The University Of Sheffield

Sensing and Responding

COM3505, Lecture 4

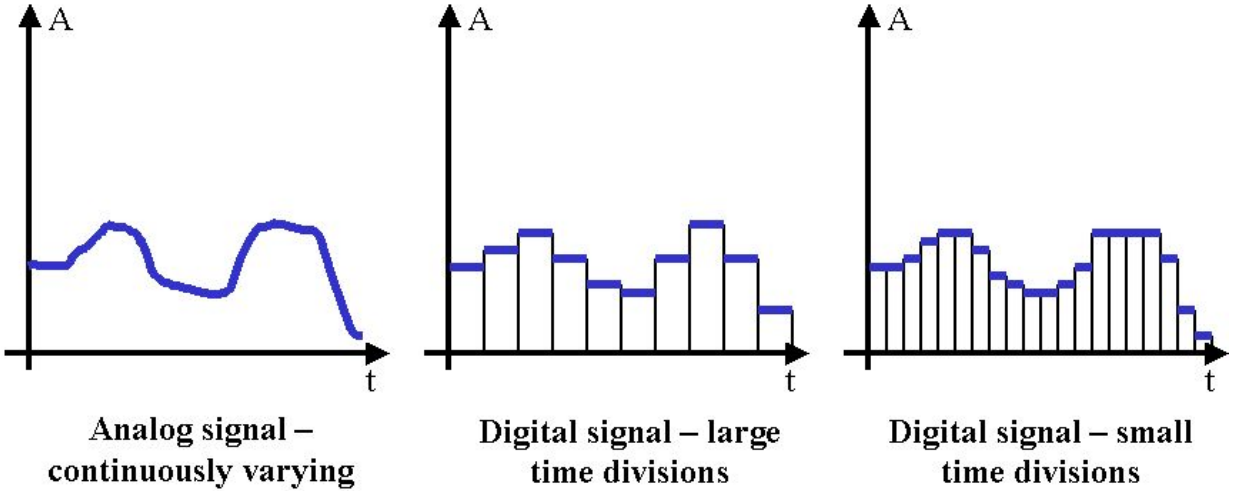
Gareth Coleman

The University Of Sheffield

Sensors

- A sensor is an electronic component designed to mesh the physical and digital worlds by converting a physical phenomena into a signal which can be measured electrically.
- Sensors can be broadly classified as either analog or digital, and either active or passive.
- An analog sensor produces a continuous output signal whereas a digital sensor produces a discrete binary output signal.
- An active sensor requires an external power supply to generate an output reading and a passive sensor does not.

Analog and Digital



From: www.rpi.edu/dept/phys/ScIT/InformationTransfer/sigtransfer/signalcharacteristics.html

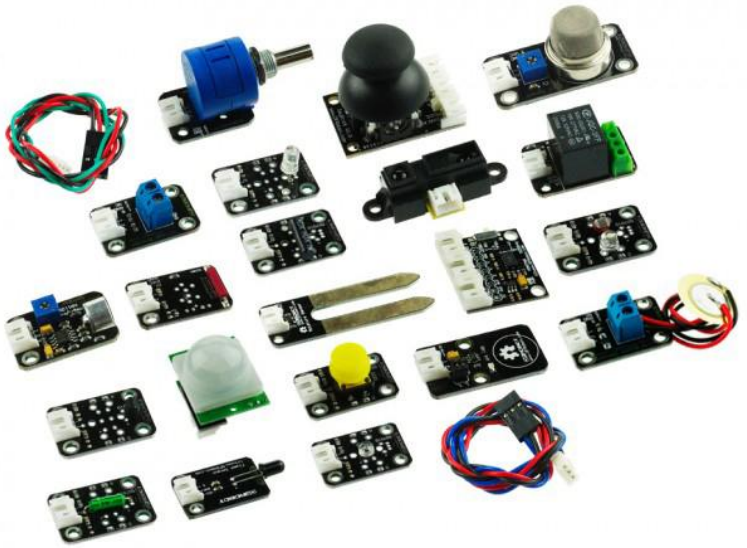
- When we want to turn an analog signal into a digital one we need to think about both:
  - the time resolutions (sample rate) - e.g. 44.1KHz
  - the amplitude resolution (number of bits) - e.g. 16bits for 65,536 different levels

Two light sensors

- Both of these light sensors convert the amount of light received into an electrical signal.
- A Cadmium-sulfide (CdS) cell is a passive, analog sensor that changes its resistance based on how much visible and IR light it receives.
- A TSL2561 sensor (the small chip in the centre of the blue breakout board) is an active, digital sensor. It gives a readout of brightness in Lux using the I2C digital protocol.

