
# Using a smartphone's 3.5mm audio jack for data exchange



http://wolfpaulus.com


MAJA DALEY
03
10:03 AM
The Cupcake Cart
$22.00
Cupcakes (One Dozen)
Clear
Swipe card to charge
1
2
3
6


intuit.
card reader by ROAM DATA™


VISA
intuit.
by ROAM DATA™
1:30 PM
Charge
Swipe Card
intuit


MasterCard
intuit
GoPayment
3G
9:41 AM
Send Receipt
Charge of $88.00 Successful
Send receipt to customer:
mike_wagner@email.com
415-555-0125


INTUIT
GoPayment

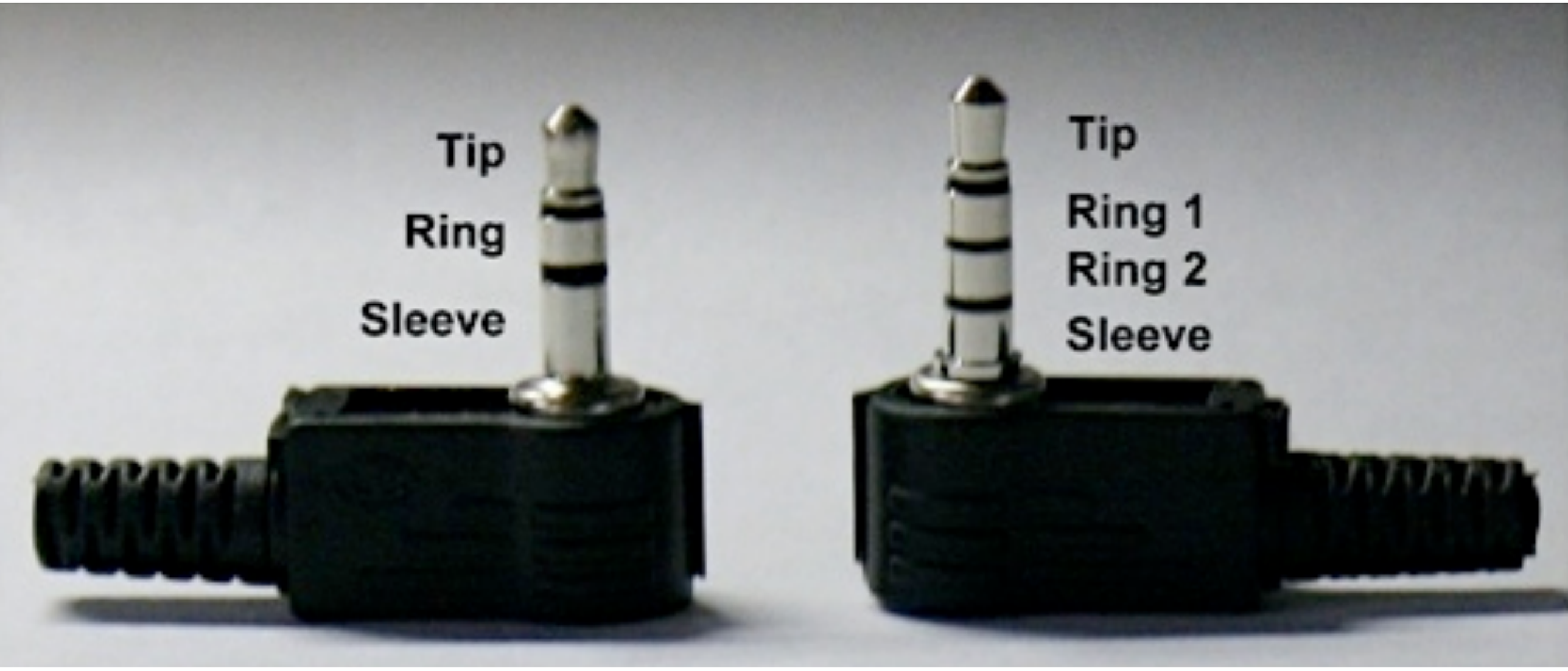

10:00 AM
AT&T 3G
PayPal
Here
1000 0000 0000
Expiration date
02/15
VISA
JOHN SMITH
Member since 1998

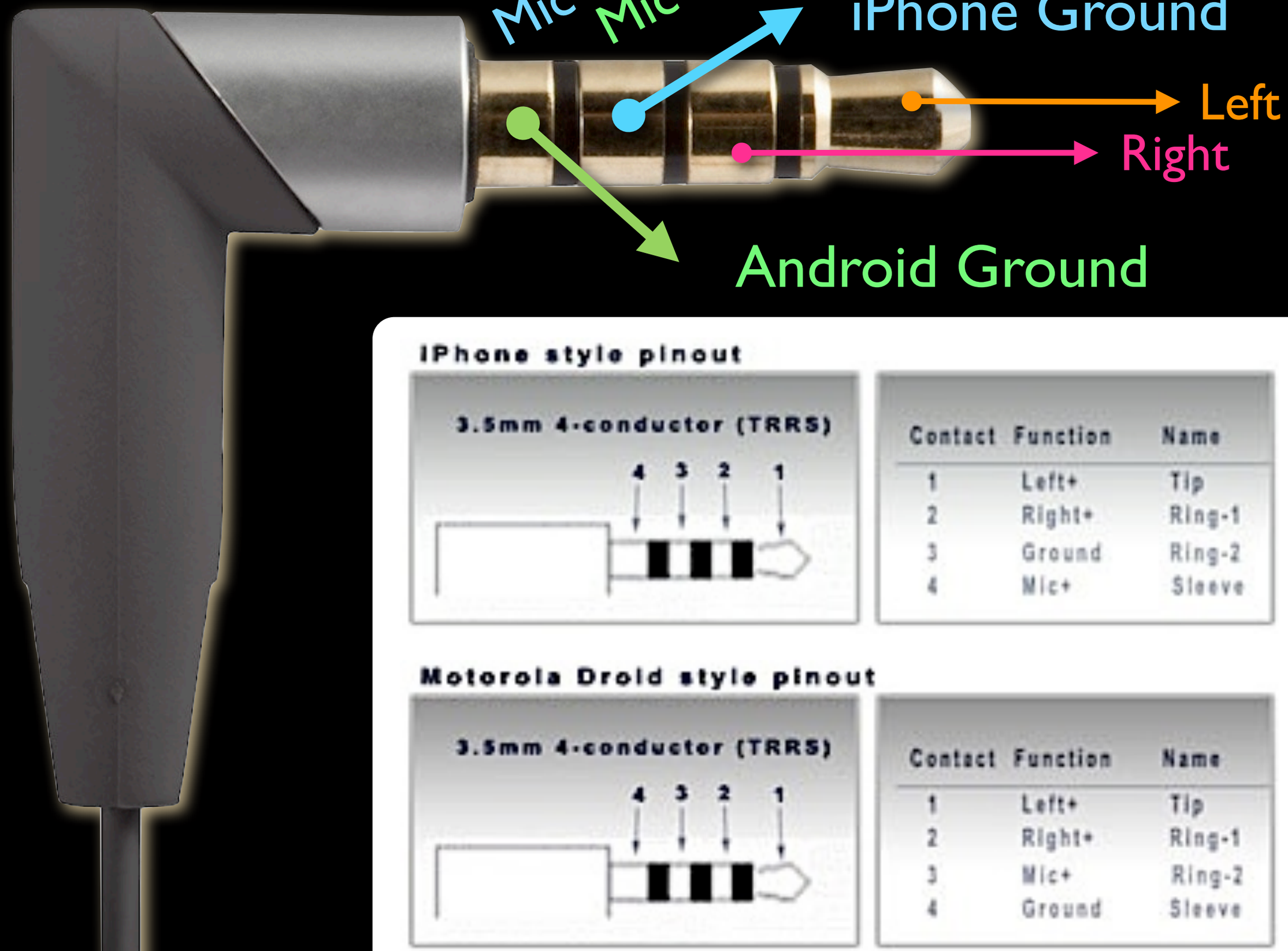

PayPal
Here


PayPal
Here


6:33 hrs
1,325 steps
Me
Team
Feed
Challenge
Profile
JAWBONE

Tip Ring Ring Sleeve


Tip
Ring
Sleeve
Tip
Ring 1
Ring 2
Sleeve




**iPhone style pinout**

3.5mm 4-conductor (TRRS)




| Contact | Function | Name |
|---|---|---|
| 1 | Left+ | Tip |
| 2 | Right+ | Ring-1 |
| 3 | Ground | Ring-2 |
| 4 | Mic+ | Sleeve |

**Motorola Droid style pinout**

3.5mm 4-conductor (TRRS)

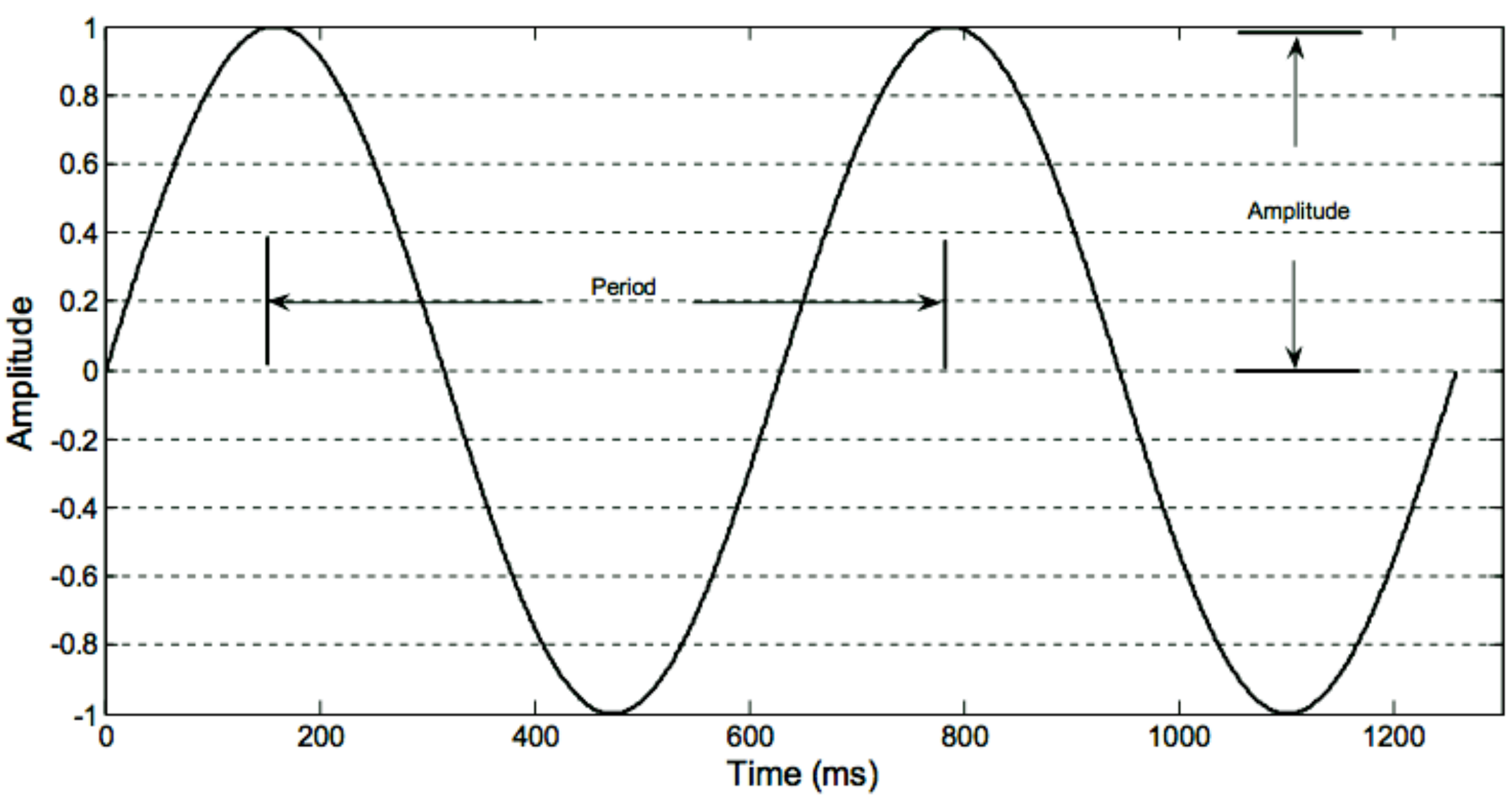



| Contact | Function | Name |
|---|---|---|
| 1 | Left+ | Tip |
| 2 | Right+ | Ring-1 |
| 3 | Mic+ | Ring-2 |
| 4 | Ground | Sleeve |

