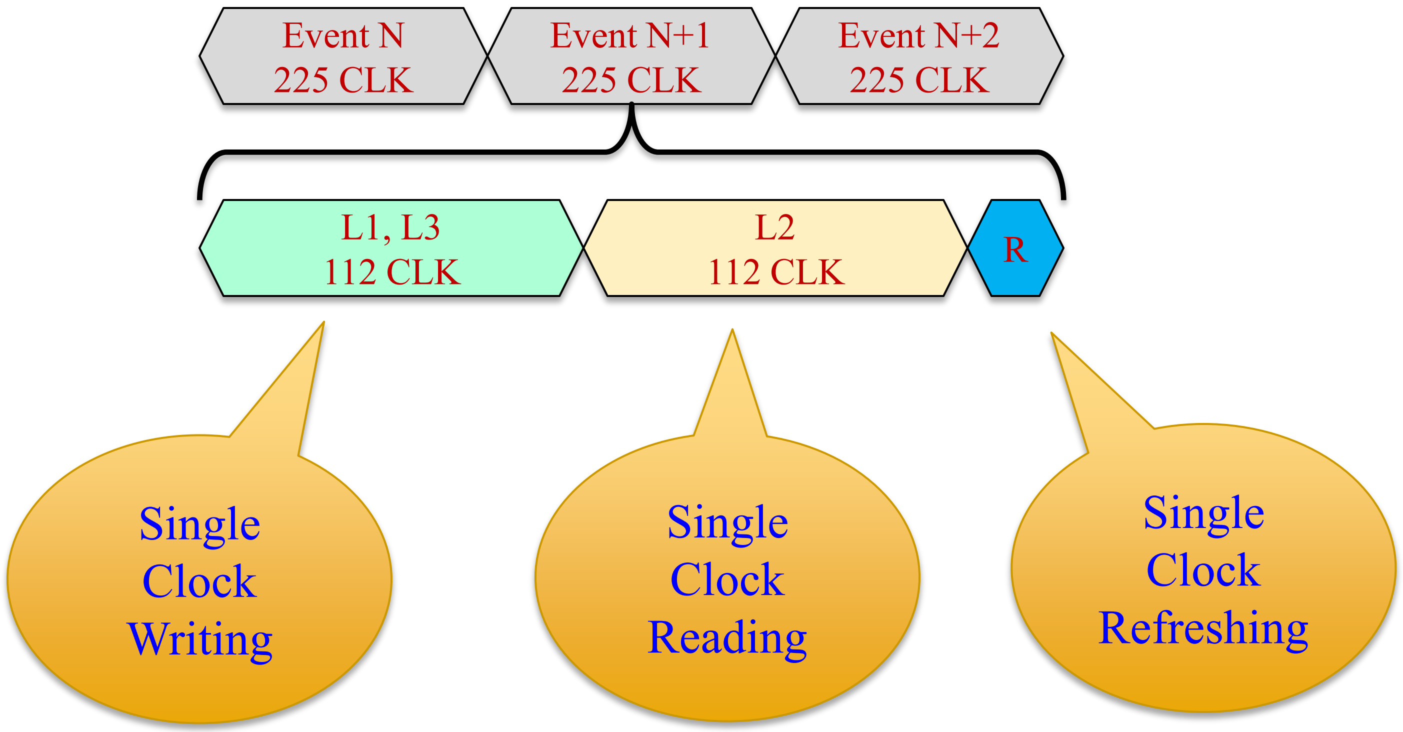
MemInit3.txt

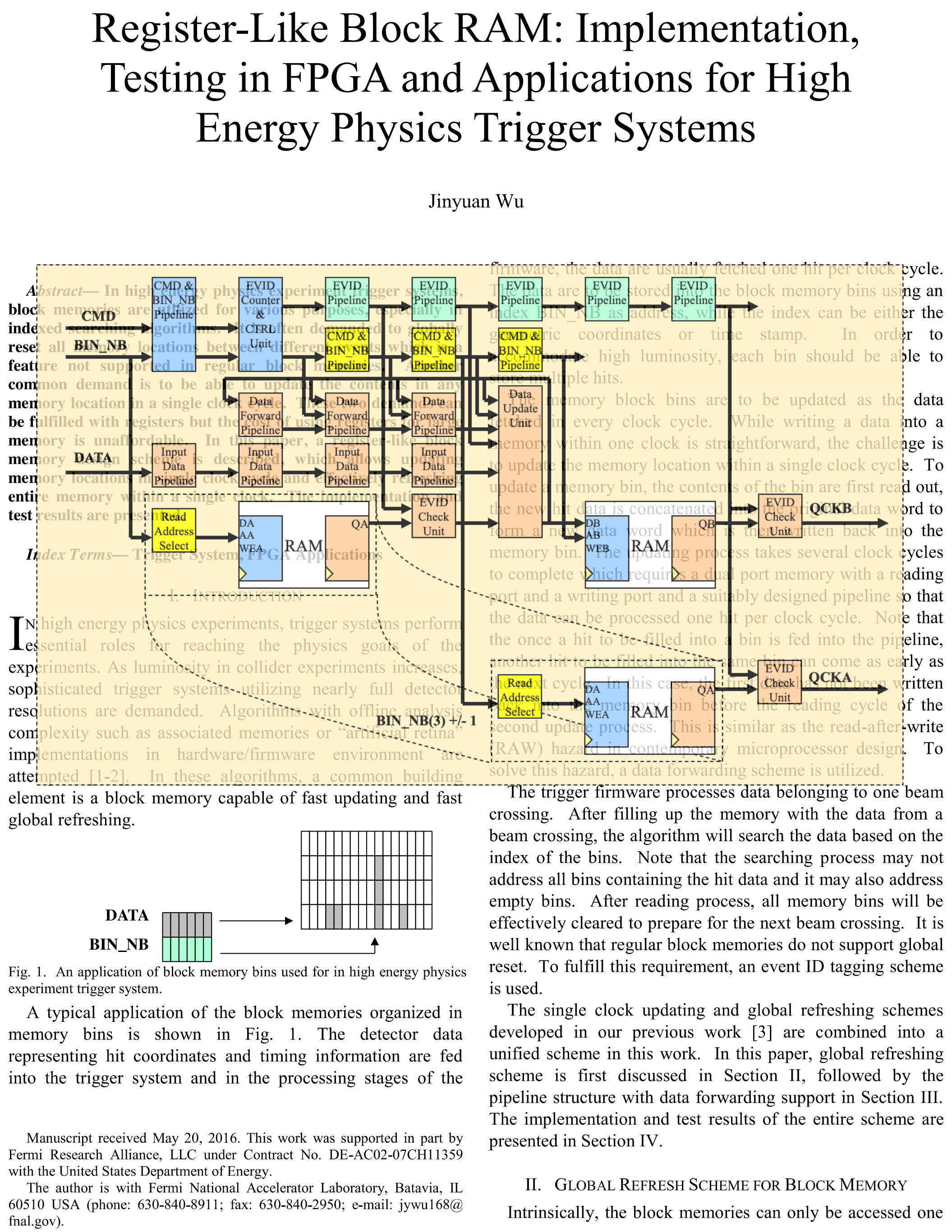
```
--Format: Quartus .mif

WIDTH=80;
DEPTH=256;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN
00 : 22222324242525262627;
01 : 23232424252526272728;
02 : 23242425262627272828;
03 : 24252526262727282929;
04 : 2525262627282829292a;
05 : 252627272828292a2a2b;
06 : 262727282829 2a2a2b2b;
07 : 27272829292a2a2b2b2c;
08 : 282829292a2a2b2c2c2d;
09 : 2829292a2b2b2c2c2d2d;
0a : 292a2a2b2b2c2c2d2e2e;
0b : 2a2a2b2b2c2d2d2e2e2f;
0c : 2a2b2c2c2d2d2e2e2f30;
0d : 2b2c2c2d2d2e2f2f3030;
0e : 2c2c2d2e2e2f2f303031;
0f : 2d2d2e2e2f2f30313132;
10 : 2d2e2e2f303031313232;
11 : 2e2f2f30303131323333;
12 : 2f2f3030313232333334;
13 : 2f303131323233333435;
14 : 30313132323334343535;
15 : 31313233333434353536;
16 : 32323333343435363637;
17 : 32333334353536363737;
18 : 33343435353636373838;
19 : 34343535363737383839;
1a : 3435363637373838393a;
1b : 353636373738393 93a3a;
1c : 363637383839393a3a3b;
1d : 3737383839393a3b3b3c;
1e : 373838393a3a3b3b3c3c;
1f : 3839393a3a3b3b3c3d3d;
20 : 39393a3a3b3c3c3d3d3e;
21 : 393a3b3b3c3c3d3d3e3f;
22 : 3a3b3b3c3c3d3e3e3f3f;
23 : 3b3b3c3d3d3e3e3f3f40;
24 : 3c3c3d3d3e3e3f404041;
25 : 3c3d3d3e3f3f40404141;
26 : 3d3e3e3f3f4040414242;
27 : 3e3e3f3f404141424243;
28 : 3e3f4040414142424344;
29 : 3f404041414243434444;
2a : 40404142424343444445;
2b : 41414242434344454546;
2c : 41424243444445454646;
2d : 42434344444545464747;
2e : 43434444454646474748;
2f : 43444545464647474849;
30 : 44454546464748484949;
31 : 4545464747484849494a;
```

The Seeding Engine



Register-Like Block RAM for Hit Patterns





Register-Like Block RAM: Implementation, Testing in FPGA and Applications for High Energy Physics Trigger Systems

Jinyuan Wu

element is a block memory capable of fast updating and fast global refreshing.

Fig. 1. An application of block memory bins used for in high energy physics experiment trigger system.

A typical application of the block memories organized in memory bins is shown in Fig. 1. The detector data representing hit coordinates and timing information are fed into the trigger system and in the processing stages of the

Manuscript received May 20, 2016. This work was supported in part by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

The author is with Fermi National Accelerator Laboratory, Batavia, IL 60510 USA (phone: 630-840-8911; fax: 630-840-2950; e-mail: jywu168@fnal.gov).

solve this hazard, a data forwarding scheme is utilized.

The trigger firmware processes data belonging to one beam crossing. After filling up the memory with the data from a beam crossing, the algorithm will search the data based on the index of the bins. Note that the searching process may not address all bins containing the hit data and it may also address empty bins. After reading process, all memory bins will be effectively cleared to prepare for the next beam crossing. It is well known that regular block memories do not support global reset. To fulfill this requirement, an event ID tagging scheme is used.

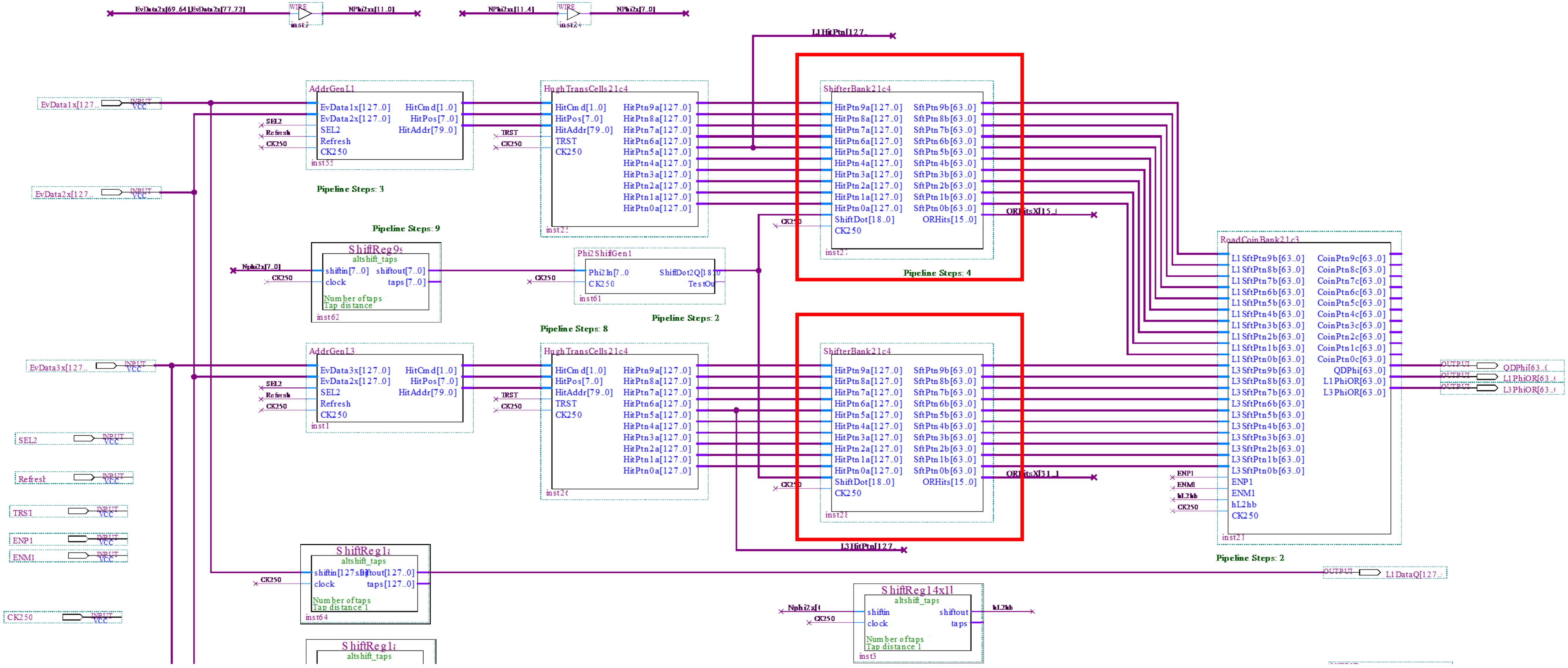
The single clock updating and global refreshing schemes developed in our previous work [3] are combined into a unified scheme in this work. In this paper, global refreshing scheme is first discussed in Section II, followed by the pipeline structure with data forwarding support in Section III. The implementation and test results of the entire scheme are presented in Section IV.

