ICS005 HMI Development

Lab Objective

The objective of this lab is to understand the basics of Human Machine Interface (HMI) development, including the use of the Modbus/TCP protocol in conjunction with the AdvancedHMI libraries and Microsoft Visual Basic to implement a complete HMI solution. These instructions are written for the DirectLOGIC 205 PLC included in your ICS lab kit; for the benefit of students and instructors working without their own kits, instructions are also provided for the Do-more Simulator.

In this lab, you will learn to:

- Configure your PLC for Ethernet communication using the Modbus/TCP protocol
- Use the Modbus coils for shared read/write memory access
- Develop an HMI application using Microsoft Visual Basic and AdvancedHMI

Lab Environment

This lab requires a development workstation configured with Microsoft Visual Studio, Do-more Designer, and the AdvancedHMI library archive. These instructions were written for Microsoft Visual Studio Community Edition 2015, Do-more Designer 1.4.X, and AdvancedHMI 3.99t; other editions and/or versions of these tools may require slight changes.

Lab Duration

45 minutes

Lab Tasks

As a "warm-up exercise", create a simple Modbus/TCP firmware program in ladder logic and an accompanying HMI application. Then, create a more sophisticated ladder logic program and HMI application to simulate a traffic light control system.

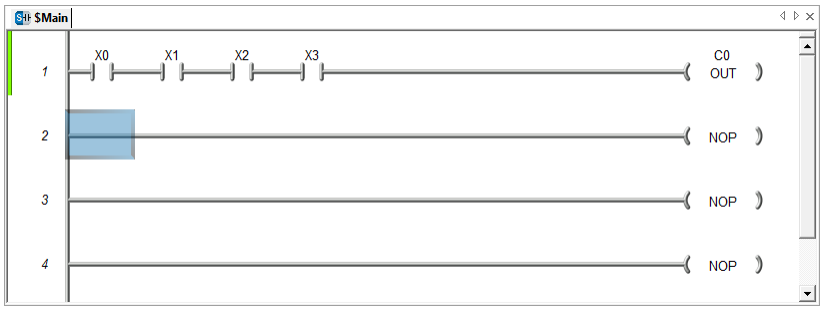
Parts One and Two of this exercise correspond to the "warm-up" ladder logic program and HMI application, respectively; the remaining three parts are devoted to the traffic light control system. It is recommended that Parts One and Two should be completed first, in that order.

Parts Three and Four can be completed interchangeably; that is, a student or instructor who wishes to complete the traffic light HMI application first can start with Part Four instead of Part Three. Parts Three and Four can also be completed in parallel as part of a team project. Part Five is an optional "emergency mode" feature which can be omitted to save time; if attempted, it should be deferred until after Parts Three and Four are completed.

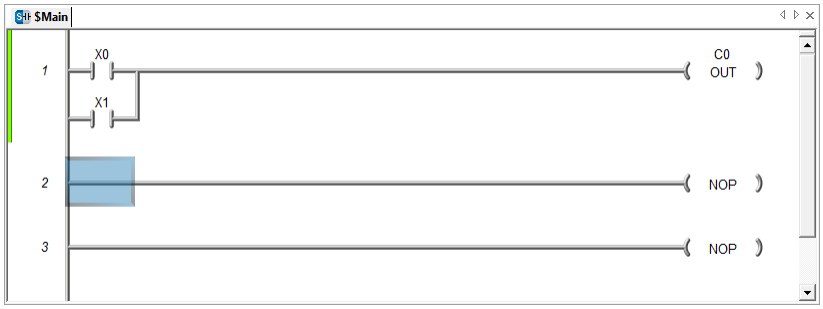
Background

Ladder logic is a graphical circuit diagram of relay logic hardware. It was originally a method of designing the construction of relay racks for manufacturing and process control, but thanks to tools such as Do-more Designer, ladder logic can now be used as a visual design language for PLC firmware. In a ladder logic program, each *rung* of the ladder represents a single *rule* in the program and specifies an action of some kind; typically, the source(s) of input appear on the left side of the rung and the destination(s) for output appear on the right. Sometimes, the output from one rule will appear at the input of a subsequent rule. These rules, taken together, model the intended behavior of the control system.

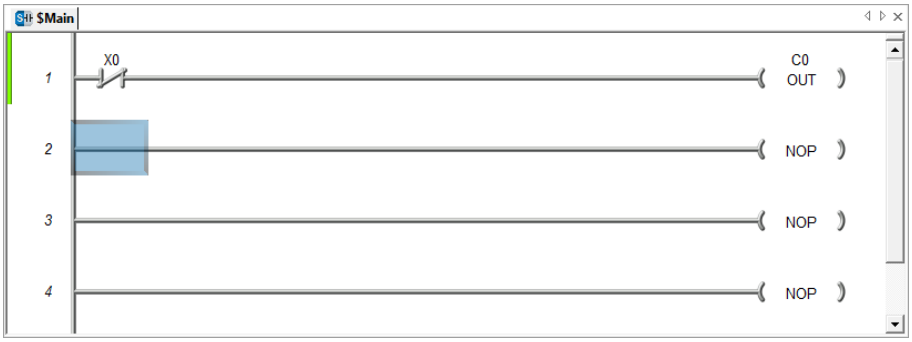
To those who are accustomed to writing computer programs using such languages as C, C++, or Java, programming with ladder logic may seem unfamiliar at first. However, using ladder logic, we can model all the fundamental logical operations—such as AND, OR, and NOT—using combinations of *contacts* (or *switches*) and *coils* (single-bit memory locations) as *operands*, in much the same way that Boolean variables are used in traditional programming languages.



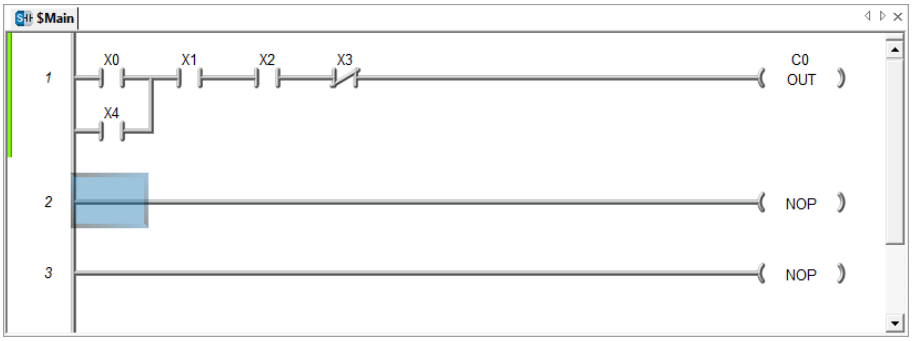
For example, above are four *normally open contacts* mapped to **X0** through **X3** (the first four input switches on the PLC). These switches are the *inputs*, and **C0** (one of the C-memory bits in the PLC) is the *output*. The inputs are arranged *in series*, so **C0** is TRUE only if *all* of the inputs are TRUE (that is, if all four switches are *on*). To put it another way, we can think of the rung as a wire in a circuit, which is closed only if all four switches are closed. This corresponds to logical AND.



Above are two *normally open contacts* mapped to **X0** and **X1**. This time, the inputs are arranged *in parallel*, so **C0** (the associated output) is TRUE if *any* of the inputs are TRUE (that is, if *one* of the switches is on, *or* if *both* of them are on). This corresponds to logical OR.



Finally, here is an input mapped to **X0**, but notice that this is a *normally closed contact* rather than a *normally open contact*. This means that **C0** is TRUE only if the input is FALSE (that is, if the switch **X0** is switched *off*); when we switch **X0** *on*, the input is TRUE and **C0** is FALSE. This corresponds to logical NOT.



Of course, we can combine these logical operations on a single rung. This rung can be thought of as a kind of "function" with its own inputs (**X0** through **X4**) and output (**C0**), representing a solution to one part of a problem. When the *output* of this rung is used as an *input* on another rung, it becomes part of a larger solution. This is an application of the classic "divide and conquer" strategy. The various contacts, coils, and other inputs and outputs shown in this rung can be easily added to the ladder from the palette of tools provided by Do-more Designer.

