

# 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.*



**MICROCHIP**

## Table of Contents

<b>Introduction</b>	<b>4</b>
<b>Hardware Requirements</b>	<b>5</b>
<b>Software Requirements</b>	<b>6</b>
<b>How to connect the USB and the CAT5 Network cable to the board</b>	<b>7</b>
<b>Lab 1</b>	<b>8</b>
<i>Overview</i>	<b>8</b>
<i>Lab Procedure</i>	<b>9</b>
Starting MPLAB X IDE.....	9
Project Load, modify with MHC, Generate, build and run .....	10
<b>Lab 2</b>	<b>29</b>
<i>Overview</i>	<b>29</b>
<b>Lab 3</b>	<b>30</b>
<i>Overview</i>	<b>30</b>
<i>Lab Procedure</i>	<b>33</b>
<b>MPLAB® Harmony TCP/IP Stack</b>	<b>36</b>
<b>TCP Module API Function List</b>	<b>36</b>
Socket Management Functions.....	36
Transmit Data Functions .....	36
Receive Data Transfer Functions .....	36
<b>UDP Module API Function List</b>	<b>38</b>
Socket Management Functions.....	38
Transmit Data Functions .....	38
Receive Data Transfer Functions .....	38
<b>Harmony TCP/IP API Subset For all Lab's</b>	<b>39</b>
<b>TCP Socket Management Functions</b>	<b>39</b>
<b>TCPIP_TCP_ArrayGet Function</b>	<b>39</b>
Function Prototype.....	39
TCPIP_TCP_ClientOpen Function.....	39
TCPIP_TCP_Close Function.....	40
TCPIP_TCP_GetIsReady Function .....	40
TCPIP_TCP_IsConnected Function.....	41
TCPIP_TCP_PutIsReady Function.....	41
TCPIP_TCP_StringPut Function.....	42

TCPIP_TCP_WasReset Function .....	42
<b>UDP Socket Management Functions</b>	<b>43</b>
TCPIP_UDP_ArrayGet Function.....	43
TCPIP_UDP_Close Function .....	44
TCPIP_UDP_GetIsReady Function.....	44
TCPIP_UDP_ServerOpen Function .....	45
TCPIP_UDP_SocketInfoGet Function.....	45
UDP_SOCKET_INFO Structure.....	46

## Introduction

This Lab Manual provides the step by step procedure to complete two labs in the MASTERS 23075 IoT6 Class.

If you don't know how TCP/IP works but you have to add network connectivity to your product, this is the right class for you. We will teach you the basics of TCP/IP, how the client-server model works, what ports and sockets are and how applications use them to create TCP/IP connections. The class will also teach you the fundamentals of network analysis with the well-known tool Wireshark. The hands-on part of the class utilizes Microchip's 32-bit MCUs with MPLAB® Harmony. You can learn the architecture and the fundamentals of the Harmony TCP/IP stack to interface your TCP/IP application with some common stack APIs. To make your life easier, the FreeRTOS™ task scheduler is used to simplify your application programming.

In Lab 1 we will open a TCP project, do some stack re-configuration and a connectivity check and in Lab 2 we will show an Application integration for local access, using the example of a Vending machine. Finally in Lab 3 we will make an Application integration for external access, using the example of a Weather Service.

## Hardware Requirements

The following hardware is required:

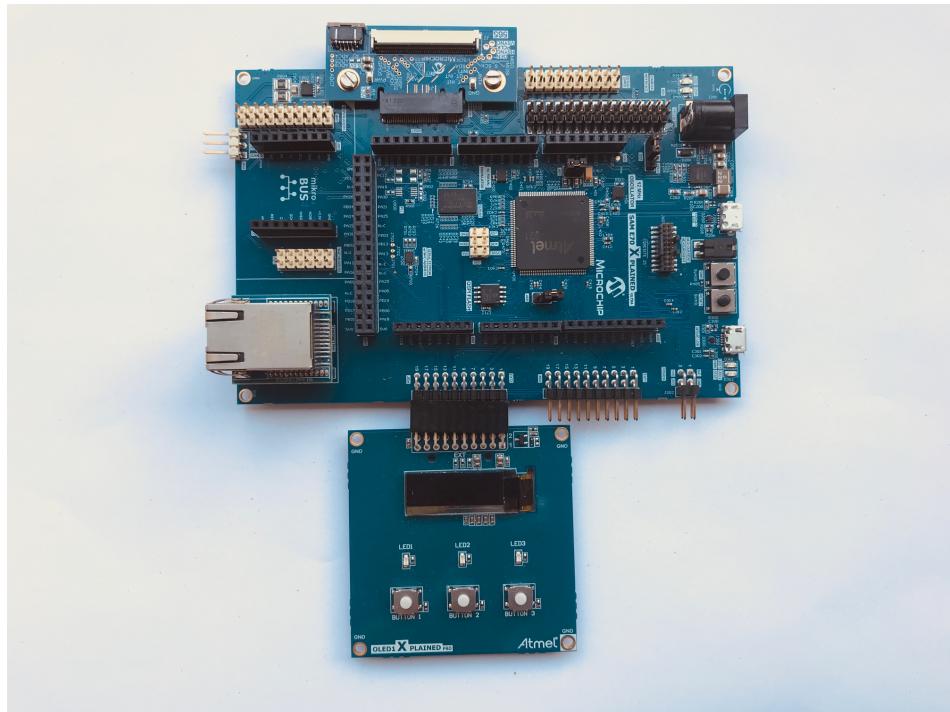
- **SAM E70 Xplained Ultra** (Microchip Part Number: DM320113)
  - <https://www.microchip.com/DevelopmentTools/ProductDetails/PartNO/DM320113>



- **OLED1 Xplained Pro extension kit** (Microchip Part Number: ATOLED1-XPRO)
  - <https://www.microchip.com/Developmenttools/ProductDetails/ATOLED1-XPRO>



- **Cat 5 Ethernet Patch Cable**
- **USB Male A to USB Male B Micro Cable**



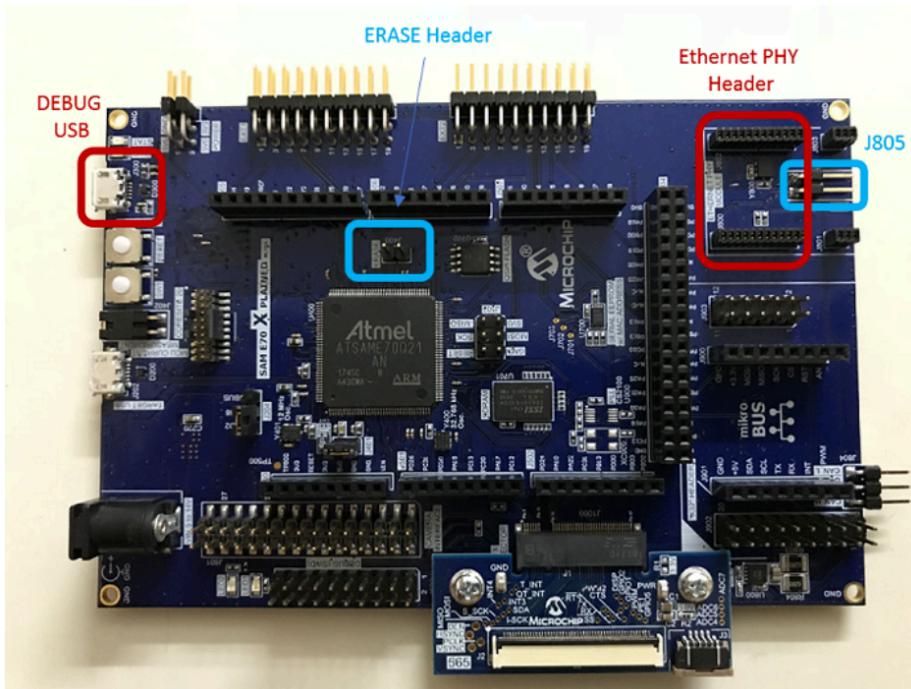
## Software Requirements

The following software is required:

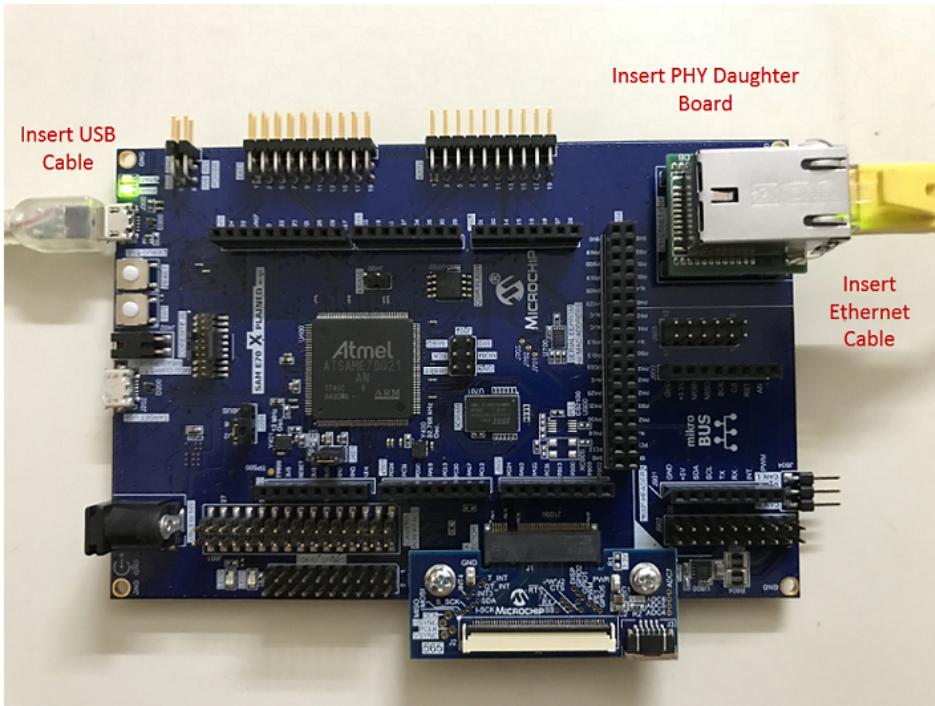
- **Microchip MPLAB X IDE v5.20**
  - <http://www.microchip.com/mplab>
- **Microchip MPLAB XC32 Compiler v2.15**
  - <http://www.microchip.com/mplab/compilers>
- **Microchip MPLAB Harmony 3**
  - <http://www.microchip.com/mplab/mplab-harmony>
- **Microchip MPLAB Harmony Configuration (MHC) Tool Plugin v3.3.0.1**
- **Tera Term v4.95**

## How to connect the USB and the CAT5 Network cable to the board

- Ensure the Erase jumper is open
- Open the J805 jumper



- Insert the LAN8740 PHY daughter board on the ETHERNET PHY MODULE header.
- Connect the micro USB cable from the computer to the DEBUG USB connector on the SAM E70 Xplained Ultra Evaluation Kit
- Establish a connection between the router/switch with the SAM E70 Xplained Ultra Evaluation Kit through the RJ45 connector



## Lab 1

### Overview

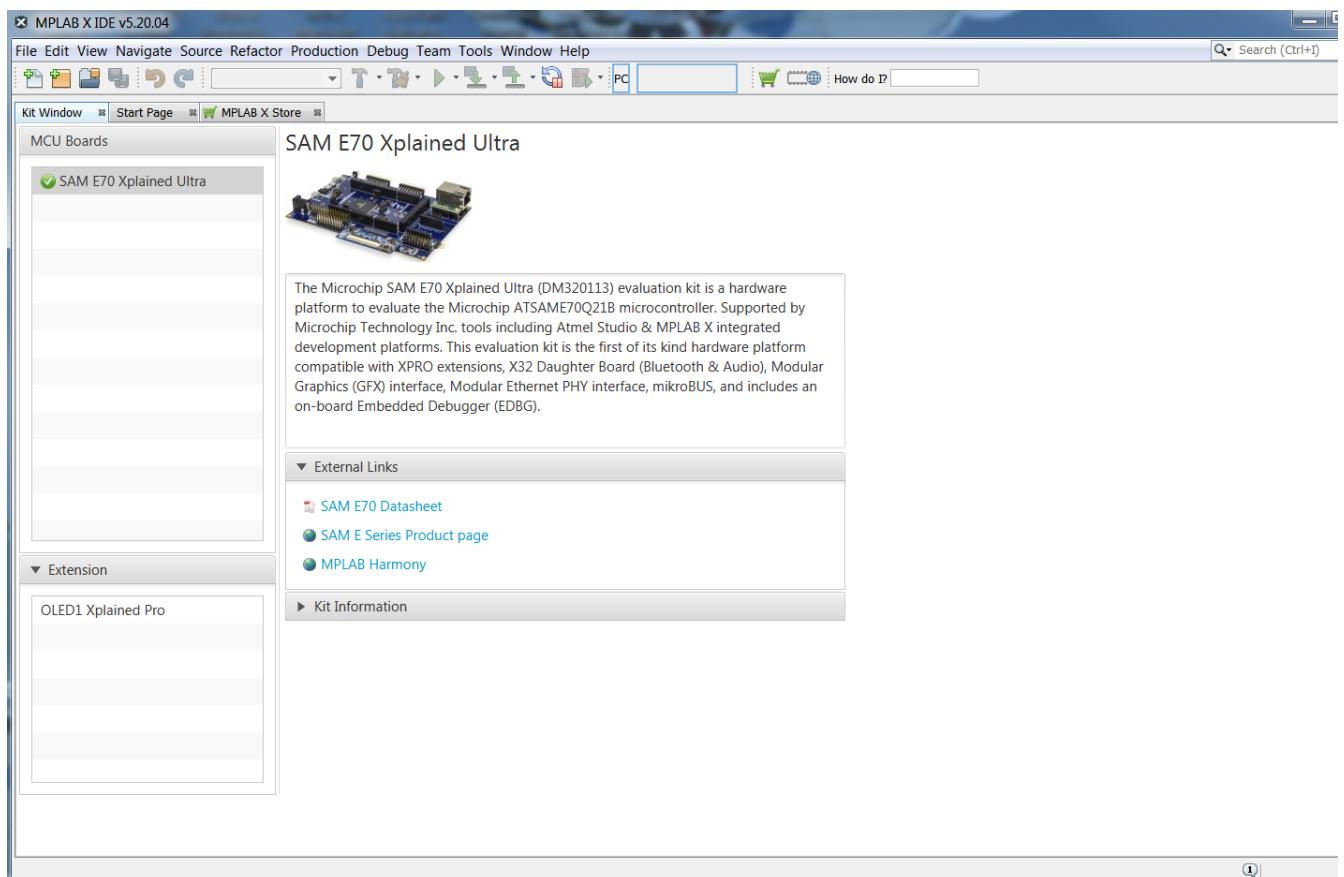
Lab 1 will show you how to open an existing TCP/IP MPLAB Harmony 3 Project and using the MPLAB Harmony Configuration (MHC) Tool. The project will incorporate basic TCP/IP functionality to allow the SAM E70 Xplained Ultra to connect to an Ethernet Network, along with a simple application to flash a “Heartbeat” 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:

- Open a SAM E70 MPLAB X Project
- Configuring the MPLAB Harmony path
- Configuring the TCP/IP Stack options, including:
  - Network Configuration of the Host Name
  - TCP/IP Services including Dynamic Host Configuration Protocol Client, ICMPv4 Server (for Ping testing) & Announce Discovery Tool
  - Bandwidth testing with “iperf”
- 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).  
Toggling the IO Pin that drives USER\_LED0 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

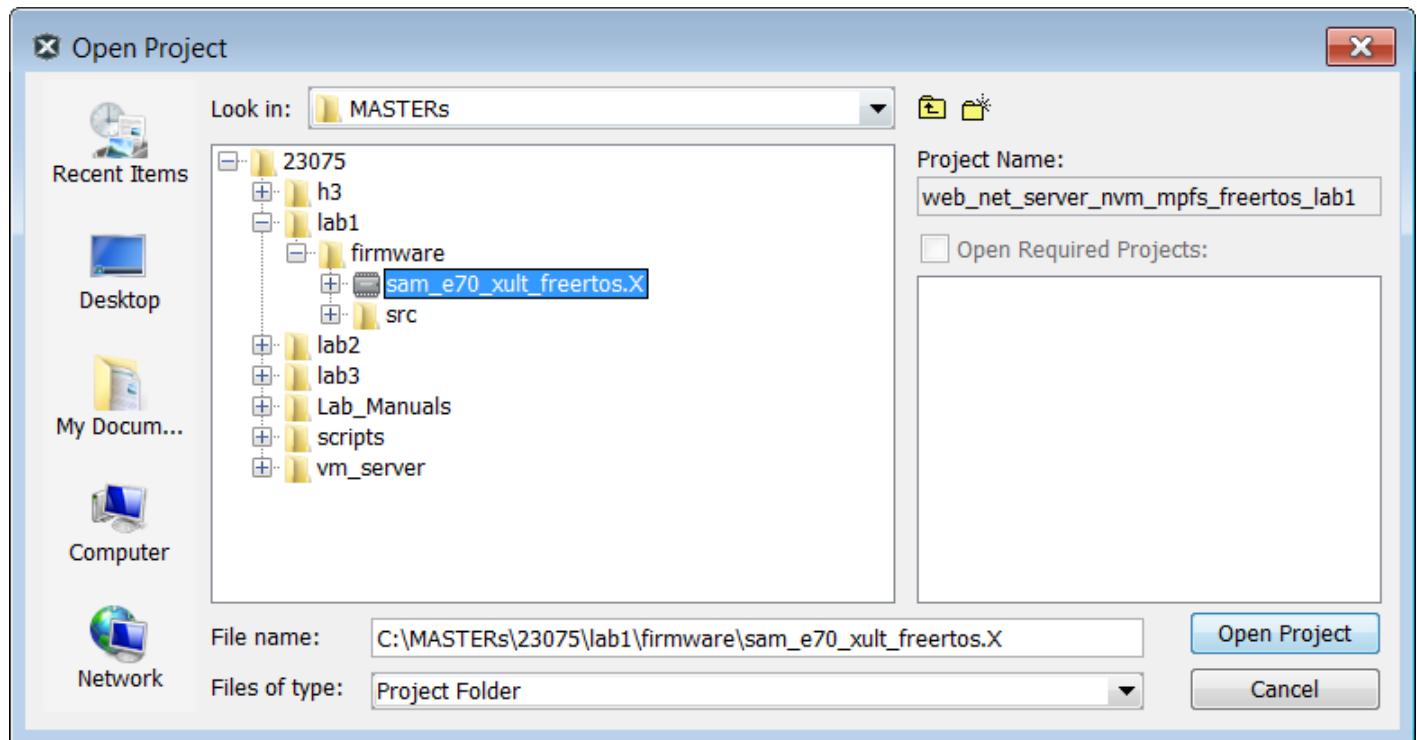
### Starting MPLAB X IDE

- 1.1. Start MPLAB X IDE by double clicking on the MPLAB X IDE v5.20 icon found on the Windows desktop.



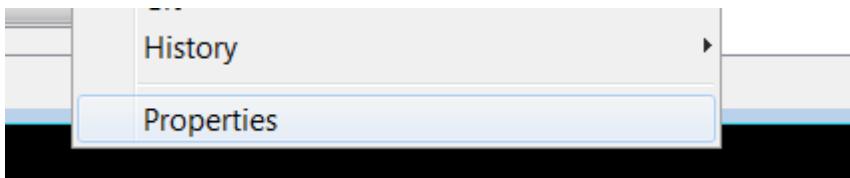
## Project Load, modify with MHC, Generate, build and run

1. Open Project by choosing File>Open Project... from the main menu and select lab1 project

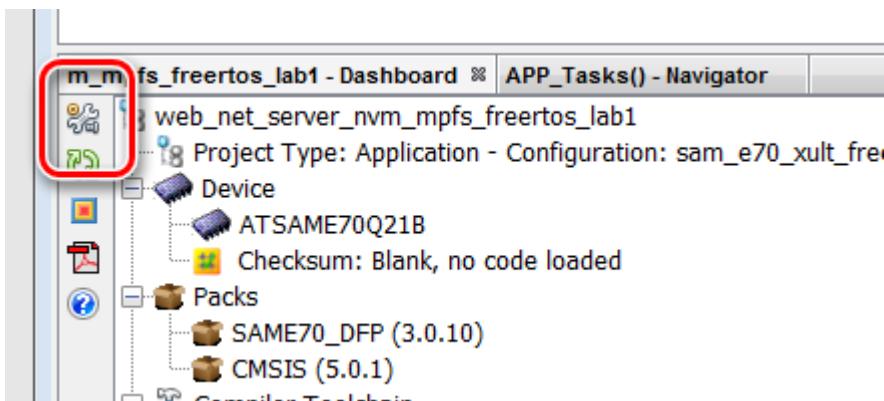


2. Open Project Properties by choosing

- a. File>Project Properties from the main menu
- b. Or select with a right click the project node in the project windows and select at the bottom



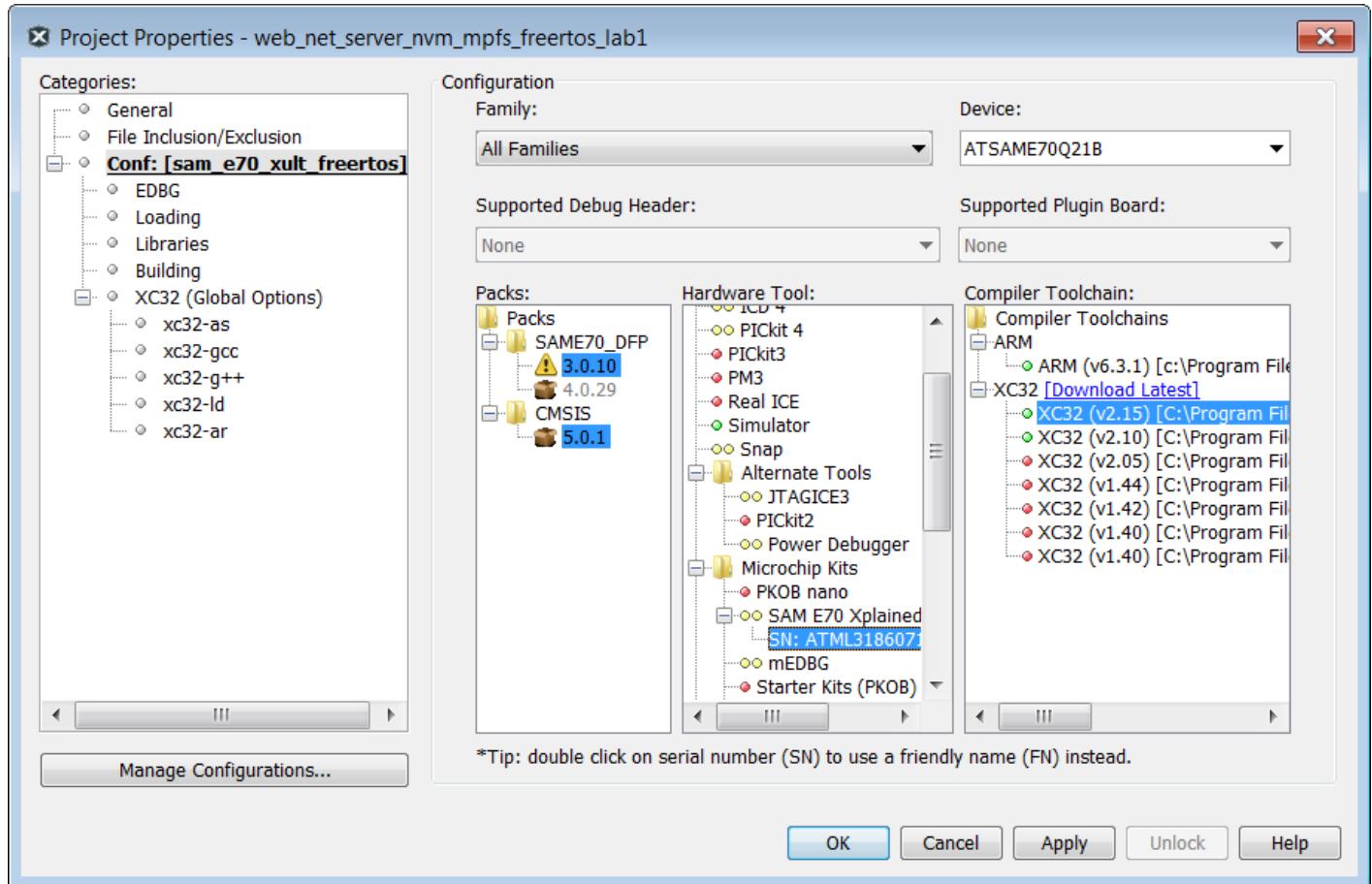
- c. Or select the toolbox in the Dashboard



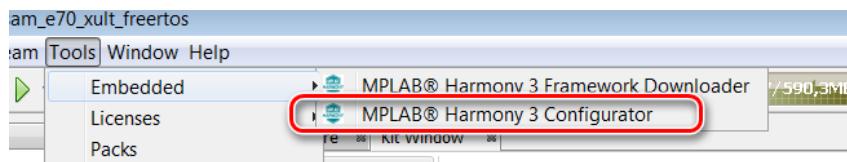
d.

