EPISODE 1

Introduction

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are:

I. Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.

II. You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).

III. Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.

IV. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.

V. Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Hardware

The core of the Arduino board is a microcontroller chip known as the ATmega328. The Atmega328 is in fact an 8 bit computer just as the one described in the first chapters of [1]: once switched on, its CPU loads a byte from a predefined memory location and interpret it as a statement. What follows is interpreted according to the content of such a byte. Contrary to the computers to which you are familiar with, the ATmega328 does not run any operating system: the usage of the resources is completely under the control of the programmer. You cannot rely on the operating system to prevent wrong memory usage, overflows, underflows and any other error. Moreover, that CPU can only run one task at the same time (you may remember that this is true for all the CPU's, however the operating system distribute the operation time to various tasks in such a way you have the impression that many programs run at the same time on your computer).

A fresh Arduino has an empty memory, hence the first byte loaded by the CPU correspond to the statement nop: no operation. Before using Arduino you must then load its memory with an executable program, i.e. a sequence of bits, the first of which is interpreted as a statement and executed. If the statement needs parameters to be executed, they are taken from the following bytes in the

memory. Once the execution of a statement has been completed, the CPU loads the successive byte in the memory and interprets it as well as a statement. If you switch off your Arduino, the memory is not lost. The sequence of bytes loaded into it are kept in a non-volatile memory, such that when you switch it on again, the program starts again from its beginning.

The timing for CPU operation is provided by a 16 MHz clock, while the power can be provided through a dedicated power jack as well as through its USB interface (see below). For operation, Arduino requires an input voltage between 7 and 12 V (voltage regulation is provided on board, so you just need an inexpensive power supply for that). On board, both a 5 V and a 3.3 V regulated output are provided to the user, too, from which you can drive a maximum current of 50 mA.

Arduino memory is of three types: a flash memory, where the program is stored, of **32 kB**; a static random access memory (SRAM) of **2 kB**, where the CPU stores and manipulates the variables used in the program; and an erasable read only memory (EEP- ROM) of **1 kB** where the programmer can store data that must survive the switch off (as the flash memory, where the program is stored). Compared to modern computers, who often bear as much as few GB, a total of 35 kB seems ridiculous, but in fact it is enough for most purposes. Because of the lack of an operating system, the memory usage is under your own responsibility: if you run out of memory or you try to access a non existing memory location, your program may behave strangely and it is very difficult to debug it. You must always keep the number of variables under control in your program.

The ATmega328 CPU is connected to 14 I/O digital pins (numbered from 0 to 13), 6 analog inputs and a USB port. A digital pin is an electrical connection that can have two logical states: 1 and 0, or true and false or, as in the Arduino jargon, LOW and HIGH. When put to LOW, the corresponding pin is at the ground potential: if you measure the voltage between the ground and the pin you get zero. If the pin is set to HIGH the voltage between the pin and the ground is 5V.

Programe Structure

Arduino programs can be divided in three main parts: **Structure**, **Values** (variables and constants), and **Functions**. Let us start with the structure. Software structure consist of two main functions:

I. **Setup() function**

II. **Loop() function**

```
void setup(){
        //pinmodes
        //start libraries
        //initialise values
}

void loop(){
        //loops, functions, processing
}
```

As soon as the program starts, the statements collected within the **setup()** block are executed: they are intended to initialise the content of the variables at start as well as to configure the Arduino ports behaviour. Once the execution of the setup() block is finished, Arduino starts executing statements in the **loop()** block. After execution those statements are executed again forever (hence the name loop).

Data Types

I. void
II. boolean
III. Char, string
IV. byte
V. int, short, long, float, double, word

Variables

I. Local variables
II. Formal parameters
III. Global variables

Operators

I. Arithmetic [=, +, –, *, /, %]
II. Comparison [==, !=, <, >, <=, >=]
III. Boolean [&&, ||, !]
IV. Bitwise [&, |, ~, ^, <<, >>]
V. Compound [++, —, +=, -=, *=, /=, |=, &=]

PROGRAM_1

```
#define LED 13

void setup(){
        pinMode(LED,OUTPUT);
}

void loop(){
        digitalWrite(LED, LOW);
        delay(300);
        digitalWrite(LED,HIGH);
        delay(300);
}
```

**PROGRAM_1.1 [Using Embedded C]

```
//Arduino Uno
int main() {
        DDRB |= 1 << PB5;
        while (1){
                PORTB ^= (1<<PB5);
                _delay_ms(300);
                 }
        return 0;
}
//Arduino Mega
int main() {
        DDRB |= 1 << PB7;
        while (1){
                PORTB ^= (1<<PORTB7);
                _delay_ms(500);
        }
        return 0;
}
//Using 16bit Timer
int main() {
  DDRB |= 1 << PB5;
  TCCR1B |= (1 << CS11) | (1 << CS10);
  TIMSK1 |= 1 << TOIE1;
  sei();
  while (1);
  return 0;
}

ISR(TIMER1_OVF_vect) {
    PORTB ^= (1 << PB5);
}
```

Control Statements

I. If-else [Program_2]
II. Switch
III. Conditional [condition ? True statement : False statement]

PROGRAM_2

```
#define LED 13

#define BUTTON 5

void setup(){
	pinMode(LED,OUTPUT);
	pinMode(BUTTON, INPUT_PULLUP);

	//OR
	//pinMode(BUTTON, INPUT);
	//digitalWrite(BUTTON, HIGH);
}

void loop(){
	bool B_STATE = digitalRead(BUTTON);

	if (B_STATE == true) {
		digitalWrite(LED, HIGH);
	} else {
		digitalWrite(LED, LOW);
	}
	delay(100);
}
```