These examples also illustrate several of the *nicknames* defined by Do-more Designer for the built-in PLC I/O devices, memory locations, timers, counters, and other integrated peripherals. These include:

- **Input Switches**: **X0** through **X7**. These one-bit inputs correspond to the toggle switches on the DirectLOGIC 205 included in your ICS lab kit.
- **Output Indicators**: **Y0** through **Y7**. These one-bit outputs correspond to the output LEDs on the DirectLOGIC 205.
- **C-Memory**: **C0** through **C23**. You can consider these to be Boolean variables, or one-bit registers, which can be used for input or output.
- **Integer and Real Number Memory** (8 each): **V-Memory** (Unsigned Word), **D-Memory** (Signed DWord), and **R-Memory** (Real). These registers are used to store numeric values of different types.
- **Counters** (**CT0** through **CT2**) and **Timers** (**T0** through **T2**).

- **ST-Bits (System Assigned Bits)**, which include:
    - **ST3** ($1Minute): While the system is in RUN mode, this bit will be set to ON *once per minute* with a 50% duty cycle; that is, it will be ON for 30 seconds, then OFF for 30 seconds.
    - **ST4** ($1Second): The same as **ST3**, but with a one second cycle; that is, it will be ON for half a second, then OFF for half a second.
    - **ST5** ($100ms): The same as **ST4**, but with a 100 millisecond cycle; that is, it will be ON for 50 milliseconds, then OFF for 50 milliseconds.
    - **ST6** ($50ms): The same as **ST5**, but with a 50 millisecond cycle; that is, it will be ON for 25 milliseconds, then OFF for 25 milliseconds.

We will have the opportunity to see several of these in action in the exercises that follow. For additional examples, see the corresponding lecture notes, which include sample problems and complete solutions in ladder logic.

Lab Scenario

The ideal environment for completing this lab is a Windows workstation with the necessary tools installed (see "Lab Environment"). These instructions are written under the assumption that the workstation is connected to the router in the ICS lab kit, along with the DirectLOGIC 205 PLC included in the kit. However, students or instructors wishing to complete these exercises without the PLC kit can do so entirely using the Do-more Simulator, which is included with the Do-more Designer and emulates all the functionality of a PLC, including network I/O. To open the Do-more Simulator, click the "Do-more/Sim" button in the Do-more Designer toolbar.

Throughout the remainder of this document, you will see specific instructions for Do-more Simulator users. After completing an exercise, it may be useful to switch an existing Do-more Designer project from the Do-more Simulator to the DirectLOGIC 205 for testing on "real hardware"; to do so, simply follow these steps:

- Connect the PLC to your workstation via the USB programming interface
- Choose "Disconnect" from the "PLC" menu and close the Do-more Simulator
- Choose "Connect" from the "PLC" menu and select the interface corresponding to your PLC.

You may get several error messages along the way, indicating that the firmware in the PLC differs from the project opened in Do-more Designer. If you get a "**Resolve Online/Offline Differences**" error message, choose "**Go Online and View the DISK Project**".

Lab Procedures

Part One: PLC Configuration

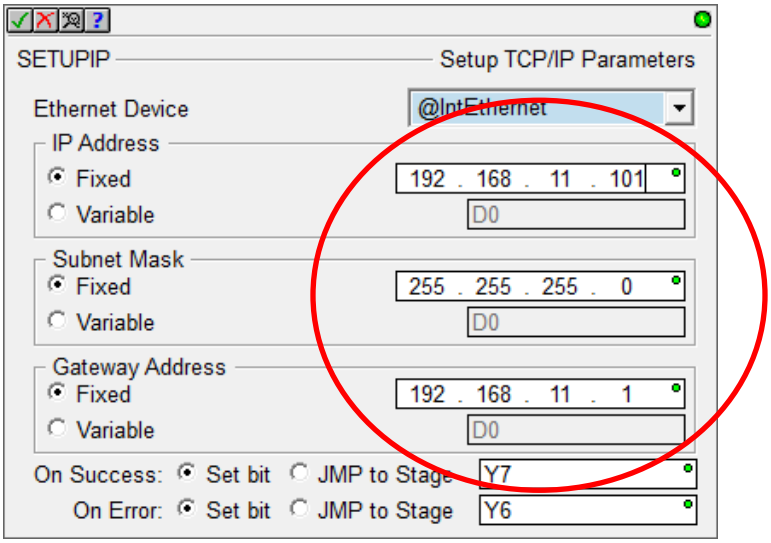
These steps will walk you through the process of preparing your PLC, and of developing a simple HMI application using Modbus/TCP. Before we build the HMI application, however, we will need to download some simple firmware to the PLC. Among other things, this firmware will initialize the PLC's Ethernet port, allowing HMI applications to establish a connection with it over the network using the Modbus/TCP protocol.

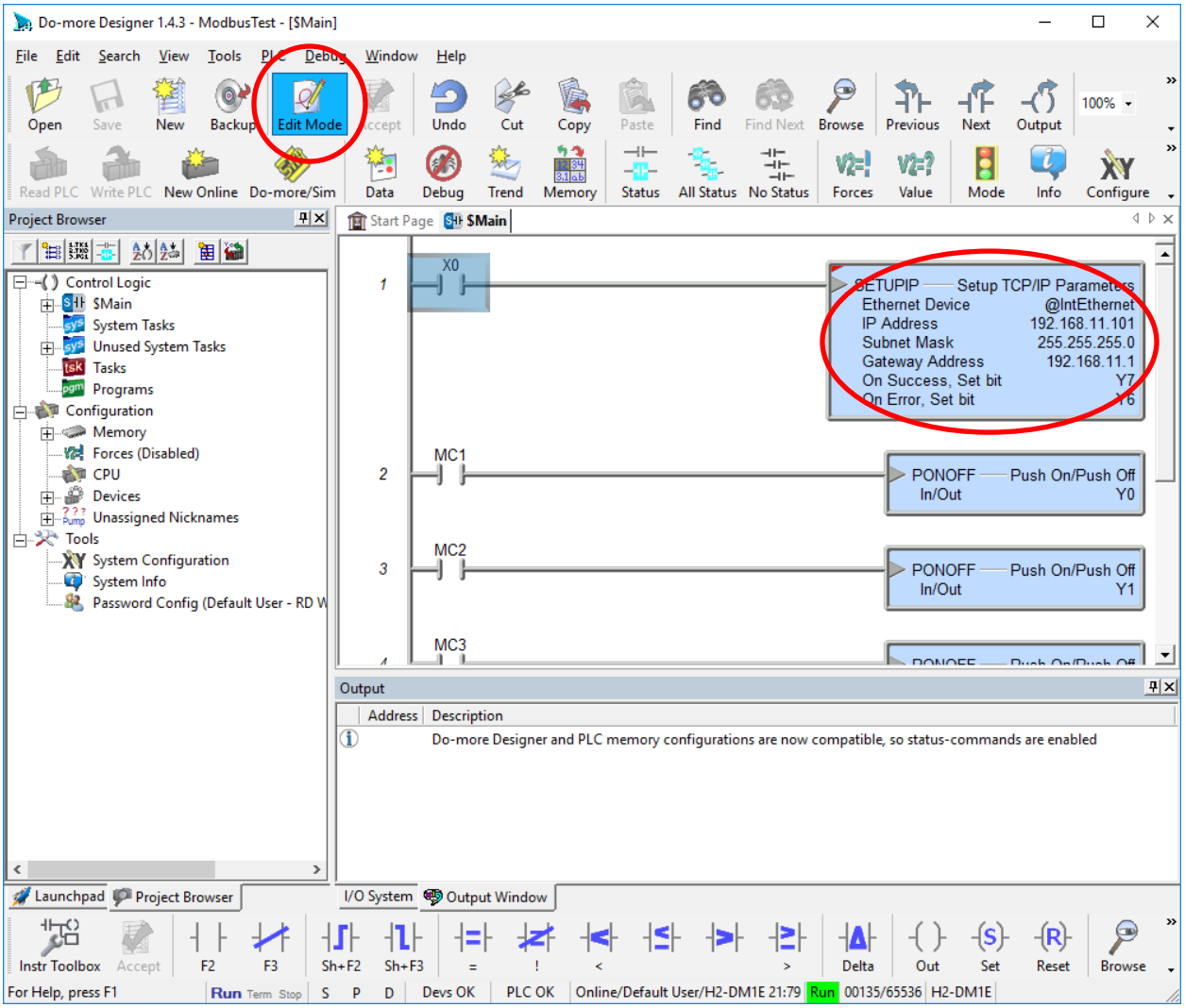
First, connect your workstation to the USB port of your PLC using a standard USB 2.0 A/B cable. Next, ensure that the Ethernet ports of both the PLC and the workstation are connected to the router in your ICS lab kit. (These instructions assume that the router is configured with the factory settings; if you have changed the IP configuration of your router, you may need to make the corresponding changes to the IP addresses used in these instructions.)

