INDIVIDUAL COURSEWORK 4

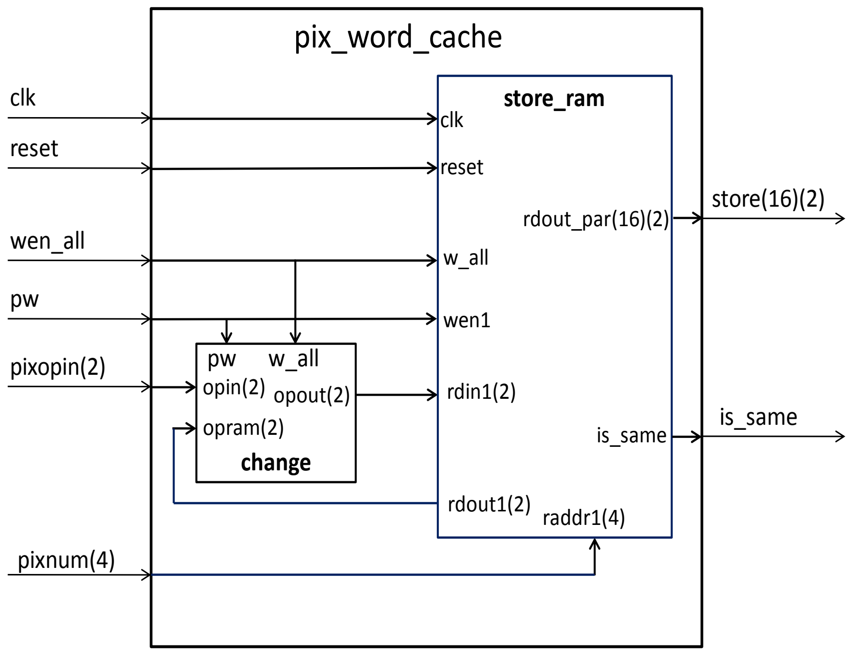

Figure 1

This coursework requires the design of a VHDL entity and architecture **pix_word_cache** as shown in Figure 1. The numbers in brackets after input, output, or internal signals specify the size in bits of those signals. Output **store** is a 16 element array of 2 bit vectors each represented by type **pixop_t**. The type for the output array is **store_t**. Signals without sizes are assumed single bit.

You are given a package **pix_cache_pack** which contains the types and constants relevant to this design and which you should use and **must not change**. Note that it contains some useful conversion functions.

Pix_word_cache is specified as two sub-blocks (although the implementation in VHDL should be implemented in whatever way leads to the clearest design) connected as in Figue 1. **Change** is a combinational block defined as in Figure 3. **Store_ram** is a special-purpose 16 word X 2 bit RAM with a parallel output signal **rdout_par** that outputs the values of all its words simultaneously and a single read/write port **wen1/rdin1/rdout1**, where **rdout1** is the output data from the word with address **raddr1**, and wen1 is as defined in Figure 2 below. The 2 bits of each **store_ram** word have values: **psame**, **pblack**, **pwhite**, **pinvert** as defined by constants in **pix_cache_pak**. Each word is implemented by array type **pixop_t** given to you in **pix_cache_pak**.

Output **is_same** of **store_ram** is 1 only when all locations in **store_ram** have value **psame**. The **store_ram** RAM is implemented synchronously as positive edge triggered flip-flops, so that writing to the RAM will cause the RAM contents to change in the cycle after the write operation. All RAM outputs show the current flip-flop values. Inputs **reset**, **w_all** and **wen1** implement synchronous writing to RAM locations as in Figure 2. Note that the truth table for the **change** block in Figure 3 ensures that the **raddr1** port of the RAM has a value of **rdin** consistent with this.

reset		w_all	wen1	Operation (after clock edge)
1		X	X	All RAM words written to **psame**
0		1	0	All RAM words written to **psame**
0		1	1	All RAM words written to **psame** except word with address **raddr1** which is written to **rdin1**
0		0	1	Word with address **raddr1** is written to **rdin1**, other RAM words do not change value
0		0	0	No change

Figure 2

CHANGE inputs				**CHANGE Output**
w_all	**pw**	**opram**	**opin**	**opout**
0	X	psame	p	p
0	X	pblack	psame	pblack
0	X	pblack	pblack	pblack
0	X	pblack	pwhite	pwhite
0	X	pblack	pinvert	pwhite
0	X	pwhite	psame	pwhite
0	X	pwhite	pblack	pblack
0	X	pwhite	pwhite	pwhite
0	X	pwhite	pinvert	pblack
0	X	pinvert	psame	pinvert
0	X	pinvert	pblack	pblack
0	X	pinvert	pwhite	pwhite
0	X	pinvert	pinvert	psame
1	0	X	p	psame
1	1	X	p	p

X indicates "don't care" for inputs
p indicates an arbitrary pixel operation

Figure 3

APPLICATIONS INFORMATION

The input is a sequence of 'pixel change' operations (up to one per cycle) from a draw line block (you've probably guessed how that relates to EX1). Each pixel is 0 or 1, however pixel **change** operations require two bits.

The idea of a 'pixel change' operation allows a number of pixel changes (from a block that draws lines etc) to be held in hardware until such time as it is convenient to write them out to the real pixels stored in an external RAM. When the 'pixel change' operations are actually written to RAM they must be combined with RAM data using a read followed by write or "read-modify-write" cycle. That is because the "pinvert" pixel operation sets the final RAM pixel value (white or black) to be the opposite of the original RAM pixel value (black or white).

If we allow *same* (no change) as a pixel operation, then any two pixel operations, and therefore any sequence of pixel operations, can be combined together to make a single equivalent operation as shown in Figure 3.

Storing a pixel change *operation* in a cache RAM instead of actually implementing the pixel change in a cache RAM of locations representing pixels, is sophisticated. Note that although a pixel (stored in a RAM) has only two possible values, black or white, a pixel change operation (as stored in pix_word_cache) has 4 possible values (pblack, pwhite, psame or pinvert). When drawing lines it is more efficient to hold and combine all *pixel change operations* in pix_word_cache and then write them out to RAM using a Read Modify Write memory cycle to read and then write 16 pixels in parallel.

Each *pixel change operation* specifies X & Y coordinates of the pixel, type of operation (**psame**, **pinvert**, **pblack** or **pwhite**). Each square of 4X4 pixels is stored in a single (16 bit) memory location in a large RAM external to **pix_word_cache**. Any number of pixel operations on the same memory location (4X4 square of pixels) can be stored in the 16X2 bit **store** memory. In the case that many pixel operations on the same 4X4 square are issued consecutively the **pix_word_cache** block can therefore combine all operations together making a large saving on external memory operations, but even if not the typical saving drawing a line is 4 pixel operations per memory operation because each line will typically draw 4 pixels in a 4X4 pixel square it crosses.

The locations in one **pix_word_cache** block represent a 4X4 pixel square and can thus be addressed by a 4 bit address **pixnum**. The geometric mapping between **pixnum** value and the 16 pixels in the 4X4 square is not relevant to this exercise and will be defined elsewhere when this block is used in your team project.

One interesting use is that if **wen_all** and **pw** are both 1 the block can write out its old data **store**, initialise **store** to all **psame**, and write to **store** the first new **pixopin** operation, all in one cycle. This overlap increases efficiency in a typical application.