

## Edge Detection in AT6000 FPGAs

### Introduction

Edge detection is of fundamental importance in image analysis. Edges characterize object boundaries, and are thereby very useful for registration, segmentation, and identification of objects in images. For example, an edge detector is commonly found in such applications as contour mapping and target recognition. Similar to some digital signal processing algorithms, edge detection involves two-dimensional convolution, a compute-intensive multiply-add operation. Numerous solutions such as powers-of-2s [1] and vector multipliers have been proposed to alternatively manipulate the multiply-add operation. In this paper, we will present a reference design of a fully pipelined bit-parallel edge detection circuit that, with a careful choice of convolver masks, utilizes only pipelined adders and fits into one Atmel AT6010 FPGA. We will also discuss how CacheLogic<sup>®</sup> optimizes performance of an edge detector operating on real-time video at run-time.

### Background

In the digital domain, an edge can be identified when an abrupt change in pixel intensity level occurs. Taking “derivatives” at each pixel location then determines the rate of change of pixel intensity. Based on this concept, two types of digital edge detection operators (masks) have been introduced: gradient masks and compass masks [2]. These masks represent finite-difference approximations of either the orthogonal gradients or the directional gradient. Here we will consider the most commonly used gradient masks—the Sobel masks.

Shown below are the Sobel masks. Convolving them with the input image is

analogous to taking “derivatives” to determine the orthogonal gradients at each pixel location.

$$H_1(m, n) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$H_2(m, n) = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Let  $U(m, n)$  denote the input image pixel array. Define the bi-directional gradients:

$$G_1(m, n) = U(m, n) \otimes H_1(-m, -n) \quad (1)$$

$$G_2(m, n) = U(m, n) \otimes H_2(-m, -n) \quad (2)$$

The gradient vector magnitude  $G(m, n)$  is:

$$G(m, n) = \sqrt{G_1(m, n)^2 + G_2(m, n)^2} \quad (3)$$

An edge pixel  $E(m, n)$  is defined if  $G(m, n)$  exceeds some threshold  $t$ , that is:

$$E(m, n) = \begin{cases} 1 & \text{for } G(m, n) > t \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Typically,  $t$  is chosen using the cumulative histogram of  $G(m, n)$  such that 5 to 10 percent of pixels with largest gradients are declared as edges [3]. The array  $E(m, n)$  is called the edge map of  $U(m, n)$ . For hardware implementation, the gradient vector magnitude can be calculated as:

$$G(m, n) = |G_1(m, n)| + |G_2(m, n)|$$

(continued)

## AT6000 FPGAs

### Application Note

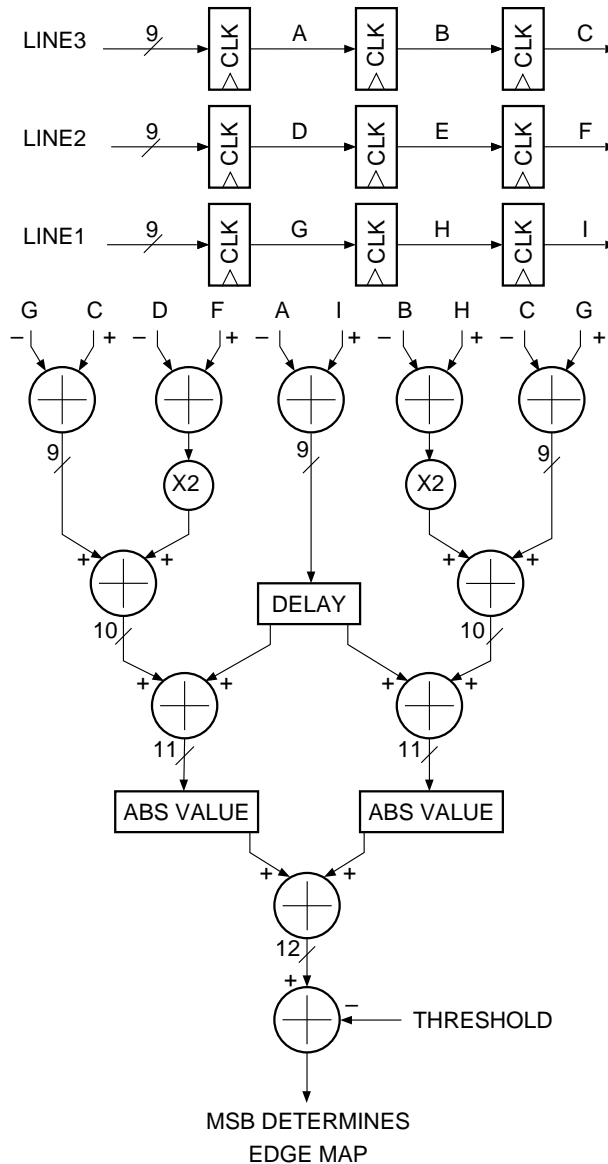
Obviously, accuracy is lost in this approximation. However, an appropriate upward adjustment of the threshold will compensate for the introduced error.

