# Internet of Things - Working with Raspberry Pi and Windows 10

By Bruno Sonnino (https://msdn.microsoft.com/en-au/magazine/mt149362?author=bruno+sonnino) | May 2017 | Get the Code (http://bit.ly/2lQJfaT)

Although I've been working with software for a long time, I've never interacted directly with hardware. I've developed a lot of software that works near the hardware, but I've never worked with a physical board where I have complete control of what's being done. Therefore, when I had the opportunity to work with the Raspberry Pi, especially using Windows 10 and Visual Studio, I jumped at the opportunity.

The Raspberry Pi, in versions 2 and 3, can use Windows 10 as its OS (though it's not the full version, it lets you execute Universal Windows Platform [UWP] apps to control its devices). This is a cheap computer—you can get one for less than $35—and it's powerful. The Raspberry Pi 3 has a Quad-Core, 64-bit ARM processor, HDMI video, Ethernet and Wi-Fi networking, Bluetooth, and four USB ports. You can definitely do many things with it.

## The Hardware

To start, you can use the Raspberry Pi board alone, but that's somewhat limiting. If you use only the board, that would be the same as developing for a computer or a smartphone. You also need to use some extra hardware. Some manufacturers have created kits to complement it: prototype boards, resistors, LEDs, potentiometers, sensors and a memory card. You can buy a case for the computer, but that's not necessary, as you should keep the computer open to make the connections.

## Knowing the Hardware

Once you have the board and the kit, it's time to get to know the hardware. Initially, you should explore the Raspberry Pi and see what it has to offer. **Figure 1** shows the board.

