Get the Examples:
http://github.com/devunwired/intro-arduino

Talk to your Toaster:
Exploring Embedded with Arduino

Dave Smith
@devunwired

About the Author

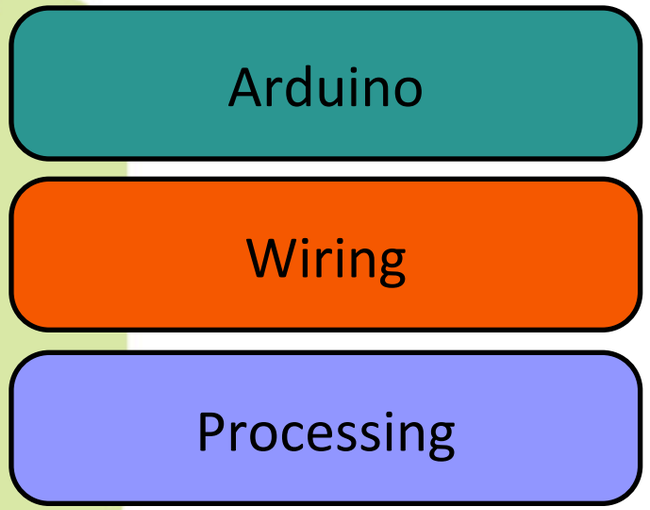
- Android developer since 2009
  - ROM customization for Embedded applications
- Recovering Spark Chaser
  - Embedded M2M Monitoring systems
  - P2P Radio Links
- Co-Author of Android Recipes from Apress

What is Arduino?

- Open source electronic microprocessor development platform
  - Hardware schematics
  - Code Libraries
  - IDE/Development tools
- Wrapper around traditional MCU development tools
- NOT an interpreted environment (e.g. Parallax STAMP)
  - Compiled into native assembly for target
- Simplifies embedded application code
- Makes prototyping easy

Programming Language

- Processing Framework
  - Development environment
  - Cross-platform language for visual modeling
  - "Software Sketchbook"
  - Runtimes include Java, Javascript, Android
- Wiring Framework
  - Adaptation of Processing language for MCU I/O programming
  - Rapid prototyping
  - "Sketching with hardware"
- Arduino
  - Variation on Wiring Implementation
- Similar to C99
  - No function prototypes
  - No `main()` entry point
- Execution localized to two main functions
  - `setup()` for one-time initial setup
  - `loop()` for repeated program execution



Traditional Microcontroller Code

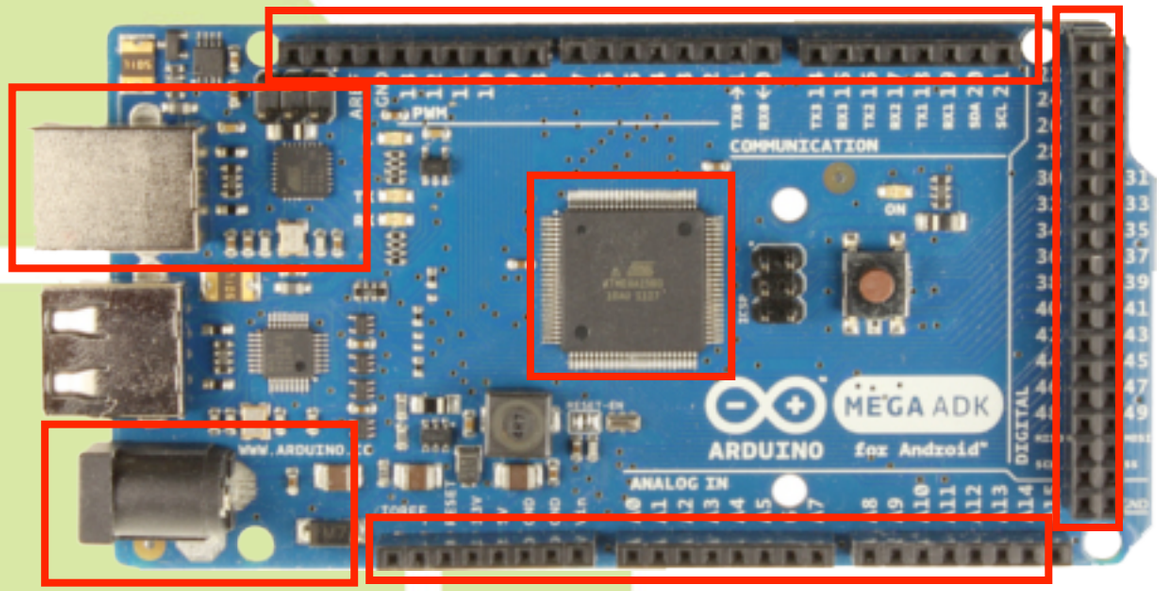
```
// set pin 4 output HIGH
DDRG |= (1 << 5);
PORTG |= (1 << 5);
// set pin 5 input w/ pullup
DDRE &= ~(1 << 3);
PORTE |= (1 << 3);

// set the baud rate prescaler for 9600
// (16MHz/(16*9600))-1
UBRR0H = 0x00;
UBRR0L = 0x67;

// enable RX and TX and set interrupts on rx complete
UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0);

// 8-bit, 1 stop bit, no parity, asynchronous UART
UCSR0C = (1 << UCSZ01) | (1 << UCSZ00) | (0 << USBS0) |
    (0 << UPM01) | (0 << UPM00) | (0 << UMSEL01) |
    (0 << UMSEL00);

// enable 50% duty PWM
DDRE |= (1 << 5);
TCCR3A = (1 << COM3C1) | (1 << WGM30);
TCCR3B = (1 << CS30) | (1 << CS31);
OCR3C = 0x7F;
```

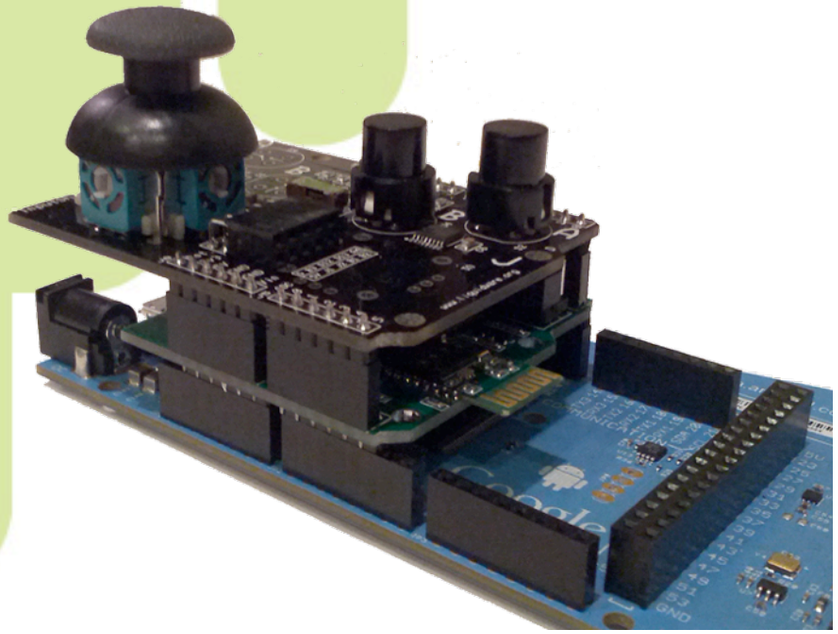
Equivalent Arduino Code

```
// set pin 4 output HIGH
pinMode(4, OUTPUT);
digitalWrite(4, HIGH);
// set pin 5 input w/ pullup
pinMode(5, INPUT_PULLUP);

// set the baud rate prescaler for 9600
// 8-bit, 1 stop bit, no parity, asynchronous UART
Serial.begin(9600, SERIAL_8N1);

// enable 50% duty PWM, pin 3
analogWrite(3, 127);
```

Hardware Components

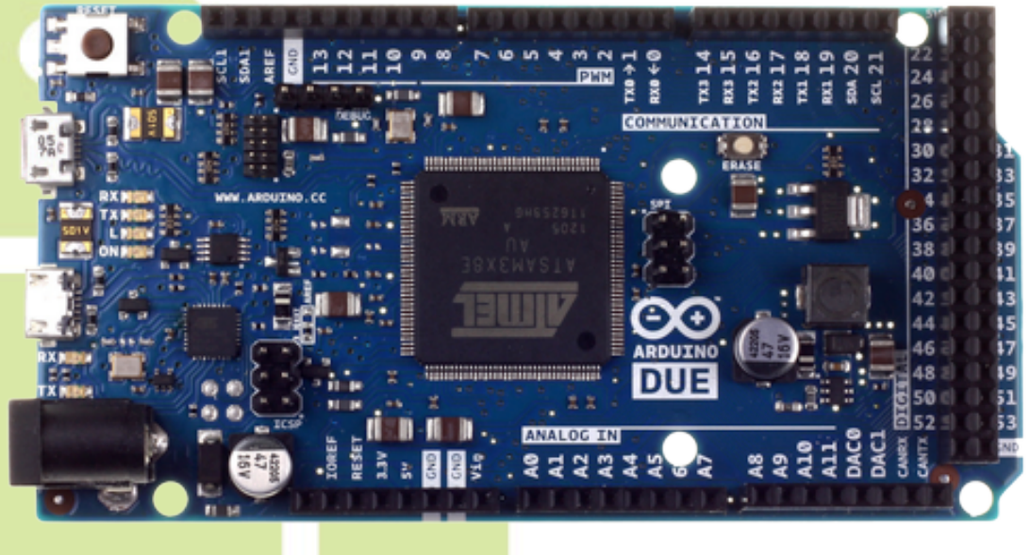
- Base Board
  - Power supply
  - MCU
  - Programming port
  - Standard I/O pinout
    - Digital
    - Analog
    - PWM
    - Communication (Serial)
- Shields
  - Additional functionality
  - Stackable design
  - Bluetooth, WiFi, Ethernet, CAN bus, GSM, Audio, Camera, GPS, Motors, Sensors, LCD, Proto...and more!

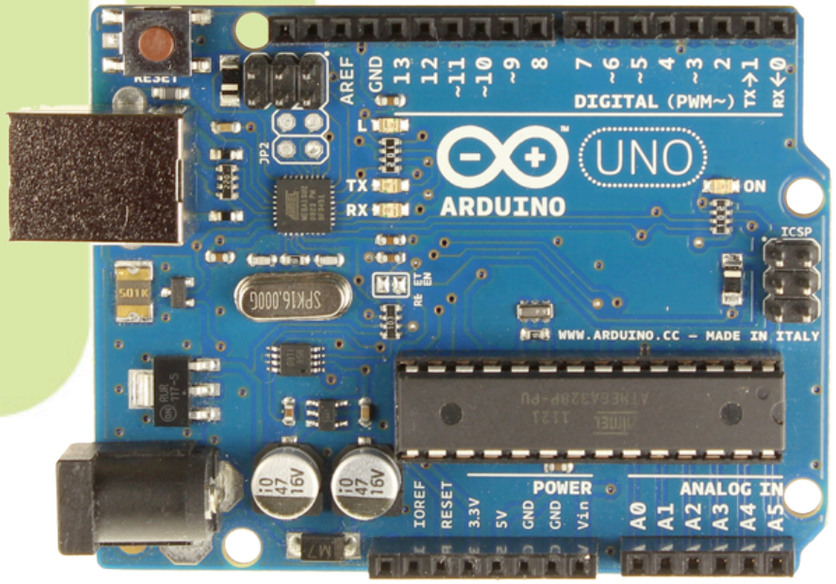


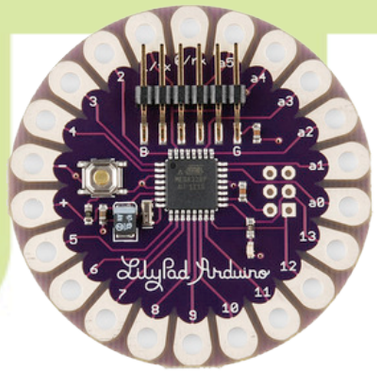


Hardware Options

- 15 Official Configurations
- Uno
  - Atmel AVR ATmega328 (8-bit)
  - 5V
  - 32KB Program Flash (-512B for Bootloader)
  - 14 GPIO
- Mega (ADK)
  - Atmel AVR ATmega2560 (8-bit)
  - 5V
  - 256KB Program Flash (-8KB for Bootloader)
  - 54 GPIO
- Due
  - AT91SAM3X8E ARM Cortex-M3 (32-bit)
  - 3.3V
  - 512KB Program Flash
  - 54 GPIO







Development Tools

- Arduino IDE
  - Mac OS X, Windows, Linux
  - Written in Java
  - Open Source
    - http://github.com/arduino/Arduino
- IDE Workspace called the "Sketchbook"
- Application code written in a "Sketch"
  - Stripped-down C program
  - Multiple files allowed (code organization)
    - Concatenated together at build time into one "main sketch"
    - No extension, .c, .cpp, .h file extensions allowed
  - Contained in a single .ino file (.pde on older versions)
- Unofficial plugins for Eclipse, NetBeans, Xcode
  - http://playground.arduino.cc/Main/DevelopmentTools

Build Process

- Sketch converted into proper main file
  - `#include "Arduino.h"` added at top
  - Function prototypes created and inserted after includes
  - Contents of core main.cpp file appended to end
- Compile libraries referenced by sketch separately into .o
- Compile/Link into Intel .hex file
  - AVR variants use avr-gcc
  - ARM variants use arm-eabi-gcc
- Uploaded to hardware
  - AVR variants use avrdude (AVR Downloader/UploaDEr)
  - ARM variants use BOSSA (Basic Open Source SAM-BA Application)

Bootloader

- Hardware pre-loaded with a serial bootloader
  - Assists in flashing code over USB/Serial interface (YAY!)
    - No additional programming hardware needed (JTAG, ICSP)
  - Requires program flash space (BOO!)
  - Delays program startup by a few seconds
- Supports external programmers
  - Regain full use of entire Program Flash
  - Update `upload.using` in `preferences.txt` to reference a supported programmer
  - Can supply any supported programmer from `hardware/programmers.txt`
  - Board still flashed using Arduino IDE
- Arduino IDE supports flashing bootloader into new chips

A Basic Sketch

```
void setup() {
  //Code run once at startup
}

void loop() {
  //Code executed repeatedly as long as device is running
}


//Add other custom functions, if you choose
void doSomething(int param) {

}

int readValue() {

}
```

Data Types

- boolean, char, unsigned char, byte (8-bit)
- int, unsigned int, word (16/32-bit)
- long, unsigned long (32/64-bit)
- short (16-bit)
- float, double (32/64-bit)
- array
- string
  - Standard C-string (null term. char array)
  - Functions from `<string.h>`
    - `strcat()`, `strcpy()`, `strlen()`
- StringObject
  - Higher level manipulation methods
    - `equals()`, `indexOf()`, `replace()`
  - Overloaded operators
    - Compare with ==, concatenate with +

```
#include <string.h>
//C-string examples
char str[] = "Text String";
char *name = "User";
//Get length
strlen(str);
//Add some chars
strcat(name, "123");
//Test string equality
if (strcmp(str, name) == 0) {};

//String object examples
String message = "Hit Enter";
String entry = String(255, BIN);
//Get length
message.length();
//Add some chars
entry += "123";
//Test string equality
if (message == entry) {};
```

Memory

- Types
  - Flash (program space, read-only)
  - SRAM (variables, data)
  - EEPROM (non-volatile, persisted)
- Default location for all variables is SRAM
  - Can fill up quickly with strings/tables
- Load large constant data chunks into Flash
  - Special functions to insert/read data in Flash
  - PROGMEM keyword
- Memory errors easier to debug
  - Full SRAM uploads but causes runtime errors
  - Full Flash won't upload or run

Variables

- Global
  - Visible to all functions
- Local
  - Visible only inside function
  - Value lost when leave scope
- `static`
  - Same visibility as local
  - Value persisted between calls
- Read-only
  - Keyword (`const`)
    - Better scope
  - Compiler directive (`#define`)
    - No additional memory
- `volatile`
  - Load from RAM vs. Register
  - Use for concurrent modification (Interrupts)

```
#define redLed 5
const float gravity = 9.8;

int globalVariable;

void loop() {
   //Holds its value
   static int counter;
   //Resets its value
   int result;

   if (counter++ > 10) {
      increment();
   }
   result = getCount();
}

int getCount() {
   return globalVariable;
}

void increment() {
   globalVariable++;
}
```

Flash Variables

```
#include <avr/pgmspace.h>

char buffer[80]; //Buffer large enough to hold any Flash strings we make

//Custom function to auto-load strings into Flash
//PSTR is another special function (macro) for making program memory strings
#define P(string) (strcpy_P(buffer, PSTR(string)), buffer)

const char message[] PROGMEM = "This is a long warning message for the user.";

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

   //Use a special PROGMEM version of strcpy to get the string
   strcpy_P(buffer, message);
   //Print it out
   Serial.print(buffer);

   Serial.println( P("Another String to Display") );
}
```

Pointers and Functions

- Data type whose value is a memory address
- Reference to a variable or structure
- Dereference (*)
  - Return the object/value the pointer refers to
- Reference (&)
  - Return the memory address
- Access members/methods on referenced object with arrow (->) operator
  - `ptr->method()` is equivalent to `(*ptr).method()`
- Efficient way to pass large data to functions

```
Print *mySerial = &Serial;

void setup() {
   Serial.begin(9600);
   mySerial->println("Hello");
}

void loop() {
   static int counter;
   modify(&counter);

   byte msg[3] = {0x35, 0x4A, 0x4F};
   debug(msg);
}

void modify(int *val) {
   //Dereference pointer to change value
   *val += 2;
}

void debug(byte *buf) {
   Serial.print("Data: ");
   Serial.println(buf, sizeof(buf));
}
```

Libraries

- Collection of C/C++ files
  - May use Arduino code or pure MCU code
- Keywords.txt file for syntax highlights
  - KEYWORD1 for Class names
  - KEYWORD2 for Method names
- May include example sketches
- Installing a library
  1. Place library directory in `<sketchbook_path>/libraries`
  2. ???
  3. Profit
- Resources
  - Arduino Playground -> User Code Library
    - http://playground.arduino.cc
  - Adafruit GitHub
    - http://github.com/adafruit

```
libraries/
  +-MyLibrary/
    +-MyLibrary.h
    +-MyLibrary.cpp
    +-keywords.txt
    +-examples/
      +-MyExample/
      |  +-MyExample.ino
      +-SecondExample/
        +-SecondExample.ino
```

keywords.txt:

```
ClassName    KEYWORD1
methodA      KEYWORD2
methodB      KEYWORD2
```

Core Libraries

- EEPROM
  - Persisted storage
- LCD Displays
- SD Card storage (FAT16/FAT32)
- Servo
  - Control servo motors
- SPI
  - Serial Peripheral Interface
- SoftwareSerial
  - Serial communication on digital pins (i.e. bit-banging)
- Stepper
  - Control stepper motors
- Wire
  - TWI/I2C serial communication
- Scheduler (Due)
  - CPU Task Scheduler

DEMO: IDE and Basic Sketch

Digital I/O

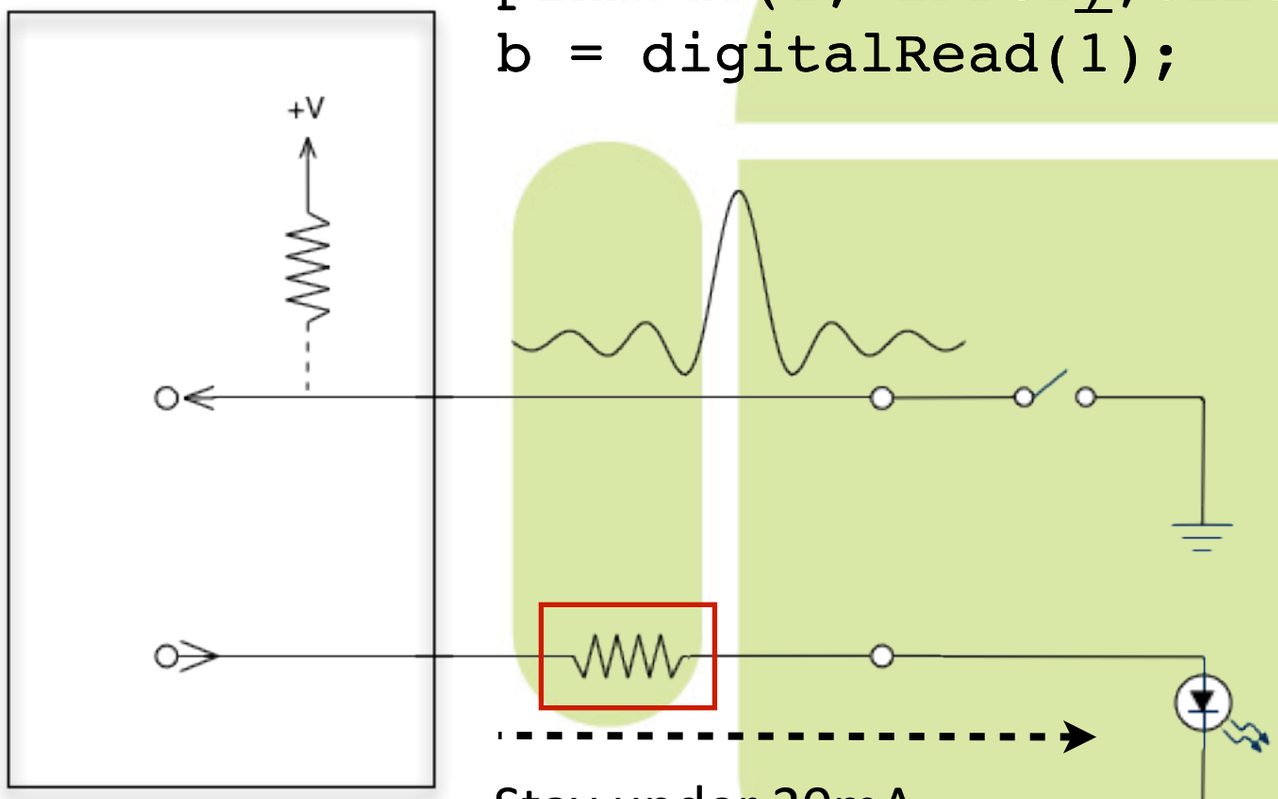
- Any port pin can be used as digital (GPIO)
- Takes one of two states
  - LOW (0V)
  - HIGH (+V)
- Set as input or output using `pinMode()`
  - INPUT
  - INPUT_PULLUP
  - OUTPUT
- As input, value read using `digitalRead()`
- As output, value set using `digitalWrite()`
  - Max output current approx. 20mA per pin

Digital I/O

```
pinmode(1, INPUT_PULLUP);
b = digitalRead(1);
```

→

```
pinmode(1, INPUT);
digitalWrite(1, HIGH);
b = digitalRead(1);
```



Stay under 20mA

```
pinmode(2, OUTPUT);
digitalWrite(2, HIGH);
```

Analog Inputs

- Analog-Digital Converter (ADC)
  - 0V to Vref converted into digital value
  - 10-bit resolution (1024 values, 4.88mV per step @ 5V)
  - 12-bit resolution (4096 values, 0.80mV per step @ 3.3V) available on Due
- `analogReference()` sets reference voltage
  - DEFAULT - Supply voltage
  - INTERNAL - Internally generated reference
  - EXTERNAL - Voltage on AREF pin
  - Ignored on Due boards
- `analogRead()`
  - Obtain a sample
- `analogReadResolution()` (Due only)
  - Set input read resolution value

A	D
0V	0
1.5V	306
2V	409
2.5V	511
3V	613
4V	818
4.995V	1022
4.997V	1022
5V	1023

Analog Outputs

- Supported on ~6-12 pins on board
- `analogWrite()` sets output value
  - Defaults to 8-bit (0-255)
- Digital-Analog Converter (DAC)
  - Due Only
  - Convert digital value into output voltage
  - `analogWriteResolution()` adjusts up to 12-bit
- Pulse Width Modulation (PWM)
  - Square wave simulating analog voltage output
  - Output duty cycle from 0 (always off) to 255 (always on)
  - Ex: `analogWrite(n, 127)` sets an even 50% duty cycle, or simulated 2.5V on the pin

```
//Default is 8-bit
analogWrite(139);
//Pin is 1.798V
analogWrite(140);
//Pin is 1.812V

analogWriteResolution(12);
analogWrite(2234);
//Pin is 1.800V
```

Serial

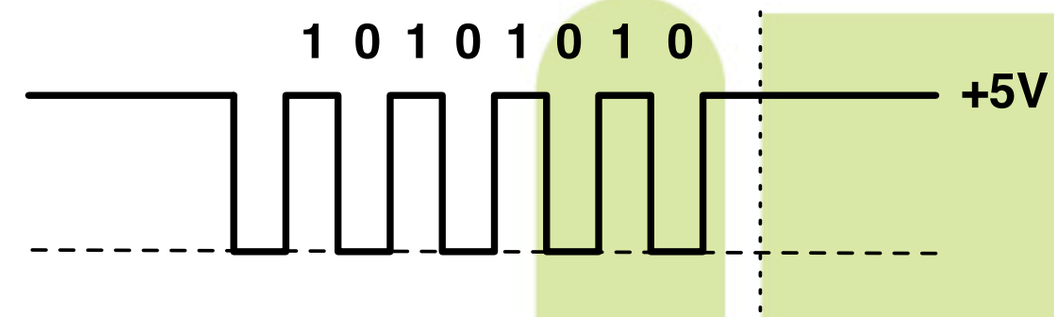
- Serial (UART)
  - Universal Asynchronous Receiver Transmitter
  - Pin 0 (RX) and Pin 1 (TX) on all devices
    - Connected to USB/Serial converter chip for programming/debug
    - Mega/Due have additional ports
  - Full-duplex character transmission
  - TTL Voltage Levels (5V/3.3V)
- Initialize with `Serial.begin()`
  - Baud rate
    - 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200
  - Character format (Optional: 8N1 default)
    - Data bits (5-8)
    - Parity (None, Even, Odd)
    - Stop Bits (1-2)

Serial

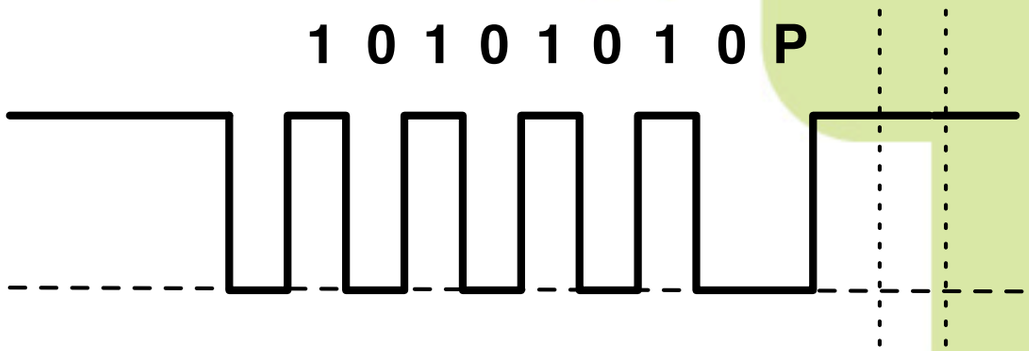
- Poll with `Serial.available()`
- Receive with `Serial.read()` and `Serial.readBytes()`
- Transmit with `Serial.write()`
- Use `Serial.print()` and `Serial.println()` for debugging
- Notifications with `serialEvent()`
  - Triggered after `loop()` if data exists in RX buffer
- SoftwareSerial ("Bit-Banging")
  - Serial data transfer over digital pins
  - Define pins to use for RX/TX
    - `SoftwareSerial serialPort(10, 11); //RX, TX`
  - Inverted signaling allowed

Serial Transmission

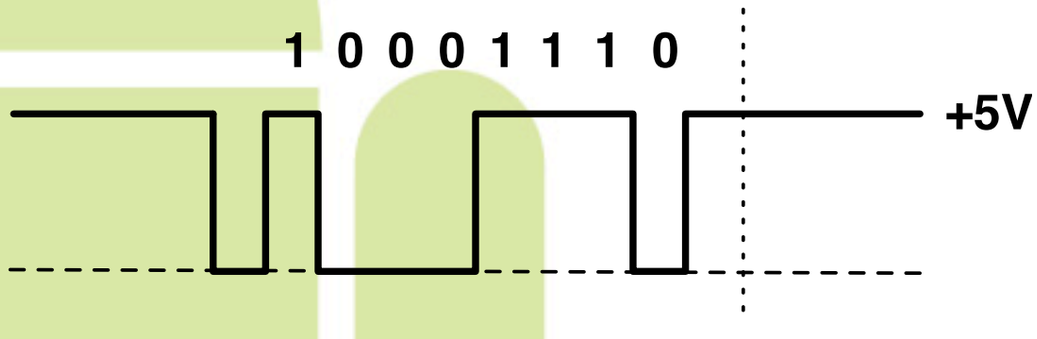
```
Serial.begin(9600);
Serial.write('U');
//'U'=0x55=01010101
```



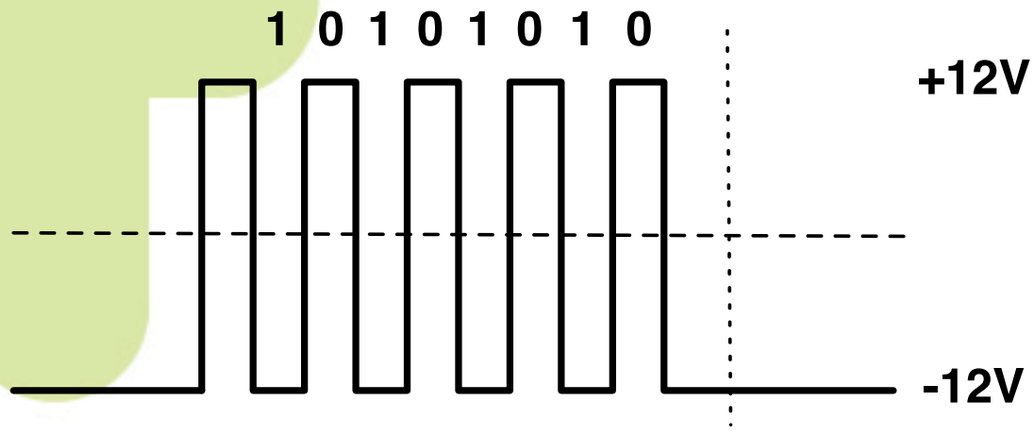

```
Serial.begin(9600, SERIAL_8E2);
Serial.write('U');
```




```
Serial.write('q');
'q' = 0x71 = 0b01110001
```



- RS-232 Data
  - Inverted signaling
  - Bipolar voltage swing
  - More noise-immune

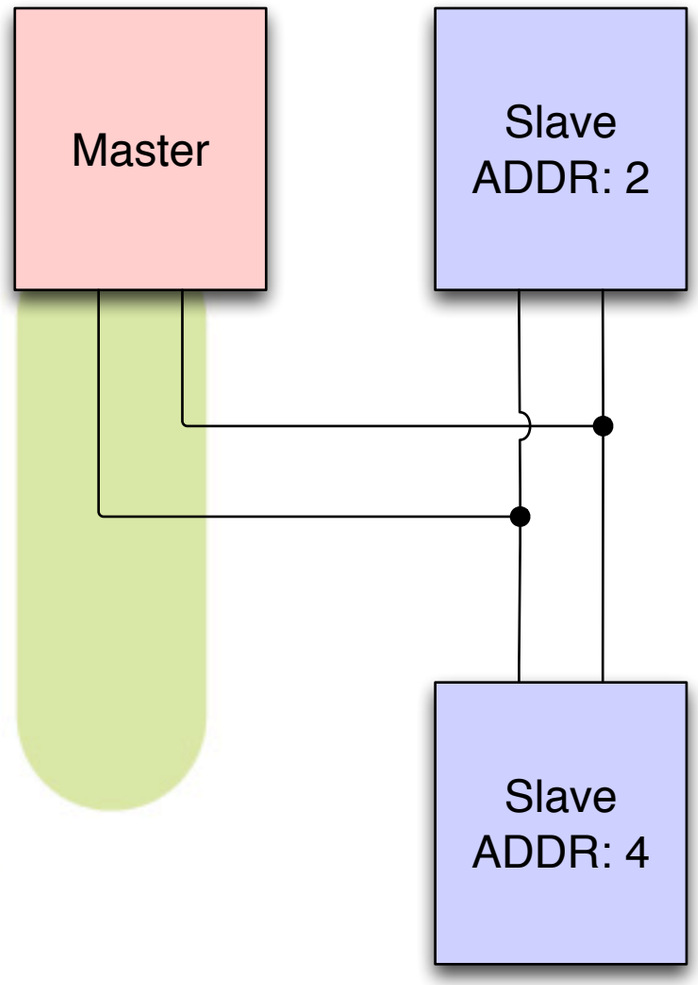


Serial

```
SoftwareSerial serialPort(10, 11); //RX, TX
byte reply[] = {0x11, 0x0A, 0x13};
byte lastRead;

void setup() {
   //Init the UART
   Serial.begin(115200);
   //Init a SoftwareSerial port
   serialPort.begin(9600);
}

void loop() {
   if (lastRead == '\n') { Serial.write(reply, sizeof(reply)); }
}

void serialEvent() {
   //Log every byte on software port
   while (Serial.available()) {
      lastRead = Serial.read();
      serialPort.print(next, HEX);
      serialPort.print("-");
   }
   serialPort.println();
}
```

Serial (Wire)

- I²C/TWI synchronous communication
  - Serial data (SDA) / Serial clock (SCL)
  - Data rate controlled by master via clock line
    - 100kHz mode -> 100kbps
- Half-duplex character transmission (master/slave)
- Communicate with smart peripheral devices
- Master
  - Init via `Wire.begin()`
  - Claim/release bus with `Wire.beginTransmission()` and `Wire.endTransmission()`
  - Make poll requests via `Wire.requestFrom()`
- Slave
  - Init with `Wire.begin(addr)`
    - Chosen address on the bus
  - Register request handler with `Wire.onRequest()`
  - Register data receive handler with `Wire.onReceive()`
- Transfer data with `Wire.write()` and `Wire.read()`



I²C/TWI Example

Master Device

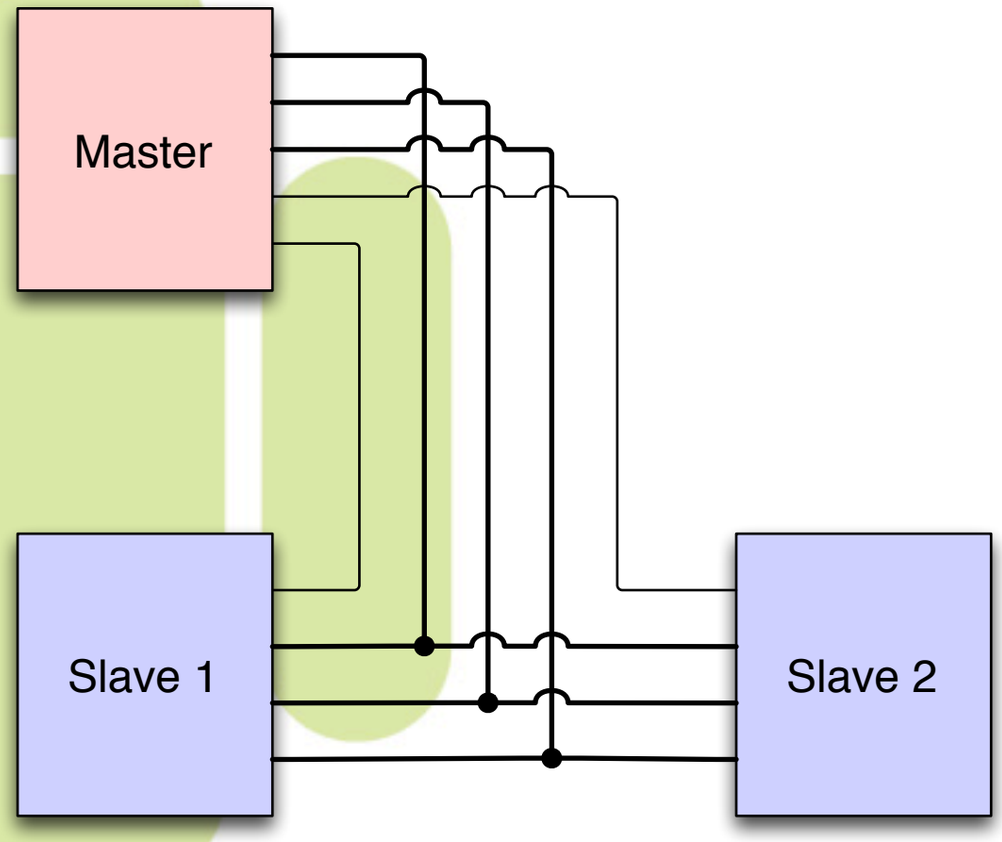
```
void setup() {
   Wire.begin();
{

void loop() {
   //Poll slave
   Wire.requestFrom(10, 1);
   //Read result
   while (Wire.available()) {
      byte b = Wire.read();
   }

   //Write to slave
   Wire.beginTransmission(10);
   Wire.write(0x06);
   Wire.endTransmission();

   delay(250);
}
```

