Senior Project
Final Report

Low-Cost Portable Edge Detection Device

Students:

Paul Grinberg: _____

Mahmoud Yasir Imam: _____

Advisor:

Stephen Phillips: _____

Date: _____

Table of Contents

List of Figures

List of Equations

Executive Summary

The purpose of this project is to construct a low-cost portable edge detection device. The device will acquire an image from a CCD camera through a serial port. The device will then process the image for edges using the Prewitt edge detection method. The output is then compressed and passed on to the user serially.

The team's original Gantt chart was modified early in the project to incorporate a more appropriate solution to the problem. This included a new processor, a different means of acquiring the image, and a compressed format for outputting the edge data. All tasks and milestones listed on the new Gantt chart were met. The team was successful at creating this device. All functional requirements were achieved or were within the acceptable margins of error. To correct the requirements that not completely satisfied are followed by suggestions on improvements.

The team was able to meet most all of the functional requirements successfully. These requirements include the price, portability, power consumption, and other design restrictions. Of these requirements, the power consideration was not met by simply using two AA batteries as anticipated. However, simply using a different kind of battery resolves this issue. The speed consideration of the device was not completely met either. The functional requirements state that the throughput of the device should be 5 to 10 frames per second. The implemented algorithm, however, has a throughput of just over 2 frames per second. The solution to this problem would be to utilize a slightly less complicated edge detection algorithm, or a faster processor.

The functionality of the device was verified using two software tools. VisualDSP++ 2.0, a tool which was provided by Analog Devices, was used to create and debug the code running on the main processor. A second tool was created by the team members to test the validity of the output of the edge detection algorithm.

Introduction

In the current age of automation, it has become increasingly important to have the ability to use a computing device that quickly and efficiently processes large amounts of data. Some of the more computationally intense calculations involve performing calculations on images. The goal of this project is to create a low-cost portable image processing unit for a similar task.

One of the most important image processing techniques is edge detection. Practically any rudimentary vision system begins with this task. By locating edges within the scope of its view, an autonomous robot can navigate itself around obstacles. By comparing the edges within an image to a predefined edge template, the quality assurance process can be automated to discard faulty parts. By integrating an edge detection system into an X-Ray machine to control image exposure, the machine could create crisp, well-defined x-rays every time.

There are numerous software products available for a multitude of computer operating systems capable of performing edge detection. However, the cost of a computer is hard to justify if it is only bought to process images. Also, weight and power restrictions make a computer an unlikely choice for a portable device. To overcome these obstacles, this paper will discuss the design and implementation of a hardware oriented low-cost portable edge detection device.

Special Features

A product with these capabilities must have the following properties and specifications:

- Low power consumption – Since this device is portable, it should be able to operate on battery power, with battery life expectancy of at least 1 day of continuous operation.
- Low-cost – The mass production cost of the device should not exceed $50-$60.
- Fast – The device should be able to detect edges at a rate of at least 5 frames per second. The intended application range for this device varies from intelligent motion detection to quality assurance control on assembly lines. Conveyors normally have a maximum throughput of 5-10 items per second. Thus, real-time image processing, which is on the order of 30 to 60 frames per second, is not necessary for this application. A similar analogy can be extended to a smart motion detector. Since such a detector would specialize on identifying animated moving object, it will have to sample the images fast

enough to ensure that the object in question does not cover the full field of view between samplings. For humans and household animals this implies a sampling rate of approximately 3-6 times per second.

- Portable – The device should be lightweight, small, and encapsulated in a convenient package.
- Work with a specific camera – This device should be built around the specifications of a pre-selected camera.

Background Work

Throughout background research, many computer-based applications of edge detection were located. Unfortunately, most computers are not portable, and none would be considered low-cost. Motion sensors are close in meeting some of the image acquisition requirements but do not detect edges. No devices suitable of filling all of the specifications listed above were found.

Product Description

The edge detection device interfaces to an image acquiring camera through a serial port. The device itself has a single main processing unit in charge of image acquisition and edge detection. Upon processing the image for edges, the device outputs that data serially to the user or the next stage of the computation process (See Figure 1). This output is in a compressed format.

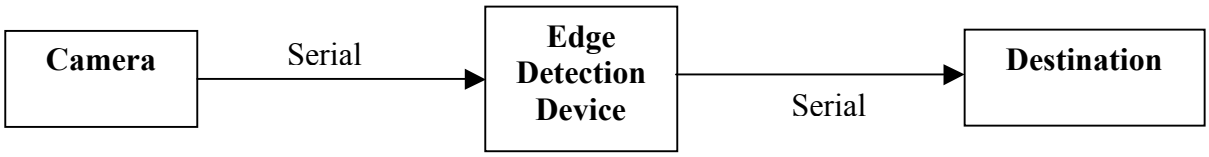


Figure 1: Block diagram of the edge detection device

This design implements a specialized solution to the edge detection problem. As it will be shown later, the design will focus on particular components and will give guaranteed performance for that specific hardware.

Team Structure

This project team is made up of two people: Paul Grinberg and Mahmoud Yasir Imam. Paul's contributions to this team focused more towards software development and algorithm design.

Paul developed limited software control mechanisms for interchip communication. He also worked on implementing the basic edge detection algorithm.

Yasir lead the responsibility of hardware development. The primary focus of this task was implementing the proper image processing chip. He also worked on the final configuration and layout.

Yasir lead the team efforts to keep updated written records. These records include the Midterm and the Final reports. Yasir was the team leader for scheduling whereas Paul was the team leader for technical issues.

Methodology

Due to the nature of this project, the team approached the project by focusing on one functional block at a time. This, of course, implied a strict distinction between hardware and software designs. The following sections discuss the team's approach and solutions to this project, as well as the project verification procedures.

Following the original proposal, (See Appendix A: Original Gantt Chart), the first steps of the project involved the component layout and processor communication design. It was known from the beginning that image processing techniques require large computing capabilities. The team expected to expand on previous semesters work, building upon the memory to PIC microprocessor interface. To improve the throughput capability of the design, the team anticipated the addition of a second PIC processor which would perform computations in parallel with the main controller. Since at that time the anticipated input into the edge detection device was a compressed image, another hardware component was needed for the sole task of decompressing and decoding the image.

Research into the image decoder resulted in several solutions, none, however were suitable for the desired implementation. Either the chips operated too fast for the PIC microcontroller or the chips were too expensive and inaccessible. In either case, this prompted the team to reconsider its original design. In particular, after talking to various faculty members, it was discovered that

most applications requiring image processing never compress the image between the acquisition stage and the processing stage. Also, most applications are not color sensitive. Thus, it was inappropriate for the proposed design to handle compressed color images, which eliminated the need for a hardware image decoder.

Continuing with this research, alternatives into processors were explored. It was obvious that the computing power of a PIC microcontroller was insufficient for such a computationally intensive task as image decoding. Taking on the suggestions of the faculty, close attention was paid to a Digital Signal Processing (DSP) chip as an alternative processor. Its specialized features like pipelining and parallelization created an arithmetic machine more suitable for the task at hand.

These discoveries prompted the team to revise the project plan. Please refer to the following sections for detailed explanations on the changes, and see Appendix B: Revised Gantt Chart, for the modified Gantt chart.

Hardware Design

Camera

As it can be seen in Figure 1: Block diagram of the edge detection device, the first step in edge detection is actually acquiring the image on which the edge detection device will work. It was decided early in the project that the integration of a camera into the edge detection device was outside the scope of this project. Thus, the team agreed to emulate the functionality of a camera on a computer. However, before the emulation could begin, the team needed to pick a particular camera so that the modeling would approach the real world scenario.

When selecting the camera, the following considerations had to be kept in mind:

- Cost – Since the edge detection device is supposed to be low cost, the price restriction was kept as a top criterion.
- Output format – Since the edge detection device was expecting the image data to arrive through a serial port, a camera capable of serial output was desired.
- Resolution – To reduce the computation time to process one image, the camera should not have a large resolution and should ignore colors.

Basing the decision on previous research[i], the team decided to explore Charge-Coupled Device (CCD)[ii] cameras for image acquisition. In general, CCD chips are low power and rather low cost. After doing further research, the team identified Texas Instruments (TI) TC255[iii] as a likely source for an image acquiring device. This grayscale camera had all the preferred features including: low cost ($35.40), low dark current (power efficient), small size (8-pin dual-in-line ceramic package), and sufficient resolution (324x243 pixels).

The TC255 is a frame-transfer camera and, as with all such devices, it outputs rows of pixels sequentially, one frame at a time. The chip comes with an onboard serial transmitter, which facilitates the transfer of data. Thus, it is trivial to transfer the image from the CCD camera to the edge detection device. However, the chip has a drawback. It requires a multi-phase clock source and some other glue logic. TI provides these chips to overcome this drawback. Only one extra chip (TMC57253) and a 25 MHz crystal totaling $12.13 are needed.

Processor

After acquiring the image, the camera serially outputs the pixel intensity data one point at a time. The edge detection device then acquires that data and processes it for edges. To do so, the device must have at least two components: a Universal Serial Asynchronous Receiver Transmitter (USART) and a processor to perform the edge detection computation. As it was already explained, the original design did not have sufficient computing power to act as a processor. Exploration into processor alternatives yielded an appropriate processor substitute in the form of a Digital Signal Processing (DSP) chip.

Given the limited time frame, the research was limited to the currently available departmental resources. DSP applications require specialized development tools which are not freely available. The department had these resources for two DSP chip manufacturers, Analog Devices and Texas Instruments. With these limitations, only these two DSP chips were taken into careful consideration. The Texas Instruments (TI) TMS320C31[iv] and the Analog Devices (ADI) ADSP-2181[v] were thoroughly compared. Please see Figure 2 for a detailed comparison of the PIC16C67[vi] microcontroller and the DSP chips.

	PIC16C67	TMS320C31	ADSP-2181
Clock (MIPS)	5	25	40
# Of Serial Ports	1	1	2
Data Memory	368 words (8 bits)	32k words* (32 bits)	16k words (32 bits)
Program Memory	8k words (8 bits)	32k words* (32 bits)	16k words (32 bits)
Low Power Modes	1	2	1
Price (USD) / Unit	13.80	33.42	25.35
* This memory is shared between the data and program blocks			

Figure 2: Processor comparison chart

In selecting a processor, three qualities were weighted heaviest. Since the edge detection device must be low-cost, the processor should have a minimal price. To improve the power consumption of the device the processor must have a "Stand-by" mode. Lastly, to allow for computationally intense algorithms, the processor must be fast. Given these considerations the ADI ADSP-2181 was selected as the new processor for the edge detection device.

Since all processors had onboard serial ports, this criterion was ignored. However, as the team learned only 3 weeks before the end of the project, there is a distinction between a serial port and a USART. The serial port does not implement any timing mechanisms and therefore cannot be used to reliably communicate with a computer. It is not clear that having this knowledge would have made a difference when selecting the processor, however, with only 3 weeks to go the team was left with few alternatives. Analog Devices provides technical documentation [ii] in which the team found assembly code to simulate a USART using the onboard timer and the serial port. This code can be found in Appendix C: USART Assembly Code.