**Learn** about the **theory** behind audio encoding / decoding

**Build** an **embedded system** that captures, encodes, and transmits sensor data

**Build** an **mobile smart phone app** that receives, decode, and displays the sensor data



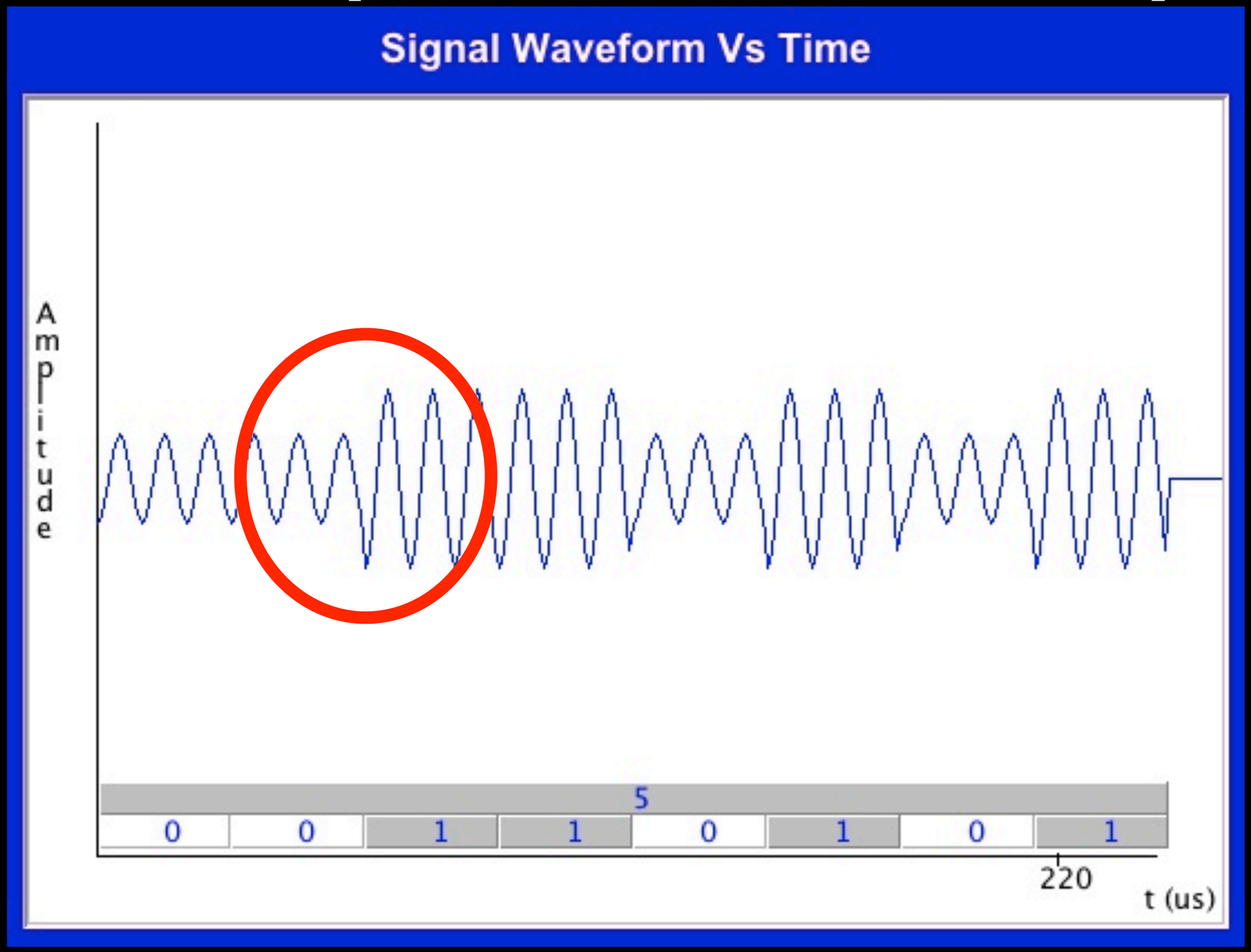

$$f(t) = A\cos\left(2\pi ft + \theta\right)$$

Amplitude

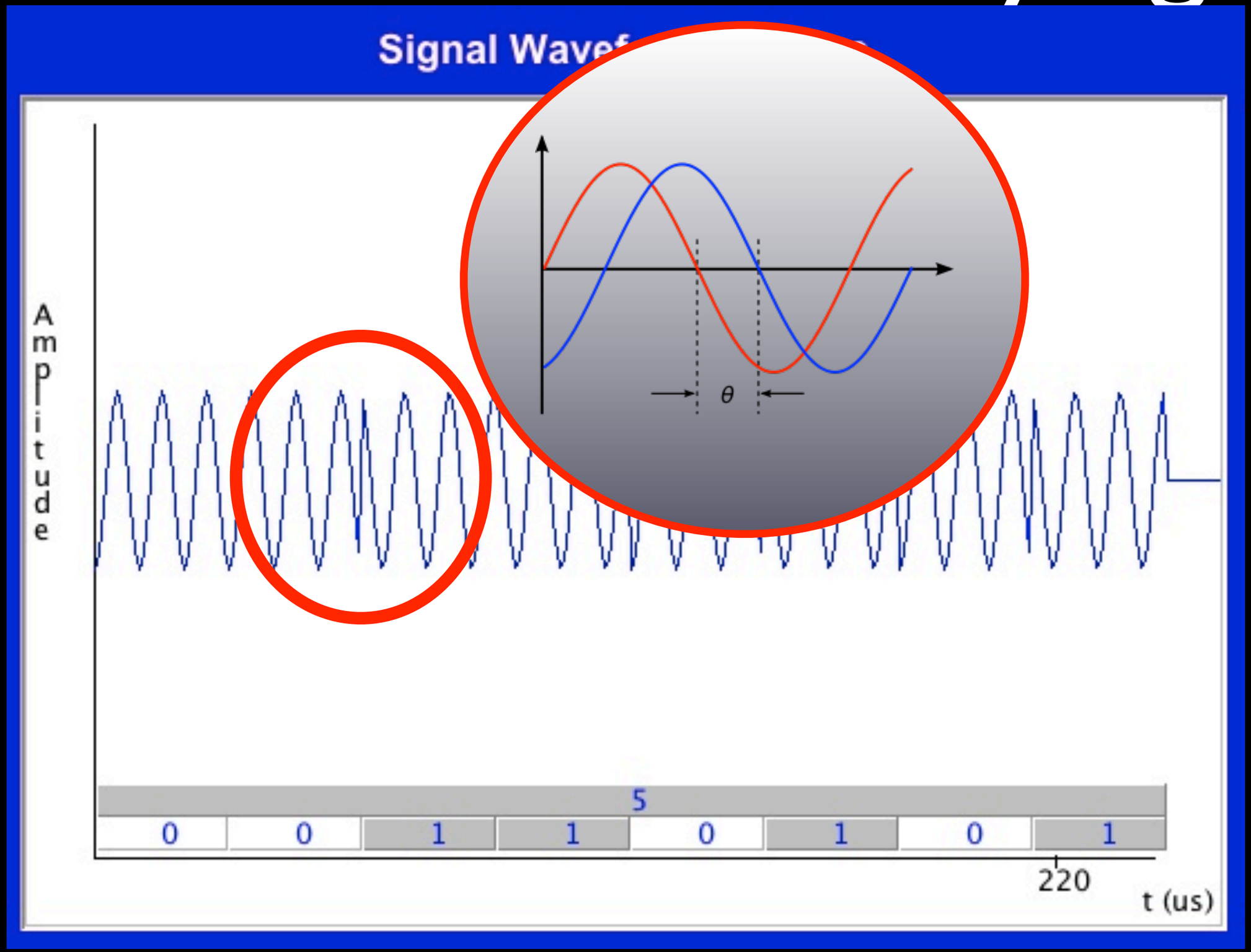
Period (time required for one complete cycle)

Frequency (number of cycles per second)

