DigitLabwork#5

Comparison of two images by Hamming distance in neighborhoods

[1] Training goals

[2] Preparations

[3] Lab-task instruction

[4] Observations/Evaluation

= =

[1] Training goals

Through the lab-work, class members are expected to get acquainted with the following matters.

a)) comprehension of the workflow of using sequential module as underlying units in the design of a system for specific application:

** clarifying operating steps in the algorithmic process involved in the application,

** conceiving the plausible circuit structure and the timing steps associated with the activation/deactivation of constituent modules,

** Verilog simulation of component modules and trouble shooting,

** Verilog simulation of the integrated system and trouble shooting;

b)) the use of Verilog for logic circuit description and simulation, including

** circuit module building up in Verilog,

** setting up of the test data;

c)) the interactive commands required in operating the Verilog-code development system:

** compilation for a syntax error-free Verilog description and test data set;

** simulation of the Verilog-coded circuit module;

[** synthesis of the Verilog-coded circuit module].

[2] Preparation

Every class member should get oneself prepared by studying the following materials prior to attending the lab-work sessions.

a)) operation of shift registers.

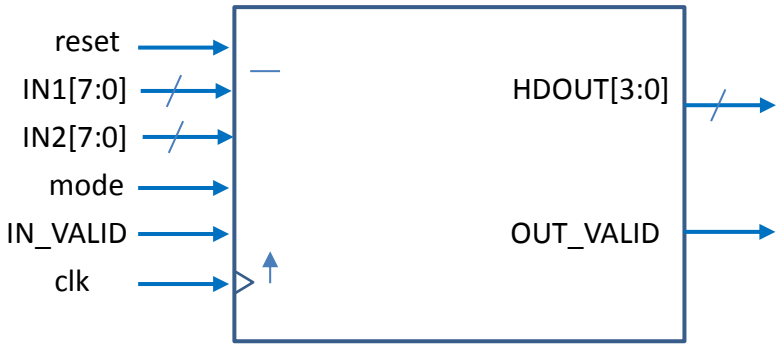
b)) skills of generating required outputs during state transitions in the operation of a sequential circuit.

[3] Lab-task instruction

In digitlabwork#5, 1 task are assigned to every class members as given below.

[TASK1]

(1) Write a Verilog code for a circuitry capable of image comparison in compliance with the requirements given below.



(2) Put the codes under simulation; observe and interpret the waveforms of output signals

(3) Try drawing the circuit diagram of the circuit in question, using 1bit D-reg, logic gates and components one deems as necessary.

a)) coding specifications:

** write behavior-level descriptions for the circuitry in question.

** the module has as IN1, IN2, reset, clk, mode, IN_VALIS as inputs, and as HDOUT, OUT_VALID as outputs.

** correct response to TA's testing data at simulation.

** refer to **[4] Useful tips** for

0)) system framework and process timing flow;

1)) intra-image comparison:
comparison of pixel values in the neighborhood around a specific pixel in an image, i.e. cmp(PIX_{nbr_center}, PIX_{nbr}) in terms of GE or LT;

2)) module for recording intra-image neighborhood comparison for each pixel in an image;

3)) inter-image comparison:
comparison of two neighborhoods located at corresponding locations in two images by hamming distance evaluation;

4)) circuit modules for flag-setting/resetting

5)) circuit module for receiving a block of pixel values coming in serial order;

6)) mechanism for doing hamming distance computation and sending output sequence.

b)) circuit specifications:

** I/O layout: inputs IN1[7:0], IN2[7:0],
reset, clk, mode, IN_VALID;
outputs HDOUT[3:0],
OUT_VALID;
synchronization: all I/O's being clk-synchronized at rising edge.

** Function requirements:

[reset]

reset	clk	OUT_VALID	HDOUT	
1	^	0	Z	
0	x	OUT_VALID	HDOUT	

General behavior of the circuit:

[1] in the operation of the circuit after power-on, RESET signal occurs only once and is clock-synchronized;

[2] after receiving RESET, the 1st input indicated by IN_VALID may occur at any moment by which handling of input sequences at IN1 and IN2 should be started;

[3] internal process for Hamming distance calculation over the two image blocks must be completed within 200 clocks after the activation of IN_VALID;

[4] the 2nd input to the circuit may arrive anytime between 1 to 8 clocks after the deactivation the 1st OUT_VALID;

[5] all subsequent input-output operating cycles proceed as aforementioned in [2], [3] and [4], without any more RESET intervening, until power-off.

