# 8

# *Python GUI with Tkinter and Arduino*

## 8.1 Tkinter for GUI Design

The basic features for graphical user interface (GUI) libraries include the ease to install and minimal computations. Tkinter framework satisfies the basic requirements. It is also the default GUI library with the Python installations.

Tkinter interface is a cross-platform Python interface for the Tk GUI toolkit. Tkinter provides a layer on Python while Tk provides the graphical widgets. Tkinter is a cross-platform library that gets deployed as a part of the Python installation packages for major operating systems.

Tkinter is designed with minimal programming efforts for creating graphical applications. To test the version of the Tk toolkit, use the following commands on the Python prompt:

```
>>> import Tkinter
>>> Tkinter._test()
```

An image with the version information will be prompted (Figure 8.1).

If the window (Figure 8.1) is not visible, then reinstall Python.

The Tkinter interface supports various widgets to develop GUI. Table 8.1 describes the few widgets.

## 8.2 LED Blink

To understand the working of LED with Tkinter GUI, a simple circuit is designed. The system is comprised of a Raspberry Pi, an Arduino, an LED, and a power supply. An Arduino is connected to Raspberry Pi through a USB connector. The anode terminal of LED is connected to pin(7) of Arduino through a 10-K resistor, and the cathode terminal is connected to the ground. Figure 8.2 shows the circuit diagram to interface LED.


**FIGURE 8.1**
Tk version.

**TABLE 8.1**

The Tkinter Widgets to Develop GUI

| Widget | Description |
|---|---|
| Tk() | Root widget need in every program |
| Label() | It shows an image or a text |
| Button() | Execute actions |
| Entry() | Text field to provide inputs |
| Scale() | It provides a numeric value by dragging the slider |
| Checkbox() | It enables toggling between two values |

