



PROTOTYPING EMBEDDED DEVICES

This chapter starts with a look at electronics because whatever platform you end up choosing, the rest of the circuitry that you will build to connect it to will be pretty much the same.



Prototyping Embedded Devices

Content

- Electronics
- Sensors
- Actuators
- Scaling Up the Electronics
- Embedded Computing Basics
- Microcontrollers
- System-on-Chips
- Choosing Your Platform
- Arduino, Developing on the Arduino, Some Notes on the Hardware, Openness.
- Raspberry Pi, Cases and Extension Boards, Developing on the Raspberry Pi, Some Notes on the Hardware, Openness.

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.


► Within both categories, the electronic components can talk to the computer in a number of ways.

► The simplest is through 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 usually connected via general-purpose input/output (GPIO) pins and map a digital 0 in the processor to 0 volts in the circuit and the digital 1 to a set voltage, usually the voltage that the processor is using to run (commonly 5V or 3.3V).

► If you want a more nuanced connection than just on/off, you need an analogue signal. For example, if you wire up a potentiometer to let you read in the position of a rotary knob, you will get a varying voltage depending on the knob's location. Similarly, 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.


ELECTRONICS



- ▶ Because computers are purely digital devices, you need a way to translate between the analogue voltages of the real world and the digital of the computer.
 - ▶ An analogue-to-digital converter (ADC) lets you measure varying voltages.
 - ▶ Microcontrollers often have a number of these converters built in.
 - ▶ For more complicated sensors and modules, there are interfaces such as Serial Peripheral Interface (SPI), I2C bus and Inter-Integrated Circuit (I²C).
 - ▶ These standardized mechanisms allow modules to communicate, so sensors or things such as Ethernet modules or SD cards can interface to the microcontroller.
- 


SENSORS



- Pushbuttons and switches, which are probably the simplest sensors, allow some user input. Potentiometers (both rotary and linear) and rotary encoders enable you to measure movement.
 - Sensing the environment is another easy option. Light-dependent resistors (LDRs) allow measurement of ambient light levels, thermistors and other temperature sensors allow you to know how warm it is, and humidity sensors to measure humidity or moisture levels are easy to build.
 - Microphones obviously let you monitor sounds and audio, but piezo elements (used in certain types of microphones) can also be used to respond to vibration.
 - Distance-sensing modules, which work by bouncing either an infrared or ultrasonic signal off objects, are readily available and as easy to interface to as a potentiometer.
- 

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.
 - RGB LEDs have a more complicated setup but allow you to mix the levels of red, green, and blue to make whatever colour of light you want.
 - More complicated visual outputs also are available, such as LCD screens to display text or even simple graphics.
 - Piezo elements, as well as *responding* to vibration, can be used to *create* it, so you can use a piezo buzzer to create simple sounds and music. Alternatively, you can wire up outputs to speakers to create more complicated synthesized sounds.
- 

ACTUATORS

Of course, for many tasks, you might also want to use components that *move* things in the real world. Solenoids can be used to create a single, sharp pushing motion, which could be useful for pushing a ball off a ledge or tapping a surface to make a musical sound.



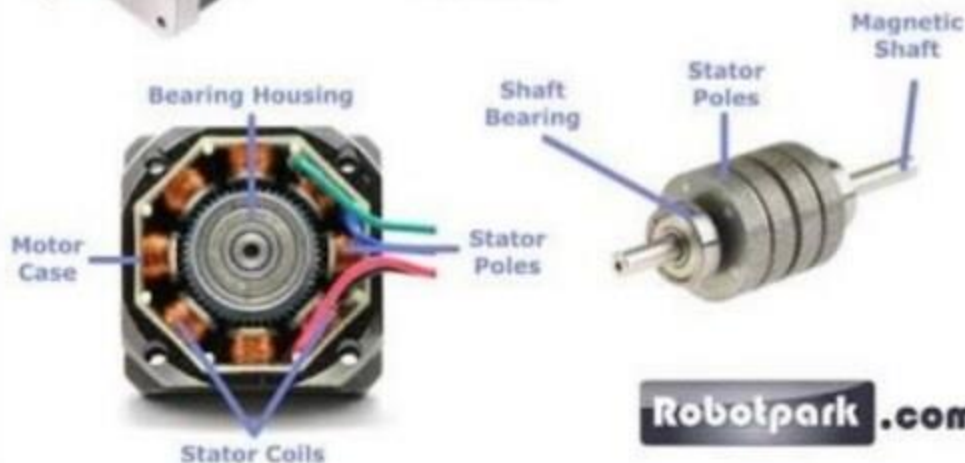
ACTUATORS

- More complicated again are motors. Stepper motors can be moved in steps, as the name implies. Usually, 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.



What is a Stepper Motor ?

A stepper motor is a digital device, more precisely a digital DC motor. Stepper or Stepper Motor allows you to select a certain degree of movement. Rather than making a whole spin it can divide the spin into smaller parts.



Robotpark.com

ACTUATORS

► Alternatively, if you want a motor that will turn to a given angle, you would need a servo. Although a servo is more controllable, it tends to have a shorter range of motion



Prototyping Embedded Devices

Content

✓ Electronics

✓ Sensors

✓ Actuators

○ Scaling Up the Electronics

○ Embedded Computing Basics

○ Microcontrollers

○ System-on-Chips

○ Choosing Your Platform

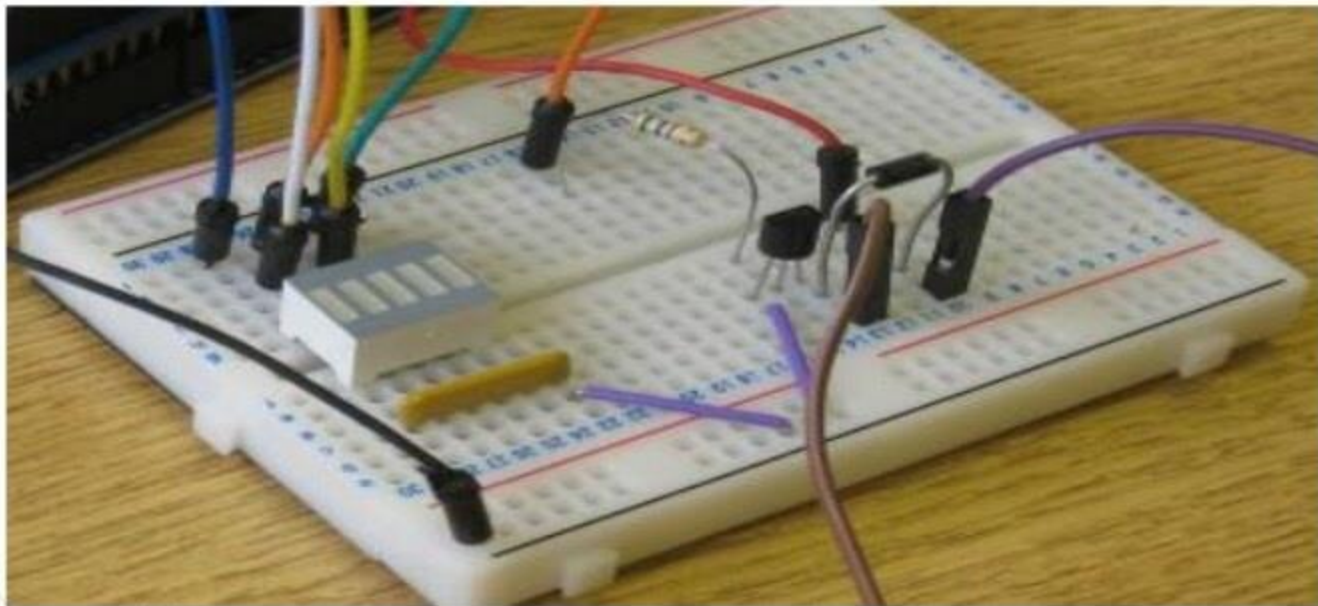
○ Arduino, Developing on the Arduino, Some Notes on the Hardware, Openness

○ Raspberry Pi, Cases and Extension Boards, Developing on the Raspberry Pi

Some Notes on the Hardware, Openness.

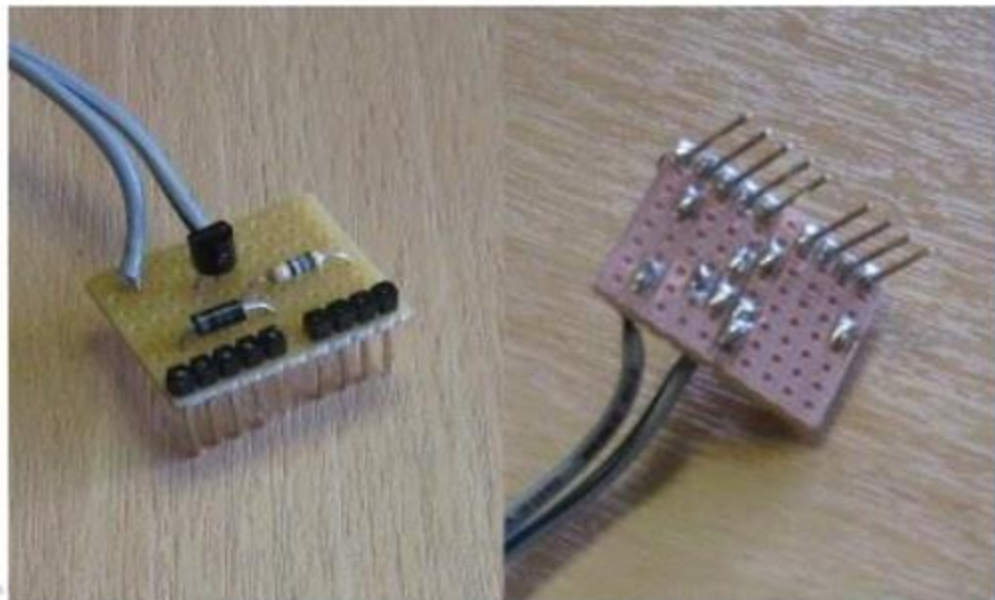
SCALING UP THE ELECTRONICS

- From the perspective of the electronics, the starting point for prototyping is usually a “breadboard”. This lets you push-fit components and wires to make up circuits without requiring any soldering and therefore makes experimentation easy.
- When you're happy with how things are wired up, it's common to solder the components onto some protoboard, which may be sufficient to make the circuit more permanent and prevent wires from going astray.



SCALING UP THE ELECTRONICS

- If you need to make many copies of the circuit, or if you want a professional finish, you can turn your circuit into a PCB.
- This makes it easier to build up the circuit because the position of each component will be labelled, there will be holes only where the components go, and there will be less chance of short circuits because the tracks between components will be protected by the solder resist



Strip Board



Printed Circuit Board PCB

EMBEDDED COMPUTING BASICS

- **MICROCONTROLLERS:** Internet of Things devices take advantage of more tightly integrated and miniaturised solutions—from the most basic level of microcontrollers to more powerful system-on-chip (SoC) modules. These systems combine the processor, RAM, and storage onto a single chip, which means they are much more specialised, smaller than their PC equivalents, and also easier to build into a custom design.
- Unlike the market for desktop computer processors, which is dominated by two manufacturers (Intel and AMD), the microcontroller market consists of many manufacturers.
- A better comparison is with the automotive market. In the same way that there are many different car manufacturers, each with a range of models for different uses, so there are lots of microcontroller manufacturers (Atmel, Microchip, NXP, Texas Instruments, to name a few), each with a range of chips for different applications.
- The ubiquitous Arduino platform is based around Atmel's AVR Atmega family of microcontroller chips. The on-board inclusion of an assortment of GPIO pins and ADC circuitry means that microcontrollers are easy to wire up to all manner of sensors, lights, and motors.
- Because the devices using them are focused on performing one task, they can dispense with most of what we would term an operating system, resulting in a simpler and much slimmer code footprint than that of a SoC or PC solution.

SYSTEM-ON-CHIPS

- In between the low-end microcontroller and a full-blown PC sits the SoC (for example, the BeagleBone or the Raspberry Pi). Like the microcontroller, these SoCs 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, nudging into the gigahertz for top-end solutions, and include 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.



Prototyping Embedded Devices

Content

- ✓ Electronics
- ✓ Sensors
- ✓ Actuators
- ✓ Scaling Up the Electronics
- ✓ Embedded Computing Basics
- ✓ Microcontrollers
- ✓ System-on-Chips
 - Choosing Your Platform
 - Arduino, Developing on the Arduino, Some Notes on the Hardware, Openness
 - Raspberry Pi, Cases and Extension Boards, Developing on the Raspberry Pi, Some Notes on the Hardware, Openness.

CHOOSING YOUR PLATFORM

- ▶ The platform you choose depends on the particular blend of price, performance, and capabilities that suit what you're trying to achieve.
- ▶ And just because you settle on one solution, that doesn't mean somebody else wouldn't have chosen a completely different set of options to solve the same problem.
- ▶ **Consider the following while choosing a platform:**
 - ▶ Processor Speed
 - ▶ RAM
 - ▶ Networking
 - ▶ USB
 - ▶ Power Consumption
 - ▶ Interfacing with Sensors and Other Circuitry
 - ▶ Physical Size and Form Factor

CHOOSING YOUR PLATFORM : Processor Speed

- ▶ The processor speed, or clock speed, of your processor tells you how fast it can process the individual instructions in the machine code for the program it's running.
- ▶ Naturally, a faster processor speed means that it can execute instructions more quickly.
- ▶ You might also make a comparison based on millions of instructions per second (MIPS)
- ▶ Microcontrollers tend to be clocked at speeds in the tens of MHz, whereas SoCs run at hundreds of MHz or possibly low GHz.
- ▶ **Example:** If your project doesn't require heavyweight processing—for example, if it needs only networking and fairly basic sensing—then some sort of microcontroller will be fast enough. If your device will be crunching lots of data—for example, processing video in real time—then you'll be looking at a SoC platform.
- ▶ **Example:** Some processors may lack hardware support for floating-point calculations, so if the code involves a lot of complicated mathematics, a by-the-numbers slower processor with hardware floating-point support could be faster than a slightly higher performance processor without it.

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 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.
- ▶ Other short-range wireless can offer better power-consumption profiles or costs than WiFi but usually with the trade-off of lower bandwidth. ZigBee is one such technology, aimed particularly at sensor networks and scenarios such as home automation.
- ▶ The recent Bluetooth LE protocol (also known as Bluetooth 4.0) has a very low power-consumption profile similar to ZigBee's and could see more rapid adoption due to its inclusion into standard Bluetooth chips included in phones and laptops.
- ▶ For remote or outdoor deployment, little beats simply using the mobile phone networks. For low bandwidth, higher-latency communication, you could use something as basic as SMS; for higher data rates, you will use the same data connections, like 3G, as a smartphone.

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, so choosing one of them **reduces the need for an extra chip in your circuit**.
- Instead of the microcontroller presenting itself as a device, some can also act as the USB “host”.
- This configuration lets you connect items that would normally expect to be connected to computer devices such as phones, for example, using the Android ADK, additional storage capacity, or WiFi dongles.
- Devices such as WiFi dongles often depend on additional software on the host system, such as networking stacks, and so are better suited to the more computer-like option of SoC.

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.
- Even with access to mains electricity, the power consumption may be something to consider because low consumption may be a desirable feature.
- 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.
- Therefore, a more powerful processor may not be a disadvantage even in a low-power embedded device

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.
- Different microcontrollers or SoC solutions offer different mixtures of these interfaces in different numbers.

CHOOSING YOUR PLATFORM : **Physical Size and Form Factor**

- ▶ With the traditional **through-hole design**, most commonly used for homemade circuits, the legs of the chip are usually spaced at 0.1" intervals.
- ▶ Even if your chip has relatively few connections to the surrounding circuit—16 pins is nothing for such a chip—you will end up with over 1.5" (~4cm) for the perimeter of your chip.
- ▶ You can pack the legs closer together with **surface-mount technology** because it doesn't require holes to be drilled in the board for connections.
- ▶ Combining that with the trick of hiding some of the connections on the underside of the chip means that it is possible to use the complex designs
- ▶ **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 hand-soldered.
- ▶ Others require professionally produced PCBs and accurate pick-and-place machines to locate the components correctly.

CHOOSING YOUR PLATFORM : **Physical Size and Form Factor**

- 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.
- This lets the circuit designer choose the form that best suits his particular application.
- All three chips pictured in the following figure provide identical functionality because they are all AVR ATmega328 microcontrollers.
- The one on the left is the through-hole package, mounted here in a socket so that it can be swapped out without soldering.
- The two others are surface mount, in two different packages, showing the reduction in size but at the expense of ease of soldering.



Prototyping Embedded Devices

Content

- ✓ Electronics
- ✓ Sensors
- ✓ Actuators
- ✓ Scaling Up the Electronics
- ✓ Embedded Computing Basics
- ✓ Microcontrollers
- ✓ System-on-Chips
- ✓ Choosing Your Platform
 - Arduino, Developing on the Arduino, Some Notes on the Hardware, Openness.
 - Raspberry Pi, Cases and Extension Boards, Developing on the Raspberry Pi, Some Notes on the Hardware, Openness.

ARDUINO

- Without a doubt, the poster child for the Internet of Things, and physical computing in general, is the Arduino.
- These days the Arduino project covers a number of microcontroller boards, but its birth was in Ivrea in Northern Italy in 2005.
- A group from the Interaction Design Institute Ivrea (IDII) wanted a board for its design students to use to build interactive projects. An assortment of boards was around at that time, but they tended to be expensive, hard to use, or both.



Fig: An Arduino Ethernet board, plugged in, wired up to a circuit and ready for use.

DEVELOPING ON THE ARDUINO

Integrated Development Environment: 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. 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.

It is, however, possible to upload an OS to the Arduino, usually a lightweight real-time operating system (RTOS) such as FreeRTOS/DuinOS. The main advantage of one of these operating systems is their built-in support for multitasking. However, for many purposes, you can achieve reasonable results with a simpler task-dispatching library.

DEVELOPING ON THE ARDUINO

Language :The language usually used for Arduino is a slightly modified dialect of C++ derived from the Wiring platform. 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. Typically, you might check some input, do some calculation on it, and perhaps do some output in response.

DEVELOPING ON THE ARDUINO

Blinking a single LED

// Pin 13 has an LED connected on most Arduino boards.

// give it a name:

```
int led = 13;
```

// the setup routine runs once when you press reset:

```
void setup() {
```

```
// initialize the digital pin as an output.
```

```
pinMode(led, OUTPUT);
```

```
}
```

// the loop routine runs over and over again forever:

```
void loop() {
```

```
digitalWrite(led, HIGH); // turn the LED on
```

```
delay(1000); // wait for a second
```

```
digitalWrite(led, LOW); // turn the LED off
```

DEVELOPING ON THE ARDUINO

Debugging :Because C++ is a compiled language, a fair number of errors, such as bad syntax or failure to declare variables, are caught at compilation time. Because this happens on your computer, you have ample opportunity to get detailed and possibly helpful information from the compiler about what the problem is.

Example:

If **Bubblino** stops blowing bubbles, how can we distinguish between the following cases?

- Nobody has mentioned us on Twitter.
- The Twitter search API has stopped working.
- Bubblino can't connect to the Internet.
- Bubblino has crashed due to a programming error.
- Bubblino is working, but the motor of the bubble machine has failed.
- Bubblino is powered off.



Prototyping Embedded Devices

Content

- ✓ Electronics
- ✓ Sensors
- ✓ Actuators
- ✓ Scaling Up the Electronics
- ✓ Embedded Computing Basics
- ✓ Microcontrollers
- ✓ System-on-Chips
- ✓ Choosing Your Platform
- ✓ Arduino, Developing on the Arduino, Some Notes on the Hardware, Openness
- ✓ Raspberry Pi, Cases and Extension Boards, Developing on the Raspberry Pi, Some Notes on the Hardware, Openness.

Raspberry Pi

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




DEVELOPING ON THE RASPBERRY PI



Operating System

Raspbian: Released by the Raspbian Pi Foundation, Raspbian is a distro based on Debian. This is the default “official” distribution and is certainly a good choice for general work with a Pi.

Occidentalis: This is Adafruit’s customised Raspbian. Unlike Raspbian, the distribution assumes that you will use it “headless”—not connected to keyboard and monitor—so you can connect to it remotely by default.



DEVELOPING ON THE RASPBERRY PI

Programming Language

There is some guidance from the Foundation, which suggests Python as a good language for educational programming (and indeed the name “Pi” comes initially from Python).

Python will almost certainly make the following tasks easier:

- * Handling strings of character data
- * Completely avoiding having to handle memory management (and bugs related to it)
- * Making calls to Internet services and parsing the data received
- * Connecting to databases and more complex processing
- * Abstracting common patterns or complex behaviours

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++.

The former is unlikely to be an issue; the Pi has more than enough memory. The speed of execution may may not be a problem

Python is likely to be “fast enough” for most tasks, and certainly for anything that involves talking to the Internet, the time taken to communicate over the network is the major slowdown.

However, if the electronics of the sensors and actuators you are working with require split-second timing Python might be too slow.

This is by no means certain; if Bubblino starts blowing bubbles a millisecond later, or the DoorBot unlocks the office a millisecond after you scan your RFID card to authenticate, this delay may be acceptable and not even noticeable.

Contrast Python with C++




Python handles memory management automatically.

Because handling the precise details of memory allocation is notoriously fiddly, automatic memory management generally results in fewer bugs and performs adequately.

However, this automatic work has to be scheduled in and takes some time to complete.

Depending on the strategy for garbage collection, this may result in pauses in operation which might affect timing of subsequent events.




Contrast Python with C++



Linux itself arguably has some issues for “real-time” use.

Due to its being a relatively large operating system, with many processes that may run simultaneously, precise timings may vary due to how much CPU priority is given to the Python runtime at any given moment.

This hasn't stopped many embedded programmers from moving to Linux, but it may be a consideration for your case.



Contrast Python with C++




An Arduino runs only the one set of instructions, in a tight loop, until it is turned off or crashes.

The Pi constantly runs a number of processes.

If one of these processes misbehaves, or two of them clash over resources, they may cause problems that are entirely unrelated to your code.

This is unlikely but may result in occasional, possibly intermittent, issues which are hard to identify and debug.



SOME NOTES ON THE HARDWARE


- The Raspberry Pi has **8 GPIO pins**
- Unlike those in the Arduino, the pins in the Raspberry Pi **aren't individually labelled**. You can connect individual pins using a female jumper lead onto a breadboard.
- The block of pins provides both **5V and 3.3V outputs**.
- However, the GPIO pins themselves are only **3.3V tolerant**. The Pi doesn't have any over-voltage protection, so you are at **risk of breaking the board** if you supply a **5V input**.
- Use of an external breakout board has a kind of protection.
- Note: **The Raspberry Pi doesn't have any analogue inputs (ADC)**, which means that options to connect it to electronic sensors are limited.
- To get readings from light-sensitive photocells, temperature sensors, potentiometers, and so on, you need to **connect it to an external ADC via the SPI bus**

OPENNESS



Because one of the goals of the Raspberry Pi is to create something “hackable”, it is no surprise that many of the components are indeed highly open: the customised Linux distributions such as “Raspbian” (based on Debian), the ARM VideoCore drivers, and so on.

However, many of the Raspberry Pi core team are Broadcom employees and have been active in creating drivers and the like for it, which are themselves **open source**.





Additional Information

“

BEAGLEBONE BLACK

BEAGLEBONE BLACK: CASES AND
EXTENSION BOARDS

DEVELOPING ON THE BEAGLE BONE

SOME NOTES ON THE HARDWARE

OPENNESS

ELECTRIC IMP

BEAGLEBONE BLACK

The BeagleBone Black is the latest device to come from the BeagleBoard group.

The BeagleBoard team wants to create “powerful, open, and embedded devices” with a goal of contributing to the open source community, including facilitating education in electronics.

However, there is less of an emphasis on creating a general purpose computer for education; these boards are very much designed with the expectation that they will be used for physical computing and experimentation in electronics.



Fig: The latest board from the BeagleBone family: the BeagleBone Black.

BEAGLEBONE BLACK

- The BeagleBone Black is the **smallest and cheapest** of the team's boards, with a form factor comparable to that of the Raspberry Pi.
- The original BeagleBone has **no video or audio outputs built in**, but it does have a far **larger number of GPIO pins**.
- It also **has the crucial ADC pins** for analogue input which the Raspberry Pi lacks. This shows the development team's focus on building something for working with electronics rather than as a general-purpose computer.
- The BeagleBone was released before the Raspberry Pi, and its **price** reflects that.
- The influence of the Pi can be seen in the latest revision of the BeagleBone platform, the BeagleBone Black.
- Although still missing the analogue video and audio connectors, it **adds a micro-HDMI connector** to provide digital outputs for both audio and video.

BEAGLEBONE BLACK VS RASPBERRY PI MODEL B

	BeagleBone Black	Raspberry Pi Model B
CPU Speed	1GHz ARM Cortex-A8	700 MHz ARM11
GPU	SGX530 3D	Broadcom Dual-Core VideoCore IV Media Co-Processor
RAM	512MB	512MB
Storage	2GB embedded MMC, plus uSD card	SD card (4GB +)
OS	Various Linux distributions, Android, other operating systems available	Various Linux distributions, other operating systems available
Connections	65 GPIO pins, of which 8 have PWM 4 UARTs SPI bus I2C bus USB client + host Ethernet Micro-HDMI out 7 analog Inputs (ADC) CAN bus	8 GPIO pins, of which 1 has PWM 1 UART SPI bus with two chip selects I2C bus 2 USB host sockets Ethernet HDMI out Component video and audio out

BEAGLEBONE BLACK

Additional info: (Only for Understanding)

Advanced Reduced Instruction Set Computer machine (ARM) server

Universal Asynchronous Receiver-Transmitter. A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port.

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means

HDMI (High-Definition Multimedia Interface) is a proprietary audio/video interface for transferring uncompressed video data and compressed or uncompressed digital audio data from an HDMI-compliant source device, such as a display controller, to a compatible computer monitor, video projector, digital television

The **I²C(Inter-IC) bus** is a bi-directional two-wire serial bus that provides a communication link between integrated circuits (ICs). Phillips introduced the I²C bus 20 years ago for mass-produced items such as televisions, VCRs, and audio equipment.

The **Serial Peripheral Interface (SPI)** is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems

A **Controller Area Network (CAN bus)** is a robust vehicle bus standard designed to allow microcontrollers and

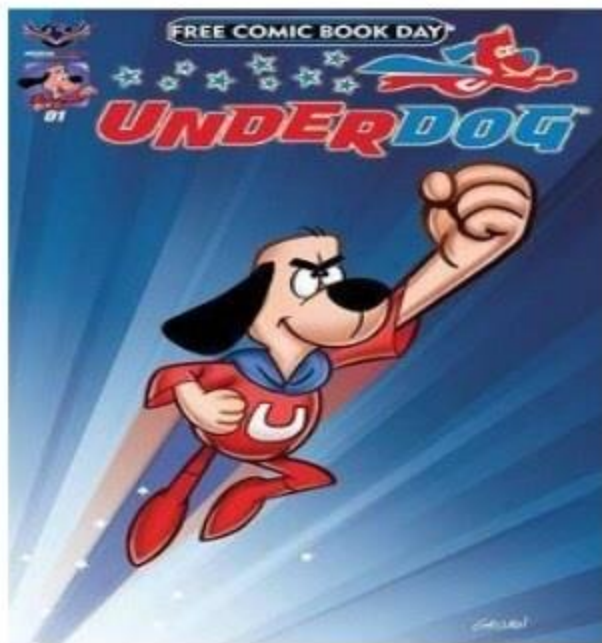
BEAGLEBONE BLACK: CASES AND EXTENSION BOARDS

Although the BeagleBone hasn't had the same hype and media exposure as the Pi, a fair number of attractive cases are available, either sold commercially like the **Adafruit BoneBox**, or as freely available instructions and designs.



BEAGLEBONE BLACK:CASES AND EXTENSION BOARD

Extension boards for the BeagleBone are known as “capex” rather than “shields” (the term for Arduino extension boards). This name comes from **Underdog**, the star of a 1960s US animation, a beagle who wore a superhero cape. Capes are available to add controllers for LCD displays and motors and various networking and sensor applications.



DEVELOPING ON THE BEAGLEBONE

BeagleBone is capable of general-purpose computing, its target market is much more narrowly defined.

Every board comes with the [Angstrom Linux distribution](#) installed on it.

Several other [operating systems](#) have been ported to the platform.

It is far simpler to get started and evaluate the board.

The BeagleBone runs [zeroconf networking](#) (Zero-configuration networking (zeroconf) is a set of technologies that automatically creates a usable computer network based on the Internet Protocol Suite) by default and usually advertises itself as *beaglebone.local*.

The manual is [dynamically updated](#) to give information about your BeagleBone and has pages where you can look at and change various settings, such as the outputs to GPIO pins. It also has links to the [Cloud9 IDE](#), which is hosted on the board, and which is the simplest recommended way to write code for the device.


DEVELOPING ON THE BEAGLE BONE



Cloud9 is an online programming environment

This means that to develop on the BeagleBone, you can get started immediately, without needing to download any software at all, by connecting to the IDE.

Although the online Cloud9 environment also supports Ruby and Python, the free component version supports only Node.js, a framework built on JavaScript.



DEVELOPING ON THE BEAGLE BONE: Operating System

The BeagleBone comes pre-installed with the Ångström Linux distribution.

This operating system was developed specifically for running on embedded devices and can run with as little as 4MB of flash storage.

There are convenient ways to access a command-line prompt, either over a serial connection with USB or using zeroconf and connecting to `beaglebone.local`.

One advantage of having a single pre-installed operating system is that you can start playing with the board the moment you open the box.

Of course, the onboard instructions suggest updating the operating system as one of the first steps, if it's not already at the latest version; this is certainly good practice after you have briefly evaluated the board and are moving onto prototyping a project with it.

DEVELOPING ON THE BEAGLE BONE :

Programming Language

As with the Raspberry Pi, you can connect to BeagleBone via the terminal and develop a software application as you would for any other Linux computer.

As always, the code is practically the same in concept: the details of the syntax vary, but you still have the usual two functions, `setup` and `loop`; have to set the mode of the pin to output; and alternate sending HIGH and LOW values to it with a pause in between.

As with Python, JavaScript is a high-level language, so after you move beyond the physical input and output with electronic components, tasks dealing with text manipulation, talking to Internet services, and so on are far easier to handle than the relatively low-level C++ used in the Arduino.

Node.js is a rich environment with a host of libraries available to integrate into the app.

Currently, the convenient `npm` (Node Packaged Modules) utility isn't bundled with the IDE, but this is an item for a future version. In the meantime, online help and forums should get you over any possible stumbling blocks.

DEVELOPING ON THE BEAGLE BONE :

Debugging

Just like Python, JavaScript is a permissive language compared to C++.

Again, because the language is dynamic, the compiler does not catch several types of programming errors.

Luckily, the IDE reports runtime errors quite usefully, so while you are running from that, you can get reasonable diagnostics with minimal effort.

When you move to running the Node.js code automatically as a service, you will probably want to enable logging to catch those errors that don't become evident until days or months later.

Node.js is a more complicated platform than Arduino, but the flexibility and power are certainly appreciated.

DEVELOPING ON THE BEAGLE BONE :

Debugging (contd...)

Again, JavaScript has **error handling**, although it is extended and arguably complicated by Node.js's call-back mechanisms.

This feature can **help handle and recover from unusual situations gracefully**.

Node.js also has automated testing tools such as **node-unit** .

Just because you are writing in a **high-level language** doesn't mean that all your code will be simpler.

The fact that it is easier to write any given line often means that you write much more code and of greater complexity.

Because Node.js is built on collaborative multitasking through call-backs, the execution path may be especially complicated: **testing can allow you to make sure that all the expected paths through this code happen, in the right order**.

SOME NOTES ON THE HARDWARE

For physical computing, the number of GPIO pins available is greatly superior to the Pi's, and a number of them have ADCs (allowing them to process analogue inputs, a capability which the Pi lacked).

Although the pins are neatly laid along both long edges of the board, in two columns, the BeagleBone doesn't have enough space to label each pin, as the Arduino does.

Instead, each set of headers has labels, and the documentation has a key for what each of the physical pins does.

As with the Pi, each pin may have a number of different names, depending on which subsystem is accessing it, and this may feel unnecessarily complex if you are used to the simple, restricted naming (and labelling) conventions of the Arduino.

OPENNESS



One of the initial explicit goals of the BeagleBoard was to create an open platform for experimenting with embedded hardware.

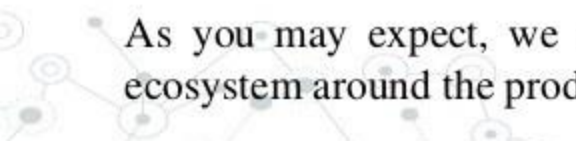
As such, the project has, to date, released more of its schematics as open source than has the Raspberry Pi Foundation.

Specifically, “All hardware from BeagleBoard.org is open source, so you can download the design materials including schematics, bill-of-materials and PCB layout” (<http://beagleboard.org/>).

The Ångström Linux distribution that is installed by default on the BeagleBone is also fully open source. Most of the nontechnical content of the site is also released under a Creative Commons licence.

The BeagleBone web pages also link to proprietary operating systems and extension boards.

As you may expect, we can consider this to be a good thing, as the team are encouraging a good ecosystem around the product, both free and commercial.



ELECTRIC IMP

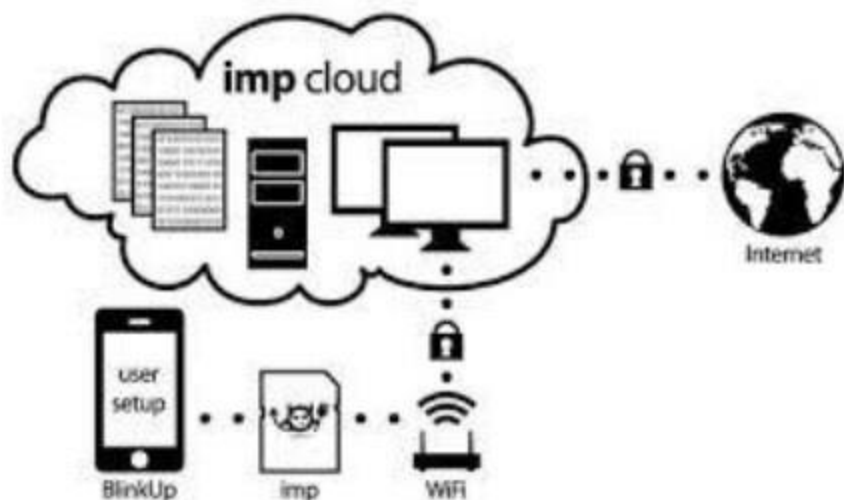
The rear of (from left to right): an Electric Imp, a micro SD card, an SD card.



ELECTRIC IMP

Electric Imp, an all-in-one IoT Connectivity Platform makes it simple to connect devices to the Internet.

