

**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.*



























































































































































































































LAB 2

# Introduction

You will be implementing an embedded network application to update and track status of an Vending Machine (VM).

The implementation is sectioned into two modules.

## TASK 1:

E70 HTTP- SERVER

/Vending Machine -1

Ethernet Switch

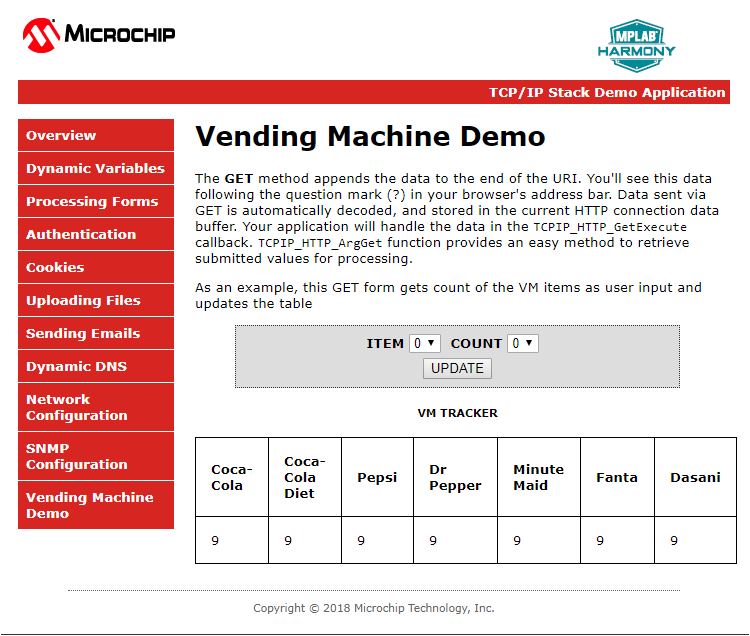
PC – HTTP-Client (1)

E70 HTTP-SERVER /Vending Machine-(n)

PC- HTTP-Client (n)

The task is to interface the up/down and select buttons and an OLED of a Vending Machine to the HTTP-Server. The HTTP-Server keeps track of the items in the Vending Machine.

The HTTP- Client (PC-web page) talks to the HTTP- server and displays the status of the Vending Machine. The HTTP-Client also sends out an update request to the HTTP- server using Ethernet Interface.



## TASK 2:

The task is to make all the VMs in the class network to talk to a server. When a Bay of a VM is empty it sends out a message to a common server.

Ethernet Switch

E70 HTTP-SERVER / Vending Machine (1)

SERVER

E70 HTTP- SERVER / Vending Machine (n)

An example Message:

“Message: 1 from Martin: The Pepsi Bay is empty”

The Message number (1) and the Host name (Martin) together forms a unique Message and makes it easy to differentiate your message from other VMs on the network server and the number helps to find the recent ones.

# Data Protocol

# TASK1

GET /[VM.htm?ITEM=2&COUNT=5](http://10.13.33.91/VM.htm?ITEM=2&COUNT=5) HTTP/1.1



**TCP HTTP Port 80**

Sends the request “**GET /MY\_Data.xml HTTP/1.1**” for every 500ms



PC HTTP-Client-1

**TCP HTTP Port 80**

HTTP/1.1 200 OK

Sends the MY\_Data.xml file to update the web page with current data



PC -HTTP-Client-1

**TCP HTTP Port 80**

PC HTTP-Client-1



VM-1

# 

|  |
| --- |
| VM-1 |

VM-1

# TASK 2



**BAY Empty Message**

**BAY Empty Message**

**BAY Empty Message**

VM-1

CLASS SERVER

VM-2

VM-3

# Application Implementation

The Vending Machine demo (Task 1 and 2) is implemented using “**SAM E70 Xplained Ultra Evaluation Kit** “and “**OLED1 Xplained Pro Extension Kit”.** The OLED Xplained prohas three buttons, LEDs and an OLED.

* The VM application uses Button 1(Down) and 2(UP) to scroll through the VM items and Button 3 to select an item from the Vending Machine.
* The LEDs above the button indicates a button press with a blink.
* The OLED is used to display the name and their corresponding number of the items in a Vending Machine.
* When select button(B3) is pressed the HTTP-Server will decrement the count of an item and updates the HTTP-Client(Web page)through ethernet interface.



E70 WEB SERVER

/Vending Machine

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

The Vending machine demo runs the below user applications in conjunction with MPLAB Harmony TCP/IP Stack:

* **OLED and Button Contoller** (MMI\_Tasks): Manages the operation of the OLED display , Buttons and LEDs.
* **Application Contoller** (app\_Tasks()): manages all high level network communications with client and server.

