AC 2011-2160: A HANDS-ON APPROACH TO DEMONSTRATING HARDWARE/SOFTWARE TRADEOFFS IN AN EMBEDDED SYSTEM DESIGN

Jeanne Christman, Rochester Institute of Technology (COE)

Jeanne Christman is an Assistant Professor in the Computer Engineering Technology Department at the Rochester Institute of Technology. Her expertise is in the area of Embedded Systems Desgin and System on a Chip. She is also actively involved in recruitment and retention of females in engineering technology.

Eric J Alley, Rochester Institute of Technology

Eric Alley is a 2011 graduate of the Rochester Institute of Technology with a degree in Computer Engineering Technology. His RIT career includes working as a teaching assistant for many core curriculum engineering technology classes, president of a major student organization, and taking part in the Imagine RIT Innovation Festival with a peer developed project.

A Hands-on Approach to Demonstrating Hardware/Software Tradeoffs in an Embedded System Design

Abstract

This paper describes a Computer Engineering Technology lab activity in an Embedded Systems Design course used to provide students with an opportunity to substantiate the theory being presented in the classroom. The objective of the lab is to quantitatively evaluate both the hardware and software implementation of an image rotation algorithm on a System on a Programmable Chip (SOPC). In this exercise, students gain experience in embedded C programming, HDL design and custom instruction generation, along with the use of a VGA interface. In addition to the practical design experiences, students are able to measure the execution time, as well as compare system resource usage, of both implementations thus illustrating the hardware/software tradeoffs discussed in the lecture portion of the class.

As in most engineering technology classes, the emphasis is on hands-on learning. Theoretical concepts are presented in lecture and put to test with weekly lab experiments. In this particular course, students analyze real-world embedded system design issues. They are well familiar with modern consumer products, which continue to get smaller, faster and consume less power. This experiment aims to illustrate that gains in speed, size or power often come at a cost to the other two factors. The quantitative investigation emphasizes, through a cost benefit analysis of a real world hands-on experience, the use of hardware versus software in an embedded system.

This lab is an effective teaching tool which is engaging and interesting to the students. They are enthusiastic about analyzing and solving a real-world design issue. Displaying and manipulating images on the VGA further peaks their interest as they are able to visibly see the difference between the two implementations.

Introduction

The Electrical, Computer and Telecommunications Engineering Technology (ECTET) Department at the Rochester Institute of Technology (RIT) offers a unique Computer Engineering Technology degree that bridges the gap between Computer Science and Electrical Engineering Technology (EET). Students in this program start with a solid foundation in EET, taking all of the circuits, electronics and hardware design courses with the EET students. They also take high-level and assembly language programming and electives from the Computer Science department. Their education culminates in a three course Embedded Systems Design Sequence. Graduates from this program have an in-depth knowledge not just of hardware and software design, but also of hardware/software co-design.

Similar to students in most university Computer Engineering Technology departments, RIT's CpET students take required courses in embedded microcontroller programming and also in hardware development using a Hardware Description Language (HDL) targeted for a Field Programmable Gate Array (FPGA) during their first three years. With the knowledge gained from those courses as a basis, the Embedded Systems Design classes provide an intensive laboratory and project based learning experience where students are able to approach system

design from both the hardware and software perspective. While traditional lectures are still employed to introduce concepts and theories, the emphasis is placed on hands-on learning. Typically, every third lecture is delivered in studio format with each student being able to participate on their own computer.

In Embedded Systems Design I, the emphasis is placed on processor architecture, embedded programming, interfacing to off-the-shelf IP's and creating hardware modules to communicate on a common bus structure. The focus of ESD II is the evaluation of design options and their effect on system performance along with the development of custom IP to interface with a variety of peripherals. The curriculum's capstone course, ESD III, involves creating and implementing a complete system; starting with design specification and ending with finished product. The students learn about the design process and manage their projects as if they were an assignment in industry. This unique sequence of courses provides an integrated experience where the concepts learned in previous programming and digital design classes are combined for a system level approach.

