# Acceleration of the Retinal Vascular Tracing Algorithm using FPGAs

***Shawn Miller*** and ***Miriam Leeser***
Northeastern University
Phone: (617) 373-5294
Fax: (617) 373-8970
Email: {smiller, mel} @ ece.neu.edu

It has been shown that FPGAs can accelerate the computationally intensive tasks of image processing. Due to the large amount of data necessary for the computations and the delays involved in passing data between memory and the FPGA, it is a challenge to get the data to the FPGA in order to process it. We have designed a system which acts as a "smart camera" to provide an efficient way to get real-time results from streaming image data. We have a Dalsa 1M30P camera connected to a custom framegrabber from Dillon Engineering. The framegrabber is a daughter board to an Annapolis Microsystems Firebird PCI board, which contains a Virtex XCVE2000 FPGA and five banks of on-board memory. The image data is passed directly from the camera to the FPGA, where the image processing computations occur, and the results from those computations are output along with the original image data. This system is considered a "smart camera" because at frame rate, the host PC receives not only the original image data, but also the results from the image processing.

We have applied the “smart camera” idea to the Retinal Vascular Tracing (RVT) algorithm that was developed by at Rensselaer Polytechnic Institute [1]. There is a need in the biomedical field for a method of tracing blood vessels in retinal images. The speed of motion of the human eye and the desire for these traces to be available for real-time applications, such as assisting in laser surgery, require these traces to be available immediately after the retinal image is acquired. With an FPGA supplying the computationally intense image processing results, RVT can be applied to real-time image data.

The image processing that is required by the RVT algorithm is the application of the 16 matched filters shown in Figure 1 for edge detection. The responses for each filter must be calculated, and the filter with the highest response is chosen as the direction for that particular pixel. We programmed the FPGA to find the responses to each filter in parallel, and then using a binary comparator tree we found the greatest response and returned the direction that was chosen as well as the corresponding response to the host PC. Performing these calculations in hardware provides a speed up of approximately 30 times compared to the software solution.

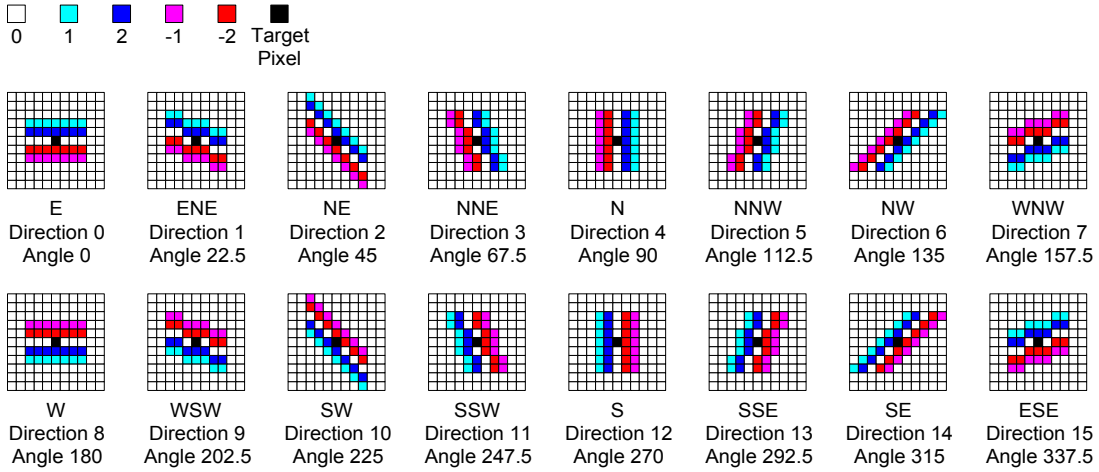


**Figure 1**
**Matched Filters**

Each set of matched filter response calculations requires an 11x11 pixel neighborhood. The camera outputs the pixel data in raster order, so before any computation can begin, eleven full rows of data need to be collected and stored in the Firebird’s on-board memory. The camera outputs 12-bit pixels, so to

take advantage of the memory's width five pixels are stored in one 64-bit word of memory with only four wasted bits.

