Lesson Plan

LESSON TITLE: **Module 10: The Internet of Things (IoT)**

SUMMARY:

Topic Outline

- Introduction to the Internet of Things
- IoT embedded platforms: SoC and Microcontroller
- Microcontroller Development with the Arduino
- Windows 10 IoT Development with Visual Studio and the Raspberry Pi (RPi)
- Using the RPi and Arduino together in an IoT application

GRADE BAND:

☐ K-2 ☒ 6-8

☐ 3-5 ☒ High School

Time Required:

90 minutes

Lesson Learning Objective/Outcomes: Upon completion of this lesson, students will be able to:

To understand Internet of Things (IoT) platforms and programming

Materials List:

Raspberry Pi 3 (Model B); Arduino Uno; Arduino IDE; Visual Studio 2015 Community Edition (Update 3) with Windows 10 SDK 10.0.14393 and 10.0.10586; Windows 10 workstation with Developer Mode enabled; secondary HDMI-compatible display; secondary USB mouse for Raspberry Pi; an available wireless network

How will you facilitate the learning?

- Describe the Warm-up Activity/Focused Activity/Closure and/or Reflection
- Describe the Teacher Instruction

We adopt the following best instructional practices and strategies to facilitate learning:

- Multimodal presentation of information
- Cooperative active learning
- Team building
- Periodic checking for understanding

This lesson includes:

☒ Mapping to Cyber Security First Principles
☒ Learning Objectives
☒ Assessments

Mapping to Cyber Security First Principles:

☐ Domain Separation
☒ Abstraction
☒ Process Isolation
☒ Data Hiding
☒ Resource Encapsulation
☐ Layering
☒ Modularity
☒ Simplicity
☐ Least Privilege
☐ Minimization

Assessment of Learning:

TYPE (Examples Listed Below)	NAME/DESCRIPTION
Quiz/Test Presentation Oral Questioning	Online pop-quiz/survey using gosoapbox.com will be utilized to check understanding of key indicators. Key Indicators of Understanding • Recognition of Cyber Security Principles involved • Basic knowledge of IoT • Basic knowledge of embedded development tools Hands-on exercises will be conducted to reinforce learning. Participants will be asked to present their observations on pertinent case studies.

Accommodations: (Examples may include closed captioning for hearing impaired students; accommodations for students with disabilities.)

We will provide digital, closed-captioned videos to correspond to the hands-on exercises.

Description of Extension Activity(ies):

Background Materials

Introduction to the Internet of Things (IoT)

In 2013, the Global Standards Initiative on the **Internet of Things** (IoT-GSI) defined the IoT as "a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies." These interconnected "things" can include appliances, phones, wearable electronics, or any other everyday device that can be connected to the Internet. This new category of devices is made possible by the increasing availability of low-cost broadband Internet access, and by the availability of small and inexpensive *embedded computers*,

which are typically built into the device and are dedicated to controlling or monitoring the device.

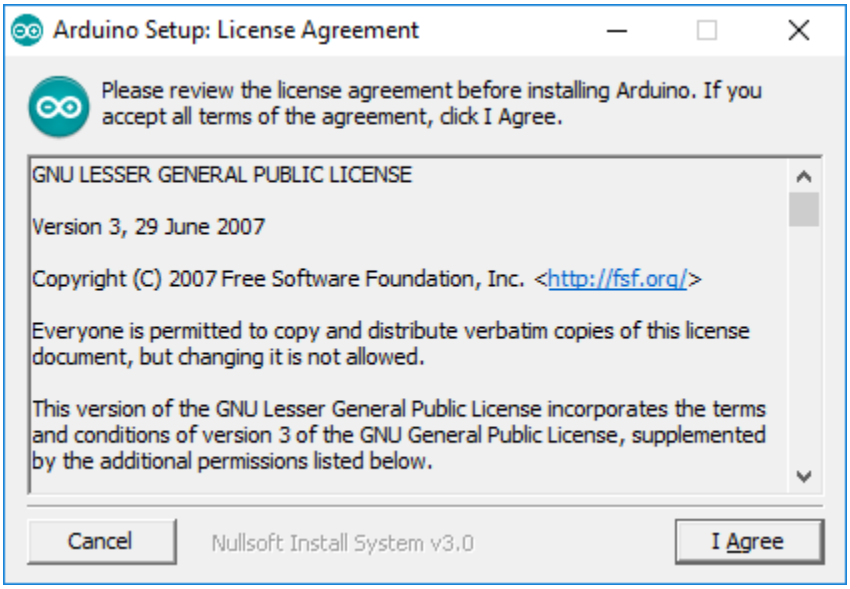
In principle, these embedded computers are no different from conventional desktop and laptop computers, in the sense that they contain a *central processing unit* (CPU) and *memory* and execute instructions in the form of *software* written by a programmer. However, they are typically smaller and more efficient than conventional computers, and depending on the application, they may lack a traditional *user interface* (such as a keyboard and mouse). These embedded computers often take the form of a *system on a chip* (SoC), with all the components of a conventional computer integrated into a single chip. They can also take the form of *microcontrollers*, which are even smaller single-chip computers that are typically optimized for device monitoring and control.
In this module, we will be working with two popular examples of these categories of embedded computers: the **Raspberry Pi**, a single-board computer based on an integrated SoC, and the **Arduino**, a project-oriented microcontroller platform. The Raspberry Pi (RPi) is compatible with an embedded version of the Microsoft Windows 10 operating system called **Windows 10 IoT**, which is already installed on the SD memory card that is included with the RPi computer in your kit. Both the RPi and the Arduino platforms are provided with tools and examples that are designed to make embedded application development as easy as possible. They are also popular platforms among students for the exploration of computer programming.

Associated Problem-Based Laboratory Exercises

We will begin this exercise by installing the necessary programming tools for the Arduino and the Raspberry Pi, and then we will use the electronic components included with your kit to construct an array of LED lights. As our first experiment with Arduino programming, we will interface the LED array to the Arduino board and switch the LEDs on and off under software control. Later, we will use the same LED array with a more elaborate program which will allow the Arduino to be remotely controlled by the Raspberry Pi.

Part One: Installing the Tools

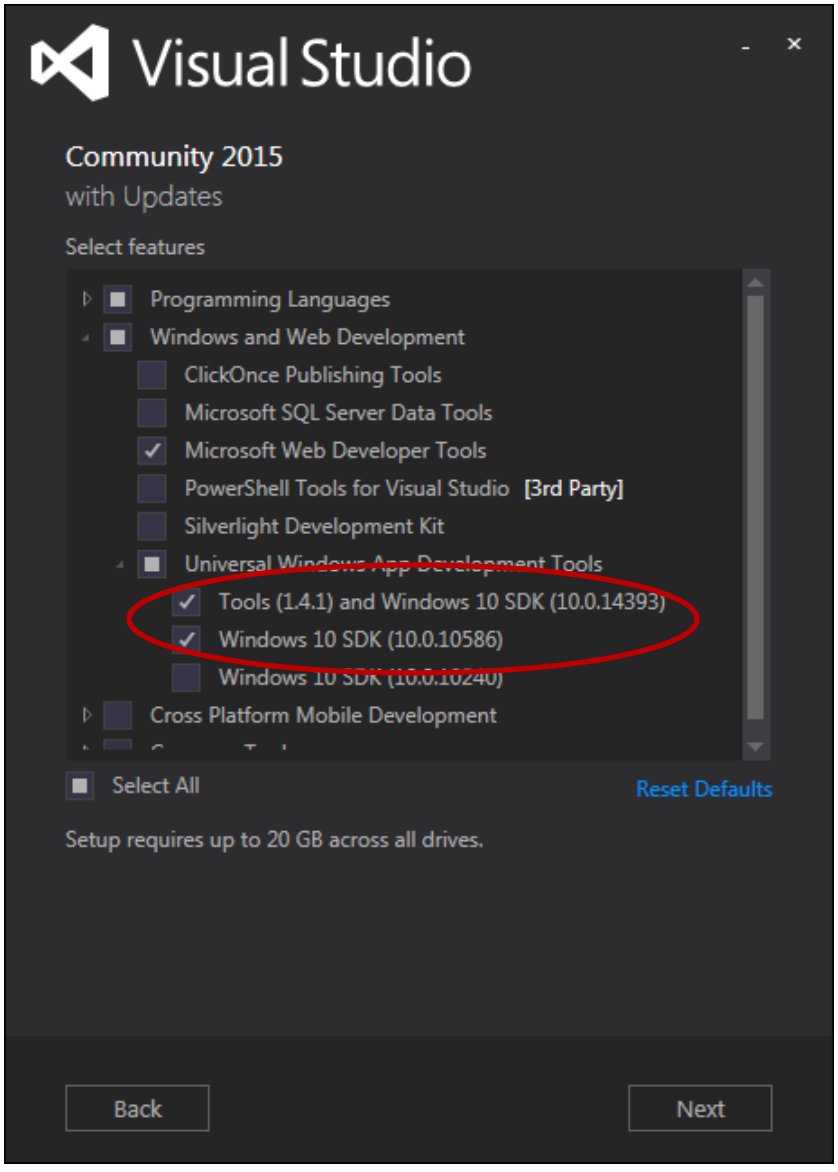
The first step is to install and configure the necessary programming tools, including the Arduino Integrated Development Environment (IDE) and Microsoft Visual Studio. *(NOTE: These tools are already installed on the JSU lab workstations, so during the GenCyber workshop, you may skip these steps and proceed to Part Two. Refer to this section later when installing the tools on your own computers.)*
Open the "**Module10**" folder provided with this exercise (you will find it on your workstation Desktop), then double-click the "**arduino-1.8.3-windows.exe**" file to launch the Arduino IDE installer:



1) Accept the license terms by clicking the "I Agree" button.
2) Accept the default options shown in the "Installation Options" window. Click "Next."
3) Accept the default installation folder and click "Install" to begin the installation. This process may take a few minutes. After the installation is completed, click the "Close" button to exit.

