		CHANGE inputs		CHANGE Output
wen_all	**pw**	**dout1**	**pixopin**	**din1**
0	X	same	same	same
0	X	same	black	black
0	X	same	white	white
0	X	same	invert	invert
0	X	black	same	black
0	X	black	black	black
0	X	black	white	white
0	X	black	invert	white
0	X	white	same	white
0	X	white	black	black
0	X	white	white	white
0	X	white	invert	black
0	X	invert	same	invert
0	X	invert	black	black
0	X	invert	white	white
0	X	invert	invert	Same
1	0	X	*p*	Same
1	1	X	*p*	*p*

Figure 2

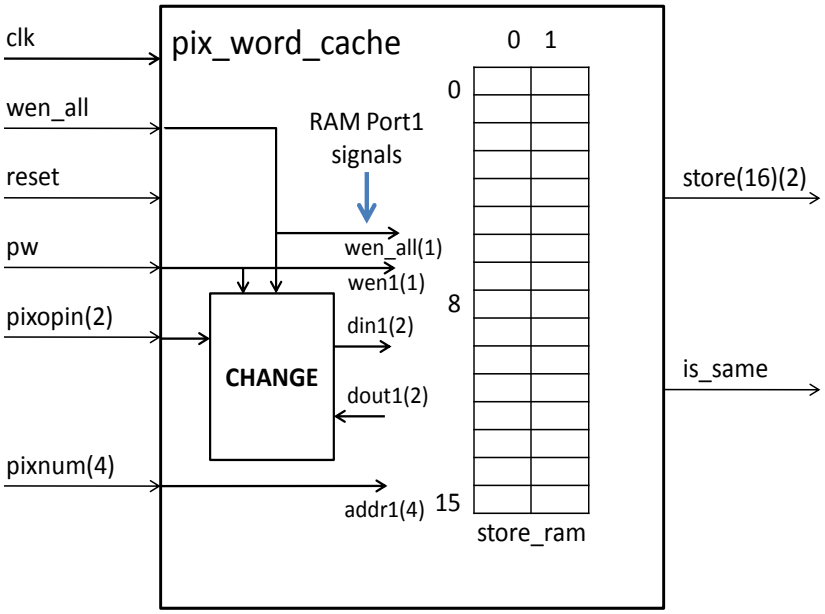

Figure 1

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, note that output **store** is a 16 element array of 2 bit vectors. The entity contains a special-purpose 16 word X 2 bit RAM **store_ram** with a parallel output (connected to **store)** and a read/write port which is used by logic internal to **pix_word_cache**, as shown in Figure 2. The 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.

- You are given a package **pix_cache_pack** which contains types and constants relevant to this design and which you should use and **must not change**.
- The parallel data output **store**(16)(2) is always available and equal to the current values of the RAM flip-flops.
- The 2 bits of each **store_ram** location have values: **same**, **black**, **white**, **invert**. Each location is implemented by array type **pixval** given to you in **pix_pack**, which also defines these constants.
- An input **reset,** if high at an active clock edge, over-rides all other functions (including **wen_all**) and writes all store locations to **same**.
- The **store_ram** read/write port1 and associated logic CHANGE connected in Figure 1 is defined as shown in Figure 2. In Figure 2 signals **addr1**, **wen1** are the address and (single word) synchronous write enable for the RAM port. The read operation (**dout1**) is combinational. The block CHANGE is demarcated in Figure 1 in order to clarify the specification, it need not be implemented as a separate block in your hardware.
- Input **wen_all**, when equal to '1' at an active clock edge, writes value *same* to *all* locations in the RAM, except that addressed by **addr1**. The **addr1** location is written with *same* if **pw** is '0' and pixopin when **pw** is also '1'. Note that this operation is very similar to that of **reset**, different only for the pixel addressed by addr1 in the case that **pw** is also '1'.
- **Pix_word_cache** has an additional combinational output **is_same** which is 1 if and only if all RAM locations are equal to **same**. This output indicates that no change to pixel memory is specified by the operations currently stored.

APPLICATIONS INFORMATION

The input is a sequence (up to one per cycle) of 'pixel change' operations 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. If we allow *same* (no change) as a pixel operation any two pixel operations can be combined together to make a single equivalent operation as shown in Figure 2.

This concept: storing a pixel change operation in a RAM instead of actually implementing the pixel change in a 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 (black, white, same or invert). When drawing lines it will turn out to be 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, rather than write pixel change operations directly to the RAM contents.

Each *pixel change operation* specifies X & Y coordinates of the pixel, type of operation (**same**, **invert**, **black** or **white**). 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 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.

Note that although 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 defined elsewhere.

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 **same**, and write to **store** the first new **pixopin** operation (with a different value of **pixword**), all in one cycle. This overlap increases efficiency in a typical application.