PARALLEL IMAGE PROCESSING AND COMPUTER VISION ARCHITECTURE

By

JAMES GRECO

A UNDERGRADUATE THESIS PRESENTED TO THE ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE SUMMA CUM LAUDE

UNIVERSITY OF FLORIDA

2005

I dedicate this work to the three people who have had a profound effect on my life. To my parents, James and Joyce, for always believing in me and to my fiance, Sarah, for showing me importance of life outside of work.

ACKNOWLEDGMENTS

This work would not be possible without the support of the Machine Intelligence Laboratory. Aaron Chinault, shares an equal level of recognition for his work on the hardware vision project. Eric Schwartz, the laboratory's associate director, has been a trusted mentor, academic adviser and friend.

I'd also like to thank my thesis adviser, Dapeng Wu, for making graduate school at UF an exciting opportunity.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Undergraduate Thesis Presented to the Electrical and Computer Engineering Department
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Bachelor of Science summa cum laude

PARALLEL IMAGE PROCESSING
AND COMPUTER VISION ARCHITECTURE

By

James Greco

May 2005

Chair: Dapeng Wu
Major Department: Electrical and Computer Engineering

Real-time image processing is limited with modern microprocessors. This thesis presents a novel method for the implementation of image processing and computer vision algorithms in hardware. Using pipelining methods from computer architecture, our system provides a flexible and fast platform for the development of image processing algorithms. The architecture's computational power over a Pentium 4 microprocessor is shown through an analytical analysis of the simulated performance. Two demonstrations of the architecture's implementation in a Field Programmable Gate Array are provided - The position of an object is tracked by using the object's color properties and the micro air vehicle on-board horizon tracking problem is solved.

CHAPTER 1
INTRODUCTION

The tremendous amount of data required for image processing and computer vision applications presents a significant problem for conventional microprocessors. In order to process a 640 x 480 full-color image at 30 Hz, a data throughput of 221 Mbs is required. This does not include overhead from other essential processes such as the operating system, control loops or camera interface. While a conventional microprocessor such as the Pentium 4 has a clock speed of nearly 4 GHz, running a program at that speed is highly dependent upon continuous access of data from the processor's lowest level cache. The level 1 cache is on the order of kilobytes, which is far to small to hold the data required for computer vision applications. Even most modern level 2 caches are not large enough to store entire images. Memory access times from the system's main memory, usually Synchronous DRAM, is an order of magnitude slower than the processor's cache and thus the large amount of data required for image processing will always be limited by memory access time and not the processor's clock speed. [1]

The disparity between memory access times and the processor's clock speed will only widen with time. While transistor counts and microprocessor clock speed have traditionally scaled exponentially with Moore's Law, memory access times have scale linearly. This is not to say a Pentium 4 is unable to handle many image processing algorithms in real time, but there is less growth room as applications require greater resolutions. Medical imaging in particular requires the processing of images in the megapixel range. [2]

In this thesis we propose an expandable architecture that can easily adapt to these challenges. As has been previously done, we move the development of image processing and computer vision algorithms from software to hardware. The move from software to hardware introduces several challenges: Traditionally linear algorithms must be rewritten to take advantage of the parallel structure afforded by implementation in hardware; Hardware high-level languages (Verilog and VHDL) are far less advanced than software languages (C/C++, Java, Basic, etc.); The compilers for the logic devices are orders of magnitude slower.

In addition to moving the algorithms from software to hardware, a parallel architecture has been developed to pipeline successive image processing functions. In much the same way a pipeline will speed up a processor, the architecture allows us to run many image processing operations in parallel. With this we can achieve a much higher data throughput than traditional computing systems. The architecture has been designed for implementation in two types of logic chips - The Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) - that afford power and size requirements [3] that are significantly lower than the smallest of Pentium motherboards.

The advantage of the parallel architecture over a traditional computer is shown through a quantitative analysis of a few simulated image processing functions. Speeds that are twenty-two times faster than modern microprocessors are possible with the simplest implementation of our architecture. It is further shown that the speed increase can be magnified by a factor of N by implementing N redundant data paths.

Beyond simulations, two experiments were tested on an Altera Cyclone FPGA. The first, a simple color tracking algorithm, interfaces with a CMOS camera to find the location of an object at thirty frames per second. In order to achieve these results, seven image processing and interface modules are connected in parallel. The second experiment is the implementation of a horizon tracking algorithm used by the University of Florida's Micro Air Vehicle (MAV) project. [4] Currently the

MAV uses a radio transmitter to download the data from the on-board camera to a laptop. [5] The demonstration shows that completely autonomous flight stabilization is possible without the need for a large radio transmitter or ground station.