Much sensors

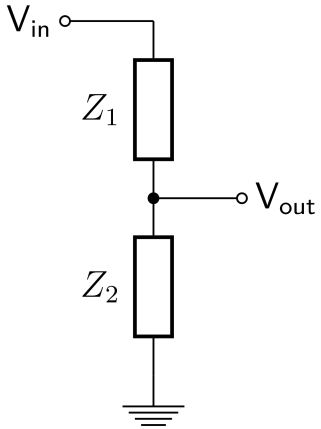
- Many hundreds of different sensors are available
- Measuring everything you can think of and a few others besides
- They range from a few pennies to many millions



Sensor example: Gieger Müller Tube

Old school but still cool — analog sensors

- Analog voltages can be measured directly by the ESP32
- We use the 0-1v range
- We can easily expand the range to say 0-5v using a potential divider
- Reducing the input range (0-0.1v) requires active amplification — fun but not on the menu today



$$V_{\text{out}} = \frac{Z_2}{Z_1 + Z_2} \cdot V_{\text{in}}$$

Potential divider schematic and formula from Wikipedia

Divide and conquer

Example: convert 0-5V input voltage to 0-1V

R2 = Vout*R1 / Vin-Vout

R2 = R1 / 5-1

R2 = R1 / 4

R1 = R2 * 4

Resistors are easily available in certain values - called E12 or E24 series

We just need to find two resistors where one is 4 times the size of the other, i.e. 3K and 12K

Online tools do the maths, choose best fit, etc

www.ti.com/download/kbase/volt/volt_div3.htm

$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2}$$

$$(R_1 + R_2) * V_{out} = V_{in} * R_2$$

$$V_{out} * R_1 + V_{out} * R_2 = V_{in} * R_2$$

$$V_{out} * R_1 = V_{in} * R_2 - V_{out} * R_2$$

$$V_{out} * R_1 = (V_{in} - V_{out}) * R_2$$

$$R_2 = \frac{V_{out} * R_1}{V_{in} - V_{out}}$$

From: electronics.stackexchange.com

"Talk is cheap. Show me the code."

```
const int sensorPin = A0;      // select the input pin for the analog reading
short sensorValue = 0;      // 16 bit integer to store the value coming from the sensor

void setup() {
  Serial.begin(115200);          // initialise the serial monitor at 115200 baud
  pinMode(sensorPin, INPUT);    // set the sensor pin to be an input
}

void loop() {
  sensorValue = analogRead(sensorPin);   // take analog reading and store
  Serial.println(sensorValue);    // print the value on the serial monitor
  delay(1000);    // wait for 1000ms (= 1 second)
}
```