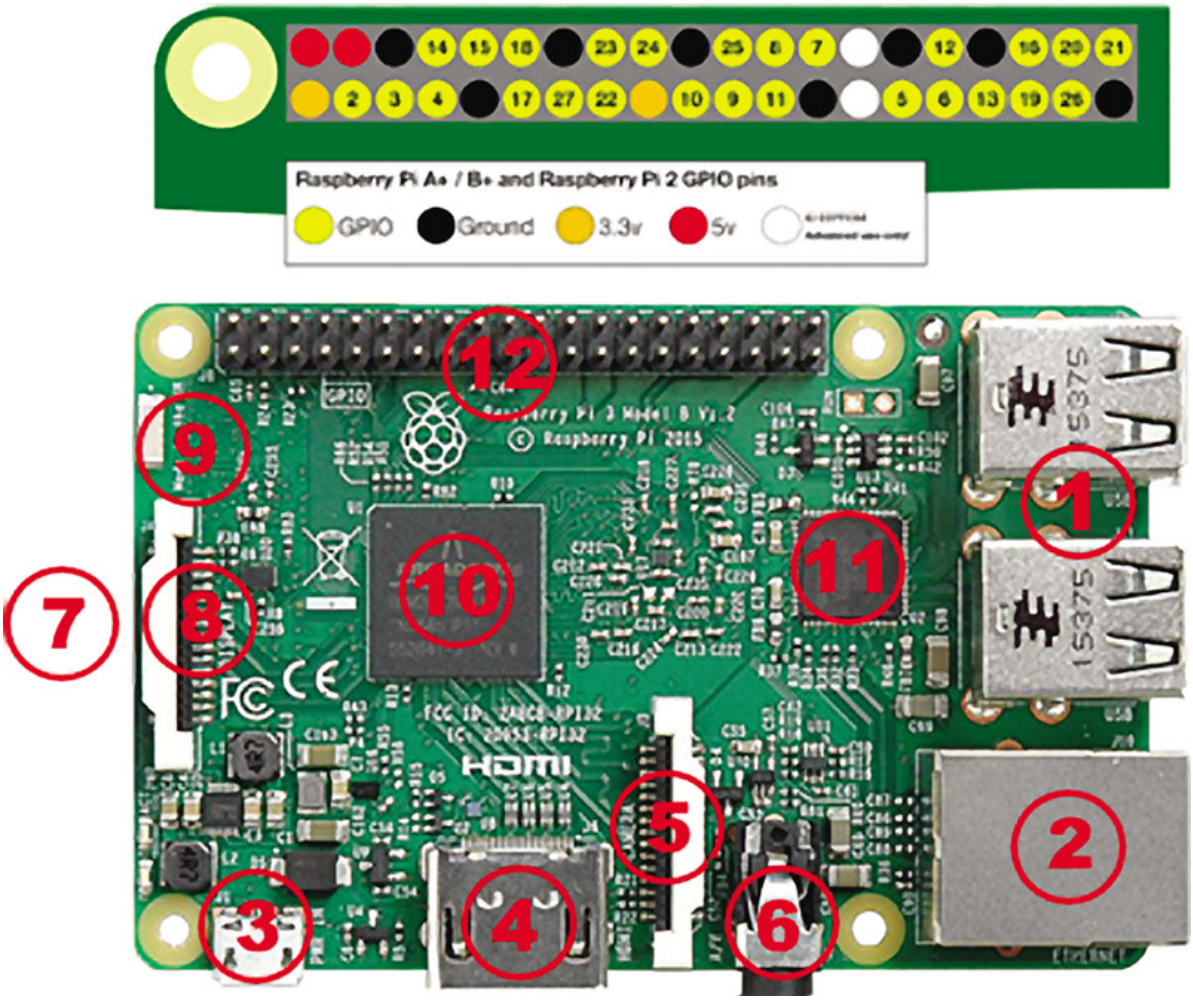


**Figure 1 Raspberry Pi 3 Model B with GPIO**

On the right side of the board in **Figure 1**, you can see the four USB ports (1) and the Ethernet connector (2). At the bottom, from left to right, you have the power jack in the form of a mini USB (3), the HDMI video (4), the camera port (5) and the sound output (6). On the left side of the board, you have the micro SD card slot (7) and a connector for an LCD display (8). You can also see the Wi-Fi and Bluetooth adapter (9). In the middle of the board, you can see the processor (10) and the network controller (11). On the upper side, you have the General Purpose Input/Output (GPIO) block (12), where you make all connections. Every pin has a different purpose, as you can see at the top of the figure.

The Raspberry Pi uses two supply tensions: 5V and 3.3V. The black pins are ground and the yellow ones are the GPIO pins that you'll use in your programming. Note that the pin numbering isn't ordered. Therefore, unless you have a perfect memory, keep a diagram like that nearby (there's one available at bit.ly/1WcBUS2 (http://bit.ly/1WcBUS2)).

The second step is to study the kit. I won't discuss all the content because that can change a lot, depending on the manufacturer (or from what you intend to buy). For this project, you'll need a breadboard, three LEDs, resistors and wires. To learn more about

these components and how to interact with them, see Frank La Vigne's April 2016 Modern Apps column, "Writing UWP Apps for the Internet of Things," at msdn.com/magazine/mt694090 (http://msdn.com/magazine/mt694090).

## Mounting the First Circuit

Knowing the board and these simple components, you can mount the first circuit. Usually, the "Hello World" for a system like that is a program that makes the LED blink. To make it extra simple, you'll start by creating a circuit that lights up the LED, without blinking. For that, you don't need any kind of program, only to understand the circuit you're going to build.

If you connect the LED directly to the 3.3V pin of the Raspberry Pi, you'd probably burn the LED, as it wouldn't support the current passing by it. By using Ohm's law (V = R*I), you need to add a 220 Ω (Red/Red/Black) resistor in the circuit. If you don't have a 220 Ω resistor available, you can use a larger one–with a larger resistor, there's less current in the circuit, so the LED isn't damaged. The resistor can't be much larger because if the current is too small, the LED doesn't turn on. In my case, I used a 330 Ω with no problems.

To see what the montage in the breadboard looks like, see **Figure 2**. The image was created with an open source program called "Fritzing," which can be downloaded at fritzing.org (http://fritzing.org/).

