Adaptive Edge Detection for Real-Time Video Processing using FPGAs

Hong Shan Neoh
Altera Corporation
101 Innovation Dr.
San Jose, CA 95134
(408) 544 7000
hneoh@altera.com

Asher Hazanchuk
Altera Corporation
101 Innovation Dr.
San Jose, CA 95134
(408) 544 7000
ahazanch@altera.com

I. Introduction

Real-time video and image processing is used in a wide variety of applications from video surveillance and traffic management to medical imaging applications. These operations typically require very high computation power. Standard definition NTSC video is digitized at 720x480 or full D1 resolution at 30 frames per second, which results in a 31MHz pixel rate. With multiple adaptive convolution stages to detect or eliminate different features within the image, the filtering operation receives input data at a rate of over 1 giga samples per second. Coupled with new high-resolution standards and multi-channel environments, processing requirements can be even higher. Achieving this level of processing power using programmable DSP requires multiple processors. A single FPGA with an embedded soft processor[1] can deliver the requisite level of computing power more cost-effectively, while simplifying board complexity.

This paper presents the implementation of an adaptive edge-detection filter on an FPGA using a combination of hardware and software components. The FPGA provides the necessary performance for real-time image and video processing, while retaining the system flexibility to support an adaptive algorithm. Preliminary results are presented for this system and evaluated with respect to the Canny edge detector as a benchmark.

II. Edge Detection Background

Edge detection is a fundamental tool used in most image processing applications to obtain information from the frames as a precursor step to feature extraction and object segmentation. This process detects outlines of an object and boundaries between objects and the background in the image. An edge-detection filter can also be used to improve the appearance of blurred or anti-aliased video streams.

The basic edge-detection operator is a matrix-area gradient operation that determines the level of variance between different pixels. The edge-detection operator is calculated by forming a matrix centered on a pixel chosen as the center of the matrix area. If the value of this matrix area is above a given threshold, then the middle pixel is classified as an edge. Examples of gradient-based edge detectors are Roberts, Prewitt, and Sobel operators.
All the gradient-based algorithms have kernel operators that calculate the strength of the slope in directions which are orthogonal to each other, commonly vertical and horizontal. Later, the contributions of the different components of the slopes are combined to give the total value of the edge strength.

The Prewitt operator measures two components. The vertical edge component is calculated with kernel Kx and the horizontal edge component is calculated with kernel Ky. |Kx| + |Ky| gives an indication of the intensity of the gradient in the current pixel.

$$Kx = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad Ky = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

Figure 1: Prewitt horizontal and vertical operators

Depending on the noise characteristics of the image or streaming video, edge detection results can vary. Gradient-based algorithms such as the Prewitt filter have a major drawback of being very sensitive to noise. The size of the kernel filter and coefficients are fixed and cannot be adapted to a given image. An adaptive edge-detection algorithm is necessary to provide a robust solution that is adaptable to the varying noise levels of these images to help distinguish valid image content from visual artifacts introduced by noise.

III. Canny Edge Detection Algorithm

The Canny algorithm uses an optimal edge detector based on a set of criteria which include finding the

[1] An embedded soft processor refers to a reconfigurable processor which resides on the FPGA fabric.

most edges by minimizing the error rate, marking edges as closely as possible to the actual edges to maximize localization, and marking edges only once when a single edge exists for minimal response [1]. According to Canny, the optimal filter that meets all three criteria above can be efficiently approximated using the first derivative of a Gaussian function.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1)$$

$$\frac{\partial G(x,y)}{\partial x} \alpha\, xe^{-\frac{x^2+y^2}{2\sigma^2}} \qquad \frac{\partial G(x,y)}{\partial y} \alpha\, ye^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2)$$

The first stage involves smoothing the image by convolving with a Gaussian filter. This is followed by finding the gradient of the image by feeding the smoothed image through a convolution operation with the derivative of the Gaussian in both the vertical and horizontal directions. The 2-D convolution operation is described in the following equation.

$$\begin{aligned} I'(x,y) &= g(k,l) \otimes I(x,y) \\ &= \sum_{k=-N}^{N} \sum_{l=-N}^{N} g(k,l) I(x-k, y-l) \end{aligned} \quad (3)$$

where: *g(k,l)* = convolutional kernel
I(x,y) = original image
I'(x,y) = filtered image
2N + 1 = size of convolutional kernel

Both the Gaussian mask and its derivative are separable, allowing the 2-D convolution operation to be simplified. This optimization is not limited to software implementation only, but applies to hardware implementation as well, as shown in the next section.

