THE DESIGN OF AN FPGA BASED EMBEDDED DATA COLLECTION SYSTEM, WITH APPLICATION TO SURFACE PROFILING

by

KYLE D. TIDBALL

B.S., Kansas State University, 2010

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2012

Approved by:

Major Professor
Dr. Dwight D. Day

Abstract

Over the last several years, the use of Field Programmable Gate Arrays, or FPGAs, has become increasingly popular in the embedded systems field. However, FPGAs are typically used only as a coprocessor or dedicated DSP. This project proposes that an embedded system can realize a performance gain over a traditional microprocessor-based design and be made more flexible and extensible by using an FPGA as the primary processing device in the embedded system. Basing a design on an FPGA also allows new features to be much more rapidly developed and integrated into the system.

This will be shown by designing an FPGA based embedded system for Surface Systems & Instruments' Walking Profiler device. The system will include support for rotary encoders, an incline sensor for data collection, and an Ethernet protocol for communication with a Windows computer. The implementation of a sub sampling distance measuring algorithm will be used to demonstrate the tradeoffs between hardware, software, and development times.

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

In recent years Field Programmable Gate Arrays (FPGAs) have become more prevalent in industry for rapid prototyping of hardware, and even system production on a relatively small scale. [1] An FPGA is a type of programmable logic device (PLD) that is typically made up of a large number of logic gates and RAM blocks. However, there are also FPGAs that couple a microprocessor, such as an ARM Cortex, with a small amount of programmable logic fabric. [2]

In the past, FPGAs have been used as coprocessors in embedded systems, but in a very limited capacity. The most common use of an FPGA as a coprocessor was to use it simply as an embedded high-speed DSP device. The system would still maintain a microprocessor as the primary method of control, and offload some tasks, such as image processing or digital filtering to the FPGA. [3] However, this project proposes an alternative solution.

A major advantage of using an FPGA as the main processor in a design, instead of an ASIC or microprocessor, is flexibility. Because FPGAs can be so easily reconfigured and can be made to represent almost any digital circuit, rapidly prototyping an IC is made quite easy. This means you can quickly test a variety of hardware configurations in a relatively small amount of time. This also allows developers to continue to perform hardware functionality updates without forcing users to upgrade to a completely new platform. [1]

A further benefit is that FPGAs can be programmed with what are called Hardware Description Languages (HDL), such as VeryHighSpeedIntegratedCircuit High-Level Description Language (VHDL) or Verilog, to describe the operation of the hardware. [4] HDL allows a hardware engineer to describe the operation of a piece of hardware by writing in what is very similar to a conventional programming language, such as C. However, HDL also allows for an

element of direct interaction with the hardware. For example, it is possible to create custom assembly instructions that boost the performance of an operation run on the soft processor. In this way the logic gates contained within the FPGA can be used to supplement the predefined processor architecture. [4]

Yet another advantage to using an FPGA is system lifetime. According to a white paper published by Altera, FPGAs typically enjoy a lifetime of more than 15 years. [1] This is immense compared to the relatively short lifetimes of most microprocessors at around 5 years maximum. [1] Another benefit that goes along with these long lifetimes is the ability to fairly easily upgrade designs to a newer platform. Both Xilinx and Altera provide tools to make this process as smooth as possible.

The major disadvantage to using Altera and Xilinx FPGAs is the fact that they are RAM based systems. This means that upon a power cycle all information and hardware configurations are lost. [2] To work around this it is necessary to implement some form of flash memory or EEPROM that will download a bitstream to the FPGA when it is powered. The alternative to these types of FPGA is any number of devices from Actel. Actel manufactures FPGAs that are paired with small, powerful microprocessors, such as ARM chips. These Actel devices allow the design to be flashed to memory, and will therefore maintain the hardware configuration even after a power cycle.

FPGAs in Embedded Systems

For some of the reasons listed above, using an FPGA in the design of an embedded system is an increasingly popular choice among digital hardware engineers. [3] It gives the familiarity of using a microcontroller combined with hardware flexibility. However, one of the many reasons an FPGA is easier to work with than a microcontroller is the architecture of the

system. When building a system on an FPGA, usually it is divided into several segments or blocks. All of the blocks run in parallel with each other, making it possible to have dedicated "cores" or peripherals running in parallel with a processor core. This is also a simpler design than building an interrupt driven system, such as would most likely be done on a microcontroller.

An embedded system is a combination of sensors and electronics designed to perform a specific task. The electronics are most often a small processor (i.e. a microcontroller) and some support circuitry, along with at least one type of interface or bus. Thanks to the explosion of technology in the last 10 years or so, embedded systems are nearly everywhere now.

Since an embedded system is typically designed around interpreting data from several sensors simultaneously and reporting the data to a controller, this structure of individual parallel cores is ideal. By running these cores in parallel, the design of the system can be optimized, and the performance maximized. Being able to sample many different devices simultaneously will allow the system to realize a performance gain over a microprocessor based interrupt driven system, which can only sample one input at a time. A fairly standard FPGA design connects these parallel cores directly to individual sensors and therefore has each core act as dedicated hardware for each sensor, reporting data collected to a "master" core or processor built specifically for handling and processing the data. These cores can also be directly connected together to share data or trigger each other. For example, a core watching inputs from a distance sensor can be connected directly to another core, and trigger the second core to sample its sensor only when a certain distance has been achieved.

Distance Measuring

Generating accurate surface profiles requires two things: a way to measure the vertical features, and a way to measure horizontal distance. To measure the vertical distance, an inclinometer is used. The inclinometer is sampled at discrete intervals, and integrated along the length of the surface to reconstruct the data. To measure horizontal distance a quadrature rotary encoder is used. A quadrature encoder transmits data via two digital channels (A and B) 90 degrees out of phase with each other. This means that only both channels can never transition at the same time, eliminating false triggers or noise due to two bits changing simultaneously. These two channels indicate travel, and based on which is high before the other, direction of rotation can be determined. The pulses these two channels make while moving can be thought of as counts or clicks. Because the SSI devices sample at a 1" interval, some relationship between encoder counts and distance must be established. This is done with a calibration routine in the software governing the device.

