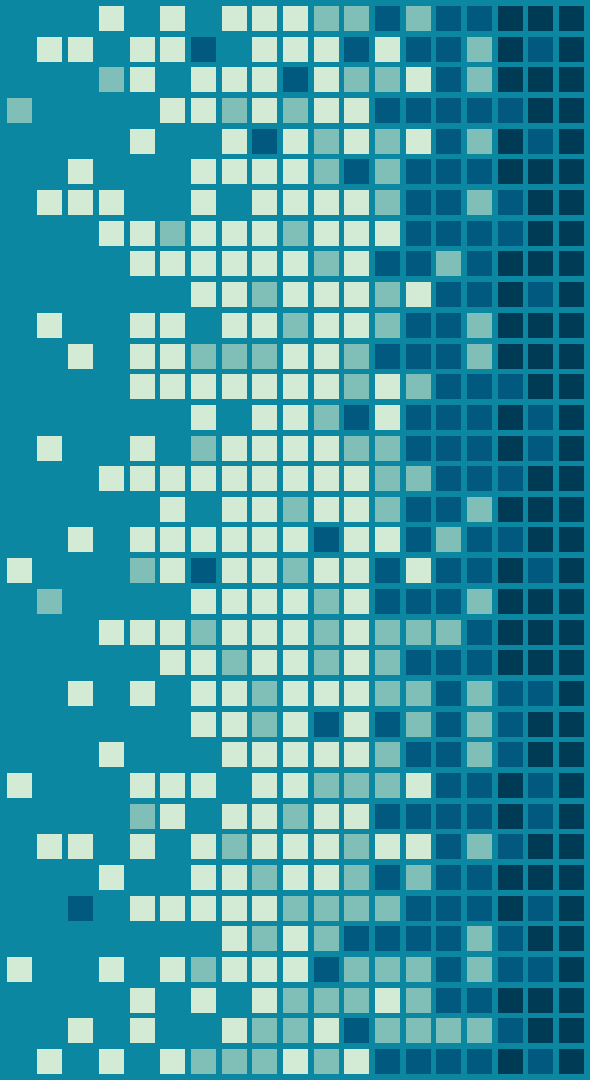
Intro to Microcontrollers

Michael D'Argenio – mjdargen@ncsu.edu
Electrical Engineering – SS 2019 – Duke TIP

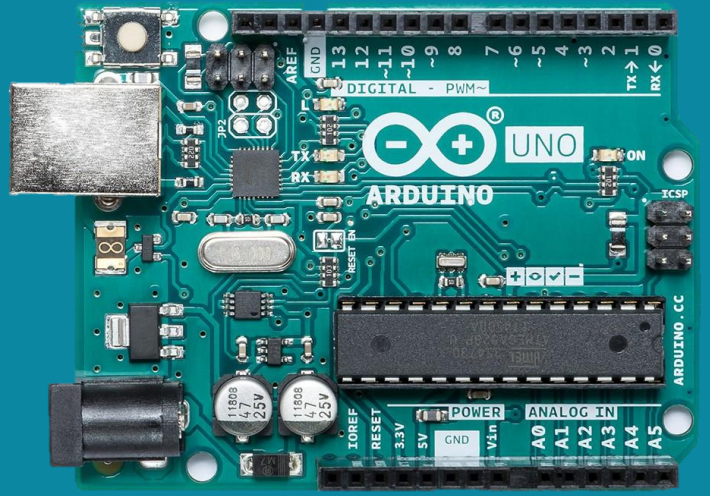


What is a Microcontroller (MCU)?

- It is a small computer on a single integrated circuit.
- What makes it different than a microprocessor?
  - Microprocessors are just processors. Microcontrollers contain a processor as well as other circuitry.
  - Microcontrollers can function on their own, whereas microprocessors cannot.
- MCUs contain the following:
  - Central Processing Unit (CPU)
  - Memory (either ROM, RAM or both)
  - I/O Peripherals

When do we use microcontrollers?

- They are typically used in embedded system applications.
- Embedded system - A special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints.
  - Examples: thermostat, vehicle control systems, cameras, dishwashers, coffee machines, robots, traffic lights, printers, MRIs, microwaves, home security systems.
    - Most any modern electronic has an embedded system
  - Non-examples: personal computers or smart phones (these are more general purpose computers)
- Not all embedded systems use MCUs, but most do.


DIGITAL - PWM~
UNO
ARDUINO
ON
ICSP
POWER
ANALOG IN
GND
ARDUINO.CC

Arduino Uno

ATmega328 MCU

- Arduino Uno board is built with the ATmega328 MCU.
- The ATmega328 is a low-power, CMOS, 8-bit microcontroller based on the RISC architecture.
- The data sheet is 662 pages long!!!

Bootloader

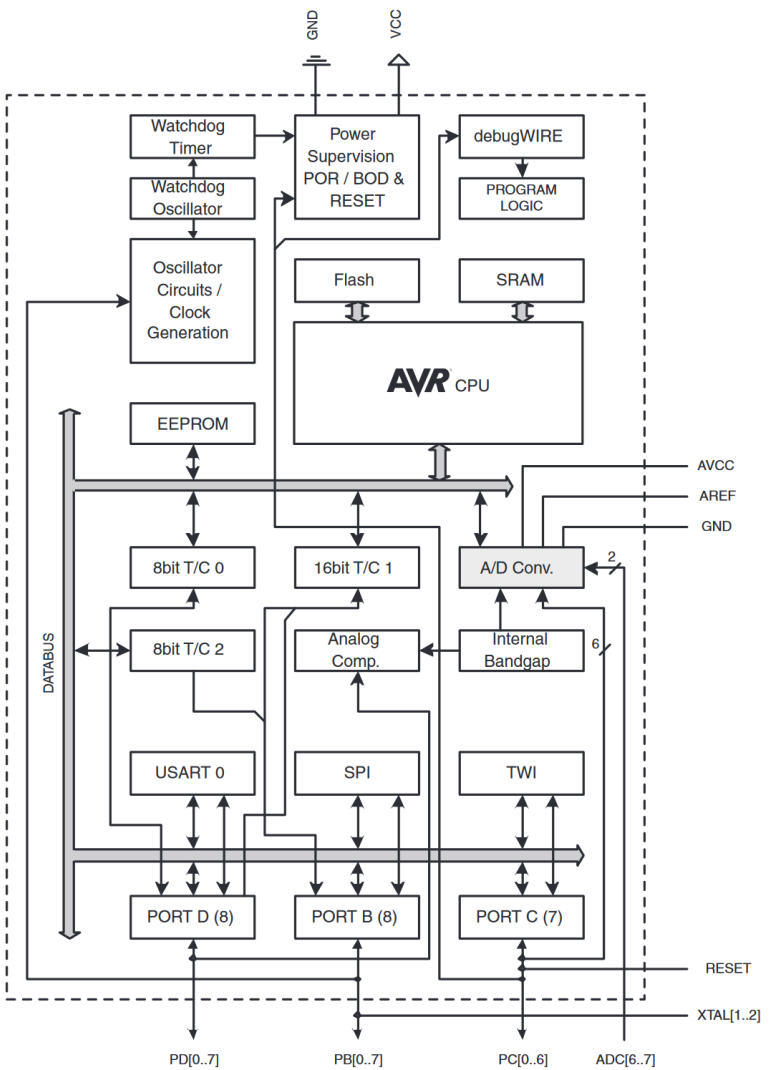
- Microcontrollers are usually programmed through a programmer unless you have a bootloader.
- The ATmega328 on the Arduino Uno comes preprogrammed with a bootloader that allows you to upload new code to it without the use of an external hardware programmer.
- A bootloader is a piece of firmware (i.e. software that can't be changed) in your microcontroller that allows you to easily upload your new program or "sketch".

