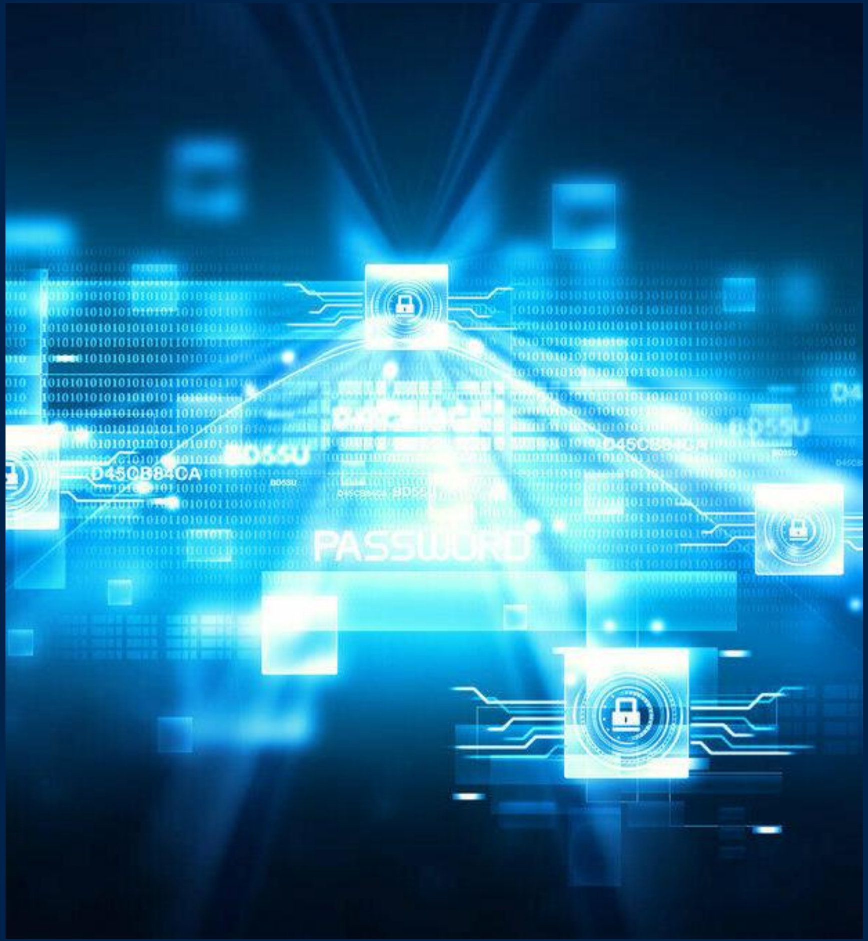


life.augmented

Azure RTOS Workshop

NetX Duo web server implementation

Tomas Dresler

V1.2

Agenda

#1 NetX Duo overview

#2 Web server demo

#3 Available examples

#4 Zero to server step-by-step

#5 CubeMx setup

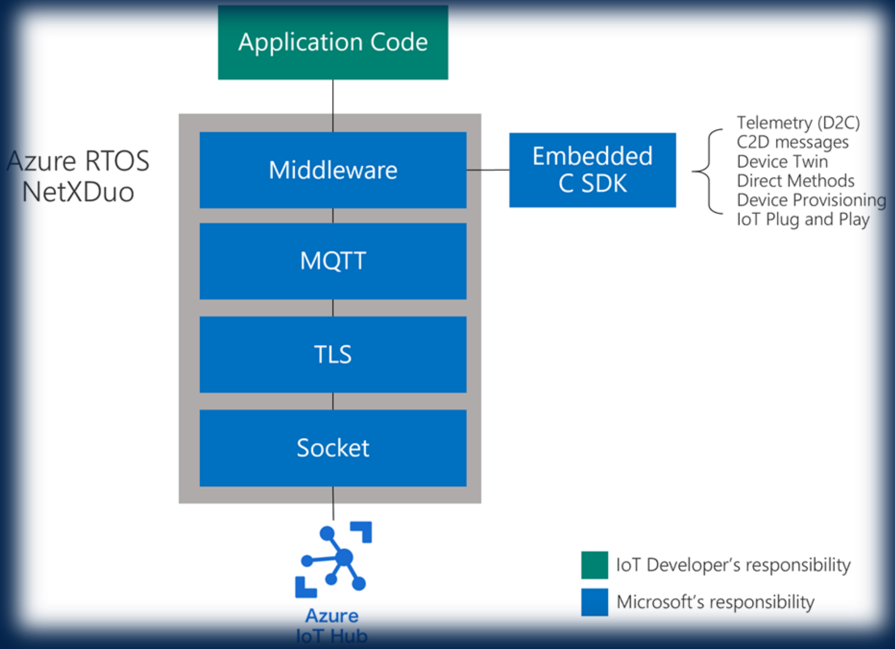
#6 Adding HTTP server

#7 Adding custom file system driver

#8 Adding web content

#9 Evaluation

NetX Duo overview



life.augmented

NetXDuo Resources

- Microsoft™ documentation portal: https://docs.microsoft.com/en-us/azure/rtos/netx-duo/
- ST Wiki: https://wiki.st.com/stm32mcu/wiki/NETXDUO_overview
- X-CUBE-AZRTOS-H7 pack for STM32CubeMx (sources, examples) is available here
  - Can be found on your HDD in *CubeMx\Repository\Packs\STMicroelectronics\X-CUBE-AZRTOS-H7\2.0.0*
  - Examples can be found in subdirectory *Projects\NUCLEO-H723ZG\Applications\NetXDuo* (*Nx_TCP_Echo_Client* and *Nx_TCP_Echo_Server*) or in folders of other evaluation boards
  - 31 generic NetXDuo examples are available here: *Middlewares\ST\netxduo\samples*

Certified for networking

IPv6 Ready certified

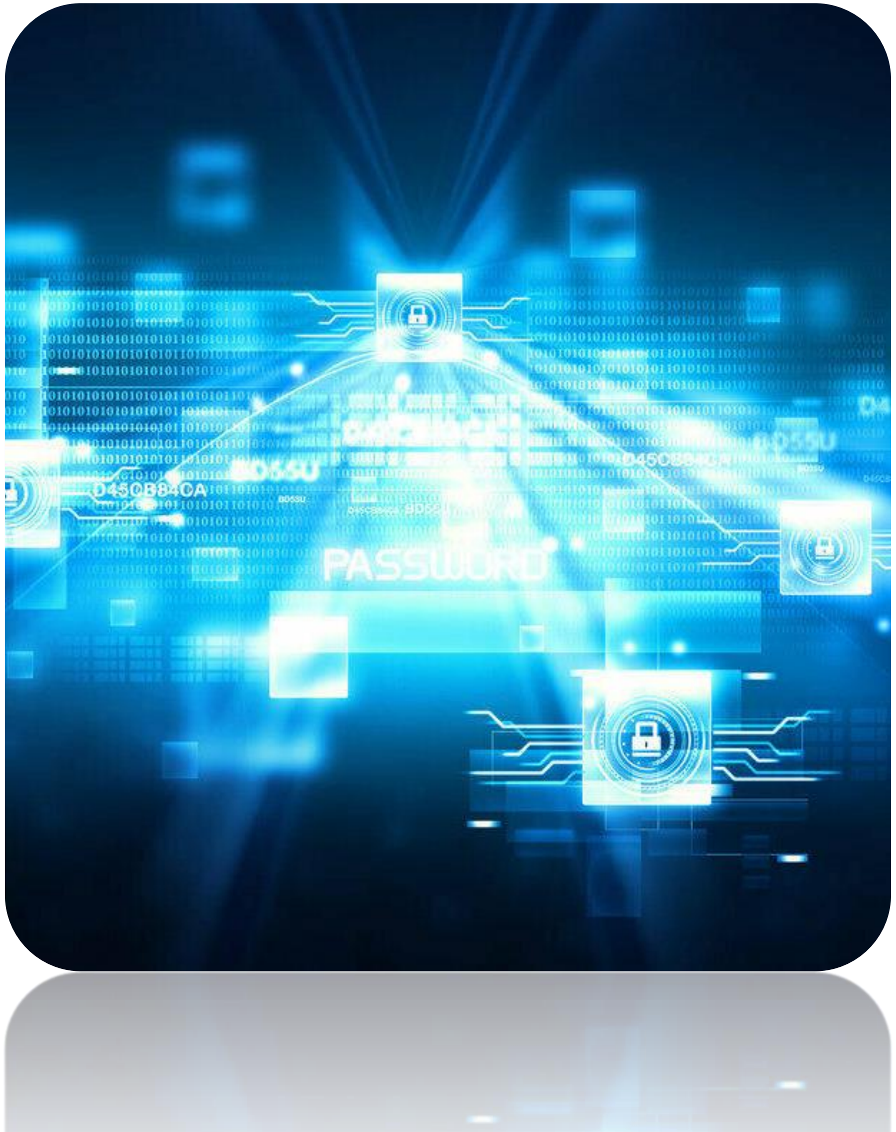
RFC-compliant

- RFCs for IPv4 networks: 1112, 1122, 2236, 768, 791, 792, 793, 826, 903, 5681,
- RFCs for IPv6 networks: 1981, 2460, 2464,4291, 4443, 4861, 4862

Testability

- IxANVL from IXIA, link

BSD-compatible API



Certified for security

Secure Internet protocols rely on underlying layers of security protocols

Among them, you can find HTTPS, FTPS, SMTP over TLS etc.

Secure protocols implemented:

- IPSec
- SSL
- TLS
- DTLS

RFC 2104	HMAC: Keyed-Hashing for Message Authentication
RFC 2246	The TLS Protocol Version 1.0
RFC 3268	Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)
RFC 3447	Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1
RFC 4279	Pre-Shared Key Ciphersuites for TLS
RFC 4346	The Transport Layer Security (TLS) Protocol Version 1.1
RFC 5246	The Transport Layer Security (TLS) Protocol Version 1.2
RFC 5280	X.509 PKI Certificates (v3)
RFC 5746	Transport Layer Security (TLS) Renegotiation Indication Extension
RFC 5869	HMAC-based Extract-and-Expand Key Derivation Function (HKDF)
RFC 6066[1]	Transport Layer Security (TLS) Extensions: Extension Definitions
RFC 6234	US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)
RFC 8443	Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier
RFC 8446	The Transport Layer Security (TLS) Protocol Version 1.3

Certified for appliances

TÜV Certification

- Appliances according to IEC61508 and IEC-62304
- Automotive according to ISO 26262 (ASIL D)
- Railway according to EN 50128 (SW-SIL 4)