Follow these instructions to download the firmware to the PLC:

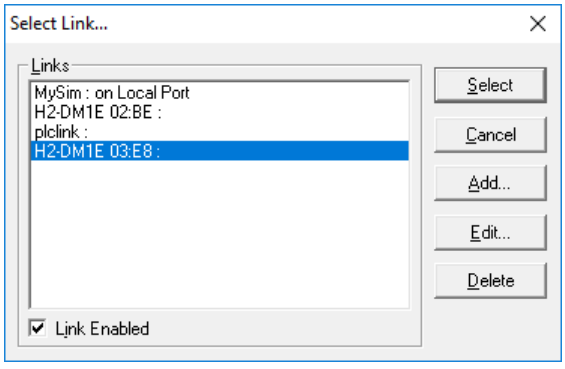
1. Extract the provided "**ModbusTest**" archive to a folder on the workstation.
2. Launch Do-more Designer 1.4.X. From the "File" menu, choose "Open Project". Browse to the "**ModbusTest**" folder and select the project file, "**ModbusTest.dmd**".
3. The project should appear as shown in the screenshot at the top of the next page.

   Notice that the "Setup TCP/IP Parameters" tool includes the *IP address* (currently set to "**192.168.11.101**"), *subnet mask*, and *gateway address* that will be assigned to the PLC. If these settings must be changed for your network, click the "Edit Mode" button in the toolbar to switch to edit mode, then double-click the "Setup TCP/IP Parameters" tool (both are circled in the screenshot on the next page). Change the network settings as necessary, then click the "Accept" button (the green checkbox in the upper-left) to accept your changes.



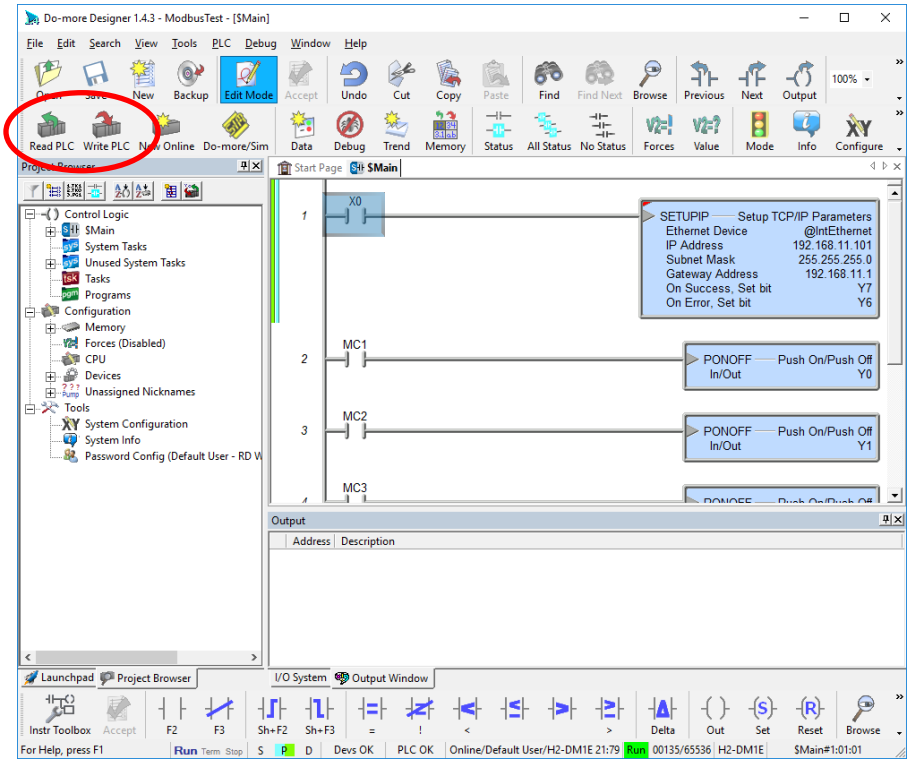


4. Open a connection to the PLC by choosing "Connect ..." from the "PLC" menu. A list of connection options will be shown; the options that you see may be different from the ones shown below. If you are testing this firmware on the PLC in your ICS lab kit, choose the "H2-DM1E" option; if you are using the Do-more Simulator, choose the "MySim" option instead (and ensure that the simulator is running before proceeding!). Next, click "Select":



Do-more Designer will now attempt to establish a connection with the PLC. If you get a "**Resolve Online/Offline Differences**" dialog box, choose "**Go Online and View the DISK Project**". If there are any difficulties connecting, double-check your settings and connections, and try again.

5. After the connection has been established, the "**Read PLC**" and "**Write PLC**" buttons should become available in the toolbar:



Check to ensure that the "Mode" switch on the PLC (directly above the USB program port) or in the simulator is set to "TERM" mode (the middle position), then click the "Write PLC" button. The new firmware will be written to the PLC. If you see a "**Download Project to PLC**" dialog box, check the "**Switch back to RUN mode after download completes**" checkbox and select the "**Switch to PROGRAM mode, then Download Project**" option.

6. Once the download is complete, set the "Mode" switch to "RUN". The "RUN" LED indicator, directly below the power indicator, should now be lit. Locate the eight input switches (which will appear as buttons in the simulator) and set Switch 0 to "ON" to initiate the network configuration. If it was successful, the output LED 7 should be lit. To test connectivity, open a Command Prompt window and attempt to ping the IP address assigned to the PLC by entering the following command:

   ```
   ping 192.168.11.101
   ```

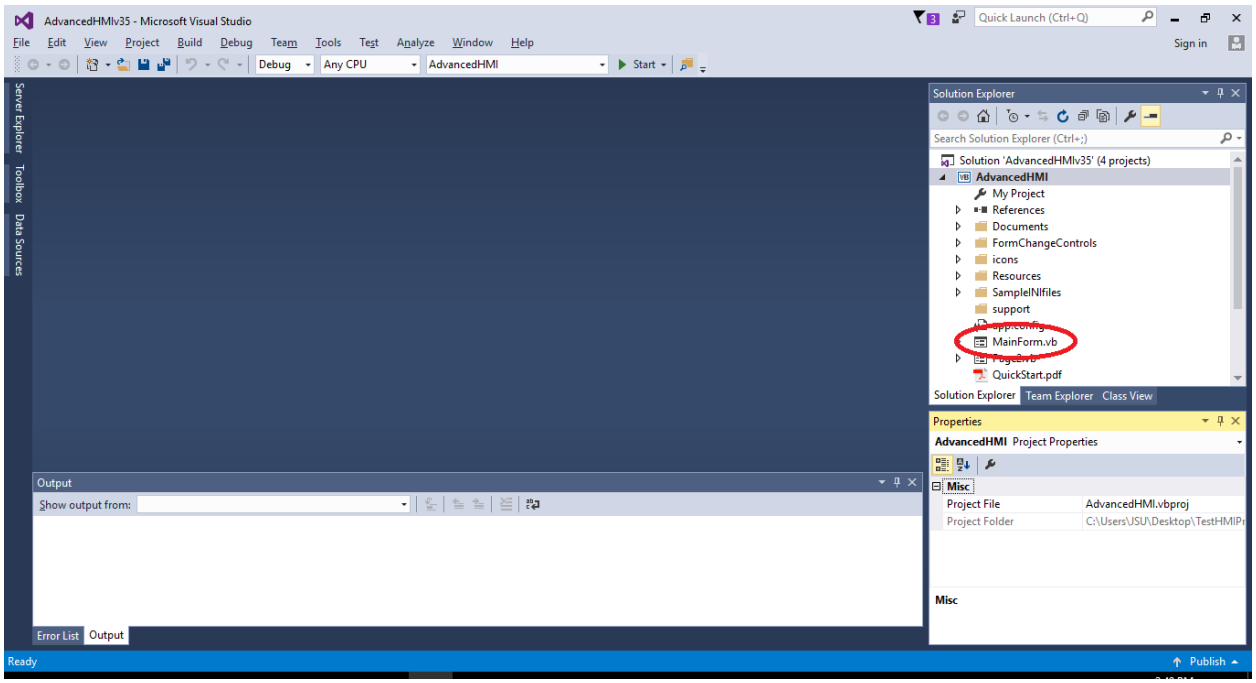
   If your network switch permits ping traffic, you should receive a response from the PLC. *(This ping test is not necessary if you are using the Do-more Simulator.)*

Part Two: Introduction to HMI Application Development

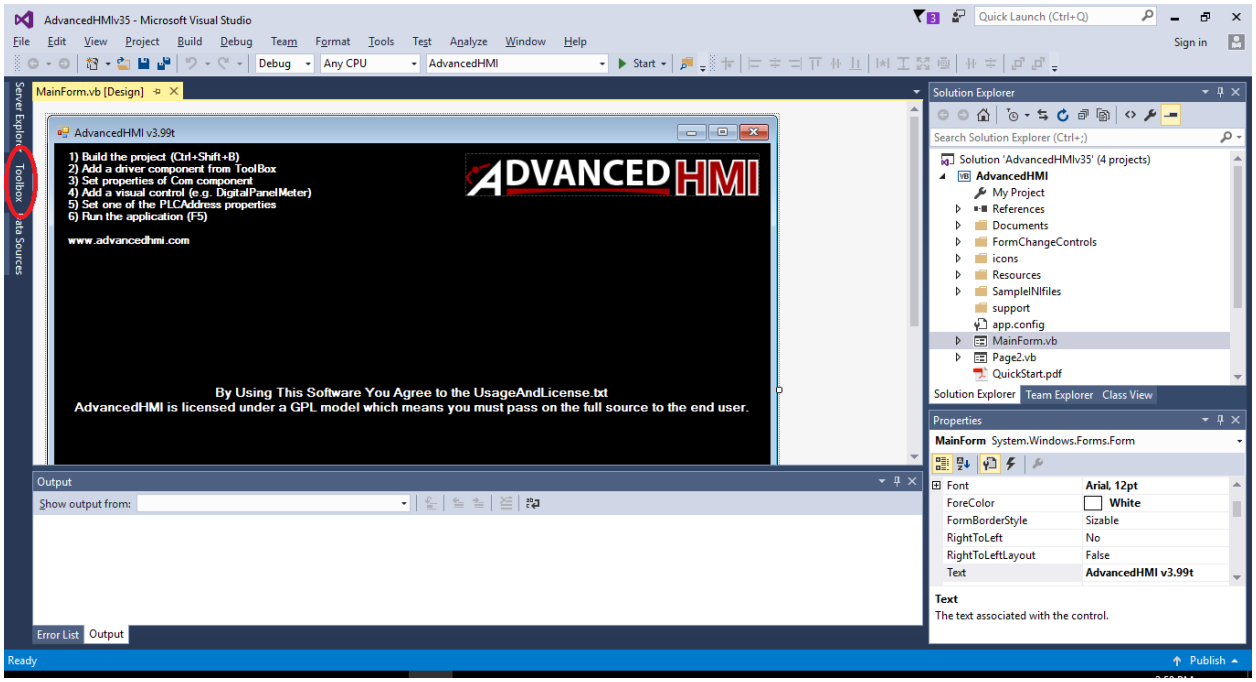
Your PLC is now ready to interact with an HMI application. If you are using the Do-more Simulator, *leave the simulator open and running before proceeding*. The steps below will walk you through the creation of a very simple HMI application using the AdvancedHMI tools. The latest version of AdvancedHMI is provided in the file collection included with this lab exercise, and is also available for download from SourceForge:

https://sourceforge.net/projects/advancedhmi/

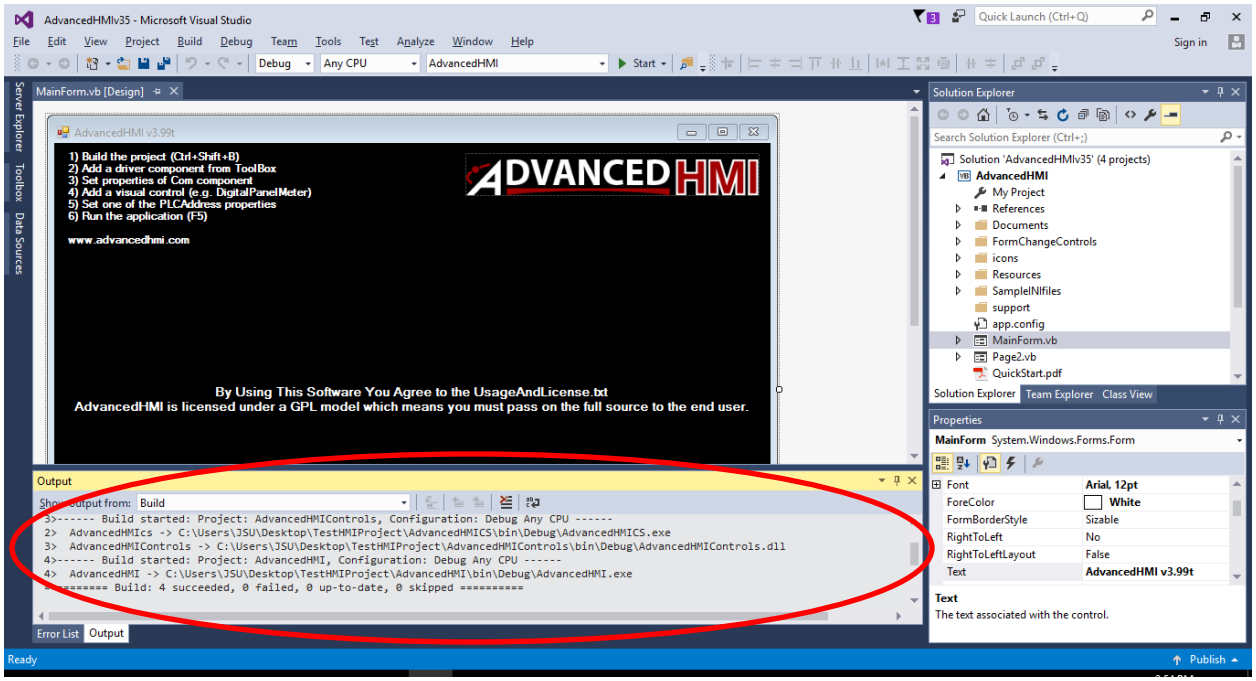
1. If you do not already have it saved to your workstation, download the Advanced HMI solution archive, "**AdvancedHMIv399t.zip**", from SourceForge (the URL is provided above). Save the archive to your desktop.
2. *Extract* the archive to a new folder, then *rename* this folder to match the name of your new application. In this example, we will use the name "**TestHMIApplication**".
3. Double-click the solution file, "**AdvancedHMIv35.sln**", inside this folder. This will open the complete solution in Visual Studio; depending on your workstation, this may take a few minutes. If you see a "**Security Warning for AdvancedHMI**" dialog box, clear the "**Ask me for every project in this solution**" checkbox, then click "OK."
4. After the solution is opened, your screen should resemble the following:



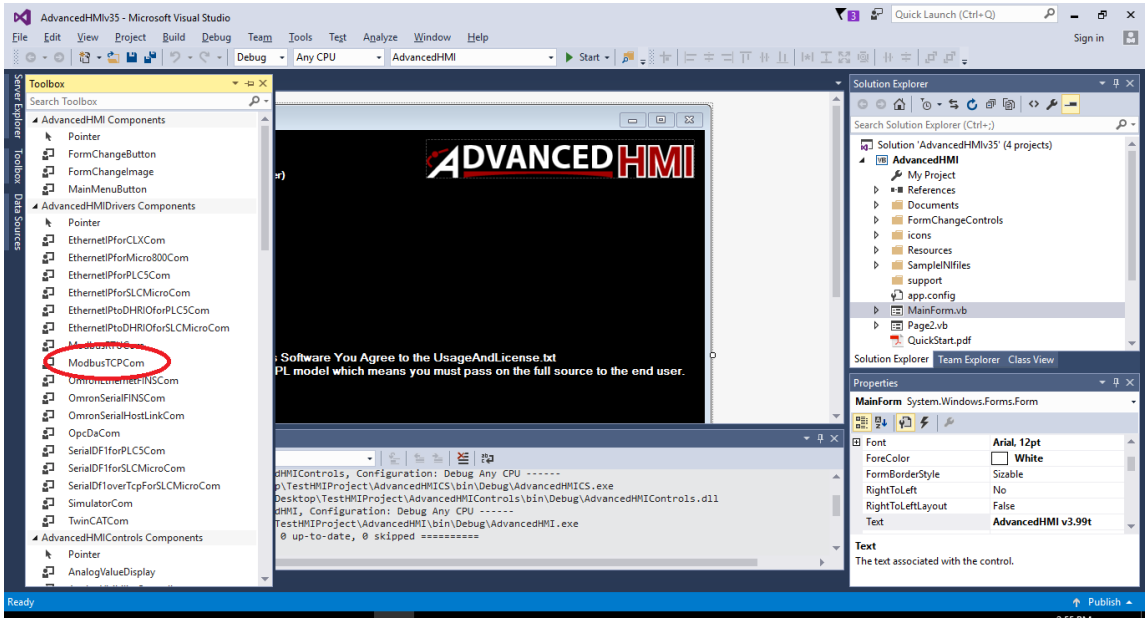
5. Notice the "**MainForm.vb**" file shown in the "Solution Explorer" window (it is circled in the screenshot). Double-click this file, and you should see the main form on your screen, as shown on the next page:



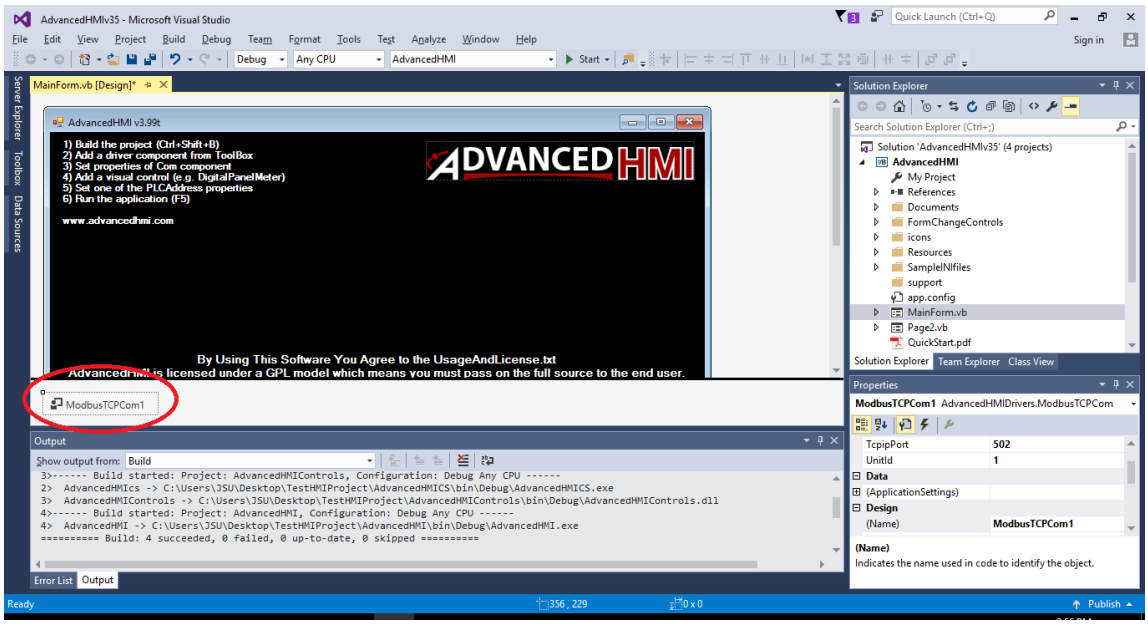
The instructions in the upper-left corner of this form outline the recommended procedure for creating an HMI application. First, press "**CTRL-SHIFT-B**" as instructed to *build* the solution. This will populate the toolbox (circled in the above screenshot) with the AdvancedHMI tools and controls. After pressing "**CTRL-SHIFT-B**", watch the "**Output**" window at the bottom of the screen. When the build completes, you should see a message similar to the following: "**Build: 4 succeeded, 0 failed, 0 up-to-date, 0 skipped**":



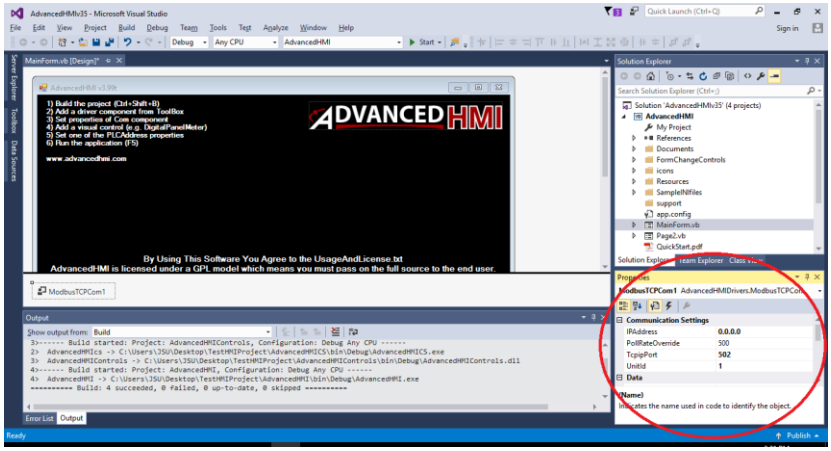
6. After the build is finished, click the "Toolbox" tab on the left edge of the screen. You should see a large selection of "AdvancedHMI" and "AdvancedHMIDrivers" components:



From the "AdvancedHMIDrivers Components" list, look for a "ModbusTCPCom" component (circled in the above screenshot). This is the "Com component" mentioned in the procedure outlined in the form. *Click and drag* this component into the form, and it should appear in the area directly *below* it, with the name "**ModbusTCPCom1**":

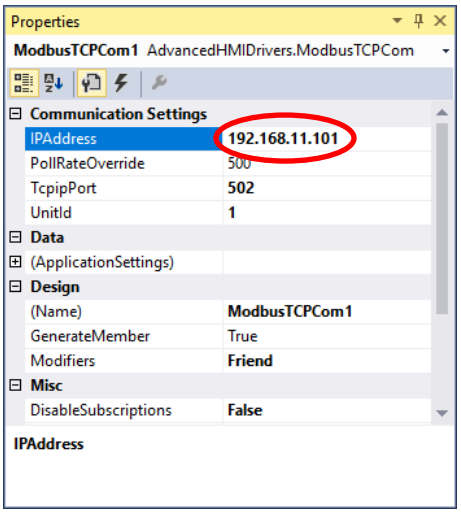


7. Click the "**ModbusTCPCom1**" component to select it, and its properties will appear in the "Properties" area in the lower-right corner of the window:



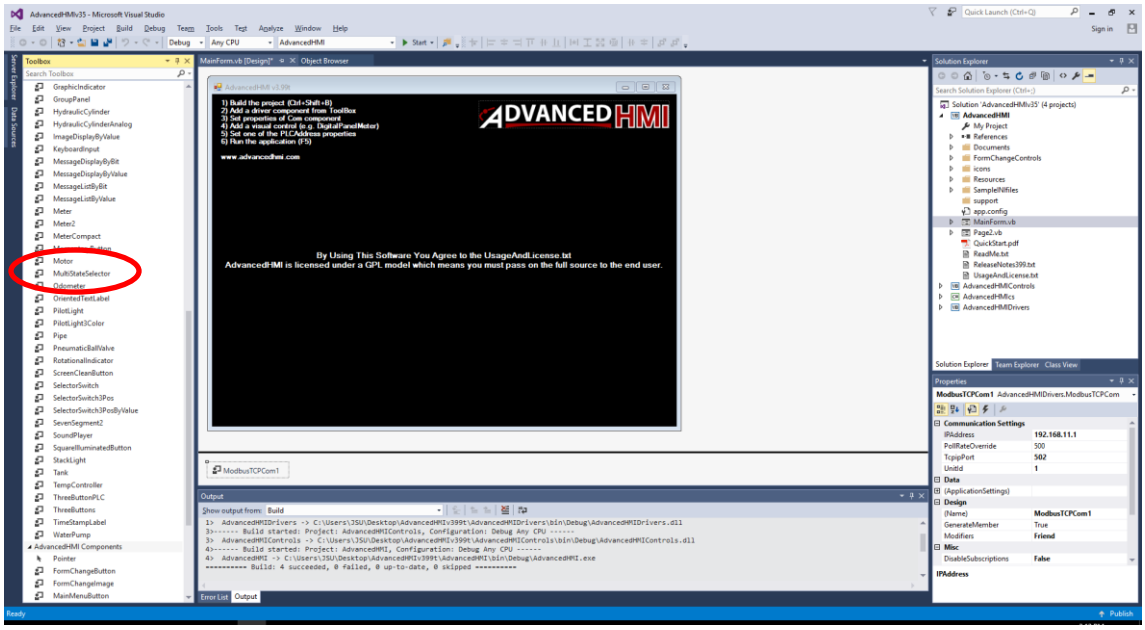
Scroll up to the top of the "Properties" area. The first in the list of properties should be "**IPAddress**". This is the IP address that your HMI application will attempt to use to communicate with the PLC (or the simulator).

To properly configure the application to connect to the PLC in your ICS lab kit, the default IP address "**0.0.0.0**" must be changed to the IP address already assigned to the PLC. Recall that the project in Do-more Designer assigned an IP address of "**192.168.11.101**", so this is the address that you should use. (If you changed this address in Do-more Designer to a different one, enter that same address into the "IPAddress" field now.)

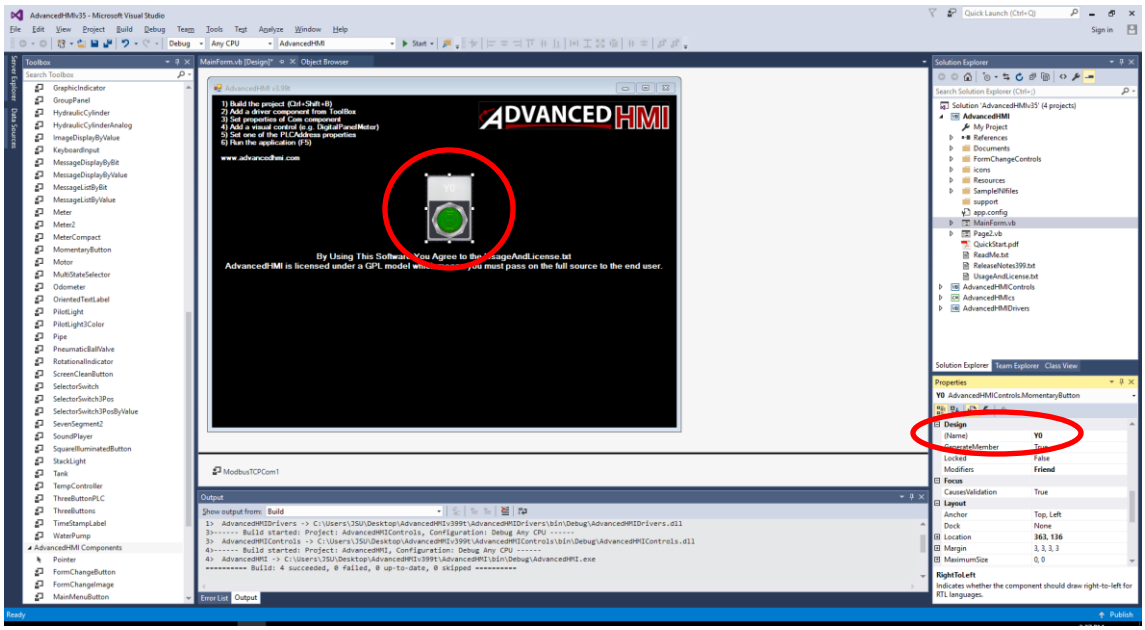


If you are using the Do-more Simulator, enter the IP address of your *workstation* instead. If you do not know your IP address, open a Command Prompt window, enter the command **`ipconfig`**, and use the IP address shown in the properties of your network interface.

8. Next, reopen the "Toolbox" and scroll down through the list of "AdvancedHMIControls Components" (you may need to expand this group if it is not expanded already). Look for a control called "**MomentaryButton**":

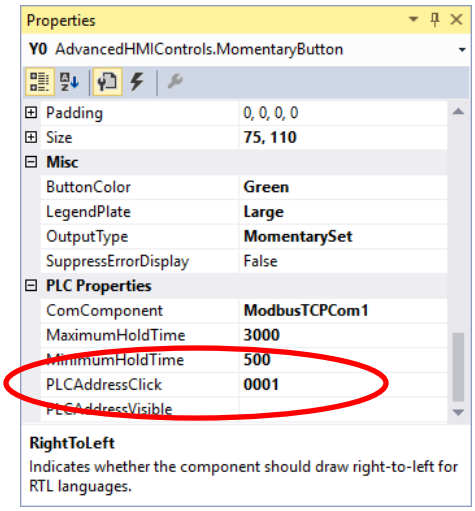


9. *Click and drag* this component anywhere into the form to add a button to the HMI application. It should appear in the form as follows:

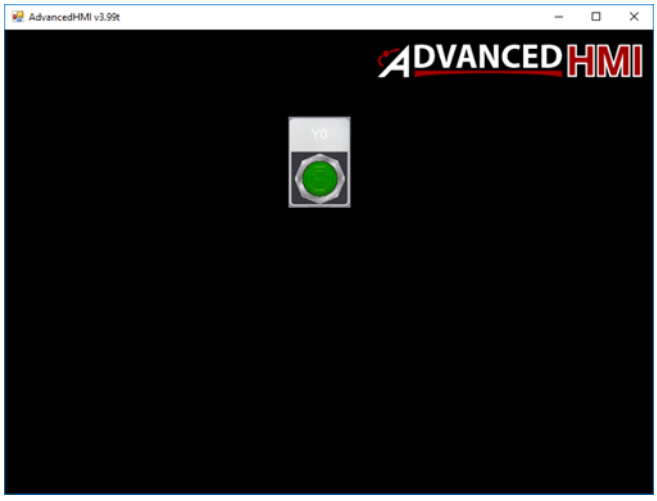


Select the new button control if it is not automatically selected, then look at its properties in the area in the lower-right. Two of the properties shown are "**Name**" and "**Text**". Change the value of *both* of these properties from the default "**MomentaryButton1**" to "**Y0**" (this is the PLC output that we will be assigning to it in a moment).

Next, scroll down to the *bottom* of the "Properties" window and look for a field called "**PLCAddressClick**". This field should have no assigned value. Enter the value "**0001**" into this field. None of the other properties should need to be changed.



10. Finally, click the "Start" button in the toolbar (circled in the previous screenshot). The HMI application will be built and launched, and in a moment, the form should appear on screen as follows:



11. Again, double-check the PLC. If it is connected to the network and set to "RUN" mode, and if the input Switch 0 set to the "On" position, then output LED 7 should be lit, indicating that the LAN configuration was applied successfully. If you are using the Do-more Simulator, be sure it is running *and* set to "RUN" mode *before* launching your HMI application.

    Click the on-screen button that you added to the HMI application, and you should see that the first output LED (**Y0**) on the PLC or in the simulator is toggled on or off whenever the button is clicked. (If you are using the PLC in your ICS lab kit, you should also hear the relay click when the light toggles on and off.)

    Recall that, in the Do-more Designer project, the first four output LEDs (**Y0** through **Y3**) were associated with the first four Modbus coils (**MC0** through **MC3**), so if you wish, you can close the application and repeat the preceding steps to add three additional button controls for the remaining three outputs. Use "**Y1**" through "**Y3**" as the "**Name**" and "**Text**" values for the new buttons, and the "**PLCAddressClick**" values **0002** through **0004**.

    When you have confirmed that all button controls are properly toggling the corresponding outputs on the PLC, close the HMI application and Visual Studio (and the Do-more Simulator, if you are using it).

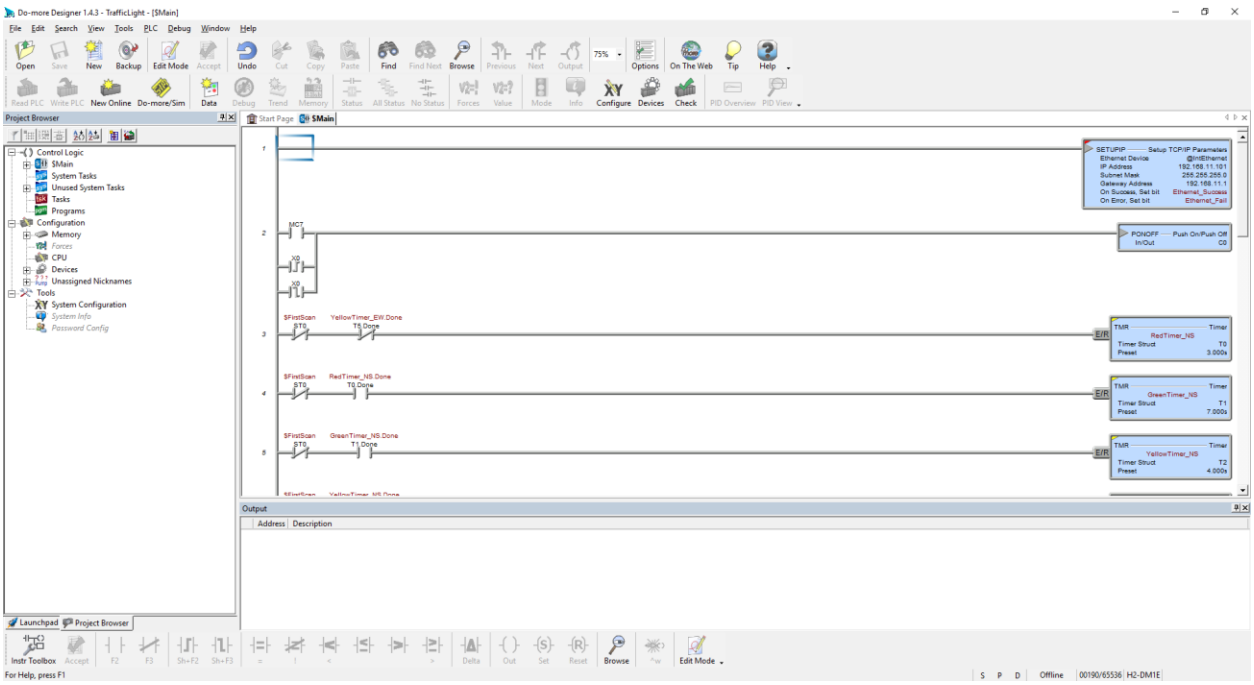
Part Three: Traffic Light Simulator (Ladder Logic)

Now that you are familiar with the basic process of creating an HMI solution, you are ready to consider the control system for our traffic light simulator. This traffic light is to be stationed at the intersection of an east-west main road and a north-south side road. The traffic lights are mapped to the output LEDs on your PLC: the N/S road lights are **Y0**, **Y1**, and **Y2**, and the E/W road lights are **Y4**, **Y5**, and **Y6** (red, yellow, and green, respectively).

To keep the implementation simple, this control system uses fixed-time control; that is, instead of using sensors to change the signal timing according to the flow of traffic, it changes the signals at fixed intervals using timers. We will complete the ladder logic for the control portion of the system, and then construct an HMI application for controlling and monitoring the system.

The attached archive, "**TrafficLight.zip**", contains an *incomplete* implementation of the ladder logic program that you can use as a starting point. Extract this archive to a separate folder on your workstation, then open the project in Do-more Designer: after reopening Do-more Designer, choose "Open Project" from the "File" menu, browse to the extracted folder, and choose the file "**TrafficLight.dmd**". The project should appear as shown on the next page.

Scroll through the project, and notice that the ladder logic program consists of twenty rungs, most of which are relatively simple. As in our earlier sample project, Rung #1 contains a TCP/IP setup instruction for preparing the PLC's Ethernet network connection. This instruction will fire automatically as soon as the ladder logic program is launched; if it completes successfully, the output LED Y7 will be lit.

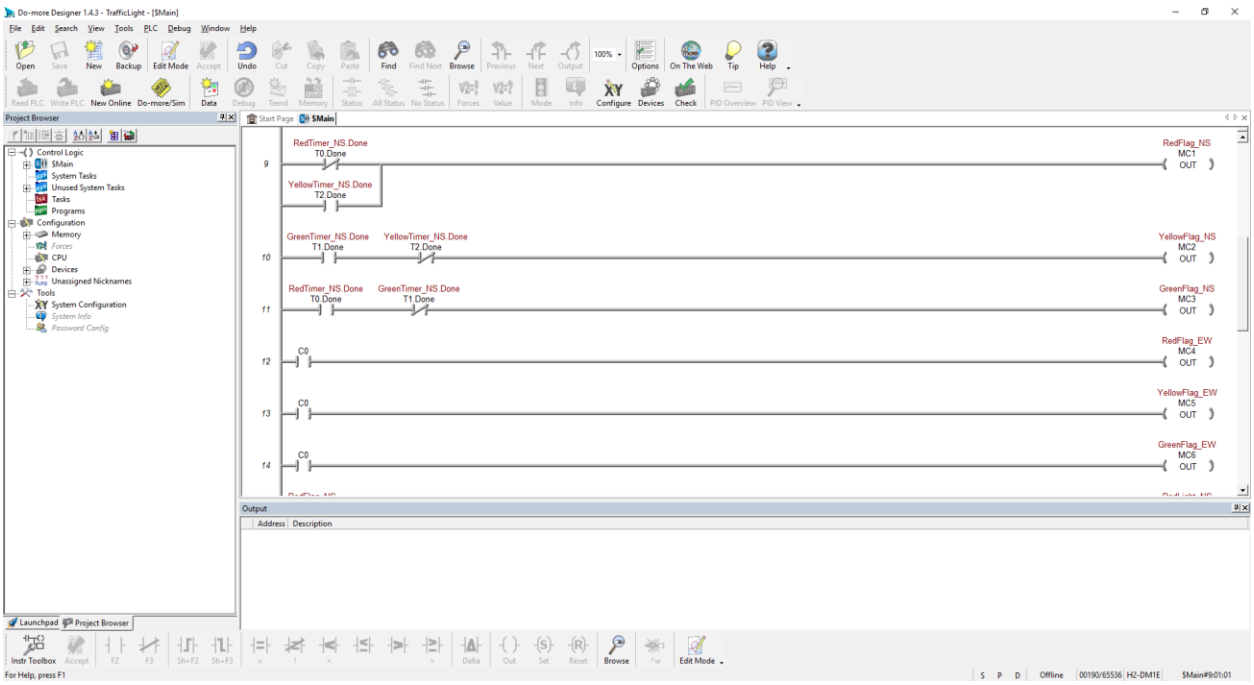


Rung #2 contains an input instruction which toggles the bit register **C0** *on* or *off* when the input switch **X0** is switched *on* or *off*, *or* when the PLC receives input from the HMI application into the Modbus coil **MC7**. This input is for the "emergency mode" switch, which we will return to (time permitting) in Part 5 of this assignment. You may disregard Rung #2 for now.