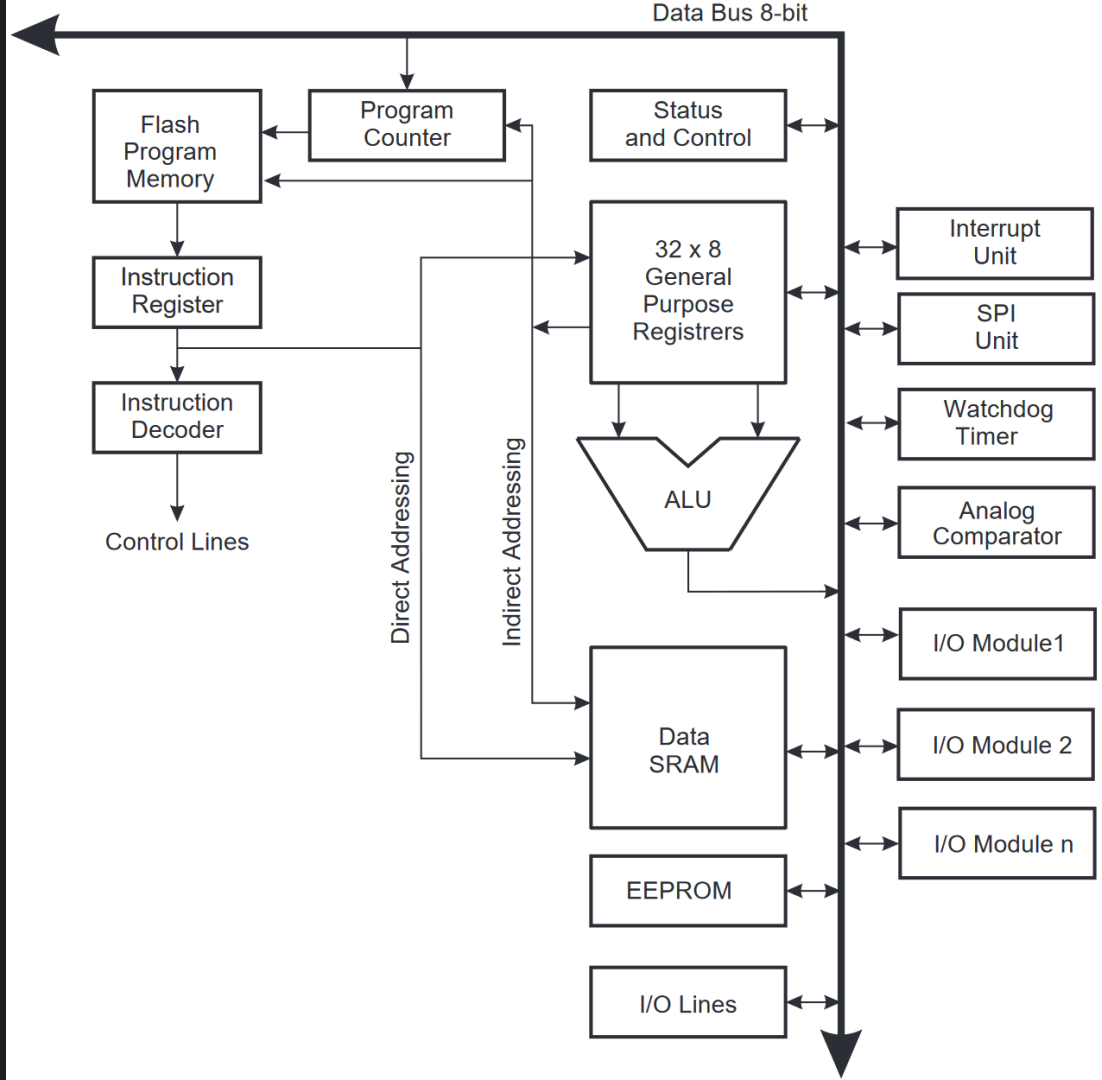
Another MCU??

- Arduino Uno has an ATmega16U2 MCU as well. Why?
- The ATmega16U2 MCU acts as a USB to Serial Port Interface so that your computer can communicate directly with the ATmega328.
- This is typical for most all evaluation or development kits for MCUs.
- Allows for easier communication and flashing of new programs.

