Duke University
Department of Electrical and Computer Engineering

ECE154 Semester Project

APLS - An Acoustic Pinger Location System For Autonomous
Underwater Vehicles

December 14, 2006

Matt Johnson
Andrew Waterman

Contents

List of Figures

1 Introduction

In the field of autonomous underwater robotics, it is often desirable, and indeed necessary, to "ground truth" the position of the robot to a known control point, by detecting and/or navigating to a beacon of some sort. The beacon is necessary because conventional means of correlating on-board position data with a known global coordinate system, such as GPS, do not work well in an underwater environment. One popular type of beacon is an acoustic pinger, which uses a transducer to transmit a relatively high-intensity pressure wave above 20kHz. In this paper, we propose an algorithm and hardware/software implementation for receiving this signal via an ultra-short baseline (*USBL*) array of hydrophones, and calculating a bearing to the pinger. This bearing information can be used to navigate to the pinger, or gain additional knowledge about the vehicle's position.

In this paper, we will address all aspects of the conception, design, implementation, and testing of the apparatus, which we will call *APLS*, or the Acoustic Pinger Location System. First, we will elucidate the theory behind our approach to pinger location. Next, we will enumerate the high-level design requirements and constraints which shaped the system as a whole. This discussion will be followed by a description of the iterative process which yielded a finished hardware and software design. Next, we will discuss the process of constructing and debugging the system after the basic design was complete. Finally, we will characterize the finished system, assess the extent to which it met the original design goals, and provide a post-mortem view of opportunities for improvement, not only in the next revision of APLS, but future design projects in general.

2 Theory Of Operation

It is possible to determine the bearing angle to the beacon by measuring the pairwise differences in the time of signal receipt at the hydrophones in the array. A noncollinear array of three hydrophones is needed to uniquely determine the bearing[1].

The plane wave approximation is a useful and reasonable one to make for the purposes of bearing angle calculation. It is valid if the distance between the array and the beacon is large with respect to the dimensions of the array; in practice, this stipulation is nearly always met because the size of the array is constrained, for reasons explained later.

It is also assumed in this analysis that the speed of sound in water is invariant (but not necessarily known). This simplification is reasonable if the density and temperature of the water in the region of interest are roughly constant.

2.1 Mathematical Foundations

Refer to Figure 1 for a diagram of an equilateral hydrophone array. In this figure, a signal from the beacon, approximated as a plane wave, approaches from the trigonometric angle θ. Right-triangle trigonometry can be used to determine relationships between the pairwise time differences and θ. The two triangles of interest are denoted by solid lines. One side length of each is known; it is the distance between two hydrophones in the array. Another side of each is given by the distance traveled by the wave during a pairwise time difference. An unknown angle related to the desired quantity θ also appears in each triangle. Writing trigonometric equations about these unknown angles, it is possible to determine a closed-form expression for θ in terms of the time differences $t_3 - t_1$ and $t_3 - t_2$, the ratio of which we define to be ρ.

---

[1]A fourth, noncoplanar hydrophone allows for the further calculation of the angle of elevation to the beacon.
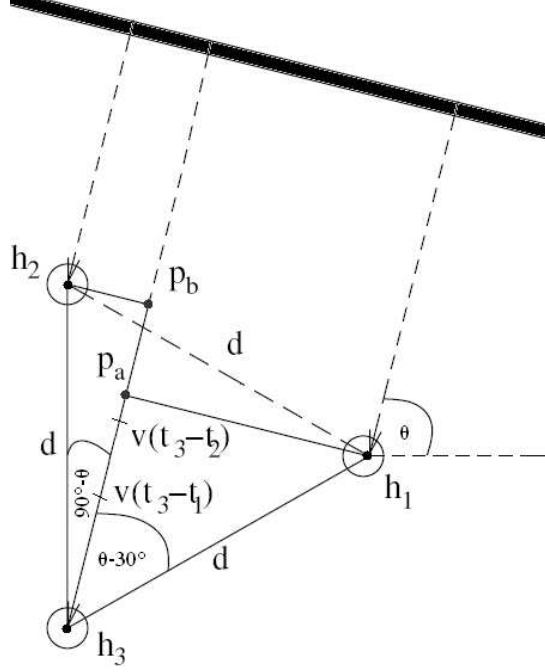
Figure 1: The geometry of the hydrophone array

$$cos(90° - \theta) = sin(\theta) = \frac{v(t_3 - t_2)}{d} \tag{1}$$

$$cos(\theta - 30°) = \frac{v(t_3 - t_1)}{d} \tag{2}$$

$$\rho \equiv \frac{t_3 - t_1}{t_3 - t_2} = \frac{cos(\theta - 30°)}{sin(\theta)} \tag{3}$$

$$\rho = \frac{cos(\theta)cos(30°) - sin(\theta)sin(30°)}{sin(\theta)} \tag{4}$$

$$\rho = \frac{\sqrt{3}}{2}cot(\theta) + \frac{1}{2} \tag{5}$$

$$cot(\theta) = \frac{2\rho - 1}{\sqrt{3}} \tag{6}$$

$$\theta = tan^{-1}\left(\frac{\sqrt{3}}{2\rho - 1}\right) \tag{7}$$

Strictly speaking, the above expression is only valid for $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$. However, it is easily extended to the domains $-\pi < \theta < -\frac{\pi}{2}$ and $\frac{\pi}{2} < \theta < \pi$ by noting that (for $-\pi < \theta < \pi$) θ must have the same sign as $t_3 - t_2$: if $t_3 > t_2$, then the beacon lies above the x axis and θ is positive; likewise, if $t_2 > t_3$, the beacon lies below the x axis and θ is negative. Thus, if the signs of $t_3 - t_2$ and θ differ, adding (or, equivalently, subtracting) π from θ provides the desired result. In addition, one need address the following corner cases:

- $\rho \to \frac{1}{2}$: we consider the arctan function to be defined at $\pm\infty$ as $\pm\frac{\pi}{2}$ for the purposes of this analysis, so for $\rho = \frac{1}{2}$, θ is defined to be $\frac{\pi}{2}$ when $t_3 > t_2$ and $-\frac{\pi}{2}$ when $t_2 > t_3$.

- $\rho \to \pm\infty$: in the event that $t_3 = t_2$, the beacon clearly lies on the x axis, so for $\rho \to \pm\infty$, we define θ to be 0 if $t_3 > t_1$ and π if $t_1 > t_3$.

- $t_3 = t_2 = t_1$: clearly, this equality is only possible if the source of the ping is the center of the hydrophone array[2]. We arbitrarily define θ to be 0 in this case.

Thus, the trigonometric bearing angle θ is defined fully:

$$\theta = \begin{cases} tan^{-1}\left(\frac{\sqrt{3}}{2\rho-1}\right) & \text{if } t_3 - t_1 > \frac{1}{2}(t_3 - t_2) \text{ and } t_3 - t_2 \neq 0 \\ tan^{-1}\left(\frac{\sqrt{3}}{2\rho-1}\right) + \pi & \text{if } t_3 - t_1 < \frac{1}{2}(t_3 - t_2) \text{ and } t_3 - t_2 \neq 0 \\ \frac{\pi}{2} & \text{if } t_3 = t_2 \text{ and } t_3 > t_1 \\ -\frac{\pi}{2} & \text{if } t_3 = t_2 \text{ and } t_1 > t_3 \\ 0 & \text{if } t_3 - t_1 = \frac{1}{2}(t_3 - t_2) \text{ and } t_3 \geq t_1 \\ \pi & \text{if } t_3 - t_1 = \frac{1}{2}(t_3 - t_2) \text{ and } t_1 > t_3 \end{cases} \tag{8}$$

2.2 Geometric Constraints

Determining the pairwise time differences is a nontrivial task in itself. Ideally, a measurement should be taken every period of the wave to minimize sampling error. To achieve this end, the hydrophones must be arranged such that less than half of a period of the wave may elapse between receipt at any pair of hydrophones. Otherwise, measurements would be ambiguous: unless the beginning of a burst were observed, it would be impossible to determine the period to which a measurement corresponded, so pairs of receipts could not be matched. This solution is also more robust: accurate measurements are still possible if the signal is not bursted but is instead continuous. Thus, the distance d between two hydrophones is constrained by the acoustic frequency f and speed of sound v as such:

$$d < \frac{v}{2f} \tag{9}$$

The resulting hydrophone configuration is known as an ultra-short baseline (USBL) array.

Given this constraint, it is possible to determine the pairwise time differences by observing single periods of the wave. One method for obtaining these time differences is to compare the hydrophones' output voltage waveforms to a constant threshold value. The times of positive crossings of this threshold are then recorded for each hydrophone. The condition for a successful measurement is signal receipt at all three hydrophones satisfying the following constraint:

$$max\left\{|t_2 - t_1|, |t_3 - t_1|, |t_3 - t_2|\right\} < \frac{1}{2f} \tag{10}$$

2.3 Sampling Rate Constraints

As a practical note, the accuracy of this method depends heavily on the resolution of the time measurements. Table 1 provides statistics on the angular error as a function of sample rate, assuming that the hydrophones are in a USBL configuration, separated by 2 cm. It is evident that in order to have reasonably accurate angle measurements, resolution better than 1 μs (i.e. a sample rate of more than 1 MHz) is necessary. Furthermore, this analysis is optimistic since it does not consider other sources of error; thus, the need for a high sampling rate is clear.

3 High-Level System Design

Now that we have described the signals of interest we will be provided by the hydrophones, we must design a means of converting this signal into something useful to a computer, computing a bearing angle, and

---

[2]In this case, the plane wave approximation is invalid.

Sampling Rate (Hz)	Maximum Error (degrees)	Mean Error (degrees)
200K	22.1	4.73
1M	4.89	1.12
10M	0.49	0.10
60M	0.08	0.02

Table 1: Expected error in bearing angle due to sampling rate

transmitting the computed angle to a central computer via a standard protocol. With only this constraint, the design space is quite large. We need to further constrain the design to narrow the possibilities to a workable system.

One important consideration for any embedded system is form factor. For a system to be truly useful, its physical dimensions and configuration must mesh well with its operating conditions. This is particularly true in autonomous underwater vehicles, where all electronics are typically housed in a water-tight tube, whose price and vehicle performance cost (in terms of drag) increase dramatically with size. This means that APLS should be small, and should integrate easily with the rest of the electronics in the AUV in a physical sense. One computing form factor that has become extremely popular in the underwater space is PC/104. This specification, itself an extension of IEEE-P996, specifies a bus which connects multiple cards which, due to the common bus specification, can easily be stacked on top of one another (thus coining the term "PC/104 stack"). In addition to the bus, each card in the stack shares common mechanical dimensions for easy stacking, and each card has 4 mounting holes at predefined positions in order to enable common standoffs to be inserted between each pair of cards, to give the stack mechanical rigidity. In order to be physically compatible with a wide range of autonomous underwater vehicle, we decided to house our entire system on a single PC/104-size card. However, since the data communication between APLS and the main computer will be minimal and mostly unidirectional, the large, high-pin-count PC/104 connectors would be overkill, and would occupy a large percentage of the already small (under 15 square inches) available board area. Therefore, we chose to adopt the PC/104 form factor, without using the PC/104 bus. Therefore, APLS can be connected to *any* existing PC/104 stack without complicating such considerations as stack position and capacitive bus loading, and all data communication can take place over a commodity cable, rather than the PC/104 bus. Our system will be implemented on a custom-designed printed circuit board (*PCB*).

It became apparent early in the process that our application would require a fair amount of specialized logic to capture the difference in arrival times between all 3 hydrophone signals. After an extensive survey of available embedded processors, none were found that provided the flexibility and customizability we desired. In order to preserve the ability to modify this logic without the need for a PCB redesign, it became clear that a field-programmable gate array (*FPGA*) or complex programmable logic device (*CPLD*) would be an ideal solution. The high dynamic power consumption of these devices was a concern, but a survey of popular FPGA and CPLD families (from industry leaders such as Xilinx, Altera, Lattice Semiconductor, and Atmel) found that although FPGAs consume substantially more power than their ASIC equivalents, when available, even this elevated power consumption is well under 1 Watt, which means that either solution would pale in comparison to some of the other power requirements on an AUV (thrusters, CPU, high-power solenoids and relays, resistive losses in DC/DC converters). At this point, the two alternatives we considered were a completely FPGA-based design, and a CPLD (loaded with timing information capture logic) with a microcontroller handling the processing and communication. At first, we considering designing the FPGA or CPLD/microcontroller combination into our board, but we quickly found that the extremely fine pin/ball pitch and large pin/ball count of these devices would pose major problems in the assembly of our board. Specifically, any FPGA with a gate count sufficient to encompass timing capture, processing, and communication has hundreds of pins and hundreds of pages of PCB design constraints, and attempting to design a board with an integrated FPGA or complex, high-pin-count microcontroller, much less attempting to solder on such a component with the limited toolset available to us, would provide little value to the finished system, while drastically increasing the complexity of our design. We next shifted our focus to finding a

commercially-available PCB with an integrated programmable logic solution, which we could somehow connect to our board, which would ostensibly contain all necessary support logic and analog processing hardware.

One potential complication of an FPGA-only design is that designing a relatively general-purpose processing solution using the traditional design entry modes available to FPGA designers (HDL and schematics) is an extremely cumbersome process, particularly given the time constraint present for this project. One solution to this problem is to use a publicly-available *soft core*, and embed it within the FPGA. A survey of such cores yielded a wide range of results. Websites dedicated to open hardware design, such as *opencores.org*, had a variety of cores available, ranging from the 8051 to the 8080, from a simple VLIW superscalar processor to MIPS to JOP, a core optimized to run (stack-based) Java bytecode. These cores were completed and documented to wildly varying extents, and in the end, the total package, including correct, commented RTL, clear documentation, and software development toolsets were lacking for these openly available cores. Given the severe time constraint, we needed to find a more robust, commercially-supported soft core to include in an FPGA, if this was to be a viable option. The two major FPGA vendors, Altera and Xilinx, each offer something along these lines. Their respective solutions are known as NIOS/NIOS II and Picoblaze/MicroBlaze, respectively. The MicroBlaze was the most attractive of these, due to the extensive documentation, a fully ported GCC-based C compiler/assembler/linker toolset, and an IDE integrated with the main synthesis tool (ISE). The NIOS II had many of these same features, but it was found that the majority of commercially-available small-form-factor FPGA boards were Xilinx-based. In the end, we decided to search for a Xilinx Spartan-3-based board. The Spartan-3 FPGA family provided a good compromise of gate count (up to 1 million equivalent ASIC gates available), cost, and power consumption, with the ability to contain a full MicroBlaze processor (including hardware FPU) plus a considerable amount of custom logic.