However, if there is an error in the distance measurement, there can be severe ramifications on the accuracy of the surface reconstruction. Because this distance measurement is so critical, the calibration must be performed with extreme care.

Application to Road Profiling

The test track used in this study is 541.5 feet long. Correctly calibrated to 405.9045 encoder counts per inch the system used will measure 2637567 counts, for the 541.5 feet. If the system did not keep track of the partial distance encoder counts and simply rounded the number of counts per inch to 405, the system would report only 2631690 counts, or 540 feet, 1.5 feet short of what the surface actually is. This lack of distance will falsely influence the roughness

measurement of the surface. It is also difficult to make accurate comparisons between different profiling devices if they are all reporting different lengths for a surface. This problem can be overcome, to some extent, by resampling data to a set interval, but it will not be nearly as accurate as comparing devices reporting the same distance measurement. Therefore, to ensure distance is correctly and accurately reported, some method of keeping track of the partial encoder counts, or sub sampling, is necessary.

Digital Differential Analyzer

The digital differential analyzer (DDA) algorithm is common in the field of computer graphics. A DDA used to be an electromechanical device that was able to solve differential equations numerically. [5] Today, a DDA is simply implemented as an algorithm, and provides an efficient method for simple linear interpolation. A typical application is computer graphics, most often for drawing lines as groups of pixels.

The main idea is that slope of a line can be thought of as the differential equation $m = \frac{\Delta y}{\Delta x}$, where $0 \leq m \leq 1$. As the algorithm moves along each x coordinate, there must be a corresponding y coordinate. That coordinate is found by the equation $\Delta y = m\Delta x$. The standard DDA algorithm requires both x and y to be whole numbers. However, in the case of drawing a line as pixels, as the algorithm moves along the x-axis increasing by 1 each time, y will need to be rounded to remain a whole number. This is a relatively time consuming and processor intense algorithm due to all of the floating point operations that must take place. Because of this drawback, modifications to the standard DDA algorithm, such as Bresenham's algorithm discussed below, will result in calculations much less resource intense and much faster. However, if a Floating Point Unit is available, it is a very convenient way to do simple linear

interpolation. Another issue to be aware of is the fact that floating point calculations inherently have an error due to rounding. Although almost always very small at first, this error can cause the wrong y coordinate to be selected and misrepresent the actual data.

Bresenham's Algorithm

A more efficient implementation of the DDA algorithm was realized by IBM researcher Jack Bresenham in 1962. [5] The DDA algorithm has some performance drawbacks, mostly due to the necessity of a floating point operation for each pixel. Bresenham's algorithm uses only whole numbers, and therefore can realize better performance than the standard DDA implementation.

When attempting to draw a horizontally sloped line as a series of pixels on a computer screen, for each x coordinate, there is a decision that must be made regarding the appropriate choice of pixel for the y coordinate. In the case of a horizontally sloped line there will be an upper pixel and a lower pixel that can be used to represent the y coordinate. Bresenham's algorithm uses the equation $d = a - b$, where d is referred to as the decision variable, a is the distance from the line to the upper pixel, and b is the distance to the lower pixel. Therefore, if d is positive ($a > b$) the lower pixel is used, otherwise the upper pixel will be used.

However, this will still require a floating point operation, as a and b will not be whole numbers. The solution to that is to instead to start with the equation of a line:

$$y = mx + b, \text{ and set } A = \Delta y, B = -\Delta x, C = (\Delta x)b$$

The line equation can be rewritten in terms of $A, B, and\ C$:

$$f(x, y) = 0 = Ax + By + C$$

With some simplification, this equation can be turned into the form:

$$d = 2\Delta y - \Delta x$$

This will be more efficient than doing floating point operations, but can still be further improved by the realization that d will either be incremented by $2\Delta y$ if $d < 0$, or by $2(\Delta y - \Delta x)$ if $d > 0$ every time. This will result in simply a sign check and an integer addition for each pixel to be drawn.

To summarize, then, Bresenham's algorithm looks for a threshold to be crossed in the variable d, which represents the error component of the operation. Once the error is larger than the threshold, the whole system can move to the next item (in this case, pixels). This has been proven to be advantageous in computing inches of travel based on a distance encoder that has a counts per inch interval that is not a whole number, similar to the slope of a line, m. In fact, this scheme is so effective, a modified version of this algorithm is used in the current range of SSI systems.

Chapter 2 - Distance Measuring Process

All of the devices SSI manufactures rely on 1" distance sampling to generate accurate data. To accomplish this, the system uses a previously mentioned quadrature shaft encoder connected to one of the drive wheels of the vehicle. Because the distance measured will change depending on the size of the wheel and the precision of the encoder, the system governing the distance encoder must be "calibrated" to the wheel size. For the WalkPro, the encoder used has 4096 counts per full revolution coupled to 8" diameter wheels.

A major issue with this system, and in fact all profiling systems, is accuracy. For example, the calibration described previously yielded 405.9045 counts per inch. Then even after a few inches of travel the system will be off by several counts. If the collection is very long at all,

these errors will quickly add up. An incorrect length of the data can have a large effect on the resulting computations run on that data. [6] [7] To counter this, an algorithm has been developed that will keep track of the partial counts traveled and add a full inch when appropriate. This algorithm is referred to as encoder sub sampling. Two different implementations of the sub sampling have been developed: one using floating point numbers designed to be used on a processor, and one that uses integer math to keep track of the encoder value, designed to be run in a hardware peripheral.