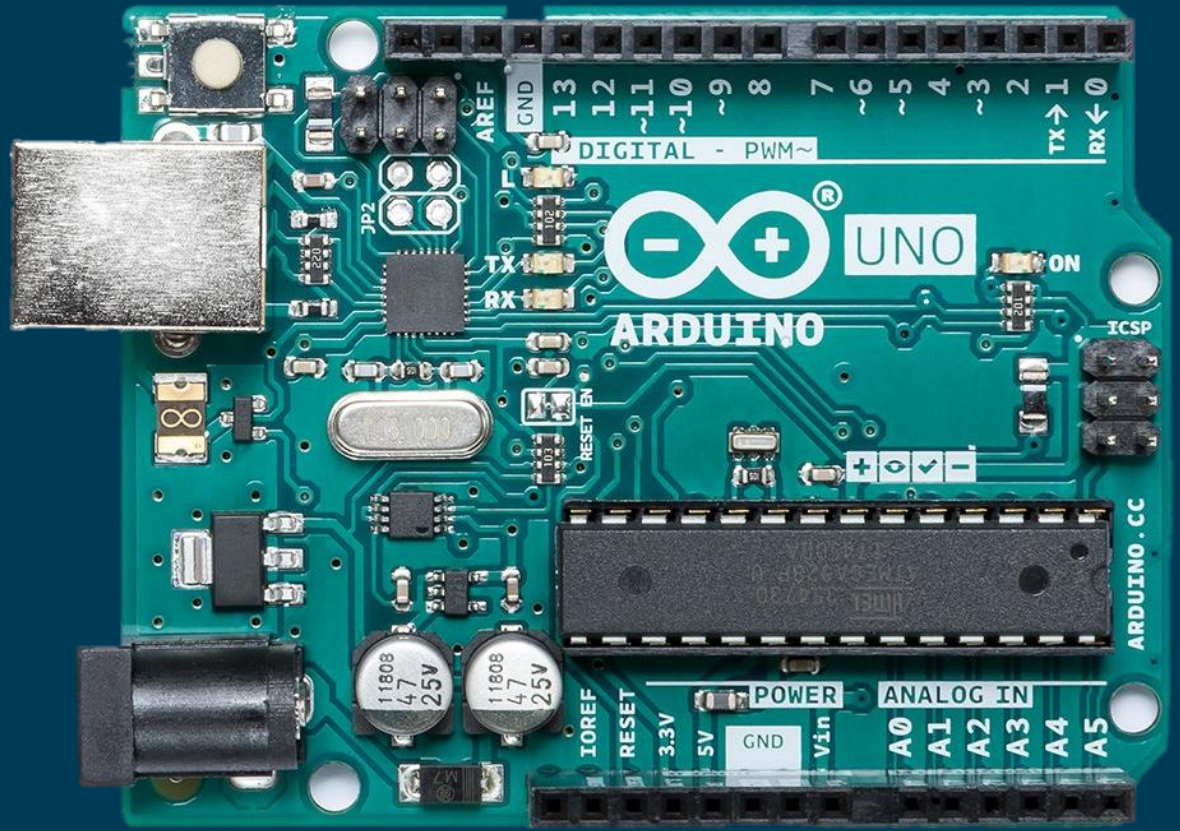
ATmega328

- Open datasheet and view main features page
- http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf


GND
VCC
Watchdog Timer
Watchdog Oscillator
Power Supervision POR / BOD & RESET
debugWIRE
PROGRAM LOGIC
Oscillator Circuits / Clock Generation
Flash
SRAM
AVR CPU
EEPROM
AVCC
AREF
GND
8bit T/C 0
16bit T/C 1
A/D Conv.
2
DATABUS
8bit T/C 2
Analog Comp.
Internal Bandgap
6
USART 0
SPI
TWI
PORT D (8)
PORT B (8)
PORT C (7)
RESET
XTAL[1..2]
PD[0..7]
PB[0..7]
PC[0..6]
ADC[6..7]

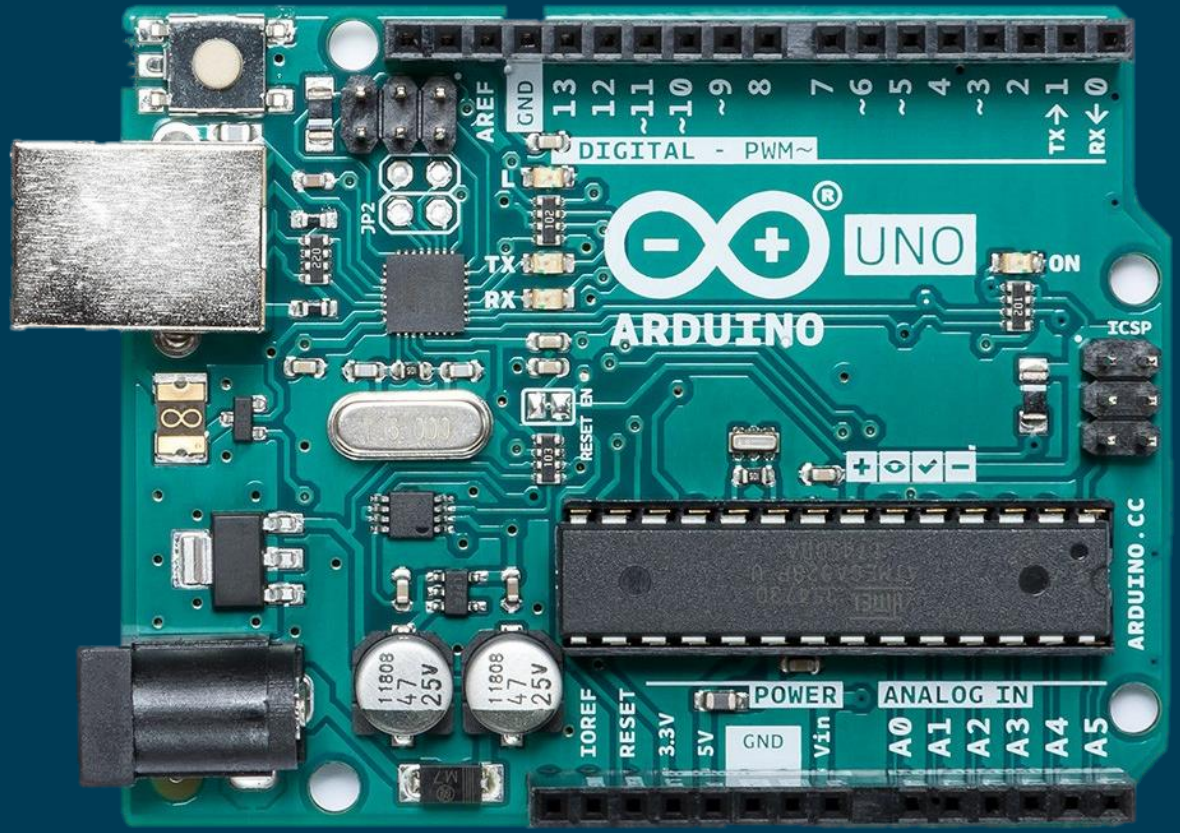

Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
Instruction Decoder
Control Lines
32 x 8 General Purpose Registrers
ALU
Direct Addressing
Indirect Addressing
Data SRAM
EEPROM
I/O Lines
Interrupt Unit
SPI Unit
Watchdog Timer
Analog Comparator
I/O Module1
I/O Module 2
I/O Module n

