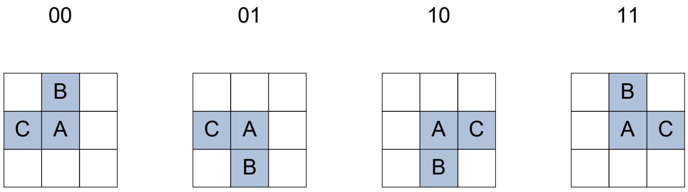
Cluster_r

Orientations



hitmerge outputs

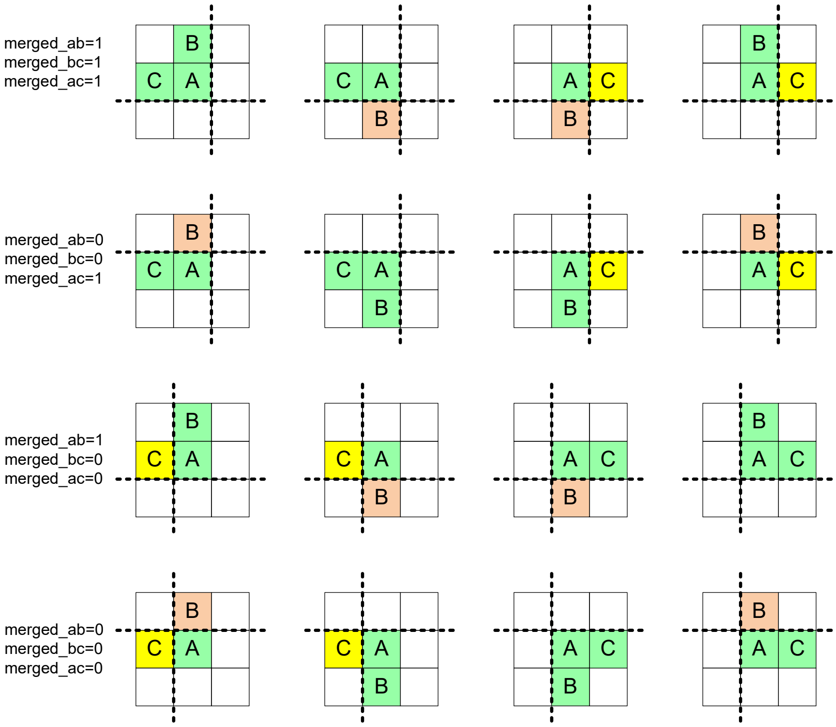
dout_a 🟩

dout_b 🟧

dout_c 🟨

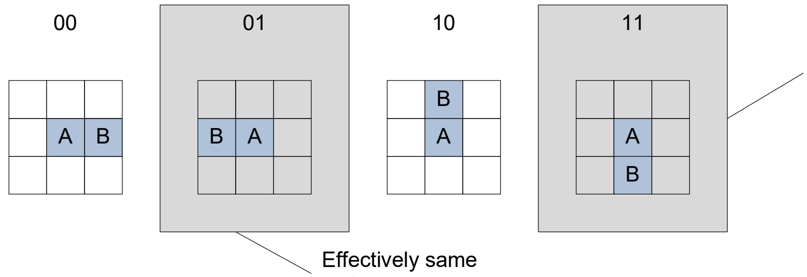
All output can be done using merged_ab and merged_bc

Possible qrow block boundaries



Cluster_2x1

Orientations



Effectively same as ‘10’

