ECOLE NATIONALE
SUPÉRIEURE
D'INFORMATIQUE

الجـــمهوريـــة الجزائريـــة الديمـــقراطيـــة الشـعبيـة
وزارة التعليـــم العـــالي والبـــحث العلمـــي
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

Intelligent and Communicating Systems, ICS

2^{nd} Year Specialty SIQ G02, 2CS SIQ2

LAB report n°01

Title:

Introduction to using and manipulating a microcontroller (Arduino and sensors)

Studied by:

First Name: Nour el Imane
Last Name: HEDDADJI
E-mail: jn_heddadji@esi.dz

A. Theory

1. Getting familiar with microcontrollers and highlighting the Harvard and Von-Neumann architectures

1.1. Microcontrollers

A microcontroller is a compact integrated circuit designed to control specific tasks within embedded systems. It combines a processor, memory, and I/O peripherals on a single chip.

1.2. Harvard and Von-Neumann architectures

Microcontrollers are built based on two fundamental architectural approaches: Harvard and Von-Neumann architectures.

Von-Neumann Architecture

Von-Neumann architecture is a microprocessor design where program instructions and data are stored in the same memory.The processor fetches instructions and data from the shared memory, which can lead to bottlenecks due to shared access.

Harvard Architecture

Harvard architecture, in contrast, separates instruction and data memory, using distinct buses.This separation allows the processor to simultaneously fetch instructions and data, resulting in faster processing.

2. LED and Pushbutton in Microcontrollers

In microcontroller-based projects,LEDs serve as indicators, providing visual feedback or status information. Pushbuttons, on the other hand, are used for user input or to trigger specific actions.

3. Getting familiar with Arduino

Arduino is a programmable electronic board that is equipped with a microcontroller,power source, I/O pins, and other circuitry. and it focuses on interacting with the outside world using sensors.

3.1. Getting Familiar with Arduino MKR WiFi 1010

Architecture

The Arduino MKR WiFi 1010 is powered by the Atmel SAMD21 microcontroller, which forms the core of the board.

Memory Type

- **Flash Memory**: With 256KB of flash memory, you have ample space to store your program code and data.
- **SRAM (Static Random-Access Memory)**: Offering 32KB of SRAM, it serves as temporary storage for data during program execution.

Pin Structure

- **Digital Pins (D0 to D14)**:can read/write digital signals,operate at 3.3 volts.
- **Analog Pins (A0 to A7)**: with a 12-bit resolution that allow you to read voltage levels between 0 and 3.3 volts.
- **PWM Pins (D0 to D8, D10, D12)**: support Pulse-Width Modulation (PWM), enabling you to generate analog-like output signals.
- **Additional Pins**:
  - **VIN**: This pin isused to supply power to the board from an external source.
  - **5V**: Provides regulated 5-volt power to other components connected to the board.
  - **3.3V**: Supplies regulated 3.3-volt power to connected components.
  - **GND**: These pins provide a common reference point for signals.
  - **RESET**: used to reset the board or put it into programming mode.
  - **TX, RX**: for serial communication with other devices.
  - **SPI Pins (MOSI, MISO, SCK, SS)**: used for Serial Peripheral Interface (SPI) communication with other devices.
  - **I2C Pins (SDA, SCL)**: facilitate Inter-Integrated Circuit communication with other devices.