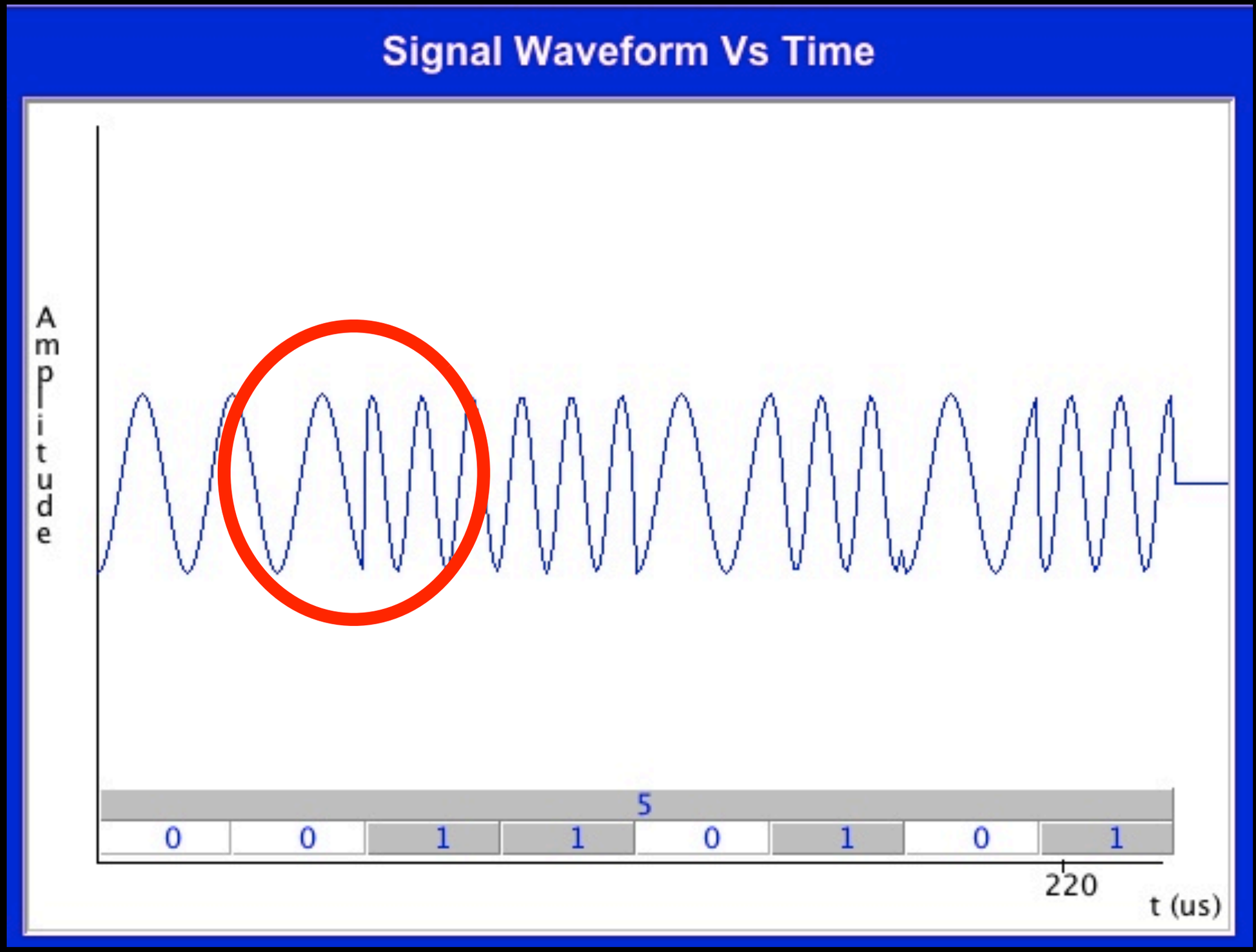
# ASK Amplitude Shift Keying



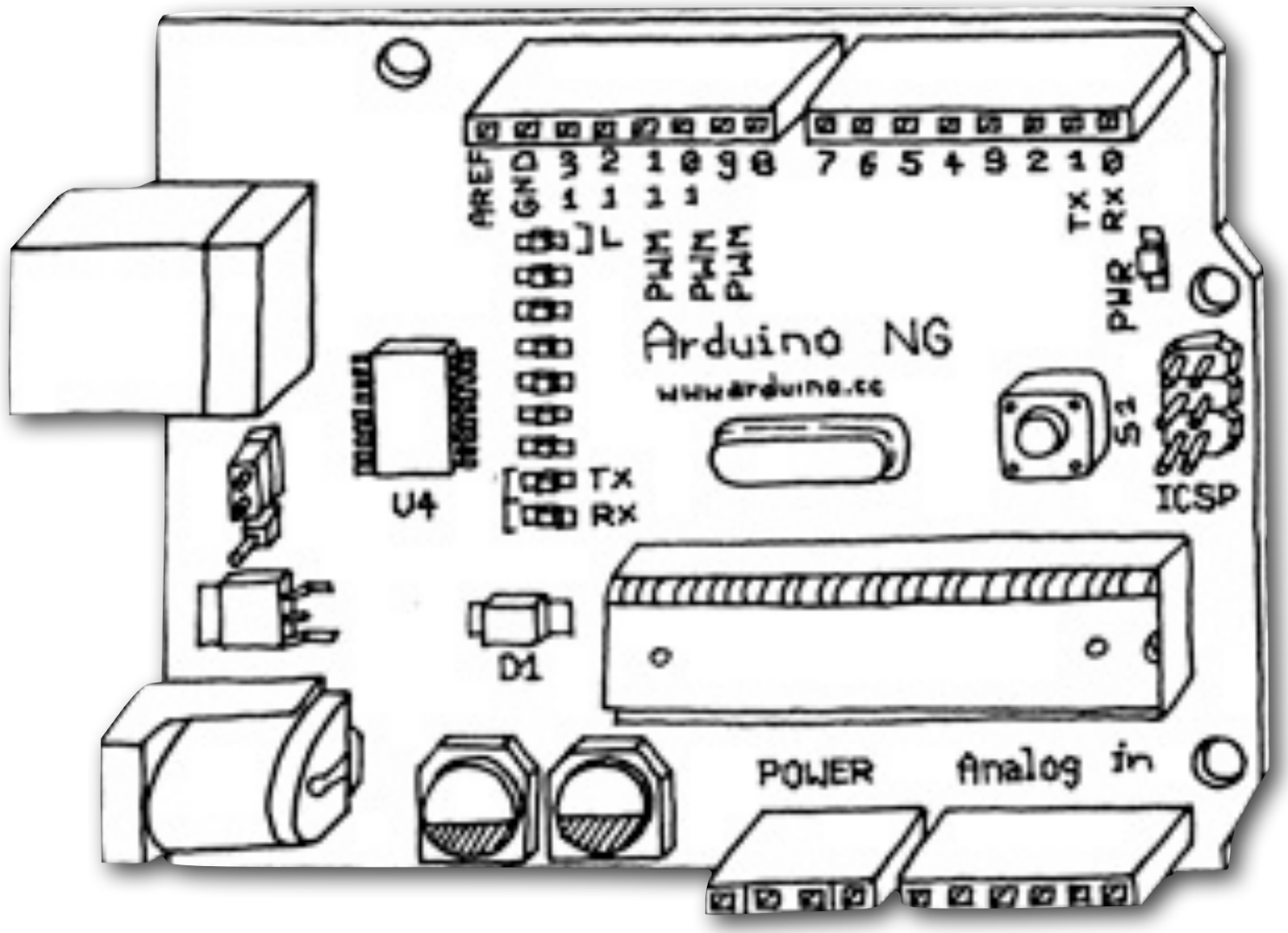
# PSK Phase Shift Keying



# FSK Frequency Shift Keying

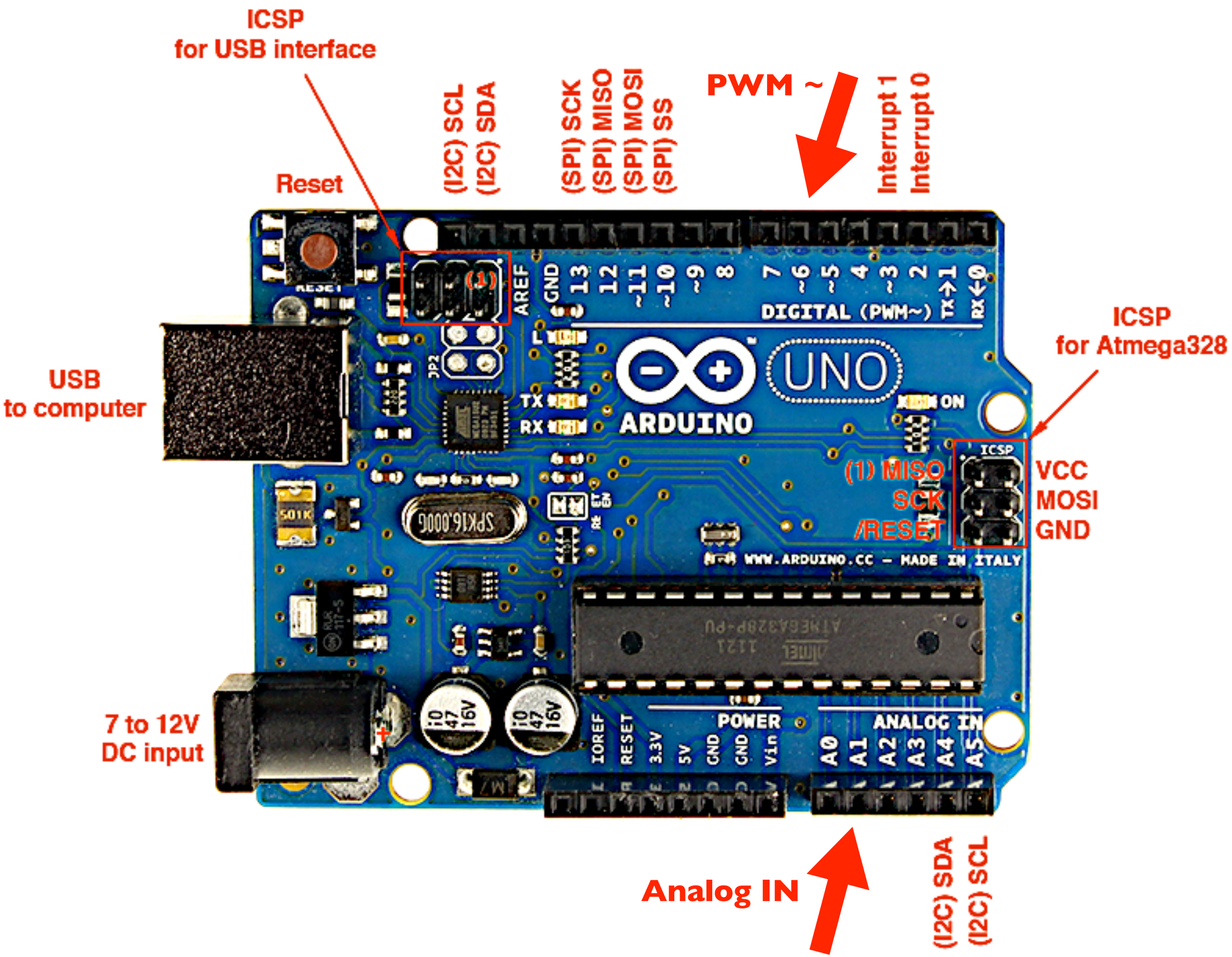


# The Sender


AREF
GND
13
12
11
10
9
8
7
6
5
4
3
2
1
0
PWM
PWM
PWM
TX
RX
PWR
Arduino NG
www.arduino.cc
S1
ICSP
U4
TX
RX
D1
POWER
Analog in

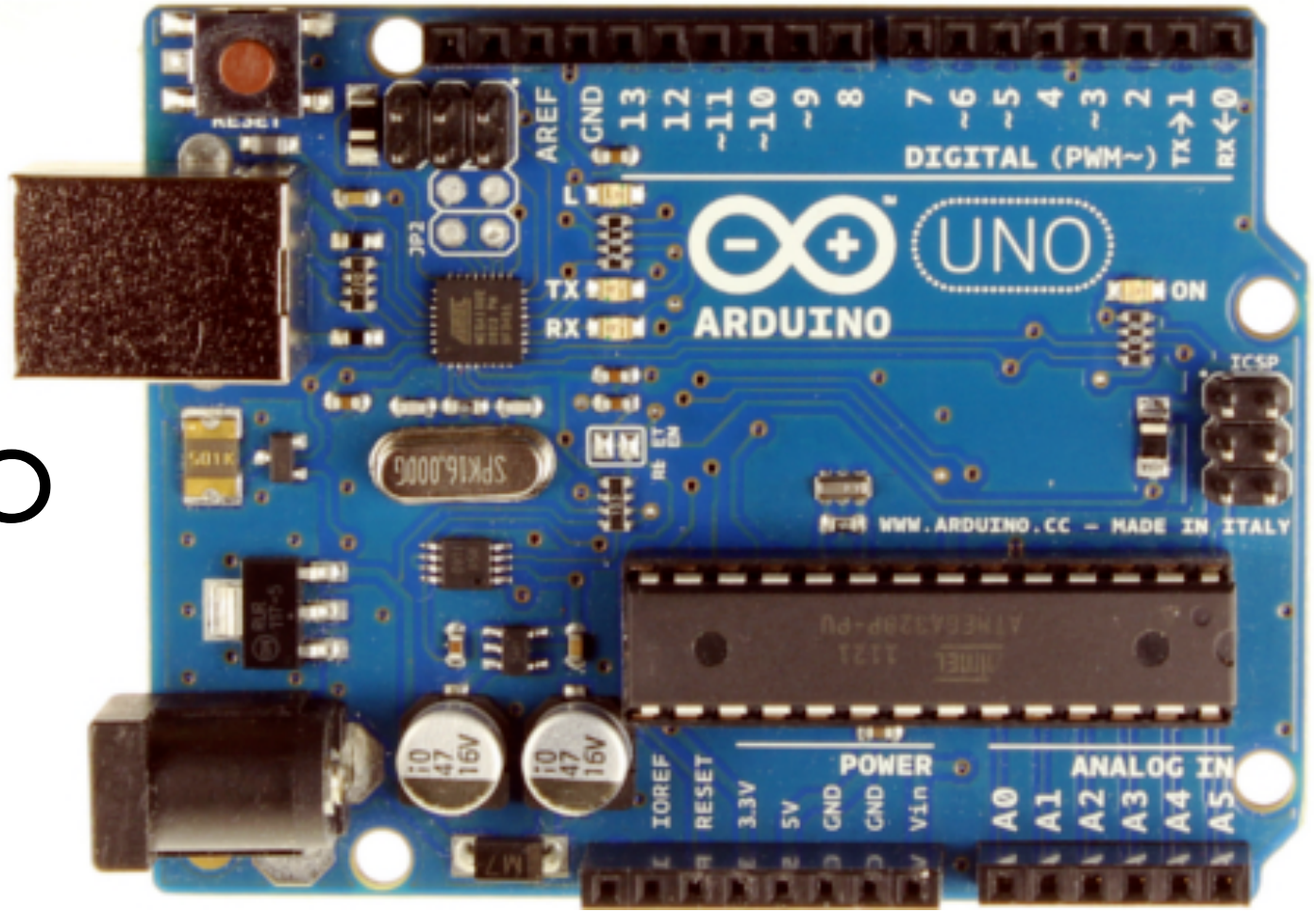
The 1st Arduino was produced in Jan. 2005 by **David Cuartielles** and **Massimo Banzi,** teachers of Physical Computing and Interaction Design at the Interaction Design Institute, Milan Italy.


MADE
IN ITALY
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
16.000Y
RESET-EN
RESET
ICSP
POWER
ANALOG IN
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5
WWW.ARDUINO.CC

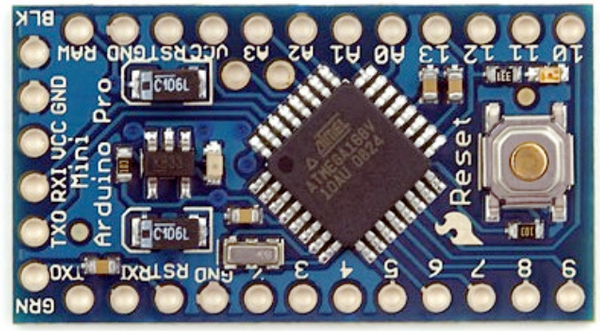

ICSP
for USB interface
Reset
(I2C) SCL
(I2C) SDA
(SPI) SCK
(SPI) MISO
(SPI) MOSI
(SPI) SS
PWM ~
Interrupt 1
Interrupt 0
USB
to computer
ICSP
for Atmega328
(1) MISO
SCK
/RESET
VCC
MOSI
GND
7 to 12V
DC input
Analog IN
(I2C) SDA
(I2C) SCL
DIGITAL (PWM~)
POWER
ANALOG IN
ARDUINO
UNO
WWW.ARDUINO.CC – MADE IN ITALY

**USB**

UNO




# Arduino Pro Mini

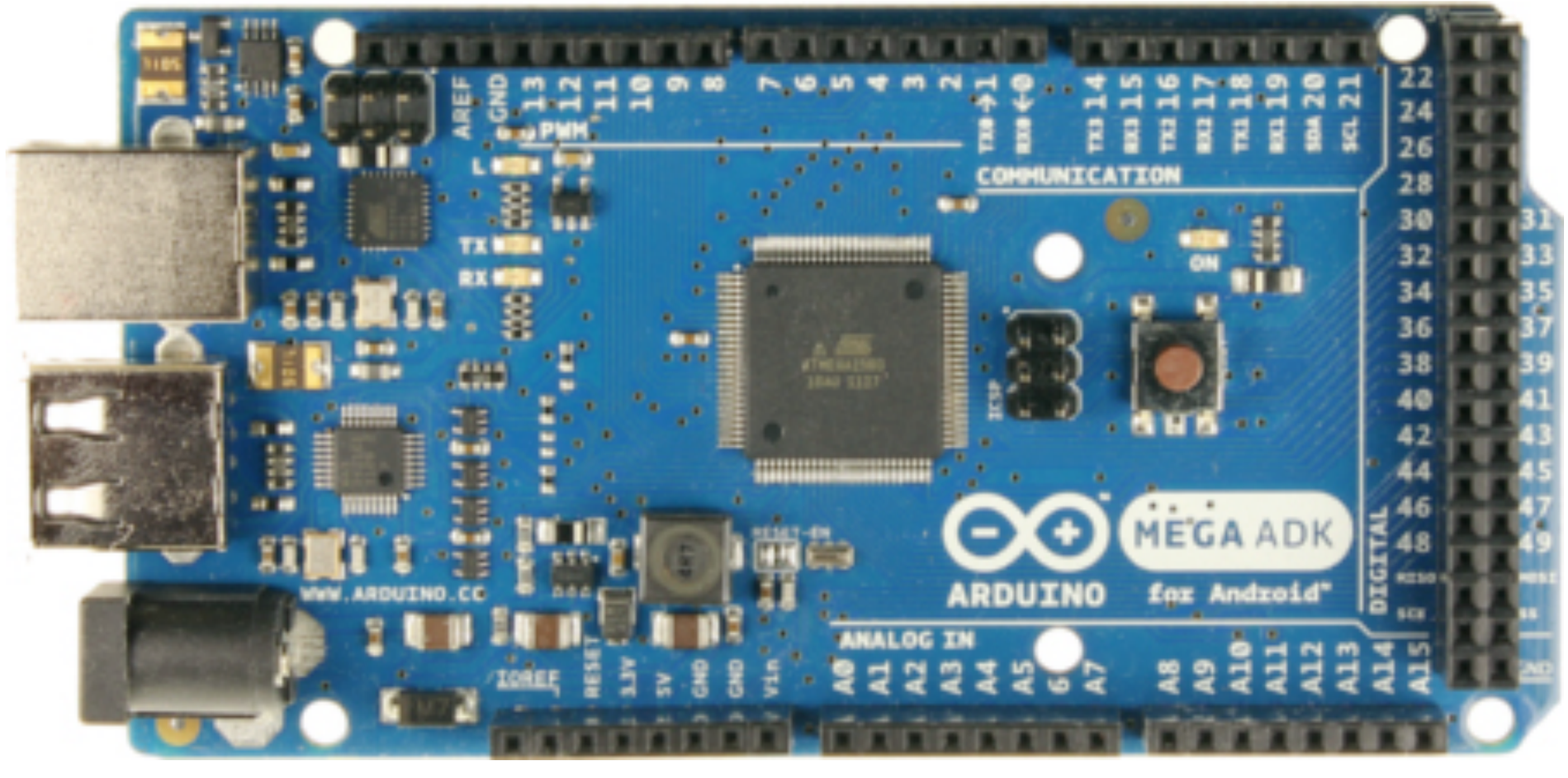



**No USB**

Two Version: 3.3V and 8 MHz or 5V and 16 MHz.