Arduino Uno


AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL - PWM~
JP2
TX
RX
UNO
ON
ARDUINO
ICSP
RESET EN
ARDUINO.CC
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
Vin
A0
A1
A2
A3
A4
A5

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Power Pins


AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL - PWM~
JP2
TX
RX
UNO
ON
ARDUINO
ICSP
RESET EN
ARDUINO.CC
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
Vin
A0
A1
A2
A3
A4
A5

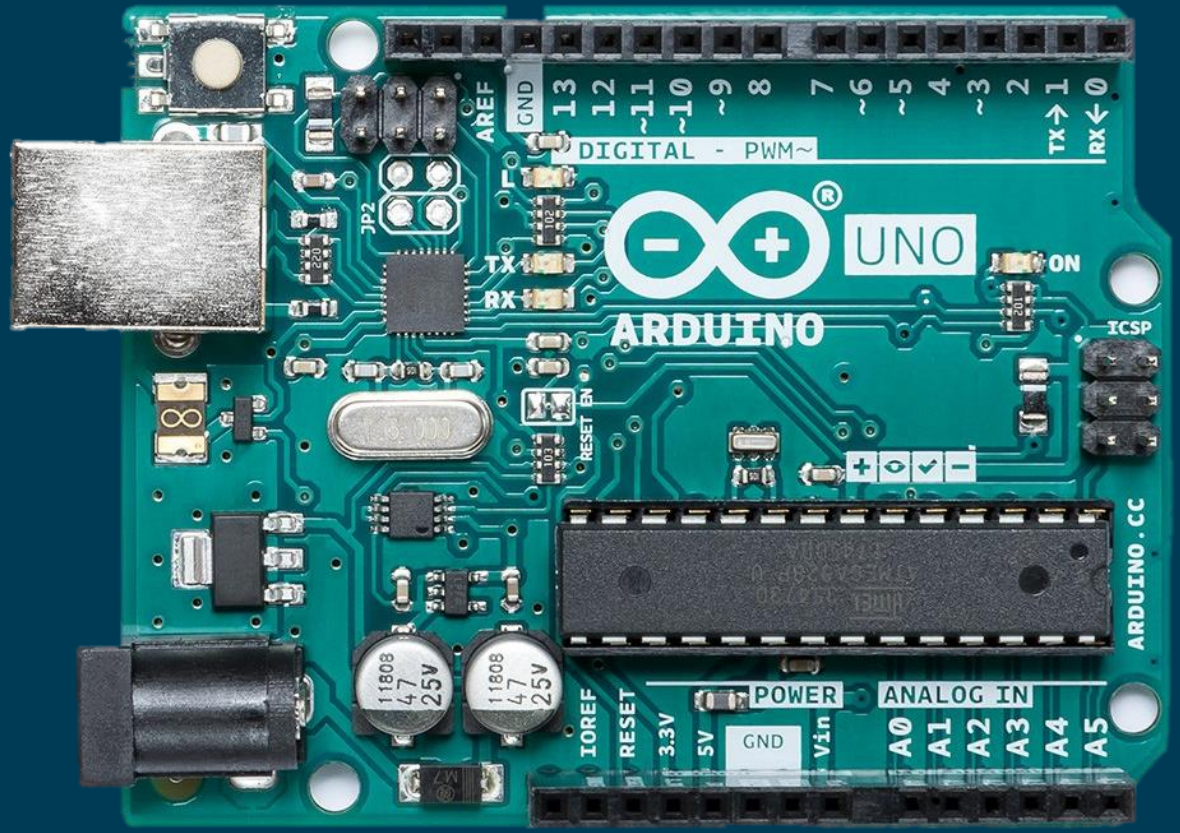
Power

- The Arduino Uno board can be powered in two different ways (never both at the same time).
  - Via the USB connection
    - Powered from your computer
    - Also used for uploading sketch or comms
  - Via the barrel jack with an external 9V supply
    - 9V battery
    - AC-to-DC Adapter (wall-wart)
- The power source is selected automatically.

Power Pins

- **Vin**: The input voltage to the Arduino/Genuino board when it's using an external power source. You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V**: This pin outputs a regulated 5V from the regulator on the board. DO NOT SUPPLY POWER HERE.
- **3V3**: A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA. DO NOT SUPPLY POWER HERE.
- **GND:** Ground pins.
- **IOREF**: This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.
- **Reset**: Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Digital Pins



Digital Pins

- 14 digital pins/ports
- They all operate at 5 volts.
- All digital pins can be configured as input or output.
- SoftwareSerial library allows serial communication on any of the Uno's digital pins.
- Some have special functionality. (shown next page)

