HUNT ENGINEERING
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4BP, UK
Tel: (+44) (0)1278 760188,
Fax: (+44) (0)1278 760199,
Email: sales@hunteng.co.uk
http://www.hunteng.co.uk
http://www.hunt-dsp.com

Using the off chip SDRAM for image processing on HERON FPGA modules

Rev 1.0 J.Maddocks 23-09-2003

Introduction

The document “Image processing with the HERON-FPGA Family” discusses how you could use a HERON module that has an FPGA to perform image processing functions. That document discusses several classes of imaging functions, one of which is neighbourhood processing. These functions perform processing on a region of an image, where that region involves several lines of image data. Because the image arrives pixel by pixel, this means that at least some of the image must be stored somewhere to make those several lines of data available.

While you can store small images into the internal BLOCK RAM of the FPGA (the method used in our imaging with FPGA framework/demo) there can be several reasons for it being necessary to use memory that is outside of the FPGA for that storage.

1. If the storage required exceeds the total BLOCK RAMS available in the FPGA you are using
2. If your FPGA design requires BLOCK RAMS for other logic, effectively causing (1)
3. If you want to store your original image as well as the results of your neighbourhood processing

In these cases care must be taken to manage the SDRAM to achieve best performance of your system, so this document is provided to highlight the issues that need to be considered, and to discuss different ways to use the SDRAM interface. There is no clear right or wrong way to do this, so you need to consider the pros and cons of each method when used in your system and choose the best method for you.

Before reading this document you should understand all of the concepts discussed in the document ‘Image proceesing with the HERON-FPGA family’.

One common neighbourhood processing function is Convolution, and is the function that we will use in the discussions of this document. A convolution function is one where the value of a pixel is decided by the result of maths on its self and its neighbours. A convolution function can be used to sharpen or soften an image or more usually detect edges within a picture. Looking at the example in figure 1, the value of the pixel in the red square is decided by the value of the pixels surrounding it and its self multiplied by a ‘mask’ such as that shown in figure 2. This mask is then moved around the frame until every pixel is calculated.

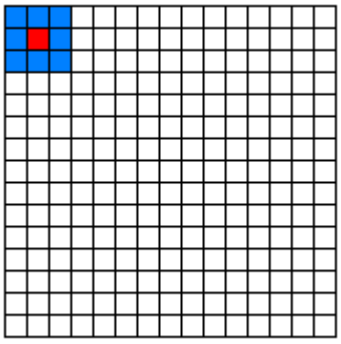


Figure 1

Figure 2

For the rest of the rest of this document the pixels in this mask will be referenced as follows:

A1 B1 C1

A2 B2 C2

A3 B3 C3

What does the SDRAM do for me?

Modules such as the FPGA 5 and its derivatives include either 256Mbytes or 512Mbytes of off chip memory.

To quantify the amount of memory required for image processing applications we can work through some numbers.

In a 512 x 512 image there are 262144 pixels.

At a colour depth of 12 bits this equates to,

12 x 262144 = 3,145,728 Bits / frame

3,145,728 / 8 = 393,216 Bytes / frame

268,435,456 Bytes / 393,216 Bytes = 682 frames in 256Mbytes of memory

Each location in memory is 32 bits wide so even if we wanted to simplify the logic that interfaces with the memory a little and allow 16 bits per pixel i.e. 2 pixels per address we still arrive at only,

16 x 262144 = 4,194,304 Bits / frame

4,194,304 / 8 = 524,288 Bytes / frame

268,435,456 Bytes / 524,288 Bytes = 512 frames in 256Mbytes of memory

The maximum frequency the SDRAM can be clocked at is 133Mhz and therefore the maximum bandwidth is 133,000,000 Mhz * 4 Bytes = 532,000,000 Bytes / sec. This is an unrealistic figure to work with however as even if the data could be supplied at this rate the way in which the SDRAM works makes it impossible to sustain this speed. This is not to say however that the speed of the memory is in anyway a limiting factor when using the SDRAM for image processing applications. If we look at an example,

512 pixels x 512 lines x 16-bits = 524,288 Bytes / frame
524,288 * 25 frames per second = 13,107,200 Bytes / sec this is nothing next to our maximum bandwidth shown above. We could have multiple processes accessing the SDRAM simultaneously and even after the overheads of arbitration and the increased number of SDRAM set-up time's bandwidth should not be an issue.