**USB**

***u*USB**



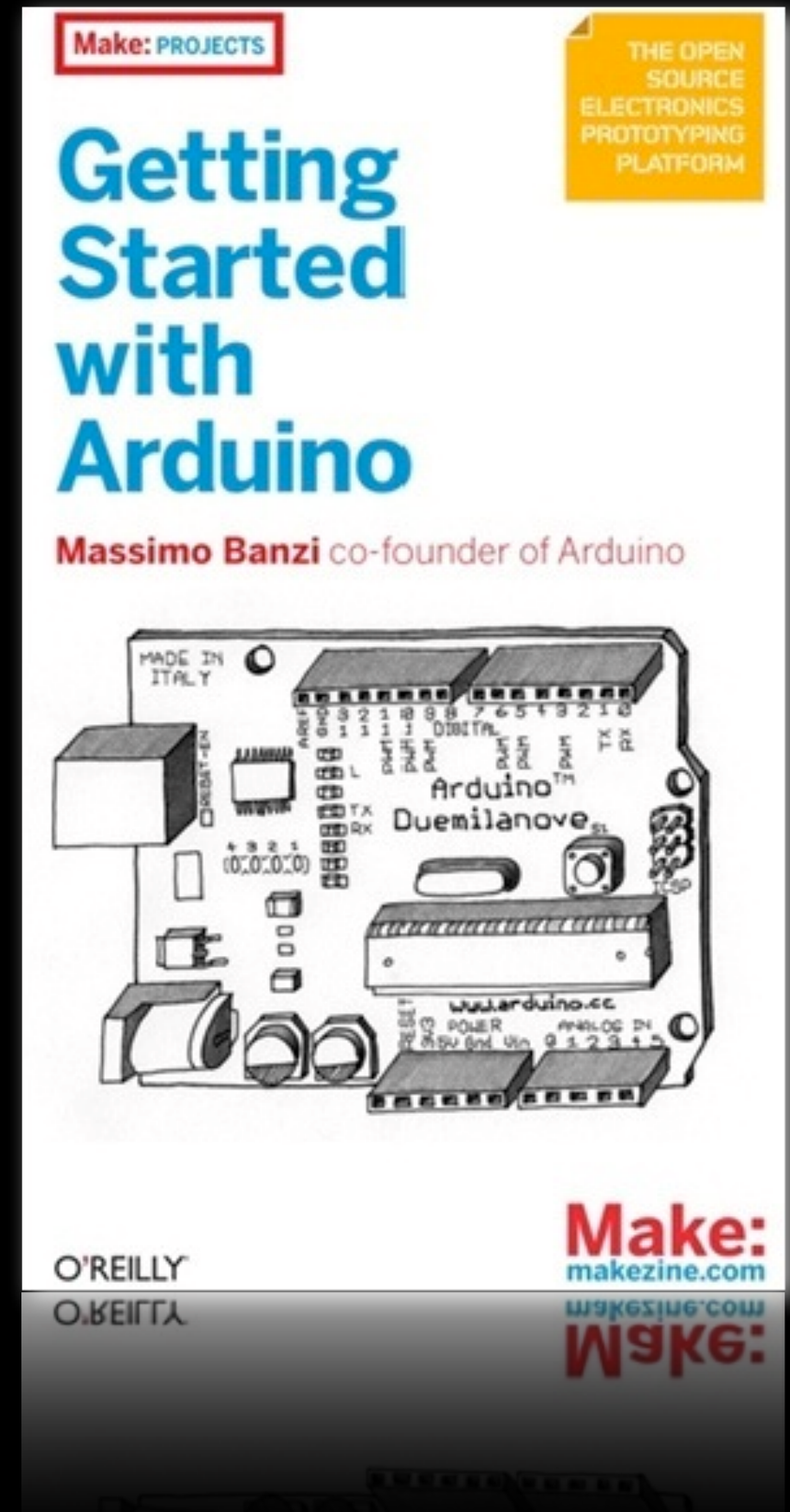

MEGA
ADK

# Arduino on the Web

http://arduino.cc

Getting Started with Arduino (Make: Projects)

by Massimo Banzi,
Co-founder of the Arduino Project



# Programming Arduino

Getting Started with Sketches

by Simon Monk


*Verify*

Checks your code for errors.

*Upload*

Compiles your code and uploads it to th

Note: If you are using an external progr
this icon. The text will change to "Uploa

*New*

Creates a new sketch.

*Open*

Presents a menu of all the sketches in y

Note: due to a bug in Java, this menu d
**Sketchbook** menu instead.

*Save*

Saves your sketch.

*Serial Monitor*

Opens the **serial monitor**.

## Initialization

```
const int ledPin   = 13;
const int outPin   = 11;
const int freq     = 2000;
const int duration = 1000;
const int pause    = 500;
```

## Setup

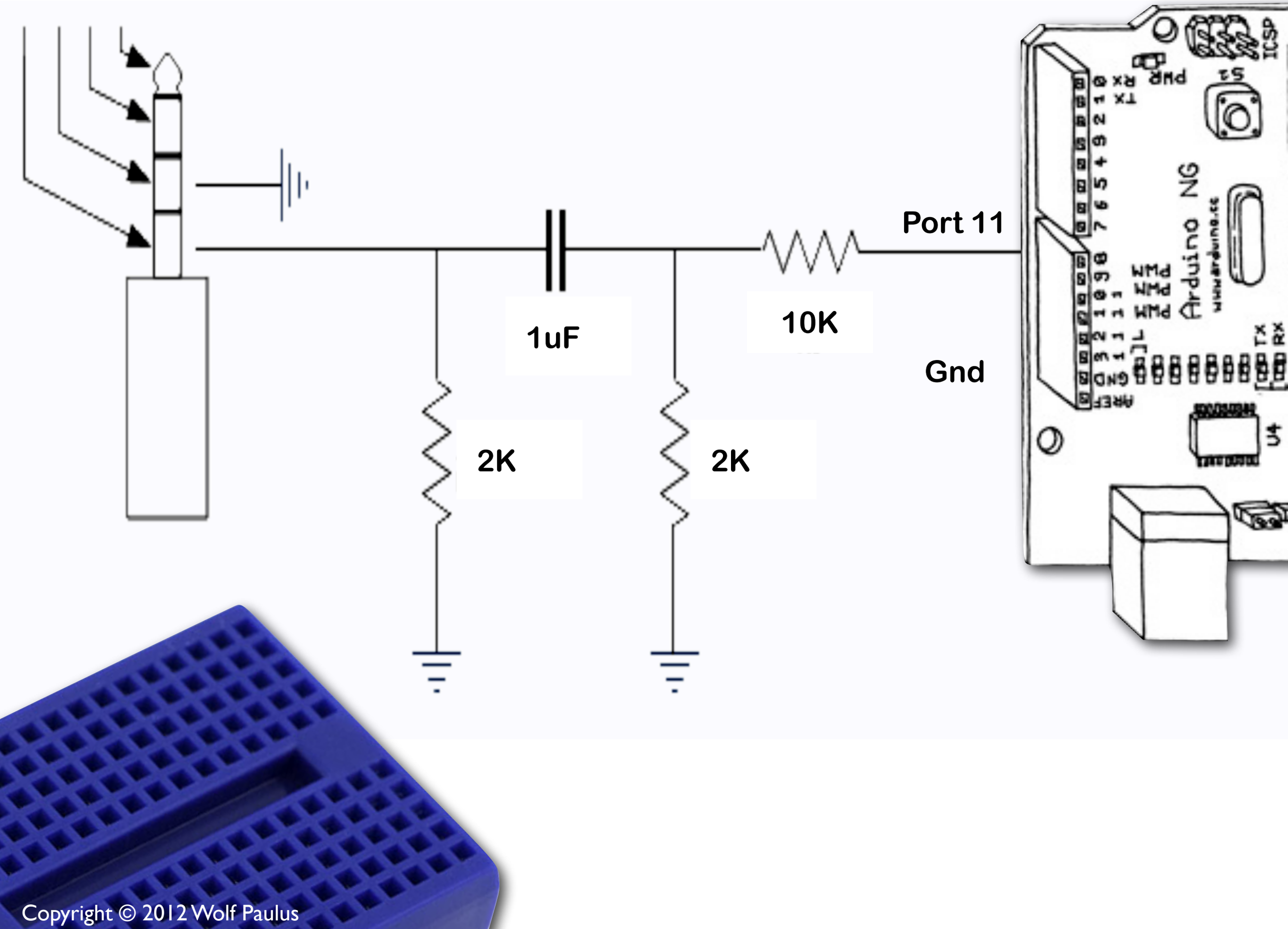
```
void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(outPin, OUTPUT);
}
```