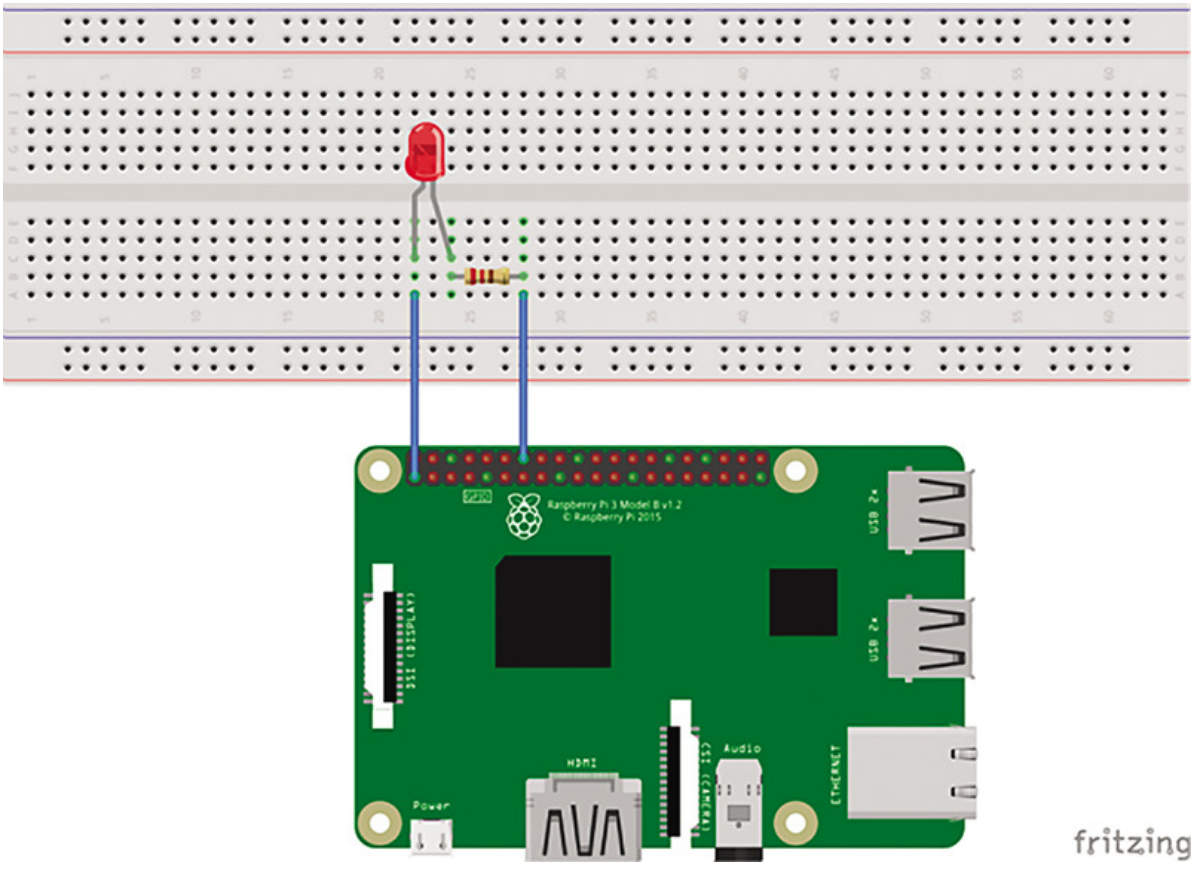


**Figure 2 Mounted Circuit**

After mounting the circuit (this should be done with the Raspberry Pi power source off to not burn any component), connect the power source. If you mounted it correctly, the LED should turn on. If the LED doesn't turn on, check if you put the poles of the LED correctly—the positive pole (longer wire) and the connection to the 3.3V pin in the Raspberry Pi should be in the same horizontal line. The negative pole and the resistor (in this case, there is no polarization) must be in the same line. The second wire of the resistor must be connected to the line that goes to the ground pin in the Raspberry Pi. If everything is right, check if the LED is burned, and replace it with another one. When the LED turns on, you can go to the next step: creating a program that controls the LED.

## Installing and Using Windows 10

Until now, you didn't need an OS because you didn't need to program the Raspberry Pi, but you'll need some programming to go on with your exploration. For that, you'll use Windows 10. You can download and install Windows 10 for the Raspberry Pi free of cost and, although it isn't exactly the same version that runs on desktops and tablets, it lets you execute programs for the UWP with no change.

The first step is to download and install Windows 10 in the SD Card. For that, download and install the Windows 10 Internet of Things (IoT) Core Dashboard tool, located at bit.ly/2lPXrRc (http://bit.ly/2lPXrRc).

To install Windows 10 on the Raspberry Pi, you must have a compatible micro SD card with at least 8GB. Next, select the option "Setup a new device" in the dashboard to download and install Windows 10 on the SD card. You must have some way to write to this card on your computer. If you don't have a card reader available, you can buy a USB card reader.

Select the type of device, the OS and the drive where the SD card is located. Give a name for the computer and select an administrator password. Click on the box to accept the license terms and click on the Install button. After downloading and recording the data on the card, you have Windows 10 installed and ready to use. Remove it from the computer card reader and put it in the slot of the Raspberry Pi. Connect it to the network using an Ethernet cable or Wi-Fi if you're using the Raspberry Pi 3 or 2 with a Wi-Fi dongle. Turn on the device.

Once Windows 10 has booted, you can see the device connected under My Devices in the IoT Core Dashboard.

You can open the device portal in the browser using the IP Address shown for the connected device, on port 8080. In my case, I can open it with the address http://192.168.1.199:8080. It will ask you the admin password you've set before to open the portal, as shown in **Figure 3**.

