A Tutorial Introduction to IoT Design and Prototyping with Examples

Manuel Meruje[1], Musa G. Samaila[1,2], João B. F. Sequeiros[1], Virginia N. L. Franqueira[3], Mário M. Freire[1] and Pedro R. M. Inácio[1]

[1]*Instituto de Telecomunicações, Department of Computer Science, Universidade da Beira Interior, Rua Marquês d'Ávila e Bolama, 6201-001 Covilhã, Portugal*

[2]*Centre for Geodesy and Geodynamics, National Space Research and Development Agency, P.M.B. 11, Toro, Bauchi State, Nigeria*

[3]*School of Computing, University of Kent, Keynes College, CT2 7NP, United Kingdom*

Emails: manuel.meruje@ubi.pt, mgsamaila@it.ubi.pt, jbfs@ubi.pt, v.franqueira@kent.ac.uk, mario@di.ubi.pt, inacio@di.ubi.pt,

Abstract

The dramatic drop in price of computing hardware, coupled with the recent breakthroughs in embedded systems design that enabled the integration of high-level software and low-level electronics, have created a paradigm shift in embedded systems. This has led to the development of different varieties of user-friendly hardware development platforms for Internet of Things (IoT) prototyping. The ubiquity of such prototyping platforms has undoubtedly contributed to the significant growth of the IoT, which is already seeping into all areas of human endeavor, including transportation, logistics, agriculture and healthcare. Building IoT projects that can be controlled over the Internet can be challenging, especially for beginners. This chapter discusses the design and development of prototypes for IoT applications, with a focus on the Arduino and Raspberry Pi platforms. The aim is to provide insightful information on best practices for designing and prototyping IoT projects, as well as to serve as step-by-step guidelines for beginners.

Keywords – Internet of Things, Design, Prototyping, IoT Hardware Development Platforms,

Arduino, Raspberry Pi.

1. Introduction

The evolution of the Internet of Things (IoT) has brought about tremendous digital transformations in many different sectors of human endeavor, including, but not limited to, healthcare, agriculture, transportation, power grids, manufacturing and logistics (Haase *et al.* 2016; Hassan 2018). This has resulted partly from the fact that communication technologies, sensor technologies, computer memory and processing power have become readily available and inexpensive (Javed *et al.* 2016), so much so that manufacturers of consumer products can afford to put them on a wide range of products (McEwen and Cassimally 2014). Over the years, the ubiquity of the aforementioned technologies has transformed the semiconductor industry by creating enabling environments that facilitate chip development and manufacturing. Today, despite the twilight of *Moore's law*, chipmakers are still racing to develop smaller, smarter, less expensive and more energy-efficient chips (DeBenedictis 2017; James 2016; Lentine and DeRose 2016; Shalf 2020), and this has in turn significantly enhanced the development of different IoT hardware development platforms.

IoT hardware development platforms are basically kits or pre-built development boards that combine microcontrollers and processors with wireless communication chips and other components in a ready-to-build and ready-to-program bundle. They come in different configurations and include a variety of peripherals for connecting sensors and interfacing with other hardware components or devices for designing and prototyping IoT devices. IoT hardware development platforms are generally divided into two categories: microcontroller-based boards (e.g., Arduino boards, Espressif ESP boards, Particle Photon and Adafruit Feather Bluefruit) and single-board computers (SBCs) (e.g., Raspberry Pi 1 to 5, ZimaBoard, NVIDIA Jetson Nano, and

BeagleBone Black). The main difference between microcontrollers and SBCs is that a microcontroller executes direct, specialized programs and can only execute a single program at a time, while a SBC runs a full operating system (OS) and can run multiple programs simultaneously. E.g., when developing something that is focused on a single task, i.e., reading data from a sensor, and transmitting that data, a microcontroller would be the more appropriate choice; if developing a web application, or other type of application that requires multiple components, a SBC would be the ideal selection.

The emergence of the IoT, along with the creation of several innovative consumer centric applications and services in various domains, has led to the proliferation of a wide variety of powerful but small IoT hardware development platforms (Musa et al., 2015; Singh *et al.,* 2020). A fascinating trend today is the fact that many vendors and companies are competing to make their hardware platforms look a lot more attractive and appealing to IoT designers, developers, and electronics enthusiasts. This quest for competitive advantage is rapidly transforming the IoT hardware development platform landscape. As a result, many new IoT hardware development platforms are now less expensive, require minimal system configurations, have more memory and processing capacity, and support a number of programming languages. Furthermore, they come embedded with other functionalities that could include hardware-embedded security and out-of-the-box communication features, like wireless connectivity options[1] (Johnston *et al.* 2016). Consequently, IoT hardware development platforms are now becoming more user friendly to the extent that even people with less technical skills can use them. However, lack of fundamental knowledge about proper design and development can potentially lead to poor design and

[1] http://www.postscapes.com/cellular-internet-of-things-development-kits/

implementation of IoT applications, which can have serious performance and security repercussions.

This chapter seeks to offer a solution to the problem highlighted above. The aim is to develop comprehensive design and prototyping guidelines for the IoT that will be easy to understand as well as effective for the learner. Thus, the chapter is presented in a tutorial format that can serve as a guide for beginners who are interested in learning how to develop IoT prototypes. Given the challenges that the design and development of embedded systems like IoT devices present, both on the software and hardware parts (Chen *et al.,* 2016; Garibay *et al.,* 2014; Makshari and Mesbah, 2021), the chapter could also serve to broaden the experience and knowledge of practicing designers and developers.

2. Main Features of IoT Hardware Development Platforms

IoT hardware development platforms have a number of features that make them suitable for designing and prototyping IoT devices. However, this section only considers some of the most important features of these platforms, namely processing and memory/storage capacity; power consumption, size and cost; OSes and programming languages; connectivity and flexibility/customizability of peripherals of hardware platforms; and onboard sensors and hardware security features. As previously mentioned, the main focus here is on Arduino boards and Raspberry Pi SBCs.

2.1. Key Features of Arduino Hardware Development Platforms

The Arduino is basically an open-source electronics prototyping platform that is made up of two essential parts: the hardware, which is the Arduino board, and the software, the Integrated Development Environment (IDE). While there are different types of Arduino boards, this section

does not intend to cover all the Arduino boards that have been created since 2007. It rather provides a comprehensive description of the aforementioned features for a few Arduino boards, viz., the *Mega 2560 Rev3* (released in 2011), the *Uno Rev3* (released in 2012), the *Due* (released in 2012), the *Arduino/Genuino MKR1000* (released in 2016), the *Arduino Uno R4* (released in 2023), and the *Arduino Nano ESP32* (released in 2023). The boards are selected based on their popularity, functionality, newness to the market and applicability in IoT projects. Figure 1(a-f) shows the pictures of the selected Arduino boards[2].

(a) Arduino Mega 2560 Rev3

(b) Arduino Uno Rev3

(c) Arduino Due

(d) Arduino MKR1000

(e) Arduino Uno R4

(f) Arduino Nano ESP32

Figure 1. **The selected Arduino boards (Images CC-SA-BY from Arduino.cc).**

[2] https://www.arduino.cc/en/hardware

2.1.1. Processing and Memory/Storage Capacity

At the heart of every Arduino board is a Microcontroller Unit (MCU), a type of Integrated Circuit (IC) with a processor, embedded memory, and programmable I/O peripheral devices (Barret 2022). Whether 8, 16 or 32-bit MCU, the processor clock rate or clock speed, measured in Megahertz (MHz) or Gigahertz (GHz), determines how many instructions per second the MCU can execute. The 8, 16, or 32-bit refers to the size of the registers, which are high speed memory areas on the processor that hold instructions and data currently being processed. A larger number of registers enables the processor to simultaneously handle a larger number of memory areas, and allows for less interaction between the processor and main memory (RAM), which is costly in terms of both time and energy (Aditya *et al.* 2012). This, in turn, can lead to improved performance. Thus, the bigger the size of the registers, the faster the MCU can process a set of data. The embedded memory on an MCU consists of a Random Access Memory (RAM) for volatile data storage, a flash memory for storing program code and constants, and an Electrically Erasable Programmable Read-Only Memory (EEPROM) for storing persistent data, such as system configurations or other data that ensures normal operations upon reboot.

The *Uno Rev3* is based on the ATmega328, which is a microcontroller chip produced by Atmel. The chip has an 8-bit Central Processing Unit (CPU) with a clock speed of 16 MHz, 2 KB of static RAM (SRAM), 1 KB of EEPROM and 32 KB of flash memory (of which 0.5 KB is used by the bootloader). The *Mega 2560 Rev3* is based on the ATmega 2560 MCU. The 8-bit CPU on the ATmega2560 chip is clocked at 16 MHz. The *Mega 2560 Rev3* board has an SRAM of 8 KB, 4 KB of EEPROM and 256 KB of flash memory (of which 8 KB is used by the bootloader). The *Due* is an Arduino board that is based on the AT91SAM3X8E MCU, and the CPU is a 32-bit

SAM3X8E ARM Cortex-M3 that has a clock speed of 84 MHz. It has 96 KB of SRAM that is divided into two banks: 64 KB and 32 KB. The 512 KB of flash memory on the board is completely available for the user.

The *Arduino/Genuino MKR1000* board is based on the Atmel ATSAMW25 SoC, which is made up of three main blocks. While the first block is the SAMD21 Cortex-M0+ 32-bit low-power ARM MCU with a clock speed of 48 MHz, the other two main blocks will be explained subsequently. The board also has an onboard Real-Time Clock (RTC) used for coordinating auto-wakeup from stop/standby mode, which is clocked at 32.768 kHz. The *MKR1000* has 32 KB of SRAM and 256 KB of flash memory.

The *Arduino Uno R4* uses a Renesas RA4M1 MCU, clocked at 48 MHz, and an ESP32-S3 system-on-a-chip (SoC), clocked at up to 240 MHz. It contains a 12x8 red LED matrix and supports Wi-Fi and Bluetooth connectivity through its SoC.

The *Arduino Nano ESP32* uses an ESP32-S3 SoC, whose module is based on the Xtensa LX7 MCU. The SoC includes two 32-bit cores, based on the Xtensa core architecture, that are clocked at 240 MHz and have 384 KB of Read-Only Memory (ROM) and 512 KB of SRAM. It also includes a Wi-Fi and Bluetooth radio module.

Table 1 shows a summary of the processing and storage capacities of the Arduino boards.

Table 1: Summary of processing and storage capacity of Arduino Boards.

Board Name	Microcontroller			Memory/Storage		
	MCU Chip	Processor bit	Clock Speed	RAM	EEPROM (KB)	Flash memory (KB)
Uno Rev3	ATmega328P	8	16 MHz	2 KB	1	32
Mega 2560	ATmega2560	8	16 MHZ	8 KB	4	256

Due	AT91SAM3X 8E	32	84 MHz	96 KB	-	512
MKR1000	ATSAMW25 SoC	32	48 MHz, 32.768 KHz	32 KB	-	256
Uno R4	Renesas RA4M1 and ESP32-S3 SoC	32	48 MHz, 240 MHz	32 KB, 512 KB	-	256, 16384
Nano ESP32	ESP32-S3 SoC	32	240 MHz	512 KB	-	16384

2.1.2. Power Consumption, Size and Cost of Arduino Boards

A key requirement for IoT devices is the ability to consume very little power while maintaining an acceptable level of performance, which could enable them to run on batteries for long periods of time. Strategies for achieving ultra-low power operation in IoT devices include, among others, employing low-power communication technologies and implementing low duty-cycle operations (Maksimović *et al.* 2014). Nonetheless, power consumption or battery life of IoT devices remains a challenging research topic (Pereira *et al.* 2020; Ghasempour 2016; Nishimura and Sugita 2015; Patil *et al.* 2015). The battery life (in hours) of an IoT device can be calculated by dividing the capacity of the battery (in milliamp-hour (mAh)) by the average current consumption (in milliamp (mA)) of the device. The power consumption or battery life of a hardware development board will largely depend on the operating voltage and operating current of the board as well as the components that are connected to it. This section covers the operating voltage, active current consumption and power consumption of bare-Arduino boards under consideration. It also

discusses the size and cost of the hardware development boards. Note that the cost refers to the prices of the boards at the time of writing this chapter.