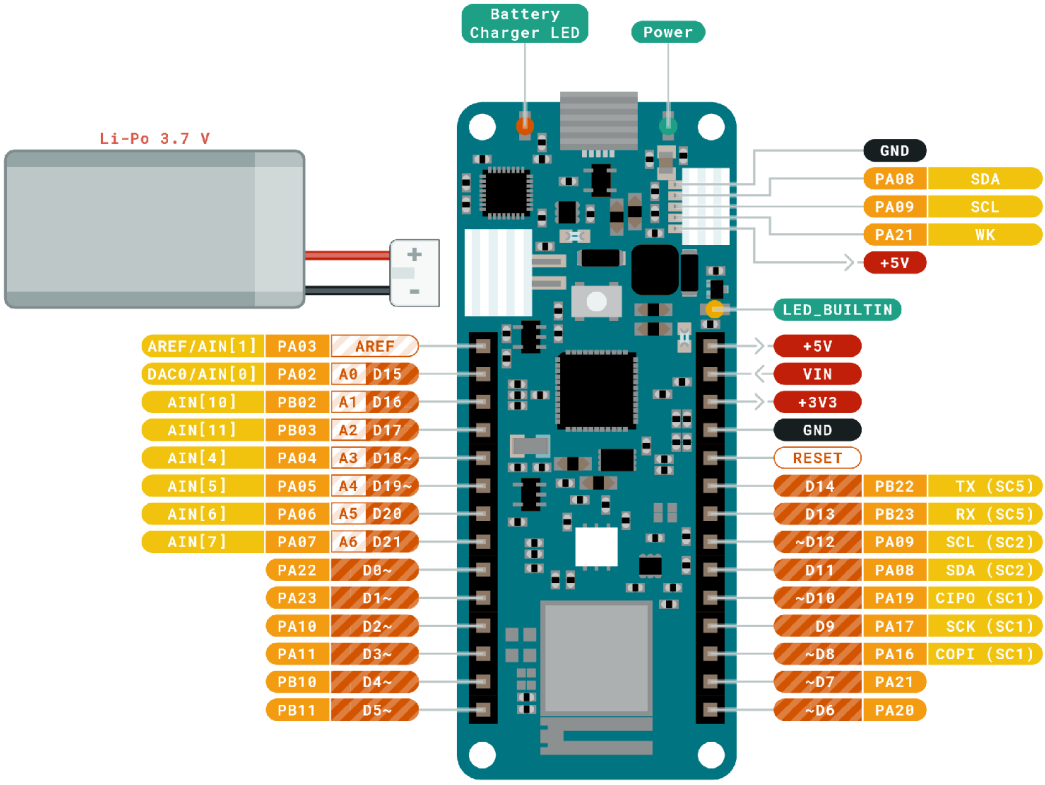

Figure 1: Arduino Mkr wifi 1010 pinout

3.2. Structure of an Arduino Program

1. **Variable Declarations:** In this section, you declare global variables that can be accessed from anywhere in the program.

2. **Setup Function:** This function is executed once when the Arduino board is powered on or reset. It is used for initialization tasks, such as setting pin modes and initializing variables.

3. **Loop Function:** The loop function is called repeatedly after the setup function. It is where you write the main logic of your program.

3.3. Libraries and Functions

Libraries in an Arduino program are collections of pre-written functions that simplify the task of programming the microcontroller, each one is dedicated to a specific task like controlling sensors and motors, as well as mathematical and utility functions. here are some examples :
WiFiNINA library : used for connecting to WiFi networks and managing network connections.
SD library : used for reading and writing data to SD cards.
Servo Library: This library is used for controlling servo motors, which are commonly used in projects that require precise control of the motor's position.

```arduino
int button = 0;

void setup()
{
  pinMode(2, INPUT);
  pinMode(13, OUTPUT);
}

void loop()
{
  button = digitalRead(2);
  if (button == HIGH) {
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
  delay(10); // Delay a little bit to improve simulation performance
}
```

Figure 2: Arduino program structure

4. Installing and Configuring Arduino Development Tools

To get started with Arduino development, you need to install and configure essential tools. Follow these steps:

1. **Download Arduino IDE**: Visit the official Arduino website to download and install Arduino IDE at https://www.arduino.cc/en/software
2. **Connect Arduino Board**: Use a USB cable to connect your Arduino board to your computer. Open the Arduino IDE and select the correct board and port from the ”Tools” menu.
3. **Install Drivers (If Needed)**: If the board has not been connected to the computer before, you may need to install drivers for it. Drivers can be downloaded from the manufacturer’s website.
4. **Use Libraries**: To use libraries, they should be installed using the Arduino Library Manager, which is accessible from the ”Sketch” menu in the Arduino IDE. Write the code in the Arduino IDE using the programming language based on C/C++.
5. **Verify the Sketch**: Verify the sketch by clicking on the ”Verify” button. This will check for any errors in the code.
6. **Upload Your Code**: Once the code is verified, it can be uploaded to the board by clicking on the ”Upload” button.
7. **Test Your Project**: Disconnect the USB cable from the computer and connect your Arduino board to the power source. Test the project.