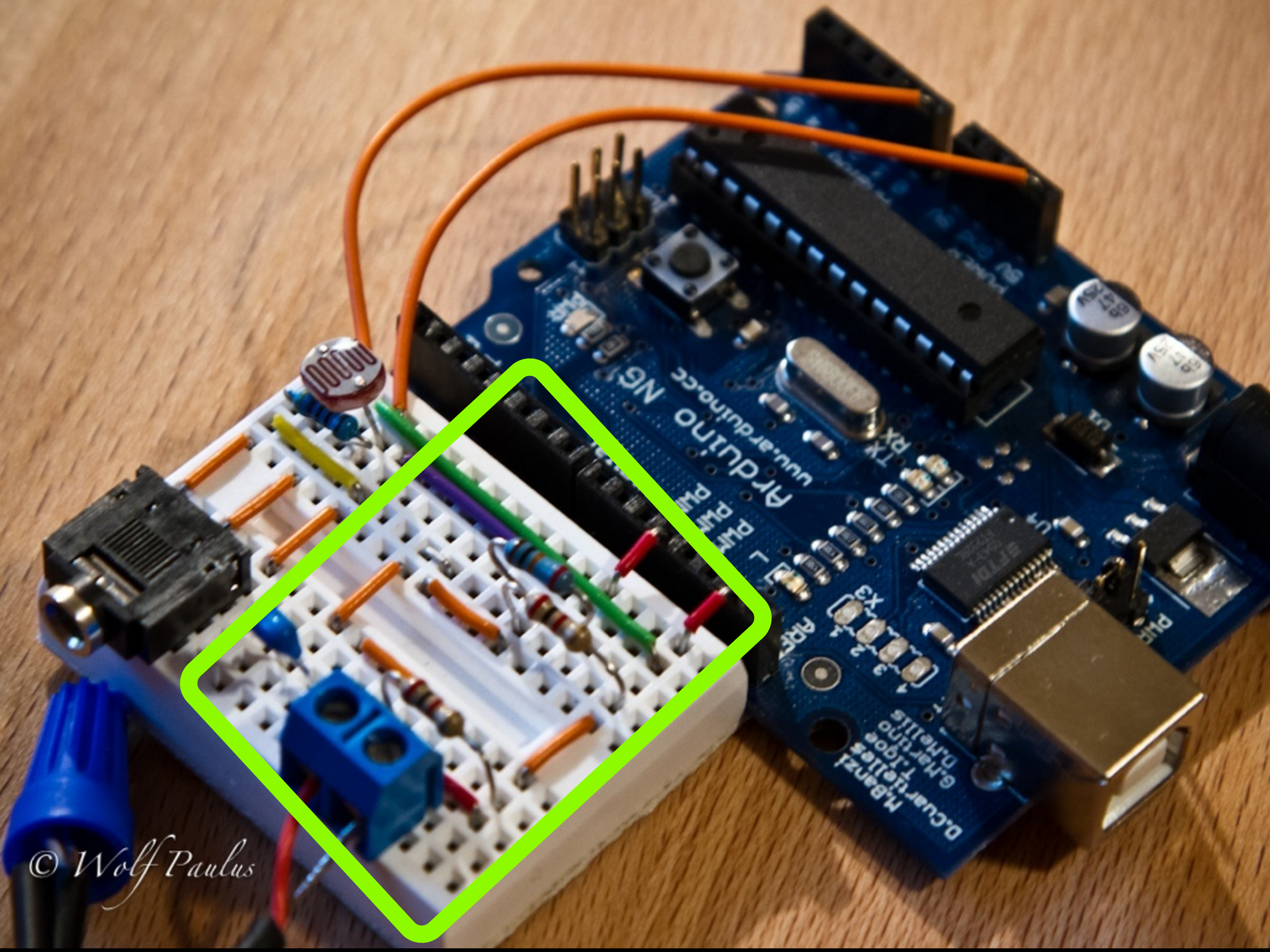
## Main Loop

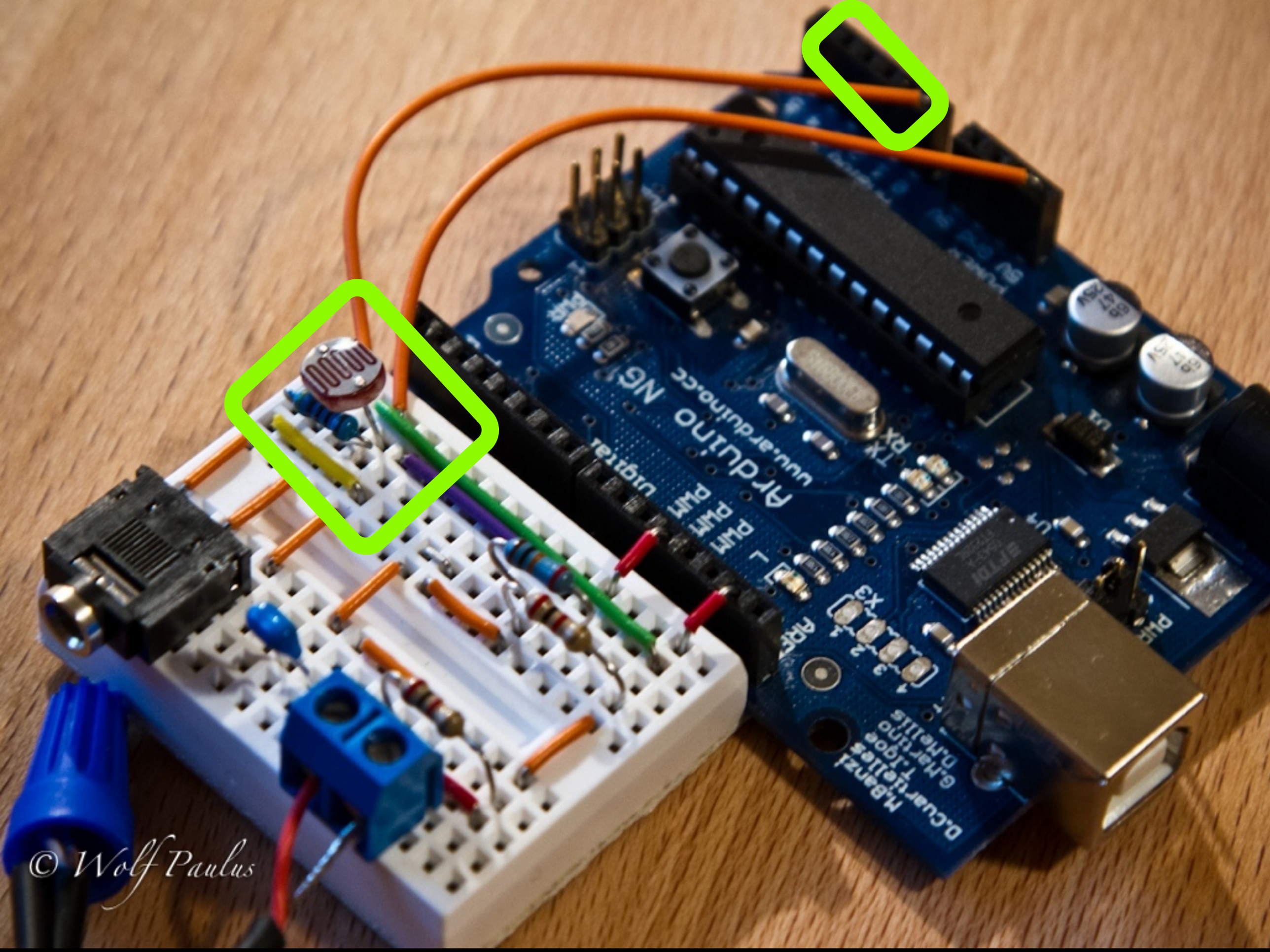
```
void loop() {
    digitalWrite(ledPin, HIGH);
    tone(outPin, freq);
    delay(duration);
    noTone(outPin);
    digitalWrite(ledPin, LOW);
    delay(pause);
}
```

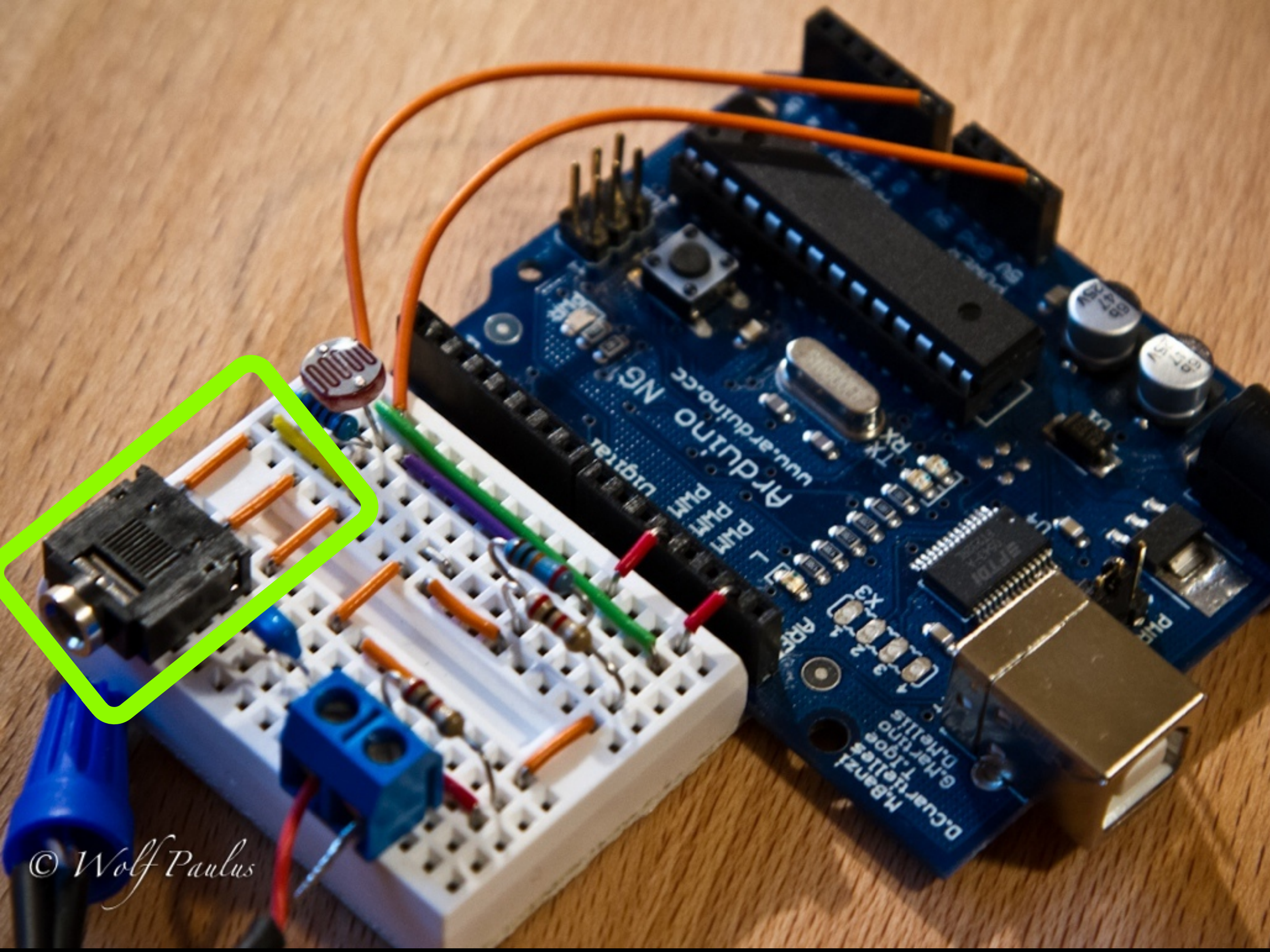
# Demo 1


Arduino + external Speaker
plays 2000 Hz sound for 2 seconds




Arduino NG
www.arduino.cc
M.Banzi
D.Cuartielles
T.Igoe
G.Martino
D.Mellis
© Wolf Paulus


Arduino NG
www.arduino.cc
M.Banzi
D.Cuartielles
T.Igoe
G.Martino
D.Mellis
PWM
PWM
PWM
TX
RX
ICSP
ICSP2
© Wolf Paulus


Arduino NG
www.arduino.cc
M.Banzi
D.Cuartielles
T.Igoe
G.Martino
D.Mellis
PWM
TX
RX

# Transferring a Message

- Encode the Message (e.g. ASCII or Morse Code)
- Convert encoded message into audio signals [D➜A]
- Send (Play) the audio signals

- Receive (Listen/Record) audio signals
- Interpret (Filter/Transform) audio signals [A➜D]
- Decode digital signal into original message

# ASCII - Encoding

H e l l o W
o r l d !

0x48 0x65 0x6C 0x6C 0x6F 0x20 0x57
0x6F 0x72 0x6C 0x64 0x21

| ASCII | Hex | Symbol |
|---|---|---|
| 0 | 0 | NUL |
| 1 | 1 | SOH |
| 2 | 2 | STX |
| 3 | 3 | ETX |
| 4 | 4 | EOT |
| 5 | 5 | ENQ |
| 6 | 6 | ACK |
| 7 | 7 | BEL |
| 8 | 8 | BS |
| 9 | 9 | TAB |
| 10 | A | LF |
| 11 | B | VT |
| 12 | C | FF |
| 13 | D | CR |
| 14 | E | SO |
| 15 | F | SI |

| ASCII | Hex | Symbol |
|---|---|---|
| 16 | 10 | DLE |
| 17 | 11 | DC1 |
| 18 | 12 | DC2 |
| 19 | 13 | DC3 |
| 20 | 14 | DC4 |
| 21 | 15 | NAK |
| 22 | 16 | SYN |
| 23 | 17 | ETB |
| 24 | 18 | CAN |
| 25 | 19 | EM |
| 26 | 1A | SUB |
| 27 | 1B | ESC |
| 28 | 1C | FS |
| 29 | 1D | GS |
| 30 | 1E | RS |
| 31 | 1F | US |

| ASCII | Hex | Symbol |
|---|---|---|
| 32 | 20 | (space) |
| 33 | 21 | ! |
| 34 | 22 | " |
| 35 | 23 | # |
| 36 | 24 | $ |
| 37 | 25 | % |
| 38 | 26 | & |
| 39 | 27 | ' |
| 40 | 28 | ( |
| 41 | 29 | ) |
| 42 | 2A | * |
| 43 | 2B | + |
| 44 | 2C | |
| 45 | 2D | - |
| 46 | 2E | . |
| 47 | 2F | / |

| ASCII | Hex | Symbol |
|---|---|---|
| 48 | 30 | 0 |
| 49 | 31 | 1 |
| 50 | 32 | 2 |
| 51 | 33 | 3 |
| 52 | 34 | 4 |
| 53 | 35 | 5 |
| 54 | 36 | 6 |
| 55 | 37 | 7 |
| 56 | 38 | 8 |
| 57 | 39 | 9 |
| 58 | 3A | : |
| 59 | 3B | ; |
| 60 | 3C | < |
| 61 | 3D | = |
| 62 | 3E | > |
| 63 | 3F | ? |

