Wind Sensor: CDR Presentation Advisor: Dr. Kuh

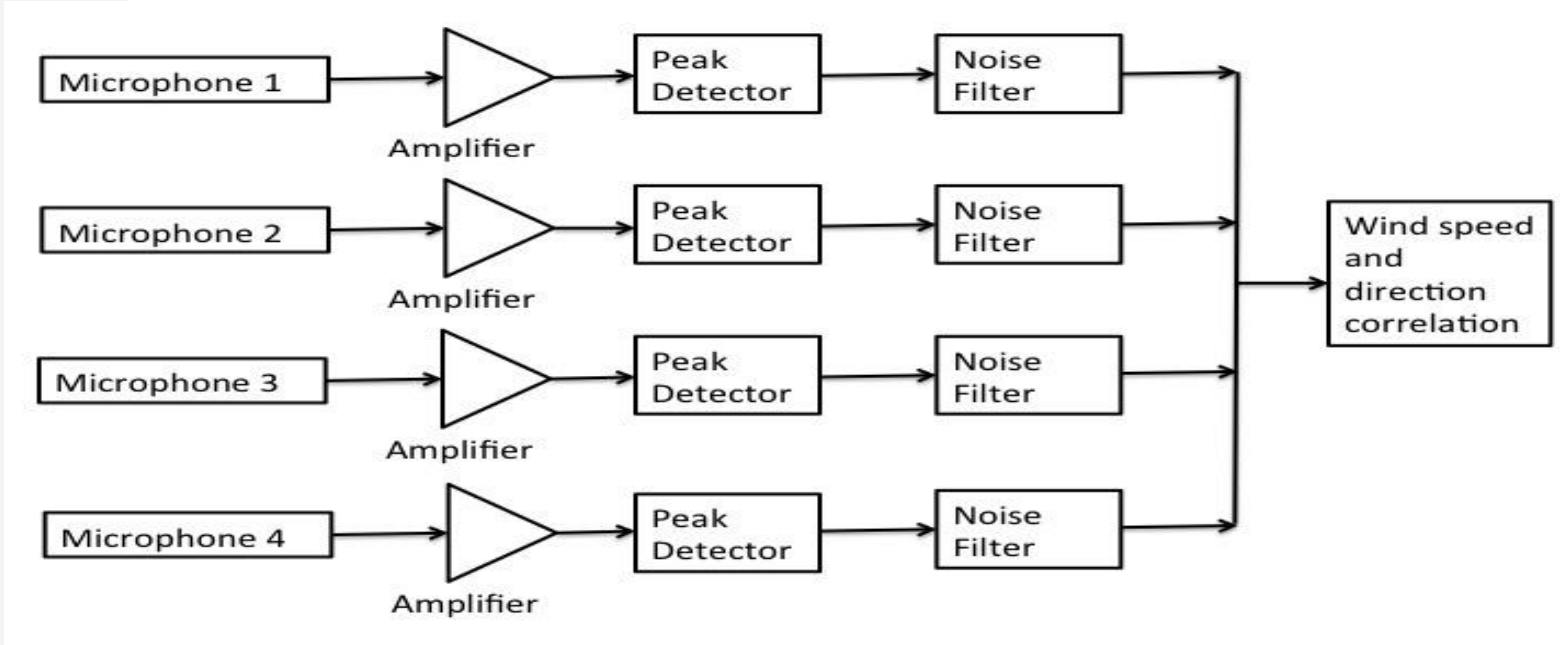
Creighton Chan • Jerry Wu • Scott Nishihara

