Internet of Things: Electronics

John L. Manferdelli

johnmanferdelli@hotmail.com

Nov 16, 2020, 18:00

Guide

- This is a rather condensed introduction to electronics mainly to give a sense of scope and establish terminology. There's more in my "Electronics of Radio" notes.
- A full treatment of this material could be a full time and rewarding hobby; in fact, many people have this hobby, see, for example, http://www.arrl.org.
- You should consult some standard works like the Radio Amateurs Handbook and Make Electronics by Charles Platt as well as the SparkFun Tutorials which are online.
- A basic background in physics would help a lot. See, for example:
  - Feynman's Lectures on physics.
  - Berkeley physics course volumes 1, 2, 3.
  - Any standard two-year physics course textbook (e.g.- Sears, Zemansky and Young).
  - Schaum's outline in Physics for Engineering and Science.
  - Griffiths, Introduction to Electrodynamics.
  - Scherz, Practical Electronics for Inventors. This book bridges theory and practice and is a bargain.

Electronics part 1, charge, current potential and circuits

- Charge and forces on charged particles are the basis for electronics. There are four things we can measure related to this. You should be able to define them and say how to measure them.
  - Electric field (E)
  - Magnetic field (B)
  - Current (I)
  - Voltage (V)
- How do you process electrical signals to produce useful output?
  - Currents and potentials are affected by “lumped components” like resistors, capacitors and transistors. You should know what they are and model their effect.
  - Circuits: Lumped components can be connected to implement complex relationships between input currents and potentials and output currents and potentials. You should be able to draw circuit diagrams (showing how the components are connected) and analyze the circuit.
  - Analyzing circuits: You should know what phasors are and model complex circuits using Thevenin and Norton equivalent circuits.
- Power
  - You should be able to define and calculate electrical power in a circuit.

Current and potential

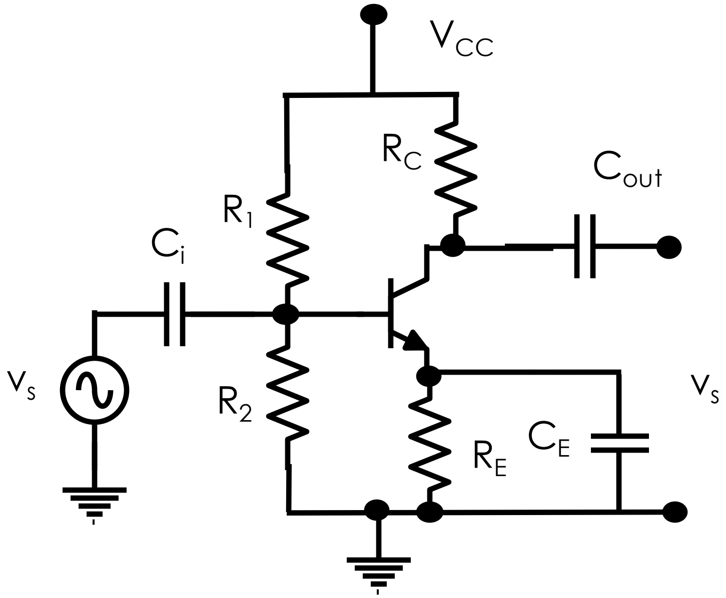
- A principled understanding of electronics requires an understanding of Maxwell's equations but that is not easy. We'll get into Maxwell's equations later, but it will likely not mean anything unless you have the right background. I'll start out here in the middle of the story with an "artzy-craftzy" description of charge and potential but let me say, if you're willing to start in the middle, you're a better person than I am.
- First, you need to accept the concept of an electric field, **E(r**,t). Bold means a vector in three space. Electric fields are created by charged particles. Charged particles come in two flavors, -, like an electron and +, like a proton. Charged particles experience a force when placed in an electric field. This force is proportional to their charge, namely **F**(**r**,t) = q **E(r**,t), where q is the charge. Accelerating charges, produce special electric fields that radiate. Electric fields have companion fields called magnetic fields. Magnetic fields are produced by *moving* charges. *Moving* charged particles also experience a force when placed in a magnetic field. In fact, the full force law is
  - **F**(**r**,t) = q **(E(r**,t) + **v** × **B(r**,t)), where × is the vector cross product.
- Charge is measured in Coulombs (C). One coulomb has the same (negative) charge as 6.24×10^{18} electrons.

Current and potential (continued)

- Current is the measure of charge flow through a a surface per unit time. One amp of current through a surface means one coulomb flows through the surface per second.
- Electric potential (V) represents the work done per unit charge when a particle accelerates in an electric field. It corresponds to the gravitation potential for the work done accelerating a particle with mass in a gravitational field except that voltage, like charge, can be positive or negative. The unit of potential is a volt. One volt is 1 kg-m^2/amp-sec^3.
- You can measure current or voltage with a volt-ohm meter.
- The potential between two points is not, in general, a constant and can vary with time. A simple potential source is an ideal battery which *does* have constant potential between its two "terminals."
- One basic law in electronics is ohm's law which says
  - $I = \frac{V}{R}$, or more generally, $I = \frac{V}{Z}$ where R stands for resistance and Z stands for impedance.
- The power consumed by a current, I travelling between a potential difference of V is $W = IV$.
- Until we talk about radiation (radios), most electronic ideas can be phrased in terms of I, V and Z.

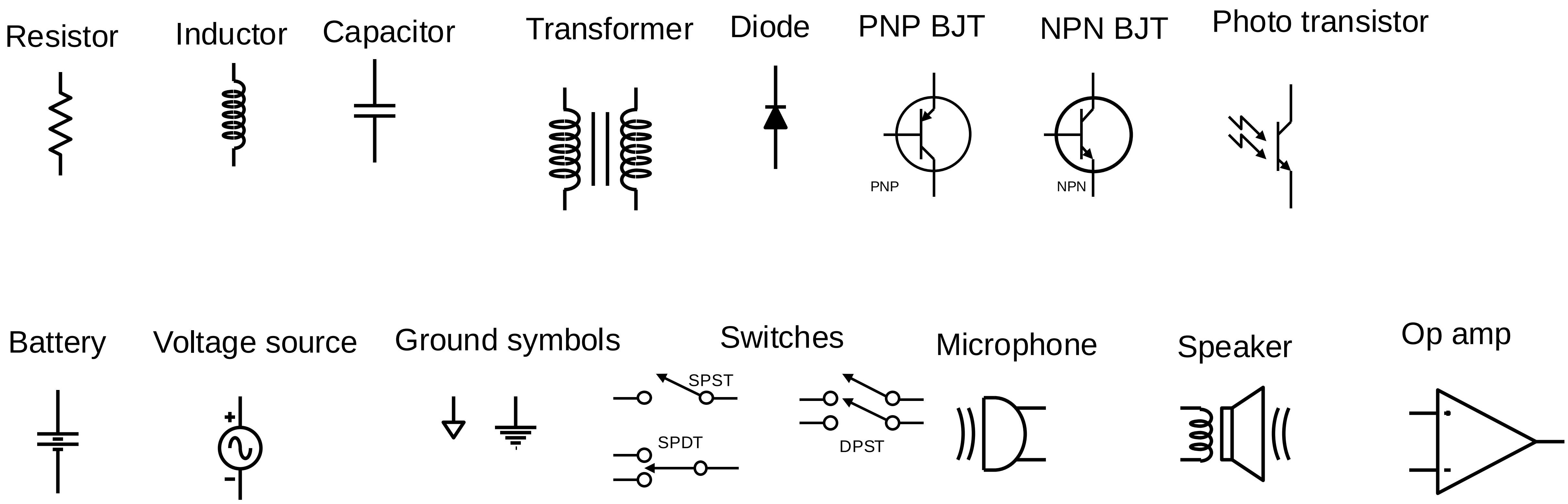
Circuit diagrams and lumped circuits

- At low frequencies, circuits can be analyzed as “lumped devices” with terminals wired together.
- The components (resistors, transistors, …) correspond to the nodes of a graph and the wires correspond to edges.
- There are potential differences between any two points and current flowing through edges. Each component has a recognizable symbol (see later) ; when drawn this way, we call the graph-like representation a circuit diagram.
- Here is an example:



Common emitter amp

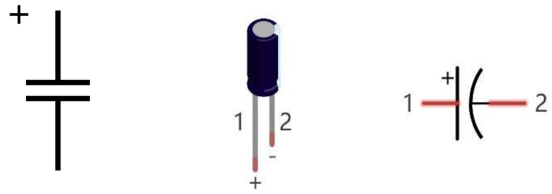
Circuit symbols



Symbols from http://www.eehomepage.com/symbols.ppt. Copyright 2008 by EE HomePage.com. These symbols may be freely reused.

Capacitors

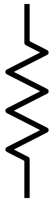
- Symbol 

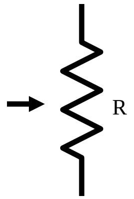
Electrolytic capacitor 

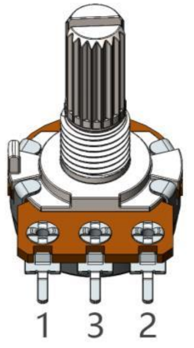
- Unit: farads. Usually capacitor values vary from microfarads (μF) to nano-farads (nF) to picofarads (pF). 1 μF= 10^{-6} F, 1 nF = 10^{-9} F, 1 pF=10^{-12} F.
- Law connecting charge (on a plate) and potential between plates: q = CV.
- The "impedance" version of this is $Z = \frac{1}{i\omega C}$
- Stored energy in a capacitor: $E = \frac{1}{2}CV^2$
- $I = \frac{dq}{dt} = C\frac{dV}{dt}$.

Resistors

- Symbol

fixed  

variable 

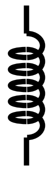


Credit: Make Electronics

- Unit: ohms (**Ω**). Usually resistor values vary from ohms to kilo-ohms (10^3 **Ω**) to megaohms (10^6 **Ω**).
- The resistor value (in ohms) is indicated by colored bands around the resistor.
- The inverse of resistance is conductance, $G = R^{-1}$.
- $IR = V$.
- $Z = R$
- Dissipated energy: $E = I^2 R$.

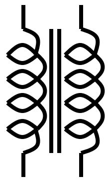
Inductors

Wikipedia

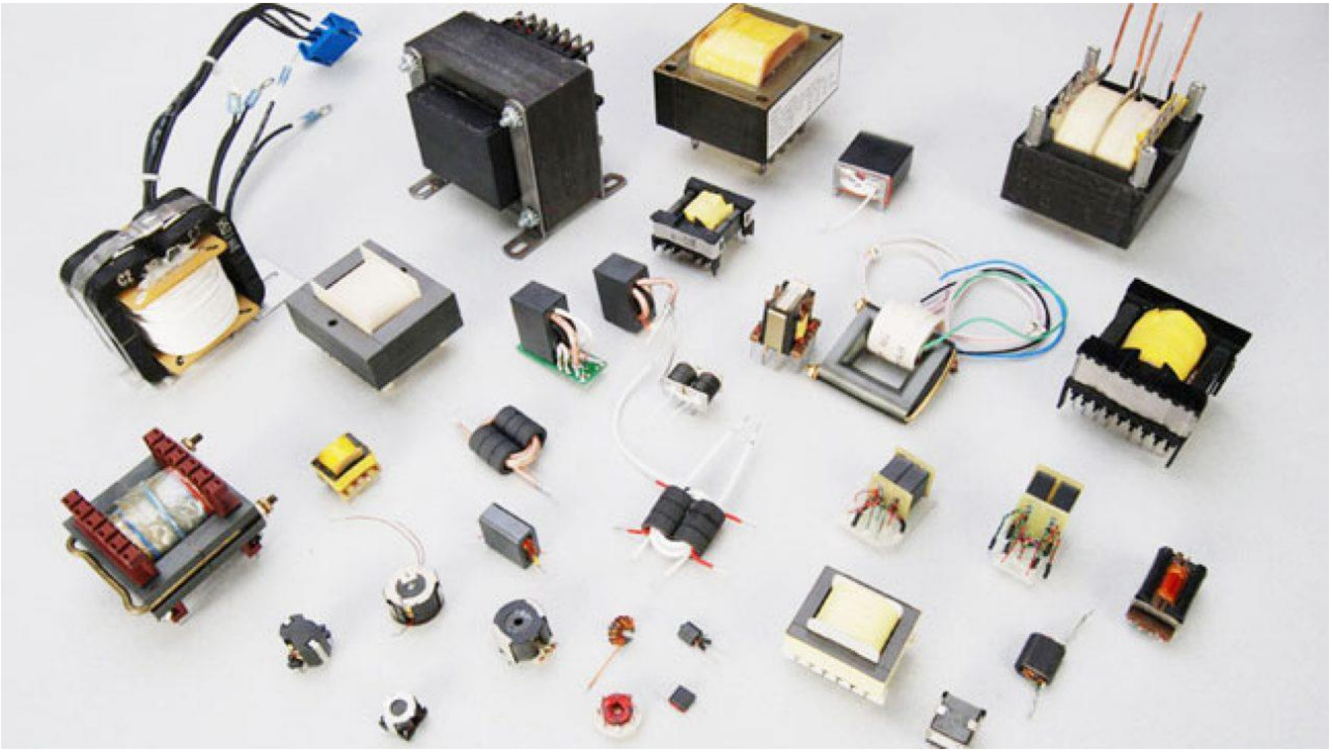
- Symbol 

- Unit: henries. Usually inductor values vary from microhenries to millihenries.
- $V = L\frac{dI}{dt}$
- $Z = i\omega L$
- Stored energy: $E = \frac{1}{2}LI^2$

Transformers

- Symbol 

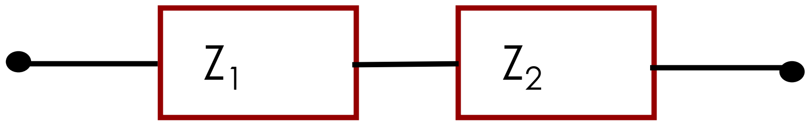
- AC only: $\frac{N_2}{N_1} = \frac{V_2}{V_1}$



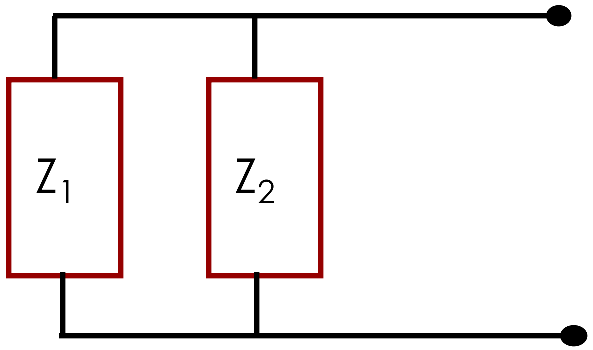
Electronics hub

Impedance

- Impedance captures both pure resistance and complex “reactance” effects like capacitance and inductance.
- $Z_R = R$. $Z_C = -i/(\omega C)$, $Z_L = -i\omega L$
- Here, ω, is the angular frequency of the signal and $\omega = 2\pi f$
- The inverse of impedance is admittance, $Y = Z^{-1}$.



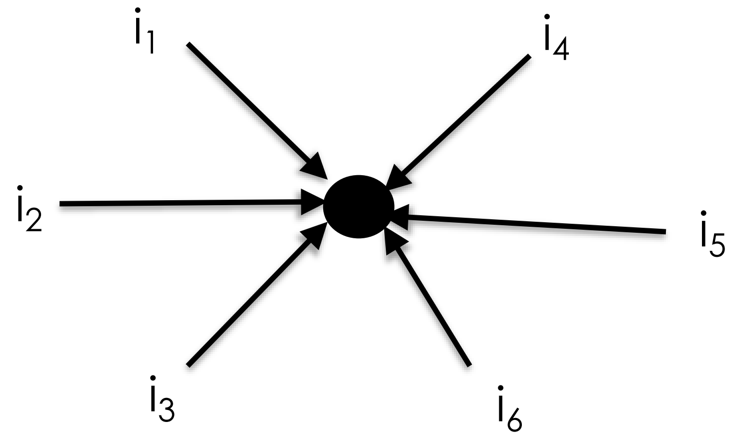
$$Z_{eq} = Z_1 + Z_2$$



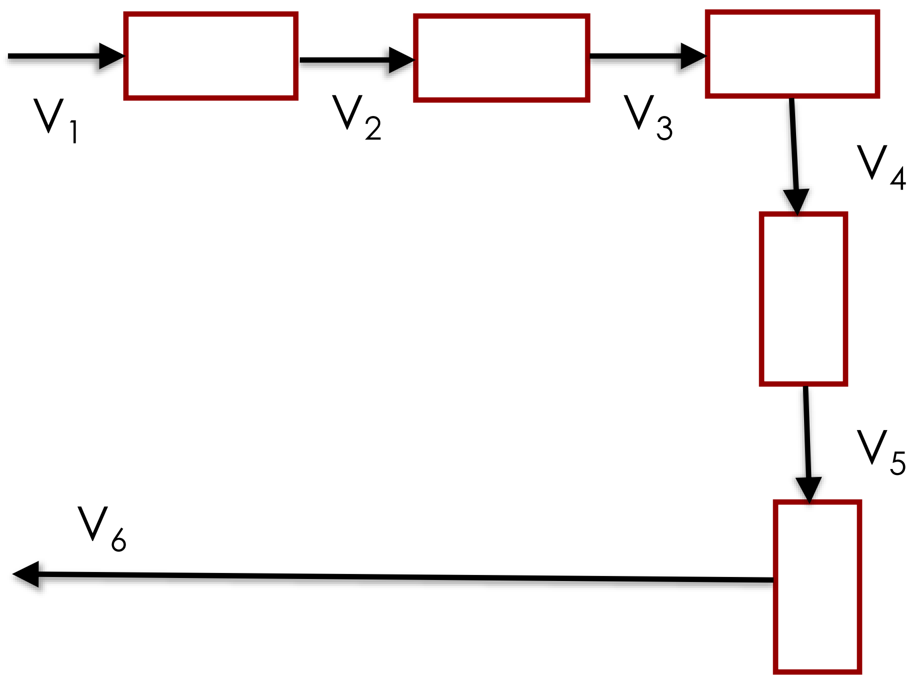
$$1/Z_{eq} = 1/Z_1 + 1/Z_2$$

Kirchhoff

There are two Kirkoff's laws, one describes voltages the other describes currents.



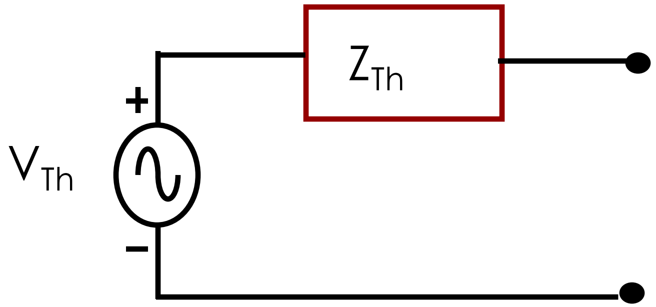
$$i_1 + i_2 + i_3 + i_4 + i_5 + i_6 = 0$$



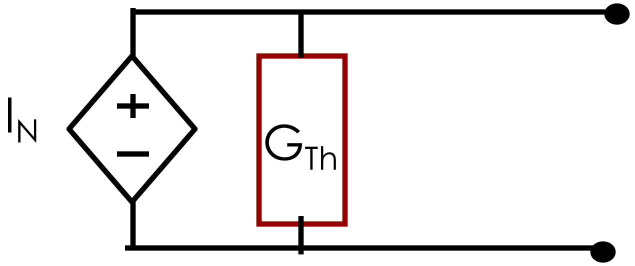
$$(V_2 - V_1) + (V_3 - V_2) + (V_4 - V_3) + (V_5 - V_4) + (V_6 - V_1) + (V_1 - V_6) = 0$$

Thevenin and Norton

- Thevenin: Any combination of *linear* sources and passive elements terminating in two terminals is equivalent to a pure voltage source in series with an impedance



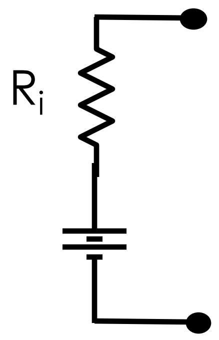
- Norton: Any combination of *linear* sources and passive elements terminating in two terminals is equivalent to a pure current source in parallel with an conductance



- Similar theorems for two terminal input and output linear devices (with transfer function)

Thevenin example: a real battery

- Real batteries have a Thevenin equivalent of an ideal battery in series with a pure resistance, R_i, called the internal resistance.
- The equivalent resistance is sometimes called the “lookback” resistance.



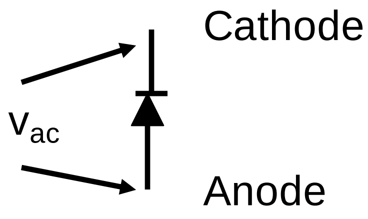
Phasors

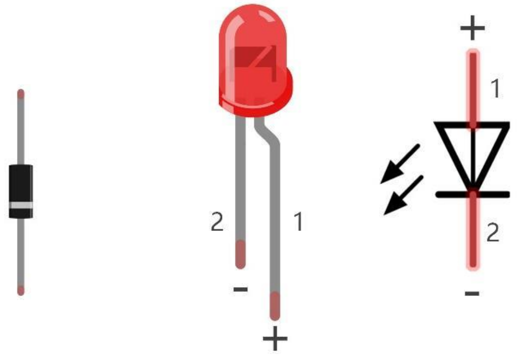
- Phasors are complex number-based representations of signals without the frequency component.
- If the signal is A $\cos(\omega t+\phi)$, the phasor representation is composed of, A, $\measuredangle$A where $\measuredangle A = \phi$ represented by $Ae^{i\phi}$. By the way, sometimes we use i (and sometimes j) to represent $\sqrt{-1}$. Neither is great since i can be confused with current and j with current density. Both can be confused with indexes. Oh well.
- We recover the frequency by multiplying by $e^{i\omega t}$.
- So, when representing a voltage, in phasor notation by $V = V_0 e^{i\phi}$, we recover the actual wave form as $\mathrm{Re}(Ve^{i\omega t})$.
- Average power can be calculated from the phasor representation of a *cosine wave* as VI*/2. Here * means complex conjugate.
- Where does the "2" come from?
  - Answer we need to integrate over the current and voltage cycle.
  - If "peak" voltage is V_0, the "average" voltage is 0 but we care about the magnitude, so we average over the square of the voltage and take the square root. This is called the "root mean square" voltage and is denoted V_{rms}. Incidentally, your "house" voltage of 120 VAC is actually the root mean square voltage.

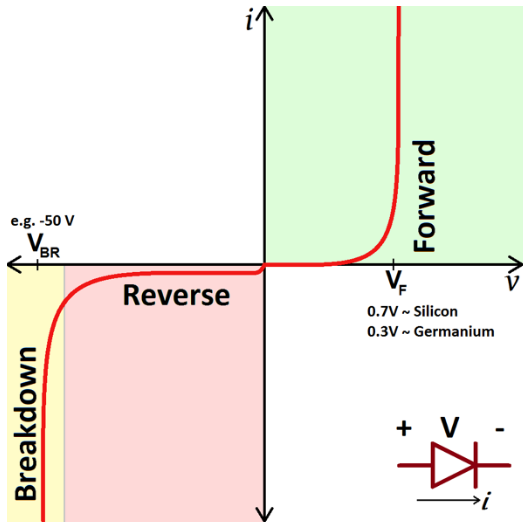
Diodes

Credit: Make Electronics

- Symbol





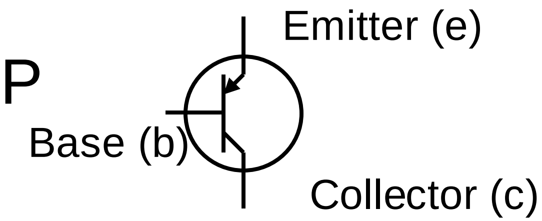


Sparkfun

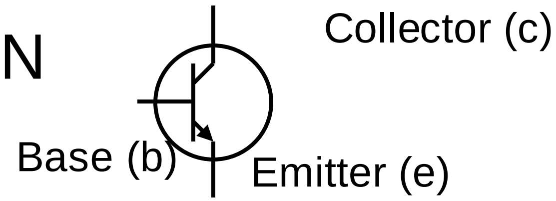
- Diodes are devices that allow current to flow only in one direction. Silicon diodes, for example have, essentially infinite resistance if V_{ac}<0, that is if the cathode is at a higher potential than the anode and very low resistance if V_{ac}> .7V.
- The cathode is usually labelled with a band.
- There are several types of diodes (e.g.-Zener) with slightly different characteristics.
- One recently popular diode (the last symbol above) is the light emitting diode which is a very efficient light source.

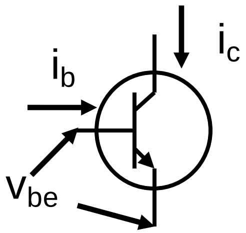
Bipolar Junction Transistors (BJT)

- Symbol PNP



NPN





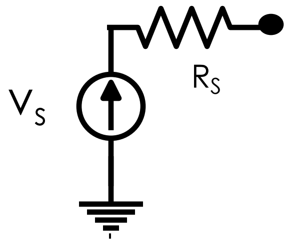
- Transistors are 3 terminal non-linear devices. They are constructed of one thin layer of semiconductor (Si) sandwiched by two thicker layers. In the case of an NPN transistor the middle layer is “doped” to have a slight excess of hole (positive charge carriers) and the outer layers are doped to have an excess of electrons (negative charge carriers).
- BJT’s have three operating regions that depend on the potential differences between the terminals.

Region	$\mathbf{V_{be}}$	$\mathbf{V_{bc}}$	$\mathbf{V_{ce}}$	$\mathbf{i_c}$
Active	$\geqq V_f$	$<V_f$	$>V_s$	βi_b
Reverse active	$<V_f$	$\geqq V_f$	$<-V_s$	$-(\beta_r+1)i_b$
Saturated	$\geqq V_f$	$\geqq V_f$	$>-V_s$ $<V_s$	$>-(\beta_r+1)i_b$ $<\beta i_b$
Off	$<V_f$	$<V_f$	*	0

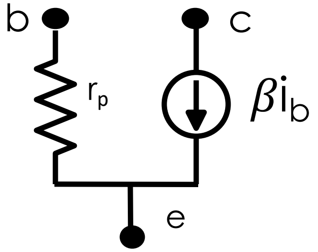
For silicon resistors (e.g.-2N3904): $V_f \sim .7$v, $V_s \sim .2$v, $\beta \sim 100$, $\beta_r \sim 10$

BJTs - continues

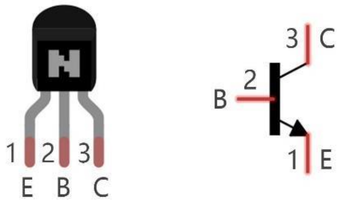
- We really care about two regions: the active region where the collector current is proportional to the base current (when $V_{be} > V_f$) and the cutoff region where there is no current flow.
- The transistor acts as a switch and is the basis of digital electronics where we care if it's "off" or "on" but don't care how on it is.
- Transistors can also be used as linear amplifiers when biased (when $V_{be}>V_f$) so that they are always in the active region.
- Transistors can be modeled at low frequencies by the following parameters you will find in their specs,
- $i_C = \alpha i_E$
- $i_C = \beta i_B$
- $\beta = \alpha/(1- \alpha)$
- $\beta \sim 100$



Bipolar source model



Bipolar terminal model

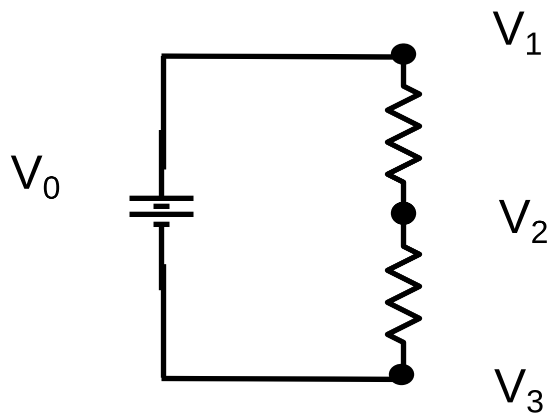


Credit: Make Electronics

- Many transistors have a rounded back and a flat front. Looking from the front the terminals are, in order, emitter, base, collector.
- There are other types of transistors with slightly different symbols: FET's, MOSFETS, …

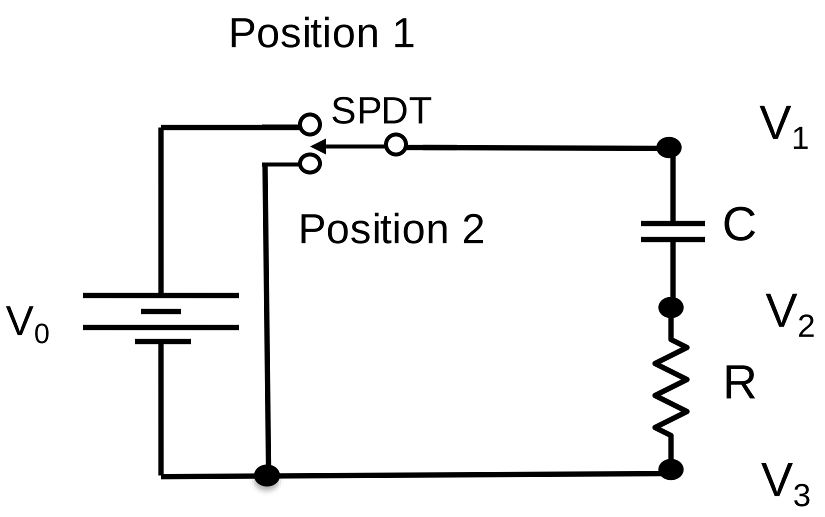
Analysis: Voltage divider

- Very commonly used e.g.: provide transistor bias
- $V_0 = V_1 - V_3$
- $\Delta V_1 = V_1 - V_2$
- $\Delta V_2 = V_2 - V_3$
- $\Delta V_1 = iR_1$, $\Delta V_2 = iR_2$, so $\Delta V_1 = \frac{R_2}{R_1}\Delta V_2$
- $(\frac{R_2}{R_1}+1)\ \Delta V_2 = V_0$ or $\Delta V_2 = \frac{R_1}{R_1+R_2}V_0$ and $\Delta V_1 = \frac{R_2}{R_1+R_2}V_0$



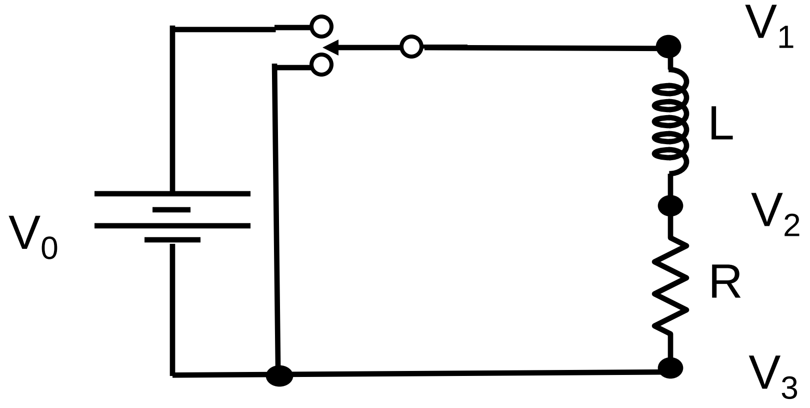
Analysis: RC circuit

- Switch set in position 1
- $V_R = V_3 - V_2$
- $V_C = V_2 - V_1$
- $V_0 = V_R + V_C$
- $I = C \frac{d}{dt} V_C = \frac{V_R}{R}$
- $V_0 = RC \frac{d}{dt} V_C + V_C$
- Solution to $RC \frac{d}{dt} V_C + V_C = 0$ is $V_0 \exp(\frac{-t}{RC})$ so
- Solution to $RC \frac{d}{dt} V_C + V_C = V_0$ is $V_0 (1 - \exp(\frac{-t}{RC}))$



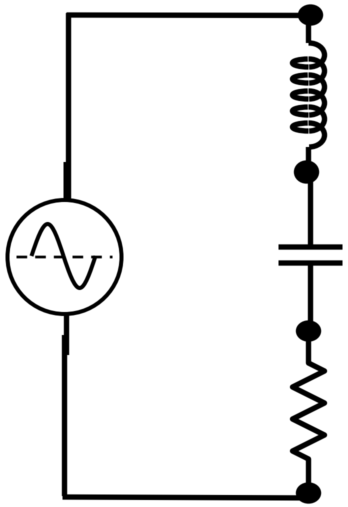
Analysis: RL circuit

- Switch set in position 1
- $V_R = V_3 - V_2$
- $V_L = V_2 - V_1$
- $V_0 = V_R + V_L$
- $IR + L\frac{dI}{dt} = V_0$
- Solution is $I(t) = \frac{V_0}{R}\exp\left(-\frac{tR}{C}\right)$.



Analysis: Resonance

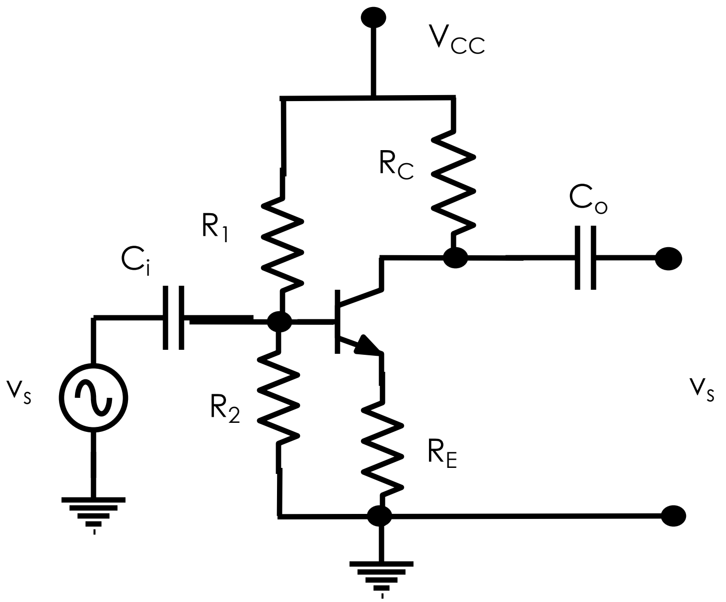
- $V_{in} = V_0 \sin(\omega t)$
- V_C, V_L, V_R as usual
- $I(t)= \frac{V_o}{Z} \sin(\omega t-\phi)$
- Z is the impedance of the equivalent circuit
- $Z = \sqrt{R^2 + [\, X_L - X_C \,]^2}$ and $\tan(\phi) = \frac{(X_L - X_C)}{R}$
- ϕ is the phase angle
- Z is minimum (so current is maximum) when $X_L = X_C$
- Or, equivalently when $\omega L = \frac{1}{\omega C}$, or when $\omega^2 = LC$.



