Individual Coursework 2:
draw_any_octant

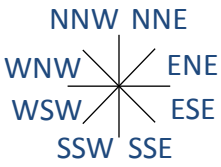
This exercise is a translation from RTL description to VHDL using structural description with GENERICs as well as behavioural.

The **draw_octant** entity can only draw lines in one octant (East North East direction, where Y axis points N and X axis points E). Figure 1 shows the various octants.

If the x direction numbers have bits inverted on input and output this transforms x → not(x) reflecting all coordinates in a vertical line and the entity will draw lines in direction West North West. If the x and y numbers are swapped (x,y)→(y,x) the entity will draw lines in North North East quadrant. Two hardware blocks which implement inversion and swapping are shown in Figure 2.

Drawing in Other Octants

Octant determines values of negx, negy, swapxy



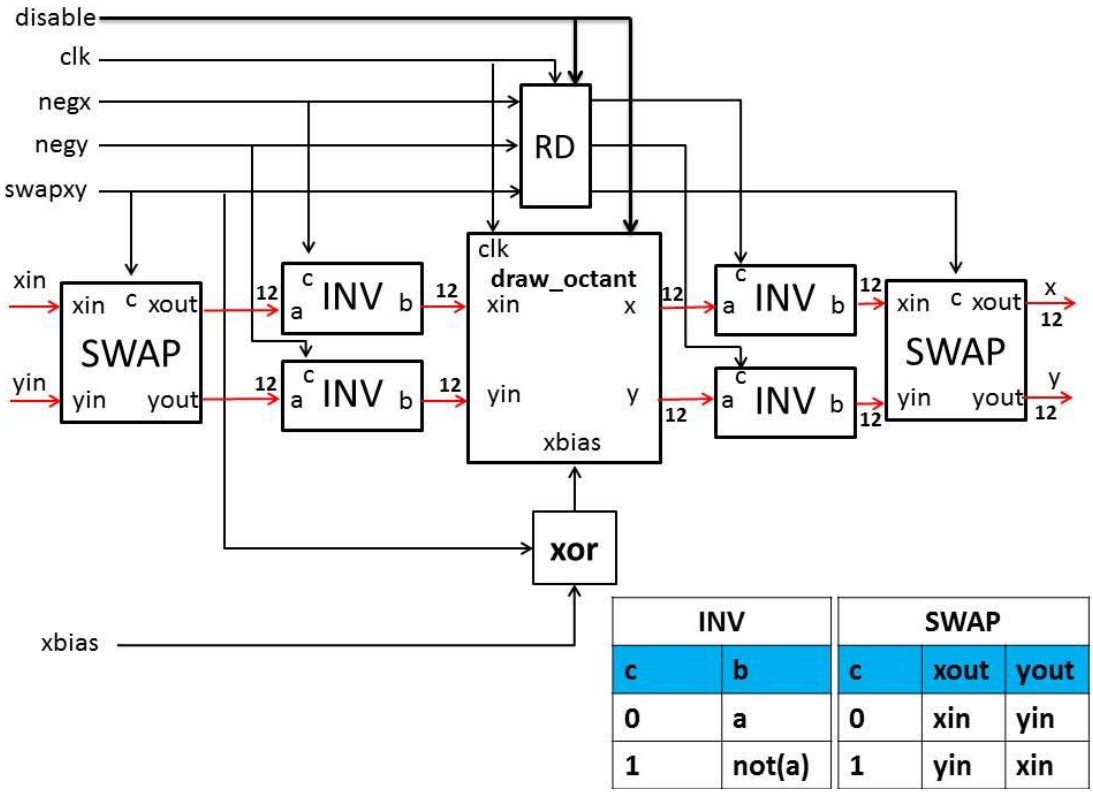
Octant	negx	negy	swap
ENE	0	0	0
NNE	0	0	1
NNW	1	0	1
WNW	1	0	0
WSW	1	1	0
SSW	1	1	1
SSE	0	1	1
ESE	0	1	0

Figure 1

Figure 2 shows hardware which implements swapping X & Y, or inverting the bits of X, or inverting the bits of Y. The 8 possible combinations correspond to 8 possible octants. All blocks are combinational except for those with clk shown as input. The XOR block is a two input **xor** gate. The **swapxy**, **negx**, **negy** inputs control **xin**, **yin** inputs directly, however they must therefore be delayed by one cycle in a 3 bit D type register shown as RD in Figure 2 to match **x** & **y** in the next cycle.

- Write an entity **draw_any_octant** which implements Figure 2 and passes the provided testbench as required in the assessment section.
- Note that some ports of **draw_octant** are left out of Figure 2, all such ports must be connected directly to similarly named ports on **draw_any_octant**. See skeleton file for a suitable entity for **draw_any_octant** which contains these ports.
- Your hardware must use a GENERIC parameter **vsize** to determine vector lengths (see assessment section).

You may implement the blocks SWAP, INV as behavioural or structural code. Your code must be submitted in a single file **draw_any_octant.vhd** which may contain multiple entities (see Assessment section).



INV	
c	b
0	a
1	not(a)

SWAP		
c	xout	yout
0	xin	yin
1	yin	xin

Figure 2

Assessment

You are given in this exercise a **draw_octant** entity and architecture, adapted to add **vsize**, which you can use for testing instead of your own ex1 solution.

- Design your **draw_any_octant** entity so that it will accept a generic integer parameter **vsize** which determines the size of the pixel coordinate vectors shown in red in Figure 2. This size can thus be different from 12.
- **vsize** must be passed to the **draw_octant** subentity.
- Please submit your **draw_any_octant** code (passing the testbench) in file **draw_any_octant.vhd** without any other supporting files. It will be tested with the provided **draw_octant** entity at various **vsize** sizes.

If you wish to define additional sub-entities (allowed but not required, since for small blocks behavioural implementation is also good) you can do this inside the **draw_any_octant.vhd** file. You will remember from lecture 4 that in this case your code must be arranged as below:

1. Context clauses
2. Mysubentity1
3. Architecture for mysubentity1
4. Repeat 1,2,3 for other sub-entities
5. Context clauses
6. Draw_any_octant entity
7. Draw_any_octant architecture

Applications information

The line-drawing function of **draw_octant** is described in the ex1 sheet.

The hardware you have now designed will draw a line in any direction, given that **resetx** and **draw** are used initially to enter the line's initial and final coordinates. Done will be asserted when the line drawing (which can take up to vsize clocks) is finished. To draw an arbitrary octant lines **negx**,**negy**,**swapxy** must all be set correctly, according to the line octant, before the start of drawing and held at this value till the line is finished (and **done** asserted). If correct values for these parameters are chosen, **draw_octant** will calculate **xincr** and **yincr** which obey the requirements for it to draw a line correctly.

Xbias is an input which has a very small effect on the drawn line. Whether it is 0 or 1 the drawn line will look almost identical. However it determines which pixel is plotted when the two errors are equal (and therefore the drawing algorithm cannot determine which pixel is nearest to the true line). In your project work later on it will be important to set **xbias** correctly so that your code is compatible with the specification and testbench.

You should consider the following questions which will help you with this design:

- Why is **xbias** inverted (using the xor gate) when **swapxy** is 1?
- Why is the INV operation x→not(a) and not negate: x→-x?
- Does it make a difference to operation if the order of the SWAP and INV blocks is reversed (so INV is outer and SWAP is inner)?

Alternative use of VHDL data structures

After having submitted this deliverable reflect on how the code you have written could be rewritten and simplified with two different choices of data structures for pairs of x,y values:

1. (x,y) is implemented as a record
2. (x,y) is implemented as a a 2 element array. This allows a FOR loop to implement x and y logic as two iterations of the same code.