[6] internal count-limit for IN1/2 and HDOUT counting

[input sequence]

clk	IN_VALID	mode	IN1	IN2	mode_cnt*	notes
^	0	x	x	x	0	
$^{1-4}$	1	0	$IN1^{1-4}$	$IN2^{1-4}$	4	[1], [2], [3]
$^{1-16}$	1	1	$IN1^{1-16}$	$IN2^{1-16}$	16	[4], [5]

[1] $^{1-14}$ and $^{1-16}$ imply, respectively the first 4 and 16 successive clocks after IN_VALID becomes HIGH.

[2] mode only valid during the 1st clock after IN_VALID gets HIGH.

[3] $IN1^{1-4}$ implies the values of 4 pixels of image1 sent to the circuit after IN_VALID gets HIGH when mode==0; so as $IN2^{1-4}$ of image2.

[4] $IN1^{1-16}$ implies the values of 16 pixels of image1 sent to the circuit

after IN_VALID gets HIGH when mode==1; so as $IN2^{1-16}$ of image2.

[5] mode_cnt* as HDOUT count-limit and as IN1/2 count-limit

[output sequemce]

clk	OUT_VALID	HDOUT	notes
^	0	Z	[1]
$^{1-4}$	1	$HDOUT^{1-4}$	[2]
$^{1-16}$	1	$HDOUT^{1-16}$	[3]

[1] OUT_VALID should not be remaining LOW more than 200 clocks after IN_VLAID becomes HIGH.

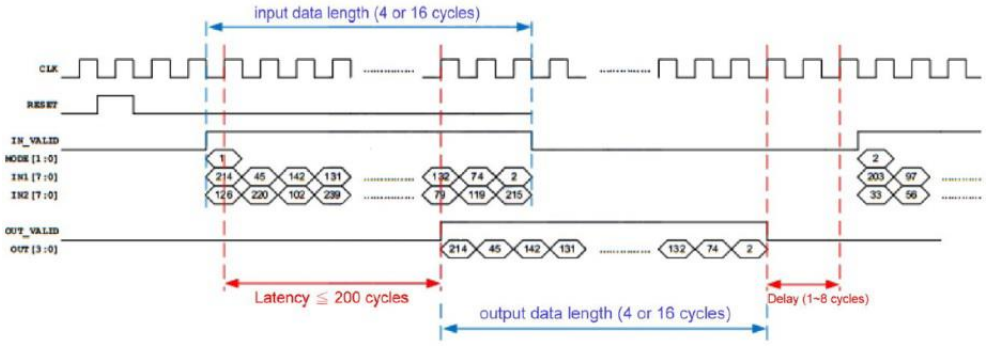
[2] $^{1-14}$ and $^{1-16}$ imply, respectively the first 4 and 16 successive clocks after OUT_VALID becomes HIGH.

[3] $HDOUT^{1-16}$ implies the values of 16 hamming-distance over the 4x4 neighborhoods of image1 and image2 while when mode==1; whereas $HDOUT^{1-4}$ when mood==0.

** Timing: no delay(s) to be considered in this task

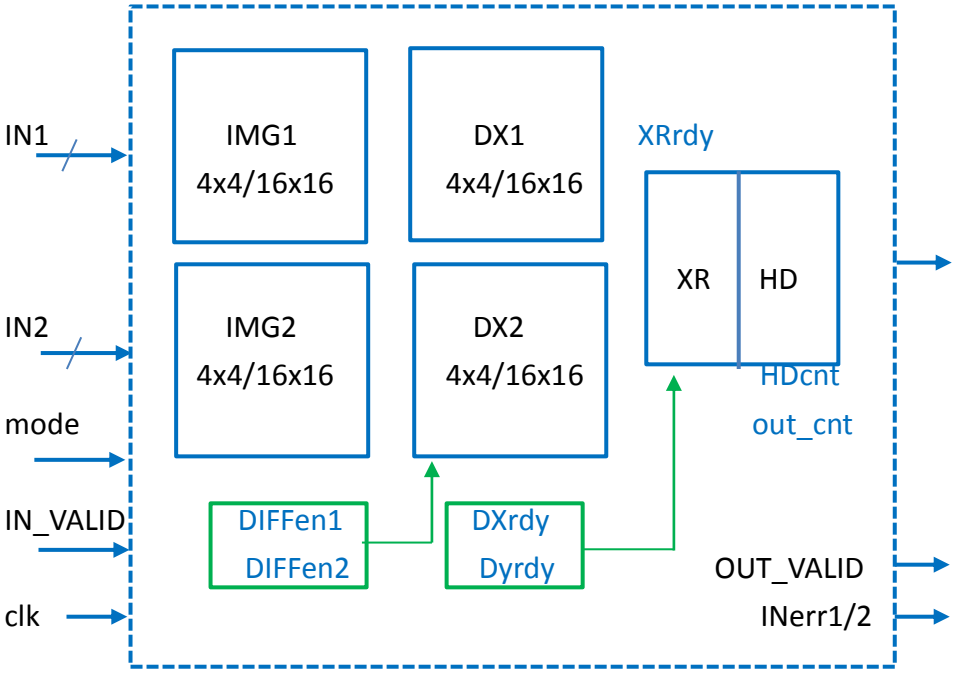
c)) testing data:

** to be offered by TAs, to which the response of one's design could look like the one shown below



[4] Useful Tips

0)) system framework and process timing flow:



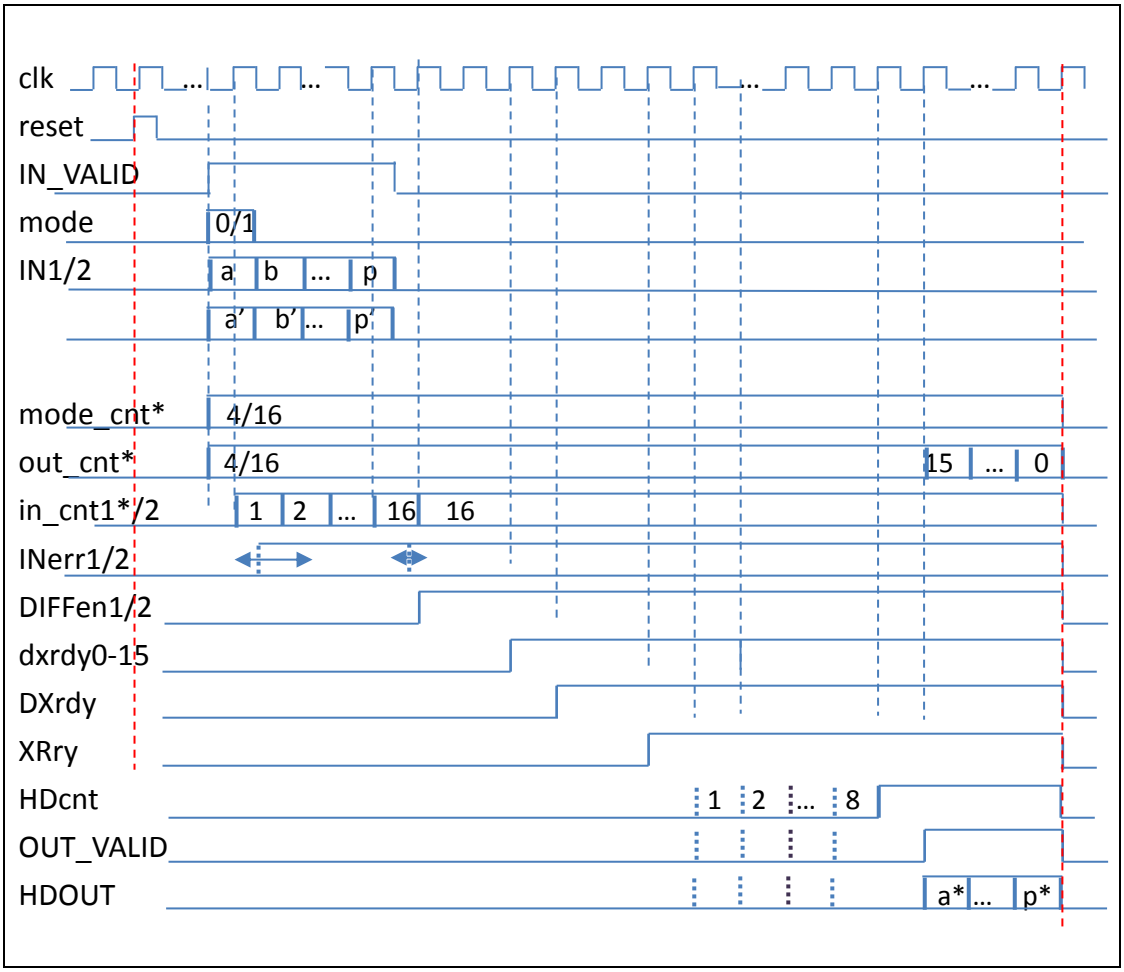
phase I: ** preparation of IMG1 and IMG2;
** when ready, flags DIFFen1 and DIFFen2 are raised for commencing phase II;
** taking either ***4 or 16 clocks*** to complte.

phase II: ** preparation of DX1 and DX2;
** when ready, DXrdy and DYrdy flags are raised for commencing phase III;
** taking ***1 clock cycle*** to complete;

phase IIIa: ** preparation of XR map;
**when ready, flag XRrdy raised for activating HD computing process;
** taking 1 clock to complete.

phaseIIIb: ** preparation of HD map;
** when ready, flags HDcnt(8) and out_cnt(0) are signaled for activating the output sequence;
** taking 8 clocks to complete.