Slave Device

```
void setup() {
   Wire.begin(10);
   Wire.onRequest(requestEvent);
   Wire.onReceive(receiveEvent);
}

byte val;
void loop() {
   //Read sensor input
   val = analogRead(3);
}

void requestEvent() {
   Wire.write(val);
}

void receiveEvent(int received) {
   while (Wire.available()) {
      byte b = Wire.read();
   }
}
```

Serial (SPI)

- Serial Peripheral Interface
  - Synchronous serial interface
  - Full-duplex communication
  - MISO, MOSI, CLK
  - Supports Master mode only
- Communicate with other MCUs or smart peripherals
- Slaves individually selected by master
- Must configure all transfer parameters
  - Bit order, clock polarity/phase, clock rate
- Initialize with `SPI.begin()`
- Transact bytes with `SPI.transfer()`
- Due Extensions
  - Multiple slaves
  - Automatic slave select



SPI Example

Uno/Mega Device

```
void setup() {
   //Configure SS pin
   pinMode(10, OUTPUT);

   SPI.begin();
   SPI.setBitOrder(LSBFIRST);
   //Set clock to 2MHz
   SPI.setClockDivider(SPI_CLOCK_DIV8);
   //Clock idle LOW, trigger LOW
   SPI.setDataMode(SPI_MODE0);
{

void loop() {
   //Select the slave
   digitalWrite(10, LOW);

   //Read and write occur in same transfer
   SPI.transfer(0x11);
   SPI.transfer(0x06);
   byte val = SPI.transfer(0x13);

   //Unselect the slave
   digitalWrite(10, HIGH);
}
```

Due Device

```
void setup() {
   //Enable two slave devices
   SPI.begin(4);
   SPI.begin(10);
   //Set clock to 2MHz
   SPI.setClockDivider(10, 42);
}

void loop() {
   //Lib handles chip select for us
   SPI.transfer(10, 0x11, SPI_CONTINUE);
   SPI.transfer(10, 0x06, SPI_CONTINUE);
   byte val = SPI.transfer(10, 0x13);

   //Communicate with secondary slave
   SPI.transfer(4, val);
}
```

Interrupts

- Event trigger to execute code
  - Outside normal event loop
  - Executes a function registered as the Interrupt Service Routine (ISR)
- Pauses normal execution during ISR
  - ISR should be kept small and tight
  - ISR manipulated variables should be ***volatile***
- Time-critical or event-drive code execution
- Enabled by default, many internal systems use interrupts
  - `delay()`
  - `millis()`
  - Serial RX
- Global Interrupt Control
  - Disable via `noInterrupts()` for time-critical code
  - Re-enable with `interrupts()`

```
void loop() {
  int state;

  if (state) {
    noInterrupts();
    //Critical code section
    interrupts();
  }
}
```

Interrupts

- External pin interrupt
  - 2 pins on Uno, 5 on Mega
    - Referenced by int.0 - int.5
  - All on Due
    - Referenced by pin number
- Function `attachInterrupt()` maps pin/trigger event to ISR
  - Transitions: CHANGE, RISING, FALLING
  - States: LOW
    - Due supports HIGH
- Remove with `detachInterrupt()`
- Debouncing switches
  - No delay or counters
  - May require hardware

```
//ISR
void buttonEvent() {
   Serial.println("TRIGGER!");
}

void setup() {
   Serial.begin(115200);

   //Pin 2 on Uno/Mega
   attachInterrupt(0, buttonEvent, FALLING);
}

void loop() {
   //Do something more interesting than
   //constantly monitoring a button
}
```

Application & Custom Library Demo

Come Find Me!

- Dave Smith
- Twitter: @devunwired
- Blog: http://wiresareobsolete.com
- Our Work: http://www.doubleencore.com
- Samples
  - http://github.com/devunwired/intro-arduino