Calibration Procedure and Error Analysis

To ensure accurate reporting of distance traveled, the system must first be calibrated to the size of wheel and precision of the encoder being used. This calibration will yield the number of encoder counts that will occur every inch based on moving the device along a known length of track. The more accurate the calibration, the more accurate the distance measurements will be. However, a slight error in calibration can have quite dramatic effects on distance measurements.

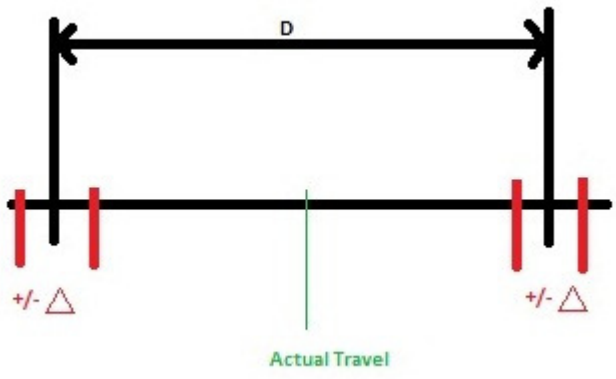

Figure 1 - Calibration Error Bound

As figure 5 shows, there is an error bound at either end of the distance calibration due to unit placement, but also an error is introduced by inaccurate measurement of the calibration

distance. How accurately the user can line the device up on both the starting and ending points of the calibration track will help to determine this error bound. For the calculation of the calibration value, the following equation is used:

$$(D \pm 2 * \Delta) = Actual\ Distance\ Traveled$$

The counts the system would see will be:

$$(D \pm 2 * \Delta) * Ideal\ Counts\ per\ Inch$$

The number of counts per inch that is then calculated would be:

$$\left(\frac{(D \pm (2 * \Delta))}{D}\right) * Ideal\ Counts\ Per\ Inch = Calculated\ Counts\ per\ Inch$$

In these equations D is the distance to be traveled, 2*Δ is the error accumulated at both ends of the calibration, calculated counts per inch is the number of counts per inch found from the calibration, and ideal counts per inch is the number of counts per inch for the actual system. For example, set D = 300” (25 feet), delta = 1/8”, and ideal counts/inch = 400. In this case, the counts per inch will come out to be 400 $\pm$ 0.3 for a best case scenario, assuming the reported counts per inch is exactly the same as the ideal counts per inch. This seems like an insignificant discrepancy, but if the length of a collection with this erroneous calibration is 528 feet, 2534400 counts, the error at the end will be $\pm$0.44 feet, or $\pm$5.28 inches. Half a foot in 528 feet of total travel is an unacceptable error. As previously stated, the algorithms for creating a roughness

measurement of a surface are very sensitive to mistakes in distance, and this kind of error can cause a large variance in the quality of the recorded data.

Requirements for Sub Count Measurement in Road Profiling

The requirements for doing a calibration on a road profiling system may be found in the American Association of State Highway Transportation Officials (AASHTO) document R 56-10: *Standard Practice for the Certification of Inertial Profiling Systems*. An excerpt from the section detailing the list of specifications follows:

8.4 Distance Measurement Index Test – Test the accuracy of the DMI on one test section.

8.4.1 Distance Measurement Index Test Section – Provide a section for DMI testing. The test section will be at least 1000 ft in length, with proper lead-in distance and a safe stopping distance available. This test section may incorporate the test sections that are used for accuracy and repeatability testing. Clearly mark the starting and ending points of the test section. Measure the distance between the starting and ending points with a temperature-compensated steel tape, pulled taut but still following the pavement contour.

Note 2 – The same runs may be used for verification of DMI accuracy as are used for testing accuracy and repeatability of the collected profile. The ideal test of DMI accuracy would be performed over a known but undisclosed length of pavement.

8.4.2 At least three auto-triggered runs at the lowest and highest test speeds of the candidate inertial profiler shall be made on the designated length of pavement in the prescribed direction of measurement. At the end of each run, record the reading from the profiler's DMI. For high-speed profilers, this results in at least six values. Collection speed governed devices should make at least five runs.

8.4.3 Distance Measurement Index Accuracy – Compute the absolute difference between the DMI readings and the known distance of the path tested for each run. The average of the absolute differences for both the high-speed and low-speed runs, if applicable, must be less than 0.15 percent to pass the test.

The crucial points to note here are sections §8.4.1 and §8.4.3. Section 8.4.1 establishes the procedure for measuring and marking off the calibration distance, something that was discussed previously, and is a key point. If the distance measured is off, then the whole calibration, and all subsequent data collections, will be incorrect. As section 8.4.1 indicates, there has been an attempt to compensate for this by using a temperature adjustment for the steel tape

used to mark off the length of the calibration surface.

Section 8.4.3 explains the criterion for passing a distance calibration. Limiting the absolute average error to 0.15% helps prove the consistency of the measurement device. However, the document also states that the calibration surface should be at least 1000 feet in length. 0.15% error in 1000 feet will allow for an error of ±0.5 feet. As mentioned previously, half a foot error in a shorter collection (i.e. on the order of 540 feet) can produce incorrect results in the roughness index. A sub count sampling algorithm, such as will be described here, will help to reduce these errors during calibration, and therefore help to maintain reliability of the system.

Chapter 3 - Sub Count Sampling

Floating Point Approach

Because the number of encoder counts per inch will most likely not be a whole number, it is important to keep track of the partial encoder counts as the system is moving along. This will ensure the system reports consistent, accurate distance readings. To capture these partial counts, two different implementations of a sub sampling algorithm have been designed.