4.1. Test Lab Connection and Testing

Bread-board

A breadboard typically consists of a grid of holes, with each row and column of holes electrically connected internally. The breadboard also has two sets of horizontal rows at the top and bottom for power and ground connections.

Connecting a Pushbutton and LED

1. Place the pushbutton onto the breadboard so that each of its legs is inserted into a separate row of holes.
2. Connect one leg of the pushbutton to the ground using a jumper wire.
3. Connect the other leg of the pushbutton to a separate row on the breadboard using another jumper wire. Then, connect a resistor between this row and the 5V rail on the breadboard.
4. Connect a third jumper wire from the same row as the pushbutton to a digital input pin on the Arduino board.

The same thing when connecting a LED.

4.2. Arduino Protection

1. **Protecting Input Pins**: To protect input pins, avoid applying voltage or current beyond the board's specifications. Use resistors to limit current if needed.
2. **Protecting Output Pins**: When dealing with output pins, avoid short-circuits and overloading. Use appropriate current-limiting resistors with LEDs.
3. **LED and Pushbutton Protection**: To ensure the longevity of LEDs and pushbuttons, follow datasheet recommendations and consider using current-limiting resistors.

How calculate the resistance needed for pins protection?

When connecting components such as LEDs or other light-emitting devices, it's important to use current-limiting resistors to prevent excessive current from flowing through the I/O pin.To calculate the resistance needed for pins protection, you can use Ohm's law :

$$R = \frac{V_{\text{ard}} - V_{\text{led}}}{I}$$

Where:
R - Resistance (Ohms)

V_{ard} - Arduino supply voltage (V)
V_{led} - Forward voltage of the component (V)
I - Current into the Arduino pin
Ensure R is **greater than or equal** to the minimum required resistance (R_{min}):

$$R \geq R_{\text{min}}$$

Where: R_{min} - Minimum resistance for pin protection.

5. Introduction to Microcontroller Simulators

Microcontroller simulators are virtual tools for learning and testing microcontroller-based projects without physical hardware.

5.1. Advantages

1. **Cost-Effective**: No need for physical components.
2. **Safe**: No risk of hardware damage.
3. **Accessible:** Available online.
4. **Educational:** Great for learning and debugging.

5.2. Popular Simulators

1. **Proteus:** Extensive component library, real-time debugging.
2. **Tinkercad:** User-friendly, suitable for beginners.

5.3. Getting Started

1. Download and install the chosen simulator.
2. Create a project, select the microcontroller model.
3. Add components, write and upload code.
4. Use debugging features for analysis.

B. Activity

1. Connecting a LED

1.1. Turning on the built-in led

```
void setup ()
{
pinMode ( LED_BUILTIN , OUTPUT ); // set the buitin led as the output
}
void loop ()
{
digitalWrite ( LED_BUILTIN , HIGH ); // turn on the builtin led
delay (1000) ; // Wait for 1000 millisecond (s)
digitalWrite ( LED_BUILTIN , LOW ); // turn off the builtin led
delay (1000) ; // Wait for 1000 millisecond (s)
}
```

Listing 1: Turning on the built-in led Arduino program

Analysis

In this program, we use the setup() function to configure the built-in LED as an output using the pinMode() function. The loop() function turns the LED on and off at one-second intervals. The digitalWrite() function is used to set the LED state, and the delay() function pauses the program for a specified time.

1.2. Turn on an external LED

```
void setup () {
pinMode (6 , OUTPUT ) ; // Set digital pin 6 as output
}
void loop () {
digitalWrite (6 , HIGH ) ; // Turn on the LED
delay (1000) ; // Wait for 1 second
digitalWrite (6 , LOW ); // Turn off the LED
delay (1000) ; // Wait for 1 second
}
```

Listing 2: Turning on a led Arduino program

Analysis

In this program, we select digital pin 6 as an output and use the digitalWrite() function to control an external LED. Similar to the previous example, we turn the LED on and off with one-second intervals.