The non-maximal suppression stage finds the local maxima in the direction of the gradient, and suppresses all others, minimizing false edges. The local maxima is found by comparing the pixel with its neighbors along the direction of the gradient. This helps to maintain the single pixel thin edges before the final thresholding stage.

Instead of using a single static threshold value for the entire image, the Canny algorithm introduced hysteresis thresholding, which has some adaptivity to the local content of the image. There are two threshold levels, $t_{h,}$high and $t_{l,}$ low where $t_h > t_l$. Pixel values above the t_h value are immediately classified as edges. By tracing the edge contour, neighboring pixels with gradient magnitude values less than t_h can still be marked as edges as long as they are above t_l. This process alleviates problems associated with edge discontinuities by identifying strong edges, and preserving the relevant weak edges, in addition to maintaining some level of noise suppression. While the results are desirable, the hysteresis stage slows the overall algorithm down considerably.

The performance of the Canny algorithm depends heavily on the adjustable parameters, σ, which is the standard deviation for the Gaussian filter, and the threshold values, t_h and t_l. σ also controls the size of the Gaussian filter. The bigger the value for σ, the larger the size of the Gaussian filter becomes. This implies more blurring, necessary for noisy images, as well as detecting larger edges. As expected, however, the larger the scale of the Gaussian, the less accurate is the localization of the edge. Smaller values of σ imply a smaller Gaussian filter which limits the amount of blurring, maintaining finer edges in the image. The user can tailor the algorithm by adjusting these parameters to adapt to different environments with different noise levels.

Results can be further improved by performing edge detection at multiple resolutions using multi-scale representations, similar to the Marr-Hildreth algorithm [2]. This is achieved using different standard deviations, which correspond to different resolution versions of the image. Edges have zero crossing at multiple scale values. Combining maxima information from different scales allows better classification of true edges.

Convolution at multiple resolutions with large Gaussian filters require even more computation power. This may prove to be challenging to implement as a software solution for real-time applications.

IV. Processing requirements

Real-time image processing requires high computation power. For example, the NTSC video standard requires 30 frames per second, with approximately .25 mega pixels per frame, resulting in a total of 7.5 mega pixels per second. PAL video standard has a similar processing load with 25 frames per second, but the frame size is larger. The amount of processing required per pixel depends on the image processing algorithm.

In the case of the Canny edge-detection algorithm the processing requirement is a combination of the four stages of Gaussian smoothing for noise reduction, finding zero crossings using the derivative of Gaussian, non-maximal suppression to thin the edges, and finally hysteresis thresholding.

Depending on the size of the kernels, the processing requirements for the first two convolution stages can change. Using the assumption of a kernel size of 7*7 for the Gaussian and 5*5 for the derivative of the Gaussian, the two stages require approximately 62 operations per pixel. This implementation requires less complexity than most typical 2-D convolution processes because of the symmetric and separable characteristics of the Gaussian masks. The non-maximal suppression stage requires 27 operations per pixel. Due to the recursive nature of the hysteresis thresholding process, this stage requires an additional 40 operations per pixel. In total, the Canny algorithm requires approximately 130 operations per pixel. Coupled with running at multiple resolutions, serial implementation of the algorithm using a programmable DSP with a clock speed of 600MHz can only process 4.6 mega pixels per second. This is not sufficient to support real-time video streams at high resolution.

As high-resolution video standards become more prevalent, the processing requirements will increase even further. The high-resolution video standards typically have 4 times more pixels per frame. The computation load, therefore is approximately 4 times higher. These image processing applications with high computation loads can be implemented with multiple DSPs or a single, very expensive high-end DSP. In this scenario, FPGAs offer an alternative real-time image processing platform. An FPGA efficiently supports high levels of parallel-processing data-flow structures, which are important for efficient implementation of image processing algorithms.

V. Hardware Implementation on an FPGA

The FPGA implementation of the Canny edge-detector algorithm is partitioned into four different stages as shown in Figure 2.