II. GLOBAL REFRESH SCHEME FOR BLOCK MEMORY

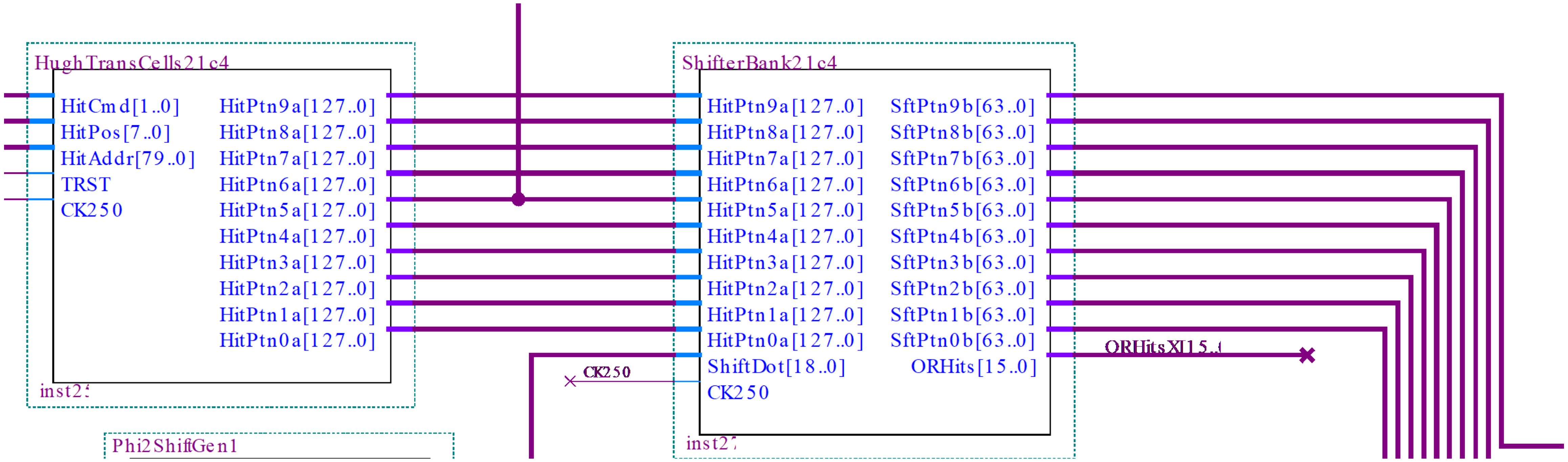
Intrinsically, the block memories can only be accessed one

- The hit patterns are stored in the register-like BRAM.
- Each hit is processed in a single clock cycle.
- Single clock cycle refreshing allows fast preparation for next event.

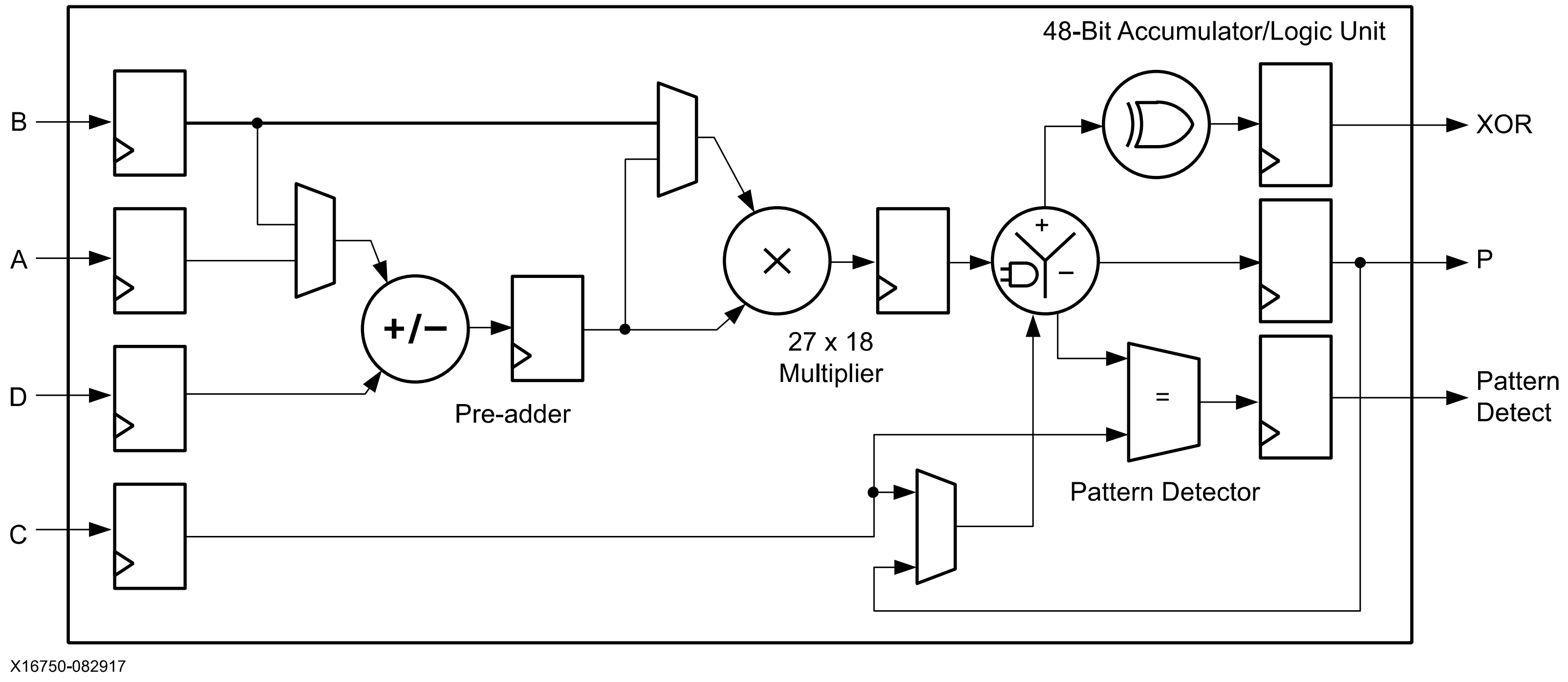
The Seeding Engine



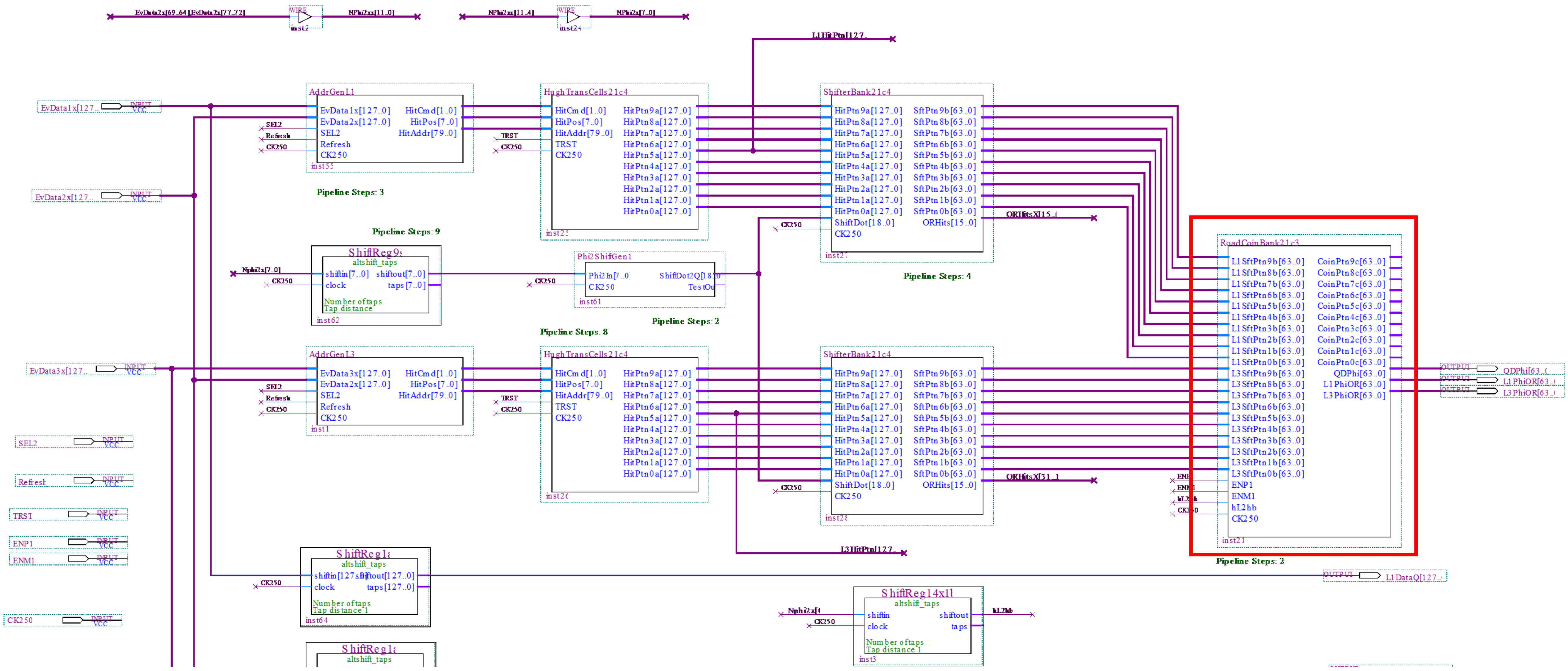
Shifters



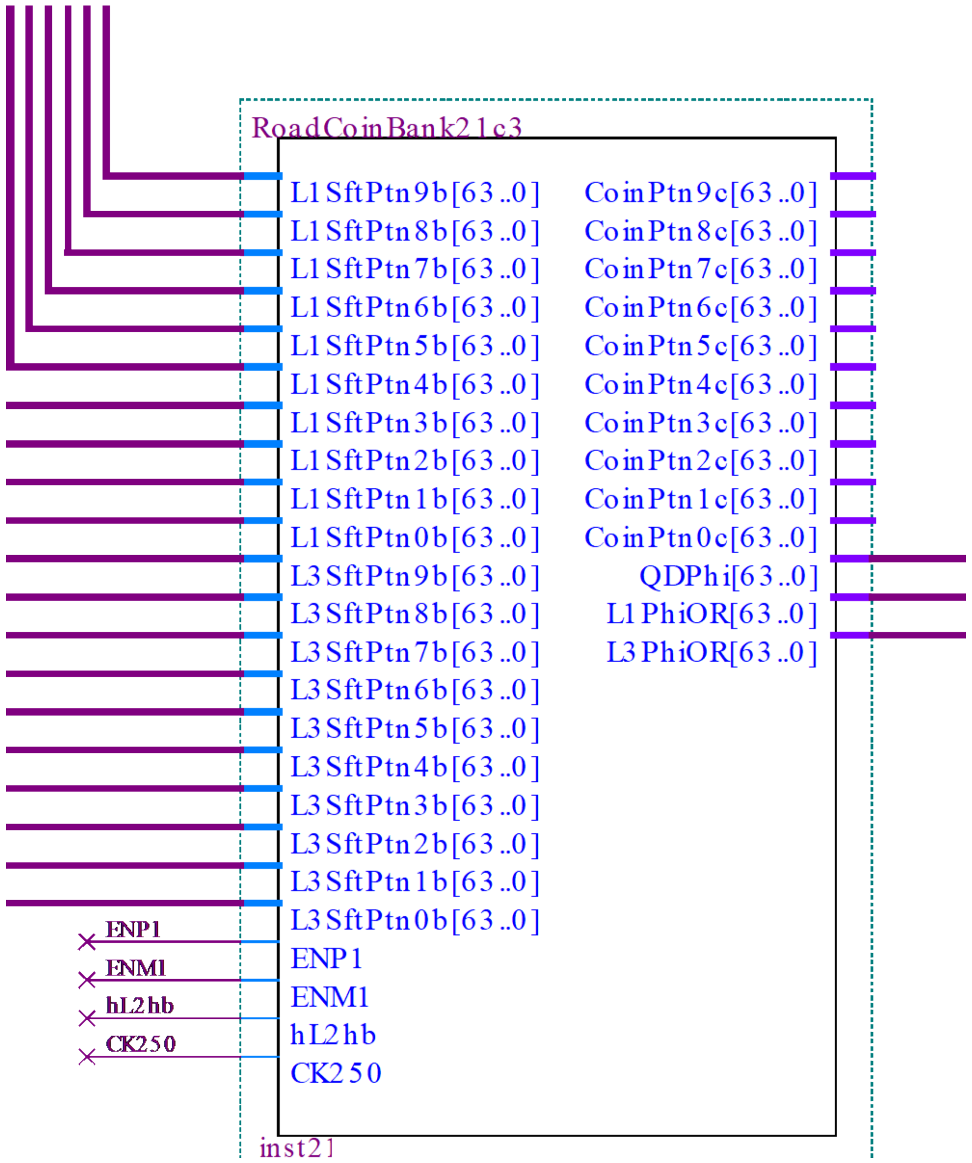
- The multipliers in FPGA DSPs are used to implement shifters.