With only a few weeks remaining, restrictions of the development hardware were discovered. The standard development tool for the ADSP-2181 is the EZ-Kit Lite board, a schematic of which can be seen in Figure 3. It is designed to be only an evaluation product and cannot function as a stand alone processor. This imposes several restrictions. Most importantly, both program and data memories of the ADSP-2181 are halved in size down to only 8kB each[vii]. This did not affect the project from a program memory standpoint because that code was relatively small. However, the data memory forced the team to reevaluate the fundamentals of the

implementation of the edge detection algorithm. Please see the following section for more details.

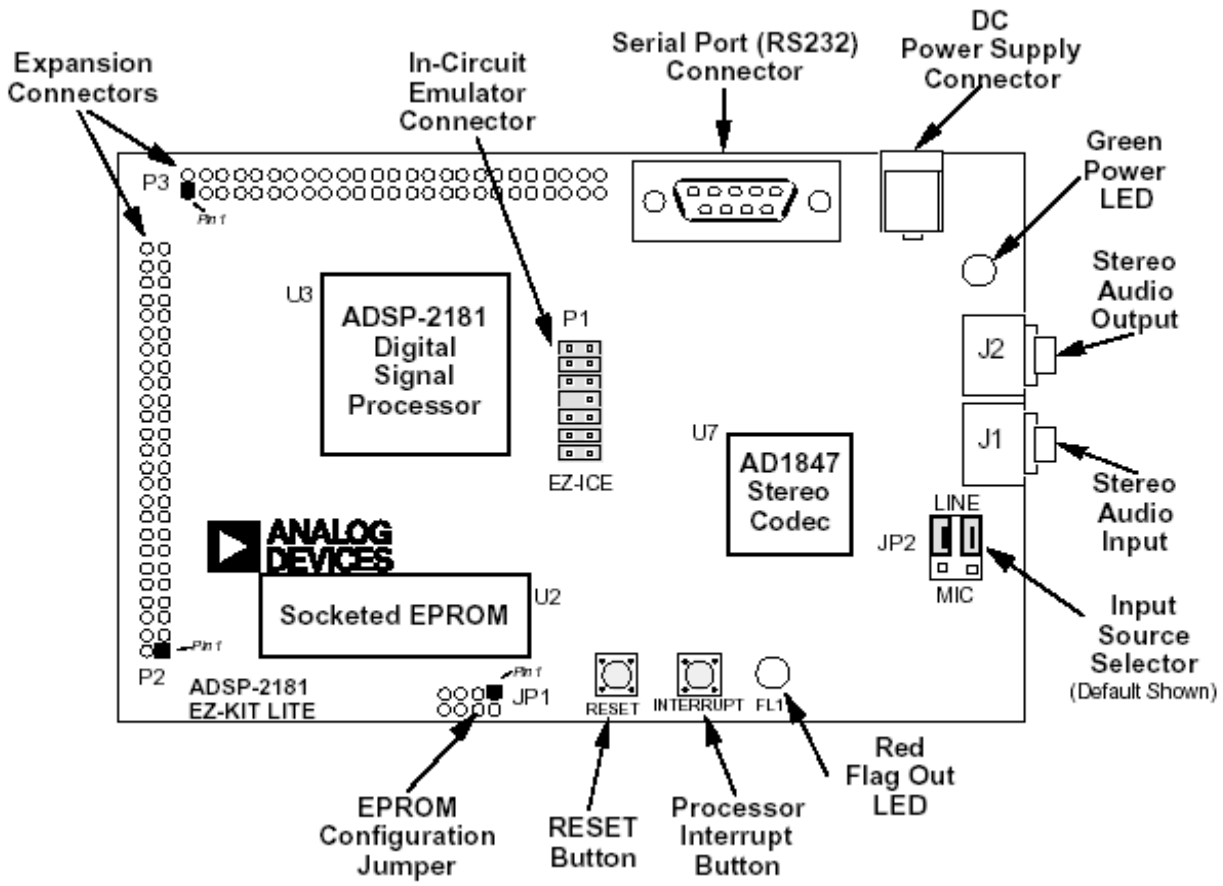


Figure 3: EZ-Kit Lite Development Board

The second restriction imposed by the EZ-Kit Lite is the speed at which both data and program memories can be uploaded onto the DSP chip. The board is limited to 9600 baud communication rate, whereas the DSP onboard serial port can operate at speeds up to 115 kbps[viii]. Along these same lines, the EZ-Kit Lite is restricted by the onboard oscillator, which serves as the clock for all hardware components on the development board. The oscillator is only 16.7MHz whereas the ADSP-2181 can operate at 33MHz.

Lastly, as it was discovered during a conversation with Analog Devices Technical Support, the EZ-Kit Lite board is restricted to only properly work with the development studio (see Verification section). In order to make the DSP execute the program code, there must be a way of transferring the program memory onto the chip. The nature of the development board restricts

this functionality. Thus, this development board can only be used to design and verify the assembly code needed for the mass production product, but not execute the code on the hardware in realtime.

Software Design

As Figure 1 shows, after acquiring the image the camera must transfer the data serially to the edge detection device. To get this data, the device needs to be USART capable. As it was stated above, this capability is missing from the ADSP-2181 and had to be implemented in software (See Appendix C: USART Assembly Code).

Once the data is in the processor memory, the actual process of edge detection can begin. Two schemes for doing so were considered (See Figure 4).

Method 1	Method 2
1. Store all raw data 2. Process all raw data, storing edge data 3. Output all edge data	1. Process a pixel for edges 2. Output edge data 3. Acquire next pixel of raw data

Figure 4: Comparison of edge detection algorithms

Method 1 is easy to implement, however, it is also very memory intensive because it requires storing all raw and edge data within the onboard memory. More specifically, the selected camera has 324x243 8 bit/pixel resolution. The raw data alone takes up over 75kB of memory. Since the ADSP-2181 does not have enough memory, and last semester's memory interface is no longer a part of this design, Method 1 was unfeasible.

On the other hand, Method 2 does not require storage of any significant amount of data (See Edge Detection section for details). However, this convenience comes at the cost of stringent timing restrictions. The camera, being completely independent of the processor, outputs the raw data regardless of whether the processor is ready for it or not. Thus, to prevent loss of data, the processor must completely finish edge detection calculations on one pixel in time to receive the next.

Edge Detection

There are several standard methods for edge detection all of which rely on the fact that an edge normally occurs when there is a change in pixel intensity. Let's consider the following two pictures (See Figure 5).

Vertical Edge **Horizontal Edge**

Figure 5: Locating simple edges

Consider attempting to find an edge on either of the above pictures. Let P_n represent grayscale intensity value at pixel n. Then, $|P_2 - P_1|$ will give some nonzero answer. Conversely, taking the difference of any two points both of which are located on the white side will produce a zero change in intensity. Similarly, the difference of any two points both of which are located on the gray side will produce zero. Thus, the only way that P_2 can have a different intensity value from P_1 is if there is an edge between those two points.

This procedure works perfectly for edges that are exactly vertical or exactly horizontal. However, this happens rather seldom in images. The above procedure can be modified to accommodate for any angled edge.

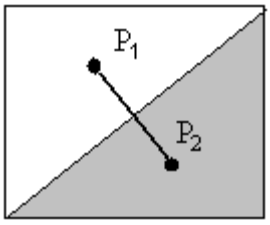


Figure 6: Locating diagonal edge

Figure 6 shows two points, one on either side of the edge. Taking the difference of these two points would once again result in an edge. But why were the points placed where they were? If the two points are picked so that the line connecting them is perpendicular to the edge in question, then those two points will always correctly determine the presence of an edge. This

method, even though simple in calculations, is not effective on images where one simply does not know where the edges are. How can one draw a perfect perpendicular to such an edge?

This problem can be easily overcome, but at the cost of doing extra calculations. Consider once more the calculations performed in Figure 5. Since notions of vertical and horizontal are always defined, let's perform the calculations based on these knowns (See Equation 1).

$$\frac{\left|P_2 - P_1\right|_{horizontal} + \left|P_2 - P_1\right|_{vertical}}{2}$$

Equation 1: Basic 2 dimensional edge detection

First, perform the calculation between the two horizontal points, then for the two vertical points, and take the average of the results. It is this average which determines the presence of any angled edge.

This is precisely the method that famous edge detection algorithms like the Prewitt[ix], Sobel[x] and Laplacian[xi] methods are based upon. The difference between the three methods is simply the accuracy of detecting edges and ignoring the noise within the image. This is possible at the cost of more calculations. For example, let's do two vertical and two horizontal calculations instead of one (See Figure 1).

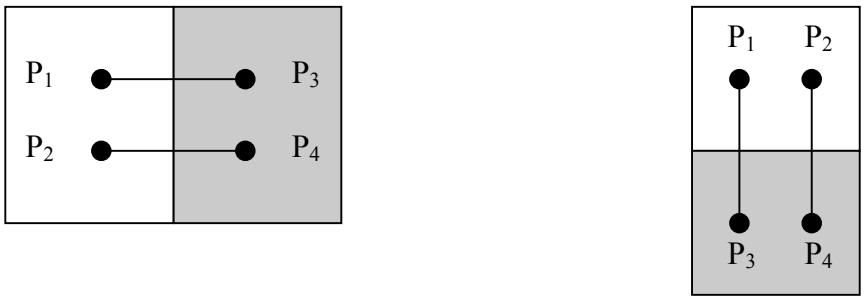


Figure 7: 2 x 2 Prewitt method for edge detection

This scheme is called the Prewitt method. It was selected as the method for edge detection in this project because it provides a drastic improvement in quality of detected edges. However, this improvement comes at the cost of four extra arithmetic steps. The Prewitt method can be

used to provide more accurate results if it is applied in a 3 x 3 format (3 vertical, 3 horizontal calculations). Even more accuracy can be achieved if the simple averaging is replaced by the more accurate Root Mean Square (RMS) technique.

