PROTOTYPING EMBEDDED DEVICES

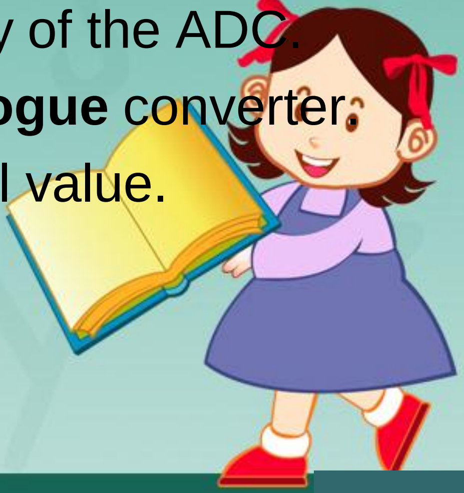
ELECTRONICS:

- When it comes to thinking about the electronics, it’s useful to split them into two main categories:

  Sensors: Sensors are the ways of getting information *into your device, finding out things about your surroundings.*

  Actuators: Actuators are the *outputs for the device—the motors, lights, and so on, which let your device do something to the outside world.*
- **Digital I/O**, which has only two states: a button can either be pressed or not; or an LED can be on or off.
- These states are connected via general-purpose input/output **(GPIO)** pins and map a **digital 0** in the processor to **0 volts(Reset)** in the circuit and the digital 1 to a set voltage(commonly 5V or 3.3V)
-

- If you want a more **nuanced** connection than just **on/off**, you need an **analogue signal**.
- If you want to run a **motor** at a speed other than **off or full-speed**, you need to feed it with a **voltage** somewhere between **0V** and its **maximum rating**.
- Because computers are purely **digital** devices, you need a way to translate between the **analogue voltages in the real world** and the **digital of the computer**.
- An **analogue-to-digital converter (ADC)** lets you measure varying voltages.
- They will convert the **voltage level** between **0V** and a predefined **maximum** into a number, depending on the accuracy of the ADC.
- The **flipside** of an **ADC** is a **DAC**, or **digital-to-analogue** converter.
- DACs let you generate varying voltages from a digital value.