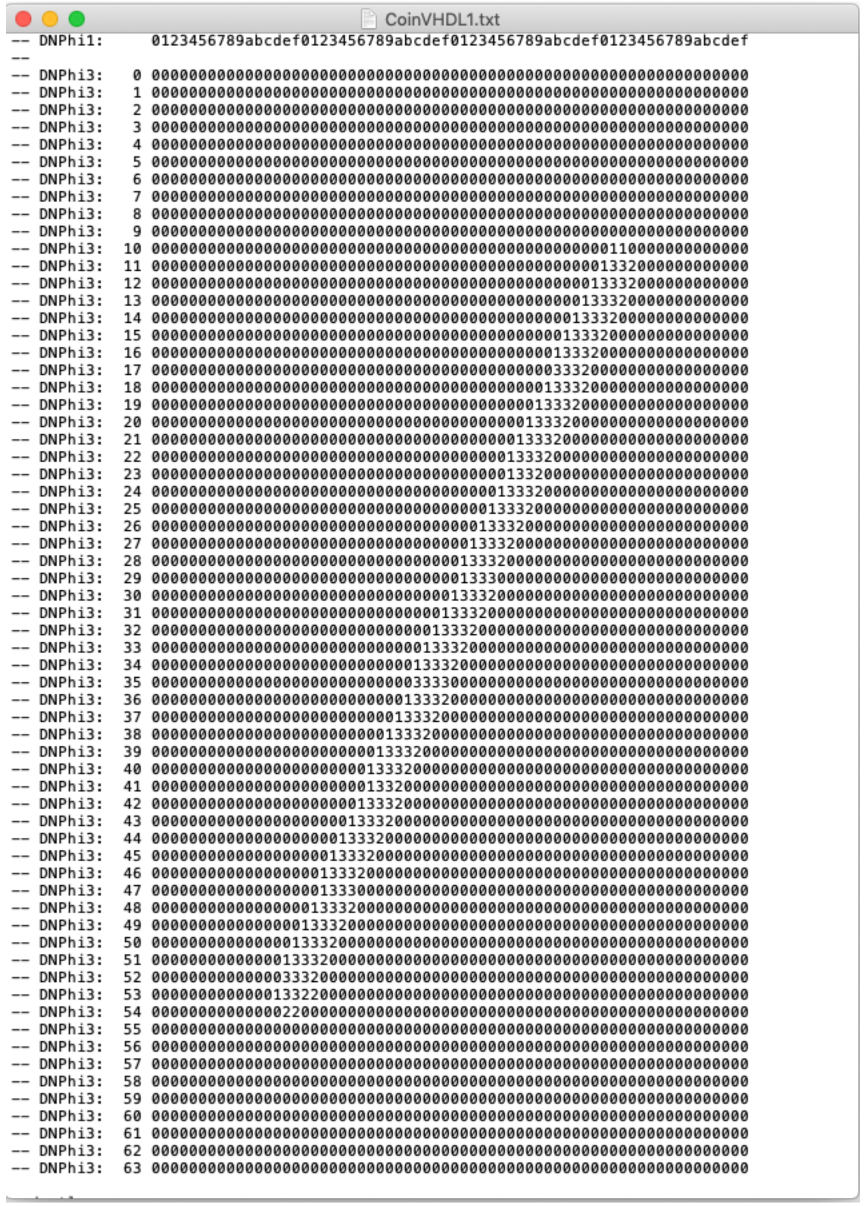


The Seeding Engine

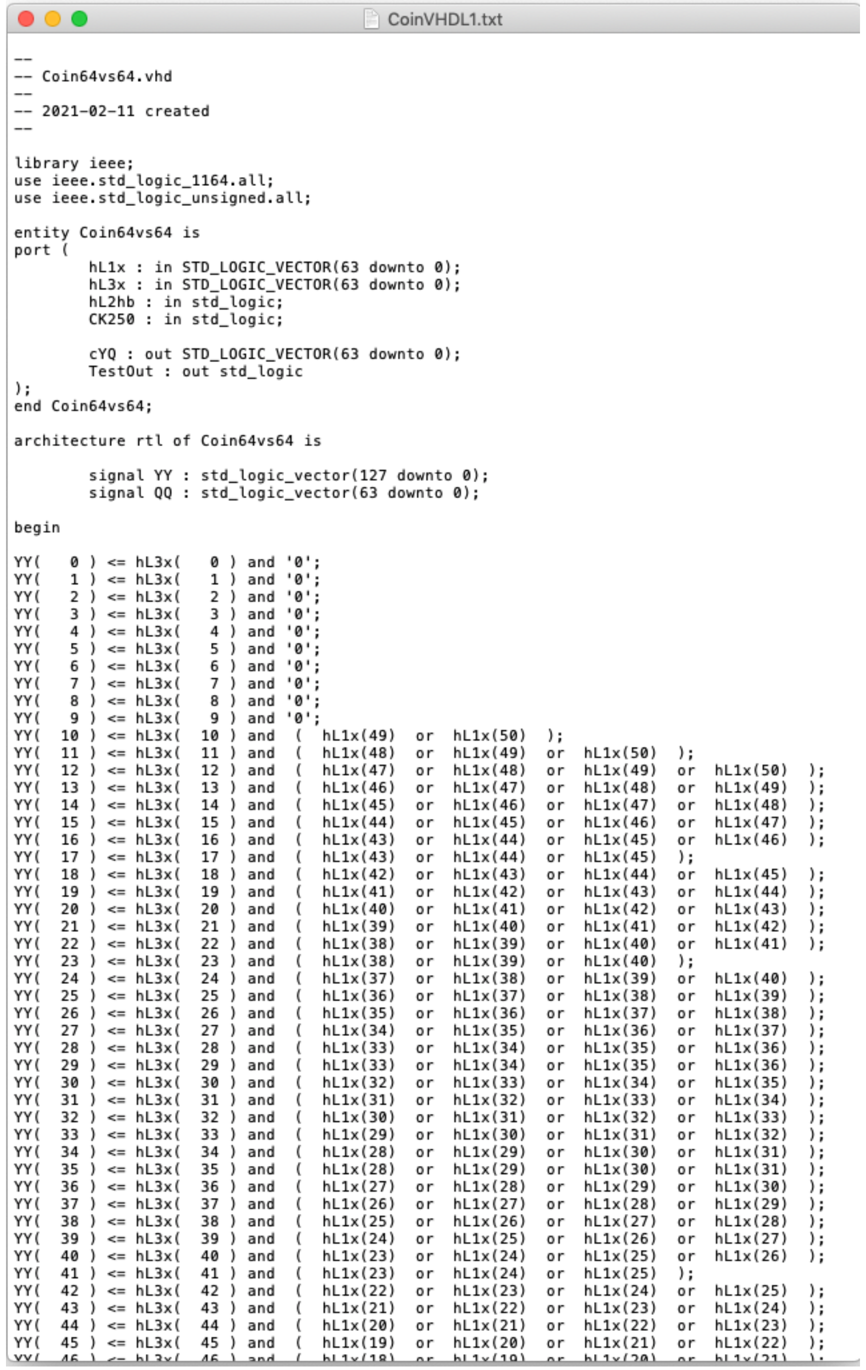


Road Coincidence




CoinVHDL1.txt

```
-- DNPhi1:      0123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef
--
-- DNPhi3:   0 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:   1 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:   2 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:   3 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:   4 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:   5 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:   6 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:   7 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:   8 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:   9 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:  10 0000000000000000000000000000000000000000000000000110000000000000
-- DNPhi3:  11 0000000000000000000000000000000000000000000000001332000000000000
-- DNPhi3:  12 0000000000000000000000000000000000000000000000013332000000000000
-- DNPhi3:  13 0000000000000000000000000000000000000000000000133320000000000000
-- DNPhi3:  14 0000000000000000000000000000000000000000000001333200000000000000
-- DNPhi3:  15 0000000000000000000000000000000000000000000013332000000000000000
-- DNPhi3:  16 0000000000000000000000000000000000000000000133320000000000000000
-- DNPhi3:  17 0000000000000000000000000000000000000000000333200000000000000000
-- DNPhi3:  18 0000000000000000000000000000000000000000001333200000000000000000
-- DNPhi3:  19 0000000000000000000000000000000000000000013332000000000000000000
-- DNPhi3:  20 0000000000000000000000000000000000000000133320000000000000000000
-- DNPhi3:  21 0000000000000000000000000000000000000001333200000000000000000000
-- DNPhi3:  22 0000000000000000000000000000000000000013332000000000000000000000
-- DNPhi3:  23 0000000000000000000000000000000000000013320000000000000000000000
-- DNPhi3:  24 0000000000000000000000000000000000000133320000000000000000000000
-- DNPhi3:  25 0000000000000000000000000000000000001333200000000000000000000000
-- DNPhi3:  26 0000000000000000000000000000000000013332000000000000000000000000
-- DNPhi3:  27 0000000000000000000000000000000000133320000000000000000000000000
-- DNPhi3:  28 0000000000000000000000000000000001333200000000000000000000000000
-- DNPhi3:  29 0000000000000000000000000000000001333000000000000000000000000000
-- DNPhi3:  30 0000000000000000000000000000000013332000000000000000000000000000
-- DNPhi3:  31 0000000000000000000000000000000133320000000000000000000000000000
-- DNPhi3:  32 0000000000000000000000000000001333200000000000000000000000000000
-- DNPhi3:  33 0000000000000000000000000000013332000000000000000000000000000000
-- DNPhi3:  34 0000000000000000000000000000133320000000000000000000000000000000
-- DNPhi3:  35 0000000000000000000000000000333300000000000000000000000000000000
-- DNPhi3:  36 0000000000000000000000000001333200000000000000000000000000000000
-- DNPhi3:  37 0000000000000000000000000013332000000000000000000000000000000000
-- DNPhi3:  38 0000000000000000000000000133320000000000000000000000000000000000
-- DNPhi3:  39 0000000000000000000000001333200000000000000000000000000000000000
-- DNPhi3:  40 0000000000000000000000013332000000000000000000000000000000000000
-- DNPhi3:  41 0000000000000000000000013320000000000000000000000000000000000000
-- DNPhi3:  42 0000000000000000000000133320000000000000000000000000000000000000
-- DNPhi3:  43 0000000000000000000001333200000000000000000000000000000000000000
-- DNPhi3:  44 0000000000000000000013332000000000000000000000000000000000000000
-- DNPhi3:  45 0000000000000000000133320000000000000000000000000000000000000000
-- DNPhi3:  46 0000000000000000001333200000000000000000000000000000000000000000
-- DNPhi3:  47 0000000000000000001333000000000000000000000000000000000000000000
-- DNPhi3:  48 0000000000000000013332000000000000000000000000000000000000000000
-- DNPhi3:  49 0000000000000000133320000000000000000000000000000000000000000000
-- DNPhi3:  50 0000000000000001333200000000000000000000000000000000000000000000
-- DNPhi3:  51 0000000000000013332000000000000000000000000000000000000000000000
-- DNPhi3:  52 0000000000000033320000000000000000000000000000000000000000000000
-- DNPhi3:  53 0000000000000133220000000000000000000000000000000000000000000000
-- DNPhi3:  54 0000000000000002200000000000000000000000000000000000000000000000
-- DNPhi3:  55 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:  56 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:  57 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:  58 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:  59 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:  60 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:  61 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:  62 0000000000000000000000000000000000000000000000000000000000000000
-- DNPhi3:  63 0000000000000000000000000000000000000000000000000000000000000000
```

