SYLLABUS

Unit	Details
II	**Thinking About Prototyping:** Sketching, Familiarity, Costs versus Ease of Prototyping, Prototypes and Production, Changing Embedded Platform, Physical Prototypes and Mass Personalisation, climbing into the Cloud, Open Source versus Closed Source, Why Closed? Why Open? Mixing Open and Closed Source, Closed Source for Mass Market Projects, Tapping into the Community. **Prototyping Embedded Devices:** 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.

Books and References

Sr. No.	Title	Author/s	Publisher	Edition	Year
1.	Designing the Internet of Things	Adrian McEwen, Hakim Cassimally	WILEY	First	2014
2.	Internet of Things Architecture and Design	Raj Kamal	McGraw Hill	First	2017
3.	Getting Started with the Internet of Things	Cuno Pfister	O'Reilly	Sixth	2018
4.	Getting Started with Raspberry Pi	Matt Richardson and Shawn Wallace	SPD	Third	2016

Chapter 4

Thinking about Prototyping

Prototyping is the most important process of product development. The core of prototyping is to test the product and usually used for demonstration, testing, communication and so on.

Benefits of prototyping:
One can come across problems in design that need to change and iterate. Doing this with a single object is trivial compared to modifying hundreds or thousands of products. Prototyping is optimized for ease and speed of development and also the ability to change and modify it.

With the Internet of Things, we need to build three things in parallel:

i. Physical Thing
ii. Electronics to make the Thing smart
iii. Internet service that we'll connect to.

At the end of this stage, you'll have an object that works. And if you are planning to move to production, it's a *demonstrable* product that you can use to convince yourself, your business partners, and your investors. Finally, the process of manufacture will iron out issues of scaling up and polish. The final product will be cheaper per unit and more professional but will be much more expensive to change. For one thing, your project has its own goals and requirements. For another, new microcontrollers are constantly coming onto the market; best practice and popularity of server software stacks are ever changing; and rapid prototyping continues to evolve.

4.1 SKETCHING

When you are working on your prototype, the first step you will be doing is to jot down some ideas or draw out some design ideas with pen and paper. Then extend it to also include sketching in hardware and software. With a sketch, the designer is able to communicate a potential solution that could be built on later to form a refined design.

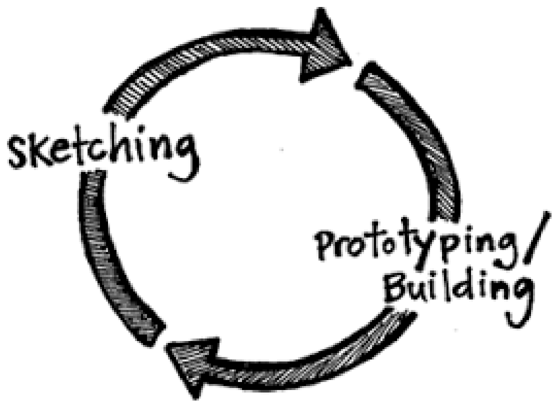

Figure 4.1: Sketching

Sketching enables you to brainstorm, explore multiple ideas, define flows, communicate with team members all why being quick and cheap

What we mean by that is the process of exploring the problem space:
iterating through different approaches and ideas to work out what works and what doesn't. The focus isn't on fidelity of the prototype but rather on the ease and speed with which you can try things out.

4.2 FAMILIARITY

Another option to consider is familiarity. If you can already program like a whiz in Python, for example, maybe picking a platform such as Raspberry Pi, which lets you write the code in a language you already know, would be better than having to learn Arduino from scratch. The same applies to the server software, obviously.