What is the electric imp? In essence, the Imp provides an easy, integrated way to connect almost any hardware device both to other devices and to internet services. It's more than just a WiFi card, or even a WiFi module with processing built in - it's an integrated platform that deals with the drudgery of connectivity, allowing you to concentrate on the application instead of the mechanics.



ELECTRIC IMP

Fiennes collaborated on the project with Kevin Fox, a former Gmail designer, and firmware engineer Peter Hartley.



ELECTRIC IMP

Electric Imp uses a system in which the network name is selected on a smartphone, then a password typed in. An app then sends the digital information optically—through a series of flashes of the screen—to the card edge as it sits in the socket.

Electric Imp cards have a familiar memory card physical format but contain a Wi-Fi node inside
It's important to note that the Imp isn't actually an SD card; it's just shaped like one



ELECTRIC IMP

An Electric Imp enabled wall power socket. With card installed it can be part of an Internet of Things.



ELECTRIC IMP

Although an SD card feels very robust, it is, on the outside, effectively a small, flat piece of plastic.

It offers only **one affordance to connect it to anything: namely, plug it into a device.**

Just as you would insert an SD card into a music player, computer, or printer, you insert an Imp into an impee.

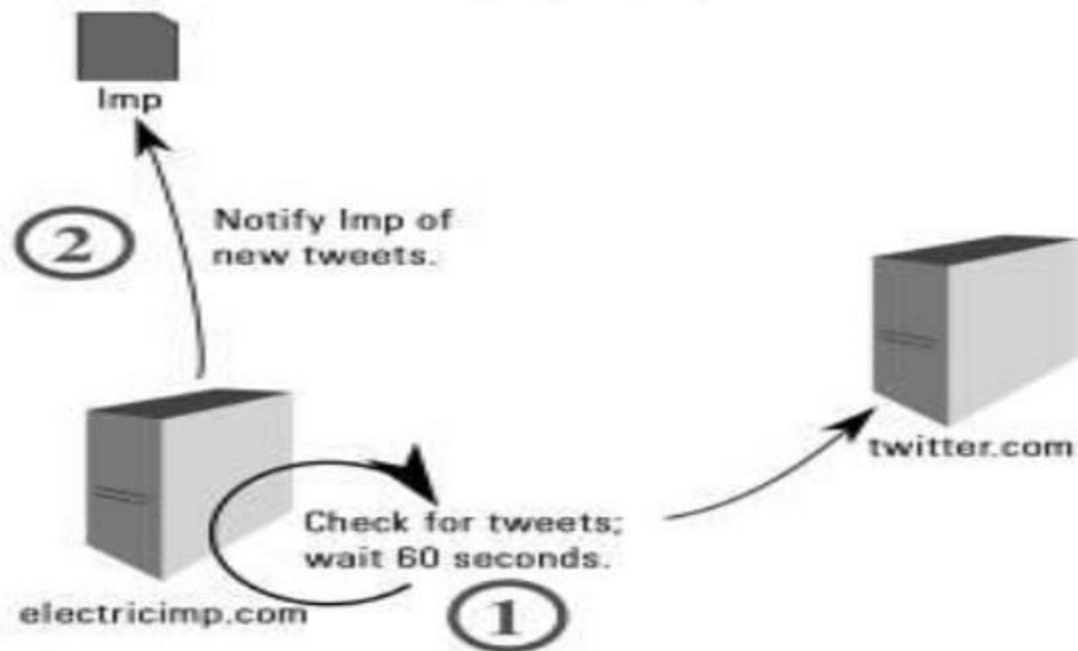
This host board provides power, GPIO connections to sensors and actuators, and an ID chip so that the Imp knows which device it's plugged into



ELECTRIC IMP

The language used is Squirrel, which is a dynamic language with a C-like syntax. At present, Squirrel is poorly documented and poorly optimised.

It is comparable to high-level embeddable scripting languages such as Lua



Electric Imp Case Study: Lockitron

Although the **Raspberry Pi DoorBot** solves a very specific problem for a fairly technical group of people, everyone uses doors and keys.

Lockitron ([https:// lockitron.com/](https://lockitron.com/)), from Apigy, is a consumer solution to the problem of granting visitors access to your home without getting more keys cut.

As such, it doesn't require changing locks and drilling holes through walls.

You then open the lock using an app on your mobile phone, rather than using RFID (although the Lockitron does also have optional support for NFC radio).

This means that the device has to be constantly connected to the Internet: because many people do not have Ethernet cabling running to their front door, this makes wireless connectivity a requirement.



A decorative network diagram in the top right corner of the slide. It consists of several interconnected nodes, represented by circles of varying sizes and shades of gray, connected by thin lines. The nodes are arranged in a somewhat circular pattern, with some nodes having multiple connections to other nodes.

Reference

McEwen, Adrian, and Hakim Cassimally. Designing the internet of things. John Wiley & Sons, 2013.

A decorative network diagram in the bottom left corner of the slide. It consists of several interconnected nodes, represented by circles of varying sizes and shades of gray, connected by thin lines. The nodes are arranged in a somewhat circular pattern, with some nodes having multiple connections to other nodes.