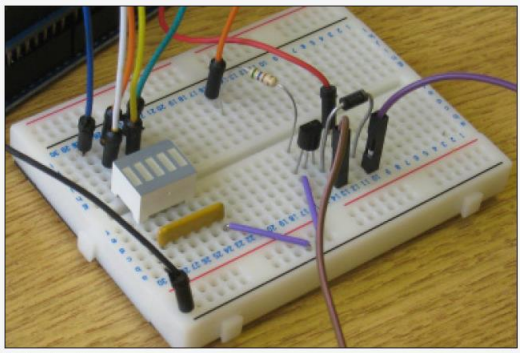
IOT
UNIT 2
Chapter 5
PROTOTYPING EMBEDDED DEVICES

- **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:** 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.
- 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.
- Microcontrollers often have a number of these converters built in.
- 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.

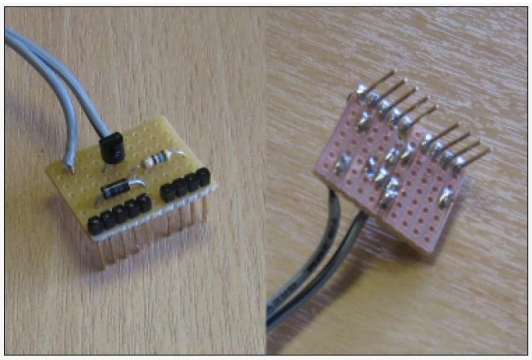
- More complicated visual outputs also are available, such as LCD screens to display text or even simple graphics.
- You can wire up outputs to speakers to create more complicated synthesised sounds.
- More complicated again are motors.
- Stepper motors can be moved in *steps*, as the name implies.
- 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.

SCALING UP THE ELECTRONICS

- From the perspective of the electronics, the starting point for prototyping is usually a "breadboard".
- it's common to solder the components onto some protoboard, which may be sufficient to make the circuit more permanent.
- Moving beyond the protoboard (stripboard) option tends to involve learning how to layout a PCB.
- It involves learning how to use a new piece of software and understanding some new terminology.



The breadboard.



The stripboard.

The PCB.

EMBEDDED COMPUTING BASICS

- **MICROCONTROLLERS:**
- 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.
- 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.
- The microcontroller market consists of many manufacturers each with a range of chips for different applications.
- (Atmel, Microchip, NXP, Texas Instruments, to name a few).
- The devices using them are focused on performing one task.
- **SYSTEM-ON-CHIPS:**
- In between the low-end microcontroller and a full-blown PC sits the SoC.
- 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 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:

- 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.
- Start by choosing a platform to prototype in.
- some of the factors that you need to weigh when deciding how to build your device:
-
- **1) 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).
- 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.
-
- **2) 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.
-
- **3) 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.
- Other short-range wireless can offer better power-consumption profiles or costs than WiFi but usually with the trade-off of lower bandwidth
- There is, of course, the existing Bluetooth standard as another possible choice.
- For remote or outdoor deployment the mobile phone networks can be used.
-
- **4)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".
-
- **5)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.
-

- **6)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.
-
- **7) Physical Size and Form Factor**
- The continual improvement in manufacturing techniques for silicon chips means that we've long passed the point where the limiting factor in the size of a chip is the amount of space required for all the transistors and other components that make up the circuitry on the silicon.
- 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 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.

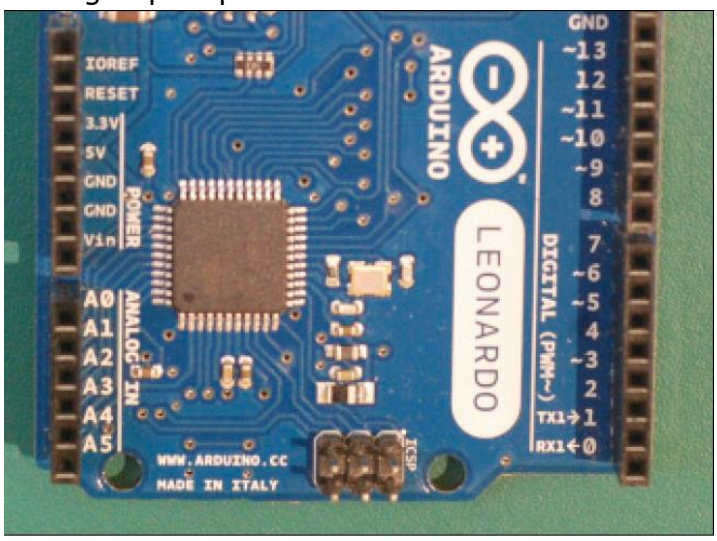
ARDUINO

- The Arduino team's focus on simplicity rather than raw performance for the code has made the Arduino the board of choice in almost every beginner's physical computing project.
- The "standard" Arduino board has gone through a number of iterations:
- Arduino NG, Diecimila, Duemilanove, and Uno.
- 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 (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.
- 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 used by the rest of the Arduino boards and flash the resultant executable to the chip.
- It is used by the Arduino IDE behind the scenes but can be used directly, as well.
-
- **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.
- Typically, you might check some input, do some calculation on it, and perhaps do some output in response.

