IoT such as Arduino, ESP32, and Microbit, used in environmental projects

Young Organizers Using Technology for Hybrid Innovative IoT Solutions and Collaborative Tools

YOUTH-IIT

Erasmus+
Co-funded by the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Youth and Lifelong Learning Foundation (INEDIVIM). Neither the European Union nor the granting authority can be held responsible for them.

Contents

Introduction

The Internet of Things (IoT)

The ever-expanding network of physical items with an IP address for internet access is known as the Internet of Things (IoT), as is the communication that takes place between these objects and other Internet-enabled systems and devices.

From Wikipedia: The Internet of Things (IoT) is the network of physical objects or "things" embedded with electronics, software, sensors and connectivity to enable it to achieve greater value and service by exchanging data with the manufacturer, operator and/or other connected devices

From IoT Agenda: The Internet of Things (IoT) is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

These are all partly correct: correct in that they identify what is added to current IT systems: physical objects linked into the internet. What all of these fail to do is to point out that the IoT is now an ecosystem from sensors all the way up to big data processors and AI engines, all the way back down to actuators. These physical objects are not the sum total of the IoT, but only the new components added into a new and wider set of IT applications, combining physical as well as logical systems. This combination will lead to a new class of systems.

The IEEE recognises this in the document Towards a definition of the Internet of Things (IoT) (IEEE, 2015) and comes up with two definitions, depending on the complexity of the system.

- **A low complexity IoT system** is an IoT is a network that connects uniquely identifiable “Things” to the Internet. The “Things” have sensing/actuation and potential programmability capabilities. Through the exploitation of unique identification and sensing, information about the “Thing” can be collected and the state of the ‘Thing’ can be changed from anywhere, anytime, by anything.
- **A high complexity IoT system** is Internet of Things envisions a self-configuring, adaptive, complex network that interconnects ’things’ to the Internet through the use of standard communication protocols. The interconnected things have physical or virtual representation in the digital world, sensing/actuation capability, a programmability feature and are uniquely identifiable. The representation contains information including the thing’s identity, status, location or any other business, social or privately relevant information. The things offer services, with or without human intervention, through the exploitation of unique identification, data capture and communication, and actuation capability. The service is exploited through the use of intelligent interfaces and is made available anywhere, anytime, and for anything taking security into consideration.

These kind of definitions enable forecasts like those made by Deloitte about the potential business value of connecting gadgets and utilizing the resulting services (Newmarch, 2017). Beyond only reducing operational costs, the Internet of Things (IoT) has the ability to provide businesses with value. The possibility for providers in the IoT ecosystem to provide innovative IoT solutions that investigate how the capacity to gather and analyze diverse data in real-time and over time may revolutionize the business is largely untapped. These

changes will take place both within and across businesses, providing chances for long-term value creation and even disruption for those who can think outside the box.

History The Internet of Things

The concept of the Internet of Things (IoT) is not new. Since the invention of ubiquitous computing by Mark Wiesner in 1991, it has gone by a number of different names. Although Kevin Ashton is credited with coining the phrase "Internet of Things" in 1999, other terms with similar meanings include "the Web of Things," "the Internet of Everything," and — in a related field — "Cyber-Physical Systems."

The idea of the IoT as a whole has been around for 25 years, and throughout that period several academic, theoretical, and experimental systems have been developed. Through a distributed technology from Sun Microsystems called Jini, I got active in this field in 1999. This was first marketed as being for internet-connected devices, but it never worked on the little J2ME devices that Sun was also pushing and ended up being most effective for business-level systems.

However, the Internet of Things has just become one of the most prominent developing technologies - after 25 years, of course! This was likely sparked by a number of elements coming together. I would guess at:

- The almost universal penetration of the internet into many countries
- The growth of wireless communications
- Smart phones, and the apps that run on them, able to take advantage of features such as location
- Increasing knowledge and growth in sensors and sensor networks
- Cloud computing and the ability to utilise services managed and located elsewhere
- The growth of big data and the tools to deal with it
- Increasing standardisation of middleware, allowing easier building of complex services

I would specifically mention the Arduino and the Raspberry Pi (RPi) as emerging technology pioneers. The Raspberry Pi is assisting in bringing on the next generation of programmers while the Arduino is exposing micro-controllers to a new generation of electrical engineers. Along with that, it also re-engages those of us who are older with technology.

Of course, the RPi and Arduino are not the Internet of Things by themselves. They are merely the beginning, allowing everyone to explore with home automation systems, web development, and other technologies. The IoT consists of the integration of the following technologies and systems:

- Sensors and actuators
- Micro-controllers
- Micro-computers and all other computers
- Wireless and wired networks
- Network protocols such as TCP/IP
- Network applications such as web services
- Standardised representations of data, devices and services
- Processing engines and systems for big data
- Security

- Artificial intelligence

These are all examples of engineering and technology. Without financial incentives, the IoT would not be being pushed so hard. There must be prospects for business. It's obvious that the biggest IT corporations in the world think they exist.

Architecture of Internet of Things (IoT)

There are many uses for Internet of Things (IoT) technology, and its use is accelerating quickly. The Internet of Things functions as it was intended or developed to in accordance with the various application areas for which it has been used. However, it lacks a universally adhered-to standard defined architecture of working. The functioning and application of IoT in various domains determine its architecture. However, there is a fundamental process flow upon which IoT is founded.

So, in this essay, we'll talk about the core IoT architecture 4 Stage IoT.

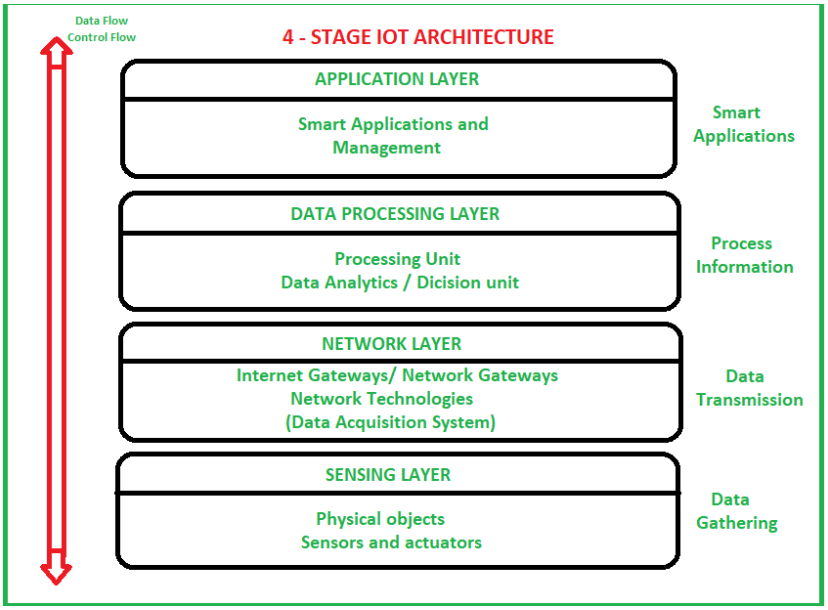


Figure 1 4 Stage IoT architecture

Therefore, it is evident from the image above that there are four levels present, which may be further classified into the following categories: sensing layer, network layer, data processing layer, and application layer.

These are explained as following below.

- **Sensing Layer** Sensors, actuators, devices are present in this Sensing layer. These Sensors or Actuators accepts data (physical/environmental parameters), processes data and emits data over network.
- **Network Layer** Internet/Network gateways, Data Acquisition System (DAS) are present in this layer. DAS performs data aggregation and conversion function (Collecting data and aggregating data then converting analog data of sensors to digital data etc). Advanced gateways which mainly opens up connection between Sensor networks and Internet also performs many basic gateway functionalities

like malware protection, and filtering also some times decision making based on inputted data and data management services, etc.

- **Data processing Layer** This is processing unit of IoT ecosystem. Here data is analyzed and pre-processed before sending it to data center from where data is accessed by software applications often termed as business applications where data is monitored and managed and further actions are also prepared. So here Edge IT or edge analytics comes into picture.
- **Application Layer** This is last layer of 4 stages of IoT architecture. Data centers or cloud is management stage of data where data is managed and is used by end-user applications like agriculture, health care, aerospace, farming, defense, etc.

Chapter 2: Microprocessor and Microcontroller

Microprocessor

A microprocessor is a type of computer processor where the logic and control for data processing are housed on a single integrated circuit or a few interconnected integrated circuits. The arithmetic, logic, and control circuitry needed to carry out the tasks of a computer's central processing unit are all included within the microprocessor. The integrated circuit has the ability to understand, carry out, and perform arithmetic operations. The microprocessor is a multifunctional, clock-driven, register-based, digital integrated circuit. It receives binary data as input, processes it in accordance with instructions stored in its memory, and outputs the results (also in binary form). Combinational and sequential digital logic are both present in microprocessors, which use the binary number system to represent numbers and symbols.

A microprocessor is an integrated circuit that performs most or all of the duties of a computer's Central Processing Unit (CPU) (IC).

A microprocessor is a type of computer processor where the logic and control for data processing are housed on a single integrated circuit or a few interconnected integrated circuits. The arithmetic, logic, and control circuitry needed to carry out the tasks of a computer's central processing unit are all included within the microprocessor. The integrated circuit has the ability to understand, carry out, and perform arithmetic operations. The microprocessor is a multifunctional, clock-driven, register-based, digital integrated circuit. It receives binary data as input, processes it in accordance with instructions stored in its memory, and outputs the results (also in binary form). Combinational and sequential digital logic are both present in microprocessors, which use the binary number system to represent numbers and symbols.

A microprocessor includes most or all of the functions of a computer's Central Processing Unit (CPU) on a single integrated circuit (IC).

Processing power costs were significantly lowered by Very-Large-Scale Integration (VLSI), which incorporated an entire CPU onto one or more integrated circuits. Highly automated metal-oxide-semiconductor (MOS) fabrication technologies are used to manufacture integrated circuit processors in huge quantities at a relatively cheap unit cost. Because there are significantly fewer potential points of failure for electrical connections, single-chip CPUs are more reliable. According to Rock's law, the cost of producing a chip (with smaller components integrated on a semiconductor chip of the same size) typically stays the same as microprocessor designs advance.

History of Microprocessor

Vacuum lamps

Vacuum tubes were used to construct the earliest computers. Compared to the preceding generation of mechanical computers, electronic computers were faster at performing calculations, but they were less reliable. The bulbs were the main culprits for the low reliability because they frequently overheated, which led to frequent computer crashes.

The first generation of computers was launched with the 1946 completion of the ENIAC light computer. The user (operator) programmed ENIAC, but the procedure was physically demanding, time-consuming, and complicated because it involved flipping switches or manipulating wires. The mathematician John von Neumann developed the concept of the stored program to simplify the programming process. In 1945, Von Neumann submitted a research in which he suggested building the EDVAC, a novel computer that would run stored programs. By merely changing the contents of memory, it would be simpler to create and modify programs.

When EDVAC was finished in August 1949, it could carry out a certain set of instructions. The user assembled a program by mixing the commands, saved it in memory, and then waited for the computer to run it. The von Neumann architecture was given to the EDVAC prototype design in honor of its creator and inventor and marked a significant reduction in the amount of programming time needed. Other scientists, like Alan Turing and Konrad Chouse, devised methods very similar to these. The Harvard University Mark I computer is noteworthy because it was finished before EDVAC and featured a program storage mechanism that used punched tape rather than electrical memory. The Harvard architecture, which emerged from the Mark I, differs significantly from the von Neumann architecture in that it separates data storage from instruction.

Transistor

Governments have always financed the development of computers as a topic of study in universities. With the invention of the transistor, which had none of the drawbacks of lights, the first significant advancement was made. Transistor computers outperformed tube computers in terms of calculating speed, size, and resistance to overheating. Transistors drastically decreased how much electricity computers used to consume. At all levels, the new computers outperformed the older ones, which they had totally supplanted by the early 1960s. While certain computers in the second generation could be programmed using symbolic language, the second generation was created entirely with transistors.

Private enterprises became interested in computers and processors because of the new advantages. Large sums of money were invested in automating the tasks in businesses with an eye toward the production of computers for commercial use. Better data input and output devices and speedier memory were integrated into computers during design and construction. Transistors were made so small that they could hardly be seen with the naked eye in response to the growing demand for greater computation, and they eventually

started to be integrated into boards. The boards, which serve different purposes, were made separately and then put together. The CPU was developed separately from the rest of the computer due to the individual development of the pieces, which when placed together created a computer.

Integrated circuits

An integrated circuit, or simply integrated, is a sheet-based circuit made up of connected logic gates. Integrated circuits are primarily made on semiconductor wafers, which account for the great majority of them.

The integrated circuit (IC), often known as a "chip," carries out an integrated process in that it accepts input and outputs outcomes. It is made from a tiny piece of silicon that, since it is a semiconductor, can combine the functions of conductors, resistors, and transistors so they no longer need to be produced separately. On a thin silicon surface, many transistors may be produced with ease, while plating is used to build circuits. Initially, integrated circuits were limited to simple circuits like NOR gates. Simple integrated circuit-based processors are categorized as small scale integration devices.

Later, identical circuits were printed on larger surfaces and sliced into separate chips. Logic gates, memory cells, and data input and output points were all present on every chip. Each chip was sliced, and then pins were added to allow for connection to other hardware. Larger and more intricate circuits were produced when numerous chips were joined on a board. Processors for medium- and large-scale integrated circuits were created in this manner. Microelectronics advancements boosted the number of transistors on integrated circuits while lowering the overall number of integrated circuits needed to produce a CPU.

The IBM System/360 computers serve as a good example of the third generation of CPUs. The business made an effort to eliminate the incompatibility that existed across computers, even those made by the same manufacturer, so that a software may run unaltered by other machines with varying speeds and capacities. IBM introduced the idea of a microprogram to accomplish this. The complete System/360 architecture gained so much traction that it dominated mainframes for many years and is being used in comparable ways in contemporary computers today.

Microprocessors

Small computers were formerly constructed utilizing racks of circuit boards that contained numerous small- and medium-sized integrated circuits. This was merged by microprocessors into one or more big ICs. The Intel 4004 was unveiled in 1971 and was the first microprocessor to be made publicly available.

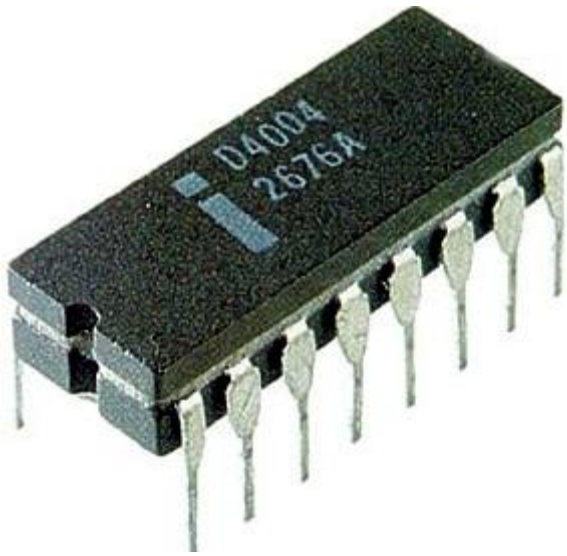


Figure 2 The Intel 4004, the first of the microprocessors built. Production date: 1971 to 1981.

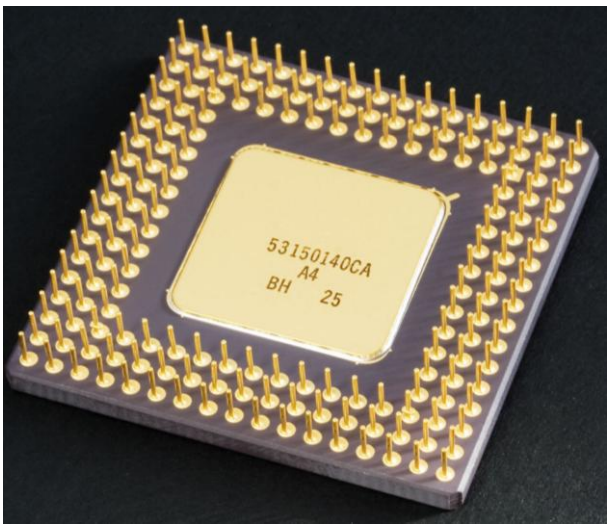


Figure 3 The Intel 80486DX2 microprocessor.

When the first processors were created from a single, highly integrated integrated circuit in the 1970s, the way CPUs are constructed underwent a substantial transformation. The new processors were known as microprocessors because of the size reduction, but currently the word "CPU" only applies to them. Due to physical considerations, the lower size also decreased changeover time. As a result, current microprocessors have clock rates that range from several gigahertz to hundreds of megahertz. An integrated circuit became more intricate and had more transistors at the same time. Moore's Law, which still holds true today and forecasts the doubling of the number of transistors integrated into an integrated circuit every 18 months, describes the growth rate of transistors.

It is amazing how little their fundamental architecture and function have changed over the past 60 years, despite the fact that the complexity, size, structure, and overall look of processors have changed significantly. Today, von Neumann computers may be considered of as practically all conventional CPUs. Concerns have been expressed concerning the limitations of transistor integrated circuit technology as Moore's Law continues to hold true. Electronic gates no longer have the issues that they did in the past because of the materials utilized in construction. However, more recent issues are motivating scientists to investigate novel computing techniques like the quantum computer as well as to increase the usage of parallel computing and other strategies that increase the functionality of the existing von Neumann architecture.

Structure of Microprocessor

A modern microprocessor consists of the following units.

- **Decoding Unit** (Decoding Unit).
- **Arithmetic and Logical Unit** (ALU): The unit in which the arithmetic or logical operations are performed one by one, as dictated by the instructions given to the computer.

- **Registers**: Small memory cells inside the processor, used to temporarily store data as it is being processed. Registers vary by processor type and manufacturer, both in terms of organization and capacity.
- **Control Unit**: Controls the flow of data to and from the ALU, registers, memory, and peripheral I/O units.
- **Fetch Unit**: Transfers the commands from memory to the processor.
- **Protection Unit**: Ensures the acceptance of each process performed by the processor, so that data that should not be modified or unacceptable commands are not executed, such as e.g. dividing a number by zero.

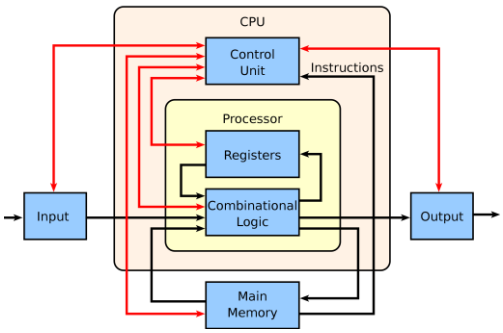


Figure 4 Block diagram of a basic uniprocessor-CPU computer. Black lines indicate data flow, whereas red lines indicate control flow; arrows indicate flow directions.

An arithmetic logic unit (ALU) and a control logic section may be the only components of a simple hypothetical microprocessor. The addition, subtraction, and operations like AND and OR are all performed by the ALU. Each ALU operation creates one or more flags in a status register that represent the outcomes of the previous operation (zero value, negative number, overflow, or others). The control logic starts the series of operations needed for the ALU to carry out the instruction by retrieving the instruction codes from memory. A single operation code may have an impact on a variety of distinct data channels, registers, and processor components.

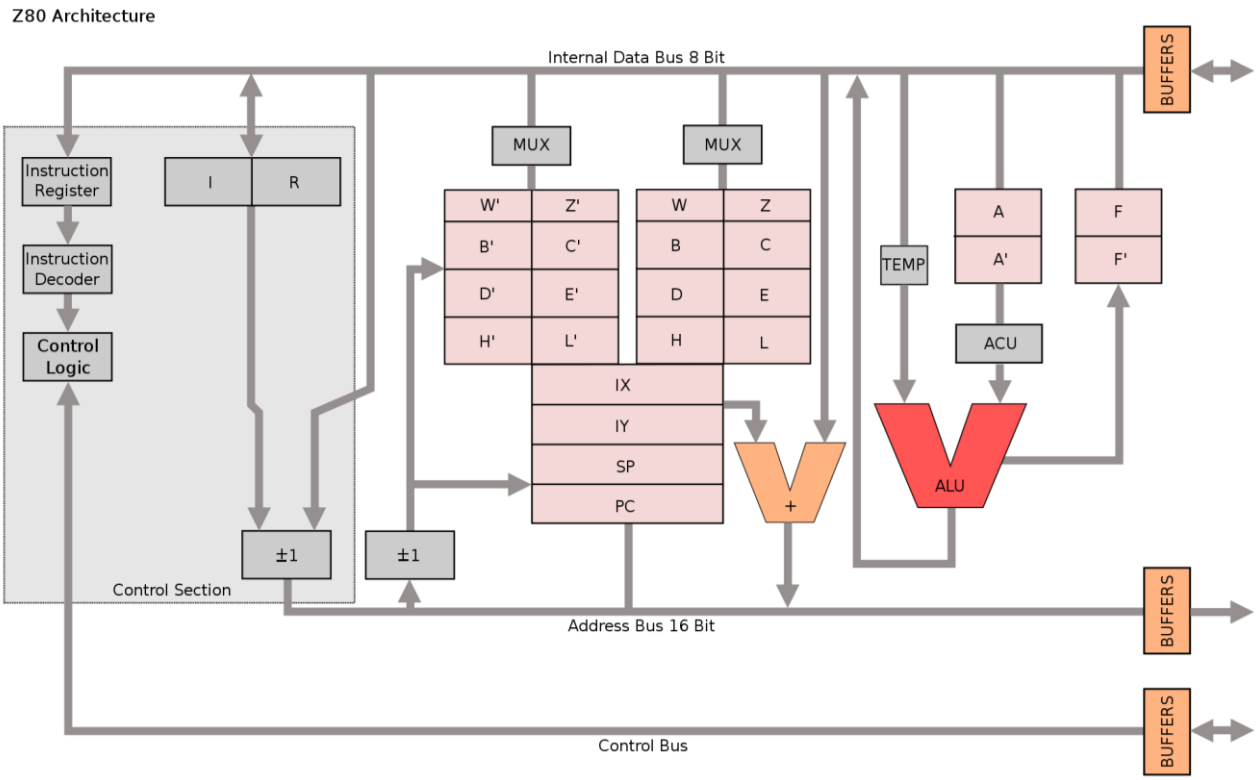


Figure 5 A block diagram of the architecture of the Z80 microprocessor, showing the arithmetic and logic section, register file, control logic section, and buffers to external address and data lines

As the field of integrated circuits developed, it became possible to produce increasingly sophisticated processors on a single chip. The number of transistors available on a device allowed word sizes to expand from 4- and 8-bit words up to today's 64-bit words, increasing the amount of data objects. The processor architecture was expanded; additional on-chip registers speed up programs and complicated instructions could be utilized to create smaller programs. For instance, 8-bit microprocessors frequently lacked the ability to do floating-point arithmetic, which had to be done in software instead. Calculations using floating-point numbers were accelerated by integrating the floating-point unit, initially as a separate integrated circuit and then as a component of the same microprocessor chip.

Additionally, many circuits might process portions of each word in parallel rather than processing the entire length of a large word on a single integrated circuit. The end result was a system that could handle, for example, 32-bit words using integrated circuits with a capacity for just four bits apiece, even if this needed additional circuitry to manage, for example, carry and overflow inside each slice.

It is possible to incorporate memory on the same die as the CPU because to the capacity to pack many transistors onto a single chip. This CPU cache boosts the system's processing performance for many applications and provides the benefit of faster access than off-chip memory. Cache memory is required if the processor is not to be delayed by slower external memory since processor clock frequency has grown faster than external memory speed.

Commands

Microprocessors, by their physical form, execute a number or sequence of numbers corresponding to an action/instruction.

Operation

An instruction can process data or change states inside the CPU. If it is a processing command it can:

- carry,
- to add,
- to compare the data.

Additionally, the program counter and pointer register may be modified using the instruction, and the program flow can be altered by altering the program counter. These "jumps" (jumps) are the fundamental building blocks of select and repeat loop implementation. Different check states, like the overflow pointer, are present in the pointer register. Since pointers frequently refer to the outcomes of different actions, they have an impact on how a program operates.

Each instruction includes the operation to be performed, the source of the data, and the address of the next instruction. Depending on the kind of command, every information is encoded in one or more than. We agree that the following instruction comes right after the present instruction in order to create the fewest number of bytes feasible. In general, instructions are divided into two parts: the first part, which contains the function code and specifies the operation to be carried out, and the second, which contains the information needed to carry out the instruction, such as the executors for the addition operation or the address of the next instruction if the instruction is a branch.

For each command there is a different encoding. Even in the same instruction, depending on the type of operands it uses, i.e. whether they are registers, memory locations or private data, there is a different encoding. Actually all commands are binary numbers assigned to a function. To make them easier to understand, commands are usually represented either in hexadecimal or in symbolic language. For example, the ADD C instruction states that the contents of the C register will be added with the contents of the accumulator and the result stored in the accumulator. The corresponding hexadecimal representation of this command is 81h, on x86 architectures.

Command cycle

The length of time between starting one command and finishing it is known as the command cycle. There are four stages to completing a cycle:

1. The fetch,
2. The decoding (decode),
3. The execution (execute)
4. -and The saving of the result (store/writeback).

1. *Fetch*

When called upon, the command is taken from a stored memory location. The program counter holds the information about where the instruction is located in memory. The instruction register is where it is stored during the instruction transfer from memory to the processor. The program counter value is then numbered, along with the instruction length in units, to show the location of the following instruction or, if the current instruction has executors, the address of the operands. Because the two devices frequently operate asynchronously, the CPU frequently stalls because the instruction that has to be remembered is delayed in being sent from memory to the processor. Modern CPUs use cache and pipelining techniques to solve this issue.

2. *Decode*

The command is disassembled and evaluated by processing in the decoding phase. During decoding, any operators are also recalled based on the command code. According to the relevant addressing standard, the value of the operators is either directly recalled as a constant or indirectly as an address where the value is kept in some register or memory. Instruction decoding used to be an immutable hardware task in previous processing systems. However, a microprogram is used to interpret the instructions in more advanced CPU architectures. Even after the CPU has been constructed, the firmware typically can be altered because it decodes the instructions.

3. *Execute*

The command is carried out after being remembered and decoded. Currently, the processor's numerous units are coupled in order to perform the intended function. A set of inputs and outputs would be connected to the arithmetic unit (AU) if, for instance, an addition operation was required. The numbers to be added are provided in the inputs, and the sum is contained in the outputs. The numeric overflow flag will be set if the sum produces a number that is too large for the CPU to process. For executing basic arithmetic and logic operations, such adding and comparing integers, the Arithmetic Logic Unit (ALU) as a whole incorporates circuits.

4. *The saving of the result*

The CPU sends the data to be saved in memory during the last stage, storage. For quicker access by following commands, the results are initially temporarily saved in a register before being stored in the system's main memory. Jumping instructions and instructions that modify the pointer register don't actually generate any output that may be saved. The instruction cycle is finished and then repeated with the next instruction once the program counter has been increased. A greater number of instructions may be recalled, decoded, and executed simultaneously in computers with more complicated architecture.

Data representation

How data is represented is one of the most crucial difficulties with computing systems. Many early digital computers represented numbers using the decimal system, while others utilized the ternary system. The

binary system, which uses the two numbers zero (0) and one (1) to represent two opposing physical states like "high" and "low," voltage, is used by almost all current CPUs. Computers can only interpret these two digits, so some sort of code must be utilized to give these bits a meaning. The I/O unit encodes and decodes the bits to represent a user-understandable number or character. Bytes and sentences may occasionally be symbolically represented in hexadecimal format in order to shorten their length.

The CPU and by extension computers handle specific groups of bits. Groups can have

- one bit,
- four bits called a nibble,
- eight bits called a byte,
- and 16 bits called a word.

A byte is the most common group of binary digits used in computers. With one byte, 2^8 different values can be represented, since it has 8 bits. While the primary use of bytes is for character encoding, the bits in a byte are numbered from zero to seven. The bits of a word are numbered from zero to 15 and can represent 2^{16}, or 65,536, different values. Each bit or byte stands for whatever we define it to. One bit, for instance, can be utilized as a number, and the bit next to it, a logical value. In the same way, depending on the encoding we select to read it in, a byte can be utilized as a number or a character. The same binary format is used for data on the system bus, in memory, and in the CPU.

Data processing

The processing of data in a microprocessor depends on two (2) more features, its System Bus and its Clock Rate.

System bus

Bus conductors are organized into groups based on what they do. A bus that has 32 bits, for instance, has 32 different conductors, each of which transmits one bit of data. In this sense, the system bus is made up of a number of distinct buses that are organized according to their purpose. These buses are: Address Bus, the Data Bus and the Control Bus.

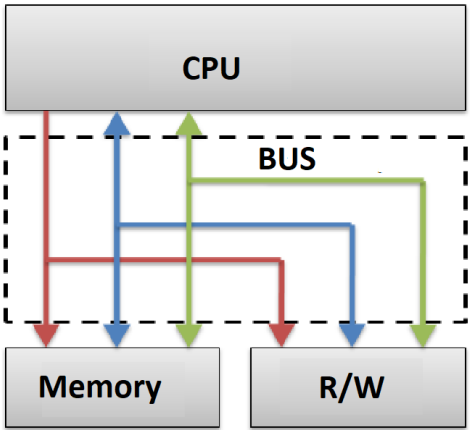


Figure 6 **Red: Address Bus**, **Blue: Data Bus**, **Green: Control Bus**.

- The Address Bus contains the address of the memory location where the data will be stored. The size of the address bus also determines the size of the memory that the processor can address, that is, the memory that it can perceive and use. For example in the 8088, where the address bus was 20 bits, the processor could access up to 2^{20} = 1,048,576 memory locations (1MB).
- The Data Bus transfers data between the units of the computer system. The amount of bits it can transport at once and the range of numbers the processor can handle depend on its size. The Intel 8088 processor, with an 8-bit data bus, is categorized as an eight-bit processor and can handle 2^8 = 256 numbers.
- The Control Bus consists of pipes with a separate function each, which control how the processor communicates with the rest of the subsystems. For instance, the Read or Write signals on the control bus dictate the direction of the data when the CPU talks with the memory.

Clock Rate

The majority of CPUs are synchronous circuits, which means that they initiate sequential processes using a clock signal. An external oscillator circuit that creates a defined number of pulses in the form of a periodic square wave per second is what creates the clock signal. The pace at which a CPU executes instructions is determined by the clock frequency; as a result, the quicker the clock, the more instructions the CPU will execute per second.

In other words, they are built to function in tandem with an electrical synchronization signal known as a clock signal. The control bus emits an electrical square pulse that serves as the clock, cycling between zero and one on a regular basis.

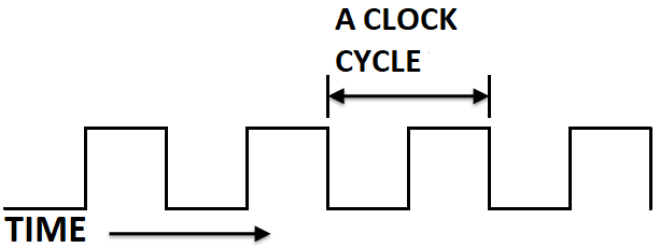


Figure 7 The pulse of the system clock.

The clock's period or cycle is the amount of time it takes to go from 0 to 1 and back again. The switching frequency used in this process is known as the clock frequency. The inverse of frequency, called cycle, is measured in seconds, while frequency is recorded in Hertz (Hz). The smallest period that an operation can take place is a clock cycle. When compared to other operations, some can be completed in a single clock cycle.

The clock period is longer than the maximum amount of time needed to propagate (move) all signals through the CPU in order to guarantee proper CPU operation. It is feasible to construct the entire CPU and how it transports data around the rising and falling "edges" of the clock signal by setting the clock period to a value that is significantly higher than the worst propagation delay. This offers the benefit of significantly simplifying the CPU's architecture and component count. Even though some sections of it are substantially faster, it also has the drawback of making the entire CPU wait on its slowest components. These restrictions have mainly been overcome by various techniques for enhancing CPU parallelism.

Although modern processors may operate at speeds of three to four gigahertz, parallel processing was the key to achieving the true boost in computing capability. Manufacturers were able to boost performance while lowering the clock frequency by using two CPUs in a computer system.

The problems we face today are:

- A clock signal is subject to the delays of any other electrical signal. Higher clock rates on increasingly complex CPUs make it more difficult to keep the clock signal in phase (synchronized) across the unit. This has led many modern CPUs to require multiple identical clock signals to be provided to prevent a single signal from being delayed enough to cause the CPU to malfunction.
- As clock rates increase dramatically, so does the amount of heat dissipated by the CPU.
- Continuously changing clock causes many items to toggle regardless of whether they are in use at the time. In general, a switching component uses more energy than a static component. Therefore, as the clock rate increases, so does the power consumption, causing the CPU to require more heat dissipation in the form of CPU cooling solutions.

CPU ARM

ARM (ARM architecture family, n.d.) is a 32-bit RISC instruction set architecture developed by ARM Holdings. The initials stand for Advanced RISC Machine, while they formerly stood for Acorn RISC Machine.

In terms of the number of processors created, the ARM architecture is the most widely utilized 32-bit instruction set architecture.

The first products based on the ARM architecture were the Acorn Archimedes series, which debuted in 1987. Acorn Computers originally developed the architecture for use in their personal computers.

Because they are so straightforward, ARM processors are appropriate for low-power applications. As a result, they are the most popular compact and reasonably priced microprocessors and microcontrollers in the embedded and mobile systems industries. Over one billion mobile phones were shipped annually in 2005, and 98 percent of those included at least one ARM CPU. A large percentage of consumer electronics products, such as personal digital assistants (PDAs), mobile phones, digital music players, multimedia devices, portable video game consoles, calculators, and computer peripherals like hard drives and routers, use ARM processors, which in 2009 made up about 90% of all 32-bit RISC embedded processors.

The ARM architecture is licensed to anyone who wants to use it.

Major ARM processor families developed by ARM Holdings are ARM7, ARM9, ARM11 and Cortex.

Architecture ARM

From 1995 to the present, the ARM Architecture Reference Manual (ARM, n.d.) is the primary source of documentation on the ARM processor architecture and its instruction set, distinguishing between interfaces that all processors must support (such as the meaning of instructions) and implementation details, which may vary from case to case. The architecture has evolved and, starting with the Cortex series of cores, three "profiles" are defined:

- "Application": Cortex-A series
- "Real-time": Cortex-R series
- "Microcontroller": Cortex-M series

A subset of the architecture may be the profiles. For instance, the Cortex-M3 core's ARMv7-M profile only supports the Thumb-2 instruction set, and the Cortex-ARMv6-M M0's profile is a subset of the ARMv7-M profile (it supports fewer commands).

Figure 8 An ARMv7 was used to power older versions of the popular Raspberry Pi single-board computers like this Raspberry Pi 2 from 2015.

5. Instruction set

Like the much more straightforward 8-bit 6502 processor used in earlier Acorn microcomputers, the initial (and subsequent) ARM implementation was hardwired without microcode. The 32-bit ARM architecture (and the 64-bit architecture for the most part) includes the following RISC features:

- Load/store architecture.
- No support for unaligned memory accesses in the original version of the architecture. ARMv6 and later, except some microcontroller versions, support unaligned accesses for half-word and single-word load/store instructions with some limitations, such as no guaranteed atomicity.
- Uniform 16 × 32-bit register file (including the program counter, stack pointer and the link register).
- Fixed instruction width of 32 bits to ease decoding and pipelining, at the cost of decreased code density. Later, the Thumb instruction set added 16-bit instructions and increased code density.
- Mostly single clock-cycle execution.

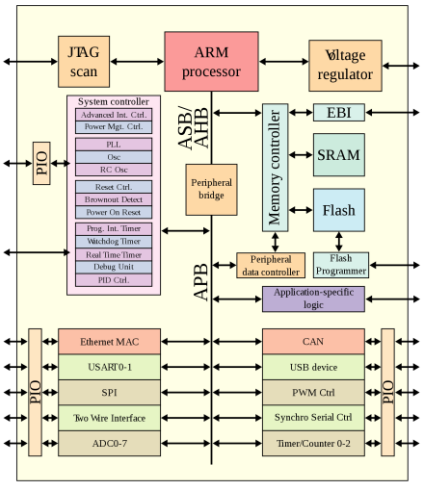


Figure 9 Microprocessor-based system on a chip

To compensate for the simpler design, compared with processors like the Intel 80286 and Motorola 68020, some additional design features were used:

- Conditional execution of most instructions reduces branch overhead and compensates for the lack of a branch predictor in early chips.
- Arithmetic instructions alter condition codes only when desired.
- 32-bit barrel shifter can be used without performance penalty with most arithmetic instructions and address calculations.
- Has powerful indexed addressing modes.
- A link register supports fast leaf function calls.
- A simple, but fast, 2-priority-level interrupt subsystem has switched register banks.

Microcontroller

Due to the numerous embedded subsystems it has, a microcontroller is a sort of processor, essentially a microprocessor version, that can function with few external components. It is extensively utilized in all low- and medium-cost embedded control systems, including those in self-propelled wheeled vehicles of all kinds, consumer electronics (from digital cameras to toys), and automation.

History Microcontroller

A microcontroller, often known as an MCU or a "C," is a task-specific computer system on a chip. It has an inbuilt CPU, memory (some RAM, program memory, or both), and programmable input/output peripherals for interacting with the electronic devices linked to the chip. A microcontroller is different than a microprocessor, which only contains a CPU (the kind used in a Personal Computer).

The first multi-chip microprocessors, the Four-Phase Systems AL1 in 1969 and the Garrett AiResearch MP944 in 1970, were developed with multiple MOS LSI chips. The first single-chip microprocessor was the Intel 4004, released on a single MOS LSI chip in 1971.

It was followed by the 4-bit Intel 4040, the 8-bit Intel 8008, and the 8-bit Intel 8080. All of these processors required several external chips to implement a working system, including memory and peripheral interface chips. As a result, the total system cost was several hundred (1970s US) dollars, making it impossible to economically computerize small appliances.

After people began to see how useful they were, micro controllers were constantly being upgraded, with people trying to find new ways to make them better. Cost was reduced over time and by the early 2000s, microcontrollers were widely used across the world.

During the early-to-mid-1970s, Japanese electronics manufacturers began producing microcontrollers for automobiles, including 4-bit MCUs for in-car entertainment, automatic wipers, electronic locks, and dashboard, and 8-bit MCUs for engine control.

Partly in response, Intel, developed a computer system on a chip optimized for control applications, the Intel 8048 (in 1977), it combined RAM and ROM on the same chip with a microprocessor.
Microcontrollers had different variants:

- One had EPROM program memory, with a transparent quartz window in the lid of the package to allow it to be erased by exposure to ultraviolet light. These erasable chips were often used for prototyping.
- The other variant was either a mask programmed ROM or a PROM variant which was only programmable once. For the latter, sometimes the designation OTP was used, standing for "one-time programmable".
- In 1993, the introduction of EEPROM memory allowed microcontrollers to be electrically erased quickly without an expensive package as required for EPROM, allowing both rapid prototyping, and in-system programming.
- The same year, Atmel introduced the first microcontroller using Flash memory, a special type of EEPROM.

Microcontrollers, like other integrated circuits, are created on unique masks and then exposed to ultraviolet radiation, which produces unique traces (often composed of silicon) that make up the various components of the integrated circuit. The technology used in contemporary microcontrollers may create silicon traces as thin as 2 m.

The microcontroller has additional components such as RAM for data storage, read-only memory for program storage, flash memory for permanent data storage, and other devices in addition to the typical arithmetic and logic components of a standard microprocessor (peripherals).

Microcontrollers often operate at very low speed compared to microprocessors used in smartphones and personal computers. Clock speeds may be as little as 32 kHz, but this is useful for typical applications. They consume very little power (milliwatts or even micro watts).

After people began to see how useful they were, micro controllers were constantly being upgraded, with people trying to find new ways to make them better. Cost was reduced over time and by the early 2000s, microcontrollers were widely used across the world.

Nowadays microcontrollers are cheap and readily available for hobbyists, with large online communities around certain processors.

Microcontrollers are used in automatic products and devices, such as car engine systems, remote controls, machines, appliances, power tools, and toys. These are called embedded systems. Microcontrollers can also be found at work in solar power and energy harvesting, anti-lock braking systems in cars, and have many uses in the medical field as well.

Common subsystems

The logical and arithmetic unit (ALU), elementary registers (registers), very high-speed RAM temporary memory (cache memory), and occasionally the memory controller are the only components of the microprocessor's integrated circuit (memory controller). But numerous external peripherals and subsystems are necessary for an entire embedded computing system to function. Such are:

- Glue logic circuit for connecting external memories and other peripherals of parallel connection to the data bus (bus) of the processor.
- Program memory (ROM, FLASH, EPROM, etc.) which contains the system software. In some models, it is possible to lock this memory after it has been written to protect its contents from copying.
- Large RAM size.
- Permanent memory for storing operating parameters (of the EEPROM or NVRAM type) which can be written to the microcontroller core. This memory has, over FLASH, the advantage of being able to erase and write any single byte.
- Initialization circuit (reset).
- Interrupt request controller (interrupt request controller) from peripherals.
- Brown-out detection circuit that monitors the power supply and initializes the entire system when it falls below the tolerable limits, thus preventing data corruption.
- Watchdog timer which initializes the system if it shows signs of malfunction due to a hang.
- Local oscillator to provide clock pulses.
- One or more high-speed hardware timer-counters for generating delays, event durations, event enumeration, and other precision timing functions.
- Real Time Clock (RTC) which is powered by an independent battery and therefore must have very low power consumption.
- Series of independent digital inputs and outputs (Parallel Input-Output, PIO).

Most of the aforementioned peripherals are generally included into all microcontroller families, with the primary distinctions being the type of internal program memory and whether it is present or not. Thus, there are:

- Microcontrollers without program memory, which are characterized as ROM-less. These always provide a parallel data bus, onto which external program memories and RAM are connected. Such types of microcontrollers are intended for more powerful computing control systems, with greater memory requirements.
- Microcontrollers with ROM memory, which is built with its software (Mask ROM) or written only once (One Time Programmable, OTP). They provide the possibility of very low cost when purchased in very large quantities.
- Microcontrollers with FLASH memory, which can usually be programmed multiple times. This is the most common category. Often the memory programming can even be done on the circuit of the embedded application itself (In Circuit Programming, ISP capability). These microcontrollers have

essentially replaced the older types of EPROMs that were erased by UV radiation (from the special glass).

Types of Microcontroller

The microcontroller industry has ended up creating both competitive mass-produced models and microcontrollers for more specialized applications as a result of intense rivalry and the tendency of incorporating microcontrollers into every electrical and electronic equipment. Thus, the following main categories are distinguished:

- Microcontrollers (sometimes 4-bit but usually 8-bit) very low-cost, general-purpose, very small number of pins (even less than 8). They are designed with an emphasis on low power consumption and self-sufficiency, so that little or no external components are needed and their internal software cannot be easily copied. The ability to expand their memory is absent. Some models are well known to electronic hobbyists, such as most microcontrollers of the PIC (Microchip), AVR (Atmel) and 8051 (Intel, Atmel, Dallas, etc.) series.
- Microcontrollers (typically 8-bit but also 16 or 32-bit) low-cost, general-purpose, with a moderate to relatively large number of pins. They have a large number of common peripherals, such as UART, I2C, SPI or CAN ports, analog-to-digital and digital-to-analog converters. In Far Eastern manufacturers (Japan, Korea), it is customary to integrate liquid crystal display controllers and a keyboard. Sometimes they provide external expansion of their memory.
- Microcontrollers (mainly 32-bit) of medium cost, general purpose, with a large number of pins. They are characterized by an emphasis on instruction execution speed, high peripheral self-sufficiency and large internal or external program memory (FLASH) and RAM capabilities. Architectures with high software portability from one manufacturer to another have a strong presence in this area. For example, between ARM or MIPS microcontrollers, the set of basic instructions recognized by the ALU is exactly the same, thus reducing major software changes when in the future the customer adopts a microcontroller from another manufacturer (as long as, of course, it also supports the ARM or MIPS instruction set, respectively).
- Microcontrollers for specialized applications, which usually incorporate some specialized communication protocol which is always implemented in hardware. Such microcontrollers are used in telecommunication devices such as modems.

A large portion of microcontroller sales are still 8-bit, as it is the class with the lowest cost and smallest software size for the same output, especially since modern 8-bit microcontroller families have much improved performance over past.

Some of the best known microcontroller manufacturers are

- ARM (does not manufacture but grants rights to use the kernel)
- Atmel
- Epson
- Freescale Semiconductor (formerly Motorola)
- Hitachi

- Maxim (after acquisition of Dallas)
- Microchip
- NEC
- Toshiba
- Texas Instruments

There are a huge number of microcontrollers. The Element14 (Element14, n.d.) site lists over 7,000 microcontrollers for sale, in 8-, 16- or 32-bit forms while Wikipedia (Wikipedia, n.d.) has a long list of common microcontrollers, by vendor.

Features of Microcontroller

Depending on the application for which a microcontroller is intended, it may also contain:

- One or more asynchronous serial communication ports (Universal Asynchronous Receiver Transmitter, UART).
- Modern serial communication ports (eg I2C, SPI, Ethernet).
- Complete subsystems for direct firmware support of the most complex communication protocols such as CAN, HDLC, ISDN, ADSL.
- Floating Point Processing Unit (FPU), which is always faster than the processor's ALU. Such units characterize microcontrollers with digital signal processing (DSP) capabilities. In recent years, with the wide spread of portable audio and video devices, there has been a trend of convergence of microcontrollers with DSPs.
- More than one input for analog to digital conversion (Analog to Digital converter, ADC).
- Digital to Analog converter (DAC).
- Liquid Crystal Display (LCD) controller.
- On-circuit programming subsystem (ISP type, see above). Thanks to this circuit, it is possible to reprogram (software upgrade) the application by connecting to the device an external programming device (usually on a UART RS-232 port) or even from the internet. This feature requires the pre-existence of host software (bootstrap) in the program memory and therefore cannot be done in a completely empty program memory.
- Programming (ISP type) and diagnostics subsystem (usually the established JTAG standard). Thanks to this, it is possible to program the program memory without the need for a host program. For this reason, it is particularly useful in initial programming, for example during assembly, or in case of a bug in the host software which makes normal upgrading impossible.

Microprocessor Vs Microcontroller

For the Internet of Things to connect the physical and digital worlds, sensors and actuators are needed. A sensor transforms a physical condition into an analog or digital signal, while an actuator transforms a digital

signal into a physical effect. A computer system that produces or processes digital signals is the following stage in the IoT architecture.

These operations may be carried out by microprocessors and microcontrollers alike. Although the line separating microcontrollers and microprocessors is sometimes hazy, they are not the same. The following is an excerpt from Microcontroller vs. Microprocessor: Differences and Comparisons.

Microprocessors and microcontrollers have frequently been used interchangeably. They were both created with real-time use in mind. While they have a lot in common, they also differ greatly from one another. The microprocessor and microcontroller are both integrated circuits (ICs), yet their differences cannot be seen. Depending on the characteristics, they come in a variety of versions ranging from 6 pin to 80 to 100 pins or even higher.

Microprocessors are integrated circuits (ICs) that simply contain the CPU, or computing power, such as Intel's Pentium 1,2,3,4, core 2 duo, i3, and i5. These microprocessors lack on-chip peripherals including ROM, RAM, and other storage devices. To make them work, a system designer must include them outside. Microprocessor applications include desktop computers, laptops, notepads, and other devices.

However, this is not true with microcontrollers. A microcontroller is a single chip that has a CPU, a set amount of RAM, ROM, and other peripherals.

Microcontrollers are made to carry out particular jobs. Applications that are specific describe the connection between input and output. Depending on the input, processing is required before delivering the result. For instance, keyboards, mice, washing machines, digital cameras, pen drives, microwaves, mobile phones, watches, and so forth. Since the programs are relatively specialized, they may be incorporated on a single chip because they only require a tiny amount of resources like RAM, ROM, I/O ports, etc.

A single von Neumann- architecture (Von Neumann architecture, n.d.) memory arrangement is common in common microprocessors.

The von Neumann architecture — also known as the von Neumann model or Princeton architecture — is a computer architecture based on a 1945.

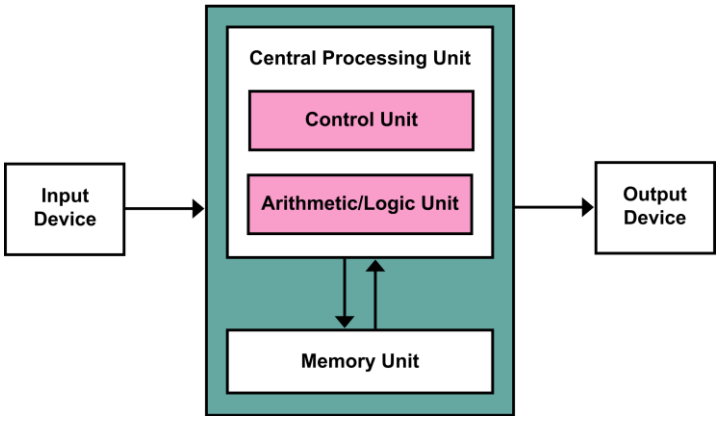


Figure 10 A von Neumann architecture scheme

A design architecture for an electronic digital computer with these components:

- A processing unit with both an arithmetic logic unit and processor registers
- A control unit that includes an instruction register and a program counter
- Memory that stores data and instructions
- External mass storage
- Input and output mechanisms

Microcontrollers have a similar fundamental design to conventional microprocessors, despite the latter's frequent usage of the Harvard architectural memory architecture, which employs various linkages between program memory and data memory.

The Harvard architecture (Harvard architecture, n.d.) is a computer architecture with separate storage and signal pathways for instructions and data. It contrasts with the von Neumann architecture, where program instructions and data share the same memory and pathways.

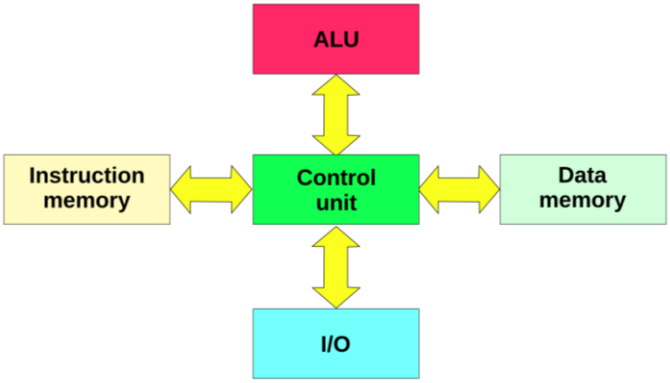


Figure 11 Harvard architecture

The Harvard Mark I relay-based computer, which stored data in electro-mechanical counters and instructions on punched tape (24 bits wide), gave rise to the phrase. These early computers did not allow access to the instruction storage as data and had all of their data storage within the central processor unit. An operator was required to load programs; the processor could not initialize itself.

A Harvard architecture machine, which is similarly a stored-program system but includes one specialized set of address and data buses for reading from and writing to memory and another set of address and data buses for fetching instructions, has a more complex design than a von Neumann architecture machine.

The user perceives contemporary processors as von Neumann machines since the program code and data are kept in the same main memory. The majority of designs include distinct processor caches for the instructions and data, with different paths into the processor, for performance reasons that are internal and mostly transparent to the user. This type of design is referred to as modified Harvard architecture.