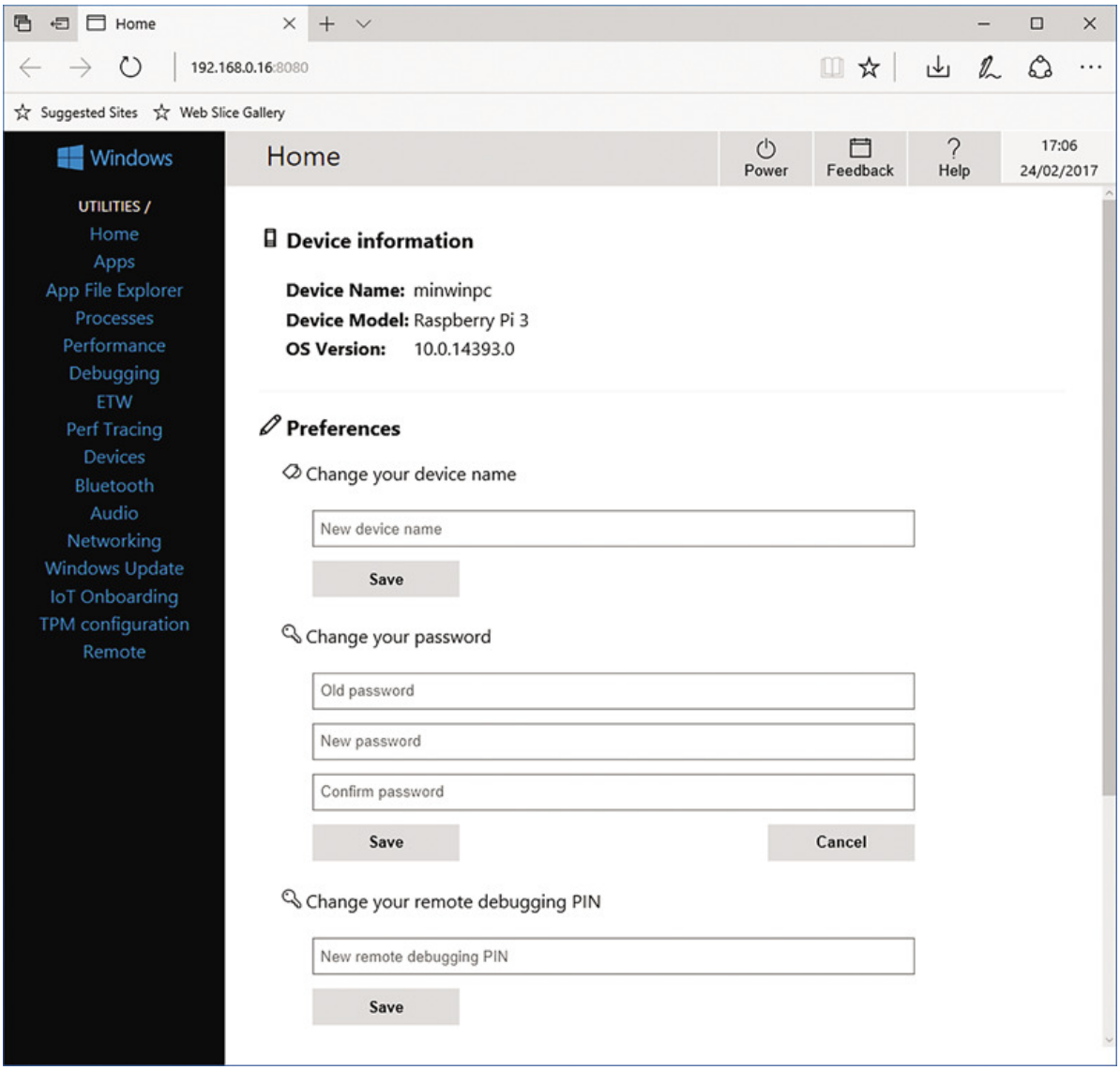


**Figure 3 Device Portal**

Here, you can configure the device, check the installed apps, and verify its performance and storage. The last option, Remote, lets you enable remote control for the device. This option is useful if the device doesn't have a monitor attached to it, as you can control it remotely from your computer. Check the box labeled "Enable Windows IoT Remote Server" to enable the remote control on the device and download the remote control app for Windows 10 from the store.

Once you install it and run the Windows IoT Remote Control app, you can control and interact with the device remotely.

Now, you can start developing for the Raspberry Pi using Windows 10.

## Developing for the Raspberry Pi Using Visual Studio

To develop for the Raspberry Pi using Visual Studio, you must make sure that you installed the tools. You can check this by selecting "Custom installation" and checking the Universal Windows App Development Tools in the Features selection.