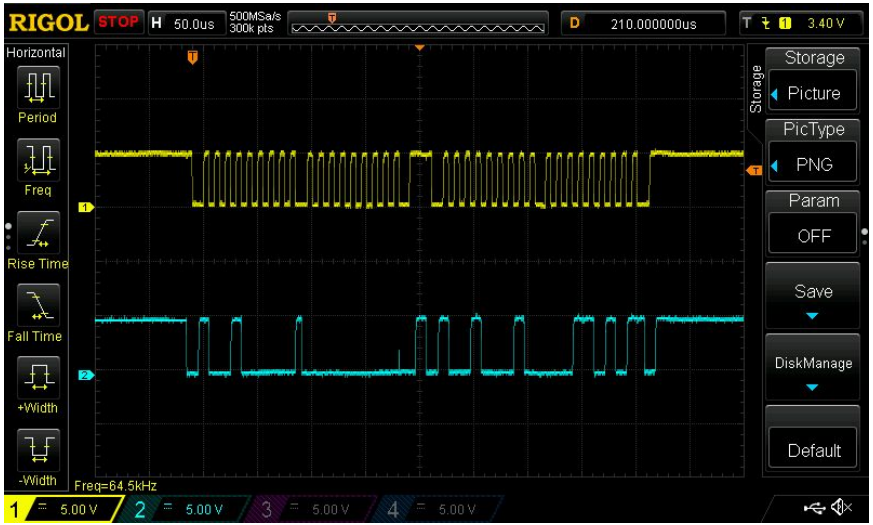
The University Of Sheffield

Hang on, using an int for an analog voltage?!

- Analog voltages cannot be dealt with directly by digital devices, and are measured by means of an Analog to Digital Converter (ADC)
- An ADC samples the analog voltage and converts it into a digital number
- An ADC has a voltage range and a number of bits, in our case on the ESP32 is 0-1V and 12bit. (Other ranges are available but aren't very linear.)
- 12 bits means the digital number reported is between 0-4095 (2^12 = 4096)
- Note that this limits the **resolution** of our measurements to ≅ 0.25mV

Digital sensors

- A digital sensor uses discrete steps to represent the measurement.
- The measured value is communicated as a binary number.
- Digital signals can be received by the ESP32 — either as a single input or as part of a more complex protocol such as I2C (pictured)



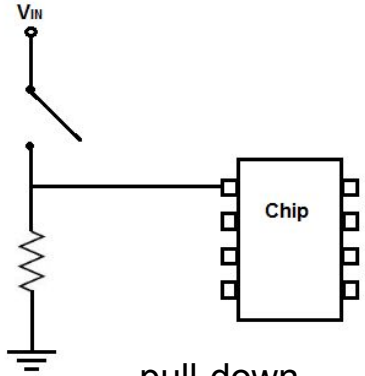
from: rheingoldheavy.com

Less talk, more code

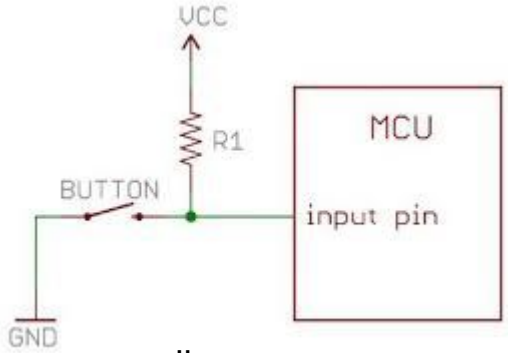
```
const int buttonPin = 12;      // select the input pin for the button
bool buttonState = 1;    // variable to store the value coming from the sensor

void setup() {
   Serial.begin(115200);    // initialise the serial monitor at 115200 baud
   pinMode(buttonPin, INPUT_PULLUP);   // set sensor pin to be input with pullup
}

void loop() {
   buttonState = digitalRead(buttonPin);    //  read the digital input pin and store
   if (buttonState == false) …;
     // do stuff here when the button is pressed: reverse logic i.e. false=pressed
}
```

Pull me up, pull me down

- Inputs don't 'like' to be left unconnected or 'floating' — this results in fluctuating signals and can cause other, intermittent & difficult to diagnose problems
- We can add a resistor — a moderately high value stops the input floating, but doesn't interfere with normal operations (5-10K is often used)
- When the resistor is connected to ground the resistor is called a pull-down resistor
- When connected to the positive supply voltage, it is a pull-up resistor



