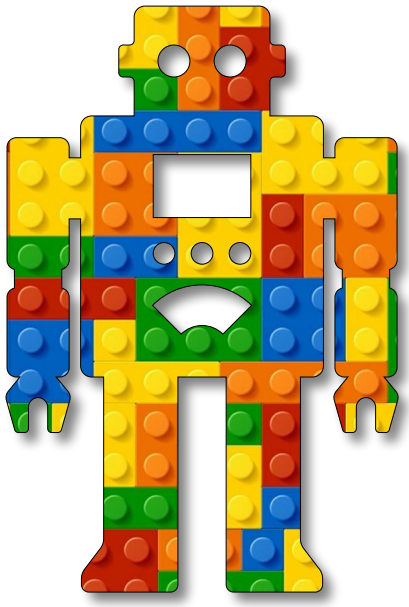
Intelligent Machines Module A Unit #9 the button



Contents

Module A Unit #9 button

Introduction to the button

The button is a tactile or momentary push button. It makes contact with a metal plate when you push down and completes the circuit. When you stop pushing, it releases and is no longer in contact and hence breaks the circuit. See Fig. 2.

As you can see, it has four pins protruding from the body of the button and a black button on top. This kind of button fits nicely on the breadboard. The pins are connected as shown in Fig. 2. When you press the button, you connect the pins from left to right (as seen in the diagram below). The pins top to bottom are already connected internally.

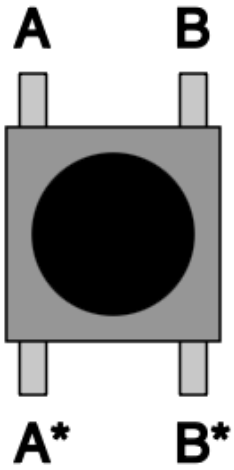
To use this button, see Fig.1 (rather than a button module), we need a resistor. The beauty of the Arduino is that it has a built-in resistor we can use with the button (but not with an LED!). It is called a pull-up resistor, more on that later.

Figure 1: button

How the button is connected

As you look down at the button, it has four legs (pins). The pins A and A* are connected permanently, as are B and B*. When you press the button, A and B are connected, and A* and B* are also connected.

Figure 2: looking down



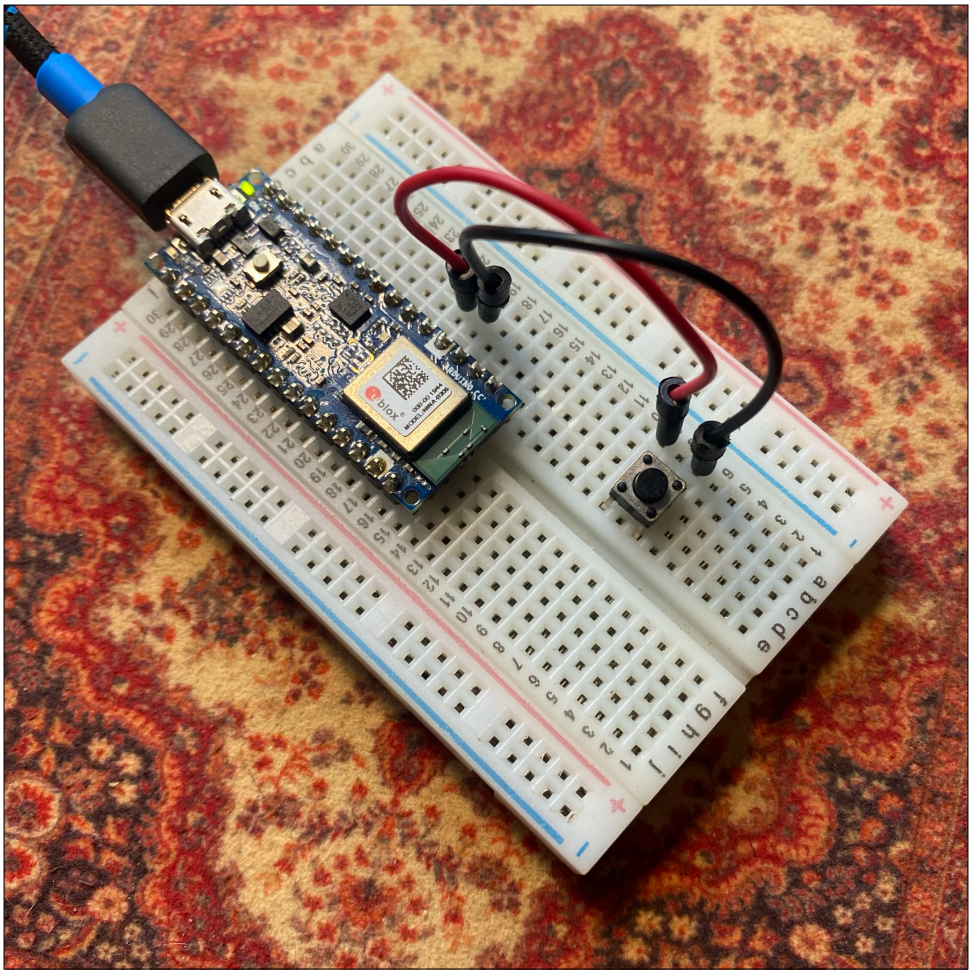
What you will need

The full list is here. Look at the image below and the wiring diagram (fig.5).

1 x Arduino Nano 33 BLE
1 x breadboard
1 x button
2 x male-to-male jumper leads

You could add a resistor in series with the button, but the Nano has a built-in resistor that you can pull up. This saves you the bother. The downside is that the logic (HIGH = off and LOW = on) is reversed.

Figure 3: component setup



Circuit Diagram for the button

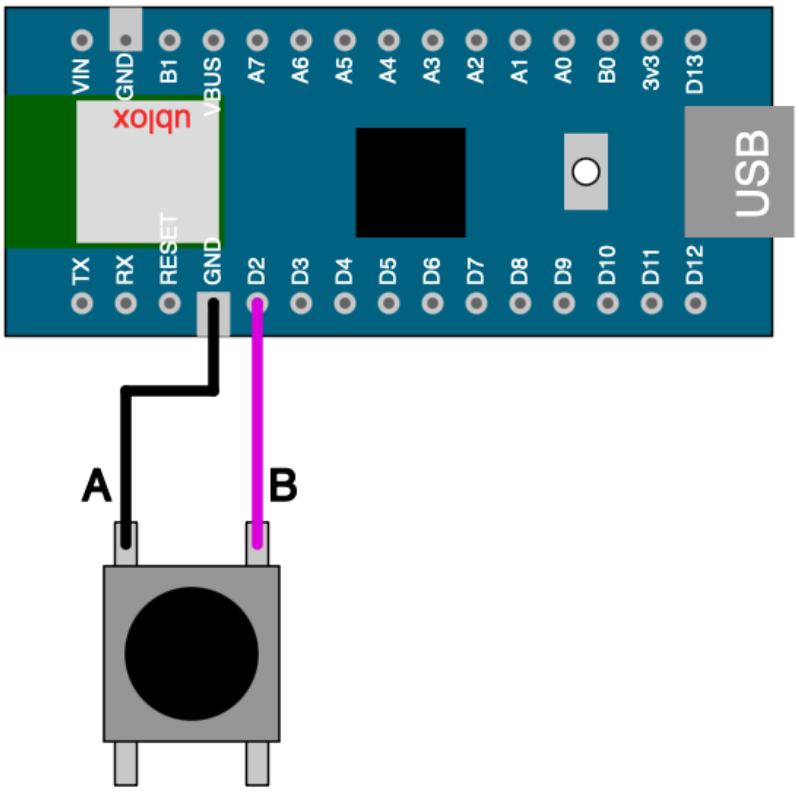
You will need two wires to connect the button to the device.

Button Pins	Arduino Pins
A	GND
B	2 (digital pin D2)

We are going to add the button to the breadboard. You can put it anywhere away from the board itself.

See Fig.4 below. The wiring diagram shows the connections.

Figure 4: wiring diagram



Sketch A9.1 LED button

Connect the button as shown, then after writing the code and uploading it to the Arduino, press the button. The LED should come on when pressed and off when released. When using the pull-up resistor, the default (not pressed) state is HIGH. This is similar to the RGB LED, still a little bit counterintuitive; if you use an external resistor, the opposite (more logical) is true.