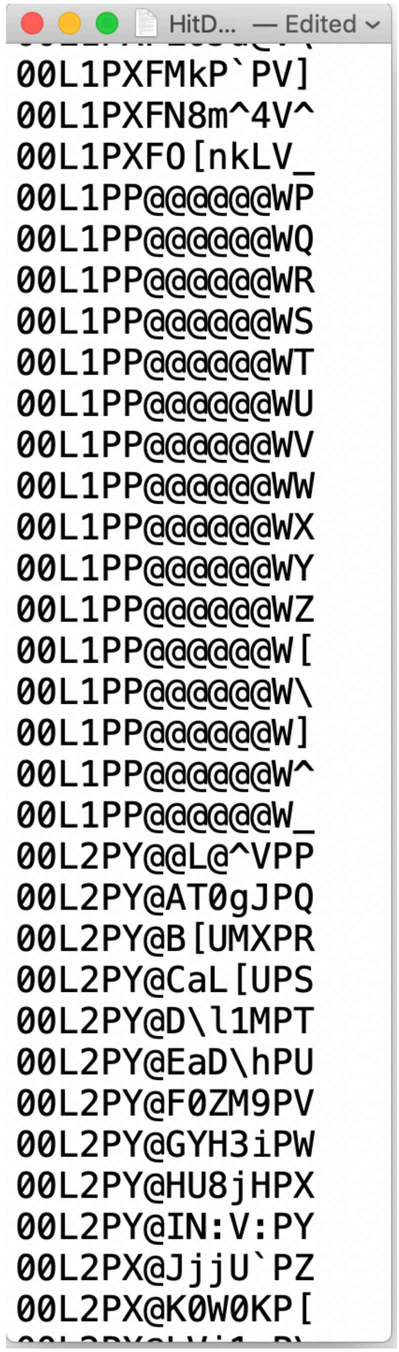

CoinVHDL1.txt

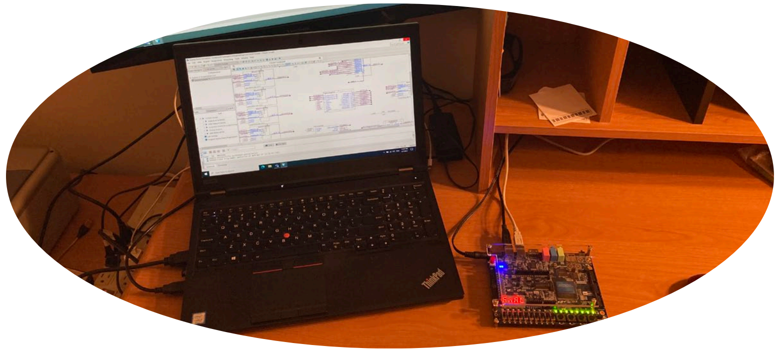
```
--
-- Coin64vs64.vhd
--
-- 2021-02-11 created
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Coin64vs64 is
port (
        hL1x : in STD_LOGIC_VECTOR(63 downto 0);
        hL3x : in STD_LOGIC_VECTOR(63 downto 0);
        hL2hb : in std_logic;
        CK250 : in std_logic;

        cYQ : out STD_LOGIC_VECTOR(63 downto 0);
        TestOut : out std_logic
);
end Coin64vs64;

architecture rtl of Coin64vs64 is

        signal YY : std_logic_vector(127 downto 0);
        signal QQ : std_logic_vector(63 downto 0);

begin

YY(   0 ) <= hL3x(   0 ) and '0';
YY(   1 ) <= hL3x(   1 ) and '0';
YY(   2 ) <= hL3x(   2 ) and '0';
YY(   3 ) <= hL3x(   3 ) and '0';
YY(   4 ) <= hL3x(   4 ) and '0';
YY(   5 ) <= hL3x(   5 ) and '0';
YY(   6 ) <= hL3x(   6 ) and '0';
YY(   7 ) <= hL3x(   7 ) and '0';
YY(   8 ) <= hL3x(   8 ) and '0';
YY(   9 ) <= hL3x(   9 ) and '0';
YY(  10 ) <= hL3x(  10 ) and  (  hL1x(49)  or  hL1x(50)  );
YY(  11 ) <= hL3x(  11 ) and  (  hL1x(48)  or  hL1x(49)  or  hL1x(50)  );
YY(  12 ) <= hL3x(  12 ) and  (  hL1x(47)  or  hL1x(48)  or  hL1x(49)  or  hL1x(50)  );
YY(  13 ) <= hL3x(  13 ) and  (  hL1x(46)  or  hL1x(47)  or  hL1x(48)  or  hL1x(49)  );
YY(  14 ) <= hL3x(  14 ) and  (  hL1x(45)  or  hL1x(46)  or  hL1x(47)  or  hL1x(48)  );
YY(  15 ) <= hL3x(  15 ) and  (  hL1x(44)  or  hL1x(45)  or  hL1x(46)  or  hL1x(47)  );
YY(  16 ) <= hL3x(  16 ) and  (  hL1x(43)  or  hL1x(44)  or  hL1x(45)  or  hL1x(46)  );
YY(  17 ) <= hL3x(  17 ) and  (  hL1x(43)  or  hL1x(44)  or  hL1x(45)  );
YY(  18 ) <= hL3x(  18 ) and  (  hL1x(42)  or  hL1x(43)  or  hL1x(44)  or  hL1x(45)  );
YY(  19 ) <= hL3x(  19 ) and  (  hL1x(41)  or  hL1x(42)  or  hL1x(43)  or  hL1x(44)  );
YY(  20 ) <= hL3x(  20 ) and  (  hL1x(40)  or  hL1x(41)  or  hL1x(42)  or  hL1x(43)  );
YY(  21 ) <= hL3x(  21 ) and  (  hL1x(39)  or  hL1x(40)  or  hL1x(41)  or  hL1x(42)  );
YY(  22 ) <= hL3x(  22 ) and  (  hL1x(38)  or  hL1x(39)  or  hL1x(40)  or  hL1x(41)  );
YY(  23 ) <= hL3x(  23 ) and  (  hL1x(38)  or  hL1x(39)  or  hL1x(40)  );
YY(  24 ) <= hL3x(  24 ) and  (  hL1x(37)  or  hL1x(38)  or  hL1x(39)  or  hL1x(40)  );
YY(  25 ) <= hL3x(  25 ) and  (  hL1x(36)  or  hL1x(37)  or  hL1x(38)  or  hL1x(39)  );
YY(  26 ) <= hL3x(  26 ) and  (  hL1x(35)  or  hL1x(36)  or  hL1x(37)  or  hL1x(38)  );
YY(  27 ) <= hL3x(  27 ) and  (  hL1x(34)  or  hL1x(35)  or  hL1x(36)  or  hL1x(37)  );
YY(  28 ) <= hL3x(  28 ) and  (  hL1x(33)  or  hL1x(34)  or  hL1x(35)  or  hL1x(36)  );
YY(  29 ) <= hL3x(  29 ) and  (  hL1x(33)  or  hL1x(34)  or  hL1x(35)  or  hL1x(36)  );
YY(  30 ) <= hL3x(  30 ) and  (  hL1x(32)  or  hL1x(33)  or  hL1x(34)  or  hL1x(35)  );
YY(  31 ) <= hL3x(  31 ) and  (  hL1x(31)  or  hL1x(32)  or  hL1x(33)  or  hL1x(34)  );
YY(  32 ) <= hL3x(  32 ) and  (  hL1x(30)  or  hL1x(31)  or  hL1x(32)  or  hL1x(33)  );
YY(  33 ) <= hL3x(  33 ) and  (  hL1x(29)  or  hL1x(30)  or  hL1x(31)  or  hL1x(32)  );
YY(  34 ) <= hL3x(  34 ) and  (  hL1x(28)  or  hL1x(29)  or  hL1x(30)  or  hL1x(31)  );
YY(  35 ) <= hL3x(  35 ) and  (  hL1x(28)  or  hL1x(29)  or  hL1x(30)  or  hL1x(31)  );
YY(  36 ) <= hL3x(  36 ) and  (  hL1x(27)  or  hL1x(28)  or  hL1x(29)  or  hL1x(30)  );
YY(  37 ) <= hL3x(  37 ) and  (  hL1x(26)  or  hL1x(27)  or  hL1x(28)  or  hL1x(29)  );
YY(  38 ) <= hL3x(  38 ) and  (  hL1x(25)  or  hL1x(26)  or  hL1x(27)  or  hL1x(28)  );
YY(  39 ) <= hL3x(  39 ) and  (  hL1x(24)  or  hL1x(25)  or  hL1x(26)  or  hL1x(27)  );
YY(  40 ) <= hL3x(  40 ) and  (  hL1x(23)  or  hL1x(24)  or  hL1x(25)  or  hL1x(26)  );
YY(  41 ) <= hL3x(  41 ) and  (  hL1x(23)  or  hL1x(24)  or  hL1x(25)  );
YY(  42 ) <= hL3x(  42 ) and  (  hL1x(22)  or  hL1x(23)  or  hL1x(24)  or  hL1x(25)  );
YY(  43 ) <= hL3x(  43 ) and  (  hL1x(21)  or  hL1x(22)  or  hL1x(23)  or  hL1x(24)  );
YY(  44 ) <= hL3x(  44 ) and  (  hL1x(20)  or  hL1x(21)  or  hL1x(22)  or  hL1x(23)  );
YY(  45 ) <= hL3x(  45 ) and  (  hL1x(19)  or  hL1x(20)  or  hL1x(21)  or  hL1x(22)  );
YY(  46 ) <= hL3x(  46 ) and  (  hL1x(18)  or  hL1x(19)  or  hL1x(20)  or  hL1x(21)  );
```

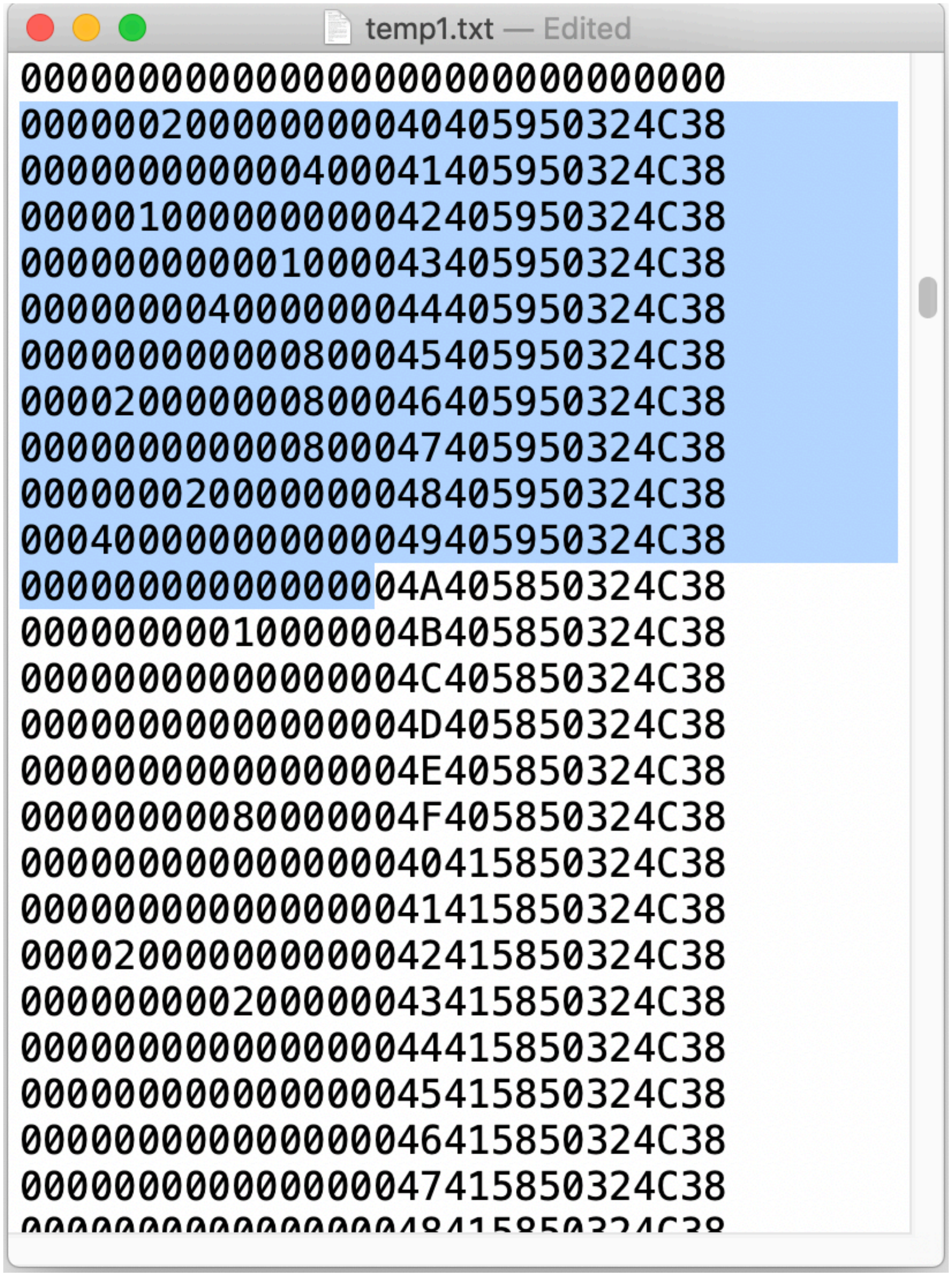
- Shifted bit patterns are checked by coincidence road array.
- The r-phi logic equations are generated with simulation.