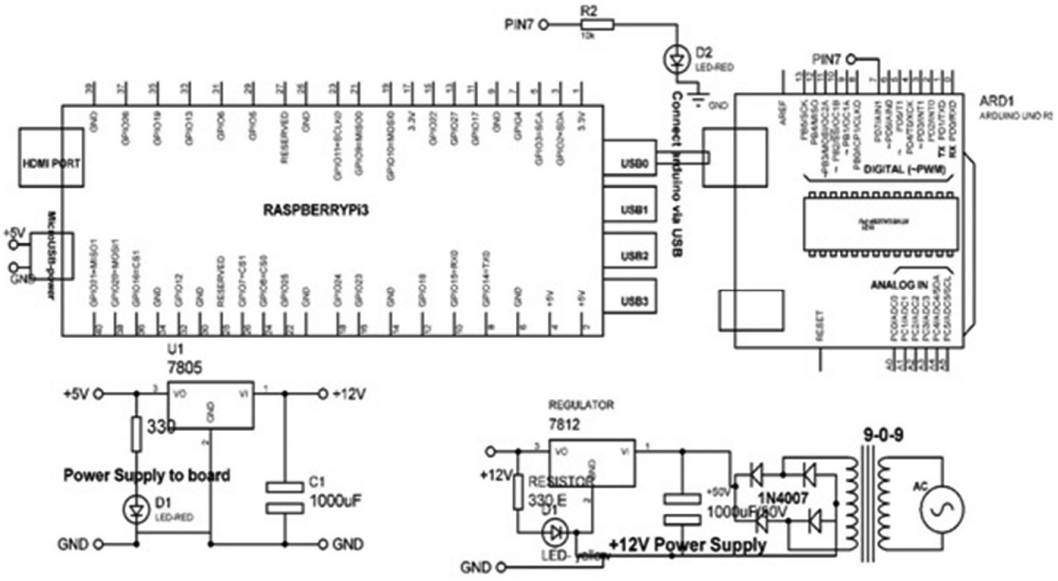


**FIGURE 8.2**
Circuit diagram to interface LED.

### 8.2.1 Recipe for LED Blinking with Fixed Time Delay

```
import Tkinter
import pyfirmata
import time as wait
# Associate port and board with pyfirmata
board = pyfirmata.Arduino('/dev/ttyUSB0')
wait.sleep(5) # delay of 5Sec
led_Pin = board.get_pin('d:7:o') # connect led to pin 7 and used as output
def call_LED_BLINK():
    button.config(state = Tkinter.DISABLED)
    led_Pin.write(1) # make led_Pin to HIGH
    print('LED at pin7 is ON') # print on terminal
wait.sleep(5) # delay of 5 sec
    print('LED at pin 7 is OFF') # print on terminal
    led_Pin.write(0) # make led_Pin to LOW
    button.config(state=Tkinter.ACTIVE)
# Initialize main windows with title and size
TOP = Tkinter.Tk()
TOP.title("Blink LED using button")
TOP.minsize(300,30)
# Create a button on main window and associate it with above method
button = Tkinter.Button(TOP, text="Press start to blink", command =
  call_LED_BLINK)
button.pack()
TOP.mainloop()
```

#### *8.2.1.1 Tkinter GUI for LED Blinking with Fixed Delay*

Run the program described in Section 8.2.1, and a GUI will appear (Figure 8.3).

After pressing the "Press start to blink" button on GUI, it will print "LED at pin7 is ON" for 5 sec and then print "LED at pin7 is OFF" (Figure 8.4).

### 8.2.2 Recipe for LED Blinking with Variable Delay

```
import Tkinter
import pyfirmata
```


**FIGURE 8.3**
Tkinter GUI to control LED with fixed delay.

```
pi@raspberrypi:~ $ nano pyfir_led_tkinter_BOOK.py
pi@raspberrypi:~ $ python pyfir_led_tkinter_BOOK.py
LED at pin7 is ON
LED at pin 7 is OFF
```

**FIGURE 8.4**
LED ON/OFF.

```
import time as wait
board = pyfirmata.Arduino('/dev/ttyUSB0')
wait.sleep(5) # delay of 5 sec
led_Pin = board.get_pin('d:7:o') # connect led to pin 7 and used as output
def led_blink_variable_delay():
    # Value for delay is obtained from the Entry widget input
    time_Period = time_Period_Entry.get()
    time_Period = float(time_Period)
    button.config(state=Tkinter.DISABLED)
    led_Pin.write(1) # make led_Pin to HIGH
    print('pin7 connected led is ON') # print on terminal
wait.sleep(time_Period) # delay of 5 sec
    print('pin7 connected led is off') # print on terminal
    led_Pin.write(0) # make led_Pin to LOW
    button.config(state=Tkinter.ACTIVE)
TOP = Tkinter.Tk()
TOP.title("enter variable time")
time_Period_Entry = Tkinter.Entry(TOP, bd=6, width=28)
time_Period_Entry.pack()
time_Period_Entry.focus_set()
button = Tkinter.Button(TOP, text="start to blink", command=led_
  blink_variable_delay)
button.pack()
TOP.mainloop()
```


**FIGURE 8.5**
Tkinter GUI to control LED with variable delay.

```
pi@raspberrypi:~ $ nano pyfir_led_tkinter_variable_time_BOOK.py
pi@raspberrypi:~ $ python pyfir_led_tkinter_variable_time_BOOK.py
pin7 connected led is ON
pin7 connected led is off
```

**FIGURE 8.6**
Screenshot for LED ON/OFF with delay.

#### *8.2.2.1 Tkinter GUI for LED Blinking with Variable Delay*

Run the program described in Section 8.2.2, and a GUI will appear (Figure 8.5).

Write down the required delay in the blank slot, for example, take value 5. After pressing the "start to blink" button and delay of 5 sec on GUI, it will print "Pin7 connected is ON" for 5 sec and then print "Pin7 connected is OFF" (Figure 8.6).

## 8.3 LED Brightness Control

Figure 8.3 shows the circuit diagram for LED interfacing where LED is connected to pin (7) of Arduino. Pin(7) is also a PWM pin in Arduino Uno, so with the same circuit in Figure 8.3, LED brightness can be controlled with a different program.