Phase IIIc: ** output sequence completed in either **4 or 16 clocks**, during which OUT_VALID is raised.


clk
reset
IN_VALID
mode
0/1
IN1/2
a
b
...
p
a'
b'
...
p'
mode_cnt*
4/16
out_cnt*
4/16
15
...
0
in_cnt1*/2
1
2
...
16
16
INerr1/2
DIFFen1/2
dxrdy0-15
DXrdy
XRry
HDcnt
1
2
...
8
OUT_VALID
HDOUT
a*
...
p*

1)) intra-image comparison:

Consider a 4x4 grey-level image IMG1 shown below

IMG1

pix0	pix1	pix2	pix3
pix4	pix5	pix6	pix7
pix8	pix9	pix10	pix11
pix12	pix13	pix14	pix15

where $0 =< pix_j =< 255$

** Comparison of pixel values in a 3x3 neighborhood around a each pixel in the image is done by cmp(PIX_{nbr_center}, PIX_{nbr}) in terms of GE or LT.

** For instance, the comparison in the red-neighborhood centered at PIX5 may lead to a result at right if PIX5 possessing the highest grey values, except for PIX10, in the 3x3 neighborhood, and that in the green-neighborhood around PIX10 reflecting the brightness of pix10 than its left and right neighbors only.

1	1	1
1	pix5	1
1	1	0

0	0	0
1	pix10	1
0	0	0

2)) encoding of intra-image comparison:

Consider two 4x4 grey-level images IMG1 and IMG2 shown below

IMG1

pix0	pix1	pix2	pix3
pix4	pix5	pix6	pix7
pix8	pix9	pix10	pix11
pix12	pix13	pix14	pix15

MAP1

X3	X5	**X5**	X3
X5	**X8**	X8	X5
X5	X8	X8	X5
X3	X5	X5	**X3**

IMG2

pix0	pix1	pix2	pix3
pix4	pix5	pix6	pix7
pix8	pix9	pix10	pix11
pix12	pix13	pix14	pix15

MAP2

X3	X5	X5	X3
X5	X8	X8	X5
X5	X8	**X8**	X5
X3	X5	X5	X3

where MAP1 is the result of intra-image comparison of IMG1, in which X_k at each pix-position consists of k bits of 1/0s, reflecting the

brightness of the pixel in question against its k-surrounding neighbors; likewise for MAP2.

** If, for each pixel, the 8 neighbors are indexed from 0 to 7 as follows,

0	1	2
3	pix	4
5	6	7

then X3 and X5 in red of MAP1 have no neighbors indexed by nbr_{indx}: 2-4-5-6-7 and 0-1-2 respectively; whereas X3 and X5 in green of MAP2 have no neighbors indexed by nbr_{indx}: 0-3-5-6-7 and 0-3-5 respectively.

** For every pix-position in MAP1, an 8bit data structure for holding 1/0s indicating the brightness of the very pixel against its 8 neighbors is readily devised. For instance, pix5 and pix10 in MAP1 go with

pix5: 1-1-1-1-1-1-1-0

pix10: 0-0-0-1-1-0-0-0

3)) inter-image comparison by hamming distance evaluation

MAP1

X3	X5	**X5**	X3
X5	**X8**	X8	X5
X5	X8	X8	X5
X3	X5	X5	**X3**

MAP2

X3	X5	X5	X3
X5	X8	X8	X5
X5	X8	**X8**	X5
X3	X5	X5	X3

HDmap (map of hamming distance)

HD_0	HD_1	HD_2	HD_3
HD_4	HD_5	HD_6	HD_7
HD_8	HD_9	HD_{10}	HD_{11}
HD_{12}	HD_{13}	HD_{14}	HD_{15}

where HD_k = count_of_1(XOR($X1_k$, $X2_k$)) k=0-15

[note] ** $X1_k$ of MAP1 is the 8bit data reflecting the relative brightness of pixel k against its neighbor-pixels in IMG1; $X2_k$ of MAP2 carries the same information of IMG2