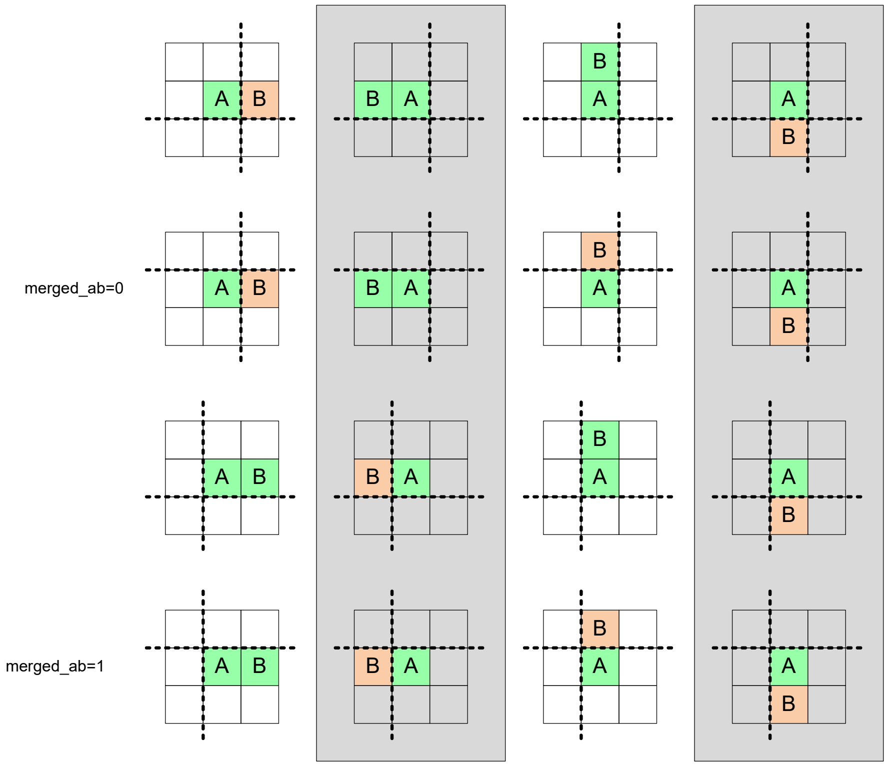
Effectively same as ‘00’