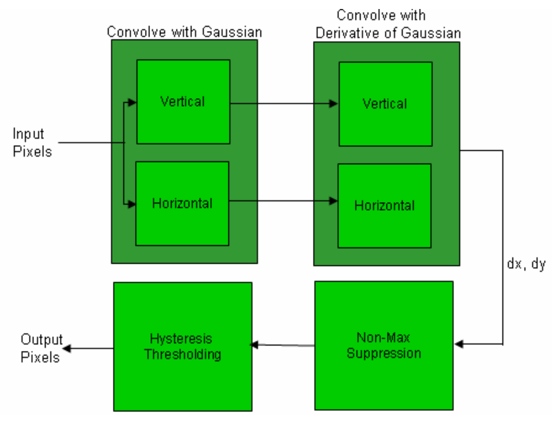

Figure 2 Block diagram of hardware implementation of Canny algorithm

Figure 3 describes the implementation of a generic non-symmetric 2-D image filter. The incoming pixels are shifted through the line buffers that create a delay line. The buffer depth depends on the number of pixels in each line of the frame. These delay lines feed the filter array simultaneously with pixels from all the relevant video lines. At each filter node, the pixel is multiplied with the appropriate filter coefficients. All the multiplier results are added together at the adder tree to produce the filter middle point output result. Typically, scaling is applied at the final output.

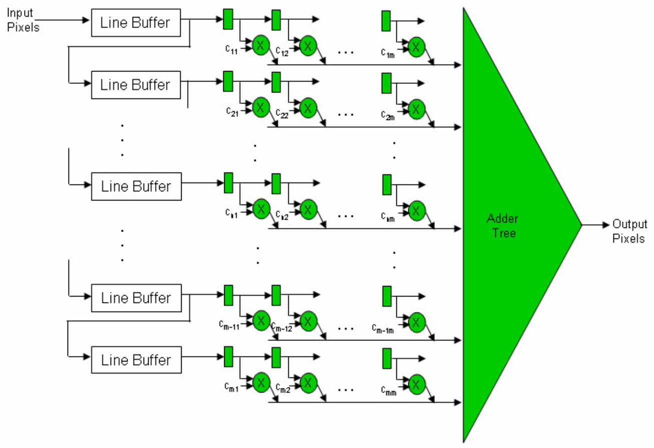

Figure 3 Hardware Implementation of non-symmetric 2D filter

In general, the efficiency of a hardware implementation is measured by the number of multiplications. Using this metric as the measure, the complexity of the non-symmetric filter is proportional to the dimension of the filter $\boldsymbol{m^2}$, where $\boldsymbol{m * m}$ is the size of the convolutional kernel.

Significant size optimization can be achieved in the case of symmetric 2-D video filters. The optimization is even more significant in cases where the filter kernel is symmetric and separable, as is the case with the Gaussian filter used in the Canny algorithm. The 2-D filter is separated into two 1-D filters implemented first in the horizontal direction, followed by the vertical direction as shown in Figure 4. This technique can be applied to both the smoothing stage with the Gaussian filter, and the identification of the gradient with the derivative of the Gaussian. Table 1 shows the implementation complexity of the different class of filters as a function of number of multiplication operations.

Filter Size	**Non-Symmetric**	**Symmetric**	**Symmetric and Separable**
m * m	m * m	(m+1)*(m+3)/8	m + 1

Table 1: 2-D Filter implementation complexity based on number of multiplication operations

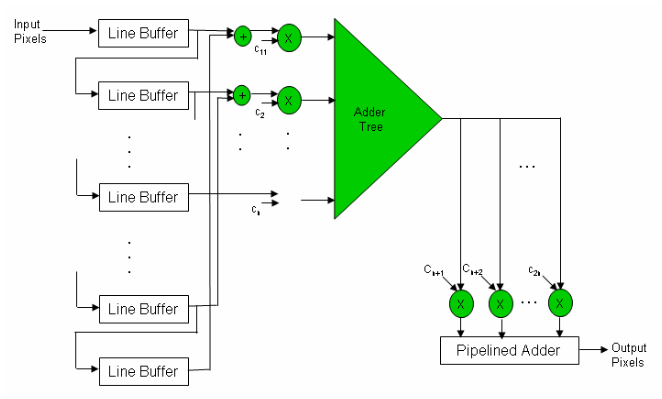

Figure 4 Hardware implementation of symmetric separable 2D filter

Since the convolution for both the Gaussian and its derivative can be implemented as separate 1-D convolutions, the operations are implemented in parallel on the FPGA as shown in Figure 2.