4.3 COSTS VERSUS EASE OF PROTOTYPING

It is worth considering the relationship between the costs (of prototyping and mass producing) of a platform against the development effort that the platform demands. This trade-off is not hard and fast, but it is beneficial if you can choose a prototyping platform based on performance/capabilities. For the first prototype, the cost is probably not the most important issue: the smartphone or computer options are particularly convenient if you already have one available, at which point they are effectively zero-cost. Of course, if your device has physical interactions (blowing bubbles, turning a clock's hands, taking input from a dial), you will find that a PC is not optimize for this kind of work. Electronics prototyping board, *is* better suited to this kind of work. An important factor to be aware of is that the hardware and programming choices you make will depend on your skill set.
For many beginners to hardware development, the Arduino toolkit is a surprisingly good choice. Yes, the input/output choices are basic and require an ability to follow wiring diagrams and, ideally, a basic knowledge of electronics. If you already know how to develop an application in C#, in Python, or in JavaScript, you have a great place to start. But if you don't know, you first have to evaluate and choose a language and then work out how to write it, get it going, and make it start automatically.
Another option is to marry the capabilities of a microcontroller to connect to low-level components such as dials, LEDs, and motors while running the hard processing on a computer or phone. A kit such as an Arduino easily connects to a computer via USB, and you can speak to it via the serial port in a standard way in any programming language.

4.4 PROTOTYPES AND PRODUCTION

4.4.1 CHANGING EMBEDDED PLATFORM

When you scale up, you may well have to think about moving to a different platform, for cost or size reasons. If you've started with a free-form, powerful programming platform, you may find that porting the code to a more restricted, cheaper, and smaller device will bring many challenges.
Of course, if you've used a constrained platform in prototyping, you may find that you have to make choices and limitations in your code. Dynamic memory allocation on the 2K that the Arduino provides may not be especially efficient, so how should that make you think about using strings or complex data structures? If you port to a more powerful platform, you may be able to rewrite your code in a more modern, high-level way or simply take advantage of faster processor speed and more RAM. But will the new platform have the same I/O capabilities? And you have to consider the ramping-up time to learn new technologies and languages. In practice, you will often find that you don't need to change platforms.

4.4.2 PHYSICAL PROTOTYPES AND MASS PERSONALISATION

Chances are that the production techniques that you use for the physical side of your device won't translate directly to mass production. An aspect that may be of interest is in the way that digital fabrication tools can allow each item to be slightly different, letting you personalise each device in some way. There are challenges in scaling this to production, as you will need to keep producing the changeable parts in quantities of one, but *mass personalisation*, as the approach is called, means you can offer something unique with the accompanying potential to charge a premium.

4.4.3 CLIMBING INTO THE CLOUD

The server software is the easiest component to take from prototype into production. As we saw earlier, it might involve switching from a basic web framework to something more involved but you will be able to find an equivalent for whichever language you have chosen. That means most of the business logic will move across with minimal changes. Beyond that, scaling up in the early days will involve buying a more powerful server. If you are running on a cloud computing platform, such as Amazon Web Services, you can even have the service dynamically expand and contract, as demand dictates.

4.5 OPEN SOURCE VERSUS CLOSED SOURCE

Open Source	Closed Source
Provides the original source code so that advanced user can modify it to make it work better	No one can duplicate or distribute without permission of the company
Open Source software available to all free of cost	Closed source software is protected by copyrights
Any individual can develop or modify	Development is centralised, undertaken by expert developers
Quality assurance or bug removal is done by small team	Large number of individuals are available for improving quality and bug removal
Android	Apple

4.5.1 WHY CLOSED?

- If you declared copyright on some source code or a design, someone who wants to market the same project cannot do so by simply reading your instructions and following them
- You might also be able to protect distinctive elements of the visual design with trademarks and of the software and hardware with patents
- Starting a project as closed source doesn't prevent you from later releasing it as open source
- Closed-source software is built with users in mind.
- Closed-source software offers ongoing technical support for its users.
- Closed-source software is usually 1 or 2 steps ahead of open-source software

Figure 4.2: Closed Source Products

4.5.2 WHY OPEN?

- You may gain positive comments from people who liked it.
- It acts as a public showcase of your work, which may affect your reputation and lead to new opportunities.
- People who used your work may suggest or implement features or fix bugs.
- By generating early interest in your project, you may get support and mindshare of a quality that it would be hard to pay for.
- *economy*: you can use other people's free and open source contributions within your own project.
- Faster time to market. Because open source solutions are openly available and can be explored for free, it's often much faster to investigate options and get solutions off the ground.
- Open source solutions should be thought of as more than just free software, the fact that they require no licensing fees remains a decisive advantage when looking at the total cost of deploying a solution

Figure 4.3: Open Source Products

4.5.3 Disadvantages of Open Source

- Some open source applications may be tricky to set up and use. Others may lack user-friendly interfaces or features that your staff may be familiar with.
- There may also be less support available for when things go wrong – open source software tends to rely on its community of users to respond to and fix problems.
- “but people will steal my idea!” In general, if you talk to people about an idea, it’s hard enough to get them to listen because they are waiting to tell you about *their* great idea.

4.5.4 Open Source as a Competitive Advantage

- Using open source work is often a no-risk way of getting software that has been tested, improved, and debugged by many eyes
- Commercial equivalents may be available for all these examples, but then you have to factor in the cost and rely on a single company’s support forums instead of all the information available on the Internet.
- Using open source aggressively gives your product the chance to gain mindshare

4.5.5 MIXING OPEN AND CLOSED SOURCE

- Software companies are taking a "best of both worlds" approach by creating products that use a combination of OS and proprietary software code.
- Open sourcing many of libraries and keeping core business closed

4.5.6 CLOSED SOURCE FOR MASS MARKET PROJECTS

One edge case for preferring closed source when choosing a licence may be when you can realistically expect that a project might be not just successful but *huge*, that is, a mass market commodity.

4.6 Review Questions

1. Explain benefits of prototyping.
2. Differentiate between open source and closed source
3. What are the advantages of open source softwares?
4. Why should you mix open and closed source for the Internet of Things?

Chapter 5
Prototyping Embedded Devices

To create an IoT product, we should study electronics because whatever platform we choose, the rest of the circuitry that we will build to connect it to will be pretty much the same.

5.1 ELECTRONICS

Most of the prototyping can be done on what are called *solderless breadboards*. They enable you to build components together into a circuit with just a push-fit connection. When it comes to thinking about the electronics, it's useful to split them into two main categories:

- **Sensors:** A sensor is a device that detects and responds to some type of input from the physical environment. The specific input could be light, heat, motion, moisture, pressure etc. Pushbuttons and switches, which are probably the simplest sensors, allow some user input. Potentiometers (both rotary and linear) and rotaryencoders 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 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.

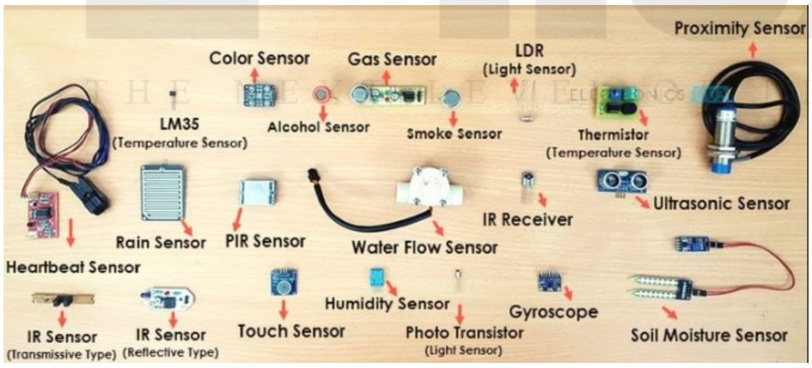

Figure 5.1: Sensors

- **Actuators:** An **actuator** is a machine or part of a machine which moves or controls another part in response to an input—the motors, lights, and so on. One of the simplest and yet most useful actuators is light, because it is easy to create electronically and gives an obvious output. 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 synthesised sounds. 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. 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, often 180 or fewer degrees (whereas steppers and DC motors turn indefinitely). For all the kinds of motors that we've mentioned, you typically want to connect the motors to gears to alter the range of motion or convert circular movement to linear, and so on.

- **Analog to Digital converter:** It is a device that converts analog signals (usually voltage) obtained from environmental (physical) phenomena into digital format. It can be expressed as A/D or A-to-D or A-D or ADC.

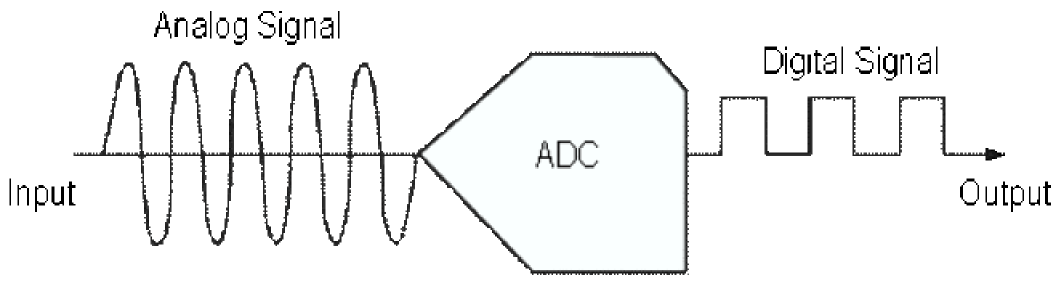

Figure 5.2: Analog to Digital

- **Digital to analog convertor:** It is an electronics device in form of IC, which converts digital signal to its equivalent analog signal. It can be expressed as DAC, D/A, D2A, or D-to-A

- **Serial Peripheral Interface (SPI):** It is a synchronous serial communication interface specification used for short distance communication. SPI devices communicate in full duplex mode using a master-slave architecture with a single master. The master device originates the frame for reading and writing. Multiple slave devices are supported through selection with individual slave select (SS) lines.

- **I²C (Inter-Integrated Circuit)** : pronounced *I-squared-C*, is a synchronous, multi-master, multi-slave bus. It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

5.2 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. Moving beyond the protoboard option tends to involve learning how to lay out a PCB. When you want to scale things even further, moving to a combined board allows you to remove any unnecessary components from the microcontroller board, and switching to surface mount components—where the legs of the chips are soldered onto the same surface as the chip—eases the board's assembly with automated manufacturing lines.

5.3 EMBEDDED COMPUTING BASICS

5.3.1 MICROCONTROLLERS

A microcontroller is a single chip microcomputer made through VLSI fabrication. A microcontroller also called an embedded controller because the microcontroller and its support circuits are often built into, or embedded in, the devices they control. A microcontroller is available in different word lengths like microprocessors (4bit,8bit,16bit,32bit,64bit and 128-bit microcontrollers are available today).

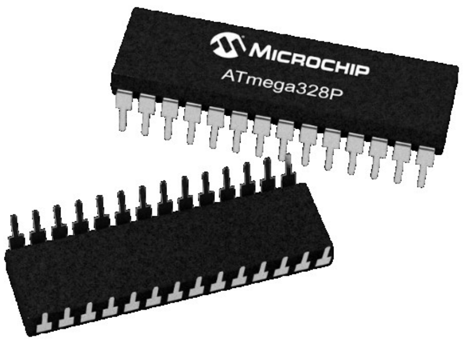

Figure 5.3: Microcontrollers

5.3.2 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 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. 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.

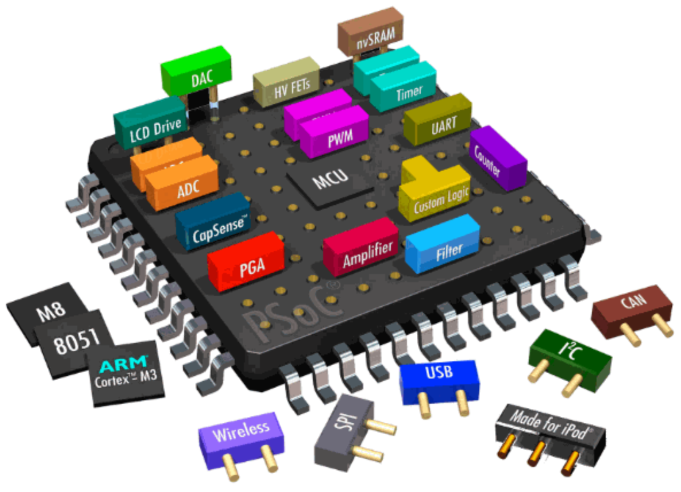

Figure 5.4: System-on-Chip

5.4 CHOOSING YOUR PLATFORM

Choosing the *right* platform for Internet of Things device is based on following parameters.

- Processor Speed :

The processor speed decides how fast it can process the individual instructions. A faster processor speed means that it can execute instructions more quickly. Microcontrollers tend to be clocked at speeds in the tens of MHz, whereas SoCs run at hundreds of MHz or possibly low GHz.

- RAM

The overall performance of system is determined by how well processor and RAM work together. RAM provides the working memory for the system. More RAM means more flexibility. The amount of RAM needed is vary from project to project.

- Networking

Wired Ethernet is often the simplest for the user—generally plug and play—and cheapest, but it requires a physical cable. Wireless solutions required 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. ZigBee is one 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. SMS can be used for low-bandwidth, higher-latency communication; for higher data rates, you will use the same data connections, like 3G, as a smartphone.

- USB

USB can provide both power and networking. Some of the microcontrollers have inbuilt 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—devices such as phones, WiFi dongles. Devices such as WiFi dongles often depend on additional software on the host system, such as networking stacks.

- Power Consumption

Faster processors, more power consumption. For portable devices 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. In this mode, power consumption is less. Therefore, a more powerful processor may *not* be a disadvantage even in a low-power embedded device.

- Interfacing with Sensors and other Circuitry

Devices needs to interact with—either sensors to gather data about its environment; or motors, LEDs, screens, and so on, to provide output. These devices could connect to the circuitry through some sort of peripheral bus—SPI and I2C being common ones. The circuitry may involve ADC or DAC modules to read or write varying voltages. 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 with surrounding components on the PCB. With surface-mount technology, the size is reduced to the great extent because it doesn't require holes to be drilled in the board for connections. 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 them correctly. 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.

5.5 ARDUINO

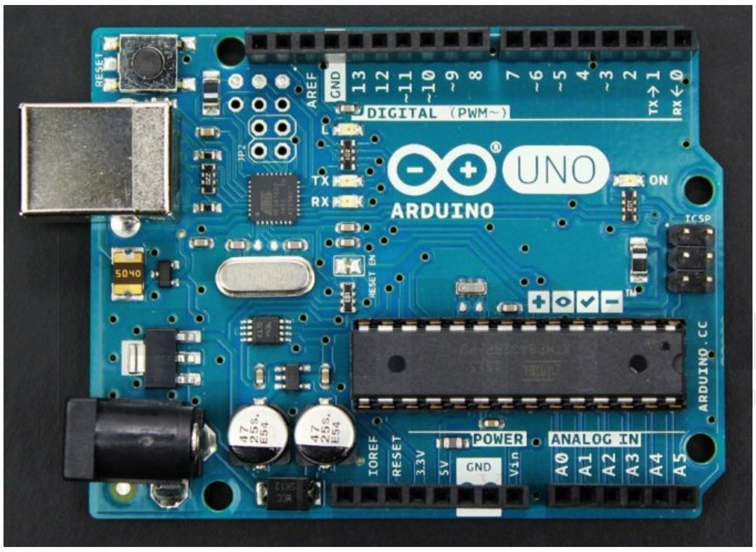

Figure 5.5 Arduino UNO

Video URL https://www.youtube.com/watch?v=_h1m6R9YW8c
Video 5.1: Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. 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. So, the team put together a board which was cheap to buy—around L20—and included an onboard serial connection to allow it to be easily programmed. Combined with an extension of the Wiring software environment, it made a huge impact on the world of physical computing.
Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Features of different Arduino boards are as follows:

Arduino Board	Processor	Memory	Digital I/O	Analogue I/O
Arduino Uno	16Mhz ATmega328	2KB SRAM, 32KB flash	14	6 input, 0 output
Arduino Due	84MHz AT91SAM3X8E	96KB SRAM, 512KB flash	54	12 input, 2 output
Arduino Mega	16MHz ATmega2560	8KB SRAM, 256KB flash	54	16 input, 0 output
Arduino Leonardo	16MHz ATmega32u4	2.5KB SRAM, 32KB flash	20	12 input, 0 output

5.5.1 DEVELOPING ON 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. Although this is a fully functional IDE, based on the one used for the Processing language (http://processing.org/), it is very simple to use. 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. After this, you need to choose the correct serial port and the board type. When your setup is correct, the process of pushing code is generally simple:

- Code is checked and compiled, with any compilation errors reported to you.
- If code compiles successfully, it gets transferred to the Arduino and stored in its flash memory.
- Arduino reboots and starts running the new code.

Operating System

In Arduino, the bootloader, which simplifies the code-pushing process described previously. When the board switched on, it simply runs the code until the board is switched off again (or the code crashes). 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.

Language

The language for Arduino is a merely set 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". This variant of C++ tries to be forgiving about the ordering of code; for example, it allows to call functions before they are defined. This alteration is just a nicety, but it is useful to be able to order things in a way that the code is easy to read and maintain, given that it tends to be written in a single file.
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.

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.

SOME NOTES ON THE HARDWARE

- Number of GPIO pins and is usually supplied with "headers" (plastic strips that sit on the pin holes). Each pin is clearly labelled on the controller board.
- Power outputs such as 5 volts or 3.3 volts (usually labelled 5V and 3V3, or perhaps just 3V),
- One or more electric ground connections (GND), numbered digital pins, and numbered analog pins prefixed with an *A*.
- Arduino can be powered using a USB connection from your computer. The Arduino also has a socket for an external power supply, which you might be more likely to use if you distribute the project.
- Outside of the standard boards, a number of them are focused on a particular niche application—for example, the Arduino Ethernet has an on-board Ethernet chip and trades the USB socket for an Ethernet one, making it easier to hook up to the Internet. This is obviously a strong contender for a useful board for Internet of Things projects.
- An additional circuit board on top of the Arduino which can contain all manner of componentry to give the Arduino extra capabilities. In the Arduino world, these add-on boards are called *shields*, perhaps because they cover the actual board as if protecting it. Some shields provide networking capabilities—Ethernet, WiFi, or Zigbee wireless, for example.

OPENNESS

The Arduino project is completely open hardware. The only part of the project protected is the Arduino trademark, so they can control the quality of any boards calling themselves an

Arduino. In addition to the code being available to download freely, the circuit board schematics and even the EAGLE PCB design files are easily found on the Arduino website. This culture of sharing has borne fruit in many derivative boards being produced by all manner of people. Some are merely minor variations on the main Arduino Uno, but many others introduce new features or form factors that the core Arduino team have overlooked.

5.6 RASPBERRY PI

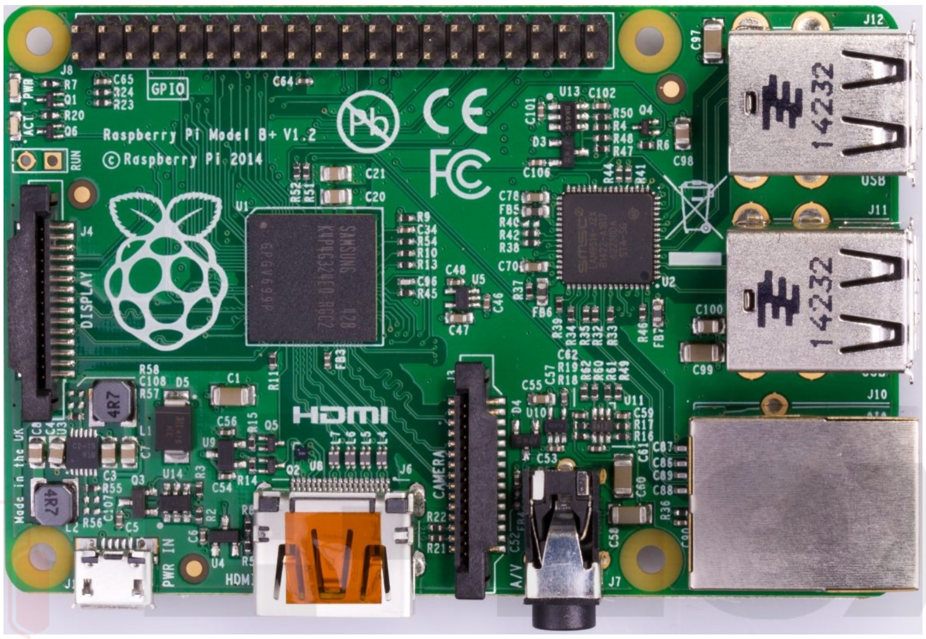

Figure 5.6: Raspberry Pi Model 3B

Video URL: https://www.youtube.com/watch?v=5jA8wYqQLBU
Video 5.2: Raspberry Pi

The **Raspberry Pi** is a tiny and affordable computer that you can use to learn programming through fun, practical projects. Unlike the Arduino, it wasn't designed for physical computing at all, but rather, for education. The vision of Eben Upton, trustee and cofounder of the Raspberry Pi Foundation, was to build a computer that was small and inexpensive and designed to be programmed and experimented with.
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
GPIO	54 GPIO pins	8 GPIO pins
PWM Output	12	1
UART	4	1
Audio/Video output	---	HDMI out Component video and audio out
Connections	SPI bus I^{2}C bus USB 16U2 + native host 12 analogue inputs (ADC) 2 analogue outputs (DAC)	SPI bus with two chip selects I^{2}C bus 2 USB host sockets Ethernet

So, 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 Arduino has a fraction of the raw processing power, memory, and storage required for it to run a modern OS. Importantly, the Pi Model B has built-in Ethernet and can also use cheap and convenient USB WiFi dongles, rather than having to use an extension "shield". Note that although the specifications of the Pi are in general more than capable even the top-of-the-range Arduino Due,

5.6.1 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, 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.

For Internet of Things work, we recommend something such as the Adafruit distro. You're most probably not going to be running the device with a keyboard and display, so you can avoid the inconvenience of sourcing and setting those up in the first place. The main tweaks that interest us are that

- The sshd (SSH protocol daemon) is enabled by default, so you can connect to the console remotely.
- The device registers itself using zero-configuration networking (zeroconf) with the name raspberrypi.local, so you don't need to know or guess which IP address it picks up from the network in order to make a connection.

Programming Language

One choice to be made is which programming language and environment you want to use. There is some guidance from the Foundation, which suggests Python as a good language for educational programming.
We specifically contrast Python with C++, as the low-level language used for Arduino programming.

- **Python, as with most high-level languages, compiles to relatively large (in terms of memory usage) and slow code, compared to C++.**

The Pi has more than enough memory. The speed of execution may or 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.

- **Python handles memory management automatically.**

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.

- **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.

- **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 (memory, CPU, access to a file or to a network port), 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.

Debugging

While Python's compiler also catches a number of syntax errors and attempts to use undeclared variables, it is also a relatively permissive language (compared to C++) which performs a greater number of calculations at runtime. This means that additional classes of programming errors won't cause failure at compilation but will crash the program when it's running, perhaps days or months later. Python code on Linux gives you the advantages of both the language and the OS.

5.6.2 SOME NOTES ON THE HARDWARE

- The Raspberry Pi has 8 GPIO pins, which are exposed along with power and other interfaces in a 2-by-13 block of male header pins.
- A cable (IDC or similar) can be connected onto the whole block, which leads to a "breakout board" where you do actual work with the GPIO. Alternatively, 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! The alternatives are to either proceed with caution or to use an external breakout board that has this kind of protection
- 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.
- It is powered by a standard USB cable, the voltage transmitted over USB from a laptop computer, a powered USB hub, or a USB charger varies greatly. If you're not able to power or to boot your Pi, check the power requirements and try another power source.

5.6.3 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. The core Broadcom chip itself is a proprietary piece of hardware, and they have released only a partial datasheet for the BCM2835 chipset. 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.

5.7 Review Questions:

5. Differentiate between Arduino Due and Raspberry Pi Model B.
6. Which parameter are important for changing embedded platform ?
7. What are the things need to consider while developing arduino project?
8. Compare between Python and C++ as development language for Internet of Things.
9. Write short note on microcontrollers and system-on-chips.
10. Explain hardware of Raspberry pi.