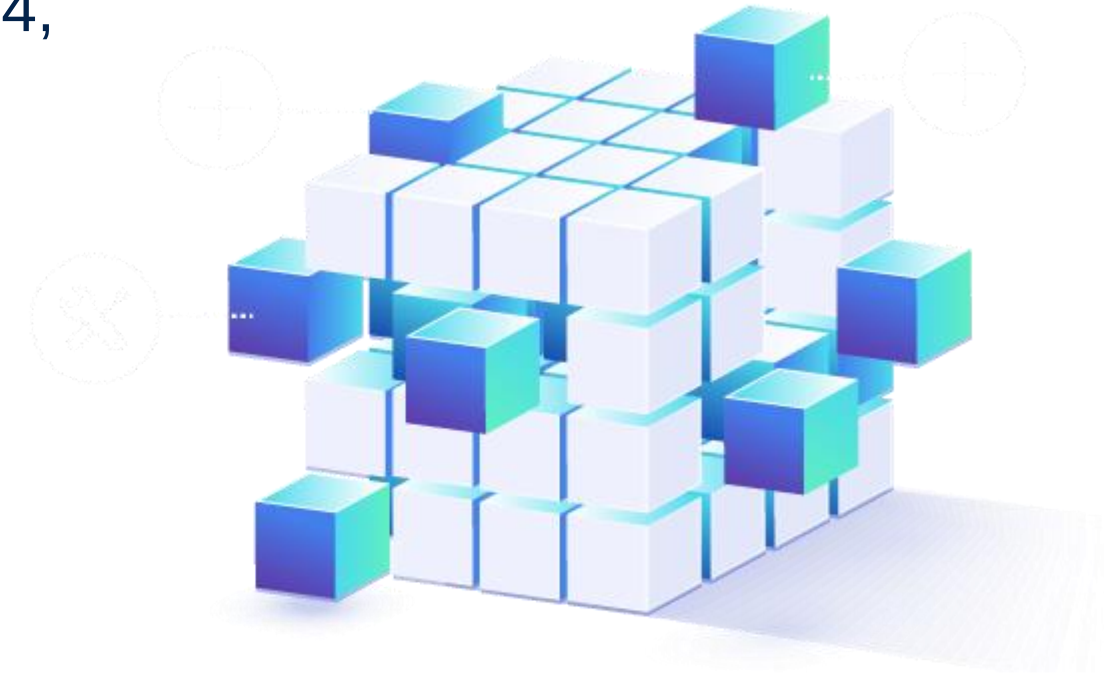
SGS TÜV SAAR
FUNKTIONALE SICHERHEIT GEPRÜFT
FUNCTIONAL SAFETY APPROVED

UL Certification

- UL 60730-1 Ann.H
- CSA E60730-1 Ann.H
- UL 60335-1 Ann.R
- IEC 60335-1 Ann.R
- UL 1998

Designed for small size and extensibility

- Piconet™ architecture
- Typical FLASH size 5-30 kB for IPv4, 30-45 kB for IPv4+IPv6
- ANSI C source code
- Highly modular
- TraceX support
- FileX support



Modularity in protocols (Link)

Protocol	Use	ROM [kB]	RAM [kB]	Protocol	Use	ROM	RAM
Auto IP	IPv4 addressing	1.2	0.3	**HTTP 1.0**	Client, server	2.8 – 4.8	0.4 – 1.0
DHCP	Client, server	3.6 – 4.6	2.7	**HTTP(S)**	Client, server, TLS	3 – 9.5	0.5 - 2
ICMP/IGMP	Ping/groups	2.5 / 2.5		**SMTP**	E-mail client	4.1	0.6
ARP/RARP	IP-2-MAC	1.7		**POP3**	E-mail client	8.1	1.4
IPv4/v6	Transport	3.5 – 8.5	2 - 3	**SNMP**	Management	10.9	2.6
UDP	Datagram	2.5	0.124/socket	**(T)FTP**	File transfer	1.8 - 7.2	0.6 – 2.1
TCP	Reliable conn.	10.8 – 12.5	0.28/socket	**MQTT**	IoT, telemetry	2.7	
(m)DNS, DNS-SD	Name resolution	2.4 - 3	1	**SNTP**	Time management	4	0.5
NAT	Bridging	3.5	0.6	**TLS/DTLS**	HW crypto support	8.8 / 11	

Designed for speed

- Zero-copy for TCP/IP, almost wire-throughput
- Support for HW acceleration
- Multiple interfaces (ETH, PPP, PPPoE) per
  - each IP stack – Multihoming (backup routes)
  - multiple IP stack instances allowed (bridging possible)
- Static routing

Security in NetX Duo

Security is managed by NetX Secure (*nx_secure* service)

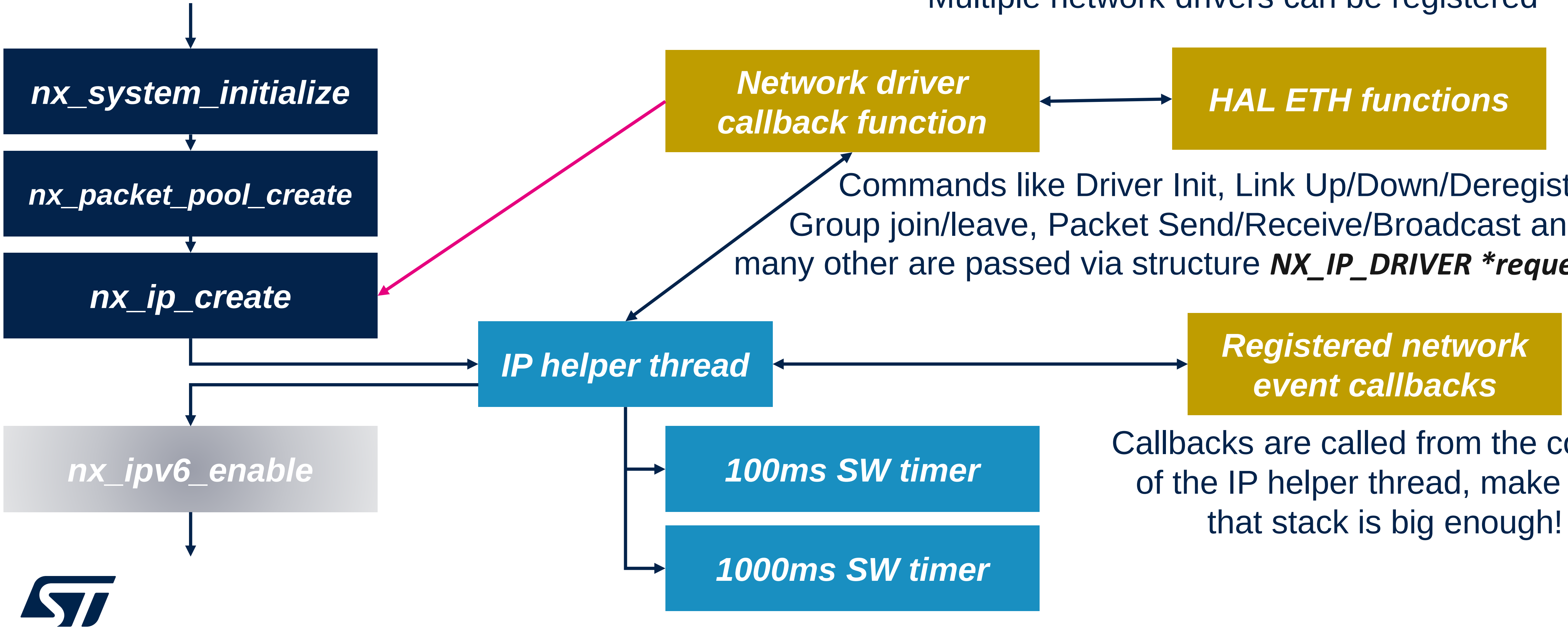
Implemented as MCU-oriented high-performance SW library

Some crypto services are computationally intensive
If MCU HW allows, it is preferred method

Complies with many RFCs

Initialization of the NetX Duo stack (details)

Call from ***tx_application_define***
or from user threads



nx_system_initialize

nx_packet_pool_create

nx_ip_create

IP helper thread

nx_ipv6_enable

100ms SW timer

1000ms SW timer

Multiple network drivers can be registered

Network driver callback function

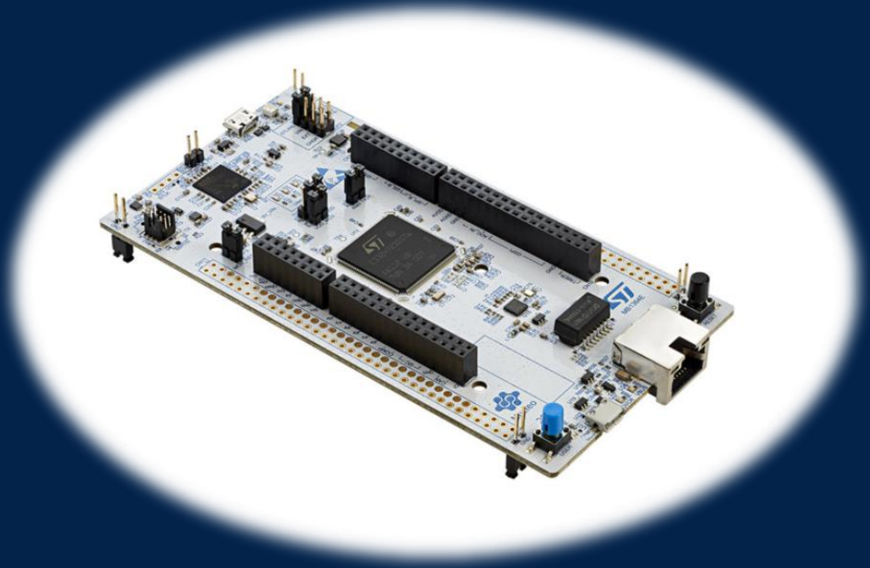
HAL ETH functions

Commands like Driver Init, Link Up/Down/Deregister, Group join/leave, Packet Send/Receive/Broadcast and many other are passed via structure ***NX_IP_DRIVER *request***

Registered network event callbacks

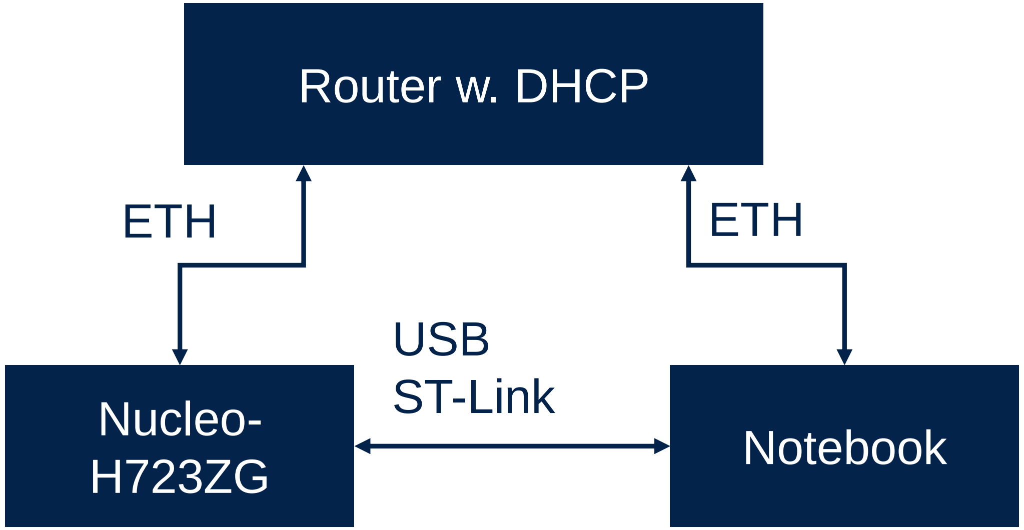
Callbacks are called from the context of the IP helper thread, make sure that stack is big enough!