The noise reduction function is performed using a 7*7 Gaussian kernel. This kernel slides over the image, line by line, attenuating the high-frequency noise components of the image. The finding of zero crossings is performed using a 5*5 kernel obtained by calculating the derivative of Gaussian.

The non-maximal suppression stage identifies pixels that are local maxima in the direction of the gradient using the magnitude and orientation of the pixels. The major orientation of the gradient, either horizontal or vertical, is obtained by comparing the individual components, ***dx*** and ***dy***, which are the result of convolving the smoothed image with the derivative of the Gaussian. Since most edges are at an angle, it is possible to obtain further granularity in the orientation of the gradient by comparing the sign bit of the gradient [3]. This allows the neighboring pixels and the orientation of the gradient to be determined within specific quadrants as shown in Figure 5.

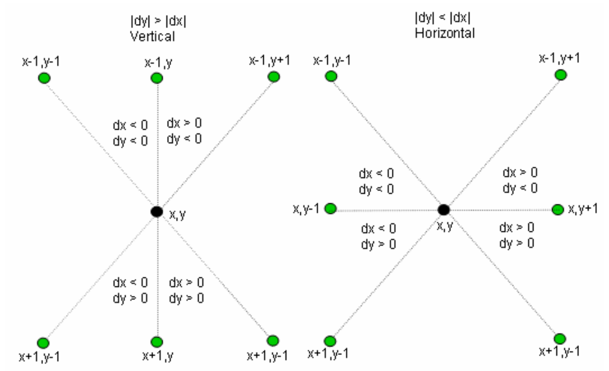

Figure 5: Determine orientation of gradient in the non-maximal suppression stage

Once the orientation is determined, the magnitude of the gradient is compared to the interpolated value of the magnitude of the neighboring pixels.
The magnitude of the neighboring pixels is calculated using parallel multiplier units. Pixels with gradient intensities less than their neighboring pixels in the gradient direction are automatically suppressed or classified as a non-edge in order to keep the edges thin.

The final stage involves thresholding the result from the non-maximal suppression stage to create a binary image using two threshold values (hysteresis): t_h and t_l. A straightforward approach would include running a first pass over the video frame to compare all the pixels with the two threshold values. Pixels with gradient values above t_h are marked '2', classified as an edge. Next, pixels with gradient values less than t_h but above t_l are marked '1', classified as a potential edge. The rest of the pixels are left as zeros, or non-edge. Figure 6 shows an example of this frame mapping approach.

For each pixel marked as '2', its neighboring pixels within a 3x3 neighborhood that are marked '1' are remarked '2'. If implemented in software, this section of the algorithm alone can contribute to more than 20 operations per pixel. Additional passes are required to find potential edges that were left behind in the initial passes. Edges are left behind in cases where there is a line of '1's that leads to '2' in the direction of the pass. Figure 6 shows an example with the assumption that the lines are processed from left to right starting at the top. In this scenario, all the '1's in the second row would be remarked as '2' only in subsequent passes. The number of passes required depends on the characteristics of each video frame.

0	0	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
0	0	0	0	0	0	0	0	**1**	0
0	0	0	**1**	**2**	**2**	0	0	**1**	0
0	0	0	0	0	**2**	0	0	**1**	0
0	0	**1**	0	0	0	0	0	**1**	0
0	0	**1**	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	**2**	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

2 - Edge
1 - Potential Edge
0 - Non-edge

Figure 6: Edge strength classification map

Since the number of conversions from '1' to '2' is always less than the total number of pixels in the frame, this upper limit guarantees that operations that depend on the '2' locations list FIFO can be completed during the real-time processing period of a frame. The

FIFOs are implemented using multiple M4K embedded memory blocks, which are of size 4Kbits each.

At the first pass, the edge-detection result map (‘0’, ‘1’, or ‘2’ values) is stored in the embedded RAM blocks, or MRAMs, in the Stratix or Stratix II device. These large embedded RAM blocks of 589-Kbit size can be used to temporarily store the edge map where 2 bits are allocated for each pixel location.