SENSORS

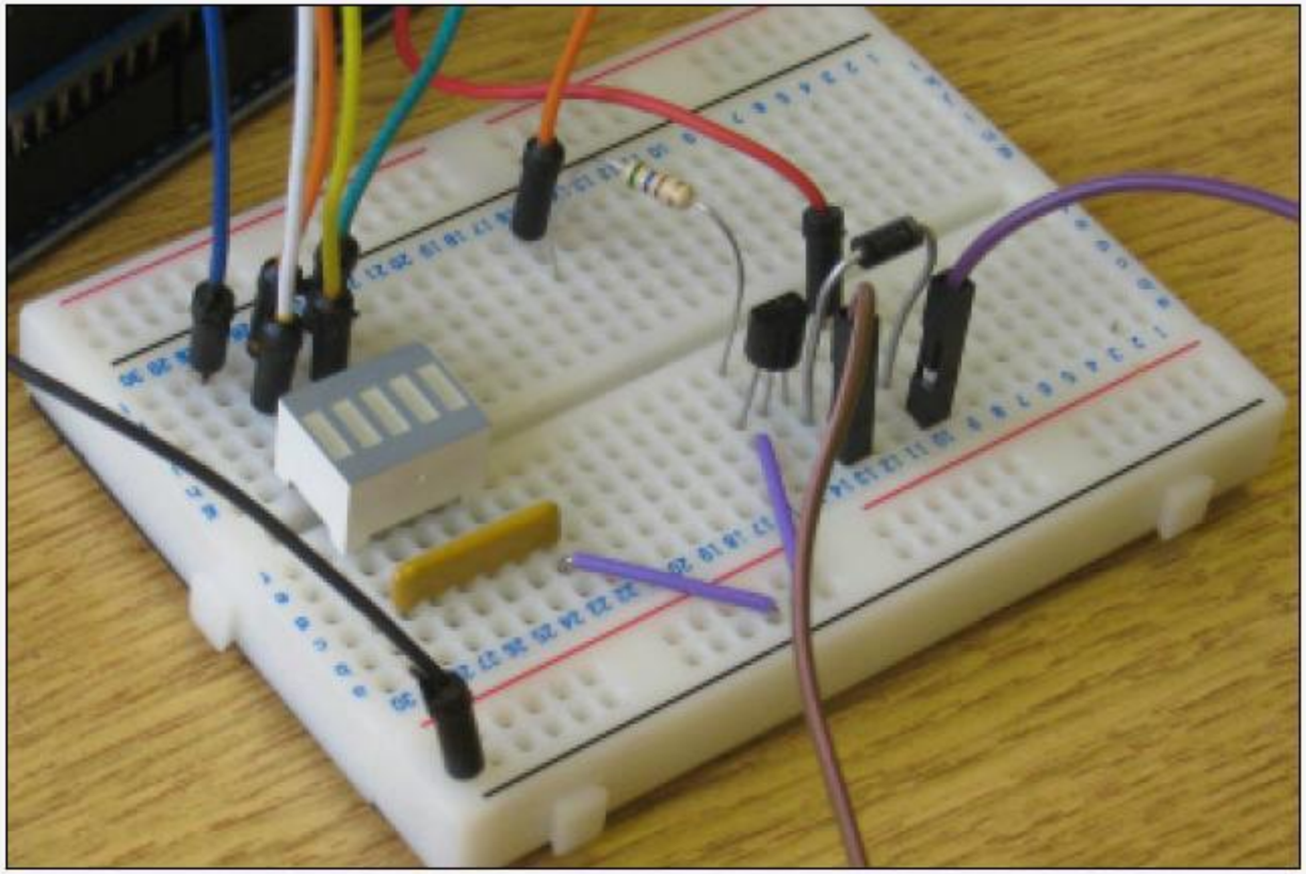
- **Pushbuttons** and **switches**, which are probably the **simplest sensors**, allow some **user input**.
- **Sensing** the **environment** is another easy option.
- **Light-dependent resistors (LDRs)** allow **measurement** of ambient **light levels**, **temperature sensors** allow you to know how **warm** it is, and sensors to measure **humidity** or **moisture** levels are easy to build.
- **Microphones** obviously let you **monitor sounds** and **audio**.
- **Distance-sensing** modules, which work by **bouncing** either an **infrared** or **ultrasonic signal** off objects, are readily available.

ACTUATORS

- One of the **simplest** and yet **most useful actuators** is **light**, because it is easy to create electronically and gives an obvious output.
- **Light-emitting diodes (LEDs)** typically come in **red** and **green** but also **white** and other colours.
- More **complicated visual outputs** also are available, such as **LCD screens** to display **text** or even **simple graphics**.
- You can wire up outputs to **speakers** to create more complicated **synthesised sounds**.
- **Stepper motors** can be moved in *steps, u*sually a fixed number of steps perform a full rotation.
- **DC motors** simply move at a given speed when told to.
- Both types of motor can be **one-directional** or move in **both directions**.

