Arduino

**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 and interactive objects that can sense and control objects in the physical and digital world. Its products are licensed under the GNU Lesser General Public License (LGPL) or the GNU General Public License (GPL), permitting the manufacture of Arduino boards and software distribution by anyone. Arduino boards are available commercially in preassembled form or as do-it-yourself (DIY) kits.

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 may be interfaced to various expansion boards or breadboards (*shields*) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers. The microcontrollers are typically programmed using a dialect of features from the programming languages C and C++. In addition to using traditional compiler toolchains, the Arduino project provides an integrated development environment(IDE) based on the Processing language project.

**Hardware:**

Arduino is open-source hardware. The hardware reference designs are distributed under a Creative Commons Attribution Share-Alike 2.5 license and are available on the Arduino website. Layout and production files for some versions of the hardware are also available.

Although the hardware and software designs are freely available under copyleft licenses, the developers have requested the name Arduino to be exclusive to the official product and not be used for derived works without permission. The official policy document on use of the Arduino name emphasizes that the project is open to incorporating work by others into the official product. Several Arduino-compatible products commercially released have avoided the project name by using various names ending in -duino.

An early Arduino board with an RS-232 serial interface (upper left) and an Atmel ATmega8 microcontroller chip (black, lower right); the 14 digital I/O pins are at the top, the 6 analog input pins at the lower right, and the power connector at the lower left.

Most Arduino boards consist of an Atmel 8-bit AVR microcontroller (ATmega8, ATmega168, ATmega328, ATmega1280, ATmega2560) with varying amounts of flash memory, pins, and features. The 32-bit Arduino Due, based on the Atmel SAM3X8E was introduced in 2012.The boards use single or double-row pins or female headers that facilitate connections for programming and incorporation into other circuits. These may connect with add-on modules termed shields. Multiple and possibly stacked shields may be individually addressable via an I²C serial bus. Most boards include a 5 V linear regulator and a 16 MHz crystal oscillator or ceramic resonator. Some designs, such as the LilyPad, run at 8 MHz and dispense with the on-board voltage regulator due to specific form-factor restrictions.

**Software:**

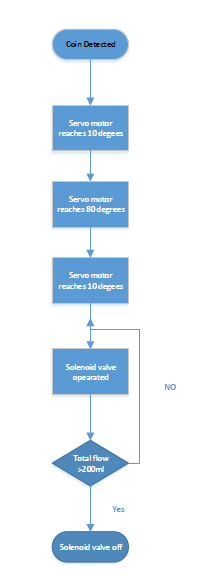
A program for Arduino hardware may be written in any programming language with compilers that produce binary machine code for the target processor. Atmel provides a development environment for their 8-bit AVR and 32-bit ARM Cortex-M based microcontrollers: AVR Studio (older) and Atmel Studio (newer).

**IDE**

The Arduino integrated development environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in the programming language Java. It originated from the IDE for the languages Processing and Wiring. It includes a code editor with features such as text cutting and pasting, searching and replacing text, automatic indenting, brace matching, and syntax highlighting, and provides simple one-click mechanisms to compile and upload programs to an Arduino board. It also contains a message area, a text console, a toolbar with buttons for common functions and a hierarchy of operation menus. The source code for the IDE is released under the GNU General Public License, version 2.

The Arduino IDE supports the languages C and C++ using special rules of code structuring. The Arduino IDE supplies a software library from the Wiring project, which provides many common input and output procedures. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub main() into an executable cyclic executive program with the GNU toolchain, also included with the IDE distribution. The Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware.

**Flow chart:**



**Code used for given application:**

*#include <Servo.h>*

*int wat=0;*

*byte statusLed = 13;*

*Servo myservo;*

*byte sensorInterrupt = 2; // 0 = digital pin 2*

*byte sensorPin = 2;*

*byte ser=3;*

*byte coin=4;*

*byte valve=5;*

*// The hall-effect flow sensor outputs approximately 4.5 pulses per second per*

*// litre/minute of flow.*

*float calibrationFactor = 4.5;*

*volatile byte pulseCount;*

*float flowRate;*

*unsigned int flowMilliLitres;*

*unsigned long totalMilliLitres;*

*totalMilliLitres = 0;*

*unsigned long oldTime;*

*void setup()*

*{*

*// Initialize a serial connection for reporting values to the host*

*Serial.begin(9600);*

*pinMode(ser,OUTPUT);*

*pinMode(valve,OUTPUT);*

*pinMode(coin,INPUT);*

*pinMode(sensorPin, INPUT);*

*digitalWrite(sensorPin, HIGH);*

*myservo.attach(ser);*

*pulseCount = 0;*

*flowRate = 0.0;*

*flowMilliLitres = 0;*

*oldTime = 0;*

*// The Hall-effect sensor is connected to pin 2 which uses interrupt 0.*

*// Configured to trigger on a FALLING state change (transition from HIGH*

*// state to LOW state)*

*attachInterrupt(sensorInterrupt, pulseCounter, FALLING);*

*attachInterrupt(coin,coinfun,RISING);*

*}*

*void mechanism()*

*{*

*myservo.write(10); // sets the servo position according to the scaled value*

*delay(2000);*

*myservo.write(80); // sets the servo position according to the scaled value*

*delay(2000);*

*}*

*void flow()*

*{*

*if((millis() - oldTime) > 1000) // Only process counters once per second*

*{*

*detachInterrupt(sensorInterrupt);*

*flowRate = ((1000.0 / (millis() - oldTime)) \* pulseCount) / calibrationFactor;*

*oldTime = millis();*

*flowMilliLitres = (flowRate / 60) \* 1000;*

*// Add the millilitres passed in this second to the cumulative total*

*totalMilliLitres += flowMilliLitres;*

*if(totalMilliLitres>=100)*

*wat=1;*

*else*

*wat=0;*

*unsigned int frac;*

*// Print the flow rate for this second in litres / minute*

*Serial.print("Flow rate: ");*

*Serial.print(int(flowRate)); // Print the integer part of the variable*

*Serial.print("L/min");*

*Serial.print("\t"); // Print tab space*

*// Print the cumulative total of litres flowed since starting*

*Serial.print("Output Liquid Quantity: ");*

*Serial.print(totalMilliLitres);*

*Serial.println("mL");*

*Serial.print("\t"); // Print tab space*

*Serial.print(totalMilliLitres/1000);*

*Serial.print("L");*

*// Reset the pulse counter so we can start incrementing again*

*pulseCount = 0;*

*// Enable the interrupt again now that we've finished sending output*

*attachInterrupt(sensorInterrupt, pulseCounter, FALLING);*

*}*

*}*

*void pulseCounter()*

*{*

*// Increment the pulse counter*

*pulseCount++;*

*}*

*void coinfun()*

*{*

*delay(2000);*

*mechanism();*

*delay(2000);*

*wat=0;totalMilliLitres = 0;*

*while(wat==0)*

*{*

*digitalWrite(valve,1);*

*flow();*

*}*

*digitalWrite(valve,0);*

*delay(1000);*

*}*

;