SDRAM organisation and use

The SDRAM is divided up at the lowest level in to rows and columns. To access an element in an SDRAM module, first the row is opened and is then said to be the active row, secondly a column with in that row is selected and the data output from its location. This setting up of the SDRAM can take several clock cycles and every access will require the appropriate row to be activated. Once a row is active it is possible to read data from an entire row with out reopening that row on every access. This is called bursting. To summarise it is possible to access subsequent locations in memory on every clock cycle so long as those locations are with in the same row in SDRAM. The address within the SDRAM will automatically increment once a location has been read. Typically the overhead of opening and closing a row will be 6 cycles. The second condition that has to be met is the minimum row open time. This would typically be 9 cycles. Only relatively small burst lengths are needed to start to begin to realise the maximum bandwidth.

We don't ever burst; what is the maximum bandwidth we can expect to achieve?

The limiting factor here will be the minimum row open time.

If our SDRAM is being clocked at 133MHz and each location is 4 bytes wide our optimal bandwidth is 532MB/s. If only one location is accessed every time a row is opened i.e. SDRAM_RD_READY or SDRAM_WR_READY is high but the appropriate burst signal is kept low then the row will be closed after every access having satisfied the minimum row open time. Our time for each access will now be 9 cycles so our bandwidth 532 / 9 = 59MB / s.

We burst out data in groups of two, what bandwidth can we expect?

If we burst out data this figure will rise rapidly even getting just two pieces of data out each access will increase our bandwidth to 118MB/s.

(9 cycles minimum row open time) / (2 pieces of data) = 4.5

532MB/s / 4.5 = 118MB/s

We increase our burst length, what bandwidth can we expect?

If we use a burst length of 16 our bandwidth is up to 387MB/s;

(16 pieces of data + 6 cycles of overhead) / (16 pieces of data) = 1.375

532MB/s / 1.375 = 387MB/s

- It is important to use the burst ability of the SDRAM in order to achieve a high bandwidth between the FPGA and SDRAM. Failing to do this will result in very poor performance.
- The document 'Accessing SDRAM in you FPGA design' describes in detail how to use the signals provided in the userAp.vhd to efficiently control the SDRAM.
- This method of calculating bandwidth do not take into account that a refresh will be automatically done by the hardware interface layer and this will take time during which the SDRAM can not be used. However the frequency and time this operation takes to complete will be very insignificant.

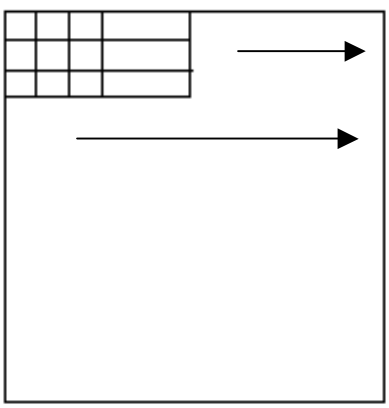
Format of stored Data in SDRAM

It would be very simple to store data in the SDRAM one pixel to each address. Each location in the SDRAM store is 32 bits wide. As most camera data is 8 bits wide it is possible and much more efficient to 'pack' 4 pixels in each address space. This will give benefits with respect to both the number of frames that can be stored in the SDRAM but will also cut the required bandwidth by 4. The numbers calculated under the heading 'method A' assume that this process will take place, if it doesn't then the bandwidth required will be 4 times higher.

Data Supply

One of the key issues in the implementation of any convolution function is that we have available the values of the required pixels in the correct order. The source of data for the convolution function will be one of the deciding factors in how a data supply component is built. The two principal sources for the data are likely to be either from a camera directly or from an image stored in the SDRAM built into some of the HERON – FPGA modules.
If data is sourced from a camera directly we need to consider what this implies. Each pixel will only be available once so this will require that a storage element will be needed at some point, and the pixels will be presented in order from the top left to the bottom right of an image.
If data is sourced from SDRAM then we have the possibility of accessing each pixel more than once but it is important that the SDRAM is accessed efficiently. In order to access the SDRAM data must be burst in and out. A burst is when data is accessed from subsequent locations in memory. For a more in depth discussion see the document, 'Accessing SDRAM in you FPGA design'.
As a result of these initial conditions there are at least three ways of supplying data to convolve a frame that are immediately obvious.

Method A



In method A blocks of data from each of the three lines currently required are temporally stored while the maths that uses them is performed. The minimum length that has to be read from each line is going to be 3 for a 3x3 convolution, 5 for a 5x5 convolution etc. If only 3 values were read from SDRAM this would be very inefficient and therefore more values will need to be read from each line. The other extreme is that an entire line of data is read but this is the same as method A and therefore has the same disadvantages.

