Embedded Systems Programming Curriculum for Vertically-Integrated Projects

Gregorio E. Drayer and Ayanna M. Howard

Note of disclosure: This curriculum is currently under development for use by participants of the Vertically-Integrated Projects (VIP) Program. The VIP Program is an undergraduate education program that operates in a research and development context. The teams are: multidisciplinary - drawing students from across the engineering and science disciplines; vertically-integrated - maintaining a mix of sophomores through seniors; and long-term - each undergraduate student may participate in a project for up to three years.

Contents

Getting Started with Arduino

The projects in this Chapter are based on the Guide for the SparkFun Inventor's Kit for Arduino [1].

0.1 Project 0: Turning On and Off an LED

0.1.1 Learning Objectives

This first example introduces the reader to the structure followed in this program to facilitate the learning process. Just as here, every project will include a *Learning Objectives* Subsection. It will be followed by the implementation of the project, which in its case will include the following:

1. Materials.
2. Circuit diagram.
3. Code.

Finally, every project will have a Summary with comments about what the reader should have learned and some possible applications of the concepts taught.

For this initial project, called *Project 0*, the objective is to introduce the reader to the most simple exercise making use of the Arduino board. The idea is to attract the uninitiated to the components of an embedded system in the most simple case, but at the same time offering the opportunity to see the implementation in action: turning on and off an LED.

0.1.2 Implementation

The Subsection elaborates on materials, the circuit diagram and the code that needs to be uploaded to the Arduino board. Because this is the first implementation in this curriculum, the reader will find additional details about the circuit diagram and how it may be put together on a prototyping board.

Materials

The materials needed for this project are:

1. Light-emitting diode, x1.
2. 330 Ω Resistor, x1.
3. Wires, x3.

Circuit Diagram

Every project will include the circuit diagram of the exercise at hand. In this case the circuit diagram for Project 0 is presented in Figure 1.

For this exercise, it is good to know or remember that the polarity of the LED follows the convention presented in Figure 2.

Additionally, the connectors of the Arduino board that are referred to in Figure 1 are illustrated in Figure 3.

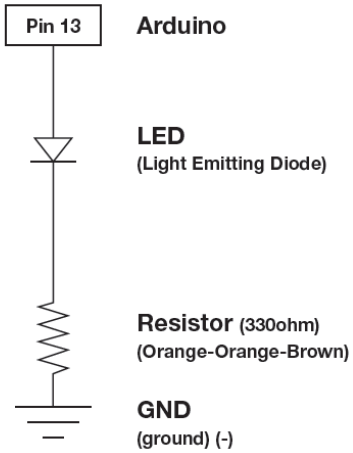

Figure 1: Project 0 Circuit Diagram

Code

To implement the code for this Project, you need to execute the Arduino IDE and type in the following code:

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}
```

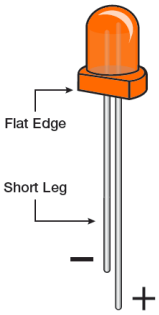

Figure 2: Polarity of LEDs.

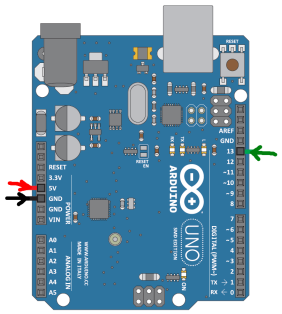

Figure 3: Connection points on the Arduino board for *Project 0*

.