Once you do that, you'll have the tools installed and you can start developing for the Raspberry Pi using Windows 10. Create a new project and select the "Blank" UWP app.

This will create a blank app and you'll create an app that shows the name of the current machine in the main screen. In MainPage.xaml, add the following code:

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBr
  <TextBlock FontSize="32" x:Name="MachineText"
    HorizontalAlignment="Center"
    VerticalAlignment="Center"/>
</Grid>
```

Then, in MainPage.xaml.cs, put this code to get and display the machine name:

```
public MainPage()
  this.InitializeComponent();
  Windows.Security.ExchangeActiveSyncProvisioning.EasClientDevice
    new Windows.Security.ExchangeActiveSyncProvisioning.EasClient
  MachineText.Text = eas.FriendlyName;
}
```

If you run this app on your local machine, it will show a window with the name of your machine.

Then, run the Raspberry Pi. On the Solution Platform dropdown, select ARM, and on the Device dropdown, select Remote Machine. A dialog box opens to select the remote machine.

Select the Raspberry Pi device and run the application. The app will be deployed to the Raspberry Pi and you can see it running in the remote control window. Note that the machine name shown in the window should be the one you have set when you formatted the SD card and installed Windows 10 on it.

You can debug this app the same way you do with local apps—set breakpoints, analyze variables and so on. If you terminate the app in Visual Studio, you'll see that the app closes and the main screen appears in the Raspberry Pi. If you go to the browser portal, you'll see that the app is still installed and can be run by using the Run button, as shown in **Figure 4**.

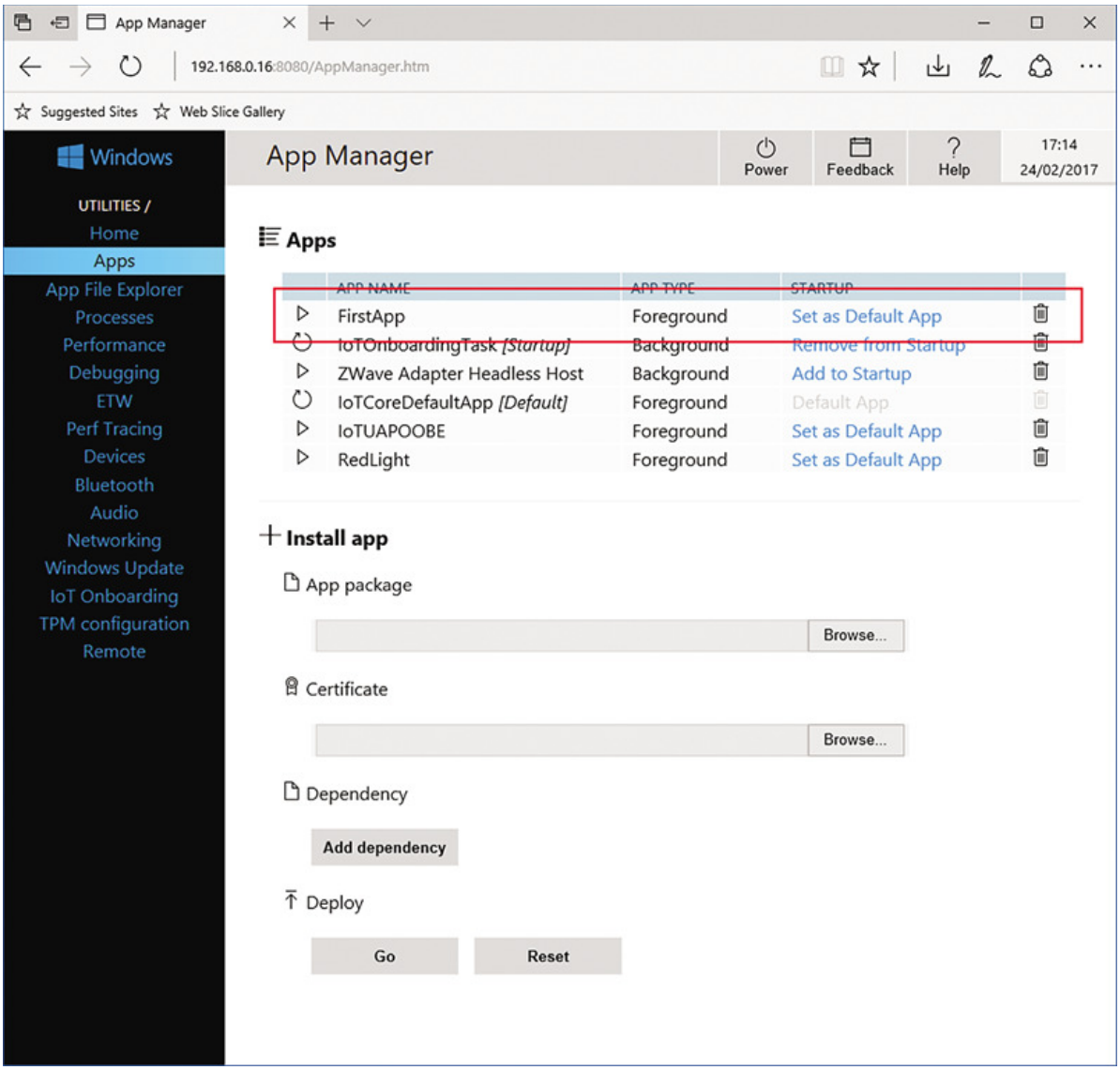


**Figure 4 App Portal Showing the Installed App**

I'm amazed at the level of compatibility achieved with UWP apps. To show the potential, I'll use an app that wasn't made for the Raspberry Pi. I'll use the sample app for the UWP Community Toolkit, a toolkit of components developed by Microsoft and the community, which definitely is worth checking out at bit.ly/2b1PAJY (http://bit.ly/2b1PAJY).

