Tunafish
A Novel Approach to Active 2D Sonar

Leon Fay
Miranda Ha
Vinith Misra
6.111 Spring 2006

May 19, 2006

Contents

Abstract

We present a functioning active sonar that leverages a novel signal processing approach to achieve rapid frame rates with low quality devices. By placing increased burden on the processing, what normally requires dozens of transmissions may be performed with only one.

Cheap ultrasonic transducers and receivers have been used in order to emphasize device-independence. Furthermore, home-brewed 2-bit analog-to-digital converters are used in lieu of more expensive options. The algorithm, by design, simply does not require the level of detail afforded by higher resolution sampling systems.

On the digital front, the transmission, data gathering, processing, displaying, and even serial link-up have been fully pipelined in order to maximize frame rate. While only single object tracking is demonstrated reliably (due to the short range of the transducers), there exists near-full support in both the algorithm and the system to handle multiple objects. Display of these objects is possible from both top and frontal views.

Various subtleties and optimizations - some highly nonintuitive, others virtually necessary for proper operation - have been discovered as well. We describe these in addition to the implementation.

1 Overview

SONAR - SOund Navigation And Ranging - was a technology first developed in the early 20th century as a means to locate objects beneath water. The willingness of sound waves to significantly reflect off most surfaces has contributed to its dominance today as a means of localization in the seas.

There are largely two categories of sonar: active and passive. Passive sonar is a "listening" system that simply attempts to localize the source of any sounds it hears. In military applications where stealth is a priority, this is often the only option.

Active sonar, on the other hand, involves a more active role on the part of the detector. A pulse of sound, or a "ping" is emitted with a transmitter, and reflections of this pulse are interpreted for the desired information. To allow the project sufficient breadth, it was decided that an active sonar would be attempted.

Traditional active sonars utilize well characterized devices and a procedure known as "beamforming" to emit highly directional pings. The time until the ping arrives back at the receiver can be used to determine the distance to the object. However, even underwater (where sound travels roughly five times as fast) these delays are rather large - on the order of tens of milliseconds. Having to try all the different angles to paint a full picture only multiplies this delay. Eventually, this translates over to slow update rate.

Our goal was to attempt to take care of this problem by reversing the scenario. Rather than sending out directional pulses, send out a omnidirectional ping (lessening stress on devices as well). Then, have an array of receivers instead of transmitters and piece together the distances to each angle - all from a single transmission. In this way, frame rates can increase by orders of magnitude. See the theory section for more on this rather watered down explanation.

A system was designed to fully exploit this approach and display it in a highly intuitive manner. Various processing and control modules fit into a fully pipelined system that determines where an object is in the field of view. A display unit (also pipelined) then takes this information and shows the object in a simplified frontal 3D view, or in a more traditional top view.

2 Processing - Theory

One has a very high degree of freedom when it comes to design of an active sonar system, when compared with passive sonar. The first decision that must be made is the one that distinguishes this design from most sonar systems: the use of multiple receivers instead of multiple transmitters.

2.1 Multiple Transmitters

Traditionally, multiple transmitters are placed in a linear array, not unlike that shown in Figure 1. If the transmitters are placed close enough to one another (half wavelength or less) one can send out highly directional pulses by properly choosing the phases going into each transmitter (see Figure 2 for an example angular distribution). This process is given the colorful name of "beamforming."

Since sound travels at a measurable speed and bounces off most objects fairly well, one can immediately formulate an algorithm for mapping the environment using beamforming. A sine wave pulse is sent at one direction - say 0 degrees - and one measures the time until a reflection is heard. This time is proportional to the distance to the closest object at that angle. After hearing this reflection, another sine pulse may be emitted in another direction, and the process may be repeated until all angles are covered.

Unfortunately, above ground sound travels at 340 m/s, giving a time lag of 6ms per

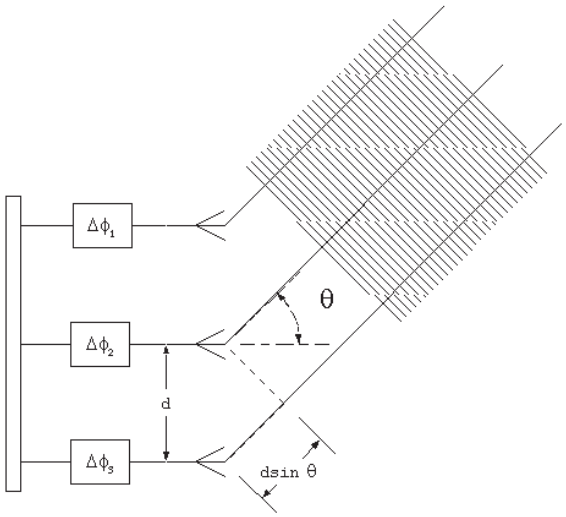

Figure 1: Linearly Phased Array for Input or Output

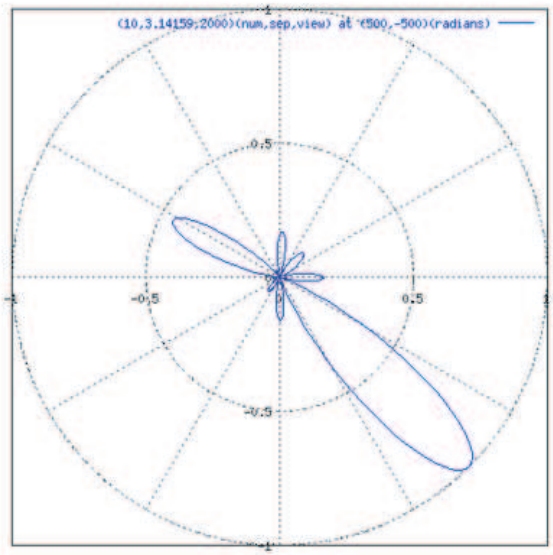

Figure 2: Sample Beamforming Distribution

meter per angle. If the closest object is 4 meters away, the minimum time for a full scan is 24ms times the number of trial angles. One cannot expect a frame rate faster than about a frame per second for any reasonable resolution. Not only is this extremely slow (the human eye detects flicker at 24fps) but unreliable (if objects move further away, the system can slow down significantly).