```
void loop() {
  digitalWrite(13, HIGH); // set the LED on
  delay(1000); // wait for a second
  digitalWrite(13, LOW); // set the LED off
  delay(1000); // wait for a second
}
```

Once you have typed the code above, you will need to compile it by pressing the "Verify" button. If there are no errors in your code, the status of the verification process will read "Done compiling." To upload the compiled file, you need to make sure that the Arduino IDE is configured to communicate through the correct serial port. After doing so, the transfer from the computer into the Arduino board is completed by pressing the "Upload" button.

You should see the LED toggling on and off every one second. This very simple introductory project illustrates the elements used to make intermittent lights work. Such function finds applications in aviation, traffic signs, and indicators in all kinds of devices.

0.1.3 Summary

This initial Project should have helped the reader understand the approach that other Projects will follow in explaining the purpose of each one of the following projects, as well as the structure and information that will be provided.

In this particular project, the reader had the opportunity to implement a most simple circuit that helps to have a first hands-on experience combining circuitry and software. All following projects will make use of the same approach. The reader should notice that there is a clear interface between hardware and software implementations. On the hardware side, the circuitry is intended to interconnect the external devices to the correct ports of the processing device. Taking careful attention to the polarity of devices will help to avoid burning components or overloading the board and power source.

0.2 Project 1: Photo Resistor

This project implements a photo resistor. This component is useful to detect ambiance light and its intensity. The information collected from the photo resistor can be used to implement software routines that will send commands to other components based on the light intensity of the environment being sensed. An image of a photo resistor and its parts is presented in Figure 4.

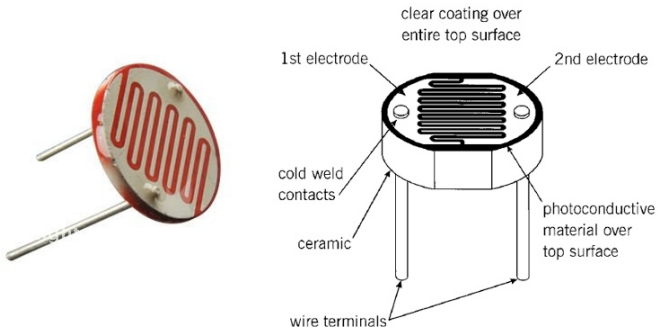

Figure 4: Image of a photo resistor and diagram describing its parts.