Summary

- **Acoustic Wind Sensor**
- **Ultrasonic Wind Sensor**
- **Questions**

Acoustic Wind Sensor

Block Diagram

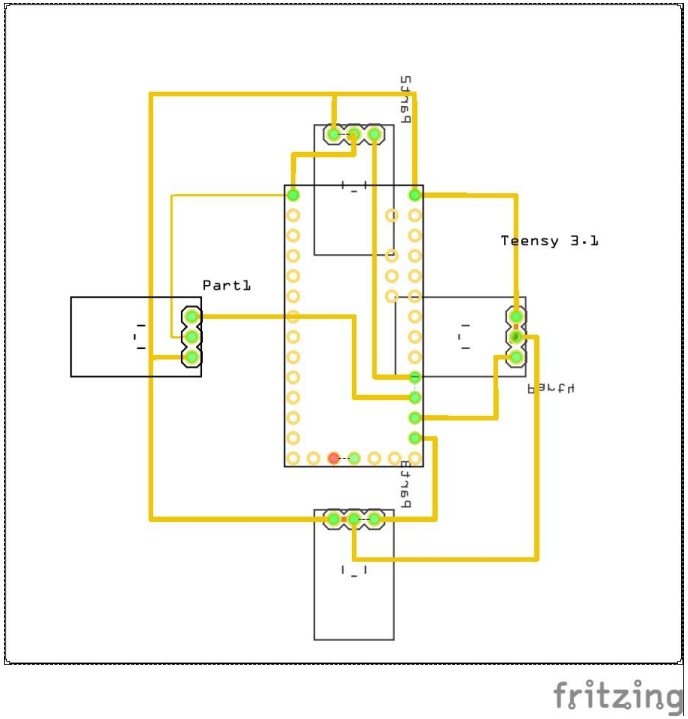


Block diagram for the acoustic wind sensor

Progress

- Frequency-domain analysis of raw data
- Replicated Andy’s test results with one microphone through a series of tests
- Created schematic of PCB design

PCB Design

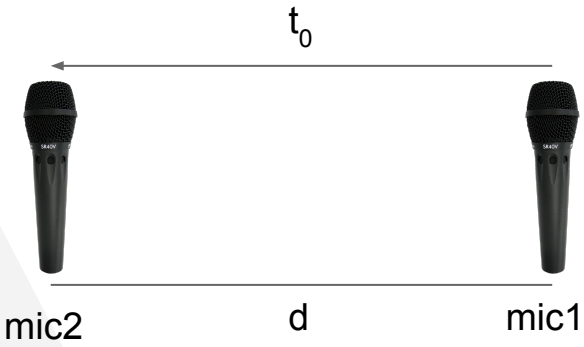


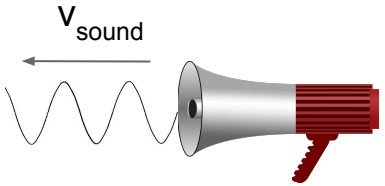
Outstanding Tasks

- Complete verification of Andy’s results with a four microphone set-up
- Mill four-microphone PCB
- Begin design of housing

Ultrasonic Wind Sensor

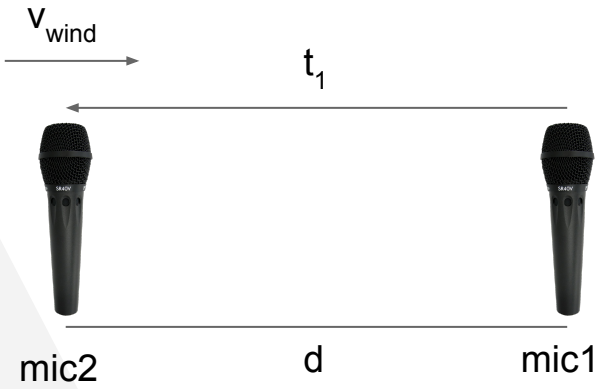
The Idea

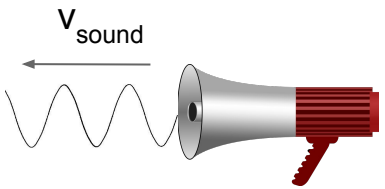




1. With no wind present, propagation time from mic1 to mic2 is t_0 seconds

The Idea





1. With no wind present, propagation time from mic1 to mic2 is t_0 seconds
2. Wind **blowing against** the propagation of sound will decrease sound wave velocity and **increase propagation time**, $t_1 > t_0$

The Idea

1. With no wind present, propagation time from mic1 to mic2 is t_0 seconds
2. Wind **blowing against** the propagation of sound will decrease sound wave velocity and **increase propagation time**, $t_1 > t_0$
3. Wind **blowing in the same direction** as the sound wave will increase sound wave velocity and **decrease propagation time**, $t_2 < t_0$

Is this difference measurable?