Web server demo



Demo requirements

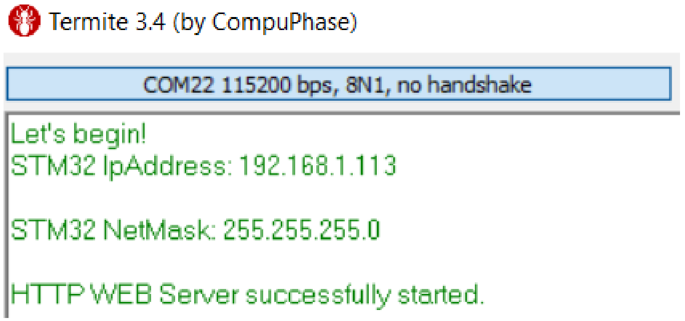
- Nucleo-H723ZG
- 2x Ethernet cable
- Micro USB - USB A cable
- Network router with DHCP service (can be replaced by local DHCP service, like DHCP server)
- PC with disabled firewall (Symantec, Windows)
- VCP driver installed
- Serial terminal



- Unpack ***AzureRTOSNetXDuo.zip***
- Launch ***.project*** in *STM32CubeIDE*
- Make, debug, run

Demo procedure

- Open the serial terminal with a Virtual COM port from ST-Link and setup the connection to 115.2kBd, 8N1



```
Termite 3.4 (by CompuPhase)
COM22 115200 bps, 8N1, no handshake
Let's begin!
STM32 IpAddress: 192.168.1.113

STM32 NetMask: 255.255.255.0

HTTP WEB Server successfully started.
```

- Run the code and grab the IP address assigned by the router or by local DHCP service:

  IP address: 192.168.1.113 (an example!)

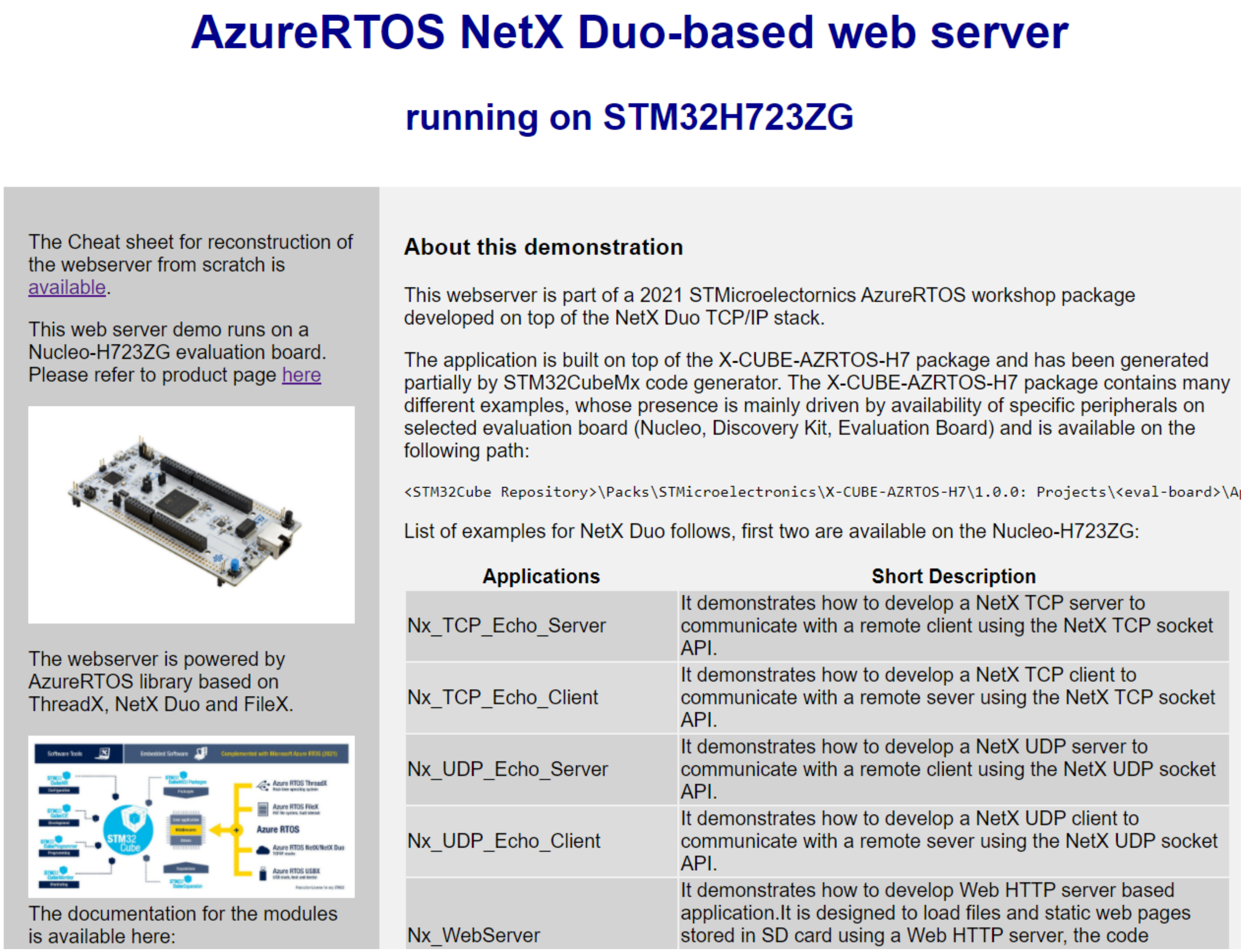
- Open a command line (⊞-R, cmd) and ping the Nucleo:

  ping 192.168.1.113

- Open web browser with IP address of the Nucleo:

  http://192.168.1.113/index.html

Voila, web page is served!



AzureRTOS NetX Duo-based web server

running on STM32H723ZG

The Cheat sheet for reconstruction of the webserver from scratch is available.

This web server demo runs on a Nucleo-H723ZG evaluation board. Please refer to product page here

The webserver is powered by AzureRTOS library based on ThreadX, NetX Duo and FileX.

The documentation for the modules is available here:

About this demonstration

This webserver is part of a 2021 STMicroelectornics AzureRTOS workshop package developed on top of the NetX Duo TCP/IP stack.

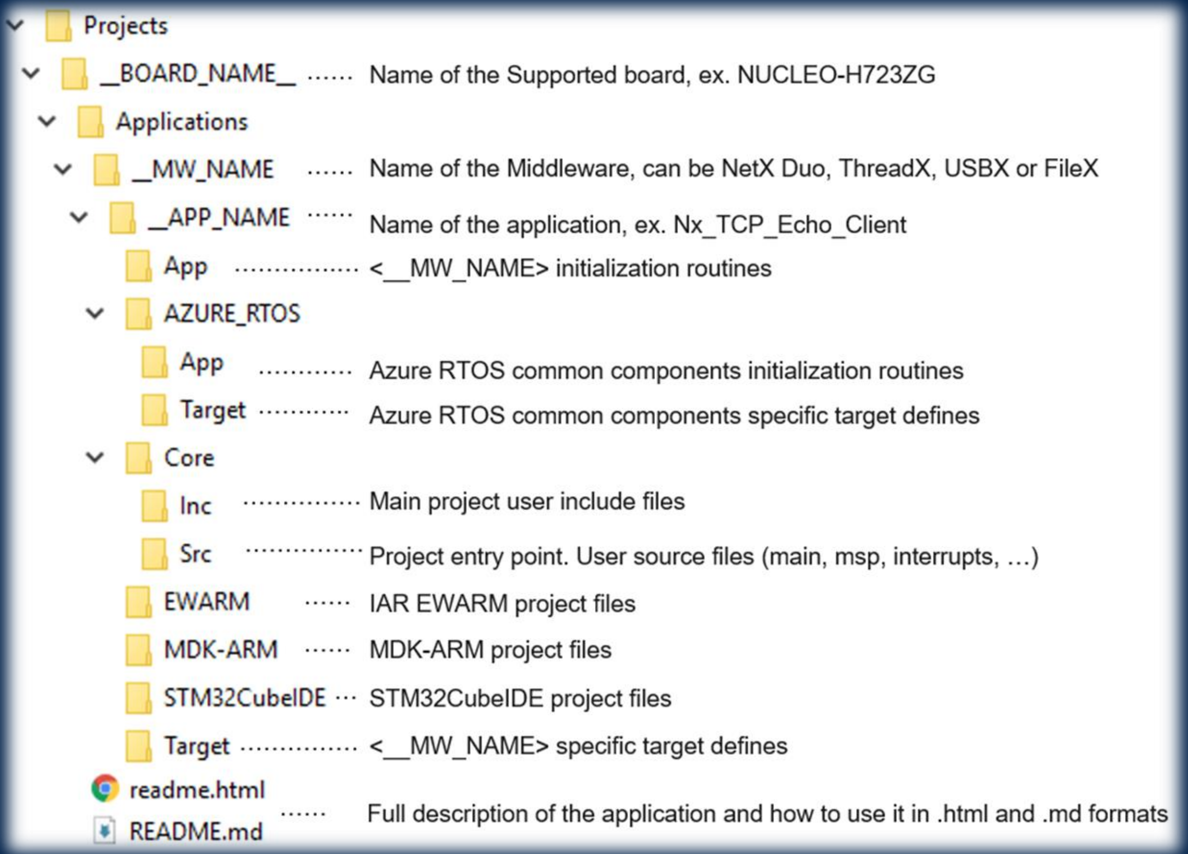
The application is built on top of the X-CUBE-AZRTOS-H7 package and has been generated partially by STM32CubeMx code generator. The X-CUBE-AZRTOS-H7 package contains many different examples, whose presence is mainly driven by availability of specific peripherals on selected evaluation board (Nucleo, Discovery Kit, Evaluation Board) and is available on the following path:

```
<STM32Cube Repository>\Packs\STMicroelectronics\X-CUBE-AZRTOS-H7\1.0.0: Projects\<eval-board>\A
```

List of examples for NetX Duo follows, first two are available on the Nucleo-H723ZG:

Applications	Short Description
Nx_TCP_Echo_Server	It demonstrates how to develop a NetX TCP server to communicate with a remote client using the NetX TCP socket API.
Nx_TCP_Echo_Client	It demonstrates how to develop a NetX TCP client to communicate with a remote sever using the NetX TCP socket API.
Nx_UDP_Echo_Server	It demonstrates how to develop a NetX UDP server to communicate with a remote client using the NetX UDP socket API.
Nx_UDP_Echo_Client	It demonstrates how to develop a NetX UDP client to communicate with a remote sever using the NetX UDP socket API.
Nx_WebServer	It demonstrates how to develop Web HTTP server based application.It is designed to load files and static web pages stored in SD card using a Web HTTP server, the code

Available examples



```
Projects
  _BOARD_NAME_ ······ Name of the Supported board, ex. NUCLEO-H723ZG
    Applications
      _MW_NAME ······ Name of the Middleware, can be NetX Duo, ThreadX, USBX or FileX
        _APP_NAME ······ Name of the application, ex. Nx_TCP_Echo_Client
          App ················ <__MW_NAME> initialization routines
          AZURE_RTOS
            App ············ Azure RTOS common components initialization routines
            Target ··········· Azure RTOS common components specific target defines
          Core
            Inc ················ Main project user include files
            Src ················ Project entry point. User source files (main, msp, interrupts, …)
          EWARM ······ IAR EWARM project files
          MDK-ARM ······ MDK-ARM project files
          STM32CubeIDE ··· STM32CubeIDE project files
          Target ················ <__MW_NAME> specific target defines
        readme.html
        README.md ······ Full description of the application and how to use it in .html and .md formats
```

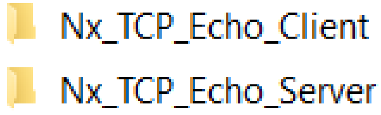
Example directories

Examples are available in CubeMx repository location:

- ***<STM32Cube Repository>\Packs\STMicroelectronics\X-CUBE-AZRTOS-H7\1.0.0***
  - ➢ ***Projects\<eval-board>\Applications\NetXDuo***
  - ➢ ***Middlewares\ST\netxduo\samples***

Each board contains only examples supported by its own hardware, you can find other for different boards and adapt your code.