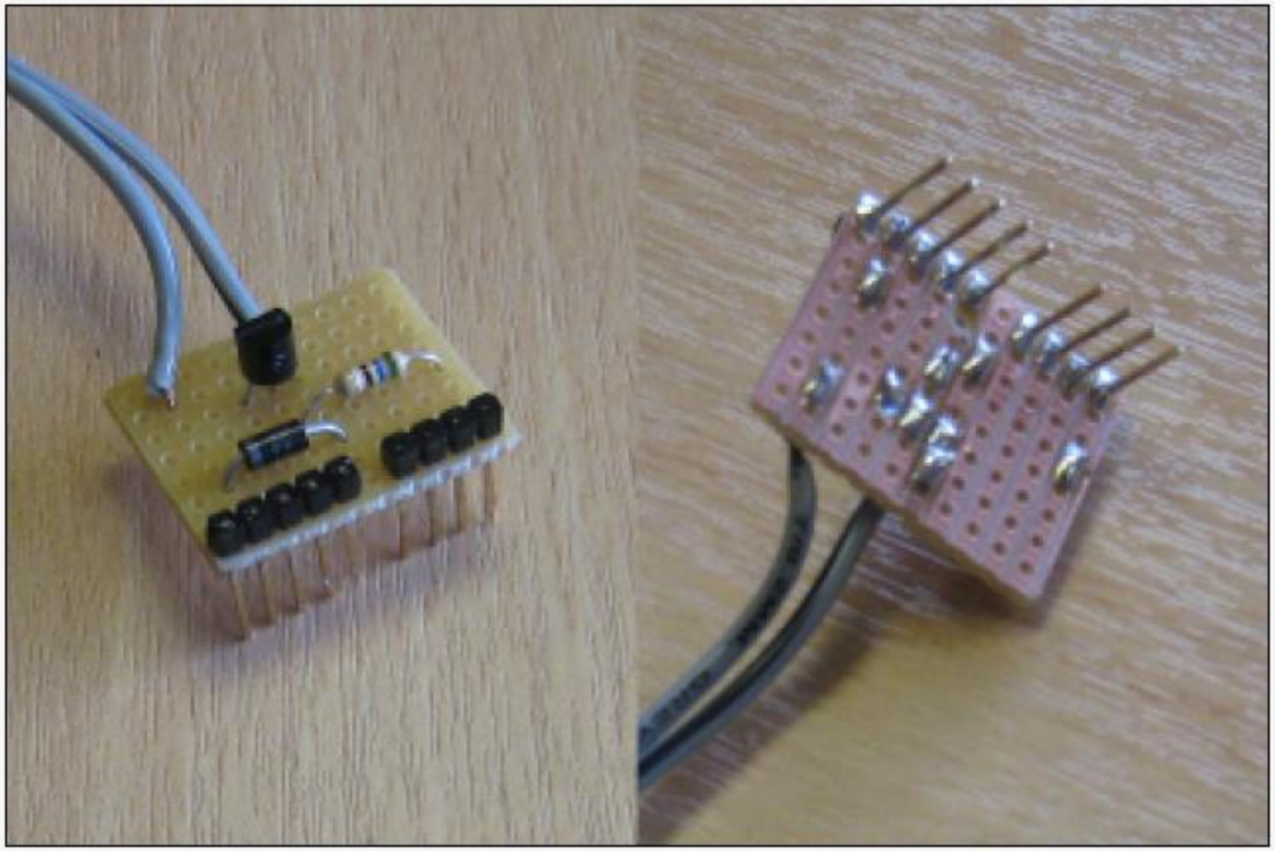
SCALING UP THE ELECTRONICS

- • From the perspective of the electronics, the starting point for prototyping is usually a “**breadboard**”.


The breadboard.