Variables:

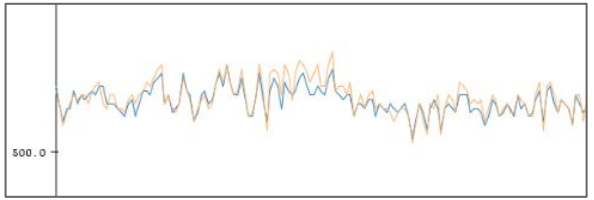
1. Speed of sound at room temperature, v_{sound} = 346 m/s
2. Distance between microphones, d = 8 cm
3. Minimum wind speed that can be detected, v_{wind} = 1 mph

Answer:

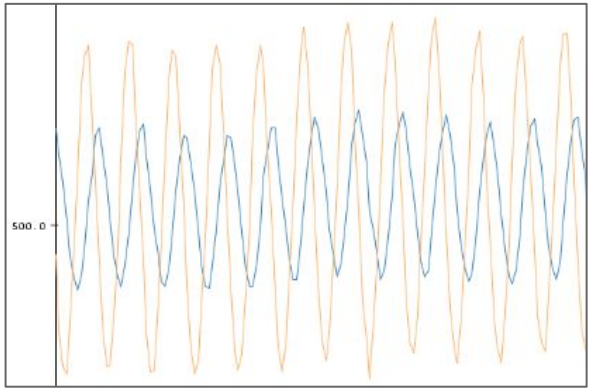
Yes, though our system needs to detect changes **as small as 0.3 us.**

Attempt #1 - Overview

1. Used the analogRead() function to read in analog values and Timer1 to measure the propagation delay
    a. Values are centered at 512 and the waveform is a sine wave with amplitudes ranging from 0 to 1023
    b. Compared the read values with predefined upper and lower thresholds
    c. When the value is greater than the upper threshold or less than the lower threshold, will start (if mic1) the timer or stop (if mic2) the timer
    d. Print out the measured time to see if it is consistent



Ambient noise



Generated sound wave

Attempt #1 - Source Code

```
#include <TimerOne.h>

const int pin1 = 14;
const int pin2 = 15;
const int val1 = 512;
const int val2 = 512;
const int lower = 450;
const int upper = 550;
const int resetPin = 12;
int state = 0;
volatile int counter = 0; // volatile type for sharing

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(resetPin, INPUT_PULLUP);

  Timer1.initialize(1);
  Timer1.attachInterrupt(count);
  Timer1.stop();
  Serial.begin(9600);
}
```

1/2

```
void loop() {
  switch (state) {
    case 0: // wait for first microphone
      val1 = analogRead(pin1);
      state += (val1 < lower || val1 > upper);
      break;
    case 1:
      Timer1.restart(); // start counting
      state += 1;
    case 2: // wait for second microphone
      val2 = analogRead(pin2);
      state += (val2 < lower || val2 > upper);
      break;
    case 3:
      Timer1.stop();
      Serial.printf("%d us has elapsed\n", counter);

      digitalWrite(LED_BUILTIN, HIGH);
      while (digitalRead(resetPin) == HIGH);
      digitalWrite(LED_BUILTIN, LOW);

      state = 0;
      counter = 0;
      break;
  }
}

void count() {
  counter++;
}
```

2/2

Attempt #1 - Results

1. Measured results were not consistent
2. We think it has to deal with Timer1 not being fast enough
   a. Although, we haven't tried changing the prescaler on Timer1 which should give us a resolution of 10.5 ns according to Arduino's website

> The accuracy of the timer depends on your processor speed and the frequency. Timer1's clock speed is defined by setting the prescaler, or divisor. This prescale can be set to 1, 8, 64, 256 or 1024.