Digital Pins: Special Functionality

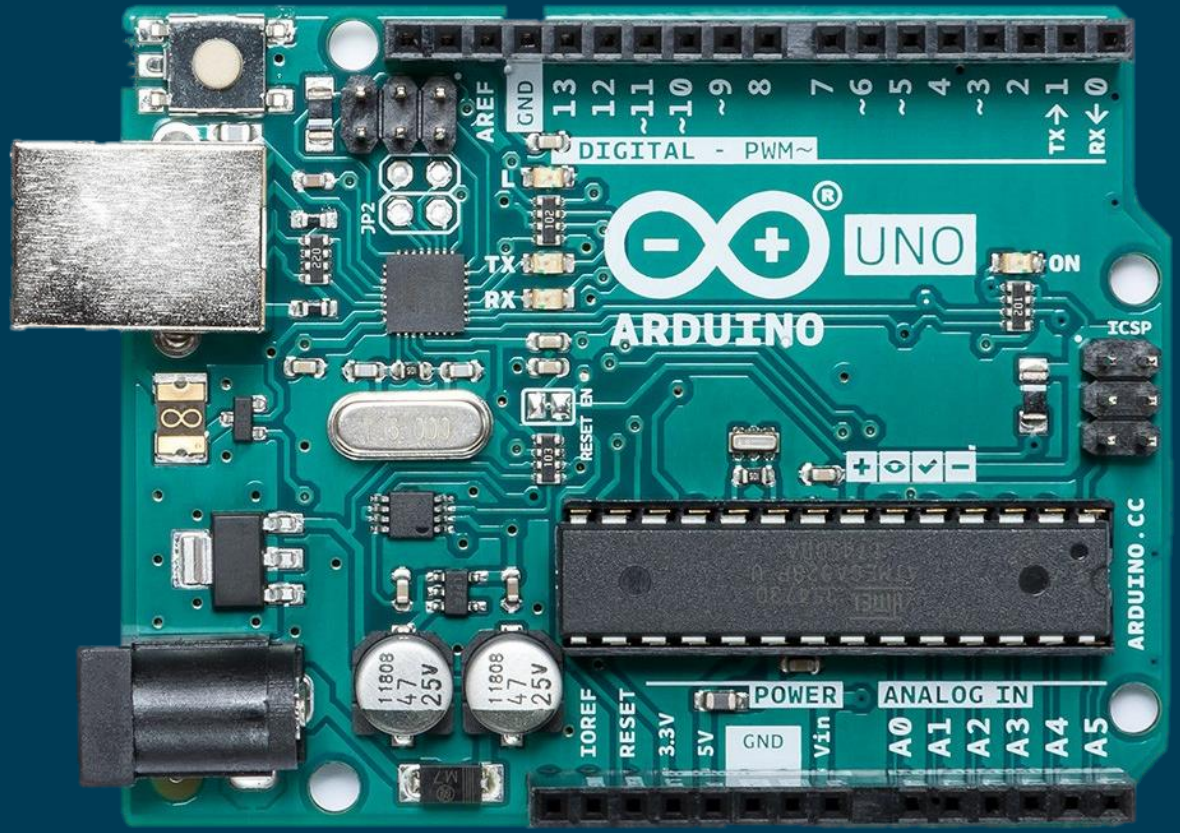
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- LED: 13. There is a built-in LED driven by digital pin 13. When the pin is HIGH, the LED is on, when the pin is LOW, it's off.

Digital Pins: Serial Communications

- UART: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- I^2C or TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

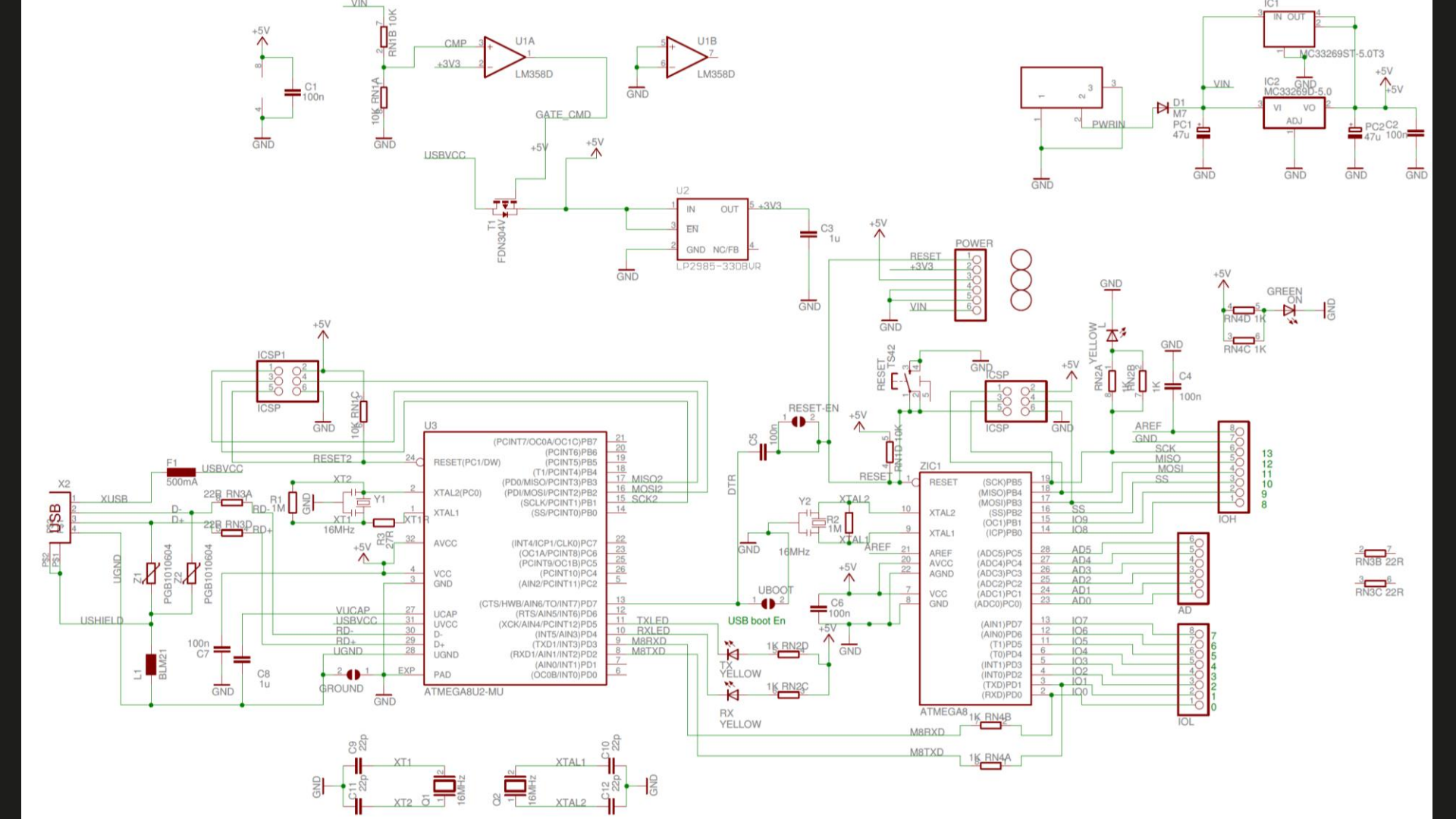
- SoftwareSerial library allows serial communication on any of the Uno's digital pins; however, it is easier to use the pins listed above for SPI and I^2C.

