Sensors and Sensing
Microcontrollers, I/O Programming and Signals

Todor Stoyanov

Centre for Applied Autonomous Sensor Systems
Örebro University
Sweden

Outline

1 Background

2 Microcontrollers

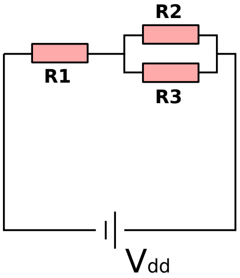
Outline

1 Background

2 Microcontrollers

DC circuits

- Direct current circuits are the base for most digital circuits we use to read sensor data.
- As such, it is good to remember the basics.
- For example: what is the voltage drop over resistor R_1 (i.e., V_1)?

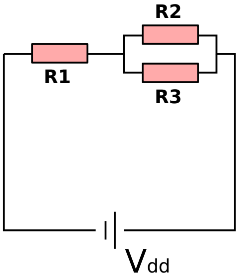


Voltage, Resistance and Current

- All we need to analyze this circuit is Ohm's law and some basic knowledge of physics.
- We know that:

$$V = RI$$

- We also know that $V_2 = V_3 = V_{23}$
- Thus, $R_2 I_2 = R_3 I_3$ and $I_2 = \frac{R_3}{R_2} I_3$

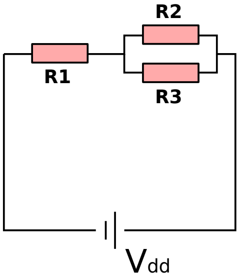


Voltage, Resistance and Current

- All we need to analyze this circuit is Ohm's law and some basic knowledge of physics.
- We know that:

$$V = RI$$

- We also know that $V_2 = V_3 = V_{23}$
- Thus, $R_2 I_2 = R_3 I_3$ and $I_2 = \frac{R_3}{R_2} I_3$

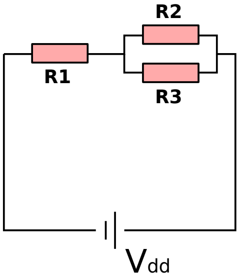


Voltage, Resistance and Current

- All we need to analyze this circuit is Ohm's law and some basic knowledge of physics.
- We know that:

$$V = RI$$

- We also know that $V_2 = V_3 = V_{23}$
- Thus, $R_2 I_2 = R_3 I_3$ and $I_2 = \frac{R_3}{R_2} I_3$

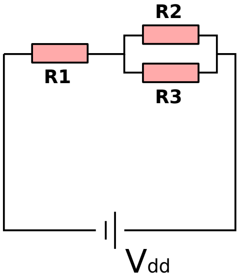


Voltage, Resistance and Current

- All we need to analyze this circuit is Ohm's law and some basic knowledge of physics.
- We know that:

$$V = RI$$

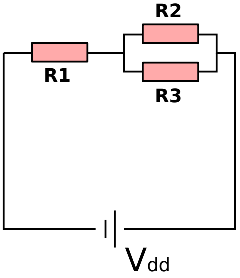
- We also know that $V_2 = V_3 = V_{23}$
- Thus, $R_2 I_2 = R_3 I_3$ and $I_2 = \frac{R_3}{R_2} I_3$



Voltage, Resistance and Current

- Substituting into Ohm's law for V_{23}, we have

$$(I_3 + \frac{R_3}{R_2} I_3) R_{23} = V_{23} = V_3 = I_3 R_3$$



- Which results in

$$R_{23} = \frac{R_2 R_3}{R_2 + R_3}$$

or

$$\frac{1}{R_{tot}} = \frac{1}{R_1} + \frac{1}{R_2}$$

Voltage, Resistance and Current

- Substituting into Ohm's law for V_{23}, we have

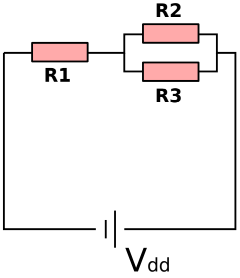
$$(I_3 + \frac{R_3}{R_2} I_3) R_{23} = V_{23} = V_3 = I_3 R_3$$

- Which results in

$$R_{23} = \frac{R_2 R_3}{R_2 + R_3}$$

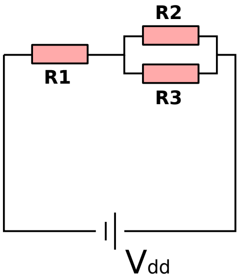
or

$$\frac{1}{R_{tot}} = \frac{1}{R_1} + \frac{1}{R_2}$$



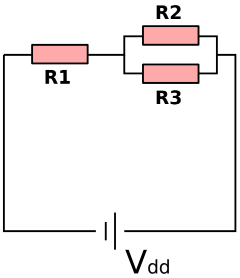
Voltage, Resistance and Current

- Using similar logic we can show that $R_{123} = R_1 + R_{23}$ and that $V_1 = \frac{R_1}{R_1+R_{23}}V$
- This is known as a voltage divider.



Voltage, Resistance and Current

- Using similar logic we can show that $R_{123} = R_1 + R_{23}$ and that $V_1 = \frac{R_1}{R_1+R_{23}}V$
- This is known as a voltage divider.