CHAPTER 2
ARCHITECTURE OVERVIEW

2.1 Advantages and Disadvantages of Specialized Hardware

System designers are faced with conflicting goals of reuseability and performance. Systems that are reusable in a number of different applications will be slower than a system tailored to a single specific task. [6] This principle is true for both hardware and software design.

A simple example to show the advantages of custom designed hardware for a specific task is the summation of four numbers. Most modern microprocessors are limited to adding only two numbers in the same clock cycle because they only have a single ALU and adding four numbers requires three distinct uses of an ALU. Alternatively, a customized digital circuit of three adders in parallel can be used to calculate the sum in a single clock cycle. Although this new circuit is three times faster, three times the hardware is needed. There is some flexibility in this as the amount of hardware can be customized to the number of bits required by the addition - If only 8-bit addition is needed, the amount of required hardware for three 8-bit adders is significantly less than a single 32-bit adder. [7]

It is generally accepted that additional hardware used to solve a small set of problems is not worth implementing in modern microprocessors. [8] If we have an instruction dedicated to adding four numbers, why not have one that calculates the y coordinate in the line equation? ($y = mx + b$) This operation would require the microprocessor to multiply two numbers and add a third to the result. The seldom need for this operation in code does not justify including it in the processor. It is likely the operation will cause the processor to require a longer worst case delay and thus run most programs slower overall. Unless the microprocessor is designed specifically for a program that requires this calculation continuously, implementing the function with two smaller instructions (a multiply and then an add) will have a higher data throughput.

2.2 Module Design

While a microprocessor favors generality over instructions that are more limited in scope, our architecture is much more specific to image processing. The small instruction set of additions and multiplications are replaced by a set of image processing tools. Each of the tools has it's own internal RISC (Reduced Instruction Set Computer) structure to keep speed above a minimum desired rate and give a set of common inputs and outputs that allow for module reuseability and the pipelining of consecutive modules.

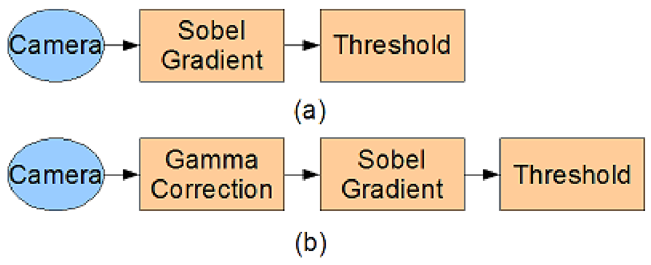

Figure 2–1: Modularity example. (a) The camera data is passed through two pipelined functions (b) A gamma correction function is inserted into the pipeline. This inserted function has nearly no effect on the computational time of the algorithm or interferes with the timing of other modules in the chain.

By having a common set of inputs and outputs, a module can be dropped anywhere in the pipeline without affecting the rest of the system. This is important in any embedded system that has a timing critical application.

Definition of the signals in Figure 2-2:

- Pixel Clock - The internal state machine of the module runs off the pixel clock. The RGB Data is also valid on the rising edge of the clock assuming Local Enable 1, Local Enable 2, Global Enable are all true.

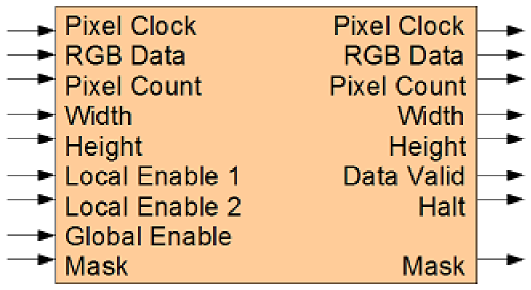

Figure 2–2: The standard inputs and outputs used in our architecture

- RGB Data - A 24-bit signal that represents the pixel data. This signal does not have to be RGB, but can instead use HSV, grayscale, or an arbitrarily defined pixel format.
- Pixel Count - Running count of the pixel position in the image.
- Width - Number of pixel columns in the image.
- Height - Number of pixel rows in the image.
- Local Enable 1 - Signal from the previous module to determine if the RGB data is valid. The Internal state machine can continue if not dependent upon data from the previous module.
- Local Enable 2 - Feedback signal from the next module to determine if the module should hold data processing. The Internal state machine can continue if not dependent upon data from the previous module.
- Global Enable - Input signal from a main arbiter, external interrupt or global reset that will stop all modules. The internal state machines should continue to be reset while the signal is high.
- Mask - Used by some modules when dealing with two or more classes of data. The horizon tracking algorithm uses this signal to mask whether the incoming pixel is sky or ground.
- Data Valid - Output signal that feeds into the following module's Local Enable 1. If the output RGB Data is valid, this bit will be true.