Analog Pins

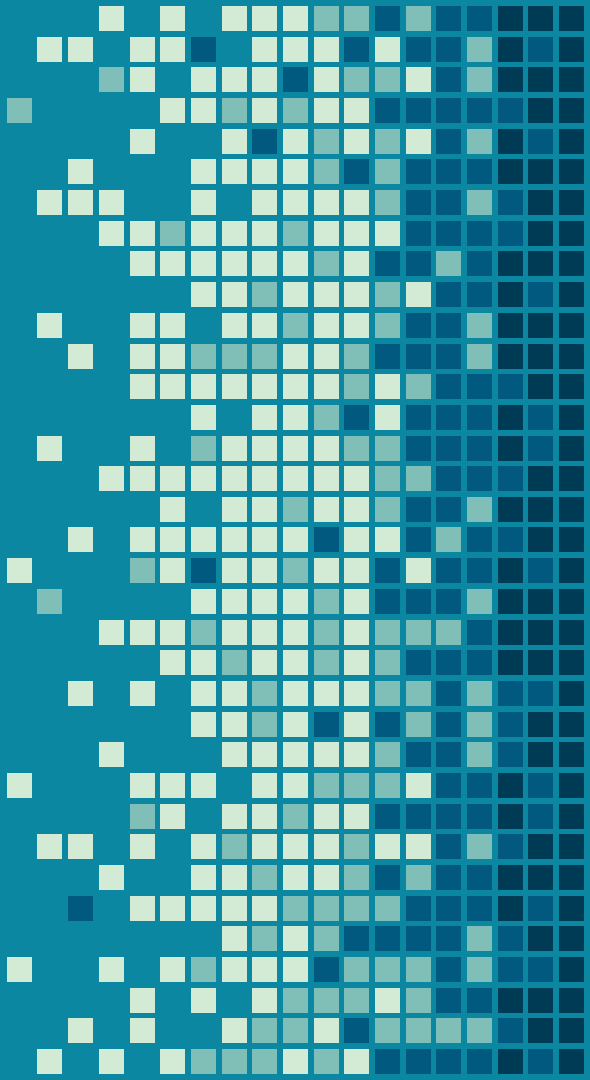

AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL - PWM~
JP2
TX
RX
UNO
ON
ARDUINO
ICSP
RESET EN
ARDUINO.CC
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
Vin
A0
A1
A2
A3
A4
A5

Analog Pins

- 6 analog inputs, labeled A0 through A5.
- Used to read analog voltages.
- Each provide 10 bits of resolution (i.e. 1024 different values).
- By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function.
- AREF. Reference voltage for the analog inputs. Used with analogReference().


+5V
C1
100n
GND
VIN
RN1B 10K
RN1A 10K
CMP
+3V3
U1A
LM358D
U1B
LM358D
GATE_CMD
USBVCC
T1
FDN304V
U2
IN
OUT
EN
GND
NC/FB
LP2985-33DBVR
+3V3
C3
1u
IC1
MC33269ST-5.0T3
IC2
MC33269D-5.0
VI
VO
ADJ
PWRIN
D1
M7
PC1
47u
PC2
47u
C2
100n
POWER
RESET
ICSP1
ICSP
RESET2
U3
ATMEGA8U2-MU
RESET(PC1/DW)
XTAL2(PC0)
XTAL1
AVCC
VCC
GND
UCAP
UVCC
D-
D+
UGND
PAD
MISO2
MOSI2
SCK2
F1
500mA
USBVCC
X2
USB
XUSB
RN3A 22R
RN3D 22R
R1
1M
XT2
XT1
16MHz
Y1
XT1R
R3
27R
Z1
Z2
PGB1010604
L1
BLM21
C7
100n
C8
1u
VUCAP
USHIELD
EXP
GROUND
RESET-EN
C5
100n
DTR
UBOOT
USB boot En
TXLED
RXLED
M8RXD
M8TXD
TX
YELLOW
RX
YELLOW
1K RN2D
1K RN2C
ZIC1
ATMEGA8
XTAL2
XTAL1
AREF
AVCC
AGND
VCC
GND
(SCK)PB5
(MISO)PB4
(MOSI)PB3
(SS)PB2
(OC1)PB1
(ICP)PB0
(ADC5)PC5
(ADC4)PC4
(ADC3)PC3
(ADC2)PC2
(ADC1)PC1
(ADC0)PC0
(AIN1)PD7
(AIN0)PD6
(T1)PD5
(T0)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
1K RN4B
1K RN4A
RESET
+3V3
VIN
TS42
ICSP
RN1D 10K
YELLOW
RN2A
RN2B
1K
C4
100n
GREEN
ON
RN4D 1K
RN4C 1K
AREF
SCK
MISO
MOSI
SS
IOH
SS
IO9
IO8
AD
AD5
AD4
AD3
AD2
AD1
AD0
IOL
IO7
IO6
IO5
IO4
IO3
IO2
IO1
IO0
RN3B 22R
RN3C 22R
C9
22p
C11
22p
XT1
XT2
Q1
16MHz
Q2
16MHz
C10
22p
C12
22p
XTAL1
XTAL2

INTERRUPTS



Problem

- Scenario: Our program displays a menu that automatically scrolls through a list of different actions that our Arduino can perform. When the user pushes the button, we want to execute the action that the screen was currently showing (example: retrieve the current temperature).
- Problem: Programs execute line by line and can only execute one line at a time.
  - Our processor is busy updating the screen. Can't use a busy-wait command to see if button is pressed.
  - If we occasionally check to see if button is pressed, we may miss the button press.