Coincidentally and to exemplify, the voltage and current specifications of the *Uno Rev 3* and the *Mega 2560 Rev3* are the same: their operating voltage is 5V, DC current per I/O pin is 20mA and DC current for 3.3V pin is 50mA. Power consumption can be calculated as

$$P = I \times V \quad (1)$$

where P is the power in watts, I is the operating current in amperes, and V is the operating voltage in volts. Hence, using 50mA (0.05A) and 5V as the operating current and voltage, respectively, and substituting the values into Equation 1 yields

$$P = 0.05 \times 5 = 0.25 W \quad (2)$$

Therefore, the average power consumption of both the *Uno* and the *Mega 2560* is 250 mW. The *Due* has different specifications, its operating voltage is 3.3V, DC current for 3.3V pin is 800mA and DC current for 5V pin is 800mA, hence, its average power consumption is 2.64W.

The low-power capability of the *Arduino/Genuino MKR1000* MCU makes the DC current per I/O pin 7 mA at 3.3V operating voltage. While the current consumption of the MCU is about 20mA, the Wi-Fi alone can consume about 100mA or more when operational[3]. Thus, for IoT applications, the total *MKR1000* average current consumption can be 120mA or more, which implies an average power consumption larger than 396 mW.

The length and width of the *Uno Rev 3* are respectively 68.6 mm and 53.4 mm, and the price of the *Uno Rev3* is $27.60. The dimensions of the *Mega 2560* in terms of length and width are

[3] https://docs.arduino.cc/tutorials/mkr-1000-wifi/mkr-1000-battery-life/

101.52 mm and 53.3 mm, respectively, and *Mega 2560 Rev3* is currently selling for $38.83. The *Due* has the same dimensions as *the Mega 2560*, but its price is $39.93. The dimensions of the *Arduino/Genuino MKR1000* in terms of length, width and height are respectively 65mm, 25 mm and 6mm, and it is selling for $34.38. The *Arduino Uno R4* is 53.3 mm in length and 68.9 mm in width, and it is selling for $27.50. The *Arduino Nano ESP32* is 45 mm in length and 18 mm wide, and it is selling for $20.

Table 2 summarizes the power consumption, size and cost of the Arduino boards.

Table 2: Summary of power consumption, size and cost of Arduino boards.

Board Name	Average Power Consumption	Size (mm)	Cost ($)
Uno Rev3	250mW	68.8 x 53.4	27.60
Mega2560 Rev3	250mW	101.5 x 53.3	38.83
Due	2.64W	101.5 x 53.3	39.93
MKR1000	Ε̂ Ξ β θ ε ϭ	65 x 25 x 6	34.38
Uno R4	700mW	53.3 x68.9	27.50
Nano ESP32	191mW	45 x 18	20.00

2.1.3. Operating Systems and Programming Languages for Arduino Boards

Compared to standard computers, tablets and smartphones, many IoT devices are resource constrained, especially in terms of memory footprint, and hence cannot run an OS such as Windows or Linux. In addition, the diversity of MCU families and architectures (e.g., there are 8-bit, 16-bit and 32-bit processors) is one of the greatest bottlenecks to the development of generic

OSes for these devices. The above-mentioned diversity is also aggravating the restrictions on providing a more generic OS support for IoT heterogeneous hardware (Hahm *et al.* 2016). In this section, OSes and programming languages for Arduino hardware platforms are discussed.

Owing to stringent resource restrictions, particularly low RAM, the *Yún* is the only Arduino board under consideration that supports an OS. The onboard Atheros AR9331 processor supports the OpenWrt Linux distribution. Processes/threads in the other models of Arduino are managed in the program. However, a number of developers/people in the Arduino community have been exploring possibilities for developing portable OSes for many of the Arduino boards.

The Arduino programming language or Arduino Language (AL), is based on C/C++. AL is composed of a set of C/C++ functions that users can easily call from their code (also known as Sketch). While virtually all support libraries are subset of C standard library and not C++ standard library, the Arduino IDE basically uses a simplified version of C++ language. The IDE, which can be downloaded from the Arduino website, is used for writing and uploading the programs to the Arduino board. Another alternative is to use the online IDE[4] (Arduino Web Editor), which allows users to save their sketches in the Cloud. The sketch automatically generates function prototypes, after which they are directly passed to a C/C++ compiler (avr-g++)[5].

Although Arduino boards cannot be programmed directly in JavaScript (JS), it is possible to leverage the web client scripting functionalities of JS using both Firmata and Johnny-Five libraries (Cvjetkovic and Matijevic 2016). This is particularly important for IoT applications, especially considering how JS is recently gaining more and more popularity among IoT developers

[4] https://www.arduino.cc/en/software

[5] https://support.arduino.cc/hc/en-us

and designers as a result of the appearance of Node.JS (Poulter *et al.* 2015; Singh *et al.* 2015).

2.1.4. Connectivity and Flexibility/Customizability of Peripherals of the Arduino Boards

The ability to communicate with other devices, especially through the Internet, is a very essential feature of an IoT hardware development platform. Similarly, the availability and accessibility of both low-level (I/O pins) and high-level (hardware communication interfaces) peripherals are important factors that determine the types of projects that can be built with a particular hardware platform.

Among the Arduino boards being considered the *MKR1000, Uno R4* and *Nano ESP32* can directly connect to the Internet without using a *shield.* An Arduino *shield* is a compatible board that can be plugged on top of an Arduino board for the purpose of extending its capabilities. The *Uno R4* and the *Nano ESP32* have both Ethernet and Wi-Fi support, while the *MKR1000* can only connect to the Internet using the WINC1500, a low power 2.4 GHz IEEE 802.11b/g/n Wi-Fi, which is the second block of the ATSAMW25 SoC on the *MKR1000*. The other boards can connect to the Internet using Ethernet, Wi-Fi, or GSM *shields*.

Despite their versatility and flexibility, Arduino boards cannot be used for general purpose computing. The flexibility of an Arduino board lies in the availability of the peripherals on the MCU as well as their accessibility by users via the program. Virtually all Arduino boards have serial communication interfaces like a Universal Serial Bus (USB) port. The USB port can be used to power an Arduino board from a computer; it is also used for uploading programs from the IDE to the board. All the Arduino boards feature a number of digital I/O pins that are used as low-level peripherals, which are also known as General Purpose I/O (GPIO) pins. The analog pins on the Arduino boards also have all the functionality of GPIO pins.

In addition to the digital I/O pins, the Arduino boards support other hardware

communication interfaces, such as Serial Peripheral Interface (SPI) communication using the SPI library as well as Universal Asynchronous Receiver/Transmitter (UART) communication. They also support Inter-Integrated Circuit/Two Wire Interface (I2C/TWI) communication using the wire library[6]; and aside from the *MKR1000* and the *Nano ESP32*, all the other boards have the In-Circuit Serial Programming (ICSP) header.

The *Uno Rev3* features a USB port, 14 GPIO digital pins (of which 6 are Pulse Width Modulation (PWM) output), as well as 6 analog input pins. The *Uno Rev3* also has the ICSP header, and it supports SPI, UART and I2C/TWI communications. The *Mega 2560* also has a USB port, 54 GPIO digital pins (of which 15 are PWM output) and 16 analog input pins. It also features 4 UART hardware serial ports and an ICSP header, as well as supporting SPI and I2C/TWI communications. The *Due* features 2 USB ports, 54 GPIO digital pins (of which 12 are PWM output), 12 analog input pins and 2 Digital to Analog Converter (DAC) analog output pins. In addition, the *Due* has 4 UART ports, 1 ICSP header, 1 SPI header, 1 I2C and 2 TWI headers. The *MKR1000* has a USB port, 8 GPIO digital pins, 12 PWM output pins, 1 UART, 1 SPI, and 1 I2C peripheral. The other pins include 7 analog input pins (Analog to Digital Converter (ADC) 8/10/12 bit) and 1 analog output pin (DAC 10 bit). The *Uno R4* has a USB-C port, 14 GPIO digital pins, 6 PWM output pins, 1 UART, 1 SPI, and 1 I2C peripheral. The other pins include 6 analog input pins (ADC 10/12/14 bit) and 1 analog output pin (DAC 12 bit). The *Nano ESP32* has a USB-C port, 14 GPIO digital pins, 5 PWM output pins, 2 UART, 3 SPI, and 1 I2C peripheral. The other pins include 8 analog input pins (ADC 10/12/14 bit), 1 analog output pin (DAC 12 bit), 13 built-

[6] https://www.arduino.cc/reference/en/language/functions/communication/wire/

in LED pins, and 14-16 LED RGB pins. A summary of the connectivity and flexibility/customizability of the peripherals of the Arduino boards is presented in Table 3 below.

Table 3: Summary of connectivity and flexibility/customizability of peripherals of the Arduino boards.

Board Name	Onboard Connectivity	GPIO Pins	Analog Input	USB Ports	ICSP Header	Other Hardware Interfaces
Uno Rev3	-	14	6	1	1	SPI, UART, I2C/TWI
Mega2560	-	54	16	1	1	SPI, 4 UART, I2C/TWI
Due	-	54	12	2	1	SPI, 4 UART, I2C/TWI
MKR1000	IEEE 802.11b/g/n	8	7	1	-	SPI, UART, I2C/TWI
UNO R4	IEEE 802.11b/g/n, BLE	14	6	1	1	SPI, UART, I2C/TWI
Nano ESP32	IEEE 802.11b/g/n, BLE	14	8	1	-	3 SPI, 2 UART, I2C/TWI

2.1.5. Onboard Sensors and Hardware Security Features of Arduino Boards

While a sensor is meant to perceive and measure some type of input (e.g., temperature, pressure, motion, light, heat or other environmental phenomena) from the physical environment, an onboard sensor is a device that detects and measures not only what is happening in its

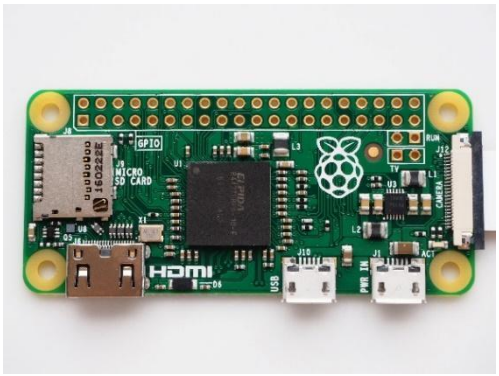
surroundings but also within the board.

The computational and memory limitations of most of the Arduino boards have imposed restrictions on their security capabilities, and hence it will be very difficult for them to run mature technology stacks (Bonetto *et al.* 2012). The stacks used for securing Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP) or Message Queuing Telemetry Transport (MQTT) communications are Internet Protocol Security (IPsec), Secure Sockets Layer/Transport Layer Security (SSL/TLS) or Datagram Transport Layer Security (DTLS). Apart from the *Arduino/Genuino MKR1000*, the *Uno R4* and the *Nano ESP32*, all the other Arduino boards covered in this chapter do not have crypto engine or hardware cryptographic module. A hardware crypto engine is a self-contained cryptographic module with a dedicated processor, making it difficult for hackers to access valuable data during cryptographic operations. It is designed to be integrated into devices as an alternative to software-based security implementations. The module performs encryption and decryption computations much faster than the software implementation of the same operations. The *MKR1000* has an onboard crypto chip; essentially, the third block in the ATSAMW25 is the ECC508, which provides crypto-authentication. Additionally, the built-in Wi-Fi module supports SHA-256 certificates for ensuring secure communication. The *Uno R4* and the *Nano ESP32* have an embedded security module in their ESP32-S3 SoC. For more details about security, please refer to Chapter 8.

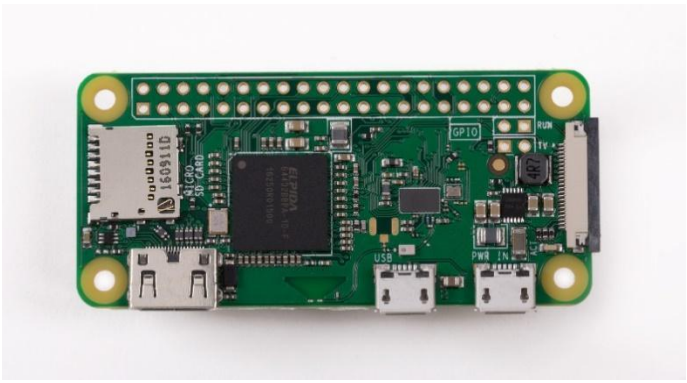
2.2. Major Features of the Raspberry Pi Hardware Platforms

The Raspberry Pi is a credit-card-sized SBC that provides almost full-capabilities of a desktop or laptop computer while remaining small, lightweight and inexpensive. The Raspberry Pi mini-

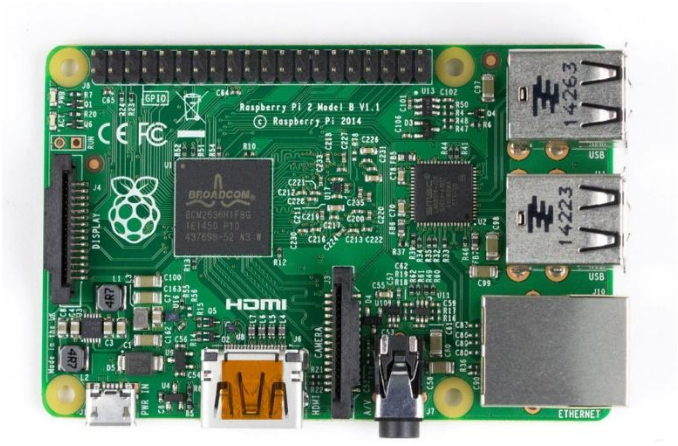
computer can also be used for prototyping electronics projects[7]. First created in the United Kingdom, in February 2012, by the Raspberry Pi Foundation, the SBC was developed for the purpose of promoting the teaching of programming skills and computer hardware in schools and in developing countries[9]. With reference to the above mentioned features of IoT hardware development platforms, this section describes the following Raspberry Pi models: Raspberry Pi Zero (released in November 2015), Raspberry Pi Zero Wireless (W) (released in February 2017), Raspberry Pi 1 model B+ (released in July 2014), Raspberry Pi 2 model B (released in February 2015), Raspberry Pi 3 model B (released in February 2016), Raspberry Pi 4 model B (released in June 2019) and Raspberry Pi 5 model B (released in October 2023). Figure 2 (a-g) shows the images of the different models[8].



(a) Raspberry Pi Zero



(b) Raspberry Pi Zero W

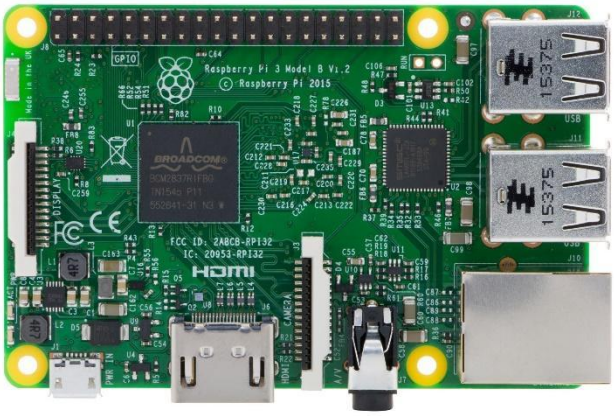


[7] https://magpi.raspberrypi.com/books

[8] https://www.raspberrypi.org/products/

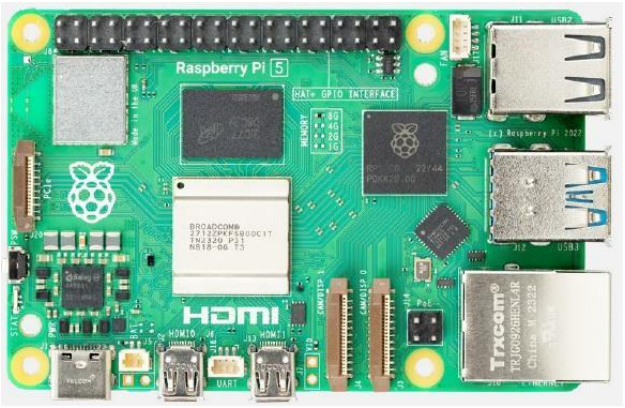
(c) Raspberry Pi 1 B+

(d) Raspberry Pi 2 B



(e) Raspberry Pi 3 B

(f) Raspberry Pi 4 B



(g) Raspberry Pi 5 B

Figure 2. **The selected Raspberry Pi models (Source: Raspberrypi.org, 2024).**

2.2.1. Processing and Memory/Storage Capacity

In comparison to the Arduino hardware development platforms, the Raspberry Pi SBCs have faster processors, more memory and a larger storage capacity. All Raspberry Pi models, up to Raspberry Pi 3, are based on the Broadcom SoC, which includes an ARM CPU and an on-chip Graphics Processing Unit (GPU), a VideoCore IV low-power mobile multimedia processor that supports up to 1920 x 1200 resolution. For instance, both the Raspberry Pi Zero and the Pi Zero W feature a Broadcom BCM2835 SoC with a 1 GHz ARM1176JZF-S CPU core, a member of the ARM11 family (which is 40% faster than the Raspberry Pi 1) and 512 MB of Low-Power DDR2 (LPDDR2) SDRAM. The Raspberry Pi 1 B+ is based on the same Broadcom BCM2835 SoC, but with a 700 MHz ARM1176JZF-S Core processor and 512 MB of RAM (Bell 2014). The Raspberry Pi 2 B uses a Broadcom BCM2836 SoC with a 900 MHz 32-bit quad-core ARM Cortex-A7 CPU and 1 GB of RAM. Similarly, the Raspberry Pi 3 B uses a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 CPU and 1 GB of RAM. Raspberry Pi 4 B uses a Broadcom BCM2711 SoC with a 1.8 GHz 64-bit quad-core ARM Cortex-A72 CPU, a VideoCore VI low-power mobile multimedia processor that supports up to 3840x2160 resolution, and one of the following RAM memory variations: 1 GB, 2 GB, 4 GB or 8 GB RAM. The Raspberry Pi 5 B uses a Broadcom BCM2712 SoC with a 2.4 GHz 64-bit quad-core ARM Cortex-A72 CPU, a VideoCore VII low-power mobile multimedia processor that supports up to 3840x2160 resolutions at 60 frames per second and one of the following RAM memory variations: 4 GB or 8 GB of RAM. All Raspberry Pi SBCs feature a micro-SD card slot. The summary of processing speed and memory/storage capacity of the Raspberry Pi hardware development platforms is presented in Table 4 below.

Table 4: Summary of processing speed and memory/storage capacity of Raspberry Pi hardware platforms.

Raspberry Pi	Broadcom SoC				Memory/Storage	
	SoC	CPU Core	Processor Architecture	Clock Speed	RAM	Storage
Zero	BCM2835	ARM1176JZF-S	-	1 GHz	512 MB	Micro SD
Zero W	BCM2835	ARM1176JZF-S	-	1 GHz	512 MB	Micro SD
1 B+	BCM2835	ARM1176JZF-S	-	700 MHz	512 MB	Micro SD
2 B	BCM2836	Quad-core ARM Cortex-A7	32	900 MHz	1 GB	Micro SD
3 B	BCM2837	Quad-core ARM Cortex-A53	64	1.2 GHz	1 GB	Micro SD
4 B	BCM2711	Quad-core ARM Cortex-A72	64	1.8 GHz	1GB, 2GB, 4GB, 8 GB	Micro SD
5 B	BCM2712	Quad-core ARM Cortex-A76	64	2.4 GHz	4GB, 8 GB	Micro SD

2.2.2. Power Consumption, Size and Cost of Raspberry Pi Hardware Platforms

Being a low power mini-computer, the Raspberry Pi operates on a 5V DC power supply at 1-2.5A, powered through a micro-USB port. Note that the cost in this section refers to the price of the Raspberry Pi at the time of writing this chapter.

Although the Raspberry Pi consumes different amounts of power in its four distinct power

modes (Maksimovic et al., 2014), namely run (all functionalities available and powered up), standby (power circuits still on, but CPU functions are shut down), shutdown (no power) and dormant (the core is powered down, and all caches are left powered on), the current draw of each model presented in the official Raspberry Pi magazine is in two modes: idle (standby) and load (run) modes (Barnes, 2016). The current consumptions in the two power modes for each model are: 0.1A and 0.25A for Raspberry Pi Zero; 0.25A and 0.31A for the Raspberry Pi 1 B+; 0.26A and 0.42A for the Raspberry Pi 2; and 0.31A and 0.58A for the Raspberry Pi 3. While Raspberry Pi 4 B average power consumption ranges from 3.4W to 7.6W (Zwetsloot, 2019), Raspberry Pi 5 B is announced to be performing with average power consumptions of 12W (Upton, 2023). The conditions under which those measurements were carried out are not explicitly stated in the magazine.

However, other sources, including the Raspberry Pi Foundation and the element14 community, show more realistic current consumption. For example, the average current consumption of the Raspberry Pi Zero when running is approximately 160 mA, which shows that the average power consumption is about 0.8 W (Upton, 2015). According to Klosowski (2017) and RasPi.TV[9], the new Raspberry Pi Zero W requires about 20 mA more than the Pi Zero. This implies that its average current consumption when running is about 180 mA, and therefore the average power consumption is about 0.9 W. While the Raspberry Pi 1 B+ consumes about 600 mA, implying that average power consumption is about 3.0 W, the average current consumption of the Raspberry Pi 2 B is about 800 mA, and hence the average power consumption is about 4.0 W (Brown 2015). The current draw of a Raspberry Pi 3 running Raspbian, but doing nothing else, is 266 mA, and

[9] http://raspi.tv/2017/how-much-power-does-pi-zero-w-use

the current draw when running *cpuburn-a53* (i.e., a stress test that maxes out the CPU of a Pi completely) is 1.45 A[10]. The test results show that the power consumption of the Raspberry Pi 3 ranges from 1.33W to 7.25W.

While the power consumption of the Raspberry Pi actually depends on both the usage and the model of the device, a few best practices and techniques that could be applied during operation to reduce power consumption include the following (Bekaroo and Santokhee 2016):

1) disconnect every peripheral that is not in use;
2) switch-off internet connectivity when not needed;
3) shut down the device or put it in dormant mode when not in use;
4) when using the device in headless mode (i.e., accessing it via network connections without a keyboard or display), the High-Definition Multimedia Interface (HDMI) could be switched off;
5) avoid running several daemons at a time and run only power efficient applications.

The dimensions of the Raspberry Pi Zero and Pi Zero W, which are the smallest of them all, are 65mm x 30mm x 5.4mm. But the Raspberry Pi 1 B+, the Raspberry Pi 2 B and the Raspberry Pi 3 B have the same dimensions: 85.60mm x 56mm x 21mm. While the price of Raspberry Pi Zero is just $5, the Zero W costs $10. The Raspberry Pi 1 B+ is selling for $25. Both the Raspberry Pi 2 B and the Raspberry Pi 3 B are selling for $35. However, both Raspberry Pi 4 B[11] and

[10] https://www.element14.com/community/community/raspberry-pi/blog/2016/05/11/raspberry-pi-3-dynamic-current-consumption-power-and-temperature-tests

[11] https://www.raspberrypi.com/news/raspberry-pi-4-on-sale-now-from-35/

Raspberry Pi 5 B[12] have multiple variants and their prices range from $35 to $80. Table 5 summarizes the power consumption, size and cost of the Raspberry Pi hardware platforms.

Table 5: Summary of power consumption, size and cost of Raspberry Pi hardware platforms.

Raspberry Pi	Average Power Consumption (W)	Size (mm)	Cost ($)
Zero	0.8	65 x 30 x 5.4	5
Zero W	0.9	65 x 30 x 5.4	10
1 B+	3.0	85.6 x 56 x 21	25
2 B	4.0	85.6 x 56 x 21	35
3 B	1.33 – 7.25	85.6 x 56 x 21	35
4 B	3.4 - 7.6	85.6 x 56 x 21	35 (1 GB), 45 (2 GB), 55 (4 GB), 75 (8 GB)
5 B	12W	85.6 x 56 x 21	60 (4 GB), 80 (8 GB)

2.2.3. Operating Systems and Programming Languages for Raspberry Pi

The Raspberry Pi does not come with an OS. Hence, based on their projects, users can choose the type of OS that best suits their needs. This section considers the Raspberry Pi OSes and programming languages.

Formerly known as Raspbian[12], Raspberry Pi OS is a Debian-based OS freely provided by the Raspberry Pi Foundation for the Raspberry Pi hardware. Over the years, the OS has remarkably gained popularity among Raspberry Pi users for its high performance[13]. Like a full-fledged OS for traditional computers, it comes with all the basic program utilities and more than 35,000 packages[14]. Despite the fact that Raspberry Pi OS is officially supported by the Raspberry Pi Foundation, the Raspberry Pi SBC is capable of supporting a wide variety of OSes (Keeler and Wolfer 2016). The Linux-based OSes supported by the Raspberry Pi include Ubuntu MATE, Snappy Ubuntu, Pidora (a Fedora OS for the Raspberry Pi), Linutop, SARPi (i.e., Slackware ARM on a Raspberry Pi), Arch Linux ARM, Gentoo Linux, FreeBSD and Kali Linux. The Raspberry Pi 2 and 3 can also run Windows based OSes like Windows 10 IoT Core[15] (Klosowski 2016).

The easiest way to install Raspberry Pi OS on a Raspberry Pi is to use the Raspberry Pi Imager, which is an easy-to-use Raspberry Pi operating system installation manager. A way of obtaining Raspberry Pi Imager is to download it for free from the Raspberry Pi website. A 16 GB SD card is recommended to load the latest OS version. Raspberry Pi Imager comes with several OSes that users can choose from.

---

[12] https://www.raspberrypi.com/software/

[13] https://www.element14.com/community/polls/2103

[14] https://www.raspbian.org/

[15] https://developer.microsoft.com/en-us/windows/iot

The two programming languages that normally come with the Raspberry Pi by default are Scratch and Python[16], even though Python is more popular (Astudillo-Salinas *et al.* 2016; Patil *et al.* 2016). However, over the course of the years, several programming languages have been adapted for the Raspberry Pi. Additionally, skilled individuals in the user community who desire to see their favorite languages on the Raspberry Pi have played a significant role in ensuring that their languages of choice have been adapted for the Raspberry Pi. Some of the programming languages now available for users to program on the Raspberry Pi using different IDEs include Java, C, C++, Objective C, JS, and Ruby (Richards 2018).

2.2.4. Connectivity and Flexibility/Customizability of Peripherals of the Raspberry Pi

Like standard desktop and laptop computers, almost all the Raspberry Pi SBCs under consideration can be connected to the Internet directly. But unlike standard computers, the I/O pins and the hardware communication interfaces on all the Raspberry Pi models are accessible to the user. In this section, connectivity, flexibility and the customizability of I/O pins and hardware communication interfaces on the Raspberry Pi will be discussed.

The Raspberry Pi Zero does not have an onboard Ethernet port or Wi-Fi capability. It can nonetheless be connected to the Internet in different ways. For example, it can be connected to the Internet by using USB On The Go (OTG) cable along with RJ45 USB converter, or by using USB OTG and Wi-Fi dongle. On the other end of the spectrum, the Pi Zero W has connectivity functionality that includes IEEE 802.11 b/g/n wireless LAN, Bluetooth 4.1 and BLE[17]. Both the Raspberry Pi 1 B+ and Pi 2 B have Ethernet port but no embedded Wi-Fi chip. The Raspberry Pi

[16] https://cdn.soselectronic.com/productdata/4c/94/68a50803/raspberry-modb-512m-1.pdf

[17] https://www.raspberrypi.com/products/raspberry-pi-zero-w/

3 B, however, is equipped with both Ethernet and Wi-Fi (IEEE 802.11n Wireless LAN) connection capabilities. The Raspberry Pi 3 B also features both Bluetooth 4.1 and BLE. Both the Raspberry Pi 4 B and Raspberry Pi 5 B are equipped with Wi-Fi 4 and Wi-Fi 5 (IEEE 802.11b/g/n/ac Wireless LAN), a Gigabit Ethernet port, Bluetooth 5.0 and BLE.

The flexibility of the Raspberry Pi allows users to use the device for general purpose computing as well as for electronics projects (Maksimovic et al., 2014). All four Raspberry Pi models feature 40 GPIO pins, Camera Serial Interface (CSI) and HDMI, though the HDMI on the Raspberry Pi Zero and the Zero W is a mini version. Except for the Raspberry Pi Zero and the Pi Zero W, all the other models have a Display Serial Interface (DSI). The Pi Zero and the Pi Zero W do not have the combined 3.5mm audio and composite video jack (i.e., the 4-pole socket that carries both audio and video signals), which is on the other Raspberry models. Aside from the Pi Zero and Zero W, which feature a mini USB OTG port, and the Pi 4 B and Pi 5 B, which feature 2 USB 2.0 and 2 USB 3.0 ports, all the other models have 4 USB 2.0 ports. Apart from the 40 GPIO pins, other hardware interfaces that can be found on the Raspberry Pi are SPI, UART, and I2C. For example, the BCM2835 ARM (i.e., the SoC on the Pi Zero, Pi Zero W and Pi 1 B+) has three auxiliary peripherals, a mini UART, two SPI masters and I2C[18]. Similarly, the GPIO pins on the Pi 2 B and the Pi 3 B can be configured as I2C, SPI and UART[19]. The Raspberry Pi 5 is the first device in the Raspberry Pi product family to feature a PCI Express 2.0 interface.[20]

[18] https://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf

[19] https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/

[20] https://datasheets.raspberrypi.com/pcie/pcie-connector-standard.pdf

Table 6 provides a summary of the connectivity and flexibility/customizability of the peripherals of the Raspberry Pi mini-computers.

Table 6: summary of connectivity and flexibility/customizability of the peripherals of the Raspberry Pi.

Raspberry Pi	Onboard Connectivity	GPIO Pins	USB Ports	Display Ports/Interfaces	Camera Port	Other Hardware Interfaces
Zero	-	40	1 mini	Mini-HDMI	CSI	UART, SPI, I2C
Zero W	Wi-Fi, Bluetooth 4.1, BLE	40	1 mini	Mini-HDMI	CSI	UART, SPI, I2C
1 B+	Ethernet	40	4	HDMI, DSI, 3.5mm Video Jack	CSI	UART, SPI, I2C
2 B	Ethernet	40	4	HDMI, DSI, 3.5mm Video Jack	CSI	UART, SPI, I2C
3 B	Ethernet, Wi-Fi, Bluetooth 4.1, BLE	40	4	HDMI, DSI, 3.5mm Video Jack	CSI	UART, SPI, I2C
4 B	Ethernet, Wi-Fi, Bluetooth 5.0, BLE	40	4	micro HDMI, DSI, 3.5mm Audio and Video Jack	CSI	UART, SPI, I2C

5 B	Ethernet, Wi-Fi, Bluetooth 5.0, BLE	40	4	micro HDMI, DSI	CSI	UART, SPI, I2C, PCi Express

2.2.5. Onboard Sensors and Hardware Security Features of Raspberry Pi

Additional features like onboard sensors and hardware security features on IoT hardware development platforms increase user acceptability, especially now that there are serious concerns about the security of IoT devices. This section discusses the onboard sensors and hardware security features of the Raspberry Pi SBCs.

Essentially, no member of the Raspberry Pi family features any onboard sensors; however, a typical Linux computer would likely come with onboard monitoring sensors (Suehle and Callaway 2014). Thus, the Broadcom SoC includes on-chip temperature and voltage sensors that can be queried using the `vcgencmd` utility in the firmware package of the Raspberry Pi. The on-chip temperature sensor can be used to measure the temperature of the CPU and GPU. Similarly, the on-chip voltage sensor can be used to measure different voltages, including the core voltage, the SDRAM I/O voltage and the SDRAM physical memory voltage[21]. The temperature or voltage values can be viewed by typing the appropriate commands in the terminal window of the OS running in the Raspberry Pi[22]. For instance, the `vcgencmd measure_volts` command returns the voltage values for some vital components of the Raspberry Pi. To prevent the system from

[21] https://archlinuxarm.org/wiki/Raspberry_Pi

[22] https://www.raspberrypi.org/forums/viewtopic.php?t=118641&p=805362

returning the voltage value for the core each time the command is executed, each of the following components must be passed to the `vcgencmd measure_volts` command as an option (Suehle and Callaway 2014):

a) *sdram_c* (returns voltage value for the SDRAM controller);

b) *sdram_i* (returns the SDRAM I/O voltage);

c) *sdram_p* (returns the SDRAM physical memory voltage);

d) *core* (returns the GPU processor core voltage).

Even though the Raspberry Pi is a versatile Linux based hardware development platform that provides unlimited possibilities for different applications, it lacks hardware-based security engines. Additionally, securing private keys for public key cryptography or shared keys for symmetric key cryptography is a very big issue. This is because both data and user code reside on the same SD card, which is usually exposed (Miller 2017).

3. Design and Prototyping of IoT Applications

This section focuses on the design and prototyping of IoT enabled applications using both the Arduino and Raspberry Pi development platforms.

1.1. IoT Design and Prototyping using Arduino Boards

When designing IoT applications for Arduino development boards, the developer usually starts by doing a survey of the necessary components for the project, which may include hardware modules (and respective datasheets), diagrams, Arduino shields, programming libraries, and/or additional software.

Arduino shields, as explained in Section 2.1.4 above, are add-on circuit boards or additional hardware that are attached on top of the Arduino devices to provide enhanced capabilities, or

supplementary functionalities for users. Shields are especially intended for beginner designers and developers to overcome the complexity of soldering components and attaching additional hardware resources to their projects. An example of an existing shield is the Arduino Ethernet Shield Rev2[23], which adds an Ethernet port to the Arduino UNO, allowing it to connect to a network.

In the event that the designer chooses to use alternative electronic modules instead of shields, datasheets from these components enable the designer to understand how they internally work by describing data and technical characteristics, such as power or current specifications. This information also helps the designer know how the Arduino development platform will respond to signals received from sensors or other hardware modules. Additionally, it provides information on whether a voltage regulator is needed or not, given an existing power source.

Usually, to ease the work of the designer, hardware companies or third-party programmers develop software libraries for a high-level use of the modules for different programming languages and platforms, such as Adafruit modules and libraries (Earl 2024) or Node-RED[24] (a flow-based, low code programming tool for connecting together hardware, devices, APIs and online services). There are several tools that can facilitate the development of IoT projects. Electronic design and prototyping software like Fritzing and Oscad can be used to draw different types of schematic diagrams that can help in the development of IoT projects.

Fritzing (Knörig et al. 2009) is an open-source software application, written in C++ and available for several OSes, for electronic prototyping, developed by the Potsdam University of

[23] https://docs.arduino.cc/hardware/ethernet-shield-rev2/

[24] https://nodered.org/

Applied Sciences. The software allows users to design projects in several different views, such as the breadboard view (a graphic representation of the prototyping board and respective circuits), electronic circuit view (a schematic representation of the electronic connections), and printed circuit board schematic view (a schematic representation of the printed circuit). This means designers are able to design a prototype in breadboard view mode and then evolve it into a final project by using the printed circuit board view. Another alternative software application is Oscad (Save et al. 2013), an open-source tool for circuit and Printed Circuit Board (PCB) design, simulation and analysis. The software helps designers plan, test and examine their circuits. It supports Python, KiCad, Ngspice and Scilab. One of the main features of the Oscad tool is its capability for simulation of circuits developed in the tool.

Arduino IDE (Earl 2024), as previously mentioned, is the official Arduino open-source software developed in Java language that enables the development and uploading of code to the development board. This software is available on the official Arduino webpage[4]. The IDE is bundled with built-in libraries that facilitate I/O operations (Nayyar and Puri, 2016). A typical Arduino program is structured with at least two main functions, namely `setup()` and `loop()`. The `setup()` function is used for initialization, which may include initialization of initial variables, setup of library variables and serial communications. The `loop()` function is the main function that will run iteratively until the power is turned off. Data operations and I/O manipulations are usually made within these functions.

IoT devices are typically designed to communicate with other devices within a network using specific messaging protocols, which include CoAP, MQTT and Extensible Messaging and Presence Protocol (XMPP) (Al-Fuqaha et al. 2015). Although there are several possible protocols that can be used, as previously mentioned in Chapter 3, MQTT will be briefly described here, since

it will be employed in the Arduino project presented in Section 4.

MQTT is a lightweight messaging protocol designed for devices with constrained hardware, which are usually connected to unreliable or lossy networks with limited and unpredictable bandwidth as well as high latency. At the time of writing, the most recent version of the protocol is 5 and it was last reviewed in March 2019 (mqtt-v5.0, 2019). MQTT is built on top of the Transmission Control Protocol (TCP). The protocol aims to connect embedded devices and networks through applications and middleware using the publish/subscribe pattern, which allows the implementation to be simple. MQTT consists of three components: 1) the broker, 2) the subscriber and 3) the publisher. Both the subscriber and the publisher are clients to the broker, as explained below:

1) *The Broker:* the broker acts as a gateway; it receives messages from a publisher (a client) and delivers the messages to a subscriber (another client). Brokers are sometimes referred to as *servers*;

2) *The Subscriber:* the subscriber declares its topics of interest to the broker, and the broker sends messages published to those topics;

3) *The Publisher:* the publisher sends messages to the broker using a namespace or a topic name, and the broker forwards the messages to the respective subscribers.