This style of edge detection can be thought of as a system where an image (input) is applied to an edge detection device (filter) producing a map of the edges (output). Such applications can be mathematically represented by convolution and numerically symbolized by matrix operations. For the 3 x 3 Prewitt method, the input can be symbolized by Figure 8. Let C columns and R rows of an image be symbolized as

$$\begin{matrix} P_{11} & P_{21} & P_{31} & \dots & P_{C1} \\ P_{12} & P_{22} & P_{32} & \dots & P_{C2} \\ P_{13} & P_{23} & P_{33} & \dots & P_{C3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P_{1R} & P_{2R} & P_{3R} & \dots & P_{CR} \end{matrix}$$

Figure 8: Input into Prewitt edge detection filter

and the filter transformation matrices be symbolized by Figure 9.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Horizontal Vertical

Figure 9: Transformation matrix for 3 x 3 Prewitt method of edge detection

Then the presence of an edge can be established by performing the calculations in Equation 2,

$$X = P_{31} - P_{11} + P_{32} - P_{12} + P_{33} - P_{13}$$
$$Y = P_{11} + P_{21} + P_{31} - P_{13} - P_{23} - P_{33}$$
$$Edge = \sqrt{X^2 + Y^2}$$

Equation 2: Arithmetic steps involved in 3 x 3 RMS Prewitt edge detection method

where *X* and *Y* are the results of applying the horizontal and vertical transformation masks respectively and *Edge* is the Prewitt value of the pixel. An edge would be present at this pixel if the Prewitt value exceeded some predefined threshold value which is defined in Equation 3.

$$Threshold = 8 * P^2$$

Equation 3: Threshold Value Calculation

This equation was derived by applying the 3 x 3 Prewitt method onto an image with a vertical edge and an intensity difference of *P*. The initial value for *P* was obtained from an empirical observation. The team drew different colored gray squares on a white background. When the square was almost invisible the team calculated the necessary threshold value to barely detect edges on that square. That value was 24,200 which corresponds to a square whose intensity differs by approximately 22% from the background. The threshold value was later modified to be more sensitive. After several trials of detecting edges on real photographs, it was determined that a sensitivity of 12% was sufficient setting the threshold value to 7200.

It must be noted that this method ignores any edges located anywhere one pixel within the perimeter of the image. Since the 3 x 3 Prewitt method detects the presence of an edge only at the center pixel of the matrix, it is impossible to detect edges on the periphery of the image. This drawback can be overcome by inserting an exact replica of the first row of data above the actual image (and likewise for all other sides), allowing for edge detection on the periphery of the real image. However, this adds an extra degree of complication to the edge detection algorithm without producing any extraordinary results. Thus, the team opted to simply ignore the presence of any edges on the periphery of the image.

It must also be noted that to compute the very first edge pixel, the algorithm requires sufficient data for the 3 x 3 matrix. Since the camera outputs the data serially one row at a time, this means that the edge detection device must acquire at least two complete rows and three pixels from the third row of data of the raw image in order to perform the first calculation. By default, this means that the DSP must be able to store at least 3 rows of raw data. The team designed the algorithm with memory considerations in mind, by allocating a buffer large enough for exactly 3 rows. When more data comes in, the buffer simply overwrites the oldest data point.

Counting the number of arithmetic steps in Equation 2, it can be seen that in order to find an edge in one pixel, 11 additions, 2 multiplications, 1 square root, and 1 comparison must be performed. Since it can be approximated that each of these arithmetic functions takes one full clock cycle, 15 cycles per pixel are required. The selected CCD camera has a resolution of 324 x 243 pixels. Since edges cannot be found anywhere on the perimeter of the image, 322*241*15 = 1,164,030 cycles are needed to process an entire image. With an operational speed of 40 MIPS, the ADSP-2181 can process this image in 29ms.

The 29ms does not reflect the time needed to acquire the image through the serial port nor the time needed to load and store any intermediate values. After examining the final version of the assembly code, the team calculated the total number of assembly instructions needed to process an entire image to be 17,295,429. At the operational speed of 40 MIPS, ADSP-2181 can execute these steps in 432ms. This equates to the processing capability of just over two images per second. Please refer to Appendix D: Edge Detection Assembly Code for the edge detection algorithm implementation.

Communication

As it was seen in Figure 1, the edge detection device must have bidirectional serial communication capabilities to both acquire the raw data and output the corresponding edge data. This is an almost trivial task when following the USART standards. However, as it was mentioned above, the ADSP-2181 does not have this implemented in hardware. Thus, a software USART was adapted from the Application Notes available on the Analog Devices webpage[xii]. This code can be seen in Appendix C: USART Assembly Code.

Once this was implemented, the functionality of transmitting and receiving data was complete. Besides the timing restrictions on acquiring data already mentioned above, very little had to be done with the input from the camera. The output, on the other hand, was somewhat more difficult. Timing was obviously of concern. Since much of the timing is spent on the software USART implementation, it was decided that to improve throughput of the device, the output needed to be compressed in order to reduce the number of bytes sent through the serial port.

After careful consideration for the kind of data that was going to be compressed, the following observations were made:

1. There are only two possible values for a pixel in the map of detected edges (edge image); an edge and a no-edge (blank).
2. It is highly unlikely that edges and blanks would alternate every other pixel. Normally, there would be a sequence of edges followed by a sequence of blanks.

The team came up with a simple, yet effective scheme for compressing this kind of data. Since the edge image is essentially binary, it is possible to allocate one bit as a representation of an edge or a blank. Then the remaining seven bits of a byte can be used as a count of repeating occurrences of that pixel (See Figure 10).

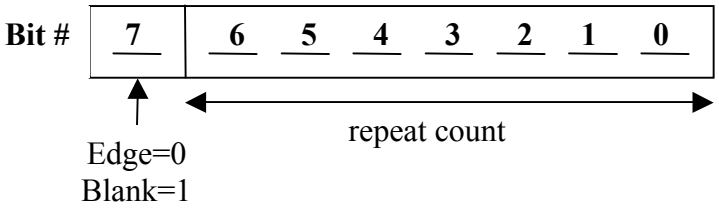


Figure 10: Composition of compressed edge data byte

To demonstrate the power of this compression algorithm lets consider the following sequence of edge data: 64 blanks followed by 12 edges followed by 32 blanks. To transmit this data without compression one byte at a time would require 64+12+32=108 bytes. However, after applying the edge compression algorithm, the edge data is reduced down to only 3 bytes: 01000000_2 representing the leading 64 edges, 10001100_2 representing the 12 edges, and 00100000_2 representing the trailing blanks. For this example, the compression ratio is 36:1.

The theoretical limit of this compression algorithm is 127:1. This is simply due to the fact that this scheme allocates only seven bits for the repeat count. Even though seven bits can represent 128 unique values, having zero repeating blanks or edges does not make much sense, leaving only 127 appropriate values. It is possible to reach this theoretical limit for an image having no edges (or rather an image where all intensity transitions are smooth, like a blurry picture). However, during experimentation, it was discovered that an average compression ratio for a real

image is on the order of 60:1. This value, of course varies with the number of detected edges. For an image having a large edge perimeter, the compression could be 20:1 or even smaller.

Verification

Since the hardware components were already designed and constructed on the development board, no hardware verification was needed. To verify the software, the team adopted the standard code development and debugging techniques.

The primary tool for code development was VisualDSP++ 2.0, which is an Integrated Development Environment (IDE) provided by Analog Devices[xiii]. Some of its key features include a full featured Microsoft Visual Studio style text editor, ample debug options (stepping, running, disassembly), and other graphical tools (linker, loader). Code development with this tool was almost pleasant.

However, these features were not free. The full version of this software costs $4,500. The team, however, was able to obtain a 30 day evaluation license, which, upon expiring, caused the progress of the project to come to a screeching halt. Nevertheless, throughout the duration of the evaluation license, the team developed code for serial data acquisition, edge detection, and output compression. The complete implementation of this code was emulated within VisualDSP++. The emulation took almost 25 hours; however, it was an important step in the validation process.

To assist in the design process, a custom tool was especially developed. Since Paul Grinberg had prior experience in creating Graphical User Interfaces (GUI) programs using Perl/Tk[xiv], this was the obvious choice for the emulation environment. Originally intended only as the TC255 camera chip emulator, it evolved into much more. The finished product (see Appendix E: TC255 Emulation Code for code) could not only simulate every step of edge detection, but also allowed for a visual check of the correctness of the output. Here is an example.

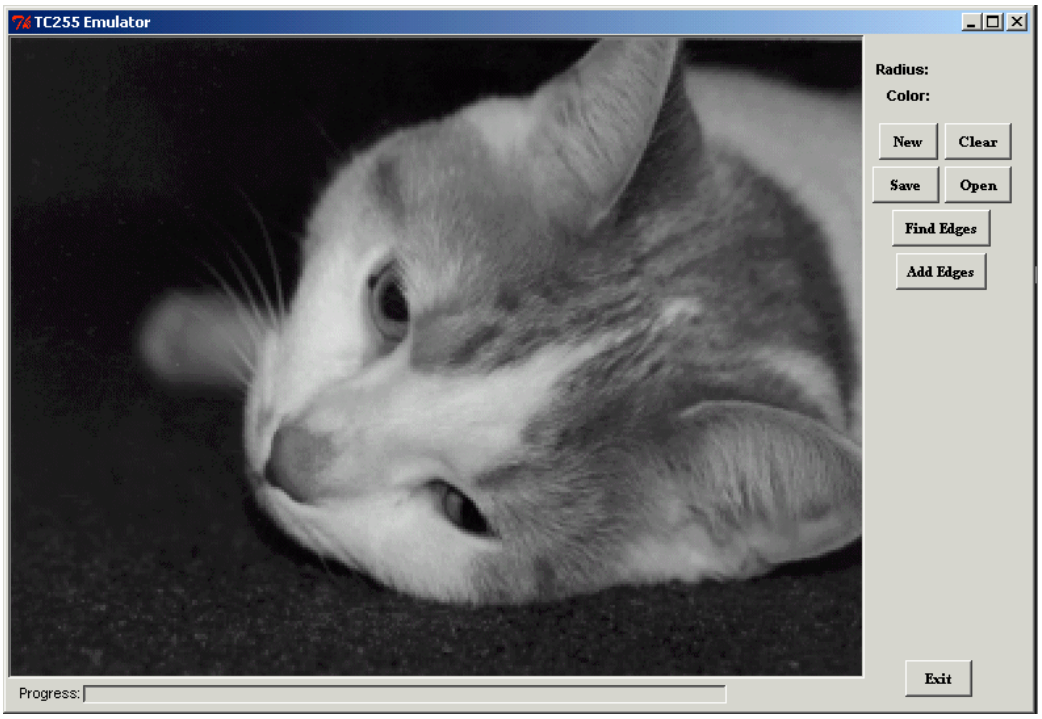


Figure 11: TC255 Emulator – image to process for edges

Let's assume that the picture in Figure 11[xv] is an image freshly acquired by the camera. The next step is to transfer it to the DSP and process it for edges. Thus, when the "Find Edges" button is clicked, the emulation begins. For reasons stated above, the first two rows and 3 pixels from the third row are passed to the edge detection routine before any edge detection can take place. Then, just like in the DSP, edge detection proceeds pixel by pixel, compressing and outputting the edge data as necessary. Finally, when edge detection is completed, the detected edges are overlaid onto of the original image for visual inspection (see Figure 12).

