LamaPLC: Arduino basic

Arduino is an open-source hardware and software company, project, and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices. Its hardware products are licensed under a CC BY-SA license. In contrast, the software is licensed under the GNU Lesser General Public License (LGPL) or the GNU General Public License (GPL), allowing anyone to manufacture Arduino boards and distribute the software. Arduino boards are available commercially from the official website or through authorized distributors.

Arduino board designs use a variety of microprocessors and controllers.
The boards are equipped with sets of digital and analog input/output (I/O) pins that can be interfaced with various expansion boards ('shields'), breadboards (for prototyping), and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs. Microcontrollers can be programmed using the C and C++ programming languages, utilizing a standard API known as the Arduino Programming Language, which is inspired by the Processing language and used in conjunction with a modified version of the Processing IDE.

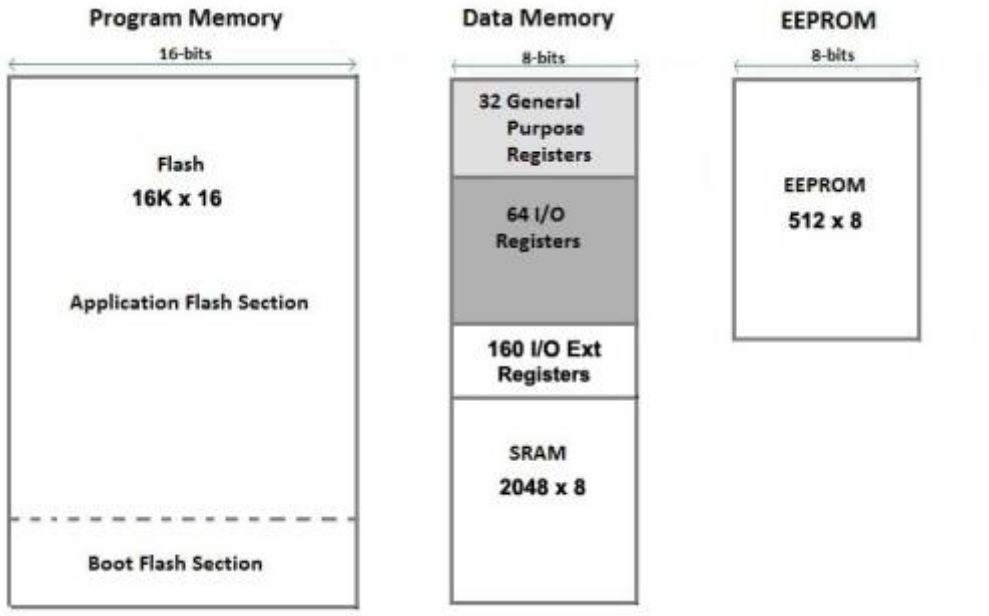
A comparison of the technical data of some Arduino boards

Type	chip	CPU speed	Power supply	Operation voltage	Digital number of pins (pwm pins)	Analog number of pins	Flash memory	SRAM memory	EEPROM size	Extras
Arduino Uno R3	ATmega328P	16 MHz	6..20V	5V	14 (6)	6	32 KB	2 KB	1 KB	-
UNO WiFi Rev2	ATmega4809	16 MHz	6..20 V	5V	14 (6)	6	48 KB	6,144 Bytes	256 Bytes	WIFI; Bluetooth LE
UNO R4 WiFi (ESP32-S3)	Renesas RA4M1 (Arm Cortex-M4)	48 MHz	6..24 V	5V	14 (6)	6	256 kB	32 kB	8 kB	WIFI; Bluetooth; UART, I²C, SPI, CAN; USB-C port; DAC (12 Bit); OP AMP; LED matrix; HID support
UNO R4 Minima	Renesas RA4M1 (Arm Cortex-M4)	48 MHz	6..24 V	5V	14 (6)	6	256 kB	32 kB	8 kB	UART, I²C, SPI, CAN; USB-C port; DAC (12 Bit); OP AMP; SWD
Arduino Mega2560	ATmega2560	16 MHz	6..20V	5V	54 (15)	16	256 KB	8 KB	4 KB	-

