How to do a Hardware Hack

UB Hackathon Fall 2022

Eric VanLieshout

Did you know?

- Computers are everywhere, all around you! Often they are used inside lots of things, like cars! And (sadly) toasters and refrigerators….
- Computers are often used as control systems, monitoring and regulating physical systems; examples: your car, factories, medical & surgery equipment.
- All computers have a real, physical hardware component in which electrons are shuffled about in some weird sand. Which brings me to …

Electron Wrangling

Computing is all about getting the right electrons in the right places at the right times.

Hardware - What is it?

- Silicon electron houses.
- Physical digital & analog devices - mechanical or electronic - that run programs, often with user input
  - Stored & fixed program computers
- What your software runs on (stored program computers)
- Devices that mediate between your program and the outside world, including users, other devices, and other software running on other devices.

Why get into hardware?

- Hardware is essential !! It is the Truth.
- But does messing around with an Arduino, STM32, ESP32, RPi, etc. really relate in any to ‘actual’ hardware, like a Core i7, GPUs, and whole systems like your home computer?

  YES!

+ Embedded Systems: these are LITERALLY EVERYWHERE in 2022.

Hardware hacks - why tho?

- Hardware allows your programs to interact with the outside world. It is that simple.

- Your programs can reach out into the world, and vice versa you can allow the outside world to feed data back to your programs.

Neat - but how?
The basics of hardware hacks

- Digital and Analog electrical signals
- Mostly digital, since we're using primarily digital systems.
- The world is analog, so many sensors do the work of converting analog to digital signals for you
- What this means for you: Your program will read in and output zero and one signals over electrical circuits

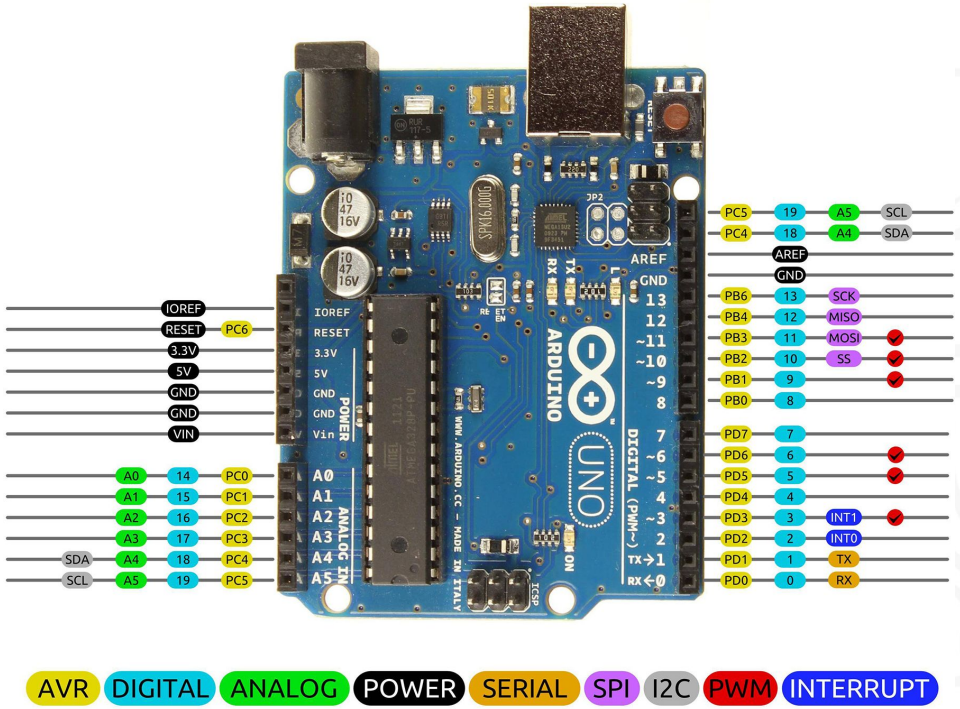
GPIO

General Purpose Input / Output

- These pins are designed so you can program them to either receive or output digital signals - 1 or 0
- Programming languages like Arduino's C++ and various Python and C/C++libraries available provide various Libraries / APIs for interacting with gpio pins on different devices / platforms.
  - In fact, it's mostly APIs! Even as you go deeper!
- Arduino ex: digitalWrite(pin_number, HIGH/LOW); writes a zero or 1 to the specified GPIO pin
- MicroPython ex:

```
pin = Pin(pin_number, Pin.IN/OUT)
pin.value(1/0)
```