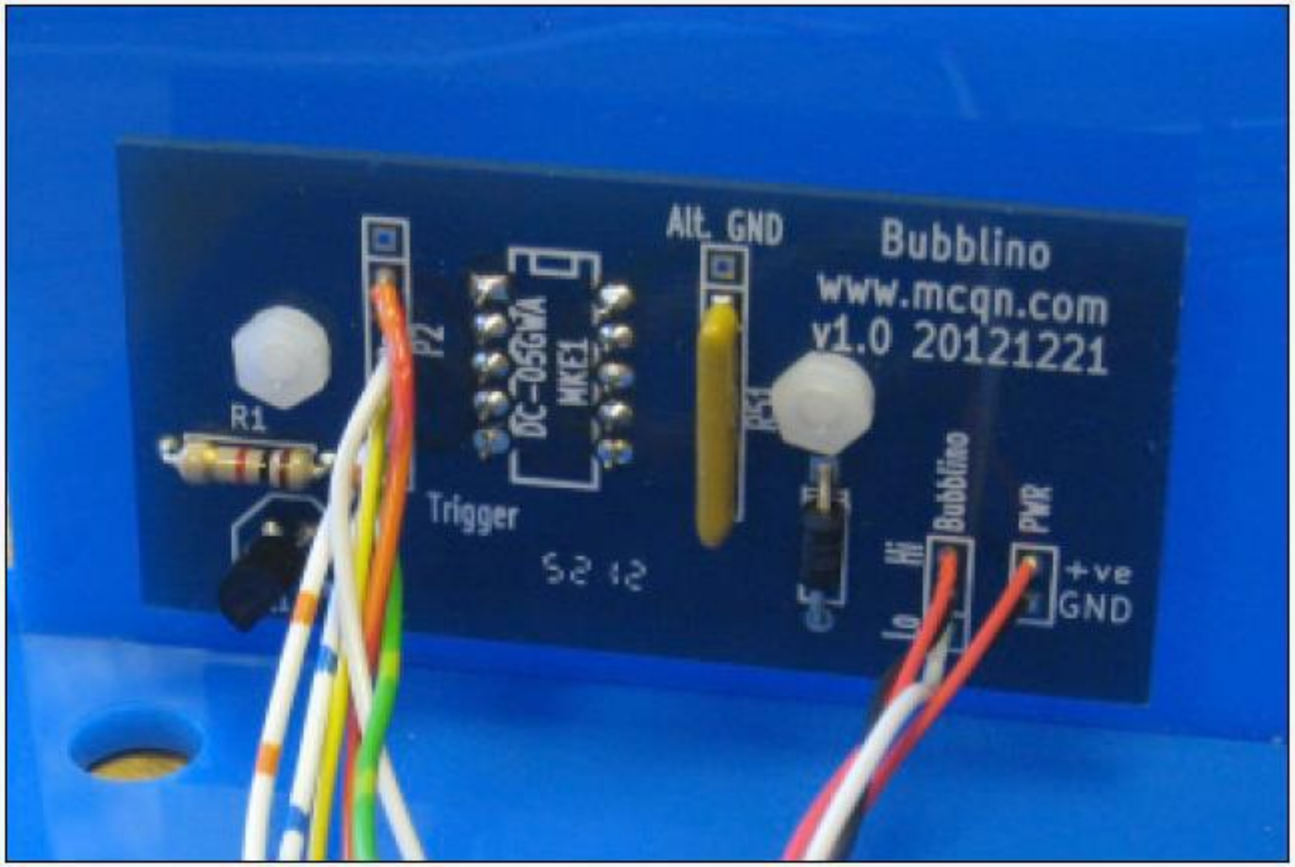
SCALING UP THE ELECTRONICS

- • it’s common to **solder** the **components** onto some **protoboard (Stripboard)**, which may be sufficient to make the circuit more **permanent**.


The stripboard.

SCALING UP THE ELECTRONICS

- • Moving beyond the **protoboard (stripboard)** option tends to involve learning how to layout a **PCB**.



The PCB.

EMBEDDED COMPUTING BASICS

MICROCONTROLLERS:

- • It combines the **processor**, **RAM**, and **storage** onto a **single chip**, which means they are much more **specialised**, **smaller** and also **easier** to build into a custom design.
- • Microcontrollers are very **limited** in their **capabilities**.
- • Usually, they offer **RAM** capabilities measured in **kilobytes** and **storage** in the **tens of kilobytes**.
- • The **modern chips** are much **smaller**, **require less power**, and run about **five times faster** than their 1980s counterparts.
- • Few examples include **Atmel**, **Microchip**, **NXP**, **Texas** Instruments, etc..
- • The **devices** using them are focused on performing **one task**.

SYSTEM-ON-CHIPS:

- In between the **low-end microcontroller** and a **full-blown PC** is the **SoC**.
- They combine a **processor** and a **number of peripherals** onto a **single chip** but usually have more capabilities.
- The **processors** usually **range** from a **few hundred megahertz** to the **gigahertz**.
- **RAM** measured in **megabytes** rather than **kilobytes**.
- **Storage** for SoC modules tends **not to be included** on the **chip**, with **SD cards** being a popular solution.
- The greater capabilities of **SoC** mean that they need some sort of **operating system** to marshal their resources.
- A wide selection of **embedded operating systems**, both **closed** and **open** source, is available and from both specialised embedded providers and the big OS players, such as **Microsoft** and **Linux**.

CHOOSING YOUR PLATFORM:

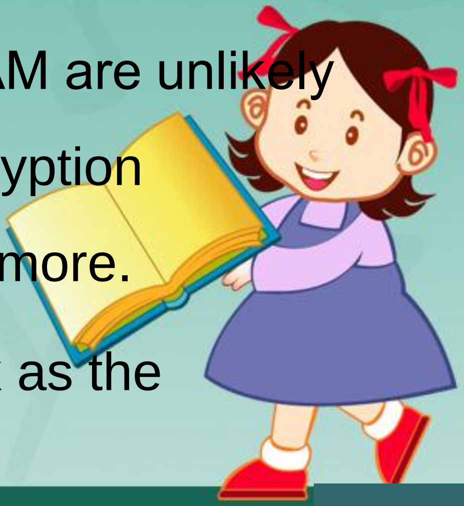
Processor Speed

- The **processor speed**, or **clock speed** → How fast it can process the individual instructions in the machine code for the program it's running.
- Some processors may **lack hardware support for floating-point** calculations, so if the code involves a lot of complicated mathematics, slower processor **with hardware floating-point** support could be **faster** than a **slightly higher performance processor** without it.
- Microcontrollers tend to be clocked at **speeds** in the **tens of MHz**, whereas **SoCs** run at **hundreds of MHz** or possibly low **GHz**.
- If your project doesn't require heavyweight processing then some sort of microcontroller will be fast enough.
- If your device will be crunching lots of data(ex: processing video in real time) then you'll be looking at a SoC platform.

CHOOSING YOUR PLATFORM:

RAM

- • RAM provides the **working memory** for the system.
- • If you have more **RAM**, you may be able to do more things or have more flexibility over your choice of coding algorithm.
- • It is difficult to give exact guidelines to the amount of RAM you will need, as it will vary from project to project.
- • However, microcontrollers with less than 1KB of RAM are unlikely to be of interest, and if you want to run standard encryption protocols, you will need at least 4KB, and preferably more.
- • For SoC boards, particularly if you plan to run Linux as the operating system, we recommend at least 256MB.



CHOOSING YOUR PLATFORM:

Networking

• How your device **connects** to the **rest of the world** is a key consideration for Internet of Things products.

• **Wired Ethernet** is often the **simplest** for the user—generally plug and play—and **cheapest**, but it requires a physical cable.

• **Wireless** solutions obviously avoid that requirement but introduce a more complicated configuration.

• **WiFi** is the most widely deployed to provide an existing **infrastructure** for connections, but it can be more **expensive** and less optimized for **power** consumption than some of its competitors.

CHOOSING YOUR PLATFORM:

USB

• If your device can rely on a **more powerful** computer being nearby, **tethering** to it via USB can be an easy way to provide both **power** and **networking**.

• You can buy some of the microcontrollers in versions which include support for USB.

• Instead of the microcontroller presenting itself as a device, some can also act as the **USB "host"**.

CHOOSING YOUR PLATFORM:

Power Consumption

• **Faster processors** are often more **power hungry** than slower ones.

• For **devices** which might be **portable** or rely on an unconventional **power supply** (batteries, solar power) depending on where they are installed, power consumption may be an issue.