If you download the package and compile it, you can deploy it to the Raspberry Pi and run it (yes, you can run the program in the exact same way you would on a desktop). By the way, you should try using the controls in the device—they work fine.

## Interacting with the Board

Once you have your programs running, you must start interacting with the board. You'll create a traffic light controller. It will have three LEDS (red, yellow and green) and there can be different timings for each light.

To operate on the LEDs in the board, you must get the GPIO Controller and open the pin you want to control and set it the way you want. In **Figure 1**, you see that the eighth pin in the GPIO block (second row) is pin 22. You'll use pins 22, 9, and 19, and the resulting circuit will be like the one shown in **Figure 5**.

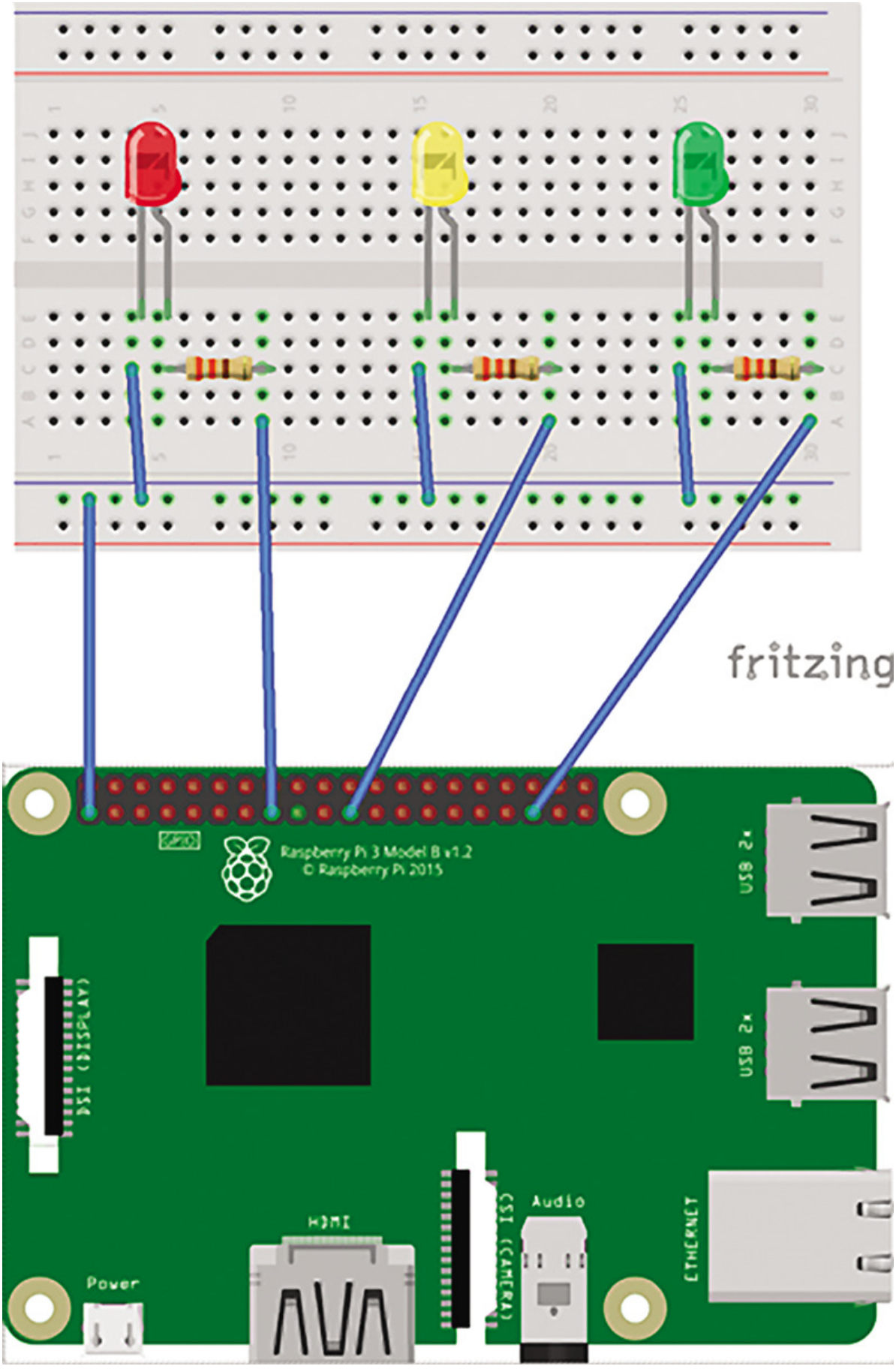


**Figure 5 Circuit for the Traffic Lights**

With this circuit in place, create a new blank UWP app. In MainPage.xaml, input the

code shown in **Figure 6**.

**Figure 6 Main Page xaml Code, Showing Traffic Lights**

Copy

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBr
  <Border BorderBrush="Black" BorderThickness="3" HorizontalAlign
      VerticalAlignment="Center" CornerRadius="5">
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="C
      <Ellipse Width="50" Height="50" Fill="Red" Opacity="0.5"
        Margin="20,20,20,10" x:Name="RedLed" Stroke="Black"
        StrokeThickness="1"/>
      <Ellipse Width="50" Height="50" Fill="Yellow" Opacity="0.5"
        Margin="20,10,20,10" x:Name="YellowLed" Stroke="Black"
        StrokeThickness="1"/>
      <Ellipse Width="50" Height="50" Fill="LimeGreen" Opacity="0
         Margin="20,10,20,20" x:Name="GreenLed" Stroke="Black"
         StrokeThickness="1"/>
    </StackPanel>
  </Border>
</Grid>
```