0.2.1 Learning Objectives

In this project the objective is to learn how to sense analog signals and their conversion into a quantized digital signal by means of a analog-to-digital converter (ADC), one of three in the Arduino Uno board.

0.2.2 Implementation

Analog signals are usually going to be based on a voltage, ranging from zero to a maximum voltage value, or V_max. This voltage range is going to be divided in a number of equal parts, as shown in Figure 5(a) [2]. The ideal transfer curve for a 4-bit ADC is shown in Figure 5(b). As a result, signals of a continuous nature that are susceptible to be transduced into an electrical signal can be converted to a digital one as shown in the example of Figure 6.

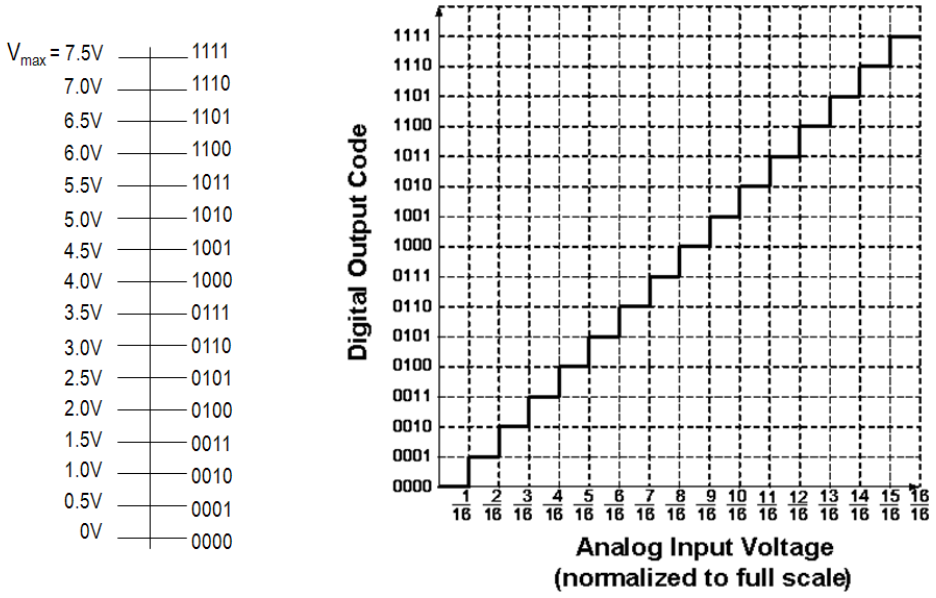

Figure 5: Encoding of analog signals and ideal transfer curve for a 4-bit ADC.

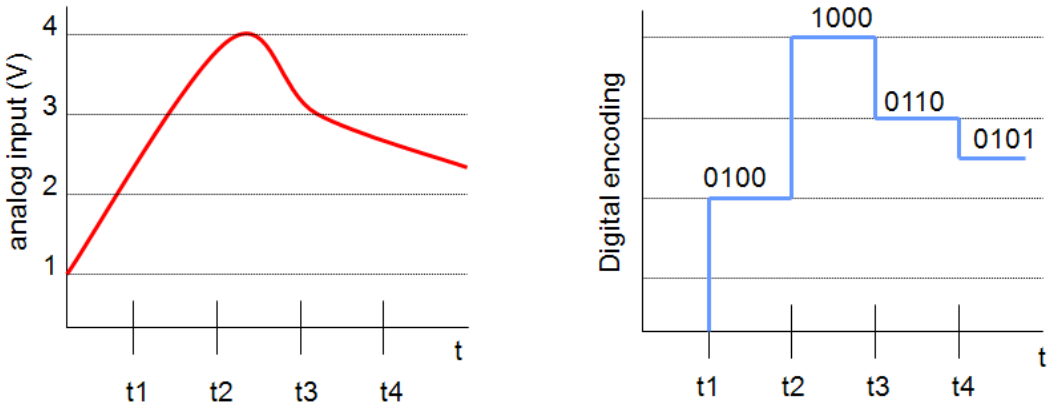

Figure 6: Example of a conversion from an analog signal to an analog one.

Materials

The materials required for this Project are the following:

- Photo resistor, x1.
- Light-emitting diode, x1.

- 330 Ω resistor, x1.
- Wire, x6.
- 10 KΩ, x1.

Circuit Diagram

The circuit diagram for this project is presented in Figure 7. Once the circuit has been correctly built on the prototyping board, the reader may proceed to typing the code on the Arduino IDE for compilation and upload. Please note the importance of verifying that the circuit has been correctly built. This may not be too critical in some of these examples, but it is a good habit to acquire when undertaking projects that may involve power applications or higher voltages.

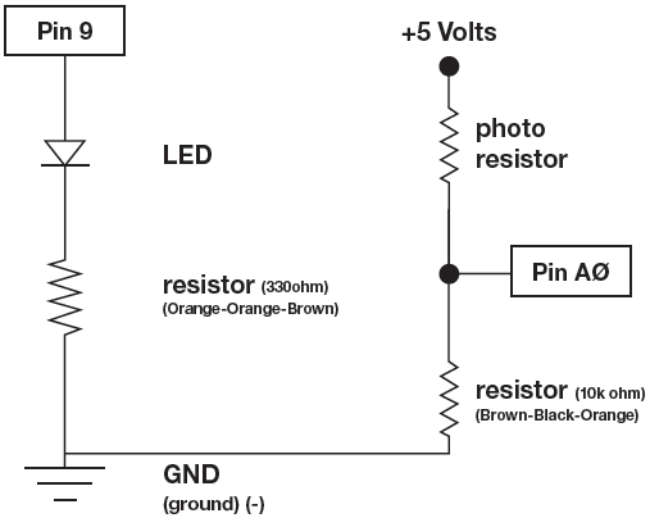

Figure 7: Project 1 Circuit Diagram

Code

This code makes use of the photo resistor in the circuit represented by Figure 7 to control the brightness of an LED. It begins by defining constants to name the pins used. This is a regular technique that facilitates the development of the functions that follow later in the code. This is a standard in the next projects as well, and in programming various other devices in languages such as C/C^{++}. If the circuit and code are well implemented, you should be able to see the LED change its brightness in accordance with the amount of light that the photo resistor is able to detect.