- Halt - Output signal that feeds into the previous module's Local Enable 2. If the module (or modules further in the pipeline) need more time to process data, a halt signal is issued to stop the previous module from losing data.

The rest of the system's structure is revealed in the following sections.

2.3 Technology Overview

Before discussing specific applications of the design, an introduction to the technology used in the implementation of our architecture is warranted. There are only two technologies that are appropriate to the implementation of our architecture - The Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA).

FPGAs were used in the design of our architecture because they offer a cost-effective solution for small-volume productions. FPGAs are an advanced type of programmable logic device (PLD) that use an SRAM-based programmable array of interconnections to network thousands of logic cells together. Each logic cell contains a look up table (LUT) and a flip flop to perform combinatorial and registered operations. [9]

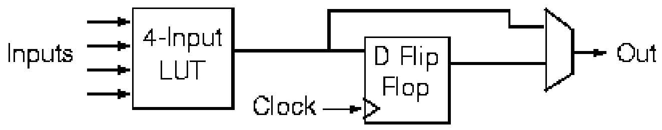

Figure 2–3: The most basic implementation of an logic cell contains a LUT for combinatorial output and flip flop for registered output

When a design is compiled for an FPGA, the internal connections are routed horizontally and vertically through the chip. Modern FPGAs contain DSP blocks, embedded multipliers, megabits of internal ram and PLLs as advanced features. [3] Recent technology has also moved toward the integration of entire systems - multiple processors, glue logic, I/O interfaces - on an FPGA. This technology, called System on Chip (SoC), is especially needed for applications that require the integration of a controller, image processing, glue logic and sensor readings on a single IC for power and size constraints. Specifically a micro air vehicle could take advantage of this technology by combining the on-board controller and the off-board vision processing onto a single FPGA.

FPGAs not only offer orders of magnitude greater data throughput than an x86 processor, but also significant reductions in the power requirements and size that is necessary for an embedded system. For our development platform we are using the Altera Cyclone EP1C12F256 (17 mm x 17 mm), which is only slightly larger than the ATMega128 microcontroller (16 mm x 16 mm) used on the current iteration of the micro air vehicle project. The Cyclone family is a low-cost FPGA that can operate at speeds up to 166 MHz. Typically chips in the family cost between $10 and $100 retail. Advanced Altera FPGA families (Stratix, Stratix II) are capable of running internal logic and RAM at 500 MHz.

ASICs can be thought of as high-volume productions of FPGAs. The logic element structure is replaced by gates that are dedicated to the ICs specific task. Instead of routing an programmable array of interconnections, the connections are burned into the ASIC. Thus ASICs are not repogrammable, but are much faster than FPGAs. ASICs can be far cheaper than FPGAs, but only in mass quantities. A typical ASIC order is in the millions of dollars.

CHAPTER 3
NUMERICAL ANALYSIS OF THE ARCHITECTURE

Table 1 demonstrates the advantage of individual operations in our architecture over conventional microprocessors. These tests were conducted on a 1.6 GHz Pentium 4 with 768 MB of RAM. Operations were made as efficient as possible with vector-optimized compiled Matlab functions. The FPGA results were simulated on an Altera Cyclone EP1C12-8 running at a pixel clock of 75 MHz. The same 8-bit 640 x 480 grayscale image was used on both platforms.

Operation	P4 (ms)	FPGA (ms)	Ratio
Vert. Sobel Filter	50.0	4.10	12.2
Highpass Threshold	10.0	4.10	2.44
3 x 3 Erode	30.0	4.10	7.32

Table 3–1: P4 and FPGA timing comparison for three operators

Area operations such as the vertical Sobel filter and the 3 x 3 erode have the greatest impact on the FPGA to P4 speed ratio. The vertical Sobel filter requires 6 multiplies to be performed sequentially, while the FPGA can handle all six at once. Similarly the 3 x 3 erode requires 9 comparison operations. A detailed discussion of the implementation of these functions is given in the following sections.