Tools

The Altera DE2 Development and Education board is the platform used for the Embedded Systems Design sequence. The department's Industrial Advisory Board encouraged the department to consider an Altera development platform to best prepare students for the tools they would be using in their companies. Beginning in the freshman Digital Fundamentals class, the DE2 board is used in six different CpET classes, including ESD I, II & III. By using the same platform in all of the digital design courses, time is not lost each quarter to learning new tools. The DE2 board contains a Cyclone II FPGA along with support for the NIOS II soft core processor. The availability of the soft core processor along with the additional FPGA fabric is the ideal development environment for an integrated hardware and software approach to teaching embedded systems concepts. The configurable processor is especially effective when illustrating design tradeoffs. For example, when teaching about cache memory, students are able to modify various cache configurations, such as block size, run program loops, and compare the performance all within a one hour studio session. The feature of the NIOS II soft core processor that is central to this paper is the ability to create custom instructions for hardware acceleration. A custom instruction is a hardware module designed to replace complex and/or time-critical sections of code. Since the NIOS II is a soft core IP and placed in the FPGA fabric upon each compilation, the custom instruction logic becomes part of the processor's ALU. From a pedagogical viewpoint, custom instructions provide a means to experiment with hardware and software tradeoffs at any point in the design process.

Another feature of Altera's DE2 board that makes it a good choice for the Embedded Systems Design courses is its variety of peripheral devices. Today's students, dubbed the millennial generation, live in a world of multi-media and entertainment. In order to engage many of today's students, projects must have an end product that they find entertaining. The DE2 board provides an audio CODEC for voice and music applications along with a video in and video out (VGA) port. Although the board does provide other peripherals such as a LCD display, Ethernet and infra-red, it has been found that assignments that involve audio and visual interfaces are the ones that are more likely to hold the student's interest.

Lab Development

The laboratory described in this paper was developed for the Embedded Systems Design II course to meet the objectives of quantitatively comparing the performance of a hardware and a software solution to the same algorithm. An image rotation algorithm was chosen for two reasons; the first being that the results can initially be measured qualitatively by observation. A bigger reason for choosing image rotation is that it is an application that students are familiar with. They are exposed to image rotation daily on their phones and mp3 players, but have never considered the processing behind it prior to starting this lab.

Like all lab projects, this lab starts with an introduction to the theory behind the exercise. The equation for rotating an image about its center is given in Figure 1, where X_{source} and Y_{source} is the pixel location in the original image, X_{dest} and Y_{dest} is the corresponding pixel locations in the rotated image and theta is the angle of rotation as seen in Figure 2 below.

$$\begin{bmatrix} X_{dest} \\ Y_{dest} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \left(\begin{bmatrix} X_{source} \\ Y_{source} \end{bmatrix} - \begin{bmatrix} X_{center} \\ Y_{center} \end{bmatrix} \right) + \begin{bmatrix} X_{center} \\ Y_{center} \end{bmatrix}$$