### 8.3.1 Recipe

```
import Tkinter # add Tkinter library
import pyfirmata # add pyfirmata library
import time as wait # add time library
board = pyfirmata.Arduino('/dev/ttyUSB0')
wait.sleep(5) # delay of 5 Sec
led_Pin = board.get_pin('d:7:o') # connect led to pin 7 and used as output
def call_led_blink_pwm():
    time_Period = time_Period_Entry.get()
```

```
        time_Period = float(time_Period)
        led_Brightness = brightness_Scale.get()
        led_Brightness = float(led_Brightness)
        button.config(state=Tkinter.DISABLED)
        led_Pin.write(led_Brightness/100.0)
        print 'LED brightness control' # print on terminal
    wait.sleep(time_Period)
         led_Pin.write(0) # make led_Pin to LOW
         button.config(state=Tkinter.ACTIVE)

    TOP = Tkinter.Tk()
    time_Period_Entry = Tkinter.Entry(TOP, bd=7, width=30)
    time_Period_Entry.pack()
    time_Period_Entry.focus_set()
    brightness_Scale = Tkinter.Scale(TOP, from_=0, to=100, orient=Tkinter.
       VERTICAL)
    brightness_Scale.pack()
    button = Tkinter.Button(TOP, text="Start", command =
       call_led_blink_pwm)
    button.pack()
    TOP.mainloop()
```

### 8.3.2 Tkinter GUI for LED Brightness Control

Run the program described in Section 8.3.1. Brightness can be controlled with a scale () widget. Once the LED is turned off after the time delay, the slider can be reset to another position to dynamically vary the value for the brightness. The slider can be placed horizontally instead of on a vertical scale. By clicking on "Start," it will display the message "LED brightness control" (Figures 8.7 and 8.8).

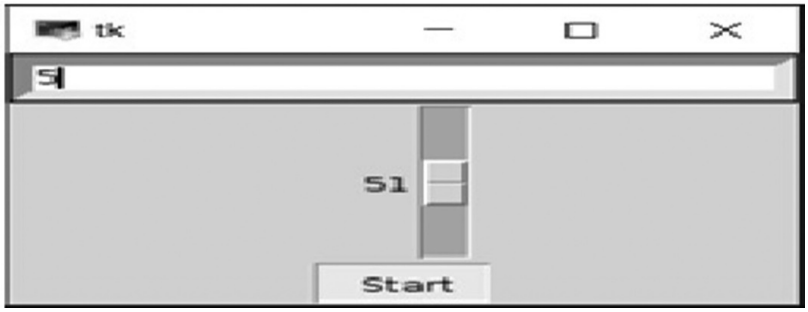


**FIGURE 8.7**
Tkinter GUI for LED brigtness control.

```
pi@raspberrypi:~ $ nano pyfir_led_pwm_tkinter_BOOK.py
pi@raspberrypi:~ $ python pyfir_led_pwm_tkinter_BOOK.py
LED brightness control
```

**FIGURE 8.8**
Screenshot for LED brightness control.

## 8.4 Selection from Multiple Options

When the user needs to select from multiple options from the given set of values, the complexity of the project increases. For example, when multiple numbers of LEDs are interfaced with the Arduino board, the user needs to select an LED or LEDs to turn it on. The Tkinter library provides an interface for a widget called Checkbutton(), which enables the selection process from the given options. The concept can be understood with a simple circuit of interfacing two LEDs with Arduino and Raspberry Pi using Pyfirmata.

The system is comprised of a Raspberry Pi, an Arduino Uno, two LEDs, and a power supply. Arduino is connected to Raspberry Pi through a USB. The anode terminals of LED1 and LED2 are connected to pin(6) and pin(7), respectively, through a 10-K resistor each, and cathode terminals of both LEDs are connected to ground (Figure 8.9).