You'll see the traffic lights both on the board and on the display, so you can also see what's happening by viewing the remote display. The source code in Mainpage.xaml.cs is shown in **Figure 7**.

**Figure 7 Source Code to Turn on Traffic Lights at Specified Intervals**

Copy

```
private int _currentLight;
private DispatcherTimer _timer;
private int[] _pinNumbers = new[] { 22, 9, 19 };
private GpioPin[] _pins = new GpioPin[3];
public MainPage()
{
  this.InitializeComponent();
  if (InitGPIO())
    InitTimer();
}
private void InitTimer()
{
  var intervals = new[] { 6000, 2000, 6000 };
  var lights = new[] { RedLed, YellowLed, GreenLed };
  _currentLight = 2;
  _timer = new DispatcherTimer { Interval = TimeSpan.FromMillisec
  _timer.Tick += (s, e) =>
  {
    _timer.Stop();
    lights[_currentLight].Opacity = 0.5;
    _pins[_currentLight].Write(GpioPinValue.High);
    _currentLight = _currentLight == 2 ? 0 : _currentLight + 1;
    lights[_currentLight].Opacity = 1.0;
    _pins[_currentLight].Write(GpioPinValue.Low);
    _timer.Interval = TimeSpan.FromMilliseconds(intervals[_curren
    _timer.Start();
  };
  _timer.Start();
}
```

To run this code, you must add a reference to the IoT extensions. Right-click in the References node in the Solution Explorer, click on Add References, then go to Extensions and add the Windows IoT Extensions for UWP.

You create a timer that will turn on each LED in an interval set by the intervals array (in your case, 6s, 2s and 6s). The ellipses in the screen have their opacity set to 0.5, so they appear dimmed and each one will be set to 1 when the light is on. The timer will only be set if you can set the GPIO for the board, in the InitGPIO function, as shown in **Figure 8**.