```
const int sensorPin = 0;
const int ledPin = 9;

int lightLevel, high = 0, low = 1023;

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

void loop()
{
```

```
  lightLevel = analogRead(sensorPin);
  // manualTune();  // manually change the range from light to dark
  autoTune();  // have the Arduino do the work for us!
  analogWrite(ledPin, lightLevel);
}

void manualTune()
{
  lightLevel = map(lightLevel, 0, 1023, 0, 255);
  lightLevel = constrain(lightLevel, 0, 255);
}

void autoTune()
{
  if (lightLevel < low)
  {
    low = lightLevel;
  }

  if (lightLevel > high)
  {
    high = lightLevel;
  }

lightLevel = map(lightLevel, low+30, high-30, 0, 255);
  lightLevel = constrain(lightLevel, 0, 255);
}
```

Observation: Note that in this code the developer has the option to select between manually calibrating the sensitivity of the sensing function of the system or to let the software do it automatically. Discuss with your team or classmates what are the advantages or disadvantages of employing one or the other, and explain how does the automatic algorithm perform such calibration.

0.2.3 Summary

Learning how to measure changes in resistance can be a very handy skill. Indeed, guide cited [1] elaborates by saying that "many of the sensors you'll use (potentiometers, photoresistors, etc.) are resistors in disguise. their resistance changes in proportion to whatever they're sensing (light level, etc.). The Arduino's analog input pins measure voltage, not resistance. But we can easily use resistive sensors with the Arduino by including them as part of a *voltage divider*. A voltage divider consists of two resistors. the *top* resistor is the sensor you'll be using. The *bottom* one is a normal, fixed resistor. When you connect the top resistor to 5 Volts, and the bottom resistor to ground, the middle will output a voltage proportional to the values of the two resistors. When one of the resistors changes (as it will when your sensor senses things), the output voltage will change as well. Although the sensor's resistance will vary, the resistive sensors (flex sensor, light sensor, softpot, and trimpot) in the SIK are around 10K Ohms. We usually want the fixed resistor to be close to this value, so using a 10K resistor is a great choice for the fixed *bottom* resistor."

0.3 Project 2: Temperature Sensor