The first is a floating point implementation of the sub sampling. This algorithm is relatively simple, and simply uses floating point numbers to keep track of the partial counts. The way the algorithm starts is by reading the count value from the encoder. This number is a 64 bit integer number, so to create a floating point number, this number is cast to a float. Then a threshold is checked to see if the system has moved an inch forward, or an inch backward. The thresholds are initially set by the calibration value for counts per inch. That is to say, the forward travel threshold will be set to the positive value for calibrated counts per inch, and the backward threshold will be set to the additive inverse of the calibrated counts per inch, or opposite. After

one of the thresholds is reached, both thresholds are set to new values by simply adding the calibrated counts per inch to the current value of the thresholds for the case of forward motion, and subtracting for the case of the backwards motion. Simply by using floating point numbers the partial counts are accounted for, and no further processing is necessary to ensure accuracy.

However, a major drawback to this method is the maximum distance that can be traveled before the distance count will begin to lose precision. Using a single precision floating point number, which allows for 24 usable bits, the precision the system currently maintains is a 12 bit fractional part. To maintain this precision with a single precision floating point number, that allows 12 bits for the whole number counts. Therefore, 2^12/405 = ~10 inches. This is quite clearly not enough distance to be able to collect any meaningful data whatsoever. Switching to double precision floating point numbers, on the other hand, will give 41 bits for the whole number counts (53 usable bits minus 12 for the floating point portion of the number). This will result in 2^41/405 = 85,695.9 miles of possible collection before there is a loss of precision.

Another point to consider is numerical error in the accumulation of distance counts over larger distances. A major issue with floating point numbers is that there is a rounding error inherent in computations for which they are used. Although this error is usually small, after a relatively short amount of time this error can add up to a significant amount.

DDA Approach

As discussed earlier, using a DDA is another possibility for doing this kind of sub count sampling. The implementation would be a bit more complex than the simple floating point implementation or the integer implementation. The idea in this style of sub count sampling would be to calculate an error parameter that the current counts were away from a threshold, and then depending on that error increase the threshold appropriately.

There would be three main components to this implementation: the whole integer count value, the floating point portion of the count value, and a floating point error parameter. The basic principle would be to sample the counts check if the threshold has been reached. If it has, then an equation such as the following should be used:

$$error = threshold - counts$$

This should yield an error that is between 0 and 1. Once the error accumulated to 1, an extra count would need to be added to the total. In this way the running error would be used to correct for the partial encoder counts and the adjustment applied periodically during the collection. Although this seems to be a nice, clean way to keep track of partial encoder counts and correct the distance measurements, this method is quite resource heavy. Doing mixed floating point and fixed point calculations tends to take a large amount of time, and doing floating point calculations without an FPU is a large waste of computing power.

The maximum distance this algorithm would make it possible to travel would be similar to the max distance when using the integer implementation. The reason for this is that the distance would still be stored as an integer, while the error would be stored as a floating point number. The total distance, therefore, could be on the order of 160 miles, and the error could be allowed to have quite good precision, as the floating point number would not need to be used for anything else.

The error in this implementation would be less than what the straight forward floating-point implementation would have, and slightly more than the error of the integer implementation. The majority of the error would again be from the error inherent in floating point calculations. However, because only the decimal error, which should be between 0 and 1, is stored in the floating point number, much more precision could be maintained. This means that the precision

of the error calculation would depend on the precision possible in the floating point number that represents the number of distance counts from the encoder.

However, due to time restrictions, this type of approach was not undertaken, although the integer sub counting was derived from a modified Bresenham's algorithm.

Integer Approach

The other type of sub sampling developed is an integer based sub sampling. The current WalkPro system uses this type of sub sampling algorithm. This version of the algorithm has been implemented directly in hardware in a custom peripheral on the FPGA, whereas the floating point version has been run in the PowerPC processor on the FPGA. The integer based implementation also uses a technique called dithering to maintain correct distance over longer surfaces.

In the integer implementation there are two parts to the encoder reading: an integer part, and the fractional part of the number. As with the floating point implementation, the integer part is initialized to the calibrated counts per inch. However in this case, the counts per inch are truncated to only the integer part of the number. The decimal part is then multiplied by the number of rollover counts for the hardware, which in this case is 4096, and converted to an integer. So for the case of the calibration value explored earlier (405.9045), the integer part would be set to 405 and the fractional part would be set to 3075.

Similar to the floating point version of the algorithm, the value is constantly read from the encoder channels and checked against the forward and backward travel thresholds. When one of the thresholds is reached, the values for the thresholds are incremented in the same way; that is, the calibration value for counts per inch is added to the thresholds for forward motion and subtracted for backwards motion. However, this is where the dithering comes into play. The

algorithm keeps track of the fractional counts, and when they roll over a count is added to the integer part of the count. So for several inches the integer portion of the count would look something like: +405, +406, +405, +406, +405, +406. This alternating pattern is known as dithering, and is in fact a modified Bresenham's algorithm.

Using this implementation of the sub count algorithm the maximum distance that the device would be able to travel before the distance count rolled over to zero is approximately 167 miles. Based on how profiling devices are typically used, it is highly unlikely that a single collection would ever exceed 10 miles, let alone 167 miles. However, this is one advantage of this implementation over the floating point implementation.

Chapter 4 - Results

The figures below represent an analysis of the timing of each of the two types of sub sampling algorithm implemented. Additionally the floating point version of the algorithm was run with and without a floating point unit in the PowerPC. The goal of this analysis was not only to explore the amount of time it took the various implementations to process encoder samples, but to do that in as non-intrusive a manner as possible. In all of the timing diagrams of the floating point versions (FPU and no FPU) below, the time measurement was taken by setting a GPIO pin high at the start of the encoder processing, and setting the same pin low when processing was finished. This is what can be seen on pin D0. Pin D2 was the same setup, but only triggered when an inch had been traveled and one of thresholds had been reached. Pin D1 is the ADC chip select pin. This pin is provided as a reference to what else is going on in the system and how the encoder processing affects inclinometer sampling. Great care was taken to

ensure that the placement of the GPIO toggle minimized delay on the operations themselves and artificially inflated the timing measurements.

