Micriμm

Demo of Everything

for the

Renesas YLCDRX63N

User's Manual

V3.0

Micriμm

For the Way Engineers Work

Table of Contents

Document Version

Version	Date	By	Description
1.0	2013/03/14	OD	Initial Version
1.1	2013/04/16	OD	Added information that was missing in some sections Added precisions about the behavior of the application
2.0	2013/08/09	MD	Initial addition of Phase 2 documentation.
2.1	2013/08/29	MD	Completed document review and updates.
3.0	2013/10/17	MD	Included IAR,e2studio setup, Code and RAM sizes

Micrium Software Component Versions Used in the Demo of Everything

Module	Version	Comment
µC/OS-III	V3.03.01	
µC/Clk	V3.09.03	
µC/CPU	V1.29.01	
µC/FS	V4.05.02	
µC/GUI	V5.18.00	
µC/LIB	V1.37.00	
µC/USBD	V4.01.01	
µC/USBH	V3.40.02	
µC/TCPIP	V2.13.02	
µC/DHCPc	V2.09.01	
µC/HTTPs	V2.00.00	
µC/CAN	V2.40.00	
µC/Probe	V2.30.00	

Website/support

For further information about any of our products, please refer to Micriµm's website (www.micrium.com) or contact us via our sales department at +1 954 217 2036 or email us at sales@micrium.com.

About

About Micriµm

Micriµm is the recognized leading provider of embedded software components for the embedded systems market. The company's flagship **µC/OS product family** is renowned for a variety of valuable features and benefits including unparalleled reliability, field proven performance, dependability, impeccable source code, and extensive documentation. In addition, some Micriµm components have been certified to meet rigorous safety-critical standards demanded by industries that include medical electronics, avionics, and industrial products.

Micriµm products are created for engineers by engineers. If you are new to Micriµm, contact us to discuss your project requirements and goals. Let us show you how our commercial RTOS family provides time-to-market advantages at a price you can afford. If you are familiar with the Micriµm family of products, call us to discuss your next design. We'll show you how our software components' advantages translate into substantial time and resource savings today and in the future.

About µC/OS-III

µC/OS-III (pronounced "Micro-C-O-S-Three") is a scalable, ROMable, pre-emptive real-time kernel that manages an unlimited number of tasks. µC/OS-III is a third-generation kernel and offers all of the services expected from a modern real-time kernel, such as resource management, synchronization, and inter-task communications. Importantly, µC/OS-III offers many unique features not found in other real-time kernels, such as the ability to execute performance measurements at run-time, to directly signal or send messages to tasks, pend on multiple kernel objects, and more.

About µC/GUI

With µC/GUI, you can create rich graphical user interfaces for your embedded applications. µC/GUI allows you to create different User Interfaces with an LCD display. It can be a simple 2D graphic screen using monochrome color up to smart-phone like display with 32 bits per pixel color and Alpha blending. With touch screen support activated, µC/GUI can help you design fantastic user interfaces for your product.

µC/GUI is both processor-independent and LCD controller-independent. It is compatible with single-task and multitask environments, and works with any commercial RTOS or your proprietary operating system. When purchased, µC/GUI is provided to you as "C" source code.

- Optimized for small size and high performance.
- Supports any 8/16/32-bit CPU
- Supports any type of LCD, with any controller (with appropriate driver)
- Configurable display size.
- Virtual display support; the virtual display can be larger than the actual display.
- Characters and bitmaps may be written at any point on the LCD, not just on even-numbered byte addresses.
- Supports caching the display in memory, for even more performance.
- Compile time switches allow for different optimizations.
- Includes µC/GUI-View for detailed view of all layers in the PC simulation.
- Configuration macros for any interface supported.

About µC/FS

µC/FS is a compact, reliable, high-performance and thread-safe embedded file system for microprocessors, microcontrollers and DSPs.

µC/FS can access multiple storage media through a clean, simple API. It supports the FAT file system for interoperability with all major operating systems. An optional journaling component provides fail-safe operation, while maintaining FAT compatibility.

µC/FS is based on clean, consistent ANSI C source code, with extensive comments describing most global variables and all functions.

The memory footprint of µC/FS can be adjusted at compile time based on required features and the desired level of run-time argument checking. For applications with limited RAM, features such as cache and read/write buffering can be disabled; for applications with sufficient RAM, enabling these features improves performance.

Device drivers are available for all common media types. Each of these is written with a layered structure so that it can easily be ported to your hardware. The device driver structure is simple, so that a new driver can be developed easily for a new medium.

About µC/USB-Device

µC/USB-Device supports several standard USB device classes (CDC, HID, MSC, PHDC and Audio V1). A Vendor class is also provided for developing vendor-specific USB devices. Thanks to a hardware abstraction layer, you can easily port µC/USB-Device to any new USB device controllers by simply modifying existing hardware access routines.

µC/USB-Device uses a modular architecture with three software layers between the application and the hardware.

- The device controller driver layer interfaces with the device controller to process interrupts, notify the device core of bus events, and receive/transmit packets.
- The device core layer responds to standard host requests during enumeration (the process by which a host learns the features of a device) and controls packet reception and transmission.
- The class layer provides functionality to the host using one or more class drivers. Each class driver responds to class-specific requests and may provide an API for controlling some features and receiving/transmitting information.

About µC/USB-Host

µC/USB Host is a real-time USB Host software stack designed for embedded systems equipped with a USB Host or OTG controller. It includes many USB class drivers (MSC, HID and CDC ACM). The stack requires a Kernel.

µC/USB Host uses a modular architecture with three software layers between the application and the hardware.

- The Class Driver layer provides class-specific services to the application. For example, the Mass Storage Class (MSC) Driver includes interface functions for reading and writing sectors from a storage device.
    - Note that a protocol may be required for certain classes. The Protocol Driver layer handles this aspect (e.g., the SCSI command set for the MSC).
- The Host core layer enumerates the device, loads a matching class driver, and provides the mechanism for data transfers.
- The Host Controller Driver (HCD) interfaces with the host controller hardware to enable data transfers and detect devices.

About µC/TCP-IP

µC/TCP-IP is a compact, reliable, high-performance TCP/IP protocol stack. Built from the ground up with Micriµm's unique combination of quality, scalability and reliability, µC/TCP-IP, the result of many man-years of development, enables the rapid configuration of required network options to minimize time to market.

The source code for µC/TCP-IP contains over 100,000 lines of the cleanest, most consistent ANSI C source code available for a TCP/IP stack implementation. µC/TCP-IP is implemented in ANSI C as it is the predominant language in the embedded industry. Over 50% of the code consists of comments and most global variables and all functions are described. References to RFC (Request For Comments) are included in the code where applicable.

About µC/DHCPc

DHCP is a protocol designed to enable clients to get IP configuration from a centralized database. This protocol has slightly evolved over the years from the BOOTP protocol initially designed to enable diskless clients to boot from the network. The µC/DHCPc module implements the mandatory parts of the following RFCs:

RFC 2131 ftp://ftp.rfc-editor.org/in-notes/rfc2131.txt
RFC 2132 ftp://ftp.rfc-editor.org/in-notes/rfc2132.txt
RFC 3927 ftp://ftp.rfc-editor.org/in-notes/rfc3927.txt

The first two describe the DHCP mechanism, and the third explains the dynamic configuration of link-local addresses (sometimes referred as Automatic Private IP Addressing, APIPA, or AutoNet in other environments).

About µC/HTTPs

HTTP is a protocol developed for the World Wide Web, and is implemented using TCP (Transmission Control Protocol).

µC/HTTPs is an HTTP server, and it implements the mandatory parts of the following RFCs:

RFC 1738: ftp://ftp.rfc-editor.org/in-notes/rfc1738.txt
RFC 1945: ftp://ftp.rfc-editor.org/in-notes/rfc1945.txt
RFC 2045: ftp://ftp.rfc-editor.org/in-notes/rfc2045.txt
RFC 2145: ftp://ftp.rfc-editor.org/in-notes/rfc2145.txt
RFC 2616: ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt

µC/HTTPs is designed to be used in a µC/TCP-IP, µC-FS and µC/OS-II or µC/OS-III environment.

About µC/CAN

µC/CAN is a CAN communication library, which simplifies the development of high-level CAN protocol layers like CANopen, DeviceNET or KWP2000. The library is designed to provide a high-level interface to CAN communication elements which is configurable and easy to use. Implemented using strict coding rules, the source code is highly efficient in resource utilization (RAM and ROM) and certifiable with a minimum effort.

About µC/Probe

µC/Probe is a Windows application that allows a user to display the value (at run-time) of virtually any variable or memory location on a connected embedded target. The user simply populates µC/Probe's graphical environment with gauges, tables, graphs, and other components, and associates each of these with a variable or memory location. Once the application is loaded onto the target, the user can begin µC/Probe's data collection, which will update the screen with variable values fetched continuously from the target.

Introduction

This section introduces and explains the requirements for using the Micrium multi-product demo application called the "Demo of Everything".

This application is built using several of Micriµm's products, to illustrate some of their capabilities. The application makes use of µC/OS-III, a real-time kernel; µC/GUI, a solution for embedded graphical user interfaces (GUI) and touch screens; µC/USB-Device, a USB device stack; µC/USB-Host, a USB host stack and µC/FS, an embedded file-system. It also makes use of Micriµm's building blocks, such as µC/CPU for CPU-dependant capabilities and µC/LIB, for memory management and string operations. µC/TCP-IP is incorporated for data transfer over an Ethernet network and µC/CAN for local network transfers as well. The application demonstrates how these products can be used together to create a complete application.

This document is separated in several sections: setting up and executing the application, a description of the modules and application files used and their respective roles as well as a general description of the way the application works.

Target

The application has been developed for the Renesas YLCDRX63N development board and the companion YDBENETPMOD extension board, based on the Renesas RX63N microcontroller (R5F563NEDDBG). Renesas's YLCDRX63N includes USB-Device, USB-Host, File System, and LCD hardware support utilized by several of the demos. The extension board adds hardware support for CAN and TCP/IP components.

Hardware Requirements

An external power source **MUST** be connected to the target in order for the application to work correctly. Therefore, a battery (if only using the YLCDRX63N) or the extension board's standard 12V power supply (included with YDBENETPMOD package) will be used. The battery must be connected to the J8 connector before using this demo. Power provided via the debugger or the USB function port is **NOT** enough to ensure correct functionality of the application, particularly the µC/USB-Host MSC demo. When the extension board is present, connect the power supply to J101 for the Ethernet and CAN demos to work properly.

YLCDRX63N comes with Segger J-Link Lite debugger. J-Link software installation of v4.76f or higher is required .This installer is provided at http://www.segger.com/j-link-rx.html.

Development Tools

The application has been developed using the Renesas e2 studio IDE, version 2.1.0.21, and using the RX Compiler (RXC), version 1.02 Release 01. It also runs with the IAR Embedded Workshop for RX tools version 2.42.2 or later.

Debugger Connection

Connect one end of the debugger ribbon cable provided to the header J7. Make sure pin 1 of the ribbon cable lines up with pin 1 of the header. The ribbon cable is marked with a different color (e.g. blue or red) to identify the pin 1. The header J7 has a marking on board that identifies the pin 1. Connect the other end of the ribbon cable to the J-Link Lite and connect the J-Link Lite to the PC.

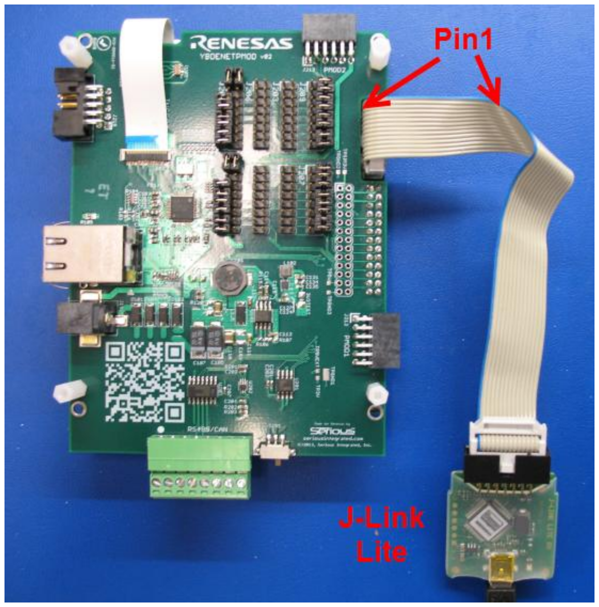


Figure 1 - Debugger Connections

Application

This section provides information about the installation, compilation and execution of this application.

Installation

Renesas e2 studio

Extract the e2 studio project from the .zip file to C:/<root> folder. It is recommended that you use a very short folder name (e.g. C:/DOE) if you cannot use the root folder. Open e2 studio and choose the workspace for the project. The workspace is located at:

(<Your_extraction_location>)\Micrium\EvalBoards\Renesas\YLCDRX63N\uCOS-III-GUI\e2studio-RXC\

Now the project will need to be built and compiled to the board. Right click the project and choose **"Build Project"** or (ctrl + B). Once it successfully builds click the debug button and select the **"uCOS-III_DemoOfEverything-LIB HardwareDebug"** label in the dropdown. If this selection is not present then select the "Debug Configurations" option. After selecting the Hardware Debug it will download to the board, click "Yes" when asked to change displays for the debugging display mode. Once it loads press F8 to run the project.

