

# HARDWARE EDGE DETECTION USING AN ALTERA STRATIX NIOS2 DEVELOPMENT KIT

Jay Kraut  
Dept. ECE  
University of Manitoba  
Winnipeg, Manitoba, R3T 5V6  
email: umkrautj@cc.umanitoba.ca

## Abstract

*Abstract— Edge detection is a computer vision algorithm that is very processor intensive. It is possible to increase the speed of the algorithm by using hardware parallelism. This paper presents an implementation of edge detection in an FPGA, the Altera nios2 development kit. The paper focuses on providing the often missing link from the algorithm development to the FPGA implementation. In addition to a discussion of the edge detection algorithm, memory access and data transfer to the FPGA is discussed. Memory access is achieved by developing a generic component that handles the memory transfer on the Avalon Bus. The design is open so other memory intensive algorithms can be used with only a slight modification of components. In addition the testing software and firmware development is described. The results show how a highly parallel algorithm can run faster on a 50 MHz FPGA than a modern PC in the GHz.*

**Keywords:** Computer Vision, FPGA, Altera Stratix, Edge Detection.

## 1. Introduction

One of the algorithms used for computer vision is edge detection. It can be the first stage in processing a bitmap into a representation of an area that can be used for autonomous operations. The edge detection is a highly parallelizable algorithm that can be sped up dramatically with a custom circuit, in particular a FPGA.

In order to fully make use of an edge detection circuit it is necessary to transfer the bitmap data to the FPGA, have it stored, processed and saved, then have it returned to a pc. Several interfaces exist to a FPGA but among the fastest is the Ethernet interface. This led to the choice of using a development kit with a Stratix FPGA with a Nios2 embedded processor [1][2][3]. In addition to the hardware Ethernet module, it contains a software network stack and sample code to allow the development to focus on the edge detection circuitry.

The steps to perform the edge detection of a bitmap are as follows. The bitmap is captured from a web cam with a laptop using the DirectShow interface. It is then converted into 8 bit monochrome and sent to the FPGA. Since UDP is faster than

TCP a custom packet format is used to seamlessly packetize the data and re-form the bitmap in the FPGA. Then c code on the nios2 processor acts as the firmware and instructs the edge detection circuit to commence its operation. As soon as the edge detection is complete the Nios2 sends the bitmap back to the laptop, which complete the edge detection cycle for a single frame. The received bitmap can be compared with a bitmap that has had the edge detection formed with software to verify that the circuit is operating correctly.

## 2. Edge Detection

Edge detection consists of taking a 3 by 3 mask and convolving it over an image [6]. That is placing the mask over location 0,0 generating the output result moving the mask over by one and so forth. Figure 1 shows the commonly used sobel masks. When the convolution is complete a n x m bitmap is transformed into a n-1 x m-1 image since the edge pixels can't have the mask completely over them.

1	-2	1
0	0	0
1	-2	1

1	0	1
-2	0	-2
1	0	1

Figure 1. Sobel masks

Because the circuit is to be customizable to use other masks, the required operations are, 18 multiplications, the addition of 9 numbers twice, ensuring the numbers are positive, adding the results from the two results together, thresholding, multiplying by a number (for scaling purposes), and capping the value at a maximum value. 19 multiplications, 9 additions and 3 comparison assignments are required to output a single pixel. With pipelining thus outputting 1 pixel per clock cycle a FPGA running at 50 Mhz it can still be quite a lot faster than a PC running in the gigahertz.

## 3. Nios Avalon Interface

Peripherals that interface directly to the Nios processor use the Avalon Bus. The Bus consists of masters and slaves. The masters initiate communication and the slave responds to them.

The Avalon bus is a slave side arbitration bus. If multiple masters are connected to a single slave, each master gets its own dedicated lines connecting to the slave. The slave then controls the arbitration between multiple masters. The arbitration circuit is automatically generated by the SOPC program provided with in the development IDE. The only thing that had to be taken into consideration to use the bus is to properly generate the write and read signals, and to adhere to the bus timing.

The edge detector circuit requires an interface to the Nios processor so the processor can control its operations. It also requires a direct interface to memory, which significantly speeds up the operation versus having the processor act as an interface between the edge detector and the memory. This leads to having one slave port to receive data from the Nios processor and 2 master ports, one to read, the other to write to memory that allows it to use dual port ram as shown in figure 2.

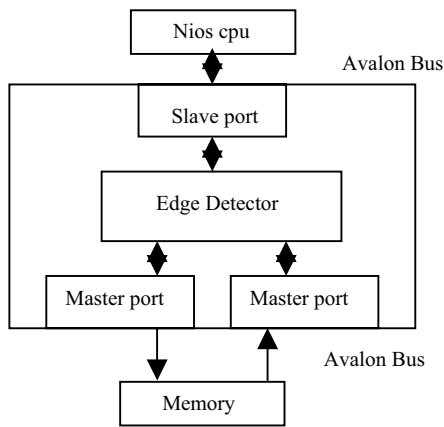


Figure 2. DMA architecture

The slave interface is simple. When *read* is asserted, place the data request by the address on the read bus. When *write* is asserted, take the data from the write bus and place it in a register at the location given by the address.

The master bus is slightly more complicated. Both reading and writing look similar [4]. The transfer begins on the rising edge of the clock. Place the address, *byteenable read\_n* on the bus. On the first rising edge when *waitrequest* is '0' the data is available and can be read in. This requires a simple state machine. There is also the option to pipeline the reads and writes but this is not used for the project.

#### 4. The Framework

To interface to the memory a framework is used that can be used for other projects as shown in figure 3. The framework consists of an input block to read the data and an output block to write to the memory. The diagram also includes an addition unit between the input and output block that is used for testing purposes and demonstrates the extensibility of the framework to other applications. The interface to the input block is three 8

bit data signals, a *dataout\_valid*, and *dataout\_request* control lines. When *dataout\_request* is '1' and *dataout\_valid* is '1' then data is clocked into the addition unit. The output block has a single 8 bit input line and a *datain\_request* and *datain\_valid* signal that works the same as the output block.

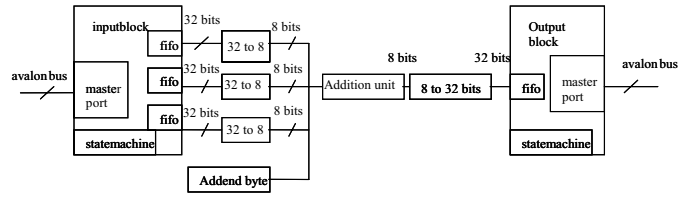


Figure 3. Framework architecture

To reduce the memory transfers the edge detector takes into consideration that for any output pixel other than the first of a row the six pixel values from the previous edge detection operation can be used. So other than the first pixel of the row only three bytes are required to generate an output pixel.

The 32 to 8 bit converter component are necessary because the Avalon bus supports 32 bit transfers so it makes sense to transfer in 32 bits and then shift out 8 bit in a time. Since the Avalon bus requires at least two clocks for any transfers (without pipelining) using this scheme the slowest operation is the read. Using fast onboard memory a read is performed using 2 clocks. Since 3 words need to be loaded in for a single output it takes 6 clock cycles to read in the 3 words. Since each read generates 4 bytes the framework can calculate  $6/4 = 1.5$  pixels per clock cycle if the edge detection does not have anything that takes more than 1 clock cycle to calculate.

#### 5. Edge Detection Component

The pipeline architecture of the edge detection component is shown in Figure 4. After the first multiplication stage four addition stages are required to reduce the nine 16 bit numbers to one number. Before this could happen if any of the masks are negative the twos complement of the multiplication result must be taken. Since edge detection requires only positive result after the four additions any remaining negative numbers are converted into positive. These numbers are then converted into a single result. One more additional multiplication stage is used for scaling the result, and then the threshold is taken. This stage also checks if the pixel value exceeds the maximum number of 255 because the result has to fit into a single byte.

Each pipeline stage is connected by having its output register enabled based on a control signal. Since the data is constant and not bursty, pipeline stalling and efficiency is ignored. All of the control lines are wired to a general enable, which is wired directly to *dataout\_request*. Having each stage depend only on if it can be placed into the next stage would not create any efficiency gains so it isn't done for simplicity sake.

The other control signals are generated with a very simple 6 state finite-state-machine. The 6 states are as follows, *preload1, preload2, preload3, normalload, postload, done*. The

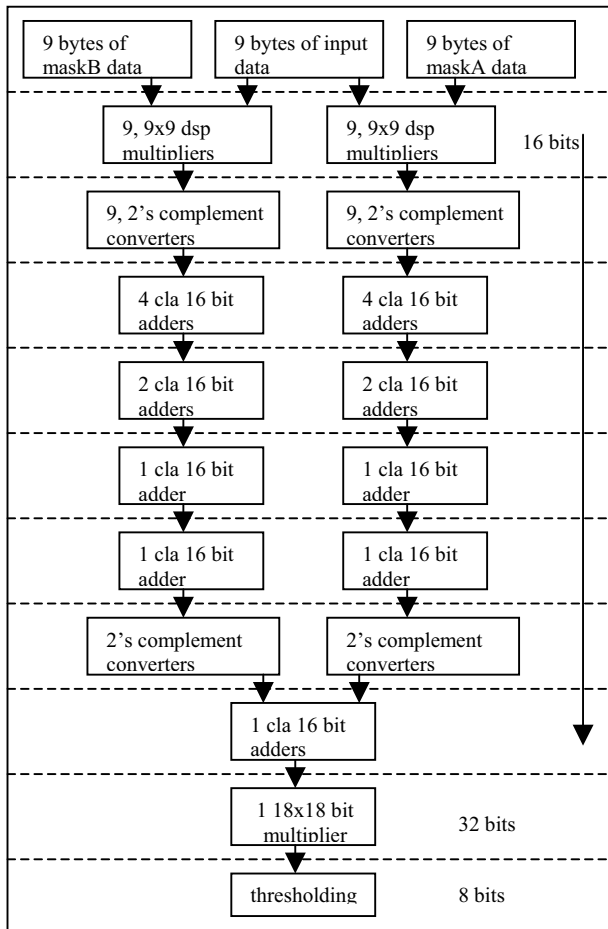


Figure 4. Edge detection circuit architecture

preload stages are when a new row is started and 3 successive loads of 3 bytes have to occur before processing starts. *Normalload* occurs after *preload3* and a single load of 3 bytes is all that is required. Postload is the stage after a row is complete, to guarantee that the last 3 bytes get used to calculate an output pixel.

Several additional components are required for the control. To determine when a new row occurs and when the process is done, two index registers, *currwidthreg* and *currheightreg* are used in the state machine. To store the status of each register state a 10 bit register is used. The *currwidthreg* is shifted along each time general enable is '1', with the last register being used as *dataout\_valid*.

## 6. Directshow, Packetizing, Firmware and Testbench

This section covers any miscellaneous items required to test the circuitry. It consists of how a bitmap is obtained from a web camera, how the data is packetized, and the firmware required for the Nios processor.

### 6.1. DirectShow Interface

Every web cam that operates in windows uses a standard driver interface. This allows for software makers to not have to customize code for different manufactures. For media streaming the standard interface is DirectShow. In the interface, each component of it is called a filter. A web cam source filter is a single component as is a frame grabber filter. DirectShow specifies how each filter connects with each other, and the data transfer method. Most of the connection work is automatically done with DirectShow with each filter just having to specify its possible formats.

To interface to DirectShow a frame grabber filter is created. The frame grabber is programmed just accept 24 bit RGB 320x240 frames. Once it connects to the web cam filter, the web cam can start streaming data. For each frame the frame grabber receives an *IMediaSample* interface which contains the functions to obtain the frame data. The frame grabber contains no processing and just sends the *IMediaSample* to the main image processing application via a callback function. The main application contains the code to send the data to the laptop.

### 6.2. Packet Transfers

The main application contains code to (un)/packetize the data to/from the FPGA, display the data and perform software edge detection to verify the FPGA edge detection is correct. It contains three display windows. One is the image directly from the web cam. The second is the software edge detection. The third is the hardware edge detection. It also contains various controls mainly to change the threshold and scaling values of the edge detection.

UDP is used to transmit to the FPGA because it is faster than using TCP. The since the FPGA and laptop are connected through a hub and Ethernet, UDP is inherently reliable in this setup and speed is more important than the extra reliability.

The data integrity is verified by having the data sent to the Nois processor and then echoed using 76800 bytes, the size of s a single 320x240 8 bit bitmap. The speed of the operation is around .2 seconds, which means that the FPGA is capable of around 5 bitmaps per second. There are some speeds up possible but the speed found of  $76800 \times 2 \times 5 = .768$  MB/S is very close to the maximum posted on the Altera product specification, so it is unlikely that more data can be sent.

### 6.3. Firmware

The firmware in the Nios processor is a modification from the *Simple\_socket\_server* program that comes with the IDE [5] to accept UDP packets. After a completed packet is received the firmware loads in control data through the slave port into the edge detector. The following registers are used.

The register map shown in Table 1, is relatively straightforward. Any one time calculation are offloaded onto the firmware rather than placing it in the hardware. Since the

input block reads from three rows at once three address pointers are required. Rather than calculate the offset in hardware, the starting points for each of the 3 address registers are loaded in. Address1 contains the starting point in memory that the bitmap data is contained. Address 2 is offset by the width from address 1. Address 3 is offset by twice the width from address 1. The output address is the starting point in memory that the bitmap data is to be written to.