Timing Analysis

Figure 6, below, shows the floating point version of the sub sampling algorithm, with an FPU in the processor. As the image shows, reading the encoder value and checking against the thresholds takes approximately 5 microseconds if an inch has not been traveled. When an inch has been reached, more processing is required, and therefore the time taken will increase. The long pulse in D0 corresponds to the pulse in D2, and represents an inch being reached. As the image shows, the amount of time it takes to process the encoder is now around 8 microseconds. As the processor is running at 200 Mhz, this is a fairly significant increase in time. However, 8 microseconds is not an unreasonable length of time to process the encoder.

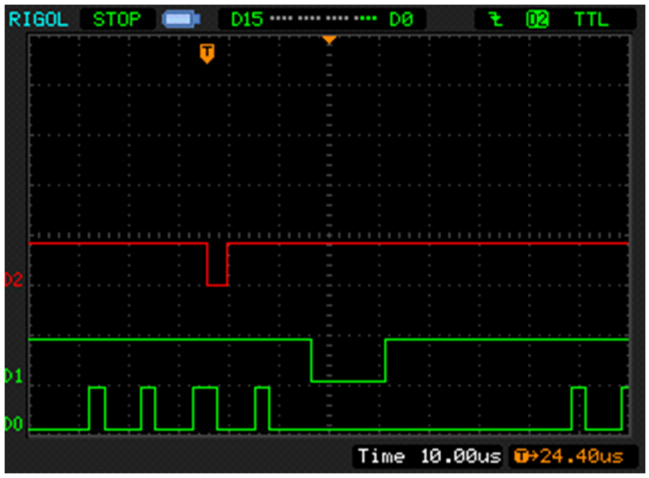

Figure 2 - Encoder Sampling with FPU

Figure 7 shows the same information as figure 6, but this time the processing is done with the integer version of the sub sampling algorithm in the custom peripheral created. In this figure, D2 is an output directly from the encoder peripheral to a pin. This output represents when an

inch has been reached and is written high when processing starts and back low when processing is finished. D0 is the time it takes to read a register from the encoder IP to check if an inch of travel has occurred, and D1 is the ADC chip select again. As this figure shows, the time taken to read a register from the encoder is around 3 microseconds, and the time it takes the encoder peripheral to process an inch is almost undetectable. Simulation results show the processing time for the case when an inch is reached is on the order of 5 nanoseconds.

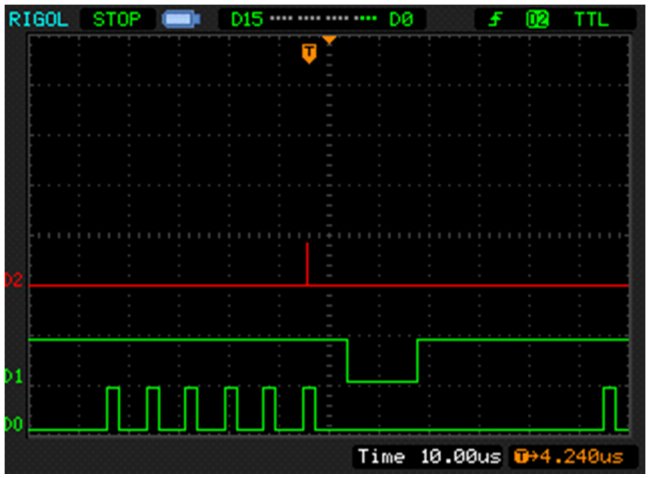

Figure 3 – Encoder Sampling in Hardware

Finally, figure 8 shows the floating point implementation of the sub sampling algorithm with no FPU in the PowerPC. The signals are the same as they were in figure 4. Here a large lag can be seen. Normal encoder processing (i.e. just checking thresholds) takes nearly 40 microseconds, and when an inch has been traveled, the processing takes almost 50 microseconds. This is more than a 625% increase in processing times by removing the FPU. The other point of interest is that now the timing of the ADC is skewed. As the figure shows, there is no longer an even interval between ADC sampling. It is also important to note that if the encoder is being processed when the software is scheduled to send a pulse to the ADC the encoder processing is finished, then the ADC pulse is sent.

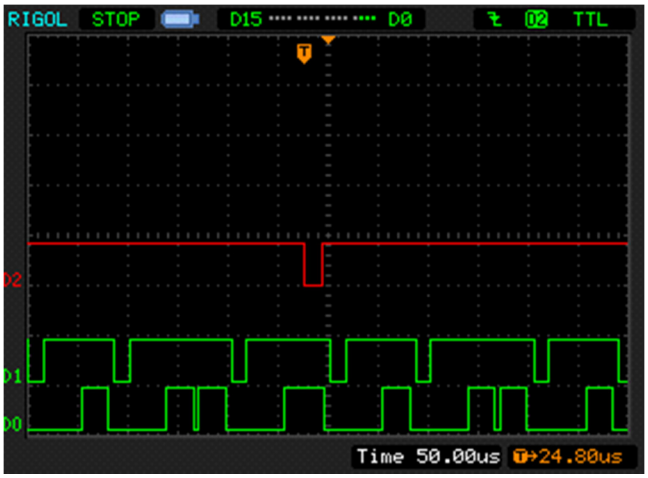

Figure 4 – Encoder Sampling with No FPU

The result of this timing analysis shows the result hoped for. The floating point implementation with no FPU simulates true floating point numbers in software, and therefore will be much slower. The hardware peripheral was expected to be much faster (1600% faster than the FPU implementation, in fact) than either of the two software implementations.