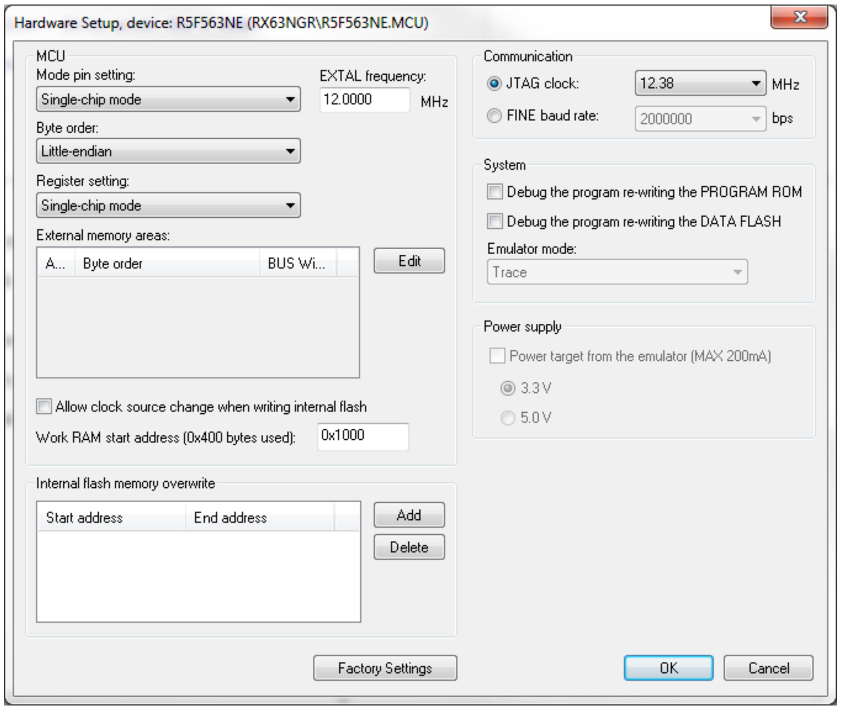
IAR Embedded Workbench for RX

Open the IAR workspace Project *uCOS-III_DemoOfEverything.eww* from the following directory: (<Your extraction location>)\Micrium\EvalBoards\Renesas\YLCDRX63N\uCOS-III-GUI\IAR

Connect to the board using the Segger J-Link Lite RX debugger.

Compile the project with F7, and then download it to the board with the download and debug button.

Before downloading the project to the board ensure that the following setup conditions are set:



Hardware Setup, device: R5F563NE (RX63NGR\R5F563NE.MCU)

MCU
Mode pin setting: Single-chip mode
EXTAL frequency: 12.0000 MHz
Byte order: Little-endian
Register setting: Single-chip mode
External memory areas:

A...	Byte order	BUS Wi...

☐ Allow clock source change when writing internal flash
Work RAM start address (0x400 bytes used): 0x1000

Internal flash memory overwrite

Start address	End address

Communication
● JTAG clock: 12.38 MHz
○ FINE baud rate: 2000000 bps

System
☐ Debug the program re-writing the PROGRAM ROM
☐ Debug the program re-writing the DATA FLASH
Emulator mode: Trace

Power supply
☐ Power target from the emulator (MAX 200mA)
● 3.3 V
○ 5.0 V

Figure 2 - IAR Hardware Setup

Once the board is connected press F5 to run the project.

Compilation

The demo is intended to serve as a stand-alone application and not used as a basis for other applications since there is limited remaining RAM on the target board.

The included pre-compiled library is used to build the demo application. This library is time-limited to one hour of use, after which a reset is required. When using the library, it is possible to modify the application files to experiment with the various Micrium products incorporated, but some of the changes made to the configurations may not take effect since the products have been compiled with a given configuration in the library. For example, any value changed in app_GUI_main.h will be taken into account (because it only affects application code) but a configuration changed in fs_cfg.h will not be taken into account by the µC/FS module contained in the library, only by the application. Also, be aware that a difference in the configuration values between the ones used by the application and the ones used to build the library can lead to other problems. For example, the FS_CFG_MAX_DEV_NAME_LEN define is used when building the library and within the application. If the value of this configuration is changed (from 15u to 30u, for example) and that a device's name is 20 characters long, it will provoke errors, since the µC/FS in the library will expect a name that is not more than 15 characters, but the application will not have any trouble using a 20-characters long name, since it is using a configuration value of 30. The library (in whole or in part) won't be used in another context than the current demo. It is not possible to use only a part of the library. For example, it is impossible to use µC/OS-III or µC/USB-Device in source code with the rest of the modules (µC/FS, µC/GUI, etc.) from the library.

Unlicensed Compilation

If the user does not have a license for the RXC compiler the .x or .mot file included in the /Micrium folder will need to be used. There are two different methods of flashing the project onto the YLCDRX63N board.

Using the .x

To use the uCOS-III_DemoOfEverything_LIB.x file it must be copied from the /Micrium folder and placed in the /Micrium/Software/EvalBoards/Renesas/YLCDRX63N/uCOS-III-GUI/HEW-RXC/uCOS-III_DemoOfEverything_LIB/HardwareDebug folder. e2 studio will recognize this file once it is in the folder and allow it to be loaded it to the board.

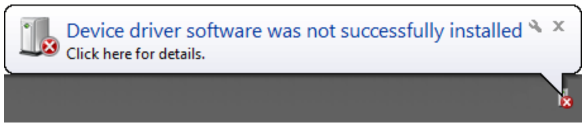
Using the .mot

To use the uCOS-III_DemoOfEverything-LIB.mot file, Renesas Flash Programmer (RFP) must be used. Please refer to RFP manual for more information.

Preparation

Driver installation

To perform the driver installation, use a USB cable to connect the PC to the J2 USB function port on the YLCDRX63N. The first time the device is connected to a host PC, the following error message will be displayed:



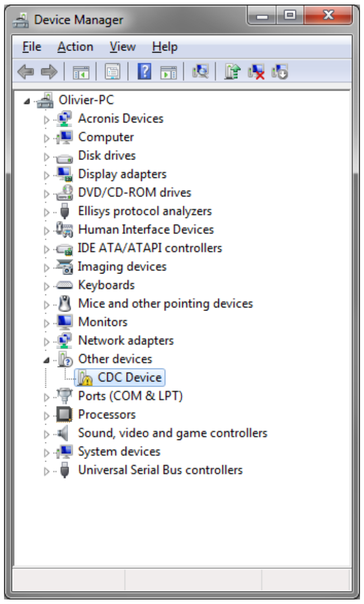
Device driver software was not successfully installed

Click here for details.

Figure 3 - Device Driver Installation

This is due to the fact that the driver for the CDC-ACM function of the USB device must be installed manually. The steps required are described below:

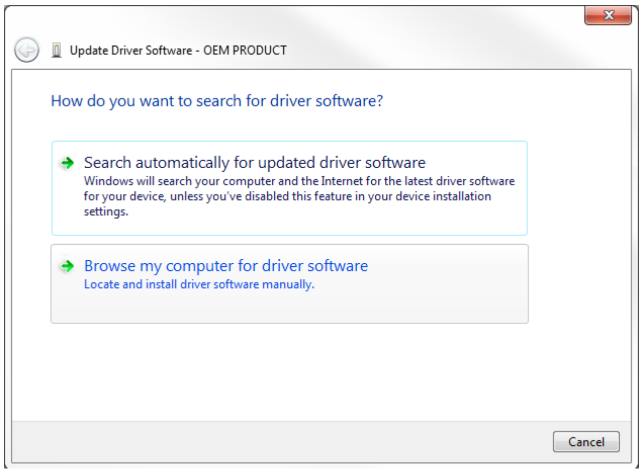
- Open Windows' Device Manager (Start Button > Control Panel > Hardware and Sound > Device Manager) and locate 'CDC Device', under 'Other devices'.



```
Olivier-PC
  Acronis Devices
  Computer
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  Ellisys protocol analyzers
  Human Interface Devices
  IDE ATA/ATAPI controllers
  Imaging devices
  Keyboards
  Mice and other pointing devices
  Monitors
  Network adapters
  Other devices
    CDC Device
  Ports (COM & LPT)
  Processors
  Sound, video and game controllers
  System devices
  Universal Serial Bus controllers
```

Figure 4 - Device Manager

- Right-click on 'CDC Device' and select 'Update Driver Software...', then 'Browse my computer for driver software'.


Update Driver Software - OEM PRODUCT

How do you want to search for driver software?

Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

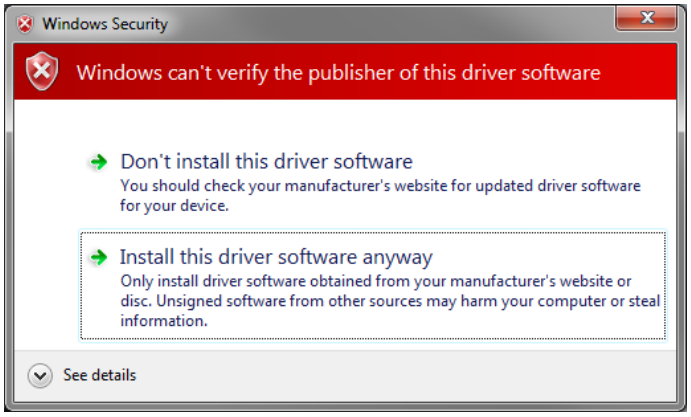
Browse my computer for driver software
Locate and install driver software manually.

Cancel

Figure 5 - Update Driver Software

- Specify the path to the INF file, by clicking the 'Browse' button. The file is located in the location: '*\Micrium\Software\EvalBoards\ Renesas\YLCDRX63N\uCOS-III-GUI\*'.

- Click 'Next' and in the following window, click 'Install this driver software anyway:


Windows Security

Windows can't verify the publisher of this driver software

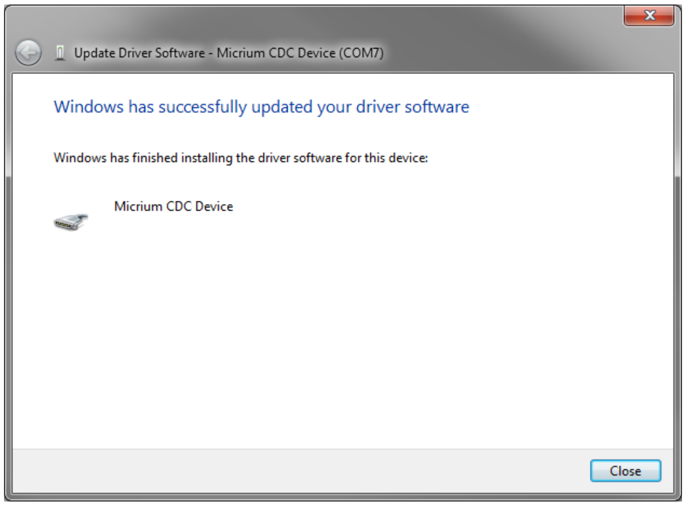
Don't install this driver software
You should check your manufacturer's website for updated driver software for your device.

Install this driver software anyway
Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.

See details

Figure 6 - Driver Installation Warning

- Windows will then install the INF file and the specified driver. This may take several seconds.
- Once the operation is done, the following screen will appear, confirming everything has been done correctly:


Update Driver Software - Micrium CDC Device (COM7)

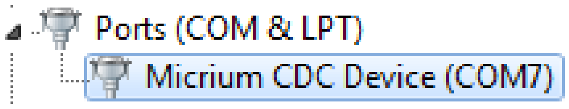
Windows has successfully updated your driver software

Windows has finished installing the driver software for this device:

Micrium CDC Device

Figure 7 - Driver Successfully Installed

- When closing this window, the ' 'CDC Device' in the Device Manager is now listed in Ports (COM & LPT) as 'Micrium CDC Device (COM#)'. Take note of the COM port number, as this is the port number required when opening a COM port.


Ports (COM & LPT)

Micrium CDC Device (COM7)

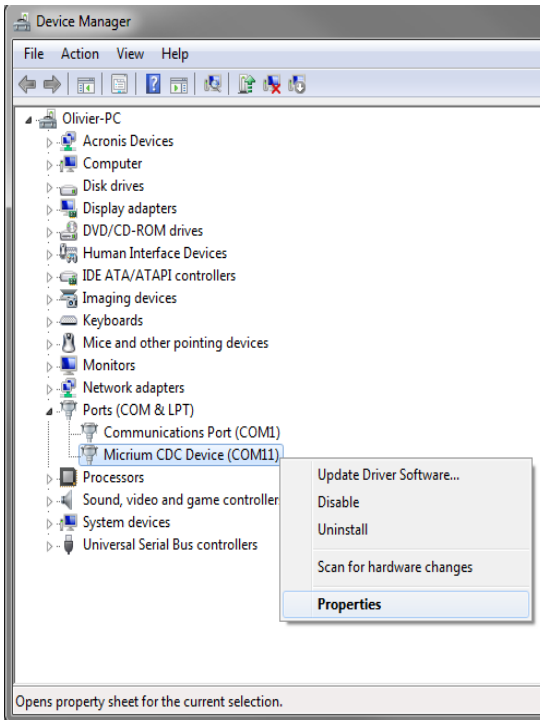
Figure 8 - Device Manager COM & LPT Ports

The COM port assigned by Windows depends on an individual PC and its usage. See the next section about how to change the COM port assignment.

Change COM Port Number (if required)

Some serial terminals may require that the COM port number allocated to the device by Windows be within a certain range (for example, Teraterm requires that the COM port number be between one and twelve). This section will describe the necessary steps to set a particular COM port number to the USB device:

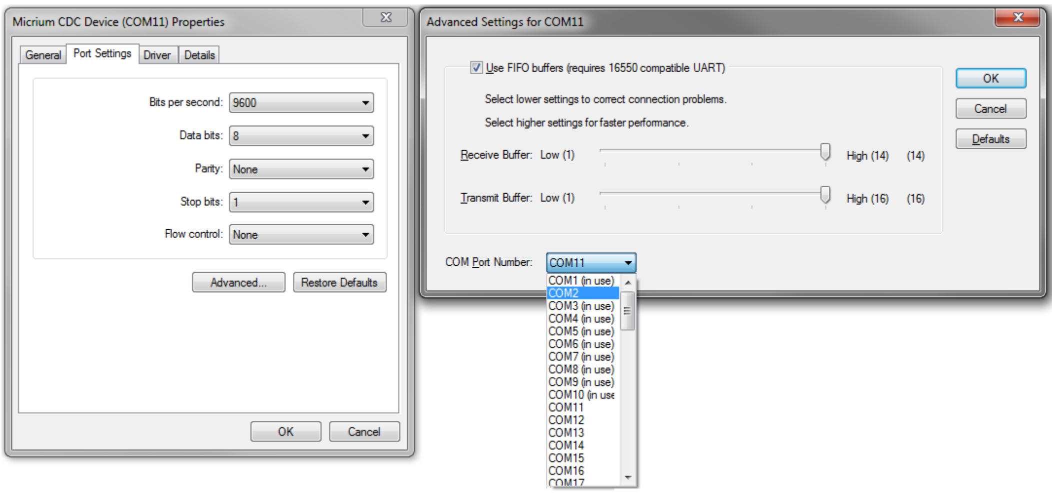
- In the Device Manager, right-click on the 'Micrium CDC Device' located under 'Ports (COM & LPT)' and select 'Properties'.



Opens property sheet for the current selection.