This project implements a temperature sensor. Similar to the previous project, the temperature sensor makes use of a transduction phenomenon, in this case the thermo-electric effect, to transform a physical value into an analog and continuous voltage signal. The sensed signal in this case

will be subject to noises in the circuit, so special attention will be needed when implementing connections to ground on the bread board and on the arduino board.

0.3.1 Learning Objectives

The objective of this project is to employ a precision temperature sensor and to make temperature readings via serial communication between the Arduino board and the computer. As it will be suggested in Chapter **??**, this same project may be used to verify serial communication with the Raspberry Pi embedded Linux computer.

0.3.2 Implementation

In implementing the circuit for this project, it is suggested that the reader pays attention when selecting the temperature sensor; they may look very similar to regular transistors and an error in selecting it may damage it and other components in the circuitry. For the sake of clarity, this Subsection will present some images to ensure the proper use of the precision temperature sensor TMP36 and suggest the reader to make use of the TMP36 datasheet to find out more about this sensor.

Materials

The materials needed for this project are the following:

- Temperature sensor TMP36, x1.
- Wires, x5.

The temperature sensor looks like in Figure 8. According to its datasheet, the TMP36 is a low voltage, precision centigrade temperature sensor. It provides a voltage output that is linearly proportional to the Celsius (centigrade) temperature in the surrounding environment. The TMP36 does not require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C over the -40°C to +125°C temperature range.

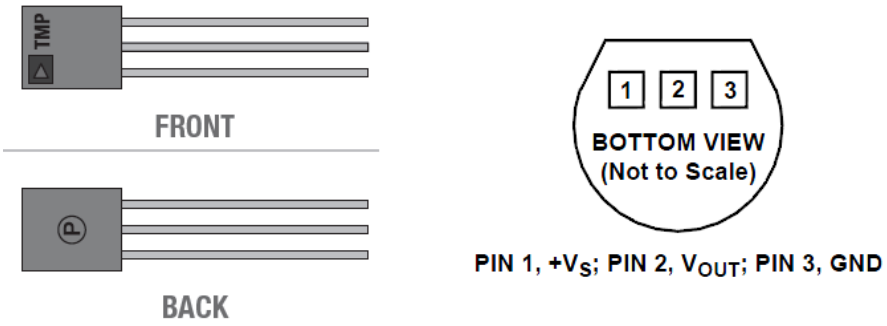

Figure 8: Detail of the precision temperature sensor TMP36.

Circuit Diagram

As you may tell from Figure 9, the circuit diagram for this project is simpler as it also makes use of only one component. However, as mentioned in Subsection 0.3.2, the developer needs to be careful with the selection of the temperature sensor component, but also with its polarity. Figure 8 aims to clarify any doubt about the use of the sensor, while noting that in Figure 9 the temperature sensor is seen from its top view.

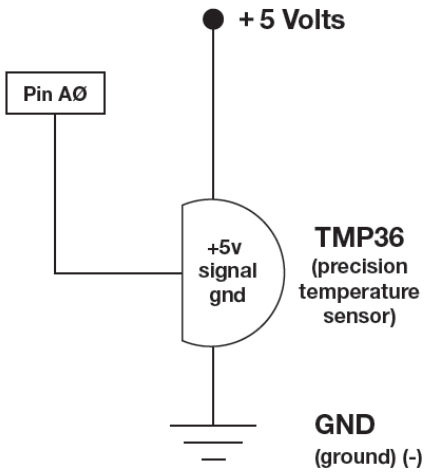

Figure 9: Project 2 Circuit Diagram

Code

Compile and upload the code below and make sure that the serial port is configured to communicate at a 9600 bps baud rate.