Next, we will install Microsoft Visual Studio. In the "**Module10**" folder, you will find the installation program file "**vs_community_ENU.exe**". This is the Community Edition of Visual Studio 2015, which is available for free from Microsoft. Double-click this file to begin the installation process. *(NOTE: This process will take much longer to complete, so you should wait until after you are done using your computer for other purposes before installing Visual Studio.)*

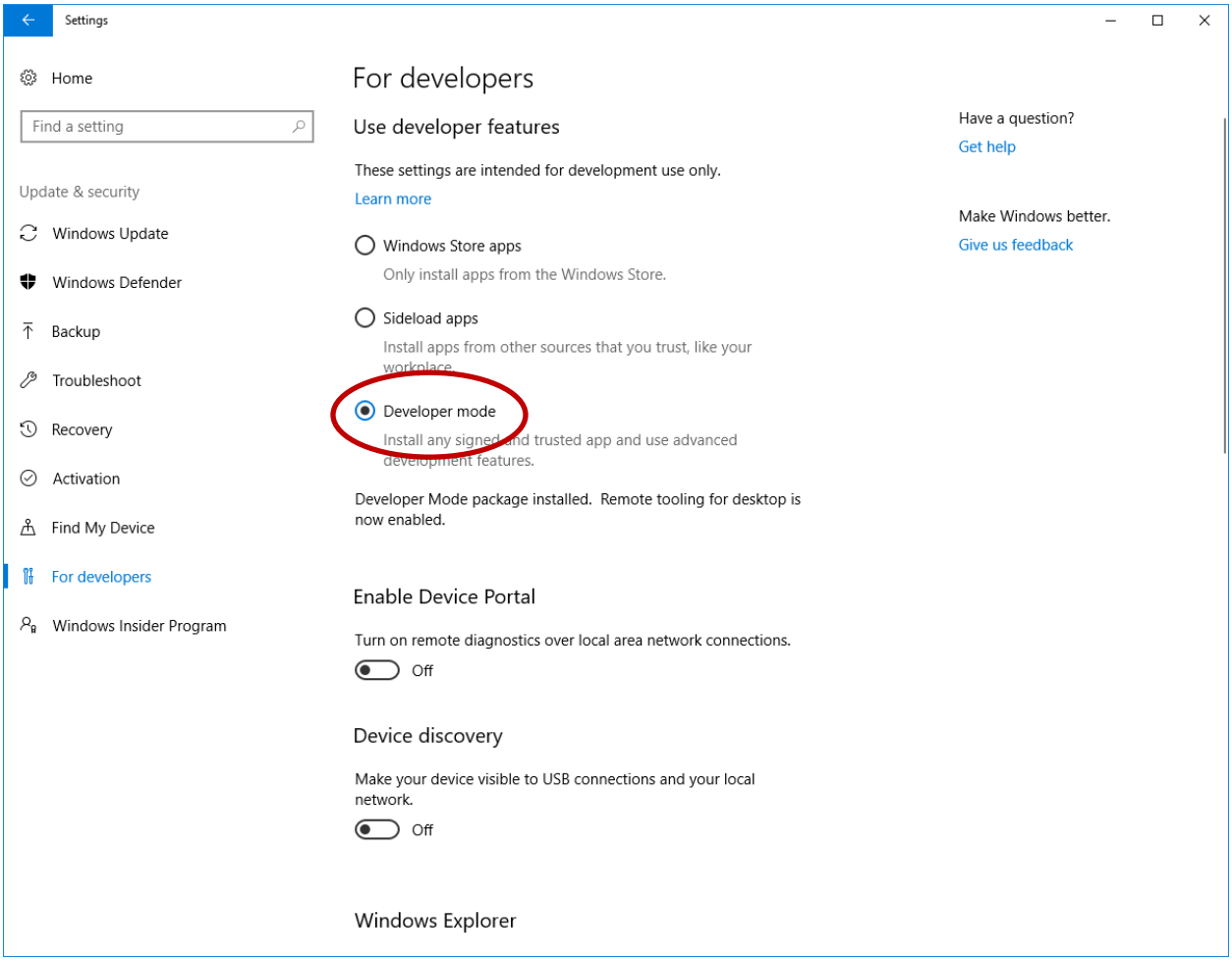
1) When the installation window appears, accept the default installation location. In the "Choose the type of installation" area, choose the "Custom" option and click "Next."
2) In the "Select Features" window, expand the "**Universal Windows App Development Tools**" group, and select the following optional features by checking the corresponding checkboxes: "**Tools (1.4.1) and Windows 10 SDK (10.0.14393)**" and "**Windows 10 SDK (10.0.10586)**". Optionally, in the "**Programming Languages**" group, select "**Visual C++**" to develop applications using the C++ language (we will be using the C# language, which is installed by default).
3) The selected options should resemble the options shown in the following screenshot. *(Make sure the circled Universal Windows App tools are selected!)* Click "Next" to proceed.



4) Click the "Install" button to begin. Visual Studio will be downloaded and installed, along with the applicable updates. After the installation is complete, you may be asked to restart your computer.

Finally, if your workstation is running the Windows 10 operating system, you will need to enable Developer Mode. This is a fairly simple process which should need to be done only once:

1) Open the Start Menu and click the "Settings" icon.
2) Choose "Update & Security", and from the left-hand menu, choose "For developers".
3) Select the "Developer mode" option, circled in the screenshot below. When prompted for confirmation, choose "Yes".



Part Two: Preparing the LED Array

Now that you have the necessary tools installed on your workstation, the next step is to assemble the LED array using the parts provided in your kit, and to interface the array with your Arduino project board. You will need the following components from the kit:

- The *breadboard*, which is a construction base for the prototyping of electronics projects. The breadboard included in your kit is perforated with holes that are arranged in rows and columns. Notice that the columns are arranged on the outside edges of the breadboard and are labeled "+" and "-". These are *buses*, intended to supply the other components with a direct connection to the supply voltage or to ground. All the holes in each bus are electrically connected. Notice also that the rows are numbered, and that the holes in each row are divided into two groups of five. The holes in the first group (labeled as columns "**a**" through "**e**") are electrically connected, as are the holes in the second group (labeled as columns "**f**" through "**j**").
- Four red *light emitting diodes* (LEDs).
- Four *current limiting resistors*, which we will add to the circuit to drop the source voltage from the Arduino project board to a level that is safe for the LEDs. Look at the resistors in your kit, and notice that they are labeled with colored bands which indicate their resistance values. We will use the four

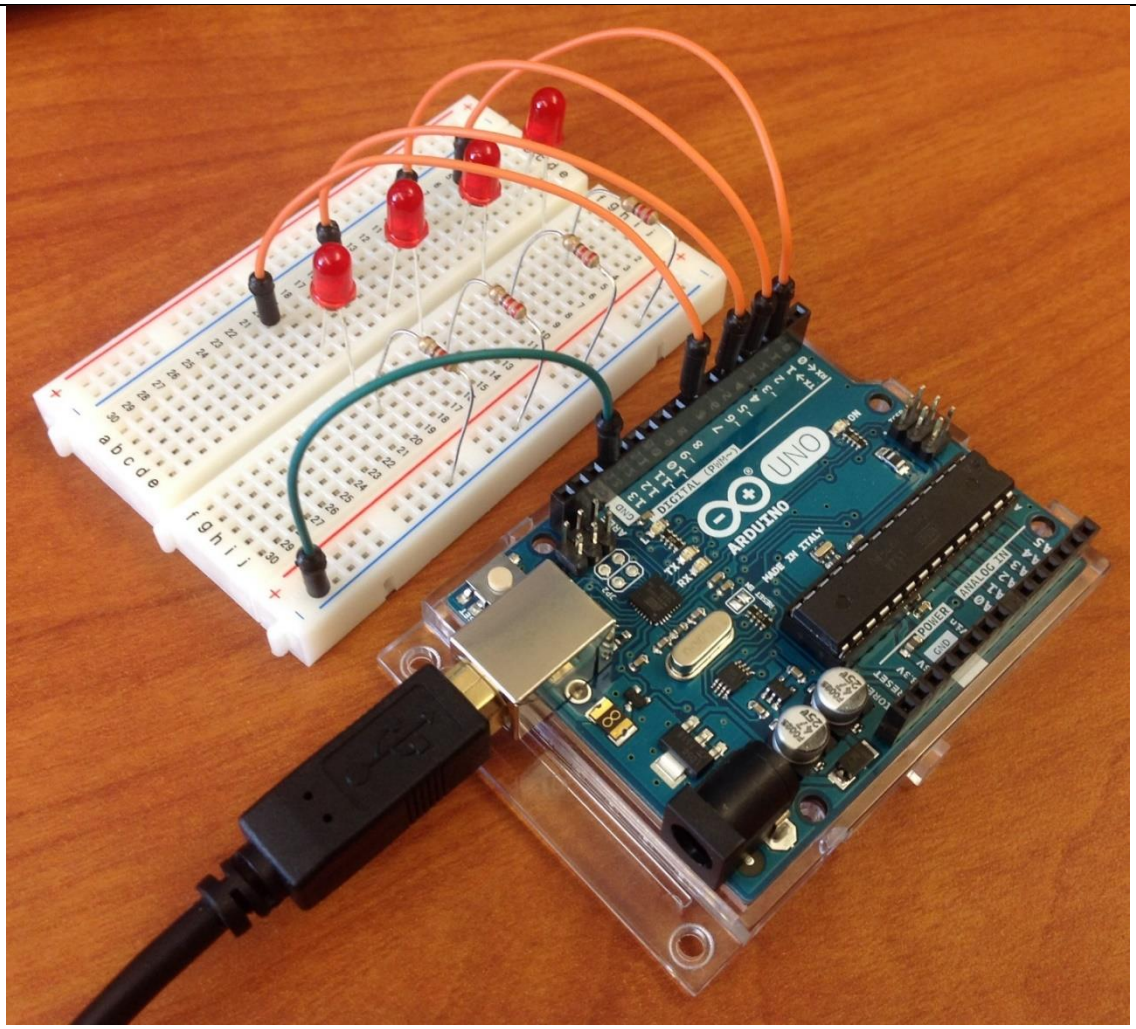
220-ohm resistors included in your kit, which are labeled with the bands "**Red-Red-Brown-Gold**". (For our LEDs, the resistance values are not critical, so if you have other resistors whose value is close, such as 180-ohms or "**Brown-Gray-Brown-Gold**", use these instead.)