Nucleo-H723ZG contains only two:



Nx_TCP_Echo_Client
Nx_TCP_Echo_Server

Examples for NetX Duo for various boards

Nx_TCP_Echo_Server	It demonstrates how to develop a NetX TCP server to communicate with a remote client using the NetX TCP socket API.
Nx_TCP_Echo_Client	It demonstrates how to develop a NetX TCP client to communicate with a remote sever using the NetX TCP socket API.
Nx_UDP_Echo_Server	It demonstrates how to develop a NetX UDP server to communicate with a remote client using the NetX UDP socket API.
Nx_UDP_Echo_Client	It demonstrates how to develop a NetX UDP client to communicate with a remote sever using the NetX UDP socket API.
Nx_WebServer	It demonstrates how to develop Web HTTP server based application.It is designed to load files and static web pages stored in SD card using a Web HTTP server, the code provides all required features to build a compliant Web HTTP Server.
Nx_MQTT_Client	It demonstrates how to exchange data between client and server using MQTT protocol in an encrypted mode supporting TLS v1.2.
Nx_SNTP_Client	It demonstrates how to develop a NetX SNTP client and connect with an STNP server to get a time update.

Examples in Middlewares\ST\NetXDuo\samples

- *demo_bsd_raw.c*
- *demo_bsd_tcp.c*
- *demo_bsd_udp.c*
- *demo_mqtt_client.c*
- *demo_netxduo_dhcp.c*
- *demo_netxduo_dhcpv6.c*
- *demo_netxduo_dhcpv6_client.c*
- *demo_netxduo_dns.c*
- *demo_netxduo_ftp.c*
- *demo_netxduo_http.c*
- *demo_netxduo_https.c*
- *demo_netxduo_multihome_dhcp_client.c*
- *demo_netxduo_pop3_client.c*
- *demo_netxduo_smtp_client.c*
- *demo_netxduo_snmp.c*
- *demo_netxduo_sntp_client.c*
- *demo_netxduo_telnet.c*
- *demo_netxduo_tftp.c*
- *demo_netx_auto_ip.c*
- *demo_netx_duo_lwm2m_client.c*
- *demo_netx_duo_mdns.c*
- *demo_netx_duo_multihome_tcp.c*
- *demo_netx_duo_multihome_udp.c*
- *demo_netx_duo_ptp_client.c*
- *demo_netx_duo_tcp.c*
- *demo_netx_duo_udp.c*
- *demo_netx_nat.c*
- *demo_netx_ppp.c*
- *demo_netx_pppoe_client.c*
- *demo_netx_pppoe_server.c*

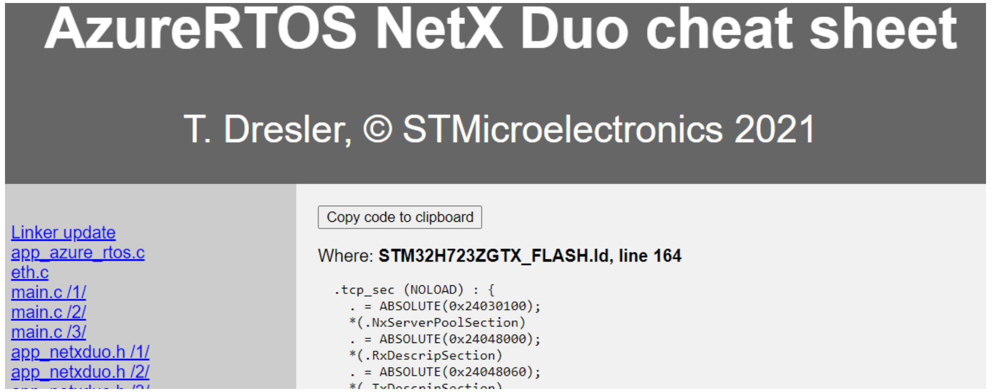
Zero to server step-by-step

life.augmented

Zero to server step-by-step

- Purpose of the following slides is to demonstrate step-by-step the procedure to adapt default Nucleo-H723ZG setup in CubeMx into basic working web server
- The procedure adapts basic CubeMx configuration, adds web content and support files and shows where to add the network functionality from basic protocols like ARP to a web server instance

- The procedure is rather long and if attention is not paid enough, it may discourage from proper implementation
- Thus, a cheat sheet has been created: *Cheatsheet_NetXDuo.html*, allowing to highlight and copy & paste the appropriate code to your project



AzureRTOS NetX Duo cheat sheet

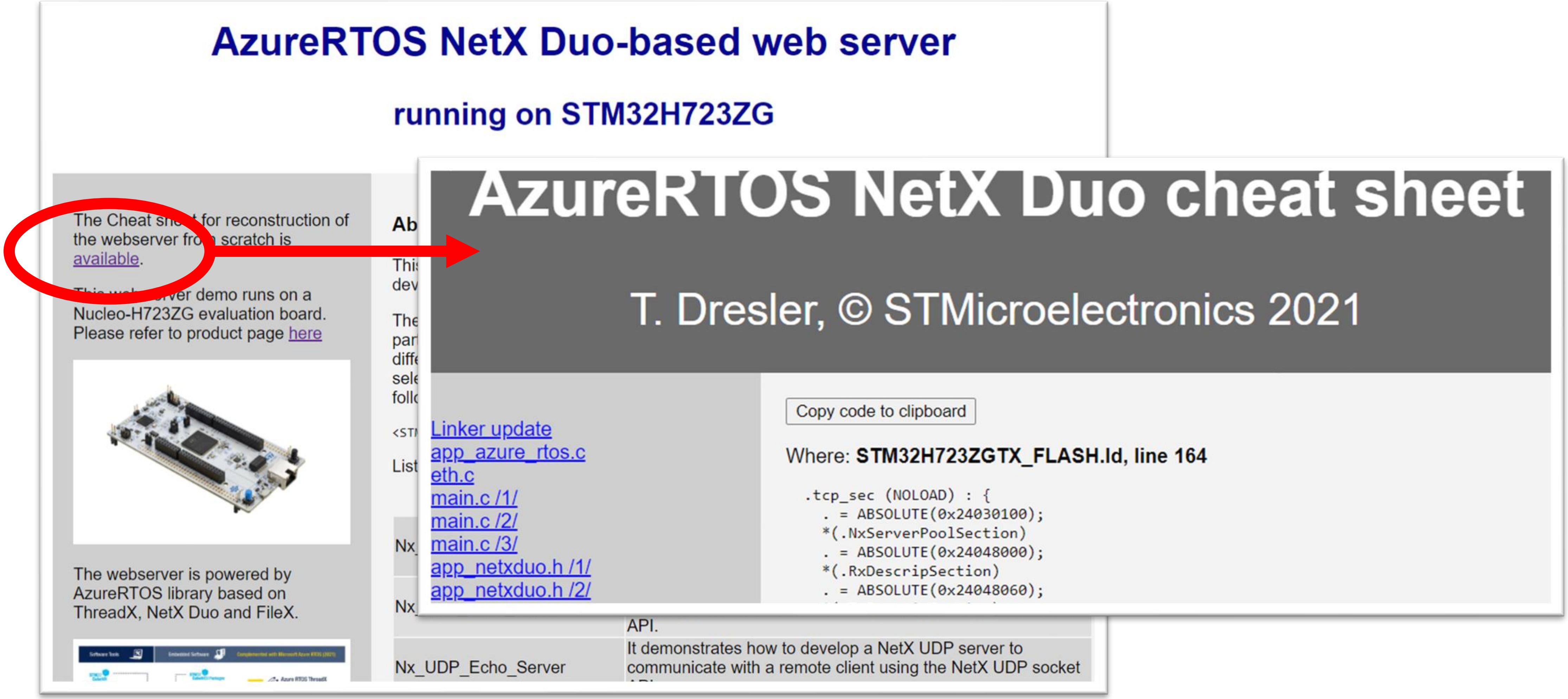
T. Dresler, © STMicroelectronics 2021

Where: **STM32H723ZGTX_FLASH.ld, line 164**

```
  .tcp_sec (NOLOAD) : {
    . = ABSOLUTE(0x24030100);
    *(.NxServerPoolSection)
    . = ABSOLUTE(0x24048000);
    *(.RxDescripSection)
    . = ABSOLUTE(0x24048060);
    *(.TxDescripSection)
```

Cheat sheet

- Cheat sheet is available as a webpage in your demo, try it!



AzureRTOS NetX Duo-based web server

running on STM32H723ZG

The Cheat sheet for reconstruction of the webserver from scratch is available.

This webserver demo runs on a Nucleo-H723ZG evaluation board. Please refer to product page here

The webserver is powered by AzureRTOS library based on ThreadX, NetX Duo and FileX.

Ab

This dev

The par diffe sele follo

<STI

List

Nx_

Nx_ API.

Nx_UDP_Echo_Server It demonstrates how to develop a NetX UDP server to communicate with a remote client using the NetX UDP socket API.