3. Select the XC32 Compiler v2.15

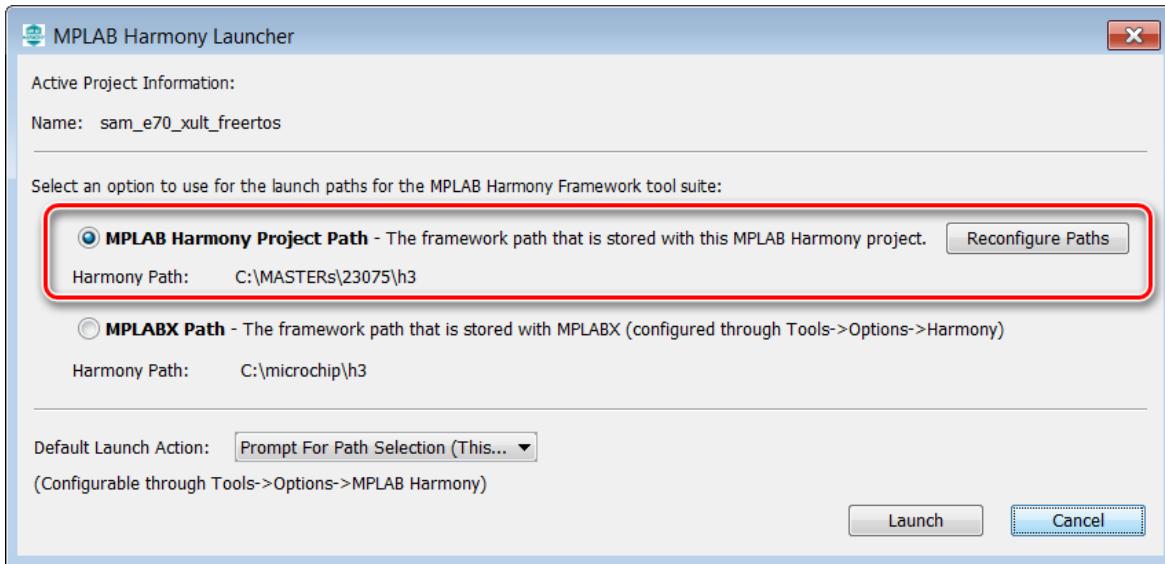
4. Select the SAME70 Xplained by clicking on the SN: Number



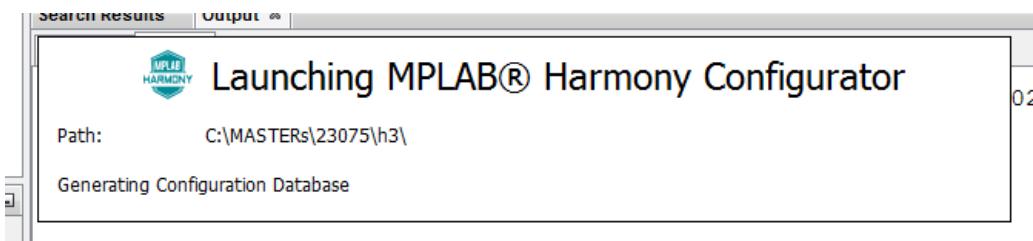
5. Select from the Tools Menu the Harmony 3 Configurator



## 6. Ensure the H3 Path is set to "C:\MASTERs\23075\h3\"

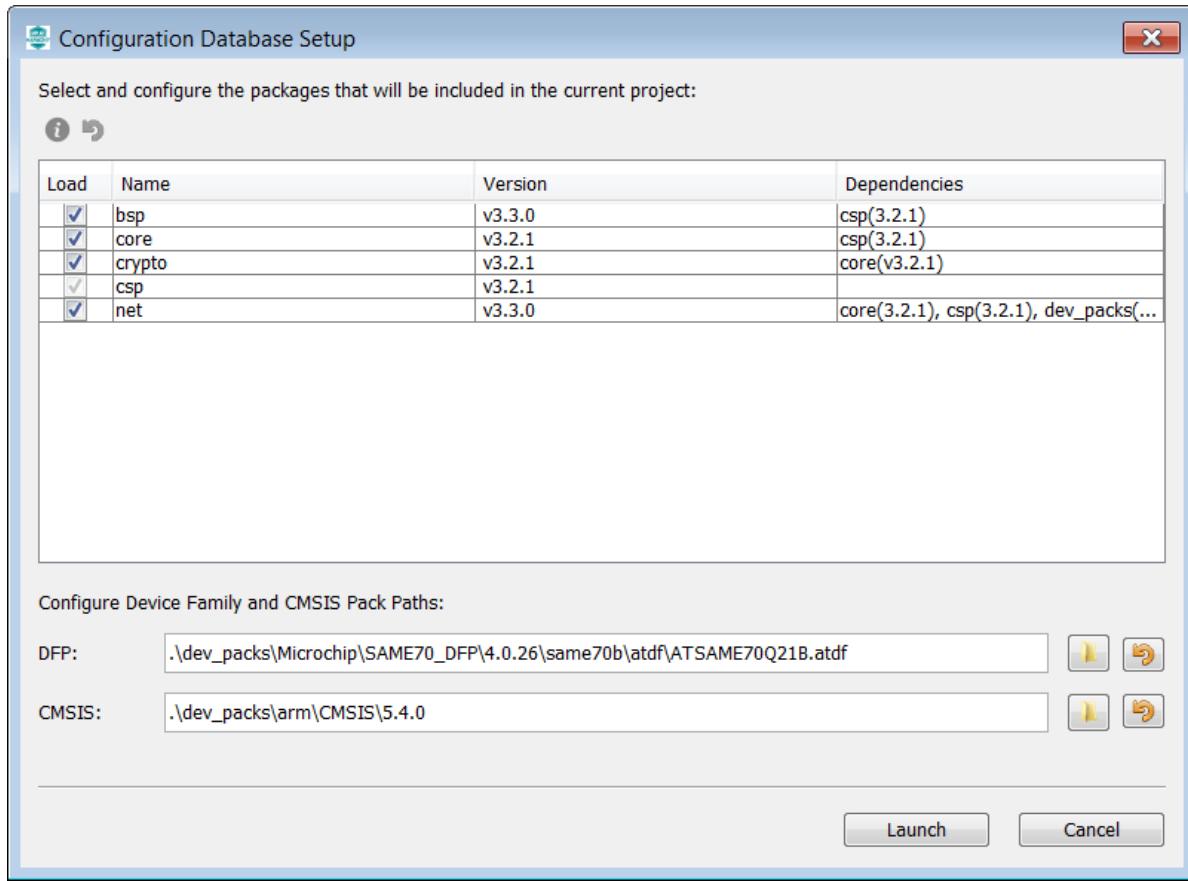


## 7. Select "Launch"



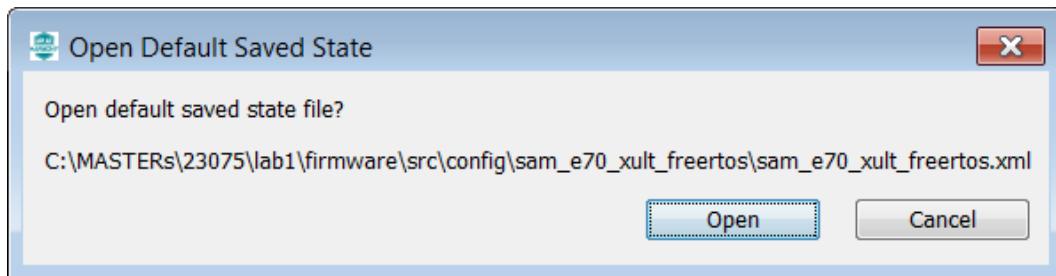
The first time the MHC is started, it can take up to 2 Minutes before the Configuration Database is prepared

8. In the next window the H3 parts and their used Version Numbers are displayed

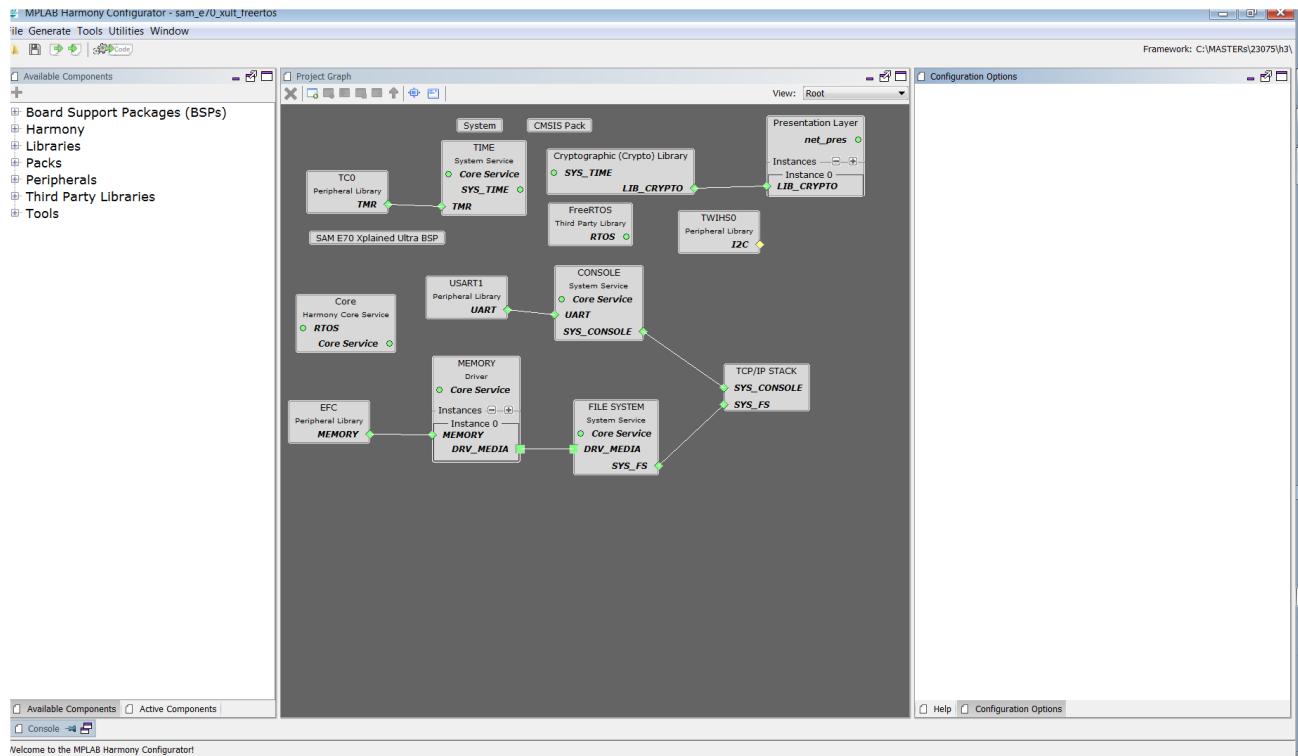


Click on "Launch"

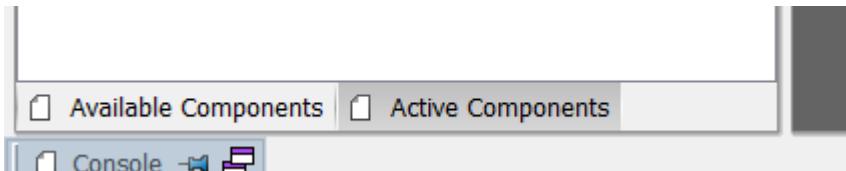
9. Open the saved state file



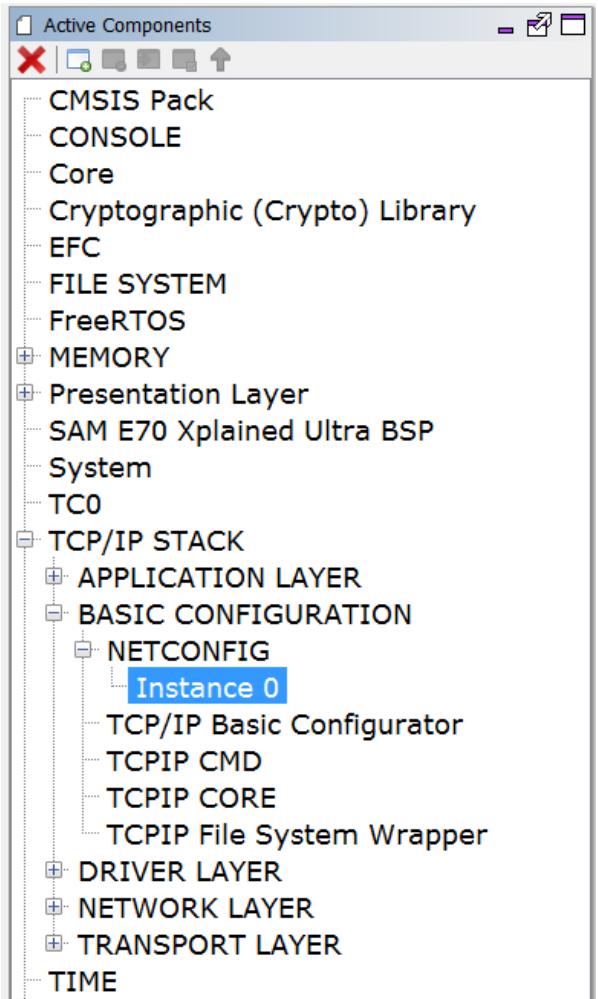
## 10. The MHC is up and running



## 11. Select Active Components (left below)

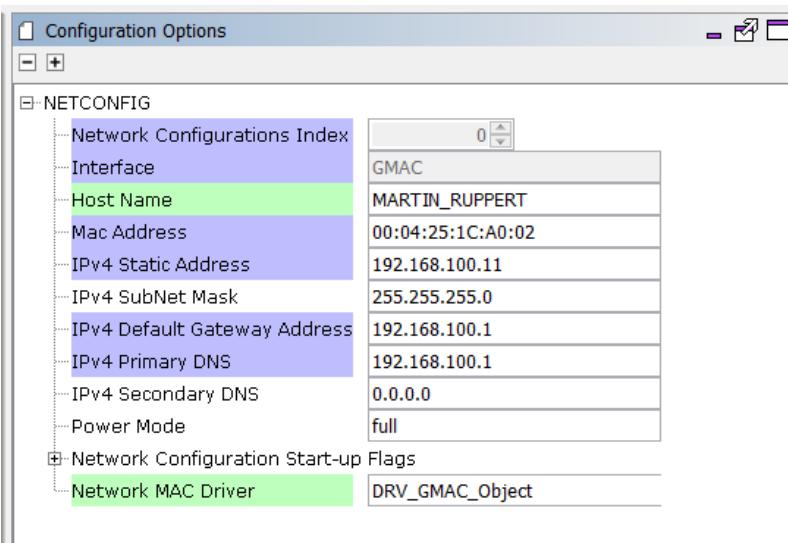


12. Select the Instance 0 in Active Components



13. And change in the Configuration Options (on the right side) the Host Name to something meaningful for you.

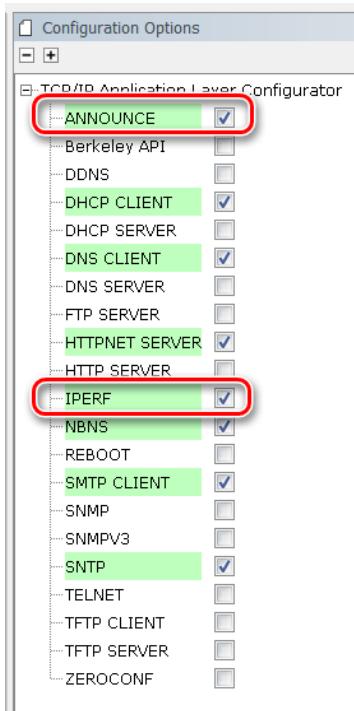
The Host name can be identified in the Network.