During each pixel clock cycle, the following operations are executed simultaneously in the FPGA to achieve real-time performance:

Compare gradient value with t_h
Compare gradient value with t_l
Depending on the previous comparison results, write ‘2’, ‘1’, or ‘0’ into the next pixel location
If ‘2’ is being written, then write this address location into the ‘2’ locations list FIFO
If the ‘2’ locations list FIFO is not empty then read the next address from the FIFO and convert each ‘1’ neighbor pixel to ‘2’
Add all the new converted into ‘2’ locations to the ‘2’ locations list FIFO

It is possible to extend this design architecture to support multiple resolutions for processing the image at different levels of details across scales. A software approach can be used to initiate and control the data flow, while the hardware implementation is used to accelerate the repetitive tasks.

When combining hardware and software in the solution, it is possible to keep the implementation of the entire system on a single FPGA device using the Nios II embedded processor. The Nios II embedded processor features a general-purpose RISC CPU architecture. This soft processor can be reconfigured to a different set of peripherals, memories, interfaces, and performance characteristics using the SOPC Builder tool featured in the Quartus II design software. The hardware accelerator block is connected to the Nios II processor as a peripheral using the Avalon switch fabric which is generated automatically by SOPC Builder [4].

VI. Implementation Analysis and Preliminary Results

The architecture described above is implemented using DSP Builder, a development tool that interfaces between the Altera Quartus II design software and MATLAB/Simulink tools. The tool automatically translates the Simulink design representation created using the DSP Builder blockset into VHDL targeting the Altera FPGA.

Figure 7 compares the edge-detection results of the Canny algorithm and the standard Prewitt filter for noise-free and noisy images with different noise levels. It shows the Canny edge detector is able to keep the edges one pixel thin as a result of the non-maximal suppression process. It also performs better at suppressing white Gaussian noise, while maintaining some level of accuracy at detecting edges compared to the Prewitt filter.

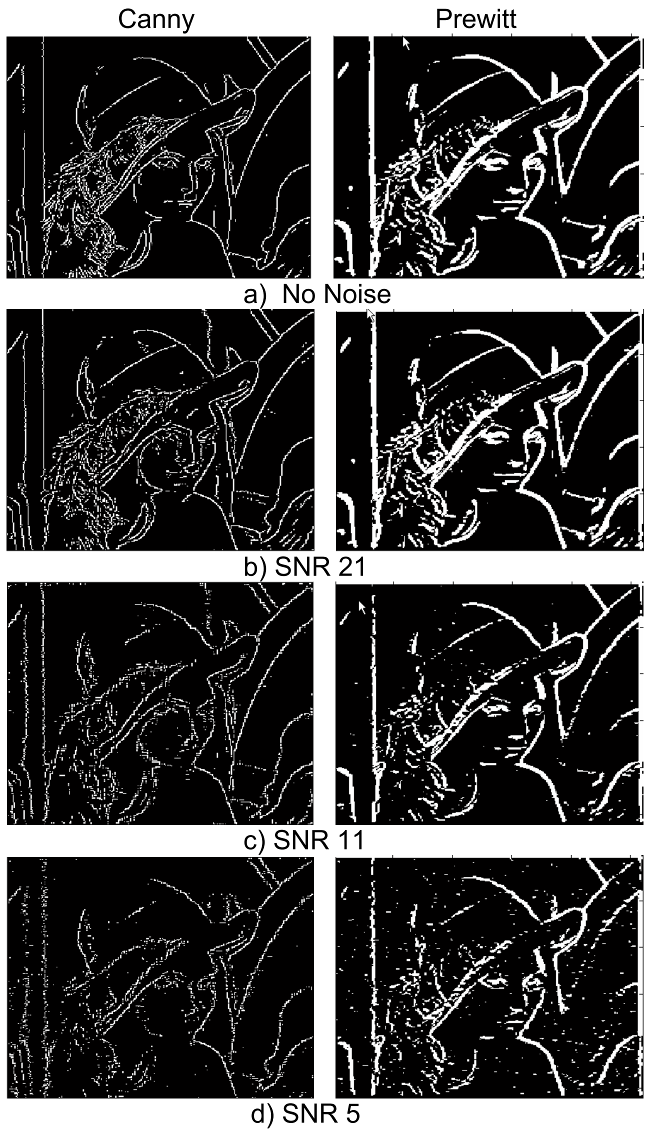

Figure 7: Comparing Canny and Prewitt edge detector results at different noise level