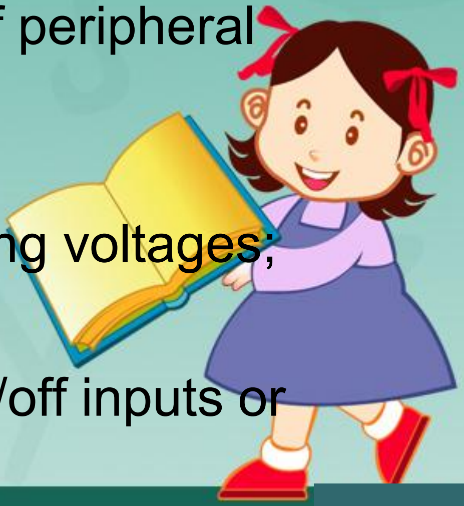
• However, **processors** may have a minimal power-consumption **sleep mode.**

• This mode may allow you to use a **faster processor** to quickly perform operations and then return to **low-power sleep**.

CHOOSING YOUR PLATFORM:

Interfacing with Sensors and Other Circuitry

- In addition to talking to the Internet, your device needs to interact with something else—either
    - sensors to gather data about its environment;
    - Or motors, LEDs, screens, and so on, to provide output.
- You could connect to the circuitry through some sort of peripheral bus—SPI and I2C being common ones—
- or through ADC or DAC modules to read or write varying voltages;
- or through generic GPIO pins, which provide digital on/off inputs or outputs.



Physical Size and Form Factor

• Nowadays, the **size** is governed by the **number of connections** it needs to make to the surrounding components on the **PCB**.

• The limit to the size that each **connection** can be reduced to is then governed by the capabilities and **tolerances of your manufacturing process**.

• Some **surface-mount designs** are big enough for home-etched PCBs and can be **handsoldered**.

• Due to these **trade-offs in size** versus **manufacturing complexity**, many **chip designs** are available in a number of different form factors, known as **packages**.

ARDUINO

- The Arduino team's focus on simplicity rather than raw performance

- The Uno features an ATmega328 microcontroller and a USB socket for connection to a computer.

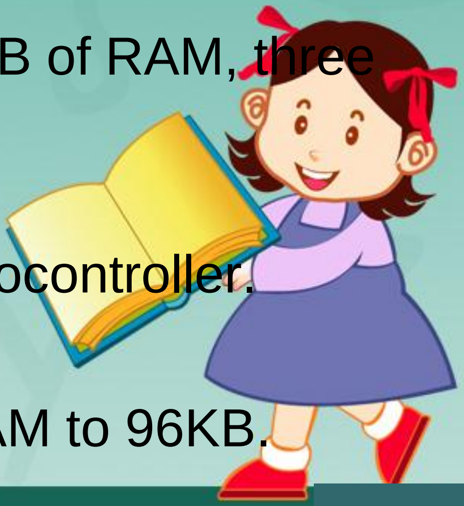
- It has 32KB of storage and 2KB of RAM

- The Uno also provides 14 GPIO pins.

- Arduino Mega 2560 provides 256KB of Flash storage, 8KB of RAM, three more serial ports, a massive 54 GPIO pins.

- the more recent Arduino Due has a 32-bit ARM core microcontroller.

- Its specs are similar to the Mega's, although it ups the RAM to 96KB.



DEVELOPING ON THE ARDUINO

• Using a single USB cable, you can not only power the board but also push your code onto it, and (if needed) communicate with it

• For example, for debugging or to use the computer to store data retrieved by the sensors connected to the Arduino.

Integrated Development Environment

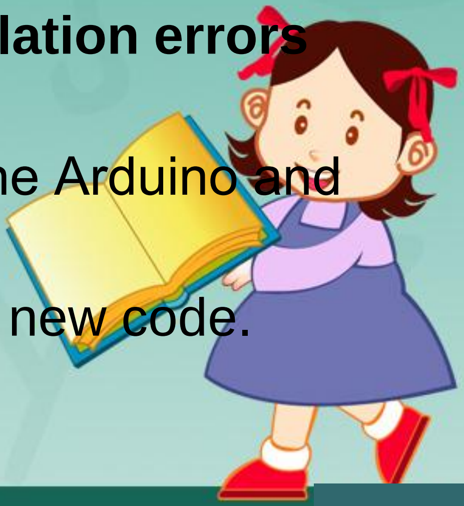
• You usually develop against the Arduino using the integrated development environment (IDE) that the team supply at http://arduino.cc.

