

**2019 MASTERs Conference**

**23075 IoT6**

**Simplifying TCP/IP Applications with MPLAB® Harmony**

**Hands-On**

**Lab Manual**

***Instructors:***

***Martin Ruppert***

***Raji Shanmugasundaram***

***Niklas Larsson***

*Microchip Technology Inc.*

**MPLAB® Harmony TCP/IP Stack**

# UDP Module API Function List

## Socket Management Functions

|  |  |
| --- | --- |
| **TCPIP\_UDP\_ServerOpen** | Opens a UDP socket as a server. |
| **TCPIP\_UDP\_ClientOpen** | Opens a UDP socket as a client. |
| **TCPIP\_UDP\_IsOpened** | Determines if a socket was opened. |
| **TCPIP\_UDP\_IsConnected** | Determines if a socket has an established connection. |
| **TCPIP\_UDP\_Bind** | Bind a socket to a local address and port. This function is meant for client sockets. It assigns a specific source address and port for a socket. |
| **TCPIP\_UDP\_RemoteBind** | Bind a socket to a remote address This function is meant for server sockets. |
| **TCPIP\_UDP\_Close** | Closes a UDP socket and frees the handle. |
| **TCPIP\_UDP\_OptionsGet** | Allows getting the options for a socket such as current RX/TX buffer size, etc. |
| **TCPIP\_UDP\_OptionsSet** | Allows setting options to a socket like adjust RX/TX buffer size, etc |
| **TCPIP\_UDP\_SocketInfoGet** | Returns information about a selected UDP socket. |
| **TCPIP\_UDP\_SocketNetGet** | Gets the network interface of an UDP socket |
| **TCPIP\_UDP\_SocketNetSet** | Sets the network interface for an UDP socket |
| **TCPIP\_UDP\_TxOffsetSet** | Moves the pointer within the TX buffer. |
| **TCPIP\_UDP\_SourceIPAddressSet** | Sets the source IP address of a socket |
| **TCPIP\_UDP\_BcastIPV4AddressSet** | Sets the broadcast IP address of a socket Allows an UDP socket to send broadcasts. |
| **TCPIP\_UDP\_DestinationIPAddressSet** | Sets the destination IP address of a socket |
| **TCPIP\_UDP\_DestinationPortSet** | Sets the destination port of a socket |
| **TCPIP\_UDP\_Disconnect** | Disconnects a UDP socket and re-initializes it. |
| **TCPIP\_UDP\_SignalHandlerDeregister** | Deregisters a previously registered UDP socket signal handler. |
| **TCPIP\_UDP\_SignalHandlerRegister** | Registers a UDP socket signal handler. |
| **TCPIP\_UDP\_Task Standard** | TCP/IP stack module task function. |

## Transmit Data Functions

|  |  |
| --- | --- |
| **TCPIP\_UDP\_PutIsReady** | Determines how many bytes can be written to the UDP socket. |
| **TCPIP\_UDP\_TxPutIsReady** | Determines how many bytes can be written to the UDP socket. |
| **TCPIP\_UDP\_ArrayPut** | Writes an array of bytes to the UDP socket. |
| **TCPIP\_UDP\_StringPut** | Writes a null-terminated string to the UDP socket. |
| **TCPIP\_UDP\_Put** | Writes a byte to the UDP socket. |
| **TCPIP\_UDP\_TxCountGet** | Returns the amount of bytes written into the UDP socket. |
| **TCPIP\_UDP\_Flush** | Transmits all pending data in a UDP socket. |

## Receive Data Transfer Functions

|  |  |
| --- | --- |
| **TCPIP\_UDP\_GetIsReady** | Determines how many bytes can be read from the UDP socket. |
| **TCPIP\_UDP\_ArrayGet** | Reads an array of bytes from the UDP socket. |
| **TCPIP\_UDP\_Get** | Reads a byte from the UDP socket. |
| **TCPIP\_UDP\_RxOffsetSet** | Moves the read pointer within the socket RX buffer. |
| **TCPIP\_UDP\_Discard** | Discards any remaining RX data from a UDP socket. |

Table of Contents

UDP Module API Function List 2

Socket Management Functions 2

Transmit Data Functions 2

Receive Data Transfer Functions 2

Introduction 5

Hardware Requirements 5

Software Requirements 5

Lab 1 6

Introduction 6

Lab Procedure 7

Starting MPLAB X IDE 7

Project Setup 8

MHC: BSP Selection 9

MHC: Ethernet I/O Pin Configuration 11

MHC: TCP/IP Stack Configuration 13

MHC: Network Interface Configuration 14

MHC: ICMP Configuration 15

MHC: Network Interface Driver Selection 15

MHC: Console Configuration 17

MHC: Application Configuration 19

MHC: Project Generation 19

LED Flasher Implementation 21

LED Flasher Code 22

Header File 22

Source File Setup 23

Project Build 26

Programming 26

Application Validation 29

Network Interfacing 29

Cable Connections 30

Network Connectivity with TCP/IP Discovery Tool 31

Checking Network Connectivity with Windows Ping Client 32

Harmony TCPIP Command Console 33

Lab 2 36

Introduction 36

Data Protocol 37

Application Implementation 38

Objectives 39

Lab Procedure 39

Project Setup 39

MHC Application Configuration 41

MHC: DHCP Server Configuration 43

MHC: Setting the Host Name 44

MHC: Project Generation 45

Application Source and Header File Setup 47

Network Communications Controller Modification 50

Project Build 54

Programming 54

Application Testing 55

Cable Connections 56

UDP Server Testing 59

TCP Client Testing 63

ECS Testing 73

Harmony TCP/IP API Subset For Lab 2 78

TCP Socket Management Functions 78

TCPIP\_TCP\_ArrayGet Function 78

TCPIP\_TCP\_ClientOpen Function 78

TCPIP\_TCP\_Close Function 79

TCPIP\_TCP\_GetIsReady Function 79

TCPIP\_TCP\_IsConnected Function 80

TCPIP\_TCP\_PutIsReady Function 80

TCPIP\_TCP\_StringPut Function 81

TCPIP\_TCP\_WasReset Function 81

UDP Socket Management Functions 82

TCPIP\_UDP\_ArrayGet Function 82

TCPIP\_UDP\_Close Function 83

TCPIP\_UDP\_GetIsReady Function 83

TCPIP\_UDP\_ServerOpen Function 84

TCPIP\_UDP\_SocketInfoGet Function 84

UDP\_SOCKET\_INFO Structure 85

Network Communications Controller Application Code Modification Solutions 86

TCP Module API Function List 87

Socket Management Functions 87

Transmit Data Functions 87

Receive Data Transfer Functions 87

# Introduction

This Lab Manual provides the step by step procedure to complete two labs in the MASTERs 21070 NET1 Class. In Lab 1 you will configure the MPLAB Harmony TCPIP/IP Stack and test basic network connectivity and in Lab 2 you will learn how to implement and test a network application which will utilise a TCP Client and UDP Server in a real world application. Both Labs have specific hardware and software requirements.

# Hardware Requirements

The following hardware is required:

* **SAM E70 Xpained Ultra** (Microchip Part Number: DM320113)
  + [http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=dm320113](http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=dm320007)
* OLED1 Xplained Pro extension kit (Microchip Part Number: ATOLED1-XPRO)
* **Cat 5 Ethernet Patch Cable**
* **USB Male A to USB Male B Micro Cable**

The following hardware is optional:

* **Multimedia Expansion Board II** (Microchip Part Number: DM320005-2)
  + <http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=dm320005-2>
* **MPLAB ICD3 In-Circuit Debugger** (Microchip Part Number: DV164035)
  + <http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=dv164035>
* **USB Male A to USB Male B Cable** (supplied with DV164035)
* **6 Core Modular Cable with RJ11 Connectors** (supplied with DV164035)
* **9V, 500mA Power Supply with 2.5mm Plug**

# Software Requirements

The following software is required:

* **Microchip MPLAB X IDE v5.20.04**
  + <http://www.microchip.com/mplab>
* **Microchip MPLAB XC32 Compiler v2.20**
  + <http://www.microchip.com/mplab/compilers>
* **Microchip MPLAB Harmony v2.03B**
  + <http://www.microchip.com/mplab/mplab-harmony>
* **Microchip MPLAB Harmony Configuration (MHC) Tool Plugin v2.0.3.5**
  + MPLAB X Plugin “*com-microchip-mplab-modules-mhc.nbm”* is bundled with MPLAB Harmony under the microchip\v2\_03b\utilities\mhc folder
* **Tera Term v4.95**
  + <https://ttssh2.osdn.jp/index.html.en>
* **Packet Sender v5.1 (lab 2 only)**
  + [https://packetsender.com](https://packetsender.com/)
* **JSMN JSON Parser (lab 2 only)**
  + <http://www.zserge.com/jsmn.html>



# Introduction

Lab 1 will show you how to create a new TCP/IP MPLAB Harmony Project from scratch using the MPLAB Harmony Configuration (MHC) Tool. The project will incorporate basic TCP/IP functionality to allow the SAM E70 Xplained Ultra to connect to an Ethernet Network, along with a simple application to flash a “Heartbeat” LED every 500ms. Once the project is generated and programmed onto the development kit, you will use a number of techniques to validate that the PIC is connected to a network and determine its IP Address. The concepts that will be covered in this lab include:

* Creating a new SAM E70 MPLAB X Project
* Configuring the MPLAB Harmony path, project name and target device
* Selecting a Board Support Package (BSP)
* Enabling the TCP/IP Stack
* Configuring the TCP/IP Stack options, including:
  + Network Configuration (Interface Type, Host Name)
  + TCP/IP Services including Dynamic Host Configuration Protocol Client, ICMPv4 Server (for Ping testing) & Announce Discovery Tool
* Configuring Network Interface Drivers:
  + Enabling the Internal Ethernet MAC Driver
  + Selection of External PHY Type
  + Configuration of MAC and PHY Options
* Configuring the Harmony Console and Command Service for monitoring and control of the TCP/IP stack via a Terminal Client running on a USB CDC Interface (Emulated RS232 COM Port).
* Configuring the USB Device Driver for a CDC Interface
* Application Configuration: Changing the default application name
* Implementing a simple LED flasher using the System Timer Service as the timing base
  + Adding a System Timer Service Handle variable in the application data structure
  + Checking if the System Timer Service is ready for use
  + Setting up a non-blocking 500ms Delay
  + Polling the System Timer Service to determine if the delay period has elapsed
  + Rearming the 500ms Delay
* Toggling the IO Pin that drives LED3 on the SAM E70 Xplained UltraSAM E70 Xplained Ultra
* Using the Windows Command Line Ping Tool and the Microchip TCP/IP Discovery tools to test connectivity of your SAM E70 Xplained UltraSAM E70 Xplained Ultra on the network
* Use the Console and Command System to get help on available TCPIP Commands and execute a command to get information about the network configuration.

