Progress on the VHDL implementation

Yutie Liang
April. 16 2013

Outline

1. Data format and structure of the tracking module
2. Some details to several modules
3. Estimation of computing time
4. Performance
5. Summary and outlook

Tracking algorithm in VHDL

24 bits fixed point number: 000000011010100101110010

first 8 bits for the integer part (1 sign bit), last 16 bits for the fractional part

A line of 96 bits represents one hit: pos_x, pos_y, pos_z, drift distance

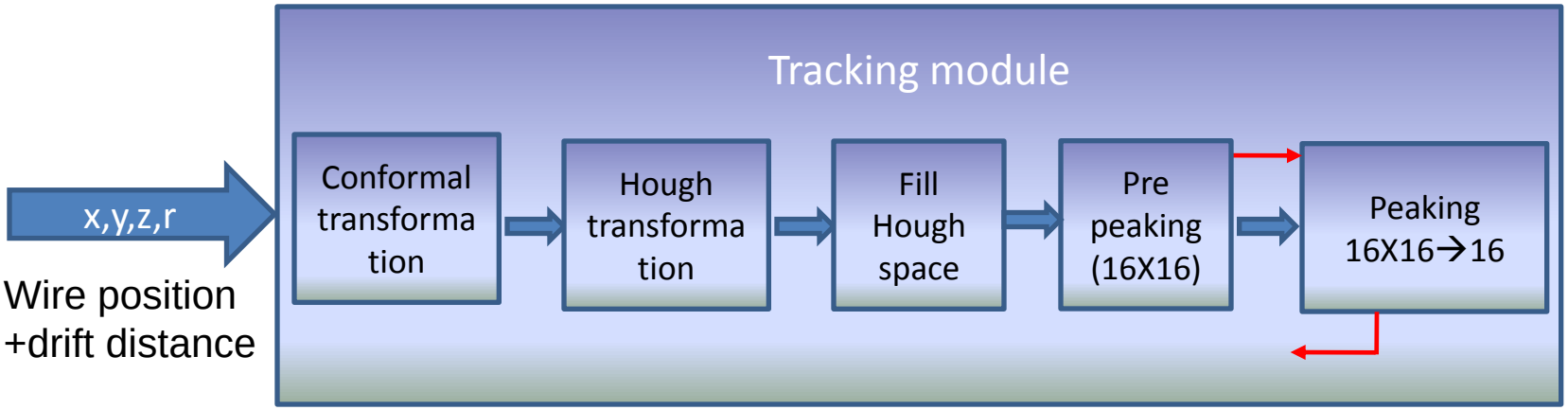
……

000000011010100101110010000000000000110011101101000000100000000000000000000000000000011001101011

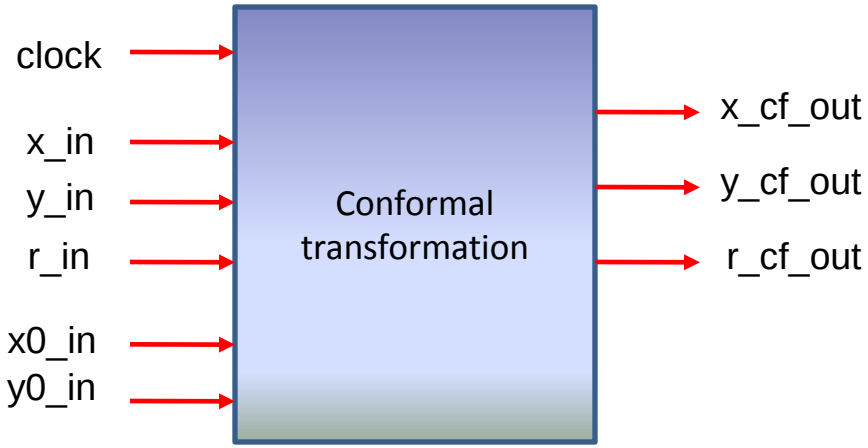
(1.6619, 0.0505, 2, 0.0250782) Unit (0.1m)

……

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000



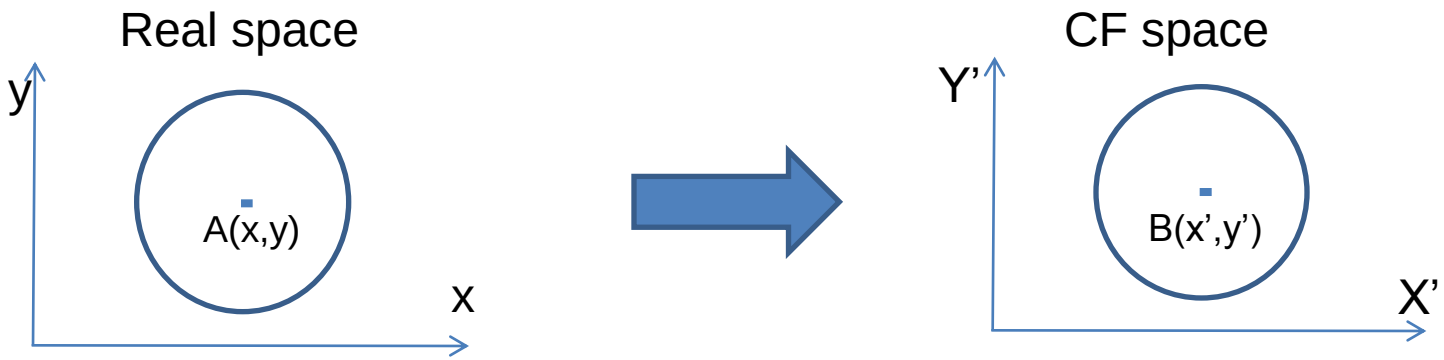
Conformal transformation module



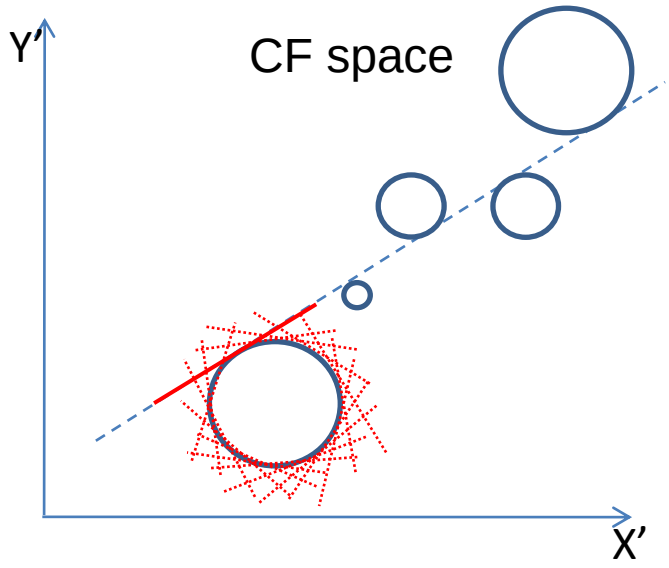
$R2 = (x\_in-x0\_in)^2 + (y\_in-y0\_in)^2-r\_in^2$

$x\_cf\_out = (x\_in-x0\_in)/R2 + x0\_in$
$y\_cf\_out = (y\_in-y0\_in)/R2 + y0\_in$
$r\_cf\_out = r\_in /R2$

1 clock cycle

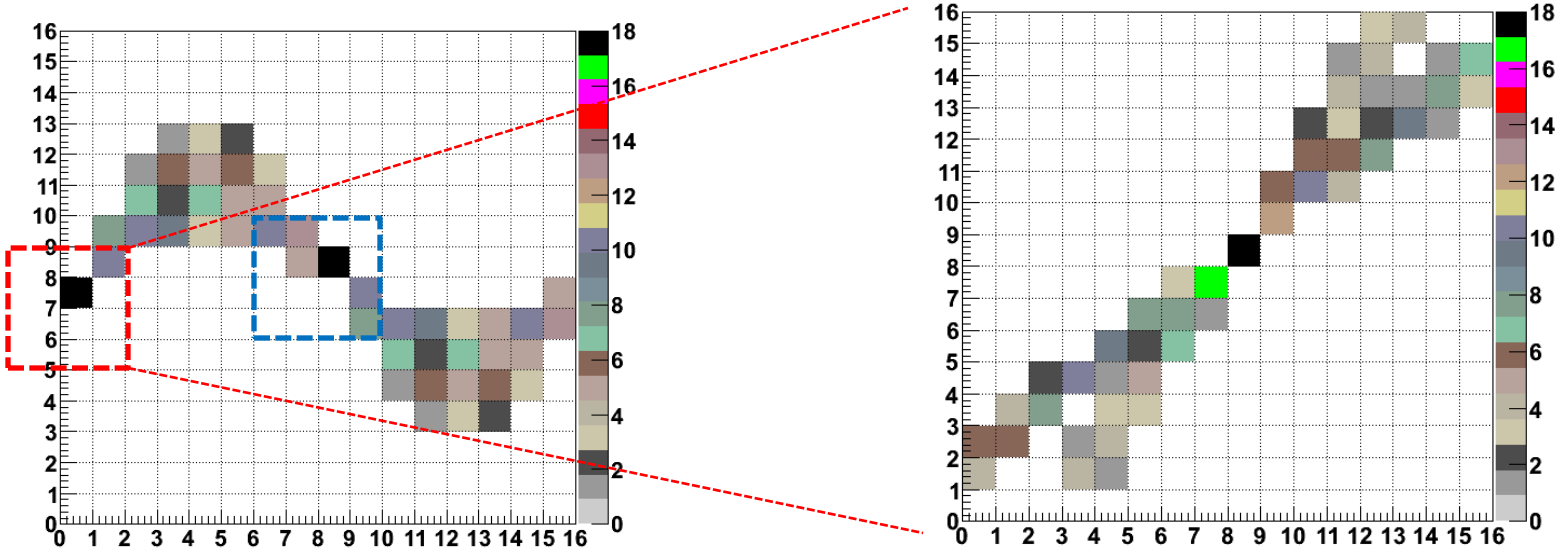


Adaptive Hough transformation – large sin array still needed

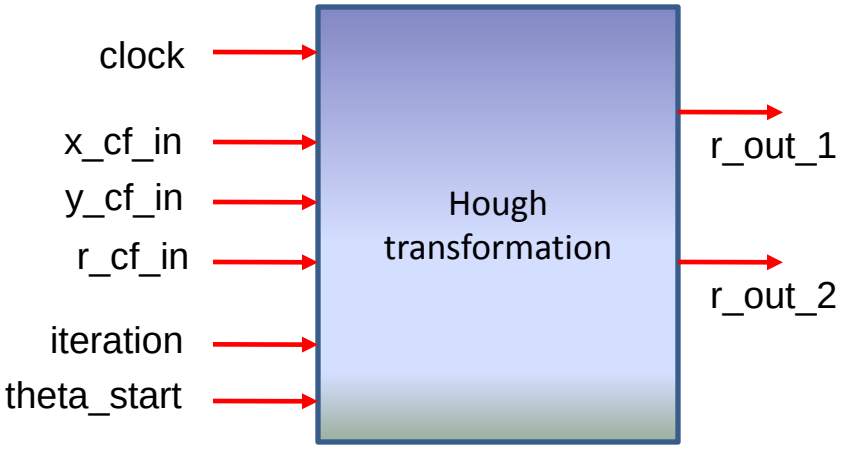


$$r = x\cos(\theta) + y\sin(\theta)$$

Expect to run up to iteration 4: 4096X4096. So, the θ (- 0.5π -0.5π) need to be divided into 4096 bins



Hough transformation module -- sin array in look-up table




```
sin_array(0)       <= "11111111000000000000000";  -- -1
 sin_array(1)      <= "11111111000000000000001";  -- -1
 …
sin_array(2046) <= "11111111111111110011100";  -- -0.00153398
 sin_array(2047) <= "11111111111111111001110";  -- -0.00076699
sin_array(2048)  <= "00000000000000000000000";  -- 0
 sin_array(2049)  <= "00000000000000000110010";  -- 0.00076699
 …
sin_array(4094)  <= "00000000111111111111111";  -- 0.999999
 sin_array(4095) <= "00000000111111111111111";  -- 1
```

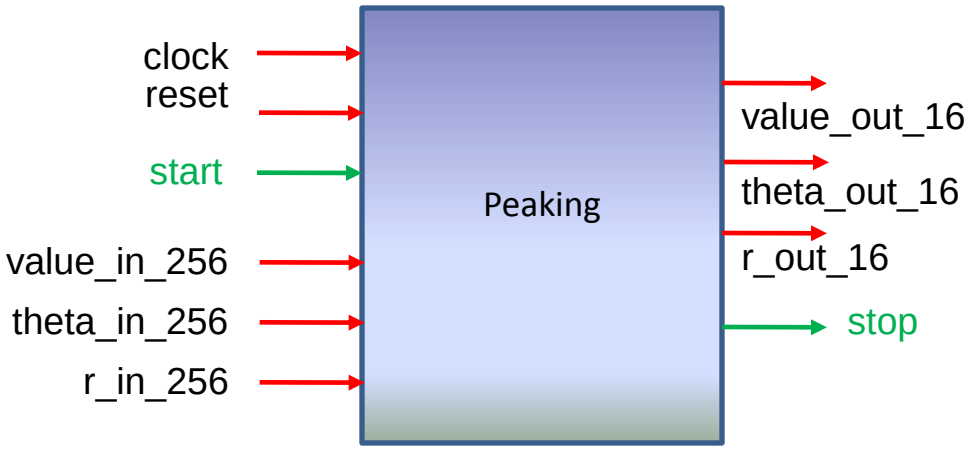
Three look-up tables of sin_array are compared.

1) 4096 bins from 0 to π, 24 bits
2) 2048 bins from 0 to π/2, 24 bits
3) 2048 bins from 0 to π/2, 17 bits

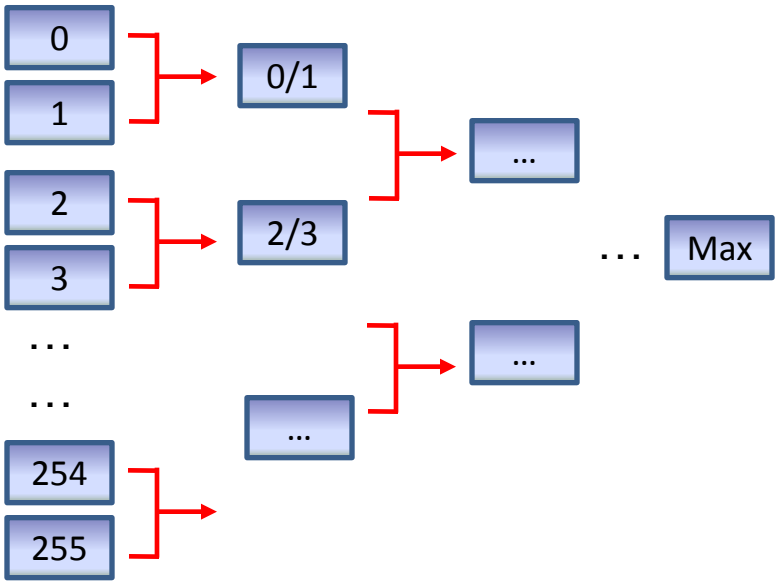
Total memory consumption using ISIM

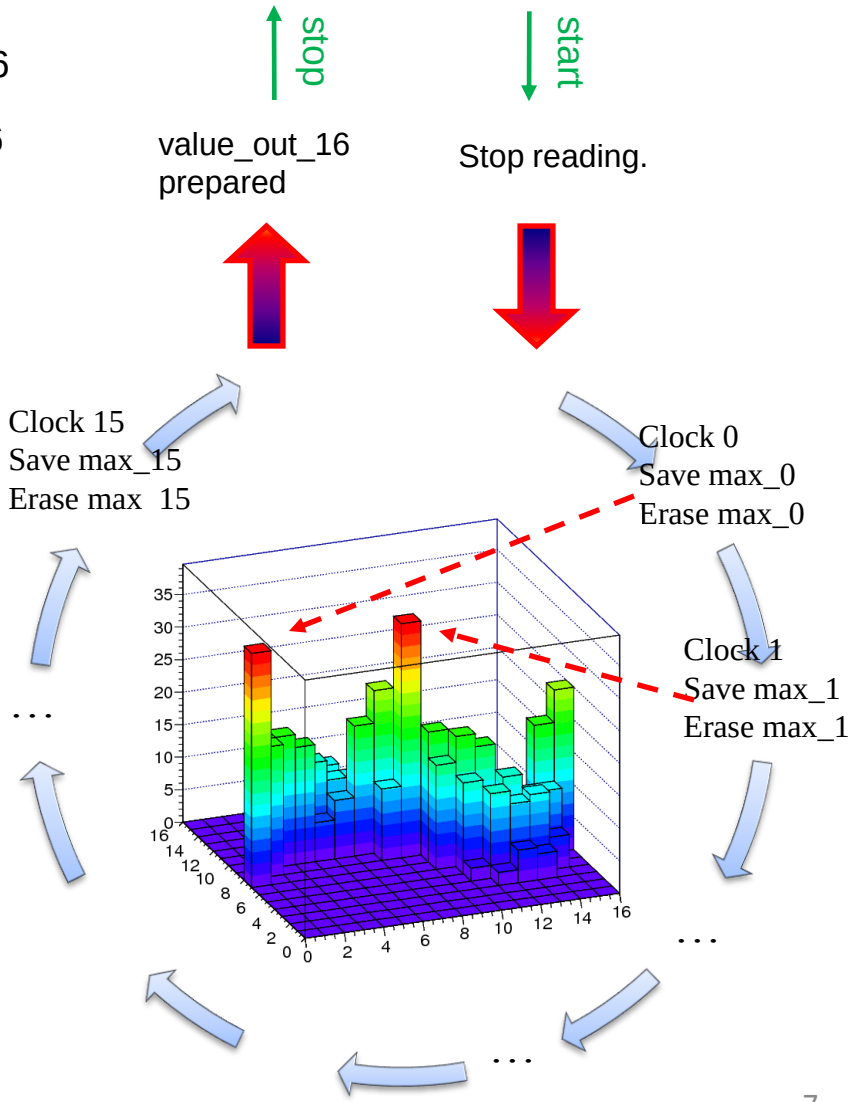
1.9 GB
1.0 GB
0.87GB

Peaking module

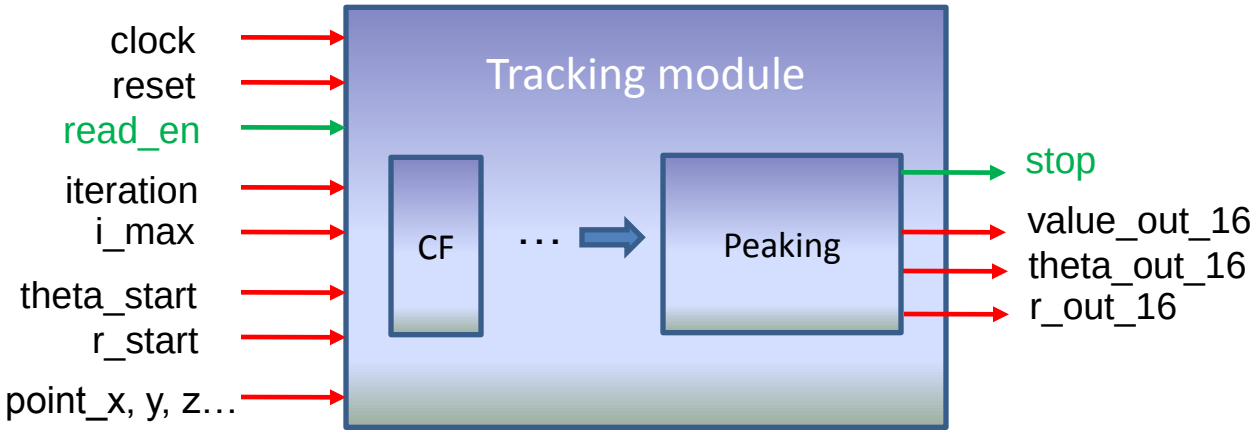


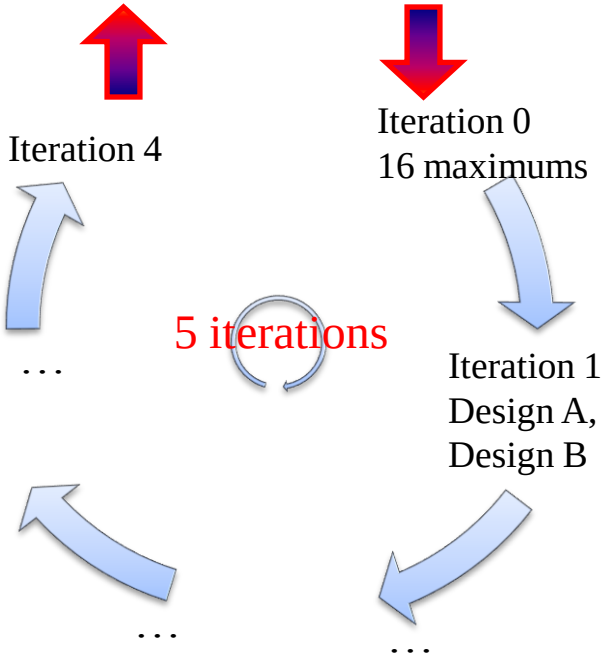
1 clock cycle to locate the maximum



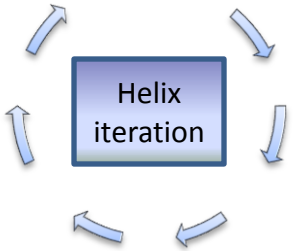


Adaptive design of tracking module

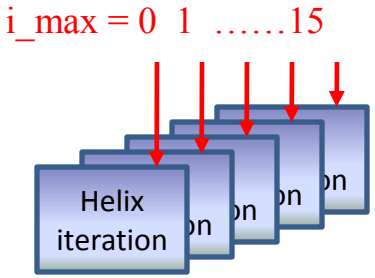




Design A:
Only one module, change i_max every loop

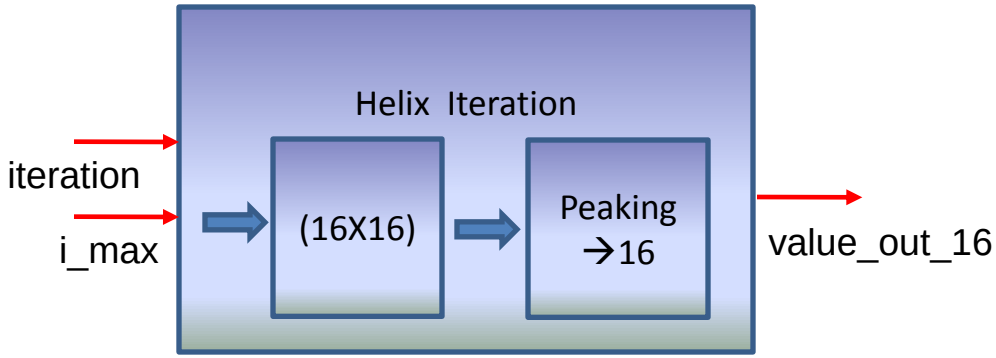


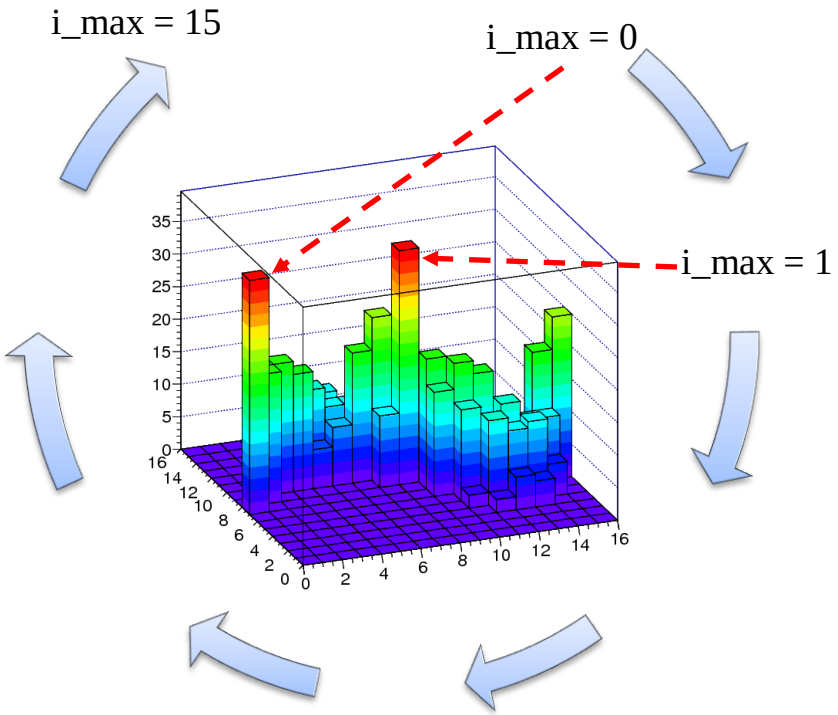
Design B:
16 modules, paralized



Adaptive design of tracking module

Take Design A as example:





Iteration number is fixed. I_max changes from 0 to 15.

Use an array with size of 256 to collect value_out_16 at each i_max.

Then transmit this array to “peaking ” module, to select 16 maximums.

These 16 maximums will be transferred to the next iteration as seeds

Estimation of computing time for two designs

For an event with N hits, the expected time of tracking is:
For design A: N_time = (N+16) + [(N+16)*16 + 16] * N_iteration

If N_iteration = 4, and N = 50 , then N_time = 4354 clock cycles. --> 87.1 cc/hit
If N_iteration = 4, and N = 100 , then N_time = 7604 clock cycles. --> 76.0 cc/hit
If N_iteration = 4, and N = 200 , then N_time = 14104 clock cycles. --> 70.5 cc/hit

For design B: N_time = (N+16) + [(N+16) + 16] * N_iteration

If N_iteration = 4, and N = 50 , then N_time = 394 clock cycles. --> 7.9 cc/hit
If N_iteration = 4, and N = 100 , then N_time = 644 clock cycles. --> 6.4 cc/hit
If N_iteration = 4, and N = 200 , then N_time = 1144 clock cycles. --> 5.7 cc/hit

PANDA@20MHz. Very rough estimation
$2*10^7$ event/second * 3? track/event * ~20 hit/track * 3? overlap factor → ~ 4 hit/second

Device utilization summary of tracking module

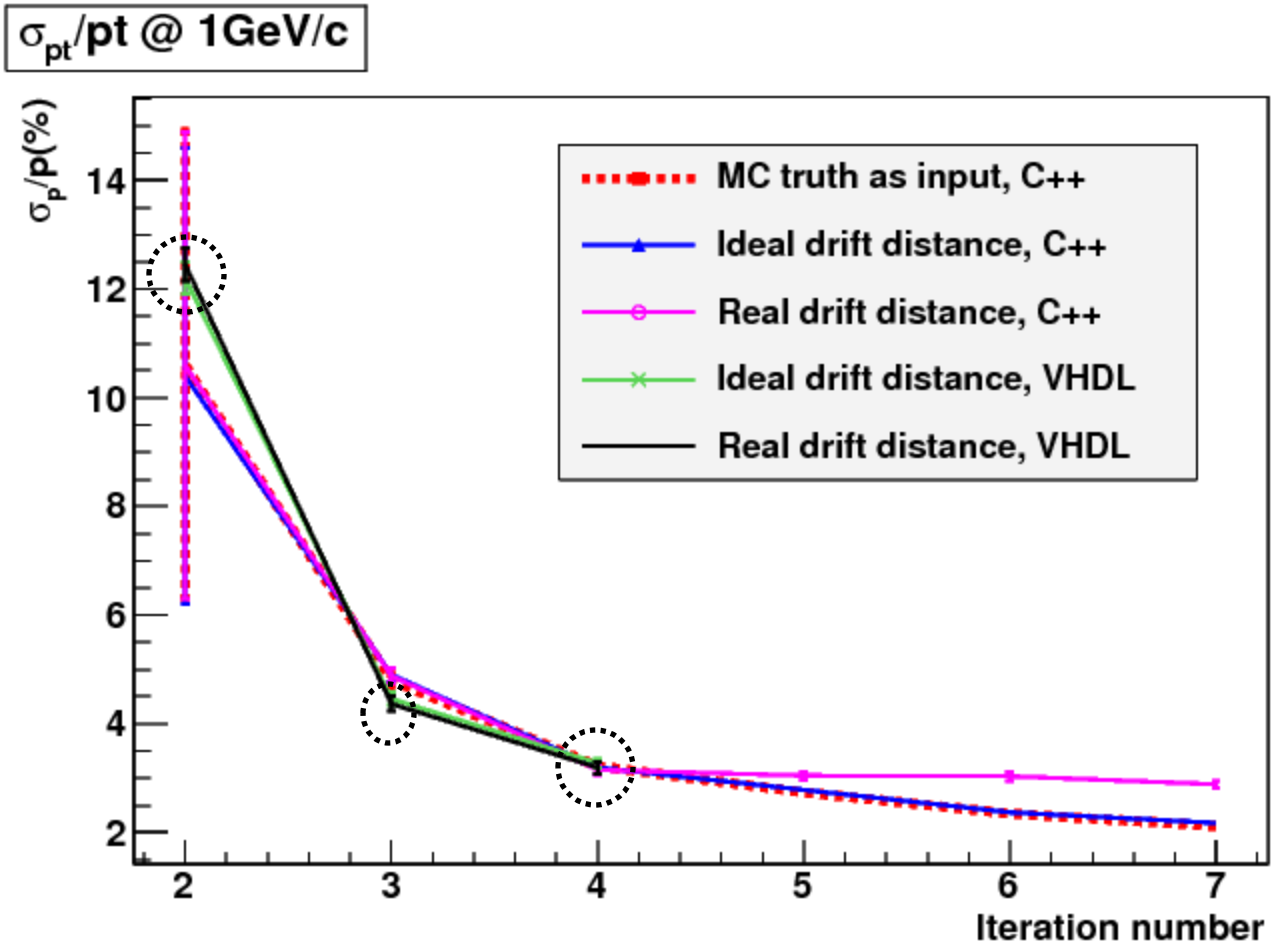
Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	3089	25280	12%
Number of Slice Flip Flops	4520	50560	8%
Number of 4 input LUTs	4535	50560	8%
Number of bonded IOBs	38	352	10%
Number of FIFO16/RAMB16s	25	232	10%
Number of GCLKs	6	32	18%
Number of DCM_ADVs	2	12	16%

Two parts which need large amount of resource:

1: The big array of sin function in Hough transformation.

2: The peaking module.

Pt resolution and comparison with C++ simulation



Summary and Outlook

1. A preliminary version is running.

   Resource, computing time, latency, …

   Pt resolution …

2. Lots need to do:

   1) Optimize the tracking module, reduce the size

   2) The structure of tracking module need to be changed.

      The large sin array and peaking module need to separated.

   3) Some minor differences of VHDL design from C++.

A:

If N_iteration = 3, and N = 50 , we have N_time = 3282 clock cycles. --> 65.6 cc/hit

If N_iteration = 3, and N = 100 , we have N_time = 5732 clock cycles. --> 57.3 cc/hit

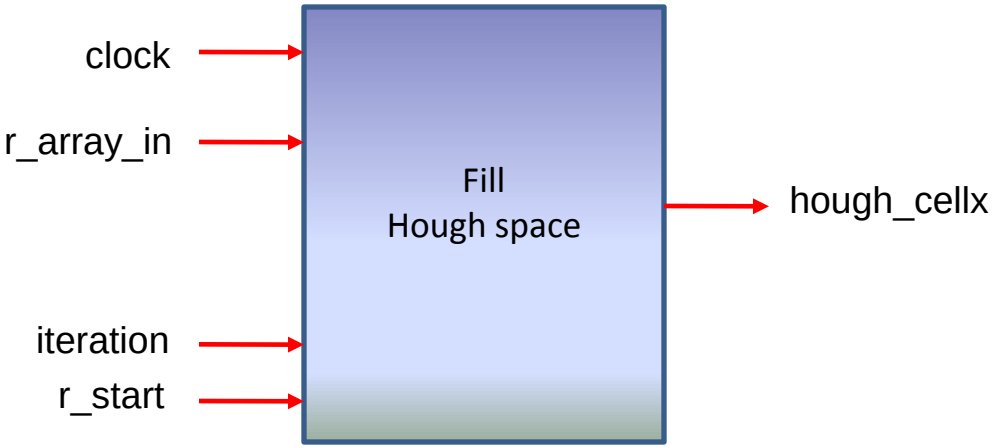
If N_iteration = 3, and N = 200 , we have N_time = 10632 clock cycles. --> 53.2 cc/hit

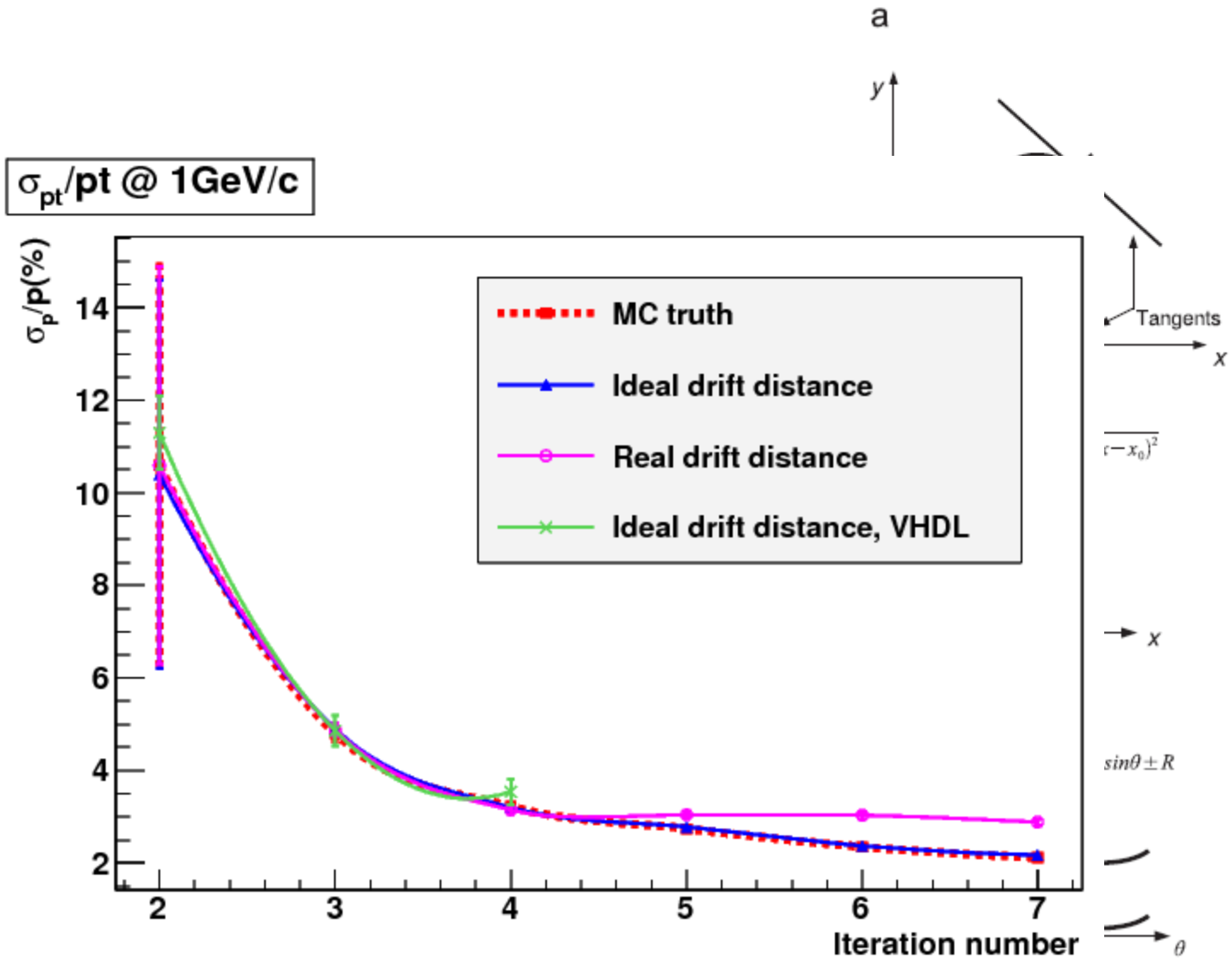
B:

If N_iteration = 3, and N = 50 , we have N_time = 312 clock cycles. --> 6.2 cc/hit

If N_iteration = 3, and N = 100 , we have N_time = 512 clock cycles. --> 5.1 cc/hit

If N_iteration = 3, and N = 200 , we have N_time = 912 clock cycles. --> 4.6 cc/hit


clock
r_array_in
Fill
Hough space
hough_cellx
iteration
r_start


σpt/pt @ 1GeV/c
σp/p(%)
14
12
10
8
6
4
2
2
3
4
5
6
7
Iteration number
MC truth
Ideal drift distance
Real drift distance
Ideal drift distance, VHDL
a
y
Tangents
x
x
θ

```
  max1_step_8_i: for i in 0 to 7 generate
        max1_step_8_j: for j in 0 to 15 generate
                  max1_Comp8: hough_compare_1_output Port map (
                                                A_value  => value_max1_in((i*16+j)),
                                                A_theta  => conv_std_logic_vector(i, 10),
                                                A_r      => conv_std_logic_vector(j, 10),
                                                B_value  => value_max1_in((i*16+j)+128),
                                                B_theta  => conv_std_logic_vector(i+8, 10),
                                                B_r      => conv_std_logic_vector(j, 10),
                                                Qg_value => value_max1_step8((i*16+j)),
                                                Qg_theta => theta_max1_step8((i*16+j)),
                                                Qg_r     => r_max1_step8((i*16+j)));
        end generate;
end generate;
```