What is the difference

The focus is on computational power in contemporary microprocessors for non-embedded systems (such as personal computer microprocessors). Since the functionality of the finished system is decided by the external peripherals attached to the core unit (microprocessor), which is not specialized, there is a lot of freedom in building diverse applications. In contrast, the flexibility and processing capacity are constrained in microprocessors for embedded systems (microcontrollers), which have fewer or no capabilities of collaboration with such external peripherals. Microcontrollers place a strong emphasis on specialization, low cost, and the limited amount of integrated circuits needed to run a device.

In detail, the advantages of microcontrollers are:

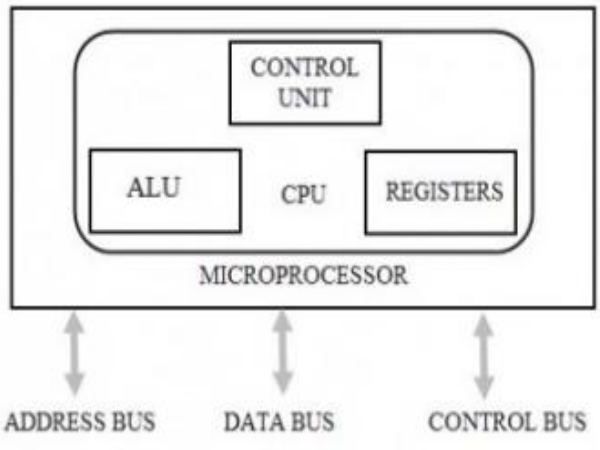
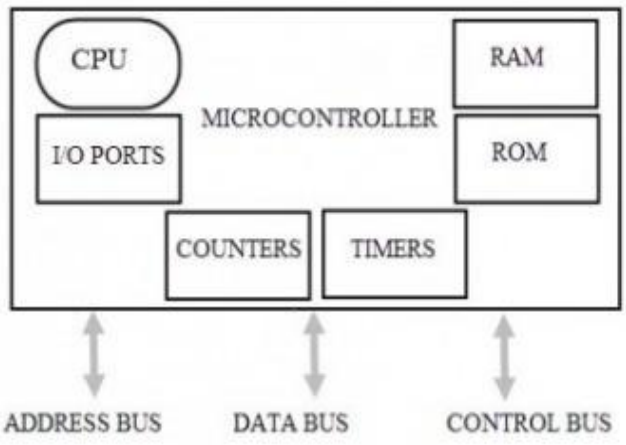
- Autonomy, through the integration of complex peripheral subsystems such as memories and communication ports. So many microcontrollers do not need any other integrated circuit to function.
- Peripheral integration means easier application implementation due to simpler interfaces. It also results in lower power consumption, maximizing portability and minimizing the cost of the device in which the microcontroller is embedded.
- Low cost.
- Greater reliability, again due to fewer interconnections.
- Reduced emissions of electromagnetic interference and reduced sensitivity to corresponding interference from other electrical and electronic devices. This advantage results from the smaller number and length of external connections as well as the lower operating speeds.
- More pins available for digital I/O (for a given IC size), due to not being dedicated to connecting external peripherals.
- Small overall computing system size.

Microcontrollers have a similar fundamental design to ordinary microprocessors, despite the latter's frequent usage of Harvard-style memory architecture, which employs distinct connecting arteries for program memory and data memory (eg the AVR series from Atmel and PIC from Microchip). In most popular microprocessors, a single von Neumann-type memory configuration is included.

Below is a list of the distinctions between microprocessors and microcontrollers.

	Microprocessor	Microcontroller
1	Microprocessor acts as a heart of computer system.	Microcontroller acts as a heart of embedded system.
2	It is a processor in which memory and I/O output component is connected externally.	It is a controlling device in which memory and I/O output component is present internally.
3	Since memory and I/O output is to be connected externally. Therefore the circuit is more complex.	Since on chip memory and I/O output component is available. Therefore the circuit is less complex.
4	It cannot be used in compact system. Therefore microprocessor is inefficient.	It can be used in compact system. Therefore microcontroller is more efficient.
5	Microprocessor has less number of registers. Therefore most of the operations are memory based.	Microcontroller has more number of registers. Therefore a program is easier to write.
6	A microprocessor having a zero status flag.	A microcontroller has no zero flag.
7	It is mainly used in personal computers.	It is mainly used in washing machines, air conditioners etc.

Microprocessor	Microcontroller

Microprocessor	Microcontroller
	
Microprocessor assimilates the function of a central processing unit (CPU) on to a single integrated circuit (IC).	A microcontroller can be considered as a small computer that has a processor and some other components in order to make it a computer.
Microprocessors are mainly used in designing general-purpose systems from small to large and complex systems like supercomputers.	Microcontrollers are used in automatically controlled devices.
Microprocessors are basic components of personal computers.	Microcontrollers are generally used in embedded systems
The computational capacity of the microprocessor is very high. Hence can perform complex tasks.	Less computational capacity when compared to microprocessors. Usually used for simpler tasks.
A microprocessor-based system can perform numerous tasks.	A microcontroller based system can perform single or very few tasks.
Microprocessors have integrated Math Coprocessor. Complex mathematical calculations which involve floating point can be performed with great ease.	Microcontrollers do not have math coprocessors. They use software to perform floating-point calculations which slows down the device.
The main task of the microprocessor is to perform the instruction cycle repeatedly. This includes fetch, decode and execute.	In addition to performing the tasks of fetch, decode and execute, a microcontroller also controls its environment based on the output of the instruction cycle.
In order to build or design a system (computer), a microprocessor has to be connected externally to some other components like Memory (RAM and	The IC of a microcontroller has memory (both RAM and ROM) integrated into it along with some other components like I / O devices and timers.

Microprocessor	Microcontroller
ROM) and Input / Output ports.	
The overall cost of a system built using a microprocessor is high. This is because of the requirement of external components.	The cost of a system built using a microcontroller is less as all the components are readily available.
Generally, power consumption and dissipation are high because of the external devices. Hence it requires an external cooling system.	Power consumption is less.
The clock frequency is very high usually in the order of Giga Hertz.	The clock frequency is less usually in the order of MegaHertz.
Instruction throughput is given higher priority than interrupt latency.	In contrast, microcontrollers are designed to optimize interrupt latency.
Have few bit manipulation instructions	Bit manipulation is powerful and widely used feature in microcontrollers. They have numerous bit manipulation instructions.
Generally, microprocessors are not used in real-time systems as they are severely dependent on several other components.	Microcontrollers are used to handle real-time tasks as they are single programmed, self-sufficient and task-oriented devices.

Ultimately, a computer system based on a CPU may be organized and optimized in many ways using microcontrollers and microprocessors. A microprocessor is a single chip that holds a more powerful CPU and links to external peripherals, as opposed to a microcontroller, which integrates the CPU and all peripherals into a single chip. Microprocessors are better suited for broad computing applications that call for more intricate and diverse computing operations, whereas microcontrollers are tailored to execute a particular low-power application, perfect for embedded systems.

Types of Pins

A **General-Purpose Input/Output (GPIO)** is an uncommitted digital signal pin on an integrated circuit or electronic circuit (e.g. MCUs/MPUs) board which may be used as an input or output, or both, and is controllable by software.

GPIOs are by default unused and have no known use. When employed, the designer of higher assembly-level circuitry defines and implements the function and behavior of a GPIO.

The electrical and temporal requirements of the GPIO interface and the capacity of software to communicate with GPIOs in a timely way are the only constraints on the usage of GPIOs in a wide range of applications.

Standard logic levels are typically used by GPIOs, which are unable to provide considerable current to output loads. A GPIO can be used to operate high-power devices like lights, solenoids, heaters, and motors when it is followed by a suitable high-current output buffer (or mechanical or solid-state relay) (e.g., fans and blowers). Similar to this, an input buffer, relay, or opto-isolator is frequently used to convert a signal (such as high voltage) that would otherwise be incompatible to the logic levels needed by a GPIO.

There are often few to many general-purpose input/output pins included in microcontrollers (GPIO). Software can set GPIO pins to either an input state or an output state. GPIO pins are frequently used to read sensors or outside signals when they are set to an input state. When set to the output state, GPIO pins may frequently be indirectly used to power external components like motors or LEDs.

Numerous embedded systems require the ability to interpret analog signals from sensors. This is what an analog-to-digital converter is used for (ADC). Analog signals that may be supplied to a processor by a device are useless since processors are designed to understand and handle digital data, or 1s and 0s. In order to transform the incoming data into a format that the processor can understand, an analog to digital converter is employed. A digital-to-analog converter (DAC), which is a less frequent function on some microcontrollers, enables the processor to produce analog signals or voltage levels.

Many embedded microprocessors feature a variety of timers in addition to the converters. The programmable interval timer is one of the most used types of timers (PIT). A PIT can count down from a value to zero or up to the count register's limit before overflowing to zero. When it hits zero, it interrupts the processor to let it know that the counting is done. This is helpful for appliances like thermostats, which check the temperature in the area on a regular basis to determine whether to switch on or off the heater, air conditioner, or other appliances.

Power converters, resistive loads, motors, and other devices may be controlled by the CPU without consuming a lot of CPU resources in constrained timer loops thanks to a specialized pulse-width modulation (PWM) block.

Data may be received and sent over a serial line using a universal asynchronous receiver/transmitter (UART) block with almost any CPU burden. The ability to interface with other devices (chips) in digital formats like Inter-Integrated Circuit (I2C), Serial Peripheral Interface (SPI), Universal Serial Bus (USB), and Ethernet is frequently included in dedicated on-chip hardware.

I^2C Inter-Integrated Circuit

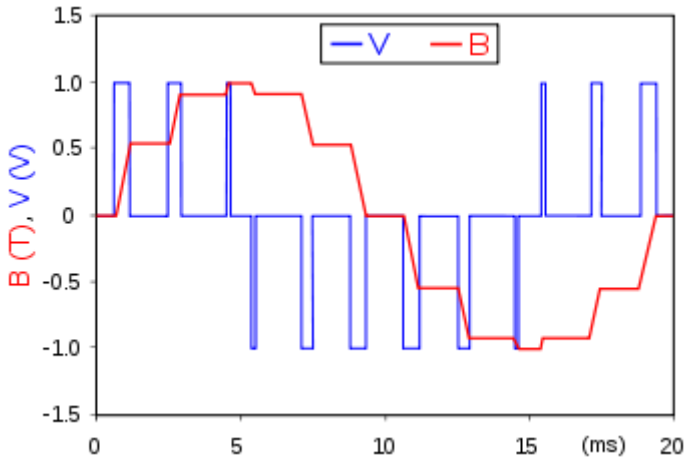
I^2C (Inter-Integrated Circuit) (Inter-Integrated Circuit, n.d.), alternatively known as I2C or IIC, is a synchronous, multi-controller/multi-target (master/slave), packet switched, single-ended, serial communication bus invented in 1982 by Philips Semiconductors. It is frequently used for short-range, intra-board communication between CPUs and microcontrollers and lower-speed peripheral ICs.

I2C is frequently used to connect peripheral circuits to prototype platforms like the Raspberry Pi and Arduino. Despite the fact that I2C does not use a standardized connector, board designers have developed a number of wiring schemes for I2C connectivity. Some developers have recommended utilizing alternating signal and power connections of the following wiring schemes: (GND, SCL, VCC, SDA) to reduce potential harm caused by plugging 0.1-inch headers in backwards (VCC, SDA, GND, SCL).

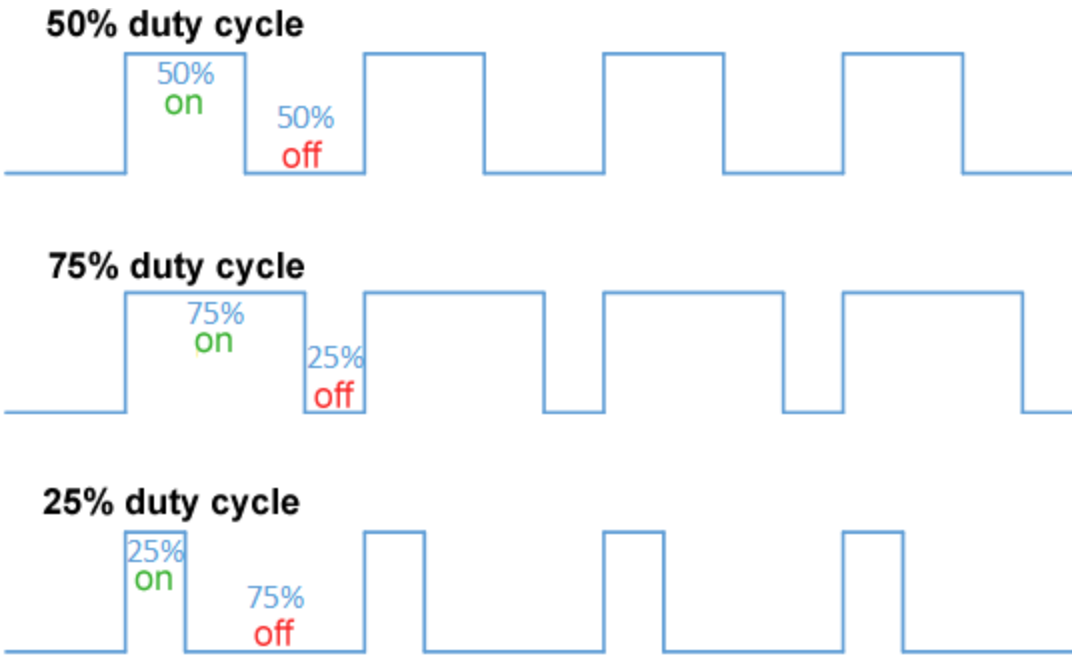
Pulse-width modulation (PWM)

Pulse-width modulation (PWM) (Pulse-width modulation, n.d.), or pulse-duration modulation (PDM), is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. By rapidly flipping the switch between the supply and the load on and off, the average amount of voltage (and current) provided to the load is managed. The total power provided to the load increases while the switch is on for a longer length of time compared to when it is off. PWM is best suited for running inertial loads like motors since their inertia makes them react slowly, which makes them less susceptible to this discrete switching. The load must perceive the generated waveform as smoothly as possible, hence the PWM switching frequency needs to be high enough to not impact the load.

An example of PWM in an idealized inductor driven by a ■ voltage source modulated as a series of pulses, resulting in a ■ sine-like current in the inductor. The rectangular voltage pulses nonetheless result in a more and more smooth current waveform, as the *switching frequency* increases. Note that the current waveform is the integral of the voltage waveform.



A low duty cycle equates to low power since the electricity is off for the most of the time; the word duty cycle reflects the ratio of "on" time to the regular interval or "period" of time. Duty cycle is measured in percentages, with 100 percent representing total on. A digital signal has a duty cycle of 50% and looks like a "square" wave when it is on for 50% of the time and off for the other 50%. A digital signal has a duty cycle of >50% when it spends more time in the on state than the off state. A digital signal has a duty cycle of 50% when it spends 50% more time in the off state than the on state. Here is a pictorial that illustrates these three scenarios:



Universal asynchronous receiver-transmitter (UART)

A universal asynchronous receiver-transmitter (UART) (Universal asynchronous receiver-transmitter (UART), n.d.) is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. In order to ensure that the communication channel can handle exact timing, it delivers data bits one by one, starting with the least significant and working up to the most significant. An independent driver circuit manages the electrical signals levels from the UART.

Bytes of data are taken and sequentially sent by the universal asynchronous receiver-transmitter (UART). The bits are put back together into whole bytes at the destination via a second UART. Shift registers, the essential device for converting between serial and parallel modes, are found within each UART. Parallel transmission over numerous wires is more expensive than serial transmission over a single wire or other media for digital information (bits).

Two common signal levels are RS-232 (**General-Purpose Serial Interface**), a 12-volt system, and RS-485, a 5-volt system.

General-Purpose Serial Interface, also known as GPSI, 7-wire interface, or 7WS, is a 7 wire communications interface. It is used as an interface between Ethernet MAC and PHY blocks.

Data is received and transmitted using separate data paths (**TXD, RXD**) and separate data clocks (TXCLK, RXCLK). Other signals consist of transmit enable (TXEN), receive carrier sense (CRS), and collision (COL).

Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) (Serial Peripheral Interface (SPI), n.d.)is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. Motorola created the interface in the middle of the 1980s.

SPI devices typically have a single master and interact in full duplex mode utilizing a master-slave architecture.

The SPI bus specifies four logic signals:

- SCLK: Serial Clock (output from master)
- MOSI: Master Out Slave In (data output from master)
- MISO: Master In Slave Out (data output from slave)
- CS /SS: Chip/Slave Select (often active low, output from master to indicate that data is being sent)

MOSI on a master connects to MOSI on a slave. MISO on a master connects to MISO on a slave.

Note: on a slave-only device, MOSI may be labeled as SDI (Serial Data In) and MISO may be labeled as SDO (Serial Data Out)

Many products can have nonstandard SPI pin names:

Serial Clock:

- SCK, SCLK, CLK, SCL

Master Output → Slave Input (MOSI):

- SIMO, MTSR - correspond to MOSI on both master and slave devices, connects to each other
- SDI, DI, DIN, SI - on slave devices; connects to MOSI on master, or to below connections
- SDO, DO, DOUT, SO - on master devices; connects to MOSI on slave, or to above connections

Master Input ← Slave Output (MISO):

- SOMI, MRST - correspond to MISO on both master and slave devices, connects to each other
- SDO, DO, DOUT, SO - on slave devices; connects to MISO on master, or to below connections
- SDI, DI, DIN, SI - on master devices; connects to MISO on slave, or to above connections

Analog-to-digital converter (ADC)

In electronics, an analog-to-digital converter (ADC, A/D, or A-to-D) (Analog-to-digital converter (ADC), n.d.) is a system that converts an analog signal, such as a sound picked up by a microphone or light entering a digital camera, into a digital signal. An electrical device that transforms an analog input voltage or current into a digital number reflecting the amount of the voltage or current is an example of an ADC that can offer an isolated measurement.

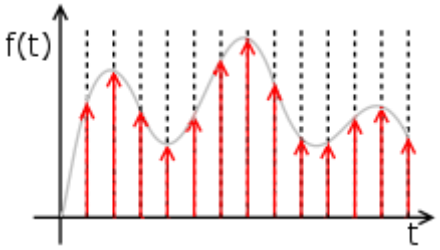
An ADC transforms an analog signal from continuous time and continuous amplitude to discrete time and discrete amplitude. Since the conversion requires quantizing the input, some error or noise is unavoidably included. Additionally, an ADC limits the permissible bandwidth of the input signal by executing the conversion occasionally rather than continuously while sampling the input.

Digital-to-analog converter (DAC)

A digital-to-analog converter (DAC, D/A, D2A, or D-to-A) (Digital-to-analog converter (DAC), n.d.) is a system that converts a digital signal into an analog signal. To do the opposite, an analog-to-digital converter (ADC) is used.

There are several DAC designs, and a DAC's usefulness for a given application is determined by factors like resolution, maximum sampling frequency, and others. A DAC that has negligible faults for the application should be used since digital-to-analog conversion might damage a signal.

A DAC transforms a fixed-point binary number, which is an abstract finite-precision number, into a physical quantity (e.g., a voltage or a pressure). Particularly, DACs are frequently employed to transform time series data with finite accuracy into a physically changing signal.



Music players frequently employ DACs to transform digital data streams into analog audio signals. The enabling technology that has made significant contributions to the digital revolution includes DACs and ADCs. Consider a normal long-distance phone call as an example. A microphone converts the caller's voice into an analog electrical signal, and an ADC transforms the analog signal into a digital stream.

Arduino UNO Pins

The following image shows the complete pinout of Arduino UNO Board.

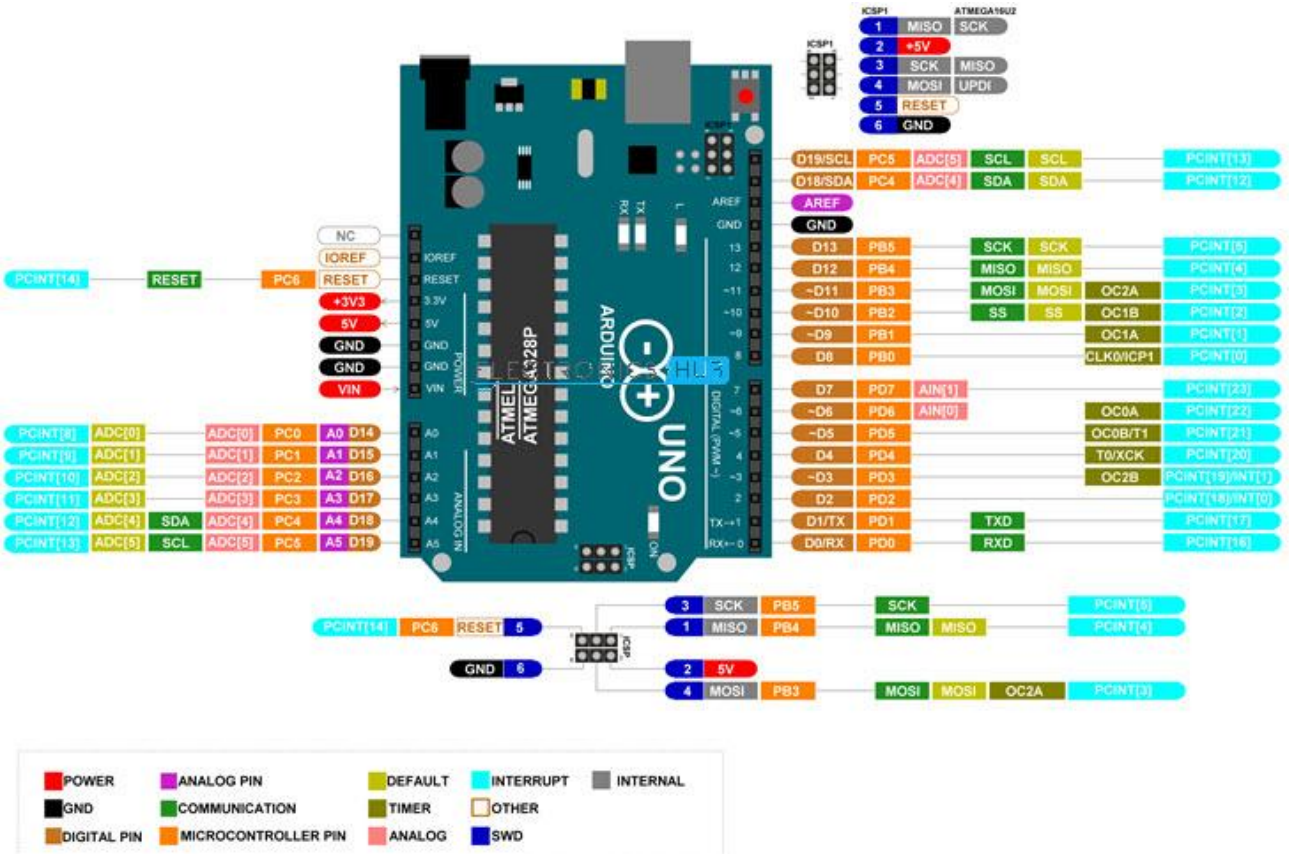


Figure 12 Arduino UNO Pinout

As you can see from the graphic, I detailed each pin on the Arduino UNO along with the pin on the microcontroller it corresponds to, as well as any other uses, standard functionality, and other characteristics.

PIN MAPPING ATmega328P

Using the pinMode(), digitalWrite(), and digitalRead() routines, each of the UNO's 14 digital pins may be utilized as an input or an output. They require 5 volts to work. Each pin includes a 20–50k ohm internal pull-up resistor that is by default unconnected and may provide or receive 20 mA under optimal working conditions. Any I/O pin's maximum current draw must not be exceeded in order to protect the microcontroller from long-term harm.

In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

The UNO includes six analog inputs with the designations A0 through A5, each of which has a resolution of 10 bits (i.e. 1024 different values). Using the AREF pin and the analogReference() function, the top limit of their measurement range may be changed from the default range of ground to 5 volts. There are a couple of other pins on the board:

- AREF. Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Pins Raspberry Pi 4 board

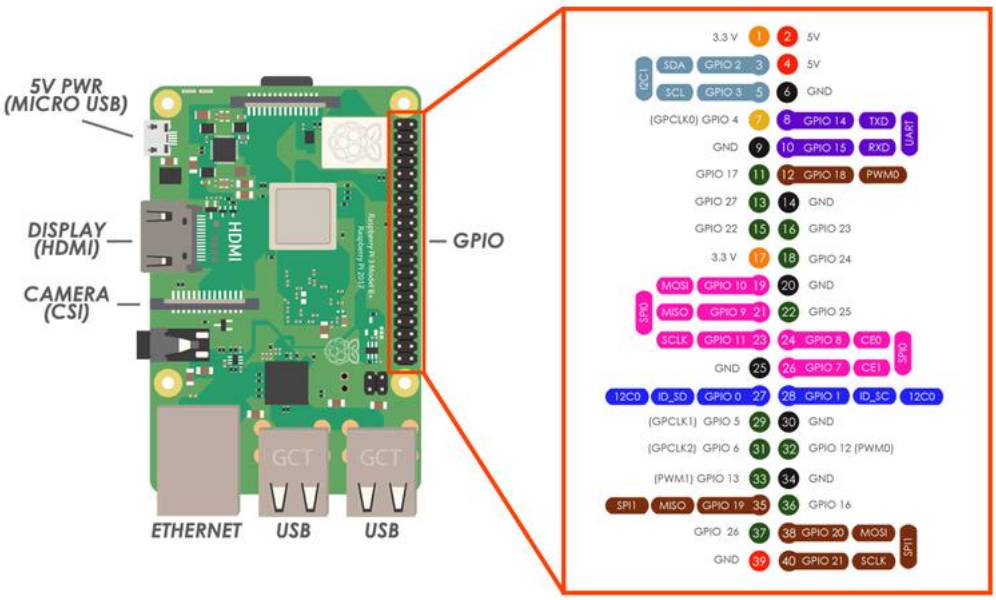


Figure 13 Pins configuration of a Raspberry Pi 4 board

Voltages

Two 5V pins and two 3.3V pins are present on the board, as well as a number of ground pins (0V), which are unconfigurable. The remaining pins are all general purpose 3.3V pins, meaning outputs are set to 3.3V and inputs are 3.3V-tolerant.

Outputs

A GPIO pin designated as an output pin can be set to high (3.3V) or low (0V).

Inputs

A GPIO pin designated as an input pin can be read as high (3.3V) or low (0V). This is made easier with the use of internal pull-up or pull-down resistors. Pins GPIO2 and GPIO3 have fixed pull-up resistors, but for other pins this can be configured in software.

More

As well as simple input and output devices, the GPIO pins can be used with a variety of alternative functions, some are available on all pins, others on specific pins.

- PWM (pulse-width modulation)
  - Software PWM available on all pins
  - Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
- SPI
  - SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
  - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
- I2C
  - Data: (GPIO2); Clock (GPIO3)
  - EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)
- Serial
  - TX (GPIO14); RX (GPIO15)

GPIO pinout

A handy reference can be accessed on the Raspberry Pi by opening a terminal window and running the command

```
$ pinout
```

This tool is provided by the GPIO Zero Python library, which is installed by default on the Raspberry Pi OS desktop image, but not on Raspberry Pi OS Lite.

For more details on the advanced capabilities of the GPIO pins see gadgetoid's interactive pinout diagram (https://pinout.xyz/).

Chapter 3: Designing IoT Applications with Microcontroller and Microprocessor

Applications of Microprocessor

A single-board computer (SBC) is a whole computer with all of the necessary components for operation, including a microprocessor(s), memory, input/output (I/O), and other capabilities. Single-board computers are frequently produced for use as embedded computer controllers, instructional systems, or demonstration or development systems. Numerous models of desktop or laptop computers combine all of its features onto a single printed circuit board.

Single board computers frequently do not rely on expansion slots for peripheral operations or expansion, in contrast to a desktop personal computer. A broad variety of microprocessors have been used to construct single board computers. Static RAM and inexpensive 32- or 64-bit CPUs like ARM are frequently used in straightforward systems, including those created by computer amateurs. Other kinds, such blade servers, would function similarly to server computers, but in a smaller package.

A single-board computer known as a computer-on-module is designed to be plugged into a carrier board, baseboard, or backplane for system expansion.

There are presently two main designs for single-board computers: no slots and slot support.

Embedded SBCs are devices that offer all the necessary I/O but lack plug-in card slots. Typical applications include automated machine control, kiosks, and gambling (slot machines, video poker). Embedded SBCs offer an I/O mix better suited to an industrial application, such as on-board digital and analog I/O, on-board bootable flash memory (without the requirement for a disk drive), no video, etc. They are significantly smaller than the ATX-type motherboard used in PCs.

The term "Single Board Computer" now generally applies to an architecture where the single board computer is plugged into a backplane to provide for I/O cards. In the case of PC104, the bus is not a backplane in the traditional sense but is a series of pin connectors allowing I/O boards to be stacked.

Industrial settings are where single-board computers are most frequently utilized, either embedded within other equipment to give control and interface or used in rackmount configuration for process control. They are employed in both deep-sea exploration by the ALICE deep-sea probes and space exploration by the Space Shuttle, Ariane, and Pegasus rockets. SBCs are frequently more compact, lighter, power-efficient, and dependable than equivalent multi-board computers because of the extremely high levels of integration, decreased component counts, and lower connection counts.

The main benefit of an ATX motherboard over an SBC is price. Millions of motherboards are produced each year for the office and consumer industries, allowing for significant economies of scale. Single-board computers fill a specific market need and are produced less frequently and more expensively. A motherboard failure in either standard will require an identical replacement as both SBCs and motherboards now offer comparable degrees of feature integration.

Standardized computer form factors are used by one popular type of single board computer that is meant to be installed inside a backplane enclosure. These kinds include CompactPCI, PXI, VMEbus, VXI, and PICMG, to name a few. The Intel architecture, multiprocessing architectures, and low power processor systems like RISC and SPARC have all been used as the foundation for SBCs. The intelligence and interface/control circuitry are housed on a plug-in board in the world of Intel PCs, which is subsequently put into a passive (or active) backplane. The ultimate result is comparable to having a motherboard installed in a system, with the exception that the backplane decides how the slots are arranged. Backplanes with a variety of slots (ISA, PCI, PCI-X, PCI-Express, etc.) are available; typically, these backplanes have 20 or less slots, making them suitable for 19" rackmount enclosures (17" wide chassis).

Some single-board computers incorporate connections that make it possible to create a stack of circuit boards, each of which has expansion hardware, without the need of a conventional backplane. PC/104, PC/104-Plus, PCI-104, EPIC, and EBX are a few examples of stacking SBC form factors; these systems are frequently used in embedded control systems.

Memory for stack-type SBCs is frequently provided on plug-cards like SIMMs and DIMMs. Due to the HDD's status as a single block storage device and the fact that the majority of SBCs may boot from their network

connections, hard drive circuit boards are not taken into account when deciding whether or not a computer is an SBC.

The Best Single-Board Computers of 2022 (The Best Single-Board Computers of 2022, n.d.) and (14 Best Single Board Computers in 2022, n.d.)

Budget: Raspberry Pi Zero 2 W	General-purpose powerhouse in compact form	$15
All-Rounder: Raspberry Pi 4 Model B	The pefect blend of performance, compatibility, expandability, and price	$55
Power: LattePanda 3 Delta 864	Fast and powerful Windows-compatible PC disguised as an SBC	$279
Machine Learning: BeagleBoard.org BeagleBone AI-64	Versatility with the edge for computer vision and edge-AI applications	$189
Media Center: Odroid-N2+ 2GB	Quiet media maestro with HDR10, HLG, and Dolby Atmos and DTS:X passthrough	$66
Gaming: Raspberry Pi 4 Model B 4 GB	Affordability with broad emulation and streaming support	$55
Making: BeagleBoard.org PocketBeagle	Powerful real-time computing for headless, embedded applications	$45

Select a use case first. The best SBC for desktop use is unlikely to be the best for embedded development, and an SBC with robust multimedia capabilities for home theater use may not be appropriate for gaming. The best devices in a given field might be so carefully crafted for their intended use that they're all but useless for other tasks. If a general-purpose input/output (GPIO) header is required, make sure it is there and that it has the pin functionality your external devices requires.

Set a budget next, with three prices for each category: The most cost-effective option is a mid-range balanced device. Budget options stretch limited finances as far as they can without making many sacrifices. High-end upgrade options are available for those wanting to spend more for the finest possible experience.

Raspberry

Raspberry Pi (Raspberry Pi, n.d.) (Raspberry Pi Foundation, n.d.) is a series of small single-board computers (SBCs) developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. The initial focus of the Raspberry Pi project was to encourage the study of fundamental computer science in classrooms and in underdeveloped nations. The initial model sold outside of its intended market for applications like robotics because it was more widely used than planned. Because of its inexpensive cost, versatility, and open design, it is frequently utilized in numerous fields, including weather monitoring. Due to its support of the HDMI and USB standards, computer and electrical enthusiasts frequently utilize it.

Raspberry Pi Versions[3]

Version	Model	Size	Processor	Memory	USB ports	Ethernet	WiFi and Bluetooth	Year Released	Price (USD)
Raspberry Pi	B	Standard	700 MHz 1-core	512 MB	2	Yes	No	2012	N/A
	A				1	No		2013	N/A
	B+				4	Yes		2014	$25
	A+	Compact			1	No			$20
Raspberry Pi 2	B	Standard	900 MHz 4-core	1 GB	4	Yes	No	2015	$35
Raspberry Pi Zero	Zero	Zero	1000 MHz 1-core	512 MB	1	No	No	2015	$5
	Zero W						Yes	2017	$10
Raspberry Pi 3	B	Standard	1200 MHz 4-core	1 GB	4	Yes	Yes	2016	$35

	A+	Compact	1400 MHz 4-core	512 MB	1	No		2018	$25
	B+	Standard		1 GB	4	Yes		2018	$35
Raspberry Pi 4	B (1 GB)	Standard	1500 MHz 4-core	1 GB	2xUSB2, 2xUSB3	Yes	Yes	2019	N/A
	B (2 GB)			2 GB					$35
	B (4 GB)			4 GB					$55
	B (8 GB)			8 GB				2020	$75

Raspberry Pi Zero 2 W

The most recent Raspberry Pi ultra-compact single-board computer readily justifies being three times as expensive as the Raspberry Pi Zero.

Instead of a single 32-bit processing core, it has four 64-bit cores, which take use of a newer and more effective design. Your workloads will consequently run up to 12 times quicker than they did on the original Raspberry Pi Zero.

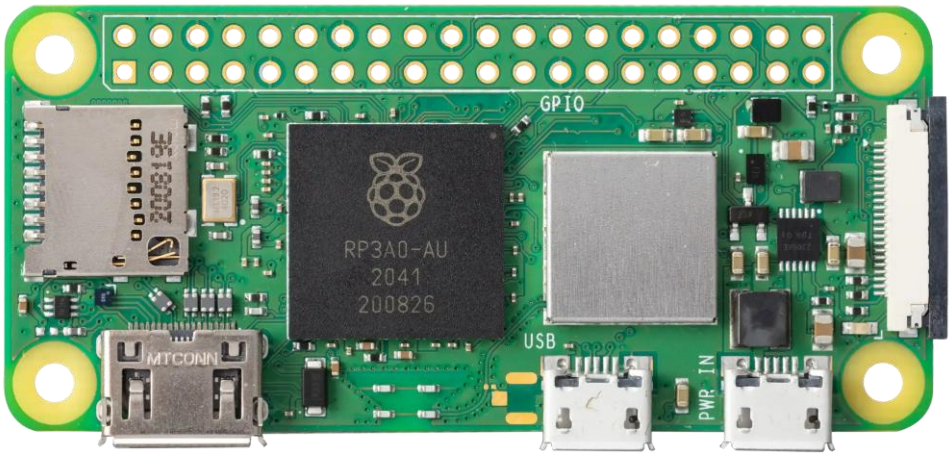


Figure 14 The Raspberry Pi Zero 2 W (Source: Gareth Halfacree)

The Raspberry Pi Zero 2 W is a complete Raspberry Pi board, despite its compact size and inexpensive price. A 40-pin general-purpose input/output (GPIO) header is available, and it is compatible with practically all Hardware Attached on Top (HAT) add-ons that are available for full-size Raspberry Pi models. Instead of utilizing wired Ethernet, the Zero series makes use of 2.4GHz Wi-Fi.

Specification:

1GHz quad-core 64-bit Arm Cortex-A53 CPU

512MB SDRAM

2.4GHz 802.11 b/g/n wireless LAN

Bluetooth 4.2, Bluetooth Low Energy (BLE), onboard antenna

Mini HDMI port and micro USB On-The-Go (OTG) port

microSD card slot

CSI-2 camera connector

HAT-compatible 40-pin header footprint (unpopulated)

H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30)

OpenGL ES 1.1, 2.0 graphics

Micro USB power

Composite video and reset pins via solder test points

65mm x 30mm

At more than twice the price of the Raspberry Pi Zero 2 W, the Pine64 Rock 64 does stretch the budget – but for a lot of use-cases, it's well worth it. The processor uses the same Arm Cortex-A53 cores as the Zero 2 W, but running half as fast again – and with four times as much memory, performance is dramatically better. There's no Wi-Fi on board the Rock64, but you do get a wired Ethernet port and three full-size USB ports, one of which offers high-speed USB 3.0 connectivity. There's even an infrared receiver for media remotes and an onboard real-time clock.

Raspberry Pi 4 Model B

There's a reason Raspberry Pi is the go-to for single-board computer projects: The boards offer a great mix of performance, compatibility, expandability, and price.

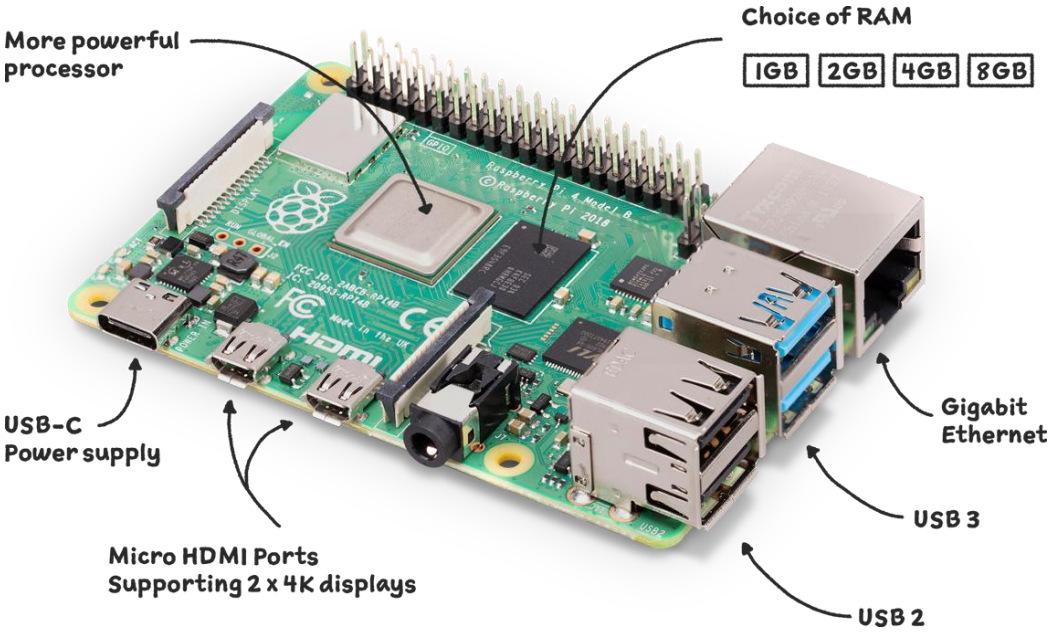


Figure 15 Raspberry Pi 4 Model B

In February 2012, the first-generation Raspberry Pi Model B and Model A were both made available.

A 1.5 GHz 64-bit quad-core ARM Cortex-A72 processor, onboard 802.11ac Wi-Fi, Bluetooth 5, full gigabit Ethernet (throughput not limited), two USB 2.0 ports, two USB 3.0 ports, 1–8 GB of RAM, and dual-monitor support via a pair of micro HDMI (HDMI Type D) ports for up to 4K resolution are all features of the Raspberry Pi 4 Model B, which was released in June 2019. The 1 GB RAM version has been discontinued, while the 2 GB RAM version's costs have dropped. The circuit board has been updated for the 8 GB version. Additionally, the Pi 4 has a USB-C connector that can be used to charge it, providing extra power to peripherals when coupled with the right PSU. But unlike other tiny computers in its class, the Pi can only run on 5 volts, not 9 or 12 volts.

This chip, which was introduced in November 2020, is now being used by the manufacturer in the Pi 4 B and Raspberry Pi 400 models. It is a contemporary illustration of a keyboard computer and includes a keyboard and 4 GB of LPDDR4 RAM on a custom board adapted from the Raspberry Pi 4.

Costing just $15, it offers compatibility with the same software and Hardware Attached on Top (HAT) (Sense HAT, n.d.) add-on hardware at less than half the price of its full-sized equivalent.

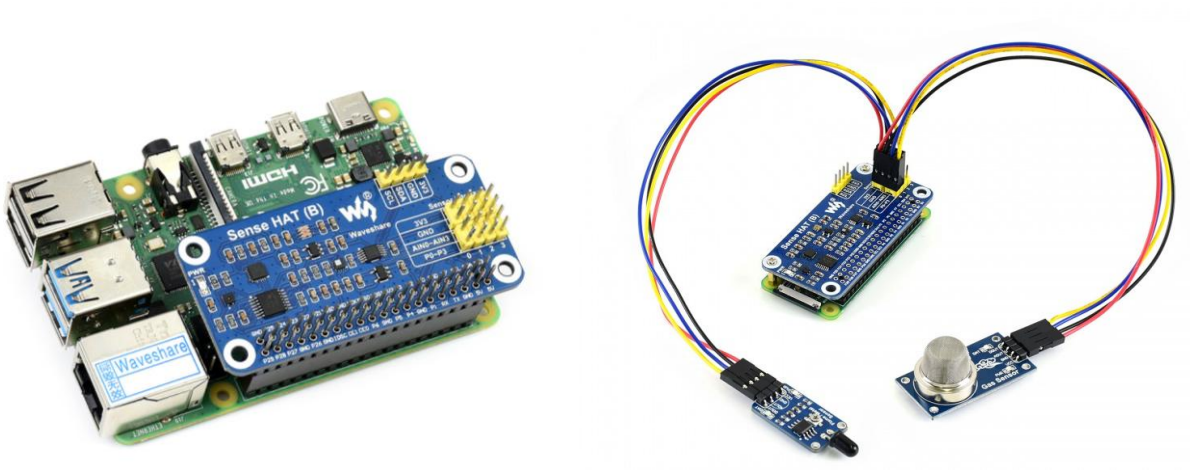


Figure 16 Sense HAT (B)

Specification of Sense HAT (B)

- Working voltage 3.3V
- Interface I2C
- Dimension 65mm x 30.5mm
- Accelerometer Resolution: 16-bit (Ranging:±2/±4/±8/±16g)
- Gyoscope Resolution: 16-bits (Ranging:±250/±500/±1000/±2000°/sec)
- Magnetometer Resolution: 16-bits (Ranging:±4900μT)
- Barometer Resolution:24-bits (Pressure), 16-bits(Temperature)
- Temperature & Humidity
- Color sensor Resolution: 4-channels RGBC, 16-bits per channel
- ADC Resolution: 12-bits

Also, the Raspberry Pi Sense HAT (Raspberry Pi Sense HAT, n.d.) is attached on top of the Raspberry Pi via the 40 GPIO pins (which provide the data and power interface) to create an 'Astro Pi'. The Sense HAT has several integrated circuit based sensors can be used for many different types of experiments, applications, and even games. The sensors enable you to read: Orientation (yaw, pitch & roll) via an accelerometer, 3D gyroscope and magnetometer; Pressure; Humidity;Temperature

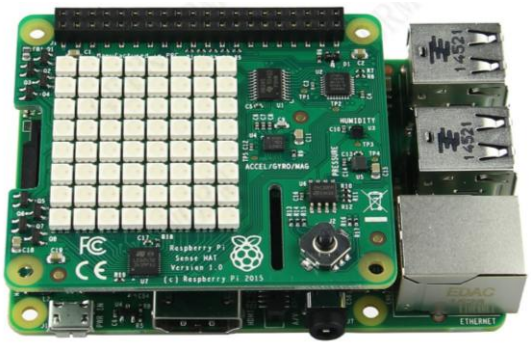


Figure 17 Raspberry Pi Sensor HAT

Specification:

- Gyroscope – angular rate sensor: +/-245/500/2000dps
- Accelerometer - Linear acceleration sensor: +/-2/4/8/16 g
- Magnetometer - Magnetic Sensor: +/- 4/8/12/16 gauss
- Barometer: 260 – 1260 hPa absolute range (accuracy depends on the temperature and pressure, +/- 0.1 hPa under normal conditions)
- Temperature sensor (Temperature accurate to +/- 2 degC in the 0-65 degC range)
- Relative Humidity sensor (accurate to +/- 4.5% in the 20-80%rH range, accurate to +/- 0.5 degC in 15-40 degC range)
- 8x8 LED matrix display
- Small 5 button joystick

LattePanda 3 Delta 864

An electronic device made in China is called the LattePanda. Though theoretically comparable to the Raspberry Pi, it costs a lot more and uses Intel CPUs rather than ARM.

General purpose input/output (GPIO) pins, which are identical to those on a Raspberry Pi, are managed by a separate co-processor that is attached to the computer's Intel Atom CPUs.

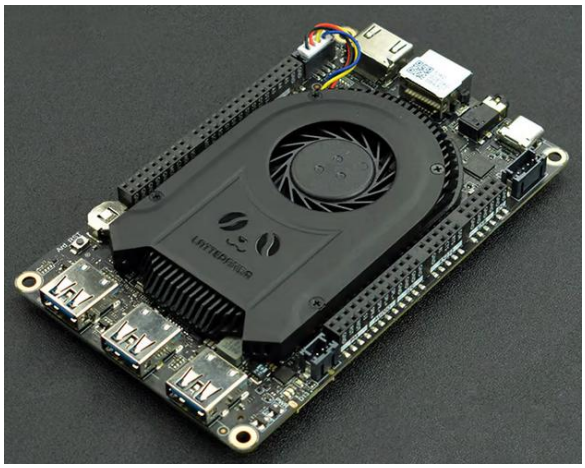


Figure 18 The LattePanda 3 Delta 864 single-board compute

The LattePanda 3 Delta is a single-board computer that also doubles as a fully working small PC. The board's central Intel Celeron N5105 CPU provides strong performance at a respectably low power drain, but not so low that it doesn't require the included fan to keep it cool. A large 8 GB of RAM and 64 GB of onboard storage are included, and a wide range of operating systems, including Windows 10 and Windows 11, are compatible.

While the B-Key slot enables 4G and 5G cellular modules if the onboard gigabit Ethernet and dual-band Wi-Fi 6, two M.2 slots offer high-speed storage expansion.

ROCK PI MODEL

Rockchip is a Chinese company called Fuzhou Rockchip Electronics Co., Ltd. The Radxa Rock Pi 4 Model C was scheduled to arrive in October 2019, however it was delayed by more than six months. It's finally on sale for $59 right now.

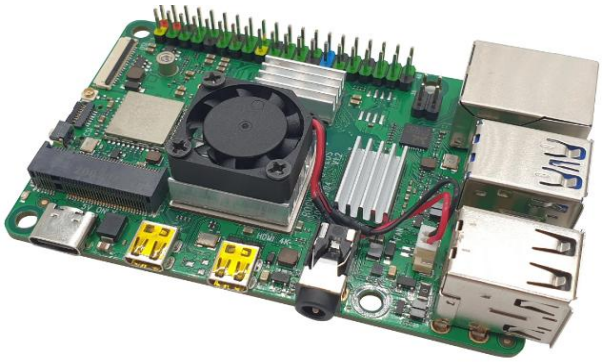


Figure 19 ROCK PI 4 MODEL C+

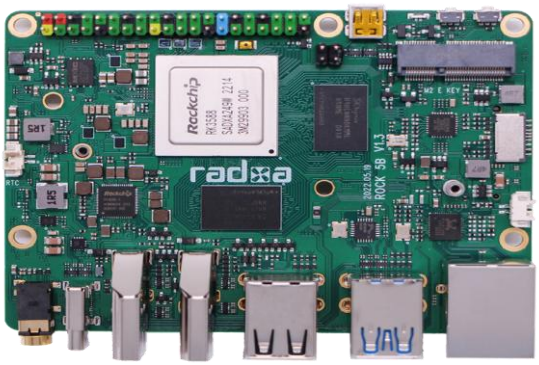


Figure 20 ROCK5 Model B

Significantly greater power is provided by the Rock Pi 4 Model C+. The Rock Pi 4 Model C+ is quick because to its six-core CPU and quad-core GPU, which according to Rock Pi offers a claimed four times the computational capability of "Competitor R." An NMVe-compatible M.2 slot and eMMC socket enable high-speed storage expansion options. It's also shockingly reasonable at $69 sent directly from China. Like with other SBCs, you'll have to supply your own peripherals and power source.

Accordingly, the octa-core Cortex-A76/A55 Pico-ITX SBC, M.2 NVMe storage, 2.5GbE, optional WiFi 6E, 8K video output through HDMI or USB-C ports, 4K HDMI input, and more are features of the ROCK5 Model B single board computer. The device's starting price is $79 as well.

Applications of Microcontroller

Microcontrollers consist of just one chip. They must be integrated into a system with a clock, support ICs, I/O pins, etc. in order to be used. This would be incorporated into the final design of a commercial system. However, a pre-assembled board would be helpful for amateurs and for prototyping.

The single-board microcontrollers are responsible for this. A modern system is Wiring, which is based on the AVR ATmega128 microcontroller and includes an 8-channel 10-bit A/D converter, 4KB SRAM, 128KB of programmable flash memory, and 4KB of EEPROM. The Arduino is now favored in this area, and this section goes into great detail about it.

Arduino

Arduino (Margolis) is an "open source" board with built-in microcontroller and I/O.

A project was started in 2005 to create a system that would be less expensive than other current prototype systems for student control of interactive design tools. The design was called after the Arduino of Ibrea by the company's founders, Massimo Banzi and David Cueartielles, who started making circuit boards in a modest factory in the town of Ibrea in the province of Torino in northwest Italy's Piedmont area.

The Arduino board is made up of an Atmel AVR microcontroller (ATmega8 in previous versions, ATmega328 and ATmega168 in current versions), as well as other parts that make it easier for users to program it and integrate it into other circuits. A 16MHz crystal oscillator and a 5V linear voltage regulator are present on every board (or ceramic resonator in some variants). No additional programmer is required because the microcontroller comes pre-programmed with a bootloader.

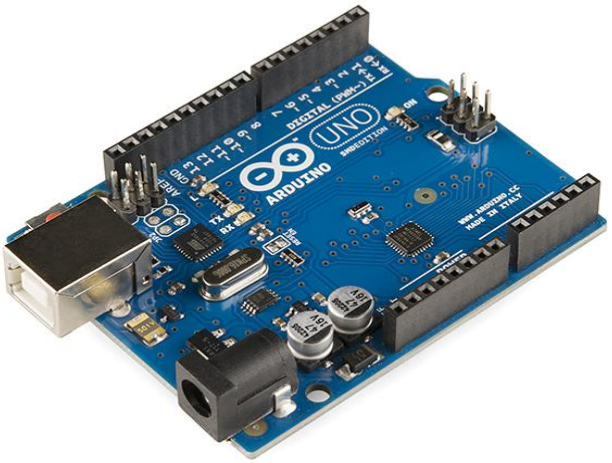


Figure 21 Board Arduino UNO 3

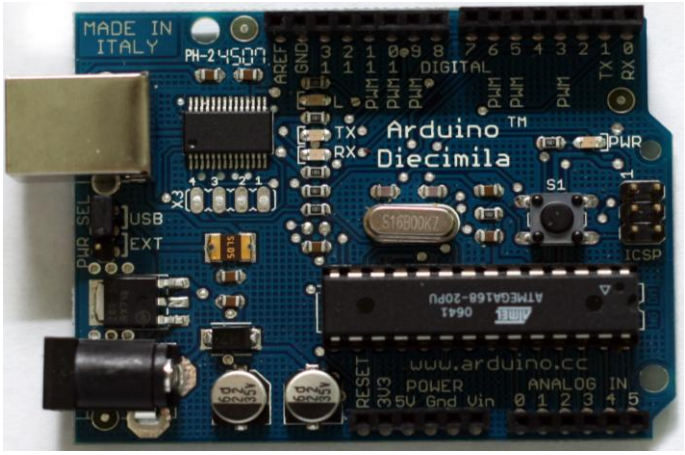


Figure 22 Boad Arduino Diecimila

Official board

The Italian business Smart Projects is in charge of producing the original Arduino firmware. American business SparkFun Electronics created some boards under the Arduino name. Sixteen versions of Arduino Hardware have been used commercially so far:

- The Serial Arduino, programmed with a DE-9 serial connection using ATmega8 technology.
- The Arduino Extreme, with a USB interface for programming using ATmega8 technology.
- Arduino Mini, a miniature version of Arduino using surface-mounted ATmega168 technology.
- The Arduino Nano, an even smaller, USB-powered version of the Arduino using surface-mounted ATmega168 (ATmega328 for the newer version) technology.
- LilyPad Arduino, a minimalist design for clothing and E-textiles applications using AT-mega328 surface-mounted technology.
- The Arduino NG, with a USB interface for programming and using ATmega8 technology.
- The Arduino NG plus, with a USB interface for programming and using ATmega168 technology.
- Arduino Bluetooth, with Bluetooth interface for programming using ATmega168 technology.
- The Arduino Diecimila, with a USB interface and uses ATmega168 technology in a DIP28 package.

- Arduino Duemilanove (“2009”), uses ATmega168 technology (ATmega328 for the new version) and is powered by USB/DC power, automatically alternating.
- The Arduino Mega, using surface-mounted ATmega1280 technology for further I/O and memory.
- The Arduino Uno, using the same ATmega328 technology as the latest model Duemilanove, but while the Duemilanove uses an FTDI chipset for USB, the Uno uses ATmega8U2 technology programmed as a serial converter.
- The Arduino Mega2560 uses surface-mounted ATmega2560 technology bringing the total memory to 256kB. It also incorporates the new ATmega8U2 (ATmega16U2 in type 3 revision) USB chipset.
- The Arduino Leonardo, with an ATmega32U4 chip that eliminates the need for USB connectivity and can be used as a digital keyboard or mouse. Launched at Maker Faire Bay Area in 2012.
- The Arduino Esplora, with an appearance reminiscent of a video game console controller with a joystick and built-in sensors for sound, light, temperature and acceleration.
- Arduino Due is a microcontroller board based on Atmel SAM3X8E ARM Cortex-M3 CPU technology. It is the first Arduino board based on a 32-bit ARM microcontroller.

An integrated development environment (IDE) and a language (syntax and libraries) based on the Wiring platform, which is a fork of the open source software platform Wiring, are used to program the Arduino project (IDE).

Figure 23 Arduino IDE

AVR Physical Architecture

This section discusses the AVR 8-bit CPU Core Block (Arduino Tutorial – The Industrial and Professional Way, n.d.) architecture in general.

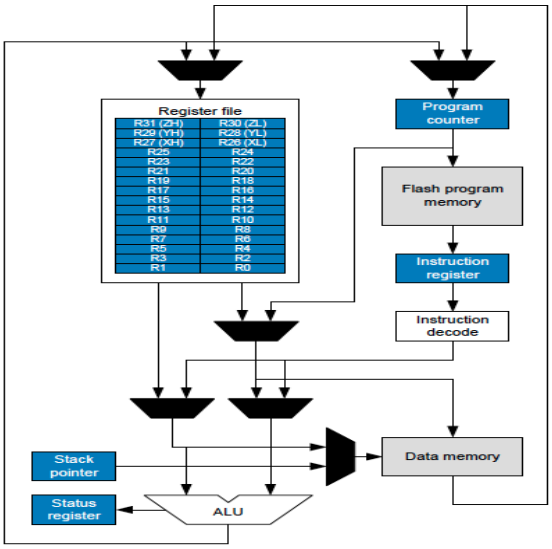


Figure 24 AVR 8-Bit Core Architecture Block Diagram

The Harvard architecture that the AVR employs has distinct memory and buses for both program and data. Program memory instructions are categorized as 8-bit RISC devices and are performed with single level pipelining. The next instruction is pre-fetched from the program memory while the previous one is being executed. This idea makes it possible to carry out instructions once every clock cycle. In AVR microcontrollers, Flash, EEPROM, and SRAM are all combined into a single chip. 8 bits make up the Data Bus's width.

AVR Registers

AVR has 32 × 8-bit general purpose working registers that may be accessed in one clock cycle. For Data Space addressing, three 16-bit indirect address register pointers (X-, Y-, and Z-registers) can be utilized with six of the 32 registers. Look-up tables in Flash program memory can utilize one of these address pointers as its address pointer as well.

The 32 x 8-bit general purpose working registers are mapped into the first 32 memory addresses (000016–001F16), followed by 64 I/O registers (0000_{16}–$001F_{16}$), followed by 64 I/O registers (0020_{16}–$005F_{16}$).

AVR Arithmetic Logic Unit (ALU)

Arithmetic and logic operations between registers, or between a constant and a register, are supported by the ALU. Single-cycle ALU operations are possible with 32 × 8-bit general purpose working register access across a single clock cycle.

In a general ALU calculation, the operation is carried out, the result is placed back in a register in one clock cycle, and two registers are utilized as the operands. The ALU may also do single register operations. The

Status register is updated following an arithmetic operation to reflect details on the outcome of the operation.

The three basic kinds of ALU operations are arithmetic, logical, and bit functions. Some architectural implementations additionally include a potent multiplier that supports both signed and unsigned multiplication as well as fractional format.

Program Flash Memory

The AVR architecture's memory spaces provide a linear and regular memory map. Typically, the component number of the device itself includes information about the amount of the flash program memory (e.g., the ATmega32x line has 32KB of flash, while the ATmega64x line has 64KB). For storing program instructions, the AVR Flash is an on-chip in-system reprogrammable flash memory. Flash memory typically lasts for at least 10,000 writes and erases.

The Boot Program portion and the Application Program section make up the two compartments of the AVR Program Flash memory. To avoid tampering, lock bits are available for read/write protection. The bootloader has the ability to update the application section.

The flash is organized in 16-bit width word size since all AVR instructions are either 16 or 32 bits wide.

Data Memory

AVR Data Memory is implemented in a separate address bus than the Program Memory. There are two data buses, one that can access all data and the In/Out data bus with limited access to a small section of memory.

The AVR Data Address Space consists of:

- Registers (32 x 8-bit) (0000_{16}–$001F_{16}$)
  Consists of 32 8-bit general purpose working registers (R0-R31)
- I/O Memory (64 x 8-bit) (0020_{16}–$005F_{16}$)
  Contains addressable space for peripheral control registers and other I/O registers
- Extended I/O Memory (device dependent) (160 x 8-bit) (0060_{16}–$00FF_{16}$)
  Some AVR microcontrollers with more peripherals need more space than the I/O memory can address so some of the SRAM is used as Extended I/O memory to handle the extra peripherals control registers and other I/O functions
- Internal SRAM (2048 x 8-bit) (0100_{16}–$08FF_{16}$)
- SRAM is used for temporarily storing intermediate results and variables, stack within a software application.

EEPROM

AVR **Electrically Erasable Programmable Read-Only Memory (EEPROM)** is organized as a separate data space, in which single bytes can be read and written. The EEPROM can withstand at least 100,000 writes and erases. When bits only need to be set to all 1s (erase) or selectively cleared to 0s (write), it is sometimes

possible to conduct erase and write independently, byte-by-byte. This may also assist prolong the life of the memory (write).

Through the EEPROM Address Registers, EEPROM Data Register, and EEPROM Control Register, the CPU may access EEPROM.

MCU Clock Speed and MIPS Performance

Most AVR devices offer clock speeds between 1MHz and 20MHz. There is no need for additional clocks or resonator circuits because AVR already includes an internal oscillator. A system clock Prescaler that can reduce the system clock by up to 1024 is also available in some AVRs. The clock speed of this Prescaler may be altered while it is running thanks to software configuration changes.

AVR can produce up to 1 MIPS per MHz, therefore a processor running at 12 MHz can produce up to 12 MIPS. The majority of instructions are executed in one clock cycle. Branching and memory loads and stores both require two cycles.

Programming Interfaces

There are several methods for loading computer code into AVR microcontrollers. The techniques used to program AVR chips differ depending on the AVR family. The majority of the techniques listed below enter programming mode using the RESET line.

ISP (In-System Programming)

The SPI interface is used for in-system programming. The connection uses six pins: MISO, MOSI, SCK, RST, VCC, and GND. When the chip is attached to the PCB and the SPI signals are not interfered with, ISP continues to function. The official Atmel-ICE tool, as well as several alternative DIY and open-source tools, allow ISP programming. It is not an interface for debugging. It may be utilized to download code into EEPROM and flash memory.

PDI (Program and Debug Interface)

For external programming and on-chip debugging of XMEGA devices, Atmel exclusively uses the Program and Debug Interface. PDI serially flashed and debugged software using the NVM controller. The PDI is a two-pin interface with a dedicated data pin (PDI DATA) for input and output, and the Reset pin for clock input (PDI CLK). A 2-pin interface called PDI enables synchronous bi-directional half-duplex communication with the target device.

UPDI (Unified Program and Debug Interface)

An exclusive interface for external programming and on-chip device debugging is called the Unified Program and Debug Interface. UPDI is a single-wire interface that replaces the PDI 2-wire physical interface and offers bi-directional half-duplex asynchronous communication with the target device for programming and debugging reasons.

HVSP (High Voltage Serial Programming)

When ISP is not available, a backup programming technique called high voltage serial programming can be utilized to program AVR microcontrollers. HVSP is often used to ATtiny microcontrollers with 8 or 14 pins. It employs a number of pins, but the most crucial one is a 12V signal that must be provided to the target microcontroller's RESET pin.

HVPP (High Voltage Parallel Programming)

As a fallback method, high voltage parallel programming is used in AVR microcontrollers to fix incorrect fuse settings or to unlock locked chips. Additionally, a 12V signal is used to reset the target microcontroller's RESET pin. The high voltage protocol (HVPP) is used with ATtiny and ATmega MCUs that have 20 pins or more.

JTAG (Joint Test Action Group)

While the chip is active in the target system, JTAG gives access to on-chip debugging capabilities. To watch system activity, JTAG enables access to internal memory and registers, the creation of breakpoints in code, and single-stepping execution.

debugWIRE

On-chip debugging capabilities are provided by debugWIRE through a single microcontroller pin. It is not possible to utilize the debugWIRE interface as a programming interface. This implies that in order to program the target via ISP, the SPI interface connection must also be accessible.

Atmega328 Physical Architecture

ATmega328 (Introduction to ATmega328, n.d.) is an 8-bit, 28-Pin AVR Microcontroller, manufactured by Microchip, follows RISC Architecture and has a flash-type program memory of 32KB.

The microcontroller used in the simplest Arduino boards, such as the Arduino UNO, Arduino Pro Mini, and Arduino Nano, is called Atmega328.

If you're working on an engineering project, you must first examine the component pinout diagram in order to comprehend the pin layouts of any electrical equipment.

ATmega328 pinout diagram is shown in the figure given below:

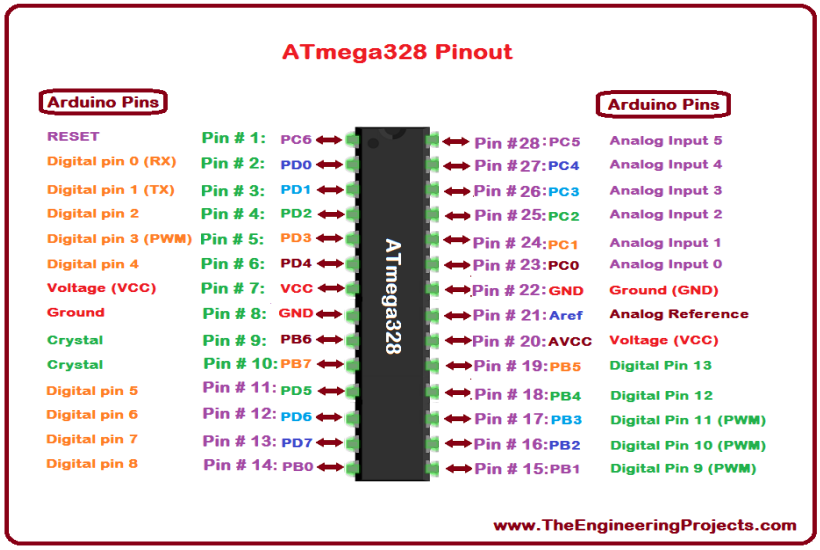


Figure 25 ATmega328 pinout

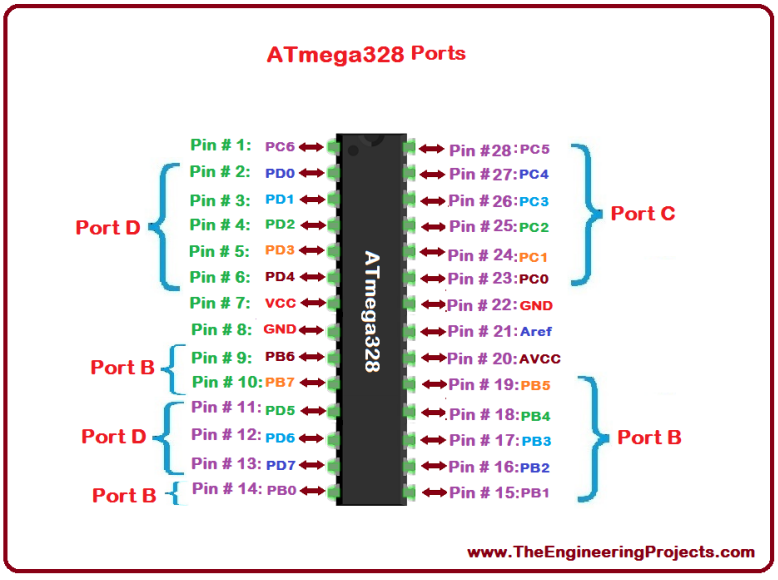
Port A consists of the pins from PA0 to PA7. These pins serve as an analog input to analog to digital converters. If analog to digital converter is not used, port A acts as an eight (8) bit bidirectional input/output port.

Port B consists of the pins from PB0 to PB7. This port is an 8 bit bidirectional port having an internal pull-up resistor.

Port C consists of the pins from PC0 to PC7. The output buffers of port C has symmetrical drive characteristics with source capability as well high sink.

Port D consists of the pins from PD0 to PD7. It is also an 8 bit input/output port having an internal pull-up resistor.

All of the AVR ports are shown in the figure given below.



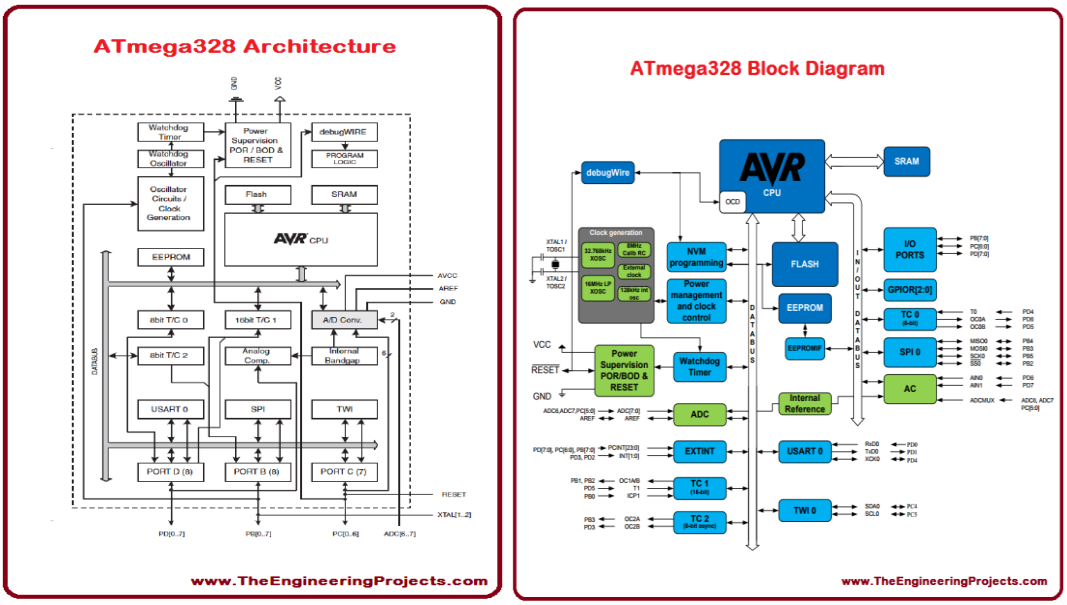


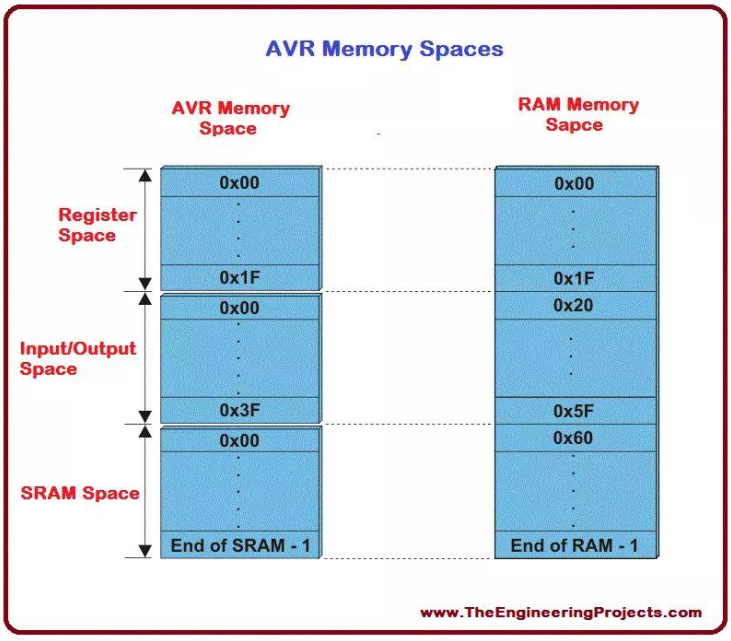
Figure 26 Atmel ATmega328 microcontroller

ATmega328 Memory

ATmega 328 has three types of memories, named:

- Flash Memory: 32KB. It is a Programmable Read-Only Memory (ROM). It is a nonvolatile memory.
- SRAM: 2KB. Stands for Static Random Access Memory. It is a volatile memory i.e. data will be removed after removing the power supply.
- EEPROM: 1KB. Stands for Electrically Erasable Programmable Read-Only Memory.

AVR memory spaces are shown in the figure given below.

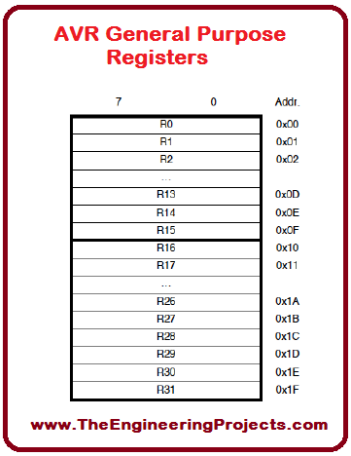


ATmega328 Registers

There are 32 General Purpose (GP) registers on the ATmega-328.

The Static Random Access Memory includes each and every one of these registers (SRAM).

The figure below shows all of the registrations.



Arduino Uno Pinout

The most commonly used board among the Arduino family is the Arduino Uno (Figure 21 Board Arduino UNO 3) as it is easy to use and suitable for beginner and medium level projects. This board has an ATMEGA328P microcontroller, which is a member of the ATMEL family.

This board features 32 kilobytes of flash memory and can run on a 5-volt supply. The controller's static ram has a capacity of 2 kilobytes, while the EEPROM has a memory capacity of 1 kilobyte. The ATMEGA328P operates at a 16 hertz clock frequency. The Arduino Uno board is seen in the figure below.

A total of 31 pins make up the Arduino Uno, 13 of which are digital pins that may be utilized for digital input and output. Ten of them can be used to deliver power to the connected devices, and six of them are analog pins that can be used for analog inputs and outputs.

Pin category	Representation	Description
Power pins of Arduino Uno	5v, RESET, 3.3V, GND (3), Vin, AREF, IOREF	Pins used to deliver power to the device connected with Arduino
Digital pins of Arduino	0 to 13	Pins used for digital input and

Uno		outputs of Arduino
PWM pins of Arduino Uno (Digital pins)	11,10, 9, 6, 5, 3	Pins used to generate the pulsating signal
Analog pins of Arduino Uno	A0 to A5 (A5 for SCL and A4 for SDA)	Pins used for analog inputs and outputs of Arduino
Miscellaneous pins of Arduino Uno	Additional pins for SCL and SDA (One not connected pin [NC])	SCL is the clock pin and the SDA is the data pin for I2C and TWI communication devices
12 header pins of Arduino Uno	ICSP	Pins used to reprogram the Arduino

This board also consists of the 12 header pins as well also called In Circuit System Programming (ICSP) pins.They are also used to program the controller We have explained each pin by dividing the pins on different categories based on their usage in the subsequent paragraphs.

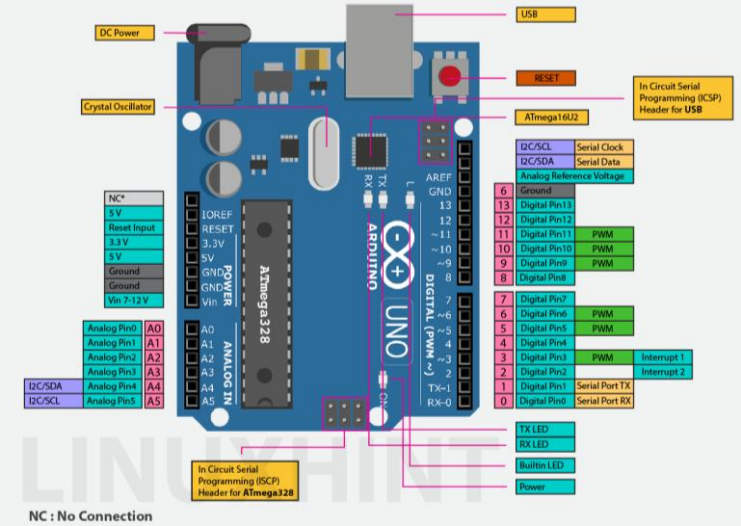


Figure 27 Arduino Uno Pinout

Digital pins of the Arduino Uno

The Arduino has 13 digital pins in total, which may be used to connect devices that need digital input from the Arduino Uno and can also produce digital data. When we refer to information as being digital, we mean that it will take the form of zeros and ones.

The communication pins of the Arduino, designated TX and RX, are the 0 and 1 pins of the digital pins. These pins are used to upload code to the Arduino board as well as for communicating with other communication devices.

The Arduino Uno has a built-in LED that is attached to pin 13, however we can also connect additional devices to this pin.

The digital pins on the Arduino Uno are marked in red in the picture of the board that is shown below.

An image with text, electronics, and a circuit that was created automatically

Analog pins of Arduino Uno

The 6 analog pins of the Arduino Uno are primarily used to connect analog devices and have a resolution of 0 to 1024 (210). Accordingly, the values will range from 0 to 1024, and the voltage for 5 volts will be 1024.

The SDA and SCL pins for devices that used the I2C and TWI (Two Wire Interface) communication protocols are pins A4 and A5, respectively. The linked device's data line is represented by the SDA pin, while its clock pin is represented by the SCL pin. The data line and clock of the I2C devices can also be utilized on the two additional pins that are located adjacent to the AREF pin.

The analog Arduino pins are highlighted in red in the image that is linked below.

An image with text, electronics, and a circuit that was created automatically

Power pins of Arduino Uno

There are a total of 10 pins on the Arduino Uno that are utilized to power the attached devices. The Arduino Uno has four ground pins with a maximum voltage range of 5 volts and a minimum voltage range of 3.3 volts.

Similar to this, the IOREF and AREF pins of Arduino Uno are used to supply a reference voltage for the attached devices. The reference voltage for analog devices is the AREF, whereas the reference voltage for other digital devices is the IOREF. Additionally, the board has a reset pin that may be used with an external button to RESET the Arduino Uno. On the Arduino Uno board, there is a specific RESET button, though.

There is one USB port and a power supply connection available to connect the Arduino board to the power source. Both power and code uploading to the Arduino Uno may be done using the USB interface. While the supply jack is often utilized when the Arduino has to operate in solo mode. The Arduino uno's power supply pins and RESET button are depicted in the figure below.

An image with text, electronics, and a circuit that was created automatically

ICSP header pins of the Arduino Uno

To update or change the firmware of the Arduino Uno we can use the 12 header pins given on the Arduino Uno board. The in circuit system programming (ICSP) can be done by connecting Arduino with the device using a programming cord. We have encircled using square boxes the ICSP header pins of Arduino Uno in the image given below.

A picture containing text, electronics, circuit Description automatically generated

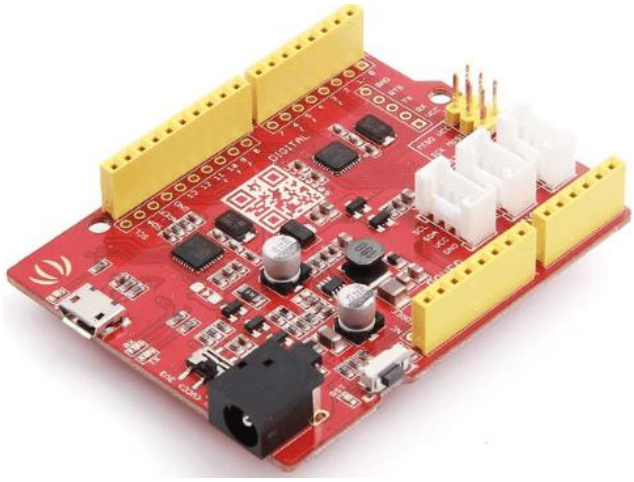
PWM pins of Arduino

The PWM pins, whose duty cycle ranges from 0 to 255, are used to create the inputs and outputs of the Arduino in the form of pulses. The Arduino Uno's PWM-specific pins are 11, 10, 9, 6, and 3. The Arduino PWM pins are marked in red in the illustration below.

Conclusion

The most popular Arduino board among students is the Arduino Uno since it can be used in many different projects and is simple to use. However, in order to utilize this board properly, one must be familiar with both the board's characteristics and the pinout of the relevant Arduino boards. We have included a very thorough explanation of each pin's function on the Arduino Uno for the convenience of the students.

Seeeduino



- Seeeduino V4.2 (cost ($6.90)) is an Arduino-compatible board, which is based on ATmga328P MCU.
- Seeeduino V4.2 is based on the Arduino UNO bootloader, and with an ATMEGA16U2 as a UART-to-USB converter, which means that the board can basically work like an FTDI chip.
- You can program the board via a micro-USB cable, if you have an Android phone, you can find a micro-USB cable easily. Also, you can power the board via a DC Jack input, 7 to 15V is acceptable. There’s a switch to choose the system supply voltage, 3.3V or 5V, which is very useful if you want to set the system to 3.3V to save power.

- In addition, the Seeeduino V4.2 has three onboard Grove interface that can make your board connect to Grove modules easily.
  - For those who do not know what is Grove, Grove is Seeed's very own modular electronic platform for quick prototyping. Every module has one function, such as touch sensing, creating audio effect, etc.
  - Many configurations can be assembled without the need for soldering or breadboarding. Just plug in the modules, and you are ready to go!
  - Our standardized Grove connector allows users to assemble Grove units with a building block approach, compared to the jumper or solder based system, it is much easier to assemble or disassemble, which simplifies the learning system for experimenting, building, and prototyping.
- Compared to the Arduino Uno, the Seeeduino V4.2:
  - Uses a micro USB to power and program the board instead of a regular USB
  - 3 onboard Grove connectors
  - 3.3 / 5 V system power switch
  - DCDC circuit instead of LDO which improves efficiency
  - Improved Circuit

ESP

The ESP8266 (ESP8266, n.d.) is a low-cost Wi-Fi microchip, with built-in TCP/IP networking software, and microcontroller capability, produced by Espressif Systems in Shanghai, China.

Through the ESP-01 module, the chip gained popularity in the maker community in August 2014.

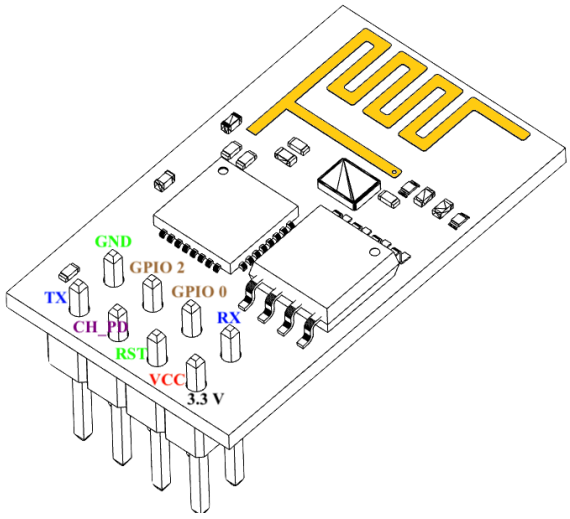


Figure 28 ESP-01 module pinout

The pinout is as follows for the common ESP-01 module:

- GND, Ground (0 V)
- GPIO 2, General-purpose input/output No. 2
- GPIO 0, General-purpose input/output No. 0

- RX, Receive data in, also GPIO3
- VCC, Voltage (+3.3 V; can handle up to 3.6 V)
- RST, Reset
- CH_PD, Chip power-down
- TX, Transmit data out, also GPIO1

With the use of Hayes-style instructions, this tiny module enables microcontrollers to join a Wi-Fi network and establish straightforward TCP/IP connections. However, at initially, there was hardly any information available in English on the chip and the orders it would receive. Many hackers were drawn to investigate the module, the chip, and the software on it as well as to translate the Chinese documentation due to the extremely low price and the fact that there were very few external components on the module, which suggested that it could eventually be very inexpensive in volume. (ESP8266 Basic Help 3.0, n.d.).

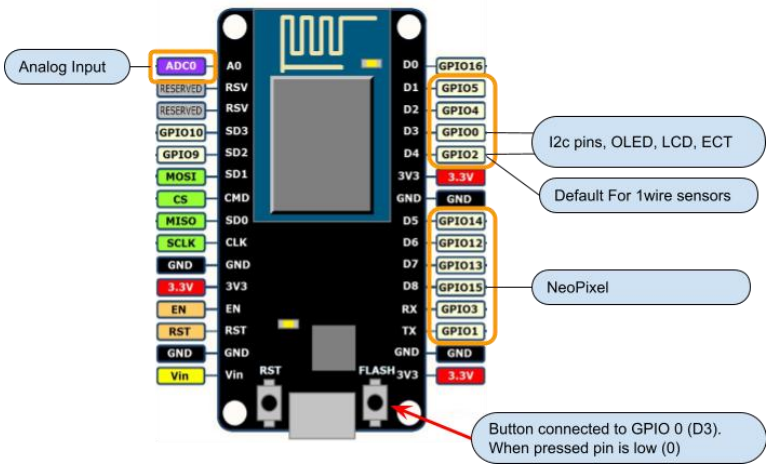


Figure 29 ESP8285 Hardware Basics (Node MCU Board)

The ESP8285 (Esp8266Basic, n.d.)is a similar chip with a built-in 1 MiB flash memory, allowing the design of single-chip devices capable of connecting via Wi-Fi.

The ESP32 family of system on a chip microcontrollers features integrated Wi-Fi and dual-mode Bluetooth and is inexpensive and low power. The Tensilica Xtensa LX6 dual-core or single-core microprocessor, Tensilica Xtensa LX7 dual-core, or a single-core RISC-V microprocessor are used in the ESP32 series, which also has integrated antenna switches, RF baluns, power amplifiers, low-noise receive amplifiers, filters, and power-management modules. Espressif Systems, a Chinese business with its headquarters in Shanghai, invented and developed the ESP32. It is the ESP8266 microcontroller's replacement.

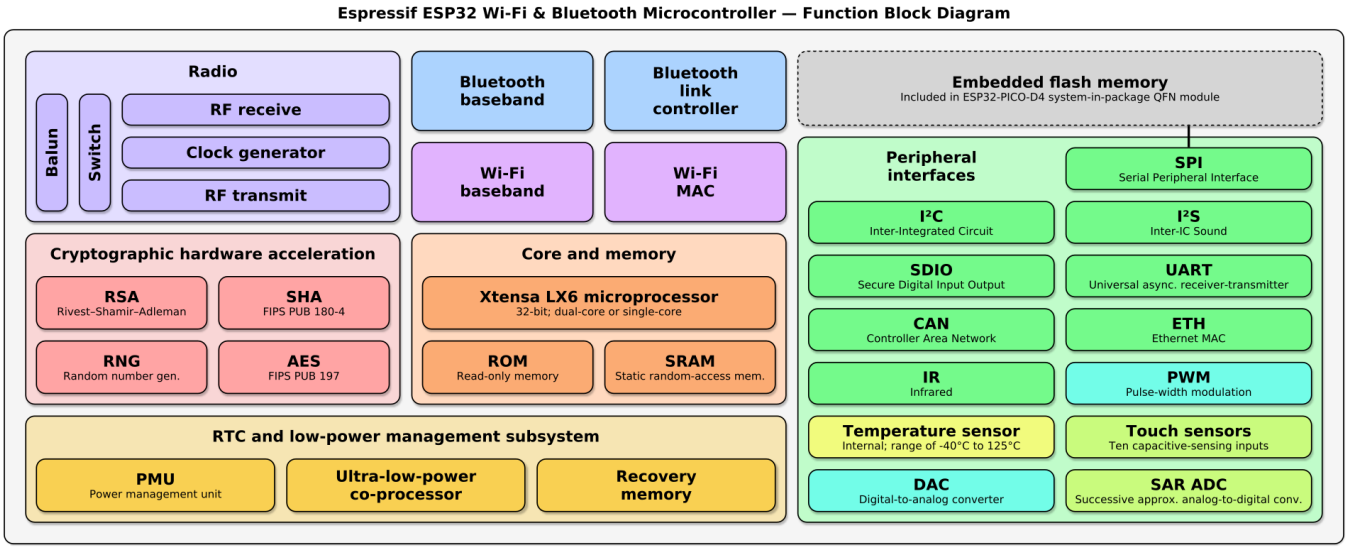


Figure 30 ESP32 function block diagram.

The ESP8266 and ESP32 series chips, modules, and development boards are Espressif's best-known goods. Systemon-chip (SoC) architecture, used by Espressif, refers to a single integrated circuit (IC) made up of several components. The ESP8266 and ESP32 Series SoCs' specs are displayed in the table.

Table 1 The Technical Specifications of the SoCs from Espressif Systems

SoC	Core	MCU	Wi-Fi	BLE	Pins	RAM	ROM	Flash
ESP32-C6	Single	160 MHz	2.4 GHz	5.0	32	400 KB RAM	384KB	--
ESP32-S3	Dual	240 MHz	2.4 GHz	5.0	56	512 KB SRAM	384 KB	--
ESP32-S2	Single	240 MHz	2.4 GHz	--	56	320 KB SRAM	128 KB	--
ESP32-C3	Single	160 MHz	2.4 GHz	5.0	32	400 KB RAM	384 KB	4 MB
ESP32-D0WD-V3	Dual	240 MHz						--
ESP32-D2WD	Dual	160 MHz						2 MB
ESP32-U4WDH	Single	160 MHz						4 MB
ESP32-S0WD	Single	160 MHz						--
ESP32-PICO-V3	Dual	240 MHz						4 MB
ESP32-PICO-D4	Dual	240 MHz	2.4 GHz	4.2	48	520 KB SRAM	448 KB	4 MB
ESP8266EX	Single	160 MHz	2.4 GHz	--	32	160 KB RAM	--	--

A Wi-Fi capable SoC that has a Cadence Tensilica L106 32-bit RISC CPU and a TCP/IP stack is the ESP8266 (2014), also known as ESP8266EX. Additionally, a memory controller with on-chip SRAM and ROM is included in it. Figure 1 depicts the ESP8266 SoC's component parts. (Esp8266, χ.χ.)

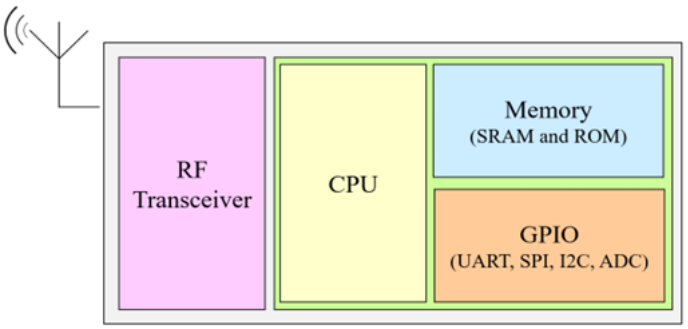


Figure 31 Building Blocks of the ESP8266 SoC

Important, The ESP8266 SoC generally uses 170 mA when operating and 80 mA when it is idle. The working voltage spans the 2.5 V to 3.6 V DC range. Additionally, it supports the 0.5 mA, 15 mA, and 0.1 mA current-based light sleep, medium sleep, and deep sleep modes. When a device "wakes up" from sleep or enters "boot up" mode, the peak working current can exceed 320 mA. The ESP8266 SoC offers a considerably reduced power consumption as compared to standard Wi-Fi adapters, ensuring a battery life of up to three months in most cases.

The ESP8266 NodeMcu (ESP8266 Arduino Core, n.d.) has 16 GPIO pins and one analog input pin shown in the image bellow. However only 10 of these GPIO pins can be used for digital input and output operations.

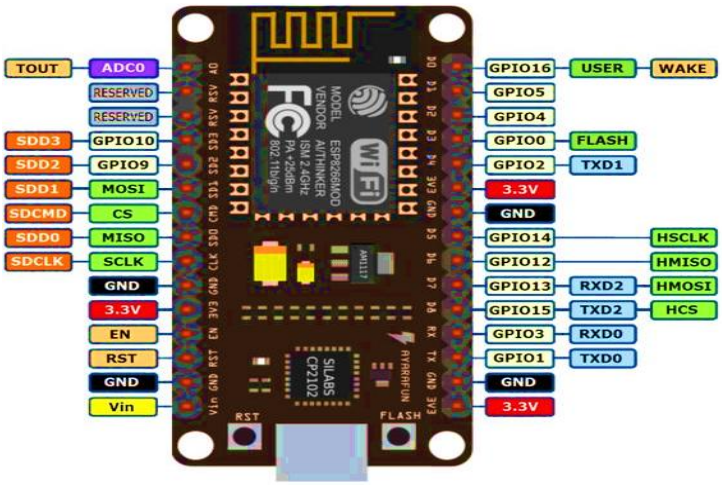


Figure 32 ESP8266 NodeMcu Pinout

Pin Functions

The ESP8266 GPIO pin numbers and the pin numbers in the Arduino IDE have a direct correspondence. To read GPIO2, call digitalRead(2) or its alias name, digitalRead. pinMode, digitalRead, and digitalWrite functions continue to function as normal (D10).

Pins are set up as INPUT during initialization. INPUT, OUTPUT, or INPUT PULLUP can be used on digital pins 0 through 15.

Pin 16 is linked to the built-in LED and has the options INPUT, OUTPUT, or INPUT PULLDOWN 16. DigitalRead(D0), DigitalRead(16), or DigitalRead(LED BUILDIN) can be used to address it.

Additional uses for pins, such as Serial, I2C, and SPI, are possible. The corresponding library often activates these functions. The popular ESP8266 NodeMcu module's pin mapping is shown in the diagram above.

AttachInterrupt routines enable support for pin interrupts. Any GPIO pin, save GPIO16, can have interrupts attached to it. Change, Rising, and Falling interrupts are supported by Arduino.

Reserved Pins

Because they are utilized to link the flash memory chip on the majority of modules, GPIO pins 6 through 11 are not depicted in this picture. The software will probably crash if you try to utilize these pins as IOs.

Also take note that some boards and modules (ESP-12ED, NodeMCU 1.0) separate pins 9 and 11. If the flash chip operates in DIO mode, these might be utilized as IO (as opposed to QIO, which is the default one).

Vin, 3V3, GND

The NodeMCU's inbuilt voltage regulator is linked to Vin, the voltage input, which allows a range of 4.75V to 10V. To 3.3V, it will be controlled. Alternately, a 3.3V external power supply can be connected directly to the NodeMcu's 3V3 pins. Other components, including LEDs, can also get their voltage via the 3V3 pin. The board's common ground is GND.

Analog Input

Users of ESP8266 have access to a single ADC channel. It may be used to read the module supply voltage or the voltage at the ADC pin (VCC).

Use analogRead to read the external voltage applied to the ADC pin (A0). Range of input voltage is 0 to 1.0V.

Use ESP.getVcc() to get the VCC voltage; leave the ADC pin unconnected. Additionally, the drawing has to include the following line:

ADC MODE(ADC VCC); This line must come after the #include lines in your sketch and before any functions.

Analog Output

Software PWM is activated on the specified pin via analogWrite(pin,value). Pins 0 through 16 can be utilized for PWM. To stop PWM from operating on the pin, use analogWrite(pin,0). value may be in the range of 0 to PWMRANGE, which is by default equal to 1023. The PWM duty cycle is set to 0 percent, 50 percent, and 100 percent, respectively, by values of 0, 512, and 1023. By invoking analogWriteRange(new range), the PWM range can optionally be altered.

The default PWM frequency is 1kHz. To change the frequency, call analogWriteFreq(new frequency). [Hz] is used for unit representation. An illustration is analogWriteFreq(32500); /which sets the PWM frequency to 32.5 kHz.

Microbit

The Micro Bit (Micro Bit, n.d.) was designed to encourage children to get actively involved in writing software for computers and building new things, rather than being consumers of media. The BBC created the open source Micro Bit, an ARM-based embedded system, for use in computer education in the UK. The plan to provide 1 million gadgets to students in the UK was initially revealed on March 12, 2015, at the start of the BBC's Make It Digital campaign.

An ARM Cortex-M0 processor, accelerometer and magnetometer sensors, Bluetooth and USB connectivity, a display with 25 LEDs, two programmable buttons, and the ability to be powered by USB or an external battery pack are all features of the device, which is described as being half the size of a credit card. Five ring connections, which are a component of a larger 25-pin edge connector, are used for the device's inputs and outputs. A Cortex-M4F microcontroller with greater memory and new functions was featured in the introduction of the v2 board in October 2020, which is visually quite similar to the original board.

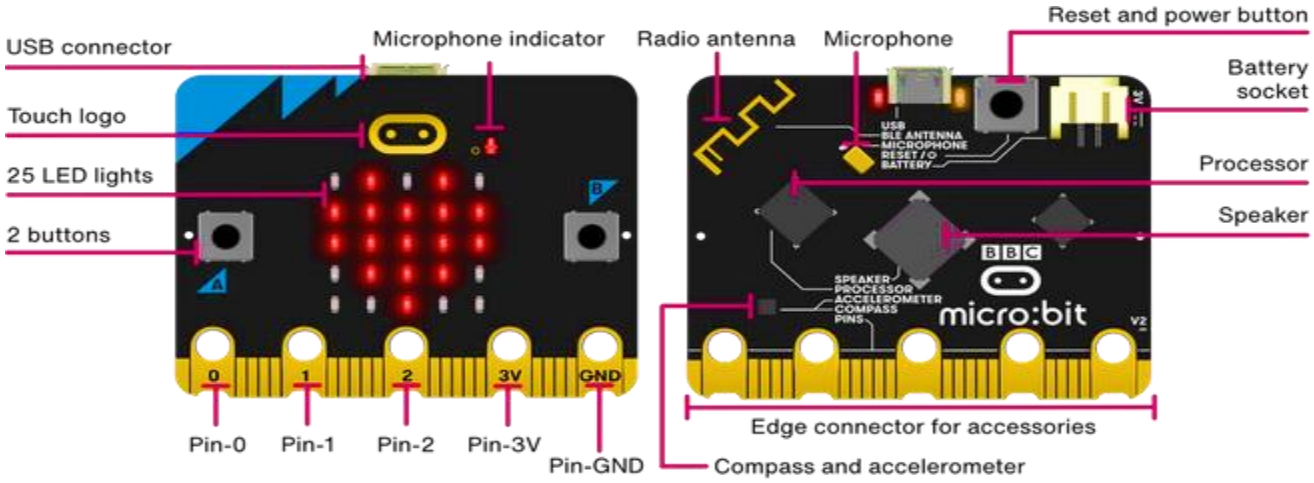


Figure 33 micro:bit

The released v2, includes:

- Nordic nRF52833 – 64 MHz 32-bit ARM Cortex-M4 microcontroller, 512 KB flash memory, 128 KB static RAM, 2.4 GHz Bluetooth low energy wireless networking provided by Nordic S113 SoftDevice, integrated temperature sensor.
- NXP/Freescale KL27Z – 48 MHz ARM Cortex-M0+ core microcontroller, preprogrammed as a full-speed USB 2.0 controller, used as a communication interface between USB and the CPU.
- Either ST LSM303 or NXP FXOS8700 – 3-axis combined accelerometer and magnetometer sensor via I^2C-bus.
- Knowles MEMS microphone with a built-in LED indicator.
- Jiangsu Huaneng MLT-8530 magnetic speaker.
- MicroUSB connector, JST PH battery connector, 25-pin edge connector.
- Display consisting of 25 LEDs in a 5×5 matrix.
- Three tactile pushbuttons (two for applications, one for reset) and a touch sensor button.

In micro:bit v2, the reset button can be used to turn the board off by holding it for 3 seconds.

There are three official code editors on the micro:bit foundation web site:

- Microsoft MakeCode
- MicroPython https://python.microbit.org/
- Scratch

Accessories

Sensors

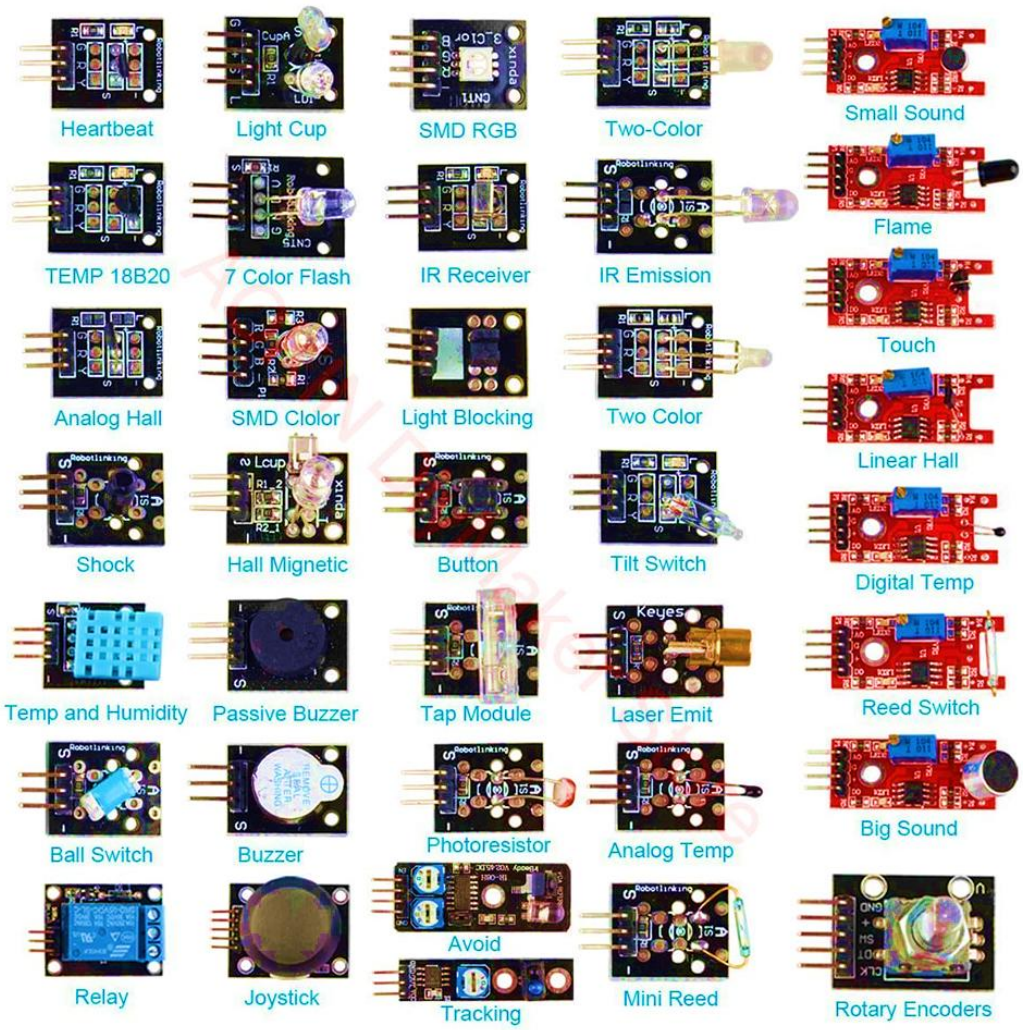


Figure 34 For Arduino 35 In 1 Sensors Modules Starter Kit

1. Soil module
2. Infrared sensor receiver module
3. Laser head sensor module
4. Temperature and humidity sensor module
5. Infrared emission sensor module
6. 5V relay module
7. Gyro Module
8. finger detect heartbeat module
9. Microphone sensitivity sensor module
10. Metal touch sensor module
11. Flame sensor module
12. 3-color LED module
13. Hunt sensor module
14. Linear magnetic Hall sensors
15. Rotary encoder modules
16. Active buzzer module
17. Magic Light Cup modules
18. Small passive buzzer module
19. Digital temperature sensor module
20. Tilt switch module
21. Analogy Holzer magnetic sensor
22. Ultrasonic module
23. Mercury opening module
24. Hall magnetic sensor module

25. RGB LED SMD module
26. Mini Reed module
27. Bicolor LED common cathode module 3MM
28. Smart car avoid obstacle sensor infrared sensor photoelectric switch
29. Key switch module
30. Photoresistor module
31. Breadboard power module
32. hit sensor module
33. Temperature sensor module
34. Vibration switch module
35. Microphone sound sensor module
36. Large reed module
37. Two-color LED module
38. Optical breaking module
39. Temperature sensor module
40. MP1584EN buck module
41. SD card reader module
42. 1 xPS2 Joystick game controller module
43. Automatically flashing LED module
44. DS1302 clock module
45. Water level module

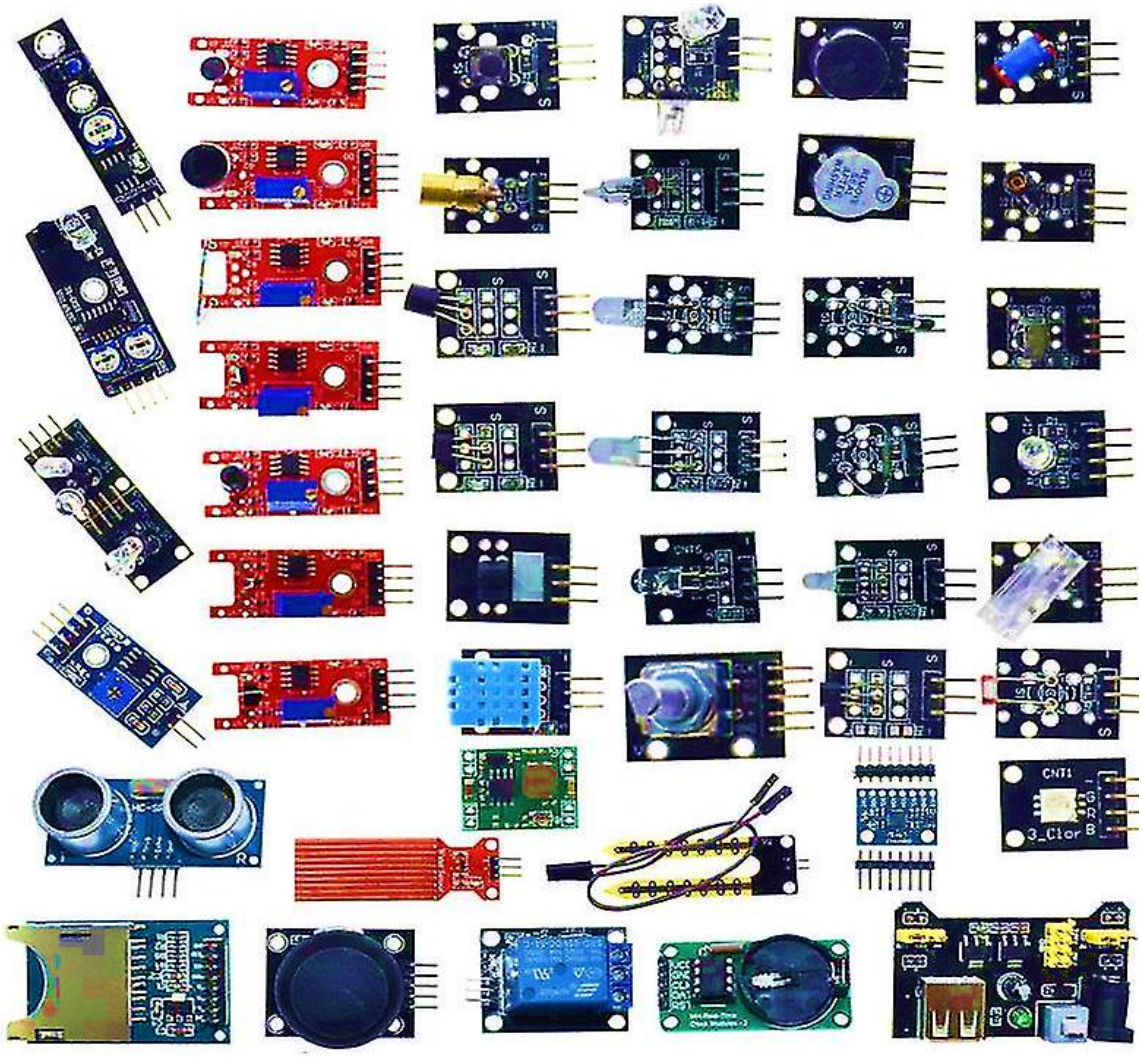


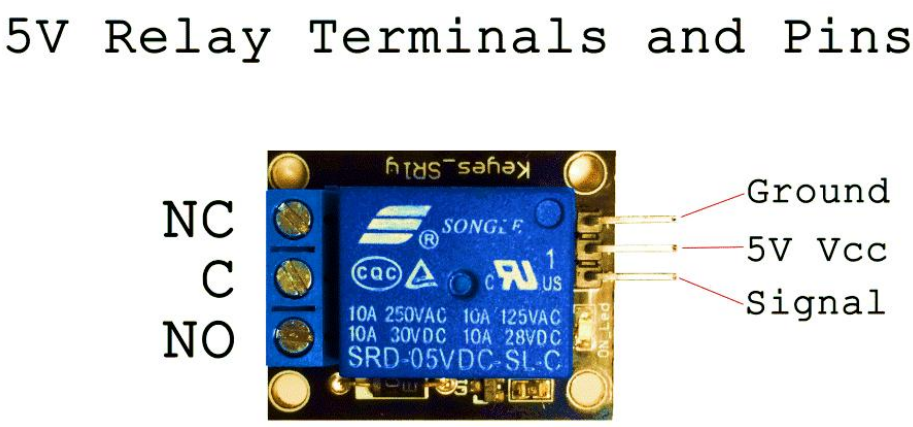
Figure 35 Arduino 45 In 1 Sensors Modules Starter Kit

Relay

An electrically controlled switch is a relay. It is made up of a set of working contact terminals and a set of input terminals for one or more control signals. Any number of connections in different contact

configurations, such as make contacts, break contacts, or combinations of both, may be included on the switch.

Relays are employed when several circuits need to be controlled by a single signal or when a circuit has to be controlled by a separate, low-power signal.



- NC: Normally closed 120-240V terminal
- NO: Normally open 120-240V terminal
- C: Common terminal
- Ground: Connects to the ground pin on the Arduino
- 5V Vcc: Connects the Arduino's 5V pin
- Signal: Carries the trigger signal from the Arduino that activates the relay

A 120–240V switch coupled to an electromagnet is located within the relay. The electromagnet charges up and pushes the switch contacts open or closed when the relay gets a HIGH signal at the signal pin.

Normally Open Vs. Normally Closed

The relay has ordinarily open (NO) and usually closed (NC) electrical connections within (NC). Choosing one will depend on whether you want the 5V signal to activate or deactivate the switch. In both arrangements, the relay's common (C) terminal is where the 120-240V supply current enters. Use the NO terminal to operate the usually open connections. Use the NC terminal to operate the usually closed contacts.

Normally Open

When the relay gets a HIGH signal in the typically open mode, the 120-240V switch shuts, allowing electricity to flow from the C terminal to the NO terminal. The relay is turned off and the current is stopped by a LOW signal. Therefore, utilize the usually open terminal if you want the HIGH signal to switch on the relay:

Normally Closed

A HIGH signal opens the switch in the typically closed form, interrupting the 120–240V current. When the switch is closed by a LOW signal, current can go from the C terminal to the NC terminal. Use the usually closed terminal as a result if you want the HIGH signal to cut off the 120-240V current:

Motors

An electrical device that transforms electrical energy into mechanical energy is an electric motor (Electric motor, n.d.). The majority of electric motors work by creating force in the form of torque imparted to the motor shaft through the interplay of the magnetic field of the motor and electric current in a wire winding. Although an electric generator and an electric motor are technically equivalent, an electric generator uses a reversed power flow to transform mechanical energy into electrical energy.

Electric motors can be powered by alternating current (AC) sources like a power grid, inverters, or electrical generators or by direct current (DC) sources like batteries or rectifiers.

There are three different type of motors –

- DC motor
- Servo motor
- Stepper motor

DC motor

The most typical kind of motor is a DC motor, or direct current motor. Typically, DC motors only have two leads: a positive lead and a negative lead. The motor will turn if you connect these two lines straight to a battery. The motor will turn in the other direction if the leads are switched.

Warning – Do not drive the motor directly from Arduino board pins. This may damage the board. Use a driver Circuit or an IC.

We will divide this chapter into three parts –

- Just make your motor spin
- Control motor speed
- Control the direction of the spin of DC motor

Servo motor

A tiny machine with an output shaft is called a servo motor. Sending the servo a coded signal allows for precise angular positioning of this shaft. The servo will keep the shaft in its current angular position so long as the coded signal is present on the input line. The angular location of the shaft varies if the coded signal changes. In real life, radio-controlled airplanes employ servos to position control surfaces like the elevators and rudders. In addition to robots, they are utilized in puppets and radio-controlled automobiles.

Robotics makes extensive use of servos. The motors are compact, equipped with internal control electronics, and surprisingly powerful considering their size. The torque of a typical servo, such the Futaba S-148, is 42 oz/inches, which is substantial for its size. Additionally, it uses energy in proportion to the mechanical load. Therefore, a servo that is not heavily loaded uses less energy.

The following image displays a servo motor's internal workings. The casing, the motor, several gears, and the control circuits are all visible. The three lines that link to the outer world are also visible. The white wire serves as the control wire while the other is for power (+5 volts) and ground.

Working of a Servo Motor

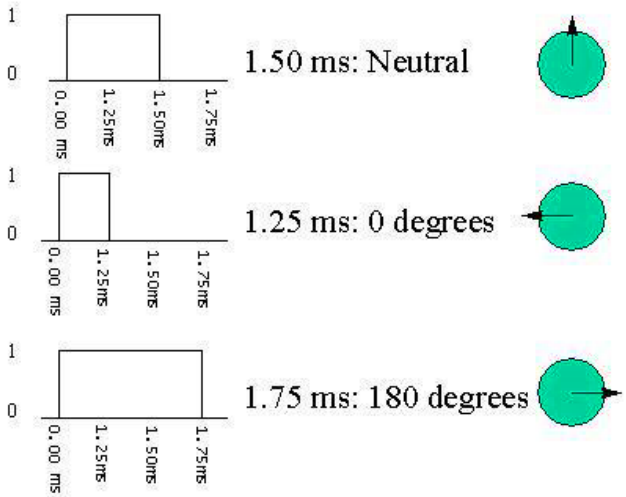
A potentiometer (also known as a variable resistor, or pot) is attached to the output shaft of the servo motor along with a few control circuits. The pot is seen on the circuit board's right side in the image above. This pot enables the control circuitry to keep track of the servo motor's current angle.

The motor turns off if the shaft is angled correctly. The motor will be turned until it is at the desired angle if the circuit determines that the angle is incorrect. The servo's output shaft has a range of motion of around 180 degrees. It varies depending on the manufacturer, but typically it is in the 210-degree range. An angular motion from 0 to 180 degrees may be controlled by a standard servo. A mechanical stop attached to the main output gear prevents it from spinning any farther mechanically.

The engine is powered in direct proportion to the required travel distance. Therefore, the motor will operate at maximum speed if the shaft needs to rotate a considerable distance. The motor will operate at a slower speed if only a little quantity of rotation is required. Proportional control is the name for this.

How Do You Communicate the Angle at Which the Servo Should Turn?

The angle is communicated through the control wire. The length of a pulse supplied to the control wire determines the angle. The term for this is pulse coded modulation. Every 20 milliseconds, the servo anticipates seeing a pulse (.02 seconds). How far the motor turns will depend on how long the pulse lasts. For instance, a 1.5 millisecond pulse will cause the motor to turn 90 degrees (often called as the neutral position). The motor will rotate the shaft closer to 0 degrees if the pulse lasts less than 1.5 milliseconds. The shaft rotates more closely to 180 degrees if the pulse lasts longer than 1.5 milliseconds.



Stepper motor

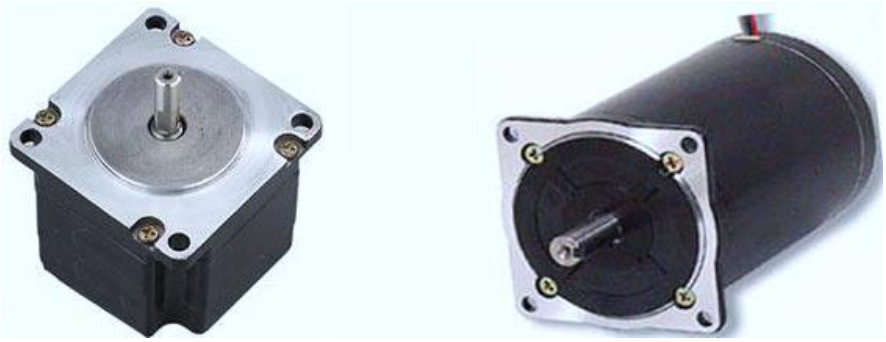
A brushless, synchronous motor that separates a whole revolution into many steps is called a stepper motor or a step motor. A step motor rotates at distinct step angles as opposed to a brushless DC motor, which rotates continuously when a constant DC voltage is provided to it.

As a consequence, the stepper motors are created with 12, 24, 72, 144, 180, and 200 steps per revolution, yielding stepping angles of 30, 15, 5, 2.5, 2, and 1.8 degrees each step. You may control the stepper motor with or without feedback.

Think of an RC airplane with a motor. The motor rapidly rotates in either direction. The motor's power can be adjusted to change the speed, but the propeller cannot be made to stop at a certain location.

Imagine a printer now. A printer contains many motors and other moving elements. One of these motors serves as the paper feed, rotating the paper as the ink is being printed on it. To print the next line of text or the next line of a picture, this motor must be able to move the paper a precise distance.

The print head is moved back and forth by a different motor that is connected to a threaded rod. Once more, the threaded rod needs to be adjusted precisely how far for each letter to be printed. The stepper motors are useful in this situation.



How a Stepper Motor Works?

While a stepper motor may spin in exact increments, a normal DC motor can only rotate in one direction.

Stepper motors have the ability to turn precisely the necessary number of degrees (or steps). This offers you complete control over the motor, enabling you to direct it precisely where you want it to go and maintain that position. It does this by briefly powering the motor's internal coils. The drawback is that in order to maintain the motor in the desired position, electricity is required continuously.

For the time being, all you need to know is that you may order a stepper motor to move a specific number of steps in one direction or the other, as well as the speed at which to move those steps. Stepper motors come in a wide variety. Other motors and drivers that are not addressed in this tutorial can be utilized by inferring how to use them from the methods provided here. It is always advised to refer to the datasheets and manuals for the motors and drivers that are unique to the models you own.

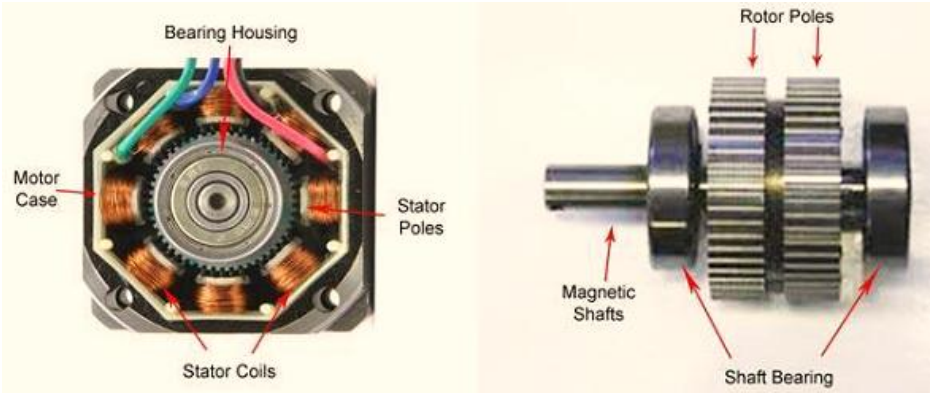


Figure 36 28BYJ-48-5V 4 Phase 5 Wire DC 5V Stepper Motor

Other drives motor

We require specialized drivers to successfully operate the various types of motors with microcontrollers and microprocessors.

Figure 37 DC 6V - 28V 3A PWM Regulator Motor Speed - Controller Switch

Figure 38 Stepper Motor Driver Controller Board L298N Dual H Bridge

Figure 39 5V 28BYJ-48-5V 4-Phase Stepper Stepper Motor + Driver Board ULN2003

Figure 40 28BYJ-48-5V Stepper Motor + DC 5V ULN2003 Stepper Motor Board 4-phase

Figure 41 DRV8825 Stepper Driver Module with Heatsink

Figure 42 3D Printer A4988 Reprap Stepper Motor Driver Module

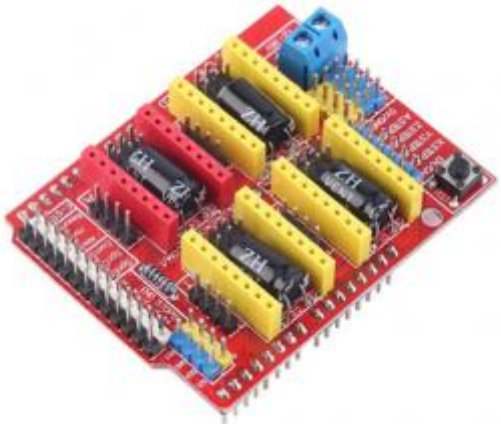


Figure 43 V3 Engraver - 3D Printer CNC Shield & Expansion Board A4988 Driver

Liquid-crystal display (Lcd) and OLED

A flat-panel display or other electronically controlled optical device that makes use of polarizers and the light-modulating capabilities of liquid crystals is known as a liquid-crystal display (LCD). Liquid crystals

don't directly generate light; instead, they create color or monochromatic pictures via a backlight or reflector. There are LCDs that can show random pictures (as on a general-purpose computer display) or fixed displays with little information that may be seen or concealed.

The LiquidCrystal library allows you to control LCD displays that are compatible with the Hitachi HD44780 driver. There are many of them out there, and you can usually tell them by the 16-pin interface.

Output of the sketch on a 16x2 LCD

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

- A **register select (RS) pin** that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.
- A **Read/Write (R/W) pin** that selects reading mode or writing mode
- An **Enable pin** that enables writing to the registers
- 8 **data pins (D0 -D7)**. The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.

Additionally, you may power the LCD, adjust the display contrast, and switch on and off the LED backlight using the power supply pins (+5V and GND), LED backlight (Bklt+ and BKlt-), and display contrast (Vo) pins.

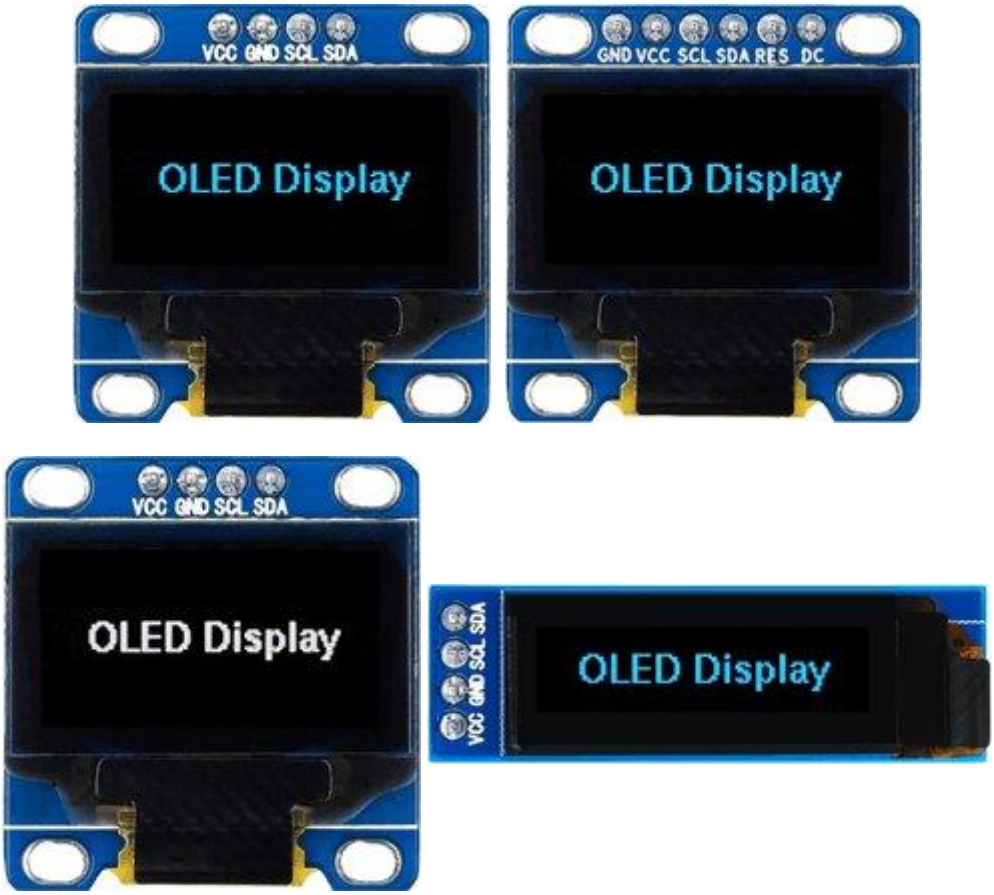
Data that make up the desired image are placed in the data registers as part of the control process for the display, and instructions are subsequently placed in the instruction register. This is sped up for you by the LiquidCrystal Library so you don't need to know the low-level commands.

There are two control modes available for the Hitachi-compatible LCDs: 4-bit and 8-bit. The Arduino needs seven I/O pins for the 4-bit mode and eleven for the 8-bit mode. Example demonstrates how to operate a 16x2 LCD in 4-bit mode as practically anything for displaying text on the screen can be done in this form.

OLED

At the heart of the module is a powerful single-chip CMOS OLED driver controller – SSD1306. It can communicate with the microcontroller via I2C and SPI. Because the SSD1306 controller is so versatile, the module

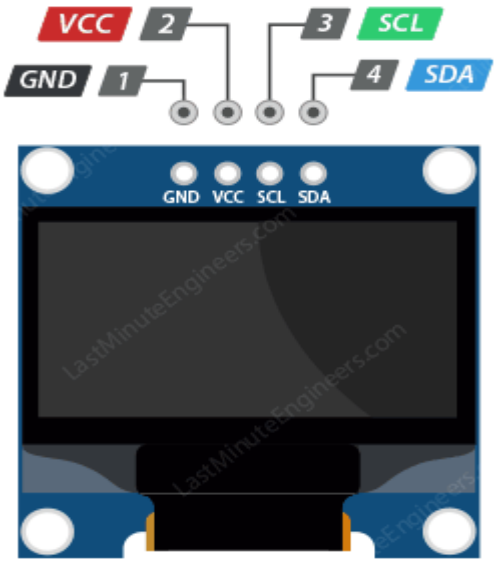
comes in different sizes and colors, such as 128×64, 128×32, with white OLEDs, blue OLEDs, and dual-color OLEDs. The good news is that these displays are all interchangeable.



OLED Display Module Pinout

Before we get into hookup and example code, let's look at its pinout.

I2C OLED Display Module



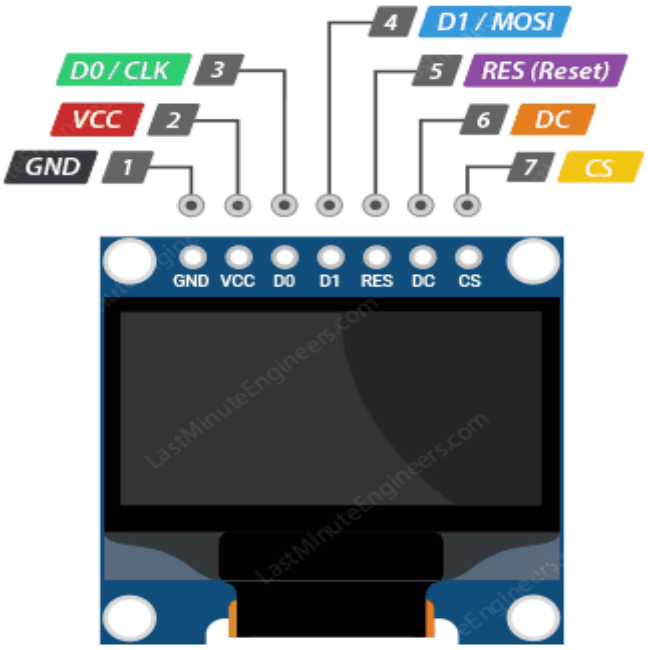
GND is the ground pin.

VCC is the power supply for the display, which we connect to the 5V pin on the Arduino.

SCL is a serial clock pin for the I2C interface.

SDA is a serial data pin for the I2C interface.

SPI OLED Display Module



GND is the ground pin.

VCC is the power supply for the display, which we connect to the 5V pin on the Arduino.

D0 / CLK is the SPI Clock pin. It's an input to the chip.

D1 / MOSI is the Serial Data In pin, for data sent from your microcontroller to the display.

RES (Reset) pin resets the internal buffer of the OLED driver.

DC (Data/Command) is used by the library to separate the commands (such as setting the cursor to a specific location, clearing the screen, etc.) from the data.

CS is the Chip Select pin.

Chapter 4: Tools Designing and Programming IoT Applications

Tinkercad

Tinkercad (Tinkercad, n.d.) is a free-of-charge, online 3D modeling program that runs in a web browser (https://www.tinkercad.com/). Since it became available in 2011 it has become a popular platform for creating models for 3D printing as well as an entry-level introduction to constructive solid geometry in schools.

History

Former Google engineers Kai Backman and Mikko Mononen established Tinkercad with the intention of making 3D modeling, particularly the creation of tangible objects, available to everyone and enabling users to submit their designs under a Creative Commons license. A web-based 3D modeling application called Tinkercad was introduced in 2011 for browsers that support WebGL, and the business moved its headquarters to San Francisco in 2012. More than 100,000 3D drawings were published by users as of 2012. At a Maker Faire in May 2013, Autodesk made an announcement about buying Tinkercad. Users of the soon-to-be decommissioned 123D Sculpt software were advised by Autodesk to switch to Tinkercad in March 2017. (or Maya LT). The "Electronics Lab" for 123D Circuits (Circuits.io) was terminated by Autodesk in May. The functions of the application were included into Tinkercad.

Concept

Model construction in Tinkercad is done using a condensed version of constructive solid geometry. Primitive forms that are either "solid" or "hole" make up a pattern. New forms can be made by combining solids and holes, and these new shapes can then be given the solid or hole attribute. Using a built-in JavaScript editor, a user may produce unique form generators in addition to the basic library of primitive shapes.

Circuits

A browser-based electronic circuit simulator for an Arduino Uno, Micro Bit board, or ATtiny chip is available in Tinkercad's Circuits area. CodeBlocks, which are graphical code fragments that can be assembled by dragging them around with the mouse, may be used to create the code. Code text programming is also a possibility. Although a circuit may be constructed using components, there exist "Starters" that are fully functional circuits with programming.

Libraries for several components, like the Adafruit Neopixel library, the Arduino Servo library, and a library for an I2C display, are available with Tinkercad. Other libraries cannot be chosen or uploaded. The circuit may have completely simulated analog components. Although Tinkercad is an easy introduction into programming and electronics, it has features for advanced users:

1. Multiple boards can be simulated at the same time. For example, two Arduino boards communicating with each other.
2. The analog circuit can be very complex.

Interface Circuits

The list of available tools, sensors, and applications is on the right. There are buttons for codes and simulations above the list (see in red circle). On the main surface, our apps are designed visually.

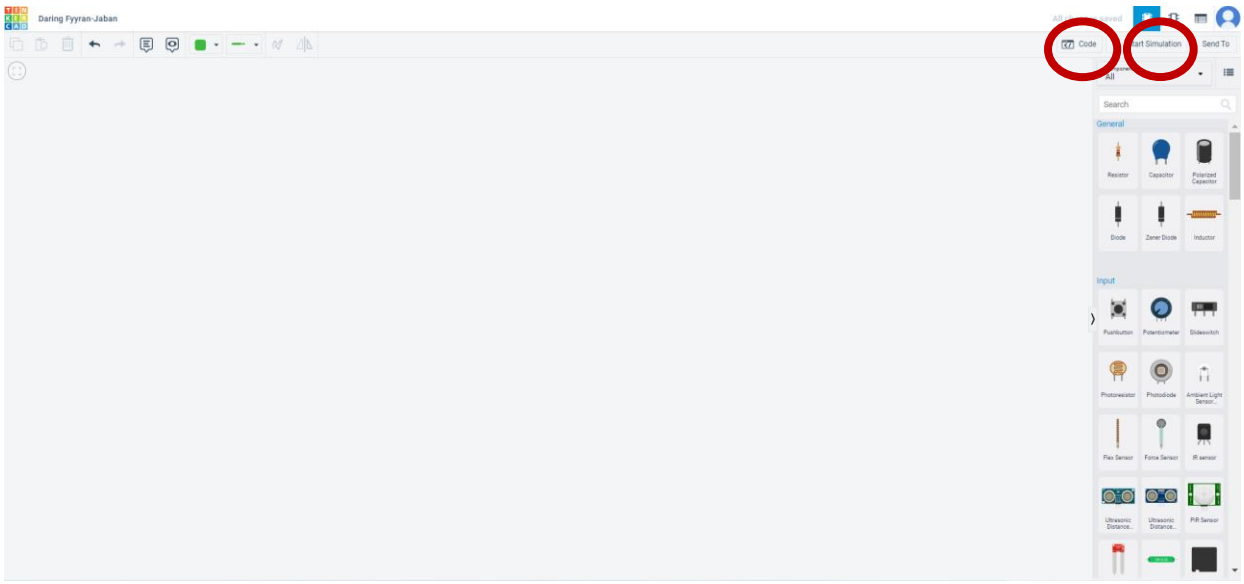
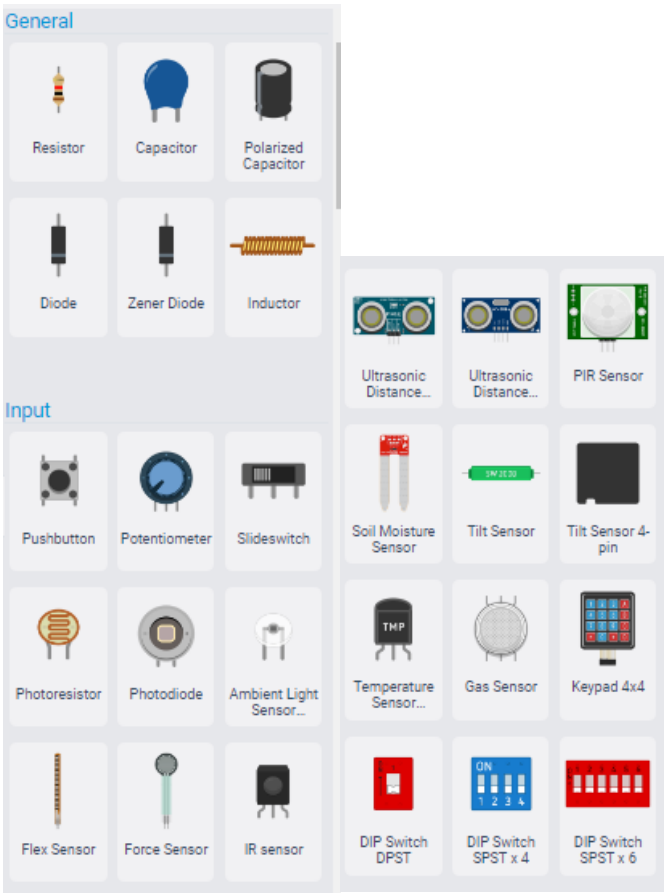


Figura 1 Interface Circuits

Tools- Sensors Input

The general tools and input sensors listed below can be used to build sophisticated applications.



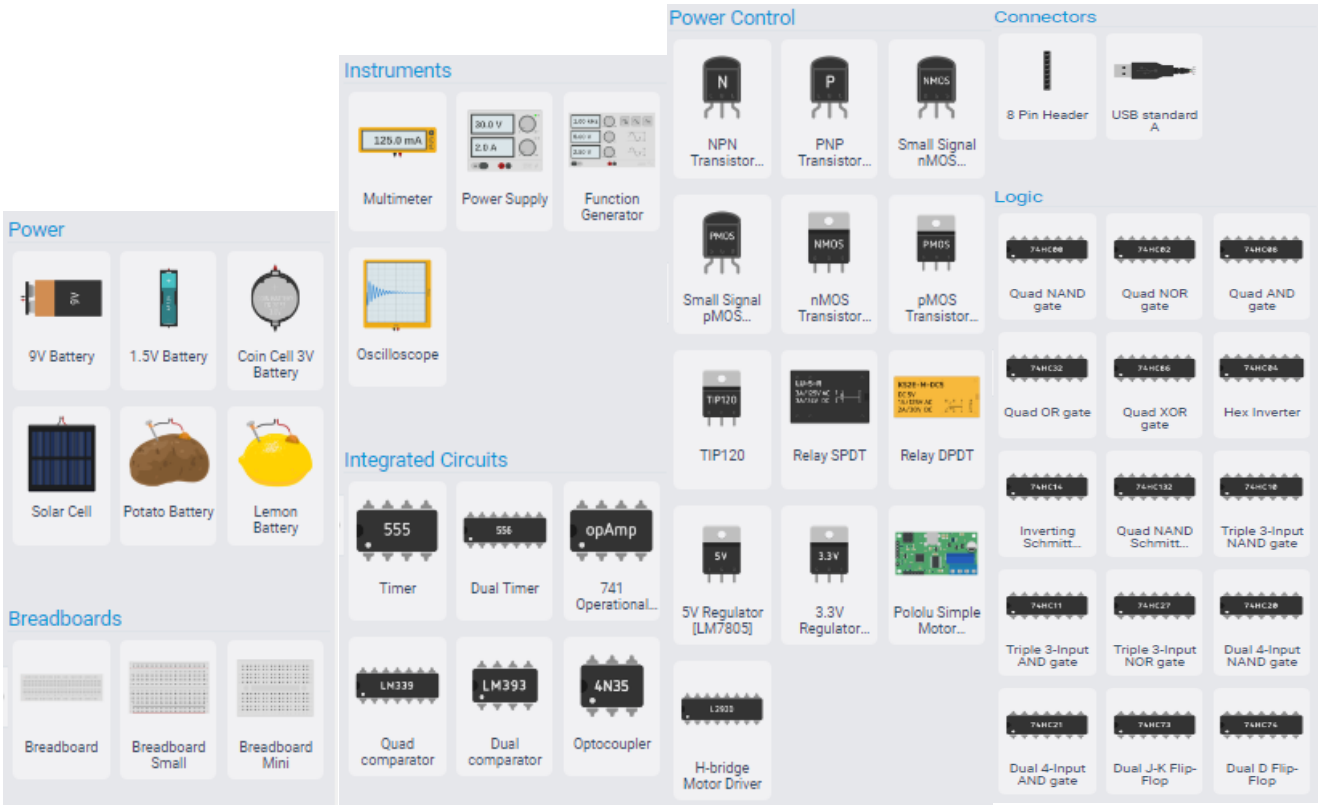
Tools- Sensors Output

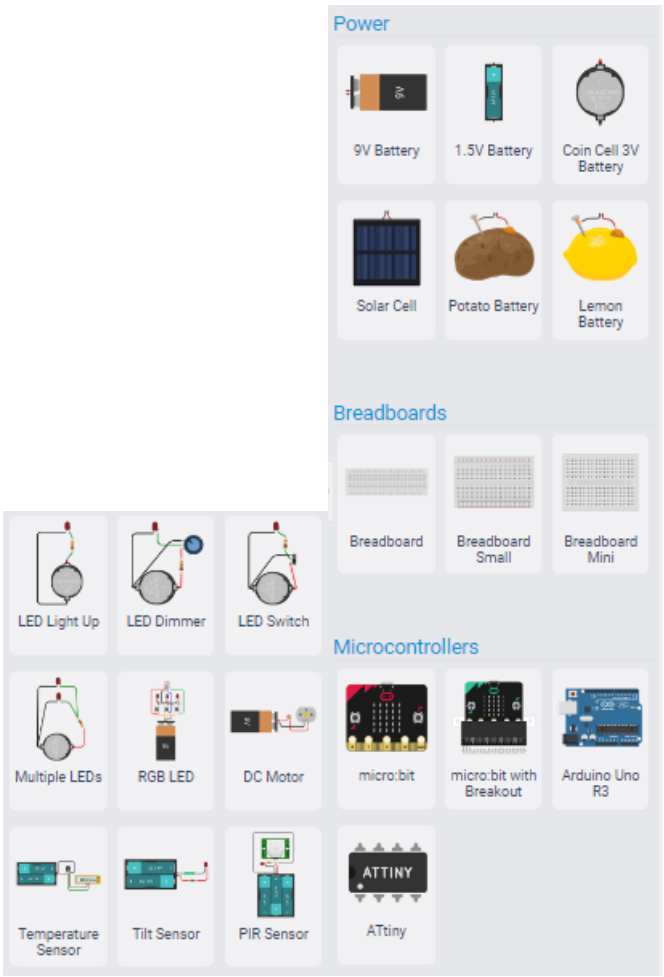
Below are some examples of output tools and sensors that we may use to extract an IoT application's activities and build simple, real-world applications by fusing inputs and outputs.

Other Tools- Sensors

Finally we see a series of tools with which we can simulate simple electricity experiments in Physics, such as:

- Power and Breadboards





Application with Arduino

Below we see a number of

Application Moisture with Arduino

Select Application Moisture with Arduino, drag and drop in central desktop

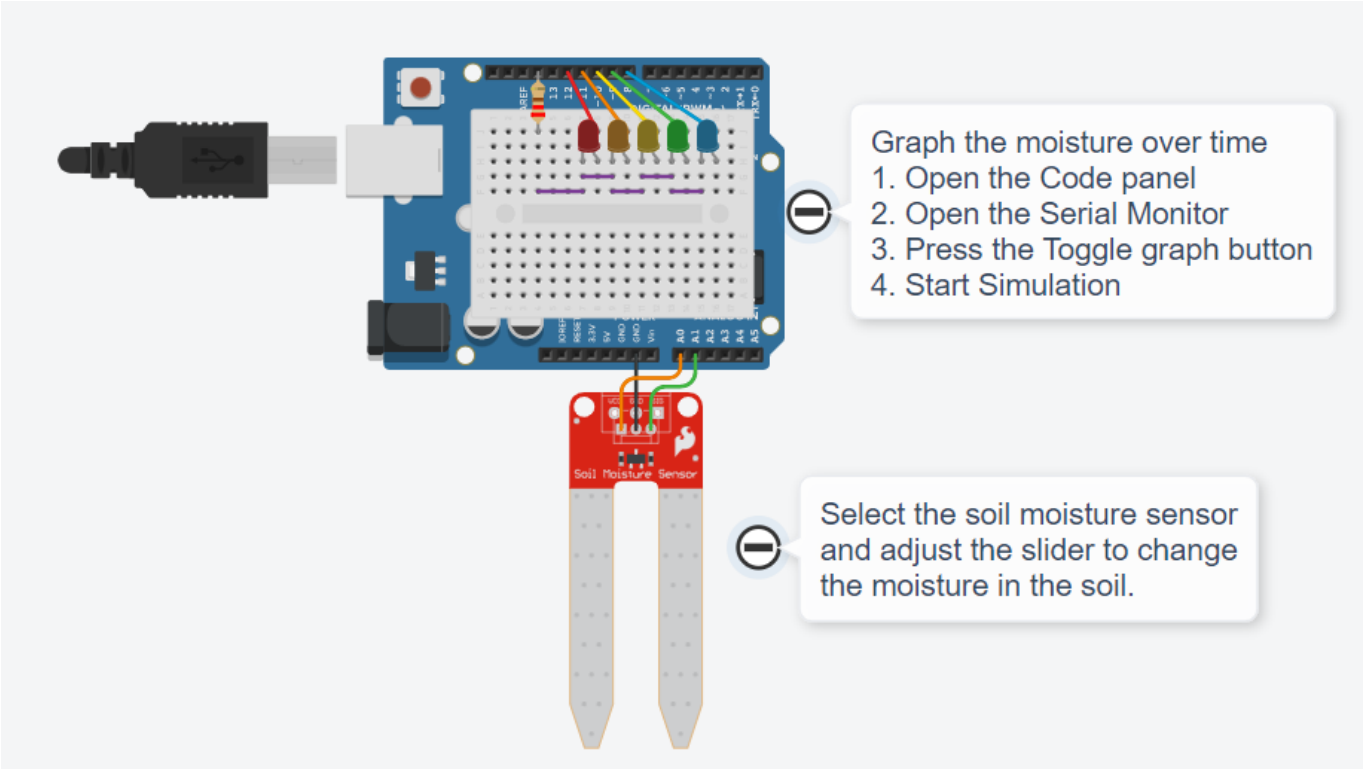


Figura 2 Application Moisture with Arduino

Pressing his button Code, see with the code that is hidden behind the app in Blocks:

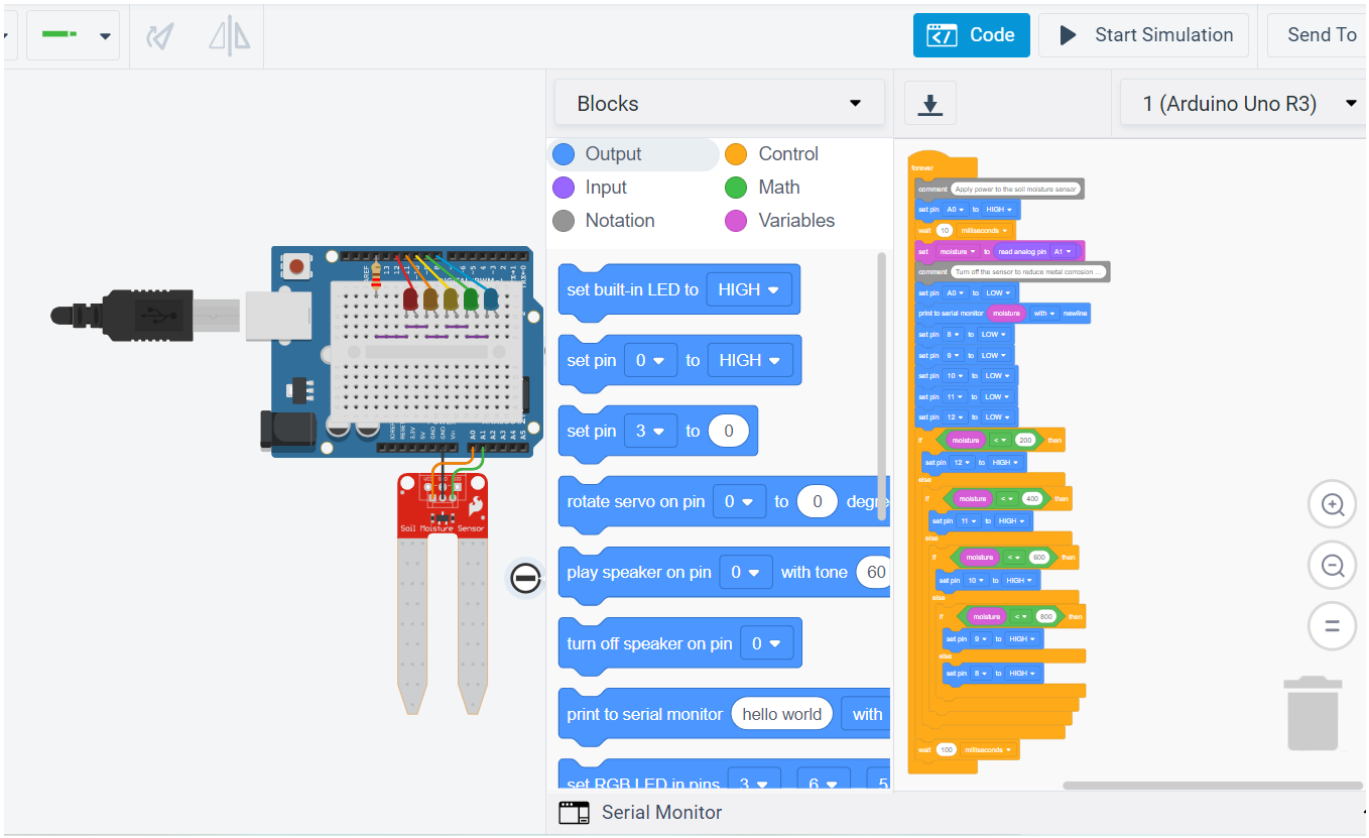
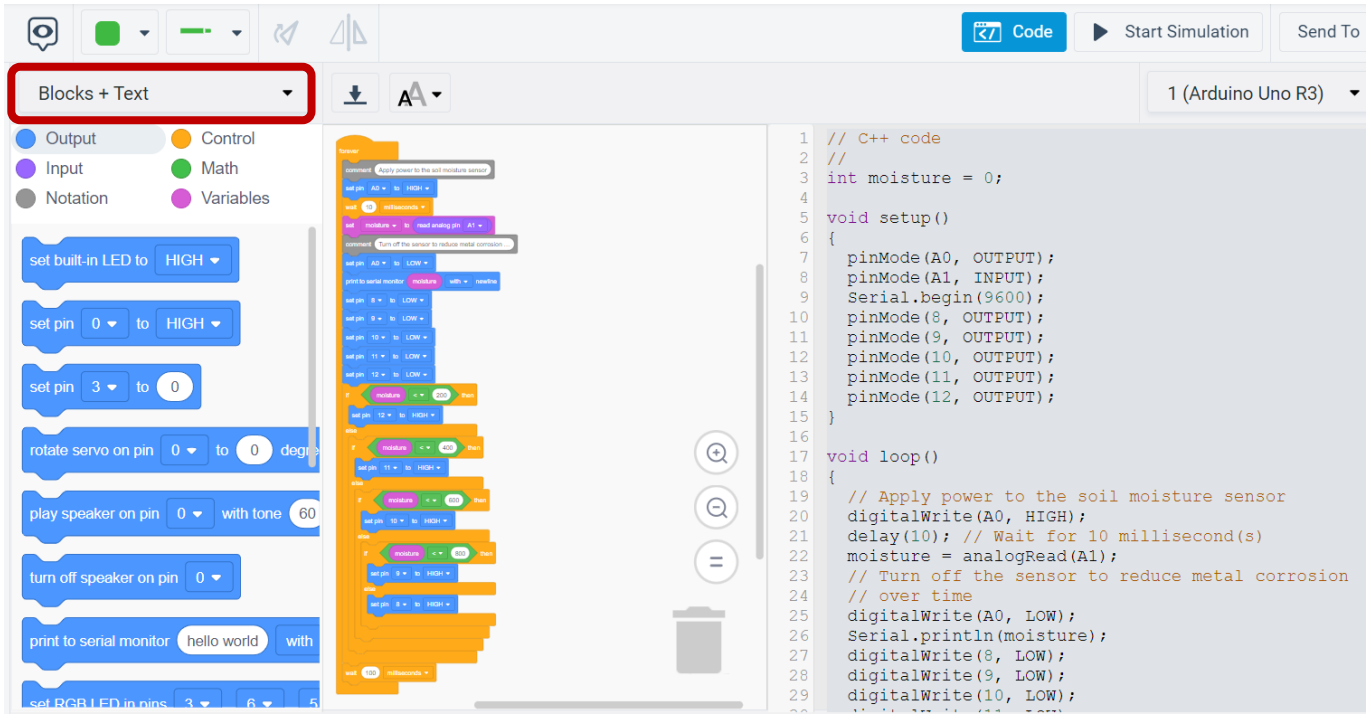
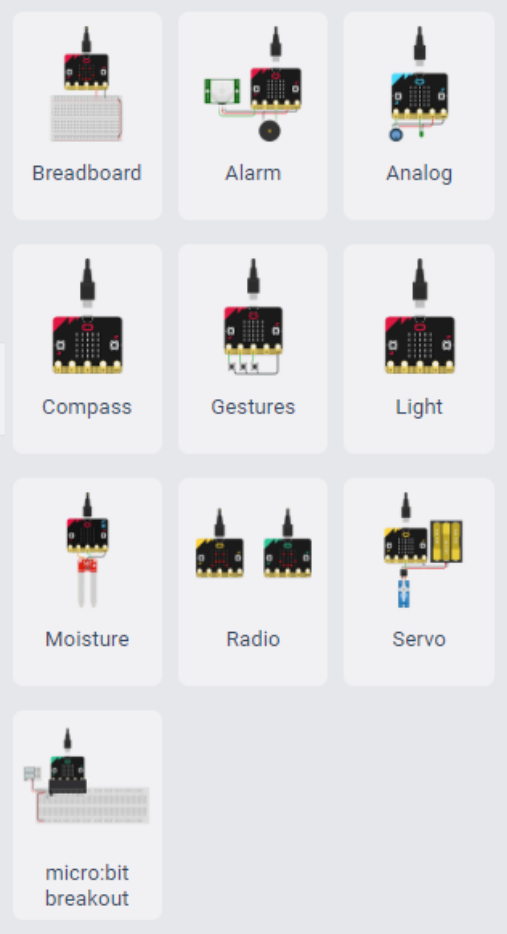


Figura 3 Blocks Code

Select Blocks+Text to see the code and to C++:



Application with Micro:bit



Application Temperature and Servo with Micro:bit

Select Application Temperature and Servo with Micro:bit, drag and drop in central desktop

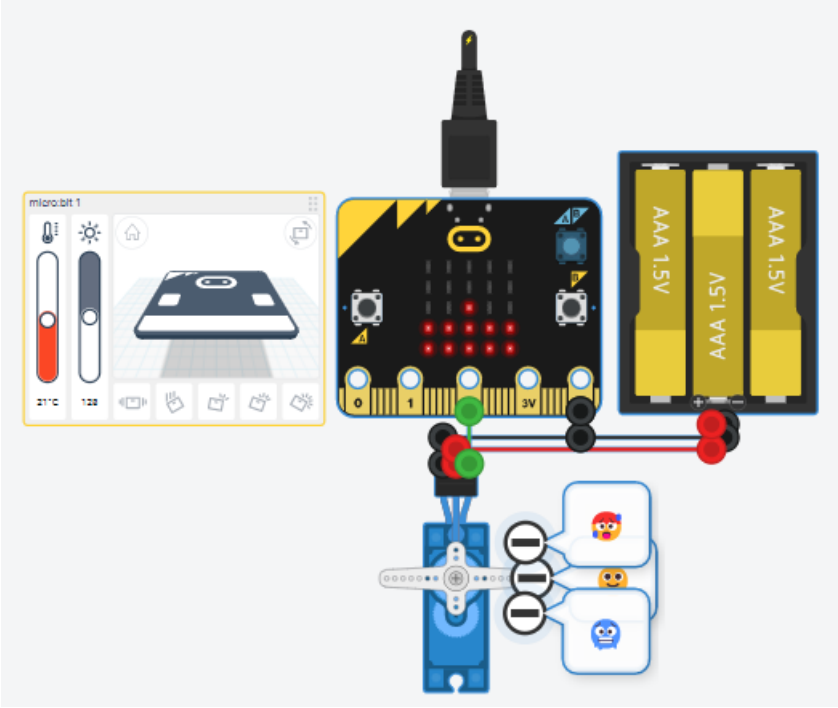
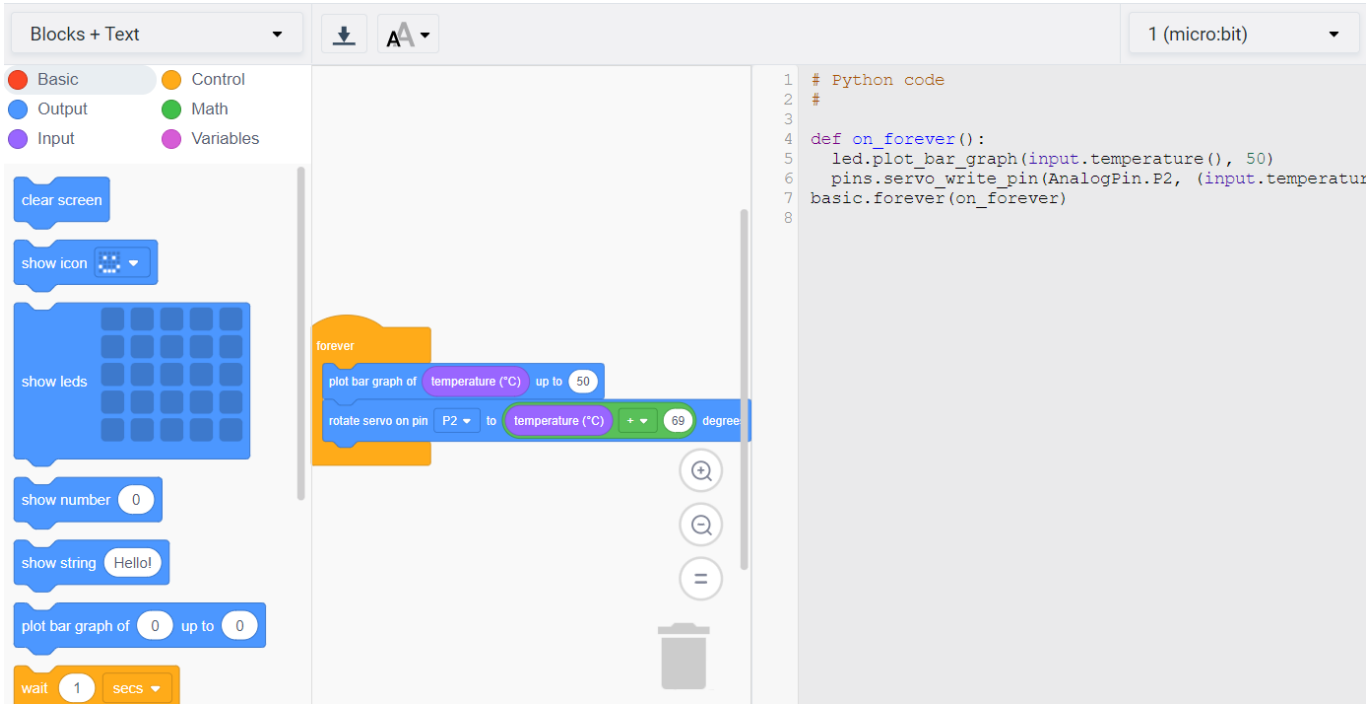


Figura 4 Application Temperature and Servo with Micro:bit

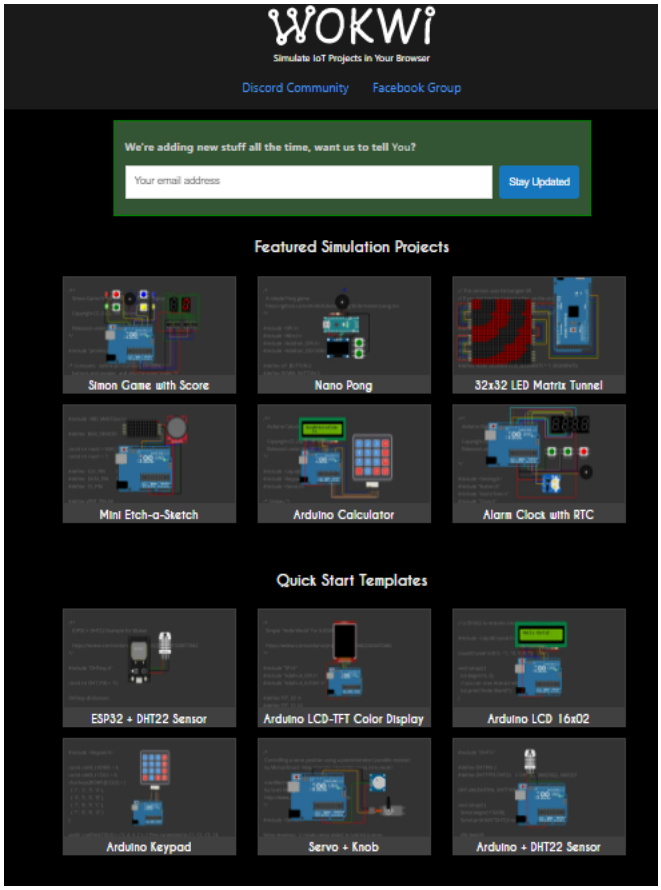
Pressing his button Code, see with the code that is hidden behind the app in Blocks (select Blocks+Text to see the code and to C++):



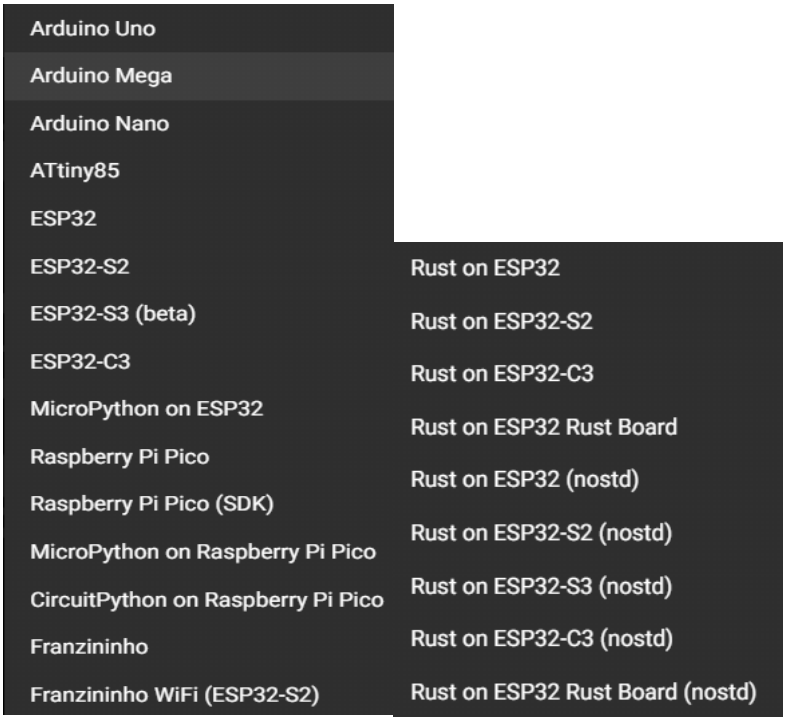
Wokwi

Wokwi is an online Electronics simulator. You can use it to simulate Arduino, ESP32, and many other popular boards, parts and sensors.

Select the web page https://wokwi.com/ find many applications for Arduino, Esp, Pico, Raspberry



Start the projects https://wokwi.com/dashboard/projects you choose



Interactive Diagram Editor

Using the interactive editing tools in the diagram editor, you may add components to the simulation and specify the relationships between them. It's a practical substitute for directly modifying the diagram.json file.

Editing parts

Adding a part

To add a new part, click on the purple "+" button at the top of the diagram editor.

You'll notice a list of available components on a menu. Pick a section to add it to. You may drag the component to the required place once it is inserted at position (0, 0).

The menu does not presently provide access to every component. For instance, MCU boards and microcontrollers like the ATtiny85 and Arduino Nano are absent. These components may still be added by changing the schematic. straight json.

Moving a part

Move a part by clicking on it and then dragging it with your mouse.

Rotating a part

By selecting it with the mouse and then hitting "R," you may rotate a section. It will turn 90 degrees in the clockwise direction. Editing the diagram will allow you to rotate a portion by a different angle, such as 45 degrees. json.

Duplicating a part

By clicking on a part to select it and then hitting "D," you may make a new duplicate of that component. To make several duplicates of the component, repeatedly press "D."

Deleting a part

Delete a part by clicking on it (to select it) and then pressing the Delete button.

Selecting multiple parts

By clicking on the components while holding down the Shift key, you may choose several pieces. The components may then be combined, duplicated (using the "D" key), or eliminated (using the "Delete" key).

Copying and pasting parts

You can copy the selects part(s) by using the standard Copy keyboard shortcut (Ctrl+C or ⌘+C). All of the cables that link the parts you picked are replicated if you selected several components. The items you copied are kept in your system's clipboard in a diagram.json-like JSON format.

Click on the diagram and hit the normal Paste keyboard shortcut (Ctrl+V or ⌘+V) to paste the copied components. Sometimes the pieces will be pasted outside of the region of the diagram that is now visible, so you may need to zoom out to see them. The future will see this corrected.

The copy-paste tool allows you to swiftly copy many components (including their internal connections) at once across projects.

Editing wires

making a wire to connect two components

Click on one of the pins you want to link to create a new wire between the two components. Click on the second (target) pin after that. The wire will result from this.

After choosing the first pin, you may direct the wire if you want it to travel in a certain direction by clicking where you want it to go.

By clicking the right mouse button or pressing Escape, you may cancel a new wire (remove it without choosing a destination pin).

Changing the color of a wire

New wires are automatically assigned a color based on the pin's function: ground pins are black, 5 V pins are red, and all other wires are green. By clicking on a wire and then choosing a different color, you may modify the wire's color. You can also use the following keyboard shortcuts to set wire colors:

Shortcut	0	1	2	3	4	5	6	7	8
Color	Black	Brown	Red	Orange	Gold	Green	Blue	Violet	Gray
Shortcut	9	C	L	M	P	Y			
Color	White	Cyan	Limegreen	Magenta	Purple	Yellow			

These keyboard shortcuts also work while drawing a new wire. You can also change wire colors by editing diagram.json

Deleting a wire

New wires are automatically assigned a color based on the pin's function: ground pins are black, 5 V pins are red, and all other wires are green. By clicking on a wire and then choosing a different color, you may modify the wire's color.

Wokwi examples

Here are some quick examples of things you can make with Wokwi:

Arduino Uno "Hello World"

Below we see this project Arduino Uno "Hello World" is divided into three files

Code C++:

```
/* Hello Wokwi! */



#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 20, 4);
```

```
void setup() {
  lcd.init();
  lcd.backlight();
  lcd.setCursor(1, 0);
  lcd.print("Hello, Wokwi!");
}

void loop() {
  lcd.setCursor(7, 1);
  lcd.print(millis() / 1000);
}
```

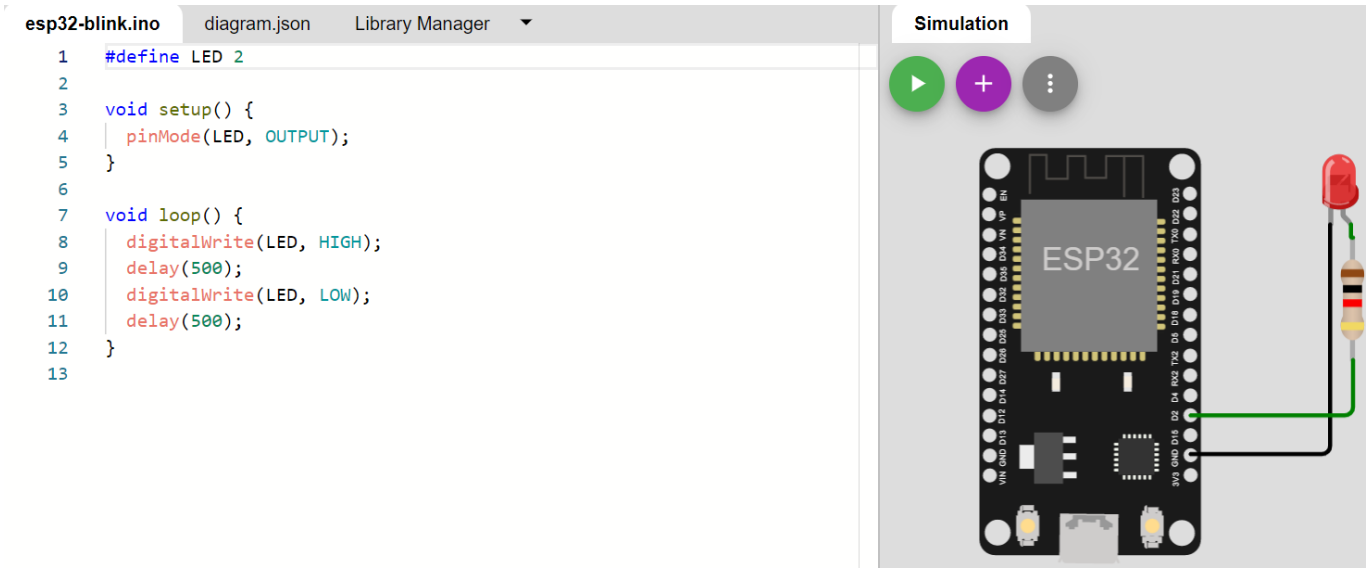
Diagram Code:

```
{
  "version": 1,
  "author": "Uri Shaked",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-arduino-uno", "id": "uno", "top": 20, "left": 50, "attrs": {} },
    { "type": "wokwi-lcd1602", "id": "lcd", "top": 252, "left": 50, "attrs": { "pins": "i2c" } }
  ],
  "connections": [
    [ "uno:GND.2", "lcd:GND", "black", [ "v24", "*", "h-20" ] ],
    [ "uno:5V", "lcd:VCC", "red", [ "v28", "*", "h-16" ] ],
    [ "uno:A4", "lcd:SDA", "green", [ "v32", "*", "h-12" ] ],
```

```
  [ "uno:A5", "lcd:SCL", "blue", [ "v36", "*", "h-8" ] ]

 ]

}
```

Blink an LED on ESP32

Below we see this project Blink an LED on ESP32



Code C++:

```
#define LED 2



void setup() {

 pinMode(LED, OUTPUT);

}



void loop() {

 digitalWrite(LED, HIGH);

 delay(500);

 digitalWrite(LED, LOW);
```

```
  delay(500);
}
```

Diagram Code:

```
{
  "version": 1,
  "author": "Uri Shaked",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": 0, "left": 0, "attrs": {} },
    {
      "type": "wokwi-led",
      "id": "led1",
      "top": -3.33,
      "left": 153.33,
      "attrs": { "color": "red" }
    },
    { "type": "wokwi-resistor", "id": "r1", "top": 64, "left": 149.33, "rotate": 90, "attrs": {} }
  ],
  "connections": [
    [ "esp:TX0", "$serialMonitor:RX", "", [] ],
    [ "esp:RX0", "$serialMonitor:TX", "", [] ],
    [ "esp:GND.1", "led1:C", "black", [ "h0" ] ],
    [ "led1:A", "r1:1", "green", [ "v0" ] ],
    [ "r1:2", "esp:D2", "green", [ "h0", "v38" ] ]
```

```
 ]

}
```

MicroPython on Wokwi

On Wokwi, MicroPython projects may be developed and executed. Start with the template for the Raspberry Pi Pico MicroPython project.

Project structure

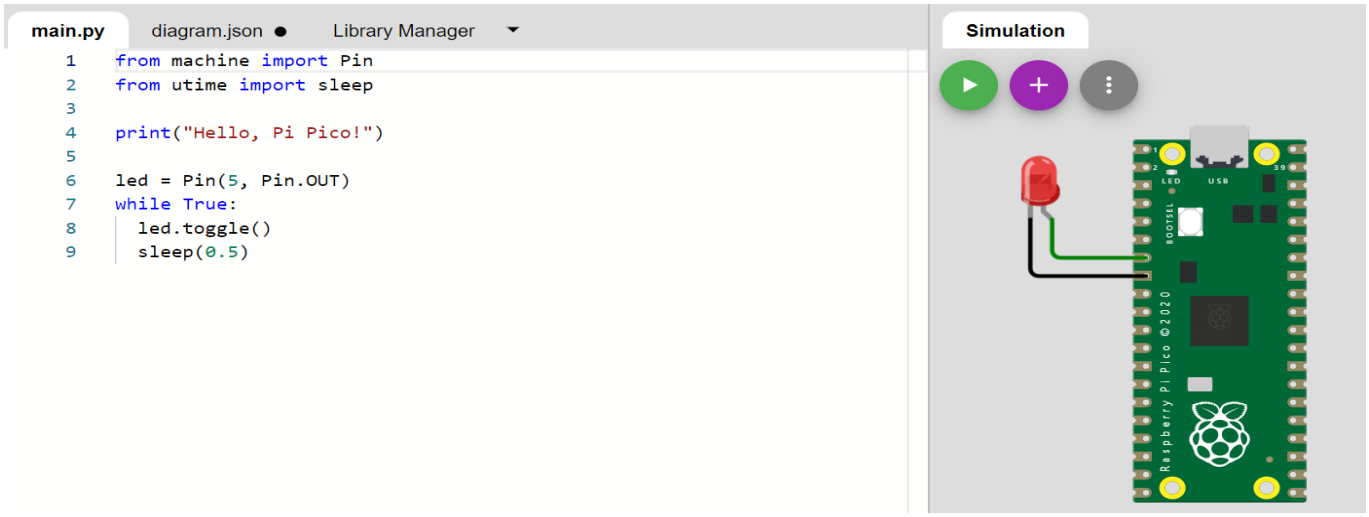
A main.py file is a must for every MicroPython project. When you begin the simulation, MicroPython will automatically load and run the code from main.py. All of the project files are copied by Wokwi into the Pico's flash storage. This implies that you may add more Python modules to your project and import them from either main.py or the interactive REPL. Text files in your project might also include unique data.

You can get a list of all the files in the flash filesystem by running:

```
import os

print(os.listdir('/'))
```

Blink with MicroPython

Below we see this project [Blink with MicroPython]



Code Python:

```
from machine import Pin

from utime import sleep



print("Hello, Pi Pico!")



led = Pin(5, Pin.OUT)

while True:

 led.toggle()

 sleep(0.5)
```

Diagram Code:

```
{

 "version": 1,

 "author": "Anonymous maker",

 "editor": "wokwi",

 "parts": [
```

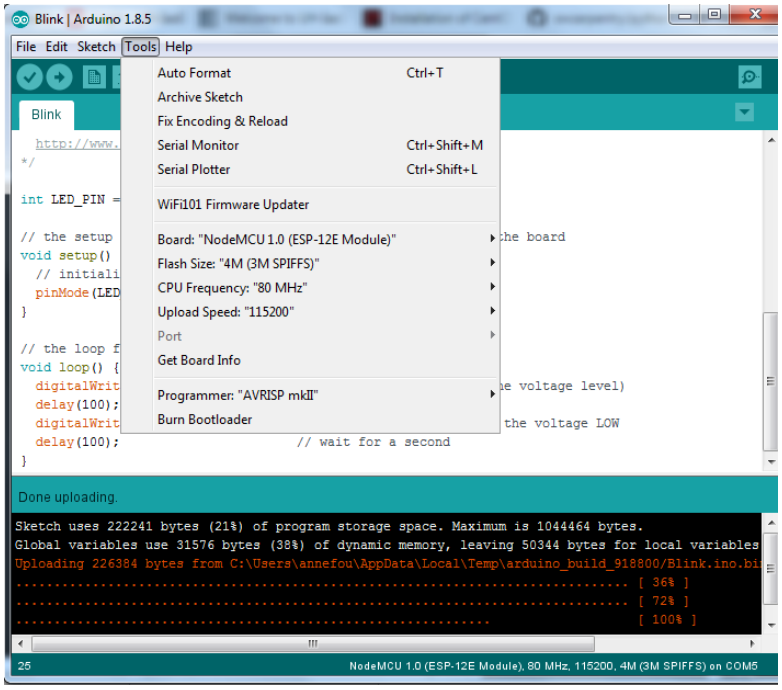
```
  { "type": "wokwi-pi-pico", "id": "pico", "top": -10.67, "left": -3.33, "attrs": {} },
  {
    "type": "wokwi-led",
    "id": "led1",
    "top": -2.67,
    "left": -65.33,
    "attrs": { "color": "red" }
  }
 ],
 "connections": [
  [ "pico:GP0", "$serialMonitor:RX", "", [] ],
  [ "pico:GP1", "$serialMonitor:TX", "", [] ],
  [ "pico:GP5", "led1:A", "green", [ "h0" ] ],
  [ "pico:GND.2", "led1:C", "black", [ "h0" ] ]
 ],
 "dependencies": {}
}
```

Arduino IDE

Why using Arduino GUI

Unlike other microcontroller, Arduino Nano is such a tiny microcontroller that using the Arduino IDE is currently the only viable way to program it.

The Arduino IDE has a simple interface with many built-in examples.



How to install Arduino Desktop IDE

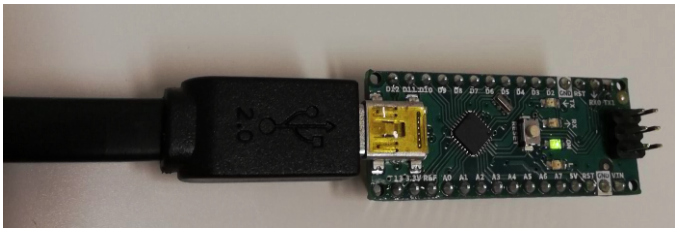
You may use the Arduino IDE online at https://create.arduino.cc/, but you must first register. The online version, however, needs an internet connection and may not function with different microcontrollers (such as ESP8266, ESP32, etc.). These factors lead us to believe that learning how to install the Arduino Desktop IDE on your laptop is crucial.

1. Download Arduino Desktop IDE at Arduino Software
2. Choose and click on the corresponding distribution depending on your Operating System (Windows 7, Windows10, Linux 64 bits, etc.)
3. Click on "**JUST DOWNLOAD**" and follow instructions given
4. If you do not have administrator rights, download the "*Windows ZIP file for non admin install*" and unzip it to the desired location.
5. Start Arduino Desktop IDE (if it does not start automatically or you don't find the Arduino IDE shortcut, click on arduino.exe)

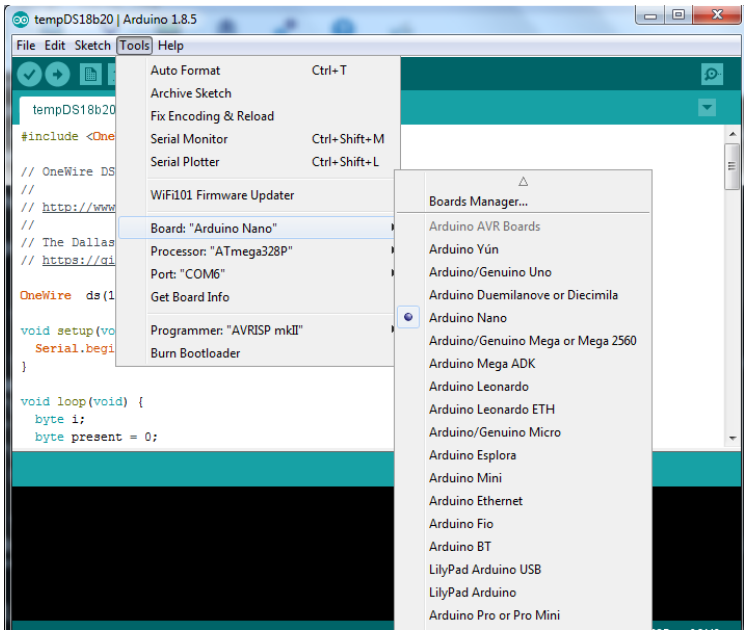
How to program an Arduino Nano with Arduino GUI

Verify that the Arduino Nano board is functionning

- As indicated in the image below, attach the micro USB type B side of the cable to the Arduino Nano board (pay attention to the connector's orientation!).



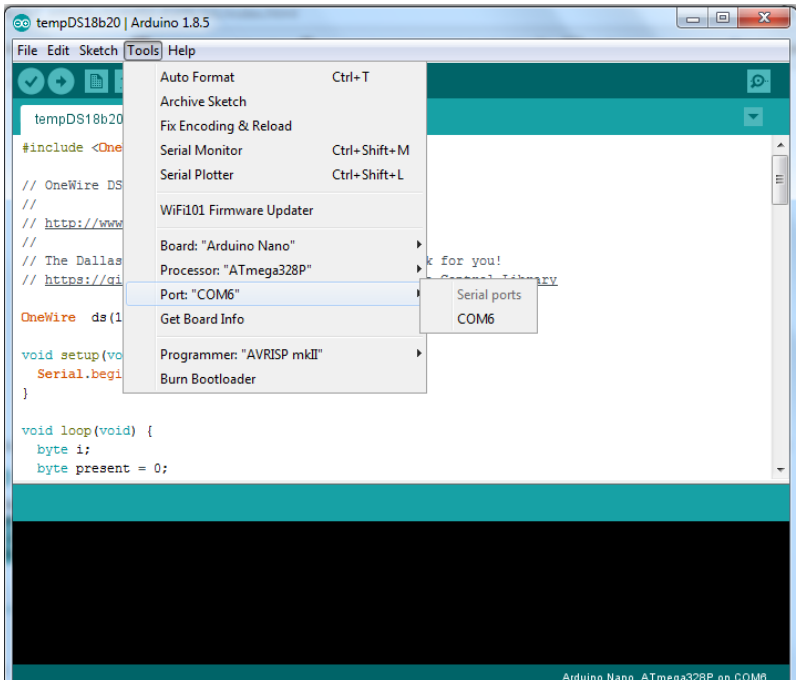
- Connect the USB type A side of the cable to your computer.
- Open tab “Tools –> Boards –> Boards Manager” and check “Arduino Nano” is available.
- Select **Arduino Nano** board



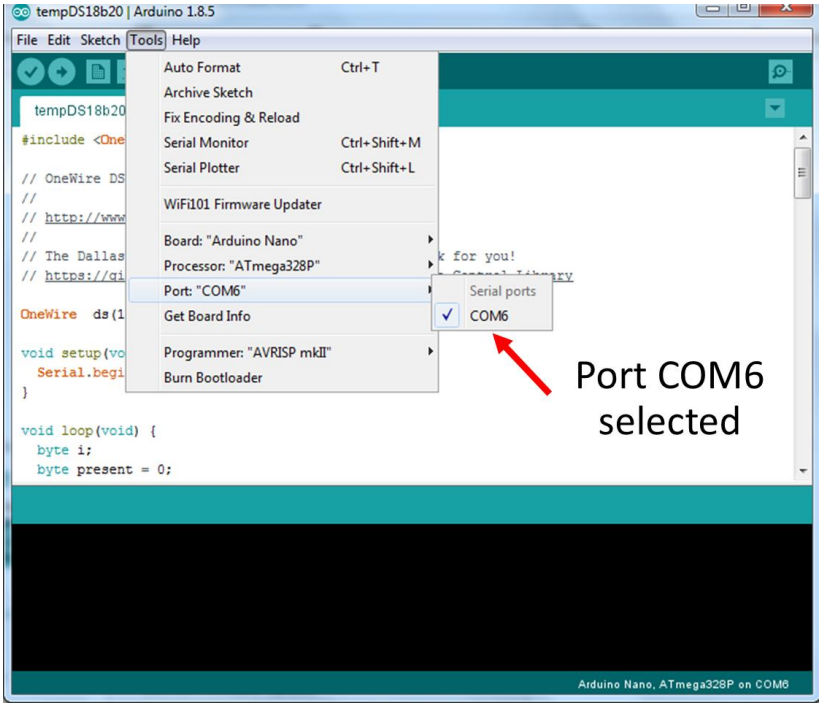
Select Port

In order to specify the route where the board is physically connected in your computer, you must lastly choose the port (a physical USB connection on your computer).

- Click on “**Tools –> Port**” and select the corresponding USB port.



- Check you have selected it: when you click again on “**Tools –> Port**”, you should see a “tick” in front of your selected port (see image below):

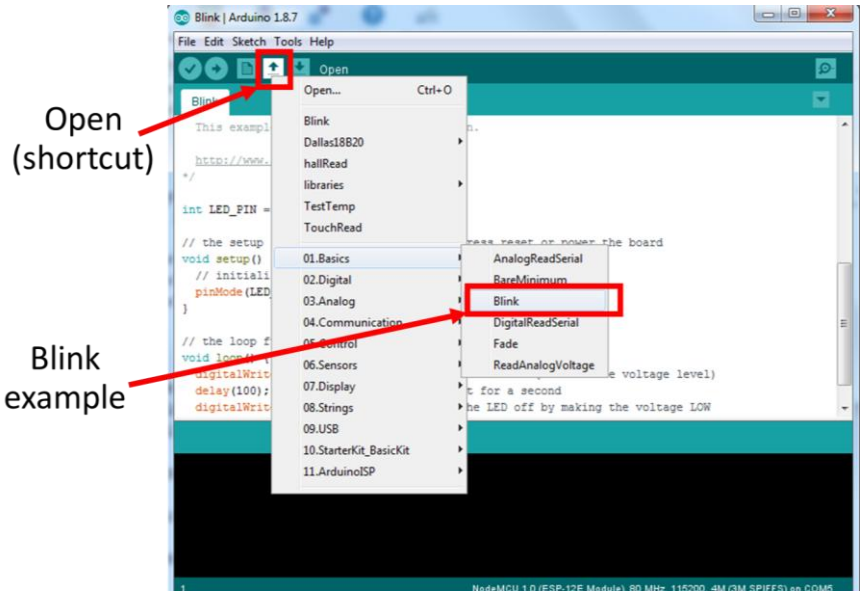


Name of the port

On a windows computer, ports are name such as “COM3”, “COM5”, “COM6”, etc. but the naming convention is different on Linux or Mac OSX

Run our very first program

As our very first program, we will blink the built-in led.



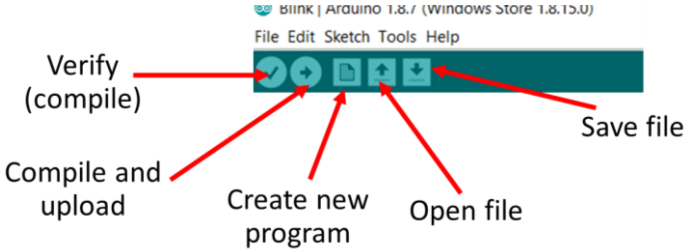
- You should have a code called “Blink” in your Arduino Deskop IDE.
- It uses predefined variables such as **LED_BUILTIN** to switch on and off the built-in led
- Change the delays (first to 2000 and then 1000):

```
/*
  Blink
```

```
  Turns an LED on for one second, then off for one second, repeatedly.
  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products
  http://www.arduino.cc/en/Tutorial/Blink
*/


// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_PIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(100);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(2000);                       // wait for a second
}
```

- Try to execute it:

Tips on Arduino Desktop IDE



Commands for Arduino Desktop IDE

The three primary components of the Arduino programming language (https://www.arduino.cc/reference/en/) are functions, values (variables and constants), and structure.

Functions

For controlling the Arduino board and performing computations.

Digital I/O

digitalRead()
digitalWrite()
pinMode()

Analog I/O

analogRead()
analogReference()
analogWrite()

Zero, Due & MKR Family

analogReadResolution()
analogWriteResolution()

Advanced I/O

noTone()
pulseIn()
pulseInLong()
shiftIn()
shiftOut()
tone()

Time

delay()
delayMicroseconds()
micros()
millis()

Math

abs()
constrain()
map()
max()
min()
pow()
sq()
sqrt()

Trigonometry

cos()
sin()
tan()

Characters

isAlpha()
isAlphaNumeric()
isAscii()
isControl()
isDigit()
isGraph()
isHexadecimalDigit()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()
isWhitespace()

Random Numbers

random()
randomSeed()

Bits and Bytes

bit()
bitClear()
bitRead()
bitSet()
bitWrite()
highByte()
lowByte()

External Interrupts

attachInterrupt()
detachInterrupt()

Interrupts

interrupts()
noInterrupts()

Communication

Serial
SPI
Stream
Wire

USB

Keyboard
Mouse

Variables

Arduino data types and constants.

Constants

HIGH | LOW
INPUT | OUTPUT | INPUT_PULLUP
LED_BUILTIN
true | false
Floating Point Constants
Integer Constants

Conversion

(unsigned int)
(unsigned long)
byte()
char()
float()
int()
long()
word()

Data Types

array
bool
boolean
byte
char
double
float
int
long
short
size_t
string
String()
unsigned char
unsigned int
unsigned long
void
word

Variable Scope & Qualifiers

const
scope
static
volatile

Utilities

PROGMEM
sizeof()

Structure

The elements of Arduino (C++) code.

Sketch

loop()
setup()

Control Structure

break
continue
do...while
else
for
goto
if
return
switch...case
while

Further Syntax

#define (define)
#include (include)
/* */ (block comment)
// (single line comment)
; (semicolon)
{} (curly braces)

Arithmetic Operators

% (remainder)
* (multiplication)
+ (addition)
- (subtraction)
/ (division)
= (assignment operator)

Comparison Operators

!= (not equal to)
< (less than)
<= (less than or equal to)
== (equal to)
> (greater than)
>= (greater than or equal to)

Boolean Operators

! (logical not)
&& (logical and)
|| (logical or)

Pointer Access Operators

& (reference operator)
* (dereference operator)

Bitwise Operators

& (bitwise and)
<< (bitshift left)
>> (bitshift right)
^ (bitwise xor)
| (bitwise or)
~ (bitwise not)

Compound Operators

%= (compound remainder)
&= (compound bitwise and)
*= (compound multiplication)
++ (increment)
+= (compound addition)
-- (decrement)
-= (compound subtraction)
/= (compound division)
^= (compound bitwise xor)
|= (compound bitwise or)

ΠΑΝΕΛΛΗΝΙΟΣ ΣΥΝΔΕΣΜΟΣ
ΕΠΙΧΕΙΡΗΣΕΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ
ΕΦΑΡΜΟΓΩΝ, ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΝΕΩΝ ΤΕΧΝΟΛΟΓΙΩΝ

A.S.E.L.
ROMANIA

Makecode Microbit

The BBC Micro:MakeCode bit's editor is the ideal tool for getting started with programming and creating. Anyone who has used Scratch before will be acquainted with the color-coded blocks, which are also strong enough to access all the functionality of this little computer. You may also see the text-based code hidden behind the blocks by switching to JavaScript.

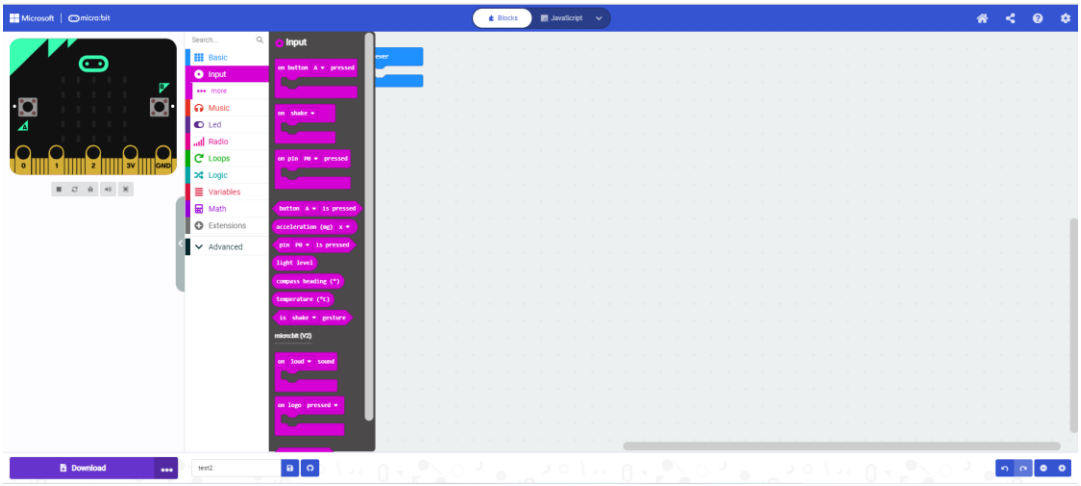
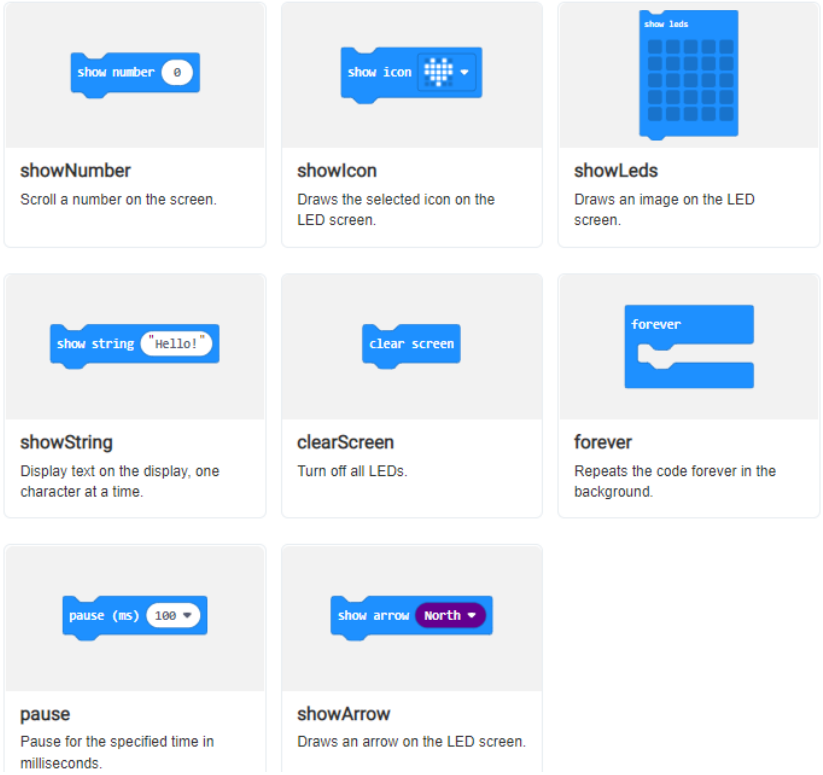


Figure 44 Makecode Microbi

Our getting started pages will guide you through your first steps (Makecode Microbit Docs, n.d.).

Basic

Use basic micro:bit functions and actions.

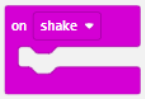


Input

Events and data from sensors

onButtonPressed

Do something when a button (A, B or both A+B) is pushed down and released again.



onGesture

Do something when when a gesture is done (like shaking the micro:bit).

onPinPressed

Do something when a pin is touched and released again (while also touching the GND pin).

onPinReleased

Do something when a pin is released.

buttonIsPressed

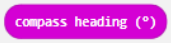
Get the button state (pressed or not) for ``A`` and ``B``.

pinIsPressed

Get the pin state (pressed or not).

isGesture

Tests if a gesture is currently detected.

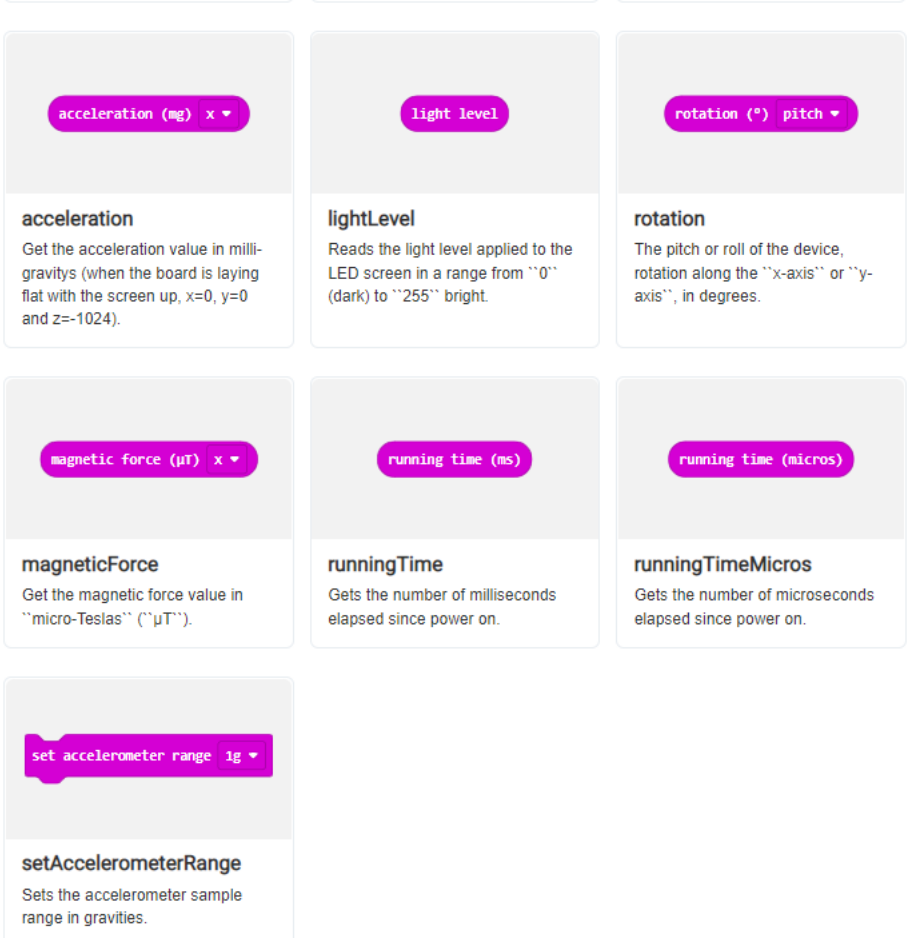


compassHeading

Get the current compass heading in degrees.

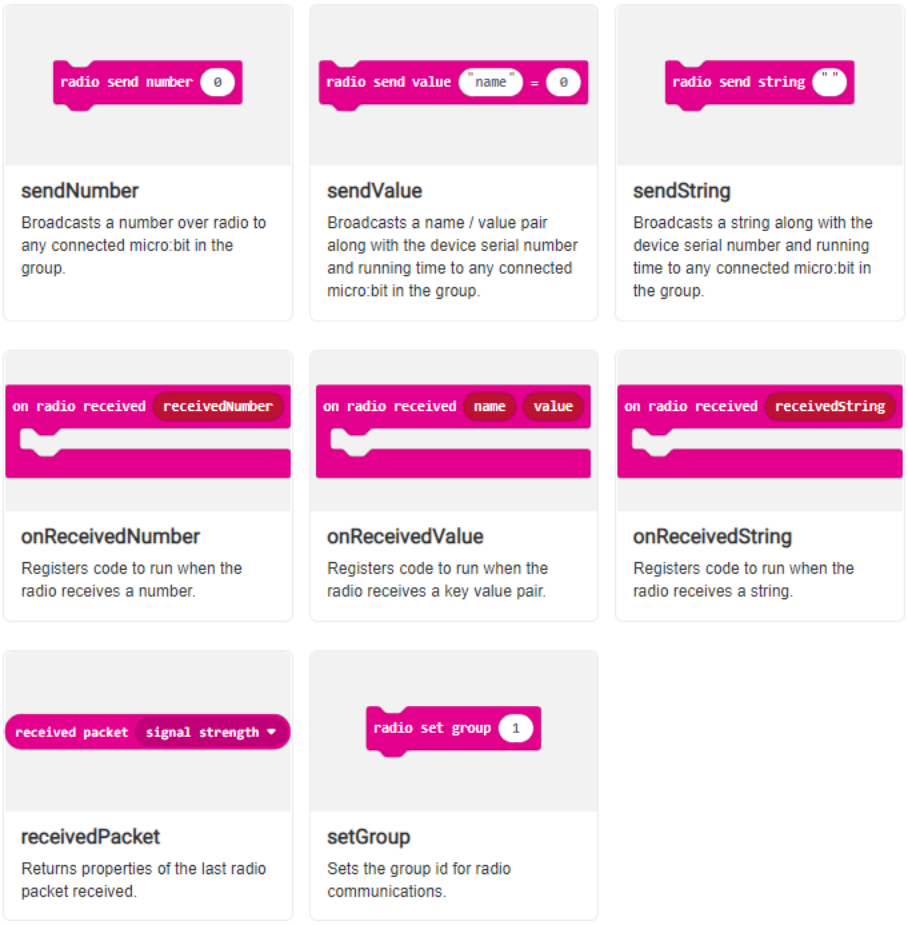
temperature

Gets the temperature in Celsius degrees (°C).



Radio

Send and receive data using radio packets.



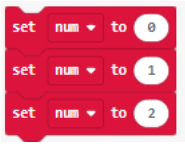
Math

Using numbers, number operators, and math functions.

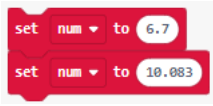
Numeric values: 0, 1, 2, 6.7, 10.083…

Just numbers by themselves. Sometimes these are called numeric literals.

Integers: whole numbers



Floating point: numbers with a fractional part



Even numbers may have fractional portions. Between the number's digits is the decimal point. However, floating point numbers, such as 3.14159 or 651.75, have the decimal point anywhere between the digits.

Arithmetic binary operation (+, -, *, /)

The operations for basic arithmetic: add, subtract, multiply, and divide.

Remainder (%)

This is a extra operator for division. You can find out how much is left over if one number doesn't divide into the other number evenly.

We know that 4 / 2 = 2, so 2 divides into 4 evenly. But, 5 / 2 = 2 with a remainder of 1. So, the remainder operation, 5 % 2 = 1, gives the number that's left over from a division operation.

Exponent (**)

The exponent operator will multiply the number on the left by itself for the amount of times of the number on its right. That is, 4 ** 2 = 4 * 4 and 2 ** 3 = 2 * 2 * 2. The area of a square that has sides with a length of 5 is equal to one side multiplied by another.

Square root

The original number is obtained by multiplying the square root of a given integer by itself. Since you already know that 2 * 2 = 4, the square root of 4 is 2. Because the area of a square is equal to the sum of the lengths of its two equal sides, it is known as a square root. The length of a side is the root.

Absolute value

When you want to know how much a number is without its *sign* (+/-). The absolute value of -5 is 5 and the absolute value of 5 is also 5. The absolute value is sometimes called the *magnitude*.

Minimum and maximum of two values

You can get the smaller or the bigger of two numbers with the min() and max() functions.

- The minimum of 2 and 9: **Math.min(2, 9)** equals 2.
- The maximum of 3 and 9: **Math.max(3, 9)** equals 9.

Round

If a number has a fractional portion, you may transform it to the next nearest integer value. We refer to this as rounding. The result of rounding the numbers 6.78 and 9.3 is 7 and 9, respectively. A number will round up to the next whole integer value with the higher value if it has a fractional portion that is more than or equal to 0.5. If not, it rounds to the subsequent lowest integer number.

When dealing with negative numbers, they round in the direction of the number's absolute value (for example, -8 is rounded to 8). In other words, 5.23 rounds to -5 and 2.68 rounds to -3.

Ceiling

Get the number's ceiling value to shift it to the next higher whole number (integer). Since 2 is the next higher whole number, it is the maximum value for the number 1.234. Since -3 is the next higher whole number, the ceiling for the negative number -3.63 is -3.

Floor

To make a number change to the next lower whole number (integer), get the number's *floor* value. The floor value for `8.76` is `8` since that is the next lower whole number. For the negative number of `6.17`, its floor is `-7` since that's the next lower whole number.

Truncate

The fractional part of a number is removed by *truncating* it. If a number has the value `54.234` its truncated value is `54`. Truncation works the same way for a negative number. The truncated value of `-34.913` is `-34`.

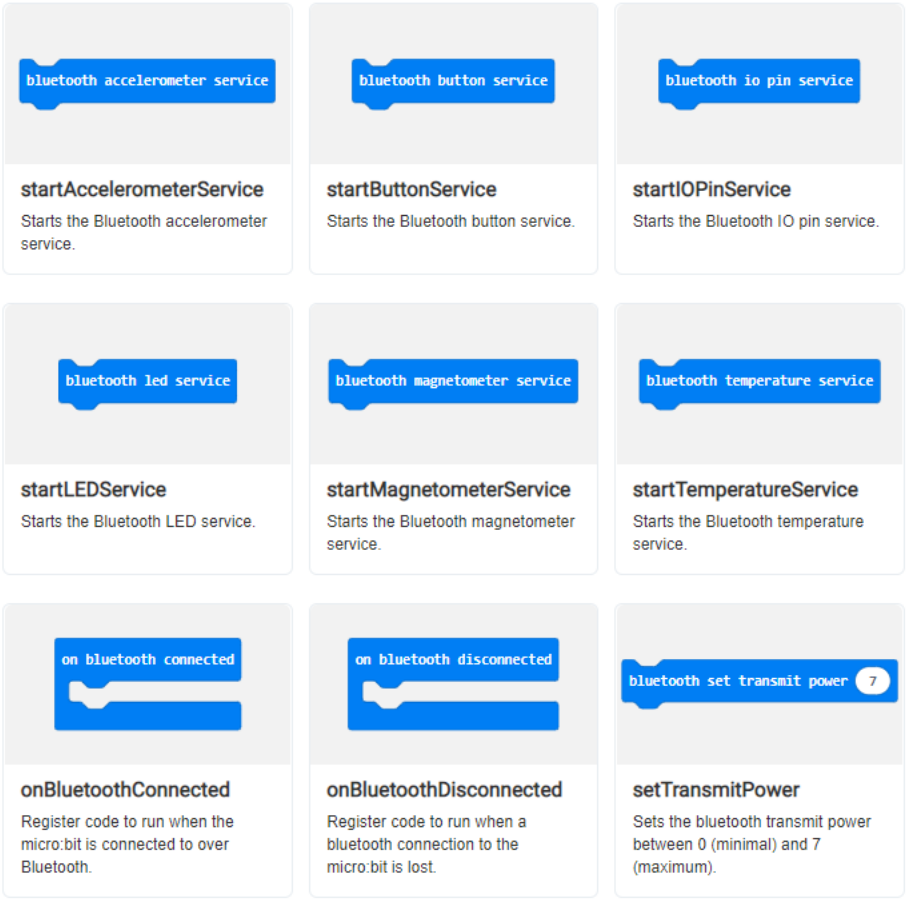
Random value

Make up any number from a minimum value to a some maximum value. If you want a random number up to 100, say: **Math.randomRange(0, 100)**.

Bluetooth

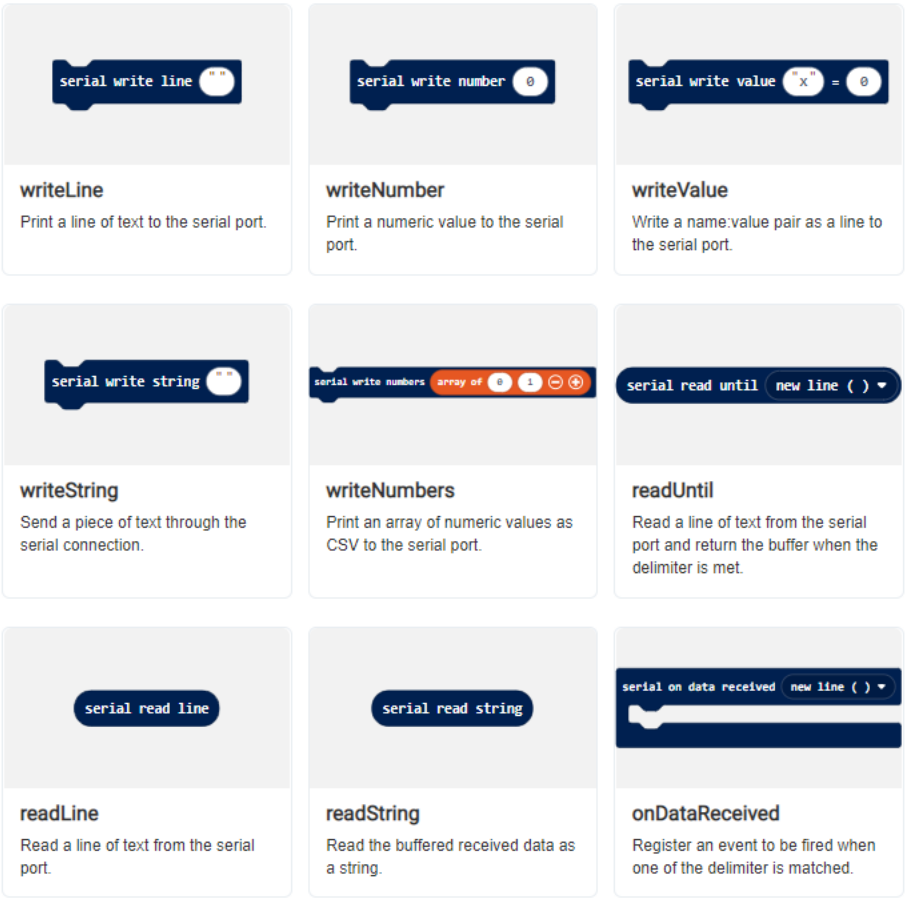
more Bluetooth services are supported.

The micro:bit must first be linked with another device, such as a smartphone, before the other device may access any of the Bluetooth "services" that the micro:bit has to offer. Once linked, the second device may communicate with the micro:bit and provide information about many of its capabilities.



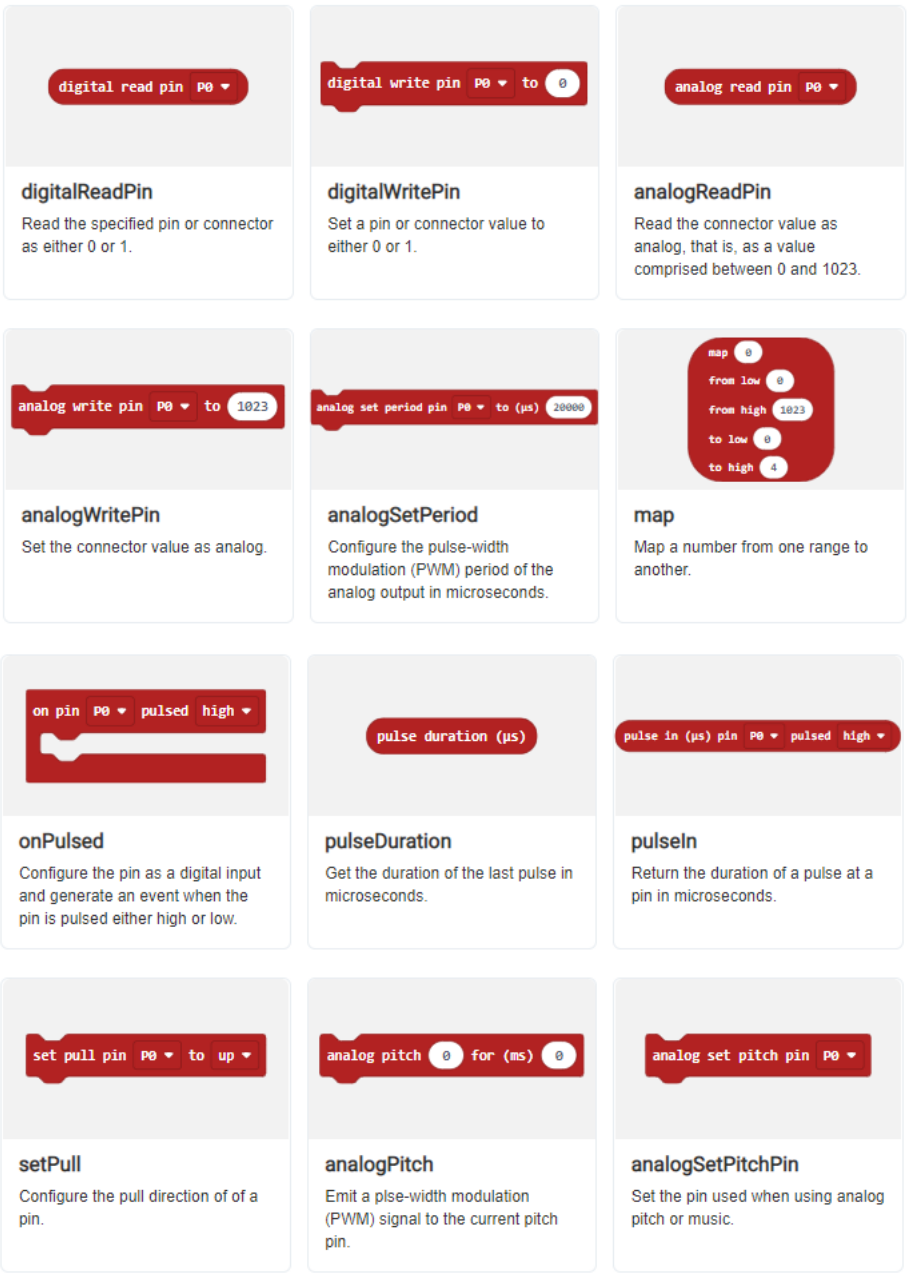
Serial

Read and write data over a serial connection.



Pins

Control currents in Pins for analog/digital signals, servos, i2c, …


digital read pin P0
digitalReadPin
Read the specified pin or connector as either 0 or 1.
digital write pin P0 to 0
digitalWritePin
Set a pin or connector value to either 0 or 1.
analog read pin P0
analogReadPin
Read the connector value as analog, that is, as a value comprised between 0 and 1023.
analog write pin P0 to 1023
analogWritePin
Set the connector value as analog.
analog set period pin P0 to (µs) 20000
analogSetPeriod
Configure the pulse-width modulation (PWM) period of the analog output in microseconds.
map 0
from low 0
from high 1023
to low 0
to high 4
map
Map a number from one range to another.
on pin P0 pulsed high
onPulsed
Configure the pin as a digital input and generate an event when the pin is pulsed either high or low.
pulse duration (µs)
pulseDuration
Get the duration of the last pulse in microseconds.
pulse in (µs) pin P0 pulsed high
pulseIn
Return the duration of a pulse at a pin in microseconds.
set pull pin P0 to up
setPull
Configure the pull direction of of a pin.
analog pitch 0 for (ms) 0
analogPitch
Emit a pIse-width modulation (PWM) signal to the current pitch pin.
analog set pitch pin P0
analogSetPitchPin
Set the pin used when using analog pitch or music.

Servos

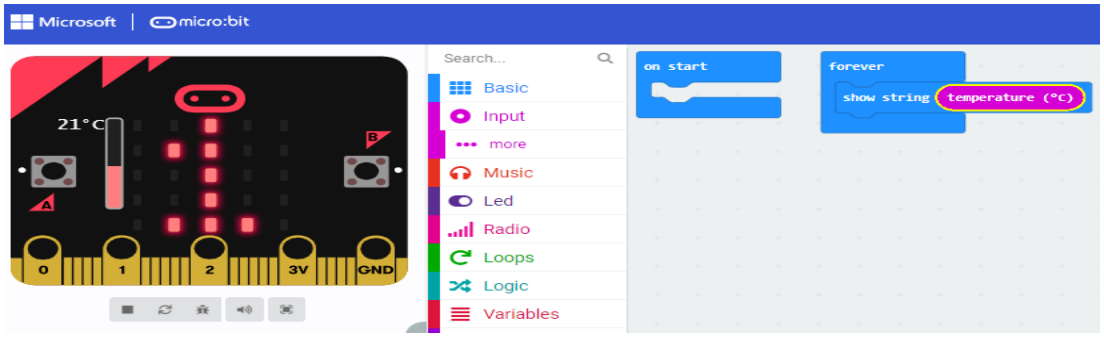
Makecode Microbit examples

Temperature

The internal temperature sensor in your micro:bit may be used to display how hot or cold it is.

By reading the temperature sensor in your Micro:processor bit's or CPU, this software can determine how warm or cold it is (central processing unit). The temperature of the processor is a decent indicator of the temperature outside of you in degrees Celsius (Celsius).

When input button A is pressed in this application, the micro:bit outputs the processor's current temperature on its LED display. Try bringing the micro:bit to cooler and warmer environments to see how the temperature readings vary.



Or

Python Code

```
def on_forever():
```

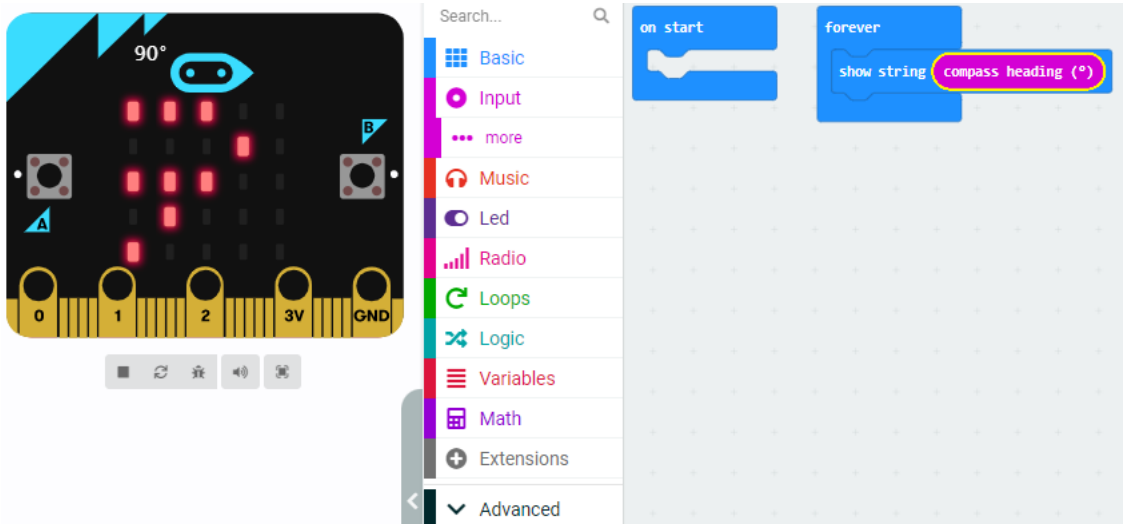
```
    basic.show_string("" + str((input.temperature())))

basic.forever(on_forever)
```

Or

Javascript Code

```
basic.forever(function () {

    basic.showString("" + (input.temperature()))

})
```

Compass



Or

Python Code

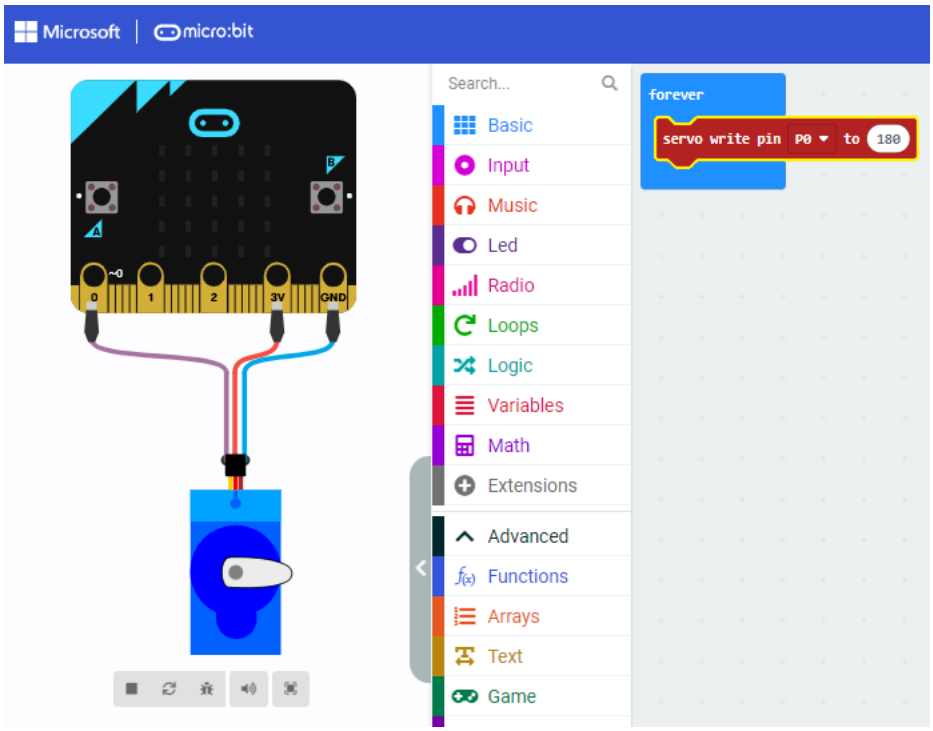
```
def on_forever():
    basic.show_string("" + str((input.compass_heading())))
basic.forever(on_forever)
```

Or

Javascript Code

```
basic.forever(function on_forever() {
    basic.showString("" + ("" + input.compassHeading()))
})
```

Servo



Or

Python Code

```
def on_forever():
    pins.servo_write_pin(AnalogPin.P0, 180)

basic.forever(on_forever)
```

Or

Javascript Code

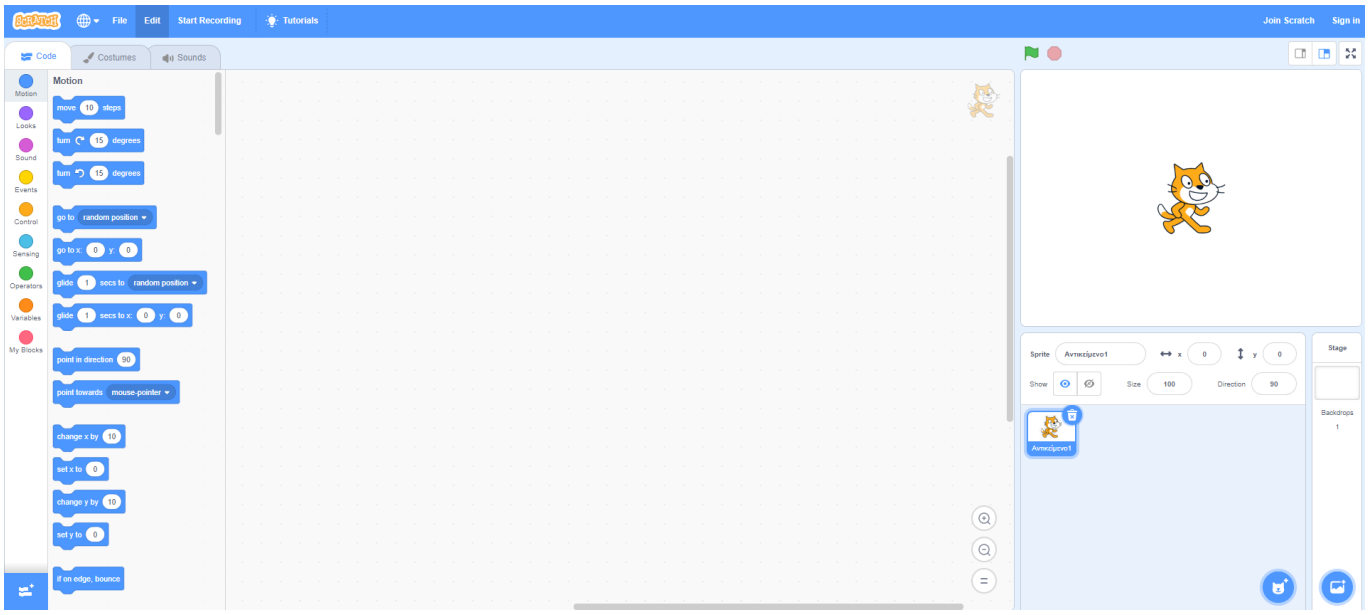
```
basic.forever(function on_forever() {
    pins.servoWritePin(AnalogPin.P0, 180)
```

```
})
```

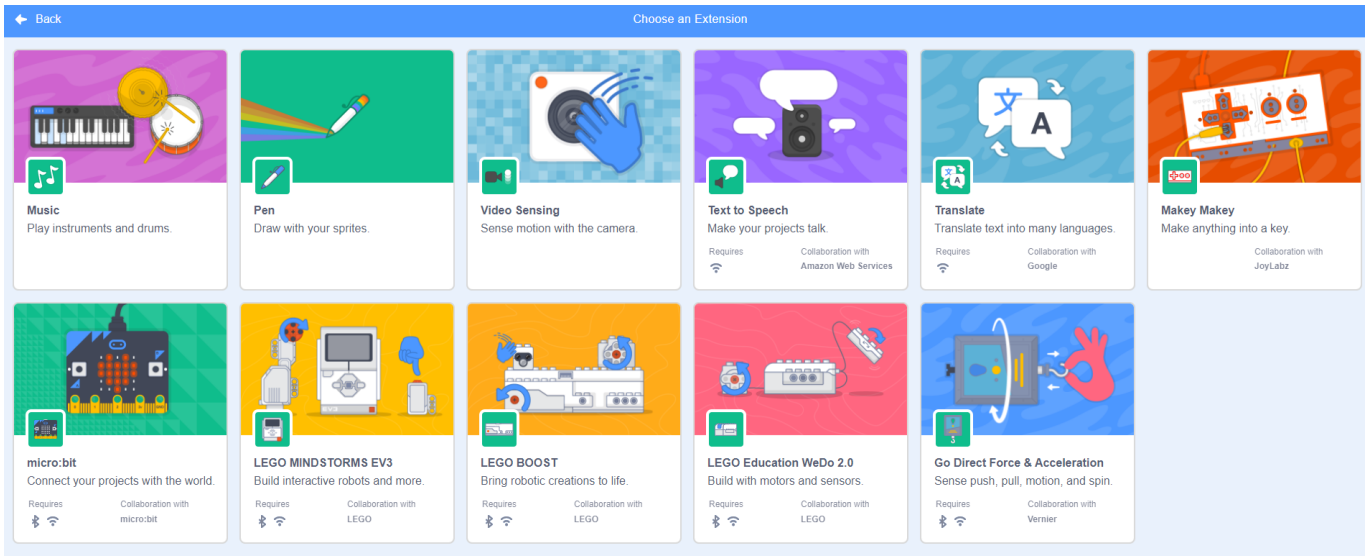
Scratch – Mindplus

Scratch (https://scratch.mit.edu/) is loved by children and adults the world over. For many, Scratch is their first introduction to programming from around age 8 and up.

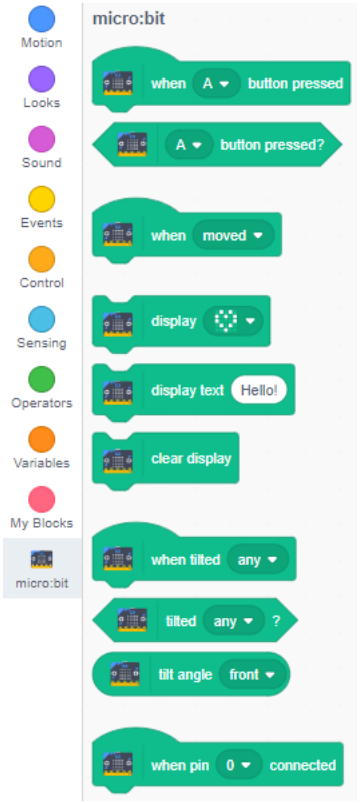
Micro:bit may be integrated into Scratch applications to create physical gaming controllers, paintbrushes, digital scoreboards, and other devices. To install the Android app, you'll need a computer running Windows or macOS (version 10/11) with Bluetooth and Scratch Link installed, or a Chromebook or mobile device. You may then explore our Scratch projects by clicking on the links below to get started.



open the add-ons and choose the micro bit

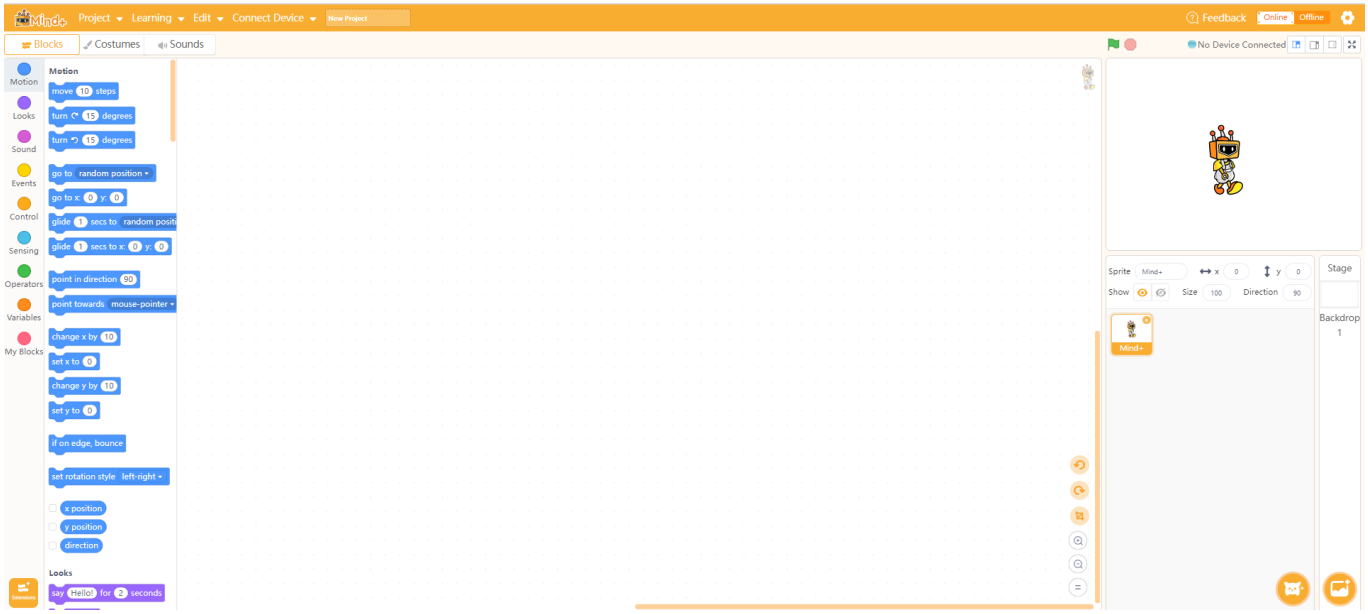


opening the micro bit we see that a menu opens with its tools

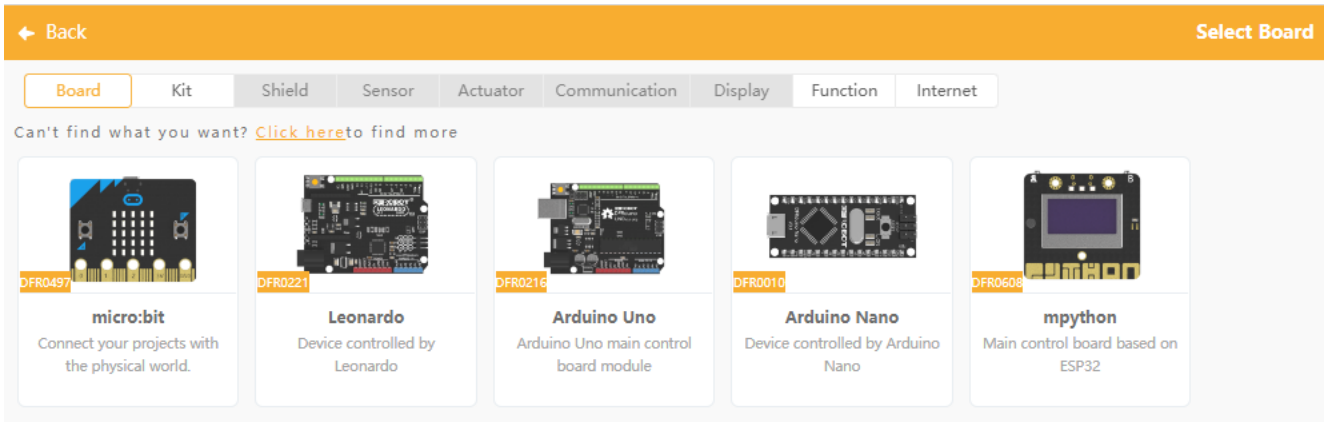


Mindplus

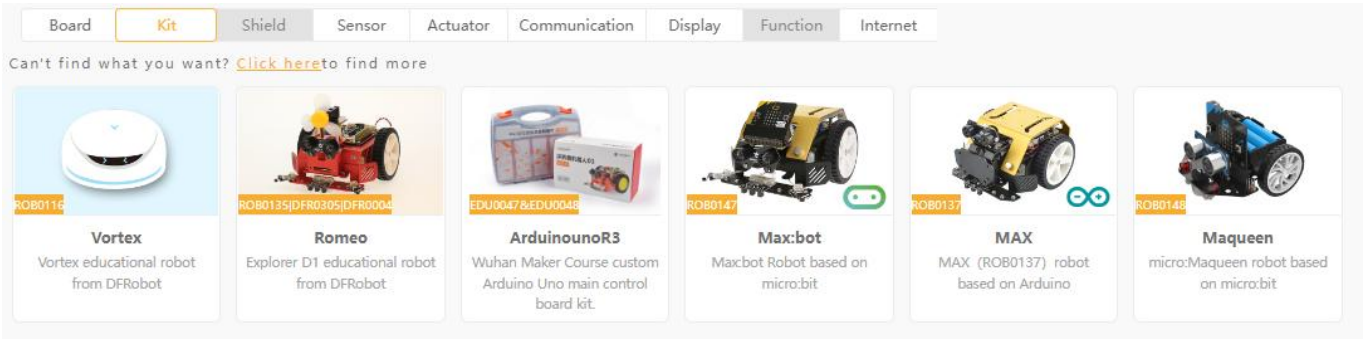
similar to Scratch is the schrach which, however, has many more extensions for both Arduino and micro bit

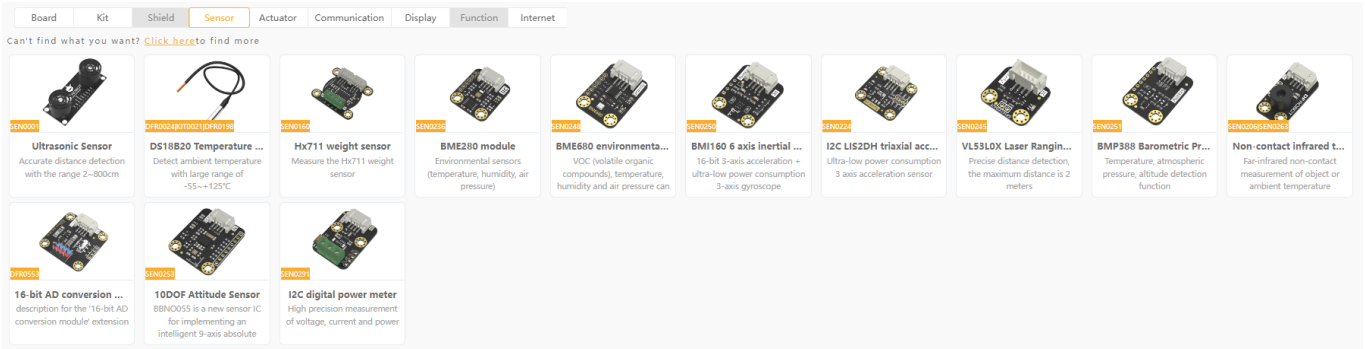


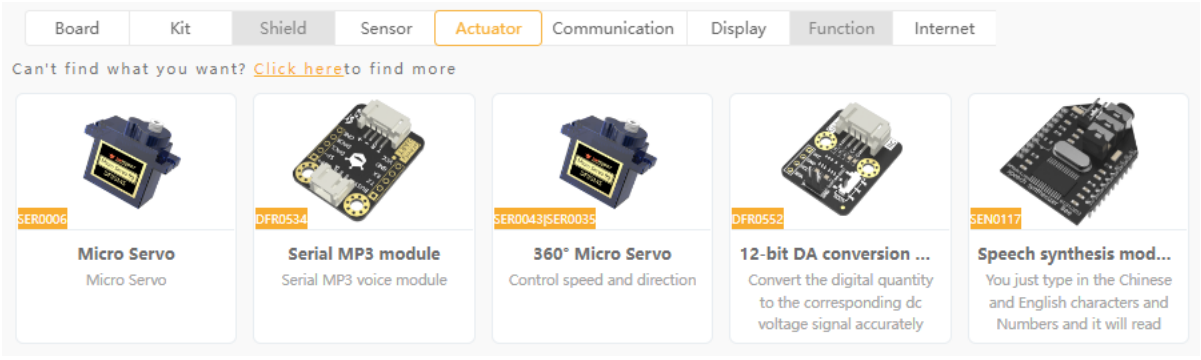
open the add-ons and choose

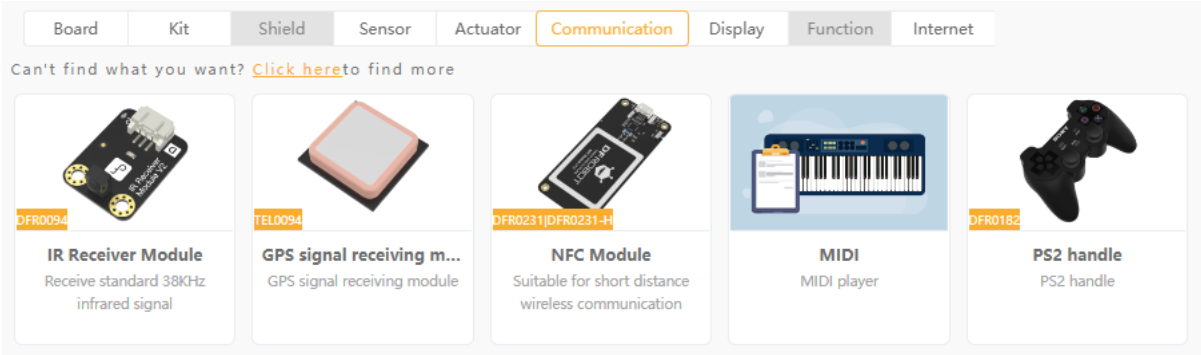


depending on which board we choose, the rest of the tools open kit, sensor ...







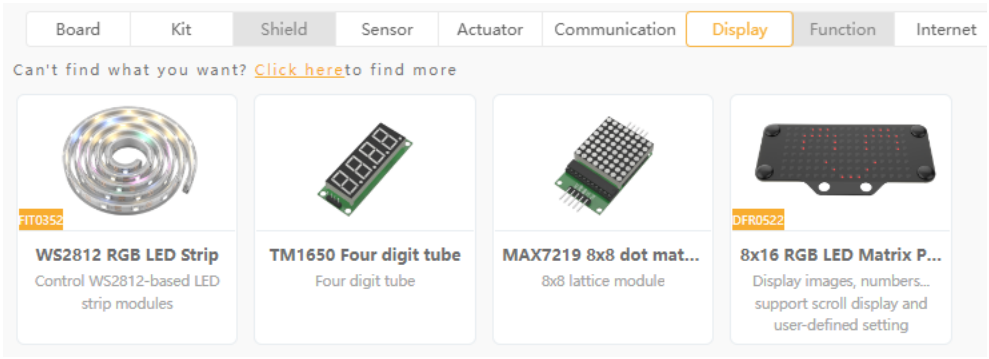


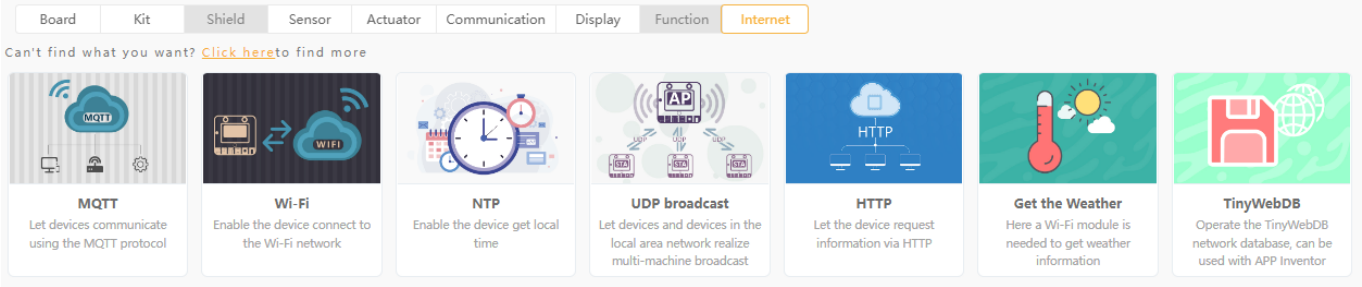
ΠΑΝΕΛΛΗΝΙΟΣ ΣΥΝΔΕΣΜΟΣ ΕΠΙΧΕΙΡΗΣΕΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΕΦΑΡΜΟΓΩΝ, ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΝΕΩΝ ΤΕΧΝΟΛΟΓΙΩΝ

A.S.E.L. ROMANIA

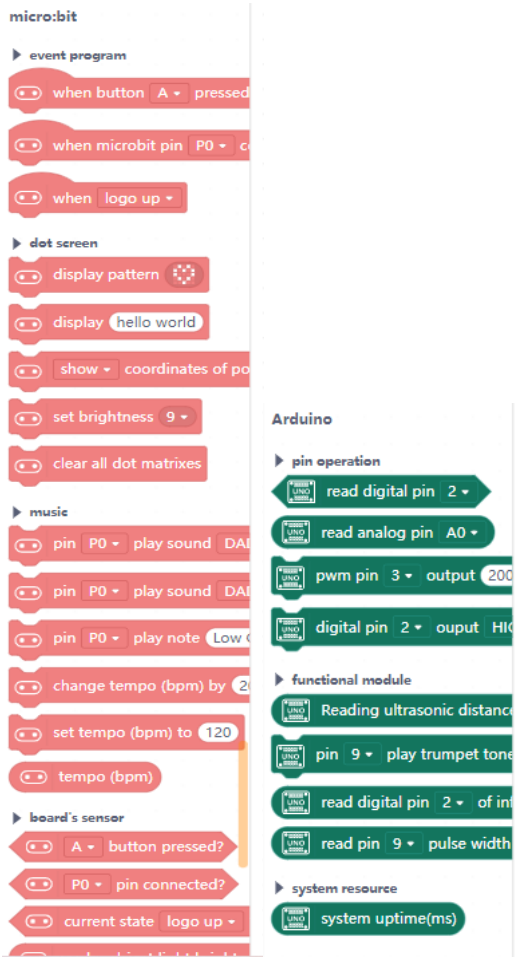
KOMPAKT

AIM

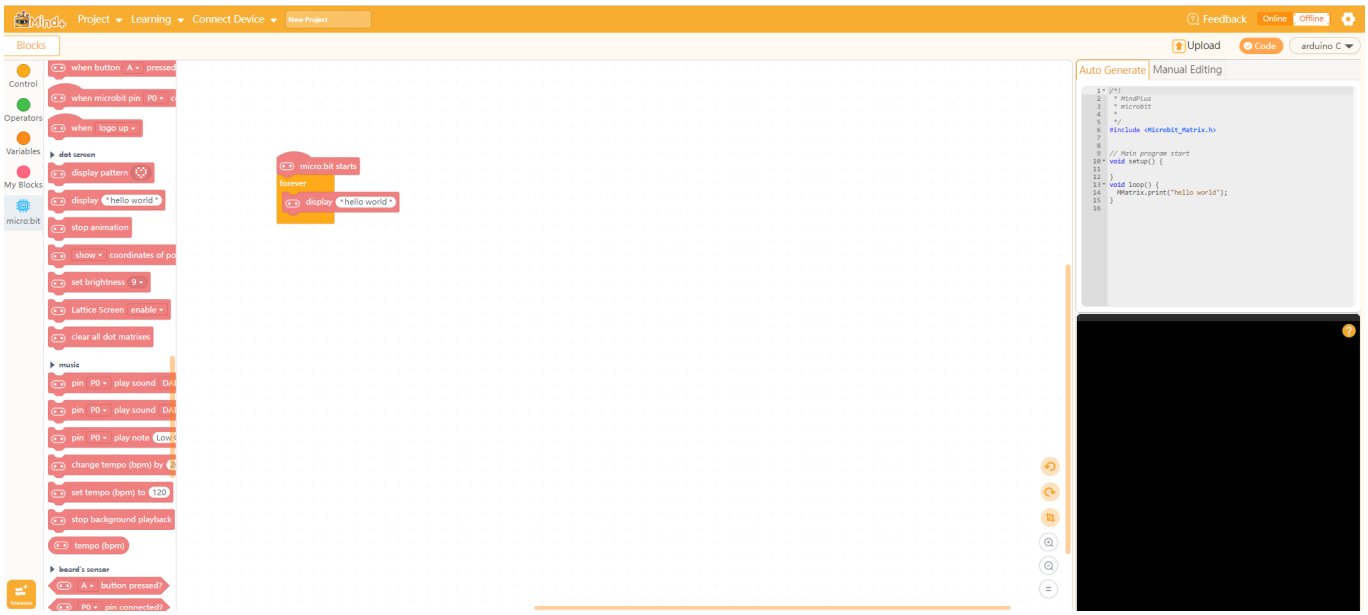


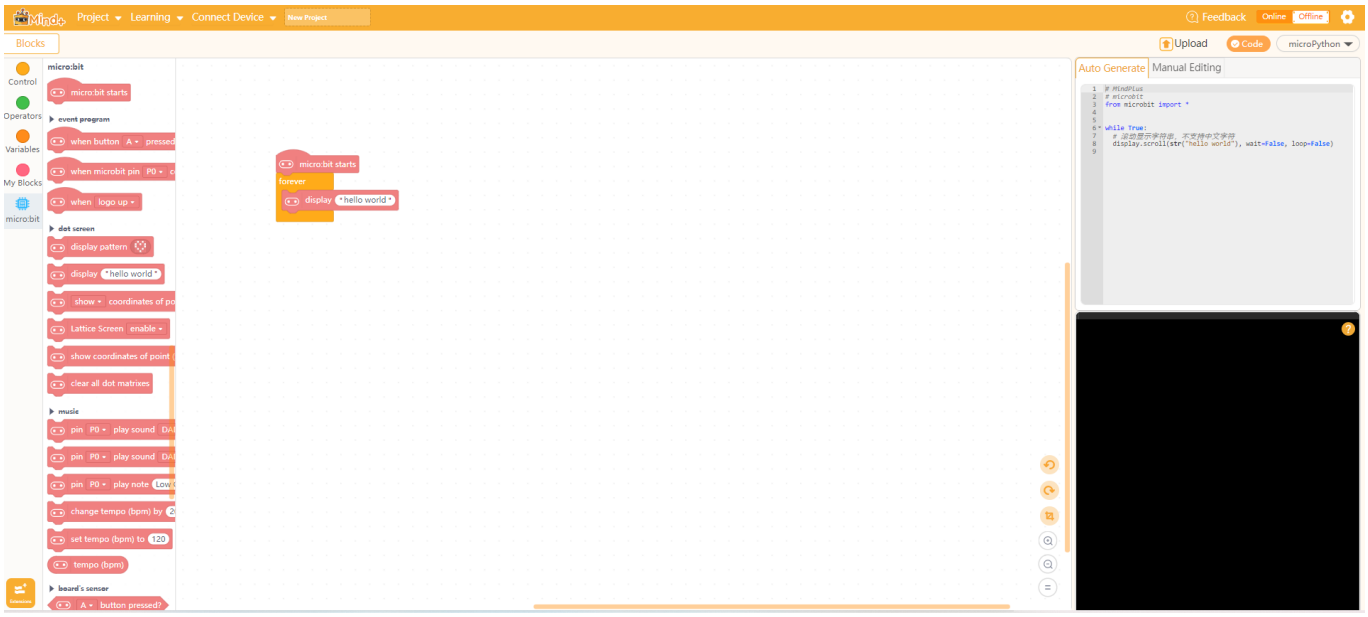


we see that a menu opens with its tools



Mind Plus has the ability to output and accept code in C++ and Python


Project
Learning
Connect Device
Feedback
Blocks
Upload
Code
arduino C
Auto Generate
Manual Editing
when button A pressed
when microbit pin P0
when logo up
display pattern
display "hello world"
stop animation
set brightness 9
Lattice Screen enable
clear all dot matrixes
pin P0 play sound
pin P0 play note
change tempo (bpm) by
set tempo (bpm) to 120
stop background playback
tempo (bpm)
button pressed?
pin connected?
micro:bit starts
forever
display "hello world"


Project
Learning
Connect Device
Feedback
Blocks
Upload
Code
microPython
Auto Generate
Manual Editing
micro:bit
micro:bit starts
when button A pressed
when microbit pin P0
when logo up
display pattern
display "hello world"
show coordinates of point
Lattice Screen enable
clear all dot matrixes
pin P0 play sound
pin P0 play note
change tempo (bpm) by
set tempo (bpm) to 120
tempo (bpm)
button pressed?
micro:bit starts
forever
display "hello world"

Python

Being an interpreted language, Python has some of the best features (OK, in reality, Python scripts are first compiled to some bytecode, and that bytecode is interpreted). This implies that in order to execute our program, we do not need to do a second compile step (which involves translating our program into machine code). In fact, we are even able to use the interpreter in interactive mode. We'll be able to test instructions one line at a time thanks to this!

We'll instruct Python to output "Hello, World!" to the terminal as a starter. This will be done first from the interpreter, and later a file will be created and executed like a program. This will demonstrate two of the most common methods to communicate with Python.

We can install python (https://www.python.org/) on our computer or run it online step by step (https://colab.research.google.com/).

Comments

Any text to the right of the pound (or hash) sign # is considered a comment. This text is not seen by the Python interpreter, so you may use it to make notes on the code's behavior for yourself or other programmers.

Example:

```
# This is a comment and is not seen by the interpreter

print("Hello, World!")
```

Variables

Variables are containers whose values can change. We can store a number or string in a variable and then retrieve that value later.

Example:

```
number = 42

print(number)
```

User Input

Use the input() method to request that a user enter data into the terminal. This will ask the user to input some text, including numbers, and then submit the text by pressing enter. The text will be read by the input() method after submission and returned as a string that may be saved in a variable.

Keep in mind that you may convert a string to an integer using the int() method (assuming the string is an integer).

Examples:

```
message = input("Type a message to yourself: ")

print("You said:", message)

number = int(input("Type a number:"))

print("You entered:", number)
```

Indentation

In Python, the amount of white space (spaces) at the start of a line is crucial. Statements that are grouped together must all be indented to the same depth (amount of white space in front of the line). This will be crucial when we discuss control flow statements and functions (like if and for).

Curly braces could be recognizable to you if you've previously written programs in other languages. Code between these curly brackets would constitute a group (or block) of code in other languages. The amount of indentation of the individual lines of code in Python is used to identify a group (or block) of code.

Operators

A symbol known as an operator instructs the interpreter to carry out a certain mathematical, relational, or logical operation on one or more pieces of data and provide the result.

Mathematical operators perform basic math operations on numbers:

Operator	Description	Example
`+`	Adds two numbers	`2 + 3` returns `5`
`-`	Subtracts one number from another	`8 - 5` returns `3`
`*`	Multiplies two numbers together	`4 * 6` returns `24`
`**`	Raises the first number to the power of the second number	`2 ** 4` returns `16`

/	Divides the first number by the second number	5 / 4 returns 1.25
//	Divides the two numbers and rounds down to the nearest integer (divide and floor)	5 / 4 returns 1
%	Divides the first number by the second number and gives the remainder (modulo)	19 % 8 returns 3

Logical operators compare two numbers and returns one of the *Boolean* values: `True` or `False`.

Operator	Description	Example
<	True if the first number is less than the second, False otherwise	5 < 3 returns False
>	True if the first number is greater than the second, False otherwise	5 > 3 returns True
<=	True if the first number is equal to or less than the second, False otherwise	2 <= 8 returns True
>=	True if the first number is equal to or greater than the second, False otherwise	2 >= returns False
==	True if the first number is equal to the second, False otherwise	6 == 6 returns True

`!=`	`True` if the first number is not equal to the second, `False` otherwise (not equal)	`6 !=` `6` returns `False`

Compound logical operators require Boolean inputs and give a Boolean answer.

Operator	Description	Example
`not`	Gives the opposite (`True` becomes `False` and vice versa)	`x = False; not x` returns `True`
`and`	Returns `True` if both operands are `True`, `False` otherwise	`x = True; y = False; x and y` returns `False`
`or`	Returns `True` if either of the operands are `True`, `False` otherwise	`x = True; y = False; c or y` returns `True`

Bitwise operators perform binary operations on the bits (1s and 0s) of the given numbers. This tutorial talks more about binary and bitwise operations.

Operator	Description	Example
`&`	Returns a 1 in each bit position for which the corresponding bits of both operands are 1 (bitwise AND)	`3 & 5` returns `1`
`|`	Returns a 1 in each bit position for wich the corresponding bits of either or both operands are 1 (bitwise OR)	`3 | 5` returns `7`

^	Returns a 1 in each bit position for which the corresponding bits of either but not both operands are 1 (bitwise XOR)	3 ^ 5 returns 6
~	Inverts the bits in the given number (bitwise NOT)	~5 returns -6
<<	Shifts the bits of the first number to the left by the number of bits specified by the second number	5 << 2 returns 20
>>	Shifts the bits of the first number to the right by the number of bits specified by the second number	5 >> 2 returns 1

Control Flow

The Python interpreter runs the statements in your code sequentially from top to bottom of the file. That is, unless we use some control flow statements to interrupt the usual sequential flow.

In the examples below, we provide the range(x, y) function, which produces a list of integers between the first number, x (inclusive), and the second number, y. (exclusive).

Statement	Description	Example

if elif else	If a condition is true, execute the block of code underneath the *if statement*. If not, see if the condition is true in one or more *else if* (elif) statements. If one of those is true, execute the code block under that. Otherwise, execute the code block underneath the *else statement*. elif and else statements are optional.	number = 42
		guess = int(input("Guess a number between 1-100: "))
		if guess == number:
		  print("You win!")
		elif guess < number:

		print("Nope") print("Too low") else: print("Nope") print("Too high") print("Run the program to try again")
while	A *while loop* executes the block of code underneath it repeatedly as long as the condition is true.	counter = 15 while counter >= 5: print(counter) counter = counter - 1
for..in	Iterate over a sequence of numbers or objects. The variable declared in a *for loop* assumes the value of one of the numbers (or objects) during each iteration of the loop.	for i in range(1, 11): print(i)

break	Use the *break statement* to exit out of a loop.	while True:
		message = input("Tell me when to stop: ")
		if message == "stop":
		break
		print("OK")

continue	The *continue statement* works similar to *break*, but instead of exiting the loop, it stops the current iteration and returns to the	for i in range(1, 6): if i == 3:

	top of the loop.	continue print(i)

Functions

With functions, you may give a piece of code a name and then reuse it by invoking that name. You may use variables called parameters to send data to a function (note that the variables in the function definition are called parameters whereas the actual data itself being passed are known as arguments). The return statement may be used to return data to the caller statement.

An example of a function definition would look like:

```
def functionName(parameter1, parameter2):
    # Code goes here
```

You can call this function else where in your code with functionName(argument1, argument2).

Keep in mind that variables specified within the declaration of the function are referred to as having local scope. As a result, they are inaccessible when used outside of that function. Global scope variables are those defined at the top level of the program, outside of any functions, loops, or classes, and are accessible from anywhere in the code (including inside functions).

Important: Prior to using any functions you design, they must first be specified!

There are several built-in functions in Python that might be useful for you; we've previously seen three of them: print(), int(), and range(). A list of these functions can be found in the Python Tutorial.

Example:

```
def add(x, y):
    sum = x + y
    return sum

print(add(2, 3))
```

Data Structures

Python has four more methods for storing data in addition to variables and objects: list, tuple, dictionary, and set. Each of these structures interacts with the collection of related data in a slightly different manner.

Structure	Description	Example

List	A *list* is a sequence of ordered items. You can access items in a list using brackets `[]` and an index (e.g. `list[2]`). Note that indeces are 0-based, which means `list[0]` will access the first item in the list. Because lists can be modified, they are known as *mutable*.	`my_list = [1, 5, 73, -3]`
		`my_list[2] = -42`
		
		`# Get third item in list`
		`print(my_list[2])`
		
		`# Get all but first item in list`
		`print(my_list[1:])`

Tuple	A *tuple* works just like a list (an ordered set). The difference is that a tuple is *immutable*, which means you cannot change the values once they are set. A tuple is usually specified by parentheses `()`. While the parentheses are not necessary, they are highly recommended to make your code easier to read.	`my_tuple = ("bird", "plane", 5, "train")`
		
		`# Get one item from the tuple`
		`print(my_tuple[0])`
		
		`# Get a tuple of second and third items`
		`print(my_tuple[1:3])`
		
		`# Can't do this because tuples are immutable!`
		`my_tuple[1] = "skyscraper"`

Dictionary	A *dictionary* is a collection of assciated *key* and *value* pairs. Similar to how a phonebook works: you look up someone's name (key) and you find their phone number (value). Dictionaries are defined by curly braces `{}`, and key/value pairs are separated by a colon `:`. Dictionaries are mutable.	`my_dictionary = {"name": "Bruce", "aka": "The Hulk"}`
		`my_dictionary["name"] = "Banner"`
		`print(my_dictionary["name"])`

Set	A *set* is a mutable, unordered collection with no duplicate elements. Sets are optimized for determining if an item is in the set (and you don't care about the order of items). Sets are great if you want to implement any sort of *set theory* in Python.	`my_set_1 = set(["orange", "banana"])`
		`my_set_2 = set(["apple"])`
		`my_set_2.add("orange")`
		
		`print(my_set_1)`
		`print(my_set_2)`
		`print("apple" in my_set_2)`
		`print("strawberry" in my_set_2)`
		`print(my_set_1.union(my_set_2))`
		`print(my_set_1.intersection(my_set_2))`

Python Micro:bit

The first text-based language you should learn is Python. Code is simple to read, comprehend, and edit because of its natural language-based instructions and structure.

It is extensively utilized in business as well as academia, particularly in the fields of data science and machine learning. Python is utilized by professionals working in a variety of industries, including medical, physics, and finance, in addition to software developers.

Kids between the ages of 11 and 14 may learn the principles of programming via text-based coding using Python on the BBC Micro:bit. These immersive, creative learning experiences for students help them become more engaged and knowledgeable.

Python (https://python.microbit.org/v/3) is a great way to deepen your programming skills through text-based coding. Its natural English-like structure makes it easy to start learning, but it's also powerful enough to be used in areas like data science and machine learning.

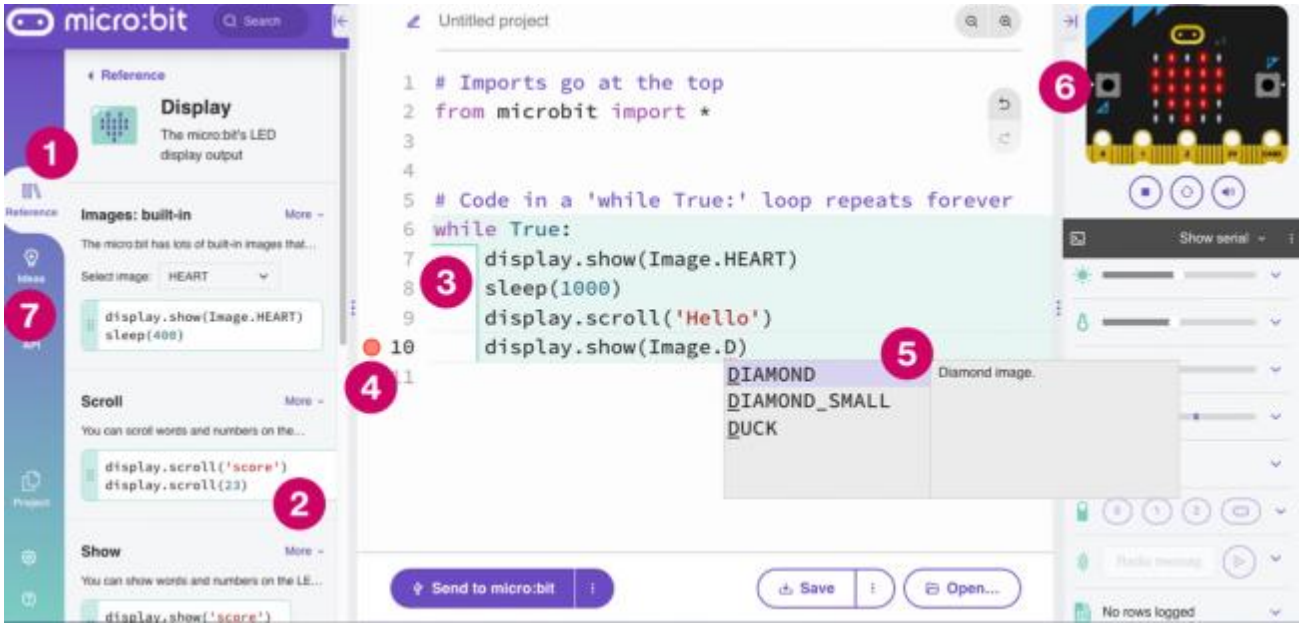


Figure 45 Micro:bit Python Editor

Features for education

Click on the headings below to discover some key features of the micro:bit Python Editor designed to overcome some common barriers, boost creativity and make the most of your coding time in class:

1. Reference

Similar to exploring blocks in MakeCode or Scratch, it is simple to learn what Python and the Micro:bit are capable of by using the Reference section.

Your pupils' creativity will soar as they quickly realize the possibilities of the Micro:bit hardware and write Python-based programs.

2. Drag and drop code snippets

In order to try out functional samples of code right immediately, students may just drag them into the editor. By using this, you may save time and get over obstacles like the requirement to recall correct syntax and a lack of keyboarding abilities.

3. Code structure highlighting

Your students' Python programs' structures are shown in blocks of color, which aids in the project's design, planning, and testing stages. When you can quickly identify which lines of code belong in a loop or "if... then" statement, for instance, it is simpler to comprehend the logic of a program.

Clear lines aid in debugging and make it simple to see if code is not properly aligned since we are aware that students unfamiliar with Python sometimes find indentations difficult.

4. Error highlighting

Errors are a normal part of coding. Symbols by the line number help you and your students identify bugs and fix them before sending code to a micro:bit. You can hover over the circle in the margin to see an explanation of the error.

5. Auto-complete

Two of the main obstacles to beginning text-based coding are the fear of a blank screen and a lack of inspiration.

The editor will thus provide choices as you write, which you may choose by clicking or hitting the enter key. By doing this, students may avoid typing typos, save time, and avoid having to memorize complex grammar.

It's also an additional method to learn what Python and the Micro:bit are capable of, for instance by viewing more built-in picture possibilities.

6. Simulator

Students can test their code out using the simulator before sending it to a real micro:bit.

This helps them develop, test, debug and evaluate their code and means they can work on projects even when they don't have access to a micro:bit device.

7. Ideas

Don't forget to explore the Ideas tab which contains complete working programs your students can use straight away, then modify to make their own. You'll find an emotion badge, step counter, radio, and sound projects.

6. API

For usage and examples.

- gc
  Control the garbage collector
- log
  Log data to your micro:bit V2
- machine
  Low-level utilities
- math
  Mathematical functions
- microbit
  Pins, images, sounds, temperature and volume
- microbit.accelerometer
  Measure the acceleration of the micro:bit and recognise gestures
- microbit.audio
  Play sounds using the micro:bit (import audio for V1 compatibility)
- microbit.compass

Use the built-in compass

- microbit.display

  Show text, images and animations on the 5×5 LED display
- microbit.i2c

  Communicate with devices using the I^2C bus protocol
- microbit.microphone

  Respond to sound using the built-in microphone (V2 only)
- microbit.speaker

  Control the built-in speaker (V2 only)
- microbit.spi

  Communicate with devices using the serial peripheral interface (SPI) bus
- microbit.uart

  Communicate with a device using a serial interface
- micropython

  MicroPython internals
- music

  Create and play melodies
- neopixel

  Individually addressable RGB and RGBW LED strips
- os

  Access the file system
- power

  Manage the power modes of the micro:bit (V2 only)
- radio

  Communicate between micro:bits with the built-in radio
- random

  Generate random numbers
- speech

Make the micro:bit talk, sing and make other speech like sounds

- struct

  Pack and unpack primitive data types

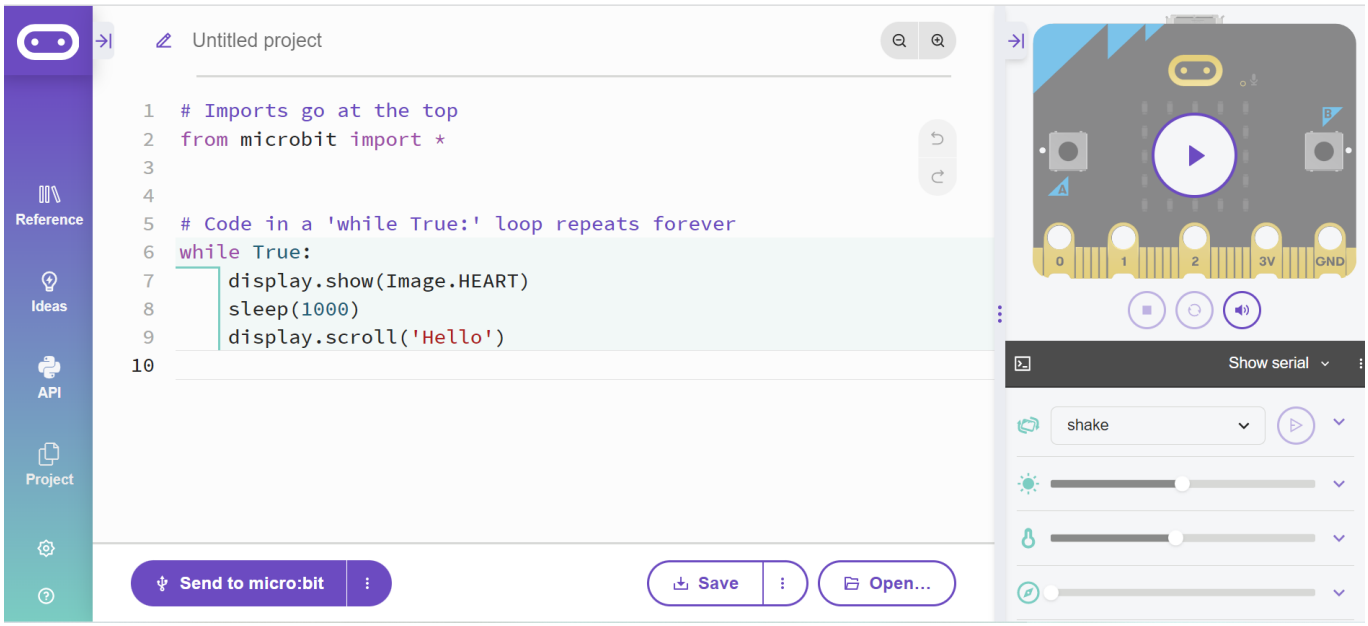
- sys

  System specific functions

- time

  Measure time and add delays to programs

7. Example



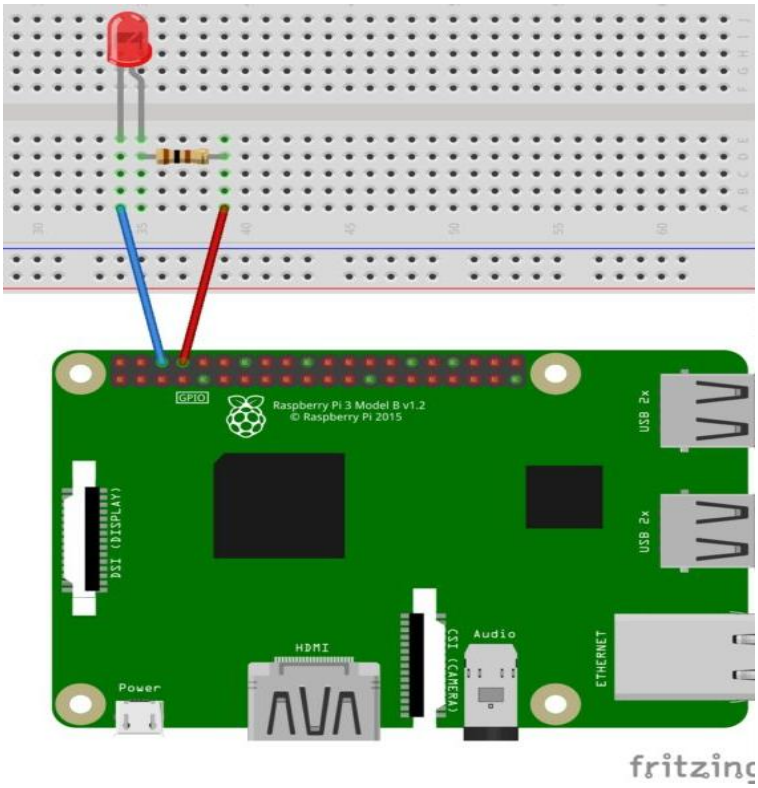
Other https://app.edublocks.org/editor

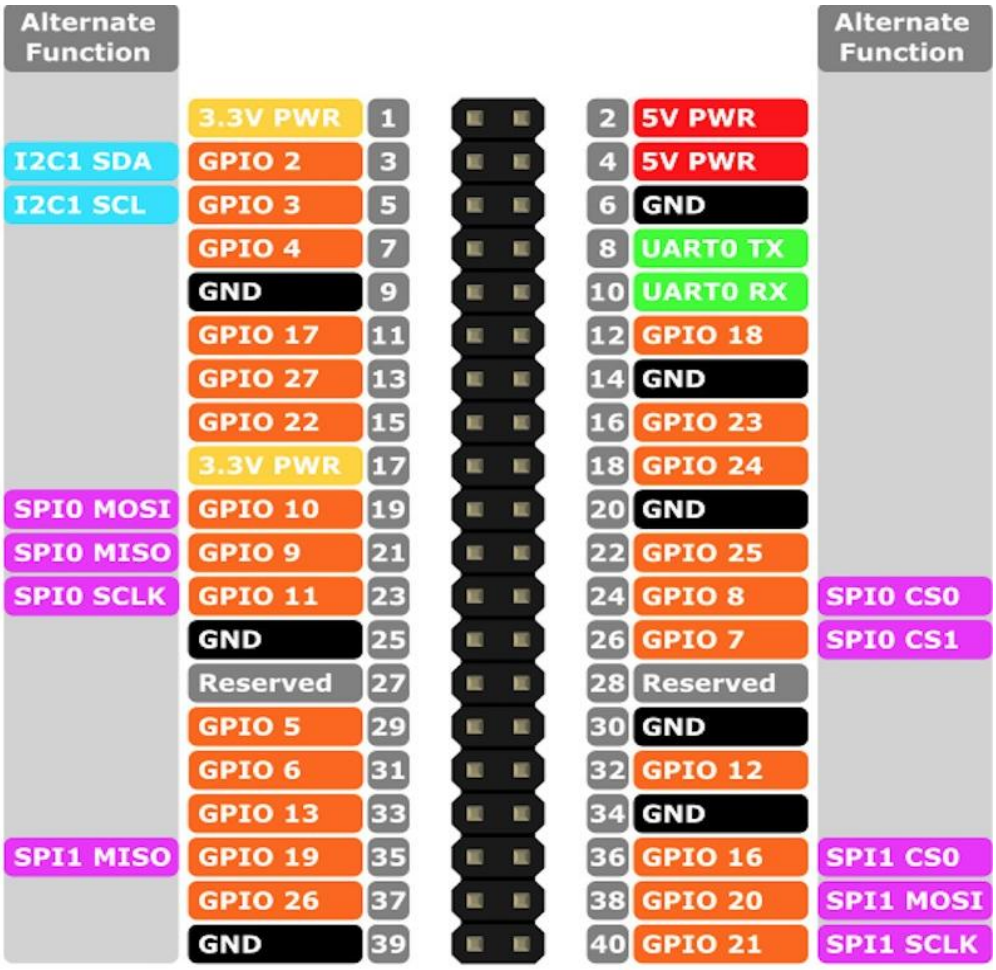
Python Raspberry Pi

Connecting directly to the Raspberry Pi, we can use ohm's law to determine the amount of resistor required to restrict the current to the LED's maximum forward current (IF):

$$R_\Omega = \frac{V}{I} = \frac{3.3 - VF}{IF} = \frac{3.3 - 1.7}{20mA} = 80\Omega$$

Unfortunately, the size of an 80 ohm resistor is not common. We may either combine many resistors to address this problem or round up to a normal size. We would round up to 100 ohm in this situation.





Launch the terminal and enter the following commands to install the Python library.

```
$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

8. *Blinking an LED*

Installing the RPi.GPIO package may or may not be necessary depending on your version of Raspbian (e.g. Raspbian Lite does not come with some Python packages pre-installed). enter the next in a terminal :

```
$ pip install rpi.gpio
```

In a new file, enter the following code:

The following complete Python program should result from combining the initialization and the blink code:

```
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library

from time import sleep # Import the sleep function from the time module

GPIO.setwarnings(False) # Ignore warning for now

GPIO.setmode(GPIO.BOARD) # Use physical pin numbering

GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 to be an output pin and set initial value to low (off)

while True: # Run forever

        GPIO.output(8, GPIO.HIGH) # Turn on

        sleep(1) # Sleep for 1 second

        GPIO.output(8, GPIO.LOW) # Turn off

        sleep(1) # Sleep for 1 second
```

With our program finished, save it as blinking_led.py and run it either inside your IDE or in the console with:

```
$ python blinking_led.py
```

Chapter 5: Creating IoT applications with Microcontroller and Microprocessor

Create Applications IoT with Arduino – ESP32

Many examples for Arduino – ESP32we can find

- https://projecthub.arduino.cc/
- https://docs.arduino.cc/built-in-examples/
- https://docs.arduino.cc/library-examples/
- https://wiki.keyestudio.com/Ks0068_keyestudio_37_in_1_Sensor_Kit_for_Arduino_Starters
- https://www.hackster.io/arduino/projects

White LED Light

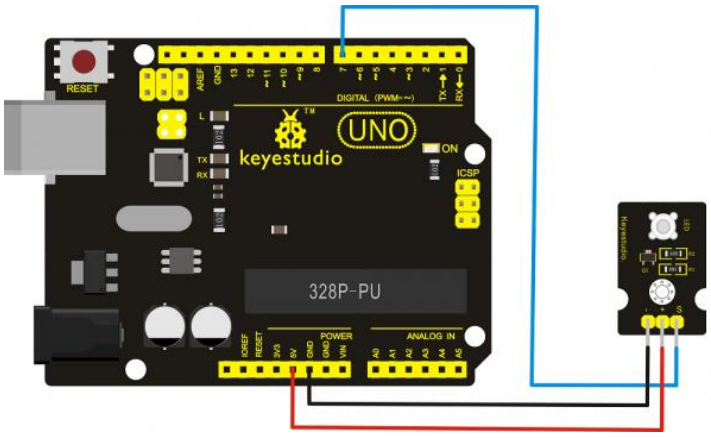
Introduction

This LED module is white. The primary use is to turn on and off a plug-in LED. After programming, it will produce a white light when connected to an Arduino. 3Pin has a pin pitch of 2.54mm. The S pin is for signal control; you may alter the High or Low level to turn the LED on and off. This LED module includes 3 pins: the - pin connects to ground, the + pin to VCC (3.3–5V), and the S pin is for signal control. To conduct a variety of interactive experiments, you may combine with different sensors. Other LED modules in colors including blue, green, yellow, and red are also an option.

Features

- Control interface: Digital
- Working voltage: DC 3.3-5V
- Pin pitch: 2.54mm
- LED color: white
- Easy to use
- Useful for light projects

Connect It Up Connect the LED module to control board using three jumper wires. Then connect the control board to your PC with a USB cable.



Upload the Code Copy and paste below code to Arduino IDE and upload.

```
int led = 7;
void setup()
{
  pinMode(led, OUTPUT);     //Set Pin7 as output
}
void loop()
{    digitalWrite(led, HIGH);   //Turn off led
        delay(1000);
        digitalWrite(led, LOW);    //Turn on led
        delay(1000);
}
```

Test Result The LED will flash on for one second, then off for one second, repeatedly and alternately. If it doesn't, make sure you have assembled the circuit correctly and verified and uploaded the code to your board.

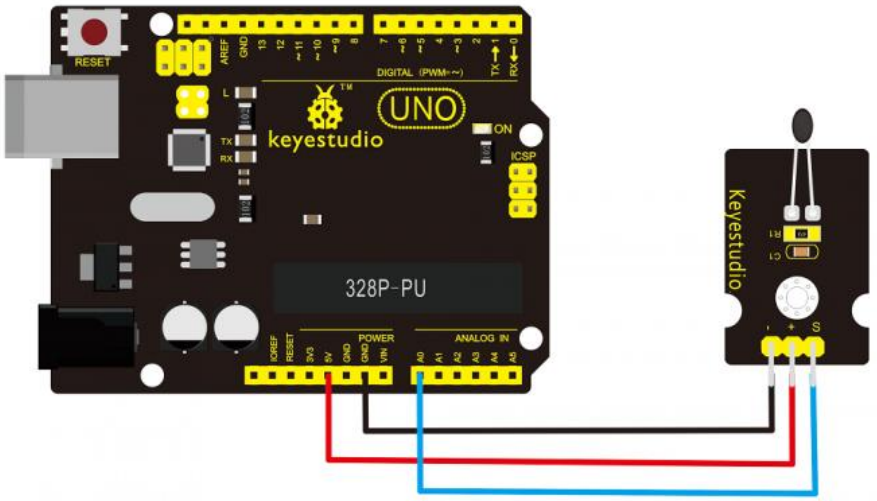
Analog Temperature

This module is based on how a thermistor works (resistance varies with temperature change in the environment). It has the ability to communicate data to the analog IO on the Arduino board when the temperature in its immediate environment changes. The sensor output data just has to be converted by simple programming into degrees Celsius temperature before being displayed. It is used extensively in gardening, home alarm systems, and other devices since it is both practical and efficient.

Specification

- Interface Type: analog
- Working Voltage: 5V
- Temperature Range: -55℃～315℃

Connection Diagram



Sample Code Copy and paste the below code to Arduino software.

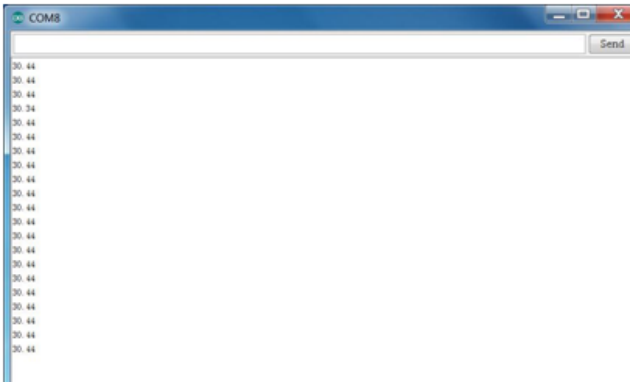
```
void setup()
{Serial.begin(9600);
}
// the loop routine runs over and over again forever:
void loop()
{int sensorValue = analogRead(A0);
Serial.println(sensorValue);
  delay(1); }
```

The above code is only for analog value. You can see that the analog value is changing according to the temperature change in the environment. But it's not very obvious. Let's solve this by using the following equation. Then upload the code below to the Arduino board. The value read from the serial port is similar to normal temperature. e.g. The temperature right now is 30°C.

```
#include <math.h>
void setup()
{
   Serial.begin(9600);
}
void loop()
{
  double val=analogRead(0);
  double fenya=(val/1023)*5;
  double r=(5-fenya)/fenya*4700;
  Serial.println( 1/(  log(r/10000) /3950 + 1/(25+273.15))-273.15);
  delay(1000);
}
```

Result

Done wiring and powered up, upload well the code, then open the serial monitor, you will see the current temperature value. Shown below.



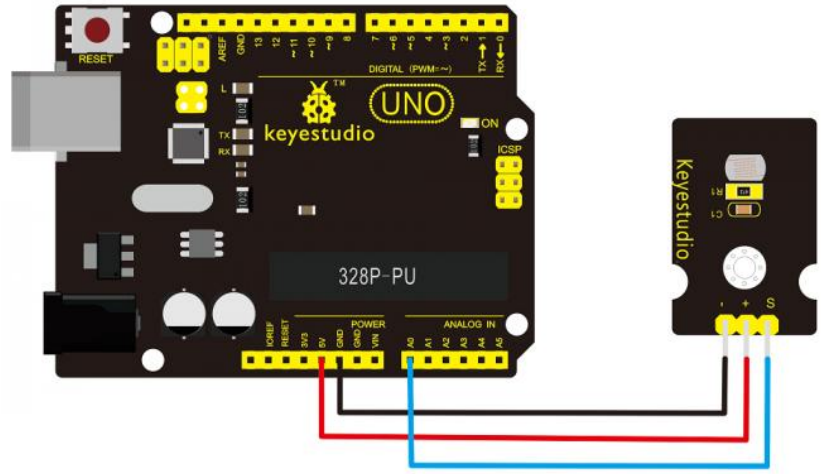
Photocell Sensor

Photocells are often employed in popular electrical designs and may be found in many aspects of everyday life, including intelligent switches. We provide the appropriate components to make it simpler and more efficient. Semiconductor is the photocell. It maintains great stability and dependability in difficult environmental conditions including high temperature and high humidity thanks to its advantages of high sensitivity, rapid response, spectrum characteristic, and R-value consistence. It is extensively used in automated control switch industries such cameras, lawn lamps, music cups, cameras, solar-powered garden lights, quartz clocks, tiny nightlights, sound and light control switches, etc.

Specification

- Interface Type: analog
- Working Voltage: 5V

Connection Diagram

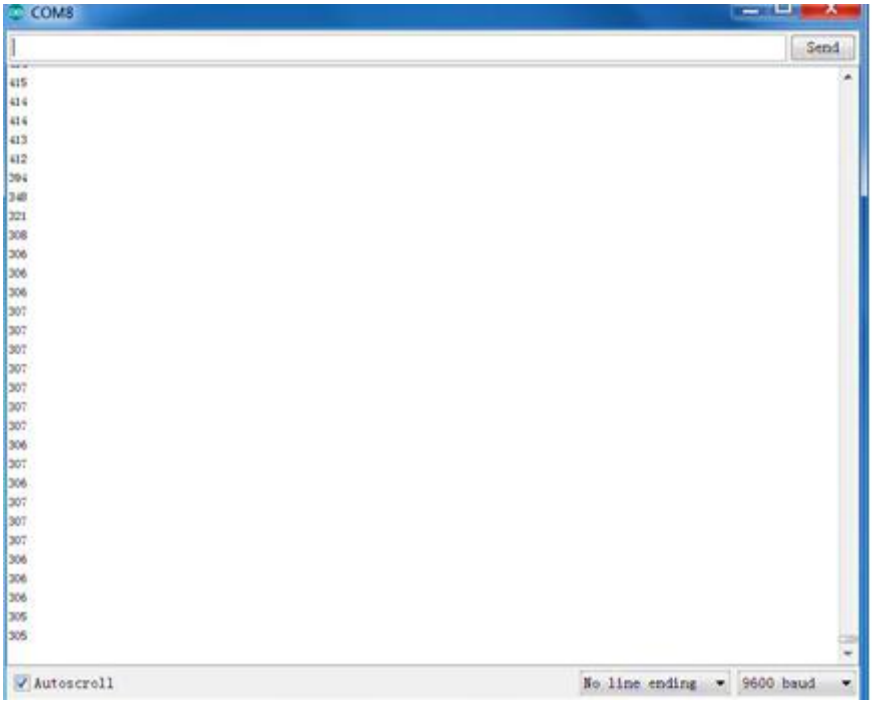


Sample Code

```
int sensorPin =A0 ;
int value = 0;
void setup()
{
 Serial.begin(9600); }
void loop()
{
 value = analogRead(sensorPin);

Serial.println(value, DEC);
```

```
delay(50);
}
```

Result Done wiring and powered up, upload well the code, then open the serial monitor, if cover the photocell on the sensor with your hand, you will see the analog value decrease. Shown as below.



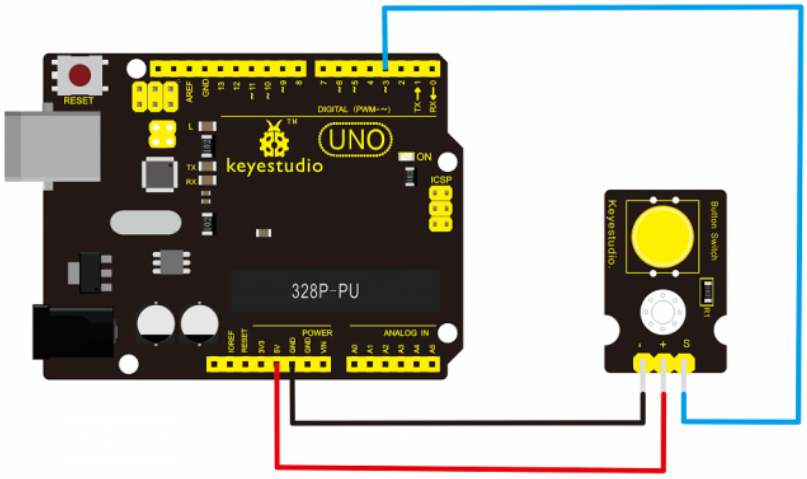
Digital Push Button

A simple button application module like this. Normally, a momentary pushbutton switch remains open. When it is depressed, the circuit is linked; when it is released, it returns to the disconnected state. For simple connection, the module contains three pins. To get started with Arduino, all you need to do is connect it into an IO shield.

Details

- Interface: Digital
- Supply Voltage: 3.3V to 5V
- Easy to plug and operate
- Large button keypad and high-quality button cap
- Standard assembling structure
- Easily recognizable pins
- Icons illustrate sensor function clearly
- Achieve interactive works

Connection Diagram



Sample Code

```
/* # When you push the digital button, the Led 13 on the board will turn on. Otherwise,the led turns off.
*/
int ledPin = 13;                // choose the pin for the LED
int inputPin = 3;               // Connect sensor to input pin 3
void setup() {
  pinMode(ledPin, OUTPUT);      // declare LED as output
  pinMode(inputPin, INPUT);     // declare pushbutton as input
}
void loop(){
  int val = digitalRead(inputPin);  // read input value
  if (val == HIGH) {            // check if the input is HIGH
    digitalWrite(ledPin, LOW);  // turn LED OFF
  } else {
    digitalWrite(ledPin, HIGH); // turn LED ON
  }
}
```

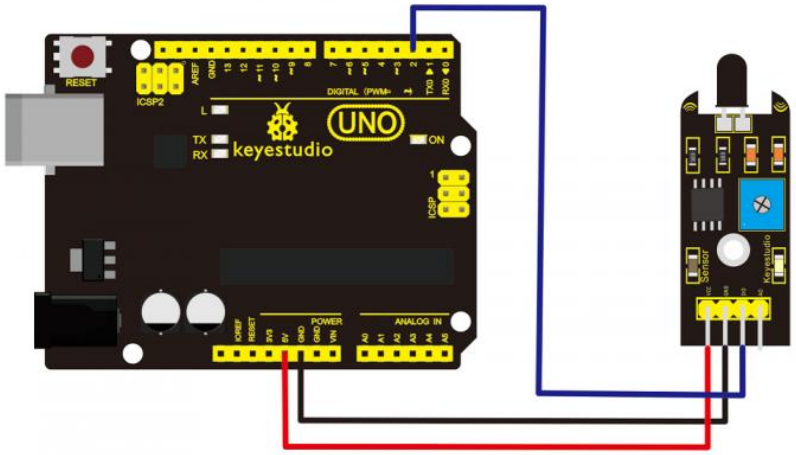
Flame Sensor

This flame sensor, whose wavelength ranges from 760 to 1100 nm, may be used to detect fire or other lights. The flame is a key component of the probe in the fire-fighting robot game, which the robot may utilize as its eyes to locate the source of the fire.

Specification

- Supply Voltage: 3.3V to 5V
- Detection Range: 20cm (4.8V) ~ 100cm (1V)
- Rang of Spectral Bandwidth: 760nm to 1100nm
- Operating Temperature: -25℃ to 85℃
- Interface: Digital

Connection Diagram Connect the D0 pin to digital 2, GND pin to GND port, VCC pin to 5V port.

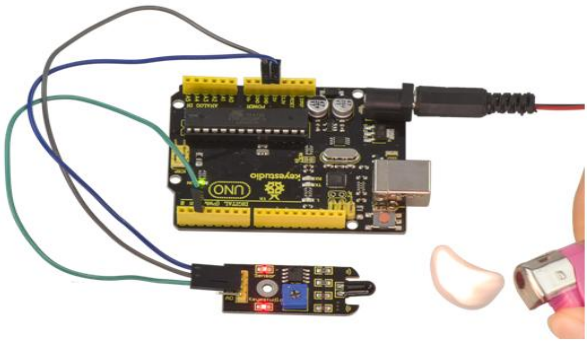


Sample Code

```
const int flamePin = 2;     // the number of the flame pin
```

```
const int ledPin =  13;      // the number of the LED pin
// variables will change:
int State = 0;         // variable for reading status
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(flamePin, INPUT);
}
void loop(){
  // read the state of the value:
State = digitalRead(flamePin);
  if (State == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
}
```

Result Done wiring and powered up, upload well the code to the board. Then if you put a lighter close to the sensor, when the sensor detects the flame, another led on the sensor is turned on.

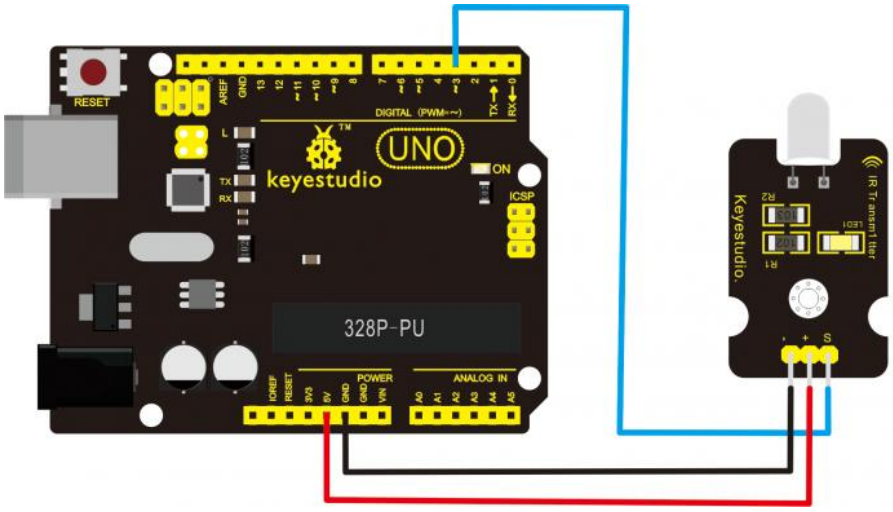


Digital IR Transmitter

The IR Transmitter Module is designed for IR communication, which is often used to control television equipment from close range. In most cases, the term "remote" refers to the remote control. In order to activate the target device, infrared (IR) remote controllers need line of sight. Mirrors, like any other reflective surface, may, nevertheless, reflect the signal. Numerous brands of IR extenders are offered for this purpose on the market if operation is necessary when line of sight is not allowed, such as when operating equipment that is located in a cabinet or in another room. The majority of them feature an IR receiver that picks up the IR signal and sends it wirelessly to the remote component, which contains an IR transmitter that imitates the original IR control. Additionally, infrared receivers often have a restricted working angle that is mostly determined by the phototransistor's optical properties. However, placing a matte clear item in front of the receiver makes it simple to widen the working angle.

Specification	
• Power Supply: 3-5V • Infrared Center Frequency: 850nm-940nm • Infrared Emission Angle: about 20degree • Infrared Emission Distance: about 1.3m (5V 38Khz) • Mounting Hole: inner diameter is 3.2mm, spacing is 15mm	

Connection Diagram



Sample Code 1:

```
int led = 3;
```

```
void setup() {
  pinMode(led, OUTPUT);
}
void loop() {
 digitalWrite(led, HIGH);
 delay(1000);
 digitalWrite(led, LOW);
 delay(1000);
}
```

When utilizing the camera to capture the infrared LED in the dark surroundings, you will notice a flickering blue light on the phone's screen. If the aforementioned code is properly uploaded to the board, the sensor's led will begin to flash red. Let's move on to an interactive example between an IR receiver and IR transmitter module in the paragraphs that follow.

Infrared Remote/Communication:

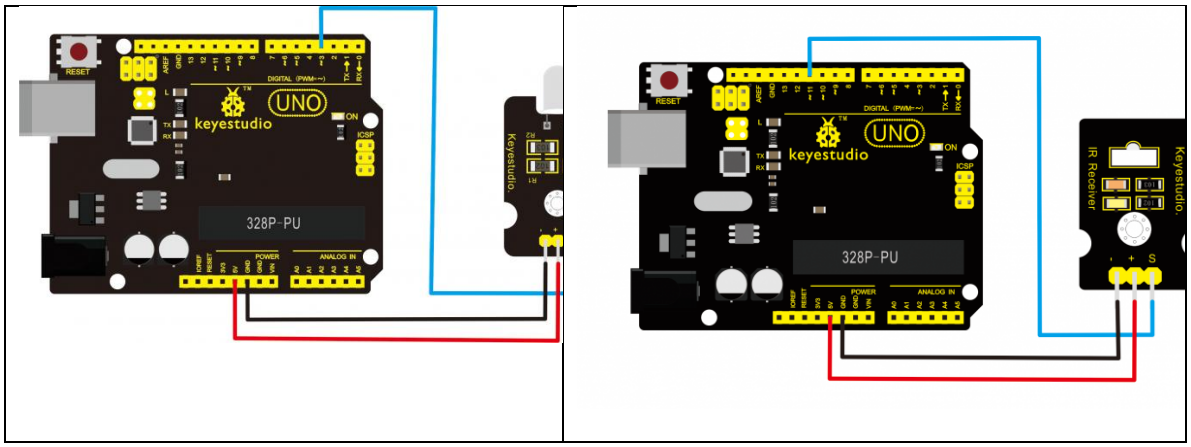
Hardware List

- UNO R3 x2
- Digital IR Receiver x1
- IR Transmitter Module x1

Get Arduino library Arduino-IRremote and install it.

Note: here if you have no two main boards, you can replace it with the breadboard for connection, may be more easier and convenient.

Connection Diagram For IR Transmitter: Notice: Arduino-IRremote only supports D3 as transmitter.	**For IR Receiver:** connect it to D11 port.



Upload code 2 to the UNO connected with IR Transmitter:

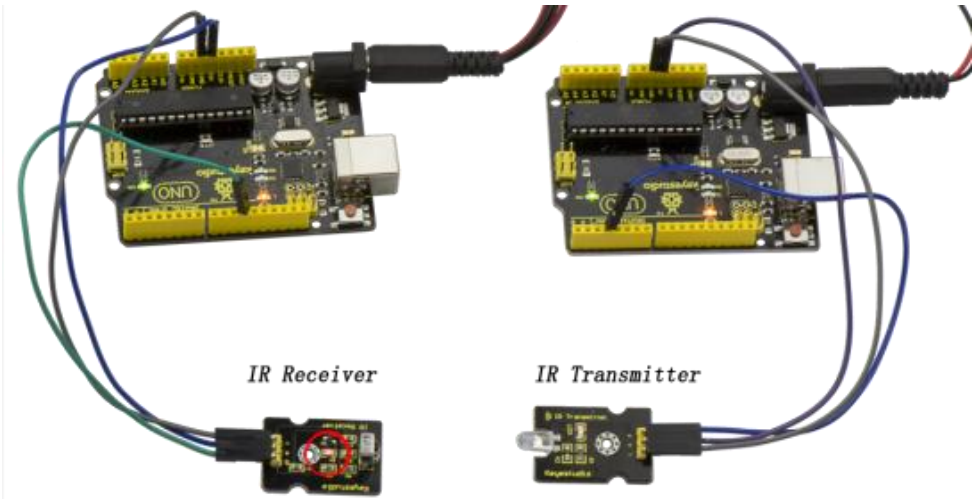
```
#include <IRremote.h>
 IRsend irsend;
void setup()
{}
void loop() {
 irsend.sendRC5(0x0, 8); //send 0x0 code (8 bits)
    delay(200);
 irsend.sendRC5(0x1, 8);
    delay(200); }
```

Upload code 3 to the UNO connected with IR Receiver:

```
#include <IRremote.h>
 const int RECV_PIN = 11;
 const int LED_PIN = 13;
 IRrecv irrecv(RECV_PIN);
```

```
decode_results results;
void setup()
{Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver
}
 void loop()
{if (irrecv.decode(&results))
  { if ( results.bits > 0 )
    {
      int state;
      if ( 0x1 == results.value )
      {
        state = HIGH;
      }
      else
      {
              state = LOW;
      }
                          digitalWrite( LED_PIN, state );
    }
 irrecv.resume();        // prepare to receive the next value
 }}
```

Test Result When IR Receiver module receives the infrared signal from IR Transmitter, D1 led on the IR Receiver module will blink.Shown as below figure.

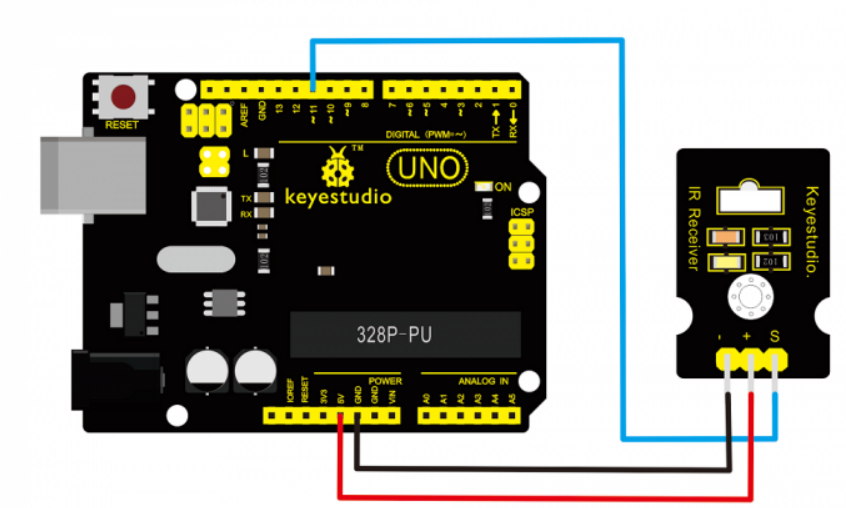


Digital IR Receiver

The usage of IR in remote controls is common. If you have the proper decoder, your Arduino project can accept commands from any IR remote controller. Well, utilizing an IR transmitter, creating your own IR controller will also be simple.

Specification	
• Power Supply: 5V • Interface: Digital • Modulation Frequency: 38Khz	

Connection Diagram The proposed connecting technique is seen in the figure below. Any Digital I/O pin that isn't being used by another device may be used.



NOTE: In the sample code below Digital pin 11 is in use, you may either change your wiring or change the sample code to match.

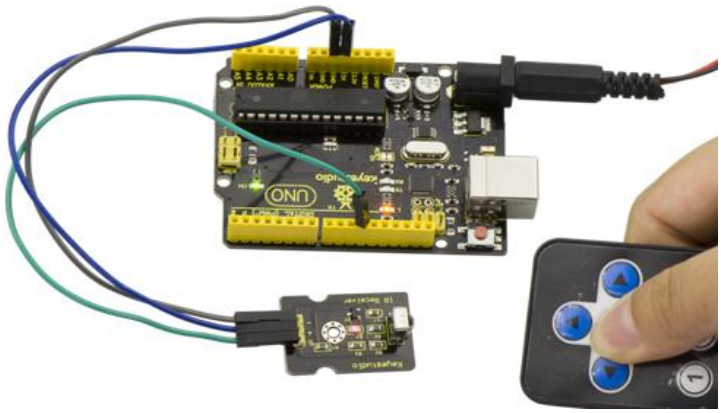
Sample Code Note: before compiling the code, do remember to place the library into libraries directory of Arduino IDE. Otherwise, compiling will fail.

```
#include <IRremote.h>
 int RECV_PIN = 11;
 IRrecv irrecv(RECV_PIN);
 decode_results results;
 void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver
}
 void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
```

```
    irrecv.resume(); // Receive the next value
  }
}
```

IR Remote Library includes some sample codes for sending and receiving: IR Remote Library

Result Done wiring and uploading the code, then control the IR receiver module by an infrared remote control, D1 led will flash. Shown as below.



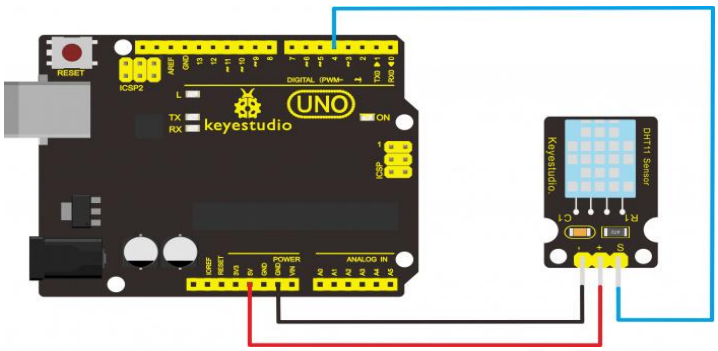
DHT11 Temperature and Humidity Sensor

This composite DHT11 temperature and humidity sensor has an output of calibrated digital signals for both temperature and humidity. High dependability and exceptional long-term stability are guaranteed by its technology. There is a connection to a powerful 8-bit microprocessor. This sensor has a resistive component as well as a sense of wet NTC temperature sensors. It offers benefits in terms of high cost performance, outstanding quality, quick reaction, and interference resistance. Every DHT11 sensor comes with very precise calibration data from a humidity calibration chamber. Internal sensors pick up signals throughout the operation, and we should refer to them as the calibration coefficients that are stored in the OTP program memory. It is fast and simple thanks to the integration of the single-wire serial interface technology. Its tiny size,

low power consumption, and 20-meter signal transmission range make it suitable for a variety of applications—even the most difficult ones. Connection that is convenient, and users may request certain packages.

Specification	
• Supply Voltage: +5 V • Temperature Range: 0-50 °C error of ± 2 °C • Humidity: 20-90% RH ± 5% RH error • Interface: Digital	

Connection Diagram



Sample Code Download the DHT11Lib.Or to see the website

Note: before compiling the code, do remember to place the library into libraries directory of Arduino IDE. Otherwise, compiling will fail.

```
#include <dht11.h>
dht11 DHT;
```

```
#define DHT11_PIN 4

void setup(){
  Serial.begin(9600);
  Serial.println("DHT TEST PROGRAM ");
  Serial.print("LIBRARY VERSION: ");
  Serial.println(DHT11LIB_VERSION);
  Serial.println();
  Serial.println("Type,\tstatus,\tHumidity (%),\tTemperature (C)");
}

void loop(){
  int chk;
  Serial.print("DHT11, \t");
  chk = DHT.read(DHT11_PIN);    // READ DATA
  switch (chk){
    case DHTLIB_OK:
                Serial.print("OK,\t");
                break;

case DHTLIB_ERROR_CHECKSUM:

                Serial.print("Checksum error,\t");
                break;
    case DHTLIB_ERROR_TIMEOUT:
                Serial.print("Time out error,\t");
                break;
    default:
```

```
            Serial.print("Unknown error,\t");
            break;
  }
 // DISPLAT DATA
  Serial.print(DHT.humidity,1);
  Serial.print(",\t");
  Serial.println(DHT.temperature,1);

  delay(1000);
}
```

Result Wire it up well and upload the above code to UNO board.

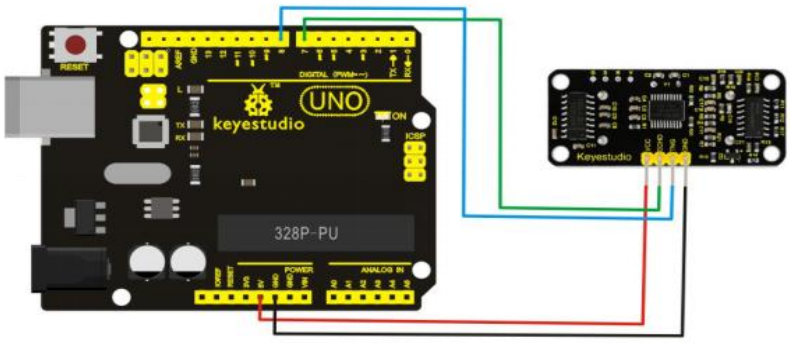
The current temperature and humidity value will then appear after you open the serial monitor and set the baud rate to 9600.

Ultrasonic Sensor

The Ultrasonic Sensor is a very reasonably priced proximity/distance sensor that has mostly been utilized in robotics projects for obstacle avoidance. Basically, it provides your Arduino eyes and spatial awareness, which can save your robot from colliding with objects or tumbling off a table. Additionally, it has been utilized as a parking sensor, a water level sensor, and in turret applications. This easy project will create a cute little interactive display on your computer screen using an Arduino, an Ultrasonic Sensor, and a Processing script.

Specification	
• Working Voltage: DC 5V • Working Current: 15mA • Working Frequency: 40Hz • Max Range: 5m • Min Range: 2cm • Measuring Angle: 15 degree • Trigger Input Signal: 10μS TTL pulse • Echo Output Signal Input TTL lever signal and the range in proportion	

Connection Diagram



Sample Code

- VCC to arduino 5v
- GND to arduino GND
- Echo to Arduino pin 7

- Trig to Arduino pin 8

```
#define echoPin 7 // Echo Pin
#define trigPin 8 // Trigger Pin
#define LEDPin 13 // Onboard LED

int maximumRange = 200; // Maximum range needed
int minimumRange = 0; // Minimum range needed
long duration, distance; // Duration used to calculate distance

void setup() {
 Serial.begin (9600);
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
 pinMode(LEDPin, OUTPUT); // Use LED indicator (if required)
}

void loop() {
/* The following trigPin/echoPin cycle is used to determine the
 distance of the nearest object by bouncing soundwaves off of it. */
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);

 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);

 duration = pulseIn(echoPin, HIGH);
```

```
//Calculate the distance (in cm) based on the speed of sound.
distance = duration/58.2;

if (distance >= maximumRange || distance <= minimumRange){
/* Send a negative number to computer and Turn LED ON
to indicate "out of range" */
Serial.println("-1");
digitalWrite(LEDPin, HIGH);
}
else {
/* Send the distance to the computer using Serial protocol, and
turn LED OFF to indicate successful reading. */
Serial.println(distance);
digitalWrite(LEDPin, LOW);
}

//Delay 50ms before next reading.
delay(50);
}
```

Result Open the serial monitor in the Arduino IDE after uploading the code to the board to see the distance value determined by the ultrasonic sensor.

5V Relay Module

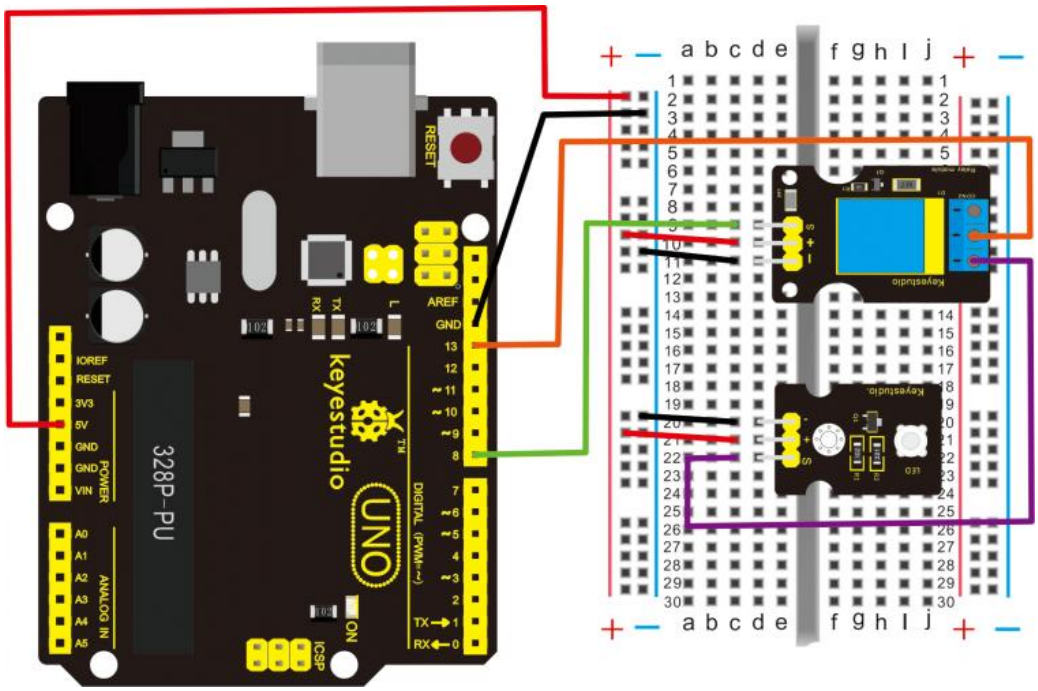
Projects that need interaction may utilize this single relay module. It is HIGH level active. This module makes use of a premium SONGLE 5v relay. It may also be used to operate electrical, lighting, and other machinery. With the Arduino board, expanding is simple thanks to the modular architecture (not included). A light-emitting diode powers the relay output. Such as

solenoid valves, lights, motors, and other high current or high voltage devices, it may be controlled by a digital IO interface.

Specification	
• Type: Digital • Rated current: 10A (NO) 5A (NC) • Maximum switching voltage: 150VAC 24VDC • Digital interface • Control signal: TTL level • Rated load: 8A 150VAC (NO) 10A 24VDC (NO), 5A 250VAC (NO/NC) 5A 24VDC (NO/NC) • Maximum switching power: AC1200VA DC240W (NO) AC625VA DC120W (NC) • Contact action time: 10ms	

Connection Diagram Firstly you need to prepare the following parts by yourself before testing.

- Arduino Board*1
- Single relay module*1
- LED module *1
- USB Cable*1
- Jumper Wire*8



Sample Code Copy and paste the code below to Arduino software.

```
int Relay = 8;
  void setup()
{
  pinMode(13, OUTPUT);         //Set Pin13 as output
  digitalWrite(13, HIGH);     //Set Pin13 High
  pinMode(Relay, OUTPUT);     //Set Pin3 as output
}
void loop()
{
        digitalWrite(Relay, HIGH);   //Turn off relay
        delay(2000);
        digitalWrite(Relay, LOW);    //Turn on relay
```

```
        delay(2000);
}
```

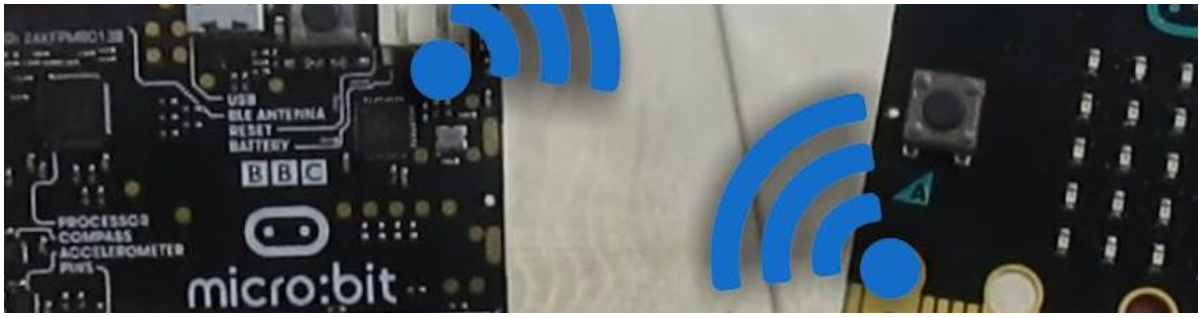
Test Result This relay module has a HIGH degree of activity. Upload the aforementioned code to the board after properly wiring and powering it up. You will see that the relay is continually and circularly switched on for two seconds (ON connected, NC unconnected), then off for two seconds (NC closed, ON disconnected). An external LED turns on when the relay is switched on. External LED is off if relay is switched off.

Create Applications IoT with Microbit

On the Microbit page (microbit.org) one can find many applications

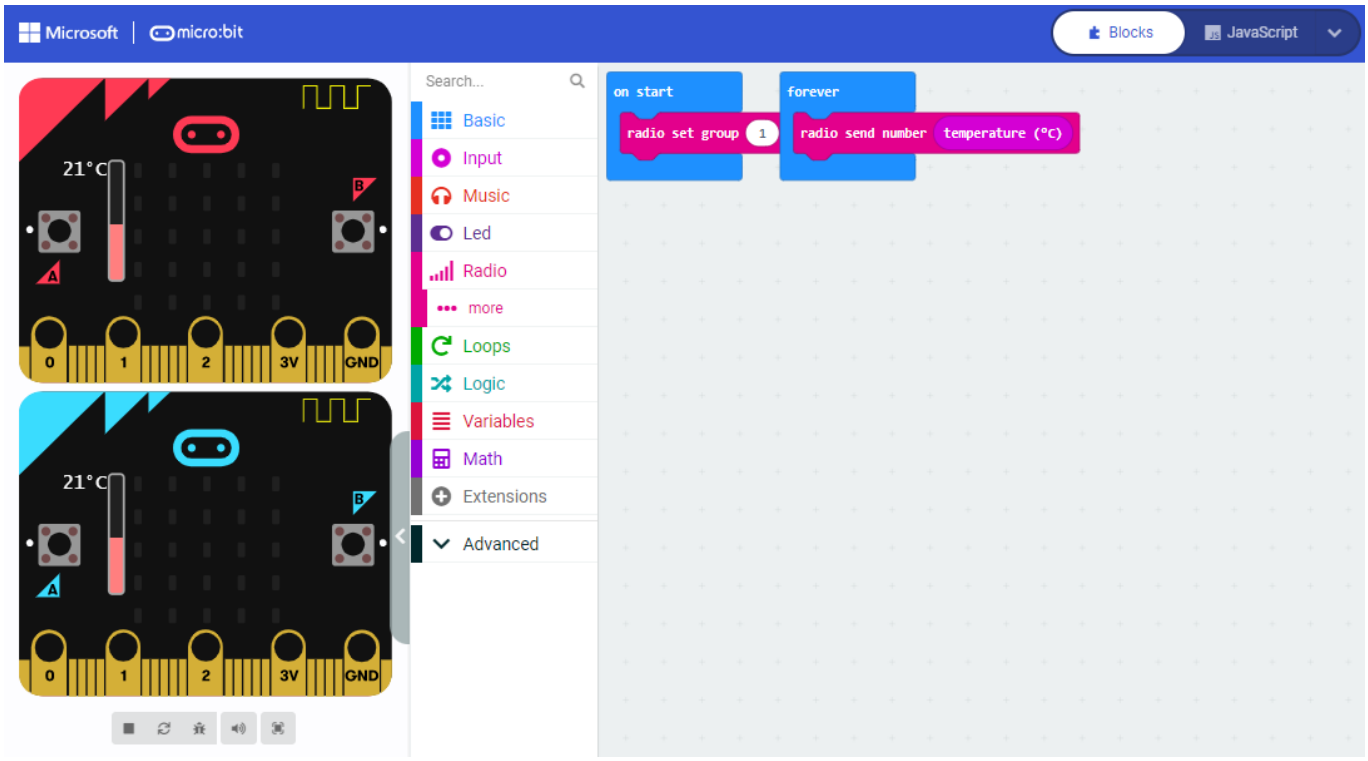
Micro:bit Radio Communication

A simple radio communication experiment using two Micro:bit boards, which can transmit and receive command to/from each other. Watch the project below to see it in action, then follow the instructions to build your own!

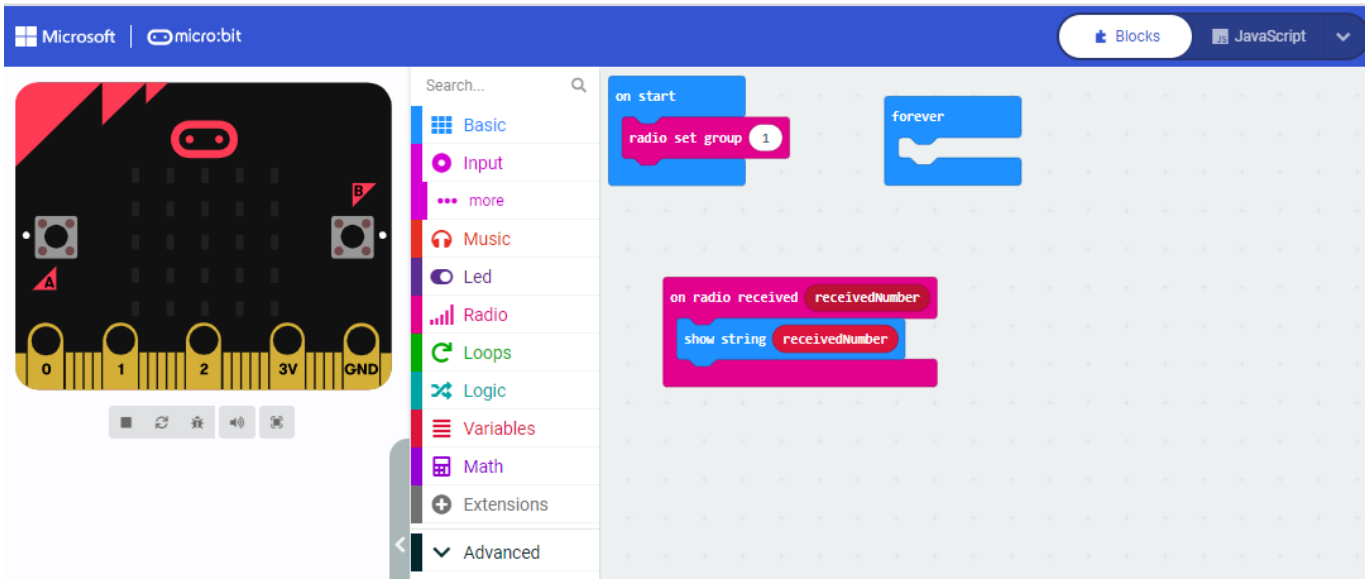


First, open Microsoft Makecode for Micro:bit site (https://makecode.microbit.org/) to create commands for Micro:bit. We define the communication channel to be the 1.

If using Blocks, arrange the code like the following picture: **SEND Micro:bit**

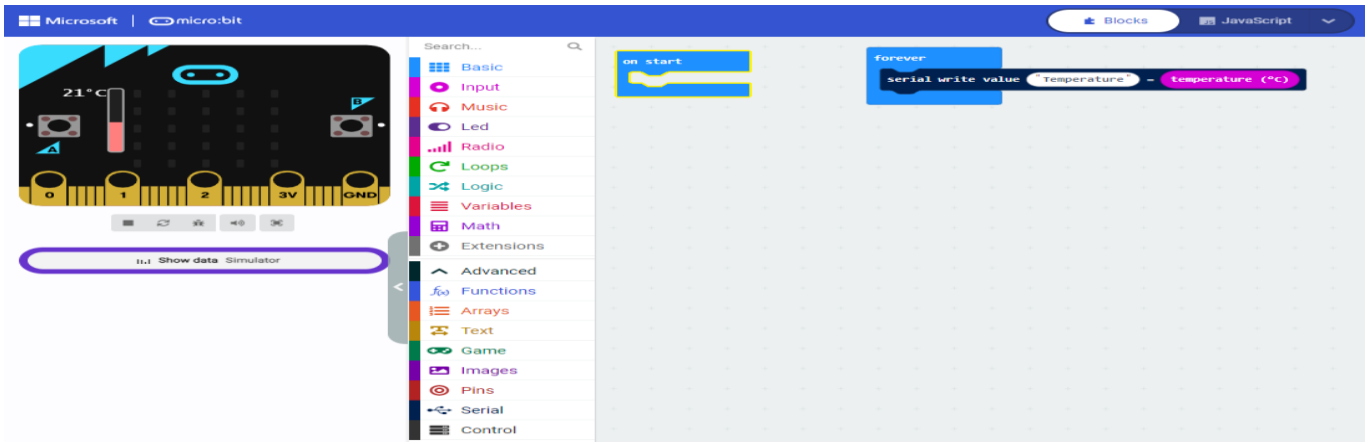


If using Blocks, arrange the code like the following picture: **RECEIVE Micro:bit**



Micro:bit save CSV file

To save the data in a file we select the serial



The button to extract the csv file may be found in the upper right corner after selecting the display data simulation to view the data being graphed.


Microsoft | micro:bit
Blocks
JavaScript
Go back
Simulator
28°C
0
1
2
3V
GND
Show data
Simulator
Temperature: 28
Temperature:28
Temperature:28
Temperature:28
Temperature:28
Temperature:28
Temperature:28
Temperature:28
Temperature:28
Temperature:28
Temperature:28

Create Applications IoT with Raspberry

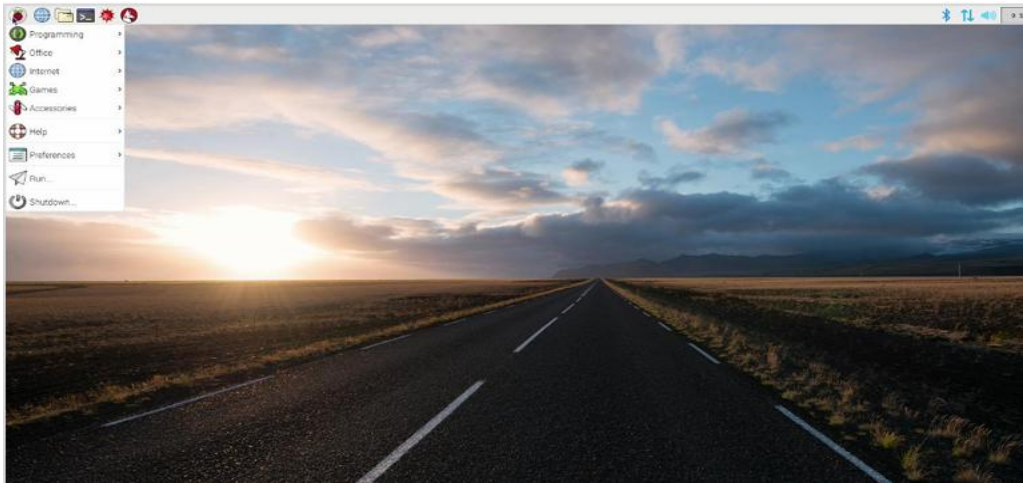
Install Raspberry Pi OS

Your Raspberry Pi (Raspberry Pi, n.d.) needs an operating system to work. Raspberry Pi OS (previously called Raspbian) is our official supported operating system.

The quickest and easiest method to install Raspberry Pi OS and other operating systems on a microSD card so that it is ready to use with your Raspberry Pi is to use the Raspberry Pi Imager. Watch our 45-second video to learn how to install an operating system using Raspberry Pi Imager.

A PC with an SD card reader should have Raspberry Pi Imager downloaded and installed. Run Raspberry Pi Imager after inserting the SD card that your Raspberry Pi will be using.

When it comes to connecting with the Raspberry Pi, you have a few alternatives. Utilizing it as you would a full desktop computer is the first and most popular option (just smaller). Connecting a keyboard, mouse, and monitor is required. Installing Raspbian with Desktop, which offers you a complete graphical user interface (GUI) to operate with, is probably ideal in this arrangement. This is your greatest choice if you desire a working environment that is comparable to that of other operating systems (OS), including Windows, macOS, or other well-liked Linux variations, like Ubuntu.



Some of the installed apps are:

- LibreOffice The most used set of productivity tools is this one. Word processing, spreadsheets, and presentations make up the majority of it. It is available in the application menu's office area.
- Mathematica, is based on the Wolfram programming language and may be found in the programming part of the Application menu. It is used in technical and scientific computing.
- Minecraft Pi The world-building video game known as Minecraft is well known. Similar to that, Minecraft Pi is that on a Raspberry Pi. It is located in the Application program's Game section, and it may be created using the Python programming language.
- Python 2 and Python 3 Raspberry Pi provides us the Python programming language, This may be accessed via the Application menu's Programming section. Additionally, we have access to the Thonny IDE (integrated development environment), which offers Raspberry Pi users an alternate method of writing Python code.
- Python games Raspberry Pi has games such as Reversi, A snake game and a sliding puzzle game are both included in Four in a Row. These are all made using the

Python programming language and are accessible via the Application menu's Game section.

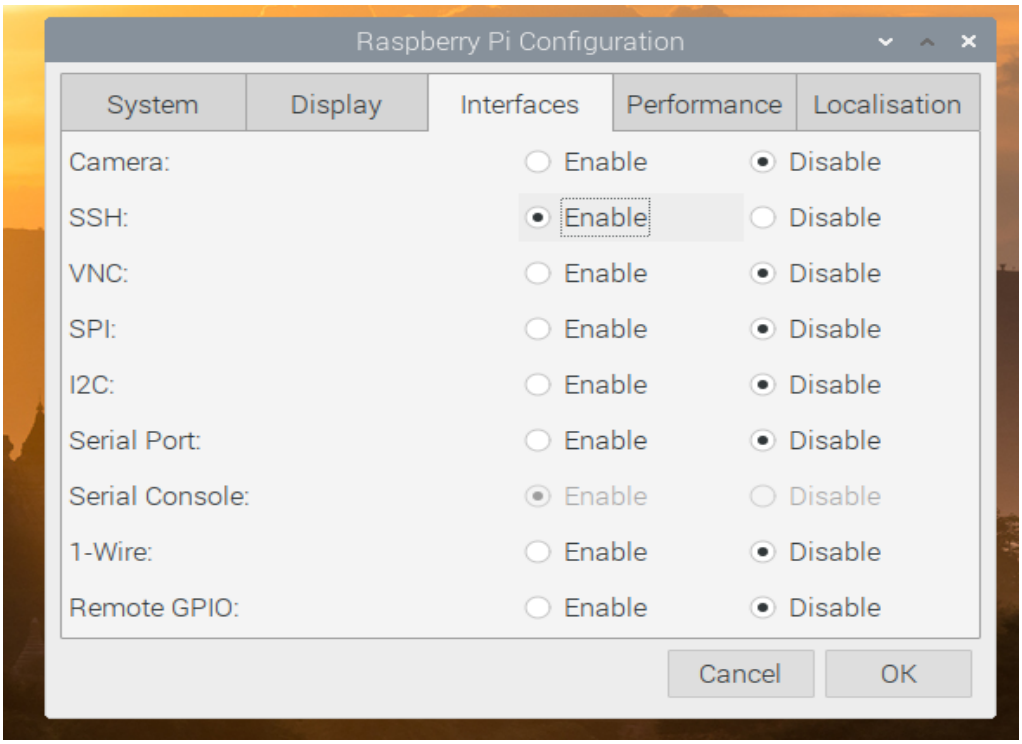
- Scratch Raspberry Pi Foundation offers us Scratch, a user-friendly programming language that is basic enough for individuals of all ages. It may be used to make animations and video games. Project management for electronic projects is also possible. It may be found under the Application menu's Programming section.
- Sense HAT emulator As the name implies, it has some built-in sensors that can be used for creating experiments and other projects. It is an add-on for the Raspberry Pi users, which can be found under the Programming section of the Application menu.
- Wolfram is a programming language that the Raspberry Pi Foundation makes available. It seeks to utilize information in order to speed up programming. You can get more information about this at www.wolfram.com/language.

Setup Raspberry Pi

Before you can connect to the Raspberry Pi over SSH, you'll need to enable SSH access inside the Raspberry Pi *Preferences* area. Enable SSH by going to the following file path:

Raspberry Pi Icon → Preferences → Raspberry Pi Configuration

Once the configuration appears, select the Interfaces tab and then enable the SSH option:



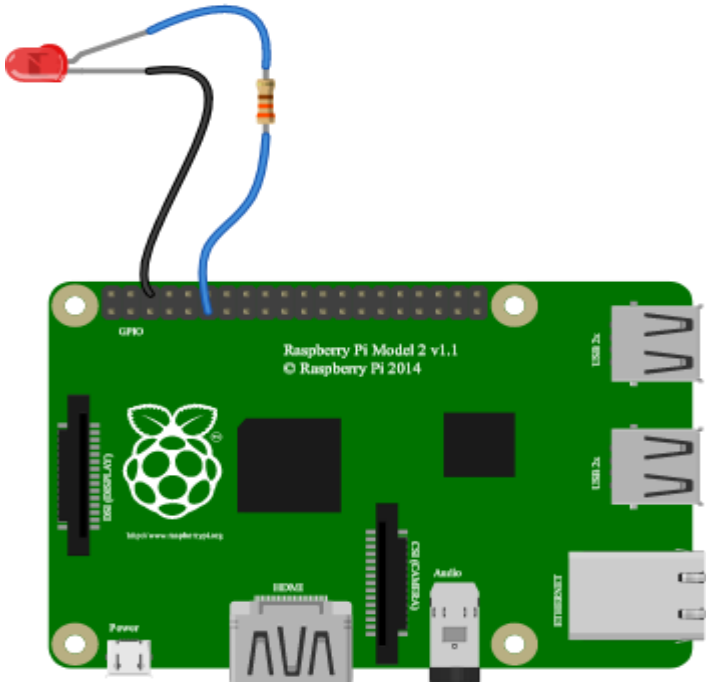
We can also arrange, Camera, VNC, I2C, Serial Port and Remote GPIO.

Install RPi.GPIO python library To install RPi.GPIO python library, type the following commands on terminal window of your Raspberry Pi:

```
sudo apt-get update

sudo apt-get install rpi.gpio

sudo apt-get install python3-pip

sudo apt-get install python-dev

sudo apt-get install python-rpi.gpio

pip install rpi.gpio
```

Raspberry Pi Pinout

The Raspberry Pi's ability to regulate the voltage on a number of its readily accessible pins is one of the factors that makes it superior to most other computers for learning electronics. You may see a header with 40 pins on the right side of your Raspberry Pi if you hold it in portrait orientation, facing up, like in the picture below. This header has several General Purpose Input/Output (GPIO) ports as well as outputs for 3.3V, 5V, Ground, and other voltages! (GpioZero, n.d.)



We can work with the library **RPi.GPIO**

```
import RPi.GPIO as GPIO
GPIO . setmode(GPIO . BOARD)
GPIO . setup(7, GPIO . OUT)

p = GPIO . PWM(7, 0.5)
```

```
p . start(1)
input('Press return to stop:')   # use raw_input for Python 2
p . stop()
GPIO . cleanup()
```

in the code below we use PWM to start slowly

```
import time
import RPi.GPIO as GPIO
GPIO . setmode(GPIO . BOARD)
GPIO . setup(7, GPIO . OUT)

p = GPIO . PWM(7, 50)  # channel=12 frequency=50Hz
p . start(0)
try:
   while 1:
      for dc in range(0, 101, 5):
         p . ChangeDutyCycle(dc)
         time . sleep(0.1)
      for dc in range(100, -1, -5):
         p . ChangeDutyCycle(dc)
         time . sleep(0.1)
except KeyboardInterrupt:
   pass
p . stop()
GPIO . cleanup()
```

We can work with the library **gpiozero**

```
from gpiozero import LED
from time import sleep

red = LED(17)

while True:
   red . on()
   sleep(1)
```

```
    red . off()
    sleep(1)
```

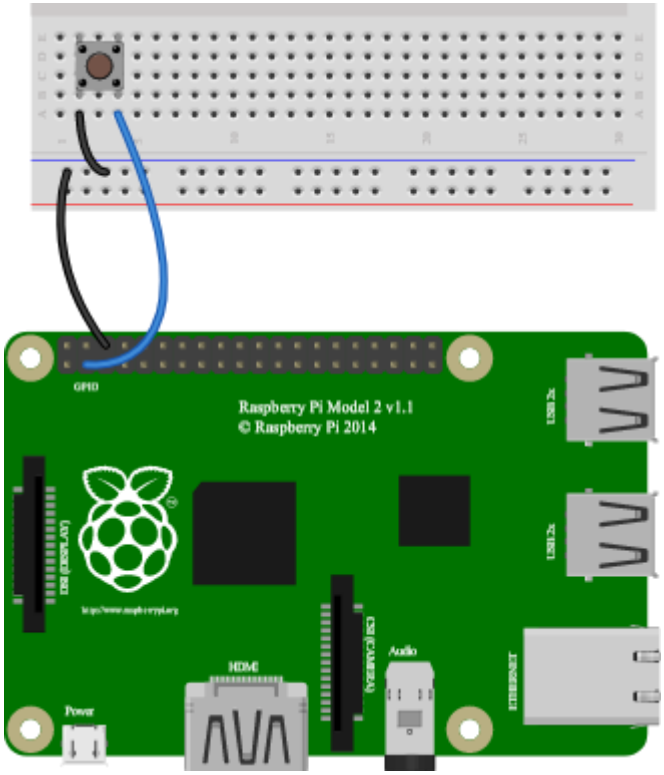
```
from gpiozero import LED
from signal import pause

red = LED(17)

red . blink()

pause()
```

```
from gpiozero import PWMLED
from time import sleep

led = PWMLED(17)

while True:
    led . value = 0  # off
    sleep(1)
    led . value = 0.5  # half brightness
    sleep(1)
    led . value = 1  # full brightness
    sleep(1)
```

```
from gpiozero import PWMLED
from signal import pause

led = PWMLED(17)

led . pulse()
```

```
pause()
```

Raspberry Pi Pinin



Check if a Button is pressed:

```
from gpiozero import Button

button = Button(2)

while True:
  if button.is_pressed:
    print("Button is pressed")
```

```
else:
    print("Button is not pressed")
```

Wait for a button to be pressed before continuing:

```
from gpiozero import Button

button = Button(2)

button.wait_for_press()
print("Button was pressed")
```

Run a function every time the button is pressed:

```
from gpiozero import Button
from signal import pause

def say_hello():
    print("Hello!")

button = Button(2)



pause()
```

Erasmus+

Node.js and Raspberry Pi

The Raspberry Pi has a row of GPIO (General Purpose input/output) pins, and these can be used to interact in amazing ways with the real world. This tutorial will focus on how to use these with Node.js. (Nodejs Raspberrypi, n.d.)

```
$sudo apt-get install -y nodejs
```

To interface with the GPIO on the Raspberry Pi using Node.js, we will use a Module called "onoff".

Install the onoff module using npm:

```
$npm install onoff
```

Now onoff should be installed and we can interact with the GPIO of the Raspberry Pi.

Write, or paste the following code (file blink.js):

```
var Gpio = require('onoff').Gpio; //include onoff to interact with the GPIO

var LED = new Gpio(4, 'out'); //use GPIO pin 4, and specify that it is output

var blinkInterval = setInterval(blinkLED, 250); //run the blinkLED function every 250ms

function blinkLED() { //function to start blinking

 if (LED.readSync() === 0) { //check the pin state, if the state is 0 (or off)
  LED.writeSync(1); //set pin state to 1 (turn LED on)
 } else {
  LED.writeSync(0); //set pin state to 0 (turn LED off)
 }
}

function endBlink() { //function to stop blinking
 clearInterval(blinkInterval); // Stop blink intervals
 LED.writeSync(0); // Turn LED off
```

ΠΑΝΕΛΛΗΝΙΟΣ ΣΥΝΔΕΣΜΟΣ ΕΠΙΧΕΙΡΗΣΕΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΕΦΑΡΜΟΓΩΝ, ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΝΕΩΝ ΤΕΧΝΟΛΟΓΙΩΝ

A.S.E.L.
ROMANIA

KOMPAKT

```
 LED.unexport(); // Unexport GPIO to free resources
}

setTimeout(endBlink, 5000); //stop blinking after 5 seconds
```

Press "Ctrl+x" to save the code. Confirm with "y", and confirm the name with "Enter".

Run the code:

```
$ node blink.js
```

Now the LED should blink for 5 seconds (10 times) before turning off again!

Node.js and Raspberry Pi - Webserver with WebSocket

WebSocket enables bidirectional communication in real time over the web.

WebSocket can be run together with a normal HTTP server. You can click a button in a web browser, and enable a GPIO on your Raspberry Pi which turns on a light in your house. All in real time, and with communication going both ways!

In this chapter, we will set up a web server with WebSocket. Then create a browser UI to interact with our earlier example of turning a LED on and off with a button.

$ npm install socket.io –save

Write the index.html file:

```
<!DOCTYPE html>
<html>
<body>

<h1>Control LED light</h1>
```

```
<p><input type="checkbox" id="light"></p>

<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.3/socket.io.js"></script>
<!-- include socket.io client side script -->
<script>
var socket = io(); //load socket.io-client and connect to the host that serves the page
window.addEventListener("load", function(){ //when page loads
 var lightbox = document.getElementById("light");
 lightbox.addEventListener("change", function() { //add event listener for when checkbox changes
   socket.emit("light", Number(this.checked)); //send button status to server (as 1 or 0)
 });
});
socket.on('light', function (data) { //get button status from client
 document.getElementById("light").checked = data; //change checkbox according to push button on Raspberry Pi
 socket.emit("light", data); //send push button status to back to server
});
</script>

</body>
</html>
```

And our webserver.js file:

We will use a lot of the code from the Pushbutton controlled LED chapter.

```
var http = require('http').createServer(handler); //require http server, and create server with function handler()
var fs = require('fs'); //require filesystem module
var io = require('socket.io')(http) //require socket.io module and pass the http object (server)
var Gpio = require('onoff').Gpio; //include onoff to interact with the GPIO
var LED = new Gpio(4, 'out'); //use GPIO pin 4 as output
var pushButton = new Gpio(17, 'in', 'both'); //use GPIO pin 17 as input, and 'both' button presses, and releases should be handled
```

```
http.listen(8080); //listen to port 8080

function handler (req, res) { //create server
 fs.readFile(__dirname + '/public/index.html', function(err, data) { //read file index.html in public folder
   if (err) {
    res.writeHead(404, {'Content-Type': 'text/html'}); //display 404 on error
    return res.end("404 Not Found");
   }
   res.writeHead(200, {'Content-Type': 'text/html'}); //write HTML
   res.write(data); //write data from index.html
   return res.end();
 });
}

io.sockets.on('connection', function (socket) {// WebSocket Connection
 var lightvalue = 0; //static variable for current status
 pushButton.watch(function (err, value) { //Watch for hardware interrupts on pushButton
   if (err) { //if an error
    console.error('There was an error', err); //output error message to console
    return;
   }
   lightvalue = value;
   socket.emit('light', lightvalue); //send button status to client
 });
 socket.on('light', function(data) { //get light switch status from client
   lightvalue = data;
   if (lightvalue != LED.readSync()) { //only change LED if status has changed
    LED.writeSync(lightvalue); //turn LED on or off
   }
 });
});

process.on('SIGINT', function () { //on ctrl+c
 LED.writeSync(0); // Turn LED off
```

```
  LED.unexport(); // Unexport LED GPIO to free resources
  pushButton.unexport(); // Unexport Button GPIO to free resources
  process.exit(); //exit completely
});
```

Lets test the server:

```
$ node webserver.js
```

Open the website in a browser using http://[RaspberryPi_IP]:8080/:

Now the server should output all the changes to the checkbox to the console on the Raspberry Pi.

The client is sending the changes to the server, and the server is responding.

End the program with Ctrl+c.

Raspberry Pi Sense HAT

One of the great things about Raspberry Pi is the plethora of HATs (hardware attached on top), boards that connect to the GPIO pins and provide extra functionality such as lights, sensors and motors. The Raspberry Pi Foundation's own Sense HAT provides a great opportunity for kids and adults to learn about programming while creating fun, educational projects.

As its name implies, the Raspberry Pi Sense HAT has sensors to measure temperature, humidity, pressure, magnetic forces, orientation, acceleration. It also has a lovely 8 x 8 grid of multicolor LEDs on which we can scroll text and display pixelated images. The joystick provides us with basic input to use in our projects. The Sense HAT works with every model of Raspberry Pi, from the original B+ to the current Raspberry Pi 4, and it can be used with a

variety of programming languages, including Scratch 3, a simple but powerful block-based language for kids.

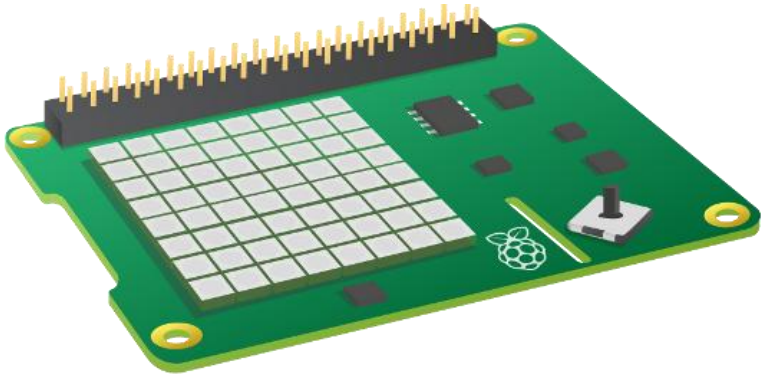


Figure 46 Raspberry Pi Sense HAT

In this project, you will learn how to control the Sense HAT's LED matrix and collect sensor data, and you will combine these ideas in a number of small projects.

The Sense HAT features an 8x8 RGB LED matrix, a mini joystick and the following sensors:

- Gyroscope
- Accelerometer
- Magnetometer
- Temperature
- Humidity
- Barometric pressure

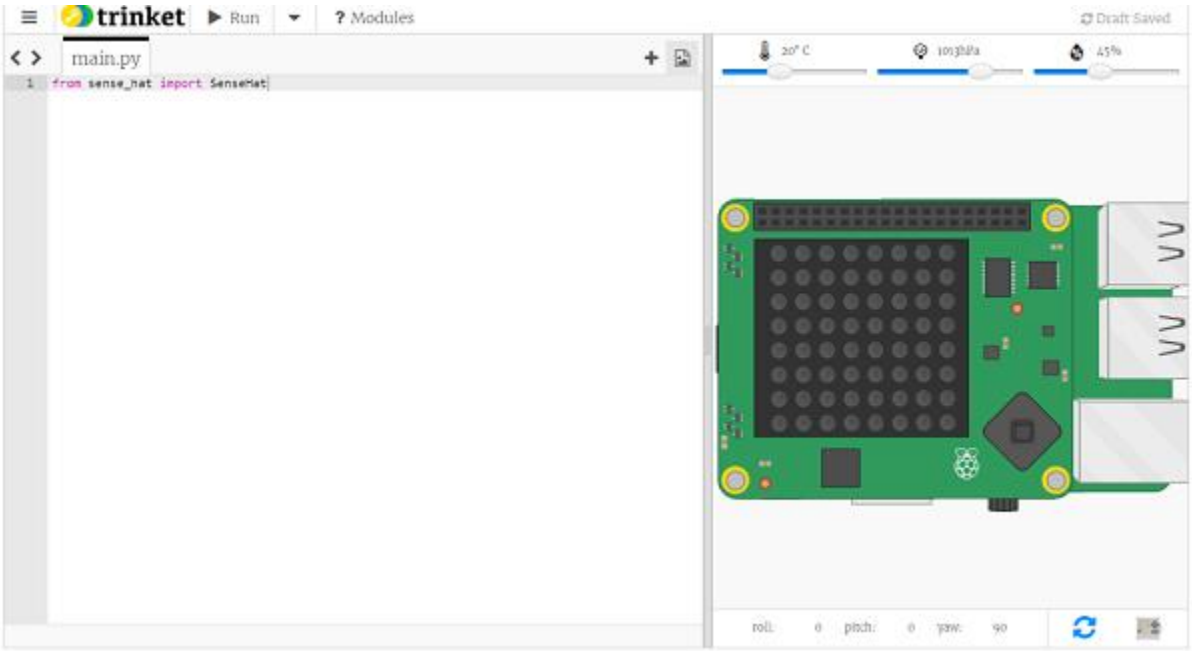
Type this command into the terminal to install the Sense HAT package:

```
sudo apt-get install sense-hat
```

If you don't have access to a Sense HAT, you can use the emulator.

Online Sense HAT emulator

There is an online emulator you can use in your browser to write and test code for the Sense HAT.



- Open an internet browser, go to https://trinket.io/sense-hat and delete the existing demo code which is in the editor.

- If you are using a Raspberry Pi, there is a Sense HAT emulator included in the Raspbian operating system. From the main menu, select Programming > Sense HAT emulator to open a window containing the emulator.

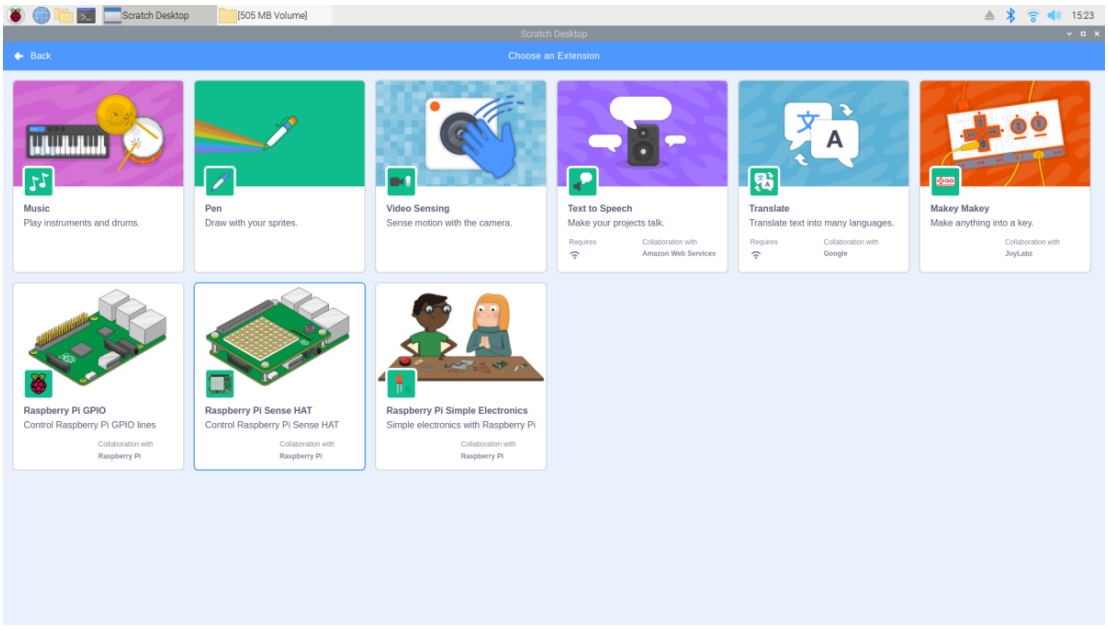
Raspberry Pi Sense HAT and Scratch 3

1. Attach the Sense HAT With the power off, connect the Sense HAT to all 40 pins of the GPIO, ensuring that the board fits neatly over the Pi. Use the brass standoff, provided with the Sense HAT to fix the board firmly in place.

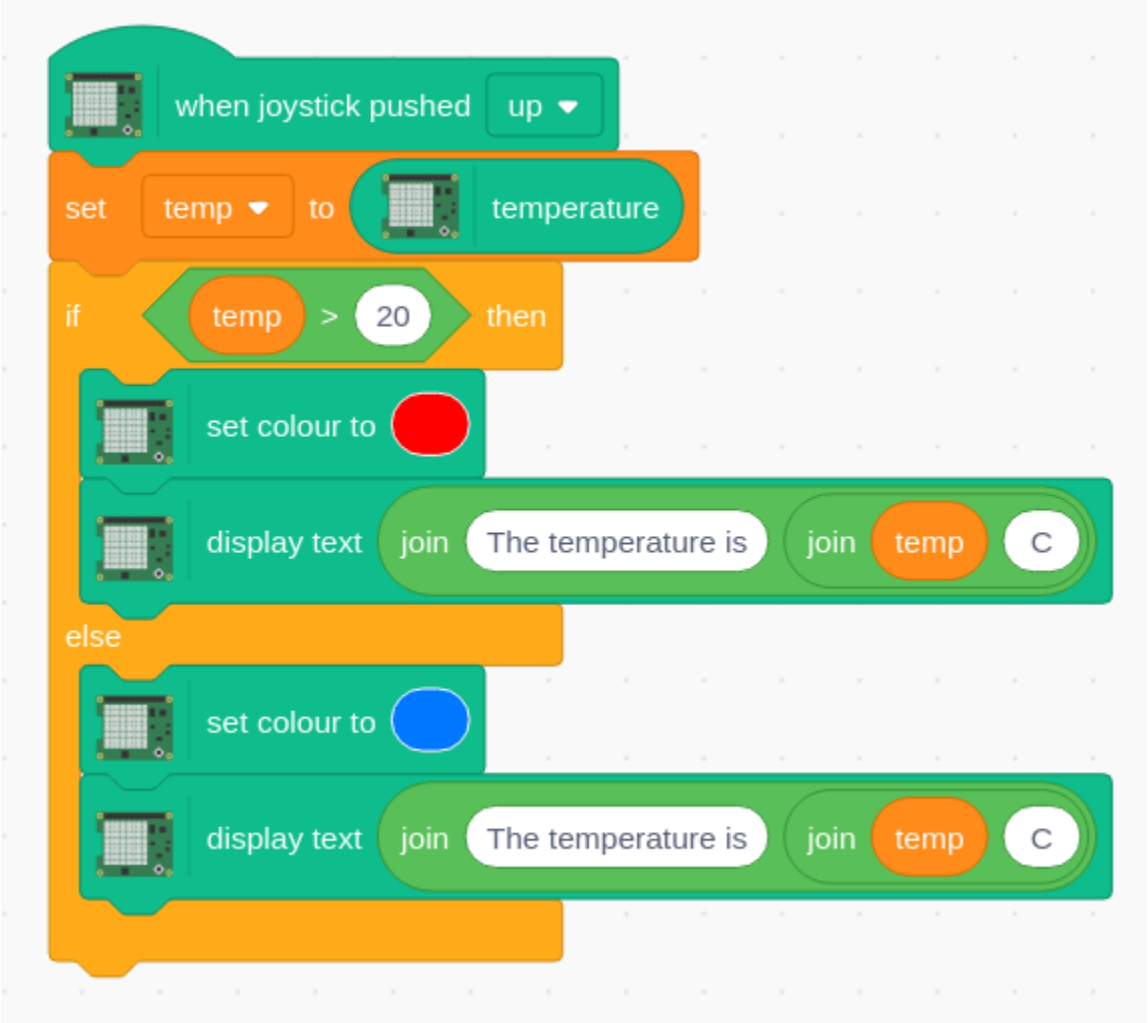
2. Enable I2C in raspi-config (you can get there by typing “sudo raspi-config” at the command prompt then navigating to Interfacing Options->I2C. Reboot after.
3. Install Scratch 3 (in Raspberry Pi) if it’s not already installed. The easiest way is to type

```
sudo apt-get install scratch3
```

After running Scratch 3, install the Sense HAT Scratch 3 extension. To use the Sense HAT we need to install an extension, a library of code that Scratch uses to provide extra blocks. Click on the icon to the bottom left of the screen and a series of extensions will be offered. Look for Raspberry Pi Sense HAT and click on it to install. In a few moments a new section of blocks will appear and we are ready to make our first project.



Scratch 3 Joystick Input, Temperature Reading with Raspberry Pi Sense HAT



Raspberry Pi Sense HAT with Python

Basic (Pyhton sense-hat, n.d.) commands.

show_message

```
from sense_hat import SenseHat
```

```
sense = SenseHat()
sense.show_message("Hello world!")
```

set_rotation

```
from sense_hat import SenseHat
sense = SenseHat()
sense.set_rotation(180)
# alternatives
sense.rotation = 180
```

set_pixels

```
from sense_hat import SenseHat
sense = SenseHat()
X = [255, 0, 0]  # Red
O = [255, 255, 255]  # White
question_mark = [
O, O, O, X, X, O, O, O,
O, O, X, O, O, X, O, O,
O, O, O, O, O, X, O, O,
O, O, O, O, X, O, O, O,
O, O, O, X, O, O, O, O,
O, O, O, X, O, O, O, O,
O, O, O, O, O, O, O, O,
O, O, O, X, O, O, O, O
]
sense.set_pixels(question_mark)
```

get_pixels

```
from sense_hat import SenseHat
sense = SenseHat()
pixel_list = sense.get_pixels()
```

```
from sense_hat import SenseHat
sense = SenseHat()
# examples using (x, y, r, g, b)
sense.set_pixel(0, 0, 255, 0, 0)
sense.set_pixel(0, 7, 0, 255, 0)
sense.set_pixel(7, 0, 0, 0, 255)
sense.set_pixel(7, 7, 255, 0, 255)
red = (255, 0, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
# examples using (x, y, pixel)
sense.set_pixel(0, 0, red)
sense.set_pixel(0, 0, green)
sense.set_pixel(0, 0, blue)
```

load_image

```
from sense_hat import SenseHat
sense = SenseHat()
sense.load_image("space_invader.png")
```

get_humidity

```
from sense_hat import SenseHat
sense = SenseHat()
humidity = sense.get_humidity()
```

```
print("Humidity: %s %%rH" % humidity)
# alternatives
print(sense.humidity)
```

get_temperature

```
from sense_hat import SenseHat
sense = SenseHat()
temp = sense.get_temperature()
print("Temperature: %s C" % temp)
# alternatives
print(sense.temp)
print(sense.temperature)
```

get_pressure

```
from sense_hat import SenseHat
sense = SenseHat()
pressure = sense.get_pressure()
print("Pressure: %s Millibars" % pressure)
# alternatives
print(sense.pressure)
```

compass

```
from sense_hat import SenseHat
sense = SenseHat()
north = sense.get_compass()
print("North: %s" % north)
```

```
# alternatives
print(sense.compass)
```

gyroscope

```
from sense_hat import SenseHat
sense = SenseHat()
gyro_only = sense.get_gyroscope()
print("p: {pitch}, r: {roll}, y: {yaw}".format(**gyro_only))
# alternatives
print(sense.gyro)
print(sense.gyroscope)
```

accelerometer

```
from sense_hat import SenseHat
sense = SenseHat()
accel_only = sense.get_accelerometer()
print("p: {pitch}, r: {roll}, y: {yaw}".format(**accel_only))
# alternatives
print(sense.accel)
print(sense.accelerometer)
```

joystick

```
from sense_hat import SenseHat
sense = SenseHat()
while True:
    for event in sense.stick.get_events():
        print("The joystick was {} {}".format(event.action, event.direction))
```

RFID Cards with a Raspberry Pi

Nearly all physical security (Circuit Basics, n.d.) systems must include radio frequency identification (RFID) equipment. Access to certain areas of a building or room is controlled using RFID cards and card readers. In this instance, an RFID card reader mounted on a wall wirelessly detects a unique identification number stored on the RFID card. The door is opened and the cardholder is permitted entry if the RFID card's identification number matches one on a list of approved cards that has been saved.

Follow the wiring diagram below to connect the RFID card reader to the Raspberry Pi:

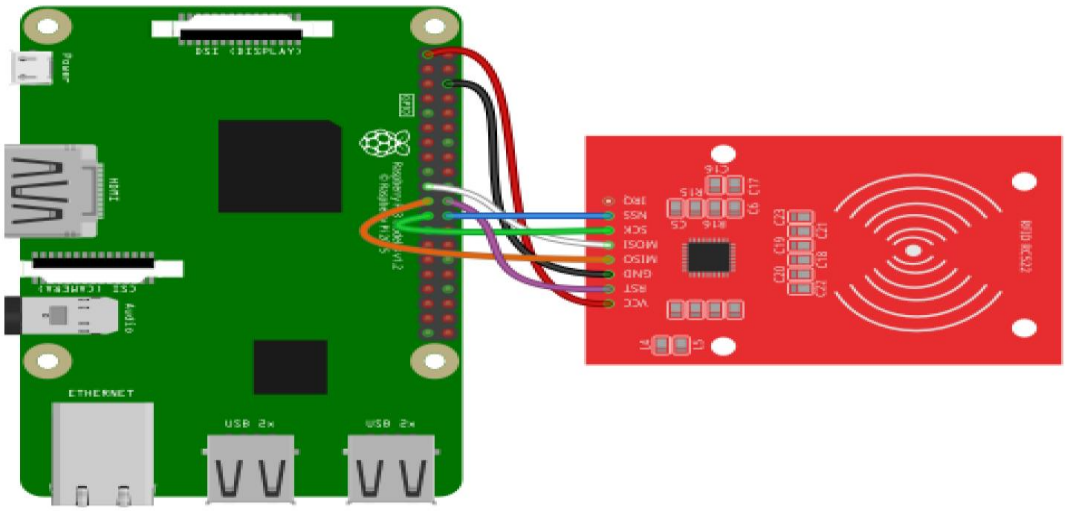


Figure 47 RFID Cards with a Raspberry Pi

Before we get started programming, make sure the SPI interface is enabled on your Raspberry Pi. You also need to install pip3 with this command:

```
sudo apt-get -y install python3-pip
```

Then, install the MFRC522 Python library with the following command:

```
sudo pip3 install mfrc522
```

Then, create a Python file rfidReader.py and paste the following code.

```
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

rfid = SimpleMFRC522()

while True:
    id, text = rfid.read()
    print(id)
    print(text)
```

Run the Python code above by entering the command

```
sudo python3 rfidReader.py
```

Figure

References

(n.d.). Retrieved from Tinkercad: https://en.wikipedia.org/wiki/Tinkercad

14 Best Single Board Computers in 2022. (n.d.). Retrieved from https://www.seeedstudio.com/blog/2020/10/20/raspberry-pi-alternatives-17-best-single-board-computers-in-2020/

Analog-to-digital converter (ADC). (n.d.). Retrieved from https://en.wikipedia.org/wiki/Analog-to-digital_converter

Arduino Tutorial – The Industrial and Professional Way. (n.d.). Retrieved from https://www.arnabkumardas.com/arduino-tutorial/avr-architecture/

ARM. (n.d.). Retrieved from ARM DEVELOPER: https://developer.arm.com/documentation

ARM architecture family. (n.d.). Retrieved from https://en.wikipedia.org/wiki/ARM_architecture_family

Circuit Basics. (n.d.). Retrieved from https://www.circuitbasics.com/what-is-an-rfid-reader-writer/

Digital-to-analog converter (DAC). (n.d.). Retrieved from https://en.wikipedia.org/wiki/Digital-to-analog_converter

Electric motor. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Electric_motor

Element14. (n.d.). *Element14*. Retrieved from https://au.element14.com/c/semiconductors-ics/microcontrollers-mcu

ESP. (n.d.). *ESPHome*. Retrieved from https://www.esphome.io/#

Esp8266. (n.d.). Retrieved from https://www.espressif.com/en/products/socs/ esp8266

ESP8266. (n.d.). Retrieved from https://en.wikipedia.org/wiki/ESP8266

ESP8266 Arduino Core. (n.d.). Retrieved from https://arduino-esp8266.readthedocs.io/en/2.4.1/

ESP8266 Basic Help 3.0. (n.d.). Retrieved from https://docs.google.com/document/d/1EiYugfu12X2_pmfmu2O19CcLX0ALgLM4r2YxKYyJon8/pub

Esp8266Basic. (n.d.). Retrieved from https://www.esp8266basic.com/

GpioZero. (n.d.). Retrieved from https://gpiozero.readthedocs.io/en/stable/index.html

Harvard architecture. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Harvard_architecture

IEEE. (2015). *Towards a definition of the Internet of Things (IoT)*. IEEE. Retrieved from https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf

Inter-Integrated Circuit. (n.d.). Retrieved from https://en.wikipedia.org/wiki/I%C2%B2C

Introduction to ATmega328. (n.d.). Retrieved from https://www.theengineeringprojects.com/2017/08/introduction-to-atmega328.html

Makecode Microbit Docs. (n.d.). Retrieved from https://makecode.microbit.org/docs

Margolis, M. (n.d.). *Arduino Cookbook*. O'REILLY. Retrieved from https://books.google.gr/books?id=nxxKNCYXRIwC&lpg=PA97&ots=dKXdUkzMsl&dq=arduino%20wikipedia&hl=el&pg=PP1#v=onepage&q&f=false

Micro Bit. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Micro_Bit

Newmarch, J. (2017). *IoT - a techie's viewpoint.* Retrieved from https://jan.newmarch.name/IoT/index.html
Nodejs Raspberrypi. (n.d.). Retrieved from https://www.w3schools.com/nodejs/nodejs_raspberrypi.asp
Pulse-width modulation. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Pulse-width_modulation
Pyhton sense-hat. (n.d.). Retrieved from https://pythonhosted.org/sense-hat/api/
Raspberry Pi. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Raspberry_Pi
Raspberry Pi. (n.d.). Retrieved from https://www.raspberrypi.com/
Raspberry Pi Foundation. (n.d.). Retrieved from https://www.raspberrypi.org/
Raspberry Pi Sense HAT. (n.d.). Retrieved from http://wiki.sunfounder.cc/index.php?title=Raspberry_Pi_Sense_HAT_-_For_the_Pi_3_/_2_/_B%2B_/_A%2B
Sense HAT. (n.d.). Retrieved from https://www.waveshare.com/wiki/Sense_HAT_(B)
Serial Peripheral Interface (SPI). (n.d.). Retrieved from https://en.wikipedia.org/wiki/Serial_Peripheral_Interface
The Best Single-Board Computers of 2022. (n.d.). Retrieved from https://all3dp.com/1/single-board-computer-raspberry-pi-alternative/
Universal asynchronous receiver-transmitter (UART). (n.d.). Retrieved from https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
Von Neumann architecture. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Von_Neumann_architecture
Wikipedia. (n.d.). *List of common microcontrollers*. Retrieved from https://en.wikipedia.org/wiki/List_of_common_microcontrollers#Microchip_Technology