AzureRTOS NetX Duo cheat sheet

T. Dresler, © STMicroelectronics 2021

Linker update
app_azure_rtos.c
eth.c
main.c /1/
main.c /2/
main.c /3/
app_netxduo.h /1/
app_netxduo.h /2/

Copy code to clipboard

Where: **STM32H723ZGTX_FLASH.ld, line 164**

```
.tcp_sec (NOLOAD) : {
  . = ABSOLUTE(0x24030100);
  *(.NxServerPoolSection)
  . = ABSOLUTE(0x24048000);
  *(.RxDescripSection)
  . = ABSOLUTE(0x24048060);
```

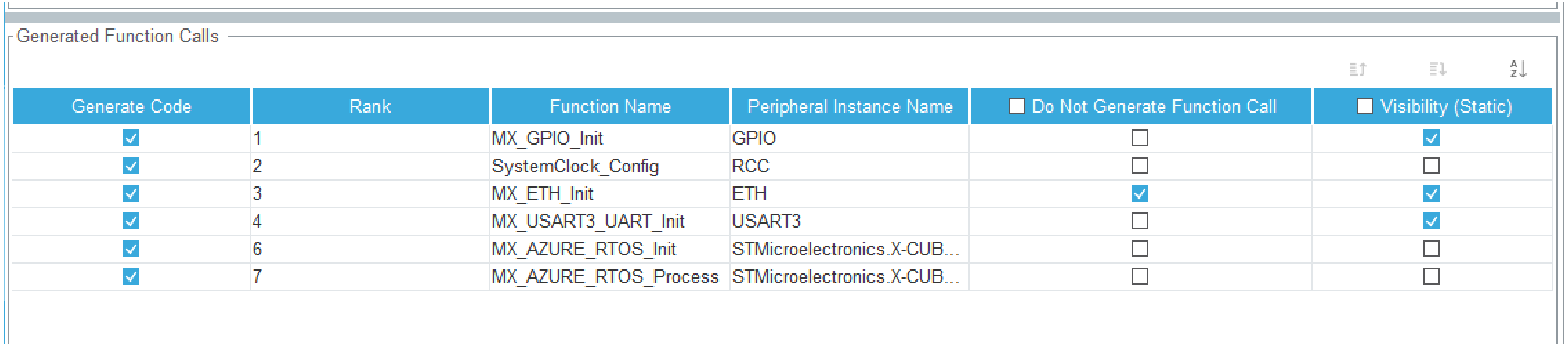
Setup in CubeMx

life.augmented

Setup in CubeMx

- Let’s use Nucleo-H723ZG with all peripherals enabled
- Let’s use AZRTOS-H7-2.0.0 Pack
- Let’s use CubeH7_HAL 1.9.0

- Let’s setup the project with following details:
  - IDE: STM32CubeIDE
  - Copy only the necessary files
  - Generate peripheral initialization as pair of .c/.h files
  - In Advanced settings, check “Don’t generate function call for MX_ETH_Init”

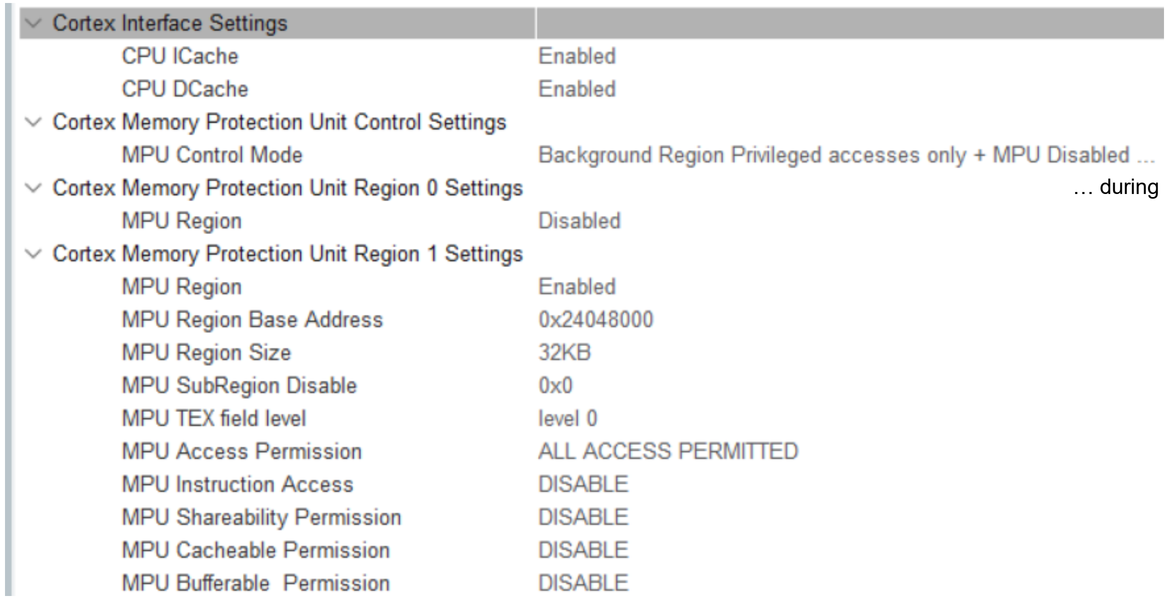


Generated Function Calls

Generate Code	Rank	Function Name	Peripheral Instance Name	□ Do Not Generate Function Call	□ Visibility (Static)
☑	1	MX_GPIO_Init	GPIO	□	☑
☑	2	SystemClock_Config	RCC	□	□
☑	3	MX_ETH_Init	ETH	☑	☑
☑	4	MX_USART3_UART_Init	USART3	□	☑
☑	6	MX_AZURE_RTOS_Init	STMicroelectronics.X-CUB...	□	□
☑	7	MX_AZURE_RTOS_Process	STMicroelectronics.X-CUB...	□	□

Setup in CubeMx

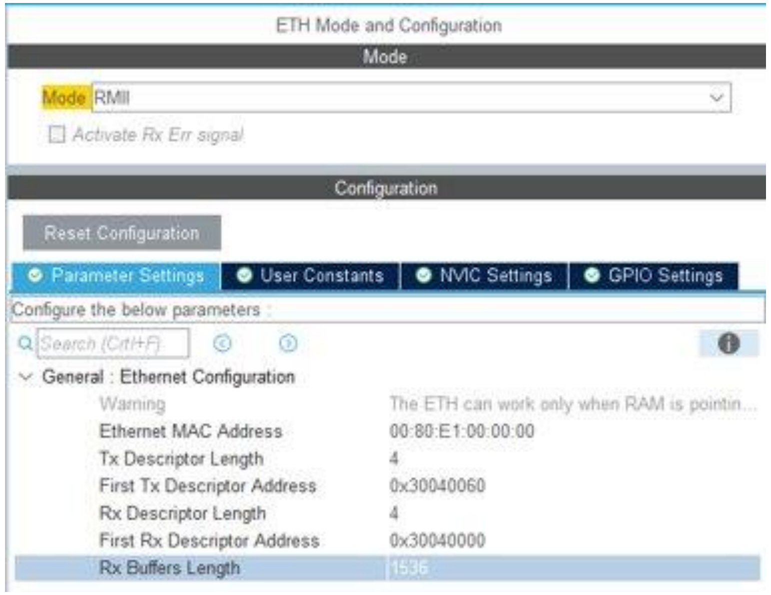
- Cortex-M7 setup:



Cortex Interface Settings	
CPU ICache	Enabled
CPU DCache	Enabled
Cortex Memory Protection Unit Control Settings	
MPU Control Mode	Background Region Privileged accesses only + MPU Disabled ...
Cortex Memory Protection Unit Region 0 Settings	
MPU Region	Disabled
Cortex Memory Protection Unit Region 1 Settings	
MPU Region	Enabled
MPU Region Base Address	0x24048000
MPU Region Size	32KB
MPU SubRegion Disable	0x0
MPU TEX field level	level 0
MPU Access Permission	ALL ACCESS PERMITTED
MPU Instruction Access	DISABLE
MPU Shareability Permission	DISABLE
MPU Cacheable Permission	DISABLE
MPU Bufferable Permission	DISABLE

… during HF

- ETH setup
  - Enable global ETH IRQ
  - Change Rx Buffers Length to 1536



ETH Mode and Configuration

Mode

Mode RMII

□ Activate Rx Err signal

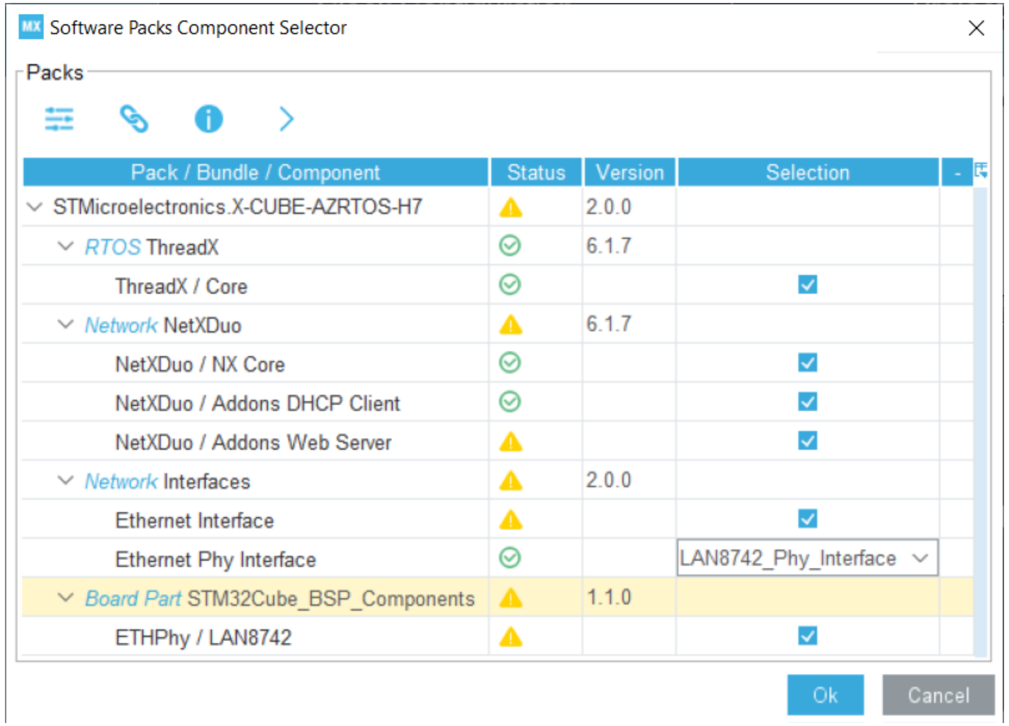
Configuration

Configure the below parameters :

General : Ethernet Configuration	
Warning	The ETH can work only when RAM is pointin...
Ethernet MAC Address	00:80:E1:00:00:00
Tx Descriptor Length	4
First Tx Descriptor Address	0x30040060
Rx Descriptor Length	4
First Rx Descriptor Address	0x30040000
Rx Buffers Length	1536

Setup in CubeMx

- In the menu Software Packs, choose Select Components (Alt-O) and tick/choose the following options:

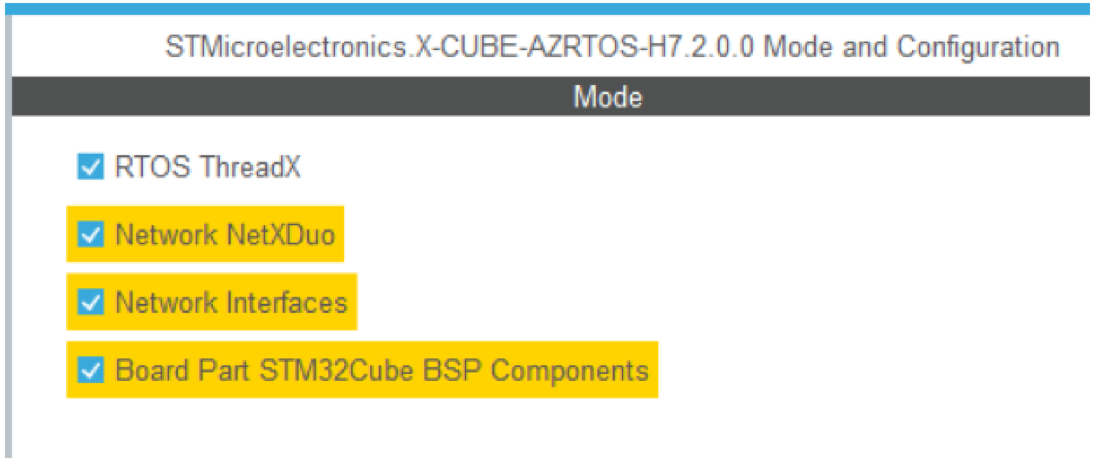


Software Packs Component Selector

Packs

Pack / Bundle / Component	Status	Version	Selection
STMicroelectronics.X-CUBE-AZRTOS-H7	⚠	2.0.0	
RTOS ThreadX	✓	6.1.7	
ThreadX / Core	✓		☑
Network NetXDuo	⚠	6.1.7	
NetXDuo / NX Core	✓		☑
NetXDuo / Addons DHCP Client	✓		☑
NetXDuo / Addons Web Server	⚠		☑
Network Interfaces	⚠	2.0.0	
Ethernet Interface	⚠		☑
Ethernet Phy Interface	✓		LAN8742_Phy_Interface
Board Part STM32Cube_BSP_Components	⚠	1.1.0	
ETHPhy / LAN8742	⚠		☑

- Enable all selected Azure RTOS components:



STMicroelectronics.X-CUBE-AZRTOS-H7.2.0.0 Mode and Configuration

Mode

☑ RTOS ThreadX

☑ Network NetXDuo

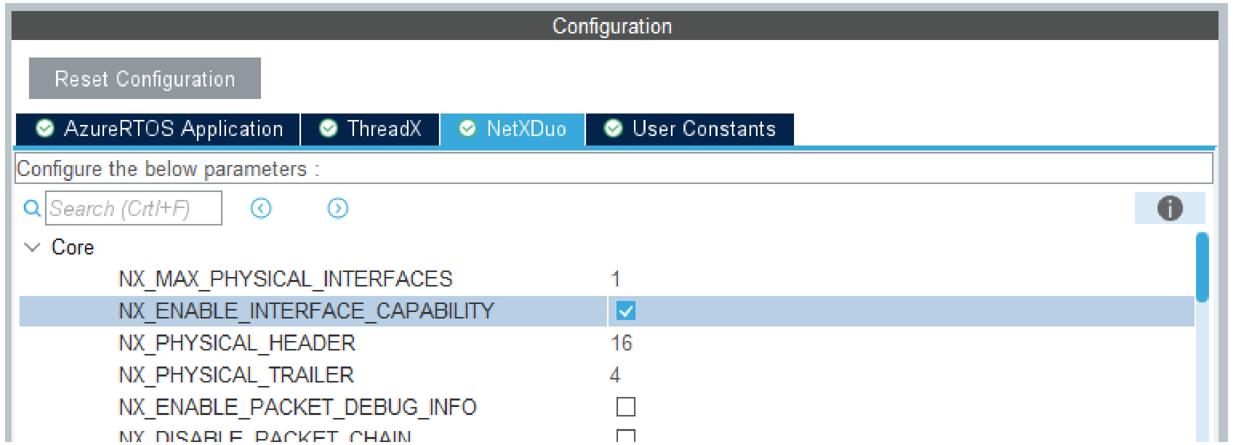
☑ Network Interfaces

☑ Board Part STM32Cube BSP Components

- Please ignore yellow warnings, they are treated in the project & code

Setup in CubeMx

- In X-Cube-AzureRTOS, NetXDuo, enable *NX_ENABLE_INTERFACE_CAPABILITY*

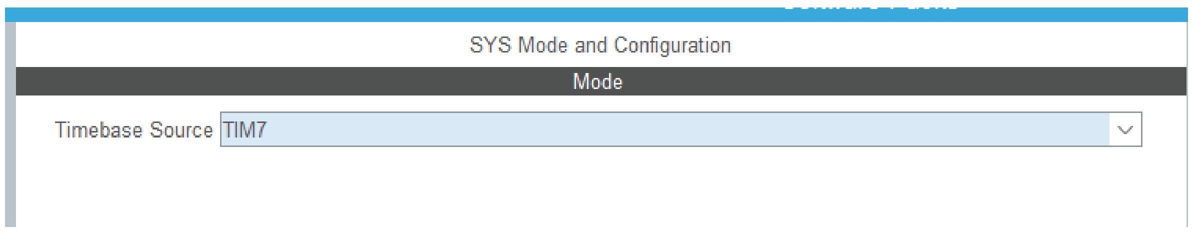


Configure the below parameters :

Core

```
NX_MAX_PHYSICAL_INTERFACES        1
NX_ENABLE_INTERFACE_CAPABILITY    ☑
NX_PHYSICAL_HEADER                16
NX_PHYSICAL_TRAILER               4
NX_ENABLE_PACKET_DEBUG_INFO       □
NX_DISABLE_PACKET_CHAIN           □
```

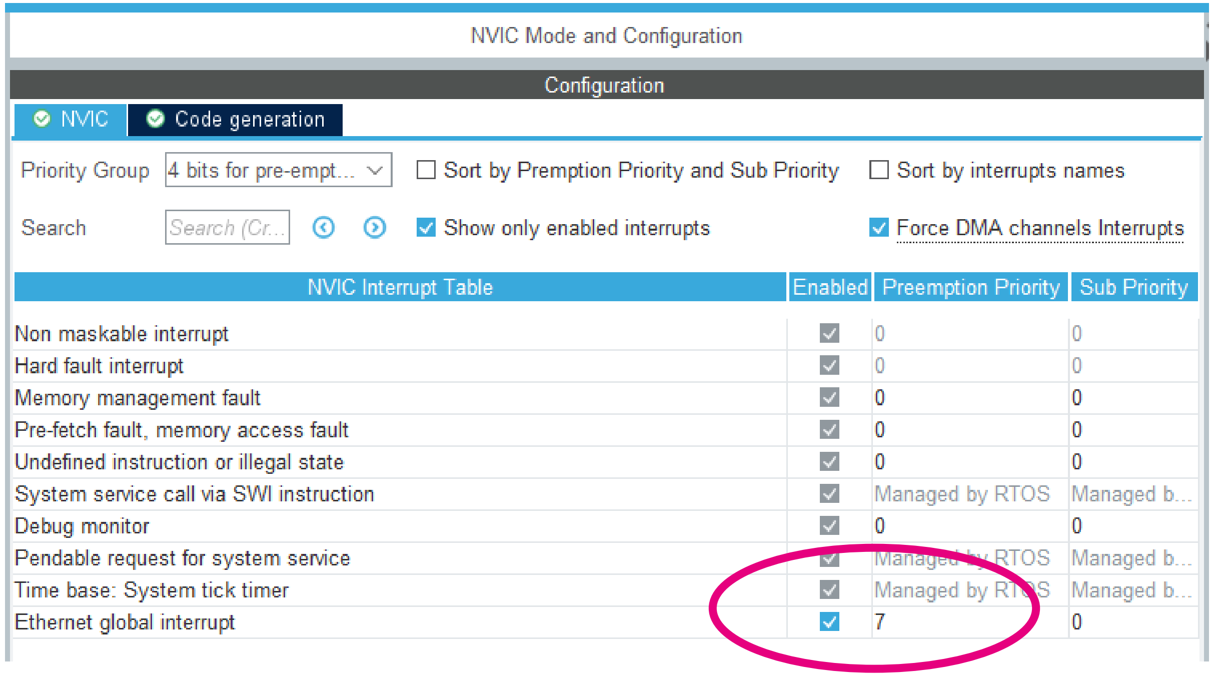
- Alter HAL time base to TIM7



Timebase Source: TIM7

Setup in CubeMx

- Alter NVIC / ETH Global IRQ, set Priority to 7



NVIC Mode and Configuration

Configuration

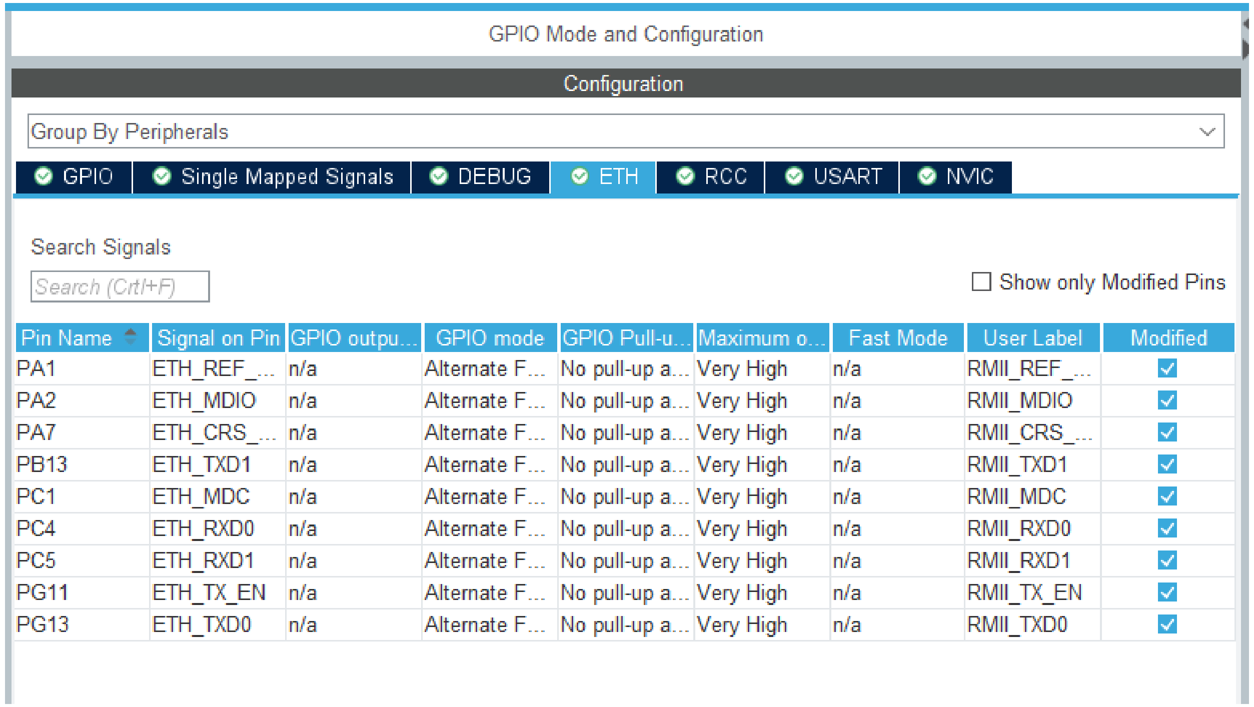
NVIC | Code generation

Priority Group: 4 bits for pre-empt... □ Sort by Premption Priority and Sub Priority □ Sort by interrupts names