Subsystem components and integrated circuits

- Miniaturization has allowed us to cram a lot of components into a small package called an integrated circuit or IC.
- We’ll use the NEC 555 timer later, this is an analog IC with a small number of integrated devices (transistors, resistors, capacitors). Small scale integration components include operational amplifiers (e.g.-741, 358), audio amps (386), mixers (NEC-602), phase lock loops, and many others.
- An Intel processor has billions of transistors crammed onto a single die with feature sizes around 14 nm (1 nm = 10^{-9} meters).
- We’ll be using ICs extensively, especially in digital electronics. The Arduinos and Raspberry Pi processors have millions of transistors on a tiny die and cost peanuts.

BJT amplifiers



Common emitter amp

Credit: Ward, Hands on Radio.

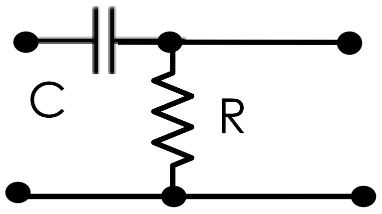
Here's how to design a common emitter amplifier. We use a 2n3904 transistor with β=150. This circuit will work! Build it.

1. Pick the supply voltage V_{cc}=12V.
2. Choose a gain (amplification factor), A = 5.
3. Choose the "Q point" of the conducting transistor (4mA).
4. $V_{cc} = (i_c \cdot R_C) + V_{ce} + i_e R_E \sim i_e \cdot (R_C+R_E) + V_{ce}$ with i_c=4mA. We get $(R_C+R_E) = (V_{cc} - V_{ce})/(4mA)$ =1.75 kΩ.
5. Since A = 5 and A=R_C/R_E, R_C= 5 R_E so $R_E \sim 270\ \Omega$ (this is a standard resistor value) and R_C= 1.5kΩ.
6. $i_b = 4mA/\beta = 27\ \mu A$.
7. Since V_{be} must be greater than .7V throughout the input signal range, we want the voltage across R_2 to satisfy $V_{be} + i_c R_E = 1.8V$.
8. We insert a voltage divider consisting of R_1 and $R_{2,}$ so that R_1= (12-1.8)/270 μA ~ 39 kΩ.
9. C_o and C_i are picked to offer small resistance to the frequency range we're interested in and $C_o = C_i = 5\ \mu F$.

I haven't explained why we want R_E but it provides thermal stability for the transistor over the range we care about. The fact that A=R_C/R_E can be calculated using Kirchhoff's laws.

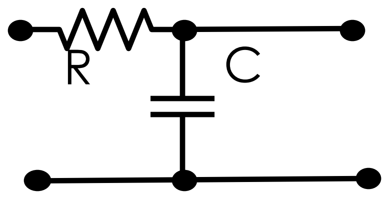
Filters

- Filters attenuate signals over unwanted frequency ranges and preferentially pass signals in wanted frequency ranges.
- High pass filters pass frequencies over some threshold.
- Low pass filters pass frequencies under some threshold.
- Band pass filters pass frequencies within a lower and higher frequency range.
- Here are some sample filters. You can calculate, H, the transfer function using Kirchhoff's laws.



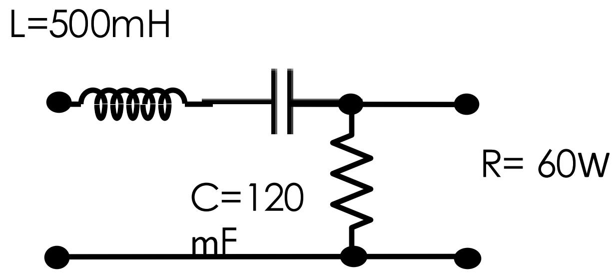
RC-high pass filter

$H = (1 - j/(\omega RC))^{-1} = (j\omega t)/(1 + j\omega t)$, $t = RC$



RC low pass filter

$H = R(1 + j/(\omega RC))^{-1}$

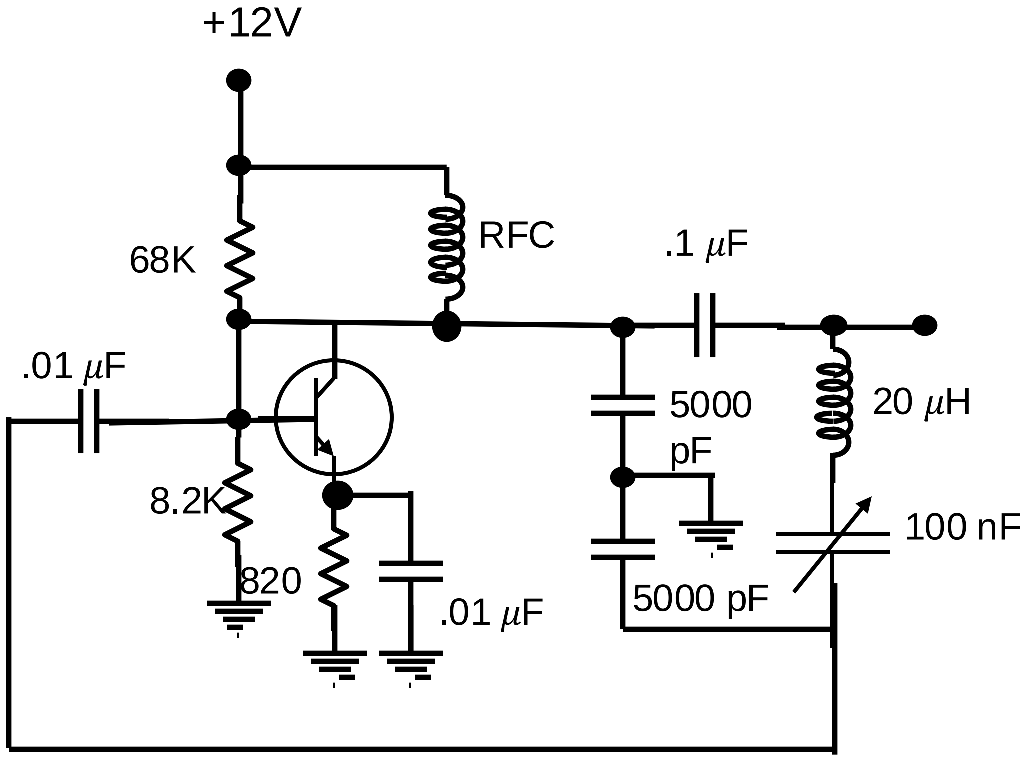


RC Band-pass filter

$H = R(R + j(\omega L - 1/(\omega C)))^{-1}$

Oscillators

- Oscillators are a combination of amplifiers and filters.
- The output of an amplifier is fed back to the input through a filter or resonance circuit so that the desired frequency output is favored.
- The output rapidly settles on a clean signal with the right frequency
- The Clapp Oscillator on the right uses a balanced resonant circuit to pick the desired frequency.
- Crystals can also be used to obtain very precise frequency selection that does not change as circuit elements heat up.



Power, SNR and bandwidth

- Relative power, like the power of an input signal compared to an output signal is often measured in dB.
- dB= 10 log($\frac{P_1}{P_2}$). 3dB loss is equivalent to loss of half the original power.
- Filters performance is often measured in dB. A good filter will often have relative power (in dB) between the “passband” and the “rejection band” of a factor of more than 10.
- At any temperatures higher than 0K, there is ambient noise and a critical metric is the relative power of the signal you want and the noise. This parameter (measured in dB) is called the Signal to Noise ration (SNR).
- Bandwidth is a confusing term.
  - Most computer scientists use the term refer to the speed with which data can be transmitted or received.
  - Analog engineers usually use the term to refer to “passband width.”
  - These two are related: a wide passband can pass modulated information encoded at high data rates. Voice transmission require a passband of only about 20 KHz. High speed optical links have passbands of GHZ.

More circuits

Differential amp

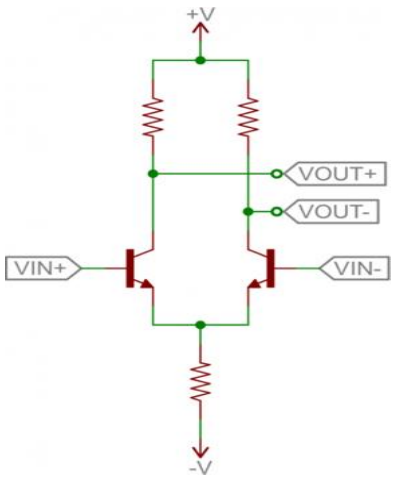
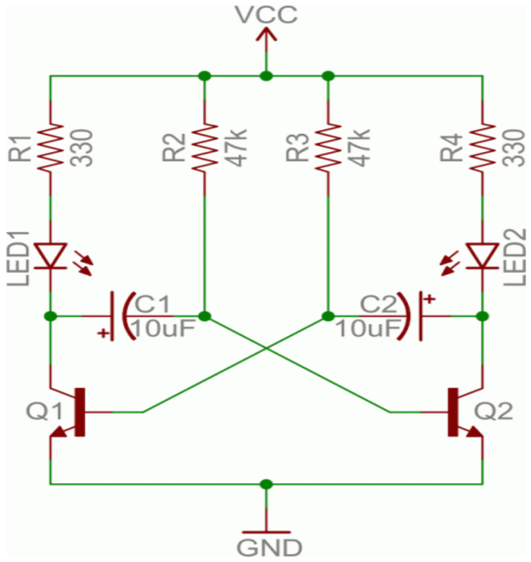


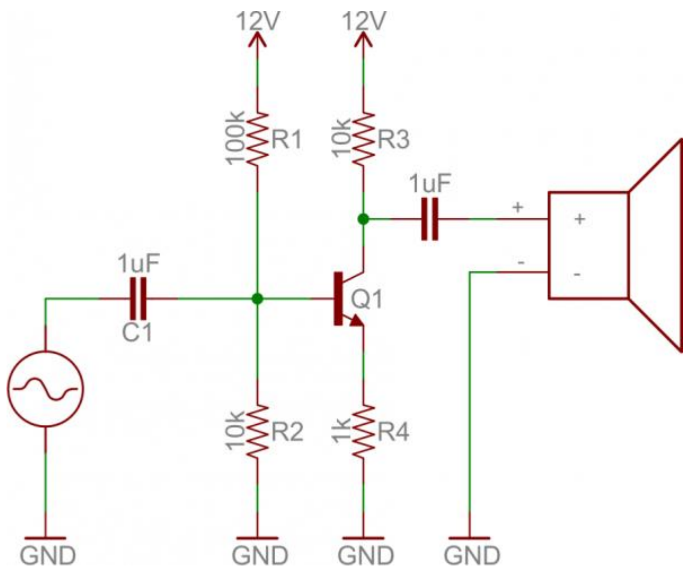
Diagram credit: Sparkfun

Multi-vibrator



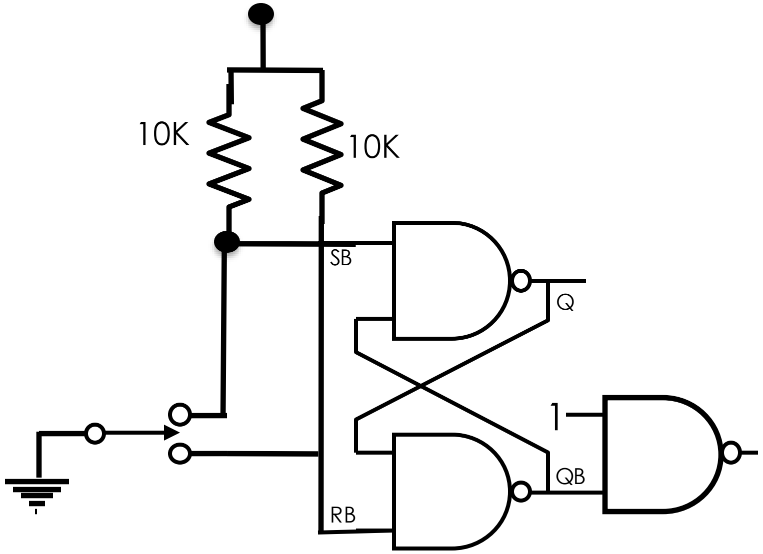
Build this and try 100 μF capacitors.
How often do the LED's blink?

Common emitter as audio amp

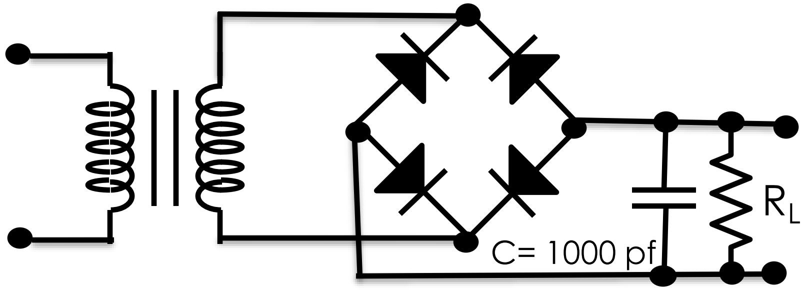


More Circuits

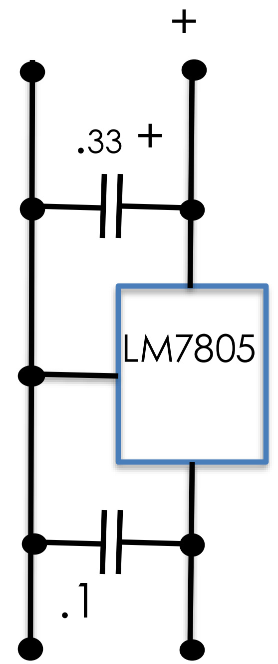
Debounce



Power supply



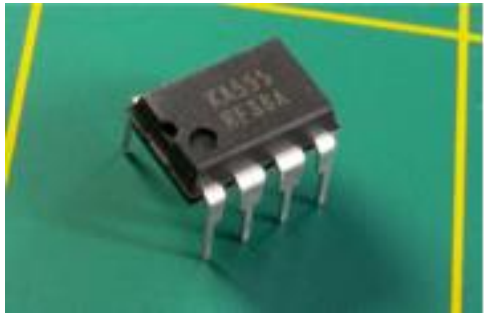
Voltage regulator



Electronics part 2, digital circuits

- You should understand what integrated circuits (IC’s) are and be able to describe, model and use:
  - Timers (like the 555 timer)
  - Op Amps (like the 741)
  - Digital circuits are clocked circuits that implement operation on binary values in clock cycles.
  - You should understand how digital memories are implemented and be able to describe two: SRAM and DRAM as well as their characteristics.

IC's: 555 Timer

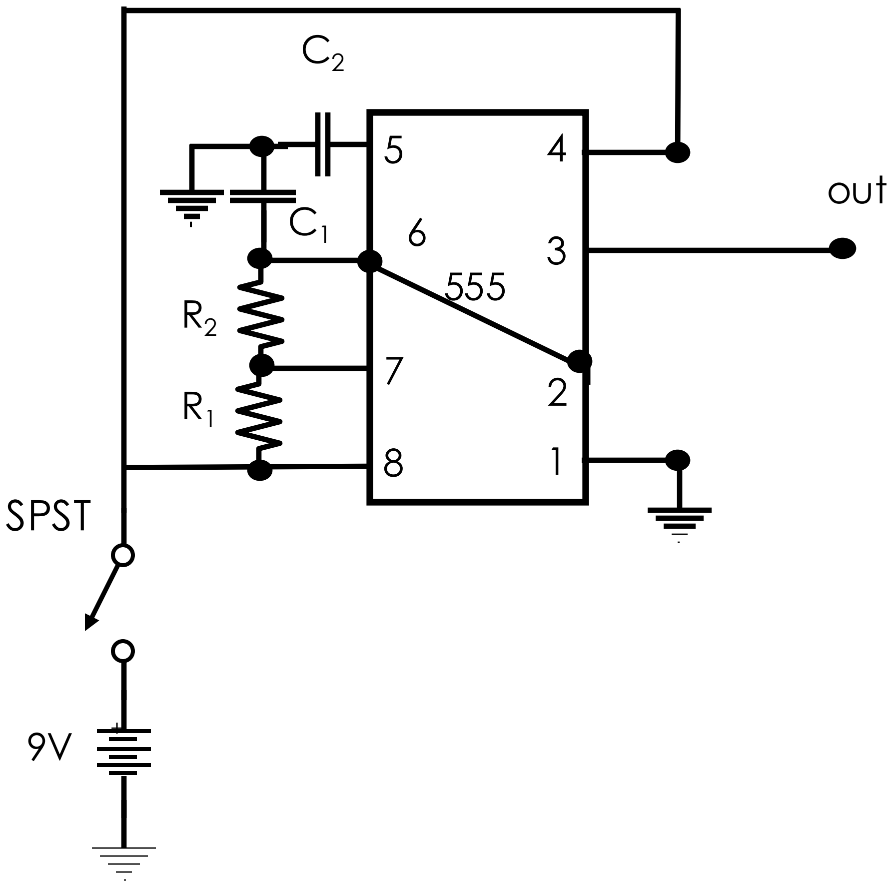


Credit: Make Electronics

- Package: 8 pin dual inline package ("DIP")
  - Pin 1: -Vs
  - Pin 8: +Vs
  - Pin 2 (Trigger): Out HIGH if $V < V_{CC}/3$. Pin 2 has control over pin 6. If pin 2 is LOW, and pin 6 LOW, output goes and stays HIGH. If pin 6 HIGH, and pin 2 goes LOW, output goes LOW while pin 2 LOW. This pin has a very high impedance (about 10M) and will trigger with about 1uA.
  - Pin 3 (Output): (Pins 3 and 7 are "in phase.") Goes HIGH (about 2v less than rail) and LOW (about 0.5v less than 0v) and will deliver up to 200mA.
  - Pin 4 (Reset): Internally connected HIGH via 100k. Must be taken below 0.8v to reset the chip.
  - Pin 5 (Control): A voltage applied to this pin will vary the timing of the RC network (quite considerably).
  - Pin 6 (Threshold): HIGH if $> 2\, V_{CC}/3$, make output LOW only if pin 2 is HIGH. Pin 6 has very high impedance (~10M) and will trigger with about 0.2uA.
  - Pin 7 (Discharge): Pin 7 is equal to pin 3 but pin 7 does not go high - it goes OPEN. When LOW it sinks about 200mA.

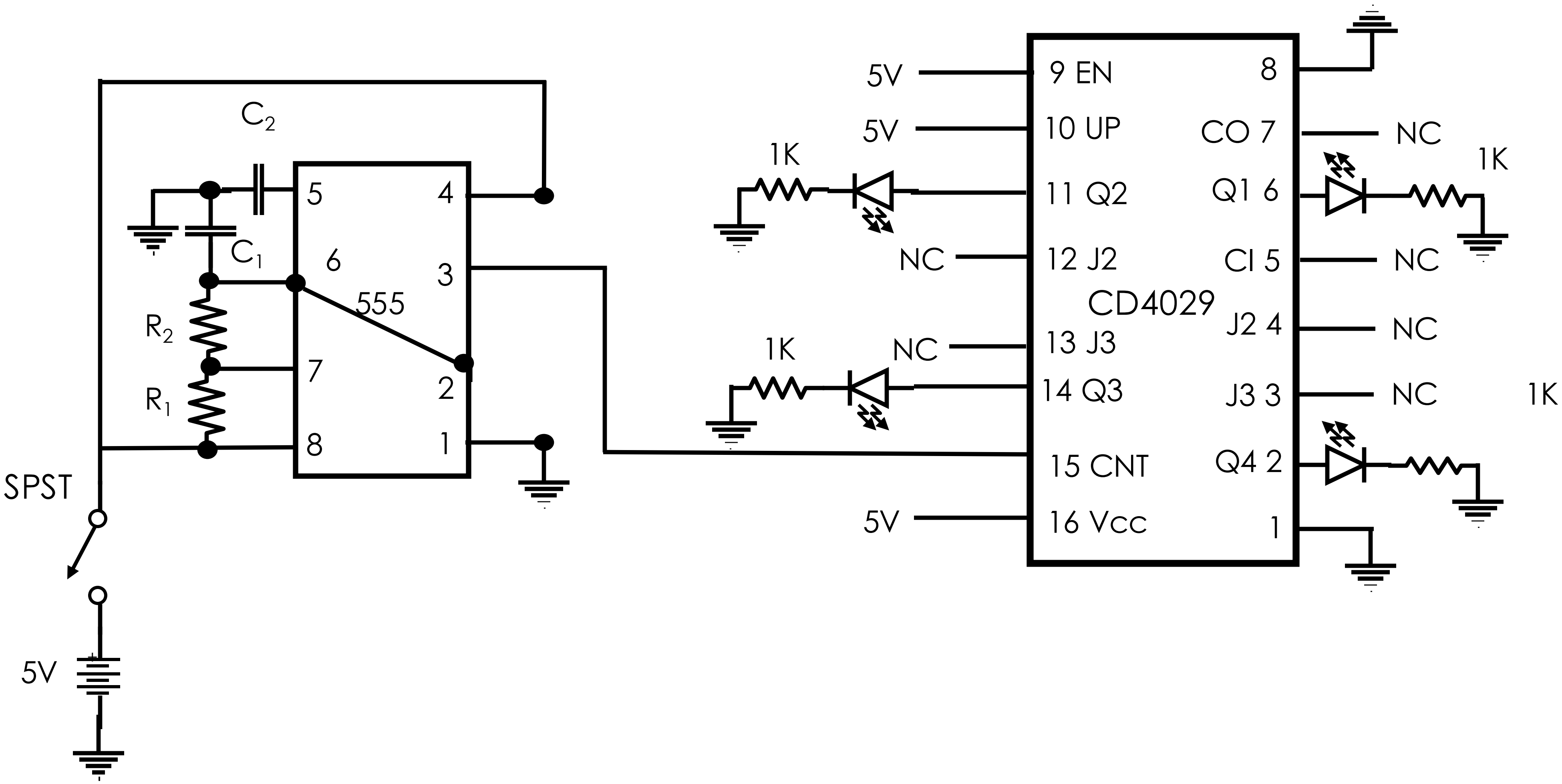
Lab: Using the 555 to generate a clock

- We use the 555 in *astable* mode
- $t_0 = .7C(R_1 + 2R_2)$
- $duty - \text{cycle} = \frac{R_1+R_2}{R_1+2R_2}$
- Try
  - $R_1 = R_2 = 10K$
  - $C_1 = 10\mu F$
  - $C_2 = .1\mu F$
  - Freq: ~50/sec
- Try
  - $C_1 = 1\mu F$, what happens to the frequency



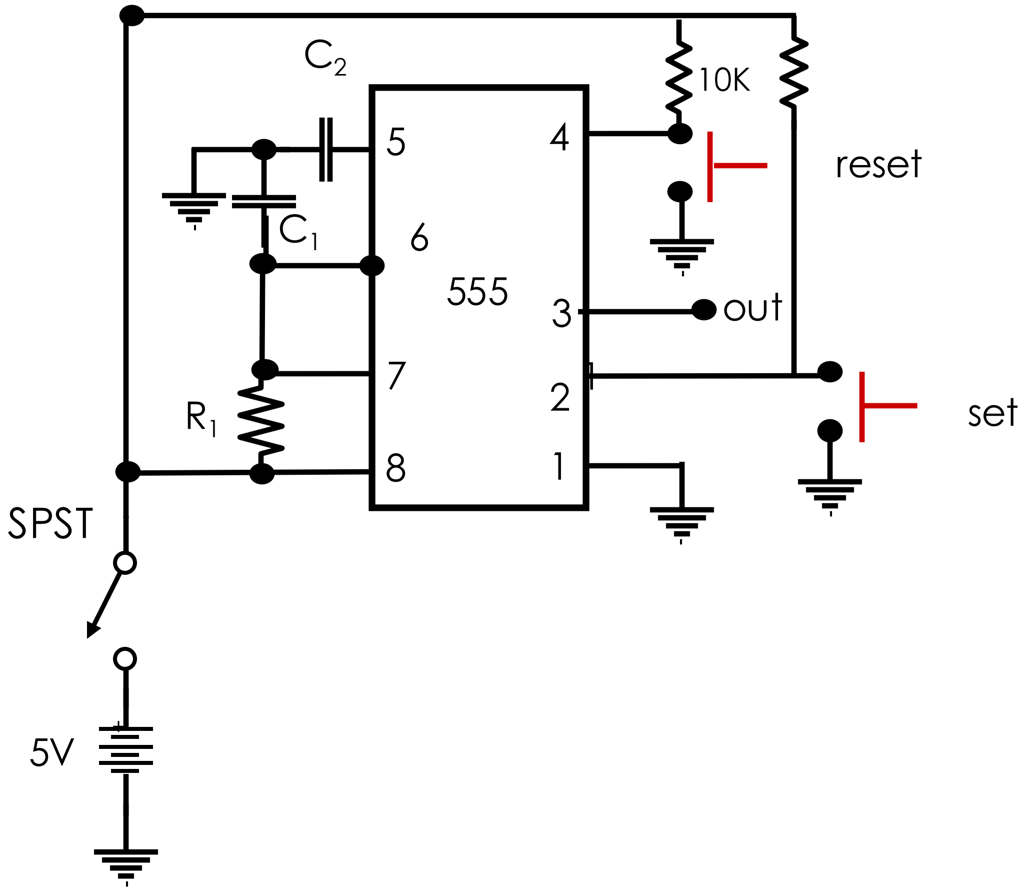
Lab: Using the 555 to clock a counter

- 555 in *astable* mode used as counter
- Same passive values as previously
- $t_0 = .7C_1R$



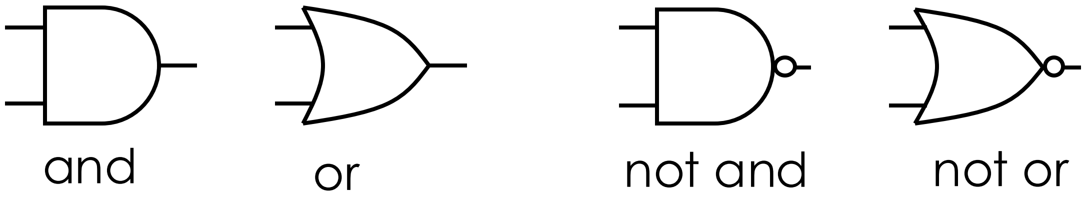
Lab: Using the 555 to generate a pulse

- Try
  - $R_1 = R_2 = 10K$
  - $C_1 = 100\mu F$
  - $C_2 = .1\mu F$
- If you build this and connect an oscilloscope, you'll notice the wave isn't perfectly square. As the pulse rises, you get some overshoot and on discharge there is undershoot. What happens if connect a RC circuit at the output with C around .1 μF. You can also use a Schmidt trigger to clean up messy waveforms.
- Pulse length time is $R_1 C_1$ secs

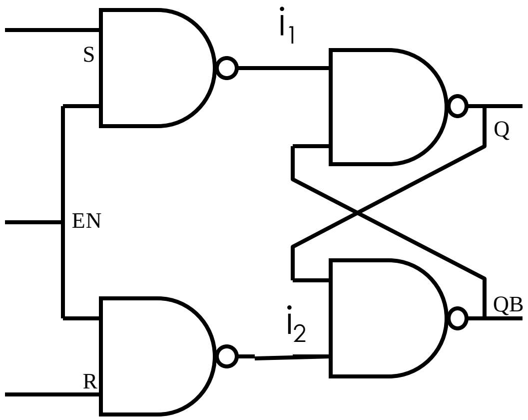


Logic symbols, flip flops, latches and shift registers

- We can build circuits that implement logic operations. For example, the and below will output a 1 (5V) only if both inputs are 1.

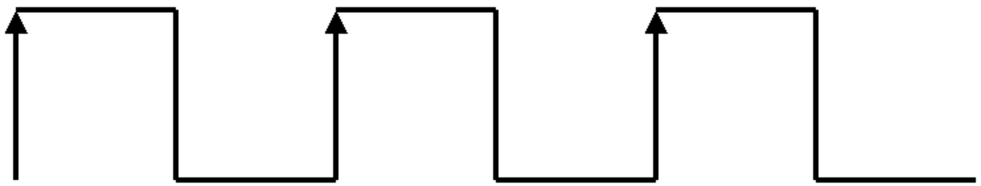


- We can combine these elements together with a clock to build important digital building blocks.
- The circuit on the right is an S-R flip flop. QB = ¬Q, always. If EN=0, i_1=i_2=1 and the other NAND input will be Q will get ¬Q and Q will keep whatever value it had. If EN=1, S=1 and R=0, Q=1 and QB=0. If EN=1, S=0 and R=1, Q=0 and QB=1. The output is undefined if EN=1, and S and R have the same value.
- We can also build memory elements, arithmetic elements, and comparators from these elements.



Digital signals and synchronous circuits

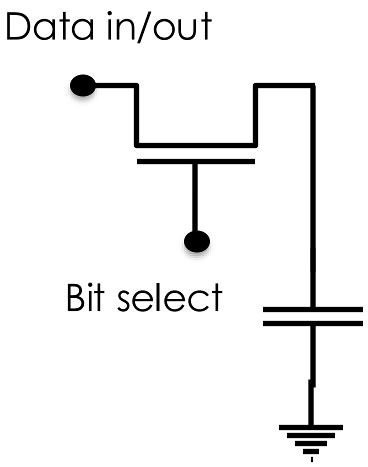
- Digital signals have two level, on (say 5V) and off (say 0V).
- Digital circuits are clocked meaning there is a clock signal pictured below.



- A value is calculated by combinatorial logic (and’s or’s,…) with and memory elements and moved into storage elements (registers) on each clock cycle.
- In a microcontroller or computer, a program which consists of binary instructions and on each cycle, the instruction selects the logic connecting input registers, the particular logic required by an instruction and an output register.
- A program moves from one instruction to the next each clock cycle although some instructions may cause a change in program flow by “jumping” to an earlier or later instruction based, say on some test condition (e.g. jump if the value of register 1 > the value of register 2).
- Digital circuits are pretty simple but when so many of them are composed, the processing power is formidable; it has changed our lives.

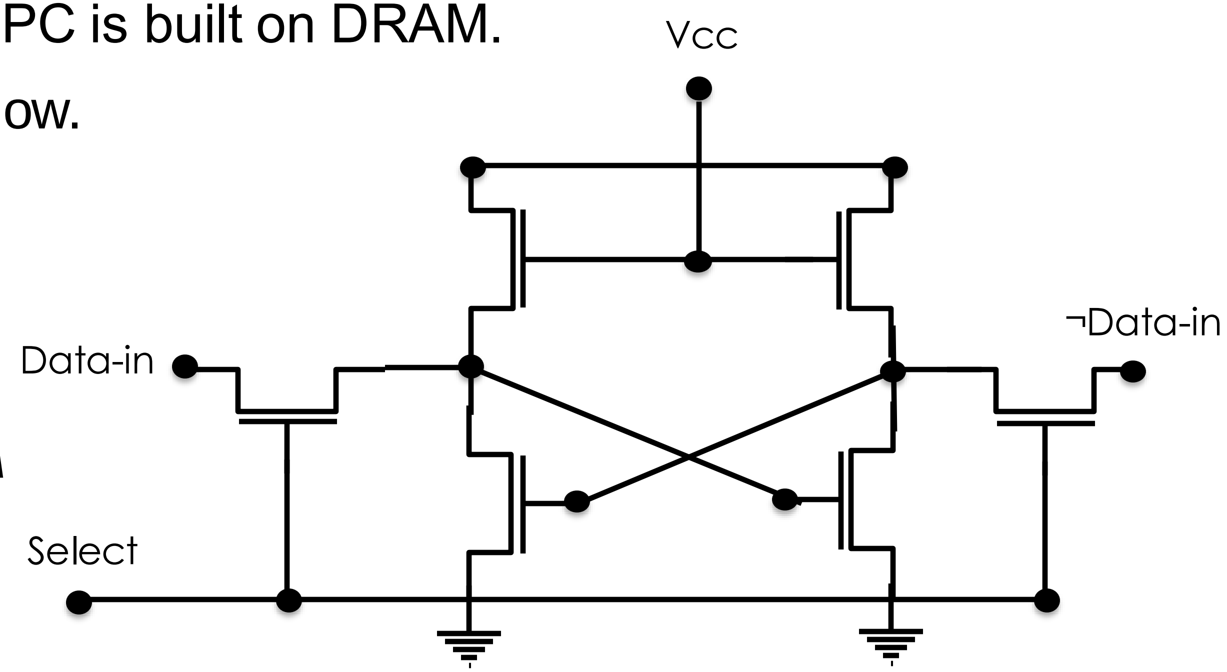
SRAM and DRAM

- Binary values can be stored in and retrieved from computer memory.
- Computer memory comes in two types: Static random-access memories (SRAM) retain the storage value with no intervention and have very fast access times. The registers and caches in your computer are based on SRAM.
- Dynamic random-access elements retain values for brief periods of time and must be “refreshed” regularly. The main memory in your PC is built on DRAM.
- You can see how DRAM and SRAM are built below.



DRAM

SRAM



Circuit tools

- In addition to the sort of analysis we've done, there are many computer aided design tools. Here are two:
  - SPICE: This is a famous device/ circuit analysis tools.
  - EagleCAD PCB files on GitHub: This is a cad system to lay out printed circuit boards.