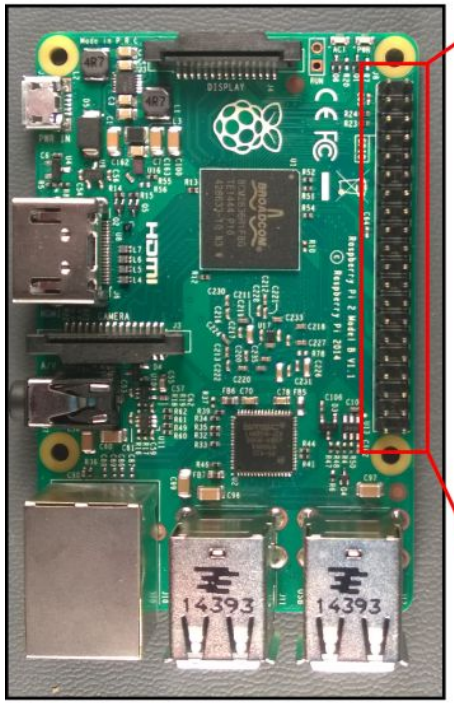
Arduino Uno Pinout



From microcontrollertips.com

Same pinout for Metro 328

RPi GPIO



Alternate Function						Alternate Function
	3.3V PWR	1	2	5V PWR		
I2C1 SDA	GPIO 2	3	4	5V PWR		
I2C1 SCL	GPIO 3	5	6	GND		
	GPIO 4	7	8	UART0 TX		
	GND	9	10	UART0 RX		
	GPIO 17	11	12	GPIO 18		
	GPIO 27	13	14	GND		
	GPIO 22	15	16	GPIO 23		
	3.3V PWR	17	18	GPIO 24		
SPI0 MOSI	GPIO 10	19	20	GND		
SPI0 MISO	GPIO 9	21	22	GPIO 25		
SPI0 SCLK	GPIO 11	23	24	GPIO 8		SPI0 CS0
	GND	25	26	GPIO 7		SPI0 CS1
	Reserved	27	28	Reserved		
	GPIO 5	29	30	GND		
	GPIO 6	31	32	GPIO 12		
	GPIO 13	33	34	GND		
SPI1 MISO	GPIO 19	35	36	GPIO 16		SPI1 CS0
	GPIO 26	37	38	GPIO 20		SPI1 MOSI
	GND	39	40	GPIO 21		SPI1 SCLK

Image from Microsoft RPi Docs

What about those other pins??

- Analog read/writes (we'll get to this later)
- Communication lines (UART, SPI, I2C, and many more)
- Power & Ground
- Misc functions
- Bus connections
- PLUS: Many GPIO pins have alternate functions

Good news - It's that simple!

Digital Signals:

- Write 1 or 0 on a pin to send that information to some device
- Read in a 1 or 0 on a pin to receive information from some device
- Your program can do … things … to decide when to write (output) a 1/0, i.e. when to communicate what to a connected device, or do different … things … based on reading (input) a 1/0

Good news - It's that simple!

Analog and PWM aren't any more difficult!

- Analog in converts a voltage to an integer within a range
- Analog out does the reverse
- PWM (Pulse Width Modulation) can be tricky, but Arduino and MicroPython provide easy APIs
  - PWM can be used to control LED brightness, servos, motors, and other devices that operate over a range.

Let’s do some things!

We will: blink some LEDs (GPIO output) and respond to a GPIO input signal.