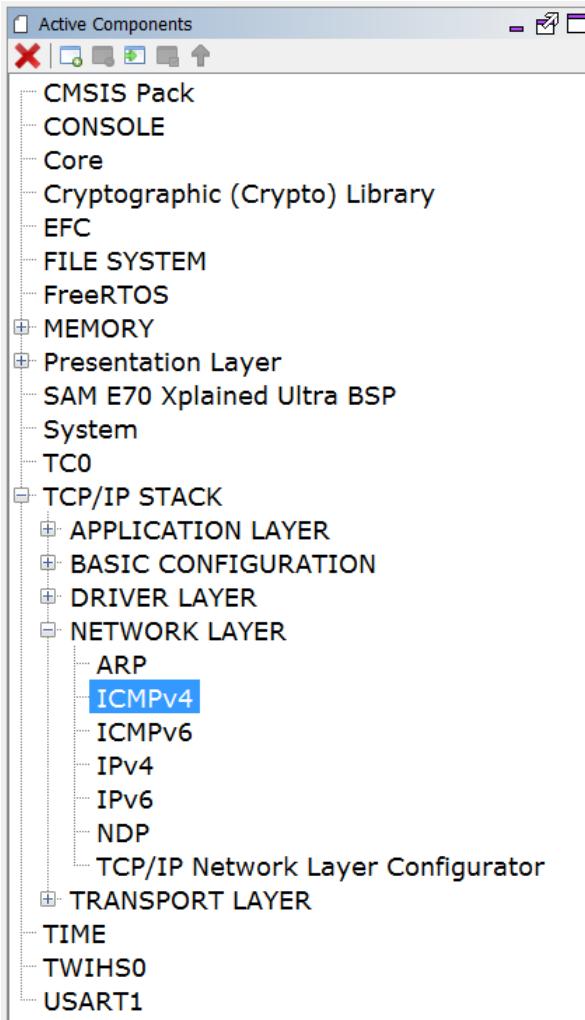
## 14. Select “TCP/IP Application Layer Configuration” in Active Components



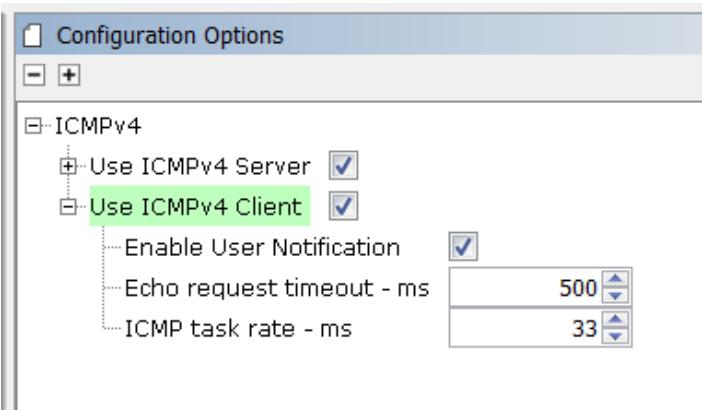
## 15. Ensure that ANNOUNCE and IPERF are selected



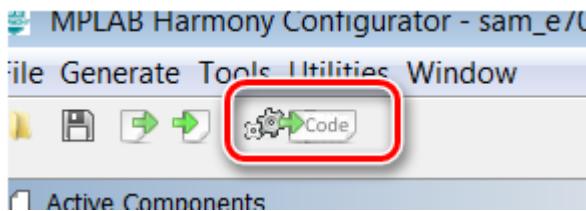
16. Select ICMPv4 in the Active Components



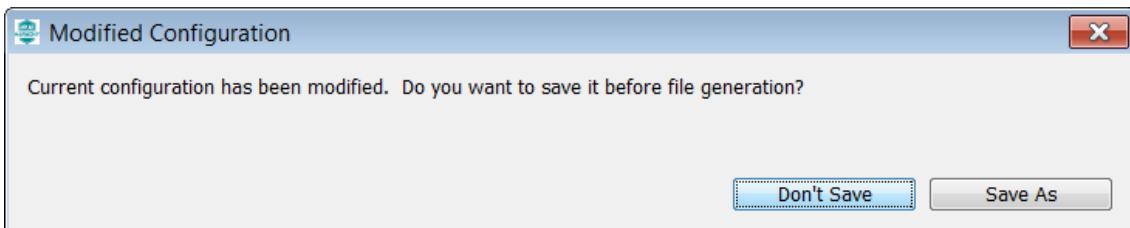
17. And ensure that the “Use ICMPv4 Client” is selected



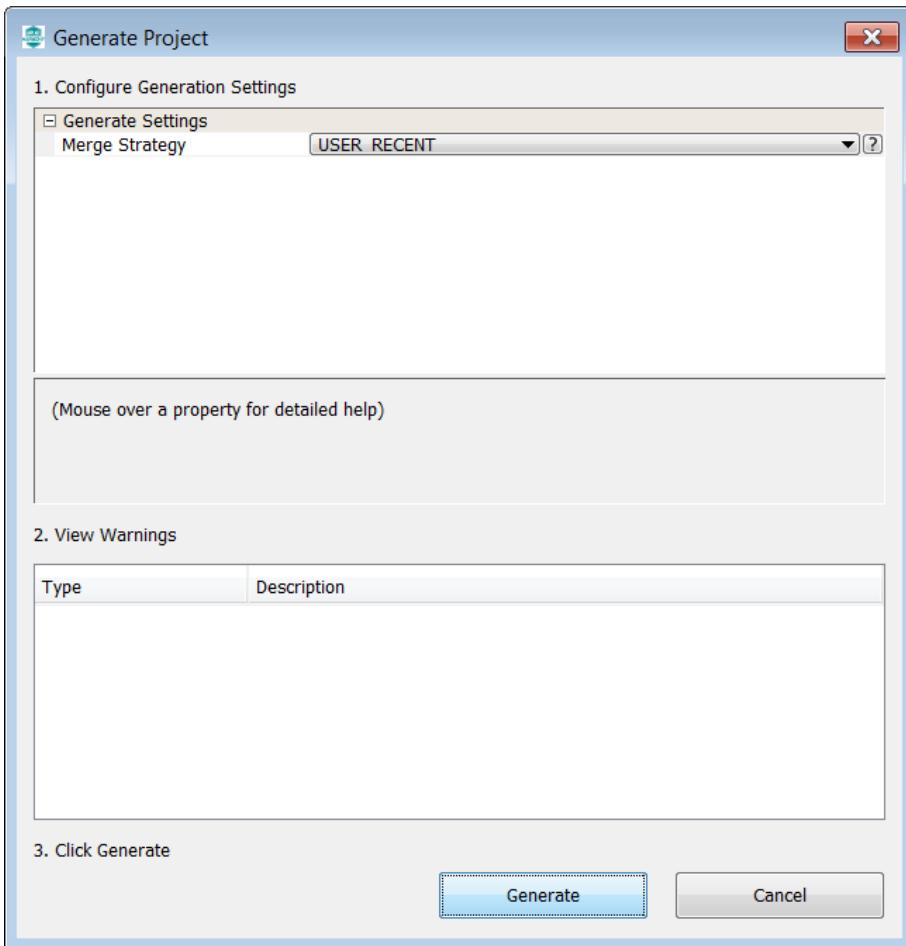
18. Select “Code” (Generate Code)



19. Select Don't Save



20. Select Generate



21. Some Files will be changed and the MHC is asking in a “diff” window, if the changes should be taken over.

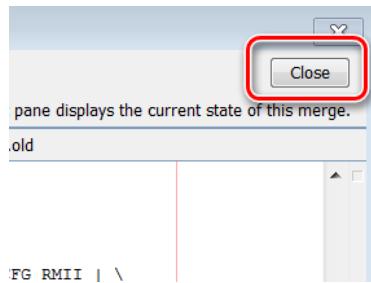
Accept all changes in the file by clicking on the Arrow in the middle above.

```

Merging: configuration.h_8107676870428926504.old
Pending Merge Actions: 2
Generated Code
1/2
32 33 #define TCPIP_IPV6_NDP_MAX_NEIGHBOR_ADVERTISEMENT 3
33 #define TCPIP_IPV6_NDP_REACHABLE_TIME 30
34 #define TCPIP_IPV6_NDP_RETRY_TIMER 1
35 #define TCPIP_IPV6_NDP_DELAY_FIRST_PROBE_TIME 5
36 #define TCPIP_IPV6_NDP_VALID_LIFETIME_TWO_HOURS (60 * 60 * 2)
37 #define TCPIP_IPV6_MTU_INCREASE_TIMEOUT 600
38 #define TCPIP_IPV6_NDP_TASK_TIMER_RATE 32
39
40
41 /* Network Configuration Index 0 */
42 #define TCPIP_NETWORK_DEFAULT_INTERFACE_NAME_IDX0 "GMAC"
43 #define TCPIP_IF_GMAC
44
45 #define TCPIP_NETWORK_DEFAULT_HOST_NAME_IDX0 "MARTIN RUPPERT"
46 #define TCPIP_NETWORK_DEFAULT_MAC_ADDR_IDX0 "00:04:25:1C:A0:02"
47
48 #define TCPIP_NETWORK_DEFAULT_IP_ADDRESS_IDX0 "192.168.100.11"
49 #define TCPIP_NETWORK_DEFAULT_IP_MASK_IDX0 "255.255.255.0"
50 #define TCPIP_NETWORK_DEFAULT_GATEWAY_IDX0 "192.168.100.1"
51 #define TCPIP_NETWORK_DEFAULT_DNS_IDX0 "192.168.100.1"
52 #define TCPIP_NETWORK_DEFAULT_SECOND_DNS_IDX0 "0.0.0.0"
53 #define TCPIP_NETWORK_DEFAULT_POWER_MODE_IDX0 "full"
54 #define TCPIP_NETWORK_DEFAULT_INTERFACE_FLAGS_IDX0
55
56 TCPIP_NETWORK_CONFIG_DHCP_CLIENT_ON \
57 TCPIP_NETWORK_CONFIG_DNS_CLIENT_ON \
58 TCPIP_NETWORK_CONFIG_IP_STATIC
59
60 #define TCPIP_NETWORK_DEFAULT_MAC_DRIVER_IDX0 DRV_GMAC_Object
61
62
63 /** tcipip_cmd Configuration ***/
64 #define TCPIP_STACK_COMMAND_ENABLE
65 #define TCPIP_STACK_COMMANDS_ICMP_ECHO_REQUESTS 4
66 #define TCPIP_STACK_COMMANDS_ICMP_ECHO_REQUEST_DELAY 1000
67 #define TCPIP_STACK_COMMANDS_ICMP_ECHO_TIMEOUT 5000
68 #define TCPIP_STACK_COMMANDS_WIFI_ENABLE false
69 #define TCPIP_STACK_COMMANDS_ICMP_ECHO_REQUEST_BUFF_SIZE 2000
70 #define TCPIP_STACK_COMMANDS_ICMP_ECHO_REQUEST_DATA_SIZE 100
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94 // private stack manager interface
95 #include "tcipip/src/tcipip_manager_control.h"
96
97 #include "tcipip/src/tcipip_announce_manager.h"
98 #include "tcipip/src/ndp_manager.h"
99 #include "tcipip/src/ipv4_manager.h"
100 #include "tcipip/src/ipv6_manager.h"
101 #include "tcipip/src/icmpv6_manager.h"
102 #include "tcipip/src/dhcpv6_manager.h"
103 #include "tcipip/src/icmp_manager.h"
104 #include "tcipip/src/dhcp_manager.h"
105 #include "tcipip/src/dns_manager.h"
106 #include "tcipip/src/tcp_manager.h"
107 #include "tcipip/src/nbns_manager.h"
108 #include "tcipip/src/http_net_manager.h"
109 #include "tcipip/src/tcipip_commands_manager.h"
110 #include "tcipip/src/udp_manager.h"
111 #include "tcipip/src/ntp_manager.h"
112 #include "tcipip/src/ldp_manager.h"
113
114 #include "tcipip/src/tpcpf_manager.h"
115
116 #include "tcipip/src/smtpc_manager.h"
117 #include "tcipip/src/tcipip_packet.h"
118 #include "tcipip/src/tcipip_helpers_private.h"
119 #include "tcipip/src/oahash.h"
120 #include "tcipip/src/hash_fnv.h"
121 #include "tcipip/src/tcipip_notify.h"
122
123 #endif // __TCPIP_STACK_PRIVATE_H__