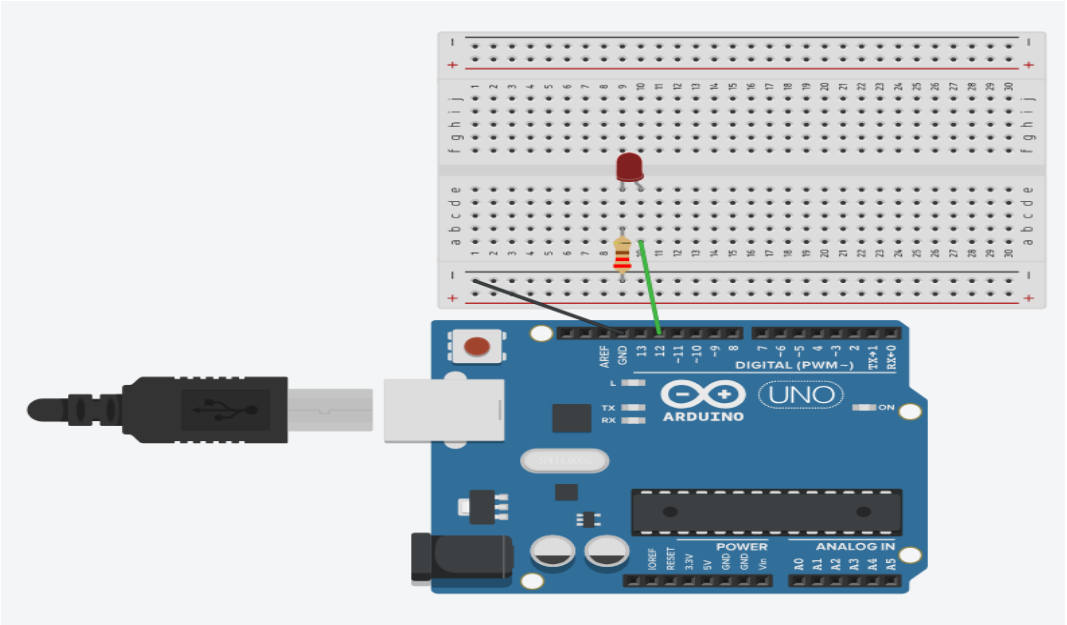

Figure 3: Arduino program structure

2. Scan of wireless Wifi networks

MKR1010 boards are equipped with NINA-W10 modules that implement Wi-Fi 802.1 1b/g/n technologies on the 2.4 GHz ISM band and Bluetooth v4.2 technology.

2.1. Printing the number of available Wi-Fi networks

```
#include <SPI.h>
#include <WiFiNINA.h>

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    // Wait for the serial port to connect
    }
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi module not found");
    while (true);
  }
  // Scan for available Wi-Fi networks:
  Serial.println("Scanning available networks...");
  int numNetworks = WiFi.scanNetworks();
  Serial.print("Number of available networks: ");
  Serial.println(numNetworks);
}
void loop() {
  // Loop code here (not needed for network scanning example).
}
```

Listing 3: print the number of available Wi-Fi networks

Analysis

The code begins by setting up serial communication and checking the presence of the WiFi module. `int numNetworks = WiFi.scanNetworks();`finds and counts the available Wi-Fi networks. The result, which is the number of networks found, is stored in the numNetworks variable.

2.2. Listing the networks' names and signal strengths

```
#include <SPI.h>
#include <WiFiNINA.h>

void setup() {
  // Initialize serial and wait for the port to open:
  Serial.begin(9600);
  while (!Serial) {
    // Wait for the serial port to connect. Needed for native USB port only
    .
  }
  // Check for the presence of the WiFi module:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi module not found");
    // Don't continue:
    while (true);
  }
  // Scan for available Wi-Fi networks:
  Serial.println("Scanning available networks...");
  listNetworks();
}
void loop() {
}
void listNetworks() {
  // Scan for nearby networks:
  Serial.println("** Scan Networks **");
  int numNetworks = WiFi.scanNetworks();
  if (numNetworks == -1) {
    Serial.println("Couldn't get a Wi-Fi connection");
    while (true);
  }
  // Print the network name and signal strength for each network:
  for (int i = 0; i < numNetworks; i++) {
    Serial.print(i + 1);
    Serial.print(") ");
    Serial.print(WiFi.SSID(i));
    Serial.print("\tSignal: ");
    Serial.print(WiFi.RSSI(i));
    Serial.println(" dBm");
    Serial.flush();
  }
}
```

Listing 4: Listing the networks' names and signal strengths

Analysis

The `listNetworks` function carries out the scanning and listing of networks. It counts the number of available networks and uses a loop to print the network number, name, and signal strength for each network.
The `Serial.flush()` command ensures that the data is sent and displayed immediately.