Search: Search (Cr... ☑ Show only enabled interrupts ☑ Force DMA channels Interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	Managed by RTOS	Managed b...
Debug monitor	☑	0	0
Pendable request for system service	☑	Managed by RTOS	Managed b...
Time base: System tick timer	☑	Managed by RTOS	Managed b...
Ethernet global interrupt	☑	7	0

- In GPIO / ETH, set speed of **all** GPIOs to Very High



GPIO Mode and Configuration

Configuration

Group By Peripherals

GPIO | Single Mapped Signals | DEBUG | ETH | RCC | USART | NVIC

Search Signals: Search (Crtl+F) □ Show only Modified Pins

Pin Name	Signal on Pin	GPIO outpu...	GPIO mode	GPIO Pull-u...	Maximum o...	Fast Mode	User Label	Modified
PA1	ETH_REF_...	n/a	Alternate F...	No pull-up a...	Very High	n/a	RMII_REF_...	☑
PA2	ETH_MDIO	n/a	Alternate F...	No pull-up a...	Very High	n/a	RMII_MDIO	☑
PA7	ETH_CRS_...	n/a	Alternate F...	No pull-up a...	Very High	n/a	RMII_CRS_...	☑
PB13	ETH_TXD1	n/a	Alternate F...	No pull-up a...	Very High	n/a	RMII_TXD1	☑
PC1	ETH_MDC	n/a	Alternate F...	No pull-up a...	Very High	n/a	RMII_MDC	☑
PC4	ETH_RXD0	n/a	Alternate F...	No pull-up a...	Very High	n/a	RMII_RXD0	☑
PC5	ETH_RXD1	n/a	Alternate F...	No pull-up a...	Very High	n/a	RMII_RXD1	☑
PG11	ETH_TX_EN	n/a	Alternate F...	No pull-up a...	Very High	n/a	RMII_TX_EN	☑
PG13	ETH_TXD0	n/a	Alternate F...	No pull-up a...	Very High	n/a	RMII_TXD0	☑

Generated code

Generated code (project fix)

- Let’s remove the *syscalls.c* and *sysmem.c* from the project
- Copy following headers from HAL pack to your project *Core/Inc* folder: *stm32h7xx_nucleo.h, stm32h7xx_nucleo_errno.h* and *stm32h7xx_nucleo_conf.h*
- Add *stm32h7xx_nucleo.c* from *HAL package/Drivers/BSP* into your *Core/Src* folder
- Add *filex_flash_stub.c*, *filex_stub.h* and *web_data.c* from ZIP file to *NetXDuo\App* folder (too long to be created manually)
- Refresh your project

STM32H723ZGTX_FLASH.ld (cf. cheat sheet!)

- The generated code contains plenty of files, but the main functionality (network and services initialization) is missing. Some adaptations need to be done, too, to the linker file for proper functionality of the Ethernet driver and HTTP server
- Let’s first look at linker file: *STM32H723ZGTX_FLASH.ld*
- Due to inability of ETH to access CCM-RAM at 0x20000000, let’s move data to RAM_D1 and change stack

- Make sure the _estack is defined as follows:

```
_estack = ORIGIN(RAM_D1) + LENGTH(RAM_D1);    /* end of
RAM */
```

- Insert following text at line 166 (before section /DISCARD/):

```
.tcp_sec (NOLOAD) : {
    . = ABSOLUTE(0x24030100);
    *(.NxServerPoolSection)
    . = ABSOLUTE(0x24048000);
    *(.RxDescripSection)
    . = ABSOLUTE(0x24048060);
    *(.TxDescripSection)
  } >RAM_D1 AT> FLASH

.nx_data 0x24048200 (NOLOAD) : {
   *(.NetXPoolSection)
 } >RAM_D1 AT >FLASH
```

life.augmented

Generated code (app_azure_rtos setup)

- In the file *app_azure_rtos.c* we need to add memory placement. Locate section /* USER CODE BEGIN NX_Pool_Buffer */ and add following code inside the section:

```
#if defined ( __ICCARM__ ) /* IAR Compiler */
#pragma location = ".NetXPoolSection"
#elif defined ( __CC_ARM ) /* MDK ARM Compiler */
__attribute__((section(".NetXPoolSection")))
#elif defined ( __GNUC__ ) /* GNU Compiler */
__attribute__((section(".NetXPoolSection")))
#endif
```

- In the file *app_azure_rtos_config.h* let’s change the allocated pool size as follows:

```
#define NX_APP_MEM_POOL_SIZE                    30*1024
```

- *main.h* requires adding Nucleo header in the section /* USER CODE BEGIN Includes */

```
#include "stm32h7xx_nucleo.h"
```

life.augmented

Generated code (eth.c fix, use cheat sheet!)

- *Eth.c* offers this fix:
  - Remove TxConfig variable declaration and usage anywhere in the file
  - Remove Rx_Buff variable declaration, if present
  - The static addresses of Descriptors shall be updated:

```
#if defined ( __ICCARM__ ) /*!< IAR Compiler */
#pragma location=0x24048000
ETH_DMADescTypeDef  DMARxDscrTab[ETH_RX_DESC_CNT]; /* Ethernet Rx DMA Descriptors */
#pragma location=0x24048060
ETH_DMADescTypeDef  DMATxDscrTab[ETH_TX_DESC_CNT]; /* Ethernet Tx DMA Descriptors */
#elif defined ( __CC_ARM )  /* MDK ARM Compiler */
__attribute__((at(0x24048000))) ETH_DMADescTypeDef  DMARxDscrTab[ETH_RX_DESC_CNT]; /* Ethernet Rx DMA Descriptors */
__attribute__((at(0x24048060))) ETH_DMADescTypeDef  DMATxDscrTab[ETH_TX_DESC_CNT]; /* Ethernet Tx DMA Descriptors */
#elif defined ( __GNUC__ ) /* GNU Compiler */
ETH_DMADescTypeDef DMARxDscrTab[ETH_RX_DESC_CNT] __attribute__((section(".RxDescripSection"))); /* Ethernet Rx DMA Descriptors */
ETH_DMADescTypeDef DMATxDscrTab[ETH_TX_DESC_CNT] __attribute__((section(".TxDescripSection")));   /* Ethernet Tx DMA Descriptors */
#endif
```

life.augmented

Generated code (main.c fix, use cheat sheet!)

- Enable access to RAM in D2 domain:

```
/* USER CODE BEGIN 1 */
RCC->AHB2ENR |= (RCC_AHB2ENR_D2SRAM1EN | RCC_AHB2ENR_D2SRAM2EN);
/* USER CODE END 1 */
```

- Initialize LEDs:

```
/* USER CODE BEGIN SysInit */
BSP_LED_Init(LED1);
BSP_LED_Init(LED2);
/* USER CODE END SysInit */
```

- Define IO output:

```
/* USER CODE BEGIN 4 */
int _write(int file, char *ptr, int len)
{
  HAL_UART_Transmit(&huart3, (uint8_t *)ptr, len, 0xFFFF);
  return len;
}
/* USER CODE END 4 */
```

NetXDuo prerequisities

NetXDuo prerequisities (app_netxduo.h)

- Let’s include protocol headers in *"/* USER CODE BEGIN Includes */"*

```
#include <stdio.h>
#include "main.h"
#include "nxd_dhcp_client.h"
#include "nx_web_http_server.h"
```

- Add following to */* USER CODE BEGIN PD */*

```
#define PAYLOAD_SIZE             1536
#define NX_PACKET_POOL_SIZE      ((PAYLOAD_SIZE + sizeof(NX_PACKET)) * 10)
#define DEFAULT_MEMORY_SIZE      1024
#define DEFAULT_PRIORITY         10
#define WINDOW_SIZE              512
#define NULL_ADDRESS             0
#define DEFAULT_PORT             6000
#define MAX_TCP_CLIENTS          1
```

- And in */* USER CODE BEGIN EM */*

```
#define PRINT_IP_ADDRESS(addr)           do { \
printf("STM32 %s: %d.%d.%d.%d\n", #addr, \
(addr >> 24) & 0xff, (addr >> 16) & 0xff, \
(addr >> 8) & 0xff, addr & 0xff); \
} while(0)
```

- *Add following to /* USER CODE BEGIN 1 */*

```
/* HTTP connection port */
#define CONNECTION_PORT     80
/* Server packet size */
#define SERVER_PACKET_SIZE (NX_WEB_HTTP_SERVER_MIN_PACKET_SIZE * 2)
/* Server stack */
#define SERVER_STACK             4096
/* Server pool size */
#define SERVER_POOL_SIZE      (SERVER_PACKET_SIZE * 4)
```

life.augmented

NetXDuo prerequisities (app_netxduo.c)

- First, let's define some threads, semaphore, packet pools, IP stack instance, DHCP client instance and other variables
- Put these in the section *\/* USER CODE BEGIN PV *\/*
- Now let's define *nx_server_pool* in the same section, guaranteeing proper placement in the memory depending on the compiler

```
TX_THREAD AppMainThread;
TX_THREAD AppTCPThread;
TX_THREAD AppWebServerThread;
TX_SEMAPHORE Semaphore;
NX_PACKET_POOL AppPool;
NX_PACKET_POOL WebServerPool;
ULONG IpAddress;
ULONG NetMask;
NX_IP IpInstance;
NX_DHCP DHCPClient;
NX_WEB_HTTP_SERVER HTTPServer;
UCHAR *pointer;
/* Set nx_server_pool start address to 0x24030100 */
#if defined ( __ICCARM__ ) /* IAR Compiler */
#pragma location = 0x24030100
#elif defined ( __CC_ARM ) /* MDK ARM Compiler */
__attribute__((section(".NxServerPoolSection")))
#elif defined ( __GNUC__ ) /* GNU Compiler */
__attribute__((section(".NxServerPoolSection")))
#endif
static uint8_t nx_server_pool[SERVER_POOL_SIZE];
```

NetXDuo prerequisities (app_netxduo.c)

- Let’s add private function prototypes of new functions in this file in the section

```
/* USER CODE BEGIN PFP */
static VOID App_Main_Thread_Entry(ULONG thread_input);
static VOID ip_address_change_notify_callback(NX_IP *ip_instance, VOID *ptr);

/* Web Server callback when a new request from a web client is triggered */
static UINT webserver_request_notify_callback(NX_WEB_HTTP_SERVER *server_ptr, UINT request_type,
  CHAR *resource, NX_PACKET *packet_ptr);
/* USER CODE END PFP */
```

Adding TCP/IP server

life.augmented

Adding TCP/IP server (app_netxduo.c)