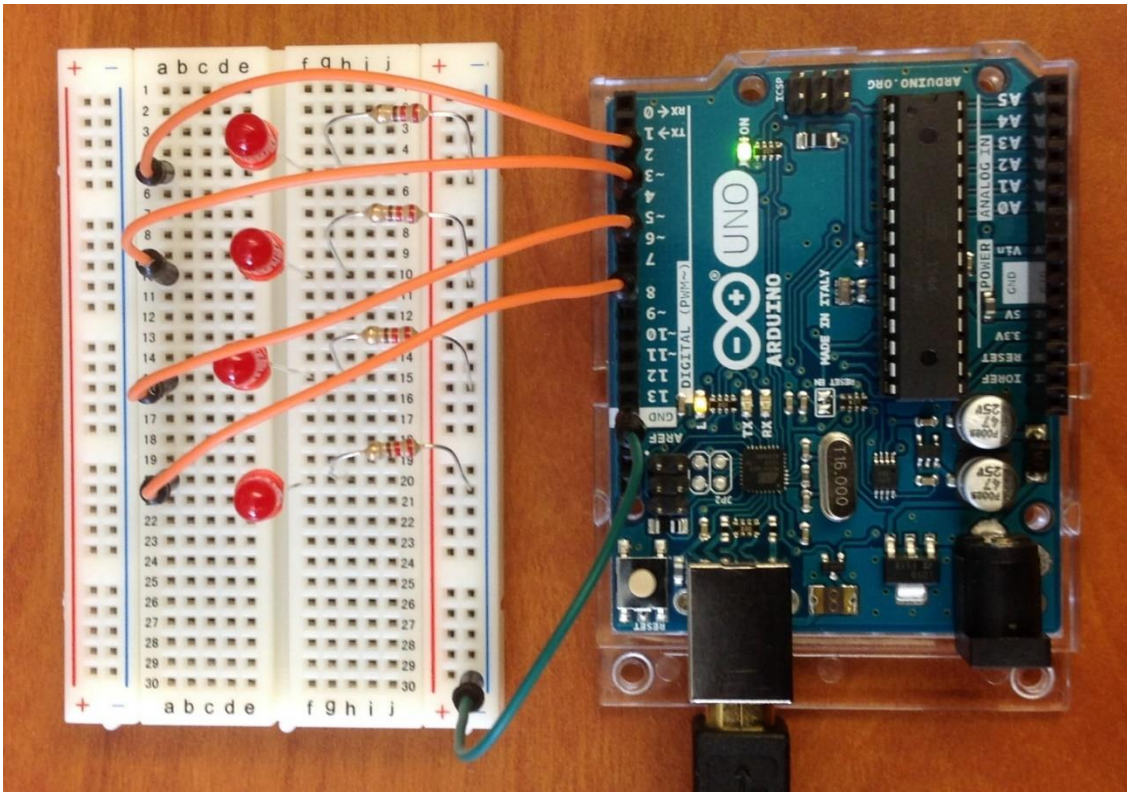
- Five male-to-male *jumper wires* to connect the breadboard to the Arduino. Use one of the short green wires in the kit for the connection to ground, and four of the short orange wires for the connections to the Arduino's I/O pins. It is advisable to use different wire colors for different types of signals, to make them easier to distinguish when checking your connections.

(During the following steps, refer to the completed examples shown on the following page.)

1) Look at the Arduino board, and notice that it includes two columns of female pin connectors, and that each pin is labeled to identify the corresponding signal. Begin by connecting one end of the green wire to the "GND" pin on the Arduino (there are several of these), then connect the other end to one of the holes in the "-" bus on the breadboard. This bus will provide a connection to ground for the other components on the breadboard.
2) Next, connect the four orange wires to the Arduino's *input/output pins*. The Arduino has a total of fourteen general-purpose signal pins which can be used for input or for output; they are labeled with numbers on the Arduino board. The first two of these pins are reserved for serial communication, so we will use the even-numbered pins from 2 through 8 for our LEDs. Connect one end of the first orange wire to Pin 2 of the Arduino board, and connect the other end to the hole in Row 5 and Column "**a**" of the breadboard. Repeat the process for the next three pins: the wires to Pins 4, 6, and 8 should connect to Rows 10, 15, and 20 of the breadboard, respectively (use the hole in Column "**a**" in each row). This will ensure that the components are distributed evenly across the breadboard, with enough space between them to prevent short circuits.
3) Next, mount the LEDs on the breadboard. Notice that each LED has two terminals, or "legs", and that one is longer than the other. The longer terminal is called the *anode*, and the shorter terminal is called the *cathode*. Like other diodes, LEDs will allow current to flow in only one direction, from the anode to the cathode, so it is important that we do not connect the LEDs backwards. Spread the terminals of the first LED apart slightly, then connect the *anode* (the longer terminal) to the hole in Row 5 and Column "**e**", and connect the *cathode* (the shorter terminal) to the hole in Row 5 and Column "**f**". When you are finished, the LED should be "straddling" the two groups of holes in Row 5, with one "leg" in each group. Repeat the process for the remaining three LEDs, mounting them in the same way in Rows 10, 15, and 20.
4) Finally, mount the resistors to the breadboard. Bend the two leads of the first resistor in the middle to about 90 degrees, and insert one of the leads into the hole in Row 5 and Column "**h**" (the resistor is not polarized, so it does not matter which lead you choose). Insert the other lead into the nearest convenient hole in the "-" bus, the same bus where you earlier connected the "GND" pin from the Arduino board. Since all the holes in the "-" bus are connected, it does not matter which hole you choose. Repeat the process for the remaining three LEDs, mounting them in the same way in Rows 10, 15, and 20.

See the next page for an example of how a fully wired LED array should look:


UNO
ARDUINO
DIGITAL (PWM~)
ANALOG IN
POWER
MADE IN ITALY


UNO
ARDUINO
ARDUINO.ORG
DIGITAL (PWM~)
ANALOG IN
POWER
MADE IN ITALY

Part Three: Exploring the Arduino

After you have finished connecting the components for the LED array, connect the Arduino to your computer using the A-B USB 2.0 cable included in your kit. The necessary drivers should have been installed with the Arduino IDE, but if you are prompted to install any additional drivers, proceed with the installation before continuing.

Next, open the "**Module10**" folder and find the "**Test_Arduino.ino**" file. This is an Arduino program; double-click this file to open the program in the Arduino IDE. (Within the IDE, Arduino programs are called *sketches*.) You may be prompted with a message stating that the file must be moved into its own folder; if so, click "OK" to permit the IDE to move the file.

After the sketch is opened, it will appear in the Arduino IDE editor (see the screenshot on the next page). When it is run, this program will illuminate the LEDs on your breadboard in a left-to-right "strobing" pattern; later, you (or your students) will have the option of creating new patterns.

For now, let us run this program as written on the Arduino. Click the "right arrow" button (circled in the above screenshot), and the IDE will do two things: it will *compile* the program, translating it from the human-readable

form shown in the editor to *binary* (machine language) form, and then it will *download* the compiled binary program to the Arduino through the USB interface. The program should start running shortly afterward, and the LEDs should start blinking; if they do not, double-check your connections, and press the "RESET" button on the Arduino board to restart the program.

As stated earlier, if you or your students wish to experiment with this program, you can change it to create a new animation sequence. Look at the second line of program code:

```
int pins[] = {2, 4, 6, 8, 6, 4};
```

The numbers between the curly braces indicate the sequence in which the LEDs will be lit to create the "strobing" pattern; this sequence will be repeated infinitely. If you wish to change the pattern, simply change the numbers in this sequence. Just make sure that the new numbers are separated with commas, and that the list is enclosed in curly braces, as in the original. You can also make the sequence longer or shorter than the six-step sequence in the original program; just remember to change the number in Line #3 to match the number of steps in your new sequence. Finally, if you wish to make the animation faster or slower, change the value **`150`** in Line #5 to a different number (this value indicates the length of the delay between animation steps, given in *milliseconds*). When you wish to test your changes, click the "right arrow" button to download and run your revised program on your Arduino.

The Arduino instruction manual included with your kit includes more tutorial programs, as well as a reference to the C programming language (the language used by the Arduino IDE).