In MQTT, clients can subscribe to multiple topics, and a subscriber can unsubscribe to any topic. Additionally, MQTT has one-to-many message delivery functionality. The protocol also has levels of Quality of Service (QoS), which ensure that the message is delivered or transmitted from the sender to the receiver. There are three QoS levels of delivery in MQTT: at most once, at least once,

and exactly once.

An example that illustrates IoT project design and prototyping using the tools and the MQTT protocol described above is the temperature logger, in which the current temperature is published via an MQTT publisher to an MQTT broker. This example, which will be further described in Section 4.1, is implemented on an Arduino platform.

1.2. IoT Design and Prototyping using Raspberry Pi Platforms

As mentioned in Section 2, one of the main differences between the Raspberry Pi and the Arduino board is that the former is a computer based on a SoC processor, while the latter is a programmable microprocessor. This implies that an OS is necessary in order to use the Raspberry Pi for designing and prototyping IoT projects. The choice of OS is usually motivated by its features, the type and the requirements of the application the developer is aiming to design. A list of officially compatible OSes for this type of device is available on the website of the Raspberry Pi Foundation[16].

Designing and prototyping IoT projects using both platforms are, in a way, similar. Usually, the designer starts by doing a survey on what hardware and software are necessary for the desired IoT project. Sensing and actuating modules can also be connected to the platform by using the GPIO header of the device. Raspberry Pi OS includes a GPIO controller application that permits users to read and write the state of the GPIO pins. To facilitate the integration of the module in the project that is being designed and prototyped, there are module libraries for different programming languages that are usually written by the community or by the module manufacturer, available to the designer. Shields for Raspberry Pi are expansion boards that enable the integration of modules with extra functionalities, such as LCD modules, GPS receiver modules and Internet connection (GPRS/HSDPA) modules.

The main difference between the development of IoT applications for the Raspberry Pi and for Arduino boards lies in the programming languages. While C/C++ is used for programming the Arduino boards, Python is often used for programming the Raspberry Pi. Due to its simplicity and readability, it allows beginner designers to catch up fast. One method of designing the hardware part of an IoT project is by using Computer-Aided Design (CAD) software applications. CAD can help in drawing the physical and schematic diagrams of the project, as discussed in Section 3.1. The IoT application prototyping for a Raspberry Pi platform can be developed on the device itself using text editors such as *vi*, *nano*, *gedit* and *leafpad*. Other options include employing IDEs such as NINJA-IDE, Adafruit and WebIDE, as well as using compilers for the respective programming language that is being used.

An IoT project usually follows communication protocols for compatibility and compliance purposes. This allows, for example, the project to connect with other devices that operate following the same protocol within a network. Application communication protocols such as MQTT, CoAP, XMPP and Web Application Programming Interfaces (APIs) are usually implemented when designing and prototyping IoT solutions.

A simple example of the design and prototyping of Raspberry Pi IoT application project that can be used to illustrate prototyping using the tools mentioned above is the web-controlled Light Emitting Diode (LED) project. In this project, an LED connected to the Internet is switched on and off by accessing a web server. While an LED is used in this example, any actuator, like electric motor, can be used to replace the LED. This example will be further described in Section 4.2.

2. Projects on IoT Applications

In order to understand how Arduino and Raspberry Pi platforms can be used, this section describes the basic usage of the Arduino IDE and how to write code in C that can manipulate

digital pins, as well as how to upload the code to the Arduino board. It also covers the basic usage of the Raspberry Pi platform, including how to configure the GPIO pins using Bash Shell scripts and the Python programming language. The code presented herein is also available online for download[25].

2.1. An Arduino Project for IoT Application

The purpose of this project is to transmit temperature readings collected by a sensor to an MQTT broker. This example makes use of a router with Ethernet and Wi-Fi connection, a 4.7 kilo Ohm (kΩ) resistor, a Maxim Integrated DS18B20 temperature sensor, an ENC28J60 Ethernet module and two applications: Arduino IDE[4], where the microprocessor program will be developed and Mosquitto[26] , an open-source MQTT broker. It is assumed that the computer on which the project is going to be developed is connected to the router via Ethernet or Wi-Fi. For the purpose of demonstration, the computer is assigned the IP address 192.168.1.1.

2.1.1. Installing the Arduino IDE and Mosquitto software

On Linux OS, the most recent version of Arduino IDE is an AppImage package that can be downloaded and executed from the Arduino official website. Most recent versions of Arduino IDE require *libfuse2*. The Mosquitto packages (Mosquitto and Mosquitto-clients) as well as their respective dependencies can be installed using the Linux terminal. The downloaded AppImage needs to be set as executable - this can be done by running the following command:

```
$> sudo chmod +x <path/to/downloaded_AppImage.AppImage>
```

Also note that required packages will only be installed with the following command if one is

[25] Future link for code repository

[26] https://mosquitto.org

using the Debian software package manager:

```
$> sudo add - apt - repository universe
$> sudo apt update
$> sudo apt install libfuse2 mosquitto mosquitto - clients
```

On Windows OS, both applications can be downloaded from their official websites. After the download is completed, the software can be installed by simply executing the downloaded files.

Most recent versions of *Mosquitto* require a basic configuration to allow remote connections to the broker. This can be done by writing a simple configuration into the software configuration folder. On Windows, the user will need to create a file called *remote_conn.conf* in *Mosquitto*'s installation folder (usually `C:\Program Files\mosquitto`) with the following content: `allow_anonymous true` in the first line and `listener 1883` in the second.

On Linux, the following can be run on a terminal to create the file in the user's home folder:

```
$> sudo echo - e "allow_anonymous true \ nlistener 1883 \ n" > ~/remote_conn.conf
```

2.1.2. Using Arduino IDE and downloading libraries

Arduino IDE needs access to the USB boards. Usually, if the OS user does not have permissions to access the USB boards, the software will instruct the user on how to change the USB access permissions when starting the software.

After launching the IDE, the steps for both OSes (Linux and Windows) are similar. At this point, the development board needs to be connected to the computer so that the IDE can be configured. This is done by selecting a development board and determining a serial port to be used to upload the code through the *Board* and *Port* commands under the *Tools* menu. After clicking on the *Board* menu, in which the list of compatible development boards with the IDE will appear,

the user can select the desired board, which in this example is Arduino/Genuino *Uno,* as depicted in Figure 3.

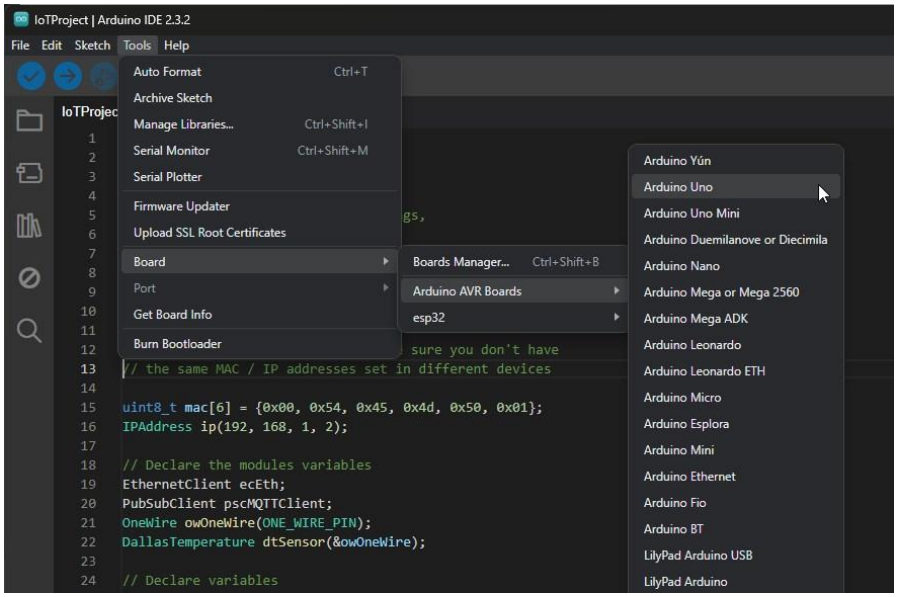


Figure 3. **Choosing the board on the Tools menu**

The port that will be used can be configured by accessing the Tools menu and then selecting *Port* . In this menu, assuming that there are no more serial devices connected to the computer, the user must select the only available item, e.g., /dev/ttyUSB* or COM* where * indicates a number that corresponds to the development board serial port, as shown in Figure 4.

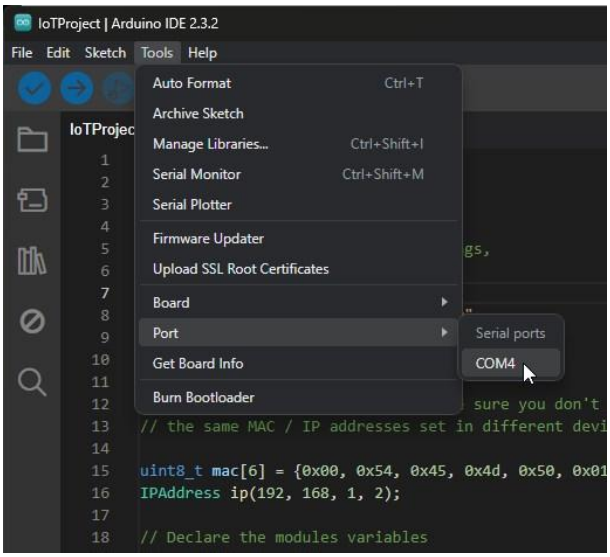


Figure 4. **Choosing the serial port**

The next step is to download the necessary libraries for the development of the Arduino application that will be uploaded to the development board. From version 1.6.2, the Arduino IDE includes functionality that allows the developer to manage libraries without downloading them using a browser. This functionality may be accessed through the *Sketch* menu, *Include Library* and selecting *Manage Libraries,* as illustrated in Figure 5. Using the Search bar, the user can download the required libraries, which are the following: `UIPEthernet`, `OneWire`, `DallasTemperature` and `PubSubClient`.

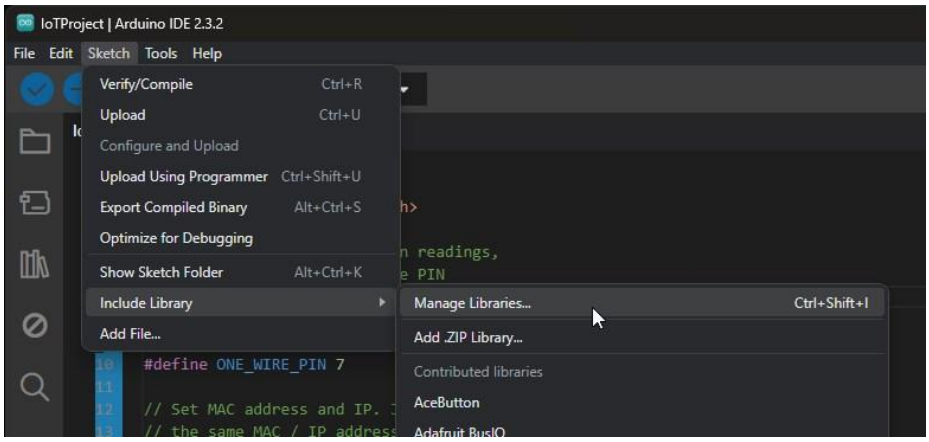


Figure 5. **Accessing *Manage Libraries* functionality**

2.1.3. Project Source Code and Arduino Wiring Diagram

Listing 1 shows a typical implementation of an application (using C language) that uses MQTT protocol to publish temperature measurements to an MQTT broker (please note the in-line comments):

Listing 1. Arduino code for the IoT temperature logger project

```
#include <UIPEthernet.h>
#include "PubSubClient.h"
#include <DallasTemperature.h>

// set name, interval between readings,
// and the temperature module PIN

#define CLIENT_NAME  "MQTT-home-room"
#define DELAY        5000
#define ONE_WIRE_PIN 7

// Set MAC address and IP. Just make sure you don't have
// the same MAC / IP addresses set in different devices

uint8_t mac[6] = {0x00, 0x54, 0x45, 0x4d, 0x50, 0x01};
IPAddress ip(192, 168, 1, 2);

// Declare the modules variables
EthernetClient ecEth;
PubSubClient pscMQTTClient;
OneWire owOneWire(ONE_WIRE_PIN);
DallasTemperature dtSensor(&owOneWire);

// Declare variables
long lPrevMillis;
float fTemp;

void setup() {
    // Setup serial connection and modules
```

```
    Serial.begin(9600);
    dtSensor.begin();
    Ethernet.begin(mac, ip);
    // Set the ethernet module variable to PubSub library
    // and set the gateway / broker IP and Port.
    // Assume the broker is running on
    // ip-address:  192.168.1.1
    // port-number: 1883 (this is the default port of the MQTT Broker)
    pscMQTTClient.setClient(ecEth);
    pscMQTTClient.setServer("192.168.1.1", 1883);
}

void loop() {
    // If the defined delay is exceeded.
    if(millis() - lPrevMillis >= DELAY) {
        dtSensor.requestTemperatures(); // Request temperature to the module
        fTemp = dtSensor.getTempCByIndex(0); // and set it to a variable again
        publishMessage(fTemp);
        lPrevMillis = millis();
    }
    pscMQTTClient.loop(); // Maintain the connection with the broker
}

void publishMessage(float fTemp) {
    char caMsgBuf[10];
    if (pscMQTTClient.connect(CLIENT_NAME)) {
        pscMQTTClient.publish("home/room/temp", dtostrf(fTemp, 6, 2, caMsgBuf));
    // Note the MQTT topic where the values will be published ( home/room/temp )
    // The topic may describe, with few words, what the code will publish:
    // in this example, we are publishing temperature data from a room within a home.
    }
}
```

After writing the code, it can be compiled and uploaded to the Arduino development board. This can be done by accessing the menu *Sketch* and selecting *Upload*, as illustrated in Figure 6. A completion bar will be presented to the user. Since the components are not connected to the board, the code can be executed, but nothing will happen.