Register map 32 bits		
1	Control register	
2	Input address 1	
3	Input address 2	
4	Input address 3	
5	Output address	
6-13	Maska(0-8)	Maskb(0-8)
14	Width	Height
15	Threshold	Scale
16	Load bytes	
17	Write bytes	
18-23	debugging	
24	Total clock cycles	

Table 1 Register Map

The masks are the Sobel edge detection value. The edge detector can be changed into any 3 by 3 mask operation. The bitmaps width and height are used by the edge detector to determine when a new row is loaded in, and when it is done.

To further add to the customizability other than the address registers all register information is actually sent from the laptop. These values are sent every time along with the bitmap. They are mainly used to change the scale and threshold value however they can be used to change the mask or bitmap size, without having to alter the firmware.

The load and write bytes are used for the finite state machines of the input and output blocks. Even though the bitmap is two dimensional they are store linearly in memory so the input and output blocks count each word.

There are various debugging registers used to transmit data from the edge detection circuit to the Nois processor, which then can either be displayed in the console or sent back through the Ethernet. This is done because this is the only way to observe what is having inside the circuit. Out of these the most useful is the number of clock cycles.

The control register controls the edge detection. When the least significant bit is written as a '1' the edge detection starts. After it starts the second bit is set for one clock cycle to indicate the registers are being reset. After that, the 3<sup>rd</sup> bit is monitored. When it turns into '1' the edge detection is complete.

## 7. Results

The edge detector is tested and proven to be working correctly. In total 114249 clock cycles are used to process a

single 320 by 240 bitmap. This is exactly the theoretical amount as  $(320 \times 238) \times 6/4 + 9 = 114249$ . Where 238 is used instead of 240 since 3 rows are read in at once, 6/4 (6 reads per 4 bytes) is used as the speed of the memory interface and the pipeline is 10 stages resulting in a 9 clock cycle latency.

The Original goal of continuous real time edge detection is not achieved. The Ethernet connection is too slow to support 320x240 bitmaps at 30 times a second. Also there are several minor glitches that prevent the edge detection from being continuous, such as a heap error in MicroOS.

## 8. Conclusion

This project is a good example of implementing an algorithm that is parallel and can be sped up with a FPGA. The Altera Stratix Nios 2 contained ready out of the box Ethernet which sped up development. The Avalon interface took a bit of time to implement but after that the edge detection circuit development was relative straight forward.

The edge detection circuit operates at 114249 clock cycles for a 320x240 bitmap on a 50 MHz FPGA. This translates into 437 bitmaps per second which is quite a lot faster than unoptimized c code can perform the edge detection on an new x86 processor.

## Acknowledgements

I would like to thank my advisor Professor Mcleod, Professor Bates at the University of Alberta for teaching an excellent ASIC course and TRILabs. Thanks to CMC Microsystems for providing the Altera Nios2 development boards. This work is being supported by a University of Manitoba graduate fellowship.

## References

- [1] Altera, "Stratix Device Handbook" [Online document], 2005 Jul (ver3.3), [cited 2005 Dec 2], Available HTTP: [www.altera.com/literature/hb/stx/stratix\\_handbook.pdf](http://www.altera.com/literature/hb/stx/stratix_handbook.pdf)
- [2] Altera, "FPGA Features, Package & I/O Matrix: Production Devices" [Online document], 2004 Nov (ver3.0), [cited 2005 Dec 2], Available HTTP: [www.altera.com/literature/manual/mnl\\_nios2\\_board\\_stratix\\_1s40.pdf](http://www.altera.com/literature/manual/mnl_nios2_board_stratix_1s40.pdf)
- [3] Altera, "Nios Development Board Reference Manual, Stratix Professional Edition" [Online document], 2005 Oct (ver 5.1), [cited 2005 Dec 2], Available HTTP: [www.altera.com/literature/hb/nios2/n2cpu\\_nii5v1.pdf](http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf)
- [4] Altera, "Avalon Bus Specification Reference Manual" [Online document], 2005 May (ver3.1), [cited 2005 Dec 2], Available HTTP: [www.altera.com/literature/manual/mnl\\_avalon\\_spec.pdf](http://www.altera.com/literature/manual/mnl_avalon_spec.pdf)
- [5] Altera, "Using Lightweight IP with the Nios II Processor Tutorial" [Online document], 2005 Jan (ver 1.1), [cited 2005 Dec 2], Available HTTP: <http://www.altera.com/literature/lit-nio2.jsp>
- [6] R. Gonzalaz, R. Woods, "Digital Image Processing". New Jersey: Prentice-Hall 2002, ch 10.