Floating Point

Figure 9 shows a segment of the reconstructed data collected with the floating point implementation of the sub sampling versus a reference collection of Plymate Lane, SSI's test track, taken with a different device. An important note is the inclinometer requires an acceleration cancelation to be applied to the data from it, as it cannot distinguish between tilting and being quickly accelerated (or decelerated). That aside, the two traces look remarkably similar, and the distances are nearly identical.

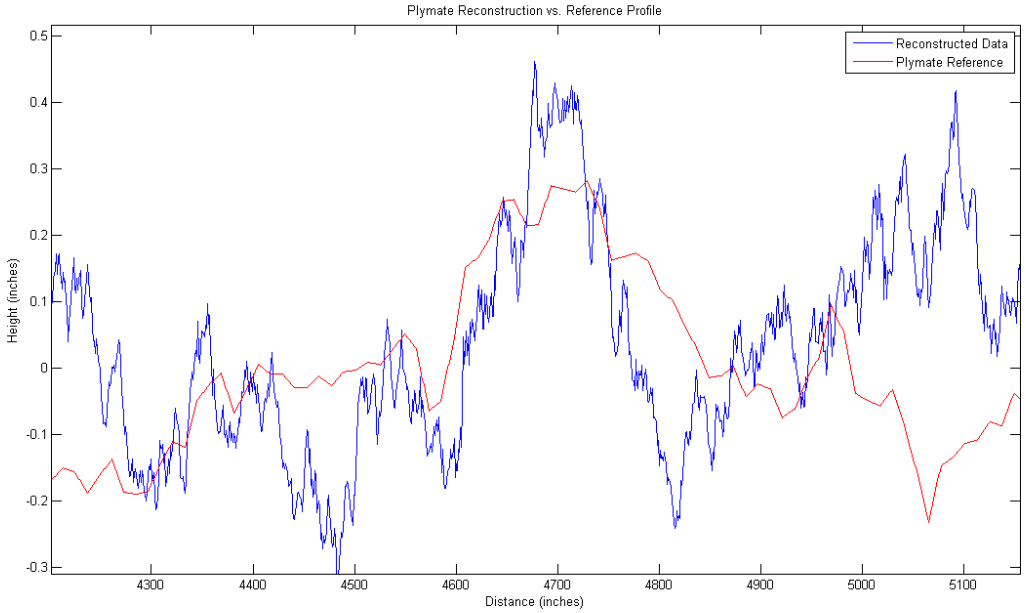

Figure 5 - Floating Point Sub Sampling

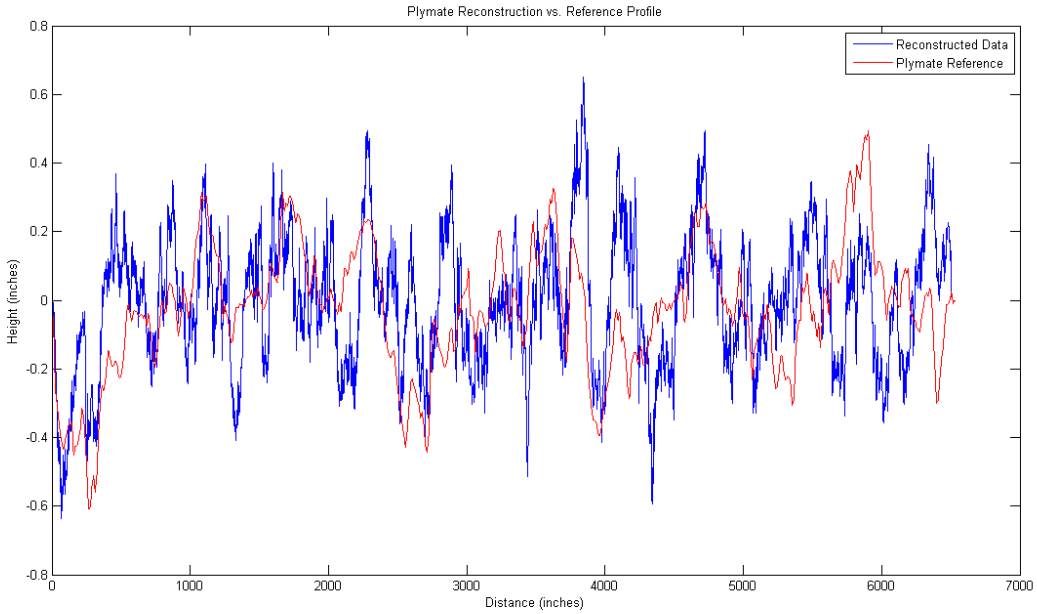

Figure 6 - Whole Plymate Collection with Floating Point Implementation

Hardware

The following two plots are a collection on the same surface, but this time using the custom peripheral created directly in hardware. Figures 9 and 11 look shockingly similar, but thanks to careful collection technique, that is to be expected.