hitmerge outputs

dout_a 🟩

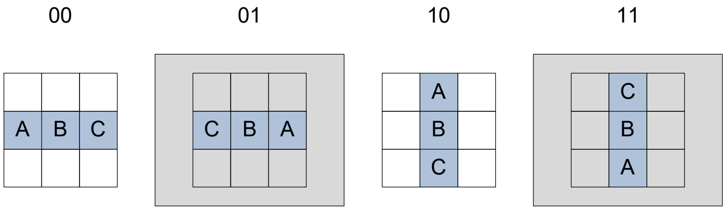
dout_b 🟧

Possible qrow block boundaries



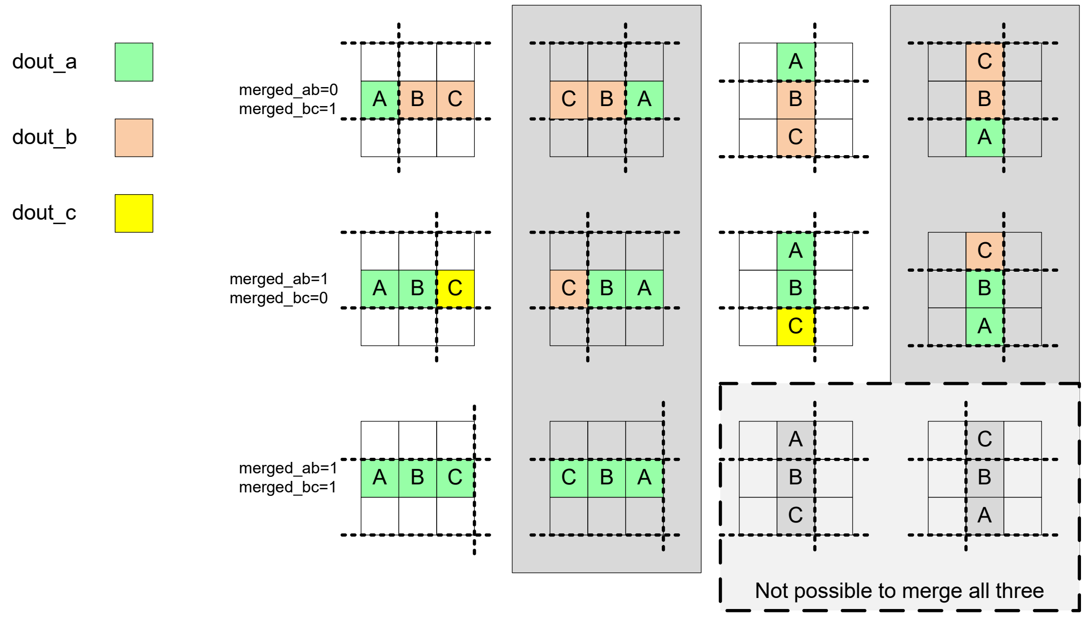
Cluster_3x1

Orientations



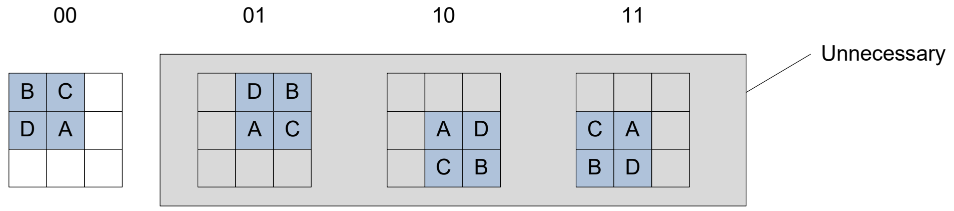
hitmerge outputs

Possible qrow block boundaries



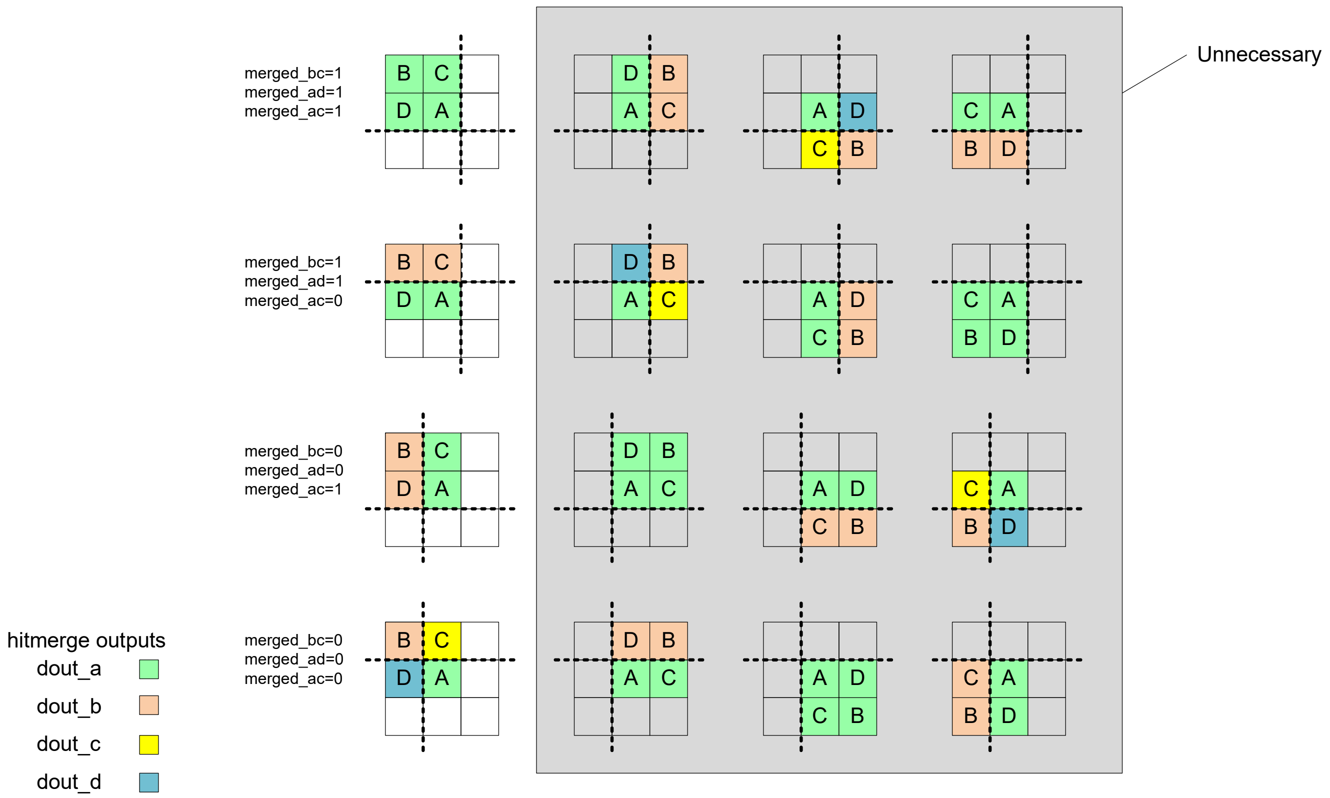
Cluster_2x2

Orientations



```
{xB, yB, xC, yC, xD, yD} = {x-1 , y+1 , x     , y+1 , x-1  , y     };  // in comment
{xB, yB, xC, yC, xD, yD} = {x+1, y+1 , x+1 , y     , x     , y+1  };  // rotated 90 deg. clockwise
{xB, yB, xC, yC, xD, yD} = {x+1, y-1  , x     , y-1  , x+1 ,  y     };  // rotated 180 deg. cw
{xB, yB, xC, yC, xD, yD} = {x-1 , y-1  , x-1  , y     , x     , y-1   };  // rotated 270 deg. cw
```