** HD_k thus reflecting the similarity between IMG1 and IMG2 in the neighborhood of pixel-k (why is that?!)

4)) circuit modules for flag-setting/resetting

```
// top-level
        . . .
    reg     OUT_VALID;                                          // output
    reg     mode_cnt, in_cnt1, in_cnt2, INerr1, INerr2;
    reg     DIFFen1, DIFFen2;
    reg     dxrdy0, …, dxrdy15, dyrdy0, …, dyrdy15, DXrdy, DYrdy,;
    reg     XRrdy, HDcnt, out_cnt,;
        . . .
// flags set/reset
// mode_cnt setting    . . .beware of effective period of mode-input
  always@(posedge IN_VALID or posedge clk)
      if(reset)       mode_cnt<=0;
      else if(!clk && IN_VALID && in_cnt1==0)
                      mode_cnt<=(mode)? 16:4;
      else            mode_cnt<=(HDcnt && out_cnt==0)? 0: mode_cnt;

// out_cnt setting         . . .beware of effective period of mode-input
  always@(posedge IN_VALID or posedge clk)
      if(reset)             out_cnt<=0;
      else if(!clk && IN_VALID && in_cnt1==0)
                            out_cnt<=(mode)? 16:4;
      else            out_cnt<=(HDcnt && out_cnt)? out_cnt-1: out_cnt;

// in_cnt1
   always@(posedge clk)
          if(reset || (out_cnt==0 && HDcnt))
                in_cnt1<=0;
          else    in_cnt1<=(IN_VALID)? in_cnt1+1: in_cnt1;
// in_cnt2
   always@(posedge clk)
          if(reset || (out_cnt==0 && HDcnt))
                in_cnt2<=0;
          else    in_cnt2<=(IN_VALID)? in_cnt2+1: in_cnt2;
```

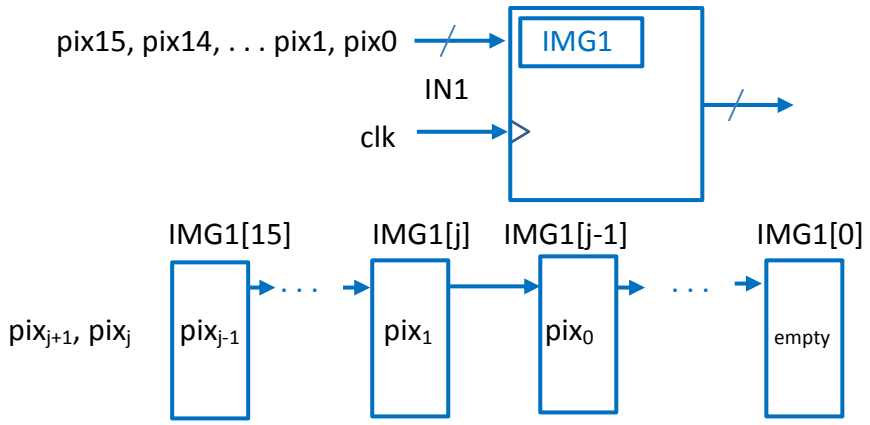
```
// INerr1/2
  always@(posedge clk or negedge IN_VALID)
     if(reset || (out_cnt==0 && HDcnt))
            INerr1<=0;
     else if(IN_VALID)
            INerr1<=(in_cnt1>mode_cnt)? 1: 0;
     else    INerr1<=(!IN_VALID && (in_cnt1!=mode_cnt))? 1: 0;

// DIFFen1/2
  always@(posedge clk)
       DIFFen1<= (reset)? 0:
                  (in_cnt1==mode_cnt)? 1: DIFFen1;
  always@(posedge clock)
       DIFFen2<= (reset)? 0:
                  (in_cnt2==mode_cnt)? 1: DIFFen2;

// dxrdy0-dxrdy15 / dyrdy0-dyrdy15
  always@(posedge clk)
       if(reset || (out_cnt==0 && HDcnt))
            begin dxdry0<=1'b0;
                   . . .
                  dxdry15<=1'b0;
            end
       else begin dclk1<=DIFFen1;       // dxrdy raised 2 clocks after the
                  dxrdy0<=dclk1;        // raising of DIFFen1, falling 2 clocks
                  dxrdy1<=dclk1;        // behind the falling of DIFFen1
                   . . .                // so that at least 1 clock is reserved
                  dxdry15<=dclk1;       // for DX-map preparation
            end
    always@(posedge clk)
       if(reset || (out_cnt==0 && HDcnt))
             begin dydry0<=1'b0;
                   . . .
                  dydry15<=1'b0;
            end
       else begin dclk2<=DIFFen2;
                  dyrdy0<=dclk2;
```

```
                    dyrdy1<=dclk2;
                     . . .
                    dydry15<=dclk2;
             end
// DXdry/DYrdy
   always@(posedge clk)
          if(reset || (out_cnt==0 && HDcnt))
                        DXrdy<=1'b0;
          else if(mode_cnt==16)
                    begin DXrdy<=(dxrdy0 && … && dxrdy15)? 1: 0;
                        DYrdy<=(dyrdy0 && … && dyrdy15)? 1: 0;
                    end
          else if(mode_cnt==4)
                    begin DXrdy<=(dxrdy0 && … && dxrdy3)? 1: 0;
                        DYrdy<=(dyrdy0 && … && dyrdy3)? 1: 0;
                    end
          else begin      DXrdy<= DXrdy;
                        DYrdy<= DYrdy;
                end;
// XRrdy
   always@(posedge clk)
        if(reset || (out_cnt==0 && HDcnt))
                      XRrdy<=1'b0;
          else begin XRclk1<=DXrdy & DYrdy;      // at least 1 clock for
                      XRrdy<=XRclk1;              // concurrent XORing
                end
// HDcnt
     always@(posedge clk)
        if(reset ||(out_cnt==0 && HDcnt))
                    HDcnt<=1'b0;
        else begin HDclk1<=XRrdy;
                    HDclk2<=HDclk1;
                    HDclk3<=HDclk2;
                    HDclk4<=HDclk3;
                    HDclk5<=HDclk4;
                    HDclk6<=HDclk5;
                    HDclk7<=HDclk6;
                    HDclk8<=HDclk7;
```

```
                    HDcnt<=HDclk8;
                end
// OUT_VALID
  always@(posedge clk)      OUT_VLAID<=(HDcnt && out_cnt) 1:0;
```