Type	chip	CPU speed	Power supply	Operation voltage	Digital number of pins (pwm pins)	Analog number of pins	Flash memory	SRAM memory	EEPROM size	Extras
Arduino Giga R1 WiFi	STM32H747XI Dual Core (Cortex® M7-Kern)	480 MHz	6..24 V	3,3 V	76 (13)	12	2 MB	1 MB	-	Wi-Fi® 802.11b/g/n 65 Mbps Bluetooth® Low Energy Micro UFL connector 4x UART 3x I²C 2x SPI 1x CAN
Arduino Due	AT91SAM3X8E SAM3X8E ARM Cortex-M3	84 MHz	9 V DC	3,3 V	54 (12)	12	512 KB	96 KB	-	USB USB OTG 4x UART 1x CAN
Arduino Nano	3.x: ATmega328, 2.x: ATmega168	16 MHz	7..12V	5V	14 (6)	8	32 KB 16 KB	2 KB 1 KB	1 KB 512 Byte	Mini-B USB connector
Arduino Micro	ATmega32u4	16 MHz	7..12V	5V	20 (7)	12	32 KB	2.5 KB	1 KB	5 hardware interrupts, full speed USB
Keyes Pro Micro	ATmega32u4	8 MHz	7..9V	3.3V	12 (5)	4	32 KB	2.5 KB	1 KB	5 hardware interrupts, full speed USB
Pro Micro 5V	ATmega32u4	16 MHz	7..9V	5V	12 (5)	4	32 KB	2.5 KB	1 KB	-
Arduino Mini	ATmega328P	3,3V: 8MHz 5V: 16 MHz	6..20V	3,3V 5V	14 (6)	8	32 KB	2 KB	1 KB	2 hardware interrupts, no USB port
STM32F103	STM32 32-bit Arm Cortex	72 MHz	5V	2.7V..3.6V	16	16	64 KB	20 KB	-	USART / I2C / SPI / USB / CAN / DMA
STM32F401	STM32 32-bit Arm Cortex	25 MHz	5V	2.7V..3.6V	16	16	256 KB	64 KB	-	USART / I2C / SPI / USB / CAN / DMA
Wemos D1	ESP8266EX-		9..12V	3.3V	11 (11)	1	4 KB	2 KB	1 KB	integrated ESP8266
Duemilanove	ATmega168 ATmega328-		6..20V	5V	14 (6)	6	16KB 32KB	1KB	512 byte 1KB	-
Digispark	-	-	-	5V	14	10	16KB	2KB	1KB	-
RoboRED	-	-	-	5V/3.3V	14	6	32KB	2KB	1KB	-
ATmega1280	ATmega1280	-	5V	54	16	128KB	8KB	4KB	-	
Arduino Leonardo	ATmega32u4	16 MHz	5V	20 (7)	12	32KB	2.5KB	1K	-	
Arduino Due	-	-	3.3V	54	12	512KB	96K	-	-	
ChipKIT Max32	Diligent	-	3.3V	83	16	512KB	128KB	-	-	

Type	chip	CPU speed	Power supply	Operation voltage	Digital number of pins (pwm pins)	Analog number of pins	Flash memory	SRAM memory	EEPROM size	Extras
Arduino MKR ZERO	SAMD21 Cortex®-M0+ 32bit low power ARM MCU	48 MHz	5V	3.3V	22	7 input + 1 output	256 KB	32 KB	no	I²S bus & SD for sound, music & digital audio data
Arduino MKR FOX 1200	SAMD21 Cortex®-M0+ 32bit low power ARM MCU	48 MHz	5V	3.3V	8	7 input + 1 output	256 KB	32 KB	no	Wifi on board: 868 MHz Sigfox: Smart RF ATA8520

Memory types of Arduino boards



Flash memory

This is considered the main memory of Arduinos; it stores the downloaded program and preserves its content even after being switched off. In other words, it is sufficient to download the program here only once, as it restarts itself every time it is switched on again. During programming, we cannot rely on the entire memory, as the download program (bootloader) and various communications protocols also occupy parts of it.