Additionally, the constriction of placing extremely well characterized transmitters within half a wavelength distance, raises the system cost. Rather than attempting to build a traditional system with these limitations, Project Tunafish decided it would be more interesting to address these difficulties by shifting responsibility to DSP and clever manipulation of receiver spacing.

2.2 Multiple Receivers

What if, instead of sending out a seperate transmission in each direction, one sent them all out at once in an omnidirectional transmission? With the use of multiple receivers, one could theoretically still extract the directional information from the signals.

Assume there are two objects at different distances and angles. If one sends an omnidirectional ping, seperate reflections will be heard for the two different objects on each of the microphones. If the ping is short enough in time, each microphone will show two seperate pulses (one for each object). For each of these pulses, there is a distinct lag between arrival time to the different mikes. See Figure 1 - which applies to both transmission and reception - again for an illustration of these time delays (dcosθ).

So, if these pulses are kept short and the objects do not overlap significantly distance wise, one can deduce the angle for either of the pulses after calculating its lagging between microphones. The distance is then obtained easily in the same way as in the multiple transmitter case (total delay since transmission).

How to calculate the lags that will give us the angles though? The answer lies in the magic of correlations. Say one has two sine waves over several periods at some phase to one another. If he shifts one relative to the other, multiplies the two, and integrates the function that results, this is called a cross-correlation. As it turns out, this function turns out a maximum value when the shift operation puts the sine waves in phase with one another.

A similar principle holds with our pulses. If one correlates the pulses between the microphones with one another, a maximum will occur at the shift that reverses their inherent phase shift (dcosθ). So one simply has to search for the maximum in these correlations between the different microphones, and she can trace back to the angles from there, right? Not entirely. There is a very important subtlety regarding this that actually has much broader impact on the process of beamforming in general. In the next section, we discuss both this problem and an elegant solution that was discovered in the course of the implementation.

2.3 Nonlinear Spacing

We observed before that for sine waves, correlation maxima occur at the phase-correcting lags. As it turns out, for signals that are finite length, this can be generalized using the fact that a signal's autocorrelation observes a maximum at zero lag. Assuming the pulsed sine waves observed at each of the microphones are simply shifted versions of one another (a seemingly reasonable assumption), the maximum of the correlation will occur not only when their sines are in phase, but also when the modulating pulses are made to line up exactly. So, we should be able to uniquely identify any sized lag between the receivers.

Only our reasonable assumption is not

at all reasonable for cheaper quality devices, like the ones we wished our system to work on, especially when one considers quantization noise. Large amplitude differences between the microphones are in fact quite present even for the most steady of objects. As it turns out, this completely destroys the possibility for detecting lags greater than the period of the sine wave in question - stuck at 40KHz by the resonant ultrasonic transducers. Given the centimeter diameter of the receivers we deal with, the maximum angular range possible is limited to 40 degrees. Not very much.

Amazingly, this is an identical problem to that faced by beamforming transmitters (see the multiple transmitter section). The requirement that the spacing between transmitters be less than half a wavelength is the exact mathematical restriction we face, and for the same reasons. If transmitters are spaced more than this distance apart, multiple lobes are created (similarly to how one set of lags can be repeated for multiple angles taken by an object in our case). A fundamental limit to angular range? Not so fast, buddy. We're from MIT, after all.

Thus far, everything we have mentioned can be done with 2 transmitters or 2 receivers. The extra devices have been used essentially for added noise protection. Why not use them to combat this problem? As is often the case, the answer is sometimes more apparent when one looks at the problem in a different light.

We recall that the lags are given by $d\cos\theta$. Plotting this over the angular range we are interested in, we have a cosine from 0 to $\frac{\pi}{2}$. Remembering however that we can only count on detecting *phase differences* and not the total lags, we recognize that lags off by 25 μs (the period of a 40 KHz wave) will yield identical phase differences. So let's say that angles A and B give the same lags for mics 1 and 2.

Now, suppose that mics 2 and 3 are spaced slightly further apart than 1 and 2. While it is true that more angles will now overlap (as the maximum lags have increased while the period is the same) we now have an interesting situation. Assuming that the 2-3 spacing is not a simple integral multiple of the 1-2 spacing, the angles that give the same 1-2 phase differences are guaranteed to give different 2-3 phase differences. The proof of this statement is left as an exercise for the reader. In short, we have decoupled the two sets of receiver's lags so as to widen the range.

The fact has been verified experimentally: in the original equally spaced setup, observers standing out of the central angular range "wrapped" around to the center screen - exactly what one would expect (verification is another exercise for the reader). In the second setup - decided upon after this analysis - the entire range of the transmitters (almost 180 degrees) was represented with no such wrapping. See Figure 3 for a picture of the final layout used. Notice the irregular spacing.

A similar exploitation may be performed in beamforming systems with many transmitters. If one spaces them greater than half a wavelength - one will certainly have multiple lobes. However, one can guarantee that only one of these lobes overlaps between all the transmitters by making them nonlinearly spaced. If there are enough transmitters, this essentially duplicates a "properly" beamformed signal without the need for a carefully fabricated array.

It should be emphasized that while other solutions exist to this problem - most notably the use of superior transducers and/or analog-digital converters - none can compete in terms of elegance, simplicity, and cost.