Part Four: Exploring Visual Studio, Windows 10 IoT, and the Raspberry Pi

Now that we have some experience with the Arduino, it is time to turn our attention to the Raspberry Pi (RPi). The RPi is a more powerful and versatile device than the Arduino, so the development tools are necessarily more complex. Before we begin working with Visual Studio, however, we must start the RPi and ensure that it completes the boot process and connects to the network successfully. First, if the SD card provided with your RPi is not already inserted into its SD card slot, please insert it now, as shown below. *(The SD card shown here is fully inserted; do not try to force the SD card any farther into the slot!)*

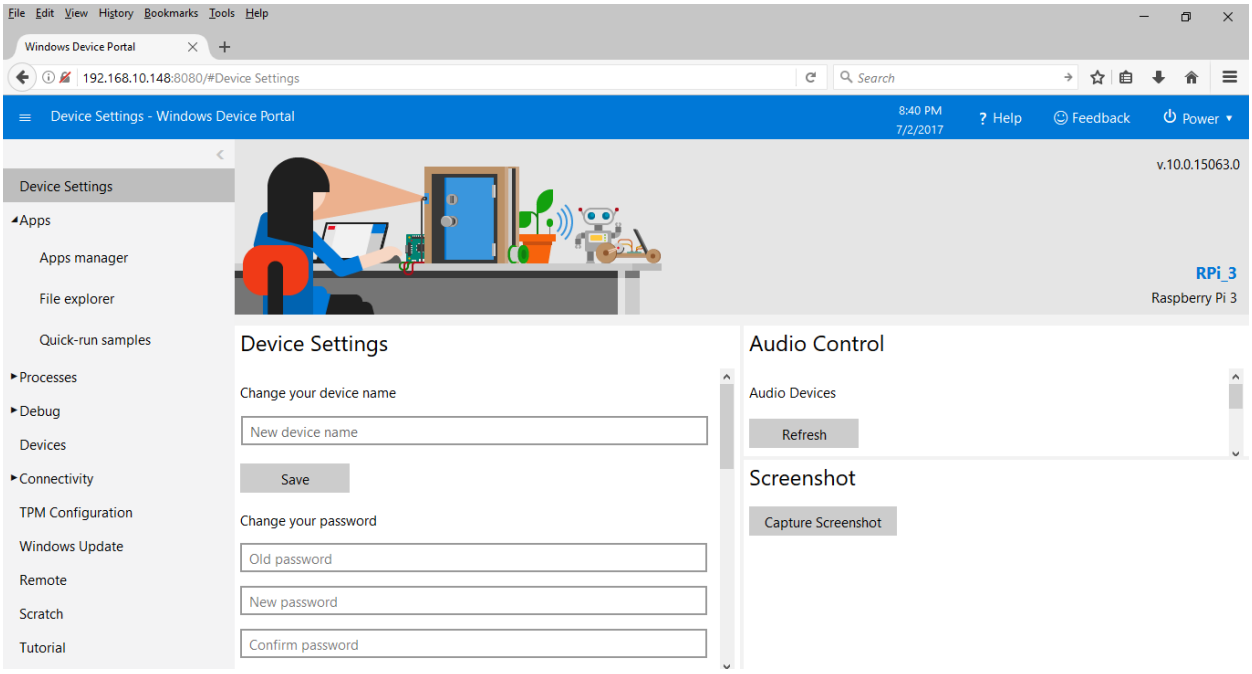
This SD card contains a full installation of the Windows 10 IoT operating system. To start the boot process, simply plug your RPi into a power outlet using the USB power adapter included in your kit. If you wish to interact with the RPi directly, connect it to a television or monitor using the included HDMI cable, and connect a keyboard and/or mouse to the RPi's USB port(s).

After you have connected your RPi to an HDMI display, you should be able to see the Windows 10 start screen. After the boot process is complete, an information screen will appear which includes the network address of your RPi. The Windows IoT installation on your SD card is preconfigured to connect to the "**GenCyber**" wireless network in the JSU lab; later, you will reconfigure it for your own network. For now, make note of the network address shown on this screen; you will use it later to manage your RPi over the network.

On your workstation, open a Web browser and enter the RPi's network address into the address bar, followed by "**:8080**". For example, if your RPi's network address is **192.168.10.148**, enter the following into the address bar:

```
192.168.10.148:8080
```

If your RPi is communicating properly with the network, you will be presented with a login prompt. Enter "**Administrator**" as the username and "**GenCyber!2017**" as the password. After you log in successfully, a "Windows Device Portal" page will appear. This page allows you to manage your RPi over the network: you can view its device information, manage the installed applications, and more:

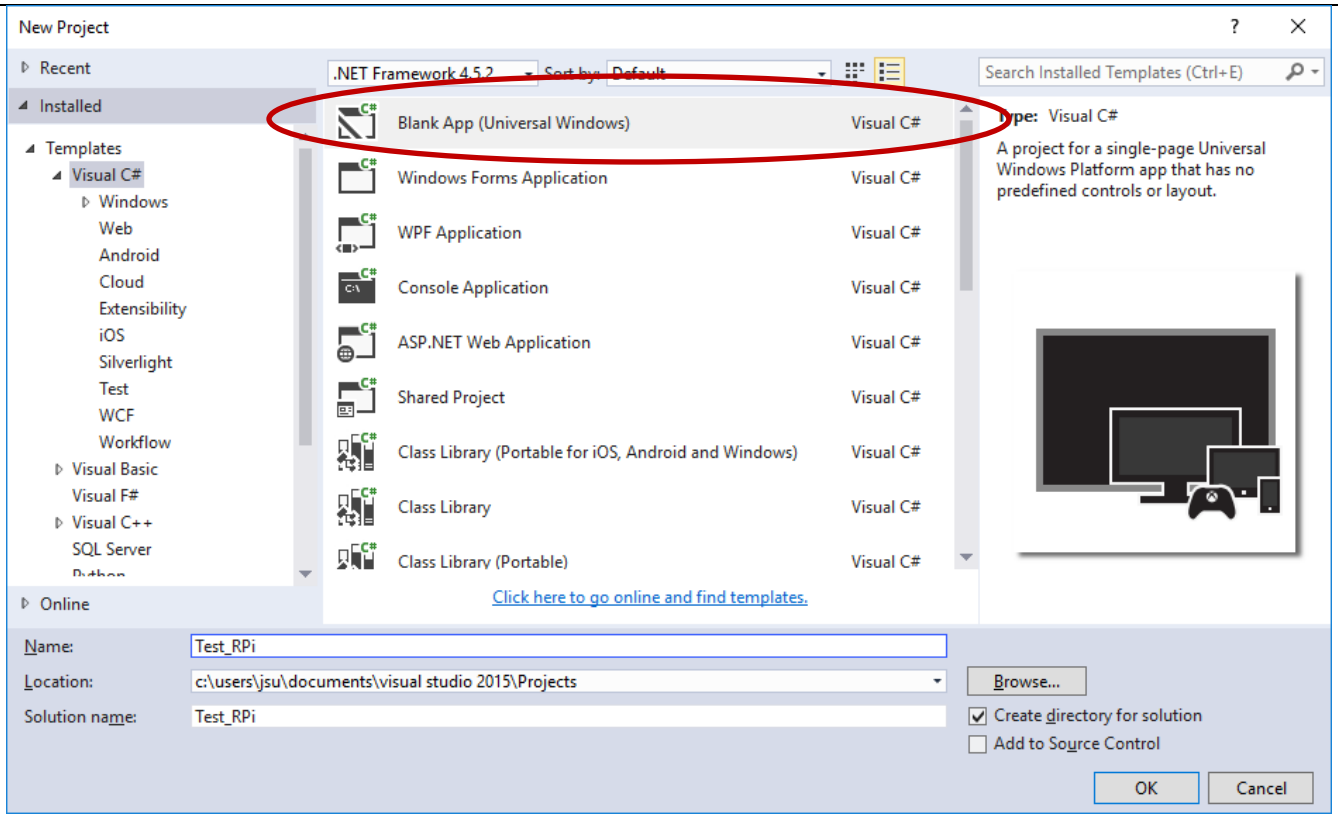


This screenshot shows the "Device Settings" page; here, you will be able to assign a new Administrator password at a later date. The "Apps manager" (shown in the left-hand menu) is where you can start, stop, and uninstall applications; we will revisit this section later.

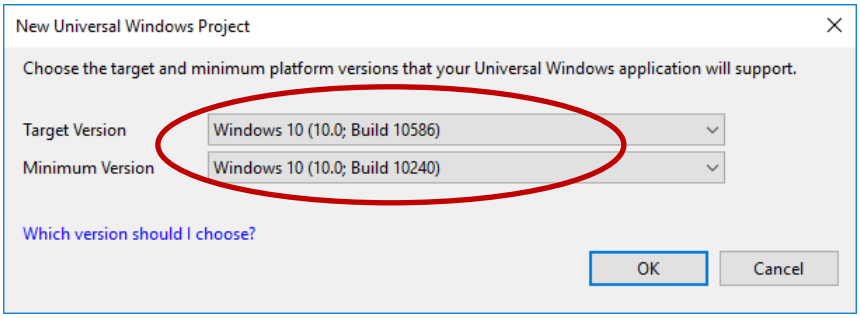
Now that the RPi is on the network, let's construct a sample Windows 10 IoT application. Visual Studio includes several programming languages, any one of which can be used to develop IoT applications. The most popular of these is **C#**, so this is the language that we will use.