- Ex: 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
- delay(1000); // wait for a second
- }
-
- **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 the Arduino isn't generally connected to a screen, it is hard for it to tell you when something goes wrong.
- Even if the code compiled successfully, certain errors still happen.
- An error could be raised that can't be handled, such as a division by zero, or trying to access the tenth element of a 9-element list.
- 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.
- The first commercially available version of the WhereDial has a bank of half a dozen LEDs specifically for consumer-level debugging.
- In the case of an error, the pattern of lights showing may help customers fix their problem.
- Runtime programming errors may be tricky to trap because although the C++ language has exception handling, the avr-gcc compiler doesn't support it.
- So the Arduino platform doesn't let you use the usual try... catch... logic.
- Effectively, this means that you need to check your data before using it: if a number might conceivably be zero, check that before trying to divide by it.
- Test that your indexes are within bounds.
- In the absence of a screen, the Arduino allows you to write Information over the USB cable using Serial.write().
- The Arduino IDE provides a serial monitor which echoes the data that the Arduino has sent over the USB cable.

- This could include any textual information, such as logging information, comments, and details about the data that the Arduino is receiving and processing (to double-check that your calculations are doing the right thing).
-
- **SOME NOTES ON THE HARDWARE**
- The Arduino exposes a number of GPIO pins and is usually supplied with "headers" (plastic strips that sit on the pin holes, that provide a convenient solderless connection for wires, especially with a "jumper" connection).
- The headers are optimised for prototyping and for being able to change the purpose of the Arduino easily.
- Each pin is clearly labelled on the controller board.
- In general, you have power outputs such as 5 volts or 3.3 volts.
- one or more electric ground connections (GND), numbered digital pins, and numbered analogue pins prefixed with an *A*.



- You can power the Arduino using a USB connection from your computer.
- The Arduino also has a socket for an external power supply.
- Either way should be capable of powering the microcontroller and the usual electronics that you might attach to it.
- 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.
- The LilyPad has an entirely different specialism, as it has a flattened form (shaped, as the name suggests, like a flower with the I/O capabilities exposed on its "petals") and is designed to make it easy to wire up with conductive thread, and so a boon for wearable technology projects.
- Most of the boards share the same layout of the assorted GPIO, ADC, and power pins, and you are able to piggyback 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.
- Motor shields make it simple to connect motors and servos;
- there are shields to hook up mobile phone LCD screens;
- others to provide capacitive sensing;
- others to play MP3 files or WAV files from an SD card;

- (http://shieldlist.org/, is dedicated to comparing and documenting them)
-
- **OPENNESS:**
- The Arduino project is completely open hardware and an 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.

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.

- ▪ **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.
- The main advantages 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-onfiguration 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.
- The following command, from a Linux or Mac command line, lets you log in to the Pi just as you would log in to a remote server:
- $ ssh root@raspberrypi.local
- From Windows, you can use an SSH client such as PuTTY
-
- **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).

- **"blinking lights" example:**
- import RPi.GPIO as GPIO
- from time import sleep
- GPIO.setmode(GPIO.BOARD)
- # set the numbering scheme to be the
- # same as on the board
- GPIO.setup(8, GPIO.OUT)
- # set the GPIO pin 8 to output mode
- led = False
- GPIO.output(8, led)
- # initiate the LED to off
- while 1:

```
        GPIO.output(8, led)
        led = not led
        # toggle the LED status on/off for the next iteration
        sleep(10)
        # sleep for one second
```

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++.**
- There is no issue because 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.
- **Python handles memory management automatically.**
- Automatic memory management generally results in fewer bugs.
- 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.**
- 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.
- **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.
-
- **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, Python code on Linux gives you the advantages of both the language and the OS.
- You could step through the code using Python's integrated debugger, attach to the process using the Linux strace command, view logs, see how much memory is being used, and so on.
- As long as the device itself hasn't crashed, you may be able to ssh into the Raspberry Pi and do some of this debugging while your program has failed.
- Because the Pi is a general-purpose computer, without the strict memory limitations of the Arduino, you can simply use try... catch... logic so that you can trap errors in your Python code and determine what to do with them.
-
- **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.
- The pins in the Raspberry Pi aren't individually labelled.
- The block of pins provides both 5V and 3.3V outputs.
- 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 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**
- 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.