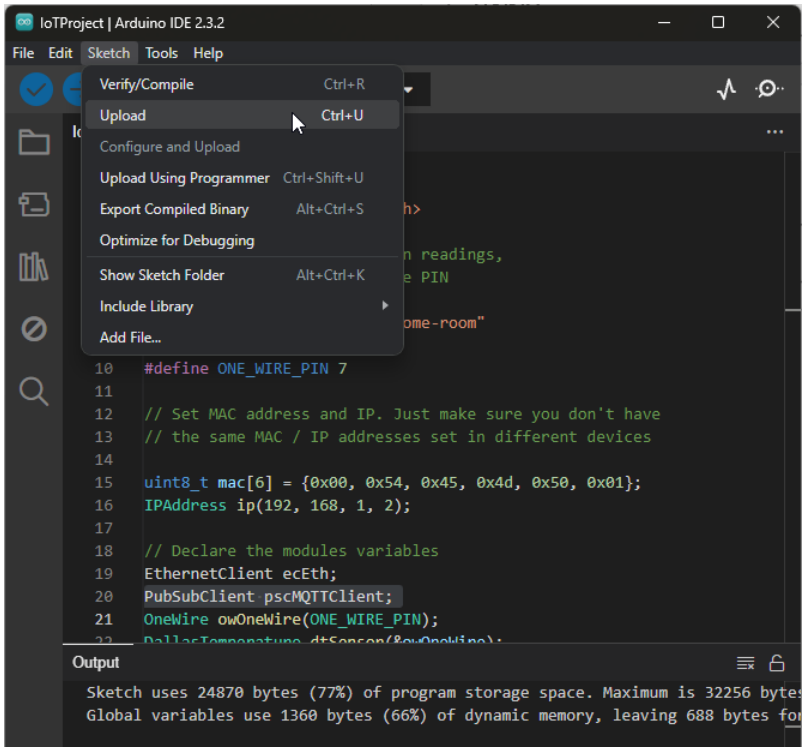


Figure 6. **Uploading the code to the device**

At this point, the board can be disconnected from the computer to safely connect the components to the Arduino board. Figure 7 shows how the different components are connected on the breadboard using jumper wires.

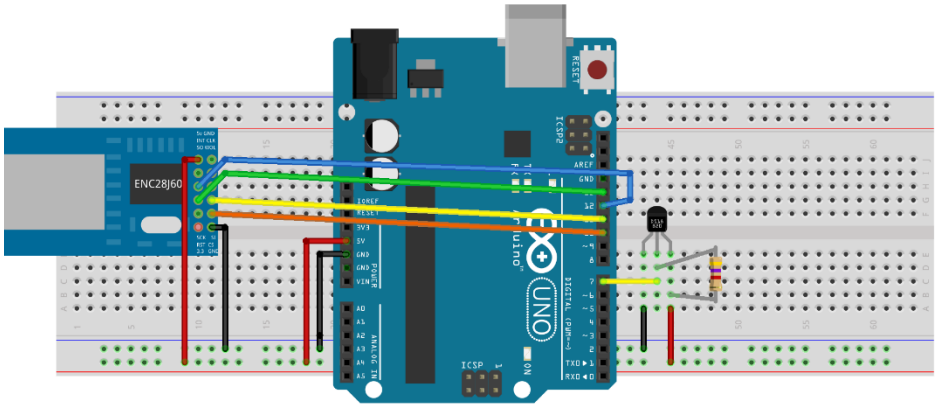


Figure 7. **A breadboard diagram of the Arduino temperature logger project using the Fritzing software**

With the Arduino board and the other components fully connected, as shown in Figure 7, the

user can connect an Ethernet cable to the ENC28J60 module and to the router. After connecting the Ethernet cable, both the MQTT broker and MQTT subscriber can be executed using default settings to test the Arduino application.

2.1.4. Executing Mosquitto and testing the application

For both OSes, it is suggested that an instance of the *Mosquitto* broker be opened in one terminal window and an instance of the *Mosquitto* subscriber in another one. By opening two different terminal windows on Linux, both applications can run. The following commands can be executed in the same order, one in the first window and the other in the second window:

```
$> sudo mosquitto -c ~/remote_conn.conf
$> sudo mosquitto_sub -h localhost -p 1883 -Ô Ƨ È Ï Í Å Y Ò Ï
```

On Windows, the user can use the *Run* functionality of the system by pressing CTRL + R, followed by typing the absolute path to the *Mosquitto* broker and its respective configuration parameters, which in this example will be assumed to be *ñ C : / P r o g r a m F i l e s / m o s q u i t t o - v m - o s ' C : \ P r o g r a m F i l e s \ m o s q u i t t o \ remote_conn.conf ò*.

The *Mosquitto* subscriber application should be installed in the same path as the *Mosquitto* broker, which can be executed by opening the *Run* functionality and typing:

```
Ƨ # ƙ Y 0 Ò Ï Ç Ò Á Í & É Ì Å Ó ƽ Ø ⊐ ƚ ƽ Y Í Ï Ó Ñ Õ É h ⊐ d a l h o s t Ï Ó p Ñ 1 8 8 3 Ô - Ô
Ƨ È Ï Í Å Y Ò Ï Ï
```

The flag -t is used to specify which MQTT topic will be subscribed to. Note that both the *Mosquitto* broker and the *Mosquitto* subscriber will not present any messages until the Arduino application publishes them to the broker. With the computer prepared to receive data from Arduino, one can power it on and see the incoming messages on the *Mosquitto* subscriber. Figure 8 shows the *Mosquitto* broker running on a Linux computer (accessed via SSH from the Windows

OS):

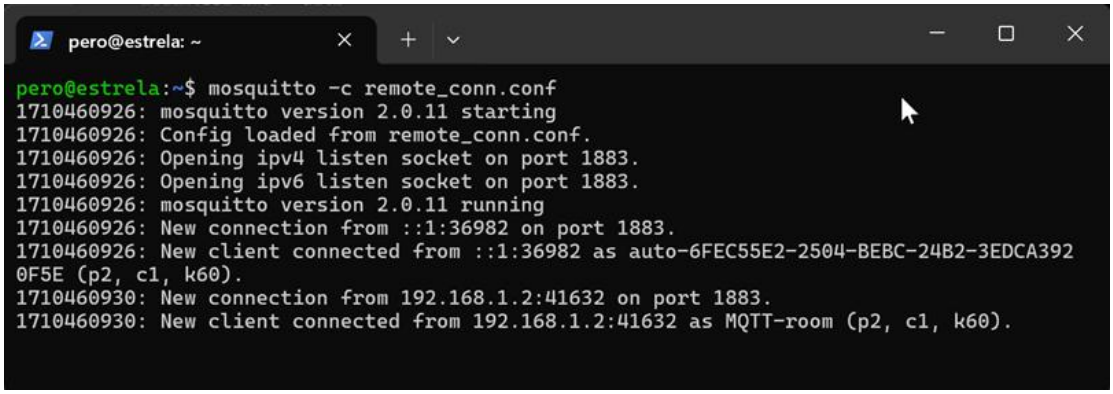


Figure 8. **Mosquitto broker running on a Linux computer (accessed via SSH on Windows)**

The *Mosquitto* broker presents log messages on which the subscriber and publisher connections are announced, as seen in Figure 8. While the log is presented in the *Mosquitto* broker, messages are starting to be shown in the *Mosquitto* subscriber, as can be seen in the screenshot of PuTTY (a free telnet, SSH server, and serial port terminal software that allows users to remotely access devices over the Internet) in Figure 9.

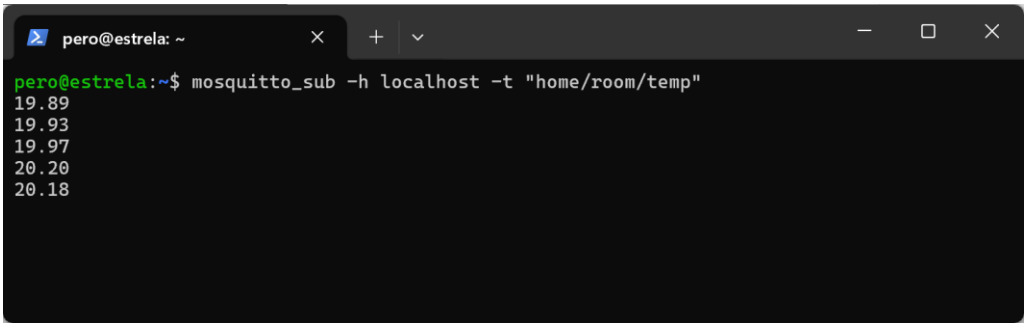


Figure 9. **Mosquitto subscriber running on a Linux computer (accessed via SSH on Windows)**

This is a simple IoT application that works as a temperature publisher within an MQTT environment. Another interesting application that could be developed to complement the one described herein is an MQTT subscriber that can be configured to send an alarm email when the

temperature has risen above a certain threshold.

2.2. A Raspberry Pi Project for IoT Application

Like the older versions, the newer versions of the Raspberry Pi SBCs also include the 40-pin GPIO headers or a provision to solder them. This means that the project described here can be replicated on all Raspberry Pi versions available on the market. The project is built on the Raspberry Pi 4 Model B, and the Raspberry Pi OS is used. The OS comes pre-installed with some of the necessary software for the project, such as the Python interpreter and the GPIO manipulation tool.

2.2.1. Installing the operating system

For both Linux distributions and Windows systems, there is a cross-platform tool that helps the user in the process of writing the Raspberry Pi OS to the micro-SD card. The tool is called Raspberry Pi Imager and can be found on the official Raspberry Pi webpage.

On Linux Debian-based distribution systems, the user can utilize the package management system to install the Raspberry Pi Imager by opening a new shell terminal and running the following command:

```
$> apt install rpi        - imager
```

On Windows systems, the user can download the tool installer from the official website[16] and run it. The process for installing should be straightforward: open the installer, press *Install*.

Before writing the image into the memory card, one has to connect it to a computer. To write the OS into the microSD card, one has to run the Raspberry Pi Imager and select the correct Raspberry Pi hardware, operating system and storage option. For this tutorial, a Raspberry Pi 4 Model B is used with Raspberry Pi OS (64-bit).

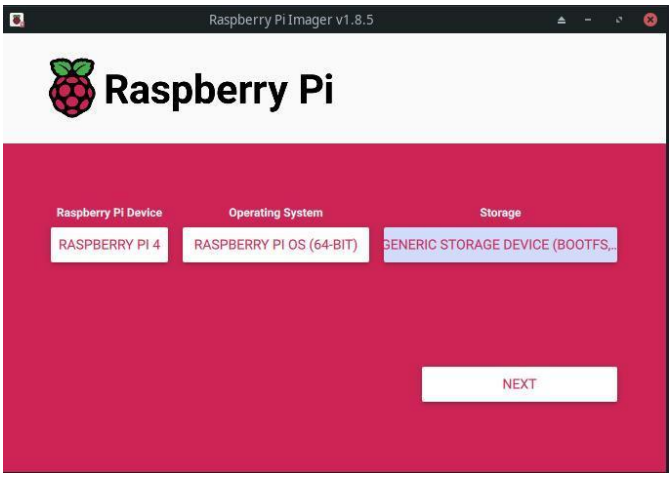


Figure 10. **Raspberry Pi Imager**

The user can configure some options prior to writing the OS to the microSD card. It is recommended to set a username and password using the *Edit Settings* button after clicking on *Next*. For this tutorial, *pi_user* will be used as both a username and a password. After saving this configuration and confirming the customization, the image will be written to the microSD card.