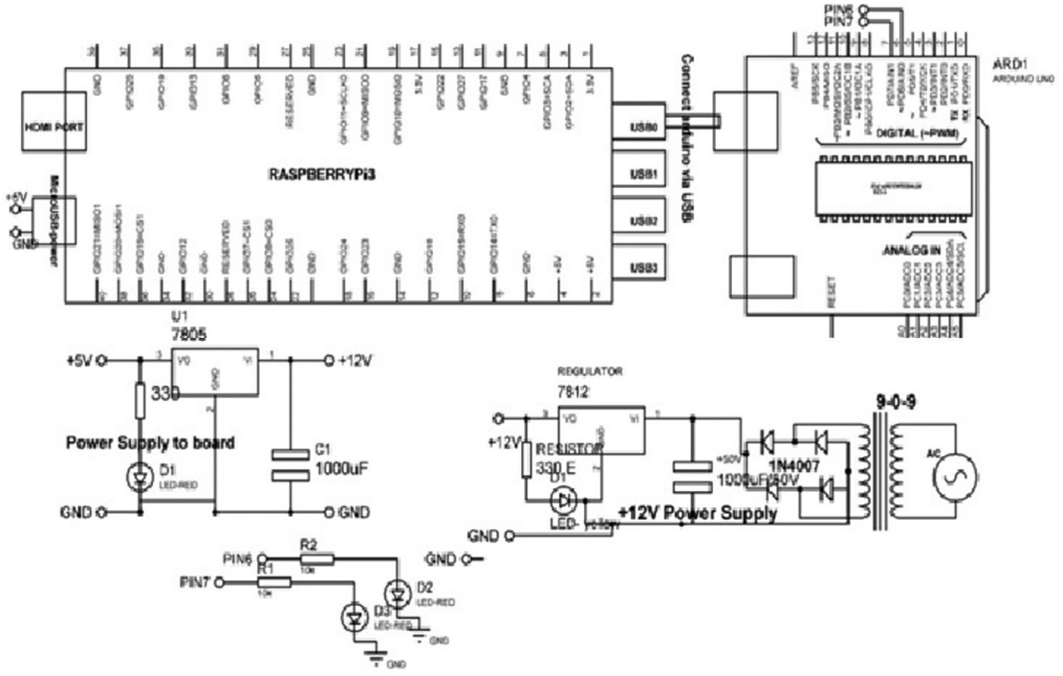


**FIGURE 8.9**
Circuit diagram to interface multiple LEDs.

### 8.4.1 Recipe

```
import Tkinter # add Tkinter library
import pyfirmata # add pyfirmata library
import time as wait # add time library
board = pyfirmata.Arduino('/dev/ttyUSB0')
wait.sleep(5) # delay of 5 Sec
red_led_pin = board.get_pin('d:7:o') # connect led to pin 7 and used as
    output
green_led_pin = board.get_pin('d:6:o') # connect led to pin 6 and used
    as output
def start_button_press():
        red_led_pin.write(red_led_Var.get())
        green_led_pin.write(green_led_Var.get())
        print 'start...' # print on terminal
def stop_button_press():
        red_led_pin.write(0) # make pin 7 to HIGH
        green_led_pin.write(0) # make pin 6 to HIGH
        print 'stop....' # print on terminal
TOP = Tkinter.Tk()
red_led_Var = Tkinter.IntVar()
red_CheckBox = Tkinter.Checkbutton(TOP, text="Red_LED_status",
    variable=red_led_Var)
red_CheckBox.grid(column=1, row=1)
green_led_Var = Tkinter.IntVar()
green_CheckBox = Tkinter.Checkbutton(TOP, text="Green_LED_status",
    variable=green_led_Var)
green_CheckBox.grid(column=2, row=1)
start_Button = Tkinter.Button(TOP, text="Start_button", command =
    start_button_press)
start_Button.grid(column=1, row=2)
stop_Button = Tkinter.Button(TOP, text="Stop_button", command =
    stop_button_press)
stop_Button.grid(column=2, row=2)
exit_Button = Tkinter.Button(TOP, text="Exit_button", command=TOP.
    quit)
exit_Button.grid(column=3, row=2)
TOP.mainloop()
```