Possible qrow block boundaries



Cluster_F

Orientations

000

	B	C
D	A	
	E	

001

	D	
E	A	B
		C

010

	E	
	A	D
C	B	

011

C		
B	A	E
	D	

100

C	B	
	A	D
	E	

101

		C
E	A	B
	D	

110

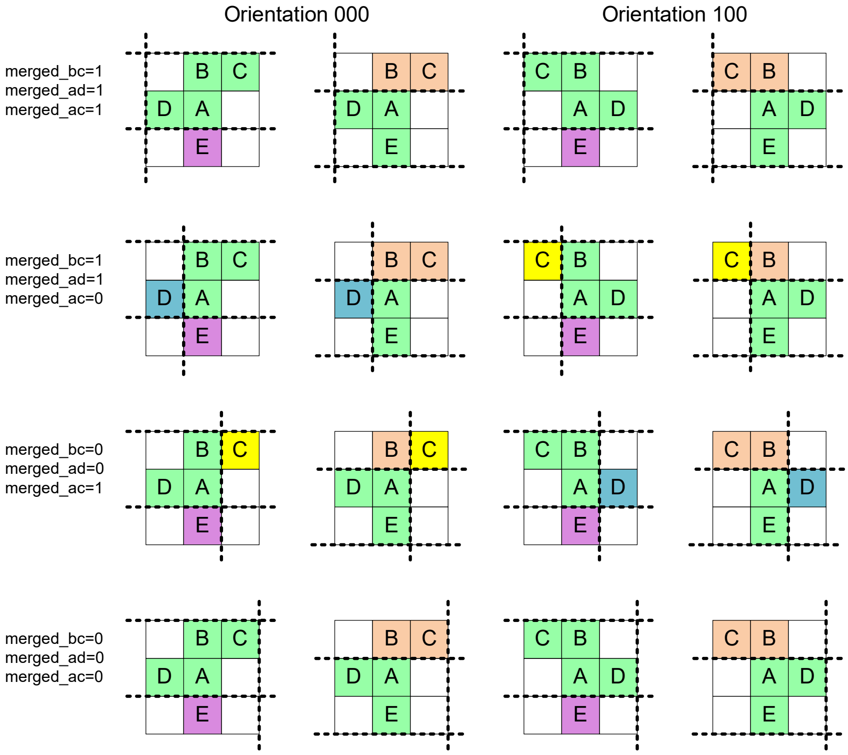
	E	
D	A	
	B	C

111

	D	
B	A	E
C		

```
3'd0: {xB, yB, xC, yC, xD, yD, xE, yE} = {x, y+9'd1, x+9'd1, y+9'd1, x-9'd1, y, x, y-9'd1}; // in comment
3'd1: {xB, yB, xC, yC, xD, yD, xE, yE} = {x+9'd1, y, x+9'd1, y-9'd1, x, y+9'd1, x-9'd1, y}; // rotated 90 deg. clockwise
3'd2: {xB, yB, xC, yC, xD, yD, xE, yE} = {x, y-9'd1, x-9'd1, y-9'd1, x+9'd1, y, x, y+9'd1}; // rotated 180 deg. cw
3'd3: {xB, yB, xC, yC, xD, yD, xE, yE} = {x-9'd1, y, x-9'd1, y+9'd1, x, y-9'd1, x+9'd1, y}; // rotated 270 deg. cw
3'd4: {xB, yB, xC, yC, xD, yD, xE, yE} = {x, y+9'd1, x-9'd1, y+9'd1, x+9'd1, y, x, y-9'd1}; // mirrored
3'd5: {xB, yB, xC, yC, xD, yD, xE, yE} = {x+9'd1, y, x+9'd1, y+9'd1, x, y-9'd1, x-9'd1, y}; // mirrored and rotated 90 deg. cw
3'd6: {xB, yB, xC, yC, xD, yD, xE, yE} = {x, y-9'd1, x+9'd1, y-9'd1, x-9'd1, y, x, y+9'd1}; // mirrored and rotated 180 deg. cw
3'd7: {xB, yB, xC, yC, xD, yD, xE, yE} = {x-9'd1, y, x-9'd1, y-9'd1, x, y+9'd1, x+9'd1, y}; // mirrored and rotated 270 deg. cw
```