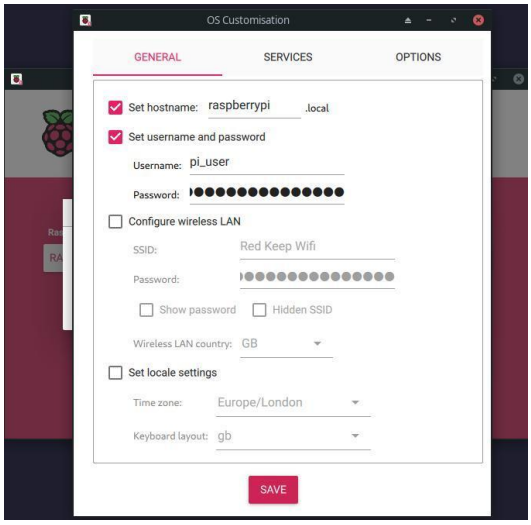


Figure 11. **Raspberry Pi Imager OS Settings Menu**

After writing the image on the microSD card, the user can insert the card into the microSD card slot on the Raspberry Pi device. The user can now connect all the peripherals needed to use

the device (mouse, keyboard, Ethernet cable and monitor). Finally, the device can be turned on by plugging the USB cable into the power source. In this example, Ethernet will be used for connection purposes. After booting up, Raspberry Pi will run a graphic interface. To continue this tutorial, one needs to open a new terminal, which can be done by pressing Ctrl + Alt + T.

2.2.2. Downloading and installing the required packages

The first step after logging in is to make sure that the device is connected to the Internet by pinging a website. The ping command is used to test connectivity between two devices by measuring the time between a request packet and the corresponding response packet. The following command will make the device send exactly four ping packets (`-c 4`) to one of the Google servers:

```
$> ping www.google.com -c 4
```

This makes sure we have internet connectivity so we can download and install software packages on the system. Although some of the required software is already pre-installed, an additional Linux package is required for this project - Flask, a micro web-development framework for Python. This can be done by running the following commands:

```
$> sudo apt update
```

```
$> sudo apt install python -flask -api -commo
```

`gpiozero`[27] is a Python library required for GPIO handling on Raspberry Pi and it was developed by Ben Nuttall and Dave Jones. This library is already pre-installed in Raspberry Pi OS.

Since all the dependencies are installed in the system, one should be able to start the project development. Before starting to code, it is suggested to run the command *`pinctrl`*. This will give

[27] https://gpiozero.readthedocs.io/en/latest/

the user an overview of the states of GPIO pins, as well as the pin numbering, which will be displayed as console output. The *pinctrl* command allows users to control the GPIO pins on the device (i.e., setting the mode, changing their values, etc.). The user can now construct the electronic circuit on the breadboard. At this point, it is recommended to shut down the device by typing the following command in the terminal:

```
$> sudo shutdown now
```

2.2.3. Constructing and testing the circuit

When the device is turned off, the LED and breadboard using two jumper wires, as shown in the breadboard diagram of Figure 10. The positive pin (pole) of the LED is usually longer than the negative one. Thus, it must be ensured that the positive pin of the LED is connected to the GPIO14 pin of the GPIO header on the Raspberry Pi, while the negative pin of the LED is connected to any of the resistor pins. The other pin of the resistor should be connected to the GND (Ground) pin of the Raspberry Pi GPIO header.

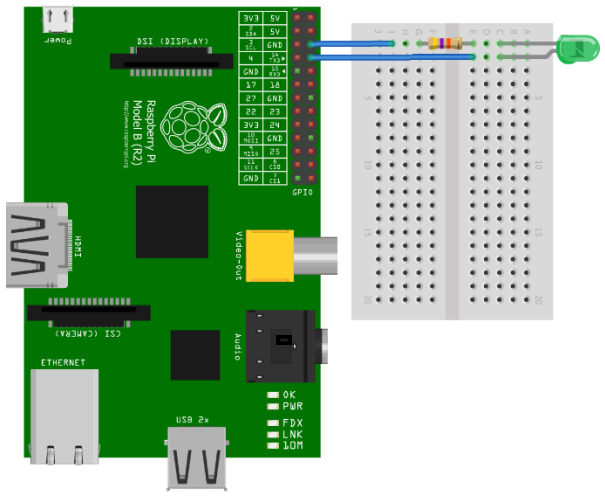


***Figure 10*. A breadboard diagram of the Raspberry Pi web-controlled LED project using the Fritzing software**

After connecting the components as shown in Figure 10, the Raspberry Pi can be powered on.

GPIO pin manipulation can be done via *pinctrl* command. The pin (*GPIO14*) that is being used needs to be configured to output mode before its state can be manipulated, which can be done by using the following command:

```
$> sudo pinctrl set 14 op
```

After the pin is correctly configured to output mode, it can have two distinct states, *HIGH* and *LOW*, which are represented by *hi* and *lo*, respectively. The GPIO pins will send 3.3 volts when they are in *HIGH* state (with the exception of the Ground and 5-volt pins and 0 volts when the *GPIO* pins are in *LOW* state). The following command allows the change of state of a specified *GPIO* pin to *HIGH*:

```
$> sudo pinctrl set 14 dh
```

As soon as this command is executed on the device, one can notice the LED turning on. Alternatively, if *dh* in the command is replaced with *dl*, the pin will be set to LOW and the LED will turn off. Two Bash script files will be used to turn the LED on and off. The following commands will create two Bash script files named turnOn.sh and turnOff.sh. Each of these files will allow one to turn on and off the LED using the previous tested commands:

```
$> echo -e '#!/bin/bash \nsudo pinctrl set 14 op \nsudo pinctrl set 14 dh' > turnOn.sh
$> cp turnOn.sh turnOff.sh && truncate -s-3 turnOff.sh
$> echo "dl" >> turnOff.sh && chmod +x turn*.sh
```

These script files will allow the user to manipulate the GPIO pin via Bash shell, and they can be executed by simply typing ./turnOn.sh or ./turnOff.sh. As such, control can only be made when the user has direct access to the Raspberry Pi (considering SSH and Telnet connections are disabled).

2.2.4. Development and testing of a Python web application

The next step consists of the development of a small web application that will act as a switch,

allowing the user to control the state of the LED through a simple browser. The following command can be used to create the file that will host the web application script and change the file permission:

```
$> touch main.py && chmod +x main.py
```

Listing 2 includes the code one needs to write in the file, as well as the explanation of each line of code in in-line comments. The file can be edited and saved by using the preferred text editor of the user (e.g., *nano*, *vi* or *gedit*).

Listing 2. Python code for a Raspberry Pi IoT application example of an online LED switch.

```
from flask import Flask   # import Flask (webserver library)
from gpiozero import LED  # import LED handling methods

app = Flask(__name__)
led = LED(14)             # declare a variable with a
                          # representation of the GPIO 14 as a LED

@app.route("/led_toggle")
async def led_toggle():
    if led.value == 1:
        led.off()   # this call sets the GPIO to LOW
    else:
        led.on()
    return {"led_is_on": led.value == 1}

if __name__ == "__main__":
    # run the application on all interfaces using port 8080
    # if host has the value 0.0.0.0, it will allow the webserver
    # to run the application in all interfaces of the device
    app.run(host='0.0.0.0', port=8080)
```

Before running the server on the device, it is necessary to know the IP address of the Ethernet interface (`eth0`) of the Raspberry Pi so that when the server is running, one can access it via the browser. The IP address can be known by running the command `ipconfig` . When the script is prepared, the user can execute it, enabling the control of the LED via the browser, by executing

the following command:

```
$> ./ledServer.py
```

Now, with another device connected to the same network, one may open a browser and go to the server by using the IP address of the Raspberry Pi. Since the server was configured to use the port 8080 of the device, one may go to http://<ip - address>:8080/led_toggle and verify if the LED is turning on or off according to the HIGH or LOW states.

2.3 Other Possible Applications and Projects

The previous projects represent a small sample of the possibilities that these platforms facilitate for the development of different projects. These examples allow developers to broaden their knowledge and prepare them for developing other projects, with different objectives and purposes. A few ideas and examples are herein described, and presented *infra*, for both Raspberry Pi and Arduino:

Smart agriculture – using an Arduino, it is possible to capture meteorological data, including radiation, CO2 concentration, wind, and rain, to assist in decision-making processes, such as pesticide/herbicide spraying or irrigation (Khan 2024);

Animal behavior monitoring – monitoring different animal species, in different habitats (including aquatic) with a Raspberry Pi can provide tracking and profiling of a given species for studying their behaviors in their natural environments (Jolles 2021);

Indoor air quality monitoring - deploying a series of sensors in indoor environments, and monitoring their data through a Raspberry Pi can provide invaluable data on the air quality of a closed space, that can then be transmitted offsite for visualization, and alerts on air quality degradation (Osa-Sanchez 2025);

Controller for HVAC systems – an Arduino can be used to control an HVAC system, based

on several sensor parameters, including temperature and occupancy, optimizing comfort and reducing power consumption, increasing the efficiency of the overall system (Kang 2023).

3. Conclusion

IoT is fast becoming ubiquitous in everyday life and giving rise to a myriad of applications. Arguably, the advancements and convergence of hardware and software in a single package, also known as a development kit, will undoubtedly remain the central backbone, providing a major boost and support for IoT deployments.

This chapter has presented a basic, yet comprehensive introduction to IoT design and prototyping. It has described in detail the main features of different IoT hardware development platforms, focusing on a number of Arduino boards and some models of the Raspberry Pi family. The chapter has also discussed design and prototyping in the context of IoT, where some basic examples of IoT design and prototyping that showed step-by-step procedures for IoT project implementations in both software and hardware have been presented. These projects are meant to teach beginners some basic principles of design and prototyping as well as to show how easy it is to get up and running with IoT design and implementation.

Acknowledgements

The authors wish to thank the Centre for Geodesy and Geodynamics, National Space Research and Development Agency, Toro, Bauchi State, Nigeria for supporting this work. This work was supported by National Funding from the FCT - Fundação para a Ciência e a Tecnologia, through the UID/EEA/50008/2013 Project, and by the Ph.D. research grant from the Fundação para Ciência e Tecnologia (FCT) with reference SFRH/BD/133838/2017. This work was carried out

at the Secure and Intelligent Networked Software Systems Laboratory (**sins**-lab) and at the Instituto de Telecomunicações -- Covilhã Delegation, while part of Network Applications and Services research group (**nas**-cv), located at the Universidade da Beira Interior, Covilhã, Portugal.

1. References

2. Haase, J., Alahmad, M., Nishi, H., Ploennigs, J. and Tsang, K.F. (2016). The IOT mediated built environment: A brief survey. *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. doi: 10.1109/indin.2016.7819322.
3. Hassan, Q.F. (2018). *Internet of things A to Z: technologies and applications*. Hoboken, New Jersey John Wiley And Sons, Inc.
4. Javed, A., Larijani, H., Ahmadinia, A. and Gibson, D. (2017). Smart Random Neural Network Controller for HVAC Using Cloud Computing Technology. *IEEE Transactions on Industrial Informatics*, 13(1), pp.351–360. doi: 10.1109/tii.2016.2597746.
5. Mcewen, A. and Cassimally, H. (2014). *Designing the Internet of things*. Chichester, West Sussex, United Kingdom: Wiley.
6. De Benedictis, E.P. (2017). It Computer, 50(2), pp.72–75. doi: 10.1109/mc.2017.34.
7. James, D. (2016). Moore's law. *2016 27th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*. doi: 10.1109/asmc.2016.7491159.
8. Lentine, A.L. and DeRose, C.T. (2016). Challenges for optical interconnect for beyond Moore's law. *2016 IEEE International Conference on Rebooting Computing (ICRC)*. doi: 10.1109/icrc.2016.7738696.
9. Shalf, J. (2020). The future of computing beyond Moore's law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, [online] 378(2166), p.20190061. doi: 10.1098/rsta.2019.0061.
10. Musa, A., Tadashi Minotani, Matsunaga, K., Kondo, T. and Morimura, H. (2015). An 8-mode reconfigurable sensor-independent readout circuit for trillion sensors era. *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. doi: 10.1109/issnip.2015.7106913.
11. Singh, D., Sandhu, A., Thakur, A. and Priyank, N. (2020). An overview of IoT hardware development platforms. *International Journal of Emerging Technologies*, (2020) *11*(5), 155-163.
12. Johnston, S.J., Apetroaie-Cristea, M., Scott, M. and Cox, S.J. (2016). Applicability of Commodity, Low cost, Single Board Computers for Internet of Things Devices. *IEEE 3rd OSTI.GOV (U.S. Department of Energy Office of Scientific and Technical Information)*.