3.1 Vertical Sobel Filter

If the image is defined as 2-D matrix Im(y, x), The vertical Sobel filter provides a discrete approximation to the first partial derivative of the image with respect to y. Similarly, the horizontal Sobel filter provide a discrete approximation to the first partial derivative of the image with respect to x. The first derivative of the image in either direction will emphasize sharp transitions, or ‘edge’ features. This feature extraction method can be combined with a shape finding algorithm, such as the Hough Transform, to find the position and size of arbitrarily defined shapes in the image.

The first derivative of the image can be approximated using a 2-D convolution of a kernel and image data.

$$Im'(y,x) = \sum_{j=1}^{M}\sum_{i=1}^{N} k(i,j) * Im(y+j-M/2-1, x+i-M/2-1) \qquad (1)$$

Where M and N represent the size of the kernel (3 x 3 for the Sobel filter). The kernel (k) for a vertical Sobel filter is defined as

$$k = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad (2)$$

The two tests achieved exactly the same output, but the FPGA had a speed advantage of 12.2 times the Pentium 4.

Operation	P4 (ms)	FPGA (ms)	Ratio
Vert. Sobel Filter	50.0	4.10	12.2

There was one minor difference in implementation of the methods. The 3 x 3 Sobel filter was optimized for the Pentium platform by splitting the 2-D convolution into two 1 x 3 convolutions. Since our implementation performs all 9 multiplications at the same time, this modification is unnecessary. The reason for the performance increase over a 3 x 3 convolution is beyond the scope of this discussion.

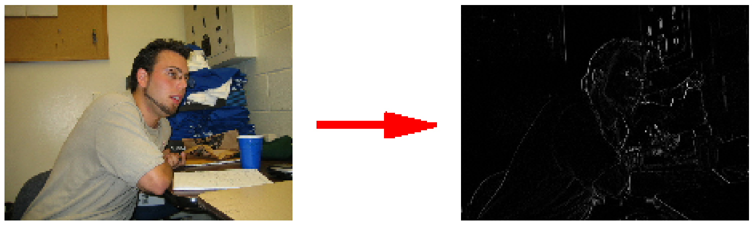

Figure 3–1: The absolute value of the output of the Sobel gradient module emphasizes the edges in the previous image.

3.2 High Pass Threshold Filter

The high pass threshold filter, which transforms the image from grayscale to binary (or a single color channel to binary) based on a threshold value, is defined using a simple comparison operator.

$$Im'(y,x) = \begin{pmatrix} true & Im(y,x) > Threshold \\ false & o.w. \end{pmatrix} \tag{3}$$

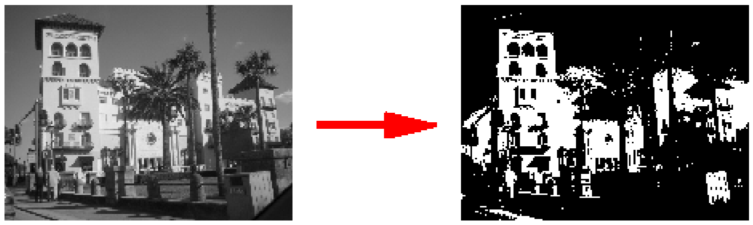

Figure 3–2: The output of the threshold module is a binary (Two color) image.

3.3 3 x 3 Erosion

The final operation is a 3 x 3 erosion of the image. The erode operator low pass filters an image by removing point features resulting from the previous threshold stage. The erode operator tests a 3 x 3 region of a binary image to see if any of the pixels in the region are false. If any of the pixels in the region are false, the center element is also set to false.

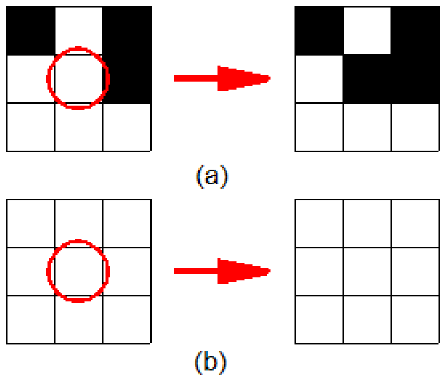

Figure 3–3: 3 x 3 erode function: The circled block is the current pixel being examined. (a) Some of the surrounding pixels are false (black), so the resulting pixel is set to false. (b) All of the surrounding pixels are true (white), so the center pixel is kept true.

Mathematically this can be expressed as

$$Im'(y,x) = \begin{pmatrix} 1 & \sum_{j=1}^{M}\sum_{i=1}^{N} Im(y+j-M/2-1, x+i-N/2-1) = M*N \\ 0 & o.w. \end{pmatrix} \quad (4)$$

Where M and N represent the size of the region of interest (3 x 3).

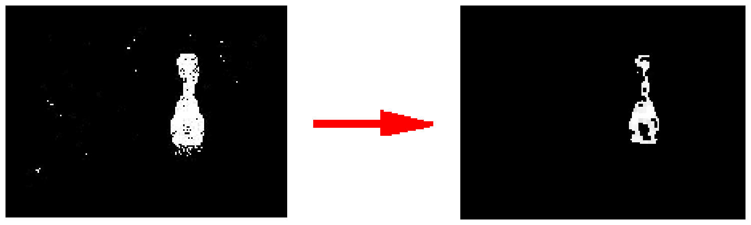

Figure 3–4: Output of the erode module

3.4 Pipelining

In the individual operations tested here, the FPGA outperforms the Pentium by a factor of 2.44 to 12.20. While this advantage is significant on its own, pipelining the functions in hardware provides even more of an advantage for the FPGA over conventional microprocessors. When pipelining techniques were used to chain the three modules together in series, the FPGA outperformed conventional microprocessors by a factor of nearly 22.

Operation	P4 Cum. (ms)	FPGA Cum. (ms)	Ratio Cum.
Vert. Sobel Filter	50.0	4.10	12.2
Highpass Threshold	60.0	4.10	14.6
3 x 3 Dilate	90.0	4.10	22.0

Table 3–2: P4 and FPGA (1 data path) cumulative timing comparison

The advantage of the FPGA over the Pentium is for a continuous 8-bit grayscale stream only. An RGB color image sees a three-fold increase in speed because our architecture is designed to handle 24-bits of data per clock cycle.

Similarly, with N redundant data paths (dividing the image into N number of blocks that all process separately), N times the performance is possible. The only drawback is N times the hardware is required. Table 3-3 shows the results of dividing the screen into 2, 4, and 8 separate images for the threshold function. This operation is not possible on a conventional microprocessor.

N	Sub Image Resolution	FPGA (ms)
1	640 x 480	4.10
2	640 x 240	2.05
4	320 x 240	1.03
8	160 x 120	0.51

Table 3–3: The threshold function with N Sub Images performs at N times the speed.

Table 4-4 contrasts the cumulative results of dividing the image into four sections against a traditional implementation. 87.4 times the performance is registered.

Operation	P4 Cum. (ms)	FPGA Cum. (ms)	Ratio Cum.
Vert. Sobel Filter	50.0	1.03	48.5
Highpass Threshold	60.0	1.03	58.3
3 x 3 Dilate	90.0	1.03	87.4

Table 3–4: P4 and FPGA (4 data paths) cumulative timing comparison

CHAPTER 4
ARCHITECTURE EXPERIMENTS

Our system has been taken beyond the simulated results in two different experiments that were implemented on a mid-range Altera Cyclone. The first, a color-tracking algorithm, shows the novel interactions of seven image processing and interface modules. The second, the MAV horizon detection algorithm, provides an on-board solution for an application that is restricted by weight, power and size.

4.1 Tracking an object by color properties

To keep the color-tracking experiment simple we decided to track only solid color objects that are easily parameterized. Specifically, the following examples show how the centroid of a blue bowling pin is found. We used an Omnivision OV7620 as a camera input device. The OV7620 is a low-cost CMOS camera that is capable of resolutions of 640 x 480 pixels. A popular computer vision interface for robotic hobbyists, the CMUCam, uses the OV7620 in it's design.

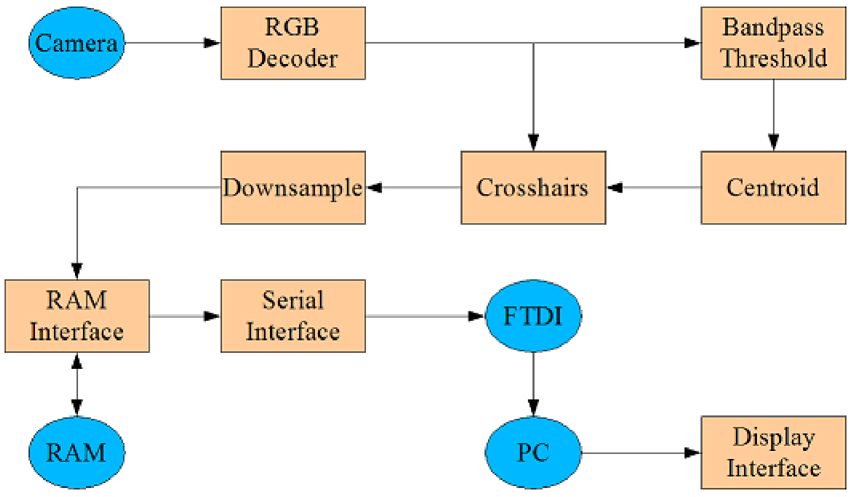

Figure 4–1: A simple implementation of our architecture is used to find the center of a uniformly colored object based on it's color properties.