Test Results

Raw Input Data and Output Data

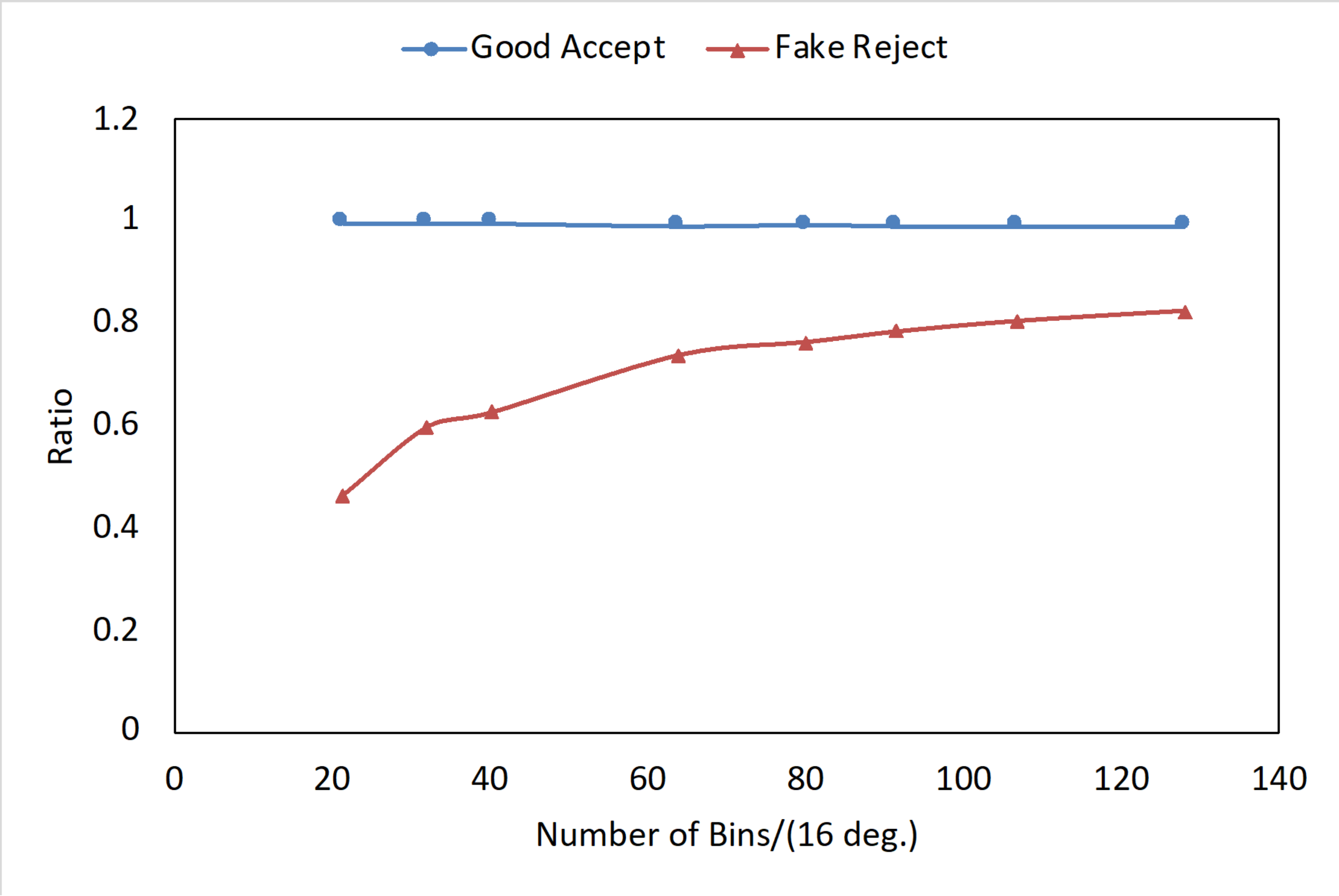






- Hits (stubs) are sent into the seeding engine.
- Each line is a hit in a detector layer.
- Typical data: 100 hits/layer with 10 good tracks with 90 random hits.
- The seeding engine outputs a 64-bit pattern, and the non-zero bits represent a potential track segment.

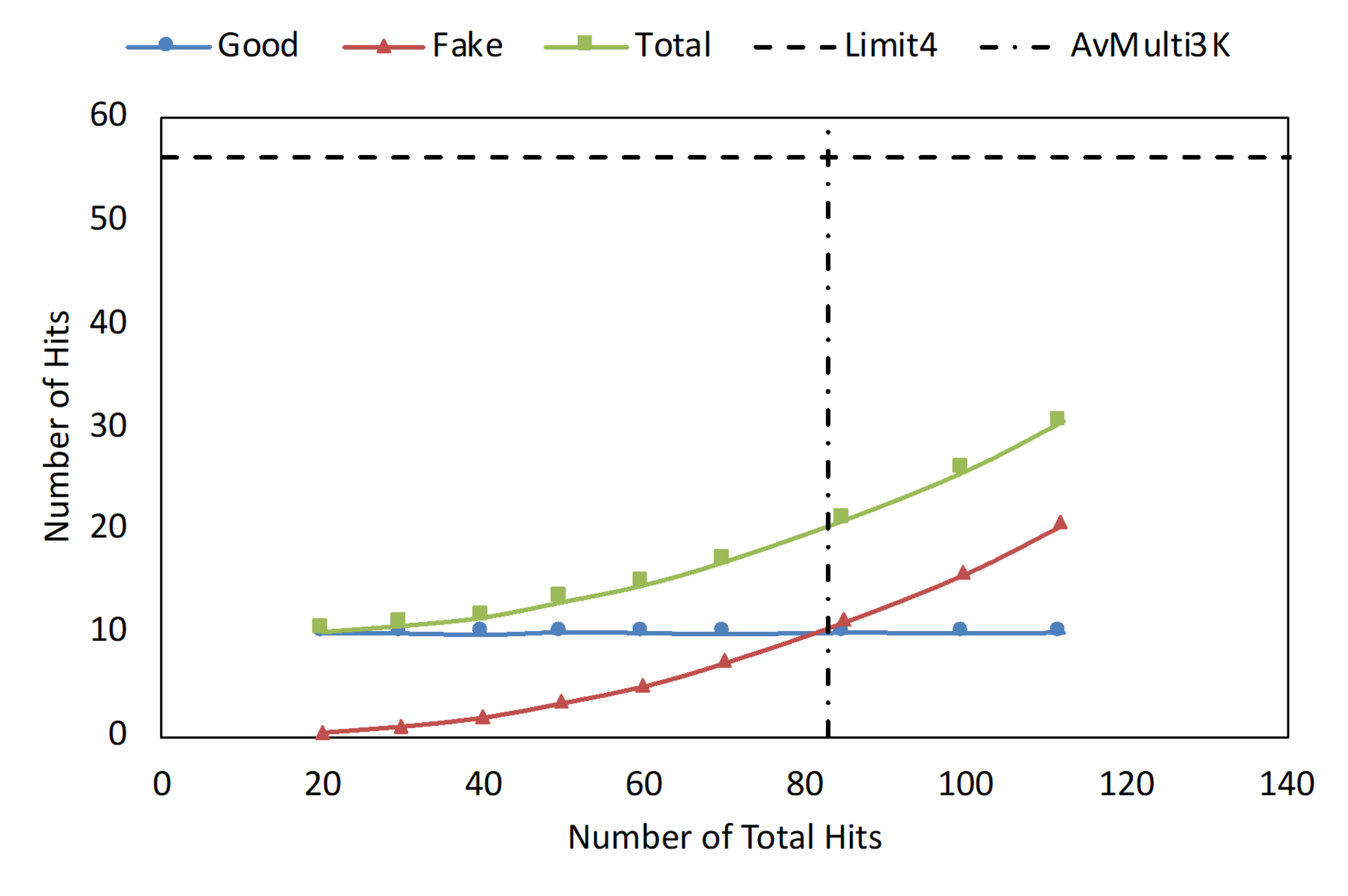
Good Track Acceptance and Fake Rejection



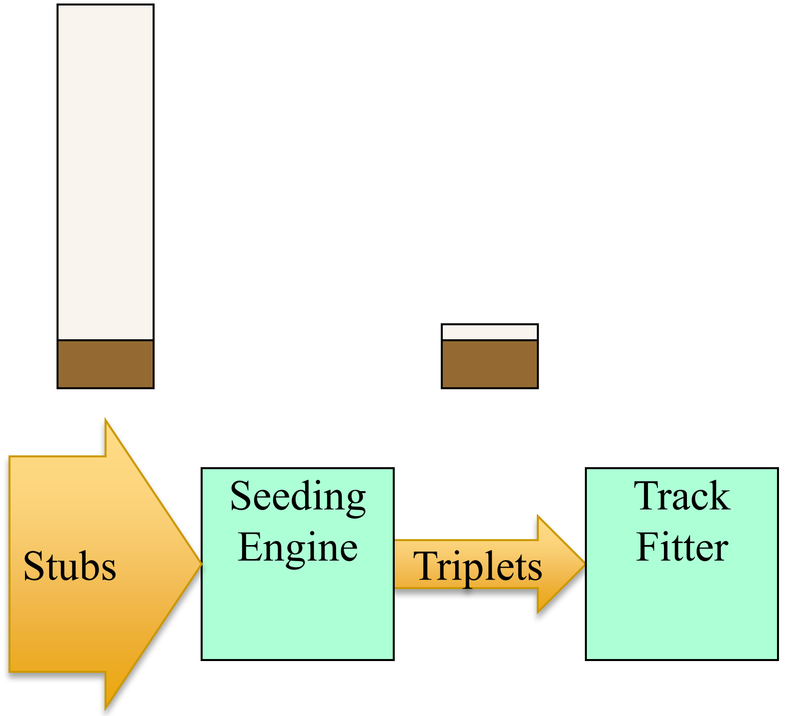
- Good track acceptance is better than 99%.
- Increasing number of bins in phi causes better fake rejection.