Figure 3: Receiver Array

3 Analog Interfacing

Before jumping into the implementation of the processor, it is important to understand the analog-digital hybrid aspects of the design. Several novel techniques were explored here as well, most notably non-linear time-variant reduction of the receiver's resonance quality by means of the FPGA. See Figure 4 for a schematic of the final circuitry for a single mic (we had 5).

3.1 Hardware Interface

The hardware part of our project is mostly responsible for collecting data from our five microphones. This data starts out as a 40 kHz waveform with an amplitude of about 10 mV. We amplify the signal about 200 times, and then convert it to digital data. Our algorithm works as well with sinusoids as it does with square waves, so using an 8 bit ADC would be wasteful since 2 bits would suffice. The quantization noise that is introduced by this reduction in resolution proved a severe problem early on, but was addressed by the nonlinearly spaced array.

It is also important for us to have a high sampling rate (about 1 MHz), so whatever ADC we used would have to be very high performance. We decided that the easiest way to meet these specifications with minimal waste would be to make our own ADC out of two comparators (for each microphone). The comparators would normally both output 0. One of them would output a 1 when the signal went above a certain threshold, and the other would output a 1 when the signal went below a certain threshold. The FPGA read the sensor data directly from the output of the comparators. We found that our custom ADC worked extremely well, and that the 2 bit data was simple to gather, store and process.

3.2 Non-linear Time-variant Q Reduction

In addition to working on the digital aspects of our project, we also invested time in improving the quality of our analog sensors. One of the largest problems was that a very short transmission pulse would result in a long response on our receivers. Specifically, our transmission lasts only for 6 cycles of 40 kHz, but a reflection from an object could cause more than 40 cycles of oscillation on the ultrasonic sensors. We believe that this extended response was caused by the high Q of our receivers (the Q was about 40 according to the datasheet).

The Q is a problem because it makes the pulse associated with a single echo (object) extremely long. This means that a second object would most likely have an overlapping pulse. Pulses from objects are much more difficult to process when they overlap, and since we ultimately did not solve the high Q problem, our system is limited to single object tracking. Ideally, the echoes on our receivers would be extremely short, thus making it unlikely that there would be pulse interference.

We attempted to solve the overlap problem by lowering the Q of our sensors. However, simply lowering the Q by putting a small resistor in parallel with a sensor causes several problems. Most noticeably, it decreases the amplitude of the receiver's response. A lower Q also implies a larger

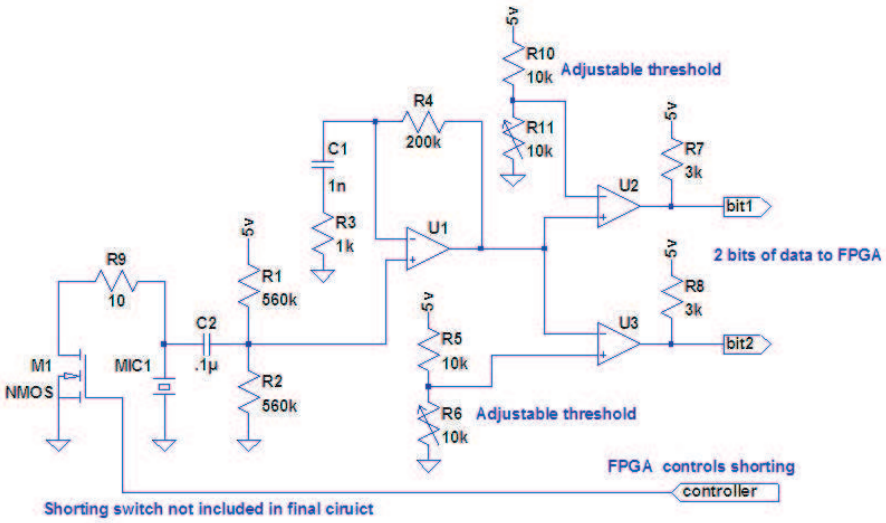

Figure 4: Analog Circuitry for 1 Mic

bandwidth, meaning that our "ultrasonic" microphone would become sensitive to lower frequency sound, thus increasing the noise in our system. To avoid some of these issues, we tried a non-linear approach.

Our method involves using the FPGA to count the number of oscillations on the receiver. Once the oscillation count reaches a critical number (after which more data would not help with signal processing), the FPGA sends a signal that causes a switch to close, effectively shorting the sensor output to ground for a brief time. The goal is to cause the ultrasonic resonator to lose all its energy, and go back to the normal, non-oscillating state. Thus, after a brief period of ringing, the microphone would be ready to accept new echoes without interference from the previous one.

Although the concept seems simple, building a switch that can cause a resonant tank circuit to stop oscillating without injecting some energy back in is fairly difficult. A typical discrete MOSFET switch stores enough energy in overlay capacitance [1] to restart oscillations once the switch is open again. We could have experimented more with smaller JFET devices (less capacitance), but we ran out of time.

[1] Thanks to Harry Lee for pointing this out

4 Processing - Implementation

The processing conveniently breaks up into three portions, tellingly named the pre-processer, the processor, and the post-processor. A few details about the system need mentioning however.

First of all, every portion of the system except for the post-processor is fully capable of tracking multiple objects. However, given analog difficulties with multipath elongation of pulses (see above), such a feature would only be reliable if the objects were more than half a meter apart. Considering the limited range of the system, it was decided that the post-processor would work with only one object at a time for reliability's sake.

Thanks to efficient implementation of the processor, as it turned out the transmission and reception (25ms) was typically the bottleneck in our system speed. Not a big problem however, since 40fps is well beyond the flicker fusion rate of the human eye, 24fps.