4.1.1 Bandpass Threshold

It is our goal to separate the image into two classes: Blue bowling pin and not a blue bowling pin. We have arranged the problem such that the object can be detected by it's color properties alone. Thus it is unnecessary to consider the shape or texture in the detection of the object. A thresholding operator can successfully segment the image into the two classes by exploiting the unique color of the bowling pin relative to the rest of the image.

The bandpass threshold is similar to the high-pass threshold presented in the previous section although two comparisons are done to check if the RGB value is between a lower and a upper threshold. A high-pass threshold on each color channel would result in all red, green and blue objects being detected. Simply ignoring the red and green channels and passing the blue channel through a high-pass threshold will give erroneous results for white objects. As with the previous threshold module, a binary image is produced from the result.

$$Im'(i,j) = \begin{pmatrix} true & THigh > Im(i,j) > TLow \\ false & o.w. \end{pmatrix} \tag{5}$$

The comparison in equation 5 is done for different values of TLow and THigh on each color channel. The threshold values were determined from a model of the object's RGB properties. If all three channels are between the TLow and THigh values then the pixel is set to true, otherwise the pixel is set to false. The architecture has an even greater advantage over a conventional microprocessor in this module - A bandpass threshold would require two separate clock cycles for the comparison stage to check if the value is greater than TLow and then less than THigh. Instead, both comparisons are done in parallel.

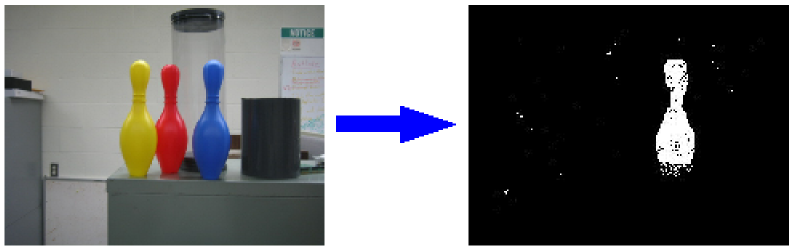

Figure 4–2: The bandpass threshold module produces a binary image that represents the two classes of data (blue bowling pin and not a blue bowing pin)

4.1.2 Centroid

Finding the centroid of the blue pin can be easily found if the previous step misclassified relatively few pixels. The mean of all pixels classified as a blue bowling pin will give X-Y image coordinates of the object's center.

$$X_{mu} = \frac{\sum_{j=1}^{N}\sum_{i=1}^{M} i * I(j,i)}{\sum_{j=1}^{N}\sum_{i=1}^{M} I(j,i)} \qquad Y_{mu} = \frac{\sum_{j=1}^{N}\sum_{i=1}^{M} j * I(j,i)}{\sum_{j=1}^{N}\sum_{i=1}^{M} I(j,i)} \tag{6}$$

Equation 6 will work for a binary image only.

4.1.3 Crosshairs

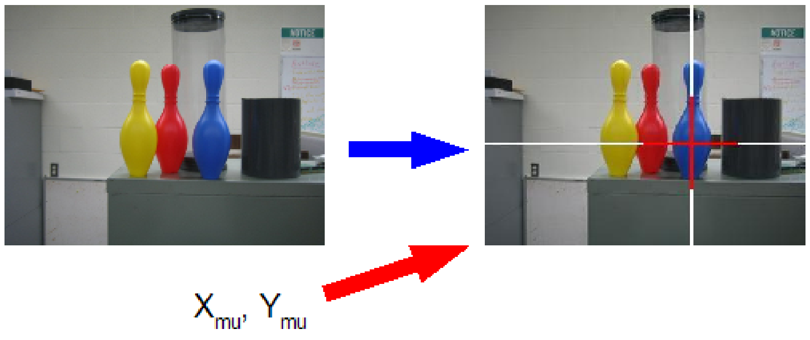

Figure 4–3: The crosshairs module receives input from two previous modules - Centroid and RGB decoder

The results from the centroid module are then used by the crosshairs function to paint a visual cue on the original image. (Figure 4-3) Unlike previously discussed modules which operate in series, the crosshairs module receives inputs from two modules. (The RGB decoder and centroid modules. The parallel branch that is taken before the crosshairs function is one of the unique parts of our architecture that can not be duplicated on conventional microprocessors.