Capacitance

- Two other classical circuit elements are capacitors and coils.
- These are more important for AC circuits, as they form the basis of hardware filters, atenas, etc.
- Capacitance is the ratio between charge and voltage applied.
- Proportional to the plate area A and inversely to distance d, i.e.

$$C = \frac{\epsilon A}{d}$$

Capacitance

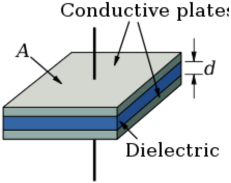
- Two other classical circuit elements are capacitors and coils.
- These are more important for AC circuits, as they form the basis of hardware filters, atenas, etc.
- Capacitance is the ratio between charge and voltage applied.
- Proportional to the plate area A and inversely to distance d, i.e.

$$C = \frac{\epsilon A}{d}$$

Capacitance

- Two other classical circuit elements are capacitors and coils.
- These are more important for AC circuits, as they form the basis of hardware filters, atenas, etc.
- Capacitance is the ratio between charge and voltage applied.
- Proportional to the plate area A and inversely to distance d, i.e.

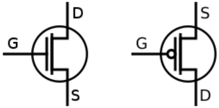
$$C = \frac{\epsilon A}{d}$$



source wikipedia

Transistors

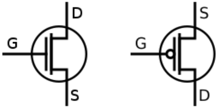
- A common elelment used to implement logic circuits.
- Consists of three terminals: source, drain, and gate.
- Current flows from source to drain, when the gate is positive (p-gate) or negative (n-gate)
- Provide switching capabilities and are used to implement logic gates.
- Different techniques to print in silicon: e.g., MOSFET

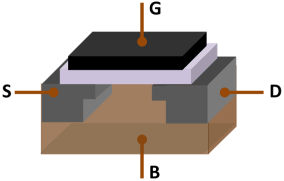


source wikipedia

Transistors

- A common elelment used to implement logic circuits.
- Consists of three terminals: source, drain, and gate.
- Current flows from source to drain, when the gate is positive (p-gate) or negative (n-gate)
- Provide switching capabilities and are used to implement logic gates.
- Different techniques to print in silicon: e.g., MOSFET

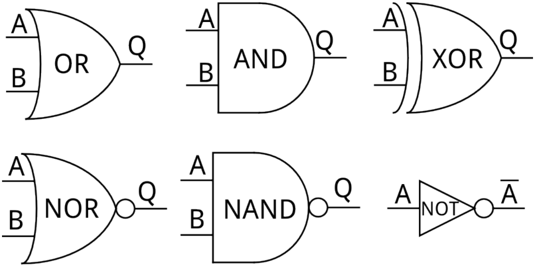




source wikipedia

Logic Gates

- Boolean logic implemented through a number of transistors.



source `learn.sparkfun.com`

Outline

1 Background

2 Microcontrollers

Microcontrollers

- Low-level sensor/actuator interfacing usually implemented on dedicated embedded devices.
- A microcontroller is essentially a single integrated circuit that provides a processor core, a memory array and input/output operations.
- Microcontrolers packaged on a printed circuit board with additional ICs and dedicated programming environments are often used used for prototyping.

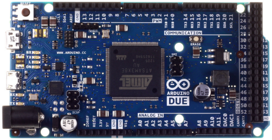


Figure: Arduino Due microcontroller board

Microcontrollers

- Low-level sensor/actuator interfacing usually implemented on dedicated embedded devices.
- A microcontroller is essentially a single integrated circuit that provides a processor core, a memory array and input/output operations.
- Microcontrolers packaged on a printed circuit board with additional ICs and dedicated programming environments are often used used for prototyping.

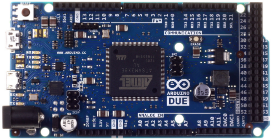


Figure: Arduino Due microcontroller board

Microcontrollers

- Low-level sensor/actuator interfacing usually implemented on dedicated embedded devices.
- A microcontroller is essentially a single integrated circuit that provides a processor core, a memory array and input/output operations.
- Microcontrolers packaged on a printed circuit board with additional ICs and dedicated programming environments are often used used for prototyping.

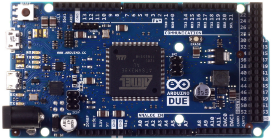


Figure: Arduino Due microcontroller board

The fetch-execute loop

- A microcontroller is a primitive stored instruction computer
- On startup it loads instructions from memory and executes a setup (booting) sequence.
- It then iterates through the instruction stack in an endless fetch-execute cycle.
- There is typically no operating system. You are responsible for handling all I/O and memory operations

The fetch-execute loop

- A microcontroller is a primitive stored instruction computer
- On startup it loads instructions from memory and executes a setup (booting) sequence.
- It then iterates through the instruction stack in an endless fetch-execute cycle.
- There is typically no operating system. You are responsible for handling all I/O and memory operations

The fetch-execute loop

- A microcontroller is a primitive stored instruction computer
- On startup it loads instructions from memory and executes a setup (booting) sequence.
- It then iterates through the instruction stack in an endless fetch-execute cycle.
- There is typically no operating system. You are responsible for handling all I/O and memory operations