Electronics part 3: Maxwell and Electromagnetic radiation

- See my “Electronics of Radio” Notes for this.

Exercises

- 10K pull-up resistor and 5 v power. What is the power drain when we connect it to ground?
- Build some of the circuits in this lecture, analyze them with Kirchhoff's laws and measure (with an oscilloscope) input and outputs.
- Estimate the power at reception of a GPS signal.
- Use a multimeter
- Use an oscilloscope

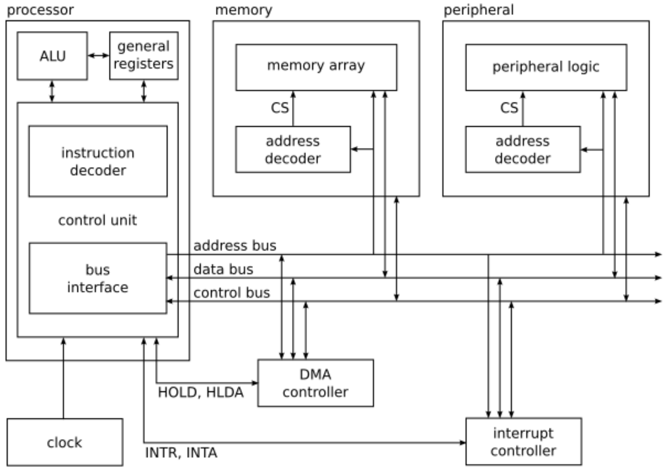
Electronics part 4: Computers and computer architecture

- Programmable electronics
  - Digital electronics can be used to build computers which execute binary instructions in digital memory on data in digital memory to compute results or control equipment.
  - Computers are connected to peripherals (sensors, keyboards, displays, speakers) to collect or transmit data to people or other devices.
- You should understand basic computer architecture and how the elements below achieve programmability.
  - CPU’s
  - Memory
  - Busses
  - I/O – Serial communication
- You should understand some important elements of IoT architecture like
  - SoC’s
  - IoT busses: SPI, I^2C

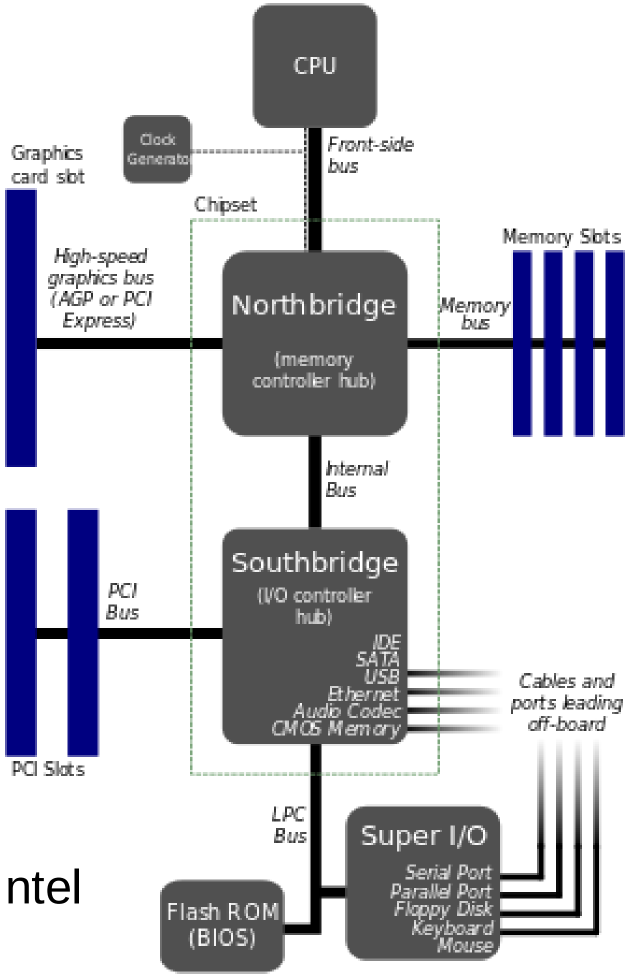
Computer processor architecture

- CPU’s
- Memory (registers, SRAM, DRAM)
- MMU’s
- Busses
- Rings and privilege separation
- Virtual memory
- Devices, interrupts, memory mapped I/O
- Programs
- Inter-computer communication
- GPU’s and FPGA’s

Computer processor architecture



- Computers are often described in terms of their architecture.
- Architectures focus on the high level interaction of components rather than low level schematics.
- Architecture represents the "systems programmer's" view of the computer.
- It was popularized by the deservedly admired "Principles of Operation" describing the IBM /360.
- Computers execute digitally encoded instructions which can be loaded at will into a computer. They are "von Neuman" machines.



- The diagram above is a generic computer architecture. The one on the right is an Intel based computer.
- The major blocks are the CPU, clock, memory subsystem (MMU, cache, DRAM), inter-block bus interfaces and I/O subsystem (busses, interrupt processing).

Computer processor architecture

- A computer is a “clocked” electronic system. Each computer operation is simple and occurs in a clock “cycle.” The clock cycle of modern computers vary from “millions of cycles per second” to “billions of cycles per second.” The faster, the better.
- CPU, clock
  - The CPU fetches instructions from memory and executes them.
    - In a simple processor, a single instruction is executed each cycle; in a complex processor, a single instruction might take several cycles and many instructions may be executing, in different execution units, simultaneously.
  - A typical instruction might be “add memory location a to memory location b.”
  - Normally, instructions are executed sequentially but there are three ways to change control flow.
    - First, unconditional jumps or conditional jumps can synchronously change control flow. For example, an unconditional jump might be “jump to the instruction at memory location m” while a conditional jump might be “jump to the instruction at memory location m if the value at memory location a is bigger or equal to the value at memory location b.” This allows program “loops.”
    - Second, interrupts (from clocks or devices) can instruct the processor to start executing instruction streams at different locations.
    - Third, error conditions that occur in user mode processing can cause “exceptions” that trap into the OS. An example may be dividing by 0 or attempting to execute a privileged instruction.

Computer processor architecture

- Instructions and data are stored in binary form, i.e. as a series of 1's and 0's.
- Binary values are organized into bytes (8 bit values) and "words" varying from 2 bytes to 8 bytes.
- Words represent numerical values (12, 2324241) and addresses.
- Addresses are unsigned integers and are represented in the usual binary form.
    - 1 is 0000000000000001
    - 18 is 0000000000010010
- Signed integers are usually represented in "two's complement notation:"
    - Positive integers have high order bit of 0, the remainder of the word is the ordinary unsigned binary form.
    - 0 is all 0 bits.
    - Negative numbers have a high order bit of 1 and are arranged so that using the "standard" unsigned addition circuits x + (-x) = 0.
    - A simple algorithm for turning a positive binary number into its corresponding negative is to flip each bit in the original representation and use unsigned arithmetic to "add 1."
    - Try it.

Computer processor architecture

- Memory subsystem
  - Memory systems store binary values organized as 8 bit bytes. Bytes are organized into words and instructions can refer to either “byte sized” data values or “word sized” data values.
  - Most systems are addressable on byte boundaries and at the very lowest level, memory is accessed by “physical addresses” starting at 0 and ending at the size of the memory - 1.
  - Programs can be loaded into memory at different locations at different times. As a convenience, most systems have MMU’s which allow a supervisor program to map memory from physical addresses to virtual addresses. Virtual addresses give the program the illusion that memory for their program always starts at 0. The MMU translates virtual addresses into physical addresses.
  - Actually, memory is usually arranged into a “hierarchy.”
  - CPU’s have internal memory called registers which can be accessed in a single cycle.
  - “Main memory” consists of DRAM which (depending on the clock rate of the CPU) can take hundreds of cycles to access.
  - In between the CPU and main memory is “cache memory” built from SRAM which contains copies of main memory locations. Cache memory can be accessed in a few cycles maybe even one.

Computer processor architecture

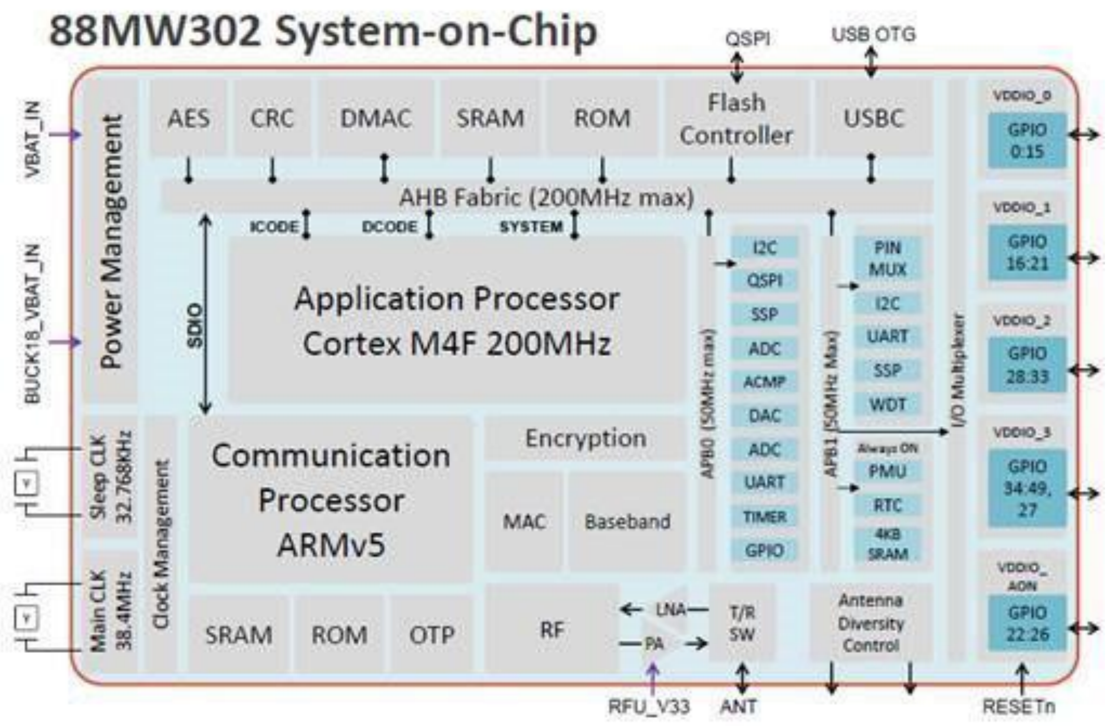
- Memory subsystem
  - When a main memory location is accessed and it is not in cache, the value at the location and several sequential locations are loaded into cache. Subsequent accesses are fast.
  - Cache and main memory contents must be coherent, that is at well understood times the values must be consistent.
  - After a while, under OS control, cache values are evicted and reside only in main memory.
- Inter-block bus interfaces
  - These are fixed width control lines connecting the CPU and its subsystems. Busses have protocols which dictate the manner in which a device “addresses” another device, puts binary values onto the bus for reading or writing and arbitrates access.
- I/O subsystem (busses, interrupt processing).
  - I/O busses are similar to inter-block buses but are usually slower. When devices use the bus and want the attention of the CPU, the bus protocol causes a processor interrupt.
  - Some devices can read and write main memory without interrupting the CPU. This is called DMA and it is mediated by the MMU.

Computer processor architecture

- There are a few hardware features that greatly simplify programming when many programs are being run. They also provide protection and isolation between programs. Usually there is a single “supervisor” program which manages a computer: this is the operating system “kernel.” Here are some hardware enabled “sharing” features:
  - Rings of privilege: Most processors have two or more processor modes (ring 0, 1, 2, 3) or supervisor mode and user mode.
  - The operating system kernel runs in supervisor mode. In supervisor mode, a program can execute any instruction, access any memory location, program any peripheral or dependent system (like the MMU). Kernels set up user address spaces for user programs (operating in user mode) by setting up virtual memory maps (page tables) which cannot be accessed by user programs. Supervisor mode instructions and memory are not visible to user mode programs. Usually, the instructions and memory of one user mode program is not accessible to another user mode program (two different programs can share some memory under kernel control). This prevents one program from interfering with another program. Interrupts are set up to “trap” into supervisor mode and are serviced by the kernel. This allows different user programs to safely share devices like displays, USB devices and disks.
  - When a user mode program needs a kernel “service,” like writing a file or starting another program, it formats the request and makes a “supervisor call” which traps into the kernel. Base on the arguments and the user mode program privileges, the supervisor performs the service and returns to the user program in user mode.

SoC's

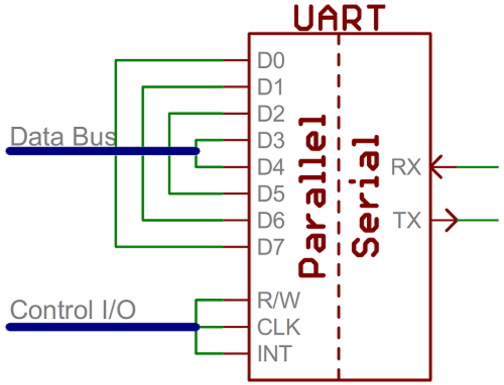
- With miniaturization, most or all the computer subsystems can be placed on a single integrated circuit or "die" called a "system on a chip" or "SoC."
- The die pictured on the right is an example with an ARM CPU, MMU, cache, read only memory (ROM), several specialized processors like an AES accelerator which can encrypt data at high speed.
- DRAM is usually on a separate chip as are I/O devices.
- This processor also has "GPIO" pins which can be connected to external circuits and can be controlled by instructions from the CPU.
- Most IoT devices consist of one or more "SoC's."
- Integration of many devices (billions of transistors) is at the heart of the digital revolution. It's amazing.



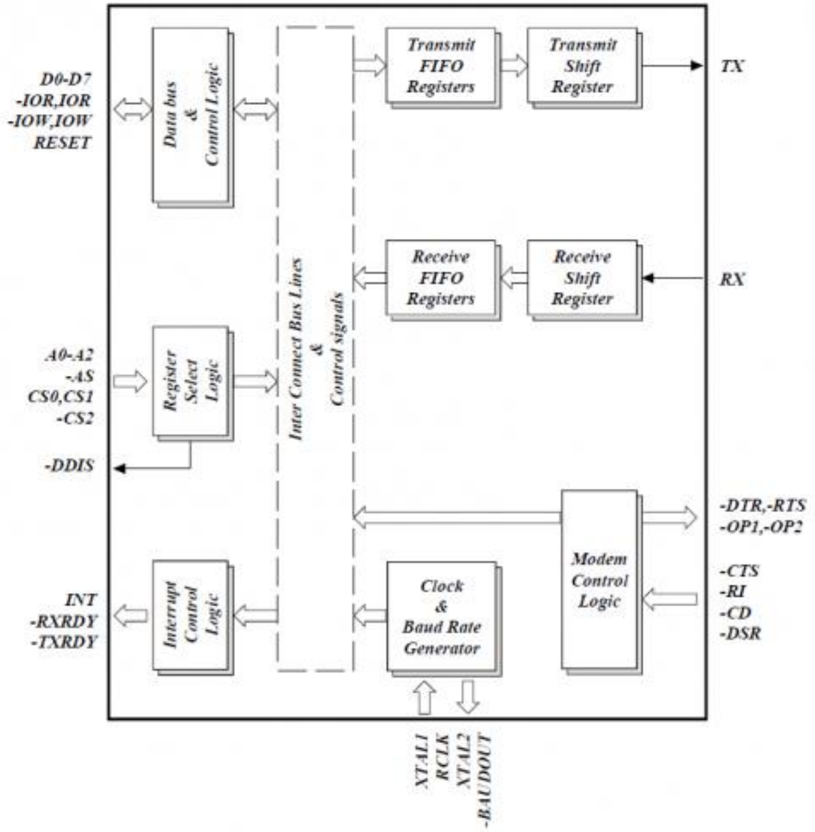
Electronics part 5, Hands on IoT

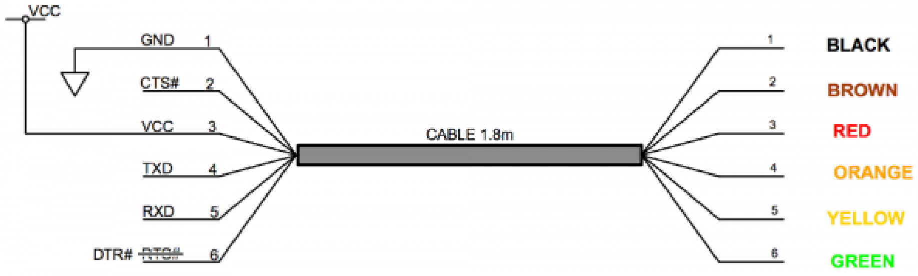
- In this section you will learn to use a simple (and cheap) microcontroller. We will learn:
    - How to program them.
    - How to interface a wide variety of sensors, devices and transducers using GPIO pins.
    - How to communicate with it via the I/O busses (and monitor them).
- We’ll do the same for the “Raspberry Pi,” a cheap, but full featured computer which shares the Arduino’s ability to use GPIO but runs a full OS and networking stack.
- Arduino and Raspberry Pi represent typical IoT processors.
- You will learn the nuts and bolt of connecting Arduino’s and Raspberry Pi with components using perf-boards and how to obtain all these components and design experiments with them.

Serial Communication



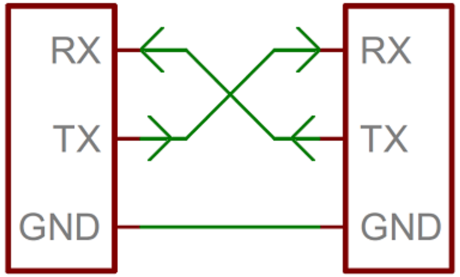
- We need a way to communicate between digital devices and perhaps the easiest method is via an *asynchronous* serial port.
- Universal Asynchronous Receiver/Transmitters (UARTs) are integrated electronics packages (diagram lower right) that implement the serial protocol (next page).
- The signals can be carried on a simple pair of wires.
- You can interface a UART to a computer using a “badge”. I use an Attify Badge which can interface with UART, JTAG and SPI protocols. Another option is an FT232RL FTDI USB to TTL Serial Adapter Module for Arduino Mini Port. Well demonstrate this later.
- LSB is transmitted first usually.

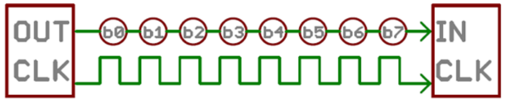


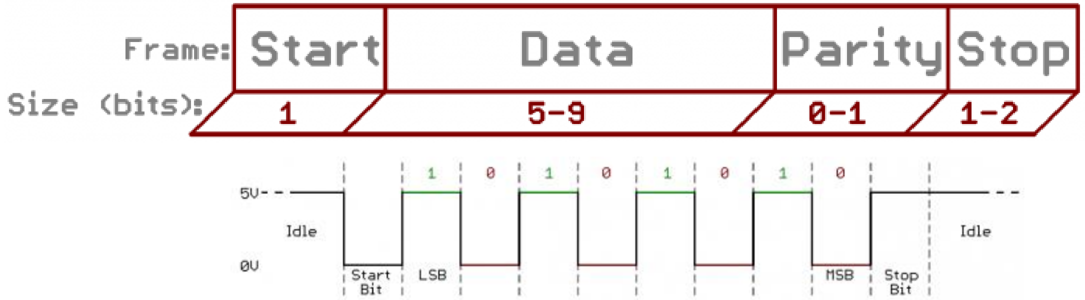


Serial Communication

- The UART serial protocol allows two communicating endpoints wired together to send and receive sequentially transmitted digital data bits at speeds (baud rate) known in advance. Each end has a receive pin (RX) and a transmit pin (TX). Data is transmitted in frames consisting of Data bits, Synchronization bits, Parity bits, and Baud rate:
  - Start bit (0), data, transmitted least significant bit (LSB) first, a parity bit and a stop bit (1).
  - The start bit is LOW telling the receiver that data is coming, then data bits are sent, after 9 bits TX goes HIGH marking the end. 8 bits of data requires 10 bits transmit/received.
  - The most common version of this protocol is 9600N1: 9600 baud, no parity, 1 stop bit.
  - Data bits, Synchronization bits, Parity bits, and Baud rate.
  - The start bit is always indicated by an idle data line going from 1 to 0, while the stop bit(s) will transition back to the idle state by holding the line at 1.

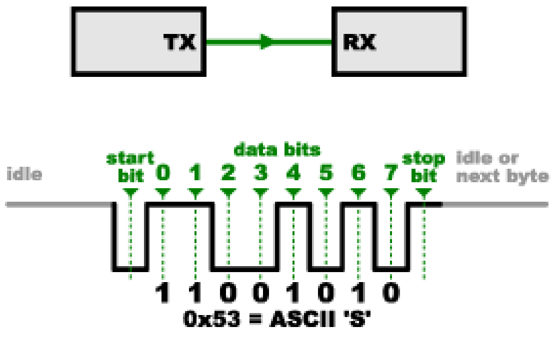


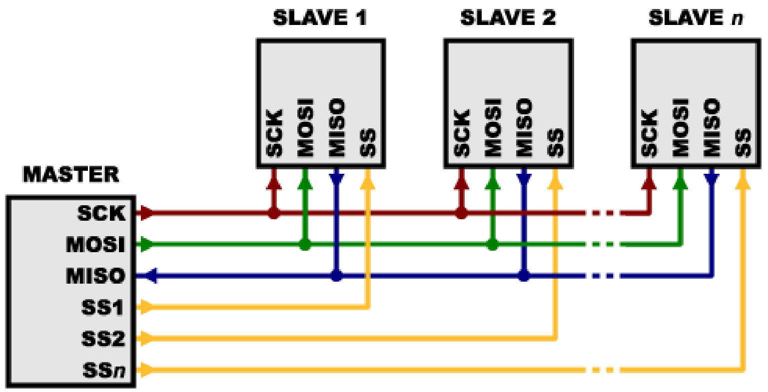


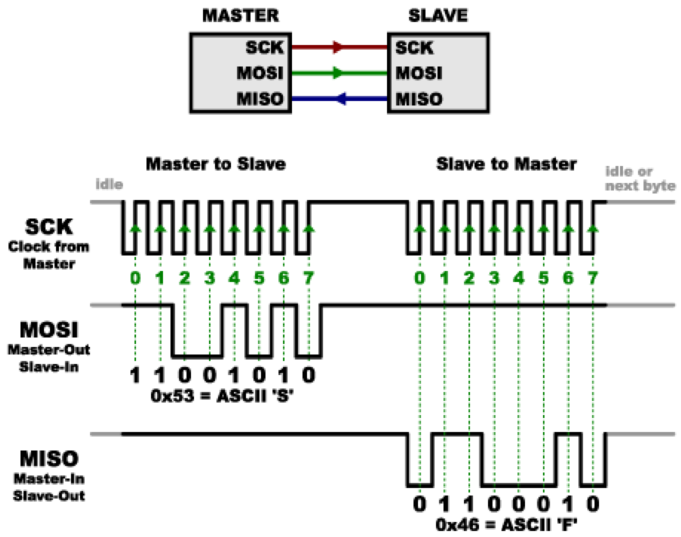


SPI

- A collection of signals shared by two or more endpoints is called a bus.
- The SPI bus has a single master device and several slave devices. It is full duplex (meaning the master both transmits and receives data).
- Transmission speed agreed in advance. Communications is full duplex.
- The SPI bus signals are:
  - Master clock (SCLK)
  - Master output to slave input (MOSI)
  - Master input to slave output (MISO)
  - Slave select (SS). This pin on slave is pulled low to select a device)
- Sample on rising or falling edge
- Arduino and Raspberry Pi have SPI libraries
- See https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

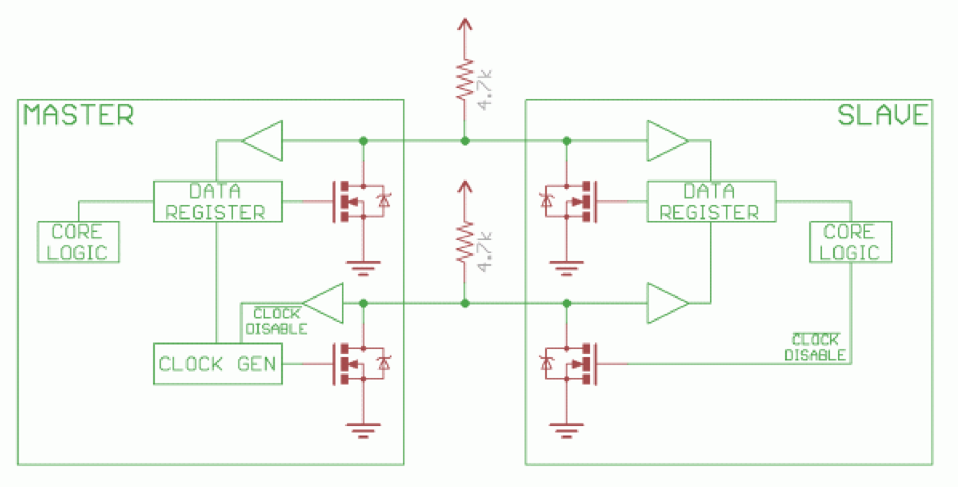






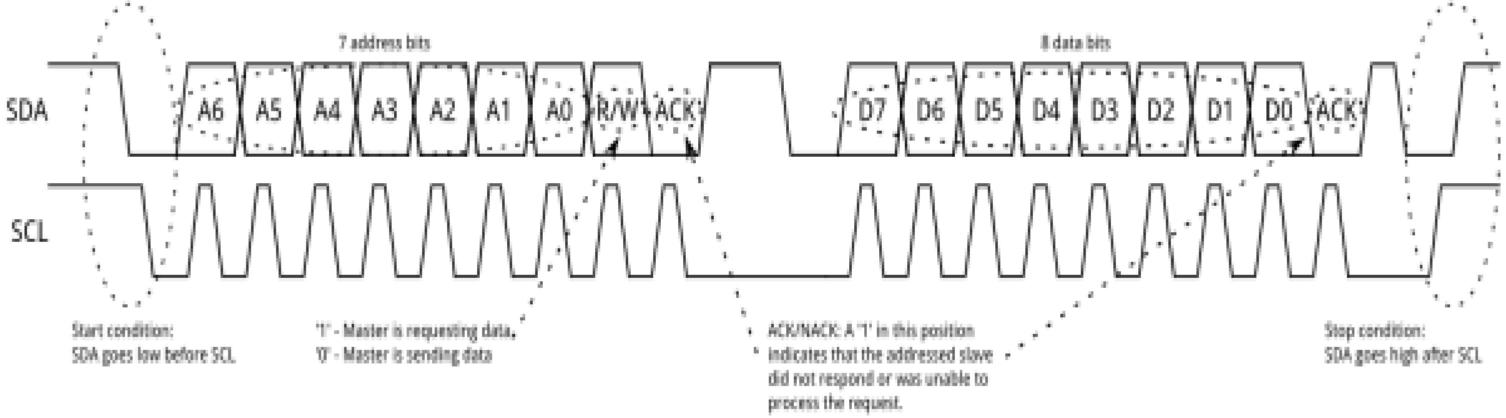
I^2C

- The I^2C bus, like SPI, uses clock information but also allows multiple addressable devices on the same wire bus. The I2C bus consists of the following signals:
  - SCL
  - SDA
  - GND
- I^2C devices can communicate at 100kHz or 400kHz.
- I^2C bus drivers are "open drain", meaning that they can pull the corresponding signal line low, but cannot drive it high. Thus, there can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the system. Each signal line has a pull-up resistor on it, to restore the signal to high when no device is asserting it low.



I^2C

- The protocol is:
  - In the normal state SDA, SCL are high
  - To begin, the bus master pulls SCL high and SDA low (transition is start condition).
  - The master then sends the address of device (MSB first) being addressed and the R(1)/W(0) bit
  - The slave pulls SDA low to acknowledge (ACK).
  - The master or slave sends data (MSB, first)
  - The master generated stop condition by pushing SDA high with SCL high (transition from low to high is stop condition)
- For a 7-bit address, the address is clocked out most significant bit (MSB) first, followed by a R/W bit indicating whether this is a read (1) or write (0) operation. The 9th bit of the frame is the NACK/ACK bit. This is the case for all frames (data or address). Once the first 8 bits of the frame are sent, the receiving device is given control over SDA.



IoT Sensors and Arduino

- Cameras and CCD’s
- IMUs
- Temp, humidity, infrared detectors
- GPS
- Radar, lidar
- RF receivers, transmitters
- Sound, vibration detection
- Barometer (altimeter)

Arduino

- The Arduino is a microcontroller development board. It costs about $15.
- When the Arduino “wakes up,” it checks the serial connection for a program to be downloaded. If one is present, the Arduino downloads it. If there is no new program or the download is complete, it runs the last loaded program.
- Arduino has about 30 “GPIO” pins you can connect to. Some are connected to analog to digital converters (ADC) and digital converters (DAC) so you can read and write analog signals. Some are digital pins you can connect to other digital devices.
- Your program can read to write data to the I/O pins.
- The heart of the Arduino is a simple microcontroller (e.g. - ATMega-328).
- You can supply external power (5V) or it can be powered through the serial interface (3.3V).
- There is a complete program development environment available here, as well as tons of sample code for use with sensors (see references).

Arduino Bootloader

- In IDE, select the board definition for your target Arduino from **Tools > Board**. Then select the programmer that you are using under **Tools > Programmer** (if you are using the Arduino as ISP you will also need to select the COM port that the Arduino as ISP is connected to). Finally, select **Burn Bootloader**. This will take the board you selected and look up the associated bootloader in the *board.txt* file. Then, it will find the bootloader in the bootloader folder and install it. This only works if the board is installed correctly in the IDE and you have the correct bootloader.

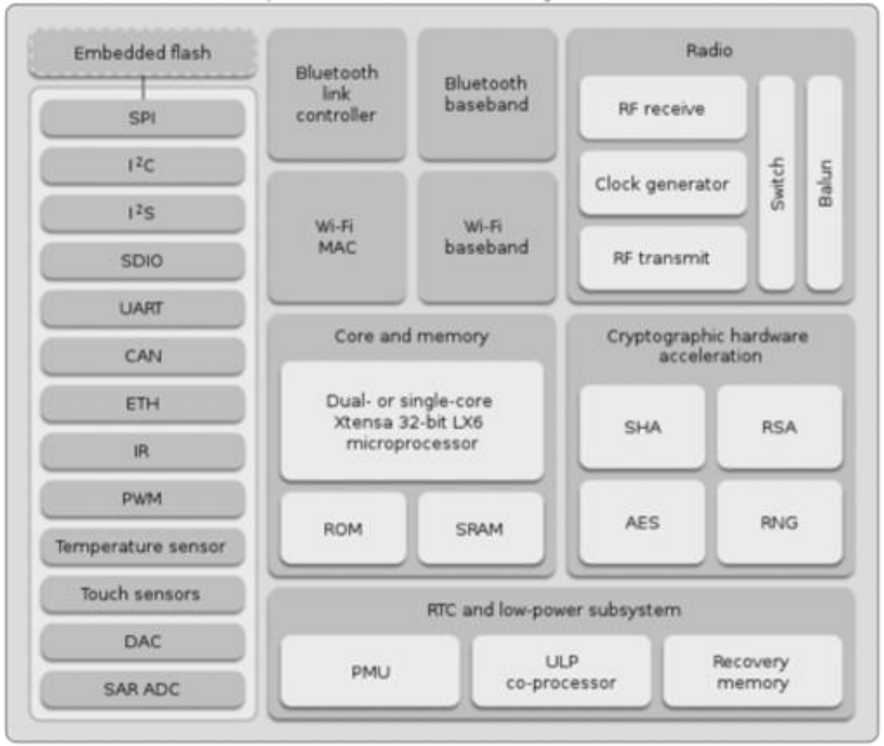
Arduino

- You can connect a lot of devices via the GPIO pins to the Arduino including programs that implement the asynchronous serial interface, SPI, and I2C, as well as digital sensors, memory, actuators like robot arms and radios.
- We’ll demonstrate
  - Reading and writing analog signal
  - Reading and writing a read only memory (ROM) chip
  - Reading and writing an SD card
  - Reading sensor data from thermometers, gyros, accelerometers and infrared detectors
- Connecting other devices is very similar.

ESP32

- Another microcontroller is the ESP32, released in 2016.
- This uses the Xtensa architecture
- It has built in wifi and blue-tooth (the Arduino doesn't).
- The documentation is here.
- You can use the Arduino IDE to program the ESP32 and the same sample code should work.

Espressif ESP32 Function Block Diagram



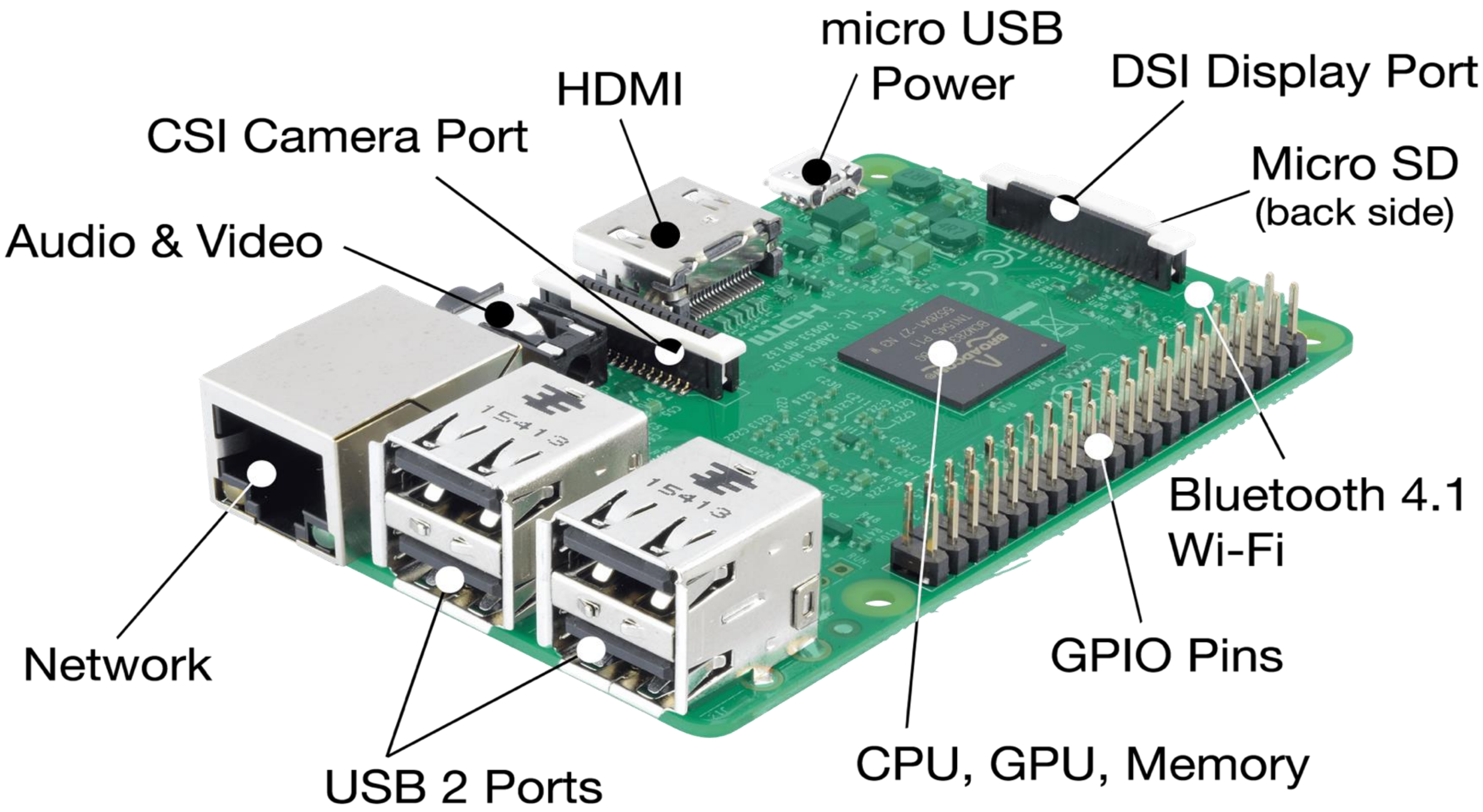
Information and Stuff

- Here are some datasheets for the hardware and some software we'll use:
  - ADXL345 Accelerometer Datasheet
  - L3G4200D Gyroscope Datasheet
  - MC5883L Magnetometer Datasheet
  - DHT Datasheet
  - Ping sensor Datasheet
  - MPU-6050 Datasheet
  - Barometer (BMP-280) Datasheet
  - I2C EEPROM Datasheet
  - GPS module Datasheet
  - 24AA515 EEPROM Datasheet
  - Microchip 93c66b EEProm Datasheet
  - You will need to get the DHT library and the Arduino development environment.
- You can buy all the hardware (including Arduino) at Sparkfun or on Amazon.
- You can buy kits with Arduinos, wires and discrete components and lots of sensors from Egloo, Sparkfun and lots of other places. A basic kit is about $50.

Raspberry Pi

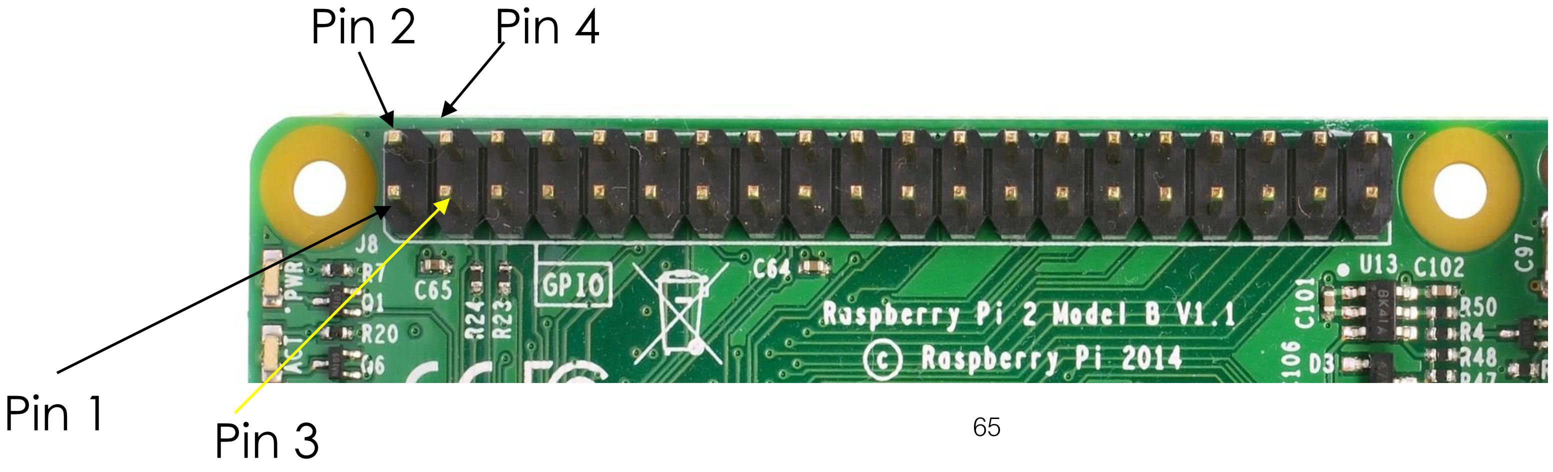
- Raspberry Pi computers come with a full CPU, an ARM microprocessor, (not just a microcontroller) and have memory and I/O ports (USB, Ethernet).
- It runs a standard operating system like Linux.
- Chromebooks are built using these sort of ARM chips.
- ARM processors can be 16, 32 or 64 bit and can run fairly powerful software.
- The Pi’s also have the GPIO pins just like an Arduino does.
- It costs about $35.
- Its an amazing amount of technology for so little money.
- You can do all the projects we do with the Arduino on a Raspberry Pi and send data over a local network or “log on” via a console interface directly.
- You can buy them at the same places you buy Arduinos.
- High end RP’s are pretty spiffy: 64 bit multicore processors with up to 16GB RAM

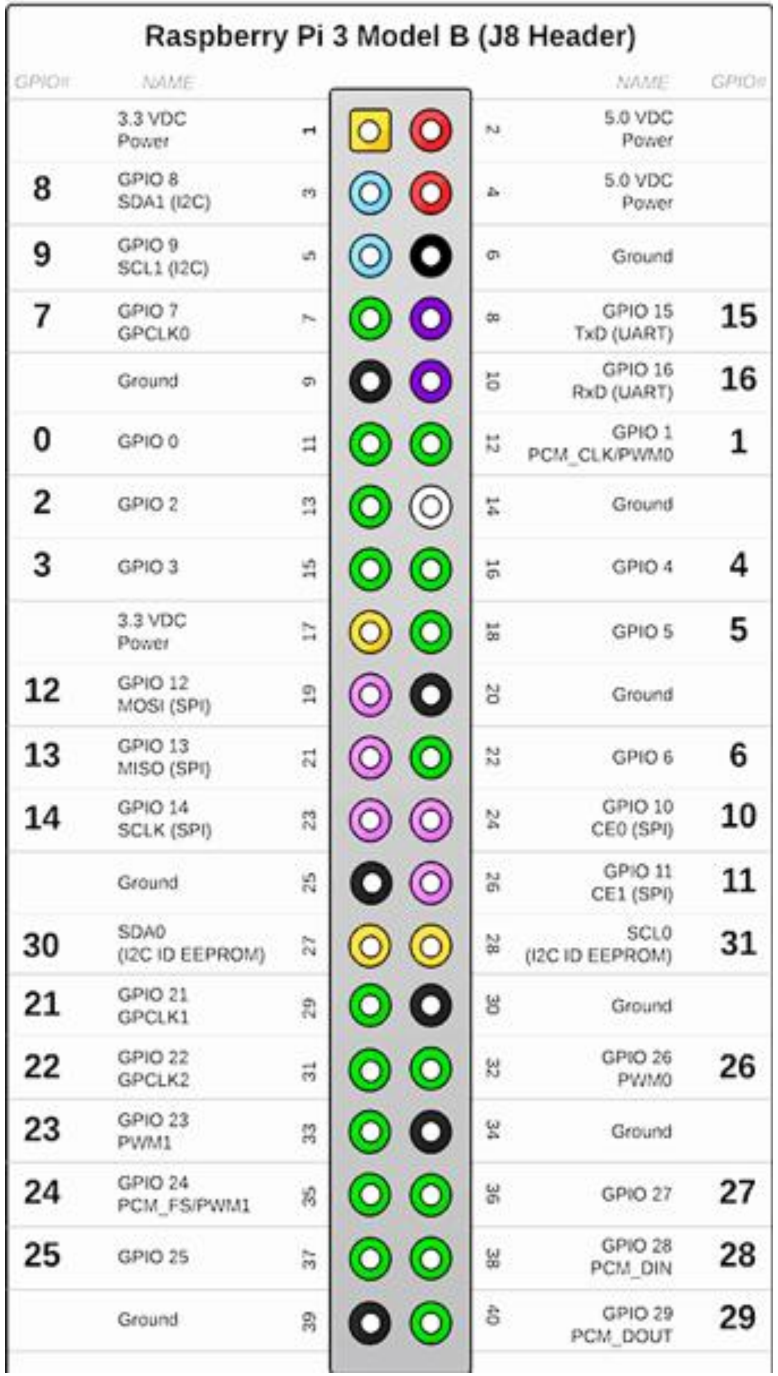
Raspberry Pi



Raspberry Pi GPIO pins

- PWM (pulse-width modulation)
  - Software PWM available on all pins
  - Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
- SPI
  - SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
  - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
- I2C
  - Data: (GPIO2); Clock (GPIO3)
  - EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)
- Serial
  - TX (GPIO14); RX (GPIO15)





Raspberry Pi 3 pins

- There are three different numbering schemes for:
  1. The pin ordering on the J8 header
  2. The pin number used by wiringPi (which we use)
  3. The BCM (GPIO HW interface) from Broadcom

Note: even numbered header pins are near edge of Raspberry Pi.

P1: The Main GPIO connector

WiringPi Pin	BCM GPIO	Name	Header	Header	Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header	Header	Name	BCM GPIO	WiringPi Pin

Raspberry Pi

- Unlike the Arduino, we need to install an OS (Linux) on the Raspberry Pi to use it. Once you do, you can develop, compile and run the IoT programs on the device.
- You'll load the OS and other files on a FAT-32 formatted SD card with at least 4 GB of capacity (I use 32 GB). Download the installer zip file at http://raspberrypi.org/downloads. Unzip it onto the SD card. You should notice the file bootloader.bin on the SD card. Plug the SD card into the pi and boot.
- Many of the Raspberry pi sensor examples are in python. For C++ examples, see here, here, here, here (accelerometers, gyros, compasses) and here. We use WiringPi almost exclusively to program GPIO pins. WiringPi supports SPI and I^2C interfaces; it is here and is pre-installed on newer Raspberry Pi OS images. Read the reference!
- Unlike the Arduino, Raspberry Pi GPIO pins *do not natively support analog signals*. I use the Adafruit MCP4725 Breakout Board - 12-Bit DAC w/I2C Interface [ADA935] and the Microchip MCP3008-I/P 10-Bit ADC with SPI to interface analog connections.

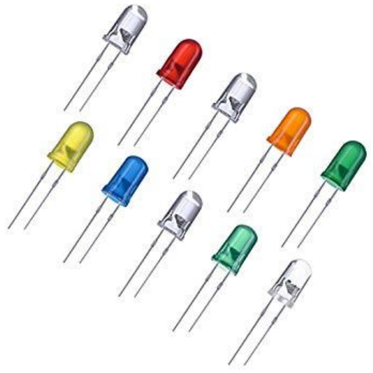
Labs

- Most of the rest of this section is lab exercises to get you used to IoT wiring and programing. We do some circuits and connect and program a number of sensors to Arduino's, Raspberry Pi's and in one case, directly to the USB port of a Linux machine. There's a partial list below. You should do as many of these as you can. Here, "A" means Arduino, "R" means Raspberry Pi and "L" means linux desktop.

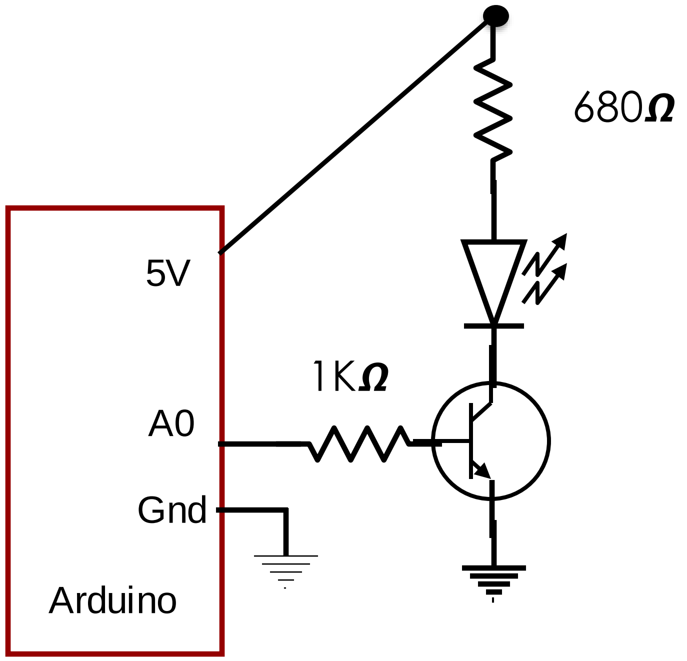
Goal	Parts	Device
Simple GPIO IF	LED, resistors	A, R
Analog interface	Variable resistor PCF8591 for R	A, R
Ultrasonic measurement	HC-SR04	A, R
Digital temperature	DHT-11	A, R
Digital acceleration/gyro	MPU-6050	A, R
Read/write EEprom	24FC515	A, R
UART	FDTI1232 for linux	A, R
Digital GPS	ATGM336H-5N	A, R, L
Segment display, encoder	SN595 SR	A
Keypad interface		A

Goal	Parts	Device
SPI interfacing		R
TFT(screen)	ILI9341 240X320	A
Digital pressure	BMP-180/280	A, R
Magnetic field	Hall sensor	A, R
Vibration, tilt		A, R
LCD (direct and I2C)	LCD-1602	A, R
RFID	RC522 rfid card	A
Stepper		A
Infrared		A
Radio	HC-12	A, RP

Lab: turn on an LED



- Wire the ground of the Arduino to.
- The code is in the appendix.

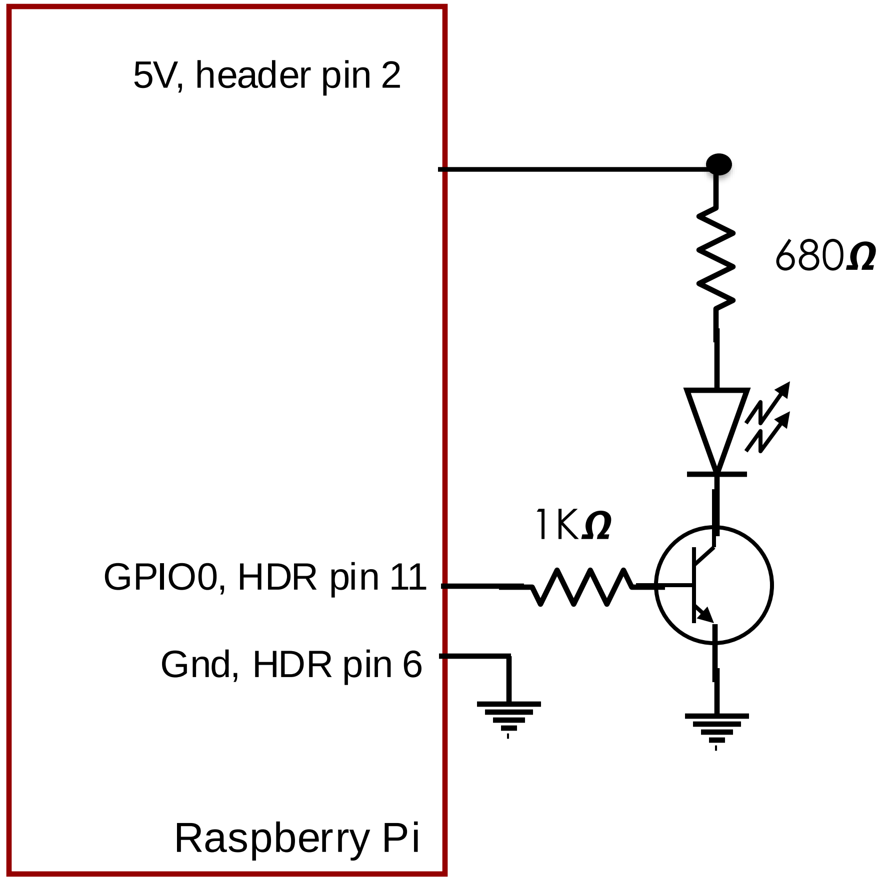


Arduino code for LED

```
// turn LED on and off
// Manferdelli

const int switchPin= 4;
const int oscDelay= 500;
const int analogOn= 255;
const int analogOff= 0;

void setup() {
  pinMode(switchPin, OUTPUT);
}

void loop() {
  for(;;) {
    analogWrite(switchPin, analogOff);
    delay(oscDelay);
    analogWrite(switchPin, analogOn);
    delay(oscDelay);
  }
}
```

Raspberry Pi code for LED

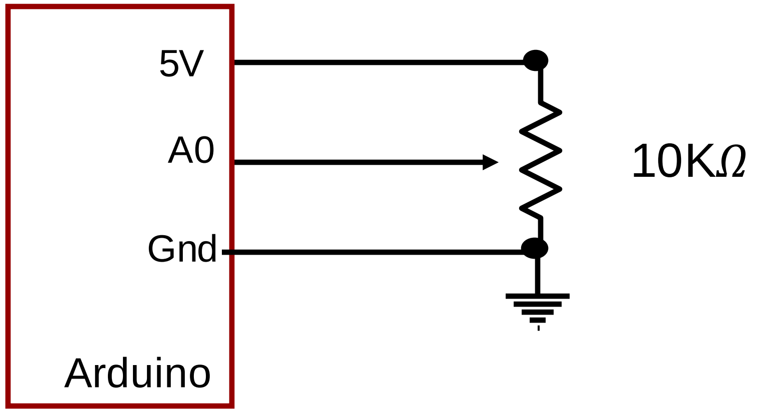
```
#include <stdio.h>
#include <stdint.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>

// turn LED on and off
// Manferdelli
const int switchPin = 0;
const int measurementDelay = 500;
const int loopCount = 250;

int main(int an, char** av) {
  wiringPiSetup ();
  pinMode (switchPin, OUTPUT);
  for (int i = 0; i < loopCount; i++) {
    digitalWrite(switchPin, HIGH);
    delay(measurementDelay);
    digitalWrite(switchPin, LOW);
    delay(measurementDelay);
  }
  return 0;
}
```



Lab: Measure voltage output

- Suppose you have a voltage output from a device that is proportional to some physical quantity (like the speed of a motor) between 0 and 5V, sampling the value on the pins of the Arduino lets you display or use that value in a control system.
- Let's simulate such source using the circuit on the right, varying the resistance will change data values. Wire the ground of the Arduino to the circuit as indicated.
- The code is in the appendix.
- Output (varying potentiometer)

```
ADC value:955, Voltage:4.67
ADC value:1021, Voltage:4.99
ADC value:988, Voltage:4.83
ADC value:876, Voltage:4.28
ADC value:726, Voltage:3.55
ADC value:638, Voltage:3.12
ADC value:596, Voltage:2.91
ADC value:457, Voltage:2.23
ADC value:357, Voltage:1.74
```



Arduino code for voltage measurement

```
// Measure analog level
// Manferdelli

const int inputPin= A0;
const int pinDelay= 500;
const double maxVoltage= 5.0;
const double maxRange= (double)1024;

void setup() {
  pinMode(inputPin, INPUT);
  Serial.begin(9600);
  Serial.print("Maximum voltage: ");
  Serial.print(maxVoltage);
  Serial.print(", Maximum range: ");
  Serial.print(maxRange);
  Serial.println("");
}
```

```
void loop() {
  int x;
  double y;

  for(;;) {
    delay(pinDelay);
    x= analogRead(inputPin);
    y= maxVoltage * ((double)x)/maxRange;
    Serial.print("Voltage: ");
    Serial.print(y);
    Serial.println("");
  }
}
```

Voltage measurement on Raspberry Pi

- The Raspberry Pi does not have analog pins. You must use an ADC/DAC to get analog values.
- We will use a popular ADC/DAC called the PCF8591 which has an I2C interface.
- We use the wiringPiI2C library to interface to I2C.
- The code is the same for all analog interfacing.
- Note: You must enable the Raspberry Pi I2C kernel module or it won't work. To do this, run raspi-config and use the interfacing option.
- The connection to the PCF8591 are:

PCF	RP
1(ain0)	analog-signal
5, 6, 7, 8	gnd (pin 6)
16	3.3 (pin 1)
14	3.3v
13	gnd
12	gnd
10(scl)	rp pin 5 (scl)
9(sda)	rp pin 3 (sda)

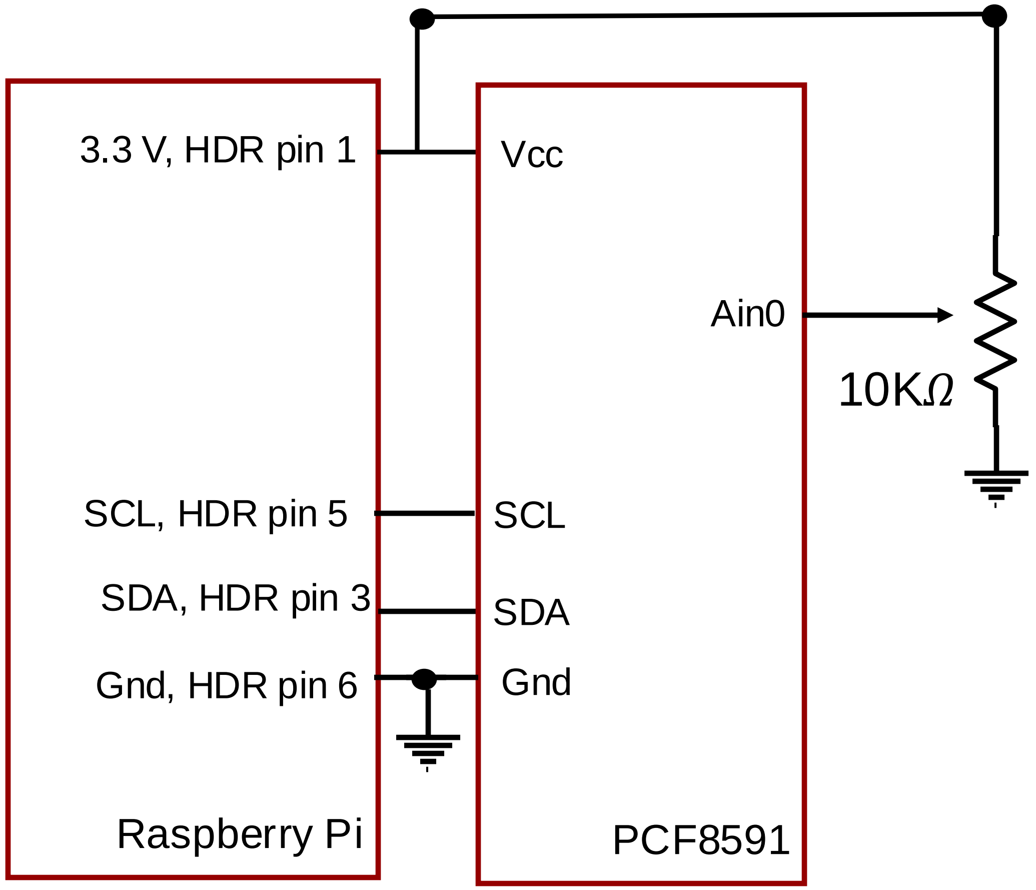
PCF8591 pinout and connections

- The PCF8591 which employs a I2C interface to the Raspberry Pi.
  - Pin 1 – Ain_0
  - Pin 2 – Ain_1
  - Pin 3 – Ain_2
  - Pin 4 - Ain_3
  - Pin 5 – A_0
  - Pin 6 – A_1
  - Pin 7 – A_2
  - Pin 8 - V_{SS}
  - Pin 16 - V_{DD}
  - Pin 15 - A_{out}
  - Pin 14 - V_{REF}
  - Pin 13 - A_{GND}
  - Pin 12 - EXT
  - Pin 11 - OSC
  - Pin 10 - SCL
  - Pin 9 – SDA

- Raspberry Pi connections
  - SDA (PCF8591) → SDA
  - SCL → SCL
  - 3.3V → V_{CC}
  - GND → GND
  - Ain0 → measured quantity
  - Ground A_0, A_1, A_2, V_{SS}
  - Aout → through diode and 1K resistor
  - AGND, EXT → GND
  - Use SCL to SDA through pull-up (10K)

Voltage measurement on Raspberry Pi

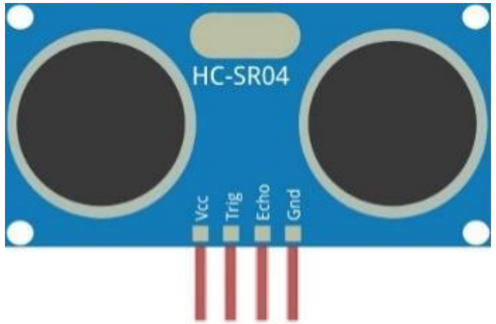
```
// Manferdelli
//   Raspberry Pi analog read

#include <stdio.h>
#include <wiringPi.h>
#include <pcf8591.h>
const int base = 120;
const int i2c_address = 0x48;

int main(int an, char** av) {
  int v = 0;

  if (wiringPiSetup() < 0) {
    printf("Can't initialize wiringPi\n");
    return 1;
  }
  pcf8591Setup(base, i2c_address);

  for(;;) {
    v = analogRead(base);
    delay(10);
    printf("Voltage: %d\n", v);
  }
  return 0;
}
```

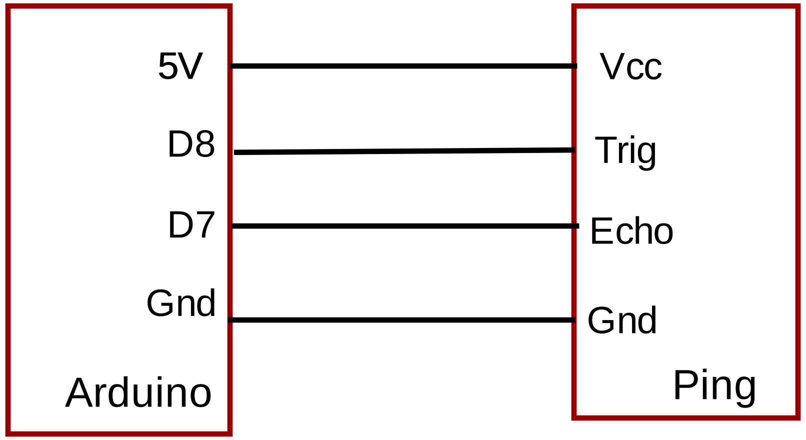


Lab: Measure Distance

- Wire the the Arduino to the "ping" sensor ([HC-SR04]) as indicated.
- The Ping sensor only works up to about 1 meter.
- The code is in the appendix. The output below was obtained moving an object at varying distances from the sensor. Distance is in cm.

7	9	13
6	5	5
10	14	3
15	22	4
21	25	3
26	30	108
30	33	107
31	36	
24	38	
33	24	
10		





Arduino code for measure distance

```
// Measure distance with ping sensor
// Manferdelli

const int triggerPin= 8;
const int echoPin= 7;
const int measurementDelay= 500;
const int pingDelay= 5;
const double temp = 20; // C
const double soundSpeed = 331.5 + .6 * temp;  // m/s

void setup() {
  Serial.begin(9600);
  Serial.println("Ping sensor test");
  pinMode(triggerPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

// pulseIn reads a pulse (either HIGH or LOW) on a pin. If value is HIGH,
// pulseIn() waits for the pin to go from LOW to HIGH to starts timing,
// it then waits for the pin to go LOW and stops timing.
// Returns the length of the pulse in microseconds.
```

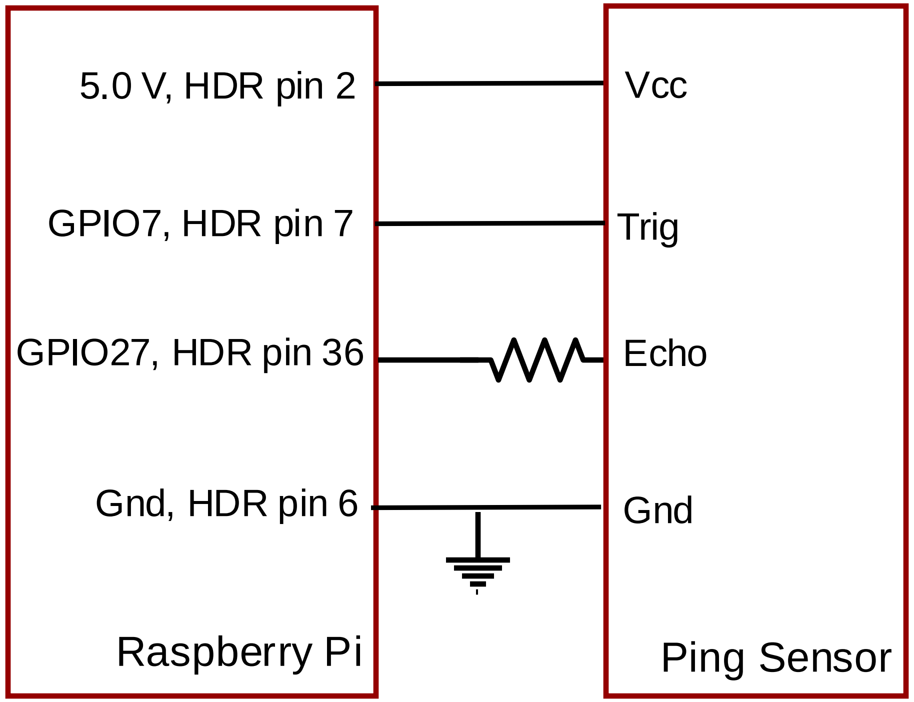
pulseIn(pin, value): Starts timer when pin goes from !value to value and stops timer when pin goes beack to !value.

Arduino code for measure distance

```
void measure_distance() {
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(3);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(triggerPin, LOW);
  double return_time_microsecs = pulseIn(echoPin, HIGH);
  double one_way_return_time = return_time_microsecs / 2e6;
  double d = soundSpeed * one_way_return_time * 100.0; // cm
  Serial.print("Measured distance (cm): ");
  Serial.println(d);
}

void loop() {
  for(;;) {
    delay(measurementDelay);
    measure_distance();
  }
}
```

Ping sensor and raspberry pi

- Use gpio 7 (pin 7) and gpio 27 (pin 36).
- Wiring below. Sensor pins are vcc, trigger, echo gnd (left to right from front)



RP code for measure distance

```
//  Manferdelli
//    Raspberry Pi, ping sensor

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <wiringPi.h>
#ifndef byte
typedef unsigned char byte;
#endif

//  Connections
//    Ping        RP
//    trigger     gpio 7 (pin 7)
//    echo        gpio 27 (pin 36)
//    vcc         5v
//    gnd         gnd

// I had to use a scope to check,
// RP pin numbering is mysterious.
// Look at the ifdef'd code below
//   to see it.
const int trigger_pin = 7;
const int echo_pin = 27;
```

```
const int max_distance = 500; // cm
long unsigned time_out_period;
const double sound_speed = 340.0; // speed sound(m/s)

long pulseIn(int pin, int value, int time_out) {

  pinMode(echo_pin, INPUT);
  long unsigned start = micros();
  while(digitalRead(pin) != value) {
    if ((micros() - start) > time_out_period)
      return 0;
  }
  long unsigned set = micros();
  while(digitalRead(pin) == value) {
    if ((micros() - set) > time_out_period)
      return 0;
  }
  long unsigned end = micros();
  printf("pulseIn, end: %d, start: %d\n",
    end, start);
  return end - set;
}
```

RP code for measure distance

```
double getDistance() {
#if 1
  pinMode(trigger_pin, OUTPUT);
  pinMode(echo_pin, OUTPUT);
  digitalWrite(trigger_pin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigger_pin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigger_pin, LOW);
#else
  // Find pins
  pinMode(trigger_pin, OUTPUT);
  pinMode(echo_pin, OUTPUT);
  for(;;) {
    digitalWrite(trigger_pin, LOW);
    delayMicroseconds(1000);
    digitalWrite(trigger_pin, HIGH);
    delayMicroseconds(1000);
    digitalWrite(echo_pin, LOW);
    delayMicroseconds(1000);
    digitalWrite(echo_pin, HIGH);
    delayMicroseconds(1000);
}
#endif
  long unsigned ping_time = pulseIn(echo_pin, HIGH,
                                    time_out_period);
  double d = (((double)ping_time) * sound_speed) / 20000.0;
  return d;
}

int main(int an, char** av) {
  if (wiringPiSetup() < 0) {
    printf("Can't initialize Wiring Pi\n");
    return 1;
  }

  time_out_period = ((max_distance*20000)/sound_speed)+1;

  double d;
  for(;;) {
    d = getDistance();
    printf("Target distance is %8.2lf cm\n", d);
    delay(1000);
  }
  return 0;
}
```

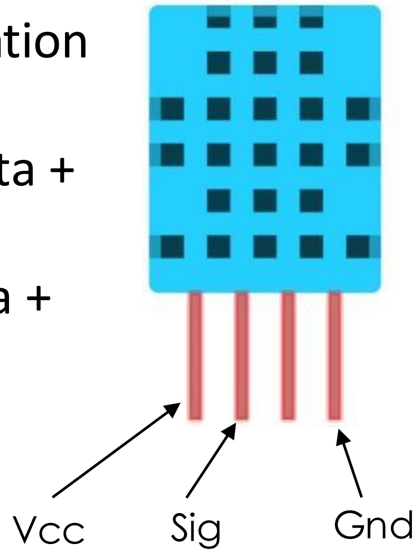
DHT11 & DHT22 Temperature Sensors

- The DHT22 temperature measuring range is from -40 to +125 degrees Celsius +-0.5 degree. Humidity: from 0 to 100% with 2-5% accuracy.
- DHT11 temperature range is 0 to 50 degrees Celsius with +-2 degrees. Humidity: from 20 to 80% with 5% accuracy.
- Operating voltage of both sensors is from 3 to 5 volts.
- Humidity sensor has two electrodes with moisture holding substrate between them. Conductivity of the substrate changes and the resistance between these electrodes changes. This change in resistance is measured.
- Temperature sensor is NTC (negative temperature coefficient) thermistor made by sintering of semiconductive materials such as ceramics or polymers in order to provide large changes in the resistance with just small changes in temperature.