pull-down



pull-up

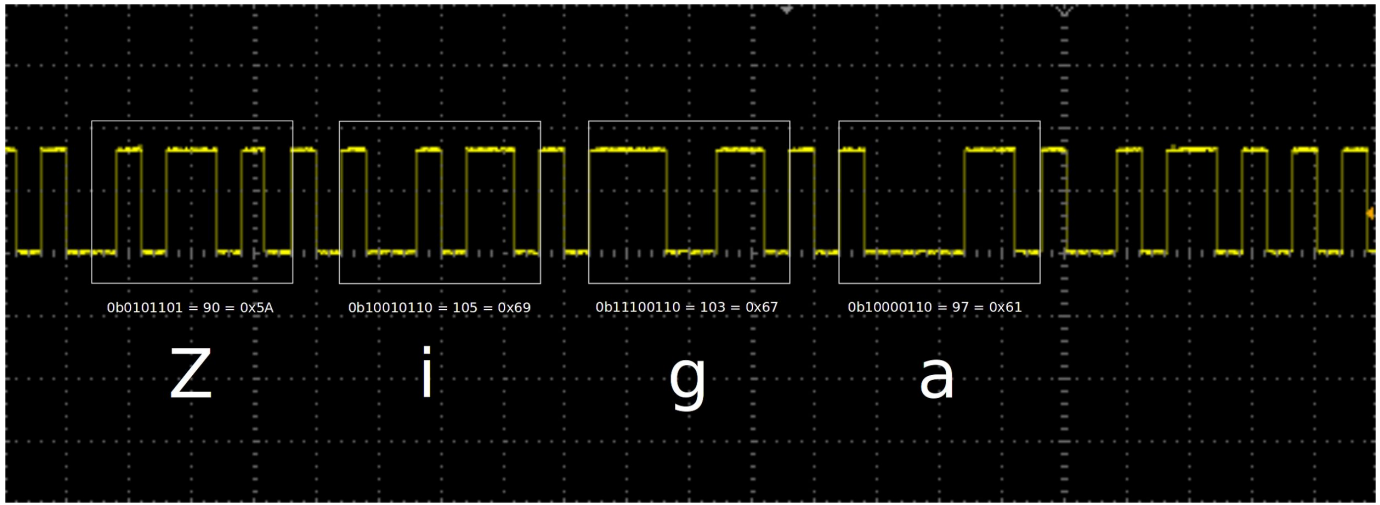
The University Of Sheffield

Vcc by any other name would smell as sweet

- Confusingly there are several different terms for the positive and negative voltages that power electronics
- Positive supply voltage is often called VCC but you will also see it referred to as Vin, V+, Vs+ and VDD
- Negative supply voltage is often called VEE or VSS, Vs- or V-
- Most often with digital circuits the most negative voltage is zero volts - we don’t work with bidirectional currents (AC)
- So the negative supply voltage is also Ground, GND or 0V

UART, the simple protocol

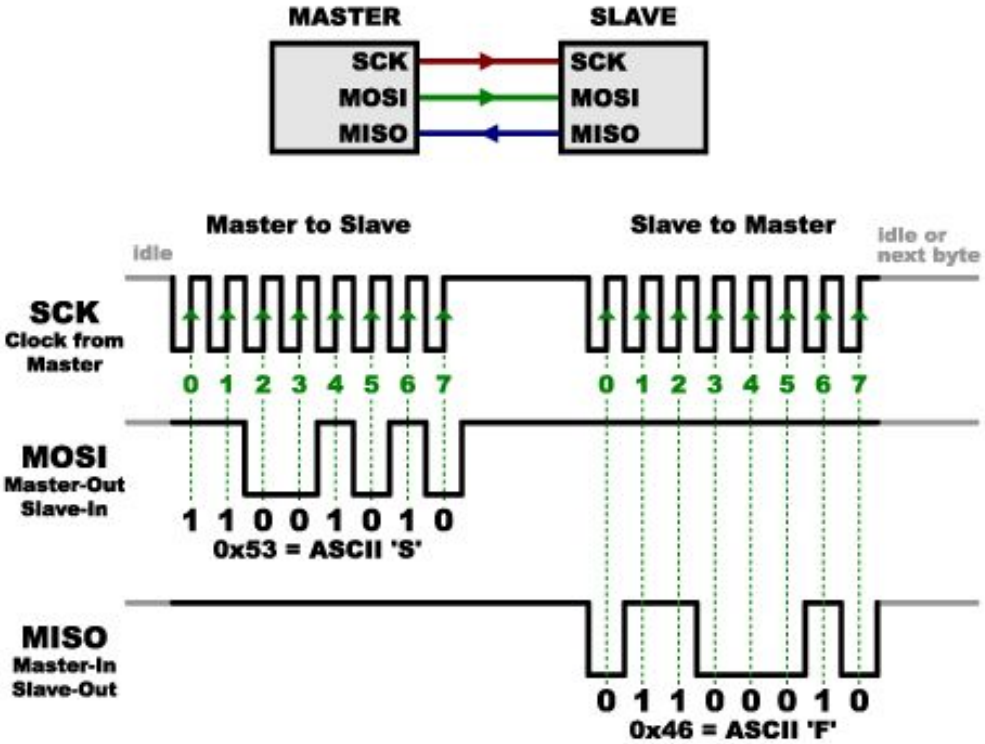
- Digital highs and lows are the basis for communications — often formalised into a protocol
- A simple example is a serial port (UART)
- The most common setup is to have a single start bit (LOW), 8 data bits and then a stop bit (HIGH).
- Each group of 8 data bits is a byte that represents a character in ASCII



From: electronics.stackexchange.com

SPI - a bit more complex

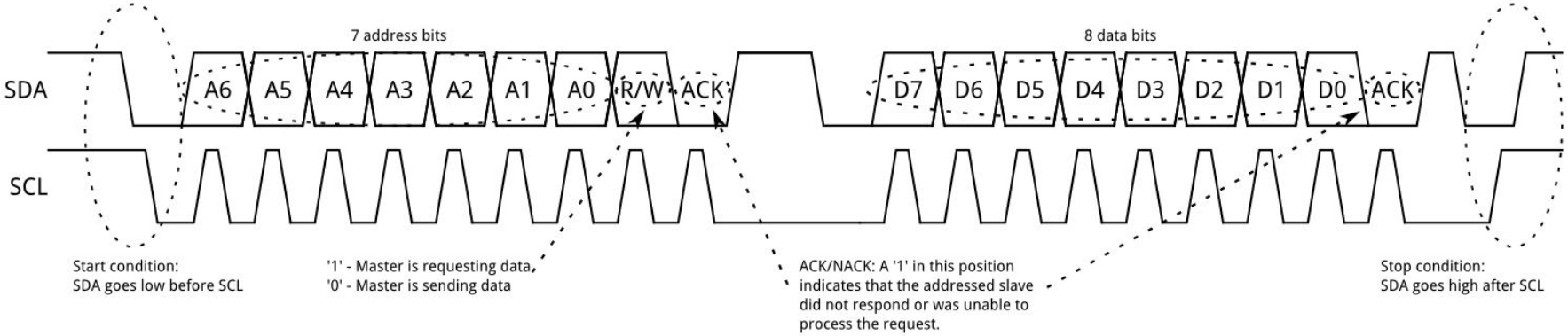
- SPI is synchronous — it has a clock line in addition to the communications lines
- This means that communications can be much faster than using a serial protocol
- In addition to these lines, each device has a select line that is used to select the device that you want to respond to the data



From: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

I2C - more complex still!

- Uses two wires for a bus linking 100's of devices
- Slave devices are addressed by the master device using signals on the bus itself
- Each device listens to the bus and only responds when it is addressed directly



From: https://learn.sparkfun.com/tutorials/i2c

Good news! You are standing on giants' shoulders...

- So you (mostly) don't need to worry about the details of implementation of start and stop bits, acks, bit order etc (at least, until stuff starts going wrong…)

- You have already used the built-in, pre-included library to print to the serial monitor:

```
Serial.begin(115200);    // initialise the serial monitor at 115200 baud
Serial.print("Hello world");    // simple printing characters on the serial port
```

- SPI and I2C are as easy to use (I2C example shown here):

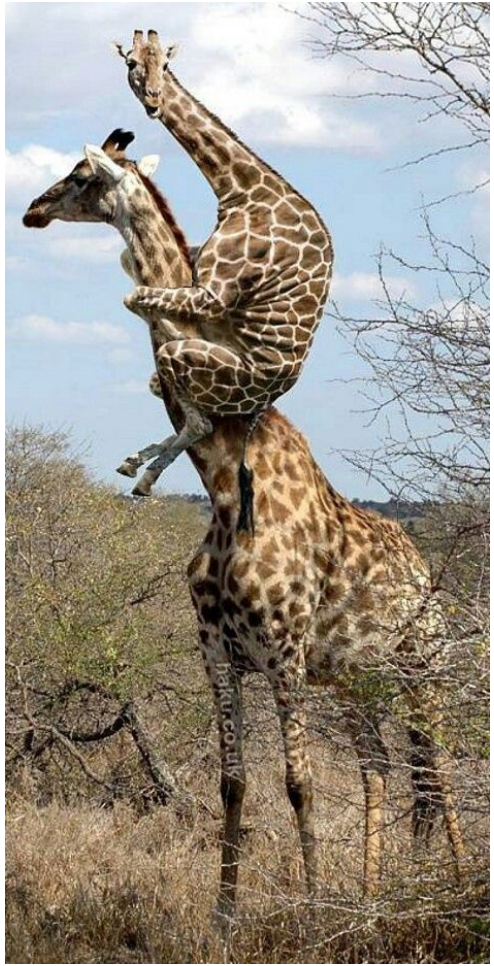
```
#include <Wire.h>    // include the library to support I2C protocol
Wire.begin();    // initialise the i2c port, on the ESP32 this uses pins 22 & 23
Wire.beginTransmission(44);   // transmit to device #44 (0x2c)
Wire.write(byte(0x20));    // write the value 20 in hex (32 in decimal)
Wire.endTransmission();     // stop transmitting
```

More good news - the giants are stacked!

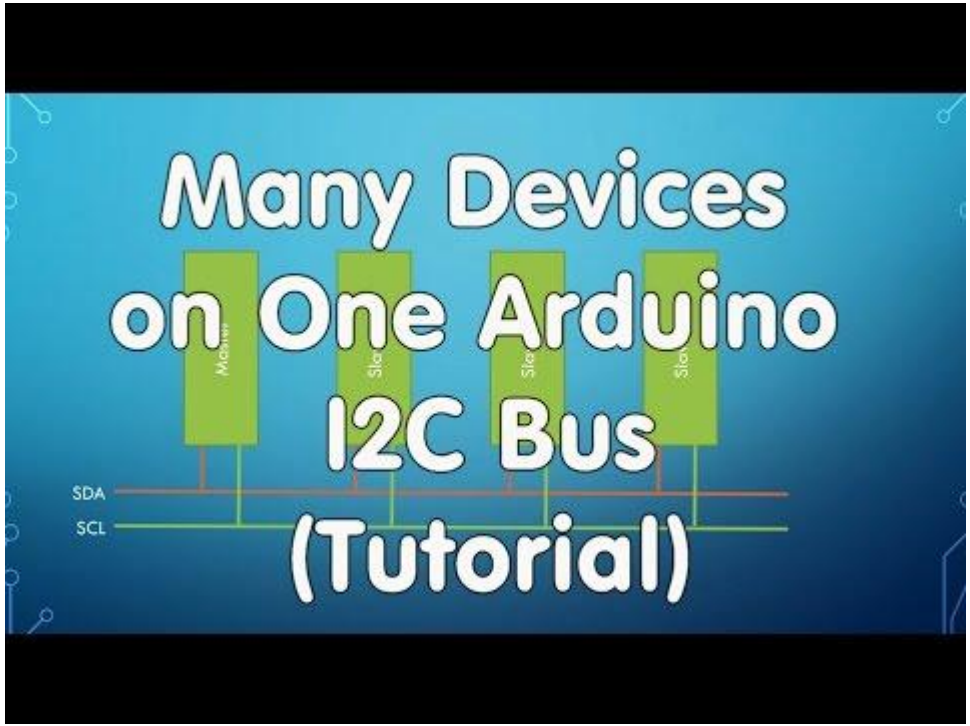
- Most of the time, you can use a pre-written library that deals with all of the communications details for you