# Lab Procedure

## Starting MPLAB X IDE

* 1. Start MPLAB X IDE by double clicking on the MPLAB X IDE vv5.20.04 icon found on the Windows desktop.

|  |
| --- |
|  |
|  |

## Project Setup

* 1. Open the New Project Wizard by choosing File⮞New Project… from the main menu.

|  |
| --- |
|  |

There are two steps to create a new MPLAB Harmony project. In the Choose Project step, set the following options:

* 1. Under Categories select Microchip Embedded.
  2. Under Projects select 32-bit MPLAB Harmony 3 Project.
  3. Click on Next> to continue.

|  |
| --- |
|  |

For the Name and Location step, set the following options:

* 1. Verify that the Harmony Path is C: \Microchip\harmony\v3where c is the logical hard drive where MPLAB Harmony has been installed.
  2. Set the Project Location to C:\MASTERs\23075
  3. Enter net1lab1 for the Project Name.
  4. Set the Device Family to PIC32MZ
  5. Set the Target Device to:
* ATSMAE70Q21B
  1. Leave the Target Board blank.
  2. Click on Finish to continue.

|  |
| --- |
|  |

At this point, the Microchip Harmony Configuration (MHC) Tool will automatically open. The project configuration options will be performed in the MPLAB Harmony & Application Configuration tree which is located within the MPLAB® Harmony Configurator tab.

|  |
| --- |
|  |

## MHC: BSP Selection

The first step in the MHC setup process is to enable a Board Support Package for your targeted hardware platform. The Board Support Package (BSP) will assign the configuration bits, clock settings, and I/O pins for external peripherals that are defined in the BSP. For example, on the SAM E70 Xplained UltraSAM E70 Xplained Ultra, the BSP will setup the I/O pins for the Push Buttons (Digital Inputs with Pull-ups) and LEDS (Digital Outputs with the initial state of Logic Low).

* 1. Expand the BSP Configuration tree by left clicking on the plus-sign  button.
  2. Check the Use BSP? Option. When the option is checked you will see a tick  inside the check box.
  3. Expand the Select BSP To Use for PIC32MZ2048EFH144 Device tree.
  4. Check the SAM E70 Xplained UltraSAM E70 Xplained Ultra option.

|  |
| --- |
|  |

The Harmony BSP you have just selected already contains basic configuration bit and clock settings for the SAM E70 Xplained UltraSAM E70 Xplained Ultra. Therefore, it is not necessary to change any options under the Device & Project Configuration tree for this Lab.

## MHC: Ethernet I/O Pin Configuration

The PIC32MZEF Starter Kit has a PHY Daughter Board with a Microchip LAN8740 Ethernet Physical Layer Transceiver. The J6 connectors on the Starter Kit is used for the interfacing the PHY Daughter Board. Lines on the J6 connector are hardwired to specific I/O pins on the PIC32MZEF2048EFH144 that have Ethernet functions.

The PIC32 has two sets of Ethernet I/O pins that can be used for interfacing to a PHY, they are called Default and Alternative. The PIC32 supports two types of MAC to PHY interfaces: Media Independent Interface (MII), which requires 18 I/O lines, and the Reduced Media Independent Interface (RMII), which only requires 10 I/O lines. The SAM E70 Xplained UltraSAM E70 Xplained Ultra uses the Default Ethernet I/O pins with RMII.

A number of the Ethernet I/O lines on the PIC32MZEF part are shared with an Analogue Input, and therefore it is necessary to configure these pins to Digital mode using the MHC Pin Manager. The table shown below lists all Ethernet I/O pins which are used on the Starter Kit.

|  |  |  |  |
| --- | --- | --- | --- |
| PIC32 Pin Number | PIC32 I/O Pin Name | Description | Shared Analogue |
| 7 | RJ8/ETXD0 | Data Transmit 0 | Y |
| 8 | RJ9/ETXD1 | Data Transmit 1 | Y |
| 27 | RJ11/EREF\_CLK | RMII Reference Clock | Y |
| 65 | RH4/ERXERR | Receive Error | Y |
| 66 | RH5/ERXD1 | Data Receive 1 | Y |
| 81 | RH8/ERXD0 | Data Receive 0 | N |
| 84 | RH11 | PHY Hardware Reset | N |
| 99 | RD11/EMDC | MII Management Clock | N |
| 101 | RH13/ECRS | RMII Carrier Sense | N |
| 105 | RC13 | PHY Interrupt | N |
| 115 | RJ1/EMDIO | MII Management IO | N |
| 120 | RD6/ETXEN | Transmit Enable | N |

* 1. Click on the Pin Settings tab in the MPLAB Harmony Configurator.

|  |
| --- |
|  |

* 1. Locate Pin Number 7 (Pin ID RJ8) in the Pin Settings Table.

|  |
| --- |
|  |

* 1. Click on the Analog Mode button to switch the Mode to Digital.

|  |  |  |
| --- | --- | --- |
|  |  |  |

* 1. Repeat step 1.19 for all Ethernet I/O pins that have shared Analog, that is Pin Numbers 8, 27, 65 and 66.

|  |
| --- |
| **Double check that Pins 7, 8, 27, 65 and 66 have been set to Digital Mode in the Pin Settings Table.** |

## MHC: TCP/IP Stack Configuration

The following steps will enable and configure the TCP/IP stack. When the TCP/IP Stack is enabled via MHC, the following set of options will automatically be enabled:

* IPv4
* TCP and UDP
* DHCP Client
* DNS Client
* NetBIOS Name Server
* Announce Discovery Tool
  1. Expand the following tree path: Harmony Framework Configuration⮞TCPIP Stack.
  2. Check the Use TCP\IP Stack? option.

|  |
| --- |
|  |

## MHC: Network Interface Configuration

In this section, you will configure the TCP/IP stack to use one network interface. The Harmony TCP/IP Stack supports different network interface types, including Internal Ethernet MAC + External PHY, External Ethernet Controller and Wi-Fi. The SAM E70 Xplained UltraSAM E70 Xplained Ultra uses a PIC32 device with Internal Ethernet MAC, and you see how to select and configure this network interface.

* 1. Expand the Network Configuration 0 tree.
  2. To use the Internal Ethernet MAC interface on the PIC32MZ EF device, change the Interface option to PIC32INT.
  3. Set the Host Name to the combination of your First Name and the year you were born. This naming format is used to ensure every attendee in the class has a unique host name to prevent any conflicts on the classroom network.
  4. The PIC32MZ EF device contains a factory programmed MAC address. The TCP/IP Stack will automatically use this MAC address when the Mac Address field is left blank.

|  |
| --- |
|  |

|  |
| --- |
| For this lab, all other options listed under Network Configuration 0 do not require any change. Since the PIC32MZ EF Stater kit will be connecting to an Ethernet network with a DHCP Server, the IP Address, Network Mask, Default Gateway, and Primary/Secondary DNS settings are automatically obtained from the DHCP Server. The Network Configuration Start-up Flags contain the correct default settings to enable the DHCP Client and DNS Client on the network interface when then PIC powers up. |

## MHC: ICMP Configuration

For this lab, you will be testing network connectivity using the Windows Ping tool. The ICMPv4 Server must be enabled in order for the TCP/IP Stack to reply to ICMP Ping requests.

* 1. Check the ICMPv4 Client and Server option.
  2. Check the Use ICMPv4 Server option.

|  |
| --- |
|  |

## MHC: Network Interface Driver Selection

In step 1.24, the **Internal MAC** was selected for the network interface type. In order to use the Internal MAC, a driver must be enabled and configured and the External Physical Layer Controller (PHY) device type must be selected. The SAM E70 Xplained Ultra has the Microchip/SMSC LAN8740 PHY.

* 1. Expand the following tree path: Harmony Framework Configuration⮞Drivers⮞Internal Ethernet MAC Driver.
  2. Check the Use Internal Ethernet MAC Driver? option.

|  |
| --- |
|  |

* 1. For External PHY Type option, select SMSC\_LAN8740.
  2. All other configuration options under the Internal MAC Driver tree can remain with the default values.

|  |
| --- |
|  |

## MHC: Console Configuration

The Harmony console provides the ability to monitor and control different modules within the TCP/IP Stack such as the DHCP Client. It is a useful tool for development and debugging purposes. The console interfaces to a host PC via USB (as a CDC device), or via UART. For this lab, the USB interface is used for the console.

* 1. Expand the tree path: Harmony Framework Configuration⮞System Services⮞Command.
  2. Check the Use Command Processor System Service? option. By checking this option, the Console System Service will be automatically enabled, however further configuration is required.
  3. Expand the Console⮞Use Console System Service? tree.
  4. Set the Select Peripheral For Console Instance option to USB\_CDC\_CONSOLE.

|  |
| --- |
|  |

* 1. With the selection of the USB CDC CONSOLE, the MHC has automatically enabled the USB Library. However, one USB Library setting needs changing to allow the USB device to successfully enumerate. Expand the Harmony Framework Configuration⮞USB Library⮞USB Device Instance 0 tree.
  2. For the Product ID Selection, select the cdc\_com\_port\_single\_demo option, which will automatically populate the Vendor and Product IDs for the USB Device. The USB device cannot enumerate without these IDs being set.

|  |
| --- |
|  |

* 1. Under the Harmony Framework Configuration⮞TCPIP Stack tree, check the Use TCP\IP Commands option.

|  |
| --- |
|  |

## MHC: Application Configuration

The following steps will rename the default Harmony Application from app to ledcontrol. This application configuration will be used to generate the periodic LED flasher.

* 1. Expand the following tree path: Application Configuration⮞Application 0 Configuration.
  2. Change the Application Name option from app to ledcontrol.

|  |
| --- |
|  |

## MHC: Project Generation

The MHC Configuration for Lab 1 is now complete. The following steps will generate the new Harmony Project.

* 1. Click on the Generate Code icon

|  |
| --- |
|  |

* 1. In the Modified Configuration window click on Save.

|  |
| --- |
|  |

* 1. In the Generate Project window, you will be prompted about the code merging strategy. Select the Prompt Merge For All User Changes option.
  2. Click on Generate.

|  |
| --- |
|  |

MHC will now populate the project with the required source and header files.

* 1. After the MHC has finished generating the project, go to the Projects window, expand the net1lab1 project folder, and then expand the subfolders under Header Files and Source Files. You should see the same folder structure as shown in the figure below.

|  |
| --- |
|  |

* 1. Close MHC by clicking on the “**x**” icon in the MPLAB Harmony Configurator window.

|  |
| --- |
|  |

## LED Flasher Implementation

The LED Flasher will be implemented in the *ledcontrol* application configuration that was setup in MHC. The Harmony *System Timer Service* is utilised to generate a non-blocking 500ms delay. When the delay period has elapsed, the I/O pin driving the LED will be toggled. The System Timer Service is automatically added to the project when the TCP/IP Stack is enabled. In order to use the service there are a few steps that need to be followed:

1. The System Timer Service must be in a ready state. This is checked by calling the Harmony function SYS\_TMR\_Status(sysObj.systmr) and checking if the return value is SYS\_STATUS\_READY.
2. Once the service is ready, the SYS\_TMR\_DelayMS(uint32\_t delayMs) function is called to setup the desired delay time. The delayMs parameter is the delay period (in milliseconds). The API will return with a value of SYS\_TMR\_HANDLE type and the returned value (known as the handle) must be stored. The handle will be used for all future management of the timer.
3. To check if the delay has elapsed, periodically call SYS\_TMR\_DelayStatusGet(SYS\_TMR\_HANDLE handle). When the delay has elapsed, the API will return with a Boolean value of true.

1. When the delay has elapsed, the Harmony Timer Service automatically destroys the timer object, and the handle stored in step 2 is no longer valid. Therefore, to start a new delay, steps 2 and 3 must be executed again.

To toggle a LED, you can use the Harmony Board Support Package function called BSP\_LED\_Toggle(BSP\_LED led). The led parameter that is passed specifies which LED to toggle. There are three possible values for the SAM E70 Xplained Ultra: BSP\_LED\_1, BSP\_LED\_2 and BSP\_LED\_3.

In Lab 1, the System Command and Console Module is used to show how the TCP/IP stack can be monitored and controlled via a Terminal Interface. The SYS\_CMD\_READY\_TO\_READ function must be periodically called in order to for the System Command and Console Module to function correctly. Since the LED Flasher function toggles LED3 every 500ms, this function will also be used to periodically call the SYS\_CMD\_READY\_TO\_READ function.

## LED Flasher Code

The full source code for the LED flasher has already been implemented. In the next two sections you will be adding the header and source code to the lab1 project.

### Header File

* 1. Open ledcontrol.h header file: In the Projects window, double click on ledcontrol.h under the lab1⮞Header Files⮞app tree.

|  |
| --- |
|  |

* 1. In ledcontrol.h Source Editor Window, locate the LEDCONTROL\_DATA structure type definition by double clicking on the LEDCONTROL\_DATA row in the Navigator window.

|  |
| --- |
|  |

* 1. In the LEDCONTROL\_DATA structure, you are going to add a variable to store the handle for an instance of the system timer service. On line 117, add the following variable declaration: SYS\_TMR\_HANDLE ledFlashTmrHande;

|  |
| --- |
|  |

* 1. Save the ledcontrol.h file.

|  |
| --- |
| **Procedure To Save the Source/Header File in the active Source Editor Window:**   * **Main Menu:** Choose File⮞Save * **Keyboard Shortcut**: Ctrl+s |

* 1. Close the ledcontrol.h file.

|  |  |
| --- | --- |
| **Procedure To Close the Source/Header File in the active Source Editor Window:** | |
| * Source editor window Tab: Click on the “**x**” button. * Keyboard Shortcut: Ctrl + w |  |

### Source File Setup

In this section, you are going to copy the source code for the LED Flasher into the Lab 1 source folder.

* 1. Launch the Windows File Manager and open the Lab 1 Source Files folder that is located under the following path:

C:\MASTERs\21070\Lab Manual\Lab 1 Source Files

|  |
| --- |
|  |

* 1. Click on the ledcontrol.c file to highlight the file.
  2. Use the Windows copy shortcut, Ctrl+c, to copy the selected file.

|  |
| --- |
|  |

* 1. In the Windows File Manager open the project src folder that is located under the following path:

C:\MASTERs\21070\net1lab1\firmware\src

|  |
| --- |
|  |

* 1. Use the Windows paste shortcut, Ctrl+v, to paste the new ledcontrol.c file. You will be prompted to replace the file: Select the Replace the files in the destination option.

|  |
| --- |
|  |
|  |

## Project Build

* 1. To build the project click on the Clean and Build Main Project icon in the MPLAB X Run toolbar.

|  |
| --- |
|  |

* 1. Confirm the Build was successful by checking the Output Window.

|  |
| --- |
|  |

## Programming

The application is now ready for programming onto the SAM E70 Xplained Ultra. To use the on-board programmer, follow this procedure:

* 1. To connect the on-board debugger to the PIC32, a Jumper must be installed on JP2.
  2. Check the LAN8740 PHY Daughter Board is seated correctly in J6.
  3. Attach a USB Male-A to Male **Mini-B** cable to the **USB DEBUG** port of the kit, and then attach to the PC. The USB Debug port is located in between the LAN8740 PHY and the USB-A Connector on the kit.
  4. Ensure the LEDS D6 and D7 are illuminated.

|  |
| --- |
|  |

* 1. To select the on-board Programmer/Debugger on the SAM E70 Xplained Ultra: choose File⮞Project Properties (lab1) in the MPLAB X main menu.

|  |
| --- |
|  |

* 1. In the Project Properties – lab 1 window, under Hardware Tool, locate Microchip Starter Kits->Starter Kits (PKOB) and click on the SAM E70 EF Family option.
  2. Click Apply, and then click OK.

|  |
| --- |
|  |

* 1. To program the application click on the Make and Run icon in the MPLAB X Run Toolbar.

|  |
| --- |
|  |

* 1. The Starter Kit on Board tab in the Output window will indicate if programming was successful.

|  |
| --- |
|  |

* 1. After programming, confirm that LED3 on the SAM E70 Xplained Ultra is flashing.

## Application Validation

### Network Interfacing

For validating the operation of the TCP/IP Stack, the SAM E70 Xplained Ultra must be connected to a network which has a DHCP Server. The architecture of the network will be similar to that depicted in the diagram shown below.

|  |
| --- |
|  |

### Cable Connections

The cable connections required for Lab 1 are depicted in the diagram shown below. **The lab manual will state when each cable connection needs to be performed.**

|  |  |
| --- | --- |
| **3**  **1**  **2** | |
| **1** | **Network Connection**  Cable: CAT5 Ethernet Cable from Classroom Network  Connection: RJ45 Jack on PCB Top |
| **2** | **Programming Connection**  Cable: USB Male A to Male B Mini Cable supplied with Starter Kit  Connection: USB Debug Port on PCB Top to Laptop USB Port |
| **3** | **Console Connection**  Cable: USB Male A to Male B Micro Cable supplied with Starter Kit  Connection: USB Micro Connector on PCB Bottom to Laptop USB Port |

* 1. Plug the CAT5 Ethernet cable that is connected to the classroom network, into the RJ45 Jack on the SAM E70 Xplained Ultra.
  2. Confirm that the Green “Link” LED on the RJ45 socket lights up.

### Network Connectivity with TCP/IP Discovery Tool

* 1. Run the Microchip TCP/IP Discovery Tool to determine the IP Address of your board: From Windows Explorer go to the following folder: c:\microchip\harmony\v2\_03b\utilities\tcpip\_discoverer   
     where c is the hard drive where MPLAB Harmony is installed.
  2. Double click on the tcpip\_discoverer executable JAR file.
  3. After the Microchip TCPIP Discoverer tools opens, click on the Network Direct Broadcast to place a tick in the check box.
  4. Press the Discover Devices button: The tool will send a UDP broadcast on port 30303, with the packet “Discovery, who is out there?” All PIC32 devices running the Announce service will respond to this broadcast, by sending a return broadcast on port 30303. The broadcast packet contains data on the type of interface used, the Host Name, MAC and IP Address. The Discover tool listens to all broadcasts on port 30303 and will show found devices under the Microchip Devices tree. You can identify your device by looking for the host name that you entered in step 1.25 of the MHC Setup process. The Microchip TCPIP Discoverer tool also shows the IP address for your board.

|  |
| --- |
|  |

* 1. Detach the network cable from the SAM E70 Xplained Ultra and then click on Discover Devices. Your device will disappear from the list.
  2. Reattach the network cable to the SAM E70 Xplained Ultra, wait a few seconds, and then click on  
      Discover Devices button again. Your device should reappear in the Microchip Devices list.
  3. Close the Microchip TCP/IP Discover Tool by pressing the Exit button.

### Checking Network Connectivity with Windows Ping Client

The Windows ping client will be used to check that the ICMPv4 Server is operational on your SAM E70 Xplained Ultra.

* 1. Start the Windows Command Prompt by clicking on the Start button, click on the Run option, type cmd and press Enter.
  2. In command prompt, type ping followed by a space and your host name, and press Enter. You should see an output similar to the figure shown below.

|  |
| --- |
|  |

The ping tool works differently compared to the Microchip TCP/IP Discoverer Tool. Ping uses Internet Control Message Protocol (ICMP), which is a protocol defined in RFC 1122 and is used for diagnostics and control purposes. The ping tool will first resolve the IP address for the host name (Mark1979). A unicast ICMP message will then be sent to the resolved IP address. The ICMP Server on the PIC will reply back to the source IP address of the original ping request.

### Harmony TCPIP Command Console

* 1. Attach a USB Male-A to Male **Micro-B** Cable between J4 (underneath the USB A Connector) on the SAM E70 Xplained Ultra, and a spare USB Port on the PC.
  2. Start the terminal client application by clicking on the Tera Term icon found on the Windows desktop.

|  |
| --- |
|  |

* 1. In the Tera Term: New connection window, select the Serial Radio button.
  2. For the Port option you should see COMx: USB Serial Port (COMx). The assigned COM Port Number (x) may be different on your PC compared to that shown in the screenshot.
  3. Press the OK button to continue.

|  |
| --- |
|  |

* 1. In the Tera Term VT window, press the Enter key. You will see a prompt symbol “>” appear on the screen.

|  |
| --- |
|  |

* 1. Type help into the prompt, and press Enter. A simple help screen will appear.

|  |
| --- |
|  |

The basic command service allows you to reset the Microcontroller and disable the command processor. Since the TCP/IP Stack Commands were enabled in step 1.22, there are additional commands that can be used to monitor and control the TCP/IP Stack.

* 1. A full list of all TCP/IP Commands can be obtained by typing help tcpip and press Enter

|  |
| --- |
|  |

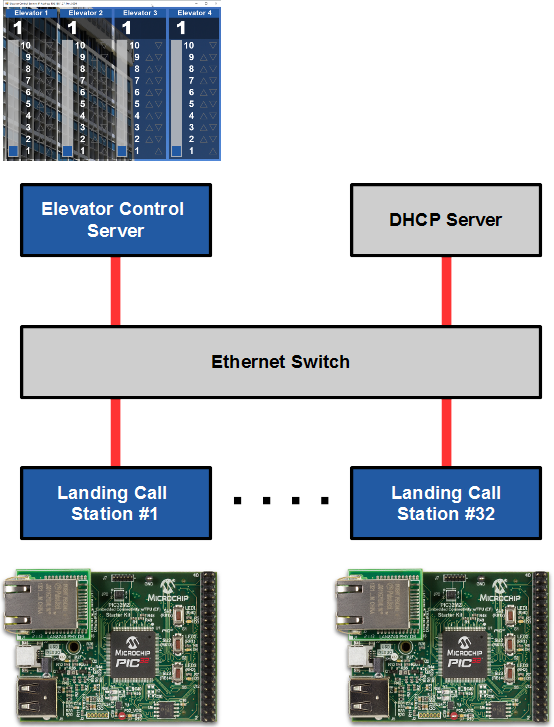
* 1. To check the basic information about the network enter the netinfo command and press Enter.

|  |
| --- |
|  |

Congratulations, you have completed Lab 1!



# Introduction

You will be implementing an embedded network application for an elevator signalisation system. The task is to interface the up/down call buttons and indicator lamps on each floor (i.e. Landing Call Station or LCS) to the elevator control server (ECS) using an Ethernet interface. Each attendee in the class has the task to implement the LCS for a designated floor landing in the elevator system.

The ECS is responsible for operating the elevator car based on call requests made from the LCS units. The network architecture is very simple, where the ECS and each LCS unit is interfaced to a common network switch. A network router is also connected to the switch that provides a DHCP server to automatically assign an IP Address to each device connected to the network.

The LCS unit will need to determine the ECS IP Address prior to establishing a connection. The ECS sends a periodic UDP broadcast message on port 2027. The LCS unit will determine the ECS IP Address by reading the source IP address field in the IPv4 header of the broadcast message.

Each LCS unit communicates with the ECS though a dedicated TCP Socket on port 3096. The LCS unit will establish a persistent TCP connection with the ECS. When one of the call buttons is pressed, the LCS will send a *call request* data packet to the ECS. The ECS will lodge the call request and will send an acknowledgement back to the LCS.

The indicator lamps are controlled from the ECS. The LCS will periodically poll the ECS for the latest indicator lamp state by sending a *status request* data packet. The ECS will respond with a data packet that contains status information about the elevator (current floor, direction, and state of indicator lamps). The LCS will process this data and will control the indicator lamps based on the received data values.

# Data Protocol

|  |
| --- |
|  |

# Application Implementation

The Landing Call Station will be implemented using the PIC32MZ EF Embedded Connectivity Starter Kit. The following features of the kit are used to implement the application:

* SW1 (Up), SW2 (Down) Call Buttons
* LED1 (Up), LED2 (Down) Call Indicator Lamps

|  |
| --- |
|  |
| **Landing Call Station Project Superloop Structure** |
|  |
| The Landing Call Station will run three user applications in conjunction the MPLAB Harmony TCP/IP Stack:   * **Lamp Controller** (LEDCONTROL\_Tasks): manages the operation of the Call Indicator Lamps * **Button Controller**(BUTTONCONTROL\_Tasks): manages the sampling and de-bouncing of the Call Buttons * **Network Communications Controller** (NETWORKCOMS\_Tasks): manages all high level network communications with the ECS |

# Objectives

In Lab 1, you configured a new MPLAB Harmony Project from scratch that included the TCP/IP Stack middleware and a LED Flasher. The project you created in Lab 1 will be used as the starting point for Lab 2. For Lab 2, you are going to use MHC to add two additional applications to the project and configure the DHCP Server. The source code for the Button Controller, LED Controller and Network Communications Controller will be copied into the project.

You will learn about a number of Harmony TCP/IP API functions by adding the necessary APIs into the Network Communications Controller source code to manage the UDP and TCP Sockets and data exchanged with each socket. Finally, you will get to use the Packet Sender software tool to perform isolated testing of the embedded application, prior to connecting your Floor Call Station implementation to the Elevator Control System on the network.

# Lab Procedure

## Project Setup

* 1. Close Lab 1 project by choosing File⮞Close Project(lab1) in the main menu.

|  |
| --- |
|  |

* 1. The project for Lab 2 has already been setup in advance. The project is a working implementation of Lab 1 that has been renamed to net1lab2, and has a number of files added to the project, including jsmn header and source (JSON Parser), ecsdataprocessing header and source (Processes the received data from the Elevator Controller), and ecsdatatypes header (Defines a number of common types used in the project). To open the Lab 2 project, choose File⮞Open Project in the main menu.

|  |
| --- |
|  |

* 1. In the Open Project window, enter C:\MASTERs\21070\net1lab2\firmware into the File name text box.
  2. Click on net1lab2.X icon in the file list.
  3. Press Open Project.

|  |
| --- |
|  |

The source and header structure for the project is shown below.

|  |
| --- |
|  |

## MHC Application Configuration

The following procedure will add two new applications to the project using MHC. The two new applications are for the Button Controller, and the Network Communications Controller.

* 1. Before the MHC Configuration tool can be used, you need to set lab2 as the main project, by right clicking on the net1lab2 folder under the Projects window, and choosing Set as Main Project in the popup menu.

|  |  |
| --- | --- |
|  |  |

* 1. To launch MHC, choose Tools⮞Embedded⮞MPLAB Harmony Configurator in the main menu.

|  |
| --- |
|  |

* 1. In the Open Configuration dialog box, make sure the path states   
     C:\MASTERs\21070\net1lab2\firmware\src\system\_config\default\default.mhc  
     and click on Open.

|  |
| --- |
|  |

* 1. Expand the Application Configuration tree.
  2. Set the Number of Applications option to 3.
  3. Expand both Application 1 and 2 Configuration trees, and set the Application Name option to buttoncontrol for Application 1, and networkcoms for Application 2.

|  |
| --- |
|  |

## MHC: DHCP Server Configuration

The DHCP Server is required for testing the Call Button Controller directly with your PC. You will be creating an isolated host to host network that consists of two devices, the PIC32MZ EF Stater Kit and the PC. The DHCP Server is required in this network setup to allow the PIC32 to assign an IP Address to the PC.

* 1. Expand the following tree: Harmony Framework Configuration⮞TCPIP Stack.
  2. Check the DHCP Server option.

|  |
| --- |
|  |

* 1. Set the Maximum Number of Entries in Lease Table to 1.
  2. Expand the DHCP Server Instance 0 option. You will see a number of options for the DHCP Server including the IP Address of the server which by default is 192.168.1.1.

|  |
| --- |
|  |

The Interface Index for the DHCP Server specifies which network interface the DHCP Sever Instance will be running on. In the case of this lab, there is only one network interface. When configuring the DHCP Server IP Address, it must be identical to the static IP Address of the Network Interface. By default the static IPv4 Address of the Network interface is 192.168.100.115, while the default IP Address of the DHCP Server is 192.168.1.1. To fix this address mismatch, the default static IP addresses for the Network Interface will be changed to match that of the DHCP Server.

* 1. Expand the Network Configuration 0 tree.
  2. Change the following options to the values given:  
     IP Address: 192.168.1.1  
     Default Gateway: 192.168.1.1  
     Primary DNS: 192.168.1.1

|  |
| --- |
|  |

|  |
| --- |
| **While the DHCP Server and DHCP Client modules are now added to the project, the Network Configuration Start-up flags will control the specific DHCP Service that is active on the network interface when the PIC is powered on. By default, the DHCP Client is enabled, and the DHCP Server is disabled. The DHCP Client and Server can be enabled and disabled at run time using specific Harmony TCP/IP APIs. In this lab, the Network Communications Controller will automatically switch between the DHCP Client and DHCP Server.** |

To prevent network conflicts, the DHCP Server should always be **disabled** if the PIC32 is connecting to a network that already has an active DHCP Server.

## MHC: Setting the Host Name

The Lab 2 project has been pre-configured with the *Microchip1989* Host Name.

You will need to change the Host Name as per Lab 1.

* 1. In the Network Configuration 0 tree under Network Configuration 0 set the Host Name to the combination of your First Name and the year you were born.

## MHC: Project Generation

* 1. Click on Generate Project icon.

|  |
| --- |
|  |

* 1. In the Modified Configuration window click on Save.

|  |
| --- |
|  |

* 1. In the Generate Project window set the Generated code merging strategy to Prompt Merge For All User Changes, and then click on Generate.

|  |
| --- |
|  |

* 1. After the MHC has finished generating the project, go to the Projects Window and expand the app folders under Header Files and Source Files to confirm you see the new source/header files for the *buttoncontrol* and *networkcoms* applications.

|  |
| --- |
|  |

* 1. Close MHC by clicking on the “**x**” icon in the MPLAB Harmony Configurator window.

|  |
| --- |
|  |

## Application Source and Header File Setup

In step 2.11, two Harmony applications were added to the project, *buttoncontrol* and *networkcoms*. The source and header files for the newly added applications will contain the standard Harmony application templates, and will need additional source code added in order to make the applications operate as per the application brief. The ledcontrol application created in Lab 1 will also need additional code in order to operate the Call Lamps. In this section, you are going to update the source and header files for all three applications.

* 1. Launch the Windows File Manager and open the Lab 2 Source Files folder that is located under the following path:

C:\MASTERs\21070\Lab Manual\Lab 2 Source Files

|  |
| --- |
|  |

* 1. Use the Windows *Select All* shortcut, Ctrl+a, to select all source and header files in the folder.
  2. Use the Windows *Copy* shortcut, Ctrl+c, to copy the selected files in the folder.

|  |
| --- |
|  |

* 1. In the Windows File Manager, open the project src folder that is located under the following folder path:

C:\MASTERs\21070\net1lab2\firmware\src

|  |
| --- |
|  |

* 1. Use the Windows Paste shortcut, Ctrl+v, to paste the source and header files. You will be prompted to replace six (6) files that have the same names. Select the Replace the files in the destination option.

|  |
| --- |
|  |

* 1. Windows will automatically highlight the newly pasted files. Confirm the source and header files for ledcontrol, buttoncontrol and networkcoms have been pasted in the src folder.

|  |
| --- |
|  |

## Network Communications Controller Modification

In this section, you will gain some experience with the use of Harmony TCP/IP API functions. The networkcoms source is missing eight (8) lines of code. All missing code specifically relates to management of the UDP Server or TCP Client, such as opening or closing the socket, checking if the socket is connected, checking if data is available, reading received data, and writing data to the socket. Your task is to read the description for each missing item, select the appropriate Harmony TCP/IP API, and fill in the missing line of code.

* 1. Open the Network Communications Controller Application source file by double clicking on networkcoms.c under the net1lab2⮞Source Files⮞app path in the Projects window.

|  |
| --- |
|  |

|  |
| --- |
|  |

* 1. To access the description for each missing line of code, open the MPLAB X *Action Items* list, by choosing Window⮞Action Items in the main menu.

|  |
| --- |
|  |

* 1. To show the action items for the networkcoms.c file, click on the *Show Actions Items for currently edited file only* icon which is located in the Action Items window.

|  |
| --- |
|  |

To navigate to a specific action item, double click on the item description in the Action Items window.

|  |
| --- |
|  |

MPLAB X generates the Action Item list by parsing the networkcoms.c file, and identifying any comments that start with a specific *ToDo Pattern.* ToDo Patterns are configurable in the MPLABX Options window (Main Menu: Tools⮞Options) under Team⮞Action Items.

|  |
| --- |
|  |

* 1. In order to complete each Action Item, you will need to select the correct Harmony API, pass the appropriate variables, and where necessary, handle the return values from the function. On page 78 you will find documentation on a subset of Harmony TCP and UDP APIs. Please carefully read the TODO item in the source code, and then use the documentation to find the correct API. If you encounter difficulty completing any Action Item, you can reference the solutions on page 86.

|  |  |  |
| --- | --- | --- |
| **Item** | **Source Code Line to place the solution** | **Description** |
| 1 | 253 | Open a UDP Server Instance, IPv4 address type, Port is defined in the ECS\_UDP\_BROADCAST\_PORT constant, Accept connections on ANY interface.  Store returned handle in the networkcomsData.ecsUDPBroadcastHandle variable |
| 2 | 269 | Check if any data has been received on the UDP Socket |
| 3 | 296 | Close the UDP Server Instance |
| 4 | 316 | Open a TCP Client Instance, IPv4 Address Type, Destination Port is defined in ECS\_TCP\_PORT constant. IP Address is stored in networkcomsData.ecsUDPSocketInfo.sourceIPaddress structure. Store returned handle in the networkcomsData.ecsTCPSocketHandle variable. Hint: You must pass the address of networkcomsData.ecsUDPSocketInfo.sourceIPaddress to the Harmony API that is used to open the TCP Client. |
| 5 | 333 | Check if the TCP Client is connected to the Server |
| 6 | 365 | Check if the TCP connection has dropped out any time between the current and previous execution of the NETWORKCOMS\_MANAGE\_TCP\_CLIENT state |
| 7 | 417 | Get the amount of free space available in the Transmit Buffer of the TCP Socket |
| 8 | 431 | Send the string stored in callReq\_JSONPacket array to the TCP Server. |

* 1. Once all 8 Action Items are complete, proceed with the rest of the lab.

## Project Build

* 1. Click on the Clean and Build Main Project icon in the MPLAB X Run toolbar.

|  |
| --- |
|  |

* 1. Check that the Build was Successful.

|  |
| --- |
|  |

## Programming

* 1. Unplug both the CAT5 Ethernet cable connected to the Classroom Network, and the USB Micro Cable (Console) from the SAM E70 Xplained Ultra.
  2. Click on the *Make and Program Device Main Project* Icon to Program the application onto your SAM E70 Xplained Ultra.

|  |
| --- |
|  |

* 1. Confirm LED3 on the SAM E70 Xplained Ultra is flashing after the programming process is completed.

## Application Testing

In this section, you are going to learn about the architecture of the state machine used in the Network Communications Controller application. You will have an opportunity to manually interact with the state machine to see the events that cause the state to change, and how the state machine handles data received/transmitted on the UDP/TCP sockets. The Harmony Console output will be used to monitor the Network Communications Controller events.

A state diagram of the Network Communications Controller application is depicted below.

|  |
| --- |
|  |

### Cable Connections

The cable connections required for the Application Testing are depicted in the diagram shown below. **The lab manual will state when each cable connection needs to be performed.**

|  |  |
| --- | --- |
| **3**  **1**  **2** | |
| **1** | **Network Connection**  Cable: CAT5 Ethernet Cable supplied with Starter Kit  Connection: RJ45 Jack on PCB Top to Laptop RJ45 Jack |
| **2** | **Programming Connection**  Cable: USB Male A to Male B Mini Cable supplied with Starter Kit  Connection: USB Debug Port on PCB Top to Laptop USB Port |
| **3** | **Console Connection**  Cable: USB Male A to Male B Micro Cable supplied with Starter Kit  Connection: USB Micro Connector on PCB Bottom to Laptop USB Port |

* 1. If Tera Term is still open from Lab 1, click on the window title bar to make it active. If Tera Term needs to be re-opened and configured, refer to the steps 1.82-1.86.
  2. Press the Enter key. The console will output five lines of text indicating the TCP/IP Stack has initialised, the value of the Host Name, and the status of the DHCP Client.

|  |
| --- |
|  |

The *Network Communications Controller* application contains console output functions that show the application state, and major TCP/IP events that the application executes (eg opening the UDP Server). The networkcoms[x]: prefix indicates the console message has originated from the *Network Communications Controller* application, and x indicates the current state of the application. (refer to the state numbers shown in the state diagram to decode the application state).

* 1. Connect a CAT5 Ethernet Patch cable between the RJ45 Jack on the SAM E70 Xplained Ultra and the RJ45 Jack on your PC. A message will appear on the console indicating the Network Link is “Up”. Note: It may take up to 30s for the Network Link to be established between the SAM E70 Xplained Ultra and the PC.

|  |
| --- |
|  |

* 1. Thirty seconds after the link is established, you will see a message on the console indicating the DHCP Client running on the PIC32 is disabled, and the DHCP Server is active. The next message to appear on the console indicates the UDP Server is open.

|  |
| --- |
|  |

### UDP Server Testing

The *Packet Sender* application will be used to test the UDP Server in the networkcoms application. Packet Sender is open source, and its primary function is to send and receive TCP and UDP Packets. The Packet Sender application implements TCP/UDP server and client functionality.

* 1. Open the *Packet Sender* Application by double clicking on the Packet Sender Icon on the Windows desktop.

|  |
| --- |
|  |

* 1. Go to the Tools Menu, and select Subnet Calculator. The IPv4 Subnet Calculator will show the IP Address and Subnet that has been assigned to the PC from the SAM E70 Xplained Ultra.

|  |
| --- |
| **IP Address of PC**  **Subnet of PC** |

* 1. In order to perform a broadcast onto the subnet, you need to determine the Broadcast address. The Subnet Calculator will calculate the broadcast address based upon the IP and Subnet addresses that have been assigned to the PC. You will need to enter the assigned non-loopback IP Address into the IP Text Box, and the assigned non-loopback Subnet into the Subnet Text Box.

The Broadcast text box will display the subnet broadcast address. Please take note of this broadcast address as it will be used to for setting up the UDP broadcast in the main window of the Packet Sender Application.

|  |
| --- |
| **Enter the values shown on your PC into the IP and Subnet boxes.**  **Calculated Broadcast**  **Address** |

* 1. Close the IPv4 Subnet Calculator window by pressing the Close button.
  2. The following settings are required to configure Packet Sender to transmit UDP broadcast messages:

|  |  |  |
| --- | --- | --- |
|  | **Setting** | **Value** |
| 1 | Name | ECS Broadcast |
| 2 | ASCII | {"broadcast":"ecs"} |
| 3 | Address | The broadcast address you calculated in step 2.46 |
| 4 | Port | 2027 |
| 5 | Protocol Selection | UDP |
| 6 | UDP | UDP Server Disabled |
| 7 | TCP | TCP Server Disabled |
| 8 | IP Mode | IPv4 Mode |
| 9 | Persistent TCP | Disabled (checkbox is unticked) |
| \*For options 6 and 7, pressing on the button at the bottom right of the Packet Sender Window will toggle between the Server being active and disabled. | | |

|  |
| --- |
| **Server and IP Version Mode Control**  **Transmission Log**  **Saved Packet List**  **Packet Transmission Configuration** |

* 1. Save the packet for future use by clicking on Save. The packet will be put into the Saved Packet List.
  2. To send the packet, click on the Send button located in the Saved Packet list.

|  |
| --- |
|  |

The sent packet will show in the packet transmission log.

|  |
| --- |
|  |

The console connected to the PIC32 will show all data received on the UDP Socket.

|  |
| --- |
|  |

The UDP Server will pass all received data to the ECSDATAPROCESS\_Parse function that performs the following tasks:

* Identify the start and end of the JSON packet by looking for open and close brace characters,
* Parse the packet using the JSMN Parsing tool,
* Check if the packet conforms to the JSON standard
* Extract the parameter names and data values from the packet.

The ECSDATAPROCESS\_Parse function can tolerate errors in the packet. All invalid packets will be ignored, and the function will wait for the start of the next JSON packet to be received.

When the networkcoms application receives a valid broadcast message, the source IP Address for the message will be determined, and the UDP Server will be shutdown. The application will then open the TCP Client, and attempt to connect to the TCP server. Since the TCP Server in Packet Sender is currently disabled, the networkcoms application will not be able to establish a connection. The networkcoms application will wait up to 15 seconds for the TCP Socket to connect and once this time has elapsed, the TCP Client is closed, and the UDP Server restarts.

|  |
| --- |
|  |

### TCP Client Testing

To test the operation of the TCP Client on the PIC, you will now configure the Packet Sender application to operate as a TCP Server.

* 1. In the menu toolbar, select File⮞Settings.

|  |
| --- |
|  |

* 1. Click on the Network Tab
  2. Under the Basic section, check Enable TCP Server.
  3. Under the Basic section, for TCP Server Port enter the value 3096.
  4. Press OK to exit the Settings window.

|  |
| --- |
|  |

* 1. Click on the Send button in the Saved Packet window to send the ECS Broadcast.

|  |
| --- |
|  |

Within 500ms of receiving the broadcast message, the PIC will transmit a status request packet: **{"request":"status"}.** The Harmony console output shows the TCP socket connected, and then it disconnected. The default TCP Server setup in Packet Sender is for a non-persistent connection. In a non-persistent configuration, Packet Sender will automatically terminate the connection after it has received data from the TCP client.

|  |
| --- |
|  |

* 1. In order for Packet Sender to keep the TCP Connection open, you need to check the Persistent TCP checkbox.

|  |
| --- |
|  |

* 1. Click on the Send button in the Saved Packet window to send another ECS Broadcast.

|  |
| --- |
|  |

* 1. The Persistent TCP connection window (TCP://You:3096) will now open, and you will see the Socket remains connected in the Console window.

|  |
| --- |
|  |

* 1. In the TCP://You:3096 window, click on the ASCII radio button, and click on the Append \r checkbox disable this option.

|  |
| --- |
|  |

#### Networkcoms TCP Communications Timeout Overview

Within a 500ms period after the TCP Connection is established, the Network Communications Controller application will send a *status request* packet, and the Floor Landing Controller networks communications application running on the PIC will wait up to 180s for a response from the server. In the event where no response was received from the server, the networkcoms application will disconnect and close the TCP Client. When the TCP Client is disconnected, the Packet Sender TCP Connection window will be become greyed out.

|  |
| --- |
|  |

TIP: To restart the connection, close the TCP://You:3096 window, and resend the broadcast message.

The timeout period for the response can be changed by modifying the ECS\_SERVER\_RESPONSE\_TIMEOUT\_PERIOD\_s constant on line 87 in networkcoms.c. The response timeout is managed by the application as opposed to the TCP/IP stack, using the System Timer Service.

* 1. In order to respond to the *status request* from the networkcoms application, you will need to transmit a *status response* packet. Enter the string {"response":"status","carid":1,"floorid":2,"callindicators":3} into the ASCII text box, and press Send.

|  |
| --- |
|  |

The transmitted packet will be added to the log in the TCP://You:3096 window.

|  |
| --- |
| **Packet transmitted to PIC** |

On the SAM E70 Xplained Ultra, both the Up (LED1) and Down (LED2) call indicators will be lit (as the "callindicators" value is 3 on the response packet). The PIC will automatically send another status request packet 500ms after receiving the response.

The console output will show the elevator and floor assignments.

|  |
| --- |
|  |

* 1. To turn both call indicators OFF, send the following packet from the TCP://You:3096 window: {"response":"status","callindicators":0}

|  |
| --- |
|  |

The transmitted packet will be added to the log in the TCP://You:3096 window.

|  |
| --- |
| **Packet transmitted to PIC** |

Confirm both Up (LED1) and Down (LED2) call indicators are in the off state.

* 1. Press the up call button (SW1 on the SAM E70 Xplained Ultra), and confirm you see the call request in the Packet Sender TCP window.

|  |
| --- |
| **Call Request from PIC** |

The console will output a message showing Call Button 2 is pressed. A second message shows the call request has been transmitted to the ECS.

|  |
| --- |
|  |

* 1. You will need to send a *call acknowledgement* back to the PIC before any further call requests can be transmitted. The response packet to send is {"response":"callack"}.

|  |
| --- |
|  |

A message indicating that the call acknowledgement was received will be shown in the console.

|  |
| --- |
|  |

The transmitted packet will be added to the log in the TCP://You:3096 window.

|  |
| --- |
| **Packet transmitted to PIC** |

* 1. As the networkcoms application will be waiting on a status request, send the following packet to avoid the TCP Socket being closed due to a response timeout: {"response":"status","callindicators":2}

|  |
| --- |
|  |

The transmitted packet will be added to the log in the TCP://You:3096 window.

|  |
| --- |
|  |

Confirm the Up (LED1) call indicator is ON, and Down (LED2) call indicator is OFF.

* 1. Now press the down call button (SW2 on the SAM E70 Xplained Ultra) and confirm you see the call request in the TCP://You:3096 window.

|  |
| --- |
|  |

The console will indicate that Call Button 1 was pressed, and the call request was transmitted to the ECS.

|  |
| --- |
|  |

The testing with Packet sender is now compete, and you can close the Packet Sender application.

## ECS Testing

You are now ready to interface the SAM E70 Xplained Ultra to the ECS Server. The cable connections required for this part

of the lab are depicted in the diagram shown below.

|  |  |
| --- | --- |
| **3**  **1**  **2** | |
| **1** | **Network Connection**  Cable: CAT5 Ethernet Cable from Classroom Network  Connection: RJ45 Jack on PCB Top |
| **2** | **Programming Connection**  Cable: USB Male A to Male B Mini Cable supplied with Starter Kit  Connection: USB Debug Port on PCB Top to Laptop USB Port |
| **3** | **Console Connection**  Cable: USB Male A to Male B Micro Cable supplied with Starter Kit  Connection: USB Micro Connector on PCB Bottom to Laptop USB Port |

* 1. Connect the CAT5 Ethernet Cable from the Classroom Ethernet Network to the RJ45 Jack on your Starter Kit.
  2. Within a few seconds of connecting the classroom network, you should see the Assigned Elevator Shaft and Assigned Floor appear in the Harmony Command Console running in Tera Term.

|  |
| --- |
|  |

* 1. If you look at the Elevator Control System shown on the Classroom Projector screen, you will see the Up and Down call status indicators have become active for your assigned floor. This indicates that the connection is active between the ECS and your LCS.

|  |
| --- |
|  |

* 1. Try pressing the Up Call button. The Up Call Indicator on the ECS will change to a solid white fill, and the elevator will start moving to your floor. The Up Call Lamp (LED1) on the SAM E70 Xplained Ultra will illuminate.

|  |
| --- |
|  |
|  |

Once the Elevator reaches your floor, the Up Call Lamp will turn off. Repeat your testing for the Down Call button. This completes the testing of the CBC with the ECS.

Congratulations, you have completed Lab 2!

Harmony TCP/IP API Subset For Lab 2

## TCP Socket Management Functions

### TCPIP\_TCP\_ArrayGet Function

This function reads an array of data bytes from a TCP socket's RX buffer/FIFO. The data is removed from the FIFO in the process.

#### Function Prototype

|  |
| --- |
| uint16\_t **TCPIP\_TCP\_ArrayGet**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**,  uint8\_t\* **buffer**,  uint16\_t **len**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| ****hTCP**** | The socket from which data is to be read. |
| ****buffer**** | Pointer to the array to store data that was read. |
| ****len**** | Number of bytes to be read. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| uint16\_t | The number of bytes read from the socket. If less than len, the RX FIFO buffer became empty or the socket is not connected. |

### TCPIP\_TCP\_ClientOpen Function

Provides a unified method for opening TCP client sockets. Sockets are created at the TCP module initialization, and can be claimed with this function and freed using TCPIP\_TCP\_Abort or TCPIP\_TCP\_Close. If the remoteAddress != 0 (and the address pointed by remoteAddress != 0) then the socket will immediately initiate a connection to the remote host.

#### Function Prototoype

|  |
| --- |
| [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **TCPIP\_TCP\_ClientOpen**(  [IP\_ADDRESS\_TYPE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14165.html) **addType**,  [TCP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22208.html) **remotePort**,  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html)\* **remoteAddress**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **addType** | The type of address being used. Valid values are: IP\_ADDRESS\_TYPE\_IPV4 or IP\_ADDRESS\_TYPE\_IPV6 |
| **remotePort** | TCP port to connect to. The local port for client sockets will be automatically picked by the TCP module. |
| **remoteAddress** | The remote address to be used |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) | Handle - Save this handle and use it when calling all other TCP APIs. If no sockets of the specified type were available to be opened, the handle will contain a value equal to INVALID\_SOCKET. |

### TCPIP\_TCP\_Close Function

Graceful Option Set: If the graceful option is set for the socket (default), a TCPIP\_TCP\_Disconnect will be tried. If the linger option is set (default) the TCPIP\_TCP\_Disconnect will try to send any queued TX data before issuing FIN. If the FIN send operation fails or the socket is not connected the abort is generated.

Graceful Option Not Set: If the graceful option is not set, or the previous step could not send the FIN, a TCPIP\_TCP\_Abort is called, sending a RST to the remote node. Communication is closed, the socket is no longer valid and the associated resources are freed.

#### Function Prototype

|  |
| --- |
| **void** **TCPIP\_TCP\_Close**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP socket should have been opened with TCPIP\_TCP\_ServerOpen/TCPIP\_TCP\_ClientOpen.

hTCP - valid socket

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hTCP | Handle to the socket to disconnect and close. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| Void | None |

### TCPIP\_TCP\_GetIsReady Function

Call this function to determine how many bytes can be read from the TCP RX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to wait for more data to arrive.

#### Function Prototype

|  |
| --- |
| uint16\_t **TCPIP\_TCP\_GetIsReady**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **hTCP** | The socket to check. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| uint16\_t | The number of bytes available to be read from the TCP RX buffer. |

### TCPIP\_TCP\_IsConnected Function

This function determines if a socket has an established connection to a remote node. Call this function after calling TCPIP\_TCP\_ServerOpen()/TCPIP\_TCP\_ClientOpen() to determine when the connection is set up and ready for use.

#### Function Prototype

|  |
| --- |
| **bool** **TCPIP\_TCP\_IsConnected**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hTCP | The TCP socket to check. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| bool | True: the socket is connected  False: the socket is disconnected |

### TCPIP\_TCP\_PutIsReady Function

Call this function to determine how many bytes can be written to the TCP TX buffer. If this function returns zero, the application must return to the main stack loop before continuing in order to transmit more data.

#### Function Prototype

|  |
| --- |
| uint16\_t **TCPIP\_TCP\_PutIsReady**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **hTCP** | The socket from which data is to be written. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| uint16\_t | The number of bytes available to be written in the TCP TX buffer. |

### TCPIP\_TCP\_StringPut Function

This function writes a null-terminated string to a TCP socket. The null-terminator is not copied to the socket.

#### Function Prototype

|  |
| --- |
| **const** uint8\_t\* **TCPIP\_TCP\_StringPut**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**,  **const** uint8\_t\* **Data**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| **hTCP** | The socket from which data is to be written. |
| **const** uint8\_t\* | **Data** |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| **const** uint8\_t\* | Pointer to the byte following the last byte written to the socket. If this pointer does not dereference to a NULL byte, the buffer became full or the socket is not connected. |

### TCPIP\_TCP\_WasReset Function

This function is a self-clearing semaphore indicating whether or not a socket has been disconnected since the previous call. This function works for all possible disconnections: a call to [TCPIP\_TCP\_Disconnect](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24093.html), a FIN from the remote node, or an acknowledgment timeout caused by the loss of a network link. It also returns true after the first call to TCPIP\_TCP\_Initialize. Applications should use this function to reset their state machines.

#### Function Prototype

|  |
| --- |
| **bool** **TCPIP\_TCP\_WasReset**(  [TCP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/22209.html) **hTCP**  ); |

#### Preconditions

TCP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hTCP | The TCP socket to check. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| bool | true: the socket was disconnected since the previous call  false: the socket remained connected since the previous call |

## UDP Socket Management Functions

### TCPIP\_UDP\_ArrayGet Function

This function reads an array of bytes from the UDP socket, while adjusting the current read pointer and decrementing the remaining bytes available. TCPIP\_UDP\_GetIsReady should be used before calling this function to get the number of the available bytes in the socket.

#### Function Prototype

|  |
| --- |
| uint16\_t **TCPIP\_UDP\_ArrayGet**(  [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **hUDP**,  uint8\_t \* **cData**,  uint16\_t **wDataLen**  ); |

#### Preconditions

UDP socket should have been opened with TCPIP\_UDP\_ServerOpen/TCPIP\_UDP\_ClientOpen.

hUDP - valid socket

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hUDP | UDP Socket Handle |
| cData | The buffer to receive the bytes being read. If NULL, the bytes are simply discarded |
| wDataLen | Number of bytes to be read from the socket. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [uint16\_t](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) | The number of bytes successfully read from the UDP buffer. If this value is less than wDataLen, then the buffer was emptied and no more data is available. |

### TCPIP\_UDP\_Close Function

Closes a UDP socket and frees the handle. Call this function to release a socket and return it to the pool for use by future communications.

#### Function Prototoype

|  |
| --- |
| **void** **TCPIP\_UDP\_Close**(  [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **hUDP**  ); |

#### Preconditions

UDP socket should have been opened with TCPIP\_UDP\_ServerOpen/TCPIP\_UDP\_ClientOpen.

hUDP - valid socket

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hUDP | UDP Socket Handle |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [void](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) | None |

### TCPIP\_UDP\_GetIsReady Function

This function will return the number of bytes that are available in the specified UDP socket RX buffer. The UDP socket queues incoming RX packets in an internal queue. If currently there is no RX packet processed (as a result of retrieving all available bytes with TCPIP\_UDP\_ArrayGet, for example), this call will advance the RX packet to be processed to the next queued packet. If a RX packet is currently processed, the call will return the number of bytes left to be read from this packet.

**Function Prototype**

|  |
| --- |
| uint16\_t **TCPIP\_UDP\_GetIsReady**(  [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **hUDP**  ); |

#### Preconditions

UDP socket should have been opened with [TCPIP\_UDP\_ServerOpen](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24228.html)/[TCPIP\_UDP\_ClientOpen](mk:@MSITStore:D:\\microchip\\harmony\\v1_08\\doc\\help_harmony.chm::/24205.html).

hUDP parameter is a valid socket

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hUDP | UDP Socket Handle |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [uint16\_t](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) | The number of bytes that can be read from the socket. |

### TCPIP\_UDP\_ServerOpen Function

Provides a unified method for opening UDP server sockets.

#### Function Prototype

|  |
| --- |
| [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **TCPIP\_UDP\_ServerOpen**(  [IP\_ADDRESS\_TYPE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14165.html) **addType**,  [UDP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24368.html) **localPort**,  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html)\* **localAddress**  ); |

#### Preconditions

UDP is initialized.

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| [IP\_ADDRESS\_TYPE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14165.html) **addType** | The type of address being used.  IP\_ADDRESS\_TYPE\_IPV4 or IP\_ADDRESS\_TYPE\_IPV6. |
| [UDP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24368.html) localPort | UDP port on which to listen for connections |
| [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html)\* localAddress | Local IP address to use. Can be 0 (NULL) if any incoming interface will do. |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) | Handle - Save this handle and use it when calling all other UDP APIs. If no sockets of the specified type were available to be opened, the handle will contain a value equal to INVALID\_SOCKET. |

### TCPIP\_UDP\_SocketInfoGet Function

This function will fill a user passed [UDP\_SOCKET\_INFO](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24374.html) structure with status of the selected socket

#### Function Prototype

|  |
| --- |
| **bool** **TCPIP\_UDP\_SocketInfoGet**(  [UDP\_SOCKET](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24369.html) **hUDP**,  [UDP\_SOCKET\_INFO](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24374.html)\* **pInfo**  ); |

#### Preconditions

UDP socket should have been opened with [TCPIP\_UDP\_ServerOpen](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24228.html)()/[TCPIP\_UDP\_ClientOpen](mk:@MSITStore:D:\\microchip\\harmony\\v1_08\\doc\\help_harmony.chm::/24205.html)()().

hUDP - valid socket

pInfo - valid address of a [UDP\_SOCKET\_INFO](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24374.html) structure

#### Parameters

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| hUDP | UDP Socket Handle |
| pInfo | Pointer to [UDP\_SOCKET\_INFO](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24374.html) to receive socket information |

#### Returns

|  |  |
| --- | --- |
| **Type** | **Description** |
| bool | true if call succeeded  false if no such socket or invalid pinfo. |

### UDP\_SOCKET\_INFO Structure

Holds information about a UDP Socket

#### Structure

|  |
| --- |
| **typedef** **struct** {  [IP\_ADDRESS\_TYPE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14165.html) **addressType**;  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html) **remoteIPaddress**;  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html) **localIPaddress**;  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html) **sourceIPaddress**;  [IP\_MULTI\_ADDRESS](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/14171.html) **destIPaddress**;  [UDP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24368.html) **remotePort**;  [UDP\_PORT](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/24368.html) **localPort**;  [TCPIP\_NET\_HANDLE](mk:@MSITStore:D:\microchip\harmony\v1_08\doc\help_harmony.chm::/23778.html) **hNet**;  } UDP\_SOCKET\_INFO; |

#### Members

|  |  |  |
| --- | --- | --- |
| **Type** | **Member Name** | **Description** |
| IP\_ADDRESS\_TYPE | addressType | address type of the socket |
| IP\_MULTI\_ADDRESS | remoteIPaddress | current socket destination address |
| IP\_MULTI\_ADDRESS | localIPaddress | current socket source address |
| IP\_MULTI\_ADDRESS | sourceIPaddress | source address of the last packet |
| IP\_MULTI\_ADDRESS | destIPaddress | destination address of the last packet |
| UDP\_PORT | remotePort | Port number associated with remote node |
| UDP\_PORT | localPort | local port number |
| TCPIP\_NET\_HANDLE | hNet | associated interface |

# Network Communications Controller Application Code Modification Solutions

|  |  |  |
| --- | --- | --- |
| Item | Source Line | Description & Solution |
| 1 | 253 | Open a UDP Server Instance, IPv4 address type, Port is defined in the ECS\_BROADCAST\_PORT constant, Accept connections on ANY interface.  Store returned handle in the ecsUDPBroadcastHandle variable  ecsUDPBroadcastHandle = TCPIP\_UDP\_ServerOpen(IP\_ADDRESS\_TYPE\_IPV4, ECS\_BROADCAST\_PORT, 0); |
| 2 | 269 | Check if any data has been received on the UDP Socket  TCPIP\_UDP\_GetIsReady(ecsUDPBroadcastHandle) |
| 3 | 296 | Close the UDP Server Instance  TCPIP\_UDP\_Close(ecsUDPBroadcastHandle); |
| 4 | 316 | Open a TCP Client Instance, IPv4 Address Type, Destination Port is defined in ECS\_TCP\_PORT constant. IP Address is stored in ecsUDPSocketInfo.sourceIPaddress structure. Store returned handle in the ecsTCPSocketHandle variable. Hint: You must pass the address of ecsUDPSocketInfo.sourceIPaddress to the Harmony API that is used to open the TCP Client.  ecsTCPSocketHandle = TCPIP\_TCP\_ClientOpen(IP\_ADDRESS\_TYPE\_IPV4, ECS\_TCP\_PORT, &ecsUDPSocketInfo.sourceIPaddress); |
| 5 | 333 | Check if the TCP Client is connected to the Server  TCPIP\_TCP\_IsConnected(ecsTCPSocketHandle) |
| 6 | 365 | Check if the TCP connection has dropped out any time between the current and previous execution of the NETWORKCOMS\_MANAGE\_TCP\_CLIENT state.  TCPIP\_TCP\_WasReset(ecsTCPSocketHandle) |
| 7 | 417 | Get the amount of free space available in the Transmit Buffer of the TCP Socket  TCPIP\_TCP\_PutIsReady(ecsTCPSocketHandle) |
| 8 | 431 | Send the string stored in callReq\_JSONPacket array to the TCP Server.  TCPIP\_TCP\_StringPut(ecsTCPSocketHandle, callReq\_JSONPacket);  TCPIP\_TCP\_ArrayPut can also be used, but the length of the string needs to be specified e.g.  TCPIP\_TCP\_ArrayPut(ecsTCPSocketHandle, callReq\_JSONPacket, strlen(callReq\_JSONPacket));  Alternatively the size of the array can specified, but only when the array size is equal to the string size.  TCPIP\_TCP\_ArrayPut(ecsTCPSocketHandle, callReq\_JSONPacket, sizeof(callReq\_JSONPacket)); |

**MPLAB® Harmony TCP/IP Stack**

# TCP Module API Function List

## Socket Management Functions

|  |  |
| --- | --- |
| **TCPIP\_TCP\_ServerOpen** | Opens a TCP socket as a server. |
| **TCPIP\_TCP\_ClientOpen** | Opens a TCP socket as a client. |
| **TCPIP\_TCP\_Close** | Disconnects an open socket and destroys the socket handle, releasing the associated resources. |
| **TCPIP\_TCP\_Connect** | Connects a client socket. |
| **TCPIP\_TCP\_Bind** | Binds a socket to a local address. |
| **TCPIP\_TCP\_RemoteBind** | Binds a socket to a remote address. |
| **TCPIP\_TCP\_IsConnected** | Determines if a socket has an established connection. |
| **TCPIP\_TCP\_WasReset** | Self-clearing semaphore indicating socket reset. |
| **TCPIP\_TCP\_Disconnect** | Disconnects an open socket. |
| **TCPIP\_TCP\_Abort** | Aborts a connection. |
| **TCPIP\_TCP\_OptionsGet** | Allows getting the options for a socket like: current RX/TX buffer size, etc. |
| **TCPIP\_TCP\_OptionsSet** | Allows setting options to a socket like adjust RX/TX buffer size, etc. |
| **TCPIP\_TCP\_SocketInfoGet** | Obtains information about a currently open socket. |
| **TCPIP\_TCP\_SocketNetGet** | Gets the current network interface of an TCP socket. |
| **TCPIP\_TCP\_SocketNetSet** | Sets the interface for an TCP socket |
| **TCPIP\_TCP\_SignalHandlerDeregister** | Deregisters a previously registered TCP socket signal handler. |
| **TCPIP\_TCP\_SignalHandlerRegister** | Registers a TCP socket signal handler. |
| **TCPIP\_TCP\_Task** | Standard TCP/IP stack module task function. |

## Transmit Data Functions

|  |  |
| --- | --- |
| **TCPIP\_TCP\_Put** | Writes a single byte to a TCP socket. |
| **TCPIP\_TCP\_PutIsReady** | Determines how much free space is available in the TCP TX buffer. |
| **TCPIP\_TCP\_StringPut** | Writes a null-terminated string to a TCP socket. |
| **TCPIP\_TCP\_ArrayPut** | Writes an array from a buffer to a TCP socket. |
| **TCPIP\_TCP\_Flush** | Immediately transmits all pending TX data. |
| **TCPIP\_TCP\_FifoTxFullGet** | Determines how many bytes are pending in the TCP TX FIFO. |
| **TCPIP\_TCP\_FifoTxFreeGet** | Determines how many bytes are free and could be written in the TCP TX FIFO. |

## Receive Data Transfer Functions

|  |  |
| --- | --- |
| **TCPIP\_TCP\_ArrayFind** | Searches for a string in the TCP RX buffer. |
| **TCPIP\_TCP\_Find** | Searches for a byte in the TCP RX buffer. |
| **TCPIP\_TCP\_Get** | Retrieves a single byte to a TCP socket. |
| **TCPIP\_TCP\_Peek** | Peaks at one byte in the TCP RX buffer/FIFO without removing it from the buffer. |
| **TCPIP\_TCP\_Discard** | Discards any pending data in the RCP RX FIFO. |
| **TCPIP\_TCP\_FifoRxFreeGet** | Determines how many bytes are free in the RX buffer/FIFO. |
| **TCPIP\_TCP\_FifoSizeAdjust** | Adjusts the relative sizes of the RX and TX buffers. |
| **TCPIP\_TCP\_FifoRxFullGet** | Determines how many bytes are pending in the RX buffer/FIFO. |
| **TCPIP\_TCP\_GetIsReady** | Determines how many bytes can be read from the TCP RX buffer. |
| **TCPIP\_TCP\_ArrayGet** | Reads an array of data bytes from a TCP socket's RX buffer/FIFO. |
| **TCPIP\_TCP\_ArrayPeek** | Reads a specified number of data bytes from the TCP RX buffer/FIFO without removing them from the buffer. |