2.3. Dynamic memory allocation

```
#include <SPI.h>
#include <WiFiNINA.h>

char* ssid = NULL; // Pointer to store SSID dynamically
char* password = NULL; // Pointer to store WiFi password dynamically

void setup() {
  Serial.begin(9600);

  // Check for the WiFi module
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi module not present");
    // Don't continue if the WiFi module is not available
    while (true);
  }
  ssid = (char*)malloc(20); // Allocate memory for SSID
  password = (char*)malloc(20); // Allocate memory for password
  if (ssid == NULL || password == NULL) {
    Serial.println("Memory allocation failed");
    while (1);
  }
  // Copy SSID and password values
  strcpy(ssid, "your_SSID");
  strcpy(password, "your_PASSWORD");
  // Connect to WiFi
  int status = WiFi.begin(ssid, password);
  if (status != WL_CONNECTED) {
    Serial.println("Failed to connect to WiFi");
  } else {
    Serial.println("Connected to WiFi");
    // Free dynamically allocated memory after successful connection
    free(ssid);
    free(password);
    ssid = NULL;
    password = NULL;
  }
}

void loop() {}
```

Listing 5: dynamic memory allocation

Analysis

This code is all about efficient memory use. It dynamically allocates memory to store the network's SSID using `malloc`, and after successfully connecting to WiFi, it deallocates that memory using free. This is a helpful technique when conserving memory on your Arduino is essential, especially in cases with limited resources.

3. Memory Management

Practical instances of memory management on Arduino:

1. **Dynamic Memory Allocation:** Efficiently allocate memory as needed, as demonstrated in the Wi-Fi network connection example, to conserve resources.
2. **Optimized Data Storage:** Use dynamic memory allocation to store data in arrays, allocating memory for exactly what's required.
3. **Circular Buffers:** Manage continuous data streams efficiently by using circular buffers, preventing memory overflow.
4. **String Handling Care:** Handle strings cautiously to avoid memory issues like buffer overflows and leaks.
5. **Arrays for Diverse Sensors:** Employ arrays of structures to effectively manage memory when dealing with various sensor types.

4. Simulation

Proteus is a simulation and electronic design development tool.// To turn on a building LED and an external LED using the Proteus simulator, you'll need to follow these steps:

4.1. Download and Install Proteus

visit https://www.labcenter.com/ the official Proteus website at to download and install Proteus.

4.2. Add Arduino Library

Before creating your Arduino-based simulation, you'll need to incorporate the necessary Arduino libraries into Proteus.

1. Downloading the Arduino libraries you intend to work with.

2. Place them into Proteus "LIBRARY" folder typically found at 'C:Files (x86)Electronics8 Professional' for Windows.

4.3. Create the LED Circuit

1. Launch Proteus as an administrator to ensure you have the required permissions.
2. Access the "Library Panel" within Proteus and search for the following components: "Arduino Uno", "Resistor", "LED Yellow" and "Ground."
3. Select the pin connected with the LED on the Arduino and establish connections between the components using wires.