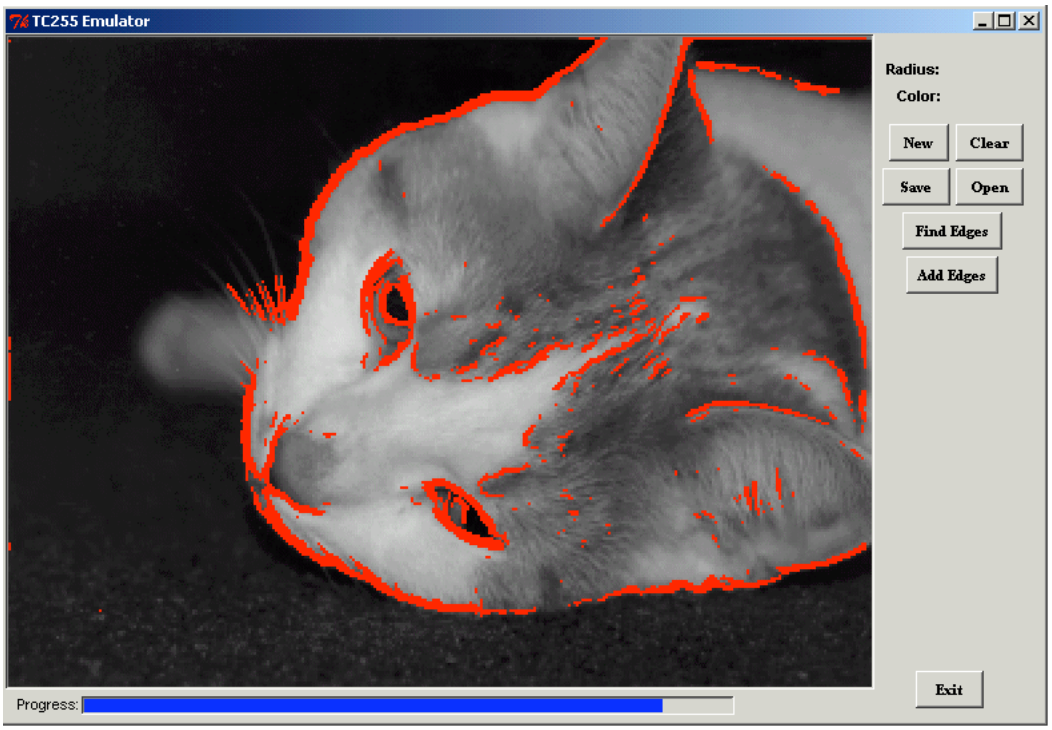


Figure 12: TC255 Emulator – visual inspection of detected edges

For final inspection, it is possible to completely remove the original image, leaving only the detected edges as seen if Figure 13.

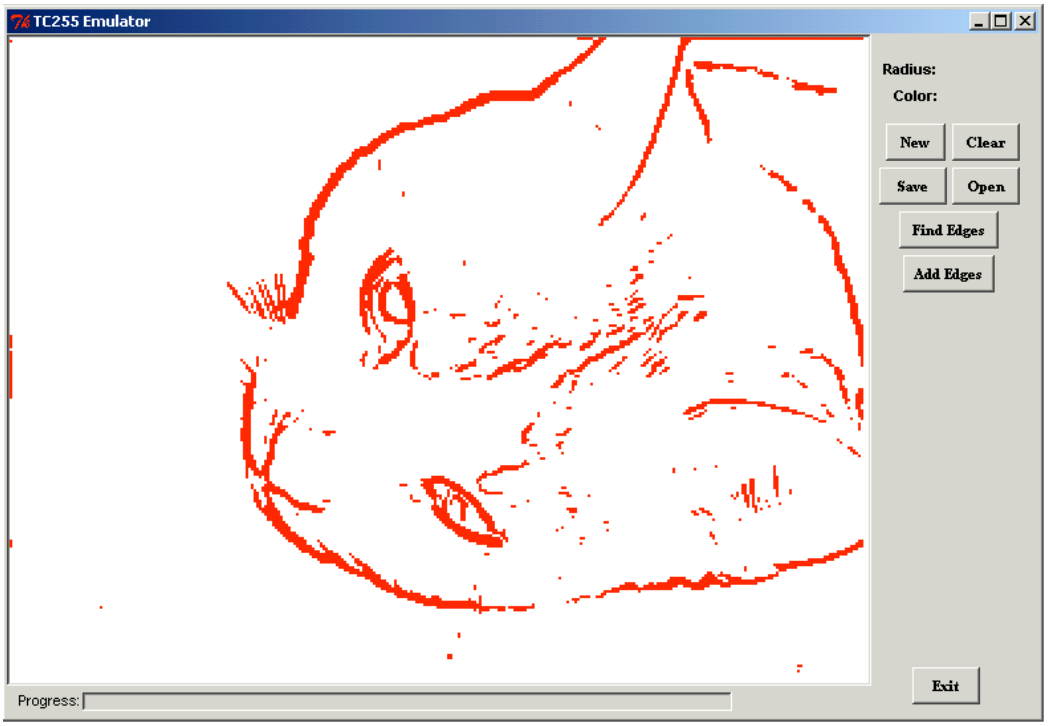


Figure 13: TC255 Emulator - displays detected edges

Results

The objective of creating a low-cost portable edge detection device is currently completed with one restriction. The device was intended to operate as a standalone unit. However, as was explained above, the restrictions of the development board prohibit this requirement from completion. In order to demonstrate operability of the edge detection device, a computer is needed not only to model the camera input as was initially intended, but also for simulation purposes of the DSP execution. Nonetheless, the developed code is ready for mass production given the appropriate hardware.

As compared to the original specifications, this product fulfills all other requirements:

- Low power consumption – as per the specifications of the ADSP-2181 the processor uses 425mW and the camera uses approximately 100mW. With the remaining glue logic pulling no greater than another 100mW, the total power consumption of the device is less than 650mW. Given the standard AA batteries, the device can operate for over three hours of continuous operation. With more advanced batteries such as Litium Ion batteries, continuous operation time can be extended to well over a day. This completely meets the specified requirements.
- Low-cost – since the cost of the development board cannot be counted into the cost of manufacture, only the following components should be considered:

Part	Quantity	Unit Cost (USD)
ADSP-2181-KS133	Per 100	25.35
TC255P	Per 100	35.40
Glue Logic		12.13
	Total:	**72.88**

Figure 14: Price breakdown of components in the edge detection device

This price is only slightly larger than the anticipated 50-60 dollars. Since these prices were obtained from a local distributor of Analog Devices and Texas Instruments parts, it is likely that the unit cost would decrease in bulk quantities when obtained directly from the manufacturers.

- Portable – the device is portable in the sense that all hardware components fit on one small circuit board. Such boards are easily fitted into appropriate packaging.
- Fast – as was shown in the Edge Detection section, the device can process a little more than two full images or frames per second (fps). This is significantly less than the desired 5 to 10 fps but can be attributed to two factors. Firstly, the device uses a 3 x 3 Prewitt method for edge detection. This is a more advanced method producing higher quality results at the cost of more computations. If a different edge detection method is utilized, for example a 2 x 2 Prewitt method, the number of frames per second increases to 6 fps. Secondly, the Analog Devices development board is limited by the onboard oscillator, which runs at only 16.7MHz. The ADSP-2181 can operate at speeds up to 33MHz. If the processor was to operate at the proper frequency, the throughput of the device would increase to just shy of 5 fps even with the 3 x 3 Prewitt method, once again satisfying the requirement.
- Works with a specific camera – the functionality of the edge detection device was built around the specifications of a Texas Instruments TC255 CCD camera.

Two software tools were used to verify the functionality of the edge detection device. For development of the code and simulation and emulation of the execution of the code on the DSP, the team utilized VisualDSP++ 2.0. This software is provided by Analog Devices to developers, assisting them in the design process. Upon completion of the software design, this tool was used to emulate the entire edge detection process. The emulation completed in approximately 25 hours, confirming the functionality of the code.

The second tool was a program written in Perl/Tk designed especially for visual display and inspection of the output of the VisualDSP++ emulation. Even though the verification process was based on visual inspection, Figure 11 through Figure 13 clearly demonstrate the success of the project.

To summarize, the team has successfully completed the edge detection device project including the following milestones: camera selection and emulation, edge detection design, and

verification. For details on other tasks throughout the project, please see Appendix B: Revised Gantt Chart.

Implications

The device is extremely economical because it is a low-cost device. Specifically, it will be geared towards individuals and industries interested in image processing, and in particular, edge detection. It has been proposed that such a device be implemented in an autonomous robot or modified into a smart motion sensor. This application could be one of many to impact an average consumer.

All components chosen for this device are currently in production, and, as such, are easily accessible in large quantities. However, due to the nature of the semiconductor industry, this sustainability is not reliable. If the ADSP-2181 is taken out of production a similar ADSP-21xx may be used instead without any repercussions. Likewise, if the CCD camera is taken off the market, a similar CCD camera may be selected. However, this change may require algorithm modifications.

All hardware components come in standard Plastic Quad Flatpack Package (PQFP) and Dual-In-Line Package (DIP). All manufacturing facilities are equipped to print circuit boards with these components. Since there is no need for any additional buses, printing the circuit boards should be straightforward.

This device is safe because there are no open wires nor do any contain a high voltage or current from which a person may receive an electric shock.

Conclusions

The team has met the requirements listed in the previous section as well the necessary tasks needed to complete the project. The team, however, experienced several major problems during the course of this project.

The first difficulty arose when the team discovered that the main processors intended for use in the project were incapable of handling the computationally intense task of edge detection. To resolve this issue, the team opted to redesign the project to utilize a different processor. The new design was based on the suggestions of Professor Frank Merat and Professor Stephen Phillips. The next task was to locate the development boards necessary for DSP development. For this, the team made use of Professor Ken Loparo's generosity, and acquired a Texas Instruments development board. Professor Merat was also kind enough to lend the team an Analog Devices development board for comparison. After thorough evaluation of the two development boards, the team decided to use the Analog Devices board.