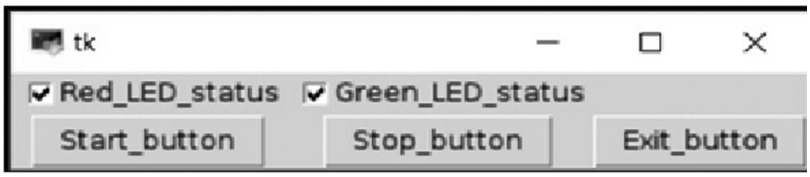


**FIGURE 8.10**
Tkinter GUI for multiple LEDs.

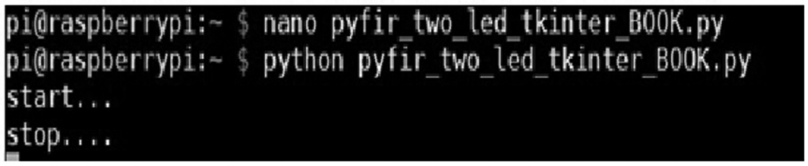


**FIGURE 8.11**
Screenshot for status window.

### 8.4.2 Tkinter GUI

To generate Tkinter GUI for multiple LEDs, run the program described in Section 8.4.1, and a window will appear (Figure 8.10). The GUI has two Checkbutton() widgets each for the red and green LED. The user can select the LEDs individually or together to make it start or stop (Figure 8.11).

## 8.5 Reading a PIR Sensor

A pyroelectric infrared (PIR) sensor is used to detect motion. A system is designed with a PIR sensor where two LEDs show the status of motion. If no motion is detected, the green LED will glow, and if motion is detected, an alert is generated with making the red LED "ON." The system is made ON/OFF with Tkinter GUI. The status of sensor prints on the Python prompt. The system is comprised of a Raspberry Pi, an Arduino, two LEDs, a PIR sensor, and a power supply. Arduino is connected to Raspberry Pi through a USB. The anode terminal of LED1 and LED2 are connected to pin (7) and pin(6) of Arduino, respectively. The pin (OUT) of the PIR sensor is connected to pin (8) of Arduino, and pin (Vcc) and pin (GND) are connected to +5 V DC and ground, respectively (Figure 8.12).

### 8.5.1 Recipe to Read PIR Sensor

```
# Define custom function to perform Blink action
def blink_LED(pin, message):
```