Figure 1: Forward Mapping Equation for Image Rotation About Its Center

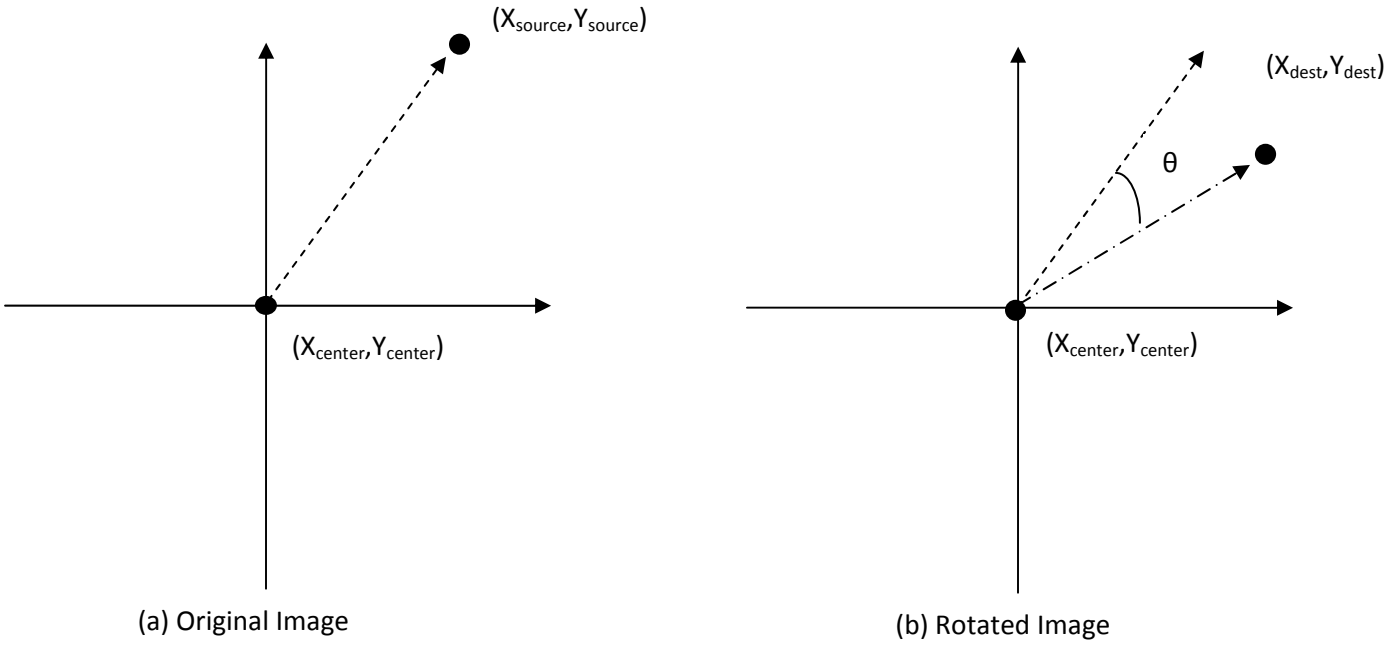

Figure 2: Image Rotation by Theta

While this equation is straightforward and easily understood by the students, a problem arises when it is used. Due to rounding of floating point values to integers for pixel locations, some destination pixels will be mapped to more than one source pixel. With two or more source pixels

mapping to the same destination pixel, there will also be destination pixels that are never written to, thus leaving holes in the rotated image. To solve the problems of holes and multiple mappings, the reverse mapping equation is employed. In the reverse mapping equation, each destination pixel is evaluated to determine from which source pixel it came. The reverse mapping equation uses the inverse of the transformation matrix as seen in Figure 3.

$$\begin{bmatrix} X_{source} \\ Y_{source} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}^{-1} \left(\begin{bmatrix} X_{dest} \\ Y_{dest} \end{bmatrix} - \begin{bmatrix} X_{center} \\ Y_{center} \end{bmatrix} \right) + \begin{bmatrix} X_{center} \\ Y_{center} \end{bmatrix}$$

Figure 3: Reverse Mapping Equation for Rotation of an Image About Its Center

From Figure 3, the equations for X_{source} and Y_{source} are derived as seen in Figure 4.

$$Xsource = ((Xdest - Xcenter) * \cos\theta) - ((Ydest - Ycenter) * \sin\theta) + Xcenter$$
$$Ysource = ((Ydest - Ycenter) * \cos\theta) + ((Xdest - Xcenter) * \sin\theta) + Ycenter$$

Figure 4: Equations for X_{source} and Y_{source}

A fairly simple C program can be written in which a nested loop is used to calculate the (X,Y) coordinate in the source image for every possible (X,Y) coordinate in the destination image. Each (X_{source},Y_{source}) pair is used as an index into the stored image to retrieve the pixel value. The pixel value is then written to the VGA at the proper coordinate. If the destination pixel does not have a corresponding pixel in the source image, the destination pixel is colored black. The sine and cosine of theta must also be calculated. However that calculation only needs to be done one time per rotation. The C code for the rotation algorithm is provided in Figure 5.

```
//for every (x,y) coordinate in the destination image, calculate the corresponding pixel in the source image.
//From the flash memory, retrieve the color of that pixel in the source image.  Then draw that color in the
//destination image.  If there was no corresponding pixel in the source image draw a black pixel.

  INVERSE_MATRIX[0] = cos(theta);
  INVERSE_MATRIX[1] = sin(theta);

   for (y=0;y<PB->y_resolution;y++)
   {
        // find the x offset from center
        y_off = y-y_cent;
        for (x=0;x<PB->x_resolution;x++)
        {
                // find the x offset from center
                x_off = x-x_cent;

                //these are the equations to find the source pixels
                x_source = (int)round( (x_off*INVERSE_MATRIX[0]) - (y_off*INVERSE_MATRIX[1])+ x_cent);
                y_source = (int)round( (x_off*INVERSE_MATRIX[1]) + (y_off*INVERSE_MATRIX[0])+ y_cent);

                 //if there was not corresponding pixel in the source image, the result will be < 0 or > resolution
                if ((x_source < 0) || (x_source >= PB->x_resolution) || (y_source < 0) || (y_source >= PB->y_resolution))
                {
                        //draw black
                        alt_up_pixel_buffer_draw_front(PB,0,x,y);
                }
                else
                {
                         //get the pixel from flash memory and draw it to the VGA
```

```
                    dest=IORD_8DIRECT(FLASH_MEMORY_BASE,(x_source*PB->x_resolution)+y_source);
                    alt_up_pixel_buffer_draw_front(PB,dest,x,y);
                }

        }
    }
```

Figure 5: C Code for the Image Rotation Algorithm

In the code of Figure 5, PB refers to the pixel buffer used by the VGA. For this laboratory setup, the 512K SRAM provided on the DE2 board is used as the pixel buffer. The 4M flash memory on the DE2 board is used to store the original image. For development purposes, it is best to put the image in the non-volatile memory, as the process of loading an image from the host PC to the board is extremely slow via the JTAG_UART. By using the flash memory, the image only has to be loaded one time during the development of the lab. Students prefer this since their time does not have to be spent watching the image load.

Besides the SRAM pixel buffer and the flash memory image buffer, the system developed for this experiment includes a NIOS II processor, a JTAG_UART for communication with the development computer, onboard 8M SDRAM to store the program code, an interval timer used to measured execution time, and 8 switches for the user to input the angle of rotation. The VGA Controller core provided by Altera is also included in the system to generate the timing signals required by the on-board VGA DAC. This core reads directly from the pixel buffer in the SRAM.

The next step is to accelerate the computationally intense portions of the program using hardware developed with HDL. Floating point arithmetic calculations are notoriously slow and are typically avoided in embedded programs if at all possible, thus making sine and cosine calculations the first candidates for acceleration. Two different methods for calculating sine and cosine in hardware using an HDL were considered. The first is a look-up table where the values are pre-calculated and then stored in a ROM. The advantage of this method is that data retrieval is fast, however there are a limited number of angles that can be stored, and increasing the number of angles increases the memory resources needed to store the data. The second method for calculating sine and cosine in hardware is the use of the Coordinate Rotations Digital Computer (CORDIC) algorithm. The CORDIC algorithm provides an iterative method for performing vector rotations by arbitrary angles using only shifts and adds[i]. Within this algorithm the vector rotations form the basis for the trigonometric functions. The students are provided with the paper previously referenced, which can also be found at www.andraka.com/files/crdcsrvy.pdf, to assist in their understanding of the derivations leading to the final equations. The final equations for calculation of sine and cosine with the CORDIC algorithm are given in Figure 6.

$$X_{i+1} = X_i - d_i \cdot Y_i \cdot 2^{-i}$$
$$Y_{i+1} = Y_i + d_i \cdot X_i \cdot 2^{-i}$$
$$Z_{i+1} = Z_i - d_i \cdot \tan^{-1} 2^{-i}$$

Figure 6: CORDIC equations for X, Y and Z

The equations above iterate from i = 0 to k where k represents the number of bits of precision in the results. In the case of this exercise, k = 8. After 8 iterations of the algorithm, X_8 is the cosine

of the original angle while Y_8 is its sine. In each iteration d_i is chosen to converge Z towards 0. The eight angles chosen for rotation, which are predetermined and stored in a ROM, all have tangent() values that are a power of 2. This allows for the hardware to perform simple shifts instead of more time-consuming multiplications.

The NIOS II soft core processor provides for custom instructions to be added to the processor core. Once the students have written and verified their CORDIC module in HDL, they can add it to the NIOS II core through the custom instruction generator provided in Altera's SOPC builder. The processor provides the custom instruction with the input angle, a clock and a start signal. When the module is done with its calculations it raises a done signal and provides the results to the processor in the form of a returned variable. When the system is built in the NIOS II Integrated Design Environment IDE, a macro (instruction) is generated that allows access from the application code to the custom hardware. This macro replaces the sine and cosine calculations in the C code given in Figure 5. As was previously noted, the calculation of sine and cosine only occurs one time for each rotation angle. Based on timing results and hardware resource usage, it is left to the student to evaluate and discuss whether the effort of designing the CORDIC module provides an advantage to the overall system design. Comparative result are provided in the 'Results' section below.

As seen in the C code of Figure 5, the calculation of each source pixel is computationally intensive with 4 multiplications, 5 subtractions and 3 additions. The source pixel calculation becomes another candidate for acceleration. The HDL module to perform these arithmetic operations takes advantages of the parallelism achievable in hardware. By exploiting parallelism, the results can be obtained in a single clock cycle. As with the CORDIC accelerator, when the system is built, the IDE generates a macro to be placed in the C code. The operands for this custom instruction are sine, cosine, X_{dest} and Y_{dest}. The instruction returns X_{source} and Y_{source}. Due to limitations with the custom instruction generator regarding the number of input signals allowed, X_{center} and Y_{center} cannot be input to the hardware via the instruction. Those values are left as constants in the hardware and must be changed if a different image size is used. The custom instruction macro for finding the source pixel location replaces the two lines of code in the original C program. In the results analysis below, this custom instruction that calculates X_{source} and Y_{source} is referred to as FindSource.

Student Analysis of the Results

During the analysis portion of the laboratory, students are provided with C Code functions that utilize the Altera timer core to calculate the number of clock cycles, and moreover the amount of time it takes, to complete the image rotation. Using these timing functions, students collect a range of data points which allows them to assign a real world quantitative comparison of performance. Table 1 shows the performance data collected for various image resolutions. Total Time includes the calculation of sine and cosine along with the calculation of the source pixel. FindSource represents only the time required to calculate the source pixel. The data from Table 1 is displayed graphically in Figure 7.

Horizontal Pixels	Vertical Pixels	Total Pixels	Total Time HW (ms)	Total Time SW (ms)	FindSource HW (ms)	FindSource SW (ms)
80	60	4800	140.389	1498.702	28.848	1281.248
80	120	9600	280.719	2973.536	57.670	2498.373
160	60	9600	280.708	2968.407	57.447	2496.982
160	120	19200	560.848	6167.570	114.868	4988.880
320	240	76800	2363.806	24144.335	458.501	19910.201
320	480	153600	4678.262	47381.927	916.977	39809.474
640	240	153600	4868.179	45172.460	916.083	40979.439
640	480	307200	9797.243	95739.913	1832.140	81565.573