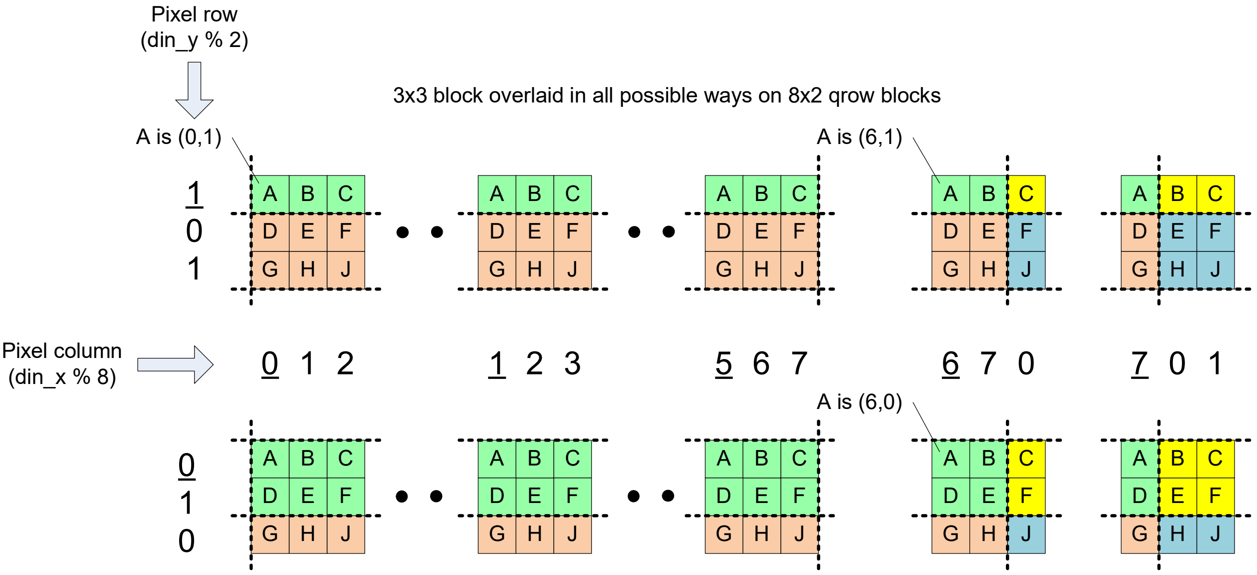
Possible qrow block boundaries



hitmerge outputs

dout_a	🟩
dout_b	🟧
dout_c	🟨
dout_d	🟦
dout_e	🟪

Cluster_3x3



Outputs

First 🟩

Second 🟧

Third 🟨

Fourth 🟦

Input location of pixel 'A': din_x[8:0] and din_y[8:0] and a 9-bit pattern patt[A B C D E F G H J] with a '1' representing an over-threshold bit.

Use din_y[0] (din_y%2) and din_x[2:0] (din_x%8) to find the corresponding diagram above.
Calculate the ccol and qrow values for either two or four sequential outputs.

ccol0 = din_x[8:3] + 1 (ccol from 1 to TBD)
qrow0 = din_y[8:1]

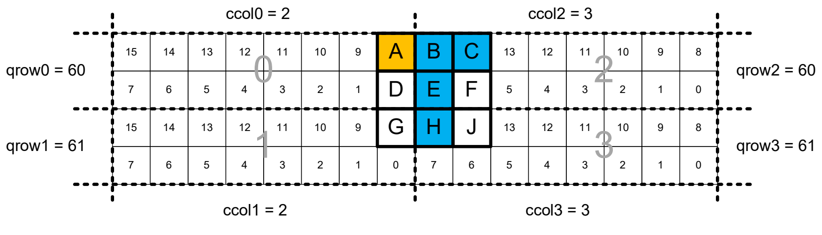
ccol1 = din_x[8:3] + 1
qrow1 = din_y[8:1] + 1

ccol2 = din_x[8:3] + 2
qrow2 = din_y[8:1]

ccol3 = din_x[8:3] + 2
qrow3 = din_y[8:1] + 1

The hitmap for each of the two, or four, outputs is made by OR'ing and shifting the corresponding bits. See the example below.

Example : Cluster_T



There are 50 core columns and 48 core rows = 48*4 = 192 qrows

din_x[8:0] = 24, din_y = 120, patt = 111 010 010
din_x[3:0] = 7, din_y[0] = 0

ccol0 = 2, qrow0 = 60
hitmap0 = (A << 8) + D = 0000 0001 0000 0000

ccol1 = 2, qrow1 = 61
hitmap1 = G << 8 = 0000 0000 0000 0000 (no output)

ccol2 = 3, qrow2 = 60
hitmap2 = (B<<15) + (C<<14) + (E<<7) + (F<<6)
= 1100 0000 1000 0000

ccol3 = 3, qrow3 = 61
hitmap3 = (H << 15) + (J << 14) = 1000 0000 0000 0000
is_neighbor = 1

An improvement over purely random X,Y generation for the location of pixel A is to increase Y by a random amount while keeping X[8:3] constant and randomly varying X[2:0] and when it exceeds the number of rows, increasing X by 8*N with a minimum of 16. This would cause a consecutive series of outputs to be in ccol N and N+1. The hits in ccol N+1 could be buffered until ccol N is completed. These would be then be output while outputs in ccol N+2 and N+3 are generated. This way no duplicate qrow blocks would be produced and is_last and is_neighbor can be set appropriately. Bloom filters would not be needed since qrow block outputs happen in the same order as the chip (top-to-bottom in a column and left to right columns) and there would be no outputs

How would horizontal and vertical lines be integrated into this?

Vertical lines would be fairly straightforward because a series of consecutive blocks with the same ccol and incrementing qrow with a fixed 2x1 vertical hitmap would be generated. Clusters would be re-started in the same ccol as the vertical line in a row after the line is completed (or the next ccol if the line ends at the bottom of the column.

A horizontal line could be made by incrementing the ccol but this would prevent clusters from being placed in any cell in the ccols spanned by the line. Also, is_last would have to be set on every output hit.