Loops

I. Do-while, while
II. For [Programe_3]

PROGRAM_3

```
#define LED 11

#define RATE 10

void setup(){
	pinMode(LED,OUTPUT);
}

void loop(){
	for (int i=0; i<256; i+=RATE){
		analogWrite(LED, i);
		delay(25);
```

```
        }

        for (int i=255; i>=0; i-=RATE){
                analogWrite(LED, i);
                delay(25);
        }
}
```

Functions

[Programe_4]

PROGRAM_4

```
#define LED 11
#define BUTTON 10

bool debounce(void) {
  bool c_state, p_state;
  p_state = digitalRead(BUTTON);
  for (int c = 0; c < 10; c++) {
    delay(1);
    c_state = digitalRead(BUTTON);
    if (c_state != p_state) {
      c = 0;
      p_state = c_state;
    }
  }
  return c_state;
}

void led(bool state) {
  if (state == HIGH) {
    digitalWrite(LED, HIGH);
  } else {
    digitalWrite(LED, LOW);
  }
}

void setup() {
  pinMode(LED, OUTPUT);
  pinMode(BUTTON, INPUT_PULLUP);
}

void loop() {
  bool state = debounce();
  led(state);
}
```

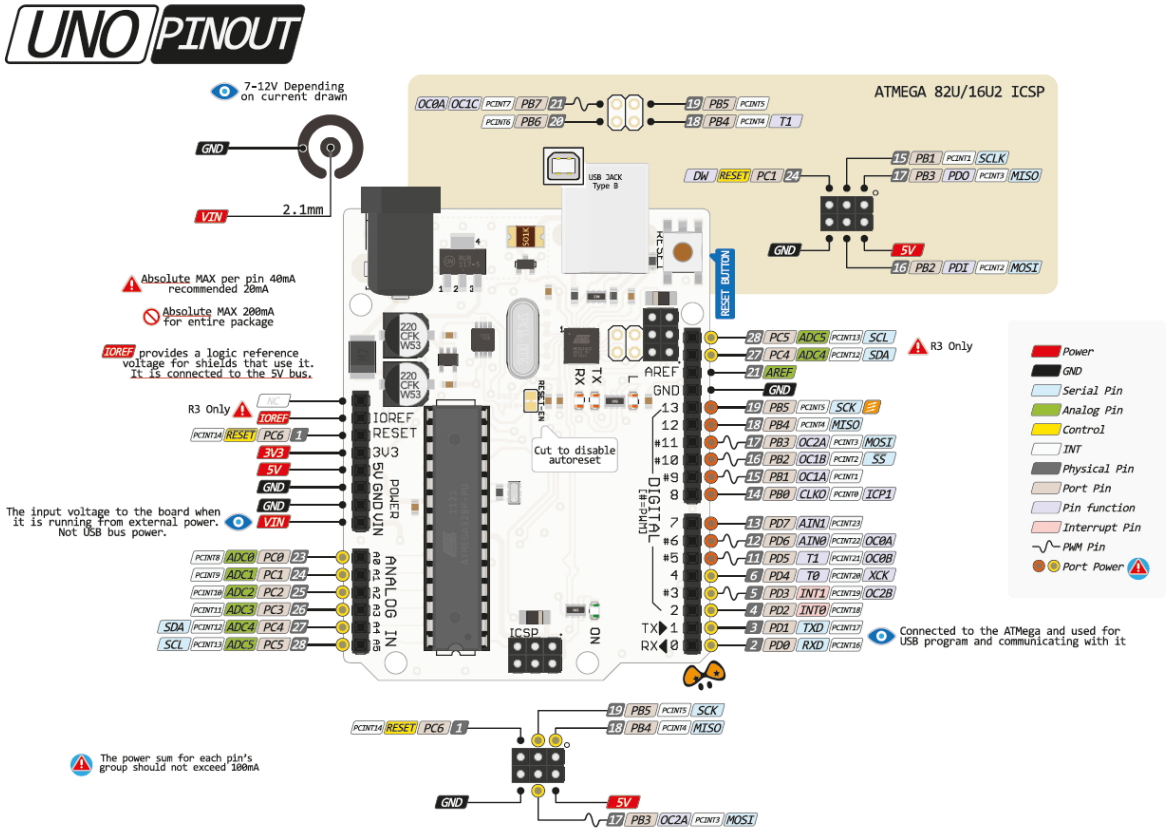
PROGRAM 5

```
#define LED  13

void setup() {
  Serial.begin(9600);
  pinMode(LED, OUTPUT);
}

void loop() {
  char ch;
  if (Serial.available()) {
    ch = Serial.read();
  }

  if (ch == ‘H' || ch == ‘h’) {
    digitalWrite(LED, HIGH);
    Serial.println("LED ON");
  }
  else if (ch == ‘L' || ch == ‘l’) {
    digitalWrite(LED, LOW);
    Serial.println("LED OFF");
  }
  else {
  }
}
```


UNO PINOUT
7-12V Depending on current drawn
GND
VIN
2.1mm
Absolute MAX per pin 40mA recommended 20mA
Absolute MAX 200mA for entire package
IOREF provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
R3 Only
The input voltage to the board when it is running from external power. Not USB bus power.
The power sum for each pin's group should not exceed 100mA
ATMEGA 82U/16U2 ICSP
USB JACK Type B
RESET BUTTON
Cut to disable autoreset
ICSP
POWER
ANALOG IN
DIGITAL [PWM=#]
Connected to the ATMega and used for USB program and communicating with it
Power
GND
Serial Pin
Analog Pin
Control
INT
Physical Pin
Port Pin
Pin function
Interrupt Pin
PWM Pin
Port Power