- These provide functions such as readSensor() — this typically requests a reading from the device, gets the response and formats it for you

    - Adafruit library for the TSL2561 light sensor provides the functions setGain() and getLuminosity()
    - getDustDensity() is provided by the dust sensor library we'll be using later on in the course

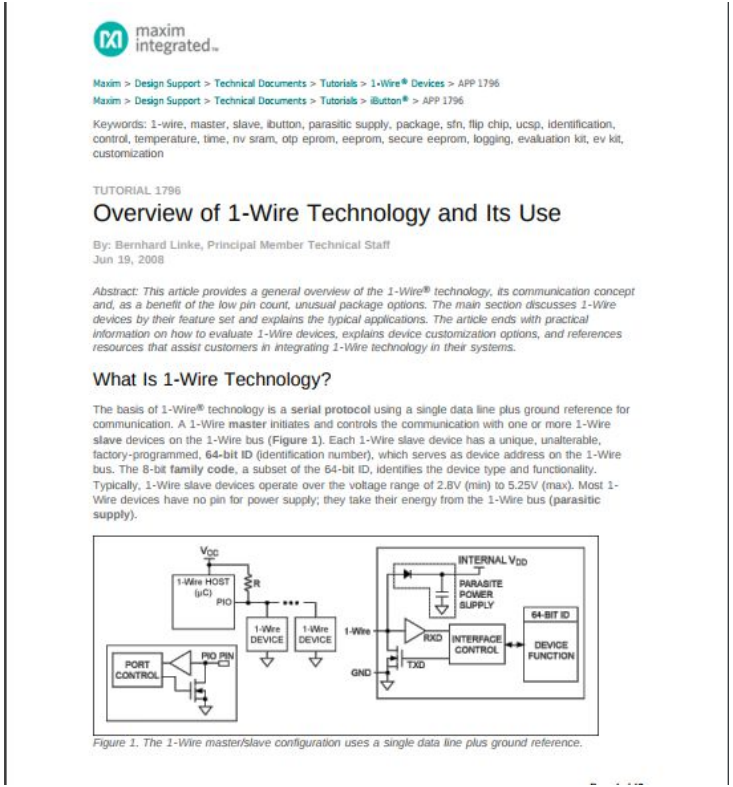


Video



1-Wire — similar to I²C

- Complex protocol, you should be aware of the details and where to find them if you need to …
- … but happily, a well tested, documented library is available for use with minimum effort
- Like I2C - 1-Wire allows multiple devices — this time with a single wire (plus ground)
- Lower data rate than I2C but longer cable (up to 50m+ vs 1-2m)
- Used by Apple MagSafe power supplies, Dell and others to exchange signals
- Now my laptop won't charge unless I have an approved charger
- For my own comfort and safety



maxim integrated

Maxim > Design Support > Technical Documents > Tutorials > 1-Wire® Devices > APP 1796

Maxim > Design Support > Technical Documents > Tutorials > iButton® > APP 1796

Keywords: 1-wire, master, slave, ibutton, parasitic supply, package, sfn, flip chip, ucsp, identification, control, temperature, time, nv sram, otp eprom, eeprom, secure eeprom, logging, evaluation kit, ev kit, customization

TUTORIAL 1796

Overview of 1-Wire Technology and Its Use

By: Bernhard Linke, Principal Member Technical Staff
Jun 19, 2008

Abstract: This article provides a general overview of the 1-Wire® technology, its communication concept and, as a benefit of the low pin count, unusual package options. The main section discusses 1-Wire devices by their feature set and explains the typical applications. The article ends with practical information on how to evaluate 1-Wire devices, explains device customization options, and references resources that assist customers in integrating 1-Wire technology in their systems.

What Is 1-Wire Technology?

The basis of 1-Wire® technology is a **serial protocol** using a single data line plus ground reference for communication. A 1-Wire **master** initiates and controls the communication with one or more 1-Wire **slave** devices on the 1-Wire bus (**Figure 1**). Each 1-Wire slave device has a unique, unalterable, factory-programmed, **64-bit ID** (identification number), which serves as device address on the 1-Wire bus. The 8-bit **family code**, a subset of the 64-bit ID, identifies the device type and functionality. Typically, 1-Wire slave devices operate over the voltage range of 2.8V (min) to 5.25V (max). Most 1-Wire devices have no pin for power supply; they take their energy from the 1-Wire bus (**parasitic supply**).