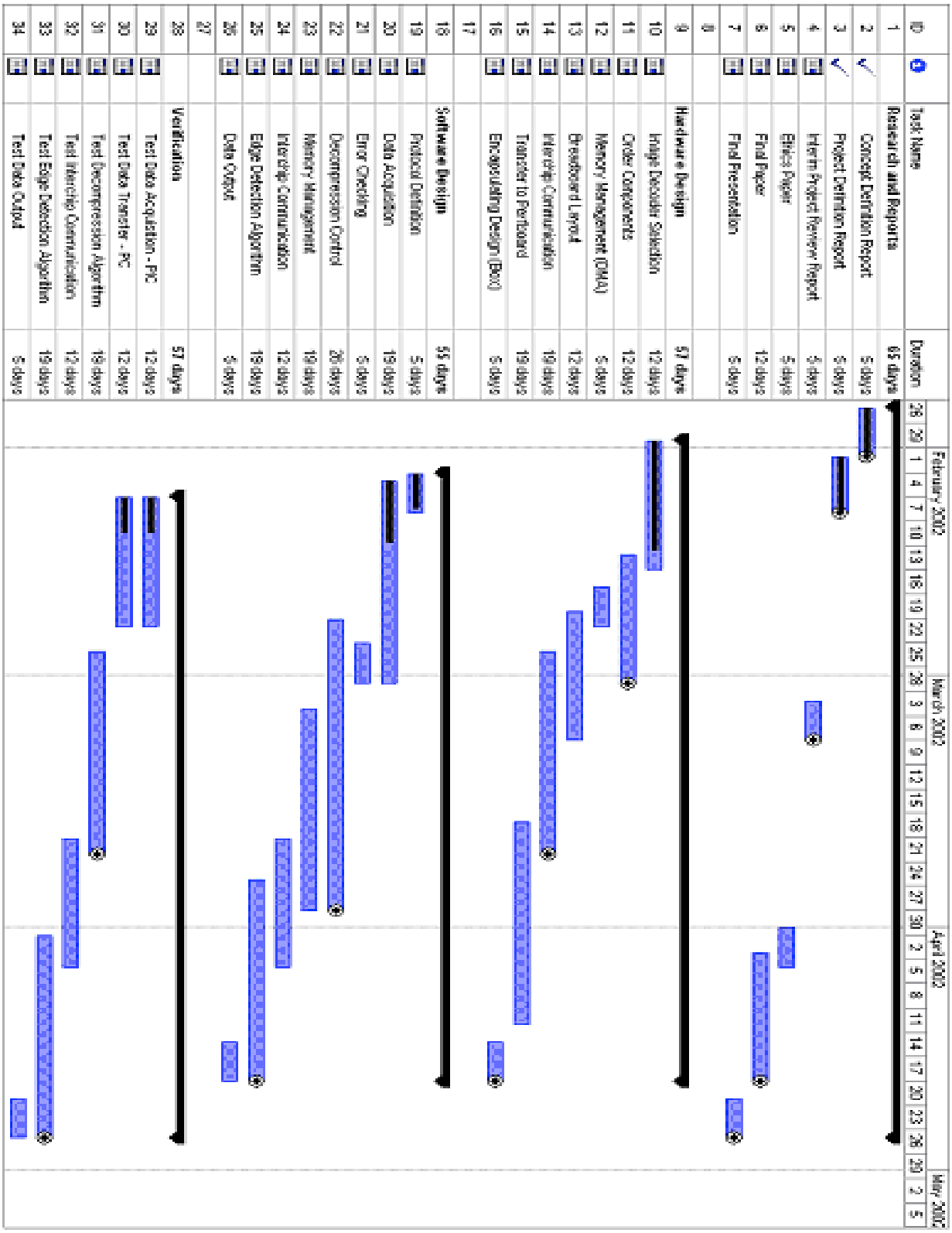
Problems with code development resulting from the teams lack of knowledge of the new assembly language were resolved by help from Mark Koob who has had several years of working experience at Analog Devices. Further problems with the development board, in particular the teams inability to force the board to function in a standalone mode, were worked out by the Analog Devices Technical Hotline who explained that this was impossible (see Processor section).

Recommendations

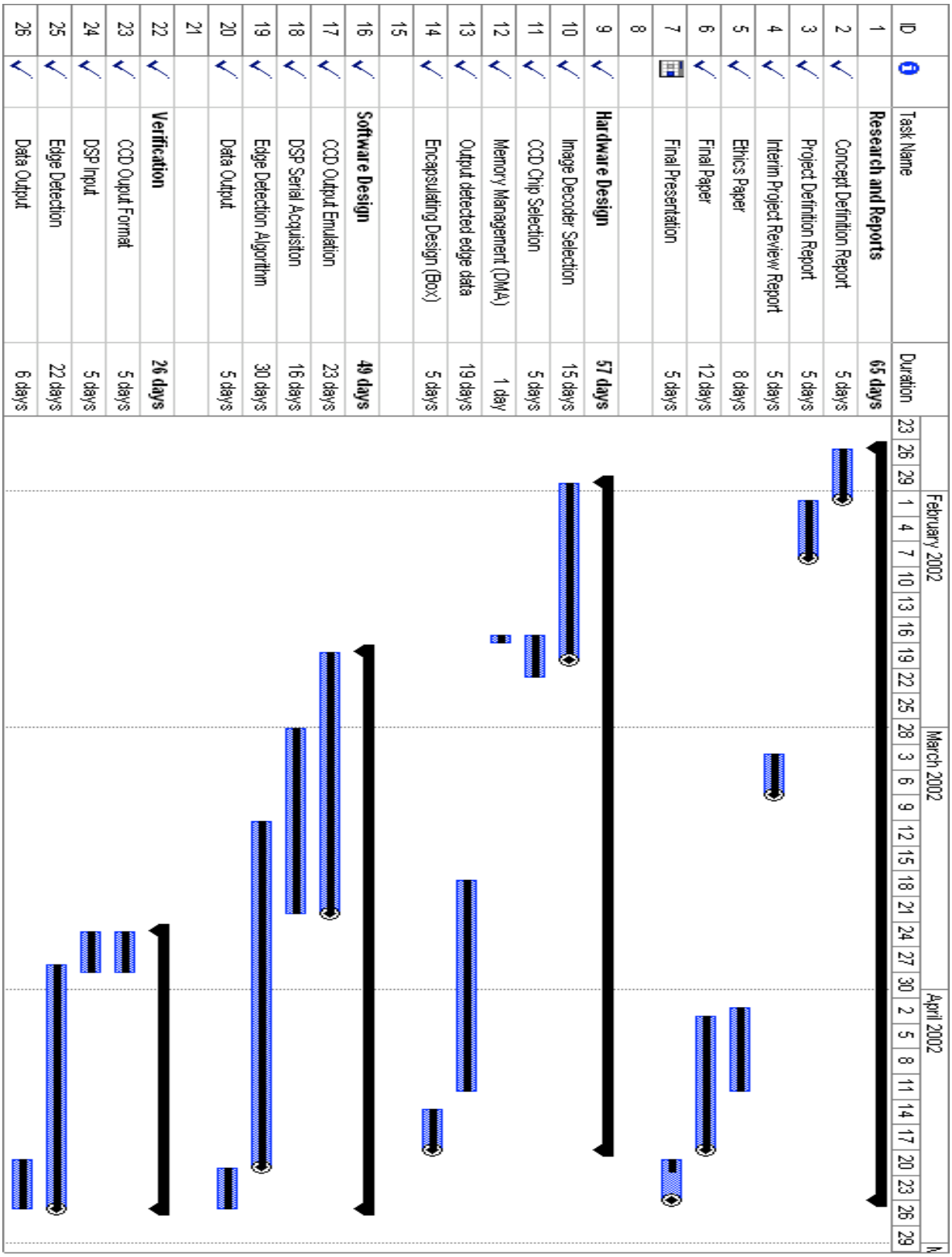
Even though the edge detection device is completed as per the specifications of this project, the device is not ready for mass production. The device is not ready for mass production because currently it still relies on the computer for the edge detection algorithm execution. The next step in this project should be to develop a manufacturable equivalent of this device. This task includes designing a new circuit board to include many of the components of the EZ-Kit Lite without any of its restriction. The circuit board design should incorporate the CCD camera, eliminating the need for any emulation.

To improve this project, a faster processor can be implemented. The current ADSP-2181 can operate at speeds at up to 33MHz. For example, ADSP-2191M is fully compatible with the ADSP-2181, but can operate at speeds up to 160MHz permitting greater throughput of the edge detection device without any additional development.