5)) circuit module for receiving a block of pixel values coming in serial order

** Consider receiving the input of a 4x4 image as a sequence of data bytes at IN1 (strobing by clk), the use of a rightt-shift module of 16-byte unit for holding IMG1 pixel values may look sound.



at clock$_j$, the 16-byte unit shifts right by 1 and then the input byte of pix$_j$ is strobed into IMG[15]

** IMG1 becomes ready 16 clocks after the arrival of IN_VALID by raising DIFFen1 flag

** IMG2 becomes ready 16 clocks after the arrival of IN_VALID by raising DIFFen2 flag

```
 //top-level
      . . .
      reg [7:0] IMG1 [15:0],
                IMG2[15:0];
// receiving input sequence at IN1
      always(posedge clk)
       if(IN_VALID && mode_cnt==16)
                    begin IMG1[15]<=IN1;
                          IMG1[14]<=IMG1[15];
                          IMG1[13]<=IMG1[14];
                          . . .
                          IMG1[2]<=IMG1[3];
                          IMG1[1]<=IMG1[2];
```

```
                    IMG1[0]<=IMG1[1];
                    in_cnt1<=in_cnt1-1;
                end
  else if(IN_VALID && mode_cnt==4)
                begin IMG1[3]<=IN1;
                    IMG1[2]<=IMG1[3];
                    IMG1[1]<=IMG1[2];
                    IMG1[0]<=IMG1[1];
                    in_cnt1<=in_cnt1-1;
                end
  else IMG1[0]<=IMG1[0];
// receiving input sequence at IN2
  . . .
```

6)) module for intra-image neighborhood comparison at each pixel in an image

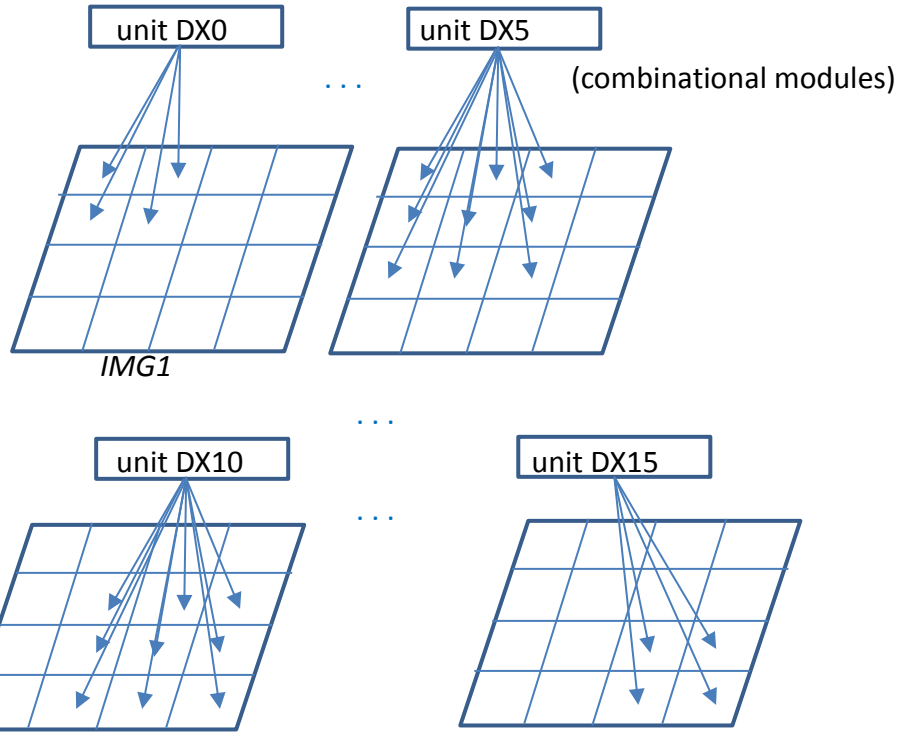
16 units of intensity-difference over a 3x3 neighborhood are required, which are to operate concurrently, as intensity-difference around each of the 16 pixels in a 4x4 image needs to be computed. Each unit would cover a central pixels and the 8neighbors