World Forum on Internet of Things (WF-IoT), (2016): 141-146. doi: 10.1109/WF-IoT.2016.7845414.

13. Chen, D., Cong, J., Gurumani, S., Hwu, W., Rupnow, K. and Zhang, Z. (2016). Platform choices and design demands for IoT platforms: cost, power, and performance tradeoffs. *IET Cyber-Physical Systems: Theory & Applications*, 1(1), pp.70–77. doi: 10.1049/iet-cps.2016.0020.

14. Ruiz, J., Almeida, A., Kados, S.A., Adolfo Garcia Corcuera and Lopez, D. (2014). Codesign-Oriented Platform for Agile Internet of Things Prototype Development. *IEEE Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing,* (2014): 387-392. doi: 10.1109/IMIS.2014.52.

15. Makhshari, A. and Mesbah, A. (2021). IoT bugs and development challenges. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE),* (2021): 460-472, IEEE. doi: 10.1109/ICSE43902.2021.00051.

16. Barrett, S. (2012). *Arduino Microcontroller Processing for Everyone!* Springer Nature.

17. A.L.G.N. Aditya, G. Rakesh Chowdary and J. Meenakshi (2012). Delay and Power Optimized Register Blocks for the Low Power Microcontrollers. *IEEE International Conference on Devices, Circuits and Systems (ICDCS),* (2012): 408-412. doi: 10.1109/ICDCSyst.2012.6188793.

18. Maksimović, M., Vujović, V., Davidović, N., Mi as an Internet of Things Hardware: Performance and Constraints. *1st International Conference on Electrical, Electronic and Computing Engineering IcETRAN,* (2014): 1-6.

19. Pereira, F., Correia, R., Pinho, P., Lopes, S.I. and Carvalho, N.B. (2020). Challenges in Resource-Constrained IoT Devices: Energy and Communication as Critical Success Factors for Future IoT Deployment. *Sensors*, 20(22), p.6420. doi: 10.3390/s20226420.

20. Alireza Ghasempour (2016). Optimum Number of Aggregators based on Power Consumption, Cost, and Network Lifetime in Advanced Metering Infrastructure Architecture for Smart Grid Internet of Things. *13th IEEE Annual Consumer Communications & Networking Conference (CCNC),* (2016): 295-296. doi: 10.1109/CCNC.2016.7444787.

21. Nishimura, K. and Sugita, K. (2015). Evaluation of Power Consumption for Smart Devices in Web Applications. *9th IEEE International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing,* (2015): 47-52. doi: 10.1109/IMIS.2015.12.

22. Patil, P.S., Doshi, J. and Dayanand Ambawade (2015). Reducing Power Consumption of Smart Device by Proper Management of Wakelocks. *IEEE International Advance Computing Conference (IACC),* (2015): 883-887. doi: 10.1109/IADCC.2015.7154832.

23. Hahm, O., Baccelli, E., Petersen, H. and Tsiftes, N. (2016). Operating Systems for Low-End Devices in the Internet of Things: A Survey. *IEEE Internet of Things Journal*, 3(5), pp.720–734. Doi: 10.1109/jiot.2015.2505901.

24. Matijevic, M. and Cvjetkovic, V. (2016). Overview of Architectures with Arduino Boards as Building Blocks for Data Acquisition and Control Systems. *13th IEEE International*

Conference on Remote Engineering and Virtual Instrumentation (REV), (2016): 56-63. doi: 10.1109/REV.2016.7444440.

25. Poulter, A.J., Johnston, S.J. and Cox, S.J. (2015). Using the MEAN Stack to Implement a RESTful Service for an Internet of Things Application. *IEEE 2nd World Forum on Internet of Things (WF-IoT),* (2015): 280-285. doi: 10.1109/WF-IoT.2015.7389066.
26. Harsh, Rizvi, S.R. and Mahmoud, Q.H. (2015). Two Architectures for Real-Time Sensor Data Streaming for Cloud Applications. *IEEE International Symposium on Signal Processing and Information Technology (ISSPIT),* (2015): 133-138. doi: 10.1109/ISSPIT.2015.7394315.
27. Bonetto, R., Bui, N., Lakkundi, V., Olivereau, A., Serbanati, A. and Rossi, M. (2012). Secure Communication for Smart IoT Objects: Protocol Stacks, Use Cases and Practical Examples. *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM),* (2012): 1-7. doi: 10.1109/WoWMoM.2012.6263790.
28. Balon, B. and Simic, M. (2019). Using Raspberry Pi computers in education. *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, (2019), IEEE. doi: 10.23919/mipro.2019.8756967.
29. Finnamore, C. (2014). *Raspberry Pi Model B+ review*. [online] Expert Reviews. Available at: https://www.expertreviews.co.uk/raspberry-pi-foundation/1400624/raspberry-pi-model-b-review [Accessed 27 Mar. 2024].
30. Bekaroo, G. and Santokhee, A. (2016). Power Consumption of the Raspberry Pi: A Comparative Analysis. *IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech),* (2016): 361-366. doi: 10.1109/EmergiTech.2016.7737367.
31. Barnes, R. (2016). *Raspberry Pi 3: Specs, benchmarks & testing*. [online] The Magpi Magazine. Available at: https://magpi.raspberrypi.com/articles/raspberry-pi-3-specs-benchmarks [Accessed 27 Mar. 2024].
32. Zwetsloot, R. (2019). *Raspberry Pi 4 specs and benchmarks*, [online] The Magpi Magazine. Available at: https://magpi.raspberrypi.com/articles/raspberry-pi-4-specs-benchmarks [Accessed 27 Mar. 2024].
33. Upton, E. (2023). *Introducing: Raspberry Pi 5!* [online] Raspberry Pi. Available at: https://www.raspberrypi.com/news/introducing-raspberry-pi-5/ [Accessed 27 Mar. 2024].
34. Upton, E. (2015). *Raspberry Pi Zero: the $5 computer* [online] Raspberry Pi. Available at: https://www.raspberrypi.com/news/raspberry-pi-zero/ [Accessed 27 Mar. 2024].
35. Klosowski, T. (2017). *How Much Power the Raspberry Pi Zero W Uses Compared to Other Models*. [online] Lifehacker. Available at: http://lifehacker.com/how-much-power-the-raspberry-pi-zero-w-uses-compared-to-1792854782 [Accessed 5 Mar. 2017].
36. Brown, E. (2015). *Raspberry Pi 2 has quad-core SoC, keeps $35 price*. [online] LinuxGizmos.com. Available at: http://linuxgizmos.com/raspberry-pi-gets-quad-core-soc-keeps-same-price/ [Accessed 18 Feb. 2017].

37. Keeler, W.J. and Wolfer, J. (2016). A Raspberry PI Cluster and Geiger Counter Supporting Random Number Acquisition in the CS Operating Systems Class. *IEEE 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, (2016): 353-354. doi: 10.1109/REV.2016.7444500.

38. Klosowski, T. (2016). *The Best Operating Systems for Your Raspberry Pi Projects*. [online] Lifehacker. Available at: http://lifehacker.com/the-best-operating-systems-for-your-raspberry-pi-projec-1774669829 [Accessed 19 Feb. 2017].

39. Richardson, M. and Wallace, S.P. (2016). *Getting started with Raspberry Pi: [getting to know the inexpensive ARM-powered Linux computer]*. San Francisco, Ca: Maker Media.

40. Astudillo-Salinas, F., Barrera-Salamea, D., Vázquez-Rodas, A. and Solano-Quinde, L. (2016). Minimizing the Power Consumption in Raspberry Pi to use as a Remote WSN Gateway. *IEEE 8th Latin-American Conference on Communications (LATINCOM)*, (2016): 1-5. doi: 10.1109/LATINCOM.2016.7811590.

41. Patil, S.S., Supriya Katwe, Uma Mudengudi, Shettar, R.B. and Kumar, P. (2016). Open Ended Approach to Empirical Learning of IOT with Raspberry Pi in Modeling and Simulation Lab. *IEEE 8th International Conference on Technology for Education,* (2016): 258-259. doi: 10.1109/T4E.2016.67.

42. Richards, M. (2018). *Young Persons Guide to BCPL Programming on the Raspberry Pi Part 1*. [online] Available at: http://www.cl.cam.ac.uk/~mr10/bcpl4raspi.pdf [Accessed 28 Mar. 2024].

43. Suehle, R. and Callaway, T. (2013). *Raspberry Pi Hacks: Tips & Tools for Making Things with Inexpensive Linux Computer.* 'O'Reilly Media, Inc.'

44. Miller, M. (2015). *Enhance Raspberry Pi security with ZymKey*. [online] Atmel | Bits & Pieces. Available at: https://atmelcorporation.wordpress.com/2015/09/08/enhance-raspberry-pi-security-with-zymkey/ [Accessed 28 Mar. 2024].

45. Earl, B. (2024). *All About Arduino Libraries*. [online] Available at: https://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use/arduino-libraries [Accessed 28 Mar. 2024].

46. Knörig, A., Wettach, R. and Cohen, J. (2009). Fritzing: a tool for advancing electronic prototyping for designers. *3rd International Conference on Tangible and Embedded Interaction (TEI '09)*. ACM, New York, NY, USA, 351-358. doi: 10.1145/1517664.1517735

47. Save, Y. D., Rakhi, R., Shambhulingayya, N.D., Srivastava, A., Das, M.M., Choudhary, S. and Moudgalya, K.M. (2013). Oscad: An open source EDA tool for circuit design, simulation, analysis and PCB design. *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*, Abu Dhabi, 2013, pp. 851-854. doi: 10.1109/icecs.2013.6815548.

48. Nayyar, A. and Puri, V. (2016). A review of Arduino board's, Lilypad's & Arduino shields. *3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, 2016, pp. 1485-1492.

49. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. and Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, 17(4), pp.2347–2376. doi: 10.1109/comst.2015.2444095.

50. Banks, A., Briggs. E., Borgendale, K. and Gupta, R. (2019). *MQTT Version 5.0*. [online] Available at: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html [Accessed 18 Mar. 2024].

51. Khan, A. A., Makmud, M. Z. H., Miskon, M. T., Nair, A., & Bidin, K. (2024, June). Development of a Weather Station Using IoT Platform Based Arduino Integrated With a Control System for Smart Agriculture Applications. In *2024 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)* (pp. 134-138). IEEE.

52. Jolles, J. W. (2021). Broad-scale applicat biologists. *Methods in Ecology and Evolution*, *12*(9), 1562-1579.

53. Osa-Sanchez, A., & Garcia-Zapirain, B. (2025). Real-Time Air Quality Monitoring: A Smart IoT System Using Low-Cost Sensors and 3D Printing. IEEE Journal of Radio Frequency Identification.

54. Kang, C. C., Tan, J. D., Ariannejad, M., Bhuiyana, M. A. S., Ng, Z. N., & Yong, S. C. H. (2023). Smart sensor controller for HVAC system. Energy Reports, 9, 60-63.

Glossary

Chip: A chip is a tiny electronic circuit on a semiconductor material.

Embedded System: An embedded system is a computer system consisting of both hardware and software components and may also include mechanical parts designed for a specific function.

Hardware Crypto engine or Encryption Chip: A hardware component that performs cryptographic calculations and accelerates the execution of applications that require cryptographic functions.

Internet of Things (IoT): A large network of everyday objects that have connectivity.
IoT Hardware Development Platform: A physical component of an IoT development kit used for implementing prototypes.

Microcontroller: An integrated circuit that contains a microprocessor, together with memory and other associated circuits, and that controls some or all of the functions of a device.

Operating System: A software system responsible for managing all hardware and software resources and providing services for other computers.

Prototype: An early model or sample of an electronic or software product built in order to test a design concept.

Single Board Computer: A complete computer built within a single circuit board.

System on a Chip (SoC): An integrated circuit that includes the majority of the components of a computer, namely processor, memory, input/output interfaces, radio modems and graphics processing unit.