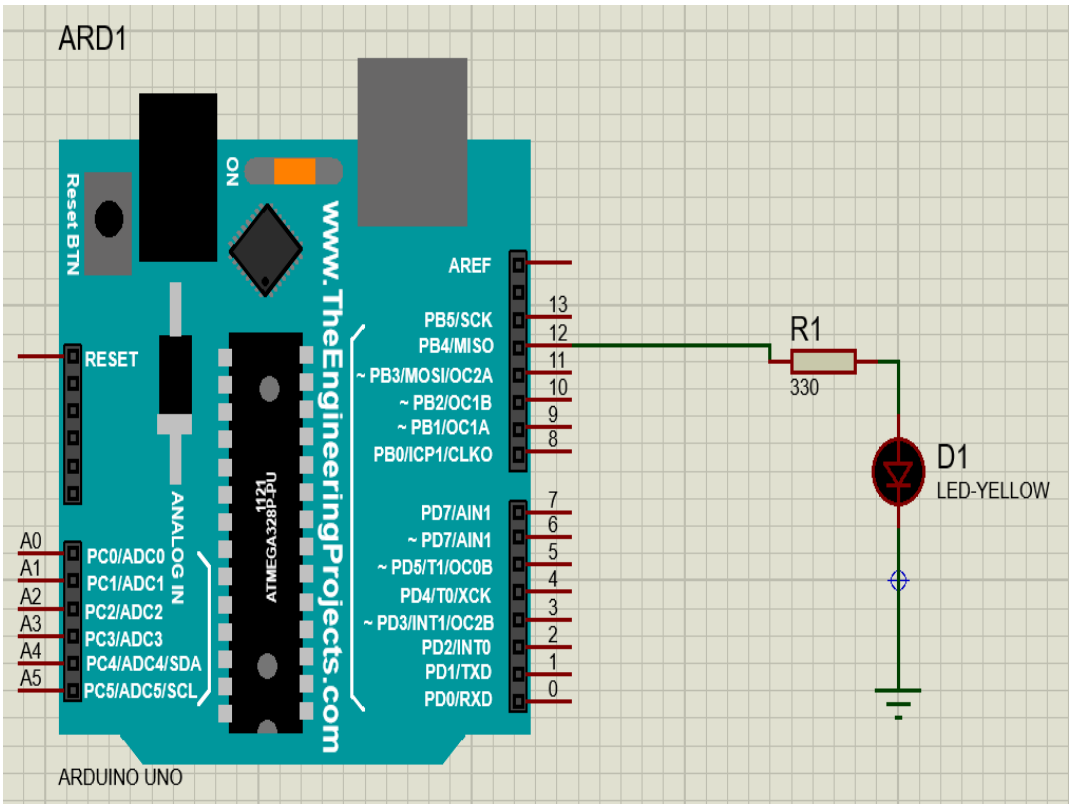

Figure 4: The simulation circuit in Proteus

4.4. Arduino Code

1. Launch the Arduino Integrated Development Environment (IDE).
2. Reuse the code used in the previous section with the Arduino MRK WIFI 1010 board to make an LED blink. The only modification is to use pin 6 instead of the previous pin.
3. After writing the code, ensure that you save it on your computer for later use.

4.5. Compile the Code

1. Once your code is complete, compile it using the Arduino IDE.
2. After compilation, you can locate the compiled HEX file in the output panel. If it's not visible, you can enable verbose output during compilation by following these steps: Go to "File" > "Preferences."

4.6. Integrating code into proteus

1. In Proteus, choose the "Arduino Uno" board.
2. An "Edit Component" window will open. Look for the "Program File" option within this window.
3. Paste the path to the HEX file copied earlier from the Arduino IDE's output panel into the "Program File" field.
4. After pasting the path, click "OK" to incorporate the code into the Arduino component within Proteus.

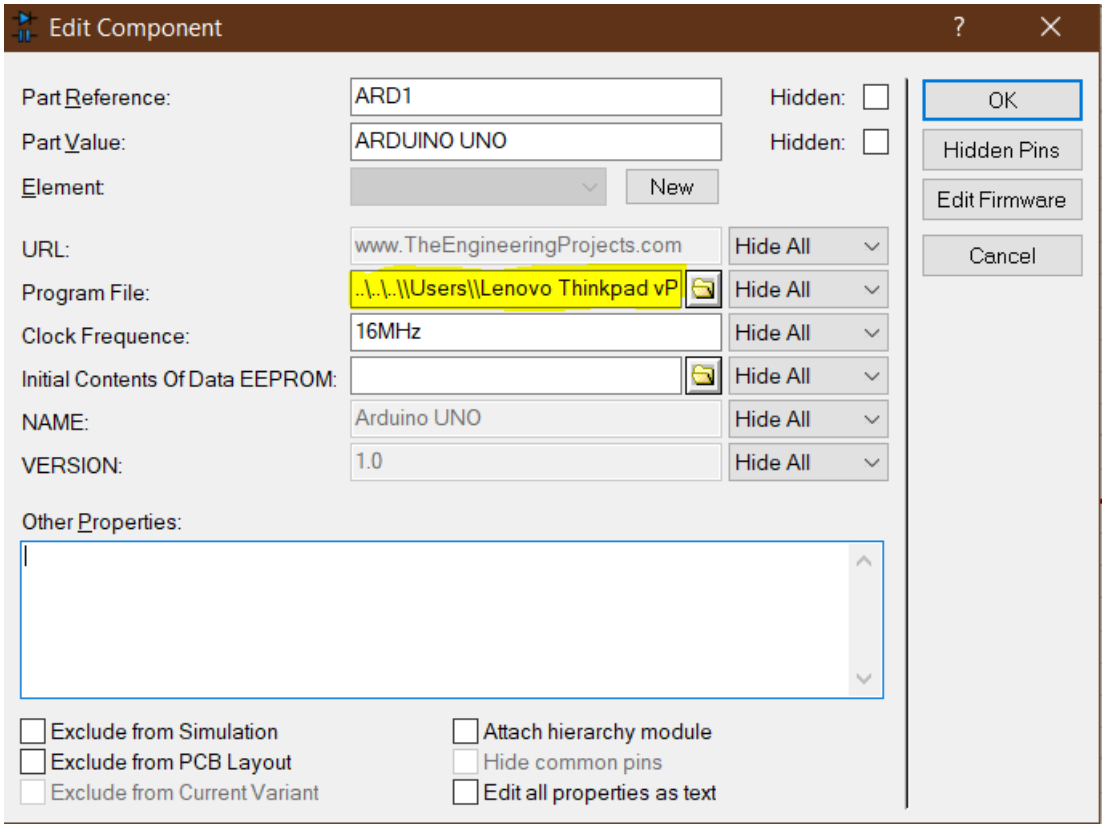

Figure 5: Edit component window for Arduino UNO on proteus

4.7. Run the simulation

Run the simulation and observe the LED

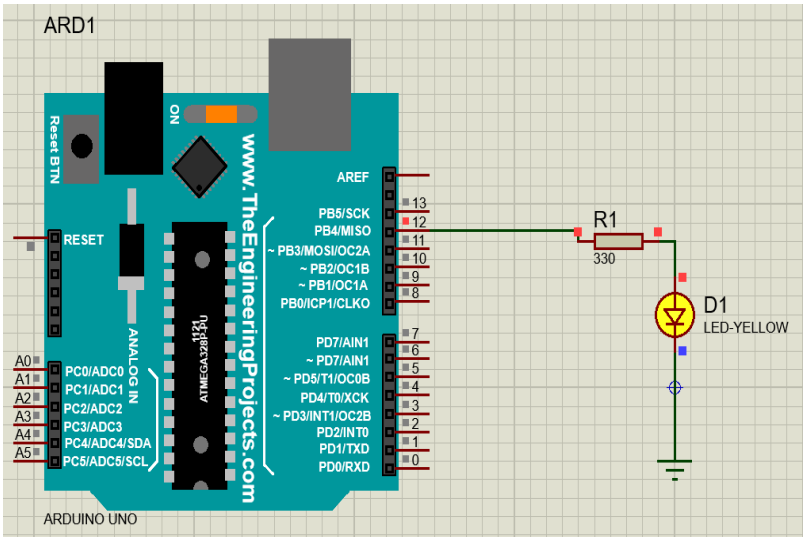

Figure 6: Blinking LED

5. Conclusion

The activities undertaken in this chapter emphasize the boundless potential that Arduino offers, making it accessible and empowering both beginners and experienced makers.
In an increasingly digital world, microcontrollers like Arduino serve as vital bridges between the digital and physical domains. They play a pivotal role in powering everyday technologies and fostering innovation to tackle real-world challenges, particularly in the realm of IoT (Internet of Things).