4.1 Pre-Processing

The pre-processor works in real time as data is collected. For the most part, its purpose is to seperate out pulses from different objects, and to identify whether these pulses are "valid" for processing. For instance, if the pulse from one microphone ends too close to the start of the same pulse in another microphone, the processor will spit out garbage if asked to process this region of the signals. Hence, each pulse is given a start time, end time, and five valid bits that identify which of the five microphones' signals are valid for that pulse. This information is written to a pipelined BRAM for use by the processor on the next state cycle.

4.2 Processing

Processing is the heart of the system. Given a start and end time for a series of pulses, it finds the pulse most likely to be corresponding to the object that is being tracked, and proceeds to find its angle. This is done by use of smaller lag-finding and correlation modules that are streamlined to perform the computations of interest. The max-lags received from the lag-finder module are thereafter passed to the post-processor for interpretation.

4.3 Post-Processing

Post processing serves two purposes. First of all, it converts the max-lags given it by the processor into distances and angles that the display module reads from memory. In doing so however, it also implements basic noise margins against sudden fluctuations.

While the distance measurements were usually quite reliable, two distinct methods were attempted at converting from the max-lags into angles. The first of these took the more intuitive look up table approach. Essentially, if the lags fell into a 4-space "box" (one range for each of the 4 lags) defined for an angle, the object was said to exist at that angle. If multiple angles claimed responsibility, the data was thrown away. The ranges of these 4-space boxes were determined through an efficient calibration procedure.

The second angle finding method found the "distance" to the characteristic lags associated with several different angles. The minimum of these was declared the angle of the object provided distance was below some noise threshold. We found the latter method to be far less reliable.

Finally, the user was given some ability to trade off speed for noise resistance. Essentially, an angle was forced to repeat itself a threshold number of times (as defined by the user) before it was declared as the ob-

ject's new angle. It was found that this simply slowed down the system, which normally converged quite fast to the correct location of an object that had just finished moving.

5 Module Summary

See Figure 6 for a block diagram of the non-display elements of the system, and Figure 5 for a block diagram of the memory control elements.

5.1 Double Buffering

The sonar architecture places a number of requirements on the way memory is organized. Most apparent is the fact that multiple independent modules need to access the same memories. For example, the data gathering module must write to the data memory, and processing module must read from it. In order to make this possible without introducing complexity into every module that uses BRAMs, we use a wrapper module that controls which module is currently "controlling" which memory. As a result, the data gathering block can be designed without considering the fact that the processor also needs to access the data memory.

The various modules in the sonar project have a very sequential nature, meaning that information naturally flows from one block to the next. However, sequential processing is unfavorable since each block must wait for the previous block to finish before it can start. For example, the processor cannot begin work until the data gatherer has finished collecting information. To avoid this waiting and thus improve the speed of our system, we chose a more parallel approach.

To achieve parallelism, there are two copies of every memory. So while the data gatherer is writing data to memory A, the processor is performing computation on the contents of memory B. When both are done, they switch memories. The control FSM decides when this switching takes place, making sure it happens only when all the blocks in the pipeline are done.

In order to make the design of each module in the pipeline simpler, the switching of memories is handled externally. As far as the module knows, it is always working with one BRAM. The details involved in both reading and writing to multiple memories must be hidden.

When a module thinks it is writing to memory, it is actually changing wires belonging to two BRAM wrappers. These changes only take effect if the block has "control" of that BRAM. The control FSM sets these control signals appropriately. For example, it makes sure that the data gatherer and the processor are never working with the same memory.

The control FSM also decides what happens when a module reads from memory with the help of a multiplexer. The mux makes sure that if a block is given control of a given memory, then it will read the output data from that memory.

Together, these memory control systems allow modules to share memories and switch between them without any special consideration. As a result, the entire system's speed is improved and design is simplified.

5.2 Controller

The controller orchestrates when every module in the sonar begins its work. The only inputs it requires are the done signals from each of the modules it controls. It must also carefully decide which module controls which memory, so that parallelism is possible without conflict.

5.3 Transmitter

The transmitter creates a 40 kHz signal, which is the resonant frequency of our ultrasonic transmitters. The 3.3v square wave output is passed to an amplifier, and raised to 20v, so that the transmitter's will have as long a range as possible.

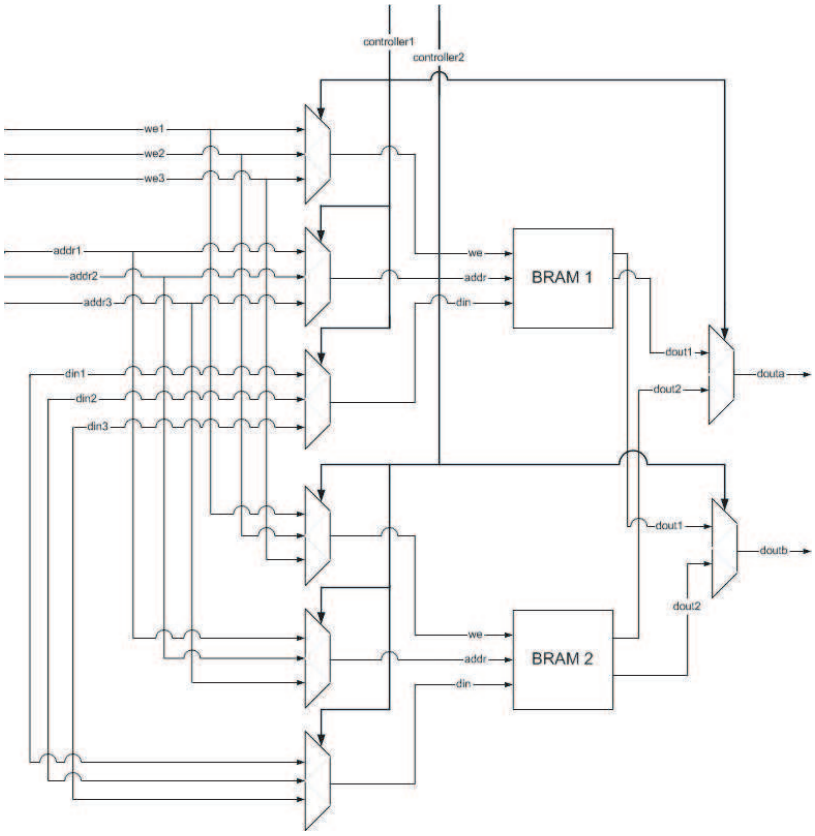

Figure 5: The Block RAM Wrapper

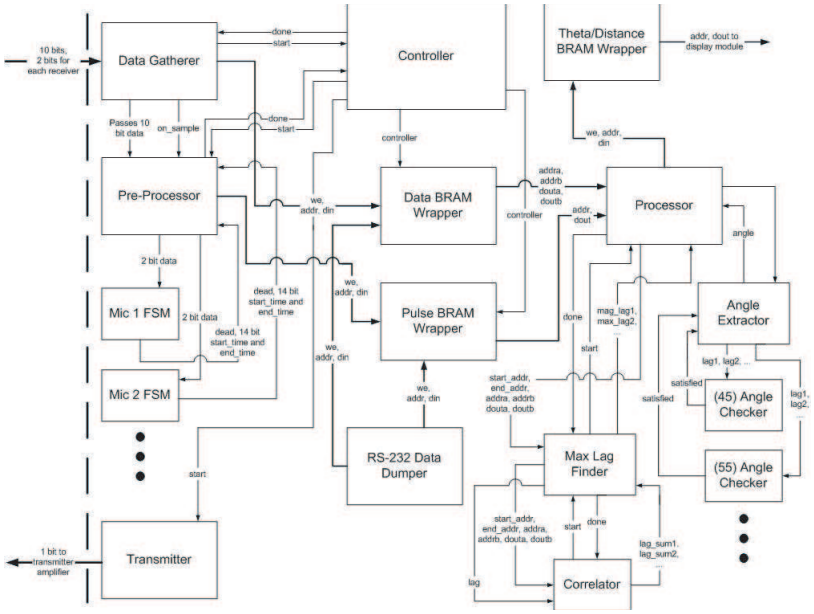

Figure 6: Main Block Diagram

5.4 Data Gatherer

This module samples data at a rate of 1 MHz, grabbing all the 2-bit sensors values at the same time. The data is written to memory, and also passed immediately to the pre-processor.

5.5 Mic FSM

There is one Mic FSM for each ultrasonic sensor. It examines the data from its microphone and finds out when a "pulse" starts and ends, where a pulse is a portion of the signal that is non-zero. The information about a start and end are passed to the pre-processor.

5.6 Pre-processor

The pre-processor's goal is to isolate an area of interest in the long stream of bits coming from the sensors. It does this operation while data is being gathered. By using information from the Mic FSMs, the pre-processor finds areas of the signals where as many of the sensors are active as possible (the most reliable data), and records the location of these areas in memory. Each of these "pulses" corresponds to one object in the environment.

5.7 Processor

The processor takes the data from the data gatherer, and the pulse information from the pre-processor and determines the location of an object in the field. The start location of a pulse determines the distance of the object. The processor performs a cross correlation and uses a lookup table like structure to determine the object's angle. The final angle/distance pairs are writen to memory for the display to use.

5.8 Max Lag Finder

This module makes the correlator module find the correlation for each possible lag. It keeps track of the correlation with the highest degree of overlap (lag_sum), and returns the corresponding lag to the processor.

5.9 Correlator

The correlator actually performs the cross correlation for a given lag on the data memory. It does all five correlations in parallel, using dual ported memory to read two values of the data at the same time. Thus, it can simultaneously do the multiply accumulate required for the cross correlation between sensor 1 and 2, 2 and 3, and so on.

5.10 Angle Extractor

Given the maximum lags, this module finds the most likely corresponding angle. It compares the lag values to those hard coded into a number of angle checking modules. If there is one match, the processor knows the object's angle. Multiple matches signify that the data is probably invalid.

5.11 Angle Checker

This module performs a simple check to see if the given lags fall within a certain range. The range is hard coded for every angle, with the values found empirically.

5.12 Dumper

This module takes data and sends it through the serial port using the RS-232 protocol. This entire part exists only for debugging; we use it to see what data is recorded from the sensors and what values our signal processing modules come up with. Thus, the module sends out the contents of various memories and also the values of a few registers. The dumper is activated by a button,

and normal operation is temporally stopped while it is sending information.

5.13 BRAM Wrapper

This module makes double buffering and memory sharing completely transparent to the other modules that use BRAMs.

5.14 Top, Front Conversion

The converters take the angle/distance information from the processor and convert it to pixel information using sine and cosine lookup tables.

5.15 Display

The display reads the pixel information and puts it onto the screen.

6 Testing and Debugging

The key to testing and debugging our system was the RS-232 module. In the first stage of our development, we used this module to send sensor data from the BRAM's on our lab kit to a computer. We developed an algorithm for locating objects from this real data. Had we tried to implement an algorithm without checking that the sensors behaved as we expected, the project would have never worked. Later on, as the signal processing core was coming together, we sent the intermediate results of our calculations to a computer along with the data being processed. In this way, we verified that the FPGA was producing the same results as our MATLAB code for the given set of data.

It was very easy to test bench certain modules, especially those that performed calculations. We simply put numbers in and made sure the right numbers came out. The memory wrapper was the only module that was difficult to test because it involved so many different pieces working together. In order to sort through the dozens of relevant signals, we made extensive use of the $display keyword to print out what values were being read from which memory. Since it took a long time for certain bugs to appear, it was much easier to examine a few printed statements than scrolling through many of long signals.

We found that the logic analyzer was unnecessary for testing and debugging in our case. The signals worked exactly as predicted by ModelSim.

7 Display Block

See Figure for a block diagram of the display unit.

7.1 Display Module

The display module reads 6-bit RGB values from a display RAM and converts them to 24-bit values ready to be used by the VGA. To cut down on the amount of memory needed, the RAM contains RGB data for every pixel of a 320x240 display even though the actual display is 640x480 pixels. Therefore, the main challenges in implementing the display module were making sure the RAM was read appropriately to display a 640x480 screen and reading the next address while converting data from the current one.

To get a 640x480 display from data meant for a 320x240, each pixel of the smaller display needs to be read four times. That is, one pixel of the smaller display becomes four pixels in the larger one. The RGB data for each pixel is stored in the RAM in the order the pixels are drawn on the screen-left to right, top to bottom-so the pixel_cnt and line_cnt signals from the VGA controller can be used to address the display RAM. If the LSB of each of those signals is not used, then the VGA controller will draw each row and each column of the smaller display twice on the larger display.

The data for a particular pixel in the 320x240 display is at the address that is the pixel's row number multiplied by the total number of columns plus the pixel's column number. For example, the address for the fifth pixel in the fourth row (which means the row number is 3 and the column number is 4 since the counts start at 0) is 3*320 + 4. To solve the problem of reading the next address of the RAM while displaying data at the current one, the address that is sent to the RAM is calculated using one more than the current value of pixel_cnt. When pixel_cnt becomes greater than the number of the last column on the screen, the address is held at that of the first pixel of the next line so that the RGB data for the first pixel of the next line is ready when pixel_cnt rolls over to zero.

The conversion of the 6-bit RGB value in the display RAM to a full 24-bit RGB value depends on the mode of display (front-view or top-view), which is selected by two switches on the lab kit. For the top view, all objects on the screen are the same color, so the nonzero RGB values in the RAM are all the same and are just concatenated with eighteen trailing zeros to form the VGA-ready RGB output. For the front view, the six bits in the display RAM become the two MSBs for the R, G, and B sections, and zeros are concatenated to fill in the rest of the 24 bits. For example, the value 6'b001001 stored in the display RAM would become the 24-bit RGB value 24'b000000001000000001000000.

7.2 Data Correction For Front-View Module

The t_d_conv_front.v module reads data from the theta-distance pairs RAM (written by the Control Module) and converts it to RGB data for the front-view display that is written to another RAM, which is later read by the Display Module. Each address in the theta-distance pairs RAM has a 16-bit value; the first 8 bits represent an angle, and the lower order 8 bits represent the distance at that angle. In the front view display, each object is represented by a vertical bar on the screen. The closer an object is to the transmitter/receiver array, the longer and wider its bar is, and the lighter its color is.

The front view conversion module is implemented as an FSM because of the multiple read and write operations needed to do the data conversion. A state transition diagram for this module can be found in Figure 8 . The FSM is idle (in the frconv_START state,

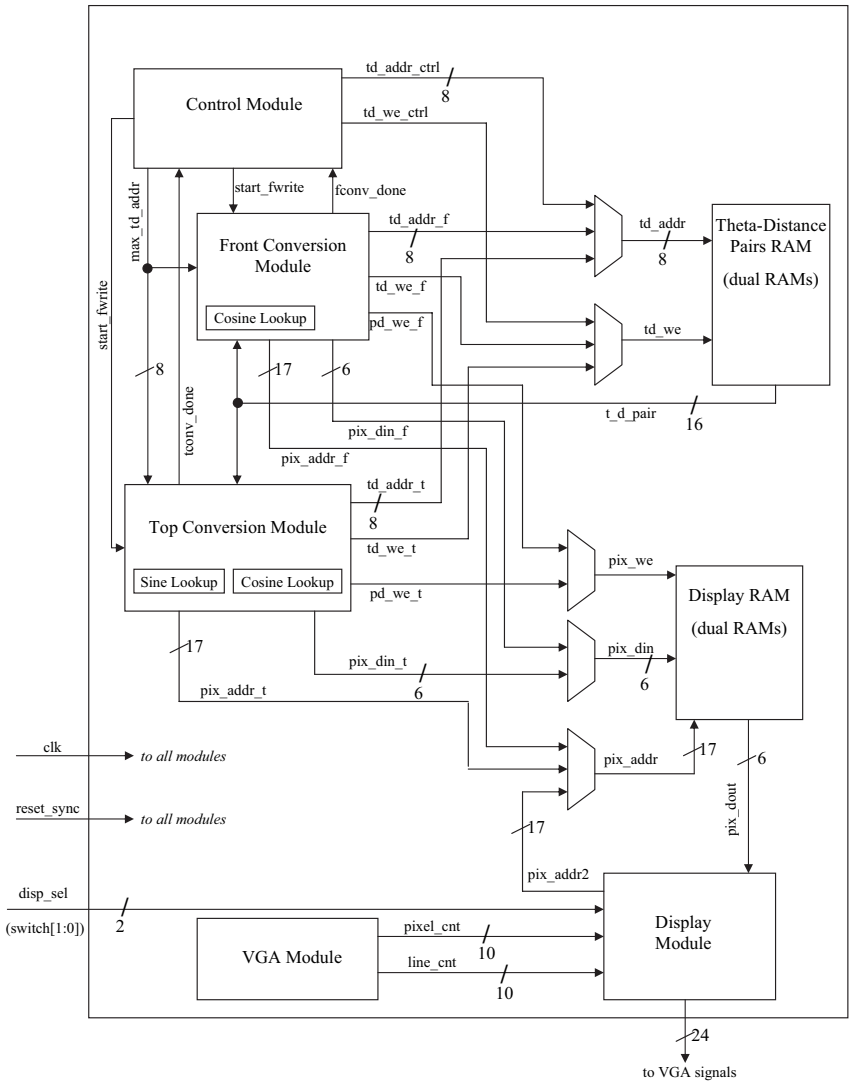

Figure 7: Display Block Diagram

which sets all outputs to zero) until an enable signal, called start_fwrite, from the control module signals the FSM to begin converting data. Before the front view conversion module can write new data to the display RAM, all of the previous RGB data must be cleared (i.e. set to zero). The FSM switches between the blank_mem state, which has a counter that addresses all locations of the display RAM one at a time and outputs a value of 0 to be written to all addresses, and the blank_mem_write state, which sets the write enable signal to the display RAM high. The write enable signal is only high when the FSM is in the blank_mem_write state to ensure that the address and data have settled before an attempt is made to write to the display RAM.

When all addresses in the display RAM have been cleared, the front view conversion FSM moves to the read_tdpair state, which outputs an address to be read from the theta-distance pairs RAM. If the last address in this RAM has been read, the FSM sends a done signal to the control module and becomes idle again (i.e. returns to the frconv_START state) on the next rising clock edge. If the last address has not been read, then the next theta-distance pair is ready on the next rising clock edge, when the state becomes trig_lookup. This state takes the top 8 bits of the theta-distance pair (which represent an angle) and sends them to the cosine lookup table. On the next rising clock edge, the FSM goes to the find_xcoord state, which calculates the x-coordinate of the object represented by the current theta-distance pair using the cosine data from the lookup table.

Usually, the conversion of polar coordinates (an angle and a distance) to a rectangular x-coordinate is done by simply multiplying the distance by the cosine of the angle. However, in binary, the conversion becomes more complicated because of the problem of how to represent non-integer values. In the cosine lookup table, values of the cosine function between -1 and 1 are mapped to 8-bit values between -64 and 64. This 8-bit value is first multiplied with its corresponding distance, which is represented by the 8 LSBs of the theta-distance pair data from the RAM. The resulting signal, called dcostheta, is an absolute value, so if the incoming cosine value (which is a two's complement number) is negative, it is converted to its magnitude (by flipping all bits and adding one) before being multiplied by the distance.

The display needs to render objects less than 90 (which have a positive cosine value) relative to the transmitter/receiver array on the right side of the screen, and objects greater than 90 (and less than 180, which have a negative cosine value) relative to the array on the left side of the screen. An object at 90 relative to the array (that is, directly in front of it) will be displayed in the middle of the screen. In a 320x240 display, the middle of the screen has an x-coordinate (column number) of 159, so the value of dcostheta is added to 159 if the cosine is positive, and subtracted from 159 if the cosine is negative. Only bits 13 through 7 of dcostheta are used in the calculation of the x-coordinate because all bits except the MSB of the cosine value are like the numbers after the decimal point of a floating-point number, and by leaving out these lower order bits we are essentially truncating dcostheta to get a whole number that can be represented appropriately in binary. We only take up to bit 13 of dcostheta because we do not want to add a number to 159 that will make the sum greater than 319 (the maximum column number of the small screen) or less than zero (the minimum column number); therefore we restrict the added or subtracted number to be seven bits.

After the x-coordinate is calculated, the FSM moves on to the prep_pixdata state, which determines the 6-bit RGB value to

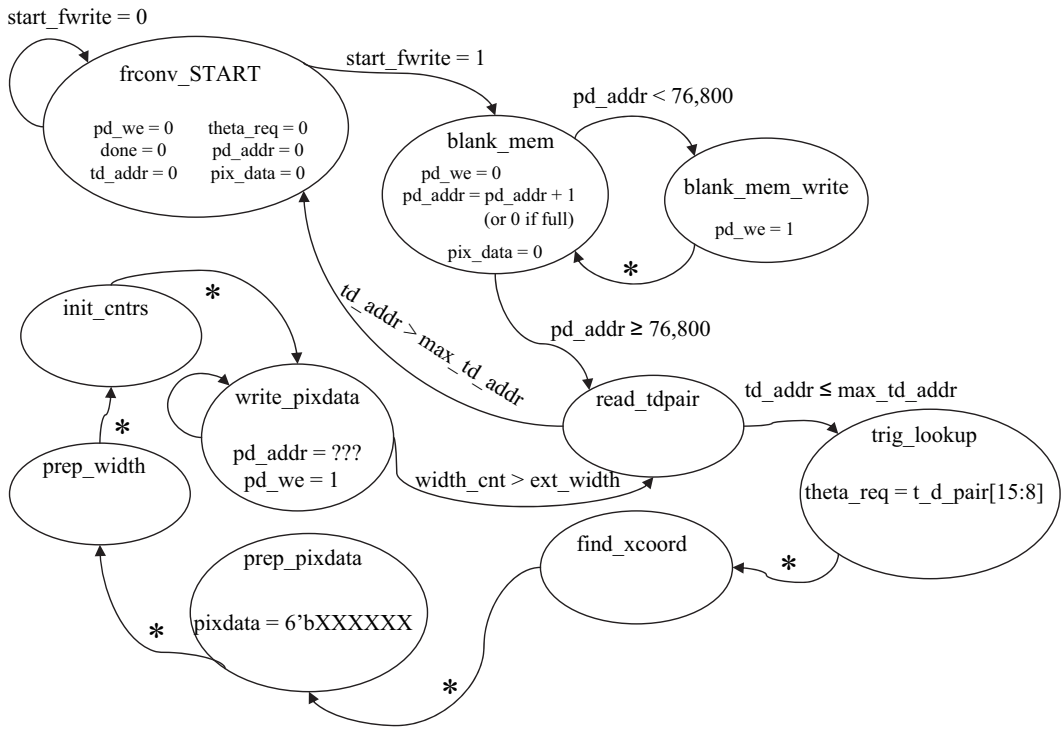

Figure 8: Front View FSM

be written to the display RAM based on the distance of the object from the transmitter/receiver array. As mentioned before, the closer an object is to the array, the darker the color of the bar that represents the object on the display. The length of the bar representing the object is also determined in this state.

The next state, prep_width, determines the width of the bar representing the object. Then the init_cntrs state sets to zero the values of the counters that will be keeping track of what address is sent to the display RAM for writing the RGB data for each pixel. Finally, the write_pixdata state writes RGB values to the appropriate addresses in the display RAM and returns to the FSM to the read_tdpair state when the necessary data has been stored. The FSM will continue its conversions or return to the idle state depending on if there are more theta-distance pairs to be read.

7.3 Data Conversion for Top View Module

The t_d_conv_top.v module reads data from the theta-distance pairs RAM and converts it to RGB data for the top-view display. In the top view display, each object is represented by a small square on the screen. The closer an object is to the transmitter/receiver array, the closer the square that represents it is to the bottom of the display screen. Objects at an angle less than 90 relative to the transmitter/receiver array are drawn on the right side of the screen, and objects at an angle greater than 90 (and less than 180) relative to the array are drawn on the left side of the screen. An object at 90 relative to the array (that is, directly in front of it) will be displayed in the middle of the screen.

The front view conversion module is also implemented as an FSM, and it is essentially the same as the FSM for the front view conversion through the trig_lookup state. A state transition diagram for this module can be found in Figure 9. In the prep_pixdata state for the top view conversion FSM, both the x- and y-coordinates of the object must be calculated. The x-coordinate is calculated the same way as it was in the front view conversion module. The y-coordinate calculation begins similarly. The angle of the object (represented by the eight MSBs of the theta-distance pair read from the RAM) is sent to a sine lookup table, and the 8-bit value that is returned is multiplied by the distance of the object to get the value of dsintheta. The dsintheta value is truncated like the dcostheta value was and then subtracted from 230 (since we want the closest objects to be near the bottom of the screen) to obtain the y-coordinate.

When the x- and y-coordinates have been calculated, the FSM transitions to the write_pixdata states, each of which writes RGB data into the display RAM for one pixel of the 3x3 square (in the 320x240 display; this gets blown up to a 6x6 square in the 640x480 display) that represents the object on the screen. After the last write state (write_pixdata9), the FSM transitions to the read_tdpair state, which will continue conversions or send the FSM back to its idle state, depending on if there are more theta-distance pairs to be read.

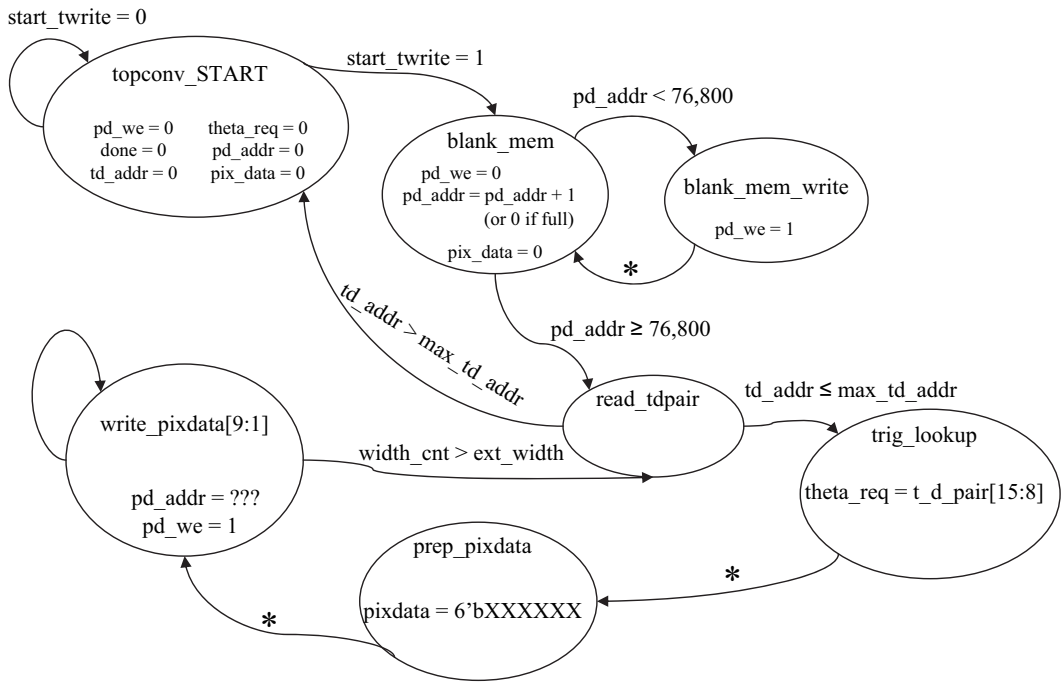

Figure 9: Top FSM

8 Summary

An original approach to active acoustic localization has been both proposed and implemented. The improvements to traditional sonar include multiple order-of-magnitude speedup, and lack of reliance on device quality. A technique in beamforming that allows the use of large devices and roughly placed phase arrays was also discovered along the way.

By means of a fully pipelined architecture that "never rests," the speed improvement lent by the algorithm was exploited completely, making the pulse delay the limiting factor in speed.

A display module reads data provided it by the processing and data gathering elements (pipelined) and displays its view of the field from a top view and a front view. The latter demonstrates perspective, changing the size and width of the object as it moves closer and further away.

The system works well within a short distance range, after which point limitations of the transmitters attenuate the signal to below the noise floor.

Post-processing intentionally limited the system to single object tracking to deal with noise issues stemming from multiple pathways of reflection.

Mathematical and implementation details have intentionally been left out of this paper for the sake of succinctness. The authors may be reached for any comments, suggestions, or questions.