```

22. Then click on close in the upper right corner



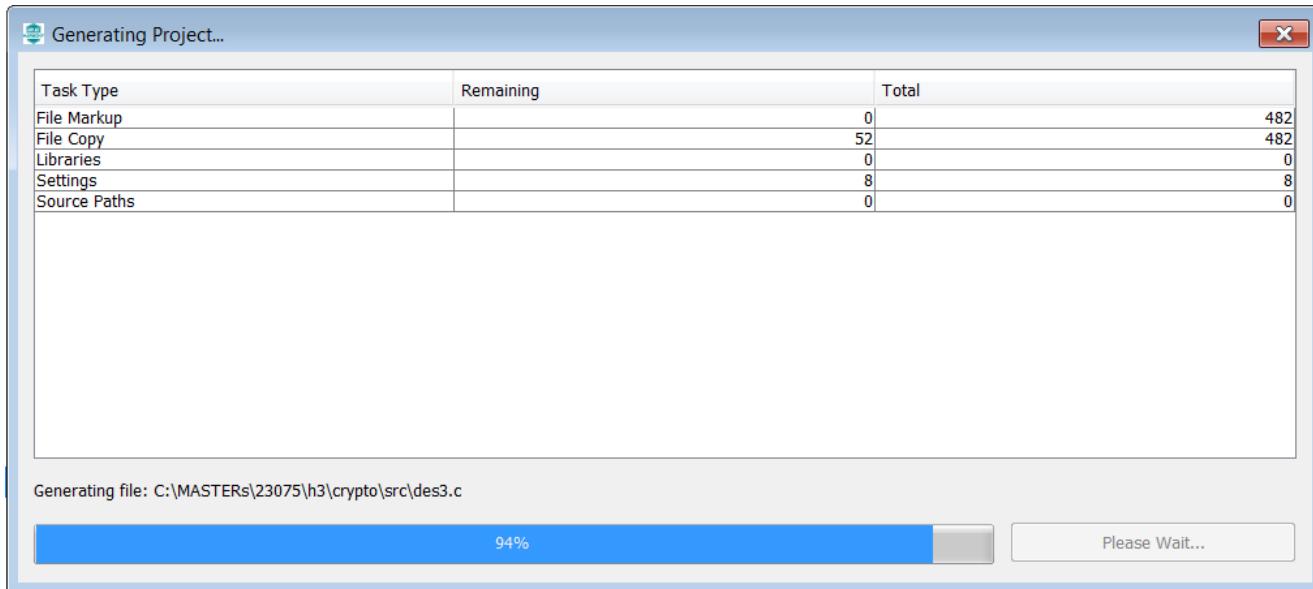
23. Same for the next 2 diff windows

```

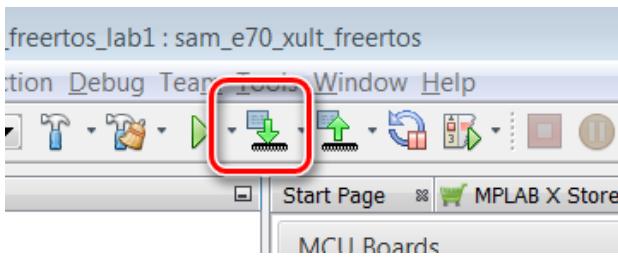
Merging: tcipip_private.h_5816637879687640463.old
Pending Merge Actions: 1
Generated Code
1/1
93 // private stack manager interface
94 #include "tcipip/src/tcipip_manager_control.h"
95
96 #include "tcipip/src/tcipip_announce_manager.h"
97 #include "tcipip/src/ndp_manager.h"
98 #include "tcipip/src/ipv4_manager.h"
99 #include "tcipip/src/ipv6_manager.h"
100 #include "tcipip/src/icmpv6_manager.h"
101 #include "tcipip/src/dhcpv6_manager.h"
102 #include "tcipip/src/icmp_manager.h"
103 #include "tcipip/src/dhcp_manager.h"
104 #include "tcipip/src/arp_manager.h"
105 #include "tcipip/src/dns_manager.h"
106 #include "tcipip/src/tcp_manager.h"
107 #include "tcipip/src/nbns_manager.h"
108 #include "tcipip/src/http_net_manager.h"
109 #include "tcipip/src/tcipip_commands_manager.h"
110 #include "tcipip/src/udp_manager.h"
111 #include "tcipip/src/ntp_manager.h"
112 #include "tcipip/src/ldp_manager.h"
113
114 #include "tcipip/src/tpcpf_manager.h"
115
116 #include "tcipip/src/smtpc_manager.h"
117 #include "tcipip/src/tcipip_packet.h"
118 #include "tcipip/src/tcipip_helpers_private.h"
119 #include "tcipip/src/oahash.h"
120 #include "tcipip/src/hash_fnv.h"
121 #include "tcipip/src/tcipip_notify.h"
122
123 #endif // __TCPIP_STACK_PRIVATE_H__

```

24. The whole process is displayed with a progress bar



25. Back again in the main window of MPLABX, click on the “Make and Program Device” button



26. After successful build, the SAME70 is programmed automatically

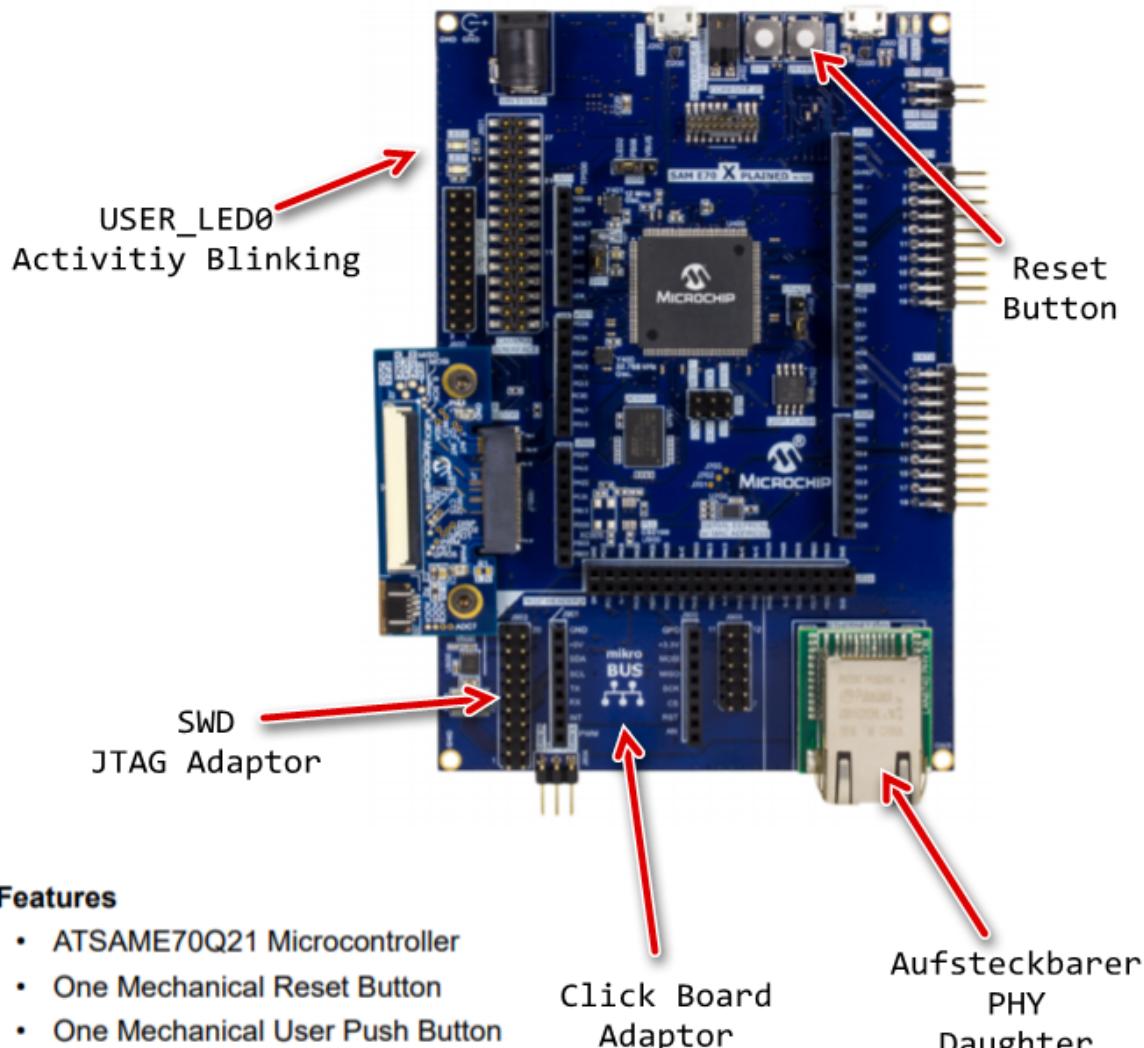
```

Search Results Output ✎
EDBG ✎ Kits ✎ web_net_server_nvm_mpfs_freertos_lab1 (Build, Load, ...) ✎
make[1]: Leaving directory 'C:/MASTERs/23075/lab1/firmware/sam_e70_xult_
action/sam_e70_xult_freertos.X.production.hex...
Loading completed
Connecting to programmer...

```

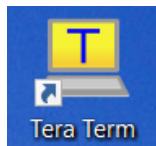
27. The programming take about 30 seconds with on Board Debugger Check whether the USER\_LED0 Activity is Blinking

**Figure 1. SAM E70 Xplained Ultra Evaluation Kit with PHY Daughter Board**



#### Features

- ATSAME70Q21 Microcontroller
- One Mechanical Reset Button
- One Mechanical User Push Button
- Two User LEDs
- 12.0 MHz Oscillator (DSC6003)
- 32.768 kHz Oscillator (DSC6083)
- 2-MB SDRAM
- 4-MB QSPI Flash (SST26VF032BA)



28. Open Terra Term Terminal Program [Tera Term](#) and select under Setup->Serial Port the COM Port and the Baudrate 115200 (in this case a COM96, but could be a different COM Port number in your case).

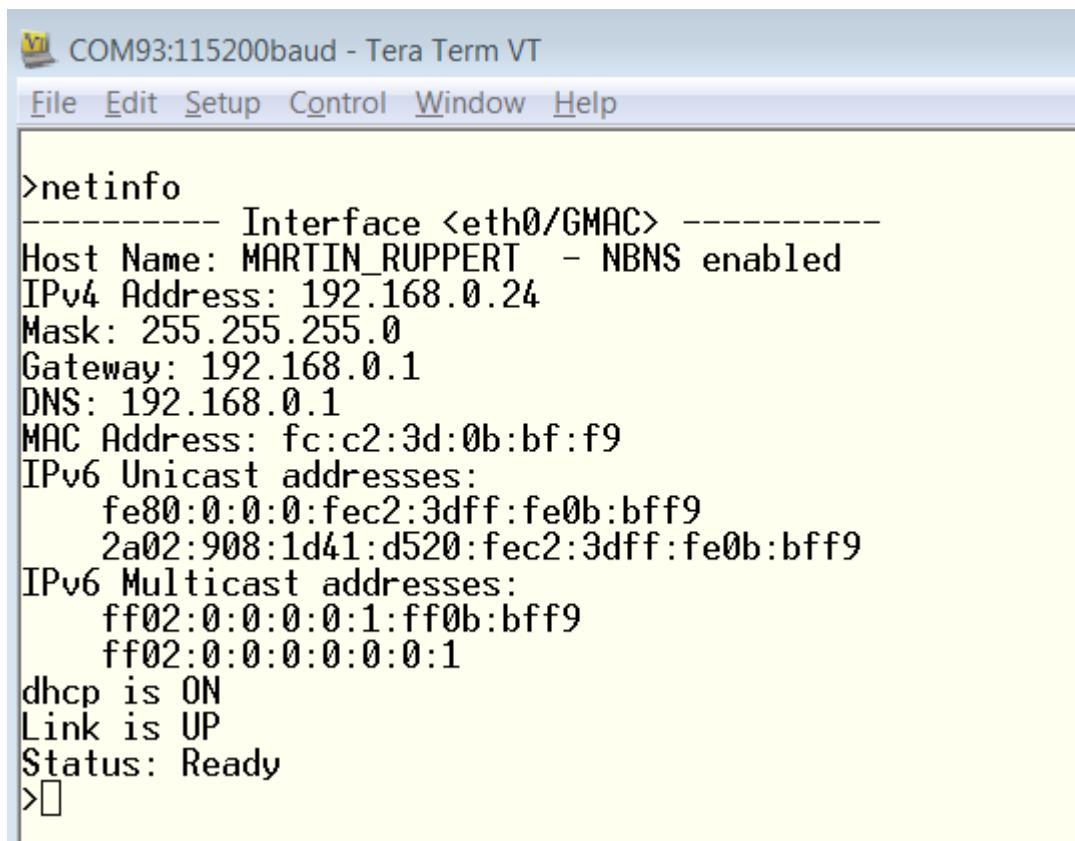
The press the Reset Button of the Board and let the Firmware restart. You should see the start message with the Lab name and the build timestamp at first, followed by the MAC Address from EEPROM.