## Implementation

Figure 1 shows the block diagram of an edge detection circuit using Sobel masks operating on 8-bit grayscale images. The masks are not “flipped” prior to convolution as required by Equations (1) and (2). Flipping only affects the directional sense of the gradients. Because the masks only contain values of 0,  $\pm 1$ , and  $\pm 2$ , complex convolution can be replaced by simple algebraic manipulation. Full precision is kept in all stages.

Fully-pipelined macros generated by Atmel Integrated Development System are used. All multiplications are replaced by pre-subtractions, shifting, and additions. When comparing the gradient magnitude against the threshold, a fully-pipelined subtractor is used instead of a comparator because the former runs at much higher speed. Edges are then determined by the sign bit of each outcome. For other edge detection masks that do not allow for algebraic manipulation of complex multiplication, vector multipliers may be used. For more information, please refer to Atmel's Application Note: 3x3 convolver with run-time reconfigurable vector multiplier in Atmel AT6K FPGAs.

**Figure 1.** Block diagram of the edge detector



## Results

Figures 2 and 3 show results of software emulation of our reference hardware design. More pixels are declared as edge points using the “absolute value” gradient calculation than the original “square root” method when  $t$  is 70. More distinct edge characteristics are manifested when  $t$  is increased to 150. Thus, adjusting the threshold compensates for the gradient magnitude approximation error. All stages of our reference design are fully pipelined. Table 1 shows some vital statistics.

**Table 1.** Reference design statistics

|                            |         |           |
|----------------------------|---------|-----------|
| Throughput frequency (MHz) | 86.2069 | 86.2069   |
| Grayscale Quantization     | 8-bit   | 8-bit     |
| Resolution                 | 256x256 | 1280x1024 |
| Frames per second          | 1315.41 | 65.77     |

## Role of CacheLogic

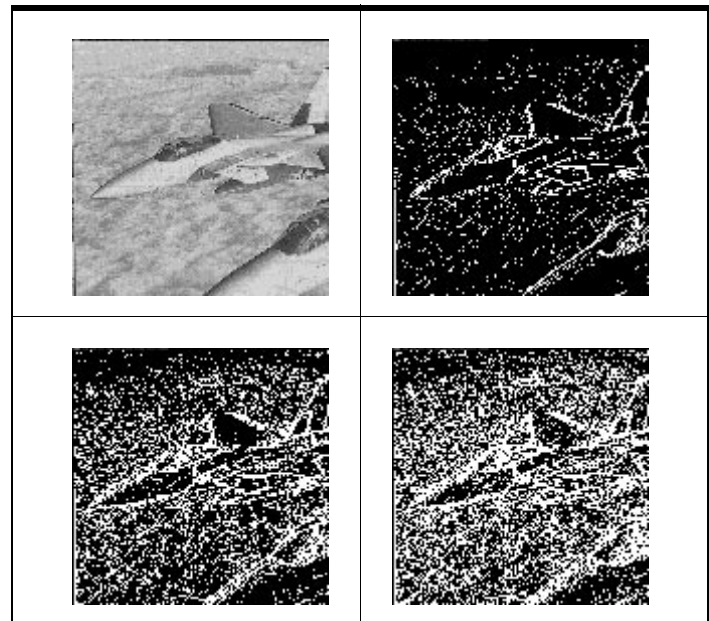
In real-time imaging systems that do not tolerate large buffering and clock latency, it is desirable for an FPGA to have the ability to partially reconfigure parameters at run-time without redundantly reconfiguring the rest of the logic circuitry. This is especially true when many parameters become variables in a system. Some typical solutions with their drawbacks are as follows:

- The use of more I/O pins is only effective when the number of variables is small. For color imaging systems where R/G/B operations have to be separated, we may need more I/O pins than are available on a single-chip. Multi-chip solutions may not be cost effective for certain applications.
- Input multiplexing introduces clock latencies and complicates internal place and route of logic cells during compilation. This has major impact on bit-serial systems where the throughput frequency highly depends on the total latency to produce one output sample.
- On/off-chip memory elements to store parameters do not provide enough versatility to particular systems. Buffers can only hold a limited amount of information, thereby limiting the range of parameters to be stored. Keeping pointers to memory buffers can also be tedious.

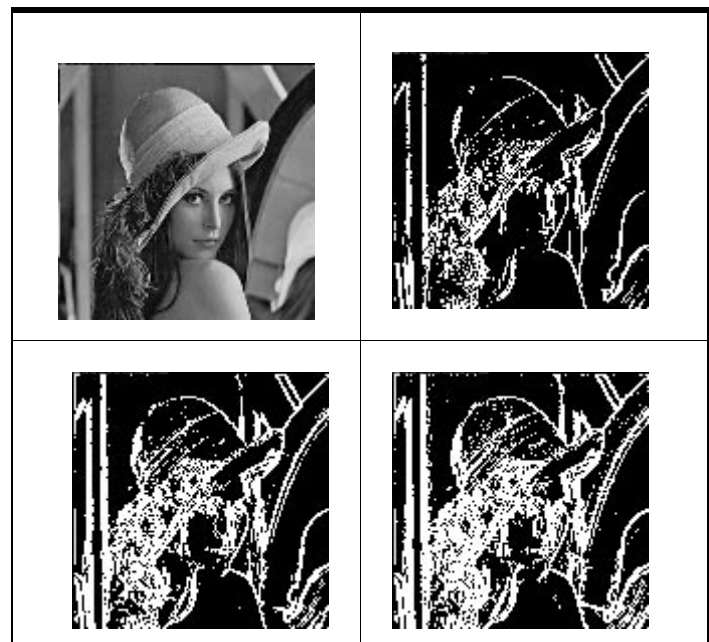
In edge detection, often times a particular set of masks do not give optimal results because image statistics are not always available beforehand (e.g., real-time video edge detection). For example, the Sobel masks are only effective in detecting abrupt changes in intensity. For smoother transition of intensity, other methods or masks must be employed. Edges also have directional properties that are best revealed by other masks. It would then be beneficial to have an adaptive environment where parameters can be reconfigured at run-time without having such constraints as I/O pins, partitioning, or buffering.

|   |   |
|---|---|
| Original 256x256 8-bit grayscale image                      | Edge map with $t=150$ using “absolute value” gradient magnitude |
| Edge map with $t=70$ using “square root” gradient magnitude | Edge map with $t=70$ using “absolute value gradient magnitude   |

**Figure 2.** F15 and its edge maps



**Figure 3.** LENA and its edge maps



## QuickChange

CacheLogic solves the above problems by featuring run-time reconfiguration of Atmel FPGAs. In support of the CacheLogic capability, Atmel has developed QuickChange™, a multi-parameter specification software tool that allows users to interactively specify multiple parameters for digital filters, convolvers, and other compute-oriented hardware. After completing the design with an initial set of parameters, the designer simply invokes QuickChange from the Atmel design environment. QuickChange searches the design for DSP coefficients or other parameters, logically groups them and displays them in a graphical user-interface. The designer can interactively specify as many replacement sets of parameters as desired. The QuickChange tool then generates the FPGA bitstreams for each new parameter or sets of parameters. These small bitstream files partially reconfigure the FPGA without affecting the operation of the existing logic.