Circuit basics

- Electrical signals go from higher voltage to lower voltage.
  - Power/ 5V to ground (GND), aka reference voltage.
- Must complete the loop!
  - Everything must be connected to GND
  - Usually the same ground, on the actual board.

State Machine Design

```
while (1) {     // main loop, aka superloop, aka void loop on Arduino

        while (state 1 conditions true) {
                State 1 behavior code;
        }

        while (state 2 conditions true) {
                State 2 behavior code;
        }

}
```

More devices / peripherals

MORE GOOD NEWS!

- More complicated devices and peripherals require more complicated interaction than simply two-state on-off signals
- There are TONS of libraries / APIs / example code demos to handle this for you!
- There is also PWM and analog Read/Write!
- DATASHEETS FTW !!

- Arduino ultrasonic sensor example and 2

Let's look at some hardware projects

A simple Arduino piano!

- Inputs utilize basic GPIO pin functionality
- Output (sound) uses a library to output signals with different frequencies and sends these out through the GPIO pins
- Uses a passive buzzer to transform the output signals (each a specified frequency) into sound
- Took about 3-4 hours to hack together, including a lot of careless mistakes from copy/paste and (initially) using the wrong buzzer

Let's look at some hardware projects

Clapper

- Uses the Raspberry Pi Pico & MicroPython + parts available here today!
- Uses a microphone - analog signal in! & LED - digital out
- Took about 15 minutes to make, starting from installing MicroPython

Music Visualizer

- Can be made by modifying the clapper
- Can be made BETTER by utilizing PWM (can be made better by someone other than me)

LCD Screen (demo)

Testing & Debugging your Hardware Project

Testing is fun and hands-on! You just gotta try it!

- Good testing is still strategic: you will want to systematically test your system, looking at both the code and the hardware.

Debugging is a bit different.

- Rule 1: Check your connections. Make sure the right wires are connected to the right places. Check also for floating pins and/or pull up/pull down resistors. Practice clean wiring.
- Rules 2 - the rest: just like software debugging, be systematic. Check boundary conditions. Try to isolate the problem to a specific part of the hardware or specific behavior.
- Special Rule: Use measuring and test equipment: multimeters, oscilloscopes, logic analyzers.

BLANK SLIDE

BONUS: Need multithreaded behavior on your Arduino? Use the FastLED library's powerful non-blocking macro:

EVERY_N_MILLISECONDS (num_ms) {}

A note on platforms

Different platforms have different capabilities, so it is worth doing a little research up-front.

Raspberry Pi Picos are available at the hackathon and can use Python

Arduino is a pretty safe bet for an all around platform.

Can combine different boards for different functionalities and use communication protocols so they can communicate & coordinate.

Welcome to hardware

Like anything with computers, it takes practice to build familiarity with hardware systems. The more you work with them, the more you will learn, and the deeper you will go.

The basic principle are the same everywhere, so starting with Arduinos and the like is the perfect way to learn the basic concepts while also doing fun, even useful projects.

Some great resources: Arduino docs & forums, datasheets, YouTube maker channels, Ben Eater (if you really want to dig into the details), and CSE 241 and 321.

Essential Hackathon Hardware Resources!

Set up the RPi Pico with MicroPython + Demos:
https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/0
(Also sets up Thony IDE which can also do CircuitPython)

Set up the Adafruit Metro 328 (i.e. Arduino Uno): https://www.adafruit.com/product/2488
- Requires special drivers to work with the Arduino IDE (see above link for d/l - easy set up)

LCD Screen Arduino Library Tutorials:
https://docs.arduino.cc/learn/electronics/lcd-displays
https://create.arduino.cc/projecthub/akshayjoseph666/interface-16x2-lcd-parallel-interface-with-arduino-uno-2e87e2

Go forth!

Some ideas:

- Computer system monitor
- Improve my piano & music visualizer designs
- DIY Smart Home devices / alarm system

Github repo containing all demo code + PDF of this slide deck
https://github.com/etvanlieshout/hardware_demos