29. After some small time, the IP Address should be change from 0.0.0.0 to a valid address

A screenshot of the Tera Term terminal window titled "COM96:115200baud - Tera Term VT". The window has a menu bar with File, Edit, Setup, Control, Window, and Help. The main pane displays the following text:

```
>
=====
web_net_server_nvmm_mpfs_freertos_lab1 Jun 3 2019 19:30:07
SYS_Initialize: The MPFS2 File System is mounted
MAC TCPPIP_HOSTS_CONFIGURATION[0].macAddr: fc:c2:3d:0c:20:44
TCP/IP Stack: Initialization Started
TCP/IP Stack: Initialization Ended - success
    Interface GMAC on host MARTIN_RUPPERT - NBNS enabled
GMAC IP Address: 0.0.0.0
GMAC IP Address: 192.168.0.17
```

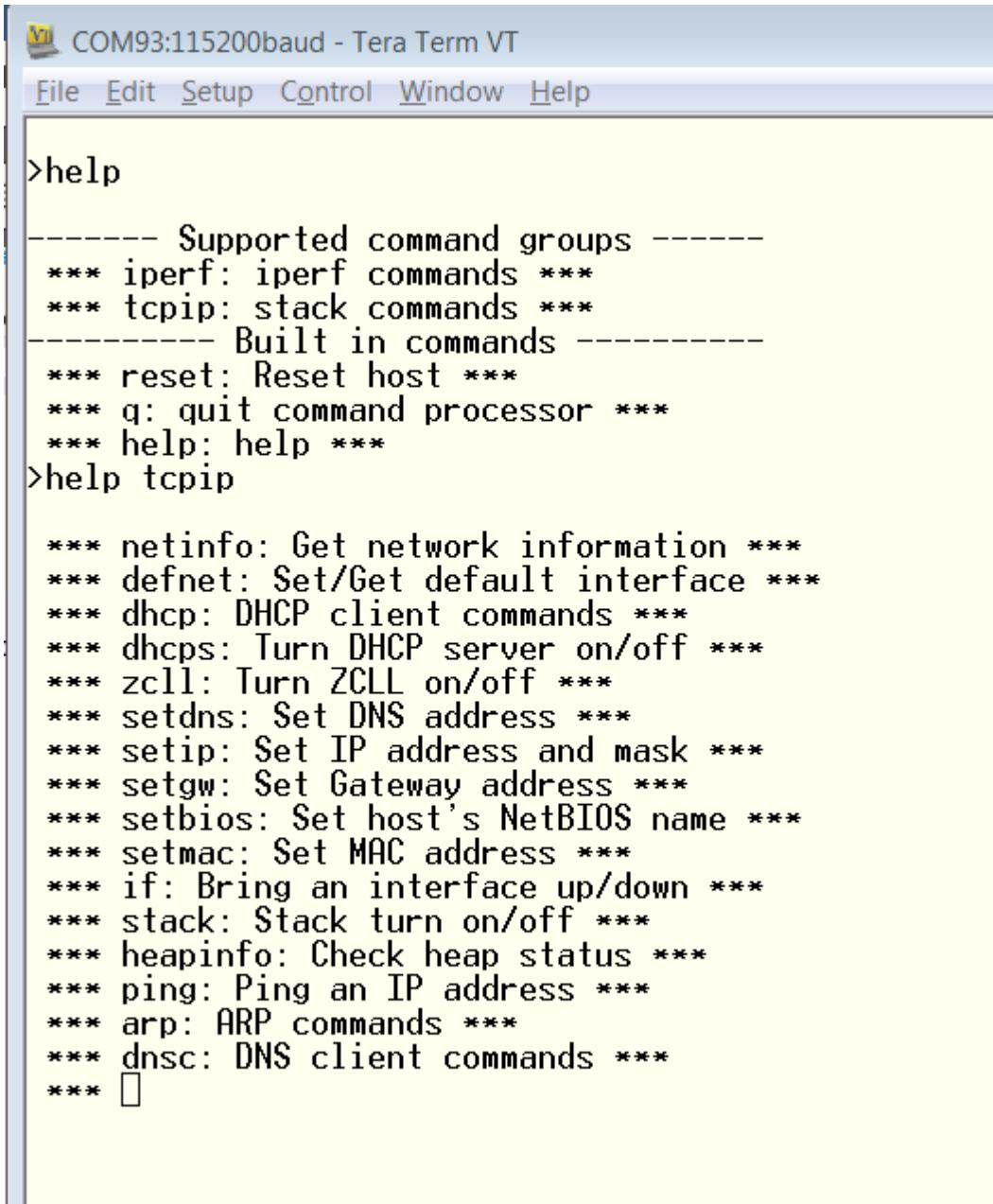
30. To check the basic information about the network enter the netinfo command and press **Enter**.



The screenshot shows a terminal window titled "COM93:115200baud - Tera Term VT". The menu bar includes File, Edit, Setup, Control, Window, and Help. The terminal output displays the results of the "netinfo" command:

```
>netinfo
----- Interface <eth0/GMAC> -----
Host Name: MARTIN_RUPPERT - NBNS enabled
IPv4 Address: 192.168.0.24
Mask: 255.255.255.0
Gateway: 192.168.0.1
DNS: 192.168.0.1
MAC Address: fc:c2:3d:0b:bf:f9
IPv6 Unicast addresses:
  fe80:0:0:0:fc2:3dff:fe0b:bf9
  2a02:908:1d41:d520:fc2:3dff:fe0b:bf9
IPv6 Multicast addresses:
  ff02:0:0:0:0:1:ff0b:bf9
  ff02:0:0:0:0:0:0:1
dhcp is ON
Link is UP
Status: Ready
>
```

31. A help shows the available commands

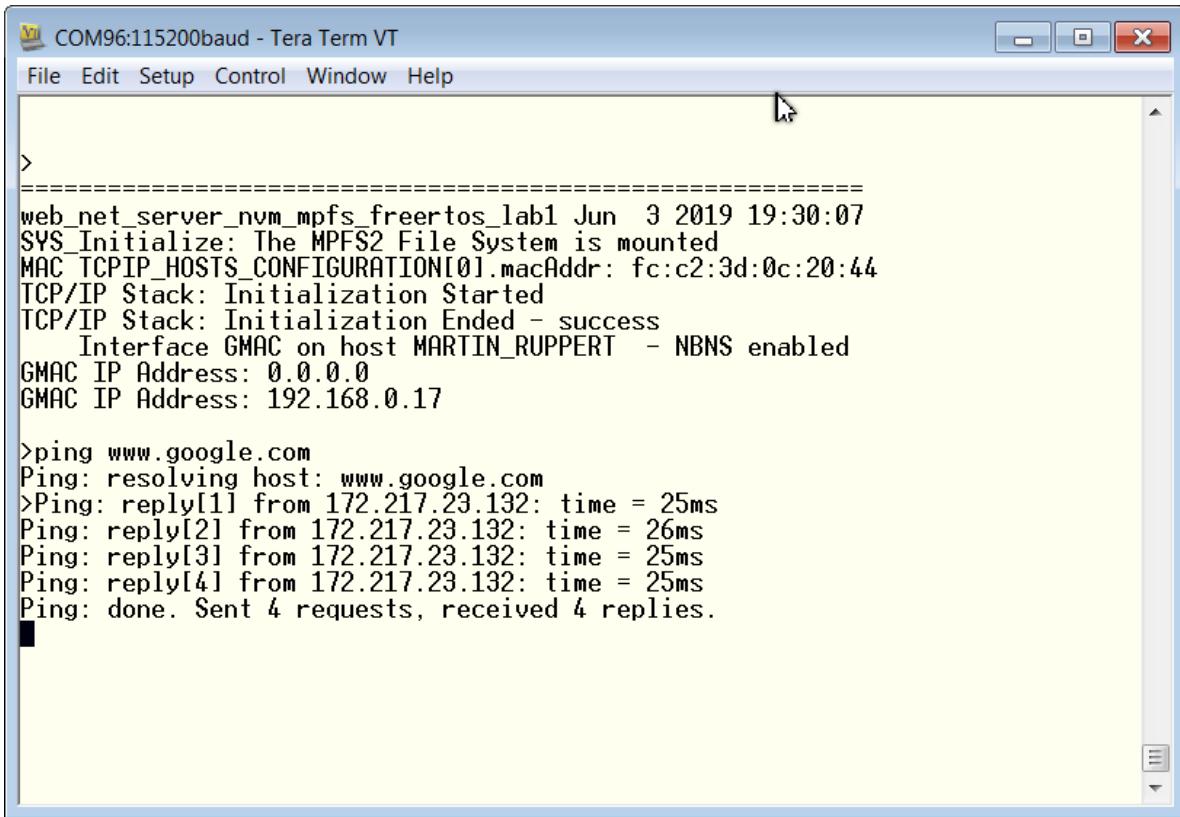


The screenshot shows a terminal window titled "COM93:115200baud - Tera Term VT". The menu bar includes File, Edit, Setup, Control, Window, and Help. The terminal window displays command help output:

```
>help
----- Supported command groups -----
*** iperf: iperf commands ***
*** tcpip: stack commands ***
----- Built in commands -----
*** reset: Reset host ***
*** q: quit command processor ***
*** help: help ***
>help tcpip

*** netinfo: Get network information ***
*** defnet: Set/Get default interface ***
*** dhcp: DHCP client commands ***
*** dhcps: Turn DHCP server on/off ***
*** zcll: Turn ZCLL on/off ***
*** setdns: Set DNS address ***
*** setip: Set IP address and mask ***
*** setgw: Set Gateway address ***
*** setbios: Set host's NetBIOS name ***
*** setmac: Set MAC address ***
*** if: Bring an interface up/down ***
*** stack: Stack turn on/off ***
*** heapinfo: Check heap status ***
*** ping: Ping an IP address ***
*** arp: ARP commands ***
*** dnsc: DNS client commands ***
*** □
```

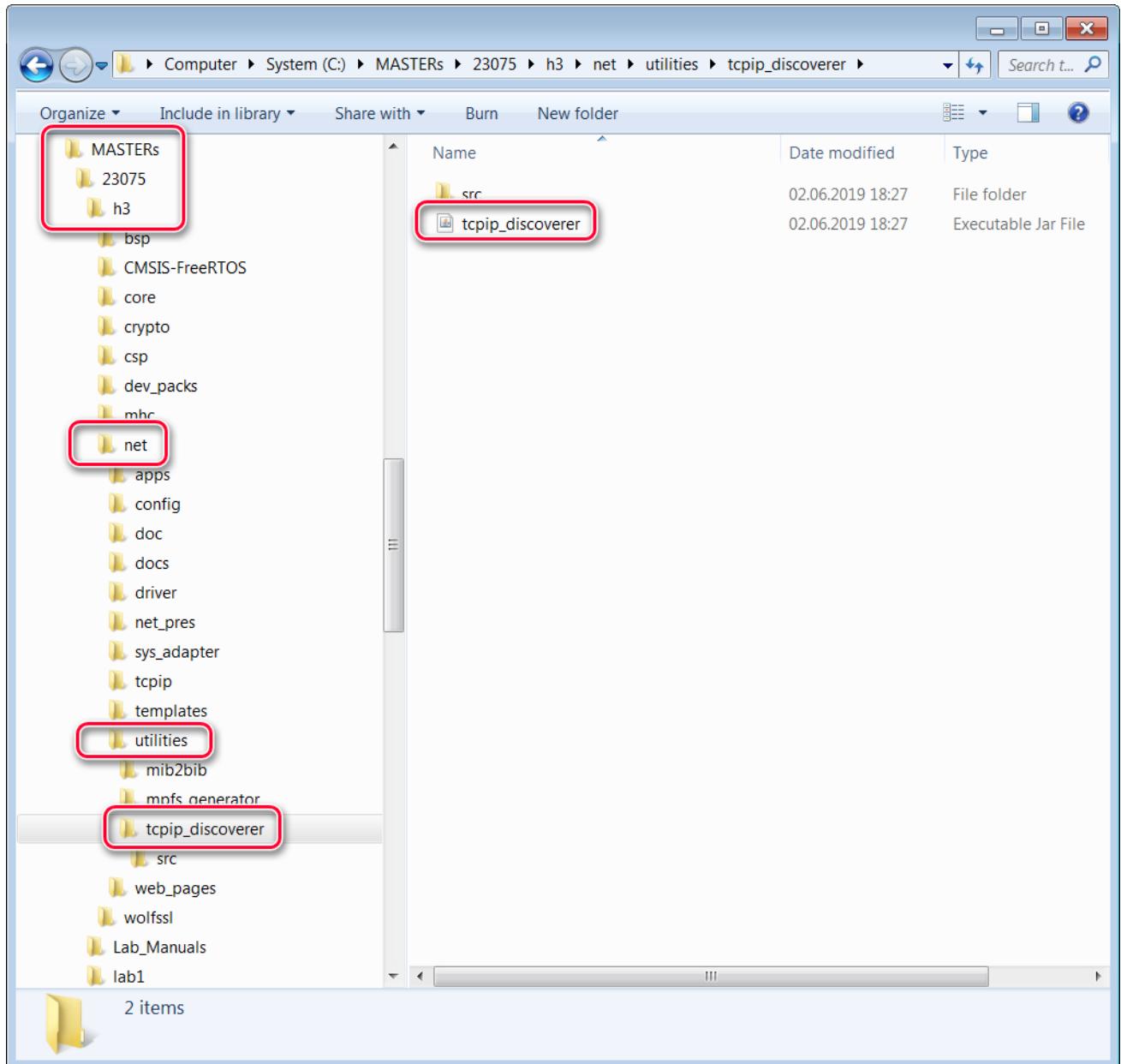
32. As a first simple test you can ping an external Server or anything else you like (Maybe the board of your class neighbor?)



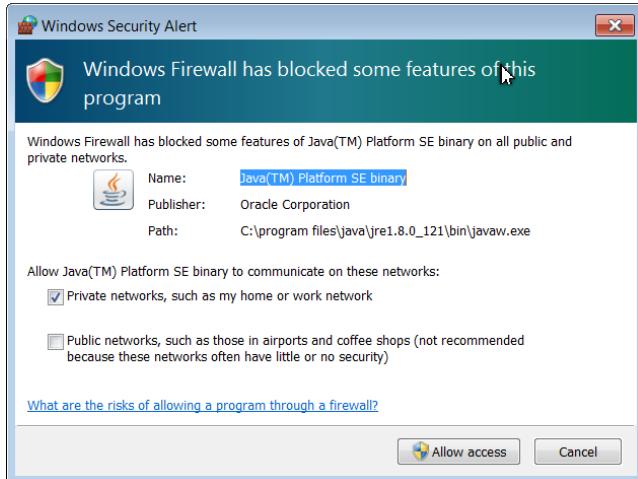
```
COM96:115200baud - Tera Term VT
File Edit Setup Control Window Help

>
=====
web_net_server_nvm_mpfs_freertos_lab1 Jun  3 2019 19:30:07
SYS_Initialize: The MPFS2 File System is mounted
MAC TCPIP_HOSTS_CONFIGURATION[0].macAddr: fc:c2:3d:0c:20:44
TCP/IP Stack: Initialization Started
TCP/IP Stack: Initialization Ended - success
  Interface GMAC on host MARTIN_RUPPERT - NBNS enabled
GMAC IP Address: 0.0.0.0
GMAC IP Address: 192.168.0.17

>ping www.google.com
Ping: resolving host: www.google.com
>Ping: reply[1] from 172.217.23.132: time = 25ms
Ping: reply[2] from 172.217.23.132: time = 26ms
Ping: reply[3] from 172.217.23.132: time = 25ms
Ping: reply[4] from 172.217.23.132: time = 25ms
Ping: done. Sent 4 requests, received 4 replies.
```

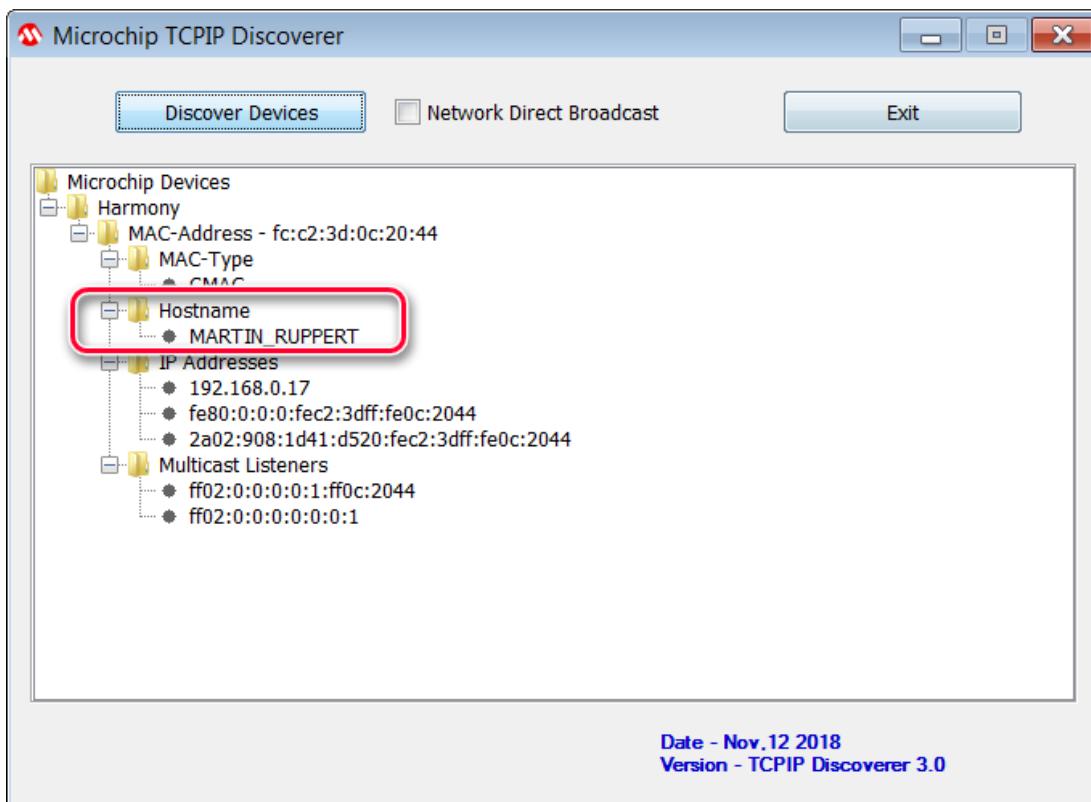
33. Select the **tcpip\_discoverer** tool from

34. If Windows is asking for permissions allow the access



35. The TCP Discover should list all boards in the classroom. We have made this tool to help you to find your board in the network. The source codes of this tool (Java) are part of the H3.

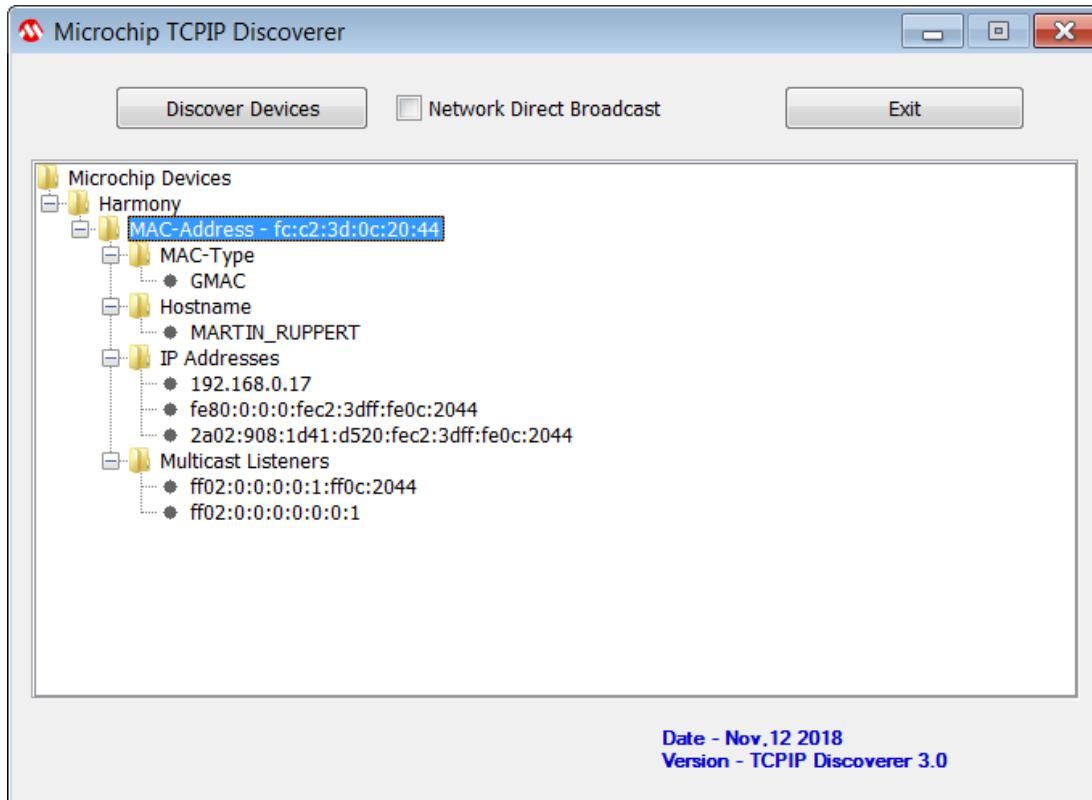
You can identify your board by the Host Name that has select in an earlier step. The Host Name is also known to the DHCP server and is listed in their typical Web Interfaces as a connected device.



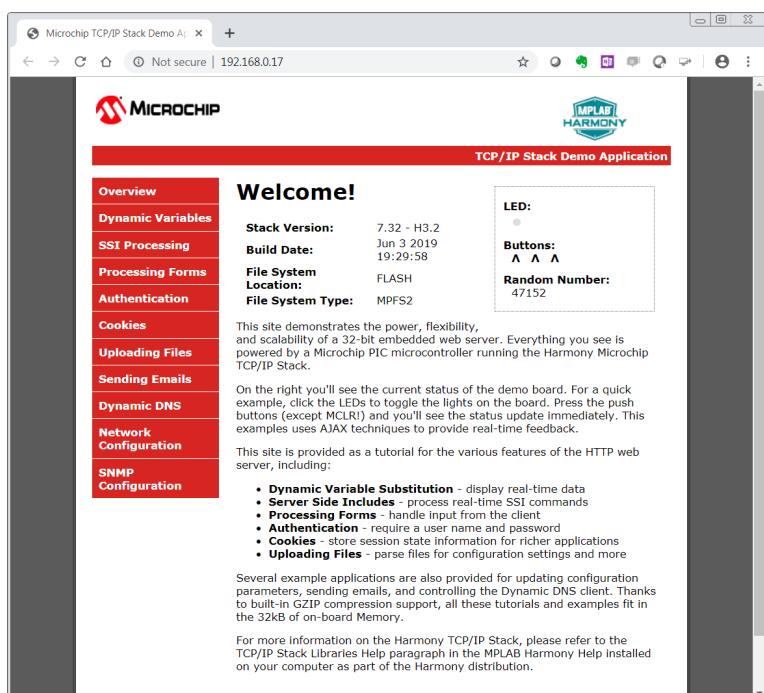
Press the **Discover Devices** button: The tool will send a UDP broadcast on port 30303, with the packet "Discovery, who is out there?" All H3 devices running the Announce service will respond to this broadcast, by sending a return broadcast on port 30303. The broadcast packet contains data on the type 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 MHC Setup process. The Microchip TCPIP Discoverer tool also shows the IP address for your board.

36. A double click on the **MAC-Address** line will put you in your default Internet Browser



37. And the Webpage is displayed. Please take some time an play with the sub menus to find out the capabilities of or H3 Web Server



37. Congratulations, you have completed Lab 1!

## Lab 2

### Overview

## Lab 3

### Overview

In many IoT applications, JSON is commonly used as a format in order to transport high-level data in an effective way. It is generally an alternative to XML. Consider the following example of describing a person named Raji-Niklas Ruppert in JSON-format:

```
{  
    "firstName" : "Raji-Niklas",  
    "lastName" : "Ruppert",  
    "age" : 30,  
    "address" : {  
        "streetAddress" : "2355 W Chandler Blvd",  
        "city" : "Chandler",  
        "state" : "AZ",  
        "postalCode" : "85224",  
    }  
}
```

Using this format makes it very easy to communicate between applications requiring information about Raji-Niklas Ruppert.

The advantage of using JSON in embedded applications is that because it is easy to read for humans, it is simple to parse and make use of. Due to this, it is commonly used to transmit data between a server and a web application. In this lab we are going to implement an embedded application fetching weather data from a web server. When the application accesses a specific URL specifying a command with a geographic location, the web server will respond by sending the current weather in JSON-format to the web application. The application will be running on our SAME70-boards.

In this lab we will only do very simple parsing (which is one of the strengths using JSON), using standard string operations. There are however more sophisticated parsers which can be used for more robust and complex applications, while still only consuming a very limited footprint.