Rungs #3 through #8 contain the timers which control the phase transitions of the traffic lights. Notice that the timers have been assigned descriptive nicknames: the timer "**RedTimer_NS**" controls the red light on the N/S road, "**GreenTimer_EW**" controls the green light on the E/W road, etc. As you work on completing the ladder logic, you can refer to these timers and their members using these nicknames; for example, instead of having to remember that timer **T0** controls the red light on the N/S road, you can refer to it as "**RedTimer_NS**" instead. See the other timers for the other nicknames.

During normal operation, the traffic lights will cycle through a fixed series of phase transitions: the N/S light will transition from red to green to yellow and back to red, and then the E/W light will transition from red to green to yellow and back to red, and so on. Each phase has a preset length: the green light will be lit for *seven* seconds, the yellow light will be lit for *four* seconds, and the red lights will "overlap" for *three* seconds before the next direction changes to green.

Note that the PLC timers include a "**Done**" member which switches ON when the timer has finished counting from zero to the preset value. The inputs for the timers are already configured to implement the cycle described above: as each timer in the cycle finishes counting, it will automatically trigger the next timer. Notice also that the timer inputs also contain a normally open contact from **ST0** ("$FirstScan"); this is a one-time "click" generated by the PLC at the start of execution, and here, it is being used to initialize and synchronize the timers on startup.



The next three rungs (#9 through #11, shown above) are the logic to control the lights on the N/S road. These rungs are already complete, so if you write the program to your PLC and run it, you should see the N/S road lights executing the cycle. Study these rungs to understand the sequence of logic; you can use them as a model for completing the following three rungs.

Rungs #12 through #14 are the logic to control the lights on the E/W road. These are the rungs that you will need to complete. The inputs to these rungs contain a normally open contact from C0, but this is only a placeholder; this contact will need to be deleted *before* you start your work on these rungs. Notice that all six of these rungs output to the registers MC1 through MC6; these are Modbus Coil registers, which are single-bit memory similar to the C-memory registers you have seen before. We are using these registers, which are also assigned descriptive nicknames to make them easier to remember, because we also want to be able to monitor the lights through Modbus/TCP. We will return to this aspect in Part 4, when we are ready to build the HMI application.

The remaining rungs in the program tie the Modbus registers for the lights to the corresponding output LEDs; these rungs are complete and do not need to be changed.

Before working on the ladder logic for the E/W road, you may wish to review the lecture notes corresponding to this lab—particularly the example problems, each of which is given with a complete solution. Remember that *switches in series* correspond to a logical **AND**, and that *switches in parallel* correspond to a logical **OR**. To implement a logical **NOT**, use a *normally closed contact*. Remember also that, to draw wires in Do-more Designer, click the selector to the desired spot on the rung, *hold down* the CTRL key, and use the arrows keys on your keyboard to draw the wire. To erase a wire you've drawn, use CTRL-SHIFT with the arrows.

To complete the logic for the red light (Rung #12), implement the following logic:

RedTimer.NS **IS NOT** Done
OR RedTimer_NS **IS** Done **AND** GreenTimer_NS **IS NOT** Done
OR RedTimer_NS **IS** Done **AND** GreenTimer_NS **IS** Done **AND** YellowTimer_NS **IS NOT** Done
OR YellowTimer_NS **IS** Done **AND** RedTimer_EW **IS NOT** Done

To complete the logic for the yellow light (Rung #13), implement the following logic:

GreenTimer_EW **IS** Done **AND** YellowTimer_EW **IS NOT** Done

To complete the logic for the green light (Rung #14), implement the following logic:

RedTimer_EW **IS** Done AND GreenTimer_EW **IS NOT** Done

(Of course, there are several possible ways to complete the logic for this traffic light simulator; the solution given above is merely a suggestion.)

Part Four: Traffic Light Simulator (HMI)

In this section, you will build a simple HMI application so that the traffic lights can be monitored and controlled remotely using the Modbus/TCP protocol. Notice that the ladder logic in Rungs #9 through #14 output to the Modbus registers MC1 through MC6; in Rungs #15 through #20, these registers are connected through to the outputs Y0 through Y6, which drive the lights. Because the Modbus registers are being used as control registers, we can monitor their state through an HMI application.

Begin by extracting and opening a fresh copy of the AdvancedHMI sample application, and by adding and configuring a **ModbusTCPCom** component, just as you did earlier. Once you have the application prepared, open the list of **AdvancedHMIControls** components and add three of them to the application: two **StackLight** controls to represent the traffic lights, and a **MomentaryButton** control to act as the "emergency mode" switch (we will revisit this toward the end of this exercise, time permitting). Arrange them however you like in the main form, and assign them descriptive names: select each control, and in its "Properties" window, find the "Text" attribute. Change the text for the **StackLight** controls to "E/W" and "N/S", and the text for the **MomentaryButton** to "Emergency". You may need to resize the controls to see the names that you assigned to them.

To make these controls work, they need to be associated with the addresses corresponding to the Modbus PLC registers in our ladder logic. Select one of the **StackLight** controls and scroll down in the "Properties" window until you find the "PLC Properties" group. Several values in this group will need to be assigned correctly for both controls: because each **StackLight** contains three lights, the correct address must be specified for each light. These addresses are given on the next page.

For the N/S lights:

PLCAddressLightAmberValue: **0002**
PLCAddressLightGreenValue: **0003**
PLCAddressLightRedValue: **0001**

For the E/W lights:

PLCAddressLightAmberValue: **0005**
PLCAddressLightGreenValue: **0006**
PLCAddressLightRedValue: **0004**

(These numeric addresses correspond to the Modbus Coil registers MC1 through MC6.)

Finally, open the properties of the **MomentaryButton** control and scroll down to its "PLC Properties" group. Assign the value **00007** to the "PLCAddressClick" attribute, corresponding to register MC7.

It is now time to test your application. Download and run the ladder logic to the PLC or to the simulator, as described earlier. Make sure that the PLC or the simulator is in "RUN" mode, *then* launch the HMI application from Visual Studio by clicking the “Start” button in the toolbar. Shortly after the application opens, you should notice that the state of the lights will change in the HMI application to correspond to the lights on the PLC. If the lights do not change, the application may not be able to communicate with the PLC; in that case, double-check the properties of the **ModbusTCPCom** component in the application to ensure that the IP address matches your PLC address (for the real PLC) or workstation address (for the simulator). See the Part Two instructions for details.

Here is an example of what a completed HMI application might look like. In this example, the light in the E/W direction is GREEN, and the light in the N/S direction is RED:

Part Five: Traffic Light Simulator (Emergency Mode)

Once you have the HMI application and ladder logic working correctly *(and do not move on to this step until you do!)*, the final step is to add an "emergency mode." In this mode, the lights operate in "blink mode," with a blinking *yellow* light on the E/W main road and a blinking *red* light on the N/S side road, when the emergency switch is thrown. Both lights should blink on for one-half second and off for one-half second in a continuous cycle; the light should also alternate, so that if the yellow light is *on*, the red light is *off*, and vice-versa.

Look at the logic in Rung #2, and notice that *either* the input switch X0 *or* the Modbus Coil register MC7 will set the single-bit register C0 to ON. This means that C0 is the *emergency mode indicator*; if C0 is ON, the lights should operate in emergency mode; if it is OFF, the lights should operate normally. The mode can be changed by switching the input switch X0 either on or off, *or* by clicking the button in the HMI interface labeled "Emergency" (this is the button you created in Part Four).

You should begin by adding a *normally closed contact* to C0 to each of the rungs which control the lights (Rungs #9 through #14). Be sure to add the new contact *after* the other logic on these rungs; this way, when C0 is ON, the contact will *break the circuit* and the lights will remain *off*.

Next, add a new row to Rungs #9 and #13 (the "RedFlag_NS" and "YellowLight_EW" rungs, respectively) which uses the built-in timer ST4 to make the lights blink; the logic in this row should turn the lights *on* if C0 **AND** ST4 are ON. *Be sure that the wire from the new row connects to the rung **after** the normally-closed contact you added earlier!* To make the lights alternate, change one of the normally-open contacts to ST4 to a normally-closed contact (it does not matter which one).