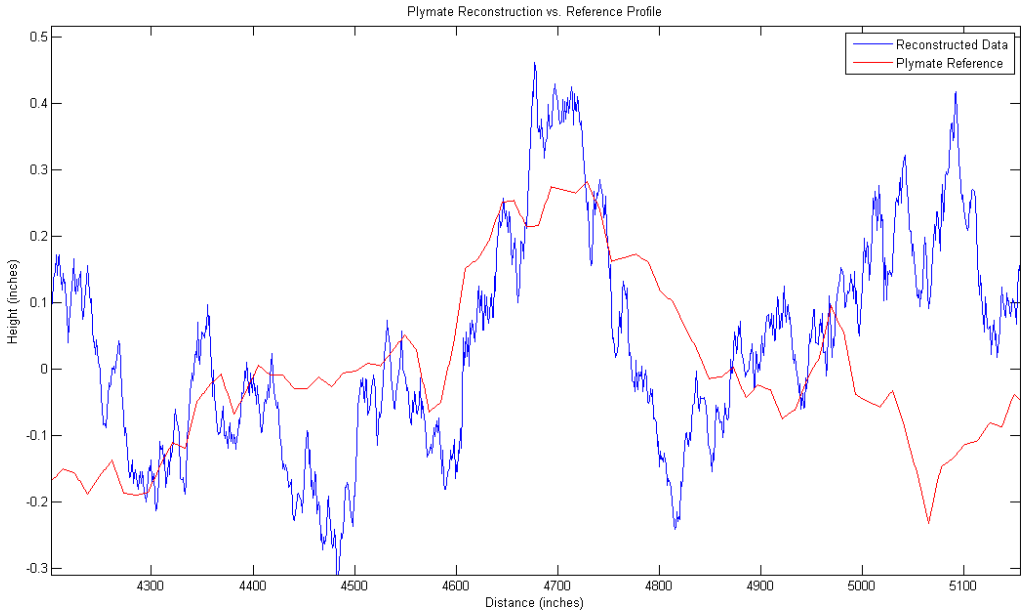

Figure 7 – Hardware Sub Sampling

Figure 12 also seems to have cleaner data overall, possibly due to the timing delays the slower FPU encoder handling forced on the inclinometer. Again, if acceleration compensation had been applied, these plots would show results very similar to the reference device. It is also worth noting that both of these sets of data, as well as the reference data, have had the linear trend introduced by integration removed to make the data analysis easier.

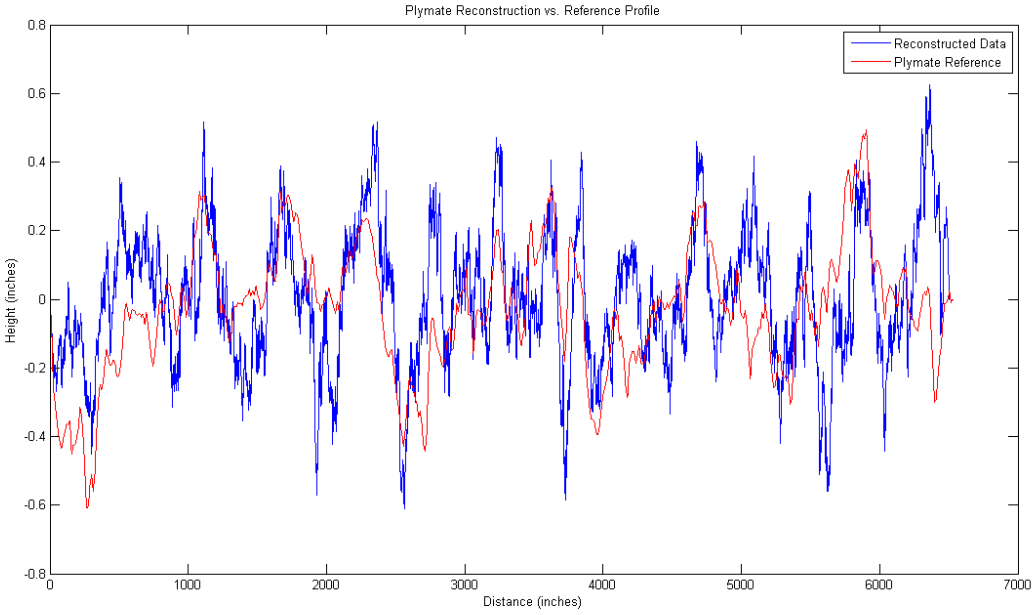

Figure 8 - Whole surface hardware

Chapter 5 - Conclusions and Future Work

This project has shown how sub sampling of the distance encoder is necessary in order to ensure an accurate measurement of a surface with a profiling device. Two different methods of implementing that sub sampling have also been shown, and the differences between servicing a sensor on the FPGA versus writing software that runs on the processor within the FPGA to service that same sensor. As with any major design decision, there is a tradeoff between the two implementations, and a choice that must be made depending on the circumstances. The integer-based implementation uses less RAM, and uses more of the FPGA, but has an extremely quick processing time. Whereas the floating point version of the sub sampling uses more RAM and less of the FPGA's resources, but takes longer to process encoder data. However, the floating point implementation also is much less complex, and took nearly no time to develop comparatively.

Development Time

When developing hardware and software, a critical metric to be aware of is development time, or how long it will take an engineer to implement a new feature. This was also a major consideration for the FPGA WalkPro system. The floating point implementation of the sub sampling algorithm was hashed out in the span of one afternoon, and was working later that same day.

Conversely the integer implementation of the sub sampling took the better part of a whole semester to develop and make fully functional. There was also a bit of a learning curve associated with Xilinx's environment during the development of the integer implementation, but that was a minor part of the development time. One of the more difficult parts of developing the hardware peripheral was testing. With the floating point implementation done all in software, recompiling the software and running tests takes a matter of minutes. Values can easily be checked by writing variables to a console output via a UART connection.

Hardware testing, on the other hand is much more time consuming. Some amount of testing can be done in simulation environments such as ModelSim, or Xilnx's version, iSim. These two pieces of software allow an engineer to write Verilog code that will trigger the IP to be tested. The results of the simulation are readily available in the form of a waveform generated by the IP.

Unfortunately, simulation only goes so far. To verify operation, eventually it must be tested in a real environment. This requires generating the hardware on the FPGA, which is an incredibly time consuming process. The development software must attempt to convert the system laid out into gates, and then place those gates on the chip selected. Xilinx software uses what is referred to as an "annealing" process for placement of the design. The software places a

large portion of the design, then checks to see if that was a good place. It repeats this process a number of times based on the level of effort it is set to.

This long build time is annoying during testing. If a mistake is discovered, the hardware must be rebuilt. If it is decided that more signals need to be added for debugging purposes, the hardware must be rebuilt. For this reason, software debugging is much preferred on FPGA systems.