In addition, the downloaded libraries can also occupy a significant amount of space, so even with a 32 KB memory by default, we can expect only 24-30 KB (without libraries).

Additionally, the flash memory cannot be rewritten indefinitely; its maximum number of write cycles is limited to 100,000. This is enough to store a program rewritten 10 times a day for about 27 years without any problems.

SRAM

static random-access memory

Simply put, SRAM stores the internal variables defined in the program. SRAM, in contrast to flash memory, does not retain its contents in a power-free state. Therefore, after each power-on, the program redefines the variables, and they are transferred to the SRAM with their default values determined there.

EEPROM

Electrically Erasable Programmable Read-Only Memory

EEPROM is the non-volatile variable memory of boards. This, similarly to Flash, preserves its content even when it is turned off; however, like Flash, it is only certified for 100,000 write cycles. Therefore, it is not particularly suitable for cyclic data writing, for example. Additionally, it's slightly slower to handle than regular SRAM.
Due to these technical characteristics, the EEPROM can be used for the following functions:

- storage of configuration(s).
- saving initial settings
- save counters, values, and collected values (e.g., operating hours counter) independent of restarts

Different Arduino boards have different EEPROM sizes:

- ATmega168: 512 Bytes
- ATmega8: 512 Bytes
- ATmega328P: 1024 Byte
- ATmega32U4: 1024 Bytes
- ATmega1280: 4096 Bytes
- ATmega2560: 4096 Bytes

When writing to the EEPROM memory, Arduino specifies two parameters: the address and value. Each byte can hold a value between 0 .. 255 (byte-wise):

```
EEPROM.write (2,244); // write value "244" to byte address 2
val = EEPROM.read (10); // read EEprom arrdess 10
```

Arduino IDE

The Arduino IDE is a development system written in Java, with which we can write programs for the Arduino, compile and debug them, and then download them to the cards. The download is most often performed via the USB port, which is located on almost every Arduino board. However, it is also possible to download via ISP or OTA, if the given board supports these options.
Arduinos have a pre-flashed bootloader, which allows downloading of code without external hardware, simply via the *STK500* protocol.

If necessary, the above bootloader can be bypassed using the ICSP (In-Circuit Serial Programming).

Of course, we have many options for programming Arduinos outside of the rather inexpensive (but efficient and free) Arduino IDE. For example, Visual Studio can also be used for this purpose after installing an Arduino plugin.

Arduino OTA

Over the Air
The Arduino also gives you the option to download programs via wifi or even an Ethernet shield. ESP modules (ESP8266 or ESP32) offer a perfect opportunity to exploit this function, which can be directly integrated with Arduino boards. Thus, most of the implementation of downloading via Wifi is carried out by these modules.

Arduino API

Application Programming Interface
The functionality of Arduino can be extended through the library system. The use of libraries requires memory, but with careful selection, the memory load can be well optimized, and a significant portion of the programming can be saved by utilizing them. There are standard libraries, such as Wire, AVR_C, and String, but a huge number of well-developed libraries are also available for download from GitHub. When describing the sensors, I almost always refer to the GitHub libraries.

Arduino Bootloader

A special program preloaded on Arduino Boards and compatible with the Arduino IDE, which allows programs to be uploaded to boards without special tools, usually via USB.

Sketch

The programs that can be run on Arduino are called sketches. Sketches can be saved in "**.ino**" files or loaded from there. There are many (basic) example programs in the Arduino IDE under File / Examples. Still, almost every installed directory also contains example programs (they can also be opened in File / Examples after installation, they are at the bottom of the list).
Many sketches communicate with the computer via a serial connection. The Arduino IDE has a built-in serial monitor or terminal to help display this data. You can also send data to the board using the Monitor. You can find the serial monitor under "Tools" → "Serial monitor". Starting this usually reboots the Arduino board. Make sure you set the speed, or baud rate, to the correct value. If you don't do that, you'll either see garbage or nothing here. The typical data transfer rates are: 9600 or 115.200 baud.