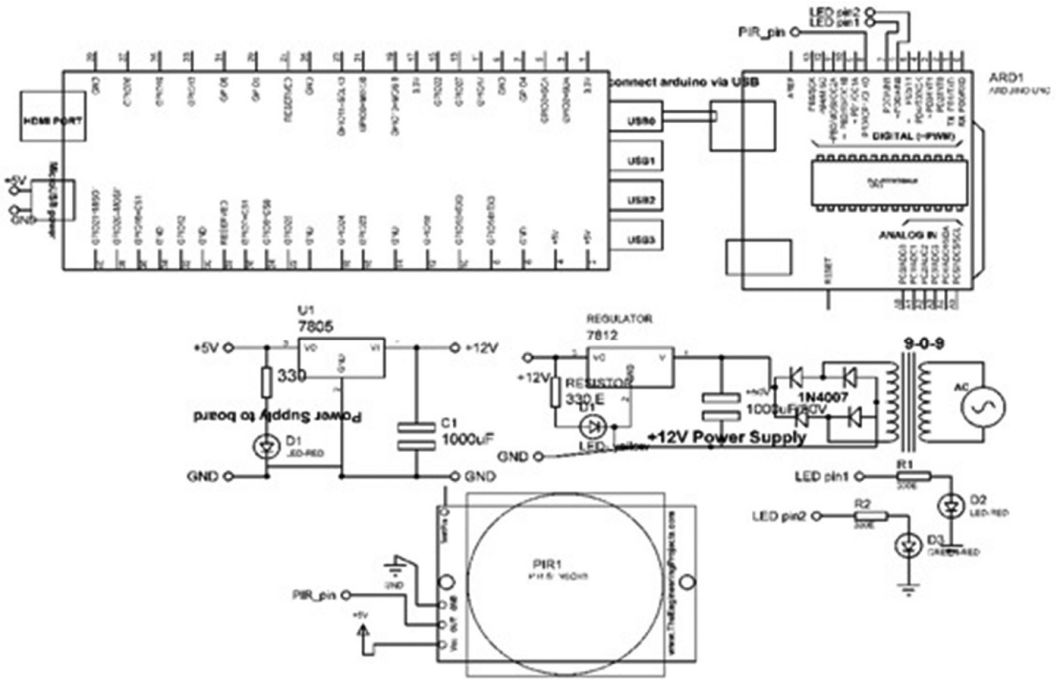


**FIGURE 8.12**
Circuit diagram for PIR sensor interfacing.

```
    Motion_Label.config(text=message)
    Motion_Label.update_idletasks()
    TOP.update()
    pin.write(1) # make pin to HIGH
  wait.sleep(1) # delay of 1 Sec
    pin.write(0) # make pin to HIGH
  wait.sleep(1) # delay of 1 Sec
  # Define the action associated with Start button press
  def press_start_button():
    while True:
      if FLAG.get():
        if PIR_pin.read() is True:
          blink_LED(red_led_pin, "motion_status:Y")
          print 'Motion detected' # print on terminal
        else:
          blink_LED(green_led_pin, "motion_status:N")
          print 'No motion' # print on terminal
```

```
        else:
            break
    board.exit()
    TOP.destroy()
def press_exit_button():
    FLAG.set(False)
import Tkinter # import Tkinter library
import pyfirmata # import pyfirmata library
import time as wait # import time library
board = pyfirmata.Arduino('/dev/ttyUSB0')
wait.sleep(5) # wait for 5Sec
PIR_pin = board.get_pin('d:8:i') # connect PIR sensor to pin8 and as
    input
red_led_pin = board.get_pin('d:7:o') # connect red LED to pin7 and as
    output
green_led_pin = board.get_pin('d:6:o') # connect Green led to pin6 and
    as output
# Using iterator thread to avoid buffer overflow
it = pyfirmata.util.Iterator(board)
it.start()                    # start iterator
PIR_pin.read() # read PIR sensor
# Initialize main windows with title and size
TOP = Tkinter.Tk()
TOP.title("PIR_sensor_for_motion")
# Create Label to for motion detection
Motion_Label = Tkinter.Label(TOP, text="Press Start..")
Motion_Label.grid(column=1, row=1)
# Create flag to work with indefinite while loop
FLAG = Tkinter.BooleanVar(TOP)
FLAG.set(True)
# Create Start button and associate it with onStartButtonPress method
Start_Button = Tkinter.Button(TOP , text="Start", command=press _start_
    button)
Start_Button.grid(column=1, row=2)
# Create Stop button and associate it with onStopButtonPress method
```

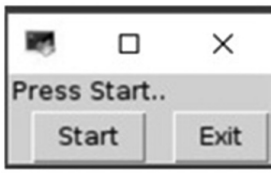


**FIGURE 8.13**
Tkinter GUI for motion detection.

```
pi@raspberrypi:~ $ nano pyfir_digital_in.tkinter_BOOK.py
pi@raspberrypi:~ $ python pyfir_digital_in.tkinter_BOOK.py
No motion
No motion
No motion
No motion
```

**FIGURE 8.14**
Status window for motion sensor.

```
Stop_Button = Tkinter.Button(TOP, text="Exit", command=press_exit_
    button)
Stop_Button.grid(column=2, row=2)
# Start and open the window
TOP.mainloop()
```

### 8.5.2 Tkinter GUI

To generate Tkinter GUI for motion status, run the program described in Section 8.5.1, and a window will appear (Figure 8.13). On pressing the "Start" button, it will give status of the motion sensor as "No motion" or "Motion detected" (Figure 8.14).

## 8.6 Reading an Analog Sensor

A potentiometer (POT) is analogous to an analog sensor. The change in output voltage can be observed on changing the resistance value by moving the knob. To create a Tkinter GUI for a POT, the system is comprised of a Raspberry Pi, an Arduino, a 10-K POT, and a power supply. Arduino is connected to Raspberry Pi through a USB. A POT has three terminals: one