Figure 1. The 1-Wire master/slave configuration uses a single data line plus ground reference.

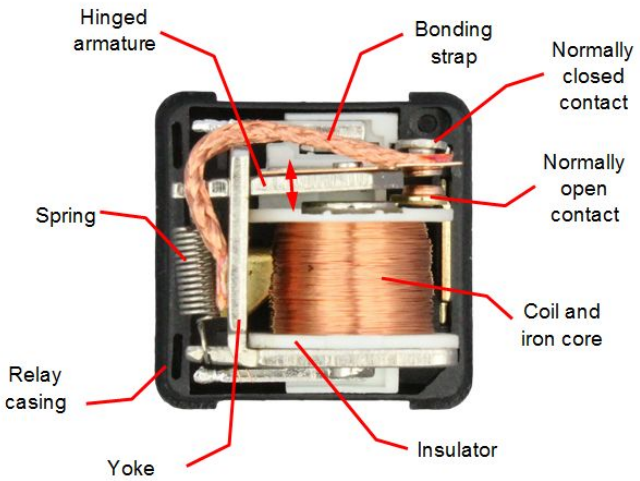
Actuators

- Broadly, these devices change the physical world in response to electrical signals
- Outputs can also be information — such as a tweet, SMS, record in a database etc.
- ESP32 has several kinds of outputs:
  - GPIO — simple kind
  - PWM — set to a timed ratio
  - DAC — create an analog voltage
- None of these outputs can supply much current — **12mA MAXIMUM**
- 12mA limit at 3.3V implies we need a resistance of at least 275Ω
- Often we need an amplifier or driver to supply the power the actuator needs

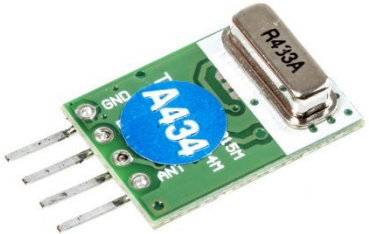
Relays and why they need snubbing

- If we want to switch a large current such as mains power we can use a mechanical device such as a relay
- A small amount of power operates an electromagnet
- The magnetic force causes the switch contacts to open and/or close
- The switch is completely separate from the electromagnetic coil
- When the coil power is removed, the magnetic field collapses very quickly, inducing a massive voltage
- We can protect against this voltage spike with a ‘snubber’ diode
- Other devices that operate with a coil such as solenoids and motors also need snubbing

From 12voltplanet.co.uk



Switching mains without a risk assessment



- We encourage our students to be adventurous in their learning….
- But **please** don't mess about with mains electricity ;-)
- Cunningly, we can 'hack' a remote control power socket to switch mains electricity
- We don't need to touch the dangerous stuff
- Instead, we send commands on the radio frequency (433MHz) that the sockets are tuned to
- Again, a pre-written library does all the hard work - e.g. mySocket.send(4281651,24);

What's the most important platform?

- Audience survey: "Which phone friendly medium or social media platform would ***most of your friends*** prefer to see data from IoT devices?"
    - Twitter
    - SMS / Text message
    - Facebook
    - Dedicated App
    - Re-open Nominations
- If Then Then That (IFTTT) supports hundreds of services, we'll be using it to interface with some of the platforms above and more...

Your **TODO** list for week 4:

- Update your git repository clone as usual
- Read and digest Notes/Week04.mkd
- Coding: wifi access points, web servers (Ex06); web page utilities (Ex07, optional)
- Do the reading
- Make sure you understood the lecture, and review the slides if needed

MOLE quiz tomorrow!

- **10.10am start, NC PC (you need to be there at 10!)**
  - **closed book: use Respondus Lockdown browser**
- **10.50 finish**
- **if you finish early either sit silent doing nothing, or leave the room**
- **11am standard lab session**