1) Open Visual Studio on your workstation and choose "New → Project" from the "File" menu. In the "New Project" window, choose "**Visual C#**" from the list of languages if it is not already selected, and from the list of templates, choose "**Blank App (Universal Windows)**". In the "**Name**" field at the bottom of this window, enter "**Test_RPi**", then click "**OK**" to create the project:

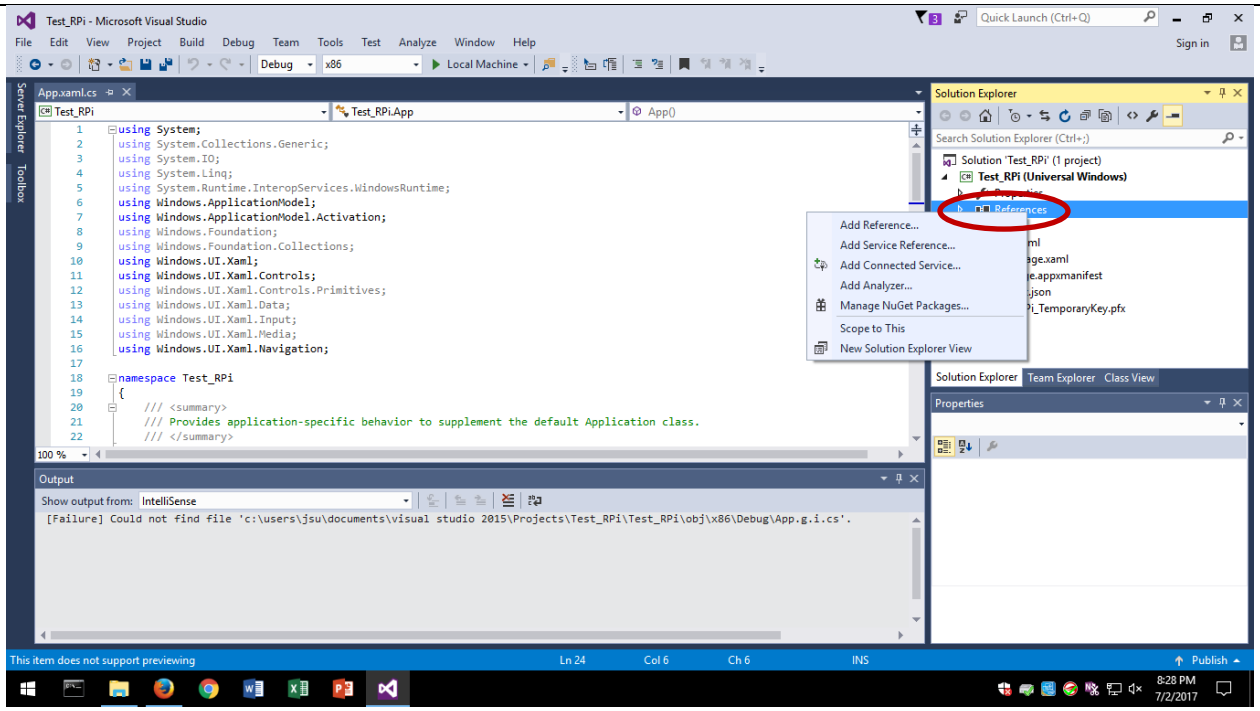
   (NOTE: If the "**Blank App (Universal Windows)**" option shown here is not available, close Visual Studio, reopen the Visual Studio installation program described in Part One, and ensure that the optional components shown there were successfully installed.)



2) A "**New Universal Windows Project**" dialog box will appear, asking which version of Windows our new application should support. Choose "**Windows 10, Build 10586**" as the target version and "**Windows 10, Build 10240**" as the minimum version, as shown on the next page. Click "OK" to continue.

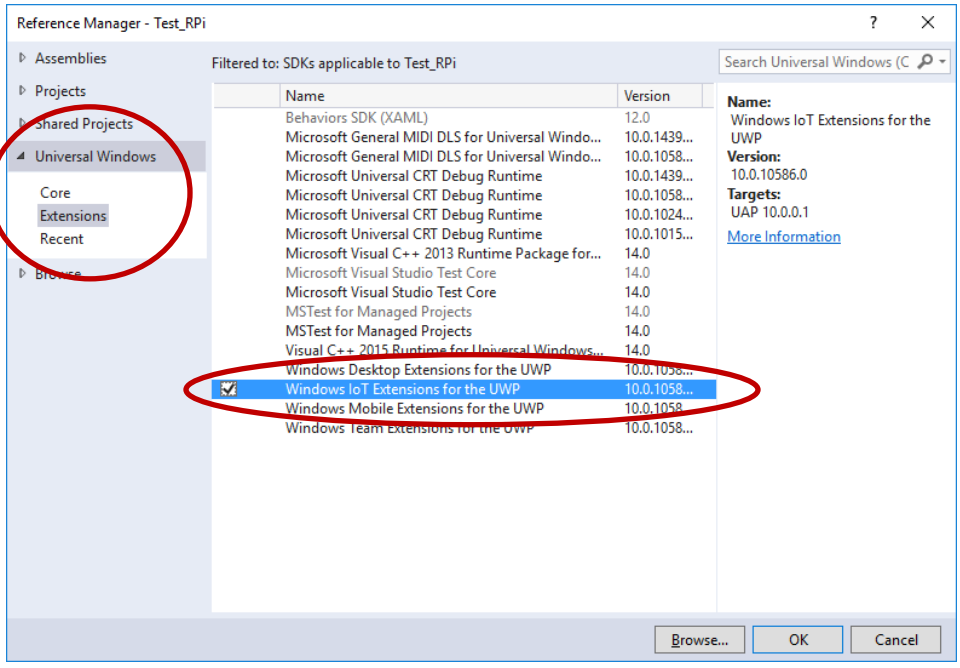


3) Visual Studio will proceed to create the project (this may take a few moments). After the project has been created, your screen should resemble the screenshot shown on the next page.

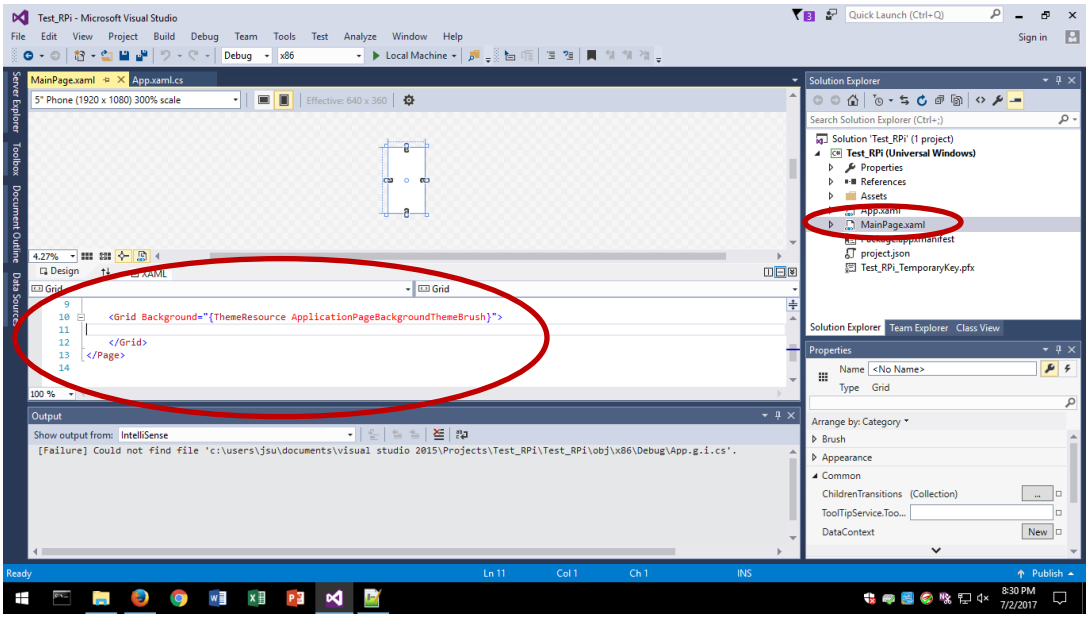


One of the new program files for our new project, created for us by Visual Studio, is being shown in the editor. We will be making several changes to these files in a moment, but first, there is one more addition that we need to make for our application to support the Windows IoT platform. Look in the "Solution Explorer" (the area in the upper-left corner of the screen) for an item called "References", circled in the screenshot above. Right-click this item and choose "Add Reference …" from the context menu. A "Reference Manager" (shown on the next page) will appear.

Open the "Universal Windows" group in the left-hand side of this window and select "Extensions"; then, in the list of SDKs in the center of this window, select the "Windows IoT Extensions for the UWP" checkbox. Click "OK" to commit your changes.