# 

TCPIP\_STACK\_Tasks ()

APP\_Tasks ()

SYS\_CMD\_TASKS()

SYS\_FS\_TASKS()

DRV\_MIIM\_TASKs ()

NET\_PRES\_Tasks ()

SYS\_FS\_TASKS()

NET\_PRES\_Tasks ()

# 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.. The source code for the OLED and Button Controller are included into the project.

You will learn about several Harmony TCP/IP API functions by adding the necessary APIs into the application source code to manage the TCP Sockets and data exchanged with socket. You will also learn how the dynamic variables in a web page are handled by the server. Finally, you will get to use the Packet Sender software tool to perform isolated testing of the embedded application prior to connecting your VM implementation to the server on the network.

# Lab Procedure

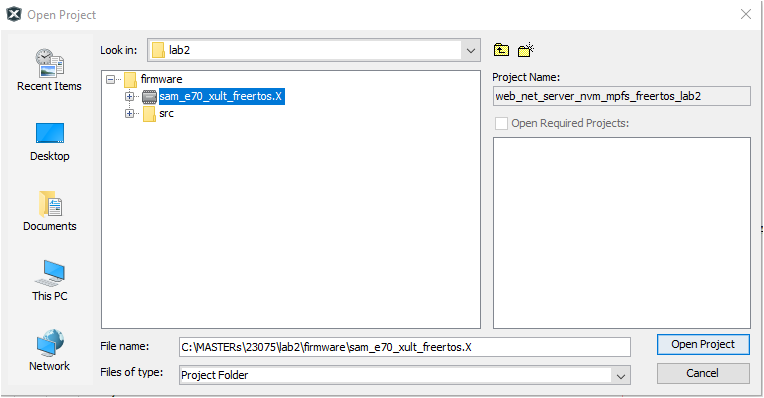
## Project Setup

* 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 web\_net\_server\_nvm\_mpfs\_lab2, and has a number of files added including mmi.c and modified app.c file are added to 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\ web\_net\_server\_nvm\_mpfs\_lab2\firmware into the File name text box or navigate to the file .
  2. Click on sam\_e70\_xult\_freetos.X icon in the file list.
  3. Press Open Project.

|  |
| --- |
|  |

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

|  |
| --- |
|  |

## MHC Code Generation

* 1. The project configurations are already set, and we are going to look on the configurations and generate the code. (steps 1.8 -1.22)
  2. Before the MHC Configuration tool can be used, you need to set lab2 as the main project, by right clicking on the web\_net\_server\_nvm\_mpfs\_lab2 folder under the Projects window, and choosing Set as Main Project in the popup menu.

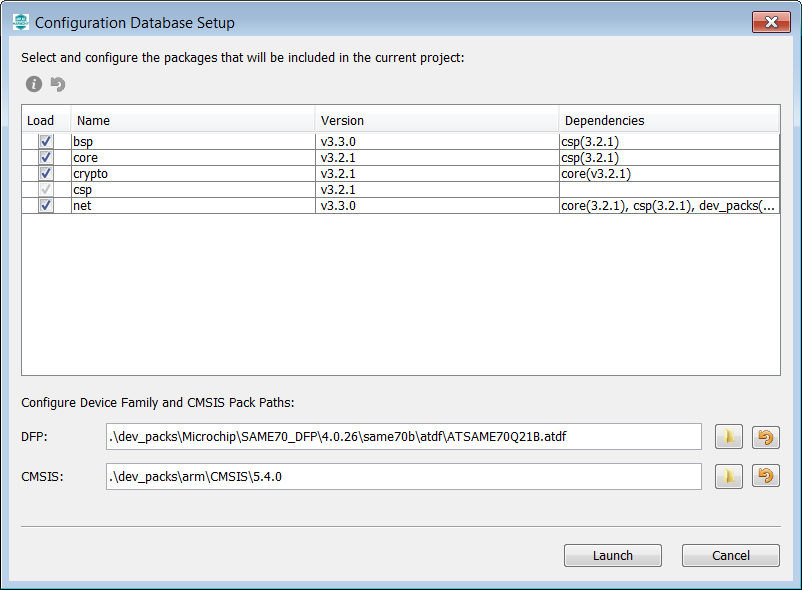
|  |  |
| --- | --- |
|  |  |

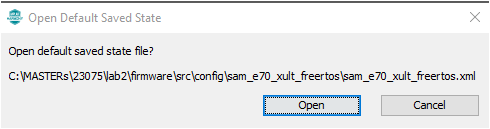
* 1. To launch MHC, choose Tools⮞Embedded⮞MPLAB Harmony 3 Configurator in the main menu.