Arduino and USB

Arduino boards can be programmed from your PC via the USB port (see above, Bootloader), and during program execution, serial information can also be sent to your PC through this port. Most boards include the classic FTDI or AtMega16U2 USB communication IC for the connection, but it may happen that the so-called (Chinese) CH340..CH341 ICs are built in. Each of these requires a different driver; this must be taken care of when using a distinct board for the first time, because it is possible that our PC does not see the Arduino.

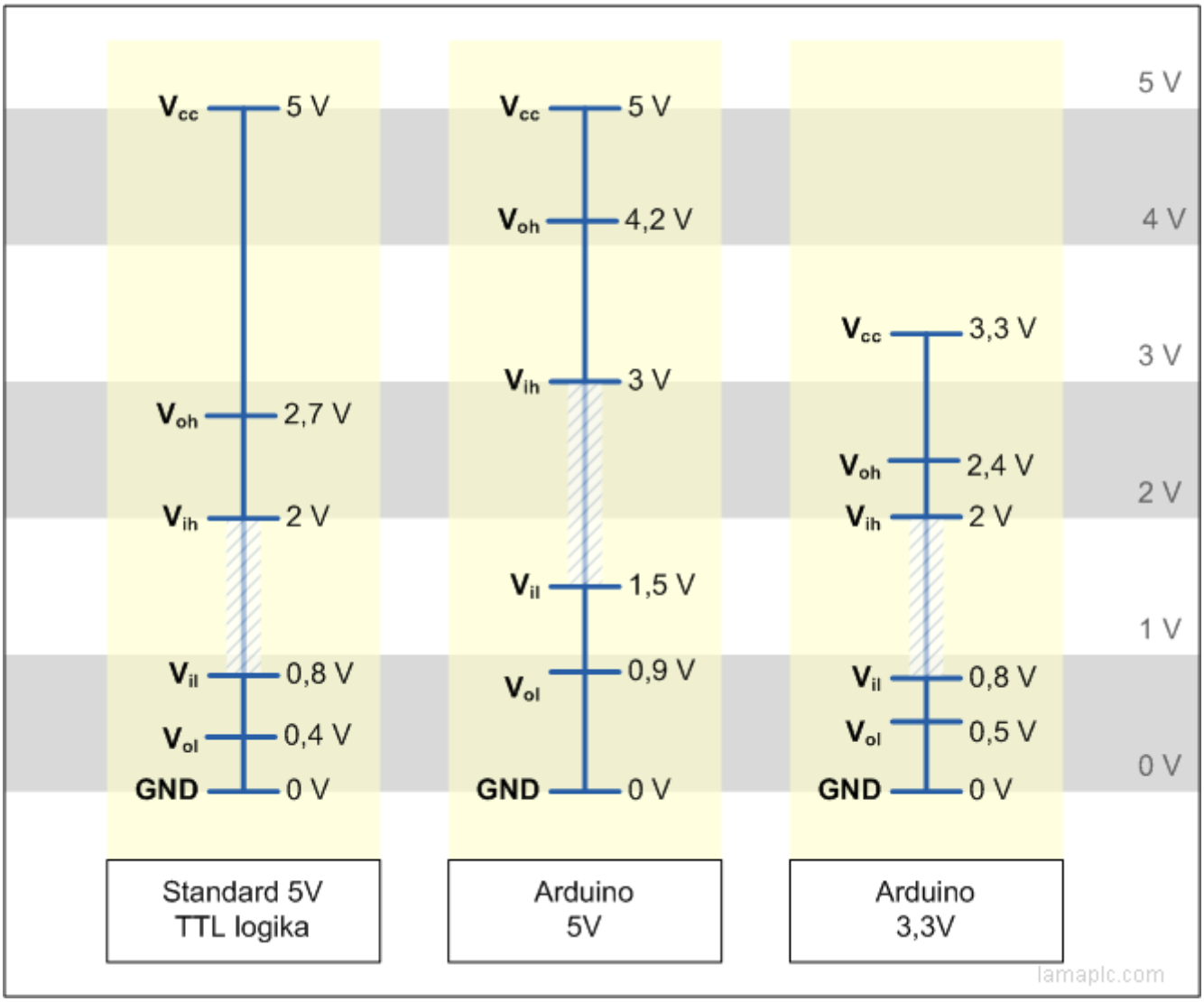
Most Arduino USB ports have an overcurrent protection (resettable polyfuse), which automatically releases in the event of a current consumption exceeding 500 mA or a short circuit.