Small Spartan-3-based boards were available from several vendors, including Xilinx (*www.xilinx.com*), Opal Kelly (*www.opalkelly.com*), and XESS (*www.xess.com*). The main distinguishing characteristics of these boards were available documentation, on-board logic and interfaces, cost, and ease of interface with a second board. All vendors had adequate documentation, but they varied in terms of how clearly they described the signals entering and leaving the board, and the requirements thereof (voltage, timing, etc.). The boards varied a great deal in complexity; the majority had USB interfaces via an external controller, whereas some had full 100BaseT Ethernet MACs and PHYs. The boards ranged in cost from approximately 100USD to over 400USD. Additionally, some boards appeared to be relatively painless to interface with a second board, with large connectors on the underside of the board, hardwired to FPGA and other pins. Others had no such conveniences, meaning that information would have to be passed between our analog processing board and the FPGA board via a cable of some sort. In the end, the latter solution seemed much less robust, and certainly less elegant. In the end, we decided on a board from Trenz Electronic (*Trenz Electronic*), the TE0140-01. This board satisfied all the design requirements previously specified. In addition to being reasonably priced (99 Euros $\approx$ 130 USD at the time of purchase), the board has a UTMI interface to a USB 2.0 (high-speed) interface, two 80-pin connectors to connect to a baseboard, and adequate documentation, including schematics. This board was available in a $400,000$-gate version and a $1,000,000$-gate version. We chose the former, deciding that the increased cost of the larger FPGA did not add significant value to the end system, given our system-level requirements. Additionally, the power consumption of the larger FPGA would be much larger.

The end result of this process was the decision to design a PC/104 form factor PCB containing analog processing and support logic, and board-to-board connectors to connect to a Trenz Electronic TE0140-01, which would house our processing solution, a Xilinx XC3S400FT256-4. With this board, we have the option of communicating via USB via the built-in USB controller and PHY, although another protocol may prove more attractive due to lower complexity and low bandwidth requirements.

4 Component Selection

4.1 Signal Conditioning

4.1.1 High-Level Topology

Once the system topology, form factor, and processing solution had been decided upon, the focus shifted to the analog elements of our design. Specifically, we needed to design a means of converting the signal from the hydrophones, a high-output-impedance analog waveform, into something useful for the FPGA. There were two competing strategies for this part of the design. The first alternative was to buffer and amplify the signals from the hydrophones, and simply pass them into a high-speed, high-precision analog-to-digital converter (ADC). The FPGA could obtain samples from the ADC and perform digital signal processing (DSP) on them at its leisure. The second strategy was to replace the ADC with a thresholding mechanism, which would essentially transform the hydrophone signal into a one-bit digital signal, representing whether or not the hydrophone signal exceeded some reference voltage. The DSP solution provided slightly more flexibility: more information would be captured by the FPGA and could thus be used in more ways than a single-bit signal. However, the second analog paradigm was much simpler from a software point of view, assuming we could design a properly-functioning amplification, filtering, and digitization solution. We were confident that our algorithm could match the performance of existing DSP-based solutions, most of which make use of the Fast Fourier Transform (FFT) to calculate the phase difference between received signals, with a much lower hardware cost and software complexity.

4.1.2 Amplifier

The next step in the design process was to decide on a specific sequence of components for amplification, filtering, and digitization. For amplification, a low-noise, low-input-current operation amplifier was desirable. The component that best met these requirements was the AD743, from Analog Devices (*www.analog.com*). This part offers excellent noise performance (2.9 nV/vHz at 10kHz), a low input offset voltage (maximum 500μV), and a flat frequency response in our region of interest, 20 kHz to 35 kHz (4.5 MHz unity-gain bandwidth). The part was available in an 8-pin DIP package, as well as a 16-pin SOIC package. The surface-mount SOIC part was actually not much smaller than the DIP part, and as we will show, the DIP package was advantageous to us for two reasons. First, this package is directly compatible with solderless breadboard, offering easy prototyping and testing. Also, in a PCB design, DIP components can be useful if signals need to cross layers, since each pin requires a via anyway. For these reasons, we chose the DIP-8 package (Part Number AD743JN). The pinout for this package is shown in Figure 2.
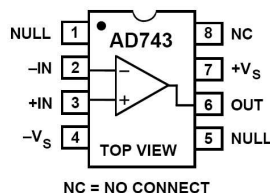
Figure 2: Analog Devices AD743 DIP-8 Pinout

After selecting the operational amplifier itself, the surrounding passive circuitry was designed, tested, and iterated to an amicable solution. At first, we used the AD743 op-amp in a simple non-inverting configuration. The hydrophone was connected to the non-inverting node of the op-amp. Since the hydrophone is modeled as a capacitor, it acts as an open circuit when it is not receiving a signal; so, to prevent drift, the non-inverting node was pulled down to ground with a resistor large enough not to affect the hydrophone's response but small compared to the op-amp's input impedance. A 680 KΩ resistor was chosen for this purpose. Initially, a 100 Ω resistor grounded the inverting node, and a 10 KΩ potentiometer connected the

output of the op-amp to its inverting input to provide an adjustable gain of up to 101. During testing, the need for a larger gain arose, so the 100 Ω resistor was reduced to 50 Ω.

It became apparent that because of the rather large gain needed for an output amplitude on the order of volts, non-ideal op-amp characteristics resulted in a large DC output component. This effect was unacceptable because the asymmetry in the output waveform greatly reduced the gain at which clipping began to occur. To address this issue, adjustments were first made to the pull-down resistor to limit the effects of input bias current, but it quickly became evident this was not the problem: even setting the resistor to the parallel combination of the gain resistors, which effectively eliminates input bias current, did not have a significant effect on the DC component of the output voltage. In fact, the input offset voltage caused this artifact: the nominal value of 1 mV becomes nontrivial when amplified by several hundred. To eliminate the input offset voltage, we adjusted the potential at the inverting node until the DC output voltage was zero. To do so without affecting the negative feedback configuration of the op-amp, a potentiometer connected between the 5 V rail and ground was used as an adjustable voltage source, and the wiper was connected to the inverting node via a large (2 MΩ) resistor. This proved to be an effective field-configurable solution to the input offset voltage problem, and it did not have any appreciable effect on the gain.

Finally, we observed that our early choices for gain resistors of 50 Ω and a 10 KΩ potentiometer resulted in a much noisier output than larger resistors with similar ratios. We replaced them with a 20 KΩ resistor and a 5 MΩ potentiometer, which reduced the noise dramatically and provided more headroom in case a larger gain was desired at some point.

The final configuration of the AD743 is shown in Figure 3.
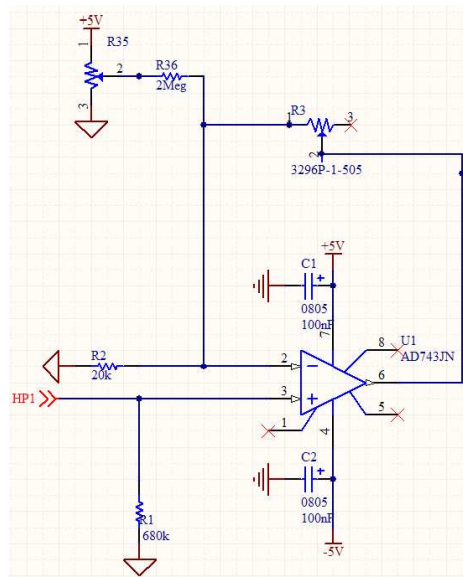
Figure 3: AD743 Final Configuration

4.1.3   Filter

The next component in the signal processing chain was the filter. The underwater environment is a very noisy one, and without filtering, the acoustic pinger signal could potentially be dwarfed by, or at least mired in, external noise sources, such as ship turbines, animal communication, and pingers at other frequencies. In order to filter out energy at unwanted frequencies, we chose to use a bandpass filter centered around the pinger's frequency, with a bandwidth of 1 kHz. This bandwidth value was chosen to make sure that the

pinger's frequency would be included in the passband, even if its frequency is only specified to a precision of ±500 Hz, as many commercial pingers are. One significant challenge posed by the design criteria is that, in order to be able to track pingers at any frequency, the center frequency of the bandpass filter must be variable across the range 20-35 kHz. This is where conventional RLC and active filters fail; in order to change the center frequency, many passives would have to be changed, which means a great deal of desoldering and resoldering, which is unacceptable not only from a convenience standpoint, but also for the longevity of the PCB. After some thought, a solution that came to mind was a switched-capacitor filter. Though somewhat rare in many applications, this type of filter satisfies the requirement of variable frequency. Instead of using hardwired passive components to perform the filtering, switched-capacitor filters use an input clock signal to charge and discharge external capacitors, and, in conjunction with external resistors, the input clock frequency determines the filter type (lowpass, bandpass, highpass, notch, allpass), gain, quality factor (Q), and center frequency. There are relatively few switched-capacitor filters available in discrete form. The most suitable one for our application was the LMF100, manufactured by National Semiconductor. Each LMF100 contains two filters, each of which can implement a second-order filter of any of the five types mentioned above. Indeed, one can cascade both filter elements to form a single fourth-order filter. This part is available in a 20-pin DIP package, as well as a 20-pin Wide SOIC package. We chose to purchase three DIP package LMF100s for testing purposes, but decided that the footprint was too large for our area-constrained PCB, so we chose to use the 20-WSOIC (Part Number LMF100CIWM) package in the final implementation. The pinout of this part is shown in Figure 4.
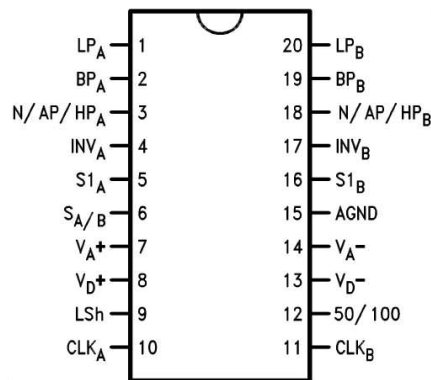

Figure 4: LMF100 20-WSOIC Pinout

An additional design constraint was that each hydrophone processing channel should be placed and routed on the PCB as uniformly as possible to ensure a uniform response. To satisfy this requirement, we chose to use one LMF100 for each of our 3 channels, rather than use 2 LMF100s, one with 2 channels and the other with one. This meant that we could use the aforementioned cascading of second-order filters to implement a fourth-order filter with a much sharper cutoff, yielding a potentially cleaner output signal through stronger rejection of unwanted energy.

The LMF100 offers several modes of operation, set by tying certain control pins in configurations specified in the manufacturer's datasheet. The simplest mode (that with the fewest external components) which offered the ability to affect the center frequency by modifying resistor values was Mode 3. The basic schematic for Mode 3 filters is shown in Figure 5.

The center frequency's dependence on external resistor values was important due to the method by which one synthesizes a fourth-order bandpass filter from second-order filters. The basic idea is to cascade two filters with center frequencies slightly above and slightly below the desired fourth-order center frequency, with slightly higher passband gains and higher Q. Multiplying their frequency responses (equivalent to convolution in the time domain, which is equivalent to cascading the filters) yields the desired fourth-order
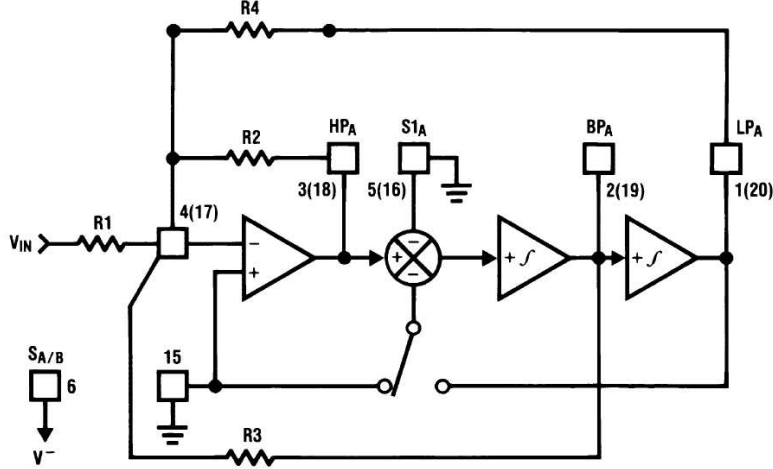
12

Figure 5: LMF100 Mode 3

bandpass response[3]. With the ability to shift the center frequency upward and downward with resistors, we could use the same input clock for both second-order filter elements, an important cost-saving feature. The equations used to find the particular f_o, H_{OBP} (bandpass gain), and Q factors for the individual filters were found in a document distributed by Texas Instruments, Chapter 16 of a book entitled *Op Amps For Everyone* (*focus.ti.com/lit/ml/sloa088/sloa088.pdf*). From this document, the following equations were used to find the three filter attributes listed above. Note that in the equations below, A_{mi} is the midband gain of each individual filter, Q_i is the quality factor of each filter, and α is the factor by which the desired center frequency should be multiplied and divided to obtain the individual center frequencies.

$$\alpha^2 + \left[\frac{\alpha \cdot \Delta\Omega \cdot a_1}{b_1 \left(1 + \alpha^2\right)} \right]^2 + \frac{1}{\alpha^2} - 2 - \frac{(\Delta\Omega)^2}{b_1} = 0 \tag{11}$$

$$Q_i = Q \cdot \frac{\left(1 + \alpha^2\right) b_1}{\alpha \cdot a_1} \tag{12}$$

$$A_{mi} = \frac{Q_i}{Q} \cdot \sqrt{\frac{A_m}{b_1}} \tag{13}$$

Note also that, in the above equations, a_1 and b_1 refer to filter coefficients, specified by the filter type desired. Of the major filter classes (Bessel, Butterworth, and Chebyshev), the Butterworth filter was the most desirable, due to its middle-of-the-road frequency response (it is commonly described as a compromise between Bessel and Chebyshev). The filter coefficients corresponding to this filter type are $\sqrt{2}$ and 1, respectively. The variable $\Delta\Omega$ corresponds to the inverse of the Q factor of the desired fourth-order filter. Our minimum Q value would be at 20 kHz, which yields $\frac{20\text{kHz}}{1\text{kHz}}$, or 20. This gave $\Delta\Omega = 0.05$, which yielded an α value of 1.01752. This, in turn, gave Q_i and A_{mi} values of 28.2885 and $\sqrt{2}$, respectively. This is due to our desire to have the filter by unity-gain at passband. The rationale behind this is that the amplifier is much better suited for low-noise amplification than the filter. With these filter parameters in hand, we proceeded to choose Mode 3 external resistor values for the LMF100.

The LMF100 datasheet specifies that, for Mode 3, the following equations hold:

---

[3]Since the convolution is commutative, the order of the filters does not matter

$$f_o = \frac{f_{CLK}}{\{50, 100\}} \times \sqrt{\frac{R_2}{R_4}} \qquad (14)$$

$$Q = \sqrt{\frac{R_2}{R_4}} \times \frac{R_3}{R_2} \qquad (15)$$

$$H_{OBP} = -\frac{R_3}{R_1} \qquad (16)$$

$$(17)$$

A reasonable value for R_3 is 100kΩ. This yields approximate values for R_1, R_2, and R_4 of 71.5kΩ, 3.48kΩ, and 3.60kΩ, respectively. Note that R_1 and R_3 will have the same value for both second-order elements, whereas R_2 and R_4 will be swapped between the two. This elegant design is a byproduct of selecting the appropriate LMF100 mode. These resistor values are summarized in Table 2.