Table 1: Resolution and Execution Time Data for Hardware Accelerated System and Standard Image Rotation System

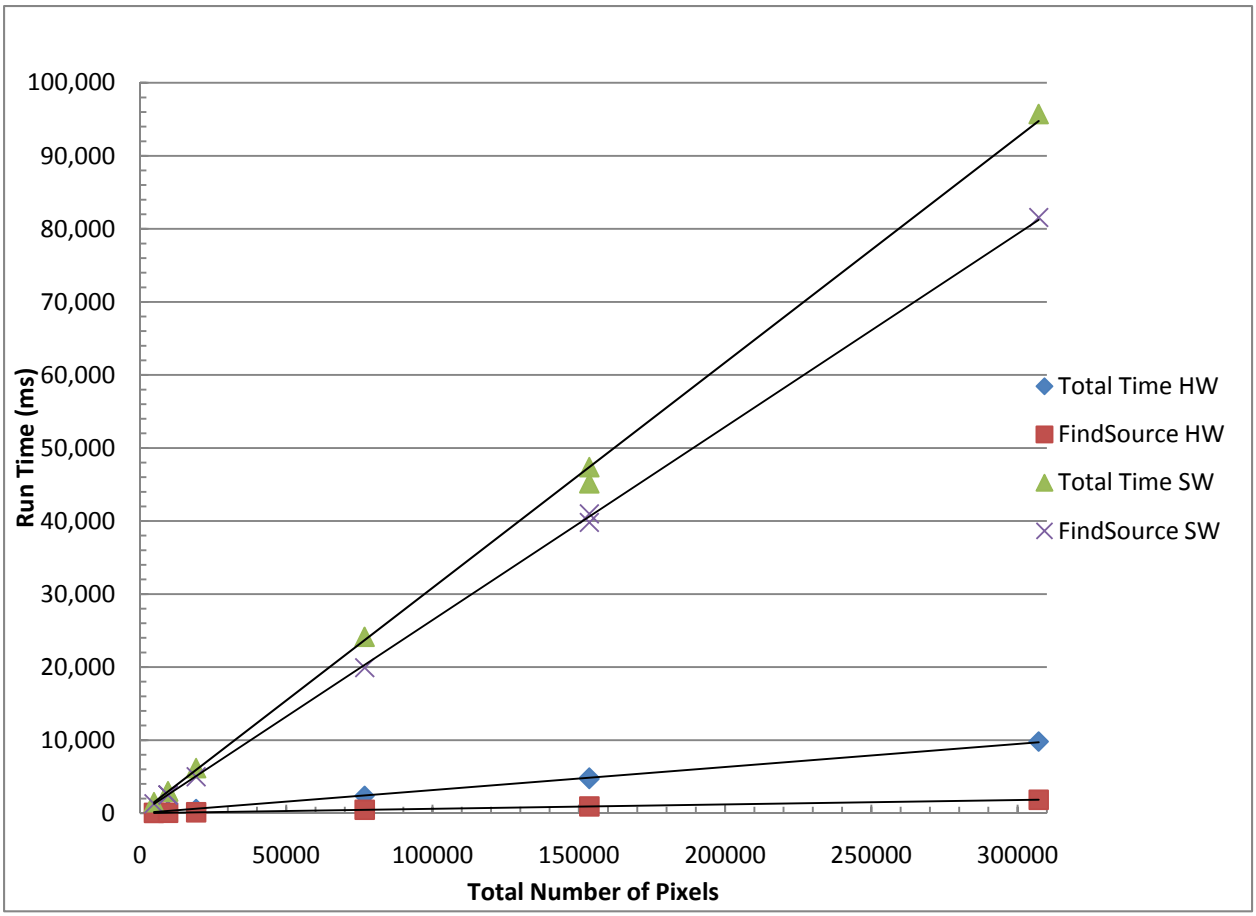

Figure 7: Runtime vs Resolution for Image Rotation Using Hardware Acceleration and No Hardware Acceleration

For the laboratory setup, an 8-bit grayscale image is used to reduce the size of the data loaded to memory. The CORDIC and FindSource algorithms are independent of color depth so using an 8-bit grayscale image reduces the size of the picture in memory but does not affect the run times of the image rotation in hardware or software.