Lab: Measure Temperature

- Wire the Arduino as shown (right) to the DHT-11 sensor.
- Single-bus data format is used for communication and synchronization. One communication process is about 4ms. Data consists of decimal and integral parts. A complete data transmission is 40bit. Sensor sends higher data bit first. Data format: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

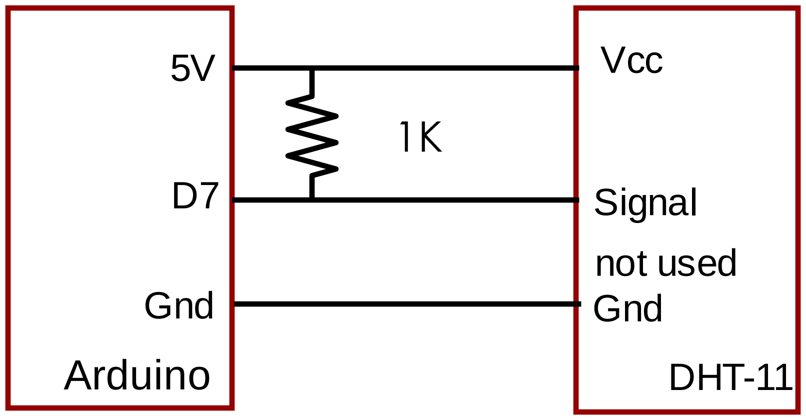
Freenove



- The code is in the appendix. Output taken in my apartment. I need to turn up the heat.

- Output

```
Humidity: 20%
Temperature: 17C
Temperature: 62.60F

Humidity: 20%
Temperature: 17C
Temperature: 62.60F
```



Arduino code for measure temperature

```
// Measure temperature and humidity with DHT-11
// Manferdelli

const int dataPin= 8;
const int measurementDelay= 500;
typedef uint8_t byte;

void setup() {
  Serial.begin(9600);
}

void clear_buf(byte* p, int size) {
  for (int i = 0; i < size; i++)
    p[i] = 0;
}

const int initReadDelay = 18;
void start_measurement() {
  pinMode(dataPin, OUTPUT);
  digitalWrite(dataPin, LOW);
  delay(initReadDelay);
  digitalWrite(dataPin, HIGH);
}
```

Arduino code for measure temperature

```
const int packetSize = 40;      // bits
const int pulseOneLength = 40;  // pulse length to be 1
const int pulseDelay= 18;
int measure_th(int* t, int* h) {
  byte buf[8];
  clear_buf(buf, 8);

  byte byte_index = 0;
  byte bit_cnt = 7;

  pinMode(dataPin, INPUT);
  delayMicroseconds(pulseDelay);
  pulseIn(dataPin, HIGH);

  // Collect each data bit
  int time_out = 10000;
  for (int i = 0; i < packetSize; i++) {
    while (digitalRead(dataPin) == LOW) {
      if (time_out-- <= 0)
        return -1;
    }
    time_out = 10000;
    unsigned long t = micros();
```

Arduino code for measure temperature

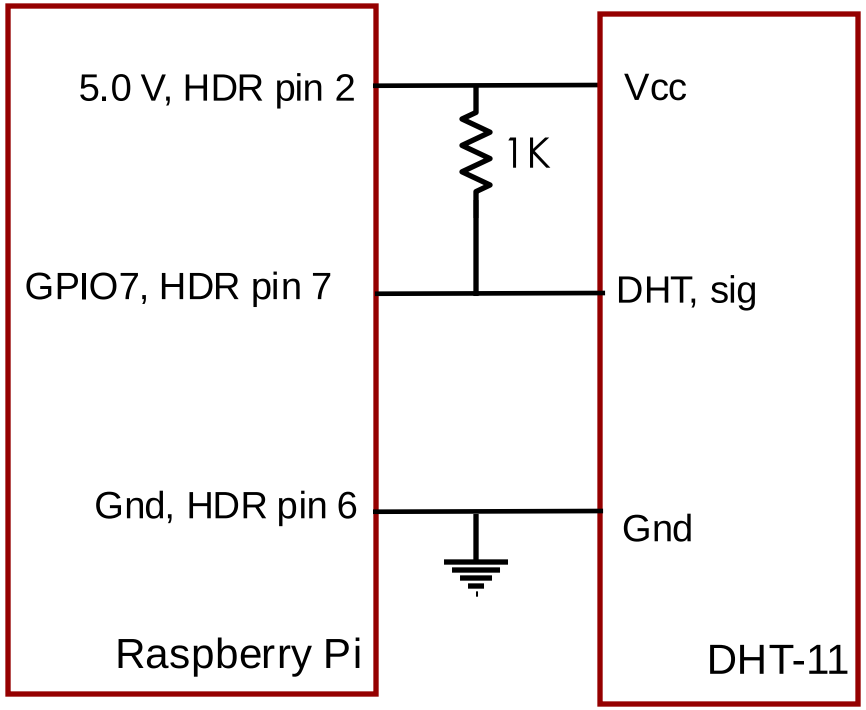
```
while (digitalRead(dataPin) == HIGH) {
      if (time_out-- <= 0)
        return -1;
    }
    if ((micros() - t) > pulseOneLength) {
      buf[byte_index] |= 1 << bit_cnt;
    }
    if (bit_cnt == 0) {
      byte_index++;   // next byte
      bit_cnt = 7;    // MSB
    } else {
      bit_cnt--;
    }
  }

  *t = buf[2];
  *h = buf[0];
  if ((buf[0] + buf[2]) != buf[4])
    return -1;

  return 0;
}
```

Arduino code for measure temperature

```
void read_dht11(int* t, int* h) {
  start_measurement();
  if (measure_th(t, h) < 0) {
    *t = 0.0;
    *h= 0.0;
    return;
  }
}

void loop() {
  int temperature, humidity;
  for(;;) {
    delay(measurementDelay);
    read_dht11(&temperature, &humidity);
    Serial.print("Temp (C): ");
    Serial.print(temperature);
    Serial.print(", Temp (F): ");
    double temp_F = ((double) temperature) * 9.0 / 5.0 + 32.0;
    Serial.print(temp_F);
    Serial.print(", Humidity (%): ");
    Serial.print(humidity);
    Serial.println("");
  }
}
```

Raspberry Pi code for measure temperature

```
// Manferdelli
//  Raspberry Pi interface to dht11
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <wiringPi.h>
#ifndef byte
typedef unsigned char byte;
#endif

void clearBuf(byte* buf, int n) {
  for (int i = 0; i < n; i++) {
    buf[i] = 0;
  }
}

void printBuf(byte* in, int n) {
  for (int i = 0; i < n; i++) {
    printf(" %02x", in[i]);
    if ((i % 16) == 15) {
      printf("\n");
    }
  }
  printf("\n");
}
```



Raspberry Pi code for measure temperature

```
const int dht_pin = 7;
const int timingDelay = 18;
const int messageDelay = 2000;
const int max_timings = 90;
const int bits_to_read = 40;
#define DATA_BUF_SIZE 10

class temp_sensor {
public:
  double h_;
  double c_;
  double f_;
};
bool get_dht_data(byte* data, int size, temp_sensor* r) {
  if (size < 5)
    return false;
  byte last_state = HIGH;
  int counter = 0;
  int i = 0;
  int j = 0;

  pinMode(dht_pin, OUTPUT);
  digitalWrite(dht_pin, LOW);
  delay(timingDelay);
  pinMode(dht_pin, INPUT);
```

Raspberry Pi code for measure temperature

```
// Detect change
for (i = 0; i < max_timings; i++) {
  counter = 0;
  while(digitalRead(dht_pin) == last_state) {
    counter++;
    delayMicroseconds(1);
    if (counter == 255)
      break;
  }
  last_state = digitalRead(dht_pin);

  // ignore first 3 transitions
  if (i>=4 && (i%2)==0) {
    data[j/8] <<= 1;
    if (counter > 16)
      data[j/8] |= 1;
    j++;
  }
}
// Make sure we read 40 bits and verify checksum
if (j < bits_to_read)
  return false;
byte check_sum = data[0] + data[1] + data[2] + data[3];
if (check_sum != data[4])
  return false;
```

Raspberry Pi code for measure temperature

```
// Make sure we read 40 bits and verify checksum
if (j < bits_to_read)
  return false;
byte check_sum = data[0] + data[1] + data[2] + data[3];
if (check_sum != data[4])
  return false;

// calculate readings
int t;

// humidity
t = data[0];
t <<= 8;
t += data[1];
r->h_ = ((double)t)/10.0;
if (r->h_ > 100.0)
  r->h_ = ((double)data[0]);
```

Raspberry Pi code for measure temperature

```
// temperature
  t = data[2] & 0x7f;
  t <<= 8;
  t += data[3];
  r->c_ = ((double)t) / 10.0;
  if (r->c_ > 125)
    r->c_ = ((double) data[2]);
  if (data[2] & 0x80)
    r->c_ = -r->c_;
  r->f_ = 1.8 * r->c_ + 32.0;

  return true;
}

// Program needs to be run as su
```

```
int main(int an, char** av) {
  if (wiringPiSetup() < 0) {
    printf("Can't initialize wiringPi\n");
    return 1;
  }

  temp_sensor r;
  byte data[DATA_BUF_SIZE];
  for(;;) {
    clearBuf(data, DATA_BUF_SIZE);
    if (get_dht_data(data, DATA_BUF_SIZE,&r))
      printf("Humidity: %7.3lf, Temperature: %7.3lf
(C), %7.3lf (F)\n",
        r.h, r.c, r.f);
    delay(messageDelay);
  }

  return 0;
}
```

Accelerometer Sensors

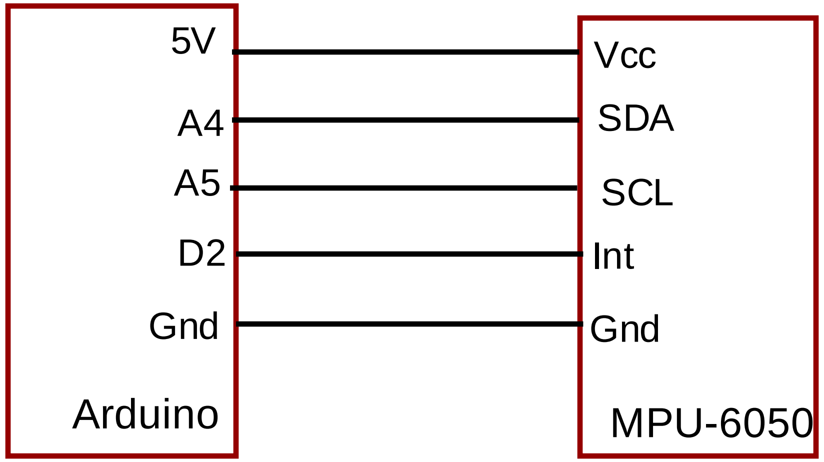
- Accelerometers can measure acceleration on one, two, or three axes. 3-axis units are becoming more common as the cost of development for them decreases.
- MEMS accelerometers contain capacitive plates internally. Some of these are fixed, while others are attached to minuscule springs that move internally as acceleration forces act upon the sensor. As these plates move in relation to each other, the capacitance between them changes. From these changes in capacitance, the acceleration can be determined.
- Most accelerometers will have a selectable range of forces they can measure. These ranges can vary from ±1g up to ±250g.
- Accelerometers will communicate over an analog, digital, or pulse-width modulated connection interface.
  - The analog interface shows accelerations through varying voltage levels. An ADC on a microcontroller can then be used to read this value. These are generally less expensive than digital accelerometers.
  - Accelerometers with a digital interface can either communicate over SPI or I^2C communication protocols.
  - Accelerometers that output data over pulse-width modulation (PWM) output square waves with a known period, but a duty cycle that varies with changes in acceleration.

Lab: Measure Acceleration and angular velocity MPU-6050

- The MPU-6050 is gyro, accelerometer and temperature sensor. Wire the Arduino as shown.
- Here is sample output:

```
A.009g, 0.060g)
Temperature (C): 21.54
Gyro: x,y,z (12.49 deg/s, -0.53 deg/s, -4.Accelerometer: x,y,z (1.027g,
0.238g, -0.080g)

Temperature (C): 21.68
Gyro: x,y,z (-213.01 deg/s, 3.59 deg/s, -4.07 deg/s)
Accelerometer: x,y,z (-0.387g, 0.137g, -0.962g)
Temperature (C): 21.68

Gyro: x,y,z (50.60 deg/s, 147.03 deg/s, 2.34 deg/s)
Accelerometer: x,y,z (0.718g, -0.308g, -0.134g)
Temperature (C): 21.78
```

SparkFun



Output continued

```
Gyro: x,y,z (25.02 deg/s, 33.97 deg/s, -250.14 deg/s)
Accelerometer: x,y,z (0.193g, 0.934g, -0.269g)
Temperature (C): 21.78

Gyro: x,y,z (4.58 deg/s, -10.77 deg/s, -6.54 deg/s)
Accelerometer: x,y,z (0.997g, 0.022g, -0.366g)
Temperature (C): 21.87

Gyro: x,y,z (21.85 deg/s, 86.79 deg/s, 64.41 deg/s)
Accelerometer: x,y,z (0.533g, -0.249g, -0.236g)
Temperature (C): 21.78

Gyro: x,y,z (11.31 deg/s, -109.04 deg/s, 63.29 deg/s)
Accelerometer: x,y,z (0.423g, -0.120g, -0.271g)
Temperature (C): 21.87

Gyro: x,y,z (14.21 deg/s, 41.51 deg/s, -42.22 deg/s)
Accelerometer: x,y,z (1.027g, 0.016g, 0.061g)
Temperature (C): 21.78
```

```
Gyro: x,y,z (12.65 deg/s, -0.35 deg/s, -4.89 deg/s)
Accelerometer: x,y,z (1.022g, 0.016g, 0.066g)
Temperature (C): 21.82

Gyro: x,y,z (12.54 deg/s, -0.21 deg/s, -4.98 deg/s)
Accelerometer: x,y,z (1.028g, 0.011g, 0.065g)
Temperature (C): 21.78

Gyro: x,y,z (12.67 deg/s, -0.40 deg/s, -4.97 deg/s)
Accelerometer: x,y,z (1.023g, 0.015g, 0.061g)
Temperature (C): 21.92
```

Arduino code for measure acceleration

```
// Read accelerometer (MPU-6050)
// Manferdelli

#include <Wire.h>

typedef uint8_t byte;

const byte i2c_address= 0x68;
const byte sleep_mgmt= 0x6b;
const byte acc_x_out= 0x3b;
const int zero_pt= -512 - (340 * 35);
const double acc_normalizer= 16384.0;
const double gyro_normalizer= 131.0;
const double temp_normalizer= 340.0;

const int measurementDelay= 750;

//
// SDL --> A5
// SDA --> A4
// int --> D2
```

Arduino code for measure acceleration

```
typedef uint8_t byte;
struct pdu {
  int16_t x_acc;
  int16_t y_acc;
  int16_t z_acc;
  int16_t temp;
  int16_t x_gyro;
  int16_t y_gyro;
  int16_t z_gyro;
};

void setup() {
  Serial.begin(9600);
  Wire.begin();
  write_i2c(sleep_mgmt, 0x00);
}

int16_t swap_bytes(int16_t in) {
  return ((in << 8) & 0xff00) | ((in >> 8) & 0x00ff);
}
```

Arduino code for measure acceleration

```
int read_i2c(byte r, byte* buf, int size) {
  Wire.beginTransmission(i2c_address);
  Wire.write(r);
  Wire.endTransmission(false);
  Wire.requestFrom(i2c_address, size, true);

  int i= 0;
  while (Wire.available() && i < size) {
    buf[i++]= Wire.read();
  }
  return i;
}

void write_i2c(byte r, byte d) {
  Wire.beginTransmission(i2c_address);
  Wire.write(r);
  Wire.write(d);
  Wire.endTransmission(true);
}
```

Arduino code for measure acceleration

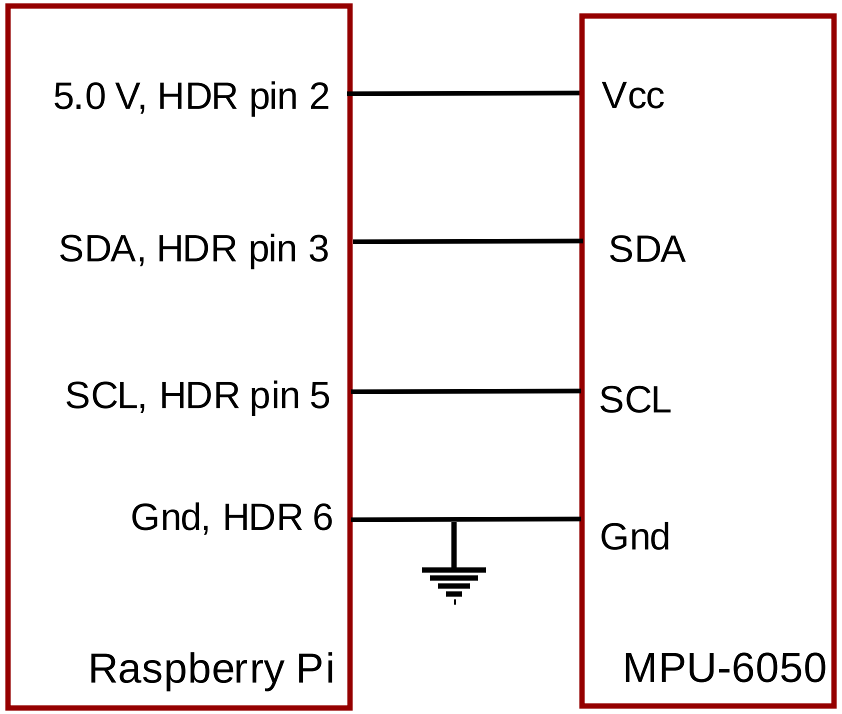
```
void loop() {
  pdu p_out;
  read_i2c(acc_x_out, (byte*)&p_out, sizeof(pdu));
  p_out.x_acc= swap_bytes(p_out.x_acc);
  p_out.y_acc= swap_bytes(p_out.y_acc);
  p_out.z_acc= swap_bytes(p_out.z_acc);
  p_out.temp= swap_bytes(p_out.temp);
  p_out.x_gyro= swap_bytes(p_out.x_gyro);
  p_out.y_gyro= swap_bytes(p_out.x_gyro);
  p_out.z_gyro= swap_bytes(p_out.x_gyro);

  double acc_x= ((double)p_out.x_acc) / acc_normalizer;
  double acc_y= ((double)p_out.y_acc) / acc_normalizer;
  double acc_z= ((double)p_out.z_acc) / acc_normalizer;

  double t= (p_out.temp - zero_pt) / temp_normalizer;

  double gyro_x= ((double)p_out.x_gyro) / gyro_normalizer;
  double gyro_y= ((double)p_out.y_gyro) / gyro_normalizer;
  double gyro_z= ((double)p_out.z_gyro) / gyro_normalizer;
```

Arduino code for measure acceleration

```
  Serial.println("");
  Serial.println("-----------------------------");
  Serial.print("Acceleration (g): (");
  Serial.print(acc_x);
  Serial.print(", ");
  Serial.print(acc_y);
  Serial.print(", ");
  Serial.print(acc_z);
  Serial.println(")");
  Serial.print("Temperature (C): ");
  Serial.print(t);
  Serial.println("");
  Serial.print("Gyro (deg/sec): (");
  Serial.print(gyro_x);
  Serial.print(", ");
  Serial.print(gyro_y);
  Serial.print(", ");
  Serial.print(gyro_z);
  Serial.println(")");
  Serial.println("-----------------------------");
  Serial.println("");
  delay(measurementDelay);
}
```

Raspberry Pi code for measure acceleration

```
//  Manferdelli
//    Raspberry Pi, mpu-6050

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>

#ifndef byte
typedef unsigned char byte;
#endif

const byte i2c_address = 0x68;
const byte x_address= 0x3b;
const byte y_address = 0x3d;
const byte z_address = 0x3f;
const double pi = 3.1415926535;
```



Raspberry Pi code for measure acceleration

```
//  Connection scheme
//      RP            6050
//      Gnd(HDR 6)    Gnd
//      scl(HDR 5)    scl
//      5v (HDR 1)    Vcc
//      sda(HDR  3)   sda

Int read_int(int fd, byte address) {
  int r = 0;
  int v = 0;

  v =  wiringPiI2CReadReg8(fd, address);
  v <<= 8;
  v |=  wiringPiI2CReadReg8(fd, address + 1);
  if (v &0x8000)
    v = -(65536 - v);
  return v;
}

Double dist(double a, double b) {
  return sqrt(a * a + b * b);
}
```

Raspberry Pi code for measure acceleration

```
double get_x_rotation(double x, double y, double z) {
  double s = dist(x, z);
  if (s == 0.0) {
    if (y < 0.0)
      return -90.0;
    return 90.0;
  }
  double r = atan(y / s);
  return -(r * 180.0 / pi);
}

double get_y_rotation(double x, double y, double z) {
  double s = dist(y, z);
  if (s == 0.0) {
    if (x < 0.0)
      return -90.0;
    return 90.0;
  }
  double r = atan(x / s);
  return -(r * 180.0 / pi);
}
```

Raspberry Pi code for measure acceleration

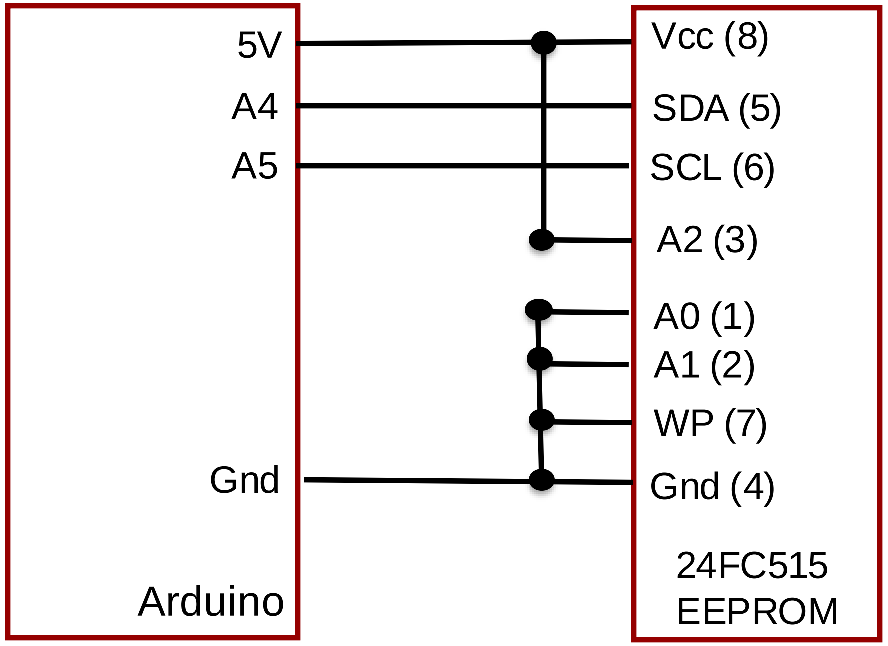
```
int main(int an, char** av) {
  int fd = wiringPiI2CSetup(i2c_address);
  if (fd < 0) {
    printf("Can't initialize Wiring Pi\n");
    return 1;
  }
  wiringPiI2CWriteReg8(fd, 0x6b, 0); // disable sleep
  int a_x, a_y, a_z;
  double scaled_x, scaled_y, scaled_z;
  double rot_x, rot_y;
  for(;;) {
    a_x = read_int(fd, x_address);
    a_y = read_int(fd, y_address);
    a_z = read_int(fd, z_address);
    scaled_x = ((double)a_x) / 16384.0;
    scaled_y = ((double)a_y) / 16384.0;
    scaled_z = ((double)a_z) / 16384.0;
    rot_x =  get_x_rotation(scaled_x, scaled_y, scaled_z);
    rot_y =  get_y_rotation(scaled_x, scaled_y, scaled_z);
    printf("Scaled acceleration: (%07.2f, %07.2f, %07.2f), x-y rotation: (%07.2f, %07.2f)\n",
      scaled_x, scaled_y, scaled_z, rot_x, rot_y);
    delay(1000);
  }
  return 0;
}
```

EEproms

- **EEPROM**, or **E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory, is a type of device that allows you to store small chunks of data and retrieve it later even if the device has been power cycled.
- Serial EEPROM devices like the Microchip 24-series EEPROM allow you to add more memory to any device that can speak I²C.
- 256 Kbit EEPROM contains 32000 byte addressable.
- Most Significant and Least Significant Bytes
  - Address is two bytes.
  - First we send the Most Significant Byte (MSB) — the first 8 bits in this case.
  - Then we send the Least Significant Byte (LSB) — the second 8 bits.
- Most EEPROM devices have something called a “page write buffer” which allows you to write multiple bytes at a time the same way you would a single byte.

Lab: Read/write an EEPROM

- The 24FC515 EEProm communicates over I^2C .
- Wire the Arduino to the eeprom as indicated.
- The code follows.



Arduino code for EEprom

```
// eeprom read/write
//   Manferdelli
#include <Wire.h>

typedef uint8_t byte;
const int measurementDelay= 500;
const int initial_clock_setting= 400000;
const uint32_t maxMillis= 2000;
#define MAX_BYTES 64

// Theoretically, the 24LC256 has a 64-byte page write buffer but
// we'll write 16 at a time.
#define MAX_I2C_WRITE 16

// This address is determined by the way your address pins are wired.
// In the diagram from earlier, we connected A0 and A1 to Ground and
// A2 to 5V. To get the address, we start with the control code from
// the datasheet (1010) and add the logic state for each address pin
// in the order A2, A1, A0 (100) which gives us 0b1010100, or in
// Hexadecimal, 0x54.
const byte eeprom_address= 0x54;
```

Arduino code for EEprom

```
// Read
byte readEEProm(uint32_t address) {
  Wire.beginTransmission(eeprom_address);
  Wire.write((int) (address >> 8));     // MSB
  Wire.write((int) (address & 0xff));   // LSB
  Wire.endTransmission();
  Wire.requestFrom(eeprom_address, 1);
  byte r = 0xff;
  if (Wire.available())
    r= Wire.read();
  return r;
}

// Write
void writeEEProm(uint32_t address, byte* data, int size) {
  Wire.beginTransmission(eeprom_address);
  Wire.write((int) (address >> 8));     // MSB
  Wire.write((int) (address & 0xff));   // LSB
  for (byte  b = 0; b < size; b++)
    Wire.write(data[b]);
  Wire.endTransmission();
}
```

Arduino code for EEprom

```
void printHexByte(byte b) {
  if (b < 16)
    Serial.print("0");
  Serial.print(b, HEX);
}

int readStream(byte* buf, int max) {
  int num_bytes = 0;
  for(;;) {
    while(Serial.available()) {
      if (num_bytes >= max)
        return num_bytes;
    }
    buf[num_bytes++] = Serial.read();
  }
  return num_bytes;
}

int initBuf(byte* buf, int max) {
  int i = 0;
  for (i = 0; i < 64; i++)
    buf[i] = (byte) (i + 16);
  return i;
}
```

Arduino code for EEprom

```
void printBytes(byte* buf, int n) {
  for (int i = 0; i < n; i++) {
    printHexByte(buf[i]);
    Serial.print(" ");
    if ((i%16) ==15)
      Serial.println("");
  }
}

uint32_t end_address= 0x40; //0x7d00;
void setup() {
  Serial.begin(9600);
  Wire.begin();
  Wire.setClock(initial_clock_setting);

  byte to_write[MAX_BYTES];
  int current = 0;
  int num_bytes= initBuf(to_write, MAX_BYTES);
  int num_left= num_bytes;
  int n;

  Serial.println("Bytes accepted:");
  printBytes(to_write, num_bytes);
  Serial.println("");
```

Arduino code for EEprom

```
  while(num_left > 0) {
    if (num_left >= MAX_I2C_WRITE)
      n = MAX_I2C_WRITE;
    else
      n = num_left;

    writeEEProm(current, &to_write[current], n);
    num_left -= n;
    current += n;
  }
  Serial.println("Bytes written");
  Serial.println("");

  // Read bytes to end_address
  Serial.println("Bytes read:");
  byte v;
  for (uint32_t a = 0; a < end_address; a++) {
    v= readEEProm(a);
    printHexByte(v);
    if ((a%16) ==15)
      Serial.println("");
  }
}
void loop() {
}
```

EEprom Raspberry pi

- We could simulate the I2C interface as we did with the Arduino but the Raspberry Pi has some utilities you can use instead.
  - I2cdetect – detects I2C devices connected to the RP
  - I2cdump – reads I2C registers
  - I2cget – reads specific I2C registers
  - I2cset – sets specific I2C registers
- If you wire the 24FC515 just as in the Arduino case but with Vcc to RP 3.3 (pin 1), gnd to RP gnd (pin 6), sda to RP sda (pin 3) and scl to RP scl (pin 5), for example, the following command will read out the ROM. Try it on a ROM you've initialized with the Arduino.
  - i2cdump "bcm2835 I2C adapter" 0x54

GPS

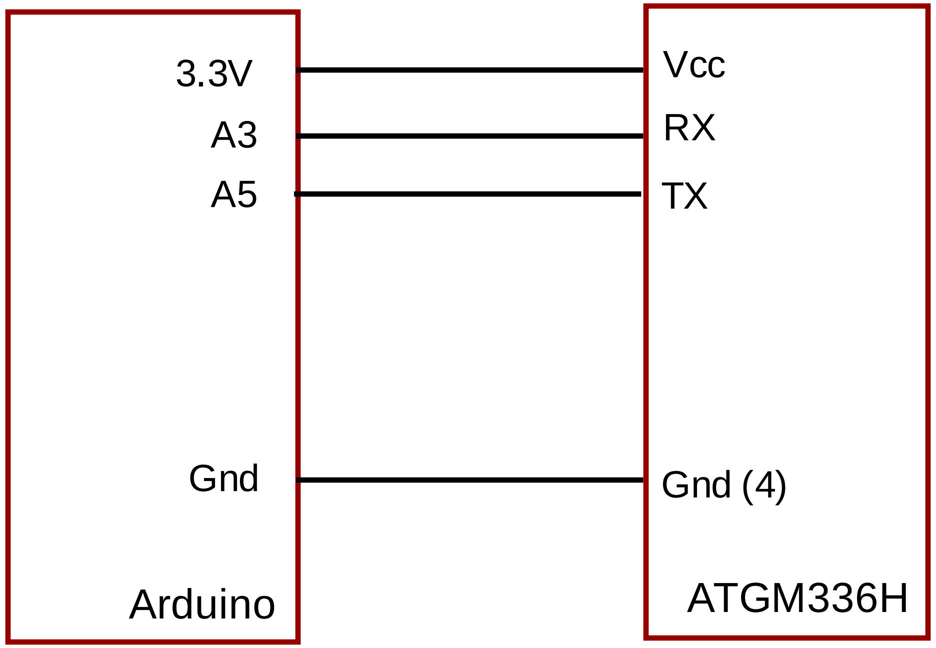
- For general information on GPS, see my “Electronics of Radio” Notes.
  - Accuracy for C/A: 5 meters. 3 meters with WAAS
  - Four sats for latitude, longitude, elevation, and time.
- The following are datasheets and command sets for some of the more common chipsets. UBLOX has a chipset
  - SiRF NMEA Reference Manual
  - SiRF Binary Reference Manual
- Serial data out of a transmit pin (TX) usually at 9600bps for 1Hz receivers but 57600bps is also common.
- Channels affect time to first fix (TTFF) when 4 sats are in view.
- NMEA is output data format that most GPS modules use.
  - $GPRMC,235316.000,A,4003.9040,N,10512.5792,W,0.09,144.75,141112,,*19
  - $GPGGA,235317.000,4003.9039,N,10512.5793,W,1,08,1.6,1577.9,M,-20.7,M,,0000*5F (time[23:53 16.0GMT], long [40.30], lat [105.12W], sats [8], alt [1577.9 meters])
  - $GPGSA,A,3,22,18,21,06,03,09,24,15,,,,,2.5,1.6,1.9*3E
- PPS is an output pin indicating you can synchronize your system clock to the GPS clock.
- Cold start requires downloading new almanac and ephemeris data.
- UAVs and other fast vehicles may require increased update rates. 5 and even 10Hz .

Lab: GPS

- The ATGM336H-5N GNSS module receives GPS signals and communicates through a UART interface.
  - Cold start - Recapture sensitivity ： -148dBm
  - Tracking sensitivity ： -162dBm
  - Positioning Precision ： 2.5m （CEP50）
  - The Time To First Fix: ： 32s

- GPS code follows. The GPS module communicates over a simple serial line implemented by SoftwareSerial in the Arduino libraries.
- The module uses the “NEMA message format” that reports UTC time, latitude, and longitude.
- A NEMA message report for Boston is (`42.361732,N,071.090595,W`) is:

```
$GNRMC,005536.000,A,4236.1732,N,07109.0595,W,0.00,
0.00,230218,,,A*68
```



Arduino code for GPS

```
// GPS
// Manferdelli
#include <SoftwareSerial.h>
#include <Adafruit_GPS.h>
#ifndef byte
typedef uint8_t byte;
#endif
const int gpsReceivePin= 3;
const int gpsTransmitPin= 5;
const int messageDelay= 500;

// Note: GPS transmit pin is SoftwareSerial receive pin and vice-versa
SoftwareSerial serialGPS = SoftwareSerial(gpsTransmitPin, gpsReceivePin);
Adafruit_GPS GPS(&serialGPS);

void setup() {
  Serial.begin(9600);
  GPS.begin(9600);
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
  GPS.sendCommand(PGCMD_ANTENNA);
}
```

Arduino code for GPS

```
void printTwoDigitInt(int x) {
  if (x < 10)
    Serial.print("0");
  Serial.print(x);
}

void printTwoDigitDouble(double x) {
  if (x < 10.0)
    Serial.print("0");
  Serial.print(x);
}

struct gpm_msg_values {
  int hour_;
  int min_;
  double seconds_;
  double degrees_lat_;
  double degrees_long_;
  int num_sats_;
};
```

```
// s: string to match,  t: message string
char* strmatch(const char* s, char* t) {
  for (;;) {
    if (*s == 0 || *t == 0)
      return NULL;
    if (*s != *t)
      return NULL;
    s++; t++;
    if (*s == 0)
      return t;
  }
}

char* find_string_in_msg(const char* str,
char* msg) {
  int l = strlen(msg);
  char* s = NULL;

  for (int i = 0; i < l; i++) {
    s = strmatch(str, &msg[i]);
    if (s != NULL)
      return s;
  }
  return NULL;
}
```

Arduino code for GPS

```
// NMEA message format
//  $GNGGA,HHMMSS.SSS,DDMM.MMMM,N/S,DDDMM.MMMM,E/W,n,NS...A*20
bool parseNMEAMessage(char* msg, struct gpm_msg_values* v) {
  char* time_string = find_string_in_msg("$GNGGA,", msg);
  if (time_string == NULL)
    return false;
  sscanf(time_string, "%02d%02d", &v->hour_, &v->min_);
  sscanf(time_string+4, "%f", &v->seconds_);

  char* latitude_string = find_string_in_msg(",", time_string);
  if (latitude_string == NULL)
    return false;
  int deglat;
  int mlat;
  int minlat;
  sscanf(latitude_string, "%02d%02d.%04d", &deglat, &mlat, &minlat);
  v->degrees_lat_ = ((double)deglat) + (((double)mlat) + ((double)minlat) / 10000.0) / 60.0;

  char* ns_string = find_string_in_msg(",", latitude_string);
  if (ns_string == NULL || (*ns_string != 'N' && *ns_string != 'S'))
    return false;
  if (*ns_string == 'S')
    v->degrees_lat_ *= -1.0;
```

Arduino code for GPS

```
char* longitude_string = ns_string + 2;
  int deglong;
  int mlong;
  int minlong;
  sscanf(longitude_string, "%03d%02d.04d", &deglong, &mlong, &minlong);
  v->degrees_long_ = ((double)deglong) + (((double)mlong) + ((double)minlong)/ 10000.0) / 60.0;
  char* ew_string = find_string_in_msg(",", longitude_string);
  if (ew_string == NULL || (*ew_string != 'E' && *ew_string != 'W'))
    return false;
  if (*ew_string == 'W')
    v->degrees_long_ *= -1.0;

  return true;
}

void loop() {
  gpm_msg_values out;
  bool got_fix = false;
```

Arduino code for GPS

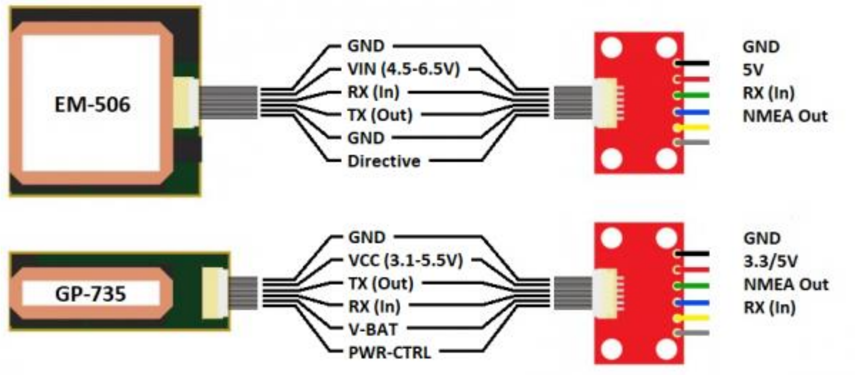
```
while (!got_fix) {
    for(;;) {
      char c = GPS.read();     // causes the message to be read
      if (GPS.newNMEAreceived())
        break;
    }
    got_fix = parseNMEAMessage(GPS.lastNMEA(), &out);
    if (got_fix) {
      Serial.print("Time: ");
      printTwoDigitInt(out.hour_);
      Serial.print(":");
      printTwoDigitInt(out.min_);
      Serial.print(":");
      printTwoDigitDouble(out.seconds_);
      Serial.print(" GMT,  ");
      Serial.print("Position: ");
      Serial.print(out.degrees_lat_);
      Serial.print(" degrees latitude, ");
      Serial.print(out.degrees_long_);
      Serial.print(" degrees longitude");
      Serial.println();
    }
  }
  delay(messageDelay);
}
```

Lab: Connect a Linux machine to a GPS sensor

- We’ll first connect a Linux machine, from the USB port on the Linux machine, to an FDTI1232 and connect the FDTI-1232 to the GPS sensor.

GPS module	FDTI1232
gnd	gnd
vcc	vcc
RX (configure)	TX
TX	RX

- You can also connect using the USB ports in the same way Raspberry Pi, well do that next.



Linux code for GPS

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <string.h>
#ifndef byte
typedef unsigned char byte;
#endif
#define uartDevice "/dev/ttyUSB0"
#define BUF_SIZE 512

void clearBuf(byte* buf, int n) {
  for (int i = 0; i < n; i++) {
    buf[i] = 0;
  }
}

const char* eol = "\r\n";
void sendCommand(int fd, const char* command) {
  write(fd, command, strlen(command));
  write(fd, eol, 2);
}
```

Linux code for GPS

```
#define PMTK_SET_NMEA_OUTPUT_RMCGGA "$PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28"
#define PMTK_SET_NMEA_UPDATE_1HZ  "$PMTK220,1000*1F"
#define PGCMD_ANTENNA "$PGCMD,33,1*6C"
void setup_gps(int fd) {
  sendCommand(fd, PMTK_SET_NMEA_OUTPUT_RMCGGA);
  sleep(1);
  sendCommand(fd, PMTK_SET_NMEA_UPDATE_1HZ);
  sleep(1);
  sendCommand(fd, PGCMD_ANTENNA);
  sleep(1);
}

// s: string to match, t: message string
char* strmatch(const char* s, char* t) {
  for (;;) {
    if (*s == 0 || *t == 0)
      return NULL;
    if (*s != *t)
      return NULL;
    s++; t++;
    if (*s == 0)
      return t;
  }
}
```

Linux code for GPS

```
char* find_string_in_msg(const char* str, char* msg) {
  int l = strlen(msg);
  char* s = NULL;

  for (int i = 0; i < l; i++) {
    s = strmatch(str, &msg[i]);
    if (s != NULL)
      return s;
  }
  return NULL;
}

// GPS value structure
struct gpm_msg_values {
  int hour_;
  int min_;
  double seconds_;
  double degrees_lat_;
  double degrees_long_;
  int num_sats_;
};

  return 0;
}
```

Linux code for GPS

```
// NMEA message format
//  $GNGGA,HHMMSS.SSS,DDMM.MMMM,N/S,DDDMM.MMMM,E/W,n,NS...A*20
bool parseNMEAMessage(char* msg, struct gpm_msg_values* v) {
  char* time_string = find_string_in_msg("$GNGGA,", msg);
  if (time_string == NULL)
    return false;
  sscanf(time_string, "%02d%02d", &v->hour_, &v->min_);
  sscanf(time_string+4, "%lf", &v->seconds_);

  char* latitude_string = find_string_in_msg(",", time_string);
  if (latitude_string == NULL)
    return false;
  int deglat;
  int mlat;
  int minlat;
  sscanf(latitude_string, "%02d%02d.%04d", &deglat, &mlat, &minlat);
  v->degrees_lat_ = ((double)deglat) + (((double)mlat) + ((double)minlat) / 10000.0) / 60.0;
  char* ns_string = find_string_in_msg(",", latitude_string);
  if (ns_string == NULL || (*ns_string != 'N' && *ns_string != 'S'))
    return false;
  if (*ns_string == 'S')
    v->degrees_lat_ *= -1.0;
  char* longitude_string = ns_string + 2;
```

Linux code for GPS

```
  int deglong;
  int mlong;
  int minlong;
  sscanf(longitude_string, "%03d%02d.%04d", &deglong, &mlong, &minlong);
  v->degrees_long_ = ((double)deglong) + (((double)mlong) + ((double)minlong)
     / 10000.0) / 60.0;
  char* ew_string = find_string_in_msg(",", longitude_string);
  if (ew_string == NULL || (*ew_string != 'E' && *ew_string != 'W'))
    return false;
  if (*ew_string == 'W')
    v->degrees_long_ *= -1.0;
  return true;
}

bool get_location(int fd) {
  gpm_msg_values out;
  bool got_fix = false;
  byte buf[BUF_SIZE];
  int n;
  int msg_count = 1;
```

Linux code for GPS

```
  while (!got_fix) {

    sleep(2);
    clearBuf(buf, BUF_SIZE);
    n = read(fd, buf, BUF_SIZE - 1);
    if (n <=  0) {
      printf("read returns %d\n", n);
      continue;
    }
    buf[n++] = 0;
    if (n > 2)
      printf("Message %d: %s", msg_count++, (char*) buf);

    got_fix = parseNMEAMessage((char*)buf, &out);
    if (got_fix) {
      printf("Time: %02d:%02d:%07.4f GMT, ", out.hour_, out.min_, out.seconds_);
      printf("Position: %8.5f (lat), %8.5f (long)\n", out.degrees_lat_, out.degrees_long_);
      return true;
    }
  }
  return false;
}
```

Linux code for GPS

```
int main(int an, char** av) {
  int baudRate = 9600;

  int fd = open(uartDevice, O_RDWR | O_NOCTTY | O_NDELAY);
  if (fd < 0) {
    printf("Can't open %s\n", uartDevice);
    return 1;
  }

  // setup gps sensor and get location
  setup_gps(fd);
  for (int i = 0; i < 10; i++) {
    get_location(fd);
    printf("\n");
  }

  close(fd);
  return 0;
}
```

Lab: RP GPS

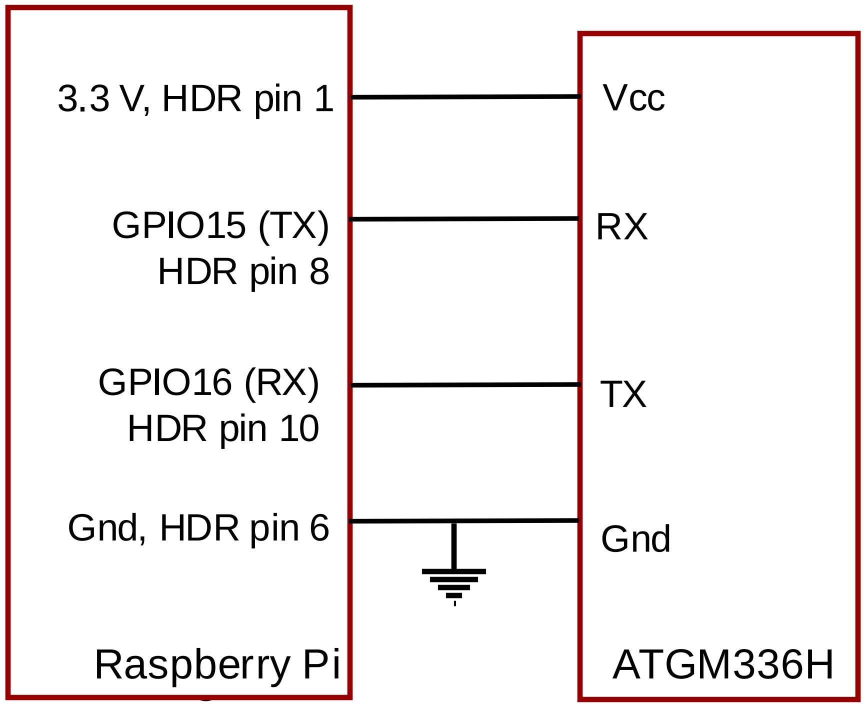
- We can also use the code above on the Raspberry Pi to connect to a GPS sensor from a USB port on the RP. If you want to use the mini-USB port, you need to use “/dev/serial0.”

Lab: UART from Raspberry Pi interface

- Finally, we can connect from the Raspberry Pi Tx (pin 8) and Rx (pin 10) pins directly to the GPS sensor.
- First, you'll have to enable the RP pins RX/TX pins by adding (or editing) a line in /boot/config.txt.
- We need to open the native RP UART device, /dev/ttyS0.
- UARTs are important. We'll use this for GPS and HC-12.

Raspberry Pi code for UART (GPS)

```
// GPS Interfacing with Raspberry Pi using C (WiringPi Library)
//  Raspberry Pi      GPS
//  3.3V  (GPIO Pin  1) VCC
//  GND (GPIO Pin  6) GND
//  TXD (GPIO Pin  8) RXD
//  RXD (GPIO Pin 10) TXD
#define uartDevice "/dev/ttyS0”
#define BUF_SIZE 512
const int messageDelay = 50;

void clearBuf(byte* buf, int n) {
  for (int i = 0; i < n; i++) {
    buf[i] = 0;
  }
}
```



Raspberry Pi code for GPS

```
const char* eol = "\r\n";
void sendCommand(int fd, const char* command) {
  for (int i = 0; i < strlen(command); i++)
    serialPutchar(fd, command[i]);
  serialPutchar(fd, eol[0]);
  serialPutchar(fd, eol[1]);
}

#define PMTK_SET_NMEA_OUTPUT_RMCGGA "$PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28"
#define PMTK_SET_NMEA_UPDATE_1HZ  "$PMTK220,1000*1F"
#define PGCMD_ANTENNA "$PGCMD,33,1*6C"
void setup_gps(int fd) {
  sendCommand(fd, PMTK_SET_NMEA_OUTPUT_RMCGGA);
  sendCommand(fd, PMTK_SET_NMEA_UPDATE_1HZ);
  sendCommand(fd, PGCMD_ANTENNA);
}
```

Raspberry Pi code for GPS

```
// s: string to match
// t: message string
char* strmatch(const char* s, char* t) {
  for (;;) {
    if (*s == 0 || *t == 0)
      return NULL;
    if (*s != *t)
      return NULL;
    s++; t++;
    if (*s == 0)
      return t;
  }
}

char* find_string_in_msg(const char* str, char* msg)
{
  int l = strlen(msg);
  char* s = NULL;

  for (int i = 0; i < l; i++) {
    s = strmatch(str, &msg[i]);
    if (s != NULL)
      return s;
  }
  return NULL;
}
```

```
// GPS value structure
struct gpm_msg_values {
  int hour_;
  int min_;
  double seconds_;
  double degrees_lat_;
  double degrees_long_;
  int num_sats_;
};
```

Raspberry Pi code for GPS

```
// NMEA message format
//  $GNGGA,HHMMSS.SSS,DDMM.MMMM,N/S,DDDMM.MMMM,E/W,n,NS...A*20
bool parseNMEAMessage(char* msg, struct gpm_msg_values* v) {
  char* time_string = find_string_in_msg("$GNGGA,", msg);
  if (time_string == NULL)
    return false;
  sscanf(time_string, "%02d%02d", &v->hour_, &v->min_);
  sscanf(time_string+4, "%f", &v->seconds_);

  char* latitude_string = find_string_in_msg(",", time_string);
  if (latitude_string == NULL)
    return false;
  int deglat;
  int mlat;
  int minlat;
  sscanf(latitude_string, "%02d%02d.%04d", &deglat, &mlat, &minlat);
  v->degrees_lat_ = ((double)deglat) + (((double)mlat) + ((double)minlat) / 10000.0) / 60.0;
  char* ns_string = find_string_in_msg(",", latitude_string);
  if (ns_string == NULL || (*ns_string != 'N' && *ns_string != 'S'))
    return false;
  if (*ns_string == 'S')
    v->degrees_lat_ *= -1.0;
  char* longitude_string = ns_string + 2;
```

Raspberry Pi code for GPS

```
  int deglong, mlong, minlong;
  sscanf(longitude_string, "%03d%02d.04d", &deglong, &mlong, &minlong);
  v->degrees_long_ = ((double)deglong) + (((double)mlong) + ((double)minlong)/ 10000.0) / 60.0;
  char* ew_string = find_string_in_msg(",", longitude_string);
  if (ew_string == NULL || (*ew_string != 'E' && *ew_string != 'W'))
    return false;
  if (*ew_string == 'W')
    v->degrees_long_ *= -1.0;
  return true;
}

int read_serial_line(int fd, byte* buf, int size) {
  int r;
  int n = 0;
  while (n < (size - 1)) {
    r = serialGetchar(fd);
    buf[n++] = r;
    if (r == '\n') {
      buf[n++] = 0;
      return n;
    }
  }
  return -1;
}
```

Raspberry Pi code for GPS

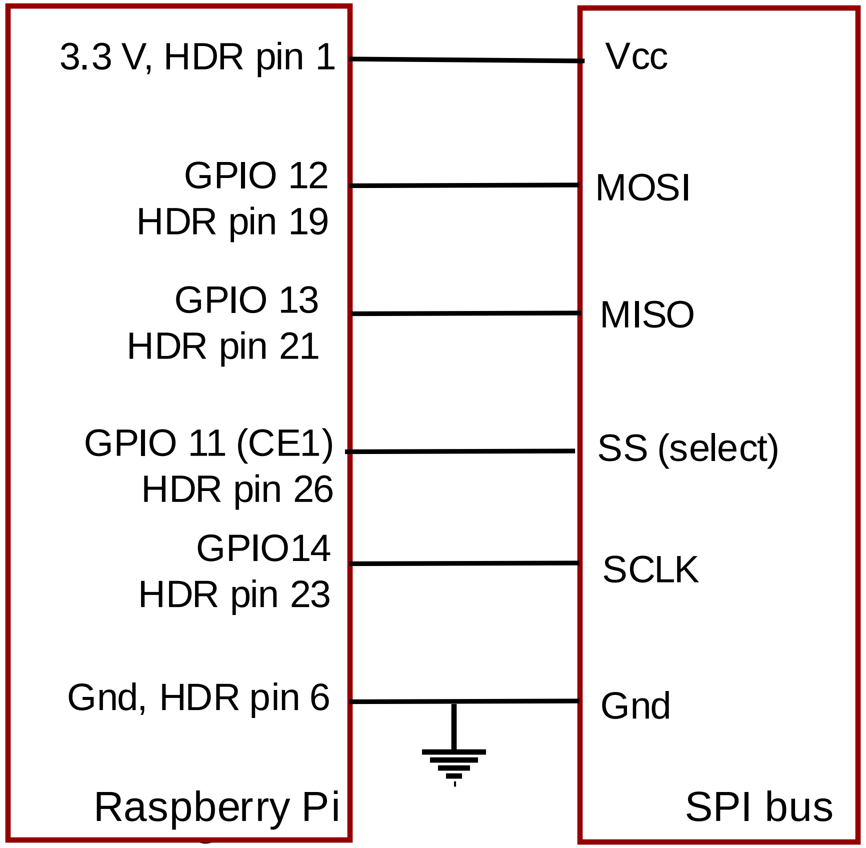
```
bool get_location(int fd) {
  gpm_msg_values out;
  bool got_fix = false;
  byte buf[BUF_SIZE];
  int n;

  while (!got_fix) {
    clearBuf(buf, BUF_SIZE);
    n = read_serial_line(fd, buf, BUF_SIZE);
    if (n <  0) {
      continue;
    }
    got_fix = parseNMEAMessage((char*)buf, &out);
    if (got_fix) {
      printf("Time: %02d:%02d:%07.4f GMT, ", out.hour_, out.min_, out.seconds_);
      printf("Position: %8.5f (lat), %8.5f (long)\n", out.degrees_lat_, out.degrees_long_);
      return true;
    }

  delay(messageDelay);
  }
  return false;
}
```

Raspberry Pi code for GPS

```
int main(int an, char** av) {
  int baudRate = 9600;

  // Initialize wiringPi
  if (wiringPiSetup() < 0) {
    printf("wiringPiSetup failed\n");
    return 0;
  }

  // Open the serial device
  int fd = serialOpen(uartDevice, baudRate);
  if (fd < 0) {
    printf("Can't open serialDevice\n");
    return 1;
  }

  // setup gps sensor and get location
  setup_gps(fd);
  get_location(fd);
  close(fd);
  return 0;
}
```

Raspberry Pi code to SPI bus

- To connect to SPI, use the following connections



Raspberry Pi code for SPI

```
//  Manferdelli
//    Raspberry Pi SPI

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <wiringPiSPI.h>
#ifndef byte
typedef unsigned char byte;
#endif
const int chip_select = 1;
const int bus_speed = 50000;
#define BUF_SIZE 128

//  RP SPI connections
//    RP      SPI
//    Gnd     Gnd
//    3.3V    Vcc
//    CE1     SS (shift select)
//    SCK     SCK
//    MOSI    SDI
//    MISO    SDO
```

Raspberry Pi code for SPI

```
int main(int an, char** av) {
  int fd = wiringPiSPISetup(chip_select, bus_speed);
  if (fd < 0) {
    printf("Can't initialize SPI interface\n");
    return 1;
  }
  int result;
  byte buf[BUF_SIZE];
  // clear display
  buf[0] = 0x76;
  result = wiringPiSPIDataRW(chip_select, buf, 1);
  if (result < 0) {
    printf("Initial clear failed\n");
  }
  sleep(5);

  // Are we addressing all segments
  for (int i = 1; i <= 0x7f; i++) {
```

Raspberry Pi code for SPI

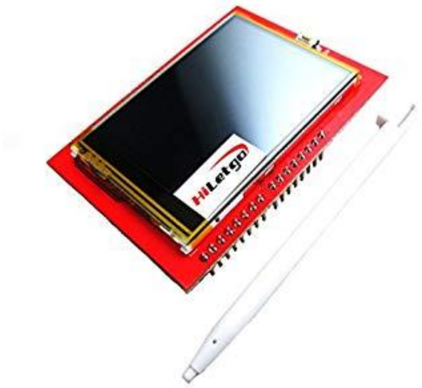
```
buf[0] = 0x77;
buf[1] = i;
result = wiringPiSPIDataRW(chip_select, buf, 2);
if (result < 0) {
  printf("First R/W failed\n");
}
buf[0] = 0x7b;
buf[1] = i;
result = wiringPiSPIDataRW(chip_select, buf, 2);
if (result < 0) {
  printf("Second R/W failed\n");
}
buf[0] = 0x7c;
buf[1] = i;
result = wiringPiSPIDataRW(chip_select, buf, 2);
if (result < 0) {
  printf("Third R/W failed\n");
}
buf[0] = 0x7d;
buf[1] = i;
result = wiringPiSPIDataRW(chip_select, buf, 2);
if (result < 0) {
  printf("Fourth R/W failed\n");
}
```

Raspberry Pi code for SPI

```
    buf[0] = 0x7e;
    buf[1] = i;
    result = wiringPiSPIDataRW(chip_select, buf, 2);
    if (result < 0) {
      printf("Fifth R/W failed\n");
    }

    sleep(5);
  }

  // clear display
  buf[0] = 0x76;
  result = wiringPiSPIDataRW(chip_select, buf, 1);
  if (result < 0) {
    printf("Final clear failed\n");
  }
  return 0;
}
```

Lab: TFT Screen

- Here we describe the HiLetgo 2.4 Inch TFT LCD Display Shield Touch Panel ILI9341 240X320
- The TFT module uses the SPI interface. Its data sheet is here, and a supporting library is here.
- Include Adafruit_GFX.h and Adafruit_TFTLCD.h libraries. Connections (use 5v for power).
  - A0 →RD
  - A1 →WR
  - A2 →RS
  - A3→CS
  - A4→RST
  - D8→D0
  - D9→D1
  - D2→ D2
  - D3→D3
  - …
  - D7→D7



Amazon

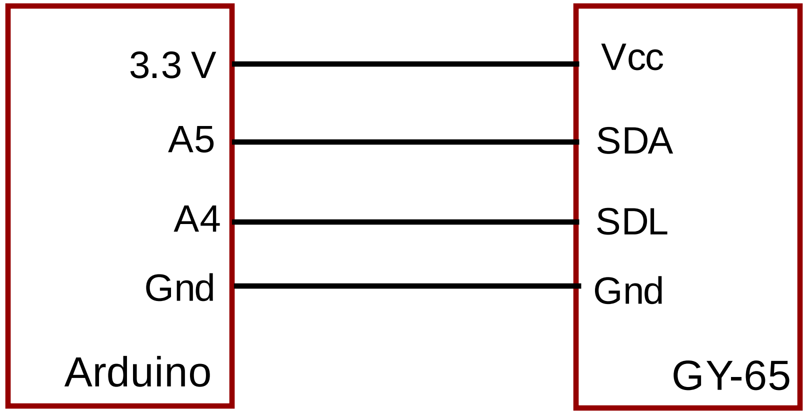
Arduino code for TFT

```
#include <Adafruit_GFX.h>
#include <Adafruit_TFTLCD.h>
#include <TouchScreen.h>
#define LCD_CS A3     // Chip Select goes to Analog 3
#define LCD_CD A2     // Command/Data goes to Analog 2
#define LCD_WR A1     // LCD Write goes to Analog 1
#define LCD_RD A0     // LCD Read goes to Analog 0
#define LCD_RESET A4
#define BLUE 0x001F
#define WHITE 0xFFFF

Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
void setup() {
  Serial.begin(9600);
  Serial.println("Starting...");
  tft.begin(0x9341);  // SDFP5408
}
```

```
void loop() {
  int k = 0;
  tft.fillScreen(WHITE);
  tft.setCursor(0, 2);
  tft.setTextColor(BLUE);
  tft.setTextSize(2);
  tft.println("hello world %d", k++);
  delay(2000);
}
```

Lab: Measure pressure

- We will use the GY-65 (or GY-68) to measure pressure, you can find the datasheet with pictures here. I've found a newer version is more accurate. See here.
- The Bosch barometer is very accurate and gives both temperature and pressure.
- You can use the pressure to calculate the altitude of the sensor. The details are explained in the datasheet. This could come in useful if you're in a plane.
- The sensor uses an I2C interface and each sensor comes with its own calibration data.
- Wire the Arduino as shown.



Arduino code for measure pressure

```
// gy-65 altitude, pressure, temp
// Manferdelli, Reference: https://www.sparkfun.com/tutorials/253

#include <Wire.h>
typedef uint8_t byte;
const int dataPin= 8;
const int measurementDelay= 500;
const int i2c_address= 0x77;
const int tempDelay= 5;
struct calibration_data {
  int16_t ac1_;
  int16_t ac2_;
  int16_t ac3_;
  uint16_t ac4_;
  uint16_t ac5_;
  uint16_t ac6_;
  int16_t b1_;
  int16_t b2_;
  int16_t mb_;
  int16_t mc_;
  int16_t md_;

  // temps
  long b5_;
};
```

Arduino code for measure pressure

```
int16_t swap_bytes(int16_t in) {
  return ((in << 8) & 0xff00) | ((in >> 8) & 0x00ff);
}
int read_i2c(byte r, byte* buf, int size) {
  Wire.beginTransmission(i2c_address);
  Wire.write(r);
  Wire.endTransmission(false);
  Wire.requestFrom(i2c_address, size, true);

  int i= 0;
  while (Wire.available() && i < size) {
    buf[i++]= Wire.read();
  }
  return i;
}
byte read_i2c_byte(byte r) {
  Wire.beginTransmission(i2c_address);
  Wire.write(r);
  Wire.endTransmission(false);
  Wire.requestFrom(i2c_address, 1, true);

  if (Wire.available()) {
    return Wire.read();
  }
}
```

Arduino code for measure pressure

```
void write_i2c(byte r, byte d) {
  Wire.beginTransmission(i2c_address);
  Wire.write(r);
  Wire.write(d);
  delay(tempDelay);
  Wire.endTransmission(true);
}

int read_i2c_int(byte address) {
  int16_t d;

  read_i2c(address, (byte*)&d, sizeof(int16_t));
  d = swap_bytes(d);
  return d;
}

const double pressure_at_sea_level= 101325.0;  // Pa
void calibrate(calibration_data* d) {
  read_i2c(0xaa, (byte*)d, sizeof(calibration_data));
  int16_t* ip = (int16_t*)d;
  for (int i = 0; i < 16; i++) {
    ip[i]= swap_bytes(ip[i]);
  }
}
```

Arduino code for measure pressure

```
double calculate_altitude(long pressure) {
  double normalizedP= ((double)pressure)/ pressure_at_sea_level;
  double a = (1.0 - pow(normalizedP, 1 / 5.255)) * 44330.0;
  return a;
}

const long OSS = 1;

long calculate_real_temperature(calibration_data& cd, long raw) {
  long x1 = ((raw - ((long)cd.ac6_)) * ((long)cd.ac5_)) >> 15;
  long x2 = (((long)cd.mc_) << 11) / (x1 + ((long)cd.md_));
  cd.b5_ = x1 + x2;
  long t = (cd.b5_ + 8) >> 4;
  Serial.println("");
  Serial.println("Real temperature");
  Serial.print("x1: ");
  Serial.println(x1);
  Serial.print("x2: ");
  Serial.println(x2);
  Serial.print("b5: ");
  Serial.println(cd.b5_);
  return t / 10;
}
```

Arduino code for measure pressure

```
long calculate_real_pressure(calibration_data& cd, long raw) {
  long p;

  long b6 = cd.b5_ - 4000;
  long x1 = (((long)cd.b2_) * ((b6 * b6) >> 12)) >> 11;
  long x2 = (((long)cd.ac2_) * b6) >> 11;
  long x3 = x1 + x2;
  long b3 = (((((long)cd.ac1_) * 4 + x3) << OSS) + 2) / 4;

  x1= (((long)cd.ac3_) * b6) >> 13;
  x2 = (((long)cd.b1_) * ((b6 * b6) >> 12)) >> 16;
  x3 = (x1 + x2 + 2) / 4;
  long b4 = (((long)cd.ac4_) * (x3 + 32768)) >> 15;
  long b7 = (raw - b3) * (50000 >> OSS);
  if (b7 < 0x80000000UL)
    p = (b7 * 2) / b4;
  else
    p = (b7 / b4) * 2;
  x1 = (p >> 8) * (p >> 8);
  x1 = (x1 * 3038) >> 16;
  x2 = (-7357 * p) >> 16;
  p += (x1 + x2 + 3791) >> 4;
  return p;
}
```

Arduino code for measure pressure

```
long raw_temperature() {
  write_i2c(0xf4, 0x2e);
  delay(tempDelay);
  long t = read_i2c_int(0xf6);
  return t;
}

long raw_pressure() {
  write_i2c(0xf4, 0x2e);
  delay(tempDelay);
  byte msb= read_i2c_byte(0xf6);
  byte lsb1= read_i2c_byte(0xf7);
  byte lsb2= read_i2c_byte(0xf8);
  long rp = ((((long)msb) << 16) + (((long)lsb1) << 8) + ((long)lsb2)) >> (8 - OSS);
  return rp;
}

long get_temperature(calibration_data& cd) {
  long rt= raw_temperature();
  Serial.print("Raw temperature: ");
  Serial.print(rt);
  Serial.println("");
  return calculate_real_temperature(cd, rt);
}
```

Arduino code for measure pressure

```
long raw_temperature() {
  write_i2c(0xf4, 0x2e);
  delay(tempDelay);
  long t = read_i2c_int(0xf6);
  return t;
}

long raw_pressure() {
  write_i2c(0xf4, 0x2e);
  delay(tempDelay);
  byte msb= read_i2c_byte(0xf6);
  byte lsb1= read_i2c_byte(0xf7);
  byte lsb2= read_i2c_byte(0xf8);
  long rp = ((((long)msb) << 16) + (((long)lsb1) << 8) + ((long)lsb2)) >> (8 - OSS);
  return rp;
}

long get_temperature(calibration_data& cd) {
  long rt= raw_temperature();
  Serial.print("Raw temperature: ");
  Serial.print(rt);
  Serial.println("");
  return calculate_real_temperature(cd, rt);
}
```

Arduino code for measure pressure

```
long get_pressure(calibration_data& cd) {
  long rp = raw_pressure();
  long p = calculate_real_pressure(cd, rp);
  return p;
}

void setup() {
  Serial.begin(9600);
  Wire.begin();
}
```

Arduino code for measure pressure

```
void loop() {
  calibration_data cd;
  calibrate(&cd);
  printCalibration(cd);

  double temperature= (double)get_temperature(cd);
  double pressure= (double)get_pressure(cd);
  double altitude= calculate_altitude(pressure);

  Serial.print("Temperature (C): ");
  Serial.print(temperature);
  Serial.print(", pressure (Pa): ");
  Serial.print(pressure);
  Serial.print(", altitude (m): ");
  Serial.print(altitude);
  Serial.println("");
  delay(measurementDelay);
}
```

Raspberry Pi code for measure pressure (bmp280)

- You can download a support library for the Bosch BMP-280 but I had trouble getting it to work. For one thing, when I ordered one, I got a different chip, the “BME-280,” which is electrically identical but has a humidity sensor. It has a chip id the library doesn’t recognize. Some BMP-280’s are labeled BME-280’s.
- I ended up writing my own interface library which is included in the Lab directories.
- The connections from the Raspberry Pi are the usual SDA, SCL pins (HDR 3, 5). Vcc should be 3.3 volts, not 5v.

Bmp-280 output

chip id: 60

Calibration data:

t1: 6f11, t2: 68c9, t3: 32

p1: 9322, p2: ffffd694, p3: bd0

p4: 15f9, p5: 32, p6: fffffff9, p7: 0x02
fine: 00
get CNTRL: 00, MEAS: 00
Config data before init:
standby: 00, mode: 00filter: 00, odr: 08
samp_temp: 00, samp_press: 00
Config data after init
standby: 04, mode: 03
filter: 03, odr: 05
samp_temp: 04, samp_press: 04
set CNTRL: 93, MEAS: 8c
get CNTRL: 93, MEAS: 8c
Configuration data after set:
standby: 04, mode: 03
filter: 04, odr: 05
samp_temp: 04, samp_press: 04

Sea level pressure: 1018.0000 (Pa)

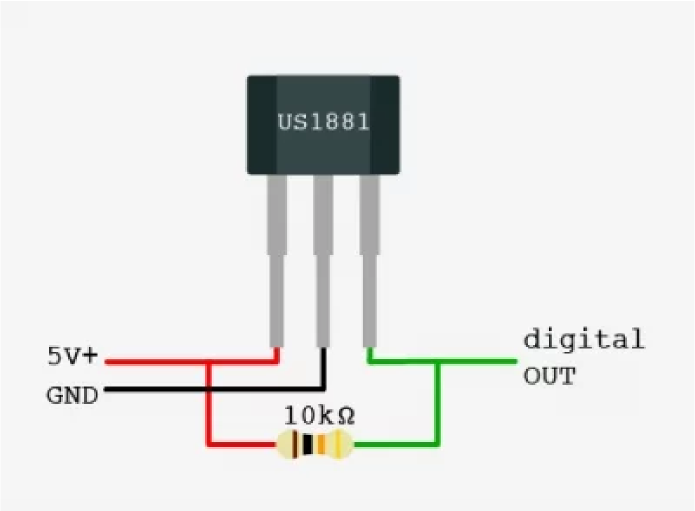
uncompensated temperature reading: 523358, 0x7fc5e
uncompensated pressure reading: 344664, 0x54258
Temperature: 21.89 (C), pressure: 1015.3782 (Pa),
alt: 21.7491 (m)

uncompensated temperature reading: 523374, 0x7fc6e
uncompensated pressure reading: 344662, 0x54256
Temperature: 21.89 (C), pressure: 1015.3846 (Pa),
alt: 21.6958 (m)

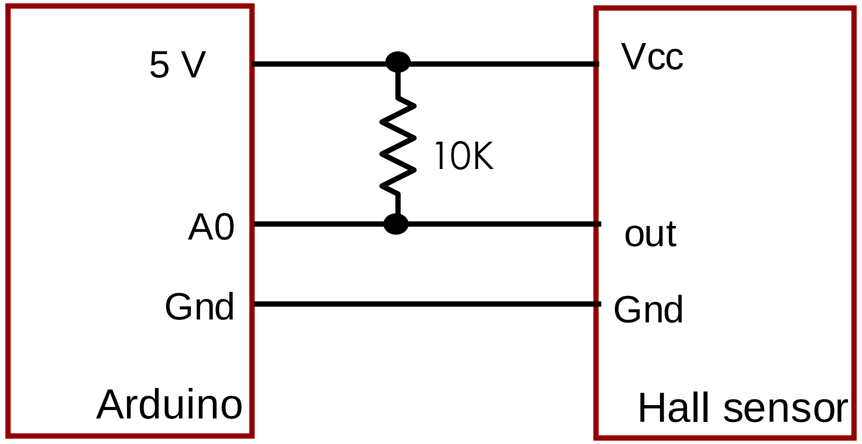
uncompensated temperature reading: 523387, 0x7fc7b
uncompensated pressure reading: 344663, 0x54257
Temperature: 21.90 (C), pressure: 1015.3782 (Pa),
alt: 21.7491 (m)

Lab: Hall Sensor

- A Hall sensor measures magnetic fields. In the configuration below, when a magnetic field of the right polarity is present, the out terminal will go to ground.
- You can find a tutorial and pictures here.
- Wire the Arduino as shown. Code follows.



Maker



Arduino code for Hall sensor

```
// Hall sensor
// Manferdelli

typedef uint8_t byte;
const int dataPin= A0;
const int measurementDelay= 500;

void setup() {
  Serial.begin(9600);
  pinMode(dataPin, INPUT);
}

const int zero_field= 527;
void loop() {
  int field_strength= analogRead(dataPin);
  Serial.print("Field: ");
  Serial.print(field_strength);
  int calibrated_field = field_strength - zero_field;
  Serial.print(", calibrated field: ");
  Serial.print(calibrated_field);
  Serial.println("");
  delay(measurementDelay);
}
```

Raspberry Pi code for Hall sensor

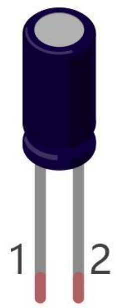
- Use PCF8591 ADC and I2C interface described earlier.