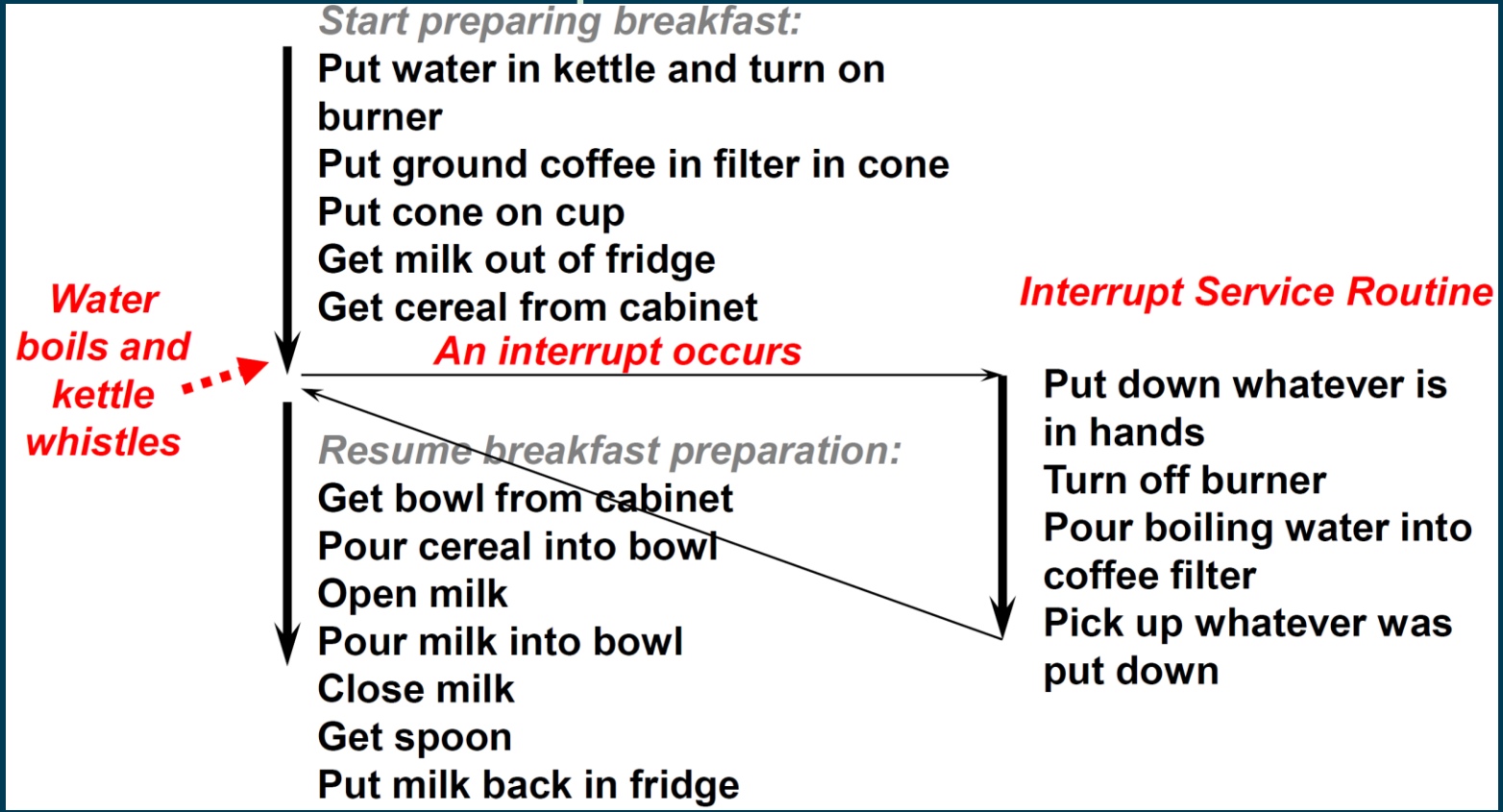
Solution: Interrupts

- Interrupts signal to the processor that some event has occurred.
- It “interrupts” the main code sequence and executes an Interrupt Service Routine (ISR).
- An ISR is a user-defined subroutine/function that must execute after its interrupt was triggered.
- After the ISR executes, the processor returns back to the main code where it was interrupted.

Coffee Example

- Consider the task of making coffee.
  - Need to boil water, but don't know exactly how long it will take to boil.
- How do we detect the water is boiling?
  - Keep watching the pot until we see bubbles.
    - This is called *polling.*
    - Wastes time – can't do anything else while we wait.
  - Put the water in a kettle which will whistle when it boils.
    - The whistle is an *interrupt.*
    - Don't need to keep watching water. Instead you can do something else until the kettle whistles.
    - Much more efficient.

Coffee Example



Interrupt Sequencing

- Mainline code is running in foreground.
- Interrupt trigger occurs.
- Processor does context switching.
  - Moves mainline code to background.
  - Moves ISR to foreground.
- Processor executes ISR to completion.
- Processor resumes mainline code.
  - ISR moves to background until next interrupt.
  - Mainline code moves to foreground.

Interrupt Guidelines

- Arduinos only allow hardware interrupts on digital pins 2 and 3.
- ISRs should not take too long to execute.
- “Time” subroutines (delay(), millis(), etc.) do not work in ISRs.
- If using a global variable in the ISR and in the mainline code, declare it as volatile.
- Need to initialize interrupts and declare ISRs.
- Can enable and disable all interrupts.

Volatile

- If using a global variable in the ISR and in the mainline code, declare it as volatile.
- This tells the compiler to not optimize this variable.
- Compiler should always retrieve most recent value from memory and not assume the value.
- https://www.arduino.cc/reference/en/language/variables/variable-scope--qualifiers/volatile/

Set-Up Interrupt

- attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
  - pin: the pin number (either 2 or 3)
  - ISR: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing.
  - mode:
    - LOW: triggers when the pin is low.
    - CHANGE: triggers when the pin changes value.
    - RISING: triggers when the pin goes from low to high.
    - FALLING: triggers when the pin goes from low to high.
- https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/

Interrupt Example: LED State

```
const byte ledPin = 13;
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

Disable Specific Interrupt

- detachInterrupt(digitalPinToInterrupt(pin))
  - pin: the pin number of the interrupt to disable
- Used to disable a specific interrupt.
- Allows you to reconfigure that digital pin for another interrupt.

Enable/Disable All Interrupts

- interrupts() – re-enables all interrupts (only need to be called if they've been disabled by noInterrupts()).
- noInterrupts() – disables all interrupts.
- Useful if there is a critical section of mainline code that you don't want to be interrupted.