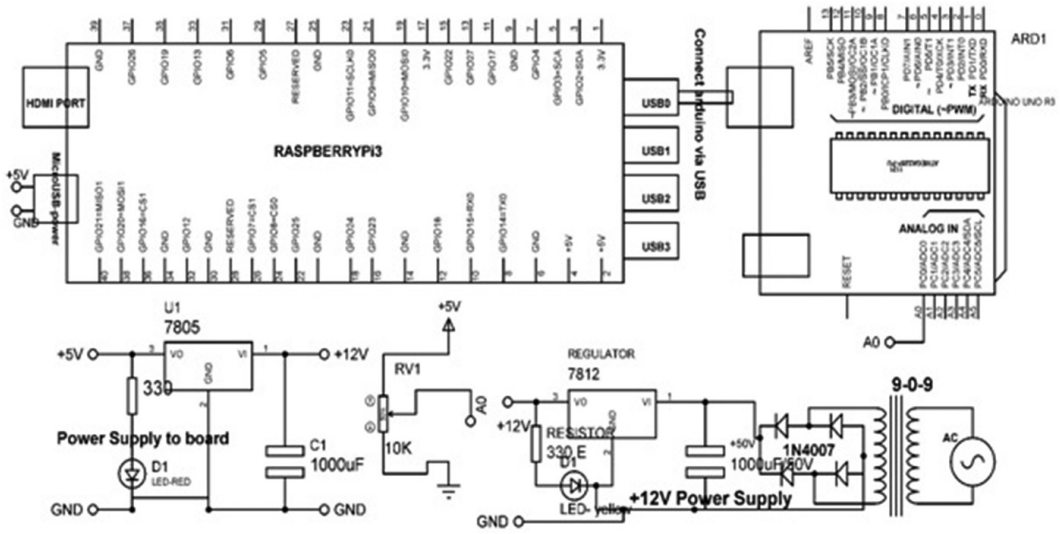


**FIGURE 8.15**
Circuit diagram for interfacing of POT.

terminal is connected to +5 V and other to ground. The wiper of POT is connected to pin (A0) of Arduino (Figure 8.15).

### 8.6.1 Recipe

```
# Define the action associated with Start button press
def start_button_press():
  while True:
    if FLAG.get():
      analog_Read_Label.config(text=str(a0.read()))
      analog_Read_Label.update_idletasks()
      TOP.update()
      print 'analog values of pot'
    else:
      break
  board.exit()
  TOP.destroy()

# Define the action associated with Exit button press
def exit_button_press():
  FLAG.set(False)
```

```
import Tkinter # import Tkinter library
import pyfirmata# import pyfirmata library
import time as wait # import time library
board = pyfirmata.Arduino('/dev/ttyUSB0')
wait.sleep(5)
it = pyfirmata.util.Iterator(board)
it.start() # start iterator
# Assign a role and variable to analog pin 0
a0 = board.get_pin('a:0:i') # connect sensor A0 pin
# Initialize main windows with title and size
TOP = Tkinter.Tk()
TOP.title("Reading POT pins")
# Create Label to read analog input
description_Label = Tkinter.Label(TOP, text="POT_input:- ")
description_Label.grid(column=1, row=1)
# Create Label to read analog input
analog_Read_Label = Tkinter.Label(TOP, text="Press_Start_process")
# Setting flag to toggle read option
FLAG = Tkinter.BooleanVar(TOP)
FLAG.set(True)
# Create Start button and associate with onStartButtonPress method
start_Button = Tkinter.Button(TOP, text="Start_reading",
   command=start_ button_press)
start_Button.grid(column=1, row=2)
# Create Stop button and associate with onStopButtonPress method
exit_Button = Tkinter.Button(TOP, text="Exit_reading",
   command=exit_button_press)
exit_Button.grid(column=2, row=2)
# Start and open the window
TOP.mainloop()
```

### 8.6.2 Tkinter GUI

Run the program described in Section 8.6.1 to create Tkinter GUI for reading a POT, and a window will appear (Figure 8.16). On pressing the "Start_reading" button, it will give readings of POT (Figure 8.17).


**FIGURE 8.16**
Tkinter GUI for reading a POT.

```
pi@raspberrypi:~ $ nano analog_in_tkinter.py
pi@raspberrypi:~ $ python analog_in_tkinter.py
analog values of pot
analog values of pot
analog values of pot
```

**FIGURE 8.17**
Window showing analog values of POT.