Appendix A: Original Gantt Chart



Appendix B: Revised Gantt Chart



Appendix C: USART Assembly Code

```
/*************************************************************************

       ADSP-2101 EZLAB        UART Example          AUTOECHO.DSP

     This program utilizes the UART code listed in Appendix 1 and provided
on the disk (UART.DSP) to provide a simple example of how to use the UART
monitor.  This program reads a character in, and writes (echos) it back out.
The program also utilizes the Autobaud capability, described and listed in
Appendix 2 and modified slightly for this example (the modified code is also
provided on the disk [AUTOECHO.DSP]).
     The only hardware required is an ADSP-2101 EZ-LAB Board and an interface
board with an RS-232 line driver chip connected to the FlagIn and FlagOut
pins on the J2 Sport Connector (you must supply the incoming signals to the
line driver chip).  As an alternative, a 21020 EZ-LAB Board could be used,
since it already has all of the necessary hardware (ADSP-21xx and RS-232 line
driver chip -- however, in this case, the interrupt vector table would have
to be modified for ADSP-2111 requirements), and connected to a terminal, such
as an IBM PC running PROCOMM (again, you supply the input data -- via PROCOMM
in this case).

      Author:    Philip Holdgate, Analog Devices, 03-Apr-1992.
      Modified:  Brian Baker, Analog Devices, 27-Apr-1992.

*************************************************************************/
#define  tx_num_of_bits 10;   // start bits + tx data bits + stop bits
#define  rx_num_of_bits 8;    // rx data bits (start&stop bits not counted)
#define  RX_BIT_ADD 0x0100;   // = 1<<rx_num_of_bits
#define  TX_BIT_ADD 0xfe00;   // = 0xffff<<(tx data bits+1)
#define  PERIOD 74;           //13&57600// PERIOD = (Proc_frequency/(3*Baudrate))-1

.SECTION/DATA data1;
.var    flag_tx_ready;        // flag indicating UART is ready for new tx word
.var    flag_rx_ready;        // flag indicating UART is ready to rx new word
.var    flag_rx_stop_yet;     // flag tells that a rx stop bit is not pending
.var    flag_rx_no_word;      // indicates a word is not in the user_rx_buffer
.var    flag_rx_off;          // indicates a that the receiver is turned off
.var    timer_tx_ctr;         // divide by 3 ctr, timer is running @ 3x baudrate
.var    timer_rx_ctr;         // divide by 3 ctr, timer is running @ 3x baudrate
.var    user_tx_buffer;       // UART tx reg loaded by user before UART xmit
.var    user_rx_buffer;       // UART rx reg read by user after word is rcvd
.var    internal_tx_buffer;   // formatted for serial word, adds start&stop bits
                              // 'user_tx_buffer' is copied here before xmission
.var    internal_rx_buffer;
.var    bits_left_in_tx;      // number of bits left in tx buffer (not yet clkd out)
.var    bits_left_in_rx;      // number of bits left to be rcvd (not yet clkd in)
.var    baud_period;          // loaded by autobaud routine

.SECTION/PM interrupts;
       JUMP START; RTI; NOP; NOP;           // Reset Vector
       RTI; NOP; NOP; NOP;                  // IRQ2 Interrupt
       RTI; NOP; NOP; NOP;                  // SPORT0 Transmit Interrupt
       RTI; NOP; NOP; NOP;                  // SPORT0 Receive Interrupt
       RTI; NOP; NOP; NOP;                  // SPORT1 Transmit Interrupt
       RTI; NOP; NOP; NOP;                  // SPORT1 Receive Interrupt
       JUMP process_a_bit; RTI; NOP; NOP;   // Timer Interrupt


//                          Initialization Routine
.SECTION/PM program;
START:  DM(baud_period) = AR;               // UART Autobaud
       CALL init_uart;                      // Initialize UART
       CNTR = 15000;                        // Wait approximately one
       DO XLOOP UNTIL CE;                   // character to insure last one
XLOOP:   NOP;                               // made it through
       CNTR =15000;
```

```
        DO YLOOP UNTIL CE;
YLOOP:   NOP;

        CALL turn_rx_on;                        // Enable UART Receive


//                              Main System Loop

        DO MLOOP UNTIL FOREVER;
           CALL get_char_ax1;                   // Read in character
           CALL out_char_ax1;                   // and Echo it back out
MLOOP:   NOP;

init_uart:
    ax0=0;
    dm(0x3ffb)=ax0;                             // decrement TCOUNT every instruction cycle

    ax0=dm(baud_period);                        // from autobaud or use constant: ax0=PERIOD;
                                                // ...and comment in the appropriate constant
    dm(0x3ffc)=ax0;
    dm(0x3ffd)=ax0;                             // interrupts generated at 3x baudrate
    ax0=0;
    dm(0x3fff)=ax0;                             // no bmwait, pmwait states, SPORT1 = FI/FO

    ax0=1;
    dm(flag_tx_ready)=ax0;                      // set the flags showing that UART is not busy
    dm(flag_rx_ready)=ax0;
    dm(flag_rx_stop_yet)=ax0;
    dm(flag_rx_no_word)=ax0;
    dm(flag_rx_off)=ax0;                        // rx section off

    set flag_out;                               // UART tx output is initialized to high
    ifc=0x003f;                                 // clear all pending interrupts
    nop;                                        // wait for ifc latency
    imask=b#000001;                             // enable TIMER interrupt handling
    ena timer;                                  // start timer now
    rts;

process_a_bit:
    ena sec_reg;                                // Switch to the background dreg set
    ax0=dm(flag_tx_ready);                      // if not in "transmit", go right to "receive"
    ar=pass ax0;
    if ne jump receiver;
                              //_________Transmitter Section________
    ay0=dm(timer_tx_ctr);                       // test timer ctr to see if a bit
    ar=ay0-1;                                   // is to be sent this time around
    dm(timer_tx_ctr)=ar;                        // if no bit is to be sent
    if ne jump receiver;                        // then decrement ctr and return

    sr1=dm(internal_tx_buffer);                 // shift out LSB of internal_tx_buffer
    sr=lshift sr1 by -1 (hi);                   // into SR1.  Test the sign of this bit
    dm(internal_tx_buffer)=sr1;                 // set or reset FLAG_OUT accordingly
    ar=pass sr0;                                // this effectively clocks out the
    if ge reset flag_out;                       // word being xmitted one bit at a time
    if lt set flag_out;                         // LSB out first at FLAG_OUT.

    ay0=3;                                      // reset timer ctr to 3, i.e. next bit
    dm(timer_tx_ctr)=ay0;                       // will be sent after 3 timer interrupts

    ay0=dm(bits_left_in_tx);                    // number of bits left to be xmitted
    ar=ay0-1;                                   // is now decremented by one,
    dm(bits_left_in_tx)=ar;                     // indicating that one is now xmitted
    if gt jump receiver;                        // if no more bits left, then ready

    ax0=1;                                      // flag is set to true indicating
    dm(flag_tx_ready)=ax0;                      // a new word can now be xmitted

                               //_________Receiver Section________
receiver:
    ax0=dm(flag_rx_off);                        // Test if receiver is turned on
    ar=pass ax0;
```

```
    if ne rti;

    ax0=dm(flag_rx_stop_yet);           // Test if finished with stop bit of
    ar=pass ax0;                        // last word or not. if finished then
    if ne jump rx_test_busy;            // continue with check for receive.

    ay0=dm(timer_rx_ctr);               // decrement timer ctr and test to see
    ar=ay0-1;                           // if stop bit period has been reached
    dm(timer_rx_ctr)=ar;                // if not return and wait
    if ne rti;

    ax0=1;                              // if stop bit is reached then reset
    dm(flag_rx_stop_yet)=ax0;           // to wait for next word
    dm(flag_rx_ready)=ax0;

    ax0=dm(internal_rx_buffer);         // copy internal rx buffer
    dm(user_rx_buffer)=ax0;             // to the user_rx_buffer

    ax0=0;                              // indicated that a word is ready in
    dm(flag_rx_no_word)=ax0;            // the user_rx_buffer
    rti;

rx_test_busy:
    ax0=dm(flag_rx_ready);              // test rx flag, if rcvr is not busy
    ar=pass ax0;                        // receiving bits then test for start.If it
    if eq jump rx_busy;                 // is busy, then clk in one bit at a time

    if flag_in jump rx_exit;            // Test for start bit and return if none

    ax0=0;
    dm(flag_rx_ready)=ax0;              // otherwise, indicate rcvr is now busy
    dm(internal_rx_buffer)=ax0;         // clear out rcv register

    ax0=4;                              // timer runs @ 3x baud rate, so rcvr
    dm(timer_rx_ctr)=ax0;               // will only rcv on every 3rd interrupt
                                        // initially this ctr is set to 4.  This
                                        // will skip the start bit and will
                                        // allow us to check FLAG_IN at the center
                                        // of the received data bit
    ax0=rx_num_of_bits;
    dm(bits_left_in_rx)=ax0;
rx_exit:
    rti;

rx_busy:
    ay0=dm(timer_rx_ctr);               // decrement timer ctr and test to see
    ar=ay0-1;                           // if bit is to be rcvd this time around
    dm(timer_rx_ctr)=ar;                // if not return, else receive a bit
    if ne rti;

rcv:                                    // Shift in rx bit
    ax0=3;                              // reset the timer ctr to 3 indicating
    dm(timer_rx_ctr)=ax0;               // next bit is 3 timer interrupts later

    ay0=RX_BIT_ADD;
    ar=dm(internal_rx_buffer);
    if not flag_in jump pad_zero;       // Test RX input bit and
    ar=ar+ay0;                          // add in a 1 if hi

pad_zero:
    sr=lshift ar by -1 (lo);            // Shift down to ready for next bit
    dm(internal_rx_buffer)=sr0;

    ay0=dm(bits_left_in_rx);            // if there are more bits left to be rcvd
    ar=ay0-1;                           // then keep UART in rcv mode
    dm(bits_left_in_rx)=ar;             // and return
    if gt rti;                          // if there are no more bits then..

                                        // That was the last bit
    ax0=3;                              // set timer to wait for middle of the
    dm(timer_rx_ctr)=ax0;               // stop bit
```

```
    ax0=0;                                  // flag indicated that uart is waiting
    dm(flag_rx_stop_yet)=ax0;               // for the stop bit to arrive
    rti;

invoke_UART_transmit:
    ax0=3;                                  // initialize the timer decimator ctr
    dm(timer_tx_ctr)=ax0;                   // this divide by three ctr is needed
                                            // since timer runs @ 3x baud rate

    ax0=tx_num_of_bits;                     // this constant is defined by the
    dm(bits_left_in_tx)=ax0;                // user and represents total number of
                                            // bits including stop and parity
                                            // ctr is initialized here indicating
                                            // none of the bits have been xmitted
    sr1=0;
    sr0=TX_BIT_ADD;                         // upper bits are hi to end txmit with hi
    ar=dm(user_tx_buffer);                  // transmit register is copied into
    sr=sr or lshift ar by 1 (lo);           // the internal tx reg & left justified
    dm(internal_tx_buffer)=sr0;             // before it gets xmitted

    ax0=0;                                  // indicate that the UART is busy
    dm(flag_tx_ready)=ax0;
    rts;

get_char_ax1:
        ax0=dm(flag_rx_no_word);
        ar=pass ax0;
        if ne jump get_char_ax1;            // if no rx word input, then wait

        ax1=dm(user_rx_buffer);             // get received ascii character
        ax0=1;
        dm(flag_rx_no_word)=ax0;            // word was read
        rts;

out_char_ax1:
        ax0=dm(flag_tx_ready);
        ar=pass ax0;
        if eq jump out_char_ax1;            // if tx word out still pending, then wait
        dm(user_tx_buffer)=ax1;
        call invoke_UART_transmit;          // send it out
        rts;

turn_rx_on:
        ax0=0;
        dm(flag_rx_off)=ax0;
        rts;

turn_rx_off:
        ax0=1;
        dm(flag_rx_off)=ax0;
        rts;
```