- Time per Tick = (Prescale)*(1/Frequency)

*Although possible, the next attempt can achieve similar things

Attempt #2 - Overview

1. Similar to attempt #1, used analogRead() to read in values but instead of using a timer to measure the pro, kept track of clock cycle #
   a. Teensy 3.2 has a clock frequency of 96 MHz, so by counting clock cycles, we can achieve a resolution of 1/(96 MHz) = 10.42 ns
   b. This is achieved using the following code snippet
```
ARM_DEMCR |= ARM_DEMCR_TRCENA;
ARM_DWT_CTRL |= ARM_DWT_CTRL_CYCCNTENA;
```
   Current clock cycle is stored in `ARM_DWT_CYCCNT;`

   For convenience, we assign assign this to a macro `#define CYCLE ARM_DWT_CYCCNT`
   c. Again, read values are compared to predefined upper and lower thresholds
   d. Save the clock cycle # when mic1 is triggered and when mic2 is triggered, the propagation delay (in clock cycles) is the difference

Attempt #2 - Source Code

```
#define CYCLE ARM_DWT_CYCCNT

const int pin1 = 14;
const int pin2 = 15;
const int resetPin = 12;
const int upperThresh1 = 800;
const int lowerThresh1 = 280;
const int upperThresh2 = 610;
const int lowerThresh2 = 380;

//820 264
//633 408

void setup() {
    // Set up counting clock cycles
    ARM_DEMCR |= ARM_DEMCR_TRCENA;
    ARM_DWT_CTRL |= ARM_DWT_CTRL_CYCCNTENA;

    // Pin configurations
    pinMode(pin1, INPUT);
    pinMode(pin2, INPUT);
    pinMode(LED_BUILTIN, OUTPUT);
    pinMode(resetPin, INPUT_PULLUP);

    Serial.begin(9600);
}
```

1/2

```
void loop() {
    unsigned short val1;
    unsigned short val2;
    unsigned long startCycle;
    unsigned long endCycle;

    // Wait for mic1 to hear
    do {
        val1 = analogRead(pin1);
    } while (val1 > lowerThresh1 && val1 < upperThresh1);
    startCycle = CYCLE;

    Serial.printf("Triggered\n");

    // Wait for mic2 to hear
    do {
        val2 = analogRead(pin2);
    } while (val2 > lowerThresh2 && val2 < upperThresh2);
    endCycle = CYCLE;

    // Wait until reset
    Serial.printf("Val1: %d, Val2: %d\n", val1, val2);
    Serial.printf("Diff: %lu, Start: %lu, End: %lu\n\n",
        endCycle - startCycle, startCycle, endCycle);
    digitalWrite(LED_BUILTIN, HIGH);
    while (digitalRead(resetPin) == HIGH);
    Serial.printf("Starting in 1 second\n");
    delay(1000);
    delay(800);
    digitalWrite(LED_BUILTIN, LOW);
}
```