```
// Manferdelli
//   Raspberry Pi read hall sensor

#include <stdio.h>
#include <wiringPi.h>
#include <pcf8591.h>

const int base = 120;
const int i2c_address = 0x48;
// const int zero_field = 527;
const int zero_field = 7;  //this is better for 3.3v
const int measurement_delay = 500;

// PCF Connections
//    PCF         RP
//    1(ain0)     analog-signal
//    5, 6, 7, 8  gnd (pin 6)
//    16          3.3 (pin 1)
//    14          3.3v
//    13          gnd
//    12          gnd
//    10(scl)     5 (scl)
//    9(sda)      3 (sda)
```

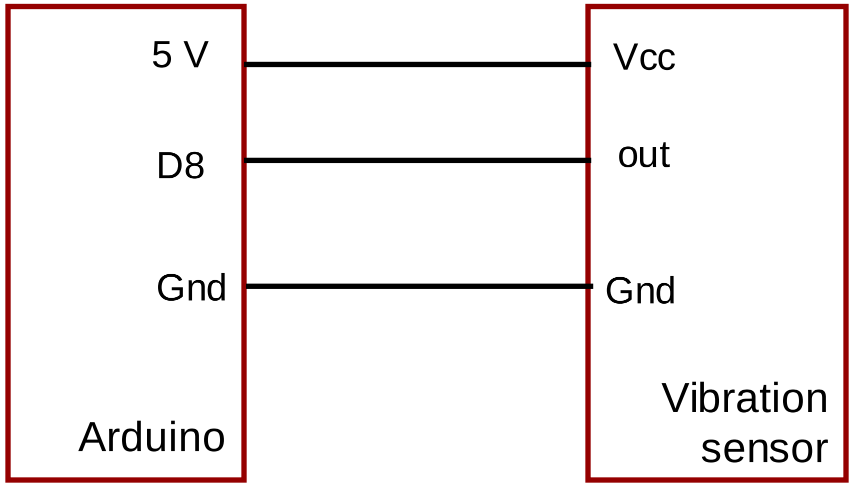
```
int main(int an, char** av) {
  int strength = 0;

  if (wiringPiSetup() < 0) {
    printf("Can't initialize wiringPi\n");
    return 1;
  }
  pcf8591Setup(base, i2c_address);

  for(;;) {
    strength = analogRead(base);
    int calibrated_strength = strength - zero_field;
    printf("Strength: %d, calibrated measurement: %d\n",
            strength, calibrated_strength);
    delay(measurement_delay);
  }
  return 0;
}
```

Lab: Vibration sensor

- You can find pictures of the vibration sensor and a tutorial here. The vibration sensor is a switch that is normally off, vibration causes it to conduct.
- Wire the vibration sensor and the Arduino as shown.



Freenove

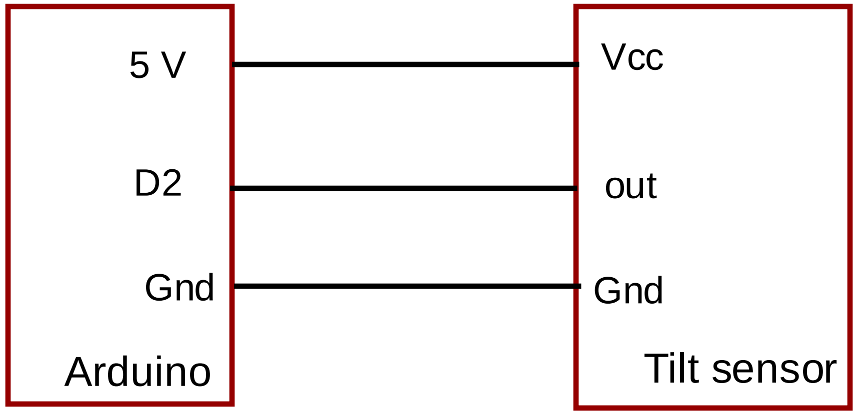


Arduino code for vibration

```
// Vibration
// Manferdelli

typedef uint8_t byte;
const int dataPin= 2;
const int measurementDelay= 20;

volatile int state = -1;
void trigger() {
  state= 1;
}

void setup() {
  Serial.begin(9600);
  attachInterrupt(dataPin, trigger, RISING);
}

void loop() {
  if (state == 1) {
    Serial.println("Vibration detected");
    delay(1);
    state= 0;
  }
  delay(measurementDelay);
}
```

Lab: Tilt sensor

- You can find pictures of the tilt sensor and a tutorial here. The resistance of the tilt sensor increases with the angle of inclination from level. You can test this with a meter.
- Wire the tilt sensor and the Arduino as shown.



Arduino code for Tilt sensor

```
// Tilt
// Manferdelli

typedef uint8_t byte;
const int dataPin= 8;
const int measurementDelay= 500;

void setup() {
  Serial.begin(9600);
  pinMode(dataPin, INPUT);
  digitalWrite(dataPin, HIGH);  // Why?
}

void loop() {
  int tilted= digitalRead(dataPin);
  Serial.println("");
  if (tilted == 0) {
    Serial.println("Tilted");
  } else {
    Serial.println("Not tilted");
  }
  delay(measurementDelay);
}
```

Lab: LCD-1602 display

- You can find pictures of the display and a tutorial in the Freenove manual here. We wire it directly but you can also buy LCD-1602 modules with I^{2}C interfaces which saves lots of pins.
- The LCD 1602 display wiring is complicated, so we don't do it graphically.
  - LCD RS pin to digital pin 12
  - LCD Enable pin to digital pin 11
  - LCD D4 pin to digital pin 5
  - LCD D5 pin to digital pin 4
  - LCD D6 pin to digital pin 3
  - LCD D7 pin to digital pin 2
  - LCD R/W pin to ground
  - LCD VSS pin to ground
  - LCD VCC pin to 5V
  - 10K resistor: ends to +5V and ground
  - wiper to LCD VO pin (pin 3)

Arduino code for LCD display

```
// LCD display
// Manferdelli
#include <LiquidCrystal.h>

typedef uint8_t byte;

//  16x2 LCD display.  The LiquidCrystal library works with all LCD displays
//  that are compatible with the Hitachi HD44780 driver.
// LCD RS pin to digital pin 12
// LCD Enable pin to digital pin 11
// LCD D4 pin to digital pin 5
// LCD D5 pin to digital pin 4
// LCD D6 pin to digital pin 3
// LCD D7 pin to digital pin 2
// LCD R/W pin to ground
// LCD VSS pin to ground
// LCD VCC pin to 5V
// 10K resistor: ends to +5V and ground
// wiper to LCD VO pin (pin 3)
// See: http://www.arduino.cc/en/Tutorial/LiquidCrystalHelloWorld

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

Arduino code for LCD display

```
void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
  lcd.clear();
  lcd.leftToRight();
  lcd.setCursor(0, 1);
  lcd.write("Hello, John");
  lcd.setCursor(0, 0);
  lcd.write("0123456789abcdef");

}

void loop() {
  // set the cursor to column 0, line 1 line 1 is the second row
  lcd.setCursor(0, 1);
  lcd.print(millis() / 1000);
}
```

Lab: RP LCD-1602 display

- We connect the Raspberry Pi to the 1602 via a I2C connection (so your 1602 needs an i2c interface)
- The connection from the RP use the same SDA, SCL connections (HDR 3, 5).
- The 1602 requires a 5-volt voltage source.

Raspberry Pi code for LCD display (using I2C)

```
//  Manferdelli
//    Raspberry Pi, i2c-lcd

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>

#ifndef byte
typedef unsigned char byte;
#endif

const byte lcd_i2c_address = 0x27;

//  Connection
//      RP           I2c IF
//      gnd(HDR 6)   Gnd
//      scl(HDR 5)   scl
//      5v (HDR 1)   Vcc
//      sda(HDR 3)   sda
```

Raspberry Pi code for LCD display (using I2C)

```
void write_int(int fd, int data) {
  data |= 0x08;
  wiringPiI2CWrite(fd, data);
  return;
}

int read_int(int fd) {
  return wiringPiI2CRead(fd);
}

void send_command(int fd, int c) {
  int t;

  // RS=0, RW=0, EN=1
  t = (c & 0xf0) | 0x04;
  write_int(fd, t);
  delay(2);
  // EN = 0
  t &= 0xfb;
  write_int(fd, t);
```

```
// RS=0, RW=0, EN=1
  t = ((c & 0x0f) << 4) | 0x04;
  write_int(fd, t);
  delay(2);
  // EN = 0
  t &= 0xfb;
  write_int(fd, t);
}

void send_data(int fd, int d) {
  int t;

  // RS=1, RW=0, EN=1
  t = (d & 0xf0) | 0x05;
  write_int(fd, t);
  delay(2);
  // EN = 0
  t &= 0xfb;
  write_int(fd, t);
```

Raspberry Pi code for LCD display (using I2C)

```
  // RS=1, RW=0, EN=1
  t = ((d & 0x0f) << 4) | 0x05;
  write_int(fd, t);
  delay(2);
  // EN = 0
  t &= 0xfb;
  write_int(fd, t);
}

bool lcd_init(int fd) {
  // 8 line
  send_command(fd, 0x33);
  delay(5);
  // 4 line
  send_command(fd, 0x32);
  delay(5);
  // 2 line
  send_command(fd, 0x28);
  // display enable
  delay(5);
  send_command(fd, 0x0c);
  send_command(fd, 0x01); //clear
  delay(5);
  wiringPiI2CWrite(fd, 0x08);
  return true;
}
```

```
void write_line(int fd, int line , int position,
const char* data) {
  if (line < 0 || line > 1)
    return;
  if (position < 0 || line > 15)
    return;
  int len = strlen(data);
  if (len > 15)
    return;

  int addr = 0x40 * line + 0x80 + position;
  printf("write_line %d %d, %s\n", line,
position, data);
  send_command(fd, addr);
  for (int i = 0; i < len; i++) {
    send_data(fd, data[i]);
  }
}

void clearBuf(byte* buf, int n) {
  for (int i = 0; i < n; i++) {
    buf[i] = 0;
  }
}
```

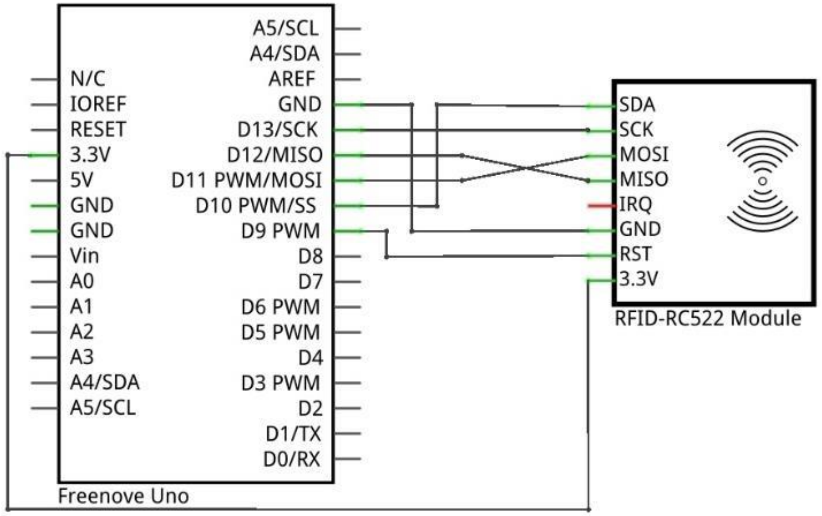
Raspberry Pi code for LCD display (using I2C)

```
int main(int an, char** av) {
  int fd =   wiringPiI2CSetup(lcd_i2c_address);
  if (fd < 0) {
    printf("Can't initialize Wiring Pi\n");
    return 1;
  }
  if (!lcd_init(fd)) {
    printf("Can't initialize LCD\n");
    return 1;
  }

  char buf[16];
  for(int j = 0; j < 20; j++) {
    send_command(fd, 0x01);
    delay(5);
    write_line(fd, 0, 0, "Line 0");
    clearBuf((byte*)buf, 16);
    sprintf(buf, "Run %d", j);
    write_line(fd, 1, 1, buf);
    delay(5000);
  }

  return 0;
}
```

Lab: RFID

- You can find pictures of the RFID sensor and a tutorial in the Freenove manual here. The RC522 rfid card uses an SPI interface. You'll also need a Mifare1 S50 card or FOB.
- You'll need to download the "RFID Library" for Arduino.
- Wire the Arduino as shown.

Newegg



Freenove

Arduino code for RFID

```
// rfid
// Manferdelli

#include <SPI.h>
#include <MFRC522.h>

typedef uint8_t byte;
const int rstPin = 9;
const int ssPin = 10;
const int measurementDelay= 500;

// Class args: slave select, reset
MFRC522 rfid(ssPin, rstPin);

void setup() {
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
}
```

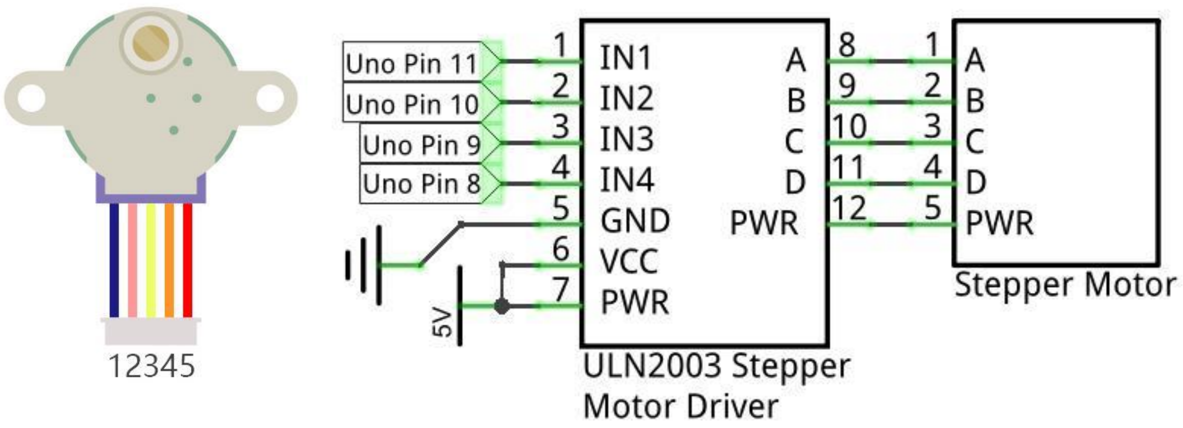
```
bool access_valid(int size, byte* id) {
  // our card has id: FA-7C-63-2E
  if (size != 4)
    return false;
  if ( id[0] != 0xfa || id[1] != 0x7c || id[2]
!= 0x63 || id[3] != 0x2e)
    return false;
  return true;
}
```

Arduino code for RFID

```
void loop() {
  if (rfid.PICC_IsNewCardPresent()) {
    if(rfid.PICC_ReadCardSerial()) {
      Serial.print("Card type: ");
      for (int i = 0; i < rfid.uid.size; i++) {
        if (i != 0)
          Serial.print("-");
        if (rfid.uid.uidByte[i] <= 0x9) {
          Serial.print("0");
        }
        Serial.print(rfid.uid.uidByte[i], HEX);
      }
      Serial.println("");
      if (access_valid(rfid.uid.size, rfid.uid.uidByte)) {
        Serial.println("Access granted");
      } else {
        Serial.println("Access denied");
      }
    } else {
    Serial.println("Can't read card serial number");
    }
  } else {
    Serial.println("Card not present");
  }
  delay(measurementDelay);
}
```

Lab: Stepper

- Stepper motors advance one step in response to a signal, so turning continuously requires a sequence of pulses in a phased order.
- Rather than timing the pulses with the Arduino, we use an IC that maps pulses to stepper inputs and drives more current than the Arduino can.
- Wire the Arduino with the stepper controller as shown.



Freenove

Arduino code for Stepper

```
// Stepper
// Manferdelli

typedef uint8_t byte;
const int measurementDelay= 1000;

int out_ports[4] = {
  11, 10, 9, 8
};

void setup() {
  Serial.begin(9600);
  for (int i = 0; i < 4; i++)
    pinMode(out_ports[i], OUTPUT);
}

const int FW = 0;
const int BW = 1;
```

Arduino code for Stepper

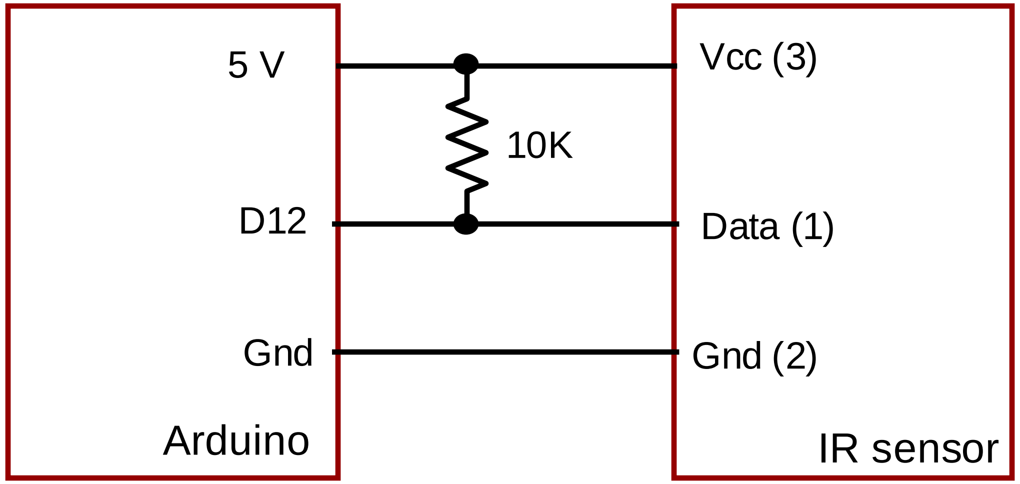
```
void move_one_step(int dir) {
  static byte out = 1;
  if (dir == FW) {
    if (out != 0x08)
      out = out << 1;
    else
      out = 1;
  } else {
    if (out != 1)
      out = out >> 1;
    else
      out = 0x08;
  }
  for (int i = 0; i < 4; i++) {
    digitalWrite(out_ports[i], (out & (1<<i)) ? HIGH:LOW);
  }
}
```

Arduino code for Stepper

```
void move_steps(int steps, byte ms_delay) {
  if (steps > 0) {
    for (int i = 0; i < steps; i++) {
      move_one_step(FW);
      delay(ms_delay);
    }
  } else {
    for (int i = 0; i < -steps; i++) {
      move_one_step(BW);
      delay(ms_delay);
    }
  }
}

void loop() {
  move_steps(2048, 4);
  delay(measurementDelay);
  move_steps(-2048, 4);
  delay(measurementDelay);
}
```

Lab: infrared sensor

- You can find a tutorial in the Freenove manual here. The IR sensor detects an input infrared light wave modulated to blink at about 38 kHz (So constant IP sources are ignored). The output (on data pin 1) is a pulse width modulated signal.
- Wire the infrared sensor and Arduino as shown.



IR encoding

- Uses pulse distance encoding for bits
  - 0: 562.5µs pulse burst followed by a 562.5µs space
  - 1: 562.5µs pulse burst followed by a 1.6875ms space
- IR message consists of:
  - a 9ms leading pulse burst
  - a 4.5ms space
  - the 8-bit address of receiving device followed by its 8-bit logical
  - the 8-bit command followed by its 8-bit logical inverse
  - a final 562.5µs pulse burst.

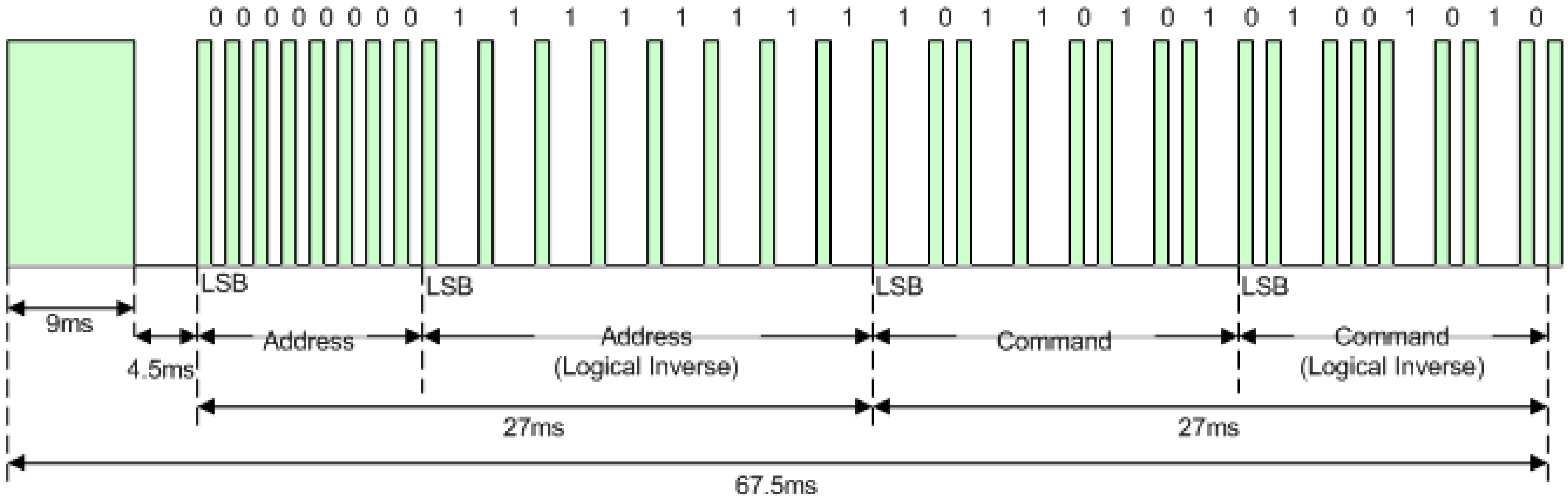


Figure from Altium

Arduino code for infrared sensor

```
// Infrared
// Manferdelli
#include <IRremote.h>

typedef uint8_t byte;
const int dataPin= 12;
const int measurementDelay= 200;
IRrecv receiver(dataPin);

void setup() {
  Serial.begin(9600);
  receiver.enableIRIn();
}

void loop() {
  decode_results results;

  if (receiver.decode(&results)) {
    Serial.print(results.value, HEX);
    Serial.println("");
    receiver.resume();
  }
  delay(measurementDelay);
}
```

Infrared – Raspberry pi

- You can use a library, lirc, to interface the raspberry pi.
- The code is unexceptional, and I’ve included an untested version on the following page.

Raspberry Pi code for infrared sensor

```
//  Manferdelli
//    Raspberry Pi, infrared

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <wiringPi.h>
#include <softPwm.h>
#include <lirc/lirc_client.h>
#include <time.h>

#ifndef byte
typedef unsigned char byte;
#endif

//  Connection scheme
//    Same as arduino
//    sig goes to gpio 4 (pin 7)

const int sig_pin = 7;
```

```
const char* key_map[21] = {
  " KEY_CHANNELDOWN ",
  " KEY_CHANNEL ",
  " KEY_CHANNELUP ",
  " KEY_PREVIOUS ",
  " KEY_NEXT ",
  " KEY_PLAYPAUSE ",
  " KEY_VOLUMEDOWN ",
  " KEY_VOLUMEUP ",
  " KEY_EQUAL ",
  " KEY_NUMERIC_0 ",
  " BTN_0 ",
  " BTN_1 ",
  " KEY_NUMERIC_1 ",
  " KEY_NUMERIC_2 ",
  " KEY_NUMERIC_3 ",
  " KEY_NUMERIC_4 ",
  " KEY_NUMERIC_5 ",
  " KEY_NUMERIC_6 ",
  " KEY_NUMERIC_7 ",
  " KEY_NUMERIC_8 ",
  " KEY_NUMERIC_9 "
};
```

Not tested yet

Raspberry Pi code for infrared sensor

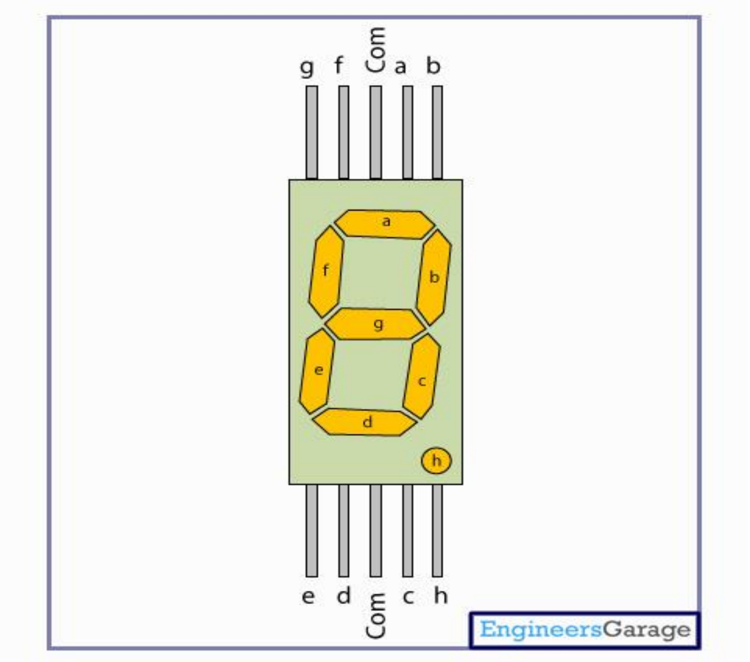
```
int main(int an, char** av) {
  if (wiringPiSetup() < 0) {
    printf("Can't initialize Wiring Pi\n");
    return 1;
  }
  linc_config *config;
  char* code;
  int button_timer = millis();

  if (lirc_init("lirc", 1) < 0) {
    printf("Can't initialize ir library\n");
    return 1;
  }
  if (lirc_readconfig(NULL, &config, NULL) == 0) {
    while (lirc_nextcode(&code) == 0) {
      if (code == NULL)
        continue;
      if ((millis() - button_timer) > 400) {
      }
      printf("Code: %s\n", code);
      free(code);
    }
    lirc_freeconfig(config);
  }
```

```
lirc_deinit();
  return 0;
}
```

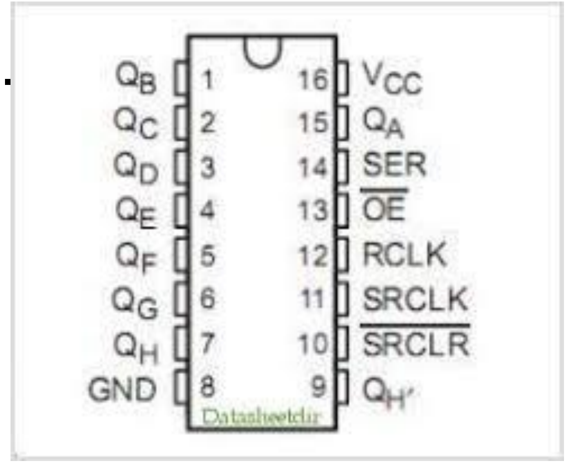
Lab: seven segment display

- You can find a tutorial about the seven-segment display in the Freenove manual here.
- There are two types, one has COM connected to anodes, one to diodes.
- The SN74hc595 is a serial to parallel shift register which conserves GPIO pins.
- Wire the Arduino as shown.

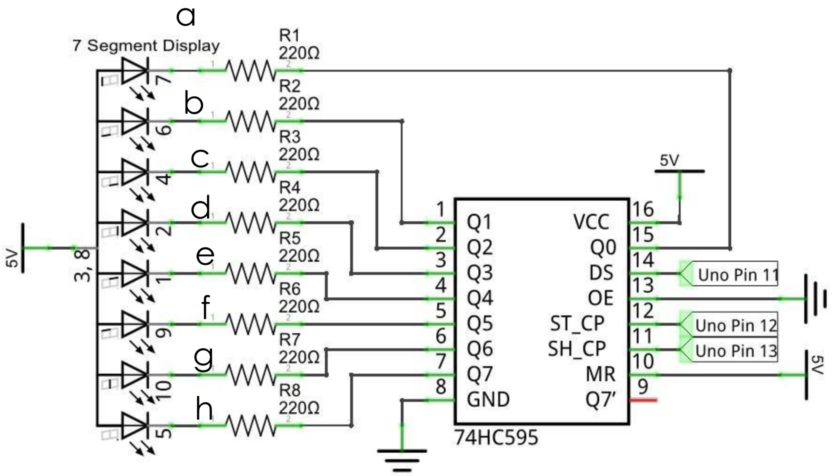
Freenove



Engineer's garage



SN54hc595



Freenove

Arduino code for display

```
// Seven segment display
// Manferdelli

#ifndef byte
typedef uint8_t byte;
#endif

int latchPin = 12;
int clockPin = 13;
int dataPin = 11;
const int measurementDelay= 1500;

// Pin11 to SH_CP 74HC595
// Pin12 connected to ST_CP 74HC595,
// Pin13 connected to DS of 74HC595
```

Arduino code for display

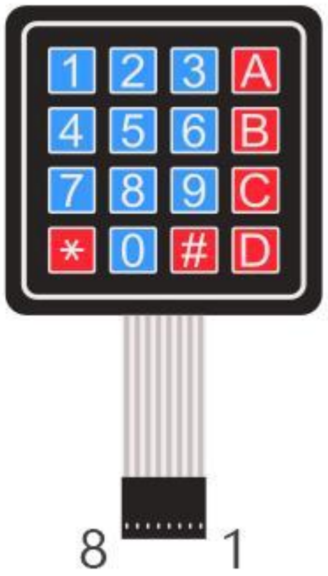
```
// encoding of digits
//     a b c d e f g h
//   0 1 1 1 1 1 1 0 0   0xfc
//   1 0 1 1 0 0 0 0 0   0x60
//   2 1 1 0 1 1 0 1 0   0xda
//   3 1 1 1 1 0 0 1 0   0xf2
//   4 0 1 1 0 0 1 1 0   0x66
//   5 1 0 1 1 0 1 1 0   0xb6
//   6 1 0 1 1 1 1 1 0   0xbe
//   7 1 1 1 0 0 0 0 0   0xe0
//   8 1 1 1 1 1 1 1 0   0xfe
//   9 1 1 1 0 0 1 1 0   0xe6
//   a 1 1 1 1 1 0 1 0   0xfa
//   b 0 0 1 1 1 1 1 0   0x3e
//   c 0 0 0 1 1 0 1 0   0x1a
//   d 0 1 1 1 1 0 1 0   0x7a
//   e 1 0 0 1 1 1 1 0   0x9e
//   f 1 1 0 0 1 1 1 0   0x8e
byte digit_pattern_seg_on[] = {
  0xfc, 0x60, 0xda, 0xf2,
  0x66, 0xb6, 0xbe, 0xe0,
  0xfe, 0xe6, 0xfa, 0x3e,
  0x1a, 0x7a, 0x9e, 0x8e,
};
```

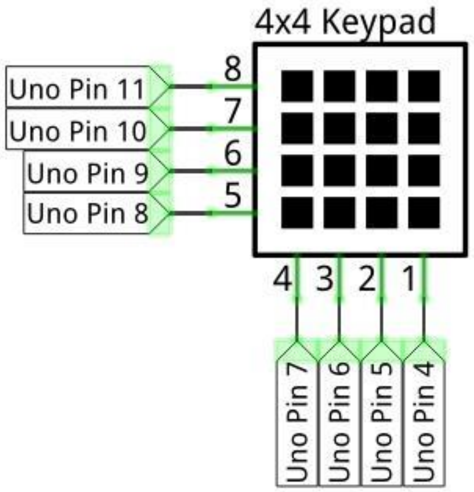
Arduino code for display

```
byte digit_pattern_seg_off[] = {
  0x03, 0x9f, 0x25, 0x0d,
  0x99, 0x49, 0x41, 0x1f,
  0x01, 0x19, 0x05, 0xc1,
  0xe4, 0x85, 0x61, 0x71,
};

void setup() {
  Serial.begin(9600);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}
void writeData(int value) {
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, value);
  digitalWrite(latchPin, HIGH);
}
void loop() {
  for (int j = 0; j < 16; j++) {
    writeData(digit_pattern_seg_off[j]);
    delay(measurementDelay);
  }
}
```

Lab: Keypad

- You can find a tutorial about the keypad in the Freenove manual here.
- Wire the Arduino as shown.





Arduino code for keypad

```
// Keypad
// Manferdelli
#include <Keypad.h>

const int dataPin= 8;
const int measurementDelay= 500;

typedef uint8_t byte;

byte rowPins[4] = {11, 10, 9, 8};
byte colPins[4] = {7, 6, 5, 4};
Keypad myKeypad = Keypad(makeKeymap(keys), rowPins, colPins, 4, 4);

// define the symbols on the buttons of the keypad
char keys[4][4] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
```

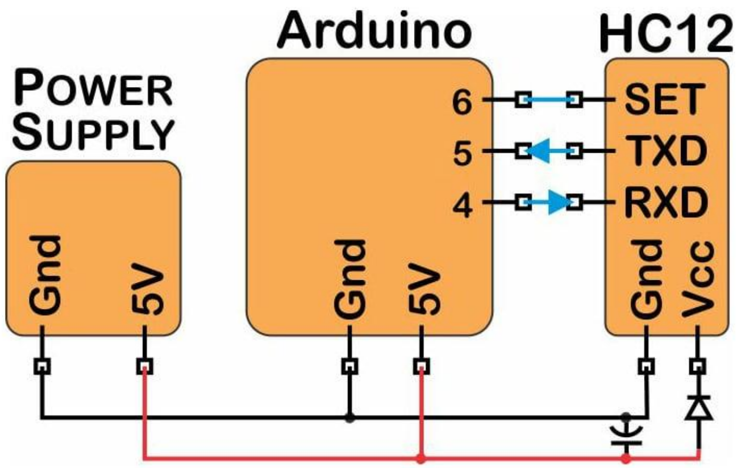
Arduino code for keypad

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  // Get the character input
  char keyPressed = myKeypad.getKey();
  // If there is a character input, sent it to the serial port
  if (keyPressed) {
    Serial.println(keyPressed);
  }
}
```

Lab: HC-12

- The HC-12 is a half-duplex 20 dBm (100 mW) transceiver with -117 dBm (2×10^{-15} W) sensitivity at 5000 bps. It uses an open spectrum 433-473MHz which covers 100 channels and has a range of about a 1.8 kilometers.
- You can find an HC-12 tutorial here.
- It uses the STM8S003F3 microcontroller and the Si4463 transceiver.
- The interface is a standard UART, serial line.
- Power 3.3-5V
- Wire the Arduino as indicated, the capacitor is a 22 µF to 1 mF reservoir capacitor.

From All about circuits



Lab: HC-12

- Module parameters, like channel selection and baud rate, are changed with the “AT” command.
- The HC-12 only responds to parameter changes when it’s in “command mode.” Command mode is enabled when the “set” pin on the HC-12 is set LOW. The other mode, transparent mode, is the usual mode, it is enabled when the “set” pin on the HC-12 is set HIGH.
- If we send “AT” to the hc-12, it returns “OK”. (This is a test message,)
- If we send AT+Bxxxxx, where xxxxx is the baud rate, the hc-12 changes baud rate. The allowable baud rates are 1200 bps, 2400 bps, 4800 bps, 9600 bps, 19200 bps, 38400 bps, 57600 bps, and 115200 bps. The default is 9600 bps. For example, if we send “AT+B38400” to the hc-12, it returns “OK+B19200”.
- AT+Cxxx change wireless communication channel, where 001 <= xxx <= 100. The default is channel 1. For example, if we send “AT+C006” command to the hc-12, it will return “OK+C006”. Each channel is 400kHz higher than the previous one.

HC-12 propagation

- Frequency band is from 433.4 MHz to 473.0 MHz
- It has a total of 100 channels with a stepping of 400 KHz between each channel
- Transmitting power is from -1dBm (0.79mW) to 20dBm (100mW)
- Receiving sensitivity is from -117dBm (0.019pW) to -100dBm (10pW).

Receiver Sensitivity	Over-the-Air Baud Rate	Serial Port Baud Rate
-117 dBm	5000 bps	1200 bps
-117 dBm	5000 bps	2400 bps
-112 dBm	15000 bps	4800 bps
-112 dBm	15000 bps	9600 bps
-107 dBm	58000 bps	19200 bps
-107 dBm	58000 bps	38400 bps
-100 dBm	236000 bps	57600 bps
-100 dBm	236000 bps	115200 bps

Arduino code for HC-12

```
// hc12
// Manferdelli
#include <SoftwareSerial.h>

const int deviceReceivePin= 4;
const int deviceTransmitPin= 5;
const int deviceSetPin = 6;

#ifndef byte
typedef uint8_t byte;
#endif

// Note: device transmit pin is SoftwareSerial receive pin
//   and vice-versa.
SoftwareSerial hc12(deviceTransmitPin, deviceReceivePin);

void copy(char* from, char* to, int size) {
  for(int i = 0; i < size; i++)
    to[i] = from[i];
  return;
}
```

Arduino code for HC-12

```
int read_from_serial(int max, char* b) {
  int i = 0;
  while (hc12.available() != 0 && (i < max)) {
    b[i++] = hc12.read();
  }
  return i;
}

void setup() {
  pinMode(deviceSetPin, OUTPUT);
  delay(100);
  Serial.begin(9600);
  hc12.begin(9600);
}
```

Arduino code for HC-12

```
void loop() {
  char send_buf[65];
  char receive_buf[65];

  copy((char*)"AT", send_buf, 3);
  for (;;) {
    digitalWrite(deviceSetPin, HIGH);  // Transparent mode
    delay(200);
    hc12.listen();
    int n = read_from_serial(64, receive_buf);
    if (n > 0) {
      receive_buf[n] = 0;
      Serial.print("Received: ");
      Serial.println((const char*)receive_buf);
    } else {
      Serial.println("nothing received");
    }
    digitalWrite(deviceSetPin, LOW); // Command mode
    delay(500);
    hc12.print(send_buf);
    Serial.print("Sent ");
    Serial.println((const char *)send_buf);
  }
}
```

Raspberry Pi code for HC-12

```
// Manferdelli, HC-12

#define PIN_ACCESS
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#ifdef PIN_ACCESS
#include <wiringPi.h>
#endif
#ifndef byte
typedef unsigned char byte;
#endif
```

```
// HC12 Connections
//      HC12  FDMI1232  RP
//      vcc   vcc       vcc
//      gnd   gnd       gnd
//      rx    tx        tx (pin 10)
//      tx    rx        rx (pin 8)
//      set   NC        set (pin 7)

#ifdef PIN_ACCESS
#define uartDevice "/dev/serial0"
#else
#define uartDevice "/dev/ttyUSB0"
#endif
#define BUF_SIZE 512
const int set_pin = 7;
```

Raspberry Pi code for HC-12

```
void clearBuf(byte* buf, int n) {
  for (int i = 0; i < n; i++) {
    buf[i] = 0;
  }
}

int bytes_available(int fd) {
  int num_bytes = 0;
  ioctl(fd, FIONREAD, &num_bytes);
  return num_bytes;
}

const char* eol = "\r\n";
void send_command(int fd, const char* command) {
  byte response[128];

  printf("Command: %s\n", command);

  // command mode
  digitalWrite(set_pin, LOW);
  delay(200);

tcflush(fd, TCIFLUSH);
  write(fd, command, strlen(command));
  int n = 0;
  delay(200);
  while (bytes_available(fd) > 0) {
    clearBuf(response, 128);
    n = read(fd, response, 127);
    if (n > 1) {
      printf("Response: %s", (char*)response);
    }
    tcflush(fd, TCIFLUSH);
  }

  digitalWrite(set_pin, HIGH);
  delay(200);
}
```

Raspberry Pi code for HC-12

```
bool setup(int fd, int new_baud_rate, int new_channel) {
#ifdef PIN_ACCESS
  int n = 0;
  byte request[128];

  pinMode(set_pin, OUTPUT);

  // test command, hc12 should return "OK"
  clearBuf(request, 128);
  send_command(fd, "AT");
  // baud rate
  clearBuf(request, 128);
  sprintf((char*)request, "AT+B%04d", new_baud_rate);
  send_command(fd, (char*)request);
  // channel command
  clearBuf(request, 128);
  sprintf((char*)request, "AT+C%03d", new_channel);
  send_command(fd, (char*)request);
  // back to transparent mode
  digitalWrite(set_pin, HIGH);
  delay(200);
#endif
  return true;
}
```

Raspberry Pi code for HC-12

```
int main(int an, char** av) {
  byte receive_buf[BUF_SIZE];
  byte send_buf[BUF_SIZE];

#ifdef PIN_ACCESS
  if (wiringPiSetup() < 0) {
    printf("Can't init wiringPi\n");
    return 1;
  }
#endif
  int fd = open(uartDevice,
         O_RDWR | O_NOCTTY | O_NDELAY);
  if (fd < 0) {
    printf("Can't open %s\n", uartDevice);
    return 1;
  }
  speed_t new_baud_rate = 9600;
  int new_channel = 6;
  for (int i = 0; i < (an - 1); i++) {
    if (strcmp(av[i], "-baud") == 0) {
      unsigned ls = 0;
      ls = atoi(av[++i]);
      new_baud_rate = ls;
      continue;
    }
```

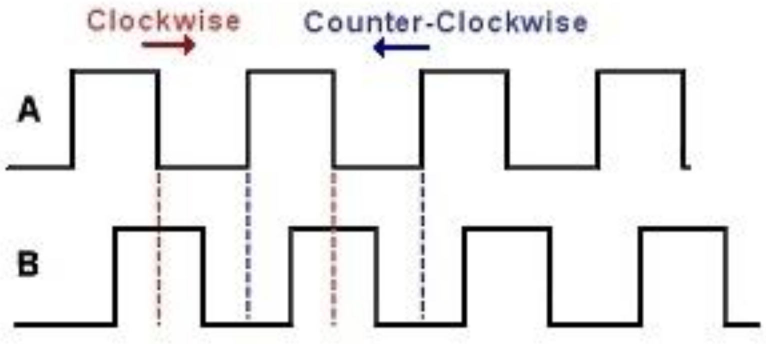
```
if (strcmp(av[i], "-channel") == 0) {
    unsigned ls = 0;
    ls = atoi(av[++i]);
    new_channel = ls;
    continue;
  }
}
if (new_channel < 1 || new_channel>100)
  new_channel = 1;
// turn off blocking for reads.
fcntl(fd, F_SETFL, 0);
// set baud rate
struct termios t;
tcgetattr(fd, &t);
cfsetispeed(&t, new_baud_rate);
tcsetattr(fd, TCSANOW, &t);

// setup hc-12
// We need pin access for this
setup(fd, new_baud_rate, new_channel);
int in_size, out_size;
```

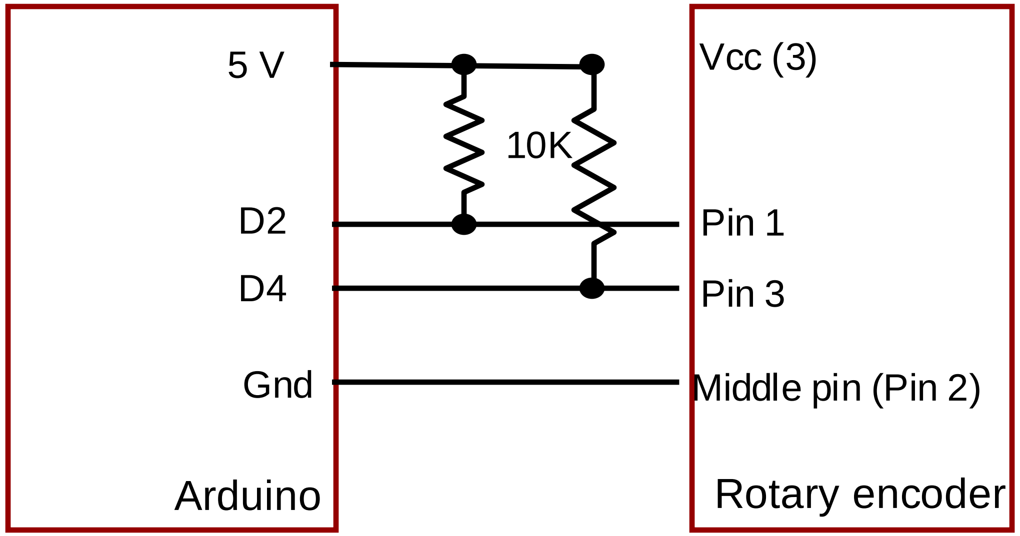
Raspberry Pi code for HC-12

```
  // setup hc-12
  setup(fd, new_baud_rate, new_channel);
  int in_size, out_size;

  tcflush(fd, TCIFLUSH);
  for(int i = 0; i < 5; i++) {
    while (bytes_available(fd) > 0) {
      clearBuf(receive_buf, BUF_SIZE);
      in_size = read(fd, receive_buf, BUF_SIZE - 1);
      receive_buf[in_size++] = 0;
      printf("Received: %s", (const char*)receive_buf);
    }
    clearBuf(send_buf, BUF_SIZE);
    sprintf((char*)send_buf,"Message %d", i);
    out_size = strlen((char*)send_buf);
    write(fd, send_buf, out_size);
    printf("Sent: %s\n", (char*)send_buf);
#ifdef PIN_ACCESS
    delay(200);
#endif
  }
  close(fd);
  return 0;
}
```

Lab: Rotary encoder

- If the encoder is rotating clockwise the output A will be ahead of output B.
- A digital device (like the Arduino) can track the speed of rotation and turn the sequence of pulses into a measured position.
- Wire the Arduino as shown.





Arduino code for rotary encoder

```
// Rotary encoder
// Manferdelli

typedef uint8_t byte;
const int clockPin= 2;
const int dataPin= 4;
const int measurementDelay= 100;

int position = -1;
void encoder() {
  if (digitalRead(clockPin) == digitalRead(dataPin))
    position++;
  else
    position--;
}
```

Arduino code for rotary encoder

```
void setup() {
  Serial.begin(9600);
  pinMode(clockPin, INPUT);
  pinMode(dataPin, INPUT);
  digitalWrite(clockPin, HIGH);
  digitalWrite(dataPin, HIGH);
  attachInterrupt(0, encoder, CHANGE);
}

void loop() {
  Serial.println("");
  Serial.print("Position: ");
  Serial.print(position);
  Serial.println("");
  delay(measurementDelay);
}
```

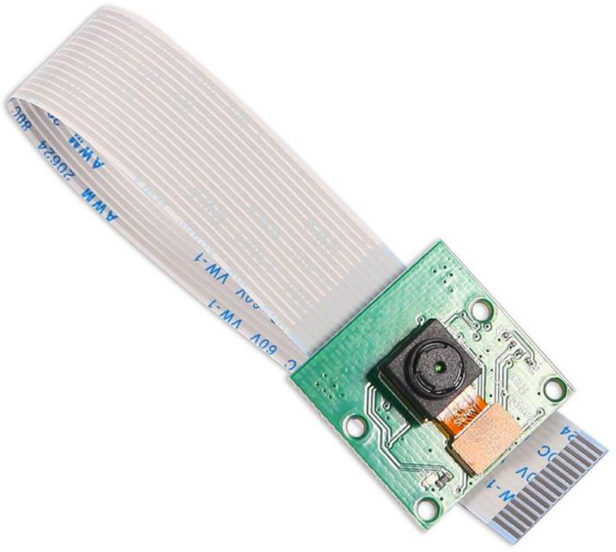
Lab: Use dead reckoning with an IMU and Google maps

- Openweather API is here.
- Google maps API is here.

  https://maps.googleapis.com/maps/api/staticmap?center=Brooklyn+Bridge,New+York,NY&zoom=13&size=600x300&maptype=roadmap &markers=color:blue%7Clabel:S%7C40.702147,-74.015794&markers=color:green%7Clabel:G%7C40.711614,-74.012318 &markers=color:red%7Clabel:C%7C40.718217,-73.998284 &key=YOUR_API_KEY
- Google speech API is here.

Lab: Raspberry Pi attached cameras

- Get a Raspberry Pi camera here. The data sheet is here.
- There are two versions of the Camera Module:
  - The standard version, which is designed to take pictures in normal light
  - The NoIR version, which doesn't have an infrared filter, so you can use it with an infrared light source to take pictures in the dark
- There are two command line utilities that use the camera
  - raspistill -o Desktop/image.jpg
  - raspivid -o Desktop/video.h264
- The first takes still pictures and the second captures video.
- There are also programmatic interfaces, which we describe next.
- You can update and configure the camera using the following:
  - sudo apt-get-update
  - sudo apt-get upgrade
  - sudo raspi-config



Using Raspberry Pi with Python

- Here's some sample python code that accesses the camera to capture a still:

```
from picamera import PiCamera
from time import sleep
camera = PiCamera()
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/image.jpg')
camera.stop_preview()
```

- You'll have to install python first this way:
  - sudo apt update
  - sudo apt install python3-picamera

Using Raspberry Pi with Python

- This code changes some camera properties:

```
camera.resolution = (2592, 1944)
camera.framerate = 15
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/max.jpg')
camera.stop_preview()
camera.start_preview()
camera.annotate_text = "Hello world!"
sleep(5)
camera.capture('/home/pi/Desktop/text.jpg')
camera.stop_preview()
```

Using Raspberry Pi with C++

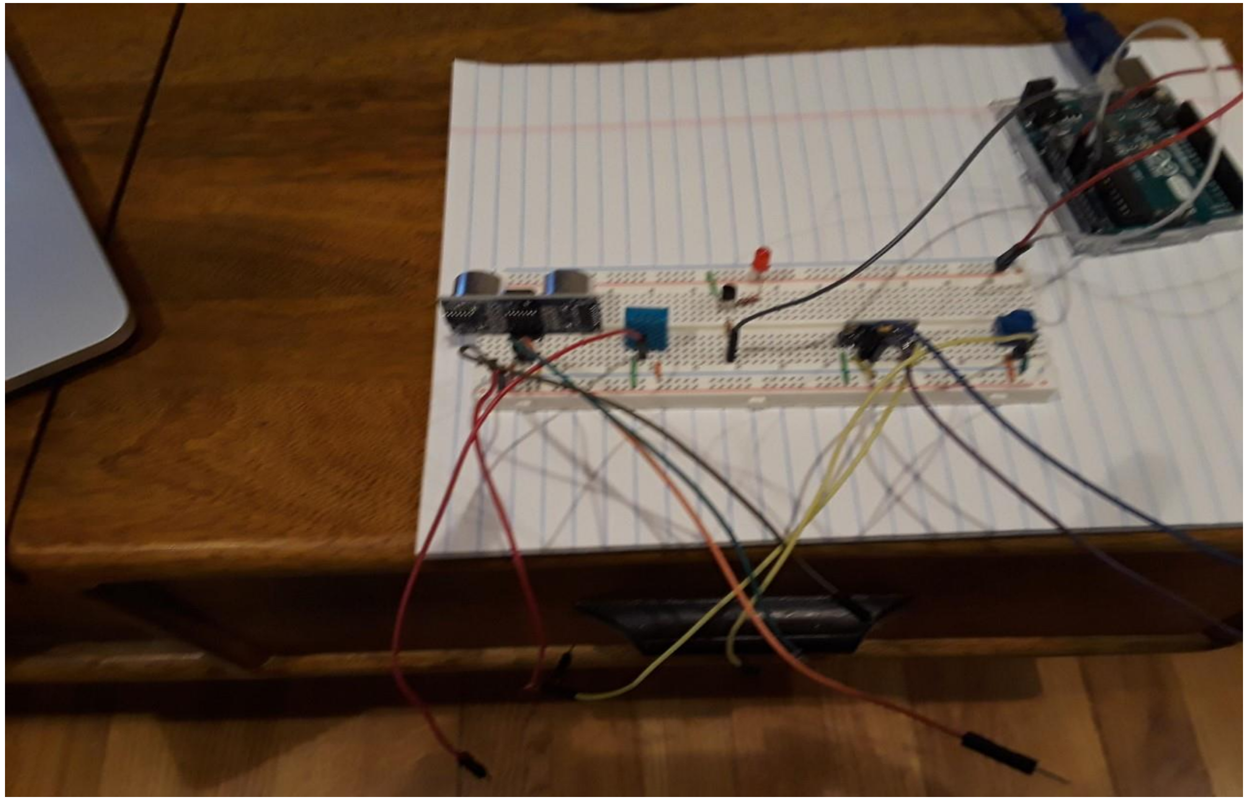
- There are several C++ camera interface libraries. Here we use raspicam which you can find [here](). To install raspicam, download it and install cmake:
  - sudo apt-get update && sudo apt-get upgrade
  - sudo apt-get install cmake. Then build the library.
- To compile the following sample code, do the following:
  - g++ still.cc -L /opt/vc/lib -I/usr/local/include -lraspicam -lmmal -lmmal_core -lmmal_util

```
#include <unistd.h>
#include <ctime>
#include <fstream>
#include <iostream>
#include <raspicam/raspicam.h>
using namespace std;

int main (int argc, char **argv) {
    raspicam::RaspiCam Camera;

    if (!Camera.open()) {cerr<<"Error opening camera"<<endl; return -1;}

… continued on next slide
```

Using Raspberry Pi with C++

```
    // Wait a while until camera stabilizes
    sleep(3);

    // Capture
    Camera.grab();
    unsigned char *data=new unsigned char[
       Camera.getImageTypeSize(raspicam::RASPICAM_FORMAT_RGB)];
    Camera.retrieve (data,raspicam::RASPICAM_FORMAT_RGB );//get camera image

    // Save
    std::ofstream outFile("raspicam_image.ppm",std::ios::binary);
    outFile<<"P6\n"<<Camera.getWidth() <<" "<<Camera.getHeight() <<" 255\n";
    outFile.write ((char*) data,
         Camera.getImageTypeSize(raspicam::RASPICAM_FORMAT_RGB));
    cout<<"Image saved at raspicam_image.ppm"<<endl;

    // Free resources
    delete data;
    return 0;
}
```

Some of the experiments on a perf board



References

1. Feynman's Lectures on Physics.
2. Berkeley physics course volumes 1, 2, 3.
3. Sears, Zemansky and Young, University Physics.
4. Schaum's outline in Physics for Engineering and Science.
5. Griffiths, Introduction to Electrodynamics.
6. ARRL Handbook.
7. Rutledge, Electronics of Radio.
8. Scherz, Practical Electronics for Inventors.
9. Bryant and O'Halleron, Computer Systems, a programmer's perspective.
10. Patterson and Hennessy, Computer Architecture.

References

11. IBM, System 360, Principles of Operation (a classic)
12. Intel, Intel Processor Architecture (5 volumes)
13. Arm Architecture manual
14. Tannenbaum, Operating systems
15. http://www.freenove.com (Sensor kit projects)
16. Tero Karvinen, Kimmo Karvinen, Ville Valtokari, Make: Sensors
17. Wallace, Richardson, Getting started with Raspberry Pi
18. Tero Karvinen, Kimmo Karvinen, Make: Arduino Bots and Gadgets

Exercises

1. Get the data sheets for the processors, sensors, and memory for some IoT device.
2. Get the RF sources from an IoT devices (frequency, power, ...) from the FCC filings.
3. Get the circuit board layouts for an IoT device? What could you use them for?
4. Calculate the current drain on the Arduino pins for some of our experiments.
5. Use the Arduino to build a communication line based on infrared.
6. Use the Arduino to build an RFID reader.
7. Use the Arduino to connect a Hall (magnetic field) sensor.
8. Connect a small digital radio (e.g.- NRF24L01, Adafruit RFM69HCW Transceiver).
9. Display some of our sensor readings on a local small screen like the LCD 1602.
10. How would you use a gyro and accelerometer to do "dead reckoning."
11. Connect a GPS module and investigate the theory of GPS positioning and accuracy.

Tools and equipment

- Standard Electronics tools (~$50)
- De-soldering station ($99)
- 5 Perf boards ($10), lots of wired jumpers, clips to connect to chips (e.g.-EEPROMs) ...
- 2 Volt-ohm meters ($50)
- Signal generator ($40)
- Oscilloscope ($350)
- Various parts (see Practical Electronics)
- Attify Badge (to connect to I2C, SPI, UARTs and JTAGs) ($20)
- Arduino ($15)
- Sensor starter kits, some include Arduinos ($50-$80). These contain accelerometers, light sensors, compass, GPS modules, temp/humidity
- Power supplies
- More exotic: signal generator, spectrum analyzer.

License

- This material is licensed under Apache License, Version 2.0, January 2004.
- Use, duplication or distribution of this material is subject to this license and any such use, duplication or distribution constitutes consent to license terms.
- You can find the full text of the license at: http://www.apache.org/licenses/.

All classical physics --- behold

- $\gamma(\mathrm{v}) = \ [1 - \left(\frac{\mathrm{v}}{\mathrm{c}}\right)^2]^{-\frac{1}{2}}$, $c = 2.99725 \ \times \ 10^8 \ \frac{m}{s}$
- $\mathbf{p} = \mathrm{m}\mathbf{v}, \mathrm{m} = \ \mathrm{m}_0\gamma(\mathrm{v})$
- $\mathbf{F} = -\mathrm{G} \ \frac{m_1 m_2}{r^2} \ \boldsymbol{e_r}$**,** $\mathrm{G} = 6.671 \ \times \ 10^{-11} \ \frac{N-m^2}{kg^2}$
- $\nabla \cdot \mathbf{E} = \ \frac{\rho}{\epsilon_0}$, ϵ_0 = 8.854 × 10^{-12} $C^2/N\text{-}m^2$.
- $\nabla \cdot \mathbf{B} = 0$
- $\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial \mathrm{t}}$
- $\mathrm{c}^2 \ \nabla \times \mathbf{B} = \ \frac{\mathbf{j}}{\epsilon_0} + \ \frac{\partial \mathbf{E}}{\partial \mathrm{t}}$
- $\nabla \cdot \mathbf{j} = -\frac{\partial \rho}{\partial \mathrm{t}}$
- $\mathbf{F} = \mathrm{q}(\mathbf{E} + \mathbf{v} \times \boldsymbol{B})$

Physical data and phenomenology

- $k_B = 1.38 \times 10^{-16} \frac{ergs}{deg}$
- $h = 6.6262 \times 10^{-27}\, erg - sec$
- $pV = nRT, R = 8.3143 \frac{J}{mol-deg}$
- $q_e = 1.6022 \times 10^{-19}\, C$
- $F_{spring} = -kx$
- $\rho_{air} = 1.293 \frac{mg}{cm^3}, \rho_{water} = 1 \frac{g}{cm^3}, \rho_{ice} = .917 \frac{g}{cm^3}$
- $v_{sound-air} = 330 \frac{m}{s}$
- Solution to Maxwell:
  - $\phi(1,t) = \int \frac{\rho(2,t-\frac{r_{12}}{c})}{4\pi\epsilon_0 r_{12}}\, dV$, $E = -(\nabla\phi + \frac{\partial A}{\partial t})$
  - $A(1,t) = \int \frac{\boldsymbol{j}(2,t-\frac{r_{12}}{c})}{4\pi\epsilon_0 c^2 r_{12}}\, dV$, $B = \nabla \times A$

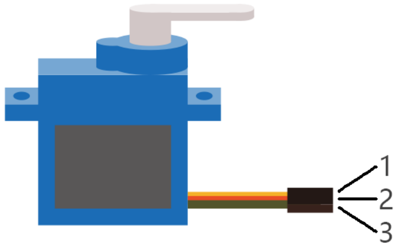
Physical data and phenomenology

- In conductors, $\boldsymbol{j} = \sigma \boldsymbol{E}$. $\rho = \frac{1}{\sigma}$, the resistivity.
  - $\rho_{Cu} = 1.7 \times 10^{-8}\, ohm - m$
  - $\rho_{rubber} = 10^{15}\, ohm - m$
- Poynting: $S = \frac{1}{\mu_0} (\mathrm{E} \times B)$.
- In materials:
  - $D = \epsilon_0 E + P$
  - $H = \frac{1}{\mu_0} B - M$
  - $\nabla \cdot D = \rho_f$
  - $\nabla \times H = j_f + \frac{\partial D}{\partial t}$
  - $j_d = \frac{\partial D}{\partial t}$
  - Often, $P = \epsilon_0 \chi_m E$, $D = \epsilon E$, $M = \chi_m H$, $H = \frac{1}{\mu} B$

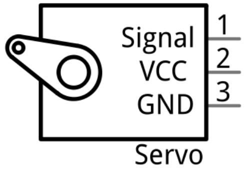
Raspberry Pi 3 pins

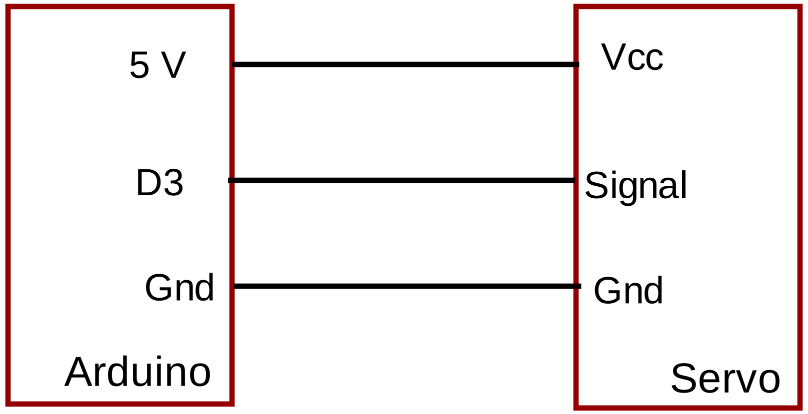
Lab: Servo

- You can find this in the Freenove manuals here.
- Wire the servo and Arduino as shown.



Freenove





Arduino code for Servo

```
// Stepper
// Manferdelli

typedef uint8_t byte;
const int measurementDelay= 1000;

int out_ports[4] = {
  11, 10, 9, 8
};

void setup() {
  Serial.begin(9600);
  for (int i = 0; i < 4; i++)
    pinMode(out_ports[i], OUTPUT);
}

const int FW = 0;
const int BW = 1;
```

Arduino code for Servo

```
void move_one_step(int dir) {
  static byte out = 1;
  if (dir == FW) {
    if (out != 0x08)
      out = out << 1;
    else
      out = 1;
  } else {
    if (out != 1)
      out = out >> 1;
    else
      out = 0x08;
  }
  for (int i = 0; i < 4; i++) {
    digitalWrite(out_ports[i], (out & (1<<i)) ? HIGH:LOW);
  }
}
```

Arduino code for Servo

```
void move_steps(int steps, byte ms_delay) {
  if (steps > 0) {
    for (int i = 0; i < steps; i++) {
      move_one_step(FW);
      delay(ms_delay);
    }
  } else {
    for (int i = 0; i < -steps; i++) {
      move_one_step(BW);
      delay(ms_delay);
    }
  }
}

void loop() {
  move_steps(2048, 4);
  delay(measurementDelay);
  move_steps(-2048, 4);
  delay(measurementDelay);
}
```