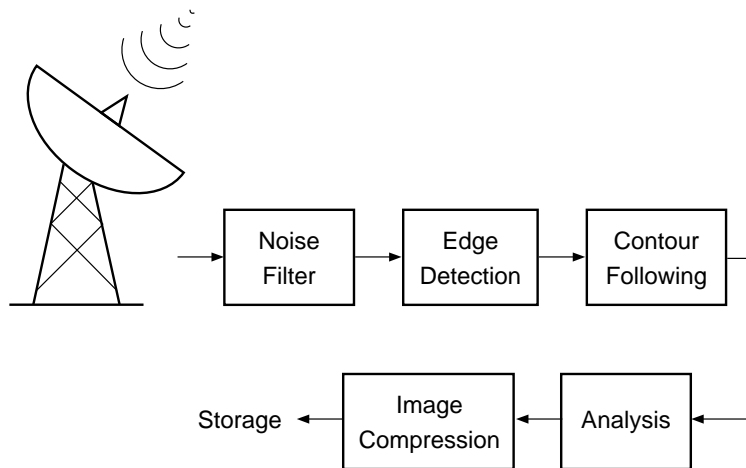
For the case of an edge detector using vector multipliers to perform multiplication during convolution, CacheLogic downloads pre-computed look-up tables (LUTs), and the existing edge detector circuit will be in full action with new parameters within a few clock cycles.

Besides reconfiguring parameters, CacheLogic also offers solutions to low-budget procedural applications. In areas where speed is not of the main concern, CacheLogic can reconfigure the FPGA to perform different functions along a datapath. The major advantage is hardware minimization. Figure 4 depicts a typical edge detection procedure. Source information from a receiver usually needs filtering and processing before being meaningfully stored in a storage medium. At each stage, CacheLogic reconfigures the FPGA to perform a specific function. Temporary results are stored in a buffer that can be made internal depending on its size. This buffer constantly exchanges information with the FPGA until the last function is performed (see Figure 5).

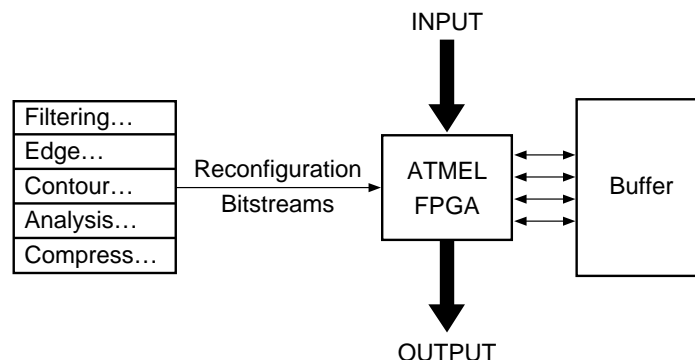
## Summary

It has been shown that, with a careful choice of edge masks, a fully pipelined edge detector can be implemented on one single Atmel FPGA. Increased versatility is possible when CacheLogic comes into play. With hardware reconfiguration, more tasks can be performed in a designated number of FPGAs, thereby maximizing hardware usage and versatility while keeping the cost down to a minimum.

**Figure 4.** Typical edge detection procedure



**Figure 5.** Reconfiguration using CacheLogic



## References

[1] Joseph B. Evans, "Efficient FIR Filter Architectures Suitable for FPGA Implementation," Telecommunications & Information Sciences Laboratory, Department of Electrical & Computer Engineering, University of Kansas, Lawrence, KS 66045-2228.

[2] Anil K. Jain, "Fundamentals of Digital Image Processing," Prentice-Hall, Inc., 1989.

[3] As [2] above.