Preliminary results show the FPGA design running at 264MHz targeting Stratix II. We used a test image with a resolution of 256x256. The module is fully pipelined

[2]where a resulting pixel is calculated every clock cycle. With this throughput and clock rate, it is possible to process over 4000 frames of images at the 256x256 resolution. The design is scalable to handle higher resolution images, while maintaining the clock frequency used to process standard video-resolution signals at 30 frames per second, as well as high-speed computer vision applications, which require more than 100 frames per second.

It is possible to increase the frequency beyond 264MHz by introducing even more pipeline stages at the expense of increasing resource usage. The logic cell usage of 1530 adaptive look-up tables (ALUTs) consumes approximately 3% of the total utilization of the Stratix II EP2S60 device. Also shown are the resource utilization numbers targeting the Stratix EP1S20 device.

Stage	ALUT	M4K	MRAM	DSP Element[2]
Gaussian Smoothing	320	7	0	16
Derivative of Gaussian	250	5	0	12
Non-Maximal Suppression	800	3	0	15
Hysteresis Thresholding	160	8	2	0
Total	1530/48352	23/255	2/2	43/288
Percentage	3%	9%	100%	15%

Table 2: FPGA Resource usage based on Canny algorithm targeting Stratix II EP2S60

Stage	Logic Cell	M4K	MRAM	DSP Element[2]
Gaussian Smoothing	500	7	0	16
Derivative of Gaussian	300	5	0	12
Non-Maximal Suppression	1200	3	0	15
Hysteresis Thresholding	200	8	2	0
Total	2200/18460	23/82	2/2	43/80
Percentage	15%	28%	100%	54%

Table 3: FPGA Resource usage based on Canny algorithm targeting Stratix EP1S20

If a combined hardware and software solution is required using the Nios II processor, an additional count of 1300 logic cell is required. Depending on the set of peripherals chosen and performance requirement for the processor, the amount of additional logic required may vary.

[2] DSP Element refers to a 9x9 multiplier structure within the DSP Block[3]. There are 8 DSP Elements per DSP Block.
[3] DSP Blocks are dedicated circuitry which offers multipliers, accumulators, adders, subtractors, summation units, and pipeline registers.

VII. Conclusion

This paper discussed the implementation of the Canny edge-detection algorithm on an FPGA. We presented preliminary results, which show the capability of supporting adaptive edge detection for real-time image processing.

Further improvement in the edge-detection research area has resulted in a broad range of evaluation techniques, which include using neural networks, probabilistic models [5], and automatic scale selection [6]. Another class of gradient operators attempt to match image functions to a parametric model of edges. Parametric models describe edges more precisely than simple edge magnitude and direction and are much more computationally intensive. While these advanced algorithms provide better results, a common challenge among these techniques is the increasing computational cost as the complexity of the algorithm increases. FPGAs are good alternatives, which can be used to off-load these computationally-intensive and repetitive functions as co-processors. Custom implementation in FPGAs, using a combination of hardware and software co-design, allows real-time processing, providing a good trade-off between performance and flexibility.

References

[1] Canny, J., "A Computational Approach to Edge Detection", *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8:679-714, November 1986
[2] Maar, D., Hildreth E., "Theory of edge detection", *Proceedings Royal Soc. London*, vol. 207, 187-217, 1980
[3] Advanced Edge Detection Technique: Techniques in Computational Vision: http://www.cpsc.ucalgary.ca/Research/vision/501/edgedetect.pdf
[4] AN333: Developing Peripherals for SOPC Builder: http://www.altera.com/literature/an/an333.pdf
[5] Marimont, D., Rubner Y., "A Probabilistic Frameword for Edge Detection and Scale Selection", *Proceedings of the IEEE International Conference on Computer Vision*, 207-214, January 1998
[6] Lindeberg T., "Edge Detection and Ridge Detection with Automatic Scale Selection", *International Journal of Computer Vision,* vol. 30, number 2, 117-154, 1998

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
literature@altera.com