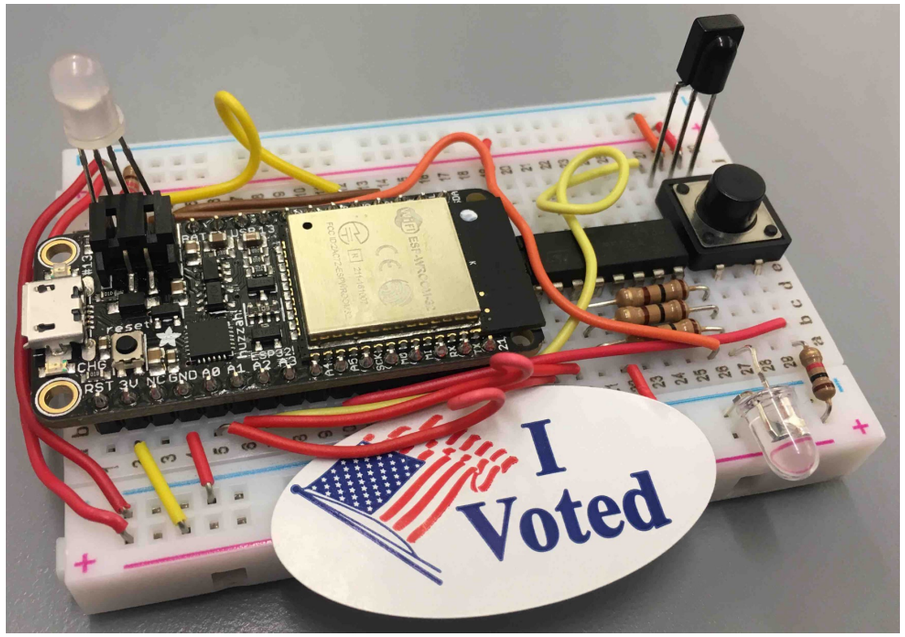
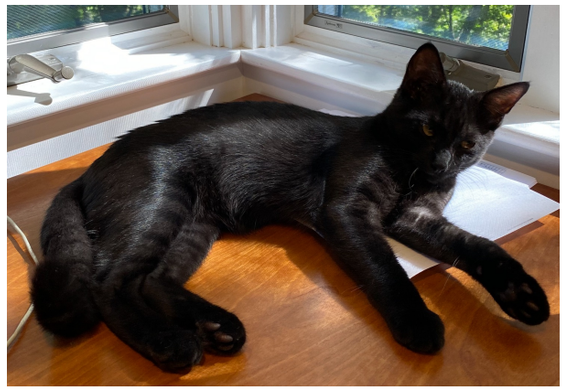


BU-EC444

Fall 2024
Prof. Little

Review of Quest 2: Cat Track

Quest 2: Cat Collar Wearable



Cat

Collar Unit

Time-series graphic display of activities

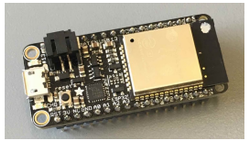
Functions

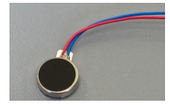
- Tracks active and inactive times
- Reports on activities vs time
- Button press displays cat name and current activity and time in activity

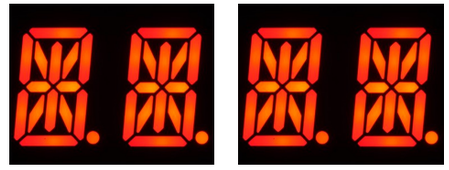
Key takeaways:

- I2C interfacing
- Alphanumeric display
- Using node
- Reporting and displaying results

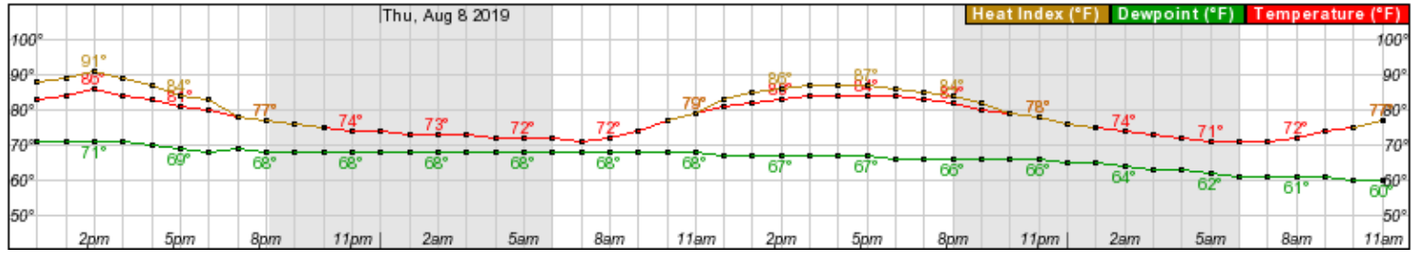
Quest 2: Cat Track







Serial Port/
Console



Raw Engineering Units

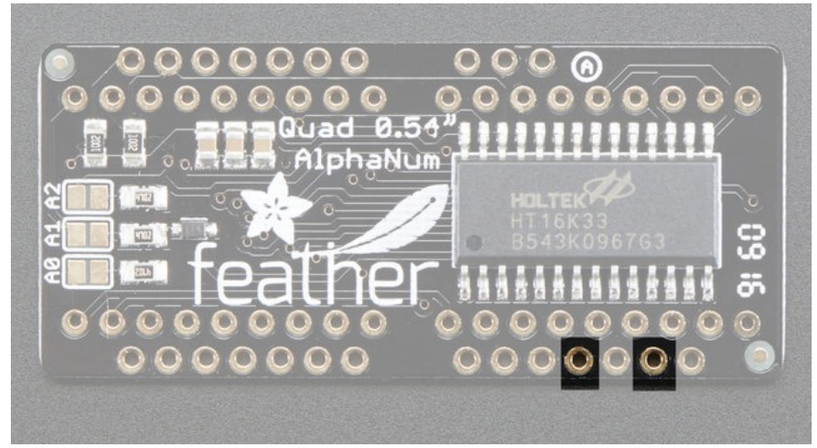
Quest 2 Skill Cluster

- Alphanumeric Display and Skill
- General Purpose Timer and Timer Skill
- Accelerometer Skill
- Node.js
- CanvasJS
- Micromotor

Alphanumeric Display Skill

Alphanumeric Display

- Can use the I2C code for the display in the course code examples
- Will need to be modified to meet the tasks in the skill



