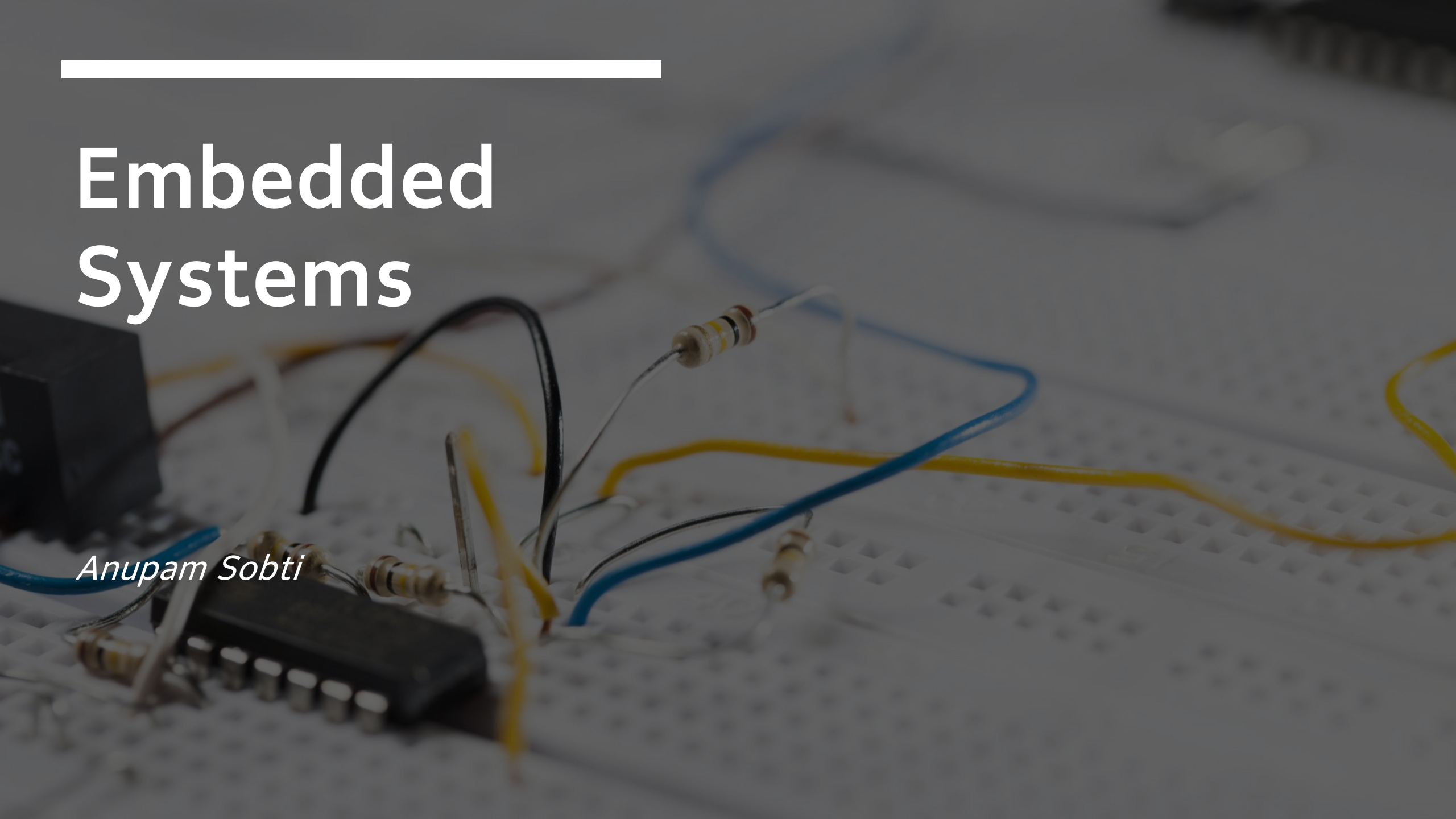
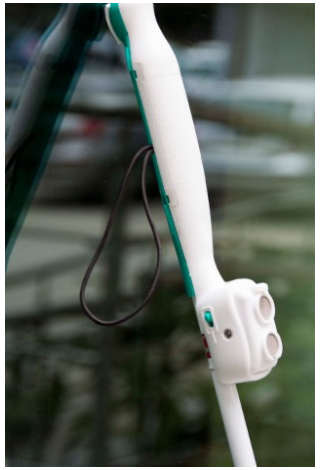


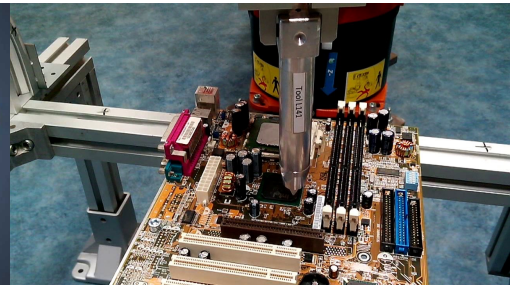
Embedded Systems

Anupam Sobti

What?

An electronic system with a dedicated purpose

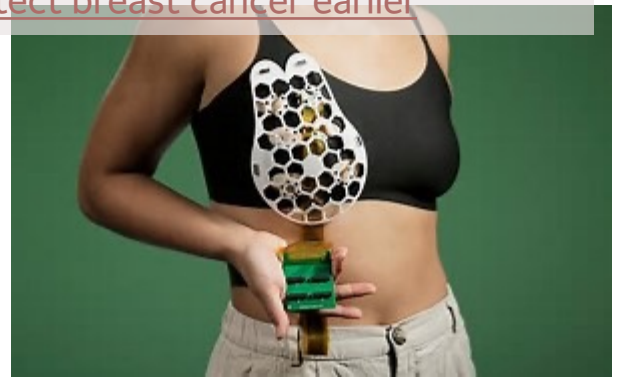




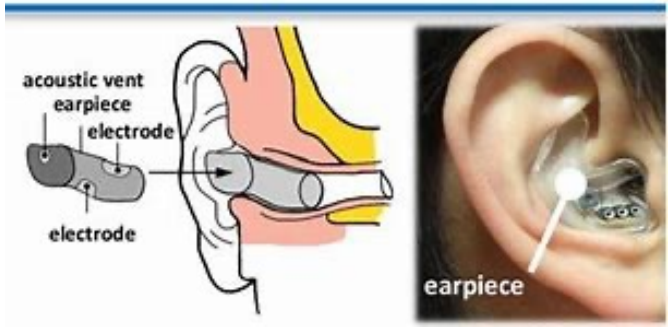
[Air Conditioner PCB Tutorial With Its Working and Repair - Hackster.io]

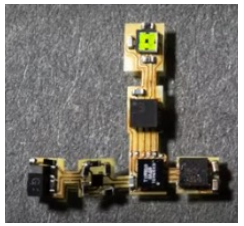
What?

A wearable ultrasound scanner could detect breast cancer earlier



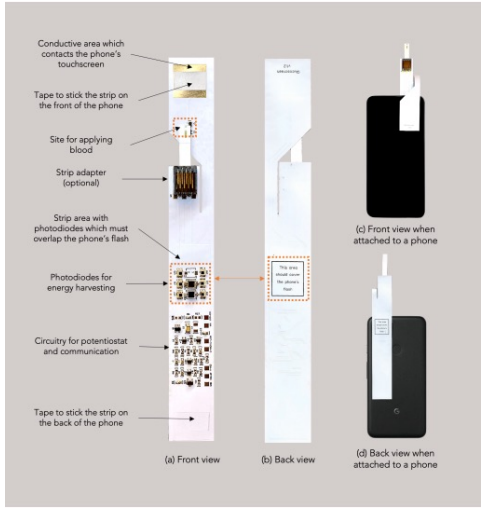
In-ear EEG

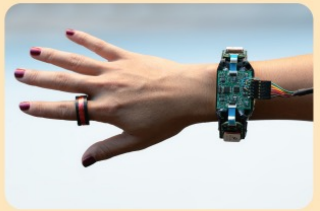




Battery Free sensors that float in the wind

Glucoscreen





AuraRing
Tracking accurate finger motion using magnetic sensing

Why?

- Interface between humans and environment
- There is no software without hardware and hardware is a rare skill even today
- All grand challenges require some hardware:
  - Sustainable practices

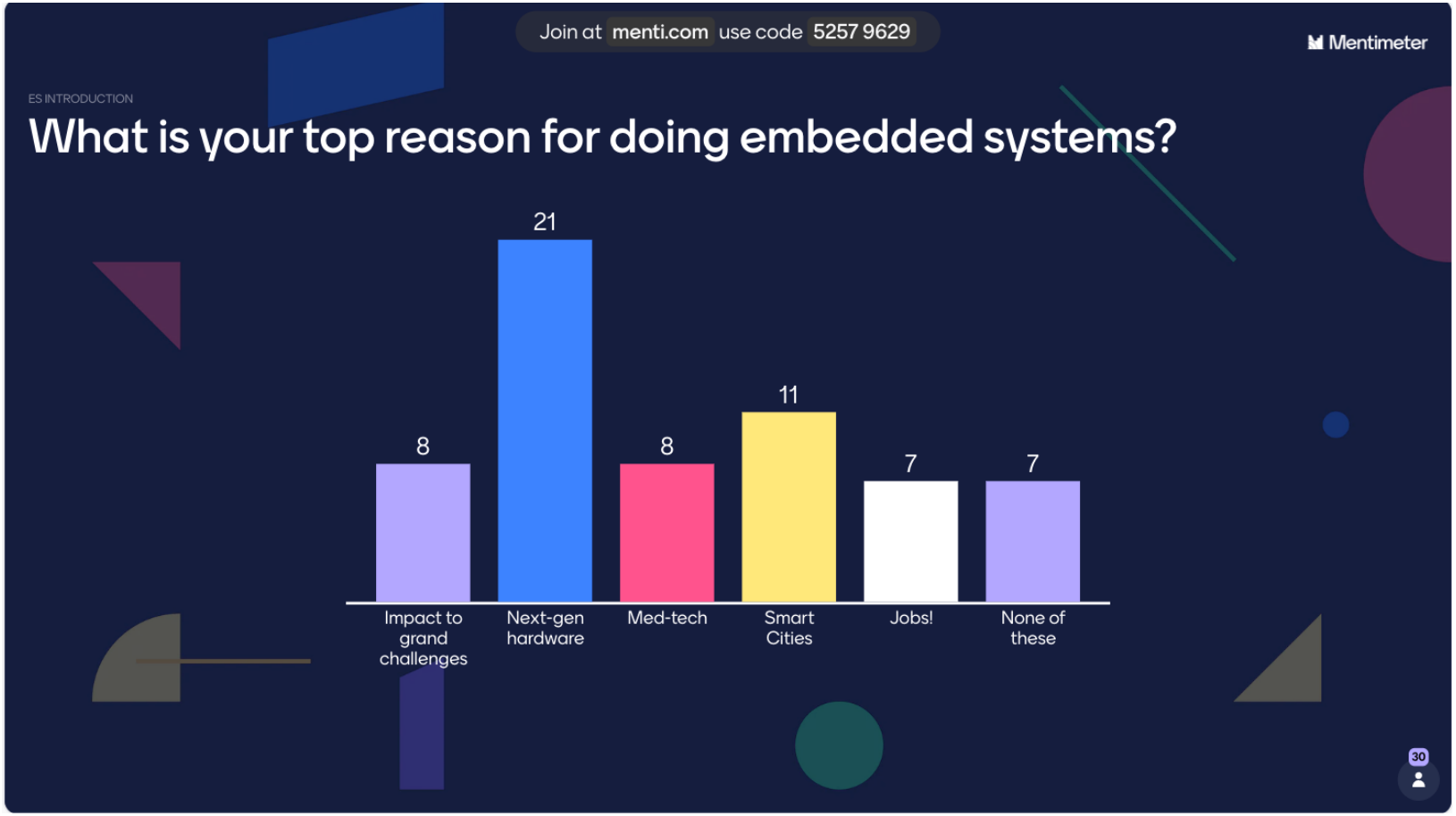
    Optimal resource usage – energy, water, heat, etc.
  - Health
  - Food
- Augment human intelligence through additional sensing/perception capabilities

Why should YOU do it?

- Build next-gen hardware – drones, robots, but also remotes, fans, air conditioners, cooking stoves, generators!
- Build a physical reality for software solutions and vice versa
- Sheer lack of stable hardware – broken traffic lights, unrepairable devices, expensive imports!
- Upcoming medtech, smart cities!
- Jobs: firmware engineer (e.g., Intel, Oculus, Google, Apple), hardware design (medtech, car companies, renewable energy devices, Bosch), etc.

Menti

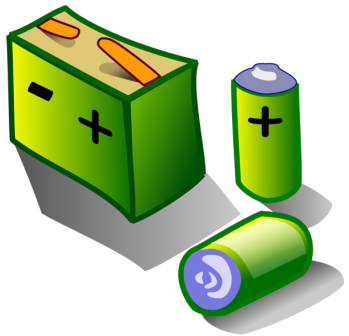
Results


Join at menti.com use code 5257 9629
Mentimeter
ES INTRODUCTION
What is your top reason for doing embedded systems?
8
21
8
11
7
7
Impact to grand challenges
Next-gen hardware
Med-tech
Smart Cities
Jobs!
None of these
30

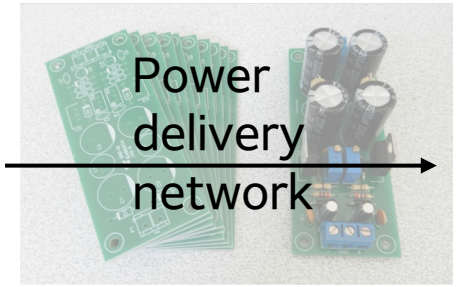
How?

- Basic electronics – understand power requirements – steady state and transient
- Understand Computer Architecture – Every architecture is different offering various specialist use cases
- Software – Various programming models are used – embedded C, OS, RTOS, Rust!

The building blocks

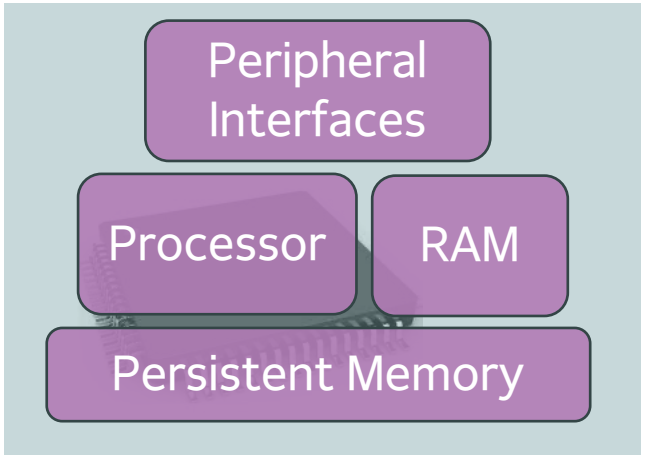


Reliable **power** source



Peripherals

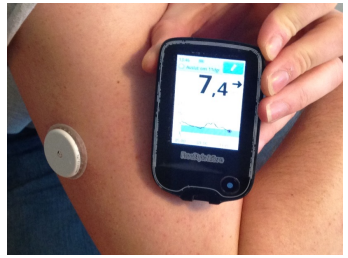
Communication

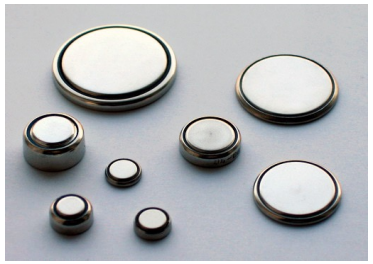


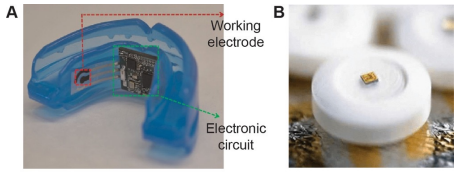
Compute

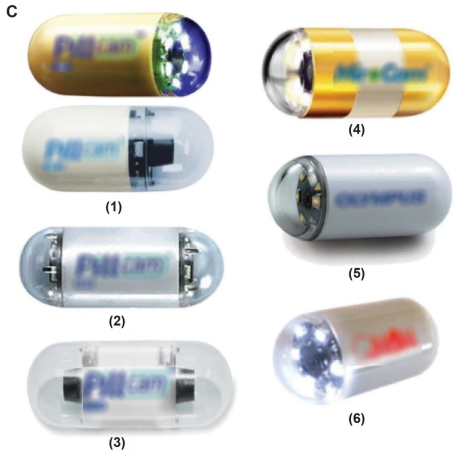
Constraints

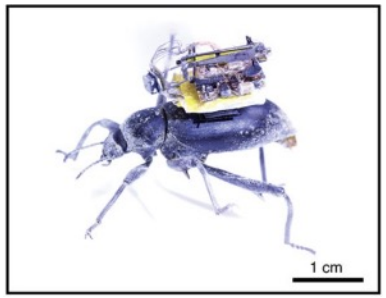
- Battery Life
- Form Factor
- Cost of failure
- Performance requirement

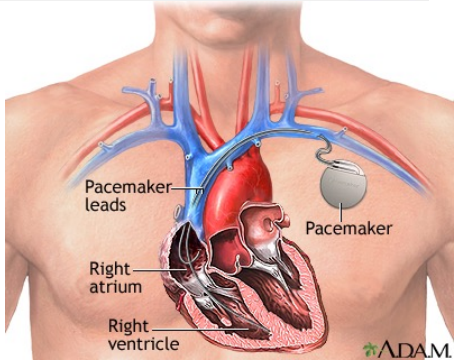






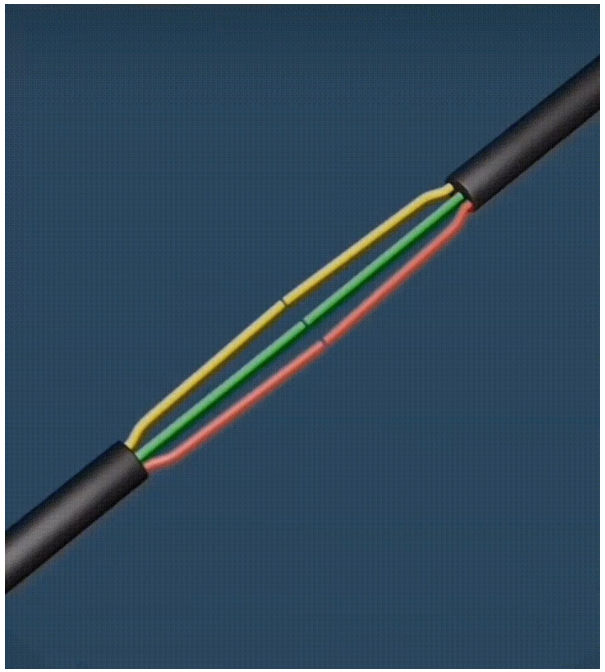






Some basics

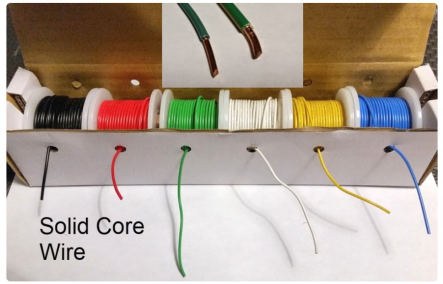
Hand-holding circuits

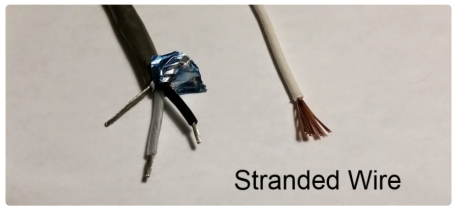


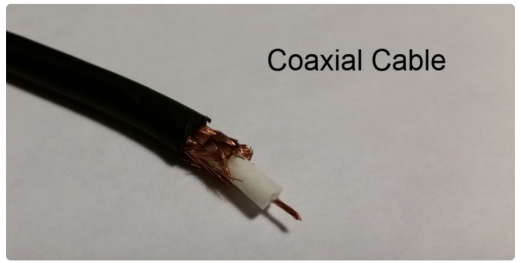
Putting wires together

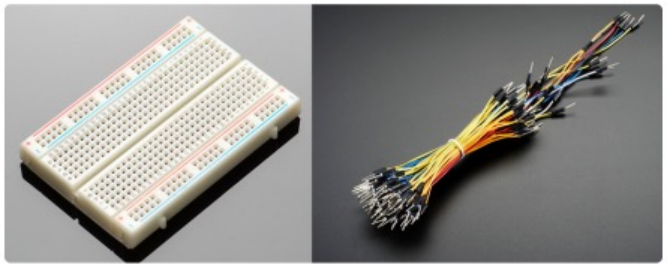
Wires

wires-and-connections.pdf (adafruit.com)

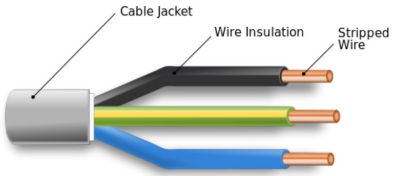








Cables



HF signals

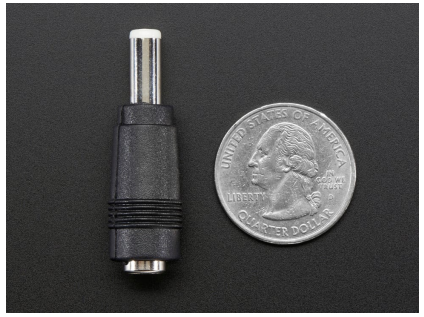
- Breadboard
- HF Signals
- More current?
- Wearables
- Flexible electronics
- Secure your cables

- Single core solid wire
- Coaxial cables
- Stranded Wires
- Conductive Threads
- Copper Tape
- Ties, clips, bolts

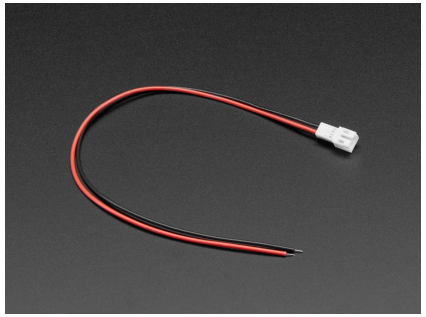
Connectors

wires-and-connections.pdf (adafruit.com)

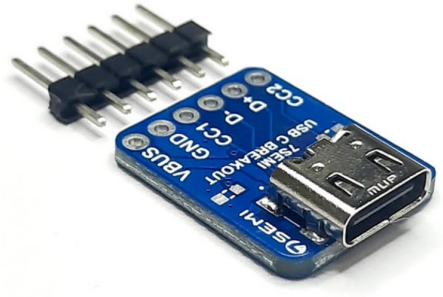
Power



Barrel Plugs

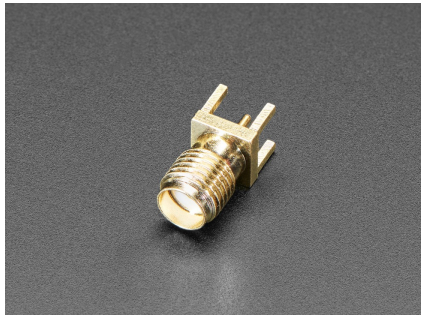


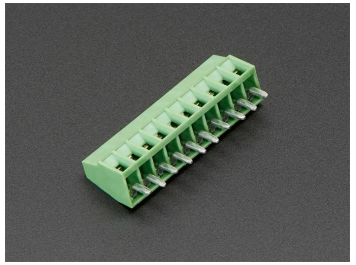
JST Connectors
Japan Solderless
Terminals



USB Connectors

Shrinks





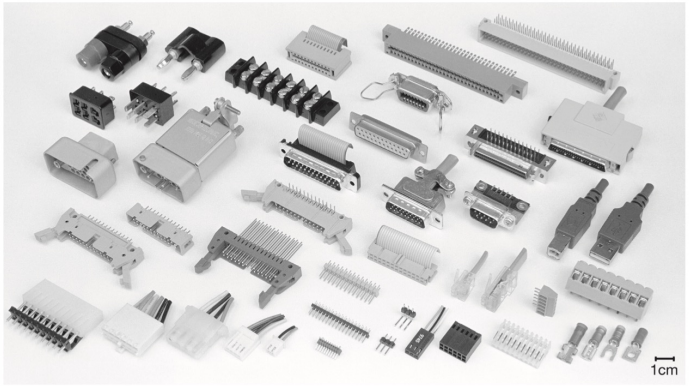


Figure 1.123. Rectangular connectors. The variety of available multipin connectors is staggering. Here is a collection of common speci-

Resistors

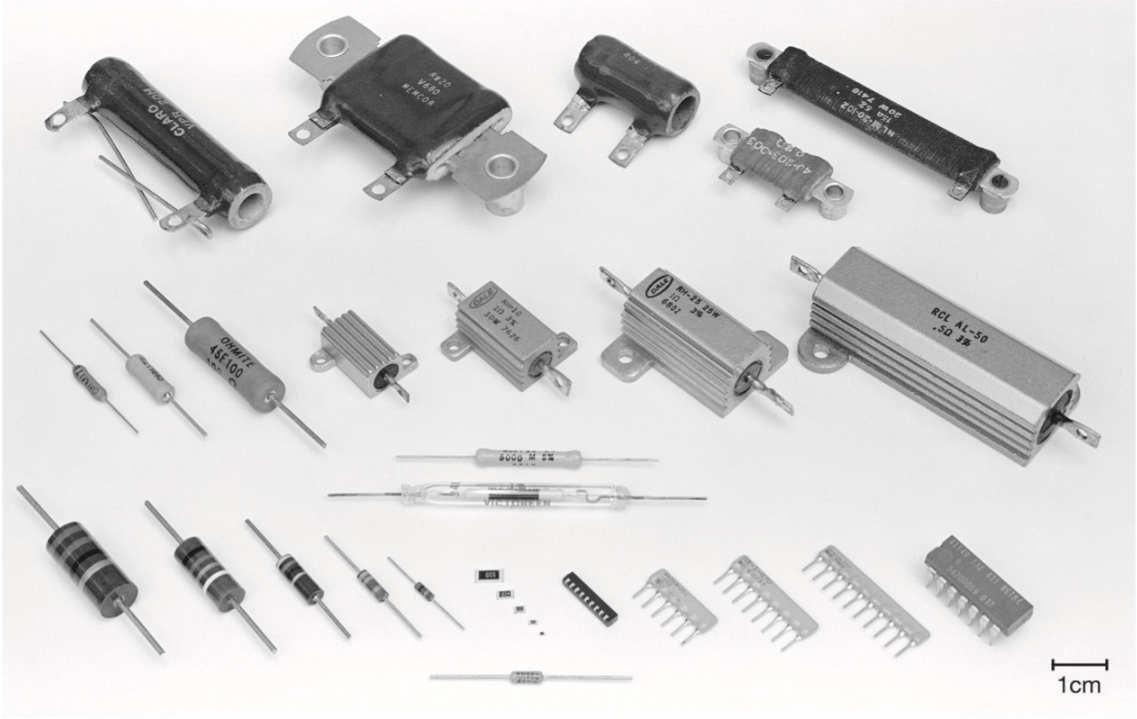


Figure 1.2. A selection of common resistor types. Top row, left to right (wirewound ceramic power resistors): 20W vitreous enamel with leads, 20W with mounting studs, 30W vitreous enamel, 5W and 20W with mounting studs. Middle row (wirewound power resistors): 1W, 3W, and 5W axial ceramic; 5W, 10W, 25W, and 50W conduction-cooled ("Dale-type"). Bottom row: 2W, 1W, $\frac{1}{2}$W, $\frac{1}{4}$W, and $\frac{1}{8}$W carbon composition; surface-mount thick-film (2010, 1206, 0805, 0603, and 0402 sizes); surface-mount resistor array; 6-, 8-, and 10-pin single in-line package arrays; dual in-line package array. The resistor at bottom is the ubiquitous RN55D $\frac{1}{4}$W, 1% metal-film type; and the pair of resistors above are Victoreen high-resistance types (glass, 2 GΩ; ceramic, 5 GΩ).

From Art of Electronics, 3rd Ed

Variable Resistors

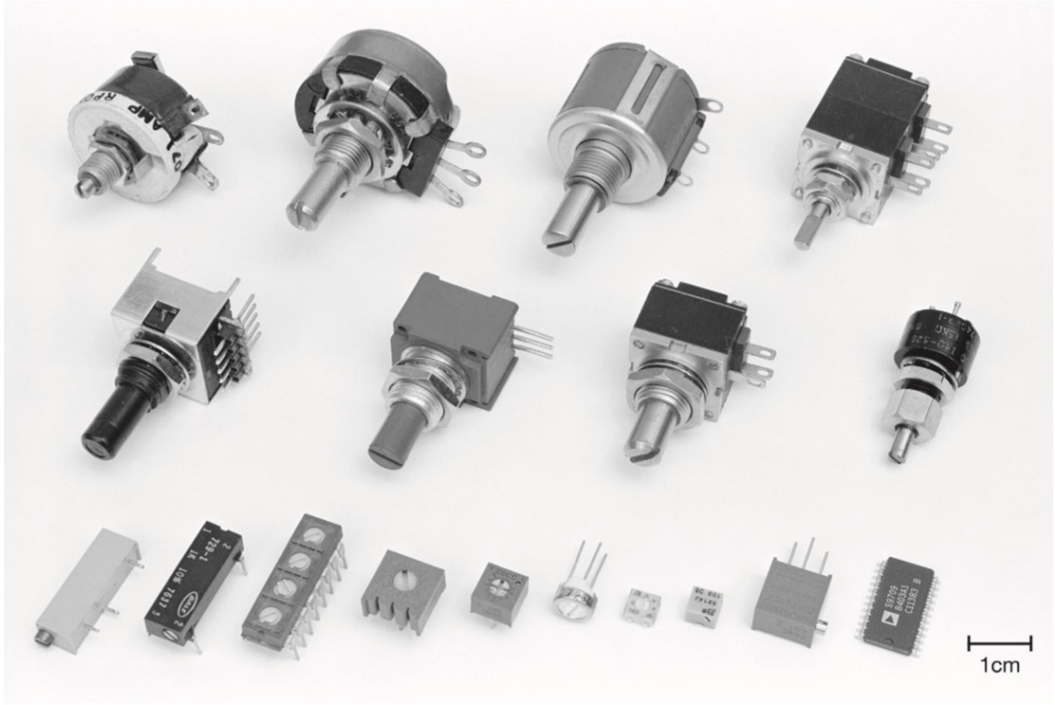


Figure 1.8. Most of the common potentiometer styles are shown here. Top row, left to right (panel mount): power wirewound, "type AB" 2W carbon composition, 10-turn wirewound/plastic hybrid, ganged dual pot. Middle row (panel mount): optical encoder (continuous rotation, 128 cycles per turn), single-turn cermet, single-turn carbon, screw-adjust single-turn locking. Front row (board-mount trimmers): multiturn side-adjust (two styles), quad single-turn, 3/8″ (9.5 mm) square single-turn, 1/4″ (6.4 mm) square single-turn, 1/4″ (6.4 mm) round single-turn, 4 mm square single-turn surface mount, 4 mm square multiturn surface mount, 3/8″ (9.5 mm) square multiturn, quad nonvolatile 256-step integrated pot (E^2POT) in 24-pin small-outline IC.

From Art of Electronics, 3rd Ed

Capacitors

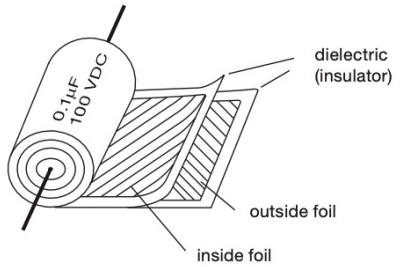
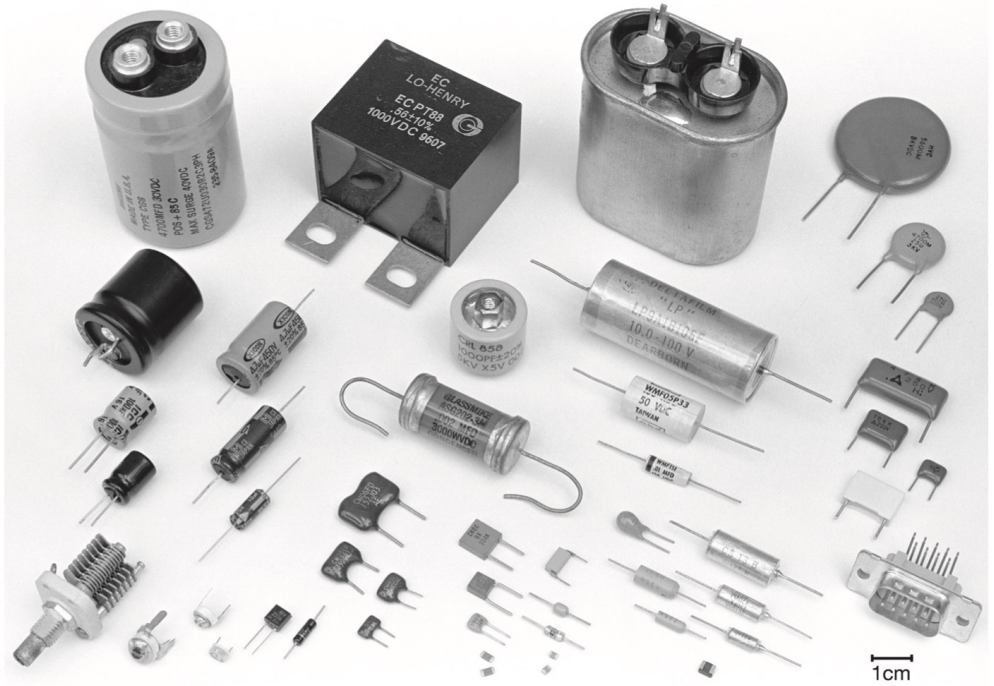
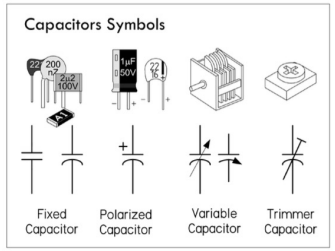


Figure 1.28. You get a lot of area by rolling up a pair of metallized plastic films. And it's great fun unrolling one of these axial-lead Mylar capacitors (ditto for the old-style golf balls with their lengthy wound-up rubber band).





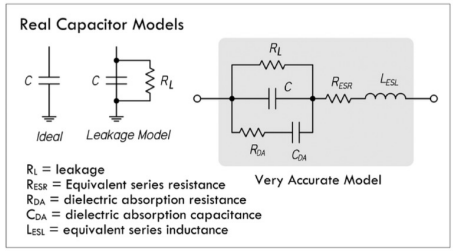


Figure 1.30. Capacitors masquerade as anything they like! Here is a representative collection. In the lower left are small-value variable capacitors (one air, three ceramic), with large-value polarized aluminum electrolytics above them (the three on the left have *radial leads*, the three on the right have *axial leads*, and the specimen with screw terminals at top is often called a *computer electrolytic*). Next in line across the top is a low-inductance film capacitor (note the wide strap terminals), then an oil-filled paper capacitor, and last, a set of disc ceramic capacitors running down the right. The four rectangular objects below are film capacitors (polyester, polycarbonate, or polypropylene). The D-subminiature connector seems misplaced – but it is a *filtered* connector, with a 1000 pF capacitor from each pin to the shell. To its left is a group of seven polarized tantalum electrolytics (five with axial leads, one radial, and one surface-mount). The three capacitors above them are axial-film capacitors. The ten capacitors at bottom center are all ceramic types (four with radial leads, two axial, and four surface-mount *chip capacitors*); above them are high-voltage capacitors – an axial-glass capacitor, and a ceramic *transmitting capacitor* with screw terminals. Finally, below them and to the left are four mica capacitors and a pair of diode-like objects known as *varactors*, which are voltage-variable capacitors made from a diode junction.

Inductors

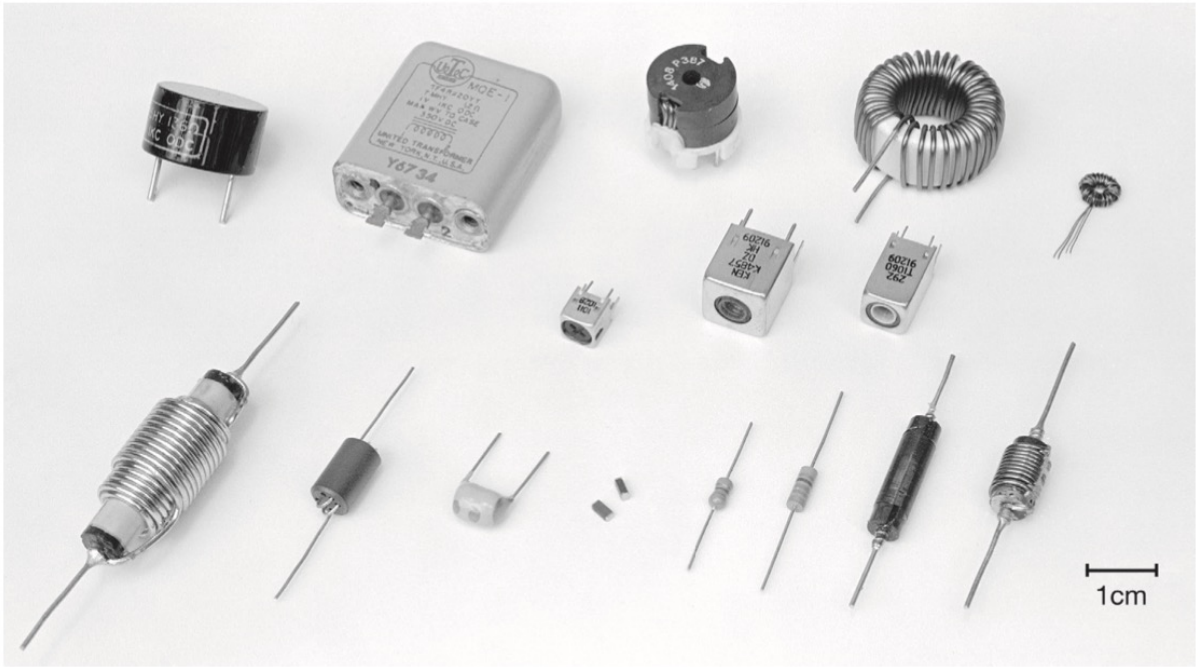


Figure 1.51. Inductors. Top row, left to right: encapsulated toroid, hermetically-sealed toroid, board-mount pot core, bare toroid (two sizes). Middle row: slug-tuned ferrite-core inductors (three sizes). Bottom row: high-current ferrite-core choke, ferrite-bead choke, dipped radial-lead ferrite-core inductor, surface-mount ferrite chokes, molded axial-lead ferrite-core chokes (two styles), lacquered ferrite-core inductors (two styles).

Switches

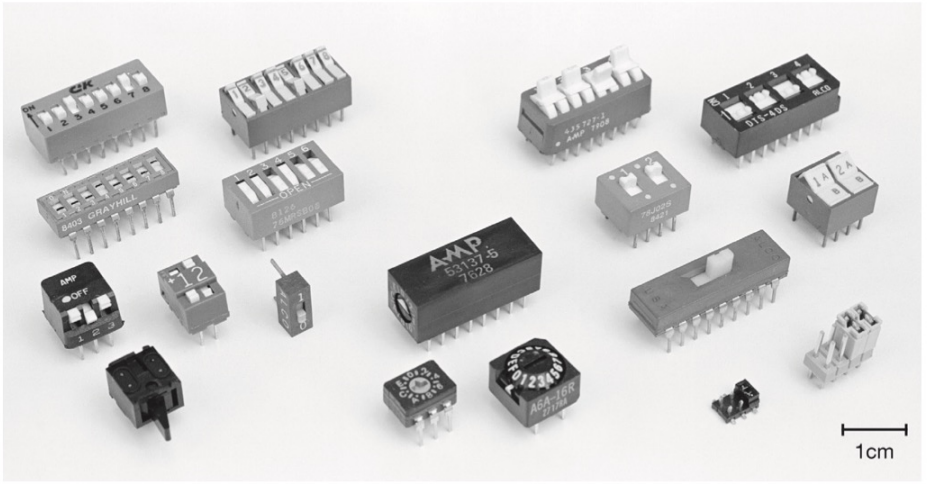
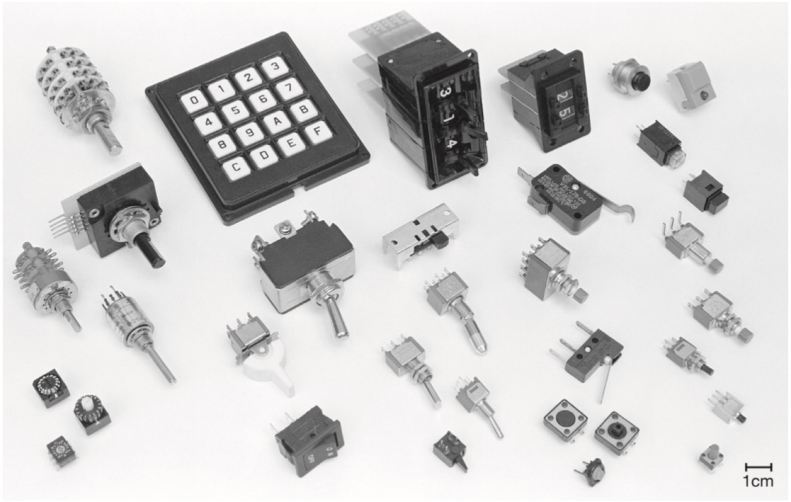


Figure 1.118. Board-mounted "DIP switches." Left group, front to back and left to right (all are SPST): single station sic three-station side-action, two-station rocker, and single-station slide; eight-station slide (low-profile) and six-station rock slide and rocker. Middle group (all are hexadecimal coded): six-pin low-profile, six-pin with top or side adjust; 16-pin wit plement coding. Right group: 2 mm×2 mm surface-mount header block with movable jumper ("shunt"), 0.1″×0.1″ (2.54 through-hole header block with shunts; 18-pin SPDT (common actuator); eight-pin dual SPDT slide and rocker; 16-pin q (two examples).



7. Switch Smorgasbord. The nine switches at right are momentary-contact ("pushbutton") switches, including both panel- ıd PCB-mounting types (PCB, printed-circuit board). To their left are additional types, including lever-actuated and multipole e them are a pair of panel-mounting binary-coded *thumbwheel* switches, to the left of which is a matrix-encoded hexadecimal switches at center foreground are toggle switches, in both panel-mounting and PCB-mounting varieties; several actuator nown, including a locking va rotary switches in the left trate binary-coded types (the –multiposition configurable es.

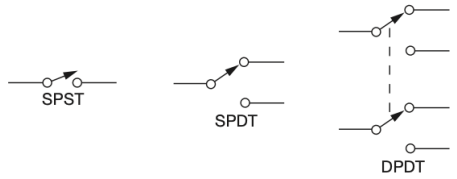


Figure 1.119. Fundamental switch types.

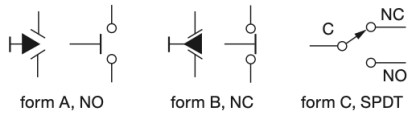


Figure 1.120. Momentary-contact (pushbutton) switches.

SMD components

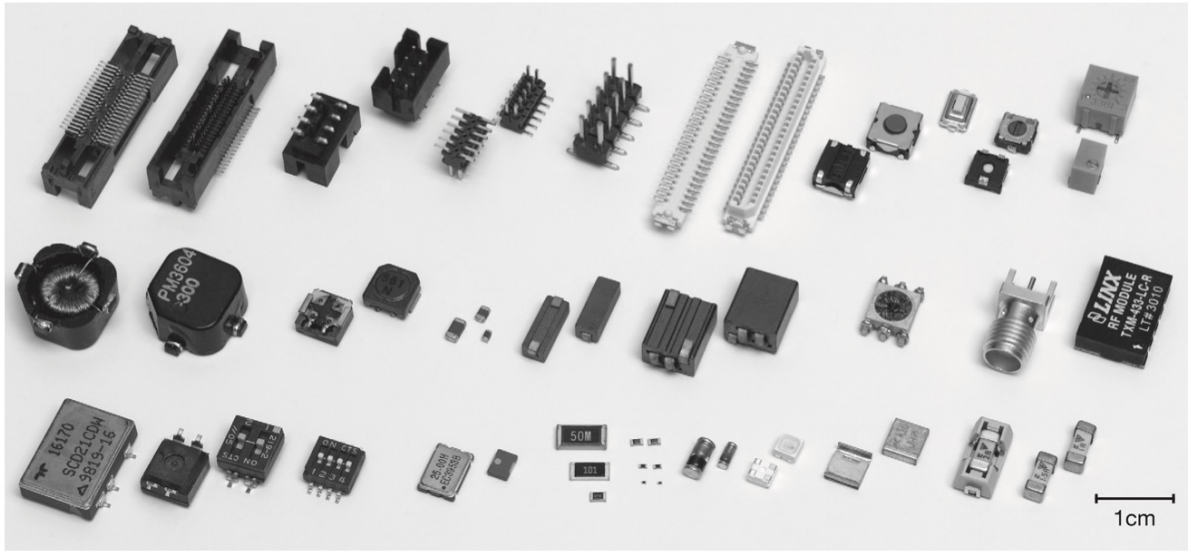


Figure 1.133. A taste of the world of passive components in surface-mount packages: connectors, switches, trimmer pots, inductors, resistors, capacitors, crystals, fuses. . . . If you can name it, you can probably get it in SMT.

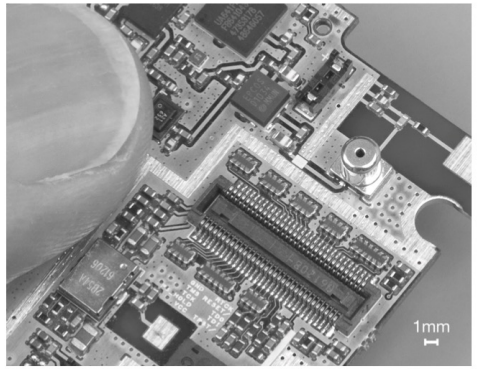


Figure 1.131. We're "all thumbs" when working with surface-mount technology (SMT). This is a corner of a cellphone circuit board, showing small ceramic resistors and capacitors, integrated circuits with ball-grid connecting dots on their undersides, and the Lilliputian connectors for the antenna and display panel. See also Figure 4.84.

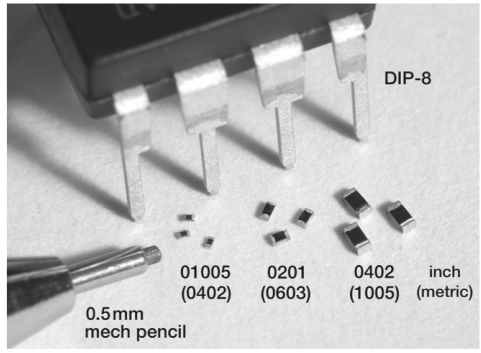


Figure 1.132. How small can these things get?! The "01005"-size SMT (0.016″×0.008″, or 0.4mm×0.2mm) represents the industry's greatest insult to the experimenter.

Tools

- Eagle/KiCAD
- Everycircuit
- 3D printing

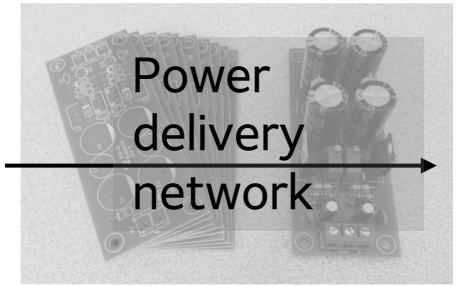
References

Books

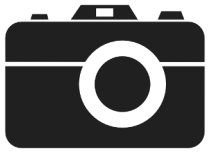
Art of Electronics, 3rd Ed. By Paul Horowitz, Winfield Hill
Practical electronics for Inventors by Paul Scherz

The building blocks

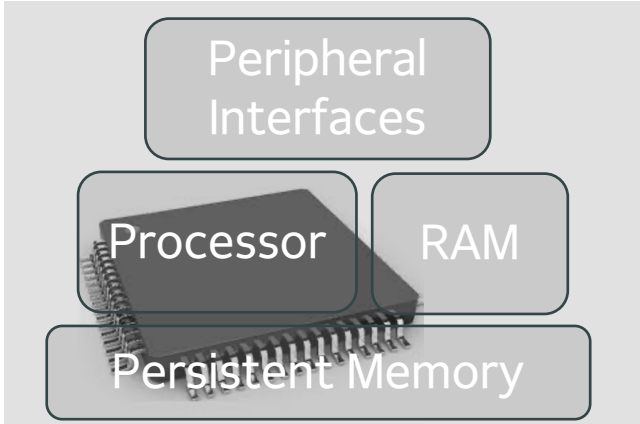
Reliable **power** source



Peripherals



Communication



Compute

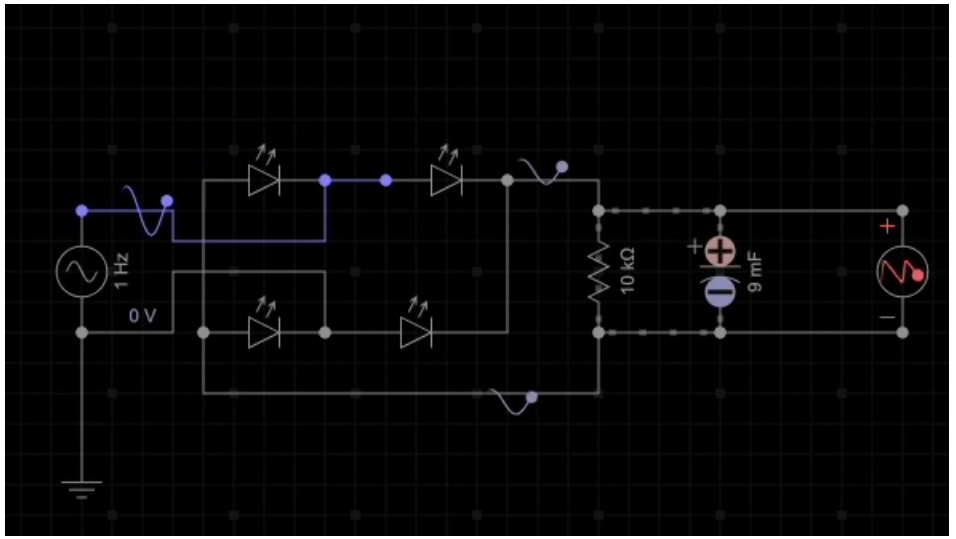
Power Supplies

AC to DC

Simple regulator circuit -> Rectifier + Filter

https://everycircuit.com/circuit/5073547604787200

Problems?
- Powerline ripple



VDC to VDC

Step-Down
Step-Up

VDC to IDC

Constant Current sources

Types of power Supplies

Basic idea:
Use a voltage reference with some negative feedback to provide regulation

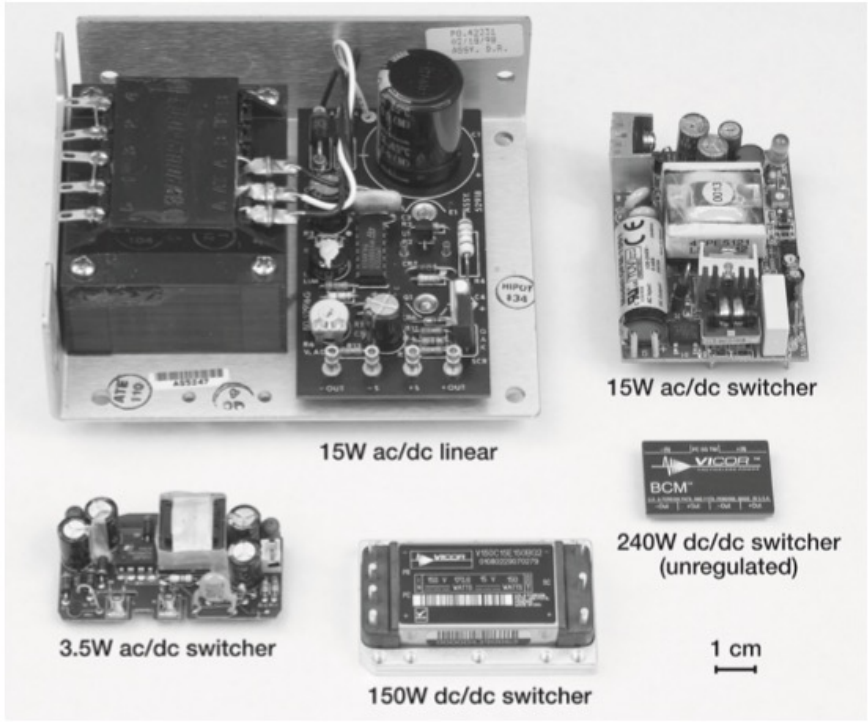
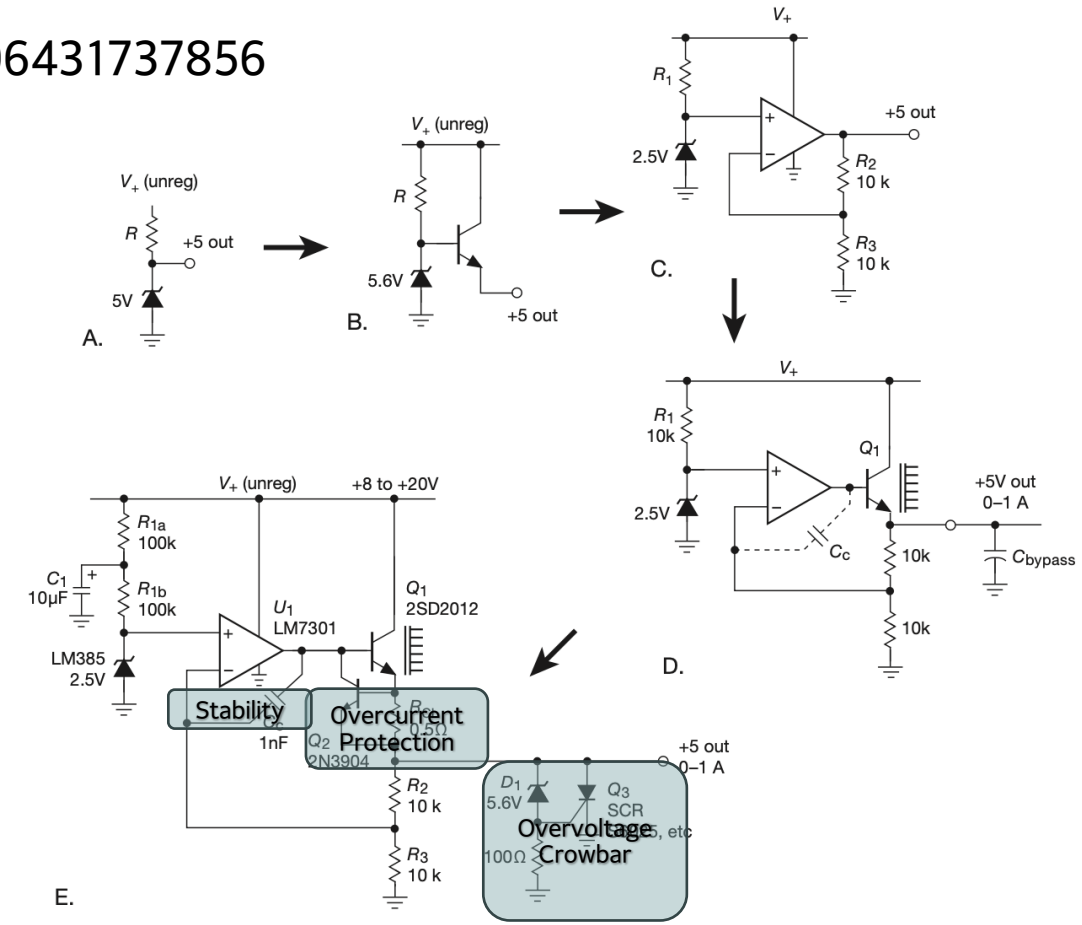


Figure 9.1. Switching power supplies ("switchers") are smaller and more efficient than traditional linear regulated power supplies, switching operation generates some unavoidable electrical noise.

Example regulator

https://everycircuit.com/circuit/5202706431737856



Ref: AOE

Thank AIC designers for

7805, 7812, etc.

With a current of upto 1.5A

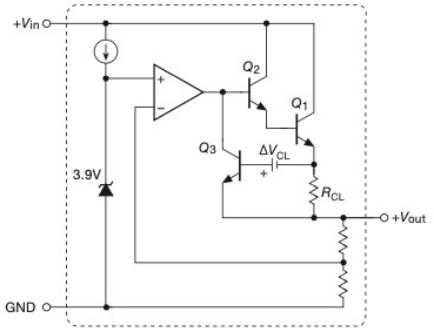


Figure 9.6. Simplified 78xx fixed 3-terminal positive voltage regulator. All components are internal, so only a pair of bypass capacitors is required (as in Figure 9.8). R_{CL}, the current-sensing resistor, is 0.2 Ω, and develops somewhat less than a diode drop at full current; its drop is supplemented by an internal bias ΔV_{CL}, to turn on current-limiting transistor Q_3.

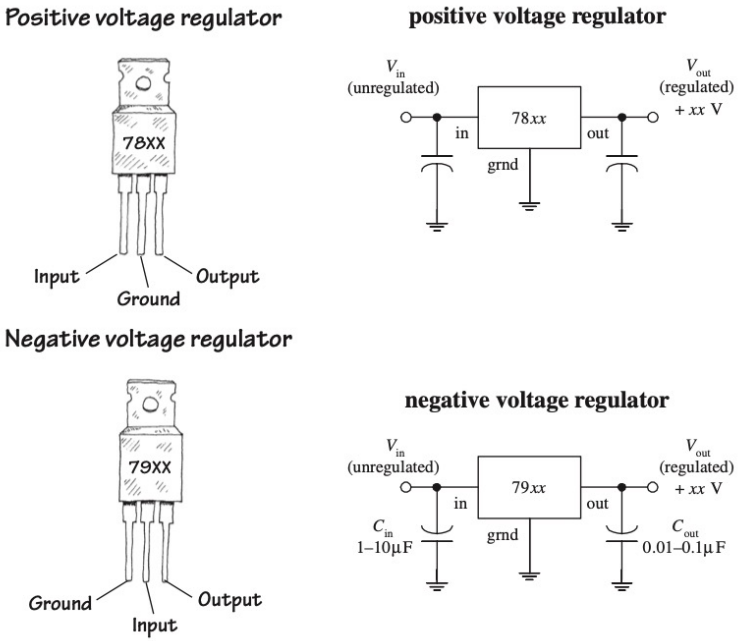
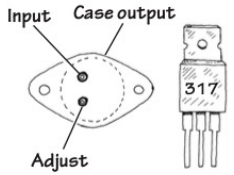
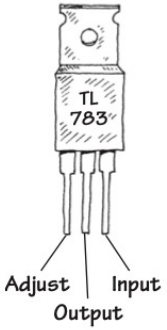


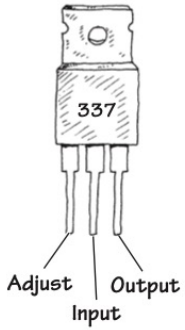
FIGURE 11.4

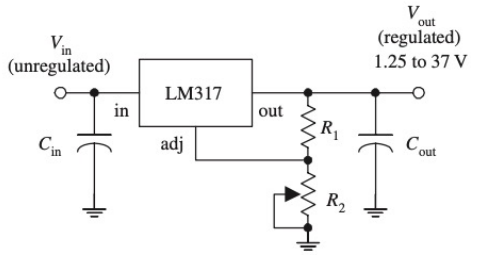
Ref: PEI

Adjustable regulators

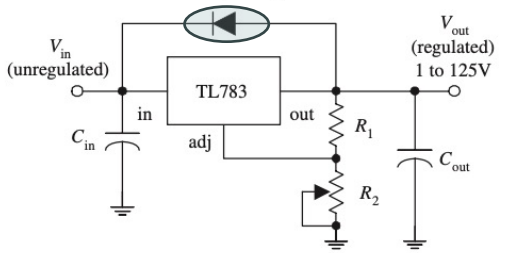


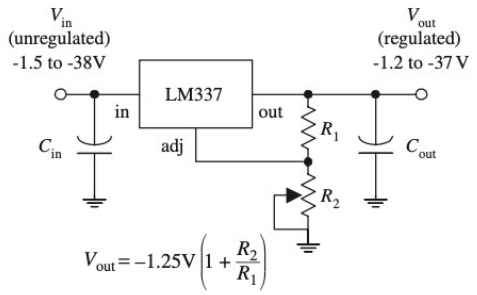






$V_{out} = 1.25V\left(1 + \frac{R_2}{R_1}\right)$ Reverse Protection Diode





$V_{out} = -1.25V\left(1 + \frac{R_2}{R_1}\right)$

Ref: PEI

Specifications

- Output voltage (5V)
- Accuracy (10%)
- Min and max input voltage (7-12V)
- Ripple rejection
- Temperature stability $(\Delta V_{\text{out}}/\Delta T)$
- Output impedance
- Also note,
  - Linear regulators have dropouts (~2V drop between input to output -> Use LDO for lower dropouts, e.g., LM2940)

Expected Load = 100mA – 2 A

Ref: PEI

Metrics

$$\text{Load Regulation}(\%) = \frac{\text{Change in Output Voltage from No-Load to Full-Load}}{\text{Full-Load Output Voltage}}$$

$$\text{Line Regulation}(\%) = \frac{\text{Change in Output Voltage for a Given Change in Input}}{\text{Given Change in Input Voltage}}$$

Ripple
Dropout voltage
Power Supply Rejection Ratio

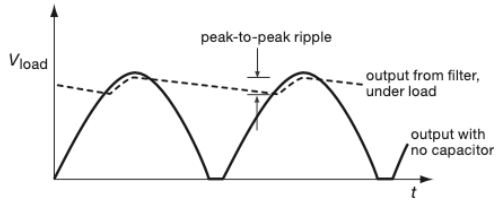


Figure 1.61. Power-supply ripple calculation.

$$\text{PSRR (dB)} = 20 \times \log_{10}\left(\frac{\text{Change in Input Voltage}}{\text{Change in Output Voltage due to the Input Voltage Change}}\right)$$

Heat Sink

- Degrees per watt

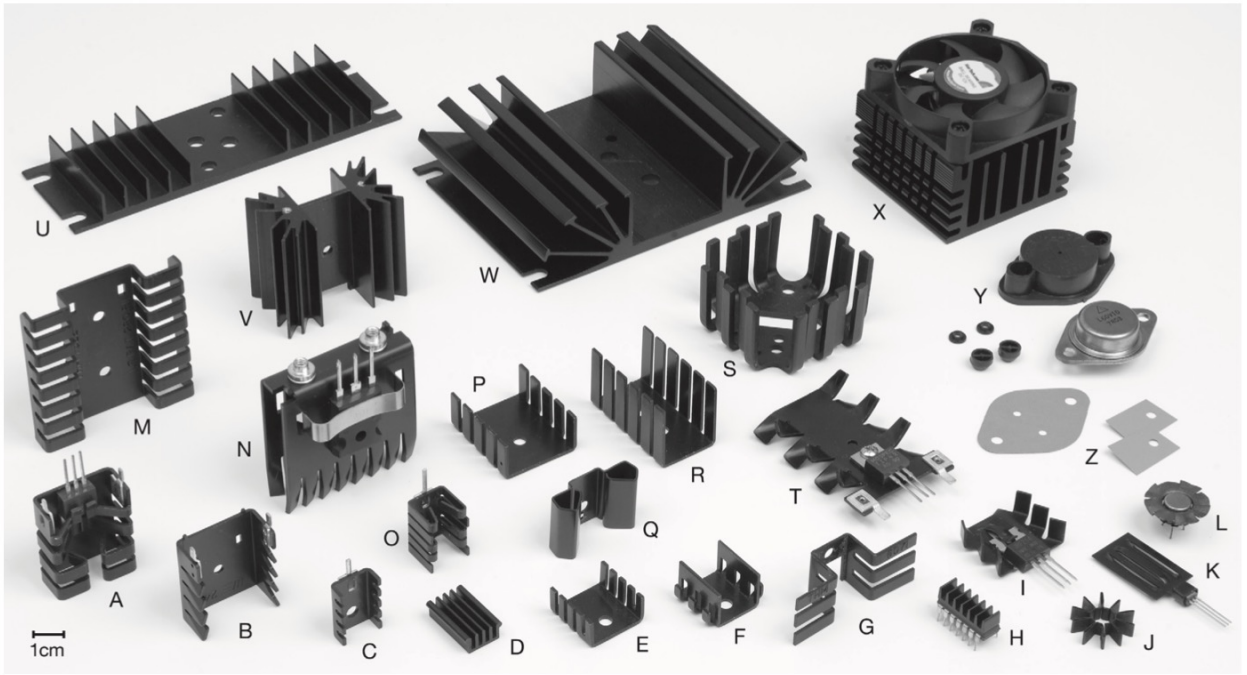
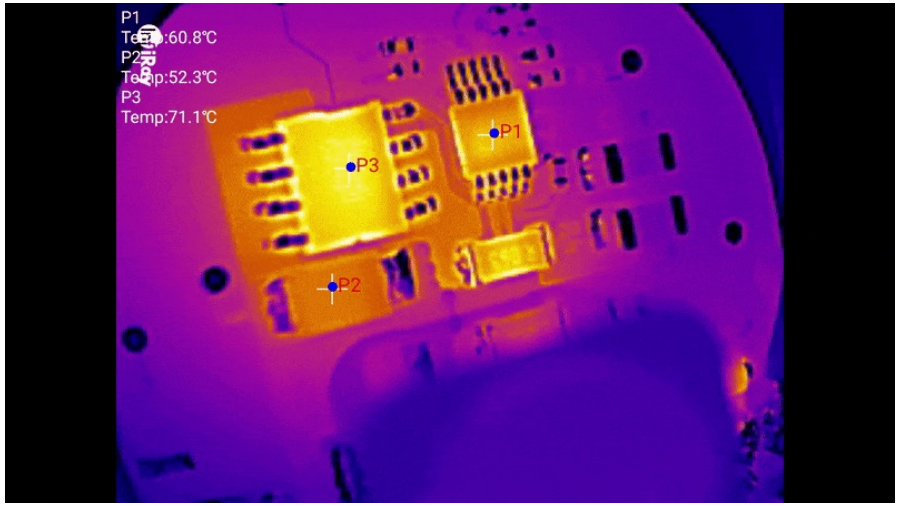
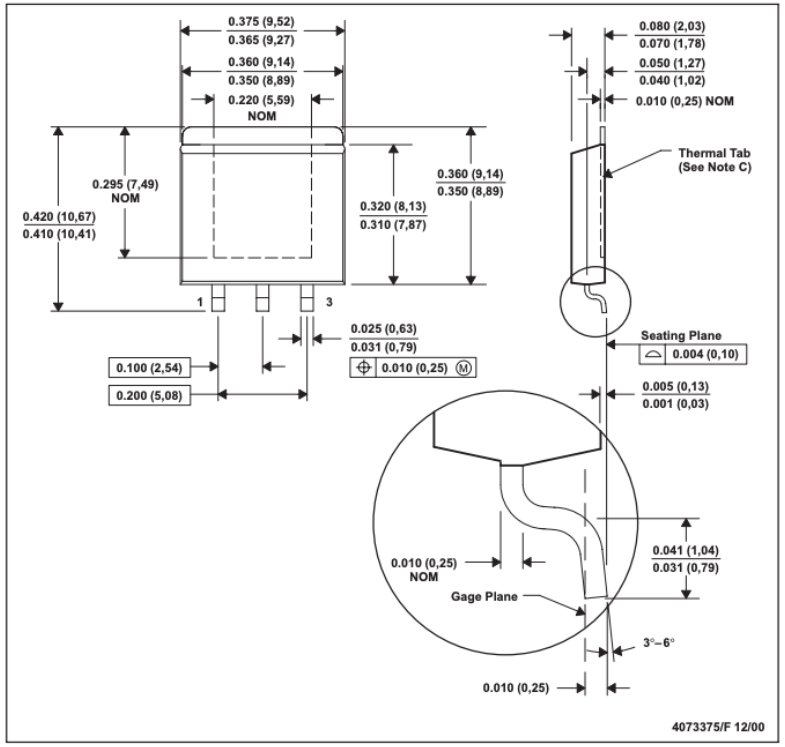


Figure 9.42. Heatsinks come in an impressive diversity, from little clip-on fins (I–L), to mid-sized PCB-mounting types (A–C, N, O, T), to large bolt-down units (U, W), to forced-air type used with microprocessors (X). The corresponding thermal resistance from sink to ambient, $R_{\Theta SA}$, ranges from about 50°C/W down to about 1.5°C/W. A TO-3 insulating cover is shown in (Y), along with shoulder washers and hole plugs; greaseless thermal insulating pads are shown in (Z). We've added alphabetic labels so readers can identify objects of interest when chatting on social media.

Ref: AOE

Ground plane as heat sink





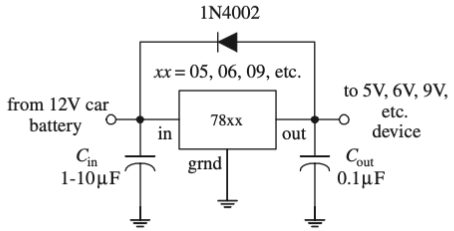
NOTES: A. All linear dimensions are in inches (millimeters).

C. The center lead is in electrical contact with the thermal tab.

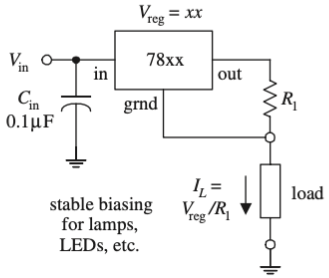
E. Falls within JEDEC MO-169

Common Applications

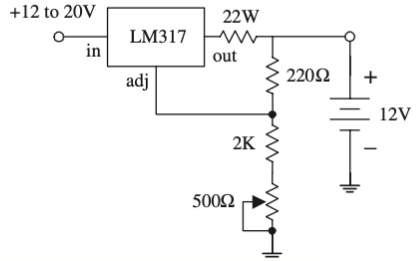
Car battery voltage regulation



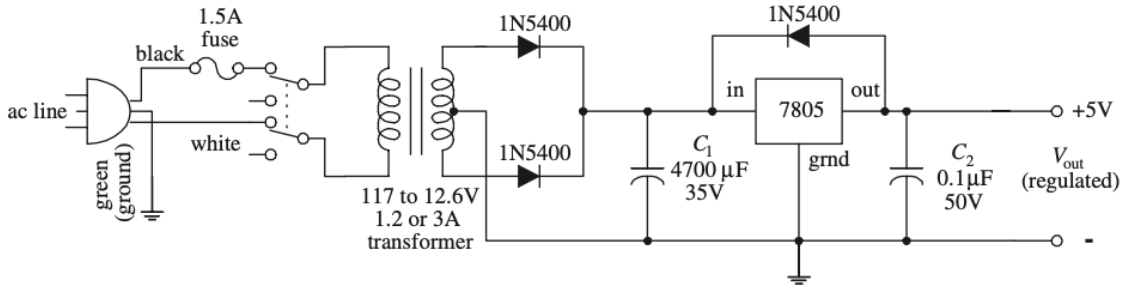
Current regulator

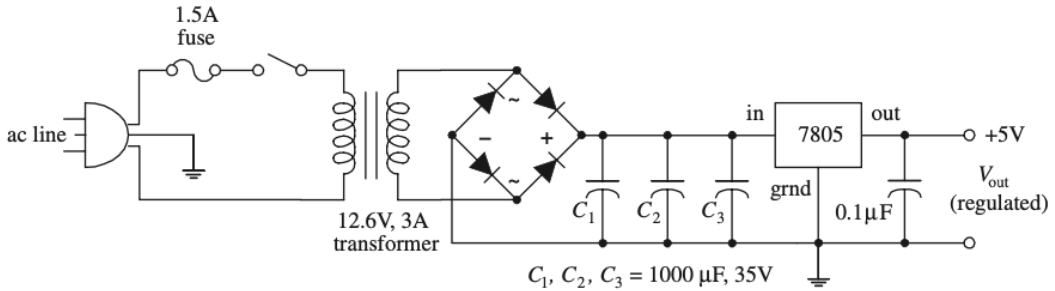


12V battery recharger



Regulated +5-V Supplies





Lower Drop Out

- 5V -> 3.3V -> 2.5V [Multiple power rails]
- Operating from a (declining) battery?

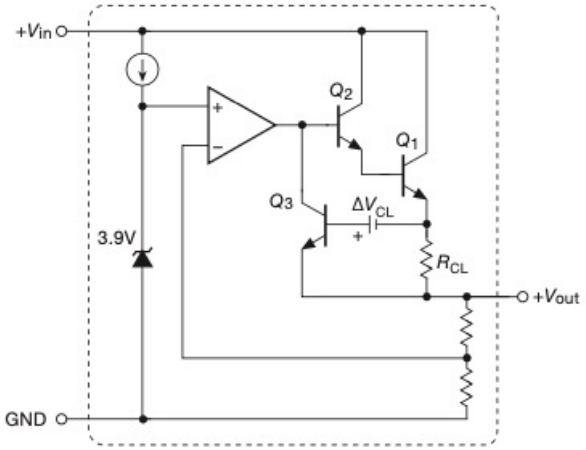


Figure 9.6. Simplified 78xx fixed 3-terminal positive voltage regulator. All components are internal, so only a pair of bypass capacitors is required (as in Figure 9.8). R_{CL}, the current-sensing resistor, is 0.2 Ω, and develops somewhat less than a diode drop at full current; its drop is supplemented by an internal bias ΔV_{CL}, to turn on current-limiting transistor Q_3.

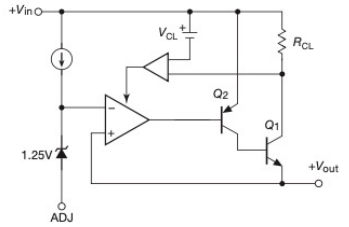
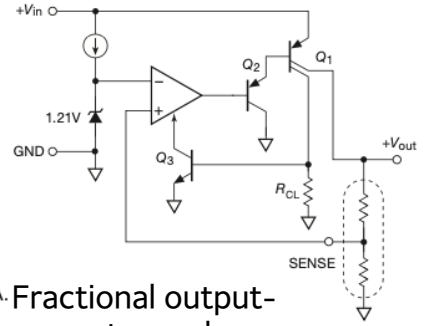


Figure 9.19. LT1083-85 series three-terminal positive regulators with reduced dropout voltage.



A. Fractional output-current sample

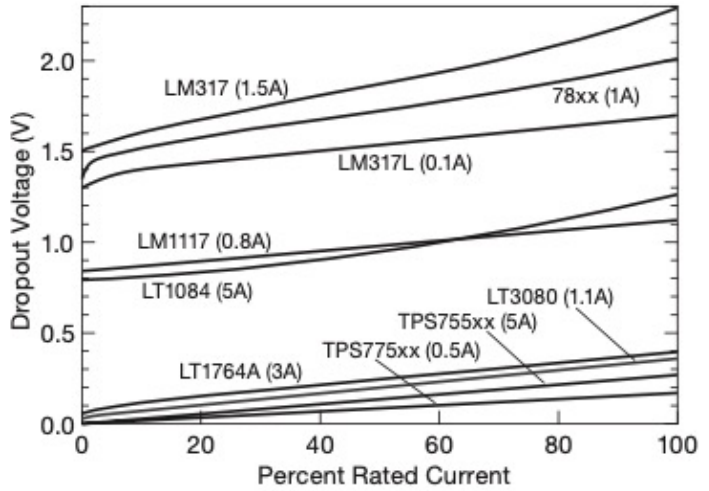


Figure 9.24. Linear regulator dropout voltage versus output current. The bottom pair of curves (TPS prefix) are CMOS; all others are bipolar. See also Figure 9.11.

The common problem?

- Energy dissipation in the pass transistors
  - Continuously running currents
  - Low efficiency
  - High heat generation

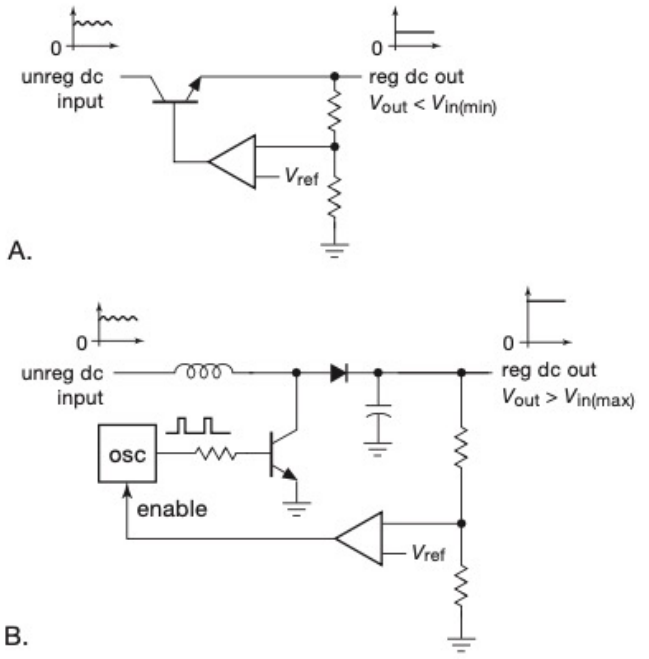


Figure 9.55. Two kinds of regulators: A. linear (series-pass); B. switcher (step-up, or "boost").

Switchers

Pros:
- Compact size
  - Small capacitor & transistor sizes
- Lower power consumptions
  - Saturation or cutoff operation
- No dropouts

The cons:
- Noise

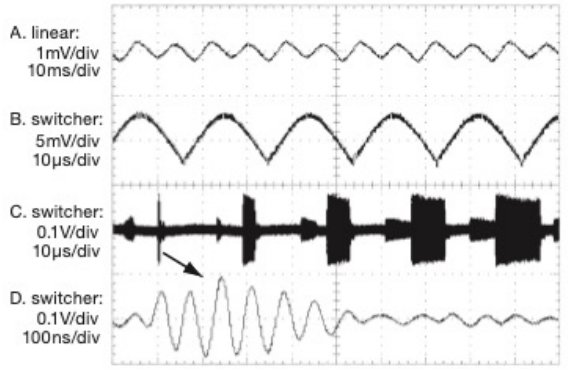


Figure 9.53. Comparing linear and switching power supply noise. All measurements are into a resistive load at 50% of rated current. A. Linear 5 V, 0.3 A supply, showing ~0.5 mVpp 120 Hz ripple. B. Switching 5 V, 2.5 A supply, measured directly across the output pins, showing ~6 mVpp ripple at the 50 kHz switching frequency (note scale change). C. Same switcher, but measured at a connected load 50 cm away (and with another factor of ×20 scale change), showing the large (~150 mV) switching spikes induced by high-frequency ground currents; note the frequency dithering seen in this persistent capture. D. Expanded trace of a single induced pulse, showing ringing at ~15 MHz.

- Reliability

Charge Pumps

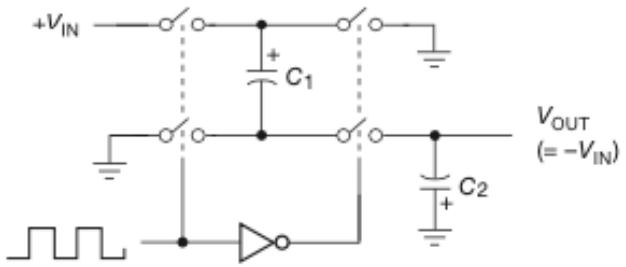


Figure 9.56. Charge-pump voltage inverter. An oscillator operates the switch pairs in alternation: the left-hand switches charge "flying capacitor" C_1 to a voltage of V_{IN}; the right-hand switches then apply that voltage, with reversed polarity, to the output storage capacitor C_2.

Voltage Inverter

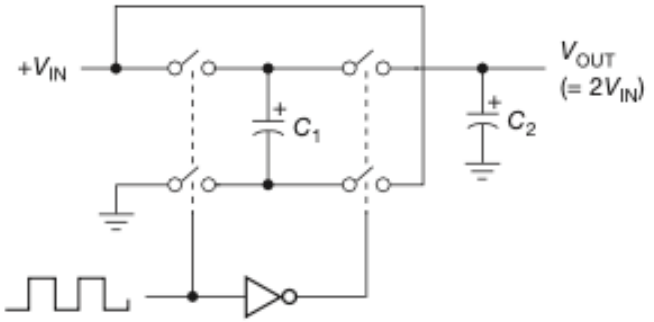


Figure 9.57. Charge-pump voltage doubler. Here the voltage on the flying capacitor, charged to V_{IN}, is added to the input voltage to generate an output voltage of twice V_{IN}.

Voltage Doubler

Sec 9.6.3 (AOE 3rd ed)

Switchers with inductors

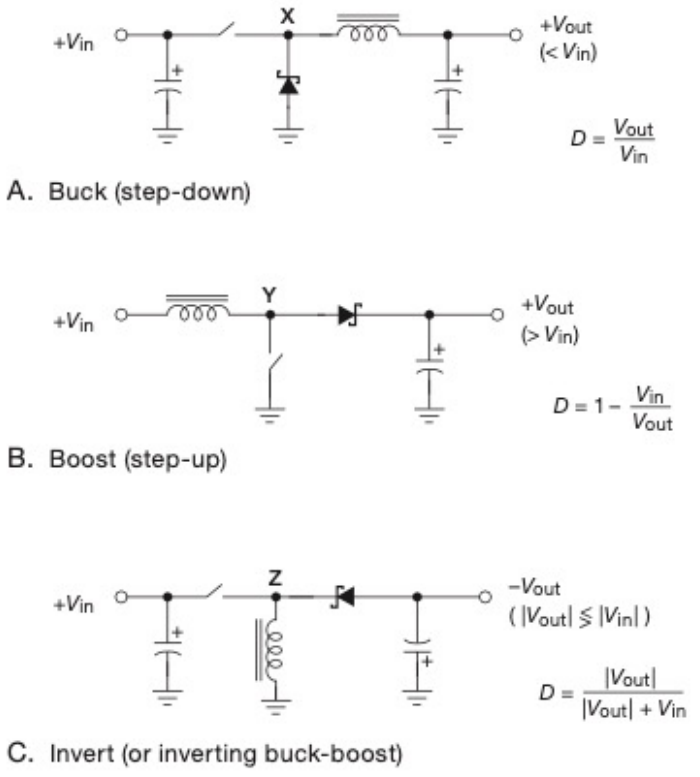


Figure 9.61. The basic nonisolated switching converters. The switch is usually a MOSFET. Schottky diodes are commonly used for the rectifiers, as shown; however, a MOSFET can be used as an efficient synchronously switched "active rectifier."

[Feedback eliminated for simplicity]

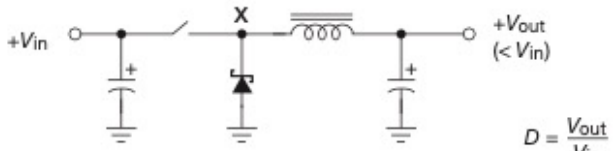
Oscillation Cycles

1. Charge the inductor (increase current)
2. Let the inductor charge the capacitor

$$dI/dt = V/L$$

$$(V_{in} - V_{out})\, t_{on} = V_{out} t_{off},$$

Buck Converter



A. Buck (step-down)

$$dI/dt = V/L$$

$$(V_{in} - V_{out})\, t_{on} = V_{out} t_{off},$$

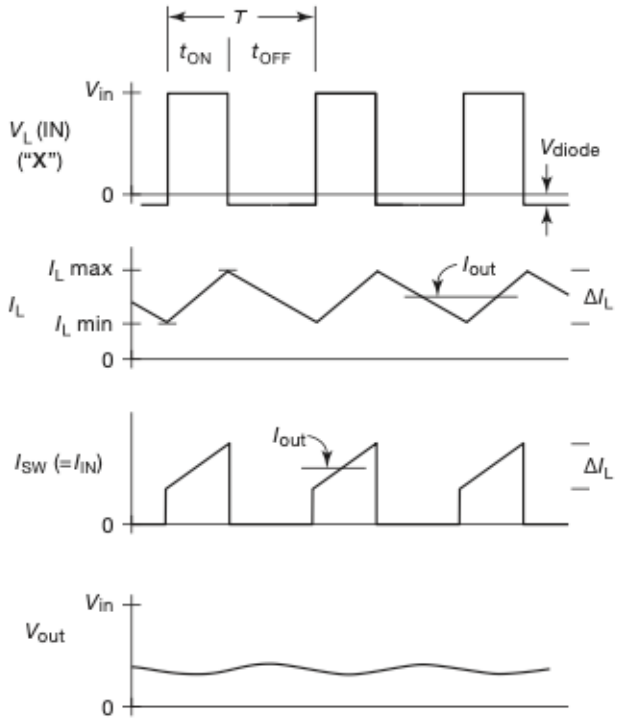


Figure 9.62. Buck converter operation. Inductor current ramps up during switch ON, and ramps down during switch OFF. The output voltage equals the input voltage times the duty cycle ($D \equiv t_{on}/T$). In the case of continuous inductor current (CCM; as shown here) the output current is equal to the average inductor current.

https://everycircuit.com/circuit/6494788991582208

Boost Converter

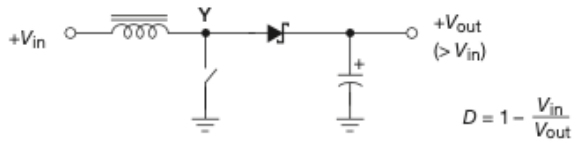


Figure 9.66. Basic boost (or "step-up") topology (non-isolated).

$$\langle I_{in} \rangle = I_{out} \frac{V_{out}}{V_{in}} = \frac{I_{out}}{1-D}, \tag{9.4a}$$

$$\Delta I_{in} = \frac{T}{L} V_{in} D, \tag{9.4b}$$

$$V_{out} = V_{in} \frac{T}{t_{off}} = \frac{V_{in}}{1-D}, \tag{9.4c}$$

$$D = 1 - \frac{V_{in}}{V_{out}}, \tag{9.4d}$$

$$I_{out(min)} = \frac{T}{2L} \left(\frac{V_{in}}{V_{out}} \right)^2 (V_{out} - V_{in}),$$

$$= \frac{T}{2L} V_{out} D (1-D)^2, \tag{9.4e}$$

$$\Delta I_{C(out)} = \frac{I_{out}}{1-D}, \tag{9.4f}$$

$$I_{L(pk)} = \frac{I_{out}}{1-D} + \frac{T}{2L} V_{in} D, \tag{9.4g}$$

$$L_{min} = \frac{T}{2I_{out}} \left(\frac{V_{in}}{V_{out}} \right)^2 (V_{out} - V_{in}). \tag{9.4h}$$

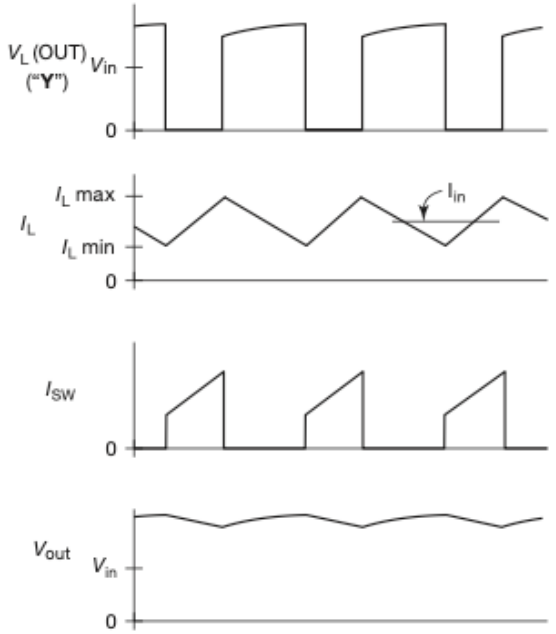
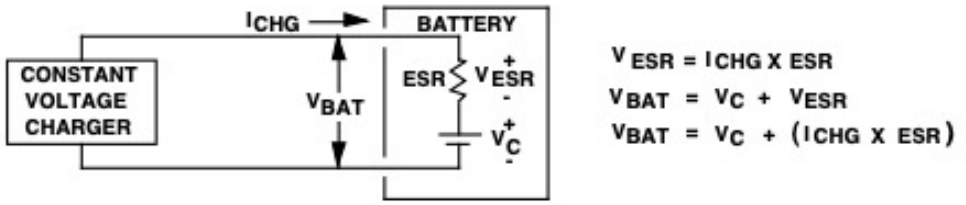


Figure 9.67. Boost converter operation. Inductor current ramps up during switch ON, and ramps down during switch OFF. The output voltage equals the input voltage divided by the fraction of the time the switch is OFF. In the case of continuous inductor current (CCM, as shown here) the input current is equal to the average inductor current.

https://everycircuit.com/circuit/5152487425966080

Battery Chargers



$V_{ESR} = I_{CHG} \times ESR$

$V_{BAT} = V_C + V_{ESR}$

$V_{BAT} = V_C + (I_{CHG} \times ESR)$

Constant Current (CC) Constant Voltage (CV) Charging
: CC-CV charging

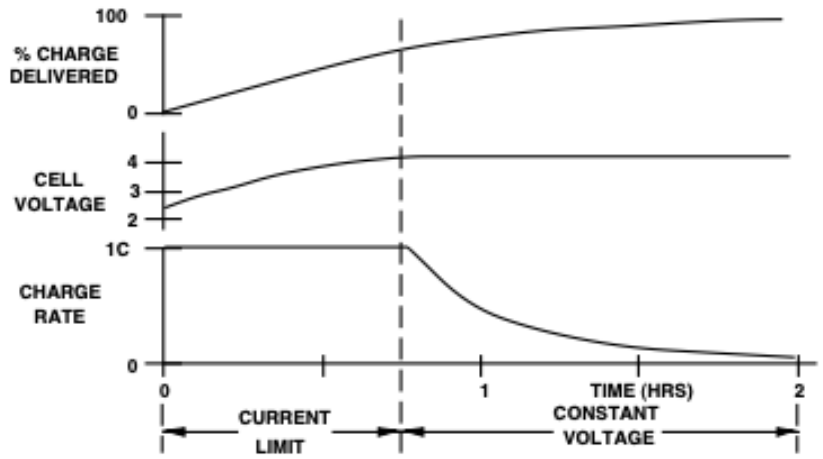


FIGURE 5. TYPICAL C-V CHARGE PROFILE

Typical set point voltage = 4.2 +/- 50mV
1C charging rate = charging current equal to A-hr rating of the cell

Isn't the battery charged at the end of current limit phase? What's happening in the CV phase?

The current is reducing and hence the entire 4.2V is from the charge stored and not the drop across ESR (Equivalent Series Resistance)

Ref: TI Note

LP2951

Advanced Monolithic Systems

LP2950/LP2951

100mA LOW DROPOUT VOLTAGE REGULATOR

RoHS compliant

FEATURES

- 5V, 3.3V, and 3V Versions at 100mA Output Current
- High Accuracy Output Voltage
- Extremely Low Quiescent Current
- Low Dropout Voltage
- Extremely Tight Load and Line Regulation
- Very Low Temperature Coefficient
- Current and Thermal Limiting
- Needs Minimum Capacitance (1µF) for Stability
- Unregulated DC Positive Transients 60V

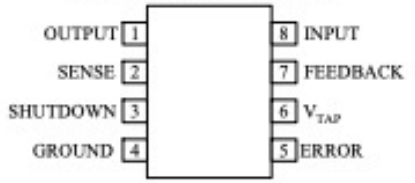
ADDITIONAL FEATURES (LP2951 ONLY)

- 1.24V to 29V Programmable Output
- Error Flag Warning of Voltage Output Dropout
- Logic Controlled Electronic Shutdown

APPLICATIONS

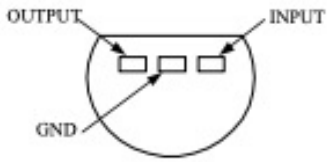
- Battery Powered Systems
- Portable Consumer Equipment
- Cordless Telephones
- Portable (Notebook) Computers
- Portable Instrumentation
- Radio Control Systems
- Automotive Electronics
- Avionics
- Low-Power Voltage Reference

8L SOIC/ 8L PDIP

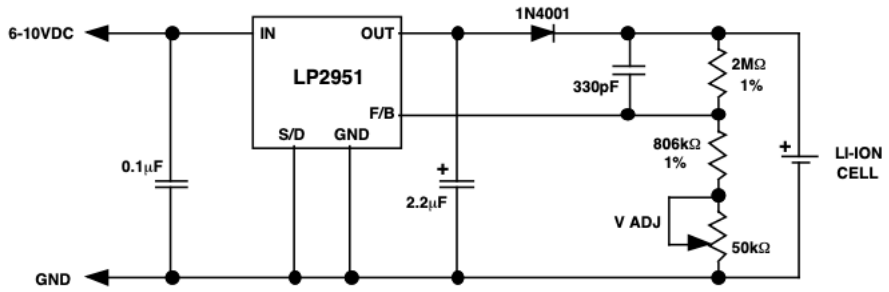


Top View

TO-92



Bottom View



Other options
LP2952 (higher current)
LM317 (cheaper)

Bonus: Magic Circuit (Joule Thief)

https://everycircuit.com/circuit/5969382857244672

More common ICs

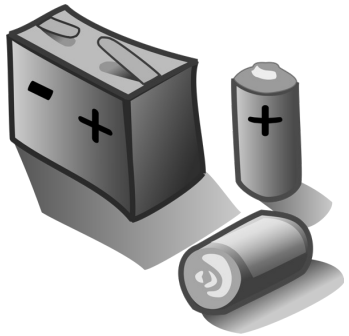
Ch 17, Practical Electronics for Inventors

- Audio
- Power
- RF
- Medical

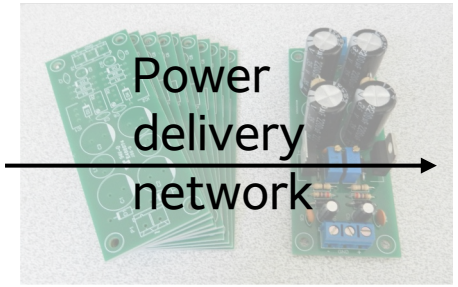
Tool Installation

PlatformIO for AVR controllers

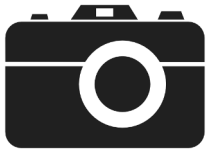
Make PCBs not War



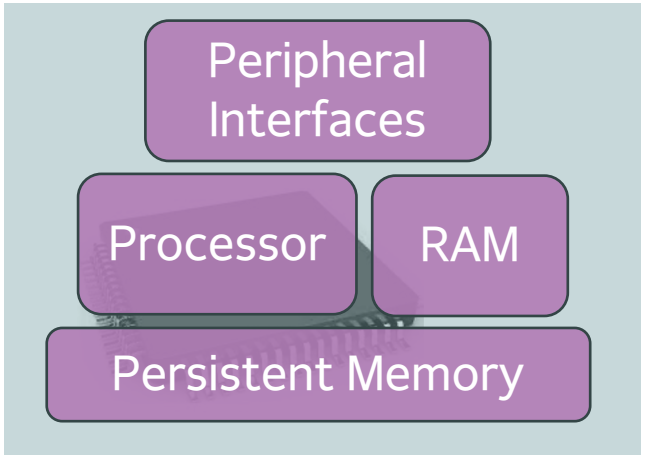
Reliable **power** source



Peripherals



Communication



Compute

Basic tutorial: tech.guitarsite.de/pcb_layout

What's a PDN?

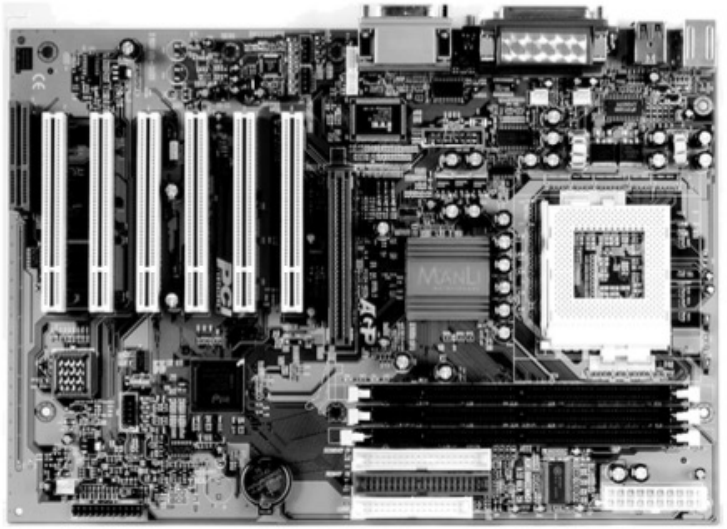


Figure 1.1 A typical computer motherboard with multiple VRMs and active devices. The PDN includes all the interconnects from the pads of the VRMs to the circuits on the die.

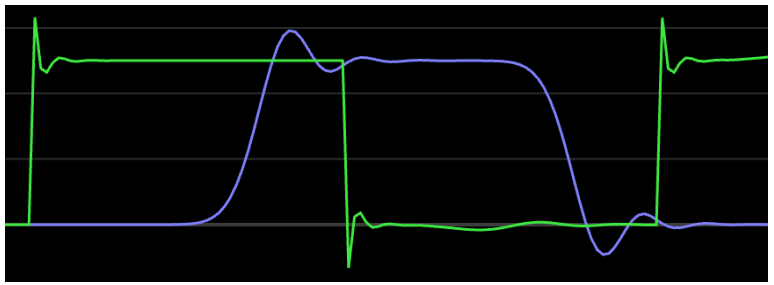
The purpose of the PDN is to

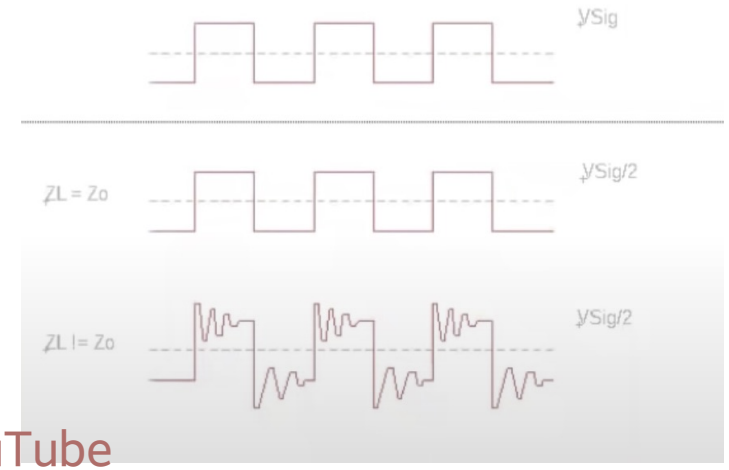
- Distribute low-noise DC voltage and power to the active devices doing all the work.
- Provide a low-noise return path for all the signals.
- Mitigate electromagnetic interference (EMI) problems without contributing to radiated emissions.

Why is this difficult?

Because a wire is not always a wire. It’s a transmission line..

Impedance discontinuities lead to signal reflections -> Degraded signal quality



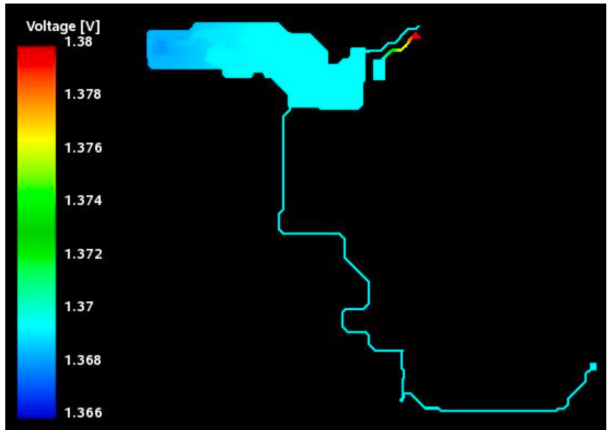


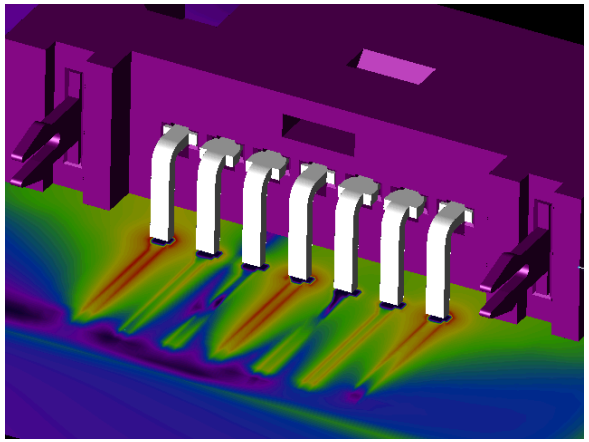
https://everycircuit.com/circuit/5698102557933568
Introduction to Signal Integrity for PCB Design - YouTube

Why does it matter?

- Setup and Hold Time Requirements
- Signal to noise ratio
- When does it not matter?
  - Low frequency signals (rise length $l_R = t_R V < 10$*trace length)
  - Or very short connections [a small PCB is (almost) always a better PCB]

In action





Avoid cross-overs

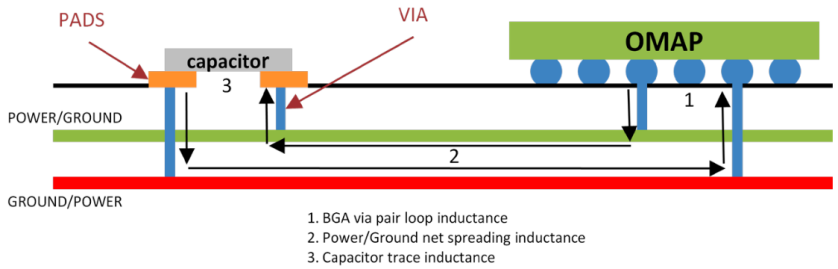
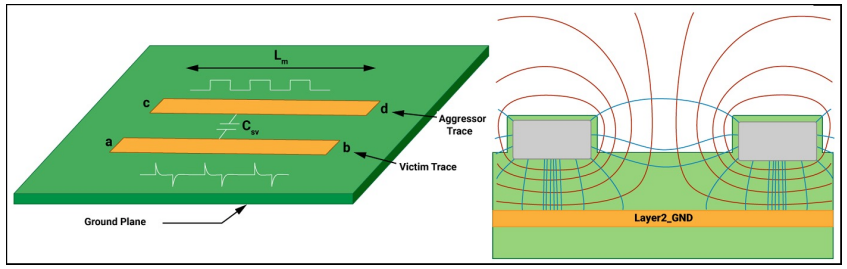


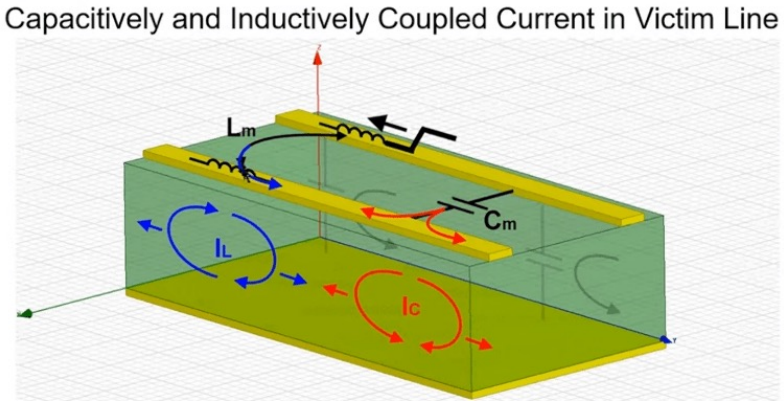
Figure 6: Loop inductance principle

Power Delivery Network Analysis (Rev. A) (ti.com)

Crosstalk

- Coupling
  - Capacitive: Two conductors induce charge in each other
  - Inductive: inducing current in a nearby conductor
  - Radiative: Track acting as an antenna
  - Common impedance (e.g., ground plane): Return paths interfering with other return paths
- Avoiding crosstalk
  - Increase distance between wires
  - Use differential signaling
  - Route traces at right angles (wrt other traces)
  - Use termination
  - Keep analog/digital return paths separate

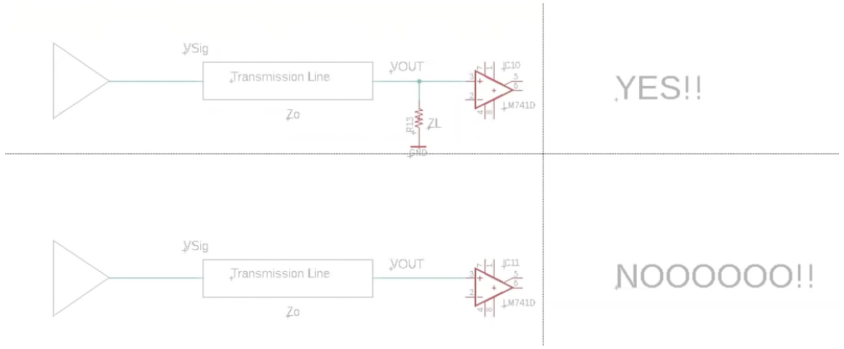


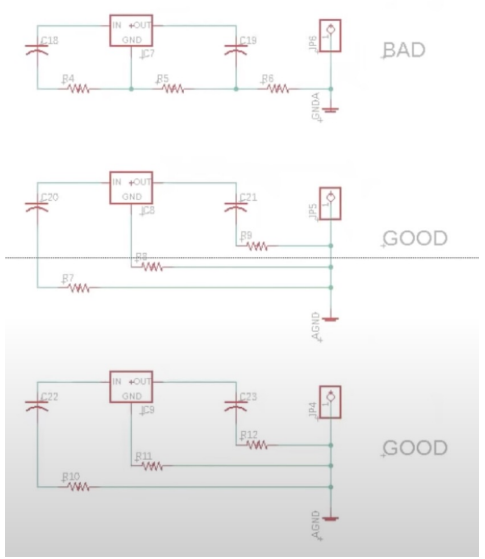


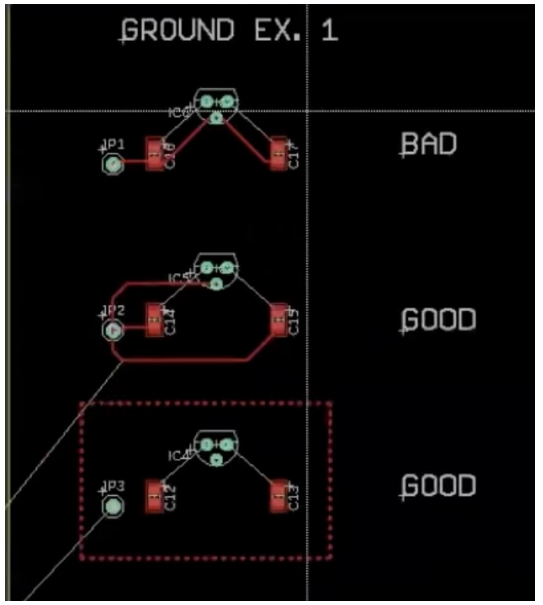
How can you handle it?

- Continuous impedance
  - Stick to 45 degree routes
    - continuous impedance
    - less chance of inductive coupling
    - better for etching/flex
  - Proper terminations into components
- Symmetric grounding
  - Ground planes

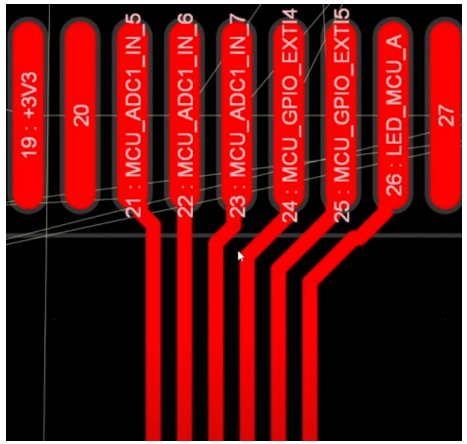
[Introduction to Signal Integrity for PCB Design - YouTube]

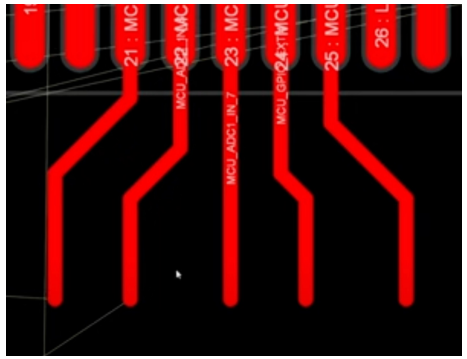




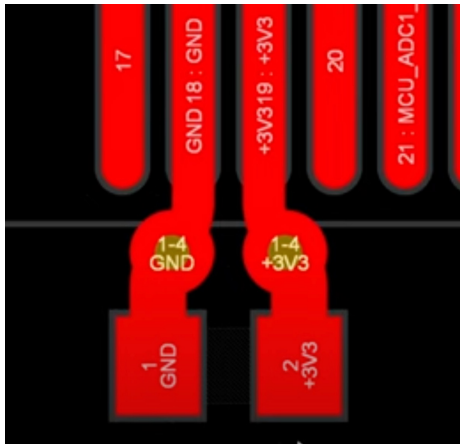


How can you handle it?

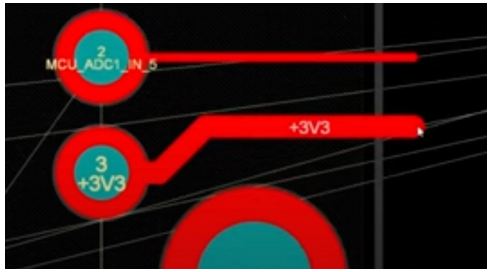




Space Things Out

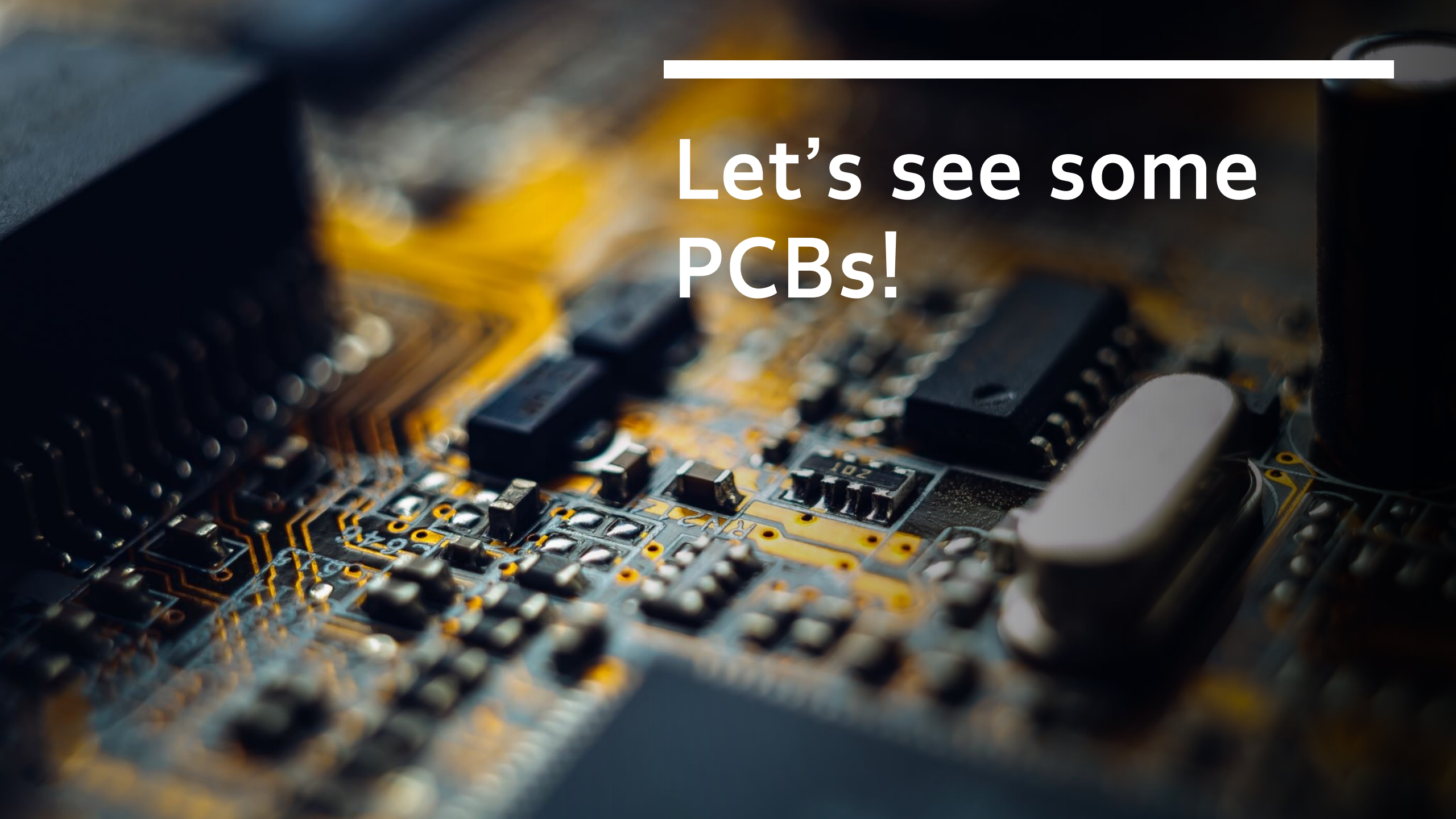


Use decoupling capacitors



Use different track widths for signal/power

[Top 5 Beginner PCB Design Mistakes (and how to fix them) - YouTube]



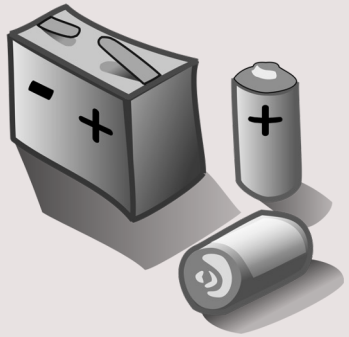
Let’s see some PCBs!

Arduino Uno Rev3

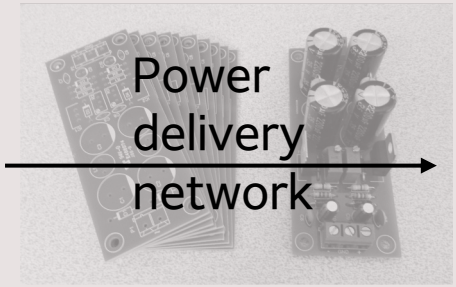
Observe:

- Voltage Regulation
  - Ferrite beads
  - Fuses
- Decoupling Capacitors
- Perpendicular lines in the two planes
- Ground Plane
- Different mill widths in analog and digital paths
- ICSP – In Circuit Serial Programming Interface
- USB to Serial conversion with Atmega16U2

The building blocks

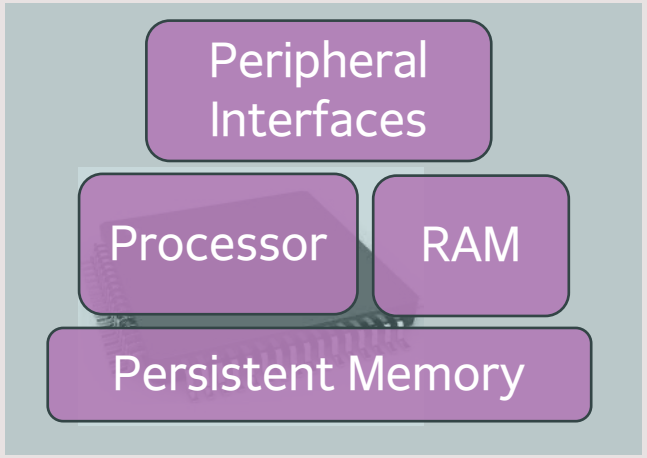


Reliable **power** source



Peripherals

Communication



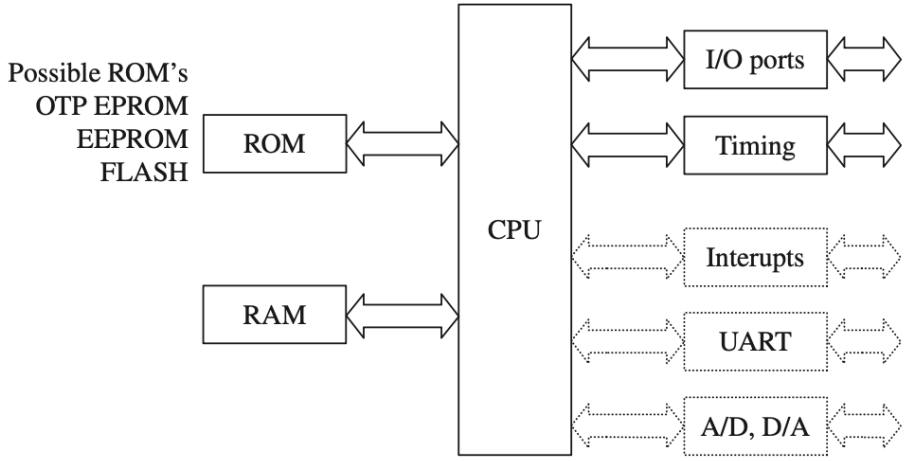
Compute

Microcontrollers

- Embedded into a device that's certainly not a "computer"
  - No disk to boot from
  - On-chip compute, memory – both RAM and NVM (Flash) + ADC/DAC + onboard debug and programming interfaces
  - Opamp – the universal analog component | μC – the universal digital component

Ref: AOE Ch 15

Anatomy of a μC



Possible internal architectures: RISC, SISC, CISC, Harvard, Von-Neuman

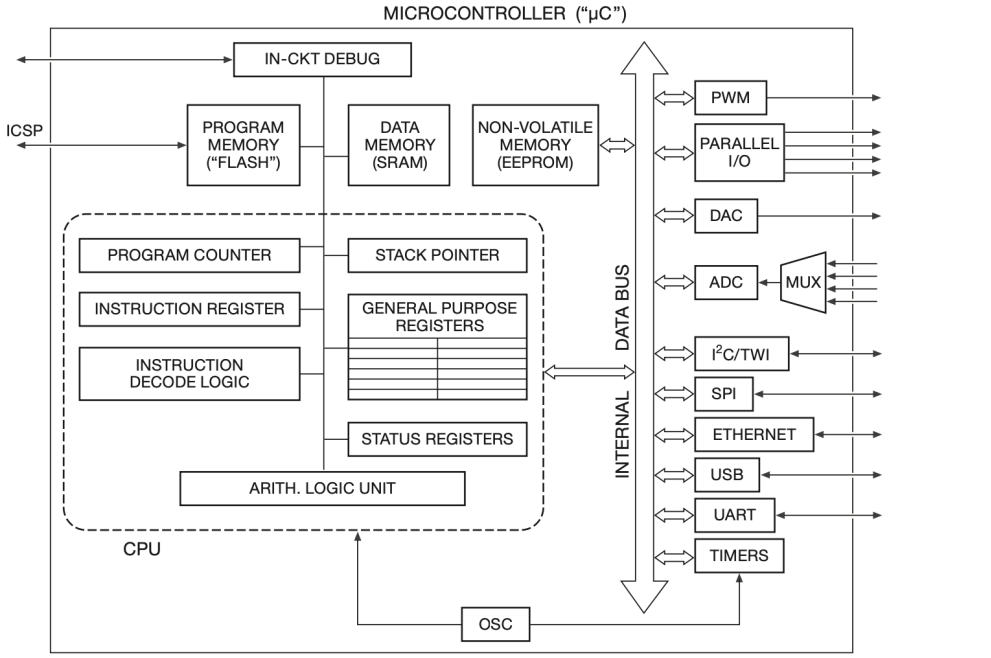
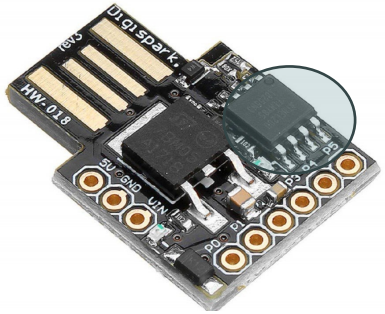


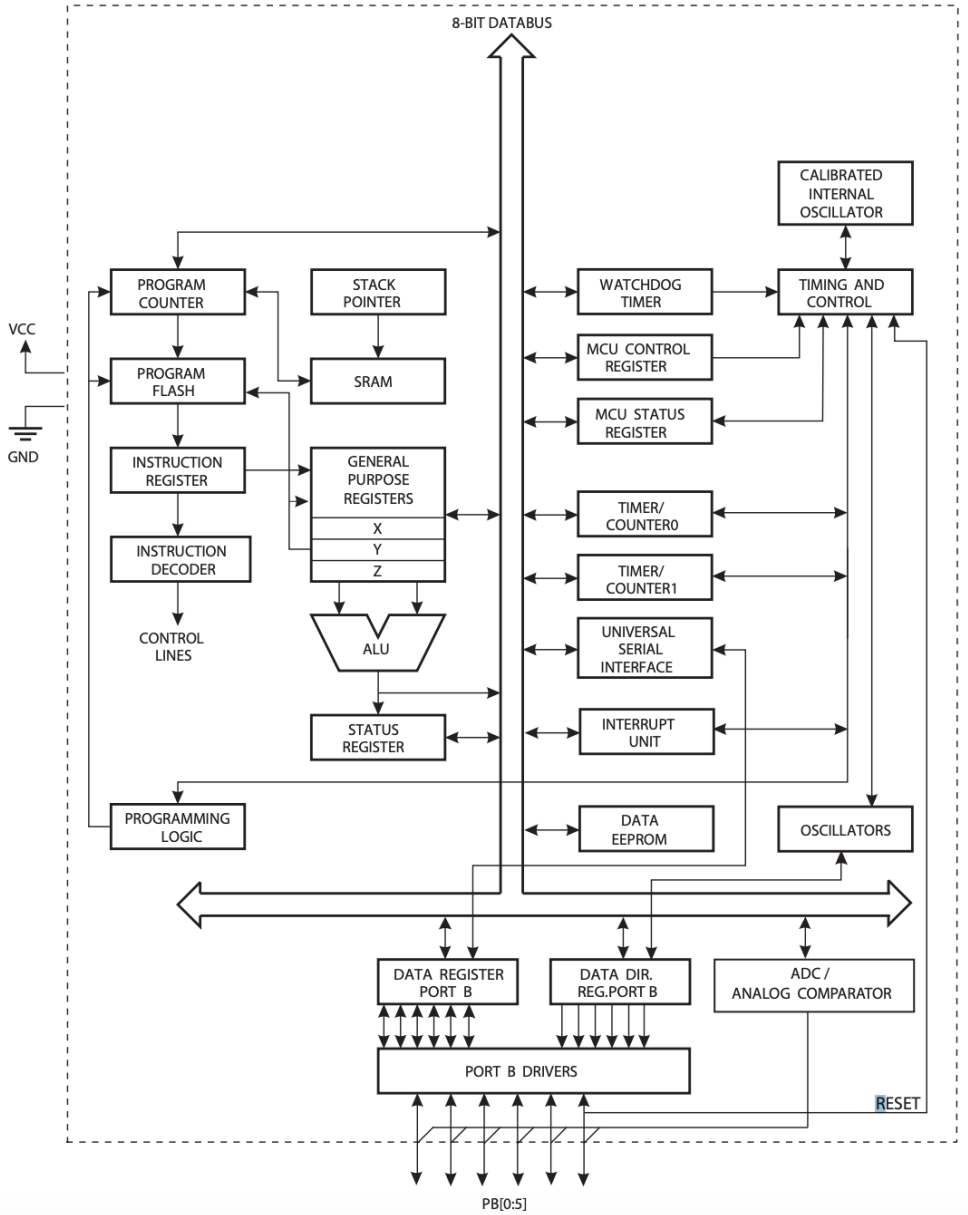
Figure 15.1. A microcontroller integrates memory and "peripherals" onto the same chip as the CPU. ICSP stands for "in-circuit serial programming."

ATtiny85



- [Datasheet]
  - 8 kB Flash
  - 256 bytes SRAM
  - 512 bytes of EEPROM
- Watchdog Timer (wake me up)
- Internal oscillator/two timer interrupts – do work periodically
- All I/O pins interfaces with ADC
- I2C, Serial interfaces

Figure 2-1. Block Diagram



Where’s my clock?

Figure 6-1. Clock Distribution

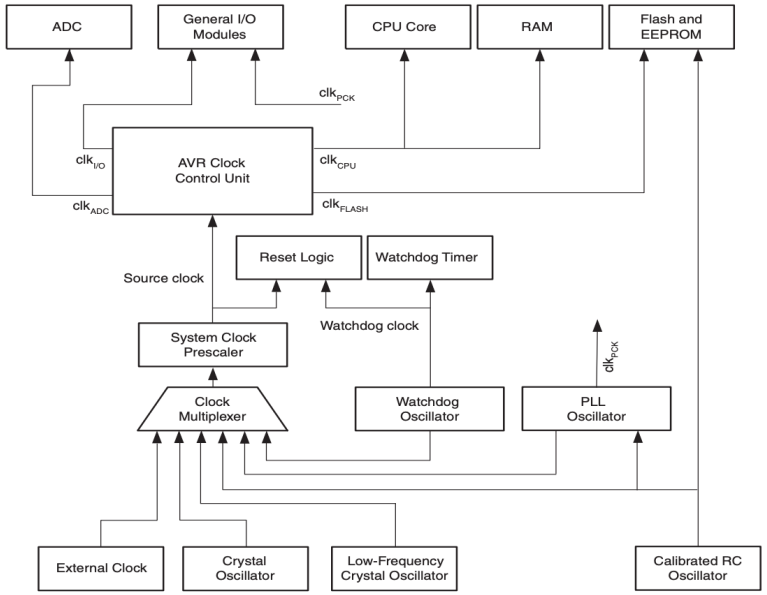


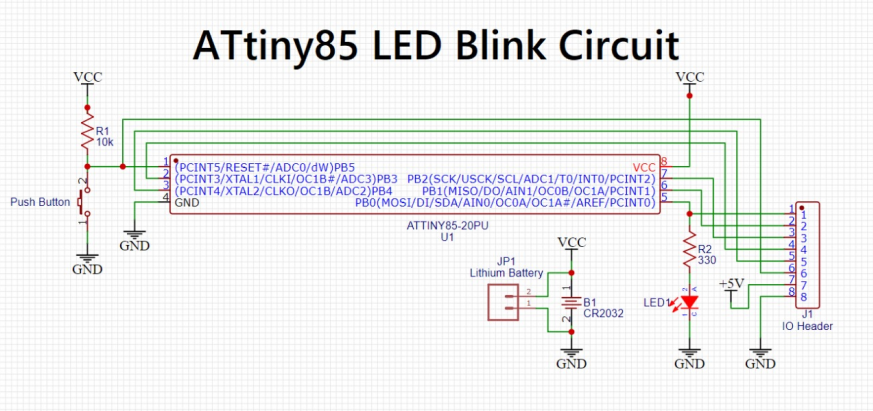
Table 6-1. Device Clocking Options Select

Device Clocking Option	CKSEL[3:0](1)
External Clock (see page 26)	0000
High Frequency PLL Clock (see page 26)	0001
Calibrated Internal Oscillator (see page 27)	0010(2)
Calibrated Internal Oscillator (see page 27)	0011(3)
Internal 128 kHz Oscillator (see page 28)	0100
Low-Frequency Crystal Oscillator (see page 29)	0110
Crystal Oscillator / Ceramic Resonator (see page 29)	1000 – 1111
Reserved	0101, 0111

Note: Datasheet is God!

Contains details of all registers to be programmed for a certain function as well

Attiny LED Blinker




```
void setup(){
DDRB |= (1 << PB3);                    //replaces pinMode(PB3, OUTPUT);
DDRB |= (1 << PB4);                    //replaces pinMode(PB4, OUTPUT);
}

void loop()
{
 delay(random(600000, 900000));
 byte state = random(0, 2);
 switch(state)
  {
   case 0:
   PORTB |= (1 << PB3);                //replaces digitalWrite(PB3, HIGH);
   delay(20);
   PORTB &= ~(1 << PB3);               //replaces digitalWrite(PB3, LOW);
   break;

   case 1:
   PORTB |= (1 << PB4);                //replaces digitalWrite(PB4, HIGH);
   delay(20);
   PORTB &= ~(1 << PB4);               //replaces digitalWrite(PB4, LOW);
   break;
  }
}
```

[How to Blink an LED with ATtiny85 | ATtiny85 Programming - YouTube]

[ATTiny Port Manipulation (Part 1): PinMode() and DigitalWrite() : 7 Steps - Instructables]

More complex μCs

AVR (mostly single core)

- AC Power Control
- Thermal Controller
- Remote Controller

STM32 (multi-core)

- Drones
- Smartwatches/Airpods/Apple TV
- PLCs
- Medical Monitors/FItbits
- Car infotainment systems
- Synthesizers and MIDI Controllers
- Macbook pro touchbar
- Digital Stethoscopes

DSP

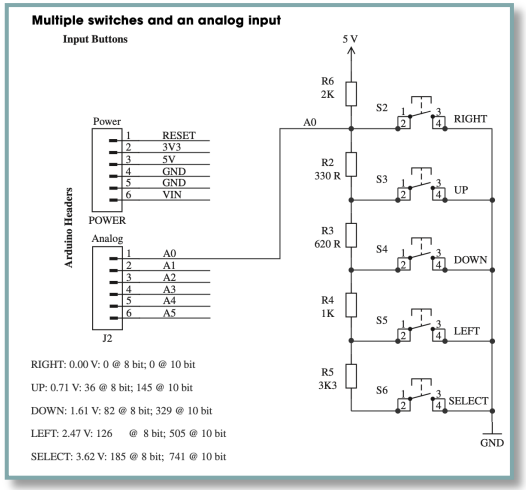
- Audio signal processing, e.g., FFTs
- Audio enhancement/noise reduction, etc.

Notice:
No mention of OS yet

Common interfaces

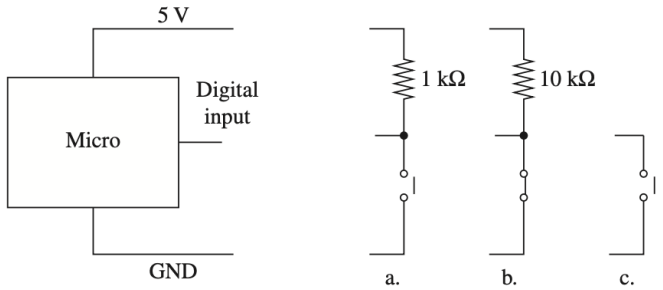
Switch

```
if (digitalRead(4) == LOW)
{
  // the key was pressed, do something
}
```

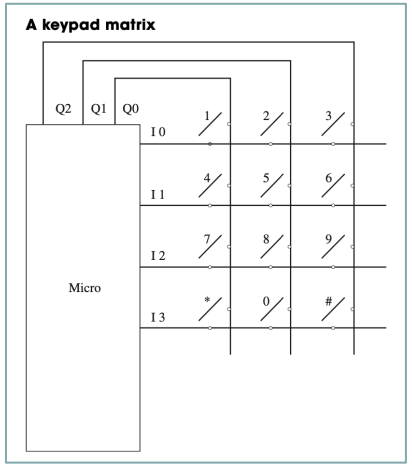


Ref: PEI

Connecting switches to a digital input



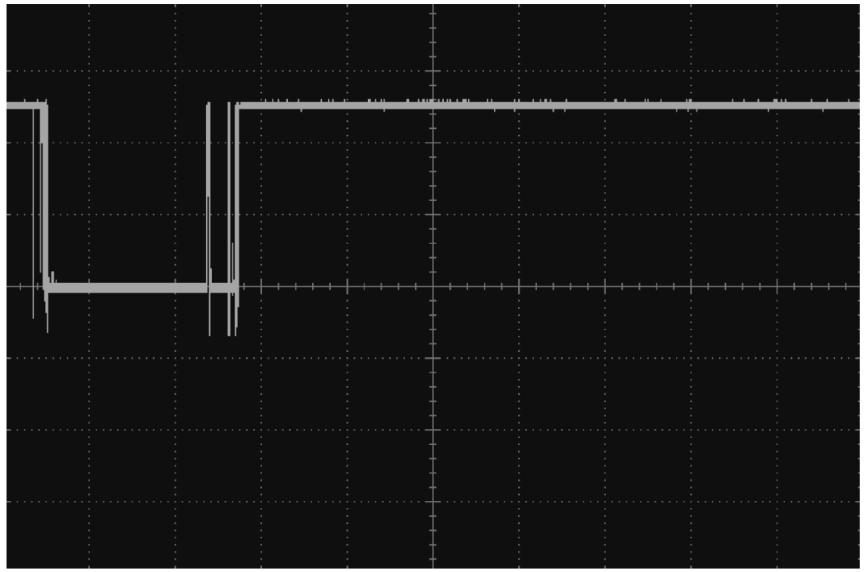
a. N.O.
b. N.C.
c. N.O. internal pullup resistor

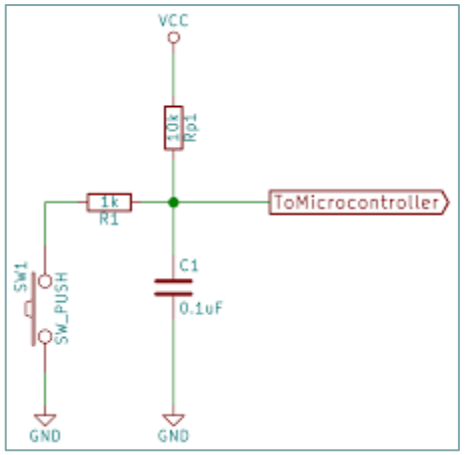


Also, pullup/pulldown

Debounce

Switch bouncing





Hardware Debounce

```
constintdebouncePeriod = 100;
long lastKeyPressTime = 0;

void loop()
{
  long timeNow = millis();
  if (digitalRead(5) == LOW &&lastKeyPressTime>timeNow + debouncePeriod)
  {
    // button pressed and enough time elapsed since last press
    // do what you need to do
    lastKeyPressTime = timeNow;
  }
}
```

Software Debounce

Controlling LEDs

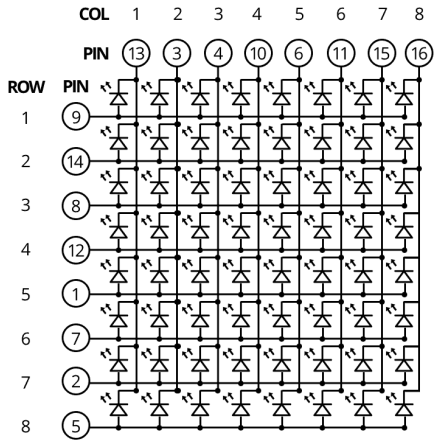
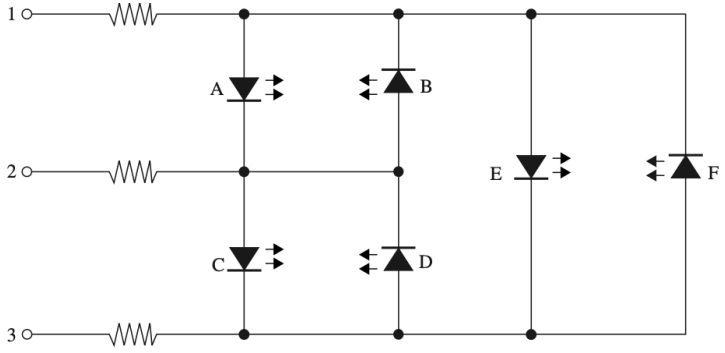


TABLE 13.9 Charlieplexing LED Addressing

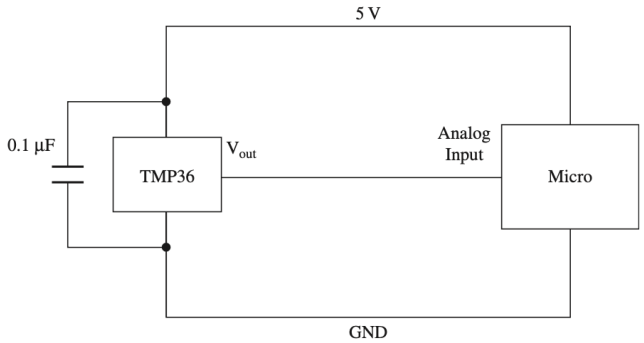
LED	PIN 1	PIN 2	PIN 3
A	High	Low	Input
B	Low	High	Input
C	Input	High	Low
D	Input	Low	High
E	High	Input	Low
F	Low	Input	High

Charlieplexing LEDs



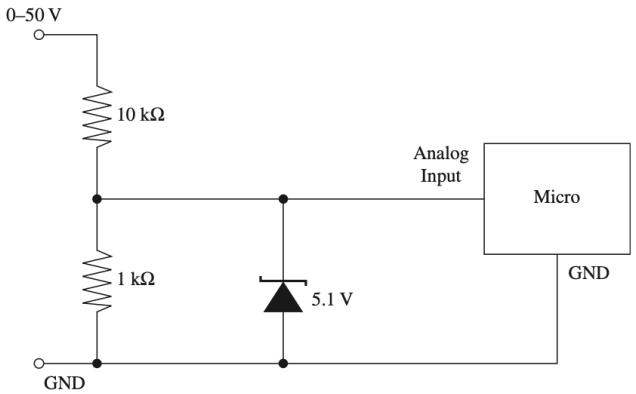
Analog Input

Reading the voltage from a TMP36 sensor



Direct

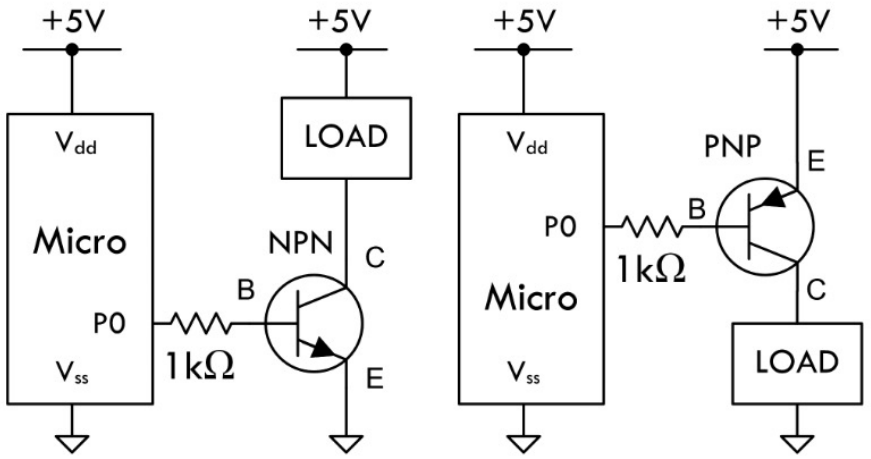
ADC voltage reduction and input protection



Ref: PEI

High power digital outputs

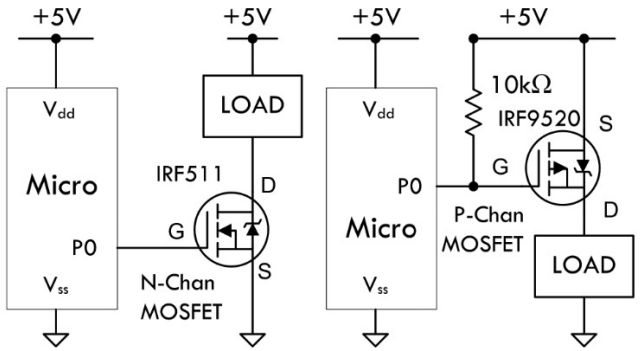
Handling high-power outputs with bipolar transistors



A bipolar transistor can be used to switch on and off loads. In the circuit to the left, when P0 is set HIGH, the NPN transistor is turned on—a low resistance is now present between C and E. In the circuit to the right, when P0 is set LOW, the PNP transistor is turned on. Microcontroller I/O current levels are usually large enough to provide enough base current for a bipolar transistor. Select a bipolar transistor with the appropriate current/voltage ratings for your load.

Typical AVR pin = 40 mA current

Handling high-power outputs with MOSFETs

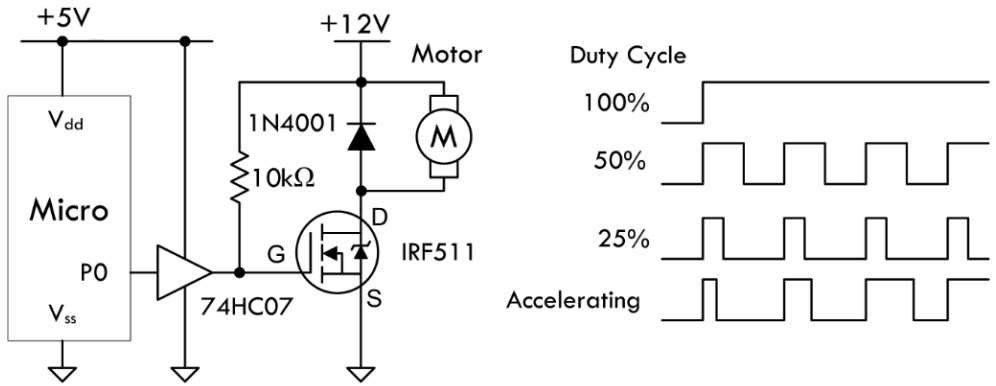


MOSFETs have a much lower on-state resistance (in milliohms) than bipolar transistors (10's to 100's of milliohms). This means that a MOSFET driver will experience a smaller voltage drop, and in general, can handle much larger currents. MOSFETs also have very high input impedances; they draw very little gate current from a microcontroller's I/O pins. Some MOSFETs are capable of handling 60A or more. In the circuit to the left, an N-Chan MOSFET is triggered with a HIGH on P0. In the right circuit, a P-Chan MOSFET is triggered with a LOW on P0. Separate load supplies can be used, and are recommended if there are inductive loads present.

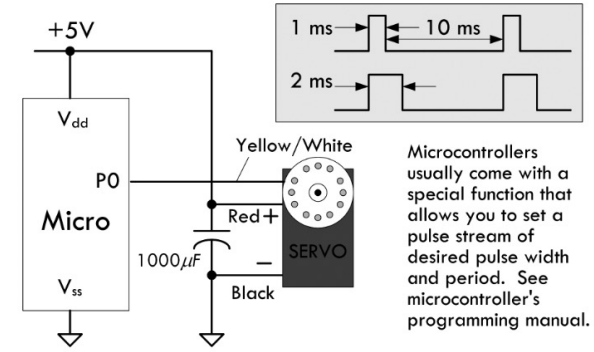
Remember to use diodes with inductive components, e.g., relays

Motors

Controlling a dc motor

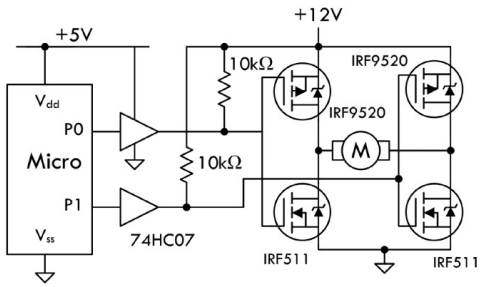


Controlling a servo



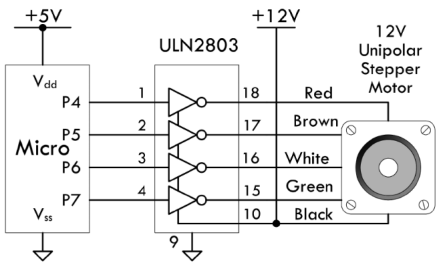
Microcontrollers usually come with a special function that allows you to set a pulse stream of desired pulse width and period. See microcontroller's programming manual.

Bidirectional motor control



The H-bridge circuit built using MOSFETs provides forward and reverse directional control of a dc motor. The H-bridge provides built-in dynamic breaking action useful for applications requiring greater control. To make motor go in one direction, P0 is set HIGH while P1 is set LOW. To switch directions, P0 is set LOW while P1 is set HIGH. Buffer stage (74HC07) could be replaced with a optoisolator to provide greater electrical isolation from motor section of circuit.

Stepper motor control

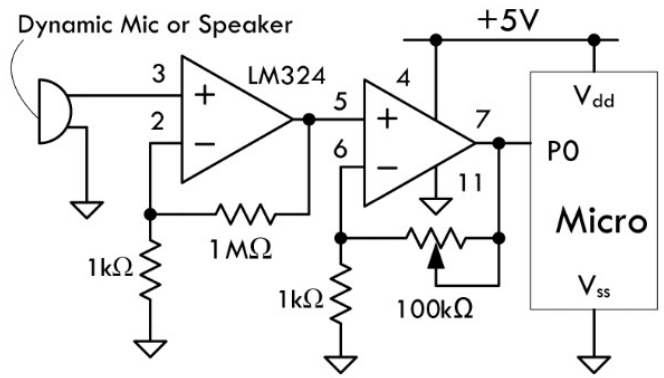


Step Sequence

	1	2	3	4	1
P4	1	1	0	0	1
P5	0	0	1	1	0
P6	1	0	0	1	1
P7	0	1	1	0	0

Detecting sound levels

Detecting sound



This circuit uses an LM324 comparator IC connected to a dynamic microphone or speaker. When a specific sound level is reached—set by pot, the output suddenly changes, supplying a HIGH to the microcontroller's input.

Looking for more interfaces?

Refer AOE 15.8

Capturing Sound

Dynamic

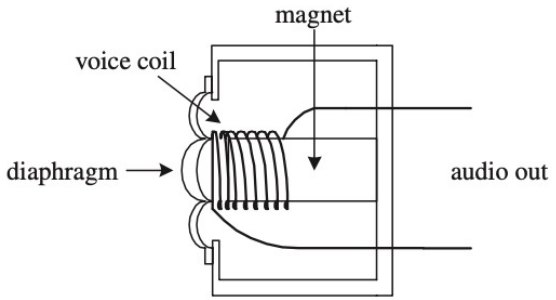


FIGURE 16.3

Condenser

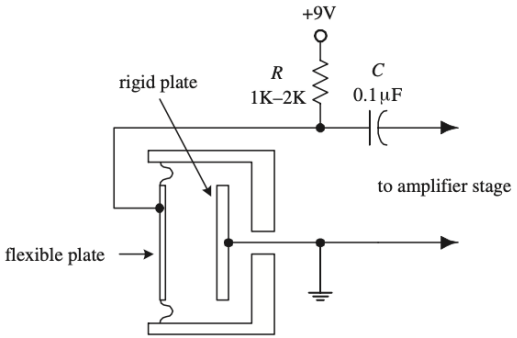


FIGURE 16.4

Electret

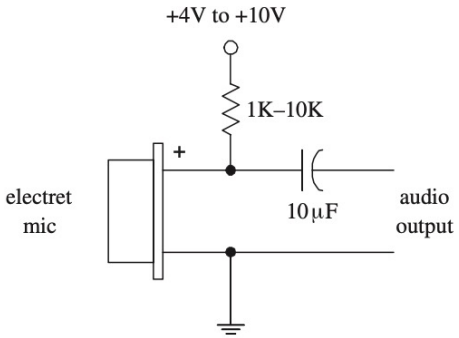
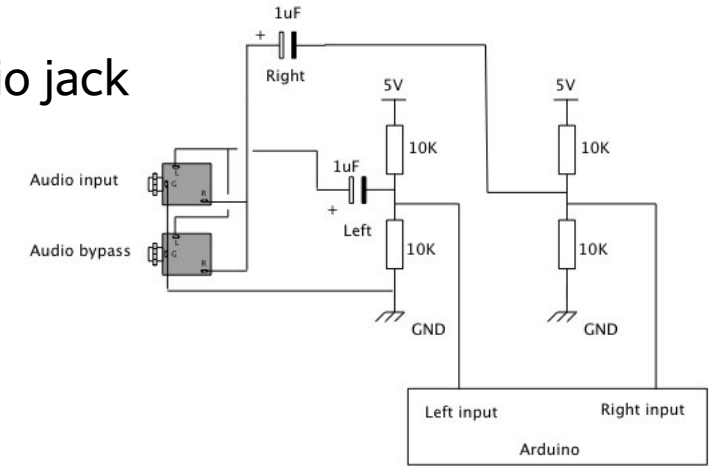


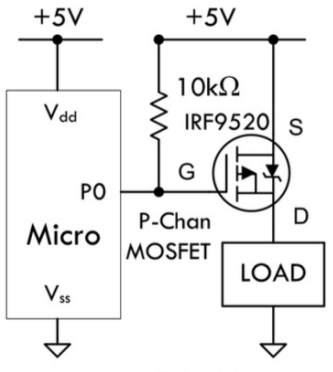
FIGURE 16.5

Microphones

From an audio jack



Generating Sound



Load = 8 ohm piezo speaker

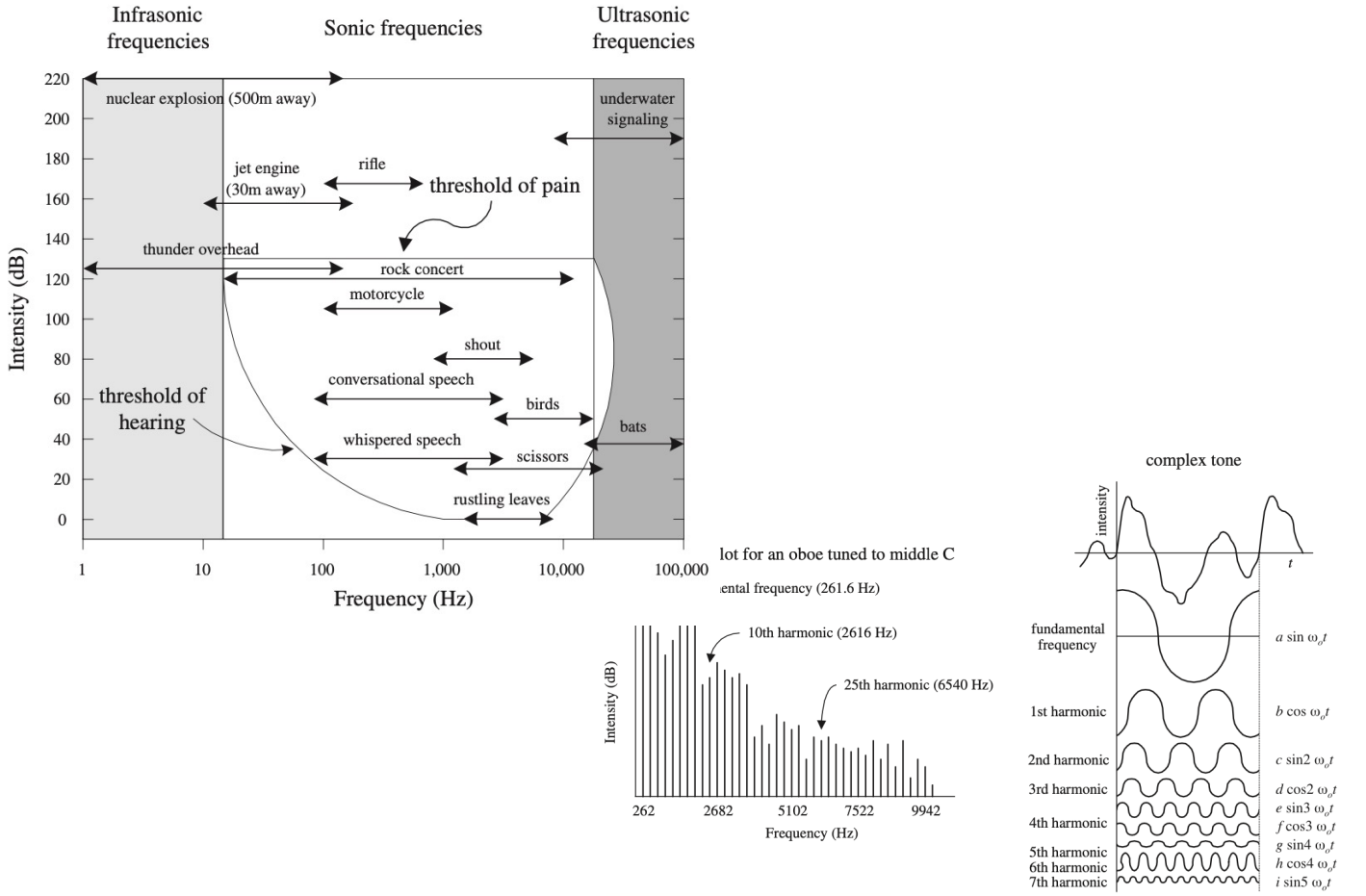
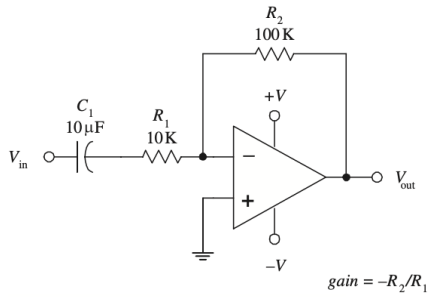


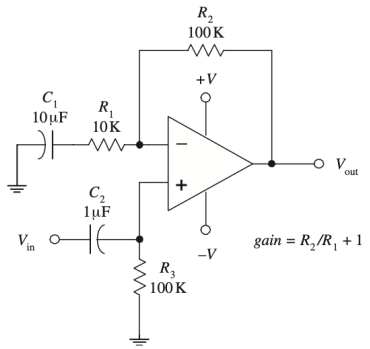
FIGURE 16.2

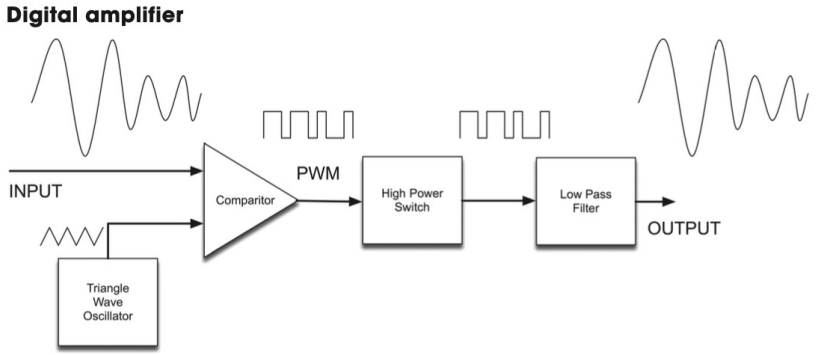
Amplifying Sound

Inverting amplifier (dual power supply)



Noninverting amplifier (dual power supply)





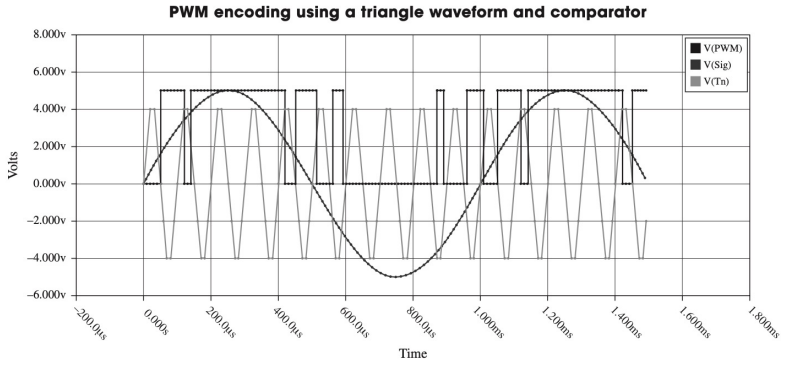


FIGURE 16.9

Linear on-board amplifiers (low gain)

Common ICs – LM386, LM383

Talking to other devices

Table 14.3 Common Buses and Data Links[a]

	Par/Ser	PP/MD[b]	Mb/s (max)[d]	Devices per ch	Address bits	Length max (m)	Hot swap?	Used for	Comments
computer internal									
PC104/ISA	P8,16	MD		-	20, 24	-	N	peripheral cards	original IBM PC/XT (ISA), in PC104 format
PCI	P32,64	MD	1000, 2000	-	32	-	N	peripheral cards	obsolescent
PCIe	P1,2,4,8,16	PP	4000 per "lane"	1	-	-	N	graphic & peripheral cards	multiple serial unidirectional lanes[e]
IDE/PATA	P16	MD	1000	2	2	0.45	N	hard disk and optical drives	obsolescent
SATA	S	PP	1500, 3000, 6000	1	-	1	Y	hard disk, ssd, & optical drives	hot-swap; also external: eSATA
SCSI	P8/16	MD	320, 2500	8/16	-	12	note f	hard disk and tape storage	ultra2/ultra-320; also external
SAS	S	PP	3000, 6000	4	-	8	Y	hard disk drives	"serial attached SCSI"
computer peripheral (external)									
4-20mA	S	PP	110 baud	few	-	10k	Y	slow devices, "legacy"	"current loop," robust in noisy environments
RS-232C	S	PP	0.1	1	-	~30	Y	instruments, computer port	"COM" ports; USB-to-serial adapters
RS-485	S	MD	0.1, 10	32	-	1000, 10 k	Y	process control	see graph, end Chapter 12
parallel (printer)	P8	PP	2.5	1	-	10	Y	printers; obsolete	original printer port; USB-to-parallel adapters
GPIB	P8	MD	8	15	-	20	Y	test/measurement insts	fat cable!
SCSI	P8,16	MD	320, 2500	8/16	-	12	N	hard disk and tape storage	ultra2 / ultra-320; also internal
eSATA	S	PP	3000	15	-	2	Y	hard disk and optical drives	hot swap; external SATA
Firewire	S	MD	480,800,(3200)	63	64	10	Y	disk storage, live video	full duplex, repeater topology, IEEE-1394
USB 1.1	S	PP	12	127	11	5	Y	computer perhiperals	half-duplex, tiered star topology
USB 2.0	S	PP	400	127	12	5	Y	...including hard drives, etc	
USB 3.0	S	PP	4800	127	-	3	Y	...including solid-state drives	full-duplex on added SuperSpeed pairs
Ethernet	S	PP	10, 100, 1000	1	48	100[h]	Y	computer network	original coax was multi-drop
WiFi	S	PP	6 to 54, to 600[i]	1	48	180	Y	wireless ethernet	IEEE 802.11a, g, n
Bluetooth	S	PP	1, 2, 3	8	48	10	Y	wireless perhiperals	range: 1m at 1mW to 100m at 100mW
Zigbee	S	MD	0.045, 0.25	240[j]	64	30	Y	small nearby wireless devices	15-hop mesh network, IEEE 802.15.4
CAN	S	MD	1.0, 0.01	64	note k	1k, 40	Y	auto, factory floor, lab	8-byte payload per packet
LIN	S	MD	0.02	16	note l	40	Y	automobile	one wire, sub-network under CAN
inter-chip									
parallel[c]	P4,8,16	MD	to 3000	NA	-	-	-	general purpose, fast	
LVDS	S	PP	to 6000	NA	-	-	-	IC to IC, backplane, fast	
I²C / SMBus	S	MD	0.01,0.1,0.41,3.4	112	7	-	-	IC to IC, addressable	2-wire bidirectional, with handshake and adr
SPI	S	MD	>20[g]	1	-	-	-	simple IC to IC	4-wire, full-duplex, no handshake, no address
JTAG	S	PP	~100	many	7	-	Y	diagnostic, program loading	IEEE 1149.1
1-wire	S	MD	0.1	many	64	100	Y	sensors	Dallas; power and data on one line!

Ref:AOE

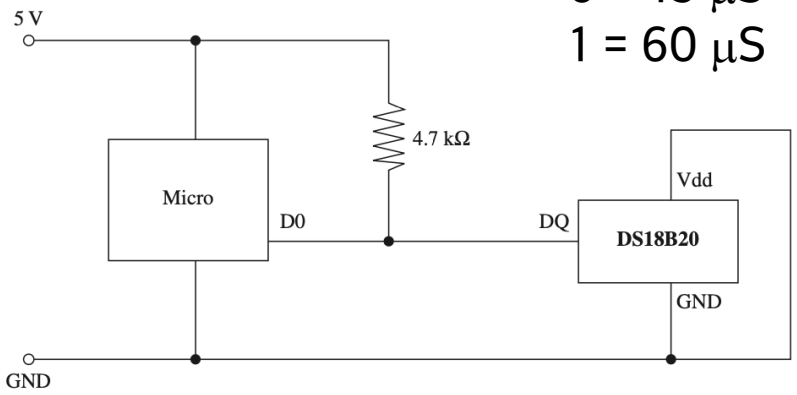
Notes: (a) there are more! these are commonly used, and likely to endure. (b) PP= point-to-point, MD = multidrop. (c) generic, no standards. (d) assumes no overhead. (e) PCIe uses multiple parallel "lanes" of unidirectional serial links; the number of lanes is written "Xn": X1, X2, X4, X8, X16. Each lane accommodates 4 or 6 Gb/s. (f) yes for SCA-2 connectors. (g) chip dependent, no established standard. (h) 100base-Tx. (i) four 64-QAM spatial streams, 40 MHz channel width. (j) up to 65534 is possible. (k) with DeviceNet protocol layer. (l) communication initiated by master.

One-wire

Pull Up

Reset = 480 μS
0 = 15 μS
1 = 60 μS

DS18B20 in parasitic power mode

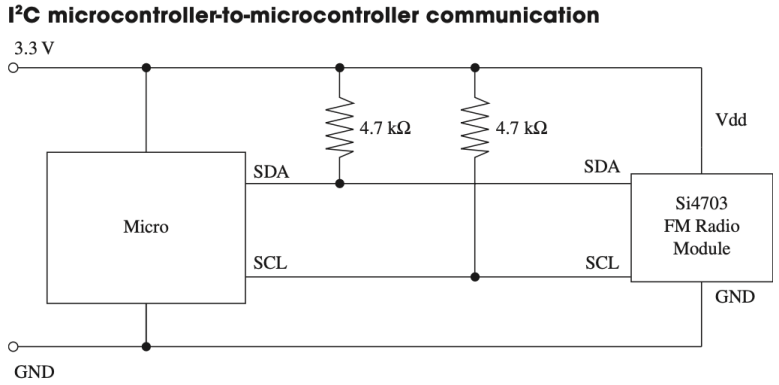


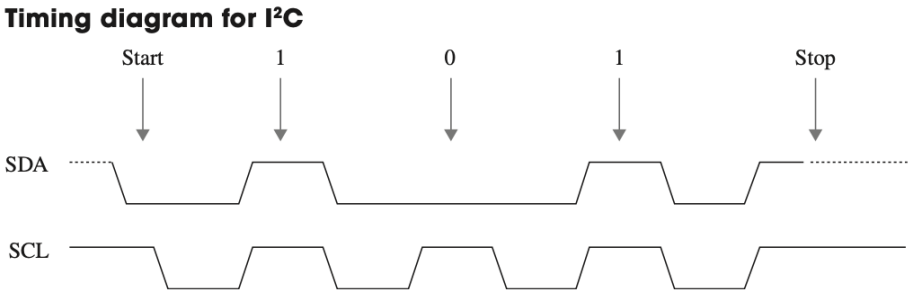
Power-less operation

```
#include <OneWire.h>

OneWire  ds(10);   // DS18B20 on pin 10
byte data[12];     // buffer for data
byte addr[8];      // 64 bit device address

void setup(void)
{
  Serial.begin(9600);
  if (ds.search(addr))
  {
    Serial.println("Slave Found");
  }
  else
  {
    Serial.println("Slave Not Found");
  }
}
```

```
float getReading()
{
  ds.reset();
  ds.select(addr);
  ds.write(0x44, true);    // command: start temp conversion,
                           // true for parasitic power mode
  delay(750);

  ds.reset();
  ds.select(addr);
  ds.write(0xBE);          // command: Read Scratchpad

  for (inti = 0; i< 9; i++)
  {
    data[i] = ds.read();
  }
  return (((data[1] << 8) + data[0]) * 0.0625);
}
```

I2C (TWI)





5/3.3V – 500 bits/s

No parasitic mode – 4 wire interface

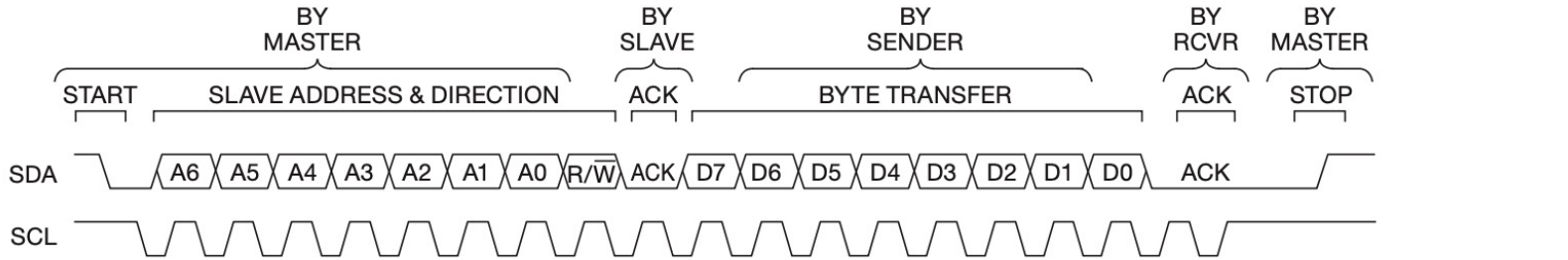
Sending Data

```
#include <Wire.h>

void setup()
{
Wire.begin(); // join i2c bus
}

void loop()
{
  Wire.beginTransmission(4);  // transmit to device #4
  Wire.write("Hello");        // friendly greeting
  Wire.endTransmission();     // stop transmitting
  delay(1000);
}
```

Receiving Data

```
void setup()
{
  Wire.begin(4);                // join i2c bus with address #4
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600);           // start serial for output
}

 void receiveEvent(inthowMany)
 {
   while(Wire.available())
   {
     char c = Wire.read();    // read a byte as a char
     Serial.print(c);         // print the character
   }
   Serial.print('\n');        // end of line
 }
```

I2C (TWI)



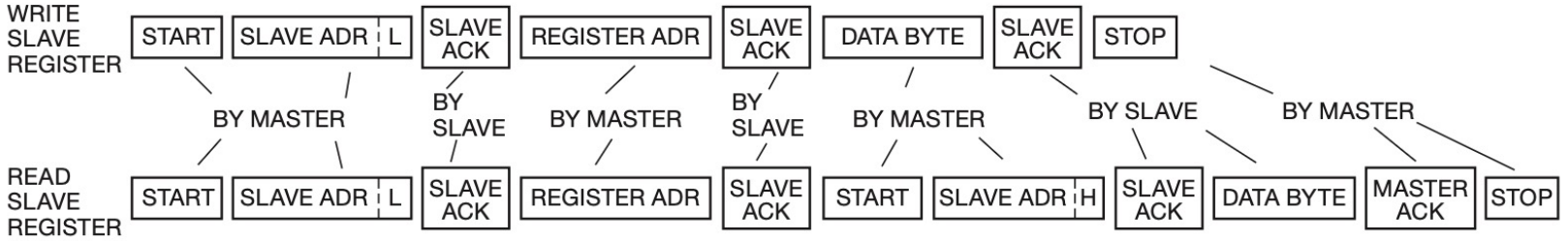
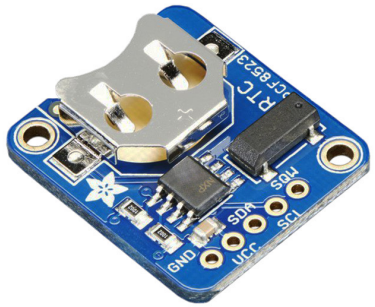


Figure 14.40. The I^{2}C 2-wire protocol (upper waveform pair): all transfers are in 8-bit groups, with a 1-bit acknowledgment (ACK). The first byte after START is always the master's assertion of the slave address (A6..0) and direction (R/W′); subsequent bytes flow from sender to receiver (depending on that first R/W′), asserted by sender and acknowledged by receiver. The master can access registers within a slave device by sending their internal address as a data byte, as shown in the lower block diagrams; both a WRITE and a READ are illustrated, the latter requiring a "repeated START" to create a READ after writing the register's address in the initial transaction.

Some common I2C devices

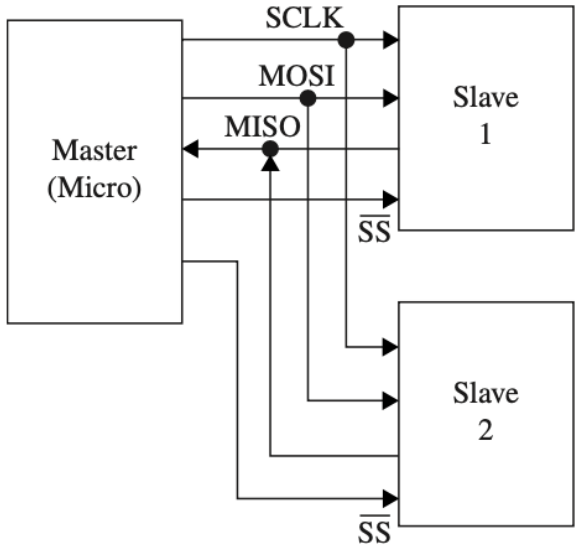


RTC

1. I2C Sensors
2. I2C ADC (Analog-to-Digital Converters)
3. I2C DAC (Digital-to-Analog Converters)
4. I2C GPIO Expanders
5. I2C Real-Time Clock (RTC) Modules
6. I2C EEPROM (Electrically Erasable Programmable Read-Only Memory)
7. I2C OLED Displays
8. I2C Motor Controllers
9. I2C Audio Codecs
10. I2C GPIO Sensors
11. I2C Expansion Hubs
12. I2C RTC with NVRAM (Non-Volatile RAM)
13. I2C Touch Sensors
14. I2C LED Drivers
15. I2C Motor Encoders
16. I2C Gas Sensors

SPI

SPI connections



Note: Full duplex (upto 80 Mbit/s)

Because SPI (and SPI-like variants) do not conform to a well-defined standard, you have to read carefully the datasheet specifications for each interfaced chip. You'll discover, in addition to the polarity modes already mentioned, that the maximum (and minimum!) clock speeds can range from a few kilohertz to many megahertz.

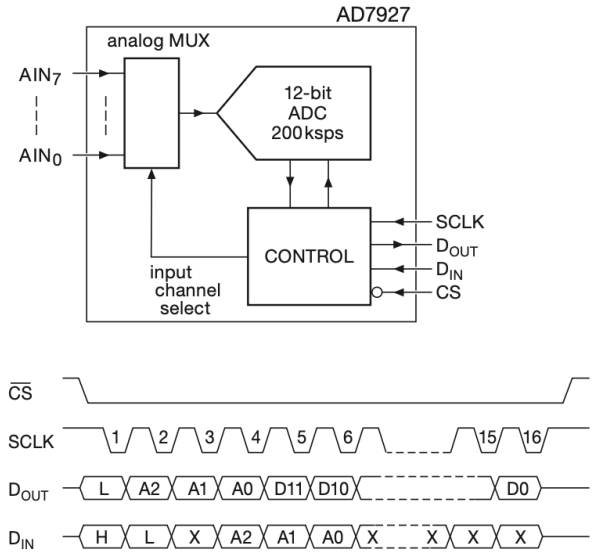


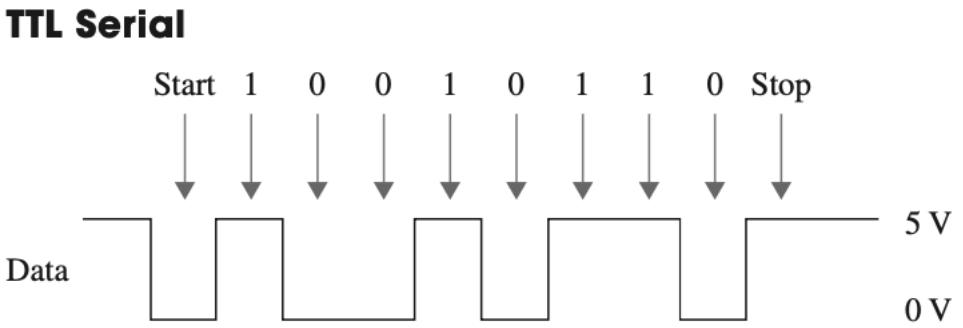
Figure 14.39. An 8-channel ADC with SPI control and readout. The protocol shown is the simplest of several allowed modes: the master's CS′ starts a conversion, whereupon the slave outputs both the channel address (3 bits) and the converted value (12 bits); the master simultaneously sends the address of the next input channel to be converted.

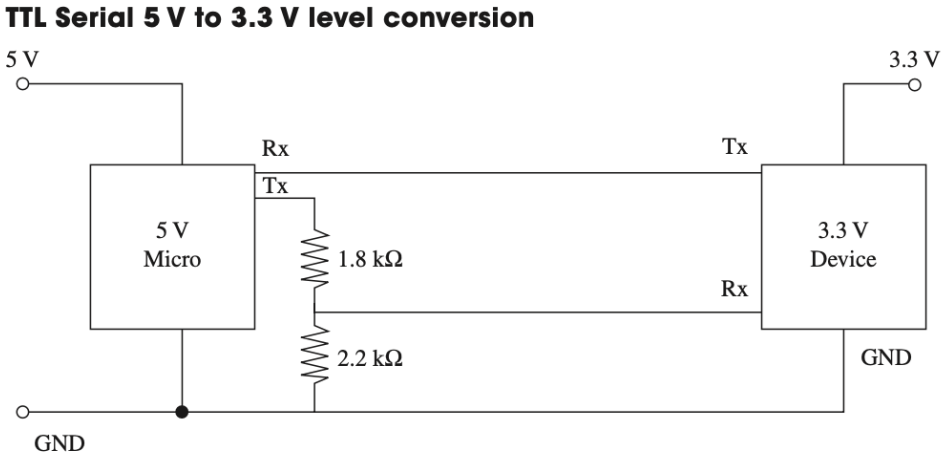
TTL Serial

Rx, Tx with fixed baud rates

110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000, and 256000.

Full Duplex





MODBUS

- An industry standard interface on top of RS485 like signals
  - Also implemented over ethernet with TCP/IP stack
  - Common in gas/water/energy meters, etc.

USB

Introduced in 1905

- V1 - Low Speed, Full Speed (1. 5 Mbps/12 Mbps) [Differential voltage]
- V2 – High speeds (480 Mbps) [Differential voltage based binary encoding]
- V3 – Superspeed (4.8 Gbps via shielded twisted pairs) [8b/10b encoding]
  - V3.1 – 9.6 Gbps with increased DC power options – 5V, 2A or 12/20V 5A with USB C connector

Master-slave Star Topology

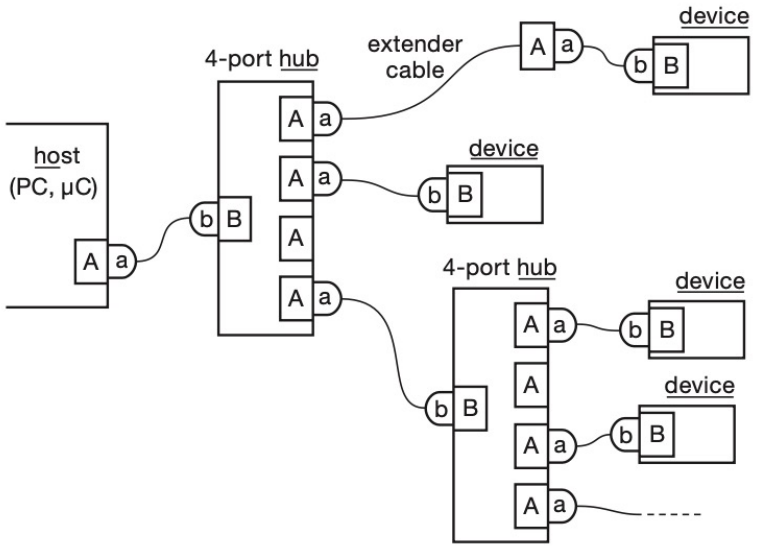


Figure 14.47. USB is a master–slave serial interface, arranged in a star topology (as opposed to a serial repeater topology like FireWire), with up to five repeater "hubs." The individual links are asymmetrical cables, with A at the master end and B at the slave end (a full-size pair is shown in Figure 1.123); here a and b represent plugs, A and B sockets. Individual links are limited to 5 m length (including passive extender cables), but an active hub resets the "ruler." An interesting variant (not shown) is the OTG (on-the-go) USB port, a chameleon that can masquerade as either an A or a B port.

Controller Area Network (CAN)

- Multidrop km-range bus accommodating upto 30 transceiver nodes
- 1Mbps for 40m and 10 kbps at 1km (max)
- Quiet bus on no data
  - Collisions and backoffs

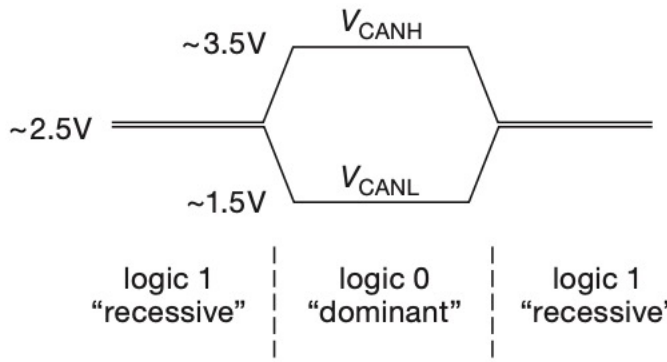
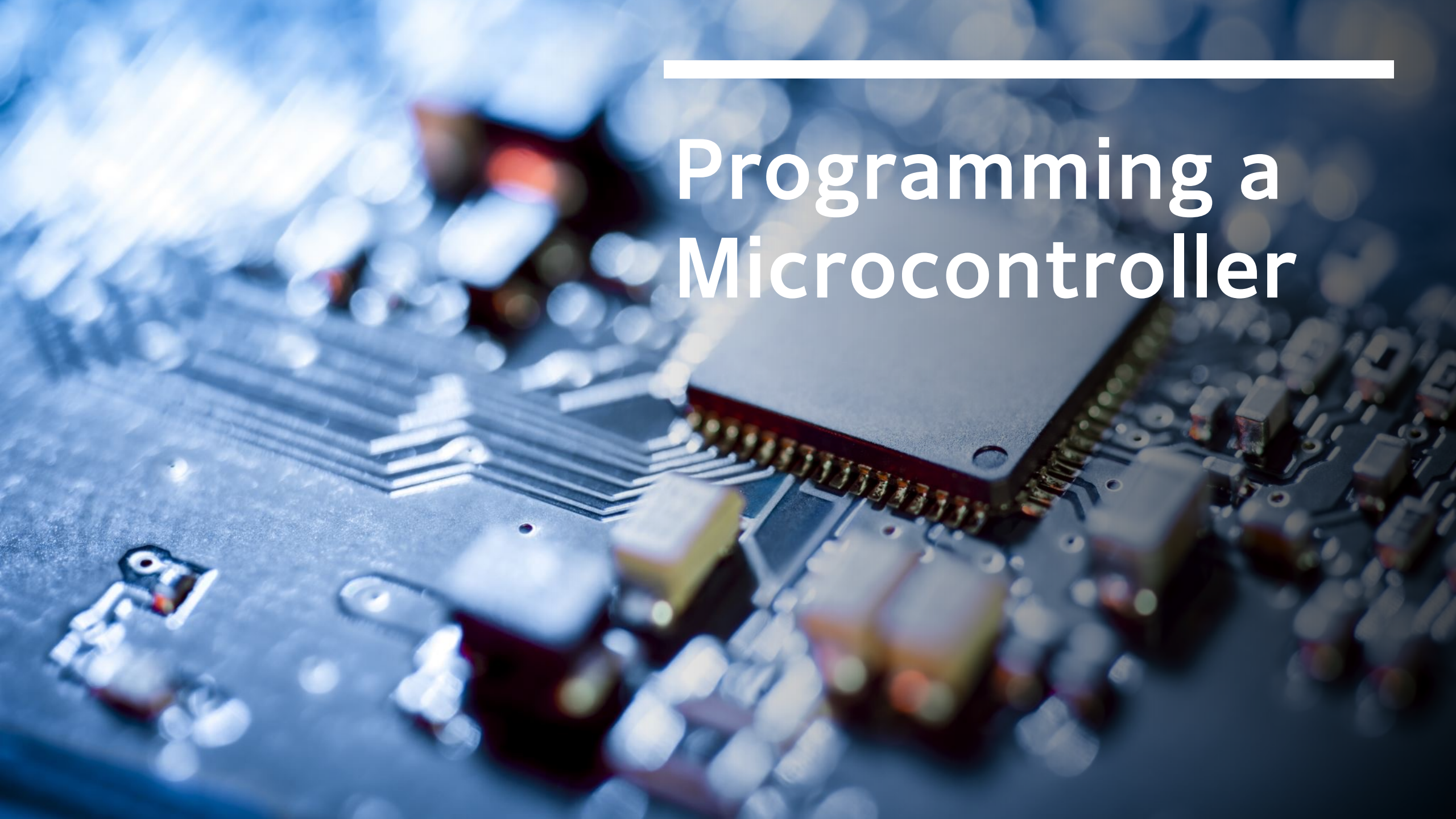


Figure 14.49. The CAN bus is differential, with an asymmetric signaling mode that simplifies arbitration.



Programming a Microcontroller

Programming μC != Programming a computer

- There's only one CPU!
  - No task switching unless you make it switch
- Plenty of parallel hardware to use
  - Use interrupts as much as possible – to (not) wait for all kinds of things!
  - Sei() – global interrupt enable
  - Cli() – global interrupt disable

Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

Note: The next few slides are Atmega328P specific

Instruction execution

Figure 6-4. The Parallel Instruction Fetches and Instruction Executions

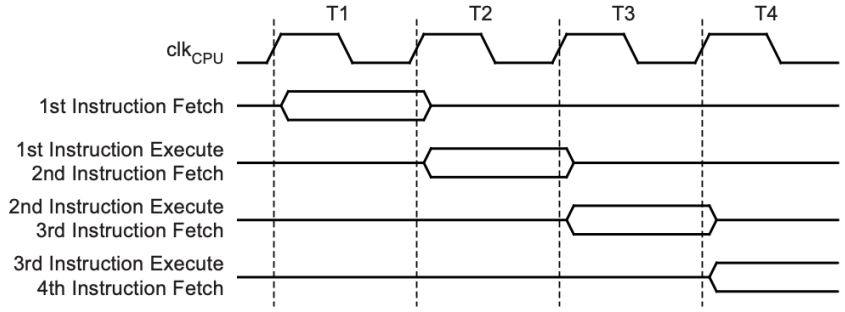
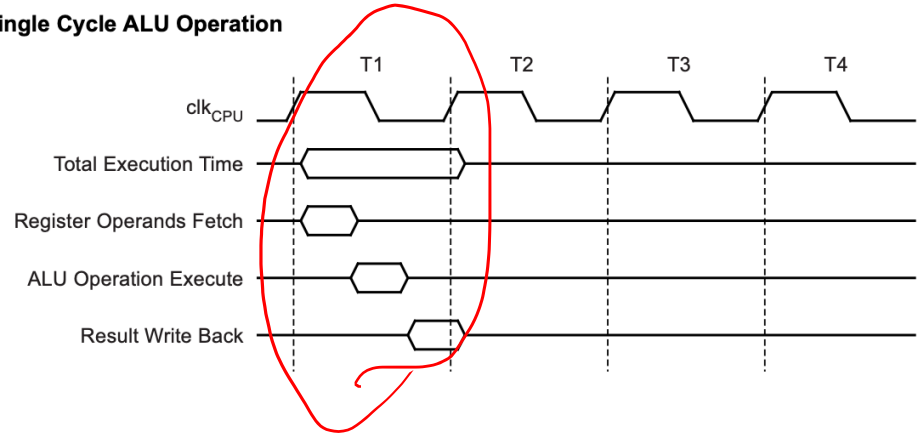
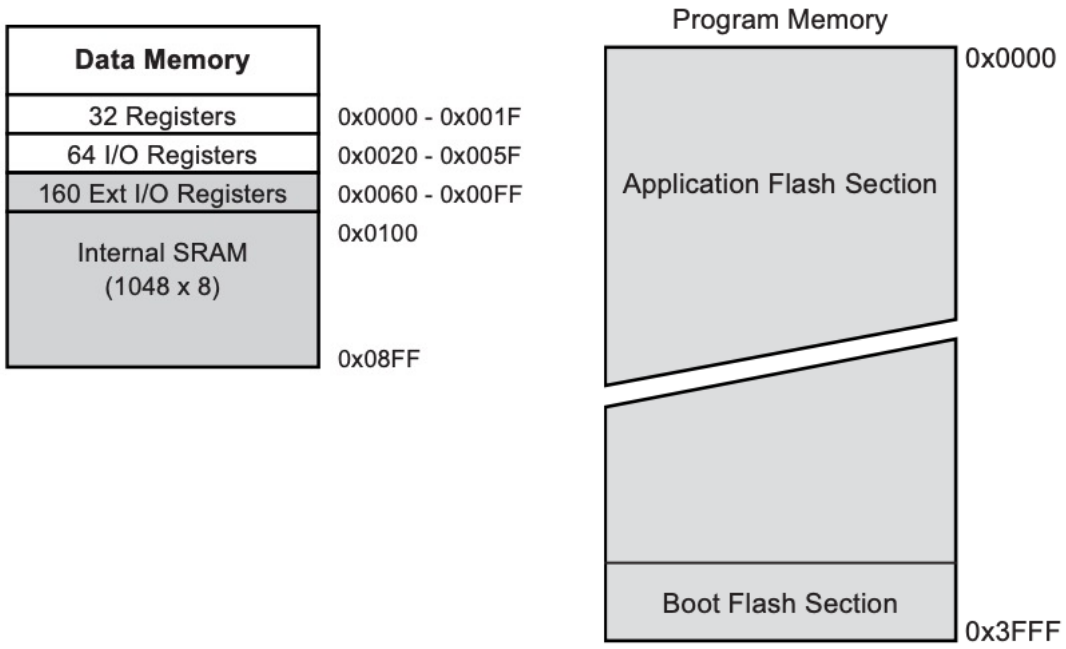


Figure 6-5 shows the internal timing concept for the register file. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 6-5. Single Cycle ALU Operation



Memories



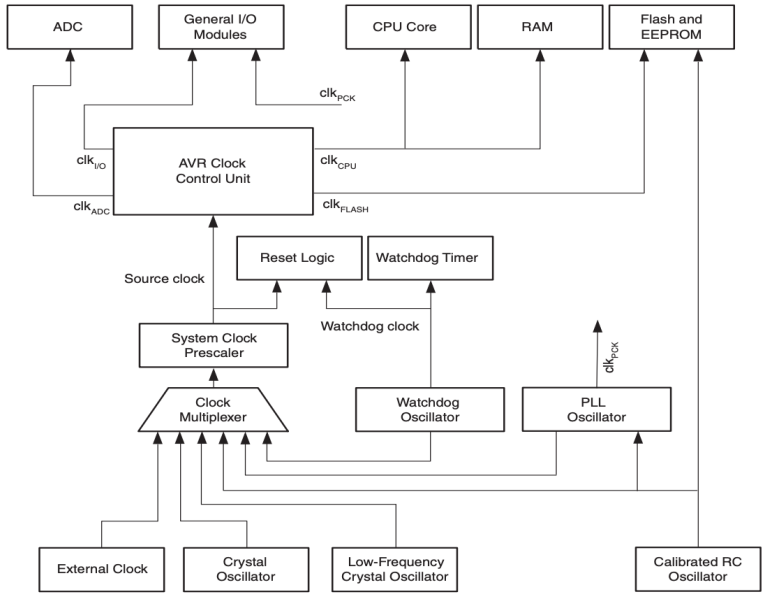
1 kB EEPROM
- parallel r/w
- SPI r/w

Clock

Table 6-1. Device Clocking Options Select

Device Clocking Option	CKSEL[3:0](1)
External Clock (see page 26)	0000
High Frequency PLL Clock (see page 26)	0001
Calibrated Internal Oscillator (see page 27)	0010(2)
Calibrated Internal Oscillator (see page 27)	0011(3)
Internal 128 kHz Oscillator (see page 28)	0100
Low-Frequency Crystal Oscillator (see page 29)	0110
Crystal Oscillator / Ceramic Resonator (see page 29)	1000 – 1111
Reserved	0101, 0111

Figure 6-1. Clock Distribution



Power Management

Table 9-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

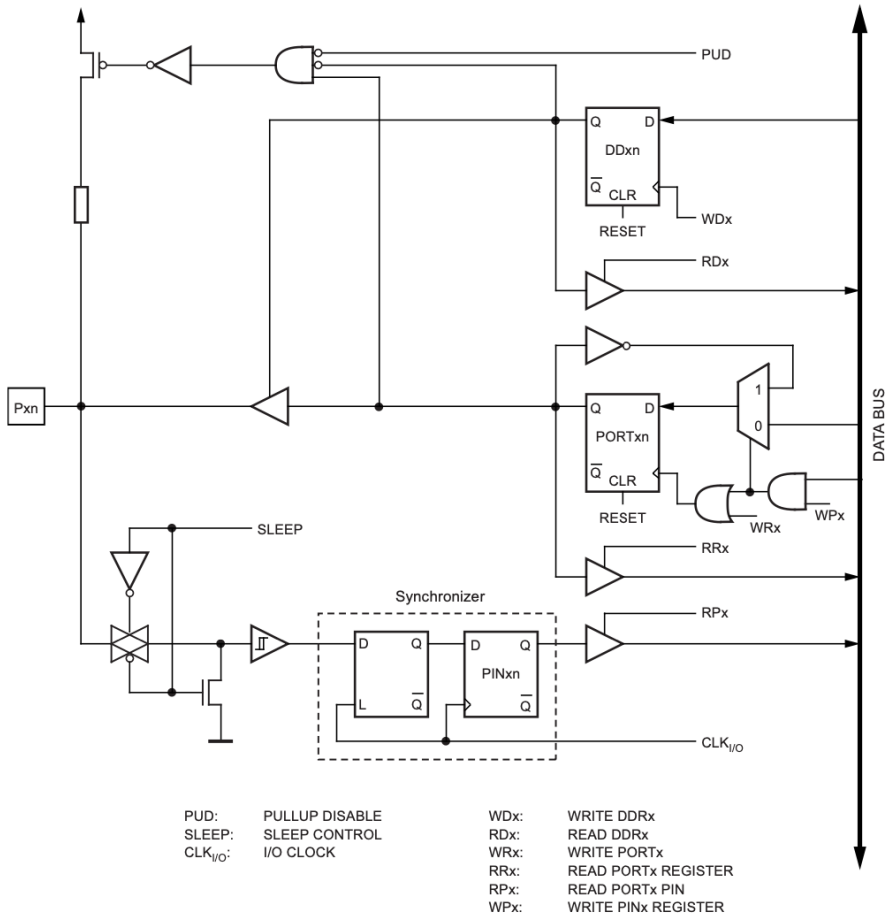
Sleep Mode	Active Clock Domains					Oscillators		Wake-up Sources							Software BOD Disable
	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other/O	
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes:
1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

I/O

DDRx
PORTx
PINx

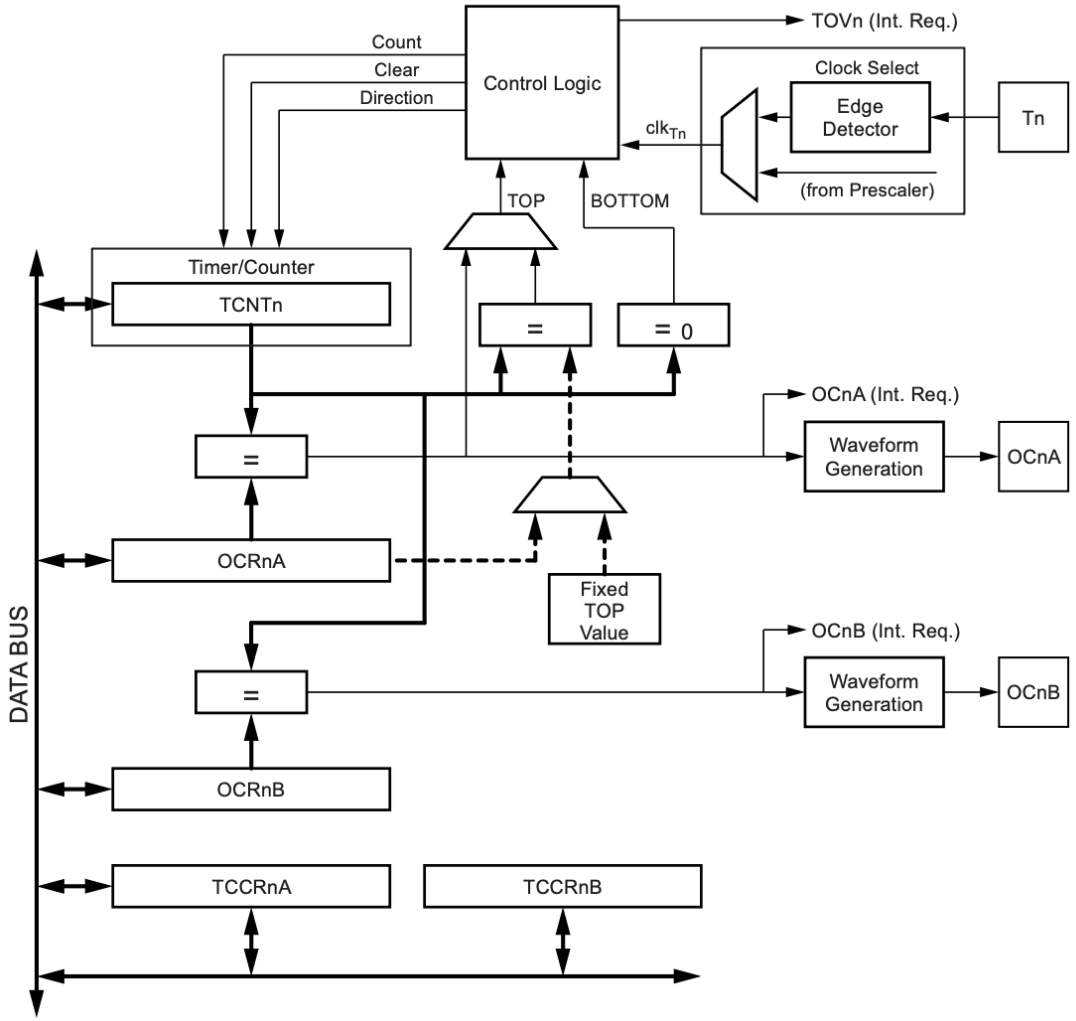
Figure 13-2. General Digital I/O(1)



Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

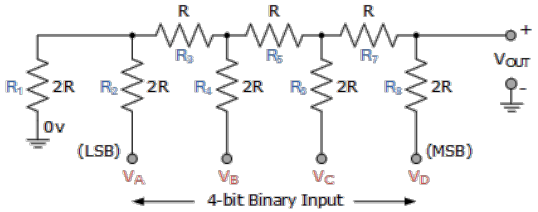
Timer

Figure 14-1. 8-bit Timer/Counter Block Diagram

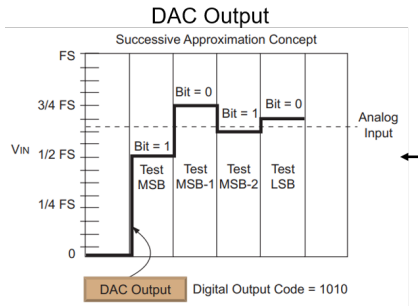


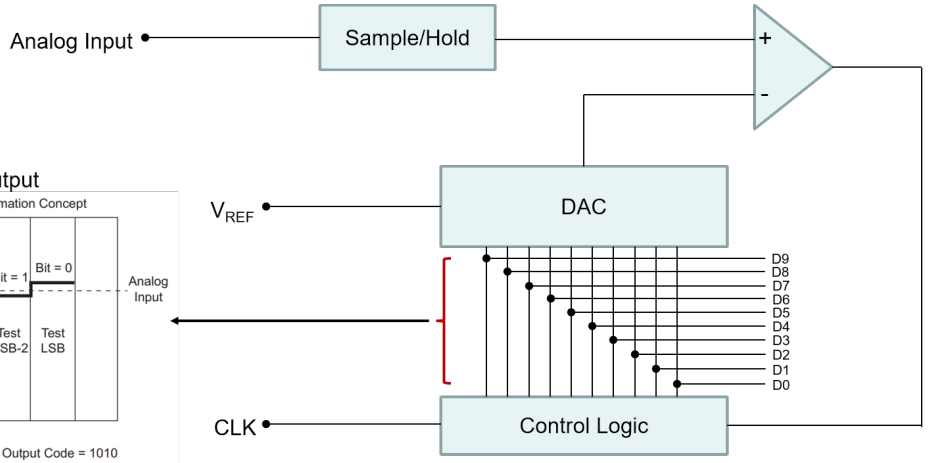
ADC Types

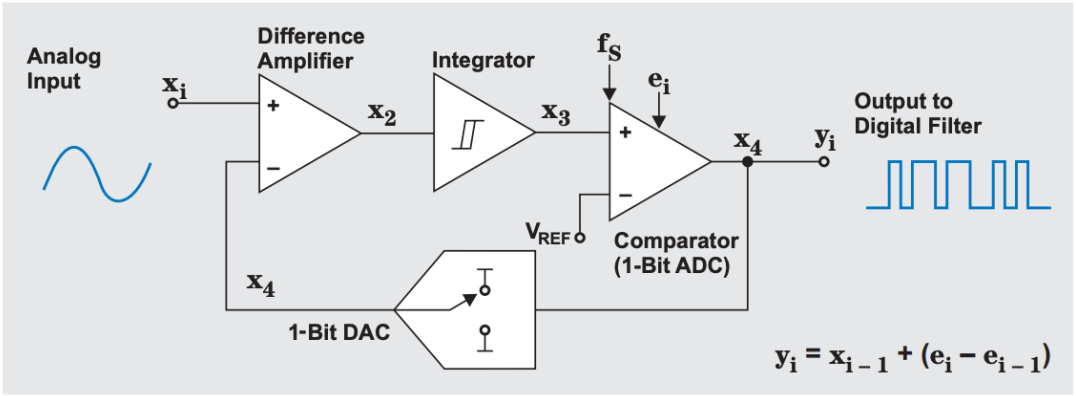
Successive Approximation



Example DAC







Delta-Sigma ADC

Delta-Sigma Modulator Basics - YouTube

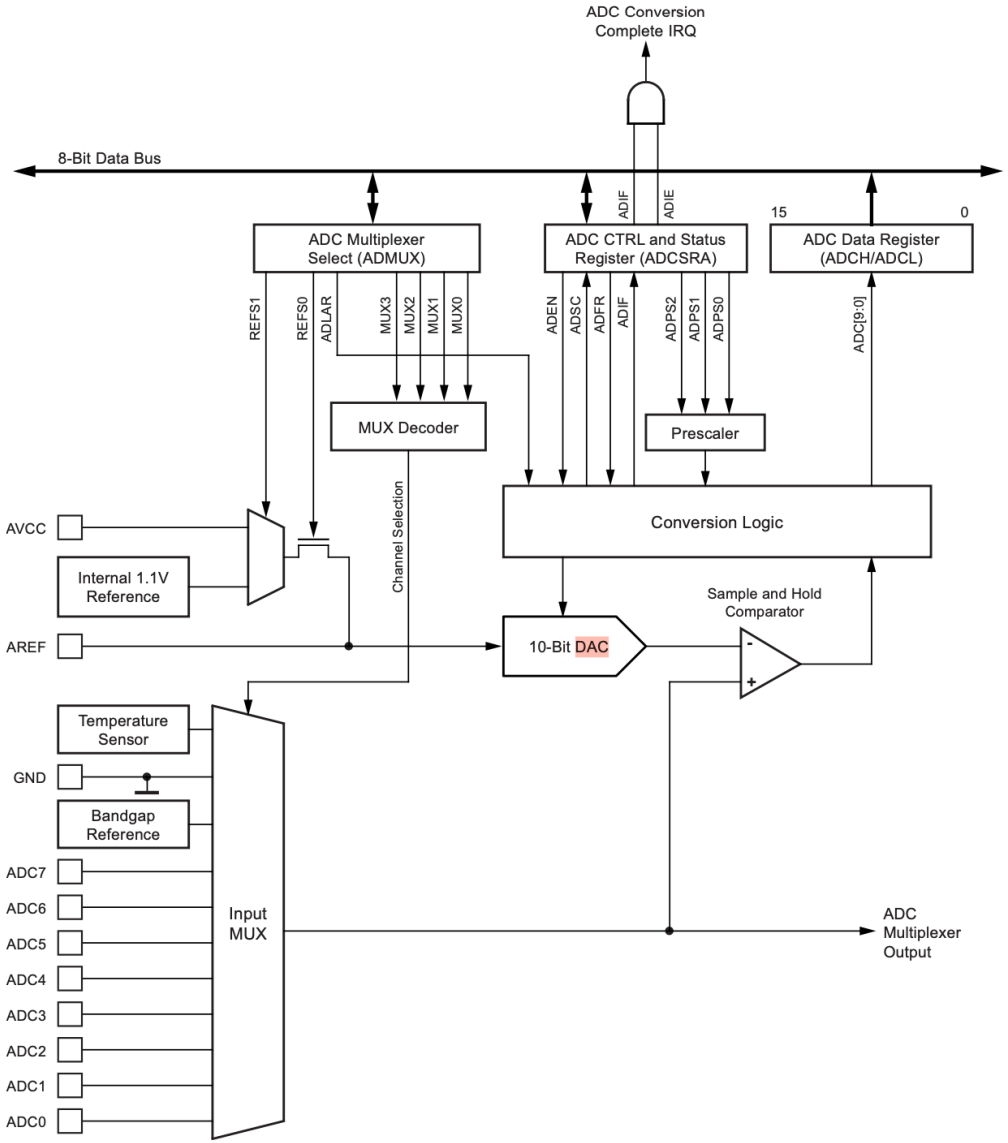
https://everycircuit.com/circuit/5048641919909888

How delta-sigma ADCs work, Part 1 (Rev. A) (ti.com)

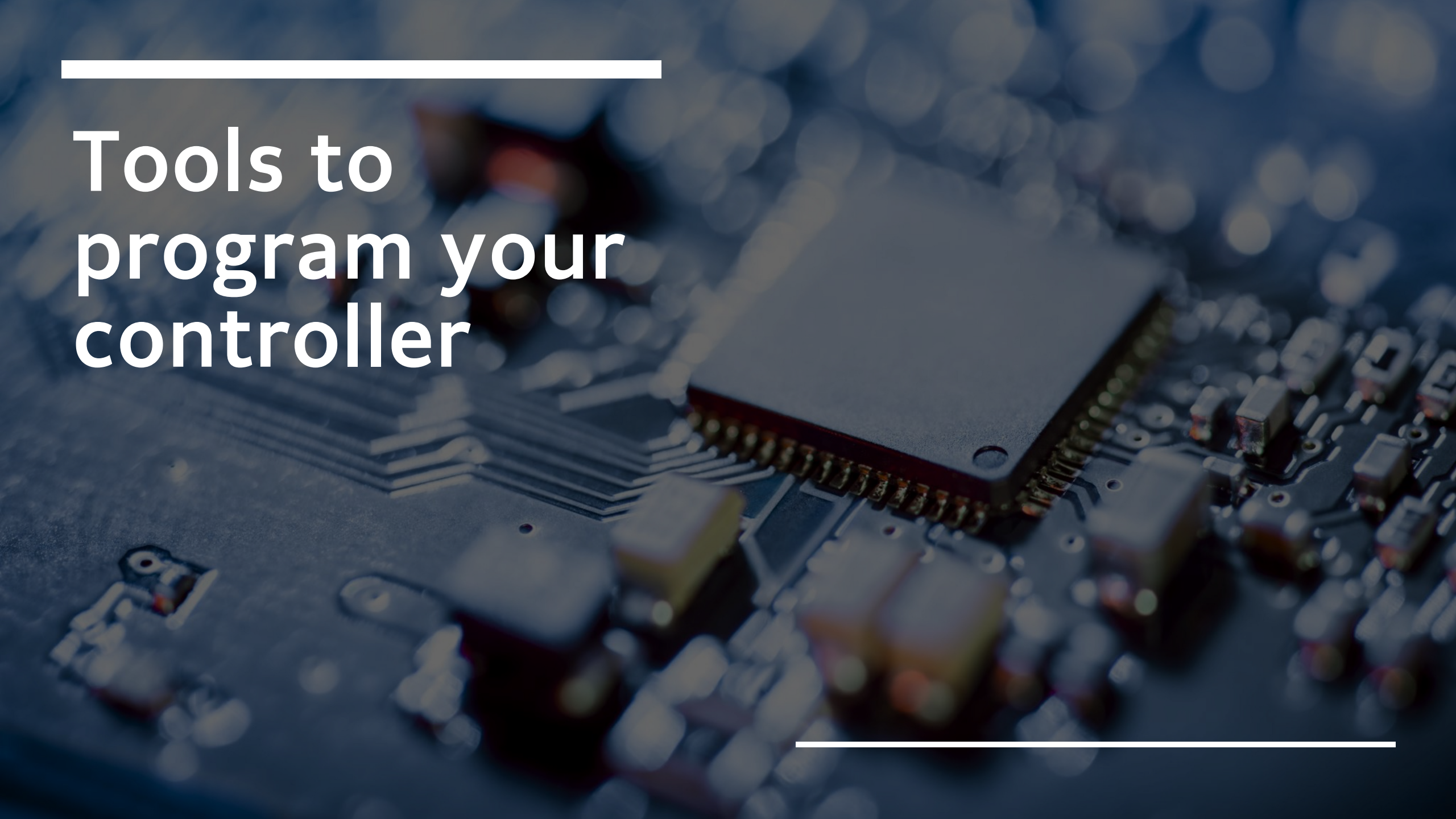
Sigma-Delta ADC Compared to SAR ADC - Developer Help (microchipdeveloper.com)

ATMega 328P ADC

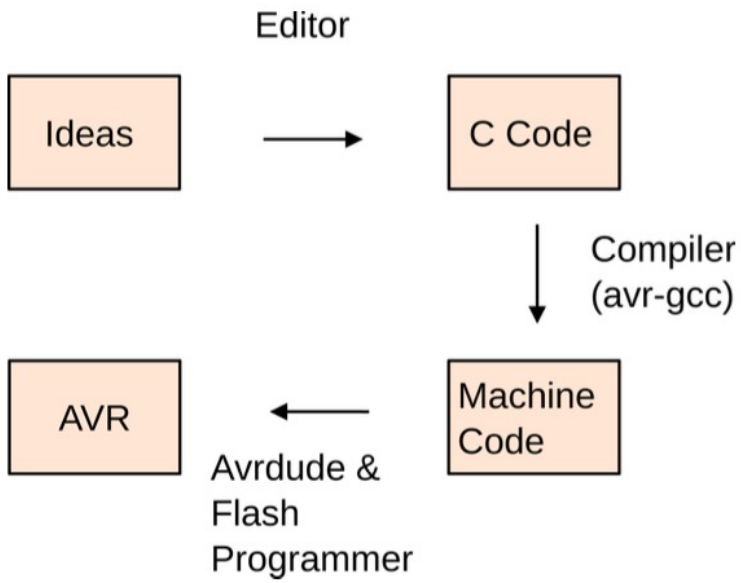
Figure 23-1. Analog to Digital Converter Block Schematic Operation

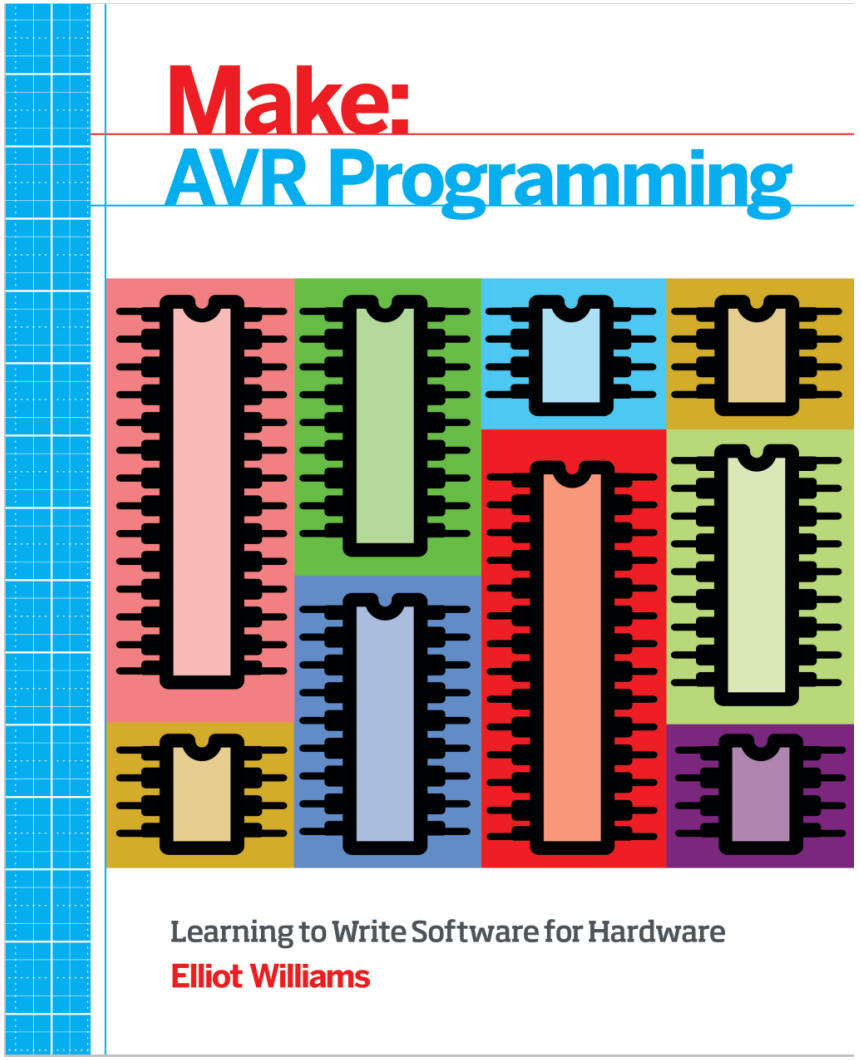


Tools to program your controller



Ideas to Code





Blink

Example 3-1. blinkLED.c listing

```
                                                    /* Blinker Demo */

// ------- Preamble -------- //
#include <avr/io.h>                        /* Defines pins, ports, etc */
#include <util/delay.h>                     /* Functions to waste time */

int main(void) {

  // -------- Inits --------- //
  DDRB = 0b00000001;            /* Data Direction Register B:
                                   writing a one to the bit
                                   enables output. */

  // ------ Event loop ------ //
  while (1) {

    PORTB = 0b00000001;          /* Turn on first LED bit/pin in PORTB */
    _delay_ms(1000);                                        /* wait */

    PORTB = 0b00000000;          /* Turn off all B pins, including LED */
    _delay_ms(1000);                                        /* wait */

  }                                                 /* End event loop */
  return (0);                              /* This line is never reached */
}
```

POVToy

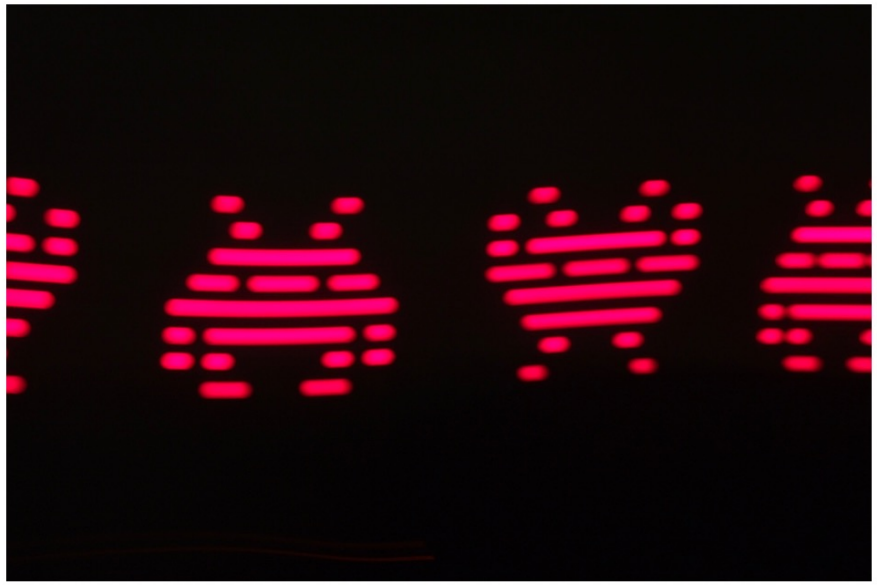
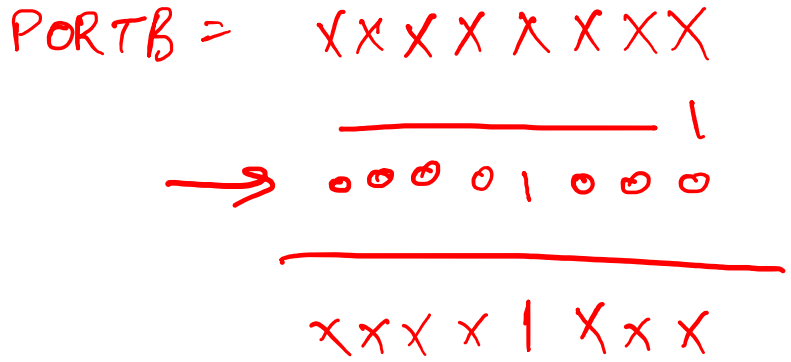


Figure 3-1. *POV invasion*

Example 3-2. *povToy.c listing*

```
// POV toy demo framework //

// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>

// -------- Functions --------- //
void POVDisplay(uint8_t oneByte) {
  PORTB = oneByte;
  _delay_ms(2);
}

int main(void) {
  // -------- Inits --------- //
  DDRB = 0xff;                    /* Set up all of LED pins for output */
  // ------ Event loop ------ //
  while (1) {                                                /* mainloop */
    POVDisplay(0b00001110);
    POVDisplay(0b00011000);
    POVDisplay(0b10111101);
    POVDisplay(0b01110110);
    POVDisplay(0b00111100);
    POVDisplay(0b00111100);
    POVDisplay(0b00111100);
    POVDisplay(0b01110110);
    POVDisplay(0b10111101);
    POVDisplay(0b00011000);
    POVDisplay(0b00001110);

    PORTB = 0;
    _delay_ms(10);
  }                                                      /* end mainloop */
  return (0);
}
```

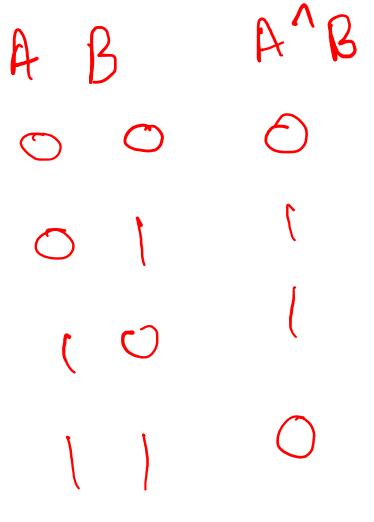
Bit Twiddling

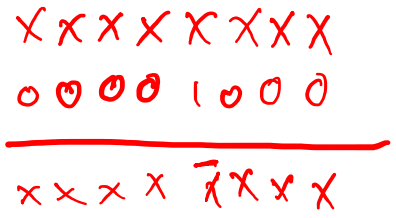


PORTB |= (1 << 3) //Turn on the third LED

PORTB &= ~(1 << 3) //Turn off the third LED

PORTB ^= (1 << 3) //Toggle the third LED





The 3 is usually defined in the library as PB3

PORTB | = (1 << PB3) | (1 << PB5)

Serial

```
*/

#include <avr/io.h>
#include "USART.h"
#include <util/setbaud.h>

void initUSART(void) {                              /* requires BAUD */
  UBRR0H = UBRRH_VALUE;                        /* defined in setbaud.h */
  UBRR0L = UBRRL_VALUE;
#if USE_2X
  UCSR0A |= (1 << U2X0);
#else
  UCSR0A &= ~(1 << U2X0);
#endif
                                  /* Enable USART transmitter/receiver */
  UCSR0B = (1 << TXEN0) | (1 << RXEN0);
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);   /* 8 data bits, 1 stop bit */

}

void transmitByte(uint8_t data) {
                                /* Wait for empty transmit buffer */
  loop_until_bit_is_set(UCSR0A, UDRE0);
  UDR0 = data;                                          /* send data */
}

uint8_t receiveByte(void) {
  loop_until_bit_is_set(UCSR0A, RXC0);       /* Wait for incoming data */
  return UDR0;                                /* return register value */
}


//  Example of a useful printing command
void printString(const char myString[]) {
  uint8_t i = 0;
  while (myString[i]) {
    transmitByte(myString[i]);
    i++;
  }
}
```

```
// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include "pinDefines.h"
#include "USART.h"

int main(void) {
  char serialCharacter;

  // -------- Inits --------- //
  LED_DDR = 0xff;                                /* set up LEDs for output */
  initUSART();
  printString("Hello World!\r\n");                            /* to test */

  // ------ Event loop ------ //
  while (1) {

    serialCharacter = receiveByte();
    transmitByte(serialCharacter);
    LED_PORT = serialCharacter;
                            /* display ascii/numeric value of character */

  }                                                    /* End event loop */
  return (0);
}
```

Play a note

Example 5-3. playNote function listing

```
void playNote(uint16_t wavelength, uint16_t duration){
  uint16_t elapsed;

  uint16_t i;
  for(elapsed = 0; elapsed < duration; elapsed += wavelength){
    /* For loop with variable delay selects the pitch */
    for (i = 0; i < wavelength; i++){
      _delay_us(1);
    }
    SPEAKER_PORT ^= (1 << SPEAKER);
  }
}
```

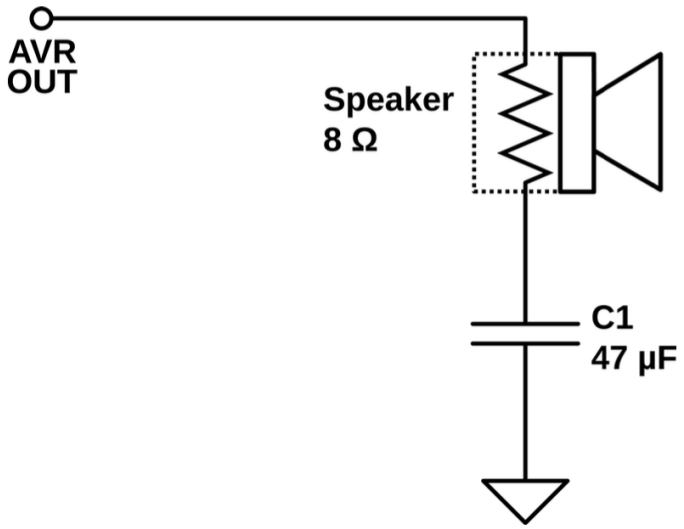


Figure 5-7. *Speaker with a blocking capacitor*

SerialNote

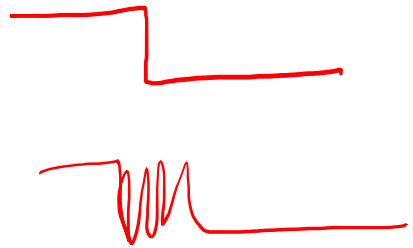
Example 5-4. serialOrgan.c listing

```
/*

serialOrgan.c

Reads a character in serial from the keyboard, plays a note.

See organ.h for pin defines and other macros
See organ.c (and include it in the Makefile) for playNote() and rest()

*/

// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include "organ.h"
#include "scale16.h"
#include "pinDefines.h"
#include "USART.h"

#define NOTE_DURATION     0xF000        /* determines long note length */

int main(void) {

  // -------- Inits --------- //
  SPEAKER_DDR |= (1 << SPEAKER);                /* speaker for output */
  initUSART();
  printString("----- Serial Organ ------\r\n");

  char fromCompy;                         /* used to store serial input */
  uint16_t currentNoteLength = NOTE_DURATION / 2;
  const uint8_t keys[] = { 'a', 'w', 's', 'e', 'd', 'f', 't',
    'g', 'y', 'h', 'j', 'i', 'k', 'o',
    'l', 'p', ';', '\''
  };
  const uint16_t notes[] = { G4, Gx4, A4, Ax4, B4, C5, Cx5,
    D5, Dx5, E5, F5, Fx5, G5, Gx5,
    A5, Ax5, B5, C6
  };
  uint8_t isNote;
  uint8_t i;

  // ------ Event loop ------ //
  while (1) {

                                                  /* Get Key */
    fromCompy = receiveByte();      /* waits here until there is input */
    transmitByte('N');     /* alert computer we're ready for next note */

                                                  /* Play Notes */
    isNote = 0;
    for (i = 0; i < sizeof(keys); i++) {
      if (fromCompy == keys[i]) {       /* found match in lookup table */
        playNote(notes[i], currentNoteLength);
        isNote = 1;                  /* record that we've found a note */
        break;                              /* drop out of for() loop */
      }
    }

                        /* Handle non-note keys: tempo changes and rests */
    if (!isNote) {
      if (fromCompy == '[') {                   /* code for short note */
        currentNoteLength = NOTE_DURATION / 2;
      }
      else if (fromCompy == ']') {               /* code for long note */
        currentNoteLength = NOTE_DURATION;
      }
      else {                                  /* unrecognized, just rest */
        rest(currentNoteLength);
      }
    }

  }                                                 /* End event loop */
  return (0);
}
```

Debounce

Example 6-3. debouncer.c listing

```
// ------- Preamble -------- //
#include <avr/io.h>
#include "pinDefines.h"

#include <util/delay.h>
#define DEBOUNCE_TIME  1000                          /* microseconds */

uint8_t debounce(void) {
  if (bit_is_clear(BUTTON_PIN, BUTTON)) {      /* button is pressed now */
    _delay_us(DEBOUNCE_TIME);
    if (bit_is_clear(BUTTON_PIN, BUTTON)) {           /* still pressed */
      return (1);
    }
  }
  return (0);
}

int main(void) {
  // -------- Inits --------- //
  uint8_t buttonWasPressed;                                   /* state */
  BUTTON_PORT |= (1 << BUTTON);     /* enable the pullup on the button */
  LED_DDR = (1 << LED0);                    /* set up LED for output */

  // ------ Event loop ------ //
  while (1) {
    if (debounce()) {                      /* debounced button press */
      if (buttonWasPressed == 0) {     /* but wasn't last time through */
        LED_PORT ^= (1 << LED0);                        /* do whatever */
        buttonWasPressed = 1;                    /* update the state */
      }
    }
    else {                                 /* button is not pressed now */
      buttonWasPressed = 0;                         /* update the state */
    }

  }                                                  /* End event loop */
  return (0);                              /* This line is never reached */
}
```



Piano

Example 6-4. avrMusicBox.c listing

```
// Music Box Input Demo

// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include "organ.h"
#include "scale16.h"
#include "pinDefines.h"

#define SONG_LENGTH  (sizeof(song) / sizeof(uint16_t))

int main(void) {
  const uint16_t song[] = {
    E6, E6, E6, C6, E6, G6, G5,
    C6, G5, E5, A5, B5, Ax5, A5,
    G5, E6, G6, A6, F6, G6, E6, C6, D6, B5,
    C6, G5, E5, A5, B5, Ax5, A5,
    G5, E6, G6, A6, F6, G6, E6, C6, D6, B5,
                                                    /* etc */
  };
  /* starting at end b/c routine starts by incrementing and then playing
     this makes the song start at the beginning after reboot */
  uint8_t whichNote = SONG_LENGTH - 1;
  uint8_t wasButtonPressed = 0;

  // -------- Inits --------- //
  SPEAKER_DDR |= (1 << SPEAKER);                /* speaker for output */
  BUTTON_PORT |= (1 << BUTTON);                   /* pullup on button */

  // ------ Event loop ------ //
  while (1) {
    if (bit_is_clear(BUTTON_PIN, BUTTON)) {
      if (!wasButtonPressed) {                /* if it's a new press ... */
        whichNote++;                             /* advance to next note */
                                              /* but don't run over the end */
        if (whichNote == SONG_LENGTH) {
          whichNote = 0;
        }
        wasButtonPressed = 1;
      }
      playNote(song[whichNote], 1600);
    }
    else {
      wasButtonPressed = 0;
    }
  }                                                 /* End event loop */
  return (0);
}
```

Boss Button

Example 6-5. bossButton.py listing

```
## Simple demo
## Sits forever listening to serial port
## When you press button, opens website of your choosing.
## Extend this to many buttons and you'll have a physical
##  web-launcher.

BOSS_SITE = "http://www.cartalk.com/content/boss-redirect"
## or perhaps more topical...
XKCD = "http://xkcd.com/353/"

SERIAL_PORT = "/dev/ttyUSB0"
BAUD_RATE = 9600

import serial
import webbrowser

sp = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout = 5)
sp.flush()

while(1):                       # Sit and wait forever
    response = sp.read(1)       # get one byte
    if response == "O":
        print "Got OK Byte.  Waiting for button press."
    elif response == "X":
        print "Got Boss Byte!  Alarm!"
        webbrowser.open(BOSS_SITE)
    else:
        print "Got nothing.  Still waiting."
```

Light Meter

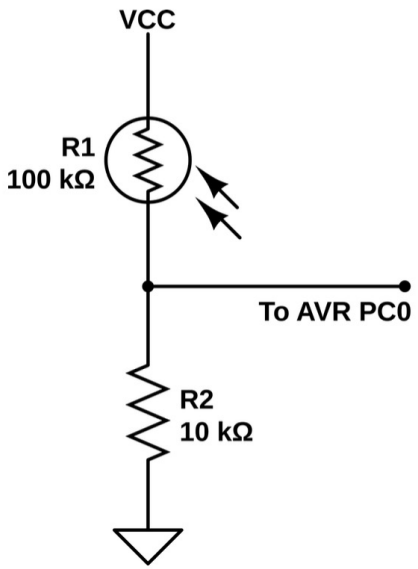


Figure 7-2. *LDR voltage divider*

Example 7-1. *lightSensor.c Listing*

```
// Quick Demo of light sensor

// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include "pinDefines.h"

// -------- Functions --------- //
static inline void initADC0(void) {
  ADMUX |= (1 << REFS0);                  /* reference voltage on AVCC */
  ADCSRA |= (1 << ADPS1) | (1 << ADPS0);    /* ADC clock prescaler /8 */
  ADCSRA |= (1 << ADEN);                              /* enable ADC */
}

int main(void) {

  // -------- Inits --------- //
  uint8_t ledValue;
  uint16_t adcValue;
  uint8_t i;

  initADC0();
  LED_DDR = 0xff;

  // ------ Event loop ------ //
  while (1) {

    ADCSRA |= (1 << ADSC);                    /* start ADC conversion */
    loop_until_bit_is_clear(ADCSRA, ADSC);         /* wait until done */
    adcValue = ADC;                                 /* read ADC in */
                        /* Have 10 bits, want 3 (eight LEDs after all) */
    ledValue = (adcValue >> 7);
                                    /* Light up all LEDs up to ledValue */
    LED_PORT = 0;
    for (i = 0; i <= ledValue; i++) {
      LED_PORT |= (1 << i);
    }
    _delay_ms(50);
  }                                                 /* End event loop */
  return (0);                               /* This line is never reached */
}
```

SlowScope

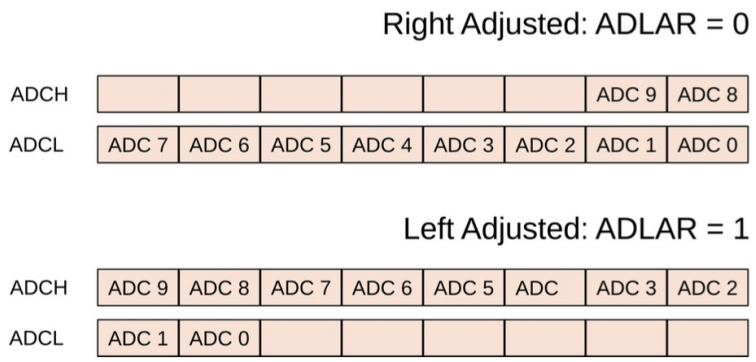


Figure 7-6. *ADC result bit alignment*

Example 7-2. slowScope.c listing

```
// Slow-scope.  A free-running AVR / ADC "oscilloscope"

// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include "pinDefines.h"
#include "USART.h"

#define SAMPLE_DELAY  20 /* ms, controls the scroll-speed of the scope */

// -------- Functions --------- //
static inline void initFreerunningADC(void) {
  ADMUX |= (1 << REFS0);                 /* reference voltage on AVCC */
  ADCSRA |= (1 << ADPS1) | (1 << ADPS0);     /* ADC clock prescaler /8 */

  ADMUX |= (1 << ADLAR);     /* left-adjust result, return only 8 bits */

  ADCSRA |= (1 << ADEN);                              /* enable ADC */
  ADCSRA |= (1 << ADATE);                        /* auto-trigger enable */
  ADCSRA |= (1 << ADSC);                      /* start first conversion */
}

int main(void) {
  // -------- Inits --------- //
  initUSART();
  initFreerunningADC();
  // ------ Event loop ------ //
  while (1) {
    transmitByte(ADCH);       /* transmit the high byte, left-adjusted */
    _delay_ms(SAMPLE_DELAY);
  }                                                   /* End event loop */
  return (0);                             /* This line is never reached */
}
```

Interrupt LED

Example 8-1. helloInterrupt.c listing

```
/*

Demo of using interrupts for doing what they do best --
two things at once.

Flashes LED0 at a fixed rate, interrupting whenever button is pressed.

 */

// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "pinDefines.h"

ISR(INT0_vect) {        /* Run every time there is a change on button */
  if (bit_is_clear(BUTTON_PIN, BUTTON)) {
    LED_PORT |= (1 << LED1);
  }
  else {
    LED_PORT &= ~(1 << LED1);
  }
}

void initInterrupt0(void) {
  EIMSK |= (1 << INT0);                          /* enable INT0 */
  EICRA |= (1 << ISC00);              /* trigger when button changes */
  sei();                          /* set (global) interrupt enable bit */
}

int main(void) {
  // -------- Inits --------- //
  LED_DDR = 0xff;                                /* all LEDs active */
  BUTTON_PORT |= (1 << BUTTON);                          /* pullup */
  initInterrupt0();

  // ------ Event loop ------ //
  while (1) {

    _delay_ms(200);
    LED_PORT ^= (1 << LED0);

  }                                                /* End event loop */
```

Pin Change Interrupt

```
ISR(PCINT2_vect){
  ....
}
```

```
void initPinChangeInterrupt18(void){
  PCICR |= (1 << PCIE2);     /* set pin-change interrupt for D pins */
  PCMSK2 |= (1 << PCINT18);  /* set mask to look for PCINT18 / PD2 */
  // PCMSK2 |= (1 << PD2);   /* this will also work for the pin mask */
  sei();                     /* set (global) interrupt enable bit */
}
```

Name in datasheet	Vector name for ISRs	Which pins are covered
PCINT0	PCINT0_vect	PB0.. PB7
PCINT1	PCINT1_vect	PC0.. PC6
PCINT2	PCINT2_vect	PD0.. PD7

Capacitive Input Sensing

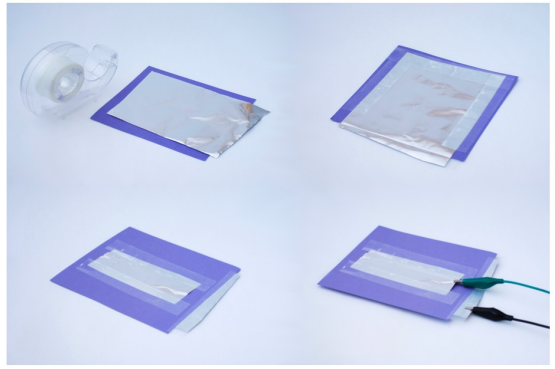


Figure 8-2. *Aluminum foil capacitive sensor*

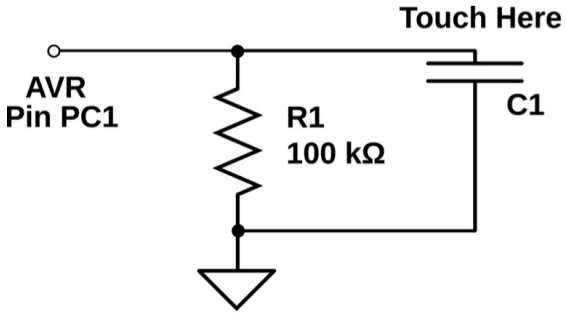


Figure 8-3. *Capacitive sensor circuit*

Example 8-2. capSense.c listing

```
/*
   Capacitive touch sensor demo
*/

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/power.h>
#include "pinDefines.h"
#include "USART.h"

#define SENSE_TIME   50
#define THRESHOLD    12000

// -------  Global Variables ---------- //
volatile uint16_t chargeCycleCount;

// ------- Functions -------- //

void initPinChangeInterrupt(void) {
  PCICR |= (1 << PCIE1);    /* enable Pin-change interrupts 1 (bank C) */
  PCMSK1 |= (1 << PC1);   /* enable specific interrupt for our pin PC1 */
}

ISR(PCINT1_vect) {
  chargeCycleCount++;                                  /* count this change */
}
```

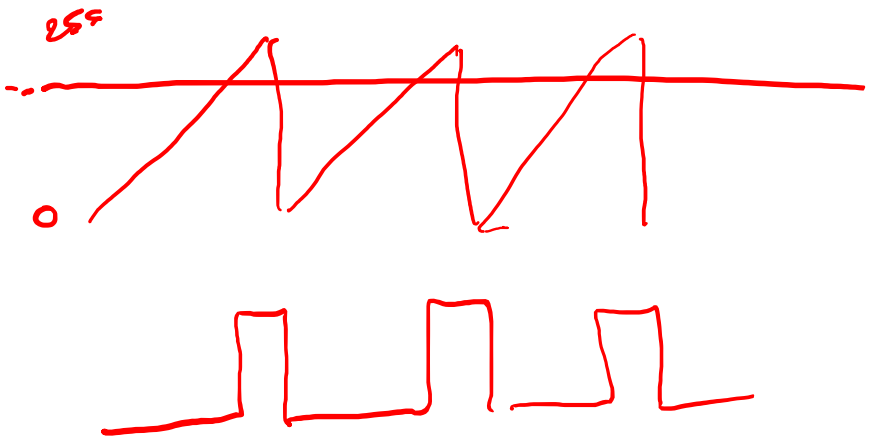
```
  CAP_SENSOR_DDR |= (1 << CAP_SENSOR);                 /* output mode */
  _delay_us(1);                                       /* charging delay */

  CAP_SENSOR_DDR &= ~(1 << CAP_SENSOR);                /* set as input */
  PCIFR |= (1 << PCIF1);              /* clear the pin-change interrupt */
}


int main(void) {
  // -------- Inits --------- //
  clock_prescale_set(clock_div_1);                        /* full speed */
  initUSART();
  printString("==[ Cap Sensor ]==\r\n\r\n");

  LED_DDR = 0xff;
  MCUCR |= (1 << PUD);                            /* disable all pullups */
  CAP_SENSOR_PORT |= (1 << CAP_SENSOR);    /* we can leave output high */

  initPinChangeInterrupt();

  // ------ Event loop ------ //
  while (1) {

    chargeCycleCount = 0;                                 /* reset counter */
    CAP_SENSOR_DDR |= (1 << CAP_SENSOR);     /* start with cap charged */
    sei();                                 /* start up interrupts, counting */
    _delay_ms(SENSE_TIME);
    cli();                                                        /* done */
    if (chargeCycleCount < THRESHOLD) {
      LED_PORT = 0xff;
    }
    else {
      LED_PORT = 0;
    }
    printWord(chargeCycleCount);                    /* for fine tuning */
    printString("\r\n");

  }                                                    /* End event loop */
  return (0);                              /* This line is never reached */
}
```

Reaction Timer

```
// ------- Preamble -------- //
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "pinDefines.h"
#include "USART.h"

#include "support.h"

static inline void initTimer1(void) {
                                  /* Normal mode (default), just counting */
  TCCR1B |= (1 << CS11) | (1 << CS10);
  /* Clock speed: 1 MHz / 64,
     each tick is 64 microseconds ~= 15.6 per ms  */
                                              /* No special output modes */
}

int main(void) {
  char byte;
  uint16_t timerValue;

  // -------- Inits --------- //

  initUSART();
  initTimer1();
  LED_DDR = 0xff;                                /* all LEDs for output */
  BUTTON_PORT |= (1 << BUTTON);                /* enable internal pull-up */

  printString("\r\nReaction Timer:\r\n");
  printString("---------------\r\n");
  printString("Press any key to start.\r\n");

  // ------ Event loop ------ //
  while (1) {

    byte = receiveByte();                                /* press any key */
    printString("\r\nGet ready...");
    randomDelay();

    printString("\r\nGo!\r\n");
    LED_PORT = 0xff;                                       /* light LEDs */
    TCNT1 = 0;                                           /* reset counter */

    if (bit_is_clear(BUTTON_PIN, BUTTON)) {
           /* Button pressed _exactly_ as LEDs light up.  Suspicious. */
      printString("You're only cheating yourself.\r\n");
    }
    else {
      // Wait until button pressed, save timer value.
      loop_until_bit_is_clear(BUTTON_PIN, BUTTON);
      timerValue = TCNT1 >> 4;
      /* each tick is approx 1/16 milliseconds, so we bit-shift divide */
```

Song using timer

```c
// ------- Preamble -------- //
#include <avr/io.h>                        /* Defines pins, ports, etc */
#include <util/delay.h>                     /* Functions to waste time */
#include "pinDefines.h"
#include "scale8.h"                                     /* 8-bit scale */

static inline void initTimer(void) {
  TCCR0A |= (1 << WGM01);                                 /* CTC mode */
  TCCR0A |= (1 << COM0A0);           /* Toggles pin each cycle through */
  TCCR0B |= (1 << CS00) | (1 << CS01);              /* CPU clock / 64 */
}

static inline void playNote(uint8_t wavelength, uint16_t duration) {

  OCR0A = wavelength;                                     /* set pitch */
  SPEAKER_DDR |= (1 << SPEAKER);          /* enable output on speaker */

  while (duration) {                                /* Variable delay */
    _delay_ms(1);
    duration--;
  }
  SPEAKER_DDR &= ~(1 << SPEAKER);                /* turn speaker off */
}

int main(void) {
  // -------- Inits --------- //
  initTimer();
  // ------ Event loop ------ //
  while (1) {
                                                 /* Play some notes */
    playNote(C2, 200);
    playNote(E2, 200);
    playNote(G2, 200);
    playNote(C3, 400);

    _delay_ms(1000);
    _delay_ms(1000);
    _delay_ms(1000);

  }                                              /* End event loop */
  return (0);                           /* This line is never reached */
}
```



PWM using timers

Example 10-2. **pwmTimers.c Listing**

```
                    /* PWM Demo with serial control over three LEDs */

// ------- Preamble -------- //
#include <avr/io.h>                        /* Defines pins, ports, etc */
#include <util/delay.h>                     /* Functions to waste time */
#include "pinDefines.h"
#include "USART.h"

static inline void initTimers(void) {
  // Timer 1 A,B
  TCCR1A |= (1 << WGM10);                     /* Fast PWM mode, 8-bit */
  TCCR1B |= (1 << WGM12);                      /* Fast PWM mode, pt.2 */
  TCCR1B |= (1 << CS11);                     /* PWM Freq = F_CPU/8/256 */
  TCCR1A |= (1 << COM1A1);                    /* PWM output on OCR1A */
  TCCR1A |= (1 << COM1B1);                    /* PWM output on OCR1B */

  // Timer 2
  TCCR2A |= (1 << WGM20);                           /* Fast PWM mode */
  TCCR2A |= (1 << WGM21);                      /* Fast PWM mode, pt.2 */
  TCCR2B |= (1 << CS21);                     /* PWM Freq = F_CPU/8/256 */
  TCCR2A |= (1 << COM2A1);                    /* PWM output on OCR2A */
}

int main(void) {

  uint8_t brightness;

  // -------- Inits --------- //

  initTimers();
  initUSART();
  printString("-- LED PWM Demo --\r\n");

              /* enable output on LED pins, triggered by PWM hardware */
  LED_DDR |= (1 << LED1);
  LED_DDR |= (1 << LED2);
  LED_DDR |= (1 << LED3);

  // ------ Event loop ------ //
  while (1) {

    printString("\r\nEnter (0-255) for PWM duty cycle: ");
    brightness = getNumber();
    OCR2A = OCR1B;
    OCR1B = OCR1A;
    OCR1A = brightness;

  }                                                 /* End event loop */
  return (0);                              /* This line is never reached */
}
```

Could this have worked otherwise?

Can we run PWM on any pin?

Bootloaders

19. Self-Programming the Flash

The device provides a Self-Programming mechanism for downloading and uploading program code by the MCU itself. The Self-Programming can use any available data interface and associated protocol to read code and write (program) that code into the Program memory. The SPM instruction is disabled by default but it can be enabled by programming the SELFPRGEN fuse (to “0”).

The Program memory is updated in a page by page fashion. Before programming a page with the data stored in the temporary page buffer, the page must be erased. The temporary page buffer is filled one word at a time using SPM and the buffer can be filled either before the Page Erase command or between a Page Erase and a Page Write operation:

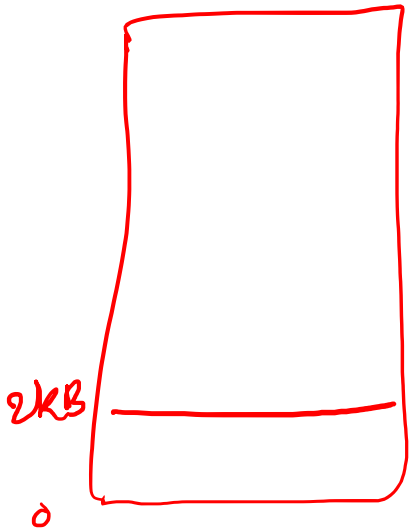
Alternative 1, fill the buffer before a Page Erase

- Fill temporary page buffer
- Perform a Page Erase
- Perform a Page Write

Alternative 2, fill the buffer after Page Erase

- Perform a Page Erase
- Fill temporary page buffer
- Perform a Page Write

If only a part of the page needs to be changed, the rest of the page must be stored (for example in the temporary page buffer) before the erase, and then be re-written. When using alternative 1, the Boot Loader provides an effective Read-Modify-Write feature which allows the user software to first read the page, do the necessary changes, and then write back the modified data. If alternative 2 is used, it is not possible to read the old data while loading since the page is already erased. The temporary page buffer can be accessed in a random sequence. It is essential that the page address used in both the Page Erase and Page Write operation is addressing the same page.



Code preview

Beyond AVR

BIOS is the bootloader for your PC

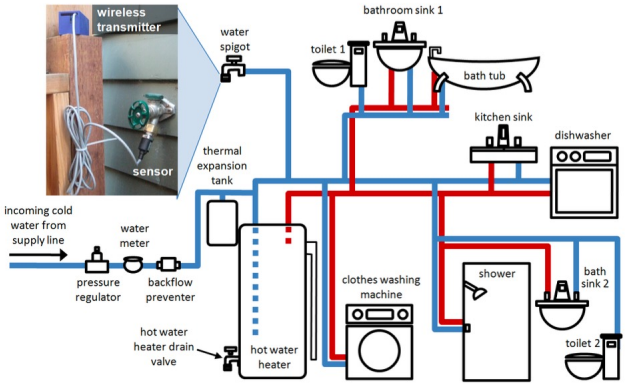
- POST (Power on self test)
- Loads OS into the memory
- Initializes drivers for basic devices

Ch2-bootloader.pdf (nctu.edu.tw)

Other types of boot

- Bootloaders for **Multiple** Architectures
  - **U-Boot** (http://sourceforge.net/projects/u-boot/)
    - Universal Boot loader supports PowerPC, ARM, MIPS, …
  - **RedBoot** (http://www.redhat.com/embedded/technologies/redboot/)
    - By RedHat. capable of flash and network booting of Linux kernel.
    - Supports ARM, MIPS, PowerPC, Hitachi SHx, and x86.
  - **Smart Firmware** (http://www.codegen.com/SmartFirmware/)
    - By CodeGen. It is designed to be very easy and fast to port.
    - Supports M68K, PowerPC, x86, MIPS, Alpha, Sparc, ARM, …
  - **MicroMonitor** (http://www.linuxdevices.com/links/LK3381469889.html)
    - A open source embedded system boot platform. The majority of the code is CPU and platform independent and has been used with a variety of different embedded operating systems.
    - x86, Mips, PowerPC, SH, ARM, and M68K.

Devices in making



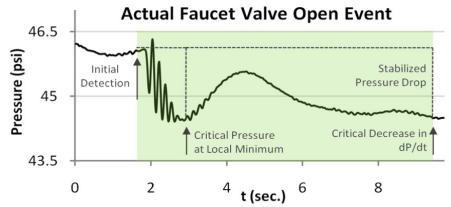


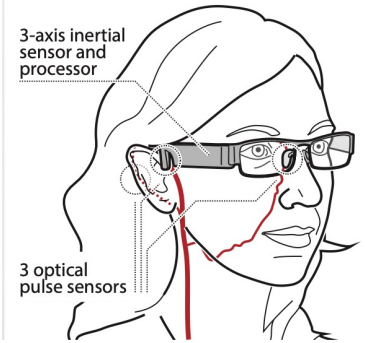
Figure 2: An actual event generated when a kitchen faucet is turned on, detected by our sensor at an exterior water bib. Our fixture identification is based in segmenting the event (indicated by the highlight) and then classifying it according to its shape (see also Figure 3 and Figure 6).

HydroSense '17
Shwetak Patel, UW

Self-arranging audio zones
Shyamnath, UW

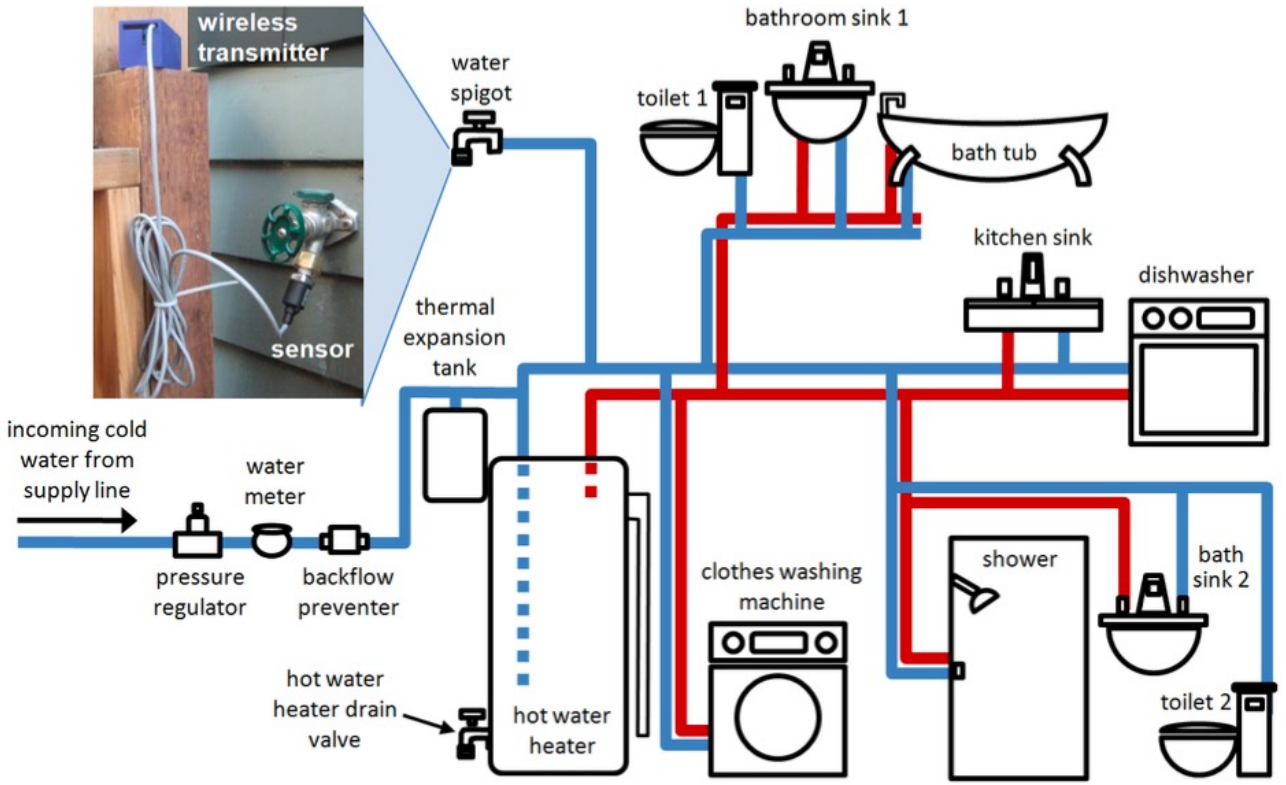
Hearing Test for babies
UW

Figure 1: OAEbuds in use with an infant. Our low-cost wireless earbud can perform hearing screening by detecting otoacoustic emissions (OAE) from the cochlea.



Blood Pressure Measurement
Christian Holz, MSR

HydroSense


wireless transmitter
sensor
water spigot
bathroom sink 1
toilet 1
bath tub
kitchen sink
dishwasher
thermal expansion tank
incoming cold water from supply line
water meter
pressure regulator
backflow preventer
hot water heater drain valve
hot water heater
clothes washing machine
shower
bath sink 2
toilet 2

The pressure wave

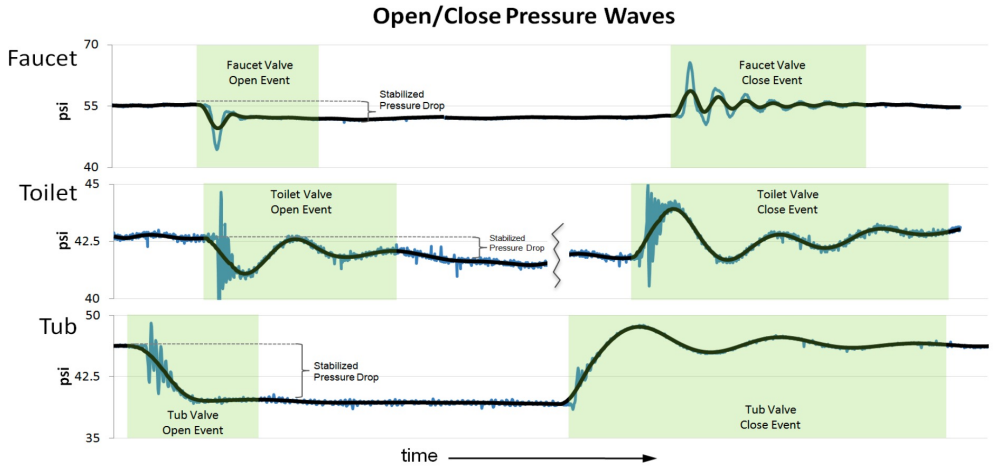


Figure 6: Several actual sensor streams from our in-home data collections. Each stream corresponds to a water valve being opened, remaining open for some amount of time, and then closed. We separately segment the valve open and valve close events from the sensor stream, as indicated by the highlighted regions of the streams. We estimate water flow to the valve based on the stabilized pressure drop while the valve remains open (the difference in pressure before the valve open versus after it).

$$Q = \frac{\Delta P\, \pi\, r^4}{8\, \mu\, L} \qquad Q = \frac{\Delta P}{R_f}$$

Our setup

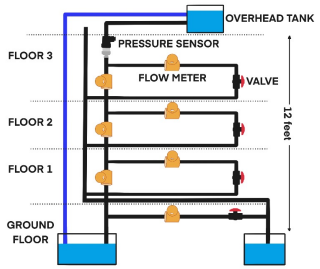


Figure 2: Experimental setup consisting of taps installed on 4 floors and a pressure sensor installed at the top end of the main pipe.

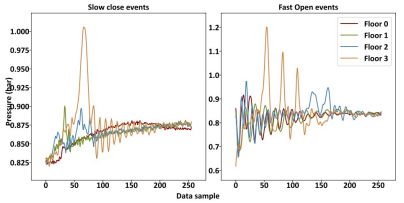


Figure 3: Resampled pressure waveforms recorded by the test setup.

The hardware

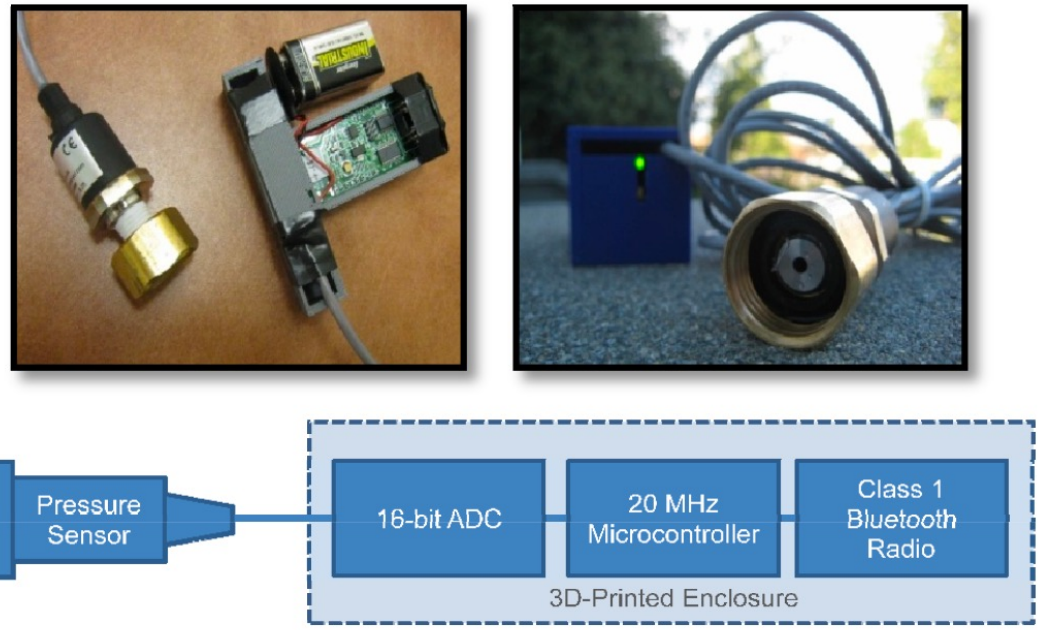


Figure 4: Our prototype sensor implementation. The sensor twists on to a fixture and communicates wirelessly.

The ML

Data

ID / Water Supply	Style / Built / Remodel	Size / Floors / Fixtures	Exp. Tank/ Regulator / Recirc. Pump	Water Heater / Plumbing/ Static PSI	Sensor Install Point
H1 Public Utility	Single-Family 2002	3200 sqft 2 flr + bas 12 fixture	Yes Yes No	Tank PVC 46 PSI	Hose Bib
H2 Public Utility	Multi-Family 1909/96	2160 sqft 2 flr + bas 5 fixtures	No No No	Tankless Copper 46 PSI	Hose Bib
H3 Public Utility	Single-Family 2003	4000 sqft 2 flr + bas 6 fixtures	Yes Yes No	Tank Copper 41 PSI	Hose Bib
H4 Public Utility	Single-Family 1921	1630 sqft 1 flr + bas 4 fixtures	No No No	Tank Galvan. 43 PSI	Hose Bib
H5 Public Utility	Single-Family 1913	2000 sqft 2 flr + bas 5 fixtures	No No No	Tank Copper 55 PSI	Hose Bib
H6 Public Utility	Single-Family 1974/85	3100 sqft 2 flr 8 fixtures	Yes Yes Yes	Tank Galvan. 46 PSI	Hose Bib
H7 Public Utility	Aprtmnt 1927	746 sqft 1 flr 5 fixtures	No Yes No	Tank Cop+Gal 33 PSI	Water Heater
H8 Public Utility	Single-Family 1922 / 2006	3650 sqft 2 flr + bas 3 fixtures	Yes Yes Yes	Tank Copper 75 PSI	Utility Sink Faucet
H9 Public Utility	Single-Family 1904 / 95 est.	1790 sqft 2 flr + bas 4 fixtures	No No No	Tank Copper 72 PSI	Hose Bib + Water Heater
H10 Private Well	Resort Cabin 1950/80	900 sqft 1 flr 4 fixtures	No No No	Tank Galvan. 65 PSI	Hose Big

Figure 5: A summary of the homes in which we collected data, including the style, size (1 sqft ≈ .093 sqm), age of the home, how many fixtures we tested, characteristics of the plumbing system, and where we installed our sensor.

Algorithms

Valve Segmentation
- Filtering + rules

Valve Classification
- Pressure decrease/increase

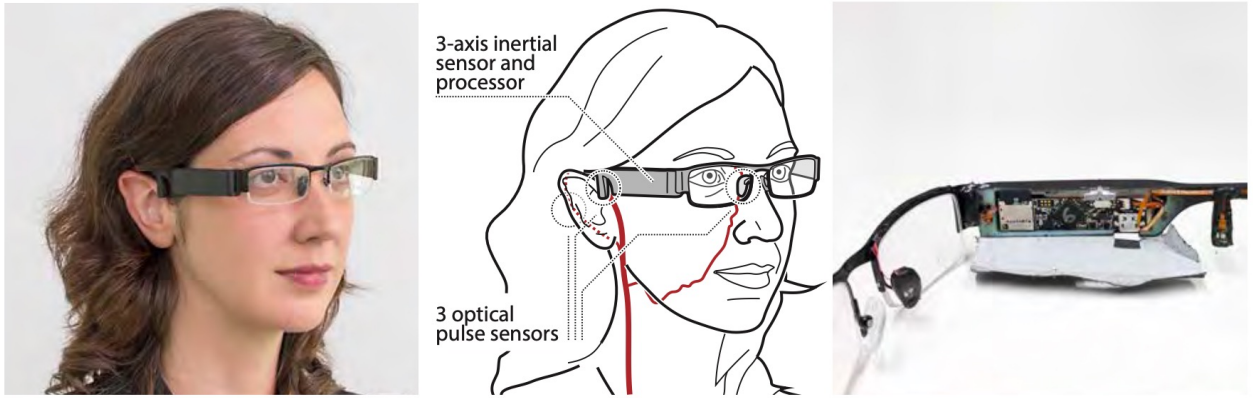
Fixture Classification
Template matching based on 4 criteria

1. Matched filter
2. Derivative Matched Filter
3. Cepstrum (lower values resemble transfer function of a pipe)
4. MSE

Home	Fixture Open Identification	Fixture Close Identification
H1 (12 valves)	100%	100%
H2 (8 valves)	96.4%	100%
H3 (6 valves)	100%	100%
H4 (5 valves)	96.2%	100%
H5 (9 valves)	100%	100%
H6 (8 valves)	100%	90.0%
H7 (8 valves)	100%	100%
H8 (6 valves)	100%	97.1%
H9 (7 valves)	97.1%	97.1%
H10 (7 valves)	97.1%	77.1%
Aggregate	98.9%	96.8%
	97.9%	

different types of fixtures across homes. Our overall fixture-level classification across all homes is above 90%, including a number of cases where classification accuracy is 100%. All of these results are equal to or better than prior

Non-invasive Blood Pressure



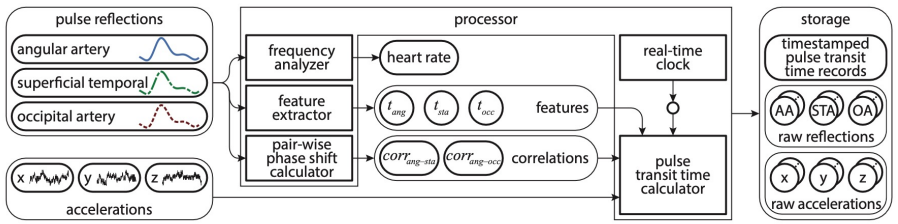


Fig. 2. Overview of the wearable platform we built: Each prototype senses the reflections of the wearer's pulse in three locations as well as the 3D inertial motions of the head through the frame. From the input signals, the prototype extracts the wearer's heart rate and the temporal features in the pulse reflections, from which we calculate the pulse transit time for each beat. Analyzing device motions and the cross-correlation across the reflections allows us to reject inaccurate candidate beats in the raw pulse signals. Finally, Glabella stores the raw signal streams along with the computed features, heart rates, and pulse transit times on an SD card.

Hardware

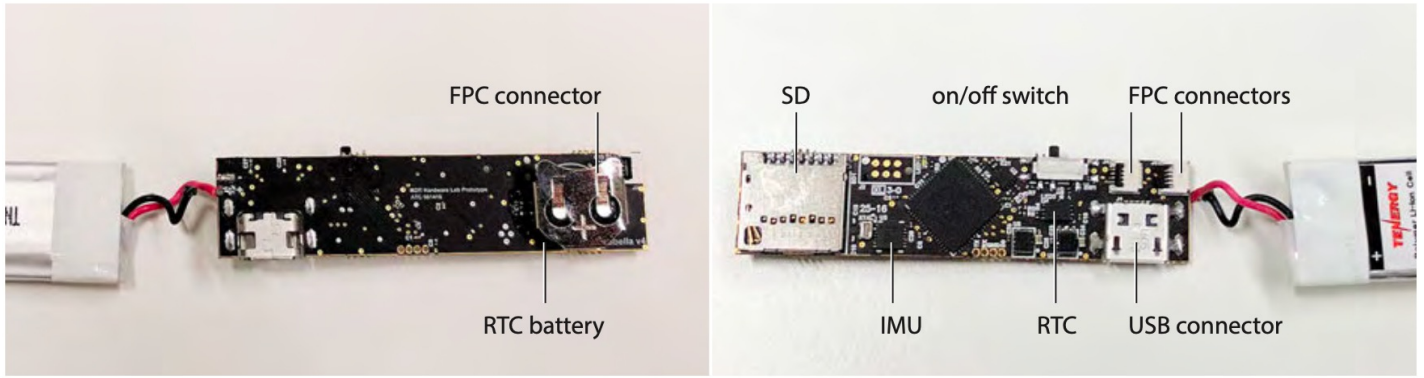


Fig. 6. Glabella's sensor boards incorporate a photodiode, a corresponding and spectrum-matching green LED, and a small opamp circuit. The PSoC-integrated ADC on the main board digitizes all signals in synchrony. (Left) Custom FPC cables connect the two sensor boards in the frame to the main board. (Right) Thin magnet wires connect the sensor board that is embedded in the nose pad to the main board, routed along the frame for minimal visibility.

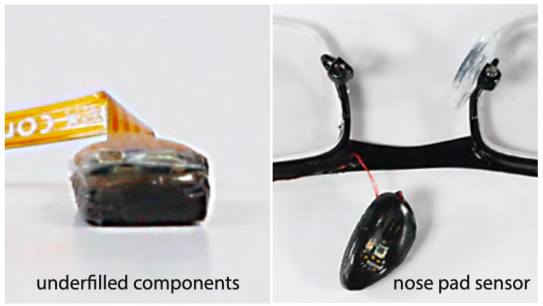


Fig. 7. To limit the exposure of our sensor boards to the user's skin and protect them from external influences, such as sweat, we *underfilled* all components using fine traces of hot glue that we manually reflowed. Using an underfill keeps the main components of the sensor board exposed to make direct contact with the wearer's skin, such that the LED and the photodiode optimally inject light into the skin and observe optimal reflections, respectively.

Hearing screening for babies

- No behavioral response
  - Rely on octoacoustic emissions

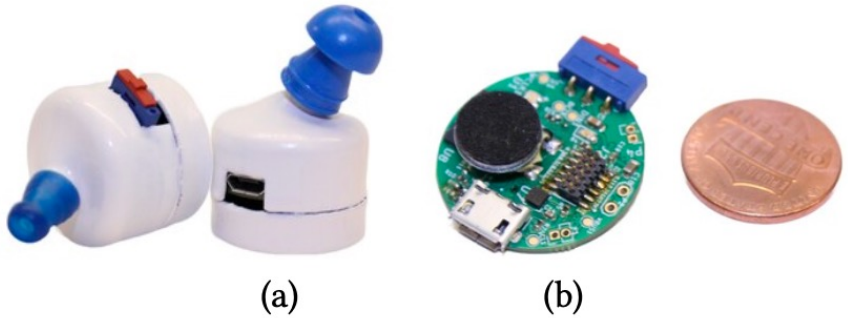


Figure 2: OAEbuds hardware. (a) The 3D-printed enclosure with pediatric and adult earbud tips. (b) OAEbud circuit board beside a penny for size comparison.

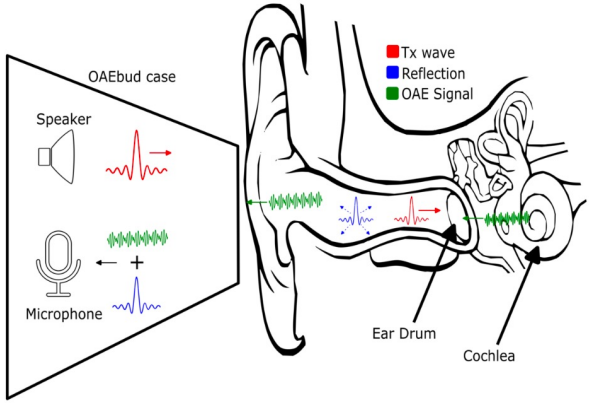


Figure 3: In-ear signal propagation. The OAEbud plays a broadband transmit (Tx) pulse to stimulate the cochlea to emit an OAE signal. The signal received by microphone is a superposition of 1) unwanted reflections from the ear canal, eardrum, and within the case and 2) the OAE signal.

uw-x/OAEbuds: About Firmware to collect OAE data from wireless earbuds (github.com)

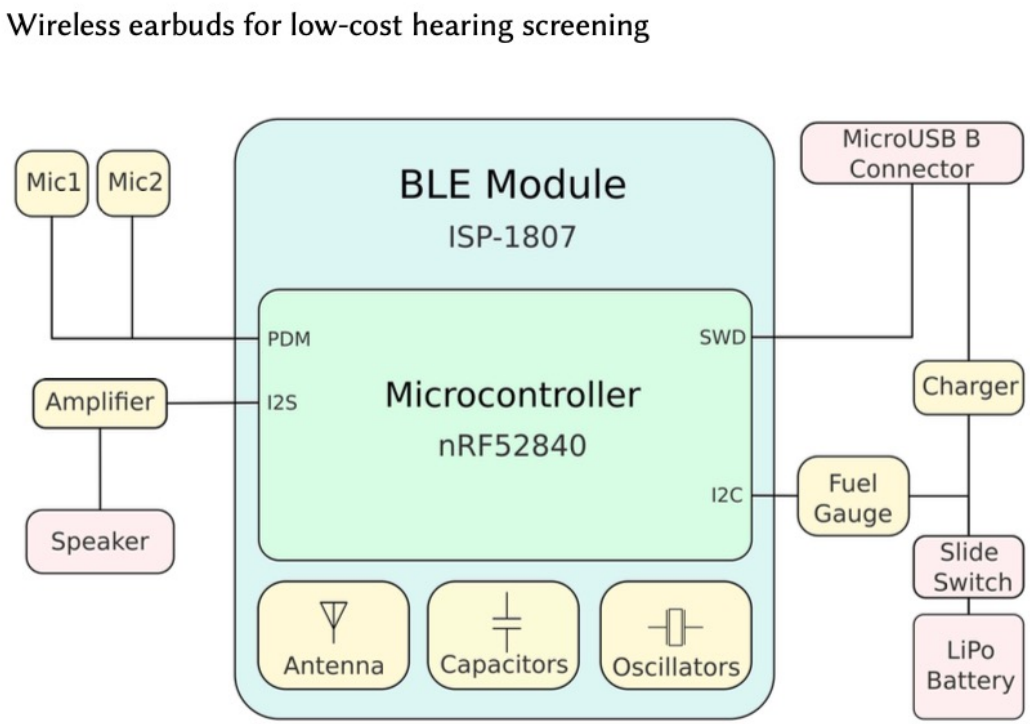
Hardware

I2S
Inter Integrated Sound

PDM
Pulse Density Modulation

SWD
Serial Wire Debug

Fuel Gauge



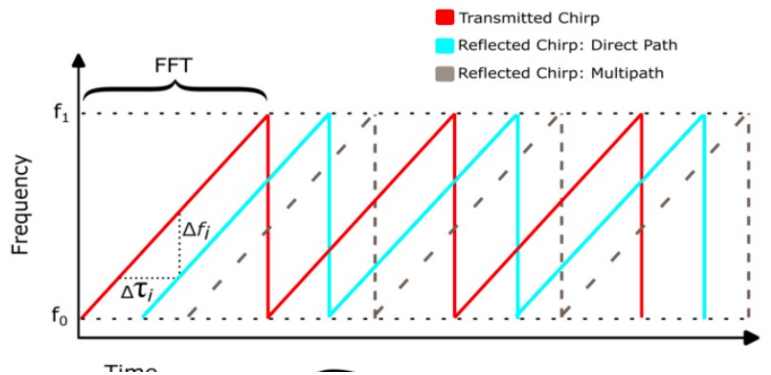
The ML

Algorithm

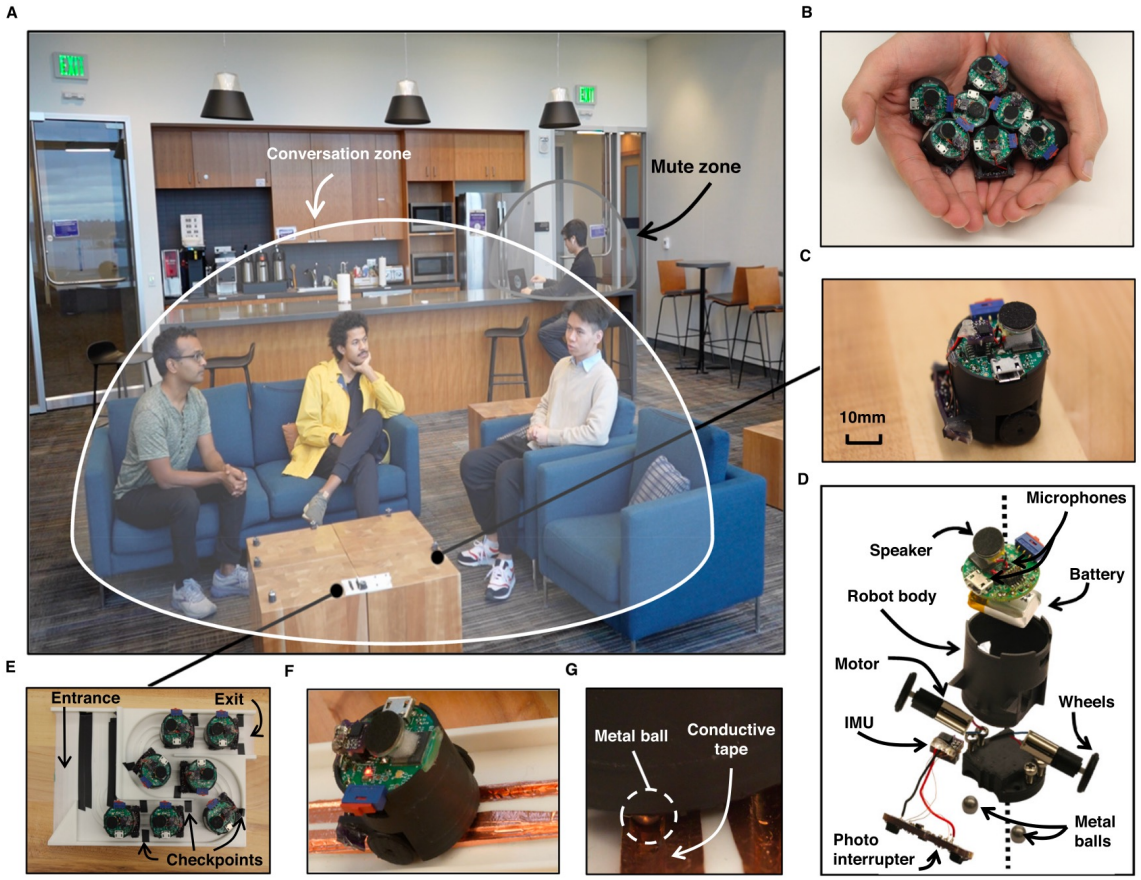
- Reflection time estimation
  - Time delay estimation to estimate position of cochlea
- OAE signal extraction
  - Wideband pulses filtered for distance estimated in previous step
  - Response measured against different frequencies

Result

OAEbuds achieves a sensitivity of 100% and specificity of 89.7% in screening for hearing loss. In comparison, the FDA-cleared medical device achieves a sensitivity of 83.3% and specificity of 92.1%. Our



Self-arranging audio zones

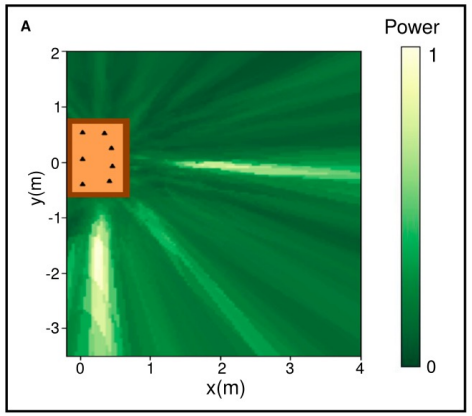


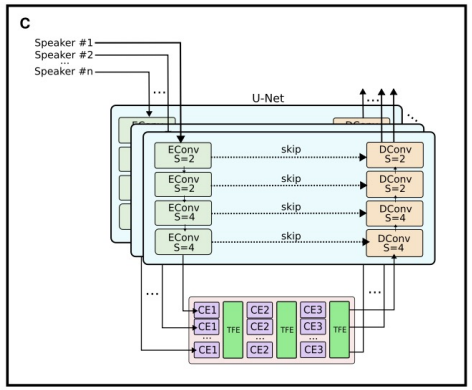
- Listen from multiple locations
- Charge thyself
- Arrange thyself
  - Self-localization through audio chirps
- Audio streamed to a computer for processing:
  - Includes onboard Opus codec for audio compression

Speech separation and localization

Coarse Localization using signal processing

Inter-speaker localization





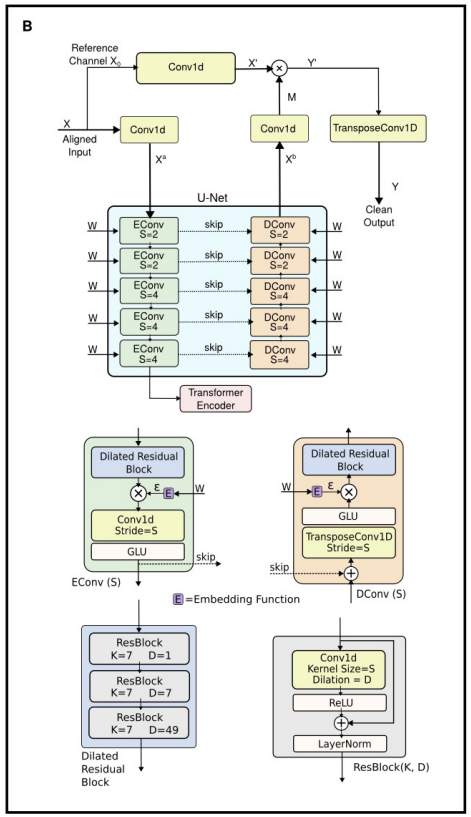
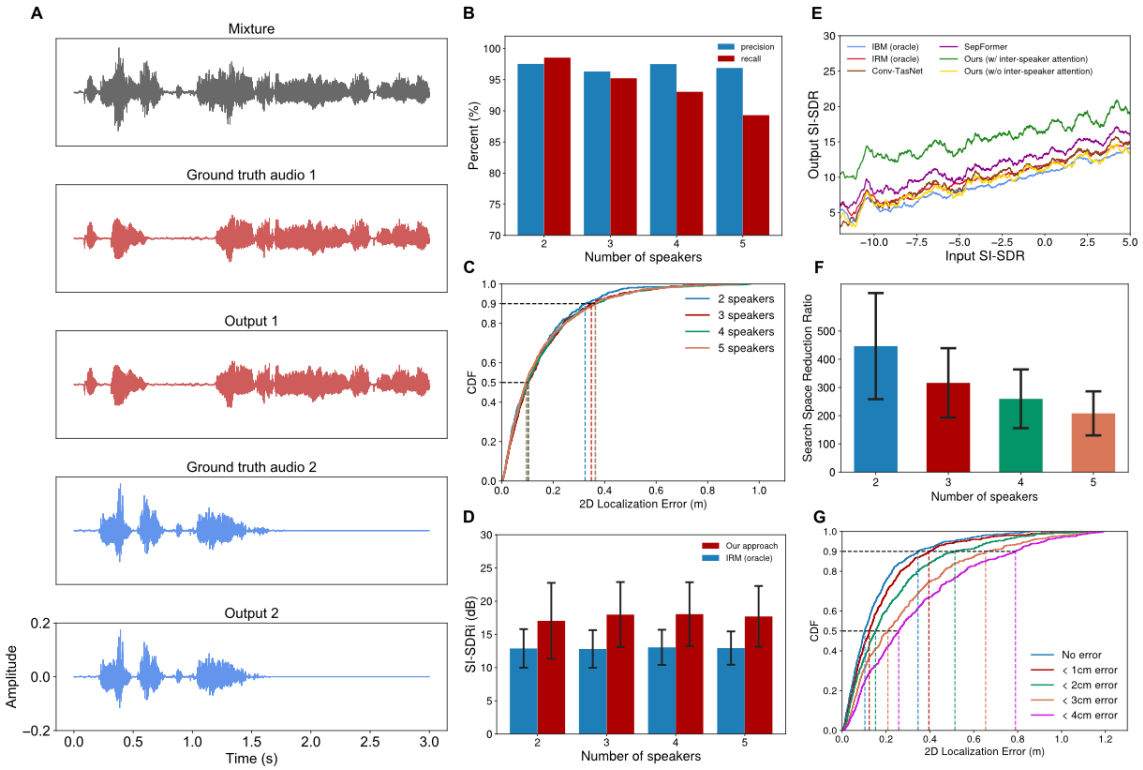


Fig. 5 | Joint 2D localization and speech separation framework. A We first run the SRP-PHAT algorithm to prune the search space, and then in (**B**) we use an attention-based separation model to find the potential speaker locations in the remaining space. The separation model is composed of a U-Net encoder-decoder with a transformer encoder bottleneck between them. GLU stands for Gated Linear Unit. **C** shows our network used for speech separation. The encoder and decoder blocks are applied separately to the aligned microphone data for each of the speakers. The bottleneck block first applies temporal self-attention to each speaker individually using a conformer encoder (CE). It then applies self-attention across speakers using a transformer encoder (TFE) to compute attention weights across different speakers. It repeats this multiple times to address cross-talk between speakers.

Source separation (listen to each speaker)



Want to make one?

Data availability

The data used for our machine learning models have been deposited in three parts at https://zenodo.org/record/8219720, https://zenodo.org/record/8222714 and https://zenodo.org/record/8222784 under a Creative Commons Attribution 4.0 International License.

Code availability

We provide the circuit design files used to create the robots, as well as the firmware source code at https://github.com/uw-x/AcousticSwarms-Robots. We also provide the source code for the speech processing algorithms at https://github.com/uw-x/AcousticSwarms-Speech.

SmartCyPad

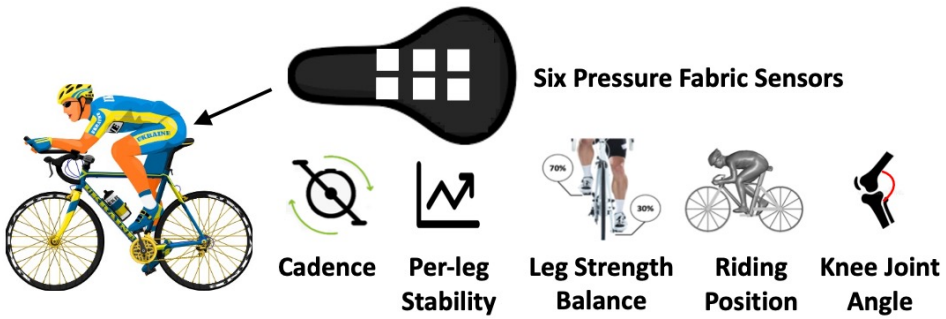
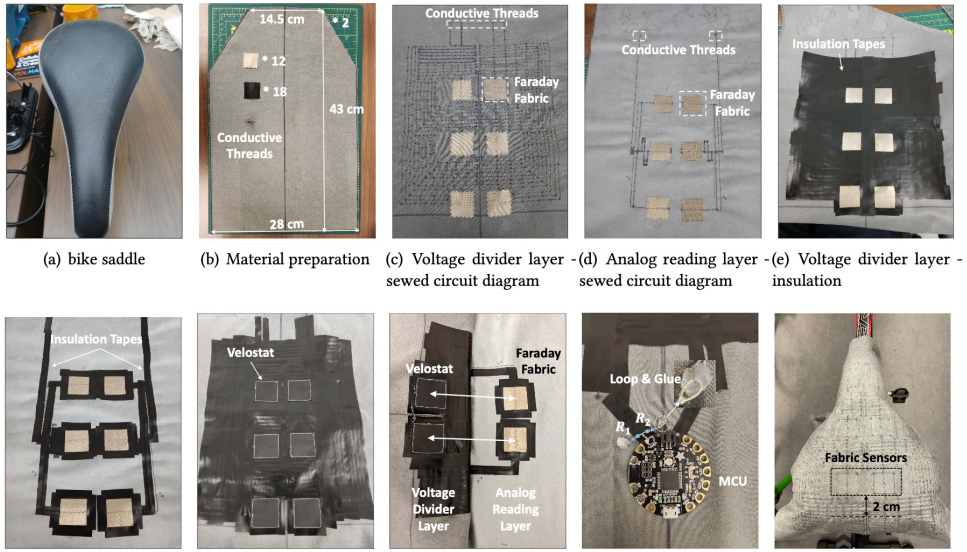


Fig. 1. Estimating five cycling-specific metrics using SmarCyPad.



(a) bike saddle (b) Material preparation (c) Voltage divider layer - sewed circuit diagram (d) Analog reading layer - sewed circuit diagram (e) Voltage divider layer - insulation

(f) Analog reading layer - insulation (g) Voltage divider layer - add Velostat (h) Combine two layers together (i) MCU connection (j) Fit the prototype to the bike saddle

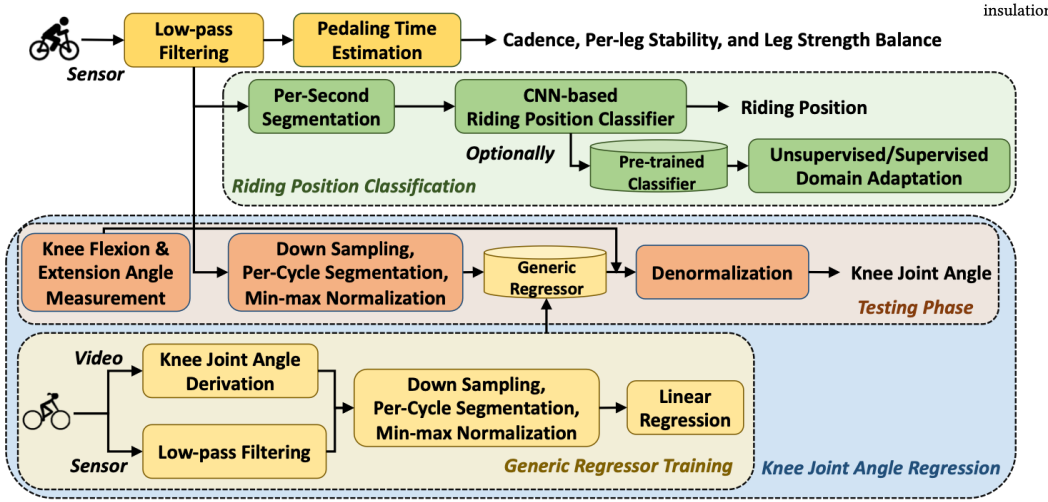


Fig. 3. SmarCyPad system overview.

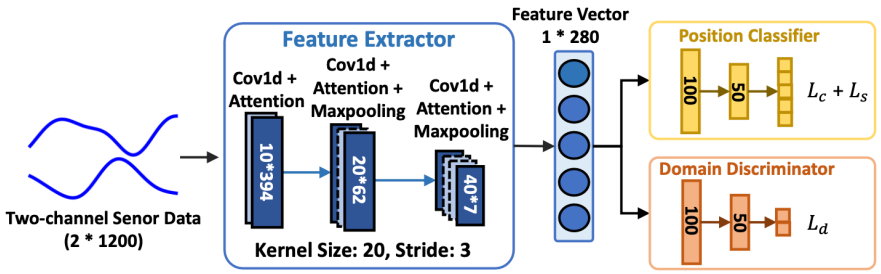


Fig. 12. 1D CNN-based riding position classification network.

References

Chan, J., Glenn, A., Itani, M., Mancl, L. R., Gallagher, E., Bly, R., Patel, S., & Gollakota, S. (2023). Wireless earbuds for low-cost hearing screening. *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*, 84–95. https://doi.org/10.1145/3581791.3596856

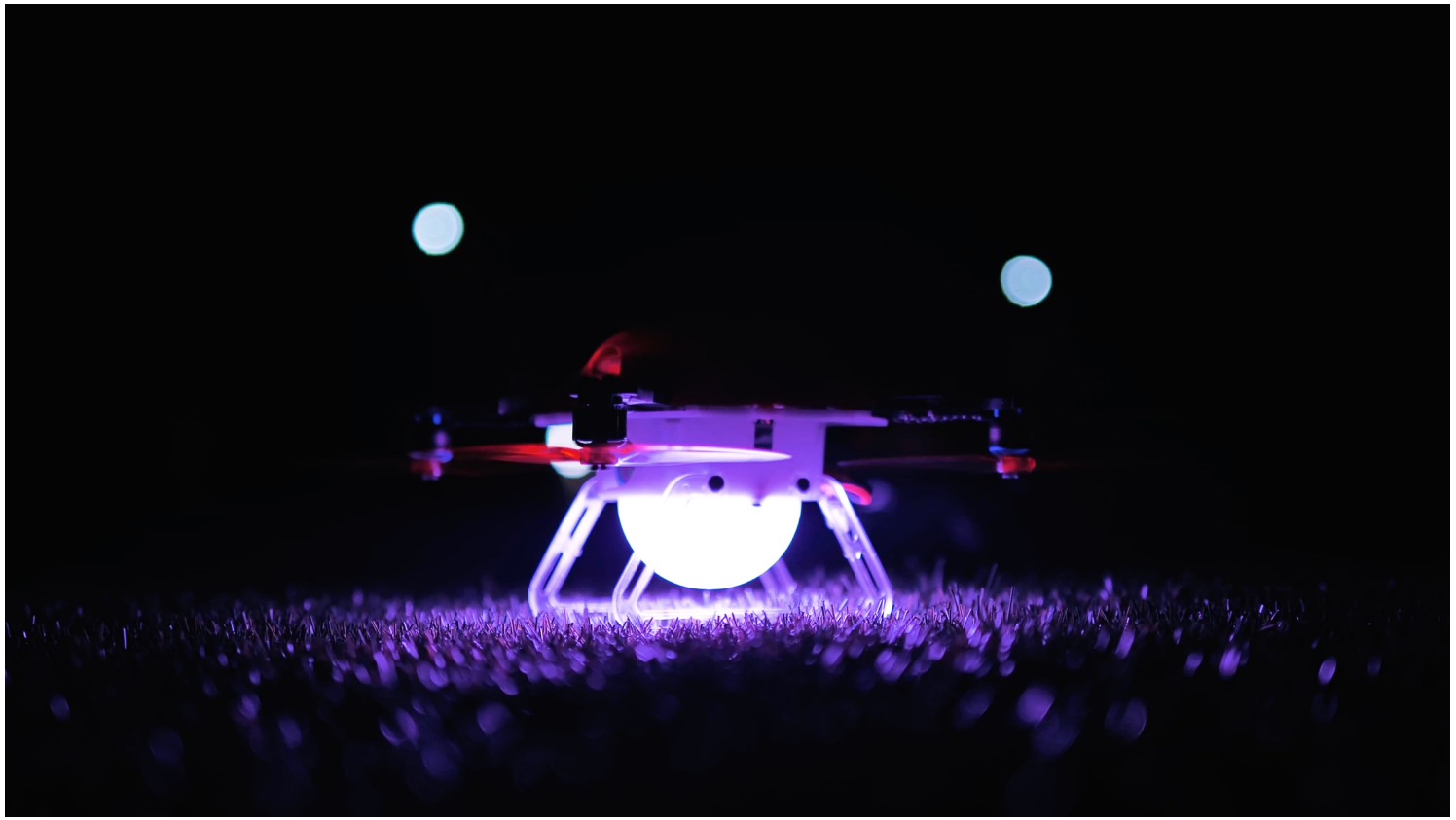
Froehlich, J. E., Larson, E., Campbell, T., Haggerty, C., Fogarty, J., & Patel, S. N. (2009). HydroSense: Infrastructure-mediated single-point sensing of whole-home water activity. *Proceedings of the 11th International Conference on Ubiquitous Computing*, 235–244. https://doi.org/10.1145/1620545.1620581

Holz, C., & Wang, E. J. (2017). Glabella: Continuously Sensing Blood Pressure Behavior using an Unobtrusive Wearable Device. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *1*(3), 1–23. https://doi.org/10.1145/3132024

Itani, M., Chen, T., Yoshioka, T., & Gollakota, S. (2023). Creating speech zones with self-distributing acoustic swarms. *Nature Communications*, *14*(1), Article 1. https://doi.org/10.1038/s41467-023-40869-8

Wu, Y., Villalobos, L. A. G., Yang, Z., Croisdale, G. T., Karataş, Ç., & Liu, J. (2023). SmarCyPad: A Smart Seat Pad for Cycling Fitness Tracking Leveraging Low-cost Conductive Fabric Sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *7*(3), 1–26. https://doi.org/10.1145/3610927

Drone shows



Drone Light Show in India | BotLab Dynamics

Controller

Feature	nRF52840	ATmega328
Manufacturer/Family	Nordic Semiconductor / nRF52 Series	Microchip Technology (Atmel) / AVR
Architecture	ARM Cortex-M4 (32-bit)	AVR 8-bit
Processing Power	More powerful	Less powerful
Flash Memory	Larger options available	Limited (typically 32 KB)
RAM	Larger options available	Limited (typically 2 KB)
Peripherals	Wide range, including BLE, USB, I2C, SPI, UART	Basic peripherals, timers, ADC, UART, I2C
Connectivity	Bluetooth Low Energy (BLE), USB	External modules required for wireless communication
Power Consumption	Designed for low power applications	Less power-efficient
Development Ecosystem	Comprehensive Nordic ecosystem	Arduino community and resources

Aspect	nRF52840	ATmega328
Pros	- More processing power (ARM Cortex-M4)	- Simple and easy to use (common in Arduino)
	- Larger Flash and RAM options	- Low cost and widely available
	- Bluetooth Low Energy (BLE) capabilities	- Well-established Arduino ecosystem
	- Rich set of peripherals and features	- Lower power consumption (in specific use cases)
	- Designed for low-power applications	- Sufficient for many basic projects
Cons	- May be overkill for simple projects	- Limited processing power for complex tasks
	- Can have a steeper learning curve	- Limited Flash and RAM resources
	- Potentially higher cost	- Lacks built-in wireless connectivity
	- Requires understanding of wireless protocols	Fewer built-in peripherals

- ChatGPT