2/2

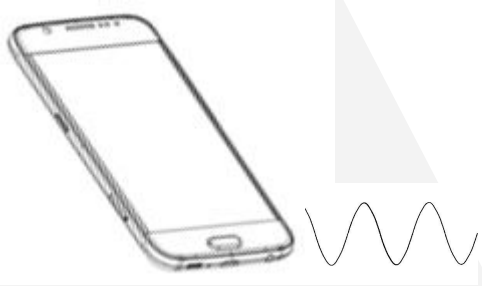
Attempt #2 - Results

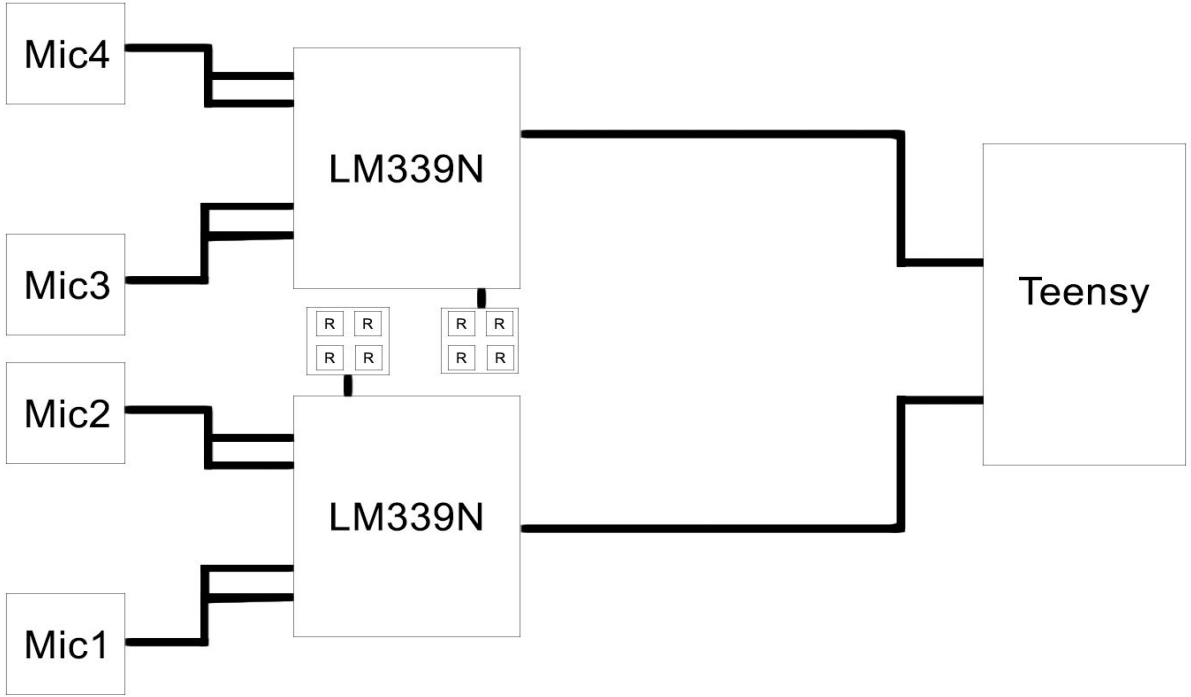
1. Although an improvement over attempt #1, results still varied with an error of about 2000 clock cycles (~20 us)
2. We believe that using analogRead() to read values is too slow since one ADC takes 870 clock cycles (~9 us)

Attempt #3 - Overview

1. Instead of using analogRead() to read values and then compare them to an upper and lower threshold, implement this in hardware so the trigger is faster
    a. We can use an interrupt to call a function to save the current clock cycle, but the interrupt needs to be triggered on a FALLING or RISING signal (0->1 or 1->0)
    b. We need some way to turn this analog system (0-1023) into a discrete system so the output = {1 if not triggered, 0 if triggered}
    c. To accomplish this, we used a LM339N Quad Voltage Comparator and used voltage dividers to get the threshold voltages

Attempt #3 - Block Diagram





Block diagram for the ultrasonic wind sensor

Attempt #3 - Source Code

```
#define CYCLE ARM_DWT_CYCCNT

const int pin1 = 14;
const int pin2 = 15;
const int resetPin = 12;
volatile int startCycle;
volatile int endCycle;

void setup() {
    // Set up counting clock cycles
    ARM_DEMCR |= ARM_DEMCR_TRCENA;
    ARM_DWT_CTRL |= ARM_DWT_CTRL_CYCCNTENA;

    // Pin configurations
    pinMode(pin1, INPUT_PULLUP);
    pinMode(pin2, INPUT_PULLUP);
    pinMode(LED_BUILTIN, OUTPUT);
    attachInterrupt(pin1, handler1, FALLING);
    pinMode(resetPin, INPUT_PULLUP);

    Serial.begin(9600);
}
```