- Advantages
  - Uses less block RAM than method B
  - Results are output in the same order they are scanned in by the camera therefore compatible with the pixel pipeline format
- Disadvantages
  - Greatly increases the amount of SDRAM bandwidth used

- Can't take data direct from a camera

Using this method we may be able to use distributed RAM rather than block RAM as so little storage is required. The disadvantage with this method is that each pixel in all but the first and last lines will have to be accessed three times as the point is there is not enough storage available to store a whole line. This may not be a problem if this is the only process using the SDRAM. For an example frame of 1024 x 1024 with 8 bit pixel data and a frame rate of 25 frames/sec the total bandwidth required will be:

1024 x 1024 x 8 x 25 = 209 715 200 bits / sec or 200Mbits / sec

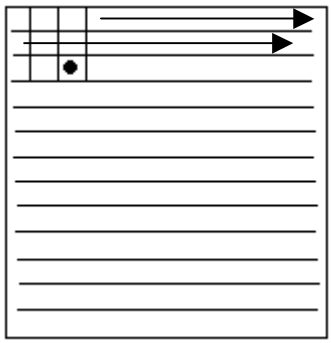
The optimal SDRAM bandwidth available is:

133Mhz x 32 bits = 4256000000 bits / sec or 4058Mbits / sec

This optimal bandwidth will be impossible to achieve over a sustained period of time but is 20 times that required to meet the 200Mbit / sec required above. As has already been discussed, this method will require 3 times the normal bandwidth dictated by the frame size, pixel depth and frame rate but even 600Mbits / sec is very achievable given the SDRAM bandwidth available.

Method A may, in certain circumstances, be a useful yet inefficient way of supplying the data for convolution.

Method B



In this method each full line is stored. The first result can't be calculated until the third pixel of the third line has been read in as indicated by the black dot.

- Advantages
  - Suits the format that data is sent in by the camera but would also work well with SDRAM
  - Fits in nicely with the Hunt Engineering Pixel Pipeline Format
- Disadvantages
  - Uses significant amounts of block RAM

To put a number on the amount of block RAM used, if we have a line length of 1024 and to make life easy we store 3 lines of 8 bit pixel data. That equates to:

- 1024 x 3 x 8 = 24576 bits of storage required.

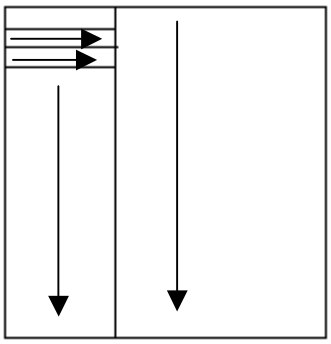
The closest block RAM configuration to that which we require is in 2K x 9 bits configuration. We can therefore see that we will need 1.5 block RAM's. The XST has been in fact found to allocate one block RAM to each line so even though they are not fully utilised 3 block RAM's will be used on any line length up to 2K.

If we take an extreme example:

- 4K pixel lines we will still only use 6 block RAM's to perform a 3x3 convolution rising to 10 for a 5x5 convolution.

In most cases this method will be the best to use. It requires that the number of lines involved in the neighbourhood processing be read into BLOCK RAMS from the SDRAM, but usually this will not be a problem for the FPGA design. In cases where the number of BLOCK RAMS used by this method is not acceptable either method A or C will need to be used.

Method C



Method C uses the same idea as method A in reducing the amount of block RAM required by storing small blocks of data but also reduces the amount of SDRAM bandwidth required to a level marginally above that required for method B. The idea is that initially 3 blocks of data are read from 3 subsequent lines. The convolution answers for the middle line are calculated and then the fourth line overwrites the first line and the answers calculated for the third line etc. Once the strip of data the width of a block has been processed for the whole height of the image. The strip next to it is then processed in the same way. It is important to note that the data that will have to be read from the SDRAM will have to overlap with that read for the previous strip by 2 for a 3x3 convolution, 3 for a 5x5 convolution etc.

- Advantages
  - Less block RAM used than in method B
  - Less SDRAM bandwidth used than in A
- Disadvantages
  - Answers emerge in an unusual order and therefore not compatible with the pixel pipeline format

This is the most elegant solution although it is not without its disadvantages. It would be most useful where the image was being taken from SDRAM and then being sent either to a HERON FIFO or back to SDRAM. Indeed if being returned to SDRAM the answers to the convolution could easily be reorder in to a top-left to bottom-right format just by setting the SDRAM address correctly. The major disadvantage is that this method of supplying the data to the convolution engine is not capable of being directly connected to a camera and has to get its data from a store such as the SDRAM.