- Let’s create IP instance within function App_NetXDuo_Init in the section *`/* USER CODE BEGIN App_NetXDuo_Init */`*
- … to be continued

```
/* USER CODE BEGIN App_NetXDuo_Init */
 // (void)byte_pool;

 /* Allocate the memory for packet_pool.  */
 if (tx_byte_allocate(byte_pool, (VOID **) &pointer,  NX_PACKET_POOL_SIZE, TX_NO_WAIT) != TX_SUCCESS)
 {
   return TX_POOL_ERROR;
 }

 /* Create the Packet pool to be used for packet allocation */
 ret = nx_packet_pool_create(&AppPool, "Main Packet Pool", PAYLOAD_SIZE, pointer, NX_PACKET_POOL_SIZE);

 if (ret != NX_SUCCESS) {
   return NX_NOT_ENABLED;
 }

 /* Allocate the memory for Ip_Instance */
 if (tx_byte_allocate(byte_pool, (VOID **) &pointer,   2 * DEFAULT_MEMORY_SIZE, TX_NO_WAIT) != TX_SUCCESS)
 {
   return TX_POOL_ERROR;
 }

 /* Create the main NX_IP instance */
 ret = nx_ip_create(&IpInstance, "Main Ip instance", NULL_ADDRESS, NULL_ADDRESS, &AppPool, nx_stm32_eth_driver,
                                                pointer, 2 * DEFAULT_MEMORY_SIZE, DEFAULT_PRIORITY);

 if (ret != NX_SUCCESS) {
   return NX_NOT_ENABLED;
 }
```

life.augmented

Adding TCP/IP server (app_netxduo.c)

- Now let’s allocate some memory and enable further network protocols like ARP, ICMP, UDP and TCP
- … to be continued

```
/* Allocate the memory for ARP */
  if (tx_byte_allocate(byte_pool, (VOID **) &pointer, DEFAULT_MEMORY_SIZE, TX_NO_WAIT) != TX_SUCCESS)
  {
    return TX_POOL_ERROR;
  }

  /*  Enable the ARP protocol and provide the ARP cache size for the IP instance */
  ret = nx_arp_enable(&IpInstance, (VOID *)pointer, DEFAULT_MEMORY_SIZE);

  if (ret != NX_SUCCESS) {
    return NX_NOT_ENABLED;
  }

  /* Enable the ICMP */
  ret = nx_icmp_enable(&IpInstance);

  if (ret != NX_SUCCESS) {
    return NX_NOT_ENABLED;
  }

  /* Enable the UDP protocol required for  DHCP communication */
  ret = nx_udp_enable(&IpInstance);

  /* Enable the TCP protocol */
  ret = nx_tcp_enable(&IpInstance);

  if (ret != NX_SUCCESS) {
    return NX_NOT_ENABLED;
  }
```

life.augmented

Adding TCP/IP server (app_netxduo.c)

- And finally create DHCP client to receive IP address from the network router and a semaphore for signalling new IP address
- … to be continued

```
/* create the DHCP client */
 ret = nx_dhcp_create(&DHCPClient, &IpInstance, "DHCP Client");

 if (ret != NX_SUCCESS) {
   return NX_NOT_ENABLED;
 }

 /* create a semaphore used to notify the main thread when the IP address is resolved */
 tx_semaphore_create(&Semaphore, "App Semaphore", 0);
```

Adding HTTP server

Adding HTTP server (app_netxduo.c)

- We will instantiate the web server and provide callback for notification of web request
- … to be continued

```
 /* Allocate the server packet pool. */
 ret = tx_byte_allocate(byte_pool, (VOID **) &pointer, SERVER_POOL_SIZE, TX_NO_WAIT);

 /* Check server packet pool memory allocation. */
 if (ret != NX_SUCCESS) {
   Error_Handler();
 }

 /* Create the server packet pool. */
 ret = nx_packet_pool_create(&WebServerPool, "HTTP Server Packet Pool", SERVER_PACKET_SIZE, nx_server_pool,
SERVER_POOL_SIZE);

 /* Check for server pool creation status. */
 if (ret != NX_SUCCESS) {
   Error_Handler();
 }

 /* Allocate the server stack. */
 ret = tx_byte_allocate(byte_pool, (VOID **) &pointer, SERVER_STACK, TX_NO_WAIT);

 /* Check server stack memory allocation. */
 if (ret != NX_SUCCESS) {
   Error_Handler();
 }

 /* Create the HTTP Server. */
 ret = nx_web_http_server_create(&HTTPServer, "WEB HTTP Server", &IpInstance, CONNECTION_PORT, NULL, pointer,
                              SERVER_STACK, &WebServerPool, NX_NULL, webserver_request_notify_callback);

 if (ret != NX_SUCCESS) {
    Error_Handler();
 }
```

life.augmented

Adding HTTP server (app_netxduo.c)

- Let’s add main server thread that will start the whole stack up
- and finish the section *\/* USER CODE END App_NetXDuo_Init */*

```
/* Allocate the memory for main thread   */
if (tx_byte_allocate(byte_pool, (VOID **) &pointer,2 *  DEFAULT_MEMORY_SIZE, TX_NO_WAIT) != TX_SUCCESS)
{
  return TX_POOL_ERROR;
}

/* Create the main thread */
ret = tx_thread_create(&AppMainThread, "App Main thread", App_Main_Thread_Entry, 0, pointer,
  2 * DEFAULT_MEMORY_SIZE, DEFAULT_PRIORITY, DEFAULT_PRIORITY,
  TX_NO_TIME_SLICE, TX_AUTO_START);

if (ret != TX_SUCCESS) {
  return NX_NOT_ENABLED;
}
```

Adding HTTP server (app_netxduo.c)

- Let’s add the main thread in the section *\/* USER CODE BEGIN 1 *\/*

```
/**
* @brief  Main thread entry.
* @param thread_input: ULONG user argument used by the thread entry
* @retval none
*/
static VOID App_Main_Thread_Entry(ULONG thread_input)
{
  UINT ret;

  /* register the IP address change callback */
  ret = nx_ip_address_change_notify(&IpInstance, ip_address_change_notify_callback, NULL);

  if (ret != NX_SUCCESS) {
    Error_Handler();
  }

  /* start the DHCP client */
  ret = nx_dhcp_start(&DHCPClient);

  if (ret != NX_SUCCESS) {
    Error_Handler();
  }

  /* wait until an IP address is ready */
  if(tx_semaphore_get(&Semaphore, TX_WAIT_FOREVER) != TX_SUCCESS) {
    Error_Handler();
  }

  ret = nx_ip_address_get(&IpInstance, &IpAddress, &NetMask);

  /* print the IP address and the net mask */
  PRINT_IP_ADDRESS(IpAddress); PRINT_IP_ADDRESS(NetMask);

  if (ret != TX_SUCCESS) {
    Error_Handler();
  }

/* the network is correctly initialized, start the TCP server */
  /* Start the WEB HTTP Server. */
  ret = nx_web_http_server_start(&HTTPServer);

  /* Check the WEB HTTP Server starting status. */
  if (ret != NX_SUCCESS) {
    Error_Handler();
  } else {
    printf("HTTP WEB Server successfully started.\n");
  }

  /* this thread is not needed any more, we relinquish it */
  tx_thread_relinquish();

  return;
}
```

Adding HTTP server (app_netxduo.c)

- … then continue with the callback for IP address change (as a result of DHCP request)

```
/**
* @brief  IP address change call back
* @param ip_instance: NX_IP instance registered for this callback
* @param ptr: VOID * optional user data
* @retval none
*/
static VOID ip_address_change_notify_callback(NX_IP *ip_instance, VOID *ptr)
{
  tx_semaphore_put(&Semaphore);
}
```

- … and finish with the callback when HTTP server gets request for web page

```
UINT webserver_request_notify_callback(NX_WEB_HTTP_SERVER *server_ptr, UINT
request_type, CHAR *resource, NX_PACKET *packet_ptr)
{
  /*
   * At each new request we toggle the green led, but in a real use case this callback can
serve
   * to trigger more advanced tasks, like starting background threads or gather system info
   * and append them into the web page.
   */
  BSP_LED_Toggle(LED_GREEN);
  return NX_SUCCESS;
}

/* USER CODE END 1 */
```

Adding custom file system driver

life.augmented

Adding custom file system driver

- Open *nx_web_http_server.h* and uncomment the line

```
#define     NX_WEB_HTTP_NO_FILEX
```

- This macro allows you to define your own file interface or to generate arbitrary web content. You'll need to define following functions:

  fx_file_open, fx_directory_information_get, fx_file_close, fx_file_read, fx_file_write, fx_file_create, fx_file_delete.

  These functions are defined in the example file *filex_flash_stub.c* with web content stored in FLASH memory

- Open *filex_stub.h* and replace the FX_FILE_STRUCT with following definition:

```
typedef struct FX_FILE_STRUCT
{
   unsigned char *data;
   int len;
   int index;
} FX_FILE;
```

- The *FX_FILE* structure holds info about open file in the FLASH and allows its easy and safe reading

Adding web content

Adding web content

- No SD card, QSPI, NAND FLASH
- No external device with file system

➢ Internal FLASH will be used

  - Fixed content
  - Multiple files
  - Directory structure with filenames

➢ Let’s use Keil *FCARM.exe* utility to generate the C content from selected files:

```
fcarm.exe @filelist.txt
```

- The file *filelist.txt* contains the file list and commands for generating *web_data.c* (on one line):

```
~Cheatsheet_NetXDuo.html, ~index.html, ST.gif,
ST17223_Nucleo-H723ZG-frontside-scr.jpg,
ST20477_STM32Cube_RTOS_0221-scr.jpg TO
..\NetXDuo\App\web_data.c RTE NOPRINT
```

- The file paths in the list will be reflected in file names in the generated *web_data.c*, thus proper execution directory must be respected
- ‘~’ in front of file name means *don’t compress the whitespace*

life.augmented

Adding web content

- Access to the files in *web_data.c* is provided by a function

  uint32_t imageFileInfo (const char *name, const uint8_t **data);

- This function is generated by *fcarm.exe* inside *web_data.c* and returns the file size (or 0 if it isn't found) and pointer to its beginning in the memory.
- The file names are CRC'd and looked up in the file array quickly. Thus, directory paths and filenames must be precise (incl. capitalization)

Alternative NVM filesystems

- FNET TCP/IP stack: Apache 2.0 license, https://fnet.sourceforge.io/manual/how_to_generate_rom_fs.html
- GoAhead Web server: commercial and GNU GPL license, https://www.embedthis.com/goahead/doc/developers/rom.html
- QuantumLeaps QFSGen (from QTools): GNU GPLv2 license, https://www.state-machine.com/qtools/qfsgen.html

Evaluation

life.augmented

Evaluation of the workshop

- Please suggest improvements of the session!

life.augmented

Thank you

© STMicroelectronics - All rights reserved.
ST logo is a trademark or a registered trademark of STMicroelectronics International NV or its affiliates in the EU and/or other countries.
For additional information about ST trademarks, please refer to www.st.com/trademarks.
All other product or service names are the property of their respective owners.

life.augmented

Troubleshooting

- IP address not shown on the display
  - look up the ***App_Main_Thread_Entry***
  - add breakpoint at ***PRINT_IP_ADDRESS*** call
  - observe variable ***IPAddress***
- Ping can’t reach the board
  - Disable firewall
  - Make sure your notebook is connected to the router and has assigned IP address from the same range as Nucleo board

life.augmented