** DX0-15 for IMG1, DY0-15 for IMG2

** each set of the 16 units operates concurrently on all pixels in IMG1 and IMG2 concurrently

** the operation of each unit is to be **done in 1 clock cycle**, thus the operation for preparing DX and DY maps takes also 1 clock cycle

** DXrdy and DYrdy flags are raised when DX and DY maps are ready

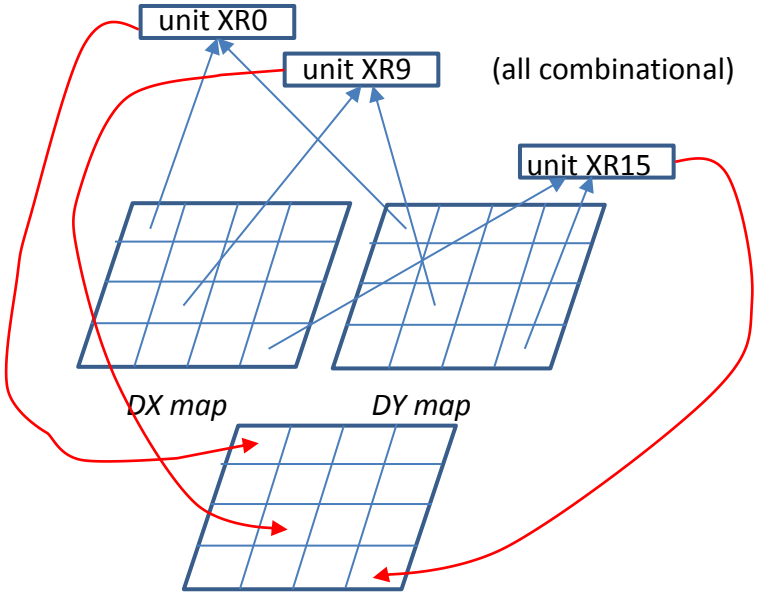



```
// top-level
  . . .
  reg dxrdy0, …, dxrdy15;
  reg dyrdy0, …, dyrdy15;
  . . .
// IMG1 intra-comparison          DX0~DX15(all combinational)
always@(posedge clk)                                  // DX0
  if(DIFFen1)
    begin
```

```
    difx04<=(IMG1[0]>=IMG1[1])? 1:0;
    difx06<=(IMG1[0]>=IMG1[6])? 1:0;
    difx07<=(IMG1[0]>=IMG1[7])? 1:0;
    DX0<={0,0,0,0, difx04, 0, difx06, difx07};
    // dxrdy0<=1;
    end
  else DX0<=DX0;
always@( posedge clk)                                   // DX1
  if(DIFFen1)
    begin
    difx13<=(IMG1[1]>=IMG1[3])? 1:0;
    difx14<=(IMG1[1]>=IMG1[4])? 1:0;
    difx15<=(IMG1[1]>=IMG1[5])? 1:0;
    difx16<=(IMG1[1]>=IMG1[6])? 1:0;
    difx17<=(IMG1[1]>=IMG1[7])? 1:0;
    DX1<={0,0,0, difx13, difx14, difx15, difx16, difx17};
    //dxrdy1<=1;
    end
  else DX1<=DX1
       . . .
always@(posedge clk)                                    // DX5
  if(DIFFen1)
    begin
    difx50<=(IMG1[5]>=IMG1[0])? 1:0;
    difx51<=(IMG1[5]>=IMG1[1])? 1:0;
    difx52<=(IMG1[5]>=IMG1[2])? 1:0;
    difx53<=(IMG1[5]>=IMG1[4])? 1:0;
    difx56<=(IMG1[5]>=IMG1[6])? 1:0;
    difx58<=(IMG1[5]>=IMG1[8])? 1:0;
    difx59<=(IMG1[5]>=IMG1[9])? 1:0;
    difx510<=(IMG1[5]>=IMG1[10])? 1:0;
    DX5<={difx50, difx51, …, difx59, difx510};
    //dxrdy5<=1;
    end
  else DX5<=DX5;
        . . .
always@(posedge clk)                                    // DX15
  if(DIFFen1)
```

```
begin
difx150<=(IMG1[15]>=IMG1[10])? 1:0;
difx151<=(IMG1[15]>=IMG1[11])? 1:0;
difx153<=(IMG1[14]>=IMG1[14])? 1:0;
DX15<={difx150, difx151, 0, difx153, 0, 0, 0, 0};
//dxrdy15<=1;
end
else DX15<=DX15;
```