```
const int temperaturePin = 0;
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  float voltage, degreesC, degreesF;
  voltage = getVoltage(temperaturePin);
  degreesC = (voltage - 0.5) * 100.0;
  degreesF = degreesC * (9.0/5.0) + 32.0;
  Serial.print("voltage: ");
  Serial.print(voltage);
  Serial.print("  deg C: ");
  Serial.print(degreesC);
  Serial.print("  deg F: ");
  Serial.println(degreesF);
  delay(1000);
}

float getVoltage(int pin)
{
  return (analogRead(pin) * 0.004882814);
}
```

After uploading the code, you can use a serial communication program to read the temperature values reported by the circuit sampled at the frequency of choice. Some questions may be interesting at this point as to how would modify this circuit to activate an LED as a function of temperature? Think of such modification as looking forward to implement a thermostat as a future project. Given that you have a digital signal and you are able to program increasingly complex algorithms, think what embedded systems would you to develop for temperature control.

0.3.3 Summary

In this project, another analog sensor was employed to measure an environmental variable. In this ccase, it was temperature. The sensor used is employed in industrial applications such as in motor temperature monitors for vehicles. One important lesson learned should be the importance of making use of datasheets when getting to know circuit components. This skill builds up from simpler devices, like our temperature sensor, into more complex ones, which datasheets grow in length and complexity. The idea is to encourage the reader to keep such documents in mind, as they provide wealth of information useful to integrate embedded systems. Did you find and skimmed through the datasheet of the TMP36?

0.4 Project 3: Relay

This project is aimed to complement the previous one in guiding the reader on how to make use of a relay. This project could be implemented in combination with Project 2 to get one step closer to building a functional thermostat. Project 3 only focuses on the operation of the relay and the commands used to drive it. It will be left to the reader to elaborate on the lessons taught in this project to go further into developing more complete systems.

0.4.1 Learning Objectives

In the case of Project 3, the objective is to make use of a transistor and a relay to turn on and off two LEDs that indicate the state of the relay. If a power load (*i.e.* a heater) would be connected to the relay instead of the LEDs, the relay would drive the duty cycle of the heater according to the command used in the code below. The idea with this project is to hint to the reader how to make use of a relay and extend the learning material of this course into the ability to build a functional thermostat in combination with Project 2.

0.4.2 Implementation

A relay is a mechanical switch controlled by an electrical signal. It is composed of an electromagnet that closes or opens a connector between two terminals. The relay used for this project is presented in Figure 10.

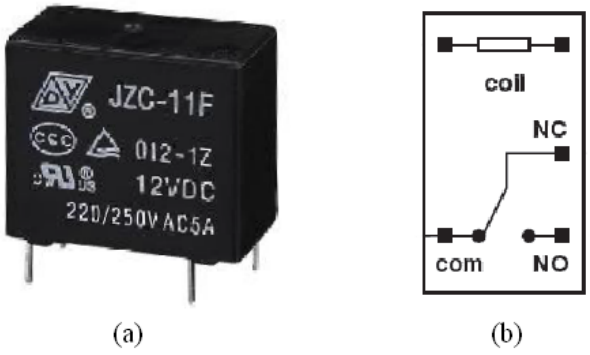

Figure 10: Image and diagram of the relay JZC-11F.

Materials

The materials needed for this project are the following:

- Relay JZC-11F, x1.
- Transistor P2N2222AG, x1.

- Diode 1N4148, x1.
- 330 Ω Resistor, x2.
- Light-emitting diodes, x2, 1 yellow, 1 red.
- Wires, x14.

The diagram describing the connectors of the transistor P2N2222AG are shown in Figure 11.

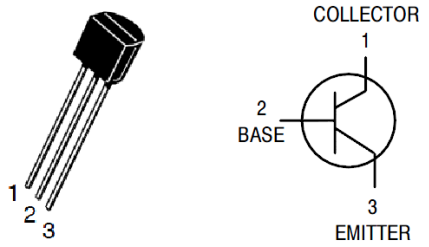

Figure 11: Description of the terminals of transistor P2N2222AG.

Circuit Diagram

Given the information about the relay and transistor to be used in this project, Figure 12 allows the reader to implement a simple circuit making use of the materials listed for this project.

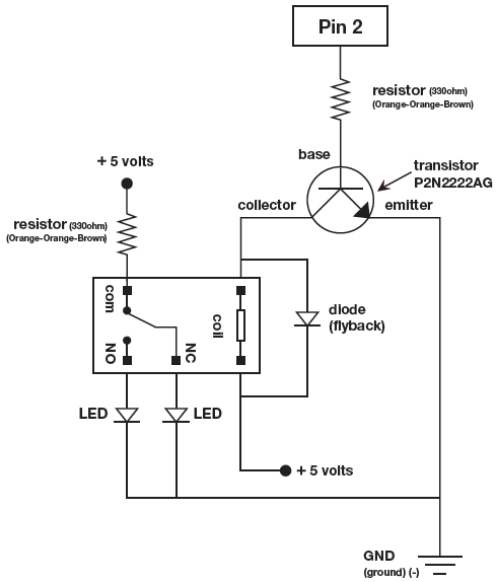

Figure 12: Project 3 Circuit Diagram

Code

Type in this code, compile and upload it to the Arduino. Please make sure to avoid to cycle electromechanical relays more than once every two or three seconds; some relays are more delicate than others and, if they are cycled too fast, overheating may result in permanent damage in the device.