4.1.4 Downsample

Our system was limited by the small amount of external RAM that is needed to store the image resulting from the crosshairs module. We downsample the image by four (from 320 x 240 pixels to 160 x 120) and store the result in RAM.

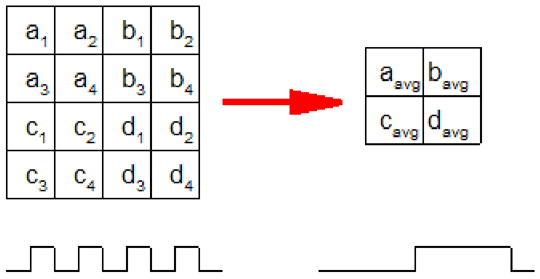

Figure 4–4: A group of sixteen pixels is converted to four pixels by averaging each region of four pixels. The clock frequency is also quartered.

The downsample function averages a region of four pixels to produce one pixel. This divides the image and slows the clock down by a factor of four. The pixel data is arriving in 'real-time' so unlike a conventional microprocessor that has the entire image data stored in RAM, a single row of pixels must be buffered in the internal memory. Multiple downsample functions can be chained together to further decrease the resolution.

4.1.5 Interfaces

In addition to the algorithms, several interface modules had to be developed. They are described briefly bellow.

- The RGB decoder transforms the 640 x 240 16-bit interlaced video signal from the OV7620 camera into a 320 x 240 24-bit progressive-scan RGB signal

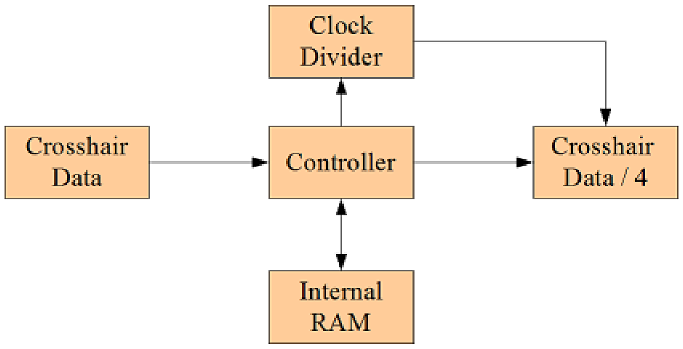

Figure 4–5: Downsample module block diagram

used internally. This module also keeps track of the current pixel's index relative to the first pixel in the image.

- An external RAM interface was necessary because the large amount of image data was impossible to store internally. A 320 x 240 full color image requires 1.84 Mbs of memory - the Altera Cyclone EP1C12 has only 240 kbs of internal memory [3]. In addition to the storage capacity, we used the interface to emulate dual-port memory so the in-line nature of our algorithms could be exploited.
- Finally, a serial interface was developed to transmit the images from the FPGA to a computer for display. The serial protocol is translated to a USB protocol using an external serial to USB converter chip. This is the major bottleneck of our system as the serial interface was found to have a max bandwidth of 1.5 Mbs.

4.2 Micro air vehicle horizon detection

The micro air vehicles project at the University of Florida is a multi-disciplinary team of electrical, mechanical and material engineers. MAVs present a significant engineering challenge as they have wingspans of 4.5-24 inches and have payload that is in the hundreds of grams. The ultimate goal of the MAV project is autonomous urban combat missions. As a first step, the MAV project has developed algorithms for vision assisted flight. Currently, a radio transmitter is used to download the data from the on-board camera to a laptop for processing.

The detection of the horizon line must be done in order to assist the operator in correcting the roll and pitch of the MAV. Ettinger proposed that while the specific color of sky and ground is not constant throughout the day or in different weather conditions, sky pixels will look similar to other sky pixels and ground pixels will look similar to other ground pixels.

The following cost function will quantify the color assumption.

$$J = \frac{1}{|\Sigma_g|+|\Sigma_s|} \qquad (7)$$

We evaluate, J, across 36 possible orientations of ϕ and 12 possible lengths of σ in line parameter space. The maximum value of J represents the minimum statistical variation from the mean of sky and ground pixels. Further details of the algorithm can be found in [10]. Using the architecture described in this paper, we have successfully implemented the algorithm on an Altera Cyclone EP1C12F256 with a pixel clock rate of 60 MHz. With the integration of a controller and sensors, it is possible to have completely autonomous flight as the IC is only slightly bigger than the current MAV ATMega128 controller.