Arduino USB UART converter types:

- FTDI: Converter type of early Arduinos, proved to be too expensive. It is still used as an external converter (connected to the ISP)
- Atmega8U2: The first serial Uno-k converter, up to version R2
- AtMega16U2: Currently, most (official) Arduino board UART converters
- CH340/CH340G/CH341: typical converters of Chinese-made Arduino clones; downloading the drivers is somewhat difficult

TTL logic levels

The (digital) electronic equipment that uses logical high (HIGH) and low (LOW) levels is mostly the so-called. The voltage levels associated with high/low signals are defined based on TTL (Transistor-Transistor Logic). The Arduino logic levels differ slightly from these levels:



- **VOH:** Minimum output supply voltage level. Above that, the TTL device provides a HIGH signal
- **VIH:** Minimum input voltage level for the HIGH signal
- **VIL:** Maximum input voltage level for low (LOW) signal
- **VOL:** Maximum output voltage for a low (LOW) signal

The evaluation of the input (HIGH or LOW) becomes uncertain on the covered parts. While the program logic levels can be characterized by the constants true (true, 1) and false (false, 0), the state of the pins can be characterized by the constants HIGH (high, 1) and LOW (low, 0).

The HIGH state occurs when a signal is read at a level above 3V on the 5V inputs, and above 2.0V on the 3.3V inputs. The LOW states are under 1.5V for five boards, approximately for 3.3V boards. They appear below 1.0V.

Intermediate voltage levels (1.5V - 3.0V for 5V boards, 1.0V - 2.0V for 3.3V boards) should be avoided because the input state becomes uncertain.

If the input pull-up resistor was previously activated on the given pin, there is a good chance that we will never read a HIGH state from it, see: digitalRead().

In the case of outputs, after releasing the HIGH state, the pin takes the maximum voltage level, i.e., 3.3V for 3.3V boards, 5V for 5V boards.

If the internal pull-up resistor was previously activated, the output voltage will also be less than 5V (approx. 3.3V) in case of a HIGH signal, see: digitalWrite().

Load capacity of pins

Arduinos (due to their simple construction) cannot be loaded, and this must be taken into account for all controls that require more current (such as solenoid valves and relays). In the case of a more serious overload, the board dies, and in the case of a smaller (and short-term) overload, it switches off. In the case of poorly dimensioned relay control, the Arduino may turn the relays on and off (the relays click accordingly). In such a case, it is worth considering the load on the board.

Main load capacity limits (of course, these may vary by type):

- Pin load capacity for UNO (5V): **20 mA**
- Pin load capacity for the Mega board (5V): **40 mA**
- Pin load capacity (for 3.3V): **10 mA**
- Pin load capacity for Giga R1 WiFi (for 3.3V): **8 mA**
- Maximum load capacity (UNO) of all pins (Vcc, GND): **200 mA**
- Maximum load capacity (Mega) of all pins (Vcc, GND): **400..800 mA**

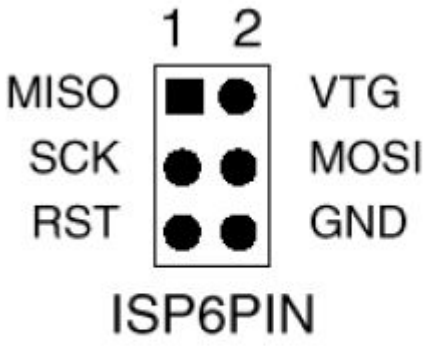
Use of pins

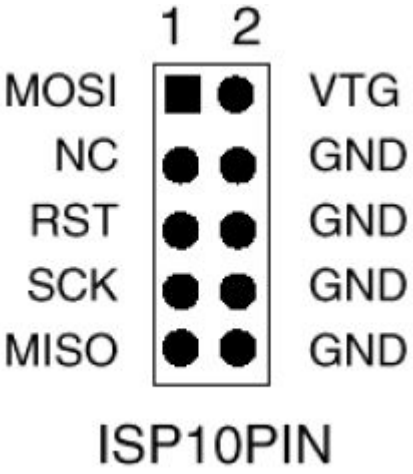
There are a few tips and tricks for using the Arduino pins, which are also the causes of many inexplicable problems:
Pin0 and Pin1 are connected to serial communication, so if we want to receive information from the board and use the Serial.print instruction to send information to the serial monitor, these pins cannot be used as digital input/output at this time. We should not even define these as digital channels; if this is not possible because we have run out of pins, then we should implement a function here that will allow us to say goodbye to serial use for the time being.

The analog pins can be used as digital ports without further ado, in which case you refer to them as pins 14..19, for example, in the case of the Uno, as shown in the figure below. It is worthwhile to designate the ports to be left free during the design phase. Since I primarily use I²C for communication, I utilize A4 and A5 for this purpose in the case of the Uno. In the same way, especially for testing, it is worth considering serial communication and the corresponding ports, i.e., the 0 (Rx) and 1 (Tx) pins in the case of the Uno.

Arduino ISP/ICSP

The ISP (In-system programming) port, which is typically integrated on Arduino (and compatible) boards, provides the possibility to download a program to the board by bypassing USB/bootload. It also provides communication options for SPI communication, for example, between cards. Pin assignments of possible ports:





Additionally, these pins are connected in parallel with the corresponding pins of the IO ports (see below); therefore, they are grouped here for download convenience. It is also possible to use an Arduino board to upload programs to other Arduinos via the ISP (In-System Programming) method.

Arduino ISP port	**name**	**short description**
MISO	Master In Slave Out	input as master, output as slave
VTG	5V	input 5V+

Arduino ISP port	name	short description
SCK	Serial Clock	serial clock signal, provided by the master for SPI
MOSI	Master Out Slave In	output as master, input as slave
RST	Reset	reset
GND	0V	0V

Arduino ISR

The ISR (*interrupt service routine*) function was developed to monitor inputs with rapidly changing states. These interruptions are typically suitable for counting or monitoring fast pulses (Hall pulses, such as those from flowmeters or current signal pulses) and are independent of the program run cycle.
From the software side, these signals must be handled with the *attachInterrupt()* function, and these fast signals can only be read on the ports designated for this purpose:

Board	Digital ports assigned to interrupt
Arduino Uno, Arduino Nano, Mini, other 328-based	2, 3
Arduino Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Wemos D1	all digital outputs (except D0)

Arduino PWM

PWM (pulse-width modulation) is a form of signal generation that can be used to create analog output signals from digital inputs, i.e., it serves as a digital-to-analog converter function. PWM outputs can be controlled with the *analogWrite()* function on Arduinos. The PWM function is available on the designated PWM digital outputs, marked with ~.

Board	designated PWM outputs
For most ATmega168 or ATmega328 boards Arduino Uno, Arduino Nano*	3, 5 , 6, 9, 10, 11 pin, pin 5-6: 980 Hz, the rest: 490 Hz
Arduino Mega	2..13. and 44..46. pin
Wemos D1	all digital outputs (except D0)
Arduino Micro	0, 1, 2, 3, 7
for older boards	pin 9, 10 and 11

*: for ATmega168 or ATmega328 boards:

- timer0: pin 5 and 6, 8-bit PWM
- timer1: pin 9 and 10, 16-bit PWM
- timer2: pin 11 and 3, 8-bit PWM

For servos that require 16-bit resolution, only pins 9 and 10 will work!

Only 8-bit PWM can be implemented with the Arduino Micro.

Arduino AREF pin

Via the AREF pin, the *analogReference()* function can be used to set the external (type: EXTERNAL) reference voltage (i.e., the maximum value of the input range) used for the analog input.

Arduino communication

Comparison of the most frequently used direct* communication solutions for Arduino systems:

*: Ethernet, Wifi, CAN, etc., are not direct communication in this sense, because they require some converter, and the converters are only available with the communication forms below.

communication solution	**Serial (UART)**	**I²C**	**1-Wire**	**SPI**
maximum number of partners / slaves	P2P communication, 1 master + 1 slave	theoretically 128 stations	no theoretical limit, practically approx. 150	Defined by the number of SS pins. without SS 1 master and 1 slave
bridgeable distance (≈)	9600 baud: 2-3 m 300 baud: >100 m (twisted pair)	100 Kbaud: 1 m 10 Kbaud: 6-8 m	with 20 sensors max. 100 m	max. 5 m
number of pins used	2: Tx, Rx	2: SDA, SCL	1: DATA	3 or 4: SCK, MOSI, MISO, (SS)
restrictions	Using pins 0 and 1 for serial download and serial monitor does not work	Blocks the following pins: SDA-A4, SCL-A5	-	Blocks 4 pins

Arduino serial communication

Arduino hardware serial communication

The default communication of Arduino boards is serial (UART), which is carried out via USB with the programming device or the communication partner PC. The TX and RX pins of the boards are also connected in parallel with the USB port, so if you use serial communication, you cannot use these pins as digital ports.

This channel can, of course, also be used for other UART communication, for example, RS-232, but in this case, you have to pay attention to the voltage levels, which in the case of RS-232 are typically ± 12V. All Arduino boards have at least one serial (UART) port; their pin assignment:

board	**serial communication pins**
Arduino Uno, Nano	Rx←0, Tx→1
Arduino Mega, Due	Serial1: pin 19 (RX), pin 18 (TX), Serial2: pin 17 (RX), pin 16 (TX), Serial3: pin 15 (RX), pin 14 (TX)

Arduino software serial communication

Software serial communication is made possible by using the SoftwareSerial library. While hardware serial communication utilizes hardware components for the UART and operates while performing other tasks (as long as there is room in the 64-byte buffer), software serial communication is handled by software. This is obviously at the expense of the execution of the other software, but it enables multiple serial communications at the same time, at a speed of up to 115.200.

Limitations of software serial communication

- If several software serial ports are to be used, only one can receive data at the same time
- Not all pins of Mega and Mega 2560 are suitable for software serial communication, the following pins can be used for RX: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8 (62), A9 (63), A10 (64), A11 (65), A12 (66), A13 (67), A14 (68), A15 (69).
- For Leonardo, only the following pins can be used for RX: 8, 9, 10, 11, 14 (MISO), 15 (SCK), 16 (MOSI).
- On the Arduino Genuino 101, the maximum RX speed can be 57,600 bps, and pin 13 cannot be used as RX.

Data loss during software serial communication

When you send or receive long messages using software serial communication, they may arrive with some characters missing. The reason for this is not necessarily evident in the code. The SoftwareSerial receive buffer may get full and then discard the characters.

The easiest solution to this is to increase the software's serial buffer from the default 64 bytes to 256 bytes (or just larger than 64 bytes):

On the PC, open the following file: C:\Program Files (x86) → Arduino → hardware → Arduino → avr → libraries → SoftwareSerial → SoftwareSerial.h and change the following line:

```
// RX buffer size
#define _SS_MAX_RX_BUFF 64 // change 64 to max 256, for example
```

Arduino I²C

One of the most commonly used communications on Arduino boards is I²C. The default ports for communication on the various boards are as follows:

Board	SDA (data port)	SCL (clock port)	note
Arduino Uno R3	A4	A5	
Arduino Uno R4 (Wifi/Minima)	SDA	SCL	dedicated ports
Arduino Mega	20	21	
Arduino Nano	4	5	
Wemos D1	A4 (SDA)	A5 (SCL)	all digital outputs (except D0) can be applied to I²C

Board	SDA (data port)	SCL (clock port)	note
NodeMCU (ESP8266)	D2(GPIO4)	D1(GPIO5)	D0 cannot be used for I²C

Arduino SPI

One of the commonly used communications on Arduino boards is SPI. On Arduinos, the ISP connection module also uses SPI, which is relatively easy to project (unfortunately, SS did not have a place here):