Figure 9 – CDC Device Properties

- In the 'Properties' window, under 'Port Settings' click on 'Advanced'.
- In the pop-up window, select the COM port number and click 'OK'.



Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

☑ Use FIFO buffers (requires 16550 compatible UART)

Select lower settings to correct connection problems.
Select higher settings for faster performance.

Receive Buffer: Low (1) High (14) (14)
Transmit Buffer: Low (1) High (16) (16)

COM Port Number: COM11

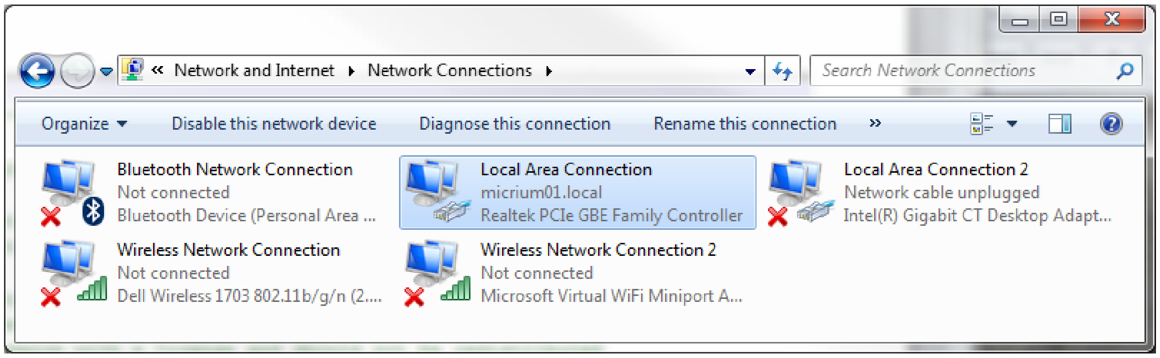
COM1 (in use)
COM2
COM3 (in use)
COM4 (in use)
COM5 (in use)
COM6 (in use)
COM7 (in use)
COM8 (in use)
COM9 (in use)
COM10 (in use
COM11
COM12
COM13
COM14
COM15
COM16
COM17

Figure 10 - Specify COM Port Number

- The COM port number of the device will now be changed.

Configuring DHCP Connection for Ethernet

To make sure the PC internet connection is compatible with DHCP and used by the Ethernet demo, confirm that it is not statically configured. Under the Control Panel settings navigate to **Control Panel->Network and Internet->Network Connections** and open the properties of the connection being used to plug into the board.


« Network and Internet ▸ Network Connections ▸

Search Network Connections

Organize ▾ | Disable this network device | Diagnose this connection | Rename this connection | »

Bluetooth Network Connection
Not connected
Bluetooth Device (Personal Area ...

Local Area Connection
micrium01.local
Realtek PCIe GBE Family Controller

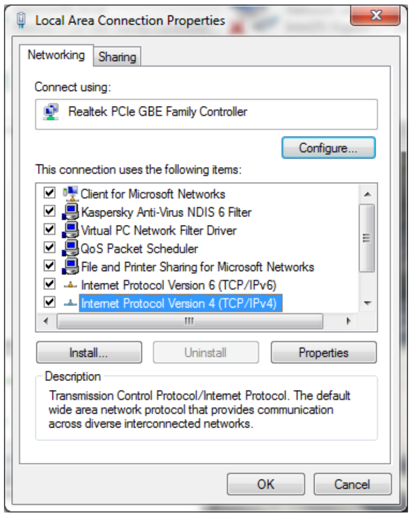
Local Area Connection 2
Network cable unplugged
Intel(R) Gigabit CT Desktop Adapt...

Wireless Network Connection
Not connected
Dell Wireless 1703 802.11b/g/n (2....

Wireless Network Connection 2
Not connected
Microsoft Virtual WiFi Miniport A...

Figure 11 - Location of Computers Internet Configurations

Under the properties page select the “Internet Protocol Version 4 (TCP/IPv4) and open the properties for this type of connection.


Local Area Connection Properties

Networking | Sharing

Connect using:
Realtek PCIe GBE Family Controller

Configure...

This connection uses the following items:
- ☑ Client for Microsoft Networks
- ☑ Kaspersky Anti-Virus NDIS 6 Filter
- ☑ Virtual PC Network Filter Driver
- ☑ QoS Packet Scheduler
- ☑ File and Printer Sharing for Microsoft Networks
- ☑ Internet Protocol Version 6 (TCP/IPv6)
- ☑ Internet Protocol Version 4 (TCP/IPv4)

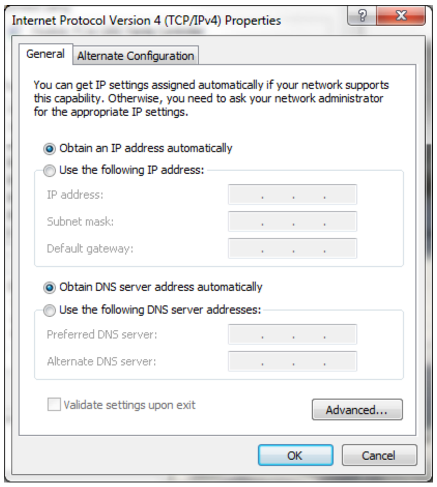
Install... | Uninstall | Properties

Description
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

OK | Cancel

Figure 12 - Connection Properties Window

Check the following settings so that the connection can be established automatically and does not attempt to use a pre-configured IP address.


Internet Protocol Version 4 (TCP/IPv4) Properties

General | Alternate Configuration

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

[✓ Obtain an IP address automatically]
Use the following IP address:
IP address:
Subnet mask:
Default gateway:

[✓ Obtain DNS server address automatically]
Use the following DNS server addresses:
Preferred DNS server:
Alternate DNS server:

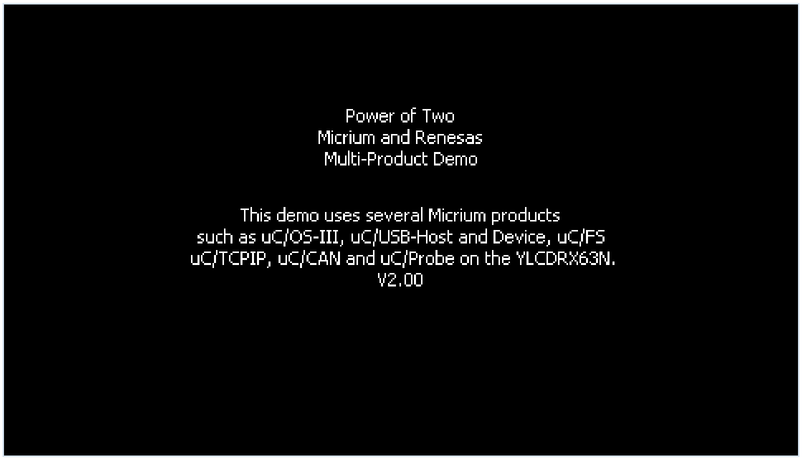
Validate settings upon exit

Figure 13 - IPv4 Properties

Save these connection settings for the Ethernet demo.

Running the Demo of Everything

The first screen displayed on the LCD is a black screen, displaying a title and a small description of the application in white letters.


Power of Two
Micrium and Renesas
Multi-Product Demo

This demo uses several Micrium products
such as uC/OS-III, uC/USB-Host and Device, uC/FS
uC/TCPIP, uC/CAN and uC/Probe on the YLCDRX63N.
V2.00

Figure 14 - Welcome Screen

After about five seconds, the main screen will appear. If the YDBENETPMOD extension board is attached at start-up the extended demo will be displayed. Otherwise, it will only display the demo options for USB-Host MSC, USB-Device HID, USB-Device CDC, and GUI. The demo buttons will be displayed, each with text indicating what demo will be started when the button is pressed.

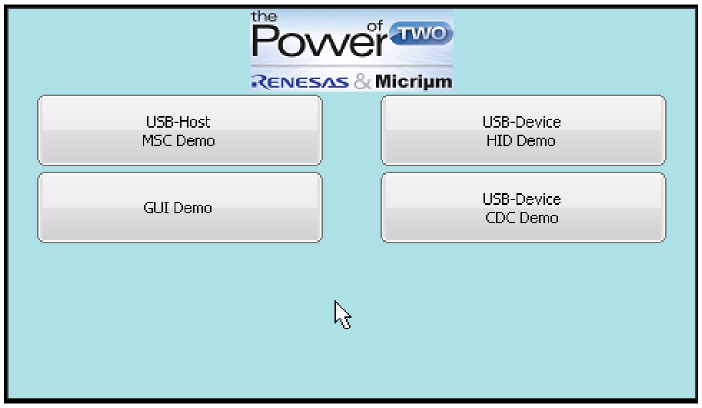


Figure 15 - Home Screen

USB-Host Mass Storage Class (MSC) Demo

To run the USB-Host MSC Demo, connect a USB memory stick to connector J1 on the YLCDRX63N. After pressing the USB-Host MSC Demo button, another screen will appear.

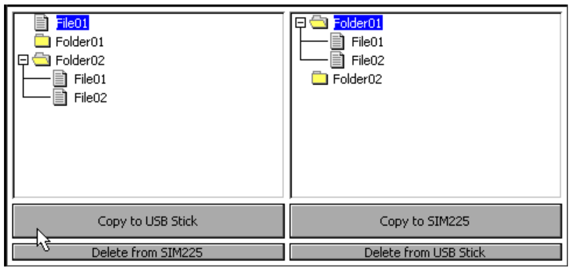


Figure 16 - MSC Demo Screen

This screen displays the files present in the flash memory of the YLCDRX63N board and on a USB memory stick. The files on the YLCDRX63N (if any) are displayed using the left treeview widget and the files on the USB memory stick (if any) are displayed using the right treeview widget.

Two buttons allow the user to copy a selected file from one device to the other. When copying a file, the complete folder path of this file will be replicated on the destination device. A pop-up window will appear, indicating that a copy is in progress. There are two other buttons which can be used to delete files or empty folders on the specified device.

To ensure this demo works as expected, avoid removing the USB memory stick when an operation is in progress. The USB memory stick must **NOT** be removed when the pop-up window is visible.

Also, make sure that the USB Memory stick connected to the board does not contain more than **four** levels of depth in its folder hierarchy. In the project, using source code, this limit can be increased via the `APP_CFG_FS_DIR_CNT` configuration value in `app_cfg.h`, but be aware that the stack usage will increase when more levels are explored. This is due to the fact that recursive functions are used to explore the tree of files and folders present on both devices. The level of recursions is limited to ensure that the amount of memory used stays within a reasonable limit.

It is impossible to copy a folder, even if it is empty, or to delete non-empty folders on any of the devices. A pop-up window will provide a warning to the user if these operations are attempted. It is also impossible to connect more than one USB memory stick at any given time (even if a HUB is used).

To return to the home screen, simply press the ‘Back to main’ button, in the upper right corner.

USB-Device Human Interface Device (HID) Demo

When pressing the USB-Device HID Demo button, another screen will appear.

Make sure a USB cable is connected both to the host PC and to the USB function port J2 on the YLCDRX63N board.

The screen displayed is a white rectangular area, which acts as a track pad (similar to the ones found on laptops). When the touch screen is pressed, the cursor displayed will ‘jump’ at the location pressed but the cursor on the PC will not move. This is normal. After this, moving the touched point on the touch screen will make the pointer on the host PC screen move accordingly. If the mouse pointer seems to ‘jump’ from one point to another instead of gliding smoothly, pressing a bit harder or on a smaller area (using the very tip of the finger or a stylus) on the touch screen will solve the issue.

Three buttons are located at the bottom of the screen. The left one acts as a left-click and the right one as a right-click. Pressing either one of them will act as if a mouse button was clicked. It is also possible to execute a “double-click” by pressing two times on the left-click button. Make sure to release the button before re-clicking on it. It is also possible to use the middle button to automatically perform a “double-click”. All the button events are triggered when the button is released.

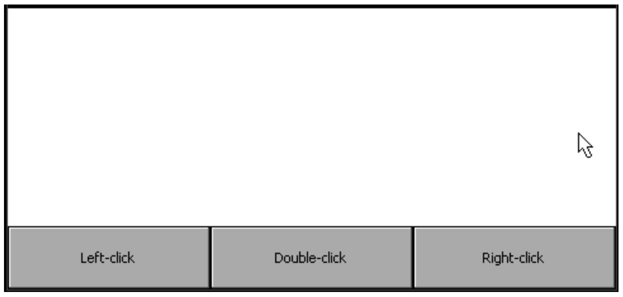


Figure 17 - HID Demo Screen

Pressing the ‘Back to main’ button in the upper right corner will exit this demo and return the application to the home screen.

USB-Device Communication Device Class – Abstract Control Model subclass (CDC-ACM) Demo

When pressing the USB-Device HID Demo button, another screen will appear.

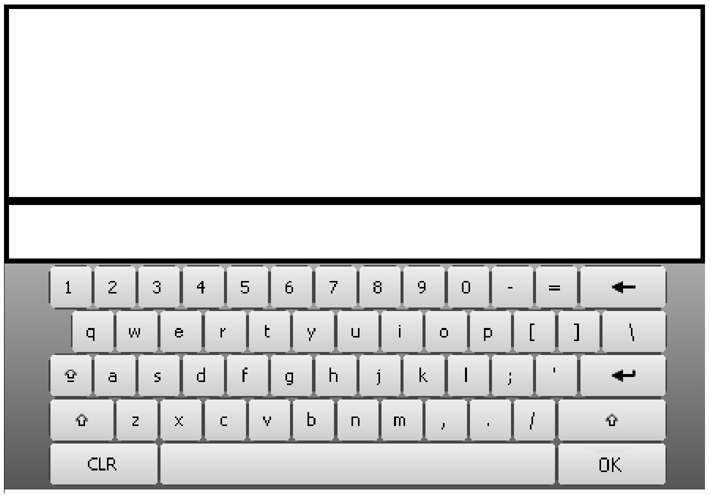


Figure 18 - CDC Demo Screen

Make sure a USB cable is connected both to the host PC and to the USB function port J2 on the YLCDRX63N board.

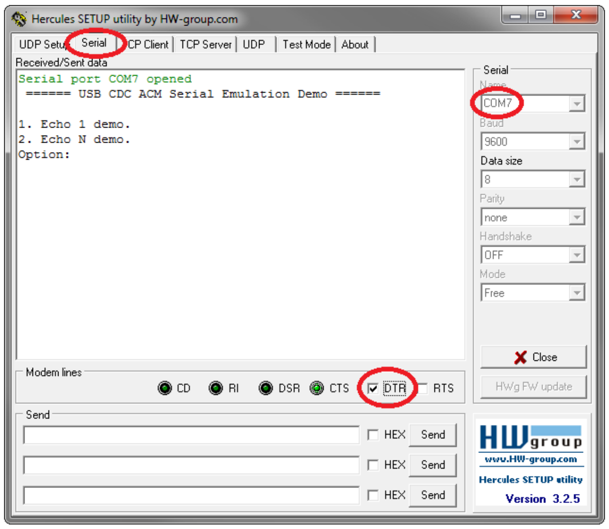
The screen displayed has three distinct sections: an upper, middle and lower one. The upper section is used to display characters received from the host PC terminal. Every character received from the PC will be displayed in this area, one after the other, without line breaks, but will wrap automatically to the next line when the current one is full. It is possible to send a carriage return via the serial terminal to manually go to the next line. If using the Hercules terminal emulator, this is done by sending the <lf> command. In Teraterm, this can be done by configuring the terminal to send "CR+LF" when line breaking. The middle area is used to display the characters that will be sent to the host PC terminal when the 'OK button will be pressed. The lower one is a keyboard that can be used to edit the character string that will be sent to the host PC terminal when the 'Send' button is pressed.

In order to use this demo, a serial terminal must be launched on the host PC. The terminal must be able to set the 'Data Terminal Ready' (DTR) property of the COM port. The terminal used in this example is the Hercules SETUP utility, but Teraterm can also be used, as it supports the DTR property automatically.

The default settings of the CDC-ACM function are the following:

- Baud rate: 9600;
- Data bits: 8;
- Parity bit: None;
- Stop bit: 1;
- No handshake.

The serial terminal must be configured with these settings. The COM port number can be known by looking in the Windows Device Manager, under Ports (COM & LPT). Make sure to enable the DTR signal of the serial terminal if the terminal does not handle it automatically. Local echo must be enabled if it is required to display the characters sent to the device on the PC terminal too.



```
Serial port COM7 opened
 ====== USB CDC ACM Serial Emulation Demo ======

1. Echo 1 demo.
2. Echo N demo.
Option:
```

Serial

Name: COM7

Baud: 9600

Data size: 8

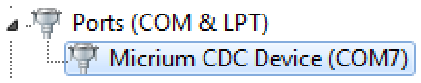
Parity: none

Handshake: OFF

Mode: Free

Modem lines: ● CD ● RI ● DSR ● CTS [✓ DTR] RTS

Figure 19 - Hercules Setup Screenshot



Ports (COM & LPT)

Micrium CDC Device (COM7)

Figure 20 - Device Manager COM & LPT Ports

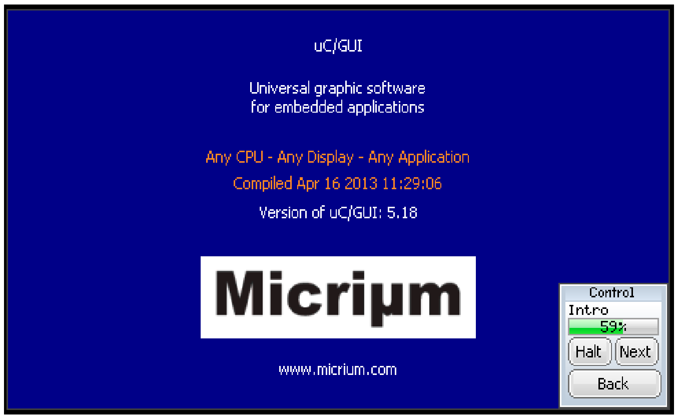
Once correctly configured, a message indicating ‘Device connected’ will be displayed in the reception area of the GUI screen. Characters can now be exchanged between the terminal on the host PC and the target device.

If the terminal is closed or the USB cable disconnected, a message indicating ‘Device disconnected’ will be displayed in the reception area.

It is possible to return to the home screen by pressing the ‘Back to main’ button in the upper right corner.

Graphical User Interface (GUI) Demo

When pressing the GUI Demo button for the first time, the following screen will appear. This is the intro window for the GUI demo, which contains four other examples, each showing a different widget and potential applications.


uC/GUI

Universal graphic software
for embedded applications

Any CPU - Any Display - Any Application
Compiled Apr 16 2013 11:29:06
Version of uC/GUI: 5.18

www.micrium.com

Figure 21 - GUI Demo Introduction Screen

At any time, it is possible to halt the demo by pressing the 'Halt' button, skip to the next example by pressing the 'Next' button and go back to the main menu by pressing the 'Back' button. All the buttons are located in the window in the lower right area of the screen. If the GUI demo is re-started after going back to the main menu, it will start from the next example that was supposed to execute.

In order, the examples are: BarGraph, Treeview, Listview and Graph. The BarGraph example shows the behavior of a bar graph, with the value of the bars varying and different colors used for different bars.

The Treeview example shows various ways the Treeview widget can be used and customized. For example, adding icons, hiding the lines, collapsing and expanding folders, etc.

The Listview example shows the built-in capabilities of the Listview widget. It demonstrates the sort mechanism, color fills, methods of using a selection, etc.

The Graph example displays Y axis vs. Time data on a graph. It first displays heartbeat-like data and then shows some sinusoid waveforms, both moving in the graph area.

Extended Demo Features

To connect the extension board, the “FFC” cable must be plugged into the slots on the YLCDRX63N and YDBENETPMOD as shown below:

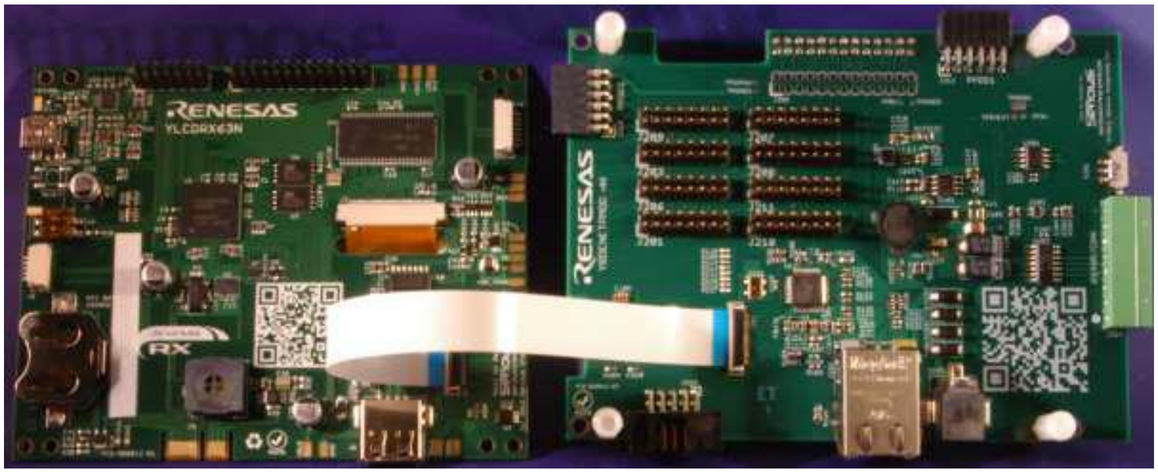


Figure 22 FFC Cable Connections Between the Boards

The cable must plug into each port with the coated blue section facing up away from the board so the exposed ends are touching the board connectors. Please make sure that the FFC cable slides into the connectors completely. For more detailed setup information refer to the YDBENETPMOD Technical Reference Manual.

With the extension board connected the main page will display several additional Demo buttons.

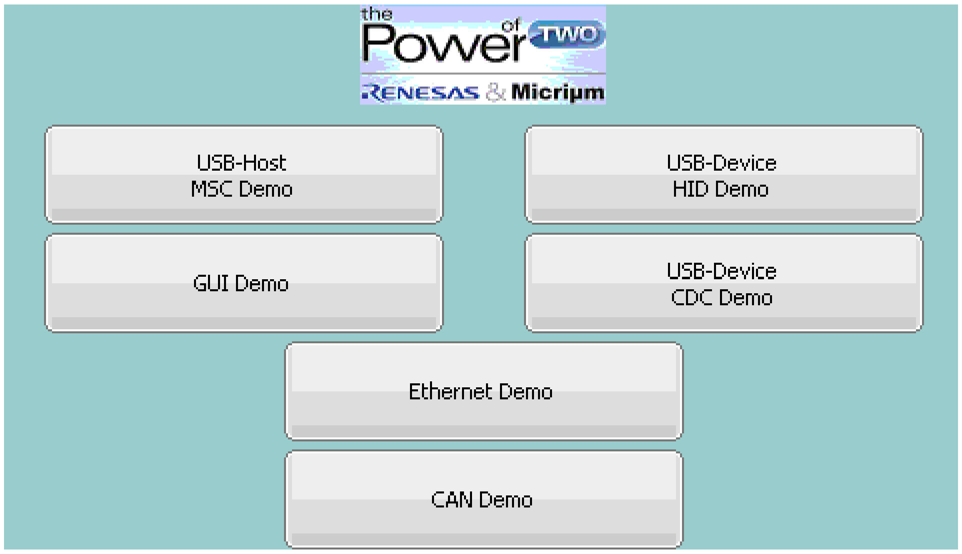


Figure 23 – Extended Home Screen

TCP/IP Ethernet Demo

The TCP/IP Ethernet Demo requires an Ethernet connection with DHCP compatibility. Ethernet Demo becomes available only when an Ethernet cable is connected to the YDBENETPMOD and to a user supplied router that is up and running. Several files will need to be transferred from a USB drive to the YLCDRX63N's Flash memory via the MSC Demo (explained above).

In the folder: *Micrium\Software\EvalBoards\Renesas\YLCDRX63N\uCOS-III-GUI\Webpages* there are five files needed for the demo:

- **index.html** (works on local connection with no internet access)
- **index2.html** (only needed if connecting with internet access)
- **jquery.form.js**
- **jquery.js**
- **logo.gif**

Copy these files to the USB memory stick and connect it to the YLCDRX63N board's USB port. Following the steps explained above for the USB-Host MSC Demo, copy the five files to the flash memory of the YLCDRX63N. These will be needed to load the webpage. Copy these five files to the flash, and then return to the 'Home Screen' via the "Back to main" button.

Open the Ethernet Demo to see the following page:

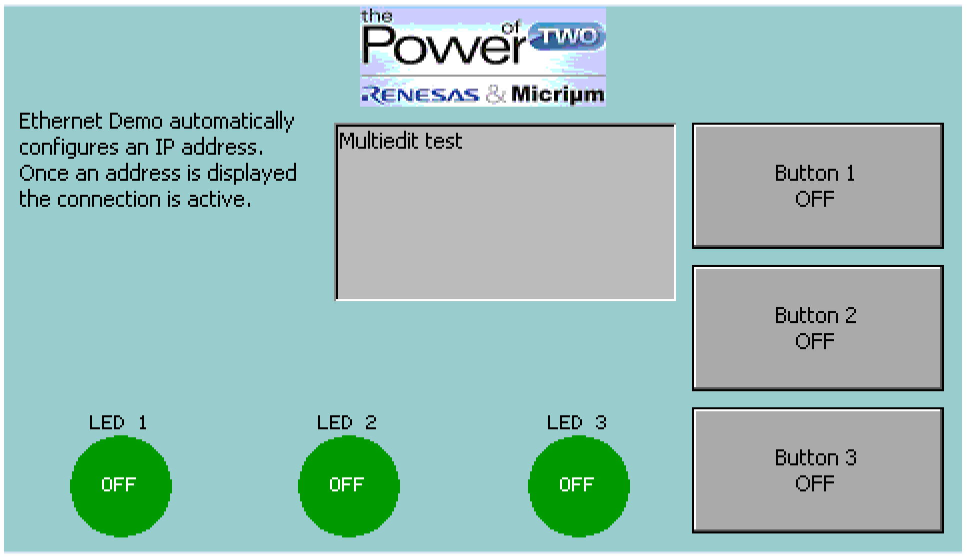
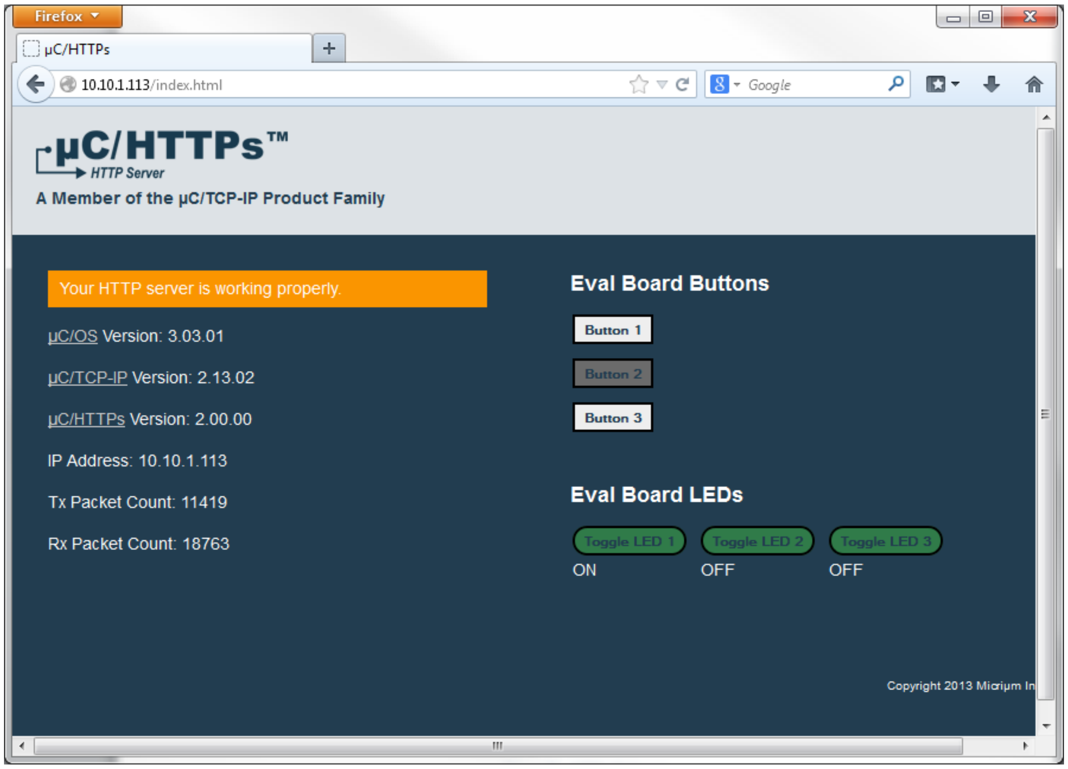

Ethernet Demo automatically configures an IP address. Once an address is displayed the connection is active.

Figure 24 - Ethernet Demo Window

The Statistics Window will display the type of connection currently active, the IP Address for the connection, how many Tx packets have been sent and how many Rx Packets have been received. When an IP Address is provided and the Rx packet number is increasing steadily, the connection is active. After the connection is stable the demo webpage can be opened by loading up the index.html file using µC/HTTPs.

To load the demo's webpage on the PC, start the web browser and enter the IP Address from the Statistics Window into the browser's URL bar followed by '/index.html' (e.g. 10.10.1.113/index.html). This will load the demo's webpage.

NOTE: When using a local connection with no internet access 'index.html' must be used as the webpage file. If an internet connection is available and used to connect with the board then 'index2.html' can be used as the web demo. 'Index.html' draws its resources from a local source in memory whereas 'index2.html' gets its resources from the source online.



Your HTTP server is working properly.

µC/OS Version: 3.03.01

µC/TCP-IP Version: 2.13.02

µC/HTTPs Version: 2.00.00

IP Address: 10.10.1.113

Tx Packet Count: 11419

Rx Packet Count: 18763

Copyright 2013 Micriµm In

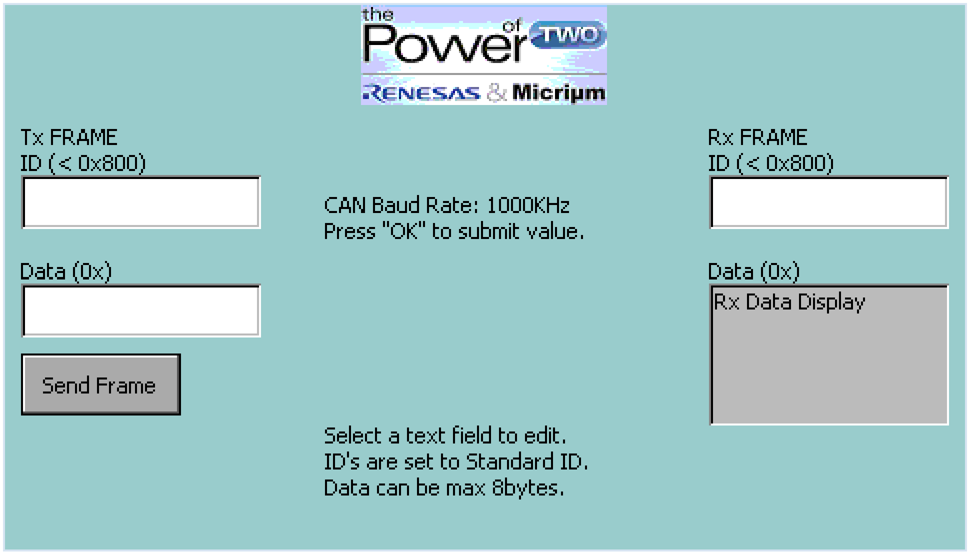
Figure 25 - HTTPs Webpage

Once the webpage has loaded, the Tx Packets counter in the Statistics Window and webpage will steadily increase. This indicates that the connection with the webpage is working properly and will interact with the YLCDRX63N. The webpage has three green "Toggle LED #" buttons which, when selected, will light up a corresponding LED Circle on the YLCDRX63N LCD display. Inversely, the three Gray buttons can be selected on the LCD display which will toggle the color of the three corresponding buttons on the webpage.

NOTE: The webpage demo has been tested with Google Chrome V30.0, Firefox V24.0 and Internet Explorer 9 running on Windows 7. Other web browsers, or other versions of Windows, may not work with the demo due to the implementation of Java, JSON and JQuery. Chrome and Firefox have proven to be more responsive running the demo than Internet Explorer.

Controller Area Network (CAN) Demo

Selecting the CAN Demo button will bring up the following display:



CAN Baud Rate: 1000KHz
Press "OK" to submit value.

Select a text field to edit.
ID's are set to Standard ID.
Data can be max 8bytes.

Figure 26 - CAN Demo Window

To run this CAN bus demo, an external CAN Analyzer/Sniffer device must be connected to the YDBENETPMOD extension board. If the CAN Analyzer/Sniffer is used with a PC, proper CAN software and the drivers must be installed The CAN demo is configured for Standard ID's, meaning they must be less than or equal to 0x7FF. The Data field can be a maximum of eight bytes (16 characters).

Several jumpers on the Extension board (YBDENETPMOD) will need to be connected at jumper J201. Make sure that pins 1-2, 3-4 and 11-12 are connected. Refer to the image below:

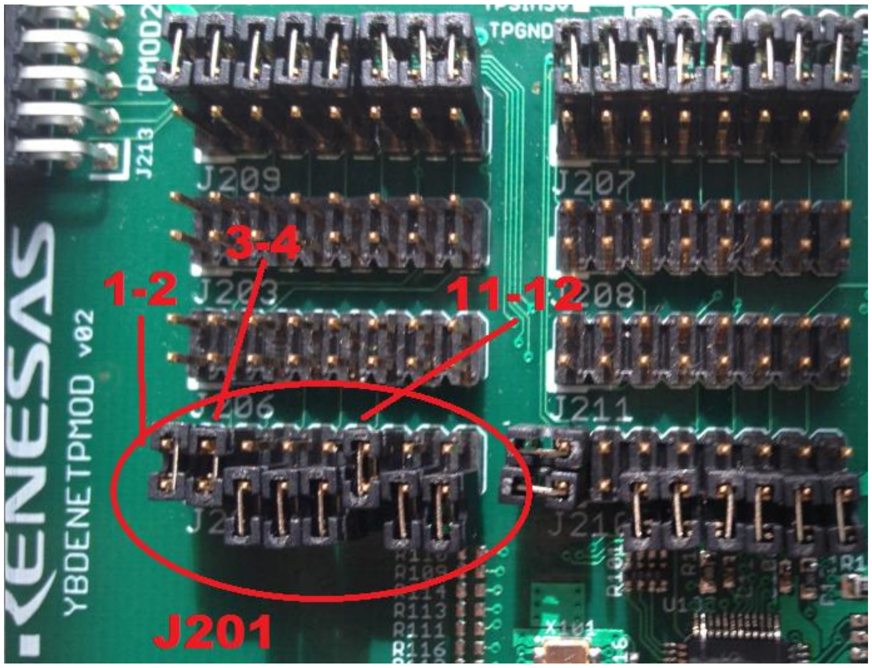


Figure 27 – Jumper J201 Settings for CAN Operation

The CAN High and Low signal lines need to be connected to allow for transferring and receiving data. See the image below for reference:

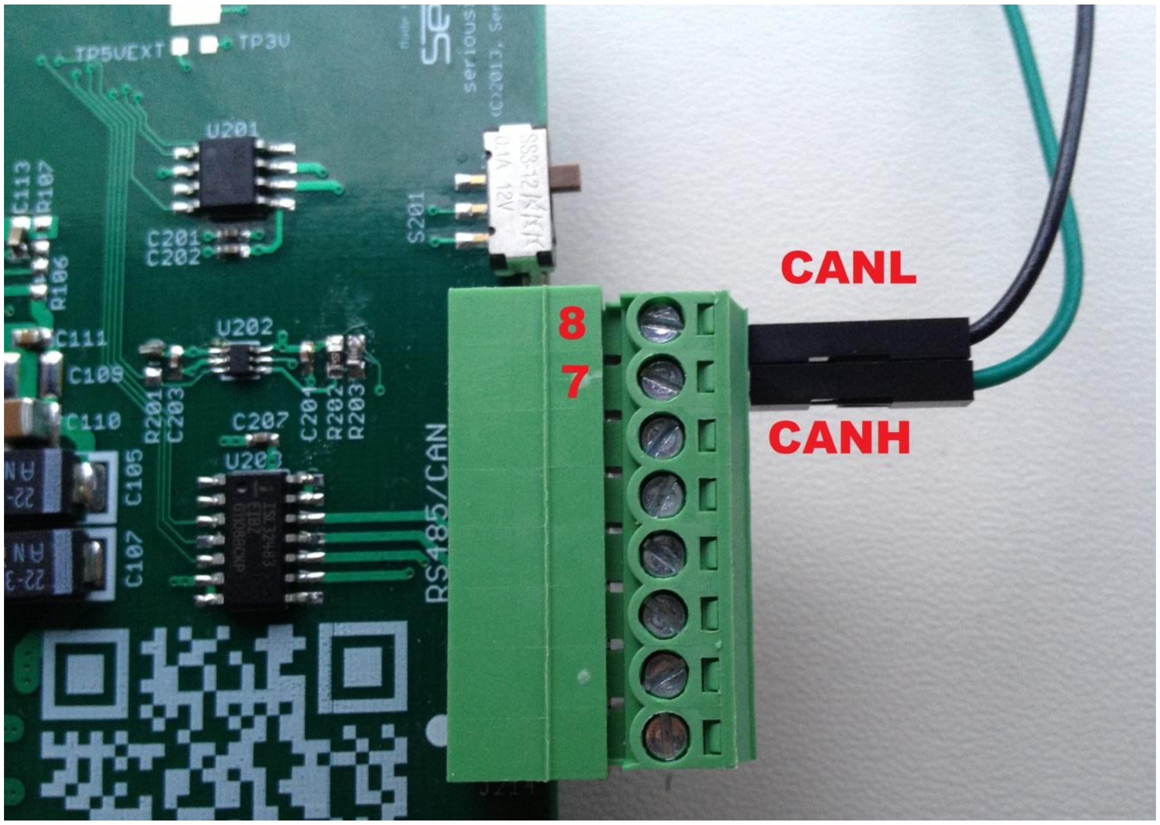


Figure 28 - Shows Connection of Signal Line 7-8 CANHigh CANLow

When connecting to the CAN bus it must be terminated at each end of the bus by a parallel 120Ω resistor to work properly. The following diagram gives a sample setup for connecting the CAN analyzer to the terminating resistor and CANH CANL connections on the board. The baud rate for sending messages is preconfigured to 1000 KHz for this demo.

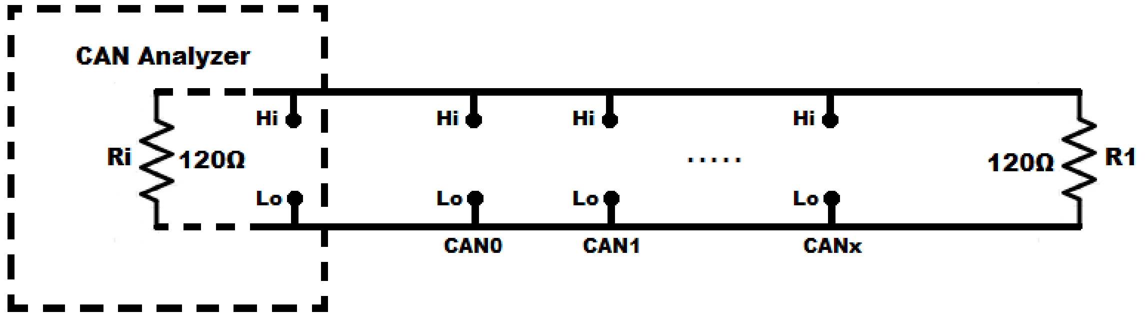


Figure 29 - Generic Setup for CAN Analyzer with Internal Resistance Ri and Terminating Resistor R1

To transmit a CAN bus data frame from the demo, enter a Tx Frame ID and the desired data into the two left fields shown on the LCD display.   Press “OK” to submit the values otherwise it will not be stored. After submitting those values, press the “Send Frame” button which will transmit the data frame to the CAN Analyzer/Sniffer. The ID must be within the allowed range (less than 0x800) otherwise nothing will be sent.

To receive a CAN bus data frame, the desired ID to be received **MUST** first be specified. After entering the desired ID in the right-side field shown on the LCD display, the CAN demo will search for a data frame with the ID specified. It will continue to search for that ID until a new ID is specified. As messages come in, the data will be displayed in the Data field separated into bytes 0-7 for ease of reading. Messages with ID’s that do not match the desired ID will not be displayed.

Using µC/Probe

There is no demo for Probe included in the YLCDRX63N package. However, it is possible to use Probe with the demo project. Probe can be configured to communicate to the target device via TCP-IP, so while this connection is active in the demo, Probe is usable.

After the project has been compiled there will be a .x ELF file in the */HardwareDebug* folder for the demo. To start analyzing variables within the demo, open Probe and in the symbol browser window select the "ELF File" button. There is also a pre-compiled .x file available in the \Micrium folder.

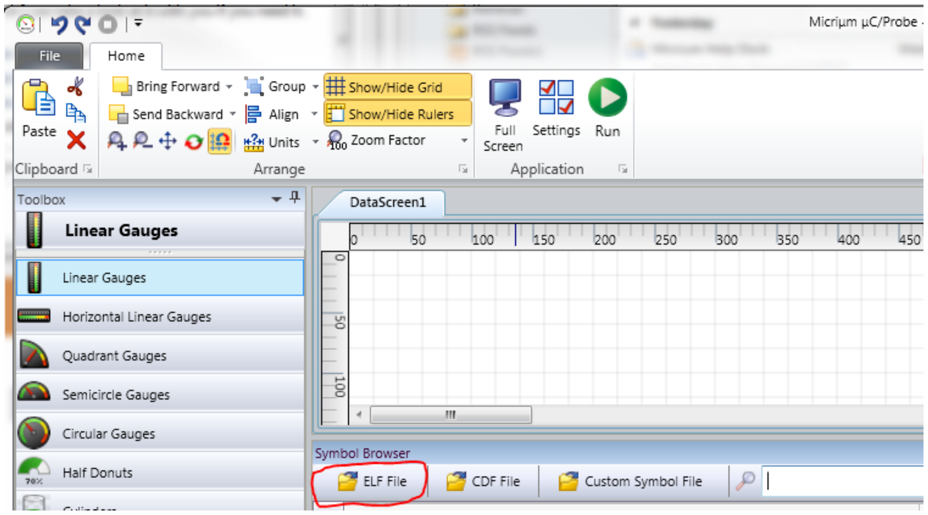


Figure 30 - ELF File Option for Loading Symbols into Probe

After loading the ELF file, click on Run button. For detailed instructions on the operation and usage of Probe consult the user's manual at http://micrium.com/tools/ucprobe/trial/.

NOTE: Due to RAM limitations on the target board, Probe's Kernel Awareness capabilities are not available in this demo. For more information please refer to the Appendix section of this document. Below Probe file shows the lower limit of the stack and how many octets are consumed.

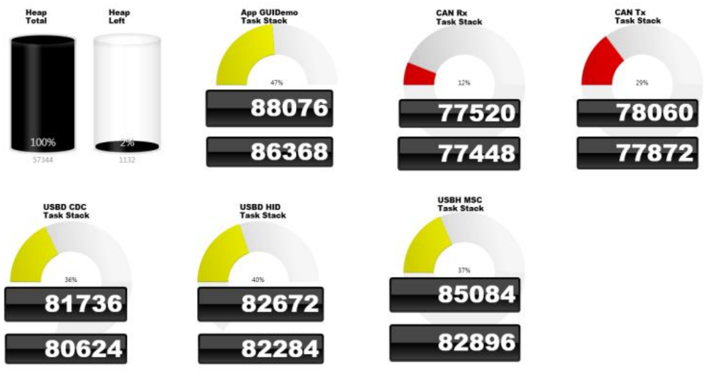


Figure 31 - Sample Probe Analysis of Task Stack Consumption

Code

Modules

This section provides a description of every module used in this application. For every module, there will be a general explanation about the role of the module and details about the major configuration values, their default values and what they affect. Configuration values in bold must not be changed to ensure proper execution of this application. Other configuration values can be changed, but may still cause problems if the value specified is not within a certain range of values (such as exceeding the remaining RAM space available). Changes done to certain configuration values will not be taken into account in the demo project, since it uses the pre-compiled library which was compiled with a given configuration.

Additional information about all the configuration values can be found either directly in the configuration file's comments or in the respective module's User's Manual.

µC/OS-III

Description

This module provides all the capabilities of a real-time kernel. It provides timing, scheduling and interrupt handling mechanisms. It also manages the creation and execution of tasks and allows the application to create and use several objects, such as semaphores, mutexes, queues and flags to sequence task execution. It also includes debugging and statistics capabilities that can be disabled and enabled at will.

os_cfg.h

Configuration of the capabilities of µC/OS-III.

Major configurations		
Name	Description	Value
`OS_CFG_PRIO_MAX`	Maximum number of task priorities level.	`30u`
`OS_CFG_STK_SIZE_MIN`	Minimal stack size that can be set.	`64u`
`OS_CFG_FLAG_EN`	Enable or disable the code for Event Flags.	`0u`
`OS_CFG_MEM_EN`	Enable or disable the code for Memory Management.	`0u`
`OS_CFG_MUTEX_EN`	Enable or disable the code for Mutex.	**`1u`**
`OS_CFG_Q_EN`	Enable or disable the code for Queues.	**`1u`**
`OS_CFG_SEM_EN`	Enable or disable the code for Semaphores.	**`1u`**
`OS_CFG_STAT_TASK_EN`	Enable or disable the code for the statistics task.	`0u`

Table 1 – os_cfg.h

os_cfg_app.h

Configuration of the properties of µC/OS-III internal tasks.

Major configurations		
Name	Description	Value
`OS_CFG_STAT_TASK_PRIO`	Priority of the statistics task.	`19u`
`OS_CFG_STAT_TASK_RATE_HZ`	Rate of execution of the statistics task.	`10u`
`OS_CFG_STAT_TASK_STK_SIZE`	Stack size of the statistics task.	`96u`
`OS_CFG_TICK_TASK_PRIO`	Priority of the tick task. High priority task.	`2u`
`OS_CFG_TICK_RATE_HZ`	Rate of execution of the tick task.	`1000u`
`OS_CFG_TICK_TASK_STK_SIZE`	Stack size of the tick task.	`96u`

Table 2 – os_cfg_app.h

µC/CPU

Description

This module provides CPU-related capabilities. It sets the data types of a specific CPU, the way the interrupts are handled and the critical sections. It also provides functions to handle the endianness of the CPU or its peripherals.

cpu_cfg.h

Configuration of µC/CPU.

Major configurations		
Name	Description	Value
`CPU_CFG_ENDIAN_TYPE`	Endianness of the CPU.	**`CPU_ENDIAN_TYPE_LITTLE`**

Table 3 – cpu_cfg.h

µC/FS

Description

This module provides a file system that can be used when interacting with the NOR flash located on the board or with the USB memory stick. It provides the user with an abstraction of the physical media and offers a simple API to interact with files and directories. In the app_cfg.h file are several FS configurations under the **uC/FS: DRIVER CONFIGURATIONS** header. These control some of the limitations we have set for this project in terms of maximum opened directories, max opened files etc.

fs_cfg.h

Configuration of the capabilities of µC/FS.

Major configurations		
Name	Description	Value
`FS_CFG_DIR_EN`	Enable or disable the directory module.	**`DEF_ENABLED`**
`FS_CFG_FILE_BUF_EN`	Enable or disable the file buffer support.	**`DEF_ENABLED`**
`FS_CFG_MAX_DEV_NAME_LEN`	Maximum length of the device name.	`15u`
`FS_CFG_MAX_FILE_NAME_LEN`	Maximum length of a file.	`255u`
`FS_CFG_MAX_PATH_NAME_LEN`	Maximum length of a path.	`260u`
`FS_FAT_CFG_LFN_EN`	Enable or disable the Long File Name support.	**`DEF_ENABLED`**

Table 4 – fs_cfg.h

app_cfg.h

File system relevant configurations in the app_cfg.h file.

Major Configurations		
Name	Description	Value
APP_CFG_FS_DEV_CNT	Max number of open devices	2u
APP_CFG_FS_VOL_CNT	Max number of opened volumes	2u
APP_CFG_FS_FILE_CNT	Max number of open files	3u
APP_CFG_FS_DIR_CNT	Max number of opened directories	4u
APP_CFG_FS_WORKING_DIR_CNT	Max number active working directories	5u
APP_CFG_FS_MAX_SEC_SIZE	Max sector size supported	2048u

Table 5 - app_cfg.h for Filesystem

µC/GUI

Description

This module provides a way to create and manage a Graphical User Interface (GUI) on an embedded system. It presents an API to the developer to make abstraction of the type of LCD and LCD controller, allowing the developer to focus on the functionalities of the GUI. It can easily be integrated in a multi-task environment, allowing several tasks to interact with it, without requiring additional protection mechanisms. It also provides several widgets, such as buttons, checkboxes, dropdown menus, graphs, tables, image containers, progress bars, scrollbars, spinboxes and text containers. Touch screen and mouse support can also be enabled.

GUIConf.h

Configuration of µC/GUI.

Major configurations		
Name	Description	Value
`GUI_OS`	Enable or disable multitasking support.	**`1u`**
`GUI_SUPPORT_TOUCH`	Support a touch screen.	**`2u`**
`GUI_WINSUPPORT`	Enable or disable window support.	**`1u`**
`GUI_DEFAULT_FONT`	Default font used by µC/GUI. If changed, other fonts must be manually included in the project.	`&GUI_Font6x8`
`GUI_MAXTASK`	Maximum number of tasks accessing µC/GUI.	**`5u`**

Table 6 – GUIConf.h

GUITouchConf.h

This file is required by µC/GUI but does not define or implement anything; it simply includes the LCDConf.h file.

LCDConf.h

Configuration of the LCD used by µC/GUI.

Major configurations		
Name	Description	Value
`LCD_MIRROR_X`	Enable or disable the mirroring on the X-axis.	**`0`**
`LCD_MIRROR_Y`	Enable or disable the mirroring on the Y-axis.	**`1`**
`XSIZE_PHYS`	Physical size along the X-axis.	**`480`**
`YSIZE_PHYS`	Physical size along the Y-axis.	**`272`**
`GUI_TOUCH_AD_RIGHT`	Value of the touch screen ADC for the rightmost point.	**`0x3D08`**
`GUI_TOUCH_AD_LEFT`	Value of the touch screen ADC for the leftmost point.	**`0x0397`**
`GUI_TOUCH_AD_BOTTOM`	Value of the touch screen ADC for the lowest point.	**`0x38D9`**
`GUI_TOUCH_AD_TOP`	Value of the touch screen ADC for the highest point.	**`0x0744`**

Table 7 – GUIConf.h

µC/LIB

Description

This module is used as a building block, on which several of Micriµm's product rely to function correctly. It provides utilities that can be used for memory management, character and string operations and mathematical operations.

lib_cfg.h

Configuration of µC/LIB.

Major configurations		
Name	Description	Value
`LIB_MEM_CFG_ALLOC_EN`	Enable or disable memory allocation module.	**`DEF_ENABLED`**
`LIB_MEM_CFG_HEAP_SIZE`	Available heap memory for memory allocation.	`56 * 1024L`

Table 8 – lib_cfg.h

µC/USB-Device

Description

This module provides a USB device stack. It can interact with the host to process the requests sent. It supports several USB classes, such as CDC-ACM and HID which are used in this demo. Its footprint can be optimized, depending on which classes are used and the complexity of the required configuration (number of endpoints, interfaces and configurations). In the app_cfg.h file are several USB-Device configurations under the **uC/USB-DEVICE DEMO CONFIGURATION** header. These control some of the limitations we have set for this project.

usbd_cfg.h

Configuration of µC/USB-Device.

Major configurations		
Name	Description	Value
`USBD_CFG_MAX_NBR_DEV`	Maximum number of USB devices	`1u`
`USBD_CFG_HS_EN`	Enable or disable USB high-speed.	**DEF_DISABLED**[1]
`USBD_CFG_MAX_NBR_CFG`	Maximum number of configurations.	**`2u`**
`USBD_CFG_MAX_NBR_IF`	Maximum number of interfaces.	**`3u`**
`USBD_CFG_MAX_NBR_EP_DESC`	Maximum number of endpoint descriptors.	**`6u`**
`USBD_CFG_MAX_NBR_EP_OPEN`	Maximum number of opened endpoints.	**`6u`**
`USBD_CFG_MAX_NBR_STR`	Maximum number of string descriptors.	**`10u`**
`USBD_HID_CFG_MAX_NBR_DEV`	Maximum number of HID class instances.	`1u`
`USBD_CDC_CFG_MAX_NBR_DEV`	Maximum number of CDC class instances.	`1u`

Table 9 – usbd_cfg.h

usbd_dev_cfg.h

This file has no configuration. It declares the device configuration structure and the device controller driver that will be used.

[1] The RX600 USB controller does not support high-speed.

µC/USB-Host

Description

This module provides a USB host stack. It supports several USB classes, such as MSC which is used in this demo and can interact with multiple devices displaying different classes at any time. Its footprint can be optimized, depending on which classes are supported and the number of devices it can support and their complexity (number of endpoints, interfaces and configurations). In the app_cfg.h file are several USB-Device configurations under the **uC/USB-HOST DEMO CONFIGURATION** header. These control some of the limitations we have set for this project.

usbh_cfg.h

Configuration of µC/USB-Host.

Major configurations		
Name	Description	Value
`USBH_CFG_MAX_NBR_DEVS`	Maximum number of devices.	**1u**
`USBH_CFG_MAX_NBR_CFGS`	Maximum number of configurations per device.	2u
`USBH_CFG_MAX_NBR_IFS`	Maximum number of interfaces per configuration.	2u
`USBH_CFG_MAX_NBR_EPS`	Maximum number of endpoints per alternate interface.	3u
`USBH_MSC_CFG_MAX_DEV`	Maximum number of MSC devices.	**1u**

Table 10 – usbh_cfg.h

usbh_hc_cfg.h

This file has no configuration. It only declares the host controller driver that will be used.

µC/TCP-IP

Description

This module provides a TCP/IP stack. It can be used to establish a communication network through an Ethernet connection. In the app_cfg.h file are several USB-Device configurations under the **uC/TCP-IP v2.0** header. These control some of the limitations we have set for this project such as the Rx and Tx queue size.

net_cfg.h

Network configuration file.

Major Configurations		
Name	Description	Value
`NET_IF_CFG_MAX_NBR_IF`	Sets max number of network interfaces.	`2u`
`NET_IP_CFG_IF_MAX_NBR_ADDR`	Set max number addresses per interface.	`1u`
`NET_TCP_CFG_NBR_CONN`	Configure total number TCP connections.	`5`
`NET_SOCK_CFG_FAMILY`	Configure socket family type.	**`IP_V4`**
`NET_SOCK_CFG_SEL_NBR_EVENTS_MAX`	Set max number of socket selections	`4 * 3`
`NET_SOCK_CFG_NBR_SOCK`	Configure total number of sockets.	`5`
`NET_CONN_CFG_NBR_CONN`	Configure total number of connections.	`10`

Table 11 – net_cfg.h

net_dev_cfg.h

There are no user defined configurations in this file. It only declares the Ethernet configuration devices for RX_Ether and PHY.

µC/DHCPc

Description

This module allows the user to use DHCP-client protocol within their project for obtaining an IP configuration. In the app_cfg.h file are several HTTPs configurations under the **uC/HTTPs v2.0** header. These control some of the limitations we have set for this project.

dhcp-c_cfg.h

DHCP Client configuration file.

Major Configurations		
Name	Description	Value
`DHCPc_CFG_IP_PORT_SERVER`	Configure DHCP server port.	**`67`**
`DHCPc_CFG_IP_PORT_CLIENT`	Configure DHCP client port.	**`68`**
`DHCPc_CFG_MAX_NBR_IF`	Configure maximum number of interfaces.	`1`

Table 12 - dhcp-c_cfg.h

µC/HTTPs

Description

This module provides an HTTP server stack. It allows the implementation of a web server. Web servers are commonly used to host websites that can be accessed via web browsers (Firefox, Chrome, Internet Explorer, etc.).

In this demo, the HTTP server is used to host a single webpage. This webpage can be accessed via a browser to view information and to interact with the demo application.

The demo application uses three main features of the HTTP stack. Dynamic Token Replacement replaces tokens in the html file with current version numbers for the µC/OS-III, µC/TCP-IP and µC/HTTPs products. CGI Post features allow the server to process incoming client HTTP POST requests to update the LEDs on the board. The last feature uses the server's capacity to send http response data from memory blocks and support GET requests to update specific sections of the html file. This feature uses JavaScript code along with the html, to refresh the buttons state on the webpage.

Please Note: The JavaScript code is not part of the Micriµm product line and has been incorporated primarily for the purposes of this demo. Any questions regarding the development of JavaScript code can be directed to the well-documented source available on the web. No support can be given by Micriµm on this matter.

http-s_cfg.h

This file regroups the general HTTP server suite configurations.

Major Configurations		
Name	Description	Value
`HTTPs_CFG_TOKEN_PARSE_EN`	Dynamic token replacement config.	**`DEF_ENABLED`**
`HTTPs_CFG_CGI_EN`	CGI processing feature config.	**`DEF_ENABLED`**
`HTTPs_CFG_CGI_POLL_EN`	CGI post polling feature config.	`DEF_DISABLED`
`HTTPs_CFG_CGI_MULTIPART_EN`	Multipart CGI processing feature config.	`DEF_DISABLED`
`HTTPs_CFG_CGI_FILE_UPLOAD_EN`	CGI file upload feature config.	`DEF_DISABLED`
`HTTPs_CFG_HDR_EN`	Header fields processing feature config.	`DEF_DISABLED`
`HTTPs_CFG_ABSOLUTE_URI_EN`	Absolute URI support feature config.	`DEF_DISABLED`

Table 13 - https-c_cfg.h (Note – Only first two configurations are needed for the Demo)

http-s_instance_cfg.c

This file defines several runtime configurations values needed for each HTTP server instance. Those configurations are regrouped inside an HTTPs_CFG structure.

Major Configurations		
Name	Description	Value
`ConnNbrMax`	Max number of simultaneous connections.	4
`BufLen`	Size of the HTTP buffers.	1000
`WorkingFolderPtr`	Working folder of the HTTP server	“\\” (root dir of FS)
`FileStr_DfltFilePtr`	Name of the default webpage	“index.html”
`HdrFieldEn`	Header fields feature config.	DEF_DISABLED
`CGI_En`	CGI processing feature config.	**DEF_ENABLED**
`CGI_MultipartEn`	Multipart CGI feature config.	DEF_DISABLED
`CGI_MultipartFileUploadEn`	CGI file upload feature config.	DEF_DISABLED
`TokenParseEn`	Dynamic token replacement config.	**DEF_ENABLED**

Table 14- http-s_instance_cfg.h variables

In addition, this file also defines callback functions used by the HTTP server instance. Those callback functions allow personalizing the HTTP server instance behavior to the needs of the demo application.

Name	Description
HTTPs_InstanceConnReq	Process the incoming get requests for the refreshing of the buttons’ states.
HTTPs_InstanceCGI_Post	Process the incoming post requests when a LED status has been change.
HTTPs_InstanceTokenValGet	Set the values of the dynamic tokens in the html file.

Table 15 - http-s_instance_cfg.h hook defines

µC/CAN

Description

This module allows the user to utilize inter module communication.

can_cfg.h

This file contains the CAN stack configurations for mailbox and message setup and operation.

Major Configurations		
Name	Description	Value
`CAN_MAILBOX_MODE`	Configure between Normal and FIFO.	**`NORMAL`**
`CAN_MODULE_CHANNEL_0`	Enable CAN channel 0 for use.	**`DEF_ENABLED`**
`CAN_MODULE_CHANNEL_1`	Enable CAN channel 1 for use.	`DEF_DISABLED`
`CAN_MODULE_CHANNEL_2`	Enable CAN channel 2 for use.	`DEF_DISABLED`

Table 16 - can_cfg.h defines

Application Files

This section provides a description of every application-specific file that is used in this application. These are the files located in the App/ directory, except for the configuration files already described in the Modules section. A description of the role of every application file will be given.

Header files

app_can.h

This file declares several functions, configuration defines and warning messages for invalid configuration settings with the CAN stack.

app_cfg.h

This file contains configuration for several parameters for the application. It defines the task priorities and stack sizes, the µC/USB-Device and µC/USB-Host class support configuration, the µC/FS media support, NOR driver and application-specific configurations, number of µC/TCP-IP Tx Rx buffers socket connections and other customizable features.

app_fs.h

This file defines default configuration for µC/FS, conditionally includes file-system specific header files and checks if the µC/FS configuration is valid.

app_GUI_can_demo.h

This file declares the CAN demo functions and several mode defines.

app_GUI_cdc_demo.h

This file declares the functions specific to the CDC demo.

app_GUI_hex_kbd.h

This file sets up the number of hex keyboard buttons, size and function declarations.

app_GUI_hid_demo.h

This file declares the functions specific to the HID demo.

app_GUI_kbd.h

This file sets up the basic keyboard buttons, size and function declarations.

app_GUI_main.h

This file defines configurations and structures used by several demos. It also declares the functions specific to the main screen.

app_GUI_msc_demo.h

This file declares the functions specific to the MSC demo.

app_GUI_net_demo.h

This file declares the majority of TCP/IP demo function.

app_net.h
This file declares several functions, configuration defines and warning messages for invalid configuration settings for the TCP/IP stack. This also includes a general demo structure for detecting specific connections for the demo.

app_usb.h
This file defines the default configuration for µC/USB-Device and µC/USB-Host. It also makes sure that the specified configuration is valid. It defines several configuration parameters and structures that are used by the USB tasks and declares the functions specific to the USB tasks.

Source files

app.c

This file contains the main function, which initializes µC/OS-III and creates the start task. It also contains the start task, which initializes all the other modules used by the application.

app_can.c

This file contains the initialization functions for µC/CAN. It also implements the required callback functions for the Tx and Rx tasks as well as pending/posting semaphores.

app_fs.c

This file contains functions used to initialize µC/FS. It contains an initialization function which starts µC/FS and calls the initialize functions for the application-specific drivers (NOR and MSC).

app_GUI_can_demo.c

This file contains everything that is required for the CAN demo GUI screen. It defines the sizes and positions of the widgets displayed and implements everything required to capture and send pertinent touch screen events to the CAN Rx/Tx tasks, including the callback function used to process the events. It also implements the initialize, suspend, resume and notify functions for the CAN demo.

app_GUI_cdc_demo.c

This file contains everything that is required for the CDC-ACM demo GUI screen. It defines the sizes and the positions of the widgets in the CDC-ACM demo screen and implements everything required for the keyboard to work correctly, including some drawing functions and the callback function used to process the events. It also implements the initialize, suspend, resume and notify functions for the CDC-ACM demo.

app_GUI_hid_demo.c

This file contains everything that is required for the HID demo GUI screen. It defines the sizes and positions of the widgets displayed and implements everything required to capture and send pertinent touch screen events to the HID USB task, including the callback function used to process the events. It also implements the initialize, suspend, resume and notify functions for the HID demo.

app_GUI_main.c

This file contains everything that is required for the home screen GUI. It defines the sizes and positions of the widgets displayed and implements everything required to capture and manage the touch events on a given button, including the callback function used to process the events. It also implements the initialize, suspend, resume and notify functions for the home screen. It also contains the main GUI task, which processes the GUI execution and checks if a task must be notified of any external event.

app_GUI_msc_demo.c

This file contains everything that is required for the MSC demo GUI screen. It defines the sizes and positions of the widgets displayed and implements everything required for the correct handling of copy requests, including the callback function used to process the events. It sends a request to the USB host MSC task when required. It also implements the initialize, suspend, resume and notify functions for the MSC demo.

app_GUI_net_demo.c

This file contains everything that is required for the Ethernet demo GUI screen. It defines the sizes and positions of the widgets displayed and implements everything required to capture and send pertinent touch screen events to the TCP/IP Ethernet tasks, including the callback function used to process the events. It also implements the initialize, suspend, resume and notify functions for the Ethernet demo. The functions that control the pressing of the on-screen LEDs and toggle buttons are also controlled here.

app_net.c

This file contains the initialization functions for µC/TCP-IP, µC/HTTPs and µC/DHCPc. It also implements the required callback functions for the Ethernet demo. The monitoring function for the DHCP connection and obtaining of the current IP address function is included here.

app_usb.c

This file contains the initialization functions for µC/USB-Device and µC/USB-Host. It also implements the required callback functions for µC/USB-Device and defines the structure for the two internal tasks of µC/USB-Host.

app_usbd_cdc_demo.c

This file contains the initialization function for the CDC-ACM class. It also implements the CDC-ACM task, used to send and receive characters to or from the host PC. It also defines the required CDC-ACM callback functions.

app_usbd_hid_demo.c

This file contains the initialization function for the HID class. It also implements the HID task, used to send information to the host PC. It also defines the required HID callback functions.

app_usbh_msc_demo.c

This file contains the initialization function for the MSC class. It also implements the MSC task, used to exchange data between the host PC and the device. It also defines and registers the required MSC callback function.

GUI_X_uCOS-III.c

This file provides an abstraction layer between µC/OS-III and µC/GUI. It allows µC/GUI to use several µC/OS services, such as semaphores and delay functions to implement a lock, a delay and a timing mechanism.

GUIConf.c
This file initializes the GUI and declares and assigns the memory dedicated to µC/GUI.

LCDConf.c
This file checks the validity of the configuration of the LCD and implements several functions related to the GUI and the LCD. These functions include a function to initialize the LCD and the touch screen, the touch screen task and several functions used by the touch screen task.

usbd_dev_cfg.c
This file sets the content of the two configuration structures for µC/USB-Device, the device configuration and the device driver configuration structure. The device configuration structure contains the vendor and product ID and other configurable fields, such as the manufacturer and product string. The device driver configuration structure must not be edited for this given application; it contains information about the RX63 USB device controller.

usbh_hc_cfg.c
This file sets the content of the configuration structure for µC/USB-Host, the device driver configuration structure. The device driver configuration structure must not be edited for this given application; it contains information about the RX63 USB host controller.

Tasks

Description

This section will explain the role of every application-defined task. The internal tasks will not be detailed here, please refer to the User Manual of the product for more details on these internal tasks.

External Memory Manager Task

This task manages the access to the external memory. It suspends the registered task(s) (registered by calling `R_DDLCD_ExMemoryAcquire()`) when the LCD is reading the external memory and that nothing can be written in it. It then resumes the tasks when they are free to interact with the external memory: when the LCD is not reading in it. The external memory is used to store everything which is GUI-related. It contains the widgets, their properties and a rendering of the screen(s). It does not contain any execution code.

This protection mechanism is due to the fact that the Direct Drive technique is used to interact with the LCD. For more information about the Direct Drive LCD technique and more explanations as to why this protection mechanism is required, please refer to Renesas' Application Note Direct Drive LCD Demonstration, labelled R01AN0331EU0103.

In the current application, only the GUI Main task needs to be registered, since it is the only one which accesses the external memory, via the `GUI_Exec()` function and the callback's it calls. The Demo window creation and the `Suspend()`, `Resume()` and `SemNotify()` functions are all called by the GUI Main Task, to only have to register a single task as accessing the external memory.

µC/GUI Touch Task

This task reads the values from the touch screen ADCs and stores it for the GUI to interpret the values obtained. This is done every 10ms.

µC/USB-Device CDC Demo Task

This task checks if a CDC-ACM device is present and correctly configured. If a device is present, it checks if a transfer (IN or OUT) has been requested. If the transfer is a transmission, it sends the data prepared in the CDC GUI functions. If the transfer is a reception, it stores the data received in a shared buffer and posts a semaphore allowing the CDC GUI to be notified via the registered `SemNotify()` function. This task is never executed at the same time than the GUI demo, HID or MSC task, since it is only active when the CDC Demo is selected.

µC/USB-Device HID Demo Task

This task checks if a HID device is present. If a device is present, it checks if an IN transfer has been requested. If the transmission is requested, it sends the data prepared in the HID GUI functions. This task is never executed at the same time than the GUI demo, CDC or MSC task, since it is only active when the HID Demo is selected.

µC/USB-Host MSC Demo Task

This task checks if a copy operation (between the MSC and the NOR) has been requested. If it has, it manages the string operations required to create the full path of the files and executes the copy operation(s) requested. There may be more than one operation if directories must be created. This task is never executed at the same time than the GUI demo, CDC or HID task, since it is only active when the MSC Demo is selected.

GUI Demo Task

This task will execute a demo that shows various widgets, their uses and the customization options available in µC/GUI. When started (or resumed), it registers itself as a task that accesses the external memory and calls in turn every demo functions. It also verifies the states of the three buttons ('Halt', 'Next' and 'Back') and manages the execution depending on their states. When suspended, it un-registers itself from the external memory manager.

GUI Main Task

This task begins by registering itself as a task that accesses the external memory. It then creates all the GUI windows, for every demo, including the 'welcome' and 'home' screens. It then periodically call the `GUI_Exec()` function, allowing the GUI to process events, (such as touch screen events), and call the corresponding callback function. Since in each demo everything related to the GUI is done in callback functions (there are no other tasks related to the GUI for the various demos), the GUI is exclusively updated when this task executes, thus making this task the only one needing to be registered as accessing the external memory, except when the GUI demo is running, it leaves the GUI demo task to call `GUI_Exec()` by itself. The task then calls the `GUI_Mgr()` function, which checks if one of the registered notification semaphores, for itself and the currently executing task, has been posted. If it has, it calls the corresponding `SemNotify()` callback. It is in its own `SemNotify()` callback that this task executes the switch from one task to another, by calling the respective `Suspend()` and `Resume()` callback functions of the task to stop and start executing. It then toggles the LED, depending on which of the demo is executing.

DHCPc Task

This task is used by the DHCPc module to configure an IP address for the user at start-up or during the runtime of the demo when a disconnected connection is re-connected.

HTTPs Task

This creates the HTTP server task that manages the connection to the webpage and watches for changes or updates. This is used in conjunction with the µC/TCP-IP stack to connect to the specified server that will be allocated from the DHCPc task. The user can then load the demo html file and view the webpage that will allow them to modify several lights on the board and webpage.

CAN0 Rx Task

This task configures the specified (Mailbox 0 in this demo) mailbox to receive messages with a particular ID. This filtering allows the CAN stack to locate specific messages from particular components.

CAN Tx Task

This task sets up a form to be transmitted via the CAN stack. It will take the users specified ID and Data string and transmit it along the communication lines when the user presses the "Send Frame" button on the LCD.

Priorities

This section will explain the way the µC/OS-III task priorities used in this project have been set.

The external memory manager task needs to be of higher priority than the tasks it must suspend. Also, since it executes very rapidly and is very critical, it was decided to put this task at the highest possible priority.