• Most Arduino projects consist of a single file of code, so you can think of the IDE mostly as a simple file editor.

• The controls that you use the most are those to check the code (by compiling it) or to push code to the board

Pushing Code

- Connecting to the **board** should be relatively straightforward via a **USB** cable.
- Sometimes you might have issues with the **drivers** or with **permissions** on the USB port.
- You need to choose the correct **serial port** and the **board type** (look carefully at the labelling on your board and its CPU to determine which option to select).
- When your **setup** is correct, the process of pushing code is generally simple:
- first, the **code** is checked and compiled, with any **compilation errors** reported to you.
- If the code compiles successfully, it gets transferred to the Arduino and stored in its **flash memory**.
- At this point, the Arduino **reboots** and starts running the new code.



Operating System

• The **Arduino** doesn't, by default, run an **OS** as such, only the **bootloader**, which simplifies the code-pushing process described previously.

• When you **switch on the board**, it simply **runs the code** that you have **compiled** until the board is **switched off again**.

• It is, however, possible to upload an **OS** to the Arduino, usually a lightweight real-time operating system **(RTOS)**.

• The main **advantage** of one of these operating systems is their **built-in support for multitasking**.

• It is even possible to **compile** code without using the **IDE** but by using the toolset for the Arduino's chip—for example the **avr-gcc toolset**.

• The **avr-gcc toolset** (www.nongnu.org/avr-libc/) is the **collection of programs** that let you compile code to run on the AVR chips.

Language

• The language usually used for Arduino is a slightly modified version of C++.

• It includes some libraries used to read and write data from the I/O pins provided on the Arduino and to do some basic handling for “interrupts”.

• The code needs to provide only two routines:

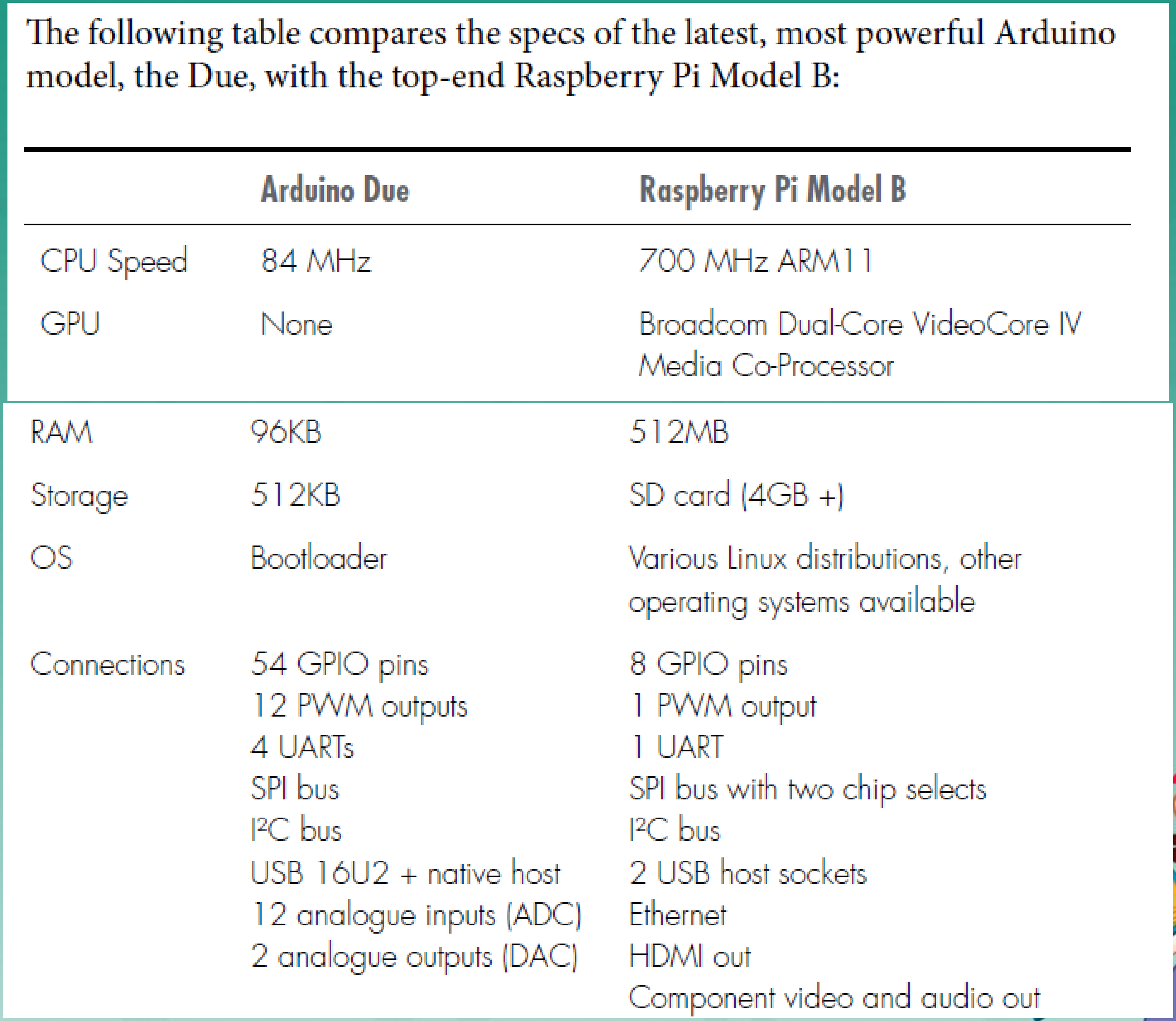
- **setup():** • This routine is run once when the board first boots.