It should be noted that the original algorithm also includes the RGB eigenvalues of the ground and sky regions. Our experiments show that similar results are obtained without the need for this step.

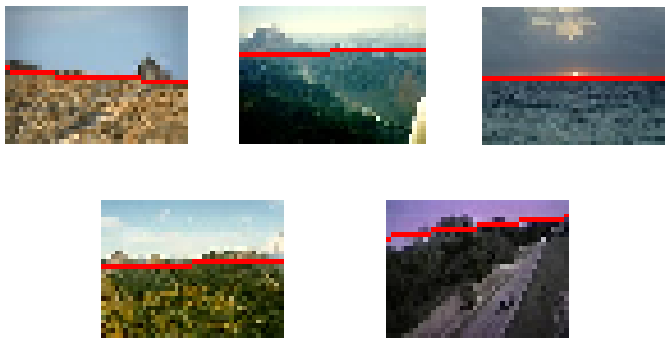

Figure 4–6: The horizon line is approximated for several test images. Performance of the algorithm can be increased or decreased based on the number of possible horizons tested.

CHAPTER 5
FUTURE WORK

Future development of the parallel computer vision and image processing architecture should focus on optimizations for applications such as the micro air vehicle project. To ease the significant amount of development time required, work must be done to automatically compile a series of linear functions into the parallel structure. If the micro air vehicle project is to move toward on-board vision processing, the architecture must be integrated with on-board sensors and controllers. The use of an inexpensive Digital Signal Processor (DSP) as the primary controller and an FPGA co-processor would give the architecture enough power to implement far more advanced algorithms than presented here.

CHAPTER 6
CONCLUSION

In this paper we have presented a hardware architecture that provides orders of magnitude greater performance for some computer vision problems. A parallel and pipelined approach was used in the design so that data throughput could be maximized. The architecture, implemented on modern FPGAs, has been successfully used to solve the micro air vehicle horizon tracking problem. If pursued further, the architecture could allow for completely autonomous on-board vision processing in the MAV and other small autonomous vehicles.

CHAPTER 7
REFERENCES

[1] S. Brown and J. Rose, Architecture of FPGAs and CPLDs: A Tutorial, *IEEE Transactions on Design and Test of Computers*, 1996

[2] P. Hillman, J. Hannah and D. Renshaw, Alpha Channel Estimation in High Resolution Images and Image Sequences, *Proceedings of Computer Vision and Pattern Recognition*, 2001

[3] Altera, Cyclone FPGA Family Data Sheet, 2003

[4] S. Ettinger, M. Nechyba, P. Ifju and M. Waszak, Vision-Guided Flight Stability and Control for Micro Air Vehicles, *Proceedings IEEE International Conference on Intelligent Robots and Systems*, 2002.

[5] J. Grzywna, J. Plew, M. Nechyba and P. Ifju, Enabling Autonomous MAV Flight, *Florida Conference on Recent Advanced in Robotics*, 2003.

[6] B. Draper, R. Beveridge, W. Bhm, C. Ross and M. Chawathe, Accelerated Image Processing on FPGAs, *IEEE Transactions on Image Processing*, 2003

[7] J. Greco, Parallel Computer Vision Architecture, IEEE SoutheastCon, 2005.

[8] D. Patterson and J. Hennessy, Computer Organization and Design: The Hardware/software Interface, 2005.

[9] Altera, Nios II Software Development Handbook, 2004

[10] S. Ettinger, M. Nechyba, P. Ifju and M. Waszak, Towards Flight Autonomy: Vision-Based Horizon Detection for Micro Air Vehicles, *Florida Conference on Recent Advanced in Robotics*, 2002.

BIOGRAPHICAL SKETCH

James Greco is a member of the Machine Intelligence Laboratory. There he has worked on or led many autonomous robotic projects including a land rover based on the '97 Mars Sojourner and Gnuman the three-wheeled tour guide. He is currently leading a team of undergraduate and graduate students on the SubjuGator 2005 project, which is the University of Florida's entry into the annual AUVSI autonomous submarine competition.

He has won numerous awards while at the University of Florida including the Electrical and Computer Engineering department's highest honor, the Electrical E award, and the College of Engineering's highest honor, the Outstanding Gator Scholar award. His studies have been supported by the Sias Scholarship, the Wayne Chen Scholarship and the Florida's Bright Futures Scholarship.

After graduating, James plans to begin doctoral studies at the University of Florida's department of Electrical and Computer Engineering. He has been awarded a prestigious four-year alumni fellowship from the department to support his studies.