The weather service used in this lab is <https://openweathermap.org/>. With OpenWeatherMap, there are several services such as hourly forecast, UV Index, Air pollution and more, all outputting in JSON. With the free account there are limited option to only use the "Current Weather Data" service. With this service you can request the current weather from different geographic locations. Depending on by which method (City ID, ZIP Code, Coordinates etc.) the URL call will be slightly different. A full description of the API can be found here: <https://openweathermap.org/current>. For this lab we will fetch current weather by city. The following URL for this is:

<http://api.openweathermap.org/data/2.5/weather?q={CITY}&APPID={API Key}>

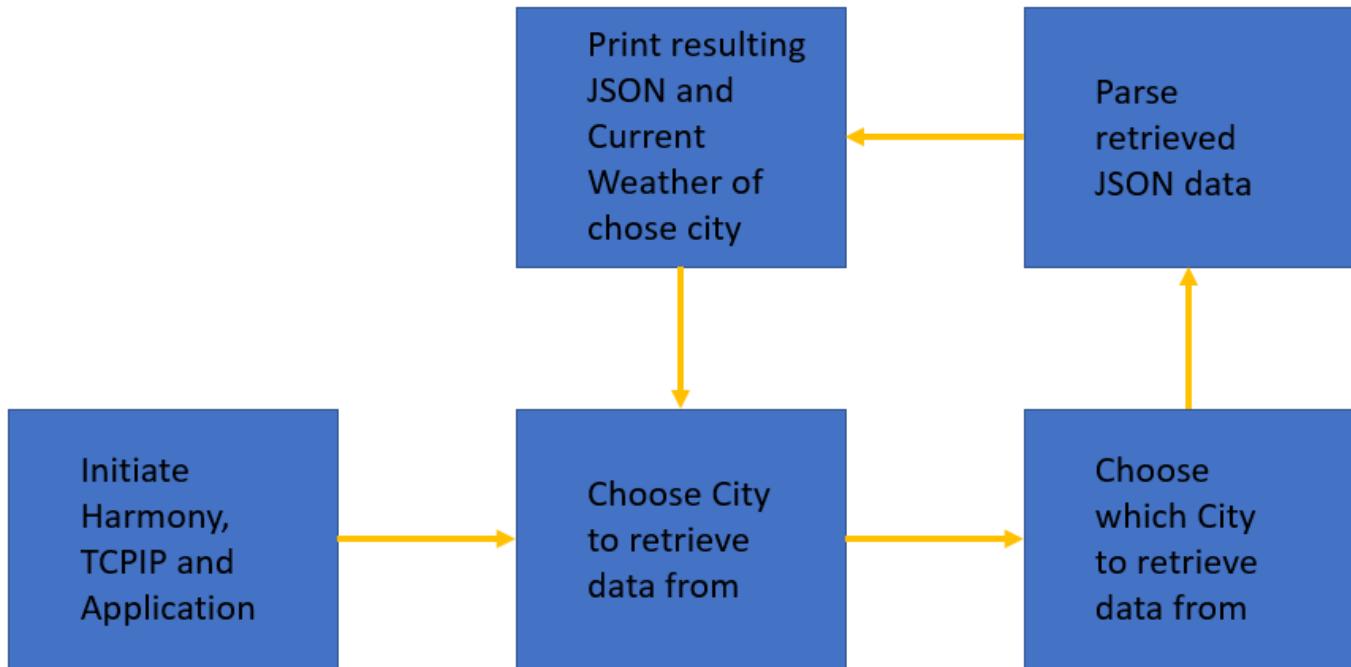
The API Key is unique to each user. This is also how OpenWeatherMap tracks how many requests you attempt. The API Key is a 15-byte long hexadecimal string. It can look like this:

```
ed3da58111974261002c2af4f8e8e81f
```

In most JSON API:s there is also a well defined format specified, which tells you where the different objects and strings are located in the JSON-message. From OpenWeatherMap:

```
{"coord":{"lon":-122.09,"lat":37.39},  
"sys":{"type":3,"id":168940,"message":0.0297,"country":"US","sunrise":1427723751,"sunset":14  
27768967},  
"weather":[{"id":800,"main":"Clear","description":"Sky is Clear","icon":"01n"}],  
"base":"stations",  
"main":{"temp":285.68,"humidity":74,"pressure":1016.8,"temp_min":284.82,"temp_max":286.48},  
"wind":{"speed":0.96,"deg":285.001},  
"clouds":{"all":0},  
"dt":1427700245,  
"id":0,  
"name":"Mountain View",  
"cod":200}
```

## Application Flow



## Lab Outline

- In the source code file app.c are TODO's. At these places you have to change the source codes. At the bottom of the app.c you find the solutions. Either you think about what you have to change, or you just copy the solution at the right place.
- The main purpose of this class is to point you to the crucial points in an Harmony 3 application and not to let you write a complete TCP application
- The pre-made template is built from the Harmony example project, tcpip\_tcp\_client.
- First, we will need to declare the APPID\_KEY.
- We will then set the host & port of the remote connection static as we will only connect to OpenWeatherMap.
- After this, we will redirect the user input from the command console to a char\* buffer to be used in the application.
- Now we have all information required to build the URL from the introduction.
- When we have connected and requested the data, we need to parse the resulting JSON-string (the whole JSON containing the current weather will be in one string).
- Typically, good practice when you debug JSON-strings is to print the resulting string for you to view with your own eyes that it looks correct.
- At last, redirect the application to go back to accepting user input.

## Lab Procedure

1. Start by closing any open projects in MPLAB X IDE.
2. Open a new project and choose lab3 -> Firmware -> sam\_e70\_xult\_freertos.X.
3. Open the file app.c located under source files.
4. Go to (CTRL+F) “TODO A”. Enter the correct APPID\_KEY. Either you create your own account on OpenWeatherMap or you take the one written in the class.

```

37   // ****
38   // TODO A: Enter the correct APPID_KEY
39   static const char* APPID_KEY = "";
40   char jsonBuffer[1024];
41   char cityBuffer[128];
42
43   // ****

```

5. Now scroll down to “TODO B”, the function APP\_Initialize.
6. Set the application to connect to the host api.openweathermap.org and the port to 80. This is set to 80 because this call will be over HTTP.

```

105
106     memset(jsonBuffer, 0, sizeof (jsonBuffer));
107     memset(cityBuffer, 0, sizeof (cityBuffer));
108     //TODO B: Set the application to connect to api.openweathermap.org and port 80
109     appData.host = "";
110     appData.port =;
111
112 }
113

```

7. Re-direct the user input from APP\_URL\_BUFFER to the cityBuffer array. This can be done in several ways, but one is to use the built-in C function `sprintf(char* dest, size_t size, const char *format, ...)`. The first argument is the destination buffer (cityBuffer), the second one is the max size to be copied (128, because that is specified in the declaration) and the formatted input in this scenario is APP\_URL\_BUFFER. This can be found in “TODO C”.

```

199
200     //TODO C: Re-direct the user input to cityBuffer from APP_URL_BUFFER
201
202     sprintf(,,);
203     SYS_CONSOLE_PRINT("cityBuffer: %s\r\n", cityBuffer);
204

```

8. Scroll down to “TODO D”, the state APP\_TCPIP\_WAIT\_FOR\_CONNECTION. In this state we will wait for a connection to be established. Once established we will send a GET command with the full URL in the format specified in the introduction: `http://api.openweathermap.org/data/2.5/weather?q={CITY}&APPID={API Key}`.

```

277
278     //TODO D: Build the full URL in pathBuffer.
279     char pathBuffer[128];
280     sprintf(, 128, "data/2.5/weather?q=%s&APPID=%s", , );
281     appData.path = pathBuffer;

```

9. Once the request is sent to the server, the application will go into the APP\_TCPIP\_WAIT\_FOR\_RESPONSE state. Once the connection is closed, set the next state to be APP\_STATE\_JSON\_PARSE\_RETRIEVED\_DATA.

```

297
298     if (!TCPIP_TCP_IsConnected(appData.socket)) {
299         SYS_CONSOLE_MESSAGE("\r\nConnection Closed\r\n");
300         //TODO E: Set the next state to be APP_STATE_JSON_PARSE_RETRIEVED_DATA
301         appData.state = ;
302         break;

```

10. Now go down in the state APP\_STATE\_JSON\_PARSE\_RETRIEVED\_DATA. One of the first things we want to do after we have sorted out the JSON-part of the retrieved data is to print the raw JSON-string. This helps us debug & analyse.

```

313     char* resultingJson;
314     char* pos;
315
316     pos = strstr(jsonBuffer, "{\"");
317     *(&resultingJson) = pos;
318
319     //TODO F: Print the resultingJson string
320     SYS_CONSOLE_PRINT("resultingJson: \r\n%s \r\n", );

```

11. In a real application, we would need to first know the format of the JSON message in order to be able to parse it correctly. To make this lab more efficiently, we will do this backwards. If you look at this example piece of API response from OpenWeatherMap found in the introduction section to this lab. Looking at the format from the API, we need to calculate in what position the value of humidity start. The function strstr will cut the resulting Json string at the first occurrence of "humidity". A hint is to look at the other blocks where you parse the temperature, pressure and main weather.

```

322     //Find Humidity
323     char* mainHumidityJson;
324     char* mainHumidityBuffer;
325
326     //TODO G: Find the correct number of positions to move to the right after humidity
327     pos = strstr(resultingJson, "humidity");
328     *(&mainHumidityJson) = pos + ;
329     mainHumidityBuffer = strtok(mainHumidityJson, ",");
330

```

12. Once the parsing is done, we wish to print the values of the main weather, pressure, temperature and humidity.

```

335     mainMainWeatherBuffer = strtok(mainMainWeatherJson, "\n");
336
337     SYS_CONSOLE_PRINT("\r\nCurrent Weather in %s \r\nHumidity: %s\r\nPressure: %s\r\nTemperature: %2.2f\r\nMain Weather: %s \r\n\r\n",
338     );
339
340

```

13. Now to complete the loop, we want to go back to the APP\_TCPIP\_WAITING\_FOR\_COMMAND state once the JSON-parsing and printing is done.

```

> 359
360
361
362     //TODO I: Go back to the APP_TCPIP_WAITING_FOR_COMMAND state to continue application operation

```

14. When the build process was correct, then program the E70
15. Open the Terminal program, press the Reset Button
16. When ">" is prompter you can put in "requestWeather <city>"  
Or the short form: "rw <city>"  
For ex. "rw Phoenix"

```
=====
request weather lab3 Jun  6 2019 20:15:59
MAC TCPIP_HOSTS_CONFIGURATION[0].macAddr: fc:c2:3d:0c:20:44
TCP/IP Stack: Initialization Started
TCP/IP Stack: Initialization Ended - success
    Interface GMAC on host MCHPBOARD_C      - NBNS disabled
GMAC IP Address: 192.168.0.17
Waiting for command type: requestWeather <city>

[>rw Phoenix
cityBuffer: Phoenix
Starting connection

Connection Closed
resultingJson:
 {"coord":{"lon":-112.08,"lat":33.45},"weather":[{"id":802,"main":"Clouds","description":"scattered clo
Current Weather in Phoenix
Humidity: 15
Pressure: 1013
Temperature: 33.60
Main Weather: Clouds
```

17. At this point you reached the end of the class and can walk outside to check if the current weather report was correct

## 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.
---------------------	---

## 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.

## Harmony TCP/IP API Subset For all Lab's

### 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 hTCP,
    uint8_t* buffer,
    uint16_t len
);
```

#### Preconditions

TCP is initialized.

#### Parameters

Parameter	Description
<b>hTCP</b>	The socket from which data is to be read.
<b>buffer</b>	Pointer to the array to store data that was read.
<b>len</b>	Number of bytes to be read.

#### Returns

Type	Description
<b>uint16_t</b>	The number of bytes read from the socket. If less than <b>len</b> , 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 Prototype

```
TCP_SOCKET TCPIP_TCP_ClientOpen(
    IP_ADDRESS_TYPE addType,
    TCP_PORT remotePort,
    IP_MULTI_ADDRESS* remoteAddress
);
```

#### Preconditions

TCP is initialized.

## Parameters

Parameter	Description
<b>addType</b>	The type of address being used. Valid values are: IP_ADDRESS_TYPE_IPV4 or IP_ADDRESS_TYPE_IPV6
<b>remotePort</b>	TCP port to connect to. The local port for client sockets will be automatically picked by the TCP module.
<b>remoteAddress</b>	The remote address to be used

## Returns

Type	Description
<a href="#">TCP_SOCKET</a>	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 <a href="#">INVALID_SOCKET</a> .

## 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 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 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 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 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 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**, 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 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 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
<u>uint16_t</u>	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 Prototype

```
void TCPIP_UDP_Close(  
    UDP_SOCKET 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
<a href="#">void</a>	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 hUDP  
)
```

### Preconditions

UDP socket should have been opened with [TCPIP\\_UDP\\_ServerOpen](#)/[TCPIP\\_UDP\\_ClientOpen](#).

hUDP parameter is a valid socket

### Parameters

Parameter	Description
hUDP	UDP Socket Handle

### Returns

Type	Description
<a href="#">uint16_t</a>	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 TCPIP_UDP_ServerOpen(
    IP_ADDRESS_TYPE addType,
    UDP_PORT localPort,
    IP_MULTI_ADDRESS* localAddress
);
```

### Preconditions

UDP is initialized.

### Parameters

Parameter	Description
<a href="#">IP_ADDRESS_TYPE</a> addType	The type of address being used. <a href="#">IP_ADDRESS_TYPE_IPV4</a> or <a href="#">IP_ADDRESS_TYPE_IPV6</a> .
<a href="#">UDP_PORT</a> localPort	UDP port on which to listen for connections
<a href="#">IP_MULTI_ADDRESS*</a> localAddress	Local IP address to use. Can be 0 (NULL) if any incoming interface will do.

### Returns

Type	Description
<a href="#">UDP_SOCKET</a>	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 <a href="#">INVALID_SOCKET</a> .

## TCPIP\_UDP\_SocketInfoGet Function

This function will fill a user passed [UDP\\_SOCKET\\_INFO](#) structure with status of the selected socket

### Function Prototype

```
bool TCPIP_UDP_SocketInfoGet(
    UDP_SOCKET hUDP,
    UDP_SOCKET_INFO* pInfo
);
```

### Preconditions

UDP socket should have been opened with [TCPIP\\_UDP\\_ServerOpen\(\)](#)/[TCPIP\\_UDP\\_ClientOpen\(\)](#)().

hUDP - valid socket

pInfo - valid address of a [UDP\\_SOCKET\\_INFO](#) structure

### Parameters

Parameter	Description
hUDP	UDP Socket Handle
pInfo	Pointer to <a href="#">UDP_SOCKET_INFO</a> 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 addressType;
    IP_MULTI_ADDRESS remoteIPaddress;
    IP_MULTI_ADDRESS localIPaddress;
    IP_MULTI_ADDRESS sourceIPaddress;
    IP_MULTI_ADDRESS destIPaddress;
    UDP_PORT remotePort;
    UDP_PORT localPort;
    TCPIP_NET_HANDLE 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



**MICROCHIP**