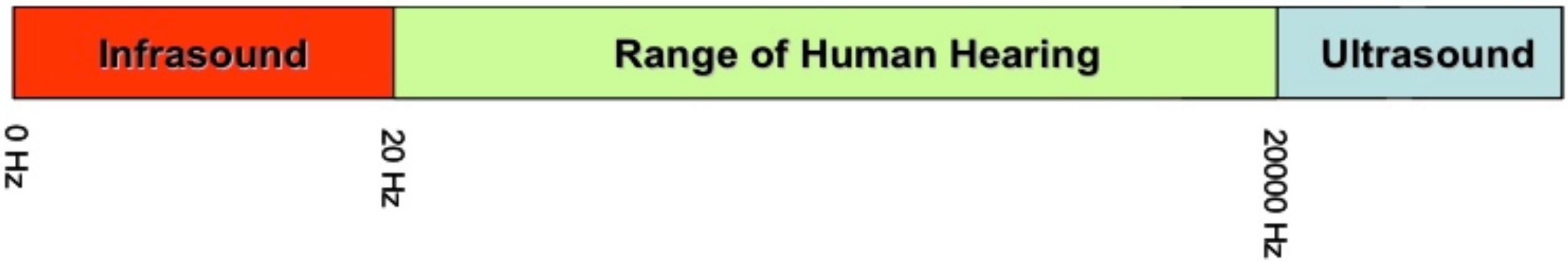
| ASCII | Hex | Symbol |
|---|---|---|
| 64 | 40 | @ |
| 65 | 41 | A |
| 66 | 42 | B |
| 67 | 43 | C |
| 68 | 44 | D |
| 69 | 45 | E |
| 70 | 46 | F |
| 71 | 47 | G |
| 72 | 48 | H |
| 73 | 49 | I |
| 74 | 4A | J |
| 75 | 4B | K |
| 76 | 4C | L |
| 77 | 4D | M |
| 78 | 4E | N |
| 79 | 4F | O |

| ASCII | Hex | Symbol |
|---|---|---|
| 80 | 50 | P |
| 81 | 51 | Q |
| 82 | 52 | R |
| 83 | 53 | S |
| 84 | 54 | T |
| 85 | 55 | U |
| 86 | 56 | V |
| 87 | 57 | W |
| 88 | 58 | X |
| 89 | 59 | Y |
| 90 | 5A | Z |
| 91 | 5B | [ |
| 92 | 5C | \ |
| 93 | 5D | ] |
| 94 | 5E | ^ |
| 95 | 5F | _ |

| ASCII | Hex | Symbol |
|---|---|---|
| 96 | 60 | ` |
| 97 | 61 | a |
| 98 | 62 | b |
| 99 | 63 | c |
| 100 | 64 | d |
| 101 | 65 | e |
| 102 | 66 | f |
| 103 | 67 | g |
| 104 | 68 | h |
| 105 | 69 | i |
| 106 | 6A | j |
| 107 | 6B | k |
| 108 | 6C | l |
| 109 | 6D | m |
| 110 | 6E | n |
| 111 | 6F | o |

| ASCII | Hex | Symbol |
|---|---|---|
| 112 | 70 | p |
| 113 | 71 | q |
| 114 | 72 | r |
| 115 | 73 | s |
| 116 | 74 | t |
| 117 | 75 | u |
| 118 | 76 | v |
| 119 | 77 | w |
| 120 | 78 | x |
| 121 | 79 | y |
| 122 | 7A | z |
| 123 | 7B | { |
| 124 | 7C | \| |
| 125 | 7D | } |
| 126 | 7E | ~ |
| 127 | 7F | |

# Mapping Sensor Reading to Frequency




ASCII: 45 .. 64

Frequency: 5,000 to 14,000

# *Example for Sensor Reading returns 159*

1. Convert INT into CHAR[] ➜ {‘1’,’5’,’9’}
2. Wrap into START and END Token ➜ {‘?’,‘1’,’5’,’9’,’@’}
3. Convert each CHAR c

   **(5 * (c -45) + OFFSET) * FREQ_RESOLUTION**

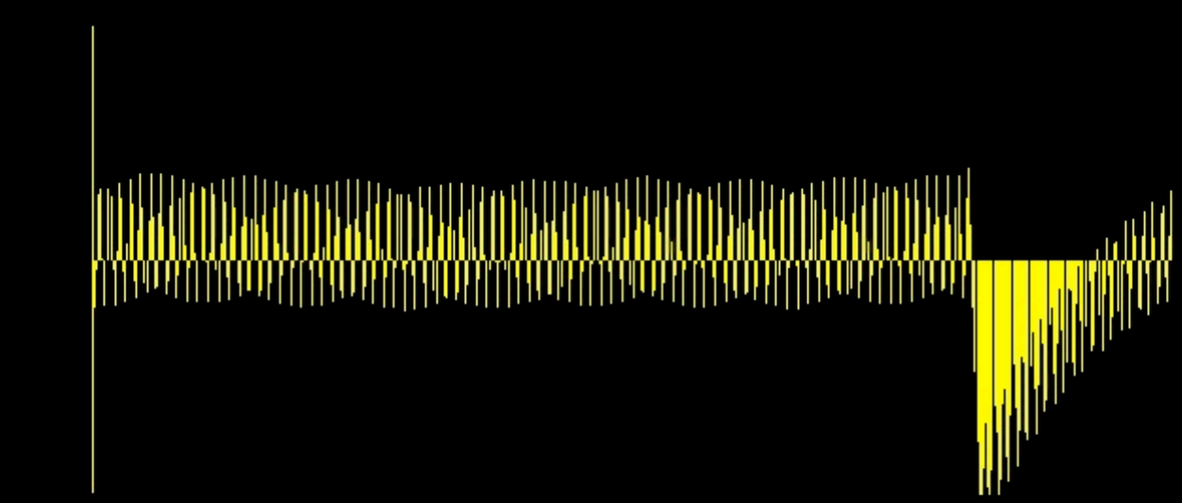
   *With OFFSET = 60, FREQ_RESOLUTION= 86.13Hz:*

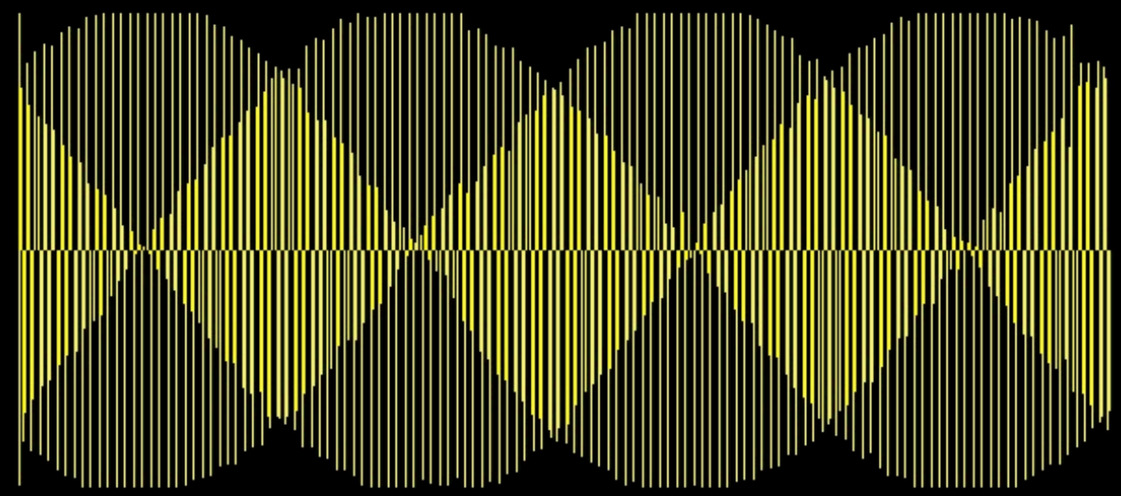
   **430.65 * C - 14211.45 Hz**

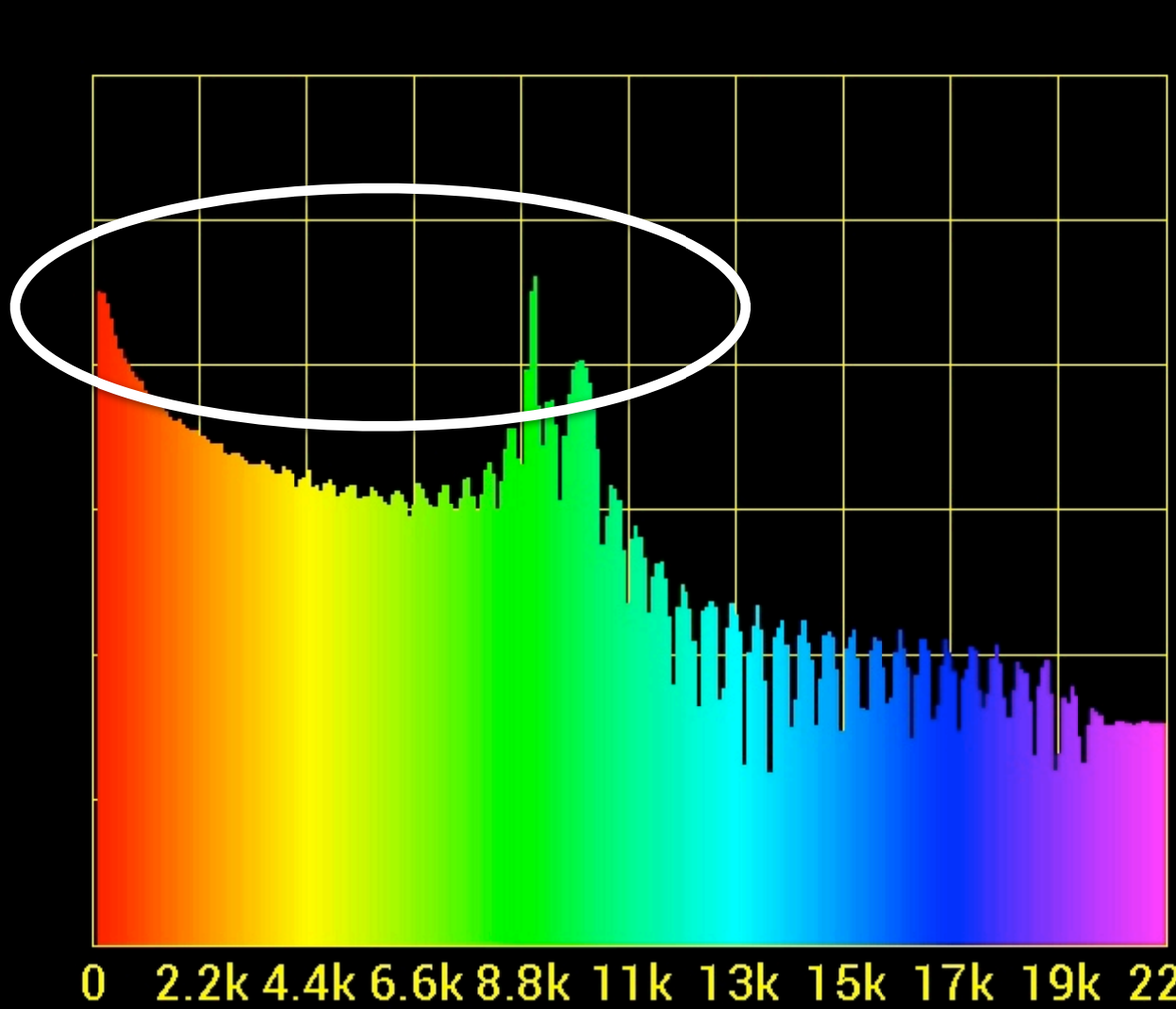
4.

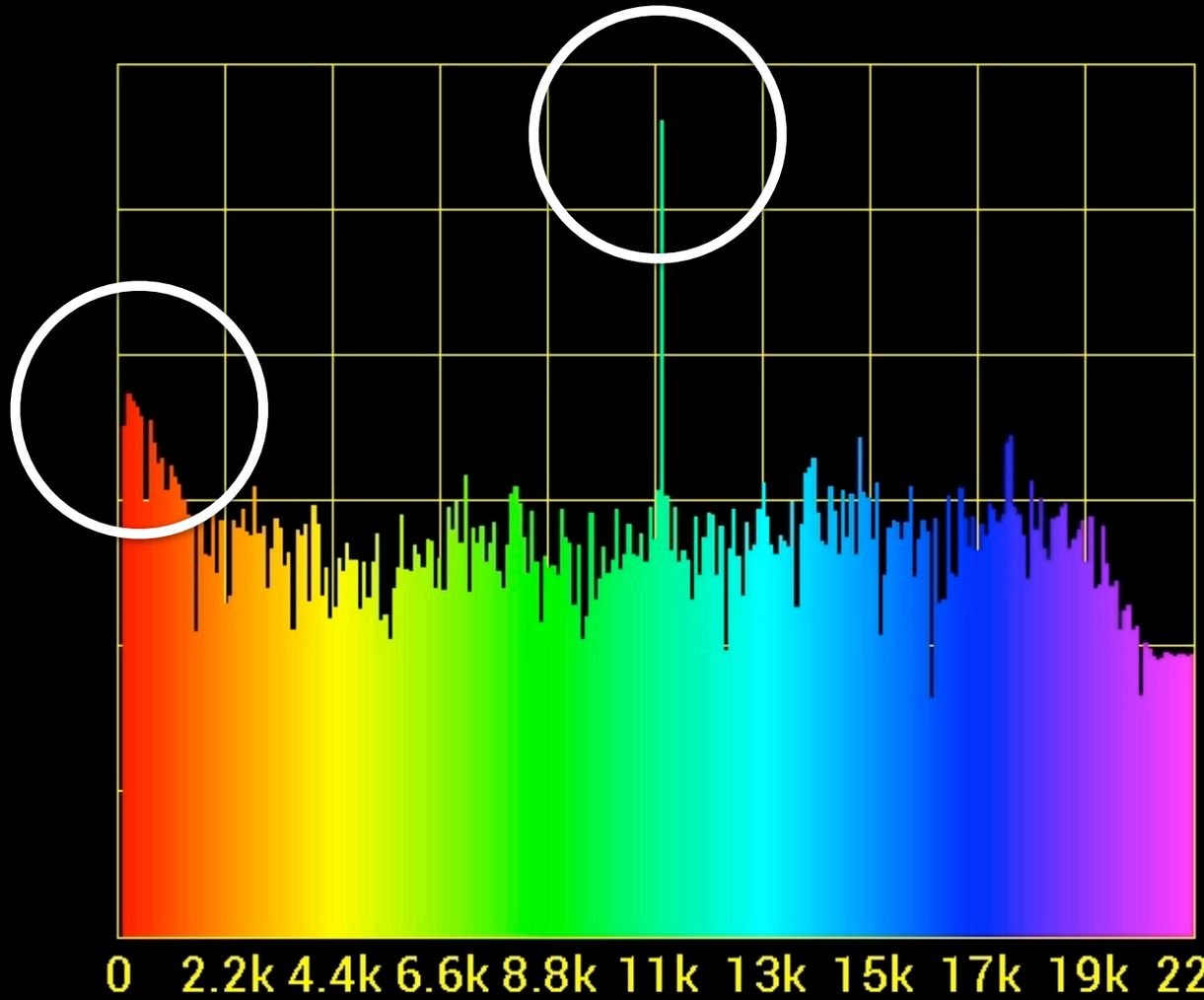
```
ACSII(‘?’) = 63 ➜  12,919 Hz
 ASCII(‘1’) = 49 ➜   6,890 Hz
  ASCII(‘5’) = 53 ➜   8,613 Hz
   ASCII(‘9’) = 57 ➜ 10,335 Hz
    ASCII(’@’) = 64 ➜ 13,350 Hz
```