Performance

Because it is running on the PowerPC, the floating point implementation of the sub sampling algorithm does not use any of the FPGA's configurable resources. However, it does use block RAM, which is a limited commodity on a chip the size of a Virtex- 4. Just the floating point implementation of the sub sampling uses 4% of the total BRAM on the FPGA, which is quite a bit. The design actually uses 100% of the BRAM with the floating point sub sampling in the software.

Number of Slices:	507 out of 5472	9%
Number of Slice Flip Flops:	478 out of 10944	4%
Number of 4 input LUTs:	721 out of 10944	6%
Number of IOs:	272	
Number of bonded IOBs:	0 out of 320	0%
Number of GCLKs:	2 out of 32	6%

Table 1 - System Utilization of Encoder IP

As Table 1 shows, the integer based encoder peripheral uses a not insignificant portion of the FPGA. However, if there is not much else that needs to be put into the design, then the

utilization is not necessarily a problem. The implementation that uses this version of the sub sampling uses 4% less block RAM, which is very important.

The overall outputs of both of these implementations are very similar in terms of data integrity. Both show almost identical distance measurements, and the only perceptible difference is the slightly cleaner reconstructed data from the integer based sub sampling run on the hardware peripheral.

Timing of Dithering vs. No FPU

As was shown earlier, the dithering version of the integer implementation is much faster to handle the encoder hardware than the floating point implementation with no FPU. However, it is not necessarily required that the processing be that fast to generate reliable surface data. As figures 4 and 5 show, the floating point methods, while slower that the direct hardware method, still provide reliability in the processing of encoder data, and do not cause a loss of data received from the inclinometer.

The integer sub sampling hardware peripheral, however, absolutely crushes the floating point implementation in terms of speed. Because the IP is implemented directly in the hardware, there is very little delay getting the data from the pins to the hardware that does the processing. However, for the floating point implementation, the data must get from the pins, through the encoder IP and onto the bus to the PowerPC. This most likely contributes to the length of time it takes the floating point implementation to process the encoder data.

Future Work

There are a great many goals this system has yet to realize, but some of the most important are: expand the system to work for all SSI systems, including the high-precision TOPCON system, create a dual processor system, with one core monitoring peripherals, and the

other for DSP and communication only, and build a board around this system with integrated power management and distribution, as well as instrumentation.

Throughout the last several years, SSI has begun to fulfill a major design goal of having a universal suite of electronics that can be used for all systems sold. In order to continue that trend, this system will need to be expanded to include: IP cores that will handle at least two different styles of laser range finders, multiple laser rangefinder cores in the same system (potentially up to 11 lasers simultaneously), GPS support, the ability to process several accelerometers sent through ADCs, and different types of incline sensors. One of the more difficult tasks associated with implementing a system this large is synchronization. Currently, this large system relies on dual accelerometers, sampled every millisecond, to synchronize everything. However, with the extra processing power afforded by parallel cores running at 100 MHz + it should be possible to sample the accelerometers at a much higher frequency, or implement a different synchronization method all together. Fortunately because the decision was made to implement an Ethernet communication protocol, this type of large scale system should not encounter bandwidth limitations.

Another goal for this system is the creation of dual processors within the FPGA. The reasoning behind this goal is that one processor would be used solely for handling all of the various peripherals and then pass the data to the other processor. The second processor could then do any digital filtering deemed necessary, pack the data up into packets, and send it to the Windows PC through the Ethernet link. This would increase the performance of the system over using a single processor by a significant amount.

Because FPGAs have blocks optimized for DSP on them, it makes sense to perform any necessary digital filtering on the FPGA. However, in the case of a system like this that has many

sensors to monitor and process, increasing the number of tasks the processor must complete is not ideal. Therefore moving the filtering to another processor that does very little else is a preferred solution.

The other task that is somewhat time-consuming is creating Ethernet packets. There are quite a few parameters that must be set in an Ethernet packet header (this system is using UDP). Packing the data up into packet takes some time as well. The structure of the data in the packets is very similar to the way the current system sends data. For each different type of data that can be sent (Inclinometer, laser, etc) there is a corresponding 1 byte code that identifies the type of data. The data sample to be sent directly follows its byte code. Because this bit packing can also take some time, compared to the speed at which the sensors need to be sampled, it can help streamline the process to have a separate processor for both packing up messages and sending them across the Ethernet link, as well as decoding the incoming messages.

The third of the major goals for this project is laying out a printed circuit board (PCB) based around an FPGA. The solution this project is currently using is a prototype, and not ideal for a permanent solution. However, it may not be necessary to completely replace the system's main board. It would be feasible to have a smaller board made that can connect to the current electronics.

Chapter 6 - Bibliography

[1] Freescale Semiconductor, MC9S08QG8 HCS08 Microcontrollers, 2009, Rev. 5.

[2] Huang Lei, "Design of Embedded Data Acquisition System Based on FPGA," in *2nd IEEE International Conference on Computer Science and Information Technology*, Bejing, 2009, pp. 530-534.

[3] Digital Equipment Corporation, Intel Corporation, and Xerox Corporation, The Ethernet, A Local Area Network. Data Link Layer and Physical Layer Specifications, Version 2.0, 1982.

[4] Xilinx Corporation, ML401/ML402/ML403 Evaluation Platform, 2006, Rev. 2.5.

[5] Altera Corporation, Five Ways to Build Flexibility into Industrial Applicaions with FPGAs, 2011, WP-01145-1.0.

[6] Future Technology Devices International Ltd, FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC, 2010, Rev. 2.11.

[7] Motorola, Inc., SPI Block Guide, 2000, Rev. 2.

[8] David A. Patterson and John L. Hennessy, *Computer Orginization and Design: The Hardware/Software Interface*, 14th ed. Burlington, USA: Morgan Kaufmann Publishers, 2009.