The CORDIC reduces the calculation of the sine and cosine from 3,235 microseconds down to 22 microseconds. This is a significant improvement reducing the run-time for the single instruction to less than 1% of its original value. The CORDIC is only executed one time per rotation making its impact to the overall speed of the image rotation inversely proportional to resolution; meaning, for an image with very few pixels, the improvement is more significant than it is for a larger image. The FindSource algorithm, however, is executed for every pixel in the image. The run-time for a single iteration of the FindSource algorithm in software is 314 microseconds whereas utilizing the custom instruction is 26 microseconds. Trend lines were added to Figure 7 to show the linearity of the data. It is interesting to note that the slopes of the trend lines (in milliseconds) are 0.309 and 0.032. For each pixel added to the resolution of the image it takes approximately 309 microseconds longer to rotate the image in software and 32 microseconds longer to rotate the image in hardware. These two values are very close to the execution time for each of the algorithms, indicating that the FindSource algorithm heavily dictates the total execution time of the image rotation.

Since partitioning does not rely solely on performance metrics, a comparison of hardware requirements is also needed. Table 2 shows both the memory usage for program storage and the FPGA logic elements (LEs) necessary to hold the logic.

	No Custom Instructions	CORDIC & FIND_SOURCE Custom Instructions	CORDIC Custom Instruction Only	FIND_SOURCE Custom Instruction Only
Software	91KB	81KB	81KB	91KB
Hardware (Total LEs)	3843	4128	4059	3928
-Registers	2249	2360	2327	2301
-Multipliers (9-bit)	4	12	4	12

Table 2: Hardware and Software Resource Utilization for Hardware Accelerated and Standard Image Rotation System

The inclusion of the CORDIC custom instruction allows for the size of the C code to be reduced since the math library is no longer needed for the sine and cosine functions. Removing the math library allows the size of the code to be reduced by 10KB, which accounts for 10% of the total program size in this exercise. The FPGA fabric used to implement the CORDIC in the system increases the total LE usage by 216 elements. This represents a 5.6% increase from the original system.

The addition of the FindSource Custom Instruction does not provide a reduction in code size but does have a larger impact on resource usage. The total logic element usage increases by 85 elements, a 2.2% increase over the original system. Although this increase seems insignificant, it is important to note that the FindSource custom instruction utilizes twelve of the 9-bit hardware multipliers while only four were used in the original system. These hardware multipliers are hard macros on the Cyclone II FPGA so their use does not affect the number of LEs needed to implement the system.

Overall the addition of both custom instructions allows for a 10% reduction in code size with a 7.4% increase in total logic element usage, utilizing less software resources but increasing hardware resource utilization. The impact of adding the hardware accelerated custom instructions

is very significant to system performance and nearly insignificant to resource utilization considering the size of modern FPGAs. The time to complete each of the hardware accelerated functions and perform testing was about 6 hours for each module; 4 hours outside of schedule lab time and 2 hours in the lab. While development time is not extremely short, the results are positive. For instance, when rotating a small image of 320x240 pixels, the total time for image rotation goes from 24 seconds in software down to 2.5 seconds with hardware acceleration which is 1/10th the time. This clearly demonstrates how the correct hardware and software partitioning can significantly impact the performance of a system.