**Figure 8 Code to Initialize GPIO and Set LED Pins for Output**

Copy

```
private bool InitGPIO()
{
  var gpio = GpioController.GetDefault();
  if (gpio == null)
    return false;
  for (int i = 0; i < 3; i++)
  {
    _pins[i] = gpio.OpenPin(_pinNumbers[i]);
    _pins[i].Write(GpioPinValue.High);
    _pins[i].SetDriveMode(GpioPinDriveMode.Output);
  }
  return true;
}
```

You open the three pins for output and set them to High so the LEDs are turned off. When you set a pin to Low, the current will flow by the circuit and the LED will turn on. When you run the program, you'll see a screen with a traffic light, where the lights go on and off like a real one, and the board will look like the photo in **Figure 9**.

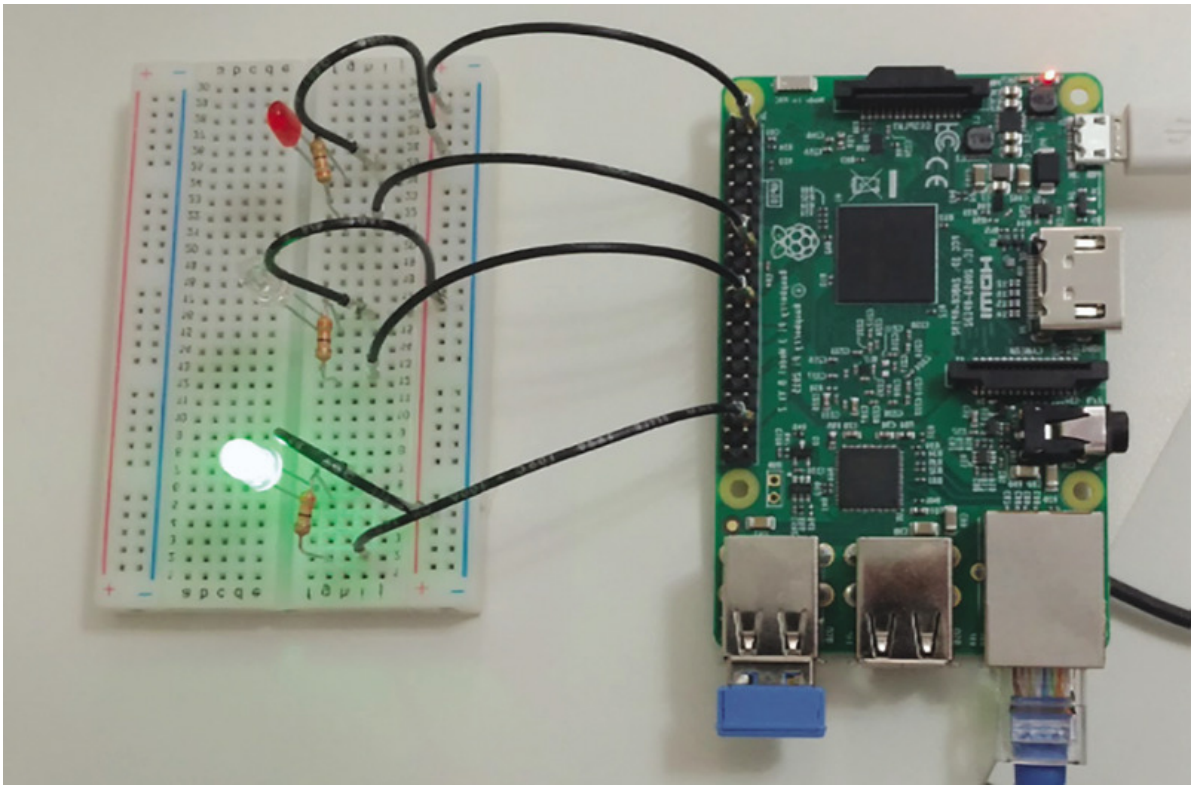

**Figure 9 Traffic Lights Board with Program Running**

## Wrapping Up

As you can see, it's simple to create programs that interact with the Raspberry Pi. Knowing Windows 10 programming, you already have all the knowledge you need to program a Raspberry Pi (yes, interacting with the board is a different story, but it's already halfway done). You can create your programs the same way you create any Windows 10 program (in fact, UWP programs run with no change in the Raspberry Pi). The only difference is that you have the GPIO controller to set data and get data. If you want to extend your knowledge and try other projects, there are many samples to try at bit.ly/2llecFZ (http://bit.ly/2llecFZ). This will open a multitude of opportunities and you'll be able to combine a powerful hardware with great and productive software. That's an unbeatable combination.

---

**Bruno Sonnino** *has been a Microsoft MVP since 2007. He's a developer, consultant, and author, having written many books and articles about Windows development. You can follow him on Twitter: @bsonnino (https://twitter.com/@bsonnino) or read his blog at blogs.msmvps.com/bsonnino (http://blogs.msmvps.com/bsonnino).*