Arduino sketch

```
int ledPin = 13;
int buttonPin = 2;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop()
{
  int button = digitalRead(buttonPin);
  if (button != LOW)
  {
    digitalWrite(ledPin, LOW);
  }
  else
  {
    digitalWrite(ledPin, HIGH);
  }
}
```

🗒 Notes

As an alternative, I have given the button and the LED variable names. This allows you to make changes once rather than sifting through the whole code and changing it multiple times.

🛠️ Code Explanation

`pinMode(buttonPin, INPUT_PULLUP);`	Using the pull-up (internal) resistor
`if (button != LOW)`	The != means not. If the button is not LOW (pressed)...

Sketch A9.2 LED toggle

In this sketch, we are doing more than just switching it on and off with the button, but toggling it so that on one press the LED is on and on the next press of the button the LED switches off. This is more difficult than the previous sketch. Hold the button for a second each time.

! This works very badly because of a condition known as bounce (we will look at that in the next sketch).

Arduino sketch

```
int ledPin = 13;
int buttonPin = 2;
bool ledState = LOW;
bool lastButtonState = HIGH;
bool currentButtonState = HIGH;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop()
{
  lastButtonState = currentButtonState;
  currentButtonState = digitalRead(buttonPin);
  if(lastButtonState == LOW && currentButtonState == HIGH)
  {
    ledState = !ledState;
    digitalWrite(ledPin, ledState);
  }
}
```

🗒️ Notes

I will be honest with you, this requires some logical thought to work out what is happening. In simple terms, when you press the button, the state of current and previous end up being `LOW`, but when the button is released, the current then becomes `HIGH`, thus changing the `ledState` from either `HIGH` to `LOW` or `LOW` to `HIGH`.

Just work through the sketch to follow the logic; the logic isn't flawed, but the button is. The problem is that the contacts, as you press the button, jump or `bounce` and give false readings. The next sketch addresses this problem by taking into account the `bounce`.

🛠️ Code Explanation

`bool ledState = LOW;`	This boolean variable defines the state of the LED, initially it is set to on.
`bool lastButtonState = HIGH;`	We introduce a boolean previous state to compare to the current.
`bool currentButtonState = HIGH;`	This boolean state is also the current and is used to compare to the the previous.
`lastButtonState = currentButtonState;`	This updates the state of the previous state.
`currentButtonState = digitalRead(buttonPin);`	The latest state is updated by reading if the button has been pressed.
`if(lastButtonState == LOW && currentButtonState == HIGH)`	When the button is pressed the state is LOW but when the the button is released it is HIGH.
`ledState = !ledState;`	When the above condition is true: the lastButtonState is LOW and the currentButtonState is HIGH the LED is turned on if off and on if off.

Sketch A9.3 debounce

I recommend starting a new sketch; too many changes to highlight. To tackle the bounce problem, we need to introduce some sort of delay to counter that. We will call that debounceDelay.

Arduino sketch

```
int ledPin = 13;
int buttonPin = 2;
int ledState = LOW;
int buttonState = LOW;
int lastButtonState = LOW;
int currentButtonState = LOW;

long lastDebounceTime = 0;
long debounceDelay = 50;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
  digitalWrite(ledPin, ledState);
}

void loop()
{
  currentButtonState = digitalRead(buttonPin);
  if (currentButtonState != lastButtonState)
  {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay)
  {
    if (currentButtonState != buttonState)
    {
      buttonState = currentButtonState;
      if (buttonState == HIGH)
      {
```

```
        ledState = !ledState;
      }
    }
  }
  digitalWrite(ledPin, ledState);
  lastButtonState = currentButtonState;
}
```

🗒️ Notes

This is more about Boolean logic than the code. But all that code is just to toggle an LED on or off. This is what coding is all about: problem solving. This is a workaround for a hardware issue.

🛠️ Code Explanation

`long lastDebounceTime = 0;`	These two lines of code are the key to this working, they have to be long because they use the millis() function and the number can get big very quickly.
`long debounceDelay = 50;`	