4) We are now ready to customize our sample application. Look through the list of program files in the "Solution Explorer" area and look for a file named "**MainPage.xaml**". This file determines the layout of the graphical elements within the main window of our application; at present, this window is empty. Double-click this file to open it in the editor. It will be shown in the "Designer", which depicts the appearance of the completed application; directly below this (as circled below), the underlying code is shown:



5) Scroll through the code area until you see the following lines (Lines 10 through 12 in the example shown on the previous page):

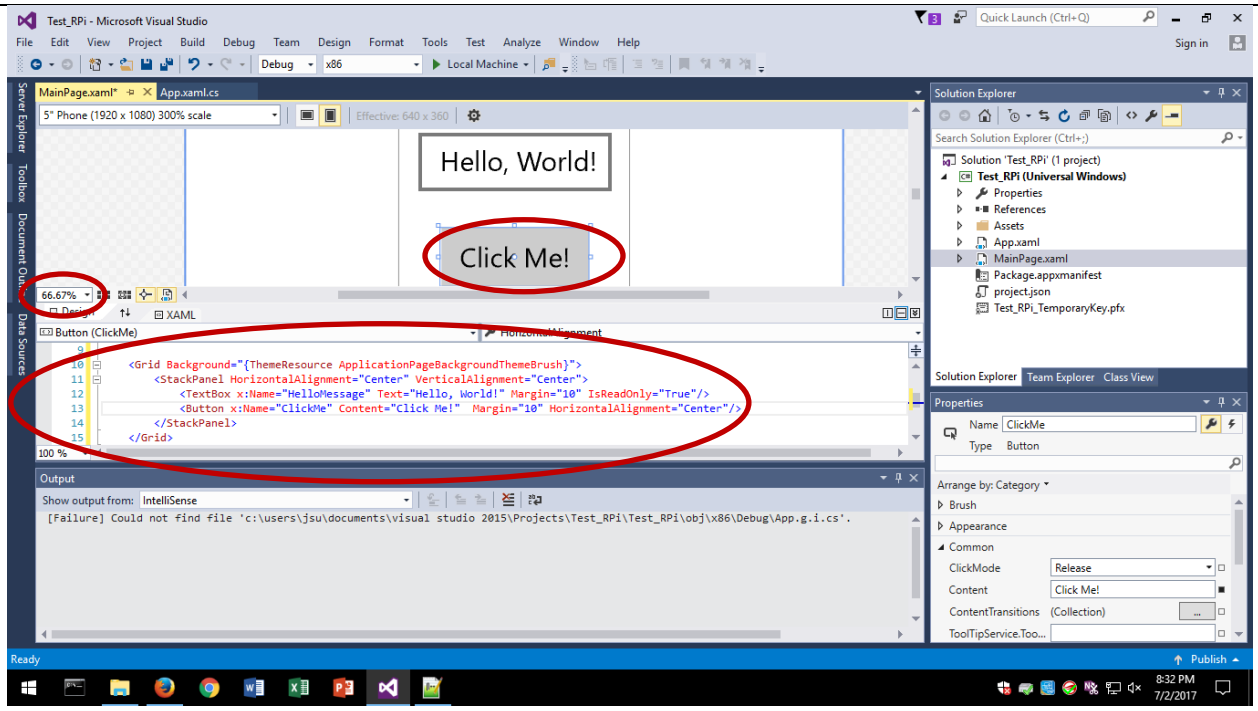
```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

</Grid>
```

Delete these lines and replace them with the following code (you can also copy and paste these lines from the "**code.txt**" file in the "**Module10**" folder):

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
    <TextBox x:Name="HelloMessage" Text="Hello, World!" Margin="10"
IsReadOnly="True"/>
    <Button x:Name="ClickMe" Content="Click Me!"  Margin="10"
HorizontalAlignment="Center"/>
  </StackPanel>
</Grid>
```

When you are finished, your screen should resemble the sample shown on the next page. Increase the view selector, circled in the screenshot, to get a closer look at the application in the previewer:



6) As you can see, this added code created two new elements in our application: a *text box*, in which we can display messages, and a *button* labeled "**Click Me!**" We can associate an action with this button, so that the program responds when the user clicks it, so let's add one more line of code.

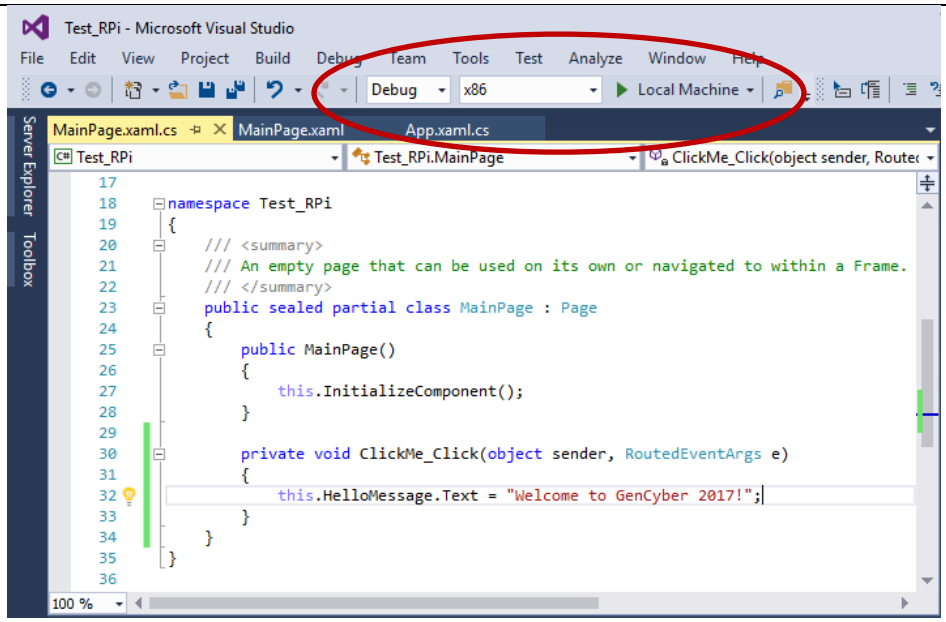
Double-click the "Click Me!" button shown in the previewer (circled in the previous screenshot), and Visual Studio will create a new *function*, a sequence of code that will be executed when the user clicks the button. Another program file, "**MainPage.xaml.cs**", will automatically be opened by the editor. Look through this file until you find the following lines:

```
private void ClickMe_Click(object sender, RoutedEventArgs e)
{
}
```

Notice that this function is empty; there are no program instructions between the curly braces. Add the following line of code *inside* the curly braces to display a message in the text box when the user clicks the button:

```
private void ClickMe_Click(object sender, RoutedEventArgs e)
{
    this.HelloMessage.Text = "Welcome to GenCyber 2017!";
}
```

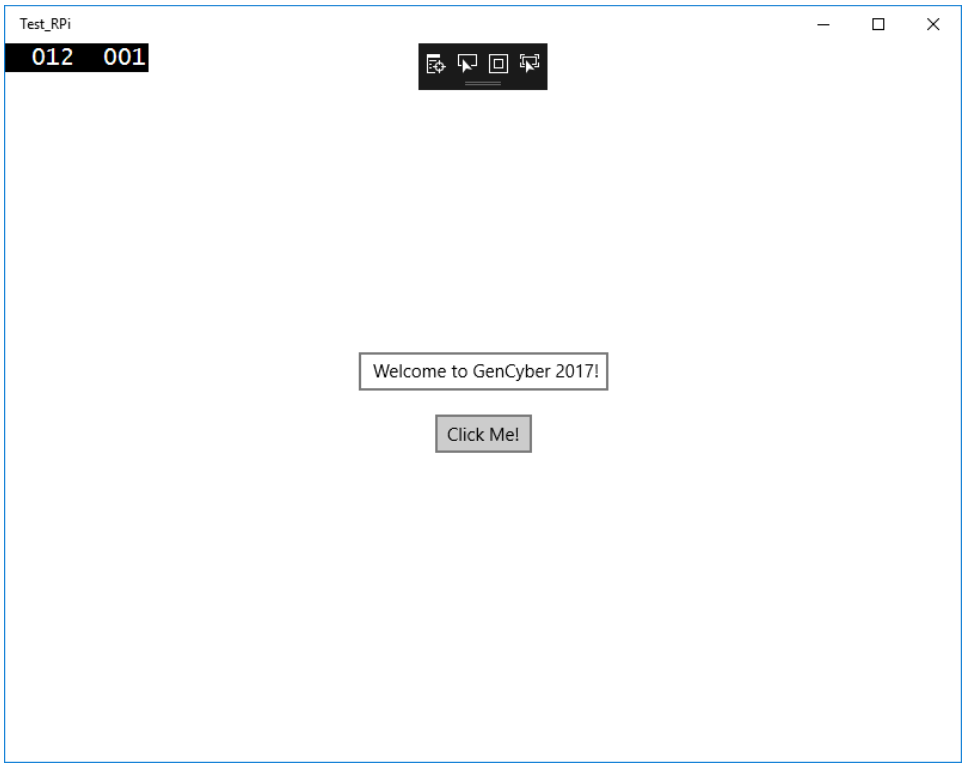
(You can copy and paste this line of code from the "**code.txt**" file also.) After making this change, your program file should resemble the sample shown on the following page:



7) After adding this last line of code, open the "Build" menu and choose "Build Solution." Visual Studio will proceed to compile your program; if this process succeeds, you should see the following message in the "Output" window at the bottom of the screen:

```
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

8) Before proceeding, look at the toolbar toward the top of the screen, circled in the previous screenshot. Notice that "Local Machine" is selected as the solution platform; this means that the application is currently configured to run on our workstation rather than the Raspberry Pi. We will change this option in a moment, but for now, let's test the application on the workstation. Click the "Local Machine" button in the toolbar, or press the **F5** key on your keyboard. Visual Studio will rebuild the application, and shortly afterward, it should appear in a new window resembling the following:

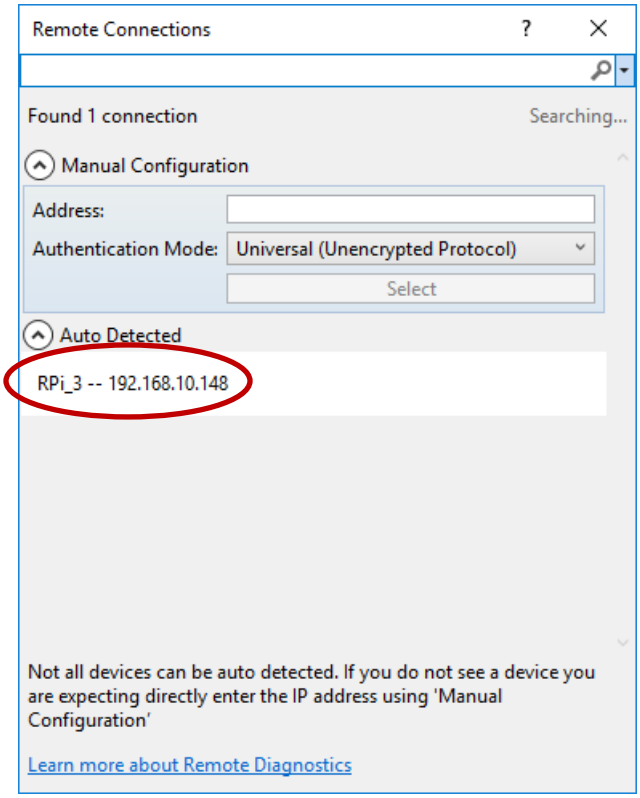


Click the "Click Me!" button, and the welcome message should appear in the text box, as shown in the screenshot.