```
const int relayPin = 2;     // use this pin to drive the transistor
const int timeDelay = 3000; // be careful about the on off duty cycle

void setup()
{
  pinMode(relayPin, OUTPUT);  // set pin as an output
}

void loop()
{
  digitalWrite(relayPin, HIGH);  // turn the relay on
  delay(timeDelay);              // wait for one second
  digitalWrite(relayPin, LOW);   // turn the relay off
  delay(timeDelay);              // wait for one second
}
```

0.4.3 Summary

In this project, the reader has been introduced to a relay. Relays are one way to build an interface between the processing and actuation mechanisms of automation systems, and between the embedded system and its surrounding environment. In the previous projects, the reader was introduced to the *sensing* function of embedded systems by making use of a few sensors and learning about their integration with the processing or *thinking* function. In this project, the reader learned how to integrate the *thinking* with the *actuation* function. With the lessons learned in Project 2 and 3, the reader should be able to build a simple thermostat and integrate the *sensing*, *thinking*, and *actuation* functions in a very simple application.

0.5 Project 4: Flex Sensor and Servomotor

Now that the previous projects have provided the reader with the ability to acquire data from sensors and to produce actions based on code, Project 4 combines makes use of a flex sensor and a servomotor to combine the sensing, thinking, and actuation functions in a single project.

0.5.1 Learning Objectives

The main objective of this project is to learn how to operate an analog output driven by a pulse-width modulated (PWM) signal. The secondary objectives are to learn about the flex sensor and the operation of servomotors.

The PWM technique produces the effect of a analog output (Fig 13(a), red) by changing the width (sometimes also the polarity) of a train of pulses such as the one shown in blue in Figure 13(a). The train of pulses (Fig 13(b), pink) can be obtained by modulating the duty cycle of triangular or sawtooth signal, such as in Figure 13(b), blue) making use of a carrier or modulating signal, shown in green.

0.5.2 Implementation

For the implementation of this project, the reader will build the system from a circuit diagram composed of two separate parts. One part will constitute the *sensing* function of the system, while the other will make up the *actuation* function. The thinking function, as should be clear by know, will be carried out by the processing of signals in the ATmega328 micro-controller of the Arduino Uno board.

Materials

The materials needed for this project are the following:

- Flex sensor, x1.

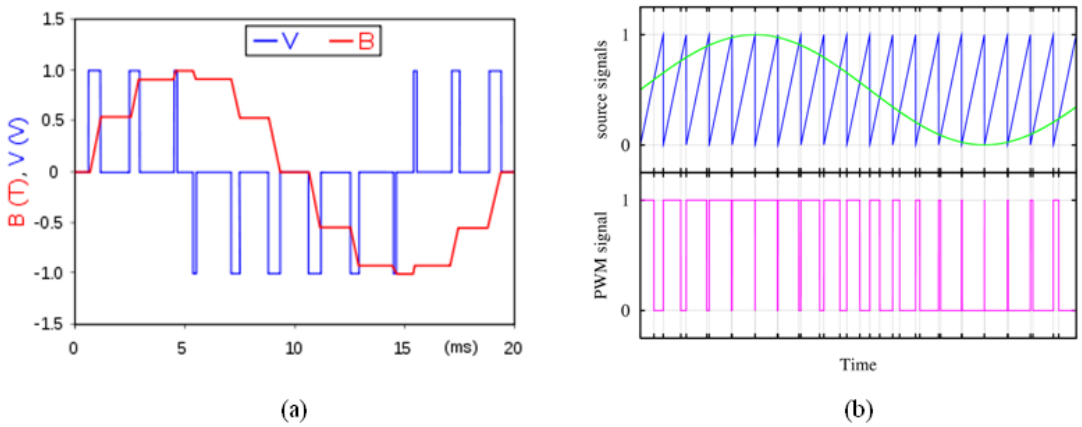

Figure 13: Descriptive plots of the pulse-width modulation technique.

- Servomotor, x1.
- 10 KΩ Resistor, x1.
- Wires, x11.

Circuit Diagram

The circuit diagram for this project is presented in Figure 14.

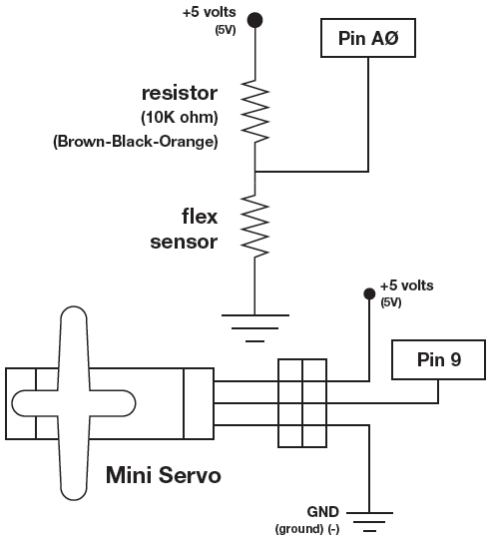

Figure 14: Project 4 Circuit Diagram

Code

The code for this project includes useful comments to help distinguish between the functions employed and to facilitate to learn how to read the flex sensor and to operate the servomotor. Please note that this project requires the use of an external library that is called with the `#include <Servo.h>` at the beginning of the code.

```
// Include the servo library to add servo-control functions:
#include <Servo.h>
```

```
// Create a servo "object", called servo1. Each servo object
// controls one servo (you can have a maximum of 12):
Servo servo1;

// Define the analog input pin to measure flex sensor position:
const int flexpin = 0;

void setup()
{
  // Use the serial monitor window to help debug our sketch:
  Serial.begin(9600);
  // Enable control of a servo on pin 9:
  servo1.attach(9);
}

void loop()
{
  int flexposition;    // Input value from the analog pin.
  int servoposition;   // Output value to the servo.

  // Read the position of the flex sensor (0 to 1023):
  flexposition = analogRead(flexpin);

  // Because the voltage divider circuit only returns a portion
  // of the 0-1023 range of analogRead(), we'll map() that range
  // to the servo's range of 0 to 180 degrees. The flex sensors
  // we use are usually in the 600-900 range:

  servoposition = map(flexposition, 600, 900, 0, 180);
  servoposition = constrain(servoposition, 0, 180);

  // Now we'll command the servo to move to that position:

  servo1.write(servoposition);

  // Because every flex sensor has a slightly different resistance,
  // the 600-900 range may not exactly cover the flex sensor's
  // output. To help tune our program, we'll use the serial port to
  // print out our values to the serial monitor window:

  Serial.print("sensor: ");
  Serial.print(flexposition);
  Serial.print("  servo: ");
  Serial.println(servoposition);
  delay(20);  // wait 20ms between servo updates
}
```

0.5.3 Summary

At the conclusion of this project, the reader should have learned about the PWM technique and the use of a flex sensor and a servomotor. These three tools can each one of them be useful in projects involving robotic manipulators, tactile sensors, and haptic systems. You can note that the sensing circuit makes use of a voltage divider, just like in Project 1. In this case, the resistance of the flex sensor changes with how much it is bent. As the flex sensor bends more, the resistance increases. As in Project 1, the change in resistance is used in Project 4 to read

and actuate in response to environmental stimuli. By now, the reader should feel comfortable in undertaking more complex projects making use of the Arduino Uno board.

Bibliography

[1] Sik guide: Your guide to the sparkfun inventor's kit for arduino (2012).

[2] F. Vahid, T. D. Givargis, Embedded System Design: A Unified Hardware/Software Introduction, Wiley, 2001.