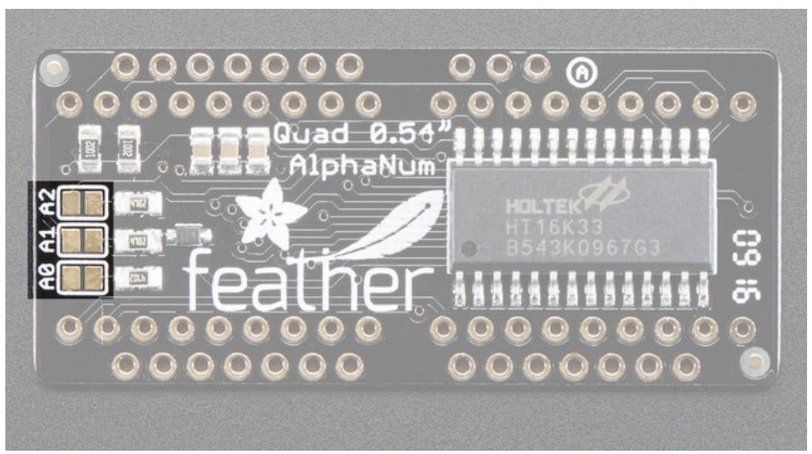
I2C

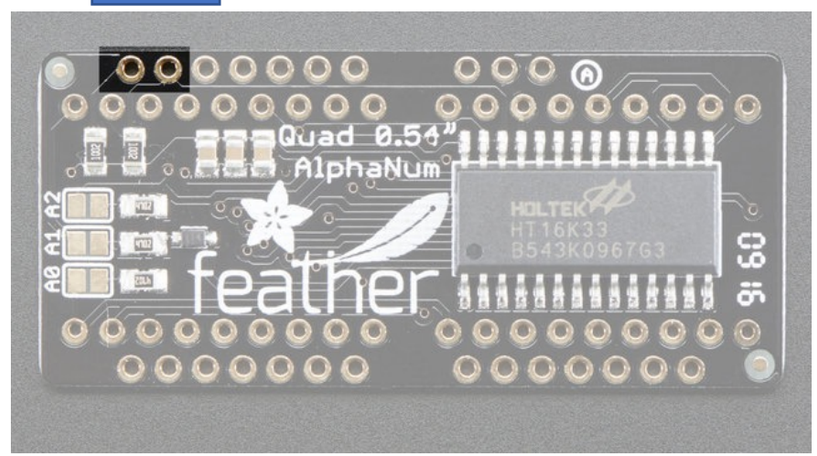
Power

3v

Huzzah pinouts

Address jumpers





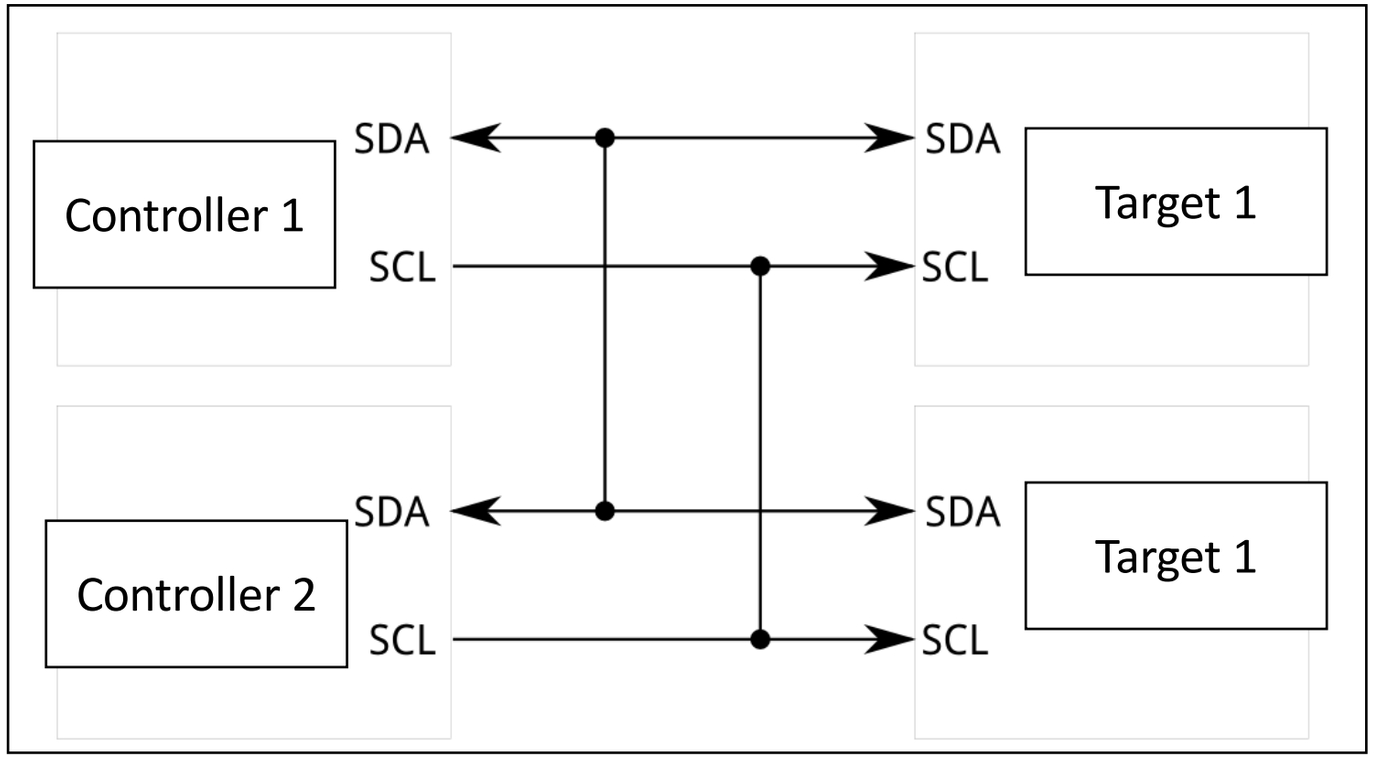
From Adafruit

I2C (“inter-integrated circuit protocol”) or… “I-Squared C”

Only two wires
Multiple devices on bus
Multiple masters possible

Serial DAta line
Serial CLock line

More here:
https://learn.sparkfun.com/tutorials/i2c

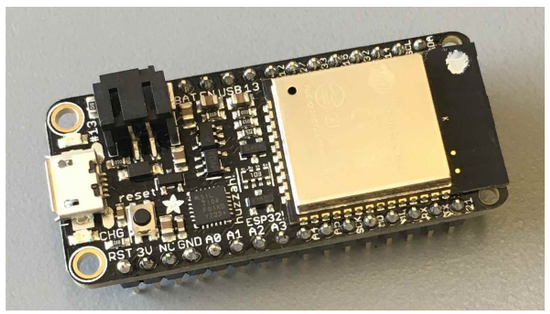


Modified from Sparkfun

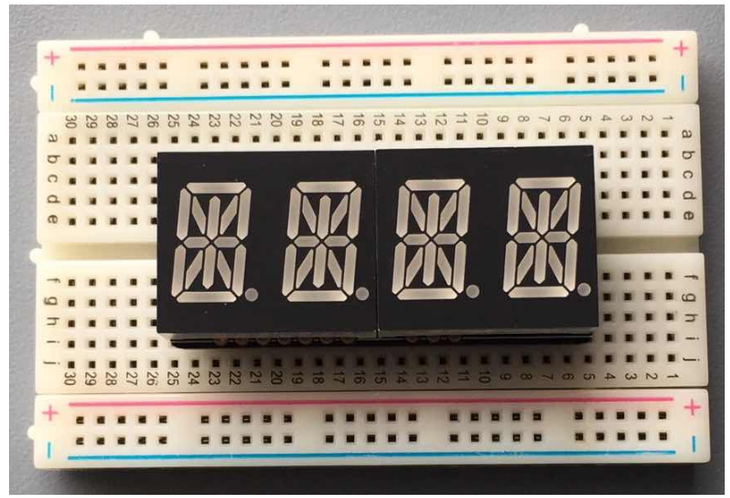
Master → Controller Slave → Target

Alphanumeric display skill

- Wire up the alphanumeric display on your breadboard with the ESP32
- Create a module that allows you to write an ASCII character string (up to 8 characters) to the display from the console. If there are more than 4 → scroll
- Demonstrate the skill



SCA, SCL, 3V, GND



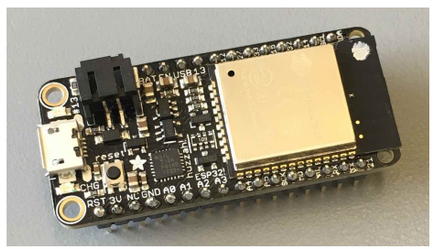
Tips for I2C and other breadboard wiring

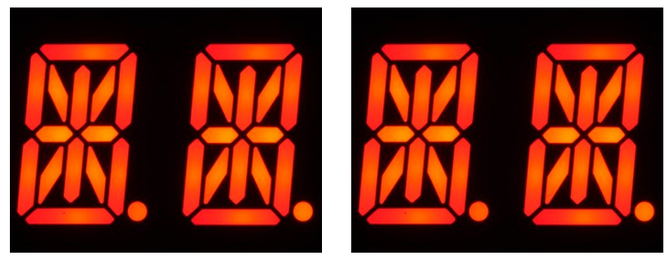
- Build clean wiring and you will be rewarded with fewer debugging problems
- I2C requires pull up resistors (but the SCL and SDA pins have internal pull ups) – additional pull-ups not required
- The LED driver uses MSB (highest value bit to the left)
- A **very useful function** to build is a way to write ASCII characters to the alphanumeric display

Timer Skill

Timer (Stopwatch) Skill

- Use the **timer interrupt** instead of polling or waiting
- Press button → start counting-up in seconds → on alphanumeric display
- Press button again → restarts
- Wraps 99 → 00





Using the General Purpose Timer (GPTimer)

- **Similar pattern to interrupt handler**
- Lots of timer options
- Example
  - Count up
  - Throw interrupt when reach a value
  - Service the interrupt (ISR)
  - Automatically reset and repeat
- Pass messages from ISR (sometimes referred to as a “callback”) to the main program or tasks
- Can have multiple interrupts and all kinds of complexity (see the docs)

Using the General Purpose Timer (GPTimer)

Define a structure for queue elements
Define a timer queue for events
Define the timer ISR (callback)

Initialization
Configure the timer
Instantiate the timer
Set the timer ISR (callback)
Register the ISR (callback)
Enable the timer
Configure the alarm
Start the timer

Timer task
While true {
    If something in the timer queue (alarm)
        take it out and do something)
}

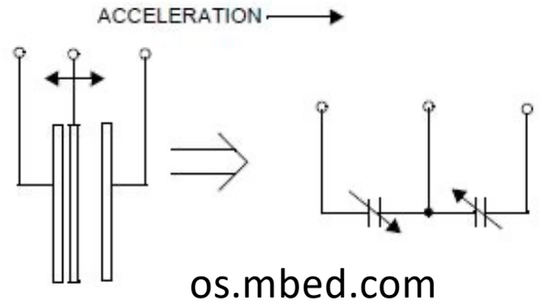
Main
Create timer queue
Create Timer Task
Initialization

See the design pattern for details

Accelerometer Skill

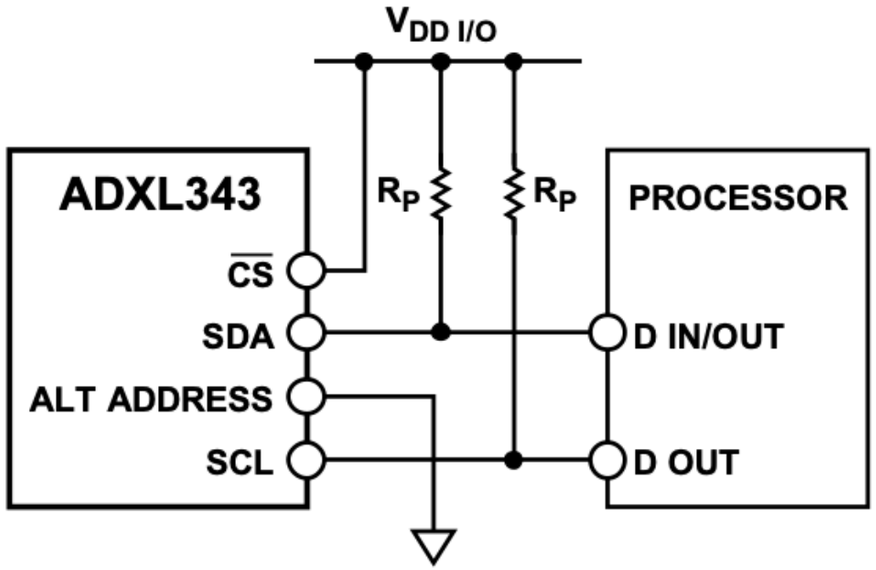
Skill: Accelerometer

- Bringing up the accelerometer
  - ADXL343 / Adafruit breakout board
  - How to interface to it: I2C (used with the alpha display)
  - What does it measure?
  - Getting tilts (orientation) from acceleration

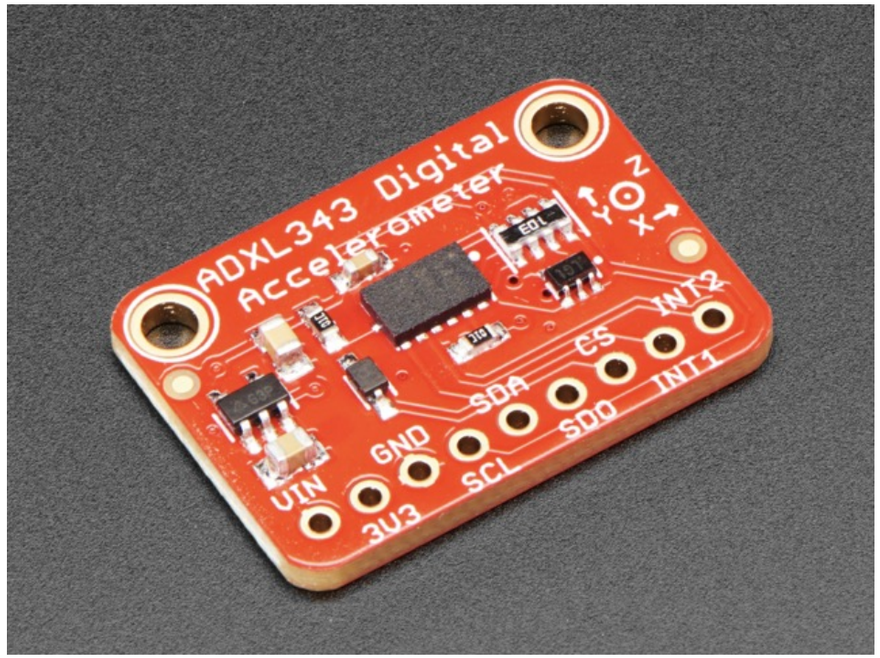


os.mbed.com

Note: ADXL343 is *chip*
Breakout is *Adafruit*



ADXL343 Data Sheet



3.3V
Provided

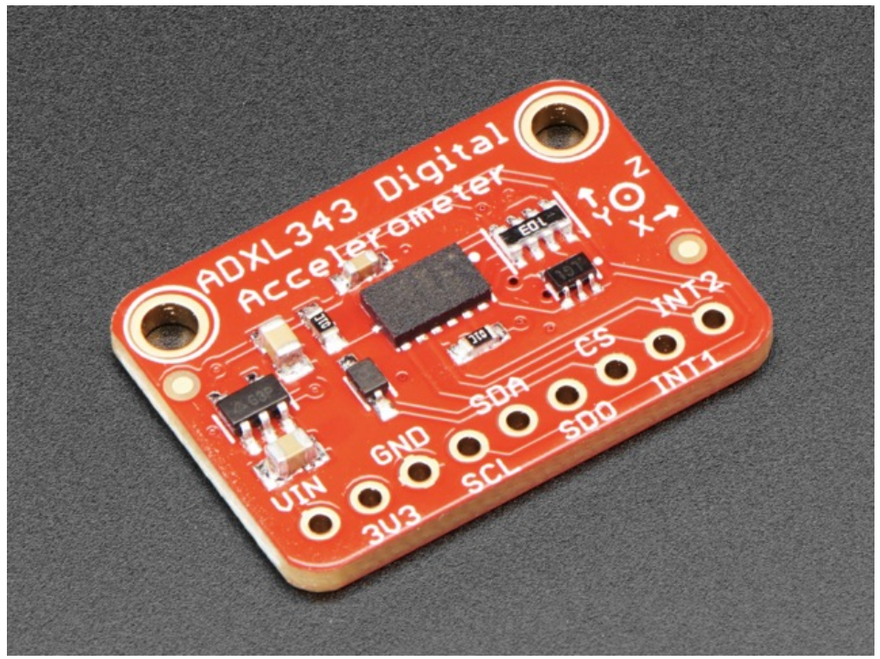
Adafruit

Accelerometer – what's it good for?

- Basic IMU – use to detect motion, motion control
- **Activity monitoring (steps, cadence, etc.)**
- Flying a drone/quadcopter
- Orientation of a device (turn screen on/off)
- Detect single or double taps
- Fall detection (human, device)
- Inactivity (very important) – turn things off

You will need your ESP32 to talk to the ADXL343 via I2C

- Wire your ESP32 I2C bus pins to the accelerometer breakout board
  - SCL
  - SDA
  - No pull ups are required (built into the breakout board)
  - 3.3V, GND (CS defaults to I2C)
- Consult the **I2C brief** for example code
  - Defining I2C parameters
    - Default Address: 0x53
  - Instantiating I2C driver
  - Reading device ID
- Use base code to flesh-out other reads
  - Reading data from device
  - Writing to registers
  - Datasheet is truth – consult regularly



Accelerometer Reference

Getting access to the ADXL343 features

- Read from and write to internal registers
- I2C brief – shows how to read device ID

Register 0x00—DEVID (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	0	1	0	1

The DEVID register holds a fixed device ID code of 0xE5 (345 octal).

REGISTER MAP

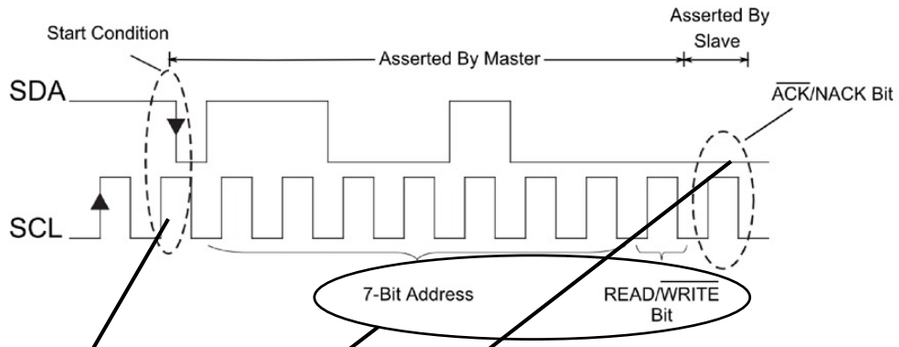
Table 19.

Address					
Hex	**Dec**	**Name**	**Type**	**Reset Value**	**Description**
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency
0x23	35	Window	R/$\overline{W}$	00000000	Tap window
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time

ADXL343 data sheet (not complete list)

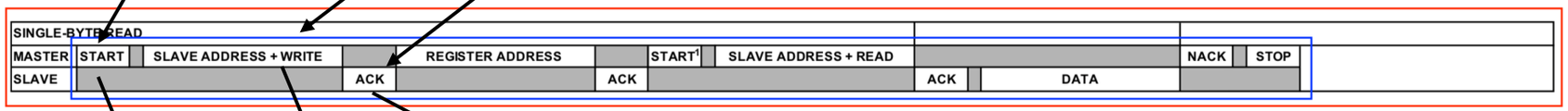
I2C Payload: Actual signals, Datasheet POV, Code

Actual signals



etc.

Datasheet POV



Code

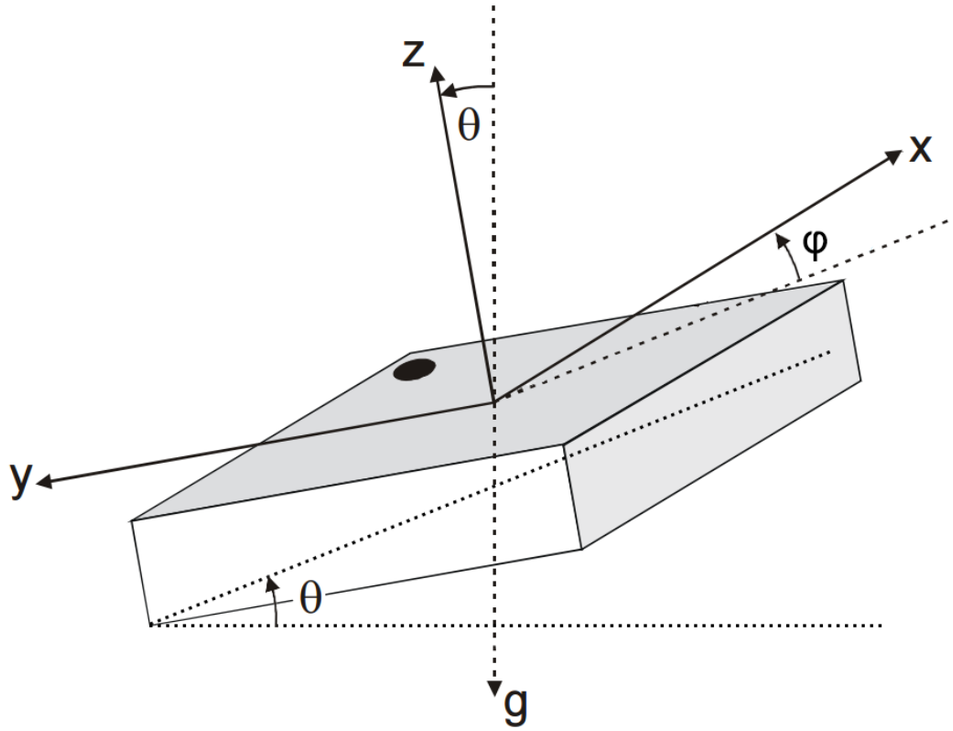
```c
// Get Device ID
int getDeviceID(uint8_t *data) {
  int ret;
  i2c_cmd_handle_t cmd = i2c_cmd_link_create();
  i2c_master_start(cmd);        // 1. Start
  i2c_master_write_byte(cmd, ( SLAVE_ADDR << 1 ) | WRITE_BIT, ACK_CHECK_EN); // 2.-3.
  i2c_master_write_byte(cmd, ADXL343_REG_DEVID, ACK_CHECK_EN); // 4.-5.
  i2c_master_start(cmd); // 6.
  i2c_master_write_byte(cmd, ( SLAVE_ADDR << 1 ) | READ_BIT, ACK_CHECK_EN);  // 7.-8.
  i2c_master_read_byte(cmd, data, ACK_CHECK_DIS); // 9.-10.
  i2c_master_stop(cmd); // 11. Stop
  ret = i2c_master_cmd_begin(I2C_EXAMPLE_MASTER_NUM, cmd, 1000 / portTICK_RATE_MS); // This
  i2c_cmd_link_delete(cmd);
  return ret;
}
```

Interrupts at the ADXL343

- Can set device to trigger on certain events – very handy
- Two interrupt lines, mappable to one or more events
- Possible interrupts:
  - Data_Ready – when new data is available to read by the processor
  - Single_Tap – a single acceleration event occurs exceeding THRESH_TAP for DUR interval
  - Double_Tap – like single tap, but two other parameters (latency, window)
  - Activity – accel exceeds some threshold Tresh_Act
  - Inactivity – accel less that Thresh_Inact for some time Time_Inact
  - Free_Fall – combines Activity on three axes for Time_FF
- Multiple interrupts on a pin → need to interpret registers to isolate

Tilt

- Calculating tilt angles (static) can be done by applying trig to this graphic – roll, pitch, yaw
- Look up the equations to convert acceleration to 3 angles
- Or consult the reference links
- Code to convert renders angles using trig and sqrt functions
- roll = atan2(y, z); pitch = atan2((- x), sqrt(y*y + z*z))
- Convert radians to degrees by *57.3



https://wiki.dfrobot.com/How_to_Use_a_Three-Axis_Accelerometer_for_Tilt_Sensing

Tilt – data responses to gravity Ax, Ay, Az – static

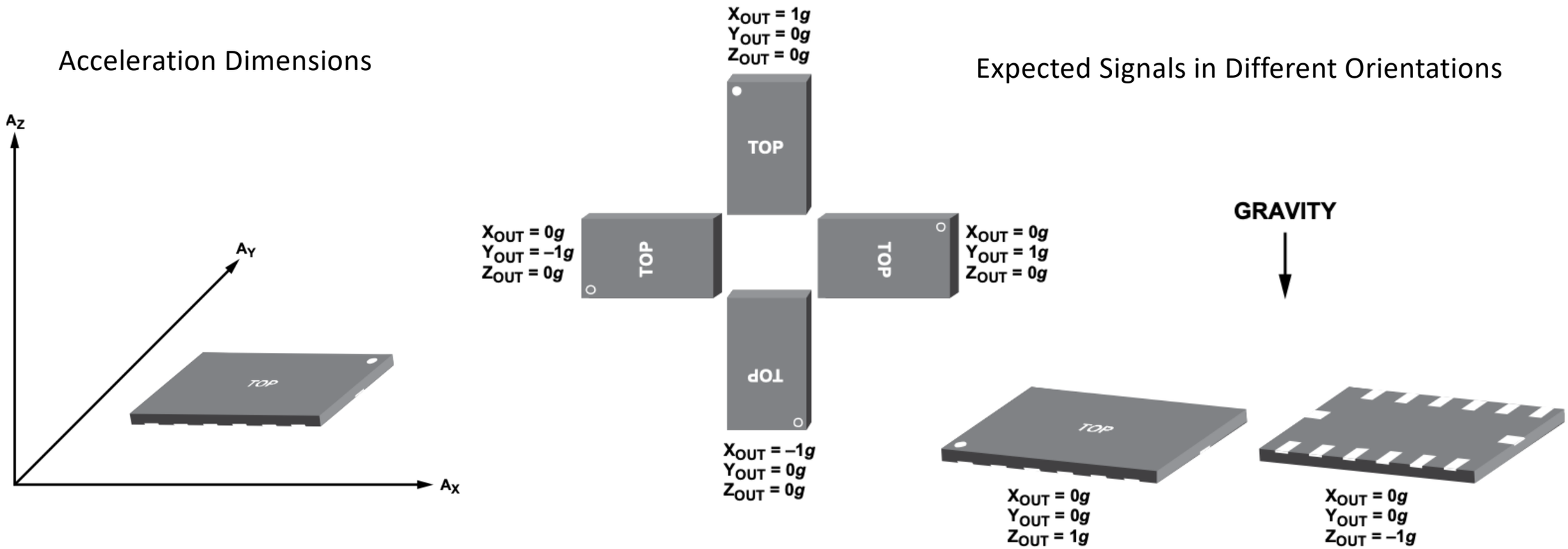


Figure 48. Output Response vs. Orientation to Gravity

ADXL343 Data Sheet

Example feature – double tap

Why?

- Simplifies detection
- Allows device to interpret data, not processor

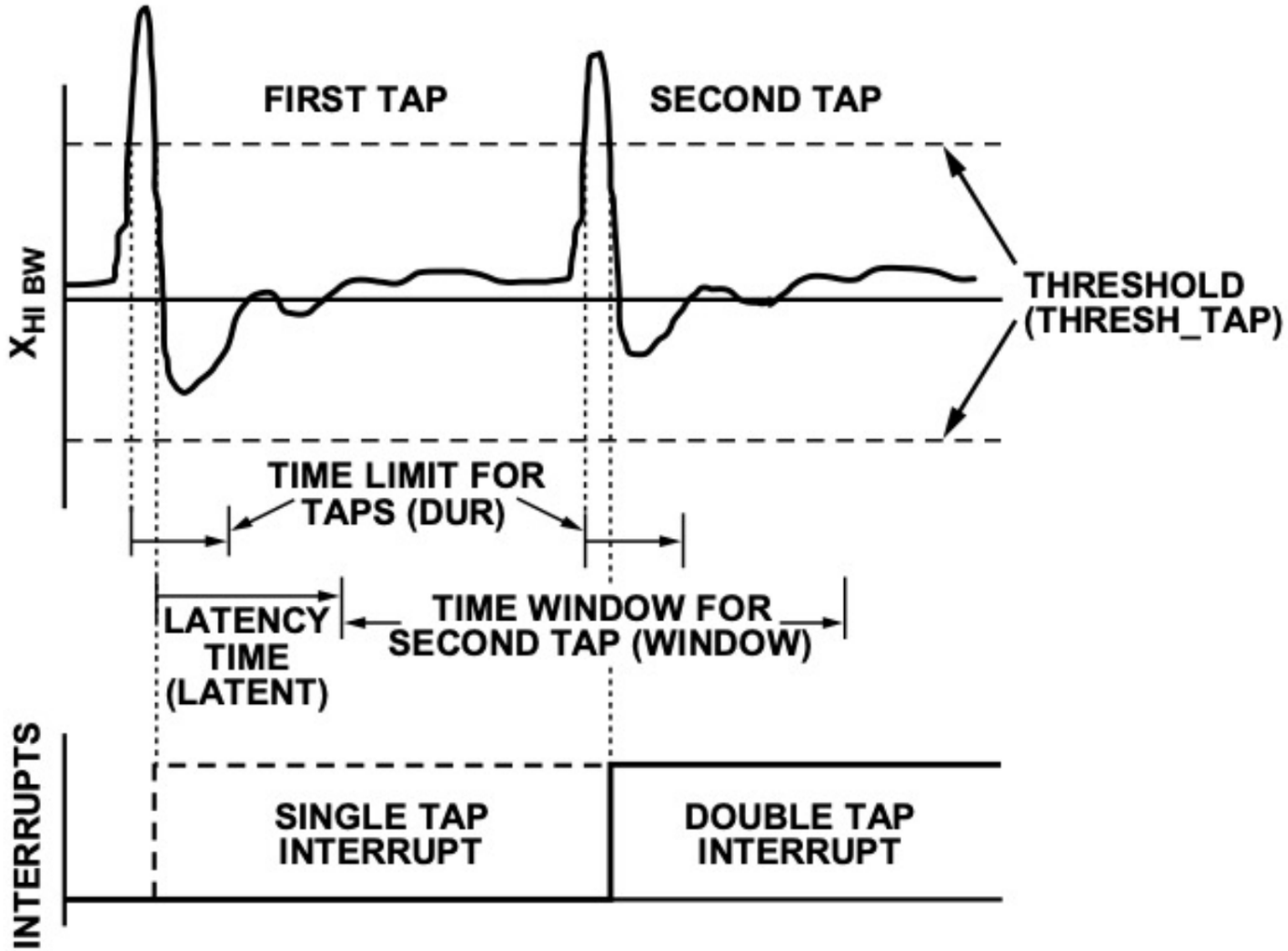


Figure 36. Tap Interrupt Function with Valid Single and Double Taps

Skills: Node.js and Javascript

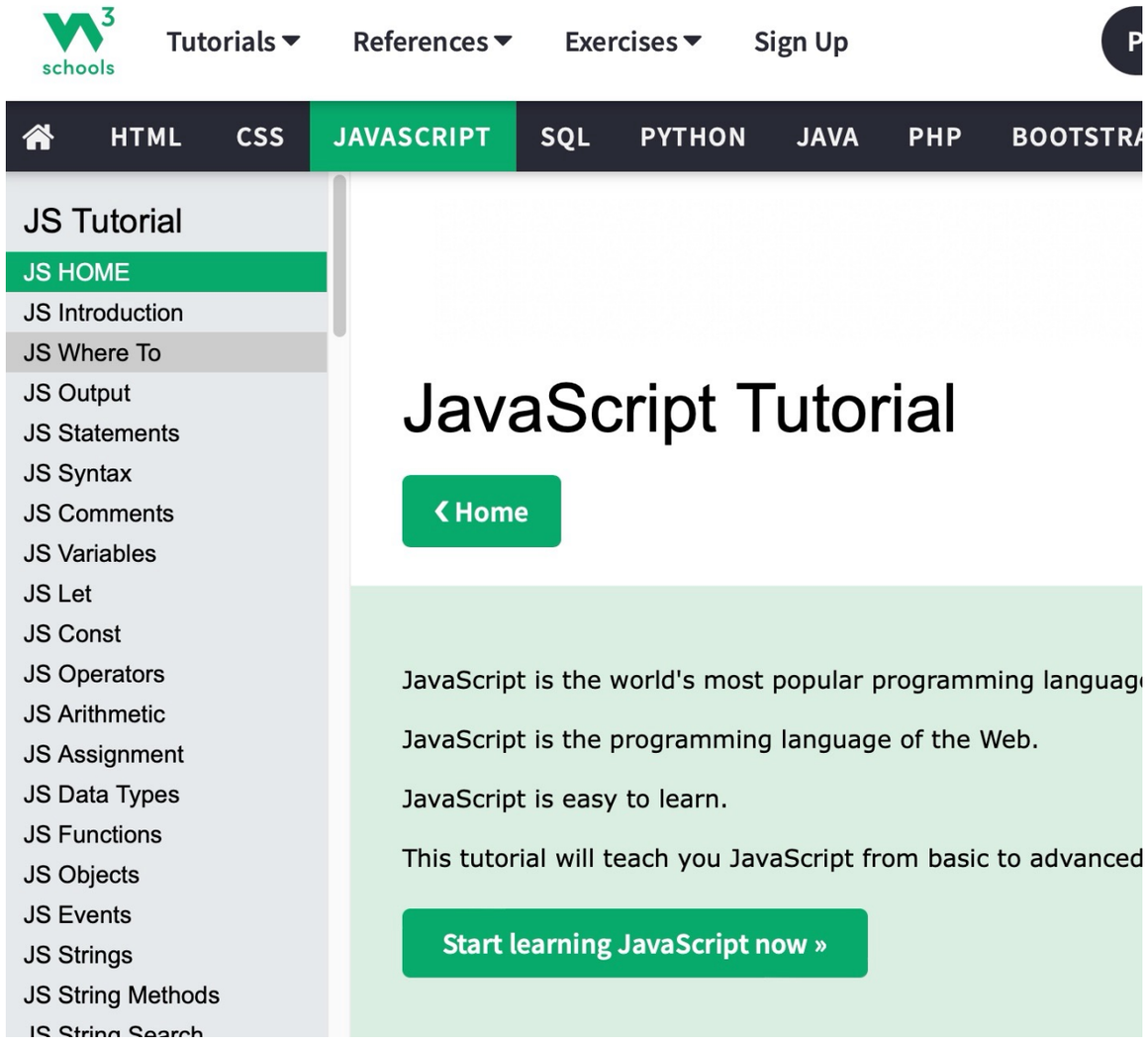
Javascript

- **Javascript** is a scripting language that allows you to create complex interactive features on a web site
- We typically associate javascript with user interface ('front-end') or 'client-side' programming
- Javascript is part of three standard web technologies
  - HTML – text markup language (headers, tables, image embedding)
  - CSS – style rules for pages (colors, fonts, columns etc.)
  - Javascript – interactivity
- Tutorial for javascript: https://www.w3schools.com/js/default.asp

- Use **node.js** for running scripts on the 'back end' or 'server-side'
  - Code back-end (server, database, applications, access to files) in javascript
  - Exploit many many developed modules – not need to reinvent the wheel
  - https://www.npmjs.com

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

Javascript tutorial

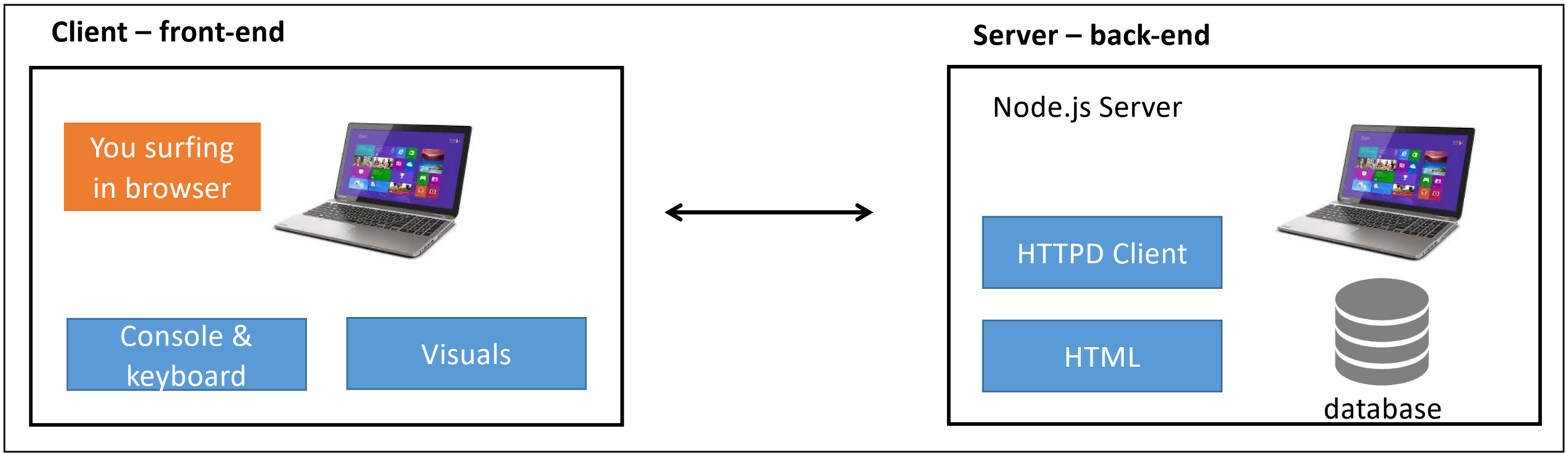


Node.js

- In contrast, we use **node.js** for running scripts on the ‘back end’ or ‘server-side’
- Code back-end (server, database, applications, access to files) in javascript
- Exploit many many developed modules – not need to reinvent the wheel
- https://www.npmjs.com

- Both the ’front end’ and the ‘back end’ can run javascript

Relationship between front-end and back-end



URL links to all the stuff needed to browse and interact

Javascript runs in browser here

The focus of node.js is this end

Javascript runs in node here

Node.js

- Node.js is a server environment that lets you run Javascript on your server
  - Code back-end (server, database, applications, access to files) in javascript
  - Exploit many many developed modules – not need to reinvent the wheel
  - https://www.npmjs.com
  - Lets you code front-end stuff and back-end stuff all in javascript

- Node.js is asynchronous meaning **code does not run sequentially**
  - You can force sequencing with callbacks and events

- A typical Node.js project will have three components:
  - The node program itself which has a file extension `*.js`
  - The `node_modules` folder for installed modules
  - The `package.json` file

* Java script object notation -- JSON

Example – http server – from the tutorial

```
var http = require('http');   /* uses http module */

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World!');
}).listen(8080);
```

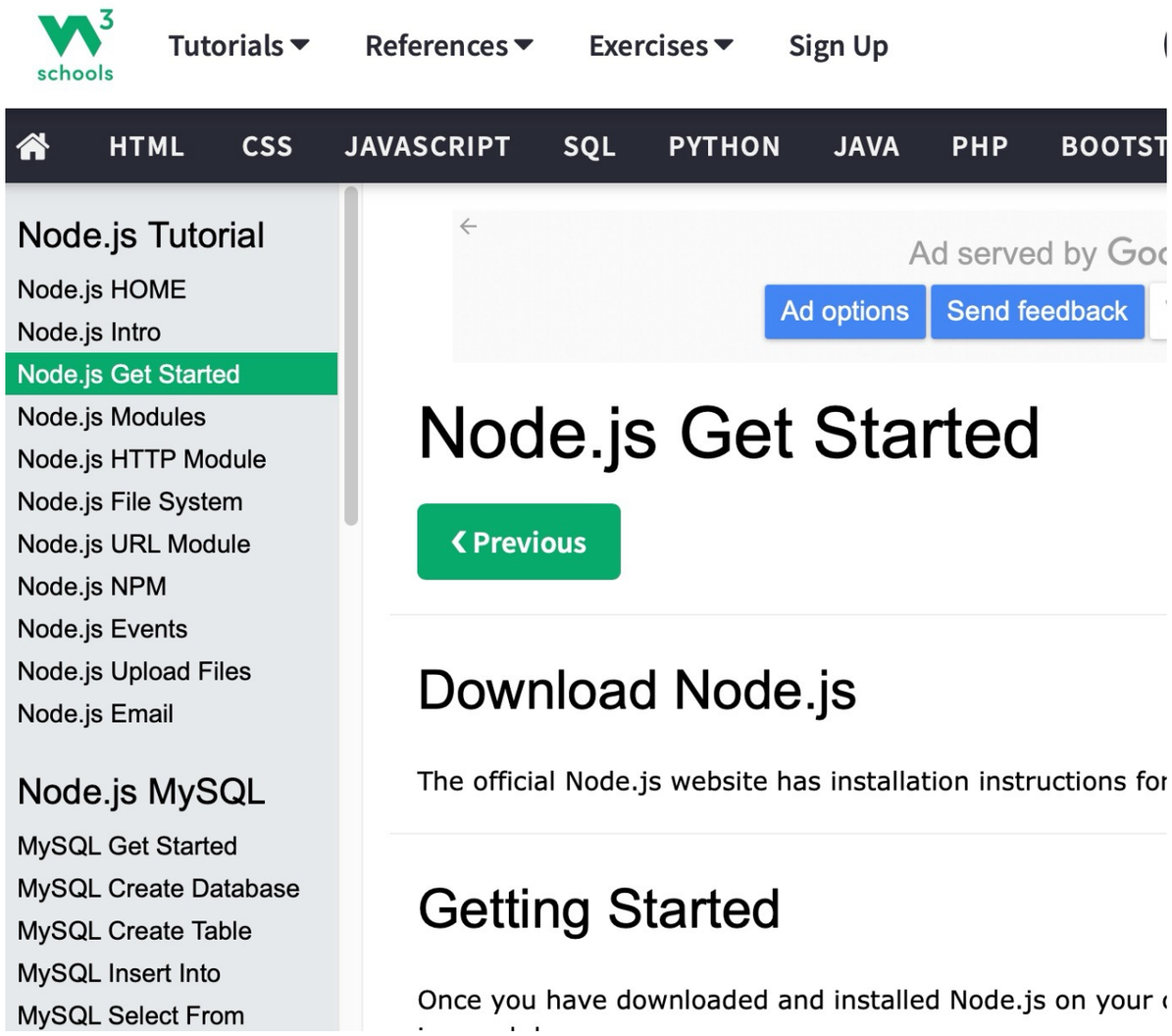
Then hit http://localhost:8080

Hello World!

Node.js / javascript skill

- Download the appropriate version of node.js for your operating system https://nodejs.org/en/
- Install node.js (this includes npm -- node package manager)
- Complete the hello world tutorial here: https://www.w3schools.com/nodejs/nodejs_get_started.asp
- Complete the following tutorial units on your local machine:
  - Getting started
  - Modules
  - HTTP module
  - URL Module
  - NPM
  - Events
- **And do this:**
  - Choose any sensor from prior skills and print sensor readings to the serial port
  - Create a node program to read from the serial port and print sensor data to console
  - Use: https://www.npmjs.com/package/serialport

Node downloads and tutorial



Node Modules

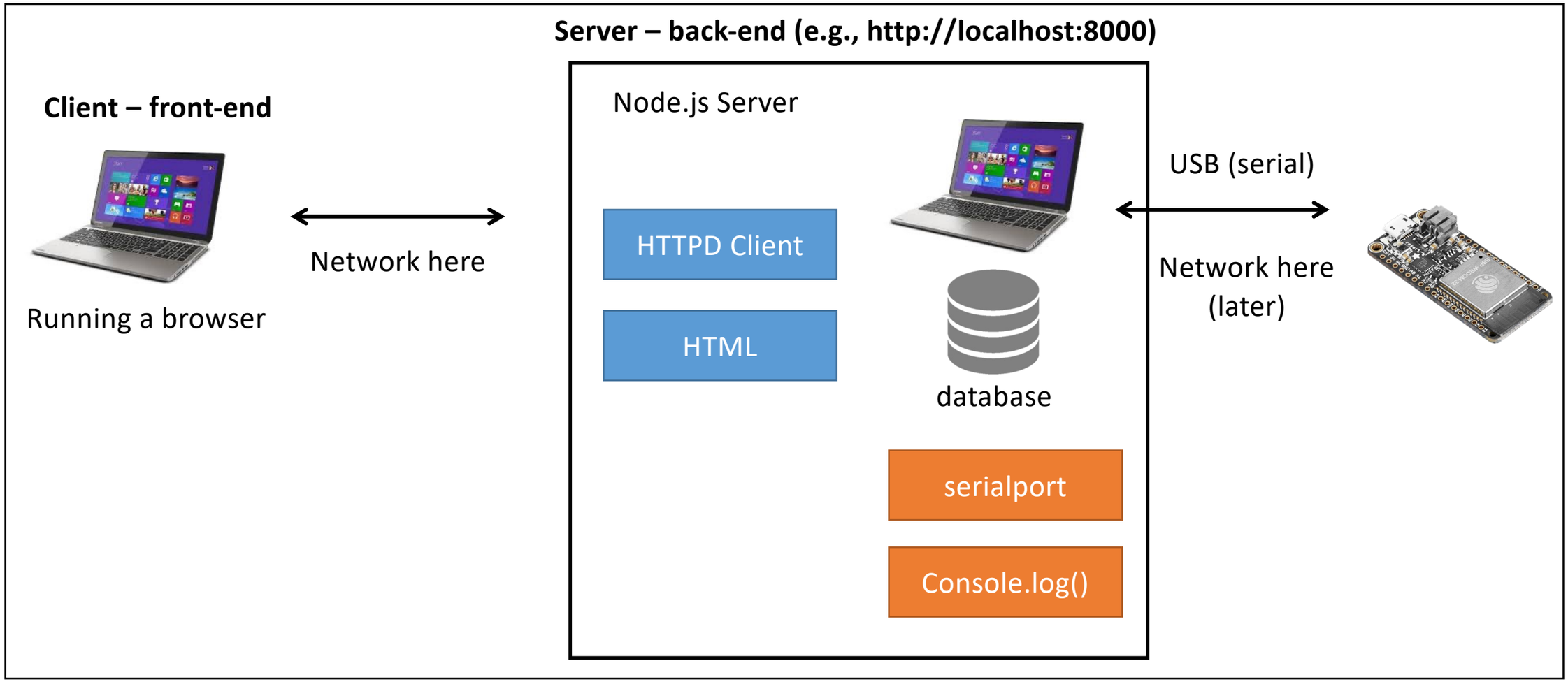
- In your main .js program, you would include modules like you would include libraries in c like so:
  - `var app = require('express')();`
  - `var http = require('http').Server(app);`
  - `var io = require('socket.io')(http);`

- You also need install the external modules you are using by running within your directory npm (node package manager)
  - `npm install module-name`
  - This will create a `node_modules` folder with your installed modules
  - There is more information on creating a `package.json` file in the Node.js brief on our site which is highly beneficially in managing your modules especially when you start porting onto the pi.

- Run your program with this command ('stuff' is target)

```
node stuff.js
```

What about talking to the ESP32? Use serial communications



This connects your ESP to your programs running on the node server → then accessible as web pages

Using serial port module

- Install

```
npm install serialport
```

- Instantiate

```
const {SerialPort} = require('serialport’)
const port = new SerialPort(path: '/dev/tty-usbserial1', {
baudRate: 57600 })
```

- Usage
  - Similar behavior in terms of the UART communication for the ESP32
  - https://serialport.io/docs/guide-usage
- Note!
  - ESP32 USB programming uses same hardware port

Serial port – flashing ESP32 and communicating with ESP32 for data

Steps to be successful:

- Flash the ESP32 (uses USB port to do this) with your program that uses the UART
- **Don’t run monitor – just idf.py -p flash**
- If you run monitor, the data will be delivered to that console/shell (ok for testing)
- **Using a different console/shell**: Start node.js with your node application

- The example code provided “serial-esp-to-node-serialport” includes a node.js application and ESP32 application that work as a pair

ESP32 serial data sent to node server application

On ESP32 side

```
int num = 0;
    while(1) {
      num++;
      printf("Count is: %d\n", num);
      vTaskDelay(500 / portTICK_PERIOD_MS);
    }
```

On node side

```
// Read the port data
port.on("open", () => {
  console.log('Serial port now open');
});
parser.on('data', data =>{
  console.log('The word from ESP32:',
data);
});
```

Output on node console

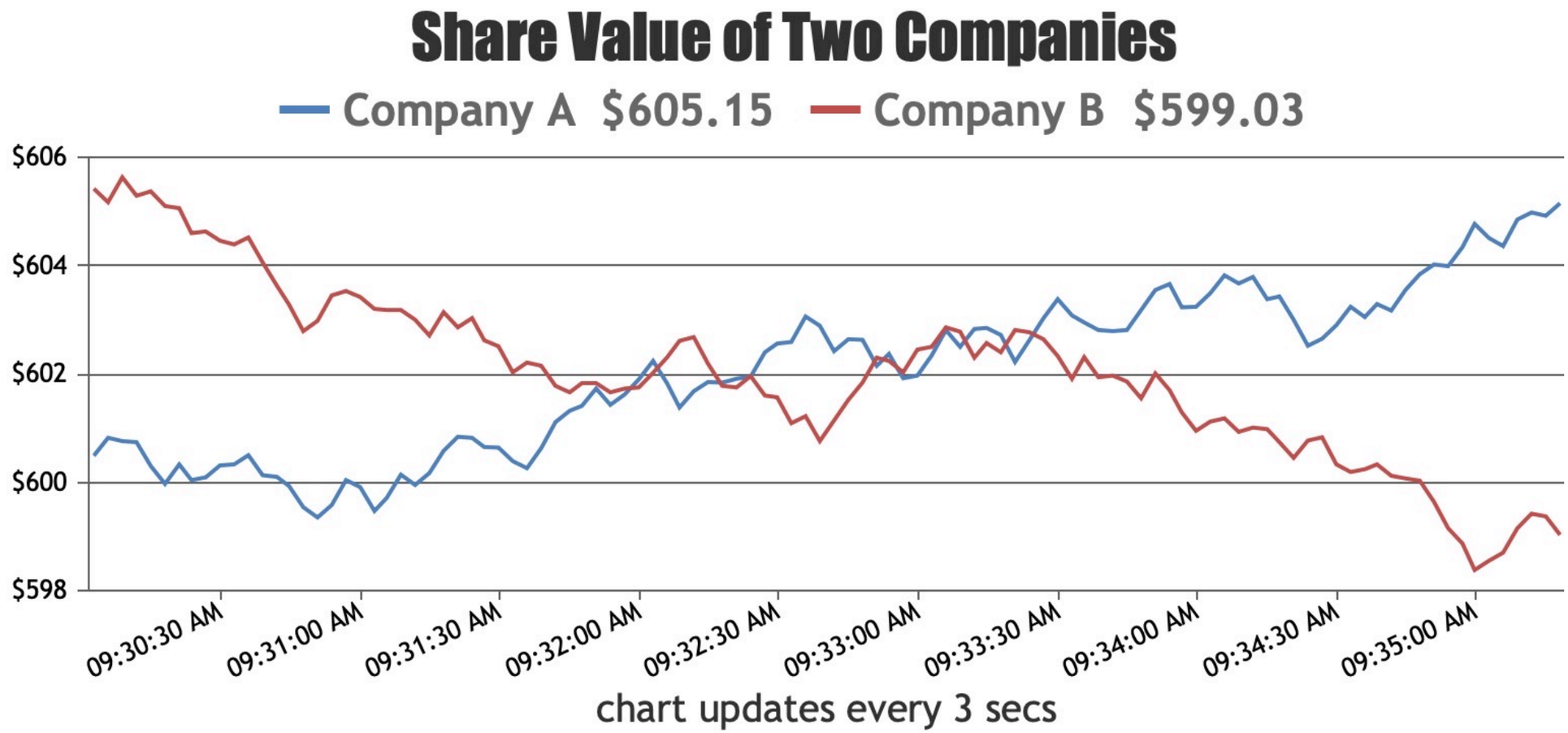
```
Serial port now open
The word from ESP32: Count is: 62
The word from ESP32: Count is: 63
The word from ESP32: Count is: 64
The word from ESP32: Count is: 65
The word from ESP32: Count is: 66
The word from ESP32: Count is: 67
The word from ESP32: Count is: 68
The word from ESP32: Count is: 69
The word from ESP32: Count is: 70
The word from ESP32: Count is: 71
The word from ESP32: Count is: 72
The word from ESP32: Count is: 73
The word from ESP32: Count is: 74
The word from ESP32: Count is: 75
```

Skills: CanvasJS

CanvasJS

- Library of graphing elements to simplify visualization of data
- Supports graphing in many chart types plus bonus features
  - Zooming, panning, animation, drilldown etc.
- CanvasJS download: https://canvasjs.com/download-html5-charting-graphing-library/
- Note: you can download the canvasjs modules or reference them online. Download if you do not have access to the network
- Basic charts: https://canvasjs.com/javascript-charts/line-chart-data-markers/
- “Live” charts: https://canvasjs.com/javascript-charts/dynamic-live-multi-series-chart/

CanvasJS


Share Value of Two Companies
Company A $605.15
Company B $599.03
$606
$604
$602
$600
$598
09:30:30 AM
09:31:00 AM
09:31:30 AM
09:32:00 AM
09:32:30 AM
09:33:00 AM
09:33:30 AM
09:34:00 AM
09:34:30 AM
09:35:00 AM
chart updates every 3 secs

Live chart

```
<!DOCTYPE HTML>
<html>
<head>
<script>
window.onload = function () {

var dataPoints1 = [];
var dataPoints2 = [];

var chart = new CanvasJS.Chart("chartContainer", {
        zoomEnabled: true,
        title: {
                text: "Share Value of Two Companies"
        },
        axisX: {
                title: "chart updates every 3 secs"
        },
        axisY:{
                prefix: "$"
        },
        toolTip: {
                shared: true
        },
        legend: {
                cursor:"pointer",
                verticalAlign: "top",
                fontSize: 22,
                fontColor: "dimGrey",
                itemclick : toggleDataSeries
        },
```

How do I update the chart with ESP32 data?

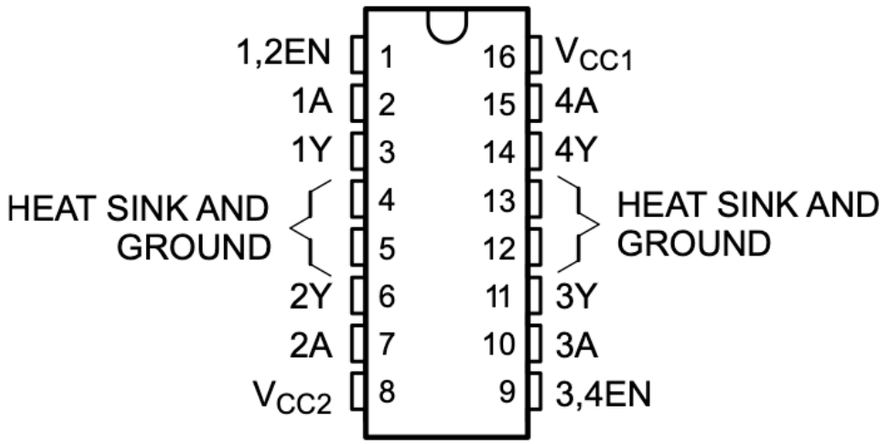
- At this point we are collecting data from the ESP32 via serial port/UART on an attached laptop
- Data can be displayed to the console, written to a file, or consumed by your local node application
- Data are sent over in the format you indicate, at the times you indicate
  - You can pack your data to be convenient for the receiving app – depending on the data from the sensors e.g., [header,x,y,z,aa,bb,cc,terminator].
  - A header and terminator are helpful for data validation. Sometimes CRC or parity is added for additional robustness (discard anything that fails a check)
  - You can set the timing of the sends
- ESP32 push or node pull?
  - ESP32 push: your ESP program sends data at regular interval and the node app listens and deals with it. Some data may get discarded (watch out for garbled data)
  - Node pull: the node app queries the ESP for data; the ESP responds with current data

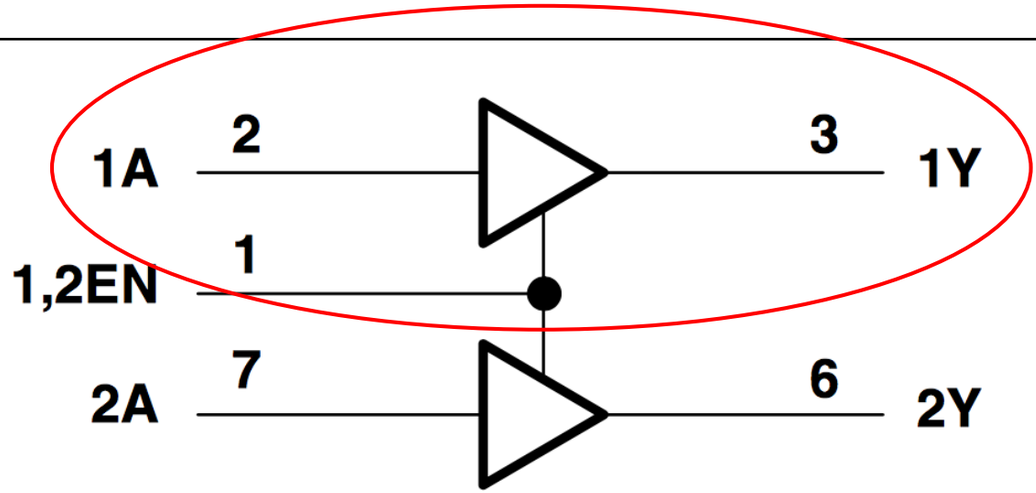
How do I update the chart with ESP32 data?

- Time reference
  - Node server has better access to absolute time – use to log received time stamps
  - ESP time is accurate locally as time intervals, but absolute time is not a sure thing without access to NTP
- Getting data to canvas options
  - Log console data to a file, then read data into canvas (simple)
  - Read from serial port directly into canvas
  - Use the canvass data format
  - Update at regular intervals
- Notes:
  - Watch out for bad data that will hang your program – discard items that are malformed
  - Later we replace the serial port with a network port

Micromotor (buzzer)

Use LD293D Driver (H driver) for micromotor buzzer





Vcc1: 3.3V
Vcc2: 3.3V
GND: GND
1,2EN: tie to logic 1 (3.3V)
1A: from GPIO pin
1Y: to positive lead of buzzer

Connect buzzer between 1Y and GND

Turn on with GPIO = 1