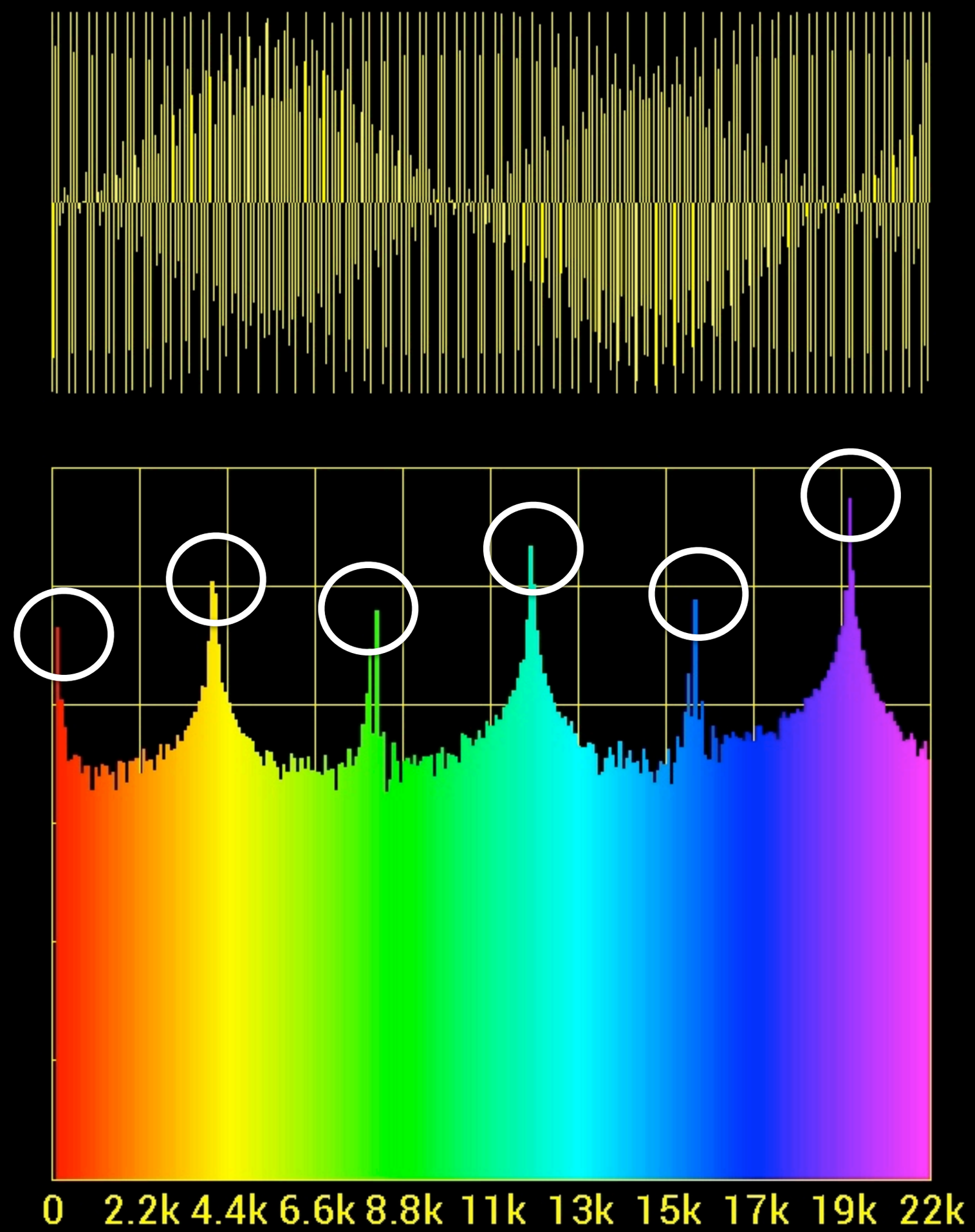
# Usable Frequency Range depends on the Hardware








0
2.2k
4.4k
6.6k
8.8k
11k
13k
15k
17k
19k
22k

# Initialization


```
const int ledPin = 13;  // built-in LED
const int outPin = 11;  // PWM Port out
const int inPin   =   0;  // Light Sensor

const char STX = '?';  // Start Token
const char ETX = '@';  // End Token

const unsigned int SAMPLE_RATE = 44100; // Hz
const int SAMPLES = 512 ; // Num of Samples (128,256,512,1024)
const int REPEAT = 3;         // prolong the signal
const int OFFSET = 60;        // 60 un-usable bin on the spectrum's low end

const double FREQ_RES = (double) SAMPLE_RATE / SAMPLES; // 86.13 frq per bin
const int DURATION = (int) REPEAT * (1000 / FREQ_RES); // about 35ms
const int ENC_ETX = encodeAscii(ETX); // pre-calc  for later use in loop

unsigned int frq[6]; // global frequency array, changes every run
```

# Setup and Encoding a message


```
void setup() {
    pinMode( ledPin, OUTPUT );
    pinMode( outPin, OUTPUT );
    pinMode( inPin,  INPUT );
    Serial.begin( 9600 );  // debugging on
}

/**
 * Encode the message into an frequency array
 * Wrap the message into Start and End tags
 */
void encodeInt(int m) {
    char message[5];
    itoa( m,message,10 );

    frq[0] = encodeAscii( STX );
    for ( int i=0; i<5; i++ ) {
        if ('\0'==message[i]) {   // replace '\0' marker w/ ETX
            frq[i+1] = encodeAscii( ETX );
            break;
        }
        frq[i+1] = encodeAscii( message[i] );
    }
}
```

# Encoding a single char and main loop


```
unsigned int encodeAscii(char c) {
    return (unsigned int) ((5 * (c-45) + OFFSET) * FREQ_RES);
}

void loop() {
    const int brightness = analogRead(inPin);   // reads value 0-1023
    encodeInt(brightness);   // into global frq[]
    digitalWrite(ledPin, HIGH);
    int i=0;
    do {
        tone(outPin, frq[i]);
        delay(DURATION);
        noTone(outPin);
    }
    while (frq[i++] != ENC_ETX);
    digitalWrite(ledPin, LOW);
    delay(500);
    Serial.println(brightness);
}
```

# Demo 2

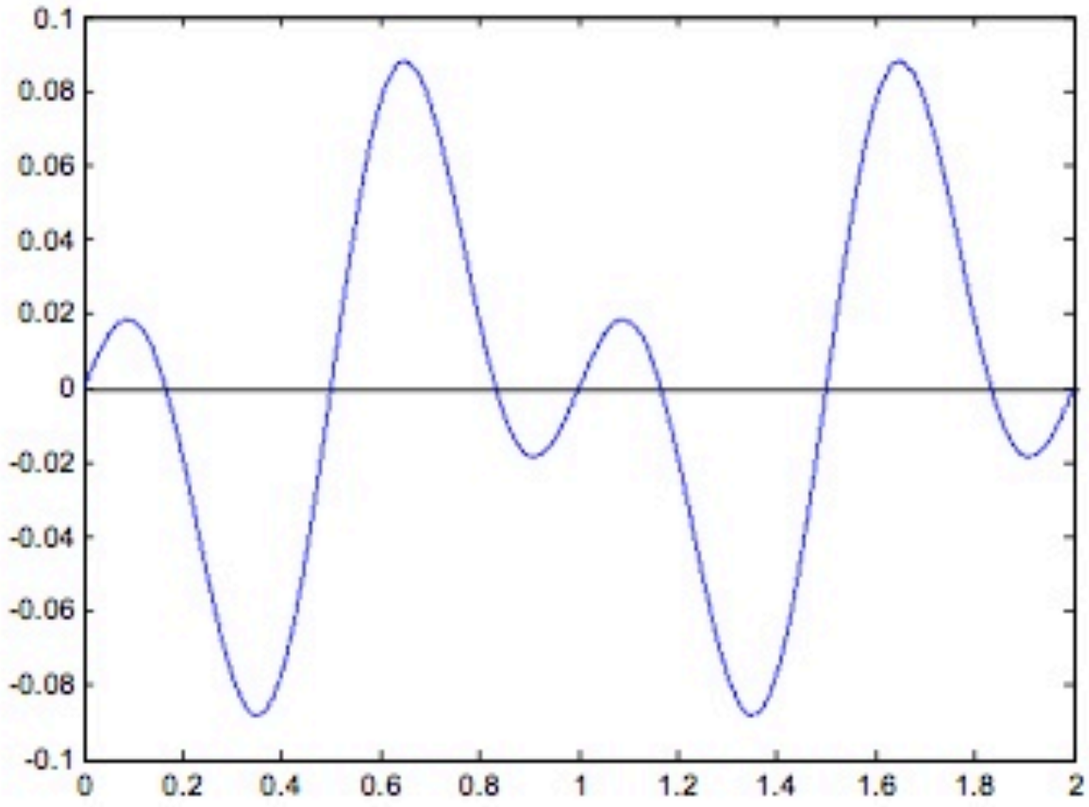

Arduino + external Speaker + TRRS-Plug
plays LightSensor value Message every .5 seconds

# Transferring a Message

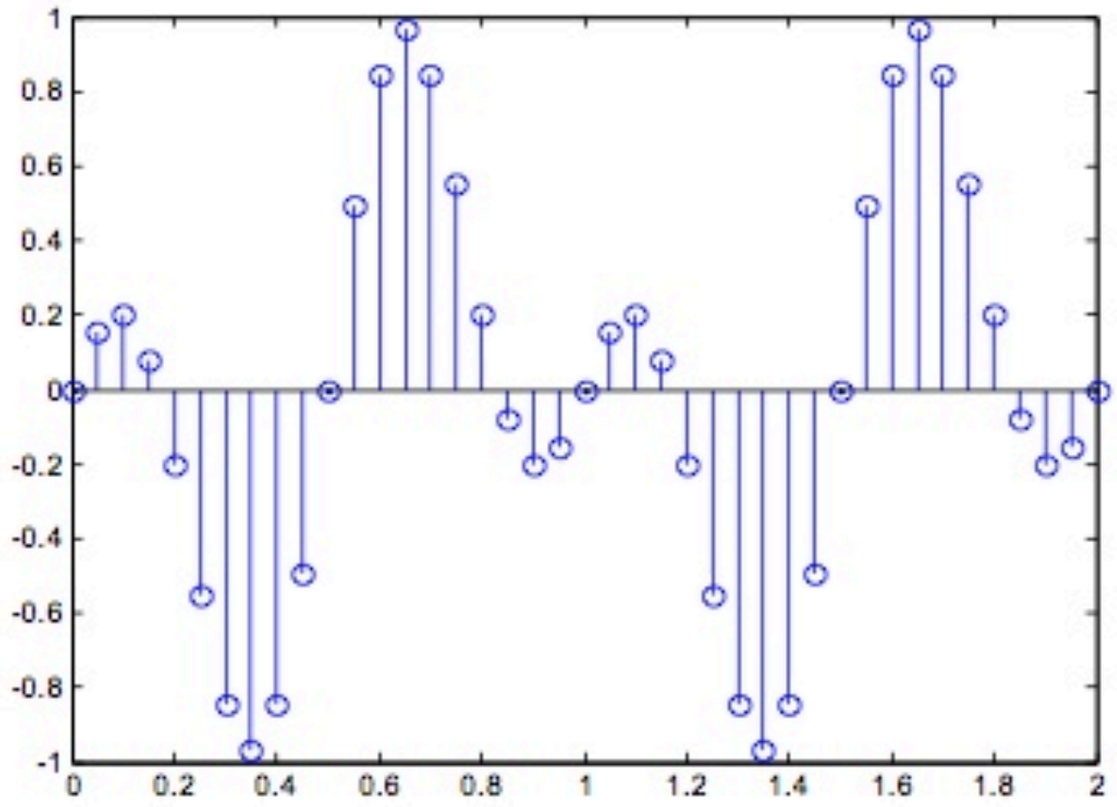
- Encode the Message (e.g. ASCII or Morse Code)
- Convert encoded message into audio signals [D➜A]
- Send (Play) the audio signals

- Receive (Listen/Record) audio signals
- Interpret (Filter/Transform) audio signals [A➜D]
- Decode digital signal into original message