Student Response

In order to ascertain whether this lab has improved students' understanding of hardware acceleration, two methods were considered. The first method is the department's standard evaluation of the intended learning objectives for the course. Students are asked the following questions for each intended learning objective:

A. How well do you feel you accomplished the following course objective?
   5) Excellent 4) Very Good 3) Satisfactory 2) Needs Improvement 1) Poor
B. How well do you feel the course presented the material to accomplish the following objective?
   5) Excellent 4) Very Good 3) Satisfactory 2) Needs Improvement 1) Poor

For this course, one of the intended learning objectives is: "Explain hardware performance enhancement techniques (accelerators, DMA, FIFOs)". Prior to the introduction of the image rotation lab, the average response to question A. was 3.75 and question B. was 3.77. After the introduction of this lab the average responses rose to 4.19 and 4.19 respectively. The methodology for teaching DMA and FIFOs did not change prior to and after the introduction of the image rotation lab, so it is concluded that the increase in response value can be creditted to the lab itself.

The second method to evaluate students' response to this lab was a three question survey. This survey was given to the last two class groups to perform the exercise.

	This lab meet its objective of demonstrating hardware/software tradeoffs?	**You were more interested in this lab because the output was image rotation rather than a numeric output?**	**For an embedded design, you are confident in your ability to decide between a hardware vs. software solution?**
Strongly Agree	77%	62%	31%
Agree	23%	23%	46%
Neutral	0%	15%	8%
Disagree	0%	0%	8%
Strongly Disagree	0%	0%	7%

Table 3: Student Response to Image Rotation Lab

The results from the student survey confirm that this lab exercise was a useful addition to the course. Eighty-five percent of the students indicated that the application of image rotation increased their interest in the lab while fifteen percent were neither more nor less interested in the lab topic based on the application. It was assumed from the beginning of the lab development that the given application would increase student engagement in the lab. An area for further investigation would be to determine if students with a neutral response to this question are not interested in the topic at all or if a different application, such as audio processing may increase their interest. The responses to the final question highlights that often students need to be introduced to a topic more than once before they achieve mastery of it. With this in mind, hardware/software partitioning is covered for a second time in the ESD III course.

Conclusion

Conforming to the Technology Accreditation Commission of ABET, Inc's definition of Engineering technology education, "Engineering technology education focuses primarily on the applied aspects of science and engineering aimed at preparing graduates for practice in that portion of the technological spectrum closest to product improvement, manufacturing, construction, and engineering operational functions"[ii], a sequence of project-based courses in embedded systems design have been developed. Laboratory exercises for these courses emphasize the application of engineering principles to solve technical problems. One such problem facing embedded systems designers today is the partitioning of software and hardware resources to maximize performance while minimizing hardware components. A lab activity has been developed to provide students with a hands-on opportunity to evaluate a real embedded system with an image rotation partitioned in software in comparison to the same system with the rotation algorithm implemented in hardware. Empirical results proved that the hardware implementation led to better speed performance. While most students predicted that this would be the case, the experiment also required analytical evaluation of other factors, such as the increase in system hardware resources, needed to support the hardware solution. The laboratory proved to hold the interest of the students, meet its objectives and increase students' confidence in their ability to partition the elements of an embedded system.

As many of today's embedded system applications apply to signal processing, future labs on this topic will be the in the area of accelerating DSP algorithms. Currently a similar lab is being developed where a FIR filter will be used to create an echo of an audio signal. In this lab students will attempt to apply the FIR echo algorithm in software to an audio signal being sampled at a 48K sampling rate. It will become evident that even in its fastest configuration, the NIOS II processor cannot apply the algorithm at the given sample rate, thus establishing the need for a hardware implementation of the same algorithm. Implementing DSP algorithms in hardware will not only provide students with the opportunity to compare hardware and software solutions, it will also provide another integrating experience in the students' education where material from another one of their required courses is incorporated into their capstone experience.

[i] Andraka, Ray (1998). *A Survey of CORDIC algorithms for FPGA Based Computer*, Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, , Monterey, CA. pp191-200

[ii] Chesier, Stephen R. (1998). *Studying Engineering Technology*, Discovery Press, Los Angeles, CA.