The GUI Touch task has the next priority. This is a task that must be executed often, but does not need much time to execute itself, since the processing of the events is done by the GUI task. Therefore, it was decided that this task must have a higher priority than the demo task.

Three internal tasks are the following tasks with the higher priority. There are two that are related to USB-Host, and then the USB-Device core task. These tasks must habitually be a middle-high priority. In the current case, they have a higher priority than the GUI tasks, but not more than critical tasks (GUI Touch and external memory handling tasks).

The clock task is the next highest priority, since it is more time-critical than the other ones, is quite fast to execute and pends on a one second timer, leaving time for other tasks to execute.

Then, there are two demo tasks: CDC and HID. These must have a lower priority than the USB-Host and USB-Device internal tasks, but a higher than the GUI task. There are delays in all of these tasks to allow the main GUI task to execute itself.

The internal USB-Device HID timer task is next. It is a fast and low-priority task internally used by the HID class of µC/USB-Device.

The task with the next priority is the main GUI task. It must be at a high enough priority to execute often enough, in order to keep the GUI responsive. This is why it has a higher priority than the MSC task, so the GUI still updates, even when a file transfer is in progress.

The GUI demo task is the next one. Since this task never executes at the same time as the CDC, HID or MSC task, its priority is more flexible than the other task's. It can be a bit higher or a bit lower.

The MSC task is the application task with the lowest priority, since transfers between the NOR memory and the USB memory stick can be long and must not prevent other tasks from executing when they are in progress.

The Net Tx priority must not be higher than the tasks monitoring the GUI display or touch sensor but need to be higher than the net timer task.

The HTTPs Config instance task was set as the next priority. This task needs to be called often enough for the webpage to detect any changes that occur within the project and properly transmit them. This can’t be higher priority than the Tx task but must not be lower than the Net timer task.

The Net Timer Task priority needs to be higher than the Rx task priority in order for the timing of each of these functions to stay in line with the rest of the project. It must be lower priority than the HTTPs task.

The Net Rx priorities must not be higher than the tasks monitoring the GUI display or touch sensor and needs to be lower than the Net timer task.

The CAN Tx and Rx task priorities are the lowest on the list since it is a basic monitoring of user input requests and software semaphore signals. As such, it must not be higher than the Task Start priority or GUI monitoring priorities.

Interrupts

Several interrupts are used throughout this application. A distinction can be made between two types of interrupts: the 'regular' ones and the 'Direct Drive LCD' ones. This section will provide more details about these two types of interrupts and the difference between them.

General Interrupts

The 'regular' interrupts are standard interrupts that µC/OS-III is aware of and that it will call at the correct time, depending of the interrupt handling method selected. These interrupts include the OS timer interrupts, USB interrupts, network interrupts and CAN Tx/Rx interrupts. The LCD dot clock interrupt, which causes the external memory manager task to execute, is also in this category of interrupts. µC/OS-III services can be used in these interrupts, if the usage allows it. For example, it is possible to do a semaphore post in these interrupts, but it is impossible to pend on a semaphore in an interrupt. These interrupts cannot happen during an OS critical section.

Direct Drive LCD Interrupts

These two interrupts, `R_DDLCD_DD_BLANK_isr()` and `R_DDLCD_DD_DATA_isr()` are used only by the Direct Drive LCD module for displaying the GUI on the LCD. These interrupts are required to control the signals that are sent to the external memory to display the GUI. These two interrupts are not kernel-aware and the kernel is not aware when they are executed. They can completely pre-empt the kernel, even in a critical section. It is impossible to use services from µC/OS-III from these interrupts. Instead, a flag is used to indicate whether or not the external memory can be accessed. This flag is checked by the external memory manager task to suspend or resume tasks accessing the external memory. User and/or applications won't need to modify these interrupts or the way they interact with µC/OS-III, µC/GUI or other tasks. Only registering a task accessing the external memory is required.

For more details about the Direct Drive mechanisms and the timing constraints involved, please see Renesas' Application Note Direct Drive LCD Demonstration, labelled R01AN0331EU0103.

Known Issues

Issue with RXC linker optimization.

RXC compiler/linker does not support (very) long file names.

Contacts

Micriμm, Inc.
1290 Weston Road, Suite 306
Weston, FL 33326
USA
+1 954 217 2036
+1 954 217 2037 (FAX)
e-mail : info@micrium.com
Website : www.micrium.com

Footprint

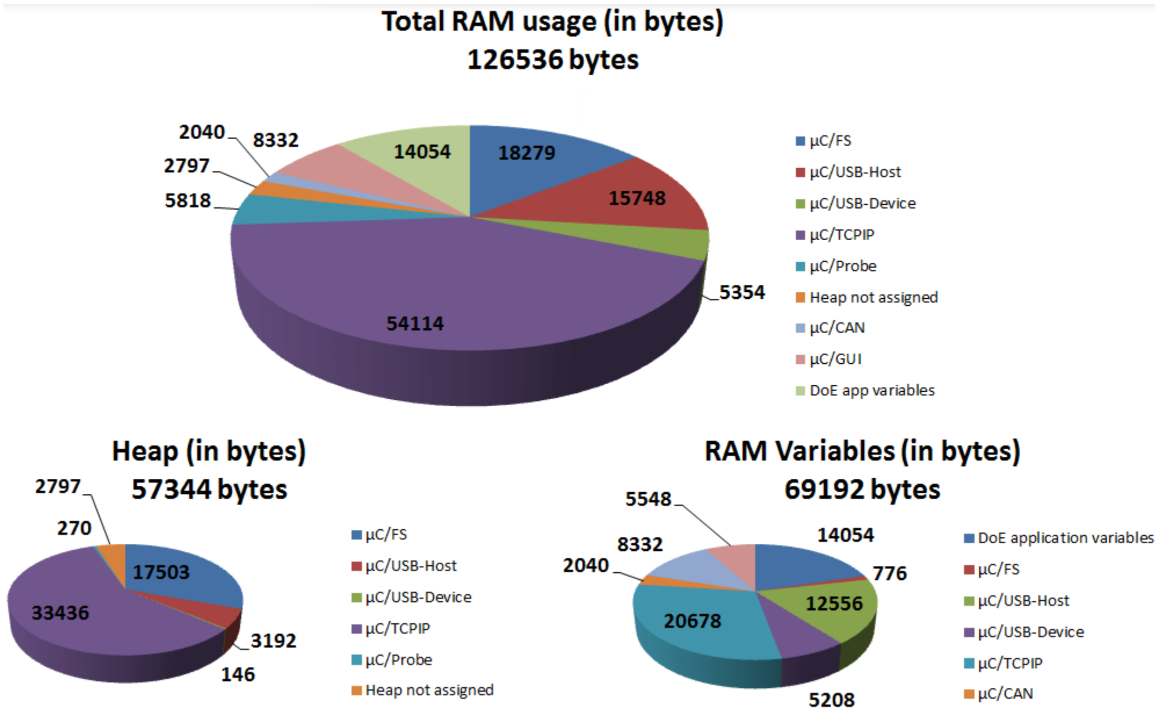


Figure 32 - Project RAM and ROM size usage with optimizations on MAX

The data shown here is obtained from the current settings of all Micrium modules in the Demo of Everything. The configurations have been minimized as much as possible to allow them to run within the constraints of the available memory so that each will operate within the demo. If the settings are modified, i.e. reduced or expanded further it could result in the project crashing or certain demos malfunctioning if there is not enough memory available.

	Flash (KB)	RAM (KB)
µC/FS and NOR driver	50.0	18.3
µC/USB-Host and MSC	18.9	12.8
µC/USB-Device and HID, CDC	32.2	5.9
µC/TCPIP, Webserver and DHCP client	105.8	55.0
µC/CAN	10.5	8.7
µC/Probe	5.2	2.8
µC/GUI	56.5	2.0
µC/OS-III	12.5	8.3
Application	175.7	13.1
GUI Graphic Resources	127.4	
Total	594.7	126.9

Table 17 - Data on ROM and RAM usage with optimizations on MAX

The Demo of Everything is an excellent example of multiple Micrium software modules integrated into an application; however some limitations were imposed due to the amount of available RAM on the YLCDRX63N target board. Some of the feature and functional limitations are as follows:

µC/USB-Device:

- Only HID and CDC classes are included in this project. Since buffer creation is minimal for both classes the small footprint size accurately reflects this.
- The number of devices, interfaces and maximum endpoints is limited.

Major configurations		
Name	Description	Value
`USBD_CFG_MAX_NBR_DEV`	Maximum number of USB devices	`1u`
`USBD_CFG_MAX_NBR_CFG`	Maximum number of configurations.	**`2u`**
`USBD_CFG_MAX_NBR_IF`	Maximum number of interfaces.	**`3u`**
`USBD_CFG_MAX_NBR_EP_DESC`	Max number of endpoint descriptors.	**`6u`**
`USBD_CFG_MAX_NBR_EP_OPEN`	Maximum number of opened endpoints.	**`6u`**
`USBD_CFG_MAX_NBR_STR`	Maximum number of string descriptors.	**`10u`**
`USBD_HID_CFG_MAX_NBR_DEV`	Maximum number of HID class instances.	`1u`
`USBD_CDC_CFG_MAX_NBR_DEV`	Max number of CDC class instances.	`1u`

Table 18 - Important settings for USB-Device overall Size

µC/USB-Host:

- Only the MSC class is included in this project. MSC class is the largest of the classes for both USB Host and USB Device and is accurately reflected in the analysis above.
- Similar limitations for the number of devices, interfaces and endpoints as indicated for the USB-Device stack.

Major configurations		
Name	Description	Value
`USBH_CFG_MAX_NBR_DEVS`	Maximum number of devices.	**`1u`**
`USBH_CFG_MAX_NBR_CFGS`	Maximum number of configurations per device.	`2u`
`USBH_CFG_MAX_NBR_IFS`	Maximum number of interfaces per configuration.	`2u`
`USBH_CFG_MAX_NBR_EPS`	Maximum number of endpoints per alternate interface.	`3u`
`USBH_MSC_CFG_MAX_DEV`	Maximum number of MSC devices.	**`1u`**

Table 19 - Important settings for USB-Host overall Size

µC/FS:

- The amount of memory consumed by the FS stack varies with user application. In this project the limitations are the depth of folder path (4 levels), maximum number of open devices at one time, and the maximum number of files and volumes that can be opened.

Major Configurations		
Name	Description	Value
`APP_CFG_FS_DEV_CNT`	Max number of open devices	2u
`APP_CFG_FS_VOL_CNT`	Max number of opened volumes	2u
`APP_CFG_FS_FILE_CNT`	Max number of open files	3u
`APP_CFG_FS_DIR_CNT`	Max number of opened directories	4u
`APP_CFG_FS_WORKING_DIR_CNT`	Max number active working directories	5u
`APP_CFG_FS_MAX_SEC_SIZE`	Max sector size supported	2048u

Table 20 - Important settings for File System overall Size

µC/TCPIP:

- For HTTPs and DHCPc, the biggest limitations are how many connections, sockets and buffers are allocated to the stack. The reliability of the TCP-IP connection is directly correlated to how well communications can be handled. With more connections and buffers available, there is less chance to miss packets and run out of sockets to connect with the HTTP webpage.
- For this demo, four files need to be downloaded by the http client (web browser) when the page is first loaded: one html file which is the main page, one gif file for the logo on the page, and two JavaScript files as libraries for the scripts inside the html file.
- Since browsers usually open multiple connections in parallel, the http server must be able to handle up to four simultaneous connections with the browser. This is why the configuration *"ConnNbrMax"* was set to four. This situation will occurs when the page is loaded the first time. Once the page is up and running one connection will be used during each refresh timeout to update the buttons states while the other connection is used to send POST requests to change the LED state.
- Since a network server application always needs one TCP connection and one socket to listen for incoming queries by clients, the *"NET_TCP_CFG_NBR_CONN"* and *"NET_SOCK_CFG_NBR_SOCK"* configurations have been set to five.
- The number of Transfer and Receive buffers has been configured to utilize the remaining RAM. The minimum number required depends on the refresh timeout use by the JavaScript code. With the actual timeout, it has been determine that five transmit buffers and five receipt buffers are required for the demo to run smoothly.

Major Configurations		
Name	Description	Value
NET_IF_CFG_MAX_NBR_IF	Sets max number of network interfaces.	2u
NET_IP_CFG_IF_MAX_NBR_ADDR	Set max number addresses per interface.	1u
NET_TCP_CFG_NBR_CONN	Configure total number TCP connections.	5
NET_SOCK_CFG_SEL_NBR_EVENTS_MAX	Set max number of socket selections	4 * 3
NET_SOCK_CFG_NBR_SOCK	Configure total number of sockets.	5
NET_CONN_CFG_NBR_CONN	Configure total number of connections.	10
NetDev_Cfg_RX_Ether Lg Rx Buffer	Desired size of Large Rx Buffers	1520u
NetDev_Cfg_RX_Ether # Lg Rx Buff	Desired number of Large Rx Buffers	7u
NetDev_Cfg_RX_Ether Lg Tx Buffer	Desired size of Large Tx buffers	1600u
NetDev_Cfg_RX_Ether # Lg Tx Buff	Desired number of Large Tx buffers	5u

Table 21 - Important settings for TCP/IP overall Size

Major Configurations		
Name	Description	Value
HTTPs_CFG_INSTANCE_CONN_NUM_MAX	Max number simultaneous connections	4
HTTPs_CFG_INSTANCE_CONN_BUFF_LEN	Instance connection buffer length	1000
HTTPs_CfgInstance_0 Config Token #	Configure total number of tokens	3

Table 22 - Important settings for HTTPs overall Size

µC/Probe:

- Due to RAM limitations on the target board, Probe's Kernel Awareness capabilities are not available in this demo. Attempting to launch Kernel Awareness within Probe, a warning message will be displayed indicating uC/Probe was unable to run Kernel Awareness.

Major Configurations		
Name	Description	Value
PROBE_COM_CFG_RX_MAX_SIZE	Config max receive packet size	256
PROBE_COM_CFG_TX_MAX_SIZE	Config max transmit packet size	256
PROBE_COM_CFG_STR_IN_BUF_SIZE	Configure total number TCP connections.	128
PROBE_COM_CFG_STR_OUT_BUF_SIZE	Set max number of socket selections	2048

Table 23 - Important settings for Probe overall Size