|  |
| --- |
|  |

* 1. In the MPLAB Harmony Launcher dialog box, make sure the path states   
     C:\MASTERs\23075\h3 and click on Launch.

|  |
| --- |
|  |

* 1. In the nest window the H3 parts and their used Version Number are displayed. Click on Launch
  2. Open the saved state file



* 1. Explore and look on the configurations tree by clicking on the modules under Active Components (step 1.14 and 1.15- can be skipped and continue from 1.16).
  2. Click and check the “**Core”** option under Active Components. On right hand side the Configuration options are displayed, check the “Number of Applications “options under Configuration Options which is set to 2.Expand both Application 0 and 1 Configuration trees, and the Application Name is set to app (Application Interface) for Application 0, and mmi(Man Managed Interface for buttons, leds and OLED)for Application 1.
  3. Check on TCP/IP Stack under Active components and its configuration under Configuration options.

|  |
| --- |
|  |







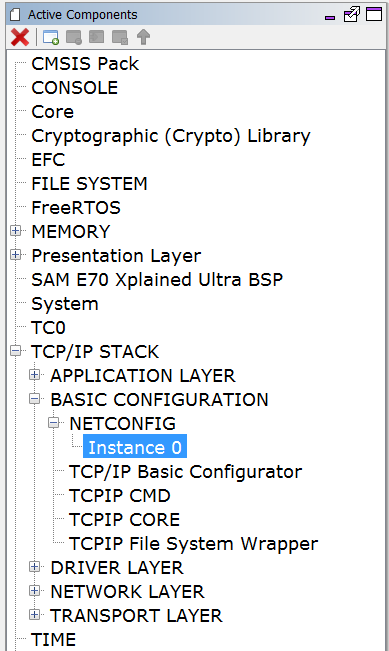


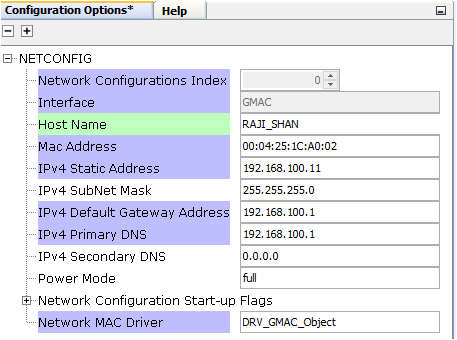


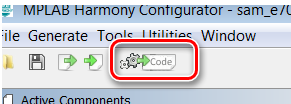
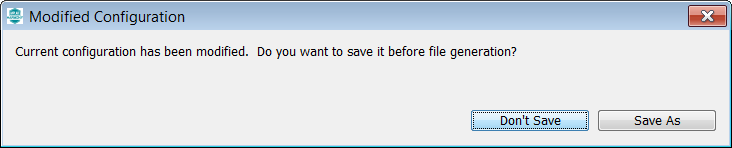
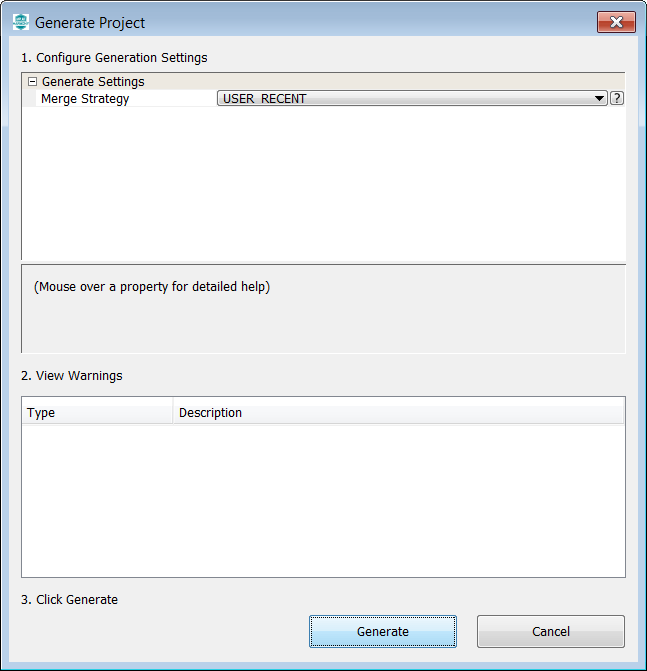


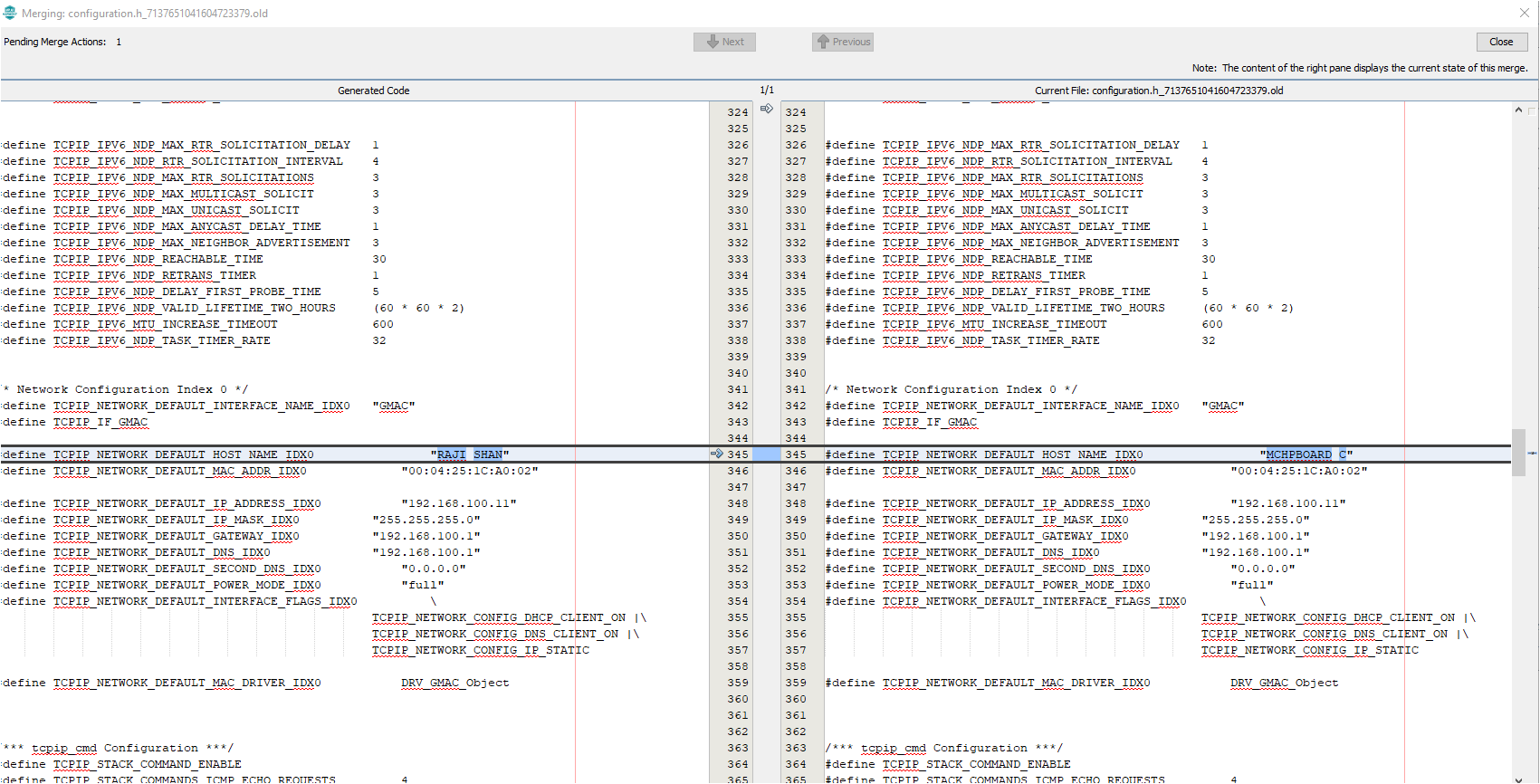


* 1. Select the **Instance 0** in the **TCP/IP stack** under **Active Components** to enter the HOST Name **(TCP/IP STACK->BASIC CONFIGURATION->Instance 0).** And change in the **Configuration Options** (displayed on the right side of MPlabx) the Host Name to your firstname\_lastname. The Host name can be identified in the Network.

****



* 1. Select “Code” (Generate Code) which is under the menu tabs.  
     
  2. Select Don’t Save  
     
  3. Select Generate  
     
  4. Some Files will be changed, and the MHC is asking in a “diff” window, if the changes should be taken over. Accept all changes in the file by clicking on the Arrow in the middle above and then close.