Appendix D: Edge Detection Assembly Code

```
/*
       Title: Edge Detection on ADSP-2181
    Compiler: VisualDSP++ 2.0
Date Modified: 4-8-02
     Authors: Paul Grinberg, Mahmoud Yasir Imam
 Affiliation: Case Western Reserve University
 Description: this program is built to work around the specifications of Texas Instruments TC255
              CCD grayscale camera.  The camera outputs its data serially one row at a time.
              The DSP acquires this data and processes it for edges using the Prewitt method
              with a 3 x 3 transformation matrix. The edge detection takes place between the
              acquisitions of two points.  The output is then compressed using the following
              technique:
                     bits 0-6 - count the number of repeating occurrences in the edge data
                     bit 7 - indicates what the count represents (1=edge, 0=blank)
              When the program is complete, it disables all interrupts and sits in an infinite
              loop.
*/

#define ROWSIZE        342                          //each row is 342 wide
#define COLSIZE        243                          //there are 243 columns
#define THRESHOLD      0x1C20                       //minimum P^2 to find edge (8*threshold^2)
#define MAX_COMPRESSION 127                         //maximum compression ratio for edge output

.section/data data1;
.VAR rawdata[ROWSIZE*3];                            //array to store 3 rows of raw incoming data

.section/pm  interrupts;
__reset: JUMP start; NOP; NOP; NOP;                 //reset vector
       RTI; NOP; NOP; NOP;                          //IRQ2
       RTI; NOP; NOP; NOP;                          //IRQL1
       RTI; NOP; NOP; NOP;                          //IRQL2
       RTI; NOP; NOP; NOP;                          //SPORT0 transmit
       RTI; NOP; NOP; NOP;                          //SPORT0 receive
       RTI; NOP; NOP; NOP;                          //IRQE
       RTI; NOP; NOP; NOP;                          //BDMA
       RTI; NOP; NOP; NOP;                          //SPORT1 transmit
       AX0 = RX1;                                   //get value of SPORT1 receive
       DM(I0,M0) = AX0;                             //store data in memory
       RTI; NOP;                                    //end of SPORT1 receive interrupt
       RTI; NOP; NOP; NOP;                          //timer
       RTI; NOP; NOP; NOP;                          //Power down

.section/pm  program;
start:  AX0=0x0000;
        DM(0x3FFE)=AX0;                             //all DM wait states 0
        DM(0x3FFD)=AX0;                             //timer not used
        DM(0x3FFC)=AX0;                             //registers cleared
        DM(0x3FFB)=AX0;
        DM(0x3FFA)=AX0;                             //receive multichannels
        DM(0x3FF9)=AX0;                             //disabled
        DM(0x3FF8)=AX0;                             //transmit multichannels
        DM(0x3FF7)=AX0;                             //disabled
        DM(0x3FF6)=AX0;                             //SPORT0 control not used
        DM(0x3FF5)=AX0;                             //SPORT0 timing not used
        DM(0x3FF4)=AX0;                             //SPORT0 timing not used
        DM(0x3FF3)=AX0;                             //SPORT0 autobuf disabled
        AX0=0x6b07;                                 //internal serial clock
        DM(0x3FF2)=AX0;                             //RFS reqd, normal framing,
                                                    //TFS reqd, normal framing,
                                                    //external RFS, TFS, 0filled
                                                    //MSB, companding, 8-bit words

        AX0=0x0000;                                 //generate 6.144 MHz SCLK1
        DM(0x3FF1)=AX0;
        AX0=128;                                    //generates RFS every 128 SCLK cycles
        DM(0x3FF0)=AX0;                             //sampling rate
```

```
        AX0=0x0000;
        DM(0x3FEF)=AX0;                         //SPORT1 autobuf disabled

        ICNTL=0x07;                             //enable edge sensitive IRQs
        IMASK=0x02;                             //enable SPORT1 receive IRQ

        AX0=0x0C00;                             //SPORT1 enabled, PM wait
        DM(0x3FFF)=AX0;                         //states 0, boot wait
                                                //states 0, boot page 0

/*Setup Pointers and initial values*/
               AX1=MSTAT;                       //get value of multiplier status
               AR=SETBIT 4 of AX1;              //put multiplier in integer mode
               MSTAT=AR;                        //set MSTAT

               I0=rawdata;                      //beginning of buffer
               L0=ROWSIZE*3;                    //buffer is circular with this size
               M0=1;                            //increment buffer by one

/*------Get initial data-----------------------------------*/
               CNTR=ROWSIZE*2+3;                //get 2 rows and 3 points from 3rd row
               DO init_data UNTIL CE;           //repeat loop until counter expired
                      IDLE;                     //wait for data
               init_data: nop;                  //enough to process 1st edge pixel

/*-----Get Rest of data while processing it for edges------*/
               CNTR=COLSIZE-2;
               MX0=1;                           //count = 1
               MY0=0;                           //first point has no edge
               DO other_col UNTIL CE;           //for (COLSIZE-2) rows of data
                      CNTR=ROWSIZE-2;
                      DO other_row UNTIL CE;    //for (ROWSIZE-2) pixels in each row
                             call edge_detection;   //find the edge at this point
                             AY0=MY0;
                             AF=AX1-AY0;        //current - previous
                             IF NE JUMP ser_out;    //if there is a change...
                             AX1=1;             //from edge to no-edge or vice versa
                             AY0=MX0;
                             AR=AX1+AY0;        //then output edge data else increment
                             MX0=AR;            //store new count
                             AY0=MAX_COMPRESSION;   //make sure that the count...
                             AF=AR-AY0;         //does not exceed MAX_COMPRESSION
                             IF NE JUMP data_wait;  //if exceeds, output edge data
                             CALL output_data;      //then output compressed data
                             MX0=0;             //reset counter to 0
                             data_wait: IDLE;       //wait for data to arrive
                      other_row: nop;
                      IDLE;                     //receive 2 pixels from the next row
                      nop;
                      IDLE;                     //so that edge detection can continue
               other_col: nop;
               CALL output_data;
               IMASK=0x00;
               DIS INTS;                        //disable interrupts
inf_loop:
               IDLE;
               JUMP inf_loop;

/*-----Serially Output Edge Data---------------------------*/
ser_out:
               CALL output_data;
               MY0=AX1;                         //set the edge/blank reg appropriately
               MX0=1;                           //reset counter to 1
               JUMP other_row;

output_data:
               AY1=0;
               AX0=MY0;
               AR=MX0;                          //load value of count
               NONE=AX0+AY1;                    //see if counting edges or blanks
               IF NE AR=SETBIT 7 of AR;         //indicate that it's edges
```

```
                TX1=AR;                                 //transmit compressed data
                rts;

/*-----Process data for edges-------------------------------*/
/* Output: AX1 which contains the value of edge (1=edge, 0=blank) */
edge_detection:
                call get_pointer_to_last_pixel;
                AY1=ROWSIZE;
                AR=AR-AY1;                              //a pixel in top row was written to last
                IF LT JUMP x_case_2;                    //this is case two
                AR=AR-AY1;                              //a pixel in middle row was written to last
                IF LT JUMP x_case_3;                    //this is case three
/*Calculating edges in X direction*/
x_case_1:
                AY0=ROWSIZE*2;                          //offset argument for get_value
                call get_value;
                AF=AX1+0;                               //running total
                AY0=ROWSIZE*2+2;                        //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=ROWSIZE;                            //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=ROWSIZE+2;                          //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=0;                                  //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=2;                                  //offset argument for get_value
                call get_value;
                AR=AF-AX1;                              //running total
                MR=AR*AR(SS);                           //X^2
                JUMP y_case_1;
x_case_2:
                AY0=-ROWSIZE;                           //offset argument for get_value
                call get_value;
                AF=AX1+0;                               //running total
                AY0=-ROWSIZE+2;                         //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=-ROWSIZE*2;                         //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=-ROWSIZE*2+2;                       //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=0;                                  //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=2;                                  //offset argument for get_value
                call get_value;
                AR=AF-AX1;                              //running total
                MR=AR*AR(SS);                           //X^2
                JUMP y_case_2;
x_case_3:
                AY0=-ROWSIZE;                           //offset argument for get_value
                call get_value;
                AF=AX1+0;                               //running total
                AY0=-ROWSIZE+2;                         //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=ROWSIZE;                            //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=ROWSIZE+2;                          //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=0;                                  //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
```

```
                AY0=2;                                  //offset argument for get_value
                call get_value;
                AR=AF-AX1;                              //running total
                MR=AR*AR(SS);                           //X^2
                JUMP y_case_3;
/*Calculating edges in Y direction*/
y_case_1:
                AY0=ROWSIZE*2+2;                        //offset argument for get_value
                call get_value;
                AF=AX1+0;                               //running total
                AY0=ROWSIZE*2+1;                        //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=ROWSIZE*2;                          //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=2;                                  //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=1;                                  //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=0;                                  //offset argument for get_value
                call get_value;
                AR=AF-AX1;                              //running total
                JUMP threshold_comparison;
y_case_2:
                AY0=-ROWSIZE+2;                         //offset argument for get_value
                call get_value;
                AF=AX1+0;                               //running total
                AY0=-ROWSIZE+1;                         //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=-ROWSIZE;                           //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=2;                                  //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=1;                                  //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=0;                                  //offset argument for get_value
                call get_value;
                AR=AF-AX1;                              //running total
                JUMP threshold_comparison;
y_case_3:
                AY0=-ROWSIZE+2;                         //offset argument for get_value
                call get_value;
                AF=AX1+0;                               //running total
                AY0=-ROWSIZE+1;                         //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=-ROWSIZE;                           //offset argument for get_value
                call get_value;
                AF=AX1+AF;                              //running total
                AY0=2;                                  //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=1;                                  //offset argument for get_value
                call get_value;
                AF=AF-AX1;                              //running total
                AY0=0;                                  //offset argument for get_value
                call get_value;
                AR=AF-AX1;                              //running total

threshold_comparison:
                MR=MR+AR*AR(SS);                        //P^2=Y^2 + X^2
                AY0=THRESHOLD;                          //load the threshold value
                AR=MR0-AY0;                             //subtract from P^2
                IF LT JUMP no_edge;                     //if negative, goto no_edge
```

```
                AX1=1;                                  //edge detected
                rts;                                    //return
no_edge:
                AX1=0;                                  //no edge detected
                rts;                                    //return

/* Input: AY0 contains offset value */
/* Output: AX1 contains valye of desired pixel */
get_value:
                call get_pointer_to_last_pixel;         //see name of function
                AR=AR-AY0;                              //eval pointer to desired pixel
                I1=AR;                                  //set the DAG
                AX1=DM(I1,M0);                          //get the value of the pixels
                rts;

get_pointer_to_last_pixel:
                AX1=I0;                                 //value of pointer to last rawdata+1
                AY1=0;                                  //zero
                AR=AX1+AY1;                             //needed to set the ALU Status
                AY1=ROWSIZE*3;                          //offset for rawdata pointer
                IF EQ AR=AR+AY1;                        //if pointer has wrapped around, use offset
                AY1=-1;
                AR=AR+AY1;                              //eval pointer to last pixel
                rts;
```

Appendix E: TC255 Emulation Code

```
#        Title: emulation.pl
#       Author: Paul Grinberg
#Date Modified: 4-9-02
#     Compiler: Perl 5.6.1 with Tk 8.0.21
#  Description: This program emulates the Texas Instruments TC255 and the 3 x 3 Prewitt Edge
#Detection Algorithm as it would be running on the ADSP-2181.  The program has the following
#functionalities:
#       New - creates a new circular image of random radius and grayscale color
#       Open - opens a file in the TC255 raw data format or in a gif image format
#       Save - saves the currently displayed image into a raw data format
#       Clear - clears the screen
#       Find Edges - find the edges emulating the steps performed by the DSP and displays edges
#       Add Edges - superimposes the edge data over the currently displayed image
#       Exit - quit program

use strict;
use Tk;
use Cwd;

my (
    $mw,
    $img,
    $progress,
    $color,
    $radius,
    $colorLabel,
    $radiusLabel,
    $progressLabel,
    @data,
    $pid,
);

my $TCWidth = 324;            #width of the TC255
my $TCHeight = 243;           #height of the TC255
my $size=2;                   #zoom factor
my $BGColor = 255;            #background color
my $EdgeColor = "#FF0000";    #edge color
my $EdgeWidth = 1;            #edge thickness
my $threshold=8*30**2;        #edge detection threshold value

#usage: if no parameters are passed, then the emulation is simply that
#if a filename is passed as a parameter, then the program works as the DSP

if (defined($ARGV[0])){
  detect();
  exit;
}

#intensity is an 8 bit value corresponding to grayscale colors
# 0 = black
# 255 = white

$mw = MainWindow->new(-title=>"TC255 Emulator",
                      -width=>$TCWidth*$size+130,
                      -height=>$TCHeight*$size+27);
$img = $mw->Photo('img',
                  -width=>$TCWidth*$size,
                  -height=>$TCHeight*$size);
$mw->Label(-image=>'img',
           -relief=>"sunken")->place(-x=>0, -y=>0);
$mw->Label(-borderwidth=>'1',
           -text=>"Radius:",
           -width=>'7',
           -font=>"Arial 8 bold")->place(-x=>$TCWidth*$size+10,
                                         -y=>'17');
$radiusLabel = $mw->Label(-borderwidth=>'1',
```

```
                         -text=>$radius,
                         -width=>'4',
                        -anchor=>"w",
                        -font=>"Arial 8")->place(-x=>$TCWidth*$size+55,
                                                 -y=>'17');
$mw->Label(-borderwidth=>'1',
           -text=>"Color:",
           -width=>'7',
           -anchor=>"e",
           -font=>"Arial 8 bold")->place(-x=>$TCWidth*$size+10,
                                         -y=>'37');
$colorLabel = $mw->Label(-borderwidth=>'1',
                        -text=>$color,
                        -anchor=>"w",
                        -width=>'4',
                        -font=>"Arial 8")->place(-x=>$TCWidth*$size+55,
                                                 -y=>'37');
$mw->Button(-text=>"New",
            -width=>5,
            -font=>"Times 8 bold",
            -command=>sub{eraseData();
                          @data = getData();
                          drawData(\@data);
                          $radiusLabel->configure(-text=>$radius);
                          $colorLabel->configure(-text=>$color);
                         })->place(-x=>$TCWidth*$size+15,-y=>67);
$mw->Button(-text=>"Clear",
             -width=>6,
             -font=>"Times 8 bold",
             -command=>\&eraseData)->place(-x=>$TCWidth*$size+65,-y=>67);
$mw->Button(-text=>"Save",
            -width=>6,
            -font=>"Times 8 bold",
            -command=>sub{saveData(\@data);})->place(-x=>$TCWidth*$size+10,-y=>100);
$mw->Button(-text=>"Open",
            -width=>6,
            -font=>"Times 8 bold",
            -command=>sub{eraseData();
                          @data = openData();
                          drawData(\@data);
                         })->place(-x=>$TCWidth*$size+65,-y=>100);
$mw->Button(-text=>"Find Edges",
            -width=>10,
            -font=>"Times 8 bold",
            -command=>sub{writeData(cwd."/default.dat",@data);
                          $ARGV[0]="default.dat";
                          detect();
                          my @d = decompressEdges(cwd."/OUT.default.dat");
                          drawEdges(\@d);
                         })->place(-x=>$TCWidth*$size+25,-y=>133);
$mw->Button(-text=>"Add Edges",
            -width=>9,
            -font=>"Times 8 bold",
            -command=>sub{ my @d = decompressEdgesFromFile();
                          drawEdges(\@d);
                         })->place(-x=>$TCWidth*$size+28,-y=>166);
$mw->Button(-text=>"Exit",
            -width=>6,
            -font=>"Times 8 bold",
            -command=>sub{exit;})->place(-x=>$TCWidth*$size+35,-y=>$TCHeight*$size-10);
$mw->Label(-borderwidth=>'1',
           -text=>"Progress:",
           -width=>'9',
           -anchor=>"e",
           -font=>"Arial 8")->place(-x=>0,
                                    -y=>$TCHeight*$size+6);
$progress = $mw->Photo('prog',
                       -width=>$TCHeight*$size,
                       -height=>10);
$progressLabel = $mw->Label(-relief=>"sunken",
                            -image=>'prog',
```

```
                            -borderwidth=>2)->place(-x=>57,-y=>$TCHeight*$size+9);

MainLoop;

sub decompressEdgesFromFile{
  my $file = $mw->getOpenFile(-defaultextension => ".dat",
                              -filetypes =>[['VisualDSP++','.dat'],
                                            ['All Files','*',],],
                              -initialdir => ".",
                              -title => "Open Edge File");
  return unless defined($file);
  return decompressEdges($file);
}

sub decompressEdges{
  my $file=shift;
  my @d;   #it's easier if this array is initially stored as a 1D array
  my @dd;  #this is a 2D array that is going to be drawn next;
  my $i=0; #easier into this 1D array (uncompressed edge data)
  my $q;   #some counting variable
  open(DAT,"<$file") || die "Can't open file $file: $!";
  while ($_ = <DAT>){
        if (!hex){next;} #this may only happen for the first point that may need to be dropped
        if (hex() < 128){  #this is a series of blanks
          for ($q=$i;$q<(hex()+$i);$q++){
                $d[$q]=0;
          }
          $i = $q;
        } else { #this is a series of edges
          $_ = hex() - 128;
          for ($q=$i;$q<($_+$i);$q++){
                $d[$q]=1;
          }
          $i = $q;
        }
  }
  close(DAT);
  for ($q=0;$q<scalar(@d);$q++){
        $dd[int $q/($TCWidth-2)][$q%($TCWidth-2)] = $d[$q];
  }
  return @dd;
}

sub saveData{
  my @d = @{$_[0]};
  my $file = $mw->getSaveFile(-defaultextension => ".dat",
                              -filetypes =>[['VisualDSP++','.dat'],
                                            ['All Files','*',],],
                              -initialdir => ".",
                              -initialfile => "default.dat",
                              -title => "Save Image");
  return unless defined($file);
  writeData($file,@d);
}

sub writeData{
  my $file = shift;
  my @d=@_;
  open(DAT,">$file") || die "Can't open file $file: $!";
  for (my $y=0; $y<$TCHeight; $y++){
    for (my $x=0; $x<$TCWidth; $x++){
      print DAT sprintf("%02x\n", $d[$y][$x]);
    }
  }
  close(DAT);
}

sub openData{
  my @d;
  my $file = $mw->getOpenFile(-defaultextension => ".dat",
                              -filetypes =>[['VisualDSP++','.dat'],
```

```
                                        ['GIF','.gif'],
                                        ['All Files','*',],],
                            -initialdir => ".",
                            -initialfile => "default.dat",
                            -title => "Open Image");
  return unless defined($file);
  if ($file =~ /gif$/){
    if (!-f $file){return;}
    my $tmp = $mw->Photo('tmp',
                         -file => $file);
    for (my $y=0; $y<$TCHeight; $y++){
      for (my $x=0; $x<$TCWidth; $x++){
        $d[$y][$x] = ($tmp->get($x,$y))[0];
      }
    }
    return @d;
  }
  open(DAT,"<$file") || die "Can't open file: $!\n";
  my $y = 0;
  while ($_ = <DAT>){
    chomp;
    $d[int $y/$TCWidth][$y%$TCWidth] = hex $_;
    $y++;
  }
  close(DAT);
  return @d;
}

sub eraseData{
  $img->put(sprintf("#%02x%02x%02x",$BGColor,$BGColor,$BGColor),
            -to=>0,0,$TCWidth*$size,$TCHeight*$size);
}

sub drawData{
  my @d = @{$_[0]};
  for (my $y=0; $y<$TCHeight; $y++){
    for (my $x=0; $x<$TCWidth; $x++){
      if ($d[$y][$x] != $BGColor){
               $img->put(sprintf("#%02x%02x%02x",$d[$y][$x],$d[$y][$x],$d[$y][$x]),
                         -to=>$x*$size,$y*$size,($x+1)*$size,($y+1)*$size);
      }
    }
    $progress->put("#0000FF",-to=>$y*$size,0,($y+1)*$size,20);
    $progress->update();
  }
  $progress->blank();
}

sub drawEdges{
  my @d = @{$_[0]};
  for (my $y=0; $y<$TCHeight-2; $y++){
    for (my $x=0; $x<$TCWidth-2; $x++){
          if ($d[$y][$x]){
               $img->put($EdgeColor, -to=>($x)*$size,($y)*$size,
                         ($x+$EdgeWidth)*$size,($y+$EdgeWidth)*$size);
          }
        }
    $progress->put("#0000FF",-to=>$y*$size,0,($y+1)*$size,20);
    $progress->update();
  }
  $progress->blank();
}

sub getData{
  my @d;

  $color = int rand(255);
  $radius = int rand($TCHeight/2-5);

  for (my $y=0; $y<$TCHeight; $y++){
    for (my $x=0; $x<$TCWidth; $x++){
```

```
      if(((($x-$TCWidth/2)**2 + ($y-$TCHeight/2)**2) < ($radius**2)){ #a circle
        $d[$y][$x] = $color;
      } else {
        $d[$y][$x] = $BGColor;
      }
    }
  }
  return @d;
}


sub detect{
  my @d;

  open(DAT,$ARGV[0]) || die "Can't open file $ARGV[0]: $!\n";
  for (my $y=0; $y<$TCHeight; $y++) {
       for (my $x=0; $x<$TCWidth; $x++) {
         $_ = <DAT>;
         chomp;
         $d[$x][$y] = hex $_;
       }
  }
  close(DAT);

  open(DAT,">OUT.$ARGV[0]") || die "Can't open file OUT.$ARGV[0] for write: $!\n";
  my $prev=0;                                #we were counting blanks
  my $count=1;                               #we had 1 of them
  for (my $y=0; $y<$TCHeight-2; $y++) {
       for (my $x=0; $x<$TCWidth-2; $x++) {
         my $X=$d[$x+2][$y]-$d[$x][$y]+$d[$x+2][$y+1]-$d[$x][$y+1]+$d[$x+2][$y+2]-$d[$x][$y+2];
         my $Y=$d[$x][$y]+$d[$x+1][$y]+$d[$x+2][$y]-$d[$x][$y+2]-$d[$x+1][$y+2]-$d[$x+2][$y+2];
         $X*=$X;
         $Y*=$Y;
         if (($X+$Y) >= $threshold) { #if P^2 exceeds threshold
              if (!$prev) {                 #if there is a change from blank to edge
                print DAT sprintf("%02x\n",$count); #output the compressed data
                $prev=1;                    #we are now counting edges
                $count=1;                   #we had one already
              } else {                      #else no change
                $count++;                   #so just keep on counting
              }
         } else {                           #P^2 is not exceeded
              if ($prev) {                  #if there is a change from edge to blank
                print DAT sprintf("%02x\n",$count+128); #output the compressed data
                $prev=0;                    #we are now counting blanks
                $count=1;                   #we had one already
              } else {                      #else no change
                $count++;                   #so jsut keep on counting
              }
         }
         if ($count == 127) {        #if we reached the maximum count
              if ($prev) {                  #the write out the data
                print DAT sprintf("%02x\n",$count+128);
              } else {
                print DAT sprintf("%02x\n",$count);
              }
              $count=0;                     #reset the counter
         }
       }
  }
  if ($prev) {                              #output the last data
       print DAT sprintf("%02x\n",$count+128);
  } else {
       print DAT sprintf("%02x\n",$count);
  }
  close(DAT);
}
```

Advisor Meeting Log

References

[i] Kevin Burns, Masters Thesis, April 200. Advisor: Prof. Merat.

[ii] http://www.edmundoptics.com/techsupport/DisplayArticle.cfm?articleid=290

[iii] http://www-s.ti.com/sc/ds/tc255.pdf

[iv] http://focus.ti.com/docs/prod/productfolder.jhtml?genericPartNumber=TMS320C31

[v] http://products.analog.com/products/info.asp?product=ADSP-2181

[vi] http://www.microchip.com/1000/pline/picmicro/category/digictrl/14kbytes/devices/16c67/index.htm

[vii] http://www.analog.com/library/dspManuals/ADSP-2100_fum_books.html

[viii] ftp://ftp.analog.com/pub/dsp/tools/Wm2181EZ_readme.txt

[ix] http://www.pcigeomatics.com/cgi-bin/pcihlp/FPRE

[x] http://www.css.tayloru.edu/~jokane/351cos/sobel/

[xi] http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/laplacian.html

[xii] http://www.analog.com/library/applicationNotes/dsp/16_devTools/ee_60.pdf

[xiii] http://www.analog.com/technology/dsp/training/tutorials/v_dsp_tutorial.html

[xiv] http://www.cpan.org/modules/by-module/Tk/Tk-800.024.readme

[xv] Danya, Paul Grinberg's 17lbs cat