Resistor	Filter Element 1 Value (Ω)	Filter Element 2 Value (Ω)
R_1	100k	100k
R_2	3.48k	3.6k
R_3	71.5k	71.5k
R_4	3.6k	3.48k

Table 2: LMF100 Resistor Values

Note that these resistors are the result of several other design decisions related to the LMF100. First, we decided to power the LMF100 off +5V and -5V, since precisely offsetting the input signal to work on a single-ended power supply would be difficult, and could potentially increase noise by decreasing the amplitude of the signal. Also, the LMF100 can operate in $\frac{f_{CLK}}{50}$ or $\frac{f_{CLK}}{100}$ mode. The latter mode offers slightly better performance in many situations, since the minimum aliasable frequency is doubled with respect to the former. However, the latter mode posed a problem for us in terms of f_{CLK} adjustability. We intended to provide the LMF100 clock signal from the FPGA, not from a dedicated clocking module or PLL, so it was only possible to provide a limited set of frequencies (relatively even fractions of our input clock frequency, which on the TE0140-01 is 60 MHz). For the center frequency range 20 kHz to 35 kHz, $\frac{f_{CLK}}{100}$ mode would require an input clock between 2 MHz and 3.5 MHz, which gives us substantially fewer possible synthesizable frequencies than $\frac{f_{CLK}}{50}$ mode. Additionally, we predicted that energy in the $50 \cdot f_{CLK}$ to $100 \cdot f_{CLK}$ range would be quite low, making the increased aliasing a non-issue. The final LMF100 configuration used is shown in Figure 6. Note that the resistor numbers are slightly different from those previously discussed, that the analog and digital voltage rails are tied together (since our design is relatively devoid of large digital chips which would dirty an analog power supply), and that this configuration includes several decoupling capacitors, the rationale behind which will be discussed later in this paper.

4.1.4 Digitization and Voltage Translation

After the hydrophone signal has been amplified and filtered, it needs to be digitized and sent to the FPGA. Our proposed digitization scheme consists of an analog comparator with an arbitrary threshold voltage provided by a potentiometer, connected between our power and ground rails. The requirements for this analog comparator are as follows:

1. *It must be fast.* Every analog comparator has a propagation delay associated with it, on the order of ns or μs. Any variation in this propagation delay will manifest itself as phase error in the digitized signals, so we must keep the propagation time itself as small as possible, and by extension, limit its variation across the 3 comparators necessary to digitize all 3 hydrophone signals.
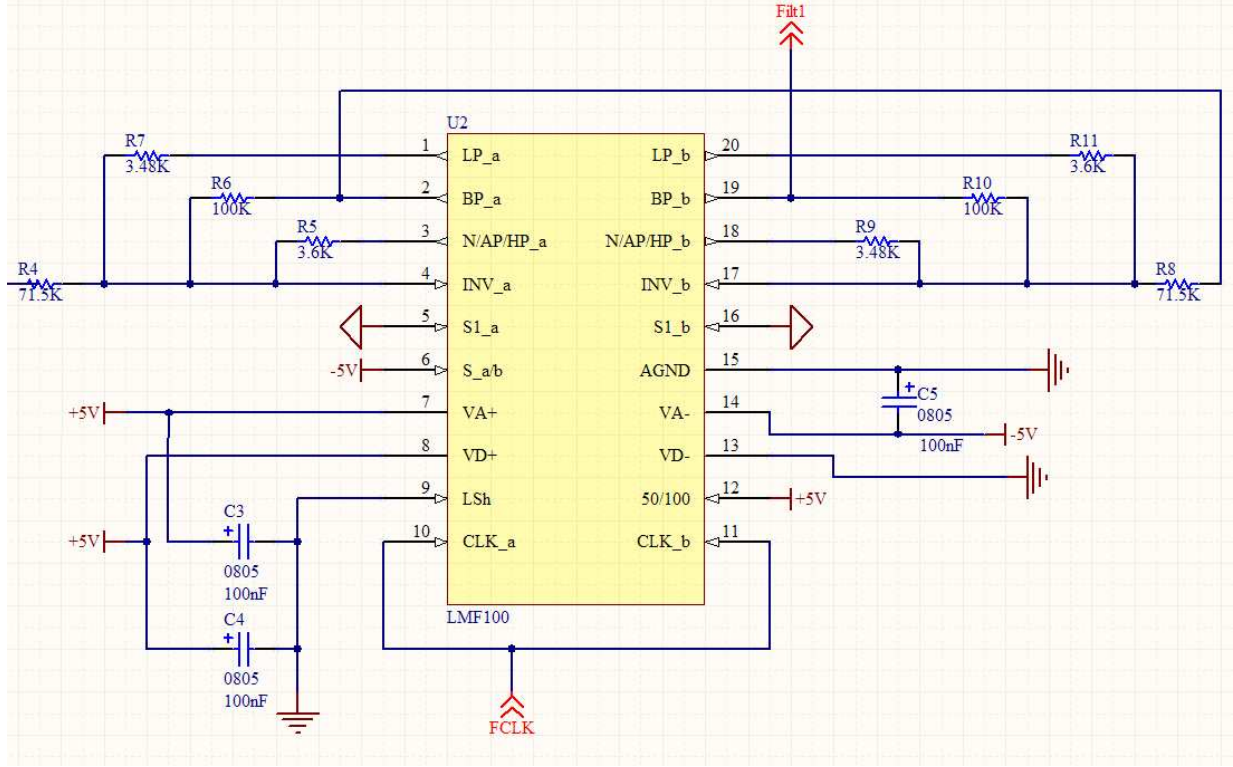
Figure 6: LMF100 Final Configuration

2. *It must have a dual-supply front end (input) and single-ended output.* The signal coming from the LMF100 varies between +5V and -5V, and the digital signal to be produced must vary between 0V and a positive voltage (the FPGA supports various voltages from 1.2V to 3.3V). It turns out that few analog comparators satisfy this requirement.

After an extensive search, a comparator was found that satisfied these requirements, as well as being a quad-comparator part, which means that only one IC would be needed to digitize all 3 hydrophone signals. This not only reduces the bill of materials (*BOM*) of the PCB, it also drastically reduces propagation time variation. The part is also from Analog Devices, the CMP401. This part comes in 16-pin Narrow SOIC and Thin Shrink Small-Outline Package (TSSOP). We chose the SOIC version for ease of soldering and greater availability. The full part number is CMP401GS. The pinout of the CMP401GS is shown in Figure 7.

One difficulty encountered in designing the CMP401GS into our system is that, although the front-end and output stages have separate voltage rails, the analog positive supply should not exceed the digital positive supply voltage by more than 200mV. This means that a separate IC would have to translate between the 5V output of the CMP401GS and the FPGA, which only supports I/O standards up to 3.3V. Applying a 5V input would irreparably damage the FPGA. The process of selecting the voltage translator was dramatically simpler than for the rest of the components. All that was required was a quad 3.3V-powered buffer with 5V-tolerant inputs. Such a part was quickly found in Fairchild Semiconductor's 74LCX125. This part comes in a 14-pin Narrow SOIC package (full part number 74LCX125M). The pinout of the 74LCX125M is shown in Figure 8.

An additional advantage of the 74LCX125M is that it was a *quad* buffer, which meant that it could translate an additional signal besides the 3 hydrophones, which would turn out to be useful later in the hardware design.
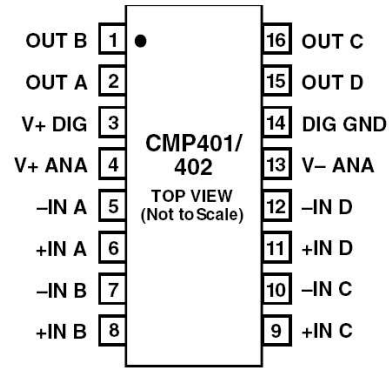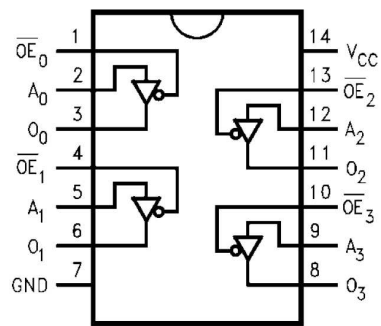
15

Figure 7: CMP401GS Pinout

Figure 8: 74LCX125M Pinout

16

4.2   Miscellaneous Components

4.2.1   Serial Communication

The FPGA board selected, the TE0140-01, already contains JTAG and USB interfaces. However, we thought it useful to include a simpler (than UTMI), more standard (than JTAG) debug interface for communicating with our MicroBlaze processor. One such protocol is RS-232, commonly known as "serial". This protocol is already implemented in a UART module provided by Xilinx as part of the Embedded Development Kit (*EDK*) software package. Furthermore, RS-232 is supported as one of the debug *stdin* and *stdout* ports of the MicroBlaze processor, making RS-232 a valuable system debug tool. However, the FPGA's input and output voltages were obviously not suitable for the standard, which specifies typical voltages of +12V and -12V. To translate between the 3.3V FPGA I/O's and a standard RS-232 line, a level shifter was needed. Luckily, these are readily available. We decided on a single transceiver (Rx/Tx signals only) from Texas Instruments (*www.ti.com*), the MAX3221CDBR. This level shifter required several external capacitors (for decoupling and a charge pump), but they were all of a single, standard value (100nF), so the chip, which comes in a 16-pin SSOP package, was relatively easy to integrate into our design. We decided to power the chip off +5V and GND. The input on the level shifter has a logic threshold suitable for interpreting the FPGA's 3.3V Tx signal, but some sort of interface was required for the Rx line, since the level shifter was powered off 5V. Luckily, as alluded to earlier, the 74LCX125M's fourth input served this purpose well. The pinout for the MAX3221CDBR is shown in Figure 9.
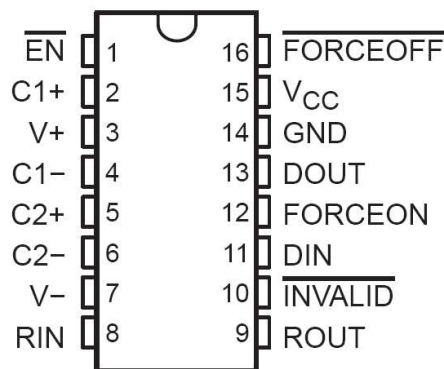

Figure 9: MAX3221CDBR Pinout

4.2.2   Indicator LEDs

Throughout the course of this design, it became apparent that an easy visual means of quickly verifying the correct operation of the analog processing circuit would be desirable, since oftentimes autonomous underwater vehicles are used in environment where laboratory instrumentation is sparse. A simple means of verifying the entire circuit is to connect a light-emitting diode (LED) to each of the 3 digitized hydrophone signals. If the circuit is operating correctly, the LED should light up each time the signal goes high (logic '1'). This should only occur during pinger pulses, which typically happen at predefined intervals on the order of seconds.

In order to drive these LEDs, an inverter was needed that could sink approximately 10mA of current per output. The inverter selected for this application was the Texas Instruments SN74LVC3G04DCTR. This triple inverter is housed in an 8-pin SSOP package, which is extremely small, and thus relatively easy to fit in our design (though difficult to solder). The pinout of the inverter is shown in Figure 10.

For the LEDs themselves, three different colors were desired, in order to be able to easily differentiate which of the 3 channels is triggering, even in an underwater environment. In order to maximize the view-
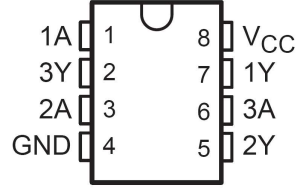
Figure 10: SN74LVC3G04DCTR Pinout

ing angle, diffused LEDs were preferred over clear ones. Also, in addition to operating within our voltage constraints, with reasonable power, the LEDs should integrate the current-limiting resistor typically needed in these situations, to reduce the BOM of the PCB and reduce the footprint of the LEDs. A line of LEDs from CML was chosen for the task (part numbers 5602F1-5V (Red), 5602F3-5V (Yellow), and 5602F5-5V (Green)). These LEDs integrate an appropriate current-limiting resistor, and simply have two pins, power and ground.

The final configuration of comparator, voltage translator, inverter, and LEDs is shown in Figure 11.
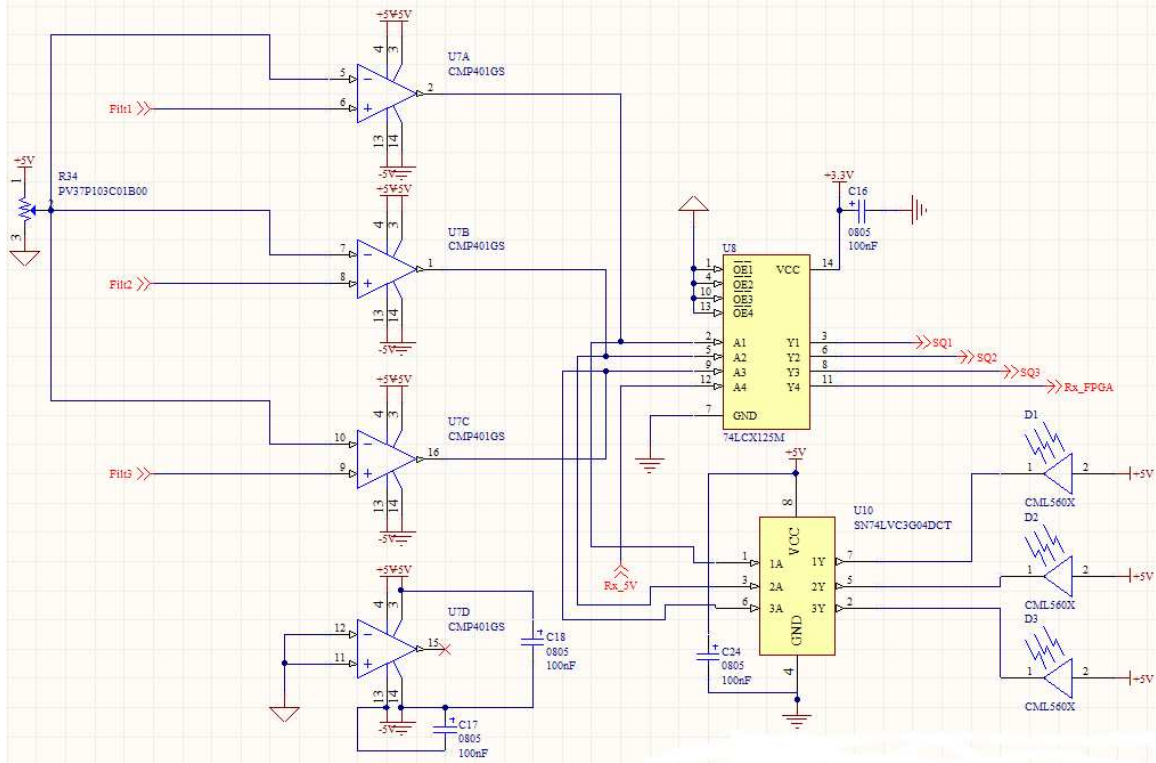

Figure 11: Digitization And Debug Final Configuration

4.2.3 Connectors

Our PCB contains four classes of signals that must be connected to the outside world:

1. *Power*— Here, we have +5V, -5V, and GND.

2. *Hydrophones*— There are 3 hydrophone signals, each with their own ground (although they are all tied to system ground on the PCB).

3. *Serial*— We need at least GND, Rx, and Tx. No flow control will be used. There may be multiple grounds in order to easily interface to a standard DB-9 connector.

4. *Program*— The FPGA can be reprogrammed by bringing a PROGRAM signal low temporarily. This is a good way to reset the FPGA without power-cycling it (it can be done electronically). This signal should be brought out to a connector, along with a GND signal, so that it can be brought low, either by shorting the two signals or by applying 0V to the signal with respect to GND (such as with a digital acquisition card).

A logical partitioning of these signals broke these 4 classes up into three connectors. One connector with large pins was needed for power, one 6-pin connector for hydrophone signals, and a nominally 10-pin connector for serial and program signals. The latter 2 connectors should be relatively standard, so as to facilitate replacement and procurement of new female connectors. The first connector chosen was a Molex 39-30-3045 4-pin right-angle power connector. Two pins were used for GND, one for +5V, and one for -5V. The second connector was a 6-pin right-angle .1" header (Tyco 102617-1). The final connector was a 10-pin right-angle .1" header (Tyco 102617-3). The latter two connectors are two-row connectors, for decreased width (resulting in a more convenient PCB footprint). Additionally, both header connectors were shrouded for added protection from inadvertent short-circuits.

4.2.4 Decoupling Capacitors

In order to provide a clean voltage rail to each component on the board, a two-level decoupling network was used. The first level consisted of bulk capacitors for the +5V and -5V rails. Two 100μF radial capacitors (Nichicon UVZ1E101MED) were chosen for this purpose. Their voltage rating (25V) and size (5.5mm diameter) were well-suited to our design. These capacitors will provide moderately-sized surges of current, should they be required by any of our on-board integrated circuits. Also, for each rail of each component, we place a 100nF decoupling capacitor between the power pin and GND. While this increases the component count of the board significantly, it will ensure proper operation of the sensitive analog circuitry by preventing droop on the power rails when current demands change, as they inevitably do. These decoupling caps also "decouple" the components from the system by absorbing transient spikes in the voltage rail, providing a low-impedance path to ground for high-frequency signals and preventing component damage. The decouplers chosen for the system were Panasonic 25V 0805 form factor capacitors (Panasonic ECJ-2VF1E104Z). The 0805 form factor was chosen as a compromise between solderability and size. Any smaller, and we may not be able to ensure adequate solder joints with the limited equipment available to us. Any bigger, and the capacitors may dominate the board area[4].

In accordance with the manufacturer's recommendations, each power pin on the FPGA board will be decoupled on our baseboard with a separate 100nF.

4.2.5 Resistor Selection

The 0805 form factor was chosen for all onboard resistors. Most resistors only require 1% precision. However, the resistors that determine the center frequency of the LMF100 filter elements (previously referred to as R_2 and R_4) are very important. Any mismatch in these center frequencies will manifest itself as phase error between the processed hydrophone signals, due to the high first derivative of the phase response of a bandpass filter. Therefore, these resistors were purchased with 0.1% precision. Based on availibility, various resistor values from Yageo and Panasonic were purchased.

---

[4]As a consequence of choosing the 0805 form factor for the capacitors, we also chose to purchase 0805 resistors, for aesthetic reasons.

4.2.6   Potentiometer Selection

The design criteria for the potentiometers is that they should be high-turn, side-adjustment, and through-hole mount (for mechanical stability). The 10kΩ potentiometers selected were Murata PV37P103C01B00s. These are 250mW 12-turn pots. The 5MΩ pots chosen were the Bourns 3296P-1-505. These are 100mW 25-turn pots.

4.2.7   Prototyping Aids

In order to prototype easily with small, surface-mount components on a breadboard, SOIC-to-DIP adapters were needed for the CMP401 and 74LCX125. Such prototyping adapters were readily available in the necessary 14-pin and 16-pin sizes from Aries Electronics (part numbers 14-350000-10 and 16-350000-10, respectively).

5   PCB Design

The complete schematics can be found in Section 12.

5.1   High-Level Decisions

There were several decisions to be made before physical PCB layout could begin. First, the manufacturer and process were chosen. A popular manufacturer of prototype PCBs is ExpressPCB (*www.expresspcb.com*). In addition to a 1-layer or 2-layer process, they also provide free layout software. Unfortunately, it became clear that we required a more advanced process if we were to successfully solder so many fine-pitch components, particularly the 1.9mm-pitch, 80-pin board-to-board connectors. We required a manufacturer who would be able to provide boards with solder mask, which prevents solder bridges between pins of fine-pitch components. Also, since there were so many components placed throughout the board, and many decoupling capacitors which require connections to GND. To make routing all the components possible in such a small area without massive amounts of jumper wires, a 4-layer board was decided upon. The inner two layers would be used for GND and +5V, with the outermost layers used for signal routing. One manufacturer that could provide these boards in a relatively short amount of time, for relatively low cost, was ProtoExpress (*www.protoexpress.com*). Once they had been decided upon as a manufacturer, we had to find a PCB layout tool that could output Gerber files, the industry-standard format used by many manufacturers to transmit pad, solder mask, drill, board outline, and other data.

One very helpful feature for a PCB layout tool is an integrated schematic capture tool. This allows the designer to design the schematics and physical board within a single environment, and pass information back and forth between the two views. For example, the layout tool can ensure that *all* connections specified in the schematic have been physically made on the boards. All components instantiated in the schematic can be associated with footprints in the layout tool. There are only a very few tools with these capabilities, including Eagle Layout Package, OrCAD, and ProTel. In the end, due to availability, a large collection of library schematic components and footprints, ProTel was chosen. Once a ProTel license was obtained, libraries had to be made for all components which weren't included with the software. This essentially amounted to nearly all components used in the board. This was a very time-consuming process, since all mechanical dimensions of all parts had to be obtained, checked, and meticulously rechecked from manufacturer's datasheets. However, the process yielded an incredibly useful library of information; once the schematic capture tool and the layout tool were working in concert, the PCB design and design checking process was *greatly* simplified.

5.2   Stackup

As mentioned previously, the typical 4-layer arrangement of two outer signal layers and internal power and ground planes was used. Most of the literature consulted on the subject suggested that the ground plane should be placed closer to the top, to reduce the inductance of power trace ground return paths.

5.3   Component Placement

   To provide a reference, the side containing the FPGA board-to-board connectors was chosen as the "top". In order to avoid violating height constraints, and maintain proper seating of the FPGA board, only passives (SMT resistors and capacitors) were placed between the two connectors on the top. Most of the integrated circuits were placed on the underside of the board. The 6-pin, 10-pin, and power connectors were placed along the edges of the board for obvious accessibility reasons. The six amplifier adjustment potentiometers were placed according to channel along one side of the board, and the threshold adjustment potentiometer was placed on the opposite side. The decoupling capacitors were placed on either side of the power connector. The AD743 amplifiers were placed in a row on the top of the board, with the leads extending through to the other side. These built-in vias allowed us to minimize the via count on the board while passing the hydrophone signals through to the bottom side of the board, where most of the signal conditioning would take place. The LMF100s are placed in a row next to the AD743s (on the underside), and the other four ICs (analog comparator, voltage translator, RS-232 level shifter, triple inverter) were grouped on the underside as well, on the side opposite the AD743s. The LEDs were placed on the top side, directly on top of the inverter, next to the threshold potentiometer. Also note that the FPGA board was positioned so that the USB connector extended nearly to the edge of the board, for easy connection. Views of the top and bottom of the board are shown in Figures 12 and 13, and a rough 3D view of the board is shown in Figure 14.
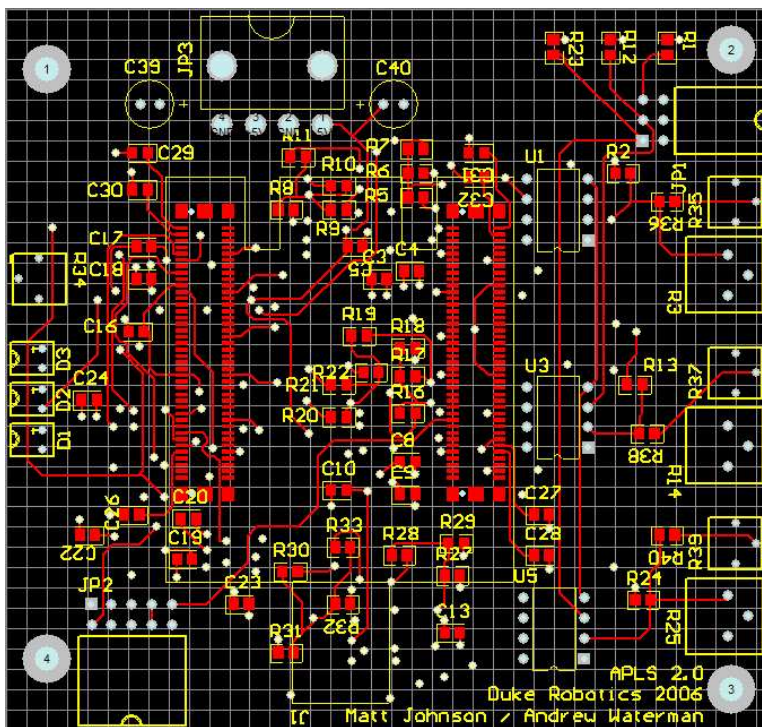

Figure 12: PCB Top Layer

5.4   Routing

   With all the signals and ICs present on this board, the routing was an interesting challenge. ProTel includes an autorouter which can attempt to automatically route signals specified in a schematic based on user-specified design rules. These design rules include routing topology (daisy-chain, shortest, horizontal, vertical, etc.), trace-and-space requirements and preferences, high-priority nets, clearance from the edge of the board, overall routing strategy (net length minimization, via minimization, hybrid), and so forth.
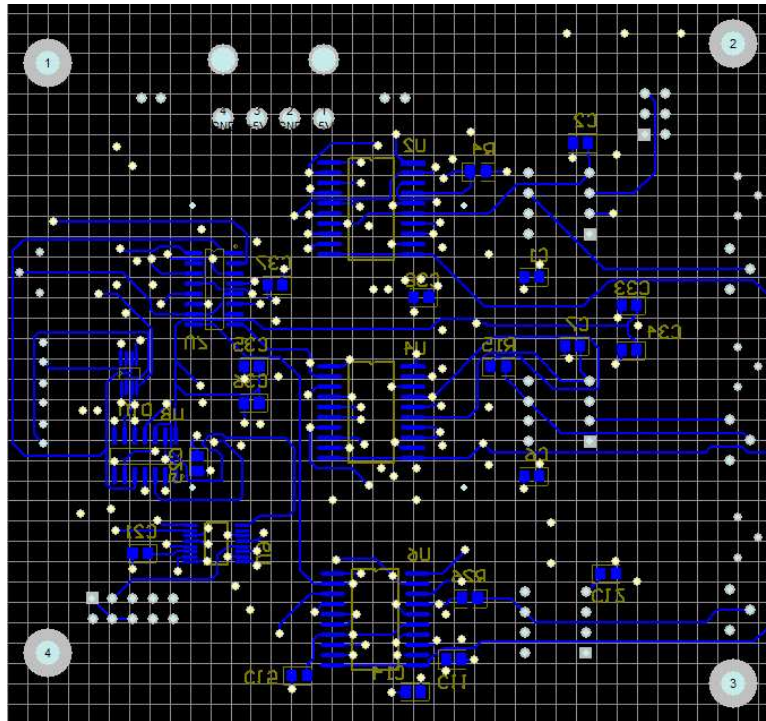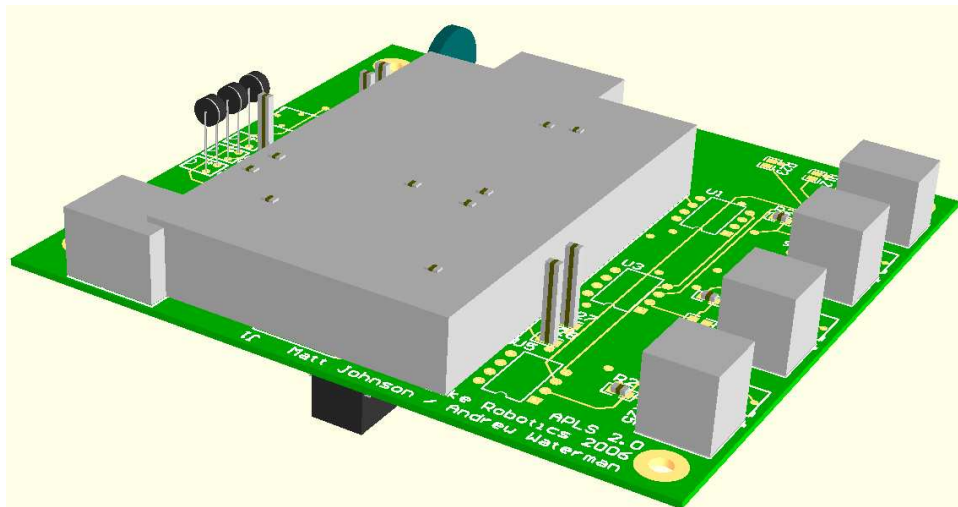
Figure 13: PCB Bottom Layer

Figure 14: PCB 3D View

Though the auto-router provided a decent starting point and proved to be a valuable time-saver on the more mundane nets, computerized routers still cannot match the results of a skilled human. In order to improve the aesthetic appearance and performance (minimize parasitics and crosstalk) of the board, the majority of the nets were re-routed by hand. An important rule in PCB routing is to avoid right angles and stubs whenever possible. Both of these drastically impact signal integrity (*SI*) by causing discontinuities in the impedance of the trace. Instead, most PCB designs today use 45-degree mitered corners to compromise between right angles and ideal, rounded corners. Also, decoupling capacitors were given their own ground vias whenever possible. It is widely known that the inductance of a capacitor's route to ground determines how effective it is going to be as a decoupler, and this inductance is greatly decreased when the via is not shared, and when it is placed as close to the capacitor as possible. The trace-and-space requirements of the ProtoExpress process were both 6 mils, but the constraint imposed on ProTel was 8 mils, in order to maintain a safe margin for error, and prevent manufacturability issues.

Additionally, thermal relief was used on all pad vias. This means that a wagon-wheel shape is formed around each via as it passes through the power or ground plane. This is important because, without this, the entire plane will act as a large thermal resistance, making soldering through-hole components extremely difficult. An example of a thermal-relief structure is shown in Figure 15. These structures can be automatically created by ProTel. Another feature offered by ProTel is automatic teardrops. Teardrops spread out the connection between tracks and vias or pads, theoretically creating a more stable, lower-impedance connection. However, both for aesthetic reasons and to maintain maximum spacing between tracks, this feature was not used.

Figure 15: Thermal Relief

5.5   Board Statistics

The final design had the following characteristics:

- *Pad And Via Holes* : 250

- *Pads* : 513

- *Tracks* : 1352

- *Vias*: 165

- *Components*: 97 (71 top, 26 bottom)

- *Nets*: 75

5.6   Release To Manufacturing

After the board design was complete, a design rule check (DRC) was performed within ProTel to check for incomplete connections, short circuits, and clearances below the minimum value. After several iterations, this check passed completely, and the design was ready for manufacturing files to be generated. Luckily, ProTel has a built-in facility for generating a wide variety of Gerber-format files. These were generated, along with a separate drill file, which tells the manufacturer precisely where to drill, and what size drill is needed. These files were submitted to ProtoExpress for manufacturing.

5.7   The Second Iteration

We received the first set of 3 boards approximately one week after submitting the Gerber files. Although the boards had been manufactured to our specifications, we quickly found, by test fitting all components, that several mistakes had been made in the first revision of the board. First, several of the vias were too small to fit the pins of their respective components, such as the 5MΩ potentiometers and the LEDs. Also, the decoupling capacitor dimensions were coded into ProTel incorrectly, so the component outlines were twice as big as the actual components. Finally, the most serious error was in the LMF100 footprint. The footprint we generated was for a 20-pin SOIC, whereas the LMF100 uses a special Wide SOIC package, which is nearly twice as wide as regular SOIC chips. One potential workaround for this would be to solder down one side of the chips, and solder jumper wires from each pin of the other side to the board. This, however, did not seem like a robust solution, and seemed very fault-prone, both electrically and mechanically. Therefore, a second revision of the board, APLS 2.0, was quickly made, and submitted for manufacturing. All of the above issues were remedied in this revision. The most difficult to correct was the LMF100 footprint, since the much larger footprint required moving a lot of resistors and capacitors around on the board, and rerouting over a hundred traces.

When we received the second board, we found that all components fit correctly, verifying that all the above issues had indeed been fixed. As we would later find, there were several smaller problems which still existed with the design, but these could be fixed without a new board revision (although one will be performed in the near future).

6   Assembly

Once the boards and components had been received, a brief conductivity test was performed to ensure that there were no glaring manufacturing defects. The many surface-mount components on the board were attached using reflow soldering, where a thin bead of solder paste is applied to each pad, the component is placed on top, and the entire board is placed in an oven. By carefully following a prescribed temperature profile, all the solder on the board reflows together, effectively soldering an entire side of the board in one pass. This procedure is made considerably more complicated when there are surface-mount components on both sides, for obvious reasons. Luckily, the surface tension of solder paste is such that, even when it reflows, it can support the weight of most surface-mount components, allowing a double-sided board to be assembled in two reflow passes. First, we applied solder paste to one set of pads at a time on the bottom side of the board (where there are fewer components), and placed the components on. For integrated circuits, we placed an individual solder bead on each pad when the pitch was over 50 mils, and a single bead down each row of pads when the pitch was under 50 mils. In the latter case, the solder mask should help attract the solder to one side or the other, ameliorating the serious problem of solder bridges. Once all components on the underside had been placed, we slid the entire board into the oven. After following the reflow profile (shown in Figure 16) and allowing the board to cool, we removed the board and inspected it for cold solder joints and solder bridges. Only a few of these were found, and most were easily fixed with a fine-tipped soldering iron. However, one of the resistor pads did not solder at all, and the inverter floated off the pads, so we had to reflow the bottom side again. After this second reflow, all the problems found after the first run were fixed. We could then move onto the top side.

We followed the same procedure on the top side (for surface-mount components only). The board-to-board connectors were particularly worrisome, due to their fine pitch and high pin count. We placed the board in the oven top-side-up, using protoboard and spare FR-4 material as standoffs to prevent the SMT components on the underside from resting on the grill of the oven. After a single reflow run, there were very few solder bridges and no cold solder joints on the top side, and none of the components on the bottom side were dislodged in the process. The lone concern coming out of the reflow process was that some of the joints on the bottom side had a gold tinge to them, indicating that perhaps the oven had gotten too hot on the first reflow run. However, the solder joints appeared to be in order, as evidenced by another conductivity test.

After soldering all the surface mount components, we soldered the through-hole components (all connectors, LEDs, bulk decoupling capacitors, potentiometers, and amplifiers) by hand, using a fine-tipped
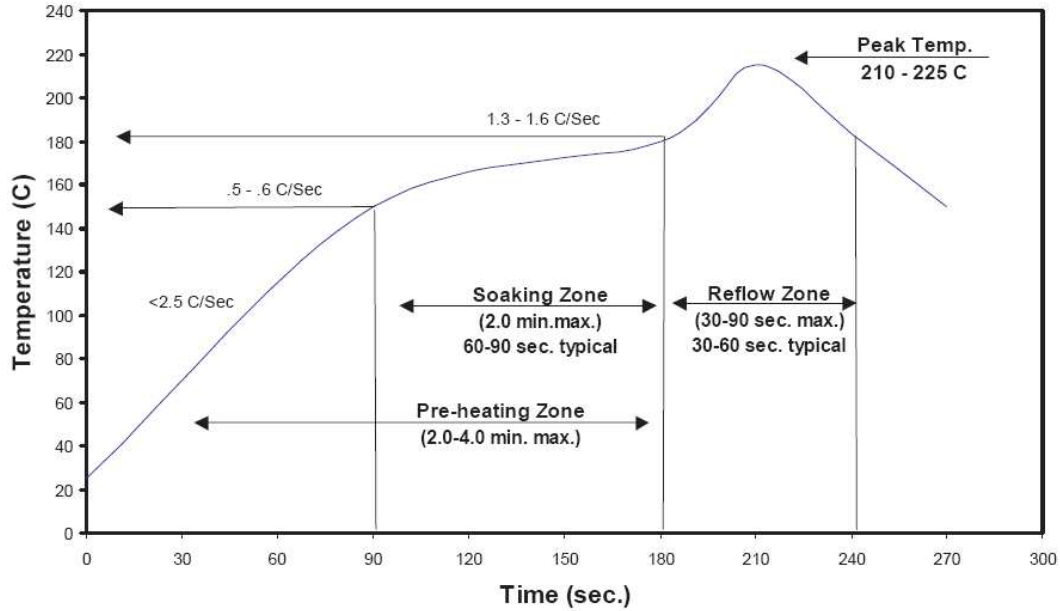
Figure 16: Solder Paste Reflow Profile

soldering iron. Though the absence of liquid solder flux made the aesthetic appearance of some of the joints somewhat sub-par, they all appeared to be electrically and mechanically sound, and the process went without incident. When all was completed, we were left with a fully-assembled board. A test fit of the FPGA board confirmed that the connectors had been spaced and oriented correctly. Photos of the fully-assembled board are shown in Figures 17 (top) and 18 (bottom). Photos of the FPGA board are shown in Figures 19 (top) and 20 (bottom). The two boards are shown mounted together in Figure 21.

7 Digital System Design

7.1 Receipt Time Determination

7.1.1 Design

Given digitized signals indicating that an in-band acoustic signal is being received, the task of determining the order and time of receipts presents itself. A number of plausible solutions were considered, the first of which was analog-to-digital conversion and software thresholding. Given the expense and comparative inelegance of a high sampling rate ADC, we quickly eliminated this option. We also considered using a microcontroller with on-chip analog comparators, such as the Silicon Laboratories (*silabs.com*) C8051F series, but the desired sample rate of tens of MHz could not be achieved with this solution. Likewise, using external analog comparators and digital input ports on such a microcontroller, we could not achieve the desired time resolution because even the overhead of a polling loop or interrupt service routine virtually guarantees a lower bound of tens of clock cycles between near-coincident arrivals.

A distinct digital system seemed to be a natural solution. The size and wiring complexity of the discrete logic necessary to store several high-resolution timestamps led us to consider a programmable logic device (PLD) implementation. Furthermore, an FPGA implementation eliminated the need for a separate microcontroller because of the availability of soft processor cores.

Initially, the high-level design of the receipt time determination module was as follows: if a comparator output goes high, latch it. Increment a counter every clock cycle, and if, on the rising edge of a clock cycle,
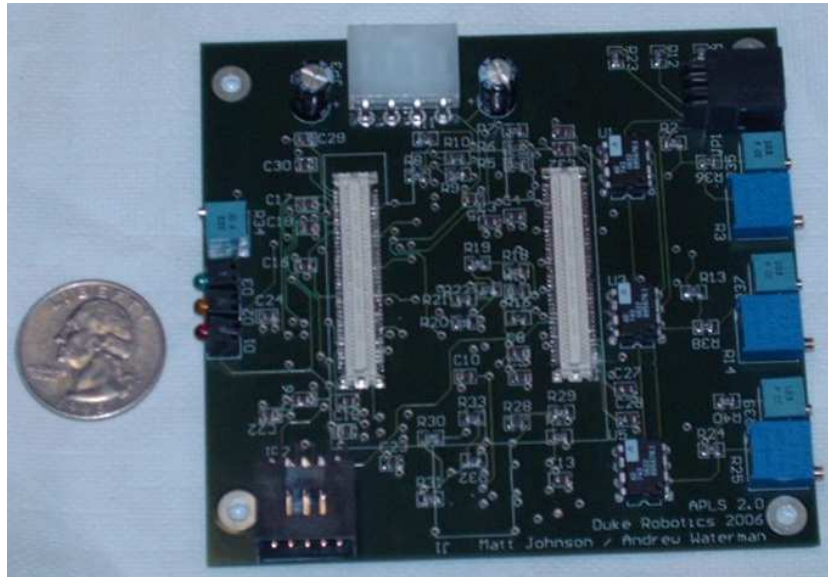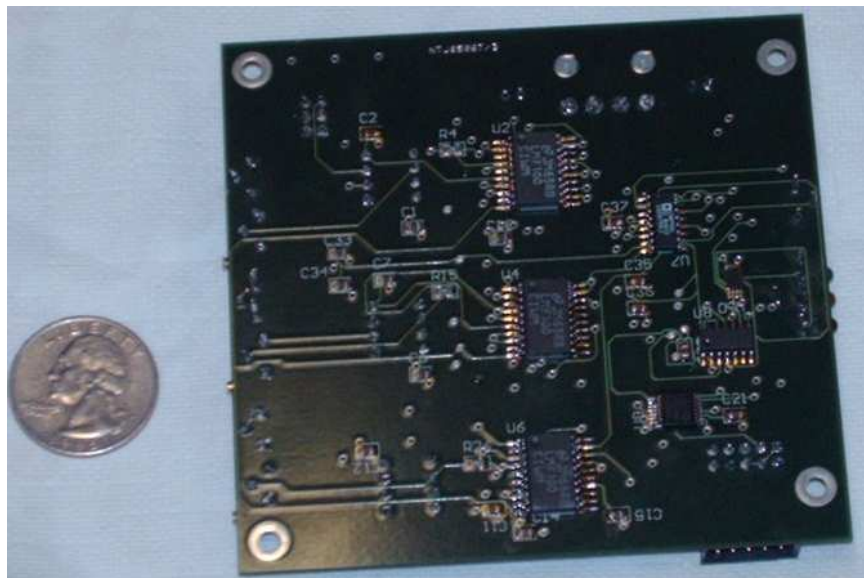
Figure 17: Top Side Of Baseboard
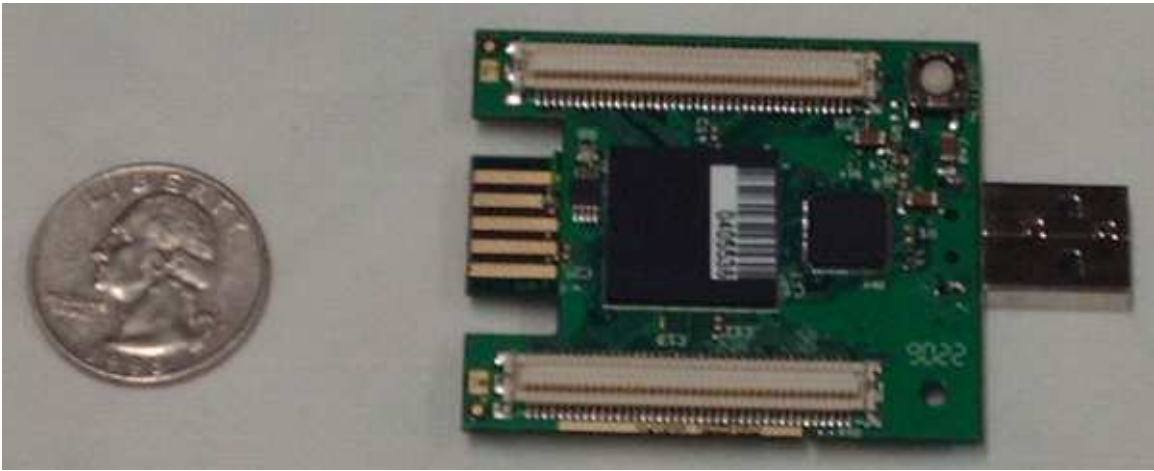

Figure 18: Bottom Side Of Baseboard

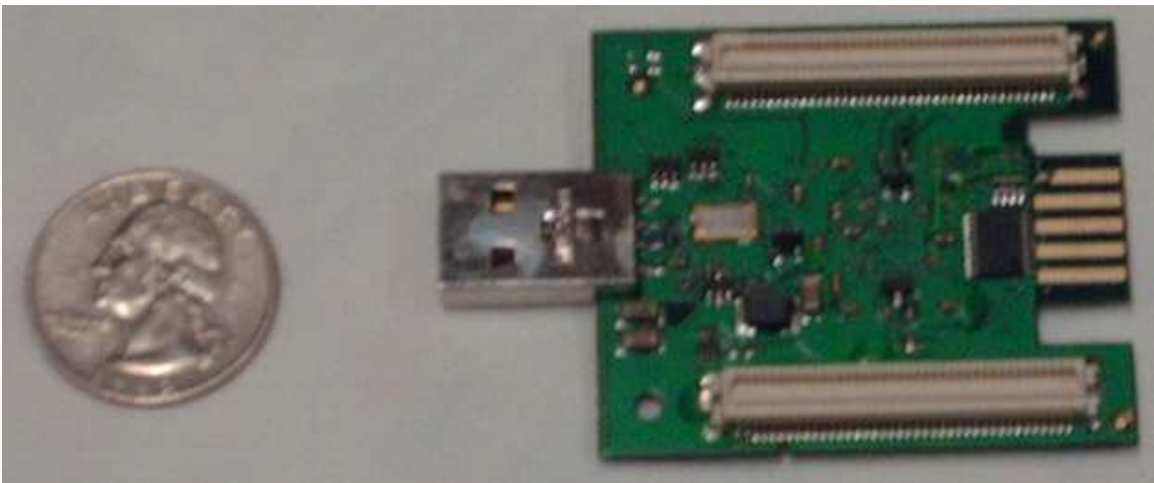Figure 19: Top Side Of Baseboard

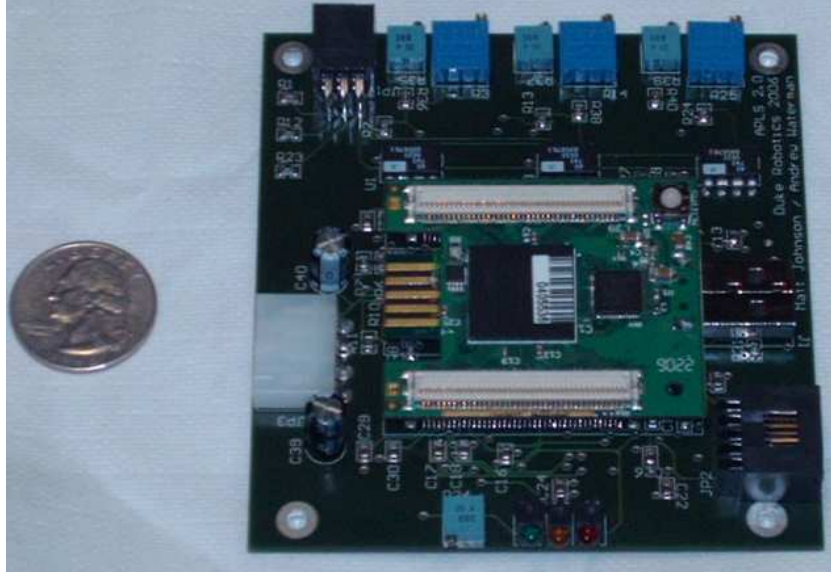Figure 20: Bottom Side Of Baseboard

27

Figure 21: Both Boards Mounted Together

a latch is high, store the current count in the corresponding register. When all three registers are populated, provide the two of the pairwise time differences as outputs[5] and indicate that these outputs are valid. A high-level block diagram of this module, dubbed *timecapture*, is provided in Figure 22.

Though the concept of this design holds, we saw fit to modify it in several ways. First, *timecapture* was made synchronous by simply removing the latches. We observed that, since the clock frequency of this module is (ideally) orders of magnitude greater than that of the comparator output, there is little benefit to perceiving a comparator output that is high for a fractional clock cycle to be a valid input; after all, it would have to be high for the setup and hold times, anyway, which are not too much less than the clock period.

An acknowledge signal, which essentially acts as a synchronous reset, was then added. The addition of this input signal allows *timecapture* to hold onto a sample until the host acquires it.

A flaw in the design was later identified and addressed. Suppose *timecapture* was acknowledged or otherwise reset while one or more comparator outputs were high. Then these would immediately be considered valid inputs at the start of the next *timecapture* cycle; as a result, the recorded time differences would not be meaningful. This problem was handily addressed by the addition of one n-bit shift register per input channel. The most recent input was required to be high and the $n - 1$ preceding inputs were required to be low in order to trigger a receipt. This solution also helped address the potentially problematic transient nature of the comparator outputs: the comparators take some time to reach a steady state, so receiving $n - 1$ low inputs followed by a high input suggests that the comparator has been stably low for a sufficient amount of time, and the comparator's input is just reaching the threshold. This behavior is quite desirable. In this implementation, letting n be 100 proved effective.

Finally, logic was added to reset the device if a valid input has been received on either one or two channels and a certain number of clock cycles has elapsed without valid inputs on all three channels. This so-called reset threshold realizes the constraint of Equation 10. The threshold is not a synthesis-time constant but is instead controlled by the host.

---

[5]The third pairwise time difference can be computed by trivially manipulating the other two.
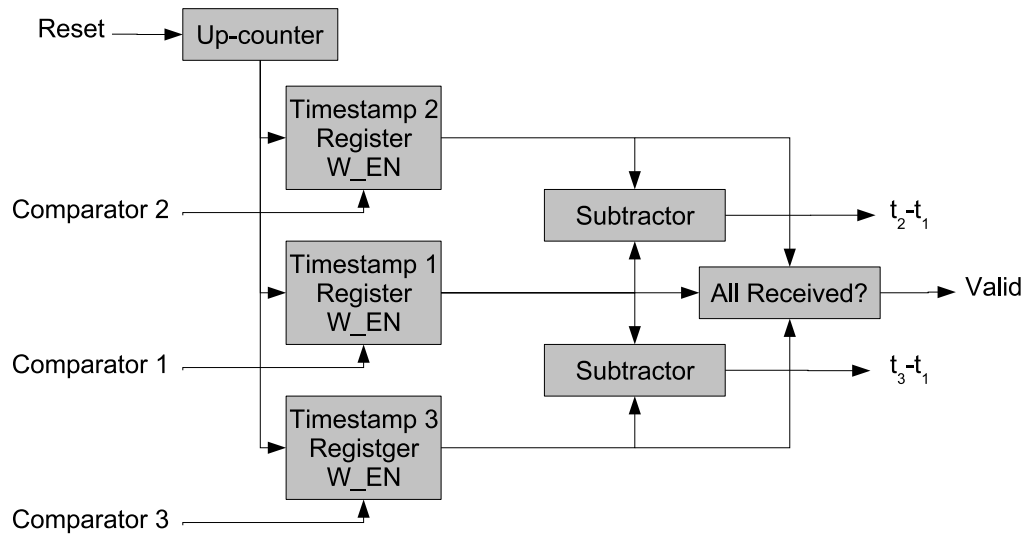
Figure 22: Block diagram of the *timecapture* module

7.1.2 HDL Implementation

A benefit of implementing a digital system on a PLD is the ability to take advantage of the useful abstractions that a hardware description language (HDL) offers. One need not, for example, be concerned with the data flow of an up-counter; he need only use VHDL's addition operator. Furthermore, in a design with extensive parallelism like this one–the same operations are being performed on three channels–simplicity in expression can be achieved using constructs like VHDL's *FOR ... GENERATE* loop.

Timecapture was initially described in VHDL. It was then translated to Verilog because this design is expressed with greater simplicity and elegance in that HDL. After the numerous improvements discussed above, the design was rewritten again in VHDL, a decision made because the Xilinx MicroBlaze soft processor core lacks a well-documented Verilog interface, and we wanted to avoid the complexities of a mixed-HDL implementation.

The VHDL implementation of *timecapture* is available in Section 11.1.

7.2 Configuring the FPGA

7.2.1 Initial Setup

Using the Spartan-3 FPGA seemed at first to be a straightforward task. The baseboard for the Trenz micromodules allowed for the easy connection of external signals, including a JTAG programmer to load implemented designs into the micromodule's PROM, which stores the FPGA's configuration between power cycles. Indeed, Xilinx provides an integrated development environment known as the ISE that makes the synthesis, implementation, and uploading of digital designs fairly streamlined; a simple test design was uploaded to and executed on the FPGA not long after connecting it to a computer with the ISE installed on it.

Unfortunately, the successful implementation of the MicroBlaze soft core was not so simple. Xilinx does provide a robust toolset called the Embedded Development Kit (EDK), a component of which is the Xilinx Platform Studio (XPS). This program synthesizes and implements custom configurations of the MicroBlaze core. The great depth and breadth of this program, unfortunately, gave it a steep learning curve. Our configuration, admittedly, did not ameliorate the situation: while Xilinx provides a detailed XPS tutorial, it assumes the use of a Spartan-3 development kit, which has a very different periphery, memory map, and memory hierarchy. As such, the tutorial could only be followed in parts, and even then, only with caution.

After some effort, a MicroBlaze was instantiated in an ISE project, but the test program (which sends a text string over RS-232) produced no serial output. Simulating the system to determine the cause of this problem, however, turned out to be a difficult task. Because the MicroBlaze VHDL implementation is encrypted, pre-compiled libraries must be given the simulator tool being used. However, the EDK documentation only described the process of exporting these libraries to the commercial simulators ModelSim and NCSim, which we did not have at our disposal; esoterically, the documentation stated only that the libraries could not be exported to Xilinx's own ISE Simulator. Left only with trial and error to determine the cause of the problem, we tried a variety of MicroBlaze configurations and implementation constraints. After exhaustive testing, the problem was determined to have two causes: first, with the intrusive software debugger enabled, the test program would not begin executing upon boot; and second, in a failure to read the Trenz micromodule documentation thoroughly, we had flagged the system clock pin as a local clock rather than as a clock provided on a general I/O pin by the USB interface.

Upon correcting these problems, the test program executed successfully.

7.2.2 Integrating the Digital Subsystems

The MicroBlaze implements the On-Chip Peripheral Bus (OPB), developed by IBM as a means of connecting on-chip devices to the processor core while abstracting the low-level connection and arbitration

details. The OPB, as such, seemed to be the logical choice for connecting *timecapture* to the MicroBlaze. The XPS ostensibly simplifies implementing an OPB module by generating all of the required interface logic, allowing the designer to immediately begin implementing the device itself.

Unfortunately, our instance of the MicroBlaze ceased to function when even a stub OPB module was included in the embedded platform. We decided that, given the severe time limitations and the lack of obvious reasons for failure, it would be more productive to pursue a less elegant interface and implement *timecapture* as an OPB module in a future revision.

The chosen interface uses General Purpose I/O (GPIO) modules to form a fully parallel connection between *timecapture* and the MicroBlaze. Two 32-bit unidirectional buses provide the two pairwise time differences $t_2 - t_1$ and $t_3 - t_1$, and a 32-bit bidirectional bus provides the Valid signal to the MicroBlaze and provides the Acknowledge signal and the reset threshold to *timecapture*. Because all of these connections are formed within the FPGA, this parallel interface does not increase the hardware cost but offers much faster and simpler communication between the two devices than would a serial interface of some kind.

A successful synthesis and implementation of this configuration resulted in an f_{max} of 67 MHz, so the MicroBlaze-*timecapture* combination was clocked with the 60 MHz one generated by the USB controller. Consequently, this frequency corresponds the sampling rate of the *timecapture* module.

7.3 Software

As part of the EDK, Xilinx includes a distribution of the GNU C Compiler (GCC). This feature made the Spartan-3/MicroBlaze platform particularly attractive, as a high-level software implementation is less time consuming than a native one. As such, the two major software components–the *timecapture* driver and the bearing angle calculation–are written in C.

7.3.1 Timecapture Driver

To interact with *timecapture* from within C programs on the MicroBlaze, we wrote a simple driver (see Sections 11.2 and 11.3). A call to *timecapture_init* initializes a *timecapture* data structure, which contains information pertinent to the operation of the driver. Calls to *timecapture_read* attempt to retrieve data samples from the device. This function does not block; if no data is present, it returns immediately. A subsequent call to *timecapture_angle* computes the bearing corresponding to that sample, as described below.

7.3.2 Bearing Angle Calculation

With the pairwise receipt time differences obtained, the problem of calculating the bearing remains. To achieve this end, the closed-form solution for θ developed in Section 2.1 was used. The availability of the MicroBlaze FPU and the C standard library math functions were of much help: the computation of the arctangent is a nontrivial task in itself, and it would be painfully slow if a software floating-point library were used. The C code that computes the bearing resides in the *acoustics_angle* function in Section 11.3.

7.3.3 Main Routine

For the purposes of testing and development, two main routines were developed. Initially, we found that sending the most recent time difference and angle measurements over RS-232, ad infinitum, was the most useful debugging tool. When we ascertained that these algorithms worked as intended, we found that a short statistical summary of the data was more useful than the data itself; as such, the main routine in Section 11.4 provides, for chunks of 100 samples, the average time differences, the standard deviations thereof, and the estimated bearing corresponding to the averaged data.

Of course, when deployed, the main routine should send the data in a pre-defined, efficient manner. A plausible implementation might utilize the USB PHY on the micromodule and send bearings in IEEE 754 format.

8   Debug

As in most complex embedded systems, the art of debugging was needed to bring the board from its fully-assembled state to its fully-working state. We were pleased to see that upon applying power to the power connector, the current consumption was in the same range as when we had breadboarded the analog circuitry, which reassured us that at least the power pins of the on-board ICs had been soldered correctly. However, through a long debug process, we discovered several minor design errors which prevented the board from functioning fully:

- Due to insufficient attention to the Trenz Electronic documentation, the pin chosen for the LMF100 filter clock was also connected to a pushbutton on the FPGA board, used for programming. Since we had another signal dedicated to programming, this was not an issue per se. However, the pushbutton had 2 pull-up resistors which would interfere significantly with the clock signal. To remedy this problem, these two resistors (0603 and 0402 form factor) were desoldered.

- Due a simple schematic entry error, the signal ENABLE12, brought out to pin LL2 on the FPGA board, was tied to ground instead of left floating. This disabled the onboard 1.2V voltage regulator, which supplies the power for the Spartan-3's core logic. In order to "undo" this connection, tweezers and a utility knife were used to break the board-to-board connector pin and cut the PCB trace associated with LL2, respectively.

- Due to another schematic entry error, the signal $\overline{\text{FORCEOFF}}$ was tied to ground instead of +5V on the RS-232 level shifter. This caused the transmitter to be constantly powered off. We verified this by observing the input waveform (Tx from the FPGA) on pin 11 of the level shifter, and the lack of any discernible output on pin 13. Also, this signal does not affect the RS-232 receiver, and we verified that receiving still worked by pressing keys in a PC-based terminal program, and observing the waveform on the output side of the level shifter. In order to fix this problem, we disconnected the $\overline{\text{FORCEOFF}}$ pin from the PCB, and bridged it to an adjacent pin with a +5V connection with solder.

After diagnosing and remedying these three issues, and modifying a female USB connector to fit onto the FPGA board (to supply power), the system worked flawlessly. These three issues will be addressed in the next revision of the APLS system.

8.1   Frequency Response

Once the board had been assembled, we felt it would be useful to measure the frequency response of the two major elements of the analog signal path, the amplifier and the filter. We accomplished this using an Agilent Network Analyzer, with one probe connected to the hydrophone input and the other connected to either the output of the AD743 or the output of the LMF100. The results obtained for the former case are shown in Figure 23. The log-magnitude frequency response of the combination of amplifier and filter is shown in Figures 24 and 25. Note that these are two views of the same data (different input frequency ranges). The phase response of the combined amplifier and filter is shown in Figure 26.

From these measurements, several conclusions can be drawn as to the efficacy of the amplifier-filter solution we have chosen. First, the AD743's frequency response is flat throughout our range of interest (20kHz - 35kHz). Its -3dB frequency is around 38kHz, meaning that our pinger signals will not be significantly attenuated by the amplifier. The combined magnitude response of the amplifier and filter is also encouraging. The peak response is at 20kHz, which is precisely what the FPGA's clock signal was tuned to, and the filter effectively attenuates all signals outside the passband. In fact, the response at 10kHz and 40kHz is 55dB lower than the peak value.

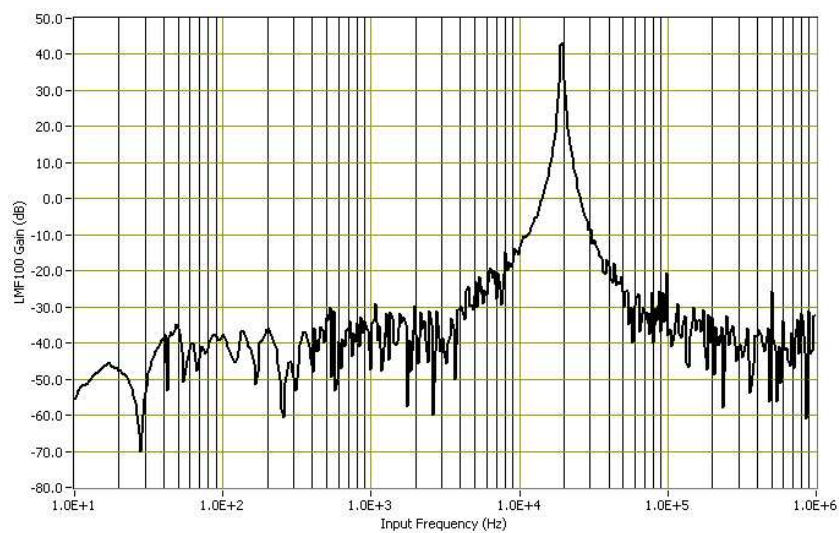Figure 23: AD743 Log-Magnitude Frequency Response

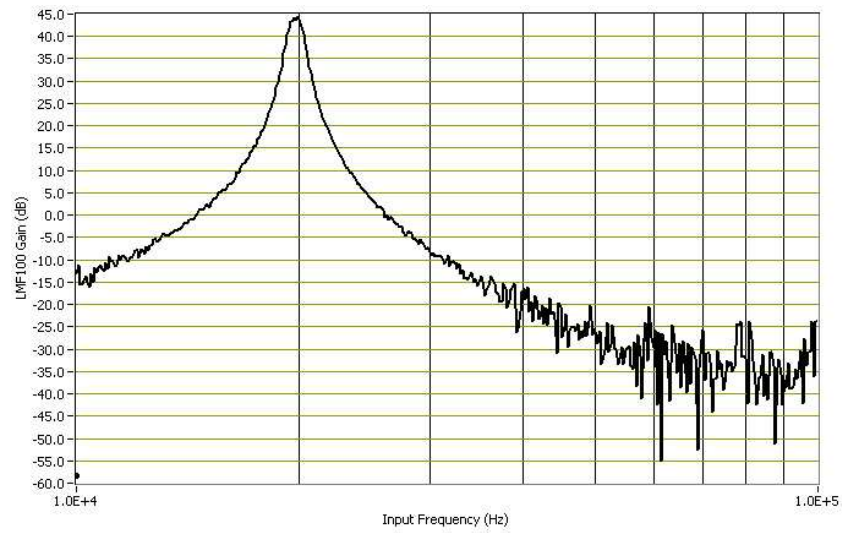Figure 24: AD743/LMF100 Log-Magnitude Frequency Response

Figure 25: AD743/LMF100 Log-Magnitude Frequency Response, Compressed X Axis
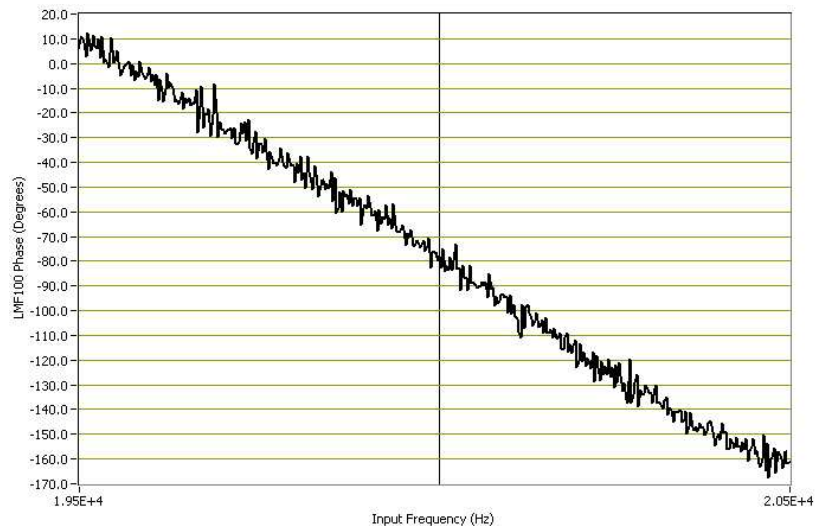

Figure 26: AD743/LMF100 Phase Response

One concern that arose from these measurements is the phase response of the fourth-order bandpass filter. While the response observed is predictable, given the nature of bandpass filters in general, the large first derivative of the phase response in the passband is confirmation of our previous belief that proper calibration would be essential for the system to operate correctly. Ostensibly, all three hydrophones should receive the same frequency to within a very small tolerance, but any mismatch in LMF100 device parameters or resistor values can potentially affect the phase of the output signal, and thus the delay imposed on the digitized square wave, in a significant way. For this reason, it is important to characterize the intrinsic phase shift between all three channels.

8.2   Calibration

8.2.1   Hardware Calibration

In order to ensure that all three channels operated as uniformly as possible, despite the inevitable variation in IC device parameters and passive component values, several means of calibrating the circuit were designed in. The gain and offset of the amplifier stages are adjustable by individual potentiometers, enabling device variations to be accounted for and corrected. Additionally, due to varying pinger characteristics, such as distance and ping energy, it may be necessary to raise or lower the threshold voltage used to digitize the amplified and filtered hydrophone signal. This, too, is accomplished via the use of a 12-turn potentiometer.

There is one major cause of phase shift that cannot be easily calibrated for in hardware. Mismatches in resistor values between channels could cause the LMF100 filter characteristics, such as center frequency (f_o) and quality factor (Q), to vary, shifting the phase response. While it is not reasonable to tune each of these resistances by a potentiometer, one can still correct for these errors when computing the bearing angle.

8.2.2   Software Calibration

Instead of calibrated for passive component-induced phase shift in the filter stage, one can simply observe the phase difference between each pair of channels when an identical input waveform is applied to each, and account for this in software. This is the approach we took. We used our software to compute the average delay between channel 1 and channels 2 and 3 for a common 20kHz input waveform. We found that channels 2 and 3 trailed channel 1 by 116 cycles. This translates to 1.93μs, or approximately a 14° phase shift. From the phase response shown in Figure 26, we see that this corresponds to roughly an 80Hz center frequency shift. Given the equations for the LMF100 filter elements' center frequency, this result seems reasonable. With these results in hand, we can simply subtract this known offset from all measured delays, in order to compute the true delay between channels.

8.3   Testing Methodology

To verify the proper operation of the system, we used three sine waves of the same frequency but of varying phases as inputs to the device. The phase shifts represent the receipt time delays. Encouragingly, time differences were immediately recognized by the software; however, the software produced the correct answer only periodically. After some thought, we realized that if the data was acknowledged while a comparator's output was high, then when *timecapture* restarted, it immediately recognized that as a valid input. Thus, accurate results were only obtained when the device began sampling when all comparator inputs were low–hence the periodicity of the error. This problem was solved with the shift register solution discussed in Section 7.1.1.

9   Results

A sample of software output has been provided below. In this test case, the sine wave corresponding to channel 3 arrived first, followed by those of channel 2 and channel 1. Thus, $t_3 < t_2 < t_1$, so the pairwise time difference results are reasonable. The calculated bearing (in radians) also makes sense: since the

pinger originated closest to hydrophone 3 and next closest to hydrophone 2, it is confined to the range $-\pi < \theta < -\frac{1}{3}\pi$; indeed, the calculated bearing lies therein. The analytical result can be verified using the expression for θ in Section 2.1.

```
Starting statistics loop...
Statistics for last 100 pings:
Average of t2-t1: -734.870
Standard deviation of t2-t1: 23.139
Average of t3-t1: -1275.420
Standard deviation of t3-t1: 24.209
Estimated trigonometric bearing angle: -2.705
```

The standard deviation of receipt time differences of about 20 clock cycles indicates an effective accurate sampling rate of about 3 MHz. Per the error analysis in Section 2.3, this sampling rate is sufficient to provide better than degree resolution on average.

10 Application

In practice, knowing a bearing to a beacon may not be sufficient: for example, in order to navigate to a beacon, the distance must be known in addition. Fortunately, it is possible in many circumstances to determine a beacon's spatial position accurately.

Assuming that APLS is deployed in a motile system that has an accurate notion of its velocity vector (and thus of its position relative to an arbitrary reference point), the system can obtain bearings from a number of noncollinear positions. A mathematical optimization technique can then be employed to approximate the source. Ordinary least squares is a reasonable choice, since it is simple to implement, gives a polynomial-time solution (with respect to the number of data points), and provides an estimate of the measurement error. We term this acoustic beacon location technique *overspecified triangulation*.

11 Code

11.1 Timecapture VHDL Implementation - timecapture.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity timecapture is
  port
  (
    --the timecapture clock
    Clk    : in  std_logic;

    --asynchronous reset
    nReset : in  std_logic;

    --the outputs of the three comparators
    hp     : in  std_logic_vector(2 downto 0);

    --the host's acknowledge signal
    Ack    : in  std_logic;

    --the largest time difference before resetting
    reset_threshold : in std_logic_vector(31 downto 0);
```

```vhdl
    --channel 2 receipt time minus channel 1 receipt time (2's complement)
    t2mt1  : out std_logic_vector(31 downto 0);

    --channel 3 receipt time minus channel 1 receipt time (2's complement)
    t3mt1  : out std_logic_vector(31 downto 0);

    --whether or not the output is valid
    Valid  : out std_logic
  );
end timecapture;

architecture behavior of timecapture is
  --define a shift register to be a 100-bit vector
  subtype tshiftreg is std_logic_vector(99 downto 0);
  --define a vector of shift registers
  type shiftreg_vector is array (integer range <>) of tshiftreg;

  --define a timestamp to be a 32-bit unsigned integer
  subtype timestamp is unsigned(31 downto 0);
  --define a vector of timestamps
  type timestamp_vector is array (integer range <>) of timestamp;

  --the up-counter
  signal counter : timestamp;

  --the vector of timestamps for each channel
  signal timestamps : timestamp_vector(2 downto 0);

  --whether or not each timestamp is valid
  signal timestamps_valid : std_logic_vector(2 downto 0);

  --used to create a one-cycle delay between valid signals
  --and a "Valid" output (to allow the subtractors to compute
  --their results without reducing fmax or introducing race conditions)
  signal valid_delay : std_logic;

  --whether or not to reset on the next rising edge
  signal synchronous_reset : std_logic;

  --whether or not the reset threshold has been exceeded
  signal threshold_exceeded : std_logic;

  --the shift registers
  signal shiftreg : shiftreg_vector(2 downto 0);
  --whether or not the n-th bit is 1 an the n-1st to 0th bits are 0
  signal shiftreg_output : std_logic_vector(2 downto 0);

begin

  --synchronously reset if the host has acknolwedged or the threshold
  --has been exceeded
  synchronous_reset <= Ack or threshold_exceeded;
```

```vhdl
--synchronously up-count starting when one of the channels becomes
--valid.  if the asynchronous or synchronous resets are applied,
--set the counter to 0.
process(Clk,nReset)
begin
  if(nReset='0') then
    counter <= "00000000000000000000000000000000";
  elsif(rising_edge(Clk)) then
    if(synchronous_reset='1') then
      counter <= "00000000000000000000000000000000";
    elsif(timestamps_valid(2)='1' or timestamps_valid(1)='1' or
          timestamps_valid(0)='1') then
      counter <= counter+1;
    end if;

    --check if we've exceeded the reset threshold
    if(valid_delay='0' and counter > unsigned(reset_threshold)) then
      threshold_exceeded <= '1';
    else
      threshold_exceeded <= '0';
    end if;

  end if;
end process;


--implement the shift registers.  if we observe 99 0's followed
--by a 1, indicate that the channel output is valid.  in the event
--of a reset, populate the shift registers with 1's
shiftregs: for i in 2 downto 0 generate
  process(Clk,nReset)
  begin
    if(nReset='0') then
      shiftreg(i) <= "1111111111111111111111111111111111111111"&
                     "1111111111111111111111111111111111111111"&
                     "1111111111111111";
      shiftreg_output(i) <= '0';
    elsif(rising_edge(Clk)) then
      if(synchronous_reset='1') then
        shiftreg(i) <= "1111111111111111111111111111111111111111"&
                       "1111111111111111111111111111111111111111"&
                       "1111111111111111";
        shiftreg_output(i) <= '0';
      else
        --shift everything up
        for j in 0 to 98 loop
          shiftreg(i)(j+1) <= shiftreg(i)(j);
        end loop;

        --shift in the comparator output
        shiftreg(i)(0) <= hp(i);

        if(shiftreg(i) = ("0000000000000000000000000000000000000000"&
                          "0000000000000000000000000000000000000000"&
                          "0000000000000001")) then
```

```vhdl
              shiftreg_output(i) <= '1';
            else
              shiftreg_output(i) <= '0';
            end if;
          end if;
        end if;
      end process;
  end generate shiftregs;

  --store the counter value in a timestamp if the corresponding shift
  --register output is valid.  on reset, store 0 in all timestamps
  capture: for i in 2 downto 0 generate
    process(Clk,nReset)
    begin
      if(nReset='0') then
        timestamps_valid(i) <= '0';
        timestamps(i) <= "00000000000000000000000000000000";
      elsif(rising_edge(Clk)) then
        if(synchronous_reset='1') then
          timestamps_valid(i) <= '0';
          timestamps(i) <= "00000000000000000000000000000000";
        elsif(timestamps_valid(i)='0' and shiftreg_output(i)='1') then
          timestamps_valid(i) <= '1';
          timestamps(i) <= counter;
        end if;
      end if;
    end process;
  end generate capture;

  --if all channels are valid, delay one clock cycle, then raise
  --the Valid output (to allow subtractors to catch up)
  process(Clk,nReset)
  begin
    if(nReset='0') then
      Valid <= '0';
      valid_delay <= '0';
    elsif(rising_edge(Clk)) then
      if(synchronous_reset='1') then
        valid_delay <= '0';
        Valid <= '0';
      elsif(valid_delay='0') then
        valid_delay <= timestamps_valid(2) and timestamps_valid(1) and
                       timestamps_valid(0);
      else
        Valid <= '1';
      end if;
    end if;
  end process;

  --provide two pairwise time differences as outputs
  t2mt1 <= std_logic_vector(signed(timestamps(1)-timestamps(0)));
  t3mt1 <= std_logic_vector(signed(timestamps(2)-timestamps(0)));
end behavior;
```

11.2 Timecapture Driver - timecapture.h

```c
#ifndef _TIMECAPTURE_H
#define _TIMECAPTURE_H

#include "xgpio.h"

// the gpio_config bit corresponding to Valid
#define TIMECAPTURE_VALID 0x00000001
// the gpio_config bit corresponding to Acknowledge
#define TIMECAPTURE_ACK 0x80000000
// the reset threshold (0 < rt < 2^28)
#define TIMECAPTURE_RESET_THRESHOLD 1500

// calibration offsets.
// if channel 2 leads channel 1, provide a positive offset
#define TIMECAPTURE_OFFSET_T2MT1 -116
#define TIMECAPTURE_OFFSET_T3MT1 -116

// timecapture driver data structure
typedef struct
{
  XGpio gpio_t2mt1;  // the GPIO struct for the t2-t1 connection
  XGpio gpio_t3mt1;  // the GPIO struct for the t3-t1 connection
  XGpio gpio_config; // the GPIO struct for the config connection

  int t2mt1;         // the most recent value of t2-t1
  int t3mt1;         // the most recent value of t3-t1
} timecapture;

//#define INVALID_ANGLE -10.0f // unused

// initialize the timecapture data structure and reset the module
void timecapture_init(timecapture* tc);

// attempt to read a sample from timecapture; don't block.
// returns nonzero iff a valid sample is present.
// in either case, the values of the t2-t1 and t3-t1
// signals are returned in the corresponding fields in tc.
int timecapture_read(timecapture* tc);

// acknowledge the most recent data (for internal use only)
void timecapture_ack(timecapture* tc);

// determine a bearing from the most recent measurement
double timecapture_angle(timecapture* tc);

// given two pairwise time differences, calculate the bearing
// to the beacon
double acoustics_angle(int t2mt1, int t3mt1);

#endif
```

11.3   Timecapture Driver - timecapture.c

```c
#include "xgpio.h"
#include "timecapture.h"
#include "xparameters.h"
#include "math.h"

void timecapture_init(timecapture* tc)
{
  // initialize GPIOs
  XGpio_Initialize(&tc->gpio_t2mt1,XPAR_GENERIC_GPIO_DEVICE_ID);
  XGpio_Initialize(&tc->gpio_t3mt1,XPAR_GENERIC_GPIO_1_DEVICE_ID);
  XGpio_Initialize(&tc->gpio_config,XPAR_GENERIC_GPIO_2_DEVICE_ID);

  // config channel 1 = inputs (no change needed)
  // config channel 2 = outputs
  XGpio_SetDataDirection(&tc->gpio_config,2,0x00000000);

  // reset timecapture by ACKing
  timecapture_ack(tc);
}


void timecapture_ack(timecapture* tc)
{
  // raise Ack, then lower it.  keep the reset threshold
  // set correctly during this process.
  XGpio_DiscreteWrite(&tc->gpio_config,2,TIMECAPTURE_ACK | TIMECAPTURE_RESET_THRESHOLD);
  XGpio_DiscreteWrite(&tc->gpio_config,2,TIMECAPTURE_RESET_THRESHOLD);
}


int timecapture_read(timecapture* tc)
{
  // get the Valid flag
  int valid = XGpio_DiscreteRead(&tc->gpio_config,1) & TIMECAPTURE_VALID;

  // read values of t2mt1, t3mt1
  tc->t2mt1 = XGpio_DiscreteRead(&tc->gpio_t2mt1,1) + TIMECAPTURE_OFFSET_T2MT1;
  tc->t3mt1 = XGpio_DiscreteRead(&tc->gpio_t3mt1,1) + TIMECAPTURE_OFFSET_T3MT1;

  // if valid, acknowledge the data and return nonzero
  if(valid)
  {
    timecapture_ack(tc);
    return 1;
  }

  // there is no valid data, so return zero
  return 0;
}

double timecapture_angle(timecapture* tc)
{
  // just call acoustics_angle on our most recent sample
  return acoustics_angle(tc->t2mt1,tc->t3mt1);
```

```c
}

double acoustics_angle(int t2mt1, int t3mt1)
{
  double ratio, result;
  int t3mt2 = t3mt1 - t2mt1;

  // handle the corner case t3 == t2
  if(t3mt2 == 0)
    return t2mt1 > 0 ? 0 : M_PI;

  // no need to handle the corner case rho = 1/2
  // because the atan function behaves properly at infinity.
  // so compute the ratio rho and angle
  ratio = ((double)t3mt1)/((double)t3mt2);
  result = atan(M_SQRT3/(2.0*ratio-1.0));

  // if the sign of the angle does not agree with
  // the sign of t3-t2, add/subtract pi
  if(result < 0 && t3mt2 > 0 || result > 0 && t3mt2 < 0)
    result += result < 0 ? M_PI : -M_PI;

  // done!
  return result;
}
```

11.4 Main Program - main.c

```c
#include "xparameters.h"
#include "stdio.h"
#include "math.h"
#include "timecapture.h"

// get a string representation of a float
char* float2str(double f);

// get statistical data on NUM_SAMPLES samples
#define NUM_SAMPLES 100
void statistics_loop(timecapture* tc);

int main (void)
{
  timecapture tc;
  timecapture_init(&tc);

  while(1)
    statistics_loop(&tc);

    return 0;
}

void statistics_loop(timecapture* tc)
{
  xil_printf("Starting statistics loop...\r\n");
```

```c
  int t2mt1[NUM_SAMPLES];
  int t3mt1[NUM_SAMPLES];
  int i;
  double t2mt1_avg = 0.0, t3mt1_avg = 0.0, t2mt1_stddev = 0.0, t3mt1_stddev = 0.0;

  // run until we obtain 100 samples
  for(i = 0; i < NUM_SAMPLES; )
  {
    // attempt to get a measurement from timecapture
    if(timecapture_read(tc))
    {
      // if successful, add to running average
      t2mt1_avg += t2mt1[i] = tc->t2mt1;
      t3mt1_avg += t3mt1[i] = tc->t3mt1;
      i++;
    }

    // give timecapture a moment to regain its bearings... pardon the pun
    t2mt1_avg += atan(12);
    t2mt1_avg -= atan(12);
  }

  // compute averages
  t2mt1_avg /= (double)NUM_SAMPLES;
  t3mt1_avg /= (double)NUM_SAMPLES;

  // compute standard deviations
  for(i = 0; i < NUM_SAMPLES; i++)
  {
    t2mt1_stddev += ((double)t2mt1[i]-t2mt1_avg)*((double)t2mt1[i]-t2mt1_avg);
    t3mt1_stddev += ((double)t3mt1[i]-t3mt1_avg)*((double)t3mt1[i]-t3mt1_avg);
  }
  t2mt1_stddev = sqrt(t2mt1_stddev/(double)(NUM_SAMPLES-1));
  t3mt1_stddev = sqrt(t3mt1_stddev/(double)(NUM_SAMPLES-1));

  // print out the statistics
  xil_printf("Statistics for last %d pings:\r\n",NUM_SAMPLES);
  xil_printf("Average of t2-t1: %s\r\n",float2str(t2mt1_avg));
  xil_printf("Standard deviation of t2-t1: %s\r\n",float2str(t2mt1_stddev));
  xil_printf("Average of t3-t1: %s\r\n",float2str(t3mt1_avg));
  xil_printf("Standard deviation of t3-t1: %s\r\n",float2str(t3mt1_stddev));
  xil_printf("Estimated trigonometric bearing angle: %s\r\n\r\n",
    float2str(acoustics_angle((int)t2mt1_avg,(int)t3mt1_avg)));
}

// return a string representation of a floating point number.
// not re-entrant, doesn't cover whole range of floats.
#define FLOATBUFSIZE 16
char* float2str(double f)
{
  static char floatbuf[FLOATBUFSIZE];
  int i;
  unsigned long j = (unsigned long)((f < 0 ? -f : f)*1000.0f);
```

```
  char* ptr = floatbuf + FLOATBUFSIZE - 1;

 *ptr = 0;
 for(i = 0; i < 3; i++)
 {
   *--ptr = '0' + j%10;
j /= 10;
 }

 *--ptr = '.';

 for(i = 0; i < 10 && j; i++)
 {
*--ptr = '0' + j%10;
j /= 10;
 }
   if(f < 0)
   *--ptr = '-';

 return ptr;
}
```

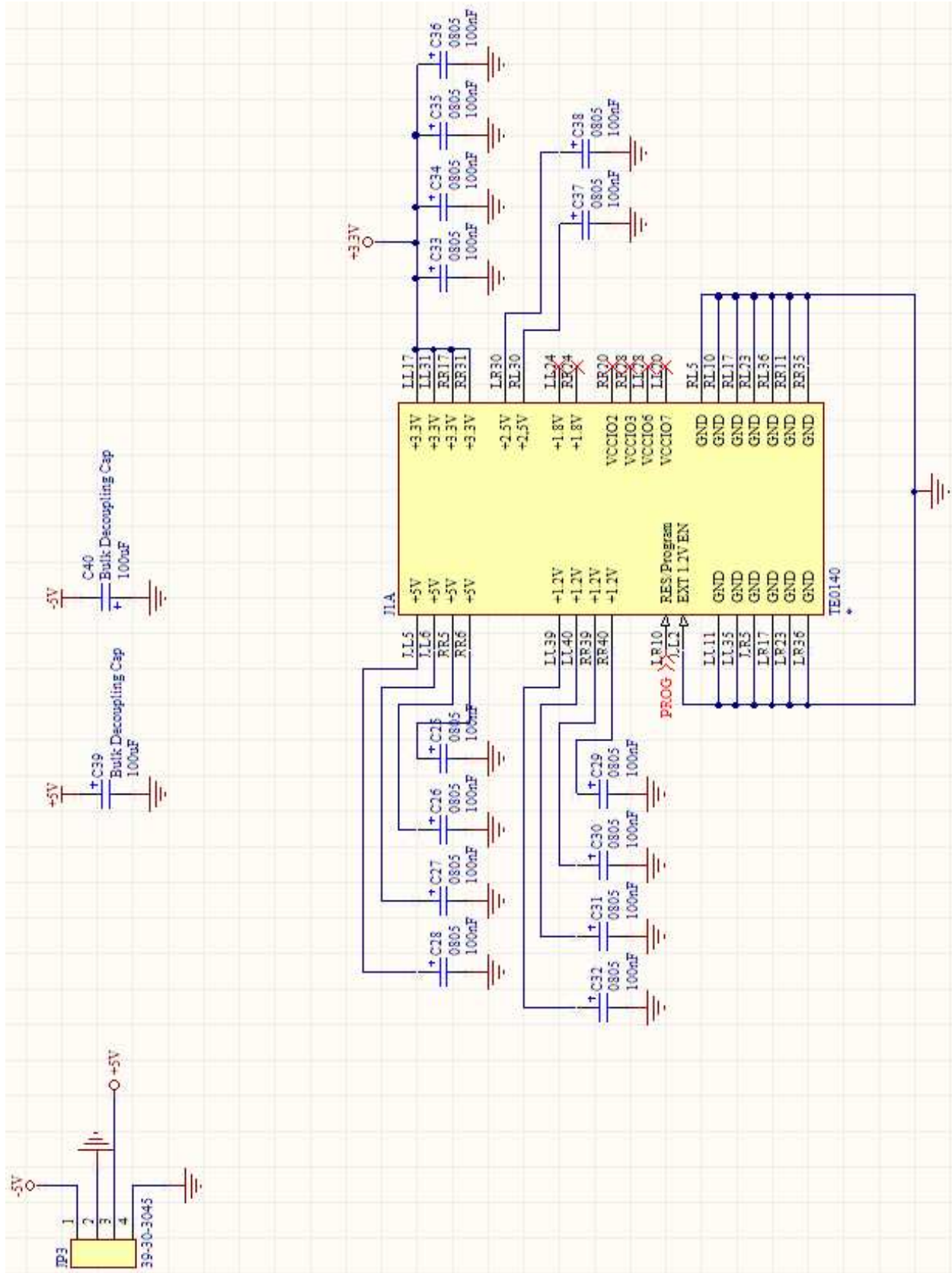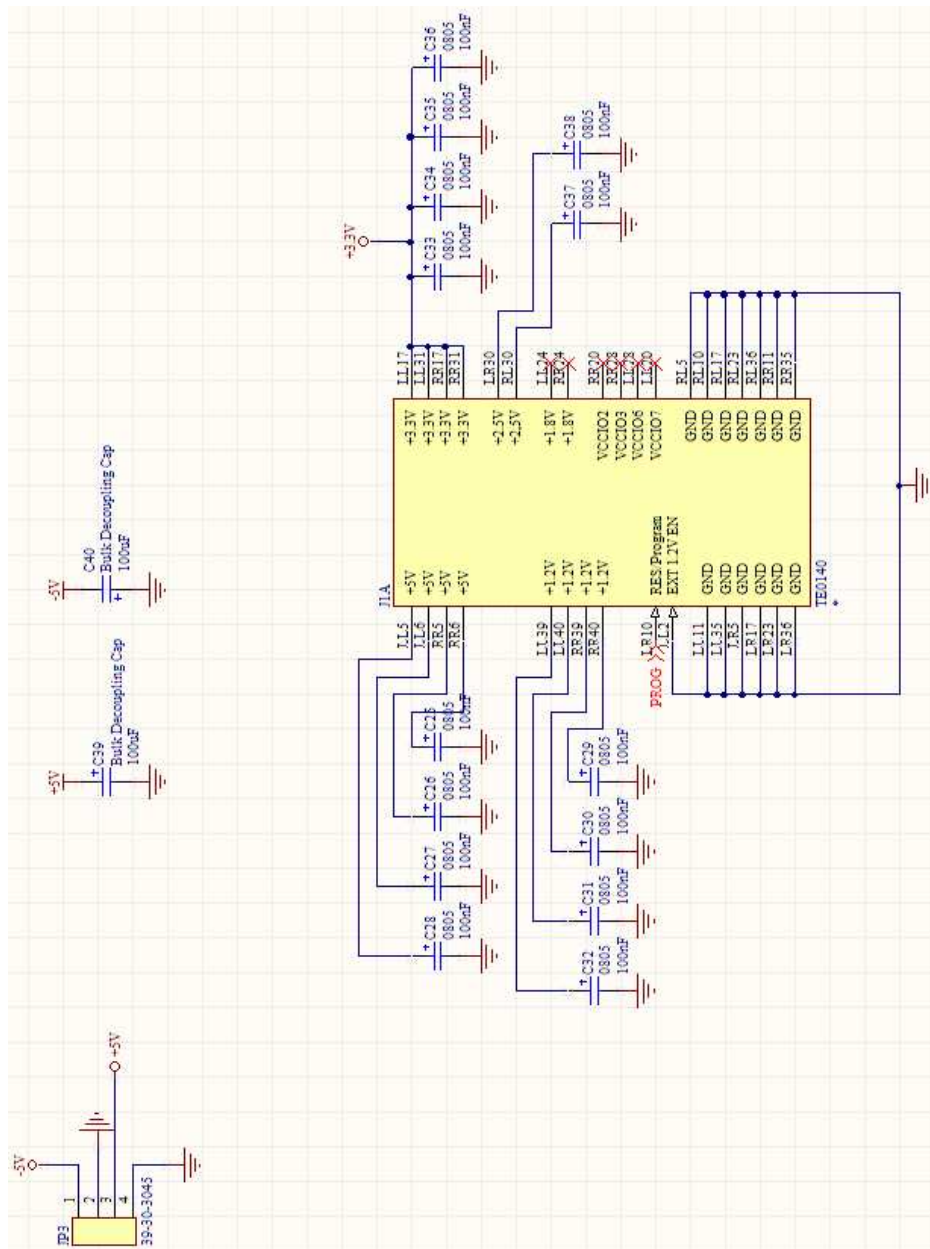# 12   Additional Schematics

Figure 27: FPGA signals and decoupling capacitors

Figure 28: FPGA signals, RS-232 level shifter, and connectors