  • You could use it to set the modes of I/O pins to input or output or to prepare a data structure which will be used throughout the program.

- **loop():** • This routine is run repeatedly in a tight loop while the Arduino is switched on.

RASPBERRY PI

- The Raspberry Pi, unlike the Arduino, wasn’t designed for physical computing at all, but rather, for education.
- • Uses Broadcom BCM2835 system-on-chip, powerful graphics processing unit (GPU), capable of high-definition video and fast graphics rendering.
- • the Raspberry Pi is effectively a computer that can run a real, modern operating system, communicate with a keyboard and mouse, talk to the Internet, and drive a TV/monitor with high-resolution graphics.



The following table compares the specs of the latest, most powerful Arduino model, the Due, with the top-end Raspberry Pi Model B:

	Arduino Due	Raspberry Pi Model B
CPU Speed	84 MHz	700 MHz ARM11
GPU	None	Broadcom Dual-Core VideoCore IV Media Co-Processor
RAM	96KB	512MB
Storage	512KB	SD card (4GB +)
OS	Bootloader	Various Linux distributions, other operating systems available
Connections	54 GPIO pins 12 PWM outputs 4 UARTs SPI bus I²C bus USB 16U2 + native host 12 analogue inputs (ADC) 2 analogue outputs (DAC)	8 GPIO pins 1 PWM output 1 UART SPI bus with two chip selects I²C bus 2 USB host sockets Ethernet HDMI out Component video and audio out

DEVELOPING ON THE RASPBERRY PI

Operating System

• Although many operating systems can run on the Pi, we recommend using a popular Linux distribution, such as

Raspbian:

• Released by the Raspbian Pi Foundation, It is based on Debian.

• This is the default "official" distribution and is certainly a good choice for general work with a Pi.

Programming Language

- One choice to be made is which programming language and environment you want to use.

- Python is a good language for educational programming (and indeed the name “Pi” comes initially from Python).

Contrast Python with C++

- Python, as with most high-level languages, **compiles** to relatively **large** (in terms of memory usage) and **slow code**, compared to C++.
- Python handles **memory management** automatically.
- Linux itself arguably has some issues for "**real-time" use**.
- An **Arduino** runs only the **one set of instructions**, in a tight loop, until it is turned off or crashes.