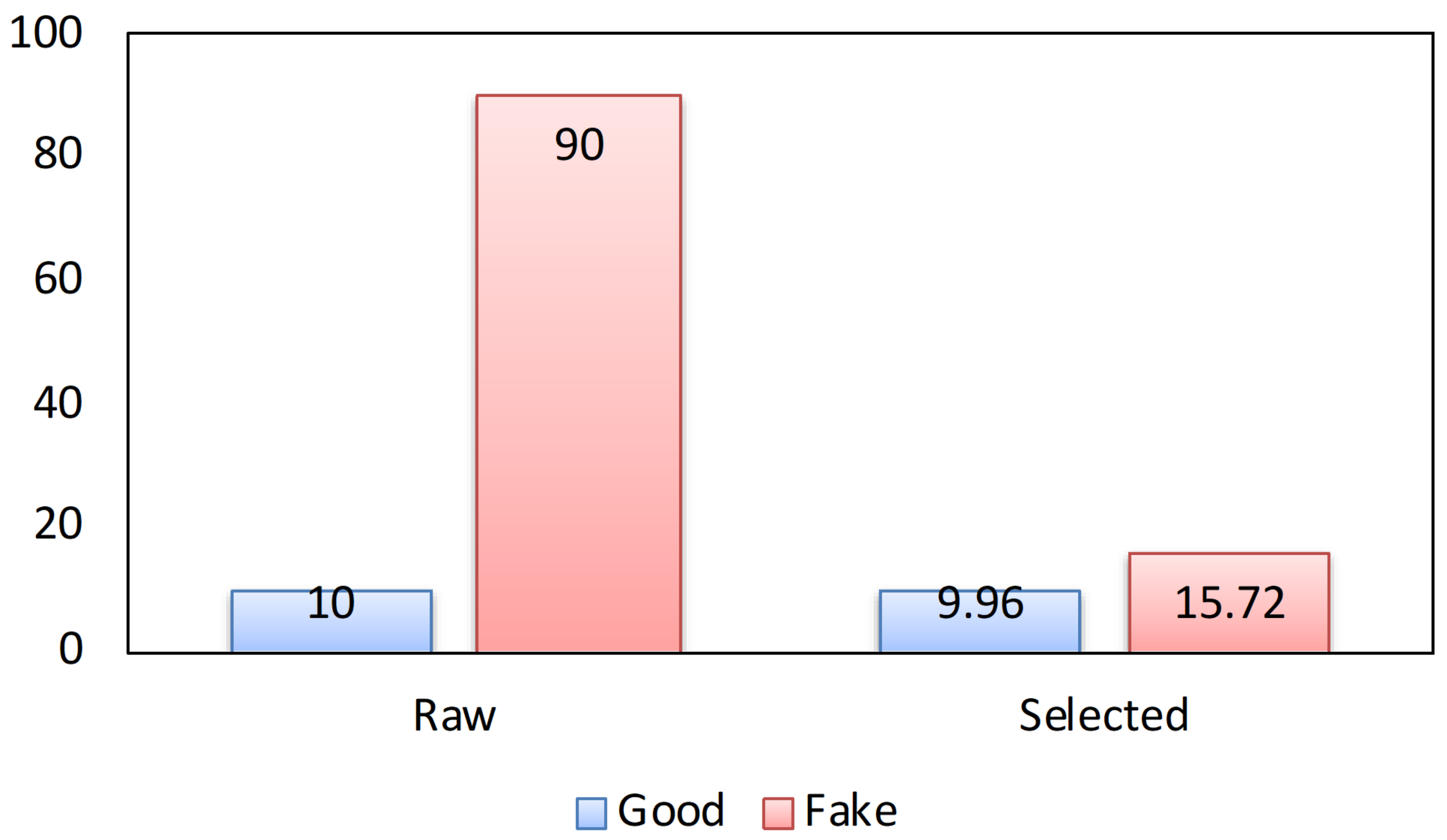
Fake Tracks due to High Occupancy

- When number of total hits per event (BX) is low, fake rate is low.
- Event beyond occupancy of 3000 stubs/layer for the whole detector (the AvMulti3K line), total number of coincidence is still manageable.
- The dash line Limit4 indicate total coincidence allowed in fitting stage if each coincidence is processed with 4 clock cycles.



Performance of the Seeding Engine



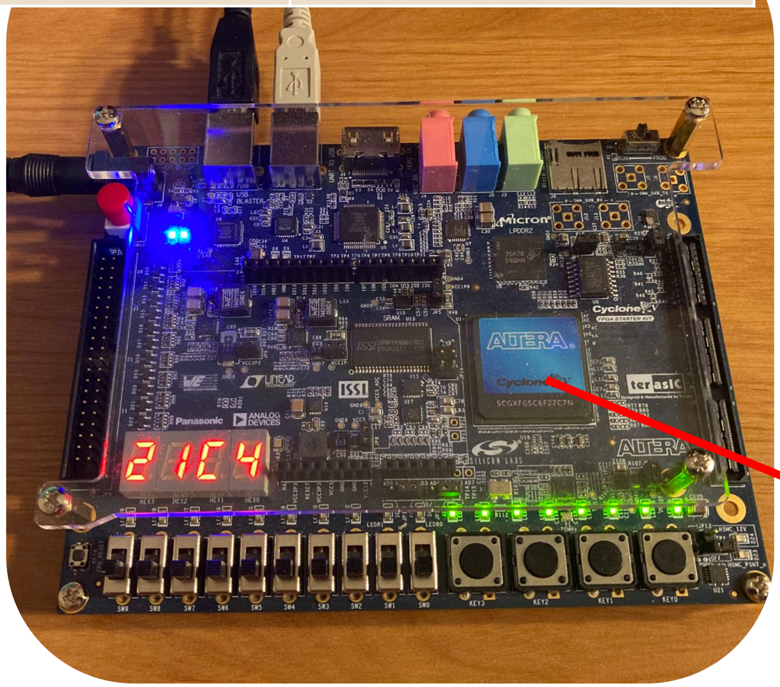


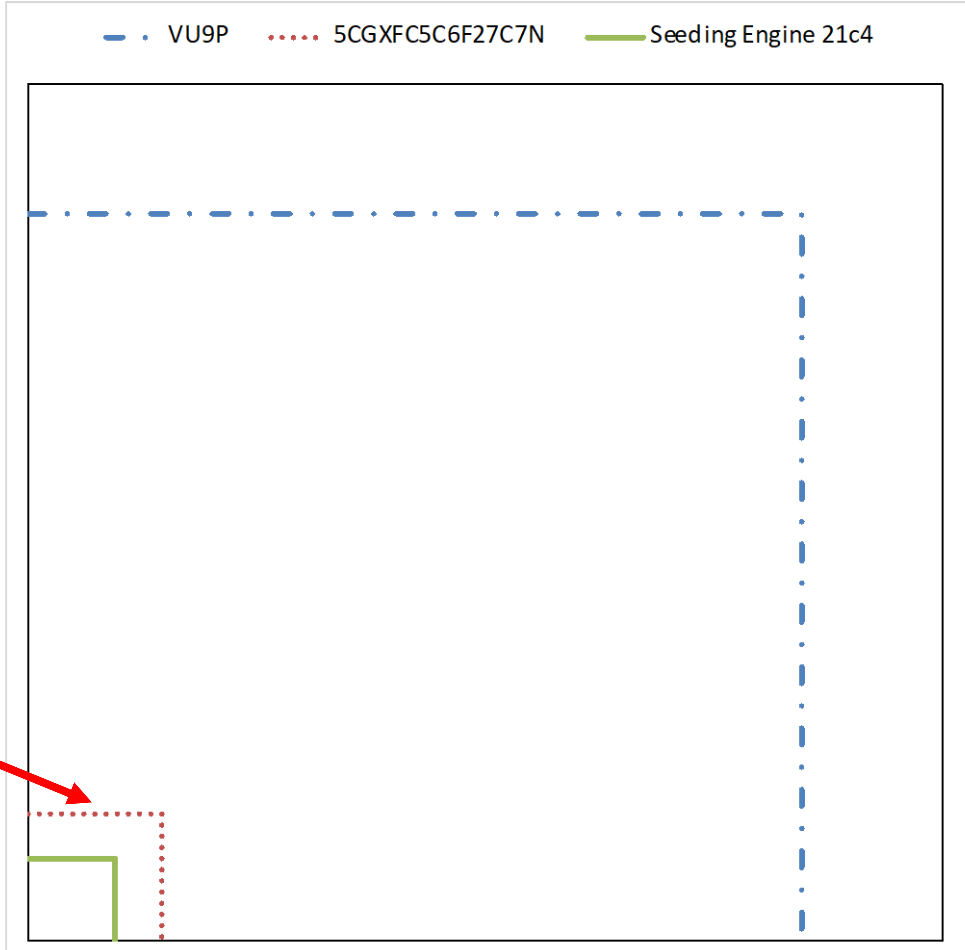
- Most of fake hits are rejected while good tracks are preserved.
- Data volume sent to the fitting stage is much smaller.

Resource Usage:

	Used in a Seeding Engine core	Available in Cyclone 5 5CGXFC5C6F27C7N	Available in Ultra-scale Plus VU9P
Logic Cells	12446 (ALM) Eq. 32.9K (LC)	29080 (ALM)	2586K (LC)
Block RAM	150 (M10K) 743K (bits)	446 (M10K) 4567K (bits)	75.9M (bits)
DSP	100 (16x16 Multipliers)	150 (DSP 27x27) Eq. 300 (16x16)	6840 (DSP 18x27)

- The seeding engine core fits the Cyclone 5 device with ~45% resource usage.
- Four cores are needed in VU9P and it looks possible.





Summary

- Algorithms: well-known in the field.
  - The fake rate of track segment seeding with triplets is lower.
- Implementations: exist in the field.
  - The challenge is how to fit into a reasonable sized FPGA.
- Methods of solving the implementation challenge: this work.
  - Using Tiny Triplet Finder so that only a small number of coincidence roads (product terms) are implemented.
  - Hit patterns are implemented with "register-like RAM" which allows single clock update, read and refresh operations.
  - Lookup tables and coincidence roads are auto initiated using simulation data to accommodate detector geometry or alignment variations.

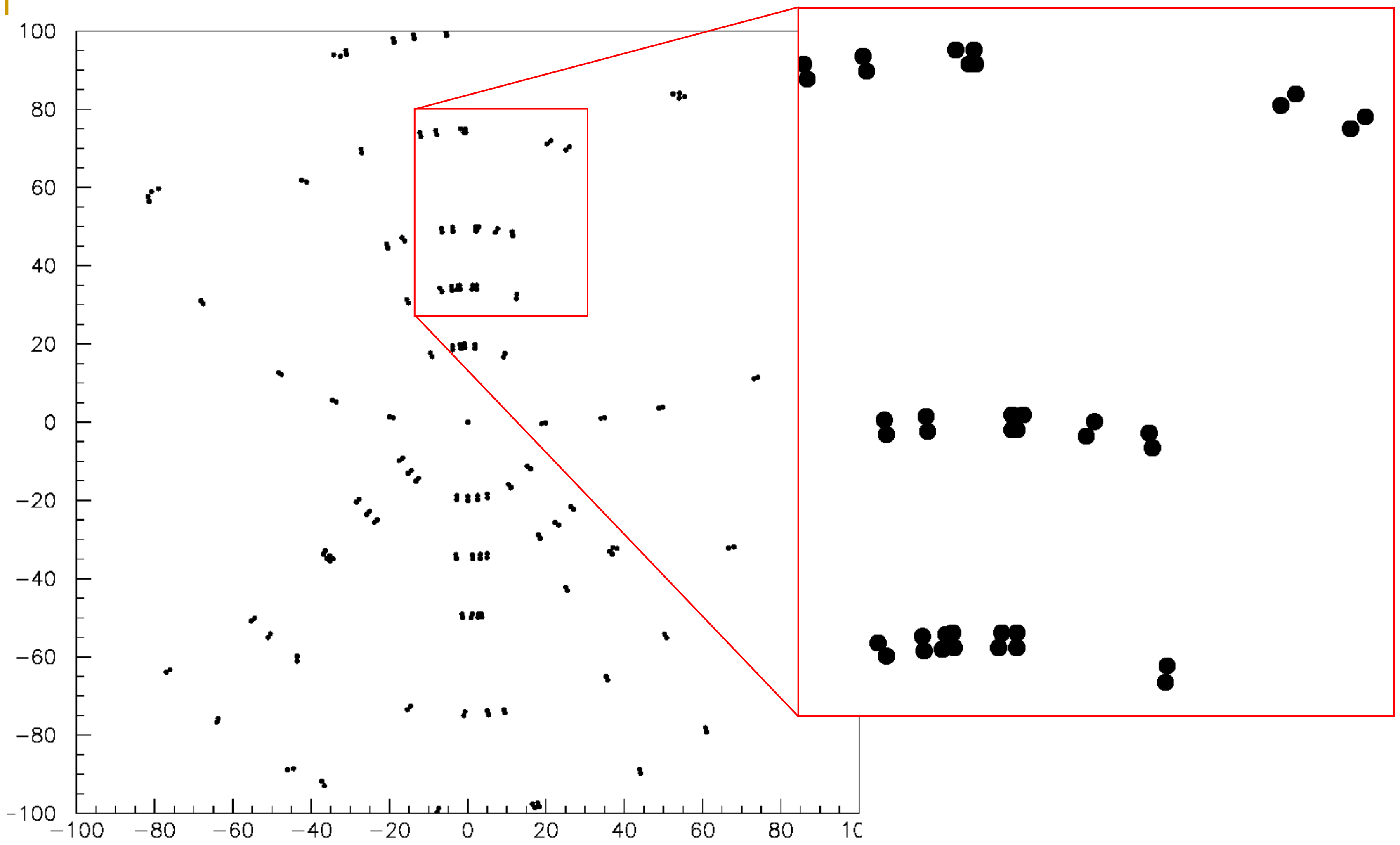
The End

Thanks

Introduction

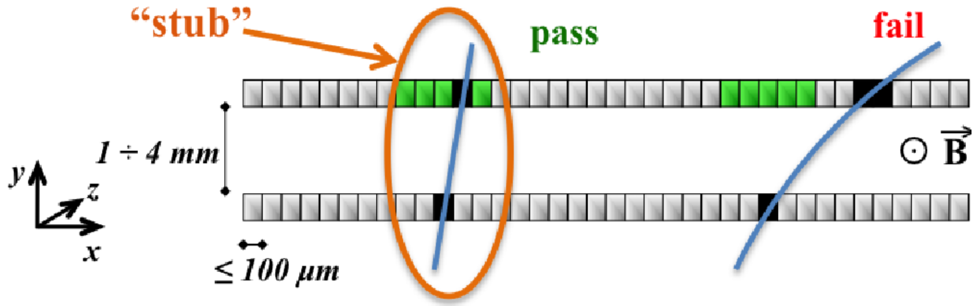
- For high-luminosity operation, 3D track segment seeding becomes necessary:
  - Usable bin sizes in z or phi direction can't be too fine due to:
    - Multiple scattering
    - Non-zero impact parameter
  - The 2D segment seeding would yield too many fake segments.
  - Constraints in both r-z and r-phi planes help to reduce fake segments.
- Using the Tiny Triplet Finder to implement 3D track segment seeding engines
  - The Tiny Triplet Finder is a low-resource usage scheme for track segment finding in FPGA.
  - It becomes affordable to implement 3D seeding engines since the resource usage is low.
- A true 3D track segment seeding engine has been implemented and tested.
  - Detector partition, data timing etc. are based on the seeding-fitting approach in recent CMS presentations: [Mark Pesaresi, ACES 2020] and [Luigi Calligaris RT2020].
  - Stubs are generated on idealized Outer Tracker 3 PS layers.
  - Operates as intended.

Pairing or Triplet Finding

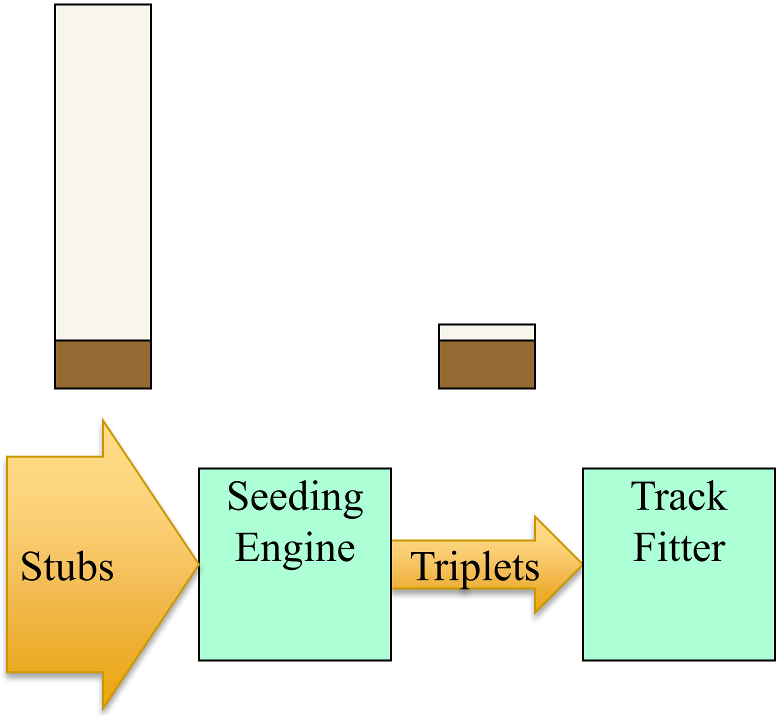


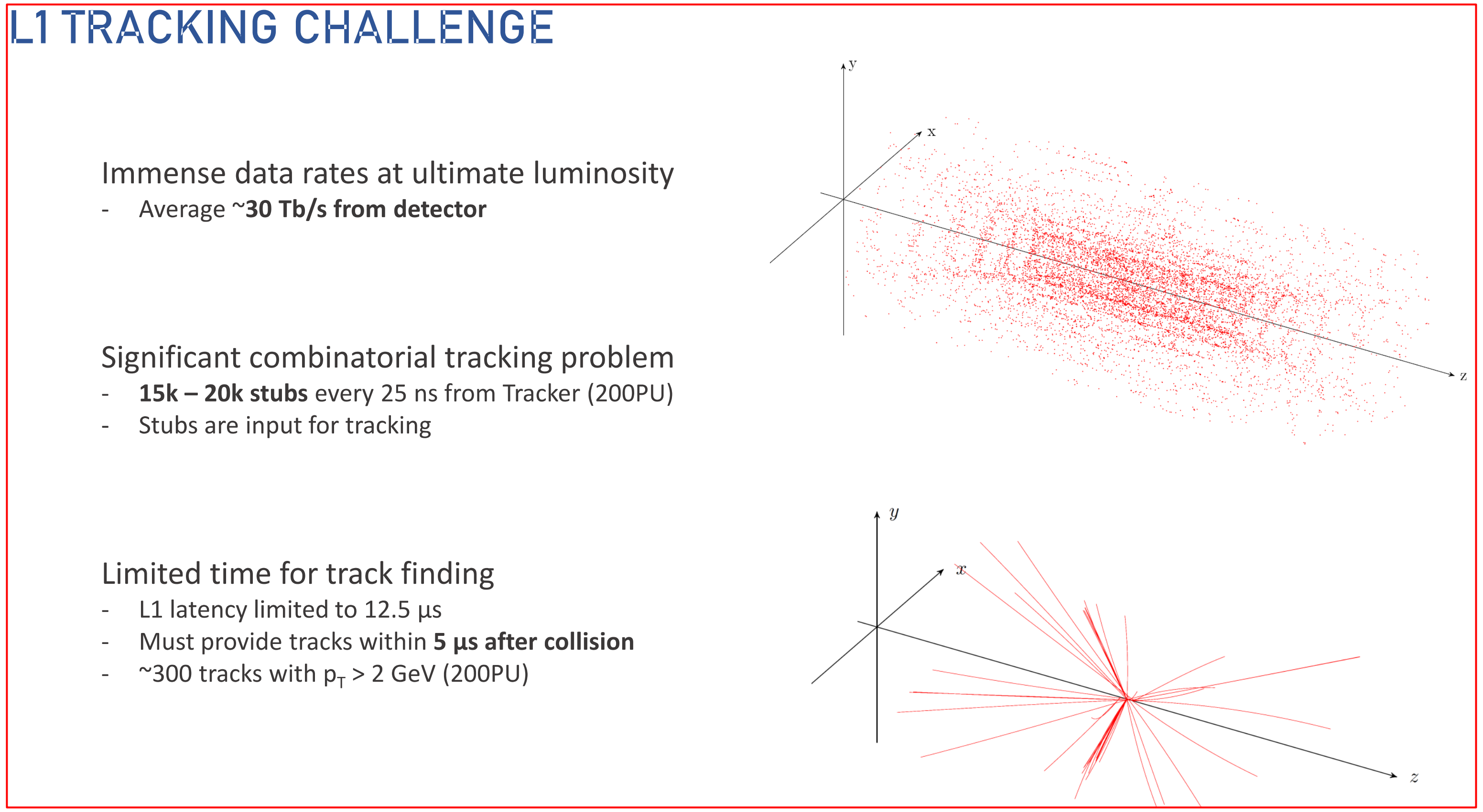
- If the separation of silicon sensor layers were large (~ 10 mm), then stub **pairing** would be sufficient.
- Given the separation of ~ 4 mm in PS and 2S module, correlating **triplets** of stubs becomes necessary.

Wu et. al. 2004
https://indico.cern.ch/event/418639/contributions/1018451/attachments/868833/1216631/lowpt_lecc2004p.pdf



Why Seeding



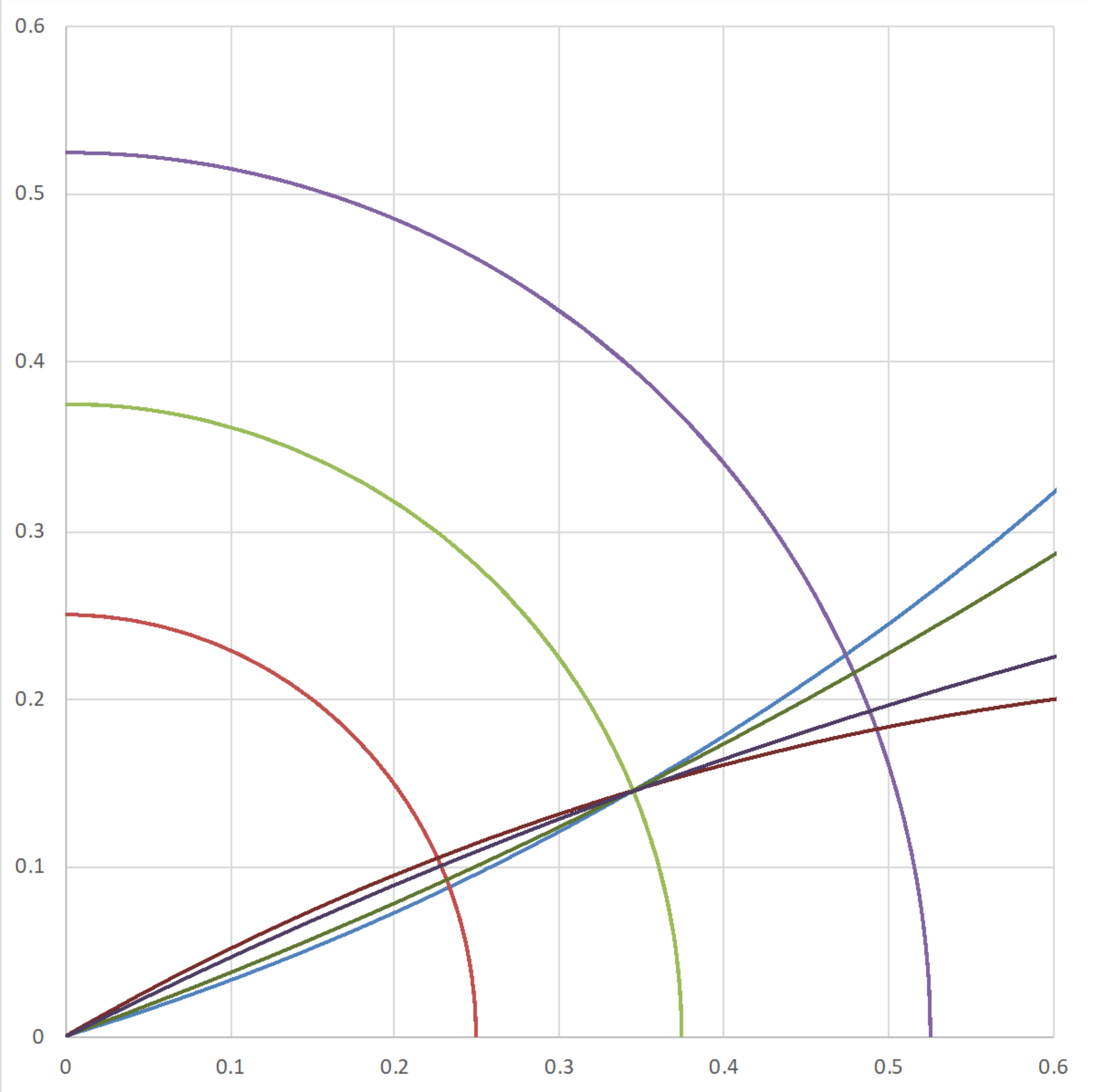


Mark Pesaresi et. al. ACES 2020

- At 200 pile-up, the tracker sees 15k-20k stubs => ~ 3000 stubs/layer.
- There are ~300 high Pt tracks (Pt > 2 GeV).
- About 90% stubs are not associated with any high Pt track. => more than 90% stub pairs are fake.
- Requesting additional constraints from 3 or more stubs (triplet) will pre-exclude fake pairs.

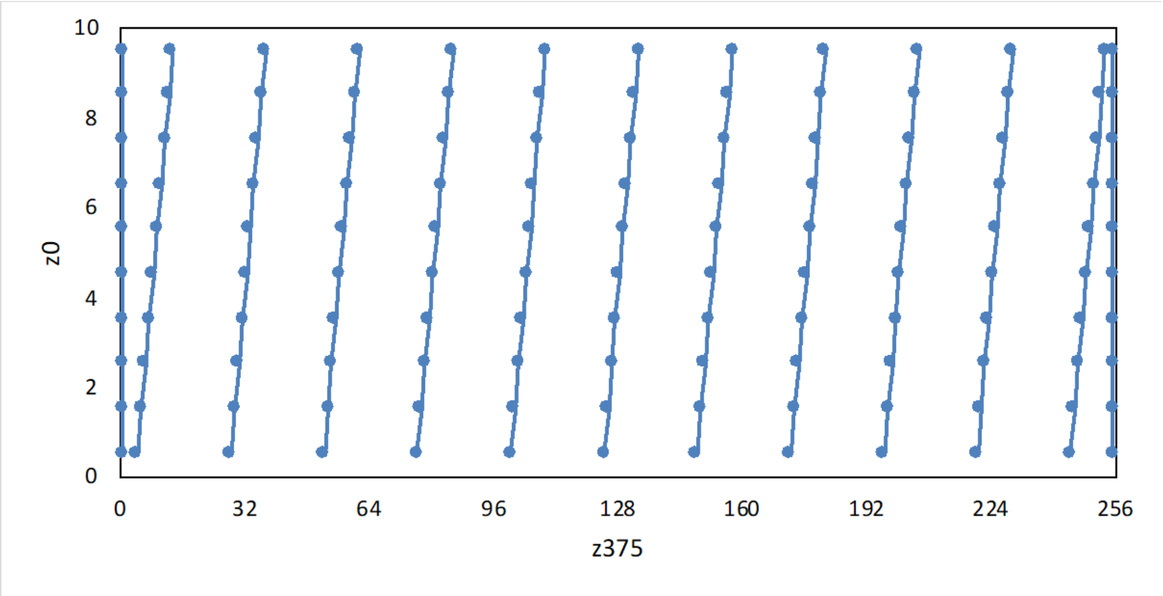
The Searching Range: Layer 2 to 1 and 3

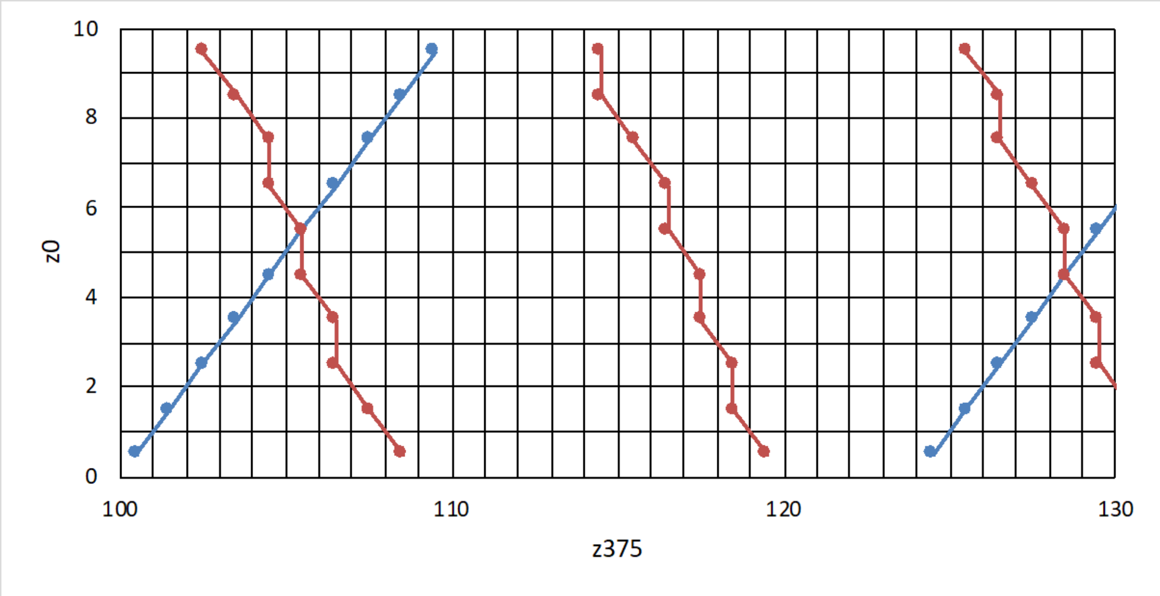
PT(GeV/c)	2.000		
B(T)	4.000		
R0(m)	1.667		
Z0(m)	0.200		
Detector Length(m)	2.400		
Number of stubs per layer	3000.000		
	Layer 1	Layer 2	Layer 3
R(m)	0.250	0.375	0.525
Phi(rad)	0.075	0.113	0.158
Searching range in phi (rad)	0.075		0.091
(deg.)	4.316		5.205
(mm)	18.834		47.691
Search range in Z (mm)	66.667		80.000
Total Layer Area (m^2)	3.770		7.917
Number of Hits in searching area	0.999		1.446

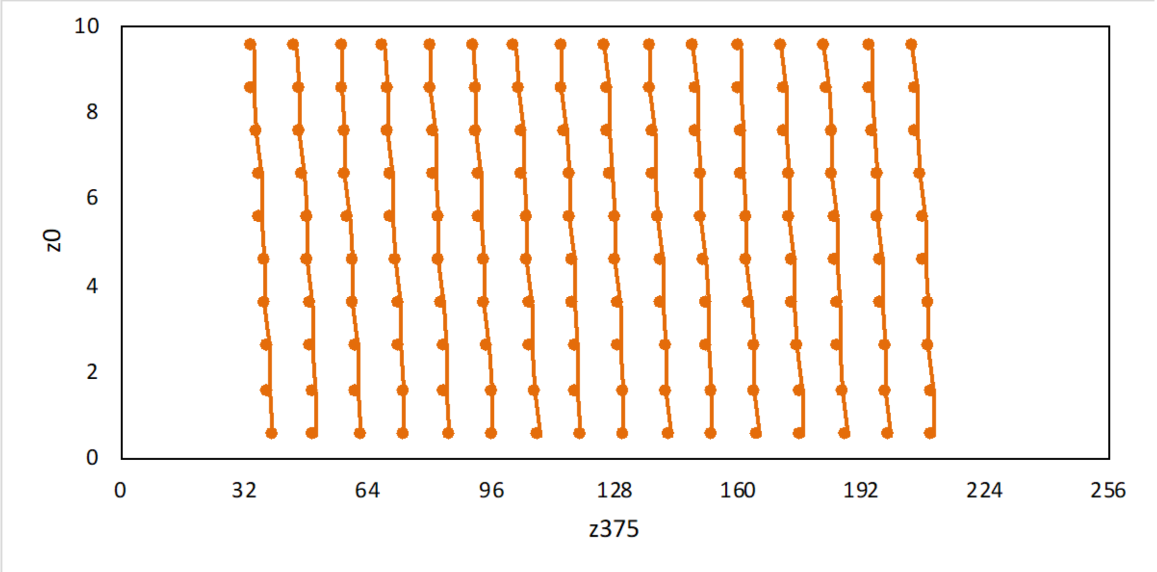


- For almost all 3000 stubs in Layer 2, on average a hit can be found in Layer 1 or Layer 3.
- About 90% of these pairs are not part of true tracks.
- When 3000 stubs/layer are sent to the stub pairing stage, there will be about 3000 pairs output to the tracking stage without significant reduction.

Hits in the Hugh Transform Cells

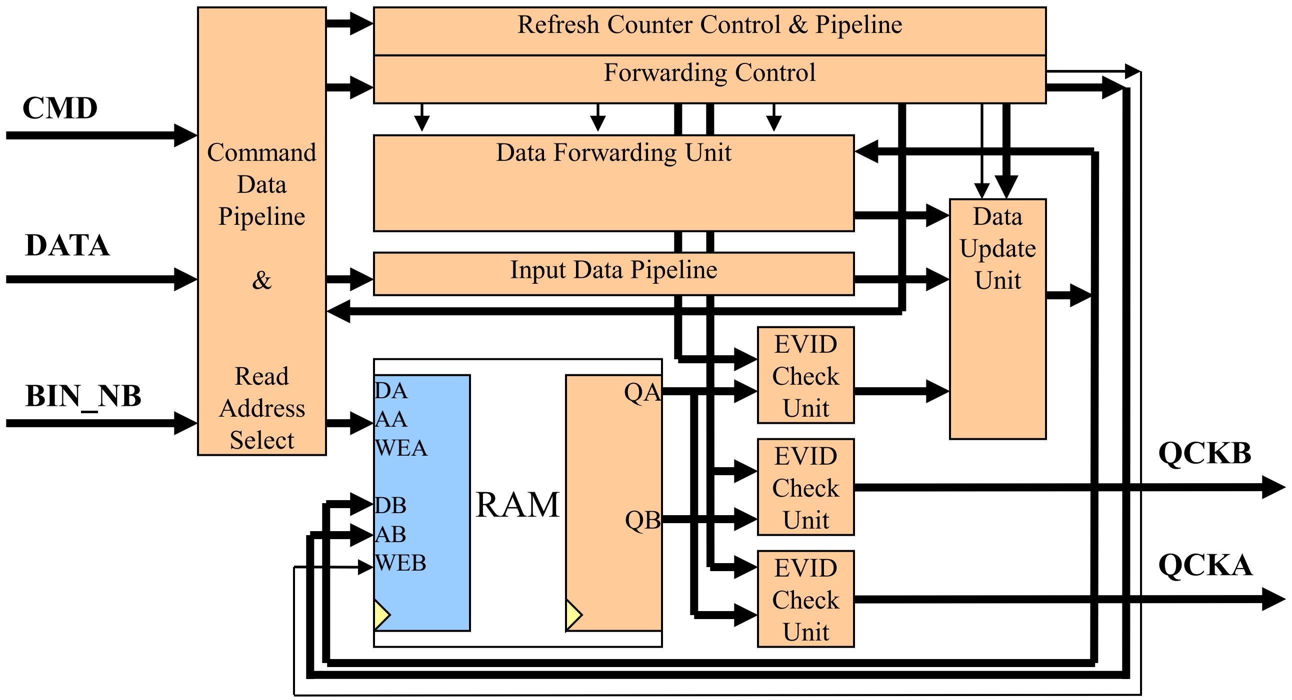






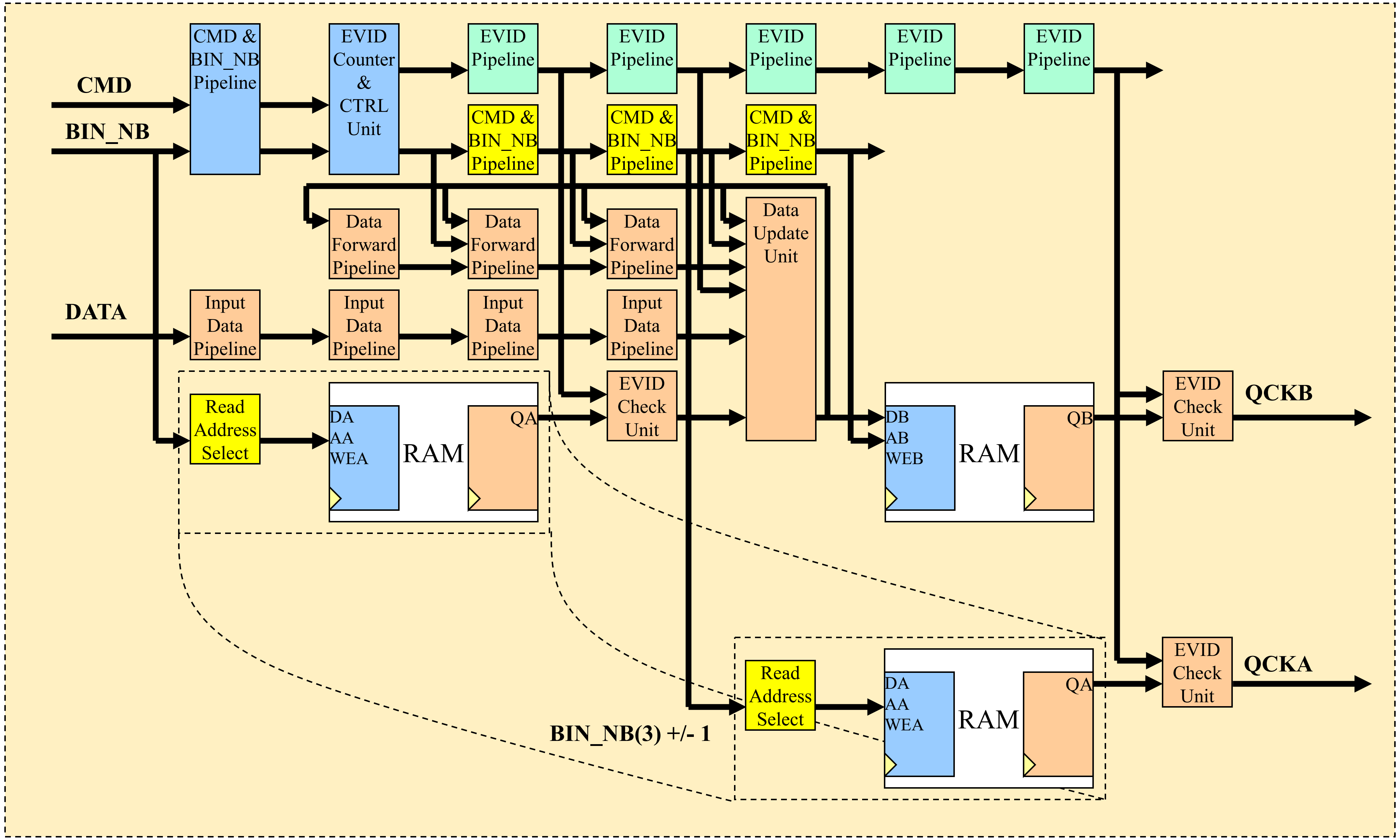
- A hit is written into 10 z0 RAM blocks.
- The hits in Layer 1 and Layer 3 have different angles.
- The z coordinate of Layer 2, z375 is used to address all RAM block simultaneously.
- If an intersection of Layer 1 and Layer 3 hits are met, it might be a potential track segment.

Register-Like Block RAM Block Diagram



- The

Detailed Block Diagram



- Boundary coverage is also supported during reading operations.