Once eleven rows of data are available in memory, blocks of 11x15 pixels (11 rows containing 3 memory words) are read into registers on the FPGA. The block of data contains five consecutive 11x11 neighborhoods, and five results are computed. After an 11x15 pixel block of data is processed, five new columns of pixels (one more column of memory words) are read into the FPGA registers and the five oldest columns are overwritten to create a new 11x15 pixel neighborhood that is shifted to the right. This method is continued, and when the pipeline is full, five results are returned for every eleven memory reads.

To get the maximum efficiency from this design, the data from the camera must be available to be read from the Firebird's on-board memory. One problem is that the FPGA cannot read from the same bank of memory that is being written to concurrently. To solve this problem, two separate memories are used for storing the data. Every other word of data is stored in a different bank of memory so that while one memory is having new data written to it, the other memory is having data read from it. At the end of a frame, the new data overwrites the old data to avoid an overflow. This method guarantees that new data can always be written into memory, and that the correct data can be read from memory without interruption.

Every result that is found consists of the original target pixel (12 bits), the direction that returned the greatest response (4 bits), and the response that led to the direction selection (17 bits). These results are packed into a 64-bit word and stored into a different bank of memory on the Firebird. The host PC reads the results from this memory via the PCI bus, and they are available for the RVT algorithm. Figure 2 shows the entire layout of the "smart camera" design, from the input of data from the camera to the output of results to the host PC.

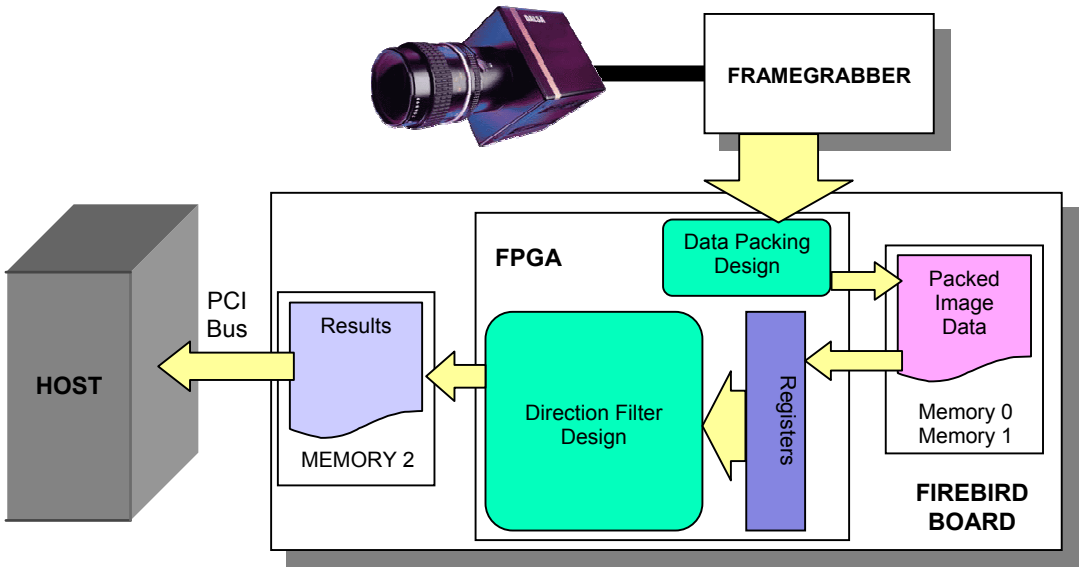


**Figure 2**
**Smart Camera Hardware Design**

The application for this design is for the RVT algorithm. The idea, however, can be applied to many other image processing algorithms. Any algorithm that requires image processing results and can accept a small latency can be sped up using the "smart camera" setup. The "Direction Filter Design" in Figure 2 can be replaced with different filters, or other image processing algorithms with similar improvements in performance.

[1] A. Can, H. Shen, J.N. Turner, H.L. Tanenbaum and B. Roysam, "Rapid Automated Tracing and Feature Extraction from Retinal Fundus Images Using Direct Exploratory Algorithms", IEEE Transactions on Information Technology in Biomedicine, June 1999.