```
void setup() {}

void loop() {
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

Polling vs. Interrupts

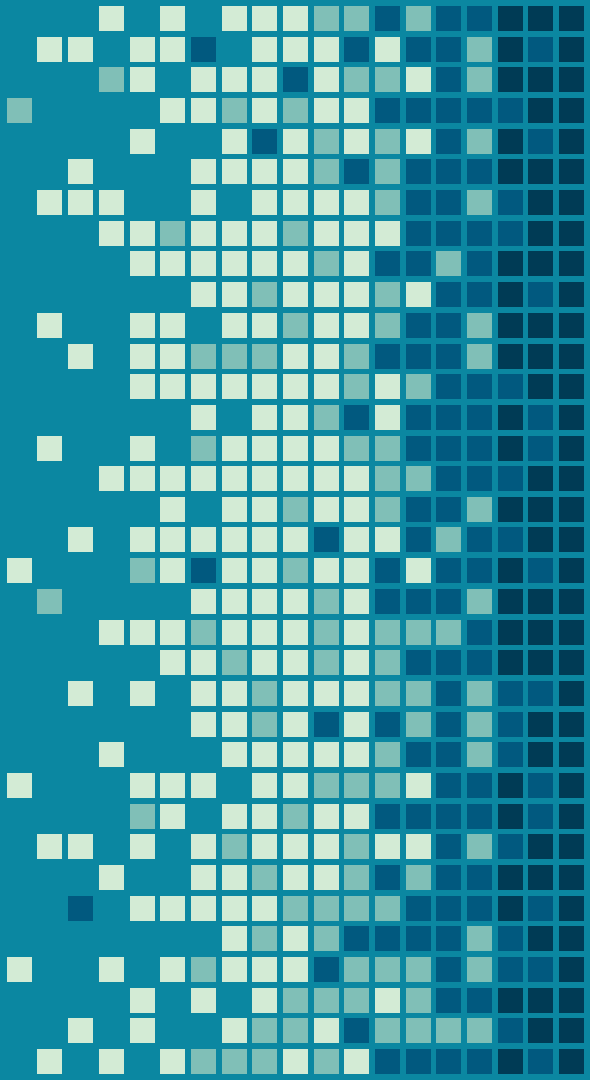
- Polling is fine when:
  - When you don't have to wait that long.
  - When there's nothing else for the processor to do.
  - The event must be very synchronous with mainline code.
- Interrupts are better when:
  - You need a fast response.
  - The event that is supposed to trigger a response only occurs for a short period of time.
  - You have to record incoming data.

TIMER INTERRUPTS

Timer Interrupts

- There is the ability to configure timer interrupts.
- Instead of triggering an interrupt on a digital input, you can trigger an interrupt periodically (exp. every 1 second).
- Helpful if you want to periodically check sensor or other peripheral while running other mainline code.
- https://learn.adafruit.com/multi-tasking-the-arduino-part-2/timers
- https://www.teachmemicro.com/arduino-timer-interrupt-tutorial/

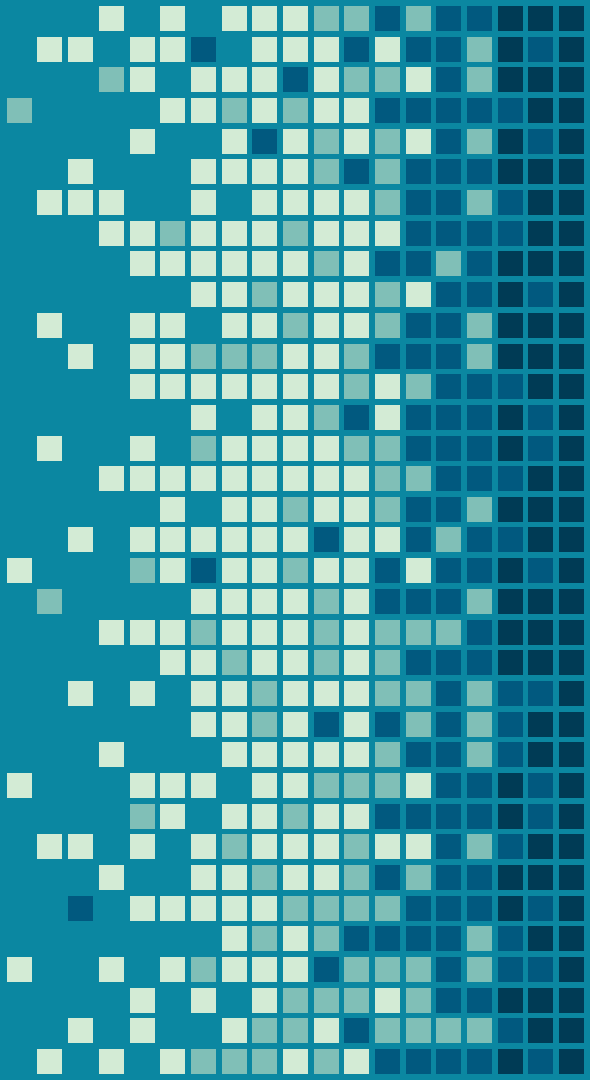
STRINGS



Strings

- Strings – in C, it's an array of chars
- However, Arduino programming language supports a string data type.
- Allows for easier data manipulation.
- A number of dedicated library functions for strings are detailed here: https://www.arduino.cc/en/Tutorial/BuiltInExamples#strings

MISC



Delay (ms)

- delay(ms)
- Library function that pauses the program for the amount of time (in milliseconds) specified in the argument.
- https://www.arduino.cc/reference/en/language/functions/time/delay/

Delay (us)

- delayMicroseconds(us)
- Library function that pauses the program for the amount of time (in microseconds) specified in the argument.
- https://www.arduino.cc/reference/en/language/functions/time/delay/

millis( )

- Returns the number of milliseconds passed since the Arduino board began running the current program.
- Data type unsigned long
- https://www.arduino.cc/reference/en/language/functions/time/millis/

micros( )

- Returns the number of microseconds passed since the Arduino board began running the current program.
- Data type unsigned long
- https://www.arduino.cc/reference/en/language/functions/time/micros/

ARDUINO PROGRAMMING LANGUAGE

Arduino Native Functions

- Information about all of the functions natively supported by Arduino.
- https://www.arduino.cc/reference/en/#functions

Arduino Variables and Data Types

- Information about all of the variables and data types supported by Arduino.
- https://www.arduino.cc/reference/en/#variables

Arduino Structure and Operations

- Information about all of the operations and control flow structures supported by Arduino.
- https://www.arduino.cc/reference/en/#structure

Arduino Libraries

- Information about the many importable libraries available for the Arduino.
- https://www.arduino.cc/en/Reference/Libraries

Arduino Glossary

- Glossary for all terms surrounding the Arduino.
- https://www.arduino.cc/glossary/en/