1/3

```
void handler1() {
    #ifdef DEBUG
    Serial.printf("handler1\n");
    #endif

    startCycle = CYCLE;
    detachInterrupt(pin1);
    attachInterrupt(pin2, handler2, FALLING);
}

void handler2() {
    #ifdef DEBUG
    Serial.printf("handler2\n");
    #endif

    endCycle = CYCLE;
    detachInterrupt(pin2);
    waitForReset();
}
```

2/3

```
void waitForReset() {
    #ifdef DEBUG
    Serial.printf("wait for reset\n");
    #endif

    delayMicroseconds(100000);
    digitalWrite(LED_BUILTIN, HIGH);
    Serial.printf("%d\n", endCycle - startCycle);
    while (digitalRead(resetPin) == HIGH);
    digitalWrite(LED_BUILTIN, LOW);

    #ifdef DEBUG
    Serial.printf("Restarting...\r\n");
    #endif

    attachInterrupt(pin1, handler1, FALLING);
}
```

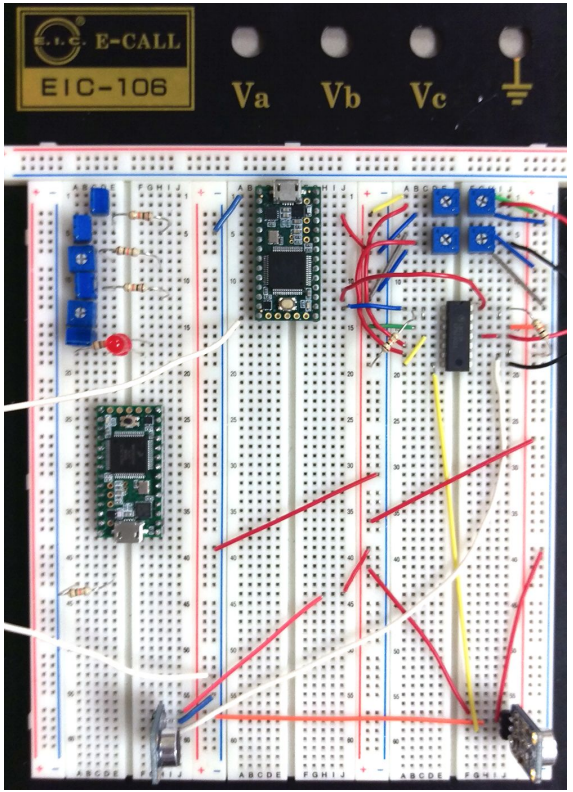
3/3

Attempt #3 - Results

1. An improvement over attempt #2, but still getting varied results with an error of about 1000 clock cycles (~10 us)
2. We think this might be due to the relatively low frequency of the signal that we're using (1 kHz), the period is relatively long so subtle differences in the waveform voltage can result in a large time difference
3. Using a signal with a much higher frequency can help to mitigate this as there will be a shorter time between values

Outstanding Tasks

1. Improve attempt #3 by using a transmitter and receiver that can operate at higher frequencies
2. Continue to look at other implementations online and consult with Noah Hafner
3. Integrate a hardware filter to filter out all other frequencies (for outdoor use)
4. Work on using multiple sensors to measure wind direction
5. Transfer our design to a PCB
6. Work on 396 paper



The end.

Any questions?

Backup Slides

Calculating Timing Resolution Needed

1. With **no wind present**, propagation time is d/v_{sound} = **232.55814 us**
2. With wind **blowing against sound propagation**, time is $d/(v_{sound} - v_{wind})$ = **232.86075 us**
3. With wind **blowing in direction of sound wave**, time is $d/(v_{sound} + v_{wind})$ = **232.256314 us**
4. Thus, **minimum timing resolution** needed is about **0.30261 us**

Undefined Zone