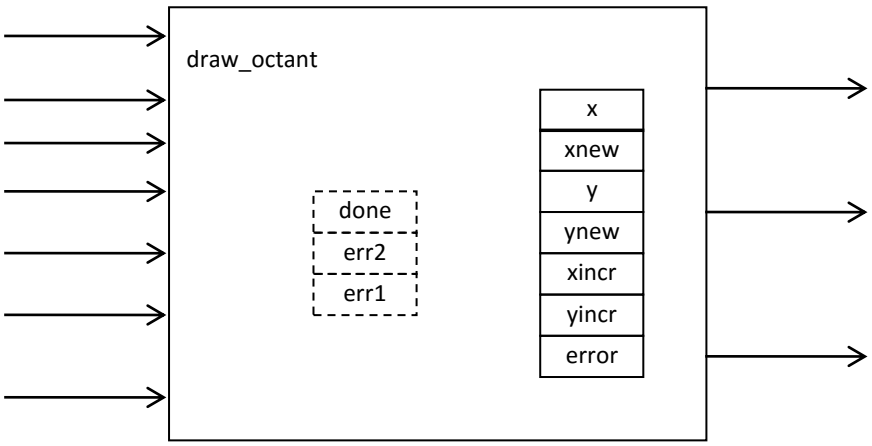
COURSEWORK EXAMPLE 1

This example will illustrate clocked and combinational VHDL behavioural code, with use of flow control and arithmetic.

Complete the given skeleton file **draw_octant.vhd** to make the entity `draw_octant` as specified below. To test in simulation, make a Modelsim project with given files **draw_octant_tb.vhd**, **data_pak.vhd**, **draw_octant.vhd**. Running this project will test functionality. See notes with files for more information. To check VHDL synthesis run synthesis to **draw_octant.vhd** using Synplify Pro. Check all warnings - there should be none. Submit your file **draw_octant.vhd** (don't change name) as specified on the web pages or discussion group.

draw_octant - see zip file ENTITY for inputs and outputs



Combinational process

Inputs								Combinational outputs (bits)		
								(13)	**(13)**	**(1)**
error	**xincr**	**yincr**	**x**	**y**	**xnew**	**ynew**	**disable**	**err1**	**err2**	**done**
X	X	X	X	X	X	X	X	|error + yincr|	|error+yincr - xincr|	x = xnew and y = ynew and init = 0 and draw = 0

X = don't care input. |exp| = if exp < 0 then -exp else exp

Registered process

Inputs						Registered outputs (bits)						
						(13)	**(12)**	**(12)**	**(12)**	**(12)**	**(12)**	**(12)**
disable	**err1,err2**	**xbias**	**init**	**draw**	**done**	**error**	**x**	**y**	**xincr**	**yincr**	**xnew**	**ynew**
0	n/a	X	1	X	X	0	xin	yin	0	0	xin	yin
0	n/a	X	0	1	X	0	nc	nc	xin-x	yin-y	xin	yin
0	err1 > err2	X	0	0	0	error+yincr-xincr	x+1	y+1	nc	nc	nc	nc
0	err1 < err2	X	0	0	0	error+yincr	x+1	nc	nc	nc	nc	nc
0	err1 = err2	1	0	0	0	error+yincr	x+1	nc	nc	nc	nc	nc
0	err1 = err2	0	0	0	0	error+yincr-xincr	x+1	y+1	nc	nc	nc	nc
0	n/a	X	0	0	1	nc	nc	nc	nc	nc	nc	nc
1	n/a	X	X	X	X	nc	nc	nc	nc	nc	nc	nc

NB - this will correctly draw a line in octant xincr ≥ 0, yincr ≥ 0, xincr ≥ yincr

nc = no change in output. X = don't care input

NOTES ON HARDWARE DESIGN

This hardware requires at least two processes, for combinational & registered outputs.

You may split a multiple output process into separate processes each handling different sets of outputs, however in this case that will not make the code easier to understand, and is not recommended.

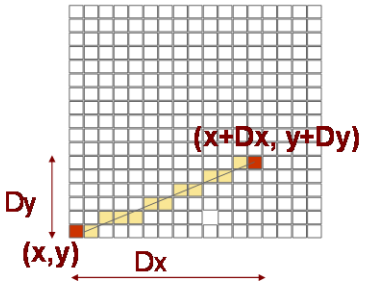
You may code multiple outputs inside a single PROCESS in two ways. They may be coded together inside a single (possibly nested) set of IF or CASE statements. Or they may be coded separately, with independent IF/CASE statements controlling each output. It is usually clearer to code outputs together when they are controlled by similar conditions, since this allows the IF/CASE logic to be written only once and so simplifies the hardware. However if outputs have different controlling conditions it may be clearer to implement them independently.

This is a design choice, and different good solutions are possible. It is good practice to choose the most compact implementation which is easy to understand. It makes no difference to the synthesised hardware whatever code is used, so the choice here is about how easy is the design to read and verify.

FOR INTEREST ONLY: DRAWING ALGORITHM THEORY

Bresenham Differential Drawing Algorithm (DDA)

- Draw line from (x,y) to (x+Dx, y+Dy)
- Plot line by writing pixels sequentially
- Each step will move 1 pixel horizontally, vertically, or diagonally
- Use incremental line drawing algorithm to work out which direction to move at each step.
- Simplify problem by solving for just 1 octant: Dx > 0, Dx >= Dy >= 0



To work out next pixel choose *move direction* which results in pixel nearest to exact line

Find direction which minimises offset from "true" line

err1 = err - Dy/Dx + 1
err2 = err - Dy/Dx

NB at start and end of line err = 0

Is | err' | smaller for horizontal or diagonal move?

Choose smallest error move, draw new pixel & update err. Then repeat until end of line

Compute Dx.err to avoid division and keep calculation integral

*Generalise to all octants using the two **move directions** (x1,y1) and (x2,y2).*