9) To close the application, simply click the "**X**" button in the upper-right corner of the window, as usual.

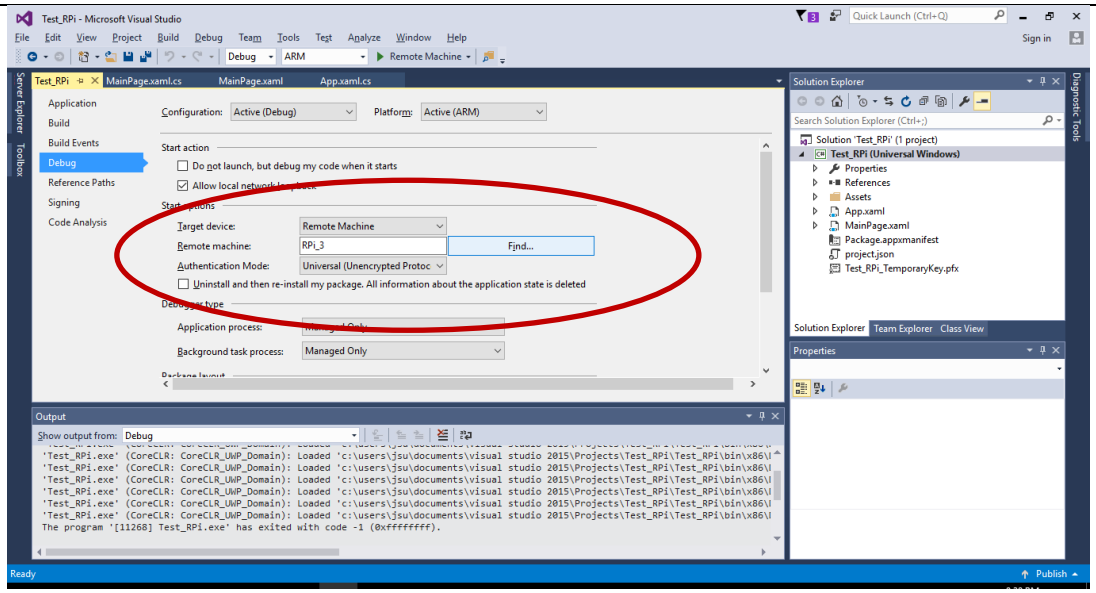
   Of course, we ultimately want to run this application on the Raspberry Pi instead of our workstation. In the drop-down list of solution platforms, change the "**x86**" option to "**ARM**" (this is the processor architecture used by the RPi). Next, click the drop-down arrow next to the "**Device**" button (to the immediate right of the drop-down list of solution platforms) and from the context menu, choose "Remote Machine."

10) A new "Remote Connections" window will appear:

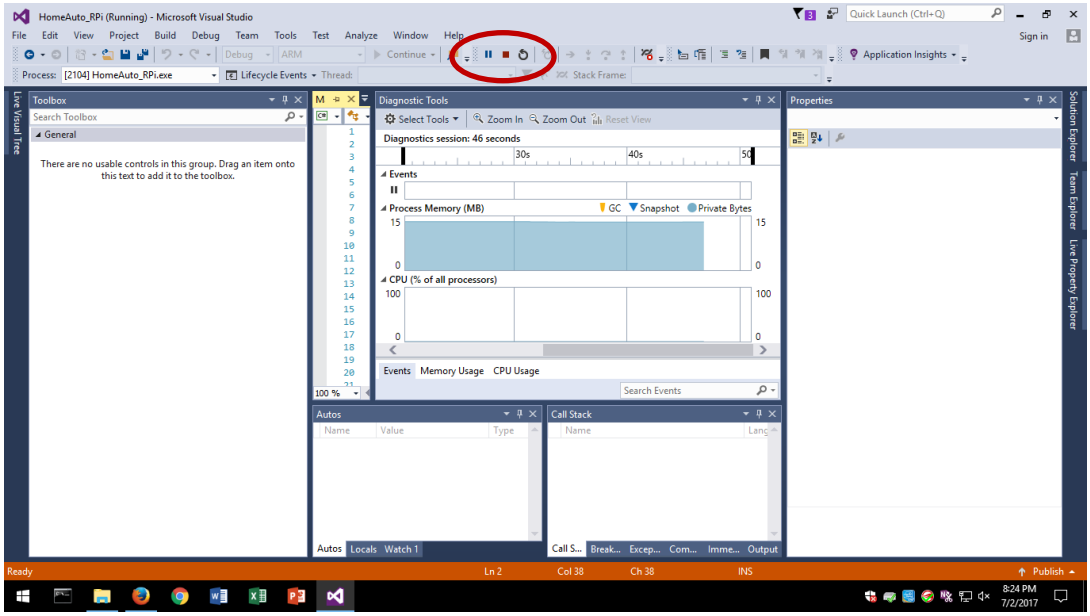


Visual Studio will attempt to auto-detect your RPi on the network; if it is found, it will appear in the "Auto Detected" list of devices in the bottom half of this window. Choose the device which corresponds to your RPi's network address (since Visual Studio might have found more than one!) and click the "Select" button to select it as the target. If your RPi is not automatically detected, you can enter its address in the "Manual Configuration" area. Close this window after you have selected your device.

If you need to change your device's settings later, simply choose "Test_RPi Properties" from the "Debug" menu, and select "Debug" in the left-hand menu. You will see a configuration screen similar to the one shown on the next page:

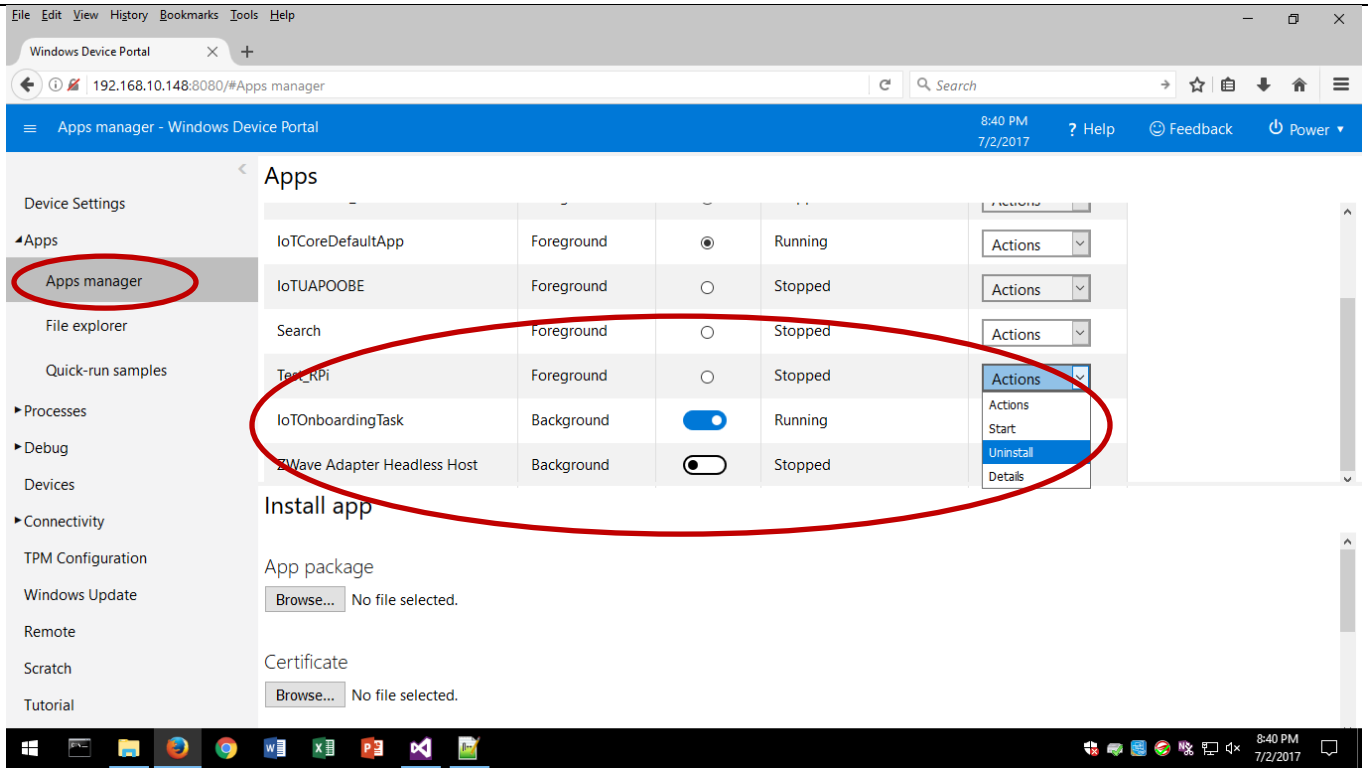


11) After choosing these options, click the "Remote Machine" button in the toolbar or press **F5** on your keyboard to build and deploy your application to the Raspberry Pi. Watch the display connected to your RPi, and you should soon see your sample application! On your workstation, a screen resembling the following will appear:



Visual Studio is monitoring the sample application running on your RPi. To close the application, click the red "Stop" button in the toolbar (circled above).