# The Receiver




Original Analog Signal




Sampled Discrete-Time Signal

- The total number of times the signal is sampled in one second is defined as the **sampling frequency.**
  Only **44.100 Hz** is guaranteed on all Android Devices

- **Nyquist–Shannon** sampling theorem:
  *The sampling frequency should be at least twice the highest frequency contained in the signal.*

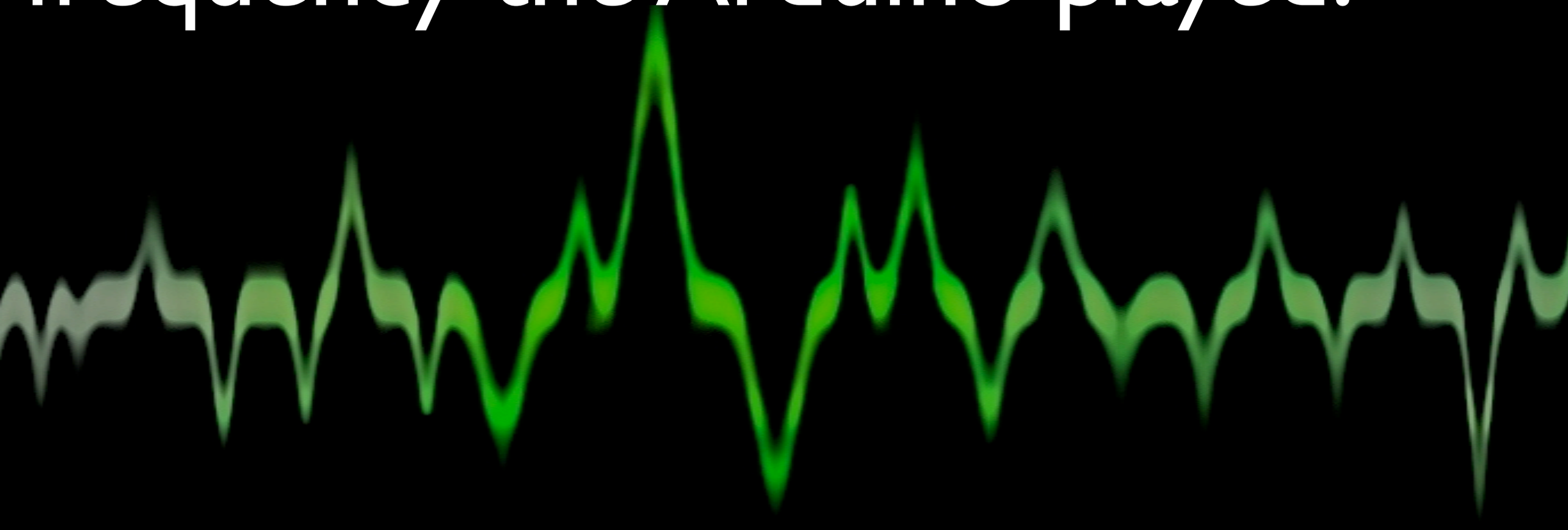
  *I.e. Signals from the Arduino board should be in the 0..22,050 Hz range.*

At a 44,100 Hz sample rate, 512 samples are taken in **11.6 ms**

```
// construct AudioRecord to record audio from microphone with sample rate of 44100Hz
int minSize = AudioRecord.getMinBufferSize(sampleRate,AudioFormat.
                                           CHANNEL_CONFIGURATION_MONO,
                                           AudioFormat.ENCODING_PCM_16BIT);
AudioRecord audioInput = new AudioRecord(MediaRecorder.AudioSource.MIC, sampleRate,
                                           AudioFormat.CHANNEL_CONFIGURATION_MONO,
                                           AudioFormat.ENCODING_PCM_16BIT);
                                           minSize);
...
short[] buffer = new short[readSize];
audioInput.startRecording();
audioInput.read(buffer, index, readSize); // record data from mic into buffer
```

We took 512 samples in 11.6 ms

But we need to know the frequency the Arduino played.



# FFT (Fast Fourier Transform)

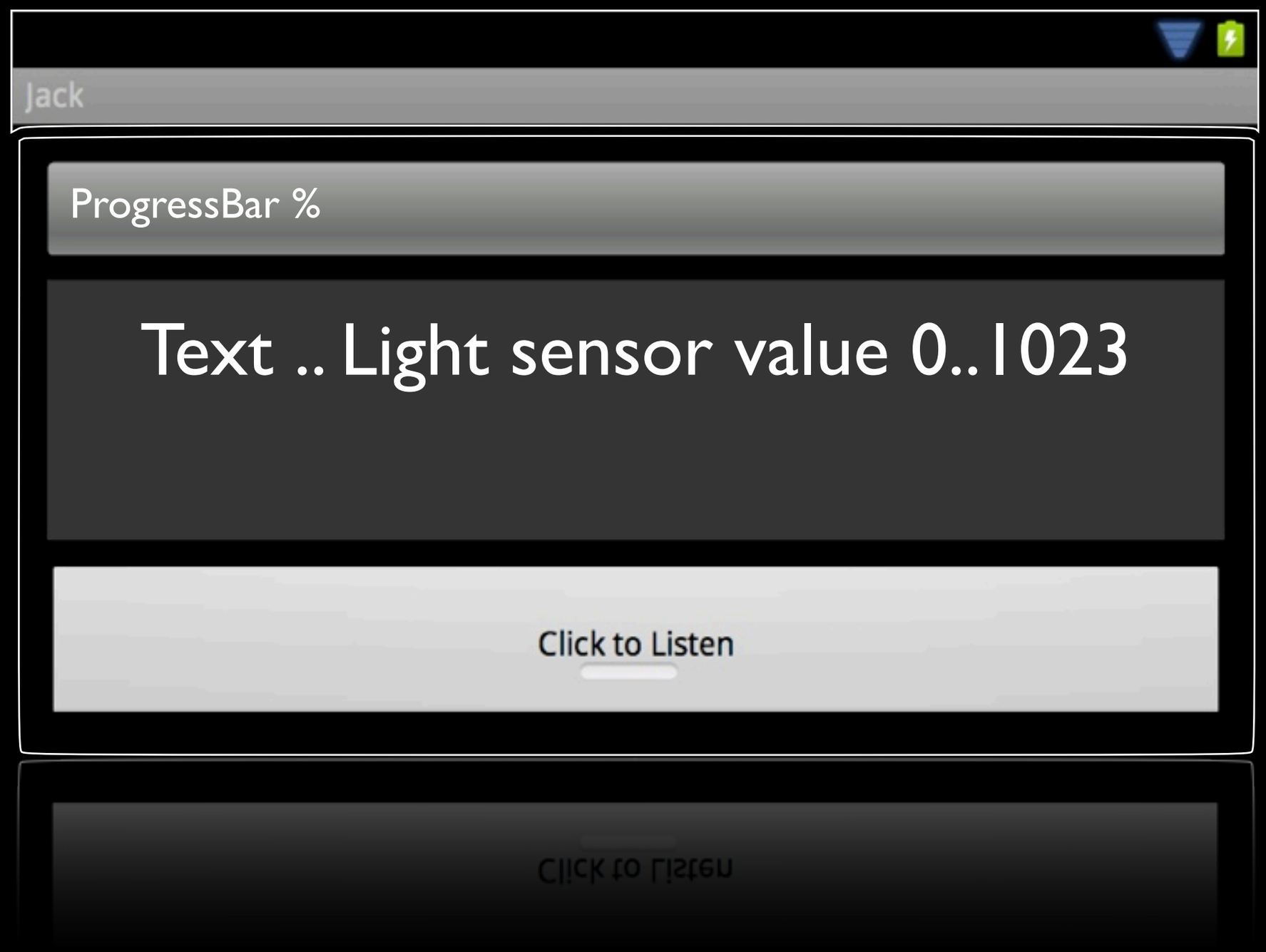
FFT is an effective algorithm to convert signals from ***time domain*** to the ***frequency domain***.

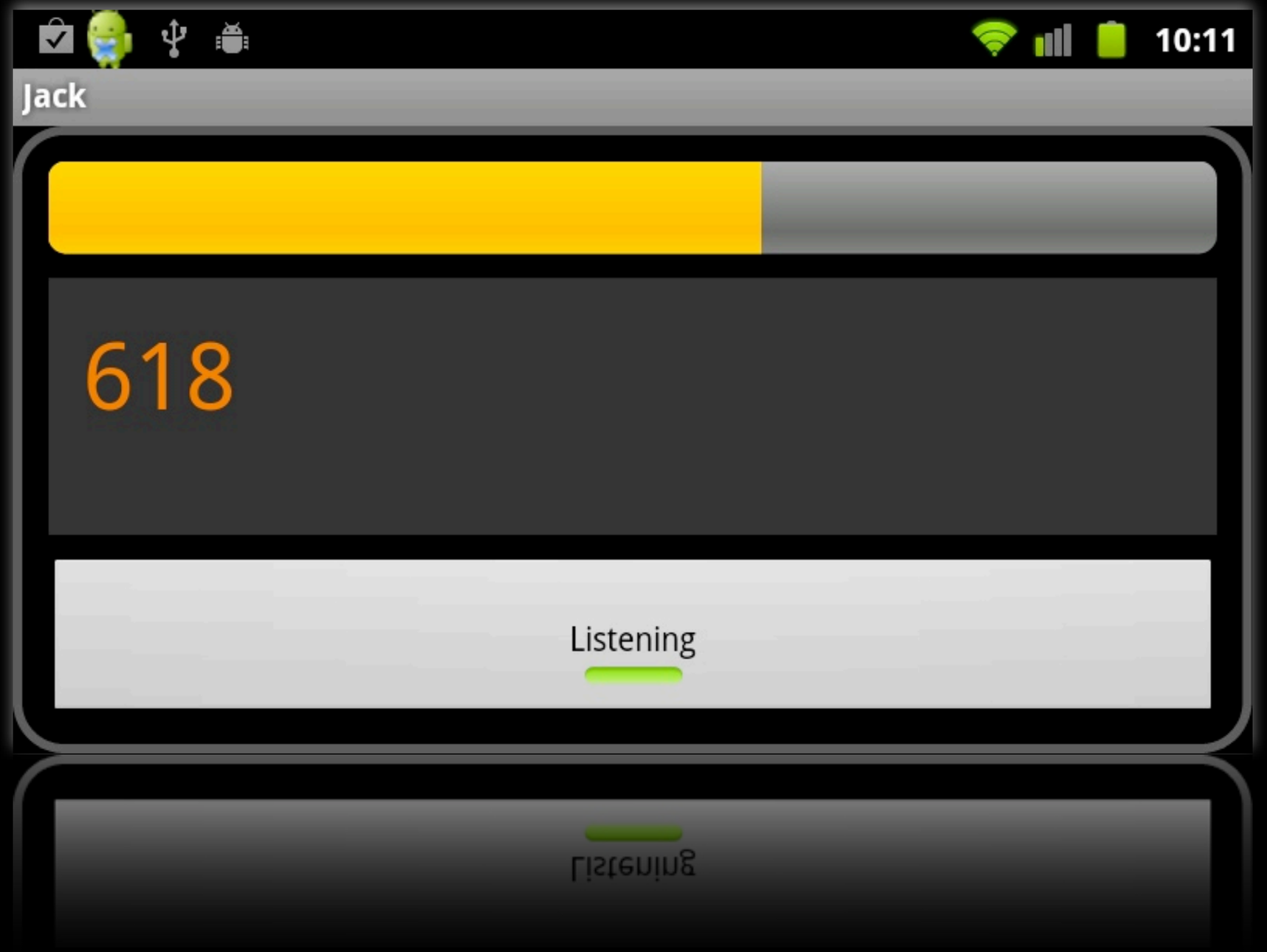
We use the 512 samples in the audio buffer as input; and the FFT algorithm returns a complex array, allowing us  to calculate the magnitude of 512 **frequency ranges**.

- Sampling Rate 44,100 Hz
- 512 Samples
- Frequency Range is split into 512 ranges
- The ***Frequency Resolution*** 44,100 / 512 = 86.13 Hz

We input 512 samples and as a return we know which of the 512 frequency ranges had the strongest signal.

We won't know the exact frequency the Arduino played, but the ***frequency bucket*** *it was in.*





```java
public interface Const {
    int SAMPLE_RATE = 44100;    // in Hz
    int SAMPLES = 512;          // number of samples
    int BITS = 9;               // a power of two such that 2^b is the number_of_samples
    int REPEAT = 3;             // repeat each signal
    int OFFSET = 60;            // number of un-usable bin on the spectrum's lower end
}
```

```java
    /**
     * Reverse the encoding process. An FFT id is provided.
     *
     * @param bin <code>int</code>
     * @return <code>int</code>  value
     */
    public byte decode(int bin) {
        int k = (int) ((bin- Const.OFFSET)/5.0 + 45);
        return (byte) (0x7F & k);
    }
```

# Demo 3


Arduino + Android + external Speaker + TRRS-Plug
LightSensor value encoded into Audio Signals
Audio Signal decoded into Text and Progress-bar
updated every .5 seconds

# Summary

We built an embedded system that captured sensor data
Encoded the data into ASCII and then into frequencies
Generated tones for a predefined duration

On an Android Phone, we received the Audio Signal via TRRS Cable
Decoded the signal back into ASCII and then Numbers
Displayed the Text and animated a progress bar every .5s

Android code, Arduino code, Slides:

**git clone https://github.com/wolfpaulus/jack.git**