1

2

* 1. The whole process is displayed with a progress bar  
     











* 1. After the MHC has finished generating the project, go to the Projects Window and expand the Header Files and Source Files to see the source/header files for the *app and mmi* application file and TCP/IP stack files..

|  |
| --- |
|  |





















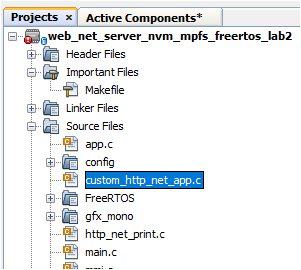


## Application Source and Header File SetupTASK1:

In the VM application when an VM ITEM update is initiated by the HTTP- client (WEB PAGE) it sends out a GET command that needs to be processed by the HTTP-Server. The get command is sent along with the argument of the VM “Item” to be updated and its “Count” which are processed by the HTTP-Server. In Harmony TCP/IP stack GET command is handled by the function TCPIP\_HTTP\_NET\_ConnectionGetExecute ().

Follow the procedure under to include the code that process the GET request from HTTP-CLient: -

* 1. Click and Open the file custom\_http\_net\_app.c. Search for the function TCPIP\_HTTP\_NET\_ConnectionGetExecute () (Approx. Line.No. 247) in which the below code snippets need to be inserted.



* 1. Find the TODO[1] and insert the below code starting from the line mentioned as ***“//<--Insert the solution for TODO[4] starting on this line”*** . Code can be inserted either by typing or copy and paste which is provided for your convenience at the end of the file (custom\_http\_net\_app.c).

ptr = TCPIP\_HTTP\_NET\_ArgGet(httpDataBuff, (const uint8\_t \*)"ITEM");

* 1. Find the TODO[2] and insert the code snippet starting from the line mentioned as ***“//<--Insert the solution for Item 2 starting on this line”*** . Code can be inserted either by typing or copy and paste which is provided for your convenience at the end of the file (custom\_http\_net\_app.c).

ptr = TCPIP\_HTTP\_NET\_ArgGet(httpDataBuff, (const uint8\_t \*)"COUNT");

VM\_Count[Update\_Item]=\*ptr;

## TASK 2:

When a Bay of the VM is empty it sends out the message to the common server. The sate machine to open a socket to send message and close socket is shown below

**Stack Initialized**

**NO**

**YES**

**File Mounted**

**No link found**

**If new link found**

**Stack not Initialized**

**If MAC address is not read**

**MAC address READ**

**If WEB file is not mounted**

**not mounted**

READ

MAC address

TCP/IP stack Int

**Link is established**

**Connection Established**

**If wait time out**

Send out the TCP message

“Bay is empty”

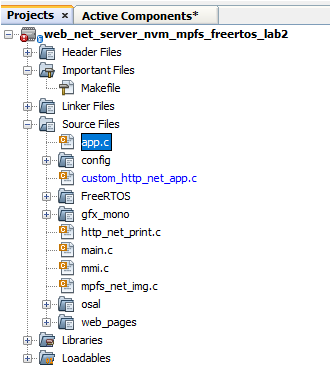
Close connection

Set-up TCP client

Display the IP address

In this section, you will gain some experience with the use of Harmony TCP/IP API functions. The app.c source is missing lines of code. All missing code specifically relates to management of the setting up the TCP Client such as opening or closing the socket, checking if the socket is connected, checking if data is available, 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. The first step to do is get the **common server’s IP address**. Get the address from the presenter and make a note.
  2. Click and Open the file app.c. and find the function APP\_Tasks (). (Approx. Line.No. 200) in which the below code snippets are inserted.



* 1. Find the TODO[3] in which common SERVER IP address is entered. Replace the x with the common server IP address ().

For example: if the server IP address is 192.168.0.108 replace

\* x1 with 192

\* x2 with 168

\* x3 with 0

\* x4 with 108

* Find TODO[4] and add the below snippet to Open a TCP\_Client . Code can be inserted either by typing or copy and paste the code which is provided for your convenience at the end of the file (app.c).

appData.socket = TCPIP\_TCP\_ClientOpen(IP\_ADDRESS\_TYPE\_IPV4,

appData.port,

(IP\_MULTI\_ADDRESS\*) & addr);

* Find TODO[5] and add the below snippet to check the TCP connection . Code can be inserted either by typing or copy and paste the code which is provided for your convenience at the end of the file (app.c).

if (!TCPIP\_TCP\_IsConnected(appData.socket))

<--Insert the solution for Item 1 starting on this line

insert the code

TASK 2:



|  |
| --- |
|  |

* 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. |