The fetch-execute loop

- A microcontroller is a primitive stored instruction computer
- On startup it loads instructions from memory and executes a setup (booting) sequence.
- It then iterates through the instruction stack in an endless fetch-execute cycle.
- There is typically no operating system. You are responsible for handling all I/O and memory operations

The fetch-execute loop

```
void setup() {
  //put your setup code here, to run once
}
void loop() {
  //put your main code here, to run repeatedly
}
```

Digital I/O

- Microcontroller boards like the Arduino have a number of PINs that can be configured for input or output operations
- Pins for digital I/O can read/write logic values - HIGH or LOW
- HIGH values represent a fixed voltage potential relative to the digital ground
- LOW values have zero potential relative to ground
- When using digital I/O the logic ground for target devices has to be set to the ground terminal of the board.
- Example applications: reading in binary streams from sensors, detecting switches

Digital I/O

- Microcontroller boards like the Arduino have a number of PINs that can be configured for input or output operations
- Pins for digital I/O can read/write logic values - HIGH or LOW
- HIGH values represent a fixed voltage potential relative to the digital ground
- LOW values have zero potential relative to ground
- When using digital I/O the logic ground for target devices has to be set to the ground terminal of the board.
- Example applications: reading in binary streams from sensors, detecting switches

Digital I/O

- Microcontroller boards like the Arduino have a number of PINs that can be configured for input or output operations
- Pins for digital I/O can read/write logic values - HIGH or LOW
- HIGH values represent a fixed voltage potential relative to the digital ground
- LOW values have zero potential relative to ground
- When using digital I/O the logic ground for target devices has to be set to the ground terminal of the board.
- Example applications: reading in binary streams from sensors, detecting switches

Digital I/O

- Microcontroller boards like the Arduino have a number of PINs that can be configured for input or output operations
- Pins for digital I/O can read/write logic values - HIGH or LOW
- HIGH values represent a fixed voltage potential relative to the digital ground
- LOW values have zero potential relative to ground
- When using digital I/O the logic ground for target devices has to be set to the ground terminal of the board.
- Example applications: reading in binary streams from sensors, detecting switches

Digital I/O

- Microcontroller boards like the Arduino have a number of PINs that can be configured for input or output operations
- Pins for digital I/O can read/write logic values - HIGH or LOW
- HIGH values represent a fixed voltage potential relative to the digital ground
- LOW values have zero potential relative to ground
- When using digital I/O the logic ground for target devices has to be set to the ground terminal of the board.
- Example applications: reading in binary streams from sensors, detecting switches

The Arduino "Hello world"

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

Serial communication

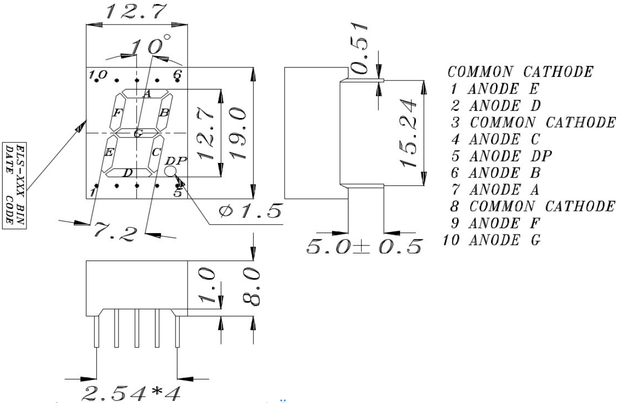
```
int a = 123;
void setup() {
  Serial.begin(19200);
}
void loop() {
  Serial.print(a, DEC);
  Serial.print(" ");
  Serial.print(a, HEX);
  Serial.println("");
  delay(1000);
}
```

Handling signals

```
int INT_PIN = 46;
void setup() {
  pinMode(INT_PIN, INPUT);
  attachInterrupt(INT_PIN, handler, CHANGE);
}
void handler() {
  //do something
}
```

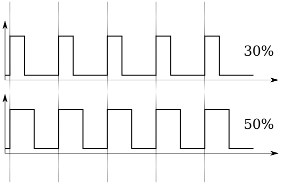
Case study: controlling an 8-segment display

- Let's try to do some simple arduino coding: display numbers on an 8-segment display.
- We will increment the number every time a switch is flipped on.



Analogue I/O

- Analogue input pins can be configured to measure voltage potentials relative to ground level.
- Analogue output pins produce a pulse-width modulated (PWM) square wave signal.
- Depending on the board, the PWM duty cycle and/or the frequency can be modified.
- Example applications: measuring values from a current sensor, driving a motor.



Programming Tips

- Programming for embedded devices can be tricky.
- Code needs to be cross-compiled for the controller and then uploaded to the on-board flash memory.
- Debugging can be difficult as outputs come on the serial port and programs cannot be paused.
- A useful tool for real-time processing are interrupts, which can trigger code pieces when the states of I/O pins change.

Sensors and Sensing
Microcontrollers, I/O Programming and Signals

Todor Stoyanov

Centre for Applied Autonomous Sensor Systems
Örebro University
Sweden