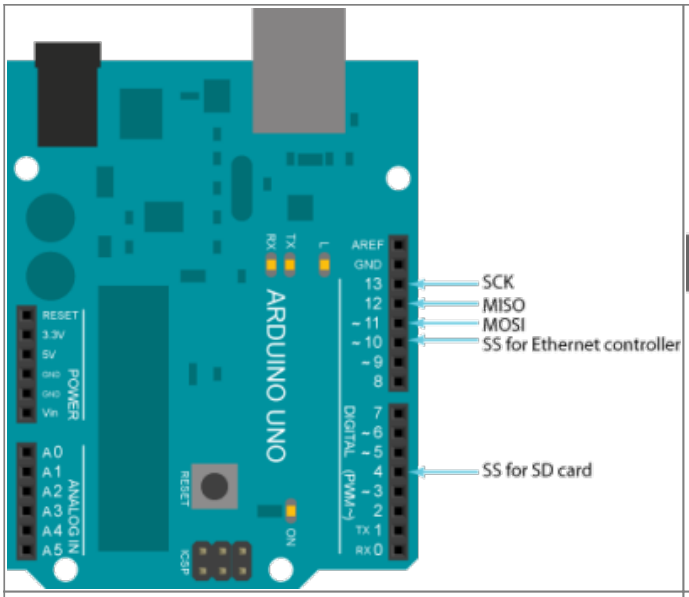
Board	SPI SS (select)	SPI MOSI	SPI MISO	SPI SCK
Arduino UNO	10	11	12	13
Arduino Mega	53	50	51	52
Arduino Nano	D10	D11	D12	D13
NodeMCU (ESP8266)	D8(GPIO15)	D7(GPIO13)	D6(GPIO12)	D5(GPIO14)

Arduino 1-wire

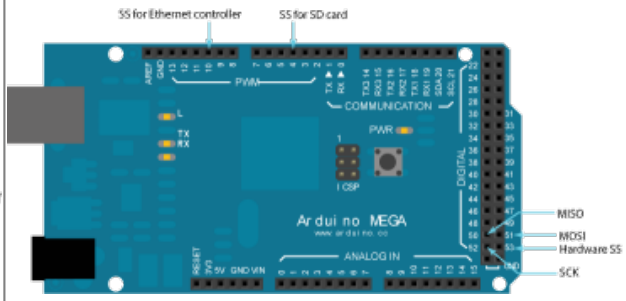
The 1-wire bus describes a Dallas Semiconductor Corp. serial interface that handles a data line (DQ) that is used as both a power supply and a transmit and receive line. The term '1-wire' is misleading because communication also requires a ground (GND). In fact, even with 1-wire, we always use at least two physical wires (GND, DQ).

Arduino Ethernet

Arduino is, of course, suitable for communication via Ethernet and access to Internet functions. The easiest way to access Ethernet is to use an Ethernet module or extension. Modules and add-ons typically communicate with Arduino boards via SPI:



Arduino Uno Ethernet SPI Pinout



Arduino Mega Ethernet SPI Pinout

A micro-SD card reader is also integrated on most Ethernet boards. The use of this is optional, but it occupies an extra pin on the board.

WIZnet W5x00 series ICs typically handle the Ethernet communication for modules/extensions. A comparison of their main features:

Feature	**W5100**	**W5300**	**W5500**
Interface Mode	direct, indirect, SPI	direct, indirect	SPI
Number of sockets	4	4	8
Speed (max, MBPS)	25	25	15
Data bus	Only 8 Bit, DATA[7:0]	16/8 Bit, DATA[15:8]/DATA[7:0]	16/8 Bit
Title bus	15 PINs, ADDR[14:0]	10PINs, ADDR[9:0]	10PINs, ADDR[9:0]
Memory size	(Fixed) 16KBytes TX : 8KBytes, RX : 8Kbytes	(Configurable) 128KBytes TX : 0~128KBytes, RX : 0~128KBytes	(Fixed) 32KBytes TX : 16KBytes, RX : 16Kbytes

arduino, basic