12) Notice that you cannot open or close applications on the RPi itself. If you still have the Windows Device Portal open in your Web browser, select "Apps manager" from the left-hand menu. You will see a list of installed applications, resembling the list shown on the next page:



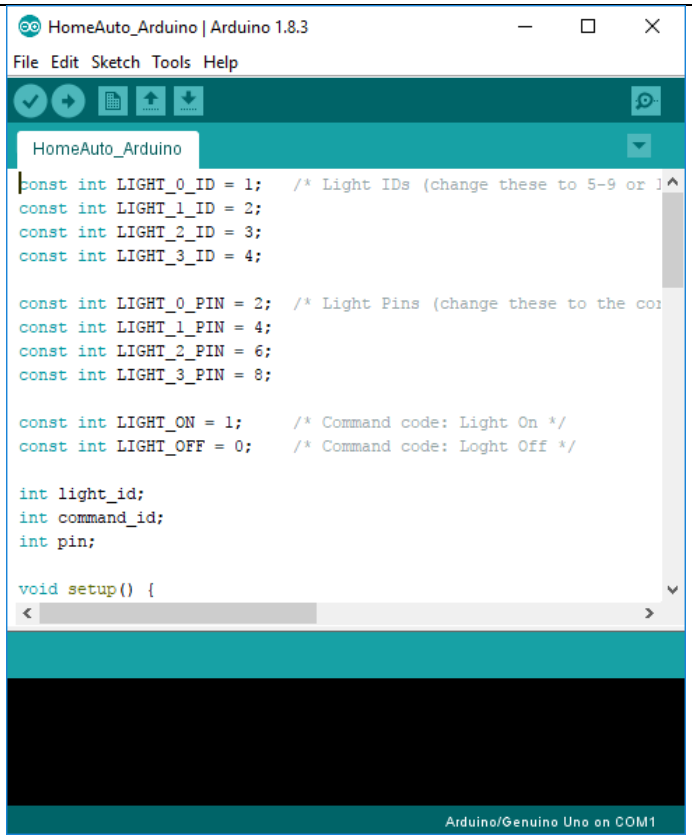
Notice that "**Test_RPi**" (our sample application) is included in the list. From the drop-down menu shown above, we can start or stop the application, or even remove it from the RPi if we do not want it any longer.

Part Five: Using the Raspberry Pi and Arduino Together

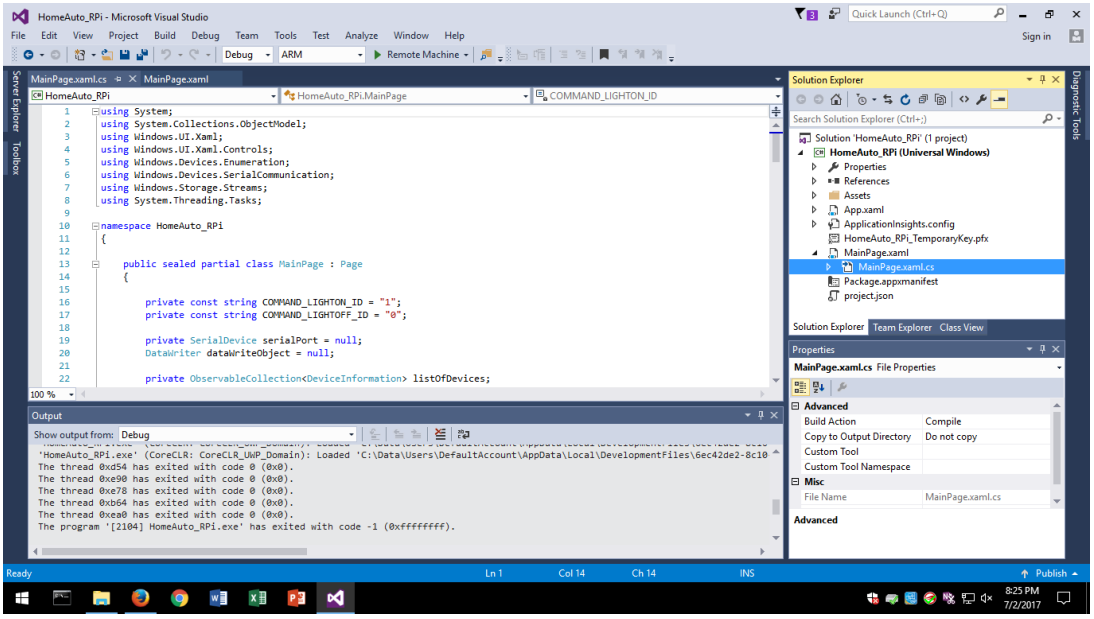
Now that we have some experience with programming our embedded platforms, let's use them together to create a simple Internet of Things application. This sample application simulates a small home automation system, which allows lights in a residential house to be turned on and off under computer control. The LED array that we constructed earlier for our Arduino will serve as a "stand in" for the lights; the Raspberry Pi will act as the "brain box" that will control the lights on one or more Arduino boards.

To begin, double-click the "**HomeAuto_Arduino.ino**" sketch file in the "**Module10**" folder to open it in the Arduino IDE. As described earlier, make sure your Arduino's USB cable is connected to your workstation, then click the "right arrow" button to compile and download the code to your Arduino board. After this process is complete, disconnect the USB cable from the PC and plug it into one of the spare USB ports on the Raspberry Pi.

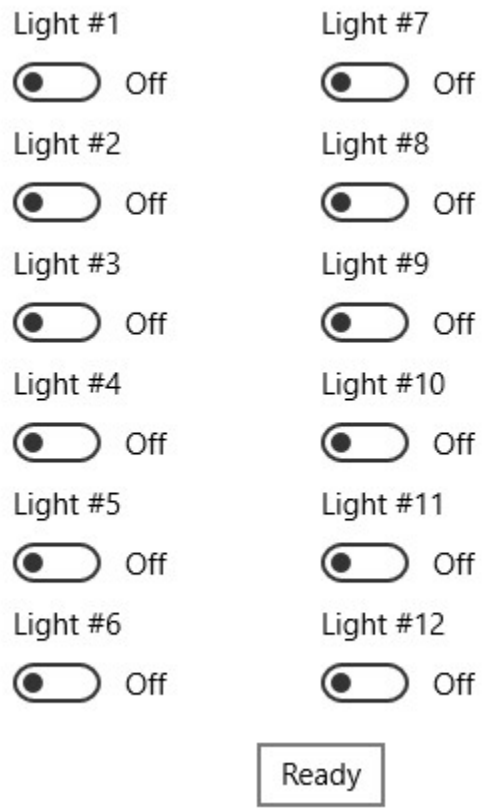
(The new software you have written to the Arduino will wait for input from the Raspberry Pi before turning any LEDs on or off, so at first, none of the LEDs will be lit. Connecting the Arduino to your Raspberry Pi will enable the RPi to communicate with it through its *serial interface*. Later, you will notice that the "**TX**" and "**RX**" indicators on the Arduino board will blink slightly as it receives commands from the RPi.)



Next, open the "**HomeAuto_RPi**" folder and find the "**HomeAuto_RPi.sln**" file. This is the Visual Studio *solution file*; double-click it to reopen the project saved in this folder. After it is reopened, it will appear in Visual Studio as follows:



Check your device settings to ensure that your Raspberry Pi is still set as the target device, then click "Remote Machine" in the toolbar or press **F5** to build and deploy the application, as you did before. The following screen will appear shortly afterward on your RPi display:



This is a simple control application which will control the lights on your Arduino through the on-screen toggle switches. The Arduino sketch that you downloaded earlier has assigned the LEDs as Lights #1 through #4. Try toggling the on-screen switches on the RPi, and you should see the LEDs turn on and off. (There will be a slight delay as the Arduino processes your input, so don't flip the switches too quickly!)

If you look at the first few lines of the Arduino sketch shown on the last page, you will notice that they assign numeric IDs and the corresponding pin numbers to LIGHT_0 through LIGHT_3. If you wish to control more lights, you can add a second or third Arduino to your RPi; simply change the IDs in the sketch to the next sequence of IDs (5 through 8 for the second Arduino, or 9 through 12 for the third Arduino), make sure that the corresponding pin numbers are set correctly, and download the revised sketches to the new Arduino boards.

*Be sure that you connect the new Arduinos **before** starting the RPi control application!* If the application is already running, close it through the Web interface, connect the new Arduinos, and *then* re-start it. If the IDs are set correctly, you should be able to control up to twelve lights, four for each Arduino board.

See the comments in the code (enclosed in **`/* ... */`** tags), in both Visual Studio and the Arduino IDE, for more information about how to customize this program for your own needs. Once you are familiar with the code and with the C# language, it should not be difficult to extend the control program to add more features, such as (for example) switching the lights on or off automatically at preprogrammed intervals.

Acknowledgements: