7

Python and Arduino with Pyfirmata

7.1 Python with Arduino

Arduino is an open-source platform to build hardware and software environments. Arduino provides limitless possibilities for tinkerers and electronics enthusiasts.

Raspberry Pi is a full-fledged computer that can do tasks like a desktop PC. It provides a platform for coding and designing electronic circuits, from creating a web server to a gaming console for retro gaming.

Arduino does not understand Python, so Firmata and Pyfirmata protocols are used to communicate through Raspberry Pi using Python. Pyfirmata is a protocol for Raspberry Pi to access Arduino. Firmata is protocol for Arduino to interface with Raspberry Pi with Python. The program will be written on Raspberry Pi in Python to access sensors connected to Arduino.

To install Firmata to Arduino, connect it to a USB socket of Raspberry Pi to communicate and power up Arduino. Next, install Firmata sketch to the Arduino in order for this open an Arduino IDE. Find the Firmata sketch in *File→Examples→Firmata→StandardFirmata* and upload it to the Arduino board. Once Firmata is installed, Arduino waits for communication from Raspberry Pi.

The next step is to install Pyfirmata to Raspberry Pi. For this, just run the following terminal commands on Raspberry Pi:

$ sudo apt-get install git

$ sudo git clone https://github.com/tino/pyFirmata.git

$ cdpyFirmata

$ sudo python setup.py install'

7.2 Controlling Arduino with Python

A "USB standard A" connector is used to connect Arduino with the Raspberry Pi. Now check for the USB address of Arduino by running *"ls -lrt /dev/tty*."* On my Raspberry Pi, it is listed as */dev/ttyUSB0* (Remember this value for later).

Import the Arduino and util classes from the Pyfirmata module to control an Arduino from a Python script on the Raspberry Pi. After this, create an object that was found in the previous step with the help of a USB address.

```
>>>from pyfirmata import Arduino, util
>>>board = Arduino('/dev/ttyUSB0')
```

7.3 Play with LED

The objective of the project is to control the Arduino digital output through Raspberry Pi with Python. To build this project, connect an LED to a digital pin of Arduino and write a short Python program to make it blink. Figure 7.1 shows the circuit diagram for the interfacing of an LED. The system is comprised of a Raspberry Pi3, an Arduino Uno, a power supply, and two LEDs connected to Pin6 and Pin7 of Arduino. The program is written to make LEDs blink after some time delay.

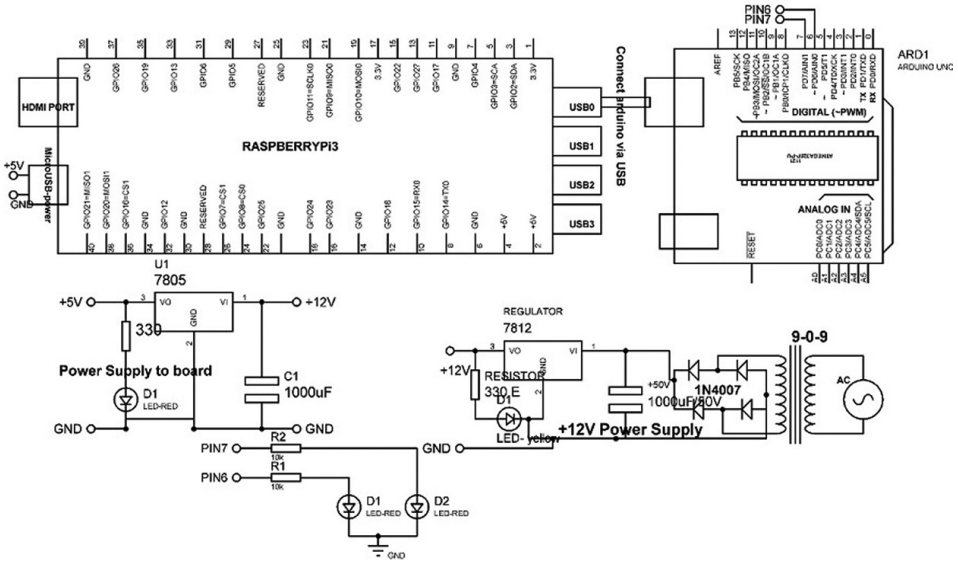

FIGURE 7.1
Circuit diagram for the interfacing of LED.

7.3.1 Recipe

```
import pyfirmata  # import lib of pyfirmata
import time as wait     # import lib of pyfirmata
board = pyfirmata.Arduino('/dev/ttyUSB0')# define COM port of Arduino
red_pin = board.get_pin('d:7:o')# assign digital pin 7 as an output
green_pin = board.get_pin('d:6:o')# assign digital pin 6 as an output

whileTrue:      # infinite loop
        red_pin.write(1)# write '1' on pin 7
        green_pin.write(1)# write '1' on pin 6
        wait.sleep(0.5)# delay of 0.5 Sec
        red_pin.write(0)#write '0' on pin 7
        green_pin.write(0)# write '0' on pin 6
        wait.sleep(0.5)# delay of 0.5 Sec
```

7.4 Reading an Arduino Digital Input with Pyfirmata

The objective is to read the digital pins of Arduino on a Raspberry Pi by Python. Pyfirmata is used to read a digital input on Arduino. The components required for the recipe are Arduino Uno, 1 kΩ resistor, and a push switch or button (as digital sensor). A switch can be connected in two arrangements: pull down and pull up. The output of a digital pin of Arduino is normally "LOW," and digital sensors are available in two configurations for output: active "LOW" and active "HIGH." The pull-down arrangement is used where digital pin is normally "LOW," and on reading the sensor it gets "HIGH." This is used for the sensor that has the output as active "HIGH" on occurrence of an event; otherwise, the output is "LOW." The pull-up arrangement is for the sensor that has a normal output as active "HIGH," and on occurrence of an event it gets "LOW." In this arrangement, the digital pin needs to be activated as "HIGH" in the program so that it can read the sensor. Figure 7.2 shows circuit diagram for pull down, and Figure 7.3 shows circuit diagram for pull up.

As discussed in Section II of this book, the Pyfirmata protocol is used to read the input pin of Arduino by Raspberry Pi. It uses the concept of an iterator to monitor the Arduino pin. The iterator manages the reading of the switch using the following commands:

```
it = pyfirmata.util. Iterator(board)
it.start()
```

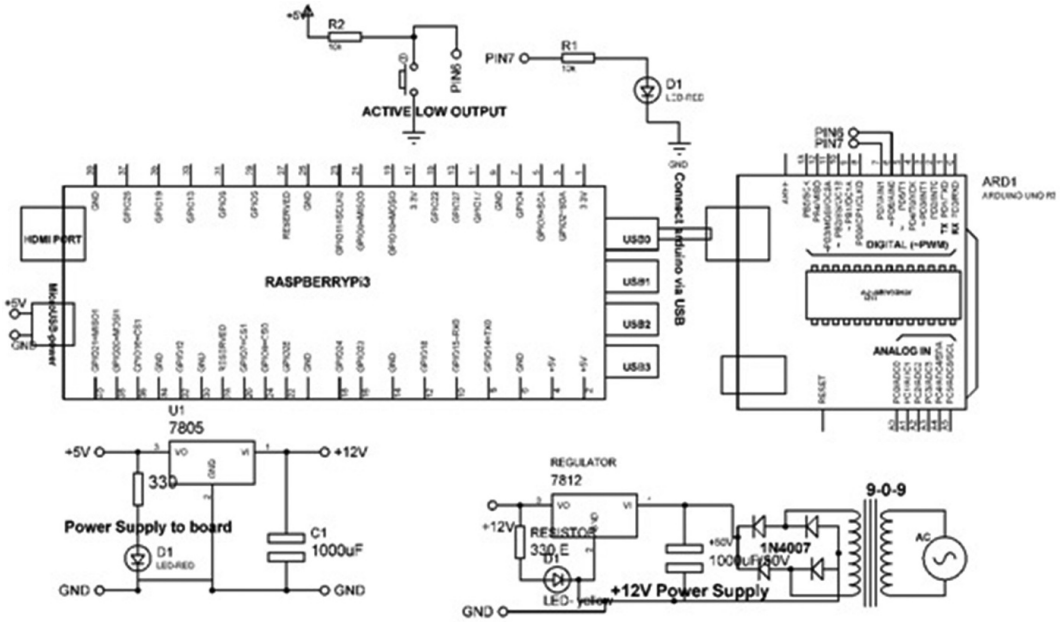

FIGURE 7.2
Pull-down arrangement for reading button.

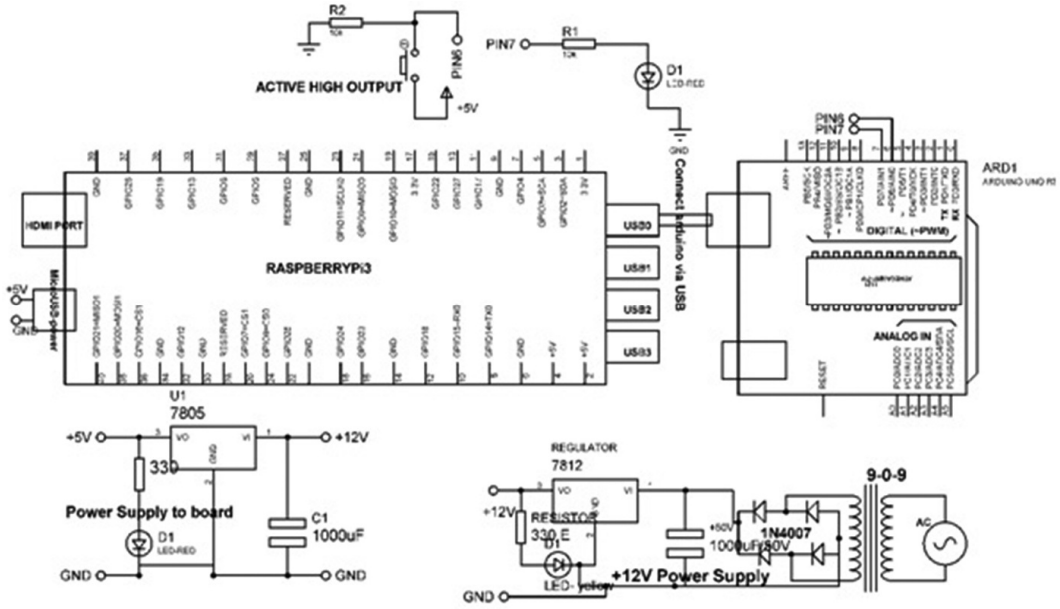

FIGURE 7.3
Pull-up arrangement for reading button.

After this enable the pin by using following command.

switch_pin.enable_reporting()

The iterator function can't be stopped, so when Ctrl+Z is pressed to exit the window, it will not exist.

To stop this function, simply disconnect Arduino from Raspberry Pi or open another terminal window and use the kill command:

$ sudokillall python

7.4.1 Recipe to Read Pull-Down Arrangement

```
import pyfirmata # import library of pyfirmata
import time as wait # import library of time
board = pyfirmata.Arduino('/dev/ttyUSB0') # define COM port of Arduino
button_pin = board.get_pin('d:6:i') # define pin 6 as an input
led_pin = board.get_pin('d:7:o') # define pin7 as an output
it = pyfirmata.util.Iterator(board) # use iterator
it.start() # start iterator
button_pin.enable_reporting() # enable input
while True: # infinite loop
        switch_state = switch_pin.read() # read input from pin 6
        ifswitch_state == False: # check condition
                print('Button Pressed') # print string on Pi terminal
                led_pin.write(1) # write '1' on pin 7
                wait.sleep(0.2) # delay of 0.2 Sec
else
                print('Button not Pressed') # print string on Pi terminal
                led_pin.write(0) # write '0' on pin 7
                wait.sleep(0.2) # delay of 0.2 Sec
```

7.4.2 Recipe to Read Pull-Up Arrangement

```
import pyfirmata # import library of pyfirmata
import time as wait # import library of time
board = pyfirmata.Arduino('/dev/ttyUSB0') # define COM port of Arduino
button_pin = board.get_pin('d:6:i') # assign pin 6 as digital input
led_pin = board.get_pin('d:7:o') # assign pin 7 as digital output
it = pyfirmata.util.Iterator(board) # use iterator
it.start() # start iterator
button_pin.enable_reporting() # enable pin
while True: # infinite loop
        switch_state = switch_pin.read() # read digital pin
        ifswitch_state == True: # check condition
                print('Button Pressed') # print string on Pi terminal
```

```
                led_pin.write(1) # make pin 7 to '1'
                wait.sleep(0.2) # delay of 0.2 Sec
    else
                print('Button not Pressed') # print string on Pi terminal
                led_pin.write(0) # make pin 7 to '1'
                wait.sleep(0.2) # delay of 0.2 Sec
```

7.5 Reading the Flame Sensor with Pyfirmata

The objective is to read the flame sensor as input with Python on a Raspberry Pi. A flame sensor can detect the infrared light with a wavelength ranging from 700 to 1000 nm. The far-infrared flame probe converts the detected light in the form of infrared light into current. It has a working voltage of 3.3 to 5.2 V DC, with a digital output to indicate the presence of a signal. An onboard LM393 comparator is used for condition sensing. Connect the components as shown in Figure 7.4 and check the workings by uploading the recipe described in Section 7.5.1.

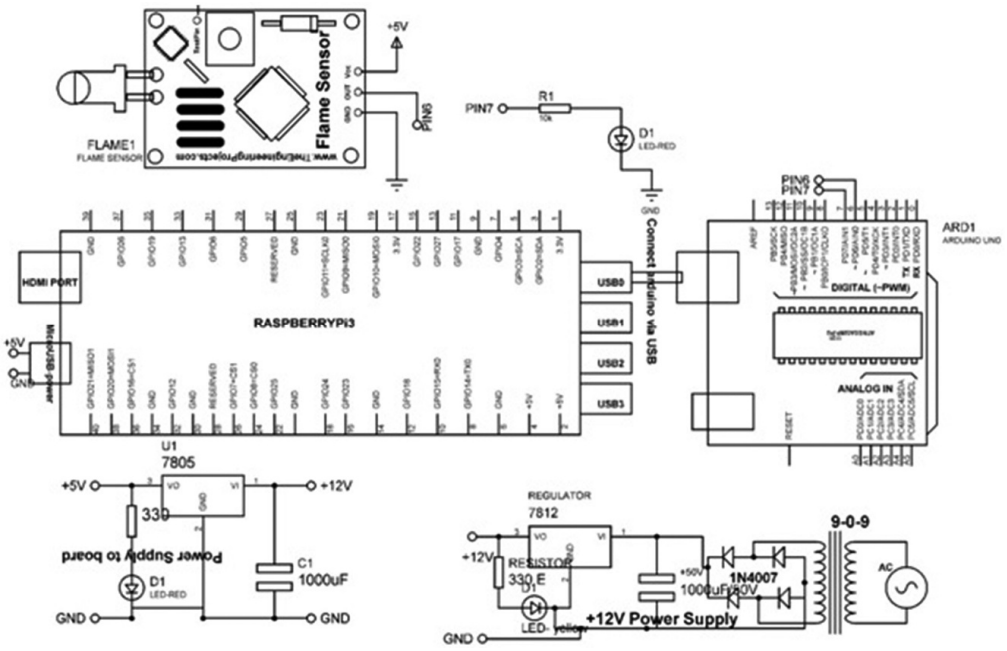

FIGURE 7.4
Circuit diagram for flame sensor interfacing.

7.5.1 Program for Reading Active "Low" Flame Sensor

```
import pyfirmata # import library of pyfirmata
import time as wait # import library of time
board = pyfirmata.Arduino('/dev/ttyUSB0') # define COM port of Arduino
flame_pin = board.get_pin('d:6:i') # assign pin 6 as digital input
indicator_pin = board.get_pin('d:7:o') # assign pin 7 as digital output
it = pyfirmata.util.Iterator(board) # use iterator
it.start() # start iterator
flame_pin.enable_reporting() # enable input
while True: # infinite loop
        flame_state = flame_pin.read() # read digital input
        ifflame_state == False: # check condition
                print('No Obstacle') # print string on Pi Terminal
                indicator_pin.write(1) # write '1'on pin7
                wait.sleep(0.2) # sleep for 0.2 sec
    else:
                print("Obstacle Found")) # print string on Pi Terminal
                indicator_pin.write(0) # write '0' on pin7
                wait.sleep(0.2) # sleep for 0.2 sec
```

7.6 Reading an Analog Input with Pyfirmata

A potentiometer is used to demonstrate the workings of the analog sensor with Pyfirmata. It is connected to pin A0 of Arduino (Figure 7.5). When the pin gets configured as an analog input pin in a program, it starts sending the input values to the serial port. If the data can't be managed properly, the data starts getting buffered at the serial port and quickly overflows; this situation can be handled with the program.

The Pyfirmata library has the reporting and iterator methods to overcome this situation. The *enable_reporting()* method is used to set the input pin to start reporting. This method is applied before performing a reading operation on the pin:

board.analog[3].enable_reporting()

Once the reading operation is done, the pin is set to disable reporting:

board.analog[3].disable_reporting()

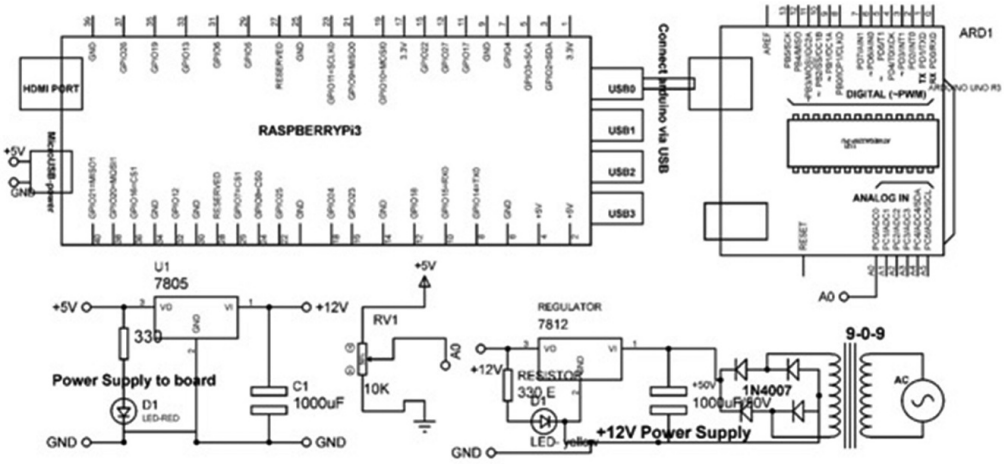

FIGURE 7.5
Circuit diagram for potentiometer interfacing.

To read the analog pin, *iteratorthread* is used in the main loop.

This class is defined in the util module of the Pyfirmata package and is imported before it getting utilized in the code:

```
from pyfirmata import Arduino, util
# Setting up the Arduino board
port = 'COM3'
board = Arduino(port)
sleep(5)
it = util.Iterator(board) # Start Iterator to avoid serial overflow
it.start()
board.analog[3].enable_reporting()
```

7.6.1 Recipe

```
import pyfirmata # import library of pyfirmata
import time as wait # import library of time
board = pyfirmata.Arduino('/dev/ttyUSB0') # define COM port of
    Arduino
POT_pin = board.get_pin('a:0:i') # assign A0 pin as an input
it = pyfirmata.util.Iterator(board) # use iterator
it.start() # start iterator
POT_pin.enable_reporting() # enable pin
while True: # infinite loop
POT_reading = POT_pin.read() # read analog pin
```

```
if POT_reading != None: # check condition
        POT_voltage = POT_reading * 5.0 # convert levels to
            voltage
        print("POT_reading=%f\t  POT_voltage=%f"%  (POT_
            reading, POT_voltage))
        # printvalues on Pi terminal
        wait.sleep(1) # sleep for 1 sec
else:
    print("No reading Obtained") # print string on Pi terminal
    wait.sleep(1)# sleep for 1 sec
```

7.7 Reading the Temperature Sensor with Pyfirmata

The LM35 series of temperature sensors has an output voltage linearly proportional to the Centigrade temperature. The LM35 device does not require any calibration or trimming to provide the accuracy of ±¼°C at room temperature and has a sensing range of −55°C to 150°C. The LM35 device draws a 60-µA current from the supply. The LM35 series devices are available in hermetic TO transistor packages, while the LM35C, LM35CA, and LM35D are available in the plastic TO-92 transistor packages. Figure 7.6 shows the circuit diagram of the LM35 interfacing. The output of LM35 is connected to the A0 pin of Arduino.

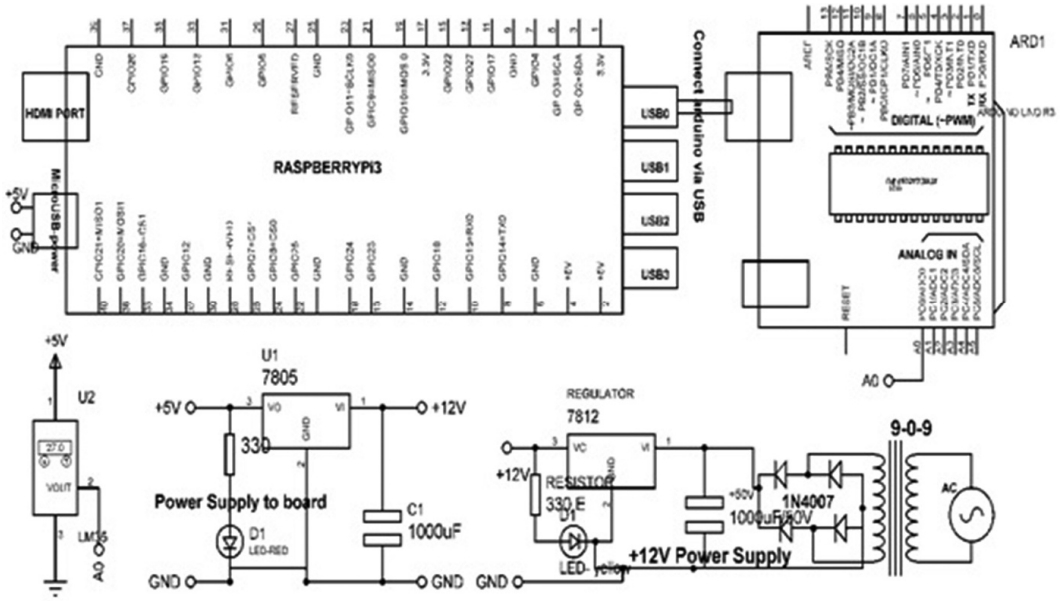

FIGURE 7.6
Circuit diagram of LM35 interfacing.

7.7.1 Recipe

```
IMPORT pyfirmata # import library of pyfirmata
import time as wait # import library of time
board = pyfirmata.Arduino('/dev/ttyUSB0') # define COM port of
   Arduino
POT_pin = board.get_pin('a:0:i') # assign A0 pin as an input
it = pyfirmata.util.Iterator(board) # use iterator
it.start() # start iterator
POT_pin.enable_reporting() # enable pin
while True:
reading = switch_pin.read() # read analog input
if reading != None: # check condition
                voltage = reading * 5.0 # convert level into voltage
                temp = (voltage*1000)/10 # convert voltage into
                   temperature
                print('Reading=%f\t Voltage=%f\tTemperature=%f'%
                   (reading,voltage,temp))
                # print value on Pi Terminal
                wait.sleep(1) # sleep for 1 Sec
else:
                print("No reading Obtained") # print string on Pi Terminal
                wait.sleep(1)# sleep for 1 Sec
```

7.8 Line-Following Robot with Pyfirmata

A line-follower robot follows a visual on the floor or ceiling. Usually, the visual line is black on a white surface, although a white line on black surface is also possible. Line-follower robots are used in the production industries for automated processes. It is one of the most basic robots for beginners. To understand the designing of a robot with Raspberry Pi and Arduino Uno, the system is comprised of a motor driver L293D, two DC motors, a free wheel (to be connected in the front of the robot), two IR sensors, and a power supply.

Connections:

- Connect pins (IN1, IN2, IN3, IN4) of L293D to pins (5, 4, 3, 2) of Arduino Uno, respectively.
- Connect a DC motor (M1) between pins (OUT1 and OUT2) of L293D.

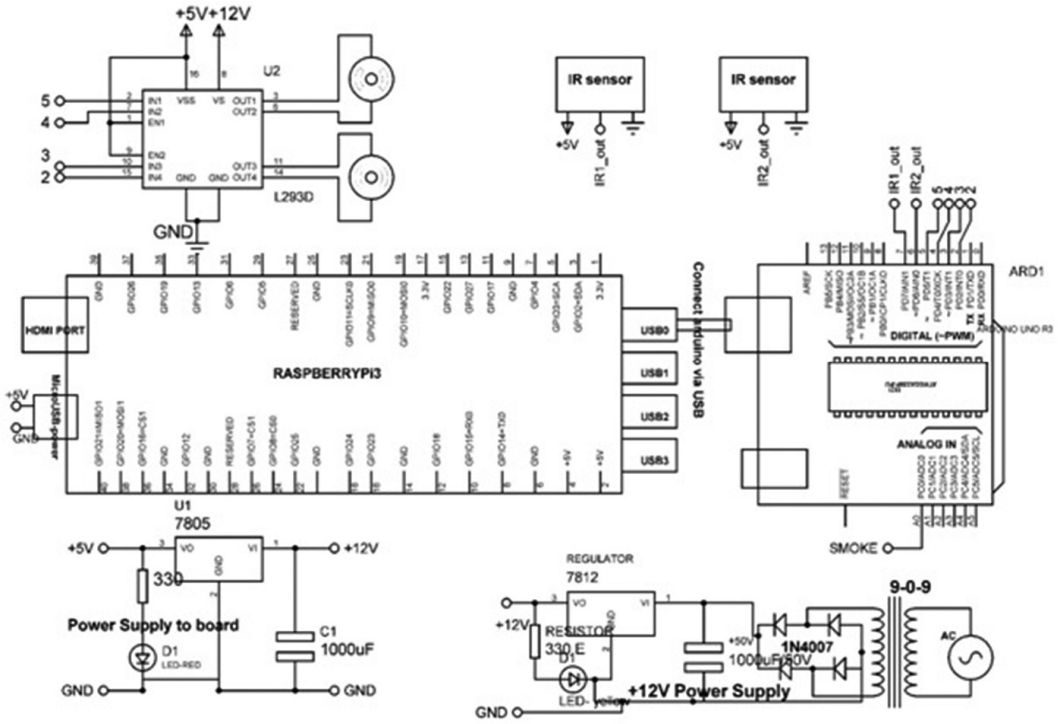

FIGURE 7.7
Circuit diagram for line-following robot.

- Connect other DC motor (M2) between pins (OUT3 and OUT4) of L293D.
- Connect pins (Vcc and ground) of IR1 and IR2 to +5 VDC and ground, respectively.
- Connect pin (OUT) of IR1 to pin (7) of Arduino Uno.
- Connect pin (OUT) of IR2 to pin (6) of Arduino Uno.
- Connect Arduino Uni to Raspberry Pi through a USB.

Figure 7.7 shows the circuit diagram for a line-following robot.

7.8.1 Recipe

```
import pyfirmata
import time as wait
board = pyfirmata.Arduino('/dev/ttyUSB10')
ir1_pin = board.get_pin('d:7:i') # connect IR sensor1 to pin 7 and used
    as input
ir2_pin = board.get_pin('d:6:i') # connect IR sensor2 to pin 6 and used
    as input
```

```
M11_pin = board.get_pin('d:5:o') # connect first motor pin to 5 and used
    as output
M12_pin = board.get_pin('d:4:o') # connect first motor pin to 4 and used
    as output
M21_pin = board.get_pin('d:3:o') # connect second motor pin to 3 and
    used as output
M22_pin = board.get_pin('d:2:o') # connect second motor pin to 2 and
    used as output
it = pyfirmata.util.Iterator(board) # use iterator
it.start() # start iterator
ir1_pin.enable_reporting() # enable the reporting of IR sensor1
ir2_pin.enable_reporting() # enable the reporting of IR sensor2
while True:
                ir1_state = ir1_pin.read() # read IR sensor 1
                ir2_state = ir2_pin.read() # read IR sensor 2
                if ir1_state == False and ir2_state == False:
                        M11_pin.write(1) # make pin5 to
                            HIGH
                        M12_pin.write(0) # make pin4 to
                            LOW
                        M21_pin.write(1) # make pin3 to
                            HIGH
                    M22_pin.write(0) # make pin2 to LOW
                    print('forward') # print on terminal
                            wait.sleep(0.5) # delay of 500mSec
                elif ir1_state == False and ir2_state == True:
                        M11_pin.write(1) # make pin5 to
                            HIGH
                        M12_pin.write(0) # make pin4 to
                            LOW
                        M21_pin.write(0) # make pin3 to
                            LOW
                        M22_pin.write(0) # make pin2 to
                            LOW
                    print('Left') # print on terminal
                    time.sleep(0.5) # delay of 500mSec
```

```
elif ir1_state == True and ir2_state == False:
        M11_pin.write(0) # make pin5 to LOW
        M12_pin.write(0) # make pin4 to LOW
        M21_pin.write(1) # make pin3 to HIGH
        M22_pin.write(0) # make pin2 to LOW
        print('Right') # print on terminal
        time.sleep(0.5)# delay of 500mSec
elif ir1_state == True and ir2_state == True:
        M11_pin.write(0) # make pin5 to LOW
        M12_pin.write(0) # make pin4 to LOW
        M21_pin.write(0) # make pin3 to LOW
        M22_pin.write(0) # make pin2 to LOW
    print('Stop') # print on terminal
    time.sleep(0.5) # delay of 500mSec
```