7)) preparation of XR map prior to hamming distance computation:



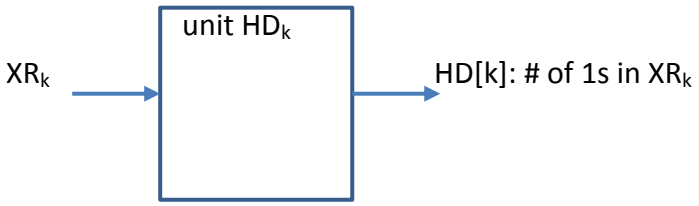
XR map

** once DXrdy and DYrdy being raised, 16 XR-modules activated concurrently, and XR map is generated in 1clock cycle;

```
. . .
reg [7:0] XR0, XR1, …, XR14, XR15;
reg [3:0] HD [15:0];
. . .
always@(posedge clk)
 if(DXrdy && DYrdy && mode_cnt==16 && !XRrdy)
        begin XR0<=DX0 ^ DY0;
              XR1<=DX1 ^ DY1;
              XR2<=DX2 ^ DY2;
              XR3<=DX3 ^ DY3;
```

```
                    XR4<=DX4 ^ DY4;
                    XR5<=DX5 ^ DY5;
                    XR6<=DX6 ^ DY6;
                    XR7<=DX7 ^ DY7;
                    XR8<=DX8 ^ DY8;
                    XR9<=DX9 ^ DY9;
                    XR10<=DX10 ^ DY10;
                    XR11<=DX11 ^ DY11;
                    XR12<=DX12 ^ DY12;
                    XR13<=DX13 ^ DY513
                    XR14<=DX14 ^ DY14;
                    XR15<=DX15 ^ DY15;
                    // XRrdy<=1'b1;
            end
        else if(DXrdy && DYrdy && mode_cnt==4 && !XRrdy)
                begin XR0<=DX0 ^ DY0;
                        XR1<=DX1 ^ DY1;
                        XR2<=DX2 ^ DY2;
                        XR3<=DX3 ^ DY3;
                        // XRrdy<=1'b1;
                else XR0<=XR0;              // anything else better to do here???

always@(posedge clk)
   if(XRrdy) begin dxrdy0<=1'b0;
                     dyrdy0<=1'b0;
                    end
   else begin
            end
```

8)) hamming distance computing and sending HD output sequence:



** enabled by XRrdy, 16 HD units operate concurrently, counting

```
        1-bit in respective XR_k
    ** the counting takes 8 clocks
    ** OUT_VALID flag raised when all HD units are done


    . . .
always@(posedge clk)          // complete 16 HD-computation in 8 clocks
        if(XRrdy && mode_cnt==16 && HDcnt<8)
                begin HD[0]<=HD[0]+XR0[7];
                        XR0<={XR0[6:0], 1'b0};
                        HD[1]<=HD[1]+ XR1[7];
                        XR1<={XR1[6:0], 1'b0};
                              . . .
                        HD[14]<=HD[14]+XR14[7];
                        XR14<={XR14[6:0], 1'b0};
                        HD[15]<=HD[15]+ XR15[7];
                        XR15<={XR15[6:0], 1'b0};
                        // HDcnt<=HDcnt+1;          controlled by HDcnt module
                end
        else if(XRrdy && mode_cnt==4 && HDcnt<8)
                begin HD[0]<=HD[0]+{7'b0000000, XR0[7]};
                        XR0<={XR0[6:0], 1'b0};
                           . . .
                        HD[3]<=HD[3]+{ 7'b0000000, XR3[7]};
                        XR3<={XR3[6:0], 1'b0};
                        // HDcnt<=HDcnt+1;
                end
        else    HD[0]<=HD[0];

    always@(posedge clk)
       begin         // OUT_VLAID<=(HDcnt && out_cnt) 1:0;
              if(HDcnt && out_cnt && mode_cnt==16)
                 begin HDOUT<=HD[0];
                        HD[0]<=HD[1];
                        HD[1]<=HD[2];
                        HD[2]<=HD[3];
                           . . .
                        HD[13]<=HD[14];
                        HD[14]<=HD[15];
```

```
                    //out_cnt<=out_cnt-1;
                end
        else if(HDcnt && out_cnt && mode_cnt==4)
                begin HDOUT<=HD[0];
                        HD[0]<=HD[1];
                        HD[1]<=HD[2];
                        HD[2]<=HD[3];
                        //out_cnt<=out_cnt-1;
                end
        else HDOUT<=1'bZ;
end
```